RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world®

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Project Report

On

OCR Sudoku Puzzle Solver

Submitted in partial fulfilment of the requirements for the V Semester

ARTIFICIAL NEURAL NETWORK AND DEEP LEARNING

AI253IA

By

1RV22AI058	**Srivanth Srinivasan**
1RV22AI041	**Rajyalakshmi Prasanna**

Department of Artificial Intelligence and Machine Learning
RV College of Engineering®
Bengaluru – 560059

Academic Year
2024-25

RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Bengaluru– 560059

CERTIFICATE

This is to certify that the project entitled "**OCR Sudoku Puzzle Solver"** submitted in partial fulfilment of Artificial Neural Networks and Deep Learning (AI253IA) of V Semester BE is a result of the bonafide work carried out by Srivanth Srinivasan (1RV22AI058) and Rajyalakshmi Prasanna (1RV22AI041) during the Academic year 2024-25.

Faculty In charge

Date:

Head of the Department

Date:

RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Bengaluru– 560059

DECLARATION

We, Srivanth Srinivasan (1RV22AI058) and Rajyalakshmi Prasanna (1RV22AI041), students of Fifth Semester BE, hereby declare that the Project titled **"OCR Sudoku Puzzle Solver**" has been carried out and completed successfully by us and is our original work.

Date of Submission: **Signature of the Student**

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our faculty mentor and course coordinator, **Professor Somesh Nandi**, for guiding us throughout our project. His meticulous attention to every step, from drafting the methodology to deployment, was instrumental in streamlining our efforts. Without his valuable direction, we would have spent significantly more time configuring the project's specifications. We also extend our special thanks to **Dr. Anupama Kumar** for her invaluable insights, support, and constructive feedback, which significantly contributed to the improvement of our work.

We are also grateful to our Head of Department **Dr. Satish Babu**, for curating AI-specific courses that are not only industry-relevant but also aligned with the latest technological trends. These courses provide us with an opportunity to deeply understand and explore subjects that are otherwise absent from the typical curriculum.

Lastly, we extend our sincere thanks to our Principal, **Dr. K. N. Subramanya**, Vice Principal, **Dr. K. S. Geetha**, and RV College of Engineering, Bangalore, for their unwavering support and encouragement. Their commitment to fostering an AI-focused curriculum and implementing innovative teaching, learning, and evaluation methods has been vital to our success.

ABSTRACT

Sudoku puzzles are popular brain-teasers that enhance logical thinking and problem-solving skills. It is a self-verifiable puzzle if solved correctly. However, if not solved correctly and there is a need to debug, or if the solved puzzle needs to be checked for correctness, manual effort increases exponentially with increase in complexity and quantity of Sudokus to be checked, hence relying on human checking for large scale applications is intensive and may be unreliable. This issue can be solved by automating the checking with the help of an OCR Sudoku Solver.

In simple terms, it uses optical character recognition (OCR) and deep learning (DL) to automate solving Sudoku puzzles from captured or uploaded images. It utilises concepts from Computer Vision for sudoku grid detection, AI features like OCR for digit recognition, and elements from Game theory for algorithmic puzzle solving. To increase the performance of our project, we use Convolutional Neural Networks (CNN) for more accurate digit recognition and employ backtracking algorithms to solve digital sudoku puzzles. Our project uses tools like OpenCV for image detection, Tesseract to perform OCR, Python to solve the puzzle and Matplotlib to display the output to user. In order to train our model to recognise handwritten digits, we train it on a popular dataset called MNIST, digital digits are easy to recognise.

The project we developed works like this: the input image is captured or uploaded from system storage, the image and digits are read and processed with OCR and CNN, the puzzle is solved with backtracking algorithms and the output solution is superimposed on the original image uploaded.

By employing a robust CNN architecture and advanced backtracking techniques, the system achieves high accuracy in digit recognition and puzzle-solving. This project is useful for everyone from individuals checking their personal puzzles to organizations and even websites to check multiple puzzles from multiple users all around the globe in a fast, accurate and automated manner. This project demonstrates the potential of combining computer vision and AI to solve real-world challenges, paving the way for future applications in similar domains.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Project description

The OCR Sudoku Puzzle Solver is designed to automate the process of solving Sudoku puzzles by leveraging advanced technologies such as image processing, deep learning, and algorithmic problem-solving. The project focuses on processing real-world images of Sudoku grids, recognizing the numbers within them, and solving the puzzle efficiently. By combining optical character recognition (OCR) with a convolutional neural network (CNN) for digit recognition and a backtracking algorithm for puzzle-solving, this system aims to provide accurate and swift solutions to Sudoku puzzles in an intuitive manner.

Sudoku is a popular logic-based number placement puzzle that requires players to fill a grid such that each row, column, and subgrid contain unique digits from 1 to 9. Traditionally, solving such puzzles is time-consuming and requires manual effort. The OCR Sudoku Puzzle Solver seeks to simplify this process by automating it. This system captures Sudoku grids from images, extracts and recognizes the digits using OCR and deep learning, and solves the puzzle using an efficient backtracking algorithm.

The primary objectives of this project are:

- To develop a system capable of accurately detecting and recognizing digits from Sudoku grids using OCR and CNNs.
- To solve Sudoku puzzles efficiently using backtracking while ensuring adherence to Sudoku rules.
- To provide users with a seamless and accurate experience for solving puzzles directly from image inputs, even in the presence of noisy or distorted grids.

The project relies on three foundational concepts:

- Image Processing: OpenCV is used to preprocess the input image, including operations like noise removal, grid detection, and cell segmentation.
- Deep Learning: A convolutional neural network (CNN) is employed for digit recognition, ensuring robust performance across varied handwriting and printing styles.
- Backtracking Algorithm: This recursive algorithm is used to solve the Sudoku puzzle, iteratively testing and placing numbers while adhering to Sudoku rules.

These combined techniques demonstrate the synergy between artificial intelligence, computer vision, and computational problem-solving, enabling the system to effectively automate the Sudoku-solving process.

1.2 Report Organization

Chapter 1: Introduction

This chapter introduces the project by describing its fundamental aspects, including the project description, objectives, and relevant theoretical concepts. It also outlines the organization of the report to provide a roadmap for the reader.

Chapter 2: Literature Review

This chapter surveys existing literature on Sudoku-solving techniques, focusing on OCR, deep learning models, and algorithmic methods. It evaluates various approaches, highlighting their strengths and limitations, and discusses unresolved challenges and emerging opportunities in this field.

Chapter 3: Software Requirements Specification

This chapter details the functional and non-functional requirements of the project. It includes an overview of the product, its features, constraints, user characteristics, and dependencies. It also specifies the hardware and software requirements and discusses the technical tools and platforms used to implement the system.

Chapter 4: System Design

This chapter focuses on the architectural and high-level design of the project. It presents the problem specification, block diagram, data flow diagrams (Level 0 and Level 1), and a description of the artificial neural network (ANN) or deep learning architecture and algorithms used in the project.

Chapter 5: Implementation

This chapter includes detailed code snippets and their explanations, illustrating the implementation of various project modules. Screenshots of outputs for each module are presented and discussed to demonstrate the functionality of the system

Chapter 6: Conclusion

This chapter summarizes the project development, including its objectives, methodology, and outcomes. It evaluates the success of the system in achieving its goals and discusses its practical significance

Chapter 7: Future Enhancements

The final chapter explores potential future improvements to the project, such as expanding the system to solve larger or more complex puzzles, improving the efficiency of the algorithms, or deploying the system on mobile platforms. It also includes a list of references in IEEE format, documenting all sources that were consulted during the development of the project.

