SUDOKU SOLVER

A PROJECT REPORT

Submitted by

SAFIN ABDUL NAZAR	(185002084)
VISHNU PRAKASH PRASANNAN	(185002124)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

Department of

Information Technology

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Rajiv Gandhi Salai (OMR), Kalavakkam – 603 110

JUNE 2022

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project titled "**SUDOKU SOLVER**" is the bonafide work of "**SAFIN ABDUL NAZAR** (185002084) and **VISHNU PRAKASH PRASANNAN** (185002124)**"**, who carried out the project work under my supervision.

Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degreeor award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. C. ARAVINDAN	**Dr. T. SREE SHARMILA**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor,	Assistant Professor,
Department of IT,	Department of IT,
SSN College of Engineering,	SSN College of Engineering,
Kalavakkam – 603 110	Kalavakkam – 603 110

Submitted for Project Viva-Voce Examination held on..................................

EXTERNAL EXAMINER **INTERNAL EXAMINER**

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We are honoured to acknowledge the below mentioned people who have been instrumental in the completion of this project.

We express our deep respect to the founder **PADMA BHUSHAN Dr. SHIV NADAR**, Chairman, SSN Institutions. We also express our appreciation to our Principal, **Dr. V.E. ANNAMALAI**, for all the help he has rendered during this course of study.

Our sincere thanks to **Dr. C. ARAVINDAN**, Professor and Head of the Department of Information Technology, for his words of advice and encouragement. We would like to thank and show our deep sense of gratitude to our guide **Dr. T. SREE SHARMILA**, Associate Professor, Department of Information Technology, for her valuable advice and suggestions as well as her continued guidance, patience and support that helped us to shape and refine our work.

Also, we would like to thank our Project Coordinators **Dr. T. SREE SHARMILA**, Associate Professor, Department of Information Technology and **Dr. N. BHALAJI**, Associate Professor, Department of Information Technology for their valuable suggestions throughout the project. We are grateful to our Project Panel Members for providing us with helpful pointers throughout the course of the project.

We would like to extend our sincere thanks to all the staff of our department who have contributed directly and indirectly during our project work. Finally, we would like to thank our parents and friends for their patience, cooperation and moral support throughout our life.

SIGNATURE

SAFIN ABDUL NAZAR (185002084)

SIGNATURE

VISHNU PRAKASH PRASANNAN (185002124)

ABSTRACT

Sudoku is a logic-based combinatorial puzzle. The game consists of a 9X9 grid divided into 9 square sub-grids (of size 3x3). Some of the squares in the grid contain a number from 1 to 9. The player is presented with a partially filled grid and their aim is to fill the rest of the squares with numbers from 1 to 9. The rules are each row, column and 3x3 grid must contain each of these numbers exactly once. This project proposes an optimal way of recognizing a sudoku puzzle using computer vision and deep learning techniques. The first step is to create a custom dataset of around 300 printed, handwritten and partially handwritten sudoku images, which are then processed using Gaussian Blurring and Adaptive Thresholding techniques. Then, the digits are recognized through optical character recognition using convolutional neural network, and the puzzle is solved by recursive backtracking algorithm with high accuracy. Also, a front-end user interface for capturing and displaying the sudoku puzzle can be done. Finally, the matrix is filled with the correct values of the solved puzzle and the final solved puzzle is displayed on the screen.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
AI	Artificial Intelligence
I/O	Input-Output
2-D	Two-Dimensional
IEEE	Institute of Electrical and Electronics Engineers
ICIIP	International Conference on Image Information Processing
IJSRD	International Journal for Scientific Research & Development
IJACSA	International Journal of Advanced Computer Science and Applications
I2CT	International Conference for Convergence in Technology
ICICCS	International Conference on Intelligent Computing and Control Systems
HGFMA	Hybrid Genetic Algorithm-Based Firefly Mating Algorithm
CNTK	Cognitive Toolkit
OCR	Optical Character Recognition
MNIST	Modified National Institute of Standards and Technology
CNN	Convolutional Neural Networks
k-NN	k-Nearest Neighbors
UI	User Interface

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

Sudoku is a logic-based combinatorial problem with a 9x9 grid divided into 9 square sub-grids (of size 3x3). A number from 1 to 9 appears in some of the squares in the grid. The player is given a partially filled grid and must fill in the remaining spaces with numbers ranging from 1 to 9. Every day, a large number of people attempt to solve sudoku puzzles. Newspapers, periodicals, and other publications frequently contain these puzzles. If a person is unable to solve a riddle, displaying the solution as a matrix would be very helpful.

1.2 NEED FOR THE STUDY

Sudoku puzzles of varied complexity levels can be found in newspapers and other print and digital media in real life. For many people, it is a popular leisure pastime. However, it has been noticed that the solution is not always available for people to verify right away. In most circumstances, consumers must either wait until the next day to check the solutions to the Sudoku puzzles they just completed, or rely on current applications, which may or may not provide the proper solution for all of the puzzle's difficulties. The need for the study is to develop an interface that can handle all Sudoku puzzle difficulties, handle fuzzy images, and come up with a high-accuracy optimised algorithm via which Sudoku solutions can be found that could be helpful for many sudoku solving enthusiasts or even just regular solvers.

1.3 OBJECTIVES OF THE STUDY

- To propose an optimal way of recognizing a Sudoku puzzle using computer vision and deep learning.
- To Detect Sudoku Grid from images even if they are slightly blurred or in different angles and dark lighting conditions and possibly achieve a high accuracy while processing the image.
- To solve the puzzle using recursive backtracking algorithm or any other algorithm that will give us a high accuracy, and display the solved puzzle as a matrix.
- To develop a front-end user interface to accept images from the user through a webcam and display the solved sudoku puzzles on the screen.

CHAPTER 2

REVIEW OF LITERATURE

Several research papers related to this project from reputed journals and conferences such as IEEE and Springer were reviewed. The papers were studied thoroughly and surveyed using different parameters such as the dataset used, methods/technologies used, the accuracy and whether there was some future scope of the papers or not. The limitations of these research papers were also studied and depending on the limitations, the objectives were set for the project and the aim was to improve on the already existing limitations.

The first paper that was reviewed was titled “Detection of Sudoku puzzle using image processing and solving by Backtracking, Simulated Annealing and Genetic Algorithms: A comparative analysis” [1]. It was Published in IEEE 2015 Third International Conference on Image Information Processing (ICIIP). The Methods used in this paper were solely based on algorithms for solving the sudoku puzzles such as Backtracking, Simulated Annealing and Genetic Algorithm. Pre-processing algorithms which include adaptive thresholding, Hough Transform and geometric transformation. Now coming to the dataset used, the proposed method for detecting sudoku on a dataset containing thirty sudoku puzzles filled with perspective distortions, various scales and ideal images. The dataset used was a Custom Dataset. In this study, they propose employing vision-based approaches to detect and decrypt a sudoku puzzle, and then solving the puzzle using three algorithms: Backtracking, Simulated Annealing, and Genetic Algorithm. The proposed method can recognise any sudoku puzzle captured with a digital camera, and after applying appropriate pre-processing algorithms such as adaptive thresholding, Hough Transform, and geometric transformation, the digits are recognised with Optical Character Recognition (OCR), and they are stored in corresponding locations in the 9x9 matrix based on their pixel locations in the image. Coming to the accuracy, the method employed was able to accomplish 73.33% success rate on the data set. Moreover, 98.835% of the numbers were detected, out of which

99.424% of the numbers were correctly guessed.

The second paper that was reviewed was titled “Processing Sudoku Images and Solving by Recursive Backtracking Algorithm” [2]. It was published in the journal IJSRD International Journal for Scientific Research & Development| Vol. 5, Issue 04, 2017. In this paper, they Use efficient area calculation algorithms to recognize the puzzle's enclosing box after applying appropriate pre-processing to the captured image. The digit places are then identified using a virtual grid. For digit recognition, the Tesseract OCR (optical character recognition) engine is employed. A backtracking algorithm is used to calculate the actual solution. They have used a Custom Dataset of Sudoku Images from books and newspapers. The solver can answer a Sudoku puzzle immediately from an image taken with any digital camera. This would mostly necessitate five steps. The first step recognises the puzzle's grid. The puzzle's grid is extracted in step two. The extraction of the numbers is the third step. Step 4 is where the extracted numbers are recognised. Step 5 uses a recursive backtracking technique to solve the puzzle. Artificial Neural Networks detect the digits using the OCR method. Future scope of this paper includes Improving Grid detection, Image Pre-processing and digit recognition to improve accuracy.

Another paper that was reviewed was titled “A Hybrid Approach for the Sudoku Problem: Using Constraint Programming in Iterated Local Search” [3]. It was published in the IEEE Intelligent Systems Volume 32, 2017. This paper mainly focused on the type of algorithm best suited for solving the sudoku puzzles such as Heuristic methods, Sudoku Iterated Local Search, Constraint programming, Min-conflicts heuristic, Hybrid techniques. In this paper they mentioned that although existing methods work well for little puzzles, larger ones are still difficult to solve. The dataset used was a Custom Dataset Of sudoku 2D matrices (arrays). The min-conflicts heuristic is used in this article to present a novel local search approach for Sudoku. In addition, the authors suggest a novel hybrid search technique that uses constraint programming as a perturbation tool within an iterated local search framework. They put their methods to the test on difficult Sudoku benchmarks and found that they outperformed state-of-the-art solutions. The accuracy achieved for the hardest instances the success rates are 38% and 12%, whereas the algorithm has success rates of 57% and 13%. When compared to state-of-the-art approaches from the literature, experimental results reveal that this algorithm is capable of solving puzzles of various levels of difficulty. For Sudoku problem instances with an order of four or five, this solver currently produces the best results.

The next paper was titled "An Efficient Machine Learning Technique to Classify and Recognize Handwritten and Printed Digits of Sudoku Puzzle" [4]. Published in the Journal: (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10 in 2019, this paper discusses Adaptive threshold, Hough Transform, and other image processing and filtering techniques that are used to recognize the Sudoku puzzle grid. The datasets used for digit recognition are MNIST and Char74K dataset for training and for the sudoku images a Custom Dataset of Sudoku Images from books and newspapers was used. The authors propose a CNN model to recognise and classify handwritten and printed digits in Sudoku puzzles taken with a camera from various periodicals and printed papers in this paper. Adaptive threshold is one of the image processing and filtering techniques used to detect the Sudoku puzzle grid. The paper's approach is thoroughly evaluated using a set of 100 Sudoku photos obtained with cameras under various situations. With a 98 percent accuracy rate, the technique appears to be promising. The relevance of the work is that, because the model can distinguish both handwritten and printed characters, it will be possible to identify and classify both handwritten and printed phrases.

A Paper titled "A Novel Automated Solver for Sudoku Images" [5] was also reviewed by us. It was published in IEEE 2019 5th International Conference for Convergence in Technology (I2CT). This study looks into using image processing, machine learning techniques for OCR, and an efficient solving algorithm to compute the correct answer to solve Sudoku puzzles (as seen in newspapers and mobile games). In order to correctly recognize the numbers, Custom Optical Character Reader (OCR) is used, which is based on the k-NN machine learning algorithm and solving the sudoku puzzle using backtracking algorithm. This process used a custom image dataset for printed digits. Coming to the accuracy, the algorithms were tested on roughly 25 newspaper photos of Sudoku puzzles and were able to properly detect 97.10 percent of the digits. Only 5 of the 25 grids could not be accurately detected/solved, giving an application accuracy of around 80%. Future Scope of this paper includes more robust image processing techniques can be utilised, and various approaches (owing to the subjective nature of the process) may work better. Because the k-nearest approach is a lazy learning algorithm, the machine learning technique utilised can be improved.

The sixth paper reviewed was titled "A Deep Learning approach to solve sudoku puzzle" [6] and was published in 2021 in the IEEE 5th International Conference on Intelligent Computing and Control Systems (ICICCS). The tools used in this paper are OpenCV, OCR, and TensorFlow. Tesseract is used to OCR the numerical data from the retrieved image, and then the numerical data is fed into the neural network (TensorFlow) model. The dataset used to train the neural network in the proposed model is taken from Kaggle. OpenCV, OCR, and TensorFlow were some of the popular tools used. The paper outlines how the authors solved the Sudoku puzzle using these methods and how they approached it. Picture pre-processing and image extraction was done using OpenCV, OCRing numerical data from the extracted image using Tesseract, and lastly feeding the numerical data extracted to the neural network (TensorFlow) model to produce the required output are all part of the approach presented in the study. After ample amount of training the neural network model gives a significant amount of accuracy which can be rounded off to 78 percentage. In terms of changes to the current code, combine the model with a web UI to boost accessibility. Some of the most cutting-edge technology, such as augmented reality, can be employed to enhance the experience. Increase the model's accuracy so that it outperforms the current one. If the input image clarity is poor, the model will reject it, ensuring that the model provides correct results.

Finally, the last paper reviewed was titled "A novel hybrid genetic algorithm-based firefly mating algorithm for solving Sudoku" [7]. It was published in the year 2021 in the Springer Journal: Innovations in Systems and Software Engineering. This paper focussed on some advanced algorithms that could be used for solving sudoku puzzles and give a very high accuracy on all levels of difficulties of the puzzle. Methods that were used in this paper for low, medium, and hard difficulty level puzzles, a genetic algorithm and a hybrid genetic algorithm-based firefly mating algorithm can solve Sudoku instances with a higher success rate. The Dataset used is a Custom Dataset Of sudoku 2D matrices (arrays) of 150 Instances with 50 Instance of each Easy, Medium and Hard Sudoku Arrays. The authors employ one of the heuristics, the genetic algorithm, to solve Sudoku successfully, but they run into complications. It has a lot of flaws, so they've hybridised it in a unique method to get around them. They present a hybrid genetic algorithm-based firefly mating algorithm that can solve Sudoku instances with a higher success rate for easy, medium, and hard difficulty levels puzzles in this research. Considering a smaller population and generation, the proposed strategy has prevented "becoming caught in local optima". The obtained accuracies are as follows: Easy – 100 %, Medium – 90%, Hard – 80%. The goal of future work for this technique is to overcome the obstacles that have been encountered so far and to implement it for other higher difficulty levels of Sudoku instances. Other projects aim to lower the number of generations required for the solution while also enhancing the method used here. Its HGFMA can be enhanced to operate better for simple cases as well.

After Reviewing all the research papers, the limitations were collected and have been listed below.

2.1 LIMITATIONS OF EXISTING WORK

There are certain limitations and gaps that are present in the existing systems:

- Algorithms used in the existing projects sometimes do not work well with varying levels of difficulties (Easy, Medium and Difficult) of the Sudoku puzzle and hence, produce wrong results.
- After capturing the Image, Image Processing of the captured Sudoku Image might not be able to Identify the grid and the digits if the image is not perfectly clear. Hence, even slight blurriness in the image causes the accuracy to reduce significantly.
- Usually, existing systems use only one algorithm to solve the puzzle, there is no comparative analysis of different algorithms and techniques.
- Due to the relatively small size of the training data for OCR, the recognition performance is likely to be impacted if the image of the puzzle used has an uncommon font. However, most published Sudoku use very similar fonts.

