

Pixel Difference Networks for Efficient Edge Detection

Zhuo Su^{1,*} Wenzhe Liu^{2,*} Zitong Yu¹ Dewen Hu² Qing Liao³ Qi Tian⁴
Matti Pietikäinen¹ Li Liu^{2,1,†}

¹Center for Machine Vision and Signal Analysis, University of Oulu, Finland

²National University of Defense Technology, China

³Harbin Institute of Technology (Shenzhen), China ⁴Xidian University, China

{zhuo.su, zitong.yu, matti.pietikainen, li.liu}@oulu.fi

{liuwenzhe15, dwhu}@nudt.edu.cn, liaoqing@hit.edu.cn, wywqtian@gmail.com

Abstract

Recently, deep Convolutional Neural Networks (CNNs) can achieve human-level performance in edge detection with the rich and abstract edge representation capacities. However, the high performance of CNN based edge detection is achieved with a large pretrained CNN backbone, which is memory and energy consuming. In addition, it is surprising that the previous wisdom from the traditional edge detectors, such as Canny, Sobel, and LBP are rarely investigated in the rapid-developing deep learning era. To address these issues, we propose a simple, lightweight yet effective architecture named Pixel Difference Network (PiDiNet) for efficient edge detection. PiDiNet adopts novel pixel difference convolutions that integrate the traditional edge detection operators into the popular convolutional operations in modern CNNs for enhanced performance on the task, which enjoys the best of both worlds. Extensive experiments on BSDS500, NYUD, and Multicue are provided to demonstrate its effectiveness, and its high training and inference efficiency. Surprisingly, when training from scratch with only the BSDS500 and VOC datasets, PiDiNet can surpass the recorded result of human perception (0.807 vs. 0.803 in ODS F-measure) on the BSDS500 dataset with 100 FPS and less than 1M parameters. A faster version of PiDiNet with less than 0.1M parameters can still achieve comparable performance among state of the arts with 200 FPS. Results on the NYUD and Multicue datasets show similar observations. The codes are available at <https://github.com/zhuoinoulu/pidinet>.

1. Introduction

Edge detection has been a longstanding, fundamental low-level problem in computer vision [5]. Edges and object

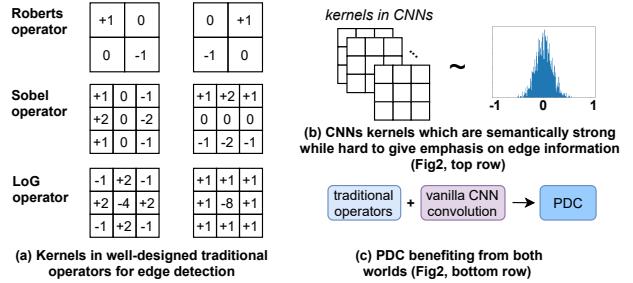


Figure 1. PDC benefits from both worlds with proper integration of traditional operators and modern CNNs.

boundaries play an important role in various higher-level computer vision tasks such as object recognition and detection [29, 11], object proposal generation [6, 54], image editing [10], and image segmentation [41, 4]. Therefore, recently, the edge detection problem has also been revisited and injected new vitality due to the renaissance of deep learning [2, 23, 47, 60, 55, 31].

The main goal of edge detection is identifying sharp image brightness changes such as discontinuities in intensity, color, or texture [53]. Traditionally, edge detectors based on image gradients or derivatives information are popular choices. Early classical methods use the first or second order derivatives (e.g., Sobel [50], Prewitt [46], Laplacian of Gaussian (LoG), Canny [5], etc.) for basic edge detection. Later learning based methods [16, 9] further utilize various gradient information [59, 37, 12, 15] to produce more accurate boundaries.

Due to the capability of automatically learning rich representations of data with hierarchical levels of abstraction, deep CNNs have brought tremendous progress for various computer vision tasks including edge detection and are still rapidly developing. Early deep learning based edge detection models construct CNN architectures as classifiers to predict the edge probability of an input image

*Equal contributions. † Corresponding author: <http://lilyliu.com>

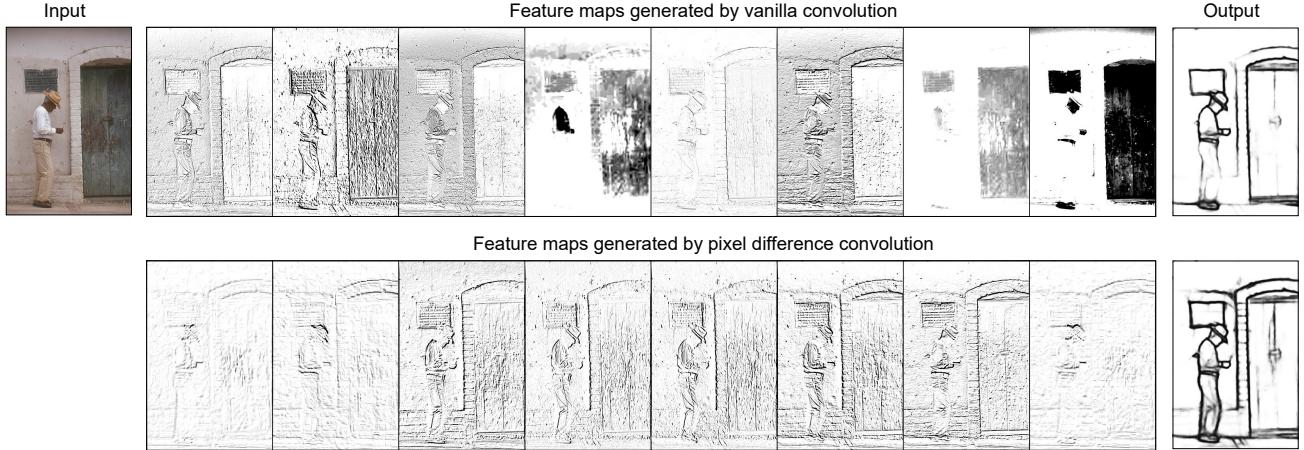


Figure 2. PiDiNet configured with pixel difference convolution (PDC) vs. the baseline with vanilla convolution. Both models were trained only using the BSDS500 dataset. Compared with vanilla convolution, PDC can better capture gradient information from the image that facilitates edge detection.

Table 1. Comparison between ours and some leading edge detection models in terms of efficiency and accuracy. The multiply-accumulates (MACs) are calculated based on a 200×200 image, FPS and ODS *F-measure* are evaluated on the BSDS500 test set.

	HED [60]	RCF [31]	BDCN [18]	PiDiNet	PiDiNet(tiny)
Params	14.7M	14.8M	16.3M	710K	73K
MACs	22.2G	16.2G	23.2G	3.43G	270M
Throughput	78FPS	67FPS	47FPS	92FPS	215FPS
Pre-training	ImageNet	ImageNet	ImageNet	No	No
ODS <i>F-measure</i>	0.788	0.806	0.820	0.807	0.787

patch [2, 47, 3]. Building on top of fully convolutional networks [33], HED [60] performs end-to-end edge detection by leveraging multilevel image features with rich hierarchical information guided by deep supervision, and achieves state-of-the-art performance. Other similar works include [62, 23, 36, 55, 61, 31, 8, 18].

However, integration of traditional edge detectors with modern CNNs were rarely investigated. The former were merely utilized as auxiliary tools to extract candidate edge points in some prior approaches [3, 2]. Intuitively, edges manifest diverse specific patterns like straight lines, corners, and “X” junctions. On one hand, traditional edge operators like those shown in Fig. 1 are inspired by these intuitions, and based on gradient computing which encodes important gradient information for edge detection by explicitly calculating pixel differences. However, these handcrafted edge operators or learning based edge detection algorithms are usually not powerful enough due to their shallow structures. On the other hand, modern CNNs can learn rich and hierarchical image representations, where vanilla CNN kernels serve as probing local image patterns. Nevertheless, CNN kernels are optimized by starting from random initialization which has no explicit encoding for gradient information, making them hard to focus on edge related features.

We believe a new type of convolutional operation can be derived, to satisfy the following needs. Firstly, it can easily capture the image gradient information that facilitates edge detection, and the CNN model can be more focused with the release of burden on dealing with much unrelated image features. Secondly, the powerful learning ability of deep CNNs can still be preserved, to extract semantically meaningful representations, which lead to robust and accurate edge detection. In this paper, we propose pixel difference convolution (PDC), where the pixel differences in the image are firstly computed, and then convolved with the kernel weights to generate output features (see Fig. 3). We show PDC can effectively improve the quality of the output edge maps, as illustrated in Fig. 2.

