Solving Real Time Sudoku Images Using Different Computer Vision Techniques With Deep Learning

Rajat[1], Rishabh Sharma[2]
[1, 2] Maharaja Agrasen Institute of technology

Abstract- *Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.*

This project aims at developing a highly effective computer vision based Sudoku solving program that identifies, recognises and understands an image to generate and solve a Sudoku. In this project user have to click/upload photo of sudoku. In this project user have to click/upload photo of sudoku. The project will automatically detect the problem and extract sudoku and interpret the value and solve the problem for you and print the output. The underlying principle is the use of object detection by finding the largest contour using adaptive threshold and Image Recognition through neural networks and finally solving the problem using search algorithms like Backtracking.We have also shown that problem specific dataset is better than using general dataset, that is, MNIST.

I. INTRODUCTION

Machine learning has been used for years to offer object detection, image recognition, spam detection, natural language processing, recommendation system and medical diagnosis. Today, machine learning algorithms can help us enhance cybersecurity, ensure public safety, and improve medical outcomes. Machine learning systems can also make customer service better and automobiles safer. With machine learning, we can solve a real-world problem but would not be too complicated to implement. Machine Learning is very much popular in gaming field also like Alpha-go which defeated the national GO champion was starting of a new era.
Sudoku is one of the most popular numbers game that we can play everyday. Sudoku is a numbers game which players have to fill each of the blank boxes in a puzzle based on specific rules. It can be one of the hardest and challenging puzzle to solve which all age group people love and help them level up their brain. Just like physical exercise, your mind need some exercise too. Playing sudoku actually exercises your brain extensively. So machine learning enable computers to do things which human can do with superhuman accuracy.

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

Completed games are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine 3×3 subregions of the 9x9 playing board.

II. RULES OF SUDOKU

A general Sudoku has a 9 X 9 array of square boxes, of which some are pre-filled and others are empty. The scattered random presence and absence of numbers in the array makes the puzzle. The difficulty of the Sudoku is measured by the number of pre-provided elements at the start of the game. A Sudoku containing more numbers is considered easier than the one having less numbers. A Sudoku is considered to be complete only when all the 81 boxes are completely filled while following the rule that no row or column and no 3 x 3 sub-block repeats of the digit from 1 to 9.

III. DATASET USED

Custom Sudoku Digit Data:

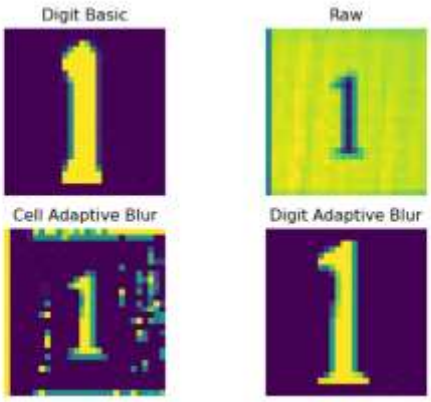


Fig.: Sample dataset and Labels

For this task we have created a digit dataset using different sudoku images. In the dataset of 130 Sudoku Images we had randomly split the image archive into training and test sets, with 80 images chosen for the training set and rest 50 for the testing set. From this we have created archives of individual digits, classified them using the .dat files and separated them into their own folders (`/data/images/classified/0-9). Finally compiles these images into a binary pickle file that can be consumed for training and testing purpose. We have used 4 different types of digit extraction method to create our dataset and find out the best method among them. Dataset is consisting of (28,28) size gray scale digits which is similar to a very popular handwritten digit classification dataset MNIST. We have used small image size in order to reduce the computational complexity. Then we have flattened all the images so that a single row with 784 pixels will represent unique image.

There are total 6500 digits in the training dataset which are classified into 10 categories from 0-9 where 0 is for all those digits which are not extracted completely and 1-9 is used to represent the number. So the size of the label array will become (6500,10) which is the one hot vector represent of 10 categories and size of the training image is (6500,784). We have also created a test for validating our model using 1 sudoku image. The test contains 4000 unique digits which are classified similarly as training dataset.

IV. IMPLEMENTATION

The custom image has to be identified from the extra surroundings of the image and then all the numbers need to be recognised so that the algorithm can solve them. This was done in the following ways.