CHAPTER 3

SYSTEM DESIGN

The system design for the sudoku solver that is proposed is shown below in [Figure 1].

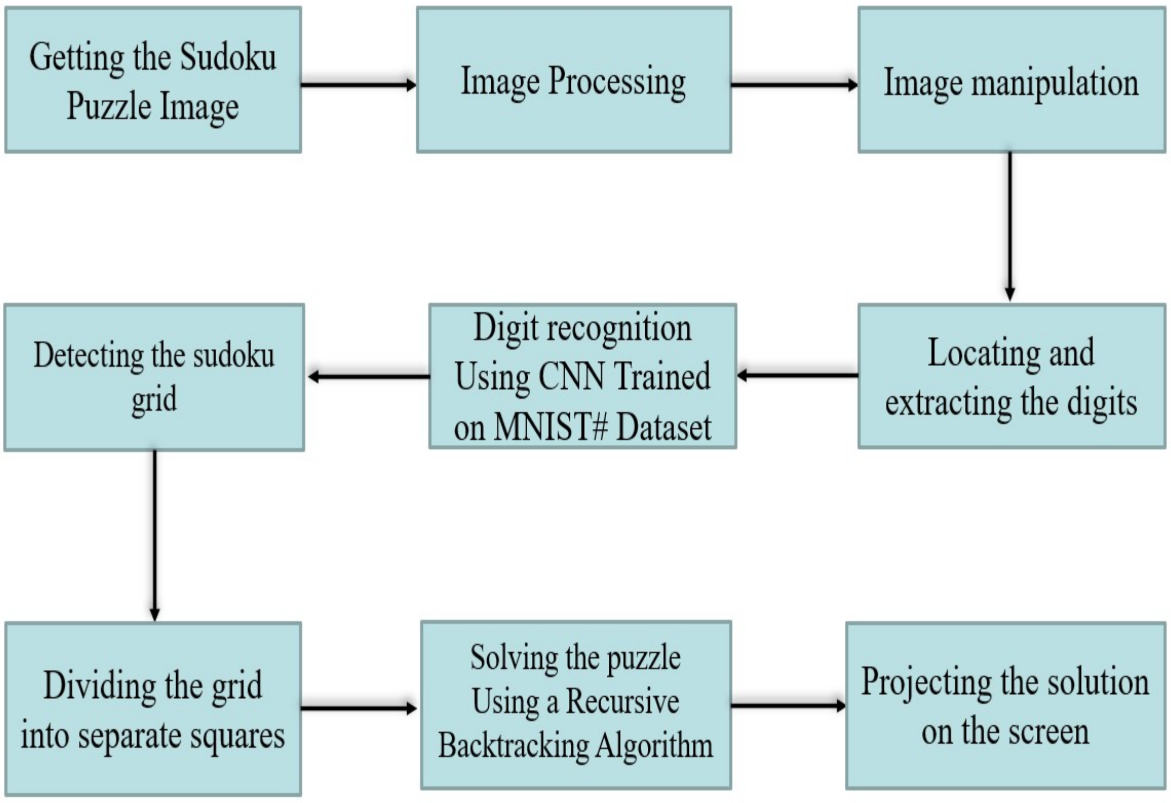

Getting the Sudoku Puzzle Image
Image Processing
Image manipulation
Locating and extracting the digits
Digit recognition Using CNN Trained on MNIST# Dataset
Detecting the sudoku grid
Dividing the grid into separate squares
Solving the puzzle Using a Recursive Backtracking Algorithm
Projecting the solution on the screen

Figure 1: System Design

The design proposed has nine steps in total. The first step starts with the collection of sudoku images i.e., creating a dataset of sudoku images either from books, newspapers or from websites. The next step involves the image manipulation and image processing part, where sudoku image is processed by removing noises and blurriness using several image processing techniques such as Gaussian Blur, Adaptive Thresholding, dilation, largest contour, cropping and warping. After the image processing step, the next step is to detect the sudoku grid, which involves detecting the boundaries of the sudoku image. In the next step, the detected grid is divided into separate (9X9) squares and the digits are placed in those separate squares. Now, the image manipulation and processing sections have been completed.

The fifth and the sixth step involve the OCR part which uses a deep learning CNN Model trained on the MNIST dataset to locate and extract the digits from the processed image and store those digits in a two-dimensional 9X9 array.

Since the extracted digits in an array have been obtained, the final steps involve solving the sudoku puzzle using recursive backtracking algorithm.

Finally, the solved sudoku puzzle can be displayed on the screen.

CHAPTER 4

IMPLEMENTATION

4.1 DATASET

For the purpose of this project, datasets have been collected from several sources. We have created a custom dataset containing around 300 images of sudoku puzzles with varying difficulty levels, categorized as printed (200 Images), handwritten (25 Images), and partially handwritten puzzles (75 Images).

Some of the images from the custom dataset have been displayed below in [Figure 2], [Figure 3], [Figure 4] and [Figure 5].

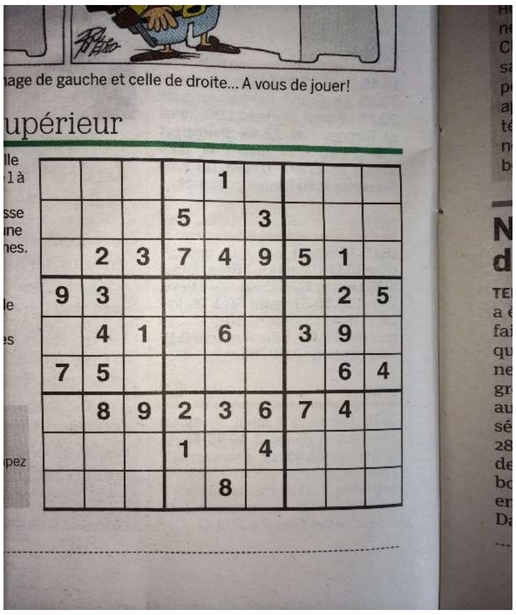


Figure 2: Custom Dataset Image 1

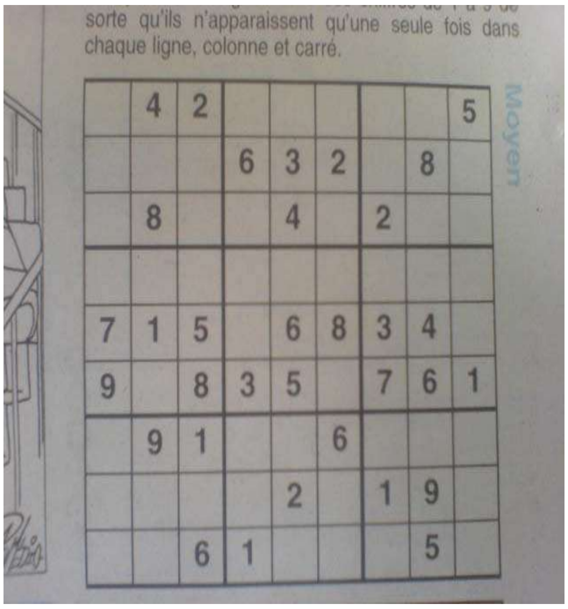
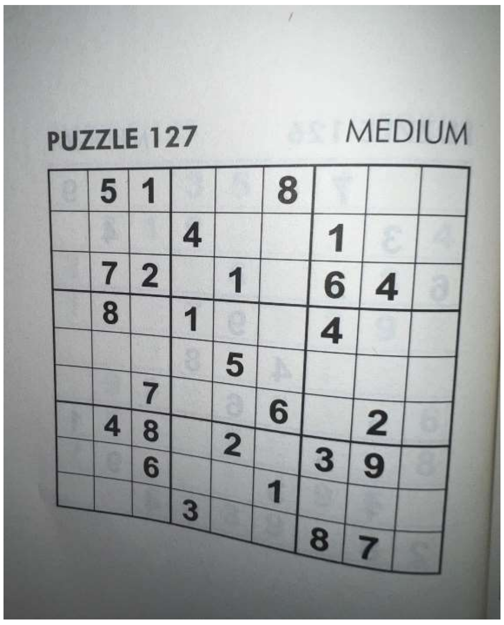

sorte qu'ils n'apparaissent qu'une seule fois dans chaque ligne, colonne et carré.
Moyen

Figure 3: Custom Dataset Image 2



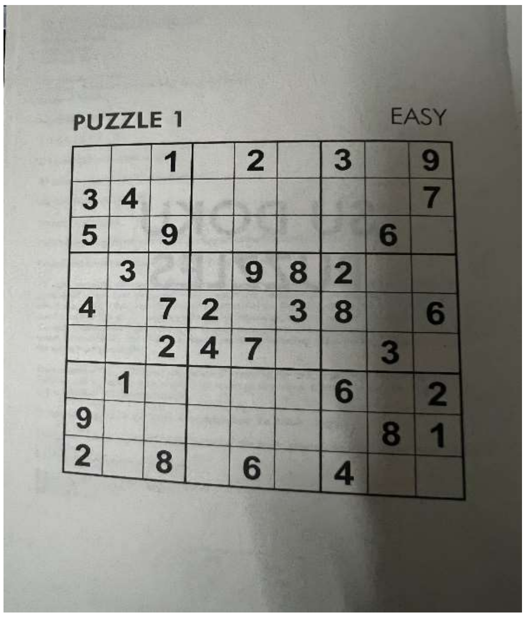
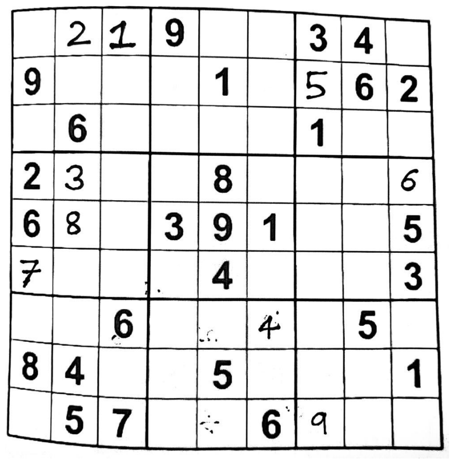


Figure 4: Custom Dataset Image 3 **Figure 5: Custom Dataset Image 4**

Examples of partially handwritten and fully handwritten images from the custom dataset have been displayed below in [Figure 6] and [Figure 7] respectively.



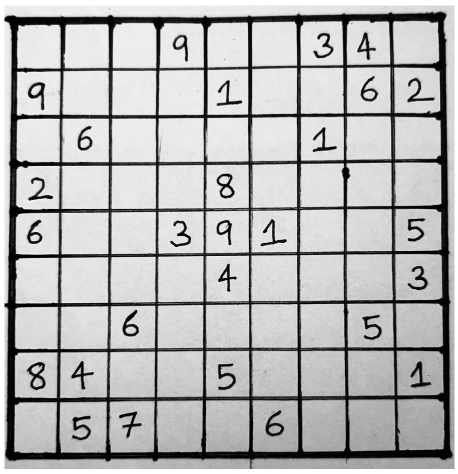


Figure 6: Custom Dataset Image 5 **Figure 7: Custom Dataset Image 6**

4.2 SUDOKU IMAGE PROCESSING AND MANIPULATION

The Implementation is divided into three parts:

1. Sudoku Image processing and manipulation.
2. Digit recognition using OCR.
3. Solving the sudoku using a suitable algorithm.

First, preprocessing of the image was done using Gaussian Blurring and Adaptive Thresholding techniques.

4.2.1 GAUSSIAN BLURRING

Instead of using the box filter, the picture is convolved with a Gaussian filter in the Gaussian Blur process. The Gaussian filter is a type of low-pass filter that reduces high-frequency components.

As demonstrated in equation [1], the Gaussian blur feature is created by blurring (smoothing) a picture with a Gaussian function to minimize noise. It can be thought of as a nonuniform low-pass filter that preserves low spatial frequency while reducing visual noise and minor features. It's usually done by using a Gaussian kernel to convolve an image. In 2-D form, this Gaussian kernel is written as

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

[1]

where sigma is the distribution's standard deviation and x and y are the location indices The value of regulates the extent of the blurring effect around a pixel by controlling the variance around a mean value of the Gaussian distribution.

4.2.2 ADAPTIVE THRESHOLDING

The threshold value in basic thresholding is global, meaning it is the same for all pixels in the image. Adaptive thresholding is a method for calculating threshold values for smaller regions, resulting in varying threshold values for different regions.

The method adaptiveThreshold() of the Imgproc class in OpenCV can be used to conduct adaptive threshold operation on an image.

Next, the colors of the image were inverted to make sure that the gridlines had non-zero-pixel values. Also, the size of gridlines was increased using the process of dilaton.

4.2.3 DILATION

Dilation's basic impact on a binary image is to gradually extend the boundaries of foreground pixels areas (typically white pixels). As a result, foreground pixel areas grow in size, while gaps within those regions shrink.
Two bits of data are fed into the dilation operator. The first is the image that will be magnified. The second is a structural element, which is a (typically tiny) group of coordinate points (also known as a kernel). The precise effect of the dilation on the input image is determined by this structural element.

The next step was to identify the corners of the largest polygon i.e., the 4 extreme corners of the largest contour in the image were found using the CHAIN_APPROX_SIMPLE Algorithm.

4.2.4 CHAIN APPROX ALGORITHM

CHAIN APPROX SIMPLE compresses horizontal, vertical, and diagonal segments along the contour, leaving only their end points. This means that any points along the straight pathways will be ignored, and only the final points will be left. Consider a contour that runs the length of a rectangle. Except for the four corner points, all contour points will be ignored. This approach is faster than CHAIN APPROX NONE since the algorithm does not retain all of the points, consumes less memory, and so runs faster.

Next, the getPerspectiveTransform() method was used to Crop and Warp the sudoku Image.

4.2.5 PERSPECTIVE TRANSFORM

The perspective of a given image or video can be shifted through Perspective Transformation using the getPerspectiveTransform() method to gain greater insights into the required information. The points on the image from which information is to be obtained must be supplied by changing the viewpoint in Perspective Transformation. The corner points within the image have to be displayed should also be specified. The perspective transform is then calculated from the two sets of points and wrapped around the original image.
cv2.getPerspectiveTransform is used first, followed by cv2.warpPerspective.

Finally, the 81-cell sudoku grid is inferred from the square image using a user defined method called parse_grid().

4.2.6 PARSE GRID

This is a user defined function to col lect all the digits from the processed image and place it back into their respective positions on a 9X9 sudoku grid.

The final processed sudoku image was thus obtained and the image was ready for the Optical Character Recognition [OCR] step.