On the other hand, leading CNN based edge detectors suffer from the deficiencies as shown in Table 1: being memory consuming with big model size, being energy hungry with high computational cost, running inefficiency with low throughput and label inefficiency with the need of model pre-training on large scale dataset. This is due to the fact that the annotated data available for training edge detection models is limited, and thus a well pretrained (usually large) backbone is needed. For example, the widely adopted routine is to use the large VGG16 [49] architecture that was trained on the large scale ImageNet dataset [7].

It is important to develop a lightweight structure, to achieve a better trade-off between accuracy and efficiency for edge detection. With pixel difference convolution, inspired by [19, 20], we build a new end-to-end architecture, namely Pixel Difference Network (PiDiNet) to solve the mentioned issues in one time. Specifically, PiDiNet consists of an efficient backbone and an efficient task-specific side structure (see Fig. 5), able to do robust and accurate edge detection with high efficiency.

2. Related Work

Using Traditional Edge Detectors to Help Deep CNN Models for Edge Detection. Canny [5] and SE [9] edge detectors are usually used to extract candidate contour points before applying the CNN model for contour/non-contour prediction [2, 3]. The candidate points can be also used as auxiliary relaxed labels for better training the CNN model [32]. Instead of relying on the edge information from the hand-crafted detectors, PDC directly integrates the gradient information extraction process into the convolutional operation, which is more compact and learnable.

Lightweight Architectures for Edge Detection. Recently, efforts have been made to design lightweight architectures for efficient edge detection [56, 57, 45]. Some of them may not need a pretrained network based on large scale dataset [45]. Although being compact and fast, the detection accuracies with these networks are unsatisfactory. Alternatively, lightweight architectures for other dense prediction tasks [13, 58, 43, 25, 38, 63] and multi-task learning [24, 26] may also benefit edge detection. However, the introduced sophisticated multi-branch based structures may lead to running inefficiency. Instead, we build a backbone structure which only uses a simple shortcut [19] as the second branch for the convolutional blocks.

Integrating Traditional Operators. The proposed PDC is mostly related to the recent central difference convolution (CDC) [66, 65, 64, 67] and local binary convolution (LBC) [21], of which both derive from local binary patterns (LBP) [42] and involve calculating pixel differences during convolution. LBC uses a set of predefined sparse binary filters to generalize the traditional LBP, focusing on reducing the network complexity. CDC further proposes to use learnable weights to capture image gradient information for robust face anti-spoofing. CDC can be seen as one instantiated case of the proposed PDC (*i.e.*, Central PDC), where the central direction is considered, as we will introduce in Section 3. Like CDC, PDC uses learnable filters while being more general and flexible to capture rich gradient information for edge detection. On the other hand, Gabor convolution [34] encodes the orientation and scale information in the convolution kernels by multiplying the kernels with a group of Gabor filters, while PDC is more compact without any auxiliary traditional feature filters.

3. Pixel Difference Convolution

The process of pixel difference convolution (PDC) is pretty similar to that of vanilla convolution, where the original pixels in the local feature map patch covered by the convolution kernels are replaced by pixel differences, when conducting the convolutional operation. The formulations

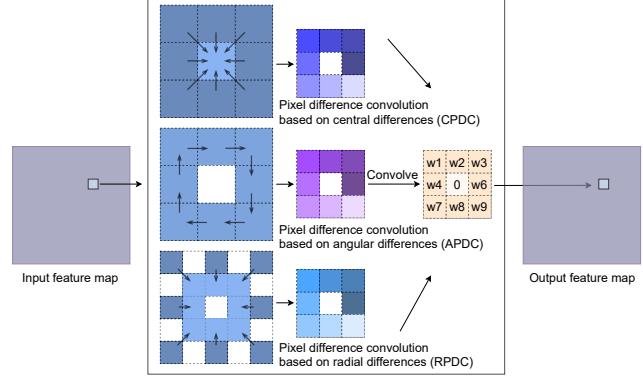


Figure 3. Three instances of pixel difference convolution derived from extended LBP descriptors [28, 30, 52]. One can derive other instances by designing the picking strategy of the pixel pairs.

of vanilla convolution and PDC can be written as:

$$y = f(\mathbf{x}, \boldsymbol{\theta}) = \sum_{i=1}^{k \times k} w_i \cdot x_i, \quad (\text{vanilla convolution}) \quad (1)$$

$$y = f(\nabla \mathbf{x}, \boldsymbol{\theta}) = \sum_{(x_i, x'_i) \in \mathcal{P}} w_i \cdot (x_i - x'_i), \quad (\text{PDC}) \quad (2)$$

where, x_i and x'_i are the input pixels, w_i is the weight in the $k \times k$ convolution kernel. $\mathcal{P} = \{(x_1, x'_1), (x_2, x'_2), \dots, (x_m, x'_m)\}$ is the set of pixel pairs picked from the current local patch, and $m \leq k \times k$.

To capture rich gradient information, the pixel pairs can be selected according to different strategies, which can be inspired from the numerous traditional feature descriptors. Here, we utilize the ideas from the work in [42, 30, 52], where the local binary pattern (LBP) and its robust variants, extended LBP (ELBP), were used to encode pixel relations from varying directions (angular and radial). Specifically, ELBP are obtained by firstly calculating the pixel differences within a local patch (from m pixel pairs), resulting in a pixel difference vector, and then binarizing the vector to create an m -length 0/1 code. Then, the bag-of-words technique [27] is usually used to calculate the code distribution (or histogram), which is regarded as the image representation. In ELBP, the angular and radial directions were demonstrated to help encode potential discriminative image cues and be complementary for increasing the feature representational capacity for various computer vision tasks, such as texture classification [30, 28] and face recognition [52].

By integrating ELBP with CNN convolution, we derive three types of PDC instances as shown in Fig. 3, in which we name them as central PDC (CPDC), angular PDC (APDC) and radial PDC (RPDC) respectively. The pixel pairs in the local patch is easy to understand. For example, for the APDC with kernel size 3×3 , we create 8 pairs in the angular direction in the 3×3 local patch (thus $m = 8$), then the pixel differences obtained from the pairs are con-

volved with the kernel by doing an element-wise multiplication with the kernel weights, followed by a summation, to generate the value in the output feature map.

The derived PDC instances based on ELBP can be seen as an extension of ELBP that are more flexible and learnable. Although being powerful, the original ELBP codes are discrete with limited representative ability. While the useful encodings of pixel relations in PDC will be preserved in the trained convolution kernels, as during the training process of CNN, the convolution kernels will be encouraged to have higher inner product with those important encodings, in order to create higher activation responses¹. By training from abundant of data, PDC is able to automatically learn rich representative encodings for the task.

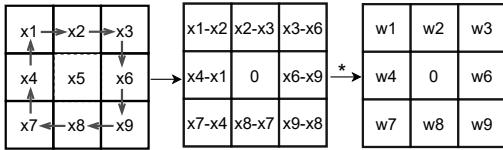


Figure 4. Selection of pixel pairs and convolution in APDC.

Converting PDC to Vanilla Convolution. According to Eq. 6, one may notice that the computational cost and memory footprint by PDC are doubled compared with the vanilla counterpart. However, once the convolution kernels have been learnt, PDC layers can be converted to vanilla convolutional layers by instead saving the differences of the kernel weights in the model, according to the locations of the selected pixel pairs. In this way, the efficiency is maintained during inference. Taking APDC as an example (Fig. 11), conversion is done with the following equations:

$$\begin{aligned} y &= w_1 \cdot (x_1 - x_2) + w_2 \cdot (x_2 - x_3) + w_3 \cdot (x_3 - x_6) + \dots \\ &= (w_1 - w_4) \cdot x_1 + (w_2 - w_1) \cdot x_2 + (w_3 - w_2) \cdot x_3 + \dots \\ &= \hat{w}_1 \cdot x_1 + \hat{w}_2 \cdot x_2 + \hat{w}_3 \cdot x_3 + \dots = \sum \hat{w}_i \cdot x_i. \end{aligned} \quad (3)$$

It is worth mentioning that we can also use this tweak to speed up the training process, where the differences of kernel weights are firstly calculated, followed by the convolution with the untouched input feature maps. We have illustrated more details in the appendix.

4. PiDiNet Architecture

As tried by some prior works [56, 45, 57], we believe it is both necessary and feasible to solve the inefficiency issues mentioned in Section 1 in one time by building an architecture with small model size and high running efficiency, and can be trained from scratch using limited datasets for effective edge detection. We construct our architecture with the following parts (Fig. 5).

