TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

======***======

BÁO CÁO BTL THUỘC HỌC PHẦN:

TRÍ TUỆ NHÂN TẠO

NHẬN DIỆN CHỮ SỐ VIẾT TAY ÁP DỤNG MÔ HÌNH CNN

GVHD:	ThS Trần Thanh Huân
Nhóm :	11
Thành viên:	Nguyễn Trúc Linh
	Nghiêm Thị Ngọc Anh
	Nguyễn Thị Bích Ngọc
	Nguyễn Đức Long
Lớp :	20242IT6094001. Khóa 18

Hà nội, Năm 2025

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành và sâu sắc tới thầy Trần Thanh Huân, giảng viên bộ môn “Trí tuệ Nhân tạo". Trong suốt quá trình học tập và tìm hiểu kiến thức, chúng em đã nhận được sự giảng dạy tận tâm, sự hướng dẫn nhiệt tình và sự quan tâm sâu sắc từ thầy. Nhờ đó, chúng em đã tích lũy được thêm nhiều kiến thức bổ ích, đặc biệt là những hiểu biết nền tảng và thực tiễn về lĩnh vực trí tuệ nhân tạo hiện đại.

Từ những kiến thức đã được học và nghiên cứu, chúng em xin trình bày bài tập lớn với đề tài: “Nhận diện chữ số viết tay áp dụng mô hình CNN”. Đây là một hướng tiếp cận hiện đại và hiệu quả trong xử lý ảnh, đặc biệt trong việc nhận dạng chữ số viết tay – một bài toán có tính ứng dụng cao trong thực tế như đọc biểu mẫu, xử lý văn bản viết tay, và tự động hóa nhập liệu.

Tuy nhiên, do thời gian có hạn và trình độ của chúng em còn nhiều hạn chế, bài làm không tránh khỏi những thiếu sót. Kính mong thầy dành thời gian xem xét và góp ý để bài báo cáo được hoàn thiện hơn.

Một lần nữa, chúng em xin kính chúc thầy luôn dồi dào sức khỏe, hạnh phúc và thành công trong sự nghiệp "trồng người" cao quý.

Chúng em xin chân thành cảm ơn!

Mục Lục

Mở Đầu

1. Đặt vấn đề

Trong thời đại công nghệ số phát triển mạnh mẽ, việc tự động hóa các quy trình xử lý dữ liệu là một trong những mục tiêu quan trọng của ngành công nghệ thông tin. Một trong những lĩnh vực được quan tâm nhiều là nhận dạng chữ viết tay, đặc biệt là nhận diện chữ số viết tay – một bài toán thường gặp trong thực tế như xử lý phiếu điểm, biểu mẫu hành chính, tài liệu viết tay,…

Việc xử lý và nhận diện chữ số viết tay một cách tự động không chỉ giúp giảm tải công việc thủ công mà còn nâng cao hiệu suất và độ chính xác trong lưu trữ và phân tích dữ liệu. Tuy nhiên, đây là một bài toán khó do nét chữ tay rất đa dạng và không có quy chuẩn cố định. Do đó, việc áp dụng các kỹ thuật hiện đại như mạng nơ-ron tích chập (CNN) để nhận diện chữ số viết tay là một hướng đi hiệu quả và tiềm năng.

2. Mục đích nghiên cứu

Đề tài "Nhận diện chữ số viết tay áp dụng mô hình CNN" được thực hiện nhằm đạt được các mục tiêu sau:

- Tìm hiểu lý thuyết và cách thức hoạt động của mạng nơ-ron tích chập (CNN).
- Ứng dụng CNN để xây dựng mô hình có khả năng nhận diện chữ số viết tay với độ chính xác cao.
- Đánh giá hiệu quả của mô hình và đề xuất hướng cải tiến nếu có.

3. Phạm vi đề tài

Trong khuôn khổ bài tập lớn môn Trí tuệ Nhân tạo, đề tài tập trung vào:

- Tìm hiểu mô hình CNN cơ bản áp dụng trong nhận dạng ảnh.
- Sử dụng tập dữ liệu MNIST gồm các ảnh chữ số viết tay (từ 0 đến 9) để huấn luyện và kiểm thử mô hình.
- Triển khai mô hình bằng ngôn ngữ Python với thư viện TensorFlow/Keras.

- Đánh giá mô hình dựa trên độ chính xác (accuracy), biểu đồ huấn luyện và ma trận nhầm lẫn (confusion matrix).

4. Phương pháp thực hiện

Để thực hiện đề tài, nhóm chúng em đã tiến hành các bước chính như sau:

- Tìm hiểu lý thuyết về mạng nơ-ron tích chập (CNN), bao gồm cấu trúc tổng quát, các lớp cơ bản như Convolutional, Pooling, Flatten, Dense, cùng với cơ chế huấn luyện và tối ưu mô hình.
- Chuẩn bị dữ liệu bằng cách sử dụng tập dữ liệu MNIST – một tập dữ liệu tiêu chuẩn gồm 60.000 ảnh chữ số viết tay cho huấn luyện và 10.000 ảnh cho kiểm thử.
- Xây dựng mô hình CNN bằng ngôn ngữ lập trình Python, sử dụng thư viện TensorFlow và Keras để thiết kế, huấn luyện và đánh giá mô hình.
- Huấn luyện và đánh giá mô hình dựa trên các chỉ số như độ chính xác (accuracy), độ mất mát (loss), biểu đồ quá trình huấn luyện và ma trận nhầm lẫn (confusion matrix).
- Phân tích kết quả và rút ra nhận xét, đồng thời đề xuất một số hướng cải tiến mô hình trong các nghiên cứu tiếp theo nếu có.

Chương 1: Tổng quan

1.1. Bài toán nhận diện chữ số viết tay

Bài toán nhận diện chữ số viết tay đã được nghiên cứu từ nhiều thập kỷ, đặc biệt nổi bật trong các cuộc thi như MNIST Challenge – nơi nhiều mô hình học sâu được đánh giá và cải tiến.

Nhận diện chữ sốviết tay là một bài toán quan trọng trong lĩnh vực nhận dạng mẫu và trí tuệ nhân tạo, nhằm mục đích tự động phân loại và nhận biết các chữ số được viết bằng tay dưới dạng hình ảnh. Bài toán này đặt ra nhiều thách thức do sự đa dạng về phong cách viết tay, kích thước, độ nghiêng, nét vẽ không đồng nhất, cũng như các yếu tố nhiễu trong quá trình thu thập dữ liệu. Không giống các phương pháp truyền thống đòi hỏi trích xuất đặc trưng thủ công, CNN có khả năng học trực tiếp từ dữ liệu ảnh thô thông qua các lớp lọc, nhờ đó đạt hiệu quả cao trong nhận diện hình ảnh.

