HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

XỬ LÝ ẢNH

LỚP D22CNPM02

Đề Tài: Nhận dạng chữ viết và hình dạng đơn giản bằng mạng neural

Tên thành viên	Mã sinh viên
Lê Vũ Thành Vinh	B22DCCN904
Phan Văn Thủy	B22DCCN844
Hoàng Việt Anh	B22DCCN017

I. Giới thiệu đề tài

1.1. Lý do chọn đề tài

Trong kỷ nguyên Công nghiệp 4.0, Thị giác máy tính (Computer Vision) đã trở thành một trong những lĩnh vực phát triển mạnh mẽ nhất của Trí tuệ nhân tạo (AI). Nhu cầu tự động hóa việc nhận dạng và phân loại đối tượng từ hình ảnh đang ngày càng cấp thiết trong nhiều lĩnh vực, từ số hóa tài liệu, nhận diện biển số xe, đến điều khiển robot trong dây chuyền sản xuất.

Tuy nhiên, các phương pháp xử lý ảnh truyền thống thường gặp khó khăn khi đối mặt với sự đa dạng của dữ liệu thực tế (như chữ viết tay với nhiều phong cách khác nhau hoặc các hình vẽ bị méo mó, nhiễu). Sự ra đời của Học sâu (Deep Learning), đặc biệt là Mạng Neural Tích chập (Convolutional Neural Network - CNN), đã tạo ra bước đột phá lớn trong việc giải quyết các vấn đề này nhờ khả năng tự động trích xuất đặc trưng và học các mẫu phức tạp.

Đề tài "Xây dựng hệ thống nhận dạng chữ viết tay và hình học sử dụng Mạng Neural Tích chập (CNN)" được lựa chọn nhằm mục đích nghiên cứu sâu về kiến trúc CNN và ứng dụng nó vào hai bài toán nền tảng:

1. **Nhận dạng chữ số viết tay:** Một bài toán kinh điển nhưng có tính ứng dụng cao trong thực tế (ví dụ: xử lý bưu chính, nhập liệu tự động từ biểu mẫu).
2. **Nhận dạng hình học:** Bước đệm quan trọng cho các hệ thống thị giác máy tính trong robotics và nhận dạng mẫu vật thể.

Đề tài không chỉ dừng lại ở lý thuyết mô hình mà còn tập trung vào việc xây dựng một quy trình xử lý ảnh hoàn chỉnh (từ tiền xử lý, sinh dữ liệu nhân tạo đến triển khai ứng dụng thực tế), qua đó cung cấp cái nhìn toàn diện về một dự án Khoa học dữ liệu hiện đại.

1.2. Mục tiêu của hệ thống

Đề tài hướng tới xây dựng một hệ thống hoàn chỉnh bao gồm huấn luyện mô hình, xử lý ảnh, và giao diện tương tác. Cụ thể, hệ thống đặt ra các mục tiêu sau:

- Nhận dạng chữ số viết tay từ 0 đến 9 dựa trên bộ dữ liệu MNIST.

- Nhận dạng các hình học cơ bản như hình tròn, hình vuông, tam giác nhờ dữ liệu sinh tự động.
- Xây dựng quy trình tiền xử lý ảnh nhằm chuẩn hoá dữ liệu đầu vào (lọc nhiễu, resize, chuyển grayscale…).
- Huấn luyện mô hình CNN cho từng bài toán nhận dạng.
- Triển khai giao diện người dùng (UI) bằng Streamlit, cho phép:
    - vẽ trực tiếp chữ số hoặc hình dạng,
    - tải ảnh từ bên ngoài,
    - xem dự đoán và độ tin cậy của mô hình.
- Tạo ra một hệ thống dễ mở rộng, có thể bổ sung thêm nhiều loại hình ảnh hoặc mô hình khác trong tương lai.

1.3. Phạm vi và đối tượng nghiên cứu

1.3.1. Đối tượng nghiên cứu

- **Đối tượng lý thuyết:**
    - Kiến trúc Mạng Neural Tích chập (CNN): Các lớp Conv2D, MaxPooling2D, Dense, Dropout.
    - Các kỹ thuật xử lý ảnh số: Phân ngưỡng Otsu, Lọc nhiễu Gaussian, Phép toán hình thái học (Morphology).
    - Kỹ thuật Tăng cường dữ liệu (Data Augmentation): Xoay, dịch chuyển, thêm nhiễu.
- **Đối tượng dữ liệu:**
    - Bộ dữ liệu MNIST: 70.000 ảnh chữ số viết tay (0-9) kích thước 28 x 28 pixel.
    - Bộ dữ liệu hình học tổng hợp: Được sinh tự động từ thuật toán, bao gồm 8 lớp: Hình tròn, Chữ nhật, Vuông, Tam giác, Ngũ giác, Lục giác, Oval (hình bầu dục), Hình thoi; kích thước 64 x 64 pixel.

1.3.2. Phạm vi nghiên cứu

- **Phạm vi nội dung:** Đề tài tập trung giải quyết bài toán Phân loại ảnh đơn đối tượng (Single-object Classification). Mỗi ảnh đầu vào chỉ chứa một đối tượng chính (một chữ số hoặc một hình vẽ) trên nền tương đối đồng nhất. Không xét đến bài toán phát hiện đối tượng (Object Detection) hay phân đoạn ngữ nghĩa (Semantic Segmentation) trên ảnh phong cảnh phức tạp.
- **Phạm vi công nghệ:**
    - Ngôn ngữ lập trình: Python.
    - Framework Deep Learning: TensorFlow/Keras.
    - Thư viện xử lý ảnh và toán học: NumPy, Pillow.
    - Giao diện: Streamlit.
- **Môi trường triển khai:** Ứng dụng chạy trên nền tảng Web cục bộ (Localhost) hoặc server, sử dụng CPU/GPU để tính toán suy luận.

1.4. Phương pháp thực hiện

Đề tài sử dụng kết hợp phương pháp nghiên cứu lý thuyết và thực nghiệm:

1.4.1. Phương pháp thu thập và chuẩn bị dữ liệu

- **Đối với chữ số:** Sử dụng API của Keras để tải bộ dữ liệu chuẩn MNIST. Thực hiện chuẩn hóa giá trị pixel về khoảng [0, 1] và reshape dữ liệu phù hợp với đầu vào CNN (28 x 28 x 1).
- **Đối với hình học:** Sử dụng phương pháp **Sinh dữ liệu mô phỏng (Synthetic Data Generation)**. Viết các script Python sử dụng thư viện **Pillow (PIL)** để vẽ ngẫu nhiên các hình học cơ bản trên nền đen. Để tạo ra tập dữ liệu đa dạng và mô phỏng các biến thể thực tế, hệ thống áp dụng các kỹ thuật tăng cường dữ liệu bao gồm **phép quay ảnh ngẫu nhiên** (sử dụng tính năng native của PIL) và **thêm nhiễu Gauss** (sử dụng các hàm tạo số ngẫu nhiên của NumPy) vào ảnh sinh ra.

1.4.2. Phương pháp xây dựng mô hình

- Sử dụng mô hình mạng CNN tuần tự (Sequential API).
- Thiết kế kiến trúc mạng gồm nhiều tầng trích xuất đặc trưng (Feature Extraction) với các lớp tích chập và gộp, theo sau là phần phân loại (Classification) với các lớp kết nối đầy đủ.
- Sử dụng kỹ thuật **Dropout** và **Batch Normalization** để chống hiện tượng quá khớp (Overfitting) và tăng tốc độ huấn luyện.

1.4.3. Phương pháp huấn luyện và đánh giá

- **Huấn luyện:** Sử dụng thuật toán lan truyền ngược (Backpropagation) với bộ tối ưu hóa Adam và hàm mất mát Sparse Categorical Crossentropy.
- **Đánh giá:** Chia tập dữ liệu thành tập huấn luyện (Training) và tập kiểm thử (Validation/Test). Đánh giá hiệu năng mô hình dựa trên độ chính xác (Accuracy) và giá trị hàm mất mát (Loss). Sử dụng biểu đồ trực quan để theo dõi quá trình hội tụ.

1.4.4. Phương pháp triển khai ứng dụng

- Tích hợp mô hình đã huấn luyện vào ứng dụng web sử dụng Streamlit.
- Xây dựng module tiền xử lý đầu vào (Input Preprocessing Module) để chuyển đổi ảnh vẽ từ Canvas hoặc ảnh upload của người dùng về cùng định dạng với dữ liệu huấn luyện, đảm bảo độ chính xác khi dự đoán.

II. Cơ sở lý thuyết

2.1. Bài toán nhận dạng chữ số và hình học cơ bản

Bài toán nhận dạng trong dự án này được định nghĩa là một bài toán phân loại đa lớp (Multi-class Classification) trong lĩnh vực thị giác máy tính.

Về mặt hình thức, gọi X là không gian các ma trận ảnh đầu vào (ảnh mức xám) và Y là không gian các nhãn rời rạc.

- Với bài toán chữ số: $Y\_digits = \{0, 1, ..., 9\}$
- Với bài toán hình học: $Y\_shapes = \{Circle, Square, Triangle, ...\}$.

Mục tiêu là tìm kiếm một hàm ánh xạ f: X -> Y sao cho sai số phân loại trên tập dữ liệu kiểm thử là nhỏ nhất.

Thách thức chính của bài toán nằm ở sự biến thiên nội lớp (intra-class variation): cùng một chữ số 5 hoặc hình tam giác có thể có nhiều kích thước, độ nghiêng, nét vẽ và độ nhiễu khác nhau tùy thuộc vào người viết.

2.2. Tổng quan về xử lý ảnh số

Xử lý ảnh số là bước tiền xử lý bắt buộc để chuyển đổi dữ liệu thô thành định dạng chuẩn hóa mà mô hình học máy có thể hiểu được. Các kỹ thuật xử lý ảnh cốt lõi được áp dụng trong file image_processor.py bao gồm:

2.2.1. Biểu diễn và chuyển đổi không gian màu (Grayscale Conversion)

Ảnh kỹ thuật số đầu vào thường ở dạng RGB (3 kênh màu). Tuy nhiên, đối với bài toán nhận dạng cấu trúc hình học và chữ số, thông tin màu sắc không mang lại giá trị phân loại mà còn làm tăng gấp 3 lần khối lượng tính toán.

Hệ thống thực hiện chuyển đổi sang không gian mức xám (Grayscale - 1 kênh) theo công thức chuẩn Luma coding:

$$I_{gray}(x, y) = 0.299 \cdot R(x, y) + 0.587 \cdot G(x, y) + 0.114 \cdot B(x, y)$$

Kỹ thuật này giảm chiều dữ liệu đầu vào từ (H, W, 3) xuống (H, W, 1), giúp tập trung tài nguyên tính toán vào các đặc trưng biên dạng và kết cấu.

2.2.2. Kỹ thuật thay đổi kích thước và Nội suy ảnh (Resizing & Interpolation)

Mạng CNN yêu cầu kích thước đầu vào cố định (Fixed Input Size). Quá trình thay đổi kích thước (Resizing) là bắt buộc để đồng bộ hóa dữ liệu từ nhiều nguồn khác nhau (Canvas vẽ tay, ảnh Upload).

- **Phương pháp thực hiện:** Hệ thống sử dụng phép nội suy diện tích (Inter-area Interpolation) thông qua hàm cv2.resize với tham số cv2.INTER_AREA.
- **Cơ sở toán học:** Khác với phương pháp láng giềng gần nhất (Nearest Neighbor) thường gây ra hiện tượng răng cưa (aliasing) khi thu nhỏ ảnh, phương pháp Inter-area tính toán giá trị pixel mới dựa trên tỷ lệ diện tích đè

lên của các pixel gốc. Điều này giúp ảnh sau khi thu nhỏ (về 28 x 28 hoặc 64 x 64) vẫn giữ được độ mượt mà và bảo toàn thông tin cấu trúc quan trọng.

2.2.3. Chuẩn hóa dữ liệu (Data Normalization)

Đây là bước tiền xử lý quan trọng nhất ảnh hưởng trực tiếp đến khả năng hội tụ của thuật toán Gradient Descent. Thay vì đưa trực tiếp giá trị thô [0, 255] vào mạng nơ-ron, hệ thống thực hiện kỹ thuật Min-Max Scaling để nén miền giá trị này về khoảng [0, 1]

- **Vấn đề:** Giá trị pixel gốc nằm trong khoảng số nguyên [0, 255]. Biên độ lớn này tạo ra bề mặt lỗi (Loss Surface) không cân đối, khiến thuật toán tối ưu hóa dễ bị sa lầy hoặc dao động mạnh.
- **Giải pháp**: Hệ thống thực hiện phép biến đổi tuyến tính để đưa dữ liệu về khoảng [0, 1] bằng công thức:

$$X_{norm} = \frac{X_{original}}{255.0}$$

- **Tác dụng:**
    1. **Tăng tốc độ hội tụ:** Giúp Gradient hướng thẳng về cực trị toàn cục nhanh hơn.
    2. **Ổn định số học:** Ngăn chặn hiện tượng bão hòa hàm kích hoạt ở các lớp nơ-ron sâu.

2.2.4. Pipeline xử lý chuyên biệt

Hệ thống thiết kế hai luồng xử lý riêng biệt phù hợp với đặc thù của từng mô hình:

- **Pipeline cho MNIST (preprocess_for_mnist):** Chuyển xám → Resize về 28×28 → Chuẩn hóa → Reshape thành vector (28,28,1)
- **Pipeline cho Hình học (preprocess_for_shapes):** Chuyển xám → Resize về 64×64 → Chuẩn hóa → Reshape thành vector (64,64,1). Kích thước 64×64 lớn được chọn để bảo toàn các chi tiết góc cạnh sắc nét của các hình đa giác.

2.3. Mạng nơ-ron tích chập (CNN)

CNN là kiến trúc mạng nơ-ron sâu tiên tiến nhất hiện nay cho các bài toán nhận dạng ảnh, được thiết kế để xử lý dữ liệu có cấu trúc lưới. Nó khắc phục được nhược điểm của mạng kết nối đầy đủ, không làm mất cấu trúc không gian của ảnh.

2.3.1. Lớp Tích chập (Convolutional Layer)

Đây là nơi trích xuất đặc trưng. Thay vì kết nối đầy đủ, CNN sử dụng các bộ lọc (kernel) K trượt qua ảnh đầu vào I. Giá trị tại vị trí (i, j) của bản đồ đặc trưng (feature map) được tính bằng:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Các bộ lọc này học cách phát hiện các đặc trưng từ đơn giản (cạnh, góc) đến phức tạp (hình dạng) thông qua quá trình huấn luyện.

