Boundary Attention: Learning to Find Faint Boundaries at Any Resolution

Mia Gaia Polansky[1,2] Charles Herrmann[1] Junhwa Hur[1]
Deqing Sun[1] Dor Verbin[1] Todd Zickler[1]

[1]Google [2]Harvard University

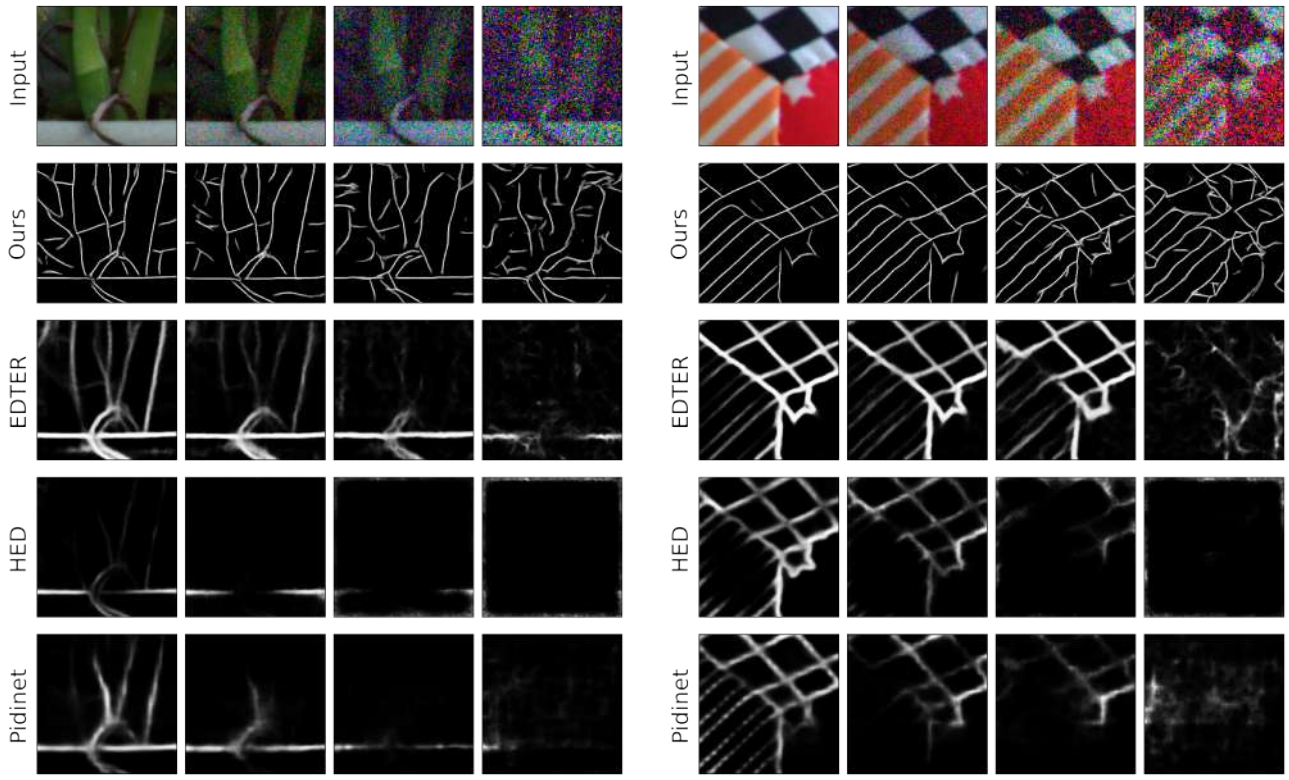

Figure 1. Despite being trained on synthetic data, our method outperforms state-of-the-art EDTER [28], HED [35], and Pidinet [31] at multiple levels of real sensor noise present in photographs from ELD [34]. Our method produces accurate and well-defined boundaries despite significant amounts of noise.

Abstract

We present a differentiable model that explicitly models boundaries—including contours, corners and junctions—using a new mechanism that we call boundary attention. We show that our model provides accurate results even when the boundary signal is very weak or is swamped by noise. Compared to previous classical methods for finding faint boundaries, our model has the advantages of being differentiable; being scalable to larger images; and automatically adapting to an appropriate level of geometric detail in each part of an image. Compared to previous deep methods for finding boundaries via end-to-end training, it has the advantages of providing sub-pixel precision, being more resilient to noise, and being able to process any image at its native resolution and aspect ratio.

1. Introduction

Converting a precise contour that is defined mathematically in continuous 2D space to its discrete representation in pixel-space is a common task in computer graphics, and

there are established techniques for rasterization [12, 27], anti-aliasing [6, 16] and so on. However, the inverse problem in computer vision of robustly inferring precise and unrasterized descriptions of contours from discrete images remains an open challenge. Earlier work in the field inferred unrasterized representations of edges [4, 8, 13, 14, 22, 25]—and sometimes corners and junctions too—and it explored algorithms for connecting them into boundaries via edge-linking and MRFs. But since the dawn of deep learning, boundaries have almost exclusively been modeled using discrete, rasterized maps. Moreover, it is not uncommon to rely heavily on supervision from human annotations, and even modern datasets can suffer from annotation errors on the order of pixels, quite far from sub-pixel accuracy.

Taking inspiration from early computer vision work, we present a model for inferring unrasterized boundaries that can be learned instead of designed. We show that this model is able to benefit from the power of deep learning while also realizing many of the advantages—such as robustness to noise, sub-pixel precision, and adaptability between signal-types—that have long been the potential strengths of classical bottom-up techniques.

The core of our model is a mechanism we call boundary attention. It is a boundary-aware local attention operation that, when applied densely and repeatedly, progressively refines a field of variables that specifies the local boundaries surrounding every pixel. The model's output is a field of overlapping geometric primitives that can be used in a variety of ways, including to produce an unsigned distance function for the image's boundaries, a boundary-aware smoothing of its channel-values, and a field of soft local attention maps that relate every pixel to its neighbors.

To evaluate our model, we consider the problem of finding boundaries in images corrupted by extreme amounts of noise [23, 24, 33], where the images can be of any size and resolution, and where we know little about the noise and the objects or shapes we are looking for. We choose this problem because it relies entirely on having a strong model for the basic topological and geometric properties of boundaries, namely that they are piecewise smooth curves which connect at corners or junctions and divide regions of homogeneity.

We show that our model has several advantages. All of its components are local and invariant to discrete spatial shifts, so it can be trained on small-sized images and then deployed on much larger and differently-shaped ones. It is also very compact, comprising only 207k parameters, and it runs several times faster than many alternatives. We also find that it can be trained to a useful state with very simple synthetic data, made up of random circles and triangles that are uniformly colored and then corrupted by noise. Despite this simplicity, we find that the model can generalize to real images surprisingly well.

Our main contributions can be summarized as follows:

1. We propose a novel network design that explicitly models boundaries using a boundary attention mechanism that can be used in any deep learning framework.
2. We demonstrate that our model based on boundary attention is more effective than current state-of-the-art methods at finding boundaries at high levels of sensor noise (Fig. 1) and can achieve sub-pixel accuracy, even in the presence of noise.

2. Related Work

It is possible to process the outputs of certain filters to localize edges with sub-pixel precision (*e.g.*, [4, 8, 13, 14, 22, 25]). But this approach struggles near corners and junctions because the edge filters are not derived for these, and accuracy breaks down. Recently, the field of junctions [33] showed improvements by expanding the library of local, unrasterized geometric primitives from classical edge-based primitives to a larger family of "generalized junctions". This allows unifying a variety of bottom-up cues that had often previously been handled separately, such as geometrically-consistent and polarity-consistent edge elements being linked into contours [4, 18, 23, 24]; contours that approach one another being joined at corners and junctions [15, 21, 36]; homogeneous regions that look the same being together; and those that look different being apart [2, 7, 11]. The field of junctions showed that putting all of these cues together leads to a dramatic improved in noise resilience, and it also allows analyzing the recovered boundaries explicitly into its component junctions, edges and corners. In this paper, we leverage all of these benefits, and we do this more efficiently and with an adaptive patch size. Additionally, our model is differentiable, meaning it can be used as a component in larger systems where accurate boundaries are necessary.

Our work is complementary to recent approaches that leverage large-scale training to perform edge detection or segmentation by internalizing the patterns in a training set. Recent examples include Segment Anything [19] and combinations of strong self-supervised features with clustering and CRFs [3, 5, 20]. These large-scale approaches rely on large training sets and high-capacity architectures that can internalize high-level cues like object and scene familiarity as well as low-level cues like spatial coherence. Our approach is different because it relies exclusively on low-level cues, has much less capacity, and uses much less training data. Despite this difference, we find that our model's inherent preference for contour-like boundaries allows it to achieve comparable (or better) results, in particular when noise is high. This suggests that future work could benefit from incorporating some of our architectural elements into larger and higher-capacity systems, in particular to improve interpretability, efficiency, generalization or task-transfer.

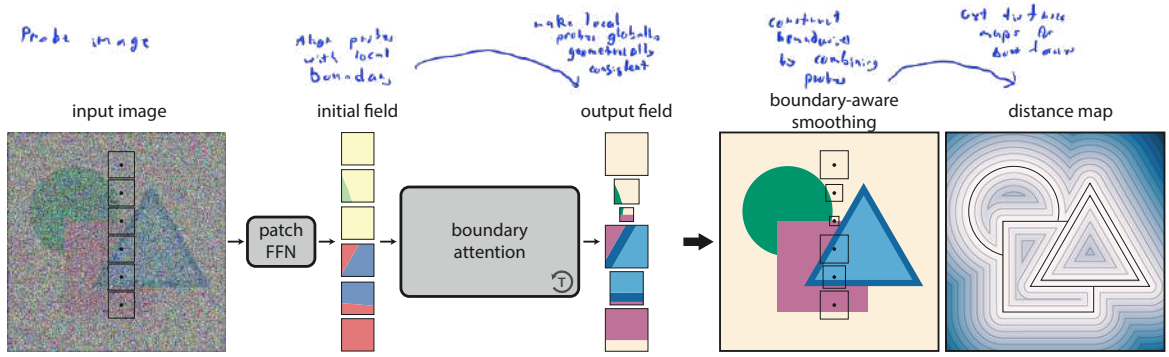

Figure 2. Our model uses neighborhood attention with dense, stride-1 tokens (but we depict non-overlapping ones here for visual clarity). The entire model is invariant to discrete spatial shifts and so applies at any resolution. Each token encodes an adaptively-sized geometric primitive that represents the unrasterized local boundaries surrounding a pixel, and the tokens evolve through boundary attention to be geometrically consistent. The output is an overlapping field of primitives that directly implies a boundary-aware smoothing of the input image and an unsigned distance map to the image boundaries.

3. Representation

Our system is depicted in Figure 2. It uses neighborhood cross-attention, a patch-wise variant of cross-attention, with pixel-resolution tokens. Critically, each pixel's token is explicitly required to encode a tuple of values representing a geometric primitive that specifies the geometric structure of the local boundaries surrounding the pixel. Because the tokens are forced through this geometric bottleneck, we refer to our model's core mechanism as *Boundary Attention*.

We instantiate our bottleneck using a learned linear mapping from the token dimension to a predefined lower-dimensional space of unrasterized boundary patterns that we call *junction space*. Our choice for this space is inspired by the Field of Junctions [33], but we use a modified parameterization (described below) that is differentiable across its entire domain. Junction space has the benefit of specifying local boundary patterns without rasterization and thus with unlimited spatial precision. As depicted in Figure 3 and described in [33], it also has the benefit of including a large family of local boundary patterns, including uniformity (*i.e.*, absence of boundaries), edges, bars, corners, T-junctions and Y-junctions.

The next section introduces our parameterization of junction space and some associated operators. Then Section 4 describes the architecture that we use to analyze an image into its field of junction values.

3.1. Boundary Primitives

Throughout this paper we use parentheses (x) for continuous signals defined on the 2D image plane $[0, W] \times [0, H]$ and square brackets $[n]$ for discrete signals defined on the pixel grid. We use $c[n]$ for the coordinates of the nth pixel's center.

Denote the K-channel input image by $\{\mathbf{f}[n]\}$, where $\mathbf{f}[n] \in \mathbb{R}^K$ is the vector image value at the discrete pixel grid index n. Our approach is to analyze the image into a field of dense, stride-1 overlapping local patches, each having a square support $\Omega_n(x)$ centered at the nth pixel.

There are many ways to partition a local region $\Omega_n(x)$, and one can define parametric families of such partitions. For example the set of oriented lines provides a two-parameter family of partitions, with each member of the family separating the region into points that lie on one side of a line or the other. This family of partitions would be appropriate for describing edges. Here we define a larger family of partitions that encompasses a greater variety of local boundary structures.

