TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN


# ĐỒ ÁN CHUYÊN NGÀNH

# NGÀNH: KHOA HỌC MÁY TÍNH

ĐỀ TÀI

# NHẬN DẠNG CHỮ SỐ VIẾT TAY SỬ DỤNG MẠNG NƠ-RON TÍCH CHẬP

**Giảng viên hướng dẫn : TS. Nguyễn Mạnh Cường**

**Lớp : KHMT01 – K16**

**Sinh Viên thực hiện : Bùi Trường Giang – 2021601506**

**Hà Nội – 2024**

# MỤC LỤC

# LỜI CẢM ƠN

Lời đầu tiên cho phép em gửi lời cảm ơn sâu sắc tới các thầy cô trong khoa Công nghệ thông tin - Trường Đại học Công Nghiệp Hà Nội, những người đã hết mình truyền đạt và chỉ dẫn cho em những kiến thức, những bài học quý báu và bổ ích. Đặc biệt em xin được bày tỏ sự tri ân và xin chân thành cảm ơn giảng viên Nguyễn Mạnh Cường người đã đọc qua bản thảo ban đầu và đưa ra các đề xuất chỉnh sửa. Sau nữa, em xin gửi tình cảm sâu sắc tới gia đình và bạn bè vì đã luôn bên cạnh khuyến khích, động viên, giúp đỡ cả về vật chất lẫn tinh thần cho em trong suốt quy trình học tập để em hoàn thành tốt việc học tập của bản thân.

Trong quá trình nghiên cứu và làm đề tài, do năng lực, kiến thức, trình độ bản thân còn hạn hẹp nên không tránh khỏi những thiếu sót và em mong nhận được sự thông cảm và những góp ý từ quý thầy cô cũng như các bạn trong lớp.

Em xin trân trọng cảm ơn!

**Sinh viên thực hiện**

Bùi Trường Giang

# LỜI NÓI ĐẦU

Trong tất cả các hình thái của thông tin, hình ảnh có lẽ là hình thái gần gũi và dễ tiếp cận với con người. Điều này cũng dễ hiểu khi trải qua bề dày hàng triệu năm tiến hóa, con người đã hình thành được cảm quan trực giác với loại thông tin này trong não bộ, việc hiểu hình ảnh đối với con người không có gì khó khăn. Nhưng với máy tính câu truyện lại khác đi một chút, máy tính chỉ hiểu được những luật tường minh, những câu lệnh cụ thể được lập trình trước. Giới hạn này làm máy tính trở lên khó khăn trong việc thực hiện những điều mà con người vốn có thể làm được một cách dễ dàng, trong số đó bao gồm cả việc hiểu hình ảnh. Thật vậy, cùng là số 1 viết trên giấy nhưng khi nghiêng, xoay, đổi màu mực… con người vẫn nhận ra đó là số 1, nhưng để viết một chương trình làm được điều tương tự thì vô cùng khó vì chúng ta không có cách tường minh diễn giải cho máy tính rằng số 1 thì phải trông như thế nào. Lĩnh vực xử lý ảnh ra đời với mục đích giải quyết vấn đề nhập nhằng trên, một trong những vấn đề đáng quan tâm của xử lý ảnh đó là bài toán nhận diện. Bài toán nhận đầu vào là một ảnh và cho đầu ra là một nhãn tương ứng với ảnh đầu vào. Việc giải được bài toán nhận diện mang đến nhiều cơ hội lớn. Đặc biệt trong kỷ nguyên số hiện nay, khi dữ liệu dạng ảnh có mặt ở hầu hết mọi nơi trên internet việc hiểu được thông tin chứa trong chúng mang lại ích lợi vô cùng lớn.

Vậy nên em quyết định chọn đề tài "**nhận dạng chữ số viết tay sử dụng mạng nơ-ron tích chập**" với mục đích học hỏi và giải quyết bài toán nêu trên. Bài toán nhận đầu vào là một ảnh chữ số viết tay có kích thước 28 x 28 và cho ra đầu ra là một nhãn tương ứng với kết quả nhận diện được là một số từ 0 đến 9.

Báo cáo này trình bày toàn bộ kết quả mà cá nhân em đã đạt được trong suốt quá trình thực hiện đề tài. Báo cáo gồm có 4 chương chính và một mục kết luận

**Chương 1: Tổng quan về đề tài và phát biểu bài toán**

Trong chương 1, báo cáo sẽ trình bày các lý thuyết cơ sở trong lĩnh vực xử lý ảnh và phát biểu bài toán nhận diện chữ số viết tay

**Chương 2: Một số phương pháp hiện có**

Ở chương số 2, báo cáo sẽ trình bày các phương pháp tiếp cận bài toán đã được sử dụng trong quá khứ.

**Chương 3: Thực nghiệm**

Trong chương 3, báo cáo sẽ trình bày kết quả cài đặt cũng như huấn luyện một số kiến trúc CNN đối với bài toán và đưa ra đánh giá khách quan.

**Chương 4: Xây dựng chương trình**

Ở chương 4, báo cáo sẽ trình bày sơ bộ qua chương trình "demo" cho bài toán trên.

**Kết luận**

Cuối cùng phần kết luận sẽ tổng hợp các kết quả đạt được, từ đó đưa ra các hướng phát triển mở rộng đề tài trong tương lai.

Qua quá trình thực hiện đề tài và trải qua nhiều lần thử sai, cá nhân em đã học được nhiều bài học giá trị. Phần lớn trong số chúng đều có ý nghĩa thực tiễn và có thể áp dụng. Đây không chỉ là một nhiệm vụ học thuật đơn thuần mà còn là một trải nghiệm quan trọng giúp em phát triển nhiều kỹ năng cần thiết cho công việc sau này.

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG BIỂU

# CHƯƠNG 1 TỔNG QUAN VỀ ĐỀ TÀI VÀ PHÁT BIỂU BÀI TOÁN

## 1.1 Tổng quan về xử lý ảnh

### 1.1.1 Khái niệm ảnh

Ảnh là thông tin về vật thể hay quang cảnh được chiếu sáng mà con người quan sát và cảm nhận được bằng mắt và hệ thống thần kinh thị giác. Ảnh được quyết định bởi ba yếu tố:

- Vật thể, không gian quan sát được chiếu sáng
- Nguồn sáng
- Cảm nhận (mắt)

Ảnh trong tự nhiên (Ảnh liên tục) là những tín hiệu liên tục về không gian và giá trị độ sáng. Tín hiệu ảnh thuộc loại tín hiệu đa chiều: tọa độ (x, y, z), độ sáng(b), thời gian(t).

Ảnh được lưu trữ trong máy tính như một mảng hai chiều chứa giá trị số. Các số tương ứng với các thông tin khác nhau như màu sắc, cường độ mức xám, độ chói, thành phần màu... Ảnh trong không gian 2 chiều được định nghĩa là một hàm hai biến S(x, y), với S là giá trị độ sáng tại vị trí tọa độ (x, y).

Với ảnh liên tục S(x, y): Miền xác định (x, y) liên tục, miền giá trị S liên tục.

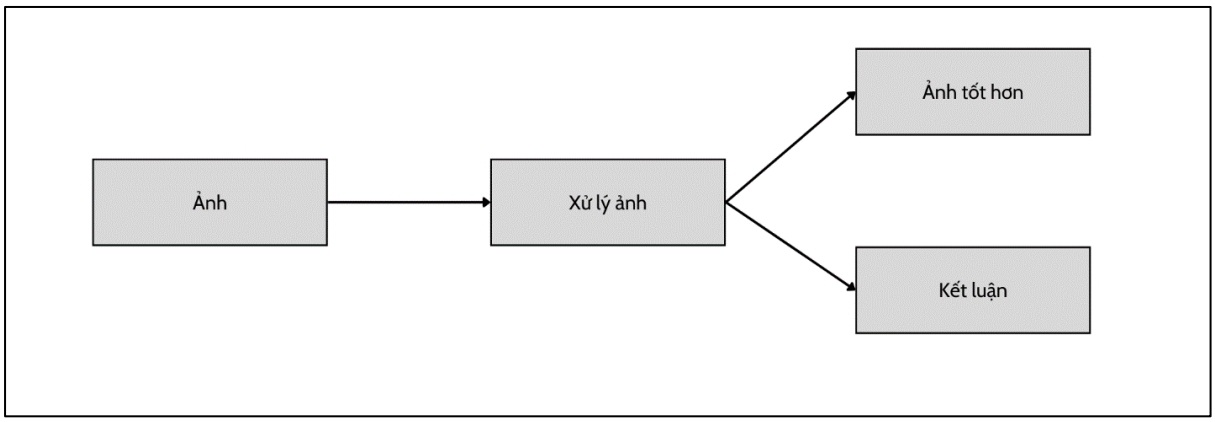
Với ảnh số S(m, n): Miền xác định (m, n) rời rạc miền giá trị S rời rạc.

Các kiểu ảnh phổ biến là:

- Ảnh nhị phân: mỗi điểm ảnh chỉ có 2 màu, hay còn gọi là ảnh trắng đen. Giá trị thường là [0;1] hoặc [0;255] tùy mục đích sử dụng.
- Ảnh xám: mỗi điểm ảnh có 1 giá trị trong khoảng [0-n], n tùy vào độ sâu màu. Ảnh này chỉ đem lại cảm nhận về hình dạng vật thể.
- Ảnh màu: mỗi điểm ảnh được tạo bởi m giá trị trong khoảng [0-n] riêng biệt tùy theo số kênh màu của ảnh. Ảnh RGB là phổ biến nhất và nó có 3 kênh màu như tên gọi (Red-Green-Blue) Đôi khi thêm một kênh Alpha tùy theo nhu cầu sử dụng là thành 4.

### 1.1.2 Quá trình xử lý ảnh

Quá trình xử lý ảnh gồm một dãy các thao tác trên ảnh đầu vào nhằm cho ra một kết quả mong muốn. Đầu ra của quá trình xử lý ảnh có thể là một ảnh tốt hơn hoặc một kết luận.




*Hình 1.1 Đầu vào và đầu ra của quá trình xử lý ảnh*

Xử lý ảnh là một tiến trình gồm nhiều công đoạn nhỏ, các công đoạn đó bao gồm:

-Thu nhận ảnh

Việc thu nhận ảnh có thể được thực hiện thông qua các thiết bị như máy ảnh, bộ cảm biến, máy quét.

-Tiền xử lý

Nhằm nâng cao chất lượng của ảnh đầu vào làm nổi bật các đặc điểm của ảnh giúp thuận lợi trong quá trình xử lý.

-Xóa nhiễu

Loại bỏ các đối tượng dư thừa trong ảnh (có thể do chất lượng của thiết bị thu nhận, do nguồn sáng)

-Nắn chỉnh hình học

Khắc phục các biến dạng do các thiết bị điện tử và quang học gây ra, có thể khắc phục bằng nhiều phép chiếu.

-Chỉnh mức xám

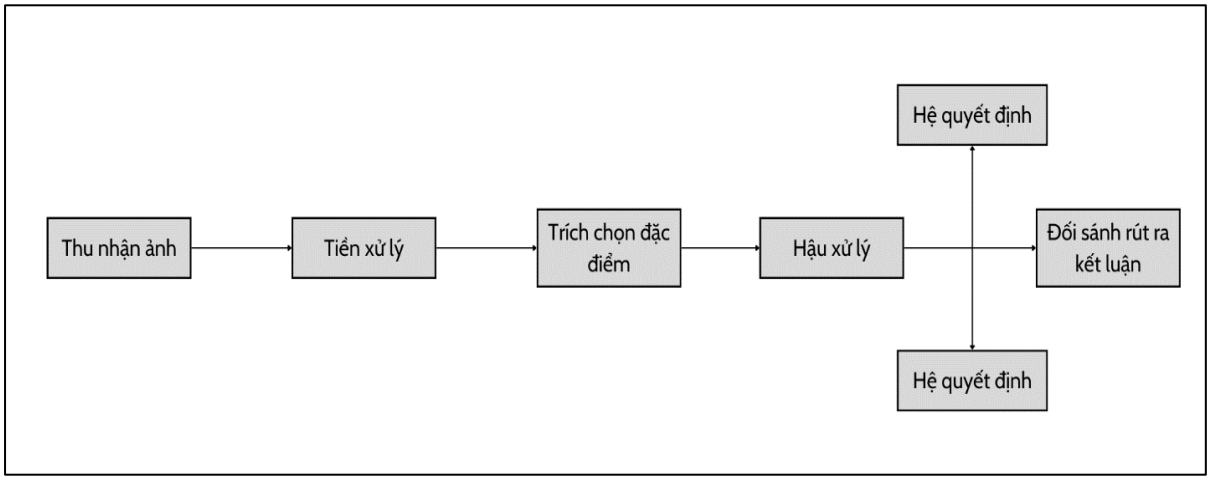
Khắc phục tính không đồng đều của mức xám, thường được dùng để tăng hoặc giảm số mức xám của ảnh.

-Trích chọn đặc trưng

Nhằm tiến tới hiểu ảnh. Có thể sử dụng các công cụ như: Dò biên để xác định, phân vùng, làm mảnh để trích xương,…

-Hậu xử lý

Nhằm hiệu chỉnh lại đặc điểm của những đặc trưng đã trích từ bước trên sao cho bước thực hiện tiếp theo được thuận tiện và nhanh chóng nhưng vẫn không làm ảnh hưởng tới kết quả.




*Hình 1.2 Các giai đoạn trong xử lý ảnh*

Tùy mục đích của ứng dụng mà chuyển sang giai đoạn khác là lưu trữ, nhận dạng, phân lớp để rút ra kết luận…

### 1.1.3 Các vấn đề của xử lý ảnh

*a, Khử nhiễu*

Trong quá trình thu nhận tín hiệu ảnh đầu vào, bộ cảm biến của thiết bị thu nhận có thể không hoàn hảo, do đó trong quá trình thu nhận có thể xảy hiện tượng nhiễu. Việc loại bỏ nhiễu trước khi tiến hành các bước tiếp theo là cần thiết để có được kết quả đầu ra mong đợi.

*b, Chỉnh mức xám*

Không đồng đều mức xám cũng là yếu tố ảnh hưởng đến chất lượng của ảnh, mức xám tập trung khiến ảnh quá sáng hoặc quá tối làm cho các đặc điểm khó lộ ra, điều này gây khó khăn trong xử lý ảnh. Việc cân bằng lại mức xám cũng là việc cần được ưu tiên trước khi tiến hành các khâu xử lý tiếp theo liên quan tới ảnh.

*c, Phân tích ảnh*

Đây là khâu quan trọng trong xử lý ảnh nhằm tiến tới hiểu ảnh. Các đặc điểm của đối tượng trong ảnh được trích chọn ra tùy theo mục đích nhận dạng trong quá trình xử lý. Trong đó có thể bao gồm: Đặc điểm không gian, đặc điểm biến đổi, đặc điểm biên và đường biên…

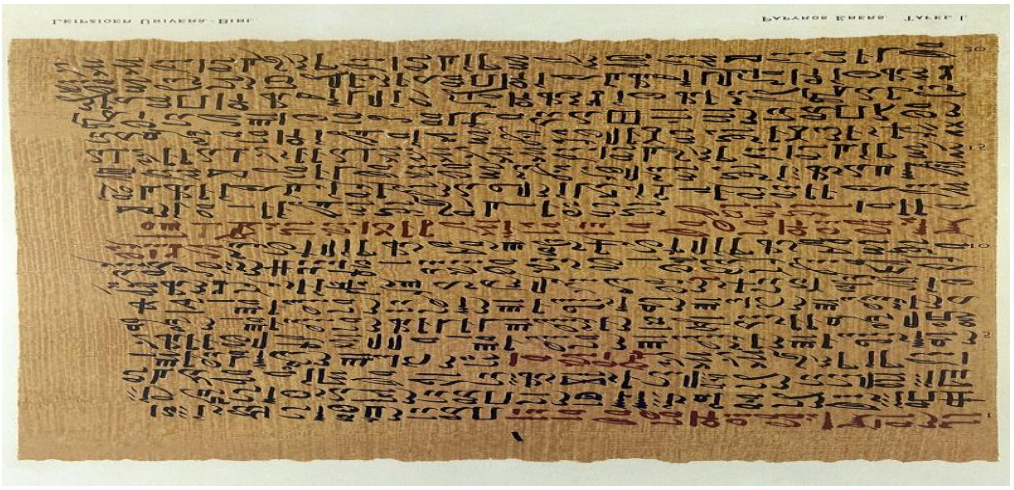
*d, Nhận dạng*

Mục tiêu giúp trả lời câu hỏi, đối tượng có tồn tại trong ảnh không, đối tượng thuộc loại nào. Đây cũng là nội dung nghiên cứu trọng tâm của đề tài này.

## 1.2 Bài toán nhận diện chữ số viết tay

### 1.2.1 Sự ra đời của bài toán và tính cấp thiết của đề tài

Chữ viết tay đã xuất hiện từ hơn 1000 năm trước kể từ khi nền văn minh con người mới bắt đầu hình thành. Các dạng thù hình đầu tiên của chữ viết tay có thể được tìm thấy ở các nền văn minh cổ đại như Ai Cập, Trung Quốc, Sumer.



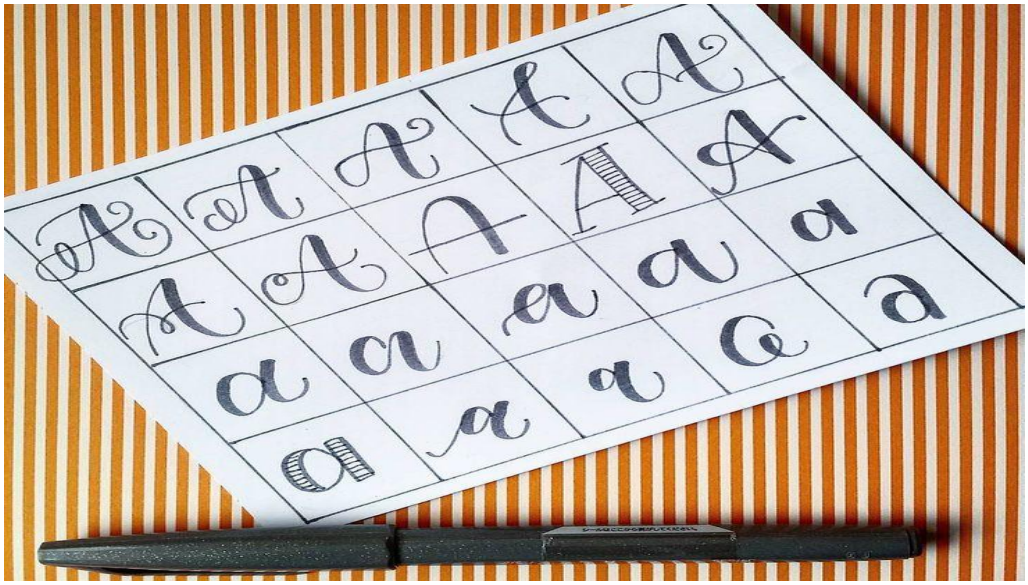
*Hình 1.3 Chữ viết của người Ai Cập cổ đại trên một mảnh giấy*

Sau hàng thiên niên kỷ phát triển của nền văn hóa, chữ viết tay cũng trở lên rất đa dạng. Nó không chỉ là công cụ để truyền tải đi thông điệp mà còn phản ảnh đặc điểm xã hội, văn hóa riêng biệt của từng vùng miền.