A Sample of sudoku images before, during and after image processing are shown below in [Figure 8] ,[Figure 9] and [Figure 10].

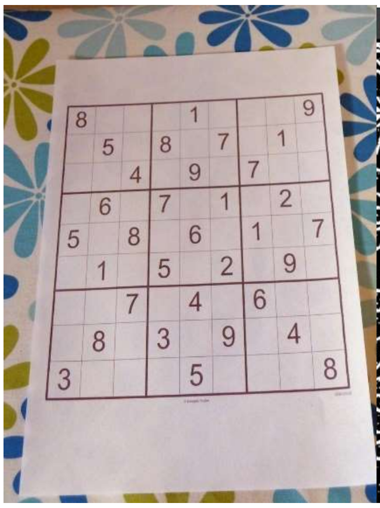


Figure 8: Image before Image Processing

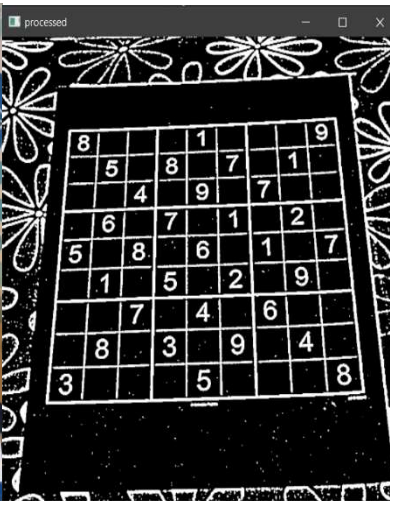


Figure 9: Image during Image Processing

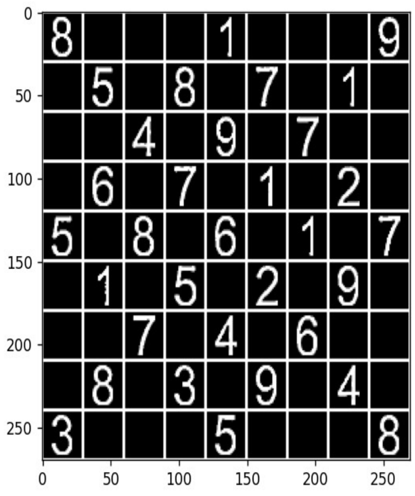


Figure 10: Image after Image Processing

The samples of image processing results have also been included for partially and fully handwritten sudoku images in [Figure 11], [Figure 12], [Figure 13] and [Figure 14].

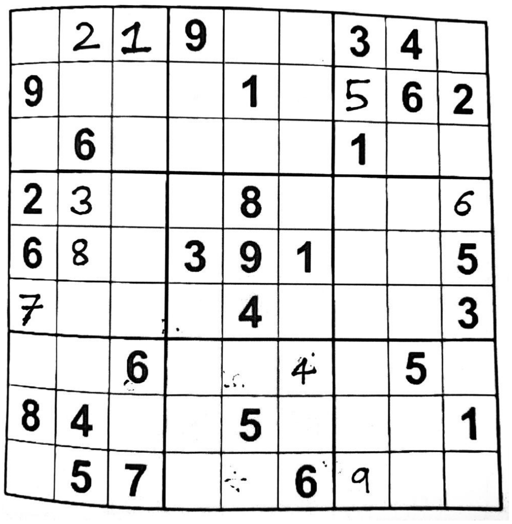


Figure 11: Partially Handwritten Image before Processing

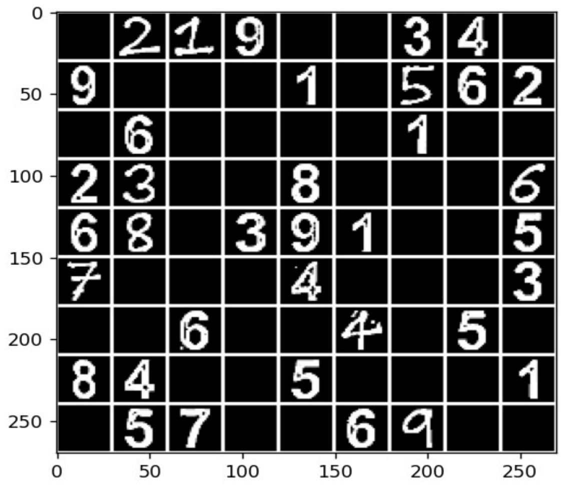


Figure 12: Partially Handwritten Image After Processing

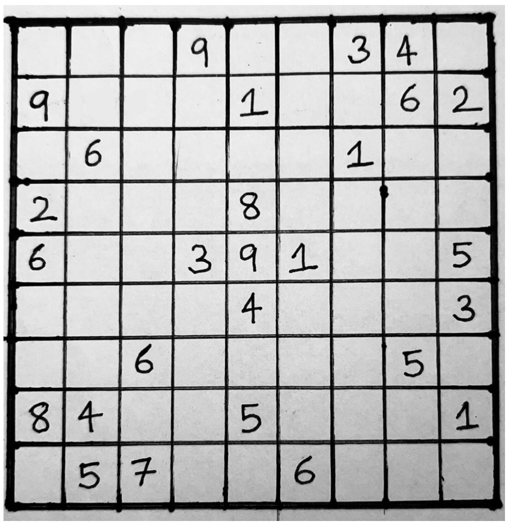


Figure 13: Fully Handwritten Image before Processing

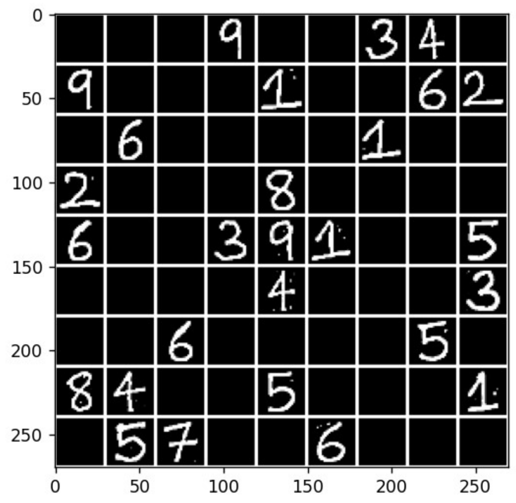


Figure 14: Fully Handwritten Image After Processing

Now, the final pre-processed image of the sudoku grid has been obtained. The next step involves the optical character recognition [OCR] in which the extracted digits in the final pre-processed are identified with the help of a trained Convolutional Neural Network [CNN] model.

The digits identified using OCR are then stored in a matrix in order to solve the sudoku puzzle using a suitable algorithm.

4.3 DIGIT RECOGNITION USING OCR

4.3.1 OPTICAL CHARACTER RECOGNITION

OCR stands for Optical Character Recognition in which text, patterns, digits present in an image or document are extracted. Text recognition is another name for optical character recognition (OCR). Data is extracted and repurposed from scanned papers, camera photos, and image-only pdfs using an OCR tool. OCR extracts letters from images, converts them to words, and then assembles sentences, allowing access to and alteration of the original text. It also removes the necessity for data entering by hand.

OCR turns physical, printed documents into machine-readable text using a mix of hardware and software. Text is copied or read by hardware (such as an optical scanner or dedicated circuit board), and advanced processing is usually handled by the software.

Machine Learning, Deep Learning and Artificial intelligence (AI) can be used in OCR to implement more complex methods of character recognition , such as distinguishing languages or handwriting styles. OCR is most typically used to convert hard copy legal or historical documents into pdf files so that users may edit, format, and search them as if they were written in a word processor.

In this case, OCR is used to extract and identify the digits (0 – 9) from the image of the sudoku puzzle.

To implement OCR on these sudoku images, a CNN Model is used which in turn has to be trained on some dataset which has a large collection of digits. For this, a CNN model was trained on the Modified National Institute of Standards and Technology dataset [MNIST], which has a huge collection of handwritten digits (0 – 9).

4.3.2 MNIST DATASET

The MNIST (Modified National Institute of Standards and Technology) database contains a significant number of handwritten digits. It has a 60,000-example training set and a 10,000-example test set. It's a subset of the larger NIST Special Database 3 (numbers written by US Census Bureau personnel) and Special Database 1 (digits written by high school students) databases, both of which contain monochrome photographs of handwritten digits. In a fixed-size image, the digits have been size-normalized and centered. The original NIST black and white (bilevel) photos were size adjusted to fit in a 20x20 pixel box while maintaining the aspect ratio. As a result of the normalization algorithm's anti-aliasing approach, the produced images have grey levels. By determining the center of the pixels and translating the image to position this point at the center of the 28x28 field, the images were centered in a 28x28 image.

[Figure 15] below shows some of the digits present in the MNIST dataset.

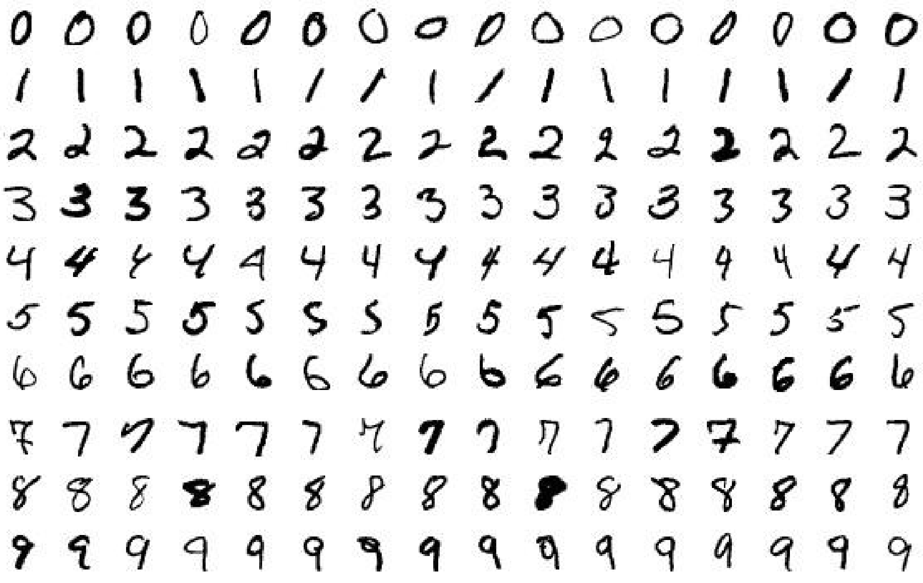


Figure 15: **Digits in MNIST Dataset**

As evident from [Figure 15], the MNIST dataset contains a wide range of handwritten digits for each of the digits (0 – 9). This dataset can be used to train a CNN model which will help in progressing with the OCR for the sudoku puzzle images.

4.3.3 CONVOLUTIONAL NEURAL NETWORK

CNNs are a type of multilayer perceptron that can use back-propagation to develop filters that need to be processed by machine learning models. Visual symbolism is the primary use of convolutional networks. A convolutional neural organization includes specific covered up layers in addition to the information and yield layers because the preparation cycle includes finding out about examples from more modest instances. These layers typically include a convolutional layer with learnt filters, a corrected long measure layer for enactment work, a pooling layer for down-testing, and a misfortune layer for determining punishment for incorrect output. To handle the CNN model, Keras is used with TensorFlow as the backend and provide a comparison of different CNN usages framed by different hyperparameters associated with each layer.

For training the CNN Model with the MNIST dataset, the TensorFlow and Keras libraries are used.

4.3.4 TENSORFLOW

Machine learning is a difficult field to master. However, thanks to machine learning frameworks like Google's TensorFlow, the process of obtaining data, training models, serving predictions, and refining future outcomes is significantly less scary and challenging than it used to be.

TensorFlow combines a variety of machine learning and deep learning (also known as neural networking) models and algorithms into a single metaphor. It makes use of Python to create a user-friendly front-end API for developing applications with the framework, which is then executed in high-performance C++.

Deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations can all be trained and run using TensorFlow. Best of all, TensorFlow allows for large-scale production prediction using the same models that were used for training.

Hence, for this sudoku solver OCR model TensorFlow is used to make the training much more efficient and robust.

4.3.5 KERAS

Keras is a machine learning framework built on top of open-source frameworks such as TensorFlow, Theano, and Cognitive Toolkit (CNTK). Keras is a Python module for doing quick numerical computations. The most well-known symbolic math library for constructing neural networks and deep learning models is TensorFlow. TensorFlow is extremely adaptable, and its main advantage is distributed computing. Microsoft created the CNTK deep learning framework. It makes use of languages like Python, C#, and C++, as well as standalone machine learning toolkits. Keras and TensorFlow are great libraries for building neural networks.

Keras is based on a simple framework that makes it simple to build deep learning models using TensorFlow or Theano. It is a deep learning framework that allows you to quickly define models.

4.3.6 TRAINING PHASE

Now the work on CNN model begins, which will use the MNIST informative index for digit perception. Preparations begin with a small group of imports. The MNIST informative collection of transcribed digits, which is integrated directly with Keras informational collections module, is the most prominent. Data collection is divided into two categories: training information and testing information. To show that the digits are grayscale, a channel measurement is added to them. They are then scaled down to a range of 0 and 1, and then a One-hot encoder is used to isolate and perceive the 0 and 1 integers. In one-hot encoding, number3 is represented as [0,0,0,1,0,0,0,0,0,0]. Preparing is triggered by a call to the fit strategy, and once it is completed, the model is then assessed and rated on the basis of the accuracy achieved.

The [Table 1] below shows the hyperparameters used in training the CNN model.

HYPER PARAMETERS	VALUES
Learning Rate	1e-3
Epochs	5
Batch size	128

Table 1: Hyperparameters used in CNN Model

The Below [Figure 16] shows the network design of the CNN model.

