Sudoku OCR Journey Documentation

Introduction

The Sudoku OCR (Optical Character Recognition) journey involves extracting a Sudoku puzzle grid from an image and solving it using computer vision techniques. The goal is to develop an automated system that can read and solve Sudoku puzzles captured in images.

Software Used

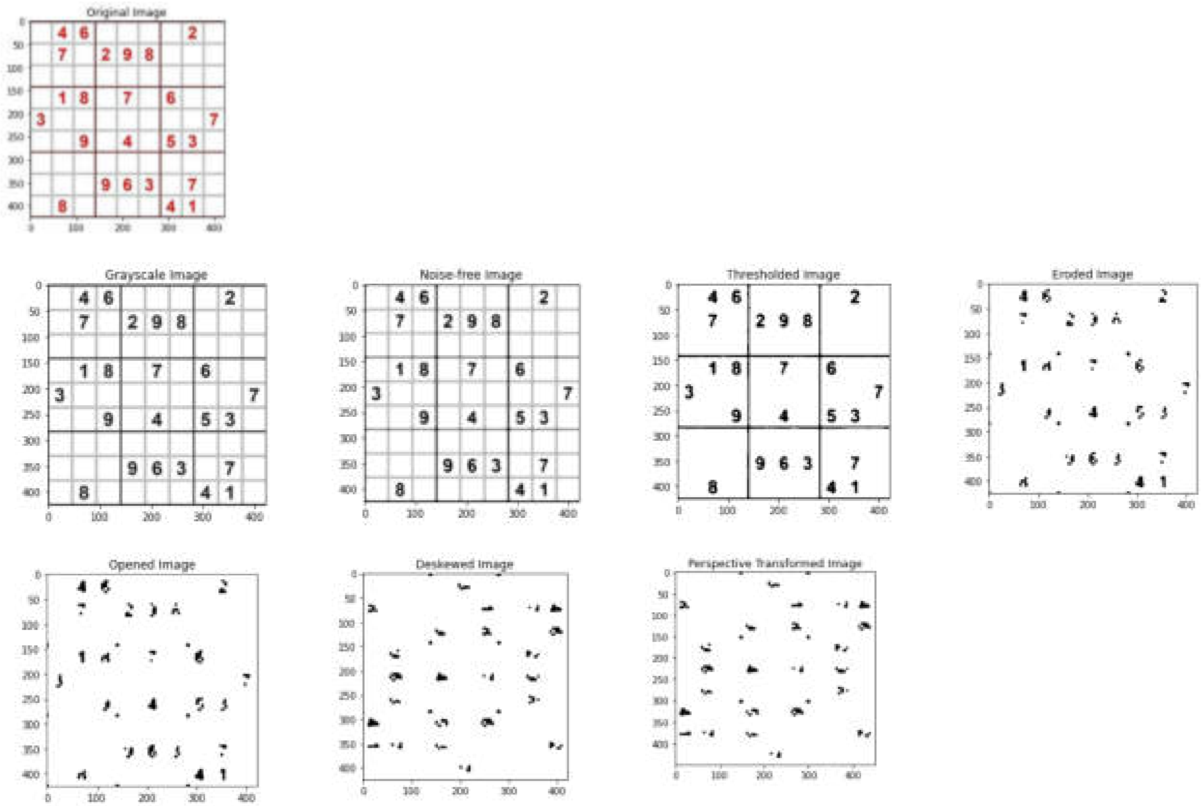
- OpenCV: A computer vision library used for image preprocessing, line extraction, noise removal, and other image manipulation tasks.
- Tesseract OCR: An open-source OCR engine used for character recognition.
- NumPy: A numerical computing library used for handling grid data structures and array operations.
- Matplotlib: A plotting library used for visualizing intermediate and final images.
- PIL (Python Imaging Library): A library used for opening, manipulating, and saving images.

Objectives

1. Preprocess the image to isolate the Sudoku puzzle grid and enhance digit readability.
2. Extract the individual cells of the Sudoku grid.
3. Perform OCR on each cell to recognize the digits present.
4. Solve the Sudoku puzzle using a backtracking algorithm.
5. Allow user interaction to edit the Sudoku grid before solving.