Mặc dù ngày nay cùng với sự phát triển khoa học công nghệ, con người đã có nhiều cách hơn để lưu trữ thông tin mà không cần dùng đến chữ viết tay. Tuy nhiên, khó có thể phủ nhận được tính tiện lợi và đơn giản của phương pháp thủ công này.

Trong thời đại số, do nhu cầu khai phá dữ liệu gia tăng, mọi thù hình thông tin đều trở lên giá trị và chữ viết tay cũng không phải ngoại lệ. Để khi xử lý thông tin ở định dạng tài liệu viết tay, người ta cần đưa nó về một khuôn dạng biểu diễn chung trên máy tính sau đó mới thực hiện xử lý. Và lúc đó bài toán nhận diện chữ viết tay được ra đời nhằm giải quyết quá trình này. Bài toán nhận đầu vào là một ảnh tài liệu hoặc chữ viết tay riêng lẻ và cho đầu ra là văn bản hoặc nhãn chữ cái tương ứng với đầu vào. Việc giải quyết được bài toán cho phép chúng ta số hóa một lượng lớn các thông tin có giá trị ở dạng văn bản viết tay. Từ nguồn thông tin đó chúng ta có thể thực hiện các thao tác phân tích phức tạp hơn.

Chữ viết tay của mỗi cá nhân mang đặc trưng riêng của chính cá nhân đó. Cùng một chữ nhưng với trăm người có cả trăm chữ viết. Cũng là chữ “a” nhưng có người viết nghiêng, có người viết thẳng, có người viết đậm, có người viết nhạt. Điều này là tích cực trong văn hóa khi cho thấy tính đa dạng về chữ viết nhưng lại là khó khăn trong bài toán nhận diện. Bởi một lẽ đơn giản như đã đề cập trước đó, máy tính không giỏi hiểu hình ảnh như cách con người hiểu chúng, do máy tính chỉ có thể làm được một công việc sau khi được lập trình cụ thể. Có vô vàn cách viết chữ “a” nhưng chúng ta không thể liệt kê hết chúng và chỉ cho máy tính biết, chúng ta chỉ có định nghĩa một cấu trúc tổng quát về chữ “a”, nói cách khác thay vì chỉ cho máy tính toàn bộ khả năng có thể tồn tại của chữ “a” ta cần phải làm cho máy tính hiểu đặc trưng gì làm lên chữ “a”.



*Hình 1.4 Sự đa dạng trong cách viết một chữ "a"*

### 1.2.2 Lợi ích và tính ứng dụng của lời giải

Việc giải quyết thành công bài toán nhận diện chữ số viết tay được xem là một bước tiến quan trọng trong lĩnh vực trí tuệ nhân tạo, mở ra nhiều cơ hội lớn cho sự phát triển của các hệ thống thông minh và tiên tiến. Đây không chỉ đơn thuần là việc giúp máy tính nhận diện được các con số viết tay mà còn đánh dấu khả năng cho máy móc hiểu và phân tích hình ảnh một cách gần gũi hơn với cách con người nhận thức thế giới xung quanh. Thành tựu này tạo nền tảng vững chắc để thúc đẩy các nghiên cứu sâu hơn trong lĩnh vực xử lý ảnh và các ứng dụng liên quan, bao gồm nhận diện mẫu, tự động hóa quy trình và cải thiện tương tác giữa con người và máy móc. Ngoài ra, sự phát triển của công nghệ nhận diện hình ảnh từ những bài toán cơ bản như thế này còn hứa hẹn mở rộng phạm vi ứng dụng sang nhiều lĩnh vực khác nhau, từ giáo dục, y tế, đến thương mại và quản lý, góp phần giải quyết những thách thức ngày càng phức tạp của cuộc sống hiện đại.

### 1.2.3 Mục tiêu nghiên cứu

Mục tiêu của nghiên cứu này là thực hiện một cách tiếp cận toàn diện đối với các vấn đề liên quan đến xử lý ảnh, đặc biệt tập trung vào bài toán nhận diện chữ viết tay. Nghiên cứu sẽ bao gồm việc tìm hiểu và tổng hợp các kiến thức lý thuyết cơ bản, phân tích các phương pháp và giải pháp đã được triển khai trong quá khứ, từ đó xác định những ưu điểm, hạn chế, cũng như các bài học kinh nghiệm có thể ứng dụng.

Một phần quan trọng của nghiên cứu là tìm hiểu sâu về mạng nơ-ron tích chập bao gồm nguyên lý hoạt động, cấu trúc, và khả năng ứng dụng trong bài toán nhận diện chữ số viết tay. Dựa trên nền tảng lý thuyết này, nghiên cứu sẽ đề xuất và triển khai một mô hình CNN phù hợp, được thiết kế và tối ưu hóa để đạt hiệu suất cao nhất trên tập dữ liệu mục tiêu. Ngoài việc phát triển mô hình, nghiên cứu cũng hướng tới việc xây dựng một chương trình phần mềm hoàn chỉnh với giao diện người dùng trực quan và thân thiện.

### 1.2.4 Phạm vi nghiên cứu

Phạm vi của nghiên cứu tập trung vào việc tìm hiểu các lý thuyết liên quan, xây dựng, cài đặt, và đánh giá một mô hình nhận diện chữ số viết tay dựa trên mạng nơ-ron tích chập.

Dữ liệu sử dụng trong nghiên cứu là bộ dữ liệu MNIST, một tập dữ liệu tiêu chuẩn và phổ biến trong lĩnh vực nhận diện chữ viết tay. Bộ dữ liệu này bao gồm 60.000 ảnh chữ số viết tay dùng để huấn luyện và 10.000 ảnh dùng để đánh giá mô hình, với các chữ số từ 0 đến 9 được lưu trữ dưới định dạng ảnh đa cấp xám kích thước 28x28 pixel. Nghiên cứu giới hạn trong phạm vi giải quyết bài toán ánh xạ từ các hình ảnh chữ số viết tay sang các nhãn số nguyên tương ứng. Nghiên cứu không mở rộng phạm vi sang các biến thể phức tạp hơn của bài toán, chẳng hạn như nhận diện chữ viết tay trên các tập dữ liệu đa dạng hơn hoặc trong môi trường thực tế với nhiễu cao. Tuy nhiên, với các kết quả thu được từ nghiên cứu này có thể cung cấp nền tảng cho việc mở rộng và áp dụng vào các bài toán nhận diện chữ viết tay phức tạp hơn trong tương lai.

### 1.2.5 Phương pháp nghiên cứu

Nghiên cứu được thực hiện trong hai giai đoạn là tìm hiểu lý thuyết cơ sở liên quan và thực nghiệm. Ở giai đoạn thứ nhất nghiên cứu sẽ xoay quanh việc tổng hợp các lý thuyết về xử lý ảnh cũng như các phương pháp đã được sử dụng cho bài toán nhận diện chữ số viết tay trước đây. Giai đoạn thứ hai sẽ dựa trên các lý thuyết đã được tổng hợp ở giai đoạn thứ nhất để tiến hành thực nghiệm từ đó thu kết quả và đưa ra nhận xét.

### 1.2.6 Những thuận lợi trong quá trình giải quyết bài toán

Nhận diện chữ viết tay là một bài toán đã được nghiên cứu từ lâu, trở thành chủ đề quan trọng trong lĩnh vực thị giác máy tính và trí tuệ nhân tạo. Sự phát triển của các nghiên cứu trước đây đã tạo nên nền tảng vững chắc, giúp cho việc tiếp cận và giải quyết bài toán này trở nên hiệu quả hơn. Thay vì phải bắt đầu từ kiến thức cơ sở, nghiên cứu có thể tham khảo và ứng dụng những ý tưởng, phương pháp và mô hình đã được đề xuất và kiểm chứng bởi cộng đồng học thuật.

Hơn nữa, nguồn dữ liệu dành cho bài toán nhận diện chữ viết tay rất phong phú và đa dạng. Các tập dữ liệu chuẩn mực như MNIST, EMNIST, hoặc IAM Handwriting Database đã được xây dựng và công bố rộng rãi, đi kèm với quá trình thu thập và gán nhãn chất lượng cao. Điều này không chỉ giúp tiết kiệm thời gian và nguồn lực mà còn đảm bảo tính đồng nhất và khả năng so sánh khi phát triển các mô hình mới. Sự sẵn có của các nguồn tài nguyên này là một lợi thế lớn, cho phép nghiên cứu tập trung vào việc cải thiện hiệu suất và khả năng tổng quát hóa của mô hình thay vì dành nhiều công sức cho khâu tiền xử lý dữ liệu.

### 1.2.7 Những khó khăn trong quá trình giải quyết bài toán

Các lý thuyết nền tảng tạo nên mạng nơ-ron tích chập (CNN) chủ yếu dựa trên các khái niệm toán học cao cấp như đại số tuyến tính, giải tích ma trận, và tối ưu hóa. Những kiến thức này đóng vai trò quan trọng trong việc hiểu sâu về cách thức hoạt động và cơ chế học tập của mô hình. Tuy nhiên, tính phức tạp của chúng có thể trở thành một rào cản nhất định đối với những người mới tiếp cận lĩnh vực này hoặc chưa có nền tảng vững chắc về toán học.

Mặc dù vậy, nhờ vào sự phát triển của các thư viện mã nguồn mở như TensorFlow và PyTorch, việc triển khai CNN đã trở nên dễ dàng hơn bao giờ hết. Các công cụ này giúp che giấu nhiều phần phức tạp, cho phép người lập trình tập trung vào việc phát triển thay vì phải hiểu cặn kẽ mọi khía cạnh toán học bên dưới. Tuy nhiên, để đạt được khả năng tối ưu hóa và tùy chỉnh mô hình một cách hiệu quả, việc nắm vững những nguyên lý toán học nền tảng vẫn là một lợi thế quan trọng nên đầu tư thời gian và công sức để tìm hiểu.

### 1.2.8 Đầu vào của bài toán nhận diện chữ số viết tay

Đầu vào của bài toán nhận diện chữ số viết tay là hình ảnh chứa các ký tự số, thường được biểu diễn dưới dạng ma trận pixel. Mỗi pixel mang giá trị số thể hiện mức độ sáng tối, đối với bài toán này hình ảnh đa cấp xám được sử dụng phổ biến. Để thuận tiện cho việc xử lý và tăng hiệu quả tính toán, các hình ảnh thường được chuẩn hóa về kích thước, chẳng hạn như 28x28 pixel trong tập dữ liệu MNIST. Mỗi đầu vào đi kèm với một nhãn đại diện cho chữ số tương ứng, giúp định hướng quá trình huấn luyện của mô hình.

### 1.2.9 Đầu ra của bài toán nhận diện chữ số viết tay

Đầu ra của bài toán nhận diện chữ số viết tay là một dự đoán về chữ số mà mô hình nhận diện được từ hình ảnh đầu vào. Thông thường, đầu ra thường được biểu diễn dưới dạng một vector xác suất, với mỗi phần tử trong vector đại diện cho xác suất tương ứng của từng chữ số (từ 0 đến 9). Phần tử có giá trị lớn nhất trong vector xác suất sẽ được chọn làm dự đoán cuối cùng của mô hình.

## 1.3 Tóm tắt chương

- Xử lý ảnh là một lĩnh vực quan trọng trong khoa học máy tính, một trong những bài toán nổi bật trong xử lý ảnh là bài toán nhận diện chữ viết tay
- Bài toán nhận đầu vào là ảnh chữ viết tay, và cho ra văn bản tương ứng
- Bài toán khó do sự da dạng trong hình dáng chữ viết, điều này đòi hỏi một giải pháp thích nghi với vấn đề trên
- Đã có nhiều phương pháp tiếp cận bài toán trong quá khứ, hiệu quả nhất là phương pháp sử dụng mạng tích chập
- Đề tài hướng đến việc xây dựng một kiến trúc tích chập cho việc nhận diện chữ số viết tay sử dụng MNIST làm bộ dữ liệu huấn luyện mạng

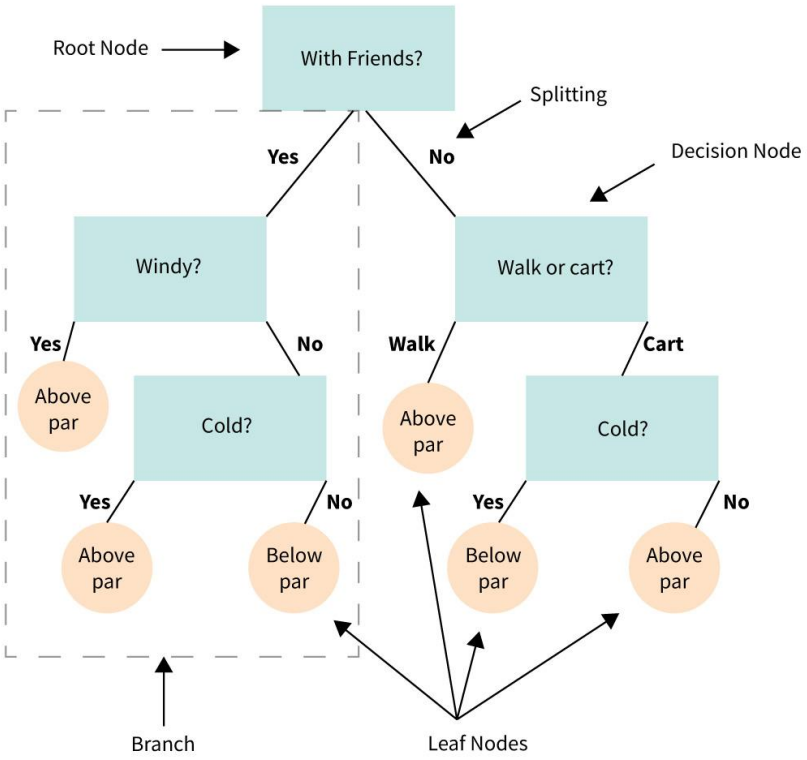
# CHƯƠNG 2 MỘT SỐ PHƯƠNG PHÁP HIỆN CÓ

Trong chương này nhóm sẽ trình bày hai phương pháp tiếp cận đã được sử dụng trong quá khứ là Random Forest và Support Vector Machine, từ đó đánh giá ưu nhược điểm của hai phương pháp trên sau đó trình bày về kiến trúc CNN và cách thức giải quyết bài toán nhận diện chữ số viết tay với kiến trúc CNN.

## 2.1 Mô hình Random Forest

### 2.1.1 Mô hình Decision Tree

Cây quyết định là một thuật toán học máy có giám sát, cây quyết định cho phép giải quyết cả bài toán hồi quy lẫn bài toán phân lớp. Cây được cấu thành từ nhiều nút có liên kết với nhau, mỗi nút trên cây quyết định là một phép toán kiểm tra điều kiện, với mỗi kết quả của phép kiểm tra điều kiện sẽ là một liên kết tới một nút con khác. Các nút lá sẽ là các nút quyết định, đưa ra kết quả dự báo cuối cùng.




*Hình 2.1 Ảnh minh họa cây quyết định*

Giống như mọi thuật toán học máy khác, cây quyết định cũng có hai pha là huấn luyện và dự báo.

*a, Pha huấn luyện*

Tại pha huấn luyện của cây quyết định, ta cần tìm cách xây dựng cây quyết định. Cây sẽ được định hình bằng việc ta lựa chọn thuộc tính nào làm thuộc tính quyết định ở mỗi nút. Để công bằng nhất cho các thuộc tính, ta sẽ xem xét từng thuộc tính và đánh giá xem thuộc tính nào mang lại nhiều "thuận lợi" nhất. Từ "thuận lợi" ở đây được định nghĩa là tính phân chia khi ta phân hoạch tập dữ liệu ban đầu. Cụ thể hơn, nếu tập phân hoạch có phân bố nhãn không đều, một nhãn xuất hiện nhiều hơn hoàn toàn các nhãn còn lại khi đó phân hoạch của ta theo một thuộc tính được đánh giá là "thuận lợi". Ngược lại nếu các nhãn phân bố đều tập phân hoạch của ta được đánh giá là không thuận lợi. Điều này có ý nghĩa gì tới việc xây dựng cây quyết định? Việc chọn được một thuộc tính phân tập dữ liệu ra thành các tập con chỉ có 1 nhãn chiếm đa số cũng đồng nghĩa với việc ta đã tìm ra được một yếu tố quyết định nhãn. Vậy nên khi xây dựng cây, tại mỗi nút ta cần quyết định thuộc tính nào là thuận "lợi nhất".

Về mặt toán học để đánh giá độ thuận lợi cho phân hoạch ta có thể sử dụng tới công thức tính entropy.

$$H(S) = \sum_{i}^{n} - p_i \log p_i$$

Với:

n là tập các nhãn trong phân hoạch S

$p_i$ là xác suất nhãn i xuất hiện trong phân hoạch s

Với công thức trên ta có thể thấy nếu các nhãn phân bố đều, H(S) sẽ đạt giá trị lớn nhất là log(n), ngược lại nếu chỉ có một nhãn duy nhất trong phân hoạch S, giá trị của H(S) sẽ đạt nhỏ nhất là 0. Từ đó ta có thể kết luận giá trị entropy trong một phân hoạch càng nhỏ thì độ "thuận lợi" càng lớn.

