CSL7590 : Deep Learning

Assignment Report 03

Name: **Shantanu Sharma**
Roll Number: **M24IRM006**
Program: **Robotics and Mobility system**

Name: **Neha Sharma**
Roll Number: **M24CSE014**
Program: **Computer Science & Engineering**

Name: **Prachi Sahu**
Roll Number: **M24MAC005**
Program: **Data and computational Science**

Sequential Model for Sketch Generation

1. Task

To develop a generative sequence model capable of synthesizing sketches corresponding to specified object classes from the QuickDraw dataset.

2. Objective

Implement and train an LSTM-based neural network using PyTorch to generate 2D stroke sequences (x, y coordinates) representing sketches for predefined classes ("apple", "eye", "square", "triangle", "circle"). The model should learn the sequential patterns of strokes for each class.

3. Data Set

1. **Source:** Google QuickDraw Dataset.
2. **Selected Classes:** "apple", "eye", "square", "triangle", "circle".
3. **Format:** Newline Delimited JSON (.ndjson), with each record containing stroke data for one drawing. Strokes are initially provided as relative (delta) coordinates.
4. **Preprocessing:**
    a. Downloaded .ndjson files for the 5 selected classes.
    b. Converted relative stroke coordinates to absolute (x, y) coordinates using a *strokes_to_absolute* function.
    c. Limited data to 100000 recognized samples per class for balanced training.
5. **Data Splitting:**
    a. Training Set: 70%
    b. Validation Set: 15%
    c. Test Set: 15%

4. Libraries Used:

1. torch.fx
2. torch.nn (as nn)
3. torch.optim (as optim)
4. torch.utils.data (specifically Dataset, DataLoader)
5. numpy (as np)
6. json
7. os
8. requests
9. matplotlib.pyplot (as plt)
10. matplotlib.animation (specifically FuncAnimation)
11. sklearn.model_selection (specifically train_test_split)
12. google.colab (specifically files)
13. torch

5. Network Architecture: Sketch Generator LSTM

1. **Input:** Integer representing the class index.
2. **Embedding Layer:** Maps the class index to a dense embedding vector. The embedding dimension (*EMBEDDING_DIM*) is set to **128**.
3. **LSTM Layer:**
    a. Processes the sequence of stroke coordinates.
    b. Input to LSTM cell at each timestep: Concatenation of the class embedding vector and the previously predicted stroke coordinates (x, y).
    c. Generates hidden states capturing sequential stroke information. The LSTM hidden size (*HIDDEN_SIZE*) is set to **256**.
4. **Dropout Layer:** Applied after the LSTM layer to mitigate overfitting. The dropout rate (*DROPOUT_RATE*) is set to **0.3**. This is used in both the LSTM layer definition (*dropout=dropout_rate*) and an explicit *nn.Dropout(dropout_rate)* layer within the SketchGenerator model.

5. **Fully Connected Layer:** Maps the LSTM output (hidden state) to the predicted stroke coordinates for the next timestep. Output dimension is 3 that is the stroke dimension is 3 (for x , y and pen state).
6. **Activation Functions: LSTM Layer** *(nn.LSTM)*: Internally, the standard PyTorch LSTM implementation uses:
    a. **Sigmoid** activation for its gates (input, forget, output gates).
    b. **Tanh** activation for the cell state and hidden state updates. These are standard internal components of the LSTM cell and are not explicitly added as separate layers in your code.

6. Initialization, Loss Function, and Optimizer

1. **Weight Initialization:** Xavier uniform initialization applied to relevant layers (e.g., LSTM, Fully Connected).
2. **Loss Function:** Mean Squared Error (MSE) Loss. Calculates the difference between the predicted (x, y) stroke coordinates and the ground truth coordinates.
3. **Optimizer:** Adam optimizer.
    a. Learning Rate: 0.001.
4. **Learning Rate Scheduling:**
    a. A scheduler reduces the learning rate when the validation loss stops improving (plateaus).
    b. The code uses *torch.optim.lr_scheduler.ReduceLROnPlateau. Specifically: scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=2*)
    c. This scheduler reduces the learning rate when the validation loss ('min') stops improving for a specified number of epochs (patience=2).
5. **Early Stopping:** Training is halted if the validation loss does not improve for 3 consecutive epochs.

7. Training Procedure

1. **Training Loop:** The model is trained iteratively over epochs (30 epochs) using mini-batches from the training set.
2. **Forward Pass:** For each sequence in a batch, the model predicts the next stroke coordinates based on the class embedding and previous strokes.
3. **Loss Calculation:** MSE loss is computed between predictions and actual stroke sequences.
4. **Backpropagation:** Gradients are computed and the Adam optimizer updates model weights.
5. **Validation:** After each epoch, the model's performance is evaluated on the validation set using the MSE loss.
6. **Model Saving:** The model weights yielding the lowest validation loss are saved.

8. Model Evaluation and Results

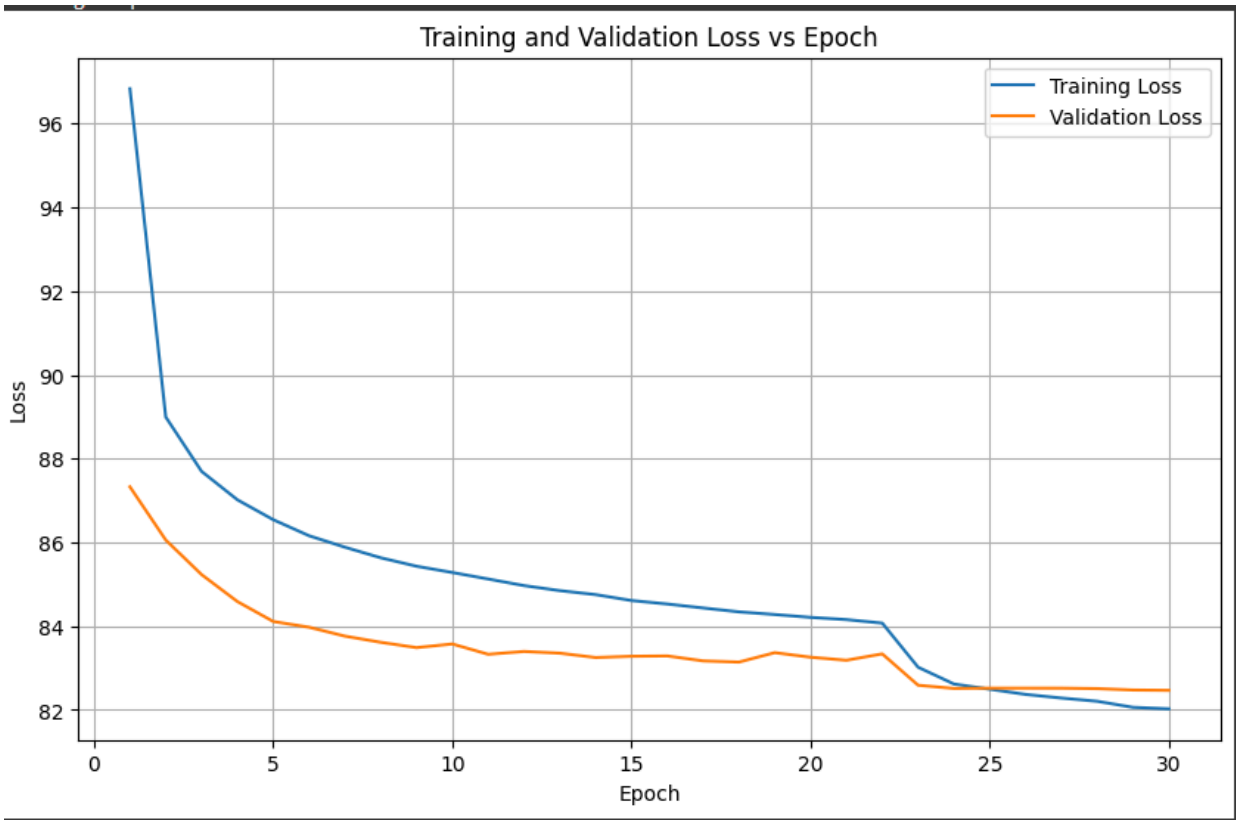
1. **Evaluation Metric:** Primarily qualitative visual assessment on the unseen test set.
2. **Procedure:**
    a. The trained model generates sketches for one sample from each of the 5 classes in the test set.
    b. The generation process involves feeding the class index and iteratively predicting the next stroke based on the previous one.
3. **Output:**
    a. Generated sketches are saved as GIF animations, showing the drawing sequence stroke by stroke.
    b. Static PNG images comparing the final generated sketch side-by-side with the actual sketch from the test set are also produced.
4. **Performance Summary:** The model demonstrated the ability to generate recognizable sketches that visually resemble the target class examples from the QuickDraw dataset. The LSTM architecture effectively captured the sequential stroke patterns.

9. Observation and result

Dataset Loading and Sampling (Classes - apple, Eye, square, triangle, circle):

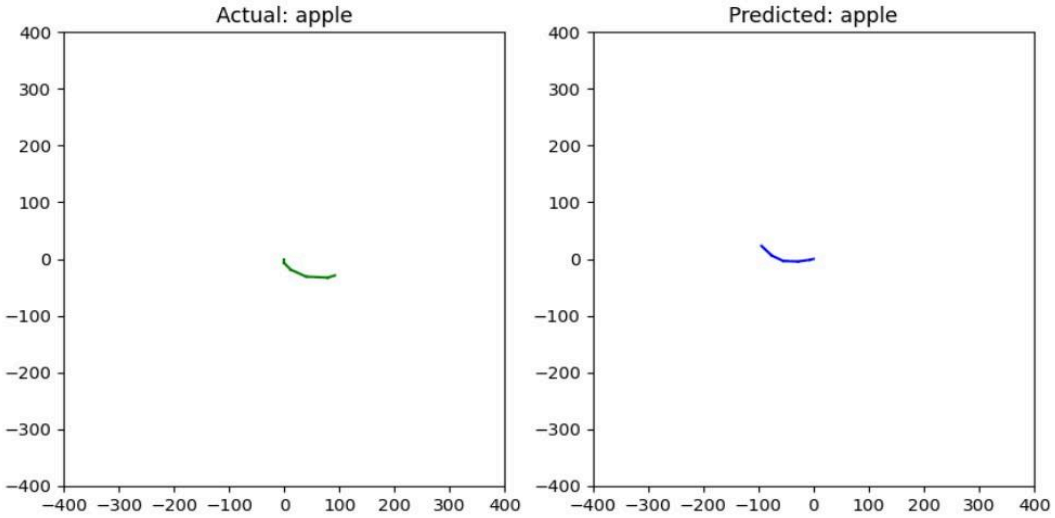
For each class - total 100000 sample are use, which are recognized as the true

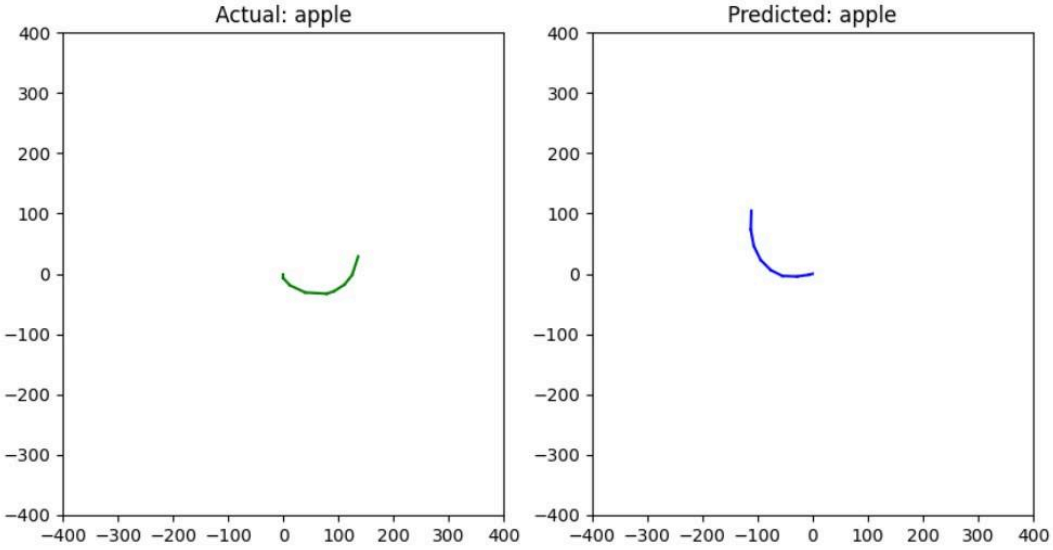
- **Model Evaluation -** The below graph shows the decrease of of the loss after each epoch

