HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÁO CÁO BÀI TẬP LỚN

ĐỀ TÀI 6: NHẬN DẠNG CHỮ VIẾT VÀ HÌNH ẢNH ĐƠN GIẢN BẰNG MẠNG NEURAL

Mã môn học : **INT13146**

Môn học : **Xử lý ảnh**

Giảng viên hướng dẫn : **GV. Phạm Hoàng Việt**

Nhóm lớp : **Nhóm 02**

Nhóm bài tập lớn : **Nhóm 11**

Danh sách sinh viên :

Hoàng Cao Nguyên : B22DCCN589

Hoàng Văn Khởi : B22DCCN469

Phạm Thành Long : B22DCCN505

Hà Nội, tháng 11 năm 2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

BẢNG PHÂN CHIA NHIỆM VỤ

STT	Mã sinh viên	Họ và tên	Nội dung phụ trách
1	B22DCCN589	Hoàng Cao Nguyên	- Tiền xử lý - Viết báo cáo
2	B22DCCN469	Hoàng Văn Khởi	- Xây dựng và huấn luyện mô hình nhận dạng
3	B22DCCN505	Phạm Thành Long	- Xây dựng ứng dụng tool

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong đời sống hiện nay, khối lượng thông tin dưới dạng hình ảnh và chữ viết tay ngày càng nhiều, đòi hỏi các hệ thống có khả năng xử lý và nhận dạng tự động. Chữ viết tay là một phương thức giao tiếp quen thuộc của con người, tuy nhiên do sự đa dạng về nét chữ, kích thước và cách viết, việc máy tính có thể hiểu và nhận dạng chính xác chữ viết tay vẫn còn là một thách thức lớn. Đây là vấn đề có tính cấp thiết, đặc biệt trong các ứng dụng như số hóa tài liệu, kiểm tra và chấm thi tự động, nhận dạng chữ viết trên thiết bị di động, hay hỗ trợ người khuyết tật trong việc giao tiếp.

Bên cạnh đó, việc phát hiện và phân loại các hình học cơ bản (ví dụ: hình tròn, hình chữ nhật) cũng giữ vai trò quan trọng trong nhiều hệ thống thị giác máy tính. Đây là bước nền tảng cho các bài toán phức tạp hơn như nhận dạng vật thể, phân tích hình ảnh y tế, hay trong các lĩnh vực công nghiệp như kiểm tra chất lượng sản phẩm và robot tự hành.

Nếu các bài toán nhận dạng chữ viết tay và hình học cơ bản được giải quyết hiệu quả, chúng sẽ mang lại lợi ích to lớn không chỉ trong lĩnh vực giáo dục và hành chính mà còn có thể mở rộng ứng dụng sang y tế, giao thông thông minh, an ninh, và nhiều lĩnh vực khác của đời sống hiện đại. Chính vì vậy, việc nghiên cứu và tìm hiểu các phương pháp nhằm giải quyết hai bài toán trên là hết sức cần thiết và có giá trị thực tiễn cao.

1.2 Mục tiêu và phạm vi đề tài

Trong lĩnh vực nhận dạng chữ số viết tay và hình dạng cơ bản, các sản phẩm dành cho người dùng cuối đã đạt được nhiều tiến bộ đáng kể, đáp ứng nhu cầu đa dạng trong giáo dục, doanh nghiệp và thiết kế sáng tạo. Các ứng dụng phổ biến bao gồm Google Keep, Microsoft OneNote, Evernote, ABBYY FineReader và Adobe Scan cho nhận dạng chữ số viết tay, cùng với Google Lens, CamScanner, PhotoMath và Concepts cho nhận dạng hình dạng cơ bản. Những sản phẩm này tận dụng công nghệ nhận dạng ký tự quang học (OCR) và thị giác máy tính, tích hợp các mô hình học máy như mạng nơ-ron tích chập (CNN) để đạt độ chính xác cao, thường trên 95% trong các điều kiện lý tưởng. Các sản phẩm này được thiết kế với giao diện thân thiện, hỗ trợ đa nền tảng (di động, desktop, web) và thường cung cấp phiên bản miễn phí để thu hút người dùng phổ thông.

Nhu cầu của người dùng cuối đối với các sản phẩm này tập trung vào tính dễ sử dụng, độ chính xác cao và khả năng di động. Trong giáo dục, các ứng dụng như PhotoMath đáp ứng nhu cầu nhận dạng hình dạng cơ bản trong bài toán hình học, hỗ trợ học sinh và giáo viên giải bài tập một cách trực quan. Trong doanh nghiệp, ABBYY FineReader và Adobe Scan được ưa chuộng để số hóa tài liệu, nhận dạng chữ số viết tay trong hóa đơn hoặc hợp đồng, góp phần nâng cao hiệu quả quản lý dữ liệu. Ngoài ra, các ứng dụng như Google Lens cung cấp khả năng nhận dạng hình dạng thời gian thực, phù hợp cho các ứng dụng sáng tạo và giám sát. Người dùng cũng mong

đợi chi phí thấp hoặc miễn phí, cùng với khả năng tích hợp đám mây để đồng bộ dữ liệu, đặc biệt trong bối cảnh làm việc từ xa và học trực tuyến ngày càng phổ biến.

Trong phạm vi chủ đề bài tập lớn, báo cáo sẽ trình bày quá trình xây dựng một công cụ nhận dạng chữ số viết tay và hình dạng cơ bản, đi từ việc xử lý dữ liệu, huấn luyện mô hình nhận dạng, cho đến ứng dụng demo hoàn chỉnh. Trong đó tập trung vào giai đoạn tiền xử lý với việc áp dụng các kĩ thuật xử lý ảnh lên dữ liệu đầu vào nhằm tăng cường hiệu quả huấn luyện mô hình CNN.

1.3 Giới thiệu bài toán nhận dạng chữ số viết tay và hình dạng cơ bản

Nhận dạng hình ảnh được định nghĩa như một quá trình trong lĩnh vực thị giác máy tính, nơi các hệ thống phân tích mẫu pixel để xác định và phân loại đối tượng trong hình ảnh. Quá trình này bao gồm việc xử lý hình ảnh để nâng cao chất lượng, sau đó áp dụng các thuật toán để nhận diện đặc trưng. Thị giác máy tính sử dụng nhận dạng hình ảnh để mô tả chính xác và hiệu quả nội dung hình ảnh, với các ứng dụng trong an ninh và giám sát. Các phương pháp tiếp cận bao gồm học máy và học sâu, nơi mạng nơ-ron được huấn luyện trên dữ liệu lớn để cải thiện độ chính xác.

Lịch sử của nhận dạng hình ảnh bắt nguồn từ những nghiên cứu ban đầu trong xử lý hình ảnh, phát triển thành công nghệ thương mại từ những năm 1950. Nhận dạng hình ảnh được áp dụng rộng rãi trong các sản phẩm như tìm kiếm hình ảnh trên Google và gắn thẻ khuôn mặt trên Facebook. Trong lĩnh vực y tế và giao thông, công nghệ này hỗ trợ phát hiện đối tượng và phân tích dữ liệu hình ảnh, góp phần vào việc tự động hóa quy trình.

Bài toán nhận dạng chữ số viết tay tập trung vào việc phân loại các chữ số từ 0 đến 9 trong hình ảnh viết tay, thường sử dụng cơ sở dữ liệu MNIST làm tiêu chuẩn. Cơ sở dữ liệu này chứa 60.000 mẫu huấn luyện và 10.000 mẫu kiểm tra, mỗi hình ảnh là ảnh xám 28x28 pixel. Các mô hình học sâu như mạng nơ-ron tích chập đạt độ chính xác cao trên tập dữ liệu này, được sử dụng trong các cuộc thi như trên Kaggle để đánh giá hiệu suất.

Nhận dạng hình dạng trong xử lý hình ảnh liên quan đến việc xác định và khớp các hình dạng cơ bản như hình vuông, hình tròn hoặc ngũ giác. Phương pháp sử dụng các thư viện như OpenCV để phát hiện hình dạng hình học đơn giản thông qua ngưỡng hóa và phân tích đường viền. Quá trình này bao gồm biểu diễn hình dạng bằng chuỗi giá trị đặc trưng, hỗ trợ trong nhận diện đối tượng và phân đoạn dựa trên mô hình.

Nhận dạng ký tự, hay còn gọi là nhận dạng ký tự quang học (OCR), là lĩnh vực nghiên cứu trong nhận dạng mẫu và trí tuệ nhân tạo, nơi văn bản từ hình ảnh được chuyển đổi thành định dạng máy đọc được. Công nghệ OCR bắt đầu từ những năm 1950 với hệ thống đọc phông chữ cụ thể cho ngân hàng và bưu điện, phát triển sang các ứng dụng số hóa sách báo và tài liệu lưu trữ.

Tiến bộ gần đây trong nhận dạng hình ảnh tích hợp học sâu để xử lý các bài toán phức tạp hơn, từ chữ số viết tay đến ký tự in ấn, với độ chính xác được cải thiện qua dữ liệu lớn và thuật toán tiên tiến. Các ứng dụng thực tế bao gồm tự động hóa nhập liệu và phân tích tài liệu, chứng minh vai trò quan trọng của công nghệ này trong các lĩnh vực khác nhau.

1.4 Bố cục bài báo cáo

Nội dung bài báo cáo được chia làm bốn phần chính, trong đó:

Chương 1: Trình bày các vấn đề, mục tiêu đề tài hướng tới và giới hạn phạm vi bài toán cho sản phẩm cuối cho đề tài này.