Image Preprocessing

General functions and their effects:



Preprocessing steps used:

1. **Grayscale Conversion**
   The image is converted from RGB to grayscale using the **cv2.cvtColor()** function from OpenCV. Grayscale conversion simplifies further processing by reducing the image to a single channel representing intensity values.

2. **Binarization**
   The grayscale image is binarized using adaptive thresholding with the **cv2.adaptiveThreshold()** function. This step separates the foreground (digits) from the background, resulting in a binary image.

3. **Line Extraction**
   Hough Transform, implemented through the **cv2.HoughLinesP()** function, is utilized to detect lines present in the binary image. This step helps identify the grid lines of the Sudoku puzzle.

4. **Line Removal**
   Detected lines from the previous step are removed from the binary image using the **cv2.line()** function. This process eliminates the grid lines, isolating the Sudoku digits for further processing.

5. **Noise Removal**
   To reduce noise and smoothen the image, a denoising filter (median blur) is applied using the **cv2.medianBlur()** function. This step improves the accuracy of digit extraction and subsequent OCR.

6. **Erosion**
   Erosion is performed using the **cv2.erode()** function to enhance the structure and readability of the Sudoku digits. It helps improve the segmentation of individual digits, making them more distinguishable.

7. **Cell Extraction**
   The eroded image is split into individual cells representing each digit of the Sudoku puzzle. The **np.vsplit()** and **np.hsplit()** functions from NumPy are used to divide the image into a 9x9 grid, resulting in 81 individual cells.

OCR and Sudoku Solving

OCR (Optical Character Recognition)

For each cell, the OCR process is performed using the Tesseract OCR engine. The **pytesseract.image_to_string()** function is used to extract text from each cell image. A custom OCR configuration is set to recognize only digits and exclude other characters.

Sudoku Puzzle Solving

The Sudoku puzzle grid is represented as a 2D list of digits. Initially, the OCR results are used to populate the grid with the recognized digits. Then, a backtracking algorithm is applied to solve the Sudoku puzzle. The algorithm finds an empty cell, tries numbers from 1 to 9, and recursively solves the puzzle until a solution is found or all possibilities are exhausted.

Editing Sudoku

The system allows users to edit the Sudoku grid before solving. Users can specify the row number and column numbers to modify, with '0' representing a blank cell. This feature enables interactive interaction and customization of the Sudoku puzzle.

Challenges Faced

1. Divisibility of Cells: One challenge encountered was ensuring that the Sudoku grid cells were correctly extracted. Resizing the image and adjusting the parameters for splitting the cells helped overcome this challenge.

2. Misreading of Digits: Another challenge was the misreading of certain digits, especially the digit '1' being mistaken for '4'. Training the OCR model specifically for Sudoku digits could help improve accuracy, but for simplicity, a basic approach was adopted.

3. Distinct Boundaries: Maintaining distinct boundaries between the cells while removing grid lines was crucial. However, this led to some double-digit numbers being misread. Striking a balance

between distinct boundaries and accurate OCR results required experimentation and fine-tuning.

Conclusion

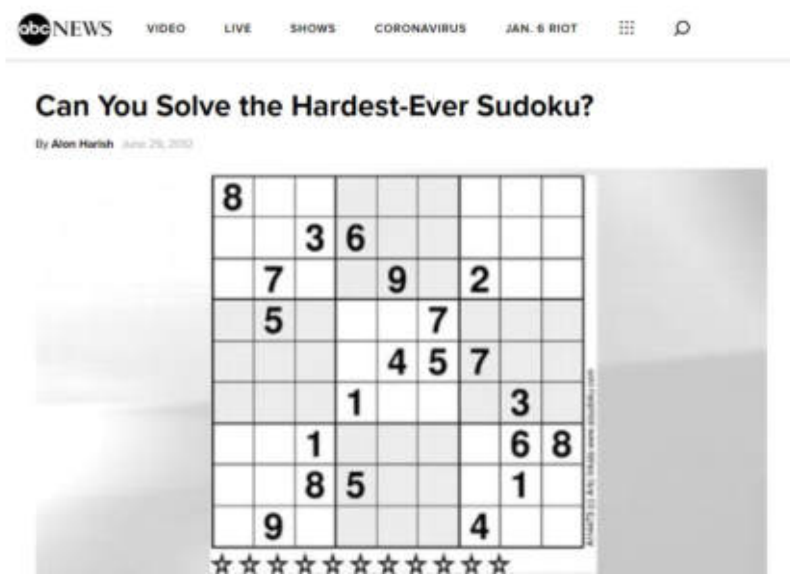
The Sudoku OCR journey encompasses various image preprocessing techniques, OCR using Tesseract, Sudoku solving algorithms, and user interaction. By combining computer vision and machine learning, the system provides an automated way to extract and solve Sudoku puzzles captured in images. The integration of OpenCV, Tesseract OCR, and other libraries offers a powerful framework for building Sudoku OCR applications and exploring other image-based puzzle-solving tasks.

