

# 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

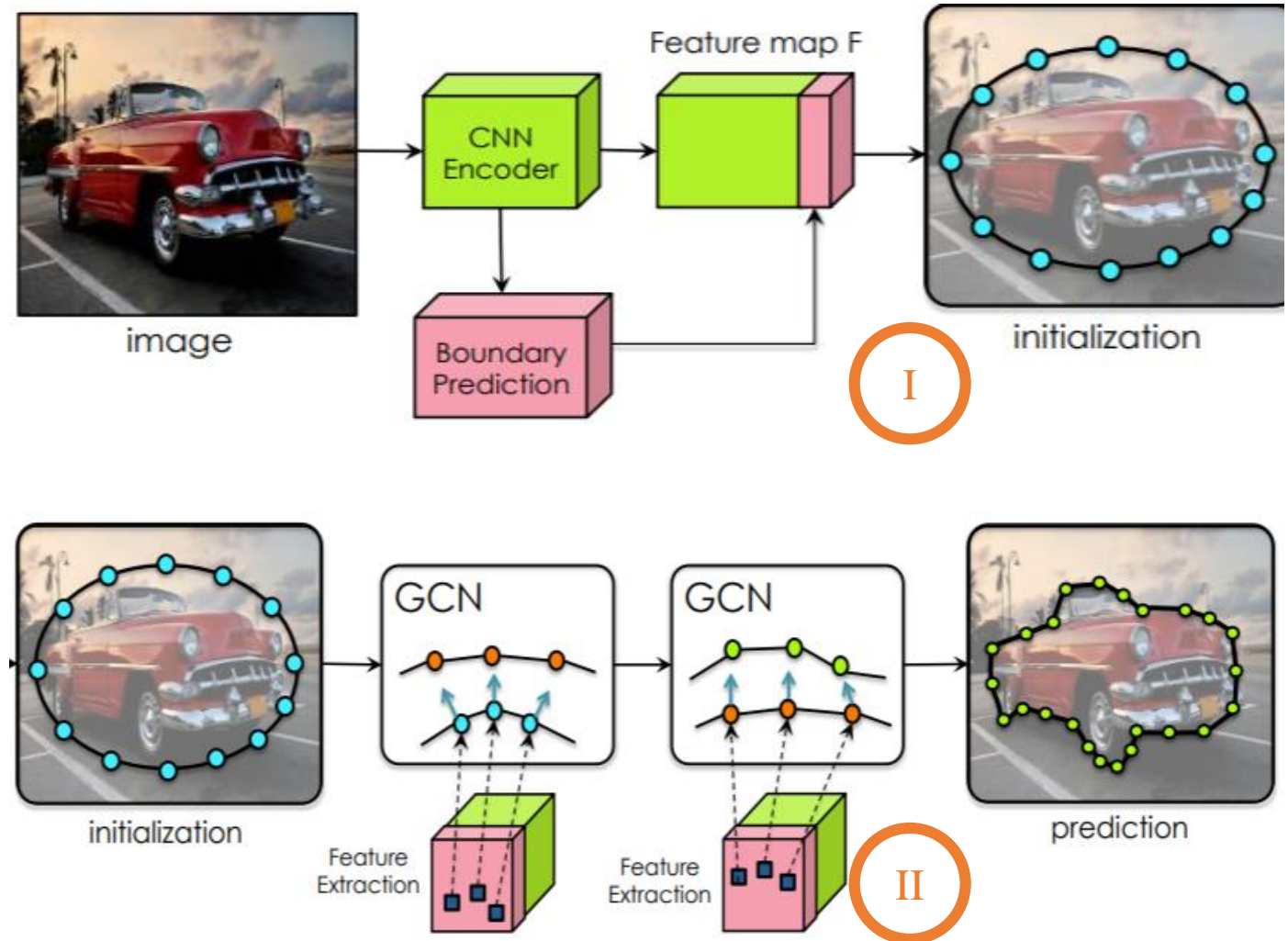
Spline

# 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.

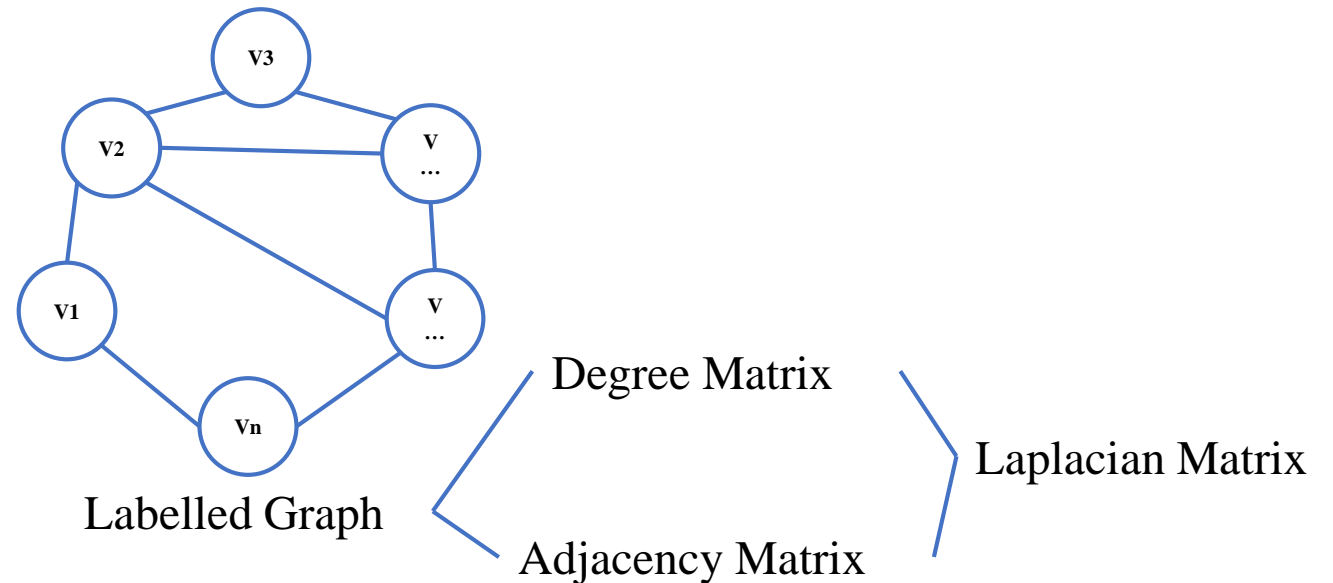
