

# Fast Interactive Object Annotation with Curve-GCN

Paper by: [Huan Ling](#)\*<sup>1,2</sup>, [Jun Gao](#)\*<sup>1,2</sup>, [Amlan Kar](#)<sup>1,2</sup>, [Wenzheng Chen](#)<sup>1,2</sup>, [Sanja Fidler](#)<sup>1,2,3</sup>

<sup>1</sup>University of Toronto <sup>2</sup>Vector Institute <sup>3</sup>NVIDIA

CVPR-2019

Presented by: To Hai Thien

Student ID: 2018-30630

Seoul National University

ECE Dept. PIDL Lab.

# Introduction and Previous Research.

- ❖ Why we need an end-to-end object annotation tool?
  - Object annotation tools are very important for many fields such as AI, Data Science, Computer Vision and so on.
  - Manually labelling objects by tracing their boundaries is an expensive process.

- ❖ Propose *end-to-end fast interactive object annotation tool* with Curve-GCN

- Automatically outlines the object.
- Allows interactive corrections.
- Automatically re-predicts with *either polygon or spline*.
- *Predicting all vertices (or control points) simultaneously* using a Graph Convolution Network (GCN) → corrects car area faster than PolygonRNN++



**Curve-GCN**

Interactive Object Annotation Tool



 Add box

**Interactive**  
Polygon  Spline

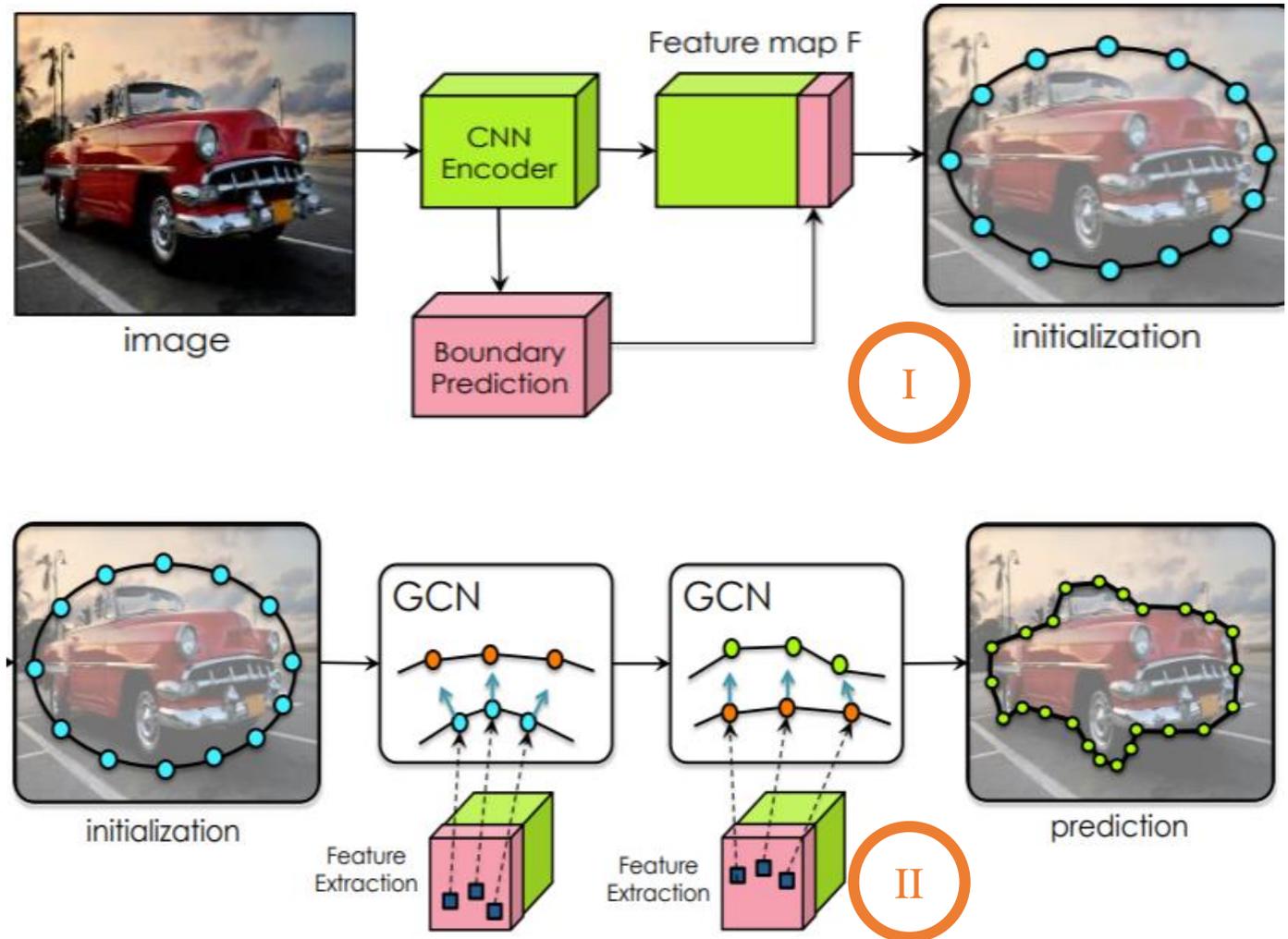


Polygon

Spline

# Curve-GCN System Overview

- ❖ From an image, *we initialize*  $N$  control points (I).
- ❖ The object is represented as graph with a fixed topology, and perform prediction using a GCN. (II)
  - *GCN Graph Definitions.*
  - *GCN Model.*
  - *Interactive GCN / Human-in-the-loop*
  - *Prediction*



*Define GCN Graph*   *GCN Model.*   *Interactive-GCN*   *Prediction*

# Curve-GCN Graph Definitions.

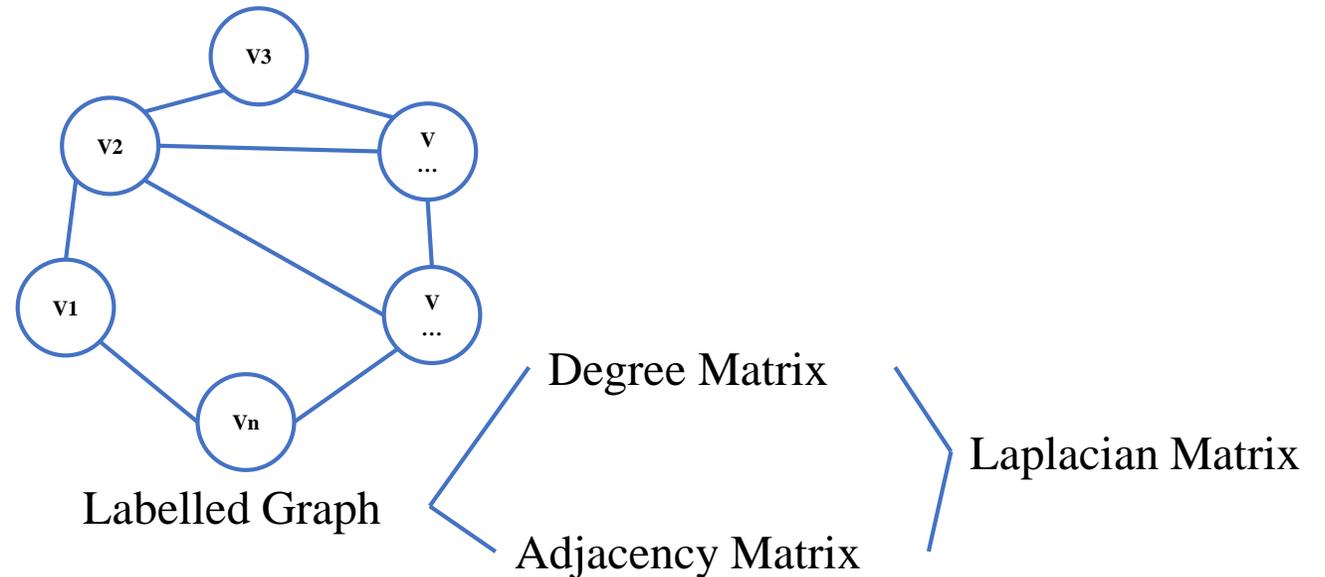
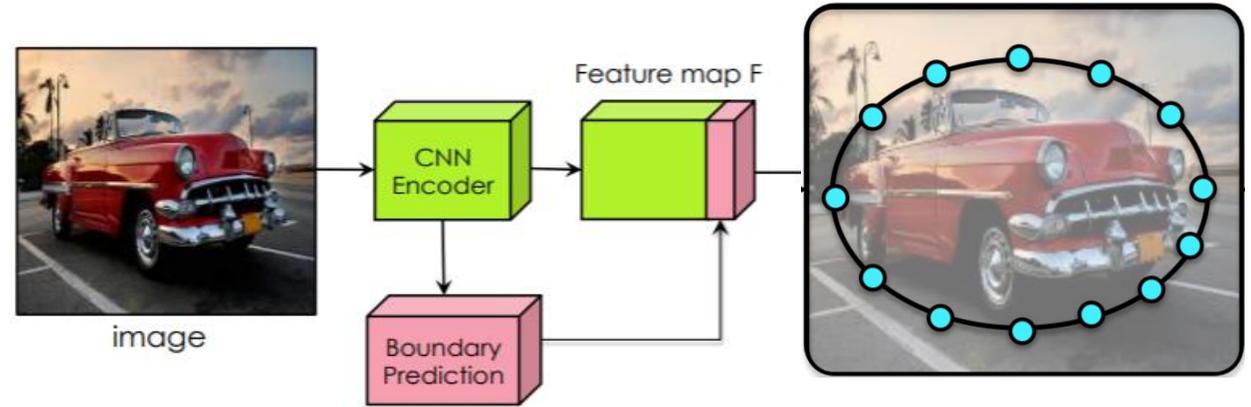
Let:

- $cp_i = [x_i, y_i]^T$  denote the location of  $i$ -th control point.
- $V = \{cp_0, cp_1, \dots, cp_{N-1}\}$  be the set of all control points.

→ They define a graph.

$$\mathbf{G} = (\mathbf{V}, \mathbf{E}).$$

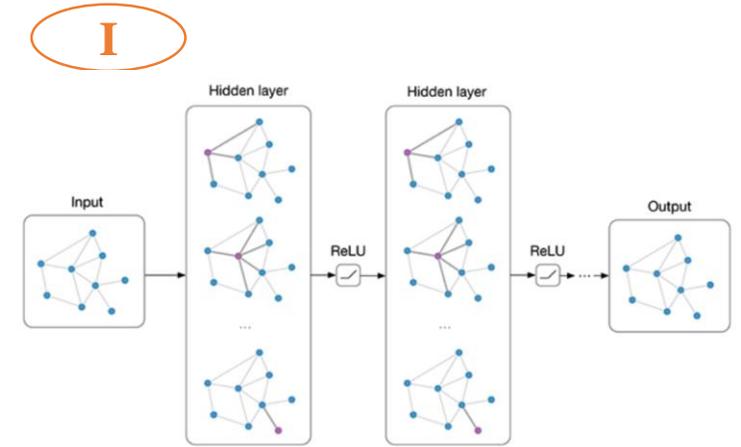
$V$  are the nodes.  $E$  are the edges.



# From Graph $\rightarrow$ GCN Model Propagation

❖ Graph propagation rule is: 
$$f_i^{l+1} = w_0^l f_i^l + \sum_{\mathbf{cp}_j \in \mathcal{N}(\mathbf{cp}_i)} w_1^l f_j^l$$

- $\mathcal{N}(\mathbf{cp}_i)$  denotes the nodes that are connected to  $\mathbf{cp}_i$  in the graph.
- $w_0^l, w_1^l$  are weight matrices.



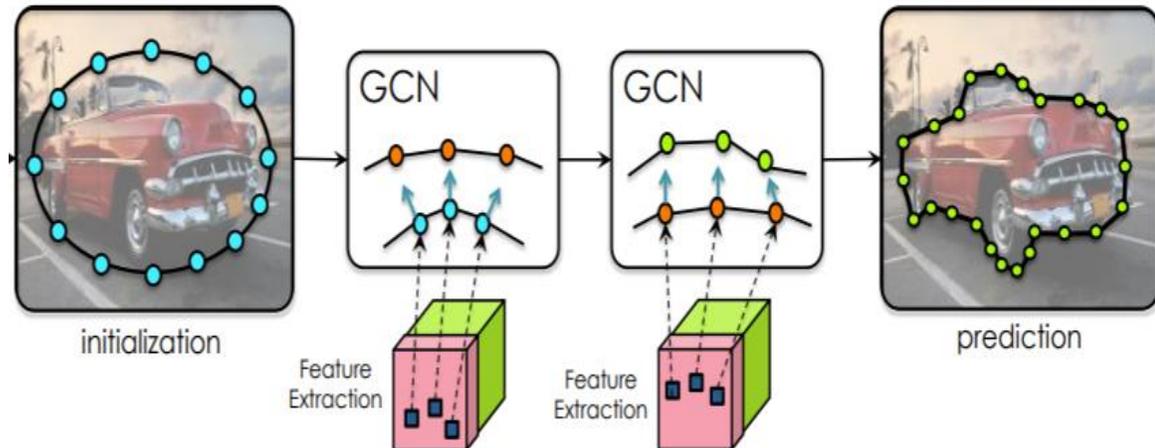
❖ Features extracted from the corresponding location in map F:

$$f_i^0 = \text{concat}\{F(x_i, y_i), x_i, y_i\} \text{ s.t. } F(x_i, y_i), \text{ is computed using bilinear interpolation.}$$

**II**

❖ We utilize a Graph-ResNet to propagate information between nodes in the graph as a residual function.

❖ Then takes the following form.



$$r_i^l = \text{ReLU} \left( w_0^l f_i^l + \sum_{\mathbf{cp}_j \in \mathcal{N}(\mathbf{cp}_i)} w_1^l f_j^l \right)$$

$$r_i^{l+1} = \tilde{w}_0^l r_i^l + \sum_{\mathbf{cp}_j \in \mathcal{N}(\mathbf{cp}_i)} \tilde{w}_1^l r_j^l$$

$$f_i^{l+1} = \text{ReLU}(r_i^{l+1} + f_i^l),$$

**III**

# Interactive GCN (Annotator in loop).

$$\diamond f_i^0 = \text{concat}\{F(x_i, y_i), x_i, y_i, \Delta x_i, \Delta y_i\}.$$

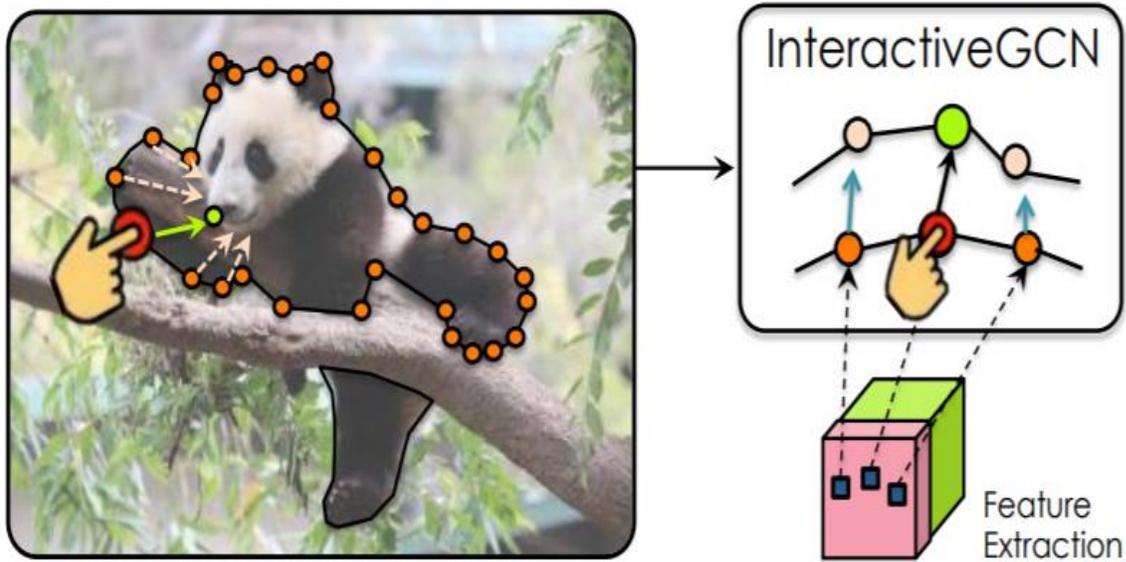


Figure 4: **Human-in-the-Loop**: An annotator can choose any wrong control point and move it onto the boundary. Only its immediate neighbors ( $k = 2$  in our experiments) will be re-predicted based on this interaction.

