HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1
o0o

BÁO CÁO ĐỀ TÀI

Hệ thống Nhận diện chữ số và hình dạng đơn giản viết tay với mạng CNN

MÔN: XỬ LÝ ẢNH

Giảng viên: Phạm Hoàng Việt
Số thứ tự nhóm: 20

Ngô Văn Bộ MSSV: B22DCCN085
Nguyễn Việt Anh MSSV: B22DCCN037

HÀ NỘI, 11/2025

Mục lục

Danh mục hình ảnh

Danh mục bảng biểu

MỞ ĐẦU

0.1 Lý do chọn đề tài

Trong kỷ nguyên của cuộc Cách mạng Công nghiệp 4.0, **Trí tuệ Nhân tạo (AI)** và các lĩnh vực con như **Học máy (Machine Learning)** và **Học sâu (Deep Learning)** đã và đang tạo ra những bước đột phá mạnh mẽ, len lỏi vào mọi khía cạnh của đời sống và sản xuất. Một trong những ứng dụng quan trọng và có tầm ảnh hưởng lớn nhất của học sâu chính là **Thị giác máy tính (Computer Vision)** – lĩnh vực nghiên cứu giúp máy móc có khả năng "nhìn" và "hiểu" thế giới hình ảnh như con người.

Các bài toán nhận diện đối tượng, đặc biệt là nhận diện chữ viết tay và các hình dạng cơ bản, là những bài toán **nền tảng** và mang tính lịch sử của thị giác máy tính.

Tầm quan trọng của nhận diện chữ viết tay (MNIST): Bộ dữ liệu **MNIST (Modified National Institute of Standards and Technology)** được coi là một trong những bộ dữ liệu **"kinh điển"**, là phép thử tiêu chuẩn (**benchmark**) cho bất kỳ mô hình nhận diện hình ảnh nào. Khả năng nhận diện chính xác chữ số viết tay là yếu tố then chốt trong nhiều ứng dụng thực tiễn như tự động hóa việc đọc **mã bưu chính**, **xử lý séc ngân hàng**, **số hóa tài liệu** lưu trữ, và **nhận dạng biển số xe**.

Ý nghĩa của nhận diện hình dạng cơ bản: Tương tự, khả năng phân biệt các hình dạng cơ bản (như hình tròn, hình vuông, hình tam giác) là bước đệm thiết yếu cho các hệ thống thị giác phức tạp hơn. Ứng dụng của nó trải dài từ việc **phân tích bản vẽ kỹ thuật**, **robot học** (giúp robot nhận diện và tương tác với vật thể), cho đến các **phần mềm thiết kế đồ họa** thông minh.

Về mặt kỹ thuật, **Mạng nơ-ron tích chập (Convolutional Neural Network - CNN)** đã chứng minh được hiệu quả vượt trội so với các phương pháp truyền thống trong việc trích xuất đặc trưng và phân loại hình ảnh. Tuy nhiên, khi xây dựng các mô hình CNN ngày càng sâu để giải quyết các vấn đề phức tạp, các nhà nghiên cứu đã gặp phải vấn đề **suy giảm độ dốc (vanishing gradient)**. Kiến trúc **Mạng dư (Residual Network - ResNet)** ra đời như một giải pháp đột phá, cho phép huấn luyện các mạng nơ-ron cực sâu (lên đến hàng trăm, thậm chí hàng nghìn lớp) mà vẫn duy trì được hiệu suất cao và khả năng hội tụ tốt.

Xuất phát từ những lý do trên, việc nghiên cứu và áp dụng kiến trúc ResNet tiên tiến để giải quyết các bài toán nhận diện cơ bản nhưng mang tính nền tảng như MNIST và hình dạng viết tay là vô cùng cần thiết. Đề tài không chỉ giúp củng cố kiến thức về các kỹ thuật học sâu hiện đại mà còn tạo ra một mô hình hiệu quả, có khả năng ứng dụng thực tế cao. Chính vì vậy, chúng tôi quyết định chọn đề tài: **"Nhận diện chữ viết tay MNIST và hình dạng cơ bản viết tay sử dụng**

mạng CNN dựa trên ResNet".

0.2 Mục tiêu nghiên cứu

Trên cơ sở lý do chọn đề tài đã trình bày, đề tài này tập trung vào các mục tiêu chính sau đây:

0.2.1 Mục tiêu về lý thuyết

- Nghiên cứu và hệ thống hóa kiến thức cơ bản về xử lý ảnh kỹ thuật số, thị giác máy tính và học sâu (Deep Learning).
- Tìm hiểu sâu về kiến trúc, nguyên lý hoạt động và các thành phần cốt lõi của **Mạng nơ-ron tích chập (CNN)**, vốn là kiến trúc tiêu chuẩn cho các bài toán nhận diện hình ảnh.
- Phân tích chi tiết kiến trúc **Mạng dư (ResNet)**, làm rõ ưu điểm của các **khối dư (residual blocks)** trong việc giải quyết vấn đề suy giảm độ dốc và cho phép huấn luyện các mạng rất sâu.
- Nghiên cứu đặc điểm của bộ dữ liệu MNIST và các phương pháp tiền xử lý phổ biến áp dụng cho nó.
- Tìm hiểu các phương pháp để xây dựng và chuẩn hóa bộ dữ liệu cho bài toán nhận diện hình dạng cơ bản viết tay.

0.2.2 Mục tiêu về thực nghiệm

- Xây dựng và tiền xử lý thành công hai bộ dữ liệu: (1) Bộ dữ liệu MNIST tiêu chuẩn và (2) Bộ dữ liệu tự tạo về các hình dạng cơ bản viết tay (hình chữ nhật, hình tròn, hình tam giác).
- Thiết kế và triển khai cụ thể mô hình CNN dựa trên kiến trúc ResNet (ví dụ: **ResNet18**, **ResNet34** hoặc một phiên bản tùy chỉnh) để giải quyết hai bài toán nhận diện một cách độc lập hoặc song song.
- Huấn luyện thành công các mô hình đã thiết kế, áp dụng các kỹ thuật tối ưu hóa (**Optimizer**) và các phương pháp chống quá khớp (**overfitting**) hiệu quả như tăng cường dữ liệu (**data augmentation**) và **dropout**.
- Đánh giá chi tiết hiệu suất của các mô hình thu được bằng các chỉ số đo lường phổ biến trong học máy (**Accuracy**, **Precision**, **Recall**, **F1-Score**) trên các tập dữ liệu kiểm thử.
- Phân tích các trường hợp mô hình dự đoán sai, so sánh kết quả và rút ra nhận xét về khả năng của mô hình trên hai bộ dữ liệu có đặc điểm khác nhau.

0.2.3 Mục tiêu tổng quát

Mục tiêu cuối cùng là xây dựng được một hệ thống hoàn chỉnh, ứng dụng kiến trúc **ResNet** để nhận diện chính xác chữ số viết tay **MNIST** và các hình dạng cơ bản, làm tiền đề cho việc phát triển các ứng dụng thị giác máy tính phức tạp hơn trong tương lai.

0.3 Phạm vi và giới hạn của đề tài

Để đảm bảo tính khả thi và tập trung vào mục tiêu nghiên cứu, đề tài được thực hiện trong phạm vi và tồn tại một số giới hạn như sau:

0.3.1 Phạm vi nghiên cứu

Về dữ liệu:

- Đối với nhận diện chữ số: Đề tài chỉ tập trung sử dụng bộ dữ liệu **MNIST** tiêu chuẩn (ảnh thang xám, kích thước 28x28) cho 10 lớp (từ 0 đến 9).
- Đối với nhận diện hình dạng: Đề tài giới hạn ở việc nhận diện ba hình dạng cơ bản viết tay là hình chữ nhật, hình tròn và hình tam giác. Dữ liệu này sẽ được thu thập hoặc tạo mới và tiền xử lý để phù hợp với mô hình.

Về mô hình:

- Đề tài tập trung nghiên cứu và triển khai mô hình **Mạng nơ-ron tích chập (CNN)** có kiến trúc dựa trên **ResNet** (ví dụ: sử dụng các khối dư - residual blocks).
- Chúng tôi có thể sử dụng một kiến trúc ResNet tiêu chuẩn (như **ResNet18**, **ResNet34**) hoặc một phiên bản tùy chỉnh (**custom model**) được xây dựng dựa trên nguyên lý của ResNet để phù hợp với tính chất đơn giản của dữ liệu.

Về nhiệm vụ:

- Đề tài chỉ giải quyết bài toán **phân loại hình ảnh (image classification)**, tức là xác định một hình ảnh đầu vào thuộc lớp chữ số nào (0-9) hoặc hình dạng nào (chữ nhật, tròn, tam giác).
- Đề tài không đi sâu vào các bài toán phức tạp hơn như **phát hiện đối tượng (object detection** - định vị hình ảnh trong một bức ảnh lớn) hay **phân đoạn hình ảnh (segmentation)**.

0.3.2 Giới hạn của đề tài

- Giới hạn về dữ liệu hình dạng: Bộ dữ liệu hình dạng cơ bản viết tay được tự xây dựng hoặc thu thập có thể bị hạn chế về số lượng và tính đa dạng (ví dụ: ít người viết, ít kiểu viết, ít sự biến đổi về độ dày nét, độ nghiêng...) so với bộ dữ liệu MNIST đã được chuẩn hóa. Điều này có thể ảnh hưởng đến khả năng tổng quát hóa (**generalization**) của mô hình nhận diện hình dạng.

- Giới hạn về tính phức tạp của dữ liệu: Cả hai bộ dữ liệu (MNIST và hình dạng) đều là các ảnh tương đối đơn giản (ảnh thang xám, độ phân giải thấp, nền sạch). Do đó, kết quả và hiệu suất của mô hình ResNet trên các tập dữ liệu này có thể không phản ánh đầy đủ hiệu năng của nó khi áp dụng vào các bài toán thị giác máy tính trong thế giới thực với ảnh màu, độ phân giải cao, nhiều nhiễu và bối cảnh phức tạp.

- Giới hạn về tài nguyên tính toán: Quá trình huấn luyện các mô hình học sâu, đặc biệt là ResNet (dù là phiên bản nhỏ), đòi hỏi tài nguyên phần cứng (**GPU/TPU**). Những hạn chế về mặt phần cứng có thể ảnh hưởng đến khả năng thử nghiệm các kiến trúc lớn hơn, thời gian huấn luyện hoặc mức độ tinh chỉnh siêu tham số (**hyperparameter tuning**).

0.4 Phương pháp nghiên cứu

Để đạt được các mục tiêu đã đề ra, đề tài sử dụng kết hợp các phương pháp nghiên cứu chính sau đây:

0.4.1 Phương pháp nghiên cứu lý thuyết

- Tiến hành thu thập, đọc hiểu, phân tích và tổng hợp các tài liệu, giáo trình, bài báo khoa học, và tài liệu kỹ thuật (documentation) liên quan đến các lĩnh vực: xử lý ảnh, học sâu, mạng nơ-ron tích chập (**CNN**) và đặc biệt là kiến trúc mạng **ResNet**.

- Nghiên cứu các bộ dữ liệu tiêu chuẩn (**benchmark**) như **MNIST** để hiểu rõ đặc điểm và các kỹ thuật tiền xử lý phổ biến.

0.4.2 Phương pháp nghiên cứu thực nghiệm

Đây là phương pháp chủ đạo của đề tài. Quá trình thực nghiệm bao gồm:

- **Thu thập và Xây dựng dữ liệu:** Sử dụng bộ dữ liệu MNIST có sẵn và tiến hành xây dựng một bộ dữ liệu mới cho các hình dạng viết tay (hình chữ nhật, tròn, tam giác).

- **Tiền xử lý dữ liệu:** Áp dụng các kỹ thuật chuẩn hóa, biến đổi kích thước (resizing), và tăng cường dữ liệu (**data augmentation**) để cải thiện chất lượng dữ liệu đầu vào và tăng khả năng tổng quát hóa của mô hình.

- **Thiết kế và Xây dựng mô hình:** Dựa trên cơ sở lý thuyết về ResNet, tiến hành thiết kế và cài đặt mô hình CNN phù hợp với đặc thù của hai bài toán.

- **Huấn luyện và Tinh chỉnh:** Chạy thực nghiệm huấn luyện mô hình với các bộ tham số khác nhau (ví dụ: learning rate, batch size, số epochs) để tìm ra bộ tham số tối ưu.

0.4.3 Phương pháp phân tích, so sánh và đánh giá

- Sử dụng các chỉ số đo lường (**metrics**) định lượng phổ biến trong học máy như **Accuracy** (Độ chính xác), **Precision** (Độ chuẩn), **Recall** (Độ phủ) và **F1-Score** để đánh giá hiệu suất của mô hình trên tập dữ liệu kiểm thử (**test set**).
- Sử dụng **ma trận nhầm lẫn (Confusion Matrix)** để phân tích chi tiết các trường hợp mô hình dự đoán đúng/sai cho từng lớp.
- Thực hiện phân tích định tính bằng cách trực quan hóa và kiểm tra các mẫu dự đoán sai (ví dụ minh họa) để hiểu rõ hạn chế của mô hình.
- So sánh hiệu suất và kết quả đạt được giữa hai nhiệm vụ (nhận diện MNIST và nhận diện hình dạng) để rút ra các kết luận và nhận định.

