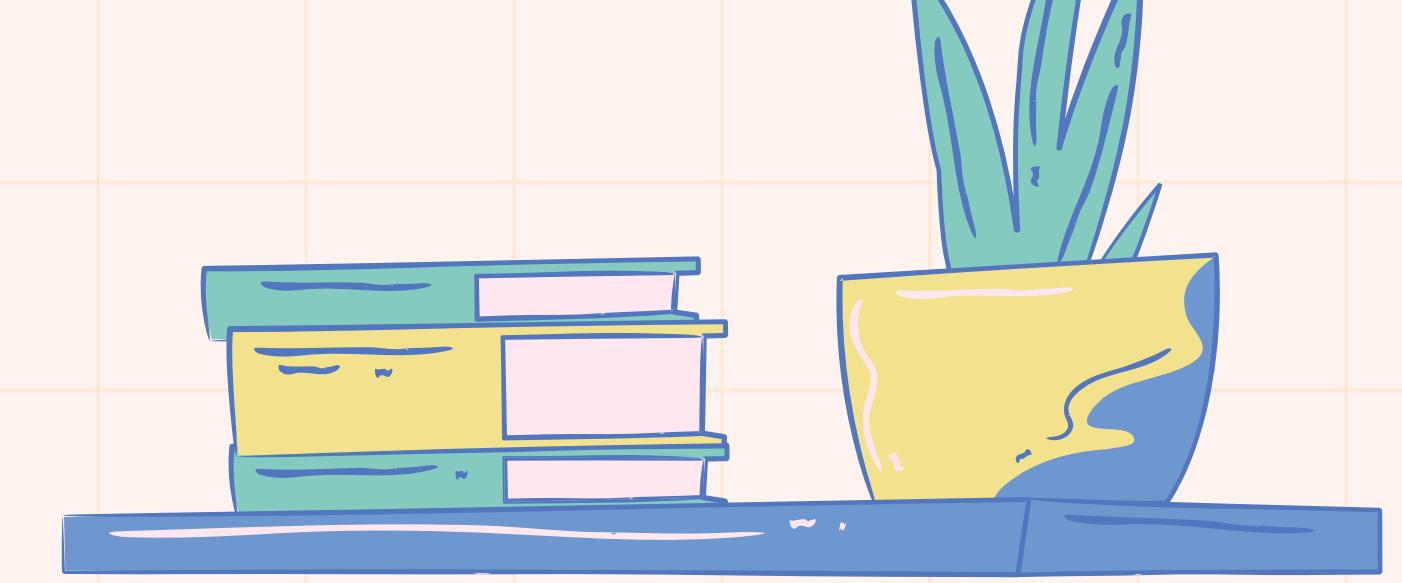


NHẬN DIỆN CHỮ VIẾT TAY MNIST VÀ HÌNH DẠNG CƠ BẢN VIẾT TAY SỬ DỤNG MẠNG CNN DỰA TRÊN RESNET

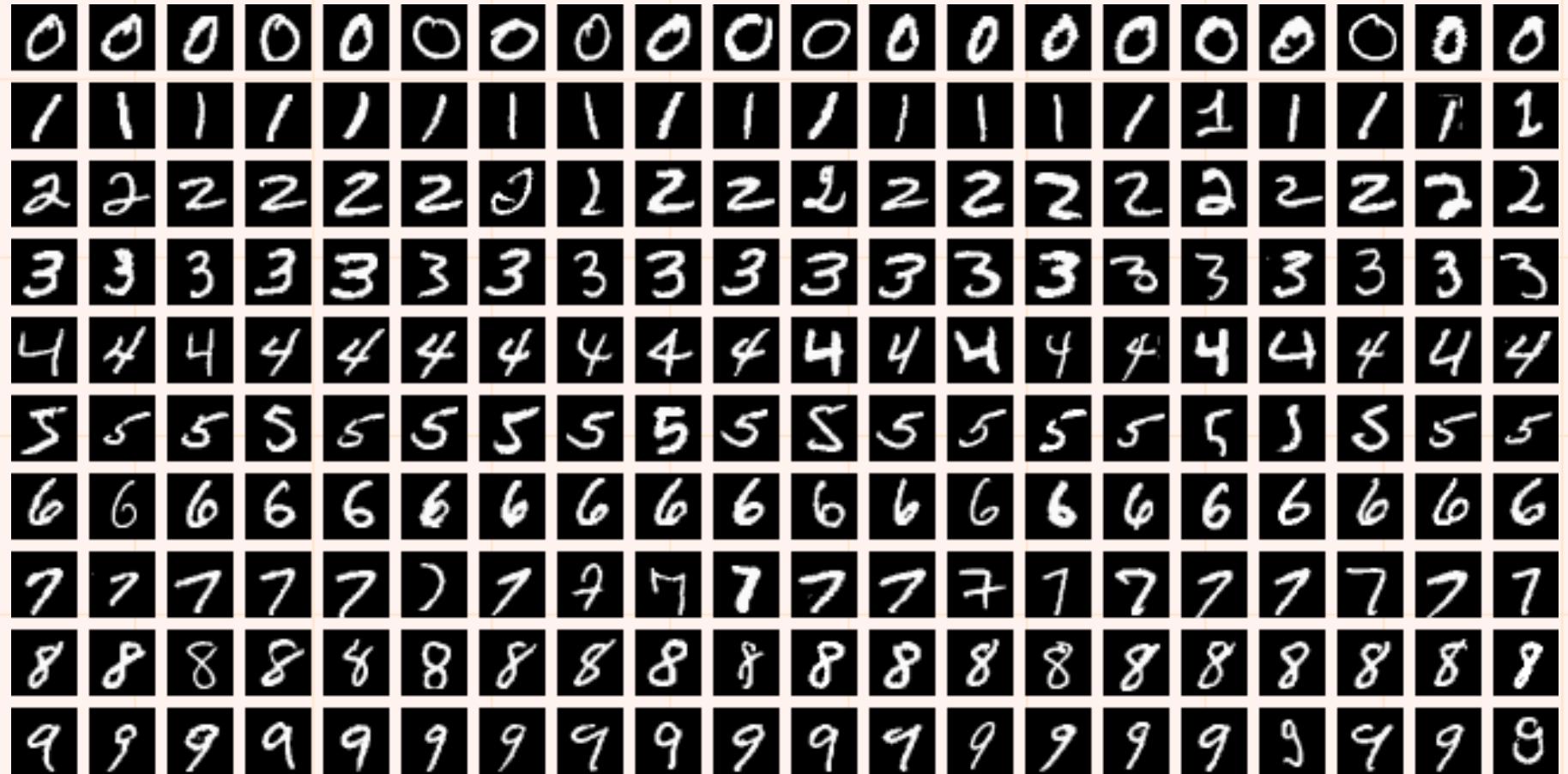


Computer Vision

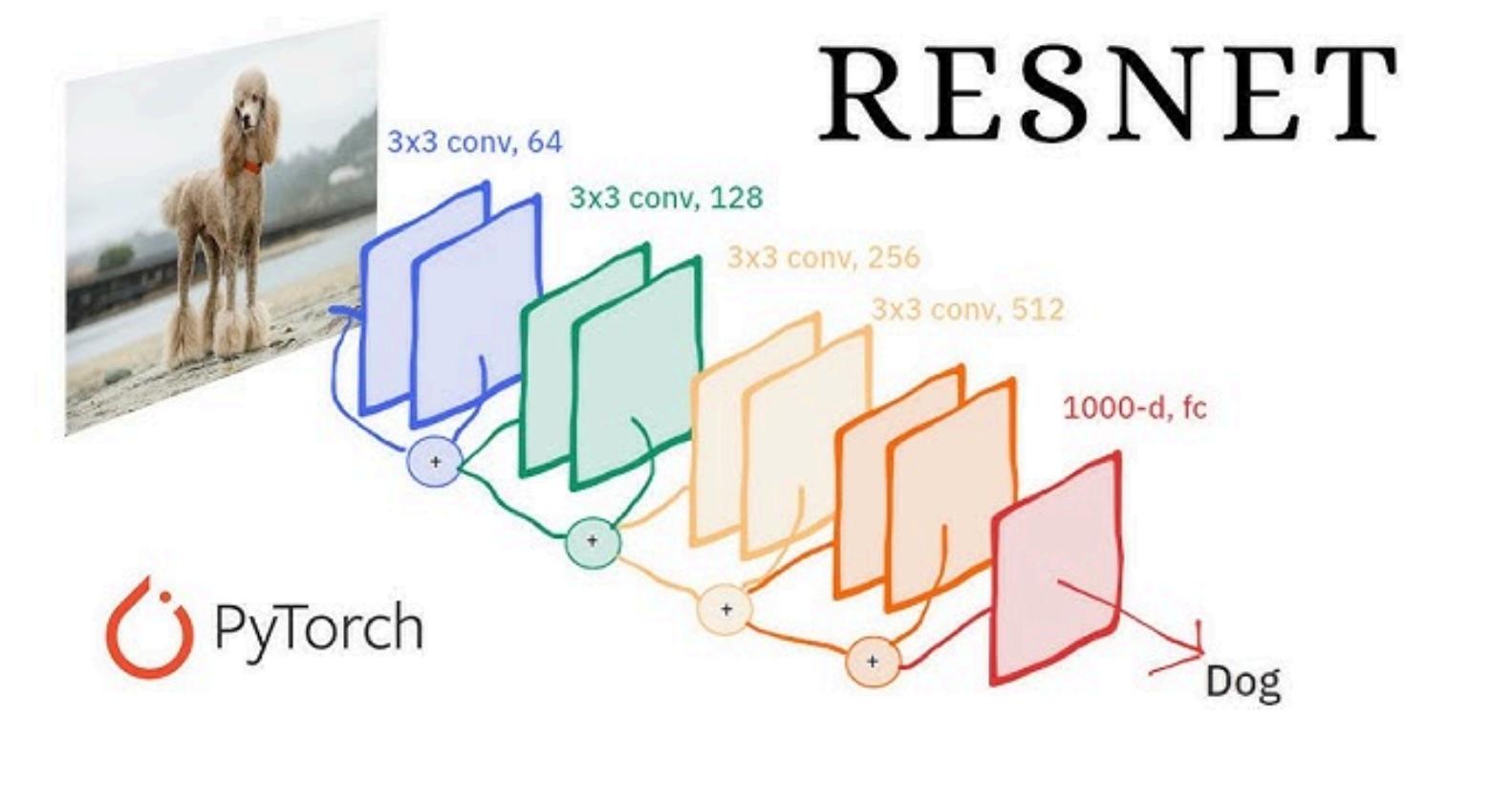


Trong bối cảnh Cách mạng Công nghiệp 4.0, Trí tuệ Nhân tạo và đặc biệt là Deep Learning đang thay đổi mạnh mẽ cách máy móc hiểu thế giới xung quanh. Một trong những hướng phát triển quan trọng nhất chính là Thị giác máy tính, nơi máy có thể “nhìn” và phân tích hình ảnh giống con người.

