Fast Interactive Object Annotation with Curve-GCN

Paper by: Huan Ling* [1,2], Jun Gao* [1,2], Amlan Kar[1,2], Wenzheng Chen[1,2], Sanja Fidler[1,2,3]
[1] University of Toronto [2] Vector Institute [3] 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 <u>end-to-end</u> 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++

Polygon

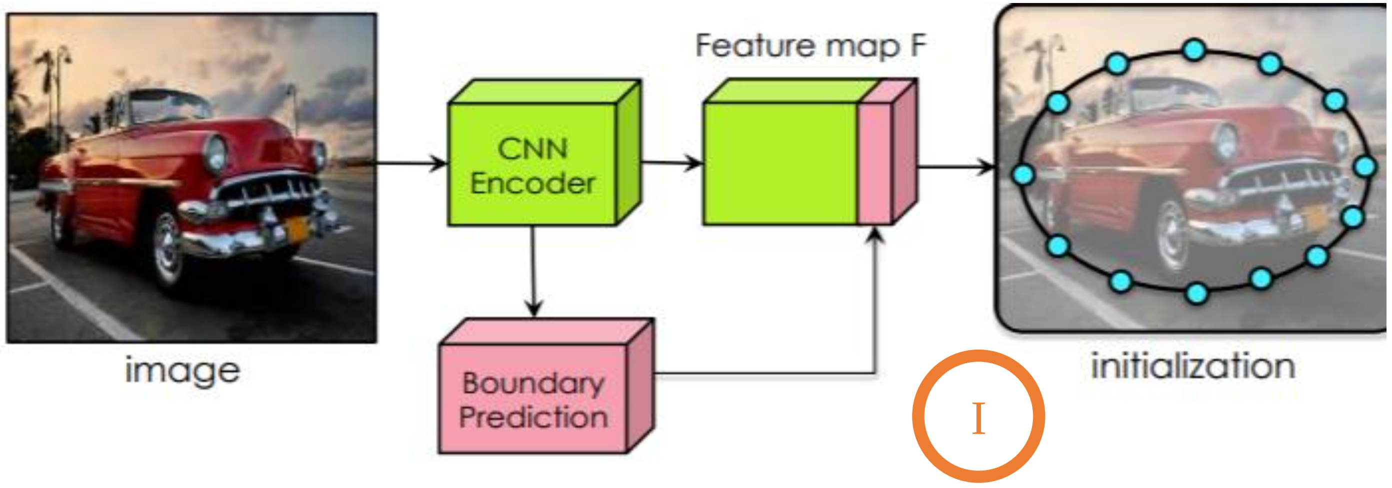
Spline

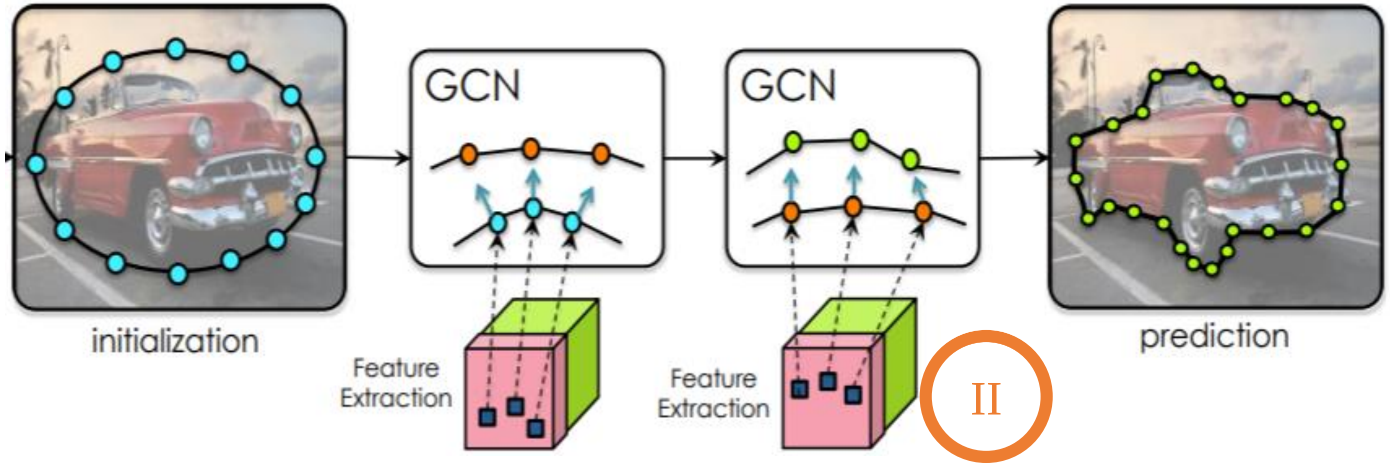
Curve-GCN System Overview

❖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***



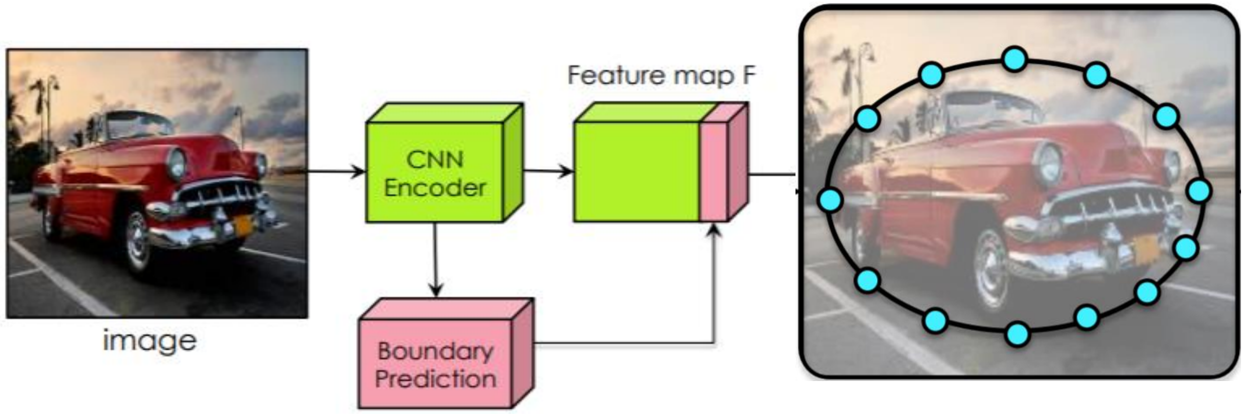


Define GCN Graph ***GCN Model.*** ***Interactive-GCN*** ***Prediction***

Curve-GCN Graph Definitions.

Let:

- $cp_i = [x_i, y_i]^T$ denote the location of i-th control point.