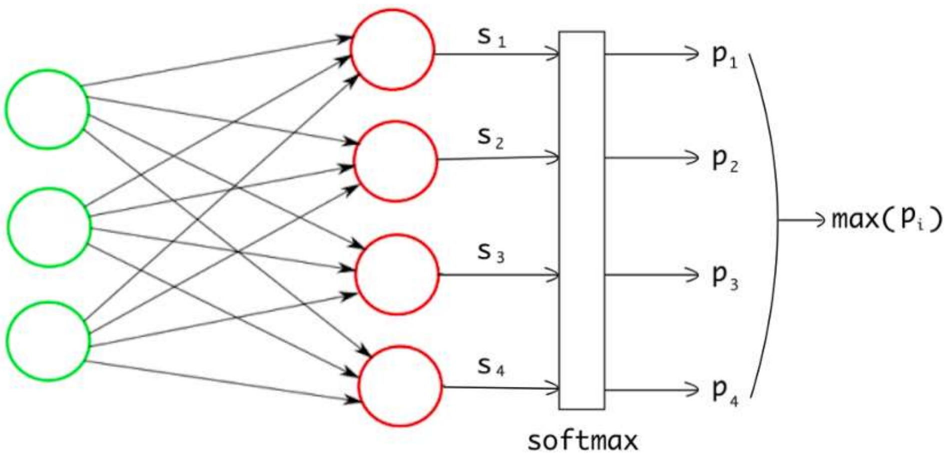
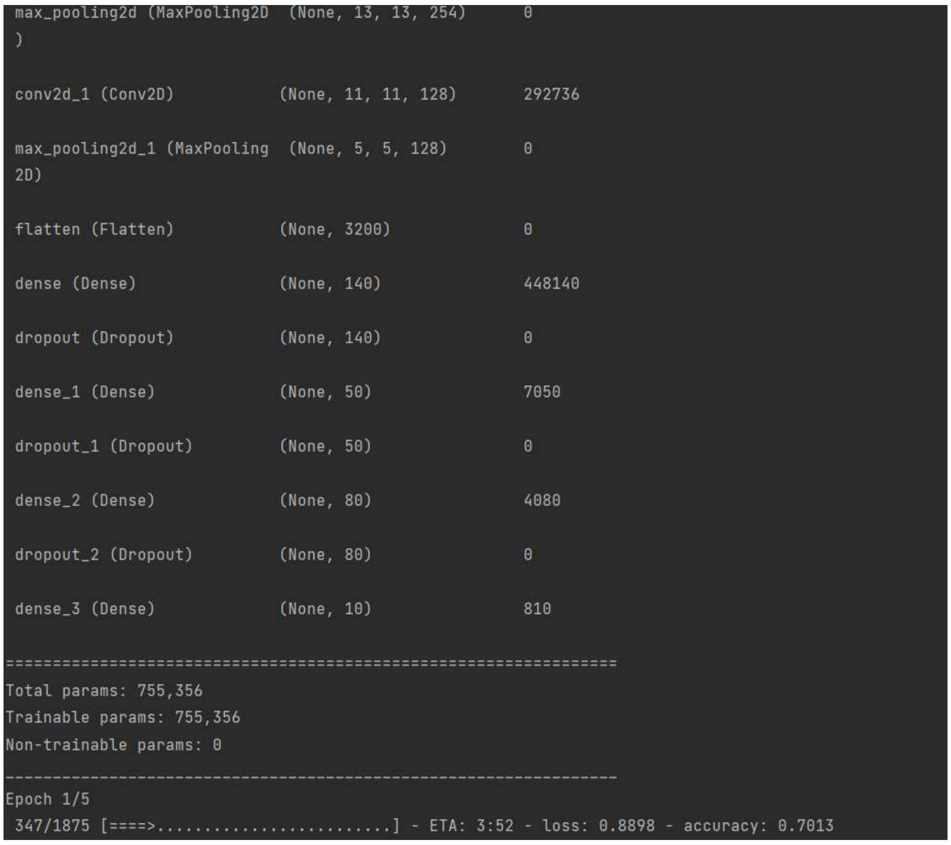


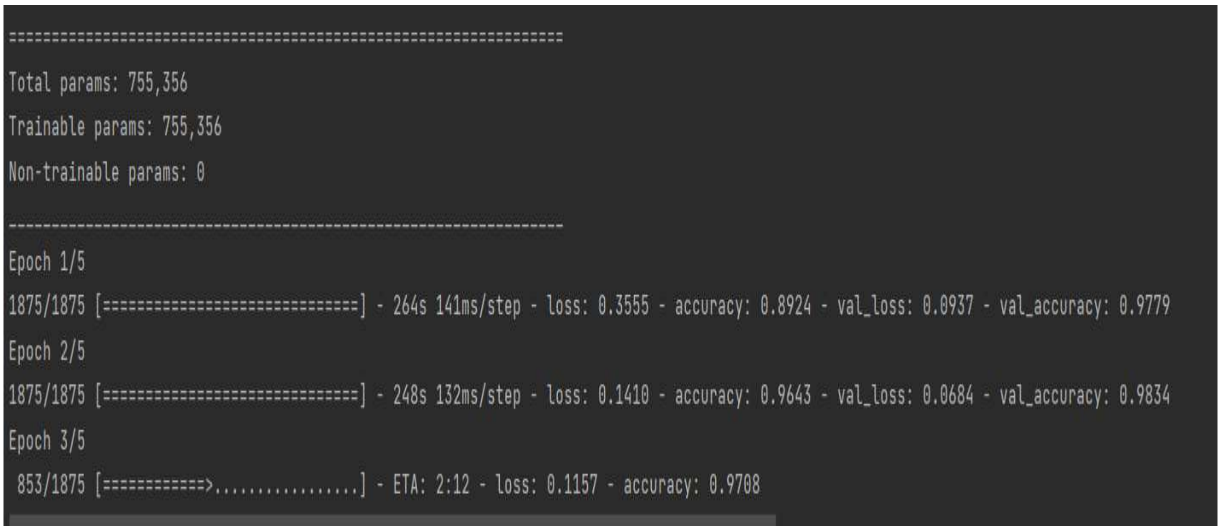
Figure 16: Network Design of CNN Model

The figures below ([Figure 17] and [Figure 18]) show the training of the CNN Model for 5 epochs and accuracy achieved using the MNIST dataset. The accuracy started out to be less than 90 percent, but as the number of epochs were increased and the model was further trained, the accuracy increased to above 90 percent. At the end of 5 epochs the model was able to achieve an accuracy of 95 percent. The trained model was then saved in model.h5 file, so that the model did not need to be trained again and again.



```
 max_pooling2d (MaxPooling2D  (None, 13, 13, 254)      0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 128)      292736

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 flatten (Flatten)           (None, 3200)             0

 dense (Dense)               (None, 140)              448140

 dropout (Dropout)           (None, 140)              0

 dense_1 (Dense)             (None, 50)               7050

 dropout_1 (Dropout)         (None, 50)               0

 dense_2 (Dense)             (None, 80)               4080

 dropout_2 (Dropout)         (None, 80)               0

 dense_3 (Dense)             (None, 10)               810

=================================================================
Total params: 755,356
Trainable params: 755,356
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
 347/1875 [====>.........................] - ETA: 3:52 - loss: 0.8898 - accuracy: 0.7013
```

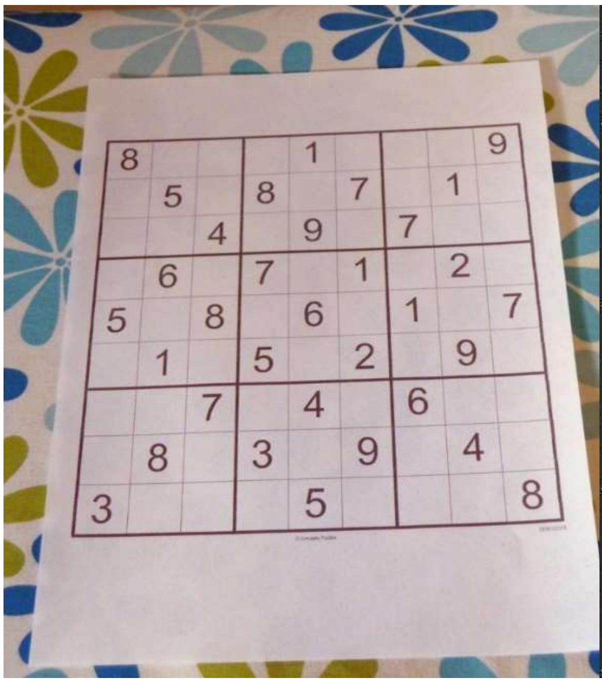
Figure 17: Training of CNN Model Image 1

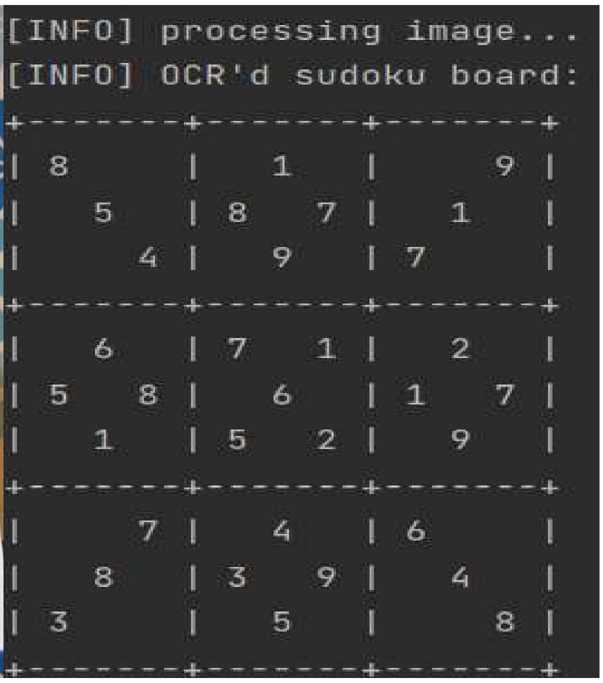


```
=================================================================
Total params: 755,356
Trainable params: 755,356
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 264s 141ms/step - loss: 0.3555 - accuracy: 0.8924 - val_loss: 0.0937 - val_accuracy: 0.9779
Epoch 2/5
1875/1875 [==============================] - 248s 132ms/step - loss: 0.1410 - accuracy: 0.9643 - val_loss: 0.0684 - val_accuracy: 0.9834
Epoch 3/5
 853/1875 [============>.................] - ETA: 2:12 - loss: 0.1157 - accuracy: 0.9708
```

Figure 18: Training of CNN Model Image 2

Now, after training the CNN model and saving it in the model.h5 file, OCR can be performed on the final pre – processed image.





```
[INFO] processing image...
[INFO] OCR'd sudoku board:
+-------+-------+-------+
| 8     |   1   |     9 |
|   5   | 8   7 |   1   |
|     4 |   9   | 7     |
+-------+-------+-------+
|   6   | 7   1 |   2   |
| 5   8 |   6   | 1   7 |
|   1   | 5   2 |   9   |
+-------+-------+-------+
|     7 |   4   | 6     |
|   8   | 3   9 |   4   |
| 3     |   5   |     8 |
+-------+-------+-------+
```

Figure 19: Sudoku Image before OCR **Figure 20: Sudoku Image after OCR**

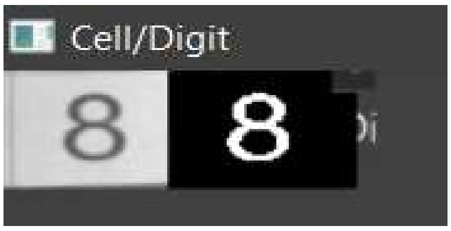


Figure 21: Digit Recognized after OCR

Above, [Figure 19], [Figure 20] and [Figure 21] show the final OCR image in which the digits have been recognized by the CNN model and finally the digits have been stored in a matrix.

Hence, now the OCR part has been completed and all the digits have been extracted from the sudoku puzzle image. The next step is to solve the sudoku puzzle and display the solution on the screen.

4.4 SUDOKU SOLVING ALGORITHM

Now, the OCR on the sudoku image has been performed, all the digits present have been extracted and stored in a 9X9 matrix. For solving the sudoku puzzle, a wide variety of existing algorithms are available which can provide a very high accuracy. Some of the few existing algorithms and methods that can be used are constraint programming, dynamic programming or backtracking.

Almost all of the existing algorithms promise a very high accuracy, so for solving the sudoku grid after the OCR process, Recursive Backtracking Algorithm was implemented.

4.4.1 DIFFICULTY LEVEL OF SUDOKU PUZZLES

Sudoku puzzles vary in difficulty depending on how the numbers are placed on the Sudoku board and how many numbers (clues) are given. In general, the most important part of Sudoku problem difficulty ratings is which approaches are necessary to solve the puzzles. In other words, it matters where the given numbers are logically put. Difficult puzzles are those that require additional approaches to solve. On the other hand, there are problems that can be solved using basic approaches and are classified as easy or medium level puzzles. As previously stated, there are three levels of difficulty employed in testing here (easy, medium and hard).

During testing, it was discovered that classifying difficulty levels is not as simple as it appears. This is because there have been puzzles that were labelled as difficult yet were solved using easy strategies, and vice versa. Some people assume that the difficulty levels are based on the quantity of numbers exposed on the puzzle board. In most cases, a Sudoku puzzle requires at least 17 clues to be solved. It means that solving a 17-clue Sudoku puzzle is more difficult than solving a 30-clue Sudoku puzzle. The more numbers provided, the easier and faster the solution becomes. This statement may not always be true, as testing has revealed that puzzles with fewer clues can be solved in less time than puzzles with more clues.

The relationship between the number of clues in the puzzles and their run-time is depicted in [Figure 22] below. It's reasonable to assume that as the number of clues increases, the time it takes to solve the problem will decrease. When solving a puzzle with 28 clues, for example, the solving time quickly increases. The reason for this is that in order to solve the puzzle, more strategies are required, or the algorithm must iterate until the solution is found.

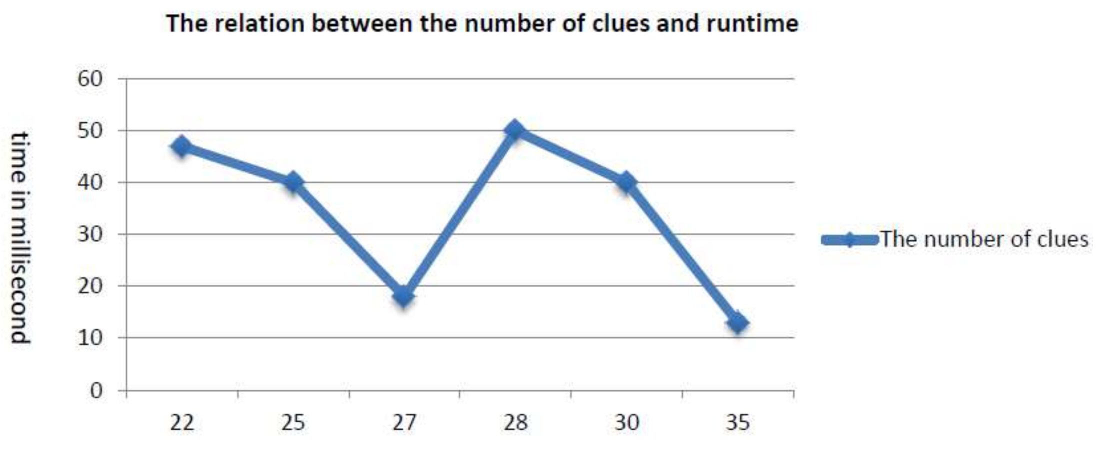


Figure 22: No. of Existing Digits vs Runtime Graph

4.4.2 BACKTRACKING

All puzzles with easy and medium levels of difficulty can be solved using the constrained programming method and the dynamic programming approach. The backtracking method was utilized to complete the algorithm in order to solve puzzles with ever more difficult levels, such as hard puzzles. The puzzle is solved by a human player using simple tactics. If the puzzle cannot be solved using these strategies, the player must guess to fill the remaining empty squares.