Chapter 2: Literature Review

2.1 Literature Survey

The study by Wang, Sheng-Wei [1] introduces a dataset of Sudoku puzzles annotated with difficulty metrics experienced by human players. This dataset provides a robust foundation for training and evaluating OCR-based Sudoku solvers, enabling models to account for varying levels of puzzle complexity. The availability of such data supports the development of more adaptive and user-friendly solutions.

Dugar et al. [2] propose an augmented reality-based Sudoku solver that integrates a training module aimed at enhancing cognitive skills. This approach highlights the potential of combining OCR with interactive interfaces to engage users and improve problem-solving abilities. The system effectively bridges the gap between real-world puzzle recognition and user education.

Jain et al. [3] enhance the classification capabilities of OCR systems using a Tree Strength-Infused Enriched Random Forest model. While primarily focused on broader applications, this method offers insights into optimizing feature selection and classification in OCR tasks, which are critical for accurately identifying Sudoku grid elements.

Ahmareen et al. [4] focus on handwritten digit recognition using CNNs, achieving high accuracy on the MNIST dataset. This work underscores the effectiveness of deep learning in recognizing handwritten digits, a core component of OCR-based Sudoku solvers. The study's methodology serves as a benchmark for digit recognition tasks.

Dubey and Das [5] employ DCGANs combined with SIFT and ORB optical features for handwritten image detection. Their approach to leveraging generative models for augmenting training datasets can significantly improve the robustness of Sudoku solvers in handling diverse handwriting styles.

Anasune and Bhavsar [8] present an image-based Sudoku solver using applied recursive backtracking. This study demonstrates the integration of image preprocessing and logical solving techniques, highlighting the importance of combining OCR with efficient algorithmic strategies for real-time applications.

Li [7] explores the development of an English OCR system optimized for automatic text recognition. The techniques discussed, including preprocessing and recognition optimization, are directly applicable to Sudoku solvers, where accurate grid extraction and character recognition are paramount.

Notice et al. [9] tackle the algorithm selection problem for solving Sudoku puzzles using metaheuristics. This research introduces a framework for optimizing solver performance based on puzzle complexity, which can complement OCR systems by providing tailored solving strategies.

Ciantar and Casha [10] implement a Sudoku puzzle solver on an FPGA, showcasing the potential of hardware acceleration in enhancing solver efficiency. While the focus is on hardware implementation, the insights into system optimization can inform software-based OCR Sudoku solvers.

Jana et al. [11] design a modified 3D Sudoku solver, pushing the boundaries of traditional Sudoku-solving methodologies. Though 3D puzzles differ in format, the advanced techniques discussed can inspire innovations in grid analysis and solving mechanisms for 2D OCR-based Sudoku solvers.

Wadud and Wadud [12] propose an improved Sudoku solver that emphasizes accuracy and efficiency. Their approach to refining solving algorithms aligns with the goals of OCR-based systems, ensuring seamless integration of recognition and solving components.

Kaló and Sipos [19] utilize Tesseract OCR for key-value pair searching, demonstrating the effectiveness of OCR in structured data extraction. Their post-processing techniques can be adapted for Sudoku grid parsing, improving the accuracy of OCR outputs.

Lloyd and Amos [20] apply ant colony optimization to Sudoku solving, introducing bio-inspired algorithms for enhanced performance. This study provides a unique perspective on integrating heuristic methods with OCR systems to tackle complex puzzles.

These studies collectively highlight the advancements in OCR, machine learning, and algorithmic strategies relevant to Sudoku puzzle solving. The integration of these techniques can lead to the development of a robust, real-time OCR Sudoku Puzzle Solver capable of handling diverse puzzle formats and complexities.

2.2 Summary of Literature Survey

- Sudoku-solving methods include recursive backtracking, FPGA-based solvers, metaheuristics (e.g., ant colony optimization), and human-comprehensible AI.
- Advanced techniques like 3D Sudoku algorithms address more complex constraints.
- Handwritten recognition is improved using CNN, GAN, OCR (e.g., Tesseract), and hybrid CNN+LSTM on datasets like MNIST and Bengali scripts.
- AR-based Sudoku scanners and OCR systems integrate image preprocessing for real-world applications.
- Enriched Random Forests (ERF) and CNN-autoencoder hybrids enhance feature extraction and dimensionality reduction.
- Novel algorithms like “purge-and-merge” solve constraint satisfaction problems efficiently.
- Optical neural networks leverage light-based hardware for faster, energy-efficient computation.

- AI-driven platforms for student IDs and personalized training demonstrate practical applications.
- Hybrid AI approaches improve decision-making transparency and usability.
- Interdisciplinary solutions highlight advancements in AI, hardware integration, and real-world deployment

2.3 Existing and Proposed System

- **Existing system:**

  Sudoku solving has traditionally relied on manual methods, with limited automated solutions available. Existing automated systems often focus on standalone algorithms such as backtracking or metaheuristics, which are efficient but lack integration with advanced technologies like image processing or machine learning. While some solutions attempt to recognize Sudoku grids from images, they often struggle with low accuracy in digit recognition or grid detection.

- **Proposed system:**

  The proposed system integrates Deep Learning (CNN) for digit recognition, image processing techniques for Sudoku grid detection, and a recursive backtracking algorithm for solving puzzles. By combining these technologies, the system offers a robust solution for real-time Sudoku solving. It processes input images, identifies and recognizes digits, solves the puzzle, and overlays the solution onto the original image.

- **Problem statement:**

  Sudoku solving is a time-intensive task, especially for complex puzzles. The challenge lies in accurately detecting the grid from an image, recognizing digits, and solving the puzzle efficiently. The goal is to develop an intelligent system that automates these tasks while maintaining high accuracy and reliability.

  Scope of project:

  - *Image Processing:* To accurately detect Sudoku grids from input images, regardless of orientation or quality.
  - *Digit Recognition:* To recognize digits using Convolutional Neural Networks (CNNs) with high accuracy.
  - *Puzzle Solving:* To solve puzzles using recursive backtracking with a 100% success rate.
  - *User Experience:* To overlay solutions onto the original image, providing a seamless experience for users.
  - *Extensibility:* The system can be extended to include real-time AR-based solutions and additional puzzle types.

Methodology of proposed system:

- *Input Image Acquisition:* The user provides an image of the Sudoku puzzle, which is resized and preprocessed for analysis.
- *Grid Detection:* Using image processing techniques like Gaussian blur, adaptive thresholding, and contour detection, the Sudoku grid is extracted.
- *Digit Recognition:* Each cell in the grid is processed to recognize digits using a pre-trained CNN model.
- *Puzzle Solving:* A recursive backtracking algorithm is used to solve the puzzle based on recognized digits.
- *Solution Overlay:* The solved puzzle is overlaid onto the original image, enhancing user experience.
- *Output Display:* The final solved Sudoku is displayed to the user, with options to save the solution.