Also, after a LONG time of trial and error yes, the order of the preprocessing steps can affect the final result. The order of the preprocessing steps for success are:

1. **Grayscale conversion**: Convert the image to grayscale.
2. **Binarization**: Apply adaptive thresholding to convert the grayscale image to a binary image.
3. **Line removal**: Use Hough Transform to detect and remove the lines from the binary image.
4. **Denoising**: Apply a denoising filter (e.g., median blur) to reduce noise in the image.
5. **Erosion**: Perform erosion to further enhance the structure and readability of the digit

   There is a factor of the extent of each that need to be tested too to get optimal results for all photos.

Robustness Test:



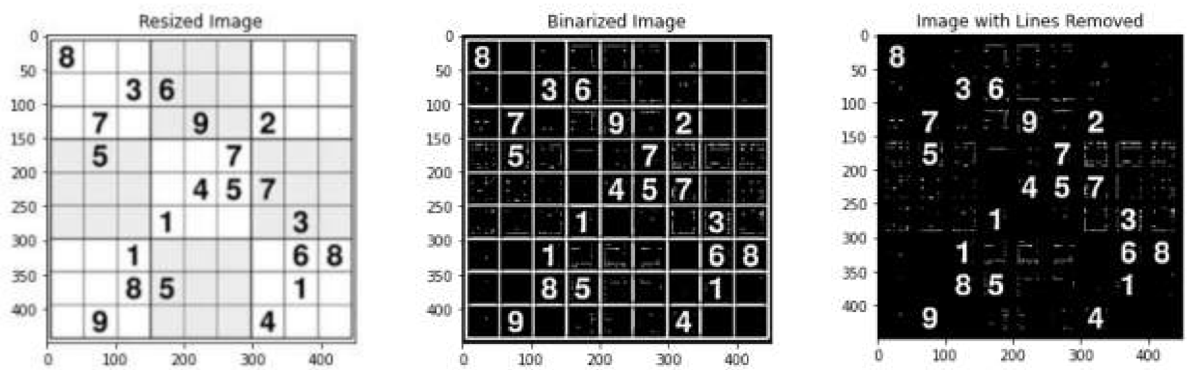
Current Sudoku Grid:
OCR Results:
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 0 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
Enter the row number to edit (1-9), or enter '0' to solve the puzzle: 6

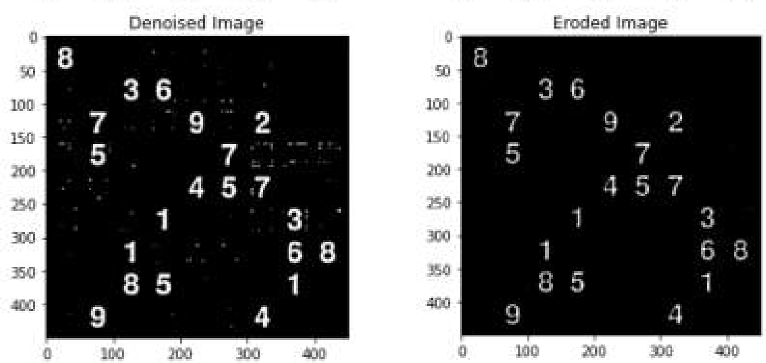
Enter the column numbers to edit (1-9), separated by spaces, with zero to represent gaps: 0 0 0 1 0 0 0 3 0

Current Sudoku Grid:
Current Sudoku Grid:
OCR Results:
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0

Enter the row number to edit (1-9), or enter '0' to solve the puzzle: 0

Answer:
Current Sudoku Grid:
OCR Results:
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2





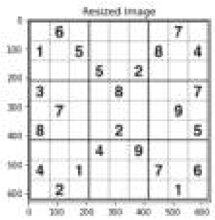
Answer:
Current Sudoku Grid:
OCR Results:
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2

8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

actual answer.

