EDTER: Edge Detection with Transformer

Mengyang Pu\textsuperscript{1,3}, Yaping Huang\textsuperscript{1,*}, Yuming Liu\textsuperscript{2}, Qingji Guan\textsuperscript{1}, Haibin Ling\textsuperscript{3}
\textsuperscript{1}Beijing Key Laboratory of Traffic Data Analysis and Mining, Beijing Jiaotong University, China
\textsuperscript{2}Shenzhen Urban Transport Planning Center Co., Ltd., China
\textsuperscript{3}Department of Computer Science, Stony Brook University, USA
{mengyangpu, yphuang, qjguan}@bjtu.edu.cn; liuyuming@sutpc.com; hling@cs.stonybrook.edu

Abstract

Convolutional neural networks have made significant progresses in edge detection by progressively exploring the context and semantic features. However, local details are gradually suppressed with the enlarging of receptive fields. Recently, vision transformer has shown excellent capability in capturing long-range dependencies. Inspired by this, we propose a novel transformer-based edge detector, Edge Detection TransformER (EDTER), to extract clear and crisp object boundaries and meaningful edges by exploiting the full image context information and detailed local cues simultaneously. EDTER works in two stages. In Stage I, a global transformer encoder is used to capture long-range global context on coarse-grained image patches. Then in Stage II, a local transformer encoder works on fine-grained patches to excavate the short-range local cues. Each transformer encoder is followed by an elaborately designed Bi-directional Multi-Level Aggregation decoder to achieve high-resolution features. Finally, the global context and local cues are combined by a Feature Fusion Module and fed into a decision head for edge prediction. Extensive experiments on BSDS500, NYUDv2, and Multicue demonstrate the superiority of EDTER in comparison with state-of-the-arts. The source code is available at https://github.com/MengyangPu/EDTER.

1. Introduction

Edge detection is one of the most fundamental problems in computer vision and has a wide variety of applications, such as image segmentation [8, 23, 39, 44, 45, 47], object detection [23], and video object segmentation [5, 57, 59]. Given an input image, edge detection aims to extract accurate object boundaries and visually salient edges. It is challenging due to many factors including complex backgrounds, inconsistent annotations, and so on.

*Corresponding author.

![Figure 1. Examples of edge detection.](image-url)
The overall framework of the proposed EDTER is illustrated in Fig. 2. EDTER explores the full image context information and fine-grained cues in two stages. In Stage I, we first split the input image into a sequence of coarse-grained patches and use a global transformer encoder to learn the global context information. Then a Bi-directional Multi-Level Aggregation (BiMLA) decoder to boost the performance of edge detection. RCF [36] combines hierarchical features from all convolutional layers into a holistic architecture. To achieve effective results, BDCN [22] uses layer-specific supervision inferred from a bi-directional cascade structure to guide the training of each layer. PiDiNet [53] integrates the traditional edge detection operators into a CNN model for enhanced performance.

Vision transformer. First introduced to handle natural language tasks [13, 30, 56], transformer is later extended to vision tasks owing to its capacity in modeling long-range dependencies including image classification [16], semantic segmentation [72], and object detection [7]. Recently, it is applied in conjunction with CNN in DETR [7] and the other variants [10, 27, 31, 58, 73]. More recently, vision transformer (ViT) [16] directly uses the transformer to the sequences of image patches and achieves the state-of-the-art. This architecture brings direct inspiration to other computer vision tasks [32, 38, 55, 68, 72]. For example, SETR [72] shows superior accuracy in semantic segmentation using a pure transformer on image patches. These works demonstrate the effectiveness of transformers in capturing long-range dependencies and global context.

Our work is inspired by the above pioneer studies [16, 38, 72], but is significantly different in two aspects. First, the proposed EDTER, to the best of our knowledge, is the first usage of the transformer for generic edge detection. Second, our key idea is to learn the features that contain the global image context and fine-grained local cues by a two-stage framework with an affordable computational cost. With the integration of the global context and local cues, EDTER is superior in edge detection.
Multi-Level Aggregation (BiMLA) decoder is used to generate the high-resolution features. In Stage II, the whole image is divided into multiple sequences of fine-grained patches by sampling with a non-overlapping sliding window. Then we execute a local transformer encoder on each sequence in turn to capture short-range local cues. We integrate all local cues and input them into a local BiMLA decoder to achieve the pixel-level feature maps. Finally, the global and local features are integrated by a Feature Fusion Module (FFM) and then are fed into a decision head to predict the final edge maps.