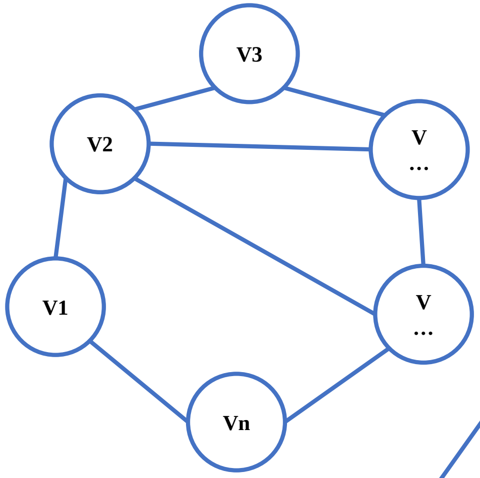


- V = $\{cp_0, cp_1, \ldots, cp_{N-1}\}$ be the set of all control points.

→They define a graph.

G = (V,E).

V are the nodes. E are the edges.



Labelled Graph

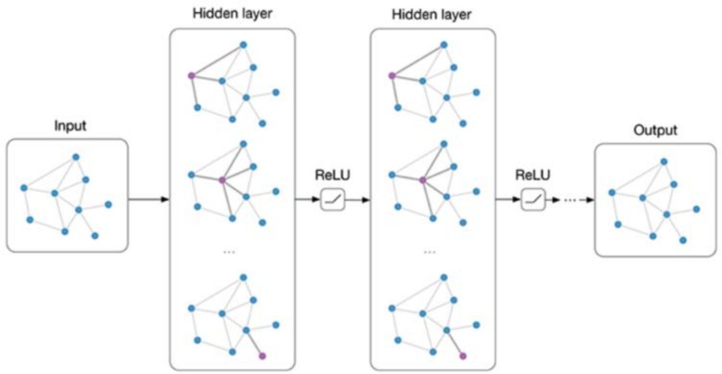
Degree Matrix

Adjacency Matrix

Laplacian Matrix

From Graph → 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



- $N(cp_i)$ denotes the nodes that are connected to cp_i in the graph.
- w_0^l, w_1^l are weight matrices.

❖Features extracted from the corresponding location in map F:

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

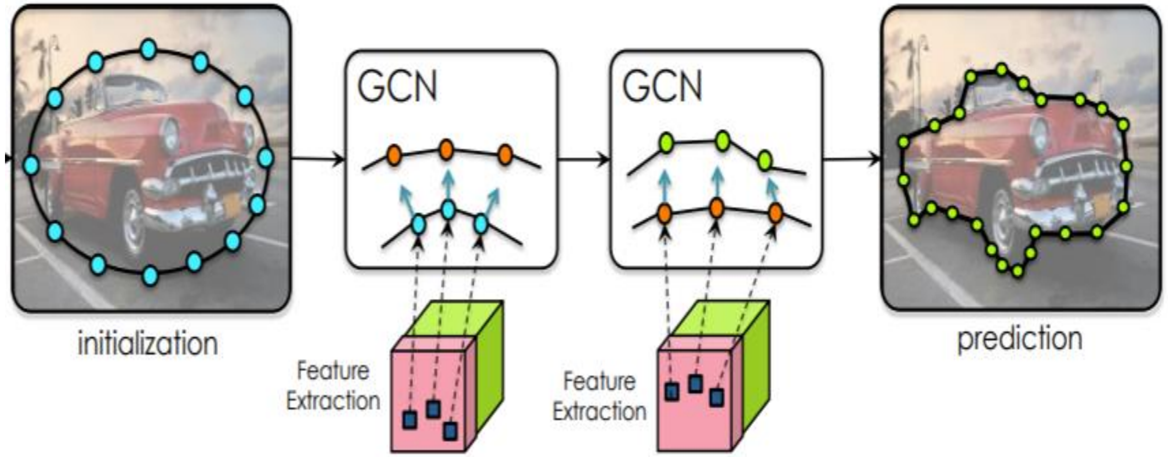
❖We utilize a Graph-ResNet to propagate information between nodes in the graph as a residual function.