Do một thuộc tính có thể cho ra nhiều phân hoạch, do đó ta cần tìm cách đánh giá độ "thuận lợi" cho thuộc tính đó mà vẫn giữ được tính công bằng cho từng phân hoạch. Công thức đó được xây dựng về mặt toán học như sau.

$$H(x, S) = \sum_{i}^{k} \frac{m_i}{n} H(S_i)$$

Với:

x là thuộc tính đang xét

S là phân hoạch hiện tại

k là tập các giá trị của thuộc tính x

$m_i$ là số phần tử trong phân hoạch hiện tại có thuộc tính x mang giá trị i

n là số phần tử trong phân hoạch hiện tại

$S_i$ là phân hoạch con của phân hoạch hiện tại chứa các phần tử có thuộc tính x mang giá trị i

Với công thức trên ta tổng hợp được mã giả như sau cho quá trình xây dựng quyết định

```
def xây_dựng_cây_quyết_định(tập_dữ_liệu) :
    if có_phải_chỉ_có_một_nhãn(tập_dữ_liệu) == 0 :
        return Nút_quyết_định(nhãn_duy_nhất)
    nút_hiện_tại = Null
    thuộc_tính_thuận_lợi = Null
    min_h = infinity
    for thuộc_tính in tập_thuộc_tính :
        entropy = h(tập_dữ_liệu, thuộc tính)
        if entropy < min_h :
            thuộc_tính_thuận_lợi = thuộc_tính
            min_h = entropy
    nút_hiện_tại.thuộc_tính_quyết_định = thuộc_tính_thuận_lợi
    for giá_trị in các_giá_trị_của(thuộc_tính_thuận_lợi) :
        cây_con = xây_dựng_cây_quyết_định(phân hoạch(tập dữ liệu, thuộc_tính_thuận_lợi,
giá_trị))
        nút_hiện_tại.bổ_sung_cây_con(giá_trị, cây_con)
    return nút_hiện_tại
```

*b, Pha dự báo*

Sau khi quá trình huấn luyện diễn ra, mô hình sẽ được làm việc với dữ liệu thực tế chưa được thấy trong tập huấn luyện. Để dự báo nhãn cho một mẫu, ta sẽ thực hiện duyệt cây từ gốc xuống lá. Tại mỗi nút, ta sẽ quan sát thuộc tính mà nút đó yêu cầu rồi di chuyển đến nút con thỏa mãn điều kiện tương ứng với giá trị thuộc tính. Quá trình này kết thúc khi ta di chuyển được đến nút lá(nút quyết định) và có được nhãn.

Ta tổng hợp được mã giả cho quá trình dự báo:

```
1. def dự_báo(mẫu_quan_sát, nút_cây_quyết_định) :
2.     if là_lá(nút_cây_quyết_định) :
3.         return nút_cây_quyết_định.giá_trị_dự_báo
4.     thuộc_tính_cần_quan_sát = nút_cây_quyết_định.thuộc_tính_cần_quan_sát
5.     return dự_báo(mẫu_quan_sát, nút_con_theo_giá_trị_thuộc_tính)
```

Hệ thống nhận dạng thường bao gồm hai bước là rút trích đặc trưng từ ảnh và học tự động từ các đặc trưng để có thể nhận dạng ký tự. Hiệu quả của một hệ thống nhận dạng phụ thuộc vào các phương pháp được sử dụng ở từng giai đoạn trên. Đối với hướng tiếp cận sử dụng rừng ngẫu nhiên cho bài toán nhận diện chữ số viết tay, ta cần làm rõ được giai đoạn trích chọn đặc trưng.

Trong nhiều năm trở lại đây, cộng đồng nghiên cứu về thị giác máy tính và xử lý ảnh đặc biệt quan tâm tới hai phương pháp trích chọn đặc trưng rất hiệu quả là đặc trưng cục bộ không đổi (SIFT) của Lowe và đặc trưng toàn cục của Oliva và Torralba. Các vector mô tả ảnh được trích xuất theo phương pháp SIFT có một tính chất quan trọng đó là không bị thay đổi trước các phép biến đổi tỷ lệ, tịnh tiến, phép quay, không bị thay đổi một phần với phép biến đổi hình học affine (thay đổi góc nhìn) và khả năng ổn định với những thay đổi về độ sáng, sự che khuất hay nhiễu. Tuy nhiên sự bất biến với phép quay của phương pháp SIFT không phù hợp với bài toán nhận diện chữ số do số 6 và số 9 được coi tương đương. Ở mặt khác GIST không gặp những khó khăn như SIFT, do đó GIST được ưa chuộng hơn.

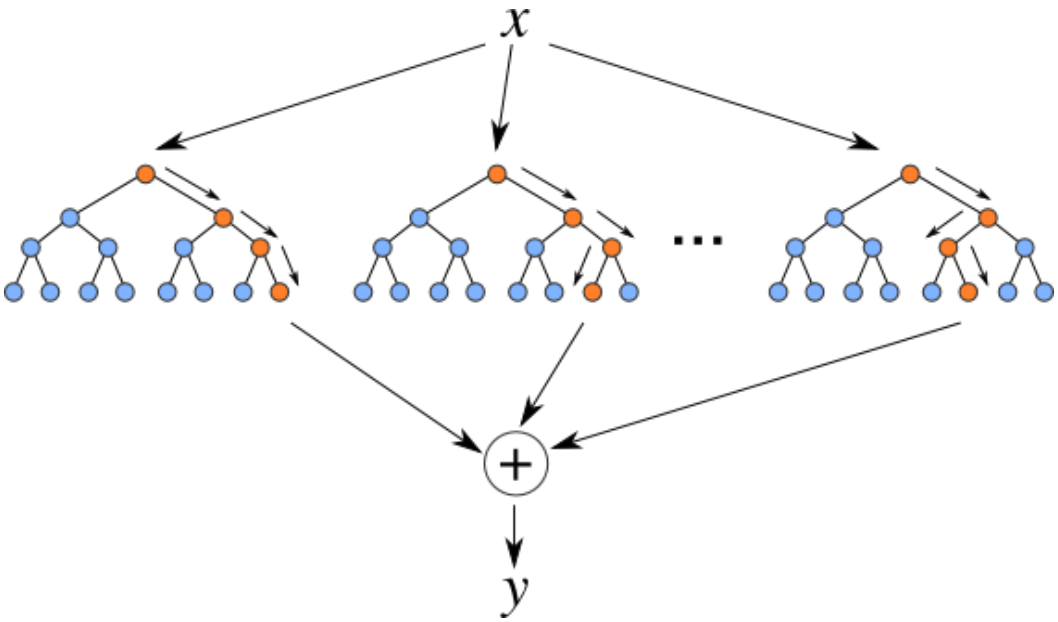
Trong bài báo “Nhận Dạng Ký Tự Số Viết Tay Bằng Giải Thuật Máy Học” tác giả sử dụng GIST cho bước trích chọn đặc trưng và rừng ngẫu nhiên cho bước dự báo. Hướng tiếp cận này cho ra kết quả ấn tượng khi đạt độ chính xác lên tới 99.12% cho tập dữ liệu MNIST.

### 2.1.2 Khắc phục Overfitting bằng việc sử dụng nhiều cây quyết định

Việc sử dụng duy nhất một cây quyết định trên toàn tập dữ liệu nhiều khả năng sẽ khiến cho mô hình của chúng ta gặp vấn đề quá khớp, bởi chỉ cần gia tăng chiều sâu của cây ta có thể biểu diễn thông tin về gần như toàn bộ các mẫu trong tập huấn luyện.

Để giải quyết vấn đề này, một phương pháp tiếp cận tốt hơn đã được đề xuất. Thay vì chỉ dùng duy nhất một cây quyết định với tập huấn luyện, ta có thể xây dựng

nhiều cây quyết định khác nhau từ các tập con bất kỳ của tập huấn luyện, quyết định về nhãn cuối cùng trong pha dự báo sẽ là quyết định đa số. Phương pháp này được gọi là rừng ngẫu nhiên. Rừng ngẫu nhiên tỏ ra vô cùng hiệu quả trong bài toán dự báo, quyết định cuối cùng không được đưa ra bởi một cây quyết định duy nhất mà là từ nhãn chiếm đa số từ tất cả các cây, điều này giảm thông số variance đi đáng kể khiến cho mô hình đạt được trạng thái lý tưởng.




*Hình 2.2 Ảnh minh họa quá trình dự đoán với rừng ngẫu nhiên*

Hệ thống nhận dạng thường bao gồm hai bước là rút trích đặc trưng từ ảnh và học tự động từ các đặc trưng để có thể nhận dạng ký tự. Hiệu quả của một hệ thống nhận dạng phụ thuộc vào các phương pháp được sử dụng ở từng giai đoạn trên. Đối với hướng tiếp cận sử dụng rừng ngẫu nhiên cho bài toán nhận diện chữ số viết tay, ta cần làm rõ được giai đoạn trích chọn đặc trưng.

Trong nhiều năm trở lại đây, cộng đồng nghiên cứu về thị giác máy tính và xử lý ảnh đặc biệt quan tâm tới hai phương pháp trích chọn đặc trưng rất hiệu quả là đặc trưng cục bộ không đổi (SIFT) của Lowe và đặc trưng toàn cục của Oliva và Torralba. Các vector mô tả ảnh được trích xuất theo phương pháp SIFT có một tính chất quan trọng đó là không bị thay đổi trước các phép biến đổi tỷ lệ, tịnh tiến, phép quay, không bị thay đổi một phần với phép biến đổi hình học affine (thay đổi góc nhìn) và khả năng ổn định với những thay đổi về độ sáng, sự che khuất hay nhiễu. Tuy nhiên sự bất biến với phép

quay của phương pháp SIFT không phù hợp với bài toán nhận diện chữ số do số 6 và số 9 được coi tương đương. Ở mặt khác GIST không gặp những khó khăn như SIFT, do đó GIST được ưa chuộng hơn.

Trong bài báo "Nhận Dạng Ký Tự Số Viết Tay Bằng Giải Thuật Máy Học" tác giả sử dụng GIST cho bước trích chọn đặc trưng và rừng ngẫu nhiên cho bước dự báo. Hướng tiếp cận này cho ra kết quả ấn tượng khi đạt độ chính xác lên tới 99.12% cho tập dữ liệu MNIST.

## 2.2 Mô hình Support Vector Machine

### 2.2.1 Ý tưởng về phân chia lớp theo siêu phẳng

Giả sử tập dữ liệu của chúng ta là tập tách biệt tuyến tính, bằng trực giác ta có thể nảy ngay ra một ý tưởng xấp xỉ hàm dự đoán nhãn cho các mẫu quan sát. Đúng vậy, đó là tìm một siêu phẳng phân tách chúng, các điểm dữ liệu nằm trên cùng một bờ của siêu phẳng của dữ liệu mang nhãn giống nhau và ngược lại.

Trong không gian n chiều, siêu phẳng được mô tả là một phương trình với n tham số.

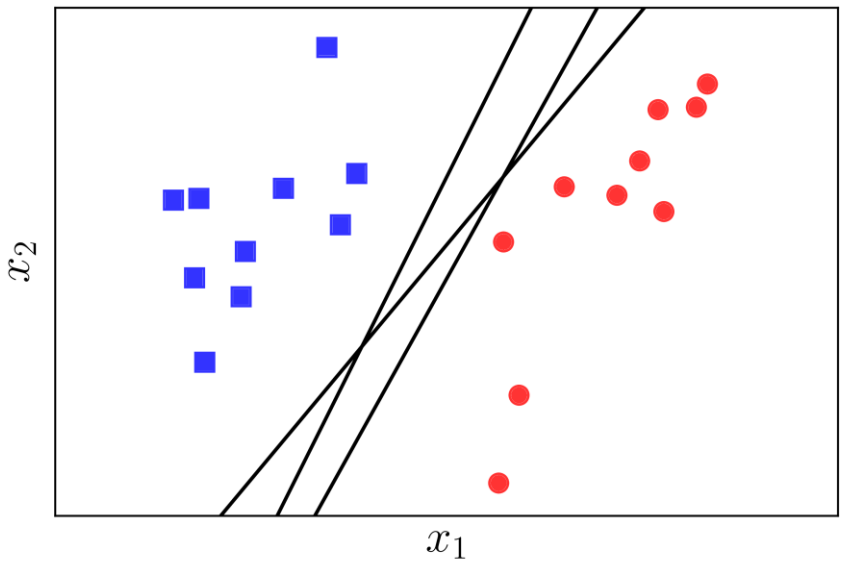
$$f(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_{n-1}x_{n-1} = 0$$

Để xác định một điểm dữ liệu thuộc bờ nào của siêu phẳng, ta chỉ cần thay điểm dữ liệu đó vào $f(\mathbf{x})$ và quan sát dấu.

Nếu $f(\mathbf{x}) = 0$ điểm dữ liệu nằm trên siêu phẳng

Nếu $f(\mathbf{x}) < 0$ điểm dữ liệu thuộc bờ âm

Nếu $f(\mathbf{x}) > 0$ điểm dữ liệu thuộc bờ dương




*Hình 2.3 Tập dữ liệu với hai nhãn phân tách tuyến tính*

### 2.2.2 Tìm siêu phẳng với Perceptron

Vấn đề đặt ra là nếu siêu phẳng tồn tại, làm thế nào để ta tìm ra được chúng và nếu có nhiều siêu phẳng cùng tồn tại cái nào là cái tốt nhất ta nên chọn. Giải quyết câu hỏi đầu tiên dẫn ta đến với giải thuật học Perceptron. Ý tưởng của thuật toán vô cùng đơn giản, ta xuất phát với một siêu phẳng bất kỳ, tại mỗi bước lặp ta lần lượt duyệt qua từng mẫu quan sát trong tập huấn luyện và dự đoán nhãn của chúng với siêu phẳng hiện tại. Nếu nhãn đúng ta không làm gì, nếu nhãn sai ta thực hiện hiệu chỉnh tham số của siêu phẳng. Thuật toán này có tính đúng và đã được chứng minh hội tụ, nếu tồn tại sau một số lần hữu hạn ta sẽ tìm được siêu phẳng phân tách hai lớp dữ liệu.

Công thức hiệu chỉnh cho các mẫu sai:

$$\mathbf{w} = \mathbf{w} + y_i * \mathbf{x}_i * \eta$$

Trong đó:

**w** là tham số của siêu phẳng

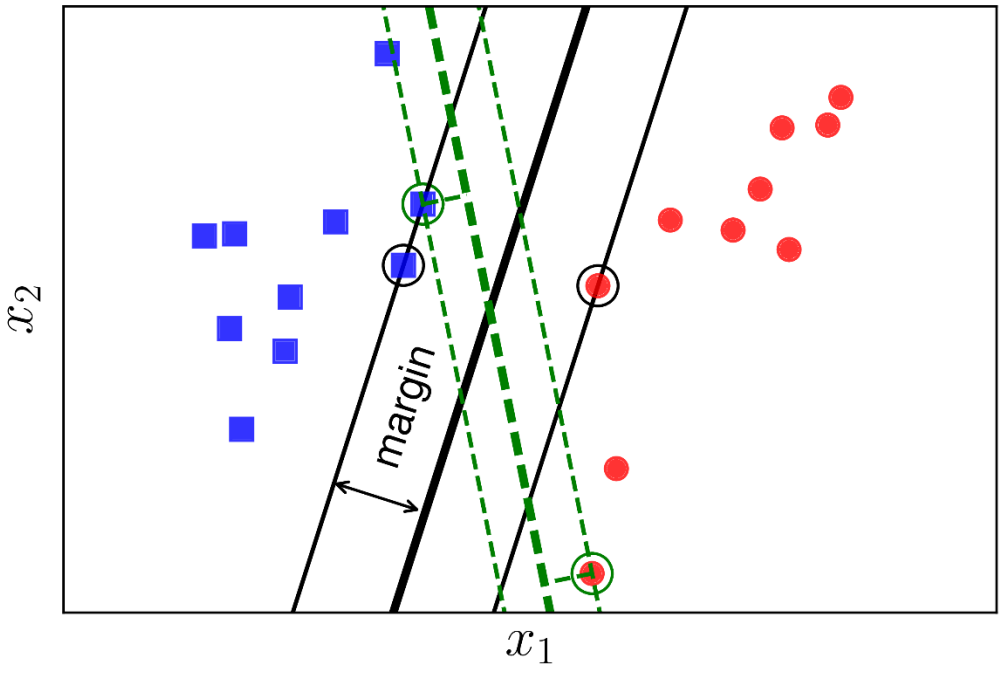
$y_i$ là nhãn thực tế của mẫu quan sát

$\mathbf{x}_i$ là vector mô tả mẫu quan sát thứ i

$\eta$ là hệ số học

### 2.2.3 Tìm siêu phẳng với Support Vector Machine

Vấn đề thứ nhất đã được giải quyết giờ đến với vấn đề thứ hai, làm thế nào để ta biết siêu phẳng nào là tốt nhất. Siêu phẳng tốt nhất có lẽ là siêu phẳng thỏa mãn hai điều kiện. Một là "công bằng" nhất, không thiên vị một nhãn nào hơn nhãn nào. Nếu định nghĩa khoảng cách một nhãn tới một siêu phẳng là khoảng cách từ điểm dữ liệu gần nhất mang nhãn đó tới siêu phẳng thì để đạt được "công bằng" khoảng cách từ hai nhãn tới siêu phẳng phải bằng nhau. Hai là "rộng rãi" nhất, khoảng cách từ hai nhãn tới siêu phẳng phải lớn nhất. Đó cũng chính là ý tưởng chủ chốt cho giải thuật Support Vector Machine (SVM).




*Hình 2.4 So sánh về tính công bằng và rộng lượng của hai siêu phẳng*

Câu trả lời trên cho câu hỏi thứ 2 lại yêu cầu ta tìm ra một hướng tiếp cận khác cho câu hỏi thứ nhất. Vậy để tìm được siêu phẳng đảm bảo cả hai yếu tố công bằng và rộng lượng ta cần làm gì.

