HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 2

BÁO CÁO CUỐI KÌ

Đề tài: Sudoku Solver

- Môn học:	**Xử lý ảnh(INT13146)**		
- Giảng viên hướng dẫn :	**ThS.Huỳnh Trung Trụ**		
- Nhóm:	**2**		
- Thành viên:	**N20DCCN036**	**-**	**Nguyễn Phúc Luân**
	N20DCCN062	**-**	**Đặng Thành Tân**
	N20DCCN073	**-**	**Phạm Đức Thắng**

Thành phố Hồ Chí Minh, tháng 12.

Mục Lục

Chương I. GIỚI THIỆU CHUNG.

1. Tổng quan.

- Sudoku puzzles là một loại trò chơi câu đố logic tăng trí tuệ, game yêu cầu người giải đố tìm ra câu trả lời và đặt các con số tự nhiên (từ 1 đến 9) vào các ô vuông. Bảng giải đố là một hình vuông lưới 9x9 có các khối con 3x3. Ban đầu, một vài con số sẽ được điền vào các ô vuông. Để hoàn thành bảng giải đố, người chơi cần đặt các con số vào ô vuông sao cho mỗi cột, mỗi hàng và mỗi khối con đều chứa tất cả số từ 1 đến 9.

2. Lý do chọn đề tài.

- Câu đố Sudoku thường thấy trên báo và sách giải đố nhưng đôi khi giải pháp không có sẵn. Mặc dù có thể nhập thủ công các số đã cho vào một chương trình hoặc ứng dụng web để giải câu đố.Tuy nhiên, việc sử dụng sẽ nhanh hơn và thuận tiện hơn nhiều nếu sử dụng xử lý hình ảnh để tự động phát hiện và nhận dạng các giá trị trong câu đố. Ứng dụng này được thiết kế để yêu cầu sự tương tác tối thiểu từ người dùng: chỉ cần hướng máy ảnh vào câu đố hoặc đưa trực tiếp ảnh chứa câu đố ứng dụng thì các giá trị giải pháp tự động xuất hiện trong các ô trống.

3. Định nghĩa vấn đề .

❖ Các vấn đề cần giải quyết:

- *Input*: Chương trình phải chấp nhận lưới 9x9 làm đầu vào, thể hiện câu đố Sudoku đã được lấp đầy một phần. Lưới đầu vào phải chứa các số từ 1 đến 9, với 0 hoặc các ô trống biểu thị khoảng trống.
- *Validation*: Chương trình phải xác thực lưới đầu vào để đảm bảo nó tuân theo các quy tắc của Sudoku. Mỗi hàng, cột và lưới con 3x3 phải chứa tất cả các số từ 1 đến 9 mà không lặp lại. Nếu lưới đầu vào vi phạm quy tắc Sudoku, chương trình sẽ chỉ ra đầu vào không hợp lệ.
- *Thuật toán giải Sudoku*: Chương trình thực hiện thuật toán quay lui để tìm lời giải cho câu đố Sudoku. Nó khám phá một cách có hệ thống các kết hợp số khác nhau bằng cách đặt các số hợp lệ vào các ô trống và quay lại bất cứ khi nào đạt được cấu hình không hợp lệ.
- *Solution Output*: Sau khi giải thành công câu đố Sudoku, chương trình sẽ đưa ra giải pháp hoàn chỉnh. Đầu ra phải là lưới 9x9 trong đó tất cả các ô trống chứa các số hợp lệ, đáp ứng các quy tắc Sudoku.
- *Efficiency*: Chương trình giải phải tìm ra lời giải cho các câu đố Sudoku ở các mức độ khó khác nhau một cách hiệu quả. Nó sẽ giảm thiểu các tính toán không cần thiết và cung cấp phản hồi kịp thời cho cả câu đố đơn giản và phức tạp.
- *Giao diện người dùng*: Để nâng cao trải nghiệm người dùng, chương trình có thể bao gồm giao diện người dùng nơi người dùng có thể nhập câu đố Sudoku, xem lời giải và tương tác với thuật toán giải. Giao diện người dùng có thể cung cấp các tính năng như gợi ý, kiểm tra lỗi và khả năng nhập câu đố từng bước.

* Mục tiêu của chương trình giải Sudoku là cung cấp giải pháp tự động cho Sudoku câu đố, cho phép người dùng giải câu đố một cách nhanh chóng và dễ dàng. Chương trình nên mang lại kết quả chính xác và hiệu quả trong khi vẫn duy trì tính toàn vẹn của các quy tắc Sudoku.

4. Mục tiêu thiết kế.

- Giải các câu đố Sudoku chính xác và hiệu quả.
- Đảm bảo đầu vào hợp lệ và loại bỏ các câu đố không hợp lệ hoặc không thể giải được.
- Cung cấp giao diện người dùng trực quan và tương tác.
- Tối ưu hóa thuật toán giải nhanh.
- Xử lý các câu đố không hợp lệ hoặc không thể giải được bằng phản hồi thích hợp.
- Hỗ trợ nhiều cấp độ khó cho các thử thách đa dạng.

5. Ứng dụng.

- Trò chơi giải Sudoku có nhiều ứng dụng khác nhau phục vụ cho những người đam mê giải đố, các tổ chức giáo dục và bất kỳ ai muốn cải thiện kỹ năng giải quyết vấn đề của mình. Đây là một số ứng dụng chính của trò chơi giải Sudoku:

  + *Giải trí*: Trò chơi giải Sudoku mang đến một hoạt động thú vị và hấp dẫn cho mọi người ở mọi lứa tuổi. Nó cung cấp một cách giải câu đố đầy thử thách trải nghiệm có thể tận hưởng trong thời gian rảnh rỗi hoặc như một bài tập tinh thần để thư giãn và nghỉ ngơi.

  + *Phát triển Kỹ năng*: Trò chơi giải Sudoku đóng vai trò như một công cụ có giá trị để phát triển và rèn luyện các kỹ năng giải quyết vấn đề. Nó tăng cường khả năng suy luận logic, phản biện tư duy, nhận dạng mẫu và khả năng lập kế hoạch chiến lược. Tham gia thường xuyên trong việc giải Sudoku có thể giúp cải thiện kỹ năng nhận thức và sự nhanh nhẹn về tinh thần.

  + *Công cụ giáo dục*: Trò chơi giải Sudoku được ứng dụng trong môi trường giáo dục, chẳng hạn như trường học và cơ sở giáo dục. Nó có thể được sử dụng để giới thiệu và dạy các khái niệm như logic, lý luận suy diễn và các mẫu số. Nó khuyến khích học sinh suy nghĩ phân tích và tăng cường khả năng giải quyết vấn đề của họ.

  + *Phát triển cá nhân*: Trò chơi giải Sudoku thúc đẩy sự phát triển cá nhân bằng cách bồi dưỡng tính kiên trì, kiên nhẫn và khả năng xử lý thử thách. Nó khuyến khích cá nhân vượt qua trở ngại, học hỏi từ sai lầm và phấn đấu liên tục sự cải tiến. Trò chơi nuôi dưỡng tư duy phát triển và cảm giác thành tựu khi giải thành công các câu đố.

  + *Rèn luyện trí não*: Trò chơi giải Sudoku thường được sử dụng như một bài tập rèn luyện trí não để giữ cho đầu óc luôn nhạy bén và năng động. Luyện tập thường xuyên giúp duy trì tinh thần thị lực và duy trì trí nhớ. Nó cung cấp một bài tập kích thích cho não và có thể là một phần của chế độ rèn luyện trí não toàn diện.

- Tóm lại, trò chơi giải Sudoku có các ứng dụng linh hoạt, từ giải trí và phát triển kỹ năng đến mục đích giáo dục và phát triển cá nhân. Nó mang lại trải nghiệm thú vị và bổ ích, thử thách trí óc, thúc đẩy khả năng giải quyết vấn đề kỹ năng, nâng cao khả năng nhận thức.

- Trực quan hóa đầu ra.

Bảng 1.1: Bảng tóm tắt các thuộc tính được đề cập trong đề tài.

Tên thuộc tính	Giải thích cách giải quyết
Độ sâu kiến thức.	Làm quen với lý luận logic, mẫu số và kỹ thuật giải quyết vấn đề là điều cần thiết để giải các câu đố Sudoku một cách hiệu quả.
Phạm vi các yêu cầu xung đột.	Cân bằng giữa hiệu quả và độ chính xác, đồng thời cung cấp các tính năng thân thiện với người dùng và duy trì sự tuân thủ các quy tắc Sudoku, đưa ra một loạt các yêu cầu xung đột trong việc giải Sudoku.
Độ sâu phân tích.	Độ sâu phân tích để giải Sudoku liên quan đến việc sử dụng các kỹ thuật suy luận logic tiên tiến, bao gồm các mô hình phức tạp và sự phụ thuộc lẫn nhau, để khám phá các mối quan hệ ẩn giấu và đưa ra lựa chọn chính xác.
Làm quen với các vấn đề.	Phát triển sự quen thuộc với các vấn đề thường gặp khi giải Sudoku, chẳng hạn như dữ liệu đầu vào không hợp lệ, câu đố không thể giải và thuật toán hiệu quả, giúp đảm bảo các giải pháp chính xác và trải nghiệm giải quyết suôn sẻ
Mức độ áp dụng thuật toán.	Các thuật toán có thể áp dụng để giải Sudoku bao gồm các thuật toán quay lui và triển khai giao diện người dùng.
Mức độ tham gia của các bên liên quan và các yêu cầu xung đột.	Cân bằng đầu vào và kỳ vọng của các bên liên quan khác nhau đồng thời quản lý các yêu cầu xung đột để tạo ra giải pháp giải Sudoku đáp ứng các sở thích và mục tiêu khác nhau của người dùng.
Sự phụ thuộc lẫn nhau.	Việc giải thành công các câu đố Sudoku phụ thuộc vào sự phụ thuộc lẫn nhau giữa suy luận logic, nhận dạng mẫu và khả năng loại bỏ các khả năng thông qua thử nghiệm lặp đi lặp lại và quay lui.