Computer Vision

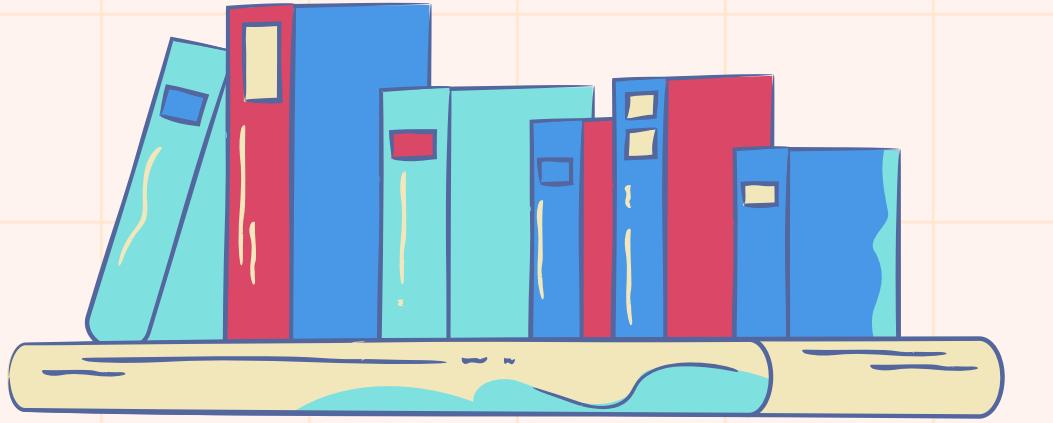


Bộ dữ liệu MNIST đã trở thành thước đo kinh điển cho mọi mô hình nhận diện ảnh.



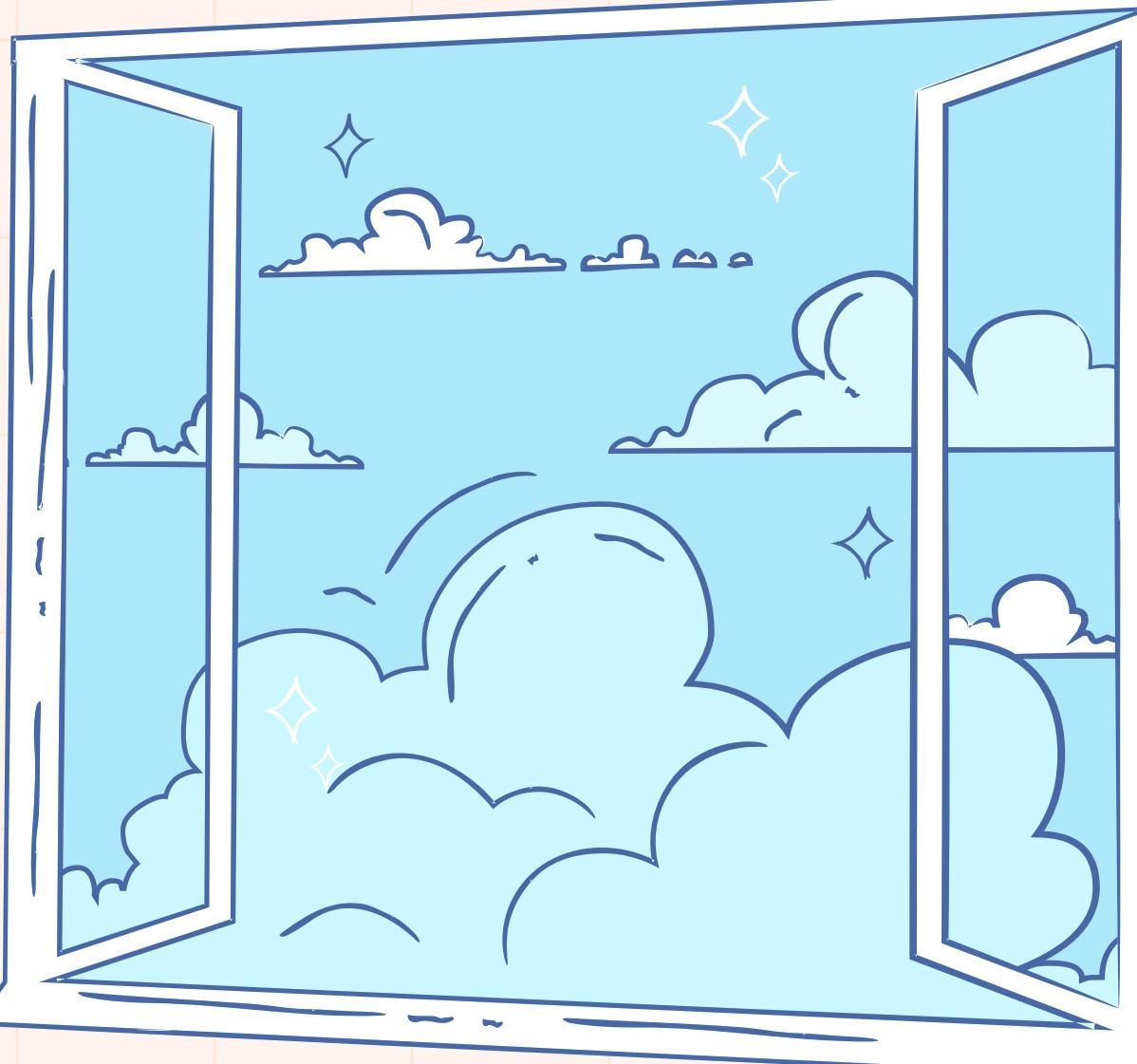
Khi các mạng CNN ngày càng sâu, chúng gặp phải vấn đề suy giảm độ dốc và sự ra đời của ResNet với cơ chế “skip connection” đã mở ra khả năng xây dựng các mô hình rất sâu mà vẫn đảm bảo hiệu quả.

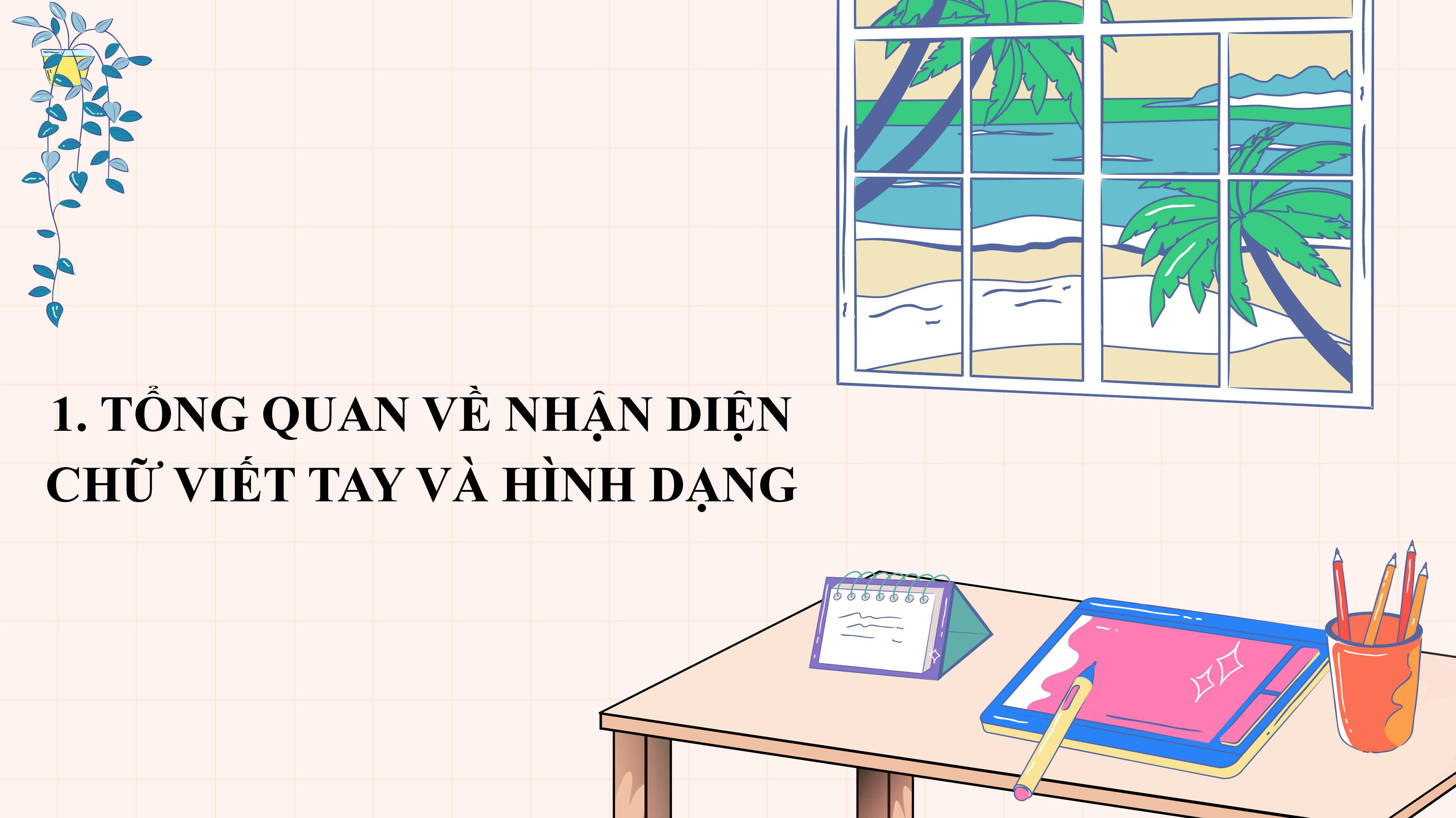
→ Nhóm quyết định nghiên cứu và ứng dụng kiến trúc ResNet cho bài toán nhận diện chữ số MNIST và hình dạng viết tay – một hướng đi vừa mang tính học thuật, vừa có giá trị thực tiễn cao.