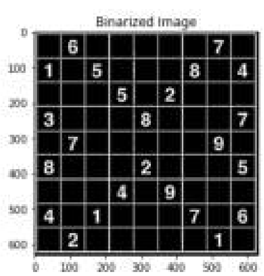
https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7 **preprocessing tips**

Notes and Chicken scratch of journey

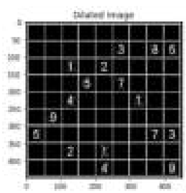


Issue was the cutting up of the square was not divisible. So resize.

Other issue now is that lines are being read as 1, and the recurved 1 is looking like a 4. It could do with training but for this model I want it to be simple.

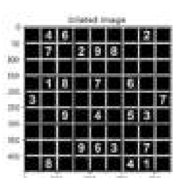


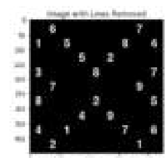
To help with distinct boundaries, but now there are more readings of double digit numbers.



0 4 6 0 0 0 0 0 2. 0
0 0 0 2 9 0 0 0 0
0 0 0 0 0 0 0 0 0
0 4 8 0 0 0 0 0 0
3 0 0 0 0 0 0 0 7
0 0 0 0 4 0 5 0 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 0 0 7 0
0 8 0 0 0 0 4 0 0

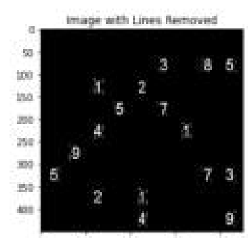
Almost, but not quite. Machine learning looks like it will have to be incorporated as 1s are often misread.





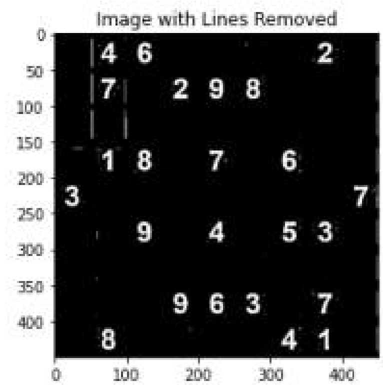
almost there!

0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 0 0 2 0 0 0
0 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0



really good except 1 is read as 4:

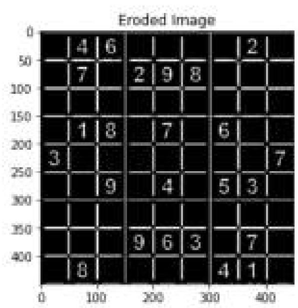
0 0 0 0 0 0 0 0 0
0 0 0 0 0 3 0 8 5
0 0 4 0 2 0 0 0 0
0 0 0 5 0 7 0 0 0
0 0 4 0 0 0 4 0 0
0 9 0 0 0 0 0 0 0
5 0 0 0 0 0 0 7 3
0 0 2 0 0 0 0 0 0
0 0 0 0 4 0 0 0 9



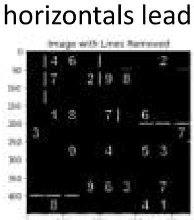
0 4 6 0 0 0 0 2 0
0 7 0 2 9 8 0 0 0
0 0 0 0 0 0 0 0 0
0 1 8 0 7 0 6 0 0
0 0 0 0 0 0 0 0 7
0 0 9 0 4 0 5 3 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 3 0 7 0
0 0 0 0 0 0 4 0 0
Missed a few on this one

Yes, the order of the preprocessing steps can affect the final result. In your desired sequence, the order of the preprocessing steps would be:

6. Grayscale conversion: Convert the image to grayscale.
7. Binarization: Apply adaptive thresholding to convert the grayscale image to a binary image.
8. Line removal: Use Hough Transform to detect and remove the lines from the binary image.
9. Denoising: Apply a denoising filter (e.g., median blur) to reduce noise in the image.
10. Erosion: Perform erosion to further enhance the structure and readability of the digit

