Efficient Interactive Annotation of Segmentation Datasets with Polygon-RNN++

David Acuna[1,3,*] Huan Ling[1,2*] Amlan Kar[1,2,*] Sanja Fidler[1,2]
[1]University of Toronto [2]Vector Institute [3]NVIDIA[†]
{davidj, linghuan, amlan, fidler}@cs.toronto.edu

Abstract

Manually labeling datasets with object masks is extremely time consuming. In this work, we follow the idea of Polygon-RNN [4] to produce polygonal annotations of objects interactively using humans-in-the-loop. We introduce several important improvements to the model: 1) we design a new CNN encoder architecture, 2) show how to effectively train the model with Reinforcement Learning, and 3) significantly increase the output resolution using a Graph Neural Network, allowing the model to accurately annotate high-resolution objects in images. Extensive evaluation on the Cityscapes dataset [8] shows that our model, which we refer to as Polygon-RNN++, significantly outperforms the original model in both automatic (10% absolute and 16% relative improvement in mean IoU) and interactive modes (requiring 50% fewer clicks by annotators). We further analyze the cross-domain scenario in which our model is trained on one dataset, and used out of the box on datasets from varying domains. The results show that Polygon-RNN++ exhibits powerful generalization capabilities, achieving significant improvements over existing pixel-wise methods. Using simple online fine-tuning we further achieve a high reduction in annotation time for new datasets, moving a step closer towards an interactive annotation tool to be used in practice.

1. Introduction

Detailed reasoning about structures in images is a necessity for numerous computer vision applications. For example, it is crucial in the domain of autonomous driving to localize and outline all cars, pedestrians, and miscellaneous static and dynamic objects [1, 18, 12]. For mapping, there is a need to obtain detailed footprints of buildings and roads from aerial/satellite imagery [34], while medical/healthcare domains require automatic methods to precisely outline cells, tissues and other relevant structures [15, 11].

Neural networks have proven to be an effective way of inferring semantic [6, 19] and object instance segmentation information [12, 18] in challenging imagery. It is well known that the amount and variety of data that the networks see during training drastically affects their performance at run time. Collecting ground truth instance masks, however, is an extremely time consuming task, typically requiring human annotators to spend 20-30 seconds per object in an image.

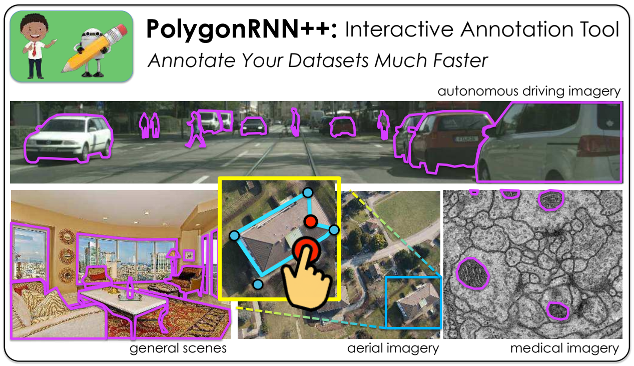
PolygonRNN++: Interactive Annotation Tool
Annotate Your Datasets Much Faster
autonomous driving imagery
general scenes
aerial imagery
medical imagery

Figure 1: We introduce Polygon-RNN++, an interactive object annotation tool. We make several advances over [4], allowing us to annotate objects faster and more accurately. Furthermore, we exploit a simple online fine-tuning method to adapt our model from one dataset to efficiently annotate novel, out-of-domain datasets.

To this end, in [4], the authors introduced Polygon-RNN, a conceptual model for semi-automatic and interactive labeling to help speed up object annotation. Instead of producing pixel-wise segmentation of an object as in existing interactive tools such as Grabcut [30], [4] predicts the vertices of a polygon that outlines the object. The benefits of using a polygon representation are three-fold, **1)** it is sparse (only a few vertices represent regions with a large number of pixels), **2)** it is easy for an annotator to interact with, and **3)** it allows for efficient interaction, typically requiring only a few corrections from the annotator [4]. Using their model, the authors have shown high annotation speed-ups on two autonomous driving datasets [8, 10].

In this work, we introduce several improvements to the Polygon-RNN model. In particular, we **1)** make a few changes to the neural network architecture, **2)** propose a better learning algorithm to train the model using reinforcement learning, and **3)** show how to significantly increase the out-

[*] authors contributed equally
[†] work done when D.A. was at UofT

put resolution of the polygon (one of the main limitations of the original model) using a Graph Neural Network [31, 17]. We analyze the robustness of our approach to noise, and its generalization capabilities to out-of-domain imagery.

In the fully automatic mode (no annotator in the loop), our model achieves significant improvements over the original approach, outperforming it by 10% mean IoU on the Cityscapes dataset [8]. In interactive mode, our approach requires 50% fewer clicks as compared to [4]. To demonstrate generalization, we use a model trained on the Cityscapes dataset to annotate a subset of a scene parsing dataset [41], aerial imagery [33], and two medical datasets [15, 11]. The model significantly outperforms strong pixel-wise labeling baselines, showcasing that it inherently learns to follow object boundaries, thus generalizing better. We further show that a simple online fine-tuning approach achieves high annotation speed-ups on out-of-domain dataset annotation.

2. Related Work

Interactive annotation. Since object instance segmentation is time consuming to annotate manually, several works have aimed at speeding up this process using interactive techniques. In seminal work, [2] used scribbles to model the appearance of foreground/background, and performed segmentation via graph-cuts [3]. This idea was extended by [20] to use multiple scribbles on both the object and background, and was demonstrated in annotating objects in videos. GrabCut [30] exploited 2D bounding boxes provided by the annotator, and performed pixel-wise foreground/background labeling using EM. [25] combined GrabCut with CNNs to annotate structures in medical imagery. Most of these works operate on the pixel level, and typically have difficulties in cases where foreground and background have similar color.

