TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGUYỄN THỊ THÚY VY - 232805404
HOÀNG ĐÌNH QUÝ VŨ - 521H0517

BÁO CÁO GIỮA KỲ

XỬ LÝ ẢNH SỐ

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGUYỄN THỊ THÚY VY - 232805404
HOÀNG ĐÌNH QUÝ VŨ - 521H0517

BÁO CÁO GIỮA KỲ

XỬ LÝ ẢNH SỐ

Người hướng dẫn
TS. Trịnh Hùng Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Chúng em xin bày tỏ lòng biết ơn chân thành đến thầy Trịnh Hùng Cường vì những kiến thức quý báu mà thầy đã truyền đạt và sự tận tâm trong việc giảng dạy môn Xử lý ảnh số. Chúng em cảm nhận được sự chuyên nghiệp và đam mê của thầy trong việc truyền đạt tri thức, và chúng em rất biết ơn vì thầy đã dành thời gian và công sức để hướng dẫn chúng em trong quá trình học tập và tìm hiểu về lĩnh vực này.

Thầy đã truyền đạt những kiến thức sâu sắc và chi tiết về Xử lý ảnh số, giúp chúng em hiểu rõ hơn về khung phát triển này và cách áp dụng vào thực tế. Nhờ những điều thầy đã truyền dạy, chúng em đã nắm vững cách xử lý dữ liệu một cách hiệu quả, đem lại kết quả tốt trong ứng dụng thực tế.

Chúng em cũng biết ơn vì sự quan tâm và hỗ trợ tận tình của thầy trong quá trình học tập. Thầy đã luôn sẵn sàng trả lời các câu hỏi của chúng em và giúp đỡ chúng em vượt qua những khó khăn trong quá trình nắm bắt kiến thức. Nhờ đó, chúng em đã có thêm niềm tin và động lực để tiếp tục khám phá và phát triển trong lĩnh vực Xử lý ảnh số.

Chúng em cảm nhận được sự chuyên nghiệp và đam mê của thầy trong việc giảng dạy. Sự cống hiến và tâm huyết của thầy đã giúp chúng em có được nền tảng vững chắc, đồng thời truyền cảm hứng để chúng em tiếp tục theo đuổi đam mê và ước mơ của bản thân.

Với tấm lòng biết ơn sâu sắc, chúng em xin kính chúc thầy Trịnh Hùng Cường sức khỏe dồi dào, hạnh phúc và ngày càng thành công trong việc truyền tải tri thức và hỗ trợ sinh viên. Mong rằng những đóng góp của thầy sẽ tiếp tục lan tỏa và mang lại những thành tựu to lớn cho thầy và cả khoa Công nghệ thông tin.

TP. Hồ Chí Minh, ngày 30 tháng 07 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

VY

Nguyễn Thị Thúy Vy

VŨ

Hoàng Đình Quý Vũ

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của TS. Trịnh Hùng Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 30 tháng 07 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

VY

Nguyễn Thị Thúy Vy

VŨ

Hoàng Đình Quý Vũ

TÊN ĐỀ TÀI

TÓM TẮT

Bài báo cáo trình bày các nghiên cứu về những vấn đề sau:

Vấn đề 1: Phương pháp giải quyết bài toán Tự động vẽ các hình chữ nhật bao quanh mỗi chữ số trong ảnh đầu vào và lưu ảnh đầu ra vào một tệp

Vấn đề 2: Kết quả bài toán

TÓM TẮT

Trong bài toán nhận dạng và phân vùng chữ số trong hình ảnh, chúng tôi triển khai một phương pháp tự động vẽ các hình chữ nhật bao quanh từng chữ số trong một hình ảnh đầu vào và lưu lại hình ảnh đầu ra sau khi đã xử lý. Quá trình này bao gồm các bước chính sau: tiền xử lý hình ảnh, chuyển đổi sang ảnh nhị phân, áp dụng các biến đổi hình thái học như dilation và opening, phát hiện đường viền, và vẽ hình chữ nhật bao quanh các ký tự.

Cụ thể, hình ảnh đầu vào được chuyển đổi sang thang độ xám và chia thành bốn phần: trên-trái, trên-phải, dưới-trái và dưới-phải. Mỗi phần được xử lý riêng biệt bằng các kỹ thuật khác nhau, bao gồm ngưỡng nhị phân, ngưỡng thích nghi và các phép biến đổi hình thái học để làm rõ các chữ số và loại bỏ nhiễu. Các đường viền sau đó được phát hiện từ các phần đã xử lý và được ghép lại thành một hình ảnh lớn. Cuối cùng, các hình chữ nhật được vẽ bao quanh từng chữ số dựa trên các đường viền đã phát hiện.

MỤC LỤC

DANH MỤC HÌNH VẼ

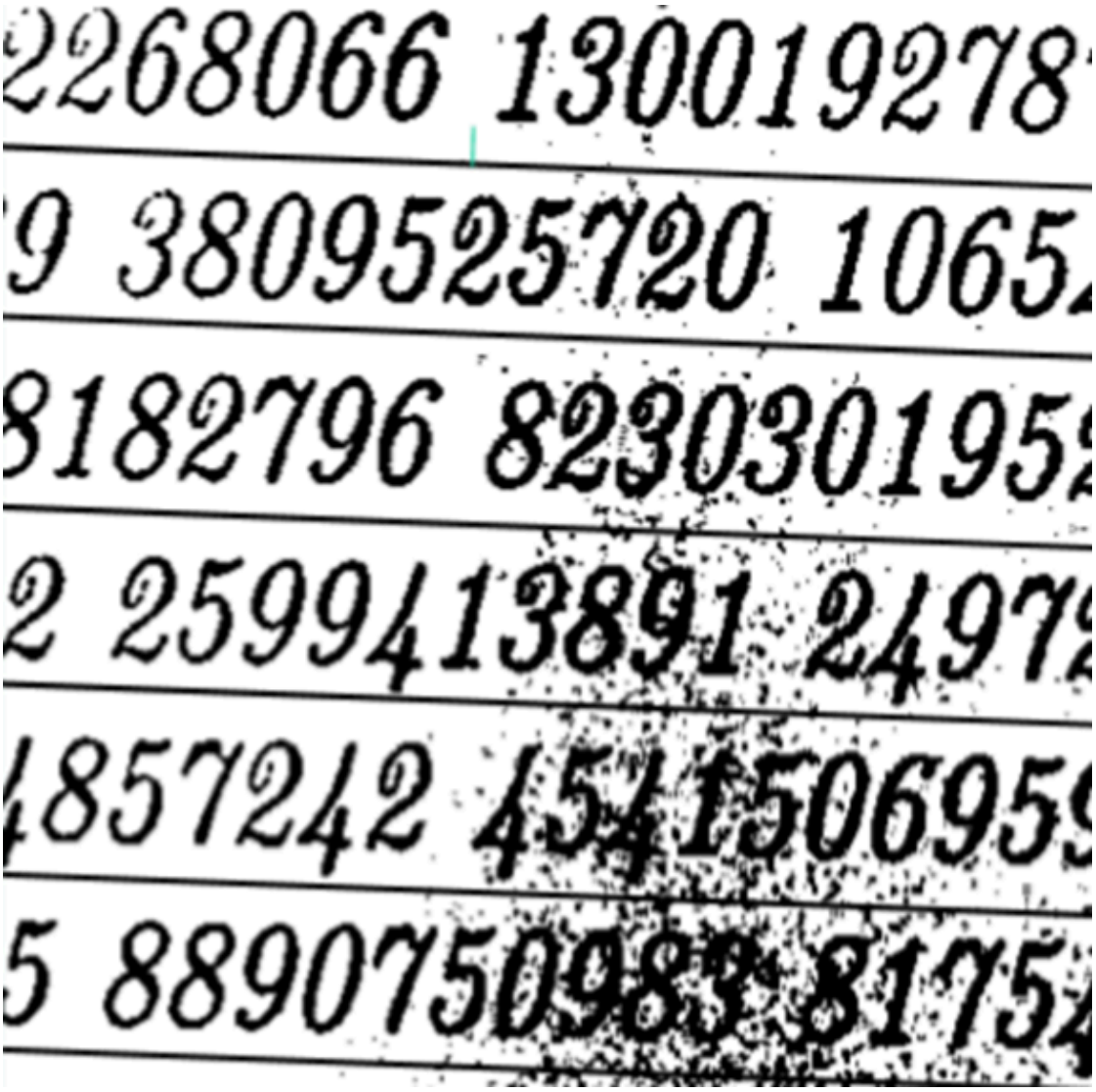
DANH MỤC CÁC CHỮ VIẾT TẮT

OCR	Optical Character Recognition

CHƯƠNG 1. PHƯƠNG PHÁP XỬ LÝ BÀI TOÁN

1.1 Bài toán

Trong lĩnh vực nhận dạng ký tự quang học (OCR), việc phát hiện và phân vùng các chữ số trong hình ảnh là một bước quan trọng. Bài toán đặt ra là tự động xác định và vẽ các hình chữ nhật bao quanh từng chữ số trong một hình ảnh đầu vào. Sau khi xử lý, hình ảnh đầu ra sẽ được lưu lại với các chữ số được bao quanh bởi các hình chữ nhật, giúp xác định rõ vị trí và kích thước của từng chữ số. Hình ảnh đầu vào cụ thể như sau:

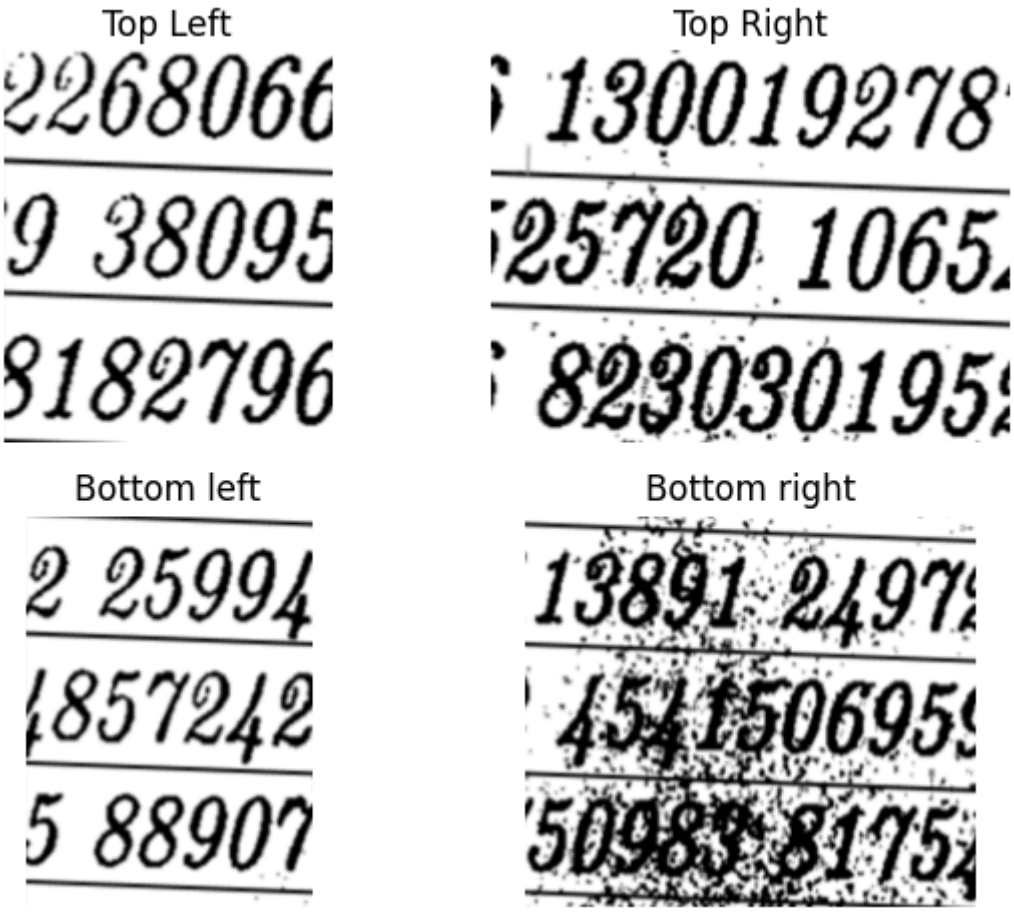


Hình 1.1.a - Hình ảnh đầu vào (input.png)