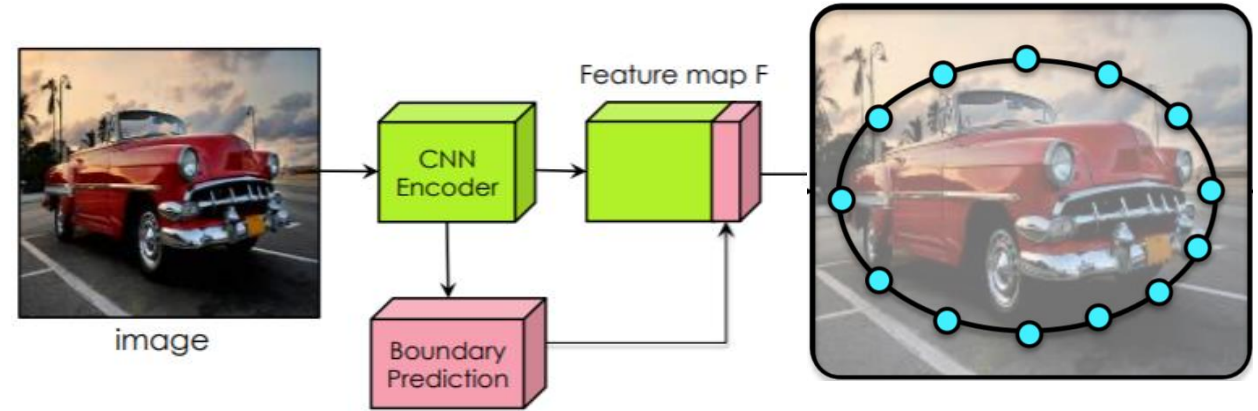
Let:

- $cp_i = [x_i, y_i]^T$  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.

$G = (V, E)$ .

V are the nodes. E are the edges.