Chương II. THIẾT KẾ, PHÁT TRIỂN, TRIỂN KHAI ĐỀ TÀI.

1. Tổng quan.

- Dự án Giải Sudoku là một ứng dụng thân thiện với người dùng tập trung vào bảng lưới 9x9. Nó cung cấp một giao diện trực quan để giải các câu đố Sudoku, khi người dùng di chuyển câu đố Sudoku đến camera thì ứng dụng sẽ nhận biết và xử lý câu đố.Ứng dụng sử dụng các thuật toán hiệu quả để xác định giải pháp, cập nhật lưới một cách linh hoạt. Ứng dụng ưu tiên trải nghiệm người dùng và thiết kế giao diện thuận tiện cho những người đam mê Sudoku.

2. Chi tiết đề tài.

- Các tính năng chính:

+ *Nhận câu đố từ camera*: Ứng dụng sẽ có camera trực tiếp để người dùng thuận tiện trong việc đưa ra câu đố cho ứng dụng. Người dùng chỉ cần di chuyển câu đố tới camera thì chương trình sẽ tự động nhận biết các ô số trong bảng Sudoku, chương trình sẽ nhận biết chữ số trong bảng và tiến hành đi tìm kiếm lời giải.

+ *Giải câu đố Sudoku*: sử dụng thuật toán hiệu quả để giải câu đố Sudoku được nhận từ đầu vào. Nó sử dụng kỹ thuật quay lui để khám phá các giải pháp khả thi và tìm ra giải pháp chính xác đáp ứng tất cả các quy tắc Sudoku. Trình giải cũng xác định các câu đố không hợp lệ hoặc không thể giải được, cung cấp phản hồi thích hợp cho người dùng.

3. Quá trình thực hiện.

- Ứng dụng sử dụng giao diện đồ họa người dùng (GUI) bao gồm bảng lưới 9x9 đại diện cho câu đố Sudoku được lấy trực tiếp từ camera. Mỗi ô trong lưới có thể là 1 trong các số từ 1 đến 9 hoặc để trống. Giao diện trực quan và thân thiện với người dùng tín hiệu trực quan để người dùng dễ dàng sử dụng.

3.1. Xử lý ảnh đầu vào.

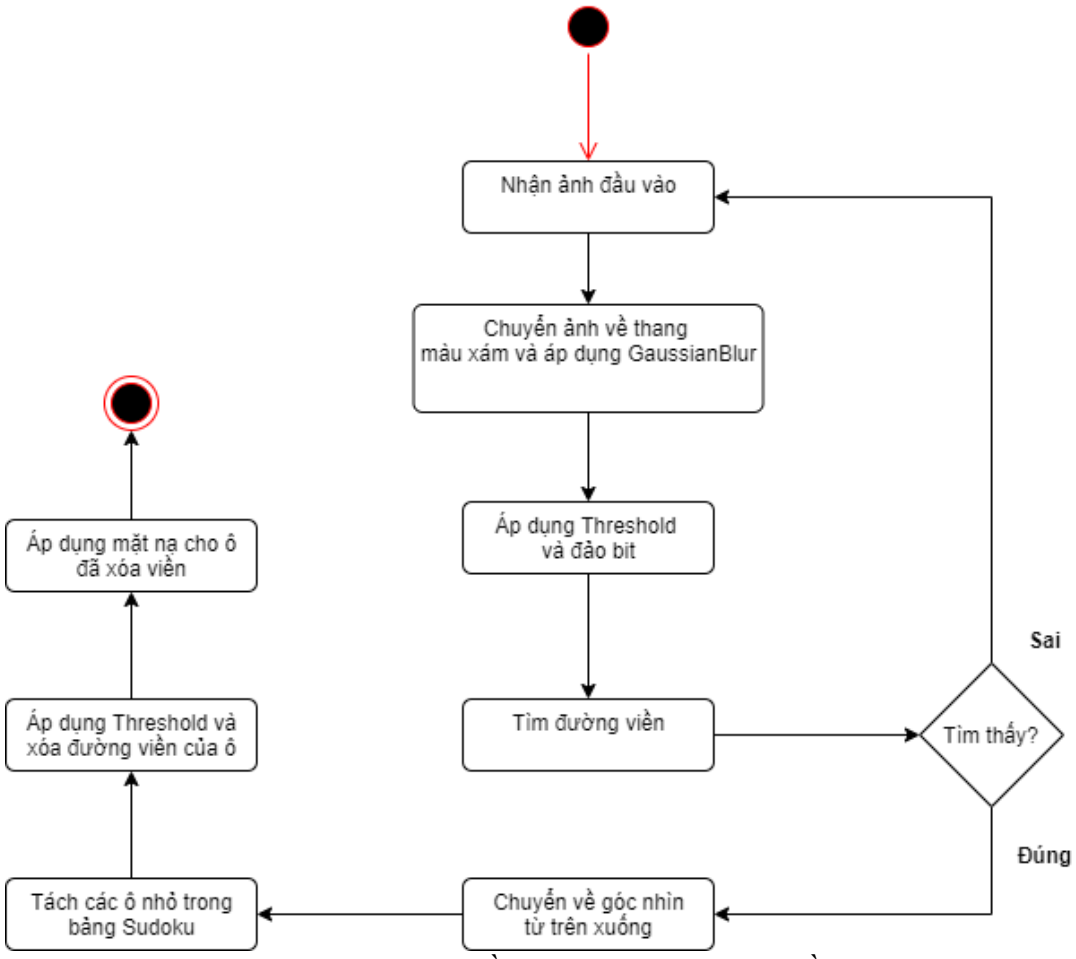
a. Tổng quan.

- Ứng dụng được thiết kế với camera trực tiếp, có nghĩa là màn hình chính của ứng dụng sẽ là video nhận được từ camera. Khi người dùng di chuyển bảng câu đố Sudoku tới camera thì chương trình sẽ nhận biết bảng số và thực hiện quá trình xử lý ảnh để nhận biết các ô có chữ số, cũng như các ô trống để tìm lời giải.

b. Thư viện sử dụng.

- OpenCV: Sử dụng thư viện này để chuyển ảnh đầu vào thành màu xám, giảm nhiễu, áp dụng Threshold, tìm ra đường viền lớn nhất của Sudoku.
- four_point_transform: Thư viện này dùng để đổi góc nhìn của vào, tức là khi người dùng đưa ảnh đầu vào bị lệch sang 1 góc nào đó, thì thư viện này sẽ chuyển về góc nhìn trực diện để thuận tiện trong việc xử lý cũng như hiển thị lời giải cho người dùng.
- clear_border: Thư viện này dùng để xóa các đường viền của các ô con trong bảng Sudoku.

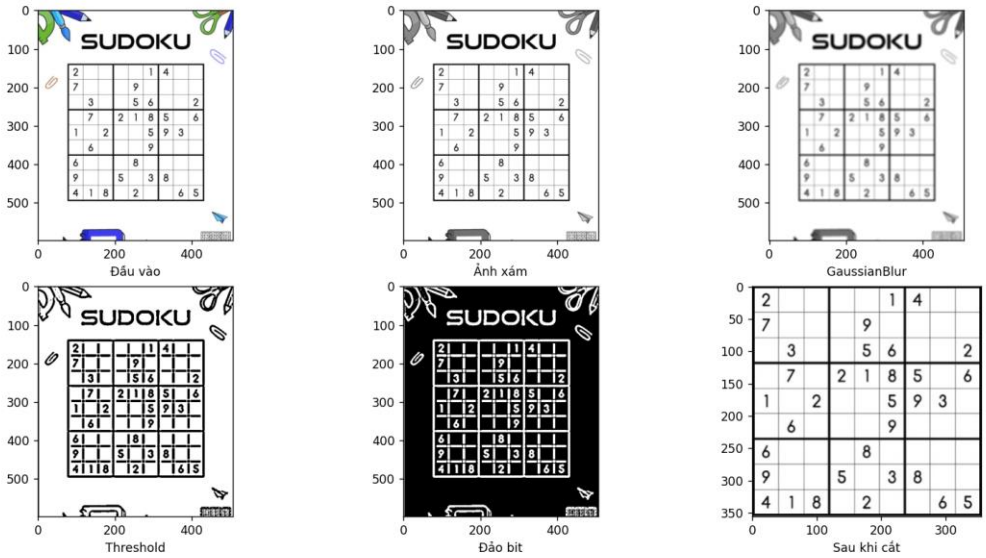
c. Sơ đồ hoạt động xử lý ảnh đầu vào.



Hình 2.1 Sơ đồ hoạt động xử lí ảnh đầu vào.

