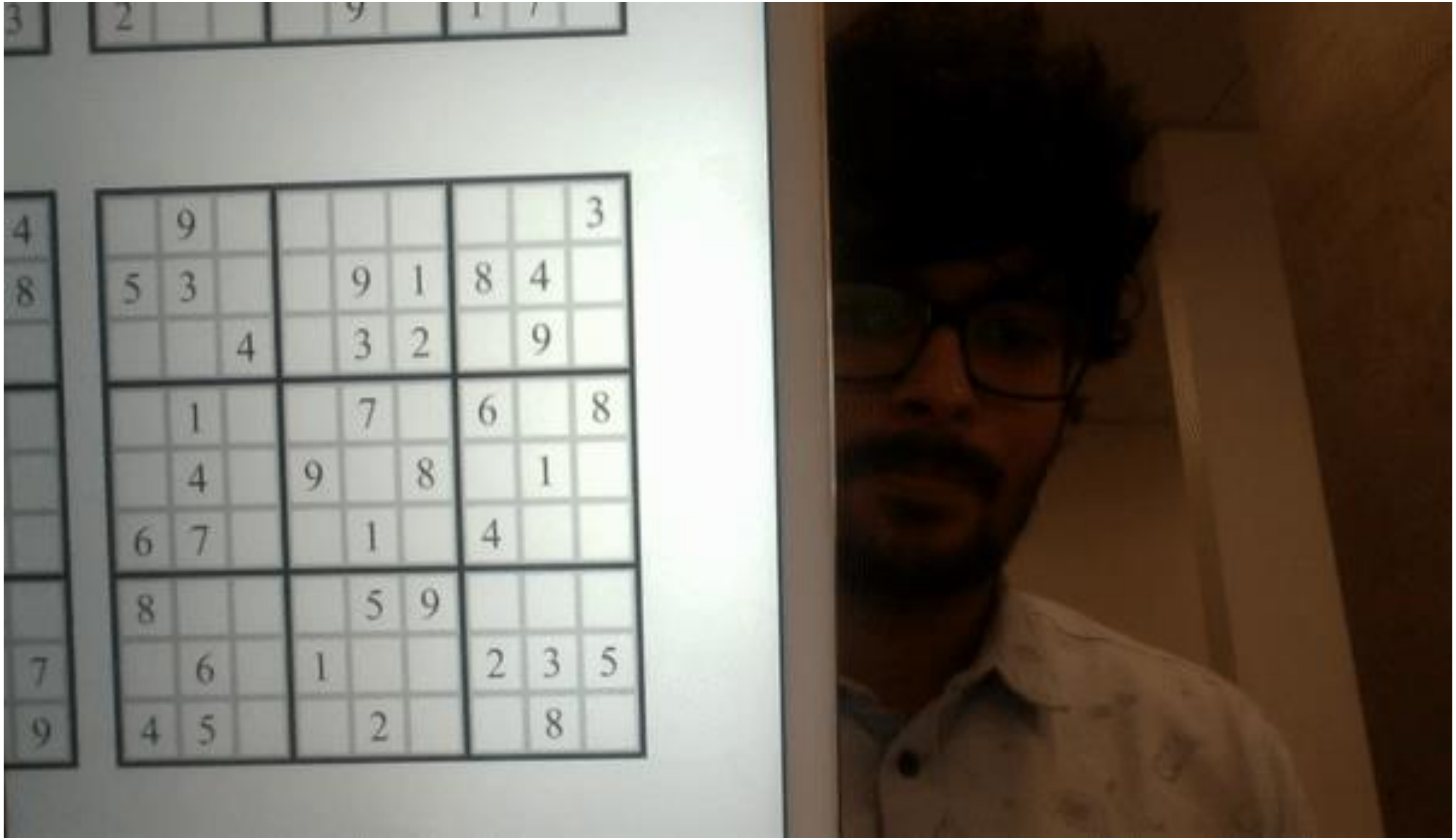
Webcam Sudoku Detection

Using OpenCV

Shivam Thakur

GOAL:



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

Motivation

- With the emergence of Virtual reality and Augmented reality in the field of Computer vision and Image processing, the possibilities are endless. Here, I wanted to show basic Image processing steps to read Sudoku image from a running webcam and the computer solving it by understanding the visual context in front of its "Eyes".