0.5 Cấu trúc báo cáo

Ngoài phần Mở đầu (trình bày lý do chọn đề tài, mục tiêu, phạm vi và phương pháp nghiên cứu) và phần Kết luận (tổng kết các kết quả, nêu hạn chế và đề xuất hướng phát triển), nội dung chính của báo cáo được tổ chức thành 4 chương:

- **Chương 1. Tổng quan về nhận diện chữ viết tay và hình dạng:** Trình bày các cơ sở lý thuyết liên quan đến đề tài, bao gồm các khái niệm về xử lý ảnh, nhận diện đối tượng, giới thiệu chi tiết về bộ dữ liệu MNIST, và đặc biệt là kiến trúc, nguyên lý hoạt động của Mạng nơ-ron tích chập (CNN) và Mạng dư (ResNet).
- **Chương 2. Phương pháp xây dựng hệ thống:** Mô tả chi tiết quy trình thực hiện đề tài, từ bước thu thập và tiền xử lý cho cả hai bộ dữ liệu (MNIST và hình dạng viết tay), đến việc thiết kế kiến trúc mô hình CNN dựa trên ResNet cụ thể, và các phương pháp luận dùng để huấn luyện và đánh giá mô hình (hàm mất mát, optimizer, các kỹ thuật chống quá khớp).
- **Chương 3. Thực nghiệm và kết quả:** Trình bày môi trường và các công cụ được sử dụng để thực nghiệm. Chương này sẽ báo cáo chi tiết các kết quả đạt được (độ chính xác, precision, recall...) trên cả hai nhiệm vụ nhận diện. Đồng thời, tiến hành phân tích các lỗi sai, đưa ra các ví dụ minh họa và so sánh hiệu suất giữa hai mô hình.
- **Chương 4. Kết luận và hướng phát triển:** Tóm tắt lại toàn bộ kết quả nghiên cứu đã đạt được so với mục tiêu đề ra. Đánh giá những đóng góp, ưu điểm và chỉ ra những hạn chế còn tồn tại của hệ thống. Cuối cùng, đề xuất một số hướng nghiên cứu tiếp theo để cải thiện và mở rộng đề tài.

Chương 1

TỔNG QUAN VỀ NHẬN DIỆN CHỮ VIẾT TAY VÀ HÌNH DẠNG

1.1 Giới thiệu về xử lý ảnh và nhận diện đối tượng

1.1.1 Xử lý ảnh (Image Processing)

Xử lý ảnh kỹ thuật số là một lĩnh vực của khoa học máy tính và kỹ thuật, tập trung vào việc sử dụng các thuật toán máy tính để thực hiện các thao tác trên hình ảnh kỹ thuật số. Mục tiêu chính của xử lý ảnh là cải thiện chất lượng hình ảnh (ví dụ: tăng độ tương phản, giảm nhiễu) hoặc để trích xuất những thông tin hữu ích, đặc trưng từ hình ảnh đó, phục vụ cho các tác vụ phân tích và hiểu biết ở cấp độ cao hơn.

Một quy trình xử lý ảnh cơ bản thường bao gồm các bước:

- **Thu nhận ảnh (Image Acquisition):** Chuyển đổi tín hiệu từ thế giới thực (ánh sáng) thành dạng dữ liệu số (pixel) thông qua các thiết bị như máy ảnh, máy quét.
- **Tiền xử lý (Preprocessing):** Cải thiện ảnh để phù hợp với các bước xử lý sau. Các kỹ thuật phổ biến bao gồm: **chuẩn hóa (normalization)**, thay đổi kích thước (resizing), **lọc nhiễu (noise filtering)**, và chuyển đổi không gian màu (ví dụ: từ ảnh màu RGB sang ảnh thang xám).
- **Trích xuất đặc trưng (Feature Extraction):** Giảm tải tính toán và tập trung vào các thông tin cốt lõi bằng cách xác định các đặc điểm quan trọng của ảnh, như cạnh (edges), góc (corners), kết cấu (texture).

1.1.2 Nhận diện đối tượng (Object Recognition)

Nhận diện đối tượng là một nhiệm vụ cốt lõi và đầy thách thức trong lĩnh vực **Thị giác máy tính (Computer Vision)**. Thị giác máy tính là một nhánh của Trí tuệ Nhân tạo, với mục tiêu giúp máy móc có khả năng "nhìn" và "hiểu" nội dung của hình ảnh hoặc video giống như con người.

Cụ thể, nhận diện đối tượng là quá trình xác định sự hiện diện và **phân loại (classification)** một hoặc nhiều đối tượng (ví dụ: con người, xe cộ, chữ số, hình dạng) trong một bức ảnh. Đây là một bài toán phân loại, trong đó hệ thống sẽ gán một nhãn (label) cho một vùng ảnh hoặc toàn bộ bức ảnh.

Trước đây, các phương pháp nhận diện đối tượng truyền thống phụ thuộc nhiều vào việc trích xuất đặc trưng thủ công (**hand-crafted features**) như SIFT, SURF, HOG, sau đó sử dụng các mô hình máy học cổ điển (như **SVM**, **K-Nearest Neighbors**) để phân loại. Tuy nhiên, phương pháp này gặp nhiều hạn chế khi đối mặt với sự đa dạng và phức tạp của dữ liệu trong thế giới thực.

Sự ra đời của **Học sâu (Deep Learning)**, đặc biệt là **Mạng nơ-ron tích chập (CNN)**, đã tạo ra một cuộc cách mạng. CNN có khả năng tự động học các **đặc trưng (features)** trực tiếp từ dữ liệu ảnh thô qua nhiều lớp (layers), từ các đặc trưng đơn giản (như cạnh, góc) ở các lớp đầu đến các đặc trưng phức tạp (như bộ phận, vật thể) ở các lớp sâu hơn.

Trong khuôn khổ của đề tài này, "xử lý ảnh" được áp dụng để chuẩn bị và chuẩn hóa dữ liệu đầu vào (ảnh chữ số và hình dạng), và "nhận diện đối tượng" chính là nhiệm vụ phân loại các ảnh đã xử lý đó vào các lớp tương ứng (0-9 hoặc hình chữ nhật, tròn, tam giác) bằng mô hình CNN dựa trên **ResNet**.

1.2 Bộ dữ liệu MNIST: Đặc điểm và ứng dụng

1.2.1 Đặc điểm của MNIST

MNIST (viết tắt của Modified National Institute of Standards and Technology) là một trong những bộ dữ liệu nổi tiếng và được sử dụng rộng rãi nhất trong lĩnh vực học máy và thị giác máy tính. Nó được coi là bộ dữ liệu **"Hello, World!"** cho bất kỳ ai bắt đầu nghiên cứu về nhận diện hình ảnh.

Bộ dữ liệu này là một tập hợp lớn các hình ảnh chữ số viết tay, được trích xuất từ các tài liệu của Cục Điều tra Dân số Hoa Kỳ và học sinh trung học.

Các đặc điểm kỹ thuật chính của MNIST bao gồm:

- **Nội dung:** Gồm 10 lớp (classes), tương ứng với 10 chữ số viết tay từ 0 đến 9.
- **Quy mô:** Toàn bộ dữ liệu được chia thành hai tập:
  - **Tập huấn luyện (Training set):** Chứa 60.000 hình ảnh.
  - **Tập kiểm thử (Test set):** Chứa 10.000 hình ảnh.

  Sự phân chia rõ ràng này giúp các nhà nghiên cứu có một phương pháp chuẩn hóa để huấn luyện và so sánh hiệu suất của các mô hình.
- **Định dạng ảnh:**
  - Mỗi hình ảnh là ảnh thang xám (grayscale).
  - Kích thước ảnh đã được chuẩn hóa về 28x28 pixel.
  - Các chữ số đã được căn giữa (centering) và chuẩn hóa về kích thước trong một ô 20x20 pixel, sau đó được đặt vào giữa ảnh 28x28. Điều này giúp loại bỏ các biến thể về vị trí và kích thước, cho phép mô hình tập trung vào đặc trưng của nét chữ.

1.2.2 Ứng dụng và tầm quan trọng

Mặc dù MNIST được coi là một bài toán tương đối **"dễ"** đối với các mô hình học sâu hiện đại (nhiều mô hình có thể đạt độ chính xác trên 99%), nó vẫn giữ một vai trò quan trọng:

- **Chuẩn mực so sánh (Benchmark):** MNIST là tiêu chuẩn vàng để đánh giá và so sánh hiệu suất ban đầu của các thuật toán phân loại hình ảnh mới. Một mô hình mới thường phải chứng minh được hiệu quả của nó trên MNIST trước khi chuyển sang các bộ dữ liệu phức tạp hơn (như CIFAR-10 hay ImageNet).
- **Giáo dục và Nghiên cứu:** Do tính đơn giản, kích thước vừa phải và đã được tiền xử lý sạch sẽ, MNIST là bộ dữ liệu lý tưởng cho mục đích giảng dạy, học tập và thử nghiệm nhanh các ý tưởng mới về kiến trúc mạng nơ-ron mà không tốn quá nhiều tài nguyên tính toán.
- **Nền tảng cho ứng dụng thực tế:** Việc giải quyết thành công bài toán MNIST đã đặt nền móng cho nhiều ứng dụng thực tế phức tạp hơn liên quan đến **Nhận dạng Ký tự Quang học (Optical Character Recognition - OCR)**, chẳng hạn như:
  - Tự động đọc mã bưu chính (ZIP code) trên phong bì thư.
  - Xử lý séc ngân hàng tự động (đọc số tiền, số tài khoản).
  - Số hóa tài liệu, biển số xe và các văn bản viết tay khác.

Trong khuôn khổ đề tài này, MNIST được sử dụng làm bài toán cơ sở để kiểm chứng hiệu quả của kiến trúc CNN dựa trên ResNet trước khi áp dụng cho bộ dữ liệu hình dạng.

1.3 Nhận diện hình dạng cơ bản viết tay (hình chữ nhật, hình tròn, hình tam giác)

Bên cạnh việc nhận diện chữ số, khả năng phân biệt các hình dạng (shape) cơ bản là một bài toán nền tảng quan trọng khác trong thị giác máy tính. Nếu MNIST là bài toán "nhận diện ký tự" dựa trên các nét, thì đây là bài toán "nhận diện cấu trúc" dựa trên hình học. Việc máy tính có thể "hiểu" được các cấu trúc đơn giản như hình chữ nhật, hình tròn, và hình tam giác là bước đệm thiết yếu để giải quyết các vấnề phức tạp hơn.

1.3.1 Tầm quan trọng

Những hình dạng này được coi là các "nguyên tố" (primitives) trong nhận diện thị giác. Hầu hết các vật thể phức tạp trong thế giới thực đều có thể được phân rã hoặc mô tả gần đúng bằng sự kết hợp của các hình dạng cơ bản này. Khả năng nhận diện chúng là cơ sở cho các tác vụ như:

- Phân tích và hiểu các sơ đồ khối (flowcharts), bản vẽ kỹ thuật.
- Hỗ trợ các phần mềm thiết kế đồ họa (CAD, UI/UX).

- Giúp robot tương tác với các vật thể trong môi trường.

1.3.2 Thách thức của dữ liệu "Viết tay"

Tương tự như MNIST, yếu tố "viết tay" (hoặc "vẽ tay") mang đến những thách thức thực tế, khiến bài toán không hề tầm thường:

- **Tính biến thể (Variation):** Mỗi người vẽ một hình tròn hoặc một hình tam giác theo một cách khác nhau.
- **Tính không hoàn hảo (Imperfection):** Các đường nét có thể không thẳng, không khép kín (hở), bị run, hoặc có độ dày mỏng không đồng đều.
- **Sự đa dạng về xoay và tỷ lệ (Rotation & Scale):** Hình dạng có thể bị xoay ở nhiều góc độ khác nhau hoặc có tỷ lệ co dãn (ví dụ: hình chữ nhật đứng, hình chữ nhật nằm).

1.3.3 Vấn đề về bộ dữ liệu

Một điểm khác biệt lớn so với MNIST là không có một bộ dữ liệu tiêu chuẩn (**standard benchmark**) duy nhất được công nhận rộng rãi cho bài toán này. Trong khi MNIST đã được thu thập, tiền xử lý và chuẩn hóa cẩn thận, các bộ dữ liệu về hình dạng viết tay thường phải được tự tạo mới, tổng hợp từ nhiều nguồn, hoặc được tạo ra thông qua các công cụ vẽ (như **Google Quick, Draw! dataset**).

Điều này đặt ra một phần công việc quan trọng trong đề tài này là phải xây dựng một bộ dữ liệu đủ tốt, đủ đa dạng và được gán nhãn chính xác để có thể huấn luyện mô hình một cách hiệu quả, như sẽ được trình bày chi tiết ở Chương 2.

Trong khuôn khổ đề tài, việc giải quyết song song bài toán MNIST và bài toán hình dạng sẽ cung cấp một cái nhìn so sánh giá trị về cách một kiến trúc mạng (ResNet) học các loại đặc trưng khác nhau: một bên là đặc trưng nhận dạng ký tự (**topological features**) và một bên là đặc trưng nhận dạng cấu trúc (**geometric/structural features**).

1.4 Mạng nơ-ron tích chập (CNN): Khái niệm cơ bản

Mạng nơ-ron tích chập (Convolutional Neural Network - CNN hoặc ConvNet) là một lớp kiến trúc mạng nơ-ron nhân tạo **học sâu (Deep Learning)** được thiết kế đặc biệt để xử lý và phân tích dữ liệu có cấu trúc dạng lưới (grid-like data), mà ví dụ điển hình nhất chính là hình ảnh.

Nếu sử dụng **Mạng nơ-ron truyền thống** (ví dụ: **Multi-Layer Perceptron - MLP**) cho hình ảnh, chúng ta sẽ phải "duỗi thẳng" (flatten) ma trận pixel thành một vector dài. Điều này làm mất đi **cấu trúc không gian (spatial structure)** quý giá của ảnh (ví dụ: các pixel nào nằm cạnh nhau) và tạo ra một số lượng tham số (weights) khổng lồ, dẫn đến chi phí tính toán cao và dễ bị **quá khớp (overfitting)**.

CNN giải quyết vấn đề này bằng cách sử dụng các cơ chế thông minh để tự động và có thứ bậc (hierarchically) học các đặc trưng (features) từ ảnh, từ các đặc trưng

đơn giản (như cạnh, góc) đến các đặc trưng phức tạp (như hình dạng, bộ phận, vật thể).