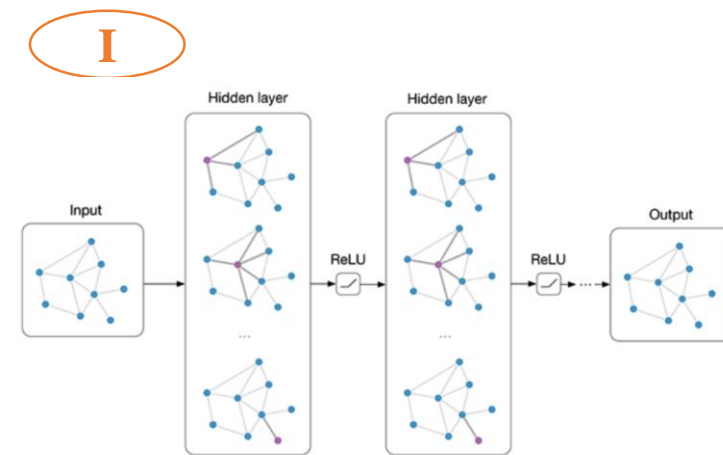




# From Graph $\rightarrow$ GCN Model Propagation

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

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



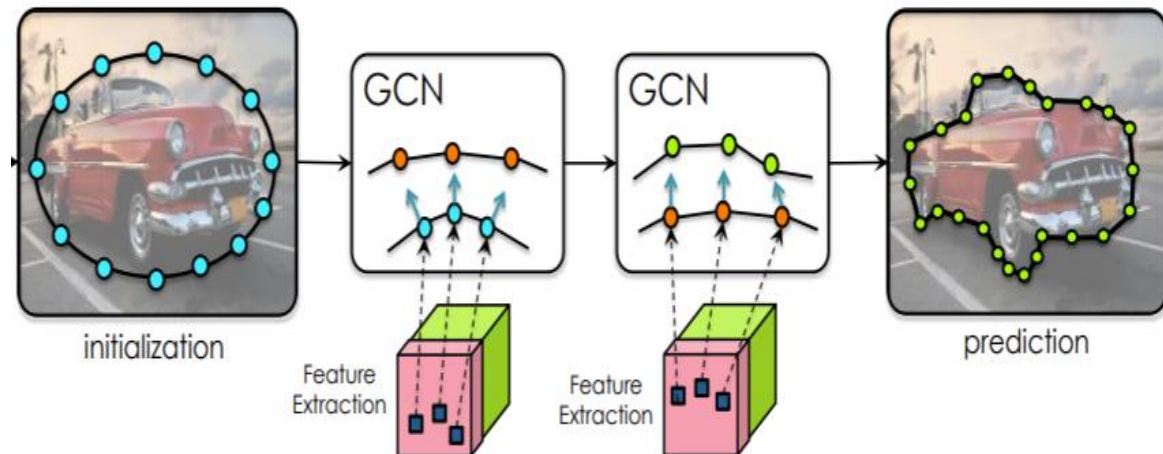
❖ Features extracted from the corresponding location in map F:

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

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).

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

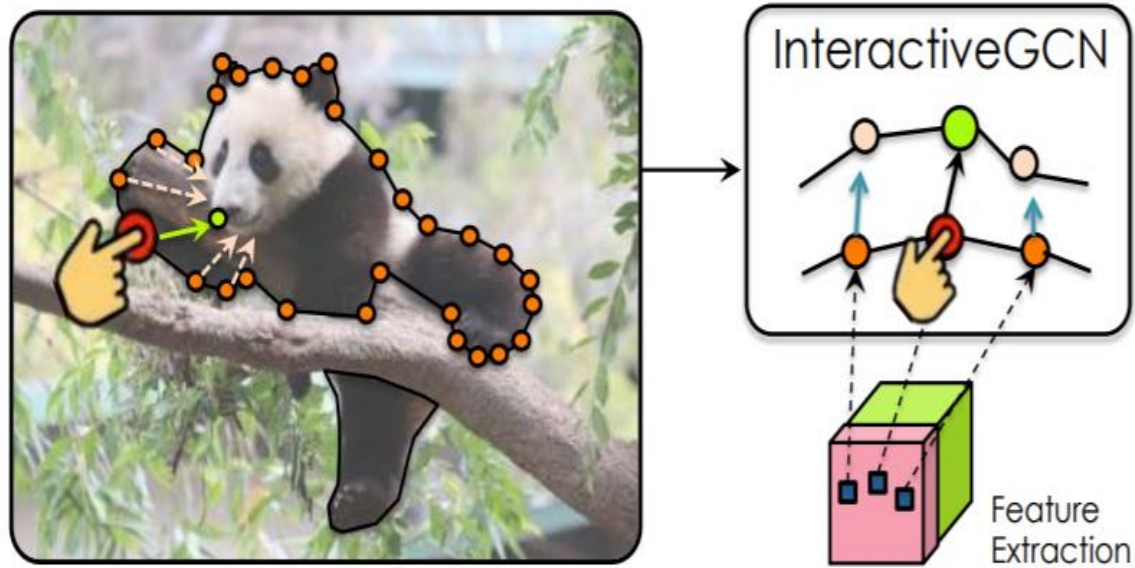


Figure 4: **Human-in-the-Loop**: An annotator can choose any wrong control point and move it onto the boundary. Only its immediate neighbors ( $k = 2$  in our experiments) will be re-predicted based on this interaction.