Chương 2: Trình bày các nội dung lý thuyết sẽ được áp dụng vào bài toán nhận dạng chữ số viết tay và hình dạng, bao gồm giai đoạn tiền xử lý ảnh, quá trình huấn luyện model và nguyên lý trích xuất đặc trưng của mô hình CNN.

Chương 3: Trình bày cách bước xây dựng công cụ nhận dạng chữ số viết tay và hình dạng (từ ảnh). Đánh giá tổng quan kết quả sau khi thực hiện.

Chương 4: Kết luận chung về đề tài, các điểm đã làm và chưa làm được. Đồng thời đề ra phương hướng phát triển cho đề tài.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Chương này sẽ trình bày những cơ sở lý thuyết quan trọng làm nền tảng cho việc xây dựng hệ thống nhận dạng chữ số viết tay và hình dạng trong đề tài. Nội dung chương tập trung vào các kỹ thuật và mô hình hiện đại trong lĩnh vực xử lý ảnh và học sâu, bao gồm quy trình tiền xử lý dữ liệu ảnh, cách thức huấn luyện mô hình học máy, cũng như cơ chế trích xuất đặc trưng đặc thù của mạng nơ-ron tích chập (Convolutional Neural Network – CNN). Các kiến thức này là cơ sở để thiết kế thuật toán, tối ưu mô hình và triển khai ứng dụng thực tế ở các chương tiếp theo. Chương này không chỉ giúp làm rõ nguyên lý hoạt động của hệ thống mà còn tạo nền tảng để đánh giá và phân tích hiệu quả của mô hình trong quá trình thực nghiệm.

2.1 Các khái niệm quan trọng

Trong mục này, các khái niệm cơ bản và quan trọng liên quan đến quá trình xử lý ảnh và nhận dạng bằng học sâu sẽ được trình bày. Đây là nền tảng giúp hiểu rõ cách hệ thống nhận dạng chữ số viết tay và hình dạng hoạt động, bao gồm nguyên lý của mạng nơ-ron nhân tạo, đặc điểm của mạng CNN, cũng như các thuật ngữ thường gặp trong tiền xử lý ảnh và huấn luyện mô hình.

2.1.1 Mô hình CNN

Mạng nơ-ron tích chập (Convolutional Neural Network – CNN) là một kiến trúc mạng nơ-ron sâu được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc dạng lưới, tiêu biểu là ảnh. CNN sử dụng các lớp tích chập để tự động học và trích xuất các đặc trưng cục bộ, giúp mô hình phát hiện các mẫu quan trọng như cạnh, góc và hình dạng mà không cần thiết kế thủ công đặc trưng đầu vào [1]. CNN được xem là nền tảng cho nhiều thành tựu lớn trong thị giác máy tính từ năm 2012 (khi AlexNet chiến thắng cuộc thi ImageNet) [2].

CNN có đặc trưng với *cấu trúc nhiều tầng (layered structure)* – gồm nhiều tầng xử lý liên tiếp, từ trích chọn đặc trưng cơ bản (cạnh, góc, đường viền) đến đặc trưng phức tạp (hình dạng, đối tượng). Ngoài ra nó còn tính chất *chia sẻ trọng số (weight sharing)* trong cùng một bộ lọc (filter/kernel), các trọng số này sẽ được áp dụng cho toàn bộ ảnh giúp giảm số lượng tham số cần học. Cuối cùng là khả năng *khai thác tính cục bộ (local connectivity)*, mỗi nơ-ron chỉ kết nối với một vùng nhỏ (receptive field), giúp mô hình tập trung vào đặc trưng cục bộ.

Các thành phần chính của CNN bao gồm::

- *Convolutional layer:* dùng các bộ lọc để phát hiện đặc trưng cục bộ.
- *Pooling layer (ví dụ: max pooling):* giảm kích thước dữ liệu, tăng tính bất biến đối với dịch chuyển và giảm chi phí tính toán.
- *Activation function (thường là ReLU):* thêm phi tuyến tính để mô hình hóa các đặc trưng phức tạp.

- *Fully Connected layer:* kết hợp các đặc trưng đã học để đưa ra dự đoán cuối cùng.

Mạng tích chập (CNN) có nhiều ưu điểm nổi bật so với phương pháp học máy truyền thống. Thứ nhất, CNN có khả năng tự động trích chọn đặc trưng, nên không cần thiết kế thủ công các bộ đặc trưng như trước đây. Thứ hai, mạng này đạt độ chính xác cao, đặc biệt trong các bài toán nhận dạng ảnh và các ứng dụng thị giác máy tính. Thứ ba, CNN có khả năng mở rộng rộng rãi, có thể áp dụng cho ảnh, video hoặc các dữ liệu chuỗi như giọng nói và ngôn ngữ tự nhiên. Cuối cùng, nhờ cơ chế chia sẻ trọng số, CNN có thể giảm số lượng tham số, từ đó huấn luyện hiệu quả trên các tập dữ liệu lớn.

Mạng tích chập (CNN) có nhiều ứng dụng quan trọng trong cả thị giác máy tính và các lĩnh vực khác. Trong thị giác máy tính, CNN được sử dụng để nhận dạng chữ viết tay và chữ số, điển hình là tập dữ liệu MNIST, cũng như phân loại ảnh và nhận dạng đối tượng trên tập ImageNet. Ngoài ra, CNN còn hỗ trợ phân đoạn ảnh trong các ứng dụng y tế và xe tự hành, cũng như xử lý video và nhận dạng hành động. Bên cạnh đó, mạng này còn được áp dụng ngoài lĩnh vực thị giác máy tính, chẳng hạn như phân tích tín hiệu âm thanh, chuỗi thời gian và văn bản.

2.2.2 Các thuật ngữ thường gặp trong quá trình huấn luyện mô hình học sâu

- *Epoch*: là một lần mô hình học qua toàn bộ tập dữ liệu huấn luyện. Ví dụ nếu tập có 60.000 ảnh (mẫu) và mô hình được huấn luyện 20 epoch thì mô hình đã nhìn thấy (học qua) toàn bộ dữ liệu 20 lần mỗi lần 60.000 ảnh. Epoch nhiều, mô hình sẽ học kĩ hơn, nhưng cũng vì thế nên dễ gây *Overfitting*.
- *Batch và Batch Size*: Do dữ liệu lớn, mô hình không thể đưa toàn bộ vào mạng cùng lúc, nên phải chia nhỏ thành các phần đưa vào lần lượt, mỗi phần này chính là Batch. Batch Size chính là số mẫu trong 1 Batch. Batch Size nhỏ thì mô hình học chậm, nhưng ổn định, ngược lại lớn thì học nhanh nhưng cần tài nguyên GPU mạnh.
- *Data Augmentation*: Dữ liệu thực tế rất đa dạng: hình bị xoay, lệch, sáng tối khác nhau, bị che khuất một phần… Nhưng dữ liệu train có trong dataset thường rất sạch và đồng nhất khiến mô hình học xong sẽ bị overfit. Data Augmentation đơn giản là kĩ thuật tạo thêm dữ liệu giả nhưng hợp lý ngay trong lúc train thay vì chỉ có 1 ảnh gốc, ta tạo ra hàng chục, hàng trăm phiên bản biến đổi của nó, buộc mô hình phải học bản chất thật sự của đối tượng chứ không học thuộc lòng.
- *Iteration*: Là số lần mô hình cập nhật trong một epoch = tổng số mẫu / Batch Size. Ví dụ 60.000 mẫu, Batch Size = 100 thì Iteration = 600.
- *Learning Rate*: Mức độ điều chỉnh trọng số mô hình trong mỗi lần cập nhật.
- *Loss Function*: Là hàm mất mát đo mức độ sai lệch giữa dự đoán và nhãn thật. Mục tiêu của huấn luyện là giảm loss xuống mức thấp nhất, tức là độ chính xác tốt nhất.

- *Overfitting*: Mô hình học quá kỹ dữ liệu huấn luyện : không tổng quát : Độ chính xác cao trên tập train nhưng thấp trên tập test (validation).
- *Underfitting*: Mô hình học chưa đủ : tỉ lệ chính xác trên tập train và test đều thấp.
- *Regularization*: Là các kĩ thuật chống hiện tượng *Overfitting*
- *Early Stopping*: Là kĩ thuật cho phép dừng huấn luyện mô hình khi chưa hết epoch nhằm tránh tiếp tục huấn luyện khi không cần thiết, tránh *Overfitting*.

2.2 Các kĩ thuật tiền xử lý

Trong mục này sẽ trình bày nội dung về các kĩ thuật xử lý hình ảnh chính được áp dụng trong pipeline xử lý đầu vào trước khi đưa vào mô hình của công cụ. Các nội dung sẽ được sắp xếp theo thứ tự thực hiện, trong đó phần 2.2.1 đến 2.2.6 trình bày các kĩ thuật tiền xử lý chung trên toàn bộ ảnh, những phần còn lại trình bày các kĩ thuật xử lý riêng lẻ trên từng ảnh chữ số (sau khi được cắt ra từ ảnh chính).

2.2.1 Chuyển đổi từ ảnh màu thành ảnh xám

Khái niệm: *Chuyển ảnh sang ảnh xám (Grayscale Conversion) là kỹ thuật trong xử lý ảnh, trong đó ảnh màu RGB (Red, Green, Blue) được biến đổi thành ảnh xám (grayscale) bằng cách kết hợp các giá trị của ba kênh màu thành một giá trị duy nhất. Mục đích là giảm độ phức tạp tính toán và giữ lại thông tin về cường độ ánh sáng và cấu trúc hình ảnh, giúp các bước xử lý ảnh tiếp theo hoặc các thuật toán học máy/trí tuệ nhân tạo hoạt động hiệu quả hơn* [3].