Mục tiêu bài học

- Tìm hiểu lý thuyết về xử lý ảnh, CNN và kiến trúc ResNet.
- Xây dựng – huấn luyện – đánh giá mô hình ResNet trên dữ liệu MNIST và bộ hình dạng viết tay tự tạo.
- Xây dựng hệ thống hoàn chỉnh nhận diện chữ số và hình dạng cơ bản với độ chính xác cao.





1. TỔNG QUAN VỀ NHẬN DIỆN CHỮ VIẾT TAY VÀ HÌNH DẠNG

Xử lý ảnh (Image Processing)



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



Xử lý ảnh (Image Processing)

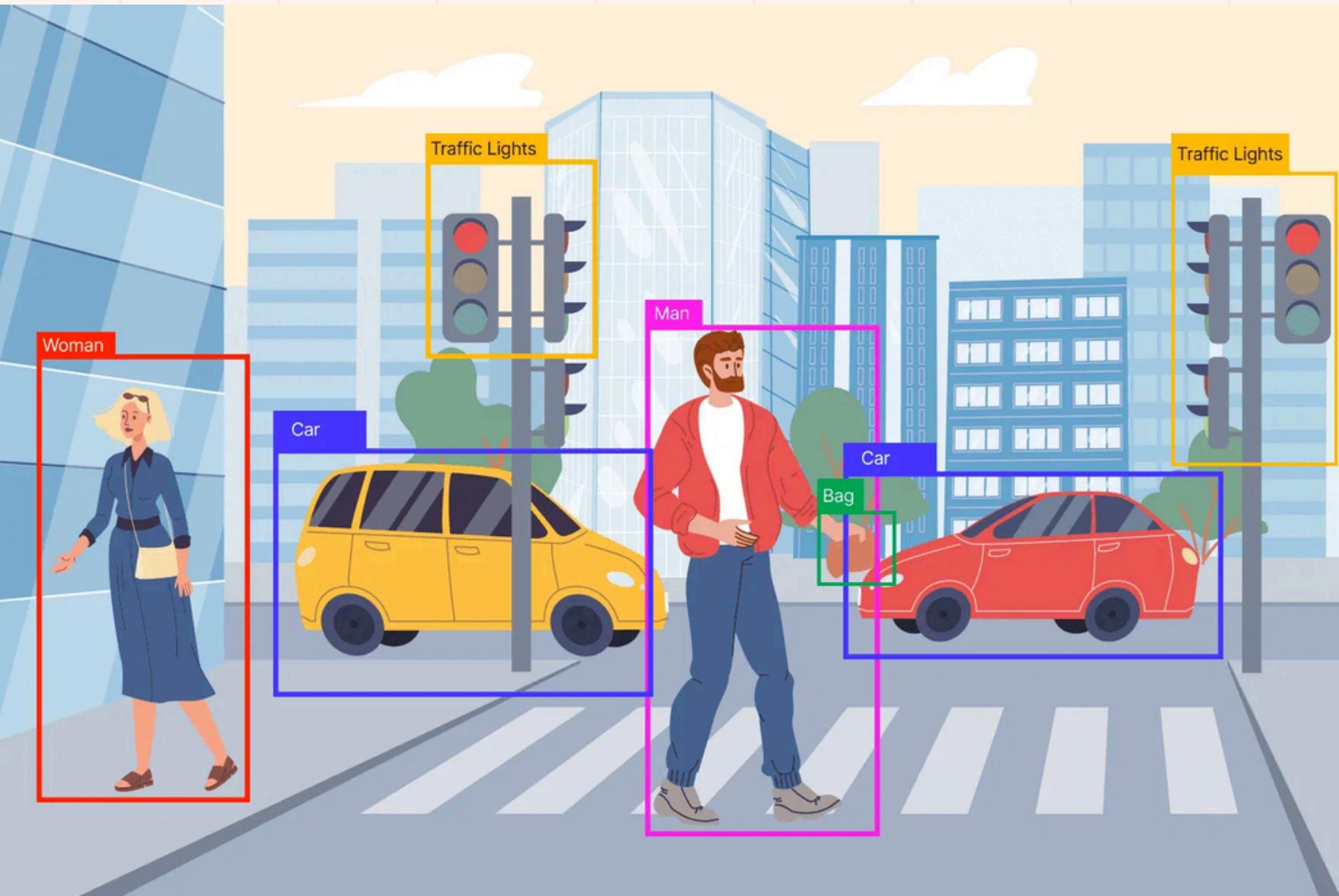
Một quy trình xử lý ảnh cơ bản thường bao gồm các bước:

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



Nhận diện đối tượng (Object Recognition)

- Nhận diện đối tượng là nhiệm vụ cốt lõi của Computer Vision, giúp máy xác định và phân loại đối tượng trong ảnh.
- Trước đây, phương pháp truyền thống dựa vào đặc trưng thủ công (SIFT, HOG) + mô hình ML cổ điển, nhưng kém hiệu quả với dữ liệu phức tạp.
- Deep Learning, đặc biệt CNN, đã thay đổi hoàn toàn lĩnh vực khi có thể tự học đặc trưng từ ảnh thông qua nhiều lớp.
- Trong đề tài này:
 - Xử lý ảnh dùng để chuẩn hóa ảnh đầu vào.
 - Nhận diện đối tượng được thực hiện bằng CNN/ResNet để phân loại chữ số và hình dạng viết tay.



Nhận diện hình dạng cơ bản viết tay

Tâm quan trọng

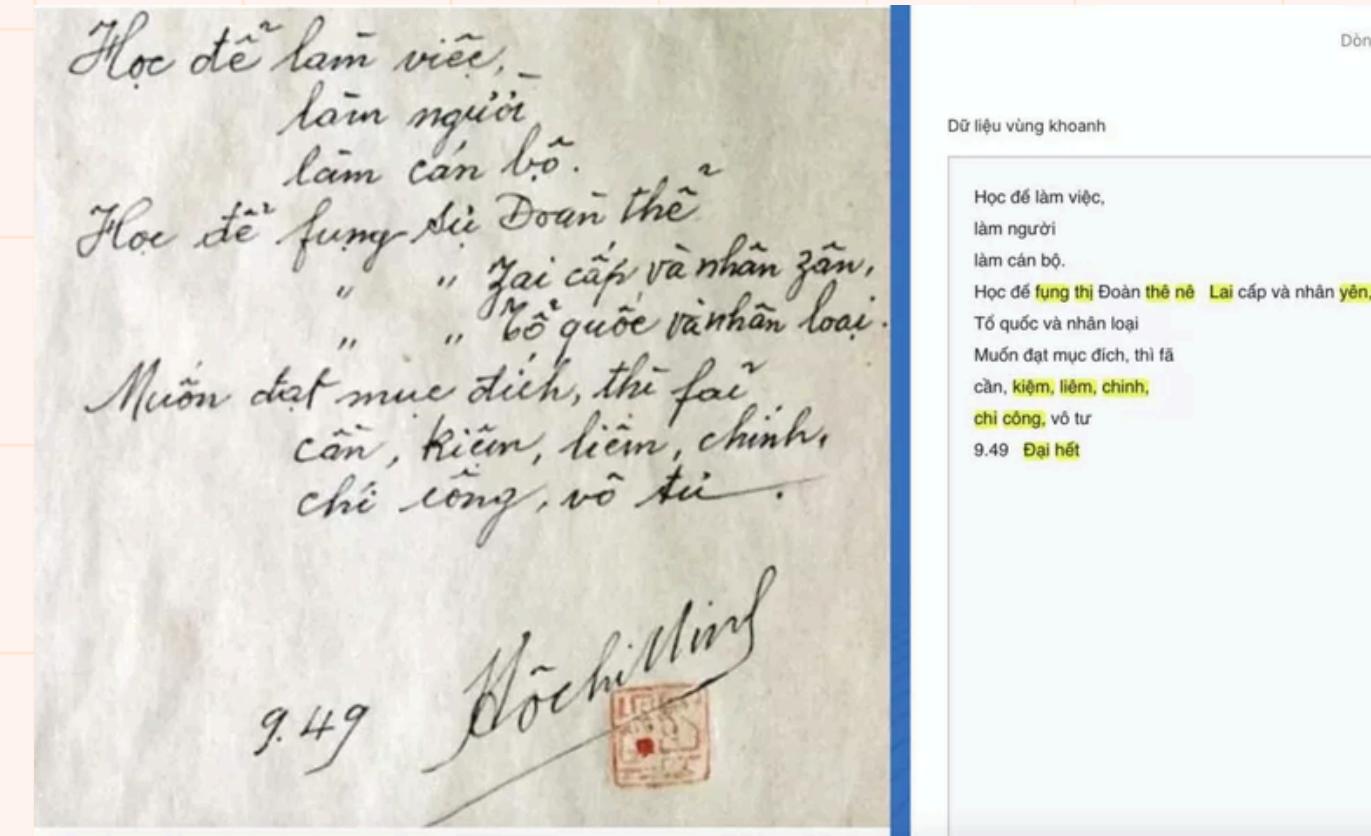
Những hình dạng này được coi là các "nguyên tố" (primitives) trong nhận diện thị giác. Hầu hết các vật thể phức tạp trong thế giới thực đều có thể được phân rã hoặc mô tả gần đúng bằng sự kết hợp của các hình dạng cơ bản này. Khả năng nhận diện chúng là cơ sở cho các tác vụ như:

- Phân tích và hiểu các sơ đồ khối (flowcharts), bản vẽ kỹ thuật.
 - Hỗ trợ các phần mềm thiết kế đồ họa (CAD, UI/UX).
 - Giúp robot tương tác với các vật thể trong môi trường.