❖Then takes the following form.

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

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

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



Interactive GCN (Annotator in loop).

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

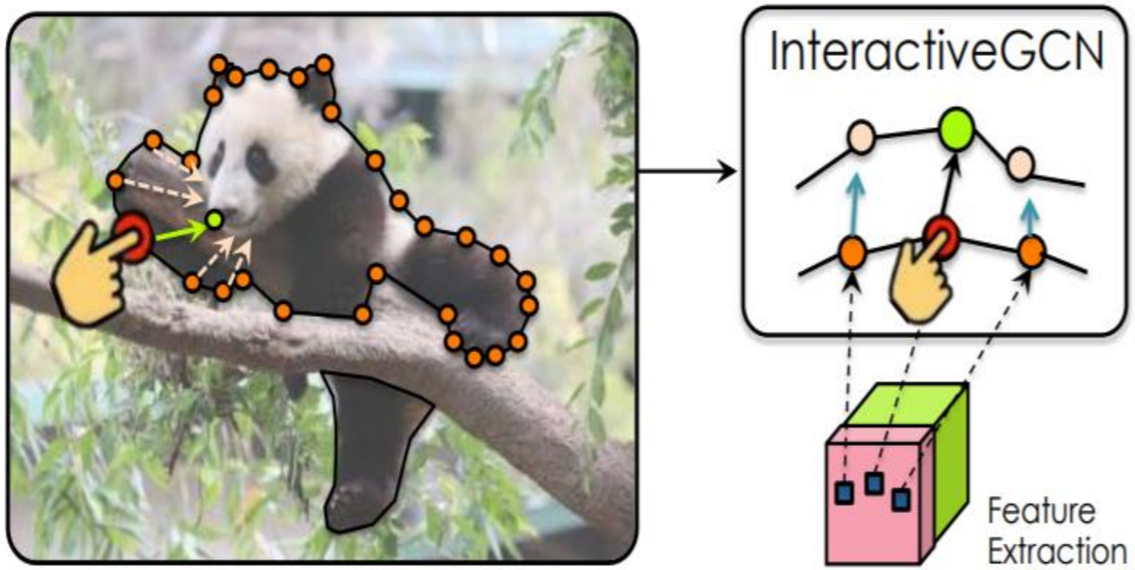
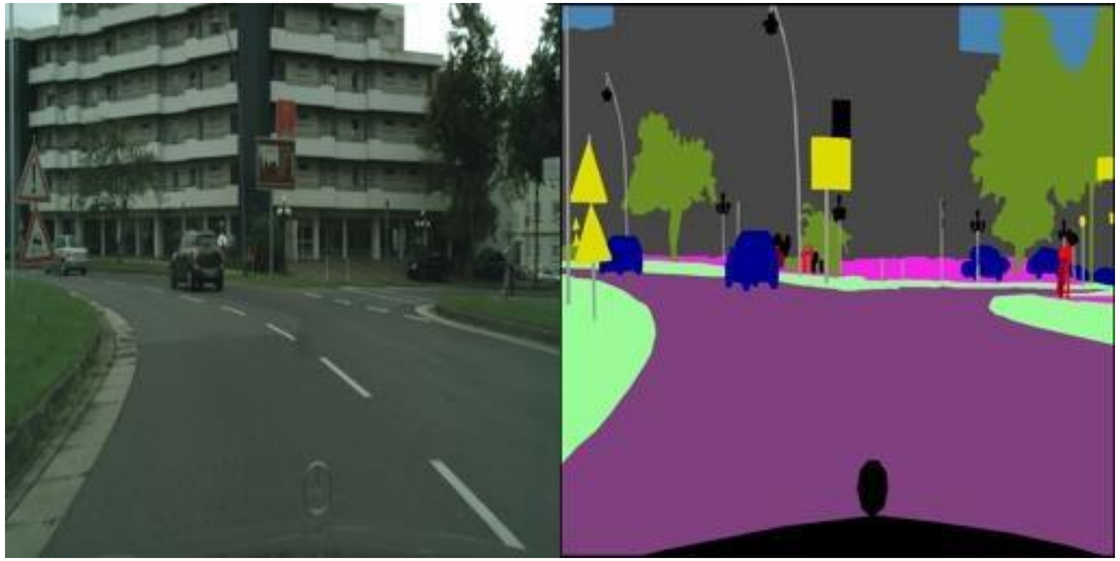


Figure 4: **Human-in-the-Loop**: An annotator can choose any wrong control point and move it onto the boundary. Only its immediate neighbors ($k = 2$ in our experiments) will be re-predicted based on this interaction.