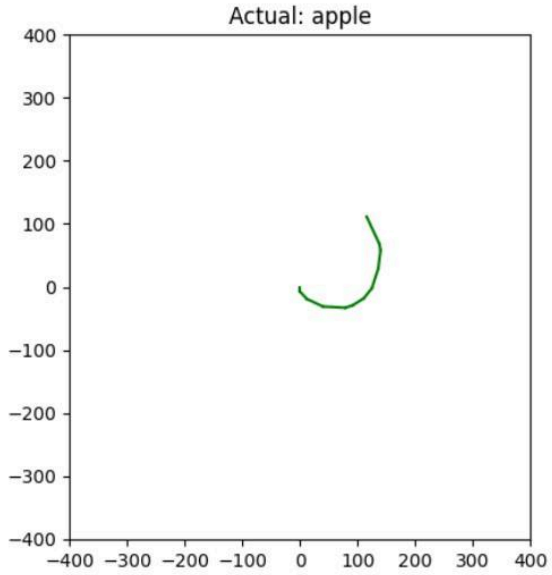


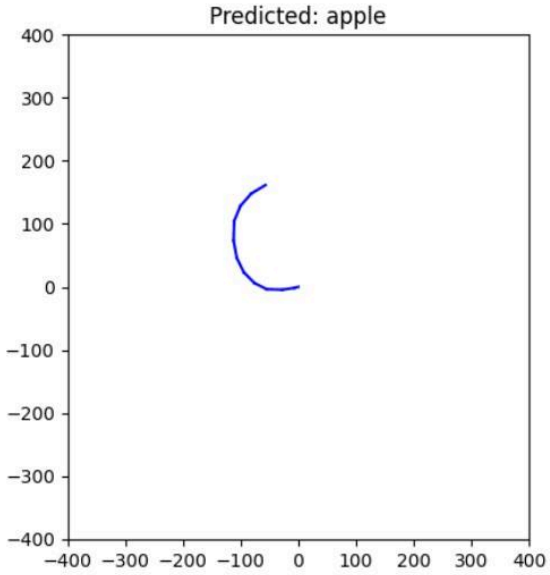
- **Stroke wise formation of the each sketch for each class**

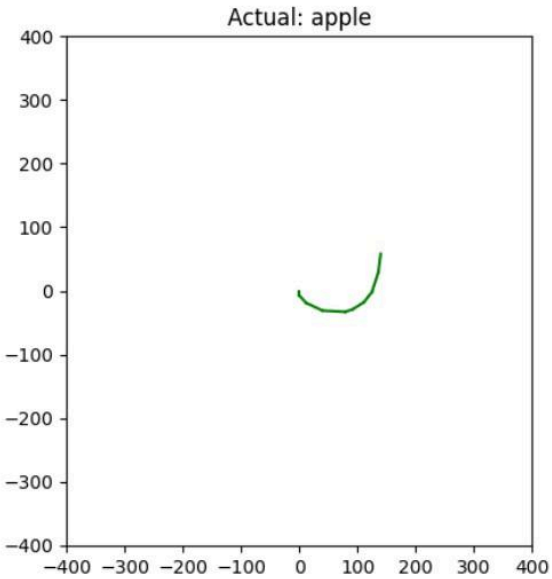
- The stroke-wise representation for Apple