As depicted in the right of Figure 3, our partitions are parameterized by $\mathbf{g} \in \mathbb{R}^2 \times \mathbb{S}^1 \times \triangle^2$, where $\mathbb{S}^1$ is the unit circle and $\triangle^2$ is the standard 2-simplex. We use the notation $\mathbf{g} = (\boldsymbol{u}, \theta, \boldsymbol{\omega})$, where $\boldsymbol{u} = (u, v) \in \mathbb{R}^2$ is the *vertex*, $\theta \in \mathbb{S}^1$ is the *orientation*, and $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)$ are barycentric coordinates (defined up to scale) for the three relative angles, ordered clockwise starting from θ. Our convention is to express the vertex coordinates relative to the center of region $\Omega_n(x)$, and we denote that the vertex is free to move outside of this region. We also note that up to two angles ω_j can be zero. This all makes it possible to represent a variety of partition types, including edges, bars, corners, 3-junctions and uniformity (*i.e.*, trivial or singleton partitions).

Fixing a value for $\mathbf{g}$ induces three binary-valued spatial support functions:

$$s_{nj}(x; \mathbf{g}) \in \{0, 1\}, \; j = 1, 2, 3, \quad (1)$$

that evaluate to 1 for points x that are in $\Omega_n(x)$ and in the jth wedge defined by $\mathbf{g}$; and 0 otherwise. It also induces an unsigned distance function:

$$d_n(x; \mathbf{g}) \geq 0, \quad (2)$$

that represents the Euclidean distance from point x to the nearest point in the boundary set defined by $\mathbf{g}$.

Defining these as continuous functions allows specifying the local boundary structure with unlimited spatial resolution. The right of Figure 3 uses three colors to visualise the

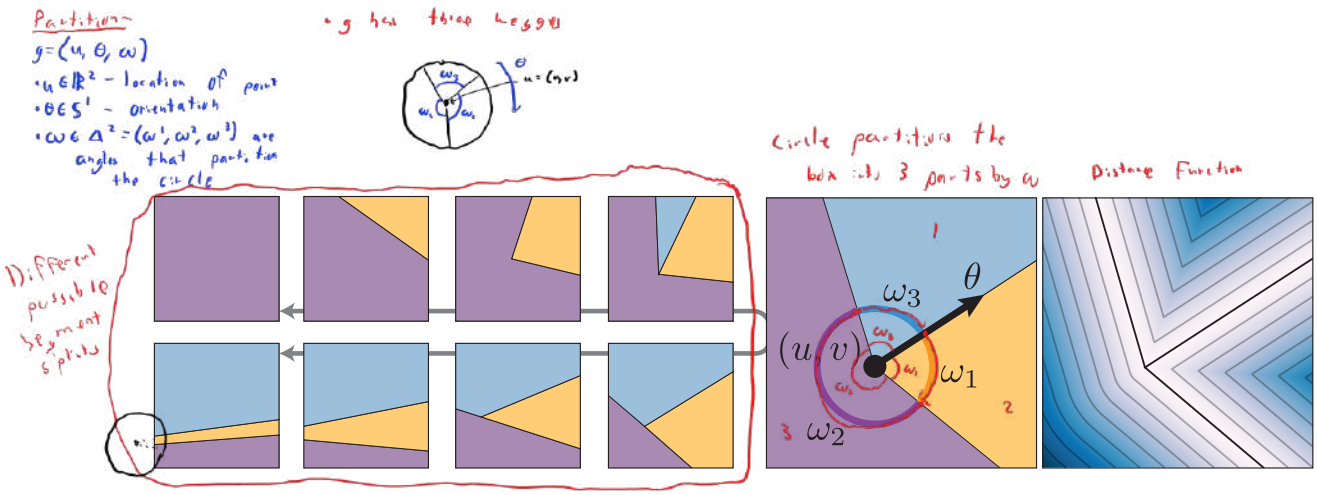

Figure 3. Samples on a smooth trajectory through our space of geometric primitives ("junction space"), with one sample enlarged and accompanied by a visualization of its distance map on the right. Each sample comprises a vertex position (u, v), orientation θ, and angles $(\omega_1, \omega_2, \omega_3)$ that partition the circle. The space includes all types of locally-uniform regions, edges, bars, corners, Y-junctions and T-junctions.

wedge supports of a junction $\mathbf{g}$, and it shows a quantized visualization of the associated distance function.

Instead of fixing the size of every patch in the image to a hand-selected value that is the same throughout, we would like the size of every region Ω_n to adapt to the local geometry. We do this by equipping each patch with a parameterized local windowing function $w_n(x; \mathbf{p}) \in [0, 1]$. Specifically, we parameterize them as convex combinations of W square window function coefficients $\mathbf{p} \in \mathcal{P} = \triangle^{W-1}$. That is,

$$w_n(x; \mathbf{p}) = \sum_{i=1}^{W=3} p_i \mathbf{1}[\|x - c[n]\|_\infty \le D_i], \quad (3)$$

where $\|\cdot\|_\infty$ is the ℓ^∞-norm, and $\mathbf{1}[\cdot]$ is the indicator function that returns 1 if the argument true; and 0 otherwise. In our experiments we use $W = 3$ and diameters $\mathbf{D} = (3, 9, 17)$.

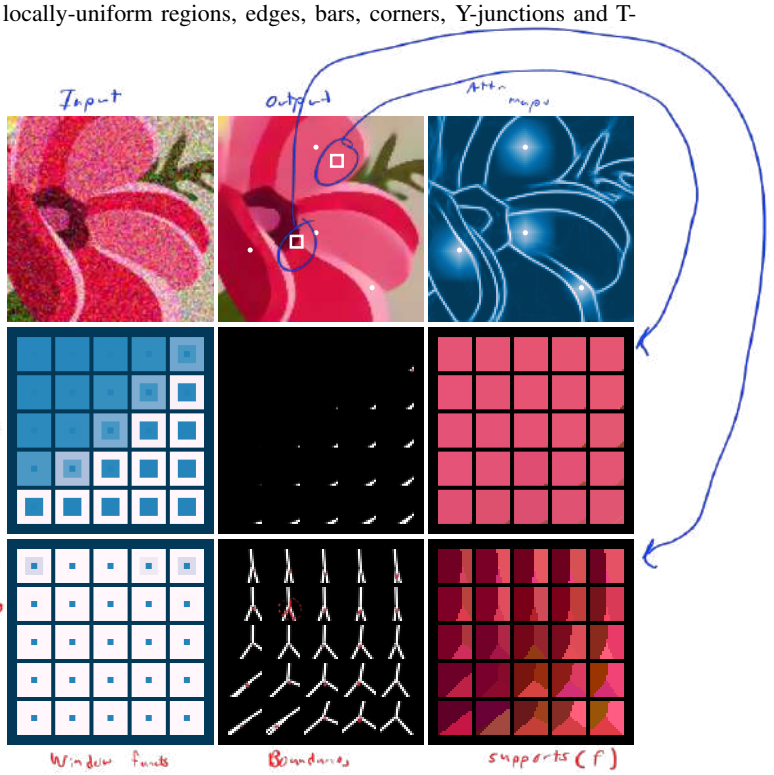

Figure 4. Visualizing our model's output. *Top row, left to right*: (*i*) input image; (*ii*) boundary-aware smoothing; and (*iii*) boundary map with attention maps at four query points. The two bottom rows show regions Ω_n unfolded from within the two white windows indicated above. *Bottom rows, left to right*: (*i*) window functions; (*ii*) boundaries; and (*iii*) support functions colored by their gathered wedge features. See text for details.

3.2. Gather and Slice Operators

Our network operates by refining the field $\{(\mathbf{g}^t[n], \mathbf{p}^t[n])\}$ over a fixed sequence of steps $t = 1, ..., T$. It uses two operators that we define here and depict in the right of Figure 5. The first operator is a patch-wise *gather* operator, in which each wedge of each region computes the weighted average of the image values it contains (recall that $c[n]$ are nth pixel's coordinates):

$$\mathbf{f}_{kj} = \frac{\sum_n \mathbf{f}[n] w_k(c[n]; \mathbf{p}[n]) s_{kj}(c[n]; \mathbf{g}[n])}{\sum_n w_k(c[n]; \mathbf{p}[n]) s_{kj}(c[n]; \mathbf{g}[n])}. \quad (4)$$

The second operation is a pixel-wise *slice* operation, where each pixel computes the means and variances, over all regions that contain it, of the per-region distance maps $d_n(x; \mathbf{g}[n])$ and gathered wedge features $\mathbf{f}_{kj}$. The expressions for the means are:

$$\bar{d}[n] = \frac{\sum_k w_k(c[n]; \mathbf{p}[k]) d_k(c[n]; \mathbf{g}[k])}{\sum_k w_k(c[n]; \mathbf{p}[k])}, \quad (5)$$

$$\bar{\mathbf{f}}[n] = \frac{\sum_k w_k(c[n]; \mathbf{p}[k]) \sum_j \mathbf{f}_{kj} s_{kj}(c[n]; \mathbf{g}[k])}{\sum_k w_k(c[n]; \mathbf{p}[k]) \sum_j s_{kj}(c[n]; \mathbf{g}[k])}. \quad (6)$$

Note that the only contributions to the sums over patches k are from patches that contain $c[n]$, *i.e.*, the sums are over $\{k \mid \Omega_k \ni c[n]\}$. The expressions for the pixel-wise dis-

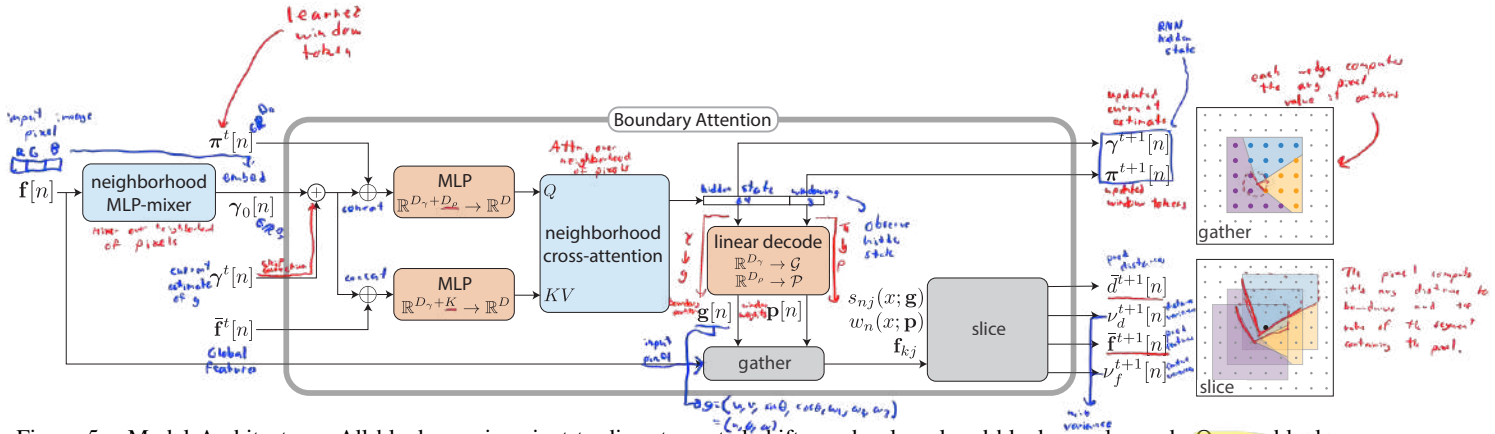

Figure 5. Model Architecture. All blocks are invariant to discrete spatial shifts, and only colored blocks are learned. Orange blocks operate at individual locations n, while blue ones operate on small spatial neighborhoods. Symbol $\oplus$ is concatenation, and gather and slice operators (Eqs. 4–6) are depicted at right. The first iteration uses $\boldsymbol{\gamma}^0[n] = \boldsymbol{\gamma}_0[n]$, $\bar{\mathbf{f}}^0[n] = \mathbf{f}[n]$, and $\boldsymbol{\pi}^0[n] = \boldsymbol{\pi}_o$ with $\boldsymbol{\pi}_o$ learned across the training set. Boundary attention repeats $T = 8$ times, with one set of weights for the first four iterations and another for the last four iterations. There are 207k trainable parameters in total.

tance map variance $\nu_d[n]$, and for the feature variance $\nu_f[n]$, which is computed across patches containing n and across their K channels, are defined accordingly and included in the supplement.

3.3. Visualizing Output

At our network's output, we expect the shapes of junction boundaries in overlapping regions to agree, so that the variances $\nu_d[n], \nu_f[n]$ are small at every pixel. Then, the fields of means $\{\bar{d}[n]\}, \{\bar{\mathbf{f}}[n]\}$ can be interpreted, respectively, as a global unsigned distance map for the image boundaries and a boundary-aware smoothing of its input channel values. Figure 4 shows an example, where we visualize the zero-set of the global unsigned distance map—we call this the global boundary map—by applying the nonlinearity:

$$\bar{b}_\eta[n] = \left(1 + (\bar{d}[n]/\eta)^2\right)^{-1}, \tag{7}$$

setting $\eta = 0.7$.