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

```
while not converged do
    (rawImage, gtCurve) = Sample(Dataset)
    (predCurve, F) = Predict(rawImage)
    data = []
    for i in range(c) do
        corrPoint = Annotator(predictedCurve)
        data += (predCurve, corrPoint, gtCurve, F)
        predCurve = InteractiveGCN(predCurve, corrPoint)
                                        ▷ Do not stop gradients
    TrainInteractiveGCN(data)
```

Experimental.



❖Dataset Preparation.

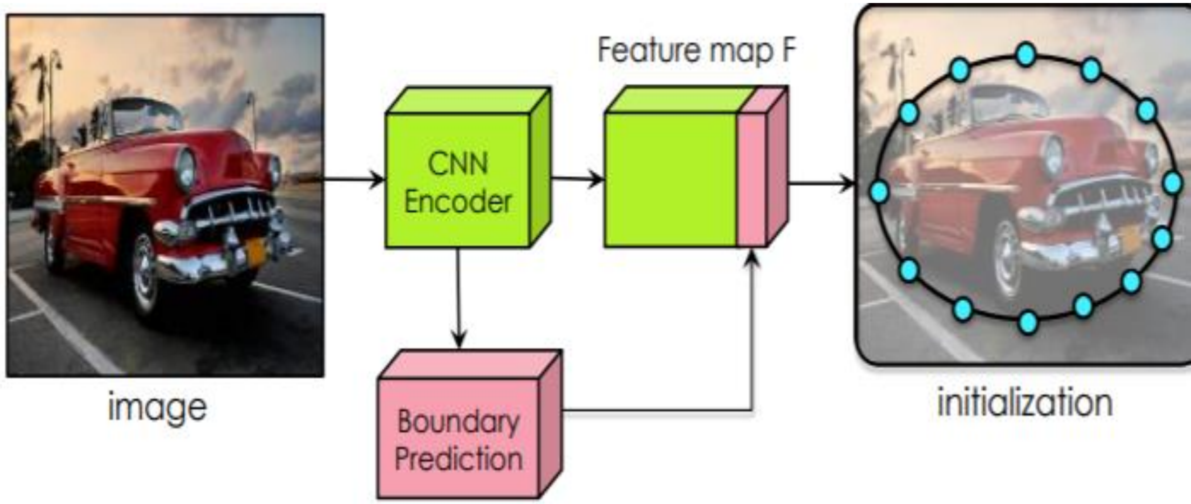
- Download the Cityscapes dataset (leftImg8bit_trainvaltest.zip) from the official.
- https://www.cityscapes-dataset.com/downloads/
- Processed annotation files from http://www.cs.toronto.edu/~amlan/data/polygon/cityscapes.tar.gz
- Using CNN Encoder and Boundary Prediction. Annotation file → extract bbox, polygon_points.

❖Compute feature map by using ResNet50V2 model.

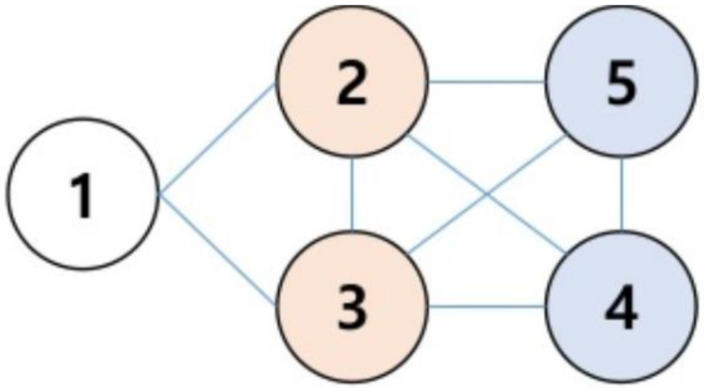
- feature_map = ResNet50V2.prediction(bbox)

❖Define graph.