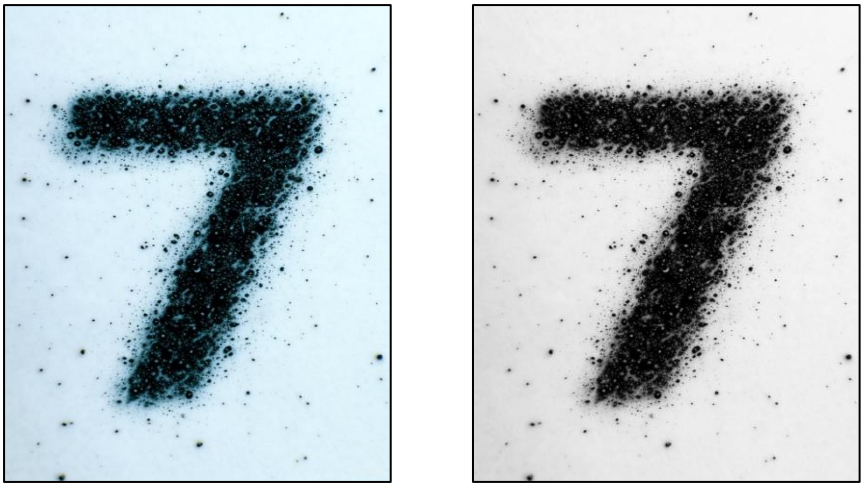
Công thức: Công thức phổ biến để chuyển một pixel ảnh màu RGB sang ảnh xám grayscale là tính giá trị xám Y dựa trên tỷ trọng của ba kênh màu [3], với công thức như sau:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Trong đó:

- R,G,B là giá trị đỏ, xanh lá, xanh dương của pixel (thường từ 0–255).
- Y là giá trị xám tương ứng (0–255).
- Các trọng số 0.299, 0.587, 0.114 là hệ số trọng số phản ánh mức độ nhạy sáng của mắt người với mỗi kênh màu: mắt nhạy nhất với xanh lá (G), ít nhạy với xanh dương (B). Các giá trị này được tham khảo từ tiêu chuẩn NTSC [4].

Ví dụ:



Hình 1: Ảnh trước và sau khi chuyển xám

```python
# 1. Đọc ảnh grayscale
def read_grayscale(path):
    """Đọc ảnh và chuyển sang grayscale theo công thức chuẩn"""
    img = Image.open(path).convert('RGB')
    arr = np.array(img, dtype=np.uint8)
    R = arr[:, :, 0].astype(np.float32)
    G = arr[:, :, 1].astype(np.float32)
    B = arr[:, :, 2].astype(np.float32)
    gray = 0.299 * R + 0.587 * G + 0.114 * B
    gray = np.clip(np.round(gray), 0, 255).astype(np.uint8)
    return gray
```

Hình 2: Mã nguồn xử lý chuyển đổi ảnh xám (python)

2.2.2 Loại bỏ nhiễu muối tiêu bằng bộ lọc trung vị

Khái niệm: *Bộ lọc trung vị (Median Filter) là một kỹ thuật lọc không gian (spatial filter) trong xử lý ảnh dùng để loại bỏ nhiễu "salt & pepper" hoặc các nhiễu đột biến mà vẫn giữ nguyên biên (edges) của ảnh [3, pp. 185-188]. Khác với bộ lọc trung bình (mean filter), bộ lọc trung vị không làm mờ các cạnh sắc nét. Thường áp dụng cho ảnh xám hoặc nhị phân.*

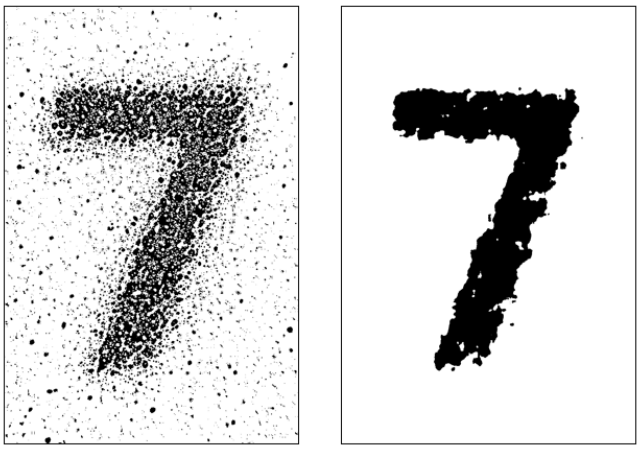
Công thức:

$$I'(x,y) = median\{ I(i,j) \mid (i,j) \in neighborhood\}$$

Trong đó:

- $I'(x,y)$ là giá trị pixel sau lọc
- Neighborhood là vùng lân cận k x k quanh pixel (x,y)

Ví dụ:



Hình 3: Ảnh trước và sau khi đi qua bộ lọc trung vị

```python
# 2. Lọc trung vị (tùy chọn - có thể bỏ qua nếu ảnh sạch)
def median_filter(gray_image, ksize=3):
    """Lọc trung vị để giảm nhiễu"""
    pad = ksize // 2
    padded = np.pad(gray_image, pad, mode='edge')
    H, W = gray_image.shape
    output = np.zeros_like(gray_image)
    for i in range(H):
        for j in range(W):
            window = padded[i:i+ksize, j:j+ksize]
            output[i, j] = np.median(window)
    return output
```

Hình 4: Mã nguồn mô phỏng bộ lọc trung vị (python)

2.2.3 Tách ngưỡng theo Otsu Threshold

Khái niệm: *Otsu Threshold là một phương pháp được dùng để tự động nhị phân hóa ảnh. Thuật toán trả về một ngưỡng cường độ duy nhất để phân chia các pixel thành hai lớp: tiền cảnh (foreground) và nền (background). Ngưỡng này được chọn sao cho độ biến thiên trong mỗi lớp (nền hoặc vật thể) nhỏ nhất, hoặc tương đương là độ khác biệt (biến thiên giữa các lớp) lớn nhất [5].*

Hay nói một cách đơn giản, đây là thuật toán chọn ngưỡng tự động từ ảnh xám đầu vào với mục tiêu một giá trị ngưỡng tối ưu để chia histogram của ảnh thành 2 nhóm có khác biệt rõ rệt nhất mà không cần thử nghiệm thủ công từng ngưỡng.

Công thức:

$$\sigma^2 = w_B . w_F . (\mu_B - \mu_F)^2$$

Trong đó:

- σ^2: giá trị phương sai dùng để đánh giá độ khác biệt

- w_B: số pixel thuộc nhóm Backgound.
- w_F: số pixel thuộc nhóm Foregound
- μ_B: giá trị pixel trung bình nhóm Backgound
- μ_F: giá trị pixel trung bình nhóm Foregound

Thuật toán:

– B1: Duyệt qua tất cả ngưỡng t từ (0 – 255)

– B2: Với mỗi t, xác định được 2 nhóm pixel

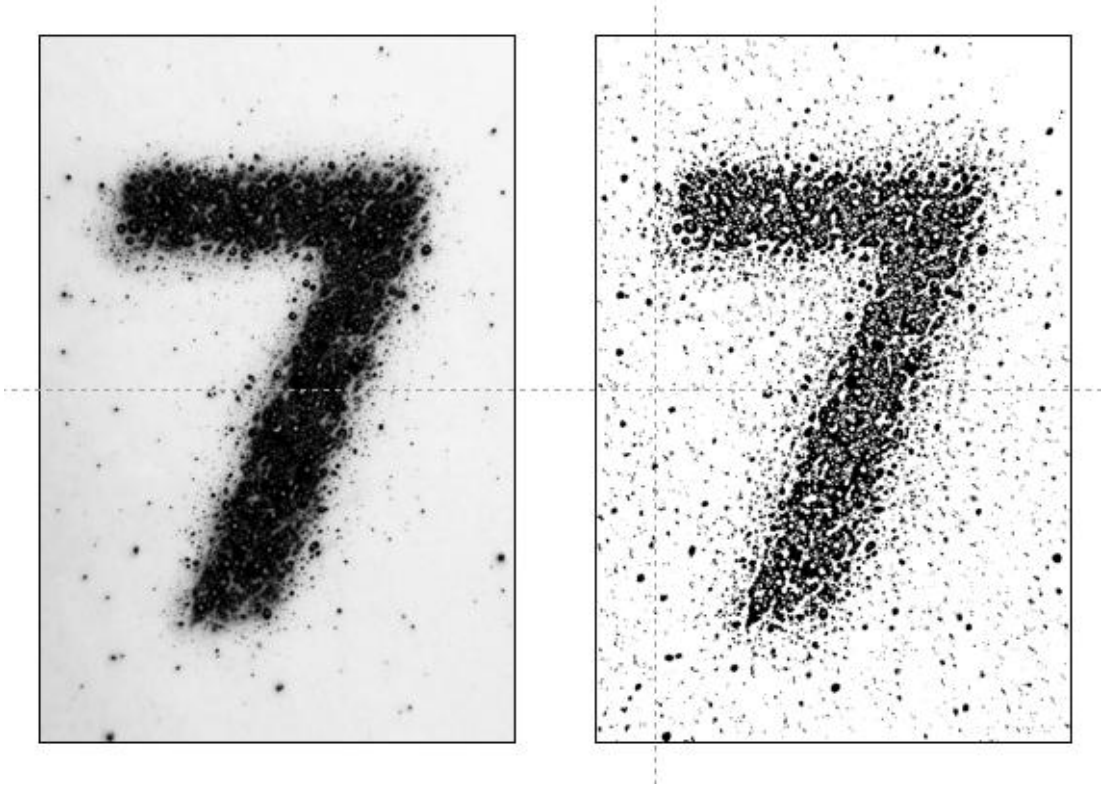
  - Backgound (value <= t)
  - Foregound (value > t)

– B3: Tính σ^2 theo công thức, lưu lại t nếu σ^2 là lớn nhất, quay lại bước 1.

