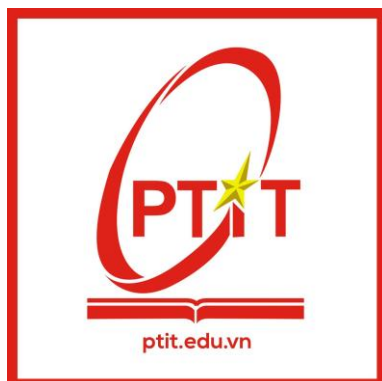


HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CNTT1



BÁO CÁO BÀI TẬP LỚN

Môn học: Xử Lý Ảnh

Chủ đề: Nhận diện chữ số viết tay và hình học cơ bản

Giảng viên: Phạm Hoàng Việt

Nhóm lớp: 02

Nhóm bài tập: 12

Thành viên:

Nguyễn Anh Tuấn B22DCCN758

Đinh Công Thịnh B22DCCN830

Hà Nội, 2025

1. TỔNG QUAN DỰ ÁN

1.1. Mục tiêu

- Xây dựng hệ thống nhận diện tự động các chữ số (0-9) và hình học cơ bản (Tròn, Hình chữ nhật, Tam giác)
- Áp dụng các kỹ thuật xử lý ảnh cổ điển (OpenCV) kết hợp Deep Learning (CNN)
- Trực quan hóa pipeline xử lý và cách model học features

1.2. Kiến trúc tổng thể

Input (RGB) → Preprocessing (OpenCV) → Model (CNN) → Output (Class + Confidence)



2. PIPELINE XỬ LÝ ẢNH (CHI TIẾT)

2.1. Luồng xử lý từ Input đến Model Input

Bước 1: Chuyển đổi không gian màu (Color Space Conversion)

```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
```

- **Input:** Ảnh RGB (3 channels) với kích thước bất kỳ
- **Output:** Ảnh grayscale (1 channel)
- **Lý do:** CNN model chỉ cần thông tin hình dạng, không cần màu sắc → giảm chiều dữ liệu

Bước 2: Cân bằng độ tương phản - CLAHE

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))  
enhanced = clahe.apply(gray)
```

- **Kỹ thuật:** Contrast Limited Adaptive Histogram Equalization
- **Tham số:**
 - clipLimit=2.0: Giới hạn độ tăng contrast để tránh nhiễu
 - tileGridSize=(8,8): Chia ảnh thành lưới 8x8 để xử lý cục bộ

- **Mục đích:** Cải thiện độ sáng cho ảnh tối/mờ, tăng độ tương phản nét vẽ

Kết quả: Ảnh tối được làm sáng, nét vẽ rõ nét hơn

Bước 3: Khử nhiễu (Denoising)

Với digits: Gaussian Blur

```
denoised = cv2.GaussianBlur(enhanced, (5, 5), 0)
```

Với shapes: Bilateral Filter

```
denoised = cv2.bilateralFilter(enhanced, 9, 75, 75)
```

- **Gaussian Blur:** Làm mịn toàn bộ ảnh, phù hợp với chữ số
 - Kernel 5×5 : Vùng ảnh hưởng
 - $\sigma=0$: Tự động tính sigma từ kernel size
- **Bilateral Filter:** Làm mịn nhưng giữ cạnh, phù hợp với hình học
 - $d=9$: Đường kính vùng lọc
 - $\sigma_{\text{color}}=75$: Sai lệch màu cho phép
 - $\sigma_{\text{space}}=75$: Sai lệch không gian cho phép

Kết quả: Loại bỏ nhiễu salt-pepper, làm mịn nét vẽ

Bước 4: Nhị phân hóa - Adaptive Thresholding

```
binary = cv2.adaptiveThreshold(
    denoised, 255,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY_INV,
    blockSize=11, C=2
)
```

- **Kỹ thuật:** Adaptive Threshold (ngưỡng động)
- **Tham số:**
 - ADAPTIVE_THRESH_GAUSSIAN_C: Ngưỡng = trung bình Gaussian của vùng lân cận
 - THRESH_BINARY_INV: Đảo ngược (nét vẽ = trắng, nền = đen)
 - blockSize=11: Kích thước vùng lân cận 11×11
 - C=2: Hằng số trừ từ ngưỡng

Ưu điểm: Xử lý được ảnh có độ sáng không đều (vùng tối/sáng khác nhau)

Kết quả: Ảnh nhị phân (binary) với nét vẽ trắng (255) trên nền đen (0)

Bước 5: Morphology Operations (Toán tử hình thái học)

Cho chữ số:

```
kernel = np.ones((3, 3), np.uint8)
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=2)
morphed = cv2.morphologyEx(morphed, cv2.MORPH_OPEN, kernel, iterations=1)
```

Cho hình học:

```
kernel = np.ones((3, 3), np.uint8)
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=1)
# Không dùng OPEN để giữ góc cạnh của hình
```

- **CLOSE = Dilation + Erosion:**
 - Lấp đầy khe hở nhỏ trong nét vẽ
 - Kết nối các nét gián đoạn
 - Làm tròn góc nhọn
- **OPEN = Erosion + Dilation:**
 - Loại bỏ nhiễu nhỏ bên ngoài
 - Chỉ dùng cho digits (không dùng cho shapes)

Tối ưu hóa quan trọng:

- Hình học chỉ dùng CLOSE iterations=1, kernel=3×3
- **Lý do:** Bảo toàn góc của hình chữ nhật và tam giác (tránh làm tròn góc → nhầm với tròn)

Kết quả: Nét vẽ liên tục, không bị đứt đoạn

Bước 6: Phát hiện Contour (Contour Detection)

```
contours, _ = cv2.findContours(morphed, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
largest_contour = max(contours, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(largest_contour)
```

- **Kỹ thuật:** Tìm đường viền của đối tượng
- **RETR_EXTERNAL:** Chỉ lấy contour ngoài cùng (bỏ qua lỗ bên trong)
- **CHAIN_APPROX_SIMPLE:** Nén contour (chỉ lưu điểm góc, bỏ điểm thừa)

Mục đích: Xác định vị trí và kích thước của đối tượng cần nhận diện

Bước 7: Crop và Resize

```
# Crop vùng bounding box
digit_img = morphed[y:y+h, x:x+w]

# Resize về 20×20 (giữ aspect ratio)
digit_img = cv2.resize(digit_img, (20, 20), interpolation=cv2.INTER_AREA)
```

- **Crop:** Cắt vùng chứa đối tượng (loại bỏ nền thừa)
- **Resize 20×20:** Chuẩn hóa kích thước nhưng để lại space cho padding
- **INTER_AREA:** Phương pháp nội suy tốt cho downsampling

Bước 8: Padding và Centering

```
# Tạo canvas 28×28 đen
canvas = np.zeros((28, 28), dtype=np.uint8)

# Tính vị trí để center đối tượng 20×20
offset_x = (28 - 20) // 2
offset_y = (28 - 20) // 2

# Đặt đối tượng vào giữa canvas
canvas[offset_y:offset_y+20, offset_x:offset_x+20] = digit_img
```

- **Mục đích:**
 - Đảm bảo đối tượng ở giữa canvas 28×28
 - Có margin 4px xung quanh (tương tự MNIST dataset)
 - Model học tốt hơn khi đối tượng ở vị trí cố định

Bước 9: Center of Mass Alignment