In [4], the authors used polygons instead. The main power of using such a representation is that it is sparse; only a few vertices of a polygon represent large image regions. This allows the user to easily introduce corrections, by simply moving the wrong vertices. An RNN also effectively captures typical shapes of objects as it forms a non-linear sequential representation of shape. This is particularly important in ambiguous regions, ie shadows and saturation, where boundaries cannot be observed. We follow this line of work, and introduce several important modifications to the architecture and training. Furthermore, the original model was only able to make prediction at a low resolution (28 × 28), thus producing blocky polygons for large objects. Our model significantly increases the output resolution (112 × 112).

Object instance segmentation. Most approaches to object instance segmentation [16, 29, 39, 37, 21, 22, 12, 1, 18] operate on the pixel-level. Many rely on object detection, and use a convnet over a box proposal to perform the labeling [21, 22, 12]. In [38, 33], the authors produce a polygon around an object. These approaches first detect boundary fragments, followed by finding an optimal cycle linking the boundaries into object regions. [9] produce superpixels in the form of small polygons which are further combined into an object. Here, as in [4] we use neural networks to produce polygons, and in particular tackle the interactive labeling scenario which has not been explored in these works.

3. Polygon-RNN++

In this section, we introduce Polygon-RNN++. Following [4], our model expects an annotator to provide a bbox around the object of interest. We extract an image crop enclosed by the 15% enlarged box. We use a CNN+RNN architecture as in [4], with a CNN serving as an image feature extractor, and the RNN decoding one polygon vertex at a time. Output vertices are represented as locations in a grid.

The full model is depicted in Fig. 2. Our redesigned encoder produces image features that are used to predict the first vertex. The first vertex and the image features are then fed to the recurrent decoder. Our RNN exploits visual attention at each time step to produce polygon vertices. A learned evaluator network selects the best polygon from a set of candidates proposed by the decoder. Finally, a graph neural network re-adjusts polygons, augmented with additional vertices, at a higher resolution.

This model naturally incorporates a human in the loop, allowing the annotator to correct an erroneously predicted vertex. This vertex is then fed back to the model, helping the model to correct its prediction in the next time steps.

3.1. Residual Encoder with Skip Connections

Most networks perform repeated down-sampling operations at consecutive layers of a CNN, which impacts the effective output resolution in tasks such as image segmentation [6, 23]. In order to alleviate this issue, we follow [7] and modify the ResNet-50 architecture [13] by reducing the stride of the network and introducing dilation factors. This allows us to increase the resolution of the output feature map without reducing the receptive field of individual neurons. We also remove the original average pooling and FC layers.

We further add a skip-layer architecture [19, 40] which aims to capture both, low-level details such as edges and corners, as well as high-level semantic information. In [4], the authors perform down-sampling in the skip-layer architecture, built on top of VGG, before concatenating the features from different layers. Instead, we concatenate all the outputs of the skip layers at the highest possible resolution, and use a combination of conv layers and max-pooling operations to obtain the final feature map. We employ conv filters with a kernel size of 3 × 3, batch normalization [14] and ReLU non-linearities. In cases where the skip-connections have different spatial dimensions, we use bilinear upsampling before concatenation. The architecture is shown in Fig. 4. We refer to the final feature map as the *skip features*.

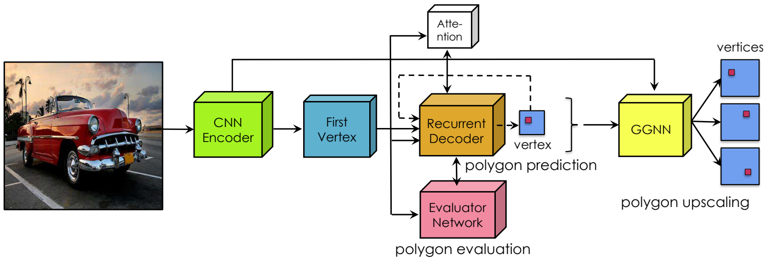

Figure 2: Polygon-RNN++ model (figures best viewed in color)

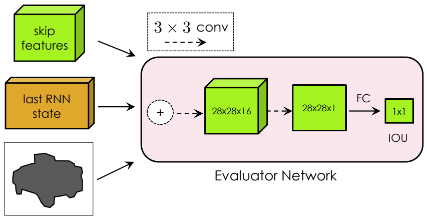

Figure 3: Evaluator Network predicting the quality of a polygon output by the RNN decoder

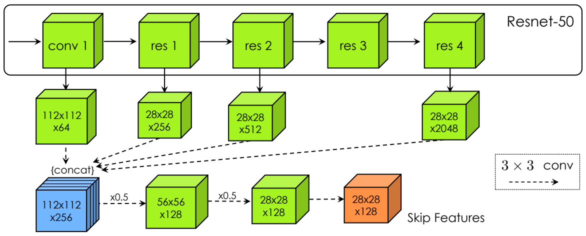

Figure 4: Residual Encoder architecture. Blue tensor is fed to GNN, while the orange tensor is input to the RNN decoder.

3.2. Recurrent Decoder

As in [4], we use a Recurrent Neural Network to model the sequence of 2D vertices of the polygon outlining an object. In line with previous work, we also found that the use of Convolutional LSTM [36] is essential: **1)** to preserve spatial information and **2)** to reduce the number of parameters to be learned. In our RNN, we further add an attention mechanism, as well as predict the first vertex within the same network (unlike [4] which has two separate networks).

We use a two-layer ConvLTSM with a 3×3 kernel with 64 and 16 channels, respectively. We apply batch norm [14] at each time step, without sharing mean/variance estimates across time steps. We represent our output at time step t as a one-hot encoding of $(D \times D) + 1$ elements, where D is the resolution at which we predict. In our experiments, D is set to 28. The first $D \times D$ dimensions represent the possible vertex positions and the last dimension corresponds to the end-of-seq token that signals that the polygon is closed.