---

## 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.

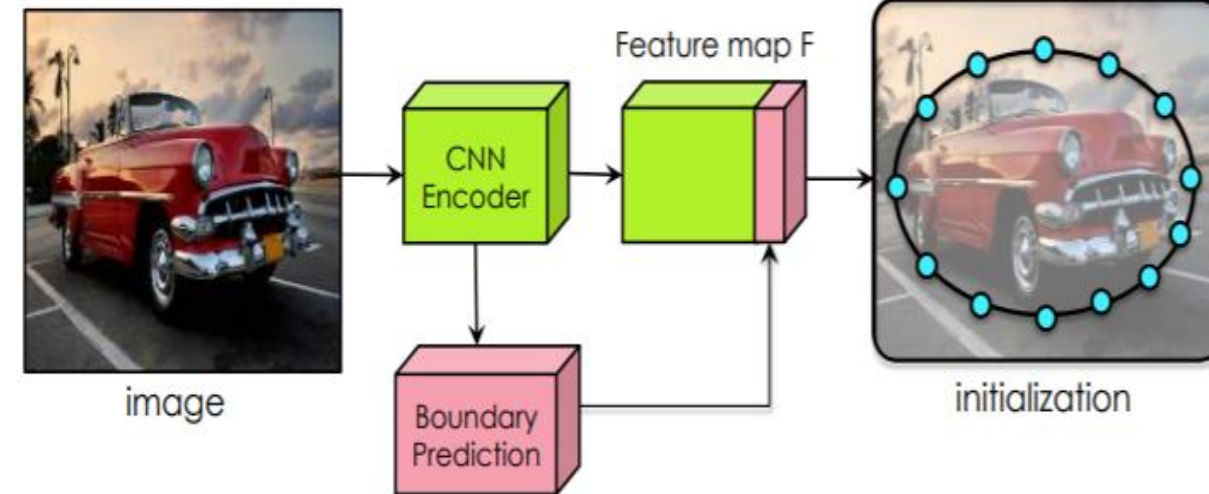
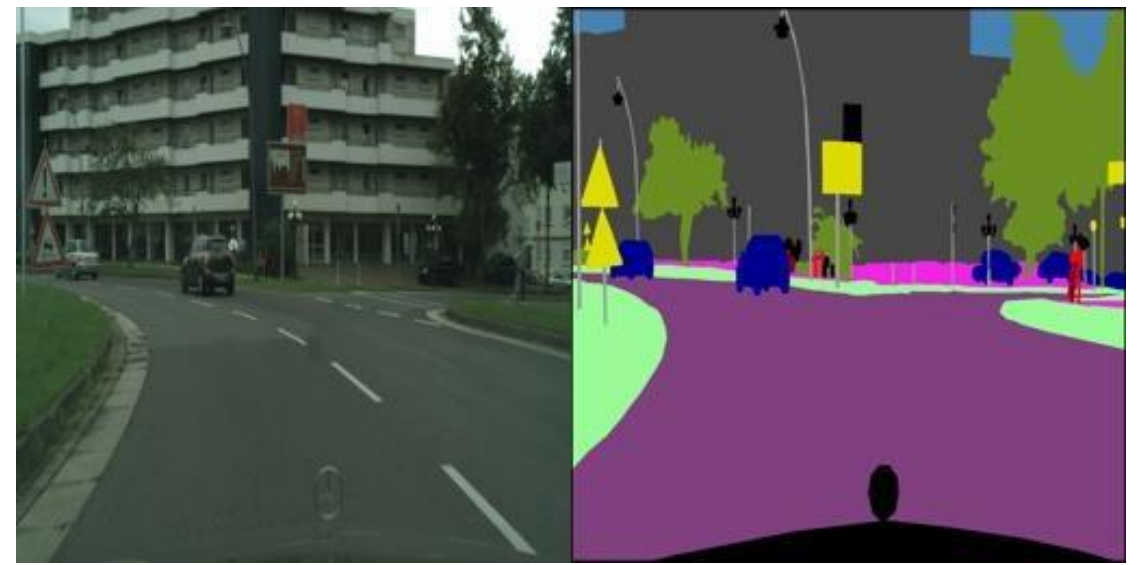
- Download the Cityscapes dataset (leftImg8bit\_trainvaltest.zip) from the official.
- <https://www.cityscapes-dataset.com/downloads/>
- Processed annotation files from <http://www.cs.toronto.edu/~amlan/data/polygon/cityscapes.tar.gz>
- Using CNN Encoder and Boundary Prediction.  
Annotation file → extract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

## ❖ Compute feature map by using ResNet50V2 model.

- `feature_map = ResNet50V2.prediction(bbox)`

## ❖ Define graph.

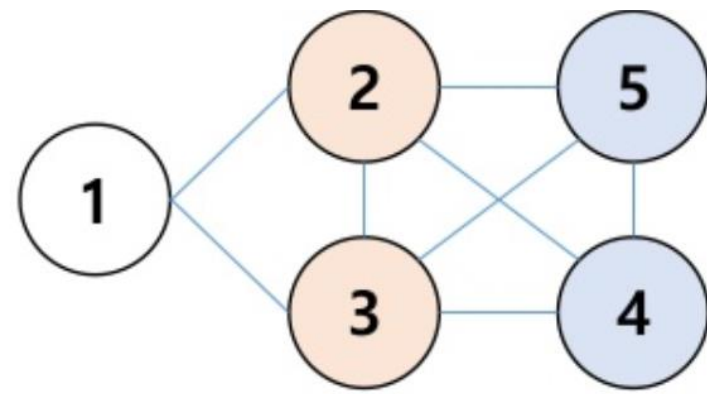
- Initial  $N = 45 \rightarrow$  Calculate feature of each node.