Fig.: Sample Input Sudoku

Preprocessing

Image preprocessing is done because we are working on a real time based image and the quality of image depends on the various factors like resolution, brightness, sharpness. Real time image contains noise that can affect the working and accuracy of the model so we need to apply some preprocessing methods

The following are key words that can be used for implementing different processing methods and hence recognition models. Many of these key words can be used in

- Basic: Simplifies heuristics for extracting the Sudoku board out of the image. This speeds up processing but reduces reliability. This excludes the following heuristics:
    - Excluding the border width of the grid when extracting a board.
    - Using a second algorithm for board extraction in certain cases.
    - Using Hough lines and linear algebra when estimating the grid.
    - Adaptive thresholding applied on the whole board instead of on a cell-by-cell basis.
    - Using Sudoku rules to determine failures in recognition and compensating.
- Cell: Applies a pre-processing mode to each cell but does not perform digit extraction and scaling.
- Digit: This implements an extractor algorithm on each cell that attempts to get the digit, isolate it and scale it. Should be used with a pre-processing mode that converts the image to binary (either pure black or white).
- Raw: This uses the full heuristics for board extraction and grid estimation but performs no pre-processing on the individual cells.
- Preprocessing applied on the input image are:

- **Converting image into grayscale image** :

Converting the image to grayscale helps in easy and fast reading of the image. With grayscale image the colour information is lost and each element simply be recognised by the change in brightness of the pixels.

- **Gaussian blur is applied** :

The raw image obtained as input is full of noise and difficult to read for any computer. So the image is blurred using Gaussian Blur and the parts of the image get smoothened out for easy reading. Gaussian Blur reduced the complexity of the image significantly and helped in retaining only those elements which we were interested in.

- **Adaptive threshold** :

Adaptive Thresholding converts the grayscale image to a binary image with just two colors black & white. This binary image was used to make the number and line recognition clear by a significant number.

- **Dilation** :

The image obtained by Adaptive Thresholding is very clear to identify by human eyes but due to the multiple modifications of the image the numeral and line thickness gets affected and hence the resultant image gives poor results in number detection. So dilation was used to gain back the lost image thickness. Dilation just makes the lines and shapes more prominent to read and understand by computer hence improving the accuracy.

Sudoku detection is done by finding the contours and since we assume in the image the most portion is covered by the puzzle so the largest contour will give the sudoku.

- **Finding all the contours in the image** :

The raw image which was grayscaled before is now ready to get recognised. We assume that the image obtained by user has Sudoku as it's prominent feature and based on this assumption the contour starts to recognise the Sudoku.

- **Get the largest contour in the image**.

The assumption of Sudoku being the main part of the image helps in finding the outline of the Sudoku easily. This detection of the Sudoku is used to find the main outline of the Sudoku image. At this stage we are enabling computer to recognise the main 9x9 matrix of the Sudoku.

- **Get the largest bounding rectangle within the contour** :

With the outline of the Sudoku getting detected by the algorithm we can start discarding the extra parts of the image which play no role in sudoku. We find the corners of the contour rectangle and convert those points to the X,Y coordinates of the image pixels.

- **Fitting the image in the center using perspective** :

Now the image is cut out of the dilated image by using the formula given in figure 4 and applying it to the coordinates obtained by largest contour detection. The image so obtained is the final Sudoku image without any noise or extra parts.

Digit isolation is done through various steps:

- **Probabilistic Hough Line Transform to find the location of all grid lines** :

Cells of the image are detected by simple probability method as the image obtained after extra part cutting and noise reduction is almost straight and simply dividing the image into 9x9 squares can help easily detect the Sudoku cells.

- **Extract the largest connected component in the image** :

Now, the position of the pre available numbers is detected by observing the pixel brightness pattern in each of the small contours of the grid. The cells with varying pixel brightness are treated to be filled with data while the rest are treated to be empty. This process is made more efficient by giving priority to the center part of each contour.

- **Removing all major noise in the cell** :

All the major contours of the image are detected and a new 9x9 grid is created showing the position of probable numbers and all the empty cells are treated with digit zero "0" in the new grid.

New 9 by 9 grid is created with the image cell detected and shifting the digit towards the center and with the cell size 28 X 28.