Một kiến trúc CNN điển hình bao gồm ba loại lớp (layers) chính:

1.4.1 Lớp Tích chập (Convolutional Layer)

Đây là thành phần cốt lõi của CNN, thực hiện nhiệm vụ trích xuất đặc trưng.

- **Bộ lọc (Filter/Kernel):** Là một ma trận trọng số (weights) nhỏ (ví dụ: 3x3, 5x5). Bộ lọc này sẽ "trượt" (slide) qua toàn bộ ảnh đầu vào.
- **Phép tích chập (Convolution):** Tại mỗi vị trí, bộ lọc sẽ thực hiện phép nhân theo từng phần tử (element-wise multiplication) với vùng ảnh mà nó đang bao phủ, sau đó tính tổng tất cả các kết quả và cộng thêm một giá trị thiên vị (bias) b.
- **Bản đồ đặc trưng (Feature Map):** Kết quả của phép tích chập này tạo ra một "Bản đồ đặc trưng" mới, đại diện cho sự hiện diện của đặc trưng mà bộ lọc đó tìm kiếm (ví dụ: một bộ lọc có thể chuyên tìm các cạnh dọc).

Công thức toán học (cho 1 kênh ảnh, 1 bộ lọc): Nếu X là ma trận ảnh đầu vào và W là bộ lọc (kernel), giá trị tại vị trí (i, j) của bản đồ đặc trưng Z được tính bằng:

$$Z[i, j] = (X * W)[i, j] + b = \sum_u \sum_v X[i + u, j + v] \cdot W[u, v] + b$$

(Trong đó u và v là chỉ số của bộ lọc W).

Sau phép tích chập, một **hàm kích hoạt phi tuyến (Activation Function)** thường được áp dụng, phổ biến nhất là **ReLU (Rectified Linear Unit)**, để đưa tính phi tuyến vào mô hình.

Công thức ReLU:

$$A = \text{ReLU}(Z) = \max(0, Z)$$

1.4.2 Lớp Gộp (Pooling Layer)

Lớp gộp (thường đặt sau lớp tích chập) có hai mục tiêu chính:

- **Giảm chiều dữ liệu (Down-sampling):** Giảm kích thước (chiều rộng và chiều cao) của bản đồ đặc trưng, từ đó giảm số lượng tham số và chi phí tính toán ở các lớp sau.
- **Tạo tính bất biến (Invariance):** Giúp mô hình có khả năng "bất biến" với các thay đổi nhỏ về vị trí, xoay hoặc tỷ lệ của đặc trưng trong ảnh.

Hai loại Pooling phổ biến nhất là:

- **Max Pooling:** Lấy giá trị lớn nhất trong một cửa sổ (ví dụ: 2x2). Công thức (với cửa sổ $k \times k$):

  $$P[i, j] = \max_{u=0..k-1, v=0..k-1} (A[i \cdot s + u, j \cdot s + v])$$

  (Trong đó s là bước nhảy - stride).
- **Average Pooling:** Lấy giá trị trung bình trong cửa sổ.

1.4.3 Lớp Kết nối đầy đủ (Fully Connected Layer)

Sau khi đi qua một chuỗi các lớp Tích chập và Gộp, các bản đồ đặc trưng đã được trích xuất (lúc này có chiều sâu lớn nhưng kích thước nhỏ) sẽ được "duỗi thẳng" (flatten) thành một vector 1D.

Vector này sau đó được đưa vào một hoặc nhiều **lớp Kết nối đầy đủ** (giống như mạng MLP truyền thống) để thực hiện nhiệm vụ phân loại.

Công thức (cho 1 nơ-ron):

$$Z = W \cdot A_{\text{flat}} + b = \sum_i (w_i \cdot a_i) + b$$

Lớp cuối cùng (lớp đầu ra) thường sử dụng hàm kích hoạt **Softmax** để chuyển đổi các điểm số (logits) thành xác suất thuộc về mỗi lớp (ví dụ: 10 lớp cho MNIST), sao cho tổng các xác suất bằng 1.

Công thức Softmax (cho lớp i trong K lớp):

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Tóm tắt: Kiến trúc CNN hoạt động bằng cách xếp chồng các lớp này: các lớp tích chập và gộp ban đầu để học các đặc trưng (feature learning), theo sau là các lớp kết nối đầy đủ để phân loại (classification). Hai ưu điểm lớn nhất của CNN là **chia sẻ tham số (parameter sharing** - một bộ lọc dùng chung cho toàn ảnh) và **kết nối cục bộ (local connectivity** - chỉ học từ các vùng lân cận), giúp mô hình học hiệu quả cấu trúc không gian của ảnh.

1.5 Mô hình ResNet: Cấu trúc và ưu điểm

1.5.1 Vấn đề của các mạng CNN sâu (Deep CNNs)

Theo lý thuyết, các mô hình học sâu (Deep Learning) được hưởng lợi từ việc "sâu" hơn. Việc xếp chồng nhiều lớp (layers) cho phép mạng học được các đặc trưng (features) ở nhiều cấp độ trừu tượng khác nhau, từ các nét cạnh đơn giản (ở lớp đầu) đến các cấu trúc phức tạp (ở các lớp sau).

Tuy nhiên, trong thực tế, việc huấn luyện các mạng "rất sâu" (very deep) gặp phải hai vấn đề chính:

- **Suy giảm độ dốc (Vanishing Gradient):** Đây là vấn đề kinh điển. Khi lan truyền ngược (backpropagation), độ dốc (gradient) được nhân với nhau qua nhiều lớp. Nếu các giá trị này nhỏ hơn 1, độ dốc sẽ trở nên "tan biến" (nhỏ đến mức gần bằng 0) khi nó đến các lớp đầu tiên, khiến cho các lớp này không thể học được gì.
- **Suy thoái (Degradation):** Một vấn đề được phát hiện khi xây dựng các mạng sâu hơn VGG. Các nhà nghiên cứu (Kaiming He et al.) nhận thấy rằng, khi thêm nhiều lớp vào một mạng "đủ sâu", hiệu suất huấn luyện (training accuracy) bắt đầu bão hòa và sau đó... giảm xuống. Điều đáng nói là, đây không phải là do **quá khớp (overfitting)**, mà là do mô hình sâu hơn gặp khó khăn trong việc tối ưu hóa, thậm chí nó còn không thể học tốt bằng mô hình nông hơn.

1.5.2 Cấu trúc của ResNet: Khối dư (Residual Block)

ResNet (viết tắt của Residual Network - Mạng dư) được giới thiệu vào năm 2015 để giải quyết triệt để vấn đề suy thoái, và đồng thời cũng giải quyết luôn vấn đề suy giảm độ dốc.

Ý tưởng cốt lõi của ResNet là giới thiệu một **"kết nối tắt" (shortcut connection)**, còn được gọi là **"ánh xạ đồng nhất" (identity mapping)**.

Thay vì hy vọng một vài lớp xếp chồng (ví dụ: 2-3 lớp Tích chập) sẽ học được một ánh xạ phức tạp mong muốn $H(x)$, ResNet giả định rằng sẽ dễ dàng hơn nếu các lớp này chỉ học phần dư (residual) của ánh xạ đó.

Cụ thể, nếu x là đầu vào của một khối (block), và $H(x)$ là ánh xạ lý tưởng mà khối đó cần học, ResNet sẽ định nghĩa lại nhiệm vụ:

- Các lớp trong khối sẽ học một hàm $F(x)$.
- Đầu ra của khối sẽ là: $H(x) = F(x) + x$

Nói cách khác, các lớp (main path) chỉ cần học $F(x) = H(x) - x$ (phần dư). Kết nối tắt sẽ lấy đầu vào x ban đầu và cộng trực tiếp (element-wise addition) vào đầu ra $F(x)$ của các lớp.

Công thức toán học của một **Khối dư (Residual Block)**: Nếu x là đầu vào và y là đầu ra của khối, ta có:

$$y = \mathcal{F}(x, \{W_i\}) + x$$

Trong đó:

- x là vector đầu vào của khối.
- $\mathcal{F}(x, \{W_i\})$ là hàm ánh xạ (thường bao gồm 2 hoặc 3 lớp tích chập với các trọng số W_i) mà các lớp trong khối cần học.
- Phép toán $+x$ chính là kết nối tắt (shortcut connection) thực hiện phép cộng theo từng phần tử.

(Lưu ý: Nếu kích thước của x và $\mathcal{F}(x)$ không khớp nhau (ví dụ: do lớp tích chập có stride=2 làm giảm kích thước), kết nối tắt x cũng sẽ được biến đổi, thường là bằng một phép tích chập 1x1, để đảm bảo chúng có cùng kích thước trước khi cộng).

1.5.3 Ưu điểm của ResNet

Kiến trúc kết nối tắt này mang lại hai ưu điểm vượt trội:

- **Giải quyết vấn đề Suy thoái (Degradation):** Đây là ưu điểm trực tiếp nhất. Giả sử trong trường hợp xấu nhất, nếu các lớp mới thêm vào không học được gì hữu ích, cách tối ưu là chúng chỉ cần hoạt động như một "ánh xạ đồng nhất" (tức là $H(x) = x$).
  - Với CNN thông thường: Rất khó để một loạt lớp (Conv, ReLU) học cách sao chép y hệt đầu vào.
  - Với ResNet: Mạng chỉ cần học cách làm cho $F(x) = 0$ (đẩy trọng số W_i về 0). Điều này dễ dàng hơn rất nhiều. Kết quả $y = 0 + x = x$.

Do đó, việc thêm các lớp mới vào ResNet sẽ không bao giờ làm giảm hiệu suất (ít nhất là trên lý thuyết).

- **Giải quyết vấn đề Suy giảm độ dốc (Vanishing Gradient):** Kết nối tắt x tạo ra một con đường "cao tốc" cho độ dốc (gradient) khi lan truyền ngược. Trong quá trình backpropagation, dựa trên quy tắc chuỗi, độ dốc được truyền về x sẽ luôn có một thành phần là "1" (đến từ nhánh x). Điều này đảm bảo rằng, ngay cả khi độ dốc từ nhánh $F(x)$ rất nhỏ (gần bằng 0), độ dốc tổng thể sẽ không bao giờ bằng 0. Nó luôn có một con đường "sạch" để đi thẳng về các lớp trước đó, cho phép các mạng rất sâu (lên đến 152 lớp hoặc hơn) vẫn có thể được huấn luyện hiệu quả.

Nhờ những ưu điểm này, ResNet đã trở thành một trong những kiến trúc **xương sống (backbone)** tiêu chuẩn và hiệu quả nhất trong hầu hết các bài toán thị giác máy tính hiện đại, và là lựa chọn phù hợp cho đề tài này.

Chương 2

PHƯƠNG PHÁP XÂY DỰNG HỆ THỐNG

Để xây dựng một mô hình duy nhất có khả năng nhận diện đồng thời 13 lớp (10 chữ số MNIST và 3 hình dạng cơ bản), bước quan trọng và nền tảng nhất là xây dựng một bộ dữ liệu (dataset) thống nhất.

Thách thức chính trong giai đoạn này là phải xử lý hai nguồn dữ liệu có đặc điểm gốc rất khác nhau (MNIST là ảnh 28x28 chuẩn hóa, dữ liệu hình dạng là ảnh vẽ tay thô) để đưa chúng về cùng một định dạng (cùng kích thước, cùng kiểu tiền xử lý, cùng thang giá trị pixel) trước khi đưa vào huấn luyện mô hình.

2.1 Chuẩn bị dữ liệu

Quá trình chuẩn bị dữ liệu được chia thành ba giai đoạn chính: (1) Tải dữ liệu MNIST tiêu chuẩn, (2) Tạo và xử lý dữ liệu hình dạng tùy chỉnh, và (3) Hợp nhất, tiền xử lý và phân chia bộ dữ liệu tổng.

2.1.1 Thu thập dữ liệu MNIST

Chúng tôi sử dụng bộ dữ liệu MNIST tiêu chuẩn có sẵn trong thư viện `torchvision`. Bộ dữ liệu này được tải về tự động, bao gồm 60.000 ảnh cho tập huấn luyện (train) và 10.000 ảnh cho tập kiểm thử (test). Các nhãn (labels) gốc của MNIST (từ 0 đến 9) được giữ nguyên.

```
# Load MNIST (data dang o dang PIL Image) PIL: anh 2D size (28, 28)
    .
train_dataset = torchvision.datasets.MNIST(
    root="/content/drive/MyDrive/XLA",
    train=True,
    download=True,
    transform=train_transform
)
test_dataset = torchvision.datasets.MNIST(
    root="/content/drive/MyDrive/XLA",
    train=False,
    download=True,
    transform=test_transform
```

```
)
```

Listing 2.1: Tải bộ dữ liệu MNIST từ torchvision

2.1.2 Tạo bộ dữ liệu hình dạng viết tay (hình chữ nhật, tròn, tam giác)

Đây là bước phức tạp nhất, đòi hỏi phải tự tạo và xử lý dữ liệu hình dạng để chúng tương đồng nhất có thể với đặc điểm của dữ liệu MNIST (nét trắng trên nền đen, kích thước 28x28).

Tạo dữ liệu (Data Generation & Augmentation)

Từ một tập hợp ảnh vẽ tay thô (nét đen trên nền trắng) cho 3 lớp, chúng tôi xây dựng một quy trình tạo dữ liệu (augmentation) tùy chỉnh để sinh ra 7.000 ảnh cho mỗi lớp. Quy trình này bao gồm:

Chuẩn hóa kiểu MNIST (Hàm `mnist_style_resize`):

- Chuyển ảnh sang thang xám (Grayscale 'L').
- Tự động cắt (crop) viền thừa xung quanh nét vẽ.
- Resize ảnh sao cho cạnh dài nhất của nét vẽ bằng 20 pixel (giữ tỷ lệ).
- Làm nét (UnsharpMask) và tăng tương phản (Contrast) để nét vẽ rõ ràng hơn.
- Tạo một ảnh nền đen 28x28 và dán (paste) ảnh 20x20 đã xử lý vào giữa.