Thách thức của dữ liệu "Viết tay"

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

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



Nhận diện hình dạng cơ bản viết tay

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

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



Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology)

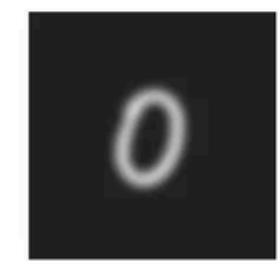
MNIST Dataset



4 (4)



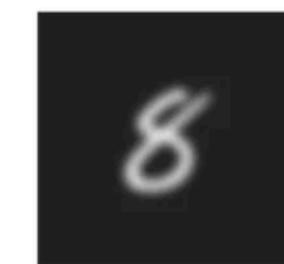
1 (1)



0 (0)



7 (7)



8 (8)



1 (1)



2 (2)



7 (7)

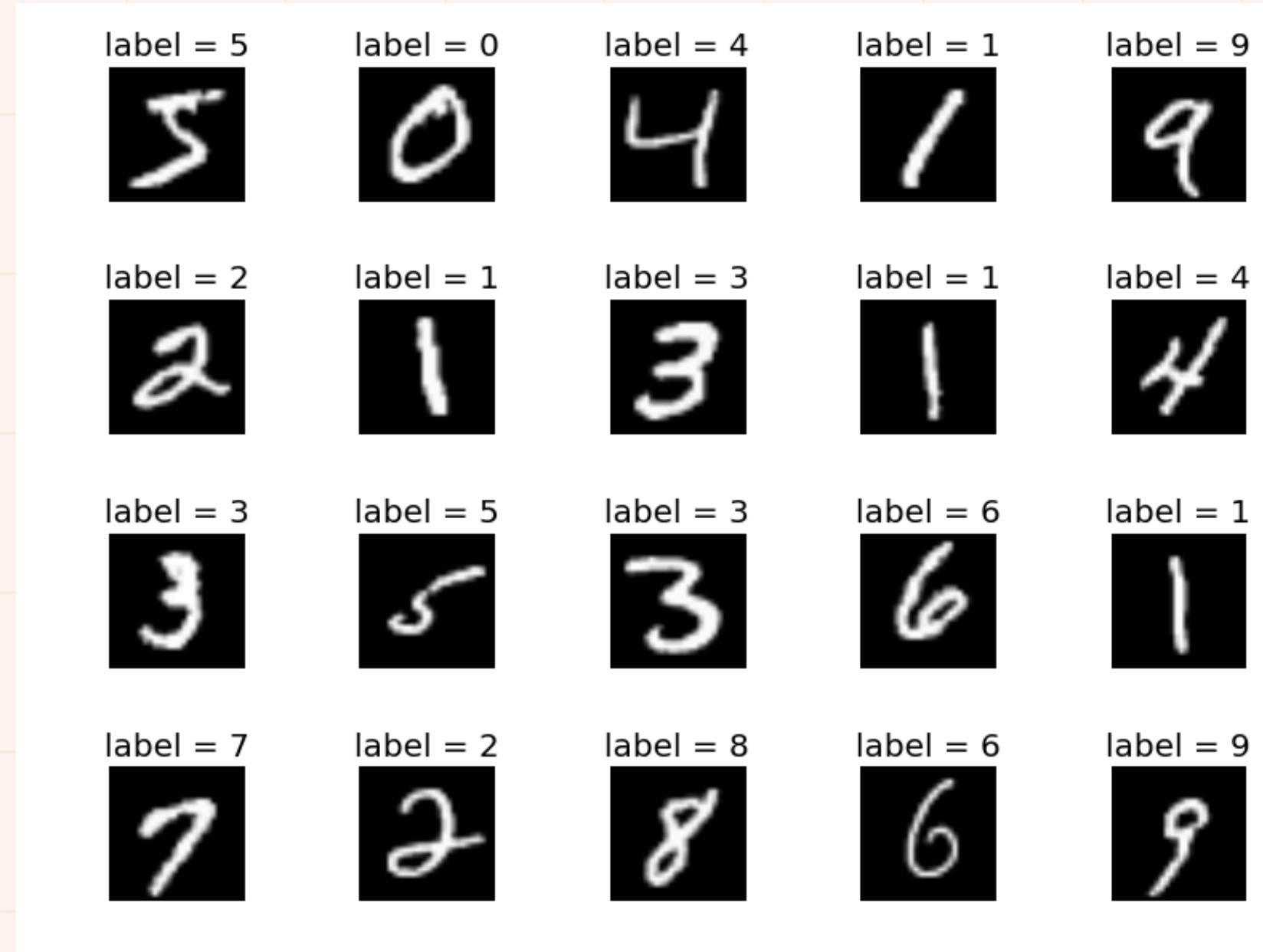


1 (1)

MNIST là một trong những bộ dữ liệu nổi tiếng và được sử dụng rộng rãi nhất trong lĩnh vực học máy và thị giác máy tính. Bộ dữ liệu này là một tập hợp lớn các hình ảnh chữ số viết tay, được trích xuất từ các tài liệu của Cục Điều tra Dân số Hoa Kỳ và học sinh trung học. Các đặc điểm kỹ thuật chính bao gồm:

- Nội dung: Gồm 10 lớp (classes), tương ứng với 10 chữ số viết tay từ 0 đến 9.
- Quy mô: Toàn bộ dữ liệu được chia thành hai tập:
 - Tập huấn luyện (Training set): Chứa 60.000 hình ảnh.
 - Tập kiểm thử (Test set): Chứa 10.000 hình ảnh.
- Giúp các nhà nghiên cứu có một phương pháp chuẩn hóa để huấn luyện và so sánh hiệu suất của các mô hình.
- Định dạng ảnh:
 - Mỗi hình ảnh là ảnh thang xám (grayscale).
 - Kích thước ảnh đã được chuẩn hóa về 28x28 pixel.
 - Các chữ số đã được căn giữa, chuẩn hóa về kích thước trong một ô 20x20 pixel và được đặt vào giữa ảnh → loại bỏ các biến thể về vị trí và kích thước, cho phép mô hình tập trung vào đặc trưng của nét chữ.

Bộ dữ liệu MNIST (Modified National Institute of Standards and Technology)



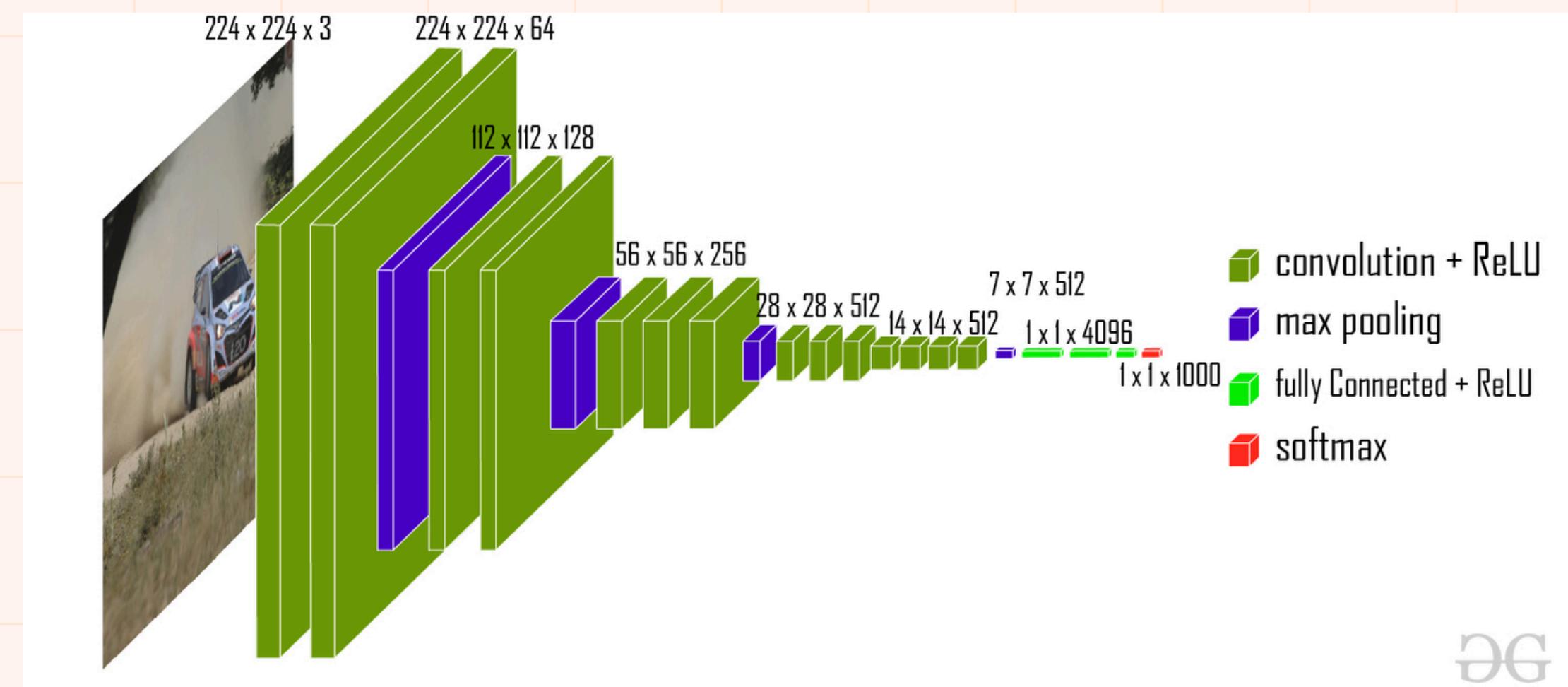
Mặc dù MNIST được coi là một bài toán tương đối "dễ" đối với các mô hình học sâu hiện đại (nhiều mô hình có thể đạt độ chính xác trên 99%), nó vẫn giữ một vai trò quan trọng:

- Benchmark kinh điển: MNIST là tiêu chuẩn vàng để kiểm chứng hiệu quả mô hình trước khi thử trên các bộ dữ liệu phức tạp hơn (như CIFAR-10 hay ImageNet).
- Phục vụ học tập & nghiên cứu: Dễ dùng, sạch, nhẹ → phù hợp để thử nghiệm nhanh các ý tưởng về kiến trúc mạng nơ-ron.
- Nền tảng cho OCR: Thành công trên MNIST tạo tiền đề cho nhiều ứng dụng thực tế như đọc mã bưu chính, xử lý séc, số hóa tài liệu, nhận dạng biển số,...

