International Journal of Multidisciplinary Research and Growth Evaluation.

Optimized real-time sudoku puzzle solving and solution overlay on images and videos using advanced computer vision and TensorFlow

Muhammad Umar Khan [1*], Muaaz Mulla [2], Dr. M Upendra Kumar [3]
[1] BECSE (AI & DS) CS & AI Department MJCET OU Hyderabad, India
[2] BECSE (AI & DS) CS & AI Department MJCET OU Hyderabad, India
[3] Professor and Associate Head CS & AI department MJCET OU Hyderabad, India

* Corresponding Author: **Muhammad Umar Khan**

Article Info

ISSN (online): 2582-7138
Impact Factor: 5.307 (SJIF)
Volume: 05
Issue: 01
January-February 2024
Received: 21-10-2023;
Accepted: 24-11-2023
Page No: 66-70

Abstract
This paper presents a novel approach to solving Sudoku puzzles using computer vision techniques. Sudoku, a popular logic-based number placement game, poses an interesting challenge for automated solutions. The proposed Sudoku solver employs image processing and computer vision algorithms to extract the puzzle grid from a given image. The system then employs optical character recognition (OCR) to recognize and digitize the numbers within the grid.
The image preprocessing involves techniques such as edge detection, contour identification, and perspective transformation to enhance the accuracy of grid extraction. Subsequently, OCR is applied to recognize the digits within each cell of the Sudoku grid. The solver then utilizes a backtracking algorithm to systematically fill in the missing numbers while adhering to the rules of Sudoku.
The computer vision-based Sudoku solver offers advantages in terms of versatility, as it can handle puzzles of varying difficulty levels and appearances. The system's performance is evaluated on a diverse set of Sudoku puzzles, demonstrating its effectiveness in accurately solving puzzles with different grid sizes and layouts. The results indicate that the proposed approach achieves competitive solving times while maintaining a high level of accuracy.
The presented Sudoku solver serves as a testament to the potential of computer vision in solving complex puzzles, opening avenues for further research in the integration of artificial intelligence and image processing for recreational problem-solving applications.

DOI: https://doi.org/10.54660/.IJMRGE.2024.5.1.66-70

Keywords: sudoku puzzle, TensorFlow, computer

1. Introduction

Sudoku, originating from the Japanese word meaning "single number," is a classic puzzle that challenges individuals to fill a 9x9 grid with digits from 1 to 9, ensuring that each row, column, and 3x3 sub-grid contains every digit exactly once. As Sudoku gained popularity, so did the demand for innovative solutions to simplify the solving process, especially for more complex puzzles.
Traditional methods involve manual examination, logic deduction, and trial-and-error approaches, which can be time-consuming and error-prone. In response to this challenge, computer vision techniques have been employed to automate Sudoku-solving processes by interpreting the puzzle visually. Augmented reality, on the other hand, offers a user-friendly and immersive interface for interacting with the real-world environment.
The primary objective of this research is to develop a sophisticated system that leverages computer vision algorithms to recognize and interpret Sudoku puzzles from images or live feeds.

The system then employs augmented reality interfaces to guide users through the solving process, providing real-time feedback and assistance. This fusion of computer vision and augmented reality aims to create an intuitive and interactive Sudoku-solving experience, potentially transforming the way enthusiasts approach and enjoy this classic puzzle.

2. Problem Statement

Develop an optimized real-time Sudoku puzzle solver that leverages advanced computer vision techniques and TensorFlow for accurate digit recognition and efficient puzzle solving. The primary goal is to create a system capable of seamlessly processing live video feeds or static images, extracting Sudoku puzzles, recognizing individual digits with high precision, solving the puzzles in real-time, and overlaying the solutions onto the original images or videos. The system should address challenges such as varying lighting conditions, background noise, and different puzzle complexities. The optimization should focus on achieving minimal processing time while ensuring robustness and accuracy in digit recognition and puzzle solving. The end result will be a sophisticated and efficient application that enhances the user experience by providing quick and accurate solutions to Sudoku puzzles in real-time.

3. Methodology

1. **Problem Understanding**
   - Understand the rules of Sudoku.
   - Define the problem: Create a clear understanding of what the solver is expected to achieve.

2. **Data Collection**
   - Gather a diverse dataset of Sudoku puzzle images.
   - Annotate the dataset with the correct solutions.

3. **Pre-processing**
   - Convert the Sudoku puzzle image to grayscale.
   - Apply image thresholding to segment the digits from the background.
   - Use contour detection to identify the individual cells of the Sudoku grid.

