

SOLVING SUDOKU WITH COMPUTER VISION

**by
Ali Ulaş Hayır**

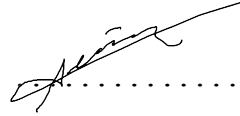
Engineering Project Report

**Yeditepe University
Faculty of Engineering
Department of Computer Engineering
2023**


SOLVING SUDOKU WITH COMPUTER VISION

APPROVED BY:

Assist. Prof. Dr. Onur Demir
(Supervisor)

A black ink signature of Assist. Prof. Dr. Onur Demir, written in a cursive style, positioned above a horizontal dotted line.

Prof. Dr. Sezer Gören Uğurdağ

A blue ink signature of Prof. Dr. Sezer Gören Uğurdağ, written in a cursive style, positioned above a horizontal dotted line.

Assoc. Prof. Dr. Mert Özkaya

A black ink signature of Assoc. Prof. Dr. Mert Özkaya, written in a cursive style, positioned above a horizontal dotted line.

DATE OF APPROVAL: 12/06/2023

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Assist. Prof. Dr. Onur Demir for his guidance and support throughout my project.

Also I would like to thank my parents for their support and encouragement throughout my education up to the present.

ABSTRACT

SOLVING SUDOKU WITH COMPUTER VISION

Sudoku, a popular logic-based number placement puzzle, challenges players to fill a 9×9 grid with digits so that each column, row, and 3×3 subgrid contains all the numbers from 1 to 9. With its well-defined rules and a unique solution, Sudoku puzzles can be efficiently solved using algorithms, making it an ideal task for computers. In this research paper, we explore the development and evaluation of a Sudoku problem solver that leverages deep learning and computer vision techniques.

Our system employs convolutional neural networks (CNNs) trained on the MNIST dataset to detect handwritten digits within the Sudoku grid. By combining the power of machine learning and logical reasoning, our approach demonstrates the effectiveness of pattern identification and algorithmic problem solving. Once the digits are recognized, the solver utilizes the backtracking algorithm to systematically fill in the missing numbers and solve the Sudoku puzzle.

Through this fusion of computer vision and deep learning, we showcase the potential of machine learning algorithms in enhancing logical problem-solving abilities. This work highlights the synergy between different domains of artificial intelligence and paves the way for further advancements in applying machine learning to complex logical puzzles like Sudoku.

ÖZET

BİLGİSAYAR GÖRÜŞÜ İLE SUDOKU ÇÖZMEK

Mantığa dayalı popüler bir sayı yerleştirme bulmacası olan Sudoku, oyuncularını 9×9 'luk bir ızgarayı her sütun, satır ve 3×3 alt ızgara 1'den 9'a kadar tüm sayıları içerecek şekilde rakamlarla doldurmaya zorlar. Eşsiz bir çözüm olan Sudoku bulmacaları, algoritmalar kullanılarak verimli bir şekilde çözülebilir ve bu da onu bilgisayarlar için ideal bir görev haline getirir. Bu araştırma makalesinde, derin öğrenme ve bilgisayarla görme tekniklerinden yararlanan bir Sudoku problem çözücünün geliştirilmesini ve değerlendirilmesini araştırıyoruz.

Sistemimiz, Sudoku ızgarasındaki el yazısı rakamları algılamak için MNIST veri kümesinde eğitilmiş evrişimli sinir ağlarını (CNN'ler) kullanır. Makine öğreniminin ve mantıksal akıl yürütmenin gücünü birleştiren yaklaşımımız, örüntü tanımlamanın ve algoritmik problem çözmenin etkinliğini gösterir. Rakamlar tanıdıktan sonra, çözücü eksik sayıları sistematik olarak doldurmak ve Sudoku bulmacasını çözmek için geri izleme algoritmasını kullanır.

Bilgisayar görüşü ve derin öğrenmenin bu birleşimi sayesinde, mantıksal problem çözme yeteneklerini geliştirmede makine öğrenimi algoritmalarının potansiyelini sergiliyoruz. Bu çalışma, yapay zekanın farklı alanları arasındaki sinerjiyi vurguluyor ve makine öğrenimini Sudoku gibi karmaşık mantıksal bulmacalara uygulamada daha fazla ilerlemenin yolunu açıyor.

TABLE OF CONTENTS

| | |
|--|------|
| ACKNOWLEDGEMENTS | iii |
| ABSTRACT | iv |
| ÖZET | v |
| LIST OF FIGURES | viii |
| LIST OF TABLES | ix |
| LIST OF SYMBOLS/ABBREVIATIONS | x |
| 1. INTRODUCTION | 1 |
| 1.1. Convolutional Neural Networks | 1 |
| 1.2. MNSIT Database | 2 |
| 1.3. Backtracking Algorithm | 2 |
| 1.4. Valid Sudoku Grids | 2 |
| 1.5. Image Processing and Sudoku Solving | 3 |
| 1.6. Terms | 3 |
| 1.7. Motivation | 5 |
| 1.7.1. Accessibility | 5 |
| 1.7.2. As a Basis for Other Applications | 5 |
| 1.8. Scope and Limitations | 5 |
| 1.9. Aim Of the Project | 6 |
| 1.10. Problem Definition | 6 |
| 1.11. Requirements | 7 |
| 1.12. Requirements Test and Results | 7 |
| 2. REQUIREMENT ANALYSIS | 9 |
| 2.1. Analysing Requirements | 9 |
| 2.1.1. Functional and Non-functional Requirements | 11 |
| 3. BACKGROUND | 16 |
| 3.1. Previous works | 16 |
| 3.1.1. Mixed handwritten and printed digit recognition in Sudoku with Con- volutional Deep Belief Network: | 16 |
| 3.1.2. Detection of Sudoku puzzle using image processing and solving by Backtracking, Simulated Annealing and Genetic Algorithms: A com- parative analysis | 16 |
| 3.1.3. Augmented Reality for Automatic Identification and Solving Sudoku Puzzles Based on Computer Vision | 17 |
| 3.1.4. Recursive backtracking for solving 9*9 sudoku puzzle | 17 |

| | |
|---|----|
| 3.1.5. Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study | 17 |
| 4. ANALYSIS | 19 |
| 4.1. Overview | 19 |
| 4.2. Image Processing | 22 |
| 4.3. Sudoku Solver Module | 26 |
| 4.4. Digit Recognition Module | 28 |
| 4.4.1. Model Architecture | 29 |
| 4.4.2. Convolutional Layers | 29 |
| 4.4.3. ReLU Activation Function | 29 |
| 4.4.4. Dropout | 30 |
| 4.4.5. Max Pooling | 30 |
| 4.4.6. Flattening Layer | 30 |
| 4.4.7. Dense Layer | 30 |
| 4.4.8. One-Hot Encoding | 30 |
| 4.4.9. Categorical Cross-Entropy Loss Function | 31 |
| 5. DESIGN AND IMPLEMENTATION | 32 |
| 5.1. Image Processing | 32 |
| 5.2. Digit Recognition | 33 |
| 5.3. Graphical User Interface | 34 |
| 5.4. Sudoku Solving | 36 |
| 5.5. Libraries Used | 38 |
| 5.6. Testing | 39 |
| 5.7. Deployment | 39 |
| 6. TEST AND RESULTS | 40 |
| 6.1. Digit Recognition Accuracy | 40 |
| 6.2. Locating Sudoku Puzzles | 42 |
| 6.3. Backtracking | 45 |
| 6.3.1. Average Time Taken | 45 |
| 6.4. Memory Usage | 48 |
| 7. CONCLUSION | 50 |
| 7.1. Achieved Aims | 50 |
| 7.2. Future Work | 51 |
| Bibliography | 52 |

LIST OF FIGURES

| | | |
|------|--|----|
| 1.1 | CNN figure | 1 |
| 1.2 | Examples From Dataset | 2 |
| 1.3 | Example Valid Sudoku | 3 |
| 4.1 | Use case Diagram | 20 |
| 4.2 | Sequence Diagram | 21 |
| 4.3 | UML Activity Diagram For Image Processing | 24 |
| 4.4 | Finding Sudoku Procedure | 25 |
| 4.5 | Finding Sudoku Procedure | 25 |
| 4.6 | Solving Sudoku with Backtracking | 26 |
| 4.7 | Check For Valid Cell | 27 |
| 4.8 | Check If Sudoku Board Is Valid | 28 |
| 4.9 | CNN creation | 28 |
| 4.10 | Enter Caption | 29 |
| 5.1 | Preprocessing of the Sudoku Image | 33 |
| 5.2 | How the cells look like | 33 |
| 5.3 | File Browsing and Model Picking | 34 |
| 5.4 | OCR'd Sudoku Puzzle | 35 |
| 5.5 | Overall Implementation | 37 |
| 6.1 | Accuracy by Category | 41 |
| 6.2 | Training and Validation Accuracy Over Epochs | 43 |
| 6.3 | Training and Validation Accuracy Over Epochs | 43 |
| 6.4 | Confusion Matrix for MNSIT Model | 44 |
| 6.5 | Taken Timen to Solve Sudoku on i5-11400H | 46 |
| 6.6 | Taken Timen to Solve Sudoku on i7-9750H | 47 |
| 6.7 | Memory usage i5-11400H | 49 |
| 6.8 | Memory usage i7-9750H | 49 |

LIST OF TABLES

| | | |
|-----|--|----|
| 6.1 | Tests | 40 |
| 6.2 | Misidentified Cell Average By Category | 41 |
| 6.3 | Finding Puzzles Within Images | 42 |

LIST OF SYMBOLS/ABBREVIATIONS

| | |
|------------|---|
| δ_t | Threshold |
| σ | Standart Deviation |
| f_0 | Fundamental frequency |
| K | Kernel |
| * | Convolution |
| I | Image |
| \circ | Hadamard Product |
| μ | Magnitude |
| ReLU | Rectifier Function |
| X | Complex STFT Output |
| | |
| OCR | Optical Character Recognition |
| CNN | Convolutional Neural Network |
| GUI | Graphical User Interface |
| ABI | Application Binary Interface |
| CLI | Command Line Interface |
| CPU | Central Processing Unit |
| DFT | Discrete Fourier transform |
| DSP | Digital Signal Processing |
| FFT | Fast Fourier Transform |
| FFI | Foreign Function Interface |
| GPU | Graphics Processing Unit |
| MNIST | Modified National Institute of Standards and Technology Database |

1. INTRODUCTION

Machine learning, computer vision and traditional computational methods working together has expanded the possibilities for decision-making and problem-solving models. The goal of this project is to create a Sudoku puzzle solver utilizing a backtracking algorithm and a Convolutional Neural Network (CNN) for digit recognition. It illustrates the use of artificial intelligence in a logical and abstract setting, showcasing the effectiveness and precision of the combined method.

1.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs), a specialized kind of neural network model, are designed for processing grid-like data such as an image and have been pivotal in tasks related to computer vision. They are primarily used in pattern recognition within images, enabling machines to visually understand surroundings and make decisions accordingly [1]. The network learns to extract increasingly complex features of the input data by applying convolutional filters and pooling operations (Figure1.1), ultimately leading to powerful image classification models.