¹Usually, higher activation responses are considered to be more salient, as adopted in many network pruning methods [17, 51]

Efficient Backbone. The building principle for the backbone is to make the structure slim while own high running efficiency. Thus we do not consider the sophisticated multi-branch lightweight structures proposed for many other tasks [13, 38, 63], since they may not appeal to parallel implementation [35], leading to unsatisfactory efficiency for the edge detection task. Inspired from [19] and [20], we use the separable depth-wise convolutional structure with a shortcut for fast inference and easy training. The whole backbone has 4 stages and max pooling layers are among them for down sampling. Each stage has 4 residual blocks (except the first stage that has an initial convolutional layer and 3 residual blocks). The residual path in each block includes a depth-wise convolutional layer, a ReLU layer, and a point-wise convolutional layer sequentially. The number of channels in each stage is reasonably small to avoid big model size (C , $2 \times C$, $4 \times C$ and $4 \times C$ channels for stage 1, 2, 3, and 4 respectively).

Efficient Side Structure. To learn rich hierarchical edge representation, we also use the side structure as in [60] to generate an edge map from each stage respectively, based on which a side loss is computed with the ground truth map to provide deep supervision [60]. To refine the feature maps, beginning from the end of each stage, we firstly build a compact dilation convolution based module (CDCM) to enrich multi-scale edge information, which takes the input with $n \times C$ channels, and produces M ($M < C$) channels in the output to relieve the computation overhead, followed by a compact spatial attention module (CSAM) to eliminate the background noise. After that, a 1×1 convolutional layer further reduces the feature volume to a single channel map, which is then interpolated to the original size followed by a Sigmoid function to create the edge map. The final edge map, which is used for testing, is created by fusing the 4 single channel feature maps with a concatenation, a convolutional layer and a Sigmoid function.

The detailed structure information can be seen in Fig. 5, noting that we do not use any normalization layers for simplicity since the resolutions of the training images are not uniform. The obtained architecture is our baseline. By replacing the vanilla convolution in the 3×3 depth-wise convolutional layer in the residual blocks with PDC, we get the proposed PiDiNet.

Loss Function. We adopt the annotator-robust loss function proposed in [31] for each generated edge map (including the final edge map). For the i th pixel in the j th edge map with value p_i^j , the loss is calculated as:

$$l_i^j = \begin{cases} \alpha \cdot \log(1 - p_i^j) & \text{if } y_i = 0 \\ 0 & \text{if } 0 < y_i < \eta \\ \beta \cdot \log p_i^j & \text{otherwise,} \end{cases} \quad (4)$$

where y_i is the ground truth edge probability, η is a pre-defined threshold, meaning that a pixel is discarded and not

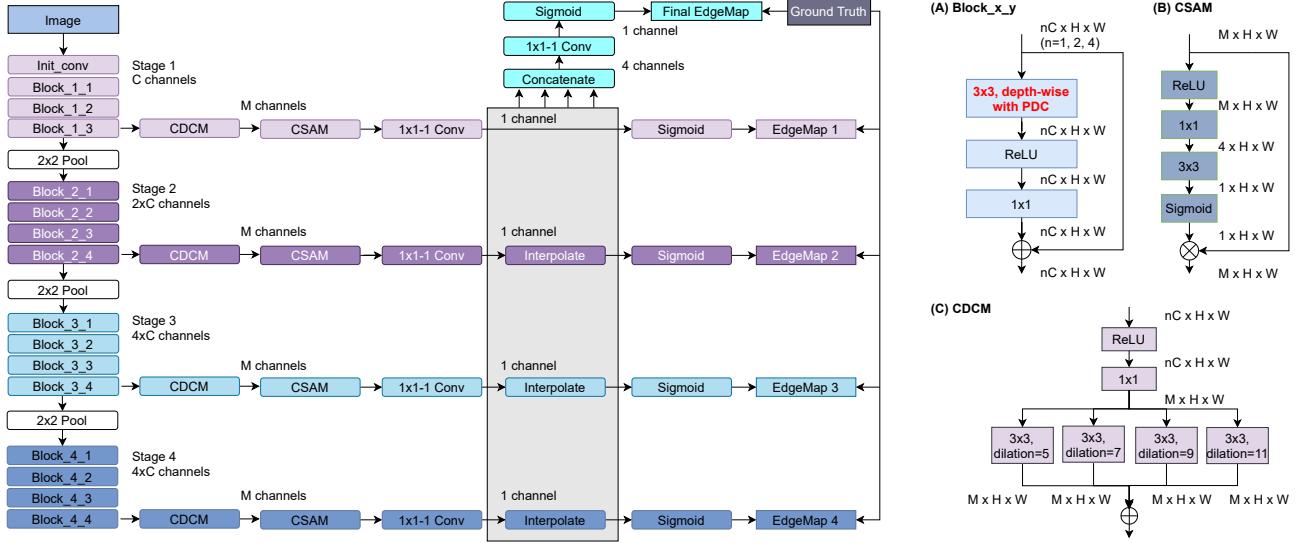


Figure 5. PiDiNet architecture.

considered to be a sample when calculating the loss if it is marked as positive by fewer than η of annotators to avoid confusing, β is the percentage of negative pixel samples and $\alpha = \lambda \cdot (1 - \beta)$. After all, the total loss is $L = \sum_{i,j} l_i^j$.

5. Experiments

5.1. Datasets and Implementation

Experimental Datasets. We evaluate the proposed PiDiNet on three widely used datasets, namely, BSDS500 [1], NYUD [48], and Multicue [39]. The experimental settings about data augmentation and configuration on the three datasets follow [60, 31, 18] and the details are given below. BSDS500 consists of 200, 100, and 200 images in the training set, validation set, and test set respectively. Each image has 4 to 9 annotators. Training images in the dataset are augmented with flipping ($2\times$), scaling ($3\times$), and rotation ($16\times$), leading to a training set that is $96\times$ larger than the unaugmented version. Like prior works [60, 31, 18], the PASCAL VOC Context dataset [40], which has 10K labeled images (and augmented to 20K with flipping), is also optionally considered in training. NYUD has 1449 pairs of aligned RGB and depth images which are densely labeled. There are 381, 414 and 654 images for training, validation, and test respectively. We combine the training and validation set and augment them with flipping ($2\times$), scaling ($3\times$), and rotation ($4\times$) to produce the training data. Multicue is composed of 100 challenging natural scenes and each scene contains a left- and right-view color sequences captured by a binocular stereo camera. The last frame of left-view sequences for each scene, which is labeled with edges and boundaries, is used in our experiments. We randomly split them to 80 and 20 im-

ages for training and evaluation respectively. The process is independently repeated twice more. The metrics are then recorded from the three runs. We also augment each training image with flipping ($2\times$), scaling ($3\times$), and rotation ($16\times$), then randomly crop them with size 500×500 .

Performance Metrics. During evaluation, *F-measure* at both Optimal Dataset Scale (ODS) and Optimal Image Scale (OIS) are recorded for all datasets. Since efficiency is one of the main focuses in this paper, all the models are compared based on the evaluations from single scale images if not specified.

Implementation Details. Our implementation is based on the Pytorch library [44]. In detail, PiDiNet (and the baseline) is randomly initialized and trained for 14 epochs with Adam optimizer [22] with an initial learning rate 0.005, which is decayed in a multi-step way (at epoch 8 and 12 with decaying rate 0.1). If VOC dataset is used in training for evaluating BSDS500, we train 20 epochs and decay the learning rate at epoch 10 and 16. λ is set to 1.1 for both BSDS500 and Multicue, and 1.3 for NYUD. The threshold η is set to 0.3 for both BSDS500 and Multicue. No η is needed for NYUD since the images are singly annotated.

5.2. Ablation Study

To demonstrate the effectiveness of PDC and to find the possibly optimal architecture configuration, we conduct our ablation study on the BSDS500 dataset, where we use the data augmented from the 200 images in the training set (optionally mixed with the VOC dataset) for training and record the metrics on the validation set.

Architecture Configuration. We can replace the vanilla convolution with PDC in any block (we also regard the ini-

Table 2. Possible configurations of PiDiNet. ‘C’, ‘A’, ‘R’ and ‘V’ indicate CPDC, APDC, RPDC and vanilla convolution respectively. ‘ $\times n$ ’ means repeating the pattern for n times sequentially. For example, the baseline architecture can be presented as “[V] $\times 16$ ”, and ‘C-[V] $\times 15$ ’ means using CPDC in the first block and vanilla convolutions in the later blocks. All the models are trained using BSDS500 training set and the VOC dataset, then evaluated on BSDS500 validation set.