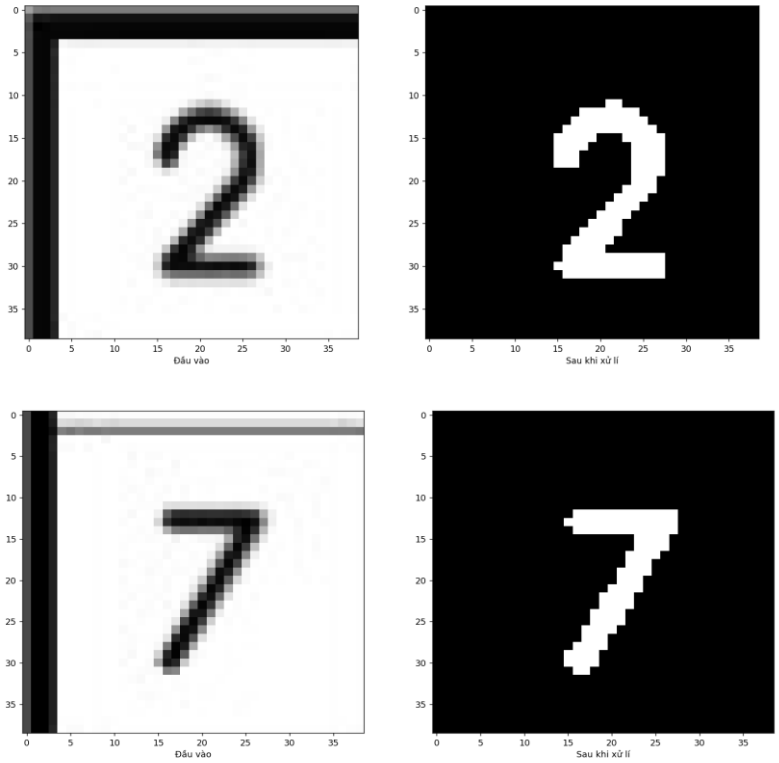
d. Khái quát các bước.

- Nhận vào ảnh và xử lý ảnh để ra ô vuông Sudoku.



Hình 2.2 Hình ảnh thể hiện các bước xử lí để lấy ô Sudoku.

- Nhận vào ô vuông nhỏ và xử lý đường viền cho ô vuông đó, nhận ra là ảnh đã xử lý.



Hình 2.3 Hình ảnh xử lí các ô vuông nhỏ.

e. Triển khai các bước.

- Nhận ảnh đầu vào: Sử dụng cv2.imread() để đọc hình ảnh, đầu vào có thể là ảnh người dùng chọn từ bên ngoài, hoặc là lấy từ video khi người dùng di chuyển bảng Sudoku tới.
- Chuyển ảnh về thang màu xám và áp dụng Gaussian Blur để giảm nhiễu:
  + Chuyển về thang màu xám: Sử dụng cv2.cvtColor(image, cv2.COLOR_BGR2GRAY), với image là ảnh đầu vào, và COLOR_BGR2GRAY để chuyển ảnh thành màu xám.

Hình 2.4 Ảnh đầu vào.

Hình 2.5 Ảnh sau khi chuyển về thang màu xám.

+ Áp dụng Gaussian Blur: sử dụng blurred = cv2.GaussianBlur(gray, (7, 7), 3).

+ (7, 7): Đây là kích thước của kernel (hoặc bộ lọc) Gaussian. Trong trường hợp này, kernel có kích thước là 7x7. Kích thước của kernel ảnh hưởng đến mức độ làm mờ của ảnh. Kernel lớn hơn sẽ tạo ra hiệu ứng làm mờ mạnh hơn.

+ 3: Đây là độ lệch chuẩn (standard deviation) của Gaussian. Nó ảnh hưởng đến mức độ làm mờ. Giá trị lớn hơn của độ lệch chuẩn sẽ tạo ra hiệu ứng làm mờ mạnh hơn.

+ Kết quả của dòng code này là ảnh blurred, là phiên bản làm mờ của ảnh xám gray sử dụng bộ lọc Gaussian. Hiệu ứng làm mờ giúp giảm nhiễu và tạo ra ảnh có độ tương phản mịn hơn, thích hợp cho nhiều ứng dụng trong xử lý ảnh như nhận diện đối tượng, trích xuất đặc trưng, và các công việc khác.

Hình 2.6 Ảnh sau khi áp dụng GaussianBlur.

- Áp dụng Threshold và đảo bit:

+ Áp dụng Threshold: thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2) sử dụng phương pháp ngưỡng hóa thích ứng để tạo ra một ảnh nhị phân từ ảnh đã được làm mờ.

* 255: Đây là giá trị ngưỡng tối đa, được sử dụng để đặt giá trị của các pixel trong ảnh nhị phân. Trong trường hợp này, pixel có giá trị lớn hơn hoặc bằng 255 sẽ được đặt thành 255 (trắng), và pixel có giá trị nhỏ hơn 255 sẽ được đặt thành 0 (đen).

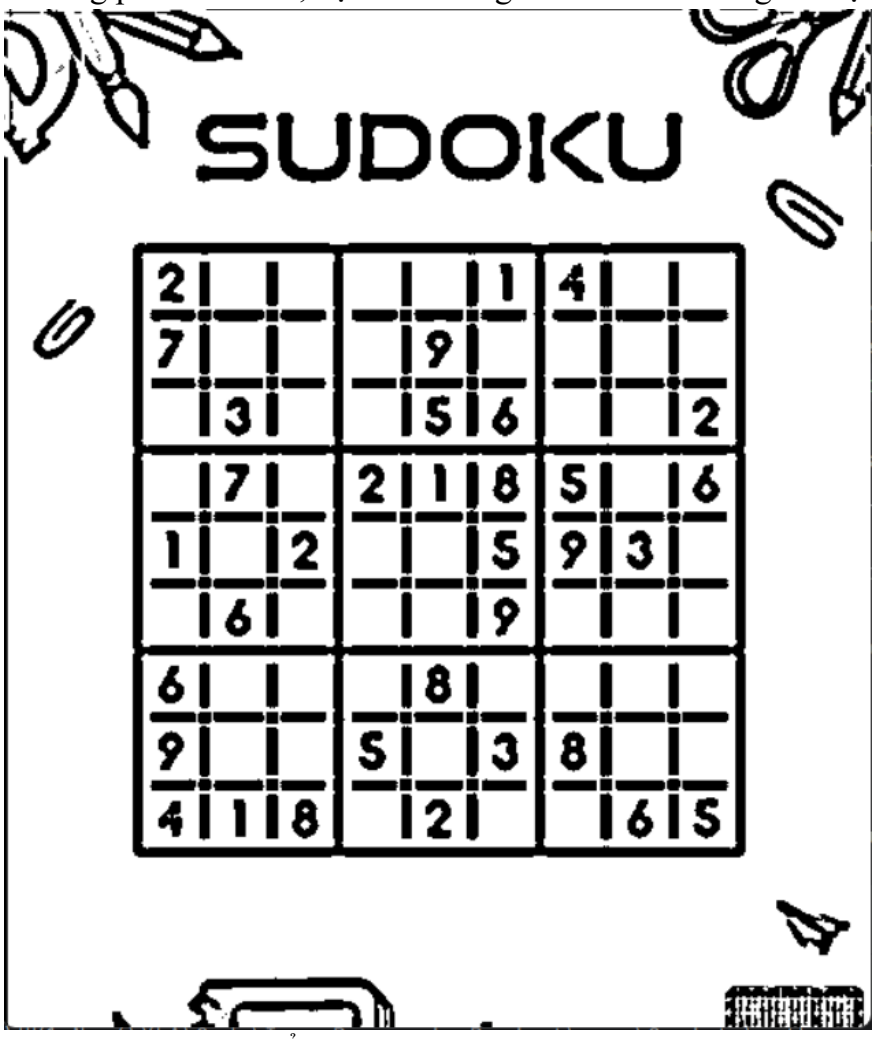
* cv2.ADAPTIVE_THRESH_GAUSSIAN_C: Đây là phương pháp ngưỡng hóa thích ứng sử dụng trung bình của các vùng lân cận. Trong trường hợp này, được sử dụng bộ lọc Gaussian để tính toán trung bình.

* cv2.THRESH_BINARY: Đây là loại ngưỡng hóa, trong trường hợp này, là ngưỡng nhị phân. Nghĩa là, giá trị pixel lớn hơn ngưỡng sẽ được đặt thành giá trị tối đa (255), và giá trị pixel nhỏ hơn ngưỡng sẽ được đặt thành giá trị tối thiểu (0).

* 11: Đây là kích thước của khu vực lân cận được sử dụng để tính toán ngưỡng. Kích thước này đặc định kích thước của vùng lân cận xung quanh mỗi pixel.

