Sudoku Solver

Pulkit Agarwal

July 2021

1 Overall Architecture

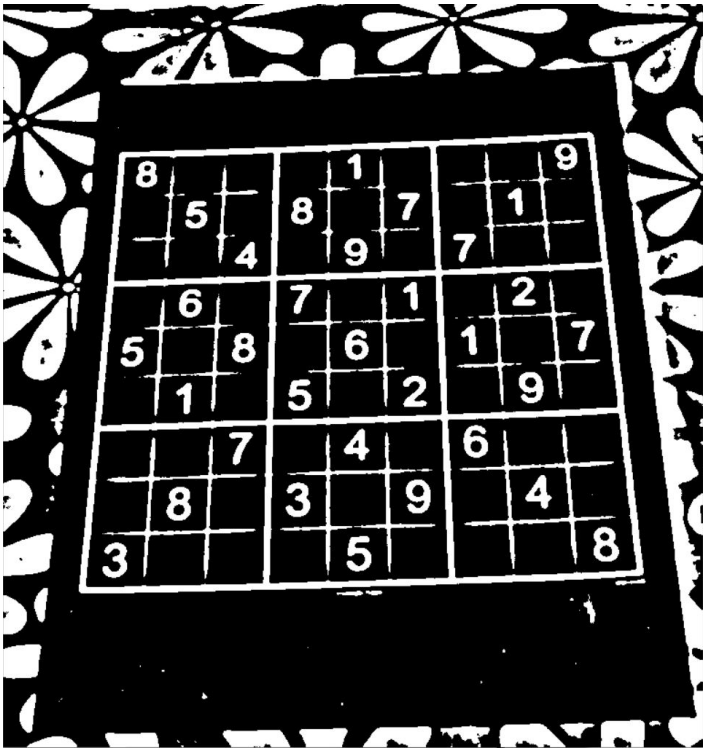
The overall code has been divided into various stages, with the most important stages and their outputs listed below:

1.1 Puzzle Extraction using Contours

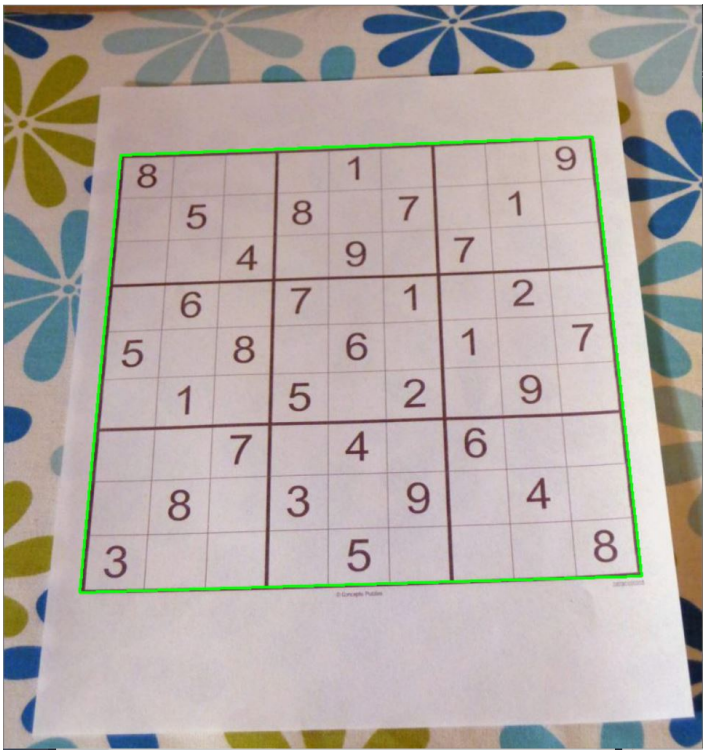
The first few stages involve pre-processing the sudoku puzzle, by using the following steps-

- Converting BGR image to Grayscale.
- Taking Gaussian Blur with kernel size $(9, 9)$.
- Applying Adaptive Threshold by computing weighted Gaussian mean over an area of 67×67, and a threshold $c = 4$, with these values set specifically for sudoku image 2. Here the final cells of sudoku were found to be of size $(53, 68)$ and on trying values in these range, 67 gave the best result.
- Taking bitwise not of the image.