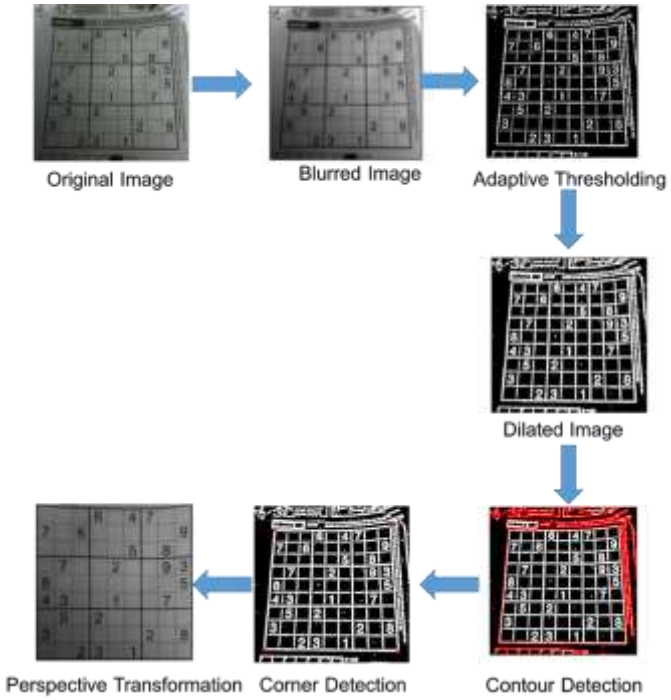


Fig.: Preprocessing Steps

Detecting , interpretation

Now the main task was to detect the digit and then predict what it is. For this we trained our deep neural net using Convolution Neural Net on our custom Sudoku image dataset which is a collection of Sudoku digits of size (28,28).

Model designing

Builds a multi-layered convolutional neural network for classifying digits drawn in 28x28 squares. Network contains 5 layers:

- Input layer: 784 neurons
- 1st Convolution layer, 5x5 patches of a 28x28 image with 32 outputs. 2x2 max pool downsamples to 14x14 image.
- 2nd Convolutional layer, 5x5 patches of a 14x14 image with 64 outputs. 2x2 max pool downsamples to 7x7 image.
- Fully connected layer: 1024 neurons
- Output layer: 10 neurons
-

```
w_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1, 28, 28, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, w_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
w_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, w_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
w_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, w_fc1) + b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
w_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y = tf.matmul(h_fc1_drop, w_fc2) + b_fc2

return y, keep_prob
```

Fig.: Final Model

Solving Sudoku

Backtracking

Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, that incrementally builds candidates to the solutions, and abandons each partial candidate (“backtracks”) as soon as it determines that the candidate cannot possibly be completed to a valid solution.

The idea is that we can build a solution step by step using recursion; if during the process we realise that is not going to be a valid solution, then we stop computing that solution and we return back to the step before (backtrack). In thecase of the maze, when we are in a dead-end we are forced to backtrack, but there are other cases in which we could realise that we’re heading to a non-valid (or not good) solution before having reached it.

Modification of Backtracking using Elimination

Backtracking solves the Sudoku by solving each and every possibility one by one. But practically it can take a very long time to solve the Sudoku this way. So we modify it a bit by simply eliminating all the useless solution before handedly. Simple smart elimination removes all the inappropriate options to give us a simple and faster backtracking algorithm. All the possible options to the blank spaces are laid out and a space with least possibilities is chosen as the starting point of the backtracking algorithm.

Rules for easy and then extra rules for hard Sudoku

Implementing Backtracking algorithm directly can solve most of the puzzles easily. But to save time and computational power we set some rules that helped us solve the Sudoku even better.

"If a unit has only one possible place for a given digit, it must be that digit."

This rule is applicable on all the Sudoku sets and was used in the whole algorithm. But to solve some hard Sudokus we needed to add some extra rules which helped in making the algorithm better. These rules are

If there are no remaining possibilities for a cell, the board is invalid.

If there are no possible positions for a digit in a unit, the board is invalid.

These rules helped in additional fast pacing of the program and made it better. For testing of the above rules a Sudoku intentionally made difficult for Brute-Force implementation was selected. The application of these rules solved a worst case puzzle in just 0.01 seconds.