Đặc biệt, trong bài toán nhận diện chữ số viết tay, hệ thống cần phân loại chính xác các hình ảnh đầu vào thành 10 lớp ký tự số từ 0 đến 9. Việc xây dựng một mô hình nhận diện hiệu quả sẽ góp phần nâng cao khả năng tự động hóa các ứng dụng như đọc số tự động trên hóa đơn, phiếu khảo sát, hay các hệ thống hỗ trợ nhập liệu bằng tay.

Mạng nơ-ron tích chập (Convolutional Neural Network - CNN) là một trong những kiến trúc mạng học sâu được sử dụng phổ biến và hiệu quả trong lĩnh vực nhận dạng hình ảnh. CNN có khả năng tự động trích xuất các đặc trưng từ dữ liệu đầu vào, giảm thiểu sự phụ thuộc vào các bước tiền xử lý phức tạp và tăng khả năng phân loại chính xác.

1.2. Ứng dụng thực tế

Trong kỷ nguyên chuyển đổi số, nhu cầu tự động hóa nhận dạng chữ viết tay tăng mạnh, đặc biệt trong các hệ thống OCR (Optical Character Recognition) tích hợp trên thiết bị di động, phần mềm quản lý tài liệu, và AI chatbot xử lý biểu mẫu.

Bài toán nhận diện chữ số viết tay có nhiều ứng dụng thiết thực trong đời sống và các ngành công nghiệp khác nhau, bao gồm:

- Hệ thống nhận dạng chữ số trên giấy tờ và biểu mẫu: Giúp tự động hóa việc nhập liệu các thông tin từ phiếu khảo sát, đơn đăng ký, hóa đơn hoặc các tài liệu giấy khác mà không cần thao tác thủ công.
- Xử lý hóa đơn và chứng từ tài chính: Tự động nhận diện số tiền, mã số hóa đơn, số tài khoản trên các chứng từ, giảm thiểu sai sót và tăng tốc độ xử lý.
- Ứng dụng trong ngân hàng và tài chính: Đọc số tài khoản, số thẻ hoặc các dữ liệu số viết tay trên các giao dịch hoặc giấy tờ liên quan.
- Giao tiếp người-máy: Ứng dụng trong các hệ thống nhận dạng chữ viết tay trên thiết bị di động, tablet, giúp người dùng nhập liệu nhanh và tiện lợi hơn.
- Hỗ trợ giáo dục và nghiên cứu: Giúp tự động chấm điểm bài thi trắc nghiệm viết tay hoặc các bài tập số học viết tay.

Dẫn chứng thực tế :

Một số hệ thống thực tế như Google Cloud Vision, Microsoft Azure OCR hay các ứng dụng chấm bài tự động cũng sử dụng công nghệ tương tự để nhận diện ký tự viết tay với độ chính xác cao.

1.3 Phương pháp tiếp cận

Bài toán nhận diện chữ số viết tay là một bài toán kinh điển trong lĩnh vực thị giác máy tính (Computer Vision) và trí tuệ nhân tạo .Mục tiêu của bài toán là xây dựng một hệ thống có thể tự động phân loại hoặc nhận dạng ký tự từ hình ảnh chữ số tay đầu vào. Trong lịch sử phát triển, bài toán này đã được tiếp cận bằng nhiều phương pháp khác nhau. Từ các thuật toán học máy truyền thống đến các mô hình học sâu tiên tiến. Trong khuôn khổ đề tài này, nhóm em tập trung vào mô hình học sâu – cụ thể là Mạng Nơ-ron Tích Chập (Convolutional Neural Network - CNN) – bởi khả năng mạnh mẽ trong xử lý dữ liệu hình ảnh.

1.4. Tệp dữ liệu MNIST

MNIST (Modified National Institute of Standards and Technology) là một trong những bộ dữ liệu cơ bản và phổ biến nhất trong lĩnh vực thị giác máy tính, đặc biệt là các bài toán nhận dạng ảnh và học sâu. Bộ dữ liệu này bao gồm tổng cộng 70.000 ảnh grayscale (đen trắng) có kích thước 28x28 pixel, trong đó có 60.000 ảnh dành cho tập huấn luyện và 10.000 ảnh dành cho tập kiểm tra.60.000 ảnh dùng để huấn luyện

Các ảnh trong MNIST là chữ số từ 0 đến 9 do nhiều người viết tay khác nhau, đã được xử lý để căn giữa ký tự, chuẩn hóa độ sáng và lọc nhiễu. Điều này giúp giảm bớt gánh nặng tiền xử lý cho mô hình, đồng thời đảm bảo tính nhất quán của dữ liệu. Mỗi ảnh chỉ chứa duy nhất một ký tự số và có độ tương phản rõ ràng giữa nền và chữ, rất phù hợp cho việc thử nghiệm và đánh giá hiệu quả ban đầu của các mô hình học sâu.

Do tính đơn giản và tính chuẩn hóa cao, MNIST thường được xem là bài toán "Hello World" trong lĩnh vực học máy. Việc sử dụng MNIST trong giai đoạn đầu giúp kiểm tra xem mô hình CNN được thiết kế có khả năng học và phân biệt các đặc trưng cơ bản của chữ số viết tay hay không.

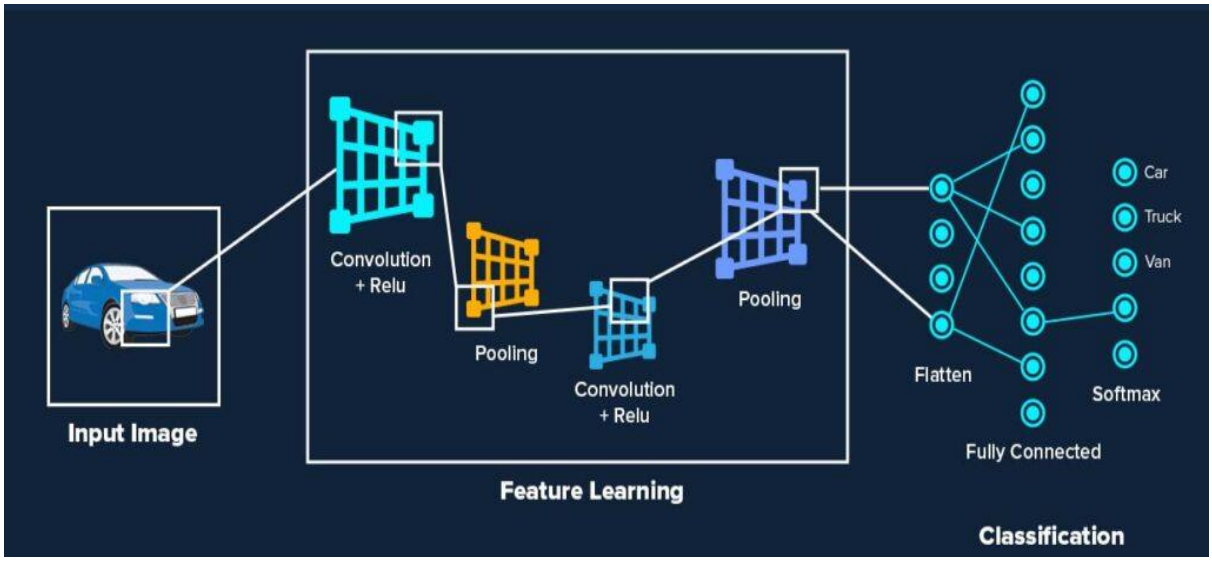
Các ảnh mẫu từ tập thử nghiệm MNIST

Chương 2: Mô hình CNN cho nhận dạng chữ số

2.1. Kiến trúc CNN tiêu chuẩn

Convolutional Neural Networks là gì?

