Name: Ngô Lê Ngọc Quý . **Student ID:** 21040003

Course: Artificial Intelligence

Lecturer: Trần Thế Vũ

CHALLENGE 1 REPORT

Table of Contents

I. Overview

The primary objective of this challenge is to train a Convolutional Neural Network (CNN) model to recognize hand-drawn and deploy this model as a web application. The challenge consists of three main phases: data collection and preprocessing, model building and training, and deployment. The model will be trained to recognize drawings of specific objects, such as cars, eyes, flowers, and houses, and users will be able to draw doodles and have them recognized through a web-based interface.

Github repository of the challenge: [https://github.com/ngocquyxs/-AI-Challenge-1]

II. Phases

1. *Data Collection and Preprocessing*

1.1. Downloading Datasets

This code is used to create a text file called "class.txt", download data from the Internet, and save it into a "data" directory.

```
# Open “class.txt” - file likely contains a list of classes.
f = open("class.txt", "r")

# Read the content of “class.txt”
classes = f.readlines()

# Close “class.txt”
f.close()

# Preprocess class names - to ensure that the class names are formatted consistently and
can be used as valid filenames.
classes = [c.replace('\n','').replace(' ','_') for c in classes]
print(classes)

# Create "data" Directory - where the dataset files will be downloaded and stored.
!mkdir data

# ‘download’ function - responsible for fetching the data files for each class from the
Quick, Draw! dataset.
def download():
  base = ‘https://storage.googleapis.com/quickdraw_dataset/full/numpy_bitmap/’
  for c in classes:
    cls_url = c.replace('_', '%20')
    path = base+cls_url+'.npy'
    print(path)
    urllib.request.urlretrieve(path, 'data/'+c+'.npy')

download()
```

1.2. Loading and Preprocessing

This code defines a function to load and preprocess image data stored in numpy files for machine-learning tasks. It handles tasks such as limiting the number of items per class, randomizing the dataset and splitting it into training and testing sets.

```
# "load_data" function
def load_data(root, vfold_ratio=0.2, max_items_per_class= 4000 ):

   # Get list of 'npy' files - The data likely involves image data stored in numpy format
(‘npy’ files). Using ‘glob’ to retrieve a list of all such files in the specified directory.
   all_files = glob.glob(os.path.join(root, '*.npy'))

   # Initializes variables - x will be a 2D array representing the input data (images), and y
will be a 1D array containing the corresponding labels.
   x = np.empty([0, 784])
   y = np.empty([0])

   class_names = []

   # Load each data file
   for idx, file in enumerate(all_files):
      data = np.load(file)

      # Limits the number of items per class - to control the dataset size and balance the
number of examples per class.
      data = data[0: max_items_per_class, :]

      # Create labels - essential for supervised learning, where the model learns to
associate input data with specific labels.
      labels = np.full(data.shape[0], idx)

      # Concatenate data and labels - to create a comprehensive dataset (x and y) that can
be easily fed into a machine-learning model.
      x = np.concatenate((x, data), axis=0)
      y = np.append(y, labels)

      # Extract class name - useful for later analysis and interpretation of the results.
      class_name, ext = os.path.splitext(os.path.basename(file))
      class_names.append(class_name)

   data = None
   labels = None

   # Randomize the dataset - to introduce variability and prevent the model from learning
patterns based on the order of the data.
   permutation = np.random.permutation(y.shape[0])
   x = x[permutation, :]
   y = y[permutation]
```