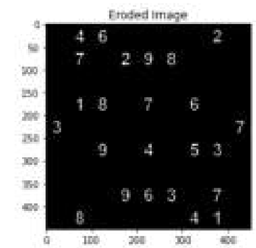


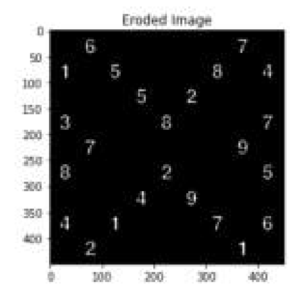
eroding lines doesn’t make much sense here as we are left with ill defined horizontals leading to this:



SUCCESS

0 4 6 0 0 0 0 2 0
0 7 0 2 9 8 0 0 0
0 0 0 0 0 0 0 0 0
0 1 8 0 7 0 6 0 0
3 0 0 0 0 0 0 0 7
0 0 9 0 4 0 5 3 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 3 0 7 0
0 8 0 0 0 0 4 1 0





erosion factor can be relaxed but it Is doing really well!

0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 5 0 2 0 0 0
3 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0

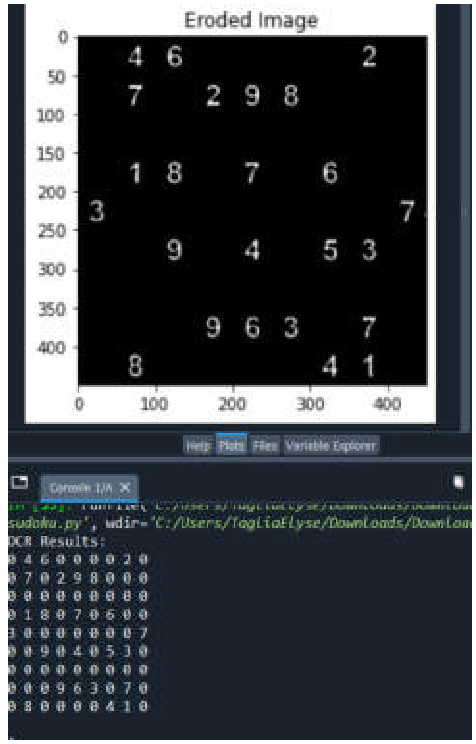
Backtracking:

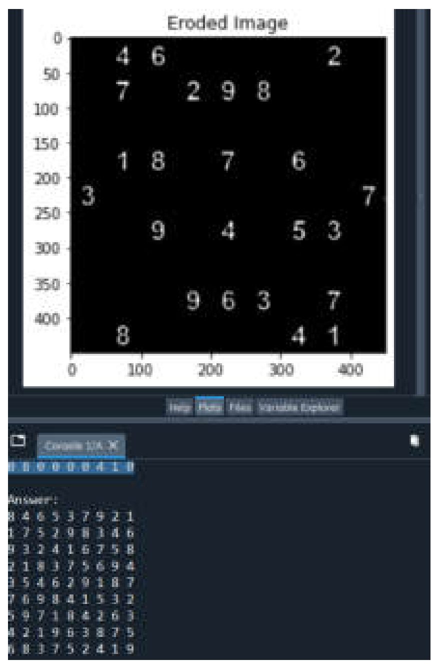
OCR Results:

0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 5 0 2 0 0 0
3 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0

Answer:

9 6 2 1 4 8 5 7 3
1 3 5 6 9 7 8 2 4
7 4 8 5 3 2 9 6 1
3 5 6 9 8 1 2 4 7
2 7 4 3 6 5 1 9 8
8 1 9 7 2 4 6 3 5
6 8 7 4 1 9 3 5 2
4 9 1 2 5 3 7 8 6
5 2 3 8 7 6 4 1 9