Convolutional Neural Networks (viết tắt là CNN) là một mô hình học sâu (Deep Learning) được thiết kế chuyên biệt để xử lý dữ liệu hình ảnh và thị giác máy tính. CNN hoạt động dựa trên nguyên lý của mạng nơ-ron truyền thống, nhưng điểm khác biệt chính là khả năng tự động trích xuất đặc trưng mà không cần sự can thiệp thủ công từ con người. Nhờ đó, CNN trở thành công cụ có khả năng nhận diện vật thể, phân loại hình ảnh và xử lý video rất hiệu quả, mạnh mẽ.



Hình 2.1:Mạng nơ-ron tích chập

Các lớp cơ bản và nguyên lý hoạt động của Convolutional Neural Networks

Convolutional Neural Networks hoạt động theo nguyên lý trích xuất và phân tích đặc trưng của dữ liệu đầu vào thông qua nhiều lớp xử lý khác nhau. Vậy các lớp cơ bản trong mạng nơ ron tích chập là gì? Và chúng hoạt động như thế nào? Cùng tìm hiểu chi tiết sau đây:

2.1.1 Convolutional layer

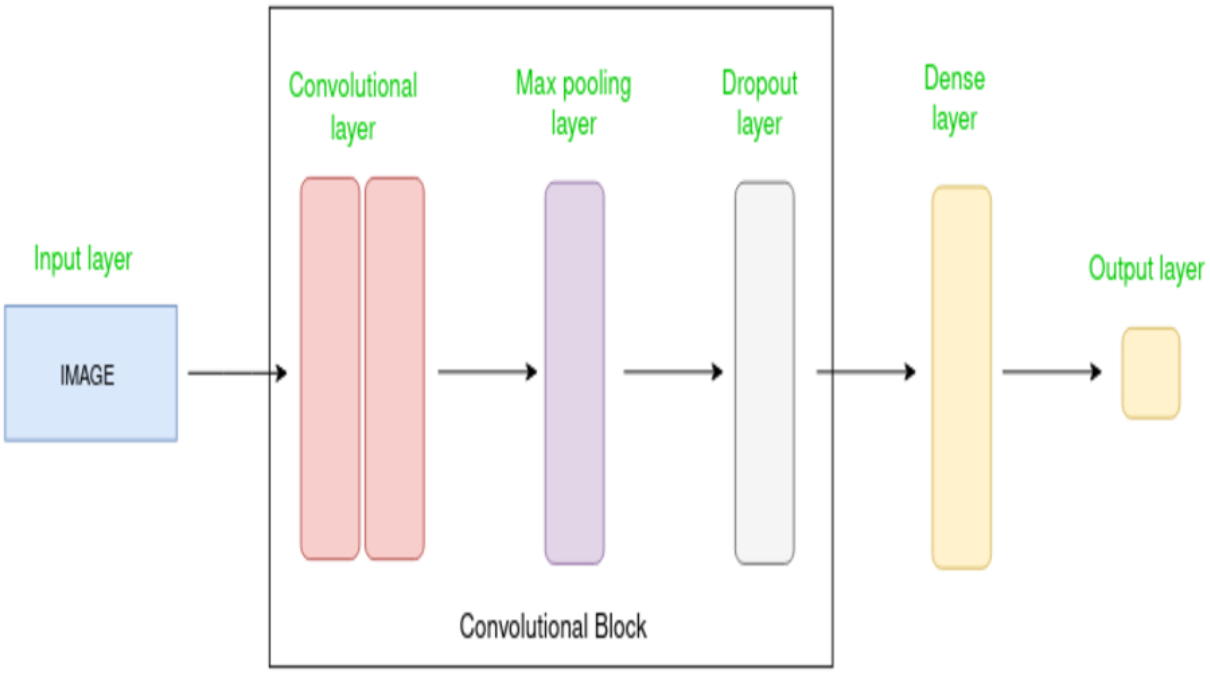
Convolutional layer là lớp tích chập là thành phần quan trọng nhất của CNN, chịu trách nhiệm trích xuất các đặc trưng từ dữ liệu đầu vào. Lớp này sử dụng một bộ lọc (kernel) - một ma trận nhỏ có kích thước phổ biến như 3x3 hoặc 5x5 - quét

qua từng vùng nhỏ của hình ảnh và thực hiện phép nhân tích chập (convolution) giữa các giá trị pixel với trọng số của bộ lọc. Kết quả của quá trình này tạo thành bản đồ đặc trưng (feature map), giúp mô hình phát hiện các đặc điểm như cạnh, góc, màu sắc hoặc kết cấu trong ảnh.

Các tham số quan trọng của lớp tích chập bao gồm: Số lượng bộ lọc, Stride (bước di chuyển của bộ lọc) và Padding (giữ kích thước ảnh). Trong đó:

- Stride xác định khoảng cách di chuyển của kernel trên ảnh đầu vào theo cả chiều ngang (trái sang phải) và chiều dọc (trên xuống dưới).
- Padding là quá trình thêm giá trị vào viền ảnh để kiểm soát kích thước feature map, bảo vệ thông tin viền ảnh khi thực hiện tích chập.

Sau mỗi phép tích chập, Convolutional Neural Networks thường áp dụng hàm kích hoạt ReLU (Rectified Linear Unit) để loại bỏ giá trị âm, tăng tính phi tuyến và giúp mô hình học hiệu quả hơn.



Hình 2.2 : Convolutional layer

2.1.2 Pooling layer

Sau khi trích xuất đặc trưng qua lớp tích chập, Convolutional Neural Networks sử dụng Pooling Layer để giảm kích thước feature map, từ đó giảm số lượng tham

số, tăng hiệu suất tính toán và tránh hiện tượng overfitting (mô hình học quá kỹ vào dữ liệu huấn luyện, nhưng lại hoạt động kém khi gặp dữ liệu mới). Pooling hoạt động bằng cách áp dụng một bộ lọc nhỏ (thường là 2x2 hoặc 3x3) để lấy giá trị đại diện cho mỗi vùng quét, giúp giữ lại những thông tin quan trọng nhất.

Có hai phương pháp pooling phổ biến: Max Pooling và Average Pooling.

