CS5830: Big Data Laboratory

Assignment 6: Build a FastAPI for MNIST digit prediction

ARUN PALANIAPPAN
ME20B036
me20b036@smail.iitm.ac.in

to
Department of Computer Science & Engineering
Indian Institute of Technology Madras
April 23, 2024

Contents

List of Figures

List of Tables

Build a FastAPI for MNIST digit prediction

In this assignment, we will be developing a FastAPI module that serves as an interface for MNIST digit prediction. The goal is to create a user-friendly API where users can upload images of handwritten digits, and our API will predict the digit shown in the image. We will start by selecting the best performing MNIST model from our previous assignment, which we'll load into our FastAPI module. Then, we'll create an endpoint that accepts image uploads, preprocesses the images to the required format, and passes them through our model to make predictions and output the digit. This entire project has been maintained in this GitHub repository.

1 Task 1

1.1 sub-task 1

We create a FastAPI module using this code line:

```
from fastapi import FastAPI
app = FastAPI()
```

1.2 sub-task 2

We take the model path as command line environment variable argument:

```
$env:MODEL_PATH="C:\Users\91979\Desktop\Jup_NoteBks\BDL\Asgt_6\model\mnist_exp_2.h5"
```

1.3 sub-task 3

Next we load the best model from previous assignment using this function:

```
def load_model(path: str) -> keras.Sequential:
    model = keras.models.load_model(path)
    return model
```

1.4 sub-task 4

Next we define a function to predict the digit which will take the image serialized as an array of 784 elements and returns the predicted digit as string:

```
def predict_digit(model, data_point: list) -> str:
    data = np.array(data_point).reshape(-1, 784) / 255.0
    prediction = model.predict(data)