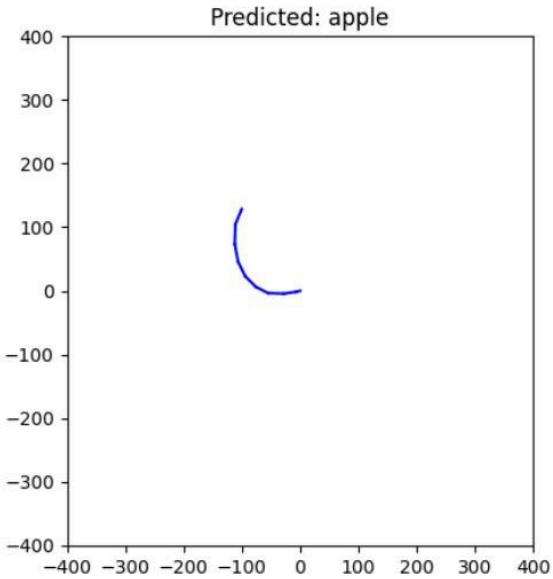


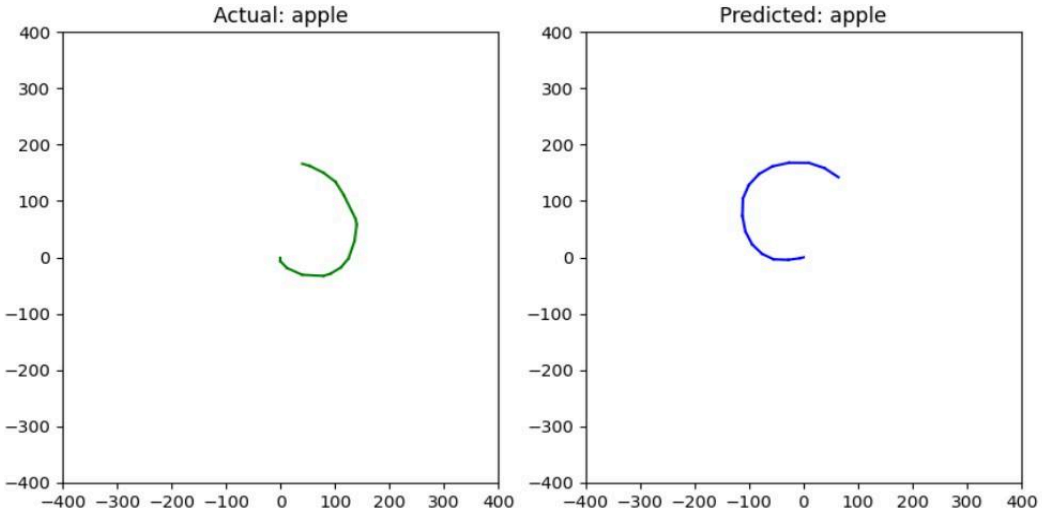


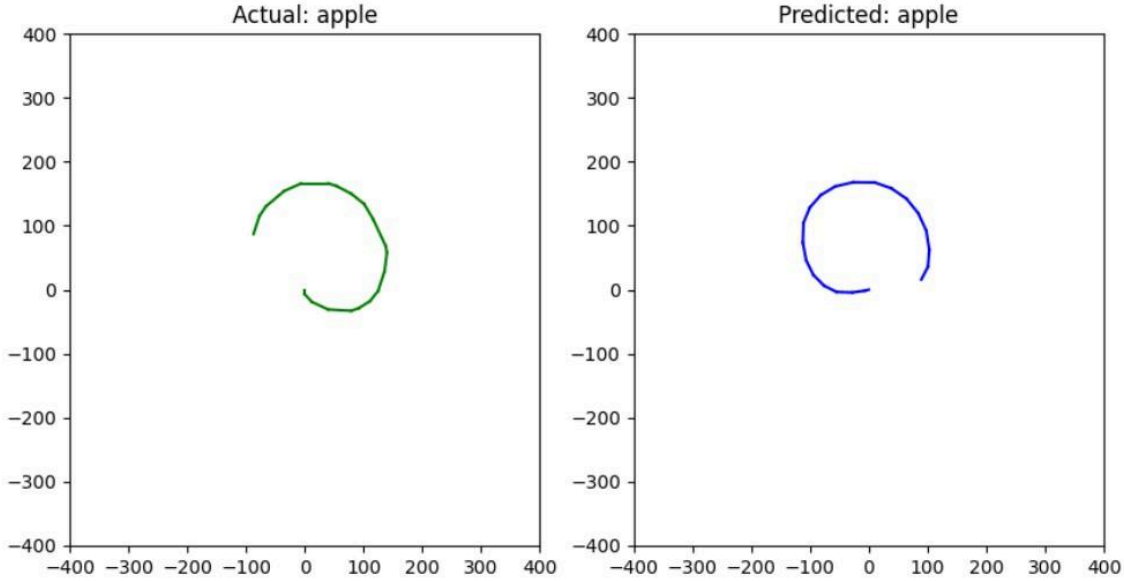

Actual: apple
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

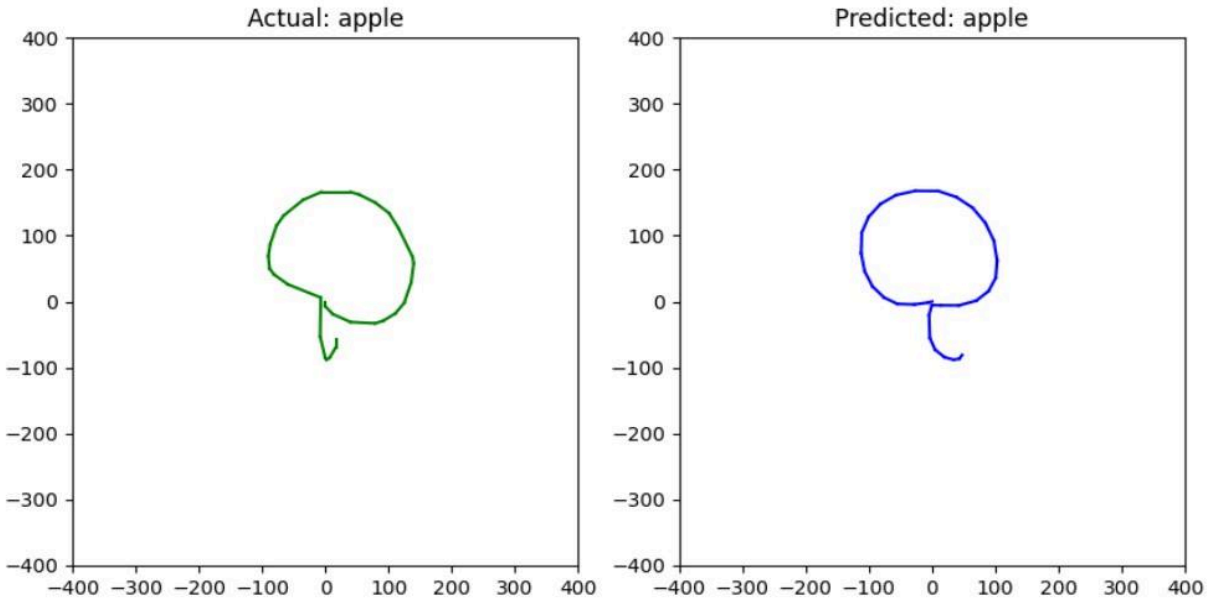

Predicted: apple
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

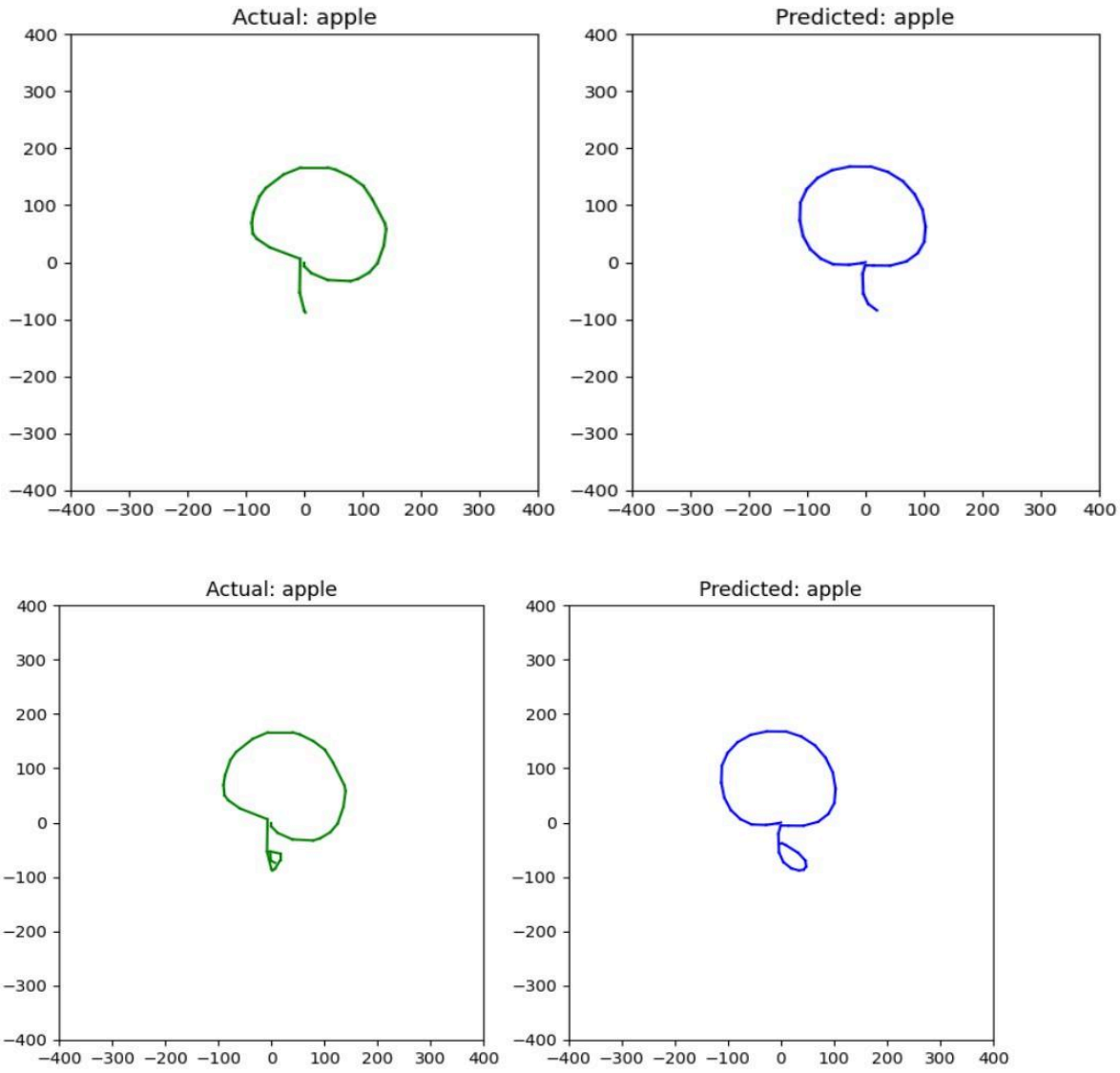

Actual: apple
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: apple
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Actual: apple
Predicted: apple

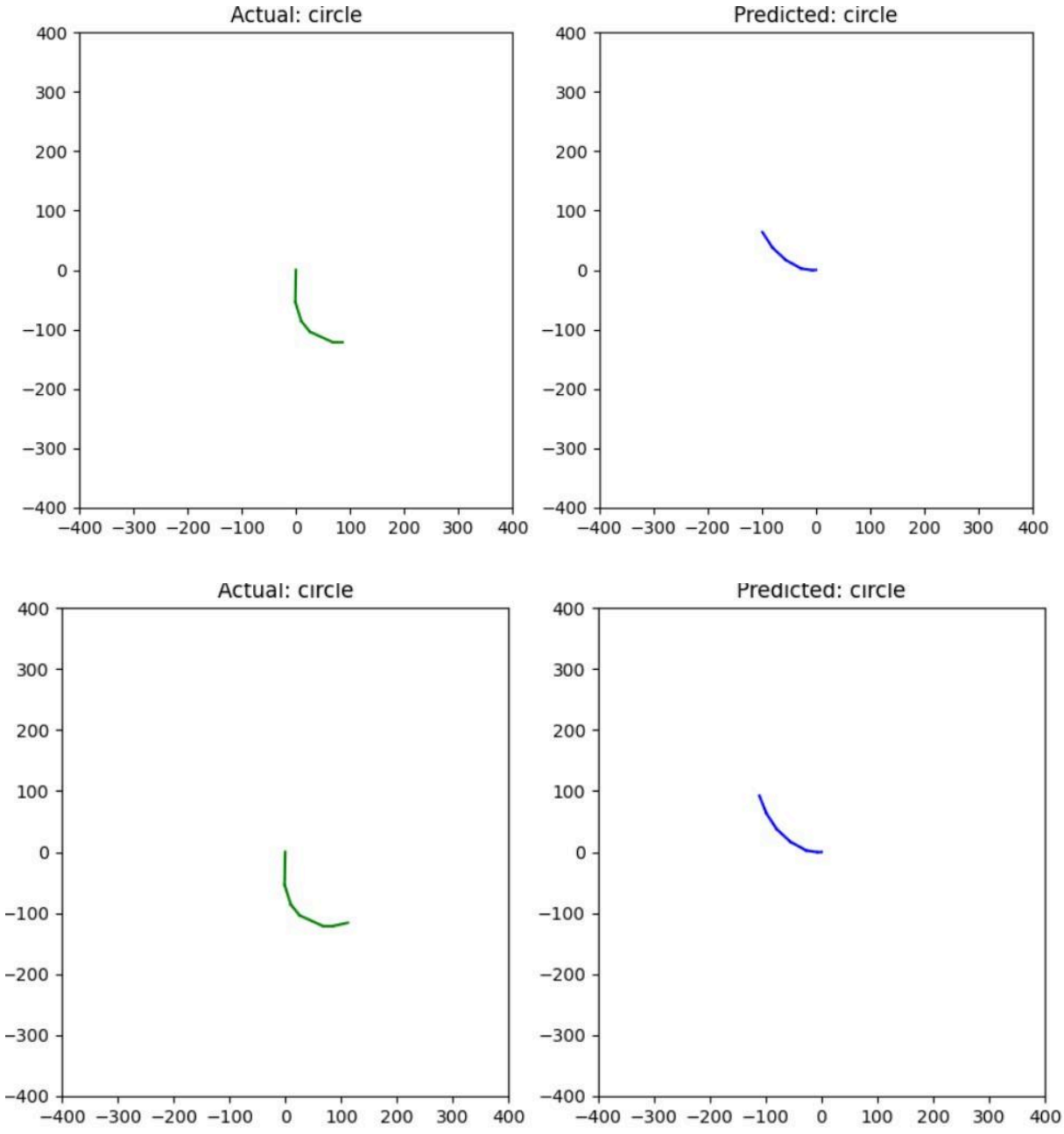

Actual: apple
Predicted: apple

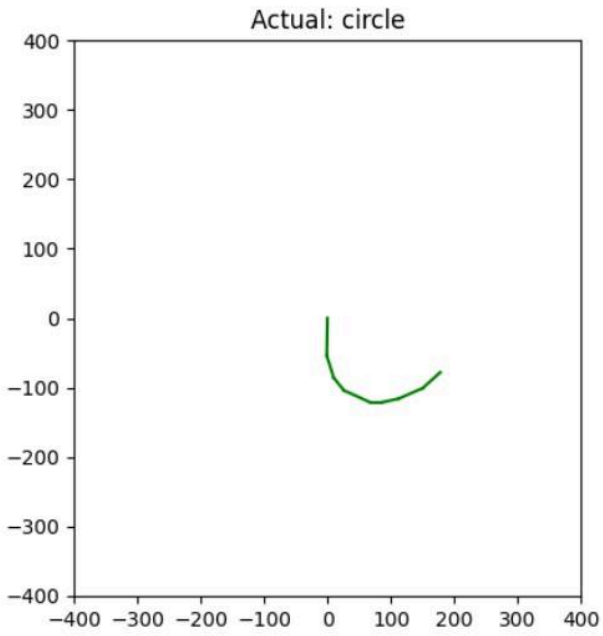

Actual: apple
Predicted: apple
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

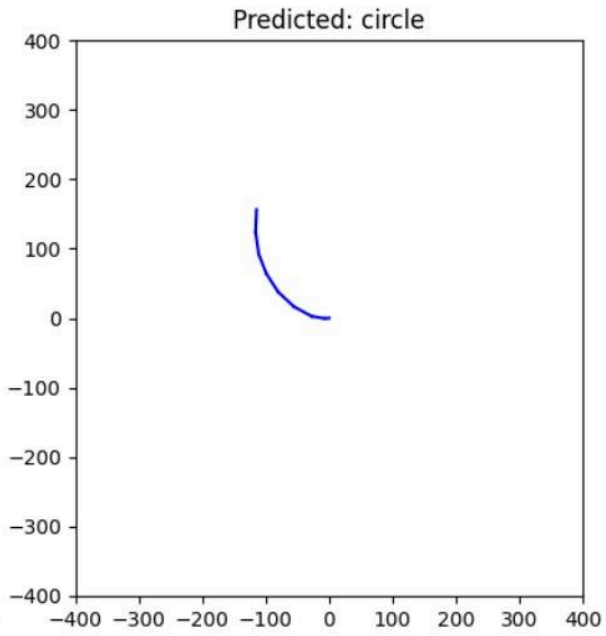


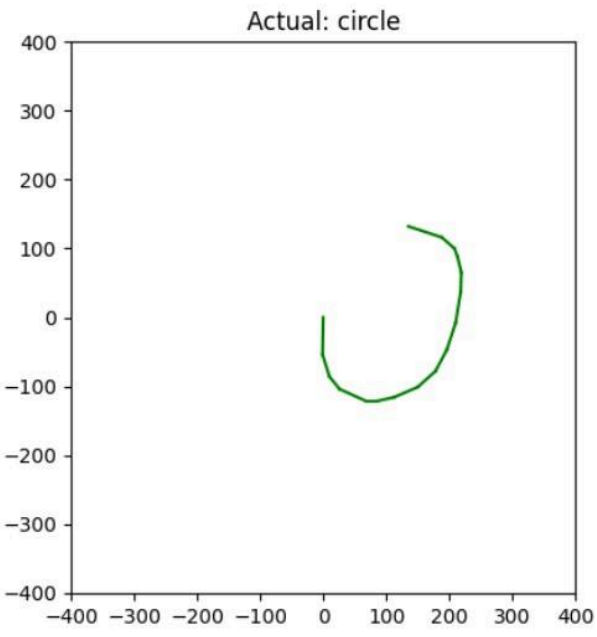
The Above image is the final image formed for the class apple

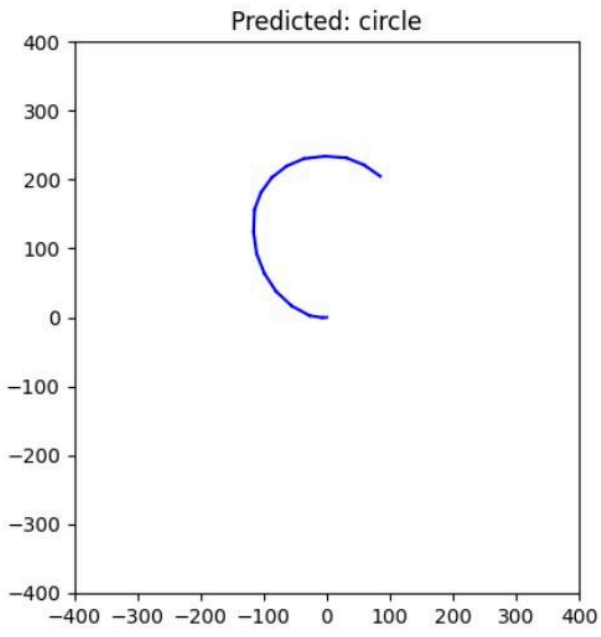
- The stroke-wise representation for circle

