LEON: LIGHT WEIGHT EDGE DETECTION NETWORK

Nasrin Akbari and Amirali Baniasadi

Department of Computer Engineering, University of Victoria, Victoria, Canada

ABSTRACT

Deep Convolutional Neural Networks (CNNs) have achieved human-level performance in edge detection. However, there have not been enough studies on how to efficiently utilize the parameters of the neural network in edge detection applications. Therefore, the associated memory and energy costs remain high. In this paper, inspired by Depthwise Separable Convolutions and deformable convolutional networks (Deformable-ConvNet), we aim to address current inefficiencies in edge detection applications. To this end, we propose a new architecture, which we refer to as Lightweight Edge Detection Network (LEON). The proposed approach is designed to integrate the advantages of the deformable unit and DepthWise Separable convolutions architecture to create a lightweight backbone employed for efficient feature extraction. As we show, we achieve state-of-the-art accuracy while significantly reducing the complexity by carefully choosing proper components for edge detection purposes. Our results on BSDS500 and NYUDv2 demonstrate that LEON outperforms the current lightweight edge detectors while requiring only 500k parameters. It is worth mentioning that we train the network from scratch without using pre- trained weights.

KEYWORDS

Edge detection, lightweight neural network, Receptive field, network pruning

1. INTRODUCTION

Edge detection is the process of finding meaningful transitions in an image. This is done by detecting discontinuities in texture, colour, brightness, etc. Edges provide boundaries between different regions in the image. Detecting these boundaries is the first step in many computer vision tasks, such as edge-based face recognition, edge-based target recognition, scene understanding, image segmentation, fingerprint matching, license plate detection, object proposal, and object detection [1].

Edge detection is widely used in a variety of applications, including fingerprint recognition in mobile devices, well-localized maps of satellite images to suppress noise and produce realistic edge maps [2], self-driving vehicles to set the steering wheel angle based on the picture of the road [3], and finding pathological objects in medical images [4]. So, it's important to pay close attention to making a neural network that works well for the implementation.

The emergence of deep learning techniques has greatly promoted edge detection research over the past few years. Traditional approaches to the BSDS500 dataset often achieve a 0.59 ODS F-measure. DL-based methods, on the other hand, can achieve a 0.828 ODS [5]. Although recently proposed architectures achieve high accuracy, they are computationally inefficient. This makes developing lightweight networks that reduce the number of parameters while maintaining the detection accuracy critical. Figure 1 shows both the detection accuracy and complexity (model

David C. Wyld et al. (Eds): CCNET, AIMLA, CICS, IOTBS, NLTM, COIT - 2023
pp. 17-27, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130402

size) of several well-known deep learning-based methods. As shown in figure 1, the orange dot indicates how well our model matches human perception in terms of accuracy with a few parameters.

Many deep learning-based edge detectors use VGGNet (Visual Geometry Group) [6] as their feature-based extractor because of its excellent performance. However, VGGNet has a pretty extensive backbone and employs a large number of parameters, which makes it appropriate to fit more complex tasks such as image segmentation and object recognition. This work is motivated by the fact that edge detection is a low-level image-processing task and does not require complex networks for feature extraction.

To decrease the number of parameters and floating point operations (FLOPs), we take advantage of depthwise separable convolutions [7] which disentangle the spatial and channel interaction that is mixed in a regular convolution operation. However, it reduces the performance in comparison to conventional convolution. To compensate for the reduced performance, we increase the receptive field by carefully choosing proper lightweight components for edge de- tection purposes. We explain the details in section 3.