Attention Weighted Features: In our RNN, we exploit a mechanism akin to attention. In particular, at time step t, we compute the weighted feature map as,

$$\begin{aligned} \alpha_t &= \text{softmax}(f_{att}(x, f_1(h_{1,t-1}), f_2(h_{2,t-1}))) \\ F_t &= x \circ \alpha_t \end{aligned} \tag{1}$$

where $\circ$ is the Hadamard product, x is the *skip feature* tensor, and $h_{1,t}$, $h_{2,t}$ are the hidden state tensors from the two-layer ConvLSTM. Here, f_1 and f_2 map $h_{1,t}$ and $h_{2,t}$ to $\mathbb{R}^{D \times D \times 128}$ using one fully-connected layer. f_{att} takes the sum of its inputs and maps it to $D \times D$ through a fully connected layer, giving one "attention" weight per location.

Intuitively, we use the previous RNN hidden state to gate certain locations in the image feature map, allowing the RNN to focus only on the relevant information in the next time step. The gated feature map F_t is then concatenated with one-hot encodings of the two previous vertices y_{t-1}, y_{t-2} and the first vertex y_0, and passed to the RNN at time step t.

First Vertex: Given a previous vertex and an implicit direction, the next vertex of a polygon is always uniquely defined, except for the first vertex. To tackle this problem, the authors in [4] treated the first vertex as a special case and used an additional architecture (trained separately) to predict it. In our model, we add another branch from the skip-layer architecture, constituting of two layers each of dimension $D \times D$. Following [4], the first layer predicts edges, while the second predicts the vertices of the polygon. At test time, the first vertex is sampled from the final layer of this branch.

3.3. Training using Reinforcement Learning

In [4], the authors trained the model using the cross-entropy loss at each time step. However, such training has two major limitations: **1)** MLE over-penalizes the model (for example when the predicted vertex is on an edge of the GT polygon but is not one of the GT vertices), and **2)** it optimizes a metric that is very different from the final evaluation metric (i.e. IoU). Further, the model in [4] was trained following a typical training regime where the GT vertex is fed to the next time step instead of the model's prediction. This training regime, called teacher forcing creates a mismatch between training and testing known as the exposure bias problem [26].

In order to mitigate these problems, we only use MLE training as an initialization stage. We then reformulate the polygon prediction task as a reinforcement learning problem and fine-tune the network using RL. During this phase, we let the network discover policies that optimize the desirable, yet non-differentiable evaluation metric (IoU) while also exposing it to its own predictions during training.

3.3.1 Problem formulation

We view our recurrent decoder as a sequential decision making agent. The parameters θ of our encoder-decoder architecture define its policy p_θ for selecting the next vertex v_t. At the end of the sequence, we obtain a reward r. We compute our reward as the IoU between the mask enclosed by the generated polygon and the ground-truth mask m. To maximize the expected reward, our loss function becomes

$$L(\theta) = -\mathbb{E}_{v^s \sim p_\theta}[r(v^s, m)] \tag{2}$$

where $v^s = (v_1^s, ..., v_T^s)$, and v_t^s is the vertex sampled from the model at time t. Here, $r = \text{IoU}(\text{mask}(v^s), m)$.

3.3.2 Self-Critical Training with Policy Gradients

Using the REINFORCE trick [35] to compute the gradients of the expectation, we have

$$\nabla L(\theta) = -\mathbb{E}_{v^s \sim p_\theta}[r(v_s, m)\nabla \log p_\theta(v^s)] \tag{3}$$

In practice, the expected gradient is computed using simple Monte-Carlo sampling with a single sample. This procedure is known to exhibit high variance and is highly unstable without proper context-dependent normalization. A natural way to deal with this is to use a learned *baseline* which is subtracted from the reward. In this work, we follow the self-critical method [28] and use the test-time inference reward of our model as the baseline. Accordingly, we reformulate the gradient of our loss function to be

$$\nabla L(\theta) = -[(r(v^s, m) - r(\hat{v}^s, m))\nabla \log p_\theta(v^s)] \tag{4}$$

where $r(\hat{v}^s, m)$ is the reward obtained using greedy decoding. To control the level of randomness in the vertices explored by the model, we introduce a temperature parameter τ in the softmax of the policy. This ensures that the sampled vertices lead to well behaved polygons. We set $\tau = 0.6$.

3.4. Evaluator Network

Smart choice of the first vertex is crucial as it biases the initial predictions of the RNN, when the model does not have a strong history to reason about the object to annotate. This is particularly important in cases of occluding objects. It is desirable for the first vertex to be far from the occlusion boundaries so that the model follows the object of interest. In RNNs, beam search is typically used to prune off improbable sequences. However, since classical beam search uses log probabilities to evaluate beams, it does not directly apply to our model which aims to optimize IoU. A point on an occlusion boundary generally exhibits a strong edge and thus would have a high log probability during prediction, reducing the chances of it being pruned by beam search.

In order to solve this problem, we propose to use an *evaluator network* at inference time, aiming to effectively choose among multiple candidate polygons. Our evaluator network takes as input the *skip features*, the last state tensor of the ConvLSTM, and the predicted polygon, and tries to estimate its quality by predicting its IoU with GT. The network has two 3×3 convolutional layers followed by a FC layer, forming another branch in the model. Fig. 3 depicts its architecture. While the full model can be trained end-to-end during the RL step, we choose to train the evaluator network separately after the RL fine-tuning has converged.

During training, we minimize the mean squared error

$$L(\phi) = [p(\phi, v^s) - \mathrm{IoU}(m_{v^s}, m)]^2 \tag{5}$$

where p is the network's predicted IoU, m_{v^s} is the mask for the sampled vertices and m is the ground-truth mask. To ensure diversity in the vertices seen, we sample polygons with $\tau = 0.3$. We emphasize that we do not use this network as a baseline estimator during the RL training step since we found that the self-critical method produced better results.