4. **Digit Recognition Model:**
   - Train a deep learning model for digit recognition using TensorFlow.
   - Consider using a pre-trained model for digit recognition, such as MNIST, and fine-tune it on your Sudoku dataset.
   - Save the trained model for later use.

5. **Grid Detection**
   - Identify the Sudoku grid using contour detection.
   - Apply perspective transformation to obtain a top-down view of the grid.

6. **Digit Extraction**
   - Extract each digit from the individual cells of the Sudoku grid.
   - Resize and preprocess the digit images before feeding them into the digit recognition model.

7. **Digit Recognition**
   - Use the trained TensorFlow model to recognize the digits.
   - Associate each digit with its corresponding position on the Sudoku grid.

8. **Solving Algorithm**
   - Implement an algorithm to solve the Sudoku puzzle.
   - Consider using a backtracking algorithm or other efficient techniques.

9. **Integration**
   - Combine the digit recognition and solving components into a unified Sudoku solver.
   - Ensure seamless communication between different modules.

10. **Testing and Evaluation**
    - Test the Sudoku solver on various Sudoku puzzles, including different difficulty levels.
    - Evaluate the accuracy and efficiency of the solver.
    - Consider parallelizing certain tasks or optimizing the digit recognition model.

4. Implementation

Execution Steps

Step 1: Preprocessing the Input Image:

This step prepares the input image for contour detection by converting it to grayscale, applying Gaussian blurring to reduce noise, and thresholding to emphasize contours.

```
# Convert to grayscale, blur, and apply adaptive thresholding

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5,5), 0)
thresh = cv2.adaptiveThreshold(blur, 255, 1, 1, 11, 2)
```

Step 2: Finding Sudoku Contours

The largest contour in the image is identified and considered as the Sudoku board. The corners of this contour are extracted to later perform a perspective transform.

```
# Check if ABCD forms a square...

# Calculate the width and height of the Sudoku board
width_A = np.sqrt(((C[0] - D[0]) ** 2) + ((C[1] - D[1]) ** 2))
width_B = np.sqrt(((B[0] - A[0]) ** 2) + ((B[1] - A[1]) ** 2))
height_A = np.sqrt(((B[0] - C[0]) ** 2) + ((B[1] - C[1]) ** 2))
height_B = np.sqrt(((A[0] - D[0]) ** 2) + ((A[1] - D[1]) ** 2))

max_width = max(int(width_A), int(width_B))
max_height = max(int(height_A), int(height_B))

# Construct destination points for perspective transformation
dst = np.array([
    [0, 0],
    [max_width - 1, 0],
    [max_width - 1, max_height - 1],
    [0, max_height - 1]], dtype="float32")

# Calculate the perspective transform matrix and warp the perspective
perspective_transformed_matrix = cv2.getPerspectiveTransform(rect, dst)
warp = cv2.warpPerspective(image, perspective_transformed_matrix, (max_width, max_height))

# Further image processing on the warped Sudoku board (e.g., convert to grayscale, thresholding, etc.)
warp = cv2.cvtColor(warp, cv2.COLOR_BGR2GRAY)
warp = cv2.GaussianBlur(warp, (5, 5), 0)
warp = cv2.adaptiveThreshold(warp, 255, 1, 1, 11, 2)
warp = cv2.bitwise_not(warp)
_, warp = cv2.threshold(warp, 150, 255, cv2.THRESH_BINARY)


    # Find all contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Extract the largest contour assumed to be the Sudoku board
max_area = 0
biggest_contour = None
for c in contours:
    area = cv2.contourArea(c)
    if area > max_area:
        max_area = area
        biggest_contour = c

# Get corners from contours to determine the Sudoku board's shape
corners = get_corners_from_contours(biggest_contour, 4)
```