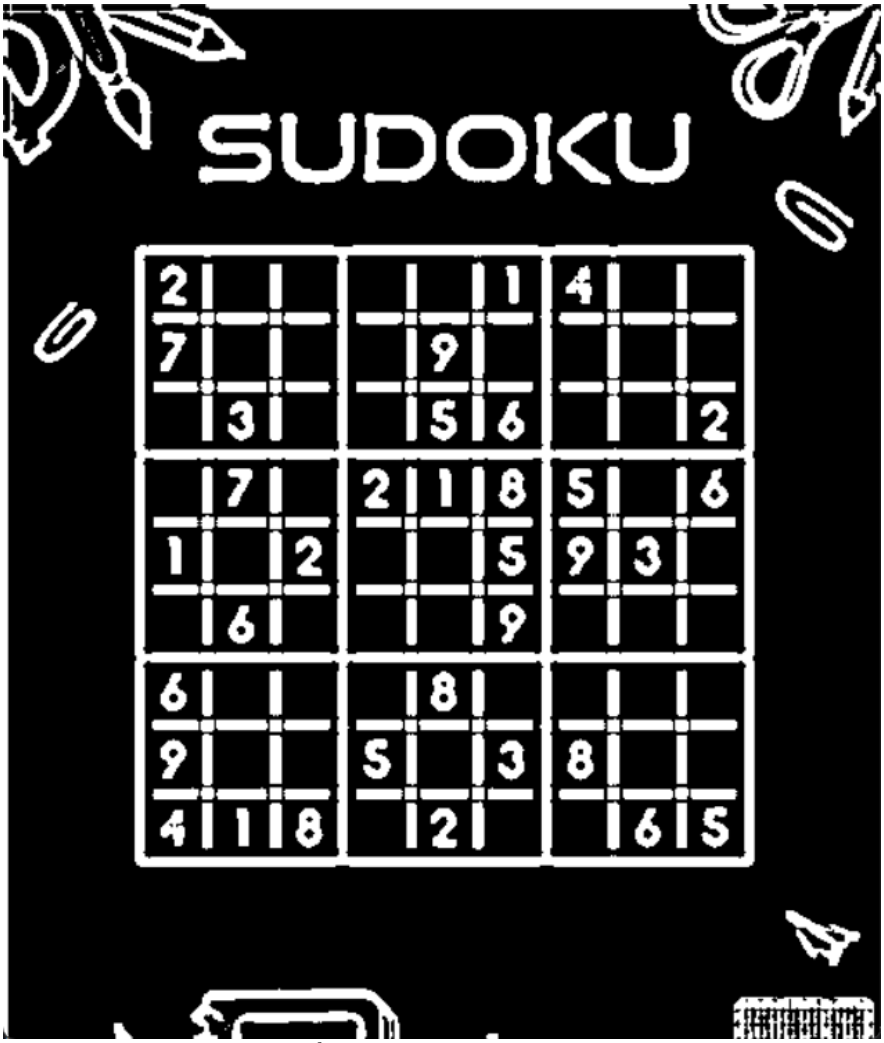
* 2: Đây là hằng số trừ đi từ trung bình tính toán. Nó giúp kiểm soát độ chệch giữa ngưỡng tính toán và giá trị thực tế của pixel.

* Kết quả của dòng code này là ảnh nhị phân, trong đó ngưỡng được xác định tự động cho từng phần của ảnh, dựa trên trung bình của các vùng lân cận.



Hình 2.7 Ảnh sau khi áp dụng Threshold.

+ Đảo bit: Sử dụng thresh = cv2.bitwise_not(thresh) để đảo các bit trong ảnh nhị phân, tức là các bit 1 sẽ về 0, và ngược lại. Điều này giúp dễ dàng hơn trong việc tìm đường viền của Sudoku.



Hình 2.8 Ảnh sau khi áp dụng đảo bit.

- Tìm đường viền: Đoạn code sau được sử dụng để tìm và xác định đường viền của một đối tượng trong ảnh nhị phân (ảnh chỉ chứa các giá trị 0 và 255), đặc biệt là một đối tượng có hình dạng gần giống hình vuông.

```
#Tim duong vien
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
puzzleCnt = []
for c in cnts:
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)
    if (len(approx) == 4):
        puzzleCnt = approx
        break
if len(puzzleCnt) == 0:
    return (None, None, puzzleCnt)
```

+ Sử dụng hàm cv2.findContours() để tìm các đường viền trong ảnh nhị phân thresh. Tham số cv2.RETR_EXTERNAL chỉ tìm các đường viền ngoại vi của các đối tượng và cv2.CHAIN_APPROX_SIMPLE chỉ giữ lại các điểm đầu cuối của đường viền thay vì giữ toàn bộ đường viền.

+ Sử dụng hàm imutils.grab_contours để trích xuất đường viền từ đối tượng trả về bởi cv2.findContours.

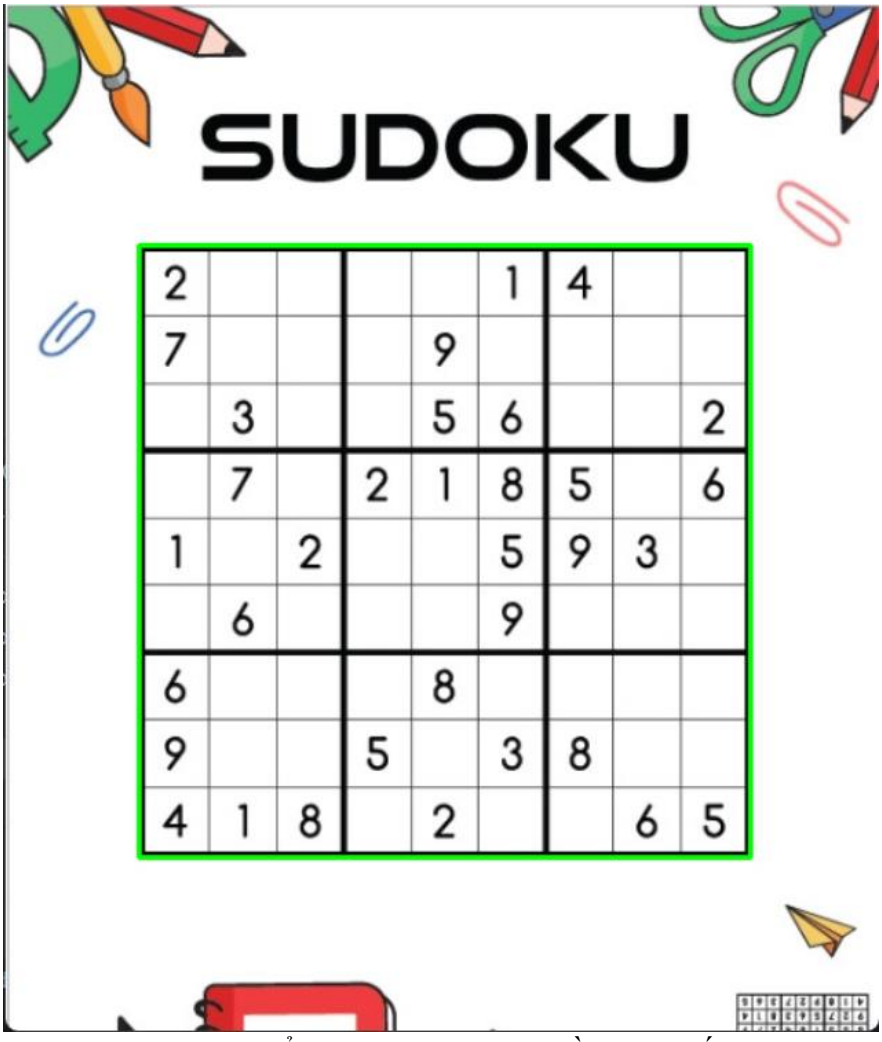
+ cnts = sorted(cnts, key=cv2.contourArea, reverse=True): Sắp xếp các đường viền theo diện tích giảm dần, để đảm bảo rằng đường viền lớn nhất (đối tượng chính) sẽ được xử lý đầu tiên.

+ peri = cv2.arcLength(c, True): Tính chiều dài của đường viền bằng hàm cv2.arcLength.

+ approx = cv2.approxPolyDP(c, 0.02 * peri, True): Xấp xỉ đa giác đối tượng bằng hàm cv2.approxPolyDP. Tham số 0.02 * peri là độ chấp nhận được cho việc xấp xỉ.

+ if (len(approx) == 4):: Kiểm tra xem có phải là đa giác có 4 đỉnh hay không. Trong trường hợp này, đa giác có thể là hình vuông.

+ Trường hợp nếu là đa giác có 4 đỉnh thì lưu lại và thoát vòng lặp, ngược lại thì trả về None.

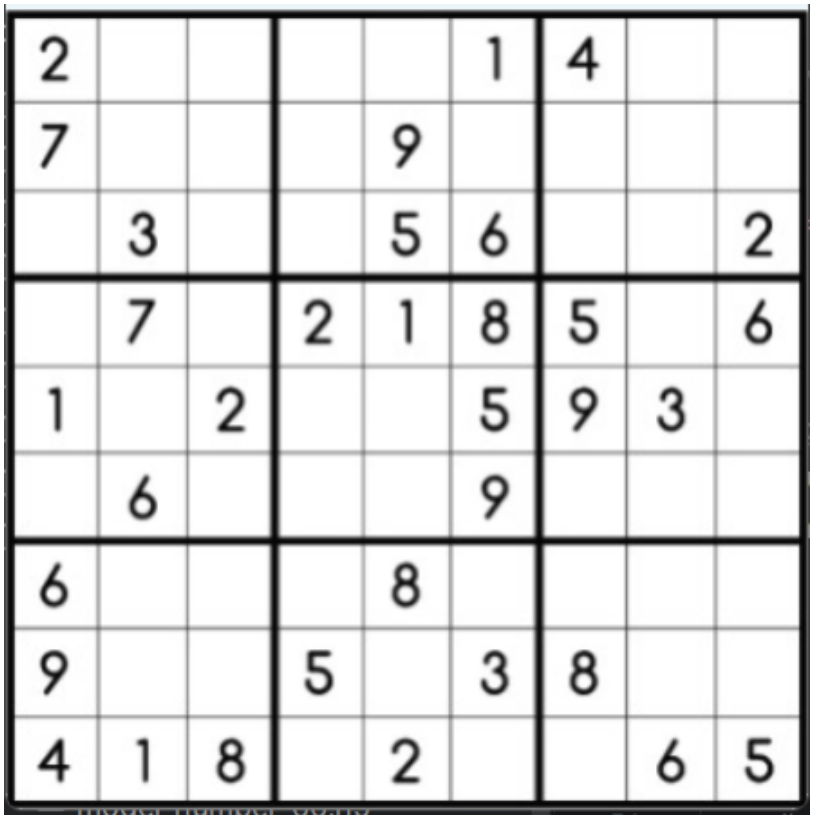


Hình 2.9 Ảnh sau khi tìm ra viền lớn nhất.