– B4: Trả về t là ngưỡng cần tìm

Sau khi nhận được ngưỡng tối ưng tiến hành tách ngưỡng thông thường để tạo ảnh nhị phân với 2 mức xám.

Ví dụ:



Hình 5: Ảnh trước và sau khi tách ngưỡng

```python
def otsu_threshold_binary(gray_image):
    """Tính ngưỡng Otsu và tạo ảnh nhị phân"""
    hist = np.bincount(gray_image.flatten(), minlength=256)
    total_pixels = gray_image.size
    total_pixel_values = np.sum(np.arange(256) * hist)

    sumB = 0
    wB = 0
    max_var = 0
    threshold = 0

    for t in range(256):
        wB += hist[t]
        if wB == 0:
            continue
        wF = total_pixels - wB
        if wF == 0:
            break

        sumB += t * hist[t]
        mB = sumB / wB
        mF = (total_pixel_values - sumB) / wF
        var_between = wB * wF * (mB - mF) ** 2

        if var_between > max_var:
            max_var = var_between
            threshold = t

    # Tạo ảnh nhị phân
    binary_image = np.zeros_like(gray_image, dtype=np.uint8)
    binary_image[gray_image >= threshold] = 255

    return binary_image
```

Hình 6: Mã nguồn xử lý tách ngưỡng ảnh xám (python)

2.2.4 Tìm biên thông qua DFS/BFS và xác định BoundingBox

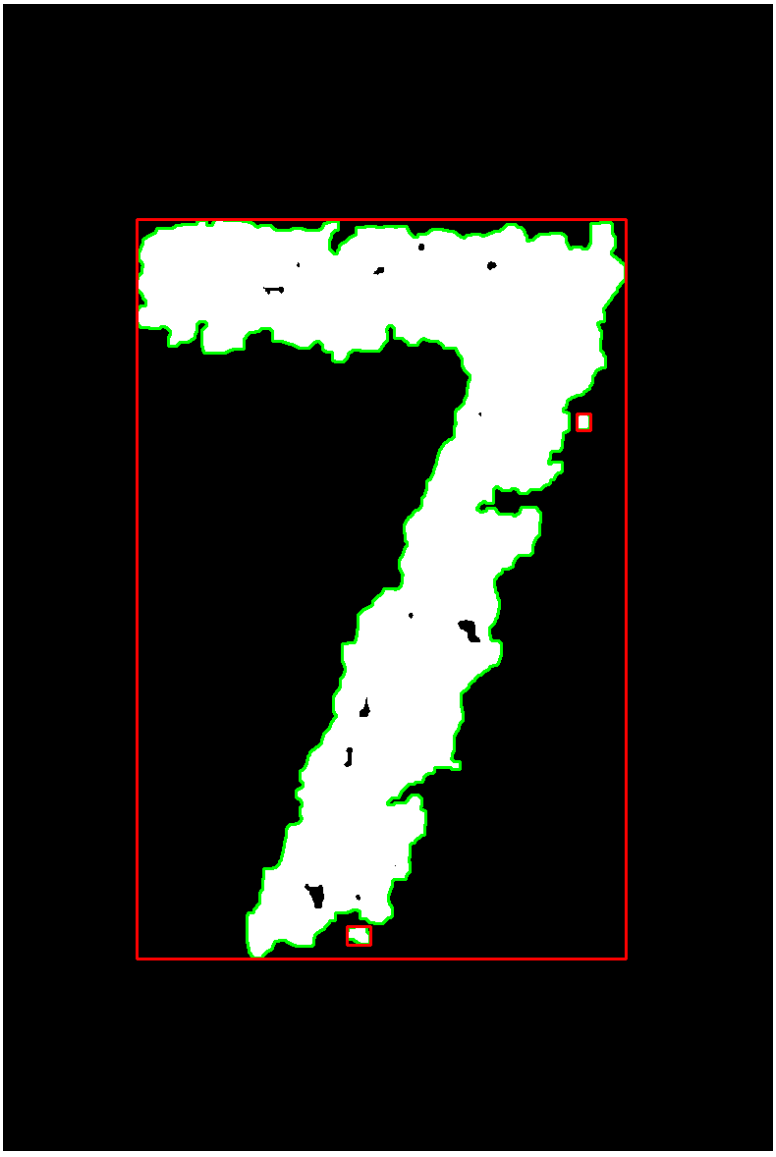
Khái niệm:

- *DFS/BFS*: hai thuật toán duyệt đồ thị hoặc cây phổ biến, trong đó BFS khám phá các đỉnh theo chiều rộng bằng cách duyệt lần lượt từng tầng từ gần đến xa, còn DFS lại đi theo chiều sâu, ưu tiên đi sâu nhất vào một nhánh trước khi quay lại các nhánh khác; cả hai đều được dùng để tìm kiếm, phân tích cấu trúc dữ liệu và xác định các vùng liên thông trong nhiều bài toán từ đồ thị cho đến xử lý ảnh.
- *BoundingBox*: là một hình chữ nhật nhỏ nhất bao trọn toàn bộ đối tượng trong ảnh. Nó xác định vị trí của một vật thể bằng 4 giá trị x, y, width, height (x, y là tọa độ điểm cao nhất bên trái của ảnh, width và height là chiều cao và kích thước của ảnh).
- *Vùng liên thông*: Là tập hợp các điểm nằm kề nhau có cùng giá trị trong đồ thị, trong xử lý ảnh nó tương đương với các vùng có các pixel có cùng giá trị nằm kề nhau.

Trước khi xác định biên, cần phải nhị phân hóa ảnh xám và xử lý nhiễu. Sau bước này, ảnh đã chỉ gồm 2 mức xám, dễ dàng thực hiện dùng thuật toán BFS hoặc DFS để duyệt tìm được vùng pixel của đối tượng theo nguyên tắc sau:

- Duyệt toàn bộ pixel của ảnh.
- Khi gặp pixel có giá trị = 255 (quy ước giá trị pixel của đối tượng) thì dùng BFS/DFS để tìm vùng liên thông chứa pixel này và lưu tại các tọa độ của mỗi vùng.
- Các pixel duyệt rồi sẽ được bỏ qua.

Cuối cùng, với mỗi vùng liên thông tìm 4 điểm tọa độ (minX, minY), (maxX, minY), (maxX, maxY), (minX, maxY) tạo thành BoundingBox cho mỗi hình.



Hình 7: Minh họa Bounding box

```python
# 4. Tìm connected components
def connected_components(binary):
    """Tìm các vùng liên thông trong ảnh nhị phân"""
    H, W = binary.shape
    visited = np.zeros_like(binary, dtype=bool)
    components = []

    for y in range(H):
        for x in range(W):
            if binary[y, x] == 255 and not visited[y, x]:
                stack = [(x, y)]
                visited[y, x] = True
                pixels = []

                while stack:
                    px, py = stack.pop()
                    pixels.append((px, py))

                    for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
                        nx, ny = px + dx, py + dy
                        if 0 <= nx < W and 0 <= ny < H:
                            if binary[ny, nx] == 255 and not visited[ny, nx]:
                                visited[ny, nx] = True
                                stack.append((nx, ny))

                components.append(pixels)

    return components

# 5. Crop component
def crop_component(binary, component):
    """Cắt vùng chứa component từ ảnh"""
    xs = [p[0] for p in component]
    ys = [p[1] for p in component]
    x1, x2 = min(xs), max(xs)
    y1, y2 = min(ys), max(ys)
    return binary[y1:y2+1, x1:x2+1].copy()
```

Hình 8: Mã nguồn minh họa cắt ảnh bằng BoundingBox + DFS (python)

2.2.5 Tách các đối tượng ra khỏi ảnh gốc

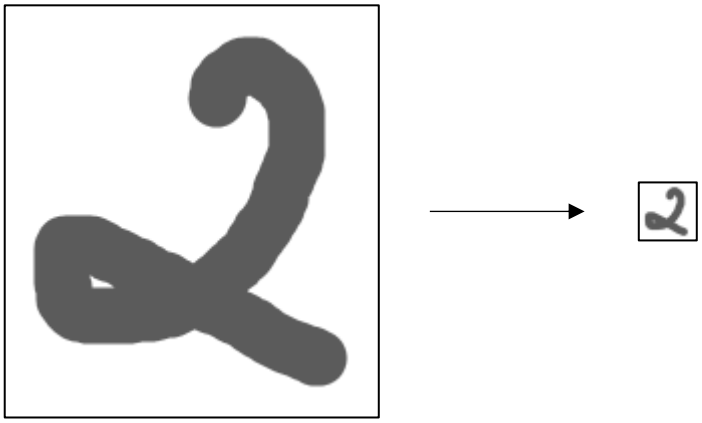
Dựa trên tọa độ các Bounding box có được từ bước trước, ta dễ dàng có thể cắt các phần ảnh chứa chữ số/hình dạng ra khỏi ảnh gốc, tuy nhiên cần lưu ý một số điều sau:

- Thứ tự ảnh cần được xác định theo tọa độ của các Bounding box so với ảnh gốc: Ví dụ xác định theo tọa độ góc trên cùng bên trái, theo thứ tự tuần từ từ trên xuống dưới, từ trái qua phải.
- Các Bounding box có kích thước nhỏ hơn đáng kể so với kích thước trung bình của các Bounding box khác cần loại bỏ để tránh còn tồn tại nhiễu

2.2.6 Chuẩn hóa kích thước ảnh

Khi có được các ảnh sau khi tách ra từ ảnh gốc, vì là chữ số viết tay/hình dạng vẽ tay nên kích thước mỗi ảnh là khác nhau, cần phải chuẩn hóa để kích thước mỗi ảnh là giống nhau (thường là 28x28). Quá trình này được thực hiện theo 2 bước sau:

- *Resize*: Tăng/giảm tỉ lệ ảnh cho đến khi chiều dài/chiều rộng bằng 20, đảm bảo các ảnh giữ đúng tỉ lệ giống với nhau không bị méo. Sử dụng phương pháp nearest neighbor với đặc điểm:
    - Khi muốn tạo một pixel mới trong ảnh resized, lấy giá trị của pixel gốc gần nhất với vị trí tương ứng của nó.
    - Không tính trung bình hay nội suy giữa các pixel lân cận => giữ nguyên giá trị nguyên bản của pixel gốc
- *Padding*: Thêm các pixel còn thiếu vào ảnh cho đến khi chiều dài bằng chiều rộng = 28 (Thêm một chút phần thừa để nhận diện tốt hơn). Để đảm bảo không làm ảnh hưởng đến quá trình trích xuất đặc trưng của mô hình CNN, giá trị padding của pixel phải trùng với giá trị pixel của màu nền (0 hoặc 255 tùy từng trường hợp).