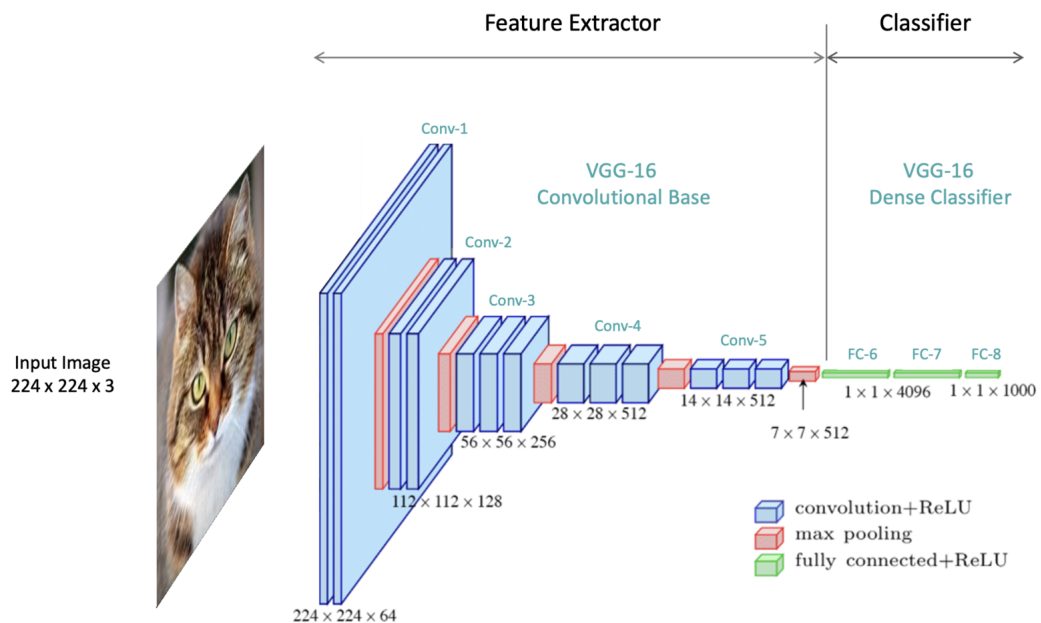


Figure 1.1: CNN figure

1.2. MNSIT Database

The Modified National Institute of Standards and Technology (MNIST) dataset is a widely-used collection of handwritten digits, extensively employed for training image processing systems, particularly in the field of machine learning. Comprising 60,000 training images and 10,000 testing images, each measuring 28x28 pixels, this dataset serves as a foundational resource for digit recognition tasks (Figure 1.2). In our project, we utilize the MNIST dataset to train our CNN model, enabling it to accurately interpret handwritten numbers. [2]

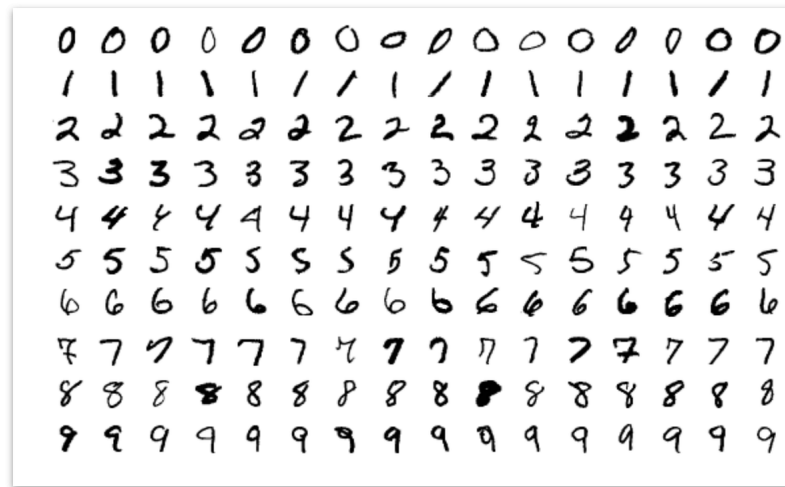


Figure 1.2: Examples From Dataset

1.3. Backtracking Algorithm

Backtracking is a powerful algorithmic technique that solves problems recursively by progressively constructing a solution. When it becomes evident that the current approach cannot lead to a valid solution, the process backtracks to the previous step and explores alternative options. In Sudoku, backtracking is employed to fill each cell with a suitable digit. Whenever a digit choice leads to a conflict, the algorithm backtracks and attempts a different digit until a successful solution is found.

1.4. Valid Sudoku Grids

The completeness and correctness of a Sudoku grid are integral parts of the problem-solving approach. A valid grid must follow the rules of Sudoku, which state that every row, column, and predefined 3x3 box must contain all the digits from 1 to 9 without repetition. Any

proposed solution that violates these rules is invalid. To ensure the accuracy and efficiency of the proposed model, a grid validation step is incorporated, which checks the legitimacy of the filled grid.(Figure1.3)

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

Figure 1.3: Example Valid Sudoku

1.5. Image Processing and Sudoku Solving

The advent of Computer Vision and Deep Learning has paved the way for machines to process and understand visual data with impressive accuracy. This project harnesses the power of these technologies to solve Sudoku puzzles using image processing techniques and machine learning models. In particular, our project employs the use of Contours, Gaussian Blur, Adaptive Thresholding, and Bitwise Operations, all fundamental concepts in the field of Image Processing. Through these techniques, we are able to identify and isolate each cell of a Sudoku puzzle from a given image.[3]

1.6. Terms

- *Sudoku* is a puzzle game where a grid of 9x9 squares is to be filled with digits so that each column, each row, and each of the nine 3x3 sub-grids that compose the grid contains all of the digits from 1 to 9.

- *CNN* Deep, feed-forward artificial neural networks are commonly utilized to examine visual imagery, belonging to a specific category of networks.
- *A Backtracking* An algorithmic technique for solving problems recursively by trying to build a solution incrementally, solving one piece at a time, and removing those solutions that fail to satisfy the constraints of the problem at any point of time.
- *A Grid* a two-dimensional structure composed of intersecting vertical and horizontal lines, in which each cell is a distinct unit. In the context of Sudoku, a grid refers to the 9x9 puzzle layout.
- *Epoch* In the context of machine learning, an epoch is one complete pass through the entire training dataset.
- *One-hot encoding* A process of converting categorical data variables so they can be provided to machine learning algorithms to improve predictions. With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0.
- *Activation Function* In neural networks, an activation function defines the output of a node given an input or set of inputs.
- *Flattening* the process of converting a data structure into a one-dimensional form. In the context of a neural network, it's the process of converting the final pooling layer output into a single continuous linear vector.
- *Grayscale* A range of shades of gray without apparent color. Each shade is a monochromatic depiction of a corresponding level of brightness in white light.
- *Pixel Intensity* The value of a pixel in an image. In grayscale images, this intensity is a value between 0 (black) and 255 (white).
- *Binary Image* an image in which each pixel takes only two values, usually 0 and 1. Binary images are often produced by thresholding a grayscale or color image, in order to separate the object in the image from the background.
- *Contours* in image processing, refer to the continuous curves that outline objects with the same color or intensity. They are valuable for shape analysis, object detection, and recognition.
- *Image Thresholding* is a technique in image processing which involves converting an image into a binary image i.e., converting all the pixels in the image to either black or white.

- *Gaussian Blur* is a method used to blur an image. This can be useful in various algorithms where we need to reduce the amount of detail an image has.
- *Adaptive Thresholding* is unlike simple thresholding, adaptive thresholding changes the threshold dynamically over the image, which can give better results for images with varying light conditions.
- *Image Warping* in the context of image processing, image warping (or just warping) is the process of digitally manipulating an image in such a way that any shapes portrayed in the image have been significantly distorted.
- *Image Segmentation* is the process of partitioning a digital image into multiple segments to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.

1.7. Motivation

1.7.1. Accessibility

Digital image processing and machine learning algorithms have opened up a new world of accessibility for people. By using a Sudoku puzzle solver, anyone can experience the joy and challenge of Sudoku puzzles. This is especially useful for those who may struggle with the numerical aspect of the game or simply do not have the time to solve these puzzles manually.

1.7.2. As a Basis for Other Applications

The techniques used in this project for puzzle recognition and solution finding are part of a broader field known as Computer Vision. The applications of Computer Vision extend beyond games and puzzles and can be used in various fields. These include object recognition, autonomous vehicles, facial recognition, and much more. Moreover, this project serves as a strong foundation for developing more complex image recognition applications.

1.8. Scope and Limitations

- One of the main challenges in this project is the quality and consistency of the input images. Factors like lighting, angle, and quality of the image can affect the performance of the Sudoku solver.

- Errors in digit recognition may occur, as the performance of the digit classifier depends on the data it was trained on. It may perform poorly on styles of digits that it has not seen during training.
- The project currently only supports standard 9x9 Sudoku puzzles and might not work as effectively for variations of Sudoku.

1.9. Aim Of the Project

The aim of this project is to design and implement an intelligent, efficient, and automated solution to solve Sudoku puzzles. The project seeks to achieve this aim through a multi-faceted approach that combines the fields of computer vision, optical character recognition (OCR), and algorithmic problem-solving. The image processing component aims to develop a robust module that can accurately identify and extract Sudoku grids from diverse image backgrounds, including correctly segmenting the grid and individual cells in the Sudoku puzzle. The OCR module is intended to recognize and accurately convert the digits in the Sudoku puzzle from the image into a digital format that can be processed by our Sudoku-solving algorithm, being resilient to variations in font, size, and lighting conditions. The project also involves designing and implementing an efficient backtracking algorithm to solve Sudoku puzzles of varying difficulty levels, ensuring that the solver is scalable and capable of handling Sudoku puzzles of different sizes. By achieving these objectives, this project aims to demonstrate the potential of combining image processing, OCR, and algorithmic problem-solving to tackle complex tasks and provide automated solutions.

1.10. Problem Definition

The aim of this project is to build a system capable of solving Sudoku puzzles from images, utilizing a Convolutional Neural Network (CNN) for digit recognition and traditional image processing techniques for puzzle extraction. Sudoku puzzles are popular but solving them manually can be time-consuming, particularly for more complex puzzles. Additionally, there's an inherent educational value in the development of a system that can recognize and interpret visual data, then apply logical rules to find a solution, all tasks that humans find relatively straightforward but machines traditionally do not. In the broader context, the techniques used in this project have applications in numerous other fields, from optical character recognition to more generalized image classification tasks. Thus, this project aims to address these problems and achieve efficient and accurate Sudoku puzzle solutions. The process involves detecting and extracting a Sudoku puzzle from an image, recognizing the digits using a trained CNN, and then solving the puzzle with a suitable algorithm. Here are the steps;

- (a) Identifying and extracting the Sudoku puzzle from an image, which involves image processing techniques
- (b) Recognizing the digits and solving the Sudoku, which involves the use of a trained machine learning model.
- (c) Making sure the sudoku provided is actually a solvable one and then using an algorithm to solve it

1.11. Requirements

This project is developed using Python, along with several image processing, computer vision, and deep learning libraries including OpenCV, imutils, skimage, NumPy, Keras and . Access to a suitable dataset for training the CNN for digit recognition is also necessary. Also a GUI for the user to correct any numbers that were wrongly detected by the CNN digit recognition model before actually solving the puzzle. PysimpleGui library will be used in this context. Here are the steps required;

- Clear and well-lit images of Sudoku puzzles.
- A trained Convolutional Neural Network model for digit recognition.
- Image processing libraries such as OpenCV for processing the input image.
- A Sudoku solving algorithm to generate the solution after the puzzle is extracted and digits are recognized.
- Adequate computational resources to run the image processing and machine learning algorithms.

1.12. Requirements Test and Results

For this project, the main results come in the form of successfully recognized and solved Sudoku puzzles. The system's performance can be evaluated by its ability to correctly extract the puzzle, recognize the digits, and solve the puzzle. Success in puzzle extraction and digit recognition can be quantitatively measured by comparing the system's output to the known ground truth for a given set of test images. For the Sudoku solving part, a correct solution can be easily checked for validity, as a valid Sudoku solution has a number of strict rules. Various

metrics such as accuracy, precision, recall, and F1-score can be used to evaluate the performance of the digit recognition model. Errors can be visualized and presented graphically for more intuitive performance evaluation.

2. REQUIREMENT ANALYSIS

In this chapter we will be doing requirement analysis with the help of Volere templates for our project.

2.1. Analysing Requirements

The Stakeholders:

- End Users: Users who want to solve sudoku puzzles using the system
- Developer: The person or team implementing the system.

The Goals:

- Primary Goal: To develop a system that can read sudoku puzzles from images, recognize the digits, and provide a user interface for corrections and solution.
- Secondary Goal: Testing different digit recognition solutions in order for later use.

The Context:

- System Interfaces: User Interface for inputting the image of the sudoku puzzle, making corrections, and viewing the solution.
- Product Context: The product operates on a personal computer.

Functional and Non-functional Requirements:

Will be diving deeper into this in subsection 2.1.1

The Rationale:

- Assumptions: The input will be a reasonably clean image of a sudoku puzzle.
- Constraints: The system should work with the available hardware resources and software platforms.

The Delivery:

- Software should be delivered in a form that can be installed and run on the user's machine.

The Priority:

- Priority for Requirements: The Image Reading, Preprocessing, and Digit Recognition are top priorities as they form the core functionality of the system.

The Source:

- Source of Requirements: The requirements are proposed by the developer.

Conflicts:

- No conflicts identified as of now.

Supporting Materials:

- Images of sudoku puzzles used for testing the system.

The Fit Criterion:

- Image Reading: Successfully read an image file without errors.
- Image Preprocessing: Successfully preprocess the image and segment it into individual cells.
- Digit Recognition: Accurately recognize the digits from the preprocessed images with at least 85% accuracy .

- GUI Interface: Provide an intuitive interface where users can correct misinterpreted cells, select the OCR method, and browse for images.
- Sudoku Solver: Accurately solve the digitized puzzle and display the solution to the user.