Architecture	C-[V] $\times 15$	A-[V] $\times 15$	R-[V] $\times 15$
ODS / OIS	0.775 / 0.794	0.774 / 0.794	0.774 / 0.792
Architecture	[CVVV] $\times 4$	[AVVV] $\times 4$	[RVVV] $\times 4$
ODS / OIS	0.773 / 0.792	0.771 / 0.790	0.772 / 0.791
Architecture	[CCCV] $\times 4$	[AAAV] $\times 4$	[RRRV] $\times 4$
ODS / OIS	0.772 / 0.791	0.775 / 0.793	0.771 / 0.787
Architecture	[C] $\times 16$	[A] $\times 16$	[R] $\times 16$
ODS / OIS	0.767 / 0.786	0.768 / 0.786	0.758 / 0.777
Architecture	Baseline	[CARV] $\times 4$ (PiDiNet)	
ODS / OIS	0.772 / 0.792	0.776 / 0.795	

Table 3. More comparisons between PiDiNet and the baseline architecture in multiple network scales by changing the number of channels C (see Fig. 5). The models are trained using the BSDS500 training set, and evaluated on BSDS500 validation set.

Scale	Baseline (ODS / OIS)	PiDiNet (ODS / OIS)
Tiny ($C=20$)	0.735 / 0.752	0.747 / 0.764
Small ($C=30$)	0.738 / 0.759	0.752 / 0.769
Normal ($C=60$)	0.736 / 0.751	0.757 / 0.776

Table 4. Ablation on CDCM, CSAM and shortcuts. The models are trained with BSDS500 training set and VOC dataset, and evaluated on BSDS500 validation set.

CSAM	CDCM	Shortcuts	ODS / OIS
\times	\times	✓	0.770 / 0.790
\times	✓	✓	0.775 / 0.793
✓	✓	✓	0.776 / 0.795
✓	✓	\times	0.734 / 0.755

tial convolutional layer as a block in the context) in the backbone. Since there are 16 blocks, and a brute force search for the architecture configurations is not feasible, hence we only sample some of them as shown in Table 2 by gradually increasing the number of PDCs. We found replacing the vanilla convolution with PDC only in a single block can even have obvious improvement. More replacements with the same type of PDC may no longer give extra performance gain and instead degenerate the model. We conjecture that the PDC in the first block already obtains much gradient information from the raw image, and an abuse of PDC may even cause the model fail to preserve useful in-

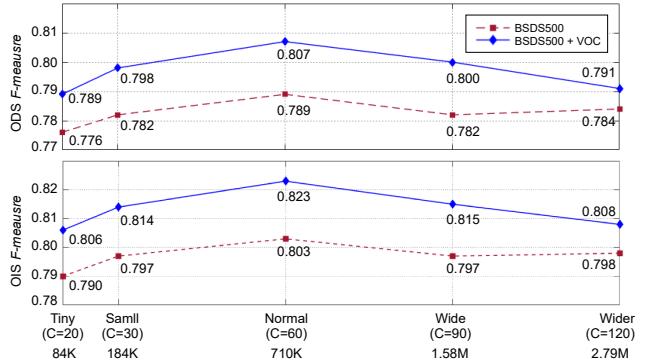


Figure 6. Exploration on the scalability of PiDiNet. The structure sizes are changed by slimming or widening the basic PiDiNet. Bottom row shows the number of parameters for each model. The models are trained with or without VOC dataset.

formation. The extreme case is that when all the blocks are configured with PDC, the performance becomes worse than that of the baseline. The best configuration is ‘[CARV] $\times 4$ ’, which means combining the 4 types of convolutions sequentially in each stage, as different types of PDC capture the gradient information in different encoding directions. We will use this configuration in the following experiments.

To further demonstrate the superiority of PiDiNet over the baseline, which only uses the vanilla convolution, we give more comparisons as shown in Table 3. It constantly proves that PDC configured architectures outperform the corresponding vanilla convolution configured architectures.

CSAM, CDCM and Shortcuts. The effectiveness of CSAM, CDCM and residual structures are demonstrated in Table 4. The addition of shortcuts is simple yet important, as they can help preserve the gradient information captured by the previous layers. On the other hand, the attention mechanism in CSAM and dilation convolution in CDCM can give extra performance gains, while may also bring some computational cost. Therefore, they can be used to tradeoff between accuracy and efficiency. In the following experiments, we note PiDiNet without CSAM and CDCM as PiDiNet-L (meaning a more lightweight version).

5.3. Network Scalability

PiDiNet is highly compact with only 710K parameters and support training from scratch with limited training data. Here, we explore the scalability of PiDiNet with different model complexities as shown in Fig. 6. In order to compare with other approaches, the models are trained in two schemes, both use the BSDS500 training and validation set, while with or without mixing the VOC dataset during training. Metrics are recorded on BSDS500 test set. As expected, compared with the basic PiDiNet, smaller models suffer from lower network capacity and thus with degenerated performances in terms of both ODS and OIS scores. At

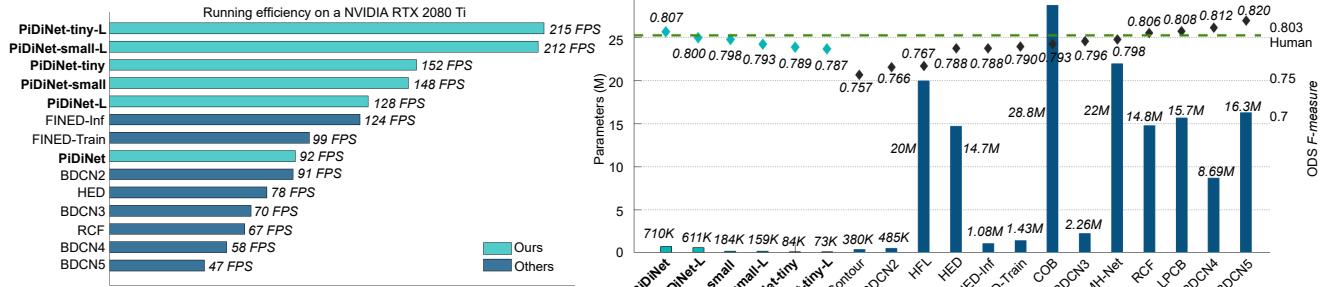


Figure 7. Comparison with other methods in terms of network complexity, running efficiency and detection performance (on BSDS500 dataset). The running speeds of FINED [56] are cited from the original paper, and the rest are evaluated by our implementations

Table 5. Comparison with other methods on BSDS500 dataset. \ddagger indicates the speeds with our implementations based on a NVIDIA RTX 2080 Ti GPU. \dagger indicates the cited GPU speeds.

Method	ODS	OIS	FPS
Human	.803	.803	
Canny [5]	.611	.676	28
Pb [37]	.672	.695	-
SCG [59]	.739	.758	-
SE [9]	.743	.763	12.5
OEF [16]	.746	.770	2/3
DeepEdge [2]	.753	.772	1/1000 \dagger
DeepContour [47]	.757	.776	1/30 \dagger
HFL [3]	.767	.788	5/6 \dagger
CEDN [62]	.788	.804	10 \dagger
HED [60]	.788	.808	78 \ddagger
DeepBoundary [23]	.789	.811	-
COB [36]	.793	.820	-
CED [55]	.794	.811	-
AMH-Net [61]	.798	.829	-
RCF [31]	.806	.823	67 \ddagger
LPCB [8]	.808	.824	30 \dagger
BDCN [18]	.820	.838	47 \ddagger
FINED-Inf [56]	.788	.804	124 \dagger
FINED-Train [56]	.790	.808	99 \dagger
Baseline	.798	.816	96 \ddagger
PiDiNet	.807	.823	92 $\dagger,*$
PiDiNet-L	.800	.815	128 \ddagger
PiDiNet-Small	.798	.814	148 \ddagger
PiDiNet-Small-L	.793	.809	212 \ddagger
PiDiNet-Tiny	.789	.806	152 \ddagger
PiDiNet-Tiny-L	.787	.804	215 \ddagger

* "PiDiNet" is slightly slower than "Baseline" because RPD is a 5x5 convolution after conversion.

the same time, training with more data constantly leads to higher accuracy. It is noted that the normal scale PiDiNet, can achieve the ODS and OIS scores at the same level as that recorded in the HED approach [60], even when trained from scratch only using the BSDS500 dataset (*i.e.*, 0.789 vs. 0.788 in ODS and 0.803 vs. 0.808 in OIS for PiDiNet vs. HED). However, with limited training data, widening the PiDiNet architecture may cause the overfitting problem,

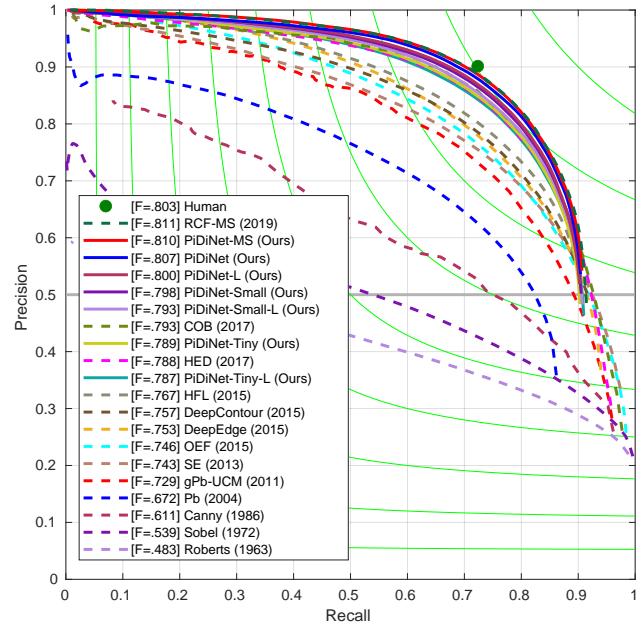


Figure 8. Precision-Recall curves of our models and some competitors on BSDS500 dataset.

as shown in the declines in the second half of the curves. In the following experiments, we only use the tiny, small, and normal versions of PiDiNet, dubbed as PiDiNet-Tiny, PiDiNet-Small and PiDiNet respectively.