---

## Algorithm 1 Learning to Incorporate Human-in-the-Loop

---

```
1: while not converged do
2:   (rawImage, gtCurve) = Sample(Dataset)
3:   (predCurve,  $F$ ) = Predict(rawImage)
4:   data = []
5:   for  $i$  in range( $c$ ) do
6:     corrPoint = Annotator(predictedCurve)
7:     data += (predCurve, corrPoint, gtCurve,  $F$ )
8:     predCurve = InteractiveGCN(predCurve, corrPoint)
9:                                     ▷ Do not stop gradients
10:  TrainInteractiveGCN(data)
```

---

# Experimental.

## ❖ Dataset Preparation.

- Download the Cityscapes dataset (leftImg8bit\_trainvaltest.zip) from the official.
- <https://www.cityscapes-dataset.com/downloads/>
- Processed annotation files from <http://www.cs.toronto.edu/~amlan/data/polygon/cityscapes.tar.gz>
- Using CNN Encoder and Boundary Prediction.

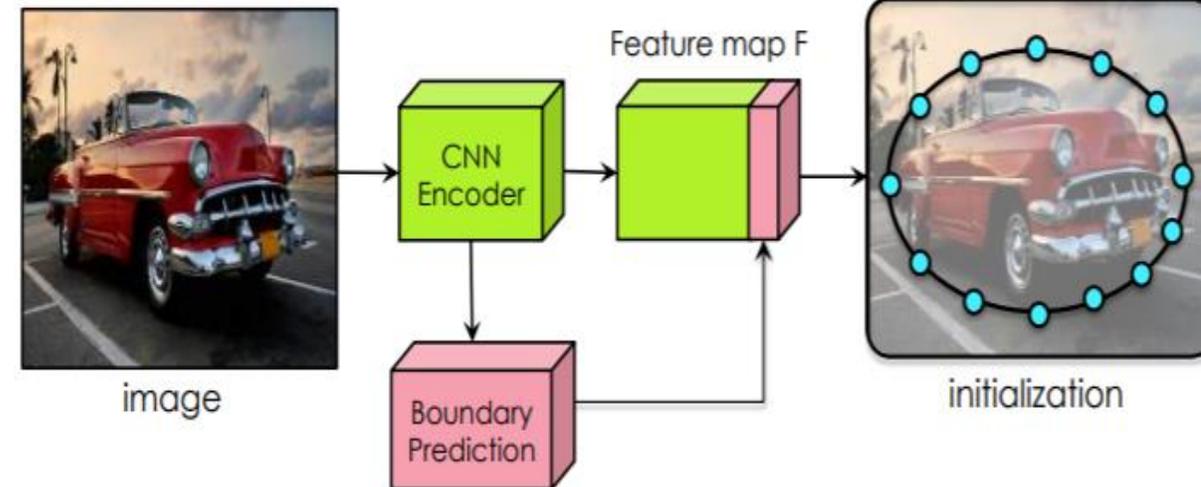
Annotation file → extract bbox, polygon\_points.

## ❖ Compute feature map by using ResNet50V2 model.

- $\text{feature\_map} = \text{ResNet50V2.prediction}(\text{bbox})$

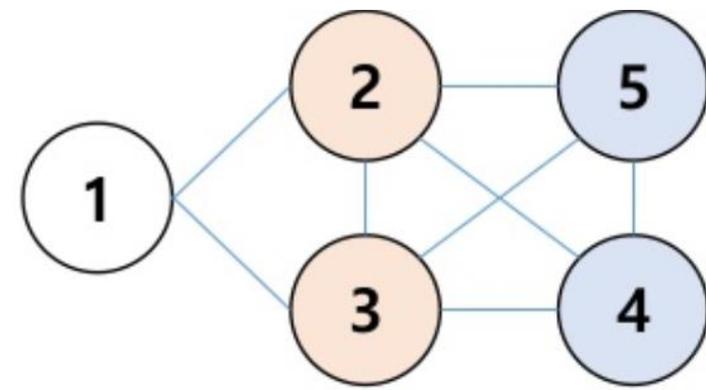
## ❖ Define graph.

- Initial  $N = 45 \rightarrow$  Calculate feature of each node.