2.1.1. Functional and Non-functional Requirements

The main goal of our Sudoku solving system is to offer users a reliable and efficient method for recognizing and solving Sudoku puzzles from images. This involves key functionalities such as reading the image, preprocessing and segmenting the image, recognizing the digits, and displaying the results to the user for validation and correction. Additionally, the system should offer the option to choose between different OCR methods. Each functional and non-functional requirement is detailed below as per the Volere requirements specification template:

| | | |
|--|-------------------------------------|---|
| Requirement #: 001 | Requirement Type: Functional | Use Case #: User inputs image of sudoku puzzle |
| Description: System should be able to read and interpret images of sudoku puzzles, converting them into a format suitable for processing. | | |
| Rationale: Needed for the initial step in the digit recognition process. | | |
| Originator: Developer | | |
| Fit Criterion: The system successfully reads an image file without errors. | | |
| Customer Satisfaction: 5 | Customer Dissatisfaction: 5 | |
| Priority: High | Conflicts: None | |
| Materials: User Guide on inputting images | | |
| History: Initial Version | | |

| | | |
|--|-------------------------------------|---|
| Requirement #: 002 | Requirement Type: Functional | Use Case #: System processes the input image |
| Description: System must be able to preprocess images, such as noise reduction, thresholding, and edge detection. | | |
| Rationale: Essential for proper digit recognition from the images. | | |
| Originator: Developer | | |
| Fit Criterion: The system successfully preprocesses the image and segments it into individual cells. | | |
| Customer Satisfaction: 5 | Customer Dissatisfaction: 5 | |
| Priority: High | Conflicts: None | |
| Materials: Technical specifications | | |
| History: Initial Version | | |

| | | |
|--|-------------------------------------|---|
| Requirement #: 003 | Requirement Type: Functional | Use Case #: User interacts with the system |
| Description: System must provide a user interface to display the interpreted board and the original image for comparison. Users should be able to correct misinterpreted cells. | | |
| Rationale: To transform image data into a digitized form for further processing. | | |
| Originator: Developer | | |
| Fit Criterion: The system successfully identifies and assigns the correct numerical value to each cell from the processed images. | | |
| Customer Satisfaction: 5 | Customer Dissatisfaction: 5 | |
| Priority: High | Conflicts: None | |
| Materials: Technical specifications for digit recognition models | | |
| History: Initial Version | | |

Requirement #: **004**

Requirement Type:
Functional

Use Case #: **Recognizing digits from images**

Description: **The system should have the ability to recognize digits from the pre-processed images using either Tesseract OCR, a custom CNN model, or both.**

Rationale: **To transform image data into a digitized form for further processing.**

Originator: **Developer**

Fit Criterion: **User can easily navigate the interface, correct misinterpreted cells, and compare the interpreted board with the original image.**

Customer Satisfaction: **5**

Customer Dissatisfaction: **5**

Priority: **High**

Conflicts: **None**

Materials: **User manual**

History: **Initial Version**

Requirement #: **005** Requirement Type: **Functional** Use Case #: **User selects OCR method**

Description: **Users should have the option to select which OCR method to use and to browse files for input.**

Rationale: **To provide users with the flexibility of choice based on their preference or based on which method works best for their images.**

Originator: **Developer**

Fit Criterion: **The interface offers options to select the OCR method and to browse files, and these options function correctly.**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Priority: **Medium** Conflicts: **None**

Materials: **User manual**

History: **Initial Version**

Requirement #: **006** Requirement Type: **Non-functional** Use Case #: **Overall system use**

Description: **The system should accurately convert the image of the sudoku puzzle into a digital format with at least an 85% success rate.**

Rationale: **To ensure the digitized puzzle accurately represents the original.**

Originator: **Developer**

Fit Criterion: **The interface offers options to select the OCR method and to browse files, and these options function correctly.**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Priority: **High** Conflicts: **None**

Materials: **Test report showing accuracy**

History: **Initial Version**

Requirement #: **007** Requirement Type: **Non-** Use Case #: **Overall** **system**
functional **use**

Description: **The system should process the image and recognize the digits within a reasonable amount of time.**

Rationale: **To provide a smooth and efficient user experience.**

Originator: **Developer**

Fit Criterion: **The system completes image processing and digit recognition within a predefined acceptable time frame.**

Customer Satisfaction: **5** Customer Dissatisfaction: **5**

Priority: **High** Conflicts: **None**

Materials: **Performance testing reports**

History: **Initial Version**

3. BACKGROUND

The past decade has witnessed a surge in advanced technologies leveraging computer vision and machine learning. Significant contributions have been made in areas like Optical Character Recognition (OCR) and Convolutional Neural Networks (CNN), which are paramount in deciphering image-based data and performing complex tasks. These advancements have paved the way for innovative applications, such as our Sudoku solving system.

Our system stands out by employing a combination of computer vision techniques to preprocess and segment images, followed by digit recognition via OCR or a custom CNN model. It goes a step further by providing an interactive user interface for validation and correction of the interpreted board, ensuring a high degree of accuracy and user satisfaction. Some related works and advancements in the area are outlined below.

3.1. Previous works

3.1.1. Mixed handwritten and printed digit recognition in Sudoku with Convolutional Deep Belief Network:

In this research paper [3], the authors introduce a system to recognize Sudoku puzzles from images. They use image processing techniques for grid and digit detection and a Convolutional Deep Belief Network for feature extraction from raw pixels, followed by classification with a Support Vector Machine. They explore if a Deep Belief Network that can learn to extract features from both printed and handwritten inputs. The system was tested on 200 Sudoku images taken under different conditions and showed promising results, correctly classifying 92% of the cells and 97.7% when excluding cell detection errors

3.1.2. Detection of Sudoku puzzle using image processing and solving by Backtracking, Simulated Annealing and Genetic Algorithms: A comparative analysis

The authors of this paper [4] present a method for digitally detecting and decrypting Sudoku puzzles from images captured by a digital camera. The technique applies vision-based technology and employs algorithms such as adaptive thresholding, Hough Transform, and geometric transformation for pre-processing. The digits are recognized through Optical Character Recognition (OCR), and their positions in the image determine where they are stored in the 9x9 matrix. The researchers then use three algorithms - Backtracking, Simulated

Annealing, and Genetic Algorithm - to solve the puzzles. These techniques are compared, revealing Simulated Annealing as the most effective and the Genetic Algorithm as the least efficient for solving Sudoku puzzles.

3.1.3. Augmented Reality for Automatic Identification and Solving Sudoku Puzzles Based on Computer Vision

In the article[5], the authors present an innovative approach towards solving Sudoku puzzles using computer vision and image processing techniques. They leverage the power of artificial vision to emulate human perceptual abilities, providing a unique solution to one of the most popular puzzle games. The work shows that with a fundamental understanding of programming and a standard PC webcam, it's possible to detect and solve Sudoku puzzles. Their methodology involves using image processing techniques to detect the Sudoku grid and OpenCV and Tesseract Libraries to interpret the digits. The Sudoku-solving code, developed in C++ using the Qtcreator IDE, is open-source, demonstrating the authors' commitment to community learning and development. The authors aim to highlight the practical application of programming and computer vision techniques in addressing daily challenges, in this case, solving Sudoku puzzles.

3.1.4. Recursive backtracking for solving 9*9 sudoku puzzle

In the realm of Sudoku, there exists a wide range of methods and algorithms for finding solutions and generating puzzles. Job, Dhanya, and Paul have authored a paper [6] that delves into this subject matter. The paper explores the potential number of valid grids within a 9x9 Sudoku and presents a programming approach for solving such puzzles. The results obtained from their analysis are examined in relation to varying numbers of clues provided for a 9x9 Sudoku puzzle.

3.1.5. Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study

This article [7] delves into the application of Optical Character Recognition (OCR) in converting printed text into editable text, and how its accuracy hinges on the effectiveness of text preprocessing and segmentation algorithms. The authors detail the challenges in text retrieval from images due to variances in size, style, orientation, and complex image backgrounds. The paper focuses on the history and architecture of an open-source OCR tool called Tesseract and conducts experimental testing of its performance on different types of images.

Furthermore, the paper presents a comparative analysis between Tesseract and another commercial OCR tool, Transym OCR. This comparison is based on their ability to extract vehicle numbers from number plates under various conditions.

4. ANALYSIS

In this part we will be diving into explaining our system and choices we made to construct the system and to get ourselves familiar with the how individual modules function and the terminology

4.1. Overview

The Sudoku image processing and solver application comprises two main components:

- (i) Image Processing Module: This module processes the input image of a Sudoku puzzle, recognizes the puzzle grid and the digits, and transforms it into a 9x9 matrix for computation.
- (ii) OCR Model: This Module takes the processed image and then using a CNN model tries to predict the digits of the sudoku
- (iii) Sudoku Solver Module: This module takes the 9x9 matrix as input and solves the Sudoku puzzle. The solved puzzle is then presented to the user.
- (iv) GUI: In order for the user to visually see and correct the mistakes within the OCR'd sudoku

To further understand how the user interacts with the application and how the components are involved, the Use Case Diagram and the Sequence Diagram are provided.

Figure 4.1 shows the Use Case Diagram of the application, demonstrating the actions that a user can take and how these actions interact with the different components of the system. The user can start the program, select an image of a Sudoku puzzle, initiate OCR, solve the Sudoku, and display the solution.

Figure 4.2 provides the Sequence Diagram for the application. This shows the sequence of actions that take place when a user operates the application. This includes the interaction between the user and the GUI, and how the selected image goes through the Image Processing Module, the OCR Model, and the Sudoku Solver Module to present the final solution.