5.4. Comparison with State-of-the-arts

On BSDS500 dataset. We compare our methods with prior edge detection approaches including both traditional ones and recently proposed CNN based ones, as summarized in Table 5 and Fig. 8. Firstly, we notice that our baseline model can even achieve comparable results, *i.e.*, with ODS of 0.798 and OIS of 0.816, already beating most CNN based models like CED [55], DeepBoundary [23] and HED [60]. With PDC, PiDiNet can further boost the performance with ODS of 0.807, being the same level as the

Table 6. Comparison with other methods on NYUD dataset. \ddagger indicates the speeds with our implementations based on a NVIDIA RTX 2080 Ti GPU.

Methods	ODS	OIS	ODS	OIS	ODS	OIS	FPS
gPb-UCM [1]	.632	.661					1/360
gPb+NG [14]	.687	.716					1/375
SE [9]	.695	.708					5
SE+NG+ [15]	.710	.723					1/15
	RGB		HHA		RGB-HHA		
HED [60]	.720	.734	.682	.695	.746	.761	62 \ddagger
LPCB [8]	.739	.754	.707	.719	.762	.778	-
RCF [31]	.743	.757	.703	.717	.765	.780	52 \ddagger
AMH-Net [61]	.744	.758	.716	.729	.771	.786	-
BDCN [18]	.748	.763	.707	.719	.765	.781	33 \ddagger
PiDiNet	.733	.747	.715	.728	.756	.773	62 \ddagger
PiDiNet-L	.728	.741	.709	.722	.754	.770	88 \ddagger
PiDiNet-Small	.726	.741	.705	.719	.750	.767	115 \ddagger
PiDiNet-Small-L	.721	.736	.701	.713	.746	.763	165 \ddagger
PiDiNet-Tiny	.721	.736	.700	.714	.745	.763	140 \ddagger
PiDiNet-Tiny-L	.714	.729	.693	.706	.741	.759	206 \ddagger

recently proposed RCF [31] while still achieving nearly 100 FPS. The fastest version PiDiNet-Tiny-L, can also achieve comparable prediction performance with more than 200 FPS, further demonstrating the effectiveness of our methods. Noting all of our modes are trained from scratch using the same amount of training data as in RCF, LPCB, BDCN, *etc.* (*i.e.*, the training and validation set, mixed with the VOC dataset), without the ImageNet pretraining. We also show some qualitative results in Figure 9. A more detailed comparison in terms of network complexity, running efficiency and accuracy can be seen in Fig. 7.

On NYUD dataset. The comparison results on the NYUD dataset are illustrated on Table 6. Following the prior works, we get the ‘RGB-HHA’ results by averaging the output edge maps from RGB image and HHA image to get the final edge map. The quantitative comparison shows that PiDiNets can still achieve highly comparable results among the state-of-the-art methods while being efficient. Please refer to the appendix for the Precision-Recall curves.

On Multicue dataset. We also record the evaluation results on Multicue dataset and the comparison results with other methods are shown on Table 7. Still, PiDiNets achieve promising results with high efficiencies.

6. Conclusion

In conclusion, the contribution in this paper is three-fold: Firstly, we derive the pixel difference convolution which integrates the wisdom from the traditional edge detectors and the advantages of the deep CNNs, leading to robust and accurate edge detection. Secondly, we propose a highly efficient architecture named PiDiNet based on pixel difference convolution, which are memory friendly and with high inference speed. Furthermore, PiDiNet can be trained

Table 7. Comparison with other methods on Multicue dataset. \ddagger indicates the speeds with our implementations based on a NVIDIA RTX 2080 Ti GPU.

Method	Boundary		Edge		FPS
	ODS	OIS	ODS	OIS	
Human [39]	.760 (.017)		.750 (.024)		
Multicue [39]	.720 (.014)		.830 (.002)		-
HED [60]	.814 (.011)	.822 (.008)	.851 (.014)	.864 (.011)	18 \ddagger
RCF [31]	.817 (.004)	.825 (.005)	.857 (.004)	.862 (.004)	15 \ddagger
BDCN [18]	.836 (.001)	.846 (.003)	.891 (.001)	.898 (.002)	9 \ddagger
PiDiNet	.818 (.003)	.830 (.005)	.855 (.007)	.860 (.005)	17 \ddagger
PiDiNet-L	.810 (.005)	.822 (.002)	.854 (.007)	.860 (.004)	23 \ddagger
PiDiNet-Small	.812 (.004)	.825 (.004)	.858 (.007)	.863 (.004)	31 \ddagger
PiDiNet-Small-L	.805 (.007)	.818 (.002)	.854 (.007)	.860 (.004)	44 \ddagger
PiDiNet-Tiny	.807 (.007)	.819 (.004)	.856 (.006)	.862 (.003)	43 \ddagger
PiDiNet-Tiny-L	.798 (.007)	.811 (.005)	.854 (.008)	.861 (.004)	56 \ddagger

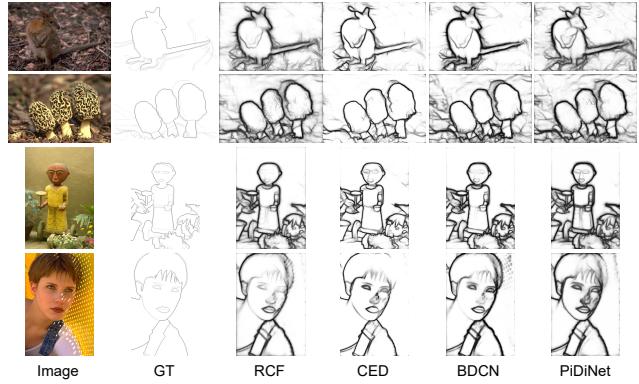


Figure 9. A qualitative comparison of network outputs with some other methods, including RCF [31], CED [55] and BDCN [18].

from scratch only using limited data samples, while achieving human-level performances, breaking the convention that high performance CNN based edge detectors usually need a backbone pretrained on large scale dataset. Thirdly, we conduct extensive experiments on BSDS500, NYUD, and Multicue datasets for edge detection. We believe that PiDiNet has created new state-of-the-art performances considering both accuracy and efficiency.

Future Work. As discussed in Section 1, edge detection is a low level task for many mid- or high-level vision tasks like semantic segmentation and object detection. Also, some low level tasks like salient object detection may also benefit from the image boundary information. We hope pixel difference convolution and the proposed PiDiNet can go further and be useful in these related tasks.

Acknowledgement. This work was partially supported by the Academy of Finland under grant 331883 and the National Natural Science Foundation of China under Grant 61872379, 62022091, and 71701205. The authors also wish to acknowledge CSC IT Center for Science, Finland, for computational resources.