Bằng chứng minh toán học, người ta đã xây dựng được bài toán tối ưu cho vấn đề này. Bài toán được xây dựng là bài toán quy hoạch toàn phương và hoàn toàn có thể giải quyết được bằng công cụ tìm nghiệm

Đối với SVM khâu trích chọn đặc trưng sẽ khác với rừng ngẫu nhiên. Do SVM chỉ làm việc tốt trên các tập dữ liệu được phân tách tuyến tính do đó trong trường hợp bộ MNIST không thỏa mãn đặc điểm trên có thể là một rào cản lớn. Do đó cần có một ánh xạ đưa tập MNIST huấn luyện ban đầu sang một không gian mới mà tại đó dữ liệu từ các lớp khác nhau có thể được phân tách bởi các siêu phẳng.

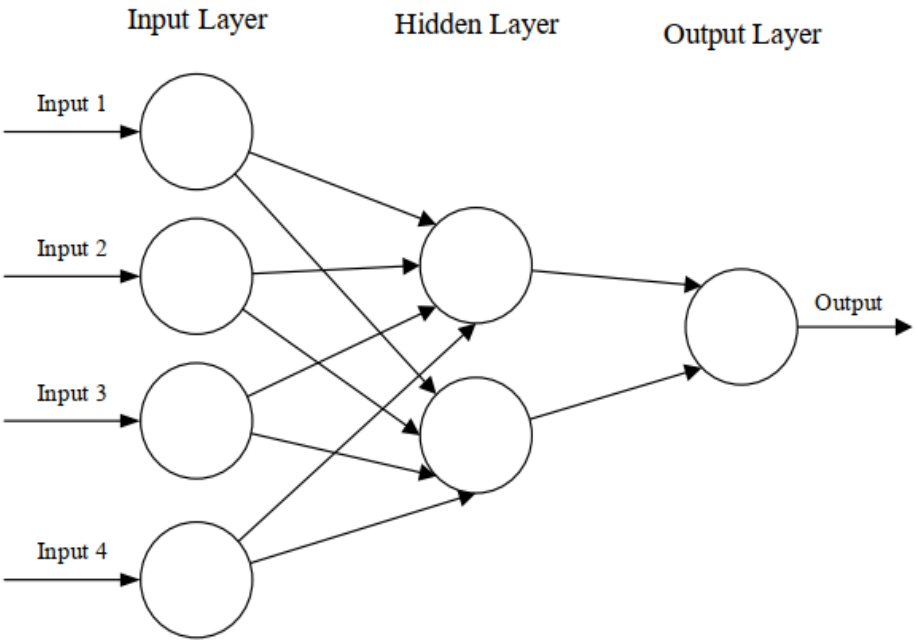
Kernel là một phương pháp phổ biến được sử dụng, ý tưởng của kernel là thêm một chiều mới mang giá trị được tính toán từ các chiều đã có cho các điểm dữ liệu. Đây cũng là hướng tiếp cận được tác giả Krzysztof Sopyła nhắc tới trong nghiên cứu của mình. Bằng phương pháp này, tác giả đã huấn luyện được mô hình phân lớp với SVM cho bộ dữ liệu MNIST với độ chính xác lên tới 98.52%. Dù rằng con số này không quá ấn tượng so với các phương pháp hiện đại khác nhưng vẫn là đáng suy ngẫm về mặt nghiên cứu.

## 2.3 Mô hình mạng CNN

### 2.3.1 Mô hình mạng Dense

Mạng nơ ron nhân tạo là một hệ thống tính toán và xử lý thông tin được lấy cảm hứng từ hệ thần kinh sinh học. Mạng nơ ron nhân tạo bao gồm các nút xử lý (gọi là nơ ron) được là kết nối với nhau. Mô hình cơ bản nhất của mạng nơ ron nhân tạo đó chính là mạng Dense.

Như được minh họa ở hình dưới, mạng Dense bao gồm các lớp chứa các nơ ron, mỗi nơ ron tại mỗi lớp đều có kết nối tới các lớp tiếp theo và trước đó. Lớp đầu tiên được gọi là input layer, lớp cuối cùng cho đầu ra được gọi là output layer và các lớp ở giữa được gọi là các hidden layer. Vector đầu vào sẽ đi qua lớp input và lớp hidden sau đó đầu ra sẽ được thể hiện ở lớp output. Quá trình tính toán dữ liệu tại các lớp sẽ được quyết định bởi các tham số của mạng, cụ thể để tính đầu ra của một nút ta sẽ nhân vô hướng vector đầu vào tại nút đó với các trọng số **w** của nút rồi cộng với một lượng bias sau đó đưa kết quả qua một hàm gọi là hàm kích hoạt để tạo tính phi tuyến cho mô hình. Các trọng số **w** và *bias* sẽ được tính toán trong quá trình học của mạng thông qua một thuật toán gọi là lan truyền ngược.




*Hình 2.5 Kiến trúc mạng Dense đơn giản với 3 lớp*

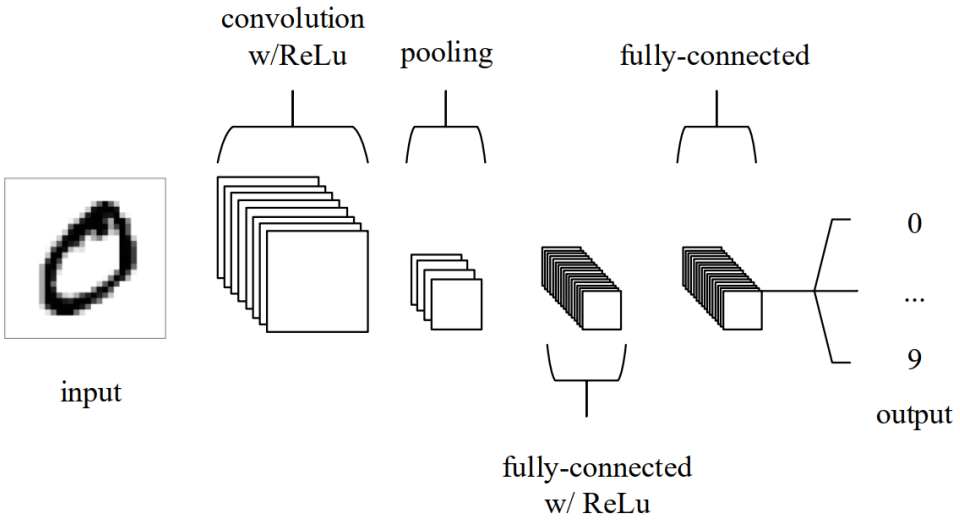
Mặc dù cho ra kết quả ấn tượng trong nhiều bài toán với dữ liệu đầu vào phức tạp, nhưng mạng Dense lại tồn tại một điểm yếu chí mạng khiến cho nó không phù hợp với các bài toán trong xử lý ảnh.

Mạng Dense gặp khó khăn trong việc tính toán đối với các bài toán xử lý ảnh bởi nó yêu cầu một số lượng tham số khổng lồ. Để dễ hình dung ta có thể lấy ví dụ với bộ dữ liệu MNIST, bộ dữ liệu bao gồm ảnh các chữ số viết tay ở dạng đa cấp xám với kích thước 28*28 pixels. Với kích thước bộ dữ liệu trên ta có thể thấy ngay từ lớp input ta đã cần tới tận 784 nơ ron cho chỉ việc biểu diễn các ảnh đen và trắng. Con số này còn lớn hơn nữa nếu ta liệt kê cả số lượng tham số ở các lớp ẩn.

Không chỉ dừng lại ở đó, nếu như ảnh đầu vào của chúng ta có kích thước lớn hơn, thậm chí còn có cả màu sắc, số lượng tham số để biểu diễn mạng xấp xỉ bộ dữ liệu trên quả là giá trị điên rồ cho khâu huấn luyện. Việc số lượng tham số lớn không chỉ tạo khó khăn cho tính toán mà còn có thể gây lên nhiều tác dụng phụ khác trong đó có cả hiện tượng quá khớp. Điều này đặt ra yêu cầu về một giải pháp giúp giảm bớt độ phức tạp tính toán của mạng Dense, và đó là lúc kiến trúc CNN được ra đời.

### 2.3.2 Các loại lớp trong kiến trúc CNN

Khác với mạng Dense truyền thống khi chỉ có một loại lớp đó là fully-connected, trong kiến trúc CNN có thêm hai loại lớp mới là lớp pooling và lớp tích chập.




*Hình 2.6 Kiến trúc CNN*

Cơ chế làm việc của kiến trúc trên có thể được mô tả thông qua hoạt động của các thành phần như sau:

1. Đầu tiên giống như ở mạng Dense, lớp input sẽ nhận giá trị các điểm ảnh của đầu vào.
2. Tiếp theo sau đó ảnh đầu vào sẽ được nhân chập với các mặt nạ có kích thước và số lượng xác định, công việc này được diễn ra ở lớp tích chập, kết quả của phép tích chập sẽ đi qua một hàm kích hoạt với mục đích tạo tính phi tuyến cho mô hình. Thông thường hàm được sử dụng sẽ là hàm ReLu. Đầu ra của lớp tích chập sẽ là một loạt các ảnh con, tương ứng với các mặt nạ đã nhân chập.
3. Các ảnh sau khi thu được từ quá trình tích chập sẽ được làm giảm kích thước bằng việc chia nhỏ thành cách phần bằng nhau sau đó tính toán ảnh kết quả bằng việc lấy giá trị max ở mỗi phần. Quá trình này diễn ra tại lớp pooling. Kết quả của quả của lớp pooling là tập các ảnh nhỏ hơn ảnh đầu vào, đây cũng chính là mục đích chính của mạng tích chập, giảm kích thước đầu vào gốc bài toán. Đối với đầu vào lớn và phức tạp quá trình tính tích chập và pooling có thể được diễn ra nhiều lần.

4. Tiếp ngay sau đó ảnh kết quả sẽ được làm phẳng và đưa vào lớp fully-connect có cấu trúc tương tự trong mạng Dense.

*a, Lớp tích chập*

Cái tên đã nói lên tất cả, lớp tích chập là thành phần đóng vai trò quan trọng trong cách vận hành của mạng CNN. Ý tưởng tích chập được lấy cảm hứng từ phép nhân chập ảnh, người ta tin rằng nếu nhân chập với một mặt nạ phù hợp, ta sẽ được ảnh kết quả làm lộ ra đặc trưng của ảnh ban đầu. Thật vậy, chúng ta đã quá quen thuộc với khái niệm này trong môn xử lý ảnh. Với mỗi mặt nạ khác nhau, sau khi nhân chập chúng ta nhận được ảnh phản ánh một đặc trưng nào đó của ảnh ban đầu.

Ví dụ với mặt nạ:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$ Ta có phép làm mịn

Với mặt nạ:

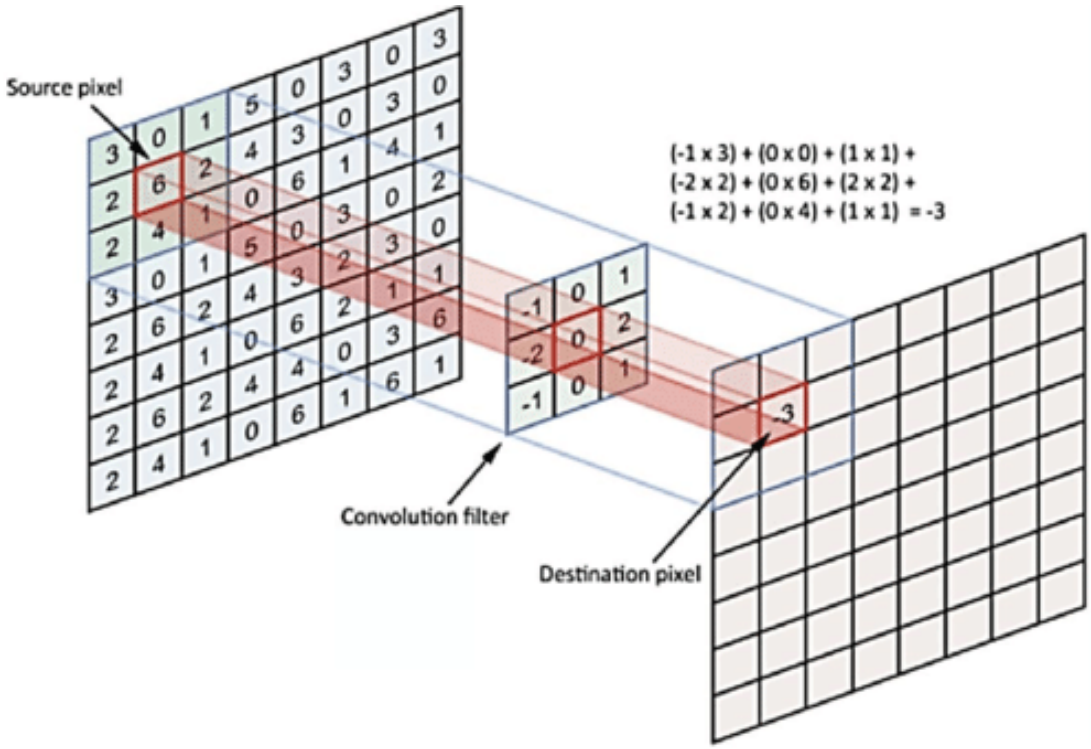
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$ Ta có phép tìm biên theo trục x

Với mặt nạ:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$ Ta có phép tìm biên theo trục y

Điều tương tự cũng diễn ra trong lớp tích chập nhưng với mặt nạ ở đây là tham số cần học của mô hình CNN. Hay nói cách khác các giá trị của mặt nạ sẽ được hiệu chỉnh trong quá trình học để làm sao cho việc làm nổi bật đặc trưng trở lên tối ưu nhất. Các đặc trưng mà mặt nạ trong các lớp tích chập của CNN phản ánh thường có phần "không gần gũi" với nhãn quan của con người nên việc hiển thị kết quả nhân chập cũng khó có thể biết đặc trưng mà mặt nạ phản ánh là gì. Các đặc trưng như vậy thường xuất hiện trong quá trình huấn luyện mạng nơ ron và được gọi là các đặc trưng ẩn.

Cụ thể hơn về khái niệm nhân chập, để nhân chập một mặt nạ với một ảnh ta sẽ cần di chuyển tâm của mặt nạ đi qua tất cả các điểm ảnh trên ảnh gốc, tại mỗi vị trí mà mặt nạ đi qua ta sẽ thực hiện phép nhân các phần tử tương ứng trên mặt nạ với các phần tử trên ảnh mà mặt nạ đè lên.




*Hình 2.7 Hình ảnh mô tả quá trình nhân chập*

Đến đây một vấn đề mới phát sinh, khi di chuyển tâm của mặt nạ ra tới các điểm ở rìa, ta nhận thấy sẽ có vài điểm trên mặt nạ không nằm trong ảnh. Điều này có thể được giải quyết bằng việc không tính các vị trí ở rìa hoặc đệm thêm một viền toàn các giá trị 0 ở ngoài, kỹ thuật này được gọi là **zero padding**.

Sau khi nhân chập, output sẽ được đưa qua một hàm kích hoạt, hàm này thường sẽ là hàm phi tuyến với mục đích làm bẹp giá trị giúp mô hình có thể xấp xỉ các bộ dữ liệu phức tạp hơn.

Như vậy chung quy lại để định nghĩa một lớp tích chập trong kiến trúc CNN chúng ta cần định nghĩa các siêu tham số sau:

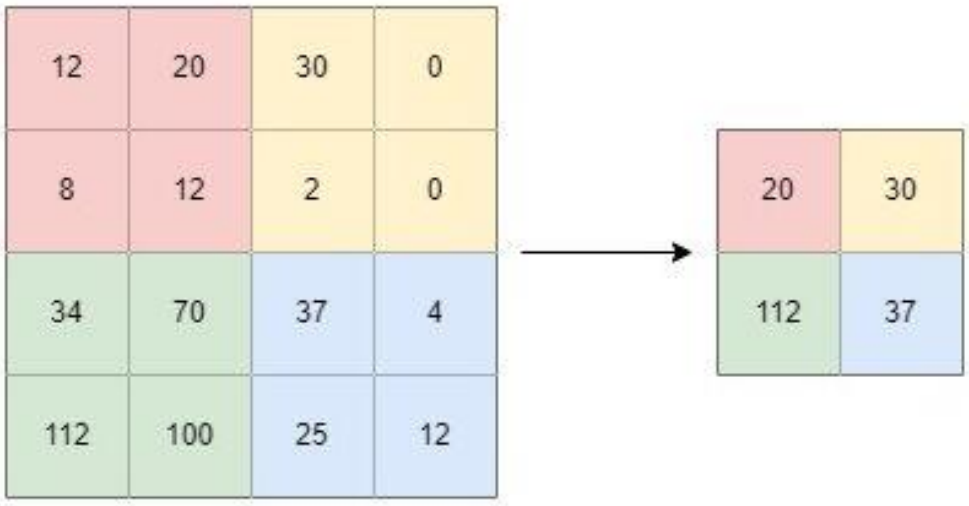
- Số lượng các mặt nạ
- Kích thước các mặt nạ
- Hàm kích hoạt

- Phương thức nhân chập (bỏ viền hay không bỏ viền)

*b, Lớp pooling*

Như đã đề cập ở trên, mạng CNN ra đời với mục đích làm giảm độ phức tạp tính toán cho mô hình nhưng ở lớp tích chập phía trên ta không những chưa thấy mạng giảm kích thước ảnh đầu vào mà còn lại thêm khối lượng tính toán cho mô hình khi cho ra kích thước output còn lớn hơn cả input. Vậy quá trình giảm khối lượng tính toán được diễn ra ở đâu? Câu trả lời đó chính là ở lớp pooling.

Lớp pooling hoạt động bằng cách chia ảnh ban đầu thành các phần hình vuông có kích thước bằng nhau gọi là các pool, sau đó thay thế mỗi pool đó bằng giá trị trung bình hoặc giá trị lớn nhất trong đó. Bên cạnh tính toán các pool thông qua việc lấy max hoặc giá trị trung bình, chúng ta cũng có thể sử dụng các phương thức khác như lấy giá trị norm L1, L2. Mục đích của quá trình trên là để giảm kích thước ảnh nhưng vẫn lưu giữ được phần nào thông tin ban đầu của ảnh.