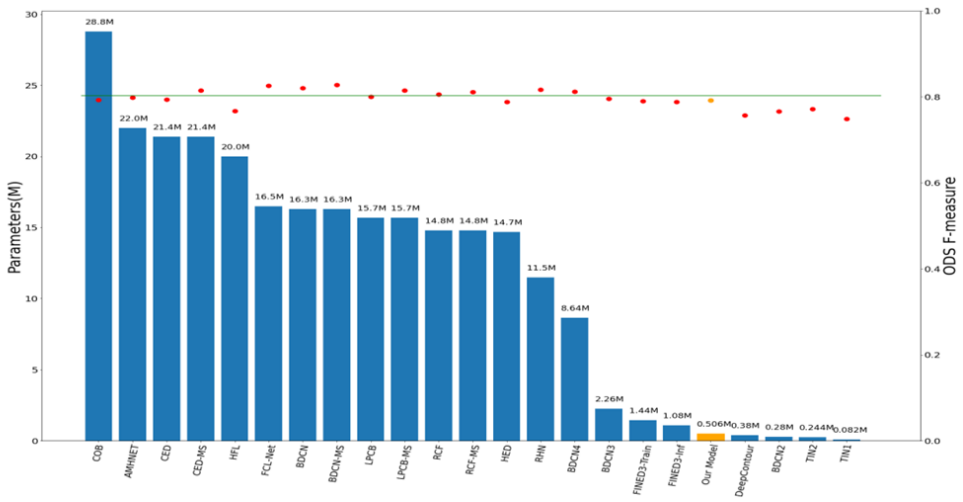

Figure 1. Comparison of complexity and accuracy performance among various edge detection schemes. Our proposed methods (orange).

The rest of this paper is organized as follows. Section 2 reviews related works and their issues. The proposed network architecture is described in section 3. Section 4 presents experimental results and compares them to the state-of-the-art edge detector networks using (Berkeley Segmentation Dataset 500) BSDS500 [8] and NYUDv2 [9] datasets. In section 5, we offer concluding remarks and discuss future research directions.

2. RELATED WORK

Over the past few years, a number of edge-detection solutions have been developed. Almost all edge detection approaches can be generally categorized into three groups, traditional edge detection, learning-based ones using handcrafted features, and deep learning networks. In the following paragraphs, we review some techniques that have been developed in recent years.

Intensity and colour gradients were the main focus of early pioneering edge detection methods. The Sobel [10] operator measures the 2-D spatial gradient of an image, emphasizing regions of

high spatial frequency that correspond to edges. The Canny algorithm [11] is a multistage edge detector. In this algorithm, the intensity of the gradients is computed by employing a filter based on the derivative of a Gaussian. The Gaussian filter reduces the impact of image noise. Subsequently, by removing non-maximum pixels of the gradient magnitude, possible edges are decreased to 1-pixel curves. Finally, applying the hysteresis threshold to the gradient magnitude, edge pixels are kept or eliminated. Zero-crossing theory based algorithms are proposed by [12, 13]. Traditional approaches suffer from some limitations, including merely focusing on the changes of local intensity while failing to recognize and remove the non-edge texture.

The introduction of learning-based edge detectors made it possible to partially overcome challenges such as texture detection problems in traditional approaches. In this group of detectors, hand-craft features are initially extracted. Later, classifiers trained using these features are applied to identify edges. The first data-driven approaches were proposed by Konishi et al. [14] who used images to learn the probability distributions of responses that correspond to the two sets of edge filters. In another work [15], random decision forests were applied to show the structure presented in local image patches. The structured forest uses colour and gradient features to high-quality output edges.

The aforementioned techniques are developed according to handcrafted features, which mostly fail to provide high-level information for semantically meaningful edge detection and have a limited capability of capturing edges at different scales. To address these issues, a number of CNN-based algorithms with strong learning capabilities have been proposed in recent years. One of the most influential in DNN-based edge detection is HED[16]. This study uses fully convolutional neural networks and deeply supervised nets to find the edge probability for every pixel. HED uses VGGNet [6] for the feature extraction and fuses all the side outputs of VGGNet features to minimize the weighted cross-entropy loss function. Since then, various extensions based on HED and VGGNet have been developed, including CED [17], AMHNet [18], RCF [19], LPCB [20], and BDCN [21].

While CNN is a very successful model, it often requires high computational power and resources. Hence, the current trend is to design efficient CNN structures that overcome such issues. Fined [22], dense extreme inception network [23], and TIN[5] have proposed a lightweight architecture for edge detection. Although these networks are light and fast, they have low detection accuracy. To achieve a better trade-off between accuracy and efficiency for edge detection, we need to optimize the architecture and initial parameters of deep learning models so that they consume fewer resources while maintaining accuracy. In this paper, we build our model by simplifying the backbone for feature extraction and carefully choosing the proper components. Therefore, we achieve good edge quality with a much simpler model compared to other studies.