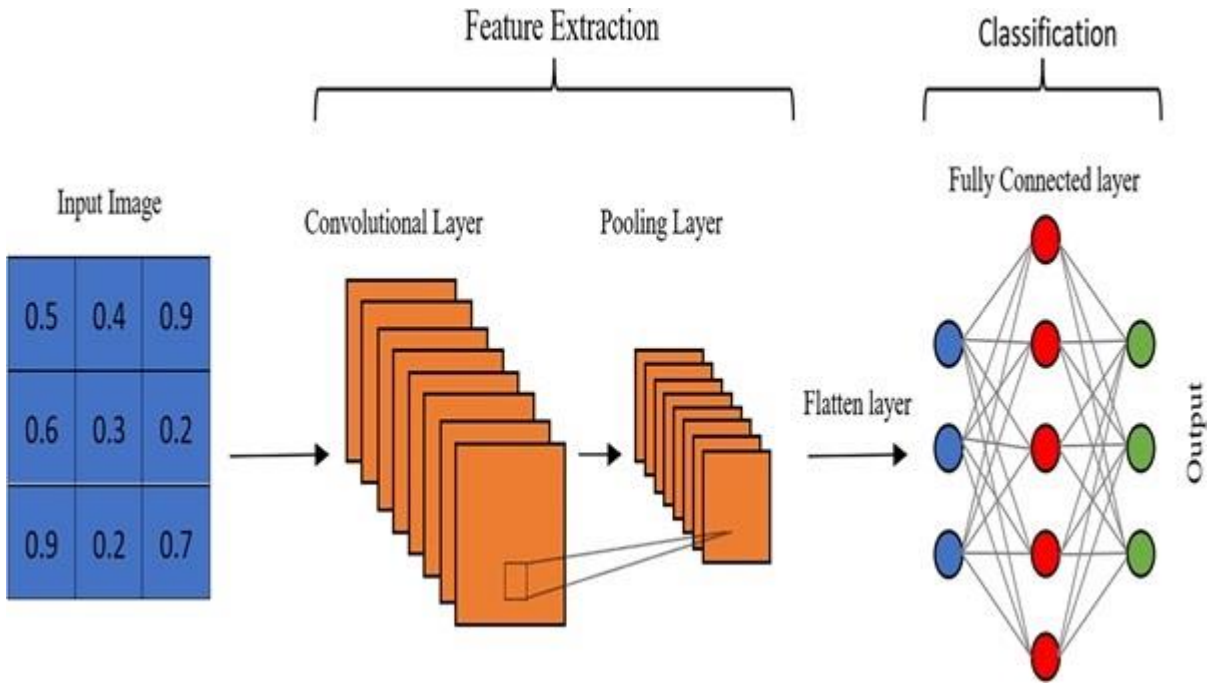
- Trong Max Pooling, giá trị lớn nhất trong vùng quét sẽ được giữ lại, giúp mô hình tập trung vào những đặc trưng nổi bật nhất.
- Average Pooling tính trung bình các giá trị trong vùng quét, giúp tổng hợp thông tin thay vì chỉ giữ giá trị lớn nhất như Max Pooling.

Mặc dù pooling làm mất đi một số thông tin, nhưng đổi lại, nó giúp mô hình hoạt động hiệu quả hơn, giảm thiểu độ phức tạp và cải thiện khả năng tổng quát hóa đối với dữ liệu mới.

2.1.3 Fully connected layer

Fully connected layer là lớp kết nối đầy đủ nằm ở cuối mạng Convolutional Neural Networks, đóng vai trò tổng hợp tất cả các đặc trưng đã trích xuất và thực hiện nhiệm vụ phân loại hình ảnh. Ở lớp này, mỗi nơ-ron được kết nối với toàn bộ nơ-ron ở lớp trước, tạo nên một mạng lưới liên kết chặt chẽ. Các giá trị từ feature map trước đó sẽ được chuyển thành một vector một chiều, một chuỗi dài duy nhất và đưa vào lớp fully connected để xử lý. Quá trình này được gọi là Làm phẳng Flattening.

Tiếp đó, CNN sử dụng các hàm kích hoạt phi tuyến như Softmax hoặc Sigmoid để tính toán xác suất cho từng lớp đầu ra. Điều này giúp cho mô hình đưa ra quyết định cuối cùng, chẳng hạn như phân loại hình ảnh thành các nhóm khác nhau (ví dụ: chó, mèo, ô tô, v.v.).



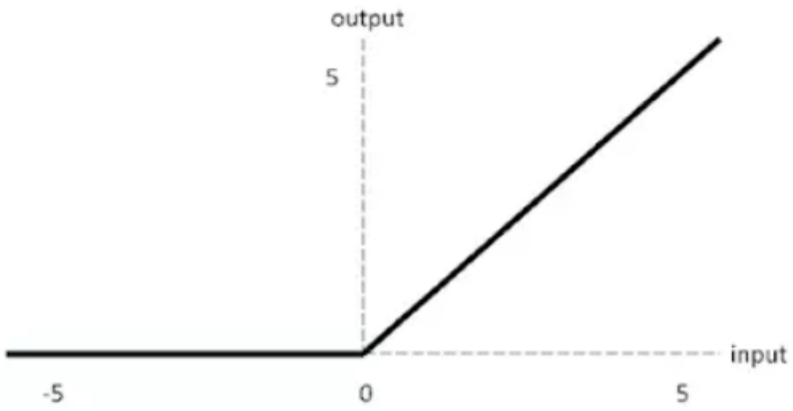
Hình 2.3 : Fully connected layer

2.2. Các hàm kích hoạt (ReLU, Softmax)

- **ReLu**

Đây là một hàm activation rất được ưa chuộng sử dụng. Công thức hàm Relu như sau:

$$f(x) = \max(0, x)$$



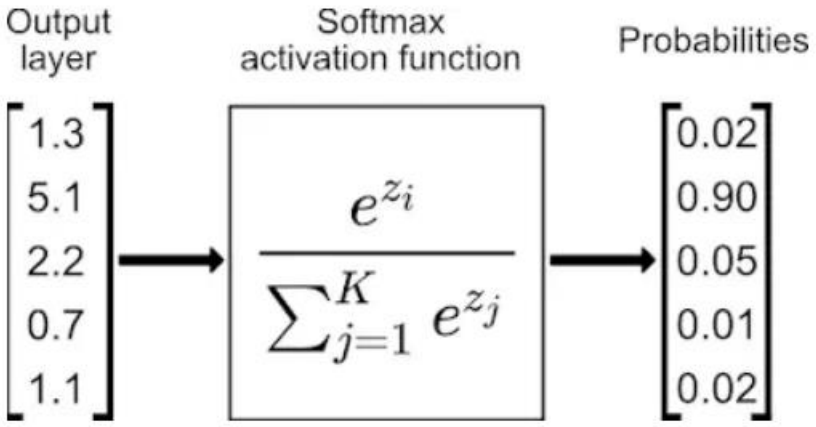
Ưu điểm của hàm Relu là tính đơn giản của nó và nó đã được chứng minh là giúp tăng tốc quá trình training. Tiếp theo là nó không bị chặn như hàm sigmoid hay Tanh nên nó không phải là nguyên nhân gây ra hiện tượng vanishing gradient.