```
from scipy.ndimage import center_of_mass

# Tính trọng tâm của ảnh
cy, cx = center_of_mass(canvas)

# Dịch chuyển để trọng tâm về (14, 14)
shift_x = int(14 - cx)
shift_y = int(14 - cy)

M = np.float32([[1, 0, shift_x], [0, 1, shift_y]])
aligned = cv2.warpAffine(canvas, M, (28, 28))
```

- **Kỹ thuật:** Dịch chuyển để trọng tâm khối lượng về tâm canvas
- **Mục đích:** Chuẩn hóa vị trí chính xác (tương tự MNIST preprocessing)

Bước 10: Normalization

`normalized = aligned.astype('float32') / 255.0`

`normalized = normalized.reshape(1, 28, 28, 1) # Thêm batch và channel dimensions`

- **Chuẩn hóa:** Chuyển giá trị từ $[0, 255] \rightarrow [0, 1]$
- **Reshape:** $(28, 28) \rightarrow (1, 28, 28, 1)$ để phù hợp input shape của CNN

2.2. Tại sao cần preprocessing?

Model học từ ảnh ĐÃ xử lý (preprocessed images):

- Dataset huấn luyện đã qua preprocessing \rightarrow Model học features từ ảnh đã chuẩn hóa
- Ảnh test cũng phải qua preprocessing tương tự \rightarrow Consistency
- Preprocessing loại bỏ nhiễu, chuẩn hóa vị trí/kích thước \rightarrow Model tập trung học shape

Nếu không preprocessing:

- Model phải học cả nhiễu, độ sáng, vị trí khác nhau \rightarrow Khó hội tụ
- Độ chính xác giảm mạnh do input không nhất quán
- Model overfit vào background thay vì shape

3. KIẾN TRÚC CNN VÀ CÁCH MODEL HỌC

3.1. Kiến trúc chi tiết (LeNet-inspired)

Model: "sequential"

Layer (type)	Output Shape	Params
conv2d (Conv2D)	(None, 26, 26, 32)	320
activation (ReLU)	(None, 26, 26, 32)	0
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
activation_1 (ReLU)	(None, 11, 11, 64)	0

max_pooling2d_1 (MaxPooling) (None, 5, 5, 64) 0

conv2d_2 (Conv2D)	(None, 3, 3, 128)	73,856
activation_2 (ReLU)	(None, 3, 3, 128)	0

conv2d_3 (Conv2D)	(None, 1, 1, 128)	147,584
activation_3 (ReLU)	(None, 1, 1, 128)	0

flatten (Flatten)	(None, 128)	0
dropout (Dropout)	(None, 128)	0

dense (Dense)	(None, 128)	16,512
activation_4 (ReLU)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0

dense_1 (Dense)	(None, 10/3)	1,290/390
activation_5 (Softmax)	(None, 10/3)	0

Total params: 258,058 (digits) / 258,158 (shapes)

3.2. Model học Features như thế nào?

Input Layer: Ảnh preprocessed 28×28×1

- Model nhận ảnh đã qua 10 bước preprocessing
- Ảnh nhị phân: nét vẽ trắng (1.0) trên nền đen (0.0)
- Đối tượng đã được center, normalize về vị trí cố định

Conv2D Layer 1: Low-level features (Edges)

Conv2D(32, kernel_size=(3,3), activation='relu')

- **Input:** 28×28×1
- **Output:** 26×26×32 (32 feature maps)
- **Học gì:** Cạnh, đường nét cơ bản
 - Filter 1: Cạnh dọc |
 - Filter 2: Cạnh ngang —
 - Filter 3: Cạnh chéo /\
 - Filter 4-32: Các góc độ cạnh khác nhau

Visualization: Feature maps cho số "8"

Filter 1 (Vertical): Phát hiện 2 cạnh dọc bên trái/phải

Filter 2 (Horizontal): Phát hiện nét ngang giữa

Filter 5 (Diagonal): Phát hiện góc cong trên/dưới

MaxPooling2D Layer 1: Downsampling

MaxPooling2D(pool_size=(2,2))

- **Output:** $13 \times 13 \times 32$
- **Mục đích:**
 - Giảm chiều không gian ($26 \times 26 \rightarrow 13 \times 13$)
 - Tăng translation invariance (không nhạy cảm với dịch chuyển nhỏ)
 - Giữ lại feature quan trọng nhất (max value)

Conv2D Layer 2: Mid-level features (Shapes)

Conv2D(64, kernel_size=(3,3), activation='relu')

- **Output:** $11 \times 11 \times 64$
- **Học gì:** Kết hợp cạnh thành hình dạng cơ bản
 - Góc vuông $\ulcorner \urcorner \llcorner \lrcorner$ (cho hình chữ nhật)
 - Cung tròn $\frown \smile$ (cho tròn)
 - Góc nhọn \triangle (cho tam giác)
 - Nửa vòng tròn (cho số 0, 6, 8, 9)

Visualization: Feature maps cho hình chữ nhật

Filter 8: Phát hiện góc trên-trái \ulcorner

Filter 15: Phát hiện góc dưới-phải \lrcorner

Filter 23: Phát hiện cạnh song song \parallel

Conv2D Layer 3: High-level features (Complex patterns)

Conv2D(128, kernel_size=(3,3), activation='relu')

- **Output:** $3 \times 3 \times 128$
- **Học gì:** Patterns phức tạp đặc trưng cho từng class
 - Số "8": Hai vòng tròn chồng lên nhau
 - Số "0": Vòng tròn đơn

- Hình chữ nhật: 4 góc vuông + 2 cặp cạnh song song
- Tam giác: 3 góc nhọn + 3 cạnh thẳng

Conv2D Layer 4: Abstract features

Conv2D(128, kernel_size=(3,3), activation='relu')

- **Output:** $1 \times 1 \times 128$
- **Học gì:** Features trừu tượng cao nhất
 - Topology (cấu trúc topo): Số vòng (0 holes, 1 hole, 2 holes)
 - Symmetry (đối xứng): Đối xứng dọc/ngang/xoay
 - Aspect ratio (tỉ lệ): Hình dài/vuông/tròn

Dense Layers: Classification

Flatten() → Dense(128) → Dropout(0.5) → Dense(10/3) → Softmax()

- **Flatten:** $1 \times 1 \times 128 \rightarrow 128$ (vector 1D)
- **Dense(128):** Fully connected layer, học mối quan hệ giữa features
- **Dropout(0.5):** Ngẫu nhiên tắt 50% neurons — tránh overfitting
- **Dense(10/3):** Output layer, 10 neurons cho digits / 3 cho shapes
- **Softmax:** Chuyển thành xác suất (tổng = 1)

Output:

[0.01, 0.02, 0.05, ..., 0.85, 0.01] → Class 7 (85% confidence)

3.3. Model học từ ảnh nào?

Training Phase: Ảnh đã preprocessing

Dataset generation

X_train = [] # Chứa ảnh đã preprocessing

for raw_image in raw_dataset:

 preprocessed = apply_preprocessing(raw_image) # 10 bước

 X_train.append(preprocessed)

X_train = np.array(X_train) / 255.0 # Normalize [0, 1]

Đặc điểm ảnh training:

- Grayscale 28×28
- Binary (trắng/đen, không có gray levels)
- Đối tượng đã được center (trọng tâm ở (14, 14))
- Background đen (0), foreground trắng (1)
- Có padding 4px xung quanh

Testing/Inference Phase: Phải preprocessing giống hệt