Tăng cường ngẫu nhiên (Hàm `random_augment_mnist_style`):

- Áp dụng xoay ngẫu nhiên trong khoảng [-20, 20] độ.
- Áp dụng kỹ thuật làm dày nét vẽ (Hàm `thicken_lines` sử dụng `cv2.dilate`) một cách ngẫu nhiên.

```
def mnist_style_resize(img, size=28, digit_size=20):
    # Chuyen anh sang grayscale
    img = img.convert("L")
    # Cat bo vung nen thua (bounding box)
    bbox = img.getbbox()
    if bbox:
        img = img.crop(bbox)
    # Resize giu ti le sao cho canh dai nhat = digit_size
    max_side = max(img.size)
    scale = digit_size / max_side
    new_size = tuple([int(x * scale) for x in img.size])
    img = img.resize(new_size, Image.LANCZOS)
    # Lam net va tang tuong phan
    img = img.filter(ImageFilter.UnsharpMask(radius=1, percent=150,
    threshold=3))
    enhancer = ImageEnhance.Contrast(img)
    img = enhancer.enhance(2.0)
    # Tao nen den 28x28 va dan vao giua
    new_img = Image.new("L", (size, size), 0)
```

```
    paste_x = (size - new_size[0]) // 2
    paste_y = (size - new_size[1]) // 2
    new_img.paste(img, (paste_x, paste_y))
    return new_img

def thicken_lines(img, thickness=2):
    np_img = np.array(img)
    if np_img.ndim == 3:
        np_img = np_img[:, :, 0]

    # Anh MNIST co nen den (0) va net trang (255)
    kernel = np.ones((thickness, thickness), np.uint8)
    dilated = cv2.dilate(np_img, kernel, iterations=1)
    return Image.fromarray(dilated)
```

Listing 2.2: Hàm xử lý ảnh về dạng MNIST và làm dày nét

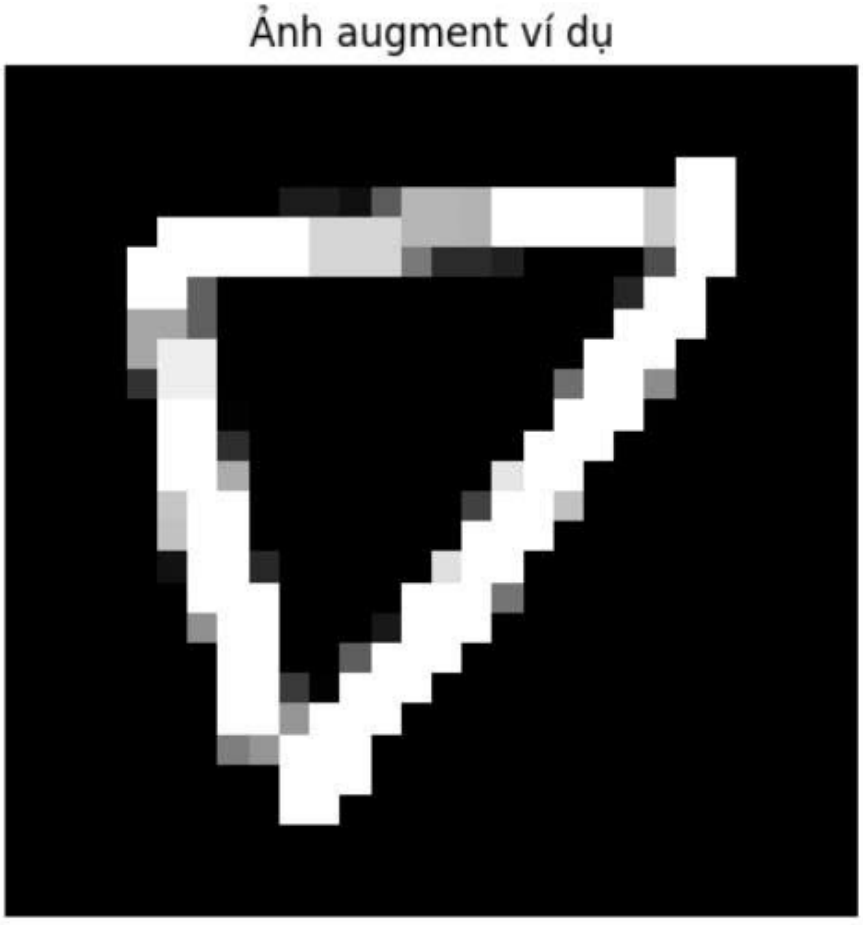
```
def random_augment_mnist_style(img):
    # Xoay ngau nhien
    if random.random() < 0.8:
        angle = random.uniform(-20, 20)
        img = img.rotate(angle, fillcolor=0)

    # Lam day net (them 2px)
    if random.random() < 0.7:
        img = thicken_lines(img, thickness=2)
    return img

def augment_with_custom_effects(input_dir, output_dir, target_count
    =7000):
    os.makedirs(output_dir, exist_ok=True)
    images = [os.path.join(input_dir, f) for f in os.listdir(
    input_dir) if f.lower().endswith(('.png', '.jpg'))]
    count = 0
    while count < target_count:
        img_path = random.choice(images)
        img = Image.open(img_path)
        img = mnist_style_resize(img)
        img = random_augment_mnist_style(img)
        save_path = os.path.join(output_dir, f"aug_{count}.png")
        img.save(save_path)
        count += 1
    print(f"Done augmenting {output_dir} with {target_count} images
    ")
```

Listing 2.3: Hàm tăng cường dữ liệu (augmentation) ngẫu nhiên

Tải dữ liệu (Data Loading)

Các ảnh đã tạo (tổng cộng 7.000 x 3 = 21.000 ảnh) được tải vào một lớp Dataset tùy chỉnh (`ShapeDataset`). Trong quá trình này, chúng tôi gán nhãn (label) mới cho 3 lớp hình dạng để phân biệt với 10 lớp MNIST:

- **Circle (Hình tròn):** Nhãn 10
- **Rectangle (Hình chữ nhật):** Nhãn 11
- **Triangle (Hình tam giác):** Nhãn 12



Hình 2.1: Ví dụ ảnh hình dạng sau khi qua quá trình chuẩn hóa MNIST và tăng cường ngẫu nhiên.

```
class ShapeDataset(Dataset):
    def __init__(self, root_dir, label, max_samples=None, transform
    =None):
        self.root_dir = root_dir
        self.files = sorted([
            os.path.join(root_dir, f)
            for f in os.listdir(root_dir)
            if f.endswith(('.png', '.jpg', '.jpeg'))
        ])
        if max_samples:
            self.files = self.files[:max_samples]
        self.label = label
        self.transform = transform
        self.targets = [label] * len(self.files)

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img_path = self.files[idx]
        img = Image.open(img_path).convert("L")
        # Ap dung transform neu co
```

```
        if self.transform:
            img = self.transform(img)
        return img, self.label
```

Listing 2.4: Lớp ShapeDataset tùy chỉnh

Tiền xử lý (Transforms) và Hợp nhất (Concatenation)

Định nghĩa Transforms: Chúng tôi định nghĩa hai quy trình biến đổi (transforms):

- `train_transform`: Áp dụng các tăng cường ngẫu nhiên như `RandomRotation(10)` và `RandomAffine`.
- `test_transform`: Chỉ chuyển sang Tensor và chuẩn hóa.

Điểm mấu chốt: Cả hai transform đều sử dụng phép chuẩn hóa `Normalize((0.1307,), (0.3081,))`. Đây chính là giá trị trung bình (mean) và độ lệch chuẩn (std) của bộ dữ liệu MNIST. Bằng cách áp dụng các giá trị này cho cả 13 lớp, chúng tôi đưa toàn bộ dữ liệu về cùng một phân phối chuẩn, giúp mô hình hội tụ tốt hơn.

```
train_transform = transforms.Compose([
    transforms.RandomApply([
        transforms.RandomRotation(10),
        transforms.RandomAffine(0, translate=(0.1, 0.1), scale
   =(0.9, 1.1))
    ], p=1.0),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# Transform cho tap test (chi chuan hoa)
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

Listing 2.5: Định nghĩa các phép biến đổi (transforms)

Phân chia và Hợp nhất: Dữ liệu Hình dạng (7.000 ảnh/lớp) được chia thành 6.000 train và 1.000 test. Sử dụng `ConcatDataset`, chúng tôi gộp các bộ dữ liệu lại:

- `train_dataset_full`: Gồm 60.000 ảnh MNIST train + (3 * 6.000 ảnh Shape train) = 78.000 ảnh.
- `test_dataset_full`: Gồm 10.000 ảnh MNIST test + (3 * 1.000 ảnh Shape test) = 13.000 ảnh.

Cuối cùng, bộ `test_dataset_full` (13.000 ảnh) được chia ngẫu nhiên (tỷ lệ 50/50) để tạo ra hai tập cuối cùng: `val_dataset` (Tập kiểm định: 6.500 ảnh) và `final_test_dataset` (Tập kiểm thử: 6.500 ảnh).

```
# Chia train / test
train_size = 6000
generator = torch.Generator().manual_seed(42)
circle_train, circle_test = random_split(circle_full_dataset, [
   train_size, len(circle_full_dataset) - train_size], generator=
   generator)
```

```
rect_train, rect_test = random_split(rect_full_dataset, [train_size, len(rect_full_dataset) - train_size], generator=generator)
tri_train, tri_test = random_split(tri_full_dataset, [train_size, len(tri_full_dataset) - train_size], generator=generator)

# Chia test -> validation + test (tach rieng cho tung shape)
val_ratio = 0.5
circle_val_size = int(len(circle_test) * val_ratio)
rect_val_size = int(len(rect_test) * val_ratio)
tri_val_size = int(len(tri_test) * val_ratio)

circle_val, circle_final_test = random_split(circle_test, [circle_val_size, len(circle_test) - circle_val_size], generator=generator)
rect_val, rect_final_test = random_split(rect_test, [rect_val_size, len(rect_test) - rect_val_size], generator=generator)
tri_val, tri_final_test = random_split(tri_test, [tri_val_size, len(tri_test) - tri_val_size], generator=generator)

# ===== Gop tap train, val, test cuoi cung =====
train_dataset_full = ConcatDataset([train_dataset, circle_train, rect_train, tri_train])
val_dataset = ConcatDataset([val_dataset_mnist, circle_val, rect_val, tri_val])
final_test_dataset = ConcatDataset([test_dataset_mnist, circle_final_test, rect_final_test, tri_final_test])
```

Listing 2.6: Chia và gộp các bộ dữ liệu

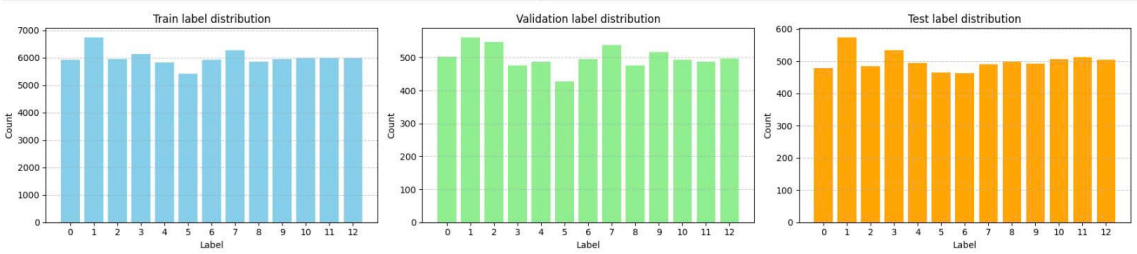
Kiểm tra và Trực quan hóa Dữ liệu

Sau khi gộp, chúng tôi tiến hành kiểm tra phân bổ nhãn (label distribution) trong cả 3 tập (train, val, test) để đảm bảo dữ liệu cân bằng tương đối.

```
def get_all_labels(dataset):
    """Trich xuat tat ca nhan tu dataset (ke ca ConcatDataset, Subset)."""
    if hasattr(dataset, 'targets'):
        return np.array(dataset.targets)
    elif hasattr(dataset, 'datasets'):  # Truong hop ConcatDataset
        return np.concatenate([get_all_labels(d) for d in dataset.datasets])
    elif hasattr(dataset, 'dataset') and hasattr(dataset, 'indices'):  # Truong hop Subset
        all_targets = get_all_labels(dataset.dataset)
        return all_targets[dataset.indices]
    else:
        raise AttributeError("Dataset khong co thuoc tinh 'targets' hoc 'datasets'.")