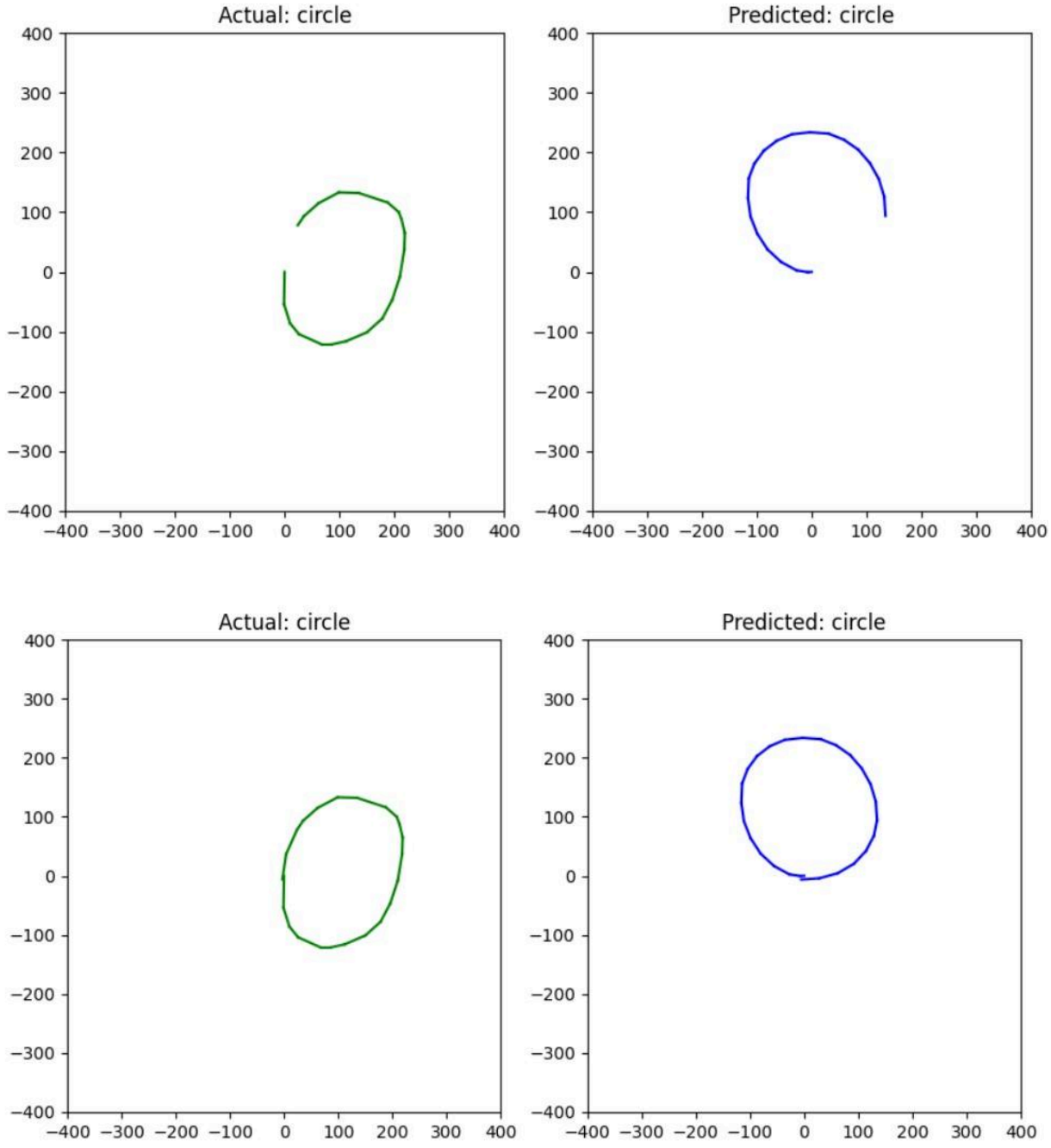



Actual: circle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: circle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

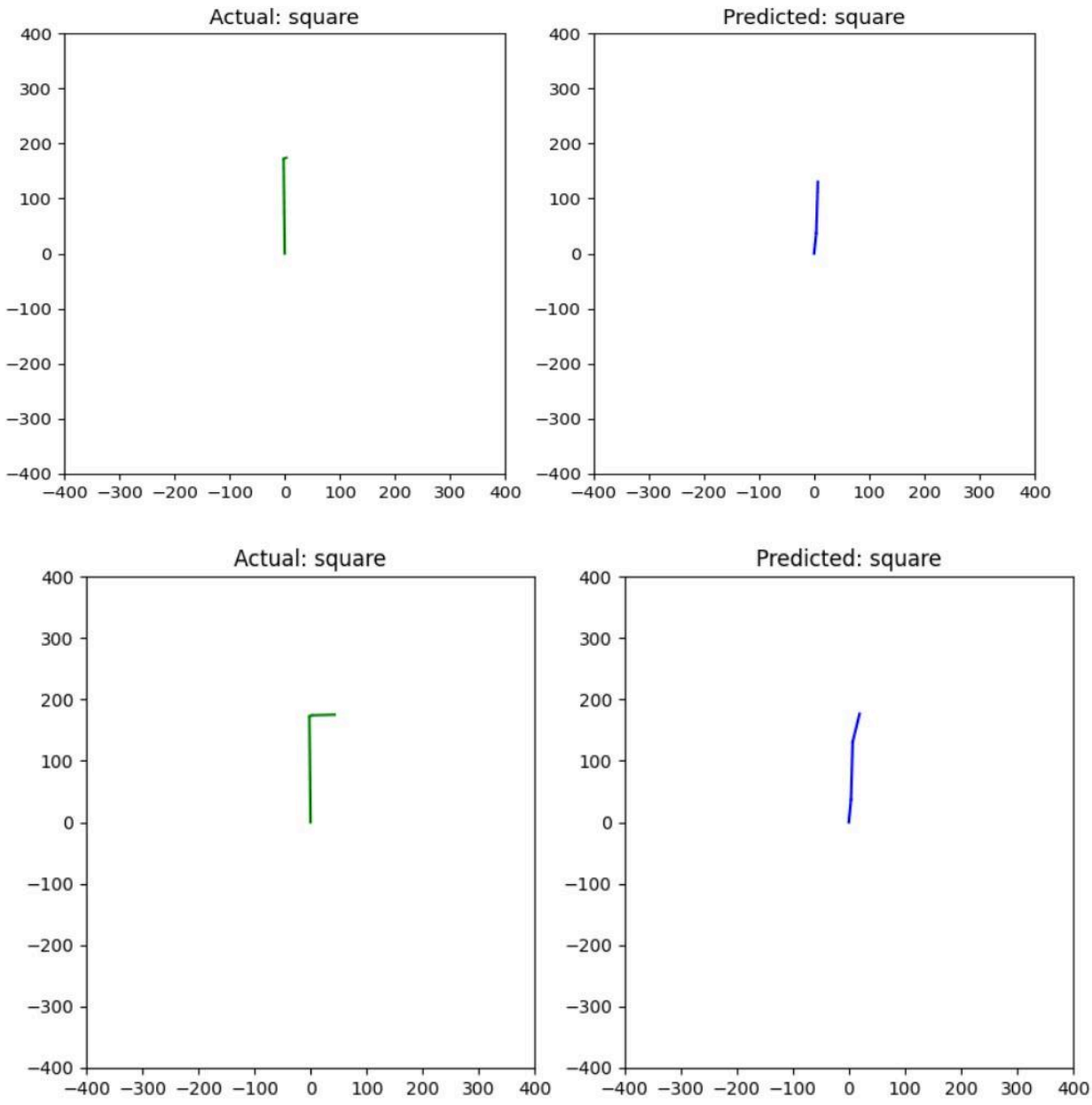

Actual: circle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

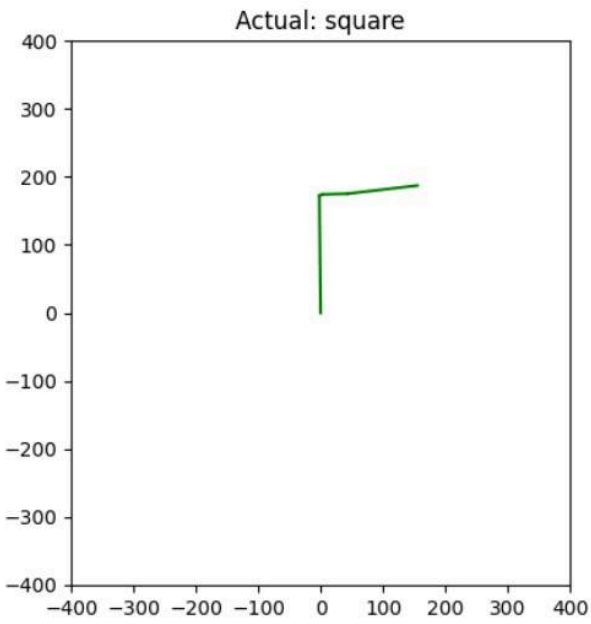

Predicted: circle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

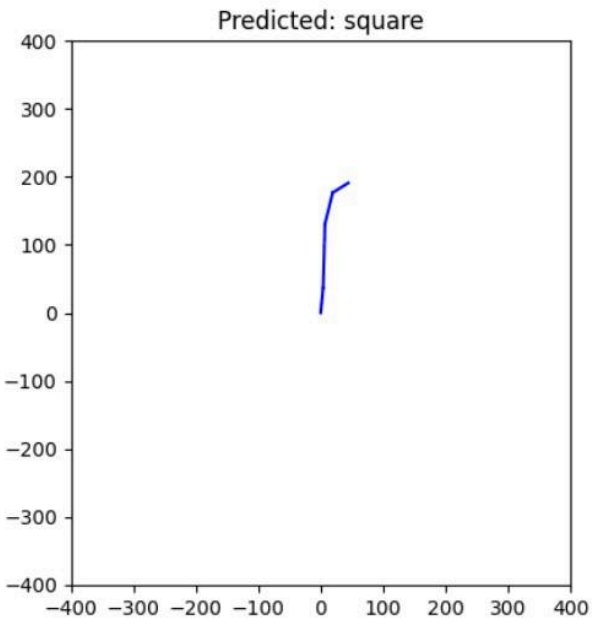


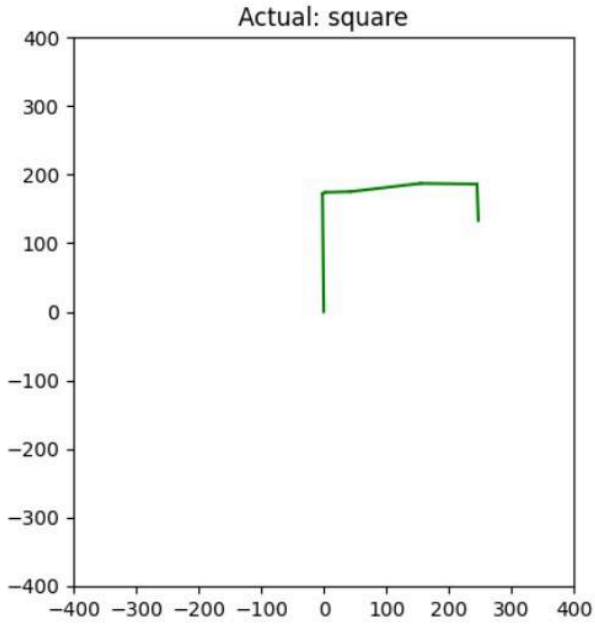
The above image is final image formed for circle