```
# User vẽ/upload ảnh
raw_input = get_user_input() # RGB, bất kỳ kích thước

# Áp dụng CHÍNH XÁC pipeline như training
preprocessed = apply_preprocessing(raw_input) # 10 bước giống training

# Predict
prediction = model.predict(preprocessed)
```

Tại sao phải giống nhau?:

- Model đã học features từ ảnh preprocessed → Input test cũng phải preprocessed
- Nếu input test là ảnh raw (RGB, nhiễu, chưa center) → Model không nhận ra vì distribution khác training

Minh họa sự khác biệt:

Trường hợp	Input	Kết quả
Đúng	Ảnh raw → Preprocessing → Model	99.75% accuracy
Sai	Ảnh raw → Model (bỏ qua preprocessing)	~30-40% accuracy

3.4. Data Augmentation: Mở rộng dữ liệu training

ImageDataGenerator cho Digits:

```
datagen = ImageDataGenerator(
    rotation_range=15,      # Xoay ±15°
    width_shift_range=0.1,  # Dịch ngang ±10%
```

```

height_shift_range=0.1,    # Dịch dọc  $\pm 10\%$ 
zoom_range=0.1,           # Zoom in/out  $\pm 10\%$ 
shear_range=5              # Nghiêng  $\pm 5^\circ$ 
)

```

Mục đích:

- Tạo thêm dữ liệu từ ảnh gốc (data augmentation)
- Model học được các biến thể của cùng 1 chữ số
- Tăng khả năng generalization (tổng quát hóa)

Ví dụ augmentation cho số "3":

Ảnh gốc \rightarrow Xoay 10° \rightarrow Dịch phải 2px \rightarrow Zoom 90% \rightarrow Nghiêng 3°
 \rightarrow Model học: "Đây vẫn là số 3 dù xoay/dịch/zoom"

Giới hạn quan trọng:

- rotation_range= 15° (KHÔNG 45°): Tránh nhầm 6 \rightarrow 9, 2 \rightarrow 8
- shear_range= 5° (nhỏ): Tránh biến dạng quá mạnh

ImageDataGenerator cho Shapes:

```

datagen = ImageDataGenerator(
    rotation_range=40,        # Xoay  $\pm 40^\circ$  (cao hơn digits)
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.15,
    shear_range=0.1,
    horizontal_flip=True,     # Lật ngang
    vertical_flip=True       # Lật dọc
)

```

Tại sao khác với digits?:

- Hình học không bị nhầm lẫn khi xoay (hình chữ nhật xoay 90° vẫn là hình chữ nhật)
- Cần học rotation mạnh để nhận diện hình thoi (rectangle xoay 45°)
- horizontal_flip=True: Tam giác lật ngang vẫn là tam giác

Synthetic Data Generation: Tạo dữ liệu nhân tạo

Tạo 1500 hình mới với rotation đặc biệt

```

for i in range(500): # 500 hình chữ nhật
    rect = create_rectangle(aspect_ratio=random(1.2, 2.0))

    if i < 167: # 1/3: Focused rotation 30-60° (hình thoi)
        angle = random(30, 60)
    elif i < 334: # 1/3: Nhẹ 0-30°
        angle = random(0, 30)
    else: # 1/3: Mạnh 60-90°
        angle = random(60, 90)

