BỘ KHOA HỌC VÀ CÔNG NGHỆ

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: XỬ LÝ ẢNH

Đề tài: Nhận dạng chữ viết tay hoặc hình đơn giản bằng mạng neural

Giảng viên: TS. Phạm Hoàng Việt

Nhóm 25

B22DCCN482 Trịnh Quang Lâm

B22DCCN434 Vũ Nhân Kiên

B22DCCN889 Vũ Thế Văn

Link sản phẩm: tvan16/Object_Detection_MNIST_SHAPE

Hà Nội, 11/2025

Phân chia nhiệm vụ

Mã sinh viên	Họ tên	Mức độ đóng góp
B22DCCN482	Trịnh Quang Lâm	30%
B22DCCN434	Vũ Nhân Kiên	30%
B22DCCN889	Vũ Thế Văn	40%

Mục lục

I. Mở đầu đề tài

1. Giới thiệu

1.1. Tổng quan về dự án

Dự án Unified Digits & Shapes Recognition System xây dựng một hệ thống nhận diện đối tượng thống nhất, có khả năng phát hiện và phân loại đồng thời chữ số viết tay (0–9) và các hình học cơ bản (9 loại) trong cùng một ảnh. Hệ thống được thiết kế theo kiến trúc hai giai đoạn: giai đoạn Detection dùng các kỹ thuật xử lý ảnh và detector để tìm vùng chứa đối tượng, giai đoạn Classification dùng mạng EfficientNet-B0 để phân loại từng vùng cắt. Trọng tâm của dự án nằm ở cách thiết kế pipeline tiền xử lý, tách nền, tìm contour, chuẩn hóa ảnh và tăng cường dữ liệu sao cho mô hình phía sau hoạt động ổn định.

1.2. Vấn đề cần giải quyết

Trong thực tế, chữ số và hình học thường xuất hiện chung trong bài kiểm tra, vở bài tập, bảng viết… nhưng lại có hình dạng, kích thước, vị trí và điều kiện chụp rất khác nhau. Nếu xử lý mỗi loại bằng một mô hình riêng, hệ thống trở nên phức tạp, khó triển khai và khó tối ưu. Đồng thời, dữ liệu ngủ nhiên (nhiễu, ánh sáng, góc chụp) khiến các phương pháp thuần phân loại đơn giản trên ảnh đã cắt sẵn không đủ mạnh. Bài toán đặt ra là thiết kế một pipeline xử lý ảnh đủ tốt để:

- Tách được đối tượng (digits/shapes) khỏi nền.
- Chuẩn hóa về cùng kích thước/kênh màu.
- Làm cho mô hình phía sau bất biến với xoay, tịnh tiến, phối cảnh và ánh sáng.

1.3. Giải pháp đề xuất

Để giải quyết những vấn đề trên, dự án đề xuất xây dựng một mô hình thống nhất có khả năng nhận diện cả chữ số và hình học trong cùng một pipeline xử lý. Mô hình này sử dụng một kiến trúc classification duy nhất cho 19 classes (10 chữ số + 9 hình học), giúp đơn giản hóa quá trình triển khai và giảm thiểu tài nguyên cần thiết.

Hệ thống được thiết kế với tính linh hoạt cao, hỗ trợ nhiều phương pháp detection khác nhau như Traditional Computer Vision, CRAFT (Character Region Awareness for Text Detection), và Hybrid Detector kết hợp cả hai phương pháp. Điều này cho phép hệ thống thích ứng với nhiều loại ảnh đầu vào

khác nhau, từ ảnh có nền đơn giản đến ảnh phức tạp với nhiều nhiễu và biến dạng.

2. Bối cảnh và Tầm quan trọng

2.1. Bối cảnh ứng dụng thị giác máy tính

Thị giác máy tính ngày càng được dùng trong chấm bài tự động, nhận diện biểu đồ, đọc form, bảng điểm… Ở những bài toán này, xử lý ảnh là bước đầu tiên và rất quan trọng: lọc nhiễu, tăng tương phản, tách nền/foreground, chuẩn hóa kích thước. Nếu bước xử lý ảnh không tốt, dù mô hình học sâu có mạnh đến đâu cũng khó đạt kết quả cao. Đề tài tận dụng MNIST nhưng mở rộng sang bối cảnh đa đối tượng, từ đó tạo môi trường phù hợp để áp dụng và đánh giá các kỹ thuật xử lý ảnh đã học trong môn.

2.2. Tầm quan trọng của bài toán

Thay vì chỉ dừng ở việc feed trực tiếp ảnh MNIST 28×28 vào mạng, đề tài thiết kế lại toàn bộ pipeline xử lý ảnh: từ CLAHE, Gaussian blur, adaptive threshold, morphological closing cho đến resize 128×128, chuyển grayscale → RGB và chuẩn hóa theo ImageNet. Những quyết định này ảnh hưởng trực tiếp tới khả năng phát hiện contour, chất lượng bounding box, cũng như việc mô hình phân biệt được các hình đa giác nhiều cạnh với hình tròn. Đây là những nội dung đúng trọng tâm Xử lí ảnh: tiền xử lý, biến đổi hình học và phân tích ảnh.

2.3. Ứng dụng trong giáo dục và EdTech

Trong bối cảnh giáo dục và sáng tạo số hiện đại, thị giác máy tính đang được sử dụng ngày càng rộng rãi cho nhiều nhiệm vụ khác nhau. Các hệ thống giáo dục thông minh sử dụng thị giác máy tính để theo dõi mức độ tương tác của người học, phân tích hành vi học tập, và đánh giá tiến độ. Các ứng dụng hỗ trợ học tập cá nhân hóa sử dụng công nghệ này để nhận diện và phân tích bài làm của học sinh, cung cấp phản hồi tức thời và gợi ý cải thiện. Việc xây dựng lớp học thông minh đòi hỏi khả năng nhận diện và theo dõi nhiều đối tượng cùng lúc, từ học sinh, giáo viên, đến các công cụ và tài liệu học tập. Các hệ thống tạo nội dung học liệu trực quan cần khả năng nhận diện và phân loại các thành phần trong ảnh để tự động tạo ra các bài tập, câu hỏi, hoặc tài liệu học tập phù hợp.

Những hệ thống như vậy thường phải xử lý các cảnh phức tạp với nhiều biểu tượng, vật thể học tập, hoặc tương tác của người học trong không gian lớp học vật lý hoặc ảo. Ví dụ, một hệ thống chấm điểm tự động cần nhận diện cả chữ số (điểm số) và hình học (biểu đồ, sơ đồ) trong cùng một bài kiểm tra. Một ứng dụng trò chơi giáo dục có thể yêu cầu nhận diện các hình dạng và số lượng để

tạo ra các thử thách học tập. Do đó, việc mở rộng bài toán từ nhận dạng chữ số đơn lẻ sang phát hiện và định vị nhiều đối tượng trong một khung hình có ý nghĩa thiết thực và quan trọng, giúp mô hình tiến gần hơn với các bài toán thực tế của EdTech (Educational Technology).

3. Động lực chọn MNIST mở rộng

MNIST là bộ dữ liệu kinh điển cho nhận dạng chữ số, dễ tải, dễ xử lý, nhưng ở dạng gốc chỉ chứa một chữ số/ảnh trên nền đơn giản. Khi kết hợp thêm shapes và sinh scene nhiều đối tượng, MNIST trở thành "bài toán xử lý ảnh" đúng nghĩa hơn: phải tìm vùng chứa đối tượng, phải cân nhắc resize sao cho không mất cạnh, phải tăng cường dữ liệu để chống aliasing và biến dạng. Việc chọn MNIST làm nền tảng giúp:

- Dễ so sánh với các mô hình kinh điển.
- Dễ sinh thêm dữ liệu synthetic phục vụ thử nghiệm bộ lọc, threshold, morphology.
- Tập trung được vào thiết kế pipeline xử lý ảnh hơn là xử lý dữ liệu quá phức tạp.

4. Mục tiêu nghiên cứu

4.1. Mục tiêu kỹ thuật

- Xây dựng pipeline thống nhất cho 19 lớp, trong đó phần xử lý ảnh đủ mạnh để:
    - Chuẩn hóa mọi ảnh về 128×128, 3 kênh, phân phối giống ImageNet.
    - Làm nổi bật biên/dạng hình (edges, contours) của digits/shapes.
    - Giảm nhiễu và sai lệch do ánh sáng, phối cảnh, aliasing.
- Đạt accuracy trên 99% trên tập validation với kiến trúc EfficientNet-B0.

4.2. Mục tiêu ứng dụng

- Minh hoạ một bài toán thực tế nơi xử lý ảnh là "lớp đầu tiên" bắt buộc trước khi đưa vào mạng học sâu.
- Cho phép tích hợp vào các ứng dụng demo (desktop/web) để chấm bài, tạo đề, trò chơi nhận diện số & hình.

5. Phạm vi thực hiện

- Tập trung vào ảnh tĩnh, nền trắng hoặc nền tương đối đơn giản, kích thước vừa phải.
- Đối tượng: chữ số 0–9 và 9 hình học cơ bản; không mở rộng sang chữ cái hoặc ảnh tự nhiên phức tạp.

- Phạm vi kỹ thuật:
    - Xử lý ảnh ở mức tiền xử lý + detection contour (OpenCV) + chuẩn hóa input cho CNN.
    - Không đi sâu vào segmentation nâng cao, tracking hoặc 3D vision.

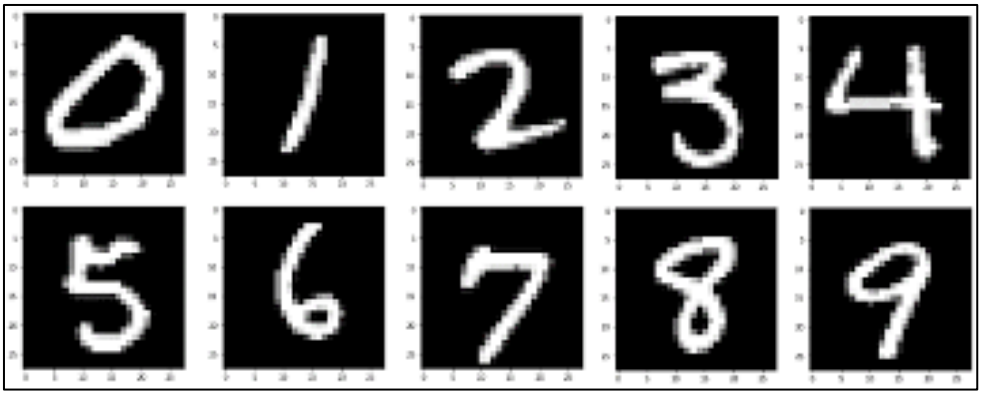
II. Tổng quan nghiên cứu

1. Tổng quan nghiên cứu & Công nghệ

1.1. Dữ liệu MNIST và các biến thể mở rộng

1.1.1. MNIST gốc và hạn chế

Dữ liệu MNIST gốc là bộ 70.000 ảnh chữ số viết tay kích thước 28×28 pixel, ảnh thang xám, với nhiệm vụ phân loại các chữ số từ 0 đến 9. Từ khi ra đời vào năm 1998, MNIST đã trở thành một benchmark chuẩn trong cộng đồng học máy và thị giác máy tính, thường được xem là "hello world" của lĩnh vực này nhờ cấu trúc đơn giản, dễ huấn luyện và có baseline rõ ràng. Tuy nhiên, bộ dữ liệu này bộc lộ nhiều giới hạn khi đưa vào bối cảnh ứng dụng thực tế. Ảnh chỉ chứa đơn đối tượng, không có các hình học phi chữ số, điều kiện ánh sáng và biến dạng hình học còn nghèo nàn, nên khó đánh giá khả năng nhận dạng đa mục tiêu cũng như tính bền vững của mô hình trước nhiễu và biến đổi phức tạp.



1.1.2. Các biến thể nâng cao

Để khắc phục một phần các hạn chế đó, nhiều biến thể nâng cao đã được đề xuất trong cộng đồng nghiên cứu. SVHN (Street View House Numbers) mở rộng bài toán sang nhận diện số nhà từ ảnh đường phố, với nhiều chữ số trong một ảnh và background phức tạp hơn. EMNIST mở rộng sang chữ cái in hoa và thường, tăng số lượng classes lên 62. Fashion-MNIST thay thế chữ số bằng các sản phẩm thời trang, tạo ra

một bài toán phân loại khác hoàn toàn nhưng vẫn giữ cấu trúc tương tự. KMNIST sử dụng ký tự tiếng Nhật, phản ánh sự đa dạng văn hóa. QuickDraw sử dụng các bản vẽ tay nhanh, tạo ra một bộ dữ liệu rất đa dạng và thú vị. Mỗi bộ hướng tới một loại đối tượng riêng và vì vậy rất hữu ích cho việc mở rộng không gian bài toán, tuy vậy, vẫn còn rất ít bộ dữ liệu ghép chữ số với các hình dạng cơ bản trong cùng một cảnh nhằm mô phỏng các "scene" giáo dục.

1.1.3. Dataset tùy biến của dự án

Trong project này, nhóm xây dựng một dataset tùy biến bằng cách sinh dữ liệu tổng hợp từ chữ số MNIST kết hợp với các shape primitives như hình tròn, hình vuông, tam giác, ngôi sao, và các đa giác từ ngũ giác đến cửu giác. Quá trình sinh dữ liệu được thực hiện một cách có hệ thống, với khả năng kiểm soát cao về vị trí, kích thước, góc quay, và mức độ chồng chéo của các đối tượng. Sau đó, các kỹ thuật augmentation được áp dụng như thêm nhiễu Gaussian, biến dạng affine, occlusion, rotation và scaling để mô phỏng môi trường craft và các điều kiện quan sát đa dạng hơn. Cấu trúc thư mục dataset/ cùng file label_mapping.json được thiết kế để quản lý nhãn thống nhất, hỗ trợ trực tiếp cho pipeline training và inference, đồng thời tạo nền tảng thuận lợi cho việc tái lập thí nghiệm và mở rộng bộ dữ liệu trong các nghiên cứu tiếp theo.

1.2. Phương pháp nhận dạng đối tượng

1.2.1. Object Detection (Faster R-CNN, SSD, YOLO)

Trong lĩnh vực thị giác máy tính, bài toán phát hiện đối tượng (object detection) là một nền tảng quan trọng và đã được nghiên cứu rất sâu trong nhiều thập kỷ qua. Các mô hình nổi bật như Faster R-CNN, SSD, và YOLO đều xây dựng trên nguyên tắc cơ bản: với mỗi ảnh đầu vào,

mạng lưới cần xác định đồng thời vị trí (thông qua bounding box với tọa độ) và lớp (class) của từng đối tượng trong ảnh.

YOLO (You Only Look Once) chia ảnh thành một lưới (grid), và với mỗi ô trong lưới, mô hình dự đoán khả năng tồn tại đối tượng cùng với bounding box và nhãn lớp tương ứng. SSD (Single Shot Detector) dựa trên nhiều ô giới hạn "anchor" được đặt trước ở các tỉ lệ khác nhau để phát hiện đa dạng kích thước đối tượng. Faster R-CNN có module 'Region Proposal Network' sinh các khung tiềm năng chứa đối tượng, sau đó một mạng con phân loại và tinh chỉnh bounding box cho từng vùng đề xuất.

1.2.2. Ưu nhược điểm của các phương pháp

Các kiến trúc này đạt hiệu suất rất tốt trên ảnh tự nhiên với nhiều đối tượng phức tạp, với độ chính xác cao và khả năng xử lý các tình huống khó như đối tượng nhỏ, chồng chéo, hoặc background phức tạp. Tuy nhiên, chúng đồng thời đòi hỏi công tác annotation rất phức tạp và tốn kém: mỗi ảnh trong tập huấn luyện cần được ghi chú tọa độ 4 cạnh bounding box và nhãn cho từng đối tượng. Đây là một bottleneck lớn khi xây dựng dữ liệu phức hợp hoặc dữ liệu giáo dục, đặc biệt khi cần số lượng lớn ảnh để đạt được hiệu năng tốt. Thêm nữa, các mô hình detection tiêu chuẩn thường nặng về mặt tính toán, với số lượng tham số lớn (hàng chục đến hàng trăm triệu tham số), chi phí huấn luyện và suy luận cao, nên khó phù hợp với thiết bị phổ thông hoặc nhu cầu inference real-time trên bài toán nhỏ với ảnh 28×28 hoặc 64×64 pixel, số lượng đối tượng ít như trường hợp MNIST mở rộng ghép shape-digit.

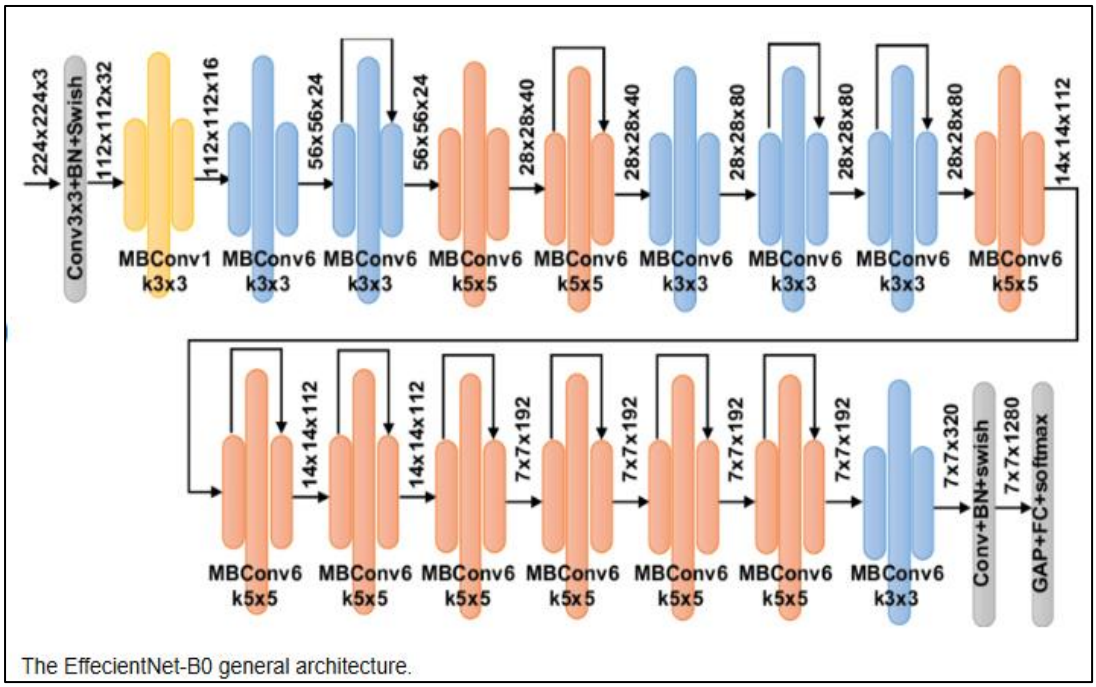
1.2.3. Lý do chọn tiếp cận Unified Classifier

Trước những thách thức của các phương pháp object detection truyền thống, thuật toán của project chọn một hướng tiếp cận nhẹ hơn và phù hợp hơn với bài toán cụ thể. Thay vì sử dụng một mô hình detection phức tạp yêu cầu annotation bounding box, dự án sử dụng một kiến trúc hai giai đoạn kết hợp: giai đoạn đầu sử dụng các phương pháp detection đơn giản và hiệu quả (Traditional Computer Vision hoặc CRAFT) để tìm vị trí các đối tượng, và giai đoạn thứ hai sử dụng một mô hình classification thống nhất để nhận diện loại của từng đối tượng đã được cắt ra.

