Data Intake Report

Name: hello_flask – Digital Classification using PyTorch, REST API, JSON and FLASK(Heroku part will come in later)
Report date: September 28th, 2023
Internship Batch: LISUM25
Version:<1.0>
Data intake by: Alison March
Data intake reviewer:
Data storage location: https://github.com/alisonjing/hello_flask

Tabular data details:https://github.com/alisonjing/hello_flask/tree/master/app/data/MNIST/raw

Total number of observations	60,000 training images and 10,000 testing images of handwritten digits, each image is 28 x 28(= 784) pixels.
Total number of files	8(4 ubyte, 4 ubyte.gz)
Total number of features	10 (digits from 0 – 9)
Base format of the file	Ubyte.gz
Size of the data	63.4 MB

Proposed Approach:

- First, we created a new directory in the terminal called hello_flask and set cd to hello_flask(in Anaconda Prompt)
- Next, we created a virtual environment named venv using this command line:
  conda create -n venv python=3.11.5 anaconda
- Activate the new virtual environment using activate venv
- Libraries and Packages Installation
  - flask
  - pytorch: torch and torchvision
  - waitress
  - gevent
- Create a folder and inside the folder create main.py. In this file, we created a predict function that completes the below 4 steps:
    - Loan image
    - Convert image to tensor
    - Make a prediction.
    - Return json data
  - The following packages are imported in the main.py

```
import os
print('current directory')
print(os.getcwd())

from flask import Flask, request, jsonify
from flask_cli import FlaskCLI
from gevent.pywsgi import WSGIServer
from torch_utils import transform_image, get_prediction
```

- Testing if Flask is running once main.py is setup:
  - In the terminal(vscode) cd app
  - I am using windows system laptop, so the commands are as follows:
    - set FLASK_APP = main.py (export FLASK_APP =main.py for linux/macOS)
    - set FLASK_DEBUG = development (export FLASK_ENV = development for linux/macOS)
    - flask run

```
D:\Documents\Data_Science\Internship\Data_Glacier\hello_flask
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

- Create a folder test and inside the test folder create a test.py

```
import requests
resp = requests.post("http://localhost:5000/predict", files = {'file': open('four.png', 'rb')} )
print(resp.text)      You, 2 days ago • Deploy models using Flask and Heroku wth PyTorch …
```

***We added in files={} part later in the resp line.**

- In the app directory, we create a new python file **torch_utils.py**

```
import io
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from PIL import Image
```

  - Next, we create a NeuralNet class and use a feedforward neural network to train and save the model, this is the load model part
  - Set the Hyper-parameters as below condition:
    - **Input_size= 784** (based on the 28 by 28 image size)
    - **Hidden_size = 500**
    - **Num_classes = 10 (0-9)**
    - **Model = NueralNet(input_size, Hidden_size, Num_classes)**
    - **PATH = "mnist_ffn_pth"**
    - **Model.load_state_dict(torch.load(PATH))**
    - **Model.eval()**
  - We convert image -> tensor: create a function called transform_image(image_bytes):

```
def transform_image(image_bytes):
    transform = transforms.Compose([transforms.Grayscale(num_output_channels=1),
                                    transforms.Resize((28,28)),
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.1307,), (0.3081,))]) #global mean and std
    image = Image.open(io.BytesIO(image_bytes))
    return transform(image).unsqueeze(0)
```

    ***We used the global mean and standard deviation values to normalize the () transformation. In addition, we set the to Grayscale image transformation and output 1 channel.**
  - Finally, we created a prediction () to output the predicted image in terms of value.

```
#predict
def get_prediction(image_tensor):
    images = image_tensor.reshape(-1, 28*28)
    outputs = model(images)
    # max returns (value, index)
    _, predicted = torch.max(outputs.data, 1)
    return predicted
```

- Add **mnist_ffn.pth** file in the hello_flask/app directory.

- **Main.py part:**
  - Import libraries:

```
import os
print('current directory')
print(os.getcwd())

from flask import Flask, request, jsonify
from flask_cli import FlaskCLI
from gevent.pywsgi import WSGIServer
from torch_utils import transform_image, get_prediction
```

  - # Create an app
    app=Flask(__name__)

  - Set up debug mode
  - Define allowed image extensions if we select ‘png’, ‘jpg’, or ‘jpeg’.
  - Create a function named allowed_file(filename) with .(dot) with maximum of 1 split, and lowercase letters mentioned in allowed image extensions.
  - Create a predict() that predicts the image of a handwritten digit.
  - Finally returns the result in jsonify().

- To test if main.py and torch_utils.py are implemented correctly, in the test.py we change the file to the actual saved image, in this case, is “seven.png”, below is the complete line of code:

```
import requests
resp = requests.post("http://localhost:5000/predict", files = {'file': open('seven.png', 'rb')} )
print(resp.text)
```

where ‘rb’ means read binary mode.

We can test this by keeping Flask running in the background, opening an Anaconda Prompt terminal, switching to the current project directory and switching to virtual environment **venv**, and cd to test folder, then run **python test.py**:

Result:

```
(venv) D:\Documents\Data_Science\Internship\Data_Glacier\hello_flask\test>python test.py
{"class_name":"7","prediction":7}
```

The result predicted the handwritten digit correctly (input image).
Note: this will not run properly if Flask is not running.

To test this code again, I created an image using PAINT app

```
(venv) D:\Documents\Data_Science\Internship\Data_Glacier\hello_flask\test>python test.py
{
  "class_name": "4",
  "prediction": 4
}
```

and made a handwritten four “4”(in 28 by 28-pixel size with black canvas and white paintbrush color) and saved it as four.png before updating the image in the test.py.

- Create a file wsgi.py and input the following line: from app.main import app
- We then create a **Procfile** file for Heroku and input the following line: Web: waitress-serve wsgi:app
- Create a **runtime.txt** with python version: python-3.11.5