Split-GCN: Effective Interactive Annotation for Segmentation of Disconnected Instance

Namgil Kim, Barom Kang, and Yeonok Cho

Abstract—Annotating object boundaries by humans demands high costs. Recently, polygon-based annotation methods with human interaction have shown successful performance. However, given the connected vertex topology, these methods exhibit difficulty predicting the disconnected components in an object. This article introduces Split-GCN, a novel architecture based on the polygon approach and self-attention mechanism. By offering the direction information, Split-GCN enables the polygon’s vertices to move more precisely to the object boundary. Our model successfully predicts disconnected components of an object by transforming the initial topology using the context exchange about the dependencies of vertices. Split-GCN demonstrates competitive performance with the state-of-the-art models on Cityscapes and even higher performance with the baseline models. On four cross-domain datasets, we confirm our model’s generalization ability.

Index Terms—Computer vision, human interactive learning, segmentation, semi-auto labeling

1 INTRODUCTION

RECENTLY, data-driven deep learning techniques have become popular due to their outstanding performances in object segmentation [11, 2, 3, 4, 5]. To improve the network’s performance, it is necessary to train with larger amounts and bigger scale of annotated data. Unfortunately, annotating high-quality ground-truth data for object segmentation is heavily time-consuming. For instance, it takes 40 seconds for humans to annotate a single object in an image.

There are several previous studies on user interactive annotation methodologies such as scribbles [6, 7], clicks [8, 9, 10], and polygons [11, 12, 13]. Scribbles allow users to draw and drag lines on the region of interest to adjust. On the other hand, clicks request users to click instead of drawing, which is similar to the scribbles but less burdens. However, both methods are different from how users generally annotate. This causes users to learn how to annotate in order to make a detailed correction. The polygon-based methods are more practical since these methods resemble the user’s actual process in the annotation. The concept of these methods is that contours are connected according to the sequence of the points like the Eulerian path. However, because of such connected contours, they have limitations in expressing the disconnected target object occluded by other objects [11, 13]. This is a fatal drawback since, in practical usage, users regularly encounter separated components of an object.

In this paper, based on the graph convolutional network (GCN), we introduce Split-GCN which tackles the problem of interactive annotations for object segmentation when occluded object exists. Starting with one initial polygon, Split-GCN yields multiple polygons, each of which corresponds to a disconnected component split by the occlusion. To do this, we propose a novel architecture called separating network which uses a self-attention mechanism. The self-attention mechanism outputs the probabilities that a pair of nodes is connected. Then the separating network leaves only the two most probable edges for each node to find the boundary of a possibly split object.

Our model also exploits the motion vector branch to recognize the direction towards the target object’s boundary, similar to looking up a map to find the exact route to an unknown destination. This directional information from the feature extraction network gives hints to the object’s boundary and, similarly, to whether two nodes are connected or not. If two very close nodes have opposite directions, it is highly likely that these two nodes are disconnected. Combining the information from motion vector and separate network, we finally obtain a full segmentation from occluded object.

We conduct various experiments to demonstrate superiority in the performance and generalizability of our proposed model on Cityscape [14] and four datasets from different domains, KITTI [15], Rooftop [16], ADE20K [17], and Card.MR [18]. Our Split-GCN shows 29.6 AP in automatic mode experiments, which is very competitive compared to the state-of-the-art models on the Cityscapes test set. Moreover, on the same validation dataset, Split-GCN shows the highest performance of 76.6 mIoU, 52.5 F1px, and 67.5 F2px scores, improving by 2.9 mIoU, 4.8 F1px, and 3.9 F2px scores from the best performing baseline model, Curve-GCN [13]. In cross-domain experiments, trained with only 10% of the total data set, our model outperforms the baseline models in all datasets as well. In summary, our contributions are three-fold:

- We introduce a novel architecture named Split-GCN, which effectively transforms the initial polygon topology into the target object topology.
- To predict the disconnected components more accurately, we develop the separating network using self-attention.
- Split-GCN demonstrates 3.9% and 2.6% bigger mIoU, respectively on Cityscapes and other cross-domain datasets, than Curve-GCN which is the latest model using polygon-based methods as our model.

2 RELATED WORK

2.1 Pixel-Based Methods

Most interactive annotation models adopt a pixel-wise approach, which predicts an instance based on the pixel unit. Early studies introduced the methods that scribble instance to distinguish a foreground/background and use graph-cuts to segment [6]. Later, Rother et al. [7] proposed GrabCut, which uses graph-cuts iteratively and requests bounding boxes to the user to perform segmentation.