1.2 Phương pháp xử lý

1.2.1 Tiền xử lý hình ảnh

- Đưa hình ảnh đầu vào (Hình 1)
- Chuyển đổi hình ảnh sang dạng ảnh xám: Hình ảnh đầu vào được chuyển đổi sang thang độ xám để giảm thiểu thông tin không cần thiết và đơn giản hóa việc xử lý
- Chia ảnh thành 4 phần theo tọa độ x, y gồm top_left, top_right, bottom_left, bottom_right. Điều này giúp xử lý cục bộ từng phần ảnh, từ đó cải thiện độ chính xác và hiệu suất xử lý. Dựa vào công cụ paint và vẽ matplotlib, nhóm em đã chọn được tọa độ x, y phù hợp ()

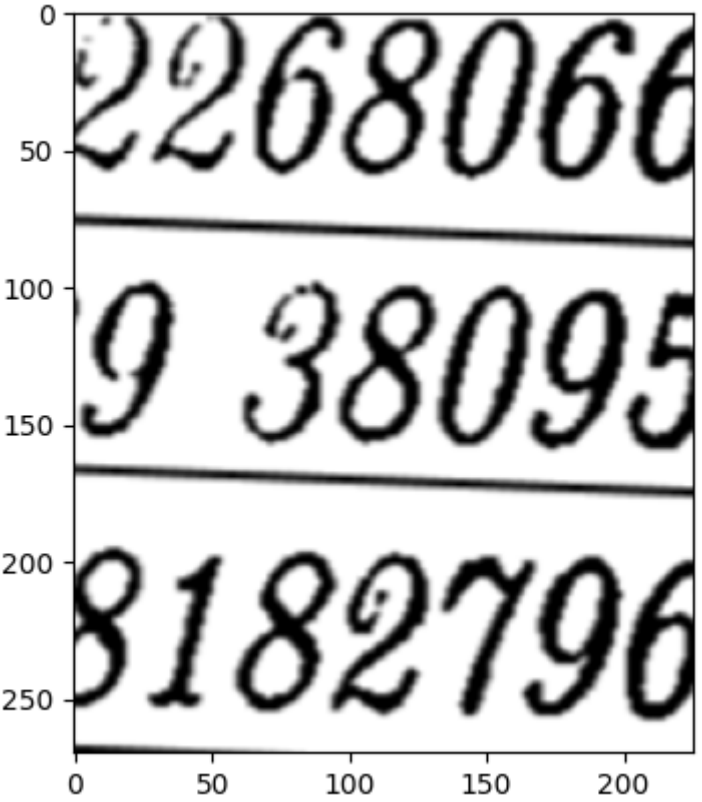


Hình 1.2.1.a - Chia ảnh đầu vào thành 4 phần

1.2.2 Xử lý cục bộ từng phần ảnh:

1.2.2.1 Phân tích phần ảnh top-left:

- Các ký tự số ở phía trên cùng bên trái của hình ảnh có vẻ hơi mờ, đặc biệt là số 2 đầu tiên. Điều này có thể khiến cho quá trình nhận diện gặp khó khăn.
- Ký tự số bị đứt nét: Số 2 hàng thứ nhất và số 3 ở hàng thứ hai có dấu hiệu bị đứt nét, có các phần bị thiếu hoặc không liền mạch.
- Phần ảnh này khá sạch, không thấy nhiễu



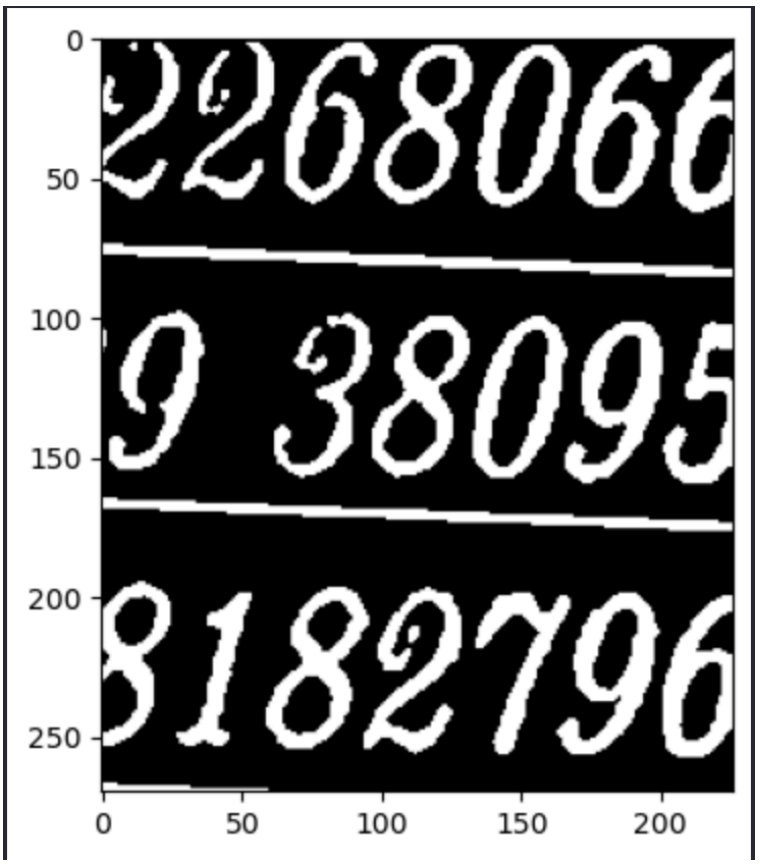
Hình 1.2.2.1.a - Phần top-left chưa xử lý

- Phương pháp xử lý:
  - Ngưỡng hóa ảnh (Thresholding): Chuyển đổi ảnh gốc thành ảnh nhị phân bằng cách sử dụng ngưỡng cố định và ngưỡng đảo, giúp làm nổi bật các ký tự cần nhận dạng.

```
_, image_binary = cv2.threshold(top_left, 150, 255,
cv2.THRESH_BINARY_INV)
```

Trong đó, tham số:

- top_left: là phần ảnh đầu vào
- 150: là giá trị ngưỡng. Mọi giá trị pixel lớn hơn 150 sẽ được chuyển thành màu đen (0), và mọi giá trị pixel nhỏ hơn hoặc bằng 150 sẽ được chuyển thành màu trắng (255)
- 255: là giá trị tối đa dùng để chuyển đổi. Pixel nào vượt quá ngưỡng sẽ được đặt về giá trị này.
- cv2.THRESH_BINARY_INV: là kiểu nhị phân với các pixel vượt giá trị ngưỡng sẽ chuyển thành giá trị 0 (đen), và các pixel dưới ngưỡng sẽ chuyển thành giá trị 255 (trắng)



Hình 1.2.2.1.b - Phần top-left đã xử lý (image_binary)

- Thực hiện các phép biến đổi hình thái học (Morphological Transformations): sử dụng Dilation giúp mở rộng các ký tự số để làm chúng rõ hơn

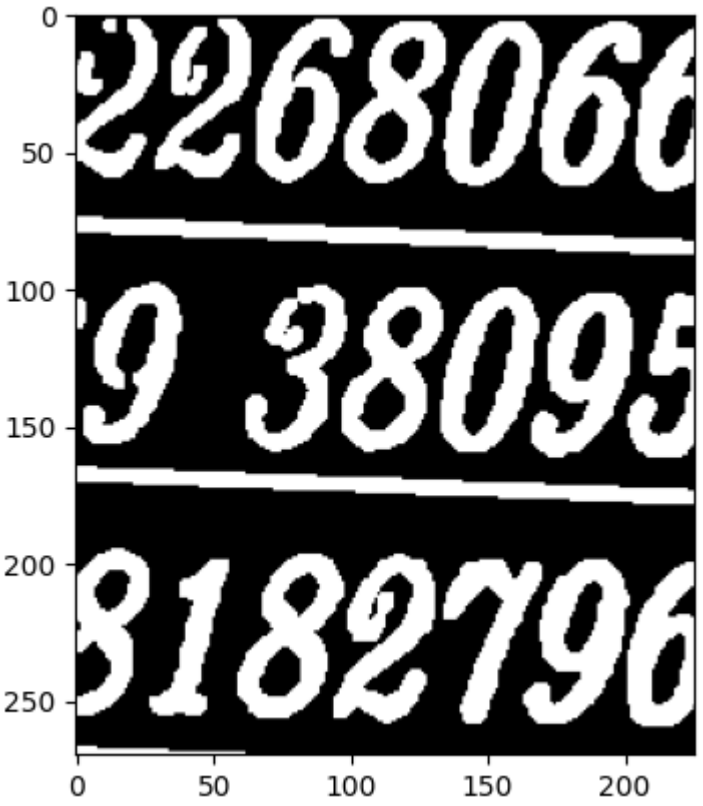
```
kernel = np.ones((2, 2))
```

```
image_processed = cv2.morphologyEx(image_binary, cv2.MORPH_DILATE, kernel, iterations=2)
```

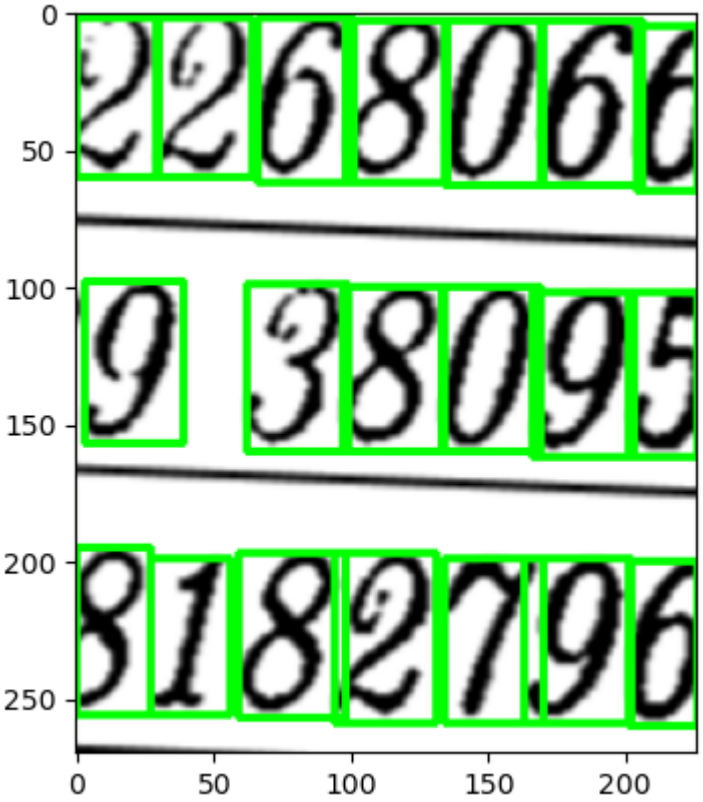
Trong đó, tham số:

- Tạo kernel 2x2 là ma trận chứa các giá trị `1`
- image_processed: kết quả của quá trình thực hiện hàm biến đổi hình thái học (cv2.morphologyEx) với phép biến đổi giãn nở (MORPH_DILATE) áp dụng kernel và được lặp lại quá trình 2 lần (iterations=2)

- Kết quả sau khi xử lý: Các ký tự số trở nên rõ ràng hơn



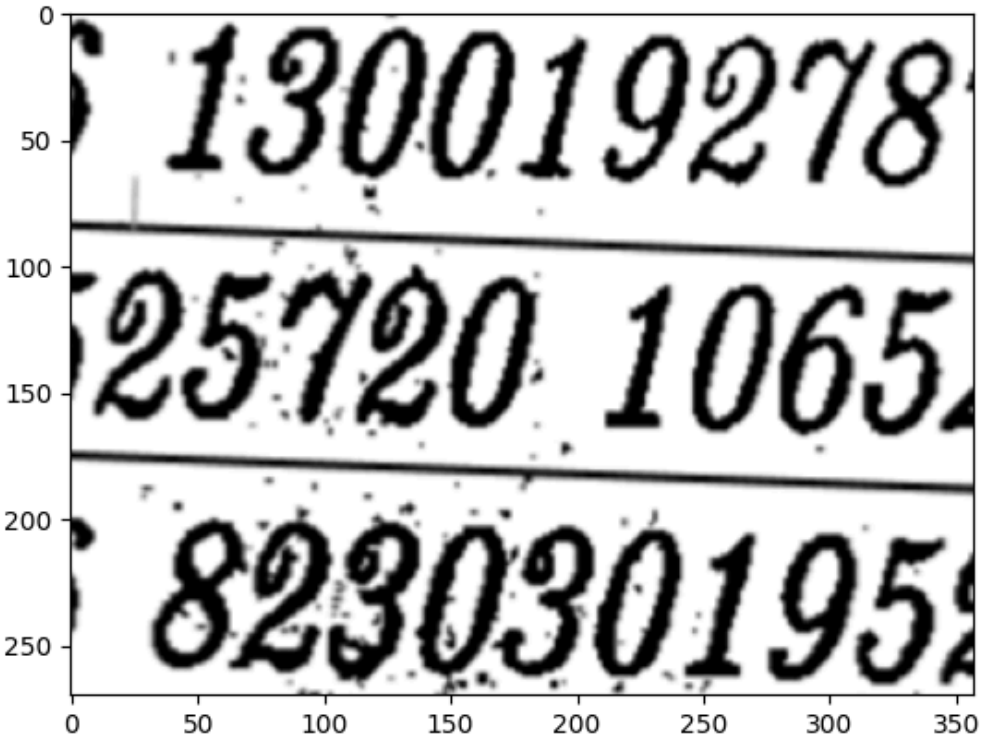
Hình 1.2.2.1.c - Phần top-left đã xử lý (image_processed)



Hình 1.2.2.1.d - Phần top-left đã xử lý viền

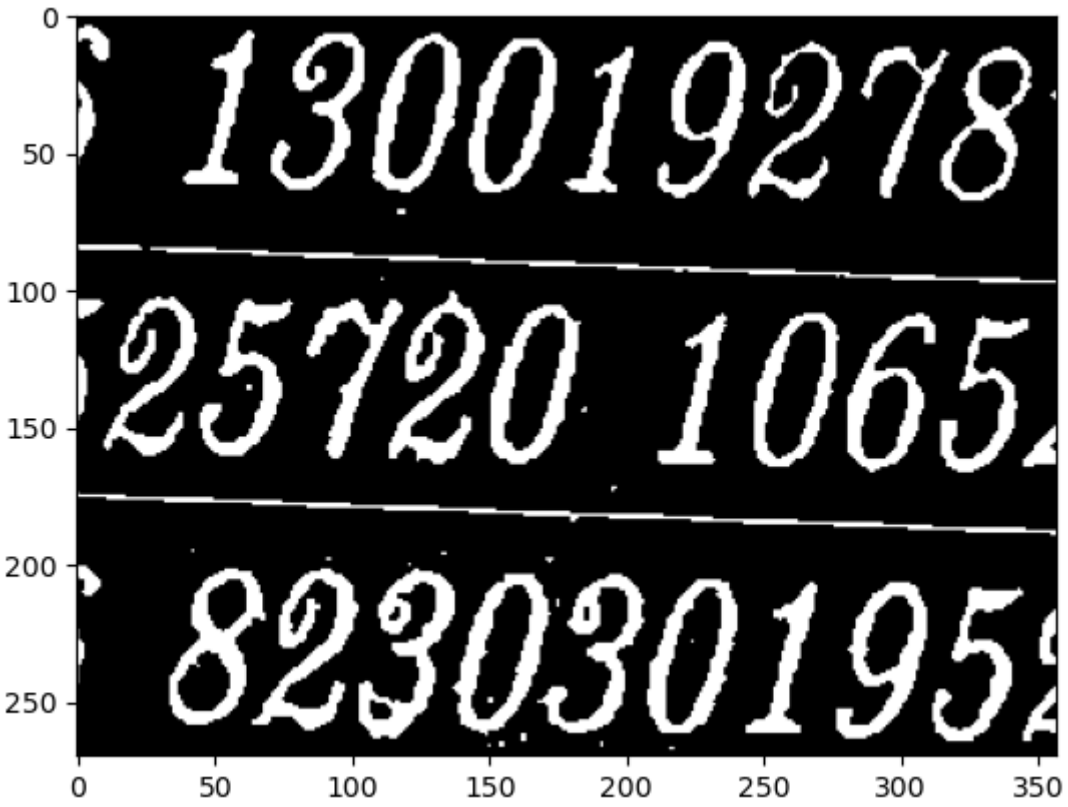
1.2.2.2 Phân tích phần ảnh top-right:

- Các ký tự số trong hình ảnh có độ tương phản tốt so với nền trắng, giúp dễ dàng nhận diện
- Hình ảnh có các nhiễu nhỏ dạng chấm đen xung quanh và bên trong các ký tự số, đặc biệt rõ ràng ở các ký tự số 25720 và 8230301



Hình 1.2.2.2.a - Phần top-right chưa xử lý

- Phương pháp xử lý:
    - Ngưỡng hóa ảnh (Thresholding): Chuyển đổi ảnh gốc thành ảnh nhị phân bằng cách sử dụng ngưỡng cố định và ngưỡng đảo, giúp làm nổi bật các ký tự cần nhận dạng. Chọn ngưỡng (threshold) là 50 vì phần ảnh có vẻ tối nên các pixel lớn hơn 50 sẽ thành màu trắng (255), còn lại sẽ thành màu đen (0)

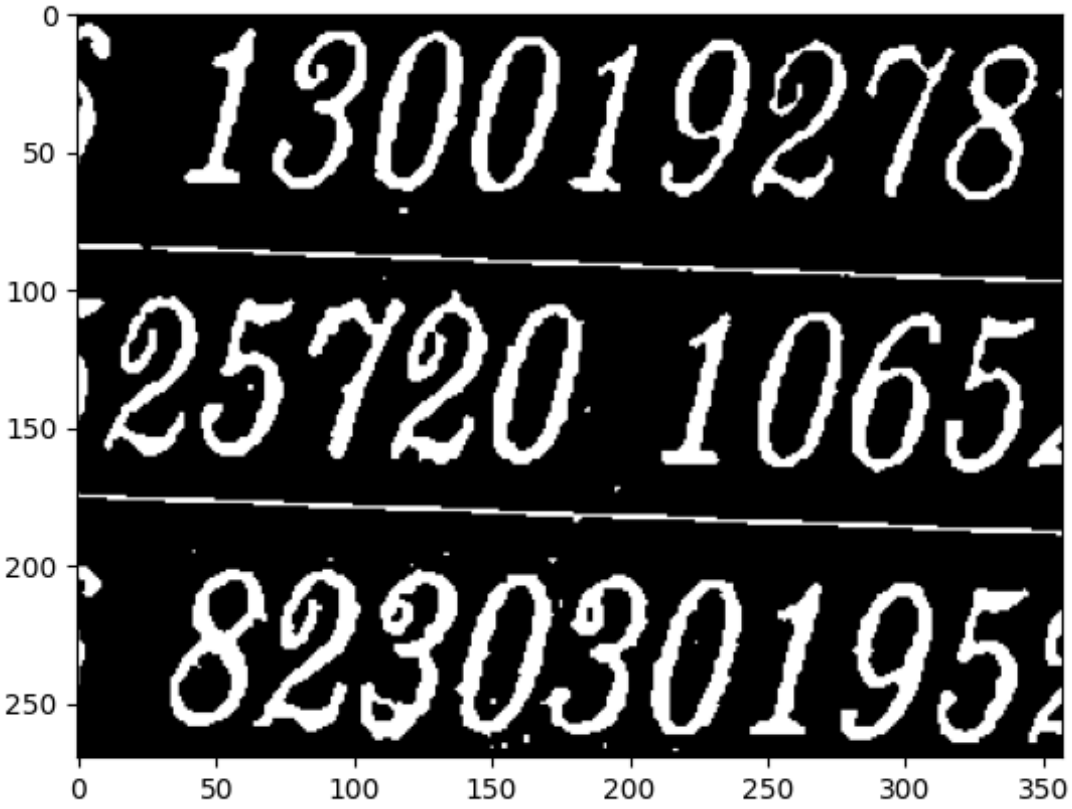


Hình 1.2.2.2.b - Phần top-right đã xử lý Thresholding (image_binary)

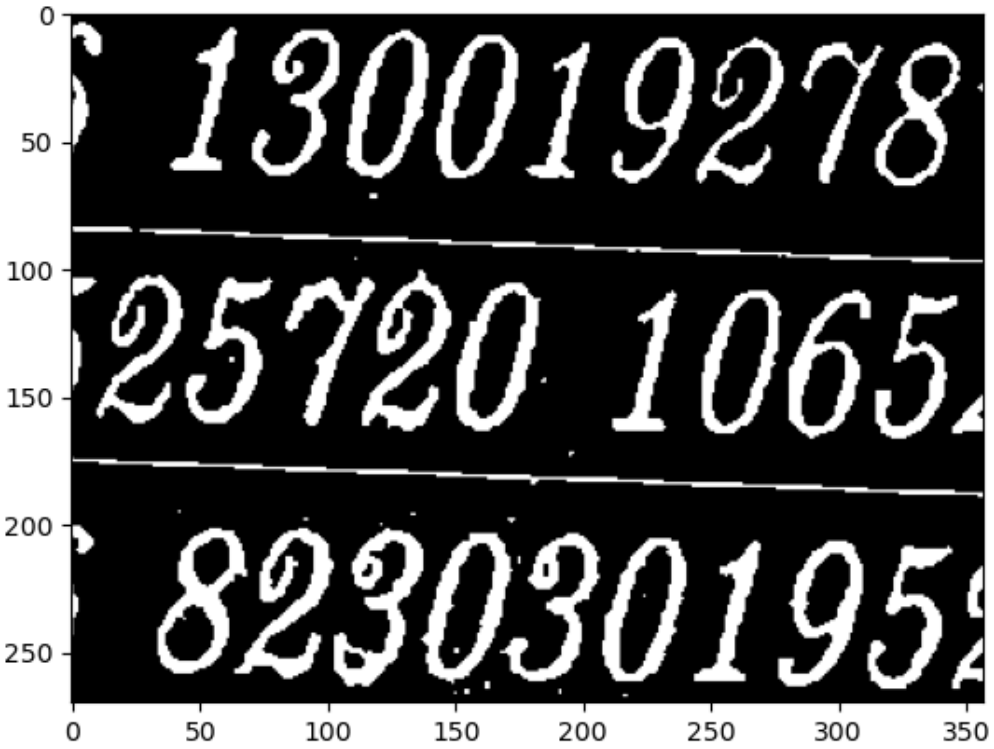
- Thực hiện các phép biến đổi hình thái học (Morphological Transformations): kết hợp phép biến đổi hình thái học mở (mở rộng sau đó co lại) (cv2.MORPH_OPEN) và phép biến đổi hình thái học mở rộng (dilation). Mục đích là loại bỏ các nhiễu nhỏ, giúp các ký tự số rõ ràng hơn, dilation giúp ký tự được mở rộng và kết nối liền mạch hơn.
- kernel = np.ones((1, 1)): Kernel là một ma trận 1x1, giúp giữ nguyên kích thước của các ký tự số mà chỉ loại bỏ nhiễu nhỏ

```
kernel = np.ones((1, 1))

    image_opened = cv2.morphologyEx(image_binary, cv2.MORPH_OPEN, kernel)

    image_processed = cv2.morphologyEx(image_opened, cv2.MORPH_DILATE, kernel)
```