3.2. Review Vision Transformer

The transformer encoders in our framework follow the vision transformer (ViT) in [16], as briefly described below.

**Image Partition.** The first step in ViT is to transform a 2D image, denoted by $X \in \mathbb{R}^{H \times W \times 3}$, into a 1D sequence of image patches $[16, 72]$. Concretely, we uniformly split $X$ into a sequence of flattened image patches of size $P \times P$, resulting in $\frac{H}{P} \times \frac{W}{P}$ vision tokens. Then, the sequence is mapped into a latent embedding space by a learnable linear projection. The projected features are called patch embeddings. Further, to preserve positional information, the standard learnable 1D position embeddings are added to the patch embeddings. Finally, the combined embeddings (denoted as $z^0$) are fed into the transformer encoder.

**Transformer Encoder.** The standard transformer encoder [56] consists of $L$ transformer blocks. Each block has a multi-head self-attention operation (MSA), a multi-layer perceptron (MLP), and two LayerNorm steps (LN). Moreover, a residual connection layer is applied after each block. Generally, MSA performs $M$ self-attentions in parallel and projects their concatenated outputs. In the $m^{th}$ self-attention, given the output $z^{l-1} \in \mathbb{R}^{N \times C}$ of the $(l-1)^{th}$ transformer block, the queries $Q \in \mathbb{R}^{N \times U}$, keys $K \in \mathbb{R}^{N \times U}$, and values $V \in \mathbb{R}^{N \times U}$ are computed by

$$Q = z^{l-1}W_Q, \quad K = z^{l-1}W_K, \quad V = z^{l-1}W_V, \quad (1)$$

where $z^{l-1} = LN(z^{l-1})$, $W_Q, W_K, W_V \in \mathbb{R}^{C \times U}$ are the parameter matrices, $C$ is the dimension of embeddings, and $U$ is the dimension of $Q$, $K$, and $V$. Then, we compute the output of the $m^{th}$ self-attention based on the pairwise similarity between two elements of the sequence by

$$y^{m}_{sa} = \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right)V. \quad (2)$$

where $y^{m}_{sa}$ is the computed attention weight. Finally, MSA can be formulated as

$$y_{msa} = \text{MSA}(z^{l-1}) = [y^{1}_{msa}, y^{2}_{msa}, \ldots, y^{M}_{msa}] W_O, \quad (3)$$

where $y_{msa}$ is the output of MSA, $W_O \in \mathbb{R}^{M \times U \times C}$ represents the projection parameters, and $[\cdot]$ is the concatenation. In this work, we fix $M = 16$ following the setting in [16].

3.3. Stage I: Global Context Modeling

Generally, edges and boundaries in images are defined to be semantically meaningful. It is crucial to capture the abstract cues and the global context of the whole image. In the first stage, we explore the global contextual features on coarse-grained patches by a global transformer encoder $g_{E}$ and a global decoder $g_{D}$.
group as inputs. Then we reshape them to 3D features with

$$z_g = \{z^1_g, z^2_g, \ldots, z^{L_g}_g\} = G_E(\mathbf{s}_g),$$  \hspace{1cm} (4)

where $z^1_g, z^2_g, \ldots, z^{L_g}_g \in \mathbb{R}^{H \times W \times C}$ represent the outputs of successive blocks in $G_E$, and $L_g$ is the number of transformer blocks in $G_E$. In our experiments, we set $G_E$ to 24 following [16]. Next, the sequence of global context features $z_g$ are upsampled to high-resolution features by the global decoder $G_D$ for incorporation.

**BiMLA Decoder.** It is crucial to generate edge-aware pixel-level representations for detecting precise and thin edges. Thus, we expect to design a practical decoder that can encourage the transformer encoder to compute the edge-aware attentions and upsample the attentions in a learnable manner. Inspired by the multi-level feature aggregation in vision tasks [22, 35–37, 65, 72], we propose a novel Bi-directional Multi-Level Aggregation (BiMLA) decoder, as illustrated in Fig. 3, to achieve the goal.