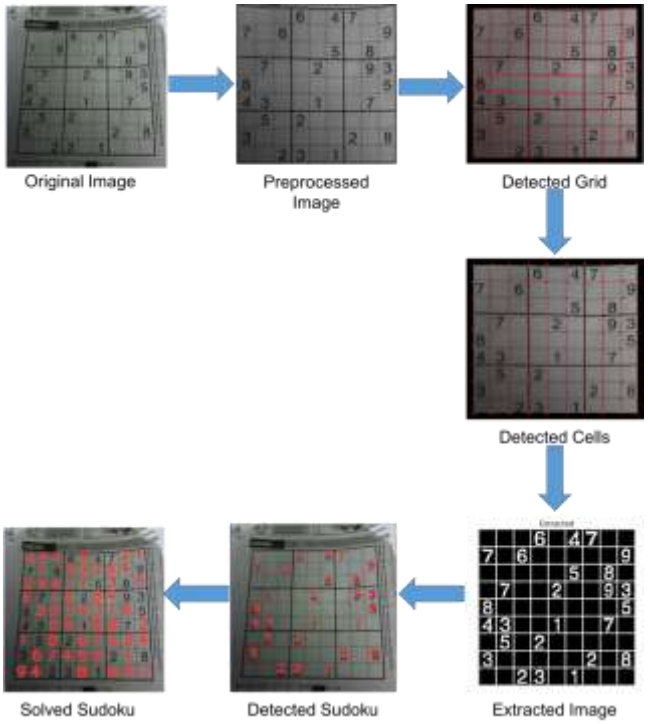


Fig.: Major Steps of the Algorithm

Result

In the given project we have 3 major task to solve in order to have a complete solution.

1) Extract the Sudoku from the given Image
2) Interpretation of the extracted Sudoku
3) Solving the sudoku

Task 1:

So we have implemented various preprocessing techniques in order to extract the correct sudoku from the given image.

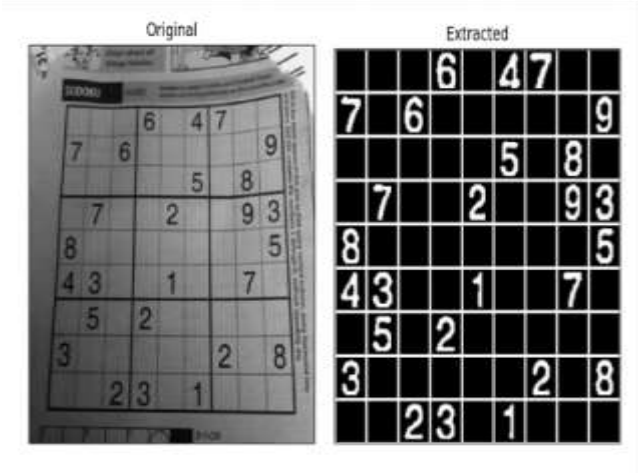


Fig.: Extraction Result

Task 2:

Now after extracting the sudoku from the image we have used our trained Convolutional Neural Net model in order to predict all the digits and creating the custom sudoku grid which will be used to solve the sudoku.

Fig.: Interpretation of the extracted sudoku

Task 3:

Now after interpreting the sudoku we have solved it using Brute force technique known as Backtracking

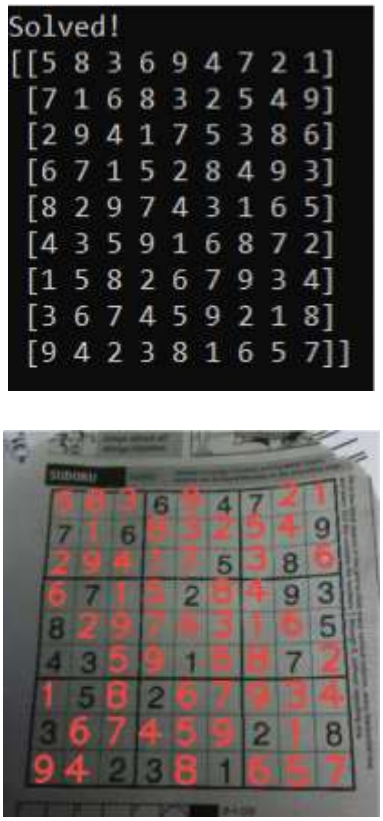


Fig.: Solved Sudoku

Enhancement in our project

We improve our techniques in order to solve sudoku more efficiently and to solve images which contain more noise and are of low resolution or highly rotated.