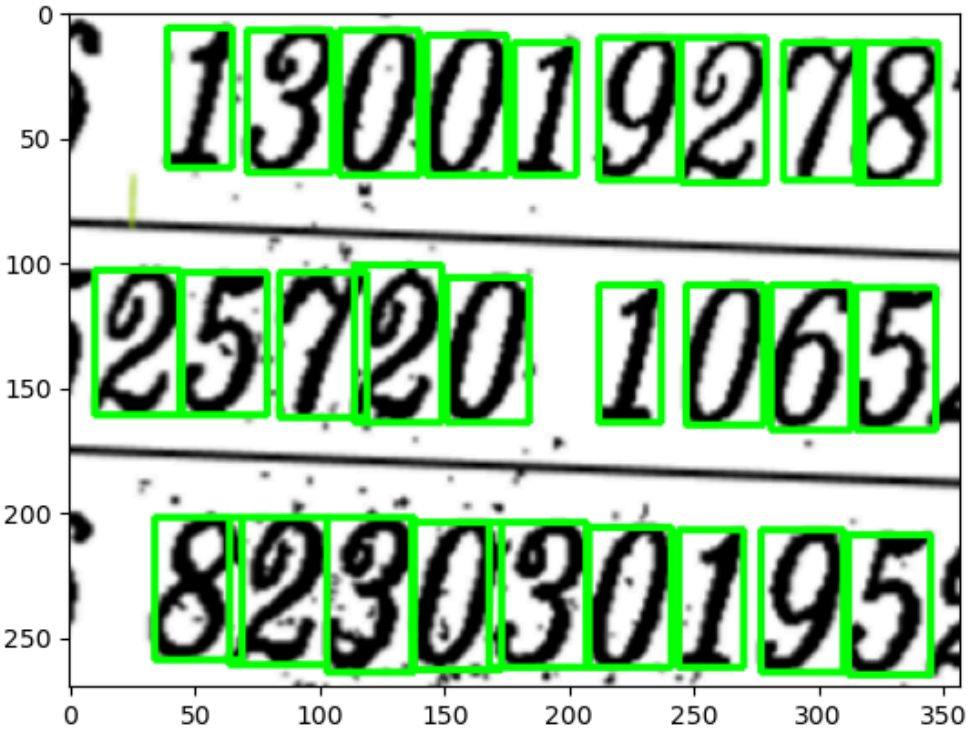
- Kết quả sau khi xử lý:
    - Ký tự số rõ ràng, đã giảm nhiễu rất nhiều



Hình 1.2.2.2.c - Phần top-right đã xử lý (image_opened)



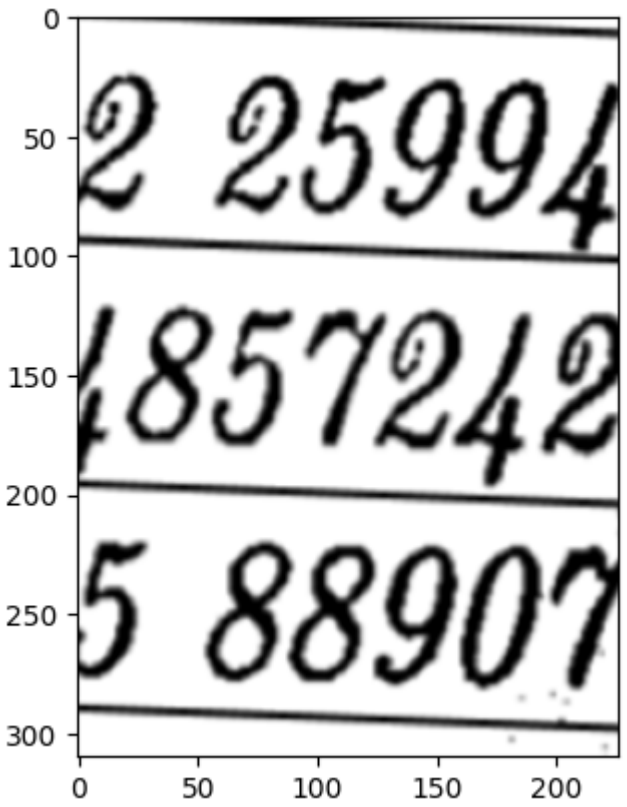
Hình 1.2.2.2.d - Phần top-right đã xử lý (image_processed)



Hình 1.2.2.2.d - Phần top-right đã xử lý viền

1.2.2.3 Phân tích phần ảnh bottom-left

- Một số ký tự số bị mờ hoặc có các phần đứt đoạn
- Có một số nhiễu nhỏ xung quanh các ký tự 0, 7
- Ký tự ‘4’ đầu tiên bên trái ở hàng 2 không được hiển thị đầy đủ
- Độ mờ/đậm của các ký tự khác nhau.



Hình 1.2.2.3.a - Phần bottom-left chưa xử lý

- Phương pháp xử lý:
  - Ngưỡng hóa ảnh (Thresholding): Chuyển đổi ảnh gốc thành ảnh nhị phân bằng cách bằng phương pháp ngưỡng thích nghi (Adaptive Thresholding)

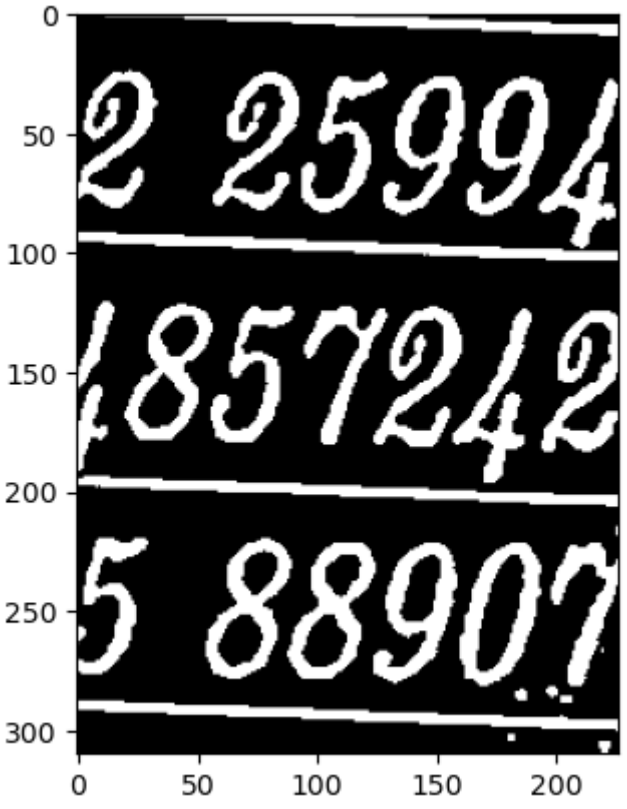
```
# Apply adaptive thresholding to invert the image
   thresholded = cv2.adaptiveThreshold(bottom_left, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 11, 2)
```

```
   # Define a horizontal kernel for morphological
operations to detect long horizontal lines
   horizontal_kernel =
cv2.getStructuringElement(cv2.MORPH_RECT, (25, 1))
   detect_lines = cv2.morphologyEx(thresholded,
cv2.MORPH_OPEN, horizontal_kernel, iterations=2)

   # The detected lines are white (255), we want to
make them black (0)
   mask_lines_black = cv2.bitwise_not(detect_lines)

   # Apply the mask to the thresholded image,
turning only the detected lines black
   result = cv2.bitwise_and(thresholded,
mask_lines_black)

   kernel = np.ones((2, 2))
   image_gray_2_open = cv2.morphologyEx(result,
cv2.MORPH_OPEN, kernel)

   kernel = np.ones((3, 3))
   image_processed =
cv2.morphologyEx(image_gray_2_open,
cv2.MORPH_DILATE, kernel)
```

Trong đó, tham số:

- ‘255’: Giá trị pixel tối đa (màu trắng) trong ảnh nhị phân

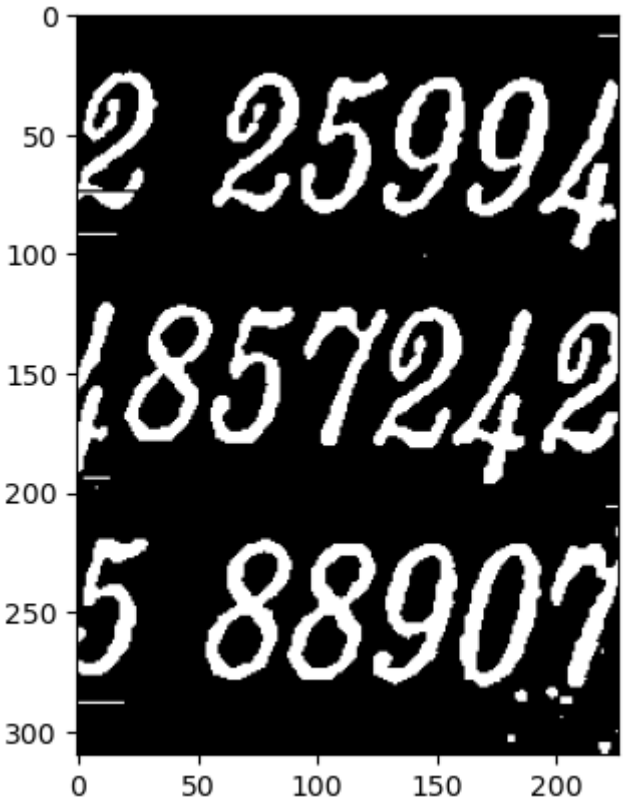
- ‘cv2.ADAPTIVE_THRESH_MEAN_C’: sử dụng phương pháp ngưỡng thích nghi, trong đó ngưỡng cho mỗi pixel được tính toán dựa trên giá trị trung bình của các pixel xung quanh
- ‘cv2.THRESH_BINARY_INV’: Chuyển đổi ảnh sang ảnh nhị phân đảo ngược, nghĩa là các pixel có giá trị lớn hơn ngưỡng sẽ được gán giá trị 0 (đen) và các pixel có giá trị nhỏ hơn ngưỡng sẽ được gán giá trị 255 (trắng)
- ‘11’: Kích thước cửa sổ (block size) để tính toán giá trị ngưỡng cục bộ
- ‘2’: Giá trị hằng số được trừ đi từ giá trị trung bình cục bộ để tính toán ngưỡng.