Mạng nơ-ron tích chập (CNN)

CNN (Convolutional Neural Network) là kiến trúc học sâu chuyên xử lý dữ liệu dạng lưới, điển hình là ảnh. Nó khắc phục nhược điểm của MLP: không cần “duỗi thẳng” ảnh → giữ được cấu trúc không gian và giảm số lượng tham số. CNN tự động học đặc trưng theo tầng bậc: từ cạnh & góc → hình dạng → vật thể. Một số điểm mạnh của CNN:

- Kết nối cục bộ (học trong vùng lân cận)
 - Chia sẻ tham số (một bộ lọc dùng chung cho toàn ảnh)
- Giúp mô hình hiệu quả, ít overfitting hơn.



Mạng nơ-ron tích chập (CNN)

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

1. *Convolutional Layer* (Lớp tích chập)

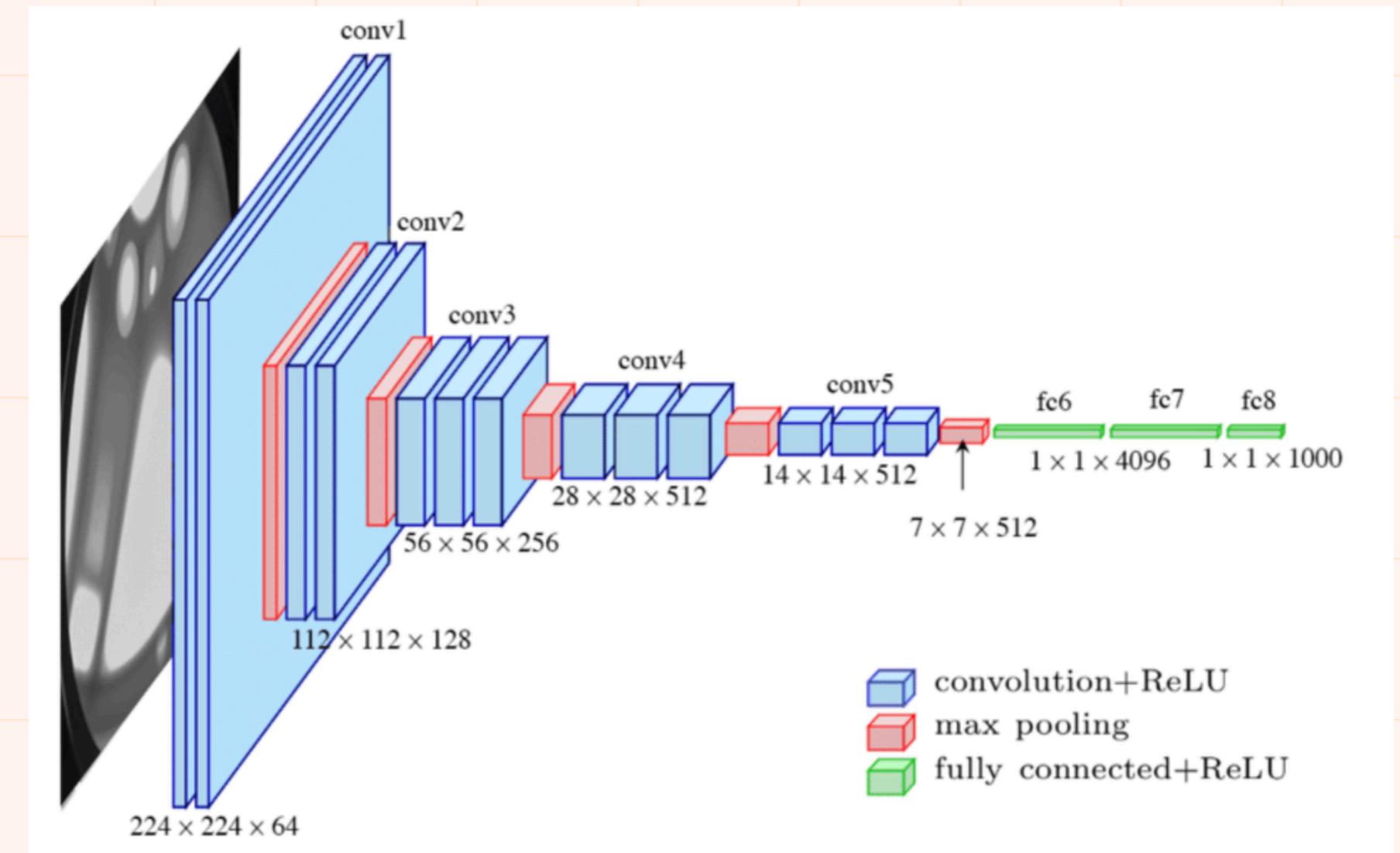
- Dùng các bộ lọc (kernel) nhỏ để trích xuất đặc trưng.
- Sinh ra feature map thể hiện các đặc trưng như cạnh, đường nét.
- Thường kết hợp với ReLU để thêm tính phi tuyến.

2. *Pooling Layer* (Lớp gộp)

- Giảm kích thước feature map → giảm tham số và chi phí tính toán.
- Tạo tính bất biến với dịch chuyển/giãn/nhỏ lệch.
- Hai loại phổ biến: Max Pooling, Average Pooling.

3. *Fully Connected Layer* (Lớp kết nối đầy đủ)

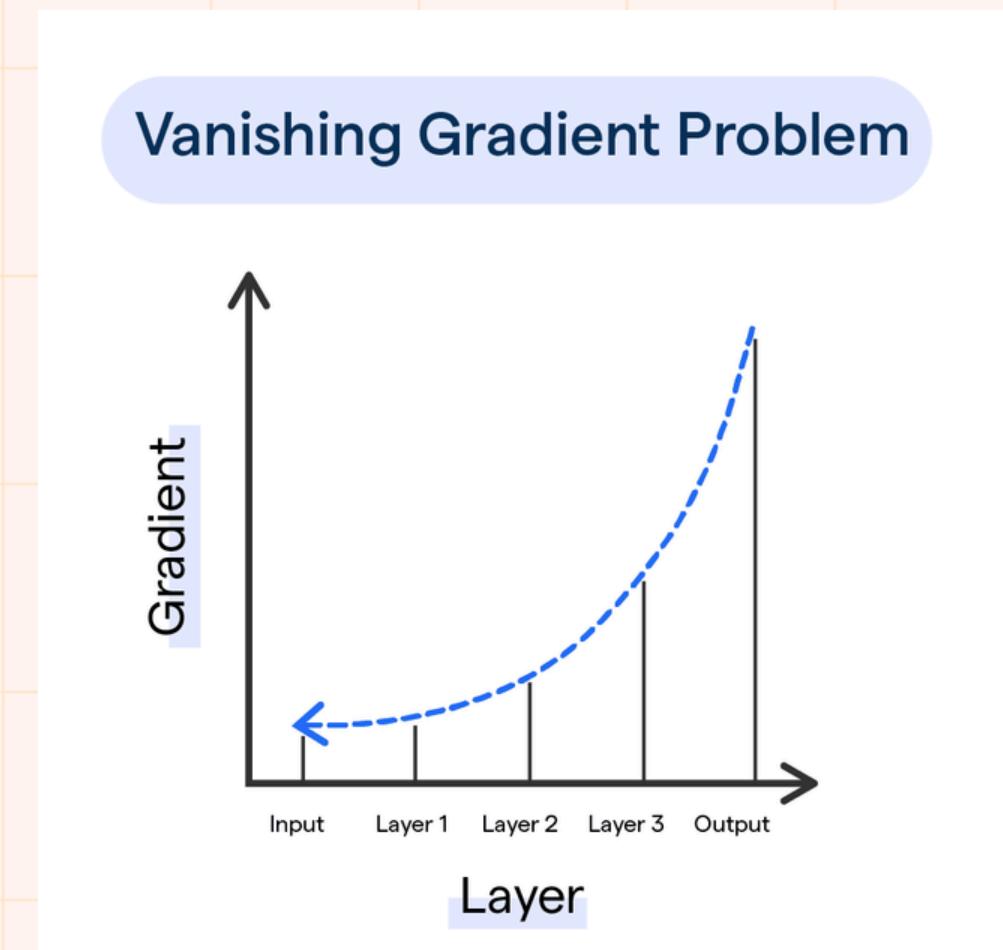
- Flatten feature maps thành vector.
- Thực hiện phân loại cuối cùng.
- Dùng Softmax để tạo xác suất cho từng lớp.



→ Các lớp tích chập và gộp ban đầu để học các đặc trưng (feature learning), theo sau là các lớp kết nối đầy đủ để phân loại (classification)

Mô hình ResNet

- Lý thuyết: các mô hình học sâu (Deep Learning) được hưởng lợi từ việc "sâu" hơn. Việc xếp chồng nhiều lớp (layers) cho phép mạng học được các đặc trưng (features) ở nhiều cấp độ trừu tượng khác nhau, từ các nét cạnh đơn giản (ở lớp đầu) đến các cấu trúc phức tạp (ở các lớp sau).
- Thực tế: Huấn luyện mạng rất sâu gặp hai vấn đề chính:
 - Suy giảm độ dốc (Vanishing Gradient): Gradient quá nhỏ ở các lớp đầu, khiến lớp đầu không học được.
 - Suy thoái (Degradation): Khi thêm nhiều lớp, hiệu suất huấn luyện bão hòa hoặc giảm, không phải do overfitting mà do khó tối ưu hóa.