The constrained programming method is assisted by the backtracking method, which is analogous to human strategy (guessing). To put it another way, if the problem cannot be filled using the constrained technique or the dynamic approach, the backtracking method will take the puzzle and fill in the remaining empty squares. When the content of other squares in the same row, column, and box are taken into account, the backtracking approach finds an empty square and assigns the lowest valid integer to it. If none of the numbers from 1 to 9 are valid in a square, the algorithm returns to the preceding square, which was just filled.

4.4.3 RECURSIVE BACKTRACKING ALGORITHM

A recursive backtracking approach is used to solve the Sudoku puzzle. In a two-dimensional array, the numbers derived from the grid in the image are stored, with the number 0 assigned to blank grid places. A blank spot needs to be found in the grid to get a solution for it. A valid assignment is found by iterating through the numbers 1-9 using Sudoku rules. Then, using this new number arrangement, the sudoku grid is solved recursively. When there are no more grid locations that need to be filled, the grid is said to be solved, and the solved grid is returned to the main algorithm.

If the sudoku grid cannot be solved recursively, an alternative number assignment is tried for a place before attempting to solve the grid again. If all number assignment combinations are exhausted without yielding a solution, it is inferred that the grid is unsolvable, and the function will return false. In this case however, it is presumed that the user's image is a real Sudoku puzzle, and the algorithm will always solve it, regardless of the puzzle's difficulty level.

Now, in regards to the algorithm, the recursive backtracking algorithm is fairly simple and easy to understand. When compared to dynamic programming, the below algorithm for recursive backtracking gives a holistic view of how the sudoku puzzle is being solved:

```
Find row, column of an unassigned cell in the matrix
If there is none, return true
For digits from 1 to 9
a) If there is no conflict for digit at row, column
assign digit to row, column and recursively try fill in rest of grid
b) If recursion is successful, return true
c) Else, remove digit and try another iteration
If all digits have been tried and nothing worked, return false (no solution).
```

Alternatively, the algorithm can also be visualised as shown below:

```
function sudokusolve ():
find the first empty cell in the grid matrix
    if empty cell found in matrix
        check for all the values from digits 1 to 9
            if safe to enter a digit
                assign the digit to the empty cell in the matrix
                    sudokusolve ()
                    if sudoku grid remain unsolved then
                    clean the assigned digit
            if no digit leads to the solution return False
                if no empty cell found
                    solution found
store the solution and increment the count of total number of solutions
    return True
```

The Pseudocode for the recursive backtracking algorithm is show below:

```
recursiveBacktracking(sudokuPuzzle[int][int]){
sudokuPuzzle [int][int] //global
solvesudoku(row, column){
if (no more choices): the puzzle is solved!
if (puzzle[row][column]= not Empty): move to the next square. for 1 to 9:
if(checkrow(row, column, digit) & checkCol(row, column, digit) &
checkbox(row, column, digit){ sudokuPuzzle[row][column]= digit; move to the
next square
} if not, valid number is found go the previous square that was recently filled
}
```

4.4.4 TIME COMPLEXITY ANALYSIS

The time complexity of an algorithm is defined as the amount of time it takes for the algorithm to run on a computer and is often stated in O notation. Counting the number of operations performed by the algorithm is one technique to determine time complexity. The worst-case time of complexity is utilized, indicated as T, because the performance time can vary with different inputs of the same size (n). It is already known that the worst-case complexity is connected to the difficulty of the most difficult task.

The time complexity of the sudoku solver is investigated below. A sudoku solver takes a sudoku board as input. A 9X9 grid is used as the conventional board, but smaller and larger boards are also utilized. These small puzzles can be solved quickly and efficiently by a computer, but only because they are small for a computer. Solving a sudoku puzzle is an NP-complete problem, which means that sudoku algorithms do not scale well to larger boards or puzzles, such as 10000X10000 squares. If the size of the input to the sudoku solver is infinite, the complexity time will grow exponentially. However, it is possible to solve puzzles with small input sizes, such as 9X9 grids, because they can be solved in polynomial time.

Mathematically, the time complexity of the recursive backtracking algorithm is as follows in [Table 2]:

CASE OF COMPLEXITY	TIME COMPLEXITY
Worst Case Time Complexity	$O(9^m)$
Average Case Time Complexity	$O(9^m)$
Best Case Time Complexity	$O(m^2)$

Table 2: Time Complexity of Recursive Backtracking Algorithm

Note: The best-case time complexity takes place when the number of backtracking steps is minimized.

Since there are 9 possibilities to investigate for every unfilled cell, and there are m unfilled cells in the sudoku, the backtracking technique has an $O(9^m)$ time complexity in the worst case.

4.4.5 COMPARISON WITH OTHER ALGORITHMS

Recursive backtracking algorithm can be compared with other algorithms such as brute force algorithm and dancing links algorithm to see how well recursive backtracking performs against the other algorithms. The comparison can be done based on two parameters. The first is based on the memory consumption which is shown in [Figure 23] and the second is based on the time taken to solve the sudoku which are shown in [Figure 24] and [Figure 25].

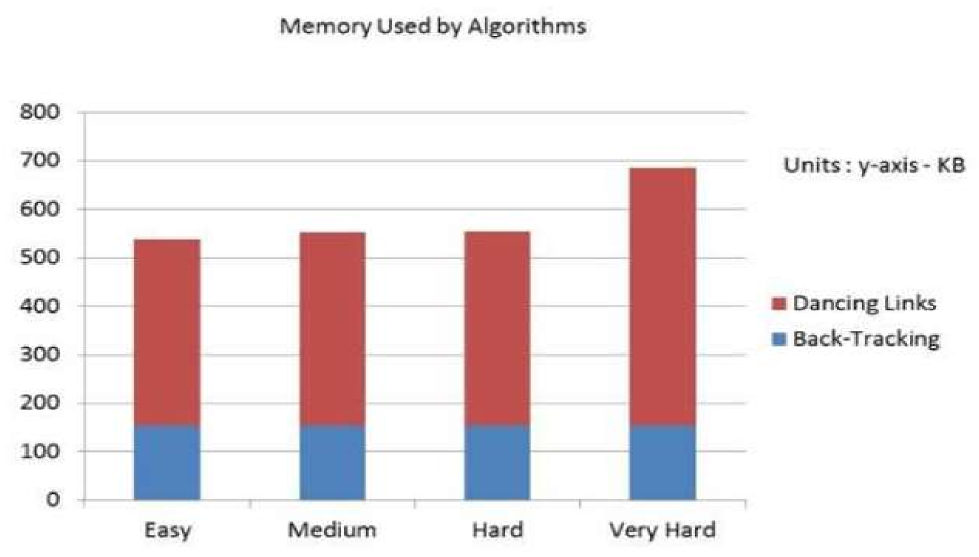
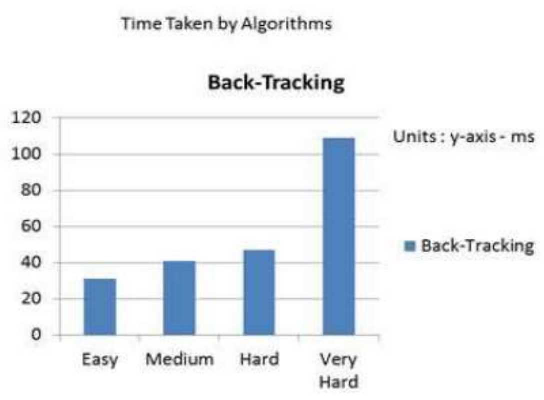


Figure 23: Memory used by Sudoku Algorithms



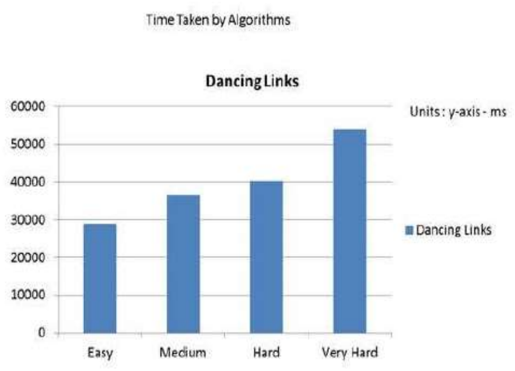
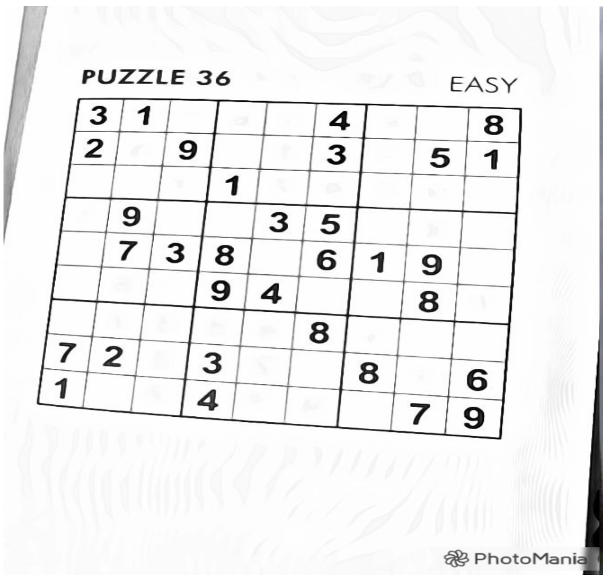


Figure 24: Time taken by Backtracking Algorithm

Figure 25: Time taken by Dancing Links Algorithm

4.4.6 SOLVED SUDOKU PUZZLES

Fully Printed Sudoku Image



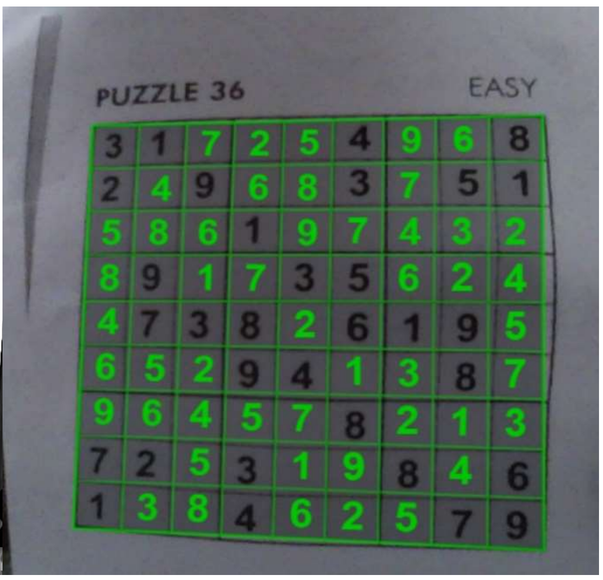
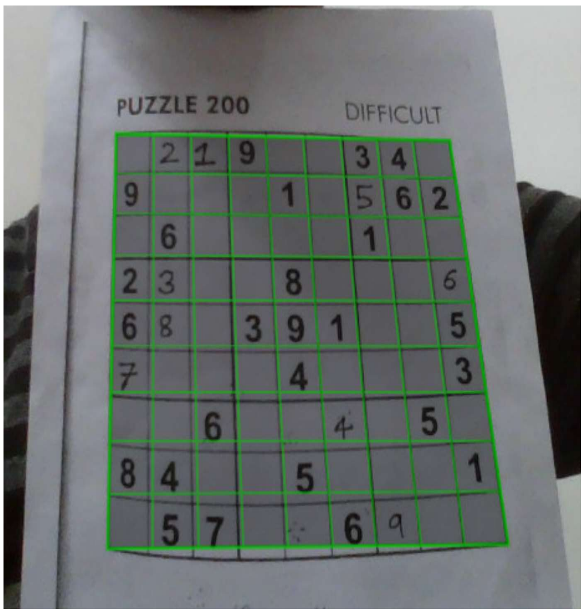


Figure 26: Printed Sudoku before Solving

Figure 27: Printed Sudoku after Solving

Partially Handwritten Sudoku Image



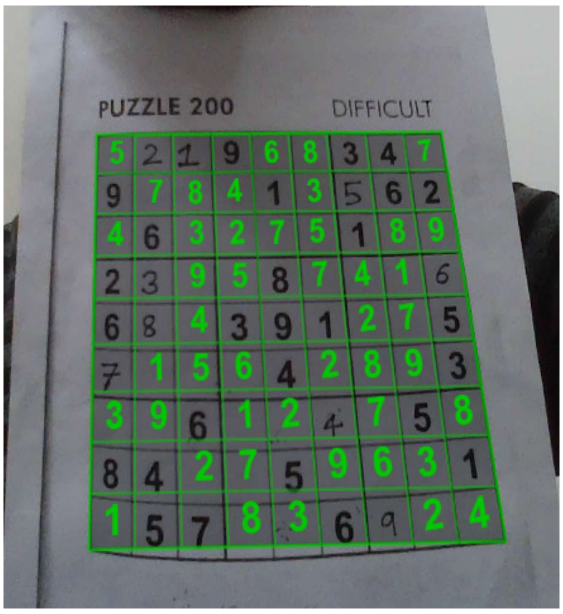


Figure 28: Partially Handwritten Sudoku before Solving

Figure 29: Partially Handwritten Sudoku after Solving

The above [Figure 26] and [Figure 27] show the solved sudoku that has all printed digits. Similarly, [Figure 28] and [Figure 29] show the solved sudoku that has both printed digits and handwritten digits.

The below [Figure 30] and [Figure 31] show the solved sudoku that has only handwritten digits and also the sudoku grid which is drawn by hand.