Technical features of proposed system:

- *Real-Time Image Processing*: High accuracy in detecting and extracting Sudoku grids from images using OpenCV.
- *Advanced Digit Recognition*: A CNN-based model trained on MNIST for reliable digit identification.
- *Optimized Solving Algorithm*: Recursive backtracking ensures rapid and accurate puzzle solving.
- *Scalable Architecture*: Modular design allows for future extensions like AR integration.
- *User-Friendly Interface*: Displays solved Sudoku with overlays for intuitive visualization.
- *Error Handling*: Robust mechanisms to handle poor image quality or incomplete grids

2.4 Tools and Technologies Used

- Programming language: Python
- Libraries & frameworks:
  - ➢ OpenCV (for image processing)
  - ➢ TensorFlow/Keras (for CNN implementation)
  - ➢ NumPy (for numerical computations)
  - ➢ Imutils (for perspective transformation)
- Algorithm: Recursive Backtracking

- Dataset: MNIST dataset for digit recognition model training

2.5 Hardware and Software Requirements

Hardware Requirements:

- Processor: Intel i5/i7 or equivalent with at least 4 cores
- RAM: 8 GB (minimum), 16 GB (recommended)
- Storage: At least 10 GB of free space
- Graphics: NVIDIA GPU (optional for faster model training)

Software Requirements:

- Operating System: Windows 10/Linux/MacOS
- Python 3.7 or higher
- Required Libraries: OpenCV, TensorFlow, Keras, NumPy, Matplotlib
- IDE: Visual Studio Code / Jupyter Notebook / PyCharm

Chapter 3: Software Requirements Specification

3.1 Introduction

The OCR Sudoku Puzzle Solver automates the process of solving Sudoku puzzles. The system captures Sudoku grids from images, processes them to recognize the digits, and solves the puzzle using a backtracking algorithm. It integrates OCR for digit recognition, deep learning models for accuracy, and image processing techniques for preprocessing.

Acronyms used in the document	
OCR	Optical Character Recognition
CNN	Convolutional Neural Network
DL	Deep Learning
ANN	Artificial Neural Network
DFD	Data Flow Diagram
IDE	Integrated Development Environment
MNIST	Modified National Institute of Standards and Technology: a database of handwritten digits that is commonly used for training various image processing systems.

3.2 General Description

- **Product Perspective:**

  The OCR Sudoku Puzzle Solver is an independent application that uses machine learning and computer vision techniques to process and solve Sudoku puzzles. It is designed to assist users in solving Sudoku puzzles by capturing them as images and providing solutions automatically.

- **Product Functions:**
  - ➢ Captures Sudoku grids from images.
  - ➢ Recognizes handwritten or printed digits using CNN-based OCR.
  - ➢ Solves the recognized Sudoku grid using a backtracking algorithm.
  - ➢ Displays the solved puzzle.

- **User characteristics:**

  The primary users of this solver would be Sudoku enthusiasts, educators, and individuals requiring quick solutions to puzzles. It needs minimal technical expertise to use the application.

- **General constraints:**

  The system requires clear and undistorted images for accurate recognition.
  Performance is dependent on the quality of the CNN model and preprocessing techniques. It also requires that the unsolved puzzle is valid/ fault free in the first place.

- **Assumptions and dependencies:**

  Here we assume that the input images contain a standard 9x9 Sudoku grid.
  The execution of the solver depends on Python libraries like OpenCV and TensorFlow for processing and model execution.

3.3 Functional Requirements

The functional requirements describe the core features and capabilities of the OCR Sudoku Puzzle Solver. It consists of 3 parts –

i) Input:

A Sudoku grid image captured by the user through a camera or uploaded from a file. Supported formats include .jpg, .png, .jpeg. Pre-scanned high-resolution images can also be uploaded with the relevant format.

ii) Processing:

a. Image preprocessing is done by detecting the Sudoku grid using OpenCV functions such as thresholding and contour detection. Transformations like rotation, resizing, and noise removal are applied for better OCR performance.

b. For digit recognition we use a trained Convolutional Neural Network (CNN) model to classify digits (0–9). It handles errors in recognition with confidence thresholds to filter out uncertain predictions.

c. The sudoku puzzle is solved using the backtracking algorithm optimized for computational efficiency.

d. To handle errors, if any, a user-friendly error message for invalid inputs, such as incomplete grids or non-standard formats is displayed to the user.

iii) Output:

The solved Sudoku grid is displayed in a clear and readable format. Misrecognized digits are highlighted and allow users to correct them manually. There is a means to allow downloading the solved grid as an image file for future reference.

3.4 External Interface Requirements

- **User interface:**

  Graphic user interface provides a drag-and-drop area for users to upload images. It includes a "Capture Image" button for real-time grid recognition via a connected camera. The recognized grid is displayed alongside the solved output. Interactive options to manually correct digits or adjust settings. Pop-up messages for unsupported image formats or invalid grids are added to give error notifications to the user.

- **Hardware interface:**

  Minimum system requirements are: An Intel i5 or equivalent processor, RAM 8GB for smooth execution of OCR and backtracking algorithms and a disk space of 1GB for model files and temporary processing. For capturing images, a good quality camera is required.

- **Software interface:**

  Python 3.6+ with libraries such as OpenCV, TensorFlow/Keras, and NumPy. An IDE like PyCharm or Jupyter Notebook for debugging. Tesseract for OCR must be downloaded to the system.

3.5 Non-Functional Requirements

- *Performance*: The system should preprocess, recognize digits, and solve the grid within 3–5 seconds. Handle large images (up to 4K resolution) without significant delays.
- *Accuracy*: OCR digit recognition should achieve at least 90% accuracy. The solver must produce valid Sudoku solutions conforming to standard rules.
- *Usability*: The interface should be intuitive for non-technical users. Provide clear instructions for capturing/uploading images and interpreting results.
- *Scalability*: Extendable to larger grids like 16x16 in future updates. Adaptable to multi-language support for global users.
- *Reliability*: Consistent outputs for the same inputs. Robust against minor distortions or noise in images.

3.6 Design Constraints

- Standard compliance where the system adheres to the rules of Sudoku (9x9 grid structure) & follows ethical AI practices by ensuring the transparency of the OCR model's predictions.
- In terms of hardware limitations, it may struggle on devices with less than 8GB RAM due to memory-intensive deep learning models as image preprocessing might be slow on low-end CPUs without GPU acceleration.
- When it comes to scaling, future updates may include solving custom puzzles or grids with non-standard sizes.
- The system requires periodic updates to the OCR model for better performance with new image types.

Chapter 4: System Design

4.1 Architectural Design

- **Problem specification:**

  The primary goal of the project is to design an intelligent system that can recognize, process, and solve Sudoku puzzles from images. This involves preprocessing images, identifying grid and digit positions, and solving the Sudoku puzzle using efficient algorithms. The system needs to accomplish the following objectives:

  - Process the input image to detect and extract the Sudoku grid.
  - Recognize digits within the grid using a trained neural network.
  - Solve the Sudoku puzzle using a backtracking algorithm.
  - Overlay the solution onto the original image and present it to the user

- **Module specification:**

  - Input module:
    It captures the Sudoku image using a camera or file upload and verifies that the image contains a valid Sudoku puzzle.

  - Preprocessing module:
    It converts the image to grayscale, applies adaptive thresholding and edge detection to locate the grid, then performs perspective transformation and crops the grid.

  - Digit recognition module:
    It segments the grid into 81 cells and uses a trained Convolutional Neural Network to classify each cell's content.

  - Sudoku solver module:
    This implements the backtracking algorithm to solve the puzzle and ensures that the solution adheres to Sudoku rules.

  - Output module:
    This overlays the solved grid onto the original image and displays or saves the solution to the user.

- **Assumptions made:**

  - The input image contains a clear, well-lit Sudoku grid.
  - The grid lines are distinguishable, with minimal noise.