Here is the image obtained after the above process was done on the second sample-

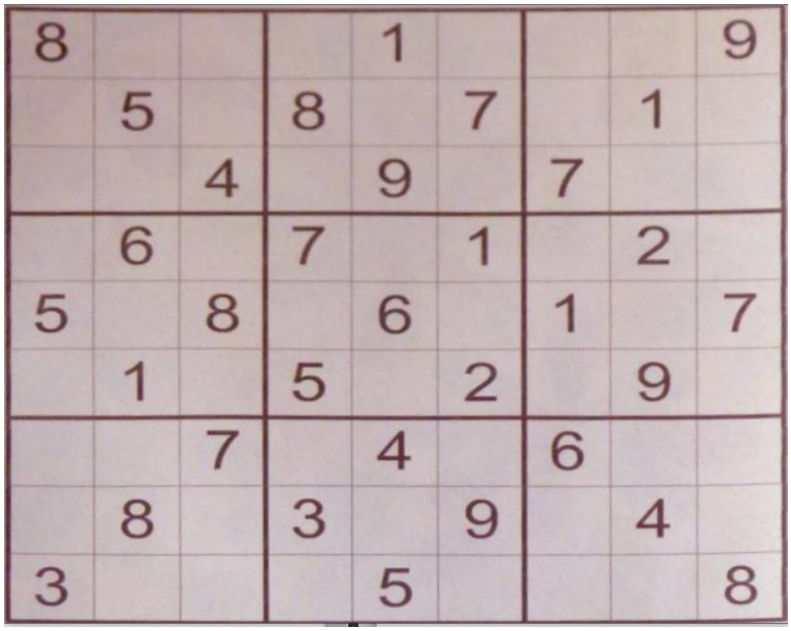


This was followed by the third stage, in which we found the contours, and sorted them in decreasing order by area. The largest area of a contour with length 4 was considered to be the outline of the sudoku puzzle. Here is the output obtained after the contours were identified (drawn on the original image)-



1.2 Perspective Transform

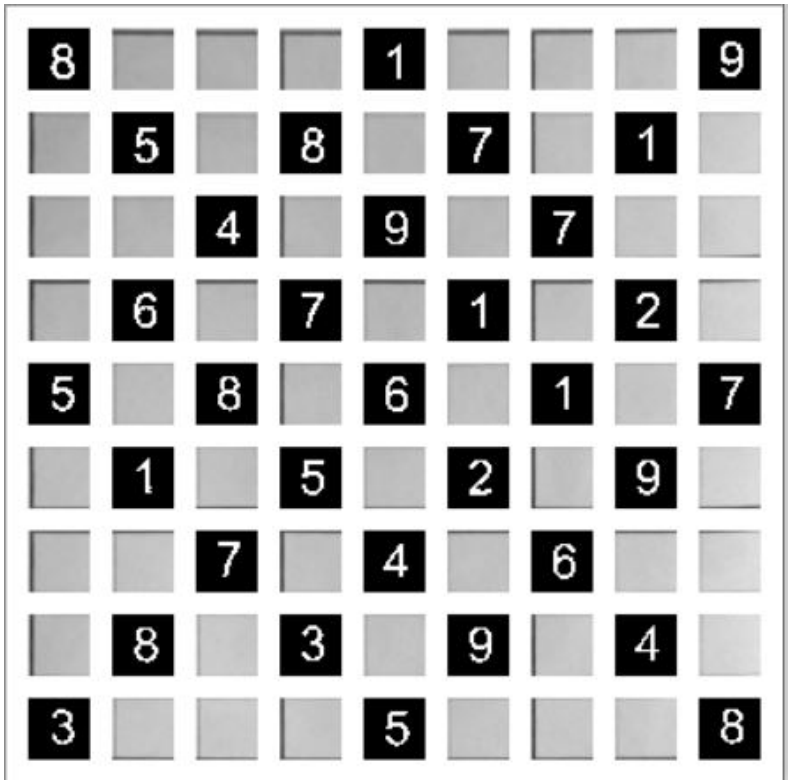
The contours only gave the outline of the sudoku, band so now it was extracted with the help of a perspective transform. Here we found the height and the width of the sudoku puzzle, and used the corners to take a warped perspective transform. However, this flipped the sudoku puzzle, so we flipped it bask to get the following image-



1.3 Digit extraction

Now we split the above image into 9×9 cells, and processed these cells to get the final digits. Here we took threshold for each such cell, and used the clear_border function from skimage module to clear out the cell borders. Then we again found contours in this image, and took the contour with the max area and drew it on an empty mask.

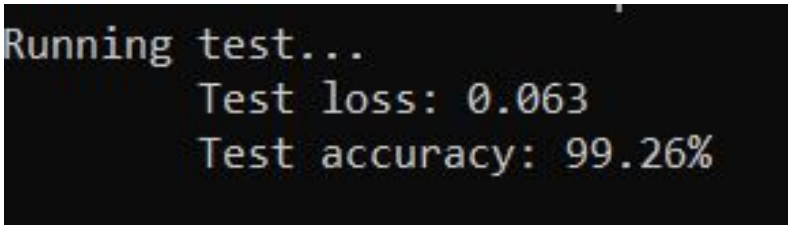
Using a threshold value of 0.03, we checked if this mask has more than 3% non zero pixels, in which case we claimed that a digit is present in that cell. Now we took a bitwise and with the original thresholded cell, and returned this cell for digit detection. Here is what our sudoku looked like after this step-



Here only those cells have been changed with the processed image which contain a digit.