3. LIGHTWEIGHT EDGE DETECTION NETWORK

Inefficiency of the models outlined in the previous section at once. In Figure 2 we present LEON's architecture. We trained the network from scratch. Below, we review the components used by LEON.

3.1. Efficient Backbone

Most deep learning-based edge detectors [17–21] employ VGGNet as their feature extraction backbone. However, we believe that edge detection is a simple task and does not need to have an extensive backbone. We reduce the backbone's complexity while keeping its efficiency by using lightweight components. To resemble the pyramid structure, we stack up three stages and use a

max-pooling operation for down sampling the features between the stages. The dimension of the output feature maps decreases as we proceed. As we move forward in the stages, the patterns get more complex; hence, there are larger combinations of patterns to be captured. Therefore, we increase the feature channel number (the number of filters) in subsequent stages to capture as many combinations as possible. Stages 1, 2, and 3 have channel numbers 16, 64, and 256, respectively. The backbone is made of mainly a combination of deformable and customized depthwise separable convolutions. To create the fused output, we use standard bilinear interpolation to up sample the low-resolution features. Then, we concatenate all the stage outputs together to form the fused output. We next elaborate on the layers and components used by LEON in detail.

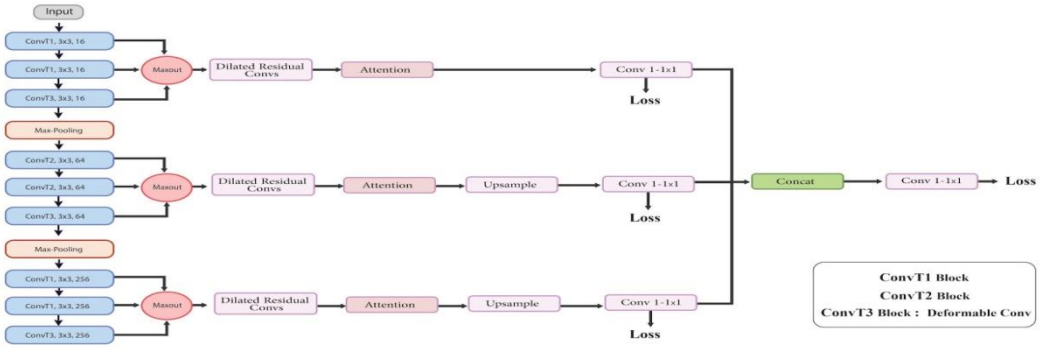

Figure 2. LEON architecture

Figure 3. Convt1 block - Convt2 block

3.1.1. Deformable convolution

Geometric transformation and variations widely existing in natural images make feature extraction tasks challenging. Standard convolution kernels have a fixed structure and have limitations in capturing geometric transformations. Deformable convolutions can address this issue efficiently. This type of convolution has the ability to change its kernel shape and the parameters within it to adapt to the image content. This adds 2D offset kernels to the regular sampling location in the standard convolution, which enables the network to have different receptive fields according to the scale of the objects. These 2D offset kernels are learnable from the preceding feature maps using additional convolutional layers and can be trained end-to-end using normal back propagation functions. We simply add this module at the end of each stage to keep our network light in terms of parameters and computation. We can strengthen our features this way before transferring them to the next stage [24].

3.1.2. Depthwise Separable Convolution

Conventional convolution performs the channels and spatial-wise computation in one step, while Depthwise Separable Convolution reduces the number of parameters by splitting the computation into two steps: 1) depthwise convolution, which applies a single convolutional filter per input channel, and 2) pointwise convolution, which creates a linear combination of the output of the depthwise convolution [7]. This approach, however, degrades accuracy. To address this problem, we reinforce the features by using additional side blocks while keeping the number of parameters as low as possible. We use RELU activation after each pointwise convolution to add non-linearity to the model for making complex decisions (Figure 3 - Convt1). To increase the accuracy of the model while keeping the number of parameters low, we modified Convt1 to Convt2 by adding pointwise convolution, which uses only a 1×1 kernel to iterate through every single point between two RELU activations. In addition, to overcome the overfitting problem, after each RELU activation, we employ a batch normalization technique as a regularizer.