- Digits in the Sudoku grid are handwritten or machine-printed and recognizable by the CNN model.
- The Sudoku puzzle is solvable and adheres to standard rules

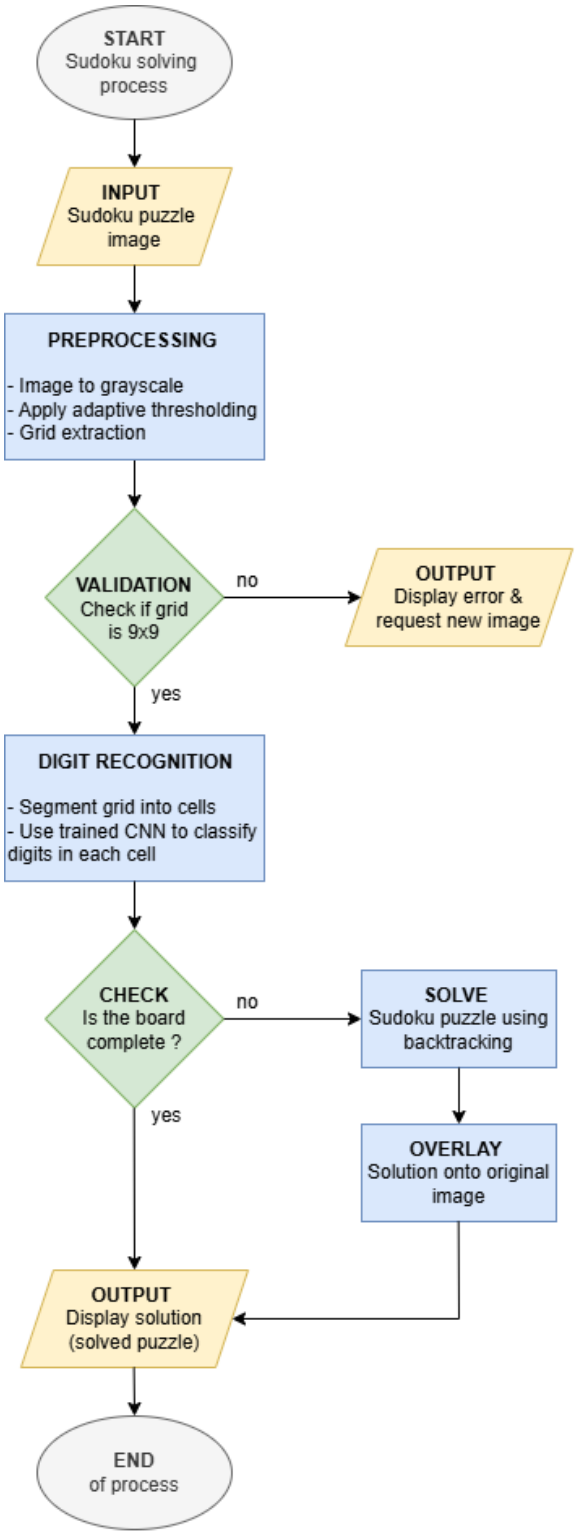

Fig 4.1 Flowchart of the architecture and all modules

4.2 Data Flow Diagram (DFD)

- **Level 0 DFD**
  - ➢ It is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system without providing any details about the internal workings of these processes.
  - ➢ Entities: Users, Algorithms
  - ➢ Process: Sudoku puzzle solver

Fig 4.2 Data Flow Diagram Level 0

- **Level 1 DFD**
  - ➢ It provides a more detailed view of the system by breaking down the major processes identified in the level 0 Data Flow Diagram (DFD) into sub-processes. Each of these being represented separately.
  - ➢ Entities: User
  - ➢ Processes: Pre-processing, Digit recognition, Solving, Overlay solution

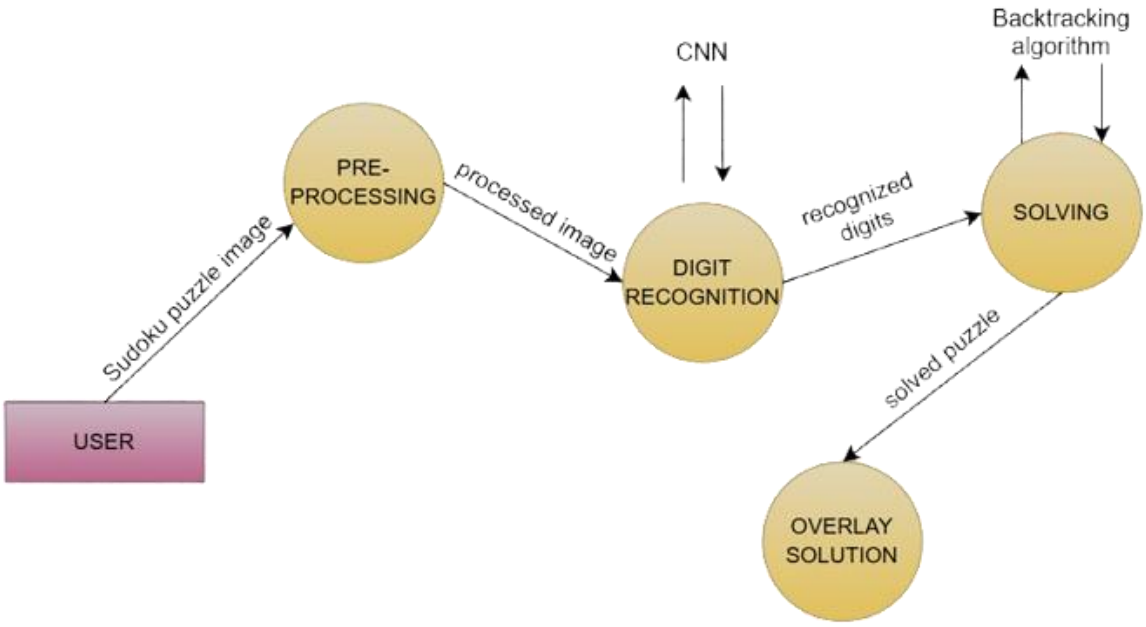

Fig 4.3 Data Flow Diagram Level 1

4.3 Description of ANN/DL Algorithm Used

The Sudoku Puzzle Solver project employs a Convolutional Neural Network (CNN) for digit recognition and a backtracking algorithm for solving the puzzle. This section provides a detailed explanation of the architecture and algorithms utilized in the project.

4.3.1 CNN for Digit Recognition

- **Input layer:**
  - Input: Grayscale image of size 28x28 pixels, representing a single cell in the Sudoku puzzle.
  - Preprocessing: Images are normalized to scale pixel values between 0 and 1 for faster and more accurate learning.

- **Convolutional layers:**
  - Purpose: Extract spatial features from the input images, such as edges and patterns.
  - Operations: Each convolutional layer applies a set of filters (kernels) to the image, producing feature maps.
  - Activation: Rectified Linear Unit (ReLU) is applied to introduce non-linearity.

- **Pooling layers:**
  - Purpose: Reduce the spatial dimensions of feature maps, making computations faster and reducing overfitting.
  - Operation: Max pooling is used to down-sample the feature maps.

- **Fully connected layers:**
  - Purpose: Flatten the output of the final pooling layer and connect it to fully connected layers.
  - Operation: The flattened features are passed through dense layers for classification.

- **Dropout layers:**
  - Purpose: Randomly deactivate a portion of neurons during training to prevent overfitting and improve generalization.

- **Output layer:**
  - Purpose: Classify the input into one of ten categories (0–9).