Hình 9: Ảnh trước và sau khi resize và padding

```python
# 6. Resize với nearest neighbor
def resize_nearest(img, new_w, new_h):
    """Resize ảnh bằng phương pháp nearest neighbor"""
    h, w = img.shape

    if new_h <= 0 or new_w <= 0:
        return np.zeros((max(1, new_h), max(1, new_w)), dtype=np.uint8)

    y_scale = h / new_h
    x_scale = w / new_w

    resized_img = np.zeros((new_h, new_w), dtype=np.uint8)

    for i in range(new_h):
        for j in range(new_w):
            src_y = min(int(i * y_scale), h - 1)
            src_x = min(int(j * x_scale), w - 1)
            resized_img[i, j] = img[src_y, src_x]

    return resized_img
```

Hình 10: Mã nguồn mô phỏng resize cho ảnh (python)

```python
# Padding về 28x28
pad_h = 28 - new_h
pad_w = 28 - new_w

pad_top = pad_h // 2
pad_bottom = pad_h - pad_top
pad_left = pad_w // 2
pad_right = pad_w - pad_left

digit = np.pad(
    digit,
    ((pad_top, pad_bottom), (pad_left, pad_right)),
    mode='constant',
    constant_values=0
)
```

Hình 11: Mã nguồn mô phỏng padding cho ảnh đơn (python)

2.2.7 Căn giữa trọng tâm ảnh

Sau khi chuẩn hóa kích thước, việc tiếp theo cần làm là căn giữa hình ảnh được thực hiện thông qua 4 bước sau:

- *Tính trọng tâm (cx, cy) của đối tượng*: Là tọa độ trung bình của tất cả pixel có mức xám > 0 hoặc < 255 tùy từng trường hợp, tính theo công thức sau:

$$cx = \frac{\sum x_i I(x_i, y_i)}{\sum I(x_i, y_i)}$$

$$cy = \frac{\sum y_i I(x_i, y_i)}{\sum I(x_i, y_i)}$$

Với $I(x_i,y_i)$ là giá trị xám của pixel

- *Tính tọa độ tâm của ảnh*: center_x = w/2, center_y = h/2
- *Xác định độ dịch chuyển*: dx = center_x – cx, dy = center_y – cy
- *Dùng affine transform để dịch toàn bộ ảnh theo dx, dy.*

```python
# 8. Center of mass
def center_of_mass(binary):
    """Tính trọng tâm của ảnh"""
    ys, xs = np.nonzero(binary)
    if len(xs) == 0:
        return binary.shape[1] // 2, binary.shape[0] // 2
    cx = np.mean(xs)
    cy = np.mean(ys)
    return cx, cy

# 9. Shift ảnh
def shift_image(img, sx, sy):
    """Dịch chuyển ảnh theo vector (sx, sy)"""
    h, w = img.shape
    new = np.zeros_like(img)

    for y in range(h):
        for x in range(w):
            nx = x + sx
            ny = y + sy
            if 0 <= nx < w and 0 <= ny < h:
                new[ny, nx] = img[y, x]

    return new
```

Hình 12: mã nguồn mô phỏng tính trọng tâm và dịch ảnh (python)

2.2.8 Chuẩn hóa đầu vào cho mô hình huấn luyện

Chuẩn hóa đầu vào là quá trình biến đổi dải giá trị pixel từ khoảng [0 => 255] của ảnh xám sang một phạm vi nhỏ hơn và ổn định hơn như [0 => 1] hoặc phân phối chuẩn (z-score). Điều này giúp giảm độ lớn của gradient, tránh hiện tượng lan truyền sai số quá mạnh hoặc quá yếu trong mạng CNN và làm tăng tốc độ hội tụ khi huấn luyện.

Có 2 phương pháp chuẩn hóa chính:

- *Chuẩn hóa về khoảng 0–1 (sử dụng nhiều nhất trong MNIST)*:

$$I_{norm} = \frac{I}{255}$$

Với I là các pixel gốc

- *Chuẩn hóa theo phân phối chuẩn (mean–std normalization)*:

$$I_{norm} = \frac{I - \mu}{\sigma}$$

Bước cuối cùng trong quá trình chuẩn hóa chính là reshape từ 28x28 thành 28x28x1. Lý do phải thực hiện điều này là do mô hình CNN yêu cầu đầu vào phải là ảnh 3 chiều có dạng (height, width, channels). Do dùng ảnh xám nên số channel = 1.

2.3 Trích xuất đặc trưng (CNN)

Một trong những ưu điểm nổi bật nhất của Convolutional Neural Network (CNN) là khả năng tự động học và trích xuất đặc trưng (feature extraction) trực tiếp từ dữ liệu ảnh mà không cần các bước thủ công như trong các phương pháp truyền thống (SIFT, HOG, SURF). Quá trình trích xuất đặc trưng của CNN diễn ra xuyên suốt qua hệ thống nhiều tầng tích chập và phi tuyến, tạo ra biểu diễn (representation) ngày càng trừu tượng và giàu thông tin hơn.

2.3.1 Học đặc trưng cục bộ qua lớp Convolution

Ở những lớp này, CNN sẽ học được các đặc điểm đơn giản như: cạnh, đường thẳng, góc, độ sáng tối, họa tiết nhỏ, … Để làm được như vậy, CNN sẽ sử dụng các kernel để nhân tích chập lần lượt qua hết tấm ảnh. Các kernel này có cấu trúc nhất định và kết quả tích chập sẽ phản ánh mức độ phù hợp với kernel và từ đó mô hình sẽ nhận ra được đặc điểm của vùng vừa được tích chập. Ví dụ với một kernel phát hiện cạnh dọc, cấu trúc của nó sẽ như sau:

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

=> Khi tích chập với ảnh, kernel này làm nổi bật những vùng có sự thay đổi mạnh theo trục ngang - tương ứng với cạnh dọc.

Các bộ lọc kernel của CNN không được thiết kế thủ công ngay từ đầu mà được khởi tạo ngẫu nhiên, và liên tục thay đổi các trọng số của mình qua các gradient descent để sao cho phù hợp nhất với ảnh đang xét, vì vậy CNN có thể học được mọi loại đặc trưng phức tạp, từ cạnh cho đến hình dạng đặc thù của chữ số hoặc ký tự viết tay.

2.3.2 Phi tuyến hóa bằng hàm kích hoạt ReLU

Sau tích chập, ảnh được đưa qua hàm kích hoạt ReLU có công thức như sau:

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

Mục đích của hàm này là loại bỏ các giá trị âm, tức là bỏ các giá trị tích chập thấp do ít tương tác mạnh với kernel, chỉ giữ lại các giá trị cao ~ các đặc trưng có tính tương tác mạnh với kernel. Điều này sẽ giúp mô hình học tập chung hơn vào cáo đặc trưng quan trọng của ảnh (mẫu).

2.3.3 Giảm chiều bằng Pooling

Ở lớp này, CNN sẽ dùng 1 cửa sổ trượt đặc biệt có nhiệm vụ tìm 1 giá trị đặc trưng duy nhất có trong 1 vùng có kích thước cố định cuả feature map. Cửa sổ này sẽ trượt lần lượt qua hết ảnh, từ đó thu được ma trận mới có kích thước nhỏ hơn gồm các giá trị đặc trưng đã thu được. Điều này giúp giảm chi phí tính toán cho các lớp sau, giảm nhiễu và làm đặc trưng nổi bật trở nên bất biến đối với các thay đổi nhỏ. Để xác định được các pixel đặc trưng từ nhóm các pixel, người ta thường dùng các hàm khác nhau như:

- *Max pooling*: Lấy ô có giá trị lớn nhất, giúp giữ các đặc trưng mạnh.
- *Average pooling*: Lấy giá trị trung bình, giúp giữ các đặc trưng tổng quát.
- *Global pooling*: Dùng cửa sổ có kích thước bằng cả feature map, lấy ra 1 giá trị duy nhất, điều này giúp giảm mạnh tham số và overfitting.

2.3.4 Các tầng sâu của CNN

Càng đi sâu, các bộ lọc kernel CNN học được đặc trưng ngày càng trừu tượng. Điều này có được do các lớp tầng bên dưới sử dụng kết quả từ tầng trên để xử lý, mà kết quả mỗi tầng là các đặc trưng nổi bật được trích xuất ra từ feature map. Vì vậy càng xuống sâu, CNN lại càng học được các đặc trưng ngày càng trừu tượng của mẫu.

