



Sudoku Solver using Computer Vision and Image Processing techniques

**JOSHIKA S 19PD16
HARIHARAN S 19PD27**



Introduction

A Sudoku is a logic-based puzzle that usually comes in the form of a 9x9 grid and 3x3 sub-grids of 1 to 9 digits. The condition to have a valid solution to this puzzle is that no digit is used twice in any row, column, or 3x3 sub-grid.

The number of possible 9x9 grids is 6.67×10^{21} so finding a solution can sometimes be challenging depending on the initial puzzle.

Detecting a Sudoku board from an image is the first step in building a Sudoku solver using computer vision techniques.



Approach

The approach is into 3 parts :-

1. Looking for sudoku in the image.
2. Extracting numbers from sudoku.
3. Solving the Sudoku using backtracking.

Part 1 - Looking for Sudoku in the Image



input image

Image processing

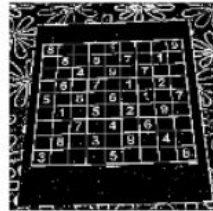


Output image

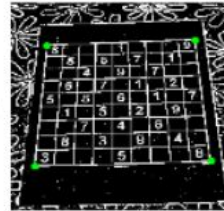
Extract the sudoku grid i.e. our Region of Interest (ROI) from the input image.



input image



Binary image



Corners of the largest polygon highlighted



Joining the corners to get the outer grid



Performing crop and warp

Steps involved :-

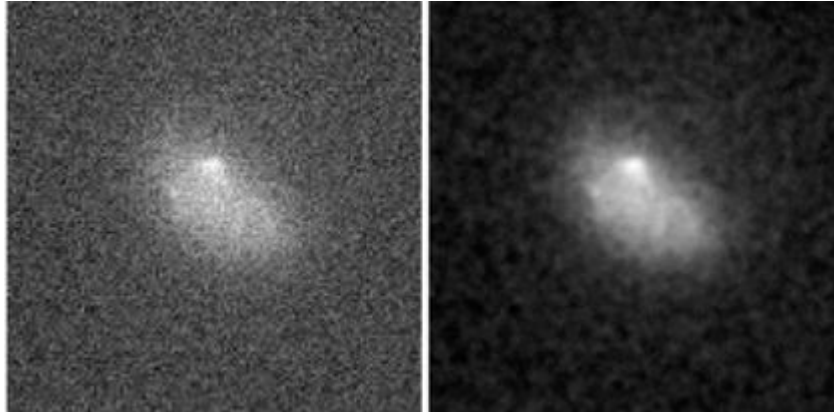
1. Converting the input image to a binary image: This step involves converting the input image to a grayscale image, applying gaussian blur to the grey-scale image, followed by thresholding, bitwise_not and dilating the image.

Gray-scale: Colored digital images are nothing but 3-dimensional arrays containing pixel data in the form of numbers. Ideally there are 3 channels: RGB (Red, Green, & Blue) and when we convert a colored image to grayscale, we just keep a single channel instead of 3 and this channel has a value ranging from 0–255 where 0 corresponds to a black pixel and 255 corresponds to a white pixel and remaining values decide how gray the pixel is (the closer to 0, the darker it is).



Gaussian Blur:

An image blurring technique in which the image is convolved with a Gaussian kernel. I'll be using it to smoothen the input image by reducing noise.

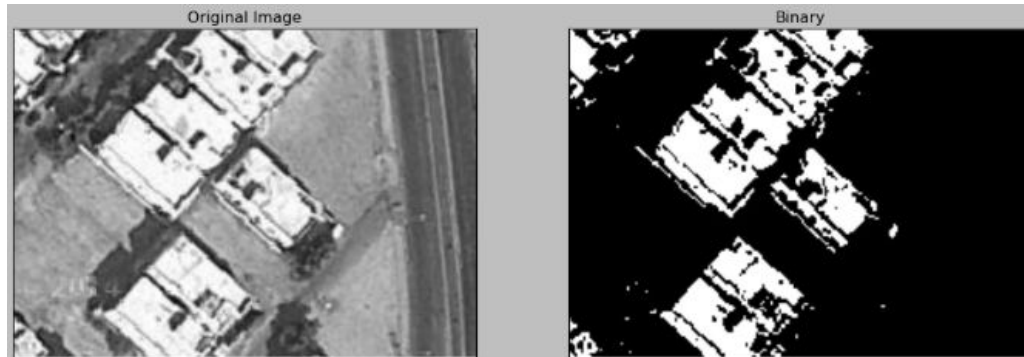


Thresholding:

In grayscale images, each pixel has a value between 0–255, to convert such an image to a binary image we apply thresholding.

To do this, we choose a threshold value between 0–255 and check each pixel's value in the grayscale image. If the value is less than the threshold, it is given a value of 0 else 1.

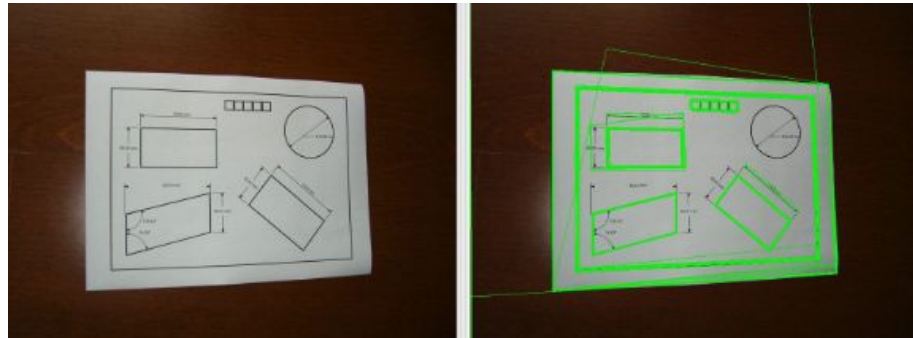
In binary images, the pixels have just 2 values (0: Black, 1:White) and hence, it makes edge detection much easier.



Detecting the largest polygon in the image:

This step involves finding the contours from the previous image and selecting the largest contour (corresponds to the largest grid). Now from the largest contour, selecting the extreme most points and that would be the 4 corners. After sorting the returned contours from the image by area, the largest contour can easily be selected. And once we have the largest polygon, we can easily select the 4 corners (displayed as the 4 green points in the third image in figure 1).

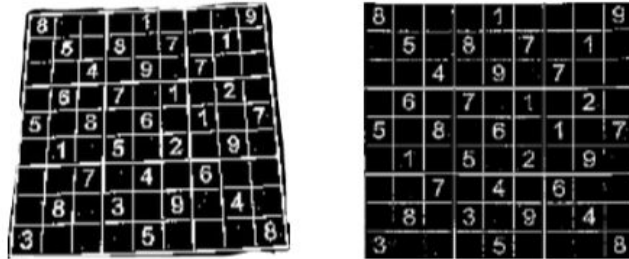
- Contour detection: Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity.



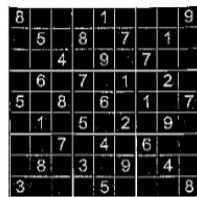
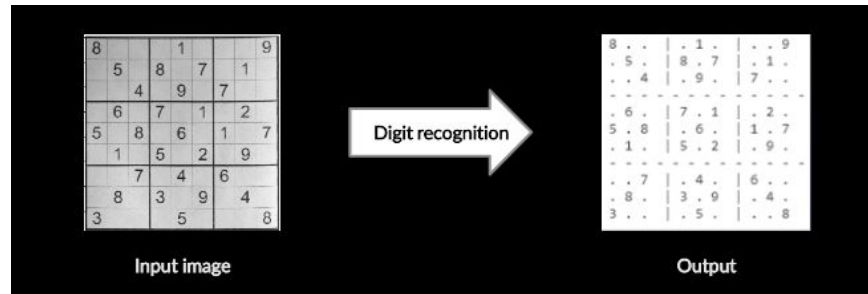
Cropping and warping the detected polygon