References

- [1] 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, 2011. [5](#), [8](#), [13](#), [14](#)
- [2] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *CVPR*, pages 4380–4389, 2015. [1](#), [2](#), [3](#), [7](#)
- [3] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision. In *ICCV*, pages 504–512, 2015. [2](#), [3](#), [7](#)
- [4] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Semantic segmentation with boundary neural fields. In *CVPR*, pages 3602–3610, 2016. [1](#)
- [5] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):679–698, 1986. [1](#), [3](#), [7](#)
- [6] Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip Torr. Bing: Binarized normed gradients for objectness estimation at 300fps. In *CVPR*, pages 3286–3293, 2014. [1](#)
- [7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009. [2](#)
- [8] Ruoxi Deng, Chunhua Shen, Shengjun Liu, Huibing Wang, and Xinru Liu. Learning to predict crisp boundaries. In *ECCV*, pages 562–578, 2018. [2](#), [7](#), [8](#)
- [9] Piotr Dollár and C Lawrence Zitnick. Fast edge detection using structured forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(8):1558–1570, 2015. [1](#), [3](#), [7](#), [8](#), [13](#)
- [10] James H Elder and Richard M Goldberg. Image editing in the contour domain. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 374–381. IEEE, 1998. [1](#)
- [11] Vittorio Ferrari, Loic Fevrier, Frederic Jurie, and Cordelia Schmid. Groups of adjacent contour segments for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1):36–51, 2008. [1](#)
- [12] David R Martin Charless C Fowlkes and Jitendra Malik. Learning to detect natural image boundaries using brightness and texture. *NeurIPS*, 2002. [1](#)
- [13] Shang-Hua Gao, Yong-Qiang Tan, Ming-Ming Cheng, Chengze Lu, Yunpeng Chen, and Shuicheng Yan. Highly efficient salient object detection with 100k parameters. In *ECCV*, pages 702–721. Springer, 2020. [3](#), [4](#)
- [14] Saurabh Gupta, Pablo Arbelaez, and Jitendra Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *CVPR*, pages 564–571, 2013. [8](#), [13](#)
- [15] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*, pages 345–360. Springer, 2014. [1](#), [8](#)
- [16] Sam Hallman and Charless C Fowlkes. Oriented edge forests for boundary detection. In *CVPR*, pages 1732–1740, 2015. [1](#), [7](#), [13](#)
- [17] Song Han, Huiyi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016. [4](#)
- [18] Jianzhong He, Shiliang Zhang, Ming Yang, Yanhu Shan, and Tiejun Huang. Bi-directional cascade network for perceptual edge detection. In *CVPR*, pages 3828–3837, 2019. [2](#), [5](#), [7](#), [8](#)
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. [2](#), [3](#), [4](#)
- [20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. [2](#), [4](#)
- [21] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. Local binary convolutional neural networks. In *CVPR*, pages 19–28, 2017. [3](#)
- [22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015. [5](#)
- [23] Iasonas Kokkinos. Pushing the boundaries of boundary detection using deep learning. *arXiv preprint arXiv:1511.07386*, 2015. [1](#), [2](#), [7](#)
- [24] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, pages 6129–6138, 2017. [3](#)
- [25] Gen Li and Joongkyu Kim. Dabnet: Depth-wise asymmetric bottleneck for real-time semantic segmentation. In *BMVC*, page 259. BMVA Press, 2019. [3](#)
- [26] Jiang-Jiang Liu, Qibin Hou, and Ming-Ming Cheng. Dynamic feature integration for simultaneous detection of salient object, edge, and skeleton. *IEEE Transactions on Image Processing*, 29:8652–8667, 2020. [3](#)
- [27] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chellappa, and Matti Pietikäinen. From bow to cnn: Two decades of texture representation for texture classification. *International Journal of Computer Vision*, 127(1):74–109, 2019. [3](#)
- [28] Li Liu, Paul Fieguth, Gangyao Kuang, and Hongbin Zha. Sorted random projections for robust texture classification. In *ICCV*, pages 391–398. IEEE, 2011. [3](#)
- [29] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2):261–318, 2020. [1](#)
- [30] Li Liu, Lingjun Zhao, Yunli Long, Gangyao Kuang, and Paul Fieguth. Extended local binary patterns for texture classification. *Image and Vision Computing*, 30(2):86–99, 2012. [3](#)
- [31] Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Jia-Wang Bian, Le Zhang, Xiang Bai, and Jinhui Tang. Richer convolutional features for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1939–1946, 2019. [1](#), [2](#), [4](#), [5](#), [7](#), [8](#), [13](#)

- [32] Yu Liu and Michael S Lew. Learning relaxed deep supervision for better edge detection. In *CVPR*, pages 231–240, 2016. 3
- [33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015. 2
- [34] Shangzhen Luan, Chen Chen, Baochang Zhang, Jungong Han, and Jianzhuang Liu. Gabor convolutional networks. *IEEE Transactions on Image Processing*, 27(9):4357–4366, 2018. 3
- [35] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, pages 116–131, 2018. 4
- [36] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. Convolutional oriented boundaries. In *ECCV*, pages 580–596. Springer, 2016. 2, 7
- [37] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, 2004. 1, 7
- [38] Sachin Mehta, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *CVPR*, pages 9190–9200, 2019. 3, 4
- [39] David A Mély, Junkyung Kim, Mason McGill, Yuliang Guo, and Thomas Serre. A systematic comparison between visual cues for boundary detection. *Vision Research*, 120:93–107, 2016. 5, 8
- [40] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, pages 891–898, 2014. 5, 13
- [41] R Muthukrishnan and Miyilsamy Radha. Edge detection techniques for image segmentation. *International Journal of Computer Science & Information Technology*, 3(6):259, 2011. 1
- [42] Timo Ojala, Matti Pietikäinen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002. 3
- [43] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 3
- [44] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019. 5
- [45] Xavier Soria Poma, Edgar Riba, and Angel Sappa. Dense extreme inception network: Towards a robust cnn model for edge detection. In *WACV*, pages 1923–1932, 2020. 3, 4
- [46] Judith MS Prewitt. Object enhancement and extraction. *Picture processing and Psychopictorics*, 10(1):15–19, 1970. 1
- [47] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Zhi-jiang Zhang. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *CVPR*, pages 3982–3991, 2015. 1, 2, 7
- [48] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 5, 13
- [49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2
- [50] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *A Talk at The Stanford Artificial Project in*, pages 271–272, 1968. 1
- [51] Zhuo Su, Linpu Fang, Wenxiong Kang, Dewen Hu, Matti Pietikäinen, and Li Liu. Dynamic group convolution for accelerating convolutional neural networks. In *ECCV*, pages 138–155. Springer, 2020. 4
- [52] Zhuo Su, Matti Pietikäinen, and Li Liu. Bird: Learning binary and illumination robust descriptor for face recognition. In *BMVC*, page 102, 2019. 3
- [53] Vincent Torre and Tomaso A Poggio. On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):147–163, 1986. 1
- [54] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. 1
- [55] Yupei Wang, Xin Zhao, and Kaiqi Huang. Deep crisp boundaries. In *CVPR*, pages 3892–3900, 2017. 1, 2, 7, 8
- [56] Jan Kristanto Wibisono and Hsueh-Ming Hang. Fined: Fast inference network for edge detection. *arXiv preprint arXiv:2012.08392*, 2020. 3, 4, 7
- [57] Jan Kristanto Wibisono and Hsueh-Ming Hang. Traditional method inspired deep neural network for edge detection. In *ICIP*, pages 678–682. IEEE, 2020. 3, 4
- [58] Tianyi Wu, Sheng Tang, Rui Zhang, Juan Cao, and Yong-dong Zhang. Cgnet: A light-weight context guided network for semantic segmentation. *IEEE Transactions on Image Processing*, 30:1169–1179, 2020. 3
- [59] Ren Xiaofeng and Liefeng Bo. Discriminatively trained sparse code gradients for contour detection. In *NeurIPS*, pages 584–592, 2012. 1, 7
- [60] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *International Journal of Computer Vision*, 125(1-3):3–18, 2017. 1, 2, 4, 5, 7, 8
- [61] 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. In *NeurIPS*, pages 3961–3970, 2017. 2, 7, 8
- [62] Jimei Yang, Brian Price, Scott Cohen, Honglak Lee, and Ming-Hsuan Yang. Object contour detection with a fully convolutional encoder-decoder network. In *CVPR*, pages 193–202, 2016. 2, 7

- [63] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, pages 325–341, 2018. [3](#), [4](#)
- [64] Zitong Yu, Yunxiao Qin, Hengshuang Zhao, Xiaobai Li, and Guoying Zhao. Dual-cross central difference network for face anti-spoofing. In *IJCAI*, 2021. [3](#)
- [65] Zitong Yu, Jun Wan, Yunxiao Qin, Xiaobai Li, Stan Z Li, and Guoying Zhao. Nas-fas: Static-dynamic central difference network search for face anti-spoofing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. [3](#)
- [66] Zitong Yu, Chenxu Zhao, Zezheng Wang, Yunxiao Qin, Zhuo Su, Xiaobai Li, Feng Zhou, and Guoying Zhao. Searching central difference convolutional networks for face anti-spoofing. In *CVPR*, pages 5295–5305, 2020. [3](#)
- [67] Zitong Yu, Benjia Zhou, Jun Wan, Pichao Wang, Haoyu Chen, Xin Liu, Stan Z Li, and Guoying Zhao. Searching multi-rate and multi-modal temporal enhanced networks for gesture recognition. *IEEE Transactions on Image Processing*, 2021. [3](#)

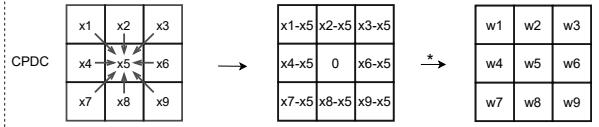


Figure 10. Selection of pixel pairs and convolution in CPDC.