# 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

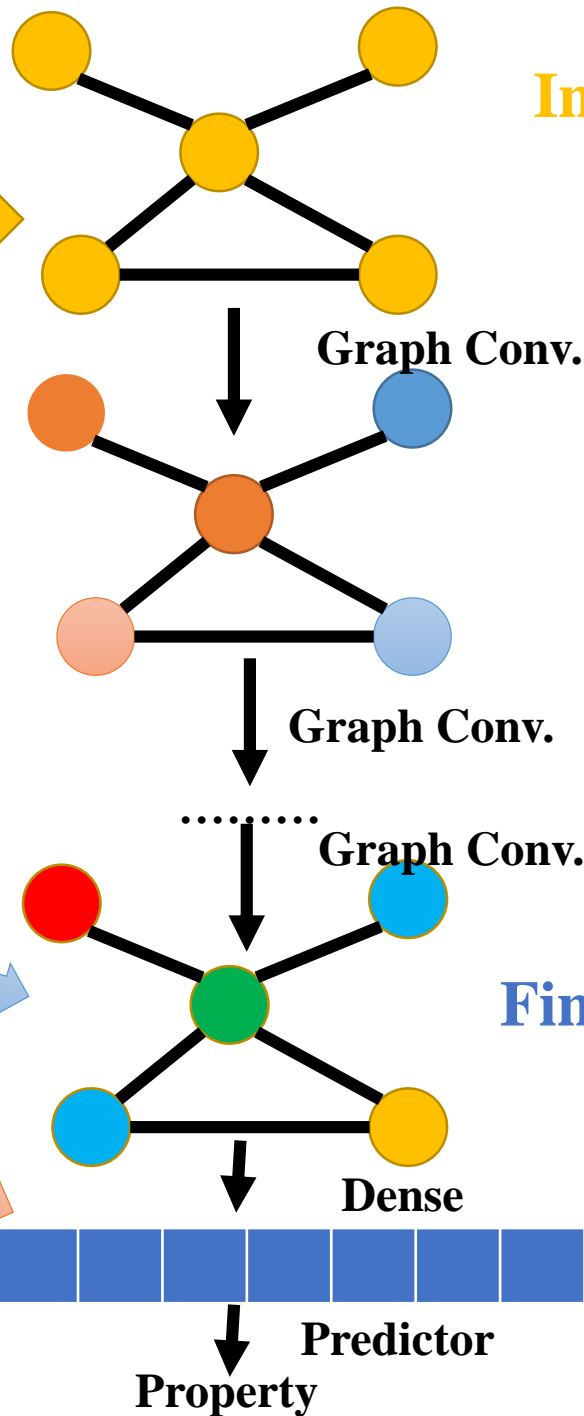
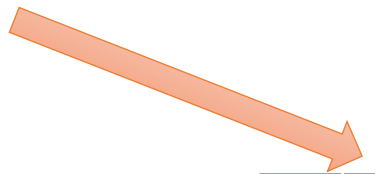
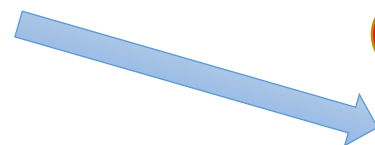
Graph Conv, 32