Mô hình ResNet

- Ý tưởng cốt lõi của ResNet là thay vì hy vọng một vài lớp xếp chồng sẽ học được một ánh xạ phức tạp mong muốn $H(x)$, ResNet giả định rằng sẽ dễ dàng hơn nếu các lớp này chỉ học phần dư (residual) của ánh xạ đó.
- Cụ thể, nếu x là đầu vào của một khối (block), và $H(x)$ là ánh xạ lý tưởng mà khối đó cần học, ResNet sẽ định nghĩa lại nhiệm vụ:
 - Các lớp trong khối sẽ học một hàm $F(x)$.
 - Đầu ra của khối sẽ là: $H(x) = F(x) + x$

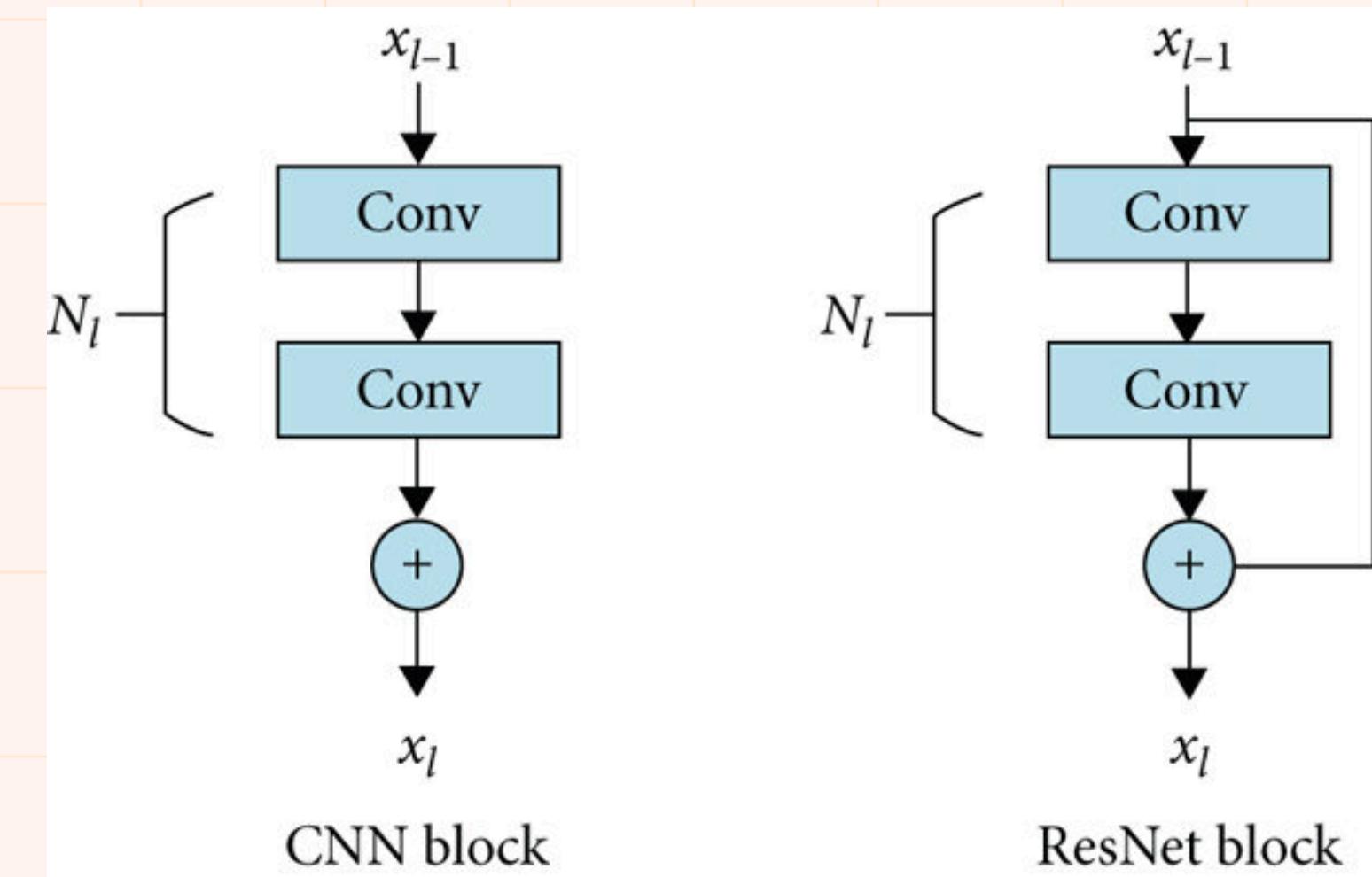
Công thức toán học của một Khối dư (Residual Block): Nếu x là đầu vào và y là đầu ra của khối, ta có: $y = F(x, \{W_i\}) + x$

Trong đó:

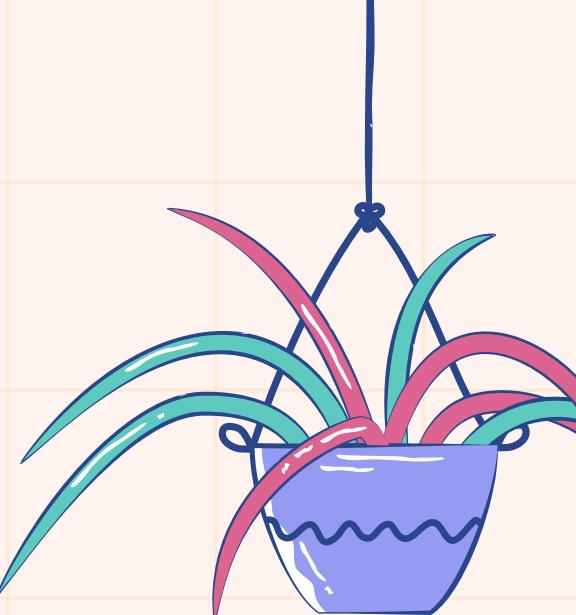
- x là vector đầu vào của khối.
- $F(x, \{W_i\})$ là hàm ánh xạ (thường bao gồm 2 hoặc 3 lớp tích chập với các trọng số W_i) mà các lớp trong khối cần học.
- Phép toán $+x$ chính là kết nối tắt thực hiện phép cộng theo từng phần tử.

→ Giải quyết suy thoái: Nếu lớp mới không học gì hữu ích, ResNet chỉ cần $F(x)=0 \rightarrow y=x$.

→ Giải quyết suy giảm độ dốc: Kết nối tắt tạo đường "cao tốc" cho gradient → gradient luôn tồn tại ở các lớp trước, cho phép huấn luyện mạng sâu (152 lớp hoặc hơn).



2. PHƯƠNG PHÁP XÂY DỰNG HỆ THỐNG



Chuẩn bị dữ liệu

1. Thu thập dữ liệu MNIST

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

```
# Load MNIST (data đang ở dạng PIL Image) PIL: ảnh 2D size (28, 28)

train_dataset = torchvision.datasets.MNIST(
    root="/content/drive/MyDrive/XLA",
    train=True,
    download=True,
    transform=train_transform
)
test_dataset = torchvision.datasets.MNIST(
    root="/content/drive/MyDrive/XLA",
    train=False,
    download=True,
    transform=test_transform
```



Chuẩn bị dữ liệu

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

- Tạo dữ liệu (Data Generation & Augmentation)

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

Chuẩn hóa kiểu MNIST (Hàm mnist_style_resize):

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

Tăng cường ngẫu nhiên (Hàm random_augment_mnist_style):