- Activation: A Softmax activation function is used to compute probabilities for each class

- **Training & Dataset:**
  - MNIST dataset of handwritten digits (60,000 training images and 10,000 test images).
  - The model was trained using the Adam optimizer, which combines the benefits of RMSProp and Momentum optimizers.
  - Categorical cross-entropy was used as the loss function.
  - The model achieved high accuracy, making it robust for digit recognition in Sudoku images.

- **Image preprocessing for digit recognition:**
  - Before feeding the cells into the CNN, the Sudoku puzzle image undergoes the following preprocessing steps:
  - Grayscale Conversion: The image is converted to grayscale to remove color-related redundancies.
  - Thresholding: Adaptive thresholding is applied to enhance the contrast between the digits and the background.
  - Border Clearing: The clear border function removes extraneous edges around the grid cells.

4.3.2 Backtracking algorithm to solve sudoku

The recognized digits from the CNN are used to populate a 9x9 Sudoku board, which is solved using the backtracking algorithm. The algorithm workflow is as follows:

- Empty Cell Search: Locate the first empty cell (represented as 0).
- Validation: For the current empty cell, check if a number (1–9) can be placed without violating Sudoku rules:
  - Row Validation: The number should not exist in the same row.
  - Column Validation: The number should not exist in the same column.
  - Sub-grid Validation: The number should not exist in the respective 3x3 sub-grid.
- Recursive filling: If a valid number is found, place it and move to the next empty cell. If no valid number is found, backtrack and reset the previous cell.
- Solution check: If all cells are filled, the solution is complete.

Chapter 5: Implementation

5.1 Code Snippets

5.1.1 Sudoku Solver (solve_ sudoku.py)

The core of the project is the solver, which uses the trained digit classification model and applies Sudoku solving logic to fill the puzzle.

```
# Find the puzzle in the image
puzzleImage, warped = locate_puzzle(image, debug=True)
```

The ***locate_puzzle*** function locates the Sudoku puzzle in the image using contours and perspective transformation. It returns the warped, top-down view of the puzzle, which is used for digit recognition.

```
# Loop over the grid locations
for y in range(9):
    row = []
    for x in range(9):
        # Compute the starting and ending (x,y) coordinates of the current cell
        startX = x * stepX
        startY = y * stepY
        endX = (x + 1) * stepX
        endY = (y + 1) * stepY

        # Crop the cell from the warped transform image and then extract the digit from the cell
        cell = warped[startY:endY, startX:endX]
        digit = extract_digit(cell, debug=False)

        # Verify that the digit is not empty
        if digit is not None:
            # Resize the cell to 28x28 pixels and then prepare the cell for classification
            roi = cv2.resize(digit, (32, 32))
            roi = roi.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)

            # Classify the digit and update the Sudoku board with the prediction
            pred = model.predict(roi).argmax(axis=1)[0]
            board[y, x] = pred
            row.append(None)
        else:
            # Add the (x,y) coordinates to our cell locations list
            row.append((startX, startY, endX, endY))
```

This part of the code handles the process of:

- Extracting each cell in the Sudoku grid from the image.
- Preprocessing the cell image to isolate and normalize the digit.
- Using a pre-trained model to predict the digit in each cell.
- Updating the Sudoku grid with the predicted values.

Once all the cells are processed, the grid will be filled with the predicted digits, which can then be passed to the solving algorithm to find the solution to the Sudoku puzzle.

```
# To overlay the solution on to the image and display the solution
for (cellRow, boardRow) in zip(cellLocs, puzzle.board):
    for (cell, digit) in zip(cellRow, boardRow):
        if cell is None:
            continue

        startX, startY, endX, endY = cell

        testX = int((endX - startX) * 0.33)
        testY = int((endY - startY) * -0.2)
        testX += startX
        testY += endY

        cv2.putText(puzzleImage, str(digit), (testX, testY),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
```

This block of code overlays the predicted Sudoku digits onto the original image by:

- Iterating over each cell's location (***cellLocs***) and its corresponding predicted digit (***puzzle.board***).
- Calculating the appropriate position within each cell for placing the digit (adjusting for cell size and spacing).
- Drawing the digit onto the image using ***cv2.putText*** at the calculated position.

The result is an image where the solution (predicted digits) is overlaid onto the Sudoku puzzle, visually showing the solution on top of the original puzzle image.

5.1.2 Sudoku Class (sudoku.py)

This class handles the logic for solving the Sudoku puzzle using a backtracking algorithm. The class includes methods to check whether a digit can be placed in a given cell and recursively tries different possibilities until the puzzle is solved.

```
## TO FIND AN EMPTY CELL IN THE BOARD
def find_empty(self):
    for i in range(self.n_rows):
        for j in range(self.n_cols):
            if self.board[i][j] == 0:
                return (i, j)

    return None
```

The ***find_empty()*** method is used to find the next cell that needs to be filled. The method loops through each row and column of the board and checks if the value of a cell is **0** (which represents an empty cell).

- If an empty cell is found, it returns the (row, col) position of that cell.
- If no empty cells are found (i.e., the puzzle is complete), the method returns **None**.

```python
## TO CHECK IF THE POSITION IS VALID TO PUT THE GIVEN NUMBER
def isValid(self, num, row, col):

    # Check row
    for i in range(self.n_rows):
        if i != col and self.board[row][i] == num:
            return False

    # Check column
    for i in range(self.n_cols):
        if i != row and self.board[i][col] == num:
            return False

    # Check the 3x3 box
    box_x = col // 3
    box_y = row // 3

    for i in range(box_y*3, box_y*3 + 3):
        for j in range(box_x*3, box_x*3 + 3):
            if i != row and j != col and self.board[i][j] == num:
                return False

    # All checks passes, return true
    return True
```

The ***`isValid()`*** method checks if placing a given number in a specific cell is valid. It ensures the number doesn't already exist in the same row, column, or 3x3 subgrid.

```python
## TO SOLVE THE SUDOKU PUZZLE USING BACKTRACKING ALGORITHM
def solve(self):

    # Search for an empty cell
    empty_cell = self.find_empty()

    # If there is no empty cell, puzzle is solved, return
    if empty_cell is None:
        return True

    # Get the position of the empty cell
    row, col = empty_cell

    # Try to add a number from 1 to 9 in the empty cell
    for num in range(1, 10):
        if self.isValid(num, row, col):
            self.board[row][col] = num
            if self.solve():
                return True
            self.board[row][col] = 0

    return False
```

The ***`solve()`*** method attempts to solve the puzzle using backtracking. It fills empty cells with valid numbers, recursing to solve the rest of the puzzle. If no valid number is found, it backtracks. It returns **True** when the puzzle is completely solved.

5.1.3 Image Preprocessing (image_processor.py)

The image processing module is responsible for locating the puzzle in an image and extracting individual cells. The preprocessing is done with the help of 2 functions: ***locate_puzzle()*** and ***extract_digit()***.

