DA5402: Machine Learning Operations Laboratory

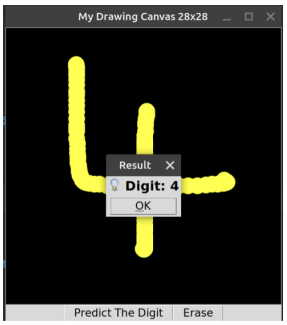
Assignment 3

Nandhakishore C S (DA24M011)

February 20, 2025

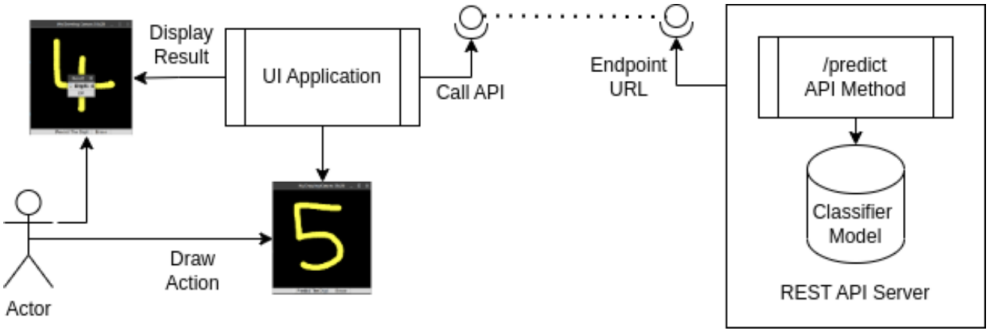
Problem Statement

We have learned about REST APIs, which has become a popular vehicle for exposing ML functionalities crafted in a Python environment. Let's build our first AI application which would allow an user to draw a number using the mouse pointer on an UI canvas, followed by recognising the number using an underlying ML classifier model. There is an application available already at https://github.com/sudarsun/HandwrittenDigitClassifier. The underlying model of this application is not great (a simple MLP) despite the accuracy level on the testing data is 90%. The apparent performance is 60% when you draw the digit on the UI canvas. Let's not worry about the correctness for now.



Task [50 points]

Task (50 points) If you look at the code base, you would observe that the model part and the UI part are tightly coupled. Basically, the modeling piece is being invoked as a native library call from the UI application. The objective of the task is to make them loosely coupled. Essentially, the modeling piece, i.e., the prediction functionality has to become an REST API end point (30 points). And the UI part should invoke the REST API call to get and display the predicted digit (20 points). You may choose to send the 28 x 28 gray scale image directly over the REST API (POST) method by putting the data in the message body. Or you may send the flattened 1-d normalized vector (each element will be in the 0.0-1.0 range) as a binary data to the API endpoint. If you choose the former, the API endpoint has to flatten the image, followed by normalizing the vector before giving it for model inference.



Module 1 [30 points]

The code base has two main sections:

1. A user defined python class, in which the drawing canvas is coded with which the model gets input as a PIL image, send it's to the Neural Network for prediction.

2. A user defined python function, with which the pickled model is loaded to do prediction.

To separate the application into two parts, the user defined function to load the ML model is migrated to a separate python file and it is converted into Rest API with Fast API and uvicorn server.
The server takes in the flattened vector of the image with 786 elements of size (786, 1). This is sent as a post request to the server. The server does prediction and sends the inference back to the app.

```python
# Data type to load the details from client.
class ImagePayload(BaseModel):
    # Expecting a flattened list of 784 values
    image: list

# create the webapp.
app = FastAPI(title="DA5402_A3")

@app.post("/predict")
def image_predict(payload:ImagePayload):
    try:
        # Convert list to NumPy array
        img_array = np.array(payload.image, dtype=np.float32).reshape(1, 784)  # Reshape for model
    input

        # Run prediction
        prediction = model.predict(img_array)
        predicted_digit = int(np.argmax(prediction))  # Get the class with highest probability

        # return {"prediction": predicted_digit} #, "confidence": float(np.max(prediction))}
        return {predicted_digit}

    except Exception as error:
        return {"error": str(error)}

# Run from command line: uvicorn apiWithBody:app --port 7000 --host 0.0.0.0
uvicorn.run(app, host='127.0.0.1', port=54320)
```

Figure 1: Snapshot of the code snippet with API

Module 2 [20 points]

The user defined class for getting the input via canvas is modified such that, the class has a method sends a requests.post request to the server to the model prediction. The response to the post request is then displayed as the output when the predict image button is used on canvas.

```python
def predict_image(self):
        # Convert the image to a vector and normalize the values (0 to 1)
        image_data = np.array(self.image).reshape(-1, ) / 255.0
        pay_load = {'image': image_data.tolist()}
        prediction = requests.post(URL, json = pay_load)
        messagebox.showinfo("Result", f"Digit: {prediction.text}")
```

Figure 2: Snapshot of the code snippet from the UI, getting input from server

Execution

To execute the assignment, open the directory with all the files in a terminal window.