Many studies recently applied deep neural networks-based models and achieved exceptional performance in the interactive annotation field [19, 20]. The first study using deep architecture was deep interactive object selection [10], which converts positive/negative clicks received by the users into the euclidean distance map to segment objects of interest. Following [10], Mahadevan et al. [21] improved the performance by iterating added user clicks on the errors in the current segmentation. Utilizing extreme clicks provided by annotators as inputs, DEXTR [8] achieved outstanding results in the interactive annotation. In recent studies, IOG [22] succeeded in obtaining high quality instance segmentation by using an annotated inner point of the object, in addition to the bounding box whose vertices serve as outer points. Finally, [23] proposed FocusCut, in which the user magnifies the region of low precision so that the model can refine the result. However, as applying the pixel-
wise method, it is challenging for most of these models to distinguish between foreground and background pixels, especially when they are similar.

### 2.2 Contour-Based Methods

There are two major contour-based methods: the level sets approach and the polygon-based approach. The level sets method predicts object boundaries by continuously taking derivative on the energy function of the curve and data. Ping Hu et al. [3] derived the level sets function using a convolutional neural network (CNN). Marcos et al. [25] and Wang et al. [9] optimized CNN based level sets in end-to-end fashion.

One of the earliest polygon-based annotation studies was Polygon-RNN [12], which sequentially predicts vertices of an instance one by one using CNN and the recurrent neural network (RNN). To solve the low-resolution accuracy from [12], PolygonRNN++ [11] improved the performance by applying a gated graph sequence neural network [26] to increase the output resolution. Curve-GCN [13] used the GCN to predict all vertices at once and significantly saved the inference time than PolygonRNN++. However, Curve-GCN shows a limited prediction of disconnected objects because it keeps the connected initial topology until the final prediction.

To solve this limitation, BCNet [27] uses bilayer GCN and self-attention to regress the object contour and masks respectively, especially for the occluded object splitted by the occluder. Our proposed model follows the polygon-based approach. We directly predict the object boundary using numerous uniformly spaced points. Especially, our approach allows an efficient prediction for the split of edge instances by transforming the initial topology.

### 3 Method

Our goal is to develop a model that can express the disconnected components of an object’s polygon shape that is occluded by another one using only a few control points. We introduce Split-GCN, a novel deep architecture that consists of the polygon-based approach [11], [13] and self-attention mechanisms [28]. Split-GCN intuitively outlines an object by connecting points in sequential order. We consider the following setting as in [11], [13]: The user receives an image and draws a bounding box, and we enlarge the box by 10-20%, randomly. Then, Split-GCN receives such cropped image as input and predicts the shape of the object as a regression problem by shifting offsets. Simultaneously, separating network reconstructs the adjacency matrix to predict whether the boundaries of the objects are connected or disconnected.

As shown in Fig. 1, the Split-GCN primarily consists of two parts: an encoder (feature extraction network) to extract the boundary information of an object and a decoder (novel graph composition network) to capture the shape of an object. We first explain the encoder architecture for feature extraction in Section 3.1 and then explain the decoder for capturing disconnected points of an object in Section 3.2. In Section 3.3, we explain how to train our model. In Section 3.4 and, finally, we describe the details of interactive inference, which interacts with a human to perform semi-auto labeling.

#### 3.1 Feature Extraction Network

The feature extraction network aims to provide efficient object boundary information so that the decoder can express a deforming polygon. We extract high-level semantic information from the object of interest using CNN. We also extract motion vector information, which guides the polygon to the object’s boundary at each pixel using the motion vector branch.

*Feature Extraction.* For extracting features, we first resize the bounding box generated by the annotator to $224 \times 224$ and then encode the cropped bounding box to high-quality semantic information using CNN. We deploy ResNet-50 [29] as a backbone of our feature extraction network. However, we remove the average
pooling and fully connected layers to attach our proposed decoder, which captures the object’s polygon shape. Then, we adopt the residual encoder architecture from [11], which increases the resolution of feature maps without reducing the receptive field.

As a result, we obtain high-quality semantic information while maintaining low-level details. As shown in Fig. 1, we use bilinear interpolation as the highest possible resolution size at each convolution stage before concatenation. After concatenation, we produce feature maps denoted as \( F_i \) in \( \mathbb{R}^{C_i \times H_i \times W_i} \) by using \( 3 \times 3 \) convolutional filters with stride 2, batch normalization [30], and ReLU activation function where \( C_i \), \( H_i \), and \( W_i \) are indicate channel, height, and width of \( F_i \), respectively.

Motion Vector Branch. Our proposed framework moves vertices of the polygon to target boundary in the decoder. We assume that the model can efficiently reach an object’s boundary if it has direction information. Motion vector branch exploits one convolutional layer and one fully connected layer to predict the ground truth (GT) motion map \( \tilde{U}_{gt} \in \mathbb{R}^{2 \times H_i \times W_i} \), which is the direction to the boundary. In other words, the motion vector branch predicts motion map \( \tilde{U}_{gt} \) with the magnitude and direction of the object boundary by utilizing \( F_i \). We denote \( \theta \) as the network’s parameters.