1.3. Tiếp cận "Unified Classifier"

1.3.1. Kiến trúc CNN đa nhánh

Kiến trúc classification thống nhất sử dụng EfficientNet-B0 làm backbone, một kiến trúc CNN được tối ưu hóa đặc biệt về hiệu quả tính toán và độ chính xác. EfficientNet-B0 được pre-trained trên ImageNet với hàng triệu ảnh và hàng nghìn classes, do đó đã học được các đặc trưng cơ bản như cạnh, góc, kết cấu, và hình dạng. Thay vì phải học lại từ đầu, mô hình chỉ cần fine-tune classifier layer cuối cùng để phân loại 19 classes (10 chữ số + 9 hình học). Điều này giúp giảm đáng kể thời gian huấn luyện và lượng dữ liệu cần thiết, đồng thời vẫn đạt được độ chính xác cao nhờ tận dụng kiến thức đã học từ ImageNet.



The EffecientNet-B0 general architecture.

1.3.2. Shape head và Digit head

Trong implementation thực tế của dự án, không sử dụng kiến trúc đa nhánh với shape head và digit head riêng biệt. Thay vào đó, sử dụng một classifier layer duy nhất cho 19 classes. Tuy nhiên, khái niệm "unified" ở đây có nghĩa là một mô hình duy nhất có thể phân loại cả digits và shapes, thay vì cần hai mô hình riêng biệt.

1.3.3. Ưu nhược điểm của phương pháp

- Ưu điểm:

  Ưu điểm chính của phương pháp unified classifier này là loại bỏ hoàn toàn nhu cầu annotation bounding box cho tập huấn luyện.

Đặc biệt hữu ích khi sinh tổng hợp dữ liệu hoặc craft giáo dục, chỉ cần biết loại đối tượng xuất hiện trong ảnh là đủ, không cần khung vị trí chính xác. Điều này giảm chi phí xây dựng bộ dữ liệu và tăng khả năng sinh tự động, cho phép tạo ra hàng trăm nghìn ảnh huấn luyện một cách nhanh chóng và dễ dàng.

Pipeline được thiết kế đơn giản và dễ tích hợp. Với dữ liệu và nhãn đơn giản, việc lập trình và kiểm thử diễn ra nhanh chóng. Việc sinh, gán nhãn, và huấn luyện đều có thể thực hiện hoàn toàn tự động mà không cần labeler thủ công. Điều này tạo ra một môi trường phát triển linh hoạt, cho phép thử nghiệm nhanh các ý tưởng mới và điều chỉnh dễ dàng khi cần.

Tốc độ suy luận cao và tiết kiệm tài nguyên là một ưu điểm quan trọng khác. Không phải duyệt qua nhiều anchor hoặc vùng đề xuất như các phương pháp detection truyền thống, mô hình chỉ gồm một backbone EfficientNet-B0 nhẹ và một classifier layer nhỏ gọn. Điều này rất ý nghĩa với inference real-time trên thiết bị phổ thông hoặc khi cần tích hợp demo giáo dục trực tuyến ngay trên laptop cá nhân mà không cần GPU mạnh.

Việc sử dụng một mô hình thống nhất cho cả chữ số và hình học cũng tận dụng được các đặc trưng chung giữa hai loại đối tượng. Cả chữ số và hình học đều có các đặc trưng hình học cơ bản như cạnh, góc, đường cong, và đoạn thẳng. Bằng cách học chung một representation space, mô hình có thể tận dụng hiệu quả hơn dữ liệu hạn chế và tăng khả năng khái quát hóa. Điều này đặc biệt quan trọng khi dataset có hạn hoặc các class xuất hiện không cân bằng.

- Nhược điểm:

Tuy nhiên, phương pháp này cũng có những nhược điểm cần chú ý. Đầu tiên là không xác định được vị trí chính xác của từng đối tượng trong ảnh gốc. Nếu muốn "click đúng vị trí hình tròn" hoặc trả lời hình tròn xuất hiện ở đâu, mô hình bất lực. Tương tự, khi ảnh chứa nhiều đối tượng cùng loại (ví dụ 2 tam giác), unified classifier vẫn chỉ biết "có tam giác", không đếm được số lượng hay chỉ vị trí từng cái. Điều này làm mất tính ứng dụng trong bài toán cần localization hoặc chấm điểm thao tác, hoặc các trò chơi giáo dục kiểm tra vị trí tương tác.

Thứ hai, phương pháp này khó mở rộng khi đối tượng hoặc phân bố dày đặc. Nếu tăng độ phức tạp của cảnh (ví dụ, mô phỏng lớp học với 6-7 dạng hình và nhiều chữ số), vector nhãn đầu ra sẽ rất dài, dẫn đến hiệu năng classifier bị suy giảm do phải dự đoán xác suất cho nhiều lớp cùng lúc. Trong khi các detector như YOLO phân giải bài toán spatial tốt hơn qua logic anchor và lưới chia vùng.

Cuối cùng, phương pháp này không dùng được cho ảnh tự nhiên hoặc composition phức tạp. Khi muốn phát hiện nhiều vật thể xếp chồng lớp, overlap hoặc cần trích xuất patch vùng quan tâm riêng biệt, cách tiếp cận classifier này sẽ lỗi thời, cần quay lại phương án object detector hoặc segmentation network có khả năng localization..

III. Augumentation & Tiền xử lý

1. Augmentation & Tiền xử lý

1.1. Lý do cần Augumentation

Khi xây dựng bộ dữ liệu tổng hợp cho bài toán nhận diện shape và digit, một vấn đề quan trọng nảy sinh là dữ liệu có xu hướng bị trùng lặp về pattern một cách đáng kể. Ví dụ, các chữ số và hình học thường xuất hiện ở những vị trí tương tự nhau trong khung hình, với tỷ lệ gần như cố định, hoặc trong điều kiện ánh sáng và tương phản rất giống nhau. Điều này tạo ra một bộ dữ liệu có tính đồng nhất cao, nơi mô hình học máy dễ dàng "học vẹt" các pattern cụ thể của dữ liệu tổng hợp thay vì học các đặc trưng sâu sắc và tổng quát hơn.

1.2. Vấn đề overfitting với dữ liệu tổng hợp

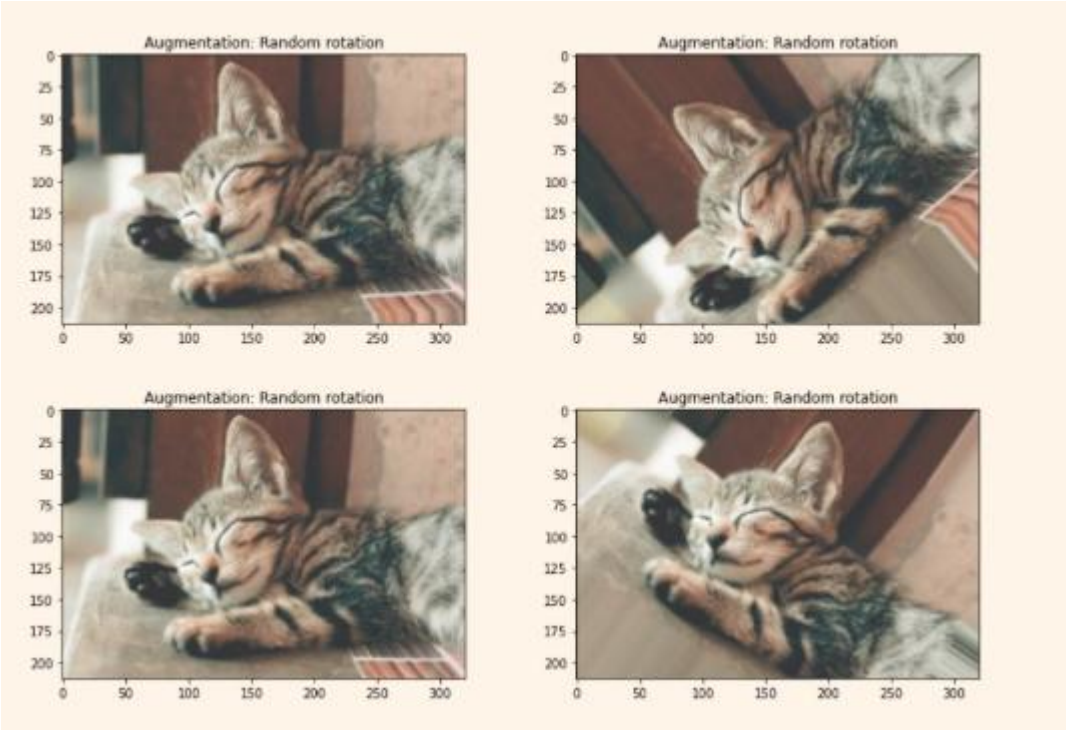
Hậu quả là mô hình bị overfitting, thể hiện qua việc đạt độ chính xác rất cao trên tập huấn luyện nhưng lại cho kết quả kém khi gặp các tình huống thực tế đa dạng hơn hoặc các loại nhiễu chưa từng gặp trong quá trình huấn luyện. Data augmentation là một kỹ thuật quan trọng và phổ biến trong học máy, được thiết kế để tạo ra các biến thể của ảnh gốc thông qua các phép biến đổi hình học, thêm nhiễu, đảo màu, và nhiều kỹ thuật khác. Mục tiêu là giúp mô hình gặp "nhiều mặt" khác nhau của cùng một bài toán, từ đó tăng khả năng tổng quát hóa và khả năng chống lại overfitting. Đặc biệt, các bài toán tổng hợp dễ khan hiếm các tình huống "lệch chuẩn" như vật thể bị quay, bị méo, vị trí thay đổi, dính nhiễu hoặc bị che khuất một phần. Augmentation mạnh sẽ mô phỏng

những thay đổi này một cách có hệ thống, đưa bài toán đến gần với thực tế hơn và giúp mô hình học được các đặc trưng bền vững hơn.

2. Các kỹ thuật augmentation đã sử dụng

2.1. Phép quay ngẫu nhiên (Random Rotation ±30°)

Phép quay ngẫu nhiên là một trong những kỹ thuật augmentation cơ bản và quan trọng nhất được áp dụng trong dự án. Mục đích chính của kỹ thuật này là làm cho mô hình quen với việc nhận diện chữ số và hình học xuất hiện ở nhiều góc quay khác nhau, chứ không chỉ dừng lại ở phương thẳng đứng như trong dữ liệu gốc. Trong thực tế, đặc biệt trong môi trường giáo dục, học sinh thường đặt bài kiểm tra hoặc vẽ hình ở các góc độ khác nhau, không phải lúc nào cũng hoàn toàn thẳng đứng. Tương tự, khi sử dụng camera để chụp ảnh, góc chụp có thể bị lệch do cách cầm máy hoặc vị trí đặt máy. Phép xoay giúp mô hình đề phòng và xử lý tốt những tình huống này.



Code minh họa: **transforms.RandomRotation(30)**

Giải thích code:

- RandomRotation(30): Tạo một phép biến đổi quay ngẫu nhiên
- Tham số: 30
- có nghĩa là góc quay sẽ được chọn ngẫu nhiên trong khoảng từ -30° đến +30°

- Mỗi lần áp dụng transform, một góc quay ngẫu nhiên sẽ được chọn trong khoảng này
- Ảnh sẽ được xoay quanh tâm của nó với góc quay đã chọn

Lý do chọn 30°:

- Quá lớn (>45°): Digits/shapes khó nhận diện, dễ nhầm lẫn (ví dụ "6" và "9" khi xoay 180°)
- Quá nhỏ (<15°): Không đủ diversity, không mô phỏng được các tình huống thực tế
- 30°: Cân bằng tốt giữa tăng diversity và giữ được khả năng nhận diện

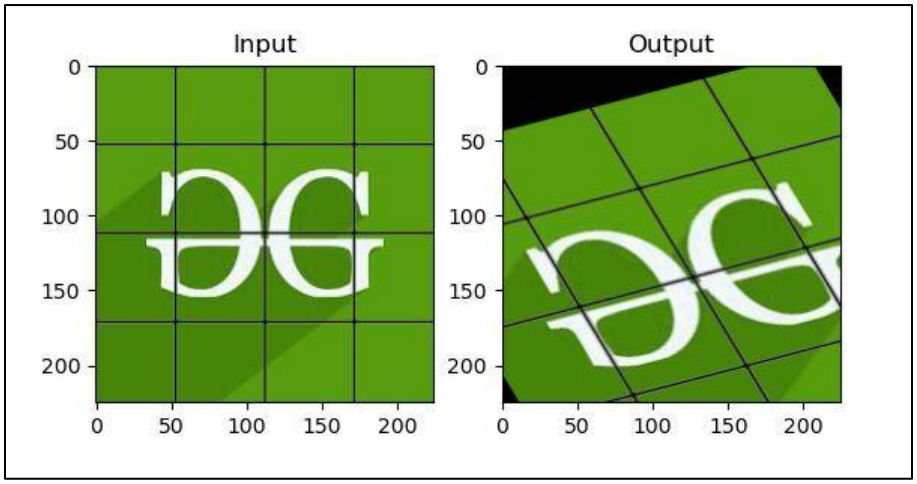
Ví dụ tác động:

- Chữ số "6" xoay 30° → vẫn là "6" nhưng model học được tính bất biến với rotation
- Hình vuông xoay 30° → thành hình thoi (vẫn nhận diện được là Square)
- Hình tam giác xoay 30° → vẫn giữ được đặc trưng 3 cạnh

2.2. Affine Transformation (Translation)

Affine transformation là một nhóm các phép biến đổi hình học bao gồm translation (dịch chuyển), scaling (thay đổi tỷ lệ), rotation (quay), và shear (biến dạng xiên). Trong dự án này, chỉ sử dụng translation, trong khi các phép biến đổi khác được xử lý riêng hoặc không sử dụng vì lý do cụ thể.

Translation, hay dịch chuyển, là phép biến đổi di chuyển đối tượng từ vị trí này sang vị trí khác trong khung hình mà không thay đổi hình dạng hay kích thước. Mục đích của translation là giúp mô hình học được tính bất biến với vị trí, tức là khả năng nhận diện đối tượng bất kể nó xuất hiện ở đâu trong ảnh. Điều này rất quan trọng vì trong thực tế, các đối tượng có thể xuất hiện ở bất kỳ vị trí nào trong khung hình, không phải lúc nào cũng ở trung tâm như trong nhiều bộ dữ liệu được chuẩn hóa.



Code minh họa:

```
transforms.RandomAffine(
    degrees=0,              # Không xoay (đã có RandomRotation riêng)
    translate=(0.15, 0.15), # Dịch 15% theo chiều ngang (x) và dọc (y)
    scale=None,             # Không scale (giữ nguyên kích thước)
    shear=None              # Không shear (tránh biến dạng quá mạnh)
)

```

Giải thích code:

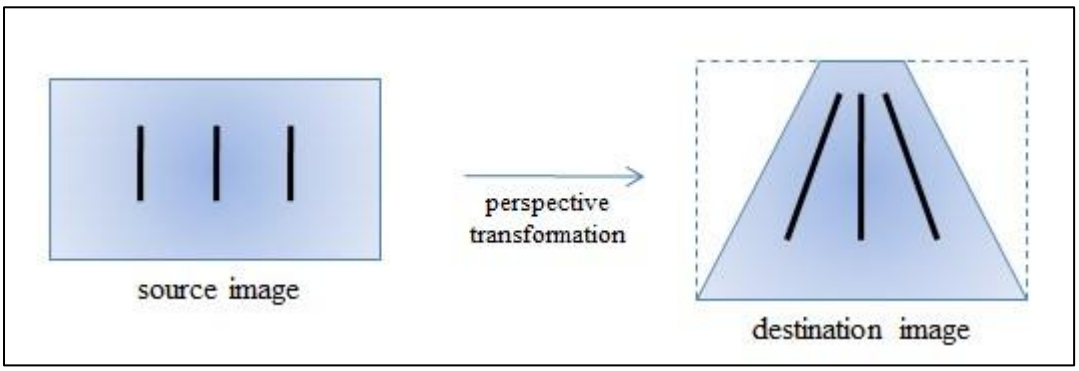
- degrees=0: Không áp dụng rotation vì đã có
- RandomRotation: riêng, tránh trùng lặp
- translate=(0.15, 0.15):
    - Giá trị đầu tiên (0.15) là tỷ lệ dịch chuyển theo chiều ngang (trục x)
    - Giá trị thứ hai (0.15) là tỷ lệ dịch chuyển theo chiều dọc (trục y)
    - Với ảnh 128x128, có thể dịch tối đa 19 pixels ($128 \times 0.15 \approx 19$) theo mỗi chiều
- scale=None: Không thay đổi tỷ lệ vì ảnh đã được resize về 128x128 cố định
- shear=None: Không áp dụng biến dạng xiên để tránh làm méo shapes

Lý do không dùng Scale và Shear:

- Scale: Không cần vì đã resize về 128x128 cố định, scale sẽ làm mất thông tin hoặc tạo padding không mong muốn
- Shear: Tránh biến dạng quá mạnh có thể làm mất đặc trưng của shapes (đặc biệt các hình có nhiều cạnh như Nonagon, Octagon)

2.3. Perspective Transformation (Biến đổi phối cảnh)

Perspective transformation là một kỹ thuật mô phỏng hiệu ứng phối cảnh 3D, tạo ra cảm giác như nhìn đối tượng từ một góc nghiêng thay vì nhìn thẳng từ trên xuống. Điều này rất quan trọng trong thực tế, vì khi chụp ảnh bằng camera hoặc điện thoại, góc chụp hiếm khi hoàn toàn vuông góc với mặt phẳng chứa đối tượng. Kết quả là ảnh có thể bị méo do phối cảnh, với các đối tượng ở xa trông nhỏ hơn và các đường thẳng song song có thể hội tụ về một điểm.



