

HỌC VIỆN VTI



ĐỒ ÁN CUỐI KHÓA

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG
NHẬN DẠNG CHỮ VIẾT TAY**

Giảng viên hướng dẫn: THS. Trần Anh Tuấn

Lớp: Python 2301

Học viên thực hiện: Hà Tuấn Anh

Hà Nội, 11/2023

MỤC LỤC

| | |
|--|----|
| LỜI CẢM ƠN | 2 |
| LỜI NÓI ĐẦU | 3 |
| CHƯƠNG 1 Giới thiệu về bài toán Nhận dạng chữ viết tay bằng CNN. | 4 |
| 1.1 Giới thiệu về nhận dạng chữ viết tay..... | 4 |
| 1.2 Giới thiệu về mạng CNN..... | 5 |
| 1.2.1 Giới thiệu | 5 |
| 1.2.2 Lóp tích chập - Convolution Layer..... | 6 |
| 1.2.3 Bước nhảy - Stride | 9 |
| 1.2.4 Đường viền - Padding | 9 |
| 1.2.5 Hàm phi tuyến - ReLU..... | 9 |
| 1.2.6 Lóp gộp - Pooling Layer..... | 10 |
| 1.2.7 Tóm tắt..... | 11 |
| 1.3 Ứng dụng CNN vào nhận dạng chữ viết tay | 12 |
| CHƯƠNG 2 Giải thích cách hoạt động của chương trình và trình bày kết quả thực nghiệm | 14 |
| 2.1 Trình bày cách hoạt động của chương trình. | 14 |
| 2.1.1 Quá trình thu thập dữ liệu..... | 14 |
| 2.1.2 Quá trình xử lý dữ liệu..... | 22 |
| 2.2 Trình bày kết quả của chương trình..... | 31 |
| CHƯƠNG 3 Đánh giá và kết luận. | 39 |

LỜI CẢM ƠN

Với tình cảm chân thành và sâu sắc nhất, em mong muốn được bày tỏ đến tất cả cá nhân, tổ chức đã tạo điều kiện hỗ trợ và giúp đỡ em trong suốt quá trình nghiên cứu đề tài. Trong thời gian qua, em đã nhận được nhiều sự quan tâm của giảng viên hướng dẫn, học viện và các anh chị trong trung tâm.

Trải qua thời gian vừa qua, em đã nhận được sự quan tâm và đồng hành từ anh Mentor. Đây là những nguồn động viên và định hướng quan trọng trong cuộc hành trình nghiên cứu và học tập của em.

Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc tới thầy ThS. Trần Anh Tuấn, người đã truyền đạt vốn kiến thức cần có cho em trong suốt quá trình học tập khóa học. Sự tận tâm và sự chỉ bảo tận tình của anh đã giúp em hoàn thành đề tài một cách tốt nhất.

Mặc dù báo cáo học phần của em được thực hiện trong khoảng thời gian ngắn, chắc chắn sẽ không tránh khỏi những sai sót. Vì vậy, em mong nhận được sự đóng góp ý kiến từ các anh chị trong trung tâm và anh mentor để giúp em hoàn thiện kiến thức và bổ sung thêm thông tin cần thiết cho báo cáo học phần của mình.

Một lần nữa, em xin chân thành cảm ơn anh Tuấn và tất cả những người đã giúp đỡ và đồng hành cùng em trong chặng đường này

Em xin chân thành cảm ơn!

Học viên thực hiện

Hà Tuấn Anh

LỜI NÓI ĐẦU

Trong thời đại 4.0, công nghệ thông tin là một phần không thể thiếu và cũng như quá đỗi quen thuộc với mỗi người dân chúng ta. Nhắc đến công nghệ thông tin, ta sẽ nghĩ ngay đến sự tiện lợi, tối ưu, nhanh nhẹn và đặc biệt là những thứ mới mẻ. Vậy công nghệ thông tin có thể xử lý những thứ gì thiết thực cho chúng ta, có thể trở thành một người “trợ lý ảo” cho chúng ta hay không? Tất nhiên là có. Đó là sự ra đời của trí tuệ nhân tạo (artificial intelligence). Với trí tuệ nhân tạo, chúng ta có thể đưa máy móc đến gần con người hơn và khiến cho những cỗ máy đó có thể “biết nghĩ, biết học...”. Và cũng để nghiên cứu và khám phá khả năng này của trí tuệ nhân tạo, em đã xây dựng nên chương trình về “Nhận dạng chữ viết tay” bằng mô hình CNN (Convolutional Neural Network) tức là mạng nơ ron tích chập. Đây là một sản phẩm nghiên cứu rất hay và thực tế khi có thể chuyển đổi những nét vẽ, những suy nghĩ của con người bằng chính thao tác tay để biến về thành những con số máy tính. Và trong báo cáo này, em sẽ trình bày 3 phần liên quan đến nội dung xây dựng nên chương trình thực nghiệm trên:

Chương 1: Giới thiệu về bài toán Nhận dạng chữ viết tay bằng CNN.

Giới thiệu tổng quan về nhận dạng chữ viết tay, giới thiệu về CNN, cách áp dụng CNN vào nhận dạng chữ viết tay

Chương 2: Giải thích cách hoạt động của chương trình và trình bày kết quả thực nghiệm

Giải thích cách hoạt động của chương trình (flow), các đoạn code sử dụng như nào và có tác dụng gì

Chương 3: Đánh giá và kết luận.

Nhận xét kết quả thực nghiệm và nêu ra hướng phát triển trong tương lai.

CHƯƠNG 1 Giới thiệu về bài toán Nhận dạng chữ viết tay bằng CNN.

1.1 Giới thiệu về nhận dạng chữ viết tay

Nhận dạng chữ viết tay, một lĩnh vực tiên tiến trong nghiên cứu trí tuệ nhân tạo, đặt ra những thách thức đầy thú vị trong việc chuyển đổi và hiểu biết về các dòng chữ viết tay thành dạng văn bản có thể xử lý bởi máy tính. Đây không chỉ là một bước tiến quan trọng để tự động hóa quy trình nhập liệu mà còn là cơ hội để máy tính hiểu và tương tác với thế giới bằng ngôn ngữ của con người.

Bài toán nhận dạng chữ viết tay không chỉ phức tạp với sự biến đổi nhanh chóng của nét chữ mà còn đối mặt với sự đa dạng trong cách mà mỗi cá nhân thể hiện sự sáng tạo qua cách viết của mình. Trong khi con người có thể dễ dàng nhận diện và hiểu biết những đặc điểm riêng biệt này, máy tính cần sự hỗ trợ mạnh mẽ từ các thuật toán máy học và mô hình học sâu để thực hiện nhiệm vụ này.

Mục tiêu chính của nhận dạng chữ viết tay là chuyển đổi các hình ảnh hoặc dữ liệu chữ viết tay thành văn bản có thể xử lý được. Các ứng dụng của nó là đa dạng, bao gồm việc nhận dạng chữ trong ảnh văn bản, xử lý tài liệu cổ điển, và thậm chí chuyển đổi những ghi chú tay tưởng chừng vô hình thành dữ liệu số, tạo ra sự thuận tiện và linh hoạt trong quản lý thông tin cá nhân và tổ chức công việc.

Bên cạnh việc giúp máy tính hiểu và xử lý ngôn ngữ viết tay, bài toán này còn đóng vai trò quan trọng trong các lĩnh vực như chữ ký số và xác minh danh tính. Sự phát triển của nhận dạng chữ viết tay đã mở ra không gian rộng lớn cho các ứng dụng di động, nâng cao trải nghiệm người dùng và cung cấp những công cụ mạnh mẽ để tương tác thông tin.

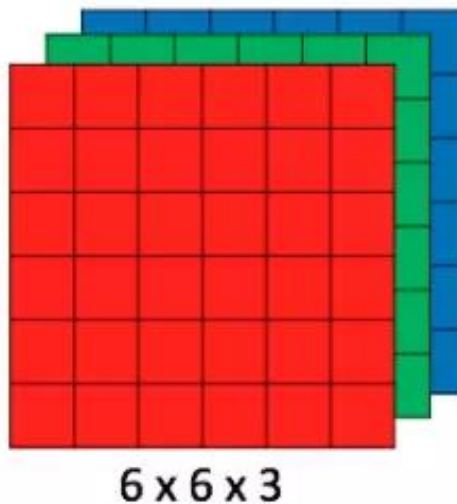
Trong bối cảnh cuộc sống ngày càng chuyển đến môi trường số, bài toán nhận dạng chữ viết tay không chỉ đóng vai trò làm cầu nối giữa thế giới ảo và thế giới thực mà còn đem lại những tiện ích vô song trong việc tạo ra giao diện linh hoạt và hiệu quả giữa con người và máy tính.

1.2 Giới thiệu về mạng CNN

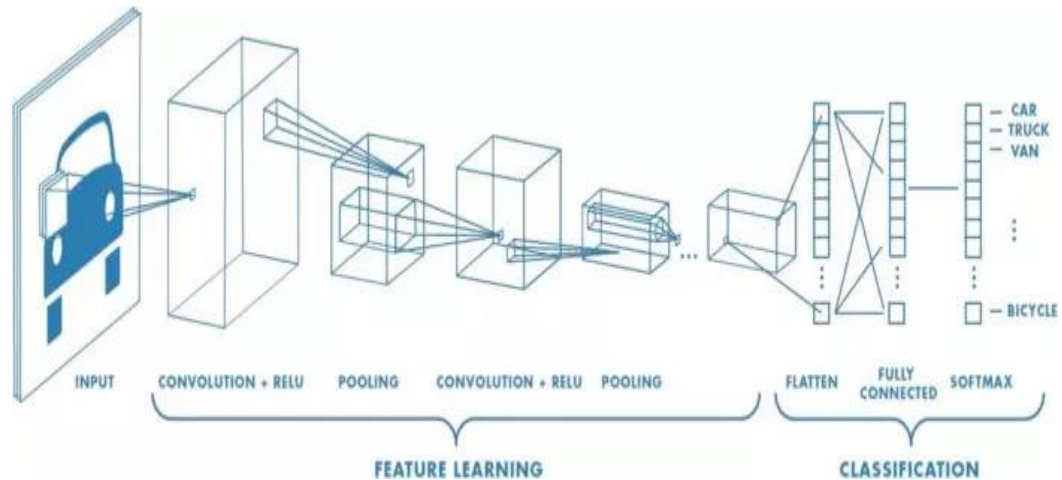
1.2.1 Giới thiệu

Trong mạng neural, mô hình mạng neural tích chập (CNN) là 1 trong những mô hình để nhận dạng và phân loại hình ảnh. Trong đó, xác định đối tượng và nhận dạng khuôn mặt là 1 trong số những lĩnh vực mà CNN được sử dụng rộng rãi.

CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định (Ví dụ: Chó, Mèo, Hổ, ...). Máy tính coi hình ảnh đầu vào là 1 mảng pixel và nó phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy $H \times W \times D$ (H: Chiều cao, W: Chiều rộng, D: Độ dày). Ví dụ: Hình ảnh là mảng ma trận RGB $6 \times 6 \times 3$ (3 ở đây là giá trị RGB).



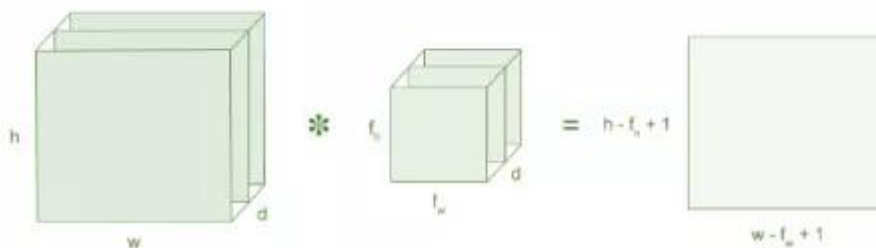
Về kỹ thuật, mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập với các bộ lọc (Kernels), tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm Softmax để phân loại đối tượng có giá trị xác suất giữa 0 và 1. Hình dưới đây là toàn bộ luồng CNN để xử lý hình ảnh đầu vào và phân loại các đối tượng dựa trên giá trị.



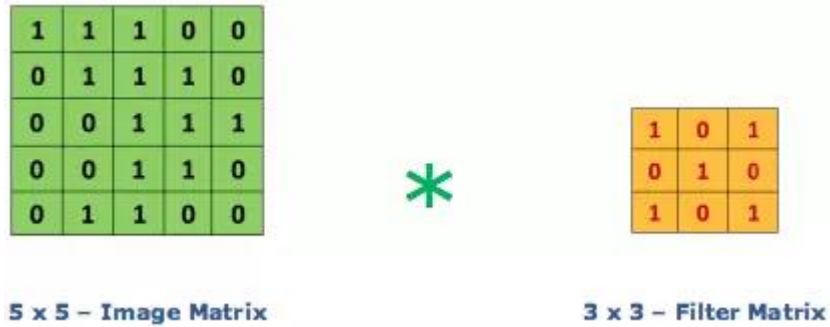
1.2.2 Lớp tích chập - Convolution Layer

Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vuông nhỏ của dữ liệu đầu vào. Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân.

- An image matrix (volume) of dimension **$(h \times w \times d)$**
- A filter **$(f_h \times f_w \times d)$**
- Outputs a volume dimension **$(h - f_h + 1) \times (w - f_w + 1) \times 1$**

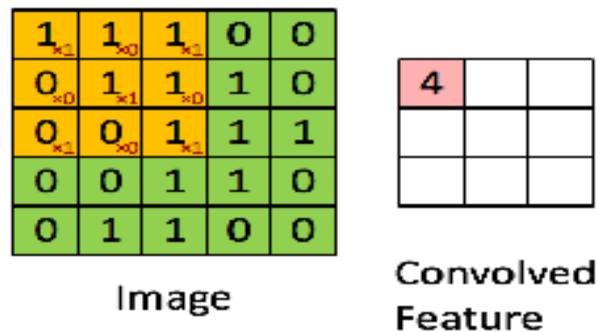


Xem xét 1 ma trận 5 x 5 có giá trị pixel là 0 và 1. Ma trận bộ lọc 3 x 3 như hình bên dưới.










-
-

Sau đó, lớp tích chập của ma trận hình ảnh 5 x 5 nhân với ma trận bộ lọc 3 x 3 gọi là 'Feature Map' như hình bên dưới.

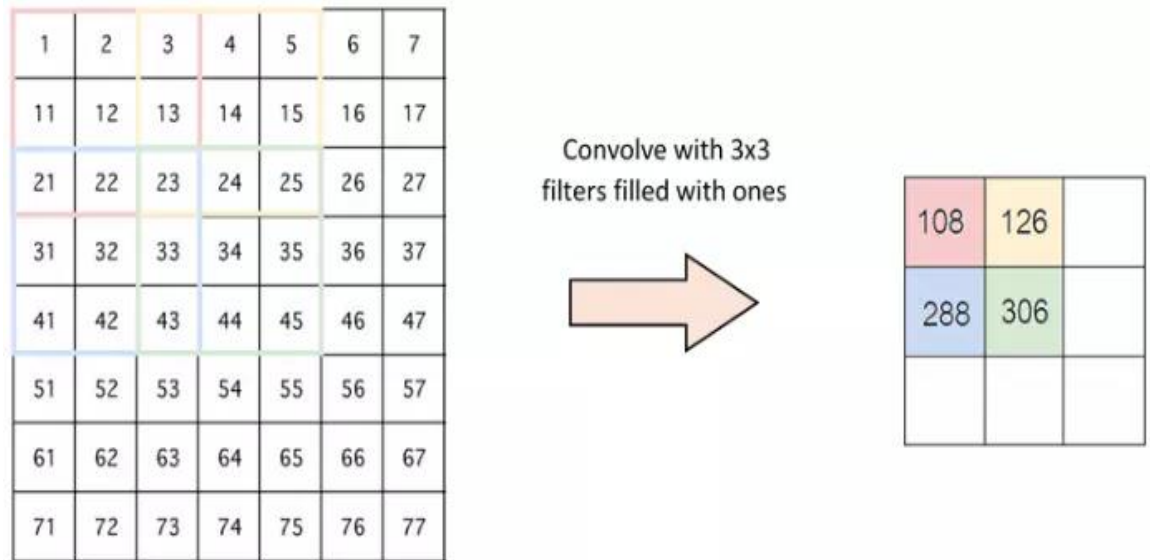


Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc. Ví dụ dưới đây cho thấy hình ảnh tích chập khác nhau sau khi áp dụng các Kernel khác nhau.

| Operation | Filter | Convolved Image |
|----------------------------------|--|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |
| Gaussian blur (approximation) | $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ |  |

1.2.3 Bước nhảy - Stride

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.



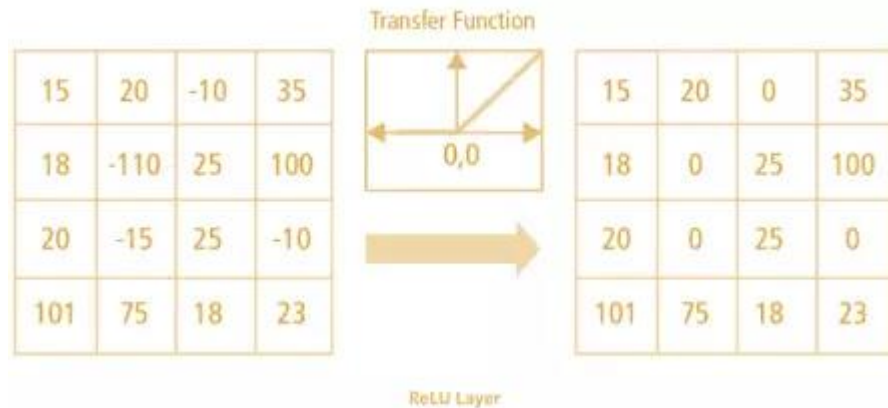
1.2.4 Đường viền - Padding

- Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:
- Chèn thêm các số 0 vào 4 đường biên của hình ảnh (padding)
- Cắt bớt hình ảnh tại những điểm không phù hợp với kernel.

1.2.5 Hàm phi tuyến - ReLU

ReLU viết tắt của Rectified Linear Unit, là 1 hàm phi tuyến. Với đầu ra là: $f(x) = \max(0, x)$.

Tại sao ReLU lại quan trọng: ReLU giới thiệu tính phi tuyến trong ConvNet. Vì dữ liệu trong thế giới mà chúng ta tìm hiểu là các giá trị tuyến tính không âm.



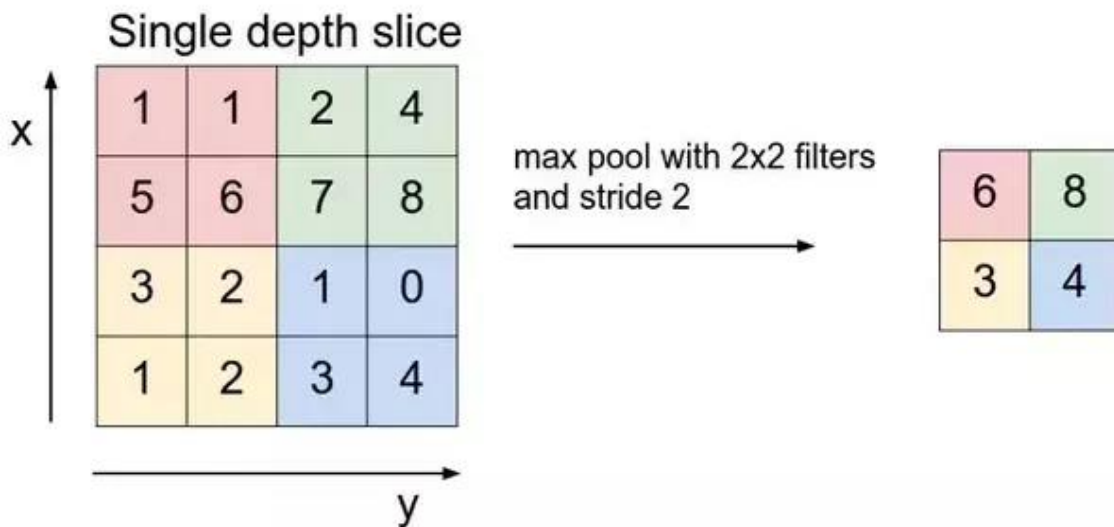
Có 1 số hàm phi tuyến khác như `tanh`, `sigmoid` cũng có thể được sử dụng thay cho ReLU. Hầu hết người ta thường dùng ReLU vì nó có hiệu suất tốt.

