BÁO CÁO ĐỒ ÁN NHÓM B3

"NHẬN DẠNG CHỮ SỐ VIẾT TAY"

THỰC HÀNH NHẬP MÔN CÔNG NGHỆ THÔNG TIN

GV HƯỚNG DẪN: THẦY PHẠM MINH HOÀNG

1/ Thông tin nhóm: Nhóm B3, gồm có 03 sinh viên:

_ 22120189: Nguyễn Minh Lợi.

_ 22120282: Song Đồng Gia Phúc.

_ 22120333: Nguyễn Quang Thắng.

2/ Chức năng: Chương trình gồm có 07 chức năng sau:

2.1/ Hàm `Flatten(matrix):`

_ Cho phép duỗi một ma trận m x n về một mảng 1 chiều gồm m x n phần tử.

_ Nhận đầu vào là ma trận `matrix`.

_ Trả về mảng một chiều.

2.2/ Hàm `Average(matrix, r = 2, c = 2):`

_ Cho phép giảm kích thước mảng bằng cách tính giá trị trung bình ở một vài phần tử kề nhau và lấy giá trị đó làm một phần tử ở ma trận mới.

_ Nhận đầu vào là ma trận cần giảm kích thước `matrix`, hai tham số `r` và `c` tương ứng là số hàng và số cột muốn gộp lại. Giá trị mặc định là `r = 2` và `c = 2` nếu không truyền vào.

_ Trả về ma trận đã giảm kích thước.

2.3/ Hàm `Histogram(matrix):`

_ Cho phép lưu lại cường độ các giá trị màu sắc có trong hình.

_ Nhận đầu vào là ma trận `matrix`.

_ Trả về mảng 256 phần tử tương ứng với cường độ 256 giá trị màu sắc có trong hình.

2.4/ Hàm `guess(matrix, method = 1, KNN = 500, r = 2, c = 2):`

_ Hàm dự đoán số được lưu trong hình `matrix` là số nào.

_ Nhận các đầu vào:

+ Ma trận `matrix`.

+ `method`: phương pháp dùng để dự đoán. Có 3 phương pháp tương ứng với các mục **2.1**, **2.2** và **2.3**. Nếu không chọn phương pháp, mặc định sẽ dùng phương pháp 1: Flatten.

+ `KNN`: Số lượng các hình gần nhất muốn xét. Sau khi quét qua hết tất cả các hình trong dataset, thuật toán sẽ chọn ra `K` hình gần gần giống với hình `matrix` nhất và xét lại lần nữa.

+ `r` và `c` để cho hàm `Average(matrix, r = 2, c = 2)` nếu dùng phương pháp này.

2.5/ Hàm `cal_accuracy():`

_ Tính tỉ lệ dự đoán đúng của từng phương pháp trong các phương pháp nêu trên.

_ Hàm không nhận tham số đầu vào. Bên trong hàm có mảng `KNNarray` để nhập tất cả các giá trị K muốn kiểm tra.

_ Trả về tỉ lệ phần trăm dự đoán đúng của từng phương pháp, ứng với từng giá trị của `K`.

2.6/ Thư viện lib.so: code bằng ngôn ngữ C/C++

_ Khi gọi hàm hàm `guess()` ở mục **2.4,** thay vì hàm xử lý bằng ngôn ngữ Python, hàm có thể gọi hàm `guess_optimize()` được viết bằng ngôn ngữ C/C++.

_ Tốc độ chạy chương trình tăng đáng kể khi sử dụng thư viện này, do tốc độ xử lý của C/C++ nhanh hơn nhiều lần so với Python.

2.7/ Hàm `import_image(img_path):`

_ Xử lý ảnh của người dùng nhập. Ảnh được chụp bằng camera của người dùng.

_ Xử dụng thư viện `cv2` của Python để hỗ trợ xử lý.

_ Nhận đầu vào là đường dẫn của ảnh.

Chỉ nhận các file ảnh (VD: .png; .jpg; …).

_ Trả về ma trận kích thước 28 x 28 là ảnh đã qua xử lý.