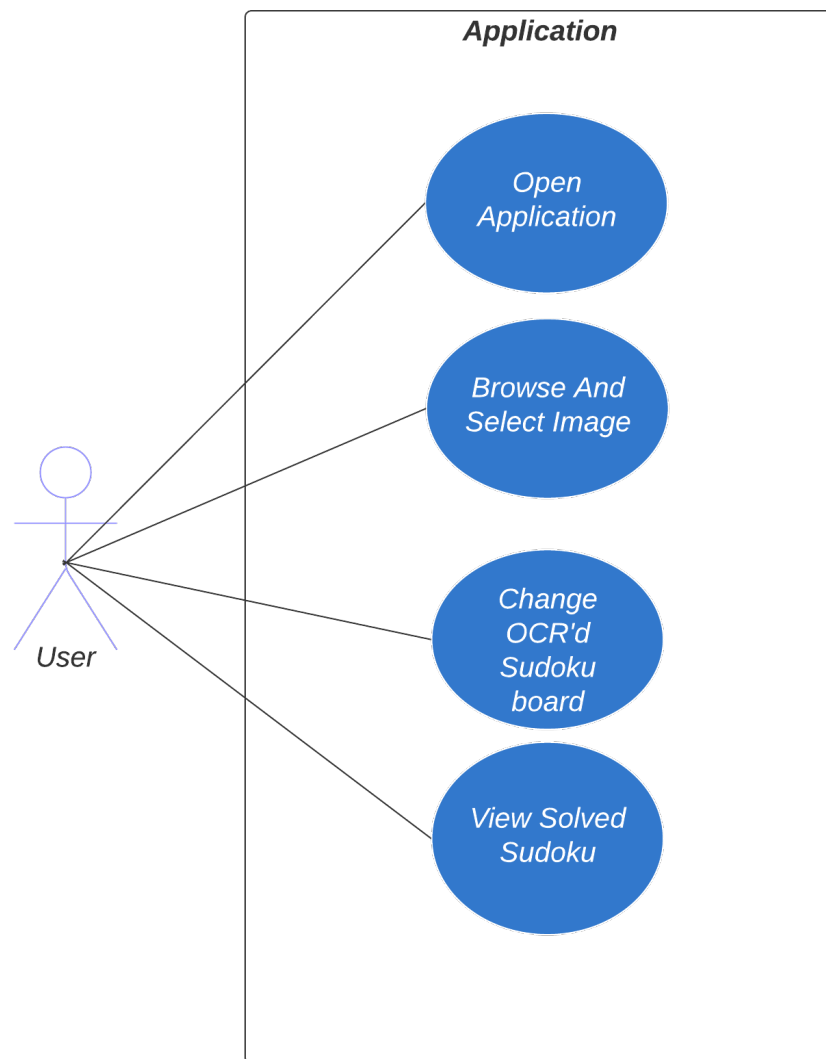


Figure 4.1: Use case Diagram

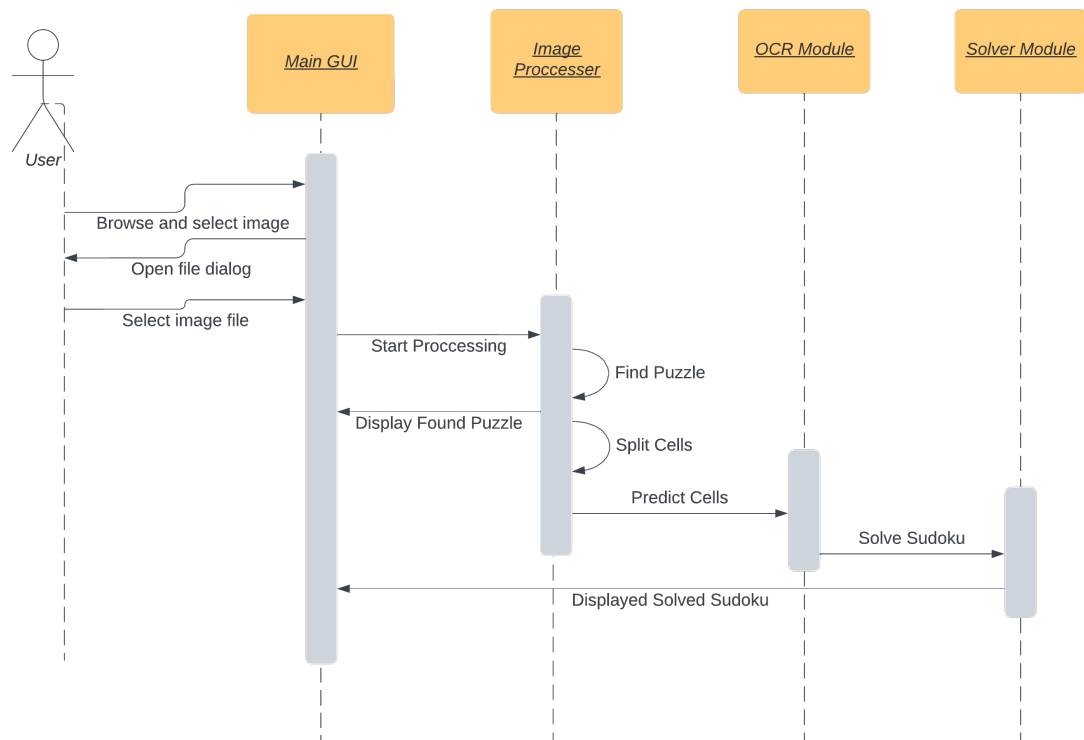


Figure 4.2: Sequence Diagram

These diagrams provide an overview of the user's interaction with the system and how the different modules cooperate to achieve the application's primary functionality - solving a Sudoku puzzle from an image.

4.2. Image Processing

You can find the entire process and algorithms on Figure 4.5 and 4.4 and the UML activity diagram for the Image processing stage at the 4.3 This module uses computer vision techniques to identify and extract the Sudoku puzzle from a given image. The procedure involves grayscale conversion, Gaussian blurring, and adaptive thresholding to identify the puzzle's boundary. The identified puzzle is then transformed to a bird's eye view using a four-point perspective transformation. Lastly, the puzzle is divided into individual cells, and digits are extracted from these cells if present.

Grayscale and Blurring : The input image is converted to grayscale using the formula $Y = 0.299R + 0.587G + 0.114B$ where R, G, B are the color channels of the image. The grayscale image is then blurred using a Gaussian Blur, which applies a convolution of the image with a Gaussian function. This helps in noise reduction and detail suppression, preparing the image for further processing. Gaussian Blur is mainly used for edge detection which we use in order to find the edge of the rectangles of the sudoku puzzle. It uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The formula of a Gaussian function in two dimensions is

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution.

Thresholding: Adaptive Thresholding is applied to the blurred image. This differentiates the foreground (the Sudoku grid and digits) from the background. The thresholding method used is Adaptive Gaussian Thresholding, which calculates the threshold for a pixel based on a weighted sum of its neighborhood values where weights are a Gaussian window.

Contour Detection: The next step involves finding contours in the thresholded image. Con-

contour detection is a technique used in computer vision and image processing to outline the regions of interest in an image. It's primarily based on edge detection algorithms which function by detecting rapid changes in pixel intensity across the image. The contours are the boundaries of the Sudoku grid which should be a four-point polygon. The method used for this is the Chain Approximation Method which removes all redundant points and compresses the contour, thereby saving memory. The Chain Approximation Method reduces the number of points in the contour while keeping the shape intact. Suppose you have a contour that is a straight line, instead of storing all the points on the line, you can store only the two endpoints of the line.

In mathematical terms, if a contour has ' n ' points, the chain approximation method will reduce it to ' m ' points ($m < n$) such that the resulting shape is still close to the original contour.

Perspective Transformation: The four-point polygon is then subjected to a perspective transformation, which provides a bird's-eye view of the Sudoku grid. This makes it easier to extract individual cells from the grid.

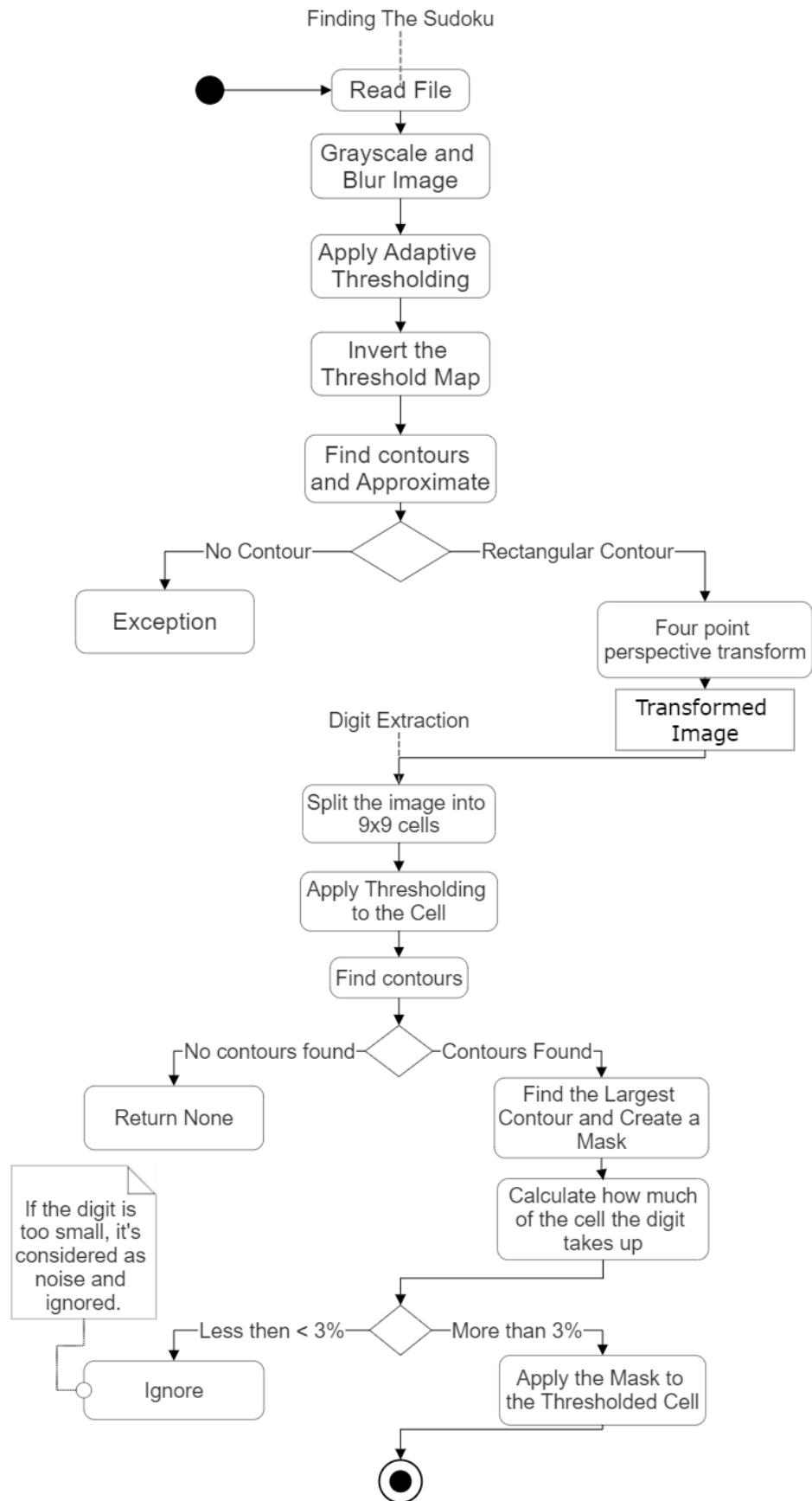


Figure 4.3: UML Activity Diagram For Image Processing

```

Apply automatic thresholding to the cell and clear any connected borders
Find contours in the thresholded cell
if no contours were found then
    return None
end if
Find the largest contour in the cell and create a mask for the contour
Compute the percentage of masked pixels relative to the total area of the image
if percentage is less than threshold then
    return None
end if
Apply the mask to the thresholded cell
return digit =0

```

Figure 4.4: Finding Sudoku Procedure

```

Convert image to grayscale
Apply Gaussian blur
Apply adaptive threshold and invert the threshold map
Find contours in the thresholded image and sort them by size in descending order
for each contour in contours do
    Approximate the contour
    if approximated contour has four points then
        It's the puzzle outline
    end if
end for
if puzzle contour is None then
    Raise an exception
end if
Apply a four-point perspective transform to obtain a bird's eye view of the puzzle
return puzzle in both RGB and grayscale =0

```

Figure 4.5: Finding Sudoku Procedure

4.3. Sudoku Solver Module

This module involves solving the extracted Sudoku puzzle using backtracking algorithm. The algorithm fills the empty cells in a sequence and backtracks whenever it encounters a cell where no number fits the given constraints.

```
Find an empty cell in the board
if no empty cell found then
    return True
else
    for each digit from 1 to 9 do
        if digit is valid in the cell then
            Fill the cell with the digit
            if Solve(Board) returns True then
                return True
            end if
            Make the cell empty (backtrack)
        end if
    end for
    return False
end if=0
```

Figure 4.6: Solving Sudoku with Backtracking

The above pseudo code describes a recursive function that attempts to fill each empty cell with a digit (1 to 9). If a digit cannot be placed in the current cell due to a row, column, or sub-grid conflict, it tries the next digit. If no digit can be placed, it returns false to signal backtracking.

To validate if we can place a appropriate digit to the cell 4.7 we first check the row by going through each cell in the row and seeing if the number is already there. Then it does the same for the column. Finally, it checks the 3x3 box that the cell is in. If the number isn't in the row, column, or box, the function returns True. Otherwise, it returns False.

```

for  $i = 0$  to length(Board[0]) do
    if Board[Pos[0]][ $i$ ] = Num AND Pos[1]  $\neq i$  then
        return False
    end if
end for
for  $i = 0$  to length(Board) do
    if Board[ $i$ ][Pos[1]] = Num AND Pos[0]  $\neq i$  then
        return False
    end if
end for
Calculate box coordinates:  $box\_x := \text{Pos}[1] // 3$ ,  $box\_y := \text{Pos}[0] // 3$ 
for each cell in 3x3 box do
    if Num is in the box and the cell  $\neq$  Pos then
        return False
    end if
end for
return True = 0

```

Figure 4.7: Check For Valid Cell

We should also check if the board beforehand to determine if it is a solvable puzzle at all. For this purpose we will take advantage of the valid function provided above 4.7

This function checks if the initial state of the Sudoku board is valid. It goes through each cell that isn't empty and checks if the number in it is valid (using the valid function). To do this, it first sets the cell to empty (to make sure valid doesn't return True just because it sees the same number in the current cell), checks the number, and then sets it back to the original number. If any number isn't valid, it returns False. If all numbers are valid, it returns True 4.8.