We design a GT motion map \( \tilde{U}_{gt} \) with two channels by differentiating each coordinate of \((x, y)\) respectively. \( \tilde{U}_{gt} \) is

\[
\tilde{U}_{gt}(x, y) = \frac{\phi_{\theta} (x, y)}{\left[ \phi_{\theta}(x, y) \right]} \tag{1}
\]

where \( \phi_{\theta}(x, y) \) is the directional information \( \leq \| (x, y) - (x_0, y_0) \|_2 \) and \( \mathcal{Q} \) is the set of GT boundary points. In practice, the model has difficulty predicting the GT boundary corresponding to the object because the boundary has the size of only one pixel. Thus, we use the \( 9 \times 9 \) Gaussian filter to make the boundary thicker.

3.2 Graph Composition Network

The graph composition network with our separating network intuitively renders the target boundary using \( N \) control points and identifies disconnected control points to express the split component. Here, intuitive rendering means connecting points one by one in a straight line. Similar to [13], we define each control point as a random variable and use GCN to find the correct position of the control point through the relation of nearby points that contains the cues of the target object. We grasp the dependencies among the vertices using our proposed separating network; we reconstruct graph topology and predict disconnected points following these dependencies. Inspired by ViT [31] successes using Transformer [28] in vision task, we utilize a self-attention Transformer network to take advantage of such complicated dependencies.

Vertex Embedding. The vertex embedding produces features for the vertices in GCN. First, we normalize each channel of the predicted motion map \( \tilde{U}_{gt} \) by \( H_i \) and \( W_i \). Then, we concatenate \( F_i \) with these two channels to guide the network to head toward the target boundary. Moreover, as low resolution produces blocky polygons, for large objects, we resize the concatenated feature to \( 112 \times 112 \), exploiting the bilinear interpolation. We denote this embedding as \( F \) (see Fig. 1).

GCN. The graph of GCN model is denoted as \( G = (V, E) \) where the vertices are \( V \), and the edges are \( E \). Here, we construct \( V \) and \( E \) to represent control points and adjacency matrix, respectively. This adjacency matrix is inferred from the separating network, and GCN model exploits it to exchange and propagate information between nodes in the graph. GCN model shifts \( N \) control points to be located uniformly at the object’s boundary through feature exchange across the \( N \) feature vectors. The feature vectors are the coordinate information of \( N \) control points extracted from vertex embedding. Here, the initialization is an equilateral \( N \)-gon concentric with the image with two edges on both sides at each point, which covers around 75% of the target image. This initial graph converges faster to the target boundary than, say, a rectangle covering the whole image, unless the object is not at the center.

We denote the predicted points set as \( P = \{ p_i = (x_i, y_i) \}_{i=0}^{N-1} \) and the GT points set as \( Q = \{ q_i = (x'_i, y'_i) \}_{i=0}^{N-1} \) where \( N \) is the number of points. The GT points set is uniformly extracted from the boundary of an object. Following [13], we use Graph-ResNet to enable a more precise prediction by exploiting shortcut connections.

The feature vector is defined as

\[
f^1_{(i, y_i)} = \text{concat}\{ F(x_i, y_i), x_i, y_i, \Delta y_{x_i}, \Delta y_{y_i}\},
\]

where \( i \) and \( l \) are the current index and layer respectively. We let \( \Delta y_{x_i} \) and \( \Delta y_{y_i} \) denote human interactive inference, where \( \Delta y_{x_i} \) and \( \Delta y_{y_i} \) represent the distance from the ground truth. If there is no interactive inference, then \( \Delta y_{x_i} \) and \( \Delta y_{y_i} \) are set to be zeros.

With the defined feature vector \( f^1_{(i, y_i)} \), the internal mechanism of the Graph-ResNet follows the steps below:

\[
r^1_{pi} = \sigma (w^0_{pi} f^1_{pi} + \sum_{z \in \mathcal{N}(pi)} w^1_{pi z} f^1_z),
\]

\[
r^{l+1}_{pi} = w^l_{pi} r^l_{pi} + \sum_{z \in \mathcal{N}(pi)} w^l_{pi z} r^l_z,
\]

where \( \mathcal{N}(pi) \) is the set of nodes connected to \( pi \). From \( r^{l+1}_{pi} \) and \( f^1_{pi} \), we have the next feature vector

\[
f^{l+1}_{pi} = \sigma (r^{l+1}_{pi} + f^1_{pi}),
\]

where \( w^l \) and \( w^l \) are the weight matrices.

We adopt a fully connected layer on the last layer of GCN to obtain \( \Delta x_i \) and \( \Delta y_i \) that are shifts needed to move from its current location. Then, the next GCN layer re-extracts the feature vectors from newly shifted location, \( (x'_i + \Delta x_i, y'_i + \Delta y_i) \).