3.2. Efficient Side Structure

3.2.1. Maxout Layer

At each stage, before transferring the inputs to the side output layers (from left to right), we do a Maxout operation instead of the standard concatenation block. Maxout activation can reduce the number of parameters significantly in comparison to the classical dense blocks. Instead of stacking the output of previous layers at each stage on top of each other, we only keep the maximum value at each position by inducing competition between feature maps and accelerating network convergence.

3.2.2. Dilated Residual Convolution Module

To enhance the extracted features by depth-wise separable convolution in the backbone, we connect every feature extraction layer to the dilated convolution module adopted in [5]. We use different dilation sizes to capture different levels of receptive fields in the image. The first dilation is 4, followed by 8, 12, and 16, and all the layers have 32 filters. After pixel-wise aggregation, we use hierarchical residual-like connections to improve the multi-scale representation ability at a more granular level. This block can be plugged into the state-of-the-art backbone without any effort. Figure 4 shows the design of the DDR module.

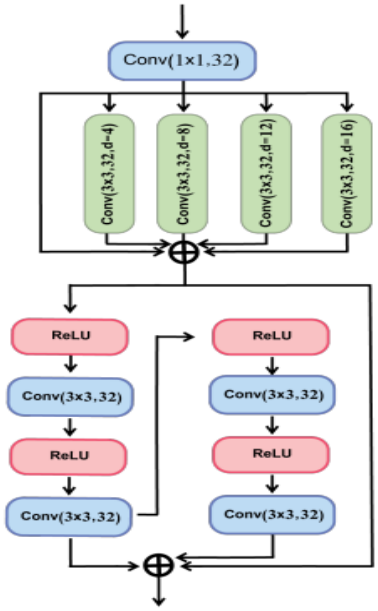

Figure 4. Visual Representation of dilated residual convolution module

3.2.3. Convolutional Block Attention Module (Cbam)

We use a lightweight spatial and channel attention module after the dilated residual convolution block to focus on the relevant features while diminishing the other parts [26]. The spatial attention extracts the inter-spatial relationships of features to find "where"' is an informative part of the image. To calculate this, we first apply average pooling and max pooling, which summarize the average presence of features and the most activated presence of a feature, respectively. Then, we use a convolution layer in addition to the concatenated feature descriptor to create a spatial attention map that specifies where to highlight or suppress features. [26].
The channel attention block redistributes the channel's feature responses to give higher importance to specific channels over others. In order to compute the channel attention, we squeeze the spatial dimension of the input feature map. [26].

3.3. Loss Function

In an image, the edge and non-edge pixel data are not equally distributed. CNN models can achieve pretty high accuracy just by predicting the majority class, but they fail to capture the minority class. Unfortunately, this accuracy is misleading. To address this problem, we adopt the weighted cross-entropy loss function proposed in [19].

To train the network, we match all the stages and fused outputs to the ground truth. The following equation compares each pixel of each image to its label as.

$$L(x_i;W)=\begin{cases}\alpha.log(1-P(x_i;W)) & if y_i=0\\ 0 & if \le y_i \le \eta\\ \beta.logP(x_i;W) & otherwise,\end{cases} \quad (1)$$

$$\alpha=\lambda.\frac{|Y^+|}{|Y^+|+|Y^-|} \qquad \beta=\frac{|Y^-|}{|Y^+|+|Y^-|} \quad (2)$$

X, P(X), Y, W, and η, respectively, denote features extracted from the CNN network, the output of the standard sigmoid function, the ground truth edge probability, all the parameters that will be learned in the CNN network, and the percentage of non-edge and edge pixels. The hyper-parameter is used to balance the number of positive and negative samples. Because each image is being labelled by multiple annotators, and humans vary in cognition, the predefined threshold is used to distinguish between edge and non-edge pixels in the edge probability map. If a pixel is marked by fewer than η of the annotators, then it is considered a non-edge pixel. To generalize the loss function to all the pixels inside the image (I), at each stage (k) and fuse layer, the following loss function is used:

$$L(W) = \sum_{i=1}^{|I|} \left(\sum_{k=1}^{|K|} L\left(x_i^k; W\right) + L\left(x_i^{fuse}; W\right) \right) \qquad (3)$$