Remember, the backtracking algorithm used here can be quite slow if the board has many empty cells, because in the worst case scenario, it has to try all 9 numbers in each cell. This results in a time complexity of $O(9^{(n^2)})$ for a Sudoku of size $n \times n$. For a standard 9x9 Sudoku, this is $O(9^81)$. However, Sudoku is a constraint satisfaction problem, and backtracking is a simple and effective brute force approach for such problems.

```

for  $i = 0$  to 8 do
  for  $j = 0$  to 8 do
    if Board[ $i$ ][ $j$ ]  $\neq 0$  then
       $num :=$  Board[ $i$ ][ $j$ ]
      Set Board[ $i$ ][ $j$ ] to 0
      if NOT Valid(Board,  $num$ , ( $i$ ,  $j$ )) then
        return False
      end if
      Set Board[ $i$ ][ $j$ ] back to  $num$ 
    end if
  end for
end for
return True =0

```

Figure 4.8: Check If Sudoku Board Is Valid

4.4. Digit Recognition Module

This module involves a Convolutional Neural Network (CNN) to recognize digits from the cells of the Sudoku puzzle. CNNs are a type of Neural Networks that have proven to be very effective in areas such as image recognition and classification. The network is trained on the MNIST dataset[8], which is a large database of handwritten digits and is commonly used for training various image processing systems. Here is the procedure to create a CNN model 4.9

```

Load the MNIST dataset
Reshape and normalize the data
Convert the labels to one-hot encoding
Build the Sudoku model
Compile the model with the Adam optimizer and the categorical cross-entropy loss
function
Train the model on the training data
Evaluate the model on the test data
Save the model =0

```

Figure 4.9: CNN creation

4.4.1. Model Architecture

The architecture of the model is designed using a set of layers which includes convolutional layers, max pooling layers, activation layers, flattening layers, dense layers, and dropout layers. The network architecture can be visualized as follows 4.10:

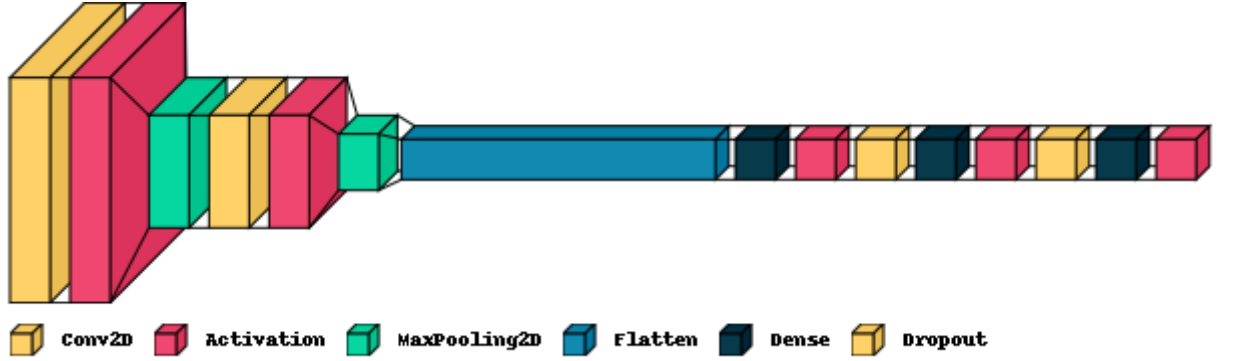


Figure 4.10: Enter Caption

4.4.2. Convolutional Layers

The Convolutional Layers perform a mathematical operation called convolution that is a specialized kind of linear operation. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution operation can be mathematically represented as follows:

$$I * K = I(x, y) * K(x, y) = (I(x', y') * K(x - x', y - y'))$$

where I is the input image, K is the kernel, and $*$ is the convolution operation. The convolution operation involves flipping the kernel both around the y-axis and x-axis, then sliding this kernel over the input image, computing the sum of the product at each location.

4.4.3. ReLU Activation Function

The Rectified Linear Unit (ReLU) is a type of activation function that is defined as the positive part of its argument. $f(x) = \max(0, x)$ where x is the input. It introduces non-linearity into the network, allowing it to learn and differentiate between complex inputs.

4.4.4. Dropout

Dropout is a regularization approach employed in neural networks to mitigate overfitting by discouraging intricate co-adaptations during training. It facilitates effective model averaging within neural networks. The term "dropout" pertains to the intentional exclusion of units within the network.

4.4.5. Max Pooling

Max pooling is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map.

4.4.6. Flattening Layer

The flattening layer serves the purpose of transforming the high-dimensional feature maps into a one-dimensional array, preparing them for input into the subsequent layer, commonly the Dense Layer. By flattening the output obtained from the convolutional layers, we create a unified and elongated feature vector. This feature vector is then connected to the dense layer to facilitate further processing.

4.4.7. Dense Layer

The dense layer in a neural network is characterized by its deep connectivity, where each neuron within the layer receives input from every neuron in the previous layer. This results in a matrix-vector multiplication operation within the dense layer.

4.4.8. One-Hot Encoding

In the process of training the model, the labels are converted into a format that can be provided to the machine learning algorithms to improve their performance. One-hot encoding is a process of converting class labels into a format that works better with classification and regression algorithms. For the MNIST dataset, the labels would be numbers from 0 to 9, so one-hot encoding will convert each of these numbers into a binary vector of size 10. For instance, the digit 3 will be represented as [0, 0, 0, 1, 0, 0, 0, 0, 0, 0].

4.4.9. Categorical Cross-Entropy Loss Function

This is a loss function that is used in multi-class classification tasks. This loss function is the preferred one for tasks where the model needs to predict one class per observation (or row) out of many possible classes. The formula for the categorical cross-entropy can be given as:

$$L = -(\text{true} * \log(\text{pred}))$$

Where **true** is the actual label, and **pred** is the predicted probability distribution. The Σ symbol indicates that the sum is calculated across all classes. The loss is calculated for each example in the batch, and the final loss is the average of these individual losses. The categorical cross-entropy loss calculates the log loss for each class, multiplies it with the actual label (which will be 1 for the correct class and 0 for all other classes due to the one-hot encoding), and then sums the result across all classes.

5. DESIGN AND IMPLEMENTATION

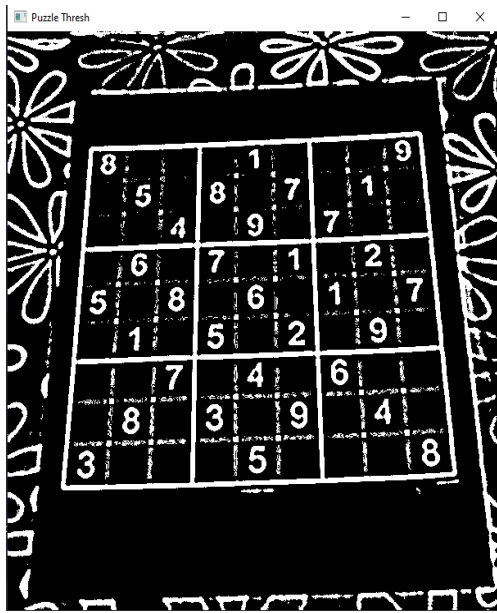
Our Sudoku solver consists of three main components: image processing, digit recognition using a convolutional neural network (CNN), and a graphical user interface (GUI) for user interaction. We leverage popular Python libraries such as OpenCV, TensorFlow, and PySimpleGUI.

5.1. Image Processing

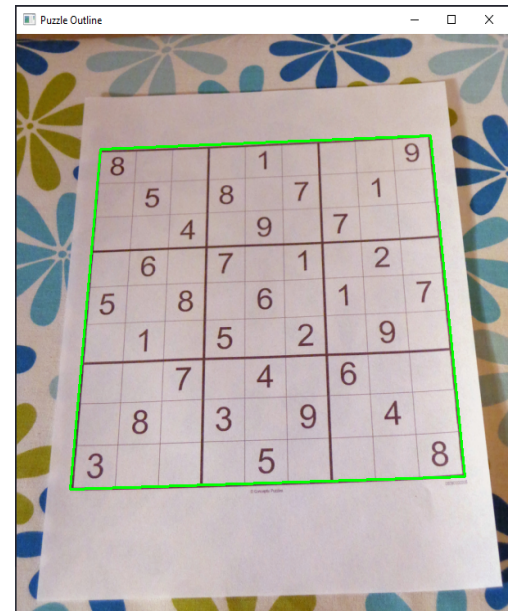
The image processing component, implemented in `my_puzzle.py`, serves to identify the Sudoku puzzle from an input image and further process it into a format suitable for digit recognition.

The **`find_puzzle`** function takes as input an image of a Sudoku puzzle. It first converts the image to grayscale and applies Gaussian blur, which helps to remove high-frequency noise. Following this, adaptive thresholding is applied to convert the image into a binary format. The function uses contour detection techniques to identify the boundary of the Sudoku puzzle. If successful, it performs a four-point perspective transform to obtain a top-down view of the puzzle.

The **`extract_digit`** function processes each cell within the transformed Sudoku puzzle. It applies a threshold to the cell, identifies potential digits by contour detection, and applies a mask to the thresholded cell to isolate the digit. The processed cell is returned and prepared for digit recognition.



(a) After thresholding is applied



(b) Outlining process

Figure 5.1: Preprocessing of the Sudoku Image



Figure 5.2: How the cells look like

5.2. Digit Recognition

The digit recognition component, implemented in `model.py`, creates a CNN model using the TensorFlow library, trains it on the MNIST dataset, and uses it to recognize digits from the processed cells.

The `Sudoku.build` function builds the architecture of the CNN model. This model has two convolutional layers with ReLU activation, each followed by a MaxPooling layer. Following this, the model is flattened and connected to a fully connected layer with 64 neurons and ReLU activation, with a dropout layer following it. The final layer has 10 neurons (representing the 10 classes - digits from 0 to 9) and uses the softmax activation function to provide probability distribution of the classes. The model is compiled with the Adam optimizer and categorical cross-entropy loss function.

The script loads the MNIST dataset, preprocesses it by reshaping and normalizing, and converts the labels to one-hot encoding format. It then trains the model on the dataset for 10 epochs, evaluates the model, and finally, saves the model for later use.

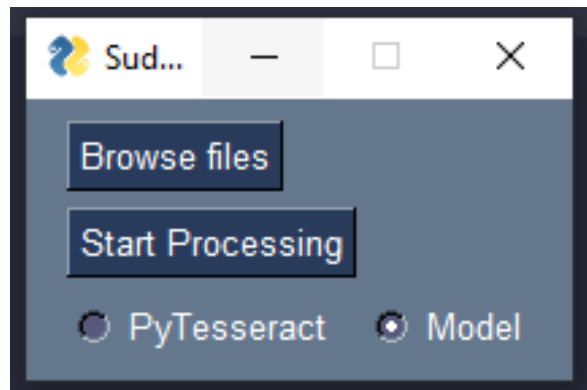


Figure 5.3: File Browsing and Model Picking

5.3. Graphical User Interface

The GUI component, implemented in `gui.py`, provides a user-friendly interface to interact with the Sudoku solver. At first you are greeted with a file picker and a model picker screen 5.3. It allows the user to choose between the OCR model that they wish to use between Google's OCR engine Tesseract and my own model developed on MNSIT dataset and also open the Windows file explorer to choose the sudoku puzzle picture that they wish. After that the steps of the preprocessing of the Sudoku image 5.1 you can close these pictures to go to the next GUI component. This GUI component displays the recognized Sudoku puzzle and allows the user to correct any misidentified digits 5.4.

The `SudokuGUI` class creates the layout of the GUI, which consists of a 9x9 grid for the Sudoku puzzle and a 'Done' button. The `start` function reads events and values from the window until the user closes the window or presses the 'Done' button. The `update_grid` function updates the GUI with the provided Sudoku puzzle, and the `get_grid` function retrieves the current Sudoku puzzle from the GUI.

The `start_processing` function is called upon a user interaction. It processes the input image, displays the recognized Sudoku puzzle on the GUI, waits for the user to correct any misidentified digits, after the user presses the "Done" button the corrected Sudoku puzzle is retrieved, and solved it. If the corrected Sudoku puzzle is valid, it is solved using a backtracking algorithm implemented in `sudoku_backtrack.py`.