    digit = np.argmax(prediction)
    return str(digit)
```

1.5 sub-task 5

Creating an API endpoint "`@app post('/predict')`" that will read the bytes from the uploaded image to create an serialized array of 784 elements.

```
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    contents = await file.read()
    img = Image.open(io.BytesIO(contents)).convert('L')
    data_point = img.flatten().tolist()
    digit = predict_digit(final_model, data_point)
    return {"digit": digit}
```

1.6 sub-task 6

To get the app running, we use `uvicorn` module.

```
$ uvicorn app_code:app

INFO:     Started server process [45716]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Swagger UI link: http://127.0.0.1:8000/docs

This is what the Swagger UI interface looks like:

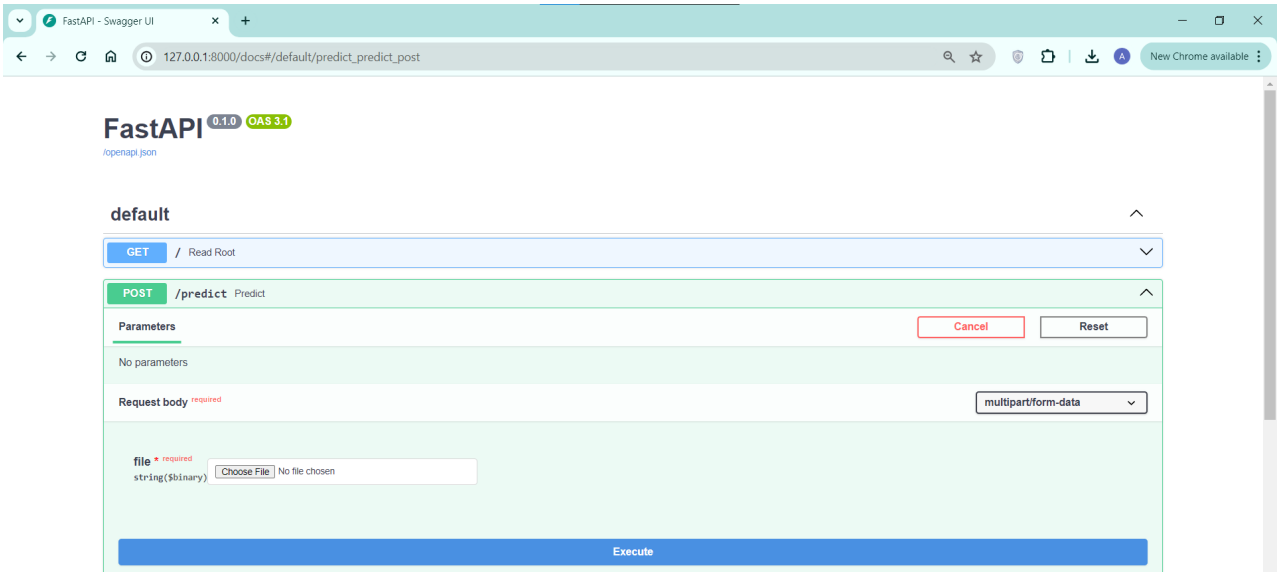


Figure 1: Swagger UI interface

Now, we will upload 28×28 images to the API and check the output:

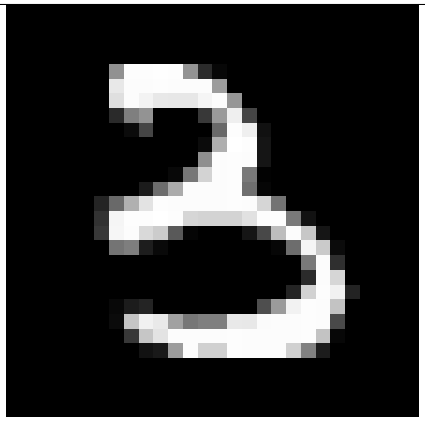
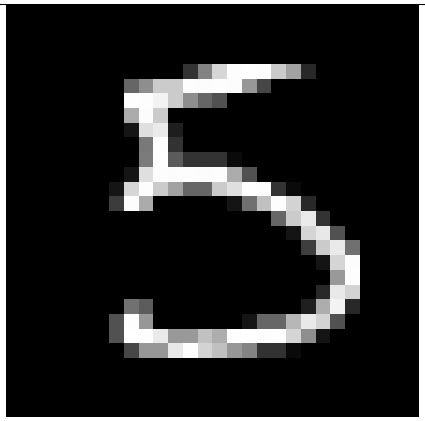
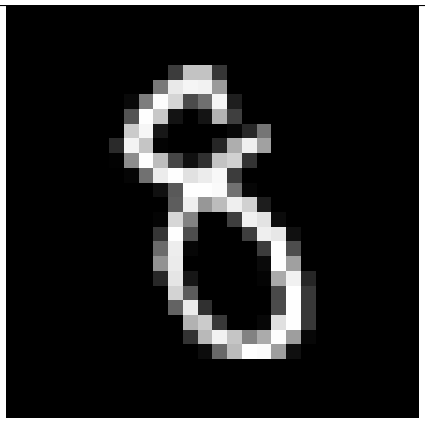
Input Image	Output Digit
	Code: 200 Details Response body { "digit": "3" } Response headers content-length: 13 content-type: application/json date: Mon,22 Apr 2024 17:10:39 GMT server: uvicorn
	Code: 200 Details Response body { "digit": "5" } Response headers content-length: 13 content-type: application/json date: Mon,22 Apr 2024 17:09:05 GMT server: uvicorn
	Code: 200 Details Response body { "digit": "8" } Response headers content-length: 13 content-type: application/json date: Mon,22 Apr 2024 17:09:47 GMT server: uvicorn

Table 1: `MNIST` dataset images

We can observe that, our model is very accurate when we pass `MNIST` dataset images.

2 Task 2

2.1 sub-task 1

Creating a new function which will resize any uploaded images to a 28×28 grey scale image:

```
def format_image(img):
    img_array = np.array(img.resize((28, 28)))
    return img_array
```

2.2 sub-task 2

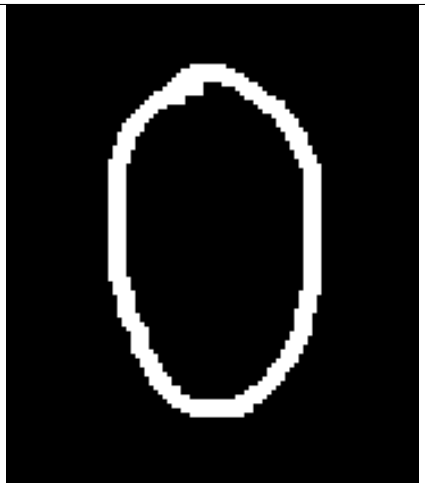
Incorporating `"format_image"` function inside the `"/predict"` endpoint to preprocess the uploaded content:

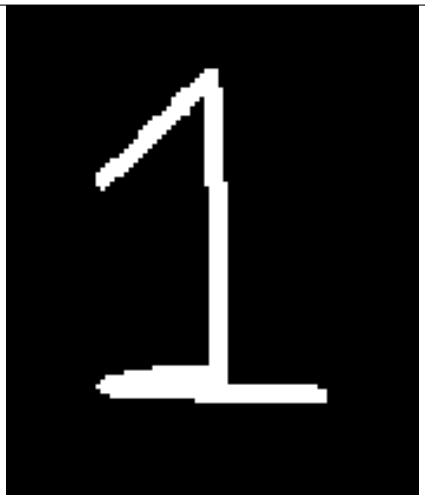
```python
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    contents = await file.read()
    img = Image.open(io.BytesIO(contents)).convert('L')
    img_array = format_image(img)
    data_point = img_array.flatten().tolist()
    digit = predict_digit(final_model, data_point)
    return {"digit": digit}
```

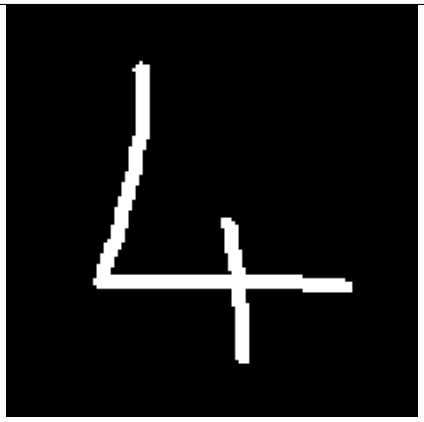
2.3 sub-task 3

Now I have drawn **10 digits** in MS Paint and given as input to the API. These are the outputs:

Input Image	Output Digit

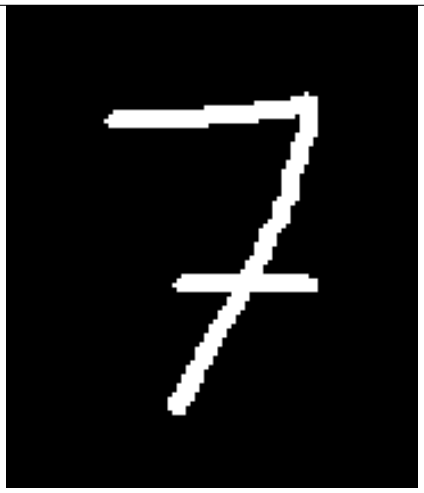
	Code: 200
	Details
	Response body
	`{ "digit": "0" }`
	Response headers
	`content-length: 13`
	`content-type: application/json`
	`date: Tue,23 Apr 2024 10:11:32 GMT`
	`server: uvicorn`

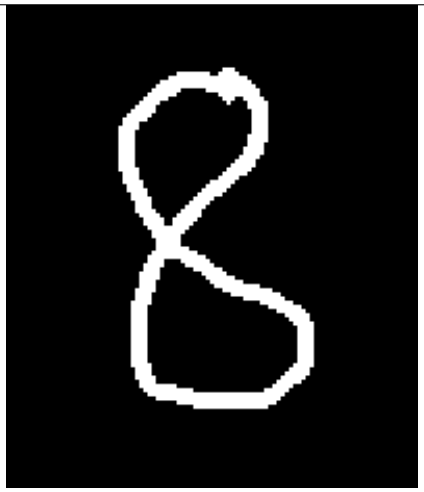
	Code: 200
	Details
	Response body
	`{ "digit": "1" }`
	Response headers
	`content-length: 13`
	`content-type: application/json`
	`date: Tue,23 Apr 2024 10:12:25 GMT`
	`server: uvicorn`

Input Image	Output Digit
	Code: 200 Details Response body { "digit": "2" } Response headers content-length: 13 content-type: application/json date: Tue,23 Apr 2024 10:12:53 GMT server: uvicorn
	Code: 200 Details Response body { "digit": "3" } Response headers content-length: 13 content-type: application/json date: Tue,23 Apr 2024 10:13:17 GMT server: uvicorn
	Code: 200 Details Response body { "digit": "4" } Response headers content-length: 13 content-type: application/json date: Tue,23 Apr 2024 10:13:42 GMT server: uvicorn
	Code: 200 Details Response body { "digit": "5" } Response headers content-length: 13 content-type: application/json date: Tue,23 Apr 2024 10:15:06 GMT server: uvicorn

Input Image	Output Digit

	Code: 200
	Details
	Response body
	{
	"digit": "6"
	}
	Response headers
	content-length: 13
	content-type: application/json
	date: Tue,23 Apr 2024 10:15:28 GMT
	server: uvicorn

	Code: 200
	Details
	Response body
	{
	"digit": "1"
	}
	Response headers
	content-length: 13
	content-type: application/json
	date: Tue,23 Apr 2024 10:16:05 GMT
	server: uvicorn

	Code: 200
	Details
	Response body
	{
	"digit": "8"
	}
	Response headers
	content-length: 13
	content-type: application/json
	date: Tue,23 Apr 2024 10:17:02 GMT
	server: uvicorn

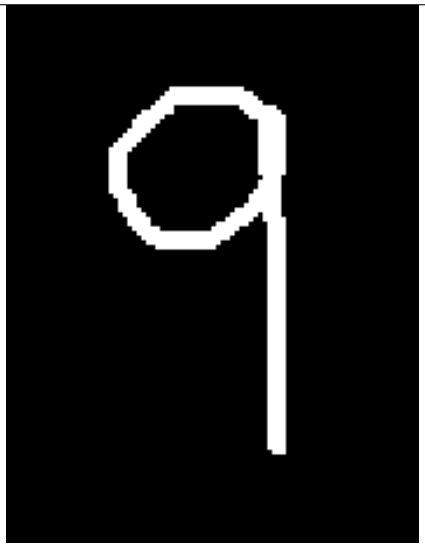
Input Image	Output Digit
	

Table 2: 10 Custom hand-drawn images prediction

Performance:

- We can observe that our model has predicted **{9/10}** digit correctly. It predicted digit 7 as 1 which is quite reasonable as they look similar.
- Hence the model performance for these 10 digits is **90%**

3 Conclusions

- Hence we have successfully deployed and tested a FastAPI for our `MNIST` model.
- The model performance for the 10 hand-drawn digits is **90%**
- The entire project has been maintained in this GitHub repository.

———*Thank You*———