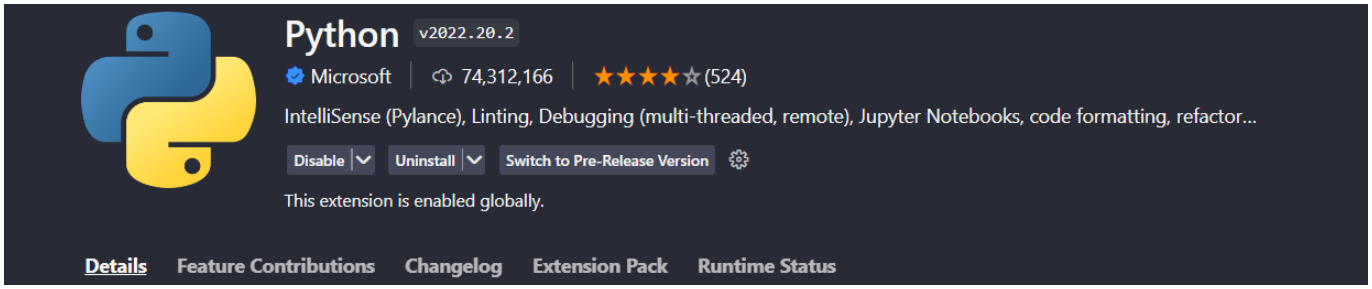
3/ Hướng dẫn:

3.1/ Môi trường lập trình:

_ Python 3: Tải tại https://docs.conda.io/en/latest/miniconda.html

_ Visual Studio Code: Tải tại https://code.visualstudio.com/Download/

_ Extension **Python**:



_ Thư viện **numpy:** gõ lệnh `pip install numpy` vào terminal Python.

_ Thư viện **matplotlib:** gõ lệnh `pip install matplotlib` vào terminal Python.

_ Thư viện **cv2:** gõ lệnh `pip install opencv-python` vào terminal Python.

3.2/ Tập dữ liệu MNIST (MNIST dataset).

_ Tải tại http://yann.lecun.com/exdb/mnist/

_ Tập dữ liệu MNIST gồm 70,000 ảnh có kích thước 28 x 28 là các chữ số được viết tay. Trong đó có 60,000 ảnh là cơ sở dữ liệu dùng dự đoán, và 10,000 ảnh dùng để kiểm tra tính đúng đắn của thuật toán dự đoán được xây dựng.

_ Sau khi tải xong, lưu các file trên vào thư mục **"data"** cùng đường dẫn với file mã nguồn Python.

_ Copy đoạn code sau và chạy thử để kiểm tra quá trình tải thư viện và cài đặt môi trường lập trình có thành công hay không. Đoạn code trên được lưu trong file `test_MNIST.py`.

test_MNIST.py > ...

```python
import matplotlib.pyplot as plt
import os
import numpy as np
import gzip

def load_mnist(path, kind='train'):
    """Load MNIST data from 'path' """
    labels_path = os.path.join(path, '%s-labels-idx1-ubyte.gz' % kind)
    images_path = os.path.join(path, '%s-images-idx3-ubyte.gz' % kind)

    with gzip.open(labels_path, 'rb') as lbpath:
        lbpath.read(8)
        buffer = lbpath.read()
        labels = np.frombuffer(buffer, dtype = np.uint8)

    with gzip.open(images_path, 'rb') as imgpath:
        imgpath.read(16)
        buffer = imgpath.read()
        images = np.frombuffer(buffer, dtype = np.uint8).reshape(len(labels), 28, 28).astype(np.float64)

    return images, labels

#-----------------------------------------------------------------------------
X_train, y_train = load_mnist('data/', kind='train')
X_test, y_test = load_mnist('data/', kind='t10k')
print('MNIST train size: %d, img size: %d x %d' % (X_train.shape[0], X_train.shape[1], X_train.shape[2]))

fig, ax = plt.subplots(nrows=4, ncols=5, sharex=True, sharey=True,)
ax = ax.flatten()
for i in range(0, 10):
    img = X_train[y_train == i][0]
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[i].set_title(i)
for i in range(10, 20):
    img = X_train[i]
    ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[i].set_title(y_train[i])

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
plt.show()

```

3.3/ Tính tỉ lệ đoán đúng của từng phương pháp:

_ Mở file `main.py`.

_ Ở hàm `cal_accuracy()`, thêm vào mảng `KNNarray` những giá trị `K` muốn thử nghiệm. Chẳng hạn, như hình bên thử nghiệm với k = 10, 100, 500.

```
def cal_accuracy():
    KNNarray = [10, 100, 500]
```

_ Gọi hàm `cal_accuracy()` và chạy file `main.py`.