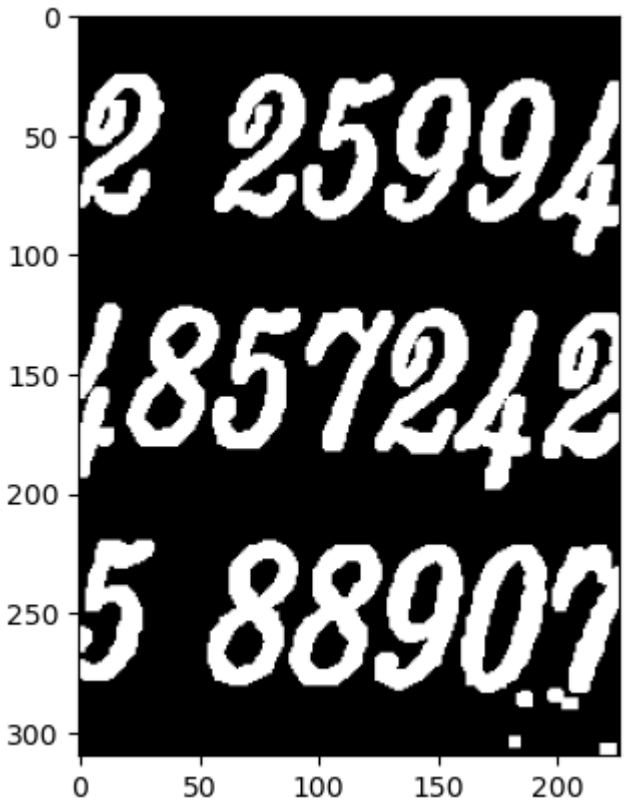
Hình 1.2.2.3.b - Phần bottom-left đã xử lý thresholded

- Thực hiện các phép biến đổi hình thái học (Morphological Transformations): kết hợp phép biến đổi hình thái học

- Sử dụng ‘cv2.morphologyEx’ với ‘cv2.MORPH_OPEN’ với kernel hình chữ nhật với kích thước (25, 1) để loại bỏ nhiễu và chỉ giữ lại các dòng ngang dài
- Sử dụng ‘cv2.bitwise_not’ đảo ngược các giá trị của các dòng phát hiện được: từ trắng thành đen.
- ‘cv2.bitwise_and’ áp dụng mặt nạ, biến các dòng phát hiện được thành đen trên ảnh ngưỡng nhị phân.
- Tạo kernel (2, 2) để thực hiện phép mở ‘cv2.MORPH_OPEN’, loại bỏ các nhiễu nhỏ.
- Tạo kernel (3, 3) để thực hiện phép giãn ‘cv2.MORPH_DILATE’, mở rộng các vùng sáng, làm nổi bật các đặc điểm cần thiết.

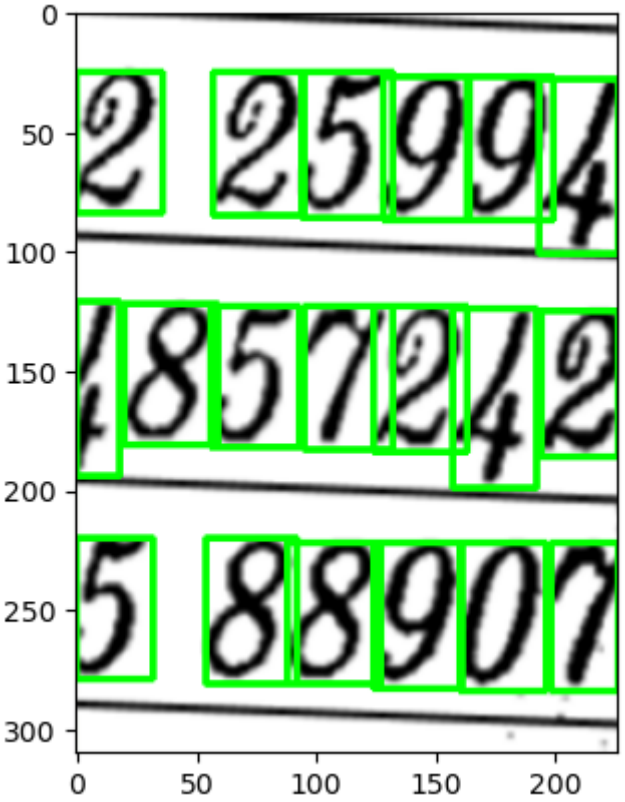


Hình 1.2.2.3.c - Phần bottom-left đã xử lý bitwise_and



Hình 1.2.2.3.d - Phần bottom-left đã xử lý (image_processed)

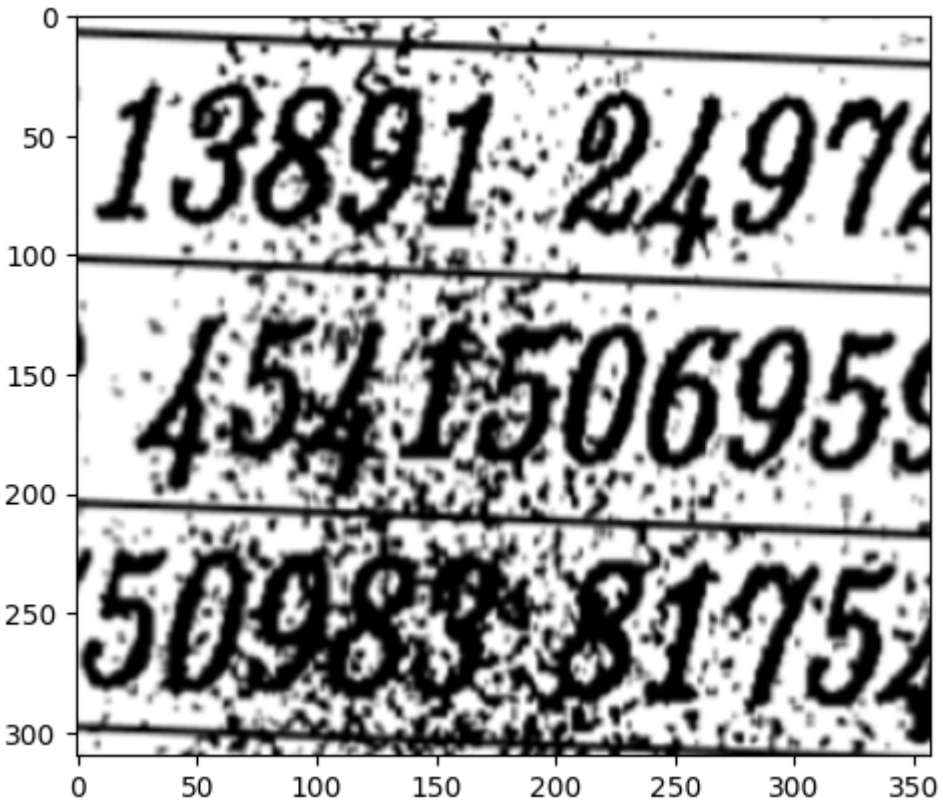
- Lý do sử dụng phương pháp Adaptive Thresholding:
    - Giúp phân tách ký tự số một cách chính xác ngay cả khi độ sáng và độ tương phản thay đổi.
    - Loại bỏ nhiễu và giữ lại các chi tiết quan trọng của ký tự, giúp cải thiện độ chính xác trong việc nhận diện
    - Kết nối các phần đứt đoạn của ký tự số
- Kết quả sau khi xử lý:
    - Các ký tự số trở nên rõ ràng và dễ nhận diện hơn
    - Các ký tự số có thể đã bị đứt đoạn hoặc không liền mạch trước khi xử lý. Sau khi áp dụng Adaptive Thresholding, các phần đứt đoạn này đã được kết nối, giúp việc nhận diện trở nên dễ dàng hơn.
    - Các nhiễu nhỏ đã được loại bỏ



Hình 1.2.2.3.e - Phần bottom-left đã xử lý viền

1.2.2.4 Phân tích phần ảnh bottom-right

- Có rất nhiều điểm nhiễu nhỏ phân tán khắp phần ảnh và mức độ phân bố không đều. Những điểm này xuất hiện cả bên trong và xung quanh các ký tự số làm cho các ký tự số khó phân biệt, đặc biệt là trong các vùng nhiễu dày đặc. Điều này gây khó khăn cho việc nhận diện và phân tách các ký tự số.
- Các ký tự không rõ nét
- Độ tương phản không đủ cao giữa ký tự và nền làm cho việc phân tách ký tự khỏi nền trở nên khó khăn. Điều này có thể dẫn đến việc bỏ sót hoặc nhận diện sai các ký tự.
- Có nhiều ký tự chưa đầy đủ ví dụ 2, 9, 4 ở bên phải



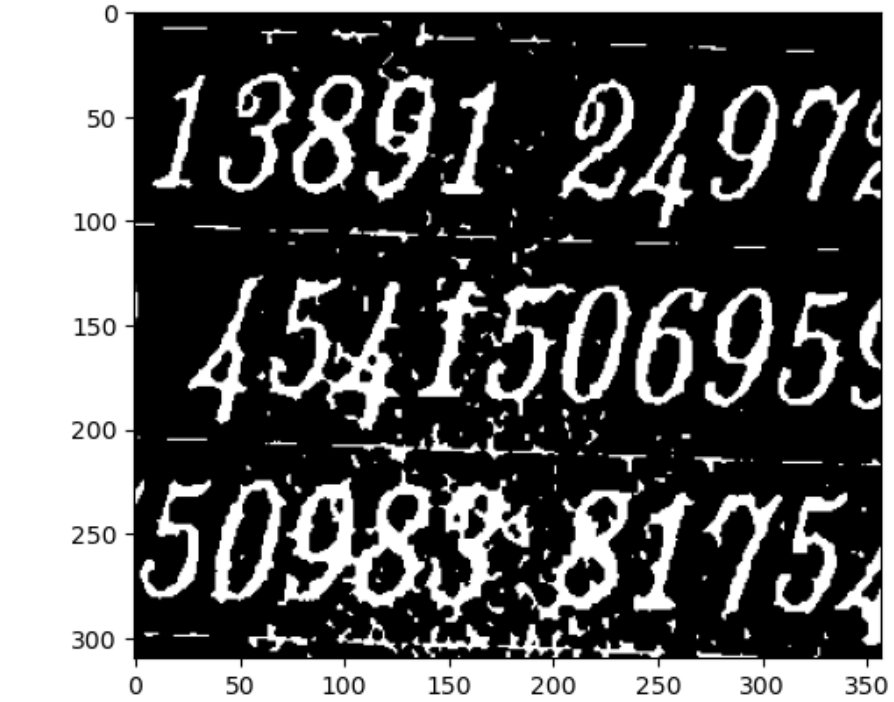
Hình 1.2.2.4.a - Phần bottom-right chưa xử lý