Ở các lớp đầu, CNN sẽ học được các đặc trưng cơ bản cấp thấp như: cạnh, góc, hướng, đường cong, …. Ở các lớp giữa, nó sẽ học được các đặc trưng trung bình từ những đặc trưng cơ bản, ví dụ như: hình vòng cung, nét chữ , đoạn cong liên tục, …. Và cuối cùng, ở các lớp cuối, CNN sẽ học được đặc trưng tổng thể của đối tượng như cấu trúc hoàn chỉnh cuả các chữ số: 2, 3, 1, .., các hình dạng tổng thể của các hình tròn, vuông, tam giác.

2.3.5 Kết hợp đặc trưng bằng Fully Connected Layer

Sau khi các tầng tích chập và pooling đã rút trích được những đặc trưng quan trọng từ ảnh, mạng CNN sẽ sử dụng Fully Connected Layer (FC) để kết hợp và học mối quan hệ giữa các đặc trưng này trước khi đưa ra dự đoán cuối cùng. Đầu vào của FC là một vector một chiều được tạo ra bằng cách "làm phẳng" (flatten) toàn bộ các feature map ở tầng cuối của khối CNN.

Fully Connected Layer hoạt động tương tự như tầng trong mạng nơ-ron truyền thống, nơi mỗi neuron kết nối với tất cả neuron ở lớp trước; điều này cho phép mô hình học được các sự phụ thuộc phức tạp giữa các đặc trưng đã trích xuất thông qua các trọng số và bias được tinh chỉnh trong quá trình lan truyền ngược. Cuối cùng, lớp Softmax nằm sau FC sẽ chuyển các giá trị đầu ra thành phân phối xác suất trên các lớp, giúp hệ thống CNN đưa ra dự đoán phân loại một cách chính xác.

CHƯƠNG 3: TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

3.1 Xây dựng CNN model

Bước đầu tiên trong quá trình triển khai là tiến hành thiết kế và xây dựng các model dạng CNN tùy chỉnh phù hợp với từng mục tiêu bài toán cần giải quyết đó là nhận dạng chữ số, chữ viết và hình dạng cơ bản.

3.1.1 Model nhận dạng chữ số

Xây dựng model tuần tự Sequential gồm 3 block chính với các layers.

```
model = Sequential([
    # Block 1
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1), padding="same"),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Block 2
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Fully Connected
    Flatten(),
    Dense(256, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),
    Dense(128, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),
    Dense(10, activation="softmax")
])
```

Hình 13: Cấu trúc model nhận dạng chữ số

(*) Block 1: Học đặc trưng cấp thấp (cạnh, góc, texture đơn giản)

- ***Conv2D đầu tiên***: Chỉ học các đường viền, cạnh thẳng, chấm, vùng sáng hoặc tối đơn giản.
- ***BatchNormalization***: Giữ cho các giá trị activation không bị lệch quá xa ngay từ tầng đầu.

- ***Conv2D thứ hai***: Kết hợp các cạnh đơn lẻ thành các mẫu hơi phức tạp hơn: góc nhọn, đường cong ngắn.
- ***BatchNormalization***: Đảm bảo tầng sau nhận dữ liệu đã được chuẩn hóa.
- ***MaxPooling2D***: Làm mờ vị trí chính xác của các cạnh (bắt đầu tạo tính bất biến dịch chuyển nhẹ), đồng thời giảm kích thước từ 28×28 xuống 14×14.
- ***Dropout 0.25***: Ngăn một số filter chỉ học một kiểu cạnh duy nhất, buộc chúng học đa dạng hơn.

(*) Block 2: Phát hiện hình dạng của các chữ số

- ***Conv2D (64 filter) đầu tiên***: Nhận diện được các bộ phận rõ ràng hơn: Nét cong, nét gập liền, …
- ***BatchNormalization***: Giữ cho tầng sâu vẫn huấn luyện ổn định dù số filter đã tăng gấp đôi.
- ***Conv2D thứ hai (64 filter)***: Kết hợp các bộ phận thành hình dạng chữ số.
- ***BatchNormalization***: Tiếp tục chuẩn hóa trước khi giảm kích thước.
- ***MaxPooling2D***: Giảm xuống còn 7×7, lúc này mạng không còn quan tâm vị trí chính xác của các đặc trưng trong ảnh, chỉ cần biết có xuất hiện hay không.
- ***Dropout 0.25***: Ngăn các filter ở tầng sâu ghi nhớ nhiễu hoặc học quá chi tiết không cần thiết.

(*) Block 3: Phân loại cuối

- ***Flatten***: Biến toàn bộ bản đồ đặc trưng 7×7×64 thành một vector dài, lúc này không còn thông tin không gian.
- ***Dense 256***: Tổng hợp lại tất cả các bộ phận đã phát hiện được thành một biểu diễn cấp cao nhất của chữ số.
- ***BatchNormalization***: Ổn định activation trước lớp phân loại quan trọng nhất.
- ***Dropout 0.5***: Cần giảm mạnh do lớp Dense có hàng trăm nghìn trọng số, dễ học nhiễu.
- ***Dense 128***: Nén biểu diễn xuống còn tinh gọn hơn, loại bỏ thông tin thừa.
- ***BatchNormalization***: Chuẩn hóa lần cuối.
- ***Dropout 0.5***: Một lần nữa giảm mạnh buộc mạng không quá dựa vào một vài neuron chính.
- ***Dense 10 + softmax***: Đưa ra xác suất cuối cùng thuộc 10 lớp chữ số.

3.1.2 Model nhận dạng chữ viết

Xây dựng model tuần tự Sequential gồm 4 block chính với các layers

```
model = Sequential([
    # Block 1
    Conv2D(32, (3, 3), activation="relu", padding="same", input_shape=(28, 28, 1)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),

    # Block 2
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),

    # Block 3
    Conv2D(128, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    Conv2D(128, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.4),

    # Fully Connected
    Flatten(),
    Dense(256, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),

    Dense(128, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),
    Dense(26, activation="softmax")
])
```

Hình 14: Cấu trúc model nhận dạng chữ viết

(*) Block 1: Phát hiện các nét cơ bản của chữ cái (đường thẳng, cong, chấm, vòng tròn nhỏ).

- ***Conv2D(32)***: Phát hiện nét ngang, dọc, chéo, vòng cung nhỏ.
- ***BatchNormalization***: Giữ activation ổn định ngay từ đầu.
- ***Conv2D(32) thứ hai***: Kết hợp các nét thành cấu trúc nhỏ: chữ "i" có chấm, chữ "t" có thanh ngang, vòng trên của "b", "d"….
- ***MaxPooling2D***: Giảm từ 28×28 : 14×14, bắt đầu bỏ qua vị trí chính xác của nét.
- ***Dropout 0.25***: Buộc các filter học đa dạng nét, không chỉ chuyên một kiểu.

(*) Block 2: Nhận diện cấu trúc đặc trưng của từng chữ cái (thân chữ, vòng, thanh ngang/dọc)

- ***Conv2D(64):*** Phát hiện rõ hơn: 2 vòng của B, 3 vòng của 8 (nếu số), vòng dưới của g, q, thanh ngang dài của E, F…
- ***Conv2D(64) thứ hai***: Kết hợp thành mẫu nhận diện mạnh: chữ A có tam giác, X có 2 đường chéo giao nhau, chữ C có hình bán nguyệt mở…
- ***MaxPooling2D***: Giảm còn 7×7, không còn quan tâm vị trí chữ nằm chính giữa hay lệch
- ***Dropout 0.25***: Vẫn giữ tính tổng quát ở tầng trung.

(*) Block 3: Phân biệt các chữ rất giống nhau

- ***Conv2D(128) + Conv2D(128):*** Tầng sâu nhất, số filter nhiều nhất : chuyên phân biệt chi tiết cực nhỏ như:
    - B và D: nối giữa
    - U và N: độ cong đáy
    - C và G: có thanh ngang nhỏ hay không
    - V và Y: đuôi có kéo dài không
- ***MaxPooling2D cuối***: Giảm còn 3×3×128 (rất nhỏ nhưng cực kỳ cô đọng)
- ***Dropout 0.4***: Tỉ lệ cao vì tầng này dễ overfit nhất khi phân biệt các chữ gần giống nhau.