- Chuyển về góc nhìn từ trên xuống và lấy ô vuông lớn nhất của bảng Sudoku:

+Sử dụng:

* warped = four_point_transform(gray, puzzleCnt.reshape(4, 2)) sẽ thực hiện biến đổi affine hoặc perspective transform để cắt và xoay ảnh sao cho vùng được bao bọc bởi đa giác (puzzleCnt) sẽ được chuyển đổi thành một hình vuông. Kết quả này thường được gọi là "ảnh đã được biến đổi" (warped image).



Hình 2.10 Ảnh sau khi cắt và chuyển về góc nhìn từ trên xuống.

- Tách các ô nhỏ trong bảng Sudoku: Sau khi tìm được viền và cắt được ô vuông lớn nhất trong hình thì tiến hành đi tách các ô nhỏ trong bảng để lấy ra các số trong ô đó. Đầu tiên sẽ đi tính chiều dài và chiều rộng của các ô nhỏ, sử dụng:

```
stepX = warped.shape[1] // 9
stepY = warped.shape[0] // 9
```

+ Sau đó thực hiện tách các ô, sử dụng:

```
startX = x * stepX
startY = y * stepY
endX = (x + 1) * stepX
endY = (y + 1) * stepY
cell = warped[startY:endY, startX:endX]
```

+ x, y ở đây sẽ bắt đầu từ 0 tới 8, tương ứng với 9 ô / 1 hàng, 9 ô / 1 cột, sau đó sẽ nhận được 1 ô cell để đi xử lý.

- Áp dụng Threshold và xóa đường viền của ô:

+ Sử dụng:

```
thresh = cv2.adaptiveThreshold(cell, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 15, 3)
thresh = clear_border(thresh)
```

* 255: Đây là giá trị ngưỡng tối đa, được sử dụng để đặt giá trị của các pixel trong ảnh nhị phân. Trong trường hợp này, pixel có giá trị lớn hơn hoặc bằng 255 sẽ được đặt thành 255 (trắng), và pixel có giá trị nhỏ hơn 255 sẽ được đặt thành 0 (đen).

* cv2.ADAPTIVE_THRESH_MEAN_C: Đây là phương pháp ngưỡng hóa thích ứng sử dụng trung bình của các vùng lân cận. Trong trường hợp này, được sử dụng trung bình.

* cv2.THRESH_BINARY_INV: Đây là loại ngưỡng hóa, trong trường hợp này, là ngưỡng nhị phân nghịch đảo. Nghĩa là, giá trị pixel lớn hơn ngưỡng sẽ được đặt thành giá trị tối thiểu (0), và giá trị pixel nhỏ hơn ngưỡng sẽ được đặt thành giá trị tối đa (255).

* 15: Đây là kích thước của khu vực lân cận được sử dụng để tính toán ngưỡng. Kích thước này đặc định kích thước của vùng lân cận xung quanh mỗi pixel.

* 3: Đây là hằng số trừ đi từ trung bình tính toán. Nó giúp kiểm soát độ chệch giữa ngưỡng tính toán và giá trị thực tế của pixel.

* clear_border để loại bỏ các đối tượng ở mép của ảnh nhị phân. Chức năng của hàm này có thể bao gồm việc loại bỏ các vùng nằm ở biên của ảnh nhị phân, giúp làm sạch và xử lý dữ liệu hình ảnh.

Hình 2.11 Ảnh các ô sau khi cắt

Hình 2.12 Ảnh các ô sau khi áp dụng Threshold và xóa viền

- Áp dụng mặt nạ cho ô đã xóa viền: Có tác dụng xác định và trích xuất một đối tượng từ ảnh nhị phân, có nghĩa là xác định xem ô đó có chữ số hay không.

```
#Tim duong vien cua cell
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

if len(cnts) == 0:
    return None
c = max(cnts, key=cv2.contourArea)
mask = np.zeros(thresh.shape, dtype="uint8")
cv2.drawContours(mask, [c], -1, 255, -1)

(h, w) = thresh.shape
percentFilled = cv2.countNonZero(mask) / float(h * w)

#neu percentFilled < 0.03
if percentFilled < 0.03:
    return None

digit = cv2.bitwise_and(thresh, thresh, mask=mask)
```

+ cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE): Sử dụng hàm cv2.findContours để tìm các đường viền trong ảnh nhị phân thresh. Tham số cv2.RETR_EXTERNAL chỉ tìm các đường viền ngoại vi của các đối tượng và cv2.CHAIN_APPROX_SIMPLE chỉ giữ lại các điểm đầu cuối của đường viền thay vì giữ toàn bộ đường viền.

+ cnts = imutils.grab_contours(cnts): Sử dụng hàm imutils.grab_contours giúp trích xuất đường viền từ đối tượng trả về bởi cv2.findContours.

+ if len(cnts) == 0: return None: Kiểm tra xem có đường viền nào được tìm thấy hay không. Nếu không, trả về None vì không có đối tượng nào được xác định.

+ c = max(cnts, key=cv2.contourArea): Chọn đường viền lớn nhất dựa trên diện tích bằng cách sử dụng hàm max và cv2.contourArea. Điều này giả sử rằng đối tượng quan tâm là đối tượng lớn nhất trong ảnh.

+ mask = np.zeros(thresh.shape, dtype="uint8"): Tạo một mảng zeros (mask) có kích thước giống với ảnh nhị phân thresh.

+ cv2.drawContours(mask, [c], -1, 255, -1): Vẽ đường viền được chọn (c) lên mask. Các pixel bên trong đường viền sẽ được đặt thành giá trị 255 (trắng).

+ (h, w) = thresh.shape: Lấy chiều cao (h) và chiều rộng (w) của ảnh nhị phân.

+ percentFilled = cv2.countNonZero(mask) / float(h * w): Tính tỷ lệ diện tích được lấp đầy bởi đối tượng so với diện tích toàn bộ ảnh.

+ if percentFilled < 0.03: return None: Kiểm tra xem tỷ lệ diện tích được lấp đầy có nhỏ hơn 3% hay không. Nếu nhỏ hơn, trả về None vì đối tượng quá nhỏ để quan tâm.

+ digit = cv2.bitwise_and(thresh, thresh, mask=mask): Sử dụng phép toán bitwise AND giữa ảnh nhị phân thresh và mask để chỉ giữ lại phần của ảnh chứa đối tượng quan tâm. Kết quả sẽ là digit, là ảnh chỉ chứa đối tượng được xác định.

Hình 2.13 Ảnh các ô sau khi cắt.

Hình 2.14 Ảnh sau khi xử lý.

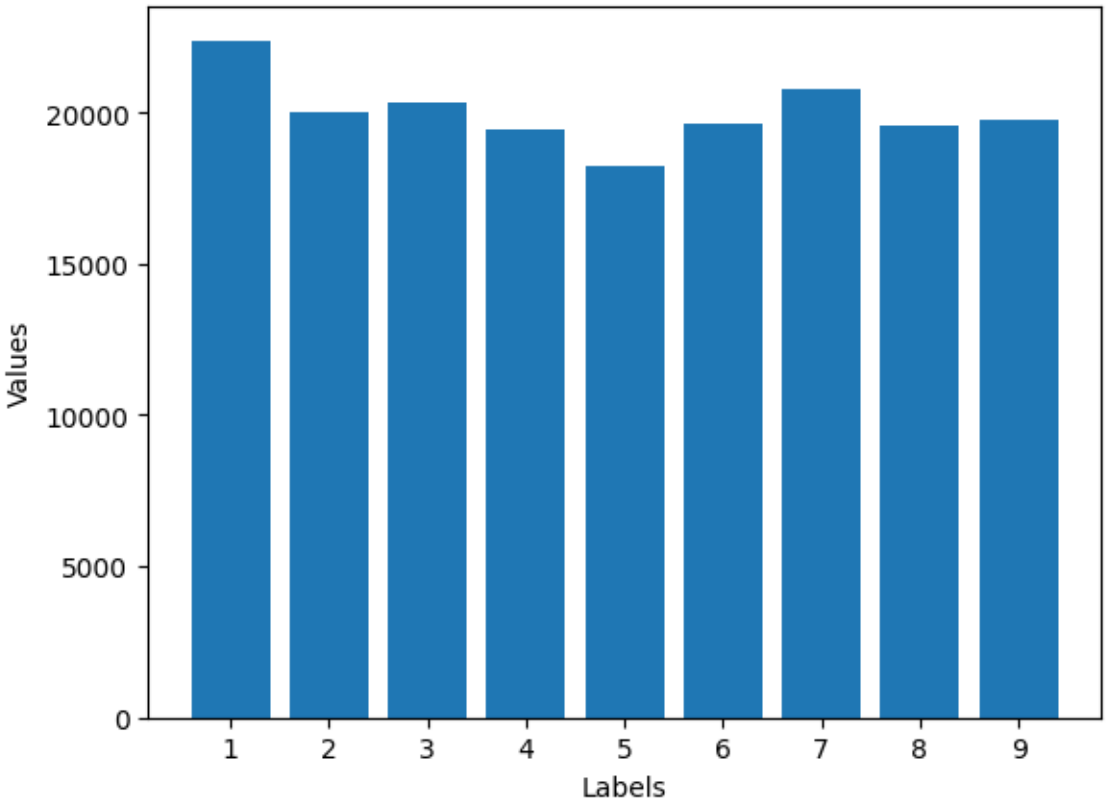
+ Chúng ta có thể thấy ô nào có chữ số thì mới thực hiện việc áp dụng mask và đảo bit, những ô nào trống thì không thực hiện việc này.