Mặc dù vậy thì tại những điểm có giá trị âm thì giá trị của Relu sẽ bằng 0 (dying relu) và theo lý thuyết nó sẽ không có đạo hàm tại các điểm 0 nhưng thực tế thì người ta thường bổ sung thêm đạo hàm của relu tại 0 bằng 0 và bằng thực nghiệm người ta cũng thấy rằng xác suất để input relu rơi vào điểm 0 là rất nhỏ. Và do nó ko được chặn trên nên cũng có một nhược điểm là gây ra hiện tượng exploding gradient nhưng thường sẽ relu sẽ hoạt động tốt trong thực tế.

- **Softmax**

Đây là một hàm activation thường được sử dụng ở layer cuối cùng của bài classification. Ở đó đầu ra sẽ là xác suất dự đoán rơi vào các class. Công thức hàm softmax như sau:

$$f_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$



2.3 Tiền xử lí ảnh chữ số viết tay

Tiền xử lý hình ảnh là các bước được thực hiện để định dạng hình ảnh trước khi chúng được sử dụng để đào tạo và suy luận mô hình. Điều này bao gồm, nhưng không giới hạn ở, thay đổi kích thước , định hướng và hiệu chỉnh màu sắc .

Tiền xử lý hình ảnh cũng có thể làm giảm thời gian đào tạo mô hình và tăng tốc độ suy luận mô hình. Nếu hình ảnh đầu vào đặc biệt lớn, việc giảm kích thước

của những hình ảnh này sẽ cải thiện đáng kể thời gian đào tạo mô hình mà không làm giảm đáng kể hiệu suất của mô hình. Ví dụ, kích thước chuẩn của hình ảnh trên iPhone 11 là 3024 × 4032. Mô hình máy học mà Apple sử dụng để tạo mặt nạ và áp dụng Chế độ chân dung thực hiện trên hình ảnh có kích thước bằng một nửa kích thước này trước khi đầu ra của nó được thu nhỏ lại về kích thước đầy đủ.

2.3.1 Chuẩn hóa kích thước

Việc chuẩn hóa kích thước ảnh dựa trên việc xác định trọng tâm ảnh, sau đó xác định khoảng cách lớn nhất từ tâm ảnh đến các cạnh trên, dưới, trái, phải của hình chữ nhật bao quanh ảnh. Thông qua khoảng cách lớn nhất đó, có thể xác định được một tỷ lệ co, giãn của ảnh gốc so với kích thước đã xác định, từ đó hiệu chỉnh kích thước ảnh theo tỷ lệ co, giãn này. Như vậy, thuật toán chuẩn hóa kích thước ảnh luôn luôn đảm bảo được tính cân bằng khi co giãn ảnh, ảnh sẽ không bị biến dạng hoặc bị lệch

2.3.2 Lọc nhiễu

Nhiễu là một tập các điểm sáng thừa trên ảnh. Khử nhiễu là một vấn đề thường gặp trong nhận dạng, nhiễu có nhiều loại (nhiễu đốm, nhiễu vệt, nhiễu đứt nét...). Để khử các nhiễu đốm (các nhiễu với kích thước nhỏ), có thể sử dụng các phương pháp lọc (lọc trung bình, lọc trung vị...). Tuy nhiên, với các nhiễu vệt (hoặc các nhiễu có kích thước lớn) thì các phương pháp lọc tỏ ra kém hiệu quả, trong trường hợp này sử dụng phương pháp khử các vùng liên thông nhỏ tỏ ra có hiệu quả hơn.

2.3.3 Tăng cường dữ liệu (Data Augmentation)

Tăng cường dữ liệu **(**Data Augmentation) là một kỹ thuật trong Machine Learning và Deep Learning nhằm tạo ra nhiều mẫu dữ liệu mới từ các mẫu dữ liệu gốc thông qua việc áp dụng các biến đổi như lật, quay, thay đổi độ sáng, hoặc thêm nhiễu. Mục đích chính của Data Augmentation là để tăng cường bộ dữ liệu, giúp cải thiện khả năng tổng quát của mô hình học máy và giảm thiểu hiện tượng overfitting.

Chương 3 :Xây Dựng Chương Trình

3.1. Công cụ sử dụng

3.1.1. Ngôn ngữ Python

Python là một ngôn ngữ lập trình bậc cao, thông dịch, hướng đối tượng và đa mục đích, được phát triển lần đầu vào năm 1991. Với triết lý thiết kế đơn giản, dễ đọc và dễ viết, Python nhanh chóng trở thành một trong những ngôn ngữ phổ biến nhất trong lĩnh vực khoa học dữ liệu và trí tuệ nhân tạo. Khả năng biểu đạt rõ ràng và cú pháp ngắn gọn giúp Python phù hợp không chỉ với người mới bắt đầu mà còn với các nhà nghiên cứu và kỹ sư phần mềm chuyên nghiệp.

Trong bối cảnh bài toán nhận diện ký tự viết tay sử dụng mạng nơ-ron tích chập (Convolutional Neural Network - CNN), Python là lựa chọn phù hợp nhờ những ưu điểm sau:

- Hệ sinh thái thư viện mạnh mẽ: Python cung cấp nhiều thư viện hỗ trợ chuyên sâu cho xử lý dữ liệu và học sâu, bao gồm NumPy, Pandas, OpenCV, TensorFlow, Keras, PyTorch và Matplotlib. Các thư viện này giúp đơn giản hóa quá trình xây dựng mô hình, xử lý ảnh đầu vào, huấn luyện, đánh giá và trực quan hóa kết quả.
- Khả năng mở rộng và tích hợp cao: Python hoạt động tốt trên nhiều nền tảng khác nhau và dễ dàng tích hợp với các công cụ như Jupyter Notebook, Google Colab hay Visual Studio Code. Điều này tạo điều kiện thuận lợi cho việc phát triển, thử nghiệm và trình bày kết quả nghiên cứu.
- Cộng đồng người dùng lớn: Với cộng đồng lập trình viên đông đảo trên toàn cầu, Python sở hữu kho tài nguyên học tập, tài liệu và ví dụ minh họa phong phú. Việc tìm kiếm giải pháp cho các vấn đề kỹ thuật gặp phải trong quá trình phát triển mô hình trở nên dễ dàng và nhanh chóng.
- Hỗ trợ tốt cho mô hình học sâu: Python có cú pháp đơn giản, dễ viết nhưng lại đủ mạnh để mô tả các kiến trúc mạng nơ-ron phức tạp. Điều này đặc biệt

hữu ích khi triển khai các mô hình như CNN, giúp tiết kiệm thời gian lập trình và giảm thiểu lỗi phát sinh.

