Pixel Difference Networks for Efficient Edge Detection

Zhuo Su\textsuperscript{1,*} Wenzhe Liu\textsuperscript{2,*} Zitong Yu\textsuperscript{1} Dewen Hu\textsuperscript{2} Qing Liao\textsuperscript{3} Qi Tian\textsuperscript{4} Matti Pietikäinen\textsuperscript{1} Li Liu\textsuperscript{2,1,†}

\textsuperscript{1}Center for Machine Vision and Signal Analysis, University of Oulu, Finland
\textsuperscript{2}National University of Defense Technology, China
\textsuperscript{3}Harbin Institute of Technology (Shenzhen), China
\textsuperscript{4}Xidian 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 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 [33]. 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...
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.

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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Params</td>
<td>14.7M</td>
<td>14.8M</td>
<td>16.3M</td>
<td>710K</td>
<td>73K</td>
</tr>
<tr>
<td>MACs</td>
<td>22.2G</td>
<td>16.2G</td>
<td>23.2G</td>
<td>3.43G</td>
<td>270M</td>
</tr>
<tr>
<td>Throughput</td>
<td>78FPS</td>
<td>67FPS</td>
<td>47FPS</td>
<td>92FPS</td>
<td>215FPS</td>
</tr>
<tr>
<td>Pre-training</td>
<td>ImageNet</td>
<td>ImageNet</td>
<td>ImageNet</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ODS F-measure</td>
<td>0.788</td>
<td>0.806</td>
<td>0.820</td>
<td>0.807</td>
<td>0.787</td>
</tr>
</tbody>
</table>

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.
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 of vanilla convolution and PDC can be written as:

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

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

where, \( x_i \) and \( x_i' \) are the input pixels, \( w_i \) is the weight in the \( k \times k \) convolution kernel. \( \mathcal{P} = \{(x_1, x_1'), (x_2, x_2'), \ldots, (x_m, x_m')\} \) 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 will 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 \( 3 \times 3 \), we create 8 pairs in the angular direction in the \( 3 \times 3 \) local patch (thus \( m = 8 \)), then the pixel differences obtained from the pairs are con-

![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.](image-url)
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 representation 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 responses\(^1\). By training from abundant of data, PDC is able to automatically learn rich representative encodings for the task.

\[
\begin{aligned}
\text{x1} & \xrightarrow{w2} \text{x2} \xrightarrow{w3} \text{x3} \\
\text{x4} & \xrightarrow{w4} \text{x5} \xrightarrow{w6} \text{x6} \\
\text{x7} & \xrightarrow{w7} \text{x8} \xrightarrow{w9} \text{x9}
\end{aligned}
\]

\[
\begin{aligned}
x_{12} & = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \\
x_{46} & = x_4 \cdot w_4 + x_5 \cdot w_5 + x_6 \cdot w_6 \\
x_{79} & = x_7 \cdot w_7 + x_8 \cdot w_8 + x_9 \cdot w_9
\end{aligned}
\]

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

**Converting PDC to Vanilla Convolution.** According to Eq. 2, 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. 4), conversion is done with the following equations:

\[
y = w_1 \cdot (x_1 - x_2) + w_2 \cdot (x_2 - x_3) + w_3 \cdot (x_3 - x_6) + ... \\
= (w_1 - w_4) \cdot x_1 + (w_4 - w_1) \cdot x_2 + (w_3 - w_2) \cdot x_3 + ... \\
= \tilde{w}_1 \cdot x_1 + \tilde{w}_2 \cdot x_2 + \tilde{w}_3 \cdot x_3 + ... = \sum \tilde{w}_i \cdot x_i. \quad (3)
\]

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).

**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 \(1 \times 1\) 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 \(3 \times 3\) 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 \(p_{ij}^\prime\), the loss is calculated as:

\[
L_i^\prime = \begin{cases} 
\alpha \cdot \log(1 - p_{ij}^\prime) & \text{if } y_i = 0 \\
0 & \text{if } 0 < y_i < \eta \\
\beta \cdot \log p_{ij}^\prime & \text{otherwise}
\end{cases}
\]

where \(y_i\) is the ground truth edge probability, \(\eta\) is a predefined threshold, meaning that a pixel is discarded and not

\(^1\)Usually, higher activation responses are considered to be more salient, as adopted in many network pruning methods [17, 51]
considered to be a sample when calculating the loss if it is marked as positive by fewer than \( \eta \) of annotators to avoid confusing, \( \beta \) 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 images 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\( \times \)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. \( \lambda \) is set to 1.1 for both BSDS500 and Multicue, and 1.3 for NYUD. The threshold \( \eta \) is set to 0.3 for both BSDS500 and Multicue. No \( \eta \) 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. ‘x’ means repeating the pattern for n times sequentially. For example, the baseline architecture can be presented as “[V]×16”, and “C-[V]×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.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>ODS / OIS</th>
<th>OIS / OIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-[V]×15</td>
<td>0.775 / 0.794</td>
<td>0.774 / 0.794</td>
</tr>
<tr>
<td>A-[V]×15</td>
<td>0.771 / 0.790</td>
<td>0.772 / 0.791</td>
</tr>
<tr>
<td>R-[V]×15</td>
<td>0.776 / 0.795</td>
<td>0.776 / 0.795</td>
</tr>
<tr>
<td>C-V×16</td>
<td>0.775 / 0.791</td>
<td>0.775 / 0.791</td>
</tr>
<tr>
<td>A-A×16</td>
<td>0.772 / 0.792</td>
<td>0.772 / 0.792</td>
</tr>
<tr>
<td>R-[R]×16</td>
<td>0.776 / 0.796</td>
<td>0.776 / 0.796</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.772 / 0.792</td>
<td>0.776 / 0.795</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Scale</th>
<th>Baseline (ODS / OIS)</th>
<th>PiDiNet (ODS / OIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny (C=20)</td>
<td>0.735 / 0.752</td>
<td>0.747 / 0.764</td>
</tr>
<tr>
<td>Small (C=30)</td>
<td>0.738 / 0.759</td>
<td>0.752 / 0.769</td>
</tr>
<tr>
<td>Normal (C=60)</td>
<td>0.736 / 0.751</td>
<td>0.757 / 0.776</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CSAM</th>
<th>CDCM</th>
<th>Shortcuts</th>
<th>ODS / OIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>0.770 / 0.790</td>
</tr>
<tr>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0.775 / 0.793</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0.776 / 0.795</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>0.734 / 0.755</td>
</tr>
</tbody>
</table>

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
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, 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
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 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


[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