Separating Network. In Split-GCN, the separating network, which consists of self-attention of the Transformer and a fully connected layer, decomposes the graph into connected components. Instead of applying split images (called by the patch) to inputs of the transformer encoder such as ViT, separating network applies extracting feature map generated by CNN using GCN to predict connectivity between nodes and utilize them as inputs of transformer encoder. More specifically, the separating network takes a feature vector \( f \in \mathbb{R}^{N \times (C + 1 \times 2)} \) extracted from the position coordinates predicted by GCN (see Fig. 1) as an input and predicts an adjacency matrix as an output. In particular, a feature vector is the concatenation of feature map \( F \) and the coordinate vector \( \Delta (x, y) \) from Human interactive inference. As mentioned earlier, the feature map \( F \) consists of two vectors: the feature map \( F_i \) and the motion vector. The input feature map \( F_i \) provides semantic perspective information on node similarity. The motion vector provides directional information at particular boundaries. If two points belong to separated components, the direction of those corresponding motion vectors will be opposite to each other, despite their proximity. We employ three feed-forward networks to obtain query, key, and value matrices namely, \( Q(f) \), \( K(f) \), and \( V(f) \). We first multiply \( Q(f) \) by \( K(f) \), and to avoid gradients too small, we divide the result by scale factor \( d_{le} \), which has the dimension of \( K(f) \) (the same dimension as \( Q(f) \)). Finally, we take softmax over it which is then multiplied by \( V(f) \) to obtain the dependencies between all points and point-wise. We can write such self-attention mechanism as

\[
\text{Attn}(Q(f), K(f), V(f)) = \text{softmax} \left( \frac{Q(f) K^T(f)}{\sqrt{d_{le}}} \right) V(f).
\]

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Fig. 3. Results of automatic mode on the cityscapes validation set. From left to right, original images, ground truth images, and our model predictions on automatic mode.

The process after the self-attention mechanism follows the encoder in the vanilla Transformer. Since the input of separating network is fixed, we apply the absolute positional encoding with the fixed sinusoidal embedding [28] to the input of \( Q, K, \) and \( V \) matrices. As seen in Fig. 2, we iterate this process six times, which makes the connectedness (or disconnectedness) of neighboring points become more evident. After the last Transformer layer, we use another fully connected layer with a sigmoid function to predict the adjacency matrix. An entry of our GT adjacency matrix has a value of 0 or 1. If two points are connected to each other, it takes the value 1, otherwise 0. However, as most of these values are 0, it causes an imbalance problem when training the model. We solve this imbalance by giving a difference ratio \( \beta \) of connected and disconnected points in training. In our experiments, we set \( \beta = 0.7 \) to make the loss be more affected by disconnected points. We denote an entry of the predicted adjacency matrix by \( A_{\text{pred}} \) and of the GT adjacency matrix by \( A_{\text{gt}} \in [0, 1] \). Polygon separating loss is defined as

\[
L_{\text{sep}}(A_{\text{pred}}, A_{\text{gt}}) = \begin{cases} 
-\beta \log(A_{\text{pred}}) & \text{if } A_{\text{gt}} = 1 \\
(1 - \beta) \log(1 - A_{\text{pred}}) & \text{otherwise}.
\end{cases}
\]

To acquire the final predicted output, we truncate all other edges except two most probable edges, i.e., each row of the adjacency matrix must have exactly three non-zero entries, including diagonal entries.

**Polygon Separating Loss.** Polygon separating loss finds the disconnected points using the adjacency matrix. An entry of our GT adjacency matrix has a value of 0 or 1. If two points are connected to each other, it takes the value 1, otherwise 0. However, as most of these values are 0, it causes an imbalance problem when training the model. We solve this imbalance by giving a difference ratio \( \beta \) of connected and disconnected points in training. In our experiments, we set \( \beta = 0.7 \) to make the loss be more affected by disconnected points. We denote an entry of the predicted adjacency matrix by \( A_{\text{pred}} \) and of the GT adjacency matrix by \( A_{\text{gt}} \in [0, 1] \). Polygon separating loss is defined as

\[
L_{\text{sep}}(A_{\text{pred}}, A_{\text{gt}}) = \begin{cases} 
-\beta \log(A_{\text{pred}}) & \text{if } A_{\text{gt}} = 1 \\
(1 - \beta) \log(1 - A_{\text{pred}}) & \text{otherwise}.
\end{cases}
\]

To acquire the final predicted output, we truncate all other edges except two most probable edges, i.e., each row of the adjacency matrix must have exactly three non-zero entries, including diagonal entries.

**Point Matching Loss.** We use point matching loss [13] to locate the predicted points around the GT boundary. The point matching loss is defined as follows:

\[
L_{\text{pmatch}}(P, Q) := \min_{\eta \in [0..N-1]} \sum_{i=0}^{N-1} \left\| p_i - q_{(\eta+i)\mod N} \right\|^2.
\]

Using the point matching loss, we get the predicted points near the GT boundary.

Fig. 4. Effect of Split factor \( k \). Our model is trained on the Cityscapes validation set with \( k = 3 \). First and second column show the predictions of three and two disconnected components respectively. The others show the predictions of not occluded single object. Top: Ground truth and Bottom: Our model results.
Comparison of our model to the state-of-the-art models trained with the Cityscapes (fine) or other datasets, MS COCO (COCO) [40] and Mapillary Vistas (MV) [5].