Fully Handwritten Sudoku Image

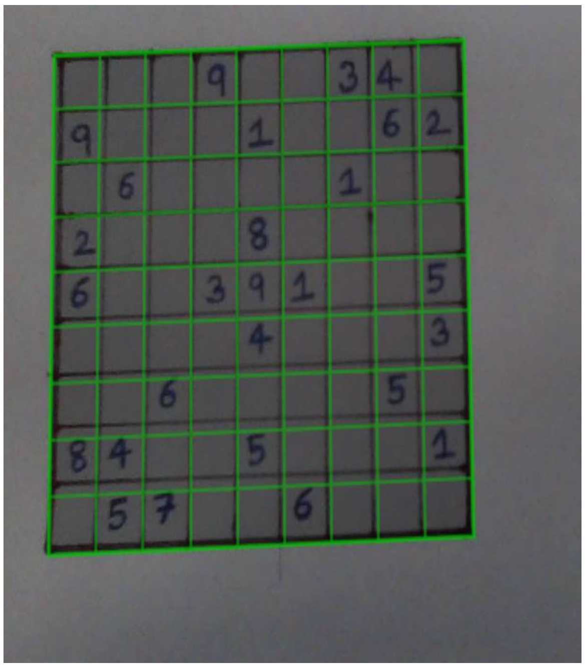


Figure 30: Fully Handwritten Sudoku before Solving

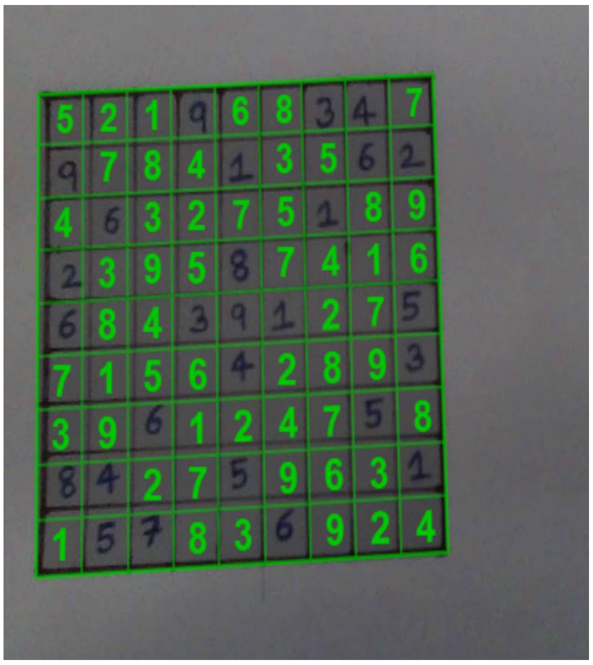
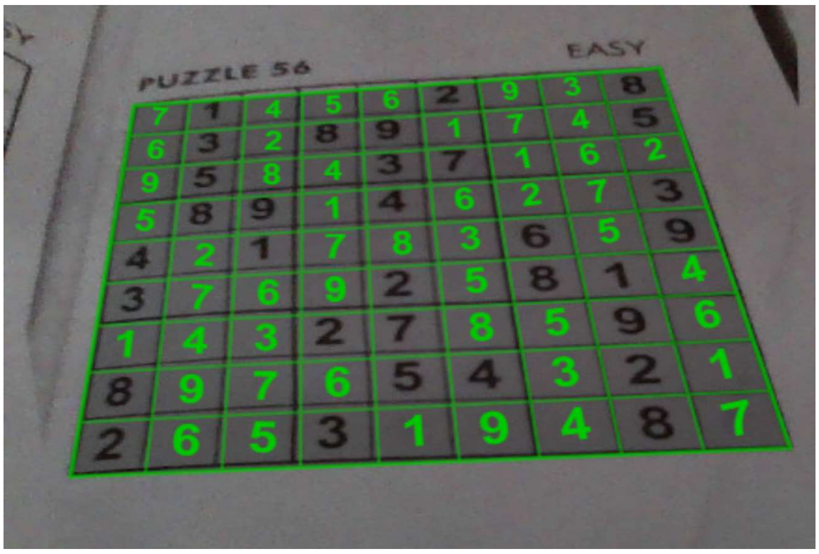


Figure 31: Fully Handwritten Sudoku after Solving

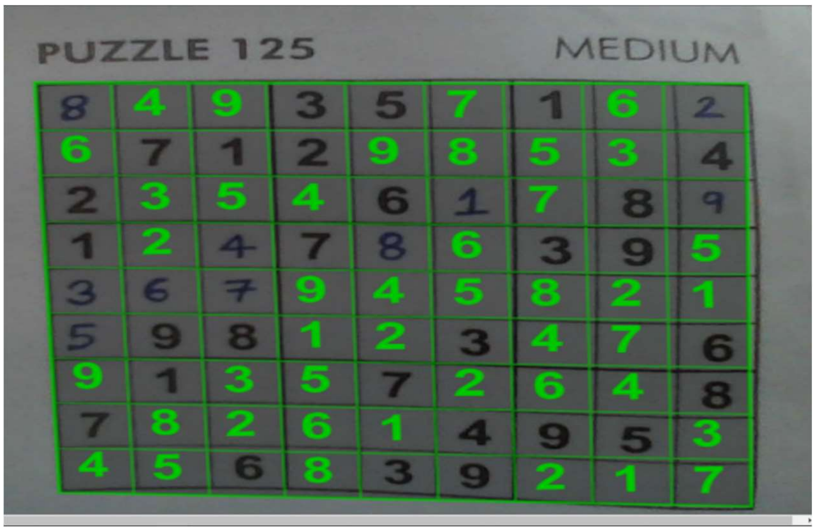
The below [Figure 32], [Figure 33] and [Figure 34] show the easy level, medium level and difficult level sudoku solved by the algorithm respectively



PUZZLE 56 EASY

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

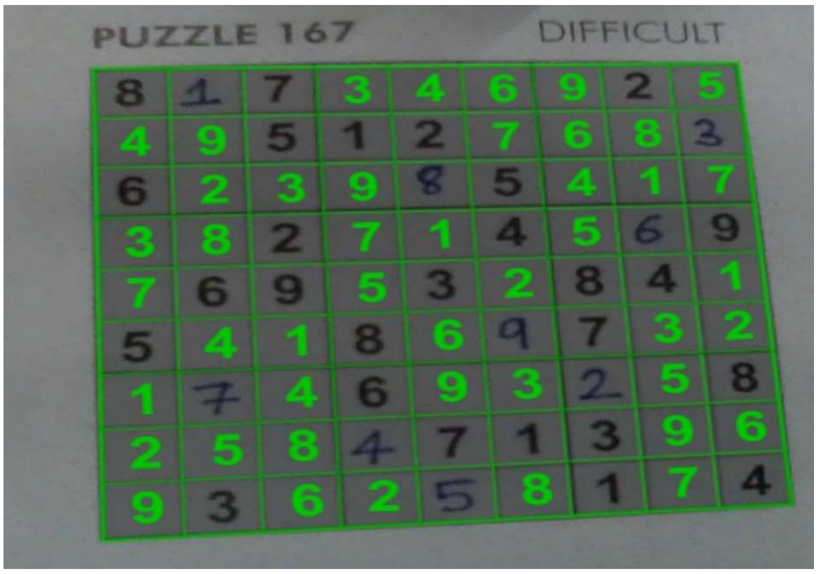
Figure 32: Easy Level Sudoku after Solving



PUZZLE 125 MEDIUM

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

Figure 33: Medium Level Sudoku after Solving

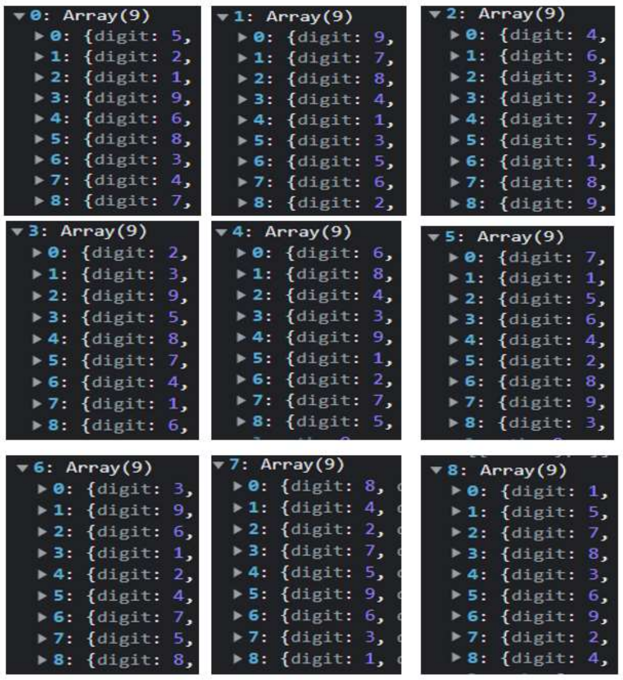


PUZZLE 167 DIFFICULT

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

Figure 34: Difficult Level Sudoku after Solving

The solutions of the sudoku puzzles are also stored in a 9X9 matrix which can be used by the user to verify the solution of the solved sudoku puzzle. [Figure 35] and [Figure 36] shows the solution stored in matrix form.



```
▼0: Array(9)
  ▶0: {digit: 5,
  ▶1: {digit: 2,
  ▶2: {digit: 1,
  ▶3: {digit: 9,
  ▶4: {digit: 6,
  ▶5: {digit: 8,
  ▶6: {digit: 3,
  ▶7: {digit: 4,
  ▶8: {digit: 7,
▼1: Array(9)
  ▶0: {digit: 9,
  ▶1: {digit: 7,
  ▶2: {digit: 8,
  ▶3: {digit: 4,
  ▶4: {digit: 1,
  ▶5: {digit: 3,
  ▶6: {digit: 5,
  ▶7: {digit: 6,
  ▶8: {digit: 2,
▼2: Array(9)
  ▶0: {digit: 4,
  ▶1: {digit: 6,
  ▶2: {digit: 3,
  ▶3: {digit: 2,
  ▶4: {digit: 7,
  ▶5: {digit: 5,
  ▶6: {digit: 1,
  ▶7: {digit: 8,
  ▶8: {digit: 9,
▼3: Array(9)
  ▶0: {digit: 2,
  ▶1: {digit: 3,
  ▶2: {digit: 9,
  ▶3: {digit: 5,
  ▶4: {digit: 8,
  ▶5: {digit: 7,
  ▶6: {digit: 4,
  ▶7: {digit: 1,
  ▶8: {digit: 6,
▼4: Array(9)
  ▶0: {digit: 6,
  ▶1: {digit: 8,
  ▶2: {digit: 4,
  ▶3: {digit: 3,
  ▶4: {digit: 9,
  ▶5: {digit: 1,
  ▶6: {digit: 2,
  ▶7: {digit: 7,
  ▶8: {digit: 5,
▼5: Array(9)
  ▶0: {digit: 7,
  ▶1: {digit: 1,
  ▶2: {digit: 5,
  ▶3: {digit: 6,
  ▶4: {digit: 4,
  ▶5: {digit: 2,
  ▶6: {digit: 8,
  ▶7: {digit: 9,
  ▶8: {digit: 3,
▼6: Array(9)
  ▶0: {digit: 3,
  ▶1: {digit: 9,
  ▶2: {digit: 6,
  ▶3: {digit: 1,
  ▶4: {digit: 2,
  ▶5: {digit: 4,
  ▶6: {digit: 7,
  ▶7: {digit: 5,
  ▶8: {digit: 8,
▼7: Array(9)
  ▶0: {digit: 8,
  ▶1: {digit: 4,
  ▶2: {digit: 2,
  ▶3: {digit: 7,
  ▶4: {digit: 5,
  ▶5: {digit: 9,
  ▶6: {digit: 6,
  ▶7: {digit: 3,
  ▶8: {digit: 1,
▼8: Array(9)
  ▶0: {digit: 1,
  ▶1: {digit: 5,
  ▶2: {digit: 7,
  ▶3: {digit: 8,
  ▶4: {digit: 3,
  ▶5: {digit: 6,
  ▶6: {digit: 9,
  ▶7: {digit: 2,
  ▶8: {digit: 4,
```

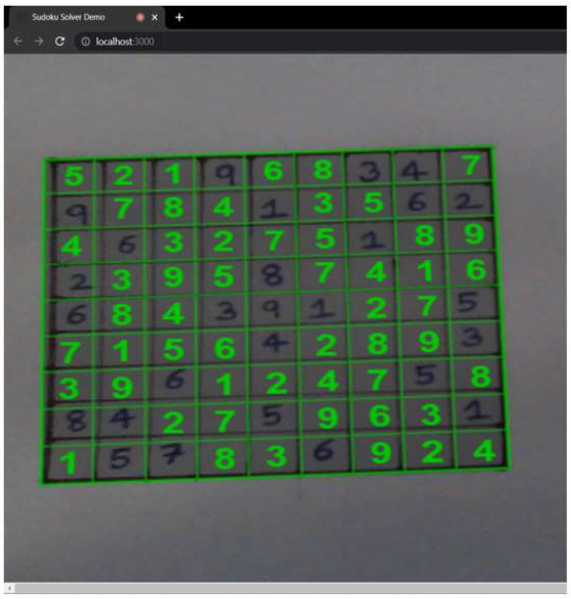
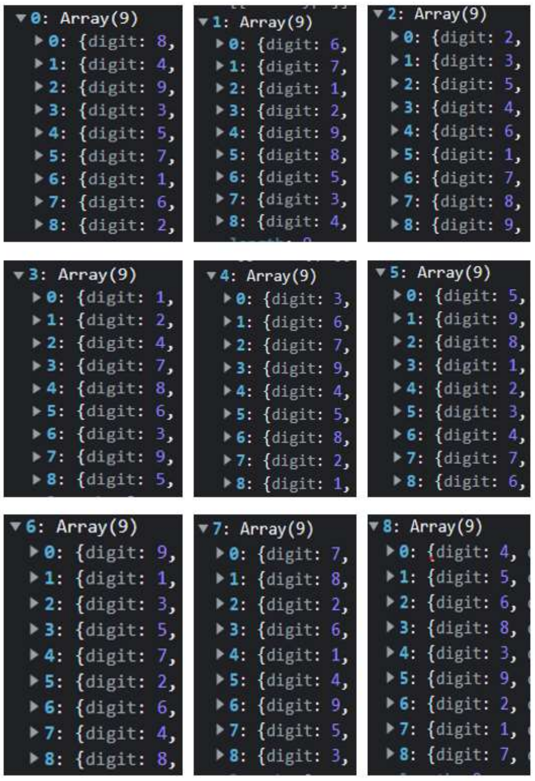