The output contains much more information than these fields of means. For any query pixel n, we can probe the wedge supports $\{s_{kj}(\cdot; \mathbf{g}[k])\}$ and windowing functions $\{w_k(\cdot, \mathbf{p}[k])\}$ that contain it, thereby obtaining a spatial attention map $a_n(x)$ that surrounds the query pixel. This is the boundary-aware spatial kernel that turns a neighborhood of input features $\{\mathbf{f}[\cdot]\}$ into the value $\bar{\mathbf{f}}[n]$:

$$\bar{\mathbf{f}}[n] = \sum_k a_n(c[k])\,\bar{\mathbf{f}}[k]. \tag{8}$$

The expression for $a_n(x)$ follows from inserting Equation 4 into 6, and its maximum diameter is twice that of $\Omega(x)$. Some examples are shown in the top-right of Figure 4.

Additionally, as shown in the bottom two rows of Figure 4, we can unfold any portion of the output field into the overlapping regions it contains. The figure shows unfolded regions $\{\Omega_k(x)\}$ within two windows of the output field. For each unfolded set we visualize: the windowing functions $w_k(x; \mathbf{p})$; the regional boundaries $b_k(x; \mathbf{g})$ obtained by applying the nonlinearity in Equation 7 to the local distance functions $d_k(x; \mathbf{g})$; and the supports $s_{kj}(x; \mathbf{g})$ colored according to the wedge features $\mathbf{f}_{kj}$ that they gather from the input image. In the top example, the neighborhood is homogeneous, so the windowing functions (via $\mathbf{p}[n]$) have large supports, and there are few boundaries (via $\mathbf{g}[n]$). In the bottom example there is fine-scale geometry. The windowing functions are narrow and the primitives agree on a Y-junction.

4. Network Architecture

We design our network to iteratively refine the fields $\{(\mathbf{g}^t[n], \mathbf{p}^t[n])\}$. We do so by embedding each field element using a higher dimensional representation, $\boldsymbol{\gamma}^t[n] \in \mathbb{R}^{D_\gamma}$ and $\boldsymbol{\pi}^t[n] \in \mathbb{R}^{D_\pi}$ respectively, which can be updated via dot-product attention. In practice, we use $D_\gamma = 64$ and $D_\pi = 8$, which provides the network with enough capacity to learn meaningful hidden states. We learn simple linear mappings $\boldsymbol{\gamma} \mapsto \mathbf{g}$ and $\boldsymbol{\pi} \mapsto \mathbf{p}$ which are used for the gather and slice operations. Importantly, all of our network's elements are invariant to discrete spatial shifts of the image, operating either on individual locations n or on small neighborhoods of locations. See Figure 5 and additional details in the supplement.

Given an input image, the network first applies a neighborhood MLP-mixer, which is a modified variation of MLP-Mixer [32] where we replace the linear operations with convolutions of kernel size 3. The other change is that we map the input pixels to the hidden state size with a pixel-wise linear mapping rather than taking patches of the input. This block, which we denote the "neighborhood MLP-mixer" transforms the input into an initial hidden state, which is then processed through eight iterations of our boundary attention to further refine the hidden state.

The eight iterations of refinement are broken into two Boundary Attention Blocks with independent weights. To process our input, we first add in a linear projection of the initial hidden state. This is essentially a skip connection that allows our network to retain information from the input pixels at later stages of processing. Next, we copy our hidden state into two identical pieces. We concatenate a dimension 8 learned windowing token to one of the copies and the current estimate of the smoothed global features to the other. We then do neighborhood cross-attention: each pixel in the first copy does two iterations of cross attention with a size 11 patch of the second copy. We add a learned 11×11 positional encoding to the patch, which allows our network to access relative positioning, even if global position cues are absent. We follow each self attention layer with a small MLP.

To transform our output or intermediary hidden state into junction space and render output images, we use a simple linear mapping. We separate the windowing token (the last 8 dimensions) from the hidden state (the first 64 dimensions) and project each through a linear layer. We map the hidden states to 7 numbers that represent $\mathbf{g} = (\boldsymbol{u}, \sin(\theta), \cos(\theta), \boldsymbol{\omega})$. These serve as the inputs to our gather and slice operators.

Overall, our network has $2.07 \cdot 10^5$ learnable parameters, making it orders of magnitude smaller than most learned boundary detectors. For contrast, EDTER [28] has 109 million parameters for its full model.

4.1. Training

We train our network in two stages: we begin by training the neighborhood MLP-mixer with the first Boundary Attention Block, and then we add the second block and retrain end-to-end. We apply our loss to the two final iterations of our network, so for the first stage of training we apply a loss to iterations 3 and 4, and for end-to-end refinement we apply a loss to iterations 7 and 8. We weigh the final loss three times higher than the second to last loss, which encourages the network to allocate capacity to producing high quality outputs, while ensuring that gradient information is shared across the network.

We train our method using a combination of four global losses applied to global (*i.e.* averaged) fields, and two patch-wise losses applied to individual patches. The first two losses are supervision losses penalizing mismatches between our network's predictions and the ground truth feature and boundary maps:

$$\mathcal{L}_f = \sum_n \alpha[n] \|\bar{\mathbf{f}}[n] - \mathbf{f}_{\text{GT}}[n]\|^2, \tag{9}$$

$$\mathcal{L}_d = \sum_n \alpha[n] \left(\bar{d}[n] - d_{\text{GT}}[n]\right)^2, \tag{10}$$

where $\mathbf{f}_{\text{GT}}$ and d_{GT} are the ground truth features and distance maps, respectively, and $\alpha[n]$ is a pixel importance function defined as:

$$\alpha[n] = e^{-\beta \cdot (d_{\text{GT}}[n] + \delta)} + C, \tag{11}$$

with β and C controlling how much weight to give pixels near boundaries. We set $\beta = 0.1$, $\delta = 1$, and increase C throughout training to give more weight to non-boundary locations. We also tested a more involved pixel importance mask that gave extra weight to visible vertices and intersections, the details for which can be found in the supplement. Note that using noiseless feature maps for supervision in Equation 9 encourages the windowing functions to be large in smooth regions.

On top of the two supervision losses we apply two consistency losses from [33], that minimize the per-pixel variances $\nu_f[n]$ and $\nu_d[n]$. Similar to the supervision losses, we weigh those by α from Equation 11. These consistency losses encourage the junction shapes $\mathbf{g}$ in overlapping regions to agree. Minimizing $\nu_f[n]$ also encourages windowing functions to be large, because that increases the gather area, which in turn reduces noise in wedge features $\mathbf{f}_{nj}$ that are sliced to compute the variance $\nu_f[n]$ at each n.

Finally, we use two patch-wise losses to encourage individual feature and distance patches to agree with the supervisory ones:

$$\ell_f = \sum_k \chi[k] \sum_{n \in \Omega_k} \alpha[n] \|\bar{\mathbf{f}}[n] - \mathbf{f}_{\text{GT}}[n]\|^2, \tag{12}$$

$$\ell_d = \sum_k \chi[k] \sum_{n \in \Omega_k} \alpha[n] \left(\bar{d}[n] - d_{\text{GT}}[n]\right)^2, \tag{13}$$

where $\chi[k]$ is a patch importance function defined as:

$$\chi[k] = \left(\sum_{n \in \Omega_k} \left(d_{\text{GT}}[n] + \delta'\right) \right)^{-1}, \tag{14}$$

and we set $\delta' = 1$. These per-patch losses gives the network a more direct signal on how to adjust its weights than purely global losses, which average over multiple patches.

5. Experiments

Implementation details. We train our model on noisy synthetic data of colorful combinations of triangles and circles. We render 240×320 images containing 15 to 20 shapes each, but use 125×125 crops for training. To those crops we add Gaussian and/or Perlin noise [26], and with probability 0.1 we average over the color channels to produce grayscale inputs. Our dataset contains 10^5 images, 90% of which are used for training, and the rest for testing. For training and optimization details, refer to the supplement.

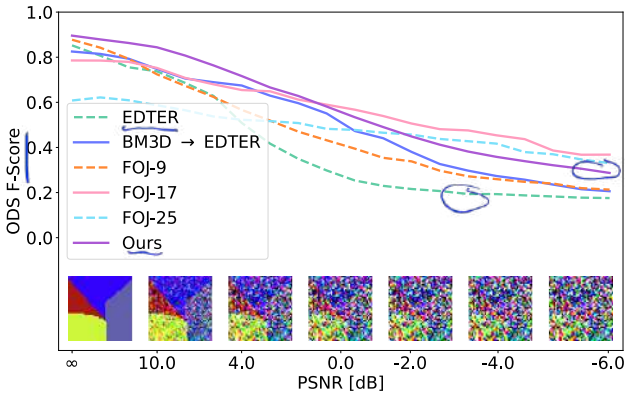

Figure 6. The ODS F-score of our method at different noise levels, compared with the Field of Junctions (FOJ) [33] run using patch sizes 9×9, 17×17, and 25×25, and EDTER [28], with and without preprocessing by BM3D [9]. The bottom insets show example patches at representative PSNR values.

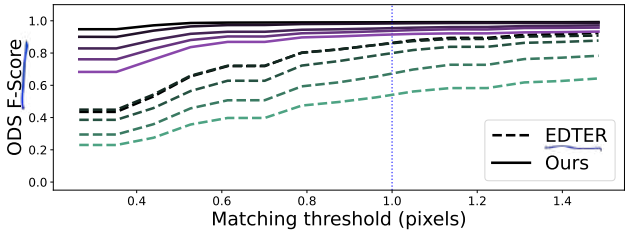

Figure 7. Our network can reliably locate boundaries with sub-pixel precision, due in part to our parametric representation for boundaries. We calculate the ODS F-Score of synthetic images as we increase the pixel matching distance threshold (how close a pixel must be to a ground truth pixel to be counted) from 0 to 1.5 and vary the PSNR of the images from ∞ to 8.0. Each line represents the performance at a noise level, with lighter-colored lines corresponding to noisier inputs. X-axis values below 1.0, denoted by the vertical dotted line, represent sub-pixel precision.

Performance w.r.t. noise levels. Figure 6 shows the comparison of our method and baseline approaches, the Field of Junctions [33] and EDTER [28] under different noise levels. We test the Field of Junctions with different patch sizes (9×9, 17×17, and 25×25) and EDTER with/without preprocessing using optimally-tuned denoising using BM3D [9]. The tuneable parameters for Field of Junctions were chosen to maximize its performance on noisy images with 17×17 patches. Our method outperforms all baselines at lower noise levels and is competitive with the Field of Junctions at higher noise levels, while being orders of magnitude faster than it (see Table 1).

Sub-pixel precision. To measure sub-pixel precision, we render 500×500 images containing pairs of overlapping circles and triangles. This gives us a precise binary boundary map for that resolution. We downsample the images to 125×125, and add varying amounts of Gaussian noise. These noisy, downsampled images serve as the inputs. To evaluate the predicted boundaries, we upsample the outputs back to 500×500 pixels. For EDTER, we use bilinear interpolation along with edge thinning to produce the upsampled boundaries. In our case, a byproduct of our parametric form for boundaries is that an intuitive method for upsampling boundaries naturally arises: by increasing the patch stride to 4, and rendering the patches as 4 times as large—68×68 pixels—we can create an upsampled version of our image while retaining our method's accuracy in boundary localization. We visualize this process in Figure 8.

We evaluate the upsampled outputs on the original 500×500 binary ground truth maps and vary the minimum distance threshold of the evaluation metric, which denotes how close a prediction must be to the ground truth to be used. The results are shown in Figure 7. Our F-score remains high across all matching thresholds, even when the inputs contain additive Gaussian noise.

Figure 8. One of the advantages of our parametric form for boundaries is that we can trivially upsample our output boundary maps by striding the predicted junction parameters, and scaling the patch size accordingly. This results in clean boundaries even when our output boundary images are upsampled from 125×125 (left) to 500×500 (right).

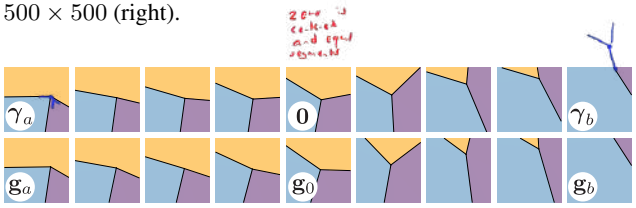

Figure 9. *Top*: Linear interpolation in our network's learned embedding space $\mathbb{R}^{D_\gamma}$ from value $\boldsymbol{\gamma}_a$ to zero and then to $\boldsymbol{\gamma}_b$. *Bottom*: A geometric interpolation in junction space $\mathbf{g} \in \mathcal{G}$ that passes through $\mathbf{g}_0 = (\mathbf{0}, 0, 1/3 \cdot \mathbf{1})$. The embedding has learned to be smooth and have an intuitive zero.