1.2.6 Lớp gộp - Pooling Layer

Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau:

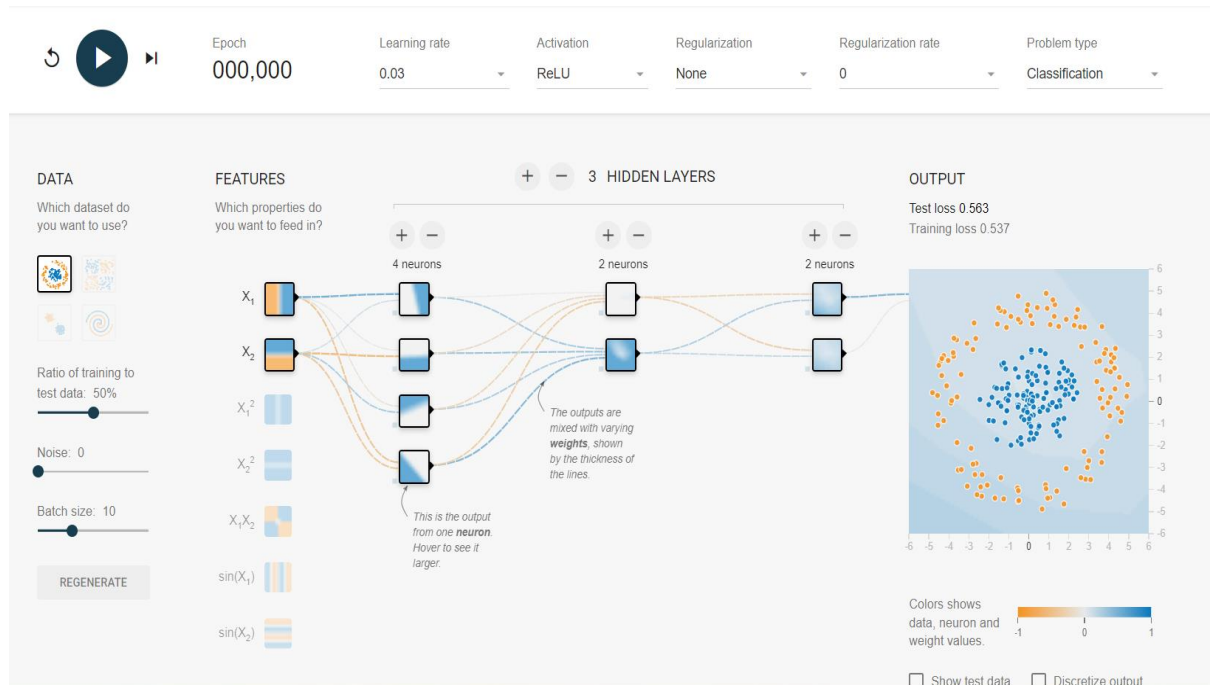
- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling



1.2.7 Tóm tắt

- Đầu vào của lớp tích chập là hình ảnh
- Chọn đối số, áp dụng các bộ lọc với các bước nhảy, padding nếu cần. Thực hiện tích chập cho hình ảnh và áp dụng hàm kích hoạt ReLU cho ma trận hình ảnh.
- Thực hiện Pooling để giảm kích thước cho hình ảnh.
- Thêm nhiều lớp tích chập sao cho phù hợp
- Xây dựng đầu ra và dữ liệu đầu vào thành 1 lớp được kết nối đầy đủ (Full Connected)
- Sử dụng hàm kích hoạt để tìm đối số phù hợp và phân loại hình ảnh.



1.3 Ứng dụng CNN vào nhận dạng chữ viết tay

Bài toán nhận dạng chữ viết tay được ứng dụng rất nhiều trong thực tế, được tích hợp vào hệ thống nhận dạng form tự động, tích hợp trong các máy PDA có màn hình cảm ứng, nhận dạng chữ ký... Do có nhiều ứng dụng quan trọng như vậy nên từ lâu bài toán nhận dạng chữ viết tay đã thu hút sự quan tâm của nhiều nhà nghiên cứu. Nghiên cứu của Norhidayu và các tác giả [5] sử dụng các mô hình phân loại SVM, KNN và mạng nơ-ron. Kết quả thực nghiệm cho thấy mô hình sử dụng thuật toán phân loại KNN cho kết quả phân loại cao nhất là 99,26%. Nghiên cứu của Ana và các tác giả [1] đã sử dụng mô hình nhị phân cục bộ (Local Binary Pattern - LBP) như là một bộ trích xuất đặc trưng và bộ phân loại KNN trên hệ thống nhận dạng chữ viết tay của họ trên mẫu C1 được sử dụng bởi ủy ban bầu cử ở Indonesia.

Kết quả thực nghiệm cho thấy phương pháp LBP có thể nhận dạng ký tự chữ số viết tay trên bộ dữ liệu MNIST với độ chính xác 89,81% và trên dữ liệu

C1 với độ chính xác là 70,91%. Souici-Meslati [8] trình bày một cách tiếp cận lại để nhận dạng số lượng chữ trên ngân phiếu. Các tác giả sử dụng ba bộ phân loại chạy song song: mạng nơ-ron, KNN và Fuzzy K-nearest neighbor. Các kết quả đầu ra được kết hợp từ cả ba bộ phân loại này. Kết quả thực nghiệm trên bộ dữ liệu của họ đạt độ chính xác là 96%. Burrow [7] áp dụng bộ phân loại KNN trên tập dữ liệu của họ và tác giả đạt được độ chính xác là 74%. Jason [3] sử dụng mô hình mạng CNN với việc xây dựng các bộ lọc với các kích thước khác nhau nhằm trích lọc được nhiều thông tin hữu ích trong ảnh. Tác giả đã xây dựng thực nghiệm trên tập dữ liệu MNIST và đạt được kết quả cao với độ chính xác đạt 99,31%.

Từ việc phân tích các nghiên cứu trên, chúng tôi nhận thấy các nghiên cứu này đã sử dụng nhiều mô hình khác nhau cũng như xây dựng các thực nghiệm trên các bộ dữ liệu khác nhau. Tuy mạng nơ-ron đã được áp dụng trong một số nghiên cứu, nhưng cấu trúc mạng của các nghiên cứu này là tương đối đơn giản, chưa khai thác hết các tính năng của các lớp trong mạng. Chính vì vậy, trong nghiên cứu này chúng tôi muốn đề xuất xây dựng một mô hình mới với sự kết hợp của các lớp trong CNN với Multi-layer Perceptron (MLP) nhằm đạt được kết quả tốt hơn cho bài toán nhận dạng chữ số viết tay.

CHƯƠNG 2 Giải thích cách hoạt động của chương trình và trình bày kết quả thực nghiệm

2.1 Trình bày cách hoạt động của chương trình.

2.1.1 Quá trình thu thập dữ liệu

Tập dữ liệu trong bài toán này gồm 2 phần: Dữ liệu sẵn có từ thư viện MNIST và dữ liệu tự thu thập.

- Tập dữ liệu MNIST

Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology) là một trong những bộ dữ liệu quan trọng và phổ biến nhất trong lĩnh vực học máy và thị giác máy tính. Được tạo ra từ dữ liệu chữ số viết tay, bộ dữ liệu MNIST đã trở thành một "bài kiểm tra" phổ biến cho các mô hình học máy và thuật toán nhận dạng hình ảnh.

Bộ dữ liệu này bao gồm một tập hợp của 70,000 ảnh chữ số từ 0 đến 9, được chia thành hai phần chính: một phần dành cho việc huấn luyện (training set) và một phần dành cho việc kiểm thử (test set). Mỗi ảnh trong tập dữ liệu có kích thước 28x28 pixel, tạo ra một ma trận hình ảnh có độ phân giải thấp so với nhiều bộ dữ liệu hình ảnh khác.

Dữ liệu trong MNIST được chuẩn hóa để có giá trị pixel nằm trong khoảng từ 0 đến 255, thể hiện mức độ xám của mỗi pixel. Mỗi ảnh được tương ứng với một nhãn chỉ định chữ số tương ứng từ 0 đến 9, điều này giúp dễ dàng sử dụng bộ dữ liệu để phát triển và kiểm thử các mô hình học máy dựa trên thị giác máy tính.

Bộ dữ liệu MNIST thường được sử dụng như một điểm xuất phát cho việc phát triển và kiểm thử các mô hình học máy cơ bản, đặc biệt là trong lĩnh vực của học máy sâu (deep learning).

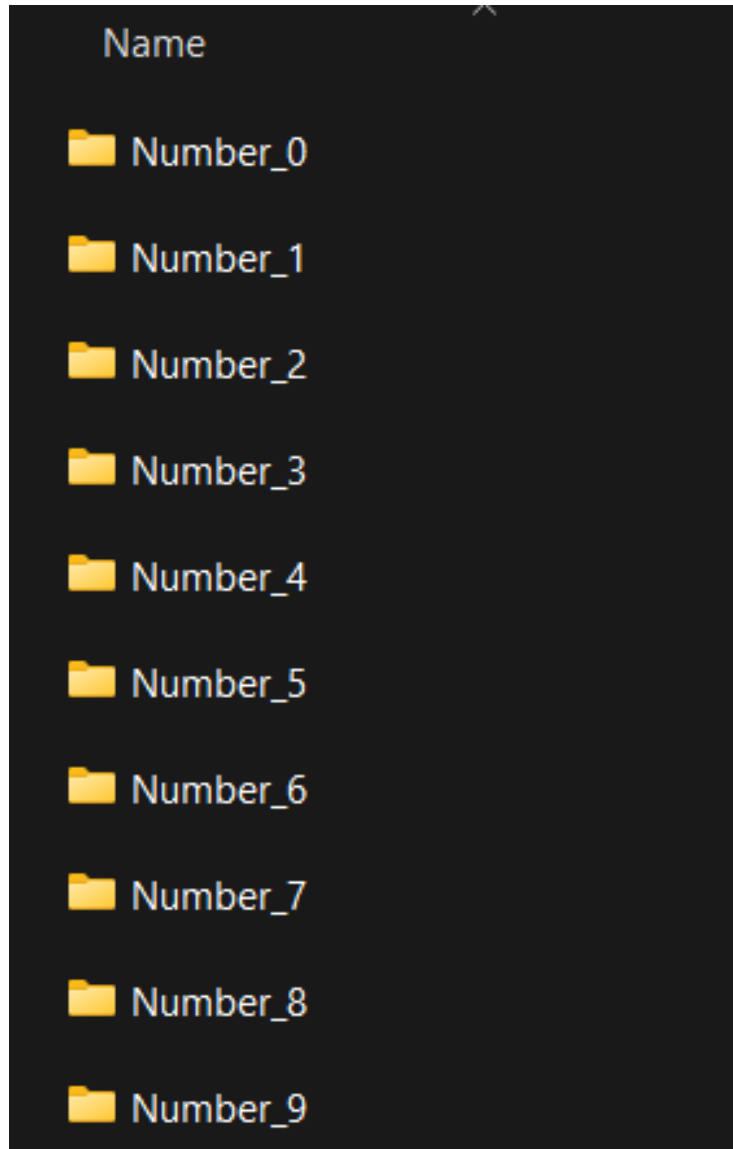
Việc nhận diện chữ số viết tay trên MNIST đã trở thành một thách thức phổ biến và thường được sử dụng để đánh giá hiệu suất của các mô hình học máy, từ các mô hình cơ bản như Mạng Nơ-ron Tích chập (CNN) đến các mô hình học máy sâu phức tạp hơn.

```
from keras.datasets import mnist
```