(*) Block 4: Phân loại cuối

- ***Dense 256*** : Tổng hợp toàn cục, hiểu được "cấu trúc tổng thể" của chữ cái
- ***BatchNormalization + Dropout 0.5*** : Cần giảm mạnh vì lớp fully-connected có hơn 1 triệu trọng số.
- ***Dense 128*** : Nén thêm lần nữa để loại bỏ nhiễu
- ***BatchNormalization + Dropout 0.5*** : Lại một lần chống overfit cực mạnh
- ***Dense 26 + softmax*** : Đưa ra xác suất cho đúng 26 lớp A, B, C, ..., Z

3.1.3 Model nhận dạng hình dạng

```
model = Sequential([
    # Block 1
    Conv2D(32, (3, 3), activation="relu", padding="same", input_shape=(IMG_SIZE, IMG_SIZE, 1)),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Block 2
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Block 3
    Conv2D(128, (3, 3), activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Fully Connected
    Flatten(),
    Dense(256, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),

    Dense(128, activation="relu"),
    BatchNormalization(),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation="softmax")
])
```

Hình 15: Cấu trúc model nhận dạng hình dạng cơ bản

(*) **Block 1: Phát hiện các nét cơ bản tạo nên hình dạng**

- ***Conv2D(32)*** : Phát hiện các đoạn thẳng ngang/dọc/chéo, đường cong nhỏ, góc nhọn
- **BatchNormalization** : Giữ activation ổn định
- ***Conv2D(32) thứ hai*** : Ghép các nét thành cấu trúc nhỏ: góc vuông (hình vuông), góc nhọn (tam giác), đoạn cong (tròn), 3–5 đoạn thẳng giao nhau (ngôi sao)…
- **MaxPooling2D** : Giảm từ IMG_SIZE×IMG_SIZE : IMG_SIZE/2, bắt đầu bỏ qua vị trí chính xác
- ***Dropout 0.25*** : Ngăn filter chỉ học một kiểu nét duy nhất

(*) **Block 2: Nhận diện cấu trúc đặc trưng của từng hình dạng**

- ***Conv2D(64)*** : Phát hiện rõ hình dạng chính:

    - 3 cạnh : tam giác
    - 4 góc vuông : vuông/chữ nhật
    - đường cong khép kín : tròn/elip
    - 5–10 đoạn thẳng đối xứng : ngôi sao
    - 2 tam giác ghép : trái tim
- ***Conv2D(64) thứ hai*** : Củng cố và làm rõ hơn các đặc trưng trên
- ***MaxPooling2D*** : Giảm tiếp xuống còn IMG_SIZE/4
- ***Dropout 0.25*** : Giữ tính tổng quát

(*) **Block 3: Tinh chỉnh và phân biệt các hình rất giống nhau** :

- ***Conv2D(128)*** : Tập trung phân biệt các trường hợp dễ nhầm:
    - hình vuông vs chữ nhật (tỷ lệ chiều dài/rộng)
    - tam giác đều vs tam giác cân vs tam giác thường
    - hình tròn vs elip
    - ngôi sao 5 cánh vs 6 cánh
    - mũi tên hướng lên vs hướng xuống
- ***BatchNormalization*** : Ổn định tầng cuối của phần convolution
- ***MaxPooling2D*** : Giảm mạnh còn IMG_SIZE/8 (thường là 7×7 hoặc 8×8 nếu IMG_SIZE=64)
- ***Dropout 0.25*** : Chống overfit

(*) **Fully Connected: Đưa ra quyết định cuối cùng**

- ***Flatten*** : Gom hết thông tin còn lại.
- ***Dense 256*** : Tổng hợp toàn cục: "có 4 góc vuông + tỷ lệ 1:1 : vuông", "có 3 cạnh + góc nhọn : tam giác".
- ***BatchNormalization + Dropout 0.5*** : Rất mạnh vì lớp Dense đầu dễ overfit nhất.
- ***Dense 128*** : Nén thông tin lần cuối.
- ***BatchNormalization + Dropout 0.5*** : Lại một lớp bảo vệ.
- ***Dense(NUM_CLASSES, softmax)*** : Dự đoán chính xác 1 trong số các hình dạng (ví dụ 10 lớp: circle, square, triangle, rectangle, star, heart, arrow_up, arrow_down, cross, hexagon…).

3.2 Quá trình huấn luyện model

3.2.1 Quá trình tiền xử lý

Để tăng tốc độ quá trình cũng như giúp mô hình đạt hiệu quả tốt nhất, quy trình tiền xử lý được nêu trong Chương 2 sẽ được áp dụng cho quá trình xử lý đầu vào ở giai đoạn sản phẩm (tức là đầu vào cho mô hình đã train). Còn ở giai đoạn huấn luyện, do có 3 mô hình khác nhau nên cần tùy chỉnh tiền xử lý khác nhau cho mỗi mô hình:

- ***Mô hình nhận dạng chữ số viết tay***: Sử dụng dữ liệu MNIST – tập dữ liệu đã được tối ưu và làm sạch vô cùng tốt, vì vậy chỉ cần thêm một bước duy nhất là

Data Augmentation để tăng tính tổng quát cho mô hình tránh bị overfiting do dữ liệu quá sạch.

```
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    zoom_range=0.15,
    fill_mode="nearest"
)
datagen.fit(x_train)
```

Hình 16: Cấu hình Data Augmentation cho MNIST

- ***Mô hình nhận dạng chữ hoa viết tay***: Sử dụng dữ liệu tự tạo từ ảnh chung nhiều chữ số viết bằng tay (Chữ đen, nền trắng) nên cần sử dụng một số bước tiền xử lý sau: Crop + Padding => Tạo ảnh vuông + căn giữa => Resize (28,28) => Nghịch đảo thành chữ trắng trên nền đen => Chuẩn hóa đầu vào thành (0,1) và reshape. Trước khi đưa vào mô hình huấn luyện cũng sử dụng Data Augmentation.
- ***Mô hình nhận dạng hình dạng cơ bản***: Sự dùng hàm tự động tạo dữ liệu huấn luyện với cơ chế giả nhiễu giúp dữ liệu đa dạng hơn so với tự tạo dữ liệu viết tay do hình dạng các hình khá đơn giản dễ gây overfiting. (chi tiết ở trong file mã nguồn). Trước khi đưa vào mô hình huấn luyện cũng sử dụng Data Augmentation.

3.2.2 Quá trình huấn luyện

Để tránh xảy ra hiện tượng overfitting và tiết kiệm tài nguyên để huấn luyện, sử dụng một số callback sau trong quá trình huấn luyện:

- ***EarlyStopping***: Dừng sớm khi xảy ra hiện tượng overfitting và trả về trọng số tốt nhất.
- ***ReduceLROnPlateau***: Tự động giảm tốc độ học khi đến điểm tối ưu, tránh hiện tượng nhảy qua lại quanh điểm tối ưu.
- ***ModelCheckpoint***: Tự động lưu backup lại phiên bản model tốt nhất có được trong quá trình huấn luyện.

```
callbacks = [
    EarlyStopping(monitor="val_accuracy", patience=15, restore_best_weights=True),
    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=5, min_lr=1e-7),
    ModelCheckpoint(MODEL_SAVE_PATH, monitor="val_accuracy", save_best_only=True)
]
```

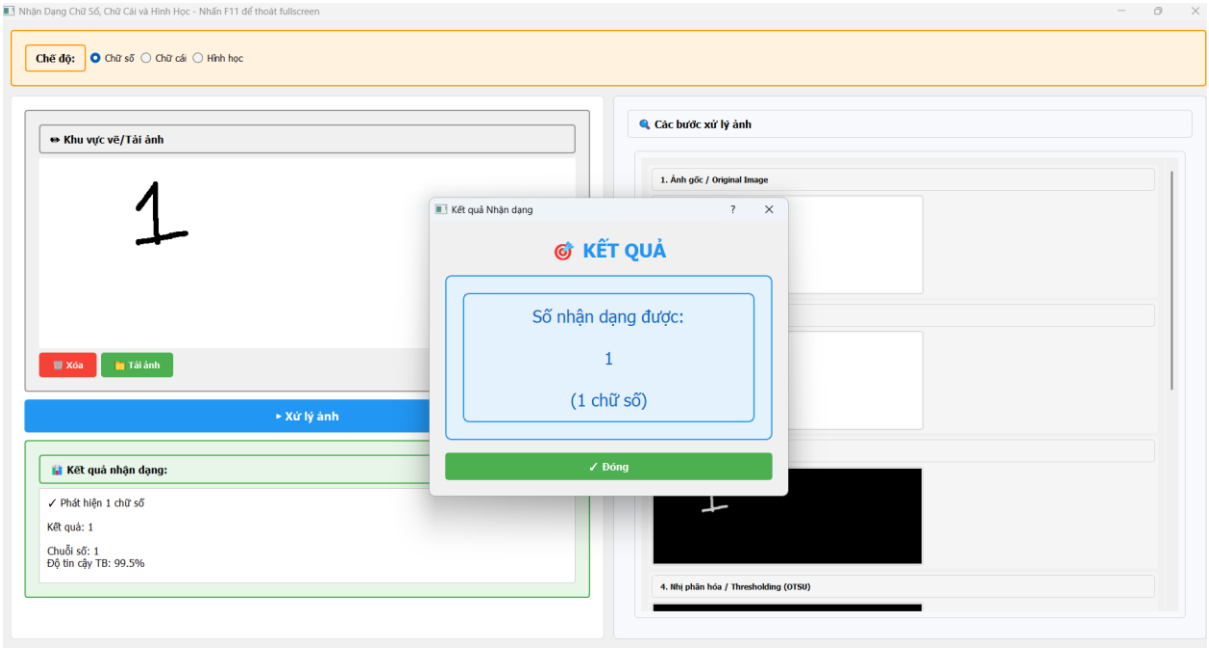
Hình 17: Một số hàm Callbacks

Kết quả quá trình huấn luyện và log theo dõi được thể hiện trong các file kết quả trong mã nguồn dự án.