Linear interpolation in junction space. Surprisingly, we note that our network learns a spatially smooth manifold of junctions in its hidden state. Figure 9 visualizes some

Figure 10. Evolution of boundaries during iterations, in reading order. Early iterations are exploratory and unstructured, while later iterations feature consistent per-patch boundaries, resulting in clean average boundary maps.

Figure 11. Our model generalizes well to real images, and can find accurate boundaries in natural images. *Top left:* The input image, a low light image from the SIDD [1] dataset. *Top right:* Our network's predicted boundaries. *Bottom left:* The output mean feature map. *Bottom right:* The predicted distance map.

properties of its learned embedding of junction space. We generate equally-spaced samples $\boldsymbol{\gamma}_i \in \mathbb{R}^{D_\gamma}$ by linearly interpolating from a particular $\boldsymbol{\gamma}_a$ to 0 and then to a particular $\boldsymbol{\gamma}_b$. We see that the embedding space is smooth, and interestingly, that it learns to associate its zero with nearly-equal angles and a vertex close to the patch center. For visual comparison, we show an analogous interpolation from $\mathbf{g}_a$ to $\mathbf{g}_0 \triangleq (\mathbf{0}, 0, 1/3 \cdot \mathbf{1})$ and then to $\mathbf{g}_b$ by using a linear geometric interpolation in junction space $\mathcal{G}$, the expressions for which are in the supplement.

Evolution of the outputs over time. Figure 10 shows an example of how the distance map $\bar{d}[n]$ evolves during refinement. Specifically, we visualize the result of slicing similar to Equation 5 but with the regional distance functions d_k replaced by their nonlinear counterparts b_k. We see that early iterations are exploratory and unstructured, and that later iterations reach agreement.

Results on real images. Figures 1 and 11 show the qualitative results on real images. Despite being trained on synthetic data, our method can outperform existing state-of-the-art methods at multiple levels of real sensor noise present in ELD [34]. Our method produces crisp and well-defined boundaries despite high levels of noise.

Table 1. Inference time (in seconds) of EDTER [28] (without preprocessing by BM3D [9]), and Field of Junctions (FoJ) [33] run at various patch sizes, compared with our method. We compare both with the original (FoJ) implementation as well as our JAX reimplementation (JAX-FoJ). Unlike FoJ [33], our method does not require striding for fitting moderately-sized images in memory (runtimes reported with * did require striding). The average run times were calculated on an Nvidia A100 GPU.

		125×125	320×320	
EDTER [28]		0.130	0.130	
FoJ [33]	9×9	36.8	206	
	17×17	88.0	76.0*	(stride-3)
	25×25	162	57.6*	(stride-5)
JAX-FoJ	9×9	25.2	91.0	
	17×17	30.7	78.0*	(stride-3)
	25×25	117	220*	(stride-5)
Ours		0.0823	0.678	

Inference time. Table 1 compares the running time for different methods at two resolutions. We denote several runs for Field of Junctions where we had to increase the patch stride (optimize junctions at strided intervals) to avoid running out of memory. Notably, we are orders of magnitude faster than Field of Junctions, despite achieving similar performance. Additionally, because our network can handle inputs of any size, the run time depends on the input shape, in contrast with EDTER that has a maximal input size of 320×320 inputs and constant run time.

6. Conclusion

We have introduced a differentiable model that explicitly reasons about geometric primitives such as edges, corners, junctions, and regions of uniform appearance, by using boundary attention. Despite being trained on simple geometric synthetic data, our method generalizes to natural images, and predicts clean boundaries, even when the images it is applied to are significantly corrupted by noise. Furthermore, the parametric form of our model elicits a natural formulation for boundaries with subpixel precision, and its bottom-up patch-wise approach enables it to process any image at its native resolution and aspect ratio.

References

[1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. A high-quality denoising dataset for smartphone cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 8

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC superpixels. Technical report, 2010. 2

[3] Shir Amir, Yossi Gandelsman, Shai Bagon, and Tali Dekel. Deep ViT features as dense visual descriptors. In *ECCV Workshops on What is Motion For?*, page 4, 2021. 2

[4] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986. 2, 15, 19

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 2

[6] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces*. The University of Utah, 1974. 2

[7] Tony F. Chan and Luminita A. Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001. 2

[8] M Concetta Morrone and DC Burr. Feature detection in human vision: A phase-dependent energy model. *Proc. Royal Soc. B*, 235(1280):221–245, 1988. 2

[9] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007. 7, 8

[10] Piotr Dollár and C. Lawrence Zitnick. Fast edge detection using structured forests, 2014. 15, 19

[11] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 2004. 2

[12] James D. Foley and Andries Van Dam. *Fundamentals of interactive computer graphics*. Addison-Wesley Longman Publishing Co., Inc., 1982. 2

[13] William T. Freeman. *Steerable filters and local analysis of image structure*. PhD thesis, Massachusetts Institute of Technology, 1992. 2

[14] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Transactions on pattern analysis and machine intelligence*, 13(9):891–906, 1991. 2

[15] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, pages 10–5244. Citeseer, 1988. 2

[16] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Citeseer, 1989. 2

[17] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. 14

[18] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988. 2

[19] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023. 2

[20] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. *Advances in neural information processing systems*, 2011. 2

[21] Michael Maire, Pablo Arbelaez, Charless Fowlkes, and Jitendra Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008. 2

[22] 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. 2

[23] Nati Ofir, Meirav Galun, Boaz Nadler, and Ronen Basri. Fast detection of curved edges at low SNR. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[24] Nati Ofir, Meirav Galun, Sharon Alpert, Achi Brandt, Boaz Nadler, and Ronen Basri. On detection of faint edges in noisy images. *IEEE Transactions on pattern analysis and machine intelligence*, 42(4):894–908, 2019. 2

[25] Pierre Parent and Steven W. Zucker. Trace inference, curvature consistency, and curve detection. *IEEE Transactions on pattern analysis and machine intelligence*, 11(8):823–839, 1989. 2

[26] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985. 6, 14

[27] Juan Pineda. A parallel algorithm for polygon rasterization. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 17–20, 1988. 2

[28] Mengyang Pu, Yaping Huang, Yuming Liu, Qingji Guan, and Haibin Ling. EDTER: Edge detection with transformer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 6, 7, 8, 15, 19

[29] Daniel Scharstein and Chris Pal. Learning conditional random fields for stereo. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. 23

[30] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, pages I–I, 2003. 23

[31] Zhuo Su, Wenzhe Liu, Zitong Yu, Dewen Hu, Qing Liao, Qi Tian, Matti Pietikäinen, and Li Liu. Pixel difference networks for efficient edge detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5117–5127, 2021. 1, 15, 19

[32] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. MLP-Mixer: An all-MLP architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021. 5, 14

[33] Dor Verbin and Todd Zickler. Field of junctions: Extracting boundary structure at low SNR. In *Proceedings of*

the IEEE/CVF international conference on computer vision, 2021. 2, 3, 6, 7, 8, 11, 12, 15, 17, 19

[34] Kaixuan Wei, Ying Fu, Jiaolong Yang, and Hua Huang. A physics-based noise formation model for extreme low-light raw denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2758–2767, 2020. 1, 8, 14, 17

[35] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1395–1403, 2015. 1, 19

[36] Nan Xue, Gui-Song Xia, Xiang Bai, Liangpei Zhang, and Weiming Shen. Anisotropic-scale junction detection and matching for indoor images. *IEEE Transactions on Image Processing*, 27(1):78–91, 2017. 2

Supplemental Material

Boundary Attention: Learning to Find Faint Boundaries at Any Resolution

S1. The space of M-junctions

Here we provide the expressions for the support functions $s_j(x;\mathbf{g})$ and the unsigned distance function $d(x;\mathbf{g})$ from Section 3 of the main paper. We also describe the differences between our parameterization of junction space and the original one in the field of junctions [33], with the new parameterization's main advantages being the avoidance of singularities and the ability to define mechanisms for smooth interpolation. Our descriptions of these require introducing a few additional mathematical details. We provide these details for the general case of geometric primitives (junctions) $\mathbf{g}$ that have M angular wedges $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_M)$, for which the paper's use of $M = 3$ is a special case.

To begin, consider a local region $\Omega(x) \subset \mathbb{R}^2$ and fix a positive integer value for the maximum number of angular wedges $M > 0$ (the paper uses $M = 3$). Our partitions are parameterized by $\mathbf{g} \in \mathbb{R}^2 \times \mathbb{S}^1 \times \triangle^{M-1}$, where $\mathbb{S}^1$ is the unit circle and $\triangle^{M-1}$ is the standard $(M-1)$-simplex (*i.e.*, the set of M-vectors whose elements are nonnegative and sum to one). We use the notation $\mathbf{g} = (\boldsymbol{u}, \theta, \boldsymbol{\omega})$, where $\boldsymbol{u} = (u, v) \in \mathbb{R}^2$ is the *vertex*, $\theta \in \mathbb{S}^1$ is the *orientation*, and $\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_M)$ are barycentric coordinates (defined up to scale) for the M relative angles, ordered clockwise starting from θ. As noted in the main paper, our convention is to express the vertex coordinates relative to the center of region $\Omega(x)$, and we note again that the vertex is free to move outside of this region. We also note that up to $M-1$ of the angles ω_j can be zero. When necessary, we use notation $\tilde{\boldsymbol{\omega}} = (\tilde{\omega}_1, \tilde{\omega}_2, \ldots, \tilde{\omega}_M)$ to represent angles that are normalized for summing to 2π:

$$\tilde{\boldsymbol{\omega}} = \frac{2\pi\boldsymbol{\omega}}{\sum_{j=1}^{M} \omega_j}. \tag{15}$$

As an aside, we note that there are some equivalences in this parameterization. First, one can perform, for any $k \in \{1 \ldots (M-1)\}$, a cyclic permutation of the angles $\boldsymbol{\omega}$ and adjust the orientation θ without changing the partition. That is, the partition does not change under the cyclic parameter map

$$\omega_j \rightarrow \omega_{j+k(\mathrm{mod}M)} \tag{16}$$

$$\theta \rightarrow \theta - \sum_{j=M+1-k}^{M} \omega_j \tag{17}$$

for any $k \in \{1 \ldots (M-1)\}$. Also, an M-junction $(\boldsymbol{u}, \theta, (\omega_1, \ldots, \omega_M))$ provides the same partition as any M'-junction, $M' > M$, that has the same vertex and orientation along with angles $(\omega_1 \ldots \omega_M, 0 \ldots)$. This captures the fact that M-junction families are nested for increasing M.

As shown in Figure 12, other geometric features of a junction can be directly derived from the orientation and angles. The *central directions* $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_M)$ are

$$\psi_j = \theta + \frac{\tilde{\omega}_j}{2} + \sum_{k=1}^{j-1} \tilde{\omega}_k, \quad j \in \{1 \ldots M\}, \tag{18}$$

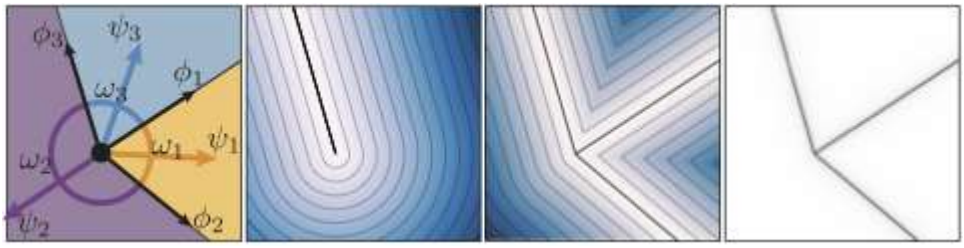

Figure 12. Anatomy of an M-junction $\mathbf{g} = (\mathbf{u}, \theta, \boldsymbol{\omega})$ with $M = 3$. *Left*: Boundary directions ϕ_j and central directions ψ_j are determined directly from relative angles $\boldsymbol{\omega}$ and orientation θ (which is equal to ϕ_1). *Middle panels*: Unsigned distance function for a boundary ray $d_3(x;\mathbf{g})$ and overall unsigned distance function $d(x;\mathbf{g})$, which is the minimum of the three per-ray ones. *Right*: Associated boundary function $b_\eta(x;\mathbf{g})$ using $\eta = 0.7$.

and the *boundary directions* $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_M)$ are given by $\phi_1 = \theta$ and

$$\phi_j = \theta + \sum_{k=1}^{j-1} \tilde{\omega}_k, \quad j \in \{2 \ldots M\}. \tag{19}$$

A key difference between our new parameterization of M-junctions and the original one [33] is that the latter comprises $(\boldsymbol{u}, \boldsymbol{\phi})$ and requires enforcing constraints $0 \leq \phi_1 \leq \phi_2 \leq \cdots \leq \phi_M \leq 2\pi$ (or somehow keeping track of the permutations of wedge indices that occur when these constraints are not enforced). The new $(\boldsymbol{u}, \theta, \boldsymbol{\omega})$-parameterization eliminates the need for such constraints.