### TABLE 1

| Model          | Train dataset | AP  | Person | Rider | Car | Truck | Bus | Train | Motorcycle | Bicycle |
|----------------|---------------|-----|--------|-------|-----|-------|-----|-------|-------------|---------|
| PANet [4]      | fine + COCO   | 36.4| 41.5   | 33.6  | 58.2| 31.8  | 45.3| 28.7  | 28.2        | 24.1    |
| Axial-DL-L [32]| fine + MV     | 38.1| 34.7   | 30.4  | 55.1| 40.9  | 49.7| 43.5  | 29.0        | 21.7    |
| Pan-DL [33]    | fine + MV     | 39.0| 36.0   | 30.2  | 56.7| 41.5  | 50.8| 42.5  | 30.4        | 23.7    |
| LevelSet R-CNN [34] | fine + COCO | 40.0| 34.3   | 33.9  | 59.0| 37.6  | 49.4| 39.4  | 32.5        | 24.9    |
| PolyTransform [35] | fine + COCO | 40.1| 42.4   | 34.8  | 58.5| 39.8  | 49.9| 41.3  | 30.9        | 23.4    |
| PolygonRNN++ [11] | fine          | 25.5| 29.4   | 21.8  | 48.3| 21.1  | 32.3| 23.7  | 13.6        | 13.6    |
| SShapeNet+ [36] | fine          | 27.3| 29.7   | 23.4  | 46.7| 26.1  | 33.3| 24.8  | 20.3        | 14.1    |
| GMIS [37]      | fine          | 27.3| 31.5   | 25.2  | 42.3| 21.8  | 37.2| 28.9  | 18.8        | 12.8    |
| AdaptIS [38]   | fine          | 32.5| 31.4   | 29.1  | 50.0| 31.6  | 41.7| 39.4  | 24.7        | 12.1    |
| SSAP [39]      | fine          | 32.7| 35.4   | 33.6  | 58.2| 31.8  | 45.3| 28.7  | 22.2        | 17.3    |
| Split-GCN(k=1) | fine          | 29.1| 32.7   | 25.7  | 49.5| 28.0  | 32.7| 21.2  | 22.2        | 17.3    |
| Split-GCN(k=3) | fine          | 29.6| 33.2   | 27.7  | 48.1| 26.6  | 33.7| 25.8  | 22.9        | 19.3    |

### TABLE 2

| Model                  | mIoU | F1px | F2px |
|------------------------|------|------|------|
| PolygonRNN++           | 71.4 | 46.6 | 62.3 |
| PSF-DeepLab            | 73.7 | 47.1 | 62.8 |
| Spline-GCN             | 73.7 | 47.7 | 63.6 |
| Split-GCN(k=1)         | 75.3 | 51.2 | 66.1 |
| Split-GCN(k=3)         | 76.6 | 52.5 | 67.5 |

### TABLE 3

| Split | mIoU | F1px | F2px |
|-------|------|------|------|
| k = 1 | 75.3 | 51.2 | 66.1 |
| k = 2 | 76.1 | 51.9 | 66.8 |
| k = 3 | 76.6 | 52.5 | 67.5 |
| k = 4 | 76.2 | 52.1 | 67.1 |
| k = 5 | 75.9 | 52.0 | 66.5 |

### TABLE 4

| Points | mIoU | F1px | F2px |
|--------|------|------|------|
| 20 pts | 71.3 | 46.4 | 61.9 |
| 30 pts | 74.8 | 50.8 | 64.7 |
| 40 pts | 76.6 | 52.5 | 67.5 |
| 50 pts | 76.3 | 52.2 | 67.4 |
| 60 pts | 76.2 | 52.2 | 67.1 |

### 3.4 Interactive Inference

The goal of interactive inference is for a model to generate refined performance through training, similar to the sophisticated modifications generated by humans. When a person modifies the vertices predicted by the model, we assume that the most efficient approach is selecting the furthest vertex of the prediction and relocating it to the GT coordinate.

To implement this, we first calculate the difference of the most distant vertex in the predicted points set $P$ and GT points set $Q$ using the Manhattan distance. Then, we substitute $\Delta_1$ for, and $\Delta_2$ for, the difference of the feature vector with the calculated difference. This may affect all nodes involved in the message passing part of GCN. Consequently, by propagating the calculated difference information to connected adjacent vertices, our model adjusts the coordinates of the connected adjacent vertices.

Point $L_2$ Loss. We need to know which of the predicted points are close to the separating points. As explained in GT construction part, a split happens only at a multiple of $m$. With the following point $L_2$ loss, our network learns this rule:

$$L_2(P, Q) := \min_p \sum_{i=0}^{N-1} \| p_i - q_i \|_2. \quad (9)$$

In our experiment, after training 10 epochs only with the point matching loss, the separating network are trained 100 epochs using both the point $L_2$ loss and the point matching loss.