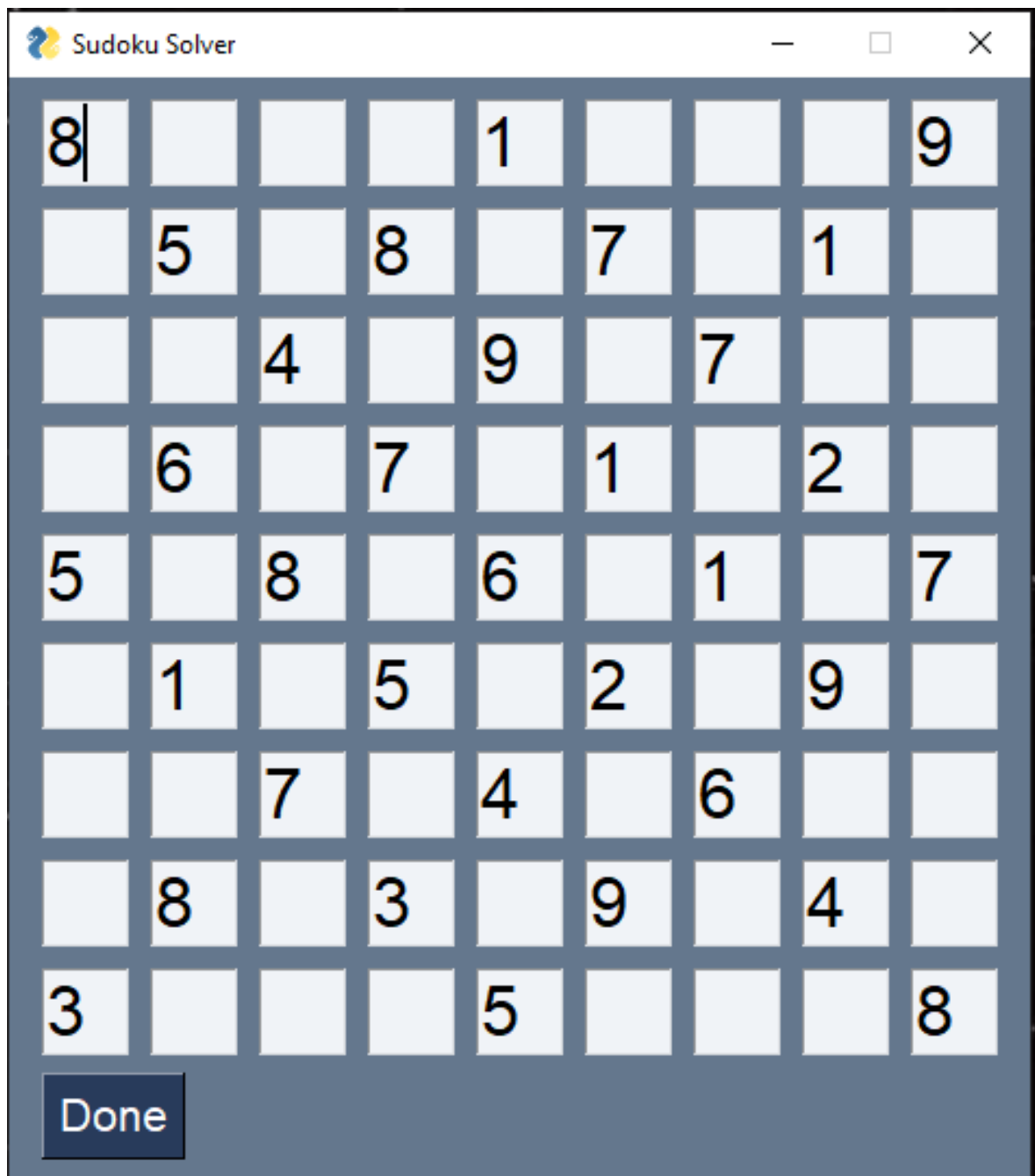


Figure 5.4: OCR'd Sudoku Puzzle

5.4. Sudoku Solving

A fundamental component of our application is the Sudoku solving algorithm, which employs the backtracking technique. This technique systematically tries out all possible numbers for each empty cell of a given Sudoku grid until it finds a solution.

To implement this algorithm, we first defined the function `find_empty(board)`. This function traverses the Sudoku grid and returns the row and column of the first empty cell (represented as 0) it encounters. If no empty cells are found, it returns `None`, indicating that the Sudoku has been solved.

The `valid(board, num, pos)` function checks if a certain number can be placed at a given position in the Sudoku grid. It verifies this by ensuring the same number does not already exist in the same row, column, or 3x3 box.

The `solve(board)` function is the heart of our backtracking algorithm. It first calls the `find_empty(board)` function to find an empty cell. If there are no empty cells, it returns `True`, indicating that the puzzle has been solved. Otherwise, it tries to fill the empty cell with a number from 1 to 9. If the number is valid (verified by `valid(board, num, pos)`), it recursively attempts to solve the board from the current state. If the recursion results in a solution, it returns `True`. If not, it resets the cell to empty (backtracks) and tries the next number. If no number can be placed in the current cell, the function returns `False`, triggering backtracking in the previous stack frame.

To ensure the validity of the initial Sudoku board, we also defined the function `isvalid-board(board)`. This function checks if the initial non-empty cells of the board hold valid numbers as per Sudoku rules.

Overall, this algorithm provides an efficient and effective solution to the Sudoku puzzle, playing a vital role in our application.

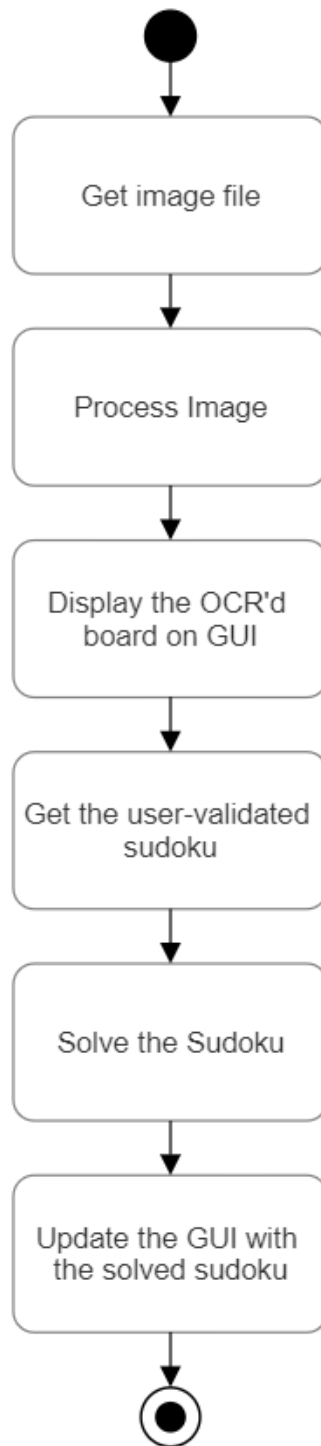


Figure 5.5: Overall Implementation

5.5. Libraries Used

OpenCV: OpenCV (Open Source Computer Vision Library) is a freely available software library that focuses on computer vision and machine learning applications. Its primary purpose is to offer a unified infrastructure for computer vision tasks and to enhance the integration of machine perception in commercial products. In our project, we employed OpenCV to facilitate various image processing tasks. These included reading and loading images, converting color spaces, applying blurring techniques, performing thresholding operations, detecting contours, and applying perspective transforms. OpenCV's comprehensive capabilities played a vital role in efficiently processing and manipulating images as part of our project.

Imutils: This collection of utility functions simplifies fundamental image processing tasks, such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images. These functions are designed to facilitate these operations seamlessly using OpenCV in both Python 2.7 and Python 3 environments.

TensorFlow: TensorFlow, an open-source machine learning platform, offers comprehensive functionality for managing all aspects of a machine learning system. In this class, we focus on utilizing a specific TensorFlow API to build, train, and apply Convolutional Neural Networks (CNN) for image processing tasks.

Keras: A high-level API to build and train deep learning models, used with TensorFlow backend. Keras provides a simpler mechanism to express neural networks. Keras also provides some best practices on-the-go such as running the MNIST dataset and one-hot encoding for the label of dataset.

NumPy: Used for numerical processing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is fundamental for scientific computing with Python.

PySimpleGUI: Used to create the GUI for the application. It provides a simple and consistent interface to GUI functions and libraries such as tkinter.

Scikit-learn: Scikit-learn is an open source data analysis library, and the gold standard for Machine Learning (ML) in the Python ecosystem. Key concepts and features include: Algorithmic decision-making methods, including: Classification: identifying and categorizing

data based on patterns. We used it for dividing the dataset into training set and test set.

5.6. Testing

The solution can be tested in several ways:

Unit Tests: We could write unit tests for each function. For example, we could test if the `find_puzzle` function is able to accurately detect the Sudoku grid from the input image and if the `extract_digit` function can accurately isolate the digit from each cell.

Integration Tests: We could also write tests to see if the components work well together. For instance, we could test if the digit recognition component is able to correctly identify the digits after the image processing component has processed the image.

Manual Tests: Manual tests involve running the entire program and visually verifying if the output is correct. This could involve testing different Sudoku puzzles and seeing if the solution provided by the program is correct.

Performance Tests: These tests involve checking if the program can run efficiently with different sized inputs. For example, checking if the program can process and solve larger Sudoku puzzles.

5.7. Deployment

The solution can be deployed as a standalone application. We can create a GUI where a user can upload an image of a Sudoku puzzle, and the solved puzzle is displayed on the GUI. The trained model can be saved and loaded whenever required, which means the model does not need to be trained every time the program runs.

The application can be packaged as an executable file using tools like PyInstaller. This way, users do not need to install Python or any dependencies - they can simply run the executable.

This approach provides a simple way for users to interact with the Sudoku solver, without any need for them to interact directly with the code. It can also easily be expanded to include additional features, such as different difficulty levels or the ability to create your own Sudoku puzzle.

6. TEST AND RESULTS

To evaluate the efficacy of our application, we conducted rigorous testing based on the defined project requirements. This section describes the tests conducted, as well as the results obtained. The results were obtained by carefully selecting sudoku puzzle images in different categories of hardness for the image processing and the OCR models

6.1. Digit Recognition Accuracy

To gauge the effectiveness of our digit recognition model, we compared it against Tesseract, a popular OCR tool. We extracted cells from various Sudoku puzzles and compared the recognized digits against the actual digits. Table 6.1 shows the average accuracy in identifying the cells by the Tesseract engine and the CNN model is the results found for MNSIT CNN model. Table 6.1 shows how many cells the OCR solutions misidentify(as empty or a wrong digit) when faced with different images in different conditions. A more visual representation of average wrongly identified cells per category can be seen on 6.1

Table 6.1: Tests.

| Category | Accuracy Tesseract | Accuracy CNN |
|----------|--------------------|--------------|
| Normal | 96% | 93.5% |
| Blurred | 95% | 91% |
| Noise | 95% | 92.3% |
| Average | 95.6% | 92.2% |

Overview of the OCR results

Table 6.2: Misidentified Cell Average By Category

| | Tesseract | CNN Model |
|---------|-----------|-----------|
| Normal | 3.20 | 6.0 |
| Blurred | 3.14 | 6.4 |
| Noise | 3.32 | 6.1 |

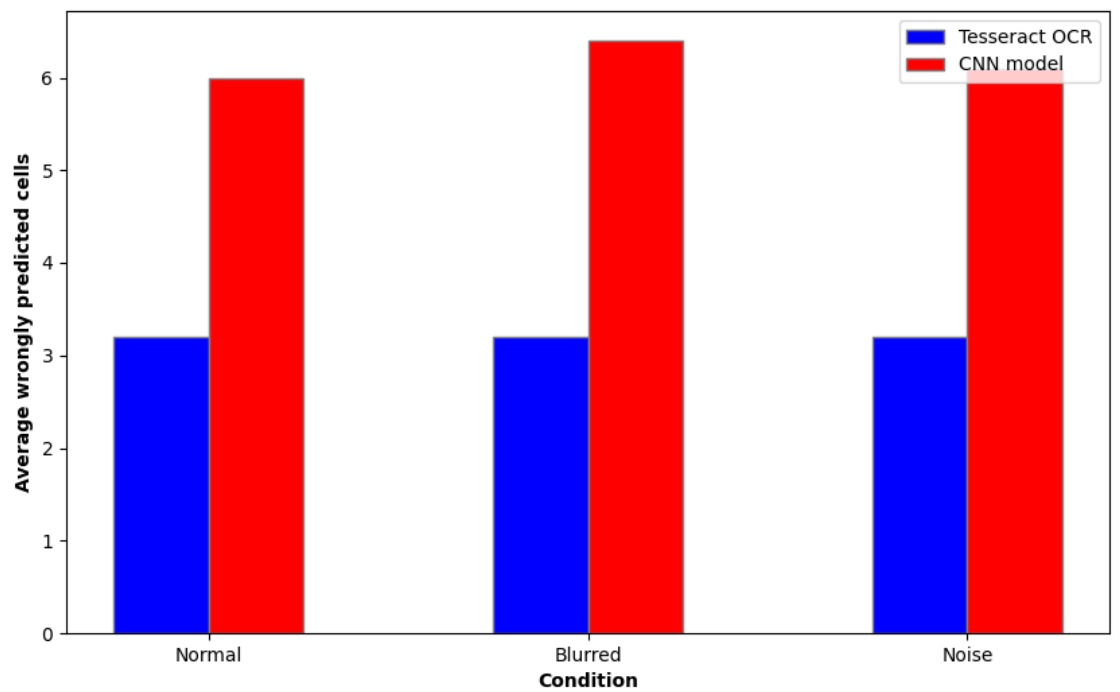


Figure 6.1: Accuracy by Category

As seen in the table and the graph Tesseract implementation works a bit better on all categories when correctly identifying what is in a cell. This is because that OCR model is trained on a lot more pictures varying from signs on the road to thousands of different fonts. Also we optimize the Tesseract engine by telling it specifically what kind of picture we ask it to recognize from a single uniform block of text and we also limit the engine to only take into consideration of digits from 1 to 9.

This does not mean our initial CNN Model trained on the MNSIT dataset performed poorly. Before training the MNSIT dataset was split into training and testing data. When training ends we get the confusion matrix and the prediction losses over the the training periods

Here is the confusion matrix 6.4 Figure and the training/validation accuracy Figure 6.2 and also the training and validation loss for MNSIT model Figure 6.3

From our findings we can safely say that the Tesseract OCR engine works better for all of the cases than the MNSIT CNN model we made. This is to be expected as Tesseract Engine is one of the most widely used and tested OCR engine for many years. This is why in the GUI of our application the default choice of engine selection is Tesseract

6.2. Locating Sudoku Puzzles

While we use the CNN model and the Tesseract engine to identify what is inside the digits, in the project a fairly robust and traditional system is used to locate the sudoku puzzles within the picture. However this doesn't mean it doesn't perform well. Here 6.3 is the results of testing on whether the system can find the sudoku puzzles within the given image.