Graph Conv, 64

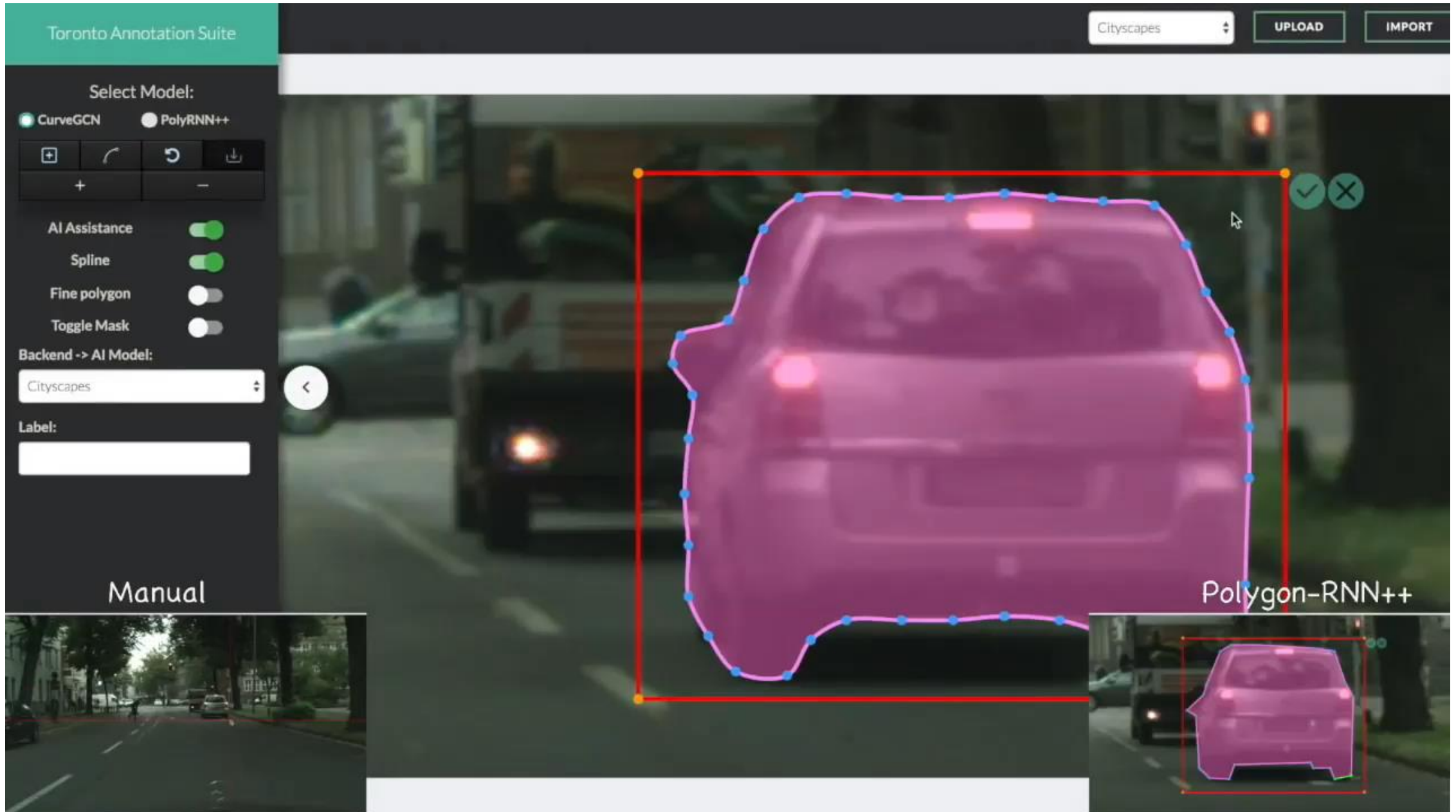
Dense, 64

Predictor, 1

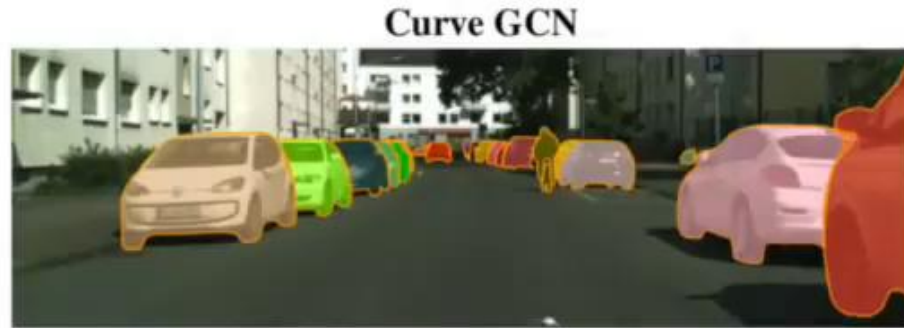
Labels



# Experimental Results (Author's Implementation)



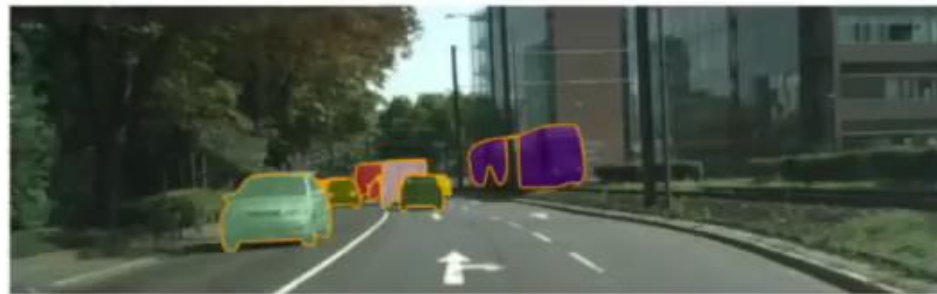
# Experimental Results (Author's Implementation)



0 clicks



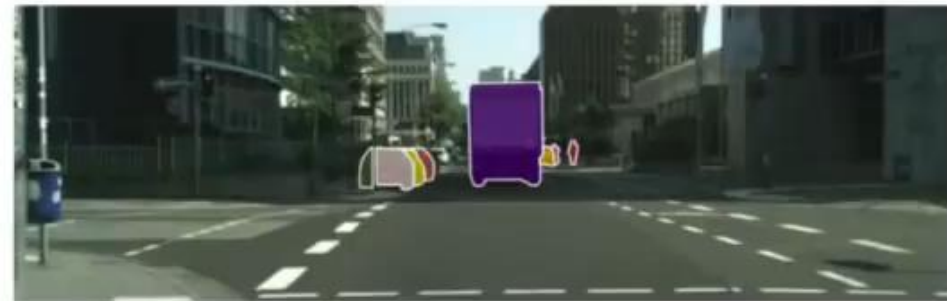
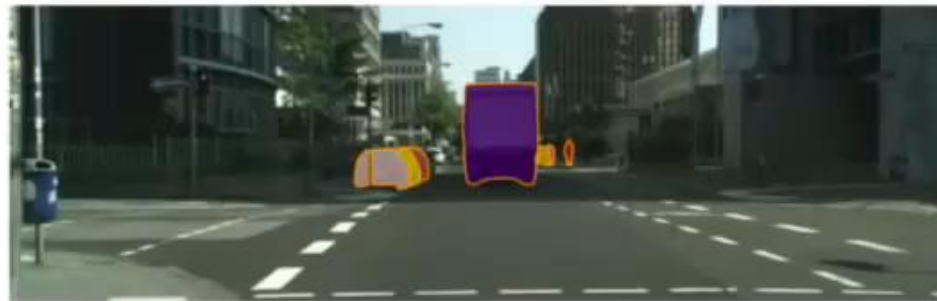
700 clicks