2.3.2. Hàm kích hoạt ReLU (Rectified Linear Unit)

Để đưa tính phi tuyến vào mô hình, hàm ReLU được sử dụng sau mỗi lớp tích chập:

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

ReLU giúp khắc phục vấn đề biến mất đạo hàm (vanishing gradient) và tăng tốc độ hội tụ nhờ tính toán đơn giản và tạo ra tính thưa (sparsity) trong mạng.

2.3.3. Lớp Gộp (Pooling Layer)

Dự án sử dụng lớp Max Pooling để giảm kích thước không gian của feature map, giúp giảm số lượng tham số và tăng tính bất biến đối với phép tịnh tiến (translation invariance).

$$y = \max(window)$$

Ví dụ: Với cửa sổ 2 x 2, ảnh đầu vào giảm kích thước đi 4 lần, chỉ giữ lại thông tin đặc trưng nổi bật nhất.

2.3.4. Hàm Softmax và Loss Function

Giai đoạn cuối cùng của mạng CNN là đưa ra quyết định phân loại dựa trên các đặc trưng đã được trích xuất. Hai thành phần toán học đóng vai trò quyết định ở

bước này là hàm kích hoạt Softmax và hàm mất mát Sparse Categorical Crossentropy.

a) Hàm kích hoạt Softmax (Softmax Activation Function)

Trong các lớp ẩn, chúng ta sử dụng ReLU để giữ tính phi tuyến. Tuy nhiên, tại lớp đầu ra (Output Layer), các giá trị thô (logits) từ mạng nơ-ron có thể là bất kỳ số thực nào, điều này gây khó khăn cho việc diễn giải kết quả.

Hàm Softmax được sử dụng để giải quyết vấn đề này bằng cách chuẩn hóa vector đầu ra thành một **phân phối xác suất**.

- Công thức toán học:

Giả sử z là vector đầu vào của lớp Softmax (logits) có kích thước C (số lượng lớp), xác suất để ảnh đầu vào thuộc về lớp thứ i được tính như sau:

$$P(y = i \mid x) = \sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

Trong đó:

  - z_i: Giá trị đầu ra thô của nơ-ron thứ i.
  - C: Tổng số lớp cần phân loại (10 lớp cho MNIST, 8 lớp cho Shape).
  - e: Hằng số Euler.

- **Đặc điểm:**
  - Mọi giá trị xác suất đầu ra đều nằm trong khoảng [0, 1].
  - Tổng xác suất của tất cả các lớp luôn bằng 1.

b) Hàm mất mát Sparse Categorical Crossentropy

Để huấn luyện mạng CNN, ta cần một thước đo để đánh giá "độ sai lệch" giữa dự đoán của mô hình và nhãn thực tế. Dự án sử dụng hàm **Sparse Categorical Crossentropy**.

Cơ chế hoạt động:

Hàm này tính toán độ sai biệt giữa phân phối xác suất dự đoán và nhãn thực tế với công thức:

$$L(y, \hat{y}) = -\log(\hat{y}_{target})$$

Trong đó $\hat{y}_{target}$ là xác suất mà mô hình dự đoán cho lớp đúng (target class).

- **Ý nghĩa:**
  - Nếu mô hình dự đoán đúng với độ tin cậy cao (ví dụ: $\hat{y}_{target} \approx 1$), thì Loss ≈ 0.
  - Nếu mô hình dự đoán sai (ví dụ: $\hat{y}_{target} \approx 0$), thì Loss $\to \infty$.

Điều này đồng nghĩa với việc thuật toán sẽ phạt nặng các trường hợp mô hình tự tin nhưng dự đoán sai. Ví dụ: mô hình khẳng định 99% ảnh là hình tròn, nhưng thực tế nó là hình vuông, thì thuật toán sẽ tính ra một mức Loss cực kỳ lớn.

2.4. Tập dữ liệu MNIST cho chữ số viết tay

MNIST (Modified National Institute of Standards and Technology database) là bộ dữ liệu chuẩn về chữ số viết tay trong lĩnh vực học sâu.

- **Cấu trúc:** Gồm 60.000 ảnh huấn luyện và 10.000 ảnh kiểm thử.
- **Đặc điểm:** Mỗi ảnh là chữ số viết tay từ 0 đến 9, đã được căn giữa (centered) trong khung hình kích thước cố định 28 x 28 pixel. Ảnh ở dạng grayscale.
- **Chuẩn hóa:** Trước khi đưa vào mạng CNN, dữ liệu pixel [0, 255] được chuẩn hóa về [0, 1] bằng phép chia cho [255,0]. Điều này giúp gradient descent hội tụ nhanh hơn và tránh bão hòa hàm kích hoạt.

2.5. Tập dữ liệu hình học tổng hợp (Synthetic Geometric Dataset)

Do sự thiếu hụt các bộ dữ liệu chuẩn hóa cho bài toán nhận dạng hình học cơ bản (tương tự như MNIST cho chữ số), đồ án áp dụng phương pháp Sinh dữ liệu tổng hợp dựa trên thủ tục (Procedural Synthetic Data Generation). Phương pháp này cho phép chủ động tạo ra một lượng lớn dữ liệu huấn luyện có nhãn chính xác, đồng thời kiểm soát được mức độ phức tạp và đa dạng của dữ liệu.

2.5.1. Quy trình sinh dữ liệu cơ sở

Quá trình tạo ảnh cơ sở được thực hiện dựa trên thư viện đồ họa Pillow (PIL) để vẽ các nguyên hàm hình học và thư viện NumPy để xử lý mảng dữ liệu ảnh. Class ShapeGenerator chịu trách nhiệm định nghĩa 8 lớp hình học mục tiêu: Circle (Tròn), Rectangle (Chữ nhật), Square (Vuông), Triangle (Tam giác), Pentagon (Ngũ giác), Hexagon (Lục giác), Oval (Bầu dục), và Diamond (Hình thoi).

Mỗi mẫu dữ liệu được tạo ra theo quy trình chuẩn sau:

1. Khởi tạo không gian vẽ (Canvas Initialization):

Hệ thống tạo một ảnh mới với chế độ màu 'L' (Luminance - ảnh mức xám 8-bit), kích thước mặc định là 64 x 64 pixel. Toàn bộ nền ảnh được tô màu đen tuyệt đối (giá trị pixel bằng 0).

2. Vẽ nguyên hàm hình học (Primitive Drawing):

Sử dụng module ImageDraw của thư viện PIL để vẽ hình học lên canvas. Đối tượng hình học được vẽ với màu trắng tuyệt đối (giá trị pixel bằng 255) để tạo độ tương phản tối đa với nền.

- Các hình cong (Circle, Oval) được vẽ bằng hàm draw.ellipse dựa trên tọa độ khung bao (bounding box).
- Các hình chữ nhật (Rectangle, Square) được vẽ bằng hàm draw.rectangle.
- Các hình đa giác đều (Triangle, Pentagon, Hexagon, Diamond) được vẽ bằng hàm draw.polygon, với tọa độ các đỉnh được tính toán dựa trên hình học lượng giác để đảm bảo tính cân xứng.

Để đảm bảo sự đa dạng, các tham số như tọa độ tâm, bán kính, chiều dài cạnh được chọn ngẫu nhiên trong một phạm vi hợp lý tùy thuộc vào kích thước ảnh, đảm bảo hình vẽ luôn nằm gọn trong khung hình.

2.5.2. Kỹ thuật Tăng cường dữ liệu (Data Augmentation)

Nếu chỉ sử dụng các hình vẽ cơ sở lý tưởng, mô hình CNN sẽ dễ bị quá khớp (overfitting) và hoạt động kém trên dữ liệu thực tế có nhiều biến thể. Để giải quyết vấn đề này, hai kỹ thuật tăng cường dữ liệu được áp dụng ngẫu nhiên ngay trong quá trình sinh dữ liệu (on-the-fly augmentation):

a) Phép quay ảnh ngẫu nhiên (Random Rotation)

- **Mục đích:** Giúp mô hình học được tính bất biến với góc xoay (rotational invariance), nhận dạng được hình học dù nó được vẽ nghiêng hay thẳng.
- **Thực hiện:** Dữ liệu ảnh dạng mảng NumPy được chuyển đổi tạm thời sang đối tượng PIL Image. Sau đó, phương thức .rotate() được gọi với một góc ngẫu nhiên $\theta \in [0,360\circ]$ .Hệ thống sử dụng phương pháp nội suy Bicubic (Image.BICUBIC) để đảm bảo chất lượng ảnh và độ mượt của các cạnh sau khi xoay không bị suy giảm đáng kể.

b) Thêm nhiễu Gaussian (Gaussian Noise Injection)

- **Mục đích:** Mô phỏng các điều kiện nhiễu thực tế (như nhiễu cảm biến camera, điều kiện ánh sáng kém hoặc chất lượng ảnh thấp), buộc mô hình phải học các đặc trưng cấu trúc cốt lõi thay vì phụ thuộc vào các giá trị pixel sạch tuyệt đối.
- **Thực hiện:** Sử dụng NumPy để tạo ra một ma trận nhiễu có cùng kích thước với ảnh, với các giá trị tuân theo phân phối chuẩn (Gaussian distribution): $I_{noisy} = I_{original} + \mathcal{N}(\mu, \sigma^2)$
  Ma trận này được nhân với mức độ nhiễu (ví dụ: 0.05 * 255) và cộng trực tiếp vào ảnh gốc. Cuối cùng, các giá trị pixel được cắt (clip) để đảm bảo nằm trong phạm vi hợp lệ [0, 255].

2.5.3. Chuẩn hóa và Đóng gói dữ liệu

Bước cuối cùng trong quy trình sinh dữ liệu (generate_dataset) là chuẩn hóa để đưa vào mô hình CNN:

1. **Chuẩn hóa giá trị:** Ảnh được chuyển sang kiểu dữ liệu số thực 32-bit (float32) và chia cho 255.0 để đưa miền giá trị pixel về khoảng [0, 1].
2. **Định hình lại (Reshaping):** Dữ liệu được định hình lại thành tensor 3 chiều có kích thước (64, 64, 1) để phù hợp với lớp Input của mạng CNN.
3. **Trộn dữ liệu (Shuffling):** Toàn bộ tập dữ liệu sinh ra được xáo trộn ngẫu nhiên để đảm bảo tính khách quan trong quá trình huấn luyện và kiểm thử.

III: Thiết kế hệ thống

3.1. Kiến trúc tổng thể hệ thống

Hệ thống được thiết kế theo mô hình **Kiến trúc Phân tầng**. Mô hình này chia tách hệ thống thành các tầng chức năng riêng biệt, đảm bảo tính module hóa, dễ dàng bảo trì và mở rộng. Các tầng giao tiếp với nhau thông qua các giao diện và luồng dữ liệu được định nghĩa rõ ràng.

Hệ thống bao gồm 3 tầng chính:

3.1.1. Tầng Giao diện (Presentation Layer)

Đây là tầng tương tác trực tiếp với người dùng, được xây dựng dựa trên framework Streamlit. Tầng này có chức năng:

- **Thu thập dữ liệu đầu vào:** Cung cấp hai phương thức nhập liệu chính là vẽ trực tiếp lên màn hình (thông qua component Canvas) và tải lên tệp tin hình ảnh (File Uploader).
- **Hiển thị kết quả:** hiển thị kết quả nhãn dự đoán, độ tin cậy và biểu đồ phân phối xác suất.
- **Điều hướng:** Quản lý các trang chức năng như Trang chủ, Nhận dạng chữ số, Nhận dạng hình học và Huấn luyện mô hình.

3.1.2. Tầng Xử lý nghiệp vụ

Tầng này đóng vai trò trung gian, chịu trách nhiệm xử lý dữ liệu thô từ tầng giao diện trước khi đưa vào mô hình, cũng như quản lý logic huấn luyện.

- **Module image_processor.py:** Thực hiện các thuật toán xử lý ảnh số như chuyển đổi không gian màu, thay đổi kích thước (resize), và chuẩn hóa dữ liệu.
- **Module shape_generator.py:** Đảm nhận logic sinh dữ liệu tổng hợp (Synthetic Data) và tăng cường dữ liệu (Data Augmentation) phục vụ cho việc huấn luyện.
- **Controller Logic:** File **app.py** đóng vai trò như một Controller, điều phối việc gọi model nào dựa trên thao tác của người dùng.

3.1.3. Tầng Mô hình và Dữ liệu (Model & Data Layer)

Đây là tầng lõi chứa các thuật toán Trí tuệ nhân tạo và dữ liệu lưu trữ.

- **CNN Models:** Bao gồm hai kiến trúc mạng nơ-ron tích chập riêng biệt:
    - MNISTModel: Chuyên biệt cho nhận dạng chữ số có kích thước ảnh 28 x 28.
    - ShapeModel: Chuyên biệt cho nhận dạng hình học có kích thước ảnh 64 x 64.
- **Lưu trữ:** Các mô hình sau khi huấn luyện được lưu dưới dạng file .keras trong thư mục models/ để tái sử dụng.

3.2. Luồng dữ liệu và quy trình xử lý ảnh đầu vào

Luồng dữ liệu (Data Flow) trong hệ thống được thiết kế khép kín từ lúc người dùng thao tác đến khi nhận được kết quả. Quy trình này khác nhau tùy thuộc vào nguồn dữ liệu đầu vào.

3.2.1. Luồng xử lý từ Canvas (Vẽ tay)

Khi người dùng vẽ trên giao diện web:

1. **Thu nhận:** Component st_canvas trả về dữ liệu ảnh dưới dạng mảng NumPy 4 kênh màu (RGBA).
2. **Trích xuất:** Hệ thống chỉ lấy kênh dữ liệu cần thiết (thường là kênh Alpha hoặc một kênh màu) để tạo thành ảnh đơn sắc, loại bỏ thông tin màu sắc không cần thiết.
3. **Tiền xử lý:** Dữ liệu được đưa qua ImageProcessor để resize và chuẩn hóa.
4. **Dự đoán:** Mảng dữ liệu đã xử lý được đưa vào hàm predict() của model tương ứng.