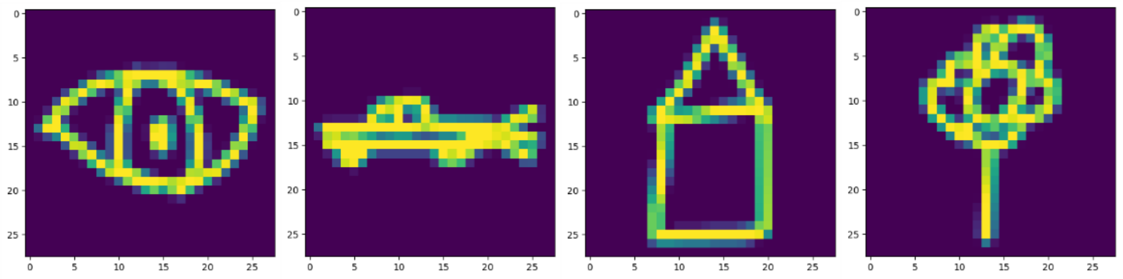
```
    # Separating into Training and Testing sets - crucial for evaluating the model's
performance on unseen data.
    vfold_size = int(x.shape[0]/100*(vfold_ratio*100))

    x_test = x[0:vfold_size, :]
    y_test = y[0:vfold_size]

    x_train = x[vfold_size:x.shape[0], :]
    y_train = y[vfold_size:y.shape[0]]

    # Returns the result
    return x_train, y_train, x_test, y_test, class_names
```

This code is a key part of the challenge as it loads and prepares the dataset for training a machine-learning model.

```
# Use 'load_data' function - to prepare a dataset for training step
x_train, y_train, x_test, y_test, class_names = load_data('data')

# Configuring the output layer of our machine-learning model
num_classes = len(class_names)

# Sets image size - to configure the first layer of our neural network to accept inputs of the
correct dimension.
image_size = 28
```

This code allows us to visually inspect a random sample, verify the correctness of our preprocessing steps, and gain an intuitive understanding of the input data before feeding it into a machine-learning model.

```
# Random Sample Selection - to ensure that each time running this code, a different
random sample is selected.
idx = randint(0, len(x_train))

# Display the Image - to display the image corresponding to the randomly selected index.
plt.imshow(x_train[idx].reshape(28,28))

# Print the class name - to understand the label of the displayed image
print(class_names[int(y_train[idx].item())])
```



This code is essential for preparing the image data in a suitable format for training a convolutional neural network (CNN)

# Reshape images *- to comply with the input shape expected by many convolutional neural network (CNN) models.*
```
x_train = x_train.reshape(x_train.shape[0], image_size, image_size, 1).astype('float32')
x_test = x_test.reshape(x_test.shape[0], image_size, image_size, 1).astype('float32')
```

# Normalise pixel values *- to help the model converge faster during training and avoid issues related to varying scales.*
```
x_train /= 255
x_test /= 255
```

# Convert class vectors to class matrices *- to make it easier to train and evaluate neural network models for classification tasks.*
```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

2. Model Building and Training

2.1. Defining Model Architecture

This code defines a convolutional neural network (CNN) architecture for image classification.

# Sequential model
```
model = models.Sequential()
```

# Convolutional layers *- to maintain spatial dimensions, ReLU activation function, and input shape compatible with the shape of the input data (`x_train`).*

# MaxPooling layers *- to reduce the spatial dimensions of the representation, providing computational efficiency and translation invariance.*

```
model.add(layers.Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:],
activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# Flatten layer - necessary before transitioning to fully connected layers.
model.add(layers.Flatten())

# Fully connected (Dense) layers - use to reduce overfitting by randomly setting a fraction
of input units to 0 during training.
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.5))

# Output Layer - Softmax converts the model's raw output into probabilities, making it
suitable for multi-class classification.
model.add(layers.Dense(4, activation='softmax'))
```

2.2. Compiling the Model

This code is a critical step in preparing the neural network for training. It sets up the model with the necessary configurations for learning from the data, including the choice of loss function, optimizer, and evaluation metrics.

```
# Compile the model
model.compile(
    loss='categorical_crossentropy', # to measure the difference between the predicted
probability distribution and the true probability distribution of the classes.
    optimizer='adam', # to adapt the learning rate during training to improve convergence
speed and performance
    metrics=['accuracy'] # to measure the proportion of correctly classified instances
)
```

2.3. Data Augmentation

This code is essential for enhancing the robustness and generalization capabilities of a deep learning model, particularly in image classification tasks.