Inference: At test time, we take K top scoring first vertex predictions. For each of these, we generate polygons via classical beam-search (using log prob with a beam-width B). This yields K different polygons, one for each first vertex candidate. We use the evaluator network to choose the best polygon. In our experiments, we use $K = 5$. While one could use the evaluator network instead of beam-search at each time step, this would lead to impractically long inference times. Our faster full model (using $B = K = 1$) runs at 295ms per object instance on a Titan XP.

Annotator in the Loop: We follow the same protocol as in [4], where the annotator corrects the vertices in sequential order. Each correction is then fed back to the model, which re-predicts the rest of the polygon.

3.5. Upscaling with a Graph Neural Network

The model described above produces polygons at a resolution of $D \times D$, where we set D to be 28 to satisfy memory bounds and to keep the cardinality of the output space amenable. In this section, we exploit a Gated Graph Neural Network (GGNN) [17], in order to generate polygons at a much higher resolution. GNN has been proven efficient for semantic segmentation [24], where it was used at pixel-level.

Note that when training the RNN decoder, the GT polygons are simplified at their target resolution (co-linear vertices are removed) to alleviate the ambiguity of the prediction task. Thus, at a higher resolution, the object may have additional vertices, thus changing the topology of the polygon.

Our upscaling model takes as input the sequence of vertices generated by the RNN decoder. We treat these vertices as nodes in a graph. To model finer details at a higher resolution, we add a node in between two consecutive nodes, with its location being in the middle of their corresponding edge. We also connect the last and the first vertex, effectively converting the sequence into a cycle. We connect neighboring nodes using 3 different types of edges, as shown in Fig. 5.

GGNN defines a propagation model that extends RNNs to arbitrary graphs, effectively propagating information between nodes, before producing an output at each node. Here, we aim to predict the relative offset of each node (vertex) at a higher resolution. The model is visualized in Fig. 5.

Gated Graph Neural Network: For completeness, we briefly summarize the GGNN model [17]. GGNN uses a graph $\{V, E\}$, where V and E are the sets of nodes and edges, respectively. It consists of a propagation model performing message passing in the graph, and an output model for prediction tasks. We represent the initial state of a node v as x_v and the hidden state of node v at time step t as h_v^t. The basic

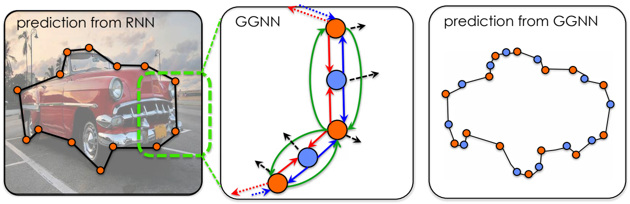

Figure 5: GGNN model: We take predicted polygon from RNN (orange vertices), and add midpoints (in blue) between every pair of consecutive vertices (orange). Our GGNN has three types of edges (red, blue, green), each having its own weights for message propagation. Black dashed arrows pointing out of the nodes (middle diagram) indicate that the GGNN aims to predict the relative location for each of the nodes (vertices), after completing propagation. Right is the high resolution polygon output by the GGNN.

recurrence of the propagation model is

$$\begin{aligned} h_v^0 &= [x_v^\top, 0]^\top \\ a_v^t &= A_{v:}^\top [h_1^{t-1\top}, ..., h_{|V|}^{t-1\top}]^\top + b \\ h_v^t &= f_{GRU}(h_v^{t-1}, a_v^t) \end{aligned} \tag{6}$$

The matrix $A \in \mathbb{R}^{|V| \times 2N|V|}$ determines how the nodes in the graph communicate with each other, where N represents the number of different edge types. Messages are propagated for T steps. The output for node v is then defined as

$$\begin{aligned} h_v &= \tan(f_1(h_v^T)) \\ \text{out}_v &= f_2(h_v) \end{aligned} \tag{7}$$

Here, f_1 and f_2 are MLP, and out_v is v's desired output.

PolygonRNN++ with GGNN: To get observations for our GGNN model, we add another branch on top of our skip-layer architecture, specifically, from the $112 \times 112 \times 256$ feature map (marked in blue in Fig. 4). We exploit a conv layer with 256 filters of size 15×15, giving us a feature map of size $112 \times 112 \times 256$. For each node v in the graph, we extract a $S \times S$ patch around the scaled (v_x, v_y) location, giving us the observation vector x_v. After propagation, we predict the output of a node v as a location in a $D' \times D'$ spatial grid. We make this grid relative to the location (v_x, v_y), rendering the prediction task to be a relative displacement with respect to its initial position. This prediction is treated as a classification task and the model is trained with the cross entropy loss. In particular, in order to train our model, we first take predictions from the RNN decoder, and correct a wrong prediction if it deviates from the ground-truth vertex by more than a threshold. The targets for training our GGNN are then the relative displacements of each of these vertices with respect to their corresponding ground-truth vertices.

Implementation details: We set S to 1 and D' to 112. While our model supports much higher output resolutions, we found that larger D' did not improve results. The hidden state of the GRU in the GGNN has 256 dimensions. We use $T = 5$ propagation steps. In the output model, f_1 is a 256×256 FC layer and f_2 is a $256 \times 15 \times 15$ MLP. In training, we take the predictions from the RNN, and replace vertices with GT vertices if they deviate by more than 3 cells.

3.6. Annot. New Domains via Online Fine-Tuning

We now also tackle the scenario in which our model is trained on one dataset, and is used to annotate a novel dataset. As the new data arrives, the annotator uses our model to annotate objects and corrects wrong predictions when necessary. We propose a simple approach to fine-tune our model in such a scenario, in an online fashion.

Let us denote C as the number of chunks the new data is divided into, CS as the chunk size, N_{EV} as the number of training steps for the evaluator network and N_{MLE}, N_{RL} as the number of training steps for each chunk with MLE and RL, respectively. Our online fine-tuning is described in Algorithm 1 where *PredictAndCorrect* refers to the (simulated) annotator in the loop. Because we train on corrected data, we smooth our targets for MLE training with a manhattan distance transform truncated at distance 2.