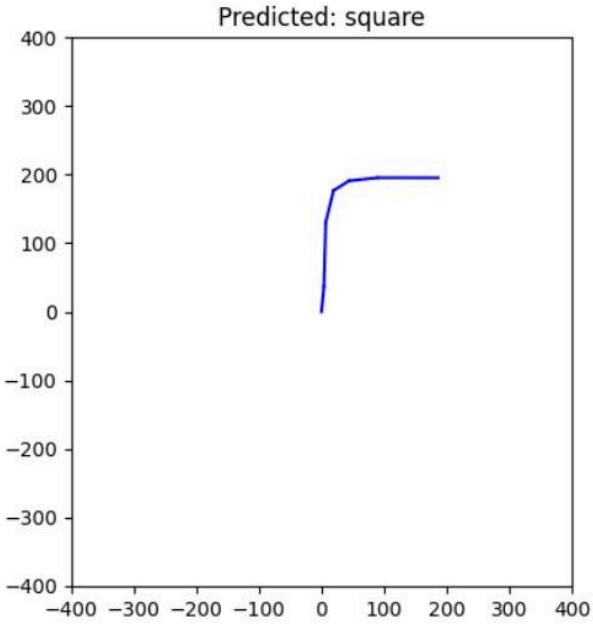
- The stroke-wise representation for square

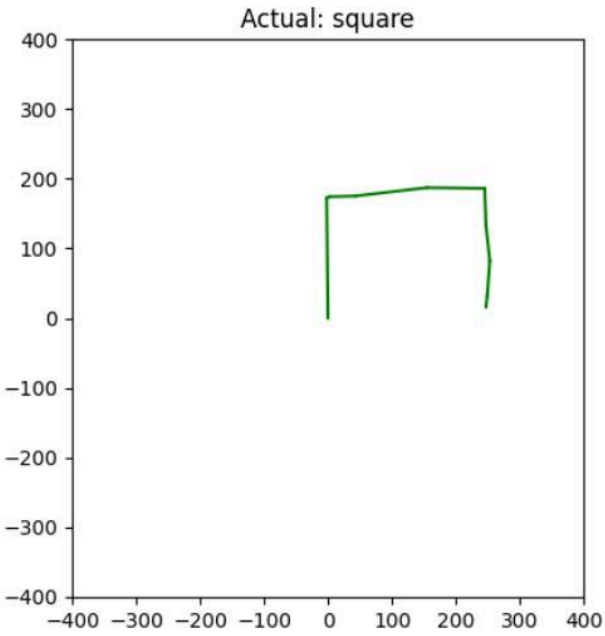


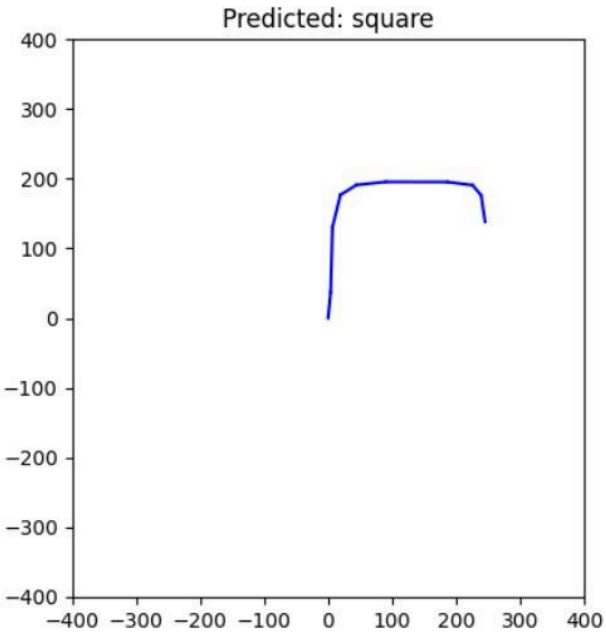

Actual: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

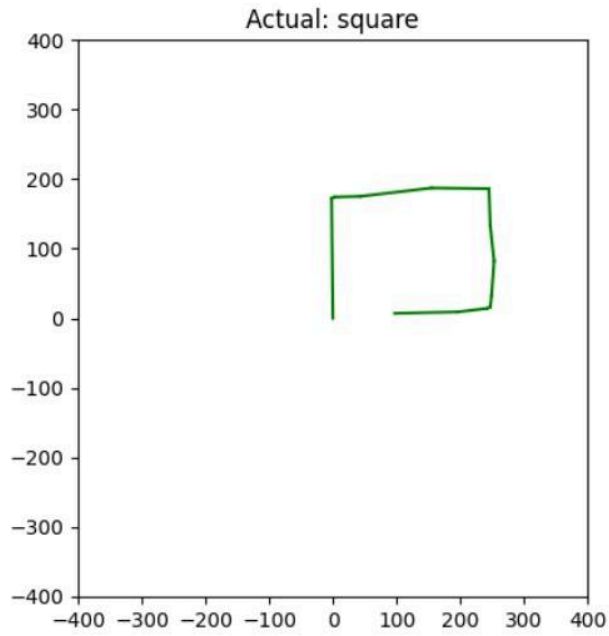

Predicted: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

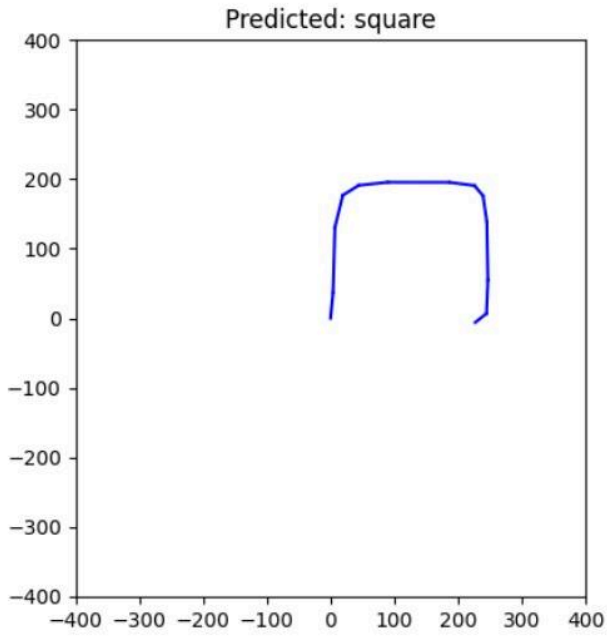

Actual: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

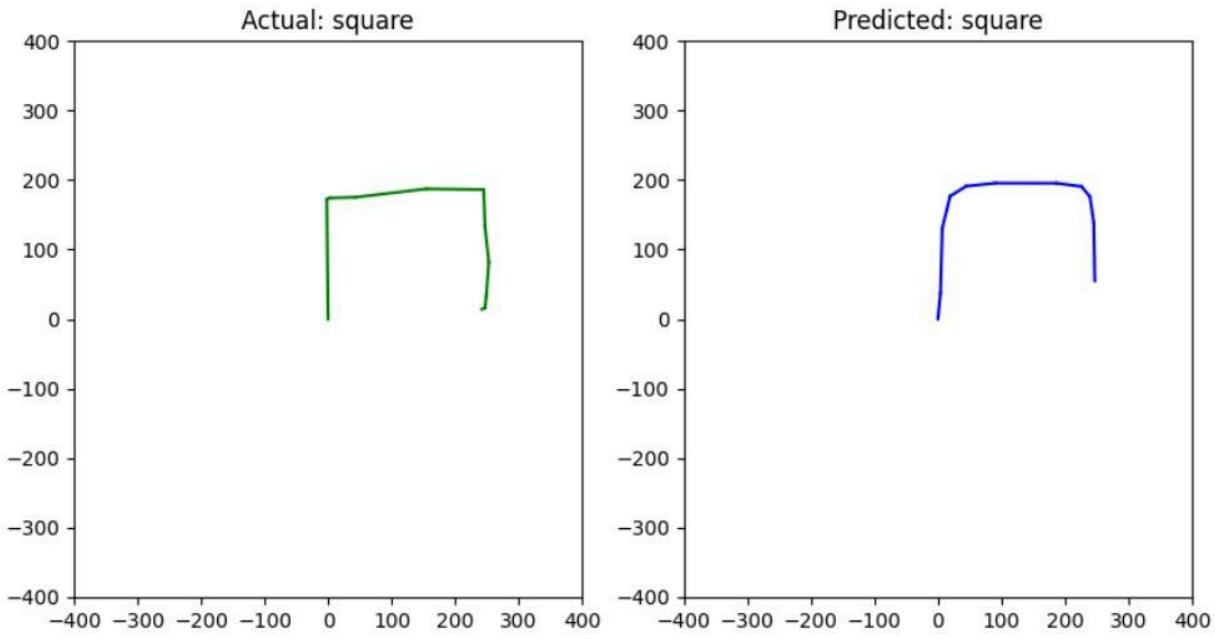

Predicted: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

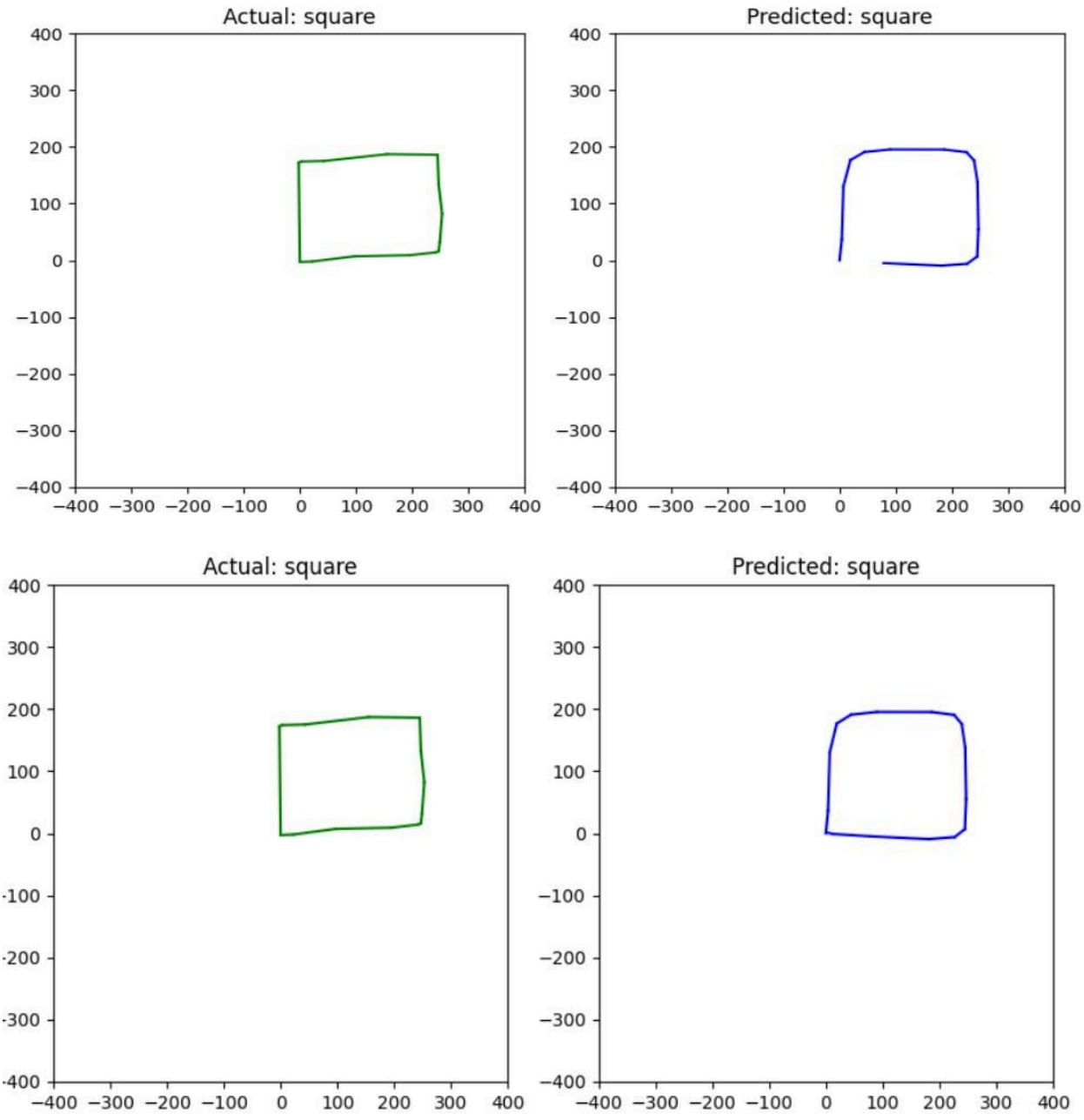

Actual: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Actual: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