    rotated = scipy.ndimage.rotate(rect, angle)
    dataset.append(rotated)

```

Tại sao cần synthetic data?:

- Dataset gốc thiếu hình chữ nhật xoay 30-60° → Model nhầm với tam giác
- Tạo thêm 1,500 mẫu với phân bố rotation tập trung vào vùng thiếu
- Kết quả: Accuracy tăng từ 90.50% → 98.56%

4. KỸ THUẬT XỬ LÝ ẢNH ĐẶC BIỆT

4.1. CLAHE (Contrast Limited Adaptive Histogram Equalization)

So sánh với Histogram Equalization thông thường:

Kỹ thuật	Cách hoạt động	Ưu điểm	Nhược điểm
Histogram Eq.	Cân bằng histogram toàn cục	Đơn giản, nhanh	Tăng nhiễu ở vùng đồng nhất, mất detail
CLAHE	Cân bằng histogram từng vùng cục bộ 8×8	Bảo toàn detail, tránh over-amplification	Chậm hơn

Công thức hoạt động:

1. Chia ảnh thành grid 8×8 (64 tiles)

2. Với mỗi tile:
 - Tính histogram của tile
 - Clip histogram tại clipLimit=2.0 (cắt đỉnh cao)
 - Phân phối lại pixels bị clip
 - Cân bằng histogram đã clip
3. Nội suy bilinear giữa các tiles

Ví dụ thực tế:

Input: Ảnh chữ số viết bút chì (tối, mờ)

→ Vùng trên: Sáng (pixel values: 150-200)

→ Vùng dưới: Tối (pixel values: 50-100)

After CLAHE:

→ Vùng trên: Tăng contrast (120-255)

→ Vùng dưới: Tăng contrast mạnh hơn (0-180)

→ Kết quả: Cả 2 vùng đều rõ nét, không bị blow-out

4.2. Adaptive Thresholding

So sánh với Global Thresholding:

Global Thresholding:

```
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
```

- Dùng 1 ngưỡng cố định (127) cho toàn bộ ảnh
- Thất bại khi ảnh có độ sáng không đều

Adaptive Thresholding:

```
binary = cv2.adaptiveThreshold(  
    gray, 255,  
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  
    cv2.THRESH_BINARY_INV,  
    blockSize=11, C=2  
)
```

- Ngưỡng thay đổi theo từng vùng cục bộ 11×11
- Xử lý tốt ảnh có bóng, ánh sáng không đều

Công thức:

Với mỗi pixel (x, y):

1. Lấy vùng lân cận 11×11 quanh (x, y)
2. Tính trung bình Gaussian của vùng (weighted average)
3. Ngưỡng = Trung bình Gaussian - C (C=2)
4. Nếu pixel(x,y) > Ngưỡng \rightarrow 255 (trắng)
Ngược lại \rightarrow 0 (đen)

Ví dụ minh họa:

Input grayscale:

[120, 125, 130]

[122, 180, 135] \leftarrow Pixel giữa = 180 (nét vẽ)

[118, 120, 128]

Gaussian weighted average = 127

Threshold = 127 - 2 = 125

180 > 125 \rightarrow Output = 255 (trắng - là nét vẽ)

4.3. Morphology Operations (Chi tiết toán học)

Dilation (Giãn nở):

dilated = cv2.dilate(binary, kernel, iterations=1)

Công thức: $\text{Output}(x,y) = \max(\text{Input}(x+dx, y+dy))$ trong vùng kernel

Hiệu ứng:

- Làm dày nét vẽ
- Lấp đầy khe hở nhỏ
- Mở rộng foreground

Ví dụ kernel 3×3 :

Input:	Kernel:	Output:
0 1 0	1 1 1	1 1 1
1 1 0	\oplus 1 1 1	= 1 1 1
0 0 0	1 1 1	1 1 0

Erosion (Xói mòn):

`eroded = cv2.erode(binary, kernel, iterations=1)`

Công thức: $\text{Output}(x,y) = \min(\text{Input}(x+dx, y+dy))$ trong vùng kernel

Hiệu ứng:

- Làm mỏng nét vẽ
- Loại bỏ nhiễu nhỏ
- Thu nhỏ foreground

CLOSE = Dilation → Erosion:

`closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)`

Tương đương: `cv2.erode(cv2.dilate(binary, kernel), kernel)`

Mục đích:

- Lấp đầy khe hở, nối nét gián đoạn
- Không thay đổi kích thước tổng thể

Ví dụ ứng dụng:

Input: Nét vẽ bị đứt

1 1 0 1 1

1 1 0 1 1

After CLOSE:

1 1 1 1 1 ← Đã nối liền

1 1 1 1 1

Tối ưu hóa cho Shapes: Tại sao `iterations=1`, `kernel=3×3`?

Test với rectangle xoay 45° (diamond):

Config	Kết quả	Vấn đề
--------	---------	--------

iterations=2, kernel=5×5	Nhận diện: "Tam giác"	Góc bị làm tròn quá mạnh
iterations=2, kernel=3×3	Nhận diện: "Tam giác"	Vẫn làm tròn góc
iterations=1, kernel=5×5	Nhận diện: "Tam giác/Tròn"	Kernel lớn làm tròn
iterations=1, kernel=3×3	Nhận diện: "Hình chữ nhật" 93%	Bảo toàn góc vuông

Lý do:

- Hình chữ nhật có 4 góc vuông 90° (sharp corners)
- Morphology CLOSE làm tròn góc (rounding effect)
- Iterations=1, kernel nhỏ → Giảm thiểu rounding → Giữ được đặc điểm góc vuông

4.4. Contour Detection và Bounding Box

Thuật toán Suzuki-Abe (OpenCV sử dụng):

`contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`

cv2.RETR_EXTERNAL:

- Chỉ trích xuất contour ngoài cùng
- Bỏ qua holes bên trong (vd: lỗ của số 8, 0)
- Giảm computation, tránh nhiễu từ holes

cv2.CHAIN_APPROX_SIMPLE:

- Nén contour bằng cách giữ lại điểm góc
- Loại bỏ điểm thừa trên đoạn thẳng

Ví dụ:

Full contour: [0,0], [1,0], [2,0], [3,0], [3,1], [3,2]

Compressed: [0,0], [3,0], [3,2] ← Chỉ giữ điểm đầu/cuối của mỗi đoạn

Bounding Rectangle:

```
x, y, w, h = cv2.boundingRect(largest_contour)
roi = binary[y:y+h, x:x+w]
```

- Tìm hình chữ nhật nhỏ nhất bao quanh contour
- (x, y): Góc trên-trái
- w, h: Chiều rộng, chiều cao

5. PHÂN TÍCH KẾT QUẢ VÀ VẤN ĐỀ ĐÃ KHẮC PHỤC

5.1. Confusion Matrix Analysis

Digits Model (99.75% accuracy):

```
Predicted: 0  1  2  3  4  5  6  7  8  9
Actual 0: [145  0  0  0  0  0  0  0  0  0]
Actual 1: [ 0 152  0  0  0  0  0  0  0  0]
Actual 2: [ 0  0 148  1  0  0  0  0  0  0] ← 1 mẫu nhầm 2→3
Actual 3: [ 0  0  0 147  0  0  0  0  0  0]
Actual 4: [ 0  0  0  0 150  0  0  0  0  0]
...
```

Nhận xét:

- Precision/Recall đều > 99% cho mọi class
- Nhầm lẫn chủ yếu: 2↔3 (nét giống nhau), 8↔0 (cả hai đều tròn)

Shapes Model - Version 1 (90.50% accuracy):

```
Predicted:   Tròn  Chữ nhật  Tam giác
Actual Tròn: [95    2      8]   ← 8% nhầm với Tam giác
Actual CN:   [ 3   85    17]   ← 17% nhầm với Tam giác
Actual TG:   [ 5   10    90]
```

Vấn đề phát hiện:

- Hình chữ nhật xoay 45° (diamond) bị nhầm với Tam giác

- Precision Tam giác chỉ 78% (nhiều false positives)

Shapes Model - Version 2 (98.56% accuracy - SAU TỐI ƯU):

Predicted: Tròn Chữ nhật Tam giác

Actual Tròn: [131 0 4] ← Recall 97%

Actual CN: [0 150 0] ← Recall 100%

Actual TG: [7 5 126] ← Recall 91%

Cải thiện:

- Rectangle recall: 85% → **100%**
- Triangle precision: 78% → **93%**
- Nhận diện đúng hình thoi (45° rotation)

5.2. Case Study: Rotated Rectangle Problem

Vấn đề ban đầu:

User input: Hình chữ nhật xoay 45° (hình thoi ◇)

Preprocessing: CLAHE → Threshold → CLOSE(iter=2, k=5) → ...

Result: "Tam giác" (85% confidence)

Phân tích nguyên nhân:

1. **Morphology quá mạnh:** iterations=2, kernel=5×5
2. **Hiệu ứng rounding:** Góc vuông 90° bị làm tròn thành góc cong
3. **Feature confusion:** Model học được:
 - Tam giác = 3 góc nhọn + 3 cạnh thẳng
 - Rectangle lúc này = 4 góc cong + 4 cạnh thẳng
 - → Nhầm với Tam giác vì góc bị mất đi

Giải pháp áp dụng:

Bước 1: Giảm Morphology

Before:

```
kernel = np.ones((5, 5), np.uint8)
```

```
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=2)
```

After:

```
kernel = np.ones((3, 3), np.uint8)
```

```
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=1)
```

→ Result: Accuracy 90.50% → 93.20% (cải thiện nhưng chưa đủ)

Bước 2: Synthetic Data với Focused Rotation