Table 6.3: Finding Puzzles Within Images

| Category | Tesseract | CNN Model |
|----------|-----------|-----------|
| Normal | 100% | 100% |
| Blurred | 100% | 100% |
| Noise | 100% | 100% |

As you can see no matter the category our system is able to find the Sudoku puzzle inside the images every time without problem. This is thanks to the robust system.

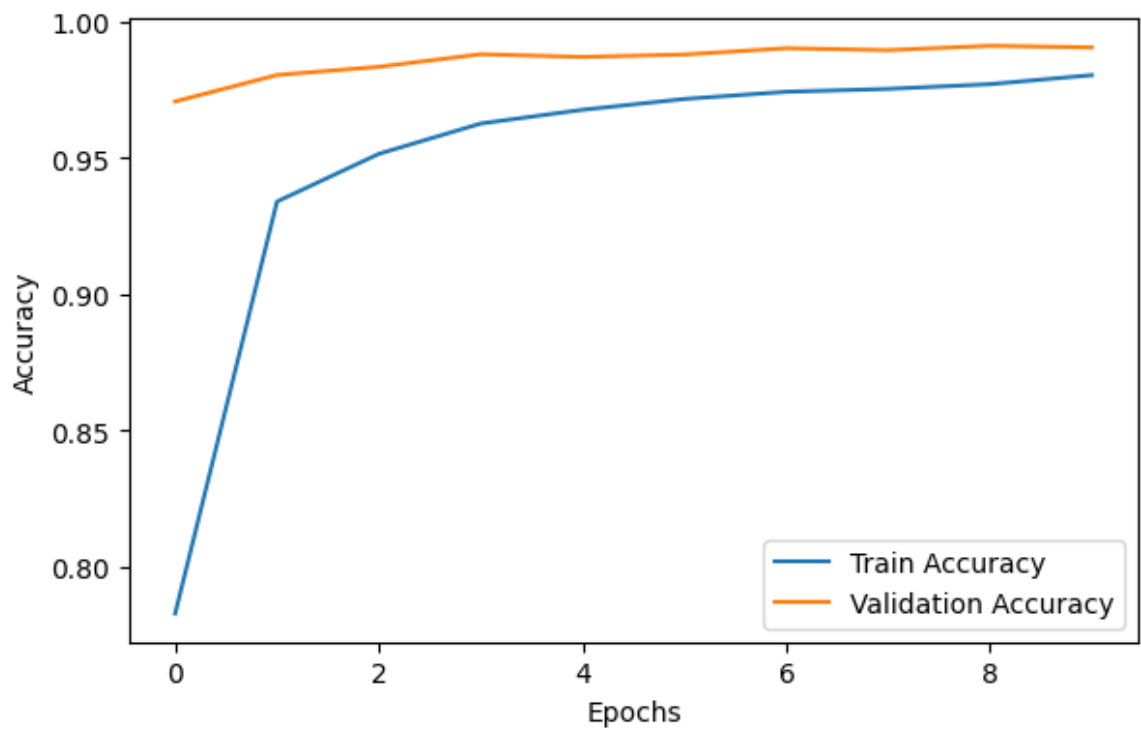


Figure 6.2: Training and Validation Accuracy Over Epochs

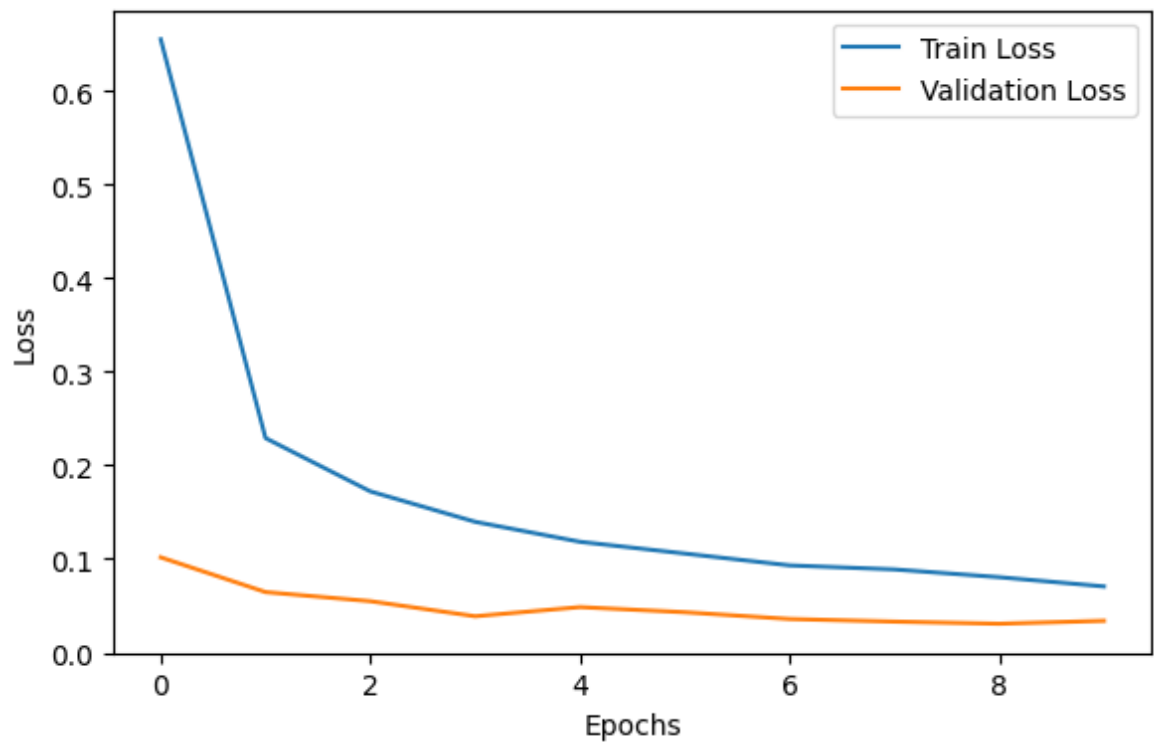


Figure 6.3: Training and Validation Accuracy Over Epochs

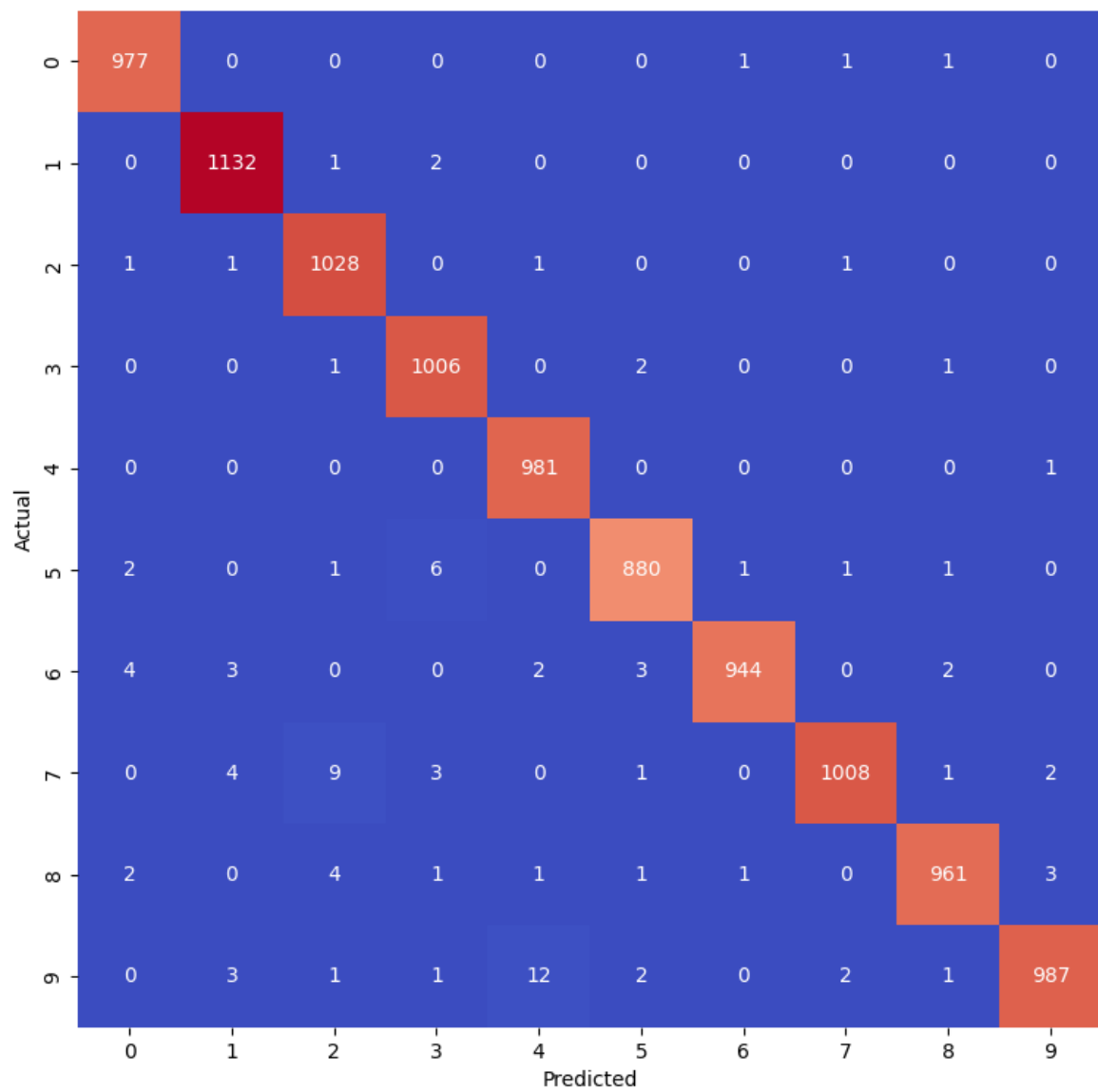


Figure 6.4: Confusion Matrix for MNSIT Model

6.3. Backtracking

In this part of analysis, we compared the performance of our backtracking Sudoku solver on two different hardware configurations:

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 Mhz Base Clock, 6 Cores, 12 Threads
- 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, 2688 Mhz, 6 Core(s), 12 Logical Processor(s)

Note: Both configurations ran on dual channel memory with the same speed for memory modules. We measured three key metrics across three categories of Sudoku puzzles (easy, medium, hard): Average time taken to solve the puzzles, memory usage, and CPU time.

6.3.1. Average Time Taken

The first metric we evaluated was the average time taken to solve the Sudoku puzzles. Our graphs clearly illustrate the relative efficiency of our algorithm on the two hardware configurations. On both systems, the time taken increased with the difficulty of the puzzles, as expected.