*Hình 2.8 Minh họa quá trình pooling trên ảnh 16 pixels*

Nếu ảnh có kích thước n pixels được pooling với kích thước pool là k pixels, sau quá trình ảnh đầu ra sẽ có kích thước xấp xỉ n/k pixels. Như vậy chỉ với kích thước các pool là 2x2 tức 4 pixels ta đã có thể giảm số lượng input ban đầu xuống còn 25%. Đây quả là một con số ấn tượng! Không những thế quá trình nhân chập và quá trình pooling

có thể được diễn ra nhiều lần, điều này mang đến một hiệu quả giảm kích thước đáng kinh ngạc

*c, Lớp fully-connected*

Kết thúc quá trình nhân chập và pooling, toàn bộ các ảnh kết quả sẽ được làm phẳng và nối thành một vector đặc trưng duy nhất. Sau đó vector trên sẽ được đưa vào lớp fully-connected, cách hoạt động ở lớp fully-connected trong CNN cũng tương tự giống như trong mạng Dense. Đều là các lớp chứa các nơ ron được liên kết đầy đủ với lớp trước và lớp sau của nó. Quá trình tính toán đầu ra của một lớp được thực hiện bằng cách nhân vô hướng đầu vào với vector trọng số **w** của lớp đó rồi cộng với một lượng *bias* và đi qua một hàm kích hoạt.

Đó là về mặt toán học, còn cách thức mà lớp fully-connected hiểu dữ liệu được diễn giải như sau. Từ input đầu vào là các đặc trưng gốc cấp thấp, qua mỗi lớp các đặc trưng cấp thấp lại được kết hợp với nhau để xây dựng lên các đặc trưng cấp cao hơn, và các đặc trưng cấp cao ở lớp sau cùng sẽ trực tiếp liên quan đến thuộc tính mục tiêu. Đó cũng chính là lý giải cho từ khóa "học sâu", khi mỗi lần thông tin đi qua một lớp ta lại được một đặc trưng sâu hơn, liên quan hơn tới thuộc tính mục tiêu của đối tượng quan sát.

Đối với bài toán phân lớp, phía ngoài cùng của mạng fully connected cần có một hàm softmax trả về một vector xác suất có số vô hướng bằng với số lớp cần phân. Lớp đầu ra sẽ là lớp có vô hướng tương ứng với nó lớn nhất.

### 2.3.3 Giải thuật học

Toàn bộ nội dung nêu ra ở trên đã mô tả chi tiết cách thức một mạng CNN tính toán đầu ra từ đầu vào dựa trên bộ tham số hiện tại của mạng. Nhưng chúng ta chưa đề cập đến việc làm thế nào để tìm ra những tham số đó. Có một phương pháp chung để làm điều này cho tất cả mạng nơ ron, đó chính là giải thuật học lan truyền ngược. Lan truyền ngược là giải thuật học đóng vai trò quan trọng trong học sâu dùng để huấn luyện các mạng nơ ron, nó giúp cải thiện hiệu suất huấn luyện mô hình học sâu nhanh hơn hàng triệu lần so với cách cài đặt ngây thơ. Không chỉ được sử dụng trong học sâu, lan

truyền ngược còn được biết đến rộng rãi với nhiều cái tên khác nhau trong các lĩnh vực khác như dự báo thời tiết, phân tích số liệu thống kê.

Trước khi đi vào cụ thể cách thức hoạt động của lan truyền ngược, chúng ta cùng nhìn qua một chút về phương pháp tối ưu mang tên gradient descent hay tên tiếng Việt gọi là giảm độ dốc. Đầu tiên ta sẽ cần định nghĩa một hàm nhận đầu vào là bộ tham số hiện tại của mô hình và tập dữ liệu sau đó tính toán cho ra đầu ra là một chỉ số đánh giá càng nhỏ thì mô hình của chúng ta được đánh giá là đã học được càng nhiều. Đối với những bài toán khác nhau sẽ có những hàm mất mát khác nhau.

Đối với bài toán hồi quy ta có hàm:

- Mean Square Error (MSE) / L2 Loss
- Mean Absolute Error (MAE) / L1 Loss

Đối với bài toán phân lớp ta có hàm:

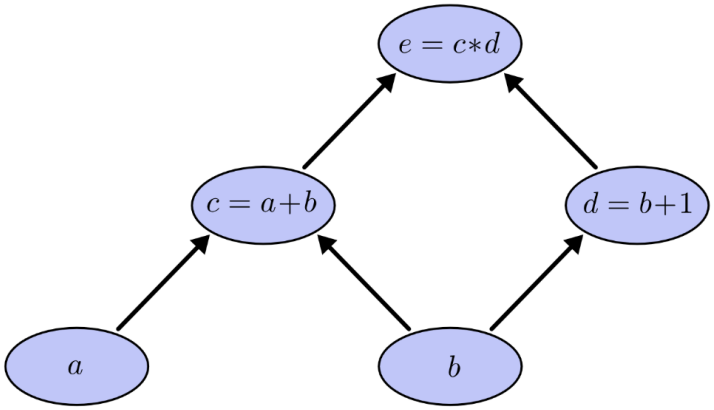
- Binary Cross-Entropy Loss / Log Loss
- Categorical Cross-Entropy Loss
- Hinge Loss

Nhiệm vụ của chúng ta sẽ là đi tìm một bộ tham số sao cho giá trị của hàm mất mát nhỏ nhất. Gradient descent tiếp cận bài toán này theo hướng khá đơn giản, lấy ý tưởng từ trong toán học giải tích muốn đi đến điểm cực tiểu ta chỉ cần đi ngược chiều đạo hàm. Từ nhận định trên ta xây dựng được một giải thuật:

- Tính đạo hàm hàm mất mát theo bộ tham số hiện tại
- Hiệu chỉnh tham số bằng việc trừ đi một lượng bằng đạo hàm
- Duy trì thực hiện các bước trên cho đến khi đạo hàm bằng 0 hoặc rất nhỏ

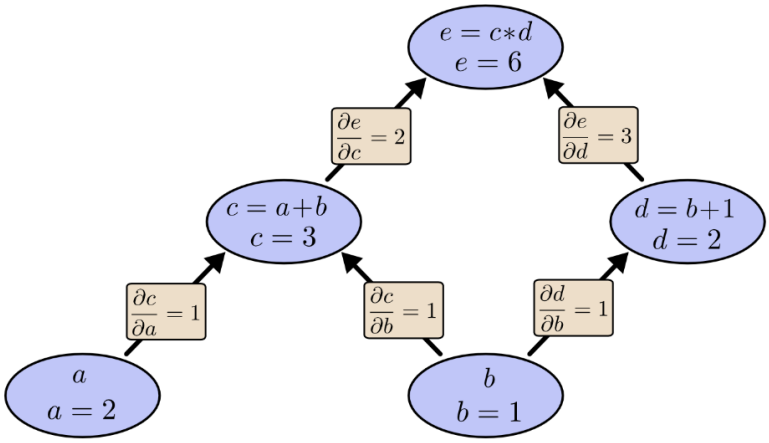
Ý tưởng trên nghe vô cùng đơn giản nhưng lại rất hiệu quả, mặc dù đôi lúc giá trị tham số mà chúng ta tìm ra không phải lúc nào cũng là cực tiểu toàn cục. Tuy nhiên vẫn còn một vấn đề mà chúng ta chưa nhắc tới, đó chính là làm thế nào để tính đạo hàm của hàm mất mát theo các tham số. Thoạt nhìn điều này nghe dễ dàng đối với các mô hình học máy cổ điển khi quá trình tính toán không có quá nhiều bước. Nhưng sang tới mạng nơ ron việc tính toán đạo hàm quả là một cơn ác mộng với các phương thức thủ công.

Để dễ hình dung ta có thể biểu diễn hàm mất mát dưới dạng một đồ thị tính toán, với các nút lá là các biến đầu vào và nút ở đỉnh là giá trị đầu ra.




*Hình 2.9 Đồ thị tính toán với 2 biến đầu vào*

Dễ dàng nhận thấy để tính đạo hàm của đỉnh đồ thị tới một nút, ta chỉ cần sử dụng đến công thức cộng và công thức nhân. Thật vậy, đạo hàm của đỉnh đồ thị theo một nút bằng tổng của tích các đạo hàm theo đường đi đến nút đó.

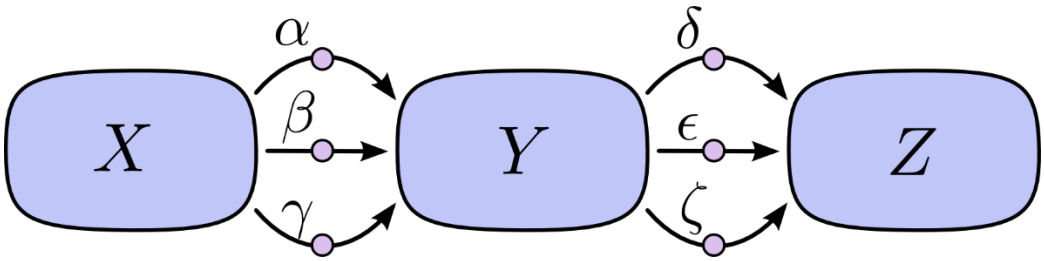



*Hình 2.10 Đồ thị đạo hàm của một nút với các nút lân cận*

Ví dụ trong đồ thị trên đạo hàm của *e* theo *b* có thể được tính theo công thức:

1*2 + 1*3 = 5

Đối với đồ thị tính toán đơn giản, việc tính đạo hàm có thể diễn ra khả thi theo cách trên, nhưng với đồ thị tính toán phức tạp, hiện tượng bùng nổ tổ hợp sẽ xảy ra.




*Hình 2.11 Đồ thị tính toán mô tả sự bùng nổ tổ hợp*

Như đồ thị trên để tính đạo hàm của *Z* theo *X* ta cần tính tổng mọi đường đi từ *X* tới *Z* và có tổng cộng 3*3 đường đi như thế.

Công thức cho đạo hàm sẽ là:

$$\frac{\partial Z}{\partial X} = \alpha\delta+\alpha\epsilon+\alpha\zeta+\beta\delta+\beta\epsilon+\beta\zeta+\gamma\delta+\gamma\epsilon+\gamma\zeta$$

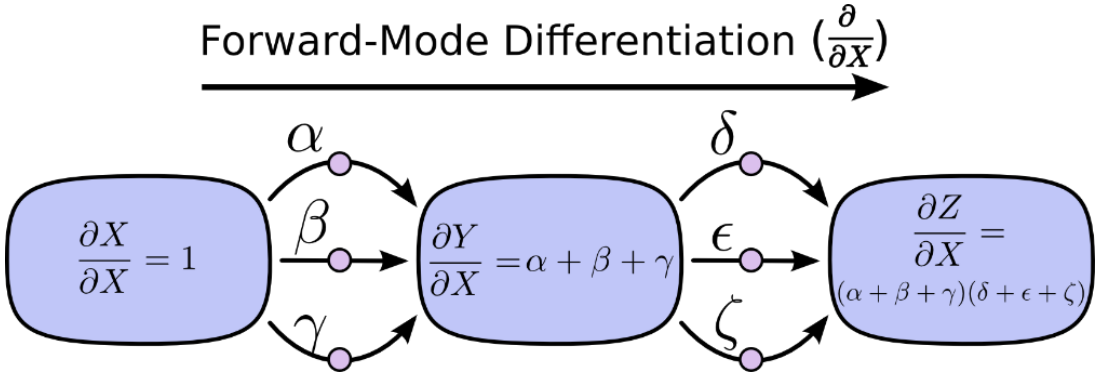
Trong mạng nơ ron, con số trên có thể lớn hơn rất nhiều kéo theo một độ phức tạp tính toán khổng lồ. Để giải quyết vấn đề này thay vì tính tổng từng đường đi một cách ngây thơ, ta có thể đặt nhân tử chung và phân tích chúng thành các thừa số.

Khi đó đạo hàm có thể viết lại dưới dạng:

$$\frac{\partial Z}{\partial X} = (\alpha+\beta+\gamma)(\delta+\epsilon+\zeta)$$

Đây là khởi điểm của khái niệm đạo hàm tiến và đạo hàm ngược, cả 2 phương pháp là giúp cho việc tính đạo hàm tổng các cạnh trở nên hiệu quả hơn. Thay vì lấy tổng của tất cả các cạnh như truyền thống thì chúng sẽ nhóm các đường cùng tới một nút với nhau lại rồi tính tổng. Như vậy, mỗi cạnh chỉ cần duyệt 1 lần duy nhất.

Đạo hàm tiến bắt đầu từ một nút đầu vào và di chuyển dần tới nút cuối cùng cần tính đạo hàm. Tại mỗi nút, nó sẽ tính tổng tất cả các đường đầu vào của nó. Đương nhiên là mỗi đường như vậy thể hiện một cách ảnh hưởng tới nút tương ứng bởi đầu vào. Bằng cách lấy tổng của chúng, ta sẽ thu được tổng tất cả các cách ảnh hưởng tới nút tương ứng bởi tất cả các đầu vào. Hay nói cách khác, nó chính là đạo hàm của nút tương ứng đó.




*Hình 2.12 Tính đạo hàm theo chiều ngược*

Đạo hàm tiến theo dõi một đầu vào ảnh hưởng tới tất cả các nút ra sao, còn đạo hàm ngược thể hiện tất cả các nút ảnh hưởng tới 1 nút đầu vào thế nào. Nói cách khác, đạo hàm tiến áp dụng phép toán $\frac{\partial}{\partial X}$ cho tất cả các nút, còn đạo hàm ngược áp dụng phép toán $\frac{\partial Z}{\partial}$ cho tất cả các nút. Đó cũng là lý do phương pháp này được đặt tên là lan truyền ngược.

Nhờ vào phương pháp trên, ta có thể dễ dàng tính toán đạo hàm của hàm mất mát đối với mọi tham số trong mạng nơ ron từ đó giúp việc sử dụng gradient descent để huấn luyện mạng trở lên đơn giản hơn rất nhiều.

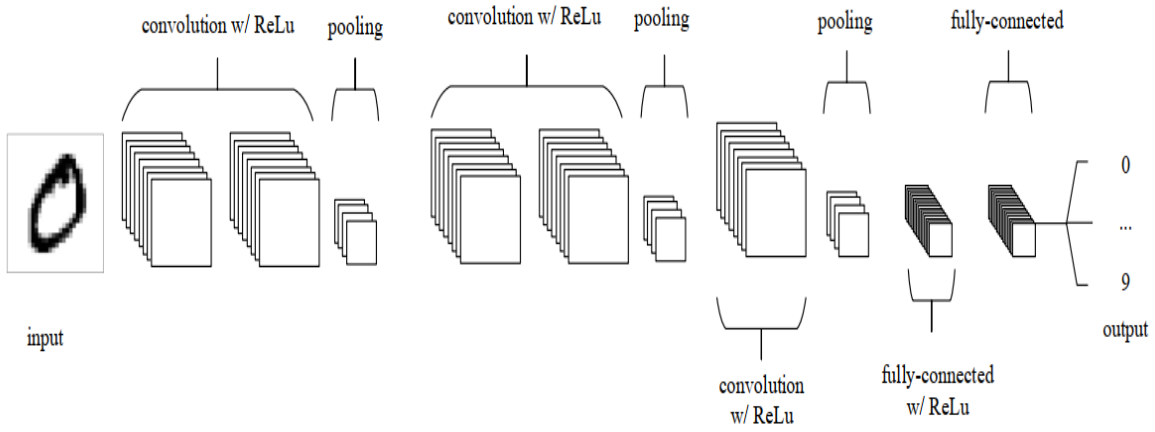
### 2.3.4 Phương pháp xây dựng một mạng CNN

Để xây dựng một kiến trúc CNN hoạt động tốt, chúng ta không thể chỉ đơn giản đặt các lớp xếp chồng lên nhau một cách tùy tiện và mong đợi chúng làm việc đúng đắn. Trong phần này, nhóm sẽ trình bày phương pháp hiệu quả để xây dựng một kiến trúc CNN.

Kiến phổ biến và được sử dụng rộng rãi nhất đó là thực hiện xếp chồng các cặp lớp tích chập và pooling lên nhau sau đó đến với các lớp fully-connected. Đây là kiến trúc cơ bản nhất của CNN, ảnh đầu vào sẽ được xử lý bắt đầu từ các đặc trưng không gian ở lớp tích chập sau đó được giảm kích thước ở lớp pooling, quá trình trên được lặp lại nhiều lần sau đó kết quả sẽ được làm phẳng và đưa vào bộ lớp fully-connected để xây dựng các đặc trưng cao hơn.

Ngoài ra để giảm độ phức tạp tính toán cho lớp tích chập, thay vì sử dụng một mặt nạ có kích thước lớn trong một lớp, ta có thể dùng nhiều mặt nạ có kích thước nhỏ

hơn ở nhiều lớp. Ví dụ như thay vì dùng 1 mặt nạ 7*7 ta có thể dùng 3 mặt nạ 3*3 ở 3 lớp tích chập liên tiếp. Qua mặt nạ thứ nhất tại một nơ ron ta có thể có thông tin trong phạm vi 3*3 nơ ron trước đó, qua mặt nạ số 2 con số trên trở thành 5*5 và qua mặt nạ số 3 ta được 7*7.