Complete Code:

```
import cv2
import numpy as np
import pytesseract
from PIL import Image
import matplotlib.pyplot as plt
# Set the path to the image file
image_path = r'C:\Users\TagliaElyse\Downloads\sudoku5.jpg'
# Set the tesseract path to the location of the tesseract executable
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
def binarize_image(image):
    return cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 11, 2)
def extract_lines(image):
    lines = cv2.HoughLinesP(image, 1, np.pi / 180, 100, minLineLength=100,
maxLineGap=10)
    return lines

def remove_lines(image, lines):
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 0, 0), 3)
    return image

def remove_noise(image):
    return cv2.medianBlur(image, 3)

def erode_image(image):
    kernel = np.ones((3, 3), np.uint8)
    return cv2.erode(image, kernel, iterations=1)
def split_cells(image):
    rows = np.vsplit(image, 9)
    cells = []
    for row in rows:
        cols = np.hsplit(row, 9)
        for cell in cols:
            cells.append(cell)
    return cells
def solve_sudoku(grid):
    # Find empty cell
    empty_cell = find_empty_cell(grid)
    if not empty_cell:
        return True  # Puzzle solved

    row, col = empty_cell
    # Try numbers from 1 to 9
    for num in range(1, 10):
        if is_valid_move(grid, row, col, num):
            grid[row][col] = str(num)

            # Recursively solve the puzzle
            if solve_sudoku(grid):
                return True

            # If the current number doesn't lead to a solution, backtrack
            grid[row][col] = "0"

    return False
def find_empty_cell(grid):
    for row in range(9):
        for col in range(9):
            if grid[row][col] == "0":
                return row, col
    return None

def is_valid_move(grid, row, col, num):
    # Check row
    if str(num) in grid[row]:
        return False

    # Check column
    for i in range(9):
        if grid[i][col] == str(num):
            return False

    # Check 3x3 box
    box_row = row - row % 3
    box_col = col - col % 3
    for i in range(3):
        for j in range(3):
            if grid[box_row + i][box_col + j] == str(num):
                return False

    return True
def main():
    # Load the image
    img = cv2.imread(image_path)
    img_resized = cv2.resize(img, (450, 450))
    plt.imshow(img_resized)
    plt.title("Resized Image")
    plt.show()

    # Convert to grayscale
    gray = get_grayscale(img_resized)
    plt.imshow(gray, cmap='gray')
    plt.title("Grayscale Image")
    plt.show()

    # Binarize the image
    binarized = binarize_image(gray)
    plt.imshow(binarized, cmap='gray')
    plt.title("Binarized Image")
    plt.show()
    # Extract lines
    lines = extract_lines(binarized)
    # Remove lines from the image
    lines_removed = remove_lines(binarized.copy(), lines)
    plt.imshow(lines_removed, cmap='gray')
    plt.title("Image with Lines Removed")
    plt.show()

    # Remove noise from the image
    denoised = remove_noise(lines_removed)
    plt.imshow(denoised, cmap='gray')
    plt.title("Denoised Image")
    plt.show()

    # Erode the image
    eroded = erode_image(denoised)
    plt.imshow(eroded, cmap='gray')
    plt.title("Eroded Image")
    plt.show()

    # Split the image into cells
    cells = split_cells(eroded)

    # Set the custom config for OCR
    custom_config = r'--oem 3 --psm 6 -c tessedit_char_whitelist=123456789
outputbase digits'

    # Format output as a grid
    grid = []
    for i in range(0, len(cells), 9):
        row = []
        for j in range(i, i + 9):
            cell_image = Image.fromarray(cells[j])
            ocr_text = pytesseract.image_to_string(cell_image,
config=custom_config).strip()
            row.append(ocr_text if ocr_text != "" else "0")
        grid.append(row)

    # Print the original grid
    print("OCR Results:")
    for row in grid:
        print(" ".join(row))

    # Solve the Sudoku puzzle
    solve_sudoku(grid)

    # Print the solved grid
    print("\nAnswer:")
    for row in grid:
        print(" ".join(row))

# Call the main function
main()
```