# Lay tat ca nhan tu cac tap
train_labels = get_all_labels(train_dataset_full)
val_labels = get_all_labels(val_dataset)
```

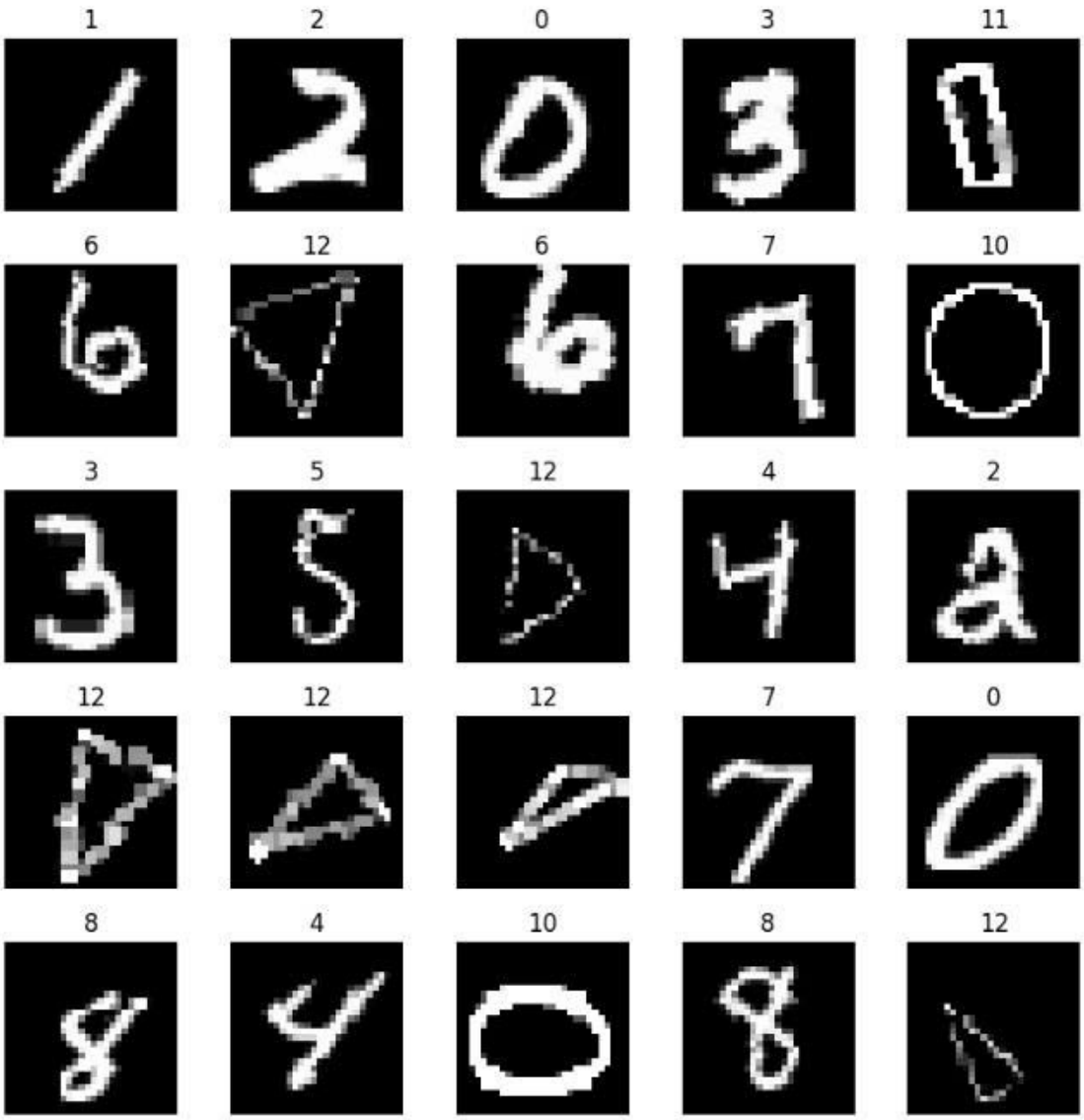
```
test_labels  = get_all_labels(final_test_dataset)
```

Listing 2.7: Hàm trích xuất nhãn từ các loại Dataset



Hình 2.2: Biểu đồ phân bố nhãn (label distribution) cho các tập Train, Validation và Test.

Đồng thời, chúng tôi trực quan hóa một số ảnh ngẫu nhiên từ tập huấn luyện tổng hợp (`train_dataset_full`) để đảm bảo cả chữ số và hình dạng đều được xử lý và gán nhãn chính xác.



Hình 2.3: Một số ảnh ngẫu nhiên từ tập huấn luyện tổng hợp (bao gồm cả MNIST và Hình dạng).

Tạo DataLoader

Cuối cùng, 3 bộ dữ liệu (train, val, test) được đưa vào `DataLoader` để chuẩn bị cho quá trình huấn luyện, với `batch_size=128` cho tập train và `shuffle=True`.

```
# DataLoader cho tung tap
train_loader = DataLoader(train_dataset_full, batch_size=128,
    shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=1000, shuffle=
    False)
test_loader  = DataLoader(final_test_dataset, batch_size=1000,
    shuffle=False)
```

Listing 2.8: Khởi tạo DataLoaders

2.2 Thiết kế mô hình CNN dựa trên ResNet

Để giải quyết bài toán phân loại 13 lớp từ ảnh đầu vào 1 kênh 28x28, chúng tôi thiết kế một kiến trúc CNN tùy chỉnh, đặt tên là **ModernCNN**, dựa trên nguyên lý cốt lõi của ResNet: sử dụng các **khối dư (Residual Blocks)**.

2.2.1 Khối dư (ResidualBlock)

Đây là đơn vị xây dựng cơ bản. Mỗi khối bao gồm:

- Hai lớp Tích chập (`Conv2d`) 3x3, kèm theo `BatchNorm2d`.
- Một **kết nối tắt (shortcut connection)** thực hiện phép cộng `out += self.shortcut(x)`.
- Nếu kích thước hoặc số kênh thay đổi, kết nối tắt sẽ dùng `Conv2d` 1x1 để điều chỉnh kích thước.
- Một lớp `Dropout2d` được thêm vào để chống quá khớp.
- Sử dụng hàm kích hoạt **GELU** (`F.gelu`) thay vì ReLU truyền thống.

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, dropout=0.2):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
                               stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.dropout = nn.Dropout2d(dropout)  # dropout trong feature maps

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.gelu(self.bn1(self.conv1(x)))
        out = self.dropout(out)                # dropout ngay sau conv1
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)                # skip connection
        out = F.gelu(out)
        return out
```

Listing 2.9: Định nghĩa ResidualBlock

2.2.2 Kiến trúc mô hình ModernCNN

Mô hình tổng thể được xây dựng bằng cách xếp chồng 4 Khối dư:

- **Đầu vào:** (1x28x28).
- **Layer 1 (ResidualBlock):** 1 → 32 kênh, stride=1. Output: (32x28x28).
- **Layer 2 (ResidualBlock):** 32 → 64 kênh, stride=2. Output: (64x14x14).
- **Layer 3 (ResidualBlock):** 64 → 128 kênh, stride=2. Output: (128x7x7).
- **Layer 4 (ResidualBlock):** 128 → 256 kênh, stride=2. Output: (256x4x4).
- **Lớp Phân loại (Classifier Head):**
  - Sử dụng **Global Average Pooling** (`nn.AdaptiveAvgPool2d`) để chuyển (256x4x4) thành (256x1x1).
  - "Duỗi thẳng" (flatten) thành vector 256 chiều.
  - Một lớp `Dropout` (0.5).
  - Một lớp `nn.Linear` (256 → 13) để ra 13 lớp đầu ra.

```
class ModernCNN(nn.Module):
    def __init__(self, num_classes=13, dropout_fc=0.5):
        super(ModernCNN, self).__init__()
        self.layer1 = ResidualBlock(1, 32, stride=1, dropout=0.1)
        self.layer2 = ResidualBlock(32, 64, stride=2, dropout=0.1)
        self.layer3 = ResidualBlock(64, 128, stride=2, dropout=0.2)
        self.layer4 = ResidualBlock(128, 256, stride=2, dropout
   =0.3)

        self.gap = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout_fc = nn.Dropout(dropout_fc)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.gap(x)             # (B, 256, 1, 1)
        x = torch.flatten(x, 1)     # (B, 256)
        x = self.dropout_fc(x)
        x = self.fc(x)
        return x
```

Listing 2.10: Định nghĩa kiến trúc ModernCNN

2.3 Huấn luyện mô hình

2.3.1 Các tham số huấn luyện

- **Hàm mất mát (Loss Function):** `nn.CrossEntropyLoss` (tiêu chuẩn cho phân loại đa lớp).

- **Hàm tối ưu (Optimizer):** `optim.Adam` (tự điều chỉnh learning rate).
- **Siêu tham số (Hyperparameters):**
  - Learning Rate: 0.001
  - Số Epochs: 10
  - Batch Size: 128

```
model = ModernCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1
    e-4)
```

Listing 2.11: Thiết lập hàm mất mát và tối ưu

2.3.2 Kỹ thuật tránh overfitting

Chúng tôi áp dụng đồng thời nhiều kỹ thuật chính quy hóa (regularization):

- **Data Augmentation:** `RandomRotation` và `RandomAffine` được áp dụng động trong `train_loader`.
- **Chiến lược Tăng cường Epoch:** Trong mỗi epoch, chúng tôi lặp lại `train_loader` 4 lần (`augment_times = 4`), buộc mô hình phải xem 4 phiên bản tăng cường khác nhau.
- **Dropout2d:** Áp dụng bên trong các `ResidualBlock` (tỷ lệ 0.1-0.3).
- **Dropout:** Áp dụng trước lớp `Linear` cuối cùng (tỷ lệ 0.5).
- **Weight Decay (L2 Regularization):** Thiết lập `weight_decay=1e-4` trong `Adam` để "phạt" các trọng số lớn.
- **Batch Normalization:** Giúp ổn định huấn luyện và cũng có tác dụng chính quy hóa nhẹ.

```
num_epochs = 10
augment_times = 4  # so lan tang cuong ngau nhien mai anh
train_losses = []
val_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    print(f"\nEpoch {epoch+1}/{num_epochs} bat dau...")

    # Lap lai train_loader 4 lan
    for aug_round in range(augment_times):
        progress_bar = tqdm(
            train_loader,
            desc=f"Epoch {epoch+1}/{num_epochs} | Aug {aug_round
   +1}/{augment_times}",
            leave=False
```

```
        )
        for data, target in progress_bar:
            data, target = data.to(device), target.to(device)
            outputs = model(data)
            loss = criterion(outputs, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            avg_loss = running_loss / ((progress_bar.n + 1) +
    aug_round * len(train_loader))
            progress_bar.set_postfix({"loss": f"{avg_loss:.4f}"})

     avg_loss = running_loss / (len(train_loader) * augment_times)
     train_losses.append(avg_loss)
     print(f"Epoch [{epoch+1}/{num_epochs}] | Avg Loss: {avg_loss:.4
    f}")

     # ===== Danh gia tren tap validation =====
     model.eval()
     val_correct, val_total = 0, 0
     with torch.no_grad():
         for data, target in val_loader:
             data, target = data.to(device), target.to(device)
             outputs = model(data)
             _, predicted = torch.max(outputs.data, 1)
             val_total += target.size(0)
             val_correct += (predicted == target).sum().item()
     val_acc = 100 * val_correct / val_total
     val_accuracies.append(val_acc)
     print(f"Validation Accuracy: {val_acc:.2f}%")

     # ===== Danh gia tran tap test =====
     (Tuong tu nhu validation...)
     print(f"Test Accuracy: {test_acc:.2f}%")
```

Listing 2.12: Vòng lặp huấn luyện (training loop)

2.4 Đánh giá mô hình

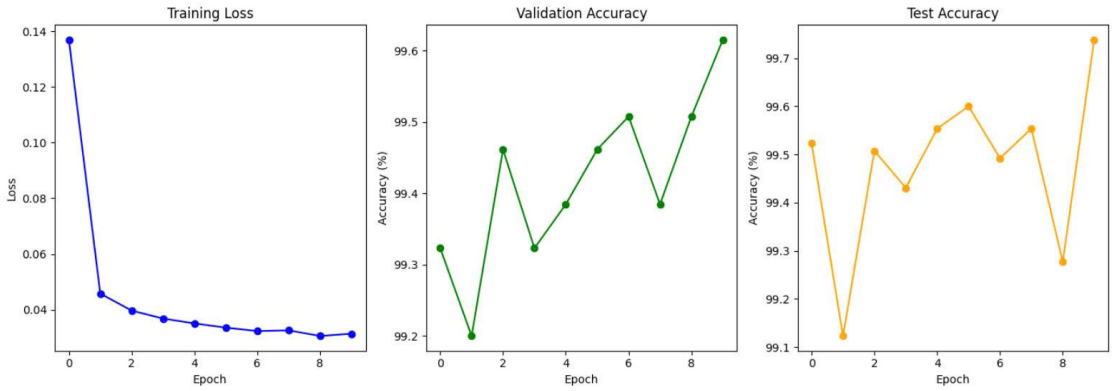
Để đánh giá một cách toàn diện hiệu suất của mô hình ModernCNN đã được huấn luyện, chúng tôi không chỉ dựa vào độ chính xác tổng thể mà còn sử dụng một bộ các chỉ số đo lường (metrics) chi tiết. Quá trình đánh giá được thực hiện trên tập dữ liệu kiểm thử cuối cùng (`final_test_dataset`, 6.500 ảnh) - một tập dữ liệu mà mô hình chưa bao giờ "nhìn thấy" trong suốt quá trình huấn luyện và kiểm định (validation).`final_test_dataset` (6.500 ảnh).

2.4.1 Biểu đồ Huấn luyện (Training Plots)

Chúng tôi theo dõi và trực quan hóa ba chỉ số chính qua các epoch:

- **Training Loss:** Cho thấy mô hình học dữ liệu huấn luyện tốt như thế nào.
- **Validation Accuracy:** Chỉ số chính để theo dõi hiệu suất trên dữ liệu mới và phát hiện **overfitting**.

- **Test Accuracy:** Kết quả cuối cùng của mô hình trên tập kiểm thử sau mỗi epoch.



Hình 2.4: Biểu đồ kết quả huấn luyện: Loss (Train) và Accuracy (Validation, Test) qua các epoch.

2.4.2 Các chỉ số Phân loại (Classification Metrics)

Chúng tôi sử dụng các chỉ số tiêu chuẩn từ `scikit-learn`:

- **Accuracy (Độ chính xác tổng thể):** Tỷ lệ dự đoán đúng.

$$\text{Accuracy} = \frac{\text{Số dự đoán đúng}}{\text{Tổng số mẫu}}$$

- **Precision (Độ chuẩn):** Trong số các ảnh được dự đoán là lớp A, có bao nhiêu ảnh thực sự là lớp A?