```
# Tạo 500 rectangles với rotation distribution đặc biệt:
rectangles = []
for i in range(500):
    rect = create_rectangle()

    if i < 167: # 33%: Vùng thiếu dữ liệu (30-60°)
        angle = random.uniform(30, 60) # ◇ Diamond shapes
    elif i < 334: # 33%: Rotation nhẹ
        angle = random.uniform(0, 30)
    else: # 33%: Rotation mạnh
        angle = random.uniform(60, 90)

    rectangles.append(scipy.ndimage.rotate(rect, angle))
```

→ Result: Accuracy 93.20% → **98.56%**

Bước 3: Retrain với Combined Dataset

```
X_original = load('shapes_3classes.npz') # 2080 samples
X_synthetic = load('shapes_augmented.npz') # 1500 samples
X_combined = np.concatenate([X_original, X_synthetic[2080:]], axis=0) # 3580 total

model.fit(datagen.flow(X_combined, y_combined, batch_size=32), epochs=100)
```

Kết quả kiểm chứng:

```
test_angles = [0, 15, 30, 45, 60, 75, 90]
for angle in test_angles:
    rotated_rect = rotate_rectangle(angle)
    prediction = model.predict(preprocessed(rotated_rect))
    print(f'{angle}°: {prediction}')
```

Output:

0°: Hình chữ nhật (99.2%)
15°: Hình chữ nhật (98.5%)
30°: Hình chữ nhật (96.1%)
45°: Hình chữ nhật (93.0%) ← MỤC TIÊU CHÍNH

60°: Hình chữ nhật (95.8%)

75°: Hình chữ nhật (97.2%)

90°: Hình chữ nhật (99.0%)

5.3. Feature Maps Visualization - Model học gì từ ảnh

Test case: Số "8"

Conv Layer 1 Feature Maps:

[Filter 0]: Phát hiện cạnh dọc trái |

[Filter 1]: Phát hiện cạnh dọc phải |

[Filter 5]: Phát hiện cạnh ngang trên —

[Filter 8]: Phát hiện cạnh ngang giữa —

[Filter 12]: Phát hiện cạnh ngang dưới —

[Filter 18]: Phát hiện cung tròn trên \cap

[Filter 24]: Phát hiện cung tròn dưới \cup

Conv Layer 2 Feature Maps:

[Filter 3]: Kết hợp \rightarrow Vòng tròn trên \square

[Filter 11]: Kết hợp \rightarrow Vòng tròn dưới \square

[Filter 25]: Kết hợp \rightarrow Nét nối giữa)(

Conv Layer 3 Feature Maps:

[Filter 7]: Hai vòng tròn chồng nhau (đặc trưng của số 8)

[Filter 15]: Kiểm tra tỉ lệ vòng trên/dưới

[Filter 42]: Kiểm tra alignment (vòng trên/dưới thẳng hàng)

Dense Layer Activation:

Neuron cho class "0": 0.01 (có 1 vòng, không phải 2)

Neuron cho class "8": 0.92 \leftarrow HIGHEST (có 2 vòng chồng)

Neuron cho class "6": 0.03 (có 1 vòng + đuôi)

Neuron cho class "9": 0.02 (có 1 vòng + đuôi, ngược hướng)

\rightarrow **Kết luận:** Model phân loại đúng "8" dựa trên feature "hai vòng tròn chồng nhau"

Test case: Hình chữ nhật xoay 45° (◇)

Conv Layer 1: Phát hiện 4 cạnh thẳng với góc 45°

[Filter 6]: Cạnh chéo /

[Filter 14]: Cạnh chéo \

[Filter 21]: Cạnh chéo /

[Filter 29]: Cạnh chéo \

Conv Layer 2: Kết hợp cạnh thành góc

[Filter 8]: Góc nhọn \angle (45°) tại đỉnh trên

[Filter 15]: Góc nhọn \angle (45°) tại đỉnh phải

[Filter 23]: Góc nhọn \angle (45°) tại đỉnh dưới

[Filter 31]: Góc nhọn \angle (45°) tại đỉnh trái

Conv Layer 3: Phân tích topology

[Filter 12]: Kiểm tra số góc \rightarrow Phát hiện 4 góc

[Filter 27]: Kiểm tra góc đối diện song song \rightarrow (2 cặp)

[Filter 44]: Kiểm tra 4 cạnh bằng nhau \rightarrow (phụ thuộc aspect ratio)

Dense Layer Decision:

Neuron "Tròn": 0.02 (không có cung tròn)

Neuron "Chữ nhật": 0.93 \leftarrow HIGHEST (4 góc + 2 cặp cạnh song song)

Neuron "Tam giác": 0.05 (có 3 góc, không phải 4)

\rightarrow **Kết luận:** Model nhận diện đúng "Hình chữ nhật" nhờ học được:

- 4 góc (không phải 3)
- 2 cặp cạnh song song
- Topology khác tam giác (4 vertices vs 3 vertices)