3.1.2 Môi trường phát triển

Trong quá trình xây dựng và huấn luyện mô hình nhận diện ký tự viết tay sử dụng mạng nơ-ron tích chập (CNN), việc lựa chọn môi trường phát triển phù hợp đóng vai trò quan trọng nhằm đảm bảo hiệu quả xử lý, tiết kiệm thời gian và thuận tiện cho việc theo dõi kết quả. Trong đề tài này, hai môi trường chính được giới thiệu là Google Colab và máy tính cá nhân (laptop).

Google Colab

1. **Google Colab**

Google Colaboratory (Colab) là một nền tảng lập trình trực tuyến do Google cung cấp, hỗ trợ Python và đặc biệt thích hợp cho các tác vụ học máy (machine learning) và học sâu (deep learning). Các đặc điểm nổi bật của Google Colab gồm:

- Hỗ trợ GPU miễn phí: Đây là lợi thế lớn khi huấn luyện mô hình CNN đòi hỏi khả năng tính toán cao. GPU giúp rút ngắn đáng kể thời gian huấn luyện so với CPU thông thường.
- Tích hợp sẵn thư viện: Colab đi kèm với các thư viện cần thiết như TensorFlow, Keras, NumPy, Matplotlib,... giúp người dùng không mất thời gian thiết lập môi trường ban đầu.
- Lưu trữ trên đám mây: Tài liệu và dữ liệu được lưu trên Google Drive, dễ dàng truy cập, chia sẻ và đồng bộ giữa các thành viên trong nhóm phát triển.
- Giao diện Jupyter Notebook: Hỗ trợ trực quan hóa dữ liệu, biểu đồ, mô hình học máy một cách dễ dàng và tương tác từng bước.

2. **Máy tính cá nhân (Laptop)**

Ngoài Google Colab, quá trình viết mã, xử lý dữ liệu đơn giản và thử nghiệm mô hình ở quy mô nhỏ cũng có thể thực hiện trực tiếp trên máy tính cá nhân. Lý do sử dụng laptop gồm:

- Chủ động không cần kết nối Internet: Các thao tác tiền xử lý dữ liệu, viết hàm hoặc kiểm thử nhanh mô hình có thể thực hiện ngoại tuyến, tăng tính linh hoạt.
- Kiểm soát môi trường cài đặt: Người dùng có thể tự cấu hình thư viện, phiên bản Python và môi trường ảo theo nhu cầu riêng.

3.2 Các bước thực hiện và kết quả

3.2.1 Các bước thực hiện

- Thêm các thư viện cần thiết

```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.model_selection import train_test_split

     from tensorflow.keras.datasets import mnist
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
     from tensorflow.keras.utils import to_categorical
```

- Tải dữ liệu MNIST

```
[ ]  # Tải dữ liệu MNIST
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

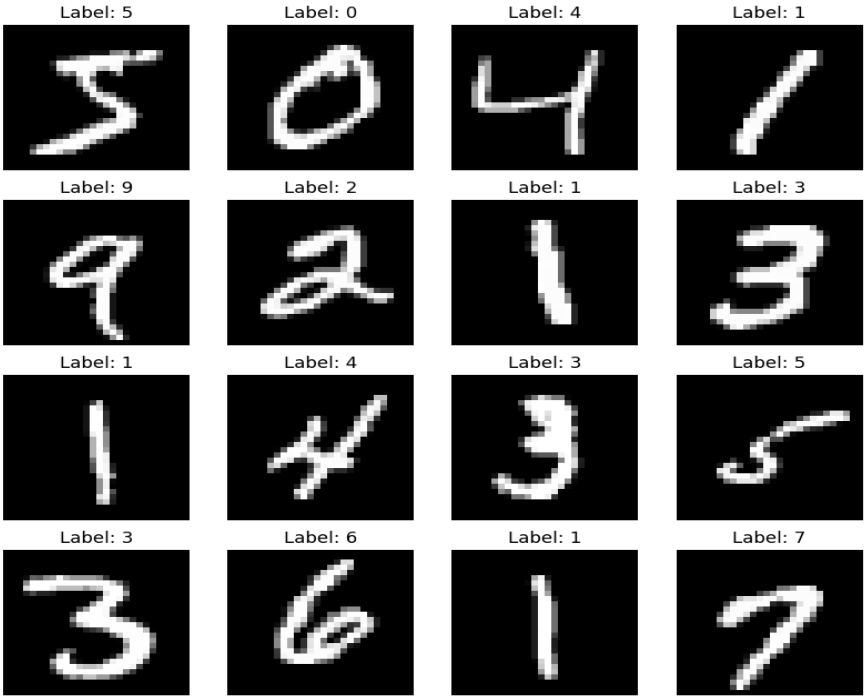
- In kích thước mảng

```
     # In kích thước mảng
     print("Train images:", x_train.shape)
     print("Train labels:", y_train.shape)
     print("Test images:", x_test.shape)
     print("Test labels:", y_test.shape)
```

- In random data trong bộ MNIST để test

```
[ ]  plt.figure(figsize=(10, 10))
     for i in range(16):
         plt.subplot(4, 4, i + 1)
         plt.imshow(x_train[i], cmap='gray')
         plt.title(f"Label: {y_train[i]}")
         plt.axis('off')
     plt.show()
```

- Kết quả



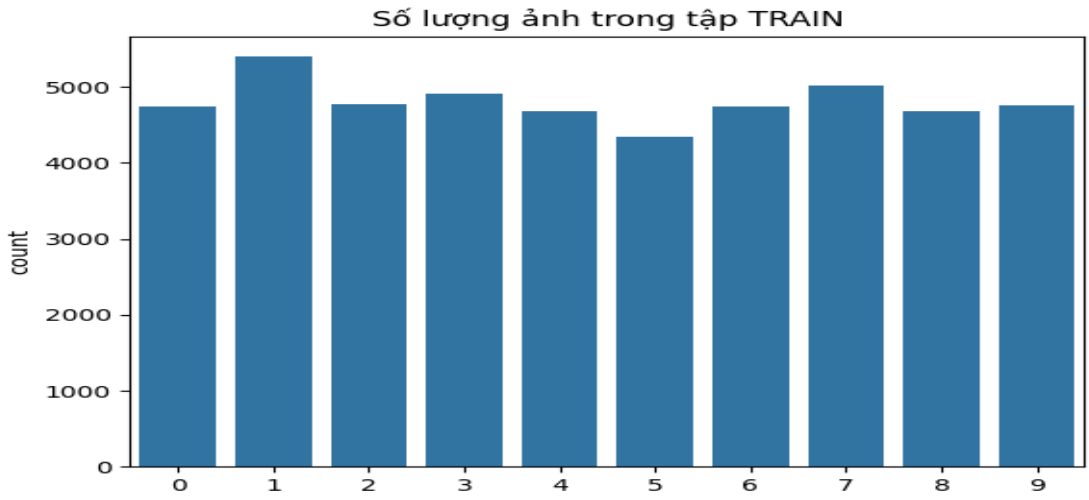
- Chia x_train thành train và validation