- Initial N = 45 → 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)

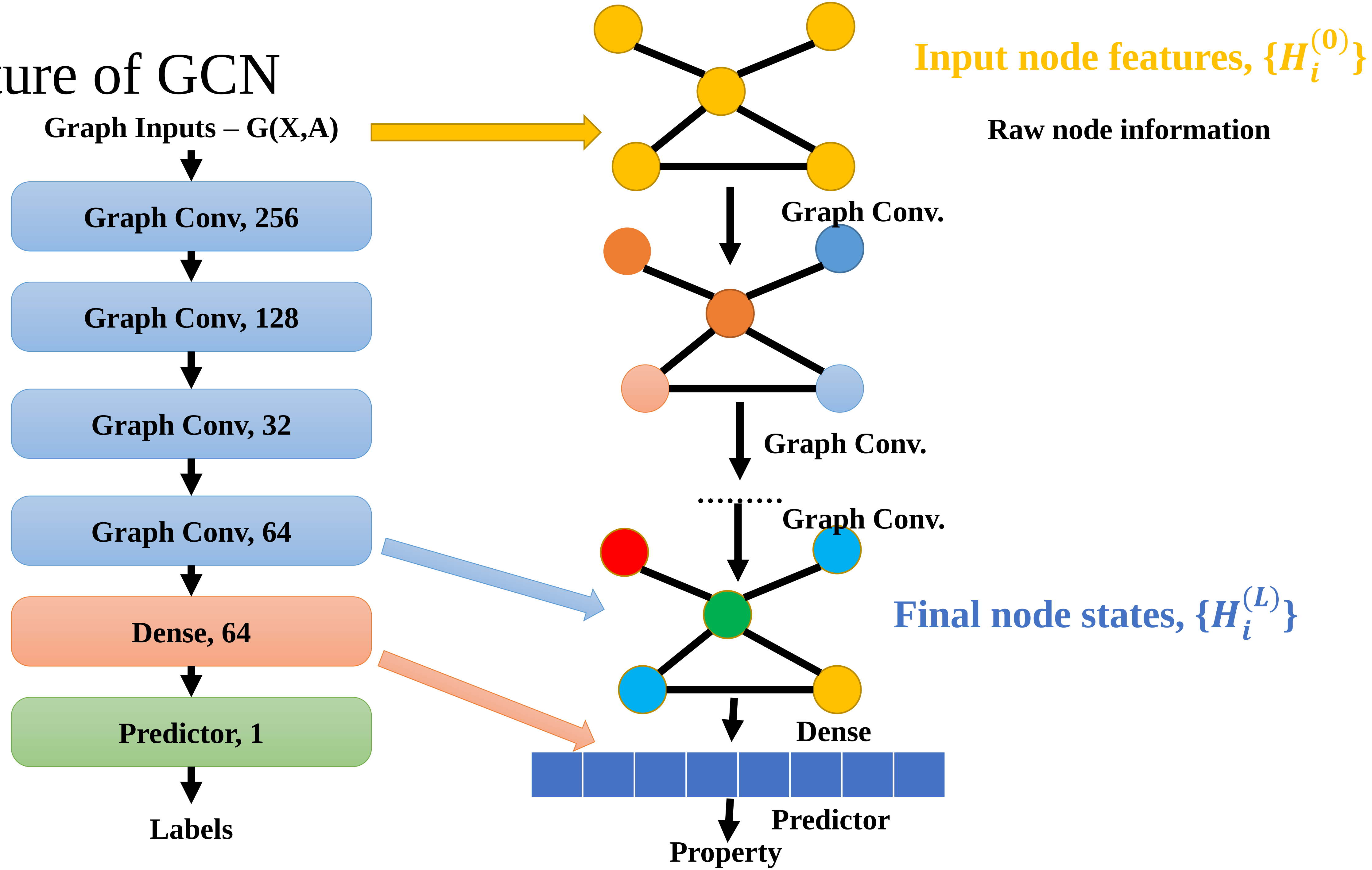
	Feat. 1	Feat. 2	Feat. 3	Feat. 4	Feat. 5
Node 1	1	1	1	0	0
Node 2	1	1	1	1	1
Node 3	1	1	1	1	1
Node 4	0	1	1	1	1
Node 5	0	1	1	1	1