ABilitiyy to edit by specifying row before solving

```
import cv2
import numpy as np
import pytesseract
from PIL import Image
import matplotlib.pyplot as plt

# Set the path to the image file
image_path = r'C:\Users\TagliaElyse\Downloads\sudoku5.jpg'

# Set the tesseract path to the location of the tesseract executable
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'

def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

def binarize_image(image):
    return cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 11, 2)

def extract_lines(image):
    lines = cv2.HoughLinesP(image, 1, np.pi / 180, 100, minLineLength=100,
maxLineGap=10)
    return lines

def remove_lines(image, lines):
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 0, 0), 3)
    return image

def remove_noise(image):
    return cv2.medianBlur(image, 3)
def erode_image(image):
    kernel = np.ones((3, 3), np.uint8)
    return cv2.erode(image, kernel, iterations=1)
def split_cells(image):
    rows = np.vsplit(image, 9)
    cells = []
    for row in rows:
        cols = np.hsplit(row, 9)
        for cell in cols:
            cells.append(cell)
    return cells
def solve_sudoku(grid):
    # Find empty cell
    empty_cell = find_empty_cell(grid)
    if not empty_cell:
        return True  # Puzzle solved
    row, col = empty_cell
    # Try numbers from 1 to 9
    for num in range(1, 10):
        if is_valid_move(grid, row, col, num):
            grid[row][col] = str(num)

            # Recursively solve the puzzle
            if solve_sudoku(grid):
                return True

            # If the current number doesn't lead to a solution, backtrack
            grid[row][col] = "0"

    return False

def find_empty_cell(grid):
    for row in range(9):
        for col in range(9):
            if grid[row][col] == "0":
                return row, col
    return None

def is_valid_move(grid, row, col, num):
    # Check row
    if str(num) in grid[row]:
        return False

    # Check column
    for i in range(9):
        if grid[i][col] == str(num):
            return False

    # Check 3x3 box
    box_row = row - row % 3
    box_col = col - col % 3
    for i in range(3):
        for j in range(3):
            if grid[box_row + i][box_col + j] == str(num):
                return False

    return True

def edit_sudoku(grid):
    while True:
        row_num = input("Enter the row number to edit (1-9), or enter '0' to solve
the puzzle: ")
        if row_num == '0':
            break

        col_nums = input("Enter the column numbers to edit (1-9), separated by
spaces, with zero to represent gaps: ").split()

        if not all(num.isdigit() and 0 <= int(num) <= 9 for num in col_nums):
            print("Invalid column numbers. Please try again.")
            continue

        row_index = int(row_num) - 1
        for col_num, value in enumerate(col_nums):
            col_index = col_num
            grid[row_index][col_index] = str(value)

        print("\nCurrent Sudoku Grid:")
        print_sudoku_grid(grid)

def print_sudoku_grid(grid):
    print("Current Sudoku Grid:")
    print("OCR Results:")
    for row in grid:
        print(" ".join(row))

def main():
    # Load the image
    img = cv2.imread(image_path)
    img_resized = cv2.resize(img, (450, 450))
    plt.imshow(img_resized)
    plt.title("Resized Image")
    plt.show()

    # Convert to grayscale
    gray = get_grayscale(img_resized)
    plt.imshow(gray, cmap='gray')
    plt.title("Grayscale Image")
    plt.show()

    # Binarize the image
    binarized = binarize_image(gray)
    plt.imshow(binarized, cmap='gray')
    plt.title("Binarized Image")
    plt.show()
    # Extract lines
    lines = extract_lines(binarized)

    # Remove lines from the image
    lines_removed = remove_lines(binarized.copy(), lines)
    plt.imshow(lines_removed, cmap='gray')
    plt.title("Image with Lines Removed")
    plt.show()

    # Remove noise from the image
    denoised = remove_noise(lines_removed)
    plt.imshow(denoised, cmap='gray')
    plt.title("Denoised Image")
    plt.show()

    # Erode the image
    eroded = erode_image(denoised)
    plt.imshow(eroded, cmap='gray')
    plt.title("Eroded Image")
    plt.show()

    # Split the image into cells
    cells = split_cells(eroded)

    # Set the custom config for OCR
    custom_config = r'--oem 3 --psm 6 -c tessedit_char_whitelist=123456789
outputbase digits'

    # Format output as a grid
    grid = []
    for i in range(0, len(cells), 9):
        row = []
        for j in range(i, i + 9):
            cell_image = Image.fromarray(cells[j])
            ocr_text = pytesseract.image_to_string(cell_image,
config=custom_config).strip()
            row.append(ocr_text if ocr_text != "" else "0")
        grid.append(row)
    # Print the original grid
    print_sudoku_grid(grid)
    # Prompt user for edits
    edit_sudoku(grid)
    # Solve the Sudoku puzzle
    solve_sudoku(grid)

    # Print the solved grid
    print("\nAnswer:")
    print_sudoku_grid(grid)

# Call the main function
main()
```