Steps Involved

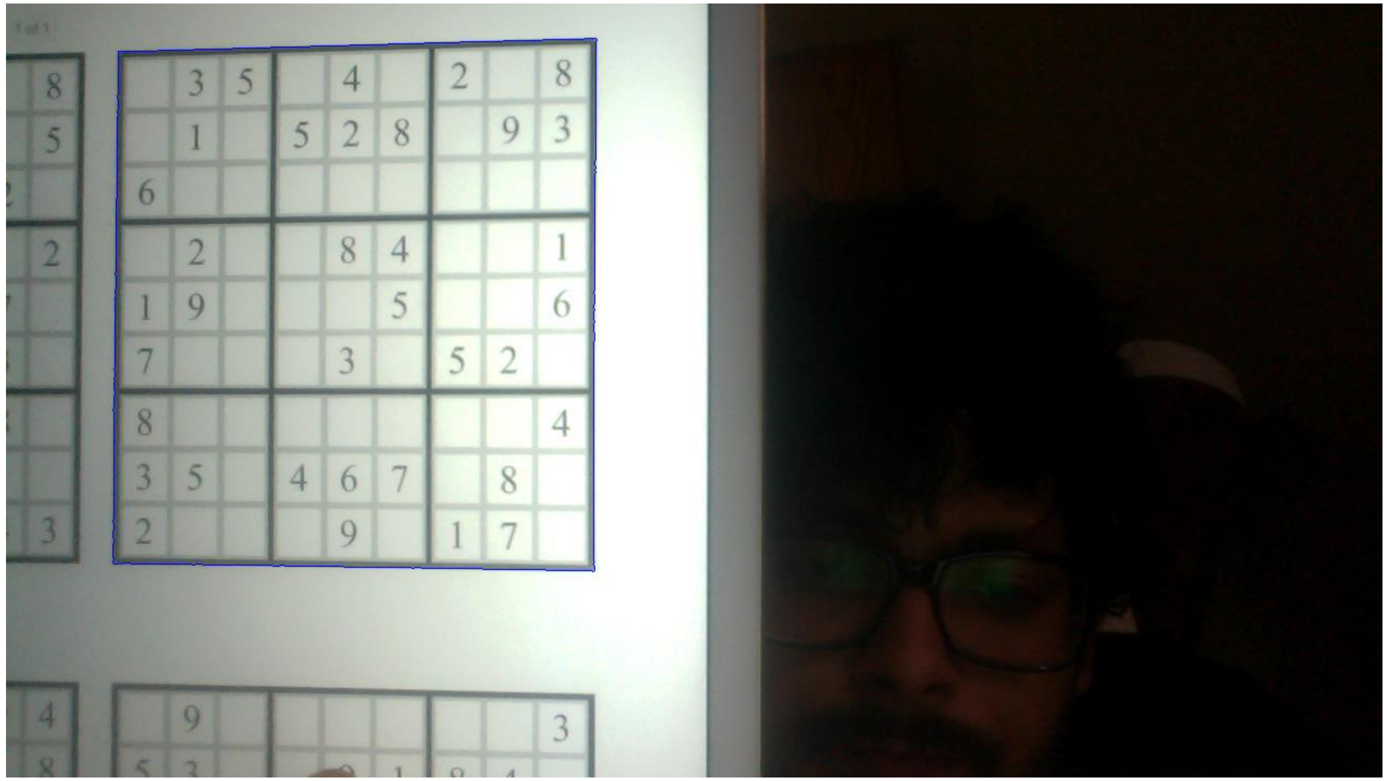
- Separating sudoku from the image.
- Extracting digits and recognizing them.
- Solving sudoku at backend.
- Showing the solved sudoku on screen overlapping the numbers on extracted sudoku grid.

Final Step

- Finally the sudoku is solved at backend using the solver.py using a simple backtracking search algorithm, we overlap the solved digits onto the initial image.

Step 1

The first step involves extracting the sudoku section from the frame(image) itself.