Algorithm 1: Online Fine Tuning on New Datasets

```
bestPoly = cityscapesPoly;
while currChunk in $(1..C)$ do
    rawData = readChunk(currChunk);
    data = $PredictAndCorrect$(rawData, bestPoly);
    data += $SampleFromSeenData(CS)$;
    newPoly = $Train_{MLE}$(data, $N_{MLE}$, bestPoly);
    newPoly = $Train_{RL}$(data, $N_{RL}$, newPoly);
    newPoly = $Train_{EV}$(data, $N_{EV}$, newPoly);
    bestPoly = newPoly;
end
```

4. Experimental Results

In this section, we provide an extensive evaluation of our model. We report both automatic and interactive instance annotation results on the challenging Cityscapes dataset [8] and compare with strong pixel-wise methods. We then characterize the generalization capability of our model with evaluation on the KITTI dataset [10] and four out-of-domain datasets spanning general scenes [41], aerial [33], and medical imagery [15, 11]. Finally, we evaluate our online fine-tuning scheme, demonstrating significant decrease in annotation time for novel datasets. Note that as in [4], we assume that user-provided ground-truth boxes around objects are given. We further analyze robustness of our model to noise with respect to these boxes, mimicking noisy annotators.

4.1. In-Domain Annotation

We first evaluate our approach in training and evaluating on the same domain. This mimics the scenario where one takes an existing dataset, and uses it to annotate novel images from the same domain. In particular, we use the Cityscapes dataset [8], which is currently one of the most comprehensive benchmarks for instance segmentation. It contains 2975 training, 500 validation and 1525 test images with 8 semantic classes. To ensure a fair comparison, we follow the same alternative split proposed by [4]. As in [4], we preprocess

Model	Bicycle	Bus	Person	Train	Truck	Motorcycle	Car	Rider	**Mean**
Square Box	35.41	53.44	26.36	39.34	54.75	39.47	46.04	26.09	40.11
DeepMask [21]	47.19	69.82	47.93	62.20	63.15	47.47	61.64	52.20	56.45
SharpMask [22]	52.08	73.02	53.63	64.06	65.49	51.92	65.17	56.32	60.21
Polygon-RNN [4]	52.13	69.53	63.94	53.74	68.03	52.07	71.17	60.58	61.40
Residual Polygon-RNN	54.86	69.56	67.05	50.20	66.80	55.37	70.05	63.40	62.16
+ Attention	56.47	73.57	68.15	53.31	74.08	57.34	75.13	65.42	65.43
+ RL	57.38	75.99	68.45	59.65	76.31	58.26	75.68	65.65	67.17
+ Evaluator Network	62.34	79.63	70.80	62.82	77.92	61.69	78.01	68.46	70.21
+ GGNN	**63.06**	**81.38**	**72.41**	**64.28**	**78.90**	**62.01**	**79.08**	**69.95**	**71.38**

Table 1: Performance (IoU in % in val test) on all the Cityscapes classes in **automatic mode**. All methods exploit GT boxes.

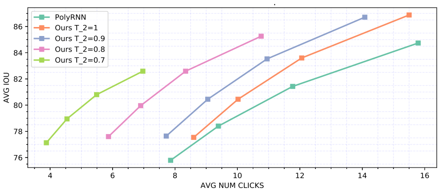

Figure 6: Interactive mode on Cityscapes

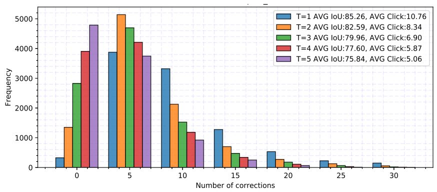

Figure 7: Interactive mode on Citysc. $(T_2 = 0.8)$

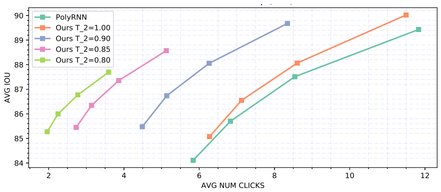

Figure 8: Interactive mode on KITTI

the ground-truth polygons according to depth ordering to obtain polygons for only the visible regions of each instance.

Evaluation Metrics: We utilize two quantitative measures to evaluate our model. **1)** We use the intersection over union (IoU) metric to evaluate the quality of the generated polygons and **2)** we calculate the number of annotator clicks required to correct the predictions made by the model. We describe the correction protocol in detail in a subsequent section.

Baselines: Following [4], we compare with DeepMask [21], SharpMask [22], as well as Polygon-RNN [4] as state-of-the-art baselines. Note that the first two approaches are pixel-wise methods and errors in their output cannot easily be corrected by an annotator. To be fair, we only compare our automatic mode with their approaches. In their original approach, [21, 22] exhaustively sample patches at different scales over the entire image. Here, we evaluate [21, 22] by providing exact ground-truth boxes to their models. As per [4], we also include SquareBox, which considers the provided bounding box as its prediction.

Automatic Mode: We compare Polygon-RNN++ to the baselines in Table 1, and ablate the use of each of the components in our model. Here, Residual Polygon-RNN refers to the original model with our image encoder instead of VGG. Our full approach outperforms all competitors by 10% IoU, and achieves best performance for each class. Moreover, Polygon-RNN++ surpasses the reported human agreement [4] of 78.6% IoU on *cars*, on average. Using human agreement on cars as a proxy, we observe that the model also obtains human-level performance for *truck* and *bus*.

Interactive Mode: The interactive mode aims to minimize annotation time while obtaining high quality annotations. Following the simulation proposed in [4], we calculate the number of annotator clicks required to correct predictions from the model. The annotator corrects a prediction if it deviates from the corresponding GT vertex by a min distance of T, where the hyperparameter T governs the quality of the produced annotations. For fair comparison, distances are computed using manhattan distance at the model output resolution using distance thresholds $T \in [1, 2, 3, 4]$, as in [4].