```
# Chia x_train thành train và validation
x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train, test_size=0.2, stratify=y_train, random_state=42
)
```

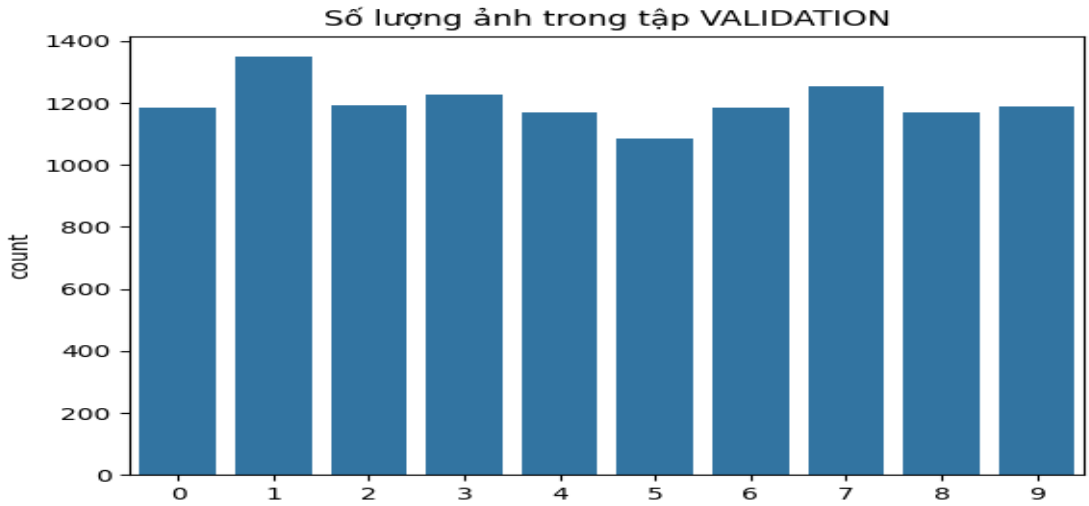
- Sử dụng mảng 1 chiều chứa nhãn số

```
[ ]
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Nếu đang dùng dữ liệu chưa one-hot, bạn có thể vẽ ngay:
    # Sử dụng trực tiếp y_train, y_val, y_test vì chúng là mảng 1 chiều chứa nhãn số
    sns.countplot(x=y_train)
    plt.title("Số lượng ảnh trong tập TRAIN")
    plt.show()

    sns.countplot(x=y_val)
    plt.title("Số lượng ảnh trong tập VALIDATION")
    plt.show()

    sns.countplot(x=y_test)
    plt.title("Số lượng ảnh trong tập TEST")
    plt.show()
```

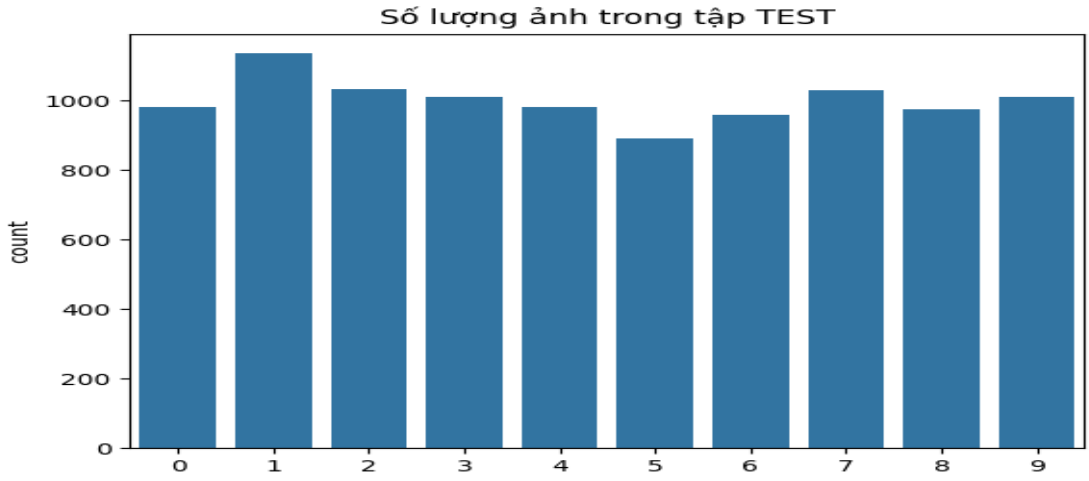
- Đồ thị số lượng các ảnh từ số 0 -› số 9 trong tập train



- Đồ thị số lượng các ảnh từ số 0 -› số 9 trong tập val



- Đồ thị số lượng các ảnh từ số 0 -› số 9 trong tập test



- Chuyển đầu ra thành dạng vector

```python
from tensorflow.keras.utils import to_categorical

# Reshape và chuẩn hóa
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_val   = x_val.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test  = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255

# One-hot encoding
y_train = to_categorical(y_train, 10)
y_val   = to_categorical(y_val, 10)
y_test  = to_categorical(y_test, 10)
```

- Xây dựng model

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

model.summary()
```

- Kết quả

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_12 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_13 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_14 (Conv2D)	(None, 3, 3, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 1, 1, 128)	0
flatten_4 (Flatten)	(None, 128)	0
dense_8 (Dense)	(None, 64)	8,256
dense_9 (Dense)	(None, 10)	650

```
Total params: 101,578 (396.79 KB)
Trainable params: 101,578 (396.79 KB)
Non-trainable params: 0 (0.00 B)
```

- Train model

```
[ ] batch_size = 32
    epochs = 20

    history = model.fit(
        x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(x_val, y_val),
        verbose=1
    )
```