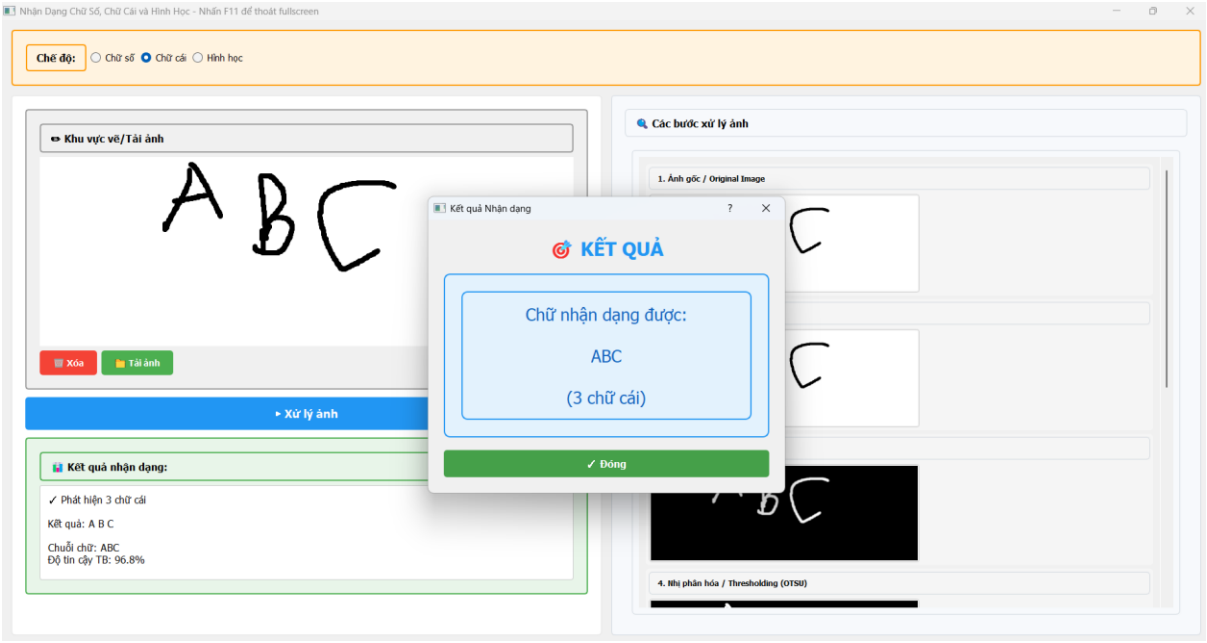
3.3 Xây dựng ứng dụng tool

Sử dụng các hàm tiền xử lý đã được trình bày ở Chương 2 và các model đã được huấn luyện để xây dựng ứng dụng dự đoạn chữ số, chữ Latinh viết hoa và hình dạng cơ bản. Đầu vào là và hình và chữ được viết trực tiếp trên giao diện ứng dụng, có thể viết nhiều chữ và nhiều hình trong một khung ảnh. Đầu ra là các ảnh hiển thị trực quan hóa sự thay đổi của ảnh khi qua các giai đoạn tiền xử lý và cuối cùng là chuỗi các kí tự đã được phát hiện hoặc danh sách các hình.

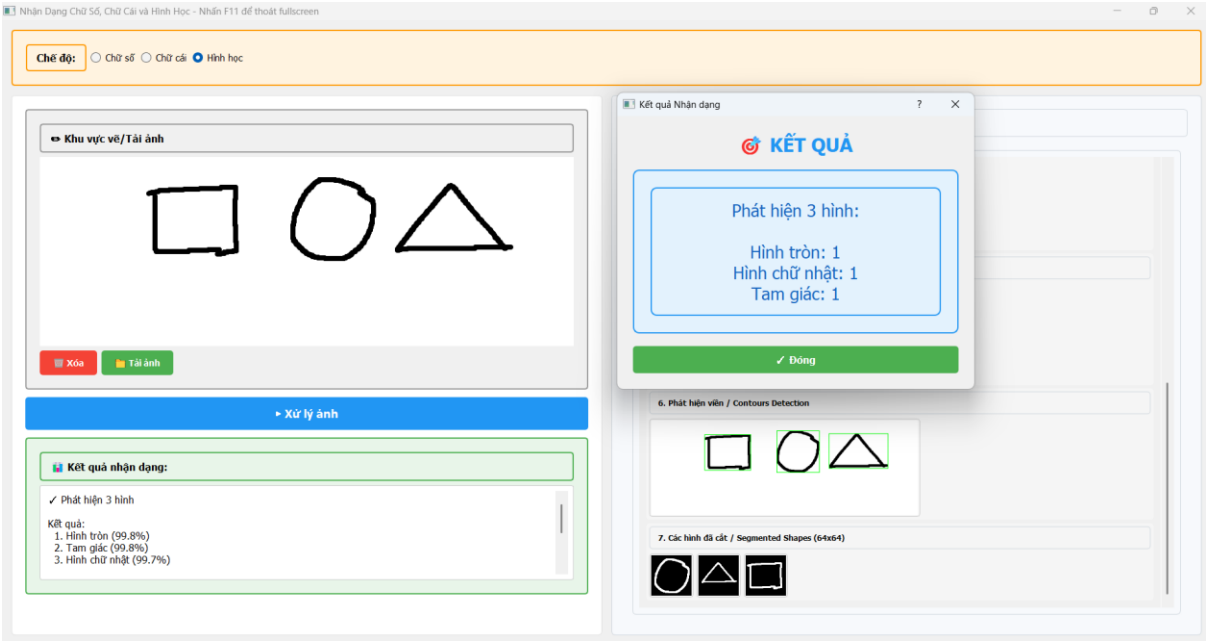
Lựa chọn sự dụng loại hình ứng dụng Desktop, công nghệ sử dụng bao gồm ngôn ngữ Python và thư viện GUI PyQt5 nhằm tăng độ tương thích với mã nguồn đã được xây dựng trước đó cho quá trình tiền xử lý và huấn luyện. Sau đây là một số hình ảnh về ứng dụng sau khi đã xây dựng hoàn thành và chạy thử nghiệm.



Hình 18: Giao diện tool - Phát hiện chữ số viết tay



Hình 19: Giao diện tool - Nhận dạng chữ cái in hoa



Hình 20: Giao diện tool - Nhận dạng hình dạng cơ bản

CHƯƠNG 4: KẾT LUẬN VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN

4.1 Kết luận và nhận xét chung

Trong khuôn khổ bài tập lớn, nhóm đã nghiên cứu và xây dựng một hệ thống đơn giản phục vụ bài toán nhận dạng chữ số viết tay và hình dạng hình học cơ bản dựa trên các kỹ thuật xử lý ảnh và mô hình học sâu, cụ thể là mạng nơ-ron tích chập (CNN). Hệ thống được triển khai theo một quy trình hoàn chỉnh theo hướng tiếp cận hiện đại trong thị giác máy tính: thu thập và chuẩn hóa dữ liệu đầu vào, thực hiện tiền xử lý ảnh (grayscale, adaptive thresholding, loại nhiễu, morphology, contour detection), trích xuất từng đối tượng trong ảnh, đưa dữ liệu vào mô hình CNN để huấn luyện, và cuối cùng xây dựng phần demo để nhận dạng kết quả.

Kết quả thực nghiệm cho thấy mô hình CNN có khả năng nhận dạng chữ số viết tay trên tập dữ liệu MNIST với độ chính xác cao, đồng thời hoạt động tốt trên các hình dạng cơ bản như hình tròn, hình vuông và tam giác thông qua kỹ thuật phân tích đường biên và xấp xỉ đa giác. Các kỹ thuật tiền xử lý ảnh đóng vai trò quan trọng giúp dữ liệu đầu vào được làm sạch, tách bạch và chuẩn hóa, từ đó tăng hiệu quả huấn luyện mô hình. Nhìn chung, hệ thống đã đáp ứng đúng mục tiêu đề ra và hoạt động ổn định trong các trường hợp thử nghiệm.

Tuy vậy, đề tài vẫn còn một số hạn chế nhất định. Mô hình hiện tại được xây dựng với kiến trúc CNN tương đối cơ bản, chưa tối ưu cho các dữ liệu viết tay phức tạp hoặc các hình dạng khó nhận dạng hơn. Hệ thống cũng chưa xử lý tốt các chuỗi ký tự viết liền hoặc các hình vẽ không chuẩn tỷ lệ. Ngoài ra, phần giao diện và chức năng demo chỉ dừng lại ở mức cơ bản, chưa đủ mạnh để trở thành một sản phẩm hoàn chỉnh.

4.2 Phương hướng phát triển

Để mở rộng và nâng cao chất lượng của hệ thống, nhóm đề xuất một số hướng phát triển trong tương lai như sau:

- *Cải thiện kiến trúc mô hình CNN*: Có thể áp dụng các mô hình mạnh hơn như LeNet-5 mở rộng, VGG nhỏ, MobileNet hoặc ResNet để tăng khả năng trích xuất đặc trưng và cải thiện độ chính xác. Ngoài ra, bổ sung các kỹ thuật như Batch Normalization, Dropout hoặc Data Augmentation nâng cao sẽ giúp mô hình tổng quát tốt hơn.
- *Mở rộng tập dữ liệu*: Tập dữ liệu viết tay thực tế rất đa dạng, do đó hệ thống cần được huấn luyện trên dữ liệu phong phú hơn theo nhiều phong cách chữ viết, nhiều độ nhiễu và tình huống khác nhau. Với hình dạng, có thể thêm các hình phức tạp như hình ngũ giác, lục giác, hoặc đa giác bất kỳ.
- *Nâng cấp pipeline tiền xử lý*: Có thể bổ sung thêm các phương pháp nâng cao như Histogram Equalization, Edge Enhancement, hay thuật toán

segmentation chính xác hơn để xử lý các ảnh có chất lượng kém hoặc có nhiều nhiễu.

- *Hoàn thiện giao diện người dùng và tích hợp thực tế*: Phát triển ứng dụng web hoặc mobile cho phép người dùng tải ảnh, vẽ trực tiếp hoặc sử dụng camera để nhận dạng real-time. Tích hợp thêm API để triển khai trong các hệ thống thực tế như chấm thi tự động, số hóa tài liệu hoặc nhận dạng chữ viết trên thiết bị di động.
- *Tối ưu hóa tốc độ xử lý*: Để mô hình chạy nhanh trên thiết bị hạn chế (như laptop yếu hoặc smartphone), có thể nghiên cứu kỹ thuật quantization, pruning hoặc chuyển mô hình sang TensorRT/TFLite.