- Phương pháp xử lý:
    - Ngưỡng toàn cục (Global Thresholding): Chuyển đổi ảnh gốc thành ảnh nhị phân bằng cách bằng phương pháp ngưỡng toàn cục với giá trị ngưỡng là 25. Lý do chọn ngưỡng là 25 để giúp loại bỏ nhiều nhiễu hơn, các pixel có giá trị xám dưới 25 thành màu trắng và trên 25 thành màu đen.
    - Thực hiện các phép biến đổi hình thái học (Morphological Transformations): kết hợp phép biến đổi hình thái học
        - Áp dụng phép giãn (Dilation) để kết nối các phần đứt đoạn của ký tự số, làm đầy lỗ nhỏ trong ký tự với kernel kích thước 2x2 và lặp lại 1 lần
        - Áp dụng phép Erosion với kernel kích thước 3x3 và lặp lại 1 lần để loại bỏ các nhiễu nhỏ xung quanh ký tự số, giúp làm sạch ảnh

- Áp dụng phép giãn (Dilation) lần 2 với kernel kích thước 2x2 và lặp lại 1 lần để kết nối lại các phần đứt đoạn sau khi co, đảm bảo các ký tự số được kết nối liền mạch

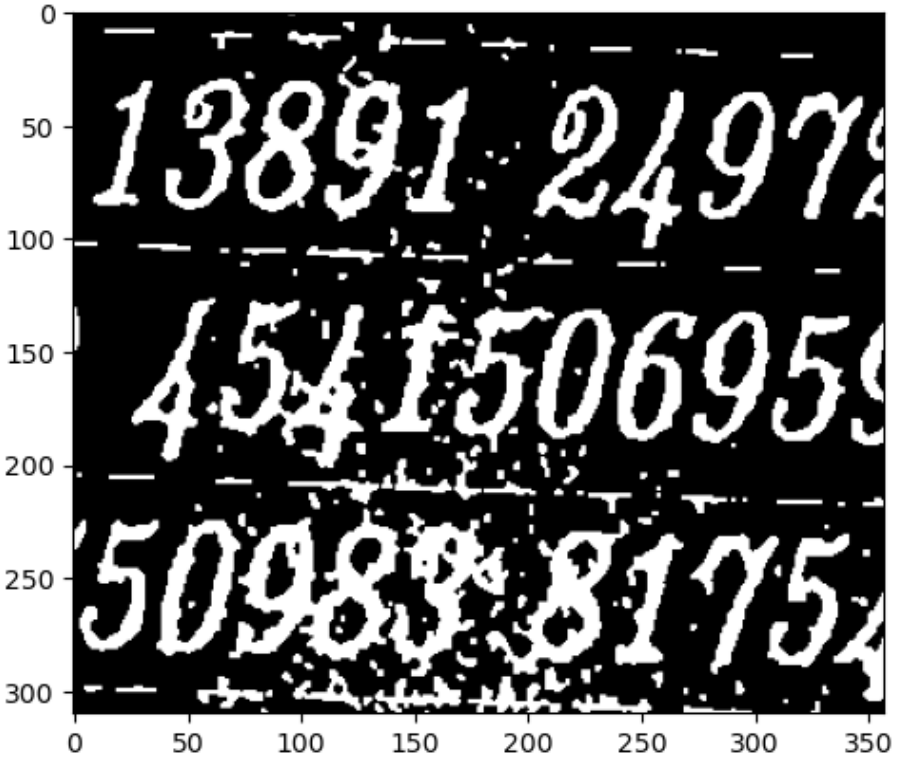
```
 _, image_binary = cv2.threshold(bottom_right, 25, 255, cv2.THRESH_BINARY_INV)


    kernel = np.ones((2, 2))
    image_dilated1 = cv2.morphologyEx(image_binary, cv2.MORPH_DILATE, kernel, iterations=1)


    kernel = np.ones((3, 3))
    image_eroded = cv2.erode(image_dilated1, kernel, iterations=1)


    kernel = np.ones((2, 2))
    image_processed = cv2.morphologyEx(image_eroded, cv2.MORPH_DILATE, kernel, iterations=1)
```

- Kết quả sau khi xử lý:
  - Các nhiễu nhỏ xung quanh ký tự số bị loại bỏ, giúp làm sạch ảnh
  - Các phần đứt đoạn của ký tự số được kết nối, làm rõ các ký tự số
  - Các ký tự số nổi bật trên nền đen, dễ nhận diện và phân tách hơn
  - Giúp tăng độ chính xác trong việc nhận diện và phân tách các ký tự số trong phần ảnh này.

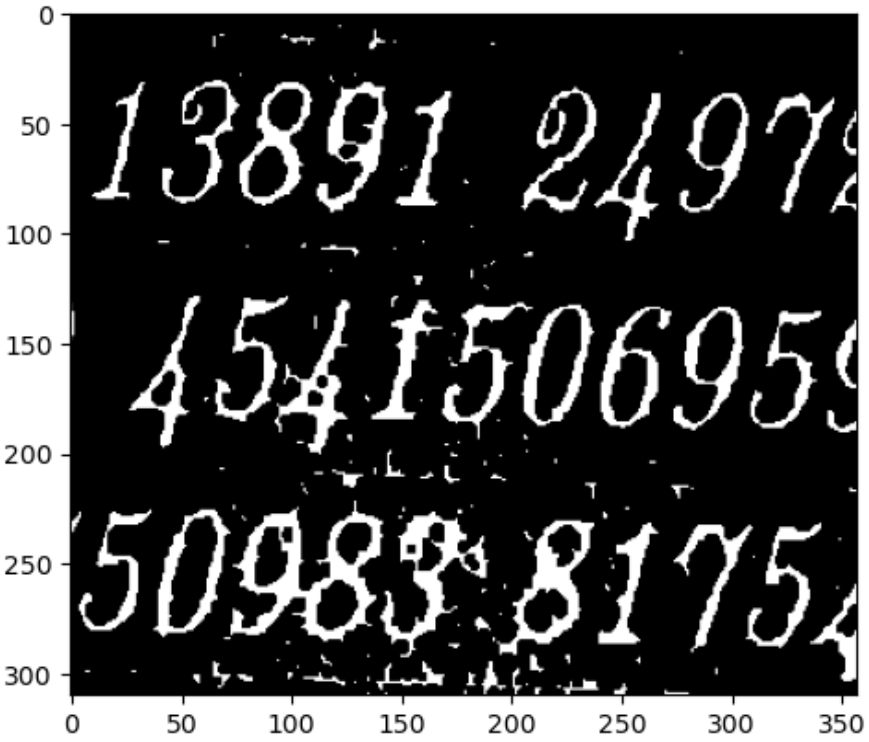


-

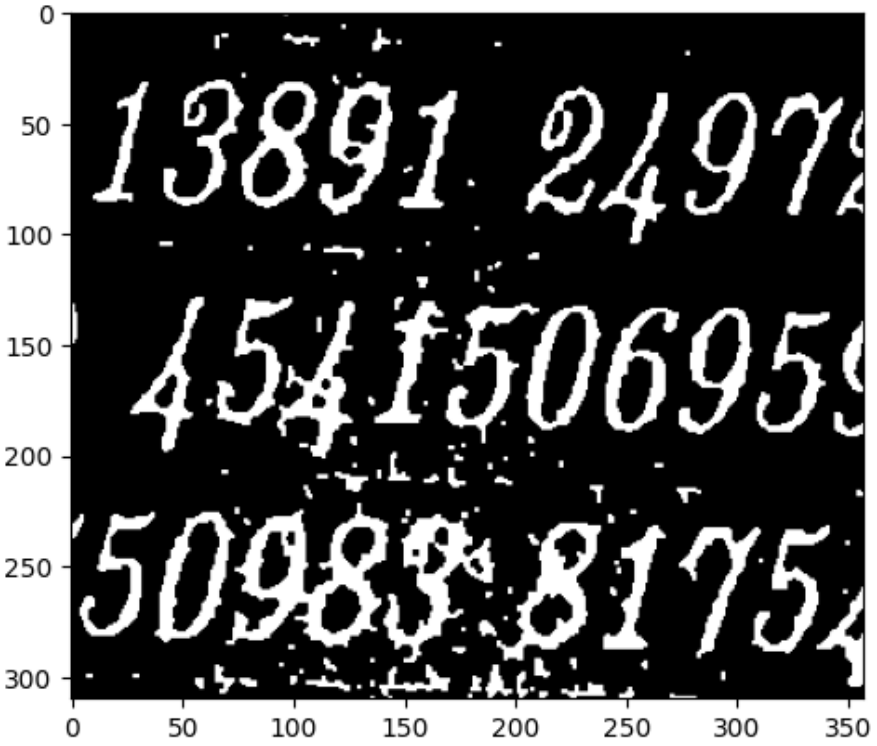
Hình 1.2.2.4.b - Phần bottom-right đã xử lý threshold



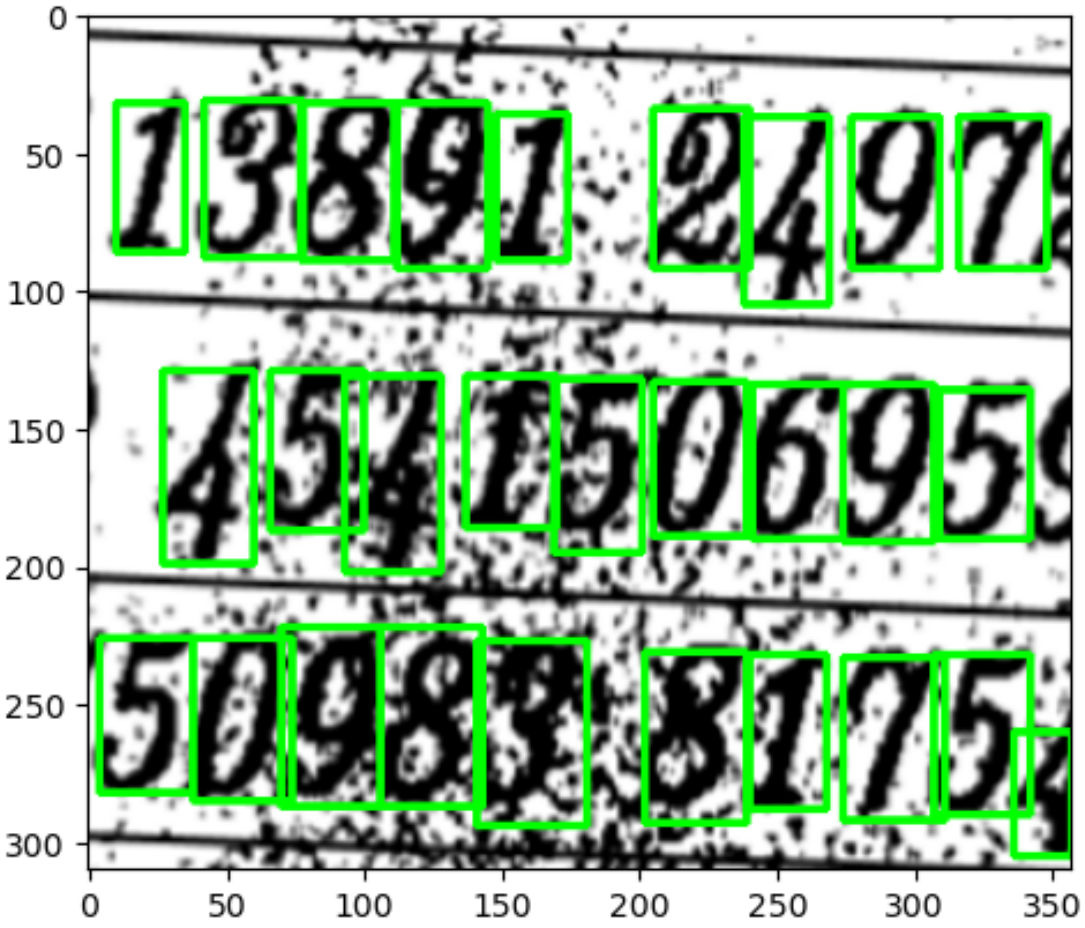
Hình 1.2.2.4.c - Phần bottom-right đã xử lý deliate lần 1