```
# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(x_train)
```

2.4. Training the Model

This code integrates early stopping into the model training process. It uses data augmentation for improved generalization, monitors the validation loss during training, and stops training early if no improvement is observed, helping create a more robust and well-generalizing model.

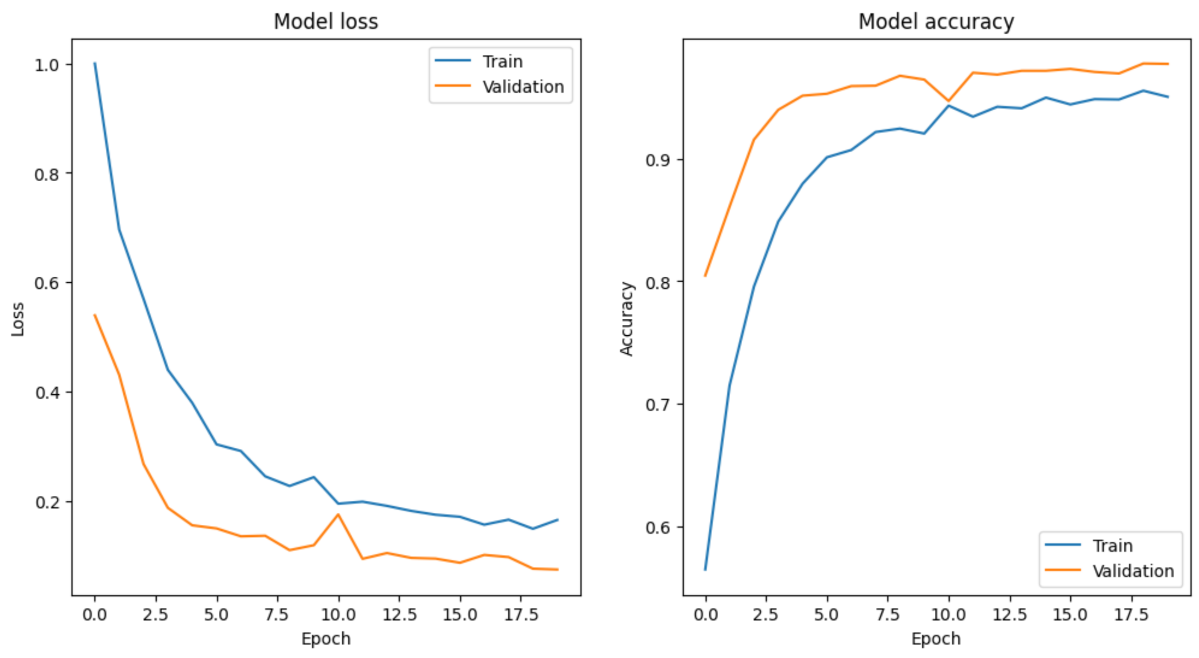
```
# Early stopping - to monitor the validation loss during training and stop the training process if the validation loss does not improve for a specified number of consecutive epochs.
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train the model
history = model.fit(
    datagen.flow(x_train, y_train, batch_size=64),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // 256,
    epochs=20,
    callbacks=[early_stopping]
)

# Evaluate the Model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuarcy: {:0.2f}%'.format(score[1] * 100))
```

```
Test accuarcy: 97.75%
```

2.5. Visualizing the Model

This code is used to create a visual representation of the model's training and validation performance, enabling a quick assessment of how well the model is learning from the data and generalizing to unseen data.

```
# Plot training & validation loss values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.show()
```



2.6. Making the Inference

This code segment facilitates the inference process, enabling us to visually inspect test images, make predictions using the trained model, and understand the model's top predicted classes for a given image.

```
# Inference
idx = randint(0, len(x_test))
img = x_test[idx]
plt.imshow(img.squeeze())

pred = model.predict(np.expand_dims(img, axis=0))[0]  # to use the trained model to
predict the class probabilities for the selected test image.
```

```
ind = (-pred).argsort()[:5]
latex = [class_names[x] for x in ind]
print(latex)
```

2.7. Saving the Model

This code helps preserve both the class names associated with the model and the trained model itself for future use or sharing.