In BiMLA, a bi-directional feature aggregation strategy is designed that includes a top-down path and a bottom-up path to boost the information flow in the transformer encoder. More specifically, we first uniformly divide $L_g$ transformer blocks into four groups, and take the embedding features $\{z^6_g, z^{12}_g, z^{18}_g, z^{24}_g\}$ from the last block of each group as inputs. Then we reshape them to 3D features with the size of $\frac{H}{16} \times \frac{W}{16} \times C$. For the top-down path, we attach the same design (one $1 \times 1$ convolutional layer and one $3 \times 3$ convolutional layer) to each reshaped feature and obtain four output features $t^{6}, t^{12}, t^{18}, t^{24}$, following the way of SETR-MLA [72]. Likewise, the bottom-up path starts from the lowest level (i.e., $z^6_g$) and gradually approaches the top level (i.e., $z^{24}_g$) by attaching one $3 \times 3$ convolutional layer on multi-level features, and finally produce another four output features $b^6, b^{12}, b^{18}, b^{24}$. Besides, unlike SETR-MLA [72] that upsamples the features via bilinear operation, our BiMLA passes each aggregated feature through a deconvolutional block, contains two deconvolutional layers with $4 \times 4$ kernels and $16 \times 16$ kernels, respectively. Each deconvolutional layer is followed by Batch Normalization (BN) and ReLU operations. The eight upsampled features from the bi-directional path are then concatenated into one tensor. Moreover, BiMLA uses an additional stack of convolutional layers to smooth the concatenated features. The stack consists of three $3 \times 3$ convolutional layers and one $1 \times 1$ convolutional layer with BN and ReLU. The process of BiMLA decoder is formulated as

$$f_g = G_D(\{z^6_g, z^{12}_g, z^{18}_g, z^{24}_g\}),$$  \hspace{1cm} (5)

where $f_g$ is the pixel-level global features, and $G_D$ represents the global BiMLA decoder. After obtaining the coarse-grained global context features, we will capture the fine-grained local context features in the next stage.

**3.4. Stage II: Local Refinement**

It is essential to explore fine-grained context features for pixel-level predictions, especially for edge detection. The ideal edge width is one pixel, while $16 \times 16$ patches are not conducive to extracting thin edges. Taking pixels as tokens sounds an intuitive remedy, however, it is practically infeasible due to heavy computational cost. Our solution is to use a non-overlapping sliding window to sample the image and then calculate the attentions within the sampled regions. The number of patches in the window is fixed, so the computational complexity is linearly related to the image size.

Thus motivated, we propose to capture the short-range fine-grained context features in Stage II, as shown at the bottom of Fig. 2. In particular, we perform the non-overlapping sliding window with a size of $\frac{H}{2} \times \frac{W}{2}$ on image $X \in \mathbb{R}^{H \times W \times 3}$, and the input image $X$ is decomposed into a sequence $\{X^1, X^2, X^3, X^4\}$. For each window, we split it into fine-grained patches of size $8 \times 8$ and compute the attentions by a shared local transformer encoder $R_E$. Then we concatenate the attentions of all windows to obtain $z_r = \{z_r^1, \ldots, z_r^{L_r}\} \in \mathbb{R}^{\frac{HW}{64}} \times C$. To further economize the computing resource, we set $L_r = 12$ that means the local transformer encoder consists of 12 transformer blocks.

![Figure 3. The detailed architecture of the BiMLA decoder consists of a top-down path and a bottom-up path.](image-url)
Similar to global BiMLA, we evenly select \( \{z_i^3, z_i^6, z_i^9, z_i^{12}\} \) from \( z \) and input them into the local BiMLA \( R_D \) to generate the local features with high-resolution,

\[
f_r = R_D(z_i^3, z_i^6, z_i^9, z_i^{12}),
\]

where \( f_r \) indicates the local features. Different from global BiMLA, we replace the 3×3 convolutional layer with the 1×1 convolutional layer in local BiMLA, so as to avoid artificial edges caused by the padding operation.