1.4 Digit Detection

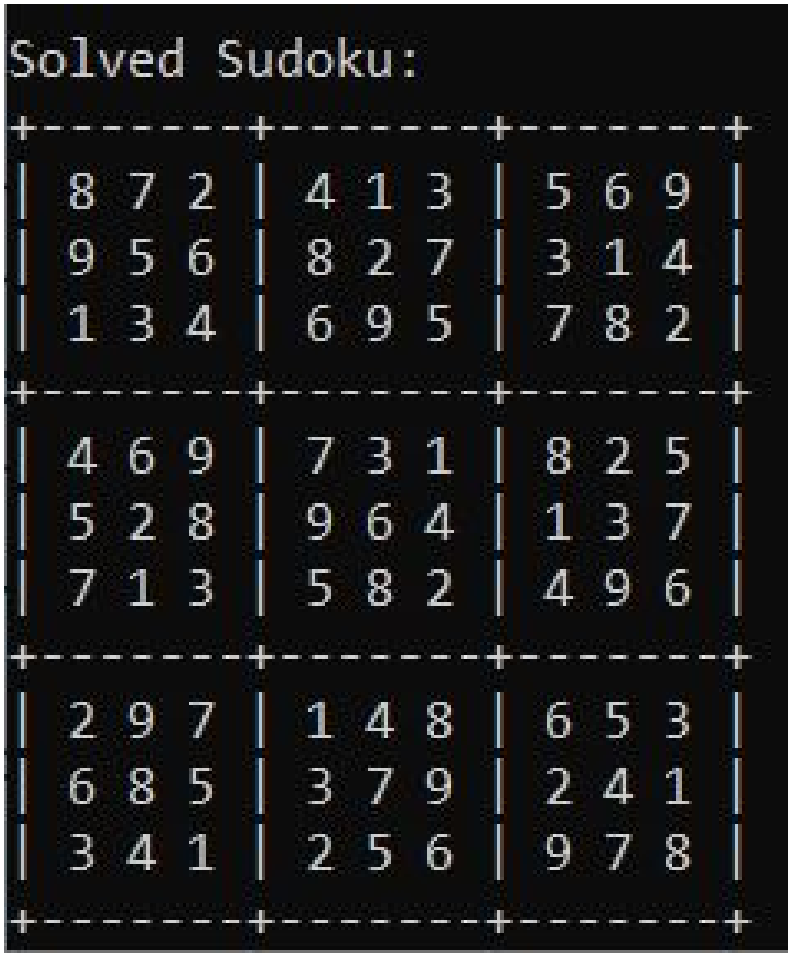
Now we used our keras model trained on the MNIST dataset to get what the actual digit in the cells are. The model is only called for cells which we determined as having a digit in the previous stage. Also, the keras model was changed significantly from the last time to get a higher accuracy, by using a 3 layer CNN model, followed by 2 fully connected layers. This gave a stupendous accuracy of 99.26% on the test set after training for 10 epochs.



After applying this model, and tweaking the parameters in the pre-processing stage of the sudoku, I was able to obtain correct results on all cells except for 1, namely the last cell in the middle row. Here the model predicted a 3 instead of 7, and changing any of the parameters only made it worse. Here is the result of what the model predicted-

2 Solving Sudoku using Backtracing

After hardcoding the correction for the recognised sudoku, the sudoku solver was implemented. This used a simple backtracing argument (probably not the fastest way to do this, but definitely the most obvious one). Here we simply checked for cells that are yet to be filled, found all cells that are not in their row or column, and then tried a recursive argument to check if we can find a possible sudoku with that cell filled by some valid integer. In case, the sudoku reached a dead end at some point, then we simply backtraced our way to the last possible moment and continue the recursion. Here is what the sudoku solver got for the corrected sample-



3 Challenges Faced

The major challenge was identifying whether the cell contains the digit, for which I had to fiddle a lot with the parameters in the pre-processing stage.

Unfortunately, I was not able to implement the same for sudoku 1. The main issue was the visibility of the cells. The clear_border function did not work properly in this case as it removed the cell content also. Another major problem was that just dividing the sudoku puzzle found after taking perspective into 9×9 cells was not the best possible method here, as the sudoku was not divided properly, and so the cells had a lot of disturbance.