As noted in the main paper's Section 3, we define the jth *support* $s_j(x; \mathbf{g})$ as the binary-valued function that indicates whether each point $x \in \Omega$ is contained within wedge $j \in \{1 \ldots, M\}$. Its expression derives from the inclusion condition that the dot product between the vector from the vertex to x and the jth central vector $(\cos\psi_j, \sin\psi_j)$ must be smaller than the cosine of half the angle $\tilde{\omega}_j$. Using Heaviside function $H(\cdot)$ we write

$$s_j(x; \mathbf{g}) = H\Big((x - \mathbf{u}) \cdot (\cos\psi_j, \sin\psi_j) - \cos(\tilde{\omega}_j/2)||x - \mathbf{u}||\Big). \tag{20}$$

As an aside, observe that this expression remains consistent for the case $M = 1$, where there is a single wedge. In this case, $\tilde{\boldsymbol{\omega}} = \tilde{\omega}_1 = 2\pi$ by Equation 15, and the support reduces to $s_1(x) = 1$ for all vertex and orientation values.

The *unsigned distance* $d(x; \mathbf{g})$ represents the Euclidean distance from point x to the nearest point in the boundary set defined by $\mathbf{g}$. It is the minimum over M sub-functions, with each sub-function being the unsigned distance from a boundary ray that extends from point $\mathbf{u}$ in direction ϕ_j. The unsigned distance from the jth boundary ray is equal to the distance from its associated line for all points x in its containing half-plane; and for other points it is equal to the radial distance from the vertex. That is,

$$d_j(x; \mathbf{g}) = \begin{cases} |(x - \mathbf{u}) \cdot (-\sin\phi_j, \cos\phi_j)|\,, & \text{if } (x - \mathbf{u}) \cdot (\cos\phi_j, \sin\phi_j) > 0 \\ \|(x - \mathbf{u})\|, & \text{otherwise.} \end{cases} \tag{21}$$

Then, the overall distance function is

$$d(x; \mathbf{g}) = \min_{j \in 1 \ldots M} d_j(x; \mathbf{g}). \tag{22}$$

Finally, analogous to Equation 7 in the main paper, we define a junction's boundary function $b_\eta(x; \mathbf{g})$ as the result of applying a univariate nonlinearity to the unsigned distance:

$$b_\eta(x; \mathbf{g}) = \left(1 + (d(x; \mathbf{g})/\eta)^2\right)^{-1}. \tag{23}$$

Figure 12 shows an example of a junction's distance function and its associated boundary function with $\eta = 0.7$.

Interpolation

Another advantage of the present parameterization compared to that of the original [33] is that it is a simply-connected topological space and so allows for defining mechanisms for smoothly interpolating between any two junctions $\mathbf{g}$ and $\mathbf{g}'$. In our implementation we simply define interpolation variable $t \in [0, 1]$ and compute interpolated junctions $\mathbf{g}^{(t)} = \{\boldsymbol{u}^{(t)}, \theta^{(t)}, \boldsymbol{\omega}^{(t)}\}$ using

$$\boldsymbol{u}^{(t)} = (1 - t)\boldsymbol{u} + t\boldsymbol{u}' \tag{24}$$

$$\tilde{\boldsymbol{\omega}}^{(t)} = (1 - t)\tilde{\boldsymbol{\omega}} + t\tilde{\boldsymbol{\omega}}', \tag{25}$$

and

$$\begin{aligned} \theta^{(t)} &= \text{atan2}(q, p), \text{with} \\ (p, q) &= \text{Slerp}\left((\cos\theta, \sin\theta), (\cos\theta', \sin\theta'), t\right), \end{aligned} \tag{26}$$

where $\text{Slerp}()$ is the 2D geometric spherical linear interpolation operator,

$$\text{Slerp}(\boldsymbol{p}, \boldsymbol{p}', t) = \frac{\sin\left((1 - t)\Delta\theta\right)}{\sin\left(\Delta\theta\right)}\boldsymbol{p} + \frac{\sin\left(t\Delta\theta\right)}{\sin\left(\Delta\theta\right)}\boldsymbol{p}',$$

with $\Delta\theta = \arccos(\boldsymbol{p} \cdot \boldsymbol{p}')$. The bottom row of Figure 9 in the main paper visualizes a set of samples from smooth trajectories in junction space using this mechanism.

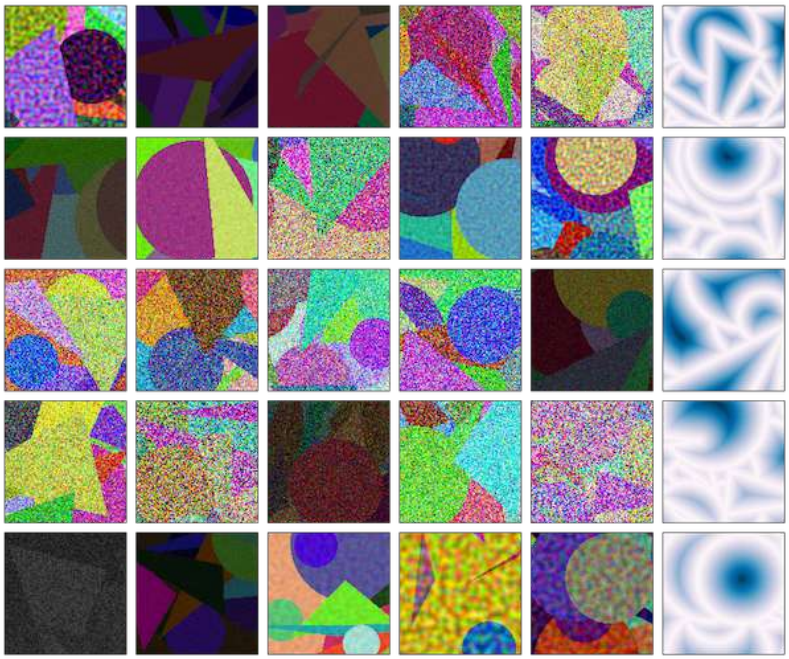

Figure 13. *Columns 1 to 5:* Examples of the synthetic data used to train our model using supervision with ground-truth boundaries. *Column 6:* Rendered distance maps corresponding to column 5. The training data contains random circles and triangles that each have a random RGB color, and the images are corrupted by various types and amounts of noise. Each noiseless image has an unrasterized, vector-graphics representation of its shapes and colors, which specify the clean image and exact boundary-distance map with unlimited resolution.

S2. Training Data

We find that we can train our model to a useful state using purely synthetic data, examples of which are depicted in Figure 13. In fact, we find it sufficient to use very simple synthetic data that consists of only two basic shapes—circles and triangles—because these can already produce a diverse set of local edges, thin bars, curves, corners, and junctions, in addition to uniform regions. We generate an image by randomly sampling a set of circles and triangles with geometric parameters expressed in continuous, normalized image coordinates $[0, 1] \times [0, 1]$. We then choose a random depth ordering of the shapes, and we choose a random RGB color for each shape. Importantly, the shape and color elements are specified using a vector-graphics representation, and the shape elements are simple enough to provide an exact, symbolic expression for each image's true boundary-distance map, without approximation or rasterization. They also allow calculating the precise locations, up to machine precision, for all of the visible corners and junctions in each image.

At training time, an input image is rasterized and then corrupted by a random amount and type of noise, including some types of noise that are spatially-correlated. This forces our model to only use color as its local cues for boundaries and grouping; and it forces it to rely heavily on the topological and geometric structure of curves, corners and junctions, as well as their contrast polarities. The highly-varying types and amounts of noise also encourages the model to use large window functions $w(x; \mathbf{g})$ when possible, since that reduces noise in the gather operation and reduces variance $\nu_f[n]$.

Our dataset, which we call Kaleidoshapes, will be released upon publication, along with the code for generation, training and evaluation.

Shapes and colors. For our experiments, we rasterized each image and its true distance map at a resolution of 240×320 images, with each one containing between 15 and 20 shapes. We used a $40:60$ ratio of circles to triangles. In terms of normalized coordinates, circles had radii in the range $[0.05, 0.2]$ and triangles had bases in the range $[0.02, 0.5]$ and heights in the range $[0.05, 0.3]$. This allows triangles to be quite thin, so that some of the local regions $\Omega(x)$ contain thin bar-like structures. Additionally, we included a minimum visibility threshold, filtering out any shapes whose visible number of rasterized pixels is below a threshold. Colors were selected by uniformly sampling all valid RGB colors. During training, batches consisted of random 125×125 crops.

Noise. For noise types, we used combinations of additive zero-mean Gaussian noise; spatially average-pooled Gaussian noise; Perlin noise [26], and simulated photographic sensor noise using the simplified model from [34]. The total noise added to each image was sampled uniformly to be between 30% and 80% of the maximum pixel magnitude, and then noise-types were randomly combined with associated levels so that they produced the total noise level. Since zero-mean noise can at times result in values below 0 or above the maximum magnitude threshold, we truncate any pixels outside of that range.

S3. Model Details

Our model is designed to be purely local and bottom up, with all of its compositional elements operating on spatial neighborhoods in a manner that is invariant to discrete spatial shifts of an image. Its design also prioritizes having a small number of learnable parameters. Here we provide the details of the two blue blocks in the main paper's Figure 3: Neighborhood MLP-Mixer and Neighborhood Cross-attention. We implement our model in JAX, and we will publicly share our model code and its pretrained weights upon publication.

S3.1. Neighborhood MLP-Mixer

Our neighborhood MLP-mixer is a shift invariant, patch-based network inspired by MLP-mixer [32]. It replaces the image-wide operations of [32] with patch-wise ones. Given an input image, we first linearly project its pixels from $\mathbb{R}^3$ to dimension $\mathbb{R}^{D_\gamma}$ (we use $D_\gamma = 64$), which is followed by two neighborhood mixing blocks. Each neighborhood mixing block contains a spatial patch mixer followed by a channel mixer. The spatial patch mixer is implemented as two 3×3 spatial convolutions with weights tied across channels. It thereby combines spatial patches of features with all channels (and patches) sharing the same weights. Following [32], we use GELU [17] activations. The channel mixer is a per-pixel MLP with spatially-tied weights. To handle border effects in our neighborhood MLP-mixer, we apply zero-padding after the initial projection from $\mathbb{R}^3$ to $\mathbb{R}^{64}$, and then we crop to the input image size after the second neighborhood mixing block to remove features that correspond to patches without full coverage, *i.e.*, patches that contain pixels outside of the original image.

S3.2. Neighborhood Cross-attention

The neighborhood cross-attention block similarly enforces shift-invariance and weight sharing across spatial neighborhoods. Inside this block are two transformer layers whose cross-attention components are replaced with neighborhood cross-attention components that are restricted to a spatial neighborhood of pixels. We use 11×11 neighborhoods in our implementation. In each neighborhood containing a query token, we add a learned positional encoding to the key/value tokens which is relative to the neighborhood's center and is the same for all neighborhoods. Then the query is updated using standard cross-attention with its neighborhood of key/values. We use 4 cross-attention heads. Like the standard transformer, each neighborhood cross attention component is followed by an MLP, dropout layer, and additive residual. To handle border effects, we zero-pad the key and value tokens so that every query attends to an 11×11 neighborhood, and then zero-out any attention weights involving zero-padded tokens.

S3.3. Training details

We pretrain the neighborhood MLP-mixer and the first boundary attention block on a simplified variation of our Kaleidoshapes dataset, where each image is 100×100 and contains a single triangle and circle with additive zero-mean Gaussian noise. We omit the global losses of Equations 9 and 10 during this pretraining phase. This primes the network to learn meaningful hidden states $\boldsymbol{\gamma}[n]$ and prevents the "collapsing" of junctions, where the boundary-consistency loss (*i.e.* the sum over pixels of variance of distance $\nu_d[n]$) dominates and the network learns to predict all-boundaryless patches that are globally consistent but inaccurate. Because of data imbalance—only a small fraction of regions $\Omega_n(x)$ contain corners or junctions—we add an additional spatial importance mask to prioritize the regions that contain a corner (*i.e.*, a visible triangle vertex) or a junction (*i.e.*, an intersection between a circle and a triangle's edge). Our data generation process produces a list of all non-occluded vertices and intersections in each image, and we use these values to create a spatial importance mask with gaussians centered at each of these points. In practice, we use gaussians with a standard deviation of 7 pixels. This mask is added to the loss constant C.