3.2.2. Luồng xử lý từ Upload (Tải file)

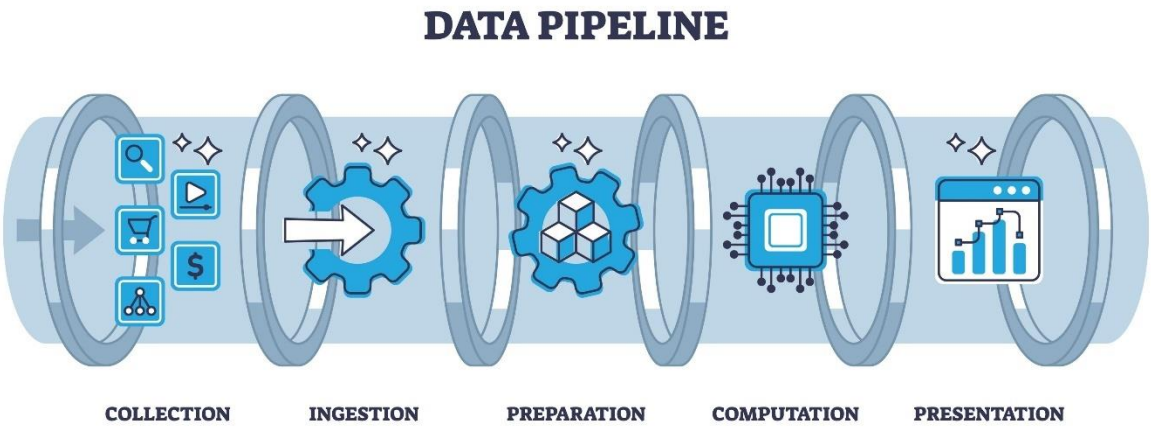
Khi người dùng tải ảnh lên:

1. **Thu nhận:** File ảnh (JPG, PNG) được đọc vào bộ nhớ thông qua thư viện PIL (Python Imaging Library).
2. **Chuyển đổi:** Đối tượng ảnh PIL được chuyển đổi thành mảng số đa chiều NumPy (np.array). cho phép thực hiện các phép toán tính toán trong các bước xử lý tiếp theo
3. **Tiền xử lý:** Áp dụng chuỗi xử lý: Grayscale → Resize → Normalize.
4. **Dự đoán:** Mảng dữ liệu đã xử lý được đưa vào hàm predict() của model tương ứng.

3.2.3. Sơ đồ quy trình xử lý (Pipeline)

Tổng quát hóa quy trình xử lý cho một ảnh đầu vào I:

$$I_{raw} \xrightarrow{\text{Grayscale}} I_{gray} \xrightarrow{\text{Resize}} I_{resized} \xrightarrow{\text{Normalize}} I_{norm} \xrightarrow{\text{Reshape}} I_{input} \xrightarrow{\text{CNN}} \text{Prediction}$$



3.3 Thiết kế mô-đun tiền xử lý ảnh (ImageProcessor)

Mô-đun **ImageProcessor** (*image_processor.py*) đóng vai trò quan trọng trong hệ thống khi đảm nhiệm chức năng chuẩn hóa và tiền xử lý dữ liệu ảnh trước khi đưa vào các mô hình CNN. Việc tiền xử lý đúng chuẩn giúp mô hình học sâu hoạt động ổn định, giảm nhiễu và đảm bảo tính đồng nhất giữa dữ liệu huấn luyện và dữ liệu dự đoán.

3.3.1 Mục tiêu thiết kế

- Biến đổi ảnh về cùng kích thước (28×28 cho MNIST, 64×64 cho Shapes).
- Chuyển ảnh sang thang xám để giảm độ phức tạp.
- Chuẩn hóa giá trị pixel về khoảng [0,1].
- Tạo đầu vào phù hợp với kiến trúc CNN (dạng tensor H×W×1).

3.3.2 Các chức năng chính

a) Chuyển ảnh sang grayscale:
Giúp giảm nhiễu màu và phù hợp với dữ liệu huấn luyện.

```
@staticmethod
def convert_to_grayscale(image):
    if len(image.shape) == 3:
        return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    return image
```

b) Resize ảnh:
Đảm bảo ảnh đầu vào có kích thước chuẩn.

```
@staticmethod
def resize_image(image, size=(28, 28)):
    if isinstance(image, Image.Image):
        image = np.array(image)
    return cv2.resize(image, size, interpolation=cv2.INTER_AREA)
```

c) Chuẩn hóa pixel về giá trị [0,1]:

```
@staticmethod
def normalize_image(image):
    image = image.astype(np.float32)
    return image / 255.0
```

d) Hàm tiền xử lý cho MNIST: trả về ảnh 28×28×1

```
@staticmethod
def preprocess_for_mnist(image):
    if isinstance(image, Image.Image):
        image = np.array(image)

    gray = ImageProcessor.convert_to_grayscale(image)

    resized = ImageProcessor.resize_image(gray, (28, 28))

    normalized = ImageProcessor.normalize_image(resized)

    reshaped = normalized.reshape(28, 28, 1)

    return reshaped
```

e) Hàm tiền xử lý cho Shapes:

```python
@staticmethod
def preprocess_for_shapes(image, target_size=(64, 64)):
    if isinstance(image, Image.Image):
        image = np.array(image)

    gray = ImageProcessor.convert_to_grayscale(image)

    resized = ImageProcessor.resize_image(gray, target_size)

    normalized = ImageProcessor.normalize_image(resized)

    reshaped = normalized.reshape(target_size[0], target_size[1], 1)

    return reshaped
```

3.4 Thiết kế mô-đun nhận dạng chữ số (MNISTModel)

Mô-đun MNISTModel trong file **mnist_model.py** xây dựng mô hình CNN phân loại chữ số 0–9 dựa trên tập dữ liệu MNIST.

3.4.1 Mục tiêu thiết kế

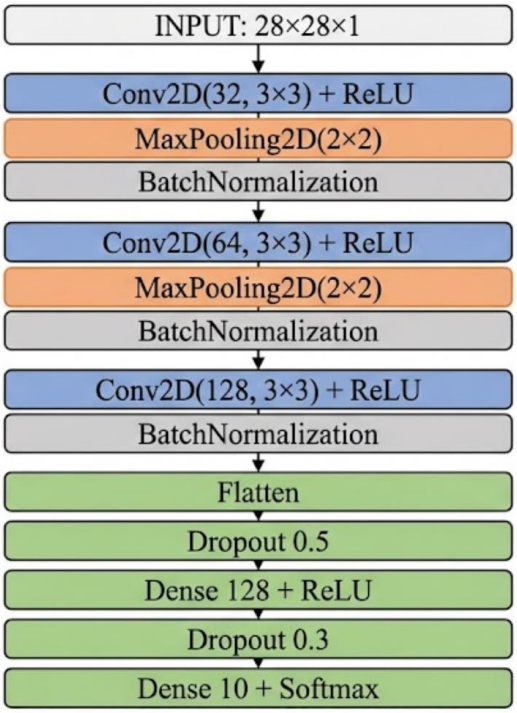
- Xây dựng mô hình CNN độ chính xác > 98%.
- Tối ưu cấu trúc để xử lý ảnh nhỏ 28×28.
- Tích hợp training, saving, và load model.

3.4.2 Kiến trúc CNN

Mô hình sử dụng API Sequential của Keras, bao gồm các lớp chính sau:

- **Input Layer**: Chấp nhận đầu vào với kích thước (28, 28, 1).
- **Khối Trích xuất đặc trưng 1**:
    - Conv2D (32 filters, kernel 3x3, activation 'relu'): Trích xuất các đặc trưng cơ bản (cạnh, góc).
    - MaxPooling2D (pool size 2x2): Giảm kích thước không gian, tăng tính bất biến với phép tịnh tiến.
    - BatchNormalization: Ổn định quá trình huấn luyện.
- **Khối Trích xuất đặc trưng 2**:

    - Conv2D (64 filters, kernel 3x3, activation 'relu'): Trích xuất đặc trưng phức tạp hơn.
    - MaxPooling2D (2x2).
    - BatchNormalization.
- **Khối Trích xuất đặc trưng 3**:
    - Conv2D (128 filters, kernel 3x3, activation 'relu').
    - BatchNormalization.
- **Khối Phân loại**:
    - Flatten: Duỗi thẳng tensor đặc trưng thành vector 1 chiều.
    - Dropout (0.5): Ngăn chặn overfitting.
    - Dense (128 units, activation 'relu'): Lớp kết nối đầy đủ.
    - Dropout (0.3): Ngăn chặn overfitting.
    - Dense (10 units, activation 'softmax'): Lớp đầu ra cho 10 lớp chữ số (0-9), trả về phân phối xác suất.



3.4.3. Quá trình huấn luyện:

Bao gồm các bước:

```python
def train(self, epochs=10, batch_size=128):
    (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    x_train = x_train.reshape(-1, 28, 28, 1)
    x_test = x_test.reshape(-1, 28, 28, 1)

    if self.model is None:
        self.build_model()

    early_stopping = keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        patience=3,
        restore_best_weights=True
    )

    reduce_lr = keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=2,
        min_lr=1e-7
    )

    self.history = self.model.fit(
        x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(x_test, y_test),
        callbacks=[early_stopping, reduce_lr],
        verbose=1
    )

    os.makedirs('models', exist_ok=True)
    self.model.save(self.model_path)

    return self.history
```

1. Tải và chuẩn hóa dữ liệu MNIST từ keras.datasets về khoảng [0, 1] và reshape về (28, 28, 1).
2. Sử dụng các Callbacks để tối ưu hóa:
    a. EarlyStopping: Dừng huấn luyện sớm nếu val_accuracy không cải thiện sau 3 epochs, tránh overfitting.
    b. ReduceLROnPlateau: Giảm tốc độ học (factor 0.5) nếu val_loss không giảm sau 2 epochs, giúp mô hình hội tụ tốt hơn ở các bước cuối.
3. Lưu mô hình đã huấn luyện vào đường dẫn models/mnist_cnn.keras

3.5 Thiết kế mô-đun sinh dữ liệu hình học (ShapeGenerator)

Do thiếu hụt các bộ dữ liệu hình học vẽ tay chuẩn, class ShapeGenerator được thiết kế để tự động sinh dữ liệu huấn luyện tổng hợp (synthetic data).

3.5.1. Các lớp hình học và Phương pháp sinh

Mô-đun định nghĩa 8 lớp hình học cơ bản thông qua dictionary SHAPE_CLASSES: Circle, Rectangle, Square, Triangle, Pentagon, Hexagon, Oval, và Diamond.

```python
SHAPE_CLASSES = {
    0: 'Circle',
    1: 'Rectangle',
    2: 'Square',
    3: 'Triangle',
    4: 'Pentagon',
    5: 'Hexagon',
    6: 'Oval',
    7: 'Diamond'
}
```

Quy trình sinh hình cơ sở được thực hiện dựa trên thư viện đồ họa **Pillow (PIL)** kết hợp với **NumPy**:

1. **Khởi tạo Canvas:** Mỗi hình ảnh được bắt đầu bằng việc tạo một nền đen tuyệt đối (giá trị pixel 0) với mode ảnh mức xám ('L') thông qua hàm phụ trợ _create_canvas.

```python
@staticmethod
def _create_canvas(size):
    """Hàm phụ trợ tạo nền đen"""
    # Tạo ảnh mode 'L' (Luminance - Grayscale 8-bit), màu đen (0)
    return Image.new('L', (size, size), 0)
```

2. **Vẽ hình học:** Sử dụng module ImageDraw của PIL để vẽ các khối đặc màu trắng (giá trị 255) lên nền đen. Các hàm vẽ được thiết kế để tạo ra hình với kích thước và vị trí ngẫu nhiên
   Ví dụ vẽ hình tròn (Circle): *Sử dụng draw.ellipse dựa trên khung bao (bounding box).*

```python
@staticmethod
def create_circle(size=64):
    image = ShapeGenerator._create_canvas(size)
    draw = ImageDraw.Draw(image)

    center = size // 2
    radius = np.random.randint(size // 4, size // 2 - 5)

    # PIL vẽ ellipse/circle qua bounding box (x0, y0, x1, y1)
    x0 = center - radius
    y0 = center - radius
    x1 = center + radius
    y1 = center + radius

    draw.ellipse([x0, y0, x1, y1], fill=255)
    return ShapeGenerator._to_numpy(image)
```

Ví dụ vẽ hình tam giác (Triangle): *Sử dụng draw.polygon với tọa độ 3 đỉnh được tính toán ngẫu nhiên.*

```python
@staticmethod
def create_triangle(size=64):
    image = ShapeGenerator._create_canvas(size)
    draw = ImageDraw.Draw(image)

    center = size // 2
    height = np.random.randint(size // 2, size - 10)
    base = np.random.randint(size // 2, size - 10)

    # Các đỉnh (points)
    p1 = (center, center - height // 2)
    p2 = (center - base // 2, center + height // 2)
    p3 = (center + base // 2, center + height // 2)

    draw.polygon([p1, p2, p3], fill=255)
    return ShapeGenerator._to_numpy(image)
```

3. **Chuyển đổi định dạng:** Ảnh sau khi vẽ bằng PIL được chuyển đổi sang mảng NumPy (np.uint8) để phục vụ các bước xử lý tiếp theo:

```python
@staticmethod
def _to_numpy(pil_image):
    """Chuyển từ PIL Image sang Numpy array"""
    return np.array(pil_image, dtype=np.uint8)
```

3.5.2. Kỹ thuật Tăng cường dữ liệu (Data Augmentation)

Để tăng tính đa dạng và khả năng tổng quát hóa cho mô hình, hai kỹ thuật tăng cường dữ liệu được áp dụng ngẫu nhiên:

- **Xoay ảnh (rotate_image):** Sử dụng phương thức .rotate() của đối tượng ảnh PIL với góc ngẫu nhiên và nội suy Bicubic để đảm bảo chất lượng ảnh:

```python
@staticmethod
def rotate_image(image_array, angle=None):
    if angle is None:
        angle = np.random.randint(0, 360)

    # Chuyển array sang PIL Image để xoay
    pil_image = Image.fromarray(image_array)
    # resample=Image.BICUBIC giúp ảnh mượt hơn khi xoay
    rotated_pil = pil_image.rotate(angle, resample=Image.BICUBIC, expand=False)

    return np.array(rotated_pil)
```

- **Thêm nhiễu (add_noise):** Sử dụng NumPy để cộng nhiễu Gauss vào ảnh gốc, sau đó dùng np.clip để đảm bảo giá trị pixel hợp lệ:

```python
@staticmethod
def add_noise(image, noise_level=0.1):
    # Noise dùng numpy nên không cần đổi
    noise = np.random.randn(*image.shape) * noise_level * 255
    noisy = image.astype(np.float32) + noise
    noisy = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy
```

Phương thức chủ đạo *generate_dataset* sẽ điều phối quy trình: lặp qua các loại hình, áp dụng ngẫu nhiên rotate_image và add_noise, chuẩn hóa dữ liệu về [0, 1]

và định hình lại (reshape) để tạo ra tập dữ liệu huấn luyện cuối cùng.

```python
@staticmethod
def generate_dataset(samples_per_class=1000, size=64, augment=True):
    shape_functions = [
        ShapeGenerator.create_circle,
        ShapeGenerator.create_rectangle,
        ShapeGenerator.create_square,
        ShapeGenerator.create_triangle,
        ShapeGenerator.create_pentagon,
        ShapeGenerator.create_hexagon,
        ShapeGenerator.create_oval,
        ShapeGenerator.create_diamond
    ]

    X = []
    y = []

    for class_idx, shape_func in enumerate(shape_functions):
        for _ in range(samples_per_class):
            image = shape_func(size)

            if augment and np.random.random() > 0.5:
                image = ShapeGenerator.rotate_image(image)

            if augment and np.random.random() > 0.7:
                image = ShapeGenerator.add_noise(image, noise_level=0.05)

            # Chuẩn hóa về khoảng 0-1
            image = image.astype(np.float32) / 255.0
            image = image.reshape(size, size, 1)

            X.append(image)
            y.append(class_idx)

    X = np.array(X)
    y = np.array(y)

    indices = np.random.permutation(len(X))
    X = X[indices]
    y = y[indices]

    return X, y
```

3.6. Thiết kế mô-đun nhận dạng hình học (ShapeModel)

Module ShapeModel (*shape_model.py*) tương tự như MNISTModel nhưng được thiết kế để giải quyết bài toán nhận dạng hình học phức tạp hơn với kích thước đầu vào lớn hơn (64 x 64).