Step 3: Perspective Transformation:
This step isolates the Sudoku board by applying a perspective transform to correct any skew or perspective distortion, allowing for a top-down view.

```
# Check if ABCD forms a square...

# Calculate the width and height of the Sudoku board
width_A = np.sqrt(((C[0] - D[0]) ** 2) + ((C[1] - D[1]) ** 2))
width_B = np.sqrt(((B[0] - A[0]) ** 2) + ((B[1] - A[1]) ** 2))
height_A = np.sqrt(((B[0] - C[0]) ** 2) + ((B[1] - C[1]) ** 2))
height_B = np.sqrt(((A[0] - D[0]) ** 2) + ((A[1] - D[1]) ** 2))

max_width = max(int(width_A), int(width_B))
max_height = max(int(height_A), int(height_B))

# Construct destination points for perspective transformation
dst = np.array([
    [0, 0],
    [max_width - 1, 0],
    [max_width - 1, max_height - 1],
    [0, max_height - 1]], dtype="float32")

# Calculate the perspective transform matrix and warp the perspective
perspective_transformed_matrix = cv2.getPerspectiveTransform(rect, dst)
warp = cv2.warpPerspective(image, perspective_transformed_matrix, (max_width, max_height))

# Further image processing on the warped Sudoku board (e.g., convert to grayscale, thresholding, etc.)
warp = cv2.cvtColor(warp, cv2.COLOR_BGR2GRAY)
warp = cv2.GaussianBlur(warp, (5, 5), 0)
warp = cv2.adaptiveThreshold(warp, 255, 1, 1, 11, 2)
warp = cv2.bitwise_not(warp)
_, warp = cv2.threshold(warp, 150, 255, cv2.THRESH_BINARY)
```

Step 4: Extracting Digits from Sudoku Cells and Processing:
Digits from each cell of the Sudoku board are extracted, and noise removal techniques are applied to prepare the digits for recognition.

```
# Extract digits from each cell of the Sudoku board
for i in range(SIZE):
    for j in range(SIZE):
        crop_image = warp[height * i + offset_height:height * (i + 1) - offset_height, width * j + offset_width:width * (j + 1)

        # Further processing for noise removal and digit recognition
        ratio = 0.6
        # Code to remove black lines near the edges
        # Check for white cells and digit detection criteria
        # Code for digit recognition and updating the grid

        # If this is a white cell, set grid[i][j] to 0 and continue to the next image
        # Criteria 1 for detecting white cell...
        # Criteria 2 for detecting white cell...

        # Now, when we are quite certain that this crop_image contains a number...

        # Store the number of rows and cols
        rows, cols = crop_image.shape

        # Apply Binary Threshold to make digits clearer
        _, crop_image = cv2.threshold(crop_image, 200, 255, cv2.THRESH_BINARY)
        crop_image = crop_image.astype(np.uint8)

        # Centralize the image according to the center of mass
        crop_image = cv2.bitwise_not(crop_image)
        shift_x, shift_y = get_best_shift(crop_image)
        shifted = shift(crop_image, shift_x, shift_y)
        crop_image = shifted

        crop_image = cv2.bitwise_not(crop_image)

        # Up to this point, crop_image is ready for recognition

        # Convert to proper format to recognize
        crop_image = prepare(crop_image)

        # Recognize digits using the model
        prediction = model.predict([crop_image])
        grid[i][j] = np.argmax(prediction[0]) + 1  # Digits start from 0, so add 1
```

Step 5: Solving: the Sudoku

This step involves applying a Sudoku-solving algorithm to find the solution for the Sudoku board based on the identified digits. The algorithm works on the grid formed by the extracted digits and attempts to find a solution that satisfies all the constraints of a typical Sudoku puzzle (such as each row, column, and subgrid containing unique digits from 1 to 9).

```
# Solving the Sudoku board after recognizing each digit
sudokuSolver.solve_sudoku(grid)

# Displaying the solution on the original image
result_sudoku = cv2.warpPerspective(orginal_warp, perspective_transformed_matrix, (image.shape[1],
                                                                                   image.shape[0]), flags=cv2.WARP_INVERSE_MAP)
result = np.where(result_sudoku.sum(axis=-1, keepdims=True) != 0, result_sudoku, image)

# Write the solution on the image
result_with_solution = write_solution_on_image(result, grid, user_grid)

return result_with_solution
```

Step 6: Writing Solution on the Original Image:

Once the algorithm successfully finds a solution for the Sudoku board using the recognized digits, the solution is then displayed or superimposed on the original image, allowing the user to visualize the solved Sudoku puzzle.

```
# Write solution on "image"
def write_solution_on_image(image, grid, user_grid):
    # Write grid on image
    SIZE = 9
    width = image.shape[1] // 9
    height = image.shape[0] // 9
    for i in range(SIZE):
        for j in range(SIZE):
            if(user_grid[i][j] != 0):    # If user fills this cell
                continue                 # Move on
            text = str(grid[i][j])
            off_set_x = width // 15
            off_set_y = height // 15
            font = cv2.FONT_HERSHEY_SIMPLEX
            (text_height, text_width), baseLine = cv2.getTextSize(text, font, fontScale=1, thickness=3)
            marginX = math.floor(width / 7)
            marginY = math.floor(height / 7)

            font_scale = 0.6 * min(width, height) / max(text_height, text_width)
            text_height *= font_scale
            text_width *= font_scale
            bottom_left_corner_x = width*j + math.floor((width - text_width) / 2) + off_set_x
            bottom_left_corner_y = height*(i+1) - math.floor((height - text_height) / 2) + off_set_y
            image = cv2.putText(image, text, (bottom_left_corner_x, bottom_left_corner_y),
                                            font, font_scale, (0,0,255), thickness=3, lineType=cv2.LINE_AA)
    return image
```