*Hình 2.13 Một kiến trúc CNN điển hình*

Một lưu ý khác để đơn giản hóa việc tính toán ở lớp pooling, kích thước của input đầu vào nên là số có thể chia cho 2 nhiều lần ví dụ như 28*28, 32*32, 64*64, 96*96, 128*128…

## 2.4 Tóm tắt chương

- Mạng Dense thuần không phù hợp với bài toán nhận diện chữ số viết tay nói riêng và xử lý ảnh nói chung do yêu cầu số lượng tham số quá lớn
- Mạng CNN ra đời với mục đích giải quyết các hạn chế trên của mạng Dense
- Thành phần chính của một kiến trúc CNN nằm ở lớp tích chập và lớp pooling, trong đó lớp tích chập chứa các mặt nạ mang tham số được dùng để nhân chập với thông tin đầu vào lớp trước đó, lớp pooling có nhiệm vụ giảm kích thước của ảnh
- Phương pháp xây dựng một kiến trúc CNN điển hình là xếp chồng các cặp tích chập và pooling lên nhau để trích chọn đặc trưng không gian, sau đó nối tiếp bởi các lớp Dense

- Trong đề tài này nhóm sẽ thực hiện huấn luyện và đánh giá với 2 kiến trúc tích chập khác nhau do nhóm tự thiết kế

# CHƯƠNG 3 THỰC NGHIỆM

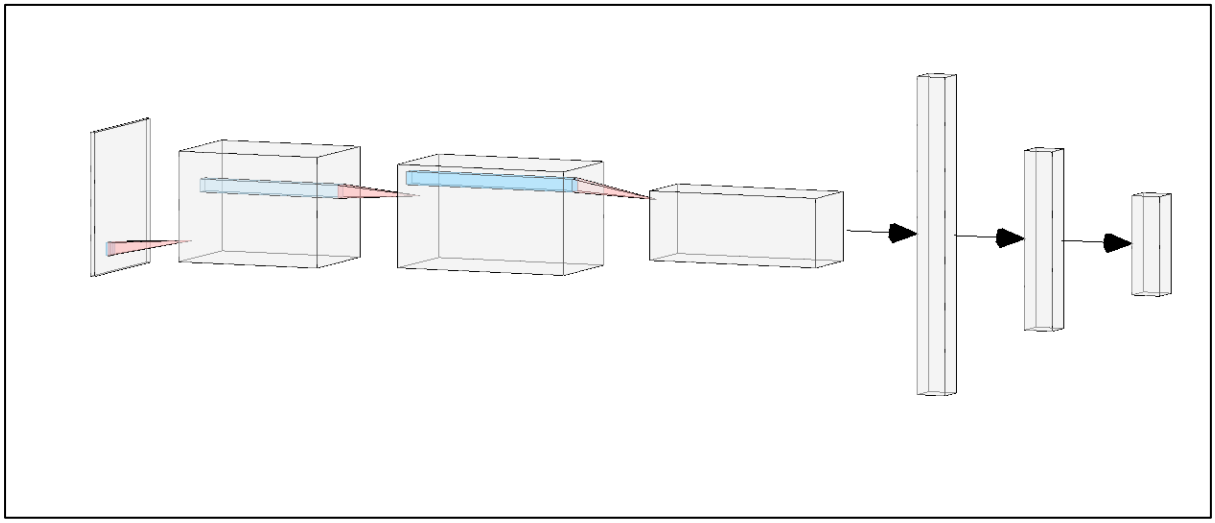
Ở chương thứ ba báo cáo sẽ trình bày quy trình thực nghiệm và kết quả thực nghiệm cho lý thuyết đã được trình bày ở các chương trước.

## 3.1 Phương pháp thực nghiệm

Phương pháp được sử dụng cho quá trình thực nghiệm là mô hình mạng CNN. Trong đề tài sẽ có hai kiến trúc CNN được cài đặt. Mô tả chi tiết về hai kiến trúc được trình bày ở phía dưới.

### 3.1.1 Kiến trúc tích chập thứ nhất

Đầu tiên chúng ta sẽ thử làm việc với kiến trúc cơ bản nhất của CNN. Trong kiến trúc này các cặp lớp tích chập và pooling sẽ được xếp cạnh nhau nhiều lần, tiếp sau đó là lớp fully-connected. Cụ thể kiến trúc được minh họa ở hình và bảng phía dưới:



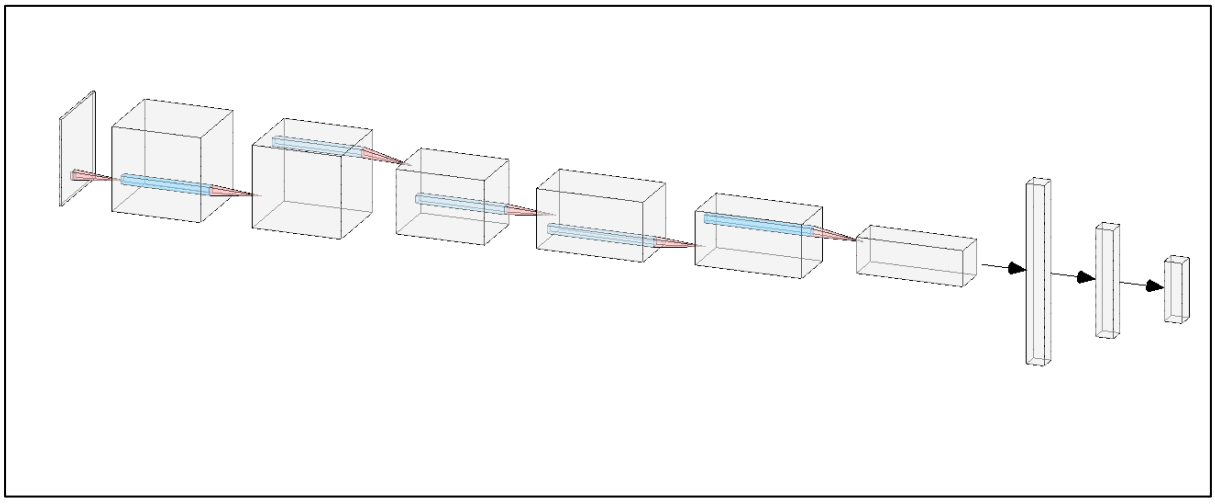
*Hình 3.1 Sơ đồ kiến trúc CNN thứ nhất*

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_2 (Conv2D) | (None, 26, 26, 32) | 320 |
| max_pooling2d_2 (MaxPooling2D) | (None, 13, 13, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 11, 11, 64) | 18,496 |
| max_pooling2d_3 (MaxPooling2D) | (None, 5, 5, 64) | 0 |
| flatten_1 (Flatten) | (None, 1600) | 0 |
| dense_2 (Dense) | (None, 64) | 102,464 |
| dense_3 (Dense) | (None, 10) | 650 |

*Bảng 3.1 Chi thiết kích thước đầu ra tại mỗi lớp trong kiến trúc thứ nhất*

### 3.1.2 Kiến trúc tích chập thứ hai

Sang kiến trúc thứ hai chúng ta hãy thử cải tiến một chút, thay vì đặt duy nhất một lớp tích chập cạnh một lớp pooling, ta sẽ thử đặt nhiều lớp tích chập cạnh nhau rồi mới đến lớp pooling để gia tăng phạm vi quan sát của các mặt nạ. Chi tiết kiến trúc được mô tả ở ảnh và bảng phía dưới.



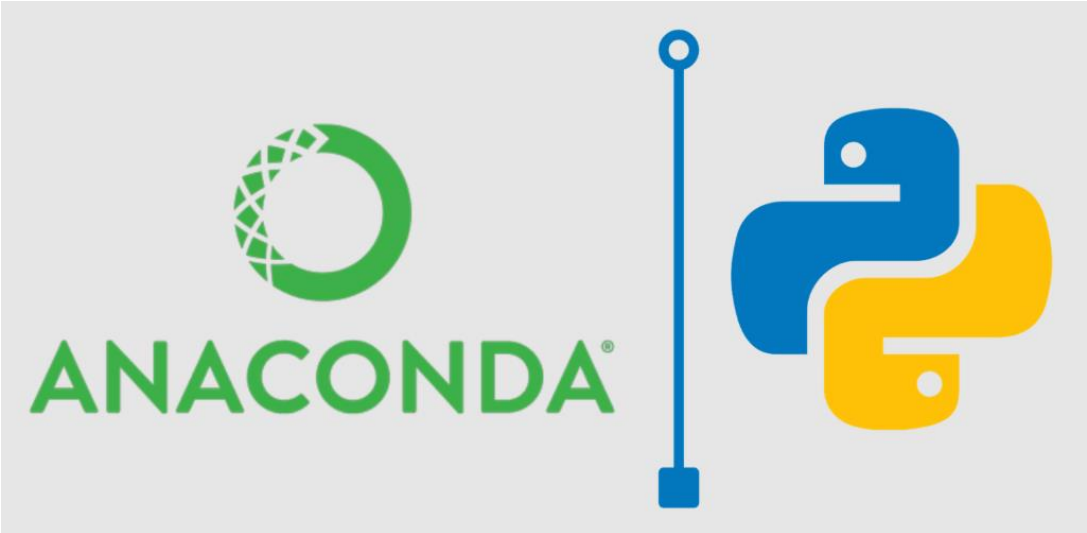
*Hình 3.2 Sơ đồ kiến trúc CNN thứ hai*

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_12 (Conv2D) | (None, 26, 26, 32) | 320 |
| conv2d_13 (Conv2D) | (None, 24, 24, 32) | 9,248 |
| max_pooling2d_6 (MaxPooling2D) | (None, 12, 12, 32) | 0 |
| conv2d_14 (Conv2D) | (None, 10, 10, 64) | 18,496 |
| conv2d_15 (Conv2D) | (None, 8, 8, 64) | 36,928 |
| max_pooling2d_7 (MaxPooling2D) | (None, 4, 4, 64) | 0 |
| flatten_3 (Flatten) | (None, 1024) | 0 |
| dense_6 (Dense) | (None, 64) | 65,600 |
| dense_7 (Dense) | (None, 10) | 650 |

*Bảng 3.2 Chi thiết kích thước đầu ra tại mỗi lớp trong kiến trúc thứ hai*

## 3.2 Môi trường thực nghiệm

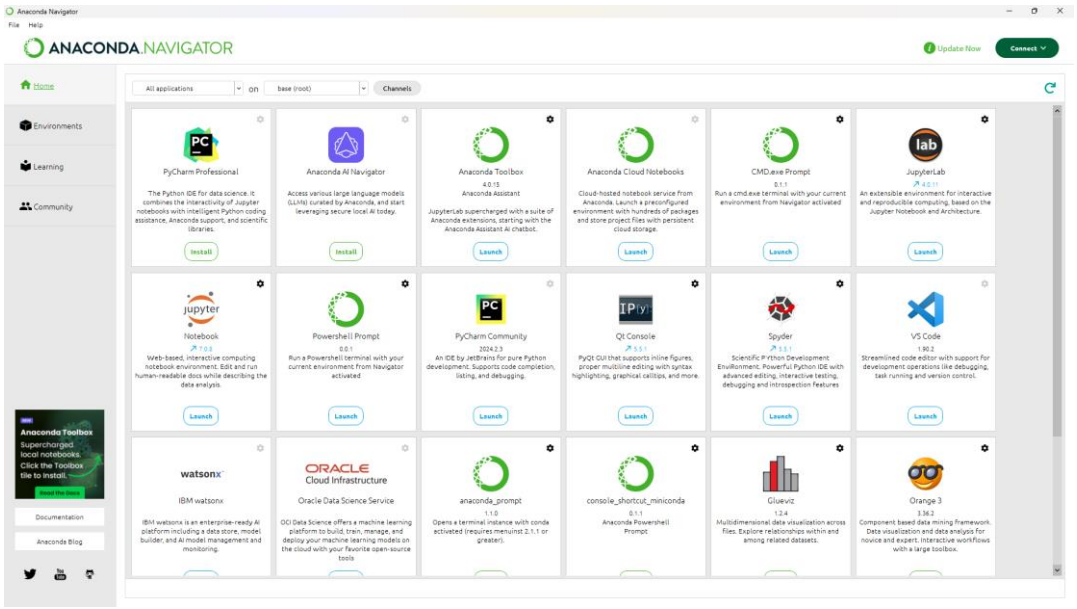
Để thiết lập môi trường huấn luyện cho đề tài, nhóm sử dụng trình quản lý gói Anaconda. Đây là một trình quản lý gói phổ biến cho ngôn ngữ Python và R được sử dụng rộng rãi trong cộng đồng khoa học dữ liệu. Anaconda mang lại nhiều lợi ích vượt trội so với các công cụ cùng loại khác. Trong đó có thể kể đến như tính dễ sử dụng, tính độc lập với môi trường phần cứng và tính hỗ trợ đa nền tảng.




*Hình 3.3 Logo Anaconda và Python*

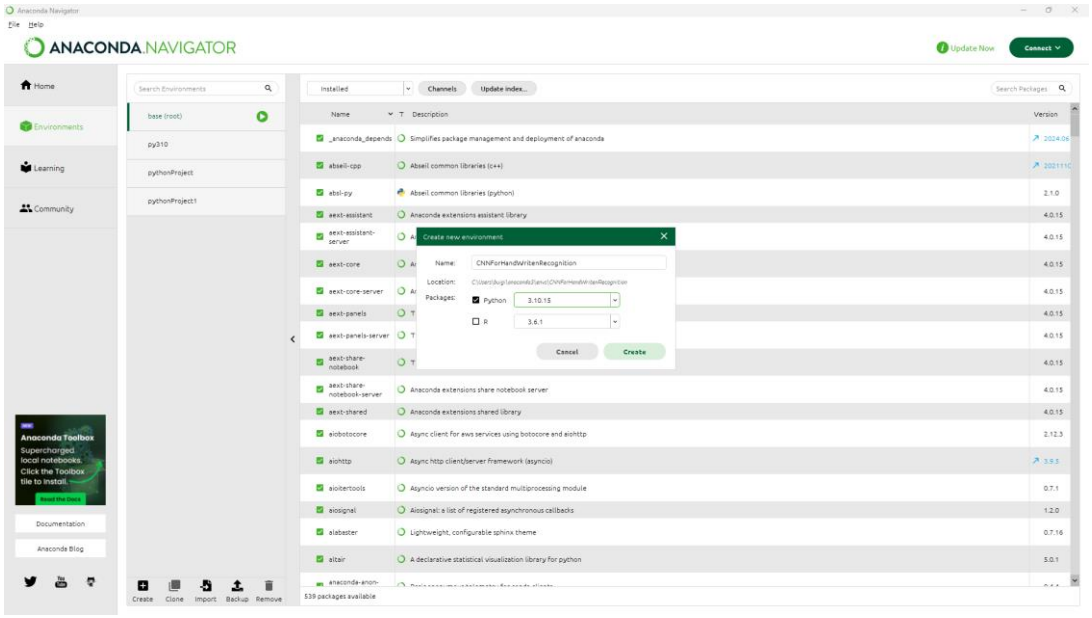
Để cài đặt Anaconda chúng ta cần truy cập vào trang chủ [www.anaconda.com](http://www.anaconda.com) sau đó điều hướng đến mục download và nhấn tải xuống cho Windows. Sau khi tải về chúng ta sử dụng file thực thi đã tải đề cài đặt.

Khi kết thúc cài đặt chúng ta có thể mở Anaconda Navigator để thiết lập môi trường thông qua cửa sổ giao diện.




*Hình 3.4 Cửa sổ Anaconda Navigator*

Để tạo môi trường cho việc cài đặt chúng ta vào mục "Environment" chọn "Create", nhập tên cho môi trường mới và chọn phiên bản Python sau đó nhấn "Create"




*Hình 3.5 Tạo môi trường mới với Anaconda*

Tại cửa sổ của môi trường vừa tạo, chúng ta tìm kiếm và tích vào các gói cần cài đặt sau đó nhấn apply, hệ thống sẽ tự động tải về và cài đặt. Một khi quá trình cài đặt hoàn tất, ta có thể mở Jupyter Notebook và bắt đầu soạn thảo mã.

## 3.3 Dữ liệu thực nghiệm

Để phục vụ cho pha huấn luyện của mô hình, nhóm sử dụng bộ dữ liệu MNIST. Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology) là một tập dữ liệu tiêu chuẩn được sử dụng rộng rãi trong nghiên cứu và thực hành về học máy, đặc biệt trong các bài toán nhận dạng chữ số viết tay. MNIST được xây dựng nhằm cung cấp một tập dữ liệu sạch, giúp các nhà nghiên cứu đánh giá hiệu quả của các thuật toán nhận dạng mẫu.

Bộ dữ liệu bao gồm tổng cộng 70.000 hình ảnh ở định dạng đa cấp xám của các chữ số từ 0 đến 9, được viết tay bởi nhiều người khác nhau. Mỗi hình ảnh có kích thước 28x28 pixel, tương ứng với một ma trận chứa 784 giá trị mức xám từ 0 (đen hoàn toàn) đến 255 (trắng hoàn toàn). Đi kèm với mỗi hình ảnh là một nhãn, đại diện cho chữ số thực mà hình ảnh đó biểu diễn. MNIST được chia thành hai tập con là tập huấn luyện phục vụ cho việc huấn luyện và tập kiểm tra phục vụ cho việc đánh giá mô hình nhận dạng mẫu. Trong đó tập huấn luyện chứa 60.000 mẫu còn tập kiểm tra chứa 10.000 mẫu. Mỗi hình ảnh trong MNIST đều đã được tiền xử lý và chuẩn hóa về kích thước 28x28 pixel. Các chữ số đã được căn chỉnh vào trung tâm khung hình, giúp giảm sự phức tạp khi xử lý đồng thời các chữ số được viết tay bởi nhiều người khác nhau, bao gồm nhiều kiểu viết và độ đậm nhạt khác nhau, giúp phản ánh tốt hơn sự đa dạng thực tế. Nhờ vào những đặc trưng đó MNIST đã trở thành một bộ dữ liệu tham chiếu chuẩn, giúp so sánh hiệu quả giữa các thuật toán khác nhau.

## 3.4 Quy trình thực nghiệm

### 3.4.1 Chuẩn bị dữ liệu và tiền xử lý

Để tối ưu hóa hiệu quả trong quá trình huấn luyện mô hình, dữ liệu đầu vào sẽ được chuẩn hóa bằng cách chuẩn hóa các giá trị về phạm vi từ 0 đến 1. Bước chuẩn hóa này không chỉ giúp đảm bảo tính đồng nhất của các đặc trưng mà còn cải thiện sự ổn định và tăng tốc độ hội tụ của thuật toán học. Đồng thời, để tăng cường sự đa dạng và

phong phú của tập dữ liệu huấn luyện, chúng ta cũng sẽ áp dụng các kỹ thuật tăng cường dữ liệu. Các kỹ thuật này bao gồm các phép biến đổi như xoay góc, thay đổi tỷ lệ phóng to hoặc thu nhỏ, và dịch chuyển vị trí. Những thao tác này giúp mô hình trở nên linh hoạt hơn, nâng cao khả năng khái quát hóa và khả năng xử lý các biến dạng hoặc nhiễu tiềm tàng trong dữ liệu thực tế.

### 3.4.2 Trích chọn đặc trưng

Trong khâu này, để làm nổi bật các đặc trưng vốn có của chữ số trên ảnh gốc, chúng ta áp dụng các phép biến đổi hình thái học trong xử lý ảnh. Những phép biến đổi này bao gồm phép co giãn và các thao tác đóng cũng như mở. Các thao tác này không chỉ giúp làm rõ các đường nét và chi tiết quan trọng mà còn hỗ trợ loại bỏ nhiễu không mong muốn, từ đó tăng cường khả năng nhận diện và phân biệt đặc trưng của mô hình trong quá trình học máy.