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Độ phủ):** Trong số các ảnh thực sự là lớp A, mô hình dự đoán đúng được bao nhiêu phần trăm?

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Trung bình điều hòa của Precision và Recall.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.4.3 Ma trận nhầm lẫn (Confusion Matrix)

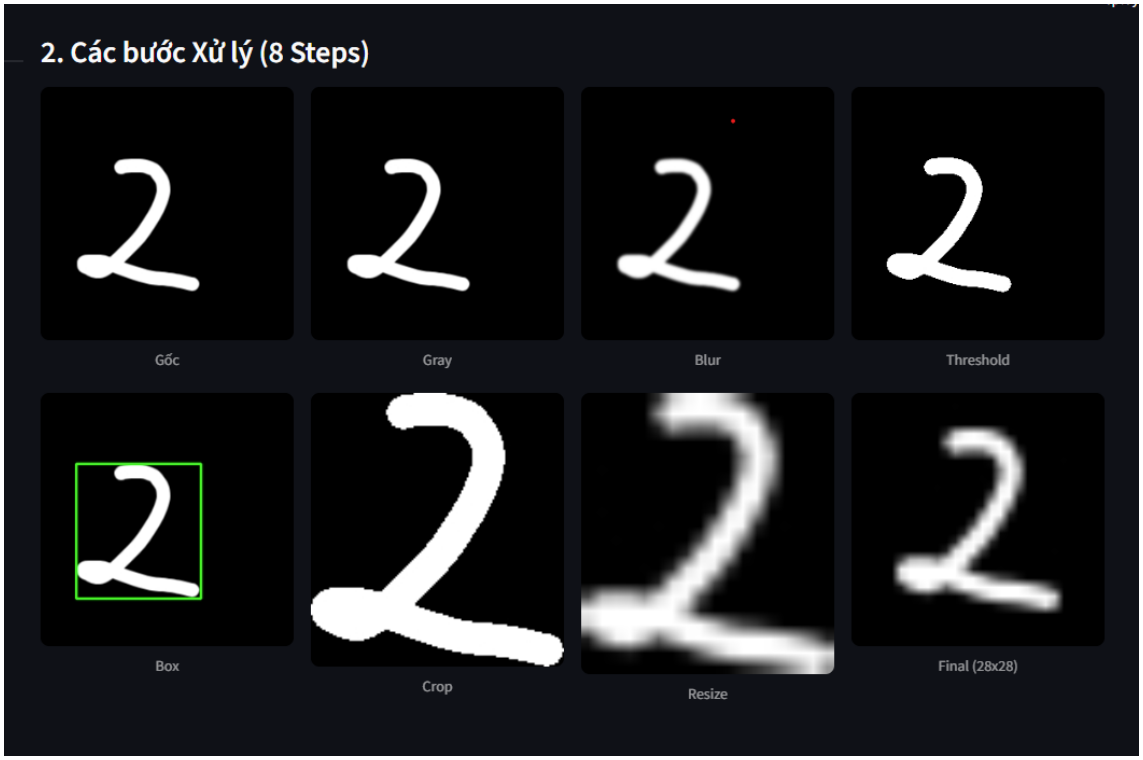
Đây là công cụ trực quan hóa quan trọng nhất để phân tích lỗi. Ma trận này cho thấy mô hình đang nhầm lẫn giữa các lớp nào. Ví dụ, nó sẽ trả lời các câu hỏi:

- Mô hình có nhầm lẫn giữa chữ số '0' và 'hình tròn' (lớp 10) không?
- Chữ số '1' có bị nhầm với 'hình tam giác' (lớp 12) không?

2.5 Quy trình xử lý ảnh đầu vào thực tế (Inference Pipeline)

Để áp dụng mô hình vào ứng dụng thực tế, chúng tôi xây dựng quy trình tiền xử lý gồm 8 bước tuần tự, được trực quan hóa trên giao diện ứng dụng để minh bạch hóa quá trình "suy luận" của máy tính:

- **Ảnh gốc (Original Input):** Thu nhận ảnh đầu vào từ Canvas (vẽ tay) hoặc Upload, chuyển đổi sang định dạng mảng (array).
- **Ảnh xám (Grayscale):** Chuyển đổi không gian màu sang thang xám để loại bỏ thông tin màu sắc nhiễu.
- **Làm mờ (Gaussian Blur):** Áp dụng bộ lọc Gaussian (kernel 7×7) để làm mịn các đường nét đứt gãy hoặc răng cưa do thao tác vẽ tay.
- **Phân ngưỡng (Thresholding):** Tự động tách đối tượng ra khỏi nền (sử dụng Adaptive Threshold nếu nền sáng, hoặc Threshold cố định nếu nền tối) để tạo ra ảnh nhị phân.
- **Xác định vùng biên (Contour Box):** Tìm kiếm đường viền (contour) lớn nhất và vẽ hình chữ nhật bao quanh (Bounding Box) để định vị đối tượng.
- **Cắt ảnh (Cropping):** Cắt vùng ảnh chứa đối tượng dựa trên Bounding Box đã xác định ở bước 5, loại bỏ hoàn toàn phần nền thừa.
- **Biến đổi kích thước (Resize 20px):** Thay đổi kích thước ảnh đã cắt sao cho cạnh dài nhất bằng 20 pixel (giữ nguyên tỷ lệ khung hình - aspect ratio), mô phỏng cách tiền xử lý của bộ dữ liệu MNIST.
- **Tạo ảnh cuối (Final 28×28):** Tạo một nền đen kích thước chuẩn 28×28 pixel và đặt ảnh (đã resize ở bước 7) vào chính giữa. Đây là đầu vào cuối cùng tương thích với mô hình ModernCNN.



Hình 2.5: Các bước xử lý để dự đoán thực tế

Chương 3

THỰC NGHIỆM VÀ KẾT QUẢ

3.1 Môi trường thực nghiệm (công cụ, phần cứng)

Quá trình thực nghiệm, bao gồm tiền xử lý dữ liệu, thiết kế mô hình, huấn luyện và đánh giá, được thực hiện trên một môi trường thống nhất.

3.1.1 Phần cứng

Toàn bộ quá trình huấn luyện mô hình học sâu được thực hiện trên nền tảng **Google Colaboratory (Colab)**. Điều này cho phép chúng tôi tận dụng tài nguyên phần cứng mạnh mẽ, cụ thể:

- **CPU:** Bộ xử lý (CPU) do Google Colab cung cấp.
- **GPU:** Quá trình huấn luyện sử dụng Bộ xử lý đồ họa (GPU) **NVIDIA** (ví dụ: Tesla T4, P100) được cấp phát bởi Colab (thông qua `device = torch.device("cuda")`). Việc sử dụng GPU giúp tăng tốc đáng kể các phép toán ma trận trong quá trình huấn luyện, rút ngắn thời gian từ vài giờ xuống còn vài phút.

3.1.2 Phần mềm và Thư viện

Hệ thống được xây dựng chủ yếu dựa trên ngôn ngữ lập trình **Python 3.x** và các thư viện mã nguồn mở phổ biến trong lĩnh vực Khoa học dữ liệu và Học sâu:

- **Ngôn ngữ lập trình:** Python 3.x.
- **Thư viện Học sâu chính: PyTorch**, một framework linh hoạt được sử dụng để xây dựng kiến trúc mạng (lớp `nn.Module`), định nghĩa hàm mất mát (`nn.CrossEntropyLoss`), và tối ưu hóa (`optim.Adam`).
- **Thư viện Thị giác máy tính:**
  - **TorchVision:** Dùng để tải bộ dữ liệu MNIST và cung cấp các hàm biến đổi (`transforms`) chuẩn hóa.
  - **PIL (Pillow):** Được sử dụng rộng rãi trong việc tạo dữ liệu hình dạng, thực hiện các thao tác xử lý ảnh cơ bản như mở, resize, crop, và lọc ảnh.

  - **OpenCV (cv2):** Được sử dụng cho tác vụ chuyên biệt là làm dày nét vẽ (`cv2.dilate`).

- **Thư viện tính toán: NumPy**, dùng để hỗ trợ các thao tác với mảng.
- **Thư viện trực quan hóa: Matplotlib**, được dùng để hiển thị các ảnh mẫu, vẽ biểu đồ phân bố dữ liệu và biểu đồ kết quả huấn luyện (loss, accuracy).
- **Tiện ích: Tqdm**, được sử dụng để hiển thị thanh tiến trình (progress bar) trực quan khi huấn luyện mô hình.

3.2 Kết quả nhận diện MNIST

Đây là phần phân tích hiệu suất của mô hình `ModernCNN` chỉ trên 10 lớp chữ số (từ 0 đến 9). Để có được kết quả này, chúng tôi lọc các dự đoán và nhãn thật từ tập kiểm thử (tổng 6.500 ảnh), giữ lại 4.976 mẫu thuộc 10 lớp MNIST.

3.2.1 Độ chính xác và các chỉ số đánh giá (accuracy, precision, recall)

Kết quả đánh giá chi tiết cho từng lớp chữ số được trình bày trong Bảng 3.1.

Bảng 3.1: Báo cáo kết quả nhóm MNIST (Lớp 0-9)

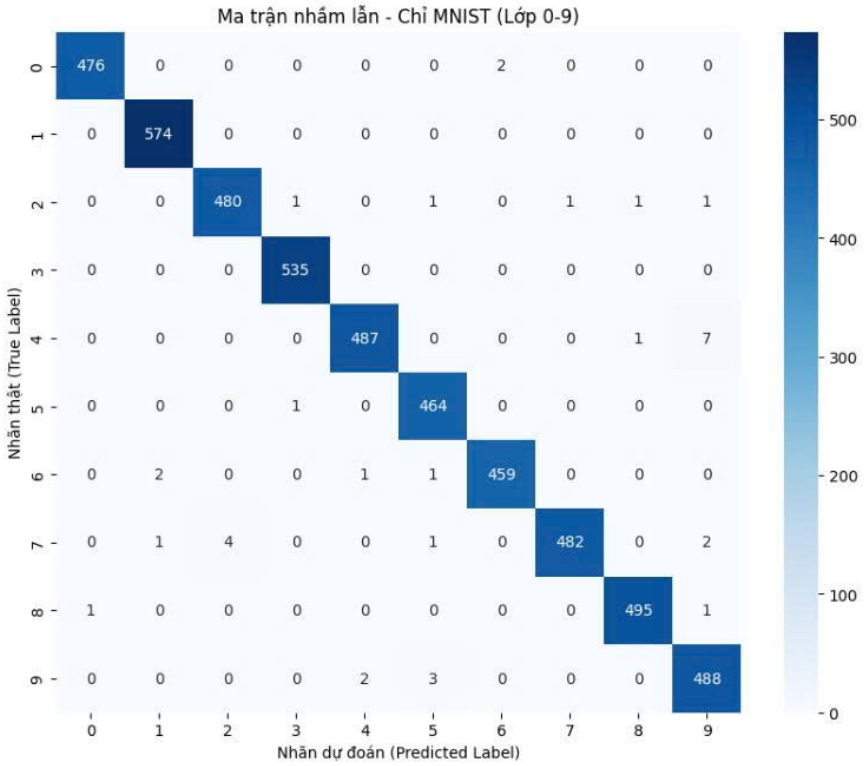
Lớp	**precision**	**recall**	**f1-score**	**support**
0	0.998	0.996	0.997	478
1	0.995	1.000	0.997	574
2	0.992	0.990	0.991	485
3	0.996	1.000	0.998	535
4	0.994	0.984	0.989	495
5	0.987	0.998	0.993	465
6	0.996	0.991	0.994	463
7	0.998	0.984	0.991	490
8	0.996	0.994	0.995	498
9	0.978	0.990	0.984	493
weighted avg	**0.993**	**0.993**	**0.993**	**4976**

Nhận xét: Kết quả (từ Bảng 3.1) cho thấy mô hình `ModernCNN` dựa trên ResNet đạt hiệu suất cực kỳ cao trên tác vụ nhận diện chữ số MNIST.

- Độ chính xác tổng thể (**Accuracy**) trên các lớp MNIST đạt **99.3%** (tính theo weighted avg).
- Các chỉ số **Precision**, **Recall**, và **F1-Score** cho tất cả các lớp chữ số đều rất cao, hầu hết đều vượt 0.99, và thấp nhất cũng ở mức 0.978 (precision lớp '9').
- Điều này chứng tỏ kiến trúc ResNet (với các `ResidualBlock`) đã hoạt động rất hiệu quả. Mặc dù được huấn luyện chung với các hình dạng khác, mô hình vẫn duy trì được hiệu suất đỉnh cao trên bộ MNIST.

3.2.2 Phân tích lỗi và ví dụ minh họa

Để hiểu rõ hơn về các trường hợp mô hình dự đoán sai, chúng tôi sử dụng **Ma trận nhầm lẫn (Confusion Matrix)** cho 10 lớp MNIST.

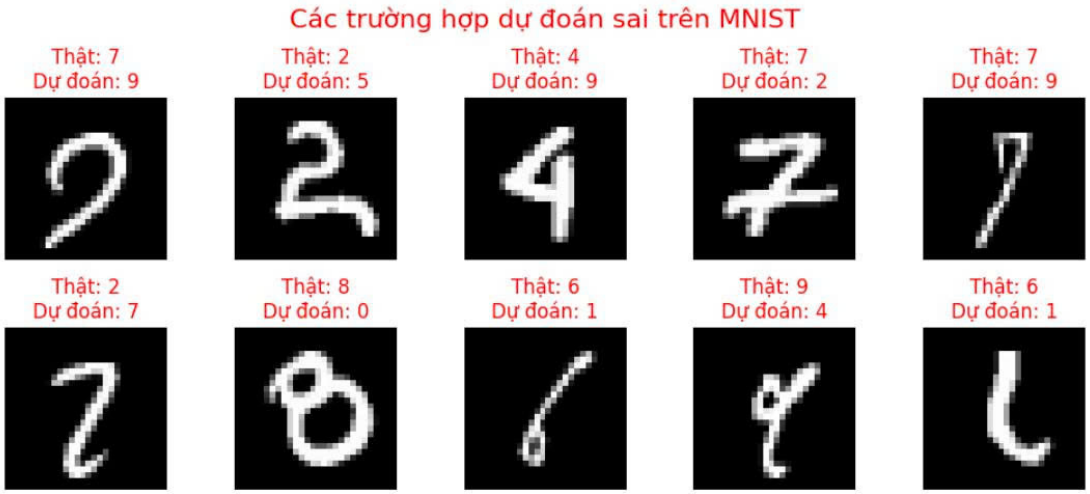


Hình 3.1: Ma trận nhầm lẫn cho 10 lớp MNIST.