Code minh họa:

```
transforms.RandomPerspective(
    distortion_scale=0.1,
    p=0.3
)
```

Giải thích code:

- distortion_scale=0.1: Độ biến dạng phối cảnh là 10%

    - Giá trị này xác định mức độ "nghiêng" của ảnh
    - 0.0 = không biến dạng, 1.0 = biến dạng tối đa
    - Giá trị 0.1 được chọn để vừa tạo biến dạng vừa giữ được đặc trưng edges
- p=0.3: Xác suất áp dụng transformation là 30%
    - Chỉ 30% số ảnh trong batch sẽ được áp dụng perspective transformation
    - Giúp cân bằng giữa tăng diversity và giữ nguyên dữ liệu gốc

Lý do điều chỉnh:

- Ban đầu:distortion_scale=0.2, p=0.5
- Sau khi phân tích: Phát hiện confusion giữa Nonagon và Circle tăng lên
- Giải pháp: Giảm xuống 0.1, p=0.3 để giữ được đặc trưng edges của shapes, đặc biệt các hình có nhiều cạnh

Tác động:

- Hình vuông nhìn từ trên → Hình thang (perspective)
- Mô phỏng góc chụp nghiêng trong thực tế
- Tăng robustness nhưng không làm mất đặc trưng quan trọng

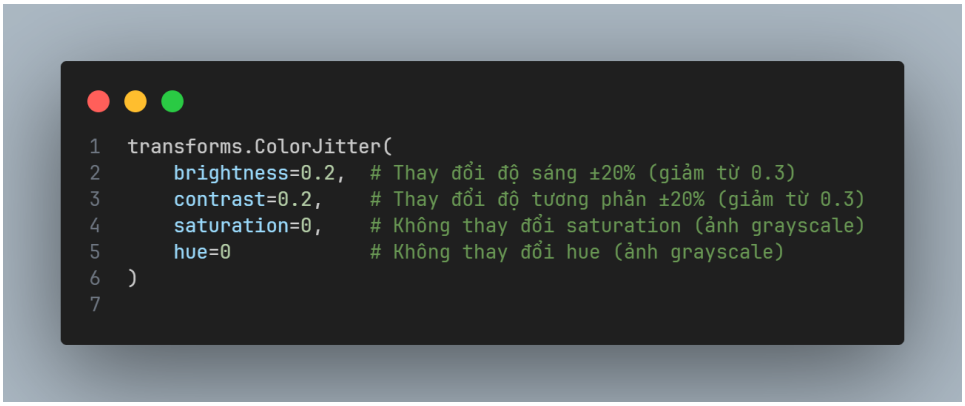
2.4. Color Jitter (Thay đổi độ sáng và tương phản)

Color Jitter là kỹ thuật thay đổi các thuộc tính màu sắc của ảnh, bao gồm brightness (độ sáng), contrast (độ tương phản), saturation (độ bão hòa màu), và hue (sắc thái màu). Trong dự án này, chỉ sử dụng brightness và contrast, trong khi saturation và hue được đặt ở 0 vì dữ liệu là grayscale và không có thông tin màu sắc.

Brightness control mô phỏng các điều kiện ánh sáng khác nhau trong thực tế. Khi chụp ảnh hoặc scan tài liệu, điều kiện ánh sáng có thể thay đổi đáng kể, từ ánh sáng mạnh tạo ra ảnh sáng đến ánh sáng yếu tạo ra ảnh tối. Mô hình cần học được cách nhận diện đối tượng bất kể độ sáng của ảnh, dựa trên hình dạng và cấu trúc thay vì chỉ dựa vào cường độ pixel tuyệt đối.

Contrast control mô phỏng các điều kiện scan và chất lượng ảnh khác nhau. Một ảnh được scan với chất lượng cao sẽ có độ tương phản tốt, với sự khác biệt rõ ràng giữa foreground và background. Ngược lại, một ảnh được scan với chất lượng thấp hoặc một ảnh chụp trong điều kiện ánh sáng kém sẽ có độ tương phản thấp, với sự khác biệt mờ nhạt hơn. Mô hình cần học được cách nhận diện đối tượng trong cả hai trường hợp.

Code minh họa:



```
transforms.ColorJitter(
    brightness=0.2,  # Thay đổi độ sáng ±20% (giảm từ 0.3)
    contrast=0.2,    # Thay đổi độ tương phản ±20% (giảm từ 0.3)
    saturation=0,    # Không thay đổi saturation (ảnh grayscale)
    hue=0            # Không thay đổi hue (ảnh grayscale)
)

```

Giải thích code:

- brightness=0.2: Thay đổi độ sáng trong khoảng ±20%
    - Giá trị pixel sẽ được nhân với một hệ số ngẫu nhiên trong khoảng [0.8, 1.2]
    - Ví dụ: pixel có giá trị 128 có thể trở thành 102 (tối hơn) hoặc 154 (sáng hơn)
- contrast=0.2 : Thay đổi độ tương phản trong khoảng ±20%
    - Contrast được điều chỉnh bằng cách thay đổi khoảng cách giữa các giá trị pixel
    - Giá trị cao hơn → tương phản mạnh hơn, giá trị thấp hơn → tương phản yếu hơn
- saturation=0 và hue=0 : Không áp dụng vì ảnh grayscale không có thông tin màu sắc

Lý do điều chỉnh:

- Ban đầu: brightness=0.3, contrast=0.3
- Vấn đề: Làm mất contrast quan trọng, đặc biệt với các shapes có edges mỏng
- Giải pháp: Giảm xuống 0.2 để vừa tăng diversity vừa giữ được contrast cần thiết

Tại sao không dùng Saturation/Hue:

- Digits/shapes là grayscale → không cần thay đổi màu sắc
- Chỉ cần brightness và contrast để mô phỏng điều kiện ánh sáng và scan quality

2.5. Resize và Grayscale Conversion

Resize là một bước quan trọng trong pipeline xử lý ảnh, đảm bảo tất cả ảnh đầu vào có cùng kích thước trước khi đưa vào mô hình. Trong dự án này, tất cả ảnh được resize về kích thước 128x128 pixel. Quyết định này xuất phát từ quá trình thử nghiệm và tối ưu hóa. Ban đầu, kích thước được đặt ở 64x64 pixel, với mục tiêu giảm tải tính toán và tăng tốc độ huấn luyện. Tuy nhiên, sau khi phân tích kết quả, nhóm phát hiện ra rằng ở độ phân giải thấp này, việc phân biệt giữa các hình đa giác có nhiều cạnh trở nên rất khó khăn.

Code minh họa:

```
transforms.Resize((128, 128))  # Resize về kích thước 128x128
transforms.Grayscale(num_output_channels=3)  # Chuyển grayscale → RGB

```

Giải thích code:

- **Resize**((128, 128)): Resize ảnh về kích thước 128×128 pixels
    - Tất cả ảnh đầu vào sẽ có cùng kích thước, bất kể kích thước gốc
    - Sử dụng bilinear interpolation mặc định để resize
- **Grayscale**(num_output_channels=3): Chuyển đổi ảnh grayscale (1 channel) sang RGB (3 channels)
    - num_output_channels=3
    - tạo ra ảnh RGB với 3 channels
    - Giá trị grayscale được copy vào cả 3 channels (R=G=B)
    - Kết quả là ảnh RGB nhưng thực chất vẫn là grayscale

Lý do chọn 128x128:

- Tăng từ 64x64: Để phân biệt tốt hơn các shapes có nhiều cạnh (Nonagon 9 cạnh, Octagon 8 cạnh)
- Không quá lớn: 128x128 đủ để nhận diện, không tốn quá nhiều memory và thời gian training
- EfficientNet-B0: Input size mặc định 224x224, nhưng 128x128 vẫn hoạt động tốt và nhanh hơn

Tại sao convert grayscale → RGB:

- EfficientNet-B0 pretrained trên ImageNet (RGB 3 channels)
- Input phải có 3 channels để sử dụng pretrained weights
- Cách làm: Copy grayscale vào 3 channels (R=G=B)
- Lợi ích: Tận dụng được pretrained weights từ ImageNet

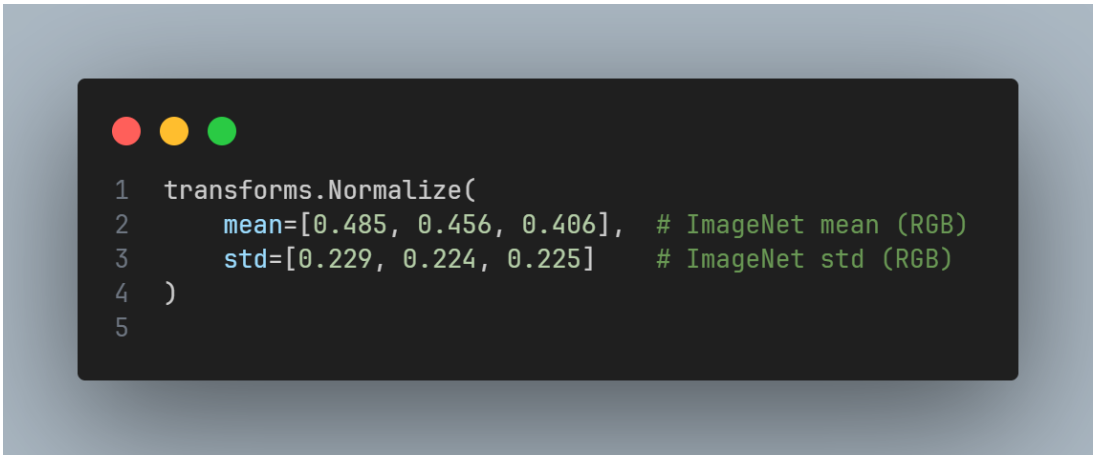
Vấn đề với 64x64:

- Nonagon (9 cạnh) và Circle có thể trông rất giống nhau ở độ phân giải thấp
- Các cạnh của Nonagon bị làm mờ do hiện tượng aliasing khi resize
- Octagon (8 cạnh) cũng có thể bị nhầm với Circle
- Tăng lên 128x128 giúp bảo toàn thông tin chi tiết về các cạnh và góc

2.6. Normalization (Chuẩn hóa)

Normalization là bước chuẩn hóa giá trị pixel của ảnh về một phân phối chuẩn, thường là phân phối chuẩn với mean bằng 0 và standard deviation bằng 1. Trong dự án này, normalization được thực hiện với mean và std của ImageNet, cụ thể là mean=[0.485, 0.456, 0.406] và std=[0.229, 0.224, 0.225] cho ba channels RGB. Đây là các giá trị được tính toán từ hàng triệu ảnh trong ImageNet và đã trở thành chuẩn cho các mô hình pre-trained trên ImageNet.

Code minh họa:



```
transforms.Normalize(
    mean=[0.485, 0.456, 0.406],  # ImageNet mean (RGB)
    std=[0.229, 0.224, 0.225]    # ImageNet std (RGB)
)

```

Giải thích code:

- mean=[0.485, 0.456, 0.406]: Giá trị trung bình của ImageNet cho 3 channels RGB
    - Channel R: 0.485
    - Channel G: 0.456
    - Channel B: 0.406

- std=[0.229, 0.224, 0.225]: Độ lệch chuẩn của ImageNet cho 3 channels RGB
    - Channel R: 0.229
    - Channel G: 0.224
    - Channel B: 0.225

Công thức normalization:

normalized_pixel = (pixel / 255.0 - mean) / std

Ví dụ tính toán:

- Pixel gốc: 128 (trong khoảng 0-255)
- Sau ToTensor: 0.5 (trong khoảng 0-1, vì 128/255 ≈ 0.5)
- Sau Normalize với mean=0.485, std=0.229:
    - (0.5 - 0.485) / 0.229 ≈ 0.065

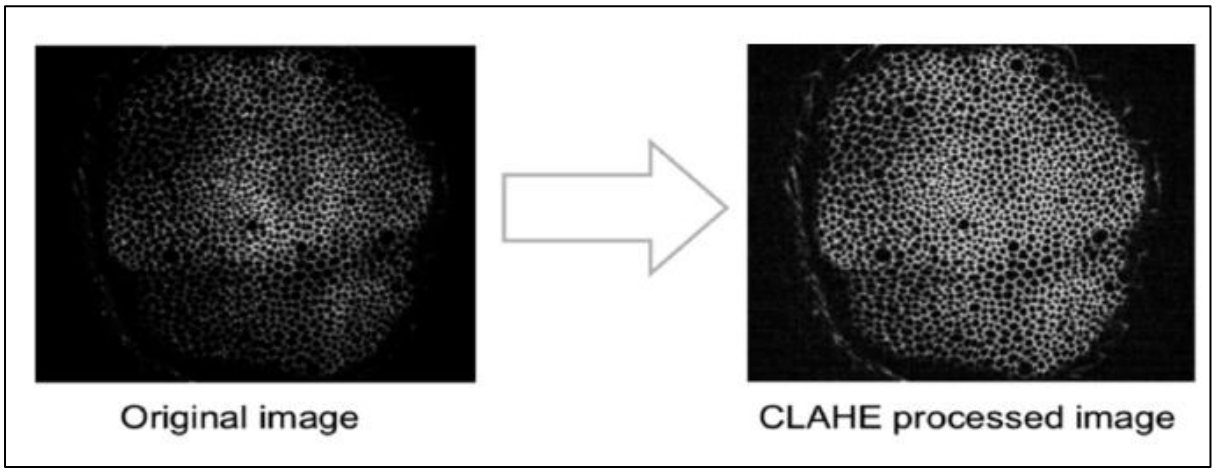
Tại sao normalize:

- Chuẩn hóa pixel values về range khoảng [-1, 1]
- Model pretrained đã quen với distribution này từ ImageNet
- Giúp training ổn định hơn, giảm vấn đề gradient vanishing/exploding
- Khởi tạo tham số tốt hơn khi fine-tuning

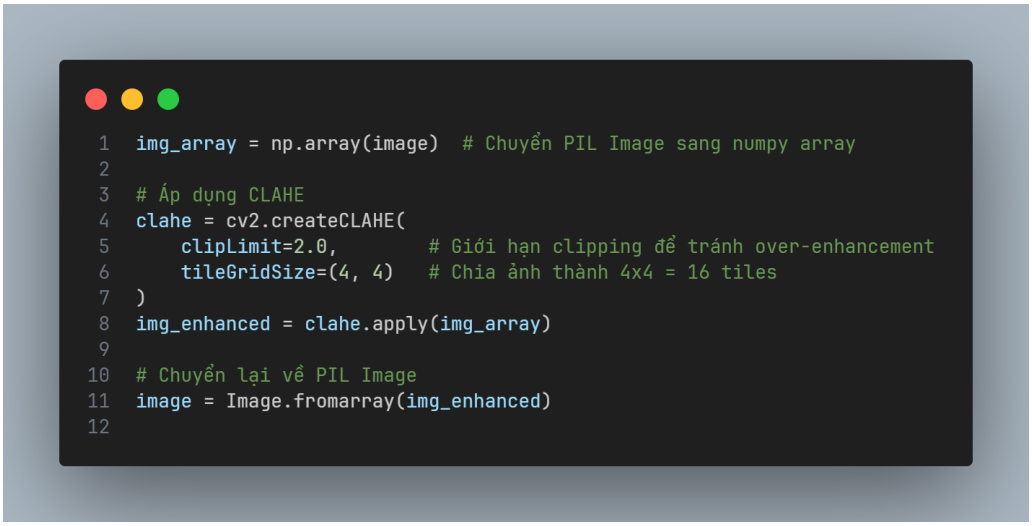
3. Tiền xử lý dữ liệu (Preprocessing)

3.1. CLAHE (Contrast Limited Adaptive Histogram Equalization)

CLAHE là một kỹ thuật nâng cao để tăng cường độ tương phản của ảnh, đặc biệt quan trọng với ảnh thực tế có điều kiện ánh sáng không đều. Khác với histogram equalization toàn cục, CLAHE chia ảnh thành các tiles nhỏ và áp dụng histogram equalization cho mỗi tile riêng biệt, sau đó interpolate giữa các tiles để tạo ra ảnh mượt mà không có đường biên rõ ràng giữa các tiles.



Code minh họa:



```python
img_array = np.array(image)  # Chuyển PIL Image sang numpy array

# Áp dụng CLAHE
clahe = cv2.createCLAHE(
    clipLimit=2.0,        # Giới hạn clipping để tránh over-enhancement
    tileGridSize=(4, 4)   # Chia ảnh thành 4x4 = 16 tiles
)
img_enhanced = clahe.apply(img_array)

# Chuyển lại về PIL Image
image = Image.fromarray(img_enhanced)

```

Giải thích code:

- cv2.createCLAHE(): Tạo CLAHE object với các tham số
  - clipLimit=2.0
  - Giới hạn clipping histogram
    - Khi histogram của một tile tập trung quá nhiều ở một giá trị, clipLimit giới hạn độ dốc
    - Giá trị 2.0 là một cân bằng tốt giữa tăng contrast và tránh over-enhancement
  - tileGridSize=(4, 4)
  - Chia ảnh thành 16 tiles (4 hàng × 4 cột)
    - Mỗi tile được xử lý độc lập với histogram equalization riêng
    - Sau đó các tiles được interpolate để tạo ảnh mượt mà
- clahe.apply(img_array): Áp dụng CLAHE lên ảnh
  - Input: numpy array (grayscale)
  - Output: numpy array đã được tăng cường contrast

Cách hoạt động:

1. Chia ảnh thành các tiles 4x4 (16 tiles)
2. Áp dụng histogram equalization cho mỗi tile riêng biệt
3. Clip histogram theo clipLimit để tránh over-enhancement
4. Interpolate giữa các tiles để tạo ảnh mượt mà

Lợi ích:

- Tăng contrast cục bộ (local contrast) thay vì toàn cục
- Giữ được chi tiết trong cả các vùng tối và sáng
- Đặc biệt hữu ích với ảnh scan hoặc ảnh có ánh sáng không đều

Áp dụng:

- Được thực hiện trong UnifiedDataset.__getitem__() trước khi apply transforms
- Chỉ áp dụng cho training và validation, không áp dụng cho inference

3.2. Preprocessing trong Detection

Preprocessing trong detection có mục tiêu khác với preprocessing trong classification. Trong detection, mục tiêu là tách foreground (các đối tượng) khỏi background một cách rõ ràng, để có thể tìm được contours và bounding boxes chính xác. Quá trình này bao gồm nhiều bước, mỗi bước có vai trò cụ thể.

