Solving Sudoku Puzzles using Image Processing and Machine Learning

Hrhishikesh Thombare
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Mumbai, India
github link : https://github.com/Hrhishikesh/sudoku-solver

Abstract
Sudoku is a popular number-based puzzle game that requires logical thinking and problem-solving skills. In this report, an approach to solve Sudoku puzzles is proposed. The proposed method consists of two main steps: *Image Processing* and *Machine Learning*. In the first step,image processing techniques are used to extract the Sudoku puzzle grid from a given image. Afterwards, the image was split into 81 boxes thereby extracting image of each digit. Then, a CNN model was used to detect digits from each box. The CNN model showed an accuracy of **98.25** on the validation data. Once the digits were recognized, the puzzle was solved using backtracking algorithm. The solution was then overlaid on top of the source image.

Contents

1 Introduction

Sudoku puzzles have gained immense popularity over the years as a fun and challenging way to test one's logic and problem-solving skills. The rules of Sudoku are simple: fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub-grids contain all of the digits from 1 to 9 without any repetition. In recent years, there has been a growing interest in developing automated techniques for solving Sudoku puzzles using computer vision and machine learning.

This project explores the use of image processing and machine learning to solve Sudoku puzzles with an end-to-end pipeline that takes an image of a Sudoku puzzle as input and outputs the solution, all while overlaying the output on top of the original image. The system utilizes various Image Processing Techniques to extract the Sudoku grid from the image and individual cells from the grid, and a Convolutional Neural Network is used to solve the puzzle. The result is a system that can solve Sudoku puzzles in a matter of seconds, providing a fast and efficient way to tackle this classic logic problem..

2 Pre-requisites

The code is divided into different sections for easier understanding, with that said, having intermediate knowledge of python would be beneficial for understanding and analysing the code. Beyond Basic ML, one should be well-versed in different types of layers used in a Neural Network. Two main python libraries were used in this project : open-cv and tensorflow, familiarity with both would be highly beneficial.

3 Datasets

Various Datasets were tested with, namely :

- MNIST Handwritten Digits Dataset
- EMNIST
- USPS
- SVHN

But all these datasets had a problem that the models trained on them don't translate well into detecting digits on a Sudoku grid as there's a lot of noise in each of the extracted cells or boxes as seen in Figure.1.

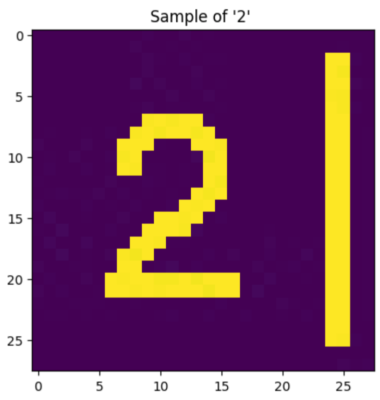


Figure 1: A sample of image '2'

Due to such noise the accuracy of these models in recognizing the Sudoku Digits decreases which is the most essential part of the models performance. The models performance on the training data can be good, but after testing on real sudoku images the model's performace was subpar.

So 'Printed Digit Dataset' was used.It Contains around 3000 images of digitally printed numeric dataset. Each image is of dimension 28x28 and is grayscale. This dataset was purposely created for sudoku digits classification hence it shows blank image for 0 (zeros).

4 Procedure

4.1 Image Processing

4.1.1 Detecting Contours

Contours are used for extracting the Sudoku image from the raw image.First the image is converted to grayscale. Then for detecting contours, first gaussian blur[6] was used on the image along with thresholding[7] with the output being as shown in Figure 2. This the essential step for detecting contours.[5] Once the contuours are detected, the sudoku image can be extracted by using the biggest external contour.