4. EXPERIMENTS AND DISCUSSIONS

4.1.1. Implementation Details

We use PyTorch for implementation and initialize the stages of our backbone networks with Gaussian distribution with zero-mean and standard deviation of 0.01. The learning rate starts from 0.01 and then is updated using a linear scaling factor, multiplying 0.1 for every two epochs. The optimizer is stochastic gradient descent, and the training process terminates at eight epochs. We conduct all the experiments on a single GPU, NVIDIA GeForce 2080Ti, with 11G memory.

4.1.2. Dataset

In order to have a fair comparison to other published works in tables 1 and 2, we evaluate our proposed network on the same Berkeley Segmentation (BSDS500) [8] and NYUDv2 [9] Dataset . BSDS500 consists of 200 training, 100 validation, and 200 test images. We combine the 200 training images with 100 validation images to create a training set. We adopt the data augmentation technique similar to RCF [19]. In addition, similar to RCF, we also added the PASCAL VOC [27] dataset and its flipped images into our training set.
The NYUD dataset is composed of 1449 densely labelled pairs of aligned RGB and depth images (HHA). This dataset consists of video sequences from various indoor scenes captured by the Microsoft Kinect's RGB and Depth cameras. It is divided into 381 training, 414 validation, and 654 testing images. Similar to RCF [19], we rotate the images and corresponding annotations to 4 different angles (0, 90, 180, and 270 degrees) and flip them at each angle.

4.1.3. Performance Metrics

Note that the share of edge pixels in each image is around 10%, whereas the share of non-edge pixels is 90%. Therefore, even when a model fails to predict any edges, its accuracy is still 90%. As such, accuracy is a poor measure for evaluating imbalanced problems such as edge detection. Therefore, we use F-Score for the evaluation of our model. The F-score combines the precision and recall of the model, where it reaches its best value at one and its worst score at 0.

Recall = TruePositives / (TruePositives + FalseNega- tives)

Precision= TruePositives / (TruePositives + FalsePosi- tives)

F-Measure = (2 * Precision * Recall) / (Precision + Recall) and

We need a threshold to binarize the output of the CNN network to make it comparable to the ground truth, which is also binarized. There are two ways to compute the optimal threshold corresponding to the F-score.

- Optimal Dataset Scale: Iterates over all possible thresholds and sets one threshold for the entire dataset. The threshold that gives the best F-score for the dataset is used to calculate ODS score.
- Optimal Image Scale: Finds the best threshold and corresponding F-score for each image. The OIS F-score is calculated by averaging all of the F-scores for all images.

4.1.4. Comparison with State-of-the-Arts

On the BSDS500 dataset: We compare our methods in terms of F-score and number of parameters to prior edge detection approaches, including both traditional ones and recently proposed CNN-based models. According to Table 1 and Figure 5, we notice that our baseline model, while using a significantly lower number of parameters, can even achieve outstanding results (ODS of 0.792 and OIS of 0.805) which are equal or better than most recent lightweight CNN models such as BDCN2, TIN1,TIN2, FINED3-Inf and FINED3-Train [22].

Table 1. Comparison to other methods on BSDS500 dataset.

Method	ODS	OIS	#P (million)
Canny	0.611	0.676	-
OEF	0.746	0.77	-
gPb-UCM	0.72	0.755	-
SE	0.743	0.763	-
AMHNET	0.798	0.829	22
BDP-Net	0.808	0.828	18.7
FCL-Net	0.826	845	16.5 M
BAN	0.81	0.827	15.6
LPCB	0.815	0.834	15.7
BMRN	0.828	0.81	+14.8
RCF	0.806	0.823	14.8
HED	0.788	0.808	14.7
COB	0.793	0.82	28.8
RHN	0.817	0.833	11.5
CED	0.815	0.834	21.4
DeepEdge	0.753	0.772	-
DeepContour	0.757	0.776	0.38
BDCN	0.82	0.838	16.3
BDCN2	0.766	0.787	0.48
BDCN3	0.796	0.817	2.26
BDCN4	0.812	0.83	8.69
TIN1	0.749	0.772	0.08
TIN2	0.772	0.792	0.24
FINED3-Inf	0.788	0.804	1.08
FINED3-Train	0.79	0.808	1.43
Our model	0.792	0.805	0.506