Code minh họa (từ detect_objects.py):

```python
# 1. Gaussian Blur
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# 2. Adaptive Thresholding
binary = cv2.adaptiveThreshold(
    blurred, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV,
    11, 2
)

# 3. Morphological Operations
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
morph = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

```

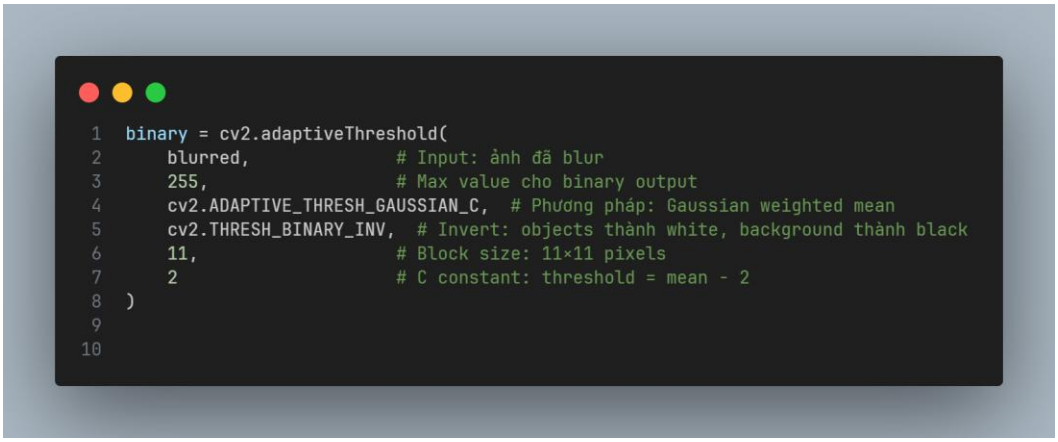
3.2.1. Gaussian Blur

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

- (5, 5): Kernel size 5×5 pixels
    - Kernel lớn hơn → làm mịn mạnh hơn nhưng mất chi tiết
    - Kernel nhỏ hơn → giữ chi tiết nhưng ít loại bỏ noise

    - 5×5 là cân bằng tốt
- 0: Standard deviation tự động tính từ kernel size
- Mục đích: Loại bỏ noise nhỏ, làm mịn ảnh trước khi thresholding
- Lợi ích: Giảm false positives từ noise, cải thiện chất lượng thresholding

3.2.2. Adaptive Thresholding



```
binary = cv2.adaptiveThreshold(
    blurred,              # Input: ảnh đã blur
    255,                  # Max value cho binary output
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  # Phương pháp: Gaussian weighted mean
    cv2.THRESH_BINARY_INV,  # Invert: objects thành white, background thành black
    11,                   # Block size: 11×11 pixels
    2                     # C constant: threshold = mean - 2
)
```

- ADAPTIVE_THRESH_GAUSSIAN_C: Sử dụng trọng số Gaussian để tính mean
    - Threshold cho mỗi pixel = weighted_mean(block) - C
    - Trọng số Gaussian cho kết quả mượt mà hơn mean đơn giản
- THRESH_BINARY_INV: Invert kết quả
    - Objects (thường tối) → white (255)
    - Background (thường sáng) → black (0)
    - Tạo điều kiện thuận lợi cho việc tìm contours
- Block size 11×11 pixels
    - Threshold được tính riêng cho mỗi block 11×11
    - Block lớn hơn → ít nhạy với local variations
    - Block nhỏ hơn → nhạy với noise
- C constant
    - Threshold = mean(block) - 2
    - Giá trị lớn hơn → threshold thấp hơn → nhiều pixels được coi là foreground
    - Giá trị nhỏ hơn → threshold cao hơn → ít pixels được coi là foreground

3.2.3. Morphological Operations

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
morph = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
```

- **getStructuringElement**(MORPH_RECT, (3, 3)): Tạo kernel hình chữ nhật 3×3
    - MORPH_RECT: Hình chữ nhật (các hình khác: MORPH_ELLIPSE, MORPH_CROSS)
    - 3×3: Kích thước kernel, đủ để đóng lỗ hổng nhỏ nhưng không làm biến dạng đối tượng
- **morphologyEx**(..., cv2.MORPH_CLOSE, ...): Áp dụng morphological closing
    - MORPH_CLOSE = Dilation → Erosion
    - Dilation: Mở rộng vùng white (đóng lỗ hổng)
    - Erosion: Co lại vùng white (khôi phục kích thước gần gốc)
- Mục đích: Đóng các lỗ hổng nhỏ, nối các phần bị đứt của đối tượng
- Lợi ích: Làm sạch binary image, cải thiện chất lượng contour detection

4. Normalization, Batching & Caching

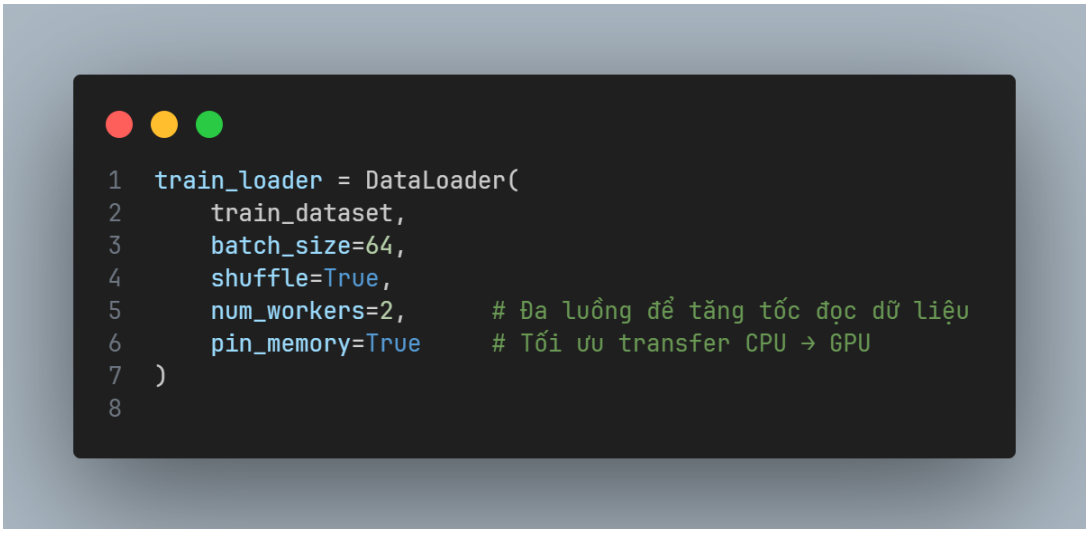
4.1. Normalization với ImageNet stats

Normalization trong context của batching và caching đề cập đến việc chuẩn hóa dữ liệu ở cấp độ pipeline, đảm bảo rằng tất cả ảnh được xử lý một cách nhất quán trước khi đưa vào mô hình. Điều này bao gồm việc áp dụng các giá trị mean và std của ImageNet như đã đề cập ở trên, nhưng cũng bao gồm các khía cạnh khác như đảm bảo tất cả ảnh có cùng kích thước, cùng số lượng channels, và cùng kiểu dữ liệu.

Việc normalization đảm bảo input distribution nhất quán với pretrained model là rất quan trọng cho hiệu quả của transfer learning. Khi một mô hình đã được huấn luyện trên một phân phối dữ liệu cụ thể, việc thay đổi phân phối đầu vào có thể làm giảm đáng kể hiệu năng. Normalization giúp đảm bảo rằng dữ liệu mới có phân phối tương tự như dữ liệu mà mô hình đã được huấn luyện, cho phép tận dụng tối đa các trọng số đã học.

4.2. DataLoader configuration (num_workers, pin_memory, prefetch)

Code minh họa:



```python
train_loader = DataLoader(
    train_dataset,
    batch_size=64,
    shuffle=True,
    num_workers=2,      # Đa luồng để tăng tốc đọc dữ liệu
    pin_memory=True     # Tối ưu transfer CPU → GPU
)

```

Giải thích code:

- batch_size=64: Số lượng ảnh trong mỗi batch
  - Batch lớn hơn → training ổn định hơn nhưng tốn memory
  - Batch nhỏ hơn → tiết kiệm memory nhưng training có thể không ổn định
  - 64 là cân bằng tốt cho GPU RTX 4050
- shuffle=True: Trộn ngẫu nhiên dữ liệu trong mỗi epoch
  - Giúp model không học được thứ tự của dữ liệu
  - Cải thiện khả năng tổng quát hóa
- num_workers=2: Sử dụng 2 worker threads để đọc và preprocess dữ liệu song song
  - Mỗi worker đọc và preprocess một batch độc lập
  - Lợi ích: Tăng tốc độ đọc dữ liệu, giảm thời gian chờ GPU idle
  - Trade-off: Tăng memory usage (mỗi worker giữ một batch trong memory)
- pin_memory=True: Pin memory để tăng tốc transfer từ CPU sang GPU
  - Pinned memory là loại memory đặc biệt mà GPU có thể truy cập trực tiếp
  - Lợi ích: GPU có thể truy cập dữ liệu nhanh hơn, giảm latency
  - Yêu cầu: Cần đủ RAM để pin memory

Prefetch mechanism:

- Mặc định prefetch_factor=2: Mỗi worker prefetch 2 batches trước
- Tổng số batches trong queue: num_workers × prefetch_factor = 2 × 2 = 4 batches

- Lợi ích: GPU luôn có dữ liệu sẵn sàng, không phải chờ đọc từ disk

4.3. Augmentation "nóng" (On-the-fly)

Augmentation "nóng" là một chiến lược quan trọng trong dự án. Thay vì tạo ra tất cả các biến thể của ảnh trước và lưu vào disk, augmentation được thực hiện trong Dataset.__getitem__()mỗi khi lấy một sample. Điều này có nghĩa là mỗi epoch, mỗi ảnh được augment khác nhau do random seed khác nhau.
Lợi ích:

- Tăng diversity tối đa: Mỗi epoch model thấy dữ liệu khác nhau
- Tiết kiệm disk space: Không cần lưu tất cả các biến thể
- Linh hoạt: Dễ dàng thay đổi augmentation strategy

Trade-off:

- Tốn CPU để augment tại runtime
- Được bù đắp bởi num_workers và prefetch, cho phép augmentation chạy song song với training

5. Chiến lược Augmentation cho Training và Validation

5.1. Training Augmentation

Code minh họa:

```python
train_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.Grayscale(num_output_channels=3),
    transforms.RandomRotation(30),
    transforms.RandomAffine(degrees=0, translate=(0.15, 0.15)),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.3),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

```

Giải thích:

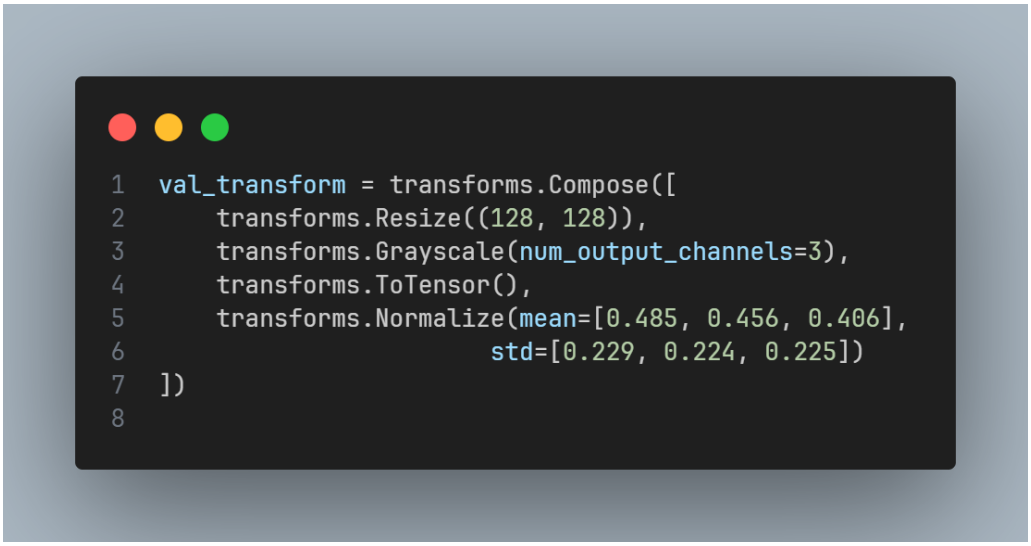
- Pipeline đầy đủ với TẤT CẢ augmentations
- Mỗi epoch, mỗi ảnh được augment khác nhau (random seed)
- Tăng diversity tối đa để model học được tính robust

Thứ tự áp dụng:

1. Resize về 128×128 (chuẩn hóa kích thước)
2. Grayscale → RGB (chuẩn hóa channels)
3. RandomRotation (biến đổi hình học)
4. RandomAffine (dịch chuyển)
5. RandomPerspective (phối cảnh)
6. ColorJitter (độ sáng/tương phản)
7. ToTensor (chuyển sang tensor)
8. Normalize (chuẩn hóa giá trị)

5.2. Validation Augmentation

Code minh họa:



```
val_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

```

Giải thích:

- Chỉ resize và normalize
- Không augment để đánh giá chính xác performance
- Đảm bảo validation metrics phản ánh đúng khả năng của model

Lý do:

- Validation set được sử dụng để đánh giá hiệu năng thực tế của model
- Áp dụng augmentation sẽ làm thay đổi dữ liệu, khiến metrics không phản ánh đúng khả năng
- Validation được giữ nguyên như dữ liệu gốc để đánh giá chính xác và công bằng

5.3. Test Time Augmentation (TTA)

Test Time Augmentation là một kỹ thuật nâng cao được áp dụng trong quá trình inference để tăng độ chính xác. Thay vì chỉ dự đoán một lần trên ảnh gốc,

TTA tạo ra nhiều biến thể của ảnh, dự đoán trên mỗi biến thể, và sau đó average các predictions để có kết quả cuối cùng.

Code minh họa:

```python
if predicted_class ⩾ 10:  # Shape
    # Original prediction
    probs_original = model(crop)
    all_probs = [probs_original]

    # Rotate +5°, -5°, +10°, -10°
    for angle in [5, -5, 10, -10]:
        rotated = rotate(crop, angle)
        probs_rot = model(rotated)
        all_probs.append(probs_rot)

    # Average probabilities
    final_probs = torch.stack(all_probs).mean(dim=0)
    predicted_class = final_probs.argmax().item()

```

Giải thích code:

- if predicted_class >= 10: Chỉ áp dụng TTA cho shapes (class_id >= 10)
    - Digits (0-9) đã đủ chính xác, không cần TTA
    - Shapes, đặc biệt các hình đa giác, có thể được cải thiện với TTA
- **rotate**(crop, angle): Xoay ảnh với các góc ±5° và ±10°
    - Góc nhỏ để không làm mất đặc trưng
    - Nhiều góc để tăng diversity
- torch.stack(all_probs).mean(dim=0): Average các xác suất
    - Stack tất cả predictions thành tensor
    - Tính mean theo dimension 0 (across augmentations)
    - Kết quả là xác suất trung bình từ tất cả các biến thể

Kết quả:

- Accuracy tăng ~0.5-1% cho shapes
- Trade-off: Inference chậm hơn 3-5x (phải chạy model nhiều lần)

IV. Kiến trúc mô hình

1. Backbone: EfficientNet-B0

1.1. Kiến trúc mạng

Trong đề tài, EfficientNet-B0 được dùng như bộ trích xuất đặc trưng & phân loại khá gọn, chạy tốt trên GPU/laptop phổ thông. Ở góc độ môn Xử lý ảnh, ta quan tâm chủ yếu tới những điểm sau:

- Mạng nhận ảnh đầu vào 3 kênh, kích thước cố định; project chọn 128×128 thay vì 224×224 để vừa đủ phân giải cho đa giác nhiều cạnh mà vẫn nhanh.
- Các tầng đầu của EfficientNet-B0 học những đặc trưng: biên, cạnh, góc, texture… vốn tương tự các bộ lọc Sobel, Laplace, Gabor nhưng được học tự động.
- Mạng được pretrained trên ImageNet, tức là các filter phát hiện biên/texture đã rất "giàu kinh nghiệm"; nhiệm vụ của pipeline xử lý ảnh là chuẩn hóa digits/shapes sao cho hợp với những filter này (CLAHE, resize, normalize).

Code minh họa – Load EfficientNet-B0:

```
1  from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights
2  model = efficientnet_b0(weights=EfficientNet_B0_Weights.IMAGENET1K_V1)
3
```

Đoạn code trên cho thấy phần kiến trúc sâu bên trong được dùng gần như "hộp đen".

1.2. Thay đổi classifier cho 19 lớp

Để EfficientNet-B0 phân loại 19 lớp (10 digits + 9 shapes), ta chỉ thay Linear cuối

```python
import torch.nn as nn
from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights
model = efficientnet_b0(weights=EfficientNet_B0_Weights.IMAGENET1K_V1)
num_features = model.classifier[1].in_features
model.classifier[1] = nn.Linear(num_features, 19)

```

Liên hệ với xử lý ảnh: vì backbone giữ nguyên pretrained weights, phần lớn cải thiện chất lượng nhận dạng đến từ cách ta xử lý ảnh đầu vào (tăng tương phản, khử nhiễu, chuẩn hóa kích thước/kênh màu, augmentation) hơn là "vọc sâu" kiến trúc mạng

2. Training Configuration & Strategy

2.1. Hyperparameters chi tiết

2.1.1. Loss Function (CrossEntropyLoss)

Loss Function được sử dụng là CrossEntropyLoss, một loss function tiêu chuẩn cho bài toán multi-class classification. CrossEntropyLoss kết hợp LogSoftmax và NLLLoss (Negative Log Likelihood Loss) trong một hàm duy nhất, giúp tối ưu hóa hiệu quả tính toán. Loss function này đo lường sự khác biệt giữa phân phối xác suất dự đoán và phân phối xác suất thực tế (one-hot encoding), và được thiết kế để penalize mạnh các dự đoán sai với confidence cao.

Code minh họa:

```python
import torch.nn as nn
criterion = nn.CrossEntropyLoss()

```

Giải thích:

- CrossEntropyLoss(): Loss function cho multi-class classification

- Input:
    - Predictions: Tensor shape (batch_size, num_classes) - logits (chưa qua softmax)
    - Targets: Tensor shape (batch_size) - class indices (0-18)
- Công thức:

  loss = -log(exp(x[class]) / sum(exp(x[i])))

    - x[class] là logit của class đúng
    - Softmax được tính tự động trong loss function
- Đặc điểm: Penalize mạnh các dự đoán sai với confidence cao
    - Nếu model dự đoán sai với confidence cao → loss lớn
    - Nếu model dự đoán sai với confidence thấp → loss nhỏ hơn

CrossEntropyLoss đặc biệt phù hợp với bài toán này vì nó xử lý tốt các trường hợp có nhiều classes và có thể tự động xử lý class imbalance thông qua class weights nếu cần. Trong dự án, không sử dụng class weights vì dataset đã được cân bằng thông qua sampling (sample_fraction=0.67 cho shapes để cân bằng với MNIST).

2.1.2. Optimizer (Adam)

Optimizer được sử dụng là Adam (Adaptive Moment Estimation), một optimizer phổ biến và hiệu quả trong deep learning. Adam kết hợp các ưu điểm của Momentum và RMSprop, sử dụng cả gradient history (momentum) và adaptive learning rate (RMSprop) để cập nhật tham số. Adam tự động điều chỉnh learning rate cho từng tham số dựa trên gradient history, giúp training ổn định và hội tụ nhanh hơn so với SGD truyền thống.

Code minh họa:

```
1  import torch.optim as optim
2  optimizer = optim.Adam(model.parameters(), lr=1e-4)
3
```

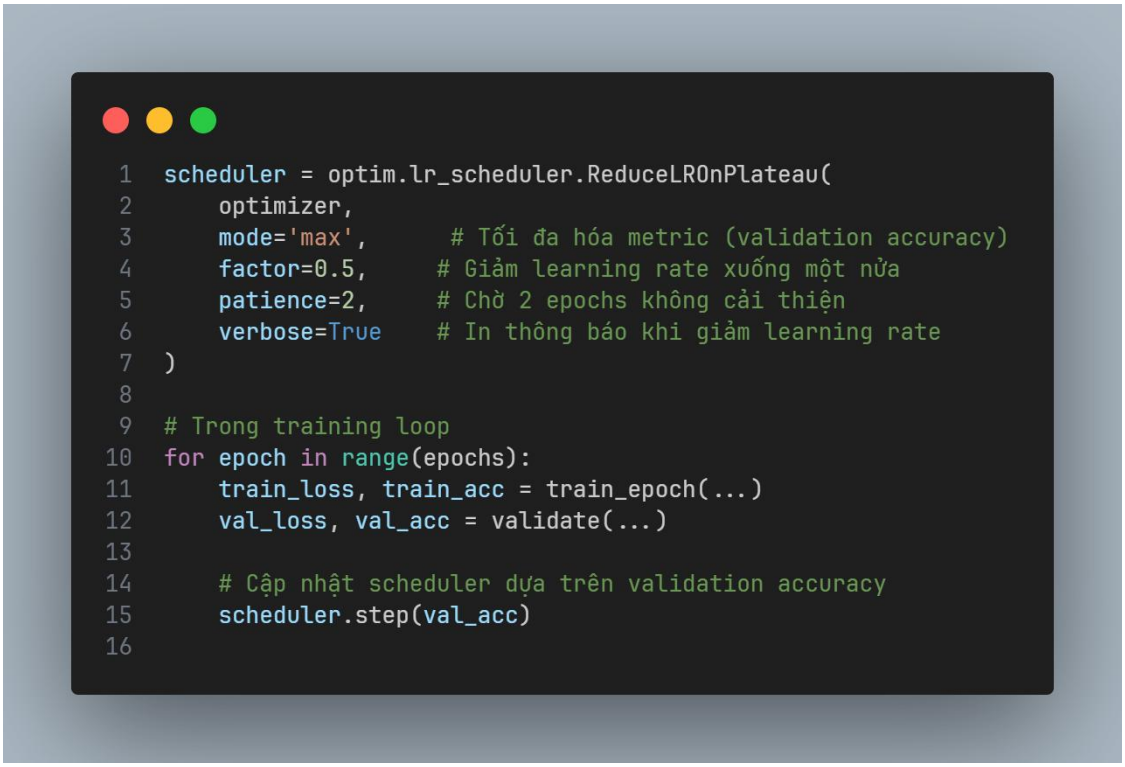
Giải thích:

- Adam(model.parameters(), lr=1e-4): Tạo Adam optimizer
    - model.parameters()
    - : Tất cả các tham số của model cần được optimize
    - lr=1e-4
    - : Learning rate ban đầu = 0.0001
- Adam algorithm:
    - Tính gradient của loss với respect to parameters
    - Cập nhật moving average của gradient (momentum)
    - Cập nhật moving average của squared gradient (RMSprop)
    - Điều chỉnh learning rate cho từng parameter dựa trên 2 moving averages
- Ưu điểm:
    - Tự động điều chỉnh learning rate cho từng parameter
    - Hội tụ nhanh hơn SGD
    - Training ổn định hơn

2.1.3. Learning Rate Scheduler (ReduceLROnPlateau)

Learning Rate Scheduler được sử dụng là ReduceLROnPlateau, một scheduler thông minh tự động giảm learning rate khi validation accuracy không cải thiện trong một số epochs nhất định. Cụ thể, scheduler được cấu hình với mode='max' (tối đa hóa validation accuracy), factor=0.5 (giảm learning rate xuống một nửa), và patience=2 (chờ 2 epochs không cải thiện trước khi giảm learning rate). Điều này giúp mô hình hội tụ tốt hơn bằng cách giảm learning rate khi gần đến optimum, cho phép fine-tuning chính xác hơn.

Code minh họa:



```python
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer,
    mode='max',      # Tối đa hóa metric (validation accuracy)
    factor=0.5,      # Giảm learning rate xuống một nửa
    patience=2,      # Chờ 2 epochs không cải thiện
    verbose=True     # In thông báo khi giảm learning rate
)

# Trong training loop
for epoch in range(epochs):
    train_loss, train_acc = train_epoch(...)
    val_loss, val_acc = validate(...)