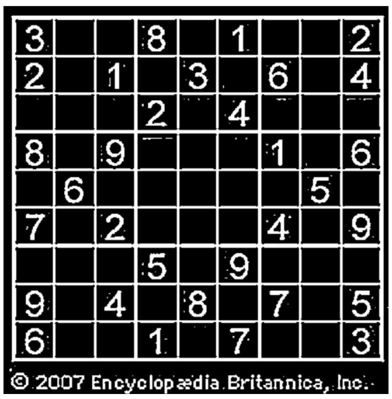


Figure 2: Thresholding on a sample sudoku image

4.1.2 Extracting Individual Cells

There are a total of 81 cells to be extracted, which will then be sent to the model to detect digits. As the image is an NxN dimensional array, it was split into 9 rows and columns using horizontal split and vertical split[8] thereby creating 81 different boxes.

4.2 Training the Model

The Final Model had the following layers :

- 2x Convolutional Layers
- 2x MaxPooling Layers
- 2x Dense Layers (relu)
- 1x Dense Output Layer (softmax)

A dropout layer was added after the final max-pooling layer, to prevent overfitting. Early Callback was also implemented to prevent over-fitting the data, as the data size was low. Before the dense layer, the Flatten layer was implemented to flatten the array to a single dimension for input in Dense Layers[1]. 'Adam' optimizer was used along with 'Categorical Cross Entropy' as a loss function. The model gave an accuracy of 98.25% on the validation dataset(Figure 3).The model was then Serialized and saved for future use.

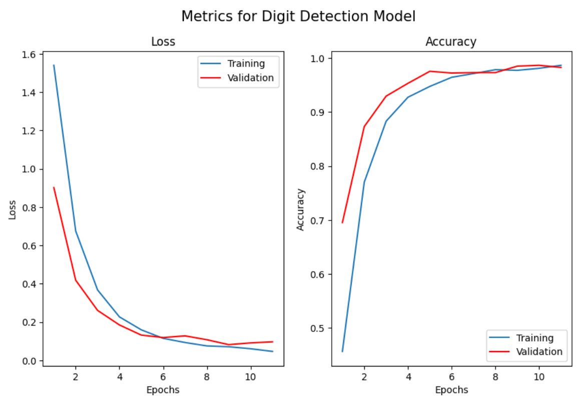


Figure 3: Metrics of the model

4.3 Preprocessing Individual Cells and Detecting Digits

As the extracted cells and the images the model was trained on were different, the cells needed to be preprocessed first before sending for detection. The images were reshaped into (1,28,28,1) dimensions each. Later 'Binary threshold'[7] was added along with 'bitwise not'[4] to get the image into the same format as figure 1. Figure 4 and 5 show the initial and final images respectively.

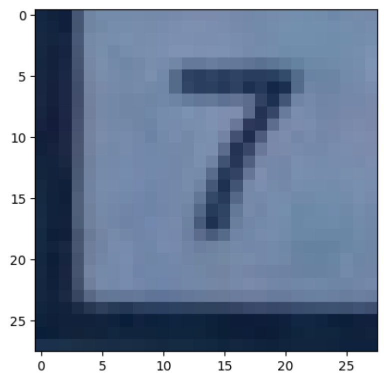


Figure 4: Before Processing

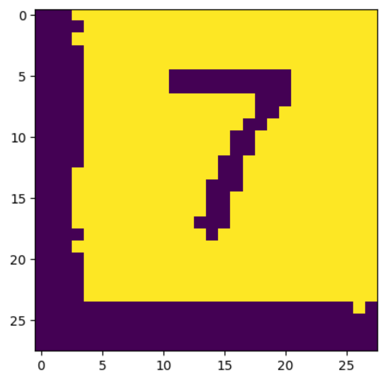


Figure 5: After Processing

The Images were then scaled and individually sent to the model for Digit Detection. The digits were predicted from the results using argmax function from numpy[10].The final result can be seen in figure 6.