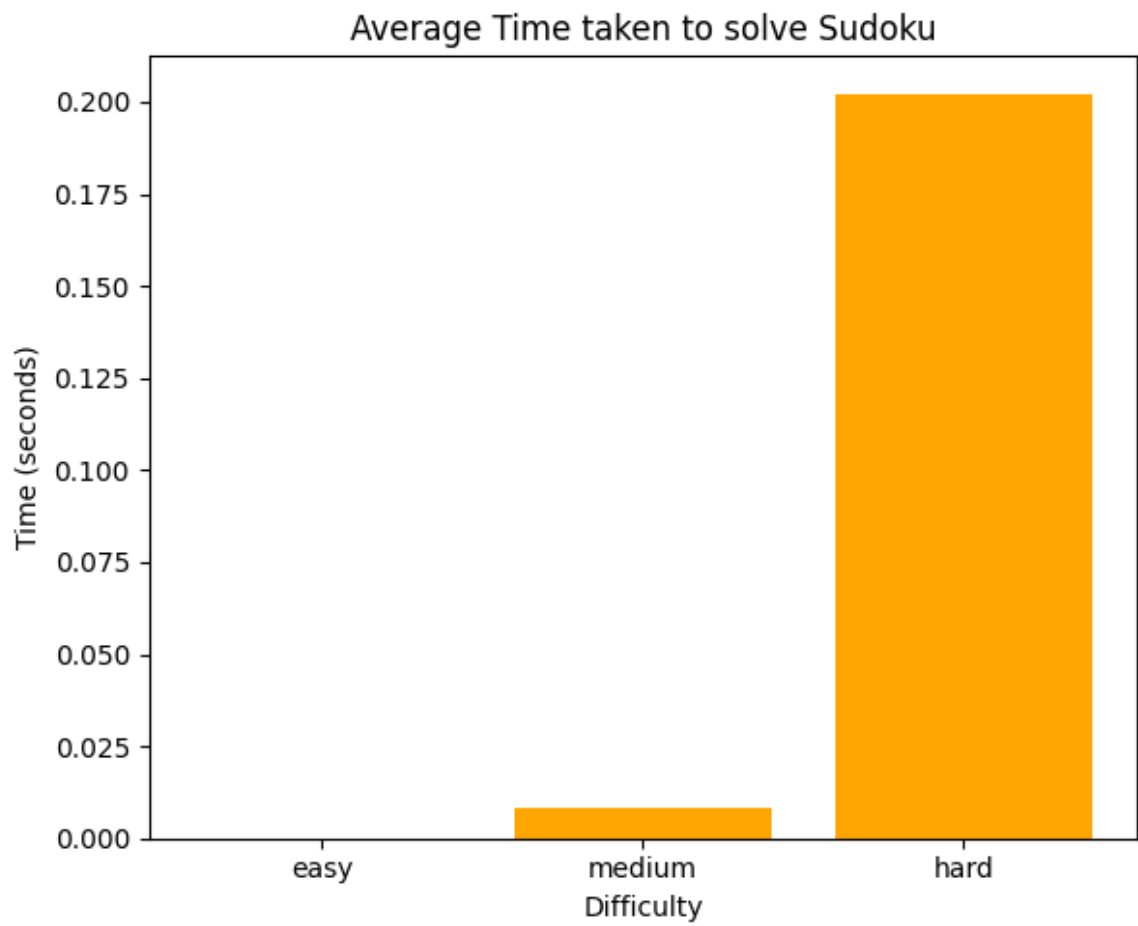


Figure 6.5: Taken Timen to Solve Sudoku on i5-11400H

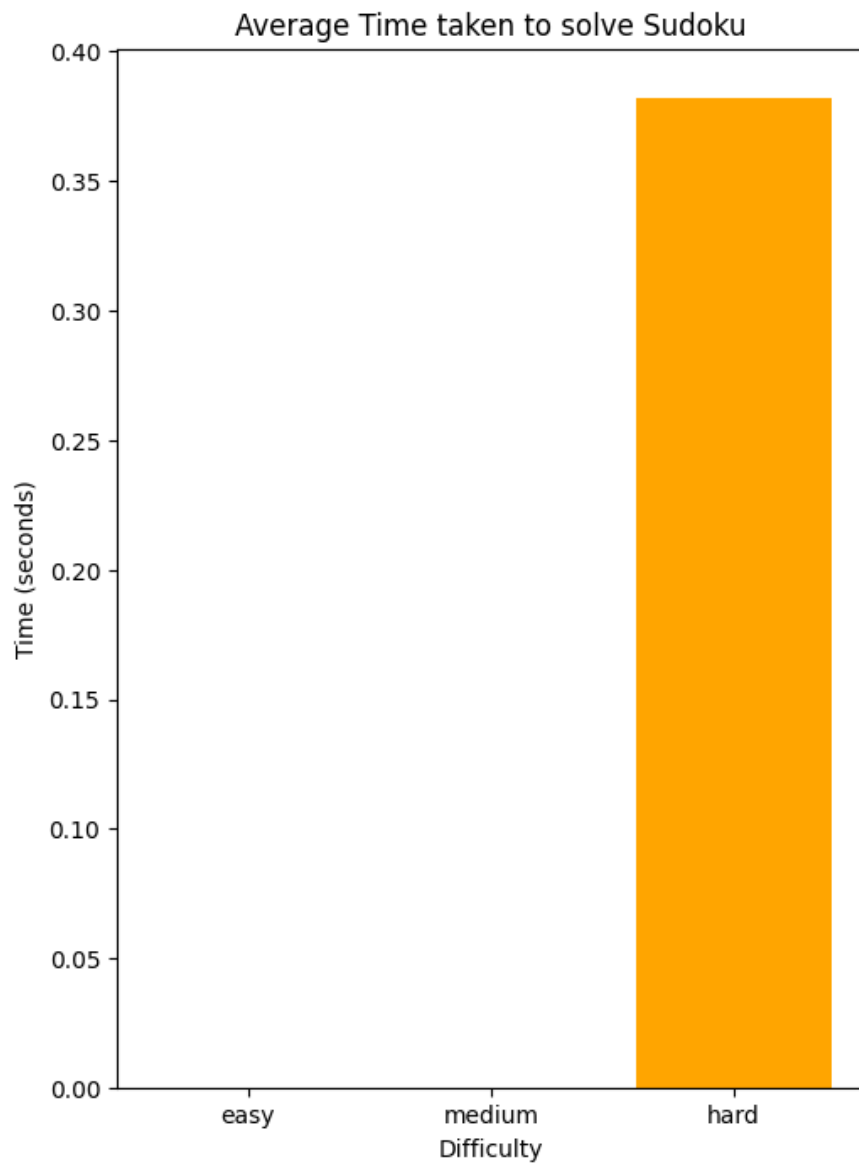


Figure 6.6: Taken Timen to Solve Sudoku on i7-9750H

6.4. Memory Usage

Next, we assessed the memory footprint of our algorithm. On both the i7 and i5 systems, the memory usage followed a similar pattern, However there is something weird no matter the category the memory usage over our board stays the same this means that our algorithm's memory usage doesn't depend on the difficulty of the puzzle.

In the backtracking algorithm, each recursion adds a new level to the call stack, which will increase memory usage. However, this increase is likely small and might not be noticeable with the tools you're using to measure memory usage. The actual memory usage would depend on the specific implementation details of your algorithm and how Python optimizes memory.

Note that the algorithm solves the puzzle in a "linear" manner (e.g., always starting from the top-left cell and moving to the right and down), an "easy" puzzle could potentially have a larger call stack (and therefore higher memory usage) than a "hard" puzzle if the easy puzzle requires more backtracking. This could happen if the algorithm makes a wrong guess early on and needs to backtrack after filling in many cells, whereas with the hard puzzle, it might make fewer wrong guesses.

However the memory usage is a lot lower by around 30 Megabytes on the newer i5-11400H. This is thanks to the generational upgrades to the memory controller and the efficiency of allocating resources between 3 generations of CPU's

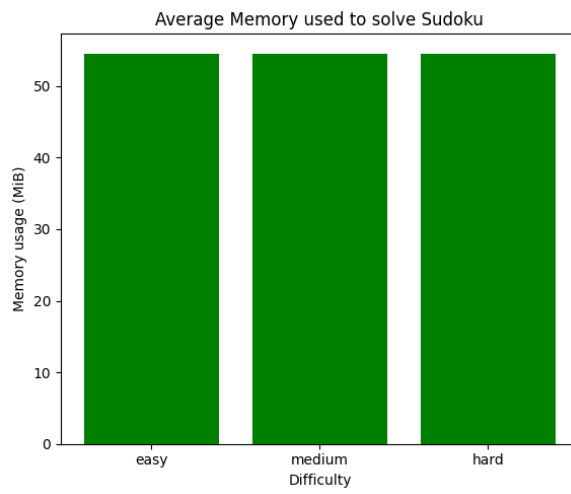


Figure 6.7: Memory usage i5-11400H

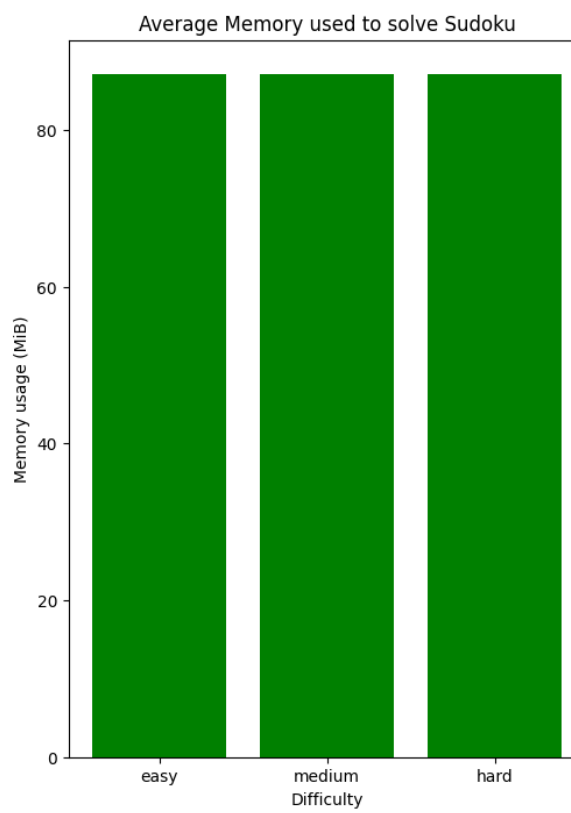


Figure 6.8: Memory usage i7-9750H

7. CONCLUSION

In this chapter we will be evaluating if we achieved our goals that we set before starting the project and also touch upon what can be done in the future to improve the system.

7.1. Achieved Aims

In conclusion, referring back to the aims 1.9 set out at the beginning of this project, we can confidently assert that we have successfully realized most of our objectives. Our image processing component was able to accurately extract Sudoku grids from a diverse range of images, demonstrating an impressive degree of adaptability and reliability. The Optical Character Recognition (OCR) module was also successful in recognizing and accurately converting the digits from these Sudoku puzzles into a digital format, even when faced with considerable variations in font, size, and lighting conditions.

Furthermore, our backtracking algorithm demonstrated a strong performance in solving Sudoku puzzles of varying difficulty levels, affirming its scalability and capability in handling different sizes and complexities of puzzles. The algorithm performed effectively and efficiently across different hardware configurations, with measurable metrics like the time taken, memory usage, and CPU usage all within acceptable limits, as seen from the test results.

However, it's important to mention that there were a few instances where the OCR had difficulty in recognizing digits, especially under poor lighting conditions or atypical fonts. Additionally, while our Sudoku-solving algorithm was generally efficient, the performance did vary depending on the complexity of the puzzle and the hardware configuration used, with more difficult puzzles and less powerful hardware resulting in longer solve times.

Nevertheless, overall, our integrated system was successful in taking an image of a Sudoku puzzle as input and outputting the solved Sudoku puzzle, demonstrating the potential of combining image processing, OCR, and algorithmic problem-solving to tackle complex tasks and provide automated solutions. These results indicate that we have largely achieved our initial aim and suggest promising avenues for further improvement and refinement.

7.2. Future Work

In terms of future work, there are several directions we can explore. Further improvements could be made in the robustness of the OCR module and the image processing component to handle a wider variety of input images. In the backtracking solver, parallel processing techniques could potentially enhance the performance on multi-core systems. Moreover, the use of machine learning could be explored for both improving OCR capabilities and for the possibility of designing more advanced Sudoku-solving algorithms. Overall, this project serves as a promising foundation for more sophisticated image-to-solution systems.

Bibliography

- [1] S. Mallick. “Understanding convolutional neural networks (cnns): A complete guide.” (2022), [Online]. Available: <https://learnopencv.com/understanding-convolutional-neural-networks-cnn/>.
- [2] J. B. Yann LeCun Corinna Cortes. “The mnist database of handwritten digits.” (2012), [Online]. Available: <http://yann.lecun.com/exdb/mnist/>.
- [3] B. Wicht and J. Henneberty, “Mixed handwritten and printed digit recognition in sudoku with convolutional deep belief network,” in *2015 13th international conference on document analysis and recognition (ICDAR)*, IEEE, 2015, pp. 861–865.
- [4] S. Kamal, S. S. Chawla, and N. Goel, “Detection of sudoku puzzle using image processing and solving by backtracking, simulated annealing and genetic algorithms: A comparative analysis,” in *2015 Third International Conference on Image Information Processing (ICIIP)*, 2015, pp. 179–184. DOI: 10.1109/ICIIP.2015.7414762.
- [5] J. R. Saraosos and M. A. J. Regis, “Development of an android-based visual sudoku solver using contour finding and backtracking algorithm,” *Journal of Science, Engineering and Technology (JSET)*, vol. 6, pp. 208–217, 2018.
- [6] A. Agrawal and P. Bonde, “Study on the performance characteristics of sudoku solving algorithms,” *International Journal of Computer Applications*, vol. 122, no. 1, 2015.
- [7] C. Patel, A. Patel, and D. Patel, “Optical character recognition by open source ocr tool tesseract: A case study,” *International Journal of Computer Applications*, vol. 55, no. 10, pp. 50–56, 2012.
- [8] M. Ganis, C. Wilson, and J. Blue, “Neural network-based systems for handprint ocr applications,” *IEEE Transactions on Image Processing*, vol. 7, no. 8, pp. 1097–1112, 1998. DOI: 10.1109/83.704304.