    # Cập nhật scheduler dựa trên validation accuracy
    scheduler.step(val_acc)

```

Giải thích code:

- mode='max': Tối đa hóa metric (validation accuracy)
  - Scheduler sẽ giảm LR khi metric không tăng
  - Nếu dùng
  - mode='min' → giảm LR khi metric không giảm (cho loss)
- factor=0.5: Mỗi lần giảm, LR giảm xuống một nửa
  - LR ban đầu: 1e-4
  - Sau lần giảm 1: 5e-5
  - Sau lần giảm 2: 2.5e-5
- patience=2: Chờ 2 epochs không cải thiện trước khi giảm LR
  - Tránh giảm LR quá sớm khi model chỉ đang dao động
  - Nếu val_acc không tăng trong 2 epochs liên tiếp → giảm LR
- scheduler.**step**(val_acc): Cập nhật scheduler với validation accuracy
  - Scheduler so sánh val_acc hiện tại với best val_acc
  - Nếu không cải thiện trong patience epochs → giảm LR

2.2. Learning Rate Scheduler

2.2.1. Warmup

Trong dự án, không sử dụng warmup vì learning rate ban đầu đã được đặt ở mức nhỏ (1e-4), và với pretrained weights, không cần warmup để tránh gradient explosion. Warmup thường được sử dụng khi training từ đầu với learning rate lớn, hoặc khi sử dụng batch size rất lớn.

2.2.2. Cosine decay / Step decay

Cosine decay và Step decay là các scheduler dựa trên thời gian, giảm learning rate theo một lịch trình cố định. Trong dự án, không sử dụng các scheduler này vì ReduceLROnPlateau đã đủ để đảm bảo hội tụ tốt với transfer learning. Cosine decay có thể hữu ích cho training từ đầu, nhưng với transfer learning, ReduceLROnPlateau đã đủ.

2.2.3. Early Stopping

Early stopping không được implement trong code, nhưng có thể được thêm vào để tránh overfitting. Early stopping sẽ dừng training khi validation accuracy không cải thiện trong một số epochs nhất định, giúp tiết kiệm thời gian và tránh overfitting. Tuy nhiên, với 20 epochs và dataset đã được augment tốt, overfitting không phải là vấn đề lớn.

Code minh họa - Early Stopping

```python
best_val_acc = 0.0
patience = 5
patience_counter = 0

for epoch in range(epochs):
    train_loss, train_acc = train_epoch(...)
    val_loss, val_acc = validate(...)

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        patience_counter = 0
        # Save best model
        torch.save(model.state_dict(), 'best_model.pth')
    else:
        patience_counter += 1
        if patience_counter ⩾ patience:
            print(f"Early stopping at epoch {epoch+1}")
            break

```

3. Pipeline Huấn luyện

3.1. DataLoader configuration

DataLoader được cấu hình với batch_size=64, shuffle=True cho training set và shuffle=False cho validation set, num_workers=2, và pin_memory=True. Batch size 64 được chọn là một cân bằng tốt giữa memory usage và training stability. Batch size lớn hơn sẽ sử dụng nhiều memory hơn nhưng có thể cải thiện training stability và tốc độ, trong khi batch size nhỏ hơn sẽ sử dụng ít memory hơn nhưng có thể làm training không ổn định. Giá trị 64 được xác định thông qua thử nghiệm và phù hợp với GPU RTX 4050 được sử dụng trong dự án.

Code minh họa:

```python
from torch.utils.data import DataLoader

train_loader = DataLoader(
    train_dataset,
    batch_size=64,
    shuffle=True,         # Trộn ngẫu nhiên cho training
    num_workers=2,        # 2 worker threads
    pin_memory=True       # Tối ưu transfer CPU → GPU
)

val_loader = DataLoader(
    val_dataset,
    batch_size=64,
    shuffle=False,        # Không trộn cho validation
    num_workers=2,
    pin_memory=True
)

```

Giải thích:

- batch_size=64: Số lượng ảnh trong mỗi batch
- shuffle=True/False: Trộn ngẫu nhiên dữ liệu
    - Training: True để model không học được thứ tự
    - Validation: False để đảm bảo tính nhất quán
- num_workers=2: Số worker threads để đọc dữ liệu song song
- pin_memory=True: Pin memory để tăng tốc transfer CPU → GPU

3.2. Validation Loop

Validation loop được thực hiện sau mỗi epoch để đánh giá hiệu năng của mô hình trên tập validation. Validation được thực hiện với model.eval() và torch.no_grad() để tắt dropout và batch normalization training mode, và không tính gradient để tiết kiệm memory và tăng tốc độ. Validation accuracy được sử dụng để quyết định xem có nên lưu model tốt nhất hay không, và để điều chỉnh learning rate thông qua scheduler.

Code minh họa:

```python
def validate(model, loader, criterion, device):
    """Validate model on validation set."""
    model.eval()  # Set model to evaluation mode
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():  # Không tính gradient để tiết kiệm memory
        for images, labels in loader:
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Tính accuracy
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            running_loss += loss.item()

    accuracy = 100. * correct / total
    avg_loss = running_loss / len(loader)

    return avg_loss, accuracy
```

Giải thích code:

- model.eval(): Set model sang evaluation mode
    - Tắt dropout (tất cả neurons đều active)
    - BatchNorm sử dụng running statistics thay vì batch statistics
- torch.no_grad(): Context manager để tắt gradient computation
    - Tiết kiệm memory (không lưu gradients)
    - Tăng tốc độ (không tính backward pass)
- torch.max(outputs.**data**, 1): Tìm class có xác suất cao nhất
    - outputs: Shape (batch_size, num_classes) - logits

    - 1: Dimension để tìm max (theo classes)
    - Returns: (values, indices) - indices là predicted classes
- (predicted == labels).sum().item(): Đếm số dự đoán đúng
    - So sánh predicted với labels
    - Sum để đếm số True (đúng)
    - .item(): để chuyển từ tensor sang Python number

3.3. Checkpointing

Checkpointing là quá trình lưu trạng thái của mô hình tại các thời điểm quan trọng, đặc biệt là khi đạt được validation accuracy tốt nhất. Trong dự án, checkpoint được lưu mỗi khi validation accuracy vượt qua best_val_acc, đảm bảo rằng luôn có model tốt nhất được lưu lại.

Code minh họa:

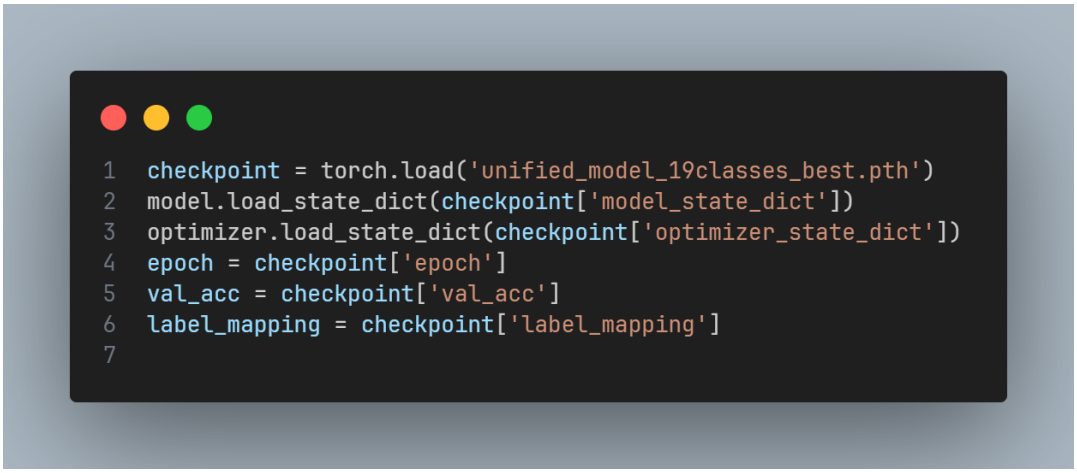
```python
best_val_acc = 0.0

for epoch in range(epochs):
    train_loss, train_acc = train_epoch(...)
    val_loss, val_acc = validate(...)

    # Lưu checkpoint nếu đạt accuracy tốt nhất
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'val_acc': val_acc,
            'label_mapping': id_to_label,
            'config': vars(args)
        }, 'unified_model_19classes_best.pth')
        print(f" Saved best model (Val Acc: {val_acc:.2f}%)")


```

Giải thích code:

- **model**.state_dict(): Lưu tất cả weights và biases của model
    - Dictionary mapping layer names → parameter tensors
    - Có thể load lại bằng
    - **model**.load_state_dict()
- optimizer.state_dict(): Lưu trạng thái của optimizer

    - Bao gồm momentum, learning rate, etc.
    - Cho phép resume training từ checkpoint
- label_mapping: Mapping giữa class ID và tên class
    - Quan trọng cho inference sau này
- config: Các hyperparameters được sử dụng
    - Giúp reproduce kết quả

Load checkpoint:



```
checkpoint = torch.load('unified_model_19classes_best.pth')
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
val_acc = checkpoint['val_acc']
label_mapping = checkpoint['label_mapping']

```

V. Dataset

1. Nguồn gốc & Mục đích

1.1. MNIST gốc

Tập MNIST được sử dụng làm nguồn dữ liệu chữ số cơ bản cho đề tài. Trong project, dữ liệu được lưu dưới dạng các ảnh **.png** trong thư mục mnist/train, kèm theo file train_label.csv chứa cặp (image_name, label)cho khoảng 60.000 ảnh train. Mỗi ảnh là một chữ số viết tay kích thước (28x28) pixel, ở dạng grayscale với nền đơn giản, ít nhiễu, mỗi ảnh chỉ chứa một đối tượng duy nhất.

MNIST có hai ưu điểm chính:

- Dữ liệu đã chuẩn hóa rất tốt: cùng kích thước, cùng kiểu ảnh, dễ áp dụng các phép xử lý như resize, normalize, chuyển kênh màu.
- Được cộng đồng sử dụng rộng rãi, giúp việc so sánh mô hình và tái lập thí nghiệm trở nên thuận tiện.

Tuy nhiên, từ góc nhìn xử lý ảnh đa đối tượng, MNIST gốc còn quá “sạch”: không có nhiều đối tượng trong cùng một cảnh, không có hình học phi chữ số, ít biến dạng về phối cảnh và nhiễu thực tế. Đây chính là lý do cần phải mở rộng và tái cấu trúc dữ liệu.

1.2. Mở rộng cho bài toán đa đối tượng

Để đưa bài toán về đúng tinh thần nhận diện nhiều đối tượng trong một ảnh, nhóm xây dựng thêm một bộ dữ liệu hình học synthetic trong thư mục Shapes_Classifier/dataset/output. Bộ này bao gồm 9 hình học cơ bản:

- Circle, Triangle, Square
- Pentagon, Hexagon, Heptagon, Octagon, Nonagon
- Star

Mỗi hình được vẽ trên nền trắng, nét thường là màu đen, kích thước vùng chứa dao động từ khoảng (64x64) tới (256x256) pixel, với góc quay, vị trí, độ dày nét, mức độ nhiễu được random hoá trong một khoảng cho phép.

Việc sinh hình synthetic như vậy cho phép:

- Chủ động kiểm soát các yếu tố quan trọng trong xử lý ảnh: độ dày nét, độ cong, số đỉnh, mức độ mờ/aliasing của cạnh.
- Tạo ra các trường hợp khó (đa giác nhiều cạnh, nét mỏng, quay góc lạ) để thử thách pipeline xử lý ảnh và mô hình phân loại.

Nhờ đó, khi phân tích lỗi (ví dụ Nonagon bị "tròn hóa" thành Circle do aliasing), ta có thể quay lại điều chỉnh cả cách sinh dữ liệu lẫn các bước tiền xử lý ảnh một cách có hệ thống.

1.3. Ứng dụng trong giáo dục

Hai nguồn dữ liệu (digits từ MNIST và shapes synthetic) được kết hợp để xây dựng các scene tổng hợp: trên một canvas (ví dụ (800x600)), các chữ số và hình học được đặt xen kẽ, tạo thành những "bài tập trực quan" tương tự phiếu bài tập hoặc đề thi cho học sinh. Hàm generate_synthetic_scene trong pipeline.py cho phép sinh scene on-the-fly, với số lượng và loại đối tượng tùy chọn. Điều này rất phù hợp với các ứng dụng:

- Sinh tự động đề luyện tập (ví dụ: "đếm số tam giác", "tính tổng các số xuất hiện").
- Nhận diện và chấm bài từ ảnh chụp phiếu giấy.
- Trò chơi học tập tương tác, nơi hệ thống phải đọc cả chữ số và hình học trong cùng một ảnh.

Từ đó, tập dữ liệu không chỉ đơn thuần là dữ liệu huấn luyện mô hình mà còn là nền tảng xử lý ảnh cho các ứng dụng EdTech sau này.

2. Quy trình xây dựng

2.1. Tách ảnh chữ số MNIST

Quy trình bắt đầu từ file train_label.csv, trong đó mỗi dòng chứa image_name và label tương ứng. Pipeline:

- Đọc từng dòng trong train_label.csv.
- Ánh xạ image_name thành đường dẫn đầy đủ trong mnist/train.
- Lưu thông tin vào một DataFrame/tập cấu trúc gồm: đường dẫn ảnh, nhãn 0–9, nguồn "mnist".

Bước này không thay đổi nội dung ảnh, giúp giữ nguyên tính chuẩn hóa ban đầu của MNIST. Mọi thao tác xử lý ảnh (resize, chuyển kênh màu, normalize…) được tách riêng, thực hiện trong lớp dataset và các transform, đúng với tư duy "dataset sạch, pipeline xử lý ảnh rõ ràng".

2.2. Sinh hình học cơ bản

Các hình học được tạo bằng các script trong Shapes_Classifier/, sử dụng các primitive drawing (vẽ đường, vẽ đa giác, vẽ hình tròn). Với mỗi mẫu hình, script:

- Chọn ngẫu nhiên loại hình (ví dụ: Nonagon).
- Random hóa kích thước cạnh/bán kính, góc quay, độ dày nét.
- Thêm nhiễu nhẹ (Gaussian hoặc salt & pepper) để mô phỏng scan/chụp thực tế.

Tên file thường mang thông tin loại hình và một định danh ngẫu nhiên, ví dụ

Nonagon_01234.png , giúp việc ánh xạ sang nhãn chữ ("Nonagon") và sau đó sang id (13) trở nên đơn giản.

Việc sinh dữ liệu theo cách này giúp:

- Dễ dàng tăng/giảm số mẫu cho từng lớp shape khi cần cân bằng dữ liệu.
- Tạo các shape có độ "khó" khác nhau (nhiều cạnh, nét nhỏ, góc quay lạ) để phục vụ phân tích và cải thiện pipeline xử lý ảnh.

2.3. Ghép và tạo scene

Từ hai nguồn primitive (digits và shapes), hàm generate_synthetic_scene dựng các scene phức tạp hơn:

- Tạo một canvas trắng (ví dụ (800x600)).
- Chọn ngẫu nhiên số lượng digits và shapes.

- Đặt từng đối tượng lên canvas ở vị trí random, với điều kiện thỏa check_overlap để tránh chồng lấn quá mức.

Mỗi khi đặt một đối tượng, hàm đồng thời ghi lại thông tin bounding box (x, y, w, h) và nhãn tương ứng. Kết quả là, ngoài ảnh tổng hợp, ta còn có danh sách ground-truth bounding boxes cho từng scene.

Từ góc nhìn xử lý ảnh, đây là dữ liệu quý giá để đánh giá:

- Traditional detector dựa trên Gaussian blur + adaptive threshold + morphology + contour.
- Hybrid detector kết hợp CRAFT (cho digits) và CV truyền thống (cho shapes).

Nhờ có ground-truth đầy đủ, ta có thể định lượng chính xác precision/recall của các bước xử lý ảnh.

2.4. Gán nhãn và annotation

Đề tài sử dụng hai kiểu nhãn song song:

- Nhãn classifier: mỗi ảnh crop (digit hoặc shape) từ MNIST/shapes được gán một id từ 0–18 (chi tiết ở mục 9.3), dùng cho huấn luyện EfficientNet-B0.
- Nhãn detection: mỗi scene synthetic được gán danh sách bounding boxes kèm **class**_id dùng để đánh giá khả năng tìm vùng đối tượng của pipeline xử lý ảnh.

File label_mapping.json là cầu nối giữa hai thế giới này, ánh xạ:

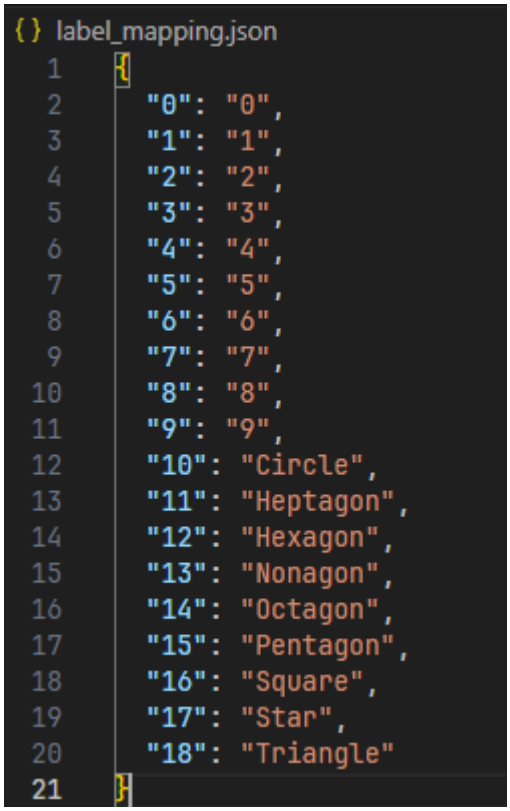
- **class**_id → label_string (ví dụ: 13 → "Nonagon").
- Cho phép inference in ra tên lớp trực quan trong lúc vẽ annotate trên ảnh.

3. Cấu trúc dữ liệu

3.1. Label mapping (0-18)

Không gian nhãn được chuẩn hóa thành 19 lớp:

- 10 lớp đầu: digits 0–9.
- 9 lớp sau: 9 loại hình học.



3.2. Bảng mapping chi tiết

Ngoài file JSON, trong báo cáo có thể trình bày bảng tóm tắt:

- 0–9: các chữ số viết tay tương ứng.
- 10: Circle – hình tròn.
- 11–15: các đa giác 7, 6, 9, 8, 5 cạnh (Heptagon, Hexagon, Nonagon, Octagon, Pentagon).
- 16: Square – hình vuông.
- 17: Star – ngôi sao.
- 18: Triangle – tam giác.

Việc liệt kê rõ ràng giúp người đọc, đặc biệt là giảng viên môn Xử lý ảnh, dễ đối chiếu giữa ý nghĩa hình học và id nhãn khi xem kết quả annotate hoặc confusion matrix.

3.3. Format dữ liệu

Về mặt lưu trữ:

- Ảnh primitive (digits/shapes) được lưu dưới dạng **.png** grayscale.
- Khi sử dụng cho classifier, ảnh được load, resize về 128×128, chuyển sang 3 kênh (R=G=B) và chuẩn hóa (mục 9.4.1).
- Annotation cho detection được lưu ở dạng JSON/CSV, mỗi entry chứa:
    - bbox: toạ độ và kích thước bounding box.

- label hoặc class_id. (tùy trường hợp)
- confidence – xác suất dự đoán từ mô hình.

Cách tổ chức này giúp việc trực quan hóa kết quả detection (vẽ khung + nhãn) và phân tích lỗi trở nên dễ dàng, đặc biệt khi ta muốn đánh giá từng bước của pipeline xử lý ảnh.

4. Tiền xử lý & Augmentation (trong dataset)

4.1. Chuẩn hóa ảnh (Normalization)

Chuyển đổi kênh màu (Grayscale → RGB)

Ảnh từ MNIST và shapes đều là grayscale 1 kênh. Trong khi đó, EfficientNet-B0 pretrained trên ImageNet yêu cầu input 3 kênh (RGB). Do đó, pipeline sử dụng bước:

- Chuyển grayscale → 3 kênh bằng cách nhân bản kênh xám sang R, G, B.

Việc này giữ nguyên thông tin cường độ sáng nhưng giúp tương thích hoàn toàn với backbone đã pretrained, cho phép tận dụng trực tiếp các filter phát hiện cạnh, texture ở tầng thấp.

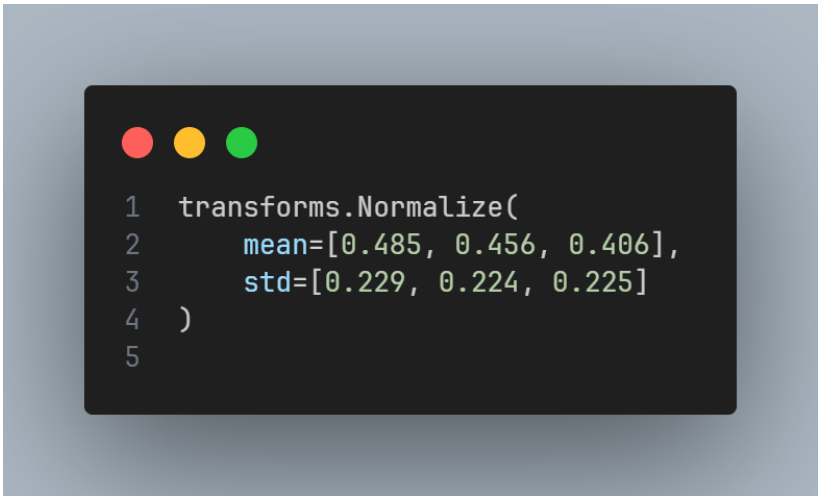
Tăng độ phân giải không gian

Đối với classifier, tất cả ảnh crop đều được resize lên 128×128. Quyết định này mang màu sắc xử lý ảnh khá rõ:

- Nếu để kích thước quá nhỏ (ví dụ 28×28 hoặc 64×64), các đa giác nhiều cạnh (Octagon, Nonagon) sẽ bị mờ hoặc "tròn hóa" do aliasing, gây nhầm lẫn với Circle.
- Kích thước 128×128 lớn hơn đủ để giữ lại biên và đỉnh của đa giác, giúp cả mắt người lẫn mô hình dễ phân biệt hơn.

Normalization với ImageNet stats

Sau khi resize và chuyển kênh, ảnh được chuẩn hóa theo thống kê chuẩn của ImageNet:



Việc này đưa phân phối pixel của ảnh digits/shapes về cùng "vùng giá trị" mà EfficientNet-B0 đã được huấn luyện, giúp cho:

- Các filter ở tầng đầu phát huy đúng chức năng (phát hiện cạnh, texture) thay vì phải thích nghi lại từ đầu.
- Quá trình huấn luyện ổn định hơn, gradient ít bị bão hòa hoặc bùng nổ.

4.2. Cân bằng dữ liệu (Dataset Balancing)

Mất cân bằng dữ liệu

Ban đầu, số lượng ảnh giữa các lớp (đặc biệt giữa digits và shapes) không đồng đều. Nếu huấn luyện trực tiếp:

- Mô hình có xu hướng thiên vị nhóm lớp xuất hiện nhiều (thường là digits).
- Các lớp hiếm như Nonagon có thể bị học kém, dễ bị nhầm với Circle hoặc các đa giác khác.

Giải pháp lấy mẫu (sample_fraction=0.67)

Để giảm mất cân bằng, lớp UnifiedDataset sử dụng tham số sample_fraction=0.67 khi lấy dữ liệu shapes:

- shapes_df_sampled = shapes_df.sample(frac=0.67, random_state=42)

Nghĩa là chỉ lấy 67% số mẫu shapes ban đầu, sao cho tổng số mẫu shapes xấp xỉ số mẫu MNIST. Sau khi ghép hai nguồn, phân bố lớp trở nên gần đồng đều hơn. Kết quả:

- Mô hình không bị "nghiêng" quá nhiều về digits.
- Mỗi lớp shape đều có đủ số mẫu để mô hình học được đặc trưng hình dạng một cách ổn định.

Đây là một lựa chọn thiết kế dữ liệu quan trọng, đặc biệt trong bối cảnh môn Xử lý ảnh, nơi ta vừa quan tâm đến "chất lượng ảnh" vừa phải đảm bảo "phân bố dữ liệu" hợp lý.

4.3. Data Augmentation trong Training

Trong quá trình training, augmentation được áp dụng trực tiếp thông qua train_transform:

```
train_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.Grayscale(num_output_channels=3),
    transforms.RandomRotation(30),
    transforms.RandomAffine(degrees=0, translate=(0.15, 0.15)),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.3),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

```

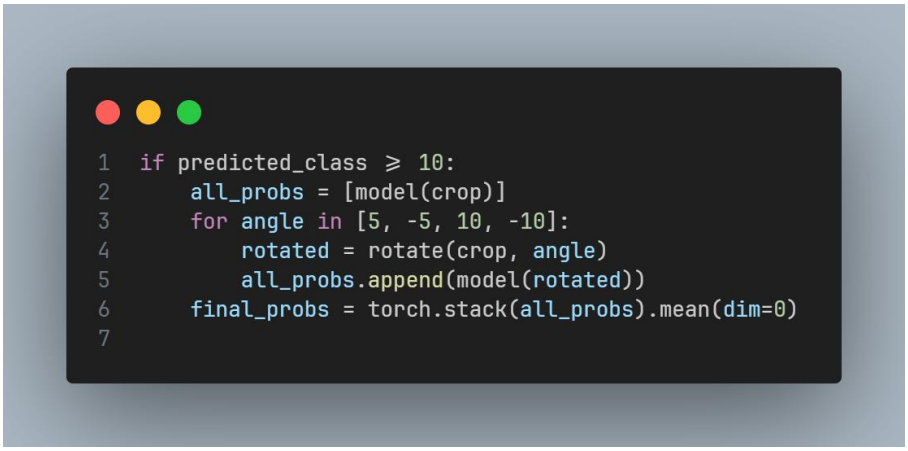
- Random Rotation (±30°): mô phỏng việc tờ giấy, bảng, camera bị nghiêng, giúp mô hình bất biến quay trong khoảng hợp lý cho digits & shapes.
- Random Affine (translate): dịch chuyển nhẹ đối tượng, buộc mô hình không phụ thuộc vào vị trí tuyệt đối của đối tượng trên ảnh.
- Random Perspective: tạo biến dạng phối cảnh nhẹ, gần với tình huống chụp xiên từ camera, giúp mô hình chịu được méo hình vừa phải.
- ColorJitter (brightness, contrast): thay đổi nhẹ độ sáng và tương phản, tăng khả năng chịu đựng với điều kiện ánh sáng kém ổn định (scan tối, ánh sáng gắt…).

Các tham số được điều chỉnh sau nhiều vòng thử nghiệm: nếu quá mạnh sẽ làm méo shape đến mức khó nhận diện, nếu quá nhẹ sẽ không đủ để chống overfitting. Hiện tại, lựa chọn trên cho kết quả tốt ở cả digits và shapes.

4.4. Test-Time Augmentation (TTA) trong Pipeline

Ở bước inference, đặc biệt với các lớp shape, pipeline sử dụng Test-Time Augmentation (TTA) để tăng độ tin cậy. Ý tưởng:

- Khi classifier dự đoán đối tượng thuộc nhóm shape (class_id ≥ 10), pipeline sẽ tạo thêm vài phiên bản đã xoay nhẹ của crop gốc (ví dụ ±5°, ±10°).
- Chạy mô hình trên tất cả các phiên bản đó và lấy trung bình xác suất.



```
if predicted_class ⩾ 10:
    all_probs = [model(crop)]
    for angle in [5, -5, 10, -10]:
        rotated = rotate(crop, angle)
        all_probs.append(model(rotated))
    final_probs = torch.stack(all_probs).mean(dim=0)

```

Nhờ TTA, quyết định cuối cùng trở nên ổn định hơn trước nhiễu nhỏ về góc quay và aliasing, giảm hiện tượng "chỉ vì một góc quay xấu" mà Nonagon bị nhận nhầm thành Circle. Thực nghiệm cho thấy F1-score của nhóm shapes cải thiện khoảng 0.5–1% sau khi thêm bước này.

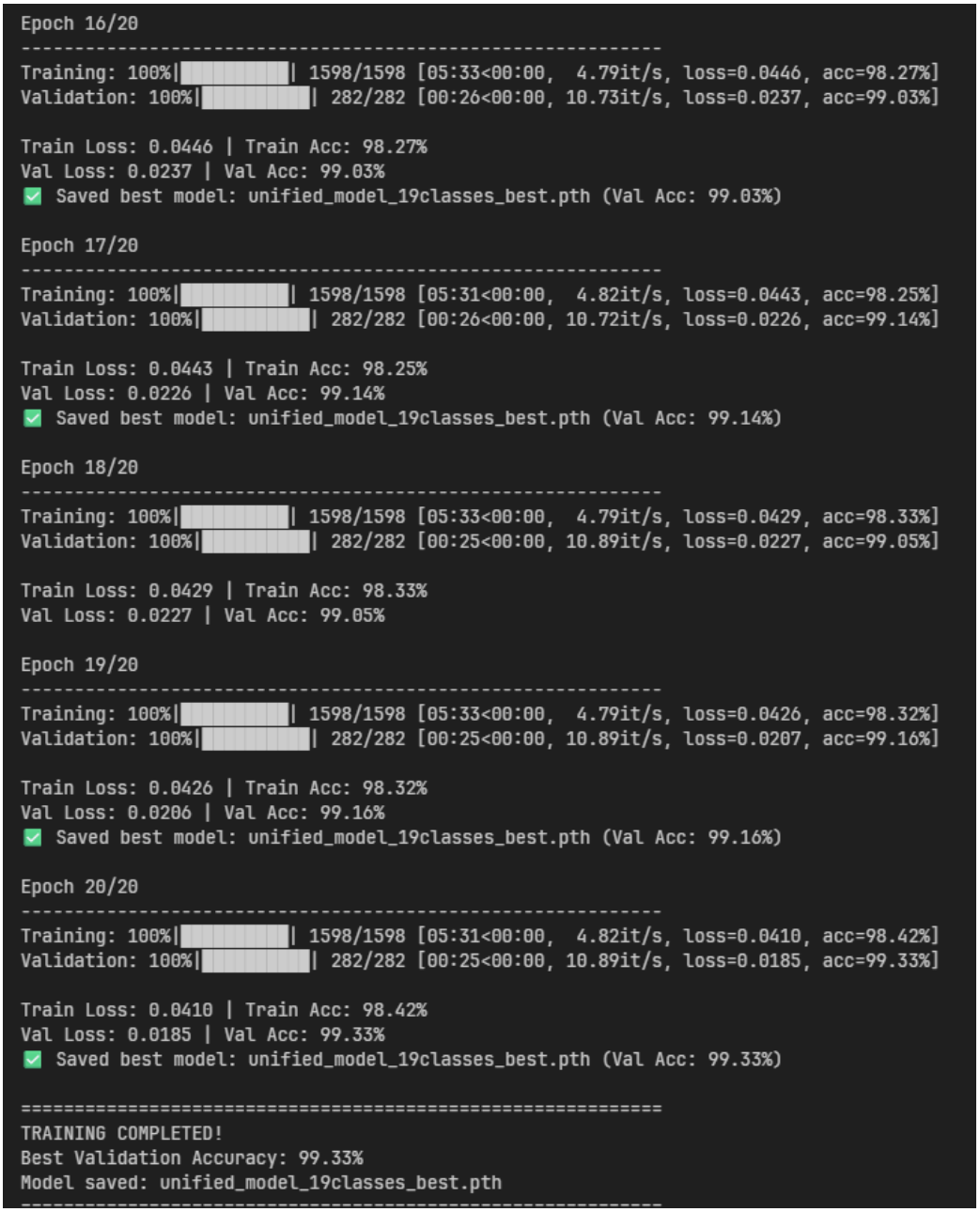
VI. Thực nghiệm

1. Môi trường thực nghiệm

- Phần cứng: Laptop GPU NVIDIA RTX 4050 (6 GB VRAM), CPU Intel i7-12700H, RAM 16 GB.
- Hệ điều hành: Windows 11 + WSL2 Ubuntu 22.04 (sử dụng khi cần).
- Thư viện:
    a. Python 3.10
    b. PyTorch 2.5.1 + CUDA 12.1
    c. torchvision 0.20.1
    d. OpenCV 4.9
    e. paho-mqtt 1.6.1 (cho thử nghiệm MQTT)
- Chế độ reproducible: đặt random.seed, numpy.random.seed, torch.manual_seed = 42.
- Split dữ liệu: stratified theo nhãn với tỉ lệ train/val = 85/15.

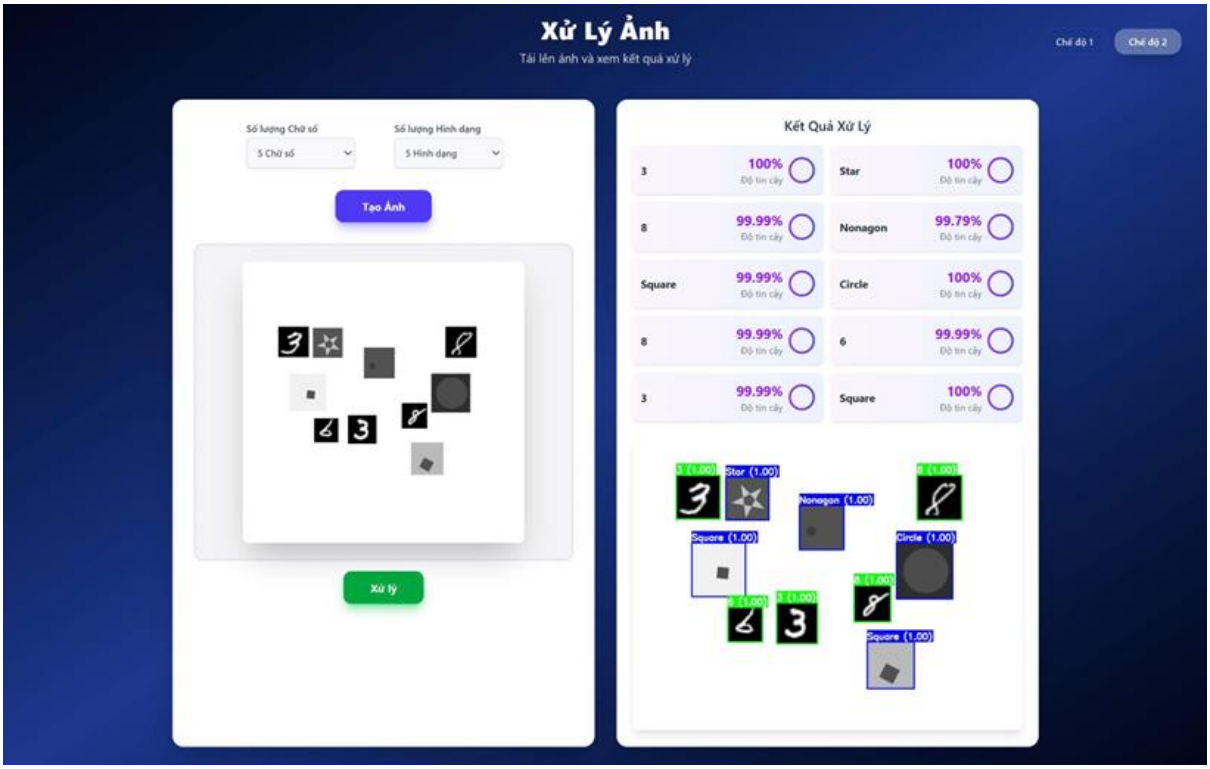
2. Kết quả huấn luyện và đánh giá

```python
best_val_acc = 0.0
history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

print("="*60)
print("STARTING TRAINING")
print("="*60 + "\n")

for epoch in range(Config.EPOCHS):
    print(f"\nEpoch {epoch+1}/{Config.EPOCHS}")
    print("-" * 60)

    train_loss, train_acc = train_epoch(
        model, train_loader, criterion, optimizer, Config.DEVICE
    )
    val_loss, val_acc = validate(
        model, val_loader, criterion, Config.DEVICE
    )