The final stage of training adds a second boundary attention block with weights that are initialized using a copy of the pretrained weights of the first boundary attention block. We use $100,000$ crops of size 125×125 from our Kaleidoshape images (10% withheld for testing) and the full set of losses; and we optimize all of the model's parameters, including those of the neighborhood MLP-mixer and the first boundary attention block. Like in pretraining, we add a spatial importance that prioritizes region containing a corner (*i.e.*, a visible triangle vertex) or a junction (*i.e.*, a visible intersection between the boundaries of any two shapes).

Our trained weights will be released upon publication.

S4. Qualitative Behavior for Natural Images

In Figures 15 and 14, we show how the model behaves on noiseless natural images that contain texture and recognizable objects. In particular, Figure 14 emphasizes how the boundary maps produced by our model qualitatively differ from those of many classical bottom-up edge-detectors and also from those of learned, end-to-end models that have been trained to match human annotations. The figure compares our output to that from Canny [4], Field of Junctions [33] with a patch size of 11, Pidinet [31], and EDTER [28]. The latter two methods are trained on human annotated data, whereas the former two methods, like our model, are not. (Note that inputs for all models besides EDTER [28] were 300×400. Input to EDTER was down-sampled to 225×300 due to its input size constraint.)

We find that our model produces finer structures than the end-to-end learned models [10, 28] because it is trained to only use local spatial averages of color as its cue for boundaries and grouping. It does not include mechanisms for grouping based on local texture statistics, nor based on non-local shape and appearance patterns that have semantic meaning to humans. Compared to the bottom-up methods of Canny [4] and Field of Junctions [33], our model has the advantage of automatically adapting the sizes of its output structures across the image plane, through its prediction of field $\mathbf{p}[n]$. In contrast, the Field of Junctions and Canny both operate at a single pre-determined choice of local size, so they tend to oversegment some places while undersegmenting others.

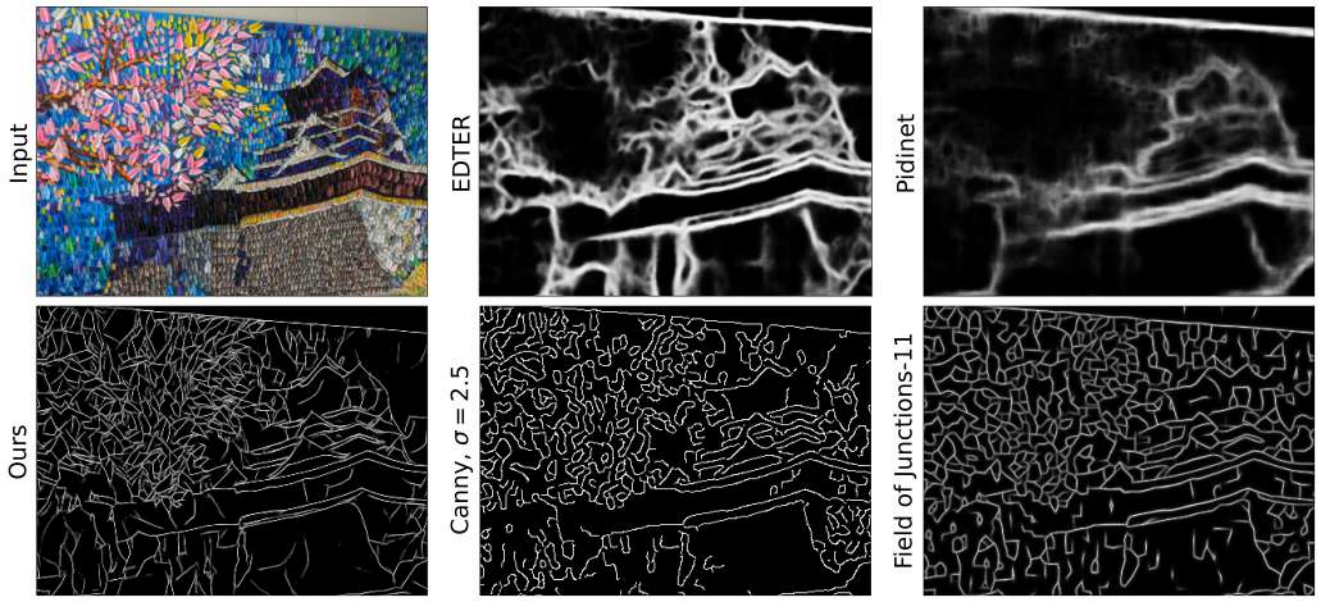

Figure 14. Qualitative behavior of our model's output boundaries $\bar{b}_\eta[n]$ on noiseless natural images, compared to those of end-to-end models EDTER [28] and Pidinet [31] that are trained to match human annotations; and compared to two bottom-up methods that, like our model, are not trained to match human annotations: Canny [4], and Field of Junctions [33] with patch size 11.

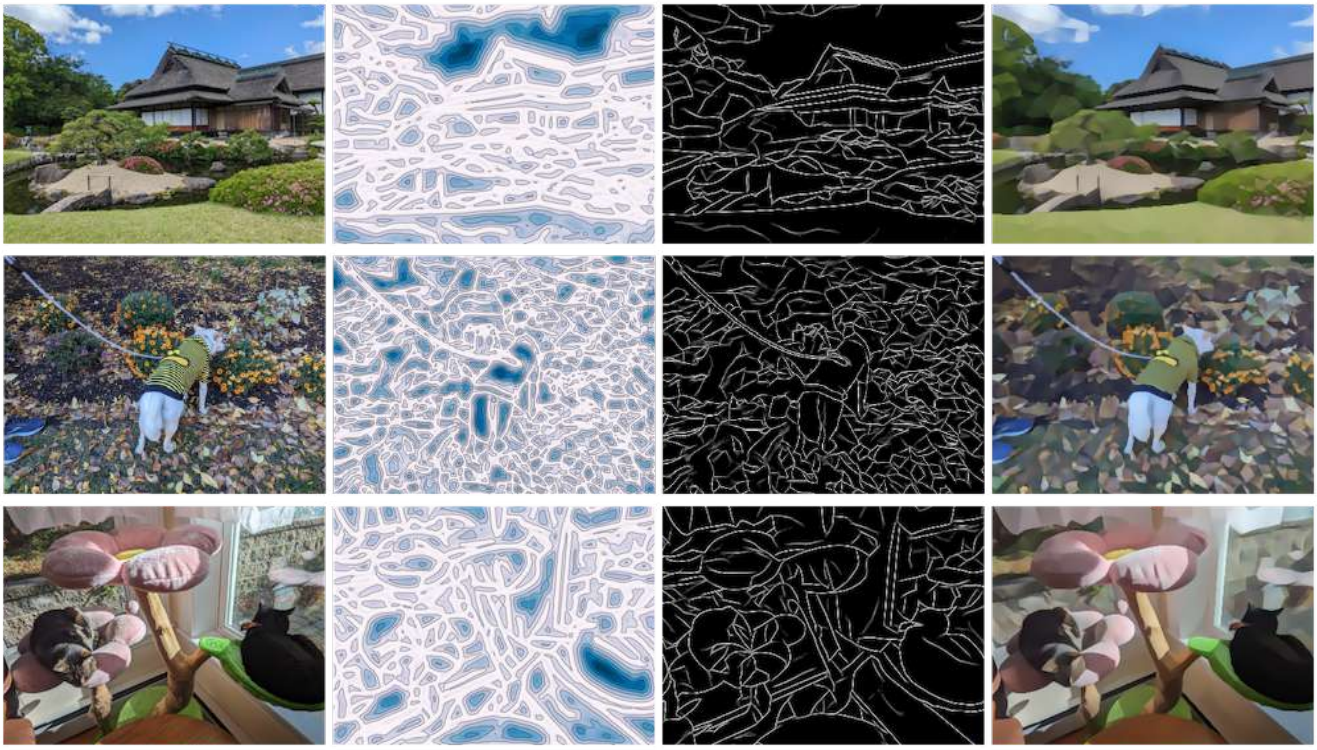

Figure 15. Qualitative behavior of our model on noiseless natural images. *From left to right:* Input image $\mathbf{f}[n]$, output distance map $\bar{d}[n]$, output boundary map $\bar{b}_\eta[n]$ with $\eta = 0.7$, and output boundary-smoothed features $\bar{\mathbf{f}}[n]$.

S5. Additional Examples for Low-light Images

Figure 16 shows examples of applying our model to indoor images taken by an iPhone XS in low light conditions.

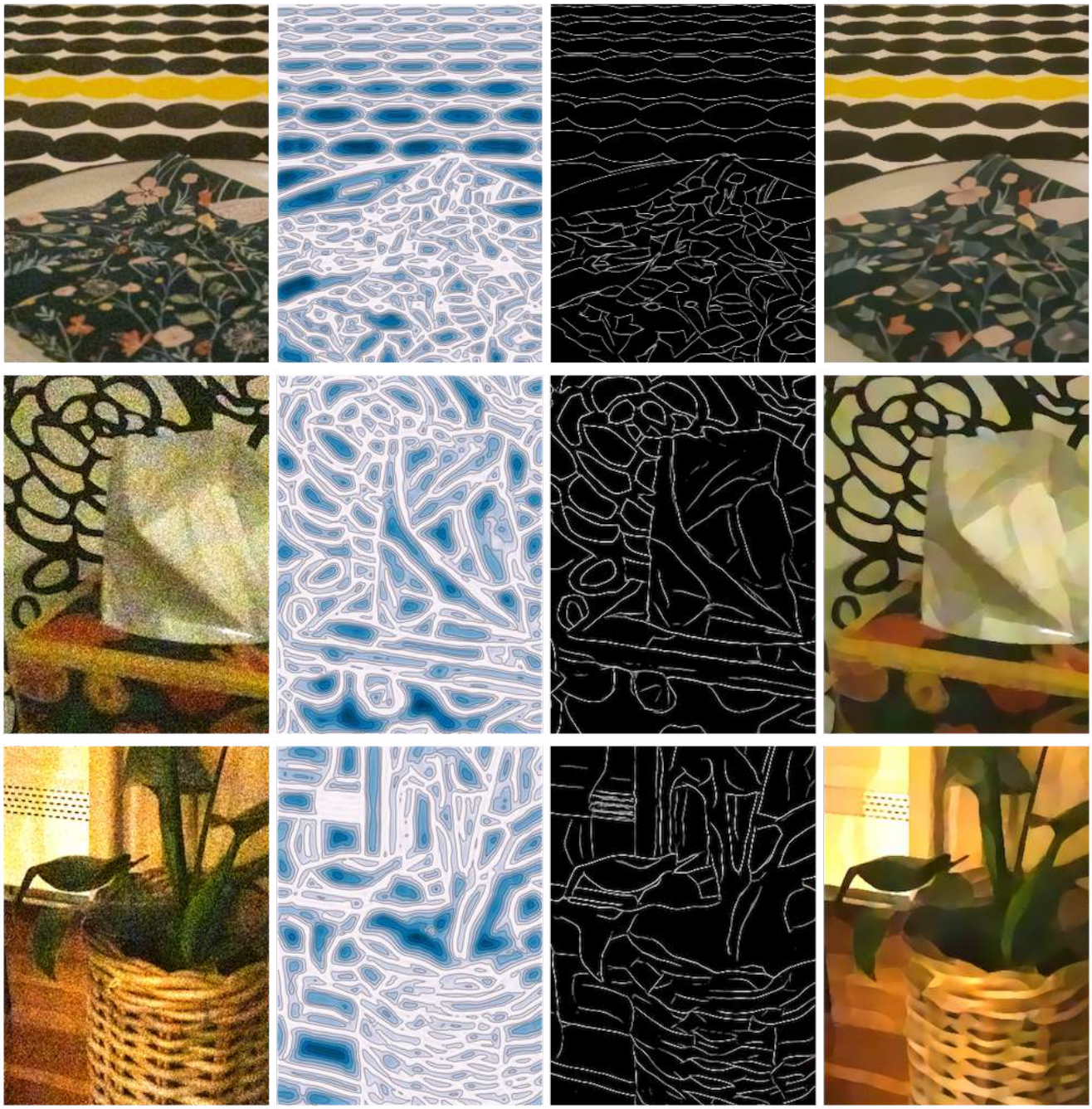

Figure 16. Visualization of our model's output for low-light images captured by an iPhone XS. *From left to right:* Input image $\mathbf{f}[n]$, output distance map $\bar{d}[n]$, output boundary map $\bar{b}_\eta[n]$ with $\eta = 0.7$, and output boundary-smoothed features $\bar{\mathbf{f}}[n]$.

Figure 17 provides additional comparisons for a sample of varying-noise images from the ELD dataset [34]. When detecting boundaries at low signal-to-noise ratios, it is difficult to accurately discern finer structures as the noise level increases. Some algorithms, such as Field of Junctions [33], have tunable parameters such as patch-size that provide control over the level of detection. A small patchsize allows recovering fine structures in lower noise situations, but it causes many false

positive boundaries at high noise levels. Conversely, a large patchsize provides more resilience to noise but has not ability to recover fine structure at all. Our model reduces the severity of this trade-off by automatically adapting its local windowing functions in ways that have learned to account for both the amount of noise and the local geometry of the underlying boundaries.