Nhận xét: Nhìn vào ma trận nhầm lẫn (Hình 3.1), ta thấy các giá trị trên đường chéo chính chiếm gần như tuyệt đối, khẳng định lại độ chính xác 99.3%. Các lỗi sai (nằm ngoài đường chéo chính) là rất ít. Phân tích kỹ các ô có lỗi, ta thấy các nhầm lẫn chủ yếu tập trung vào các cặp số vốn dĩ dễ nhầm lẫn về mặt hình thái:

- Nhãn thật '7' bị dự đoán nhầm là '1' (4 lần).
- Nhãn thật '9' bị dự đoán nhầm là '7' (3 lần) và '4' (2 lần).
- Nhãn thật '6' bị dự đoán nhầm là '2' (2 lần).
- Các nhầm lẫn khác chỉ xuất hiện 1 lần (ví dụ: '2' ra '8', '5' ra '3', '8' ra '9'...).

Dưới đây là một số ví dụ minh họa về các trường hợp mô hình dự đoán sai (dựa trên các cặp nhầm lẫn phổ biến nhất):



Hình 3.2: Ví dụ minh họa các trường hợp dự đoán sai (nhóm MNIST).

3.3 Kết quả nhận diện hình dạng viết tay

Phần này tập trung phân tích hiệu suất của cùng một mô hình `ModernCNN`, nhưng chỉ xét trên 3 lớp hình dạng cơ bản viết tay (Lớp 10: Hình tròn, Lớp 11: Hình chữ nhật, Lớp 12: Hình tam giác). Chúng tôi lọc 1.524 mẫu thuộc nhóm này từ tập kiểm thử để đánh giá.

3.3.1 Độ chính xác và các chỉ số đánh giá

Các chỉ số hiệu suất chi tiết cho 3 lớp hình dạng được trình bày trong Bảng 3.2.

Bảng 3.2: Báo cáo kết quả nhóm Hình dạng (Lớp 10-12)

Lớp	**precision**	**recall**	**f1-score**	**support**
10 (Circle)	0.978	0.878	0.925	507
11 (Rectangle)	0.956	0.856	0.903	513
12 (Triangle)	0.842	0.992	0.911	504
weighted avg	**0.926**	**0.918**	**0.913**	**1524**

Nhận xét: Kết quả (từ Bảng 3.2) cho thấy hiệu suất của mô hình trên nhóm dữ liệu hình dạng tự tạo là khá tốt, nhưng không cao bằng nhóm MNIST.

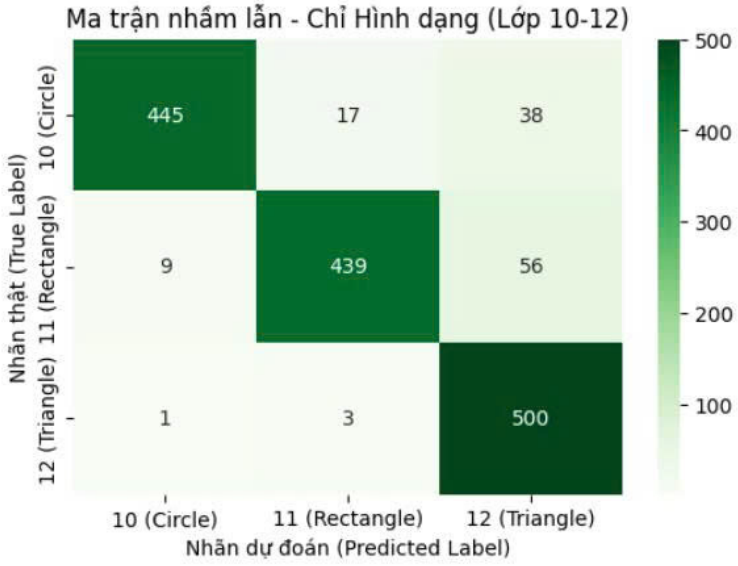
- Độ chính xác tổng thể (**Accuracy**) trên 3 lớp này đạt **92.6%** (tính theo weighted avg).
- Phân tích chi tiết các chỉ số cho thấy một số điểm mất cân bằng:
  - **Lớp 12 (Triangle):** Đạt chỉ số **Recall** rất cao (**0.992**), nghĩa là mô hình gần như không bỏ sót bất kỳ hình tam giác nào. Tuy nhiên, chỉ số **Precision** lại thấp nhất (**0.842**), cho thấy mô hình có xu hướng "quá tự tin" và đoán nhầm nhiều hình khác (Circle, Rectangle) thành Triangle.

  - **Lớp 10 (Circle) và 11 (Rectangle):** Có chỉ số **Precision** cao (0.978 và 0.956) nhưng **Recall** lại thấp hơn (0.878 và 0.856). Điều này có nghĩa là khi mô hình dự đoán là Circle/Rectangle, nó thường là đúng; nhưng nó đã bỏ sót (misclassified) một lượng đáng kể Circle/Rectangle sang các lớp khác.

- Dù sao, kết quả này vẫn chứng tỏ quy trình tiền xử lý (như `thicken_lines`, `mnist_style_resize`) đã hoạt động tương đối hiệu quả, giúp mô hình học được các đặc trưng cơ bản.

3.3.2 Phân tích lỗi và ví dụ minh họa

Ma trận nhầm lẫn (Confusion Matrix) cho 3 lớp hình dạng (Hình 3.3) cung cấp cái nhìn chi tiết hơn về các lỗi sai.



Hình 3.3: Ma trận nhầm lẫn cho 3 lớp Hình dạng (10, 11, 12).

Nhận xét: Ma trận nhầm lẫn (Hình 3.3) làm rõ các nhận xét từ Bảng 3.2:

- Đường chéo chính vẫn chiếm đa số (445, 439, 500), tương ứng với độ chính xác 91.8%.

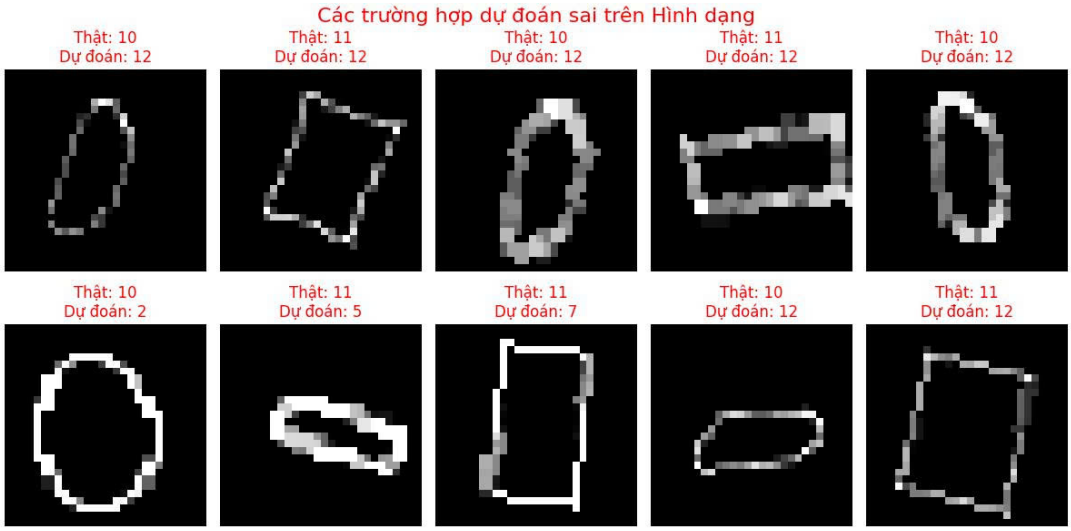
- **Lớp 12 (Triangle)** được nhận diện rất tốt (Recall cao): Trong tổng số 504 ảnh Triangle, chỉ có 1 ảnh bị nhầm thành Circle và 3 ảnh bị nhầm thành Rectangle.

- **Vấn đề nhầm lẫn chính:** Lỗi sai chủ yếu đến từ việc các lớp khác bị đoán nhầm thành **Triangle** (Precision thấp cho lớp Triangle).
  - Lớp 11 (Rectangle) bị đoán nhầm là Triangle (Lớp 12) tới 56 lần.
  - Lớp 10 (Circle) bị đoán nhầm là Triangle (Lớp 12) tới 38 lần.

- Cũng có sự nhầm lẫn giữa Lớp 10 (Circle) và Lớp 11 (Rectangle) nhưng ở mức độ thấp hơn (17 lần và 9 lần).

Lý giải: Điều này cho thấy dữ liệu hình dạng tự tạo có thể chưa đủ đa dạng, hoặc các ảnh Circle/Rectangle được vẽ "méo" hoặc "nhọn" (ví dụ: vẽ vội, các góc không vuông) khiến chúng có đặc trưng gần giống với Triangle, làm mô hình bị nhầm lẫn.

Dưới đây là một số ví dụ minh họa về các trường hợp dự đoán sai phổ biến:



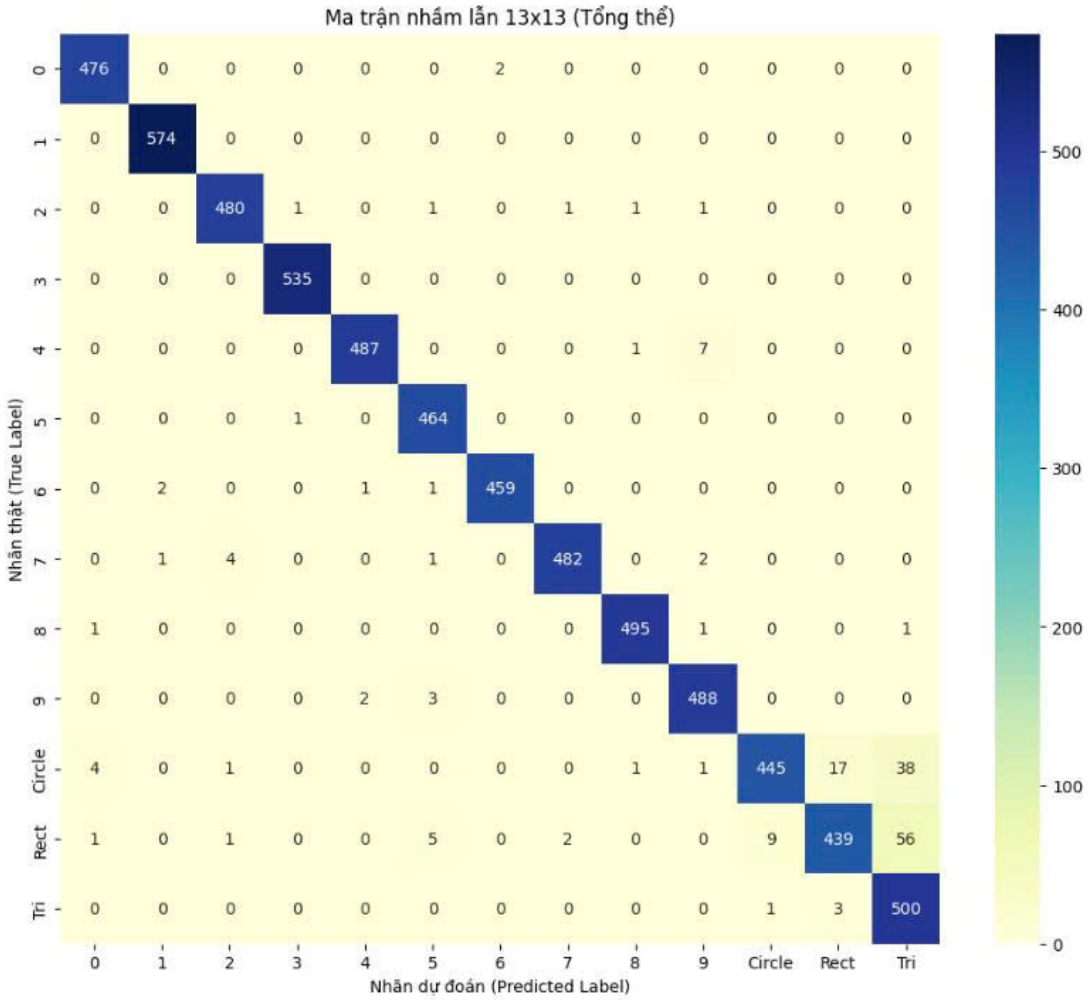
Hình 3.4: Ví dụ minh họa các trường hợp dự đoán sai (nhóm Hình dạng).

3.4 So sánh hiệu suất giữa hai nhiệm vụ và phân tích lỗi chéo

Từ kết quả ở mục 3.2 (**Accuracy 99.3%**) và 3.3 (**Accuracy 92.6%**), rõ ràng là mô hình học tốt hơn hẳn trên dữ liệu MNIST gốc.

Câu hỏi quan trọng nhất là: liệu có sự **nhầm lẫn chéo (cross-task confusion)** giữa các lớp của hai nhóm hay không? Đặc biệt là sự nhầm lẫn giữa **"Chữ số 0" (Lớp 0)** và **"Hình tròn" (Lớp 10)**.