Hình 1.2.2.4.d - Phần bottom-right đã xử lý erosion



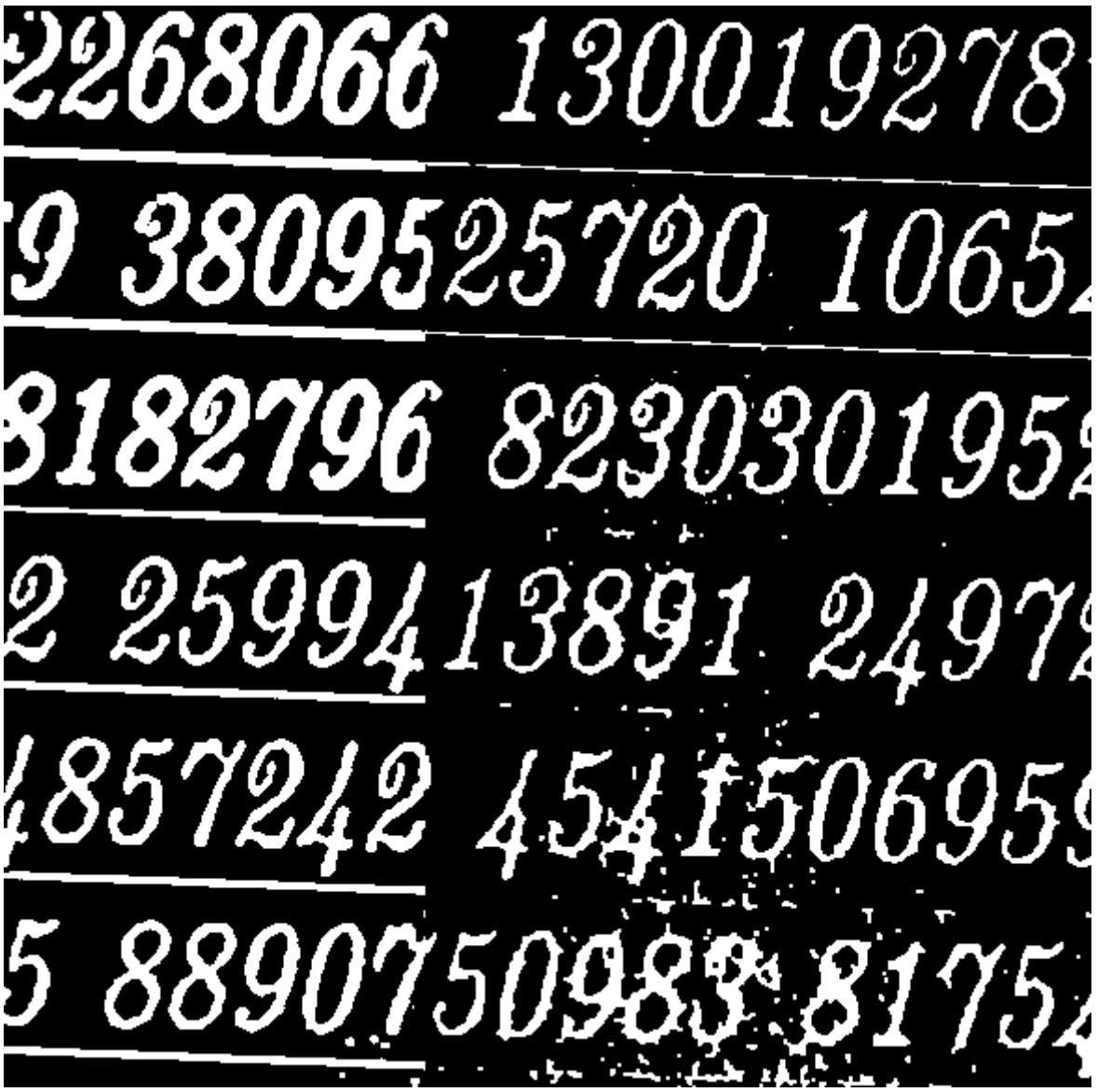
Hình 1.2.2.4.e - Phần bottom-right đã xử lý (image_processed)



Hình 1.2.2.4.f - Phần bottom-right đã xử lý viền

1.2.3 Ghép lại các phần ảnh đã xử lý

Các phần ảnh sau khi xử lý được ghép lại để tạo thành ảnh lớn ban đầu. Điều này giúp tổng hợp kết quả đã qua quá trình xử lý từng phần và chuẩn bị cho bước vẽ tự động các hình chữ nhật bao quanh từng chữ số



Hình 1.2.3.a - Ghép các ảnh nhị phân thành ảnh lớn

1.2.4 Kết quả

- Phát hiện đường viền: Sử dụng thuật toán phát hiện đường viền để xác định các khu vực chứa chữ số trong ảnh nhị phân
- Vẽ hình chữ nhật bao quanh các ký tự: Dựa trên các đường viền đã phát hiện, vẽ các hình chữ nhật bao quanh từng chữ số.

  Định nghĩa hàm contour_processing: dùng để vẽ tự động hình chữ nhật bao quanh các ký tự

- Bỏ qua các bounding box nếu height < 40 (đốm nhỏ) hoặc width > 200 (đường thẳng)
- Xử lý các bounding box chứa hai ký tự số: chia bounding box thành hai phần bằng nhau và vẽ hai hình chữ nhật bao quanh mỗi phần.

```
 if width > 65:
        width_half = width // 2
        x_half = x + width_half
        cv2.rectangle(image, (x, y), (x_half, y + height), (0, 255, 0), 2)
        cv2.rectangle(image, (x_half, y), (x + width, y + height), (0, 255, 0), 2)
        return
```

- Trường hợp còn lại 40 < width < 60:
    - Vẽ hình chữ nhật cho các ký tự số

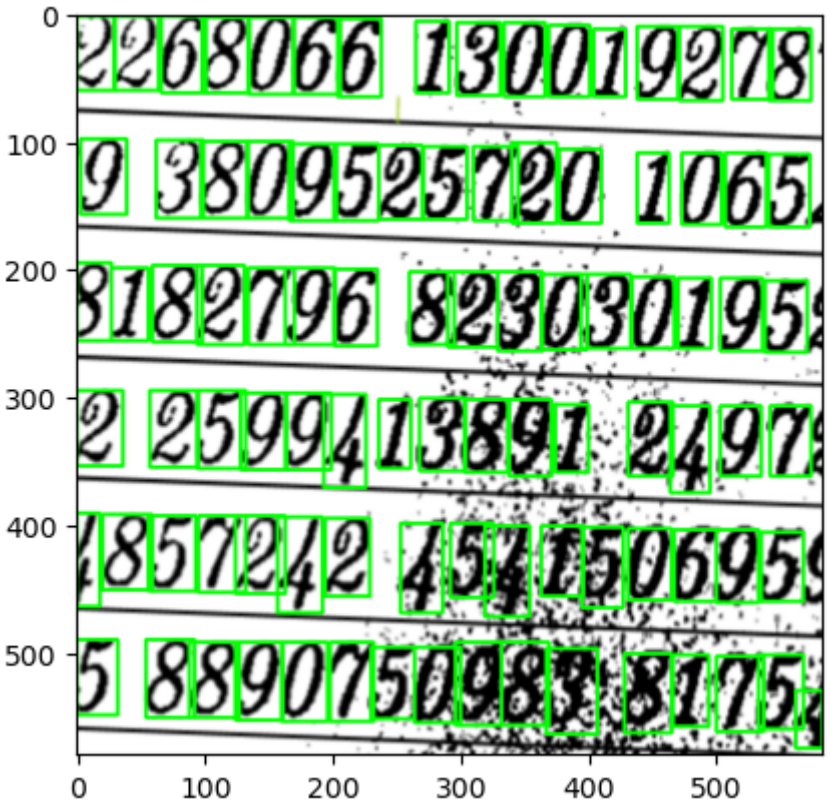
      ```
      cv2.rectangle(image, (x, y), (x + width, y + height), (0, 255, 0), 2)
      ```

    - Đối với các ký tự 3, 5, 8 ở bottom-right có cách xử lý đặc trưng

```
if 40 < width < 60:
        if TIMES == 0:  # Number 8
            cv2.rectangle(image, (x + 10, y), (x + width, y + height), (0, 255, 0), 2)
        elif TIMES == 1:  # Number 3
            cv2.rectangle(image, (x, y), (x + width - 10, y + height - 5), (0, 255, 0), 2)
        elif TIMES == 2:  # Number 5
            cv2.rectangle(image, (x, y), (x + width - 10, y + height), (0, 255, 0), 2)
```

```
        TIMES += 1
```

- Hiển thị kết quả và lưu lại file output



Hình 1.2.4 - Kết quả

CHƯƠNG 2. HIỆN THỰC HÓA BẰNG MÃ CODE

2.1 Đoạn mã đầy đủ để xử lý bài toán

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow


from google.colab import drive
drive.mount('/content/drive')


image_path = '/content/drive/My Drive/Colab
Notebooks/XLA/input.png'
path ='/content/drive/My Drive/Colab Notebooks/XLA/'


def contour_processing(x, y, width, height, image):
    """
    Objective: Process contours to handle various scenarios such as
small dots, lines, and bounding boxes.


    Parameters:
    - x, y : int
        The coordinates of the top-left corner of the rectangle.
    - width : int
        Width of the rectangle.
    - height : int
        Height of the rectangle.
    - image : np.ndarray
        The image on which to draw the rectangles.
```

```
    Returns:
    None
    """
    global TIMES

    # Ignore small dots
    if height < 40:
        return

    # Ignore lines
    if width > 200:
        return

    # Handle bounding boxes that contain two digits
    if width > 65:
        width_half = width // 2
        x_half = x + width_half
        cv2.rectangle(image, (x, y), (x_half, y + height), (0, 255,
0), 2)
        cv2.rectangle(image, (x_half, y), (x + width, y + height),
(0, 255, 0), 2)
        return

    # Handle bounding boxes for 5, 3, and 8 in the bottom right
image
    if 40 < width < 60:
        if TIMES == 0:  # Number 8
            cv2.rectangle(image, (x + 10, y), (x + width, y +
height), (0, 255, 0), 2)
        elif TIMES == 1:  # Number 3
```

```
            cv2.rectangle(image, (x, y), (x + width - 10, y +
height-5), (0, 255, 0), 2)
        elif TIMES == 2:  # Number 5
            cv2.rectangle(image, (x, y), (x + width - 10, y +
height), (0, 255, 0), 2)

        TIMES += 1
    else:
        cv2.rectangle(image, (x, y), (x + width, y + height), (0,
255, 0), 2)

def separate_image(image):
    """
    Objective: Separate the input image into four quadrants.

    Parameters:
    - image : np.ndarray
        The input image to be separated.

    Returns:
    tuple of np.ndarray
        The four separated quadrants of the image: top_left,
top_right, bottom_left, bottom_right.
    """
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    height, width = image_gray.shape

    # Cutting coordinates
    cut_x = 226
    cut_y = 270
```

```
    # Separate the image into 4 parts
    top_left = image_gray[0:cut_y, 0:cut_x]
    top_right = image_gray[0:cut_y, cut_x:width]
    bottom_left = image_gray[cut_y:height, 0:cut_x]
    bottom_right = image_gray[cut_y:height, cut_x:width]

    return top_left, top_right, bottom_left, bottom_right

def top_left_preprocessing(top_left):
    """
    Objective: Preprocess the top-left quadrant of the image.