**Feature Fusion Module.** Finally, we incorporate the context cues from both levels by a Feature Fusion Module (FFM) and predict the edge maps by a local decision head. FFM takes the global context as the prior knowledge and modulates the local context, which produces the fusion features containing global context and fine-grained local details.

As shown in Fig. 2, FFM consists of a spatial feature transform block [60] and two 3×3 convolutional layers followed by BN and ReLU operations. The former is for modulating, and the latter is for smoothing. Then the fusion features are fed into the local decision head \( R_H \) to predict the edge maps \( E_r \),

\[
E_r = R_H(FFM(f_g, f_r)),
\]

where \( R_H \) is the local decision head that consists of a 1×1 convolutional layer and a sigmoid operation.

### 3.5. Network Training

To train the two-stage framework EDTER, we first optimize Stage I to generate global features that represent the whole image context information. Then, we fix the parameters of Stage I and train Stage II to generate edge maps.

**Loss Function.** We employ the loss function proposed in [65] for each edge map. Given an edge map \( E \) and the corresponding ground truth \( Y \), the loss is calculated as

\[
\ell(E, Y) = -\sum_{i,j} (Y_{i,j} \alpha \log(E_{i,j}) + (1 - Y_{i,j})(1 - \alpha) \log(1 - E_{i,j})),
\]

where \( E_{i,j} \) and \( Y_{i,j} \) are the \((i, j)\)th element of matrix \( E \) and \( Y \), respectively. Moreover, \( \alpha = |Y^-|/(|Y^-| + |Y^+|) \) indicates the percentage of negative pixel samples, where \( | \cdot | \) denotes the number of pixels. In practice, the annotations of BSDS500 [1] are labeled by multiple annotators. Inconsistent annotations lead to problematic convergence behavior [65]. Following [36], we first normalize multiple labels to an edge probability map with ranges [0, 1], and then use a threshold \( \eta \) to select pixels. The pixel is marked as a positive sample if the probability value is higher than \( \eta \); otherwise, it is indicated as a negative sample.

#### Training Stage I

For training Stage I, we first incorporate the global decision head on the global feature maps to generate the edge maps \( E_g \) by

\[
E_g = G_H(f_g),
\]

where \( G_H \) indicates the global decision head that consists of a 1×1 convolutional layer and a sigmoid layer. Moreover, we obtain multiple side outputs \( S_{1}^g, S_{2}^g, \ldots, S_{8}^g \) by performing the same design (a 4×4 deconvolutional layer and a 16×16 deconvolutional layer) to the intermediate features \( t^6, t^{12}, t^{18}, t^{24} \) and \( b^6, b^{12}, b^{18}, b^{24} \) extracted by the global BiMLA decoder, which progressively encode the encoder to emphasize edge-aware attentions.

Stage I is optimized by minimizing the losses between each edge map and the ground truth. The loss function of Stage I is formulated as

\[
L_g = L_g^E + \lambda L_g^{side} = \ell(E_g, Y) + \lambda \sum_{k=1}^{8} \ell(S_k^g, Y),
\]

where \( L_g^E \) is the loss for \( E_g \), \( L_g^{side} \) denotes side loss, and \( \lambda \) is the weight for balancing \( L_g^E \) and \( L_g^{side} \). In our experiments, we set \( \lambda \) to 0.4.

#### Training Stage II

After training Stage I, we fix the parameters of Stage I and move on to Stage II. Similar to the training of Stage I, we perform the same operation (a 4×4 deconvolutional layer and an 8×8 deconvolutional layer) on the intermediate features extracted from the local BiMLA decoder to generate the side outputs \( S_{1}^r, S_{2}^r, \ldots, S_{8}^r \). Finally, the loss function of Stage II is defined as

\[
L_r = L_r^E + \lambda L_r^{side} = \ell(E_r, Y) + \lambda \sum_{k=1}^{8} \ell(S_k^r, Y),
\]

where \( L_r^E \) and \( L_r^{side} \) are the losses for \( E_r \) and side outputs, respectively. We again set \( \lambda = 0.4 \).

### 4. Experiments

#### 4.1. Datasets

We conduct the experiments on three popular benchmarks: BSDS500 [1], NYUDv2 [49] and Multicue [42].