```
# Store classes
with open('class_names.txt', 'w') as file_handler:
    for item in class_names:
        file_handler.write("{}\n".format(item))

# Save the model
model.save('keras.h5')
```

3. Deployment

3.1. Creating Template

This code forms the basis for a simple drawing application with additional features like clearing the canvas and integrating image recognition.

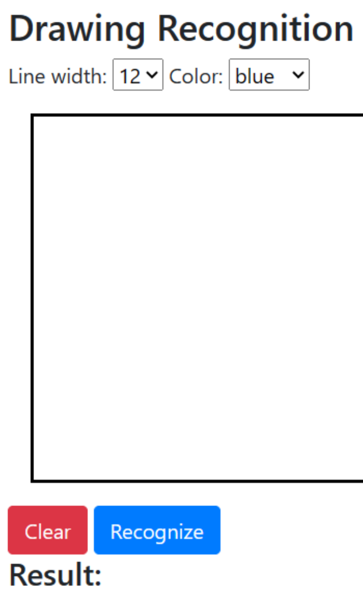
```
# 'init' function
function init() {
    // Get the canvas element and its 2D rendering context
    canvas = document.getElementById('myCanvas');
    ctx = canvas.getContext('2d');


    // Set the initial canvas background color to white
    ctx.fillStyle = 'white';
    ctx.fillRect(0, 0, canvas.width, canvas.height);


    // Event listeners for mouse actions on the canvas
    $('#myCanvas').mousedown(function(e){
        // Set the mousePressed flag to true when the mouse button is pressed
        mousePressed = true;


        // Call the draw function with the current mouse position and a flag indicating the start
of the drawing
        draw(
            e.pageX - $(this).offset().left,
            e.pageY - $(this).offset().top,
            false
        );
    });


    $('#myCanvas').mousemove(function(e){
        // Check if the mouse is pressed (drawing in progress)
        if (mousePressed) {
            // Call the draw function with the current mouse position and a flag indicating
ongoing drawing
            draw(
                e.pageX - $(this).offset().left,
                e.pageY - $(this).offset().top,
                true
            );
        }
    });


    $('#myCanvas').mouseup(function(e){
        // Set the mousePressed flag to false when the mouse button is released
        mousePressed = false;
    });


    $('#myCanvas').mouseleave(function(e){
        // Set the mousePressed flag to false when the mouse leaves the canvas
        mousePressed = false;
    });
}
```

***# ‘draw function** - allows users to draw on a canvas using the mouse, providing an interactive and visual experience.*

```
function draw(x, y, isDown) {
    if (isDown) {
        // Begin a new path for drawing
        ctx.beginPath();

        // Set drawing parameters
        ctx.strokeStyle = $('#selColor').val();  // Get selected color
        ctx.lineWidth = $('#selWidth').val();    // Get selected line width
        ctx.lineJoin = 'round';

        // Move to the last recorded mouse position
        ctx.moveTo(lastX, lastY);
        // Draw a line to the current mouse position
        ctx.lineTo(x, y);

        // Close the path and stroke (draw) the line
        ctx.closePath();
        ctx.stroke();
    }

    // Update the last recorded mouse position
    lastX = x;
    lastY = y;
}
```

```
# 'clearCanvas' function – to provide a way to reset the canvas, removing any drawn
content.
function clearCanvas() {
    // Reset the transformation matrix to the identity matrix
    ctx.setTransform(1, 0, 0, 1, 0, 0);

    // Set the canvas background color to white
    ctx.fillStyle = 'white';

    // Fill a white rectangle to clear the entire canvas
    ctx.fillRect(0, 0, canvas.width, canvas.height);
}
```

```
# 'postImage' function – to send the drawn image data to a server for recognition.
function postImage() {
    // Get the base64-encoded image data from the canvas
    var img = document.getElementById("myCanvas").toDataURL("image/png");

    // Remove the data URI prefix to obtain the raw base64-encoded image data
    img = img.replace(/^data:image\/(png|jpg);base64,/, "");

    // Make an AJAX (asynchronous) POST request to the "/recognize" endpoint
    $.ajax({
        type: "POST",
        url: "/recognize",
        data: JSON.stringify({ image: img }),
        contentType: 'application/json; charset=UTF-8',
        dataType: 'json',
        success: function(msg, status, jqXHR) {
            // On successful response from the server
            var data = JSON.parse(jqXHR.responseText);
            var prediction = data.prediction;

            // Log the prediction to the console
            console.log(prediction);

            // Update the result element on the page with the prediction
            document.getElementById("result").innerHTML = prediction;
        }
    });
}
```



3.2. Using Flask Server

This code sets up a web-based image recognition application that utilizes a pre-trained deep learning model to predict the class of drawn images. The Flask framework is used to handle the web application, and TensorFlow/Keras is used for the deep learning model integration.