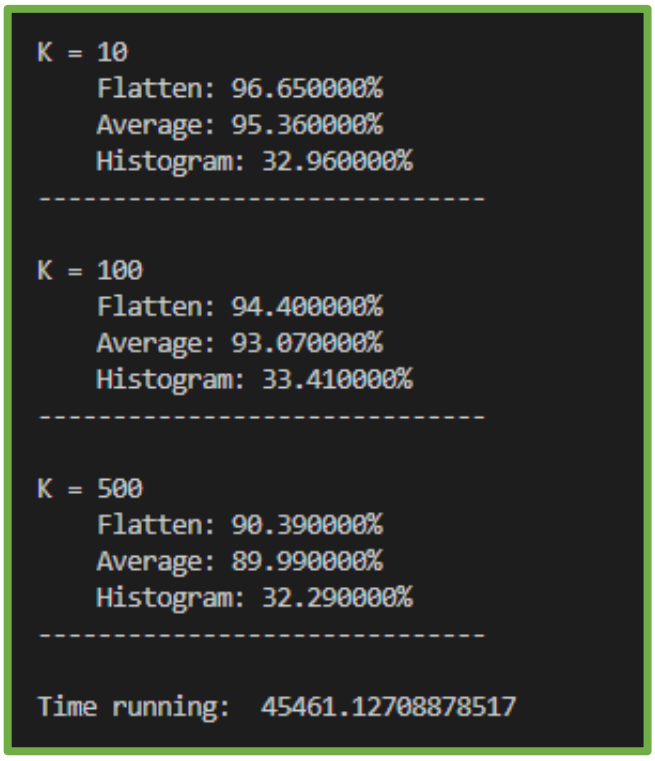
Kết quả ví dụ:

```
--------------------
K = 10
    Flatten: 96.000000%
    Average: 100.000000%
    Histogram: 36.000000%
------------------------------
K = 100
    Flatten: 96.000000%
    Average: 97.000000%
    Histogram: 34.000000%
------------------------------
K = 500
    Flatten: 89.000000%
    Average: 91.000000%
    Histogram: 35.000000%
------------------------------
```

Hình 1:

_ Chạy thử nghiệm với 100 test đầu.

_ Rút đặc trưng trung bình 2 x 2.



Hình 2:

_ Chạy thử nghiệm với toàn bộ 10,000 test.

_ Rút đặc trưng trung bình 4 x 4.

3.4/ Tối ưu tốc độ: sử dụng C/C++

Dùng các hàm viết bằng C/C++ trong thư viện lib.so.

B1: Đặt file `lib.so` cùng đường dẫn với file `main_optimize.py`.

_ `lib.so` là thư viện tự định nghĩa, chứa các hàm được viết lại bằng ngôn ngữ C/C++ để tăng tốc độ chạy.

_ Nếu không có file `lib.so`, hoặc muốn chỉnh sửa thư viện, chỉnh sửa trong hai file `lib.cpp` và `lib.hpp`, sau đó dùng lần lượt hai lệnh sau để dịch chương trình thành file `lib.so`:

```
gcc -std=c++14 -c -o lib.o lib.cpp
```

```
gcc -shared -o lib.so lib.o
```

Lưu ý: phải có Compiler C/C++ để thực hiện biên dịch.

B2: Chạy file `main_optimize.py` thay vì file `main.py`. Sự khác biệt duy nhất giữa hai file là ở hàm `guess()`, file main.py sẽ xử lý bằng ngôn ngữ python, trong khi đó file `main_optimize.py`, hàm `guess()` sẽ gọi hàm `guess_optimize()` được viết bằng ngôn ngữ C/C++.

3.5/ Chức năng phụ: **import ảnh:**

_ Gán biến src_img là đường dẫn ảnh.

_ Gọi hàm img = ImportImage(src_img).

_ Trả về ảnh đã qua xử lý (được lưu vào biến img).

_ Gọi hàm guess(img) với ảnh này và nhận kết quả.

_ Do độ tương quan giữa ảnh được chụp tay với ảnh có sẵn trong dataset nên kết quả nhận được có tỉ lệ đúng thấp hơn. Nếu xử lý thêm được vấn đề độ lệch giữa vị trí tương quan thì sẽ nhận được kết quả với tỉ lệ đúng cao hơn.