Motion Vector Loss. Here, we consider the motion vector loss as in [1], [3]. The motion vector loss can be simply regarded as a mean square angular. This angular nature of the motion vector loss enables the model to accurately predict points and be located on the object’s exact boundary. Motion vector loss is defined as

$$I_{\text{motion}}(\theta) := \sum_{i,j \in N} \left(\cos^{-1} \left( \frac{V_i(i, j) \cdot \bar{U}_{gt}(i, j)}{|V_i(i, j)|} \right) \right)^2. \quad (10)$$

square angular. This angular nature of the motion vector loss enables the model to accurately predict points and be located on the object’s exact boundary. Motion vector loss is defined as

$$I_{\text{motion}}(\theta) := \sum_{i,j \in N} \left(\cos^{-1} \left( \frac{V_i(i, j) \cdot \bar{U}_{gt}(i, j)}{|V_i(i, j)|} \right) \right)^2. \quad (10)$$

Comparison of the average number of clicks between our model and MBOX [13].
4 EXPERIMENTAL RESULTS

We evaluate our model in two aspects: in-domain and cross-domain annotation settings. For in-domain experiments, we use the Cityscapes dataset as the main benchmark to compare our model’s performance. For cross-domain experiments, we assess the generalization ability of our trained model (with the Cityscapes dataset) on several different cross-domain datasets: KITTI, Rooftop, ADE20 K, and Card.MR. As assumed in [11], [12], [13], we suppose that the users provide the ground-truth boxes and use them as the input to the model.

Image Encoder. Following the method from PolygonRNN++ [11], we use the ResNet-50 [29] as the backbone of the image encoder.

Training Details. We train the model with the structure shown in Fig. 1. We use ImageNet [41] pretrained weights only for the image encoder in the feature extraction network. For training, we use point matching loss, point L2 loss, motion vector loss, and polygon separating loss in an end-to-end manner. We set the initial adjacency matrix of GCN to a single block N-gon as mentioned in Section 3.3. Experiments regarding the value of k and N are in Section 4.1. For the most experiments, we set the split factor k = 3 and N = 40.

We train our model on a single NVIDIA Titan XP GPU for 150 epochs. We use the batch size of 24 and the initial learning rate and weight decay of 1e-4. Moreover, the decay on the learning rate is 1/10 at every 50 epochs. During training, the input image is cropped using a GT bounding box and appropriately resized with a resolution of 224 × 224. We use standard augmentation techniques (random rotation, horizontal flip, and scaling) on the data.

State-of-the-Art & Baseline Models. We first compare our model to state-of-the-art models from the Cityscapes leaderboard to demonstrate the competitiveness of our model. We also compare our model to the baseline models. The baseline models are from competitive models using the similar approaches: Curve-GCN (Spline-GCN and Polygon-GCN) [13], PolygonRNN++ [11], and PSP-DeepLab [42]. We choose PSP-DeepLab using spatial pyramid pooling [43] because it has similar encoder architecture as our model. We also choose Curve-GCN and PolygonRNN++, which are the latest models using polygon-based methodology as our model.

Metrics. We use three metrics to evaluate our model: (1) We use two Average Precision (AP) metrics to compare the performance of our model to the state-of-the-art models. First, AP is calculated by Intersection-over-Union (IoU) with an increase of 0.05 from 0.5 to 0.95 overlap thresholds. Second, AP50 is calculated by IoU with a 0.5 overlap threshold, which follows the Cityscapes metrics. (2) We use the mean IoU (mIoU) metric to compare the performance of our model to the baseline models. For each predicted instances and ground truth, we calculated the mIoU by averaging the IoU of every class. (3) As mIoU alone is insufficient to measure the inaccuracies on the object boundary, we use the boundary F score [44] as another metric to evaluate the performance on the object boundary more precisely. The boundary F score measures the performance of the precision and recall on a given boundary width of 1 pixel and 2 pixels, denoted as F1px and F2px, respectively.

4.1 In-Domain Annotation

This section compares the performance of our model with state-of-the-art models and baseline models. We evaluate automatic and interactive modes using the Cityscapes dataset. We chose the Cityscapes dataset as it is one of the most comprehensive benchmarks available to evaluate instance segmentation performance. The Cityscapes dataset instance segmentation consists of eight classes. There are 2,975,500, and 1,525 for training/validation/test images.

Automatic Mode. We perform two experiments in automatic mode. First, we compare the performances of our model and the state-of-the-art models on the Cityscapes test set. Here, we use Faster R-CNN [45] as an object detection model, which is one of the commonly used detection models. According to the results in Table 1, our model presents 29.6 AP, which is very competitive compared to the state-of-the-art models using the pixel-wise approach. Our score shows the highest among the models, only trained with the Cityscapes dataset (fine), followed by SSAP [39] and AdapIS [38]. Moreover, our model shows a 16% increase when compared to PolygonRNN++ [11], a similar model applying the polygon-based approach.