```
# Load trained model
model = tf.keras.models.load_model('keras.h5')
model.make_predict_function() // to ensure that the model is thread-safe when used in a multi-threaded environment.


# Define routes
@app.route('/')
def index():
    return render_template('index.html')
```

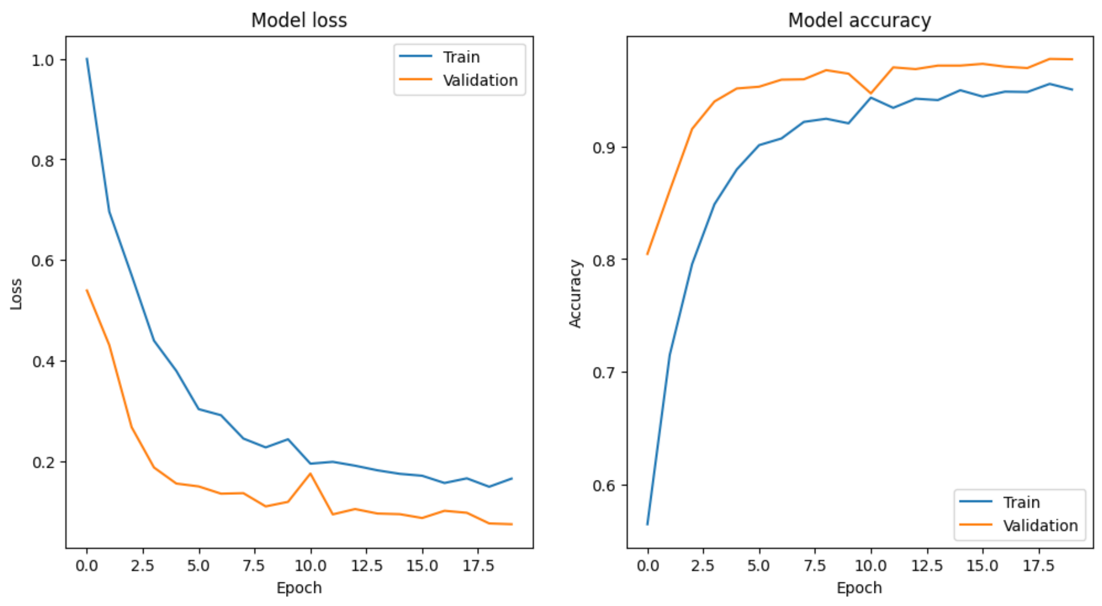
```
# Load class labels
with open('class.txt', 'r') as file:
    class_labels = file.read().splitlines()


# Print class labels
print(class_labels) #['flower', 'house', 'eye', 'car']


# Recognize route
@app.route('/recognize', methods=['POST'])
def recognize():
    if request.method == 'POST':
        # Receive and process the image for recognition
        data = request.get_json()
        imageBase64 = data['image']
        imgBytes = base64.b64decode(imageBase64)


        with open("temp.jpg", "wb") as temp:
            temp.write(imgBytes)


        image = cv2.imread('temp.jpg')
        image = cv2.resize(image, (28, 28), interpolation=cv2.INTER_AREA)
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGRA2GRAY)


        image_prediction = np.reshape(image_gray, (28, 28, 1))
        image_prediction = (255 - image_prediction.astype('float')) / 255


        # Make a prediction using the loaded model
        prediction = np.argmax(model.predict(np.array([image_prediction])), axis=-1)


        # Return the prediction in JSON format
        return jsonify({
            "prediction": str(class_labels[prediction[0]]),
            "status": True
        })