# How to calculate A,X,H in GCN

- ❖ Edge: Adjacency Matrix (A)
- ❖ Node: Node Feature Matrix (X)



## ❖ Adjacency Matrix A (n x n)

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	0	1	1	0	0
Node 2	1	0	1	1	1
Node 3	1	1	0	1	1
Node 4	0	1	1	0	1
Node 5	0	1	1	1	0

## ❖ Feature Matrix X (n x f)

	Feat. 1	Feat. 2	Feat. 3	Feat. 4	Feat. 5
Node 1	1	1	1	0	0
Node 2	1	1	1	1	1
Node 3	1	1	1	1	1
Node 4	0	1	1	1	1
Node 5	0	1	1	1	1

# Building Model

```
# Compute feature map
resnet_model = ResNet50V2(weights='imagenet')
embedding_model = Model(inputs=resnet_model.input, outputs=resnet_model.get_layer('post_relu').output)
feature_map = embedding_model.predict(resized_bb_exp)
print(feature_map)
# Define graph
N = 45
G = nx.Graph()
G.add_nodes_from(range(N))
for i in range(N):
    if i-2 < 0:
        G.add_edge(N+(i-2), i)
    else:
        G.add_edge((i-2), i)
    if i-1 < 0:
        G.add_edge(N+(i-1), i)
    else:
        G.add_edge((i-1), i)
    G.add_edge((i+1)%N, i)
    G.add_edge((i+2)%N, i)

# Initialize node values
theta = np.linspace(0, 2*np.pi, N)
x, y = 0.5 + 0.4*np.cos(theta), 0.5 + 0.3*np.sin(theta)
node_values = np.array([[x[i]*bbox.shape[1], y[i]*bbox.shape[0]] for i in range(N)])
visualize_nodes(node_values, bbox)
visualize_nodes(polygon_points, bbox)
```

# Structure of GCN

Graph Inputs –  $G(X,A)$

Graph Conv, 256

Graph Conv, 128

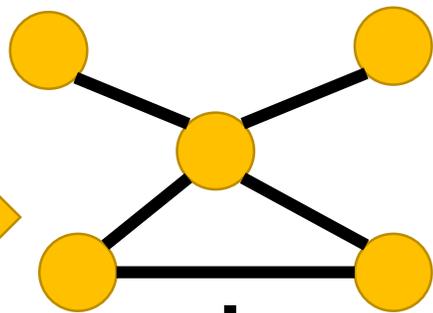
Graph Conv, 32

Graph Conv, 64

Dense, 64

Predictor, 1

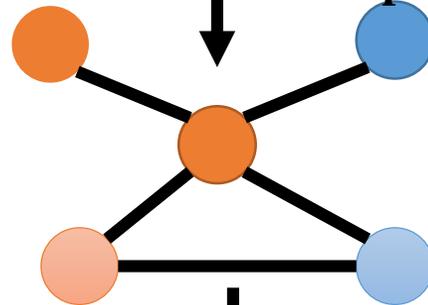
Labels



Input node features,  $\{H_i^{(0)}\}$

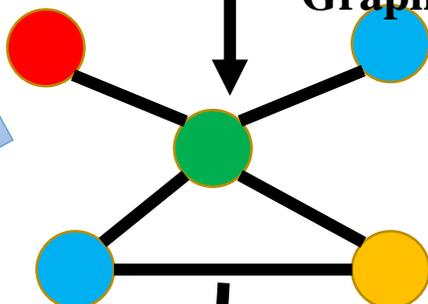
Raw node information

Graph Conv.

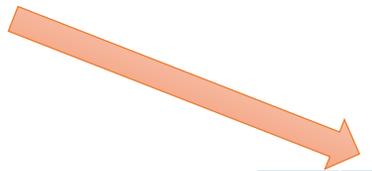
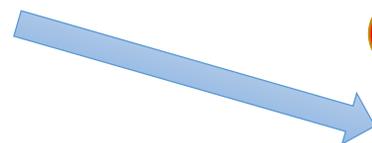


Graph Conv.

.....  
Graph Conv.