3.6.1 Mục tiêu thiết kế

- Nhận dạng chính xác 8 loại hình.
- Mô hình CNN phức tạp hơn MNIST để phù hợp hình 64×64.
- Chống overfitting nhờ nhiều lớp Dropout & BatchNormalization.

3.6.2. Kiến trúc mạng CNN

Kiến trúc mạng được thiết kế dựa trên triết lý của mạng **VGG (Visual Geometry Group)**.

Điểm đặc trưng của kiến trúc VGG là thay vì sử dụng các bộ lọc (kernel) kích thước lớn (5x5 hay 7x7) ở các tầng đầu, nó sử dụng chiến lược xếp chồng nhiều lớp bộ lọc kích thước nhỏ 3x3 liên tiếp.

Trong mô hình này, việc sử dụng hai lớp tích chập 3x3 liên tiếp mang lại hiệu quả vượt trội so với việc dùng một lớp 5x5 lớn:

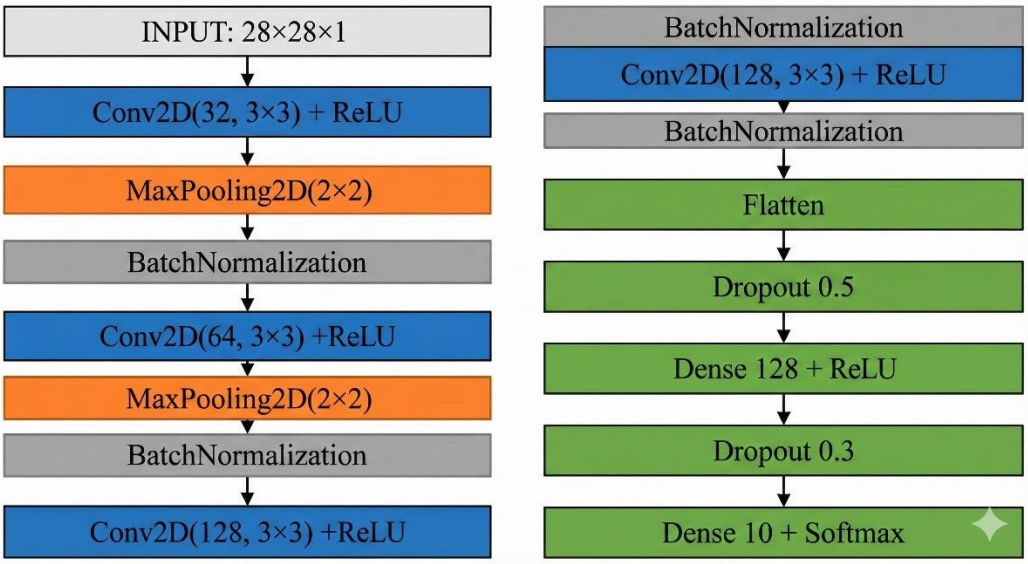
1. **Khả năng học phức tạp hơn:** Việc có nhiều lớp chồng lên nhau (đi kèm với các hàm kích hoạt phi tuyến) giúp mạng có khả năng học được các đặc trưng hình học phức tạp và tinh tế hơn.
2. **Hiệu quả tính toán cao hơn:** Hai lớp 3x3 có cùng "tầm nhìn" (trường tiếp nhận) như một lớp 5x5, nhưng lại sử dụng ít tham số hơn. Điều này giúp mô hình nhẹ hơn, huấn luyện nhanh hơn và giảm nguy cơ học vẹt (overfitting).

Kiến trúc cụ thể được triển khai trong ShapeModel như sau:

- **Input Layer:** Tiếp nhận ảnh xám kích thước (64, 64, 1).
- **Khối Trích xuất đặc trưng 1:**
    - **2 lớp Conv2D liên tiếp** (32 filters, kích thước 3x3): Sử dụng padding='same' để giữ nguyên kích thước không gian của ảnh sau khi tích chập, cho phép chồng xếp nhiều lớp.
    - MaxPooling2D (2x2): Giảm kích thước ảnh xuống một nửa 32x32.
    - BatchNormalization: Ổn định quá trình huấn luyện.
    - Dropout (0.25): Giảm thiểu overfitting ở giai đoạn đầu.
- **Khối Trích xuất đặc trưng 2:**
    - 2 lớp Conv2D liên tiếp (64 filters, kích thước 3x3, padding='same'). Tăng số lượng bộ lọc để học các đặc trưng phức tạp hơn.
    - MaxPooling2D (2x2): Giảm kích thước ảnh xuống 16x16.
    - BatchNormalization và Dropout (0.25).
- **Khối Trích xuất đặc trưng 3:**

- 2 lớp Conv2D liên tiếp (128 filters, kích thước 3x3, padding='same'). Ở độ sâu này, mạng tập trung học các cấu trúc hình học toàn thể.
- MaxPooling2D (2x2): Giảm kích thước ảnh xuống còn 8x8.
- BatchNormalization và Dropout (0.25).

- **Khối Phân loại (Classifier):**
  - Flatten: Duỗi tensor đặc trưng (8x8x128) thành vector 1 chiều.
  - Dense (256 units, 'relu') kết hợp Dropout (0.5).
  - Dense (128 units, 'relu') kết hợp Dropout (0.3). Cấu trúc Dense hình tháp giúp cô đọng thông tin dần dần.
  - Dense (8 units, 'softmax'): Lớp đầu ra với 8 nơ-ron tương ứng với 8 loại hình học, sử dụng hàm Softmax để đưa ra phân phối xác suất.



3.6.3. Quá trình huấn luyện

Các bước:

```python
def train(self, epochs=30, batch_size=64, samples_per_class=1000):
    print("Generating shape dataset...")
    X, y = ShapeGenerator.generate_dataset(
        samples_per_class=samples_per_class,
        size=self.input_size,
        augment=True
    )

    split_idx = int(0.8 * len(X))
    X_train, X_test = X[:split_idx], X[split_idx:]
    y_train, y_test = y[:split_idx], y[split_idx:]

    print(f"Training set: {len(X_train)} samples")
    print(f"Test set: {len(X_test)} samples")

    if self.model is None:
        self.build_model()

    early_stopping = keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        patience=5,
        restore_best_weights=True
    )

    reduce_lr = keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        min_lr=1e-7
    )

    self.history = self.model.fit(
        X_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(X_test, y_test),
        callbacks=[early_stopping, reduce_lr],
        verbose=1
    )
```

```python
    os.makedirs('models', exist_ok=True)
    self.model.save(self.model_path)

    test_loss, test_acc = self.model.evaluate(X_test, y_test, verbose=0)
    print(f"\nTest accuracy: {test_acc:.4f}")

    return self.history

def load_model(self):
    if os.path.exists(self.model_path):
        self.model = keras.models.load_model(self.model_path)
        return True
    return False

def predict(self, image):
    if self.model is None:
        if not self.load_model():
            raise ValueError("Model not trained or loaded")

    if len(image.shape) == 3:
        image = np.expand_dims(image, axis=0)

    predictions = self.model.predict(image, verbose=0)
    predicted_class = np.argmax(predictions[0])
    confidence = predictions[0][predicted_class]
    shape_name = self.shape_classes[predicted_class]

    return predicted_class, shape_name, confidence, predictions[0]

def predict_batch(self, images):
    if self.model is None:
        if not self.load_model():
            raise ValueError("Model not trained or loaded")

    predictions = self.model.predict(images, verbose=0)
    predicted_classes = np.argmax(predictions, axis=1)
    confidences = np.max(predictions, axis=1)
    shape_names = [self.shape_classes[cls] for cls in predicted_classes]

    return predicted_classes, shape_names, confidences, predictions

def get_model_summary(self):
    if self.model is None:
        self.build_model()
    return self.model.summary()
```

Mô hình được biên dịch với Optimizer Adam (learning rate 0.001) và Loss sparse_categorical_crossentropy.

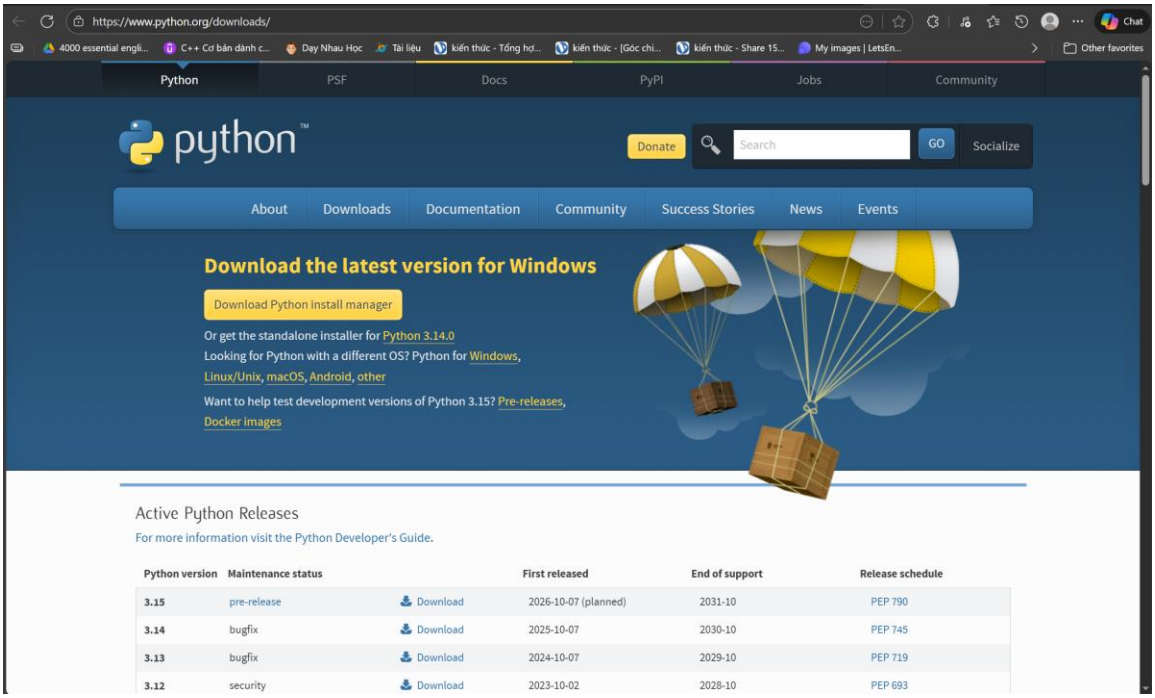
1. Sinh và chuẩn hóa dữ liệu hình học
    - Dữ liệu được tạo tự động bằng ShapeGenerator.generate_dataset().
    - Ảnh được chuẩn hóa về khoảng [0, 1] và reshape thành (64, 64, 1).
    - Dữ liệu được chia thành: 80% train và 20% test.
2. Sử dụng các callbacks để tối ưu quá trình huấn luyện
    - EarlyStopping: dừng huấn luyện sớm nếu *val_accuracy* không cải thiện sau 5 epochs → tránh overfitting.
    - ReduceLROnPlateau: giảm learning rate xuống 0.5 lần nếu *val_loss* không cải thiện sau 3 epochs → giúp mô hình hội tụ tốt hơn.
3. Lưu mô hình sau khi huấn luyện
    - Mô hình được lưu vào thư mục: models/shape_cnn_64.keras

IV: Cài đặt và kiểm thử

4.1 Cài đặt môi trường

4.1.1 Cài đặt python

- Tải python từ link sau: Link

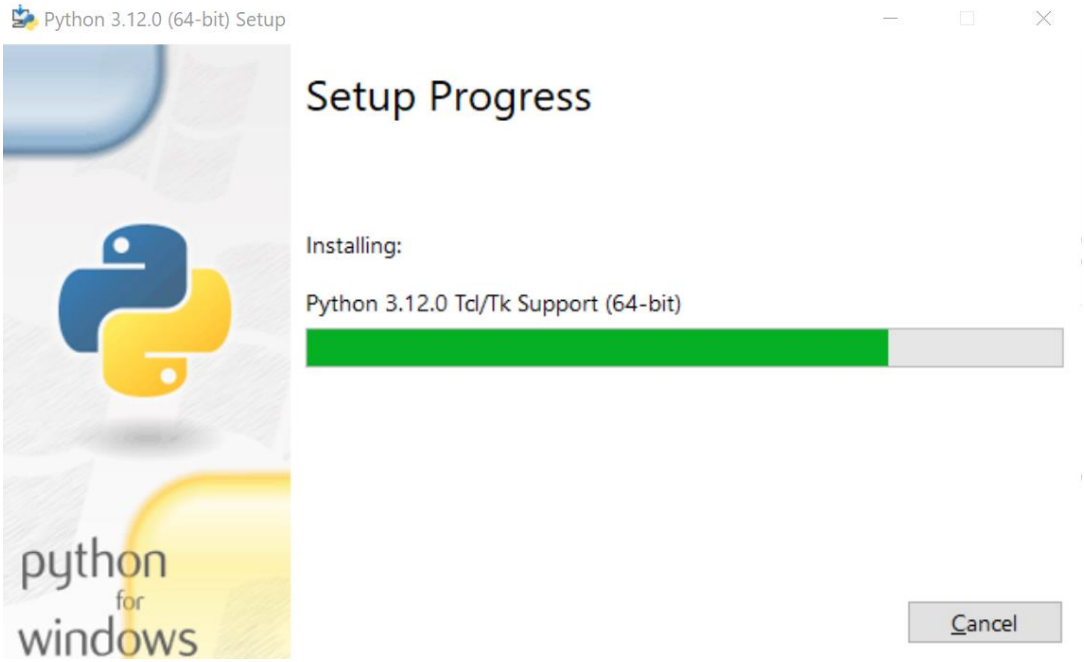


- Chọn bản windows installer:

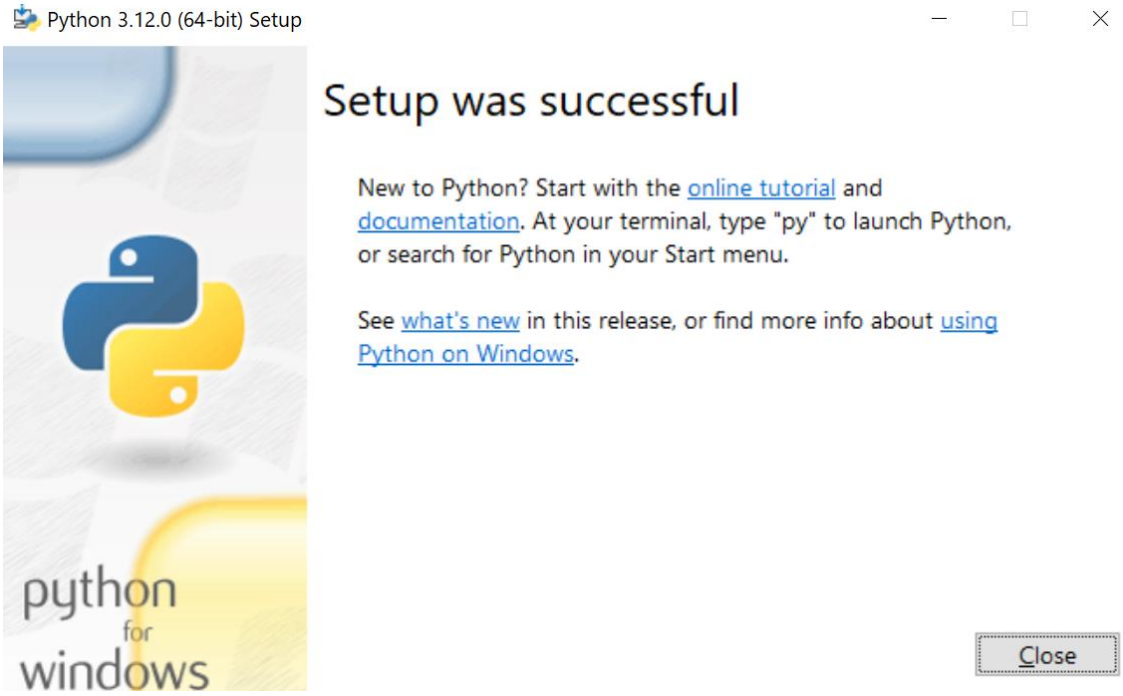
Version	Operating System	Description	MD5 Sum	File Size	Sigstore	GPG
Gzipped source tarball	Source release		d6eda3e1399cef5dfde7c4f319b0596c	25.9 MB	.sigstore	SIG
XZ compressed source tarball	Source release		f6f4616584b23254d165f4db90c247d6	19.6 MB	.sigstore	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	eddf6f35a3cbab94f2f83b2875c5fc27	43.3 MB	.sigstore	SIG
Windows installer (64-bit)	Windows	Recommended	32ab6a1058dfbde76951b7aa7c2335a6	25.3 MB	.sigstore	SIG
Windows installer (32-bit)	Windows		de59862985bf7afa639f2e4f9e2a722c	24.0 MB	.sigstore	SIG
Windows installer (ARM64)	Windows	Experimental	230c703e3b8b3d92765d118afa7b2f78	24.5 MB	.sigstore	SIG
Windows embeddable package (64-bit)	Windows		8e24d2b26a8dbf1da0694b9da1a08b2c	10.5 MB	.sigstore	SIG
Windows embeddable package (32-bit)	Windows		c2047dc270c4936f9c64619bb193b721	9.4 MB	.sigstore	SIG
Windows embeddable package (ARM64)	Windows		3da91ef1a86a8a210a32ea99c709dd93	9.8 MB	.sigstore	SIG

- Mở file .exe mới tải về, tích chọn ô add python.exe to PATH:

- Chọn Install now và chờ cho đến khi cài đặt xong:

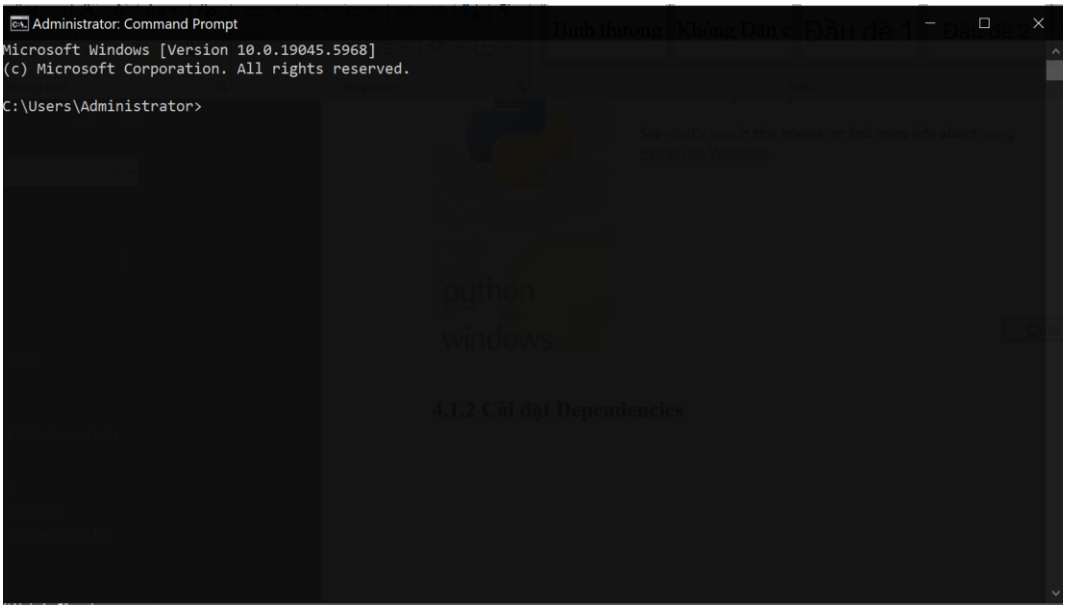


- Hoàn thành cài đặt:



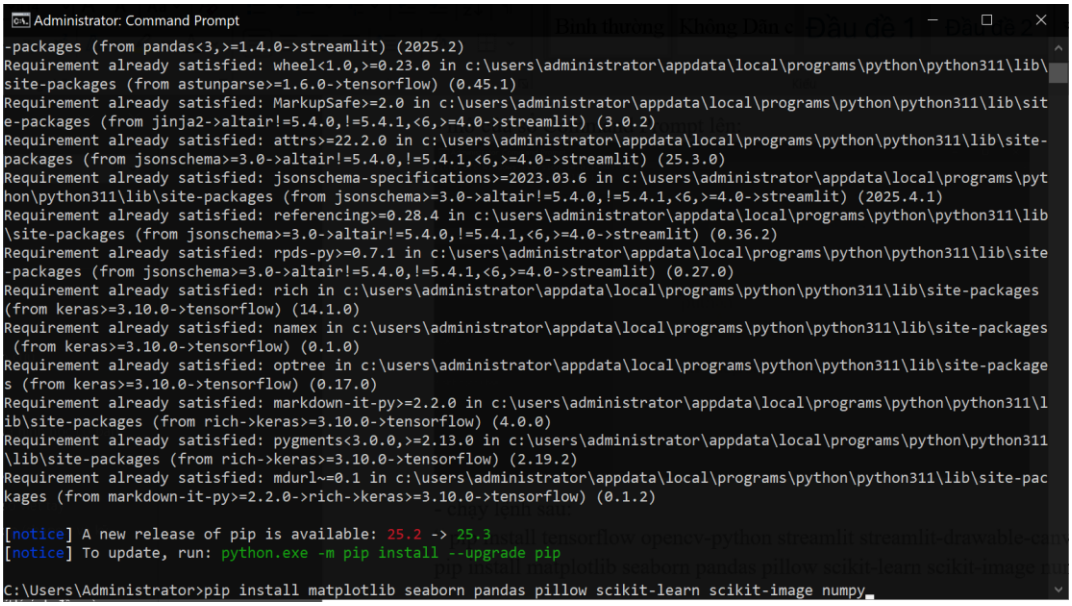
4.1.2 Cài đặt Dependencies

- mở cửa sổ Command Prompt lên:

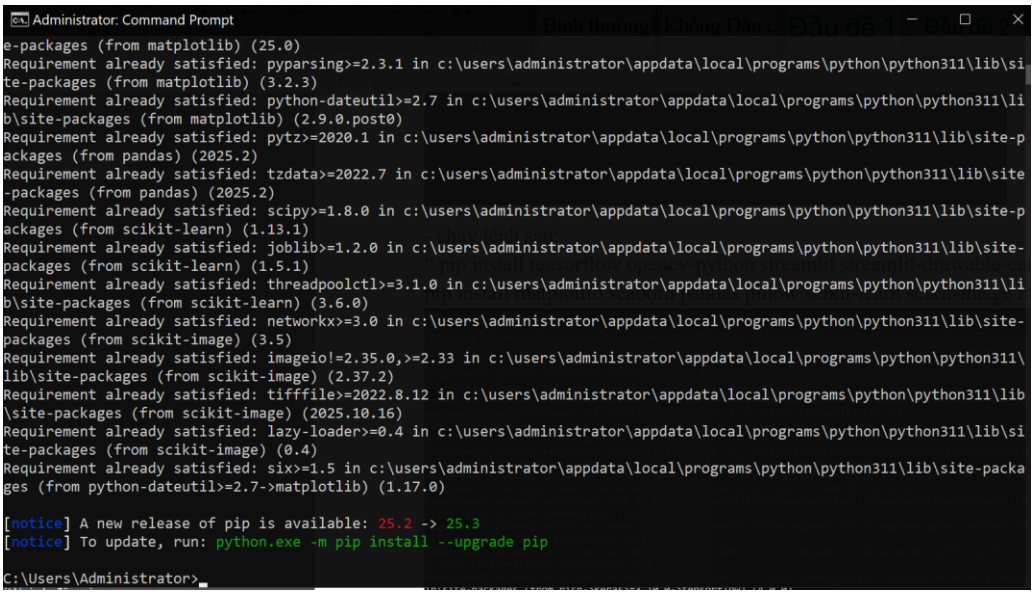


- chạy lệnh sau:

“pip install tensorflow opencv-python streamlit streamlit-drawable-canvas
pip install matplotlib seaborn pandas pillow scikit-learn scikit-image numpy”

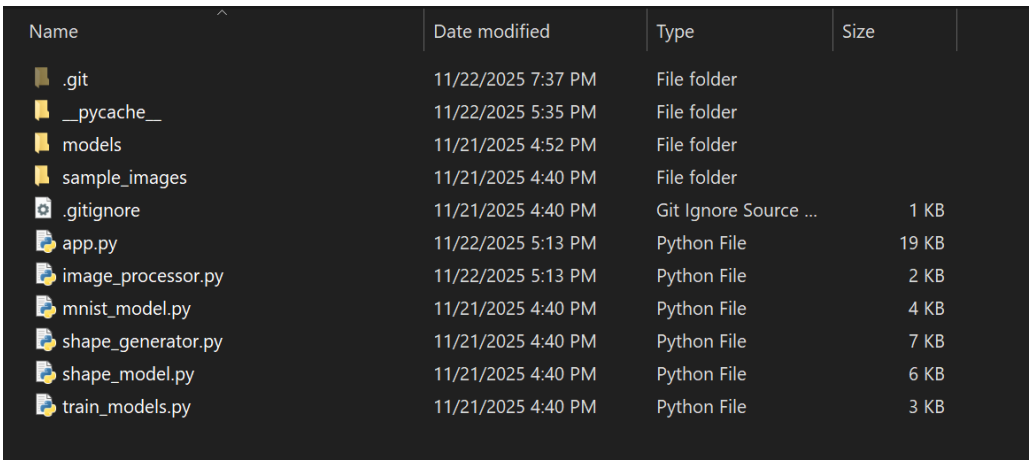


- bấm enter và chờ cho đến khi hoàn thành cài đặt:

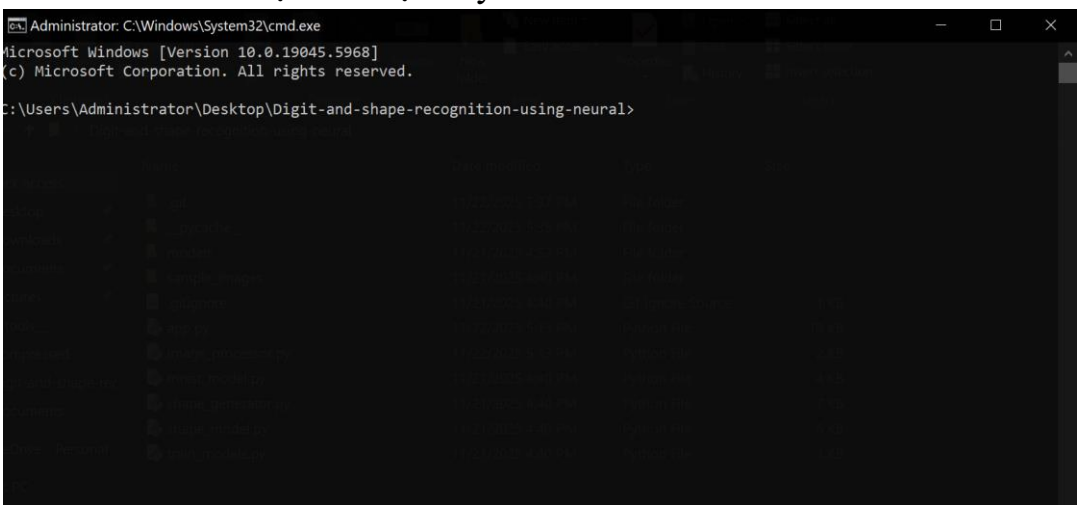


4.2 Chạy chương trình

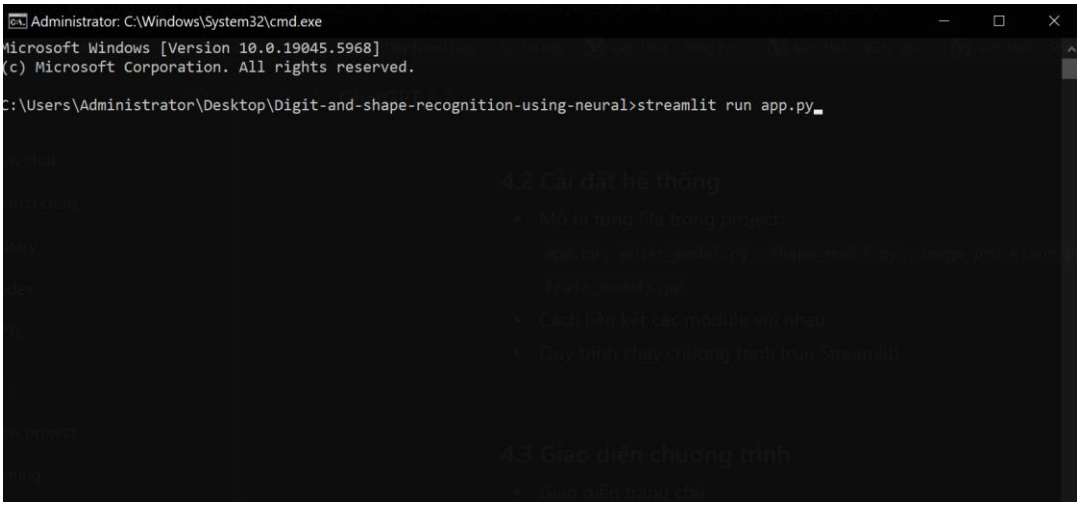
- Cấu trúc thư mục:



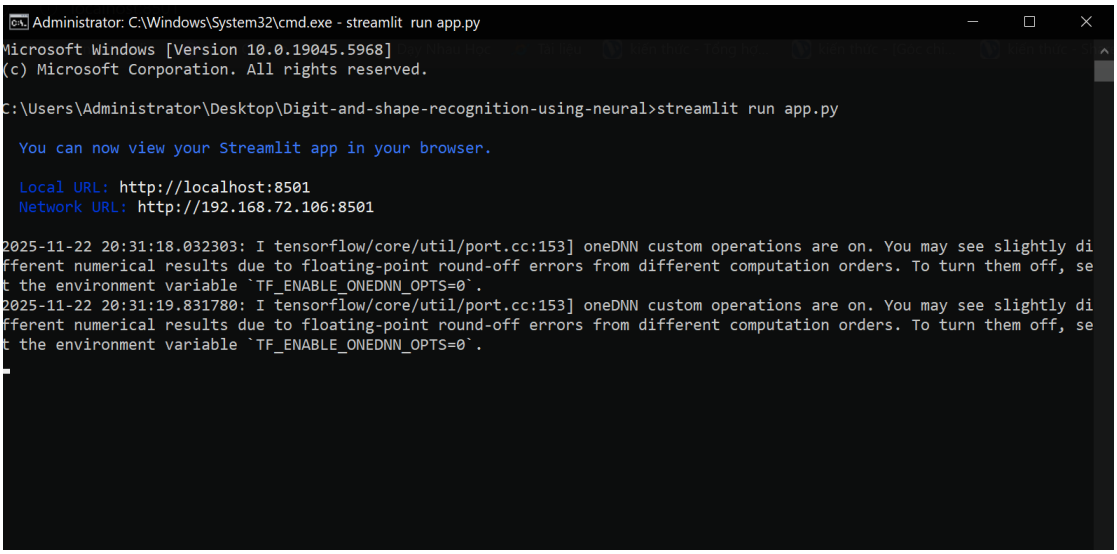
- mở cửa sổ cmd tại thư mục này:



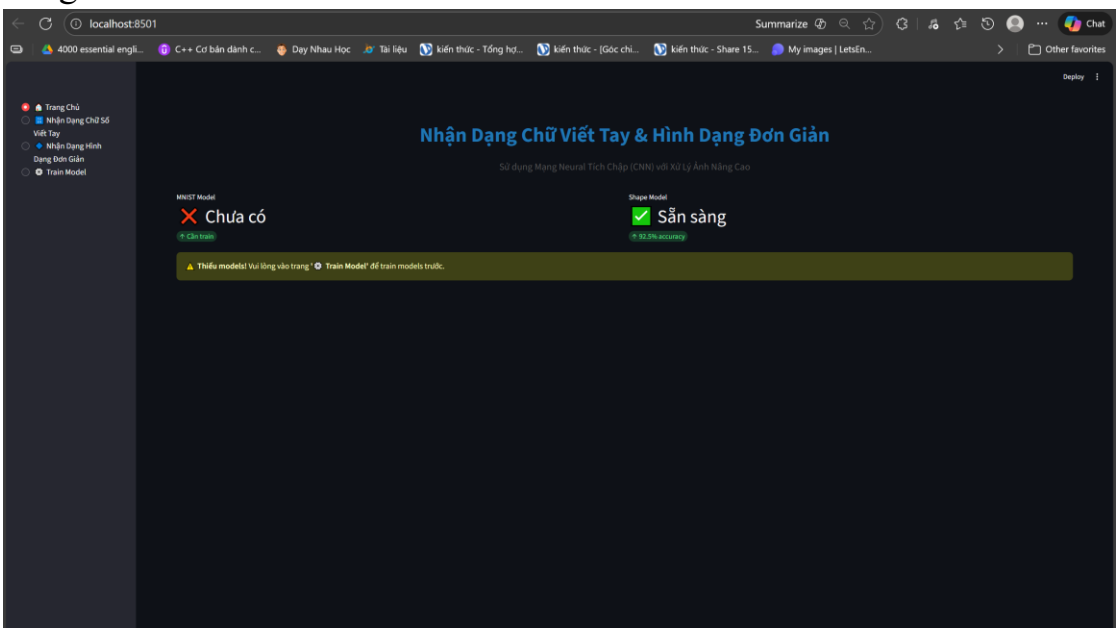
- nhập và chạy lệnh sau: “streamlit run app.py”



- nhấn enter:



- lúc này hệ thống sẽ tự động mở trình duyệt web lên và hiển thị giao diện người dùng:

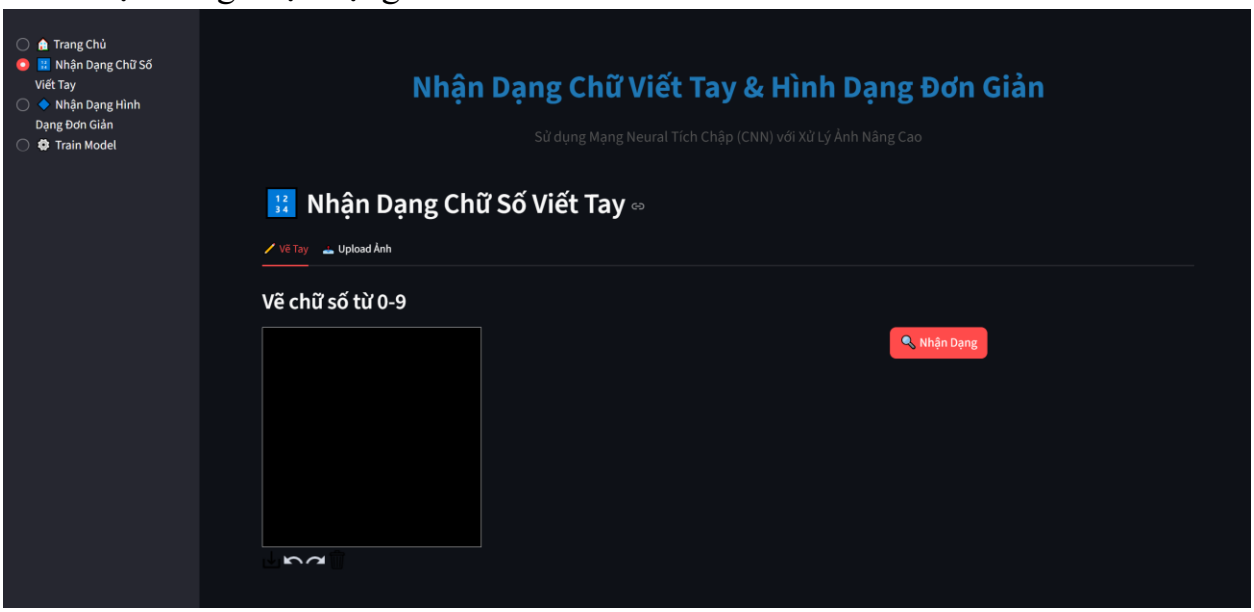


4.3 Kết quả cài đặt

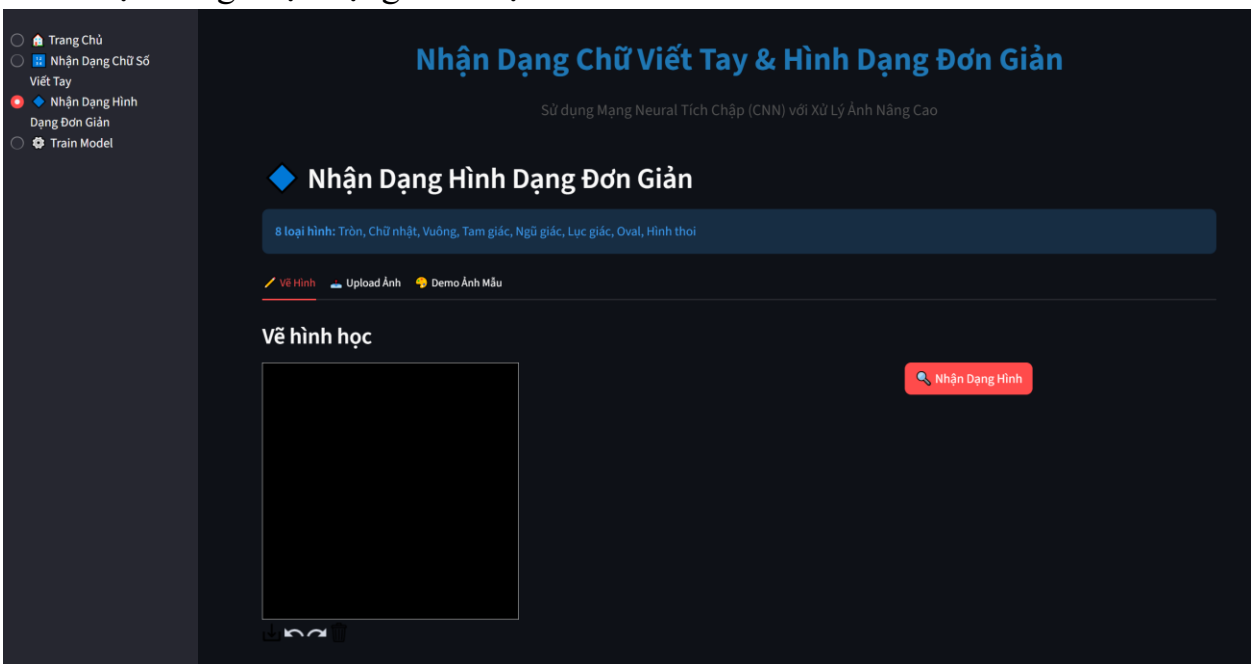
Các trang giao diện

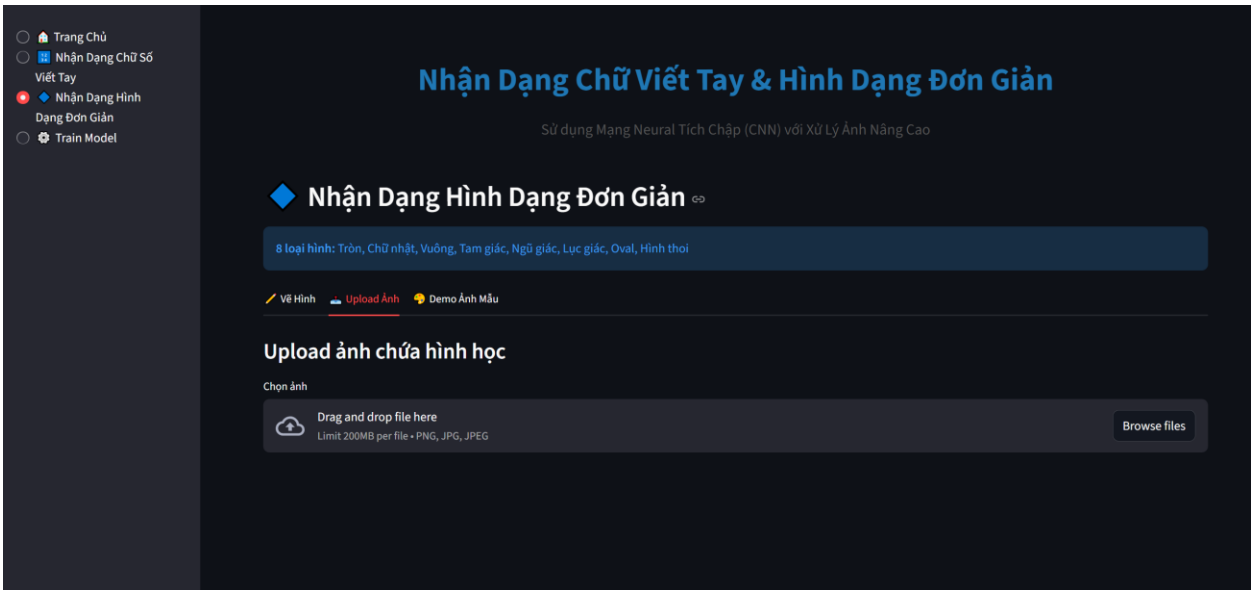
a) Giao diện trang chủ

b) Giao diện trang nhận dạng chữ số

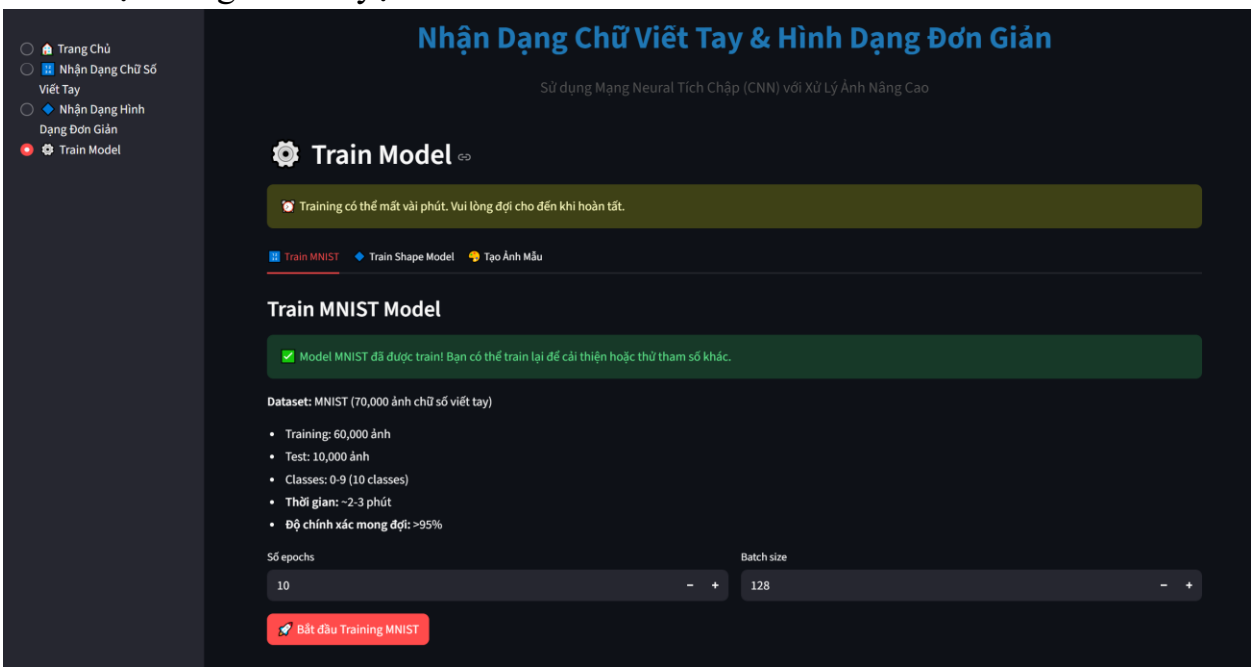


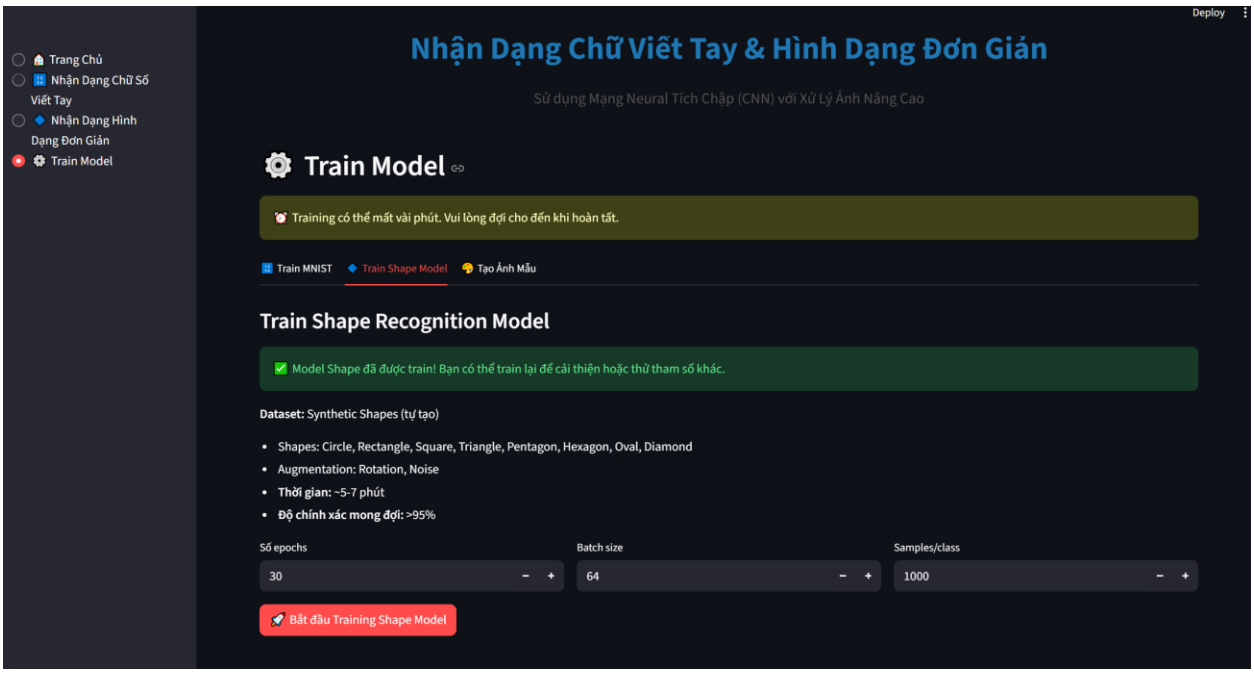
c) Giao diện trang nhận dạng hình học





d) Giao diện trang huấn luyện mô hình

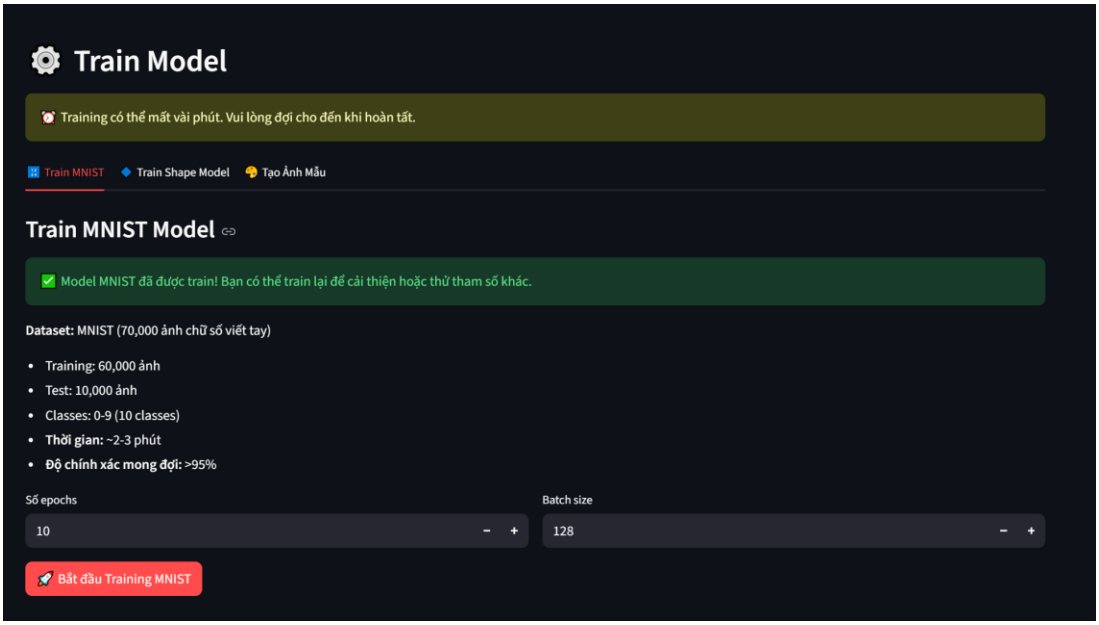




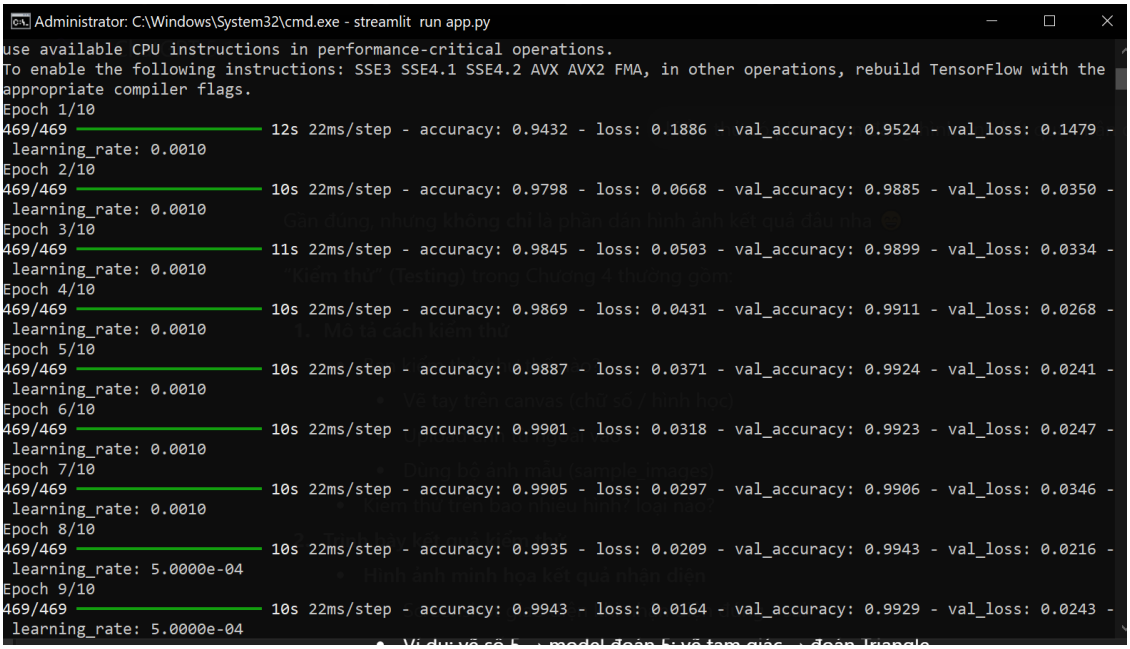
4.4 Kiểm thử

a) Huấn luyện mô hình:

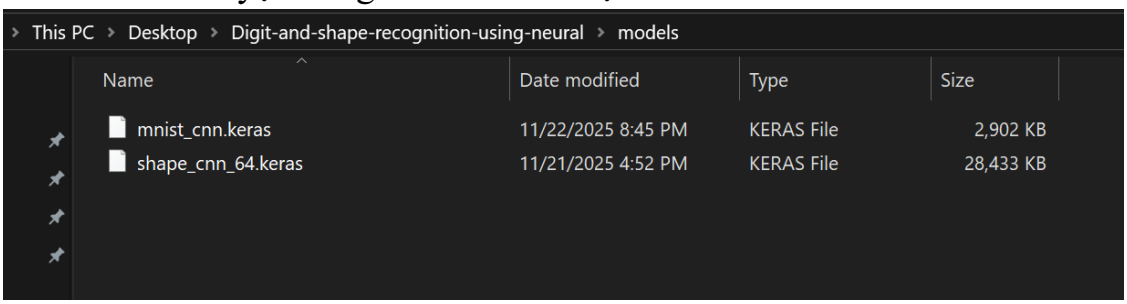
- Tại giao diện train model, ta có thể chọn huấn luyện mô hình nhận diện chữ số MNIST hoặc huấn luyện mô hình nhận dạng hình học



- Chọn số liệu huấn luyện mô hình:
  chọn số epochs và batch size đối với mô hình nhận diện chữ số.
  chọn số epochs, batch size và số lượng samples đối với mô hình nhận diện hình học
- Sau khi chọn số liệu huấn luyện thì bấm vào nút Bắt đầu Training. Ta có thể xem tiến trình huấn luyện mô hình ở cửa sổ cmd:

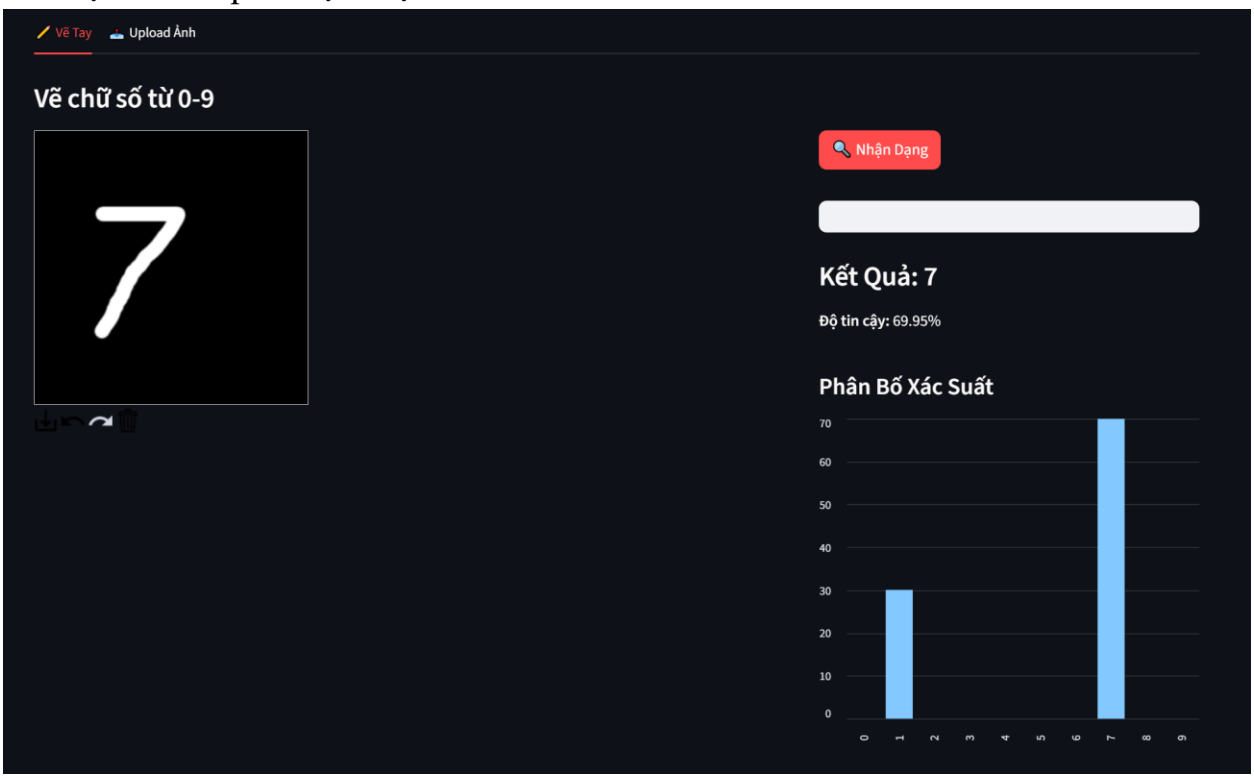


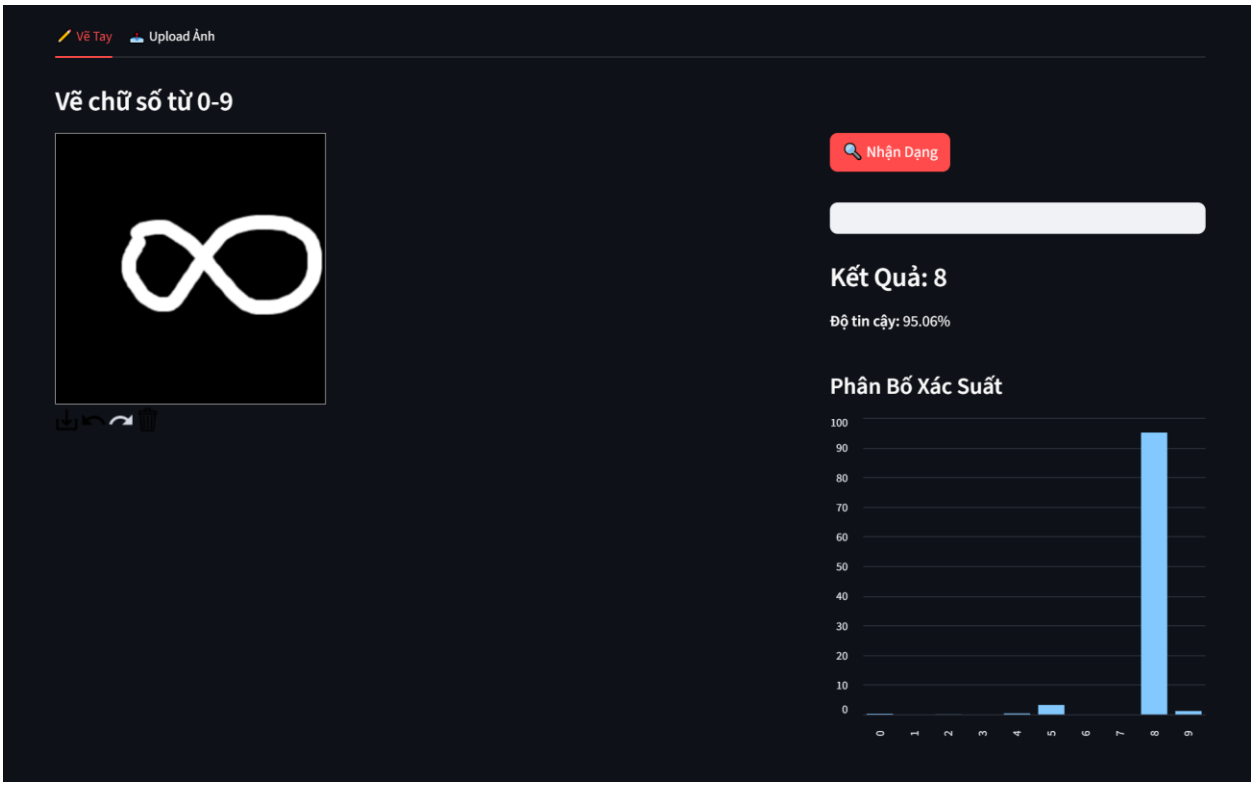
- Sau khi huấn luyện xong mô hình sẽ được lưu ở folder /models:



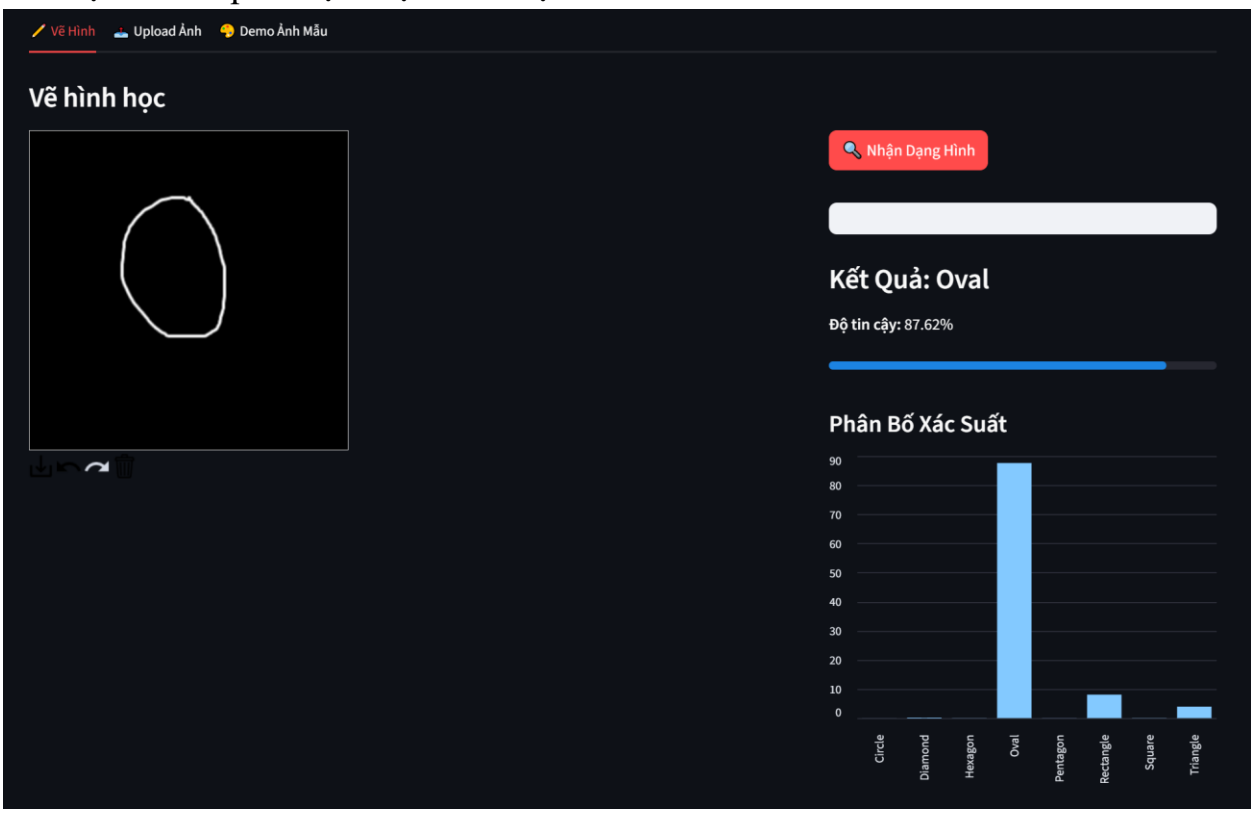
b) Nhận diện qua hình ảnh, vẽ:

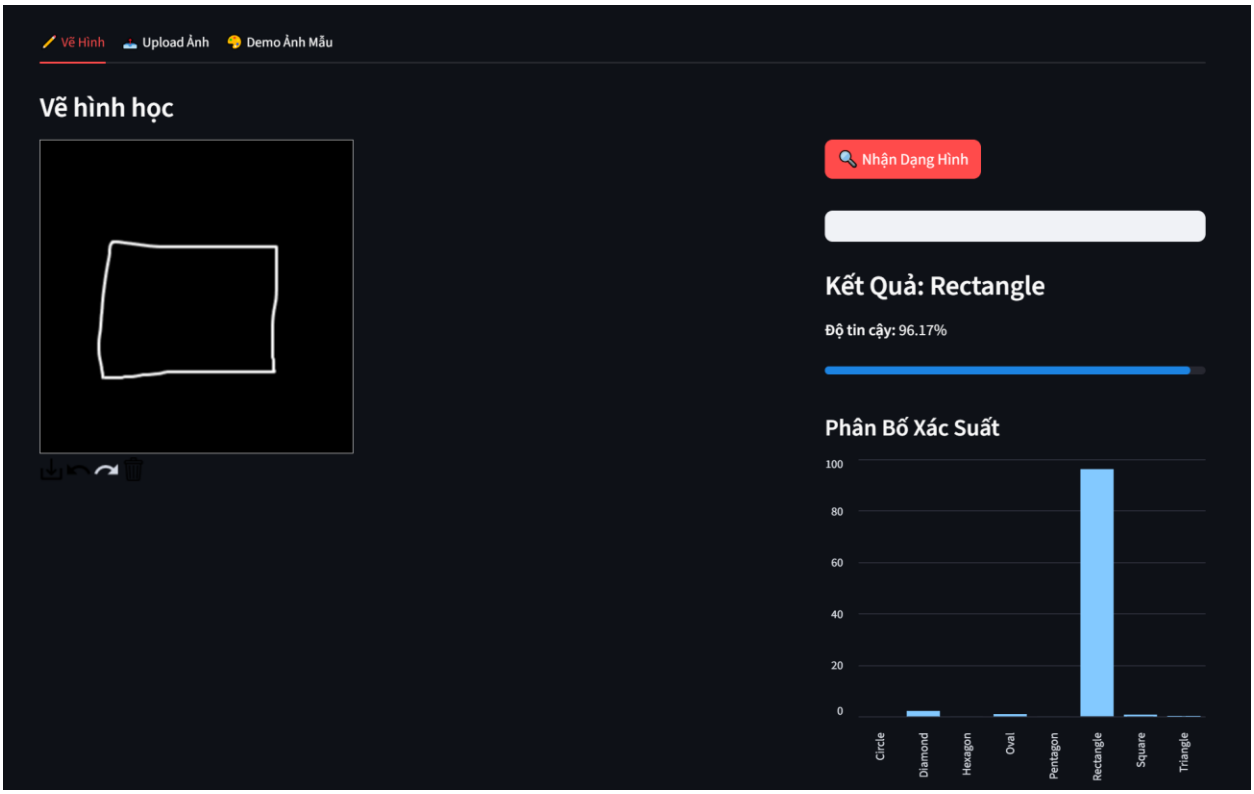
Một số kết quả nhận diện chữ số:





Một số kết quả nhận diện hình học:





Tổng kết

Kết luận chung về hệ thống

- Đề tài đã xây dựng thành công một hệ thống nhận dạng chữ số viết tay và hình học cơ bản dựa trên mô hình mạng nơ-ron tích chập (CNN) và giao diện người dùng trực quan trên Streamlit.

- Hệ thống bao gồm đầy đủ các thành phần từ nhập liệu (vẽ tay hoặc upload ảnh), tiền xử lý ảnh, dự đoán bằng mô hình đã huấn luyện, đến hiển thị kết quả cho người dùng.

- Hai mô hình chính trong hệ thống:

- MNISTModel: nhận dạng chữ số 0–9 với độ chính xác cao.
- ShapeModel: nhận dạng 8 loại hình học với độ chính xác tốt nhờ kiến trúc CNN theo phong cách VGG.

- Bên cạnh đó, việc tự xây dựng bộ dữ liệu hình học bằng ShapeGenerator giúp hệ thống chủ động về dữ liệu, không phụ thuộc vào nguồn ngoài và hỗ trợ tăng cường dữ liệu một cách linh hoạt.

- Hệ thống đã hoạt động ổn định, dễ sử dụng và phù hợp với mục tiêu ban đầu của đề tài.

Đánh giá ưu điểm và hạn chế của hệ thống

Ưu điểm

- Độ chính xác cao
    - Mô hình MNIST đạt hiệu suất tốt nhờ tập dữ liệu chuẩn.
    - Mô hình ShapeModel học được các đặc trưng hình học phức tạp.
- Kiến trúc rõ ràng, dễ mở rộng, tách thành từng module: xử lý ảnh, sinh dữ liệu, mô hình chữ số, mô hình hình học, giao diện.
- Tự động sinh dữ liệu hình học, giúp tăng độ đa dạng của dữ liệu → mô hình tổng quát tốt hơn.
- Giao diện người dùng trực quan: người dùng có thể vẽ, tải ảnh hoặc sử dụng ảnh mẫu một cách dễ dàng.

- Ứng dụng thực tế cao: có thể sử dụng trong giáo dục, mô phỏng thị giác máy, hoặc thực nghiệm AI.

Hạn chế

- Chưa hỗ trợ nhận dạng nhiều hình trong cùng một ảnh, mô hình hiện chỉ xử lý ảnh có một đối tượng duy nhất.
- Độ chính xác phụ thuộc vào nét vẽ của người dùng, chữ số hoặc hình vẽ quá xấu/méo dễ gây nhầm lẫn.
- Chưa áp dụng kỹ thuật tối ưu tiên tiến
  Ví dụ: không sử dụng Transfer Learning, không áp dụng Data Augmentation nâng cao cho MNIST.
- Thời gian train mô hình ShapeModel khá lâu, do dữ liệu tự sinh nhiều và mạng khá sâu.

Hướng phát triển và cải tiến trong tương lai

1. Mở rộng số lượng hình học, thêm các hình phức tạp hơn: ngôi sao, mũi tên, hình bất kỳ.
2. Nhận dạng nhiều đối tượng trong một ảnh, kết hợp với thuật toán phát hiện đối tượng (Object Detection) như YOLO, SSD.
3. Cải thiện mô hình bằng Transfer Learning: sử dụng các kiến trúc mạnh như MobileNet, EfficientNet, ResNet để tăng độ chính xác và tốc độ.
4. Tối ưu thời gian huấn luyện
5. Thiết kế giao diện đẹp và chuyên nghiệp hơn, có thể nâng cấp các animation, màu sắc, hoặc dashboard.
6. Đóng gói thành ứng dụng chạy độc lập hoặc triển khai trên web (Heroku, Streamlit Cloud)