# Run the application
if __name__ == "__main__":
    app.run(debug=True)
```

III. Conclusion

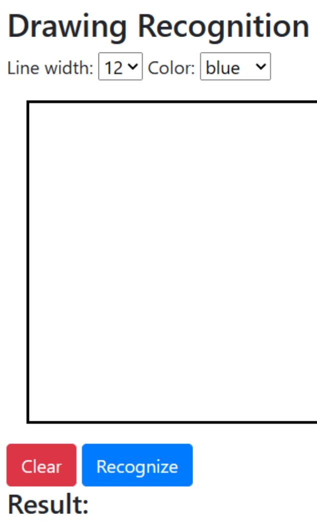
This challenge has successfully implemented a comprehensive hand-drawing recognition system, combining the training of a Convolutional Neural Network (CNN) model, evaluating its performance on test data, visualizing key training metrics, and integrating it into a Flask-based web application.

The result of this challenge

- Test accuracy: 97.75%
- Line chart after training model



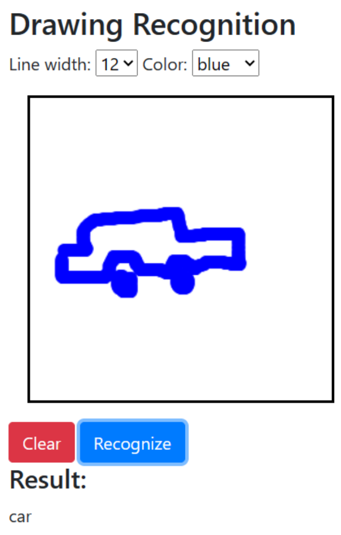
- Flask interface


Drawing Recognition

Line width: 12 Color: blue

Result:

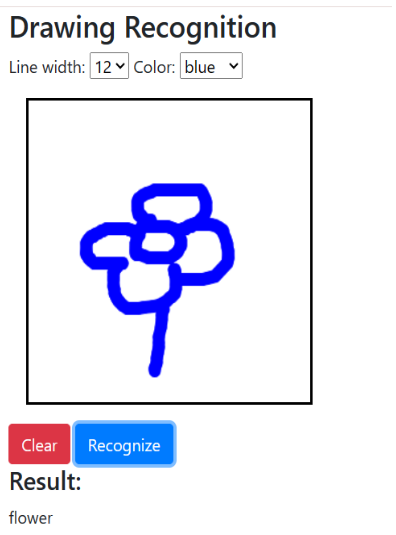
- Result when testing on interface


Drawing Recognition

Line width: 12 Color: blue

Result:

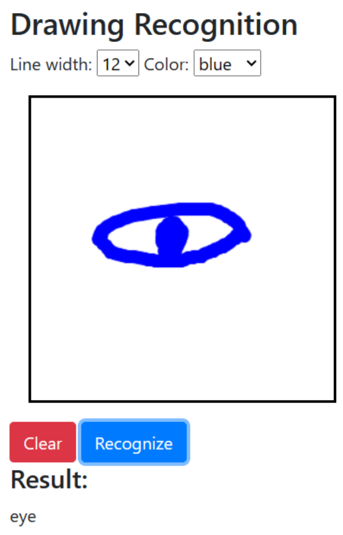
car


Drawing Recognition

Line width: 12 Color: blue

Result:

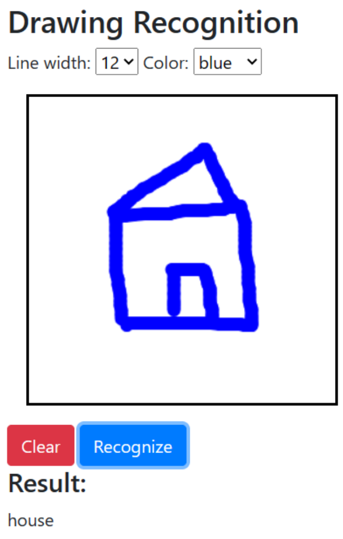
flower


Drawing Recognition

Line width: 12 Color: blue

Result:

eye


Drawing Recognition

Line width: 12 Color: blue

Result:

house

IV. Reference

The above challenge has reference from

- Code and notebooks for model:

https://github.com/zaidalyafeai/Notebooks/blob/master/Sketcher.ipynb.

- Code for Flask Server:

[https://www.youtube.com/watch?v=n0k26uJR09U]