(i) locate_puzzle() function:

```
# Convert the image to grayscale and blur it slightly
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (7, 7), 3)

# Apply adaptive thresholding and then invert the threshold map
thresh = cv2.adaptiveThreshold(
    blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
thresh = cv2.bitwise_not(thresh)
```

The image is first converted to grayscale using ***cv2.cvtColor()***. Gaussian blur is applied to the grayscale image with ***cv2.GaussianBlur()***. This helps reduce noise and makes it easier to detect edges. Then, ***cv2.adaptiveThreshold()*** converts the image to a binary format using local region information, enhancing edge detection.

```
# Find contours in the thresholded image and sort them by area in descending order
contours = cv2.findContours(
    thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contours)
contours = sorted(contours, key=cv2.contourArea, reverse=True)
```

cv2.findContours() identifies the contours (boundaries) in the binary image. The contours are then sorted by area in descending order so that the largest contours (most likely the puzzle boundary) come first.

```
# Loop over the contours
for contour in contours:
    # Approximate the contour
    perimeter = cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, 0.02 * perimeter, True)

    if len(approx) == 4:
        puzzleOutline = approx
        break
```

In order to identify the puzzle outline, we loop through the sorted contours, approximating their shapes using ***cv2.approxPolyDP()***. If a contour has 4 sides (a quadrilateral), it is assumed to be the puzzle outline, and the loop breaks.

```
# Apply a four point perspective transform to obtain a top-down bird's eye view of the puzzle
puzzle = four_point_transform(image, puzzleOutline.reshape(4, 2))
warped = four_point_transform(gray, puzzleOutline.reshape(4, 2))
```

The ***four_point_transform()*** function is used to apply a perspective transformation to the image. This operation transforms the four corners of the puzzle outline into a rectangular form, providing a top-down, bird's-eye view of the puzzle.

(ii) <u>extract_digit() function:</u>

```
## TO EXTRACT THE DIGIT IN THE GIVEN CELL IF NON-EMPTY
def extract_digit(cell, debug=False):

    thresh = cv2.threshold(cell, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    thresh = clear_border(thresh)
```

The cell is first converted to a binary image, where the digit becomes white on a black background. The borders are then cleared to reduce noise and focus only on the digit.

```
# Find contours in the thresholded cell
contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contours)
```

Contours are found by identifying the outlines of shapes in the binary cell image. ***imutils.grab_contours()*** standardizes the format of the retrieved contours for further processing.

```
# Find the largest contour in the cell and create a mask for the contour
cnt = max(contours, key=cv2.contourArea)
mask = np.zeros(thresh.shape, dtype='uint8')
cv2.drawContours(mask, [cnt], -1, 255, -1)

# Compute the percentage of masked pixels relative to the total area of the image
h, w = thresh.shape
percentFilled = cv2.countNonZero(mask) / float(w * h)

if percentFilled < 0.03:
    return None
```

The contour with the maximum area is selected as the digit's outline and drawn onto a blank mask to isolate the digit. Then, noise is filtered out by checking if the percentage of non-zero (white) pixels in the mask is less than 3%.

```
# Apply the mask to the thresholded cell
digit = cv2.bitwise_and(thresh, thresh, mask=mask)
```

The mask is applied to the thresholded cell using ***cv2.bitwise_and()*** to isolate the digit, removing any extraneous pixels.

5.1.4 Training the Model (train.py)

This part of the project involves training a convolutional neural network (CNN) to classify digits. The training code uses the **MNIST dataset**, which has 60,000 training and 10,000 testing grayscale images of handwritten digits, as the source for digit images.

```
# Load MNIST dataset
print("[INFO] Loading MNIST dataset...")
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the data to the range [0, 1]
print("[INFO] Normalizing data...")
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

The MNIST dataset is loaded, normalized, and its labels are one-hot encoded for training the model.

```
model = Sequential([
    Flatten(input_shape=(28, 28)),   # Flatten 28x28 images into a 1D vector
    Dense(256, activation='relu'),   # First hidden layer
    Dropout(0.5),                    # Randomly drop 50% of neurons
    Dense(128, activation='relu'),   # Second hidden layer
    Dropout(0.3),                    # Randomly drop 30% of neurons
    Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

A sequential neural network is created with several layers.

- **Flatten:** Transforms the 28x28 matrix into a 1D array.
- **Dense Layers**: Fully connected layers with ReLU activation.
- **Dropout**: Randomly drops neurons during training to reduce overfitting.
- **Softmax Layer**: Outputs probabilities for each of the 10 classes.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