- Vậy khi qua các bước xử lý ảnh này, đầu ra sẽ là các ô có chứa số ở bên trong, với nền đen và số là màu trắng.

3.2 Sử dụng CNN để nhận biết các số trong bảng.

a. Dữ liệu huấn luyện.

- Tập dữ liệu sử dụng trong đề tài được tổng hợp từ ba tập dữ liệu:
  + MNIST Dataset: là một trong những bộ dữ liệu phổ biến nhất trong lĩnh vực học máy và thị giác máy tính. Bộ dữ liệu này chứa các hình ảnh kích thước 28x28 của các chữ số viết tay từ 0 đến 9.
  + QMNIST Dataset: là một phiên bản mở rộng của MNIST, bao gồm cả dữ liệu viết tay, dữ liệu in, và một số yếu tố đa dạng khác.
  + Tập dữ liệu gồm chữ viết đánh máy từ một số tài liệu khác.
- Trò chơi Sudoku gồm các ô được đánh số từ 1 đến 9 nên các hình ảnh có nhãn 0 được loại bỏ để mô hình có khả năng dự đoán chính xác hơn.
- Sau khi tổng hợp các tập dữ liệu và loại bỏ các hình ảnh có nhãn 0 thì số lượng hình ảnh sử dụng để huấn luyện là 180109.



Hình 2.15 Biểu đồ số lượng hình ảnh theo các nhãn.

b. Xây dựng mô hình.

- Tiền xử lí dữ liệu.

```
# Chuyển đổi giá trị pixel từ [0..255] về [0..1]
x_train = X_train1.astype("float32") / 255
x_test = X_test1.astype("float32") / 255

# Đảm bảo hình ảnh đầu vào có dạng (28, 28, 1)
x_test = np.expand_dims(x_test, -1)
x_train = np.expand_dims(x_train, -1)

# Chuyển đổi các nhãn thành vector ma trận nhị phân
Y_train1 = Y_train1 - 1
y_train = to_categorical(Y_train1, num_classes=9)
Y_test1 = Y_test1 - 1
y_test = to_categorical(Y_test1, num_classes=9)
```

- Xây dựng mô hình.

```
input_shape = (28, 28, 1)
batch_size = 32
epochs = 50
model = Sequential()
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(input_shape)))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(16, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(9, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(
    loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"]
)
checkPoint = ModelCheckpoint('model_number_{epoch}.h5', save_freq = 'epoch')

H = model.fit(
    x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.15, callbacks=[checkPoint],verbose=1
)
```

Trong đó:

+ Conv2D: là lớp tích chập đùng để trích xuất các đặc trưng của hình ảnh, bao gồm các tham số như số lượng bộ lọc (filters), kích thước cửa sổ trượt (kernel_size), hàm kích hoạt (activation), phương thức khởi tạo trọng số bộ lọc (kernel_initializer), phương thức đệm(padding).

+ Dropout : Lớp dropout được sử dụng để ngẫu nhiên "tắt" một số đầu ra của lớp trước nó trong quá trình huấn luyện. Điều này giúp mô hình tránh việc quá tập trung vào một số đặc trưng cụ thể và cũng giúp ngăn ngừa hiện tượng quá khớp (overfitting).

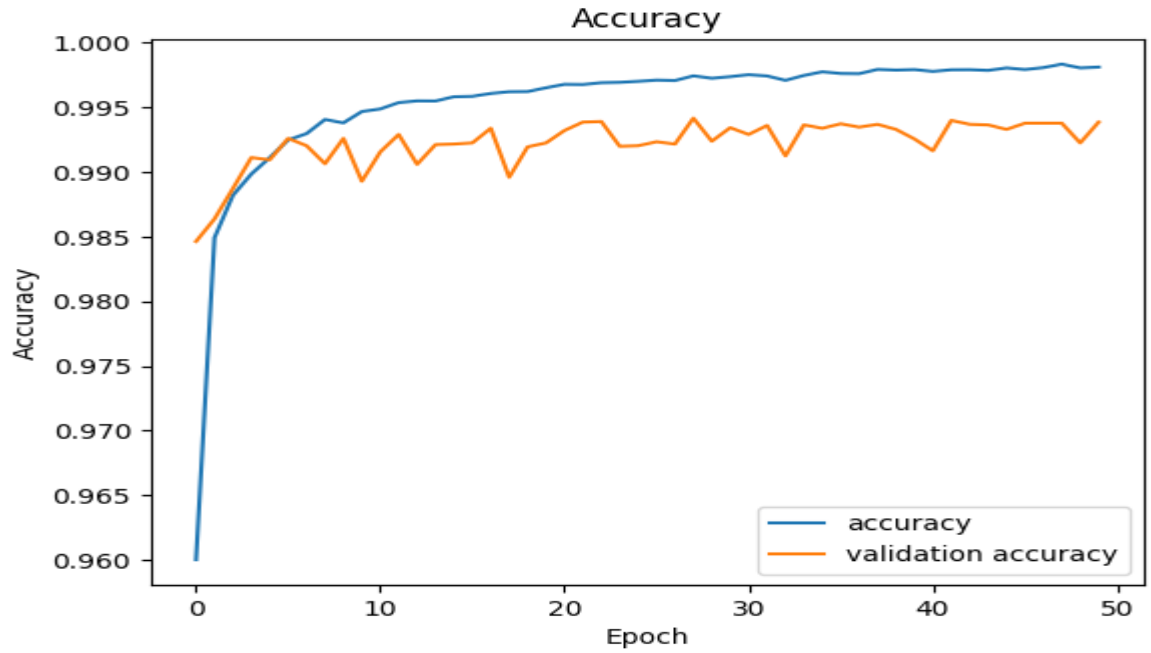
+ BatchNormalization: chuẩn hoá dữ liệu đầu ra của lớp trước đó, giúp đảm bảo các đặc trưng đầu ra có trung bình gần bằng 0 và độ lệch chuẩn gần bằng 1.

+ MaxPooling2D: thường được dùng giữa các convolutional layer để giảm kích thước dữ liệu những vẫn giữ được các thuộc tính quan trọng, kích thước dữ liệu giảm giúp giảm việc tính toán trong mô hình.

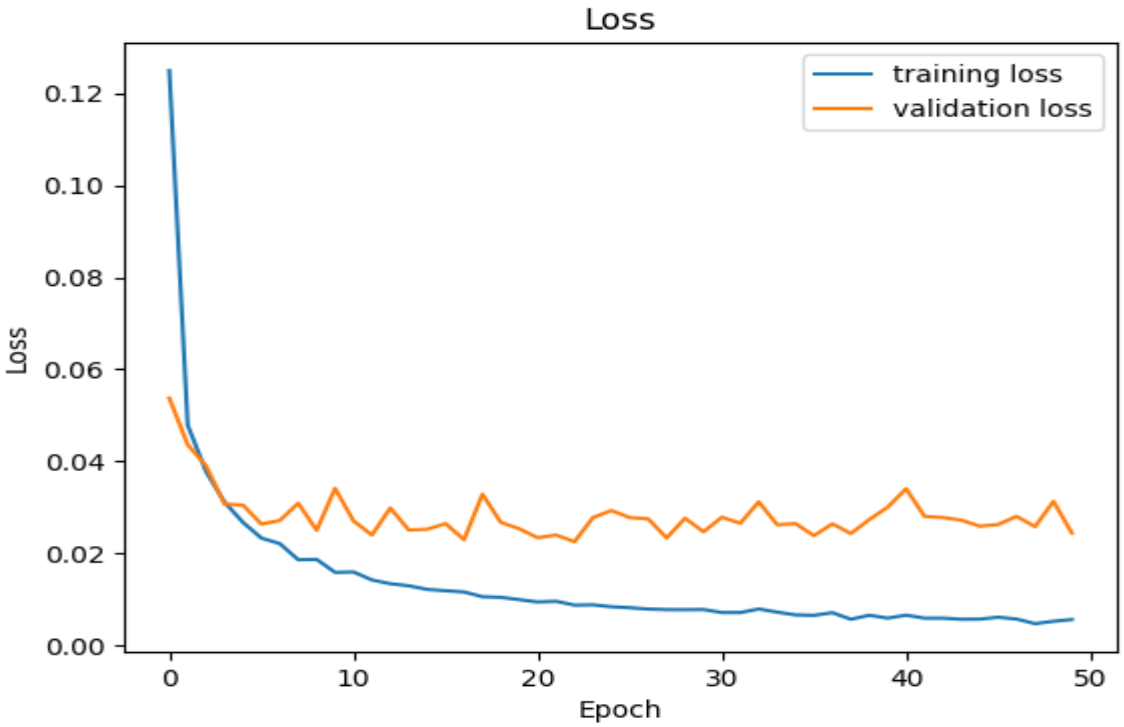
+ Flatten: thực hiện phép biến đổi từ một feature map 2D thành một vector 1D bằng cách "duỗi" (flatten) các giá trị từ các ô trong feature map thành một dãy liên tục.