Figure 35: Solution in Matrix Form Image 1



```
▼0: Array(9)
  ▶0: {digit: 8,
  ▶1: {digit: 4,
  ▶2: {digit: 9,
  ▶3: {digit: 3,
  ▶4: {digit: 5,
  ▶5: {digit: 7,
  ▶6: {digit: 1,
  ▶7: {digit: 6,
  ▶8: {digit: 2,
▼1: Array(9)
  ▶0: {digit: 6,
  ▶1: {digit: 7,
  ▶2: {digit: 1,
  ▶3: {digit: 2,
  ▶4: {digit: 9,
  ▶5: {digit: 8,
  ▶6: {digit: 5,
  ▶7: {digit: 3,
  ▶8: {digit: 4,
▼2: Array(9)
  ▶0: {digit: 2,
  ▶1: {digit: 3,
  ▶2: {digit: 5,
  ▶3: {digit: 4,
  ▶4: {digit: 6,
  ▶5: {digit: 1,
  ▶6: {digit: 7,
  ▶7: {digit: 8,
  ▶8: {digit: 9,
▼3: Array(9)
  ▶0: {digit: 1,
  ▶1: {digit: 2,
  ▶2: {digit: 4,
  ▶3: {digit: 7,
  ▶4: {digit: 8,
  ▶5: {digit: 6,
  ▶6: {digit: 3,
  ▶7: {digit: 9,
  ▶8: {digit: 5,
▼4: Array(9)
  ▶0: {digit: 3,
  ▶1: {digit: 6,
  ▶2: {digit: 7,
  ▶3: {digit: 9,
  ▶4: {digit: 4,
  ▶5: {digit: 5,
  ▶6: {digit: 8,
  ▶7: {digit: 2,
  ▶8: {digit: 1,
▼5: Array(9)
  ▶0: {digit: 5,
  ▶1: {digit: 9,
  ▶2: {digit: 8,
  ▶3: {digit: 1,
  ▶4: {digit: 2,
  ▶5: {digit: 3,
  ▶6: {digit: 4,
  ▶7: {digit: 7,
  ▶8: {digit: 6,
▼6: Array(9)
  ▶0: {digit: 9,
  ▶1: {digit: 1,
  ▶2: {digit: 3,
  ▶3: {digit: 5,
  ▶4: {digit: 7,
  ▶5: {digit: 2,
  ▶6: {digit: 6,
  ▶7: {digit: 4,
  ▶8: {digit: 8,
▼7: Array(9)
  ▶0: {digit: 7,
  ▶1: {digit: 8,
  ▶2: {digit: 2,
  ▶3: {digit: 6,
  ▶4: {digit: 1,
  ▶5: {digit: 4,
  ▶6: {digit: 9,
  ▶7: {digit: 5,
  ▶8: {digit: 3,
▼8: Array(9)
  ▶0: {digit: 4,
  ▶1: {digit: 5,
  ▶2: {digit: 6,
  ▶3: {digit: 8,
  ▶4: {digit: 3,
  ▶5: {digit: 9,
  ▶6: {digit: 2,
  ▶7: {digit: 1,
  ▶8: {digit: 7,
```

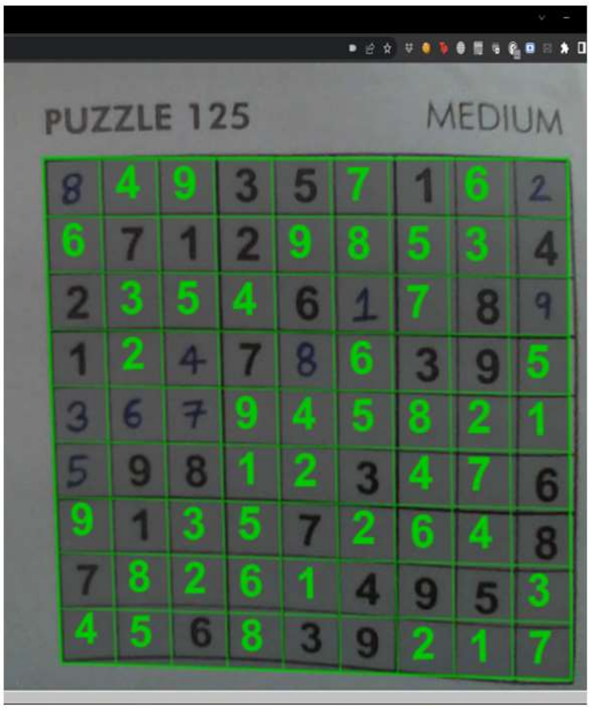


Figure 36: Solution in Matrix Form Image 2

Hence, the sudoku puzzle was solved using the recursive backtracking algorithm and the solution was displayed on the screen as well as stored in a matrix for verification by the user.

4.5 FRONTEND SUDOKU SOLVER WEBCAM

In order to make this sudoku solver more interactive, a simple front-end which takes the input of the sudoku puzzles through the webcam and displays the solved grid on the screen was developed.

The user is requested to hold the sudoku puzzle in front of the webcam. In some time, the sudoku grid is detected and the solution is displayed. Once, the solution is displayed, the user can once again try with another sudoku puzzle. This helps to avoid re-executing the code for each sudoku puzzle and provides a user-friendly interface for this project. The solution of the sudoku puzzle is also stored in a 9X9 matrix which can be accessed by the user in order to verify whether the OCR and the solution is correct or not.

The Frontend is simply a webcam page hosted on a local server using Node.js and Typescript.

4.5.1 NODE.JS

Node.js is a JavaScript-based framework that may be used to create online chat applications, video streaming sites, single-page applications, and a variety of other I/O-intensive web apps and web apps. It's based on Google Chrome's JavaScript V8 Engine, and it's utilised by a lot of programming projects. Node.js is lightning fast, and its library is incredibly quick for code execution because it is built on Google Chrome's V8 JavaScript engine. With over 50,000 bundles available in Node Package Manager (NPM), any functionality needed for an application may be quickly loaded. Hence, because of the above reasons it was decided to integrate the sudoku solver code with

Node.js in order to develop a simple frontend user interface which uses the webcam of the device to solve the sudoku puzzles.

4.5.2 TYPESCRIPT WITH NODE.JS

Microsoft maintains and develops TypeScript, a popular open-source language that is liked and used by many software engineers around the world. It's essentially a superset of JavaScript that extends the language's capabilities. Static type definitions, which are not available in simple JavaScript, are the most notable innovation. It is possible, for example, to describe what kind of parameters are expected and what is returned exactly in functions, as well as the specific structure of the object are produced. TypeScript is a fantastic technology that offers up a whole new universe of possibilities for JavaScript projects. It makes code more secure and robust by detecting many vulnerabilities before it is even released - it detects issues as code is written and works seamlessly with code editors like Visual Studio Code. For the sudoku solver front-end, typescript and node.js were used to host the webcam based sudoku solver on a local server and the gridlines and numbers on the screen were made possible due to typescript programming.

[Figure 37] is an image of how the sudoku solver user interface looks when using the webcam. It shows the Front-end hosted on a local server and sudoku image being scanned using the webcam of the device.

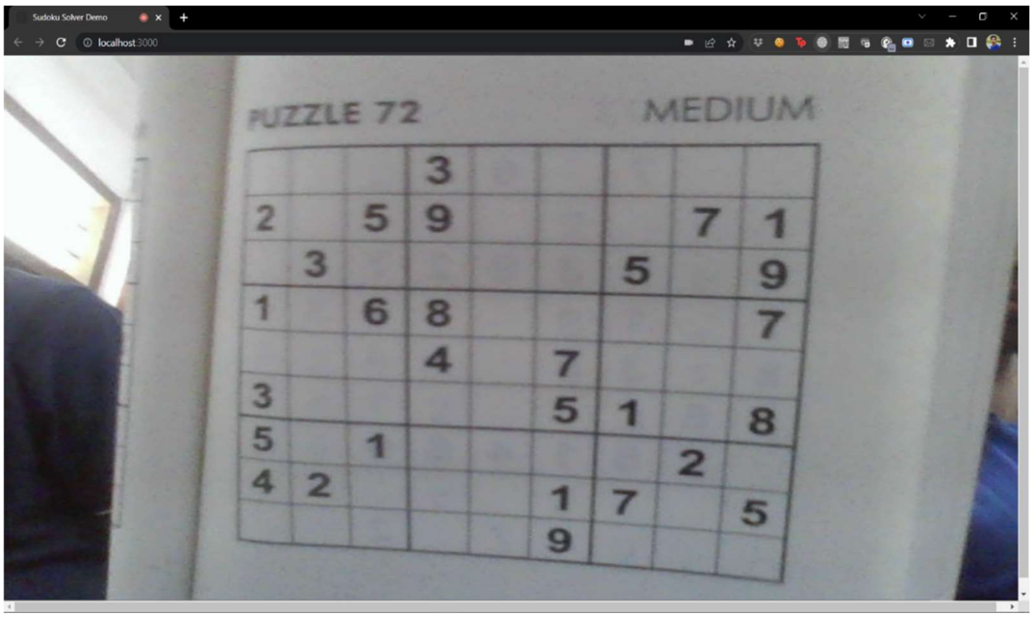
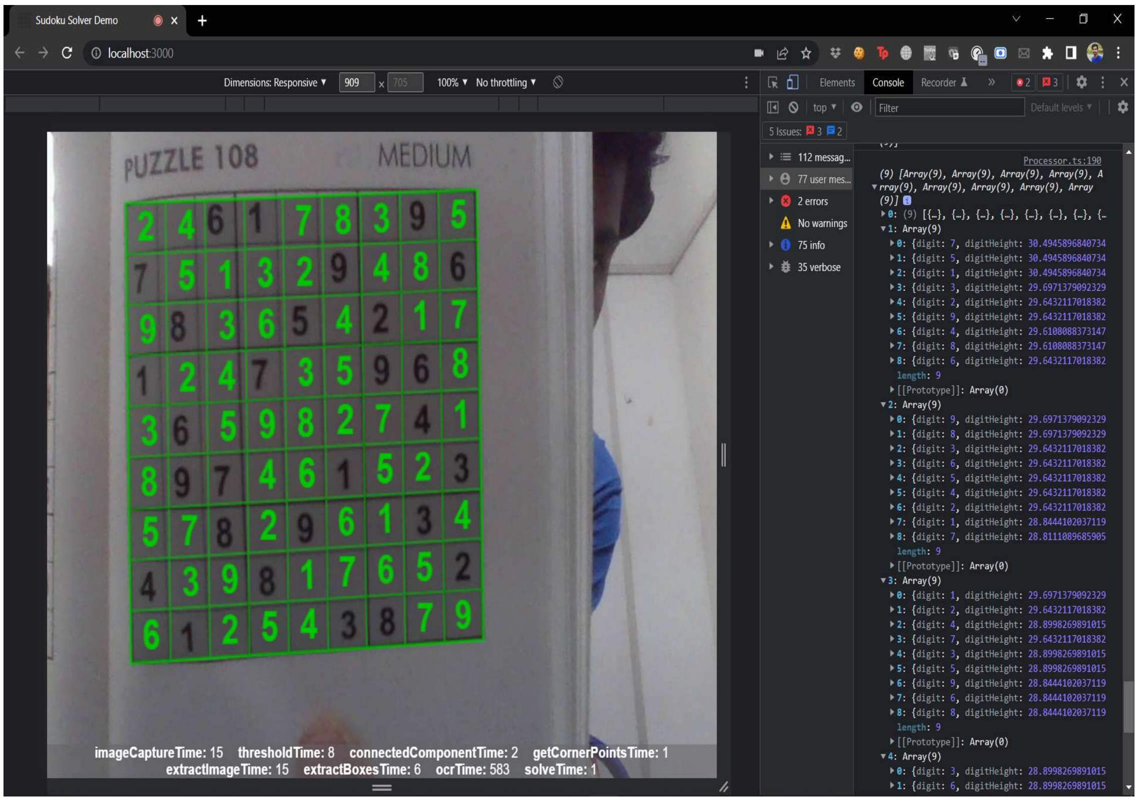


Figure 37: Sudoku Solver UI and Webcam

In order to access the solution stored in the array, the user can check the console messages in the webpage. [Figure 38] displays how the sudoku solution is stored in the matrix and how it can be accessed.



```
imageCaptureTime: 15   thresholdTime: 8   connectedComponentTime: 2   getCornerPointsTime: 1
extractImageTime: 15   extractBoxesTime: 6   ocrTime: 583   solveTime: 1
```

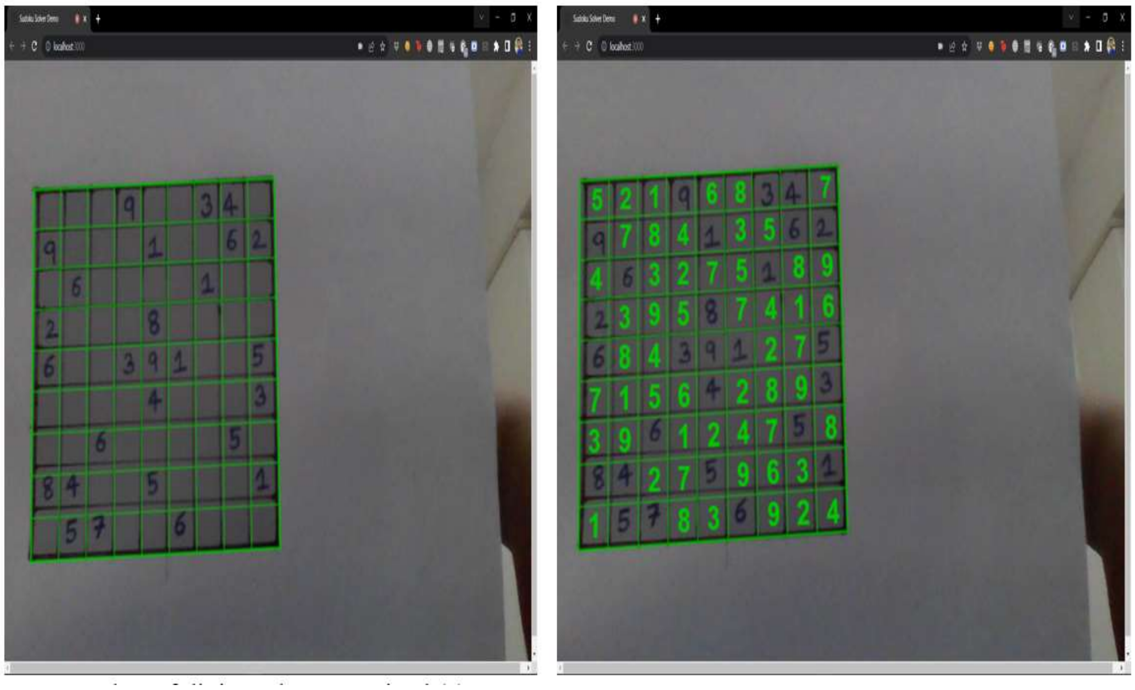
```
                                                   Processor.ts:190
(9) [Array(9), Array(9), Array(9), Array(9), A
▼rray(9), Array(9), Array(9), Array(9), Array
(9)]
 ▶0: (9) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…
 ▼1: Array(9)
   ▶0: {digit: 7, digitHeight: 30.4945896840734
   ▶1: {digit: 5, digitHeight: 30.4945896840734
   ▶2: {digit: 1, digitHeight: 30.4945896840734
   ▶3: {digit: 3, digitHeight: 29.6971379092329
   ▶4: {digit: 2, digitHeight: 29.6432117018382
   ▶5: {digit: 9, digitHeight: 29.6432117018382
   ▶6: {digit: 4, digitHeight: 29.6108088373147
   ▶7: {digit: 8, digitHeight: 29.6108088373147
   ▶8: {digit: 6, digitHeight: 29.6432117018382
    length: 9
   ▶[[Prototype]]: Array(0)
 ▼2: Array(9)
   ▶0: {digit: 9, digitHeight: 29.6971379092329
   ▶1: {digit: 8, digitHeight: 29.6971379092329
   ▶2: {digit: 3, digitHeight: 29.6432117018382
   ▶3: {digit: 6, digitHeight: 29.6432117018382
   ▶4: {digit: 5, digitHeight: 29.6432117018382
   ▶5: {digit: 4, digitHeight: 29.6432117018382
   ▶6: {digit: 2, digitHeight: 29.6432117018382
   ▶7: {digit: 1, digitHeight: 28.8444102037119
   ▶8: {digit: 7, digitHeight: 28.8111089685905
    length: 9
   ▶[[Prototype]]: Array(0)
 ▼3: Array(9)
   ▶0: {digit: 1, digitHeight: 29.6971379092329
   ▶1: {digit: 2, digitHeight: 29.6432117018382
   ▶2: {digit: 4, digitHeight: 28.8998269891015
   ▶3: {digit: 7, digitHeight: 29.6432117018382
   ▶4: {digit: 3, digitHeight: 28.8998269891015
   ▶5: {digit: 5, digitHeight: 28.8998269891015
   ▶6: {digit: 9, digitHeight: 28.8444102037119
   ▶7: {digit: 6, digitHeight: 28.8444102037119
   ▶8: {digit: 8, digitHeight: 28.8444102037119
    length: 9
   ▶[[Prototype]]: Array(0)
 ▼4: Array(9)
   ▶0: {digit: 3, digitHeight: 28.8998269891015
   ▶1: {digit: 6, digitHeight: 28.8998269891015
```

Figure 38: Displaying Sudoku Solution Matrix in Front-End

CHAPTER 5

RESULTS AND DISCUSSION

Now, that the sudoku puzzle has been solved using recursive backtracking algorithm, [Figure 39] and [Figure 40] below show the accuracy of the digits of two random sudoku puzzles.