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

Chuẩn bị dữ liệu

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

- Tạo dữ liệu (Data Generation & Augmentation)

```
def mnist_style_resize(img, size=28, digit_size=20):
    # Chuyen anh sang grayscale
    img = img.convert("L")
    # Cat bo vung nen thua (bounding box)
    bbox = img.getbbox()
    if bbox:
        img = img.crop(bbox)
    # Resize giu ti le sao cho canh dai nhat = digit_size
    max_side = max(img.size)
    scale = digit_size / max_side
    new_size = tuple([int(x * scale) for x in img.size])
    img = img.resize(new_size, Image.LANCZOS)
    # Lam net va tang tuong phan
    img = img.filter(ImageFilter.UnsharpMask(radius=1, percent=150,
                                              threshold=3))
    enhancer = ImageEnhance.Contrast(img)
    img = enhancer.enhance(2.0)
    # Tao nen den 28x28 va dan vao giua
    new_img = Image.new("L", (size, size), 0)
    paste_x = (size - new_size[0]) // 2
    paste_y = (size - new_size[1]) // 2
    new_img.paste(img, (paste_x, paste_y))
    return new_img

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

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

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

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

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

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

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

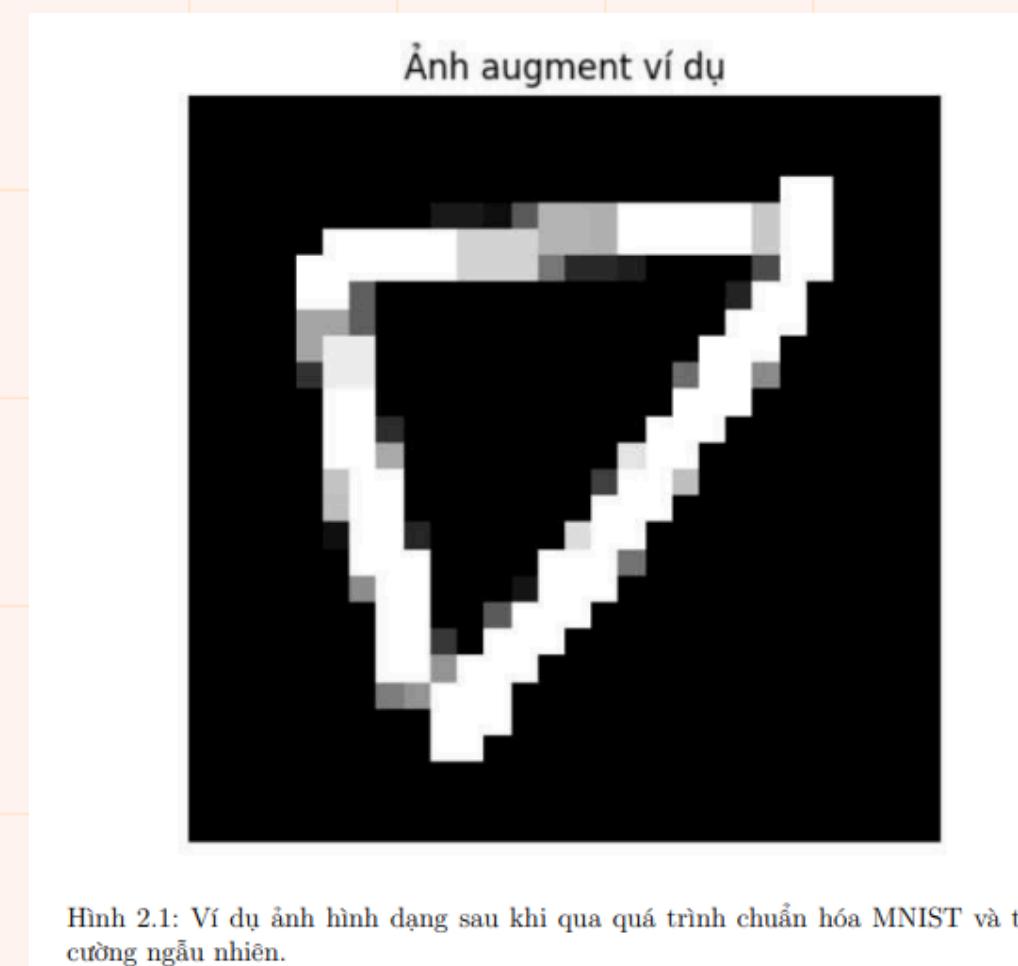
Chuẩn bị dữ liệu

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

- *Tải dữ liệu (Data Loading)*

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

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



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

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

    def __getitem__(self, idx):
        img_path = self.files[idx]
        img = Image.open(img_path).convert("L")
        # Áp dụng transform nếu có
        if self.transform:
            img = self.transform(img)
        return img, self.label
```

Listing 2.4: Lớp ShapeDataset tùy chỉnh

Chuẩn bị dữ liệu

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

- Tiền xử lý (Transforms)

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

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

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

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

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

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

Chuẩn bị dữ liệu

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

- Hợp nhất (Concatenation)

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

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

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

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

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

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

Chuẩn bị dữ liệu

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

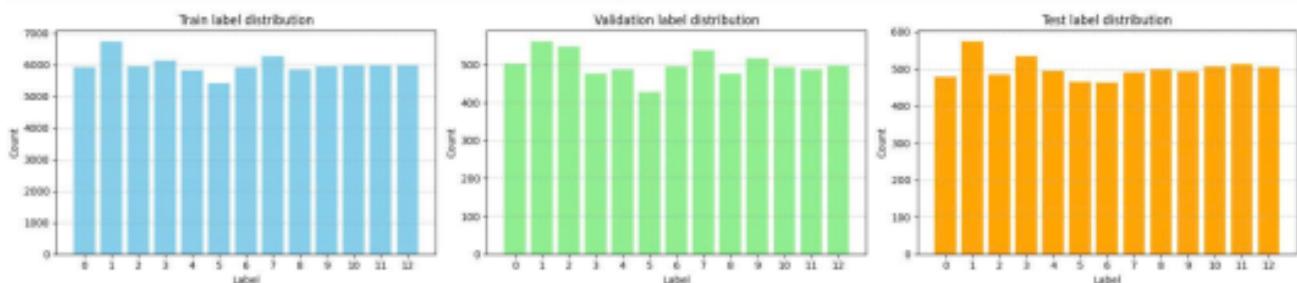
- Kiểm tra và Trực quan hóa Dữ liệu

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

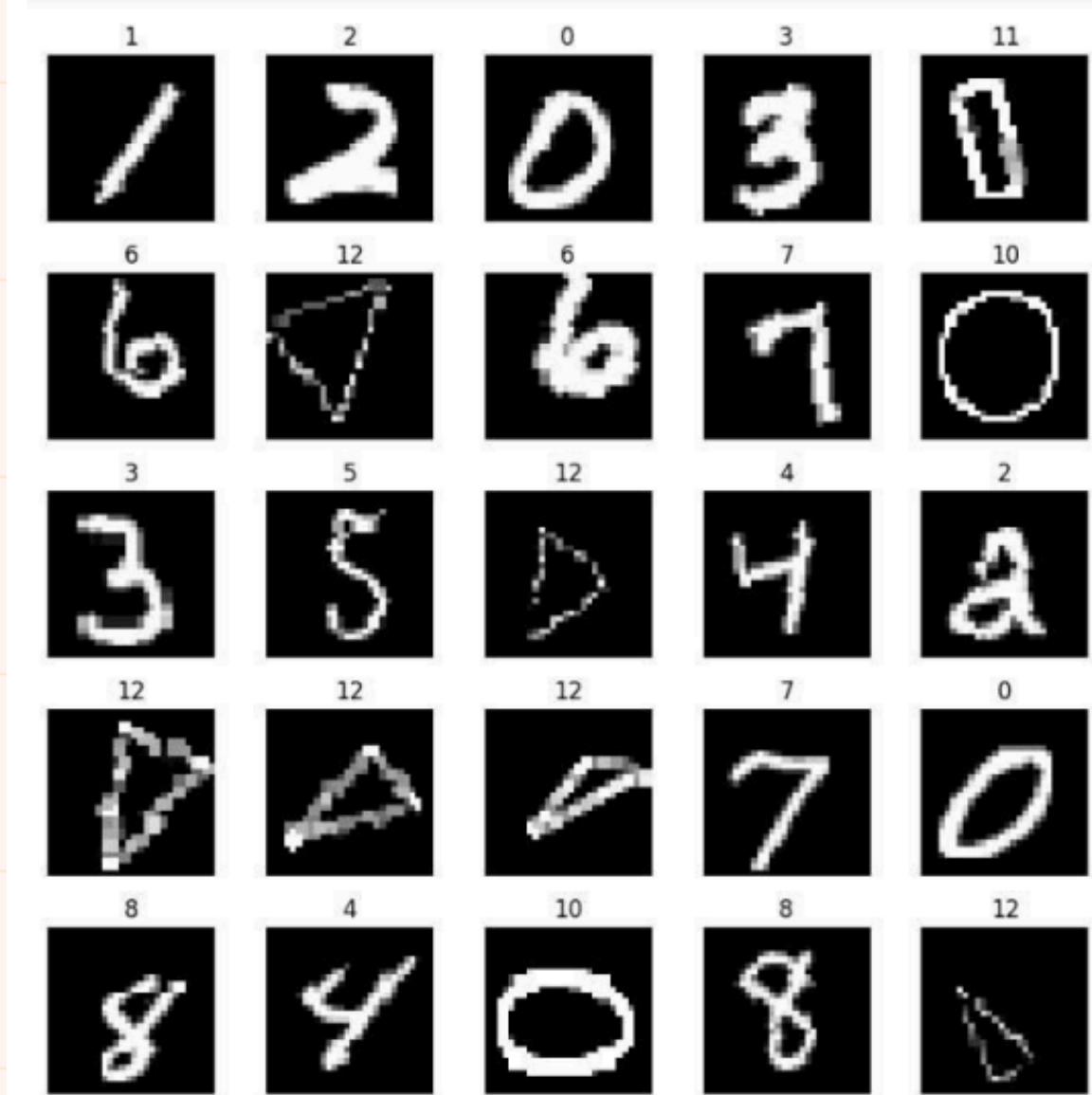
```
def get_all_labels(dataset):
    """Trích xuất tất cả nhãn từ dataset (ke ca ConcatDataset,
Subset)."""
    if hasattr(dataset, 'targets'):
        return np.array(dataset.targets)
    elif hasattr(dataset, 'datasets'): # Trường hợp ConcatDataset
        return np.concatenate([get_all_labels(d) for d in dataset.datasets])
    elif hasattr(dataset, 'dataset') and hasattr(dataset, 'indices'):
        # Trường hợp Subset
        all_targets = get_all_labels(dataset.dataset)
        return all_targets[dataset.indices]
    else:
        raise AttributeError("Dataset không có thuộc tính 'targets' hoặc 'datasets'.")
```

Lấy tất cả nhãn từ các tap

```
train_labels = get_all_labels(train_dataset_full)
val_labels = get_all_labels(val_dataset)
```



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



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

Chuẩn bị dữ liệu

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

- Tạo DataLoader

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

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

Listing 2.8: Khởi tạo DataLoaders

Thiết kế mô hình CNN dựa trên ResNet

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

1. Khối dư (ResidualBlock)

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

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

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

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

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

Listing 2.9: Định nghĩa ResidualBlock

Thiết kế mô hình CNN dựa trên ResNet

2. Kiến trúc mô hình ModernCNN

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

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

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

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

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

Listing 2.9: Định nghĩa ResidualBlock

Huấn luyện mô hình

1. Các tham số huấn luyện

- Hàm mất mát (Loss Function): nn.CrossEntropyLoss (tiêu chuẩn cho phân loại đa lớp).
- Hàm tối ưu (Optimizer): optim.Adam (tự điều chỉnh learning rate).
- Siêu tham số (Hyperparameters):
 - Learning Rate: 0.001
 - Số Epochs: 10
 - Batch Size: 128