We introduce another threshold T_2, defined as the IoU between the predicted polygon and the GT mask. We consider polygons with agreement above T_2 unnecessary for the annotator to interfere. We use this threshold due to the somewhat unsatisfactory correction simulation above: for example, if the predicted vertex falls along a GT edge, this vertex is in fact correct and should not be corrected. Note that when $T_2 = 1$, simulation is equivalent to the one above.

In Fig. 6, we compare the average number of clicks per instance required to annotate **all classes** on the Cityscapes *val* set with different values of T_2. Using $T_2 = 1$, we see that our model outperforms [4], requiring fewer clicks to obtain the same IoU. Even at $T_2 = 0.8$ our model is still more accurate than [4] at $T_2 = 1.0$. At $T_2 = 0.7$, we achieve over 80% mIoU with only 5 clicks per object on average, which is a reduction of more than 50% over [4]. Fig. 7 shows frequency of required corrections for different T at $T_2 = 0.8$. In Sec.4.4, we show results with real human annotators.

Robustness to bbox noise: We analyze the effect of noise in the bbox provided to the model. We randomly expand the box by a percentage of its width and height. Results in Table 4 illustrates that our model is very robust to some amount of noise (0-5%). Even in the presence of moderate and large noise (5-10%,10-15%), it outperforms the reported performance of previous baselines which use perfect boxes.

4.1.1 Instance-Level Segmentation

We evaluate our model on (automatic) full-image instance segmentation, by exploiting FasterRCNN [27] to detect objects. Polygon-RNN++ with FasterRCNN achieves 22.8% AP and 42.6% AP_{50} on **Cityscapes test**. Following [18], we also add semantic segmentation [40] to post-process the results. We simply perform a logical "and" operation be-

Model	ADE	Rooftop	Cardiac MR	ssTEM
SquareBox (Expansion)	42.95	40.71	62.10	42.24
Ellipse (Expansion)	48.53	47.51	73.63	51.04
Square Box (Perfect)	69.35	62.11	79.11	66.53
Ellipse (Perfect)	69.53	66.82	92.44	71.32
DeepMask[21]	59.74	15.82	60.70	31.21
SharpMask[22]	61.66	18.53	69.33	46.67
Ours w/o GGNN	70.21	65.03	80.55	53.77
Ours w/ GGNN	71.82	65.67	80.63	53.12

Table 2: Out-of-domain automatic mode performance

Model	IoU (%)
DeepMask [21]	78.3
SharpMask [22]	78.8
Beat The MTurkers [5]	73.9
Polygon-RNN [4]	74.22
Ours w/o GGNN	81.40
Ours w/ GGNN	**83.14**

Table 3: Car annot. results on KITTI in automatic mode (no fine-tuning, 0 clicks)

Bbox Noise	IoU (%)
0%	71.38
0-5%	70.54
5-10%	68.07
10-15%	64.80

Table 4: Robustness to Bounding Box noise on Cityscapes (in % of side length at each vertex)

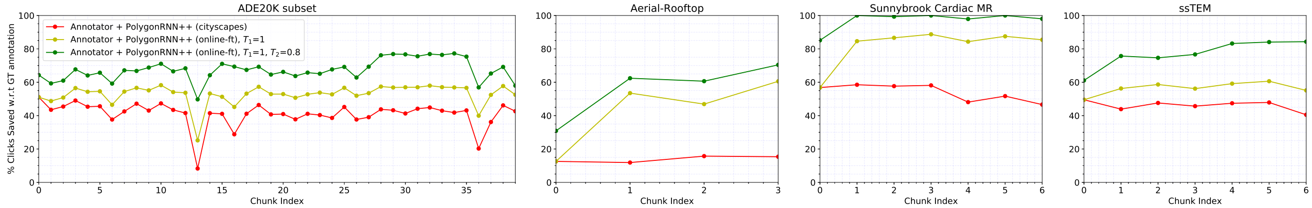

Figure 9: Percentage clicks saved with online fine-tuning on out-of-domain datasets (Plots share legend and y axis)

tween the predicted class-semantic map and our prediction. We achieve 25.49% AP and 45.47% AP_{50} on the **test set**.

4.2. Cross-Domain Evaluation

Here we analyze performance on different datasets that capture both shifts in environment (KITTI) and domain (general scenes, aerial, medical). We first use our model trained on Cityscapes **without any fine-tuning** on these datasets.

KITTI [10]: We use Polygon-RNN++ to annotate 741 instances of KITTI [10] provided by [5]. The results in automatic mode are reported in Table 3 and the performance with a human in the loop is illustrated in Fig. 8. Our method outperforms all baselines showcasing its robustness to change in environment while being in a similar domain. With an annotator in the loop, our model requires on average 5 fewer clicks than [4] to achieve the same IoU. It achieves human level agreement of 85% as reported by [5] by requiring only 2 clicks on average by the (simulated) annotator.

Out-of-Domain Imagery We consider datasets with a large domain shift wrt the Cityscapes dataset in order to evaluate the generalization capabilities of our model.

ADE20K [41]: The ADE20K dataset is a challenging general scene parsing dataset containing 20,210 images in the training set, 2,000 images in the validation set, and 3,000 images in the testing set. We select the following subset of categories from the validation set: *television receiver*, *bus*, *car*, *oven*, *person* and *bicycle* in our evaluation.

Aerial [33]: Rooftop dataset consists of 65 aerial images of rural scenes containing several building rooftops, a majority of which exhibit fairly complex polygonal geometry. Performance for this dataset is reported for the *test* set.

Medical Imagery [15, 32, 11]: We use two medical segmentation datasets [15, 32] and [11] for our experiments. The former, used in the Left Ventricle Segmentation Challenge [32], divides the data of 200 patients equally in the *training* and *validation* sets. We report the performance on a subset of the *validation* set which only includes the outer contours that segment the epicardium. The latter provides two image stacks (training and testing) each containing 20 sections from serial section Transmission Electron Microscopy (ssTEM) images of the ventral nerve cord. We use the mitochondria and synapse segmentations from this data to test our model. Since ground-truth instances for the test set are not publicly available, we evaluate on the training set.