In Figure 17 we see that our model is able to capture the double-contour shape of the curved, thin black bars, and that it continues to resolve them as the noise level increases, more than the other low-level methods. We also note that only the low-level models resolve this level of detail in the first place: The models trained on human annotations—EDTER, HED, Pidinet, and Structured Forests—miss the double contour entirely, estimating instead a single thick curve. We emphasize again that a user can adjust the behavior of Canny and Field of Junctions by tuning their local size parameters, either the filter size for Canny or the patchsize for Field of Junctions. Increasing the local size improves their resilience to noise but reduces their spatial precision. Neither system provides the ability to estimate fine grained details *and* withstand noise, like our model does.

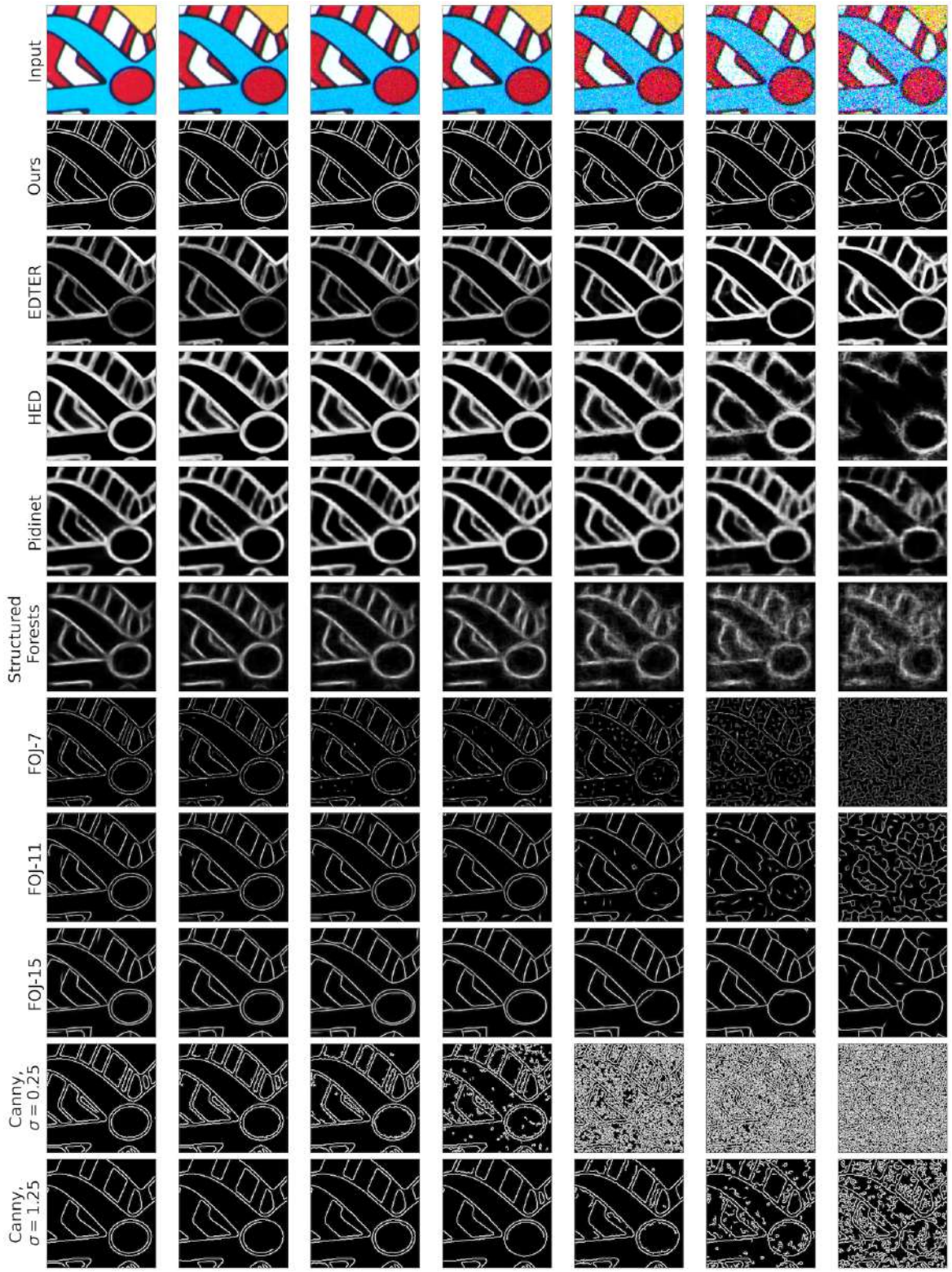

Figure 17. Qualitative comparison between our model's output boundaries $\bar{b}_\eta[n]$ and those of other methods, for a crop from the ELD dataset under increasing amounts of photographic noise. We compare to end-to-end models that are trained to match human annotations (EDTER [28], HED [35], Pidinet [31], and Structured Forests [10]) in addition to low-level models that are not (Canny [4], and Field of Junctions (FOJ) [33]).

Figure 18 contains additional examples of images cropped from the ELD dataset. Here we include examples with even higher levels of noise to show the complete degradation of our algorithm and others.