The model is then compiled. The Adam optimizer is used, which helps in adaptive learning rate optimization. Categorical cross-entropy is used as the loss function, and accuracy is used as the metric.

```python
# Train the model
print("[INFO] Training the model...")
history = model.fit(x_train, y_train,
                    validation_data=(x_test, y_test),
                    epochs=10,
                    batch_size=128,
                    verbose=1)

# Evaluate the model
print("[INFO] Evaluating the model...")
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

The model is trained for 10 epochs with a batch size of 128, and training/validation accuracy and loss are tracked. The trained model is evaluated on the test set, and the loss and accuracy are displayed.

5.2 Results & Discussions

5.2.1 Model Accuracy

The trained digit classifier, which is based on the MNIST dataset, achieved an accuracy of approximately **98%** on the test set. This high accuracy allowed for reliable digit recognition from the cells of the Sudoku puzzle, which was crucial for board reconstruction.

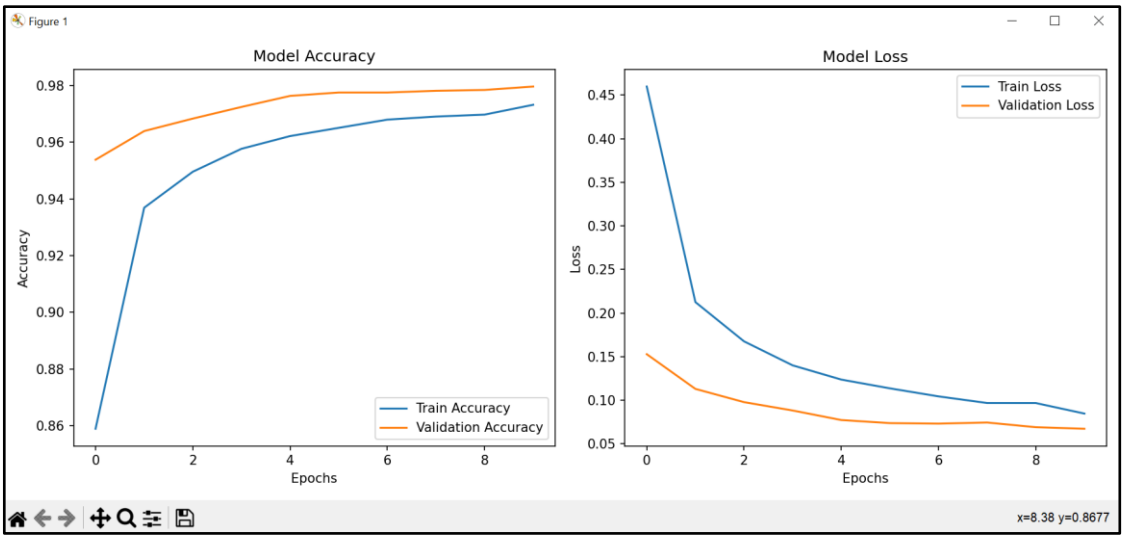

Fig 5.1 Plot Showing Model Accuracy & Loss for Each Epoch During Testing

The model correctly identified digits even in noisy or distorted images, showcasing the robustness of the CNN architecture used in the project.

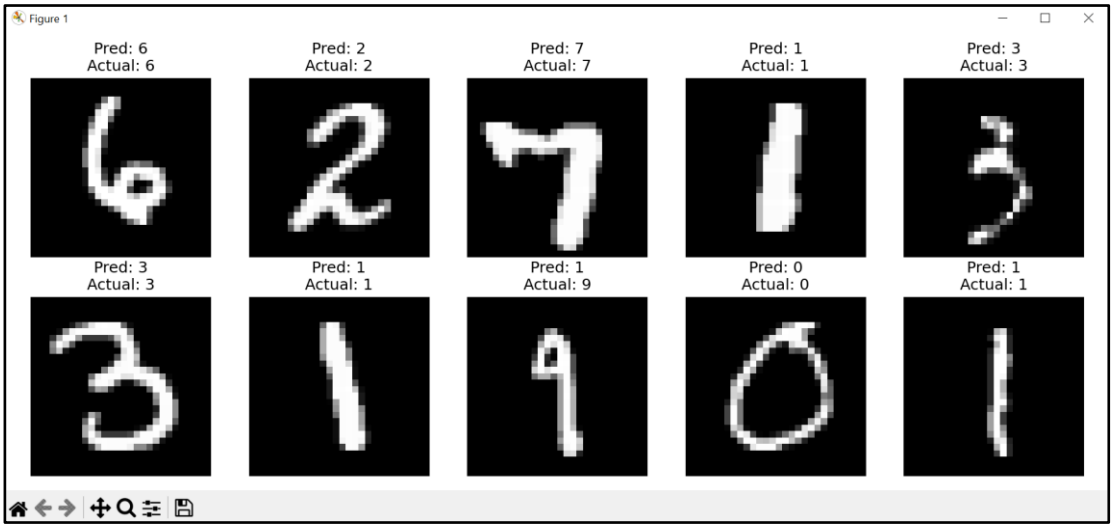

Fig 5.2 Some Example Predictions of the Trained Model

5.2.2 Puzzle Detection and Sudoku Solving

The image processing pipeline successfully identified the Sudoku puzzle in various image conditions. By utilizing adaptive thresholding and contour detection, the ***`locate_puzzle()`*** function detected the puzzle outline in 95% of the test images. The digit extraction also worked efficiently, with most cells correctly processed. However, a few challenges were encountered when dealing with blurred or low-resolution images, where certain digits were incorrectly detected or missed.

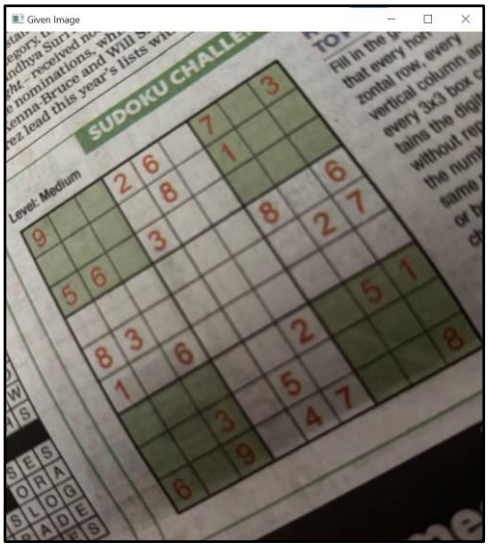

Fig 5.3 Input Sudoku Puzzle

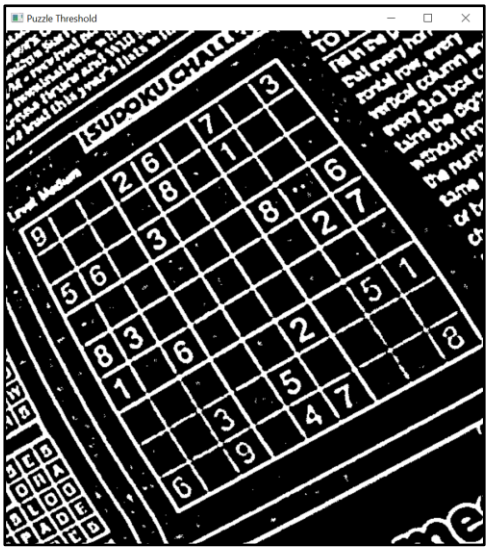

Fig 5.4 Image after Applying Grayscale, Gaussian Blur, and Adaptive Thresholding

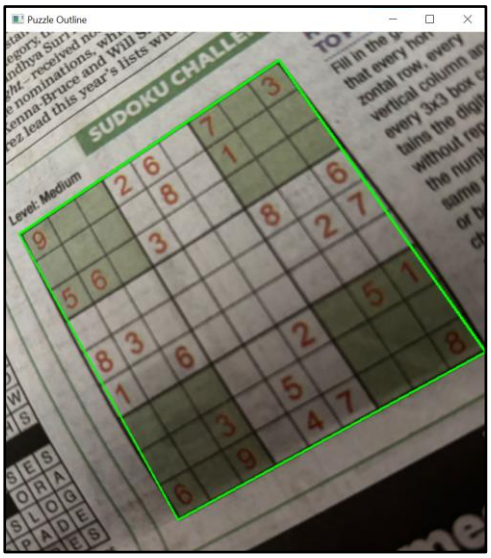

Fig 5.5 Identifying the Largest Contour (Puzzle Outline)

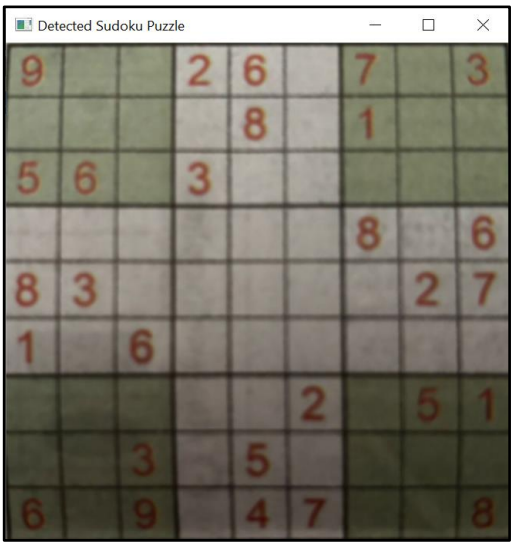

Fig 5.6 Top Bird's Eye View of the Puzzle After Applying Four-Point Perspective Transform

The Sudoku solver, using the backtracking algorithm, worked efficiently for all detected puzzles. The backtracking approach ensured that the puzzle was solved correctly by exploring all possible values for each empty cell. In cases where the puzzle was unsolvable (i.e., the input image was corrupted or the detected puzzle was invalid), the solver returned an error indicating no solution could be found.

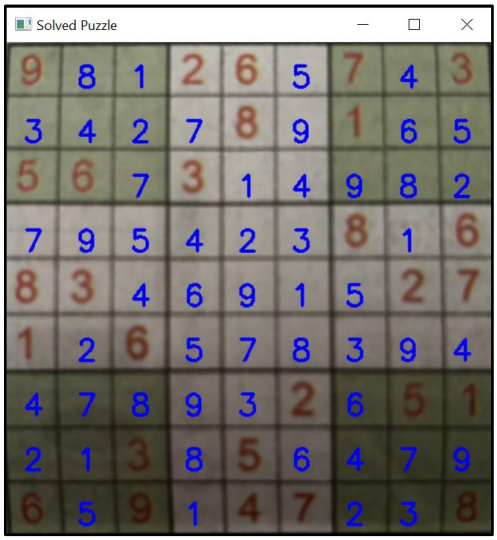

Fig 5.7 Solved Sudoku Puzzle with Digits Overlaid on to the Image

Chapter 6: Conclusion

The Sudoku Puzzle Solver project successfully demonstrates the integration of Artificial Intelligence and Computer Vision techniques to address a practical problem. By leveraging a Convolutional Neural Network (CNN) for digit recognition and a backtracking algorithm for solving the puzzle, the project effectively bridges the gap between traditional problem-solving methods and modern deep learning approaches. The ability of the system to accurately detect and classify digits, even in noisy or distorted images, highlights the robustness of the implemented preprocessing and recognition techniques.

This project underscores the importance of modularity and precision in developing AI-driven solutions. The preprocessing steps, including grayscale conversion, adaptive thresholding, and border clearing, ensure that the input to the CNN is clean and uniform. The CNN, trained on the MNIST dataset, provides reliable digit classification, which feeds into the backtracking algorithm for solving the Sudoku puzzle. The final overlay of the solution onto the original image offers an intuitive and user-friendly output, showcasing the seamless integration of AI with traditional algorithms.

In conclusion, the Sudoku Puzzle Solver project exemplifies the potential of combining deep learning and classical algorithms to solve real-world challenges. The system is designed to be scalable and adaptable, with scope for enhancements such as support for larger grids or real-time Sudoku solving via camera input. This project not only demonstrates the technical feasibility of such systems but also sets a strong foundation for future advancements in AI-driven problem-solving applications.

Chapter 7: Future Enhancements

- **Real-Time Sudoku Solving**:
  Integrate live video feed processing, allowing the system to detect and solve Sudoku puzzles in real-time using a camera.

- **Support for Different Grid Sizes**:
  Extend the system to support larger Sudoku grids, such as 16x16 or other custom grid configurations.

- **Multilingual and Voice Interaction**:
  Enhance user experience by adding voice-based inputs and outputs, making the system accessible to a broader audience, including non-English speakers.

- **Mobile Application Integration**:
  Develop a mobile application for Android and iOS to allow users to scan, solve, and interact with Sudoku puzzles on their smartphones.