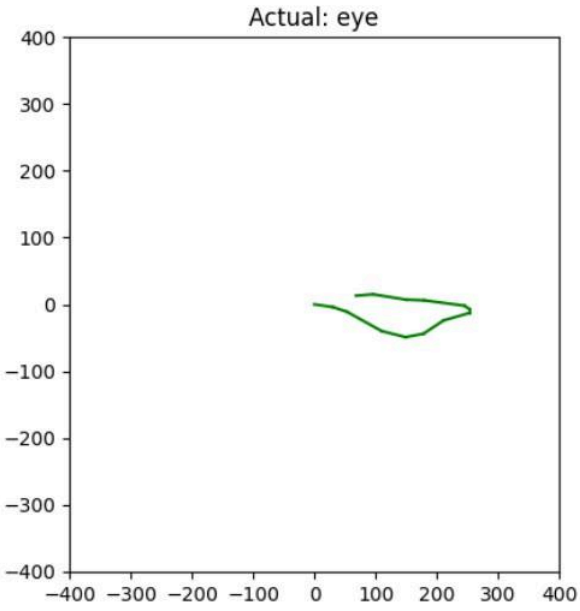

Predicted: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

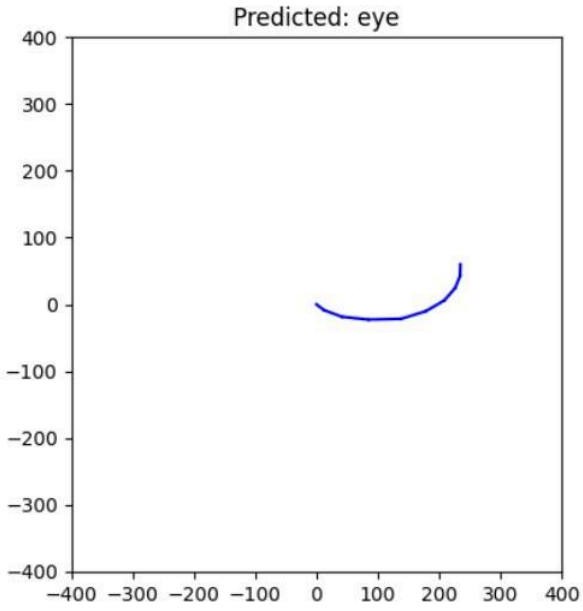

Actual: square
Predicted: square
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

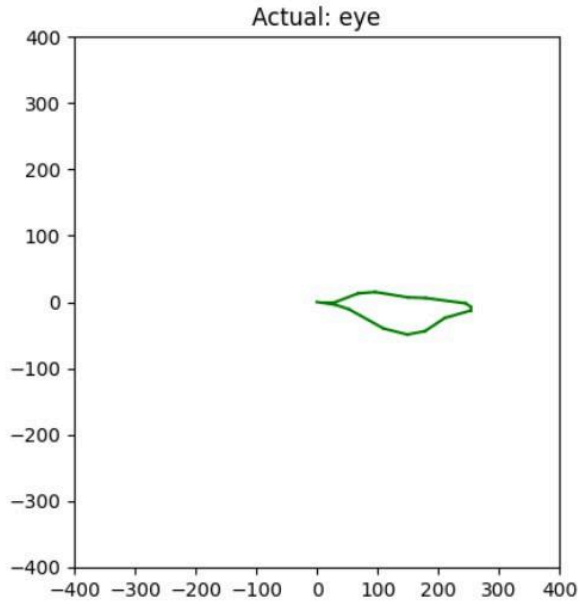


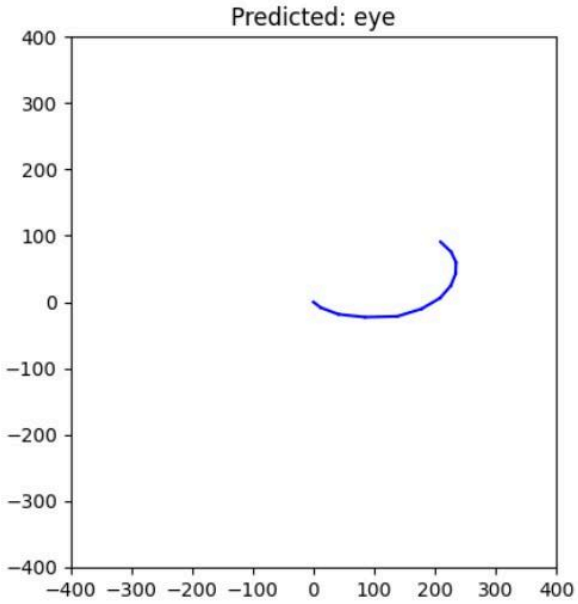
The above is the final representation for the actual sketch and predicted square

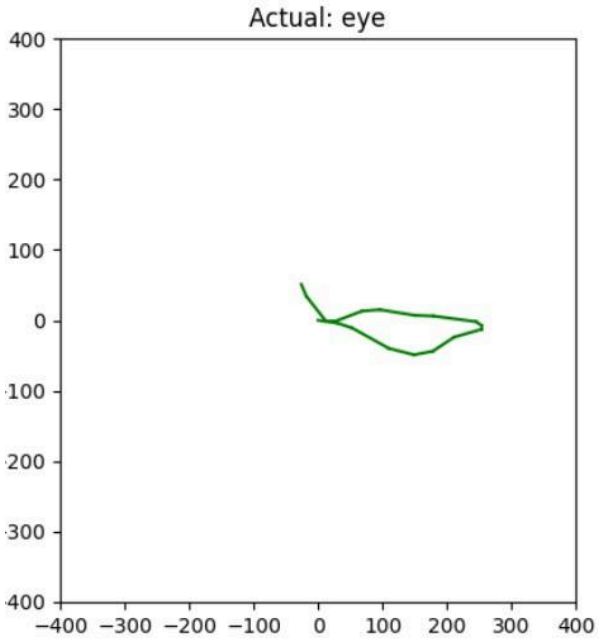
- The stroke-wise representation for eye