7. Appendix

7.1. Converting Pixel Difference Convolution (PDC) to Vanilla Convolution

The main goal of the conversion is to make PDC as fast and memory efficient as as the vanilla convolution. As introduced in the main paper, the formulations of vanilla convolution and PDC can be written as:

$$y = f(\mathbf{x}, \theta) = \sum_{i=1}^{k \times k} w_i \cdot x_i, \quad (\text{vanilla convolution}) \quad (5)$$

$$y = f(\nabla \mathbf{x}, \theta) = \sum_{(x_i, x'_i) \in \mathcal{P}} w_i \cdot (x_i - x'_i), \quad (\text{PDC}) \quad (6)$$

where, x_i and x'_i are the pixels in the current input local patch, w_i is the weight in the $k \times k$ convolution kernel. $\mathcal{P} = \{(x_1, x'_1), (x_2, x'_2), \dots, (x_m, x'_m)\}$ is the set of pixel pairs picked from the local patch, and $m \leq k \times k$.

The conversion from PDC to vanilla convolution can be done in both the training and inference phases.

Conversion in the Training Phase. Eq. 6 can be transformed to fit the form of Eq. 5, according to the selection strategies of the pixel pairs. Correspondingly, PDC can be converted to vanilla convolution by firstly transforming the kernel weights to a new set of kernel weights, followed by a vanilla convolutional operation. We will discuss Central PDC (CPDC), Angular PDC (APDC) and Radial PDC (RPDC) respectively. The selection strategies of pixel pairs in the three PDC instances are shown in Fig. 10, Fig. 11 and Fig. 12. The transformations of the equations are as follows.

For CPDC (Fig. 10):

$$\begin{aligned} y &= w_1 \cdot (x_1 - x_5) + w_2 \cdot (x_2 - x_5) + w_3 \cdot (x_3 - x_5) \\ &\quad + w_4 \cdot (x_4 - x_5) + w_6 \cdot (x_6 - x_5) + w_7 \cdot (x_7 - x_5) \\ &\quad + w_8 \cdot (x_8 - x_5) + w_9 \cdot (x_9 - x_5) \\ &= w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \\ &\quad + w_4 \cdot x_4 + w_6 \cdot x_6 + w_7 \cdot x_7 + \\ &\quad + w_8 \cdot x_8 + w_9 \cdot x_9 \\ &\quad + \left(- \sum_{i=\{1,2,3,4,6,7,8,9\}} w_i\right) \cdot x_5 \\ &= \hat{w}_1 \cdot x_1 + \hat{w}_2 \cdot x_2 + \hat{w}_3 \cdot x_3 + \dots = \sum \hat{w}_i \cdot x_i \end{aligned} \quad (7)$$

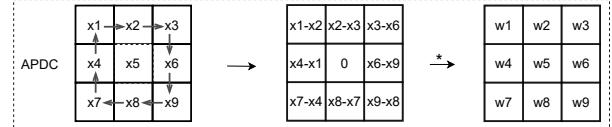


Figure 11. Selection of pixel pairs and convolution in APDC.

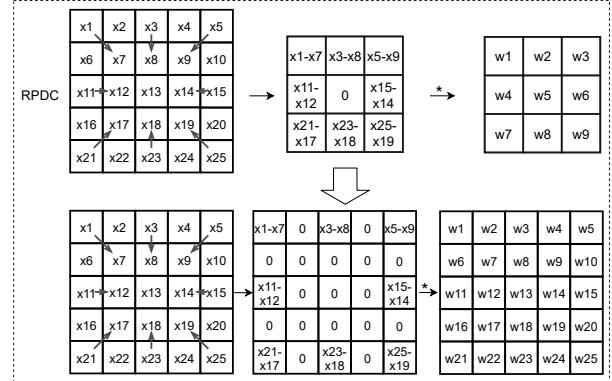


Figure 12. Selection of pixel pairs and convolution in RPDC.

For APDC (Fig. 11):

$$\begin{aligned} y &= w_1 \cdot (x_1 - x_2) + w_2 \cdot (x_2 - x_3) + w_3 \cdot (x_3 - x_6) \\ &\quad + w_4 \cdot (x_4 - x_1) + w_6 \cdot (x_6 - x_9) + w_7 \cdot (x_7 - x_4) \\ &\quad + w_8 \cdot (x_8 - x_7) + w_9 \cdot (x_9 - x_8) \\ &= (w_1 - w_4) \cdot x_1 + (w_2 - w_1) \cdot x_2 + (w_3 - w_2) \cdot x_3 \\ &\quad + (w_4 - w_7) \cdot x_4 + (w_6 - w_3) \cdot x_6 + (w_7 - w_8) \cdot x_7 \\ &\quad + (w_8 - w_9) \cdot x_8 + (w_9 - x_6) \cdot x_9 \\ &\quad + 0 \cdot x_5 \\ &= \hat{w}_1 \cdot x_1 + \hat{w}_2 \cdot x_2 + \hat{w}_3 \cdot x_3 + \dots = \sum \hat{w}_i \cdot x_i \end{aligned} \quad (8)$$

For RPDC (Fig. 12):

$$\begin{aligned} y &= w_1 \cdot (x_1 - x_7) + w_3 \cdot (x_3 - x_8) + w_5 \cdot (x_5 - x_9) \\ &\quad + w_{11} \cdot (x_{11} - x_{12}) + w_{15} \cdot (x_{15} - x_{14}) \\ &\quad + w_{21} \cdot (x_{21} - x_{17}) + w_{23} \cdot (x_{23} - x_{18}) \\ &\quad + w_{25} \cdot (x_{25} - x_{19}) \\ &= w_1 \cdot x_1 + w_3 \cdot x_3 + w_5 \cdot x_5 \\ &\quad + (-w_1) \cdot x_7 + (-w_3) \cdot x_8 + (-w_5) \cdot x_9 + \\ &\quad + w_{11} \cdot x_{11} + (-w_{11}) \cdot x_{12} + (-w_{15}) \cdot x_{14} \\ &\quad + w_{15} \cdot x_{15} + (-w_{21}) \cdot x_{17} + (-w_{23}) \cdot x_{18} \\ &\quad + (-w_{25}) \cdot x_{19} + w_{21} \cdot x_{21} + w_{23} \cdot x_{23} \\ &\quad + w_{25} \cdot x_{25} + \sum_{i=\{2,4,6,10,13,16,20,22,24\}} 0 \cdot x_i \\ &= \hat{w}_1 \cdot x_1 + \hat{w}_2 \cdot x_2 + \hat{w}_3 \cdot x_3 + \dots = \sum \hat{w}_i \cdot x_i \end{aligned} \quad (9)$$

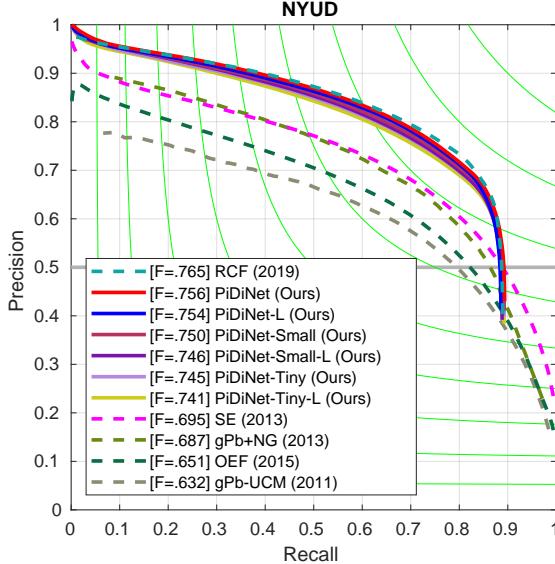


Figure 13. Precision-Recall curves of our models and some competitors on NYUD dataset.

The RPDC is converted to a vanilla convolution with kernel size 5×5 .

Conversion in the Inference Phase. After training, instead of saving the original weights w_i , we directly save the new set of weights \hat{w}_i . Therefore, during inference, all the convolutional operations are vanilla convolutions.

7.2. Precision-Recall Curves on NYUD Dataset

The Precision-Recall curves of our methods and other approaches on NYUD dataset [48] are shown in Fig. 13. The compared methods include RCF [31], SE [9], gPb+NG [14], gPb-UCM [1] and OEF [16].