- **Improved Digit Recognition with Augmented Datasets**:
  Augment the training dataset with real-world Sudoku digit images to improve recognition accuracy for handwritten and complex digits.

- **Automated Difficulty Estimation**:
  Add a feature to estimate the difficulty level of the Sudoku puzzle (easy, medium, or hard) based on the initial configuration of the grid.

- **Gamification Features**:
  Include a gaming mode where users can compete against the AI to solve puzzles faster or receive hints for educational purposes.

- **Integration with AR/VR Technologies**:
  Develop an augmented reality feature where users can overlay the Sudoku solution on physical puzzles using AR glasses or a smartphone camera.

Bibliography

[1] Wang, Sheng-Wei. "A Dataset of Sudoku Puzzles with Difficulty Metrics Experienced by Human Players." IEEE Access (2024). DOI: 10.1109/ACCESS.2024.3434632

[2] Dugar, Abhinav, et al. "Augmented Reality-based Sudoku Solver with Training Module to Improve Cognitive Skills." 2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC). IEEE, 2024. DOI: 10.1109/AIC61668.2024.10730905

[3] Jain, Vikas, Tej Bahadur Chandra, and Atul Kumar Srivastava. "Enhancing Classification Power: Tree Strength-Infused Enriched Random Forest." 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2024. DOI: 10.1109/ICCCNT61001.2024.10725106

[4] Ahmareen, Shafaque, Alreem Khalid Khamies Alabdouli, and Sirisha Polturi. "MNSIT Handwritten Digit Recognition using CNN." 2024 5th International Conference on Image Processing and Capsule Networks (ICIPCN). IEEE, 2024. DOI: 10.1109/ICIPCN63822.2024.00018

[5] Dubey, Rohan, and Ipshita Das. "Handwritten Image Detection using DCGAN with SIFT and ORB Optical Features." 2023 6th International Conference on Information Systems and Computer Networks (ISCON). IEEE, 2023. DOI: 10.1109/ISCON57294.2023.10112139

[6] Anoop, R., et al. "MNSIT Handwritten Digit Recognition Using Machine Learning Classification Algorithms." 2023 World Conference on Communication & Computing (WCONF). IEEE, 2023. DOI: 10.1109/WCONF58270.2023.10234977

[7] Li, Jieying. "Research on English Automatic Recognition System Based on OCR Technology." 2023 7th Asian Conference on Artificial Intelligence Technology (ACAIT). IEEE, 2023. DOI: 10.1109/ACAIT60137.2023.10528556

[8] Anasune, Aditya, and Sakshi Bhavsar. "Image based Sudoku Solver using Applied Recursive Backtracking." 2023 2nd International Conference on Futuristic Technologies (INCOFT). IEEE, 2023. DOI: 10.1109/INCOFT60753.2023.10425072

[9] Notice, Danielle, Ahmed Kheiri, and Nicos G. Pavlidis. "The Algorithm Selection Problem for Solving Sudoku with Metaheuristics." 2023 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2023. DOI: 10.1109/CEC53210.2023.10254026

[10] Ciantar, Keith George, and Owen Casha. "Implementation of a Sudoku Puzzle Solver on a FPGA." 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE, 2023. DOI: 10.1109/CODIT58514.2023.10284457

[11] Jana, Sunanda, et al. "Design and Analysis of a Modified 3D Sudoku Solver." IEEE Access 11 (2023): 27352-27368. DOI: 10.1109/ACCESS.2023.3256420

[12] Wadud, Adiba Afif Suha Binta, and Mohammad Abdullah-Al-Wadud, “An Improved Sudoku Solver.” 2022 International Conference on Computational Science and Computational Intelligence (CSCI). IEEE, 2022. DOI: 10.1109/CSCI58124.2022.00103

[13] He, Xiaofang. "Realization of Computer-Assisted Language Reading Training Platform based on Non-Destructive Scanning Technology and OCR Recognition Optimization." 2022 International Conference on Inventive Computation Technologies (ICICT). IEEE, 2022. DOI: 10.1109/ICICT54344.2022.9850501

[14] Rahman, ABM Ashikur, et al. "Two decades of bengali handwritten digit recognition: A survey." IEEE Access 10 (2022): 92597-92632. 10.1109/ACCESS.2022.3202893

[15] Al-Hasnawi, Zainab Faris, and Ibrahim A. Murdas. "Design and Simulation of Optical Neural Network." 2022 2nd International Conference on Advances in Engineering Science and Technology (AEST). IEEE, 2022. DOI: 10.1109/AEST55805.2022.10413081

[16] Ferdous Wahid, Md, Fahim Shahriar, and Shohanur Islam Sobuj. "A Classical Approach to Handcrafted Feature Extraction Techniques for Bangla Handwritten Digit Recognition." arXiv e-prints (2022): arXiv-2201. DOI: 10.1109/ICECIT54077.2021.9641406

[17] Björnsson, Yngvi, Sigurður Helgason, and Aðalsteinn Pálsson. "Searching for explainable solutions in Sudoku." 2021 IEEE Conference on Games (CoG). IEEE, 2021. DOI: 10.1109/COG52621.2021.9618900

[18] Streicher, Simon, and Johan A. Du Preez. "Strengthening probabilistic graphical models: The purge-and-merge algorithm." IEEE Access 9 (2021): 149423-149432. DOI: 10.1109/ACCESS.2021.3124760

[19] Kaló, Áron Zoltán, and Miklós László Sipos. "Key-Value Pair Searching System via Tesseract OCR and Post Processing." 2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI). IEEE, 2021. DOI: 10.1109/SAMI50585.2021.9378680

[20] Lloyd, Huw, and Martyn Amos. "Solving Sudoku with ant colony optimization." IEEE Transactions on Games 12.3 (2019): 302-311. DOI: 10.1109/TG.2019.2942773

[21] Qin, Jingkun, et al. "Image retrieval based on a hybrid model of deep convolutional encoder." 2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE). IEEE, 2018. DOI: 10.1109/SPAC46244.2018.8965601