    Parameters:
    - top_left : np.ndarray
        The top-left quadrant of the image.

    Returns:
    np.ndarray
        The preprocessed top-left quadrant of the image.
    """
    # Thresholding image
    _, image_binary = cv2.threshold(top_left, 150, 255,
cv2.THRESH_BINARY_INV)

    # Use morphological transformations to enhance image clarity
    kernel = np.ones((2, 2))
    image_processed = cv2.morphologyEx(image_binary,
cv2.MORPH_DILATE, kernel, iterations=2)
```

```
    plt.subplot(1, 2, 1)
    plt.title('image_binary')
    plt.imshow(image_binary, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title('image_dilated')
    plt.imshow(image_processed, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    return image_processed

def top_right_preprocessing(top_right):
    """
    Objective: Preprocess the top-right quadrant of the image.

    Parameters:
    - top_right : np.ndarray
        The top-right quadrant of the image.

    Returns:
    np.ndarray
        The preprocessed top-right quadrant of the image.
    """
    # Thresholding image
    _, image_binary = cv2.threshold(top_right, 50, 255,
cv2.THRESH_BINARY_INV)
```

```
    # Use morphological transformations to enhance image clarity
    kernel = np.ones((1, 1))
    image_opened = cv2.morphologyEx(image_binary, cv2.MORPH_OPEN,
kernel)
    image_processed = cv2.morphologyEx(image_opened,
cv2.MORPH_DILATE, kernel)

    plt.subplot(1, 3, 1)
    plt.title('image_binary')
    plt.imshow(image_binary, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.title('image_opened')
    plt.imshow(image_opened, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.title('image_dilated')
    plt.imshow(image_processed, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    return image_processed

def bottom_left_preprocessing(bottom_left):
    """
```

```
    Objective: Preprocess the bottom-left quadrant of the image.

    Parameters:
    - bottom_left : np.ndarray
        The bottom-left quadrant of the image.

    Returns:
    np.ndarray
        The preprocessed bottom-left quadrant of the image.
    """

    # Apply adaptive thresholding to invert the image
    thresholded = cv2.adaptiveThreshold(bottom_left, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 11, 2)

    # Define a horizontal kernel for morphological operations to
detect long horizontal lines
    horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
(25, 1))
    detect_lines = cv2.morphologyEx(thresholded, cv2.MORPH_OPEN,
horizontal_kernel, iterations=2)

    # The detected lines are white (255), we want to make them
black (0)
    mask_lines_black = cv2.bitwise_not(detect_lines)

    # Apply the mask to the thresholded image, turning only the
detected lines black
    result = cv2.bitwise_and(thresholded, mask_lines_black)

    kernel = np.ones((2, 2))
```

```
    image_opened = cv2.morphologyEx(result, cv2.MORPH_OPEN, kernel)


    kernel = np.ones((3, 3))
    image_processed = cv2.morphologyEx(image_opened,
cv2.MORPH_DILATE, kernel)


    plt.subplot(2, 2, 1)
    plt.title('image_binary')
    plt.imshow(thresholded, cmap='gray')
    plt.axis('off')


    plt.subplot(2, 2, 2)
    plt.title('image after remove lines')
    plt.imshow(result, cmap='gray')
    plt.axis('off')


    plt.subplot(2, 2, 3)
    plt.title('image_opened')
    plt.imshow(image_opened, cmap='gray')
    plt.axis('off')


    plt.subplot(2, 2, 4)
    plt.title('image_dilated')
    plt.imshow(image_processed, cmap='gray')
    plt.axis('off')


    plt.tight_layout()
    plt.show()


    return image_processed
```

```
def bottom_right_preprocessing(bottom_right):
    """
    Objective: Preprocess the bottom-right quadrant of the image.

    Parameters:
    - bottom_right : np.ndarray
        The bottom-right quadrant of the image.

    Returns:
    np.ndarray
        The preprocessed bottom-right quadrant of the image.
    """
    # Thresholding image
    _, image_binary = cv2.threshold(bottom_right, 25, 255,
cv2.THRESH_BINARY_INV)

    kernel = np.ones((2, 2))
    image_dilated1 = cv2.morphologyEx(image_binary,
cv2.MORPH_DILATE, kernel, iterations=1)

    kernel = np.ones((3, 3))
    image_eroded = cv2.erode(image_dilated1, kernel, iterations=1)

    kernel = np.ones((2, 2))
    image_processed = cv2.morphologyEx(image_eroded,
cv2.MORPH_DILATE, kernel, iterations=1)

    plt.subplot(2, 2, 1)
    plt.title('image_binary')
```

```
    plt.imshow(image_binary, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 2)
    plt.title('image_dilated')
    plt.imshow(image_dilated1, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 3)
    plt.title('image_eroded')
    plt.imshow(image_eroded, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.title('image_dilated')
    plt.imshow(image_processed, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    return image_processed

if __name__ == "__main__":

    TIMES = 0
    image_original = cv2.imread(image_path)
    top_left, top_right, bottom_left, bottom_right =
separate_image(image_original)
```

```
    print("Original, after seperate 4
pictures------------------------------")
    plt.subplot(2, 2, 1)
    plt.title('Top Left')
    plt.imshow(top_left, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 2)
    plt.title('Top Right')
    plt.imshow(top_right, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 3)
    plt.title('Bottom left')
    plt.imshow(bottom_left, cmap='gray')
    plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.title('Bottom right')
    plt.imshow(bottom_right, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Processed
    print("\n\n Top left
processed,------------------------------")
    top_left_processed = top_left_preprocessing(top_left)
```

```
    print("\n\n Top right
processed,------------------------------")
    top_right_processed = top_right_preprocessing(top_right)


    print("\n\n Bottom left
processed,------------------------------")
    bottom_left_processed = bottom_left_preprocessing(bottom_left)


    print("\n\n Bottom right
processed,------------------------------")
    bottom_right_processed =
bottom_right_preprocessing(bottom_right)


    print("\n\n Four pictures
processed,------------------------------")
    # Print 4 pictures after processed
    plt.figure(figsize=(15, 10))


    plt.subplot(2, 2, 1)
    plt.title('Top Left')
    plt.imshow(top_left_processed, cmap='gray')
    plt.axis('off')


    plt.subplot(2, 2, 2)
    plt.title('Top Right')
    plt.imshow(top_right_processed, cmap='gray')
    plt.axis('off')


    plt.subplot(2, 2, 3)
    plt.title('Bottom left')
    plt.imshow(bottom_left_processed, cmap='gray')
```

```
    plt.axis('off')


    plt.subplot(2, 2, 4)
    plt.title('Bottom right')
    plt.imshow(bottom_right_processed, cmap='gray')
    plt.axis('off')


    plt.tight_layout()
    plt.show()



    # Combine 4 images after preprocessing
    top_combined = cv2.hconcat([top_left_processed,
top_right_processed])
    bottom_combined = cv2.hconcat([bottom_left_processed,
bottom_right_processed])


    # Concatenate images vertically
    result = cv2.vconcat([top_combined, bottom_combined])


    # Display the result
    print("\n\n Combine 4 images,------------------------------")
    cv2_imshow(result)


    # Find contours from the processed image
    contours, _ = cv2.findContours(result, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


    # Process contours one by one
    for contour in contours:
```

```
        x, y, w, h = cv2.boundingRect(contour)
        contour_processing(x, y, w, h, image_original)


print("\n\n Result,--------------------------------")
plt.title('Result')
plt.imshow(image_original)
cv2.imwrite(path+"output.jpg", image_original)
cv2.waitKey(0)
```

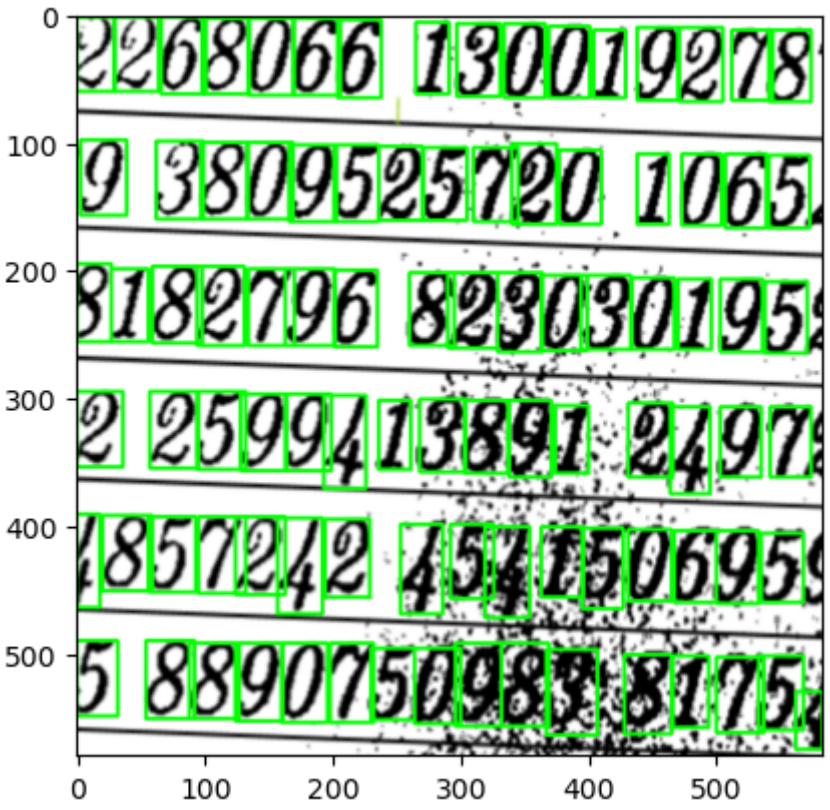
2.2 Môi trường cần để thực thi đoạn mã

- Môi trường thực thi: Google Colab
- Ngôn ngữ lập trình: Python

CHƯƠNG 3. KẾT QUẢ

Các phần nhiễu đã được giảm bớt đáng kể sau khi áp dụng các phép biến đổi hình thái học. Tuy nhiên, vẫn còn một số nhiễu nhỏ xung quanh một vài ký tự, đặc biệt là ở phần ảnh "bottom-right", nhưng không ảnh hưởng lớn đến kết quả tổng thể. Hầu hết các chữ số trong hình ảnh đều được bao quanh chính xác bởi các hình chữ nhật. Các ký tự bị đứt nét hoặc nhiễu đã được xử lý tốt, giúp cải thiện độ chính xác trong việc phát hiện và bao quanh các chữ số.

Hình ảnh đầu ra “output.jpg” (Hình 5) cho thấy các chữ số trong ảnh đầu vào đã được bao quanh bởi các hình chữ nhật màu xanh lá cây, giúp xác định rõ vị trí và kích thước của từng chữ số. Kết quả này chứng tỏ rằng các bước tiền xử lý, áp dụng biến đổi hình thái học và phát hiện đường viền đã hoạt động hiệu quả để đạt được mục tiêu của bài toán.



Hình 1.2.4 - Kết quả

TÀI LIỆU THAM KHẢO

OpenCV Documentation: https://docs.opencv.org/

Python Imaging Library (PIL): https://pillow.readthedocs.io/

Tài liệu về Morphological Transformations: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html