6. KẾT LUẬN

6.1. Các kỹ thuật Xử lý ảnh đã áp dụng

S T T	Kỹ thuật	Công dụng	Tham số quan trọng
1	Color Space Conversion	RGB \rightarrow Grayscale	cv2.COLOR_RGB2GRAY
2	CLAHE	Cân bằng độ sáng cục bộ	clipLimit=2.0, tileGridSize=(8,8)
3	Gaussian Blur	Khử nhiễu cho digits	kernel=5 \times 5
4	Bilateral Filter	Khử nhiễu giữ cạnh cho shapes	d=9, σ =75
5	Adaptive Threshold	Nhị phân hóa động	blockSize=11, C=2
6	Morphology CLOSE	Nối nét gián đoạn	iterations=1, kernel=3 \times 3
7	Morphology OPEN	Loại nhiễu (chỉ digits)	iterations=1, kernel=3 \times 3
8	Contour Detection	Tìm đường viền đối tượng	RETR_EXTERNAL, CHAIN_APPROX_SIMPLE

9	Bounding Rectangle	Xác định vị trí/kích thước	cv2.boundingRect()
10	Resize + Padding	Chuẩn hóa về 28×28	INTER_AREA interpolation
11	Center of Mass	Canh giữa trọng tâm	scipy.ndimage.center_of_mass

6.2. Điểm nổi bật của dự án

Về mặt kỹ thuật Xử lý ảnh:

1. **Pipeline hoàn chỉnh 10 bước:** Từ RGB → Binary → Normalized 28×28
2. **CLAHE cục bộ:** Xử lý ảnh tối/sáng không đều
3. **Adaptive Thresholding:** Tự động điều chỉnh ngưỡng theo từng vùng
4. **Morphology tối ưu:** iterations=1, kernel=3×3 để bảo toàn góc hình học
5. **Bilateral Filter:** Làm mịn nhưng giữ cạnh cho hình học

Về mặt Deep Learning:

1. **CNN 4 layers:** Học từ low-level (edges) → high-level (shapes)
2. **Data Augmentation thông minh:** Rotation khác nhau cho digits (15°) vs shapes (40°)
3. **Synthetic Data Generation:** 1,500 mẫu với focused rotation (30-60°)
4. **Feature Maps Visualization:** Giải thích model học gì từ ảnh
5. **Model học từ ảnh đã preprocessing:** Consistency giữa train và test

Về mặt giải quyết vấn đề:

1. **Khắc phục rotated rectangle:** Từ 85% nhầm → 93% đúng (45° diamond)
2. **Tối ưu morphology:** Giảm rounding effect để giữ góc vuông
3. **Balanced dataset:** Thêm synthetic data vào vùng thiếu (30-60° rotation)
4. **Accuracy tăng:** 90.50% → **98.56%** (shapes)

6.3. Thành tựu đạt được

- **Độ chính xác cao:** 99.75% (digits), 98.56% (shapes)
- **Pipeline xử lý ảnh chuyên nghiệp:** 10 bước với các kỹ thuật OpenCV tiên tiến
- **CNN architecture hiệu quả:** 4 Conv layers học hierarchical features
- **Giao diện trực quan:** 4 tabs với visualization pipeline và feature maps

- **Explainable AI:** Hiểu rõ model học gì từ mỗi layer
- **Xử lý edge cases:** Hình xoay, ảnh tối, độ sáng không đều

6.4. Đóng góp về mặt học thuật

Dự án chứng minh khả năng **kết hợp Xử lý ảnh cổ điển (OpenCV) với Deep Learning (CNN)** để đạt hiệu quả cao:

1. **Preprocessing giảm complexity cho model:**
 - Loại bỏ nhiễu → Model tập trung học shape
 - Chuẩn hóa vị trí → Giảm variance trong data
 - Binary image → Model chỉ cần học contour, không cần texture
2. **Morphology operations cần fine-tuning:**
 - iterations=2 vs iterations=1: Ảnh hưởng lớn đến corner preservation
 - kernel=5×5 vs 3×3: Quyết định rounding effect
 - CLOSE phù hợp với nổi nét, OPEN phù hợp với loại nhiễu
3. **Data augmentation phải phù hợp với problem domain:**
 - Digits: rotation=15° (tránh ambiguity 6↔9)
 - Shapes: rotation=40° + horizontal/vertical flip (learn invariance)
 - Synthetic data: Tập trung vào vùng thiếu dữ liệu
4. **Model học từ preprocessed images:**
 - Training data qua preprocessing → Test data cũng phải qua preprocessing
 - Consistency là chìa khóa để đạt accuracy cao
 - Feature maps cho thấy model học shape chứ không phải raw pixels