- Kết quả

```
Epoch 1/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 13s 5ms/step - accuracy: 0.8532 - loss: 0.4805 - val_accuracy: 0.9655 - val_loss: 0.1201
Epoch 2/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9751 - loss: 0.0816 - val_accuracy: 0.9807 - val_loss: 0.0647
Epoch 3/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9840 - loss: 0.0543 - val_accuracy: 0.9857 - val_loss: 0.0523
Epoch 4/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9874 - loss: 0.0416 - val_accuracy: 0.9830 - val_loss: 0.0558
Epoch 5/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9893 - loss: 0.0328 - val_accuracy: 0.9852 - val_loss: 0.0487
Epoch 6/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9924 - loss: 0.0225 - val_accuracy: 0.9818 - val_loss: 0.0592
Epoch 7/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9937 - loss: 0.0200 - val_accuracy: 0.9858 - val_loss: 0.0489
Epoch 8/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9942 - loss: 0.0184 - val_accuracy: 0.9835 - val_loss: 0.0617
Epoch 9/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9954 - loss: 0.0135 - val_accuracy: 0.9863 - val_loss: 0.0567
Epoch 10/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9956 - loss: 0.0120 - val_accuracy: 0.9837 - val_loss: 0.0602
Epoch 11/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9967 - loss: 0.0098 - val_accuracy: 0.9872 - val_loss: 0.0470
Epoch 12/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9972 - loss: 0.0091 - val_accuracy: 0.9862 - val_loss: 0.0609
Epoch 13/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9975 - loss: 0.0077 - val_accuracy: 0.9839 - val_loss: 0.0623
Epoch 14/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9965 - loss: 0.0105 - val_accuracy: 0.9873 - val_loss: 0.0578
Epoch 15/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9981 - loss: 0.0058 - val_accuracy: 0.9877 - val_loss: 0.0547
Epoch 16/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9974 - loss: 0.0072 - val_accuracy: 0.9857 - val_loss: 0.0712
Epoch 17/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9979 - loss: 0.0063 - val_accuracy: 0.9856 - val_loss: 0.0647
Epoch 18/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9982 - loss: 0.0057 - val_accuracy: 0.9867 - val_loss: 0.0612
Epoch 19/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9983 - loss: 0.0043 - val_accuracy: 0.9869 - val_loss: 0.0736
Epoch 20/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9982 - loss: 0.0052 - val_accuracy: 0.9850 - val_loss: 0.1043
```

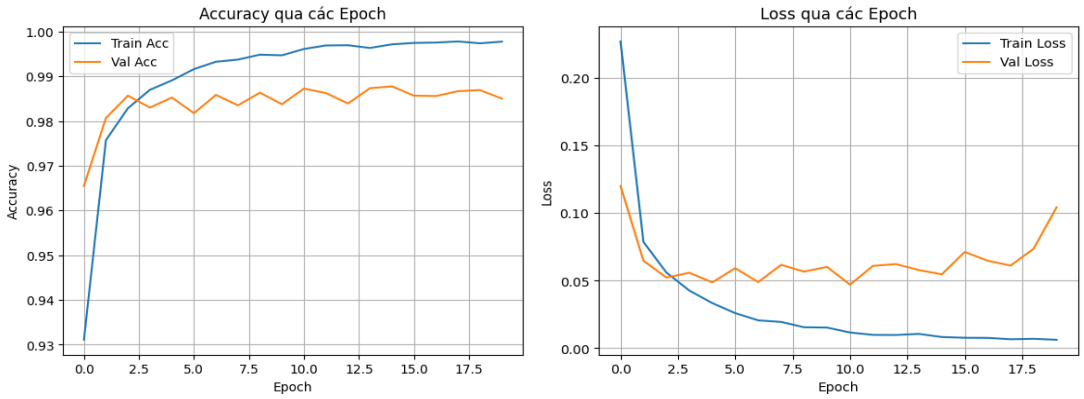
- Tính loss và accuracy của mô hình

```
[ ] loss, acc = model.evaluate(x_test, y_test)
    print(f"Độ chính xác trên tập test: {acc:.4f}")
```

- Kết quả sau khi train model

```python
# Accuracy
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Accuracy qua các Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()

# Loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss qua các Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

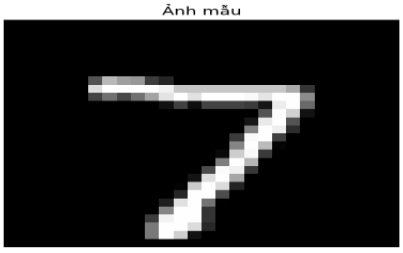
- Ta thu được hiệu suất mô hình



3.2.2 Kết quả

```python
import numpy as np

# Lấy ảnh đầu tiên từ tập test
sample = x_test[0].reshape(1, 28, 28, 1)
pred = model.predict(sample)

print("Dự đoán là:", np.argmax(pred))

# Hiển thị ảnh
plt.imshow(x_test[0].reshape(28, 28), cmap='gray')
plt.title("Ảnh mẫu")
plt.axis('off')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
Dự đoán là: 7
```



Kết luận

Trong nghiên cứu này, nhóm em đã tìm hiểu và ứng dụng mạng nơ-ron tích chập (CNN) trong nhận dạng chữ số viết tay. Quá trình nghiên cứu cho thấy phương pháp này mang lại độ chính xác cao và có tiềm năng lớn trong thực tế. Việc lựa chọn mô hình phù hợp, tối ưu siêu tham số và xử lý dữ liệu hiệu quả đóng vai trò quan trọng trong việc nâng cao hiệu suất nhận dạng.

Một trong những bài học quan trọng nhóm em rút ra là tiền xử lý dữ liệu quyết định đáng kể đến chất lượng mô hình. Hình ảnh đầu vào cần được chuẩn hóa và làm sạch để đảm bảo rằng CNN có thể học được đặc trưng chính xác. Ngoài ra, các kỹ thuật tăng cường dữ liệu giúp mô hình hoạt động ổn định hơn, đặc biệt là khi dữ liệu huấn luyện bị giới hạn.

Bên cạnh đó, việc tối ưu siêu tham số như số lượng lớp mạng, kích thước bộ lọc, hàm kích hoạt và tốc độ học giúp cải thiện hiệu suất của mô hình. Quá trình thử nghiệm với nhiều cấu hình khác nhau đã giúp nhóm em hiểu rõ hơn về cách CNN học và nhận dạng ký tự.

Cuối cùng, thông qua nghiên cứu này, nhóm em nhận thấy CNN không chỉ ứng dụng tốt trong nhận dạng chữ số viết tay mà còn có thể mở rộng sang các lĩnh vực như nhận dạng chữ ký, số hóa tài liệu, và phân loại hình ảnh. Những hiểu biết từ dự án này sẽ là nền tảng quan trọng để phát triển các ứng dụng thông minh hơn trong tương lai.

Tài Liệu Tham Khảo

[1] Ultralytics.com, "Optimization Algorithm – Giải thích các thuật ngữ trong AI," [Online]. Available: https://www.ultralytics.com/vi/glossary/optimization-algorithm.

[2] Statio.vn, "Data Augmentation là gì? Giải thích các thuật ngữ trong Machine Learning, tầm quan trọng và cách thực hiện," [Online]. Available: https://statio.vn/blog/data-augmentation-la-gi-giai-thich-cac-thuat-ngu-trong-machine-learning-tam-quan-trong-va-cach-thuc-hien.

[3] VNPT, "Convolutional Neural Networks là gì," [Online]. Available: https://vnptai.io/vi/blog/detail/convolutional-neural-networks-la-gi.

[4] Viblo, "Tại sao lại sử dụng Activation Function trong Neural Network?," [Online]. Available: https://viblo.asia/p/tai-sao-lai-su-dung-activation-function-trong-neural-network-MG24BwweJz3.