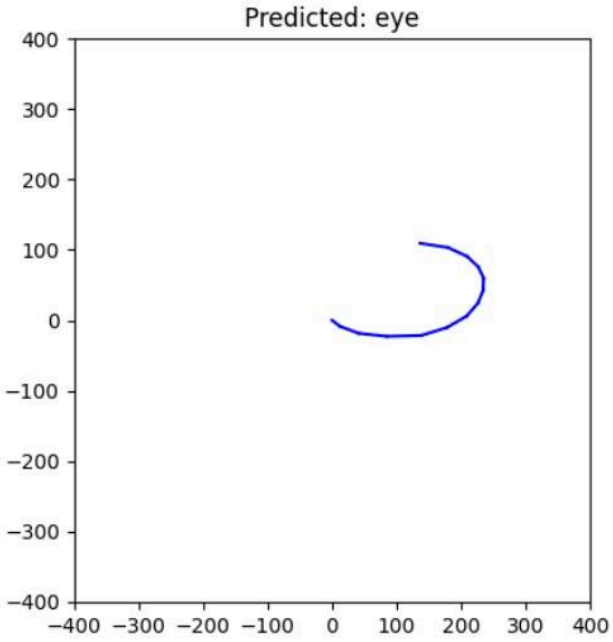


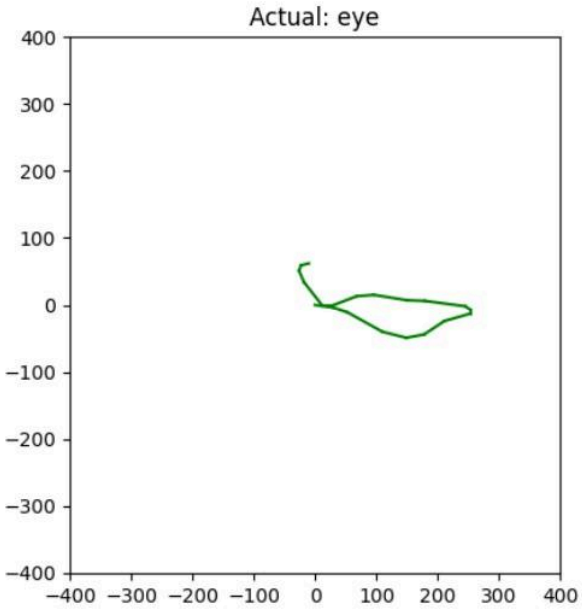


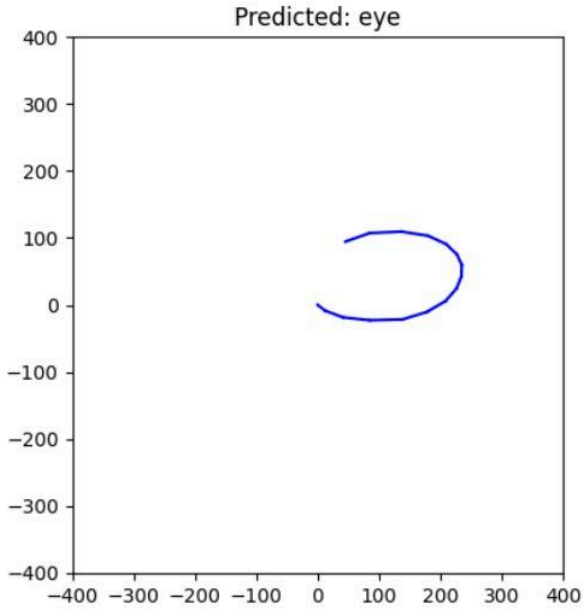


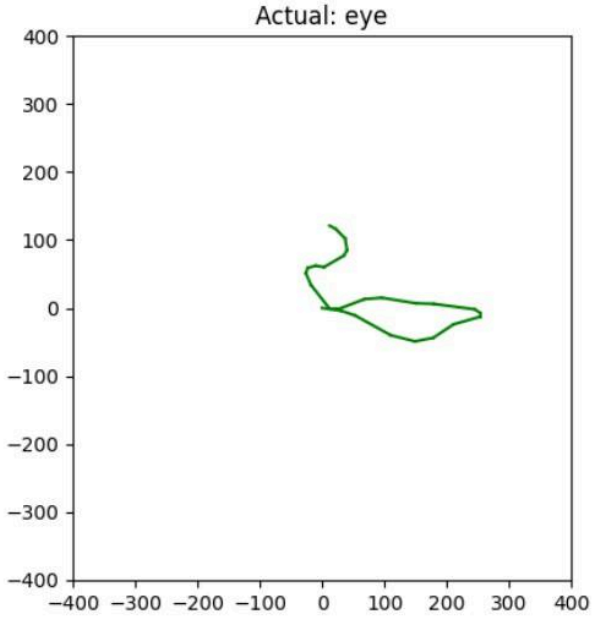


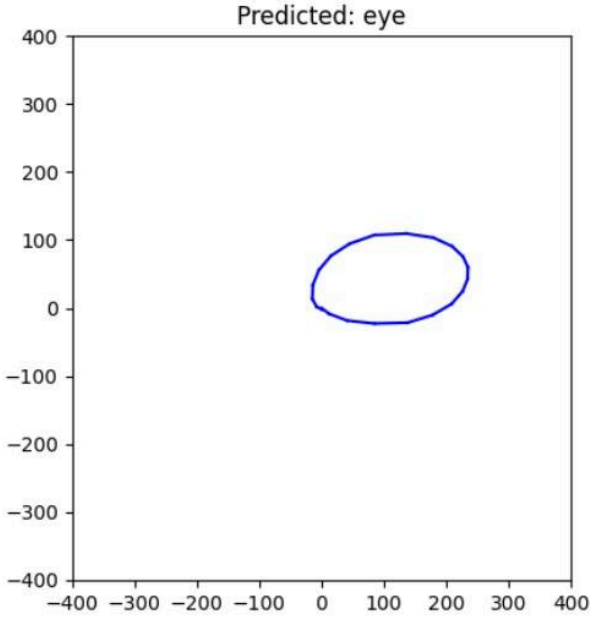

Actual: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

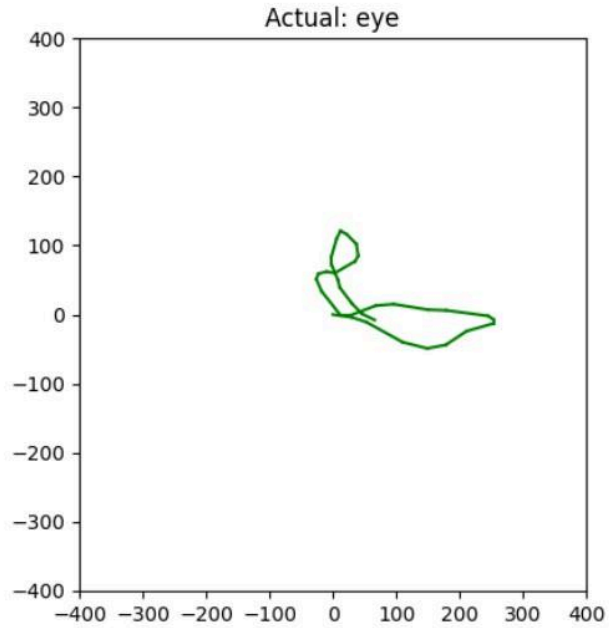

Predicted: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

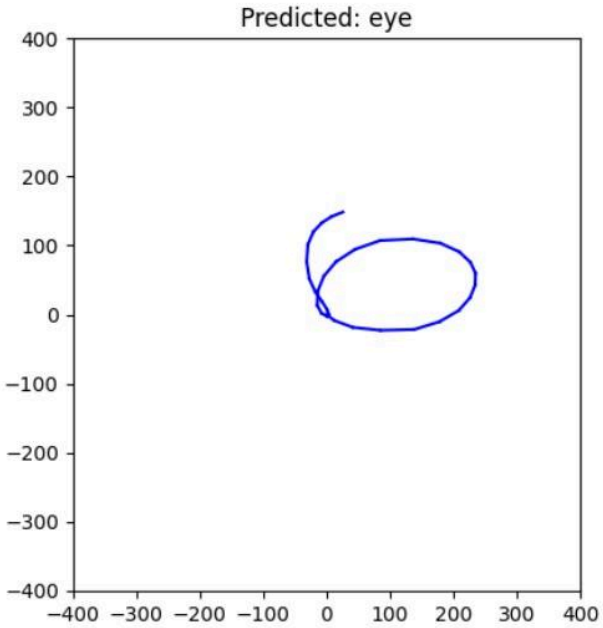

Actual: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

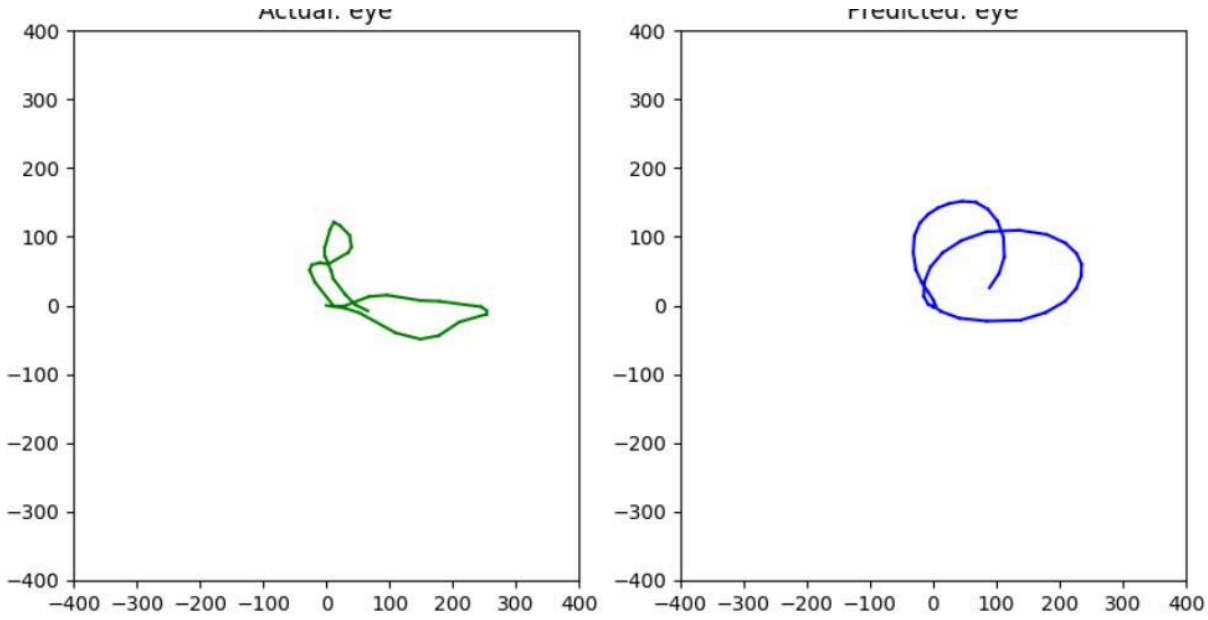

Predicted: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

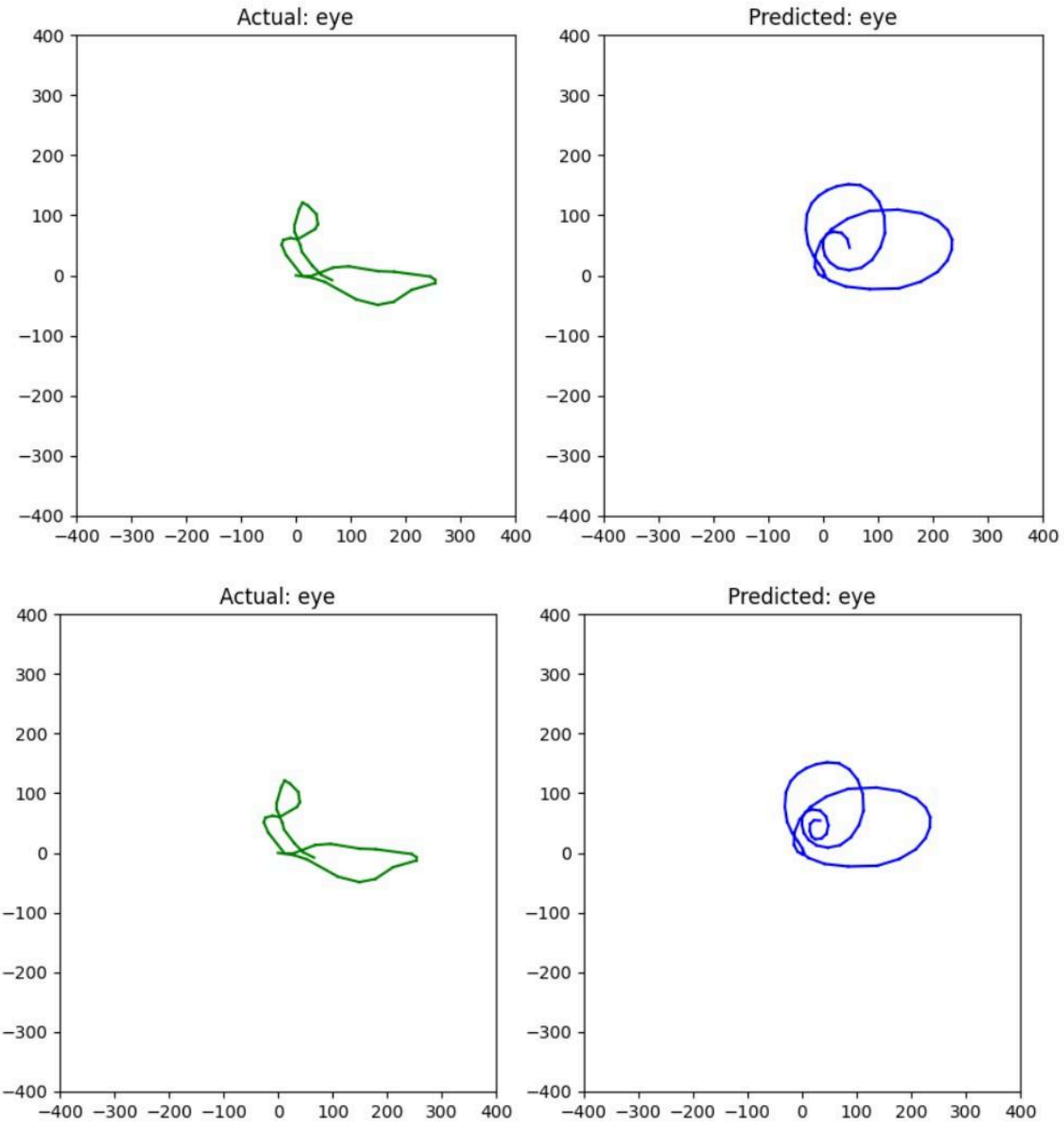

Actual: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Actual: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