+ Dense: là lớp fully connected dùng để kết hợp các đặc điểm của ảnh để ra được output của mô hình.

- Đánh giá.



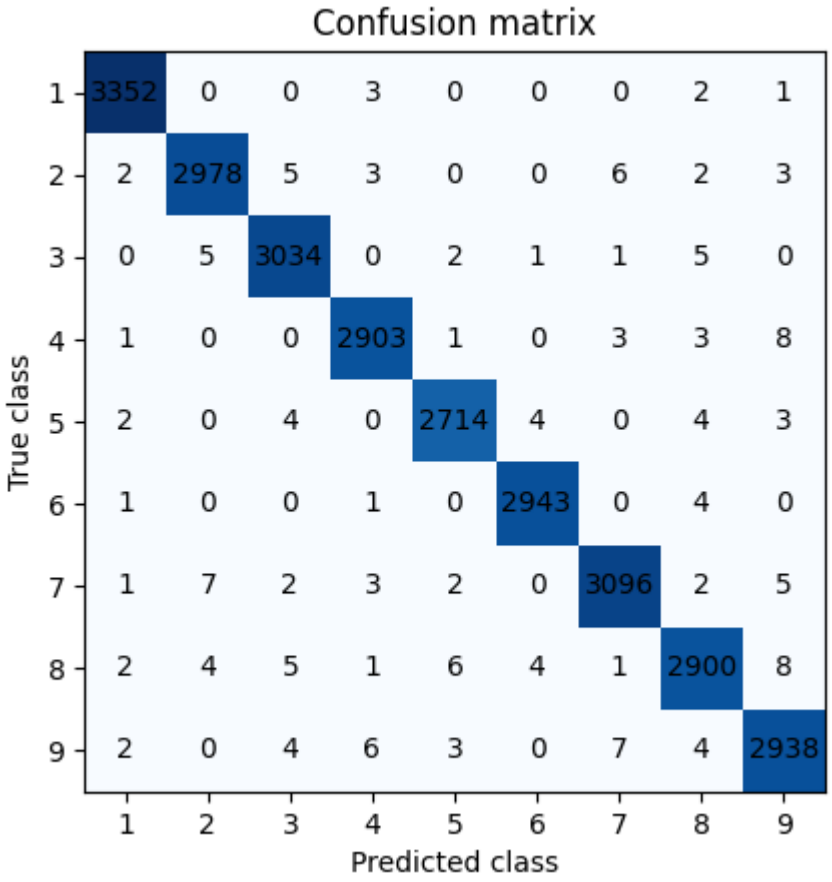
Hình 2.16 Biểu đồ thể hiện độ chính xác(accuracy).



Hình 2.17 Biểu đồ thể hiện độ lệch(loss).

+ Kết quả dự đoán trên tập dữ liệu test.

```
Test loss: 0.026069635525345802
Test accuracy: 0.9938927292823792
```



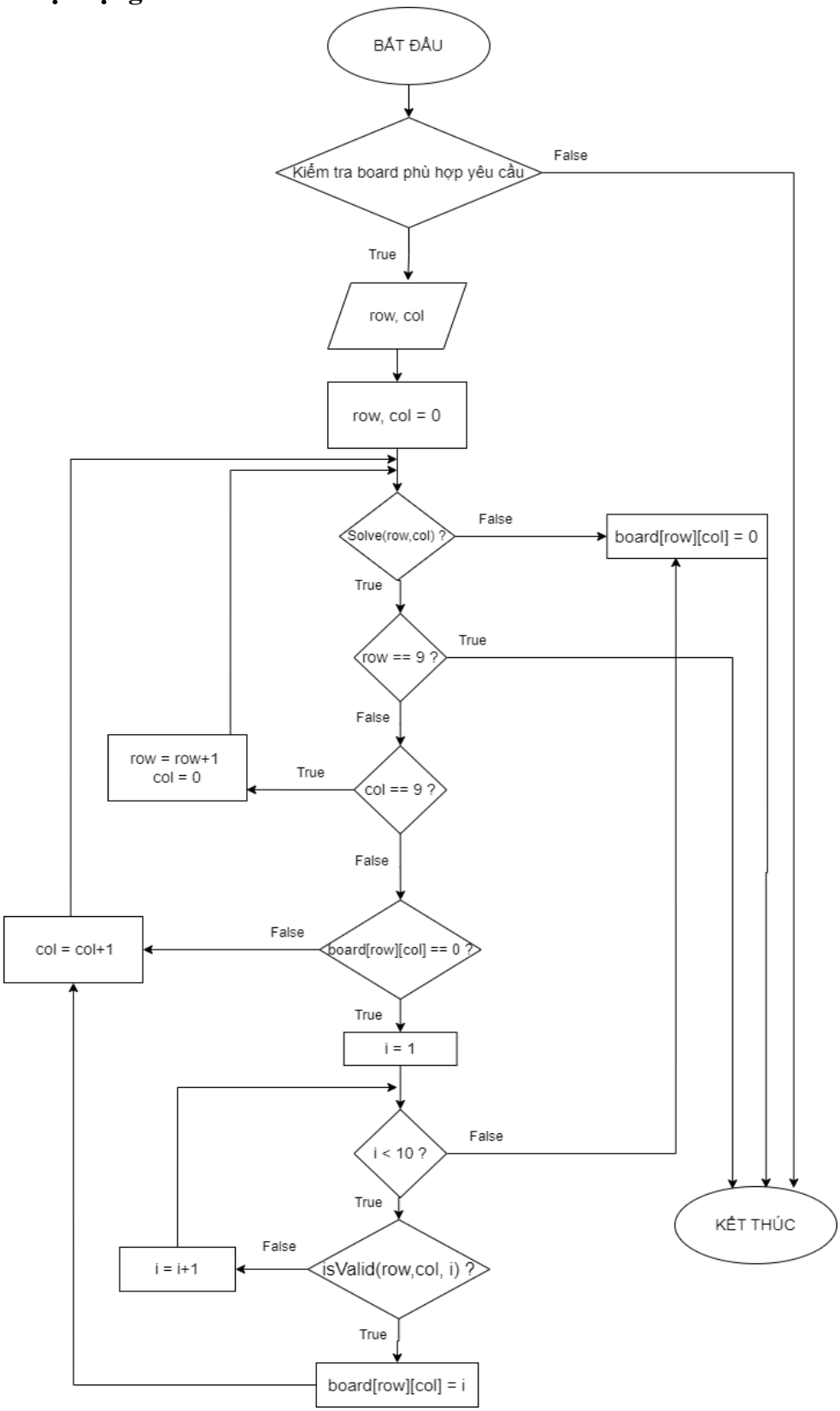
Hình 2.18 Ma trận thể hiện lỗi của mô hình.

3.3. Thuật toán giải Sudoku.

a. Tổng quan.

- Lớp Sudoku được thiết kế để giải quyết và giải đố trò chơi Sudoku thông qua một mô hình hướng đối tượng trong ngôn ngữ lập trình Python. Mô hình này sử dụng thuật toán quay lui để thử các giá trị và tìm ra giải pháp cho bảng Sudoku.

b. Mô hình hoạt động.



c. Chi tiết thuật toán.

- Phương thức “solveSudoku”: Đây là phương thức chính để giải Sudoku. Phương thức này sử dụng giải thuật đệ quy để điền các ô trống trong bảng Sudoku.

+ Hàm “isValid” :

```
def isValid(row, col, ch):
    row, col = int(row), int(col)

    for i in range(9):

        if self.board[i][col] == ch:
            return False
        if self.board[row][i] == ch:
            return False

        if self.board[3 * (row // 3) + i // 3][3 * (col // 3) + i % 3] == ch:
            return False

    return True
```

* Kiểm tra xem giá trị ‘ch’ có hợp lệ cho ô tại hàng ‘row’ và cột ‘col’ không.
* Kiểm tra hàng và cột xem đã tồn tại giá trị ‘ch’ chưa.
* Kiểm tra hình chữ nhật 3x3 tương ứng xem đã tồn tại giá trị ‘ch’ chưa.

+ Hàm “solve” :

```
def solve(row, col):
    if row == n:
        return True
    if col == n:
        return solve(row + 1, col: 0)

    if self.board[row][col] == 0:
        for i in range(1, 10):
            if isValid(row, col, i):
                self.board[row][col] = i

                if solve(row, col + 1):
                    return True
                else:
                    self.board[row][col] = 0
        return False
    else:
        return solve(row, col + 1)
solve( row: 0, col: 0)
```

* Hàm đệ quy “solve” được gọi với tham số là hàng ‘row’ và cột ‘col’.
* Nếu ‘row’ đạt đến giá trị 9, nghĩa là đã điền hết tất cả các hàng, và bảng Sudoku đã được giải xong, hàm trả về True.
* Nếu ‘col’ đạt đến giá trị 9, nghĩa là đã điền hết tất cả các cột của hàng hiện tại, chuyển sang hàng tiếp theo bằng cách gọi đệ quy solve(row + 1, 0).
* Kiểm tra nếu ô hiện tại trống (self.board[row][col] == 0), thử các giá trị từ 1 đến 9.
    - Nếu giá trị đó hợp lệ (không trùng với giá trị trong hàng, cột, hoặc hình chữ nhật 3x3), đặt giá trị vào ô và tiếp tục đệ quy với ô tiếp theo (solve(row, col + 1)).
    - Nếu không có giá trị nào hợp lệ, đặt lại giá trị của ô và trả về False để thử các giá trị khác của ô trước đó.
* Nếu ô không trống, tiếp tục đệ quy với ô tiếp theo (solve(row, col + 1)).