### 3.4.3 Huấn luyện mô hình

Tiếp theo, quá trình huấn luyện mô hình sẽ được triển khai với việc áp dụng thuật toán tối ưu hóa Adam. Đây là một phương pháp hiện đại và hiệu quả cao, kết hợp giữa ưu điểm của các kỹ thuật Momentum và RMSProp.

### 3.4.4 Đánh giá và nhận xét mô hình trên tập dữ liệu huấn luyện

Sau khi huấn luyện mô hình sẽ được đánh giá bằng nhiều các thông số khác nhau bao gồm: Accuracy, Precision, Recall. Đồng thời cũng kết hợp với các công cụ thống kê như Confusion Matrix. Sau cùng từ kết quả sẽ đưa ra nhận xét cuối cùng.

## 3.5 Kết quả thực nghiệm

### 3.5.1 Cài đặt mô hình và huấn luyện

Đầu tiên ta sẽ tiến hành chuẩn bị dữ liệu cho khâu huấn luyện

```
1. import tensorflow as tf
2. from tensorflow.keras.preprocessing.image import ImageDataGenerator
3. import numpy as np
4.
5. # Tải dữ liệu MNIST
6. (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
7.
8. # Thêm trục kênh màu và chuẩn hóa dữ liệu
9. x_train = x_train.reshape((-1, 28, 28, 1)).astype('float32') / 255.0
10. x_test = x_test.reshape((-1, 28, 28, 1)).astype('float32') / 255.0
```

```
11.
12. # Chuyển nhãn sang dạng one-hot encoding
13. y_train = tf.keras.utils.to_categorical(y_train, 10)
14. y_test = tf.keras.utils.to_categorical(y_test, 10)
15.
16. # Tăng cường dữ liệu
17. datagen = ImageDataGenerator(
18.     rotation_range=10,     # Xoay tối đa 10 độ
19.     width_shift_range=0.1, # Dịch ngang tối đa 10%
20.     height_shift_range=0.1, # Dịch dọc tối đa 10%
21.     zoom_range=0.1,        # Phóng to/thu nhỏ tối đa 10%
22.     shear_range=0.1        # Biến dạng xiên
23. )
24.
25. # Tạo generator
26. datagen.fit(x_train)
27.
28. # Kiểm tra mẫu
29. for x_batch, y_batch in datagen.flow(x_train, y_train, batch_size=32):
30.     print(f"Batch shape: {x_batch.shape}")
31.     break
```

Tiếp đến chúng ta tiến hành cài đặt kiến trúc CNN thứ nhất

```
 1. import tensorflow as tf
 2. from tensorflow.keras import layers, models
 3.
 4. # Khởi tạo mô hình CNN
 5. model = models.Sequential()
 6.
 7. # Lớp Conv2D + MaxPooling đầu tiên
 8. model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
 9. model.add(layers.MaxPooling2D((2, 2)))
10.
11. # Lớp Conv2D + MaxPooling thứ hai
12. model.add(layers.Conv2D(64, (3, 3), activation='relu'))
13. model.add(layers.MaxPooling2D((2, 2)))
14.
15. # Flatten để chuyển đổi dữ liệu sang dạng 1D
16. model.add(layers.Flatten())
17.
18. # Lớp Dense ẩn
19. model.add(layers.Dense(64, activation='relu'))
20.
21. # Lớp Dense đầu ra
22. model.add(layers.Dense(10, activation='softmax'))
23.
24. # Tóm tắt mô hình
25. model.summary()
```

Ta thực hiện tương tự với kiến trúc CNN thứ hai

```
 1. import tensorflow as tf
 2. from tensorflow.keras import layers, models
 3.
 4. # Khởi tạo mô hình CNN
 5. model = models.Sequential()
 6.
 7. # Lớp Conv2D đầu tiên
 8. model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
 9.
10. # Lớp Conv2D thứ hai
11. model.add(layers.Conv2D(32, (3, 3), activation='relu'))
12.
13. # Lớp MaxPooling
14. model.add(layers.MaxPooling2D((2, 2)))
15.
16. # Lớp Conv2D thứ ba
```

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Lớp Conv2D thứ tư
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Lớp MaxPooling
model.add(layers.MaxPooling2D((2, 2)))