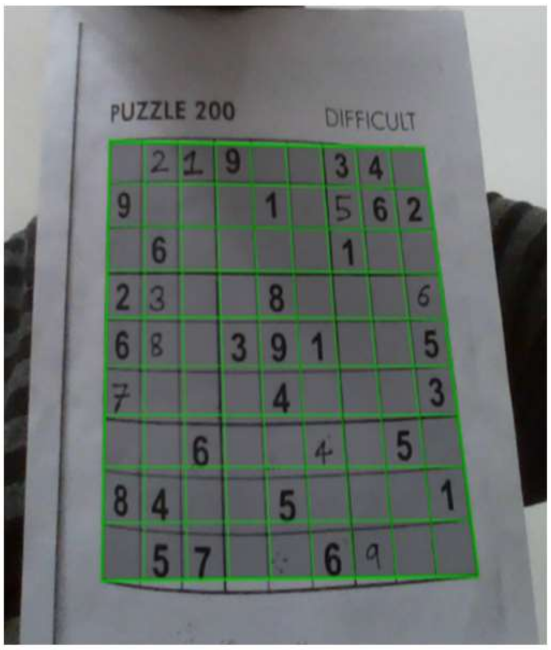


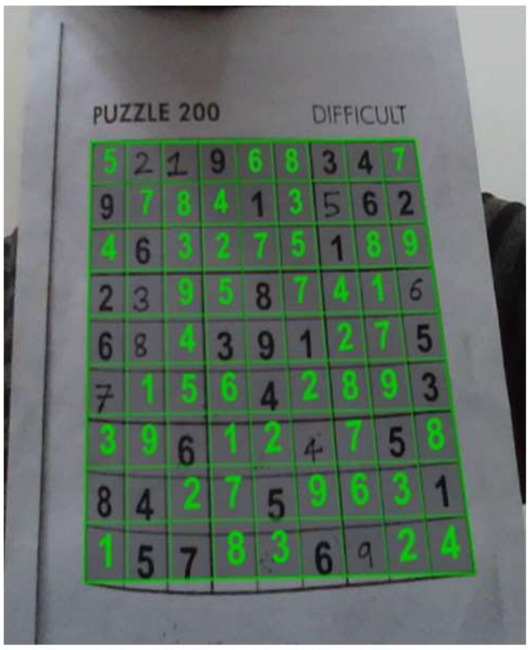
Number of digits to be recognized (a): 27

Number of digits recognized accurately (b): 27

Accuracy of current sudoku puzzle = (b/a) = 100

Figure 39: Sudoku Solver Digit Accuracy Image 1





Number of digits to be recognized (a): 36

Number of digits recognized accurately (b): 36

Accuracy of current sudoku puzzle = (b/a) = 100

Figure 40: Sudoku Solver Digit Accuracy Image 2

The overall accuracy of the sudoku solver has been tabulated in [Table 3] below.

CATEGORY OF TEST	ACCURACY OBTAINED
CNN OCR MODEL [USING MNIST DATASET]	95 %
SUDOKU IMAGES WITH PRINTED DIGITS	93 % (14 out of 15 Images successful)
PARTIALLY HANDWRITTEN SUDOKU IMAGES	90 % (9 out of 10 Images successful)
FULLY HANDWRITTEN SUDOKU IMAGES	80 % (4 out of 5 Images successful)
OVERALL ACCURACY	90 % (27 out of 30 Images successful)

Table 3: Accuracy of Sudoku Solver

Hence, the overall accuracy of the sudoku solver is 90 %.

If the image processing and OCR part is accurate, the accuracy of the recursive backtracking algorithm is extremely high. (~99%). Therefore, the accuracy of the sudoku solver can be further improved by improving the accuracy of the OCR and the image processing portions of the project.

5.1 COMPARISON WITH EXISTING WORK

When compared to this particular sudoku solver, most of the other existing work does not deal with handwritten digits or even partially solved sudoku puzzles. Instead, they only work with printed digits. This sudoku solver is able to deal with printed and handwritten digits as well as partially solved sudoku puzzles. Additionally, the image processing algorithms used in this project have been tuned to handle blurred images as well, which was a shortcoming of the existing projects.

The existing work does not have a diverse dataset, rather, they only have 20-30 images in the dataset. They also do not contain handwritten sudoku images in the dataset. But this project has a collective dataset of more than 300+ sudoku images, which allows the user to solve a diverse range of sudoku puzzles. Existing sudoku puzzles also do not explicitly classify sudoku images based on difficulty levels. The dataset in this project contains sudoku images of varying difficulties (Easy, Medium and Difficult).

Existing work also does not incorporate a front-end to test the sudoku images. This project also contains a front-end UI where the sudoku images can be scanned through a webcam and the solutions can be displayed live on the screen itself.

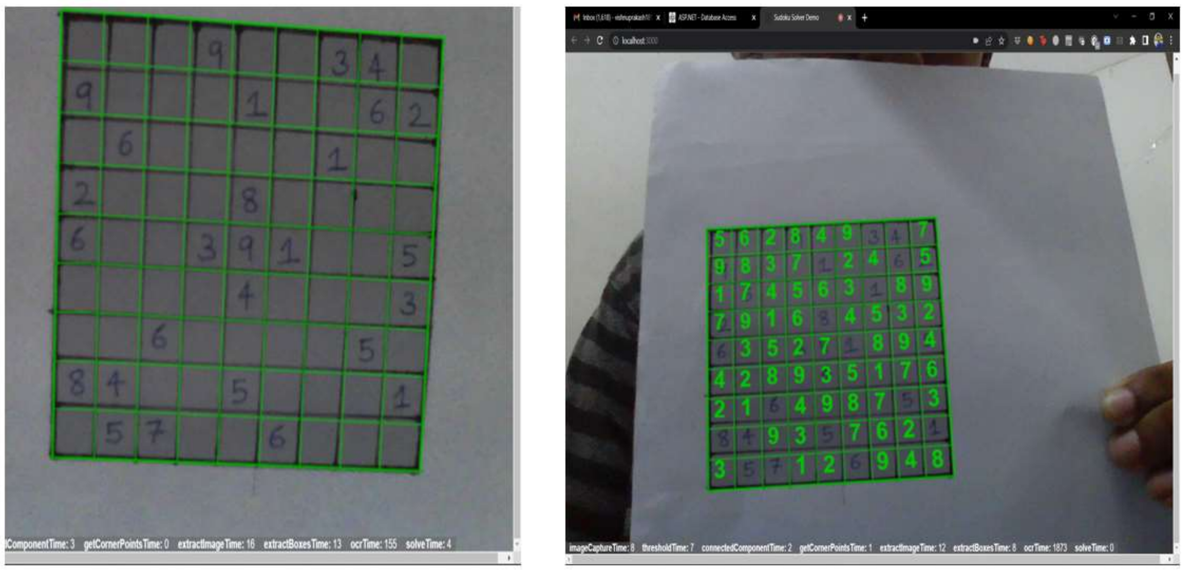
5.2 CHALLENGES AND LIMITATIONS

The current iteration of this sudoku solver, despite having an accuracy of 90 %, possesses a few limitations as well. There are still some issues with recognizing handwritten digits that are very thin. The OCR Model also sometimes tends to incorrectly recognize the handwritten digits.

Also, the image processing algorithms struggles sometimes to recognize the sudoku grid when the lighting conditions are poor, when the lighting is too bright, when the image is moving constantly or when there is too much noise in the image. The accuracy of the sudoku solver also depends on the quality of the webcam. If the webcam quality is low and has too much noise then the sudoku grid is not detected and it does not produce a solution.

Additionally, the front-end tends to lag a little bit as it very resource intensive, and sometimes takes more time to display the sudoku solution on the screen.

[Figure 41] below shows the inaccurate detection of digits of the sudoku grid due to poor quality of the webcam and the lack of thickness of the handwritten digits.



```
lComponentTime: 3   getCornerPointsTime: 0   extractImageTime: 16   extractBoxesTime: 13   ocrTime: 155   solveTime: 4
```

```
imageCaptureTime: 8   thresholdTime: 7   connectedComponentTime: 2   getCornerPointsTime: 1   extractImageTime: 12   extractBoxesTime: 8   ocrTime: 1873   solveTime: 0
```

Number of digits to be recognized (a): 27

Number of digits recognized accurately (b): 17

Accuracy of current sudoku puzzle = (b/a) = 63

Figure 41: Inaccurate Detection for Fully Handwritten Sudoku Puzzle

5.2.1 HOW TO OVERCOME THE CHALLENGES

To overcome the issue of the OCR model incorrectly detecting the handwritten digits, the OCR model can be trained with other datasets containing handwritten and printed fonts to increase the accuracy of the model.

To overcome the challenge of webcam noise and poor webcam quality, a higher resolution external webcam can be used in order to capture higher quality sudoku images and hence accurately capture the sudoku grid and display the solution.

Some other lightweight frameworks such as Vue.js can be used to make the process less resource intensive in order to overcome the issue of lag in the front-end, along with using more powerful machines to execute the server-side code.

CHAPTER 6

CONCLUSION

A sudoku solver that is capable of handling sudoku images in various lighting conditions, angles and various levels of blurriness has been developed. The program is able to work with printed digits, handwritten digits, and also with partially solved sudoku images. The sudoku solver can also handle different difficulties of sudoku puzzles (Easy, Medium and Difficult) and find the accurate solutions for each of them. It also contains a front-end user interface in which the sudoku image is solved in real time using a webcam and the solution is displayed on the screen.

A diverse dataset of more than 300+ pre-existing and custom sudoku images was collected. The sudoku images were processed using Gaussian Blurring and Adaptive Thresholding techniques. The processed images were run through an OCR model using CNN and trained on the MNIST dataset to obtain the digits and store in matrix format. The front-end UI was developed using Node.js and Typescript and hosted on a local server. Finally, the sudoku puzzle was solved using Recursive Backtracking Algorithm, which gave an overall accuracy of 90 percent.

CHAPTER 7

FUTURE WORK TO BE DONE

The current iteration of the sudoku solver has an accuracy of 90 %. The accuracy and overall user experience of the project can be improved in the following ways:

- In order to improve the image processing, other techniques which work on various lighting conditions, various angles of the sudoku images and blurry sudoku images can be incorporated or the currently employed techniques can be fine-tuned. Instead of manually increasing the exposure of dark images, the process can be automated through code.
- To improve the accuracy of the OCR Model and improve the ability to detect handwritten digits, the thickness of the handwritten digits can be increased automatically through image processing techniques.
- The sudoku solver can automate the process of verifying the solution generated by the algorithm and providing the accuracy so that the user can verify if the solution provided by the program is correct or not.
- To improve the accuracy of the OCR Model and improve the ability to detect handwritten digits, the diversity of the dataset can be increased by including more images of sudoku puzzles. Different algorithms can also be used to solve the sudoku puzzles and the speed and accuracy of those algorithms can be compared to select the most optimal one.
- The end-user experience can be further improved by automatically taking a screenshot of the solution after capturing the sudoku puzzle through the webcam. Alternatively, a matrix of the solved sudoku puzzle can be displayed after solving the puzzle.
- The front-end, which is currently only accessible through a local server, can be hosted on a web server, so that it can be accessed on the internet by anyone. Finally, mobile applications in iOS and Android can be developed for the sudoku solver so that users can solve sudoku puzzles on the go using their smartphone camera.

REFERENCES

[1] 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", 2015 Third International Conference on Image Information Processing (ICIIP), 2015, pp. 179-184, doi: 10.1109/ICIIP.2015.7414762.

[2] Mrs. Supriya H.S, Ayush Baran, Abhishek Nipun, Kanhaiya Kumar and Parul Prabhat, “Processing Sudoku Images and Solving by Recursive Backtracking Algorithm”, 2017 *IJSRD - International Journal for Scientific Research & Development| Vol. 5, Issue 04, 2017 | ISSN (online): 2321-0613*

[3] N.Musliu and F. Winter, "A Hybrid Approach for the Sudoku Problem: Using Constraint Programming in Iterated Local Search", 2017 IEEE Intelligent Systems, vol. 32, no. 02, pp. 52-62, 2017.doi:10.1109/MIS.2017.29

[4] Sang C. Suh*, Aghalya Dharshni Manmatharaj, “An Efficient Machine Learning Technique to Classify and Recognize Handwritten and Printed Digits of Sudoku Puzzle”, 2019 (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 10, No. 6, 2019 637

[5] K. Gupta, S. Khatri and M. H. Khan, "A Novel Automated Solver for Sudoku Images", 2019 *IEEE 5th International Conference for Convergence in Technology (I2CT)*, 2019, pp. 1-6, doi: 10.1109/I2CT45611.2019.9033860.

[6] K. S. Vamsi, S. Gangadharabhotla and V. S. Harsha Sai, "A Deep Learning approach to solve sudoku puzzle", 2021 *5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2021, pp. 1175-1179, doi: 10.1109/ICICCS51141.2021.9432326

[7] Sunanda Jana, Anamika Dey, Arnab Kumar Maji, Rajat Kumar Pal “A novel hybrid genetic algorithm-based firefly mating algorithm for solving Sudoku”, 2021 Journal: Innovations in Systems and Software Engineering > Issue 3/2021, Source: Springer, 2021