- Sau khi gọi solveSudoku(), bảng Sudoku trong sudoku.board sẽ được cập nhật với giải pháp nếu có.

3.4. Xử lý đầu ra.

- Sau khi giải được đề bài với solveSudoku(), hiển thị đáp án lên hình ảnh với vị trí phù hợp thông qua hàm:

```
cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])
```

Trong đó:

+ image: là hình ảnh sẽ hiển thị văn bản trên đó.

. + text: văn bản cần hiển thị.

+ org: toạ độ của văn bản hiển thị.

+ font: phông chữ.

+ fontScale: Hệ số tỷ lệ phông chữ được nhân với kích thước cơ sở của phông chữ cụ thể.

+ color: màu chữ.

+ thickness: độ lớn của nét chữ.

+ lineType: kiểu đường vẽ chữ lên ảnh.

+ bottomLeftOrigin: hệ toạ độ góc khi vẽ ảnh, True là góc dưới bên trái, False là góc trên bên trái, mặc định là False.

```
def displayNumbers(img, numbers, solved_num, color=(0, 0, 255)):
    w = int(img.shape[1] / 9)
    h = int(img.shape[0] / 9)

    for i in range(9):
        for j in range(9):
            if numbers[j][i] == 0:
                cv2.putText(img, str(solved_num[j][i]),
                            (i * w + int(w / 2) - int((w / 4)), int((j + 0.7) * h)),
                            cv2.FONT_HERSHEY_COMPLEX, 1, color,
                            1, cv2.LINE_AA)
    return img
```

- Sử dụng cv2.getPerspectiveTransform(pts1, pts2) để tạo ma trận chuyển đổi phối cảnh (perspective transformation), trong đó pst1 là tập điểm toạ độ trên ảnh gốc, pts2 là tập điểm toạ độ đích.
- Và dùng cv2.warpPerspective(masked_num, matrix, (img.shape[1], img.shape[0])) để áp dụng chuyển đổi phối cảnh, trong đó masked_num là ảnh đầu vào muốn áp dụng chuyển đổi phối cảnh lên, matrix là ma trận chuyển đổi phối cảnh, (img.shape[1], img.shape[0])là kích thước của ảnh đầu ra sau chuyển đổi.

```
def get_inv_perspective(img, masked_num, location, height=450, width=450):

    pts1 = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
    pts2 = np.float32([location[0], location[1], location[2], location[3]])

    matrix = cv2.getPerspectiveTransform(pts1, pts2)
    result = cv2.warpPerspective(masked_num, matrix, (img.shape[1],
                                                      img.shape[0]))
    return result
```

- Cuối cùng, sử dụng cv2.addWeighted() để kết hợp hình ảnh kết quả với hình ảnh gốc ban đầu.

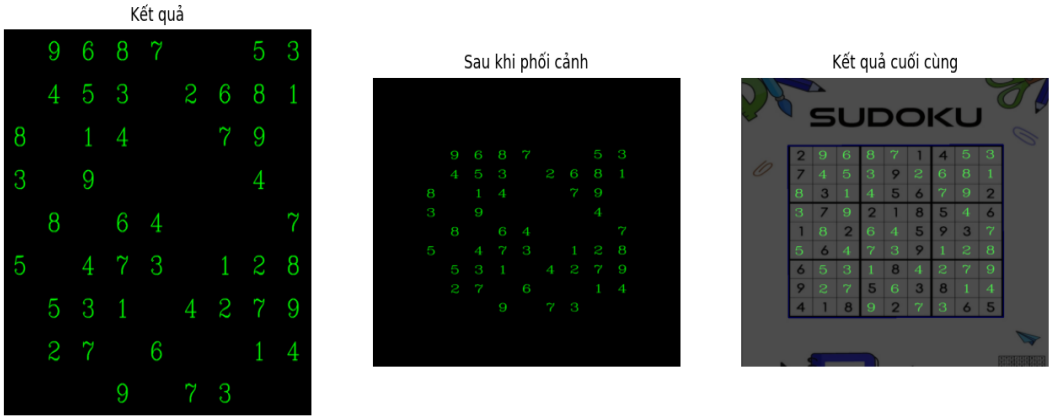
```
def inv_transformation(mask,img,predicted_matrix,solved_matrix,corners):
    img_solved = displayNumbers(mask, predicted_matrix, solved_matrix)
    inv = get_inv_perspective(img, img_solved, corners)
    img = cv2.addWeighted(img,0.5, inv,0.5, 0)

    return img,img_solved
```

- Hàm cv2.addWeighted() thực hiện phép cộng có trọng số giữa hai hình ảnh theo công thức:

$$dst = src1 \times alpha + src2 \times beta + gamma$$

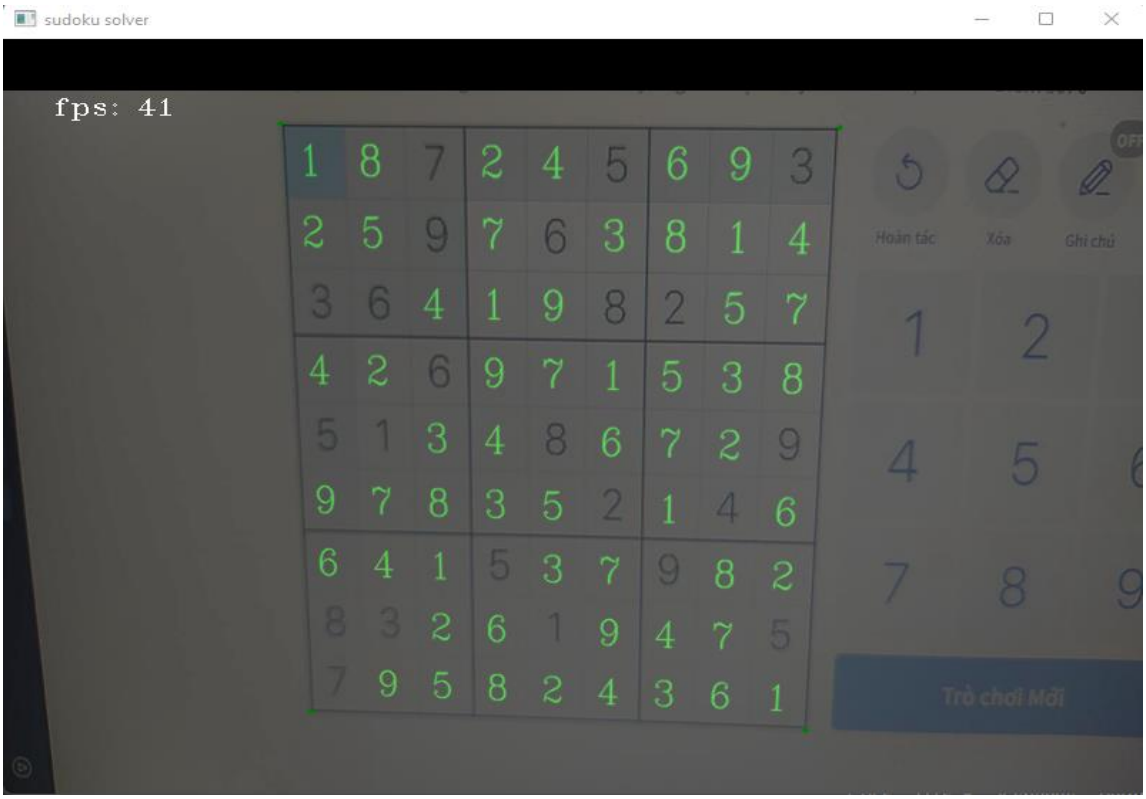
- Trong đó:
  + src1: Hình ảnh đầu tiên (img).
  + alpha: Hệ số trọng số cho hình ảnh img.
  + src2: Hình ảnh thứ hai (inv).
  + beta: Hệ số trọng số cho hình ảnh inv.
  + gamma: Hệ số độ tự tin.

- Kết quả :



Chương III. Demo.

Hình 3.1 Hình ảnh chương trình tìm kiếm ô Sudoku.



Hình 3.2 Hình ảnh chương trình hiển thị kết quả Sudoku.

Hình 3.3 Hình ảnh kết quả so khớp với kết quả giải được.

Chương IV. Kết luận.

- Trong quá trình thực hiện đồ án, nhóm đã thực hiện được :
  + Ứng dụng các kỹ thuật xử lí ảnh để trích xuất ô Sudoku, nhận diện các ô nhỏ và tải đề bài Sudoku vào trong chương trình.
  + Xây dựng được mô hình nhận diện chữ viết tay với độ chính xác cao.
  + Xây dựng thuật toán giải quyết các đề bài Sudoku.
  + Sử dụng các kỹ thuật xử lí ảnh để hiển thị kết quả cho người dùng.
  + Xây dựng được chương trình xử lí hình ảnh trực tiếp.

- Bên cạnh đó nhóm vẫn còn hạn chế chưa thực hiện được:
  + Chương trình xử lí hình ảnh trực tiếp chưa được tối ưu, mang đến cảm giác gián đoạn cho người dùng.
  + Mô hình dự đoán vẫn còn sai số dẫn đến chương trình hoạt động không chính xác.

- Qua đó, trong tương lai nhóm sẽ cố gắng cải thiện chương trình để ứng dụng giải quyết các đề bài Sudoku tốt hơn.