# Flatten để chuyển đổi dữ liệu sang dạng 1D
model.add(layers.Flatten())

# Lớp Dense ẩn
model.add(layers.Dense(64, activation='relu'))

# Lớp Dense đầu ra
model.add(layers.Dense(10, activation='softmax'))

# Tóm tắt mô hình
model.summary()
```

Ta thực hiện biên dịch mô hình và huấn luyện

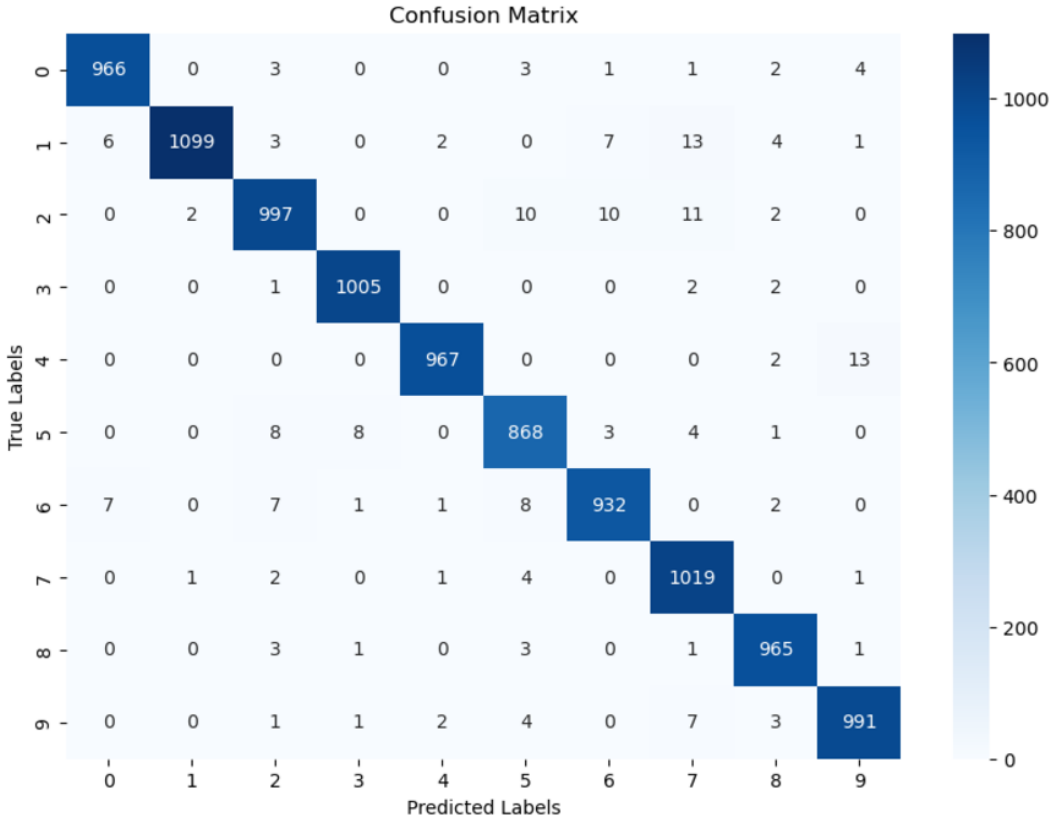
```
# Biên dịch mô hình
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Huấn luyện mô hình
history = model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test,
y_test))
```

### 3.5.2 Kết quả huấn luyện

*a, Kiến trúc CNN thứ nhất*

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.986721 | 0.985714 | 0.986217 | 980 |
| 1 | 0.997278 | 0.968282 | 0.982566 | 1135 |
| 2 | 0.972683 | 0.966085 | 0.969373 | 1032 |
| 3 | 0.989173 | 0.99505 | 0.992103 | 1010 |
| 4 | 0.993834 | 0.984725 | 0.989258 | 982 |

*Bảng 3.3 Thống kê các thông số đánh giá qua từng epoch*




*Hình 3.6 Confusion matrix cho kiến trúc CNN thứ nhất*

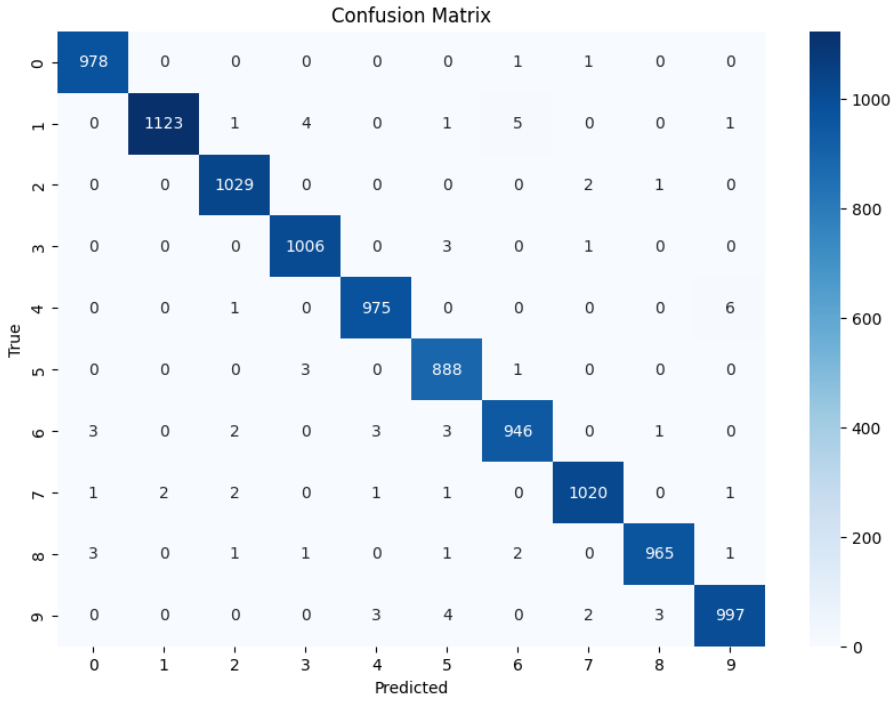
| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.986721 | 0.985714 | 0.986217 | 980 |
| 1 | 0.997278 | 0.968282 | 0.982566 | 1135 |
| 2 | 0.972683 | 0.966085 | 0.969373 | 1032 |
| 3 | 0.989173 | 0.99505 | 0.992103 | 1010 |
| 4 | 0.993834 | 0.984725 | 0.989258 | 982 |
| 5 | 0.964444 | 0.973094 | 0.96875 | 892 |
| 6 | 0.977964 | 0.97286 | 0.975406 | 958 |
| 7 | 0.963138 | 0.991245 | 0.976989 | 1028 |
| 8 | 0.981689 | 0.99076 | 0.986203 | 974 |
| 9 | 0.980218 | 0.982161 | 0.981188 | 1009 |
| **accuracy** | 0.9809 | | | 10000 |
| **macro avg** | 0.9807142 | 0.9809976 | 0.9808054 | |
| **weighted avg** | 0.98102 | 0.9809 | 0.9809063 | |

*Bảng 3.4 Tổng hợp kết quả huấn luyện kiến trúc CNN thứ nhất*

*b, Kiến trúc CNN thứ hai*

| Epoch | Train Accuracy | Val Accuracy | Train Loss | Val Loss |
|---|---|---|---|---|
| 1 | 0.765550017 | 0.950800002 | 0.70197928 | 0.147419229 |
| 2 | 0.919749975 | 0.974399984 | 0.255915254 | 0.08639843 |
| 3 | 0.940866649 | 0.97329998 | 0.191395789 | 0.080199681 |
| 4 | 0.948700011 | 0.969500005 | 0.164692283 | 0.094386868 |
| 5 | 0.953599989 | 0.977299988 | 0.148174763 | 0.069767587 |
| 6 | 0.959299982 | 0.980700016 | 0.131633699 | 0.058677655 |
| 7 | 0.961266696 | 0.981400013 | 0.125166848 | 0.059435748 |
| 8 | 0.963866651 | 0.981000006 | 0.115966715 | 0.05857807 |
| 9 | 0.964766681 | 0.98360002 | 0.114428632 | 0.048542548 |
| 10 | 0.966183305 | 0.984499991 | 0.108298399 | 0.047877919 |
| 11 | 0.967133343 | 0.970700026 | 0.104202695 | 0.093462929 |
| 12 | 0.969333351 | 0.983699977 | 0.097557604 | 0.051458869 |
| 13 | 0.97056669 | 0.984099984 | 0.095911592 | 0.0495799 |
| 14 | 0.970916688 | 0.984499991 | 0.094156675 | 0.047684617 |
| 15 | 0.970533311 | 0.986400008 | 0.094601423 | 0.047756504 |
| 16 | 0.971883357 | 0.986999989 | 0.0893666 | 0.041712806 |
| 17 | 0.972199976 | 0.986199975 | 0.089038759 | 0.040869541 |
| 18 | 0.972450018 | 0.983399987 | 0.088055216 | 0.050045107 |
| 19 | 0.973950028 | 0.987600029 | 0.082623929 | 0.038986385 |
| 20 | 0.973866642 | 0.983699977 | 0.083267614 | 0.050548505 |

*Bảng 3.5 Thống kê các thông số đánh giá qua từng epoch*




*Hình 3.7 Confusion matrix cho kiến trúc CNN thứ hai*

| Label | precision | recall | f1-score | support |
|---|---|---|---|---|
| **0** | 0.992864424 | 0.993877551 | 0.993370729 | 980 |
| **1** | 0.997333333 | 0.988546256 | 0.992920354 | 1135 |
| **2** | 0.964566929 | 0.949612403 | 0.99703125 | 1032 |
| **3** | 0.993089832 | 0.996039604 | 0.994562531 | 1010 |
| **4** | 0.982914573 | 0.99592668 | 0.989377845 | 982 |
| **5** | 0.956375839 | 0.958520179 | 0.957446809 | 892 |
| **6** | 0.974973931 | 0.975991649 | 0.975482525 | 958 |
| **7** | 0.98267565 | 0.993190661 | 0.987905177 | 1028 |
| **8** | 0.992798354 | 0.990759754 | 0.991778006 | 974 |
| **9** | 0.995029821 | 0.992071358 | 0.993548387 | 1009 |
| **accuracy** | 0.9837 | | | 10000 |
| **macro avg** | 0.983262269 | 0.98345361 | 0.983342361 | |
| **weighted avg** | 0.983692991 | 0.9837 | 0.98368048 | |

*Bảng 3.6 Tổng hợp kết quả huấn luyện kiến trúc CNN thứ hai*

## 3.6 Nhận xét

Sau khi huấn luyện kiến trúc đầu tiên, kết quả dự đoán trên tập kiểm tra đạt mức độ chính xác lên tới 99%. Phân tích chi tiết hơn qua confusion matrix, ta có thể nhận thấy rằng các nhãn thực tế và nhãn dự đoán sai thường có sự tương đồng nhất định về hình thức chữ viết. Ví dụ điển hình là sự nhầm lẫn giữa số 7 và số 1, trong đó có đến 7 mẫu thực tế có nhãn là số 7, nhưng lại bị dự đoán nhầm thành số 1. Sự nhầm lẫn này có thể giải thích một cách hợp lý bởi đặc điểm hình dạng chung của hai chữ số này, cả hai đều có một nét gạch dọc và một nét ngang, với sự khác biệt duy nhất là nét ngang của số 1 có xu hướng nghiêng hơn so với số 7.

Khi chuyển sang kiến trúc thứ hai, ta nhận thấy một sự khác biệt nhẹ nhưng quan trọng, sai số dự đoán đã giảm đi so với kiến trúc đầu tiên. Điều này có thể lý giải do số lượng tham số của kiến trúc thứ hai lớn hơn, giúp mô hình có khả năng học được những sự khác biệt tinh vi hơn giữa các hình dạng của các chữ số - điều mà kiến trúc đầu tiên chưa thể nhận diện được.

Tổng quan về kết quả dự đoán từ cả hai kiến trúc cho thấy tính hiệu quả vượt trội của mô hình mạng CNN trong việc nhận diện ảnh, đặc biệt là trong bài toán nhận diện chữ số viết tay. Mô hình không chỉ đạt được độ chính xác cao mà còn có khả năng phân biệt được những chi tiết nhỏ giữa các chữ số có hình dạng tương tự nhau, chứng tỏ sức mạnh và tính linh hoạt của mạng tích chập trong các bài toán nhận diện hình ảnh phức tạp.

## 3.7 Tóm tắt chương

- Phương pháp thực nghiệm được sử dụng là mạng CNN
- Trong nghiên cứu sử dụng hai kiến trúc CNN khác nhau
- Bộ dữ liệu sử dụng cho pha huấn luyện trong quá trình thực nghiệm là bộ MNIST
- Thực nghiệm được thực hiện trên môi trường hệ sinh thái của ngôn ngữ lập trình Python và sử dụng các thư viện liên quan
- Quy trình thực nghiệm bao gồm bốn bước:
    - Chuẩn bị dữ liệu và tiền xử lý
    - Trích chọn đặc trưng

- Huấn luyện mô hình
- Nhận xét và đánh giá mô hình trên tập dữ liệu thử nghiệm

- Quá trình thực nghiệm cho ra kết quả ấn tượng cho thấy tính hiệu quả của mô hình mạng CNN với bài toán nhận diện chữ số viết tay

# CHƯƠNG 4 XÂY DỰNG CHƯƠNG TRÌNH

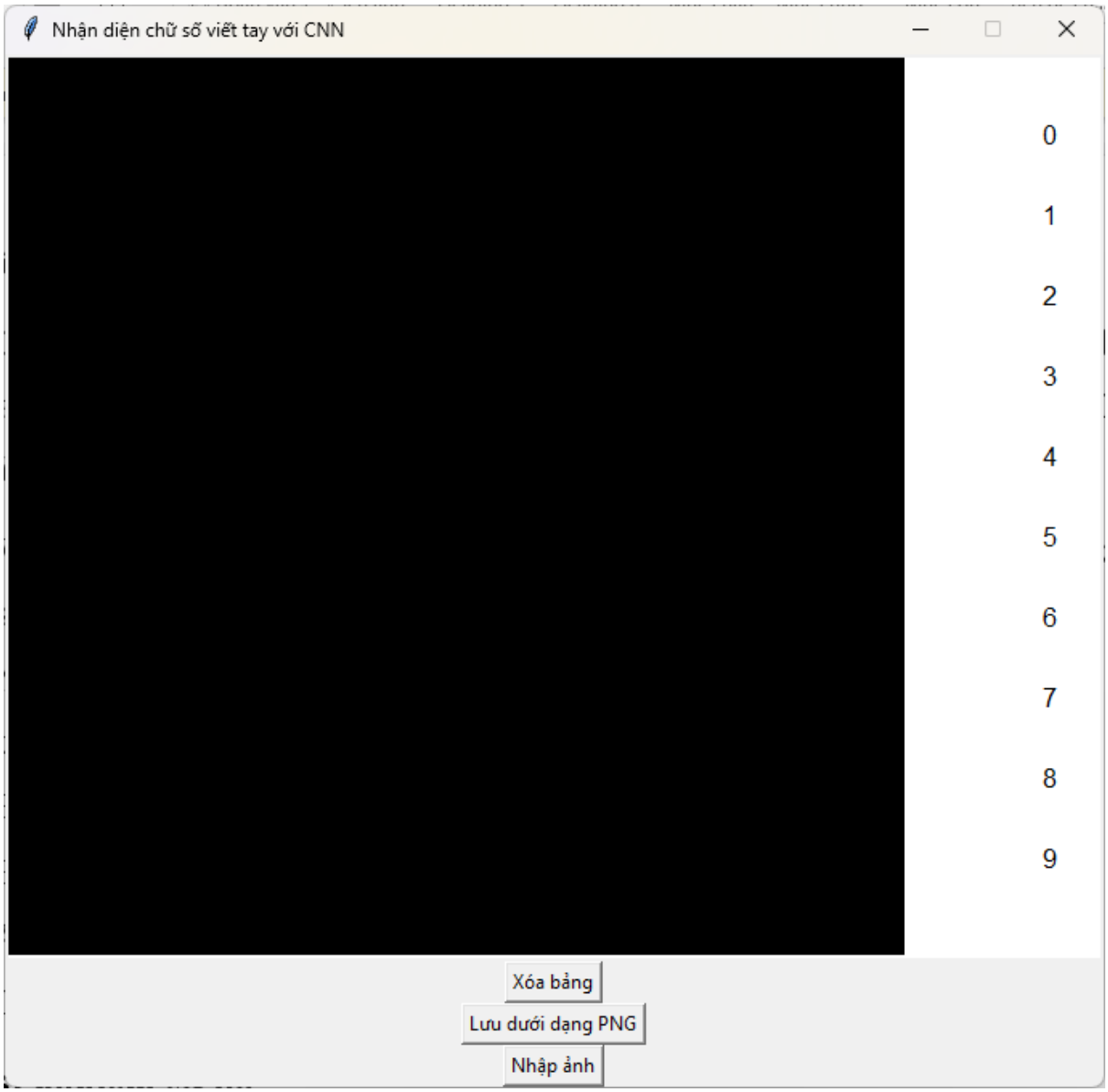
Trong chương này nhóm sẽ trình bày sơ lược về chương trình DEMO được tích hợp mô-đun đã được xây dựng trong chương trước. Nội dung chương không đi cụ thể vào các phân tích, thiết kế cũng như cài đặt mà chỉ dừng lại ở việc mô tả các chức năng và đánh giá hoạt động. Chi tiết hơn về mã nguồn được trình bày trong phần phụ lục của tài liệu.

## 4.1 Mô tả chức năng chung

Chương trình "Nhận diện chữ số viết tay với CNN" được xây dựng với mục đích trực quan hóa kết quả mô hình được huấn luyện trước đó. Chương trình bao gồm những chức năng sau:

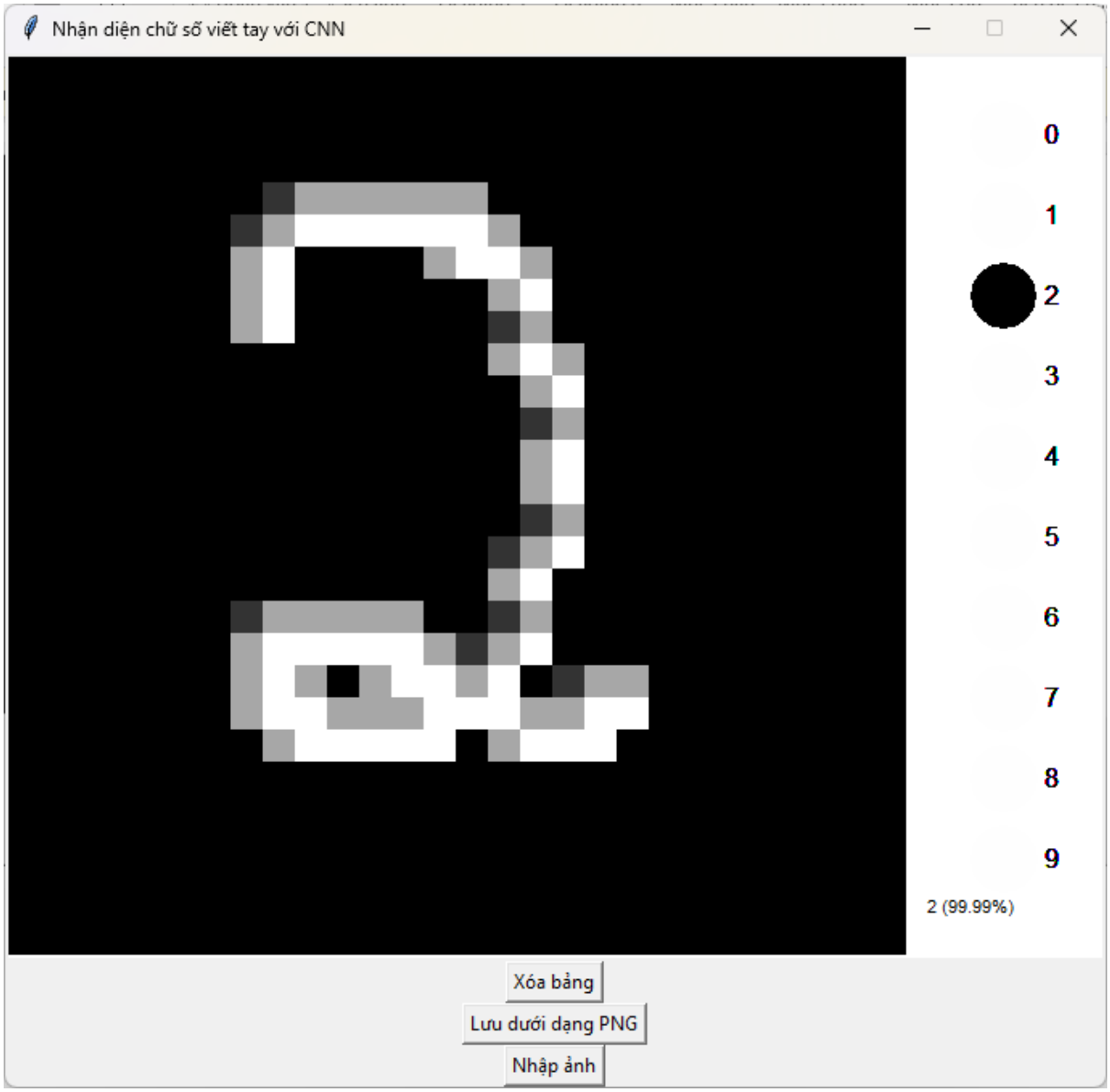
- Vẽ hình dạng chữ số bất kỳ trên ma trận đa cấp xám 28x28 hoặc nhập ảnh chữ số đa cấp xám có kích thước 28x28 từ tệp tin
- Nhận diện nhãn số tương ứng với chữ số được vẽ hoặc được trích xuất từ tệp tin, cập nhật thay đổi theo thời gian thực
- Hiển thị các chấm xám thể hiện cho xác suất của các nhãn từ 0 đến 9
- Hiển thị nhãn có xác suất lớn nhất và giá trị xác suất tương ứng
- Xóa nội dung trên bảng hiện thời
- Lưu nội dung trên bảng hiện thời ở định dạng ảnh có kích thước 28x28

## 4.2 Trải nghiệm sơ bộ




*Hình 4.1 Giao diện khi bắt đầu khởi chạy chương trình*


*Hình 4.2 Nét vẽ ngẫu nhiên khiến xác suất dự đoán phân tán ra các nhãn*




*Hình 4.3 Số 2 được vẽ và được dự đoán chính xác nhãn*

## 4.3 Nhận xét

Qua thử nghiệm với chương trình, kết quả của quá trình huấn luyện đã được trực quan hóa. Nhờ sự cập nhật thời gian thực trong kết quả dự đoán ta có thể thấy được sự ảnh hưởng trong thay đổi của các nét vẽ tới các nhãn đầu ra.

## 4.4 Tóm tắt chương

- Chương trình "Nhận diện chữ số viết tay với mạng CNN" cho phép người dùng, vẽ hoặc nhập các chữ số từ ảnh, chương trình sẽ đưa ra dự đoán về nhãn tương ứng

- Qua làm việc với chương trình ta thấy được sự nhạy bén của mạng CNN trong việc hiểu được những thay đổi về đường nét

# KẾT LUẬN

Qua quá trình nghiên cứu và thực hiện đồ án "Nhận dạng chữ số viết tay sử dụng mạng nơ-ron tích chập", cá nhân em đã đạt được những kết quả đáng khích lệ, đồng thời tích lũy thêm nhiều kiến thức chuyên sâu về lĩnh vực xử lý ảnh và trí tuệ nhân tạo. Cụ thể, em đã:

- Tìm hiểu và nắm vững các kiến thức nền tảng về xử lý ảnh, mạng nơ-ron nhân tạo cũng như các giải thuật học để tối ưu chúng
- Cải thiện được các kỹ năng mềm trong soạn thảo văn bản, xây dựng tài liệu kỹ thuật và phi kỹ thuật
- Cải thiện khả năng lập trình với các thư viện khoa học dữ liệu trong hệ sinh thái Python

Thông qua việc triển khai và thực nghiệm mô hình, em nhận thấy hiệu quả vượt trội của mạng tích chập trong việc giải quyết bài toán nhận dạng chữ số viết tay. Đề tài này có thể là bước đầu trong việc mở rộng hướng nghiên cứu trong lĩnh vực xử lý ảnh, dưới đây em xin đề xuất một số ý tưởng mở rộng

- Mở rộng tập dữ liệu huấn luyện, không chỉ giới hạn ở chữ số viết tay mà còn là toàn bộ ký tự chữ
- Kết hợp kiến trúc trên vào các kiến trúc khác đa dạng hóa khả năng giải quyết vấn đề
- Sử dụng kiến trúc trên làm module nhận diện đơn ký tự cho bộ nhận diện ký tự quang học (OCR)

# TÀI LIỆU THAM KHẢO

[1]. An Introduction to Convolutional Neural Networks - Keiron O'Shea 1 and Ryan Nash 2.

[2]. Introduction to Convolutional Neural Networks - Jianxin Wu _ May 1, 2017.

[3]. Keras for Beginners: Implementing a Convolutional Neural Network – Victor Zhou.

[4]. Calculus on Computational Graphs: Backpropagation – August 31, 2015 – colah's blog.

[5]. Chollet, François, "Deep Learning with Python", Manning Publications, 2018.

[6]. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, "Deep Learning", MIT Press, 2016.

[7]. He, Kaiming, et al., "Deep Residual Learning for Image Recognition", CVPR, 2016.

[8]. Kingma, Diederik P., and Jimmy Ba, "Adam: A Method for Stochastic Optimization", ICLR, 2015.

[9]. Krizhevsky, Alex, et al., "ImageNet Classification with Deep Convolutional Neural Networks", NIPS, 2012.

[10]. LeCun, Yann, et al., "Gradient-Based Learning Applied to Document Recognition", Proceedings of the IEEE, 1998.

[11]. Rumelhart, David E., et al., "Learning Representations by Back-Propagating Errors", Nature, 1986.

[12]. Sutskever, Ilya, et al., "Sequence to Sequence Learning with Neural Networks", NIPS, 2014.

[13]. Szegedy, Christian, et al., "Going Deeper with Convolutions", CVPR, 2015.

# PHỤ LỤC

## Mã nguồn chương trình

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import numpy as np
import tensorflow as tf

GRID_SIZE = 28
PIXEL_SIZE = 20

pixel_matrix = [[0 for _ in range(GRID_SIZE)] for _ in range(GRID_SIZE)]

model = tf.keras.models.load_model("CNN.h5")

def preprocess_input(matrix):
    array = np.array(matrix, dtype=np.float32) / 255.0
    array = np.expand_dims(array, axis=0)
    array = np.expand_dims(array, axis=-1)
    return array

def predict_and_display():
    input_data = preprocess_input(pixel_matrix)
    predictions = model.predict(input_data)[0]

    for i, prob in enumerate(predictions):
        gray_value = int((1 - prob) * 255)
        color = f"#{gray_value:02x}{gray_value:02x}{gray_value:02x}"

        canvas.create_oval(
            GRID_SIZE * PIXEL_SIZE + 40, i * 50 + 30,
            GRID_SIZE * PIXEL_SIZE + 80, i * 50 + 70,
            fill=color, outline=color
        )

        canvas.create_text(
            GRID_SIZE * PIXEL_SIZE + 90, i * 50 + 50,
            text=str(i), font=("Helvetica", 12), fill="black"
        )

    predicted_label = np.argmax(predictions)
    predicted_prob = predictions[predicted_label]

    canvas.delete("prediction_label")
    canvas.create_text(
        GRID_SIZE * PIXEL_SIZE + 40, 10 * 50 + 30,
        text=f"{predicted_label} ({predicted_prob * 100:.2f}%)",
        font=("Helvetica", 8), fill="black", tags="prediction_label"
    )

def draw_pixel(event):
    x, y = event.x // PIXEL_SIZE, event.y // PIXEL_SIZE
    intensity = 255
    weight_matrix = [
        [0.2, 0.658, 0.0],
        [0.658, 1.0, 0.0],
        [0.0, 0.0, 0.0],
    ]
    if 0 <= x < GRID_SIZE and 0 <= y < GRID_SIZE:
        for i in range(-1, 2):
            for j in range(-1, 2):
                nx, ny = x + i, y + j
```

```
                if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                    new_intensity = int(intensity * weight_matrix[i + 1][j + 1])
                    pixel_matrix[ny][nx] = max(pixel_matrix[ny][nx], new_intensity)
                    color =
f"#{pixel_matrix[ny][nx]:02x}{pixel_matrix[ny][nx]:02x}{pixel_matrix[ny][nx]:02x}"
                    canvas.create_rectangle(
                        nx * PIXEL_SIZE, ny * PIXEL_SIZE,
                        (nx + 1) * PIXEL_SIZE, (ny + 1) * PIXEL_SIZE,
                        fill=color, outline=""
                    )

        predict_and_display()

def clear_canvas():
    canvas.delete("all")
    for i in range(GRID_SIZE):
        for j in range(GRID_SIZE):
            pixel_matrix[i][j] = 0
            canvas.create_rectangle(
                j * PIXEL_SIZE, i * PIXEL_SIZE,
                (j + 1) * PIXEL_SIZE, (i + 1) * PIXEL_SIZE,
                fill="black", outline=""
            )

    for i in range(10):
        canvas.create_oval(
            GRID_SIZE * PIXEL_SIZE + 40, i * 50 + 30,
            GRID_SIZE * PIXEL_SIZE + 80, i * 50 + 70,
            fill="white", outline="white"
        )
        canvas.create_text(
            GRID_SIZE * PIXEL_SIZE + 90, i * 50 + 50,
            text=str(i), font=("Helvetica", 12), fill="black"
        )



def save_as_png():
    file_path = filedialog.asksaveasfilename(
        defaultextension=".png",
        filetypes=[("PNG files", "*.png"), ("All files", "*.*")]
    )
    if file_path:
        img = Image.new("L", (GRID_SIZE, GRID_SIZE))
        for i in range(GRID_SIZE):
            for j in range(GRID_SIZE):
                img.putpixel((j, i), pixel_matrix[i][j])
        img.save(file_path)

def import_image():
    file_path = filedialog.askopenfilename(
        filetypes=[("Image files", "*.png;*.jpg;*.jpeg;*.bmp"), ("All files", "*.*")]
    )
    if file_path:
        img = Image.open(file_path).convert("L")
        img = img.resize((GRID_SIZE, GRID_SIZE))
        img_data = np.array(img)

        for i in range(GRID_SIZE):
            for j in range(GRID_SIZE):
                pixel_matrix[i][j] = img_data[i, j]
                color = f"#{img_data[i, j]:02x}{img_data[i, j]:02x}{img_data[i, j]:02x}"
                canvas.create_rectangle(
                    j * PIXEL_SIZE, i * PIXEL_SIZE,
                    (j + 1) * PIXEL_SIZE, (i + 1) * PIXEL_SIZE,
                    fill=color, outline=""
                )

        predict_and_display()
```

```
127.
128. root = tk.Tk()
129. root.title("Nhận diện chữ số viết tay với CNN")
130. root.resizable(False, False)
131.
132. canvas = tk.Canvas(root, width=GRID_SIZE * PIXEL_SIZE + 120, height=GRID_SIZE * PIXEL_SIZE,
bg="white")
133. canvas.pack()
134.
135. for i in range(GRID_SIZE):
136.     for j in range(GRID_SIZE):
137.         canvas.create_rectangle(
138.             j * PIXEL_SIZE, i * PIXEL_SIZE,
139.             (j + 1) * PIXEL_SIZE, (i + 1) * PIXEL_SIZE,
140.             fill="black", outline=""
141.         )
142.
143. clear_button = tk.Button(root, text="Xóa bảng", command=clear_canvas)
144. clear_button.pack()
145.
146. save_button = tk.Button(root, text="Lưu dưới dạng PNG", command=save_as_png)
147. save_button.pack()
148.
149. import_button = tk.Button(root, text="Nhập ảnh", command=import_image)
150. import_button.pack()
151.
152. canvas.bind("<B1-Motion>", draw_pixel)
153. canvas.bind("<Button-1>", draw_pixel)
154.
155. root.mainloop()
```