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

Listing 2.11: Thiết lập hàm mất mát và tối ưu

Huấn luyện mô hình

2. Kỹ thuật tránh overfitting

Áp dụng đồng thời nhiều kỹ thuật chính quy hóa (regularization):

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

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

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

    # Lap lai train_loader 4 lan
    for aug_round in range(augment_times):
        progress_bar = tqdm(
            train_loader,
            desc=f"Epoch {epoch+1}/{num_epochs} | Aug {aug_round+1}/{augment_times}",
            leave=False
        )
        for data, target in progress_bar:
            data, target = data.to(device), target.to(device)
            outputs = model(data)
            loss = criterion(outputs, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        avg_loss = running_loss / ((progress_bar.n + 1) + aug_round * len(train_loader))
        progress_bar.set_postfix({"loss": f"{avg_loss:.4f}"})

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

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

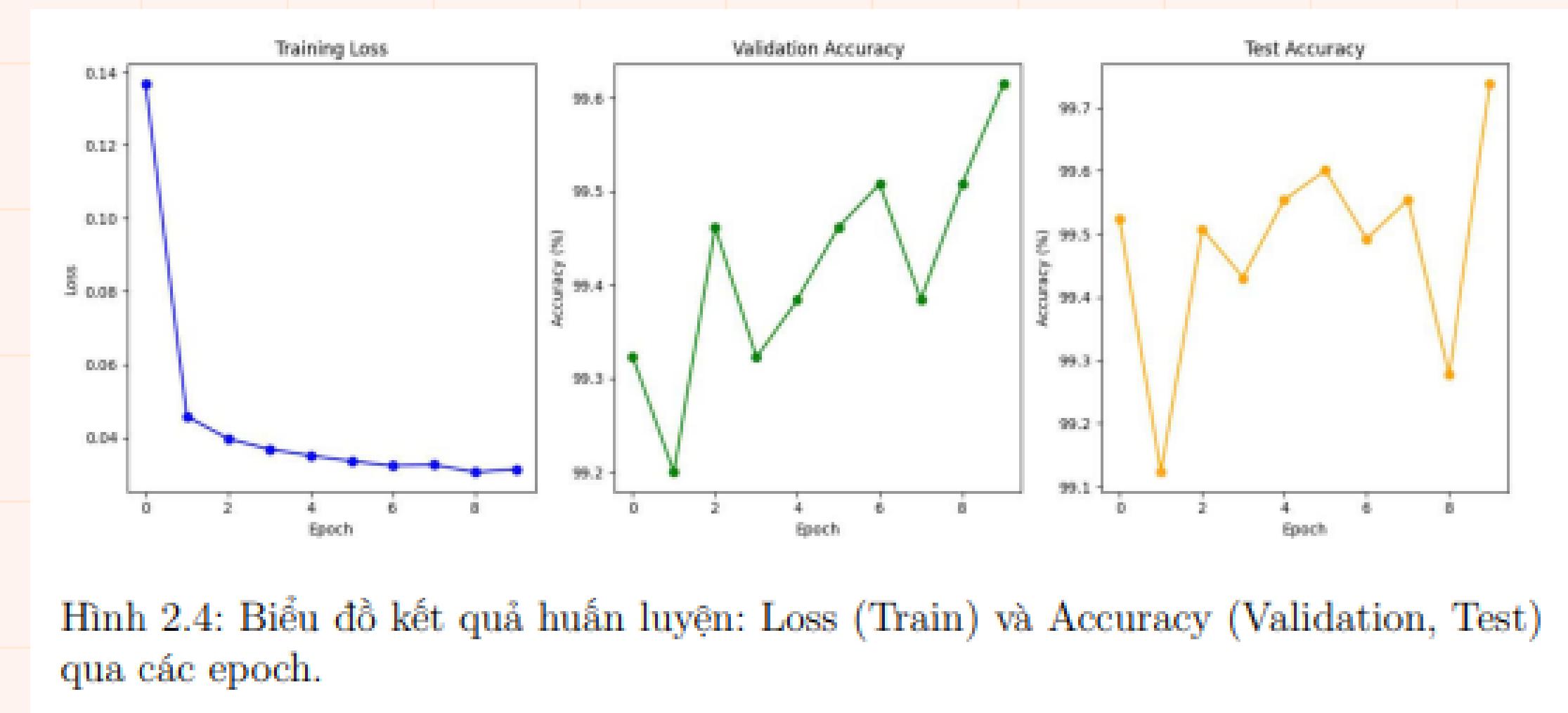
# ===== Danh gia tren tap test =====
(Tuong tu nhu validation...)
print(f"Test Accuracy: {test_acc:.2f}%")
```

Listing 2.12: Vòng lặp huấn luyện (training loop)

Đánh giá mô hình

1. Biểu đồ Huấn luyện (Training Plots)

- Chúng tôi theo dõi và trực quan hóa ba chỉ số chính qua các epoch:
- Training Loss: Cho thấy mô hình học dữ liệu huấn luyện tốt như thế nào.
- Validation Accuracy: Chỉ số chính để theo dõi hiệu suất trên dữ liệu mới và phát hiện overfitting.
- Test Accuracy: Kết quả cuối cùng của mô hình trên tập kiểm thử sau mỗi epoch



Đánh giá mô hình

2. Các chỉ số Phân loại (Classification Metrics)

Sử dụng các chỉ số tiêu chuẩn từ scikit-learn:

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

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

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

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

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

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

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

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

Đánh giá mô hình

2. Các chỉ số Phân loại (Classification Metrics)

Sử dụng các chỉ số tiêu chuẩn từ scikit-learn:

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

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

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

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

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

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

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

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

Quy trình xử lý ảnh đầu vào thực tế

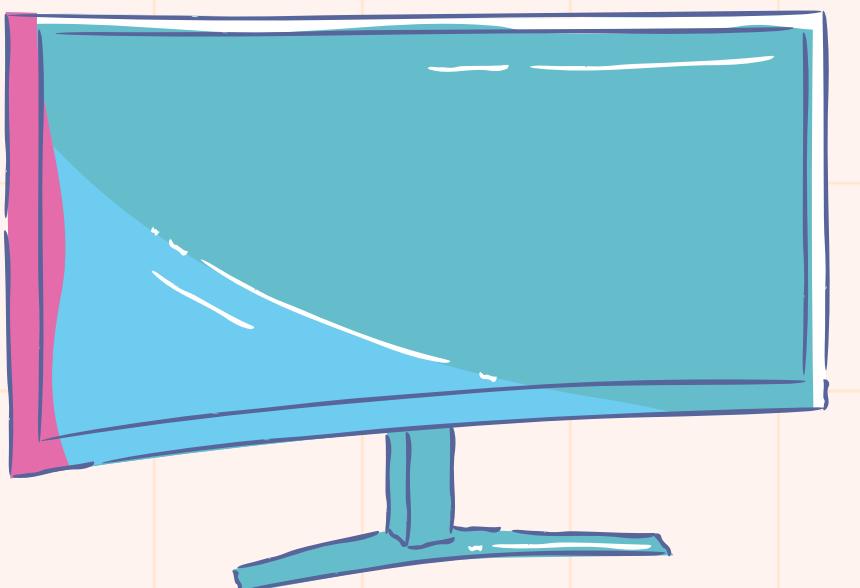
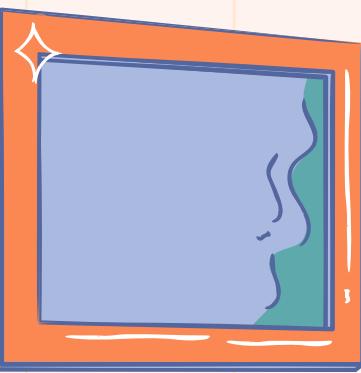
Để áp dụng mô hình vào ứng dụng thực tế, chúng tôi xây dựng quy trình tiền xử lý gồm 8 bước tuần tự, được trực quan hóa trên giao diện ứng dụng để minh bạch hóa quá trình “suy luận” của máy tính:

- Ảnh gốc: Thu nhận ảnh đầu vào và chuyển sang dạng mảng số.
- Grayscale: Chuyển ảnh sang thang xám để loại bỏ nhiễu màu.
- Gaussian Blur: Làm mờ nhẹ nhàng làm mịn nét vẽ tay.
- Thresholding: Tách đối tượng khỏi nền để tạo ảnh nhị phân.
- Contour Box: Xác định đường viền lớn nhất và vẽ bounding box bao quanh đối tượng.
- Cropping: Cắt ảnh theo bounding box, loại bỏ nền thừa.
- Resize (20px): Thu nhỏ ảnh sao cho cạnh dài nhất = 20px, giữ nguyên tỷ lệ.
- Tạo ảnh 28×28 : Đặt ảnh đã resize vào giữa nền đen 28×28 để phù hợp đầu vào ModernCNN.



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

3. THỰC NGHIỆM VÀ KẾT QUẢ



Kết quả nhận diện MNIST

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

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

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

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

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

Kết quả nhận diện hình dạng viết tay

1. Độ chính xác và các chỉ số đánh giá

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

- Độ chính xác tổng thể (Accuracy) trên 3 lớp này đạt 92.6% (tính theo weighted avg).
- Phân tích chi tiết các chỉ số cho thấy một số điểm mất cân bằng:
 - Lớp 12 (Triangle): Đạt chỉ số Recall rất cao (0.992), nghĩa là mô hình gần như không bỏ sót bất kỳ hình tam giác nào. Tuy nhiên, chỉ số Precision lại thấp nhất (0.842), cho thấy mô hình có xu hướng "quá tự tin" và đoán nhầm nhiều hình khác (Circle, Rectangle) thành Triangle.
 - Lớp 10 (Circle) và 11 (Rectangle): Có chỉ số Precision cao (0.978 và 0.956) nhưng Recall lại thấp hơn (0.878 và 0.856). Điều này có nghĩa là khi mô hình dự đoán là Circle/Rectangle, nó thường là đúng; nhưng nó đã bỏ sót (misclassified) một lượng đáng kể Circle/Rectangle sang các lớp khác.

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

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