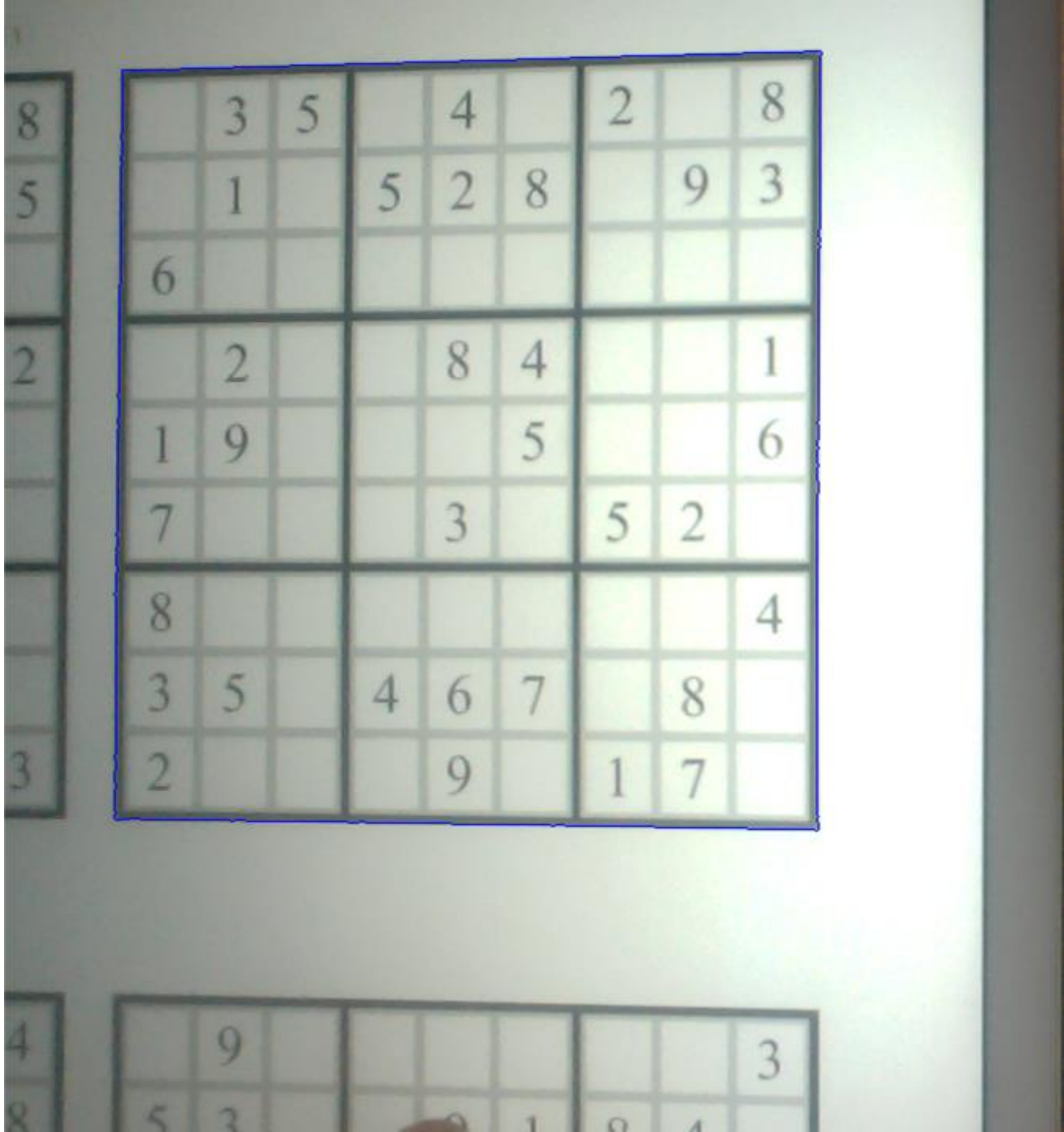


Sub steps in step 1

- Initially, the image is Grayscaled ,binary thresholded and a blur(median blur) is applied to remove any noise.
- Now I find contours in the image using open cv findContours() which returns us all the contours(all kinds of closed polygon).
- After finding the contours, I find the "Bounding Rect", since my sudoku is a square, I first find the largest contour which is a square(using openCV boundingRect()).