**BSDS500** [1] contains 500 RGB natural images, 200 for training, 100 for validation, and 200 for testing. Each image is manually annotated by five different subjects on average. Our model is trained on the training and validation sets and evaluated on the testing set. Similar to [22, 36, 65], we augment the dataset by rotating each image at 16 different angles and flipping the image at each angle. Moreover, most previous works [22, 36, 37, 62] use PASCAL VOC Context Dataset [17] as the additional training data, which provides full-scene segmentation annotations with
more than 400 classes, and consists of 10,103 images for training. The outside boundaries extracted from the segmentation annotations are beneficial to infer semantic and context cues in Stage I. Therefore, we first pre-train Stage I on PASCAL VOC Context Dataset [17] and then fine-tune it on BSDS500 [1]. The PASCAL VOC Context Dataset [17] is only used for training Stage I.

NYUDv2 [49] contains 1,449 pairs of aligned RGB and depth images, and it is split into 381 training, 414 validation, and 654 testing images. Following [36, 65], we combine the training and validation sets as the training data, and then augment them by rotating the images and annotations to 4 different angles, randomly flipping, and scaling.

Multicue [42] is composed of 100 challenging natural scenes captured by a binocular stereo camera. Each scene contains a left-view and a right-view short sequences. The last frame of left-view sequences from each scene is labeled with edges and boundaries. Following [22, 36, 65], we randomly select 80 images for training and the remaining 20 images for testing. We repeat the process three times and average the scores of three independent trials as the final results. The data augmentation follows [36, 65].

4.2. Implementation Details

We implement our EDTER using PyTorch [43]. We initialize the transformer blocks of our model using the pre-trained weights by ViT [16]. We set the threshold $\eta$ as 0.3 to select positive samples for BSDS500 and Multicue Edge, and 0.4 for Multicue Boundary. Each image has only one annotation in NYUDv2, thus no $\eta$ is needed. We use SGD optimizer with momentum=0.9 and weight decay=2e-4, and adopt a polynomial learning rate decay schedule [71] on all datasets. The initial learning rate is set as 1e-6 for BSDS500, NYUDv2 and Multicue Boundary, and 1e-7 for Multicue Edge. During training, we set the same iteration numbers for both stages. Specially, we train 80k iterations for BSDS500 and Multicue boundary, 40k for NYUDv2, and 4k for Multicue Edge. Each image is randomly cropped to 320×320 in training. Compared with BSDS500, the annotations of NYUDv2 is unitary, and the scale of Multicue is small, which quickly overfits of the model trained on them. Therefore, we set the batch size to 8 for BSDS500, and 4 for NYUDv2 and Multicue. All the experiments are conducted on a V100 GPU. The training of EDTER takes about 26.4 hours (15.1 for Stage I and 11.3 for Stage II). The inference runs at 2.2 fps on a V100. During Training, the GPU consumption of Stage I and Stage II are about 15GB and 14GB for 320×320 images respectively. Besides, EDTER brings 332.0G FLOPs in Stage I and 470.25G FLOps in Stage II.

During evaluation, we record three metrics for all datasets: fixed contour threshold (ODS), per-image best threshold (OIS), and average precision (AP). Moreover, a non-maximum suppression [6] is performed on the predicted edge maps before evaluation. Following previous works [36, 65], the localization tolerance controls the maximum allowed distance in matches between edge results and the ground truth, which is set to 0.0075 for BSDS500 and Multicue, and 0.011 for NYUDv2.

4.3. Ablation Study

Effectiveness of key components in EDTER. We first conduct experiments to verify the impact of key EDTER components: BiMLA and FFM. The quantitative results are summarized in Table 1. First, the performance of ODS, OIS, AP is largely improved (about 2.5%, 3%, 3%) by BiMLA compared with SETR-MLA [72] in both stages. The performance of Stage II significantly surpasses Stage I under either decoder. It illustrates that the two-stage strategy fuses more critical information for edge detection. Besides, we present the predicted edge maps by the SETR-MLA and BiMLA decoders shown in Fig. 4. With the BiMLA decoder, EDTER can accurately detect edges in some local areas (red bounding boxes) and produce less noisy edges. To verify the effectiveness of FFM, we remove the FFM and directly concatenate the feature maps from two stages to construct a variant of EDTER. Without using FFM (row 5), the scores drop by 0.4%, 0.6%, and 1.3% in ODS, OIS, and AP, respectively.