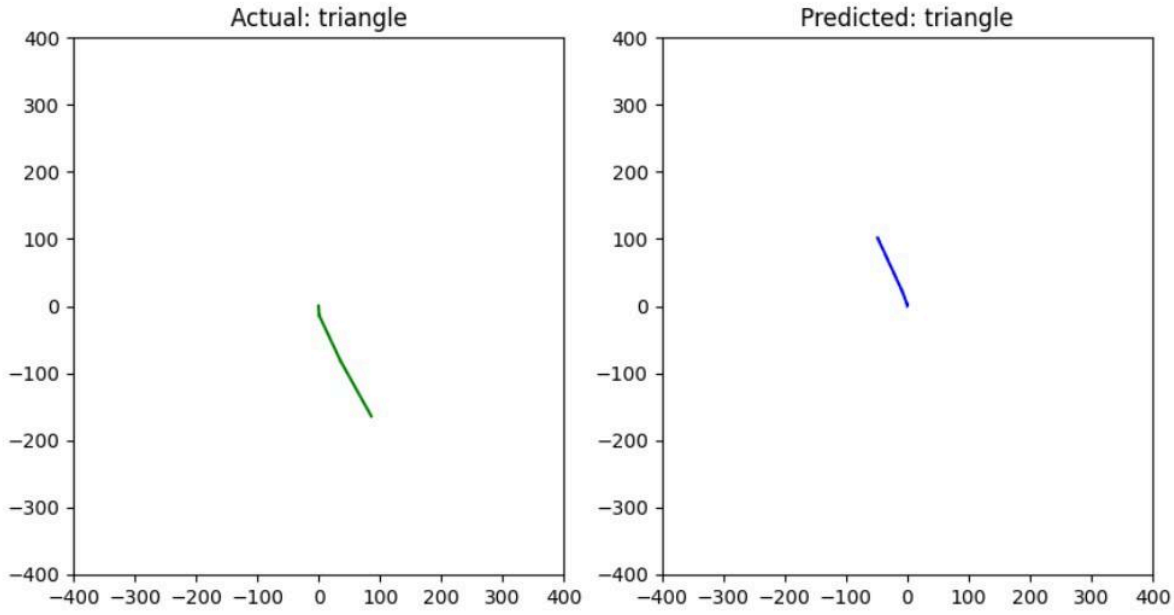

Predicted: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

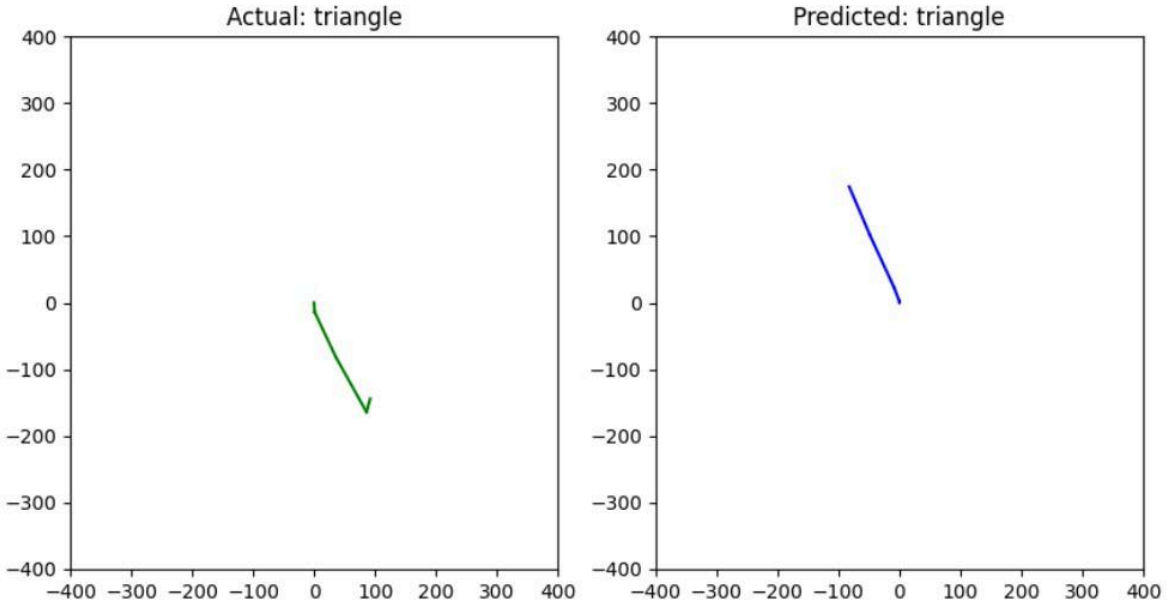

Actual: eye
Predicted: eye
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400

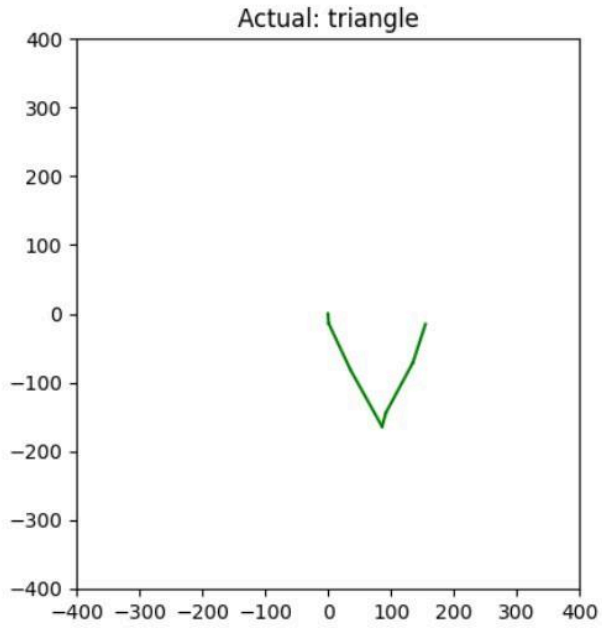


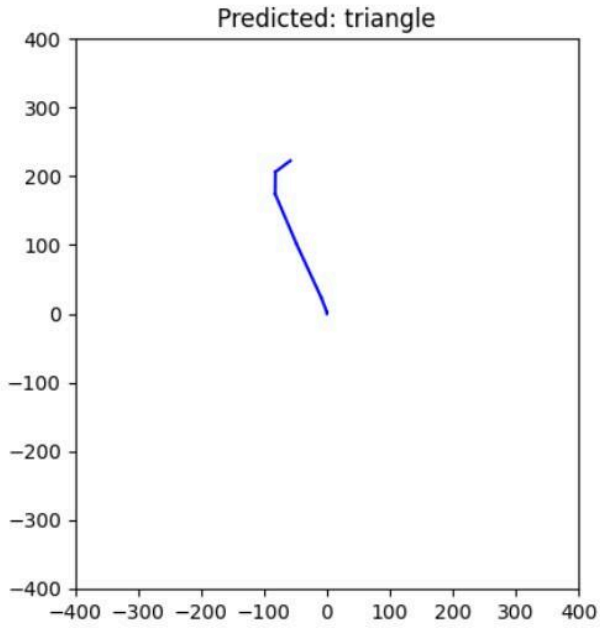
The above is the final representation for the actual sketch and predicted eye.

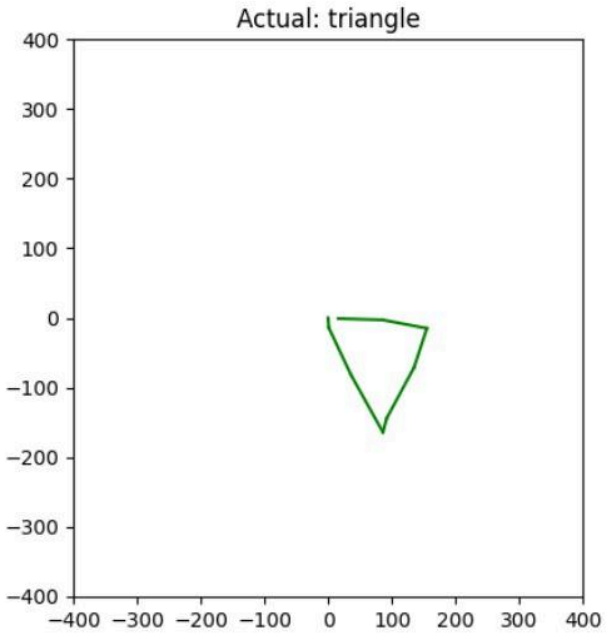
- The below is the stroke wise formation of triangle

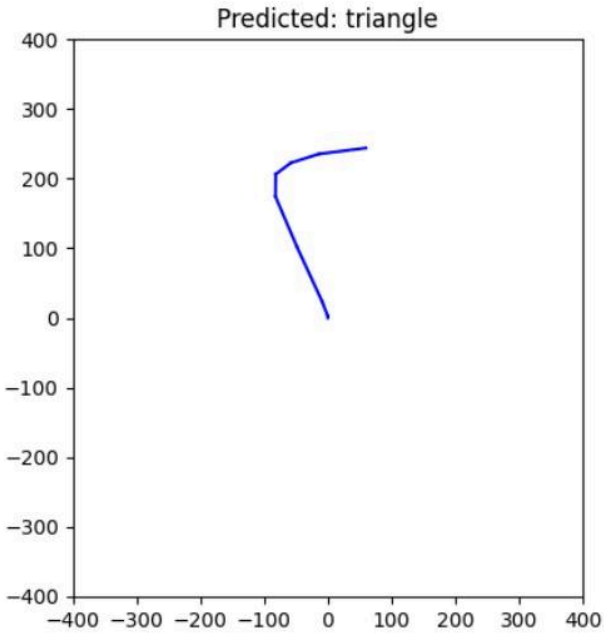


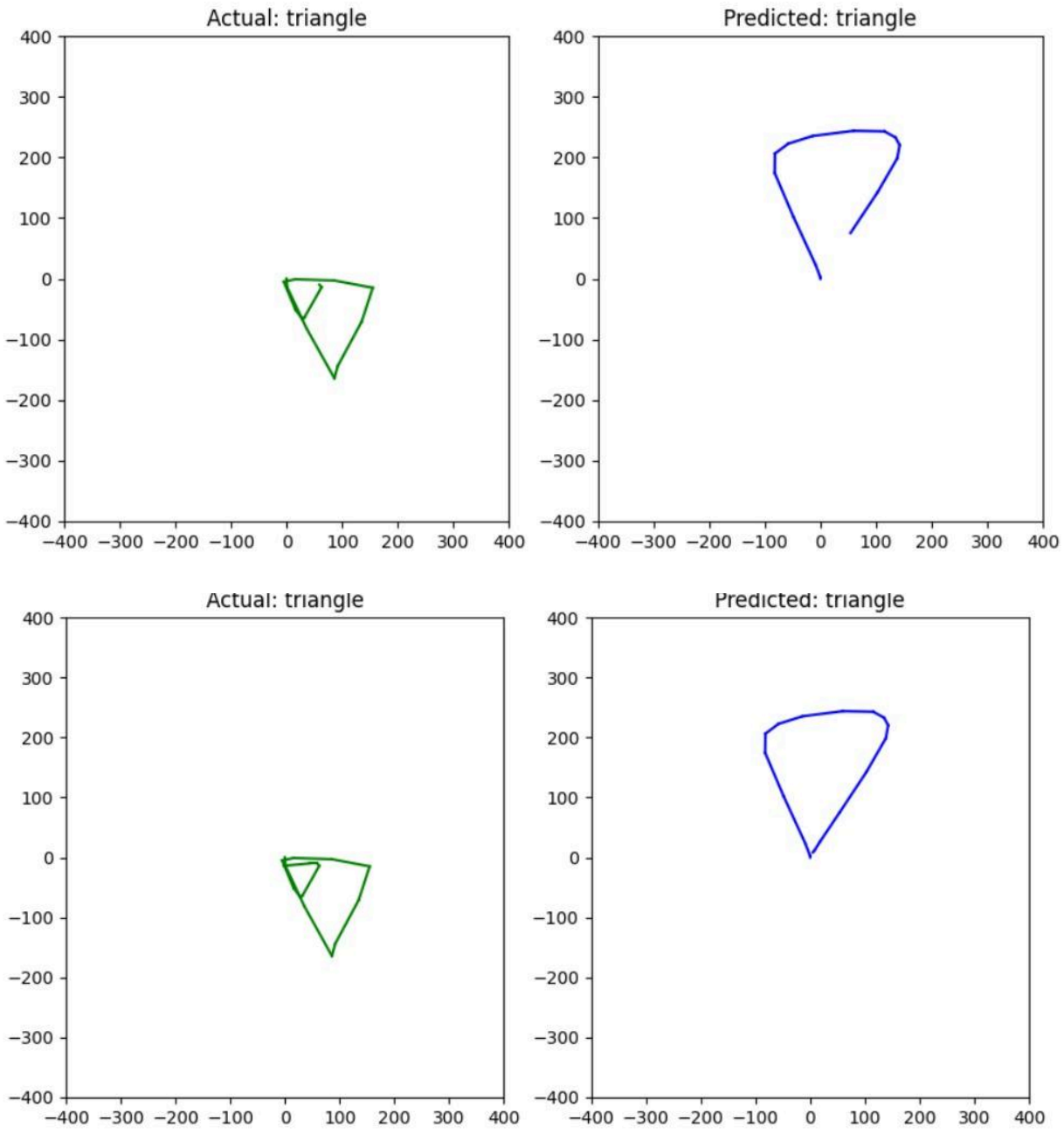



Actual: triangle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: triangle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Actual: triangle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400


Predicted: triangle
400
300
200
100
0
−100
−200
−300
−400
−400 −300 −200 −100 0 100 200 300 400



The above is the final representation for the actual sketch and predicted triangle

10. Code Link (Google Colab)

https://colab.research.google.com/drive/1dHvi_soBkwF44KyLvJieqYzT0H0dVLT8?usp=sharing

11. References:

https://github.com/magenta/magenta-demos/blob/main/jupyter-notebooks/Sketch_RNN.ipynb