Building Model

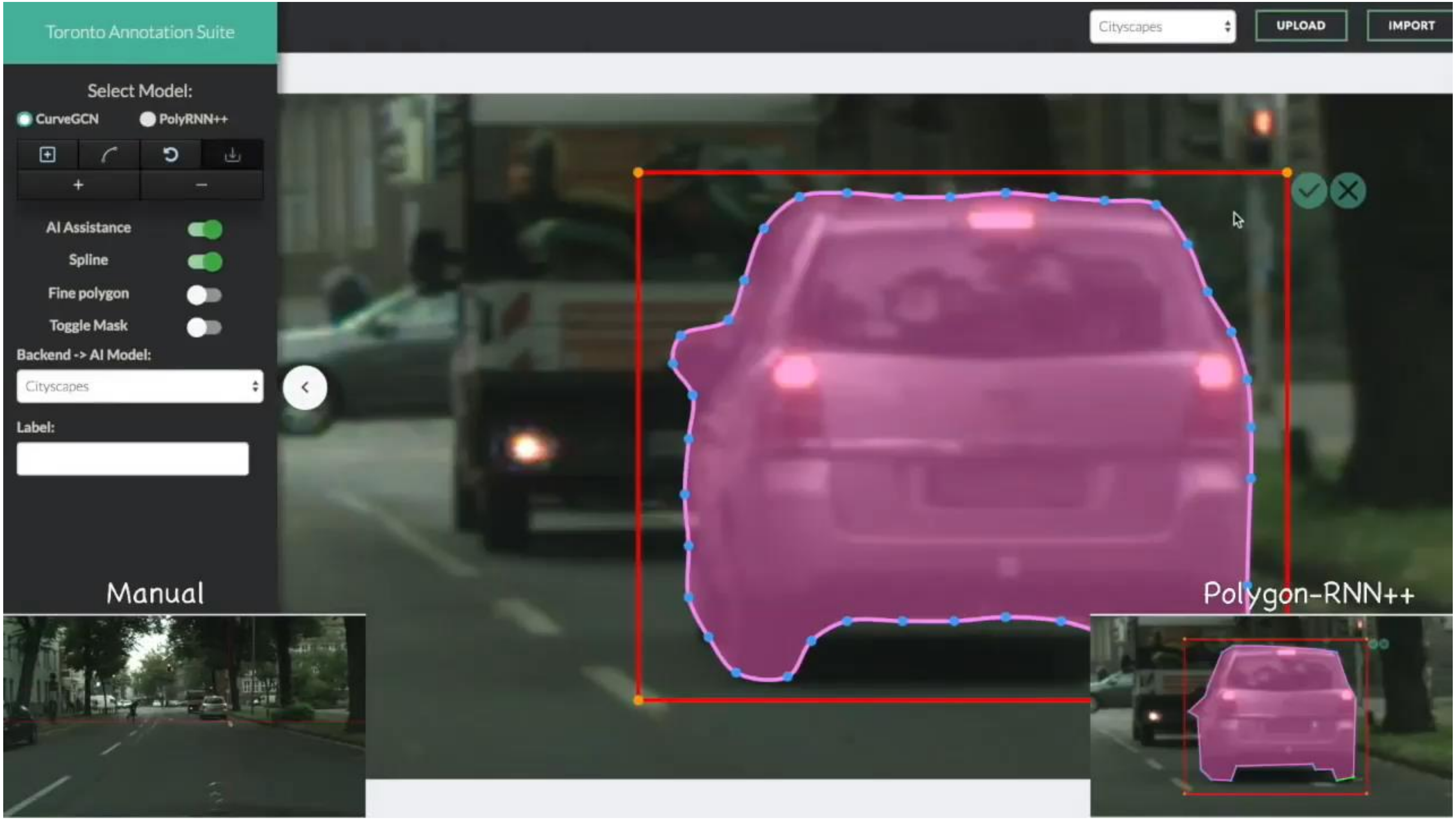
```
# Compute feature map
resnet_model = ResNet50V2(weights='imagenet')
embedding_model = Model(inputs=resnet_model.input, outputs=resnet_model.get_layer('post_relu').output)
feature_map = embedding_model.predict(resized_bb_exp)
print(feature_map)
# Define graph
N = 45
G = nx.Graph()
G.add_nodes_from(range(N))
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)
```