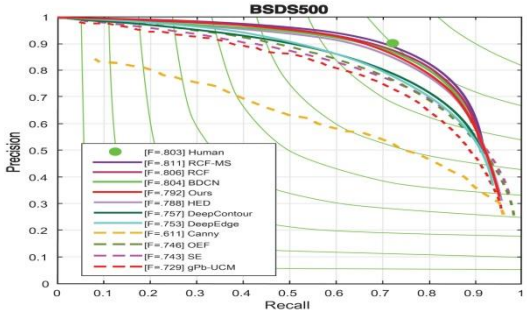

Figure 5. Precision-Recall curves of our models and some competitors on BSDS500 dataset

.

On the NYUD dataset: The comparison results on the NYUD dataset are illustrated in Table 2, and the precision- recall curves are depicted in Figure 6. For testing the model on NYUD, we use network settings similar to that used for BSDS500. Some studies use two separate models to train RGB images and HHA feature images of NYUD and report the evaluation metrics on the average for the outputs of the models. Our network is only tested on RGB images, so in order to evaluate results fairly, we contrasted our model's output with those of models that were only tested on RGB.

Table 2. Comparison with other methods on NYUD dataset.

Method	ODS	OIS	#P (million)
OEF	0.651	0.667	_
gPb-UCM	0.632	0.661	_
SE	0.695	0.708	_
SE+NG+	0.706	0.734	_
AMHNET	0.744	0.758	22
BDCN	0.748	0.763	16.3
LPCB	0.739	0.754	15.7
RCF	0.743	0.757	14.8
BMRN	0.759	0.776	+14.8
HED	0.72	0.734	14.7
Our Model	0.725	0.738	0.5

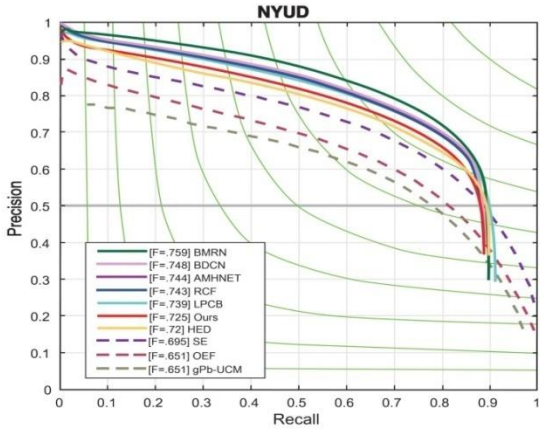

Figure 6. Precision-Recall curves of our models and some competitors on the NYUD dataset.

5. CONCLUSIONS

Edge detection has numerous practical applications in the real world; hence, we must design an efficient architecture for its implementation. Most existing deep neural networks for edge detection tasks use transfer learning from pre-trained models such as VGG16, which have a large number of parameters and are trained for high-level tasks. However, edge detection has a simple set of features and does not require a large number of convolutional layers for feature extraction. Therefore, in this research, we introduced a new architecture that is both lightweight and has state-of-the-art performance. Our network makes full use of customized depthwise separable and deformable convolutions to carry out edge detection. Besides, we use lightweight components to increase the receptive field of our model to produce high-quality edges. Our network architecture is extendable and can potentially be employed for use in other vision tasks such as salient object detection and semantic segmentation.

REFERENCES

[1] Victor Wiley and Thomas Lucas. "Computer vision and image processing: a paper review. International Journal of Artificial Intelligence Research", 2(1):29–36, 2018.

[2] Ronald J Holyer and Sarah H Peckinpaugh. Edge detection applied to satellite imagery of the oceans. IEEE transactions on geoscience and remote sensing, 27(1):46–56, 1989.