Quantitative Results: For out-of-domain datasets, we introduce a baseline (named Ellipse) which fits an ellipse into the GT bounding box which is motivated by the observation that many instances in [32] are ellipses. We show results with perfect and expanded bounding boxes (expansion similar to our model) for Square Box and Ellipse. DeepMask and SharpMask were evaluated with perfect bounding boxes with the threshold suggested by the authors. Table 2, demonstrates high generalization capabilities of our model.

Online Fine-tuning: In these experiments, our simulated annotator has parameters $T = 1$ and $T_2 = 0.8$. Fig. 9 reports the percentage of clicks saved with respect to GT polygons for our Cityscapes model and the online fine-tuned models. We see that our adaptive approach overcomes stark domain shifts with as few as one chunk of data (40 images for Sunnybrook, 3 for ssTEM, 200 for ADE and 20 for Aerial) showcasing strong generalization. Overall, we show at least 65% overall reduction in the number of clicks across all datasets, with the numbers almost at 100% for the Sunnybrook Cardiac MR dataset. We believe these results pave the way towards a real annotation tool that can learn along with the annotator and significantly reduce human effort.

4.3. Qualitative Results

Fig. 10 shows example predictions obtained in automatic mode on Cityscapes. We illustrate the improvements from specific parts of the model in Fig. 11. We see how using RL and the evaluator network leads to crisper predictions, while

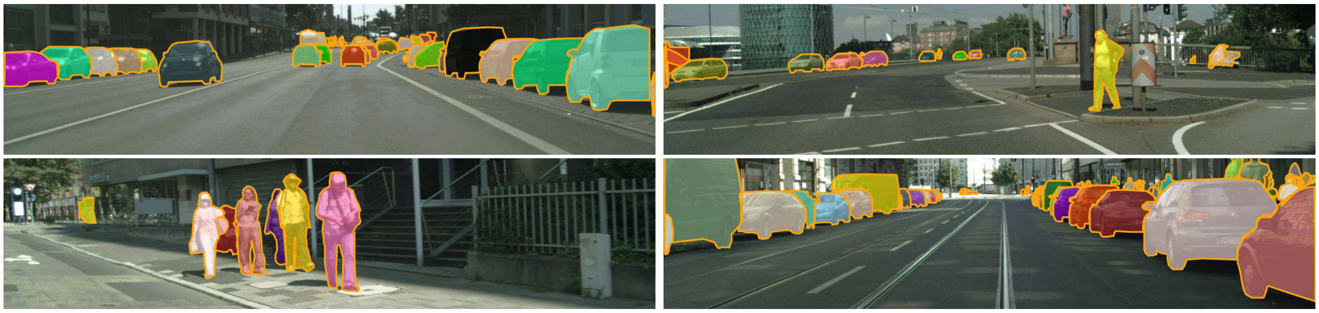

Figure 10: Qualitative results in automatic mode on Cityscapes. **Left:** requiring GT boxes; **Right:** FasterRCNN + PolygonRNN++

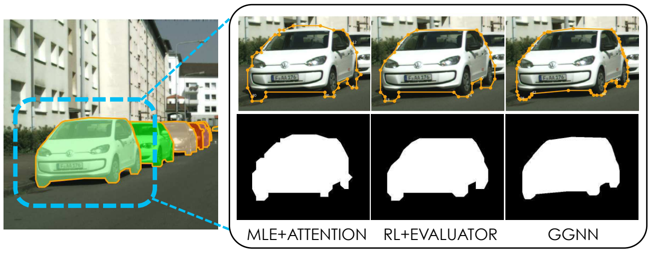

Figure 11: Results with different components of the model

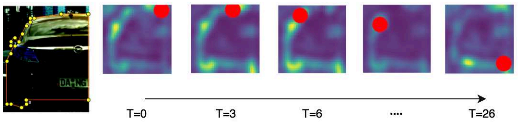

Figure 13: Visualization of attention maps in our model

the GGNN upscales, adds points and builds a polygon resembling human annotation. Fig. 12 showcases automatic predictions from PolygonRNN++ on the out-of-domain datasets. We remind the reader that this labeling is obtained by exploiting GT bounding boxes, and no fine-tuning.

4.4. Interaction with Human Annotators

We also conducted a small scale experiment with real human annotators in the loop. To this end, we implemented a very simple annotation tool that runs our model at the backend. We use 54 *car* instances from Cityscapes as per [4]. We asked two human subjects to annotate these interactively using our model, and two to annotate manually. While we explain how the tool works, we do not train the annotators to use our tool. All our annotators were in-house.

Timing begins when an annotator first clicks on an object, and stops when the "submit" button is clicked. While using our model, the annotator needs to draw a bounding box around the object, which we include in our reported timing. Note that we display the object to the annotator by cropping an image inside an enlarged box. Thus our annotators are fast in drawing the boxes, taking around 2 seconds on average.

We report results in Table 5. Annotators are 3x faster when using our model, with only slightly lower IoU agreement with GT. Note that our tool has scope for improvement in various engineering aspects. [4] reported that on these examples, human subjects needed on average 42.2 sec per object using GrabCut [30], and obtained a lower IoU (70.7).

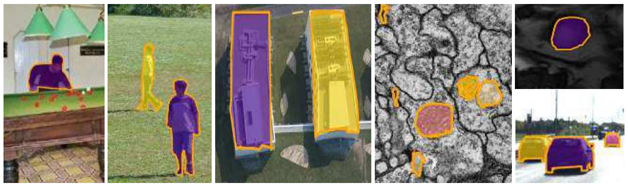

Figure 12: Qualitative results in automatic mode on different unseen datasets without fine-tuning

	Cityscapes		ADE	
	Time (s)	IoU (%)	Time (s)	IoU (%)
manual	39.7	76.2	29.2	80.63
with PolygonRNN++	14.7	75.4	19.3	75.9

Table 5: **Real Human Experiment**: **In-domain** on 50 randomly chosen Cityscapes car instances (left) and **Out-of-domain** on 40 randomly chosen ADE20K instances (right). No fine-tuning was used in ADE experiment.