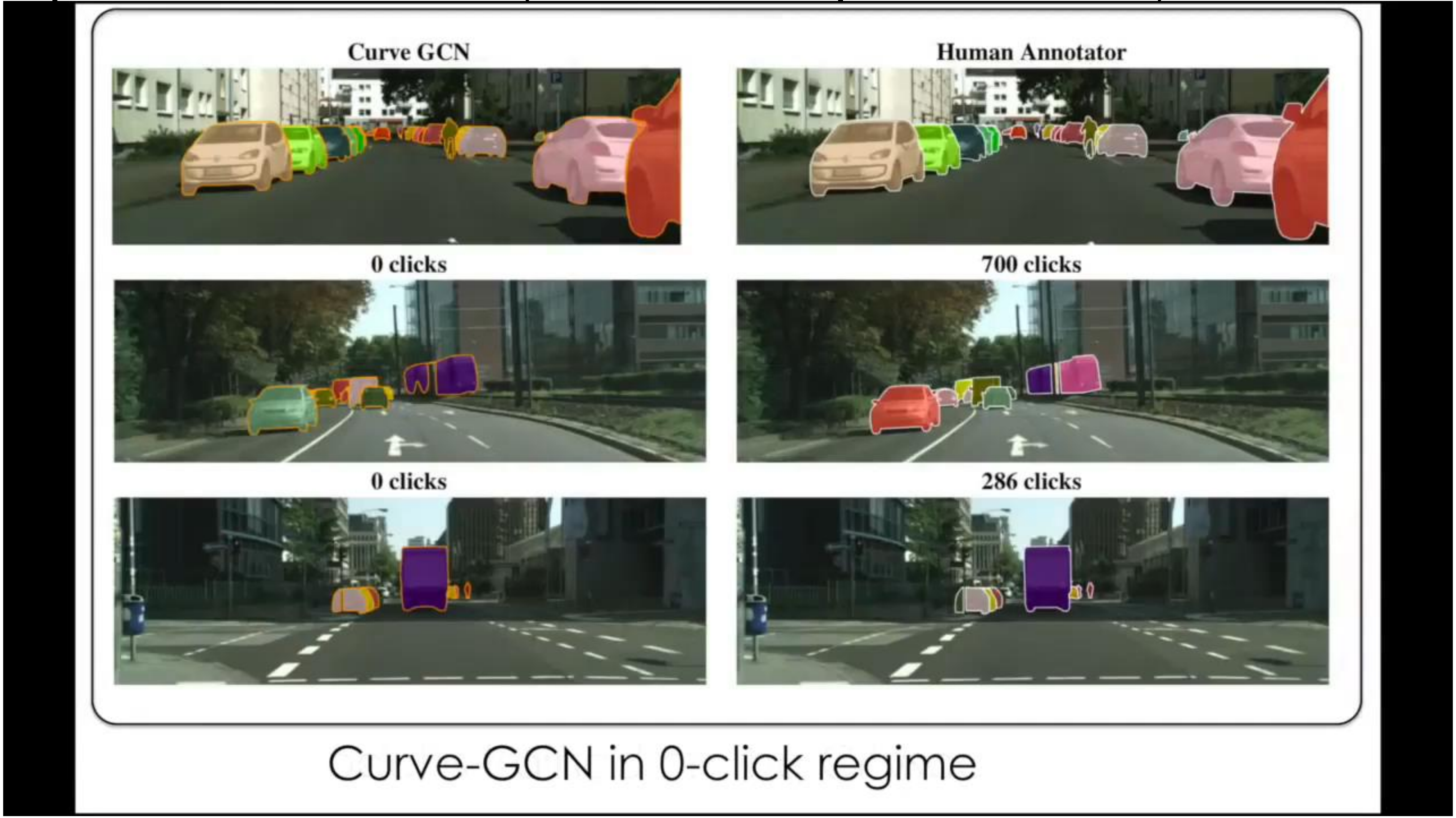
Structure of GCN



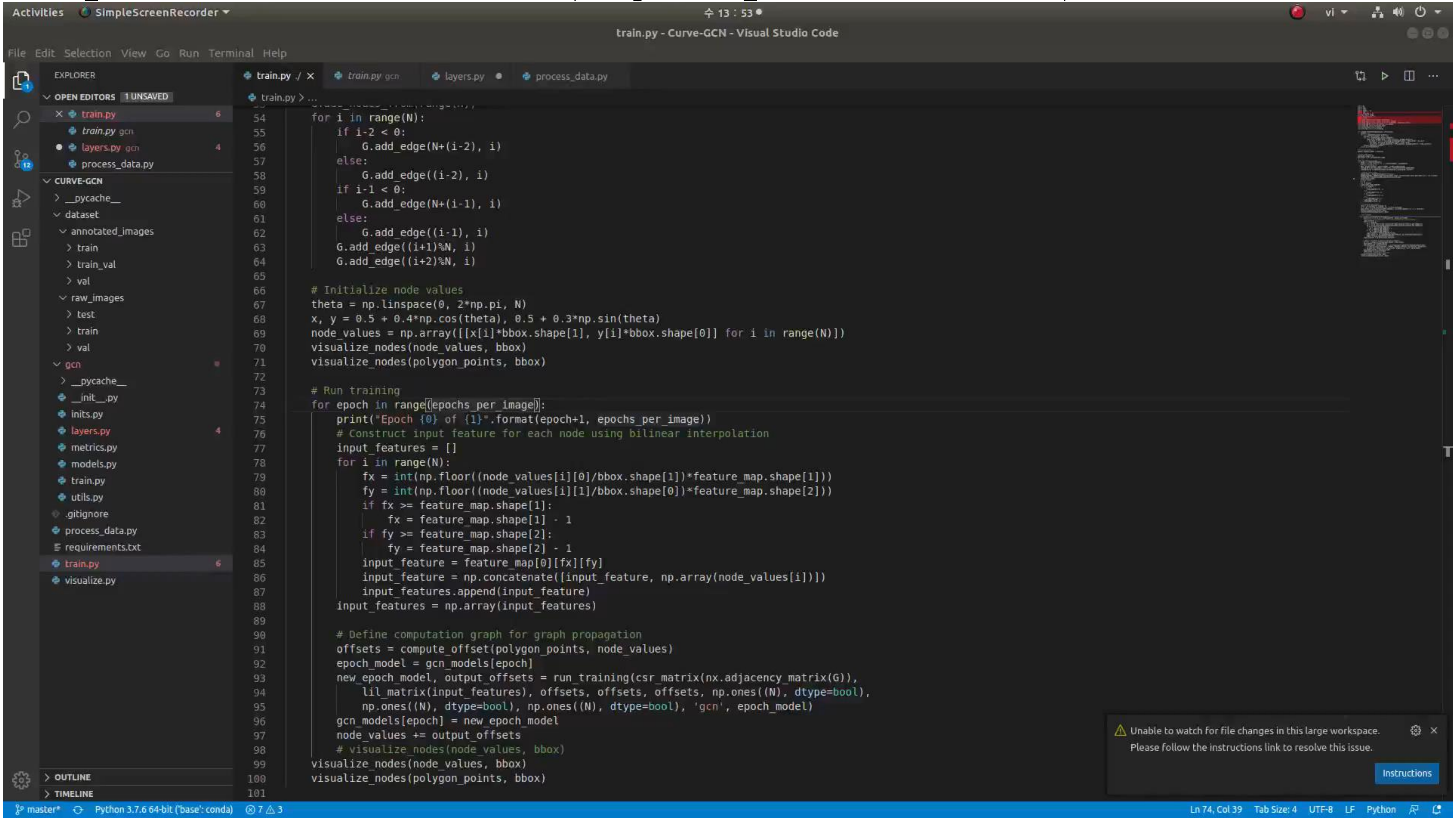
Experimental Results (Author's Implementation)