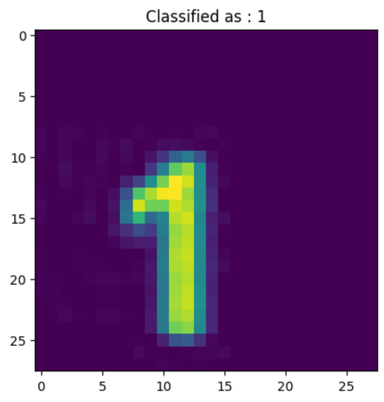


Figure 6: Detected Image

4.4 Solving the puzzle

As the digits were detected, a 9x9 board(array) was created with those digits. There are now two methods to approach the problem :

- Simple Brute Force
- Backtracking Algorithm

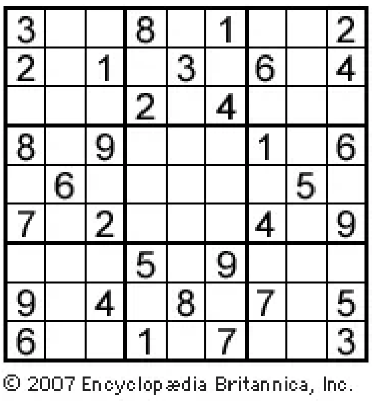
Simple Brute Forcing may work to solve the board but has a disadvantage of requiring more computational time. Hence, a variation of Brute Force was used in solving called Backtracking Algorithm[11]. Backtracking is a depth-first search (in contrast to a breadth-first search), because it will completely explore one branch to a possible solution before moving to another branch. The Advantages of Backtracking are:

- Simple to implement
- Guaranteed Solution (provided a valid Sudoku)[11]
- Solving Time mostly unrelated to Degree of Difficulty

A separate python solver file was created for solving the board by implementing Backtracking Algorithm[9].The solved board was then sent back to main file for final processing.

4.4.1 Overlaying the results on Source Image

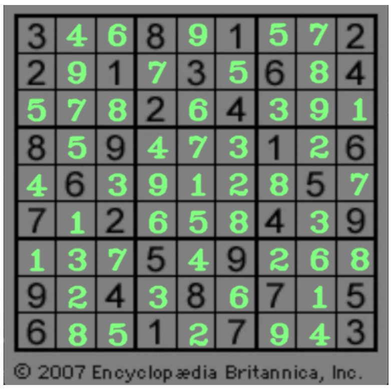
A single image is created just containing the solved digits with identical height, width and perspective as the original image. The new image is then overlaid on top of the original image. The initial and final result is as shown in Figures 7 and 8 respectively.



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

© 2007 Encyclopædia Britannica, Inc.

Figure 7: Original Image



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

© 2007 Encyclopædia Britannica, Inc.

Figure 8: Solved Image

5 Conclusion

In conclusion, the application of image processing and machine learning in solving Sudoku puzzles has demonstrated a promising avenue for automating the traditionally manual tasks. By leveraging the capabilities of image processing to extract and recognize the Sudoku grid from an image and utilizing machine learning algorithms to analyze and solve the puzzle, the accuracy and efficiency of the system can be significantly improved.Moreover, the methods and techniques presented in the report can be of practical use to puzzle enthusiasts and developers of puzzle-solving software.

6 Future Work and Improvements

- The accuracy of the model can be further improved by more augmentation of the pre-existing data or by using a different more dynamic dataset
- The model can be deployed on a web-based application by using Flask[3] or on an Android Device using Buildozer[2]
- The Sudoku puzzle once detected can be solved using more efficient Algorithms[11]

References

[1] Analytics vidhya (handwritten digits recognition). https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/.

[2] Denicz (buildozer deployment). https://youtu.be/pzsvN3fuBA0.

[3] Free code camp (cloud deployment using flask). https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/

[4] Geeks for geeks (arithematic operations). https://www.geeksforgeeks.org/arithmetic-operations-on-images-using-opencv-set-2-bitwise-operations-on-binary-ima

[5] Opencv documentation (contours). https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html.

[6] Opencv documentation (smoothing). https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[7] Opencv documentation (thresholding). https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html.

[8] Stack-overflow (splitting array). https://stackoverflow.com/questions/44782476/split-a-numpy-array-both-horizontally-and-vertically.

[9] Tech with tim (sudoku solver using backtracking). https://youtu.be/eqUwSA0xI-s.

[10] Towards data science (mnist digit recognition from scratch). https://towardsdatascience.com/mnist-handwritten-digits-classification-from-scratch-using-p

[11] Wikipedia (sudoku solving algorithms). https://en.wikipedia.org/wiki/Sudoku_solving_algorithms.