Effectiveness of stages and patch size. We run ablation experiments to verify the effectiveness of the two-stage strategy. The comparative results are presented in Table 2. Compared with Stage I (row 1), we add the second stage and set the patch size to 8×8 (row 2), which obtains the performance gain by 0.7%, 0.6%, 1.3% in ODS, IS, and AP, respectively. Moreover, as visualized in Fig. 5, the predicted edges of Stage II are more clear and crisp in some local de-
4.4. Comparison with State-of-the-arts

On BSDS500. We compare our model with traditional detectors including Canny [6], Felz-Hutt [18], gPb-owt-ucm [2], SCG [64], Sketch Tokens [34], PMI [25], SE [15], OEF [21] and MES [50], and deep-learning-based detectors including DeepEdge [3], CSCNN [24], DeepContour [48], HFL [4], HED [65], Deep Boundary [29], CEDN [67], RDS [37], COB [40], DCD [33], AMH-Net [66], RCF [30], CED [33], DeepEdge [29], HED [65], HFL [4], HED [65], AMH-Net [66], RCF [36], LPCB [12], BDCN [22], DexiNed [52], DSCD [11] and PiDiNet [53]. The best results of all the methods are taken from their publications.

Quantitative results are shown in Table 3, and Fig. 6 shows Precision-Recall curves of all methods. By training on the trainval set of BSDS500, our method achieves the F-measure ODS of 0.824 with single-scale testing and obtains 0.840 with multi-scale inputs, which already outperforms most edge detectors. With the extra training data and multi-scale testing (following the settings of RCF, CED, BDCN, etc.), our method achieves 84.8% (ODS), 86.5% (OIS) 90.3% (AP), which is superior to all the state-of-the-art edge detectors. Some qualitative results are shown in Fig. 7. We observe that the proposed EDTER shows a clear advantage in prediction quality, both crisp and accurate.
Table 4. Quantitative comparisons on NYUDv2 [49]. All results are computed with a single scale input.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pub. Year</th>
<th>ODS</th>
<th>OIS</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gPb-ucm [1]</td>
<td>PAMI’11</td>
<td>0.632</td>
<td>0.661</td>
<td>0.562</td>
</tr>
<tr>
<td>Silberman et al. [49]</td>
<td>ECCV’12</td>
<td>0.658</td>
<td>0.661</td>
<td>-</td>
</tr>
<tr>
<td>gPb+NG [19]</td>
<td>CVPR’13</td>
<td>0.687</td>
<td>0.716</td>
<td>0.629</td>
</tr>
<tr>
<td>SE [15]</td>
<td>PAMI’14</td>
<td>0.695</td>
<td>0.708</td>
<td>0.679</td>
</tr>
<tr>
<td>SE+NG+ [20]</td>
<td>ECCV’14</td>
<td>0.706</td>
<td>0.734</td>
<td>0.738</td>
</tr>
<tr>
<td>OEF [21]</td>
<td>CVPR’15</td>
<td>0.651</td>
<td>0.667</td>
<td>-</td>
</tr>
<tr>
<td>SemiContour [69]</td>
<td>CVPR’16</td>
<td>0.680</td>
<td>0.700</td>
<td>0.690</td>
</tr>
<tr>
<td>CNN-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HED [65]</td>
<td>ICCV’15</td>
<td>0.720</td>
<td>0.734</td>
<td>0.734</td>
</tr>
<tr>
<td>RCF [36]</td>
<td>CVPR’17</td>
<td>0.729</td>
<td>0.742</td>
<td>-</td>
</tr>
<tr>
<td>AMH-Net [66]</td>
<td>NeurIPS’17</td>
<td>0.744</td>
<td>0.758</td>
<td>0.765</td>
</tr>
<tr>
<td>LPCB [12]</td>
<td>ECCV’18</td>
<td>0.739</td>
<td>0.754</td>
<td>-</td>
</tr>
<tr>
<td>BDCN [22]</td>
<td>CVPR’19</td>
<td>0.748</td>
<td>0.763</td>
<td>0.770</td>
</tr>
<tr>
<td>PiDiNet [53]</td>
<td>ICCV’21</td>
<td>0.743</td>
<td>0.747</td>
<td>-</td>
</tr>
<tr>
<td>EDTER (Ours)</td>
<td></td>
<td>0.774</td>
<td>0.789</td>
<td>0.797</td>
</tr>
</tbody>
</table>