Second, we compare our model with the baseline models using the Cityscapes validation set. The set of mIoU, F1px, and F2px are reported in Table 2. We find that our model presents the highest metrics of 76.6 mIoU, 52.5 F at 1 pixel, and 67.5 F at 2 pixels, which are even better than Spline-GCN, which has the highest performance among the baseline models. As shown in Figs. 3 and 4, Split-GCN exhibits satisfying results even when the object is disconnected.

Points & Split. We evaluate how our model’s accuracy varies according to control points N and split factor k on the Cityscapes validation set. The results of the first experiment about the split factor k are in Table 3. In Table 3, k = 3 shows the best performance. Interestingly, when the value of k gets bigger than 3, the performance decreases. We think that this is because the model concentrates more on locating the disconnected components rather than deforming the object boundary. The second experiment evaluates the performance according to the different number of control points N. As reported in Table 4, we increase the control points by 10 units, where the range of N is 20 ≤ N ≤ 60. From the results in Table 4, we can see that the best performance is where N = 40.

Ablation Study. Using the Cityscapes validation set, we evaluate how our model’s performance varies depending on our proposed
motion vector branch, separating network, and interactive learning. Based on the experiment results on Tables 3 and 4, we set $k = 3$ and $N = 40$ as our best performing model. Table 5 summarizes the results. The performance increases by up to 8.2% from the original GCN as we apply the motion vector branch and separating network, and when interactive learning is applied, the performance increases by an additional 2.2%.

Interactive Mode. We test how our model performs when human interaction involves using the Cityscapes validation set (see Table 6). In interactive mode experiments, the annotator continuously corrects the model’s predicted vertices until meeting two independent conditions. The first condition is either reaching the performance of 1.0 mIoU or reaching the maximum performance (see Fig. 5). The second condition is whether locating the vertices more than five times to correct the segmentation (see Fig. 6).

In Fig. 5, our model reaches higher mIoU with fewer clicks than the two baseline models, PolygonRNN++ and Spline-GCN. This result implies that the annotator generates the ground-truth much quicker. In particular, our model improves about 1.1% mIoU per click than Spline-GCN on average. We further compare the interactive mode with the automatic mode by each class to demonstrate human-in-the-loop performance in Fig. 6. Consequently, our model’s interactive mode improves mIoU up to 12.2% than the automatic mode with only five or fewer clicks.

### 4.2 Cross-Domain Annotation

In this section, we verify the generalization ability of our model. The generalization ability is crucial for the utilization of image annotation in various fields. We also demonstrate how quickly our model can be applied when fine-tuned with only 10% of data from the new domain. For evaluation, we use our model trained on the Cityscapes dataset following [11], [13]. We analyze our model’s performance on four different data domains: KITTI (urban street scenes), Rooftop (aerial images of a rural scene), ADE20 K (general scene images), and Card.MR (medical images).

#### Qualitative Results

Table 7 shows the results of comparing our model with the baseline models on the cross-domain datasets. In the KITTI and ADE, Split-GCN ($k = 3$) outperforms when compared to $k = 1$ or the other models. In the Rooftop dataset, Split-GCN ($k = 1$) shows the performance at most 2.4% higher than Spline-GCN, and in Card.MR, 14.1% higher than PolygonRNN++.

especially for these two datasets, Split-GCN with $k = 3$ generates even better performance with 1.4% higher than $k = 1$. We demonstrate that Split-GCN adapts well to the new domains, even with limited training.

### 5 Conclusion

In this paper, we proposed Split-GCN, which predicts points to be located uniformly at the object boundary. Our model locates the disconnected location and deforms the object boundary simultaneously to make sophisticated prediction, even when the object is disconnected. Moreover, we suggested the motion vector branch, which predicts the object boundary more precisely by informing destination information to the model. Our experiments confirmed the competitive performance with 29.6 AP, 76.6 mIoU, 52.5 F1px, and 67.5 F2px scores on the Cityscapes dataset. Our model also demonstrates a compelling generalization ability by comprehending the contexts of new domain datasets. Future research should develop the technique to dynamically change the number of control points for a more detailed prediction of an object boundary.

### Acknowledgments

Namgil Kim and Barom Kang contributed equally to this work.

### References

[1] M. Bai and R. Urtasun, “Deep watershed transform for instance segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 2858–2866.

[2] J. Gao, Z. Wang, J. Xuan, and S. Fidler, “Beyond fixed grid: Learning geometric image representation with a deformable grid,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 108–125.

[3] P. Huang, B. Shuai, J. Liang, and G. Wang, “Deep level sets for salient object detection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 540–549.

[4] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path aggregation network for instance segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 8759–8768.

[5] G. Neuhold, T. Ollmann, S. R. Bulo, and P. Kontschieder, “The mapillary vistas dataset for semantic understanding of street scenes,” in Proc. Int. Conf. Comput. Vis., 2017, pp. 5000–5009.

[6] Y. Boykov and V. Kolmogorov, “An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision,” in Proc. IEEE Int. Workshop Energy Minimization Methods Comput. Vis. Pattern Recognit., 2001, pp. 359–374.