Experimental Results (Author's Implementation)

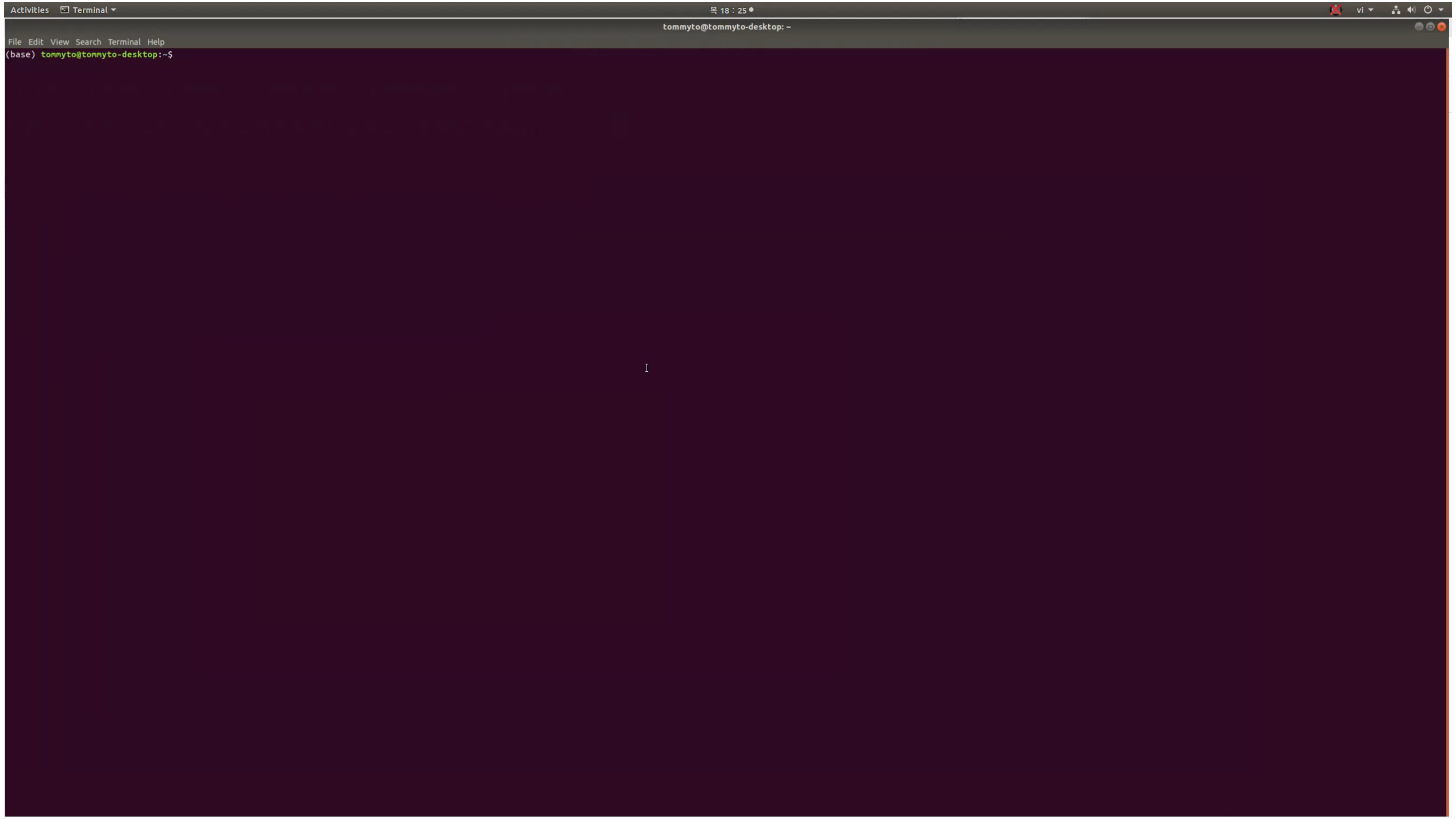


Experimental Results (My implementation)



Implementation Link

https://github.com/haithienld/Curve_GCN-Implementation



Thank you
for
your listening