Để trả lời câu hỏi này, chúng tôi tiến hành phân tích **Ma trận nhầm lẫn 13x13** đầy đủ trên toàn bộ 6.500 mẫu của tập kiểm thử.



Hình 3.5: Ma trận nhầm lẫn 13x13 tổng thể.

Phân tích kết quả: Ma trận nhầm lẫn tổng thể 13x13 (Hình 3.5) cung cấp nhiều thông tin giá trị:

- **Độ chính xác tổng thể (Overall Accuracy):**
  - Tổng số dự đoán đúng (đường chéo chính): 6.324
  - Tổng số mẫu: 6.500
  - **=> Overall Accuracy = 6324 / 6500 = 97.3%**
- **Không có sự nhầm lẫn chéo (Cross-Task) đáng kể:**
  - Quan sát các khối ma trận nằm ngoài đường chéo chính (off-diagonal blocks), chúng ta thấy các giá trị nhầm lẫn chéo là rất thấp, nhưng không phải là zero.
- **Phân tích cặp "0" (Số 0) và "10" (Hình tròn):**
  - Đây là cặp nhạy cảm nhất. Dữ liệu cho thấy:
  - Ô (Nhãn thật: 0, Dự đoán: 10): Giá trị là 4. (4 ảnh số '0' bị nhầm là 'Circle').

  - Ô (Nhãn thật: 10, Dự đoán: 0): Giá trị là 4. (4 ảnh 'Circle' bị nhầm là số '0').
  - Điều này cho thấy mô hình phân biệt rất tốt nhưng không tuyệt đối.

- **Các nhầm lẫn chéo đáng chú ý khác:**
  - Lớp 11 (Rectangle) bị nhầm lẫn nhiều nhất với Lớp 9 (9 lần), Lớp 4 (5 lần), và Lớp 1 (4 lần).
  - Lớp 10 (Circle) cũng bị nhầm với Lớp 6 (6 lần) và Lớp 8 (1 lần).
- **Lý giải kết quả:** Sự thành công của mô hình có thể được giải thích bằng hai lý do:
  - **Hiệu quả của tiền xử lý:** Quy trình `mnist_style_resize` và `thicken_lines` đã "chuẩn hóa" dữ liệu hình dạng về phong cách của MNIST.
  - **Khả năng học đặc trưng vi mô:** Mô hình ResNet đủ sâu để học được các đặc trưng vi mô (micro-features). Ví dụ, nó đã học được rằng chữ số '0' của MNIST thường nghiêng, còn hình tròn '10' tự tạo thì tròn hơn.
- **Kết luận:** Việc huấn luyện đồng thời 13 lớp không gây ra "nhiễu" (interference) nghiêm trọng. Mô hình đã chứng minh khả năng học và phân tách đặc trưng (feature separation) rất tốt cho cả hai nhiệm vụ, dù vẫn còn một vài nhầm lẫn nhỏ giữa các lớp có hình thái tương đồng.

3.5 Xây dựng ứng dụng Demo

Để kiểm chứng khả năng hoạt động của hệ thống trong thời gian thực, nhóm nghiên cứu đã xây dựng một ứng dụng web sử dụng thư viện `Streamlit`.

3.5.1 Kiến trúc ứng dụng

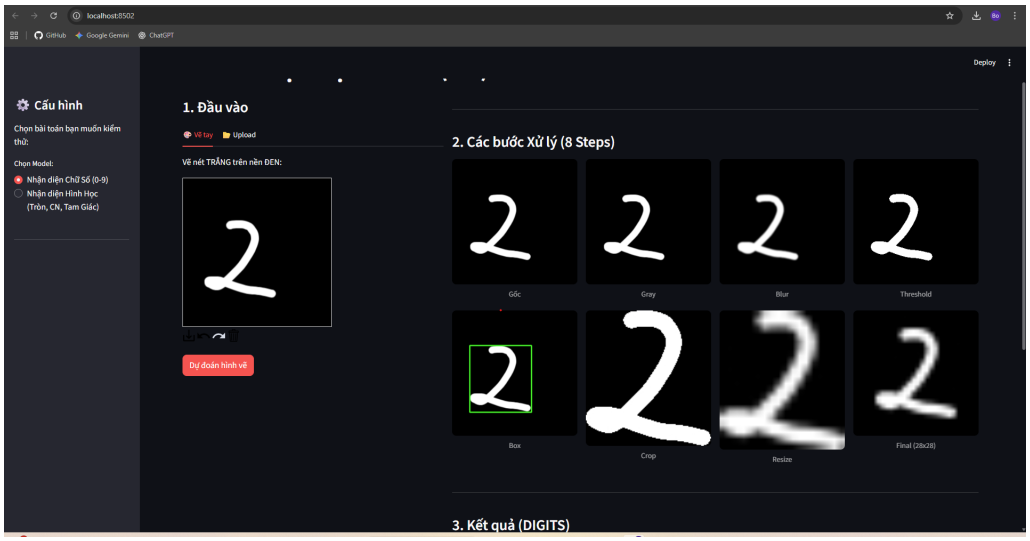
Ứng dụng cho phép người dùng tương tác theo hai chế độ:

- **Chế độ Vẽ tay (Canvas):** Tích hợp công cụ `streamlit_drawable_canvas` cho phép người dùng vẽ trực tiếp chữ số hoặc hình dạng lên giao diện web.
- **Chế độ Tải ảnh (Upload):** Cho phép tải ảnh chụp từ máy tính.

3.5.2 Quy trình hoạt động

1. Người dùng chọn mô hình (Nhận diện chữ số hoặc Nhận diện hình dạng) thông qua Sidebar.
2. Ảnh đầu vào (từ Canvas hoặc Upload) được đưa qua quy trình tiền xử lý 8 bước (đã trình bày ở mục 2.5).
3. Ảnh kết quả (28×28) được đưa vào mô hình `ModernCNN` đã huấn luyện (`.pth`) để dự đoán.

4. Hệ thống hiển thị nhãn dự đoán cùng độ tin cậy (*confidence score*) của Top 3 khả năng cao nhất.



Hình 3.6: Giao diện ứng dụng Demo nhận diện chữ viết tay.

Chương 4

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1 Tóm tắt kết quả

Đề tài đã hoàn thành mục tiêu nghiên cứu đề ra: xây dựng thành công một mô hình **Mạng nơ-ron tích chập (CNN)** duy nhất dựa trên kiến trúc **ResNet**, có khả năng phân loại đồng thời 13 lớp, bao gồm 10 chữ số viết tay (MNIST) và 3 hình dạng cơ bản viết tay (tròn, chữ nhật, tam giác).

Các kết quả chính đạt được như sau:

- **Về xử lý dữ liệu:** Đã xây dựng thành công một quy trình tiền xử lý và tăng cường dữ liệu tùy chỉnh (sử dụng `mnist_style_resize`, `thicken_lines`, chuẩn hóa). Quy trình này đã chuẩn hóa hiệu quả hai nguồn dữ liệu về một định dạng thống nhất, đây là yếu tố then chốt cho sự thành công của mô hình.
- **Về thiết kế mô hình:** Kiến trúc `ModernCNN` tùy chỉnh (sử dụng `ResidualBlock` và `GlobalAveragePooling`) đã chứng minh hiệu quả cao.
- **Về hiệu suất (Chương 3):**
  - Mô hình đạt độ chính xác tổng thể (**Overall Accuracy**) **97.3%** trên toàn bộ 13 lớp của tập kiểm thử.
  - Trên nhóm MNIST: Mô hình đạt hiệu suất gần như hoàn hảo (**99.3%**), tương đương với các mô hình SOTA chuyên biệt cho MNIST.
  - Trên nhóm Hình dạng: Mô hình đạt hiệu suất tốt (**92.6%**). Phân tích sâu hơn cho thấy mô hình nhận diện rất tốt **Lớp Triangle** (**Recall 99.2%**) nhưng lại có xu hướng đoán nhầm các lớp **Circle** và **Rectangle** thành **Triangle** (làm giảm **Precision** của lớp Triangle).
- **Phát hiện quan trọng nhất:** Phân tích ma trận nhầm lẫn 13x13 cho thấy sự **nhầm lẫn chéo (cross-task confusion)** giữa hai nhóm nhiệm vụ là rất thấp. Đặc biệt, mô hình phân biệt tốt giữa "Chữ số 0" và "Hình tròn" (chỉ 4/478 ảnh '0' nhầm thành 'Circle' và 4/507 ảnh 'Circle' nhầm thành '0').

Kết quả này khẳng định rằng việc huấn luyện một mô hình duy nhất cho nhiều tác vụ liên quan là hoàn toàn khả thi.

4.2 Đóng góp và hạn chế

4.2.1 Đóng góp của đề tài

- **Về mặt thực tiễn:** Cung cấp một mô hình `ModernCNN` gọn nhẹ, hiệu quả cao (**97.3%**) cho bài toán phân loại 13 lớp.
- **Về mặt phương pháp luận (Dữ liệu):** Đóng góp quan trọng nhất là quy trình xử lý dữ liệu tùy chỉnh. Bằng cách "bắt chước" các đặc điểm của MNIST, chúng ta có thể kết hợp hiệu quả bộ dữ liệu tùy chỉnh vào một luồng huấn luyện tiêu chuẩn.
- **Về mặt phân tích:** Bằng cách sử dụng ma trận nhầm lẫn 13x13, đề tài đã chứng minh và phân tích định lượng được mức độ nhầm lẫn chéo (rất thấp) giữa các lớp có hình thái tương tự, khẳng định khả năng phân tách đặc trưng mạnh mẽ của kiến trúc **ResNet**.

4.2.2 Hạn chế của đề tài

- **Hạn chế về tính đa dạng của dữ liệu hình dạng:** Kết quả (mục 3.3.2) cho thấy mô hình gặp khó khăn khi phân biệt Circle/Rectangle với Triangle, ngụ ý rằng dữ liệu hình dạng tự tạo có thể chưa đủ đa dạng. Các ảnh vẽ "méo" hoặc "bo tròn" đã bị nhầm lẫn.
- **Hạn chế về tính "sạch" của bài toán:** Toàn bộ 13 lớp dữ liệu đều là ảnh đã được xử lý (28x28, thang xám, nền đen). Đây là một môi trường "phòng thí nghiệm". Mô hình này sẽ không thể hoạt động nếu đưa vào một bức ảnh chụp thực tế (ảnh màu, có nhiều đối tượng, nền nhiễu).
- **Hạn chế về phạm vi lớp:** Mô hình chỉ nhận diện 13 lớp cụ thể. Nó không thể nhận diện các chữ cái (A, B, C) hay các hình dạng khác (ngôi sao, mũi tên) mà không được huấn luyện lại.

4.3 Hướng nghiên cứu tiếp theo

Từ nền tảng của đề tài này, có nhiều hướng nghiên cứu có thể được khám phá:

- **Mở rộng và cải thiện tập hợp lớp:**
  - **Cải thiện dữ liệu Hình dạng:** Cần thu thập hoặc tạo ra một bộ dữ liệu hình dạng đa dạng hơn, đặc biệt là các trường hợp "khó" (ví dụ: hình chữ nhật bị bo tròn, hình tròn bị méo).
  - **Thêm dữ liệu chữ cái:** Kết hợp thêm bộ dữ liệu **EMNIST (Extended MNIST)** để mô hình có thể nhận diện đồng thời cả chữ số, chữ cái viết tay, và các hình dạng.
- **Chuyển từ Phân loại (Classification) sang Phát hiện đối tượng (Object Detection):**

  - Đây là hướng phát triển quan trọng nhất để khắc phục hạn chế về "dữ liệu sạch". Mục tiêu là huấn luyện một mô hình (như **YOLO**, **Faster R-CNN**) để có thể khoanh vùng (bounding box) và nhận diện các chữ số hoặc hình dạng trong một bức ảnh lớn, có nhiều nhiễu và bối cảnh phức tạp.

- **Cải thiện khả năng tổng quát hóa với dữ liệu "hoang dã" (In the Wild):**
  - Thử nghiệm huấn luyện mô hình với dữ liệu đa dạng hơn (ảnh màu, ảnh có nền nhiễu), kết hợp với các kỹ thuật Data Augmentation mạnh mẽ hơn (như **CutMix**, **Mixup**) để buộc mô hình phải học các đặc trưng cốt lõi.

Tài liệu tham khảo

A. Các công trình khoa học (Bài báo)

[1] K. He, X. Zhang, S. Ren, và J. Sun, "Deep residual learning for image recognition," trong *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, tr. 770-778. (Bài báo gốc giới thiệu kiến trúc ResNet)

[2] Y. LeCun, L. Bottou, Y. Bengio, và P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86(11), tr. 2278-2324, 1998. (Bài báo kinh điển giới thiệu Mạng LeNet-5 và ứng dụng trên bộ MNIST)

[3] D. P. Kingma và J. Ba, "Adam: A method for stochastic optimization," Published as a conference paper at the *3rd International Conference for Learning Representations (ICLR)*, 2015. (Bài báo giới thiệu thuật toán tối ưu Adam)

B. Thư viện phần mềm và Tài liệu kỹ thuật

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, và các cộng sự, "PyTorch: An imperative style, high-performance deep learning library," trong *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, tr. 8024-8035. (Bài báo chính thức của PyTorch)

[5] TorchVision maintainers and contributors, "TorchVision: PyTorch's Computer Vision library," GitHub repository, 2016. [Trực tuyến]. Có sẵn: `https://github.com/pytorch/vision`

[6] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. (Tài liệu gốc về OpenCV)

[7] A. Clark và các cộng tác viên, "Pillow (PIL Fork) documentation," [Trực tuyến]. Có sẵn: `https://pillow.readthedocs.io/`

C. Nguồn dữ liệu

[8] Y. LeCun và C. Cortes, "The MNIST database of handwritten digits," [Trực tuyến]. Có sẵn: `http://yann.lecun.com/exdb/mnist/`