Table 5. Comparisons on Multicue [42]. All results are computed with a single scale input.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pub. Year</th>
<th>ODS</th>
<th>OIS</th>
<th>AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human [42]</td>
<td>VR’16</td>
<td>0.750</td>
<td>0.024</td>
<td>-</td>
</tr>
<tr>
<td>Multicue [42]</td>
<td>VR’16</td>
<td>0.830</td>
<td>0.002</td>
<td>-</td>
</tr>
<tr>
<td>HED [65]</td>
<td>ICCV’15</td>
<td>0.851</td>
<td>0.014</td>
<td>0.864</td>
</tr>
<tr>
<td>RCF [36]</td>
<td>CVPR’17</td>
<td>0.857</td>
<td>0.004</td>
<td>0.862</td>
</tr>
<tr>
<td>BDCN [22]</td>
<td>CVPR’19</td>
<td>0.891</td>
<td>0.001</td>
<td>0.898</td>
</tr>
<tr>
<td>DSCD [11]</td>
<td>ACMMM’20</td>
<td>0.871</td>
<td>0.007</td>
<td>0.876</td>
</tr>
<tr>
<td>PiDiNet [53]</td>
<td>ICCV’21</td>
<td>0.855</td>
<td>0.007</td>
<td>0.860</td>
</tr>
<tr>
<td>EDTER (Ours)</td>
<td></td>
<td>0.894</td>
<td>0.005</td>
<td>0.900</td>
</tr>
<tr>
<td>Boundary</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human [42]</td>
<td>VR’16</td>
<td>0.760</td>
<td>0.017</td>
<td>-</td>
</tr>
<tr>
<td>Multicue [42]</td>
<td>VR’16</td>
<td>0.720</td>
<td>0.014</td>
<td>-</td>
</tr>
<tr>
<td>HED [65]</td>
<td>ICCV’15</td>
<td>0.814</td>
<td>0.011</td>
<td>0.822</td>
</tr>
<tr>
<td>RCF [36]</td>
<td>CVPR’17</td>
<td>0.817</td>
<td>0.004</td>
<td>0.825</td>
</tr>
<tr>
<td>BDCN [22]</td>
<td>CVPR’19</td>
<td>0.836</td>
<td>0.001</td>
<td>0.846</td>
</tr>
<tr>
<td>DSCD [11]</td>
<td>ACMMM’20</td>
<td>0.828</td>
<td>0.003</td>
<td>0.835</td>
</tr>
<tr>
<td>PiDiNet [53]</td>
<td>ICCV’21</td>
<td>0.818</td>
<td>0.003</td>
<td>0.830</td>
</tr>
<tr>
<td>EDTER (Ours)</td>
<td></td>
<td>0.861</td>
<td>0.003</td>
<td>0.870</td>
</tr>
</tbody>
</table>

5. Conclusion and Limitation

In this paper, we propose a novel two-stage edge detection framework, namely EDTER. By introducing the vision transformer, EDTER captures both coarse-grained global context and fine-grained local context in two stages. Moreover, it employs a novel Bi-directional Multi-Level Aggregation (BiMLA) decoder to explore high-resolution representations. Besides, a Feature Fusion Module (FFM) incorporates global and local contexts to predict the edge results. Experimental results illustrate that EDTER yields competitive results in comparison with state-of-the-arts.

Limitation. The width of the edges extracted by EDTER occupies multiple pixels, which still has a gap with the ideal edge width. Without any post-processing, generating clear and thin edges is still a future direction to explore.

Acknowledgements. This work is supported by Beijing Natural Science Foundation (M22022, L211015), Fundamental Research Funds for the Central Universities (2019JBZ104), and National Natural Science Foundation of China (61906013, 62106017).
References


[65] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In Int. Conf. Comput. Vis., pages 1395–1403, 2015. 1, 2, 4, 5, 6, 7, 8