[7] C. Rother, V. Kolmogorov, and A. Blake, “GrabCut” interactive foreground extraction using iterated graph cuts,” ACM Trans. Graph., vol. 23, no. 3, pp. 309–314, 2004.

[8] K.-K. Mannis, S. Caelles, J. Pont-Tuset, and L. V. Gool, “Deep extreme cut: From extreme points to object segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 616–625.

[9] Z. Wang, D. Acuna, H. Ling, A. Kar, and S. Fidler, “Object instance annotation with deep extreme level set evolution,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 7581–7589.

[10] N. Xu, B. Price, S. Cohen, J. Yang, and T. S. Huang, “Deep interactive object selection,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 373–381.

[11] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2012, pp. 3354–3361.

[12] X. Sun, C. M. Christoudias, and P. Fua, “Free–shape polygonal object localization,” in Proc. Eur. Conf. Comput. Vis., 2014, pp. 317–332.

[13] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, “Scene parsing through ADE20K dataset,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 633–641.

[14] A. Suiresiaputra et al., “A collaborative resource to build consensus for automated left ventricular segmentation of cardiac MR images,” Med. Image Anal., vol. 18, no. 4, pp. 624–643.

[15] Z. Lin, Z. Zhang, L.-Z. Chen, M.-M. Cheng, and S.-P. Lu, “Interactive image segmentation with first click attention,” in Proc. IEEE/IVC Conf. Comput. Vis. Pattern Recognit., 2020, pp. 13339–13348.

[16] S. Majumder and A. Yao, “Content-aware multi-level guidance for interactive instance segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 11602–11611.

[17] M. Mahadevan, P. Voigtlaender, and B. Leibe, “Iteratively trained interactive segmentation,” 2018, arXiv:1805.04398.

[18] S. Zhang, J. H. Liew, Y. Wei, S. Wei, and Y. Zhao, “Interactive object segmentation with inside-outside guidance,” in Proc. IEEE/ACCV Conf. Comput. Vis. Pattern Recognit., 2020, pp. 12231–12241.

[19] Z. Lin, Z.-P. Duan, Z. Zhang, C.-L. Guo, and M.-M. Cheng, “FocusCut: Dividing an object to a focus view in interactive segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2022, pp. 2627–2636.

[20] V. Caselles, R. Kimmel, and G. Sapiro, “Geodesic active contours,” Int. J. Comput. Vis., vol. 22, no. 1, pp. 61–79, 1997.

[21] D. Marcos et al., “Learning deep structured active contours end-to-end,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 8877–8885.

[22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, “Gated graph sequence neural networks,” 2015, arXiv:1511.05933.

[23] L. Sun, Y.-W. Tai, and K. Huang, “Deep occlusion-aware instance segmentation with overlapping bilayers,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2021, pp. 4019–4028.

[24] A. Vasanwani et al., “Attention is all you need,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2017, pp. 6008–6009.

[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[26] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” 2020, arXiv:2010.11929.
[32] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen, “Axial-DeepLab: Stand-alone axial-attention for panoptic segmentation,” 2020, arXiv:2003.07853.

[33] B. Cheng et al., “Panoptic-DeepLab: A simple, strong, and fast baseline for bottom-up panoptic segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 12475–12485.

[34] N. Homayounfar, Y. Xiong, J. Liang, W.-C. Ma, and R. Urtasun, “Levelset R-CNN: A deep variational method for instance segmentation,” 2020, arXiv:2007.15629.

[35] J. Liang, N. Homayounfar, W.-C. Ma, Y. Xiong, R. Hu, and R. Urtasun, “PolyTransform: Deep polygon transformer for instance segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 9131–9140.

[36] B. R. Kang, H. Lee, K. Park, H. Ryu, and H. Y. Kim, “BshapeNet: Object detection and instance segmentation with bounding shape masks,” Pattern Recognit. Lett., vol. 131, pp. 449–455, 2020.

[37] Y. Liu et al., “Affinity derivation and graph merge for instance segmentation,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 686–703.

[38] K. Sofiuk, O. Barinova, and A. Konushin, “AdaptIS: Adaptive instance selection network,” in Proc. IEEE Int. Conf. Comput. Vis., 2019, pp. 7355–7363.

[39] N. Gao et al., “SSAP: Single-shot instance segmentation with affinity pyramid,” in Proc. IEEE Int. Conf. Comput. Vis., 2019, pp. 642–651.

[40] T.-Y. Lin et al., “Microsoft COCO: Common objects in context,” in Proc. Eur. Conf. Comput. Vis., 2014, pp. 740–755.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2009, pp. 248–255.

[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Semantic image segmentation with deep convolutional nets and fully connected CRFs,” 2014, arXiv:1412.7062.

[43] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017, pp. 6230–6239.

[44] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung, “A benchmark dataset and evaluation methodology for video object segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 724–732.

[45] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2015, pp. 91–99.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.