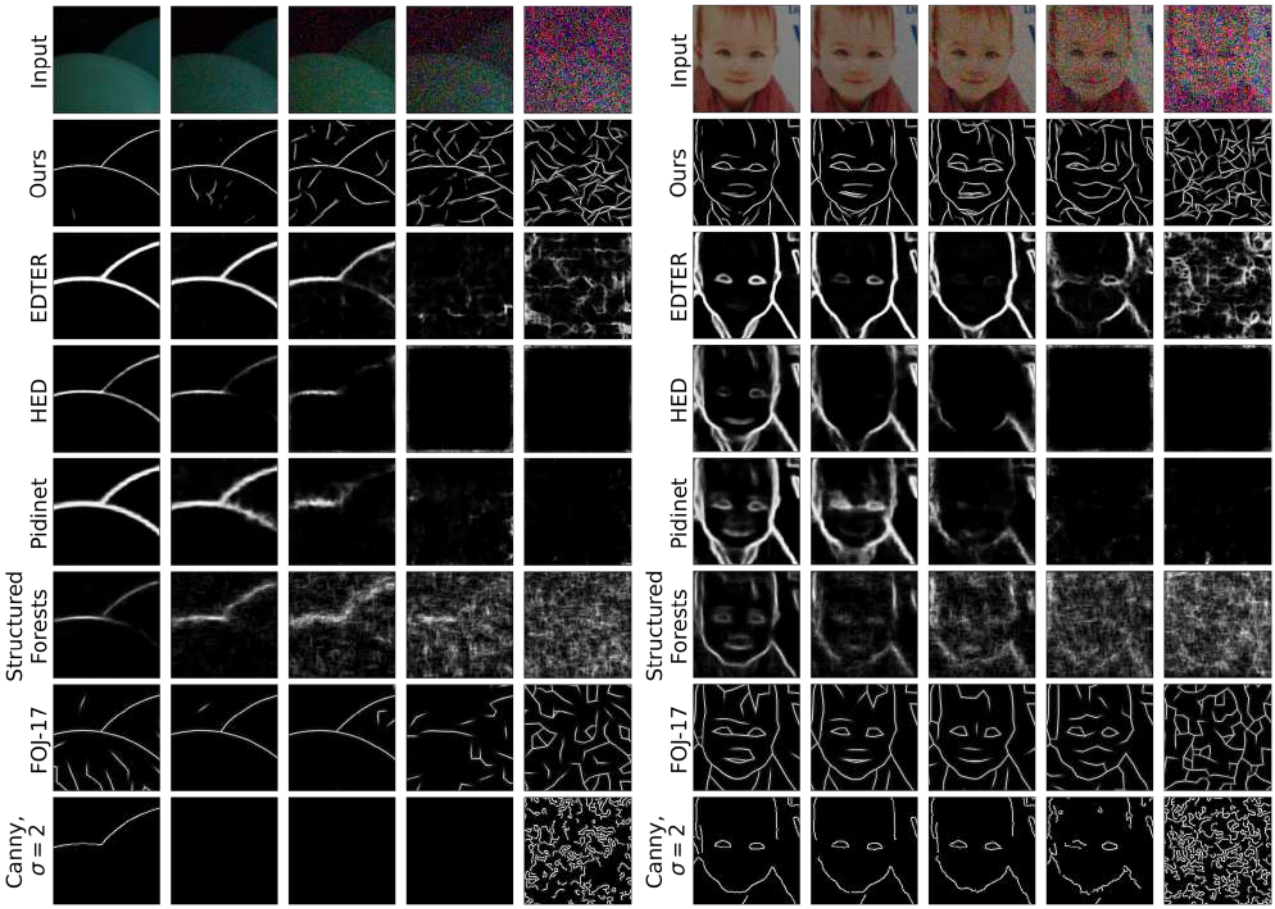

Figure 18

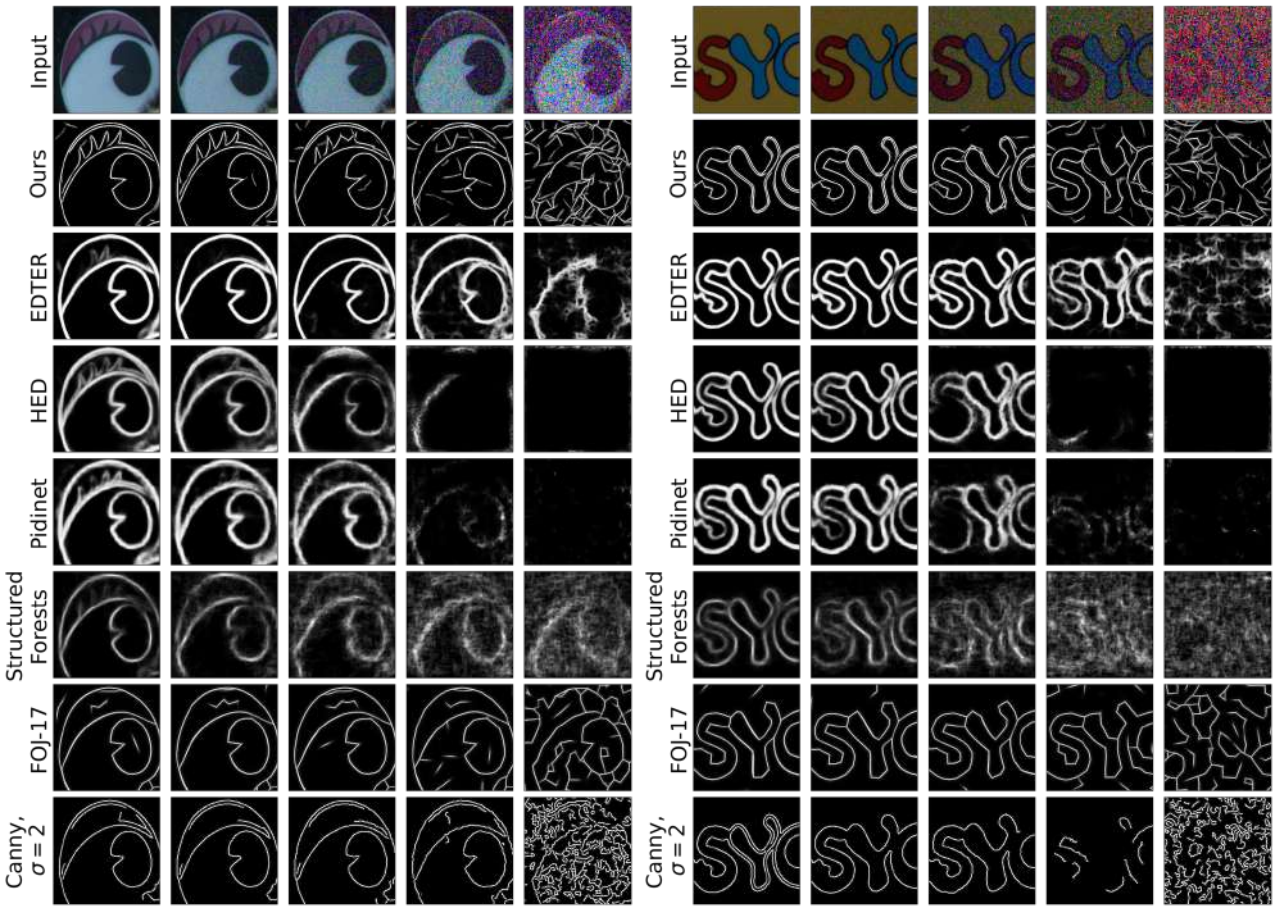

Figure 18. *(cont.)*

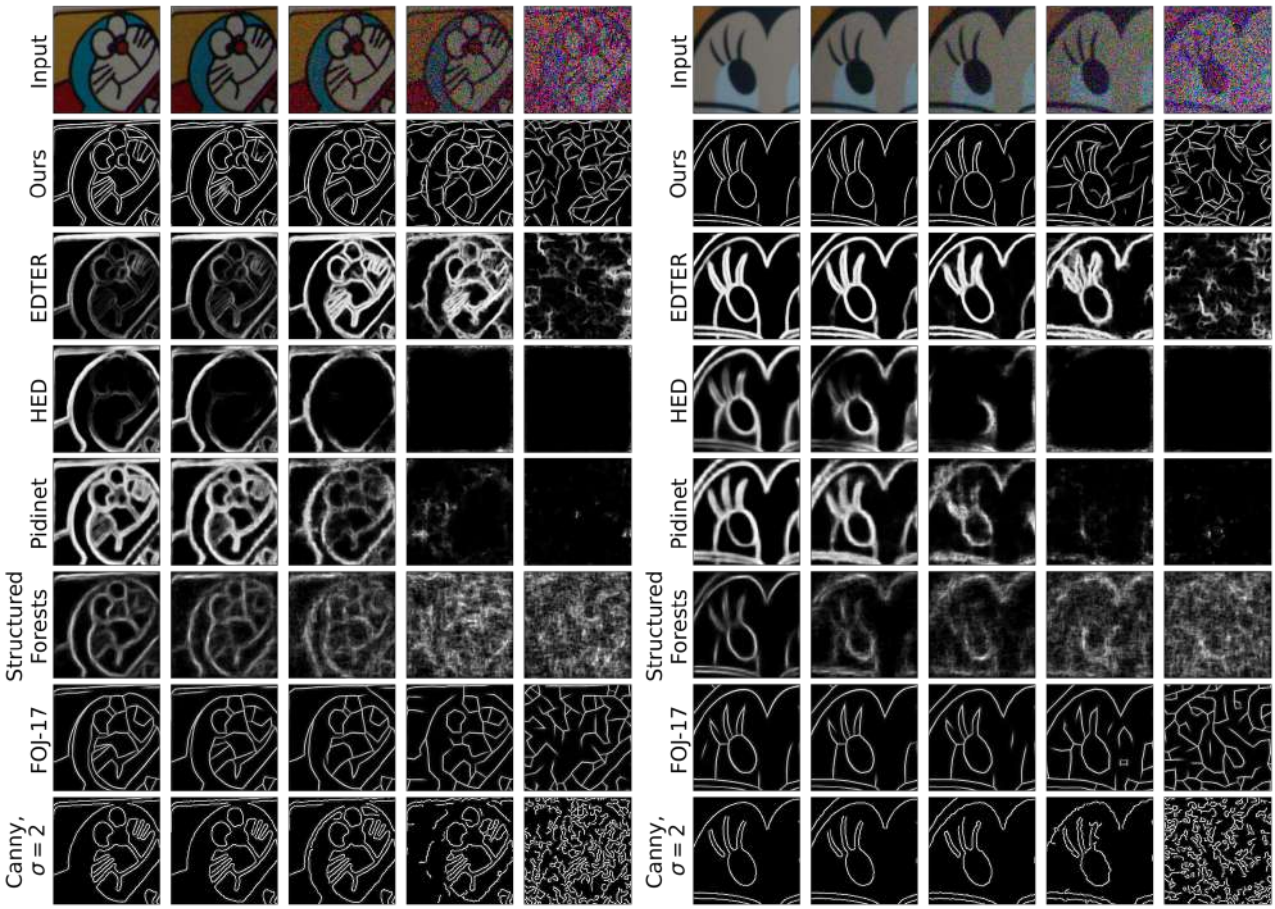

Figure 18. *(cont.)*

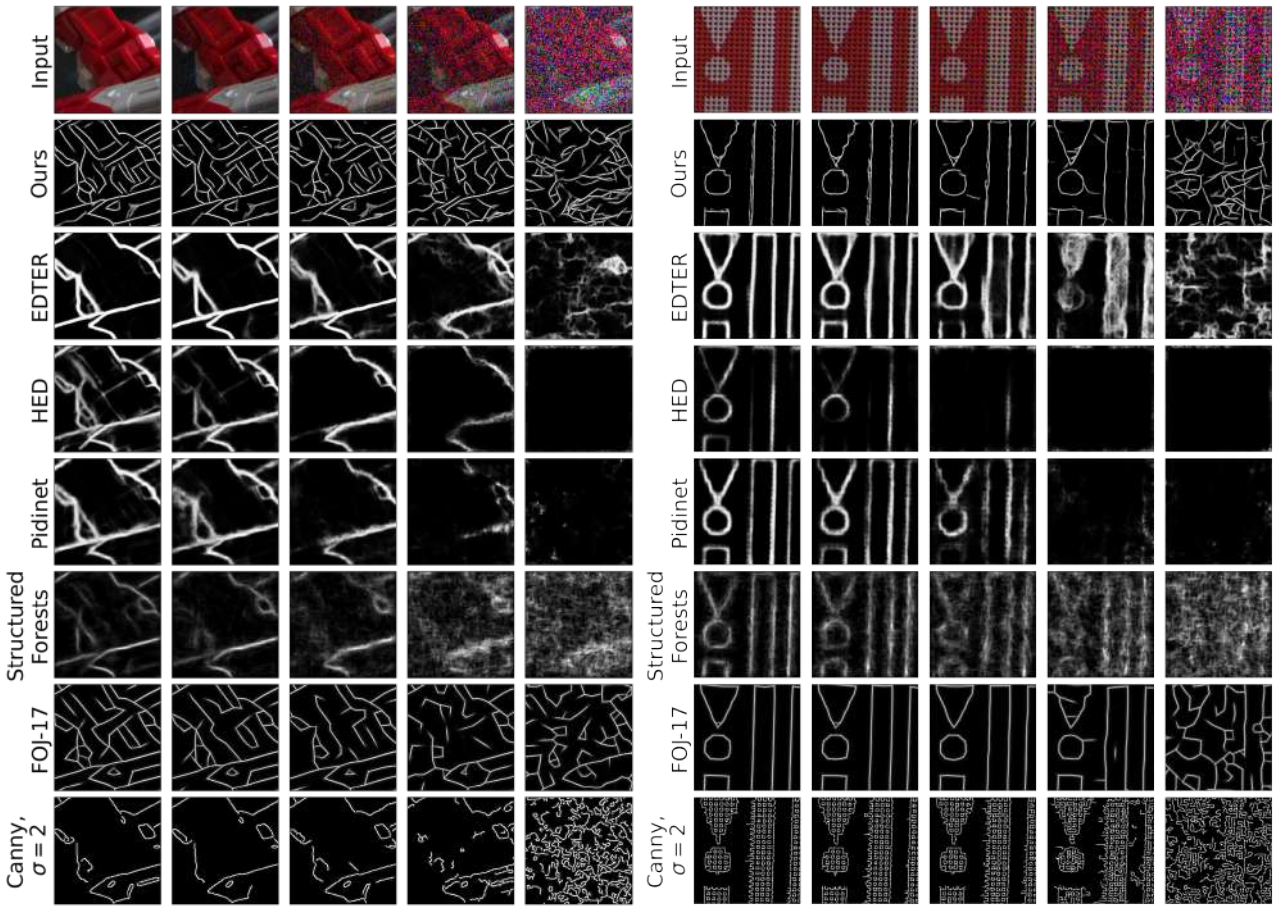

Figure 18. *(cont.)* Additional qualitative comparisons between our model's output boundaries $\bar{b}_\eta[n]$ and those of other methods, using crops from the ELD dataset under increasing amounts of photographic noise, including very high levels of noise.

S6. Additional Uses of Our Model

Here we demonstrate to uses of our model that follow directly from its output: hole-filling in RGBD images and non-photorealistic stylization.

S6.1. Color-based Depth Completion

Figure 19 shows an example of using our model for simple hole-filling in the depth channels of RGBD images from the Middlebury Stereo Datasets [29, 30]. We run our model on the RGB channels, and then for each pixel n that has a missing depth value, we use our model's output local attention kernels $a_n(x)$ to fill in that pixel's value using an attention-weighted average of the observed depth values around it. This simple algorithm can be applied whenever the hole sizes are smaller than the maximum diameter of our attention maps, which is 34×34 in our current implementation).

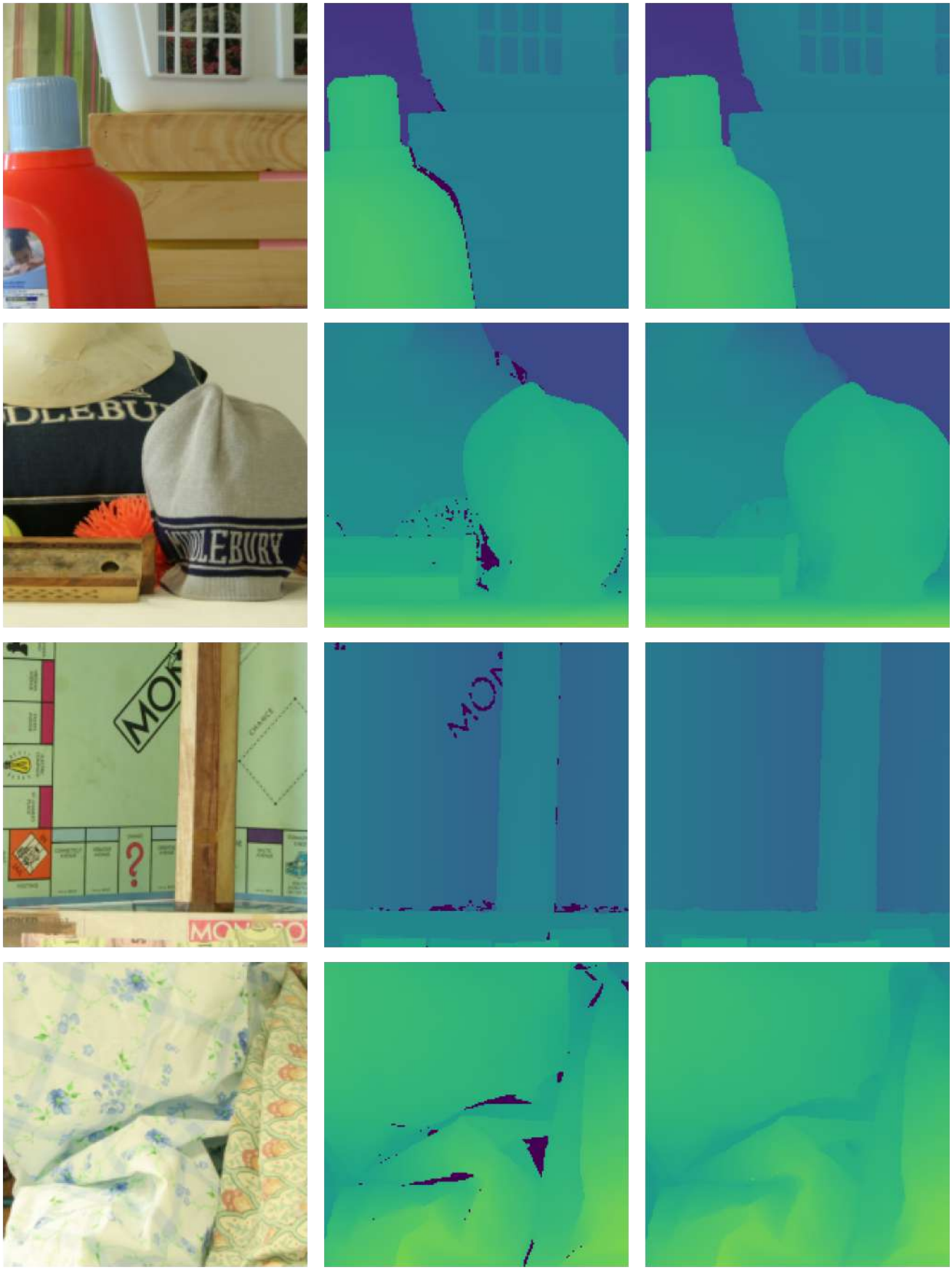

Figure 19. Using our model for depth completion in RGBD images. *Left:* Input RGB channels. *Middle:* Input depth channel, with dark blue indicating missing values. *Right:* Completed depth using our model's output attention kernels.

S6.2. Application: Photo Stylization

Figure 20 shows examples of using our model's output for image stylization, by superimposing an inverted copy of the output boundary map $\bar{b}_\eta[n]$ onto the smoothed colors $\bar{\mathbf{f}}[n]$.

Figure 20. Examples of stylized natural photographs, created by imposing our method's output boundary map onto the output smoothed colors.