1. Initiate a virtual environment and activate the environment:
   $ python3 -m venv env_name
   $ source env_name/bin/activate

2. Install the required python packages using pip.
   $ pip install -r requirements.txt

3. Create a connected terminal and run the server in the background.
   $ python3 server.py

4. In the adjacent terminal, run the application
   $ python3 app.py

The canvas appears to get the input and when the prediction is done in the UI, it invokes the API to do prediction in the server file.

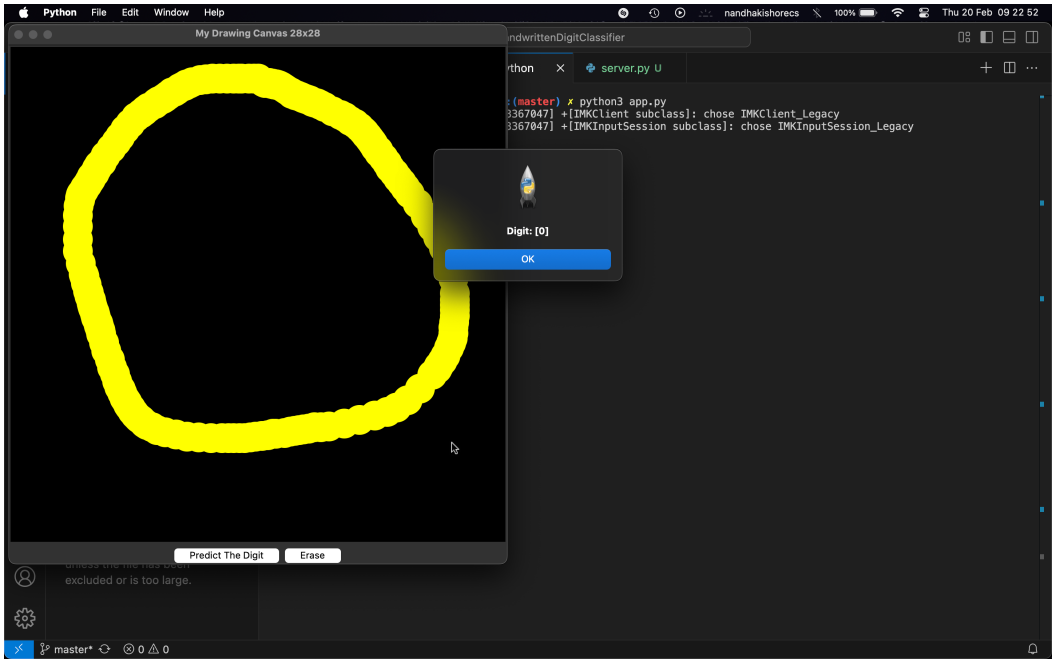


Figure 3: Screenshot of the application used with REST API

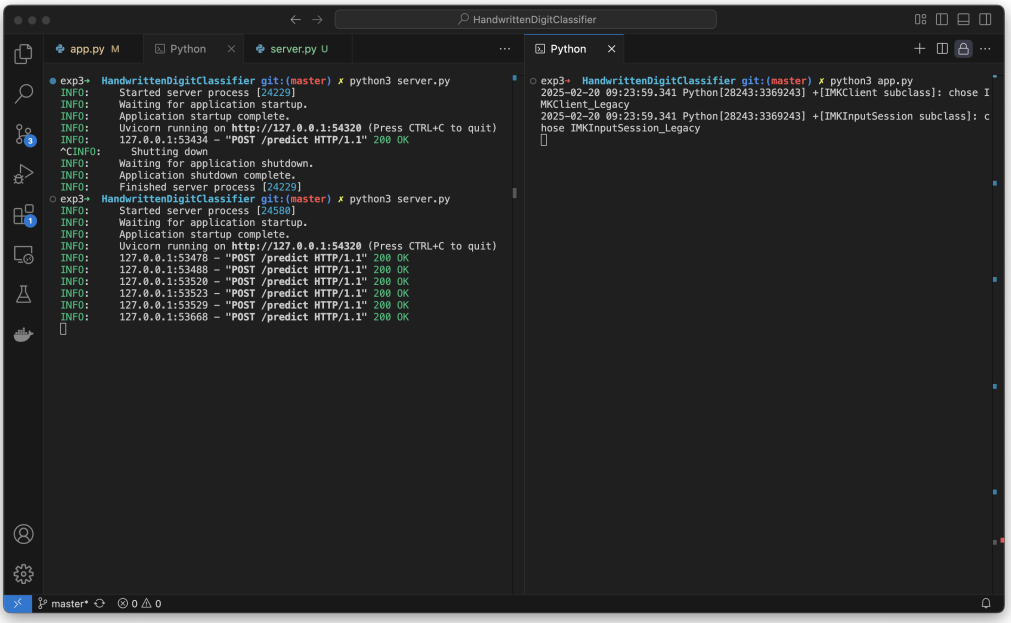


Figure 4: Screenshot of the terminal with server and application running