0 clicks



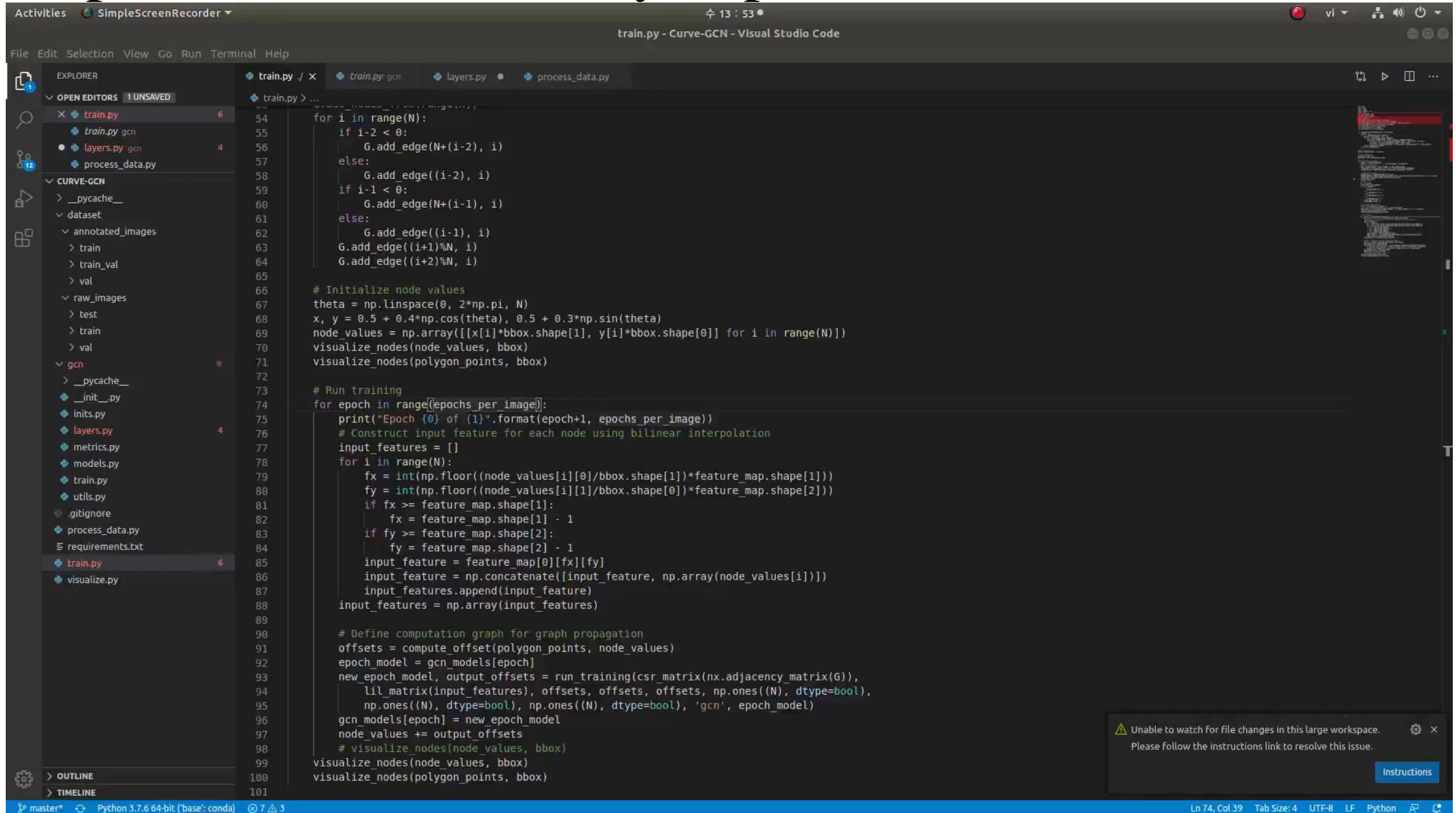
286 clicks



Curve-GCN in 0-click regime



# Experimental Results (My implementation)



```
train.py
train.py gcn
layers.py gcn
process_data.py

CURVE-GCN
__pycache__
dataset
annotated_images
train
train_val
val
raw_images
test
train
val
gcn
__pycache__
__init__.py
inits.py
layers.py
metrics.py
models.py
train.py
utils.py
.gitignore
process_data.py
requirements.txt
train.py
visualize.py

OUTLINE
TIMELINE

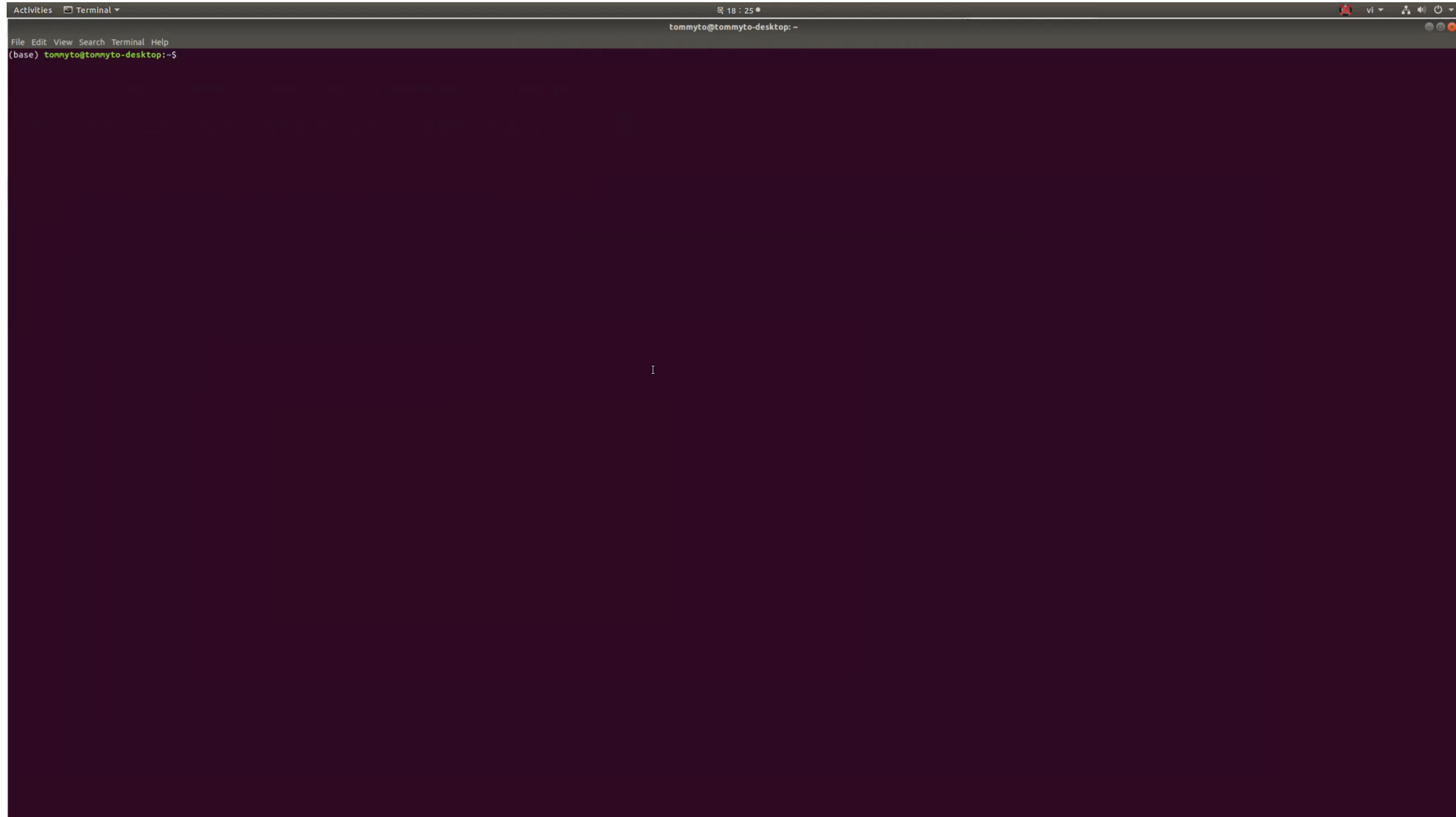
train.py
54 for i in range(N):
55     if i-2 < 0:
56         G.add_edge(N+(i-2), i)
57     else:
58         G.add_edge((i-2), i)
59     if i-1 < 0:
60         G.add_edge(N+(i-1), i)
61     else:
62         G.add_edge((i-1), i)
63     G.add_edge((i+1)%N, i)
64     G.add_edge((i+2)%N, i)
65
66 # Initialize node values
67 theta = np.linspace(0, 2*np.pi, N)
68 x, y = 0.5 + 0.4*np.cos(theta), 0.5 + 0.3*np.sin(theta)
69 node_values = np.array([[x[i]*bbox.shape[1], y[i]*bbox.shape[0]] for i in range(N)])
70 visualize_nodes(node_values, bbox)
71 visualize_nodes(polygon_points, bbox)
72
73 # Run training
74 for epoch in range(epochs_per_image):
75     print("Epoch {0} of {1}".format(epoch+1, epochs_per_image))
76     # Construct input feature for each node using bilinear interpolation
77     input_features = []
78     for i in range(N):
79         fx = int(np.floor((node_values[i][0]/bbox.shape[1])*feature_map.shape[1]))