Final node states,  $\{H_i^{(L)}\}$

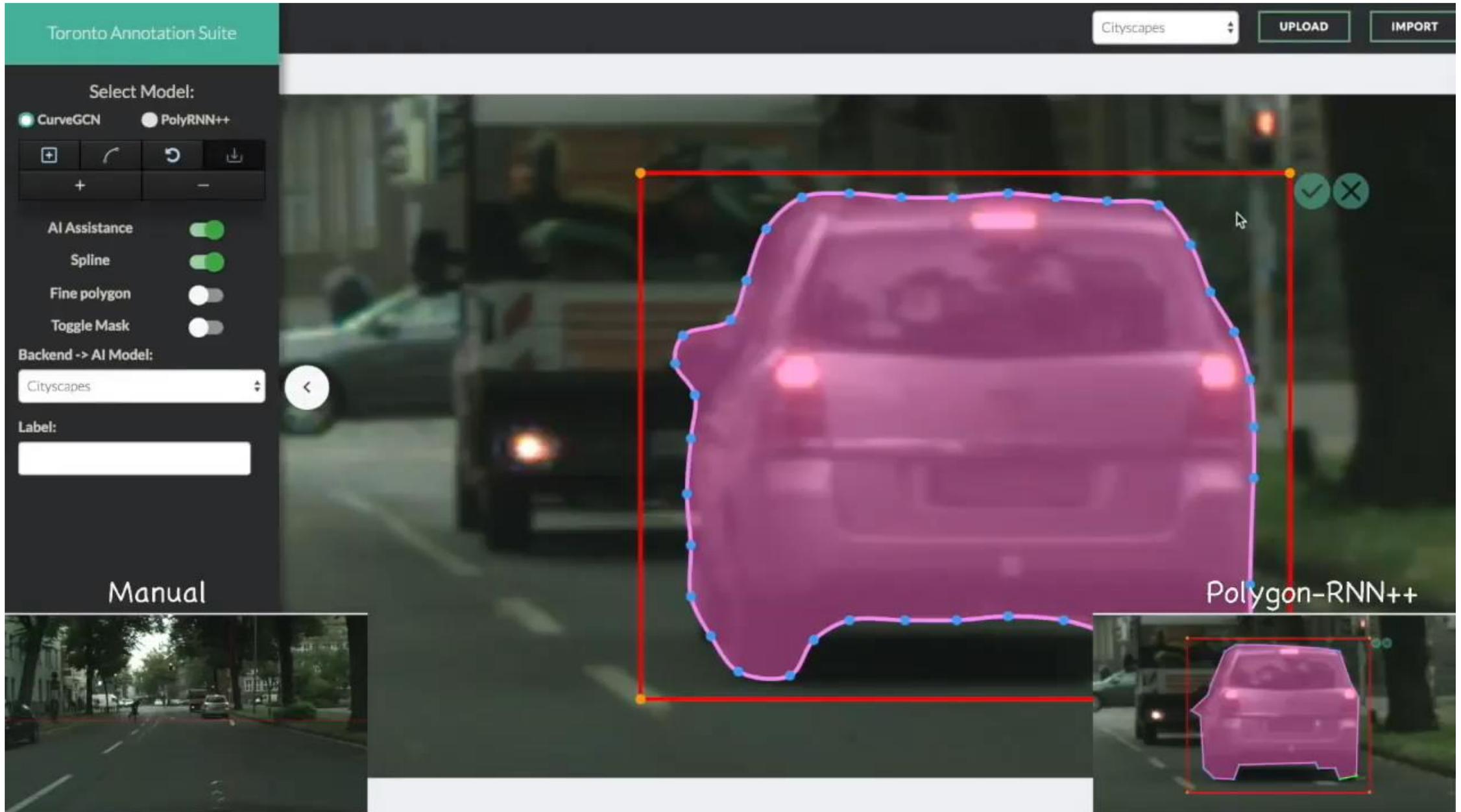


Dense

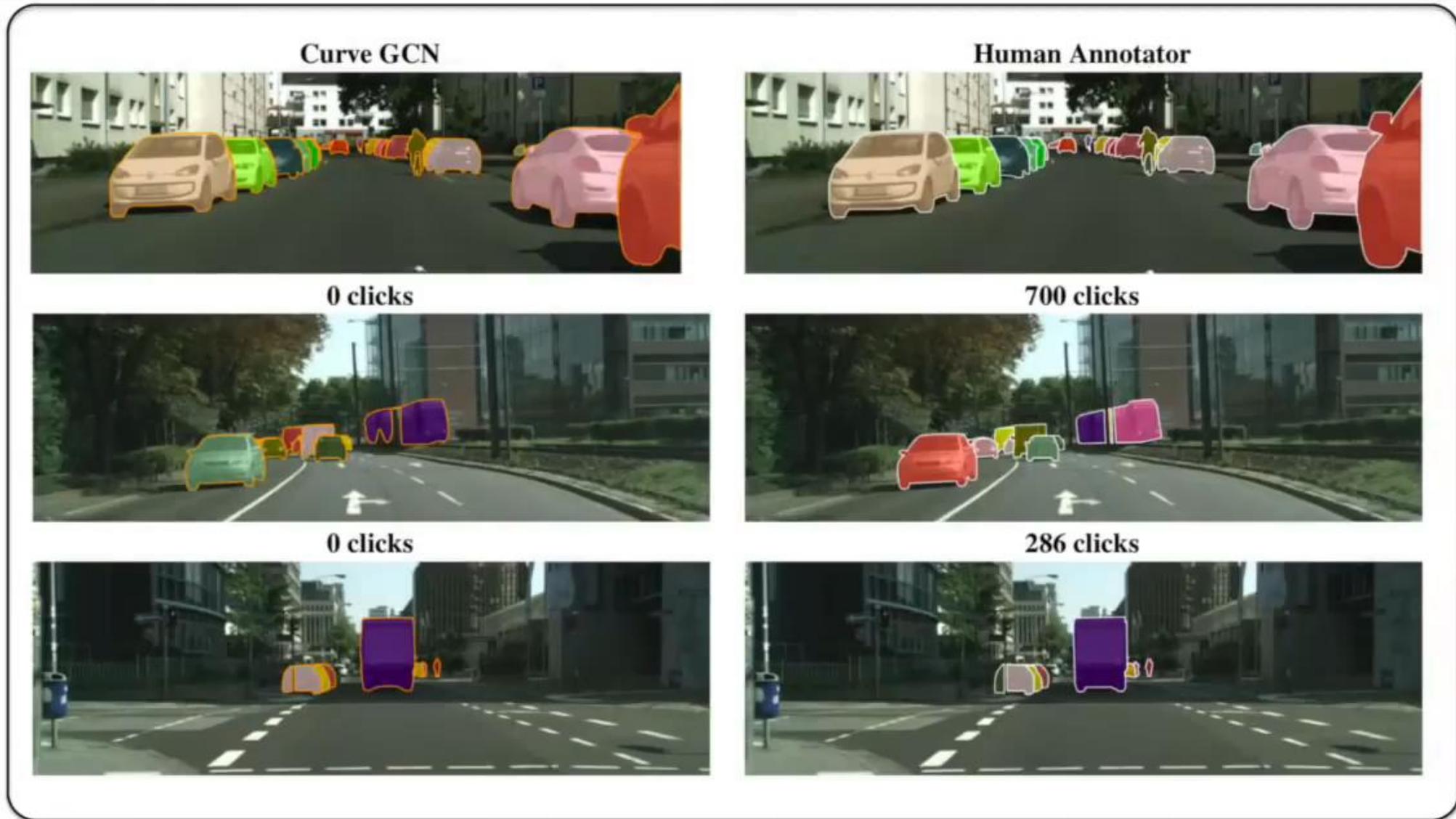


Predictor  
Property

# Experimental Results (Author's Implementation)



# Experimental Results (Author's Implementation)



Curve-GCN in 0-click regime

# Experimental Results (My implementation)

The screenshot displays the Visual Studio Code interface with the following components:

- Explorer:** Shows the project structure for 'CURVE-GCN', including folders like 'dataset', 'annotated\_images', 'raw\_images', 'test', 'train', 'val', 'gcn', and files like 'train.py', 'layers.py', 'process\_data.py', and 'visualize.py'.
- Editor:** Displays the code in 'train.py'. The code includes:
  - Graph construction: `for i in range(N):` with conditional `G.add_edge` calls.
  - Node initialization: `theta = np.linspace(0, 2*np.pi, N)` and `node_values = np.array([[x[i]*bbox.shape[1], y[i]*bbox.shape[0]] for i in range(N)])`.
  - Training loop: `for epoch in range(epochs_per_image):` with feature map construction and graph propagation.
  - Graph propagation: `offsets = compute_offset(polygon_points, node_values)` and `new_epoch_model, output_offsets = run_training(...)`.
- Warning:** A message in the bottom right corner states: "Unable to watch for file changes in this large workspace. Please follow the instructions link to resolve this issue." with an "Instructions" button.

# Implementation Link

[https://github.com/haithienld/Curve\\_GCN-Implementation](https://github.com/haithienld/Curve_GCN-Implementation)



Thank you  
for  
your listening