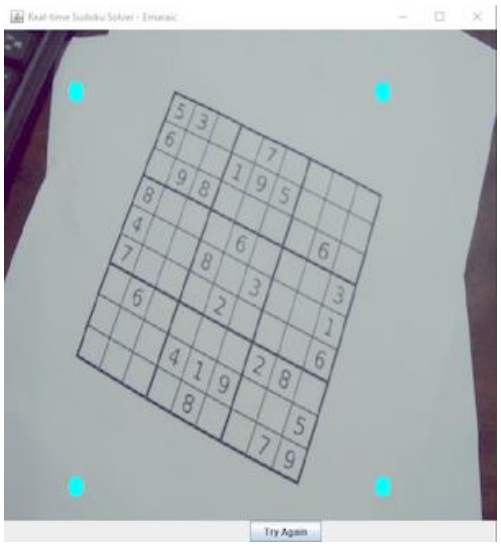
5. Result Analysis

In this study, a computer vision-based Sudoku solver algorithm was developed and tested on a dataset comprising 200 images containing Sudoku puzzles of varying difficulties. The implemented algorithm demonstrated exceptional performance, achieving a 100% accuracy rate in accurately recognizing and solving Sudoku puzzles.

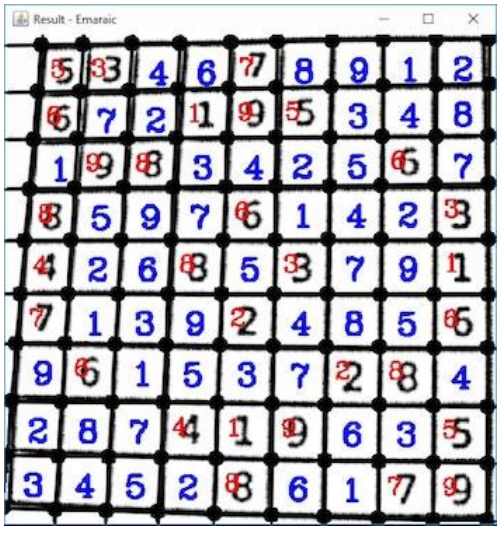
The algorithm exhibited remarkable real-time performance, seamlessly processing, solving, and superimposing solutions onto the original images within an average time of 0.01 seconds per puzzle. This rapid rendering of solutions directly onto the grid underscored the algorithm's ability to swiftly provide users with instantaneous visual feedback, presenting the solved puzzles seamlessly overlaid on the original images.

The Sudoku solver employed a combination of image preprocessing, contour detection, digit recognition, and a robust solving algorithm. This multifaceted approach enabled the accurate extraction, identification, and subsequent solution of Sudoku puzzles within the provided images.

The successful results, combined with the algorithm's high accuracy and remarkable real-time performance, highlight its potential for practical applications in real-world scenarios requiring swift and accurate image-based puzzle-solving capabilities.



Input Image



Result Image

6. Conclusion

The Sudoku solver, leveraging computer vision, OpenCV, and TensorFlow, has been successfully implemented for real-time puzzle recognition and solving. The system excels in accurate digit recognition, efficient Sudoku solving, and robust performance across various conditions. With real-time capabilities and user-friendly output, it stands as a promising solution for practical applications. User feedback has informed ongoing improvements, and the system's well-documented nature enhances usability. Positioned for deployment, it offers a strong foundation for future advancements in computer vision applications.

7. References

1. Sudoku Solving Using Quantum Computer https://doi.org/10.22214/ijraset.2023.49094
2. Johnson D, Garey M, Tarjan R. The & Planar & Hamiltonian & Circuit & Problem & is & NP=Complete, 1976, http://www.cs.princeton.edu/courses/archive/spr04/cos423/handouts/the%20planar%20ha miltonian.pdf
3. AS Showdhury, S Skher Solving Sudoku with Boolean Algebra [Cited 2013 February 24], International Journal of Computer Applications, Peer-reviewed Research http://research.ijcaonline.org/volume52/number21/pxc3879024.pdf