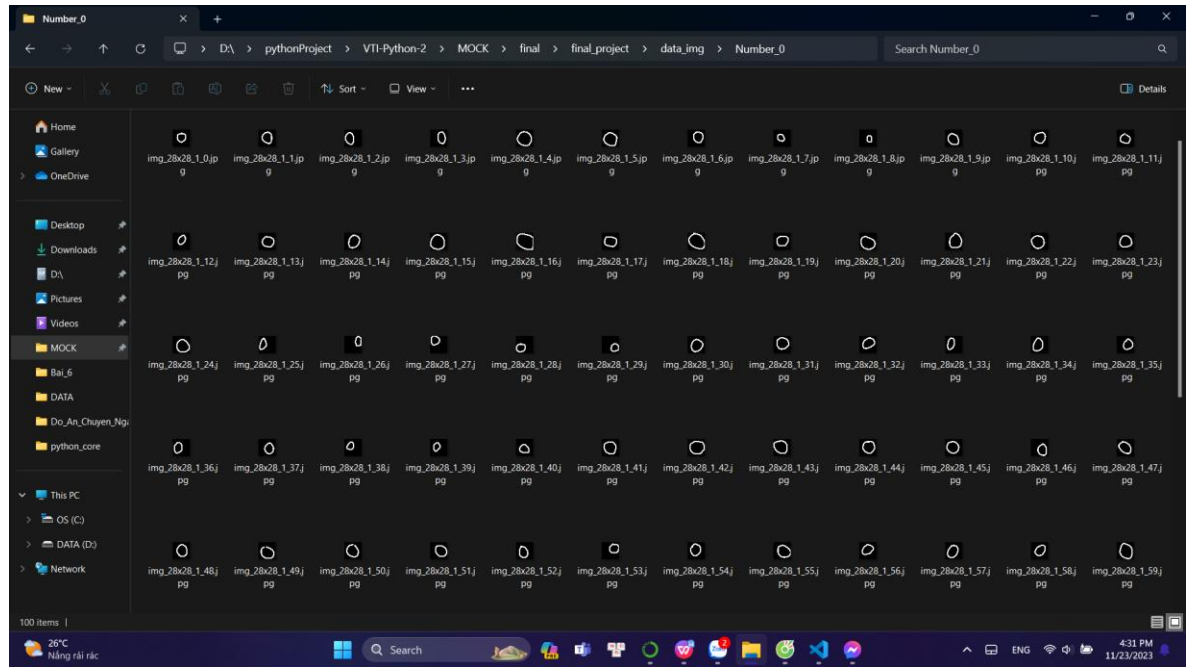


Dữ liệu tự thu thập

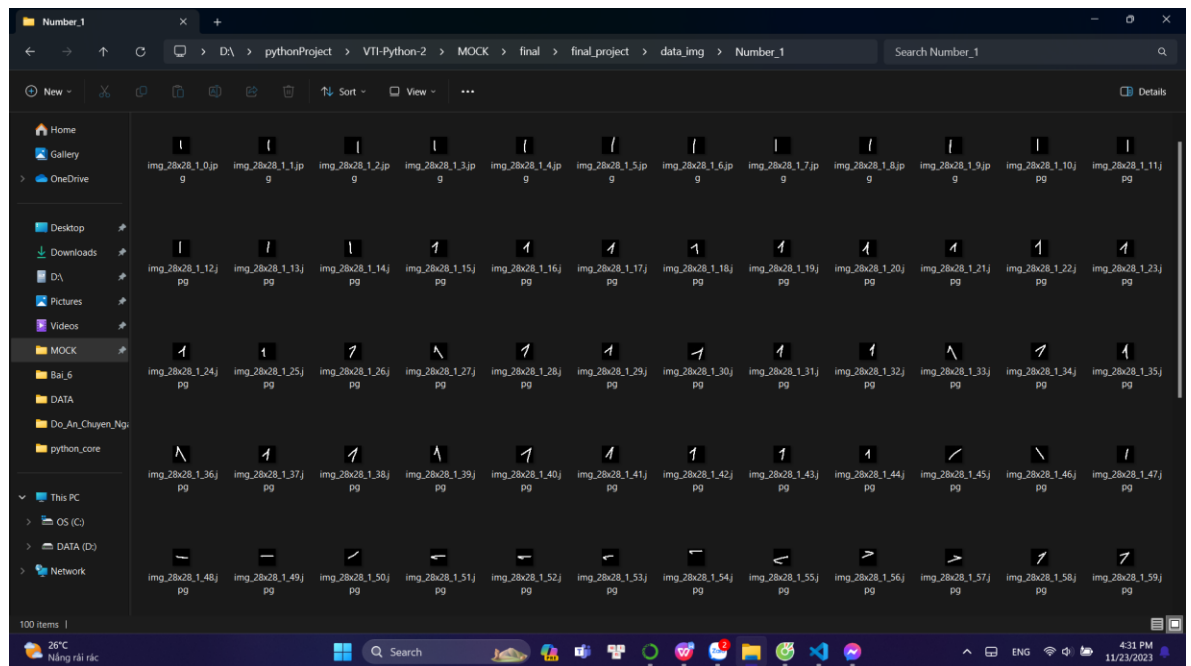
Dữ liệu tự thu thập là dữ liệu mà em tự viết bằng tay, gồm các số từ 0 đến 9, mỗi chữ số bao gồm 100 mẫu, tổng là 1000 dữ liệu tự thu thập.



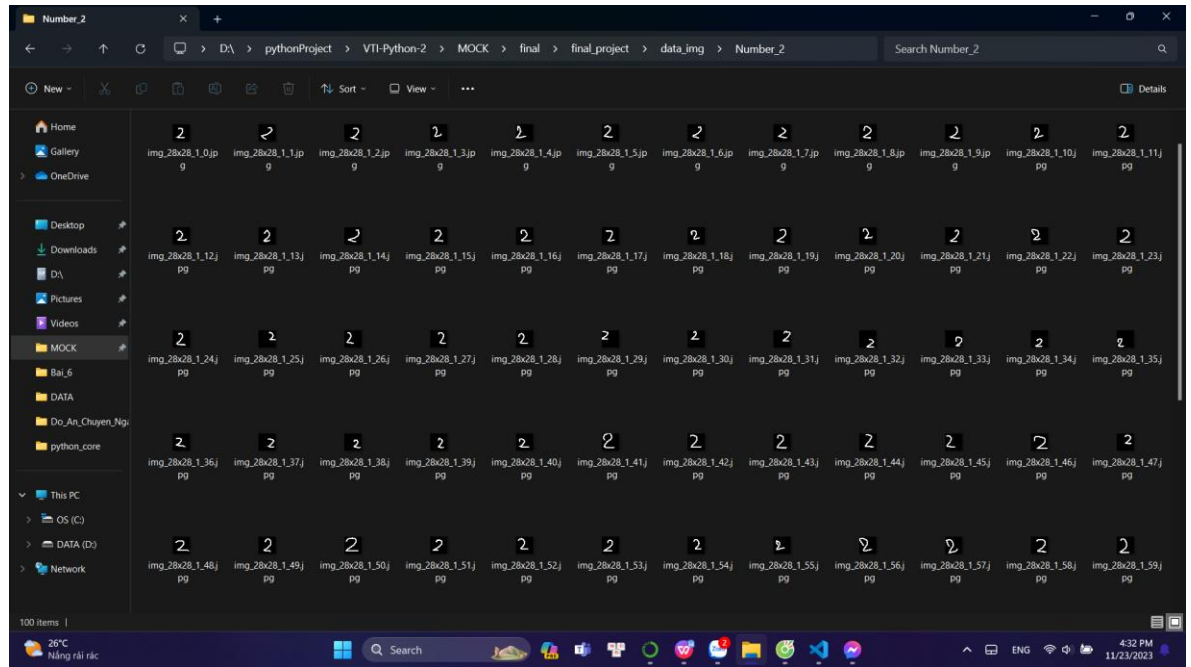
Số 0:



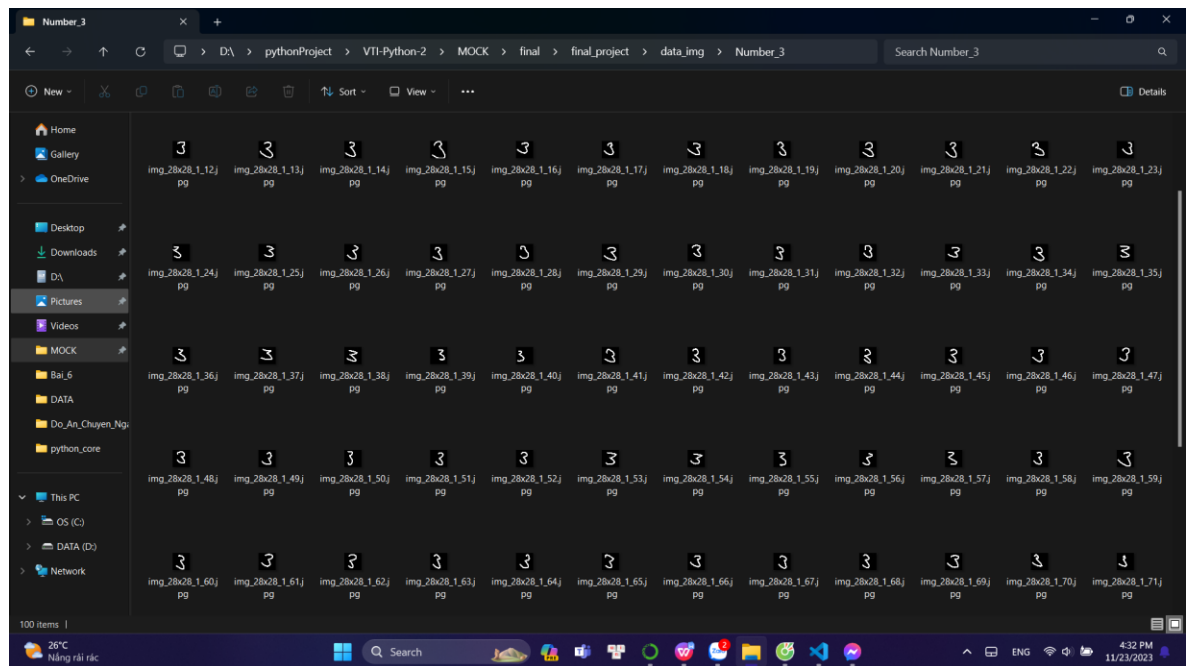
Số 1:



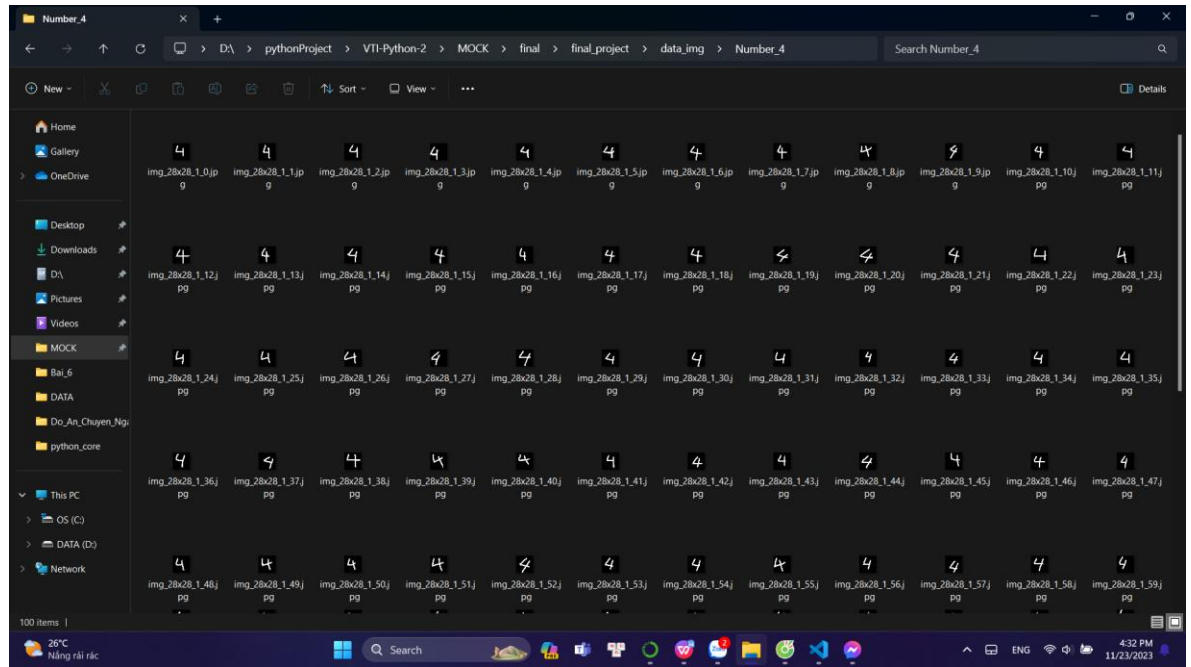
Số 2:



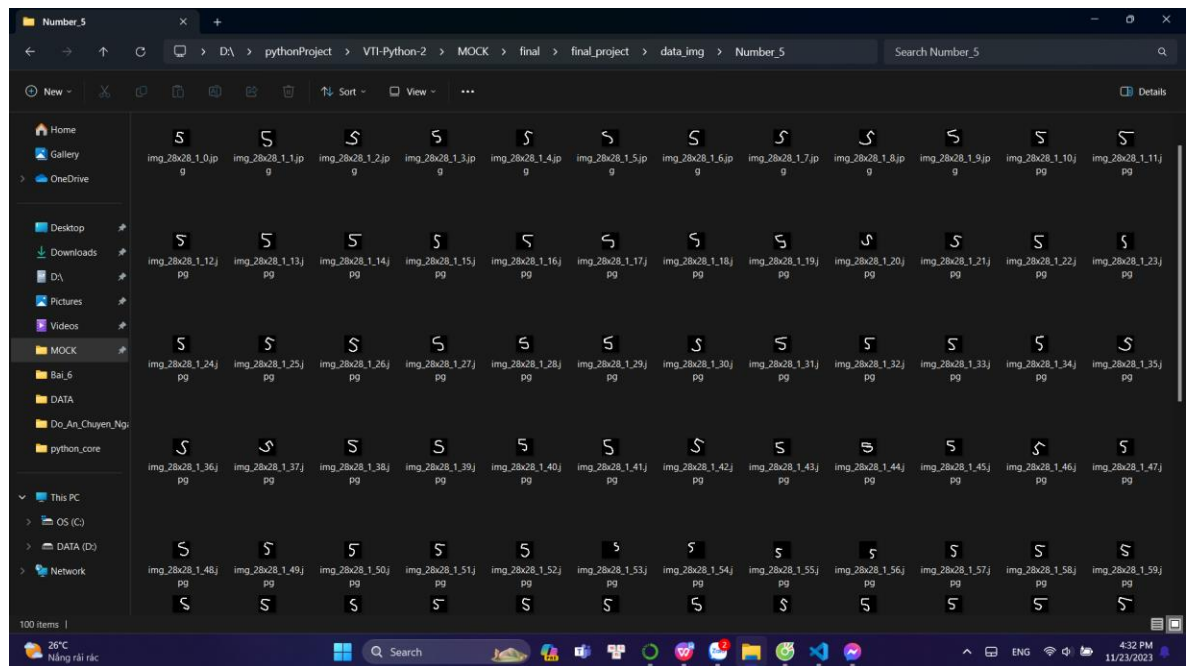
Số 3:



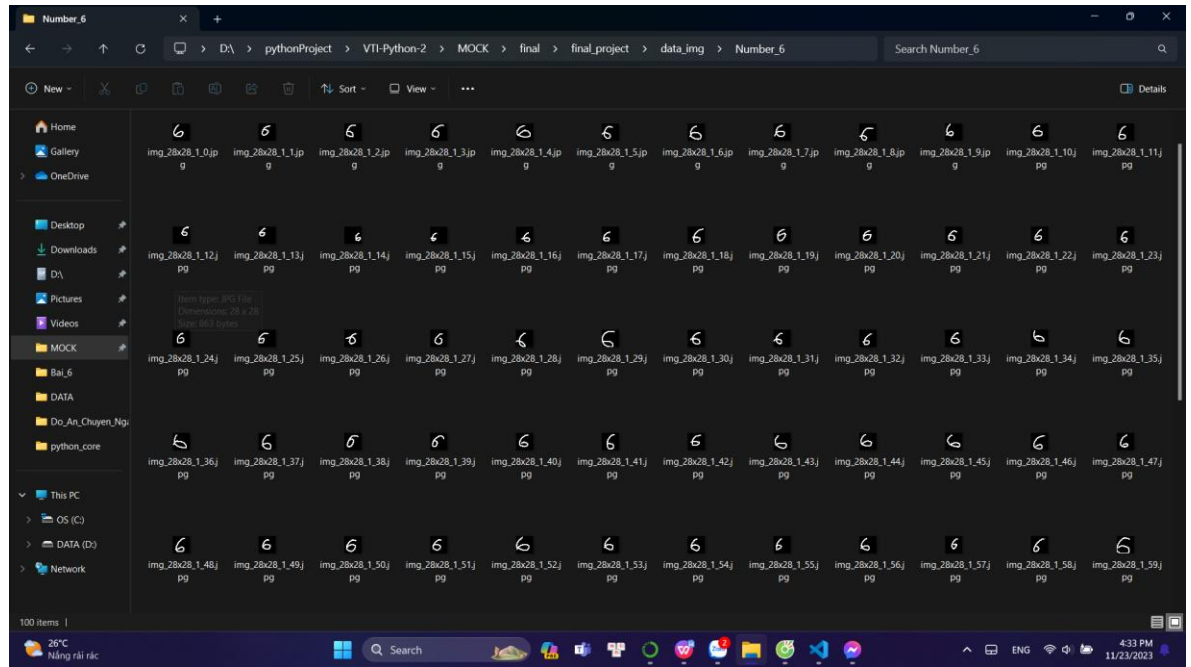
Số 4:



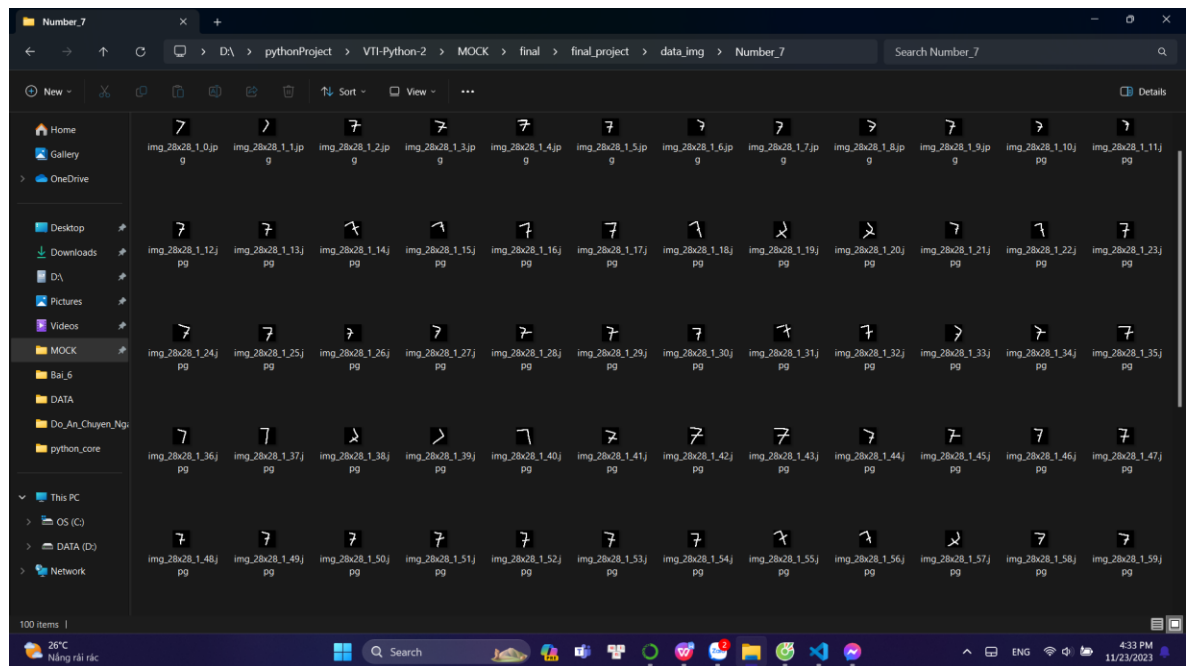
Số 5:



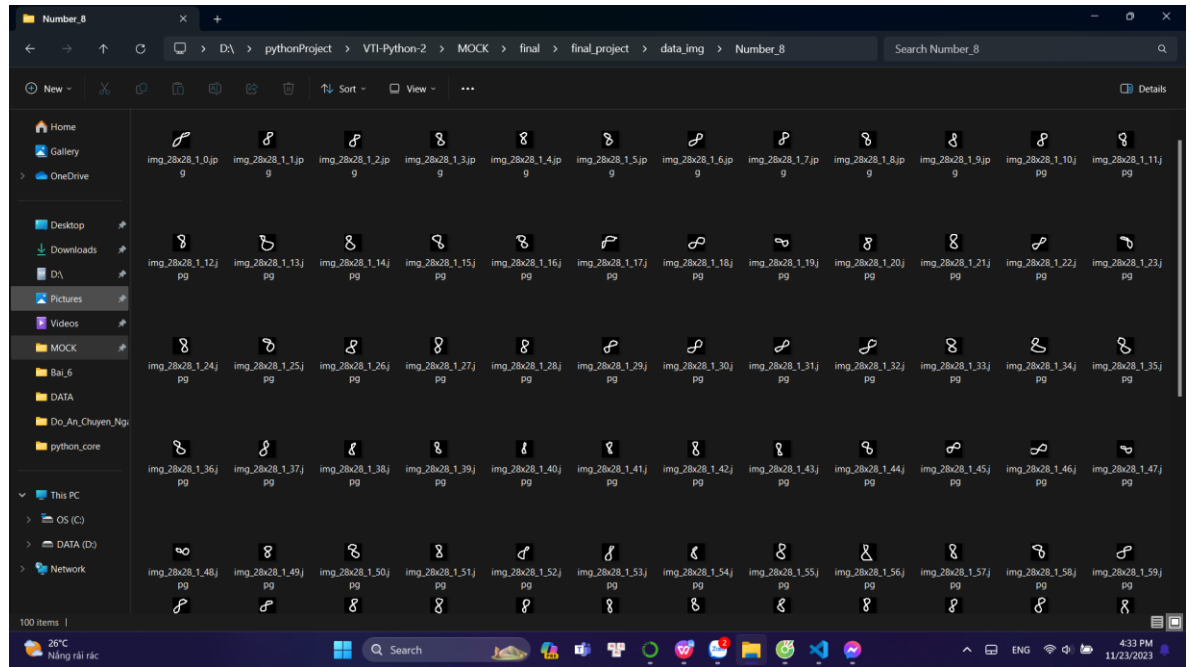
Số 6:



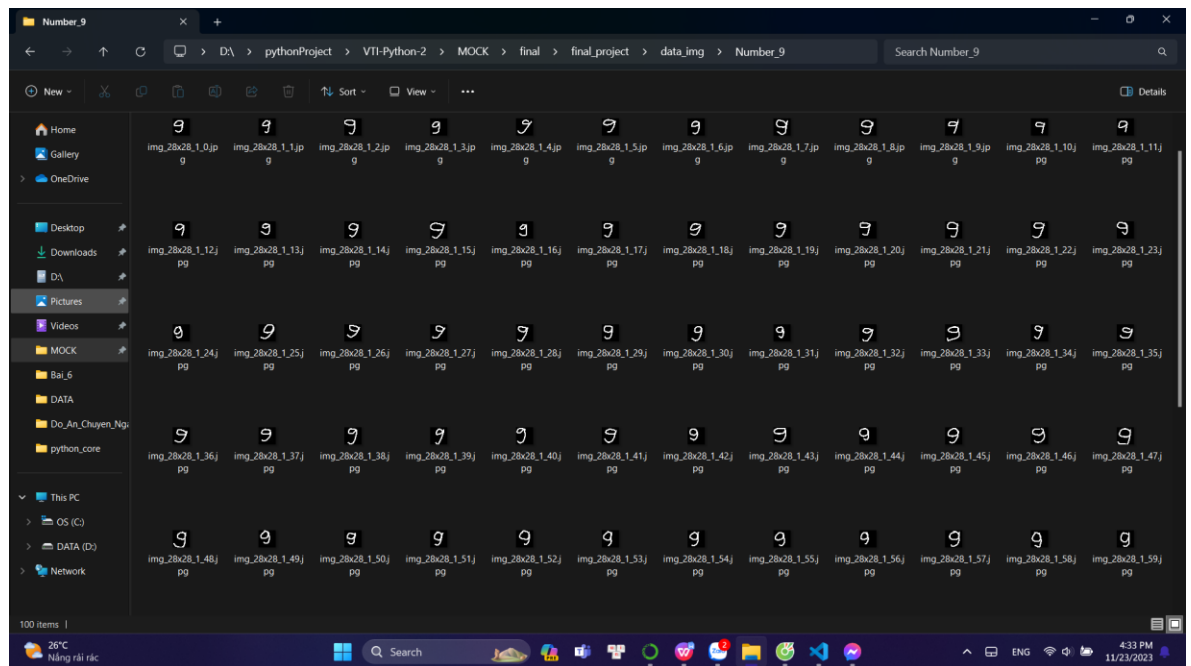
Số 7:



Số 8:



Số 9:



2.1.2 Quá trình xử lý dữ liệu

Đầu tiên, em sẽ load data từ tập mnist, lúc này data sẽ bị chia vào 4 tập là X_train, X_test, y_train, y_test. Em sẽ ghép tập X_train và X_test thành tập X, ghép tập y_train và y_test thành tập y. Ghép bằng concatenate.

```
from keras.datasets import mnist
import numpy as np

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Concatenate X_train and X_test along the first axis (axis=0)
X_1 = np.concatenate((X_train, X_test), axis=0)

# Concatenate y_train and y_test
y_1 = np.concatenate((y_train, y_test), axis=0)

# Print the shapes of the resulting X and y
print("Shape of X:", X_1.shape)
print("Shape of y:", y_1.shape)
```

Shape of X: (70000, 28, 28)
Shape of y: (70000,)

Bây giờ, ở tập dữ liệu mà em tự tạo trong các folder, em sẽ dùng vòng lặp để duyệt từng ảnh (có đuôi .jpg) ở trong các thư mục con. Thư mục tổng là data_img, bên trong thư mục tổng này là các thư mục con tên Number_0, Number_1, Number_9, trong mỗi thư mục Number lại chứa 100 ảnh mẫu về số đó. Vì vậy ta dùng vòng for để duyệt từng thư mục Number này, mỗi khi duyệt 1 folder, ta đánh nhãn tương ứng với thư mục đó, sau đó ta duyệt từng ảnh trong mỗi thư mục Number để lấy dữ liệu ảnh.

```

import cv2
import os
import numpy as np

main_folder_path = 'data_img/'

X = [] # List để lưu ảnh
y = [] # List để lưu nhãn

label_dict = {} # Từ điển để ánh xạ tên nhãn thành giá trị nguyên

# Lặp qua tất cả các thư mục con trong thư mục chính
for label_idx, subfolder in enumerate(os.listdir(main_folder_path)):
    subfolder_path = os.path.join(main_folder_path, subfolder)

    # Kiểm tra xem mục trong thư mục chính có phải là thư mục không
    if os.path.isdir(subfolder_path):
        # Ánh xạ tên nhãn thành một giá trị số nguyên
        label_dict[subfolder] = label_idx

        # Lặp lại tất cả các tệp trong thư mục con
        for filename in os.listdir(subfolder_path):
            if filename.endswith('.jpg'):
                # Xây dựng đường dẫn tập tin đầy đủ
                file_path = os.path.join(subfolder_path, filename)

                # Đọc hình ảnh bằng OpenCV
                img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)

                # Thay đổi kích thước hình ảnh nếu cần
                img = cv2.resize(img, (28, 28))

                # Nối hình ảnh vào X và nhãn tương ứng của nó vào y
                X.append(img)
                y.append(label_idx)

```

```

# Chuyển đổi danh sách thành mảng có nhiều mảng
X = np.array(X)
y = np.array(y)

```

```

# In ánh xạ tên nhãn thành giá trị nguyên
print("Label Dictionary:", label_dict)

```

```

# In các hình X và y
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)

```

```

Label Dictionary: {'Number_0': 0, 'Number_1': 1, 'Number_2': 2, 'Number_3': 3, 'Number_4': 4, 'Number_5': 5, 'Number_6': 6, 'Number_7': 7, 'Number_8': 8, 'Number_9': 9}
Shape of X: (1000, 28, 28)
Shape of y: (1000,)

```



```
x = np.concatenate((x_1,x), axis=0)
y = np.concatenate((y_1,y), axis=0)
```

Sau đó, chúng ta tiếp tục nối dữ liệu tự thu thập (mảng numpy 3 chiều) vào mảng dữ liệu vốn có mnist (mảng numpy 3 chiều).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

Sau đó, em chia dữ liệu theo train_test_split theo tỷ lệ 80-20, random state 42.

Tiếp theo là tiền xử lý dữ liệu

```
# Một lần nữa, thực hiện một số định dạng
# Ngoài trừ việc chúng tôi không làm phẳng mỗi hình ảnh thành một vector có độ dài 784 vì trước tiên chúng tôi muốn thực hiện các phép biến đổi

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1) #thêm một thứ nguyên bổ sung để thể hiện kênh đơn
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

X_train = X_train.astype('float32') # thay đổi số nguyên thành số dấu phẩy động 32 bit
X_test = X_test.astype('float32')

X_train /= 255 # chuẩn hóa từng giá trị cho từng pixel cho toàn bộ vector cho mỗi đầu vào
X_test /= 255

print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape)
```

```
# one-hot format classes

nb_classes = 10 # number of unique digits

Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

Xây dựng mô hình

```

model = Sequential()                                # Linear stacking of layers

# Convolution Layer 1
model.add(Conv2D(32, (3, 3), input_shape=(28,28,1))) # 32 different 3x3 kernels -- so 32 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
convLayer01 = Activation('relu')                    # activation
model.add(convLayer01)

# Convolution Layer 2
model.add(Conv2D(32, (3, 3)))                        # 32 different 3x3 kernels -- so 32 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
model.add(Activation('relu'))                        # activation
convLayer02 = MaxPooling2D(pool_size=(2,2))          # Pool the max values over a 2x2 kernel
model.add(convLayer02)

# Convolution Layer 3
model.add(Conv2D(64, (3, 3)))                        # 64 different 3x3 kernels -- so 64 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
convLayer03 = Activation('relu')                    # activation
model.add(convLayer03)

# Convolution Layer 4
model.add(Conv2D(64, (3, 3)))                        # 64 different 3x3 kernels -- so 64 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
model.add(Activation('relu'))                        # activation
convLayer04 = MaxPooling2D(pool_size=(2,2))          # Pool the max values over a 2x2 kernel
model.add(convLayer04)
model.add(Flatten())                                # Flatten final 4x4x64 output matrix into a 1024-length vector

```

```

# Fully Connected Layer 5
model.add(Dense(512))                                # 512 FCN nodes
model.add(BatchNormalization())                     # normalization
model.add(Activation('relu'))                       # activation