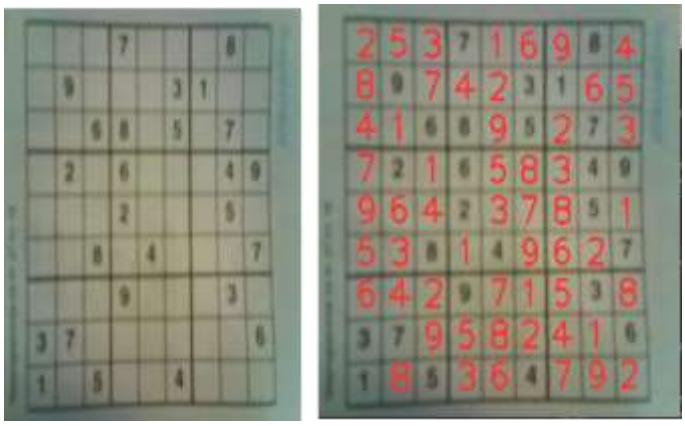


Fig.: Low resolution and blur Sudoku image

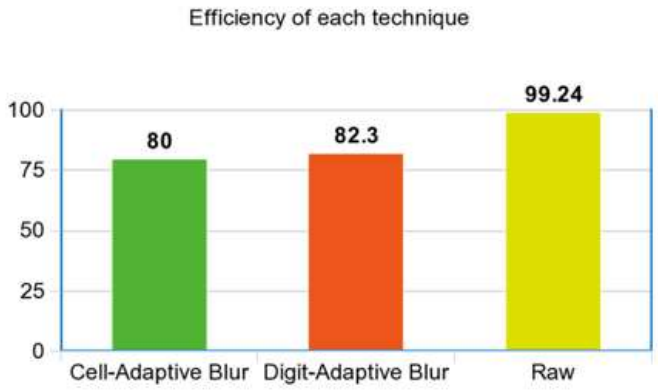
Modifications in dataset

The performance of the neural net model to predict the digit will increase with increase in the size of the dataset. So now we are using 10k images of digits which are extracted from 130 different sudoku so with increase in dataset by including more images and digit from different Fonts help us to improve the accuracy of digit detection which can be used to predict digits which are not completely extracted.

Modification in the preprocessing Technique

We improve the preprocessing technique so that all the digits are extracted properly and complete which will be used to predict the digit.

Image with low resolution, blurred and highly rotated are extracted properly due to which interpretation of the digit is not done correctly. We can improve our technique by adding few more preprocessing steps like using of Repetitive thresholding technique which find the optimal threshold and block value for adaptive threshold function.



V. CONCLUSION

From the result obtained we can say that preprocessing is required for working with real time images and we have tried different computer vision techniques for solving Sudoku and found out that Raw method works the best after sufficient amount of training. In Raw model we got 129 correctly solved Sudoku out of 130 giving us an accuracy of 99.24% .

REFERENCES

[1] Investigation on the Effect of a Gaussian Blur in Image Filtering and Segmentation by Estevão S. Gedraite, Murielle Hadad https://www.researchgate.net/publication/261278360_Investigation_on_the_effect_of_a_Gaussian_Blur_in_image_filtering_and_segmentation

[2] Gaussian Blurring-Deblurring for Improved Image Compression Moi Hoon Yap1, Michel Bister2, Hong Tat Ewe11Multimedia University (MMU)

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.375.9085&rep=rep1&type=pdf

[3] Adaptive Thresholding: A comparative study by Payel Roy, Saurab Dutta, Nilanjan Dey, Goutami Dey, Sayan Chakraborty, Ruben Ray https://www.researchgate.net/publication/269984781_Adaptive_thresholding_A_comparative_study

[4] Adaptive Thresholding Using the Integral Image by Derek Bradley - Carleton University, Canada and Gerhard Roth - National Research Council of Canada http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.7883&rep=rep1&type=pdf

[5] Contour Detection and Hierarchical Image Segmentation by Pablo Arbela´ez,Michael Maire, Charless Fowlkes and Jitendra Malik. https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/amfm_pami2010.pdf

[6] A CONVENTIONAL STUDY OF EDGE DETECTION TECHNIQUE IN DIGITAL IMAGE PROCESSING by Indrajeet Kumar, Jyoti Rawat, Dr. H.S. Bhadauria https://www.ijcsmc.com/docs/papers/April2014/V3I4201497.pdf

[7] Developing the Theory of Perspective Transformation: Continuity, Intersubjectivity, and Emancipatory Praxis

[8] https://journals.sagepub.com/doi/pdf/10.1177/0741713616674076

[9] Progressive Probabilistic Hough Transform by J. Matas, C.Galambos and J. Kittler https://arxiv.org/pdf/1502.02160.pdf

[10] A Survey on Hough Transform, Theory, Techniques and Applications by Allam Shehata Hassanein , Sherien Mohammad, Mohamed Sameer, and Mohammad Ehab Ragab https://arxiv.org/pdf/1502.02160.pdf