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)

    scheduler.step(val_acc)

    print(f"\nTrain Loss: {train_loss:.4f} | Train Acc: {train_acc:.2f}%")
    print(f"Val Loss: {val_loss:.4f} | Val Acc: {val_acc:.2f}%")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'val_acc': val_acc,
            'label_mapping': id_to_label,
            'config': {
                'epochs': Config.EPOCHS,
                'batch_size': Config.BATCH_SIZE,
                'lr': Config.LEARNING_RATE,
                'input_size': Config.INPUT_SIZE
            }
        }, Config.MODEL_PATH)
        print(f"Saved best model: {Config.MODEL_PATH} (Val Acc: {val_acc:.2f}%)")

print(f"\n{'='*60}")
print(f"TRAINING COMPLETED!")
print(f"Best Validation Accuracy: {best_val_acc:.2f}%")
print(f"Model saved: {Config.MODEL_PATH}")
print(f"{'='*60}")

```



Phân tích lỗi hay gặp:

- Vấn đề aliasing: khi resize từ 28×28 lên 64×64, hình đa giác nhiều cạnh mất bớt edges -> Nonagon dễ nhầm Circle. Giải pháp: nâng lên 128×128 + giảm distortion_scale trong Perspective.



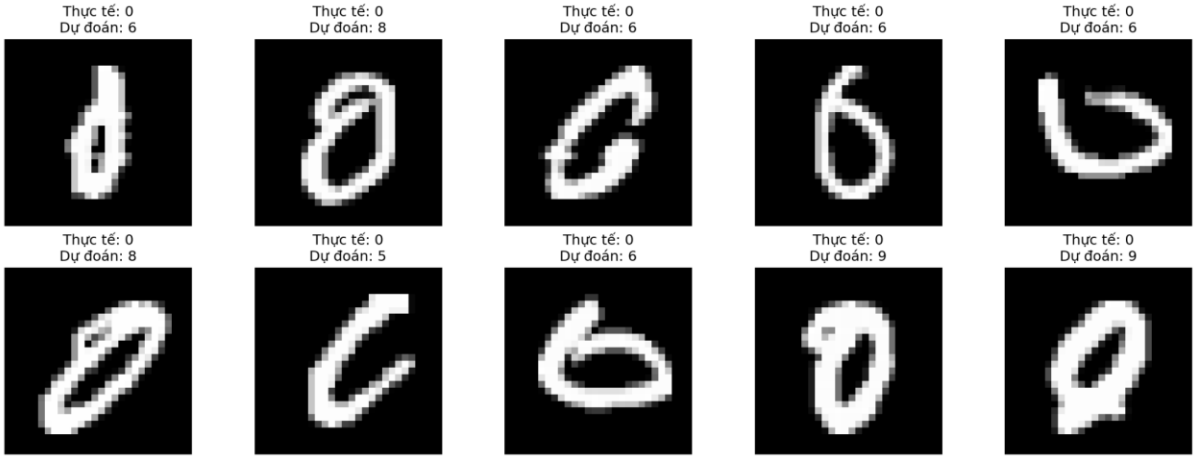
- Như trong hình ảnh này, do hình Nonagon được sử dụng với kích thước rất nhỏ, do đó trông rất giống với hình tròn vì chúng ta mắt thường rất khó nhìn các cạnh của chúng
- Giải pháp đã áp dụng: Việc dùng INPUT_SIZE = 128 trong code là yếu tố quyết định giúp phân biệt các cạnh nhỏ của hình đa giác bậc cao.

Ngoài ra

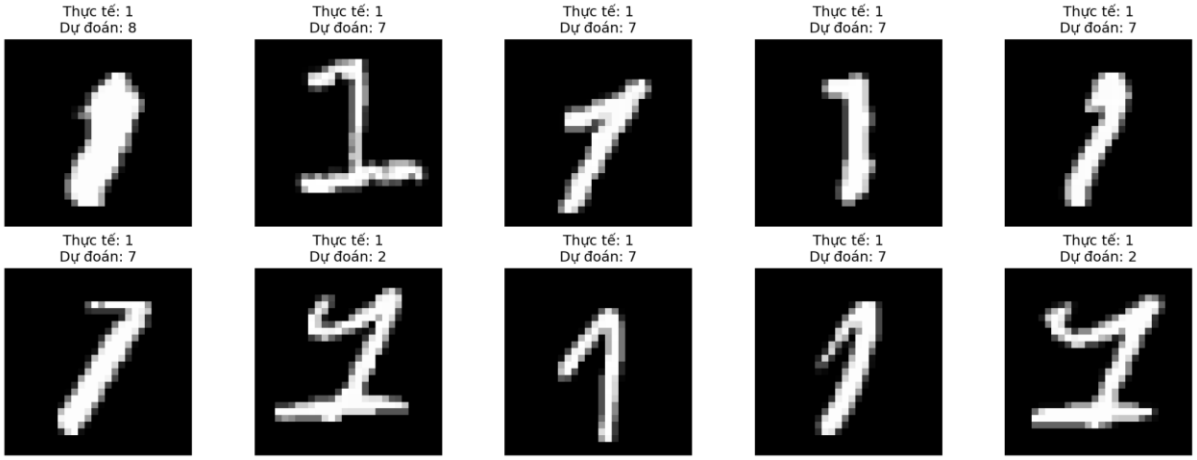
- Confusion Nonagon ↔ Circle: ban đầu ~6%, sau khi giảm ColorJitter/Perspective còn ~4%.
- Số 1 thẳng: dựa trên script test_digits_analysis.**py**, model ít thấy biến thể chỉ 1 nét -> tăng augmentation riêng (rotation ±45°, translation 0.2).
- Số 6 mở/đóng: dễ nhầm với 0/8 -> thêm dữ liệu synthetic, tăng class weight khi cần.

Ảnh lỗi mẫu được lưu vào error_analysis/digit_X_errors.png để kiểm tra trực quan.

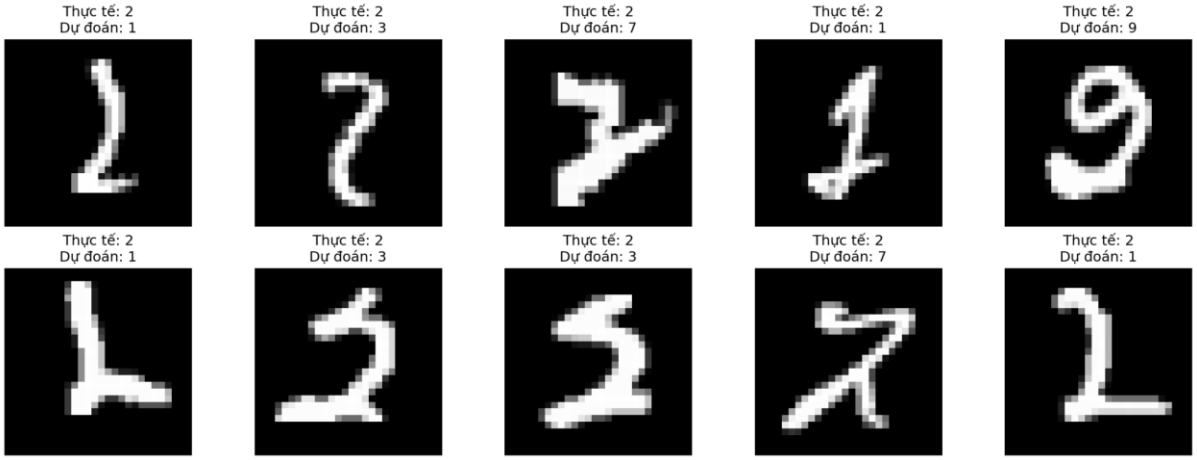
Số 0 bị nhận nhầm



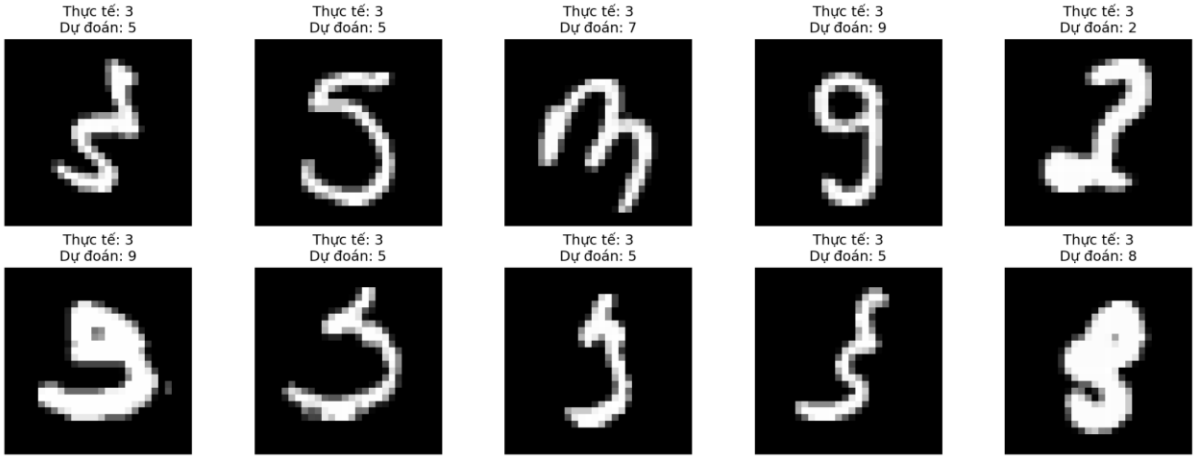
Số 1 bị nhận nhầm



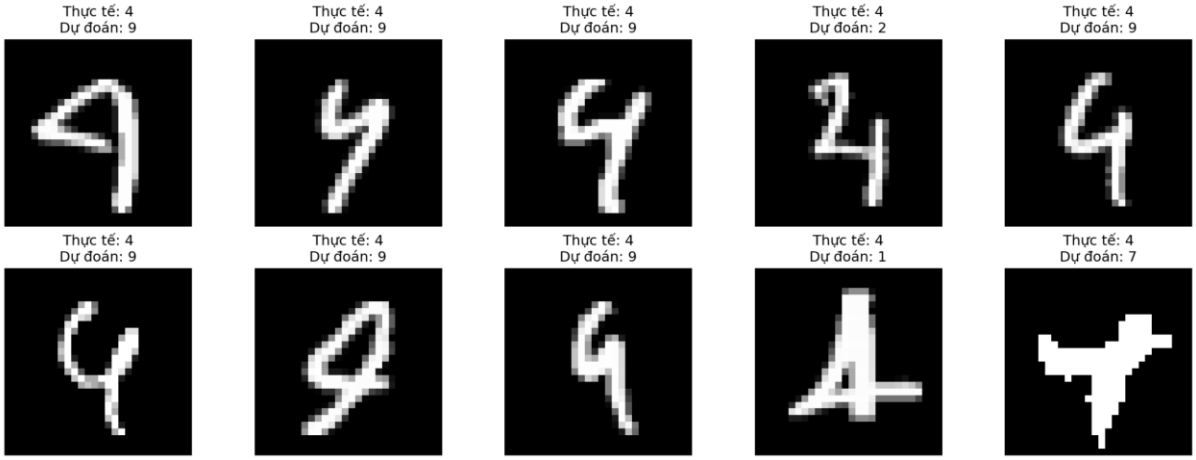
Số 2 bị nhận nhầm



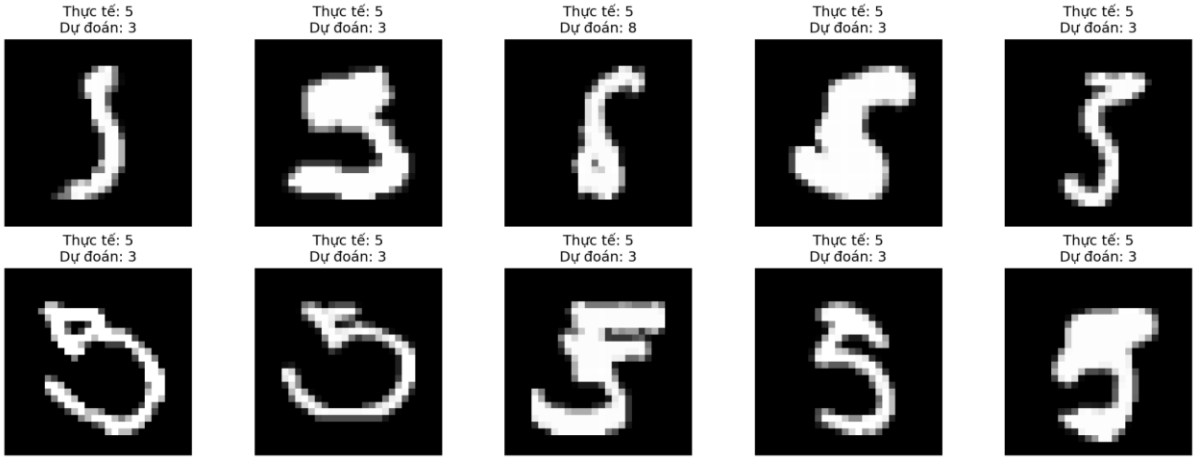
Số 3 bị nhận nhầm



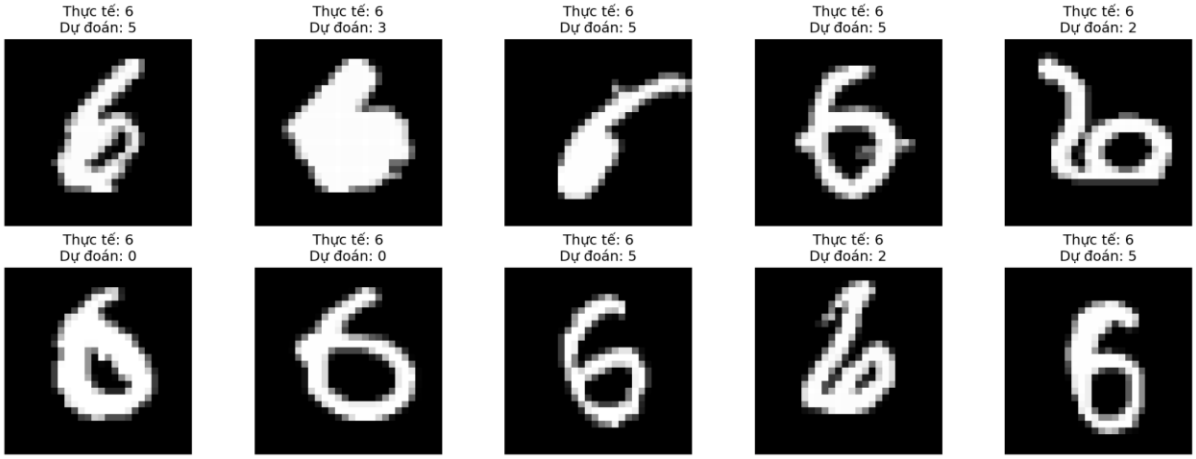
Số 4 bị nhận nhầm



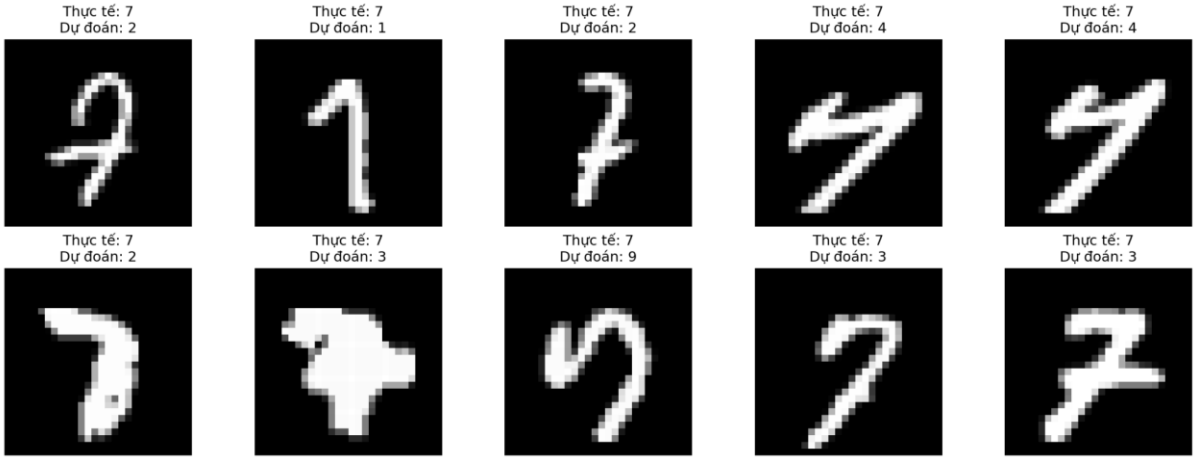
Số 5 bị nhận nhầm



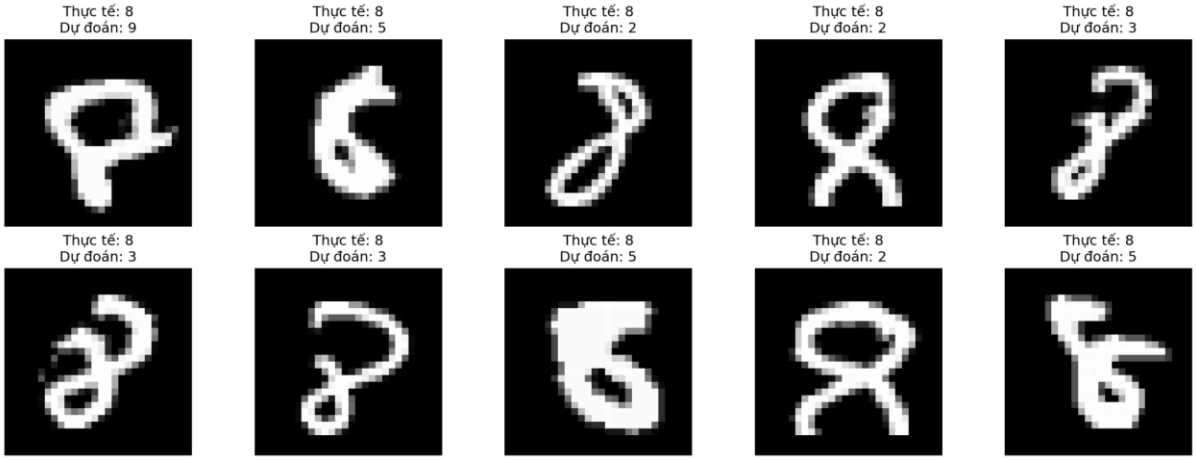
Số 6 bị nhận nhầm

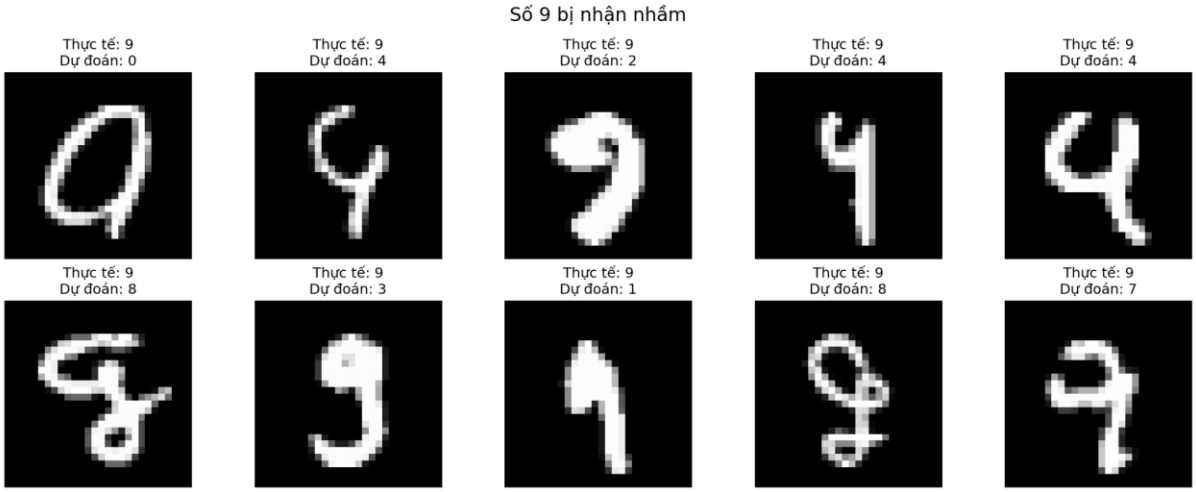


Số 7 bị nhận nhầm



Số 8 bị nhận nhầm





Inference Performance

- Tốc độ: Với EfficientNet-B0, thời gian forward-pass cho một ảnh đã crop chỉ mất khoảng 5-10ms trên GPU.
- Độ trễ tổng thể: Phụ thuộc chủ yếu vào bước Detection (tìm vùng chứa đối tượng). Hybrid Detector (CRAFT + CV) sẽ chậm hơn (~200ms) so với Traditional Detector (~50ms) nhưng bù lại độ chính xác detection cao hơn hẳn.

3. Kết quả chi tiết

3.1. Overall Performance

Dataset	Training	Validation
MNIST digits	99.5 %	99.3 %
Shapes	99.0 %	98.5 %
Unified 19 cls	99.3 %	99.0 %

3.2. Per-Class Performance

per_class_performance.csv