The approach is simple since we have the coordinates of the 4 corners, we can use this to crop and wrap the original image resulting in the final image in

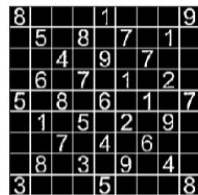
`cv2.warpPerspective` function takes 3 arguments source image (src), transformation matrix (m), and size of the destination image (size, size). transformation matrix. Now transformation matrix is a 3x3 matrix that helps us calculate the perspective transform. We can get the transformation matrix using `cv2.getPerspectiveTransform(source_image, destination_image)`.



Part 2 - (Extracting numbers from sudoku)



Converted
binary image



Noise free
binary image



Highlighting inner
grids



Sample pixels from
some of the grids




Step 1-

1. **Eliminating noise from the binary image:** We'll use a gaussian blur to remove the noise from the warped image resulting in a better image.
2. **Extracting the smaller grids:** Since we already know that a sudoku grid has 81 similar dimensional cells (9 rows and 9 columns). We can simply iterate over the row length and column length of the image, check for points every $1/9$ th of the length of row/column distance apart and stored the set of pixels that were in between in an array. Some of the sample pixels extracted are shown in the last image in figure 2.



Step 2-

1. Since we now have the individual grids that contain the digit information, we need to build a neural network that is capable of recognize the digits.
2. Data: We should expect 10 types of characters, 1–9 and null i.e. blank space that is needed to be filled. I've created the data using the above-mentioned grid extraction method from different sudoku images. 30 samples for each digit.
3. CNN: Create a convolutional neural network model .

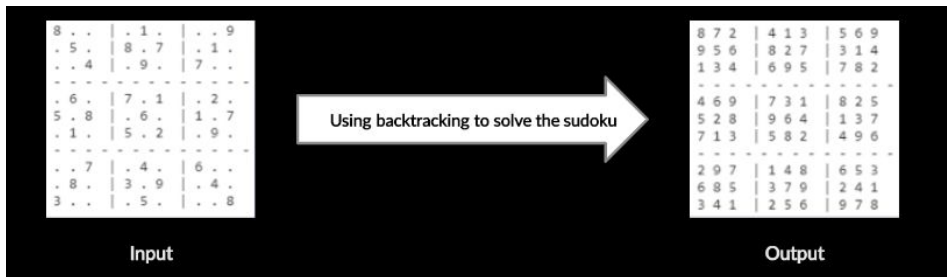
- 
- Now the task is to extract each number from the image, identify the number and save it into a 2D matrix. For digit recognition, we will be training neural network over MNIST dataset containing 60,000 images of digits from 0 to 9.
 - Reshape the images to be samples*pixels*width*height and normalize inputs from 0–255 to 0–1. After this, we will one hot encode the outputs.
 - Next, we will create a model to predict the handwritten digit.
 - After creating the model, let's compile it and fit over the dataset and evaluate it.
 - Now, we will test the above model created. Load the pre-trained model and weights.
 - It takes an image of digit and predicts the digit in the image.


Part 3 (Solving the Sudoku using backtracking):

Now that we have converted the image into an array, we just need to make a function that can fill up the blank spaces effectively given the rules of sudoku. I found backtracking very efficient at this so let's talk more about it.

Backtracking:

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point.





We will use backtracking to solve the Sudoku. This method allows us to step-by-step build candidate solutions in a tree-like shape and then prune this tree if we find out that a sub-tree cannot yield a feasible solution. The way we will do it in the case of Sudoku is as follows :

- For each cell, we compute the possible values that can be used to fill it given the state of the grid. We can do this very easily by elimination.
- We sort the cells by their number of possible values, from lowest to greatest.
- We go through the first unfilled cell and assign it one of its possible values, then to the next one and so on ...
- if we end up with a feasible solution we return it, else we go back to the last cell we assigned a value to and change its state to another possible value.



THANK YOU