We also investigate cross-domain annotation. In particular, we use the ADE20k dataset and our model trained on Cityscapes (no fine-tuning). We randomly chose a total of 40 instances of *car*, *person*, *sofa* and *dog*. Here *car* and *person* are two classes seen in Cityscapes (i.e., *person* $\sim$ *pedestrian* in Cityscapes), and *sofa* and *dog* are **unseen** categories. From results in Table 5, we observe that the humans were still faster when using our tool, but less so, as expected.

Limitations: Our model predicts one polygon per box, typically annotating the more central object. If one object breaks the other, our approach tends to predict the occluded object as a single polygon. As a result, current failures cases are mostly multi-component objects. Note also that we do not handle holes which do not appear in our tested datasets.

5. Conclusion

In this paper, we proposed Polygon-RNN++, a model for object instance segmentation that can be used to interactively annotate segmentation datasets. The model builds on top of Polygon-RNN [4], but introduces several important improvements that significantly outperform the previous approach in both, automatic and interactive modes. We further show generalization of our model to novel domains. We also show that with a simple online fine-tuning scheme, our model can be used to effectively adapt to novel, out-of-domain datasets.

Acknowledgements. We gratefully acknowledge NVIDIA for donating several GPUs used in this research. We also thank Kaustav Kundu for his help and advice, and Relu Patrascu for infrastructure support.

References

[1] M. Bai and R. Urtasun. Deep watershed transform for instance segmentation. In *CVPR*, 2017.

[2] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *ICCV*, 2001.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *T-PAMI*, 26(9):1124–1137, 2004.

[4] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017.

[5] L.-C. Chen, S. Fidler, A. Yuille, and R. Urtasun. Beat the mturkers: Automatic image labeling from weak 3d supervision. In *CVPR*, 2014.

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. In *ICLR*, 2015.

[7] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *T-PAMI*, 40(4):834–848, April 2018.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[9] L. Duan and F. Lafarge. Towards large-scale city reconstruction from satellites. In *ECCV*, 2016.

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *CVPR*, 2012.

[11] S. Gerhard, J. Funke, J. Martel, A. Cardona, and R. Fetter. Segmented anisotropic ssTEM dataset of neural tissue. *figshare*, 2013.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *ICCV*, 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[15] A. H. Kadish, D. Bello, J. P. Finn, R. O. Bonow, A. Schaechter, H. Subacius, C. Albert, J. P. Daubert, C. G. Fonseca, and J. J. Goldberger. Rationale and Design for the Defibrillators to Reduce Risk by Magnetic Resonance Imaging Evaluation (DETERMINE) Trial. *J Cardiovasc Electrophysiol*, 20(9):982–7, 2009.

[16] K. Li, B. Hariharan, and J. Malik. Iterative instance segmentation. In *CVPR*, 2016.

[17] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

[18] S. Liu, J. Jia, S. Fidler, and R. Urtasun. Sequential grouping networks for instance segmentation. In *ICCV*, 2017.

[19] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. In *CVPR*, 2015.

[20] N. S. Nagaraja, F. R. Schmidt, and T. Brox. Video segmentation with just a few strokes. In *ICCV*, 2015.

[21] P. O. Pinheiro, R. Collobert, and P. Dollar. Learning to segment object candidates. In *NIPS*, pages 1990–1998, 2015.

[22] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *ECCV*, 2016.

[23] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. Full-resolution residual networks for semantic segmentation in street scenes. *CVPR*, 2017.

[24] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *ICCV*, 2017.

[25] M. Rajchl, M. C. Lee, O. Oktay, K. Kamnitsas, J. Passerat-Palmbach, W. Bai, M. Damodaram, M. A. Rutherford, J. V. Hajnal, B. Kainz, and D. Rueckert. Deepcut: Object segmentation from bounding box annotations using convolutional neural networks. In *IEEE Trans. on Medical Imaging*, 2017.

[26] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. *ICLR*, 2016.

[27] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[28] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. *CVPR*, 2017.

[29] B. Romera-Paredes and P. H. S. Torr. Recurrent instance segmentation. In *arXiv:1511.08250*, 2015.

[30] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, 2004.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Trans. on Neural Networks*, 20(1):61–80, 2009.

[32] A. Suinesiaputra, B. R. Cowan, A. O. Al-Agamy, M. A. Elattar, N. Ayache, A. S. Fahmy, A. M. Khalifa, P. Medrano-Gracia, M.-P. Jolly, A. H. Kadish, D. C. Lee, J. Margeta, S. K. Warfield, and A. A. Young. A collaborative resource to build consensus for automated left ventricular segmentation of cardiac MR images. *Medical Image Analysis*, 18(1):50 – 62, 2014.

[33] X. Sun, C. M. Christoudias, and P. Fua. Free-shape polygonal object localization. In *ECCV*, 2014.

[34] S. Wang, M. Bai, G. Mattyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun. Torontocity: Seeing the world with a million eyes. In *ICCV*, 2017.

[35] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, 1992.

[36] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810, 2015.

[37] Z. Zhang, S. Fidler, and R. Urtasun. Instance-level segmentation for autonomous driving with deep densely connected mrfs. In *CVPR*, 2016.

[38] Z. Zhang, S. Fidler, J. W. Waggoner, Y. Cao, J. M. Siskind, S. Dickinson, and S. Wang. Super-edge grouping for object localization by combining appearance and shape information. In *CVPR*, 2012.

[39] Z. Zhang, A. Schwing, S. Fidler, and R. Urtasun. Monocular object instance segmentation and depth ordering with cnns. In *ICCV*, 2015.

[40] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[41] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017.