# Fully Connected Layer 6
model.add(Dropout(0.2))                             # 20% dropout of randomly selected nodes
model.add(Dense(10))                                # final 10 FCN nodes
model.add(Activation('softmax'))                    # softmax activation

```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|--------------------|---------|
| conv2d (Conv2D) | (None, 26, 26, 32) | 320 |
| batch_normalization (Batch Normalization) | (None, 26, 26, 32) | 128 |
| activation (Activation) | (None, 26, 26, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 24, 24, 32) | 9248 |
| batch_normalization_1 (Batch Normalization) | (None, 24, 24, 32) | 128 |
| activation_1 (Activation) | (None, 24, 24, 32) | 0 |
| max_pooling2d (MaxPooling2D) | (None, 12, 12, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 10, 10, 64) | 18496 |
| batch_normalization_2 (Batch Normalization) | (None, 10, 10, 64) | 256 |
| ... | | |
| Total params: 597738 (2.28 MB) | | |
| Trainable params: 596330 (2.27 MB) | | |
| Non-trainable params: 1408 (5.50 KB) | | |

Convolution Layer 1: Sử dụng Conv2D với 32 bộ lọc (kernels) có kích thước 3x3, và hàm kích hoạt là ReLU (Rectified Linear Unit). BatchNormalization được áp dụng để chuẩn hóa giá trị trước khi áp dụng hàm kích hoạt. Cuối cùng, áp dụng hàm kích hoạt ReLU.

Convolution Layer 2: Sử dụng thêm một lớp Conv2D với 32 bộ lọc và kích thước 3x3. BatchNormalization và hàm kích hoạt ReLU được áp dụng. Cuối cùng, sử dụng MaxPooling2D để giảm kích thước của ma trận đặc trưng bằng cách lấy giá trị lớn nhất từ mỗi vùng 2x2.

Convolution Layer 3: Sử dụng lớp Conv2D mới với 64 bộ lọc và kích thước 3x3. BatchNormalization và hàm kích hoạt ReLU được áp dụng.

Convolution Layer 4: Sử dụng thêm một lớp Conv2D với 64 bộ lọc và kích thước 3x3. BatchNormalization và hàm kích hoạt ReLU được áp dụng. Sử dụng MaxPooling2D để giảm kích thước. Flatten Layer: Chuyển ma trận 4x4x64 thành một vector có độ dài 1024.

Fully Connected Layer 5: Sử dụng lớp Dense với 512 nơ-ron và hàm kích hoạt ReLU. BatchNormalization được áp dụng.

Fully Connected Layer 6: Sử dụng Dropout với tỷ lệ 20% để ngẫu nhiên tắt 20% các nơ-ron để tránh overfitting. Lớp Dense cuối cùng với 10 nơ-ron (vì MNIST có 10 lớp dự đoán từ 0 đến 9). Hàm kích hoạt softmax được sử dụng để chuyển đổi giá trị thành xác suất.

Đây là mô hình mạng nơ-ron với 7 layer (bao gồm cả layer Flatten)

Sau đó, ta thiết lập quá trình huấn luyện mô hình

```

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# tăng cường dữ liệu ngăn ngừa việc trang bị quá mức bằng cách thay đổi một chút dữ liệu một cách ngẫu nhiên
# Keras có một tính năng tích hợp tuyệt vời để thực hiện tăng cường tự động

gen = ImageDataGenerator(rotation_range=8, width_shift_range=0.08, shear_range=0.3,
                        height_shift_range=0.08, zoom_range=0.08)

test_gen = ImageDataGenerator()

# Sau đó, chúng tôi có thể cung cấp dữ liệu tăng cường của mình theo đợt
# Bên cạnh việc xem xét chức năng mất mát như trước, phương pháp này thực sự giúp tiết kiệm bộ nhớ đáng kể
# bởi vì chúng tôi thực sự đang TẢI dữ liệu vào mạng theo từng đợt trước khi xử lý từng đợt.

# Trước khi tất cả dữ liệu được tải vào bộ nhớ, nhưng sau đó được xử lý theo đợt.

train_generator = gen.flow(X_train, Y_train, batch_size=128)
test_generator = test_gen.flow(X_test, Y_test, batch_size=128)

# Bây giờ chúng ta có thể huấn luyện mô hình được cung cấp dữ liệu bởi trình tải hàng loạt của chúng ta
# Các bước trên mỗi kỷ nguyên phải luôn bằng tổng kích thước của tập hợp chia cho kích thước lô

# TIẾT KIỆM BỘ NHỚ ĐÁNG KỂ (quan trọng đối với các mạng lớn hơn, sâu hơn)

model.fit_generator(train_generator, steps_per_epoch=X_train.shape[0]//128, epochs=5, verbose=1,
                    validation_data=test_generator, validation_steps=X_test.shape[0]//128)

```

Mã trên mô tả quá trình huấn luyện mô hình sử dụng dữ liệu được tạo ra động (data augmentation) và sử dụng các generator để cung cấp dữ liệu theo các batch.

Dưới đây là giải thích từng phần của đoạn mã:

model.compile: Đây là bước kết hợp mô hình với các thông số cần thiết để bắt đầu quá trình huấn luyện. Cụ thể, các tham số sau đây được đặt trong hàm này:

loss='categorical_crossentropy': Hàm mất mát được sử dụng trong quá trình huấn luyện. Trong trường hợp này, nó là categorical cross-entropy, phù

hợp với bài toán phân loại nhiều lớp (có thể thấy từ việc sử dụng softmax activation ở lớp cuối cùng của mô hình).

`optimizer='adam'`: Thuật toán tối ưu hóa được sử dụng để điều chỉnh các trọng số của mô hình. Ở đây, "adam" là một trong những thuật toán tối ưu phổ biến cho học máy sâu.

`metrics=['accuracy']`: Đây là các chỉ số để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Trong trường hợp này, chỉ số là "accuracy", tỷ lệ dự đoán đúng trên tổng số mẫu.

Bằng cách này, mô hình đã được chuẩn bị để bắt đầu quá trình huấn luyện sử dụng thuật toán tối ưu adam, với hàm mất mát là categorical cross-entropy và theo dõi chỉ số accuracy để đánh giá hiệu suất.

`ImageDataGenerator`: `gen` và `test_gen` là các đối tượng của lớp `ImageDataGenerator`. Đối với `gen`, nó được sử dụng để tạo dữ liệu huấn luyện với các biến đổi như xoay, dịch, căn chỉnh và thu phóng. `test_gen` được sử dụng để tạo dữ liệu kiểm thử mà không áp dụng các biến đổi này.

Data Flow using Generators: `train_generator` và `test_generator` sử dụng các generator để tạo ra các batch dữ liệu từ `X_train` và `X_test` cũng như nhãn tương ứng `Y_train` và `Y_test`. Batch size được thiết lập là 128.

Model Training:

`model.fit_generator`: Phương thức này được sử dụng để huấn luyện mô hình trên dữ liệu được tạo ra động.

steps_per_epoch được tính dựa trên kích thước của dữ liệu huấn luyện và batch size.

epochs=5: Mô hình sẽ được huấn luyện qua toàn bộ dữ liệu huấn luyện 5 lần.

verbose=1: Hiển thị tiến trình huấn luyện.

validation_data và validation_steps sử dụng dữ liệu kiểm thử để đánh giá hiệu suất của mô hình sau mỗi epoch.

Bằng cách này, mô hình được huấn luyện trên dữ liệu được tạo ra động, giúp tăng cường khả năng tổng quát hóa và chống overfitting. Quá trình huấn luyện sử dụng thuật toán tối ưu adam, hàm mất mát là categorical cross-entropy, và theo dõi chỉ số accuracy để đánh giá hiệu suất.

Kết quả mô hình :

```
Epoch 1/5
C:\Users\duc\AppData\Local\Temp\ipykernel_17136\3086208346.py:6: UserWarning: `Model.fit_generator` is deprecated and will be removed in
model.fit_generator(train_generator, steps_per_epoch=X_train.shape[0]//128, epochs=5, verbose=1,
443/443 [=====] - 80s 176ms/step - loss: 0.1432 - accuracy: 0.9569 - val_loss: 0.0697 - val_accuracy: 0.9808
Epoch 2/5
443/443 [=====] - 79s 178ms/step - loss: 0.0566 - accuracy: 0.9832 - val_loss: 0.0399 - val_accuracy: 0.9864
Epoch 3/5
443/443 [=====] - 78s 177ms/step - loss: 0.0421 - accuracy: 0.9870 - val_loss: 0.0307 - val_accuracy: 0.9913
Epoch 4/5
443/443 [=====] - 109s 245ms/step - loss: 0.0386 - accuracy: 0.9875 - val_loss: 0.0382 - val_accuracy: 0.9883
Epoch 5/5
443/443 [=====] - 109s 245ms/step - loss: 0.0337 - accuracy: 0.9894 - val_loss: 0.0299 - val_accuracy: 0.9902
```

Sau đó in ra giá trị mất mát và độ chính xác của mô hình

```
score = model.evaluate(X_test, Y_test)
print('Test score:', score[0])
print('Test accuracy:', score[1])

444/444 [=====] - 7s 16ms/step - loss: 0.0297 - accuracy: 0.9903
Test score: 0.029714003205299377
Test accuracy: 0.9902817010879517
```

Lưu mô hình

```
model.save('TA.model.h5')

c:\Users\duc\.conda\envs\anhkun\lib
saving_api.save_model(
```

2.2 Trình bày kết quả của chương trình

Bên cạnh mô hình đã được xây dựng, chúng ta sẽ xây dựng chương trình giao diện dựa trên flask

```
1  import os
2  import base64
3  import numpy as np
4  import cv2
5  from flask import Flask, render_template, request, jsonify
6
7  # Load the pre-trained MNIST model
8  from keras.models import load_model
9
10 # load mode mnist_model.h5
11 model = load_model('TA.model.h5')
12
13 app = Flask(__name__)
14
15 # Counter for naming images
16 image_counter = 0
17
18 @app.route('/')
19 def index():
20     return render_template('index_v5.html')
21
22 @app.route('/predict_digit', methods=['POST'])
23 def predict_digit():
24     global image_counter
25     data_url = request.form['data_url'] # Dữ liệu vẽ từ Canvas dưới dạng URL
26     image_data = data_url.split(',')[1] # Lấy dữ liệu hình ảnh từ URL
27     binary_data = base64.b64decode(image_data) # Chuyển đổi dữ liệu thành dạng nhị phân
28
29     # Lưu hình ảnh vào tệp ảnh tạm thời
30     temp_image_path = 'img_400x400.png'
31     with open(temp_image_path, 'wb') as f:
32         f.write(binary_data)
33
34
```



```

34
35     # Đọc và xử lý hình ảnh sử dụng OpenCV
36     img = cv2.imread(temp_image_path, cv2.IMREAD_GRAYSCALE)
37
38     # resize ảnh --> 28x28
39     img = cv2.resize(img, (28, 28))
40
41     # lưu ảnh để kiểm tra
42     # cv2.imwrite(f'data_img/Number_9/img_28x28_1_{image_counter}.jpg', img)
43
44     # reshape ảnh
45     img = img.reshape(1, 28, 28, 1)
46
47     # Normalize ảnh
48     img = img.astype('float32') / 255.0
49
50
51     # Dự đoán số
52     prediction = model.predict(img)
53     predicted_digit = np.argmax(prediction)
54
55     # Xóa tệp ảnh tạm thời
56     os.remove(temp_image_path)
57
58     image_counter += 1
59     # Hiển thị
60     return jsonify({"predicted_digit": str(predicted_digit)})
61
62 if __name__ == '__main__':
63     app.run(debug=True)
64

```

Code bên template:

```

<!DOCTYPE html>
<html>
<head>
  <title>Draw and Predict</title>
</head>
<body>
  <h1>Draw and Predict</h1>
  <canvas id="canvas" width="400" height="400" style="border: 1px solid black;"></canvas>
  <button onclick="clearCanvas()">Clear</button>
  <button onclick="predictDigit()">Predict</button>
  <label for="lineWidth">Line Width:</label>
  <input type="range" id="lineWidth" min="20" max="30" value="30" step="1" oninput="updateLineWidth(this.value); validateLineWidth()">
  <span id="lineWidthValue">30</span>
  <br>
  <label for="lineColor">Line Color:</label>
  <input type="color" id="lineColor" value="#000000" onchange="updateLineColor(this.value)">
  <br>
  <label for="backgroundColor">Background Color:</label>
  <input type="color" id="backgroundColor" value="#ffffff" onchange="updateBackgroundColor(this.value)">

  <script>
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    var drawing = false;
    var lineWidth = 30; // Độ dày của đường vẽ mặc định (ban đầu là 20)
    var lineColor = 'white'; // Màu đường vẽ
    var backgroundColor = 'black'; // Màu nền trắng

    function updateLineWidth(value) {
      lineWidth = parseInt(value);
      document.getElementById('lineWidthValue').textContent = lineWidth;
    }

    function updateLineColor(value) {

```

```

      lineColor = value;
    }

    function updateBackgroundColor(value) {
      backgroundColor = value;
      context.fillStyle = backgroundColor;
      context.fillRect(0, 0, canvas.width, canvas.height);
    }

    // Thiết lập màu nền trắng ban đầu
    context.fillStyle = backgroundColor;
    context.fillRect(0, 0, canvas.width, canvas.height);

    canvas.addEventListener('mousedown', function(e) {
      drawing = true;
      context.beginPath();
      context.moveTo(e.clientX - canvas.getBoundingClientRect().left, e.clientY - canvas.getBoundingClientRect().top);
    });

    canvas.addEventListener('mousemove', function(e) {
      if (drawing) {
        context.lineTo(e.clientX - canvas.getBoundingClientRect().left, e.clientY - canvas.getBoundingClientRect().top);
        context.lineWidth = lineWidth;
        context.strokeStyle = lineColor; // Đặt màu cho đường vẽ
        context.stroke();
      }
    });

    canvas.addEventListener('mouseup', function() {
      drawing = false;
    });

```

```

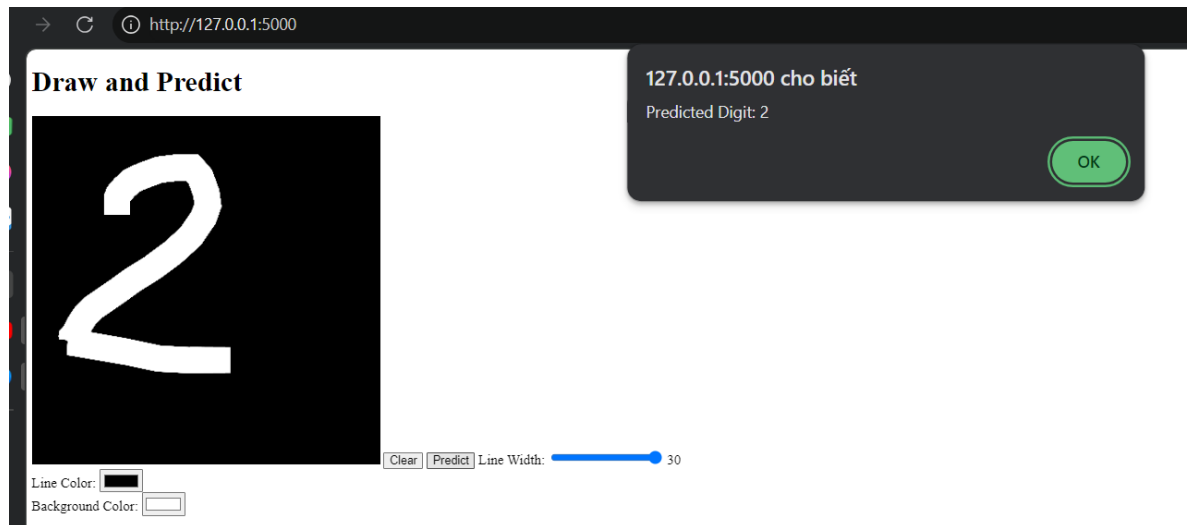
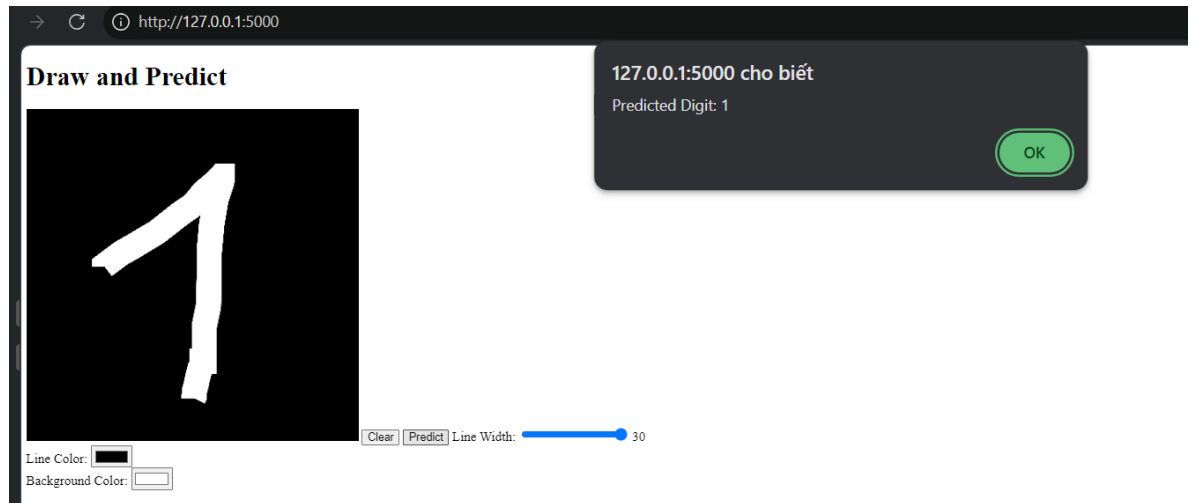
function clearCanvas() {
    context.fillStyle = backgroundColor;
    context.fillRect(0, 0, canvas.width, canvas.height);
}

function predictDigit() {
    var dataURL = canvas.toDataURL();
    fetch('/predict_digit', {
        method: 'POST',
        body: new URLSearchParams({ data_url: dataURL }),
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded'
        }
    })
    .then(response => response.json())
    .then(data => {
        var predictedDigit = data.predicted_digit;
        alert("Predicted Digit: " + predictedDigit);
    })
    .catch(error => console.error('Lỗi:', error));
}

// Hàm để đảm bảo độ dày nét vẽ nằm trong khoảng từ 20 đến 30
function validateLineWidth() {
    if (lineWidth < 20) {
        lineWidth = 20;
    } else if (lineWidth > 30) {
        lineWidth = 30;
    }
    document.getElementById('lineWidth').value = lineWidth;
    document.getElementById('lineWidthValue').textContent = lineWidth;
}
</script>
</body>
</html>

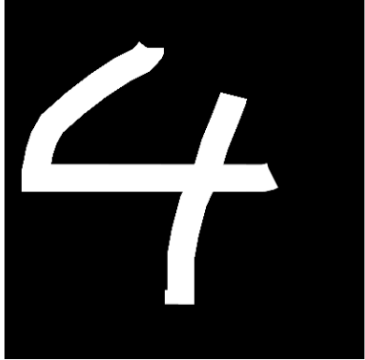
```

Kết quả của chương trình:



http://127.0.0.1:5000

Draw and Predict



127.0.0.1:5000 cho biết
Predicted Digit: 4


OK

Clear Predict Line Width: 30

Line Color: ☐ Background Color: ☐

http://127.0.0.1:5000

Draw and Predict



127.0.0.1:5000 cho biết
Predicted Digit: 5

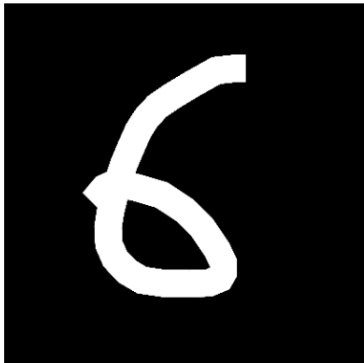
OK

Clear Predict Line Width: 30

Line Color: ☐ Background Color: ☐

http://127.0.0.1:5000

Draw and Predict



127.0.0.1:5000 cho biết
Predicted Digit: 6

OK

Clear Predict Line Width: 30

Line Color: ☐ Background Color: ☐

Draw and Predict



Line Color:

Background Color:

Clear

Predict


Line Width:

127.0.0.1:5000 cho biết

Predicted Digit: 7

OK

Draw and Predict



Line Color:

Background Color:

Clear

Predict


Line Width:

127.0.0.1:5000 cho biết

Predicted Digit: 8

OK

Draw and Predict



Line Color:

Background Color:

Clear

Predict

Line Width:

127.0.0.1:5000 cho biết

Predicted Digit: 9

OK

→ <http://127.0.0.1:5000>

Draw and Predict



127.0.0.1:5000 cho biết
Predicted Digit: 0

OK

Line Color:

Background Color:

Clear Predict Line Width: 30

CHƯƠNG 3 Đánh giá và kết luận.

Dựa vào mô hình trên, em thấy khả năng dự đoán rất chính xác, hi hữu mới có 1 vài sai sót, nhưng không đáng kể. Đây là 1 bài toán rất hay, sử dụng rất nhiều kiến thức trước giờ em được học để phát triển.

Thông qua quá trình làm bài, em đã làm được các nhiệm vụ:

- Thu thập được thêm dữ liệu giúp mô hình học tốt hơn
- Đã huấn luyện được mô hình với độ chính xác cao
- Đã lưu được mô hình dưới dạng file .h5
- Đã tạo được sản phẩm demo hoàn chỉnh
- Đã viết xong báo cáo
- Demo sản phẩm khá chính xác.

Bên cạnh đó, em vẫn chưa thể nâng cấp được giao diện cho đẹp mắt hơn, chỉ mới dừng lại ở mức cơ bản.