[3] Abhishek Gupta, Alagan Anpalagan, Ling Guan, and Ahmed Shaharyar Khwaja. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. Array, 10:100057, 2021.

[4] Wei-Chun Lin and Jing-Wein Wang. Edge detection in medical images with quasi high-pass filter based on local statistics. Biomedical Signal Processing and Control, 39:294–302, 2018.

[5] Jan Kristanto Wibisono and Hsueh-Ming Hang. Traditional method inspired deep neural network for edge detection. In 2020 IEEE International Conference on Image Processing (ICIP), pages 678–682. IEEE, 2020.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[7] Yunhui Guo, Yandong Li, Liqiang Wang, and Tajana Rosing. Depthwise convolution is all you need for learning multiple visual domains. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 8368–8375, 2019.

[8] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. IEEE transactions on pattern analysis and machine intelligence, 33(5):898–916, 2010.

[9] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In European conference on computer vision, pages 746–760. Springer, 2012.

[10] O Rebecca Vincent, Olusegun Folorunso, et al. A descriptive algorithm for sobel image edge detection. In Proceedings of informing science & IT education conference (InSITE), volume 40, pages 97–107, 2009.

[11] Renjie Song, Ziqi Zhang, and Haiyang Liu. Edge connection based canny edge detection algorithm. Pattern Recognition and Image Analysis, 27(4):740–747, 2017.

[12] Rajiv Mehrotra and Shiming Zhan. A computational approach to zero-crossing-based two-dimensional edge detection. Graphical Models and Image Processing, 58(1):1–17, 1996.

[13] James J. Clark. Authenticating edges produced by zero-crossing algorithms. IEEE Trans- actions on Pattern Analysis and Machine Intelligence, 11(1):43–57, 1989.

[14] Scott Konishi, Alan L. Yuille, James M. Coughlan, and Song Chun Zhu. Statistical edge detection: Learning and evaluating edge cues. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(1):57–74, 2003.

[15] Piotr Dollár and C Lawrence Zitnick. Fast edge detection using structured forests. IEEE transactions on pattern analysis and machine intelligence, 37(8):1558–1570, 2014.

[16] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In Proceedings of the IEEE international conference on computer vision, pages 1395–1403, 2015.

[17] Yupei Wang, Xin Zhao, and Kaiqi Huang. Deep crisp boundaries. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3892–3900, 2017.
[18] Dan Xu, Wanli Ouyang, Xavier Alameda-Pineda, Elisa Ricci, Xiaogang Wang, and Nicu Sebe. Learning deep structured multi-scale features using attention-gated crfs for contour prediction. Advances in neural information processing systems, 30, 2017.
[19] Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Kai Wang, and Xiang Bai. Richer convolutional features for edge detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3000–3009, 2017.
[20] Ruoxi Deng, Chunhua Shen, Shengjun Liu, Huibing Wang, and Xinru Liu. Learning to predict crisp boundaries. In Proceedings of the European Conference on Computer Vision (ECCV), pages 562–578, 2018.
[21] Jianzhong He, Shiliang Zhang, Ming Yang, Yanhu Shan, and Tiejun Huang. Bi-directional cascade network for perceptual edge detection. In Proceedings of the IEEE/CVF Confer- ence on Computer Vision and Pattern Recognition, pages 3828–3837, 2019.
[22] Jan Kristanto Wibisono and Hsueh-Ming Hang. Fined: Fast inference network for edge detection. arXiv preprint arXiv:2012.08392, 2020.
[23] Xavier Soria Poma, Edgar Riba, and Angel Sappa. Dense extreme inception network: Towards a robust cnn model for edge detection. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 1923–1932, 2020.
[24] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In Proceedings of the IEEE international conference on computer vision, pages 764–773, 2017.
[25] Leonie Henschel, Sailesh Conjeti, Santiago Estrada, Kersten Diers, Bruce Fischl, and Mar- tin Reuter. Fastsurfer-a fast and accurate deep learning based neuroimaging pipeline. Neu- roImage, 219:117012, 2020.
[26] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In Proceedings of the European conference on computer vision (ECCV), pages 3–19, 2018.
[27] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fi- dler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 891–898, 2014.