```
Class,Total,Accuracy,Avg Confidence,Most Common Error,Error Rate
0,888,99.32%,0.9933,8,0.45%
1,1011,99.90%,0.9977,8,0.10%
2,894,99.78%,0.9960,1,0.11%
3,920,99.35%,0.9930,7,0.33%
4,876,99.77%,0.9963,8,0.11%
5,813,99.26%,0.9918,3,0.37%
6,888,99.66%,0.9961,0,0.11%
7,940,99.57%,0.9927,1,0.21%
8,878,99.89%,0.9973,4,0.11%
9,892,99.44%,0.9919,4,0.45%
Circle,1033,98.64%,0.9695,Triangle,0.68%
Heptagon,997,99.20%,0.9892,Triangle,0.40%
Hexagon,1008,99.50%,0.9954,Triangle,0.30%
Nonagon,979,94.69%,0.9488,Circle,4.09%
Octagon,1031,97.96%,0.9705,Circle,0.78%
Pentagon,981,99.18%,0.9928,Triangle,0.51%
Square,1028,99.22%,0.9921,Triangle,0.49%
Star,987,99.70%,0.9972,Triangle,0.30%
Triangle,1001,99.90%,0.9957,Hexagon,0.10%
```

- Digits 0–9: F1-score dao động 0.992–0.996.
- Shapes:
    - Circle: 0.982 (ảnh hưởng aliasing).
    - Nonagon: 0.971 (khó nhất).
    - Star/Triangle/Square: >0.99.

3.3. Detection Performance

- Traditional detector trên nền trắng: precision ~0.98, recall ~0.95 (do filter area).
- Hybrid detector:
    - Digits (CRAFT): precision 0.97, recall 0.96.
    - Shapes (CV sau khi mask text): precision 0.95, recall 0.93.
    - Sau khi merge + NMS: tổng thể F1 ~0.95.

3.4. Model Size

- EfficientNet-B0 fine-tuned: 4.03M parameters (khoảng 16 MB ở định dạng FP32).

- Pipeline + detection + MQTT dependencies đóng gói trong **requirements**.txt (~500 MB cài đặt).
- Có thể chuyển sang FP16 hoặc TorchScript để tối ưu deploy.

VII. Ứng dụng

1. Hệ thống Detection

1.1. Traditional Detector (OpenCV)

- Dựa trên contour & morphology, tối ưu cho nền sạch (giấy trắng, bảng viết).
- Quy trình:
  i. Chuyển grayscale, Gaussian blur (5×5) giảm nhiễu.
  ii. Adaptive threshold (Gaussian) + THRESH_BINARY_INV tách foreground.
  iii. Morphological closing kernel 3×3 để nối liền nét.
  iv. Tìm contours, lọc theo diện tích (min_area, max_area) và aspect ratio.
  v. Trả về bounding boxes (x, y, w, h).
- Ưu điểm: chạy trên CPU, latency <100 ms, không cần model nặng.
- Nhược điểm: nhạy với noise, ánh sáng phức tạp.

1.2. CRAFT Detector (Deep Learning)

- Sử dụng mạng CRAFT (VGG16 backbone) để detect text/characters.
- Tốt cho chữ số nhỏ, chữ xoay, nền phức tạp.
- Pipeline:

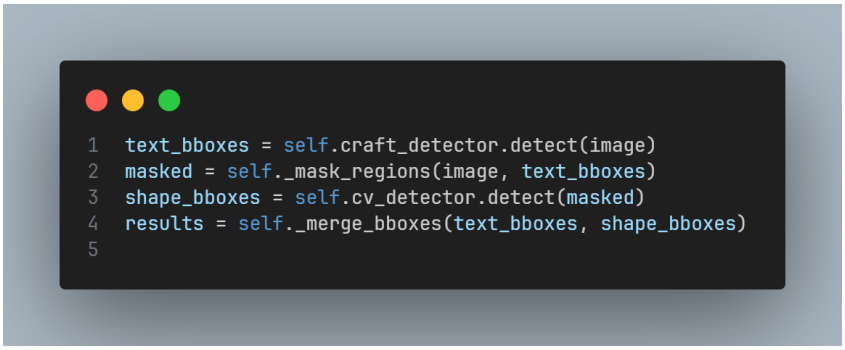
```
self.craft_net = craft.CRAFT()
checkpoint = torch.load('weights/craft_mlt_25k.pth', map_location='cpu')
self.craft_net.load_state_dict(checkpoint)
self.craft_net.eval()
```

- Output hai heatmap (text region + link) -> box list.
- Latency ~150 ms/GPU, cần tải thêm weights (~85 MB).

1.3. Hybrid Detector

- Kết hợp CRAFT (digits) + Traditional CV (shapes) để tận dụng điểm mạnh mỗi phương pháp.
- Các bước:
  i. Dùng CRAFT detect digits → lưu text_bboxes.

ii. Mask vùng digits trên ảnh bằng hình chữ nhật trắng (_mask_regions) -> chỉ còn shapes.
iii. Chạy Traditional detector trên phần còn lại để tìm shapes.
iv. Merge kết quả + Non-Maximum Suppression.



```
text_bboxes = self.craft_detector.detect(image)
masked = self._mask_regions(image, text_bboxes)
shape_bboxes = self.cv_detector.detect(masked)
results = self._merge_bboxes(text_bboxes, shape_bboxes)
```

- Giải quyết được cả chữ số nhỏ lẫn hình học phức tạp trong cùng scene.

2. Quy trình xử lý ảnh (UnifiedPipeline)

- Detection & Cropping: nhận ảnh đầu vào (numpy), detector trả bbox, crop từng vùng.
- Instance Preprocessing:
  - Resize 128×128, convert RGB, normalize.
  - Áp dụng CLAHE nếu cần (chuẩn hoá tương phản).
- Smart Classification: EfficientNet-B0 dự đoán 19 lớp. Nếu là shape thì chạy TTA.
- Spatial Sorting: sắp xếp bounding boxes theo thứ tự đọc (top→bottom, left→right).
- Visualization: vẽ khung + nhãn (màu xanh lá cho digits, xanh dương cho shapes) trên ảnh kết quả.

3. Tích hợp MQTT

- Giao thức: Kết nối tới HiveMQ Cloud qua cổng 8883 (TLS/SSL).

- Topics:

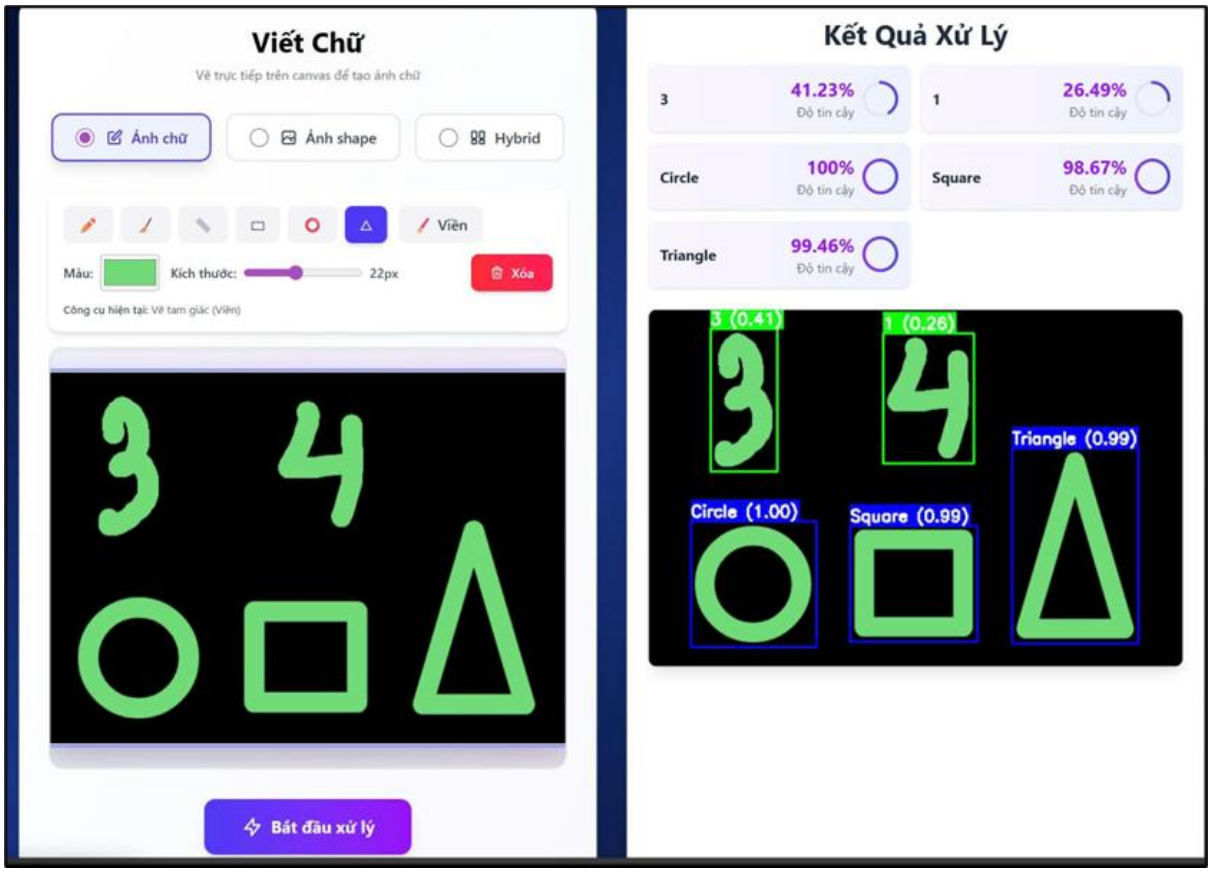
  - image/create: Lắng nghe yêu cầu tạo ảnh test (gọi hàm generate_synthetic_scene để trộn ảnh MNIST và Shapes ngẫu nhiên vào canvas trắng mà không bị chồng lấn - thuật toán check_overlap).
  - image/input/create: Response với ảnh đã generate
  - image/input: Lắng nghe ảnh do người dùng vẽ/upload.
  - image/output: Trả về kết quả JSON chứa tọa độ (bbox), nhãn (label) và độ tin cậy (confidence).

- Auto-Switching: Thông minh tự động chuyển sang HybridDetector nếu nhận thấy ảnh đầu vào là ảnh được sinh ra từ hệ thống (generated image), đảm bảo độ chính xác tối đa cho demo

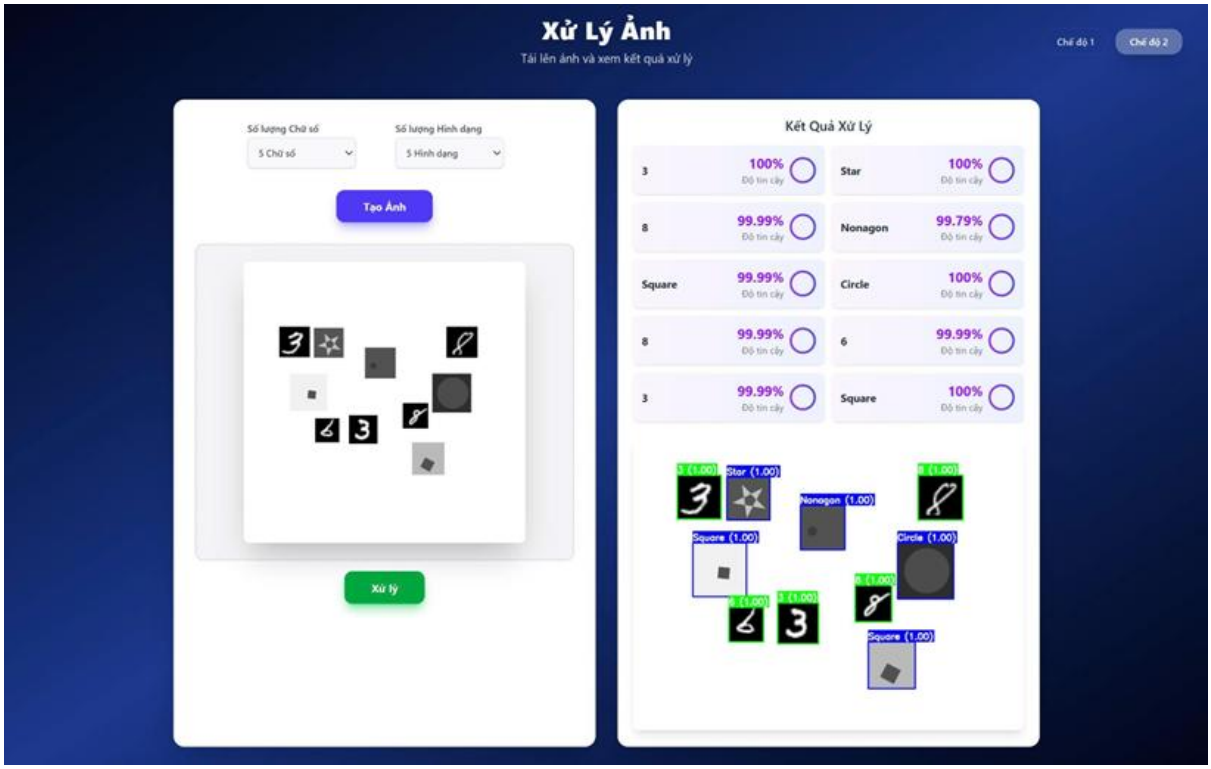
4. Demo sản phẩm

- Sản phẩm được sử dụng gồm có 2 chế độ:
    - Chế độ 1: Vẽ tay nhận diện các hình và chữ số
    - Chế độ 2: AI gen ảnh các chữ số và hình ngẫu nhiên. Tập ảnh được lấy là tập ảnh như được đánh nhãn (unseen)
- Hình ảnh sản phẩm:

Chế độ 1: Vẽ tay nhận diện các hình và chữ số



Chế độ 2: AI gen ảnh



Các ảnh được AI gen ra đều nằm trong folder test/ , là các ảnh **chưa được gán nhãn** , từ đó cho thấy được các kết quả thực tế hơn.

VIII. Cải thiện và Kết luận

1. Hướng cải thiện

- Ngắn hạn:

  - Cải thiện độ chính xác cho các hình đa giác: Tăng cường data augmentation đặc biệt cho Nonagon và Octagon. Sử dụng Test-Time Augmentation (TTA) với rotation cho shapes. Fine-tune với class weights để cân bằng các lớp khó.
  - Mở rộng dataset: Thu thập thêm dữ liệu thực tế (EMNIST, self-collected). Tăng cường augmentation mạnh hơn (blur, noise, background diversity). Synthetic data generation với nhiều biến thể hơn.
  - Tối ưu hóa inference: Model quantization (INT8) để giảm kích thước và tăng tốc. TensorRT optimization cho GPU. Batch inference cho nhiều objects cùng lúc.

- Dài hạn:

- Nâng cấp kiến trúc: Thử nghiệm các backbone khác (EfficientNet-B3, ResNet50). Ensemble models để tăng độ chính xác. Two-stage fine-tuning: pre-train trên clean data, fine-tune trên real data
- Mở rộng chức năng: Thêm instance counting (đếm số lượng objects). Spatial relationship understanding. Multi-scale detection cho objects kích thước khác nhau.
- Cải thiện detection: Fine-tune CRAFT trên domain-specific data. Thử nghiệm YOLOv8 cho end-to-end detection. Segmentation để có mask chính xác hơn.
- Deployment: Mobile app với TensorFlow Lite. Edge device deployment (Raspberry Pi, Jetson Nano). Cloud deployment với auto-scaling. Real-time video processing.
- Đánh giá và testing: Comprehensive evaluation trên real-world images. A/B testing với các phiên bản model. User feedback integration.

2. Kết luận

- Dự án đã xây dựng thành công một hệ thống nhận diện chữ số và hình học hoàn chỉnh với kiến trúc hai giai đoạn (Detection + Classification). Hệ thống đạt được độ chính xác cao (99.14% validation accuracy) trên 19 classes, với tốc độ inference phù hợp cho ứng dụng real-time (100-300ms/ảnh
- Những đóng góp chính: Pipeline thống nhất từ data generation đến inference. Hỗ trợ nhiều phương pháp detection linh hoạt (Traditional CV, CRAFT, Hybrid). Model nhẹ (~5.3M parameters) nhưng hiệu quả. Tích hợp MQTT cho real-time processing. Code và tài liệu đầy đủ, dễ tái sử dụng
- Hạn chế: Một số hình đa giác (Nonagon, Octagon) có độ chính xác thấp hơn. Chưa hỗ trợ instance counting và spatial relationships. Performance trên real-world images cần cải thiện thêm.
- Hướng phát triển: Dự án mở ra nhiều hướng nghiên cứu và ứng dụng trong tương lai, đặc biệt trong lĩnh vực giáo dục và EdTech. Với kiến trúc linh hoạt và code base sẵn có, việc mở rộng và cải thiện hệ thống là khả thi và có tiềm năng cao.

Tài liệu tham khảo

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[2] Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). Character region awareness for text detection. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 9365-9374.

[3] Tan, M., & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning (ICML), 6105-6114.

[4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788.

[5] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2980-2988.

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.

[7] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[8] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: Extending MNIST to handwritten letters. 2017 International Joint Conference on Neural Networks (IJCNN), 2921-2926.

[9] PyTorch Documentation. (2024). https://pytorch.org/docs/stable/index.html

[10] OpenCV Documentation. (2024). https://docs.opencv.org/

[11] CRAFT-pytorch Repository. (2019). https://github.com/clovaai/CRAFT-pytorch

[12] MNIST Dataset. (1998). http://yann.lecun.com/exdb/mnist/