80         fy = int(np.floor((node_values[i][1]/bbox.shape[0])*feature_map.shape[2]))
81         if fx >= feature_map.shape[1]:
82             fx = feature_map.shape[1] - 1
83         if fy >= feature_map.shape[2]:
84             fy = feature_map.shape[2] - 1
85         input_feature = feature_map[0][fx][fy]
86         input_feature = np.concatenate([input_feature, np.array(node_values[i])])
87         input_features.append(input_feature)
88     input_features = np.array(input_features)
89
90 # Define computation graph for graph propagation
91 offsets = compute_offset(polygon_points, node_values)
92 epoch_model = gcn_models[epoch]
93 new_epoch_model, output_offsets = run_training(csr_matrix(nx.adjacency_matrix(G)),
94         lil_matrix(input_features), offsets, offsets, offsets, np.ones((N), dtype=bool),
95         np.ones((N), dtype=bool), np.ones((N), dtype=bool), 'gcn', epoch_model)
96     gcn_models[epoch] = new_epoch_model
97     node_values += output_offsets
98     # visualize nodes(node_values, bbox)
99     visualize_nodes(node_values, bbox)
100     visualize_nodes(polygon_points, bbox)
101
```

Unable to watch for file changes in this large workspace. Please follow the instructions link to resolve this issue. [Instructions](#)

Ln 74, Col 39 Tab Size: 4 UTF-8 LF Python

# Implementation Link

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



Thank you  
for  
your listening