7.3. Visualization

Edge Maps. The edge maps generated from the baseline architecture and PiDiNet are shown in Fig. 14. Both models were trained using only the BSDS500 dataset without the mixed VOC dataset [40]. From the figure, it is proved that PDC can help PiDiNet effectively capture more useful boundaries, with the ability to extract rich gradient information that facilitates edge detection.

Intermediate Feature Maps. We also visualize the intermediate feature maps extracted from PiDiNet, to qualitatively demonstrate the effectiveness of the compact dilation convolution based module (CDCM) and the compact spatial attention module (CSAM), which are shown in Fig. 15. It is concluded that both CDCM and CSAM take a positive role in PiDiNet on the edge detection task.

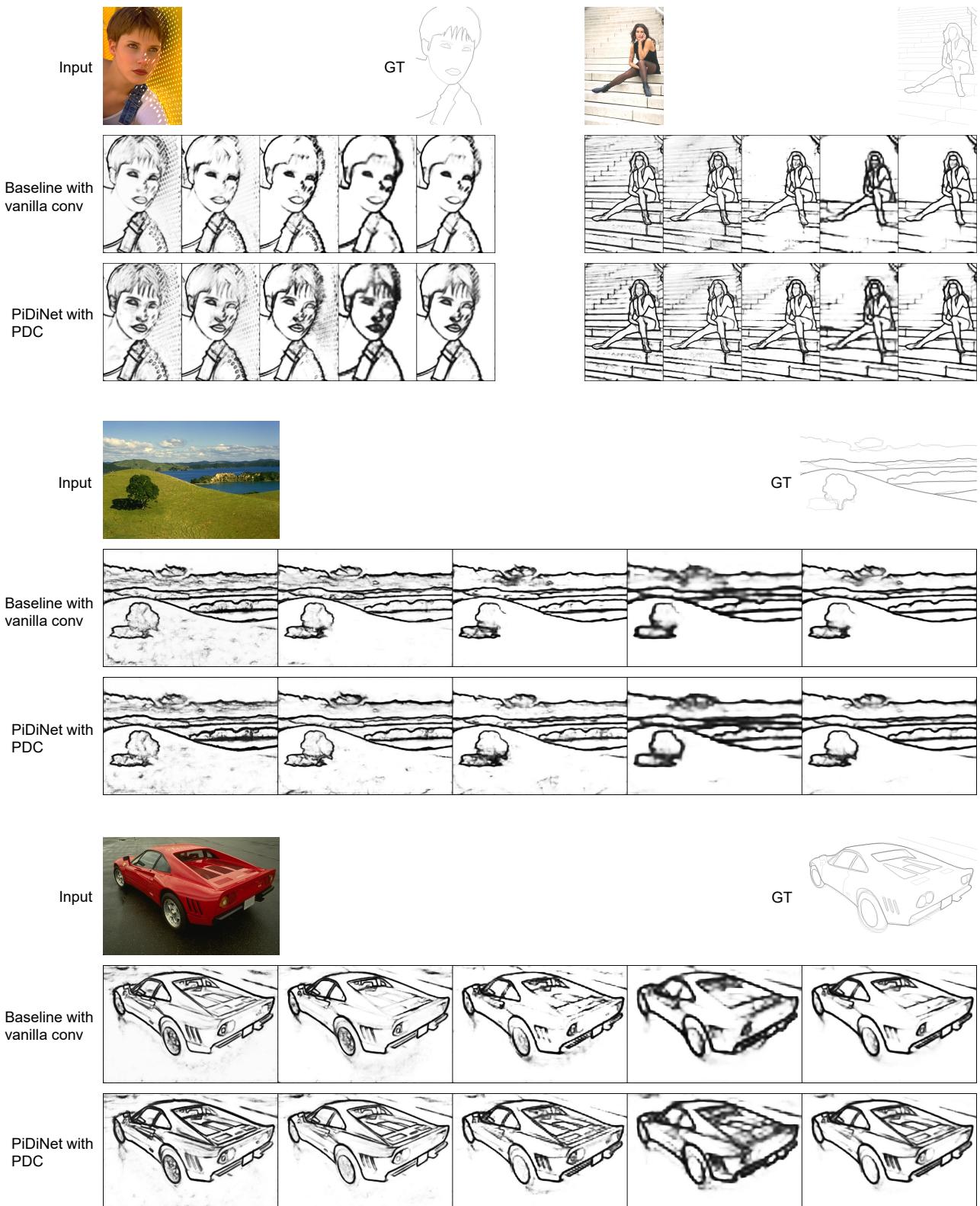


Figure 14. For each case, Top: input and ground truth image; Middle: edge maps from stage 1, 2, 3, 4 respectively and the final edge map, generated from the baseline architecture, Bottom: Corresponding edge maps generated from PiDiNet. Both the baseline architecture and PiDiNet were trained only using the BSDS500 dataset [1]. Compared with the baseline, we can see that PiDiNet can detect more useful boundaries (e.g., bangs, stairs, the contour of the tree, the characteristic textures of the car).

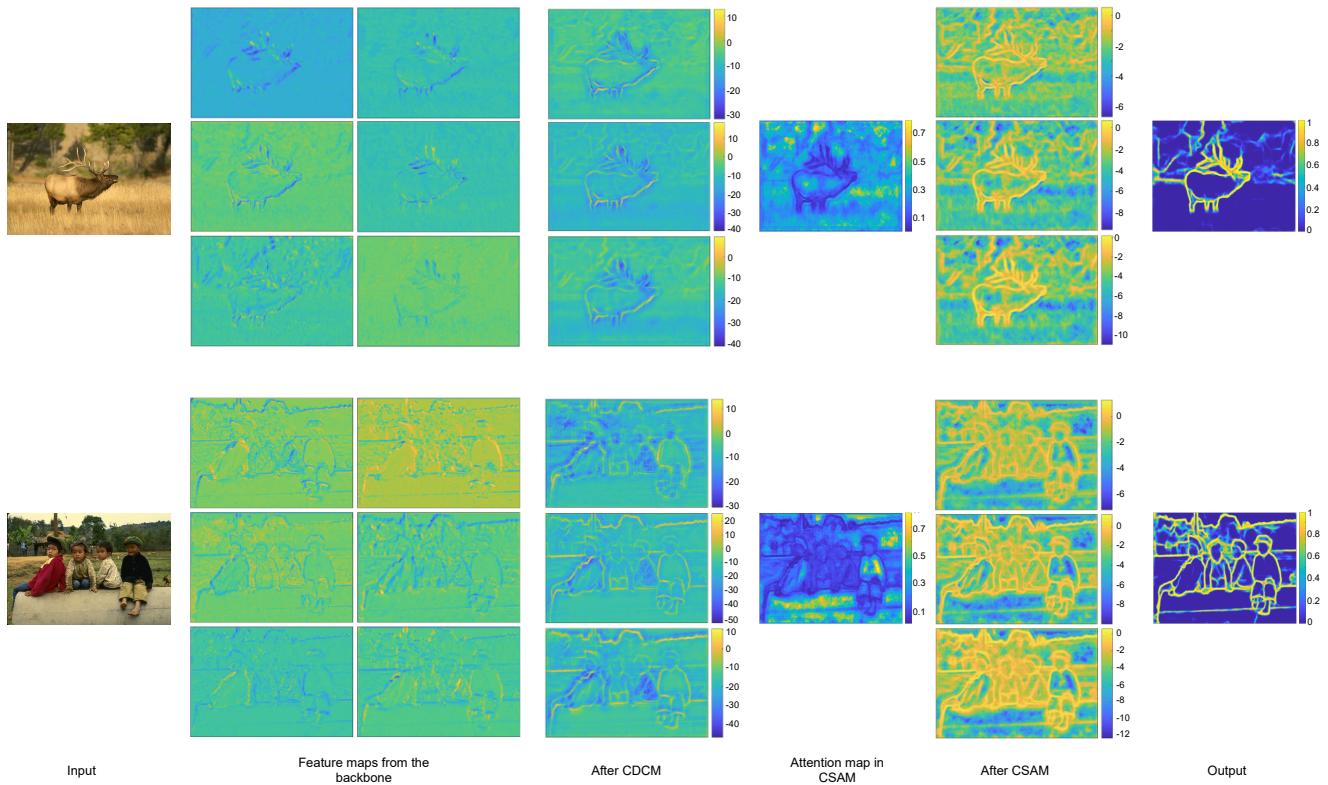


Figure 15. CDCM and CSAM can further refine the feature maps with multi-scale feature extraction and the sample adaptive spatial attention mechanism. Note that in the attention maps generated by CSAM, pixels in the background show higher intensities. This makes sense as the background pixels after CDCM have negative values, hence they will be additionally suppressed through CSAM.