The resulting sudoku detection

The grid detected shown in blue line

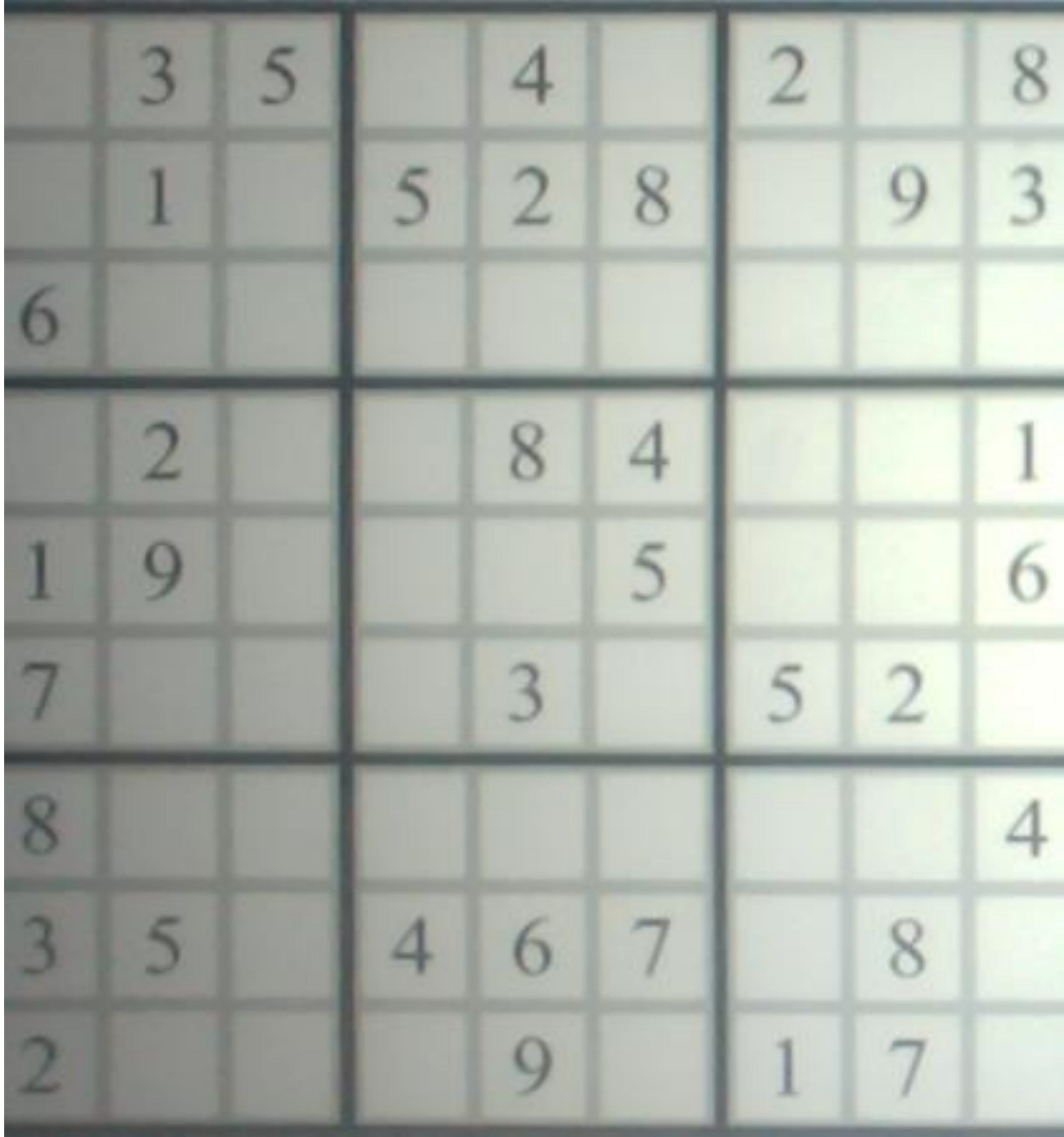


Sub steps involved in step 1(cont..)

- After having found the square that contains the Sudoku grid, we come across a problem.
- The sudoku image is not always held parallel onto the screen and the square could be rotated at an angle.
- To overcome this we do something called the Warp Perspective where warped image can be converted into a straight undistorted image.(This is a necessary step for digit recognition,will be explained later why!)
- I used the openCV getPerspective () transform which takes four corners of the detected Sudoku image(rotated or straight) and converts it into a straightened image using the warpPerspective()

The resulting image

- The Image is now straightened.



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

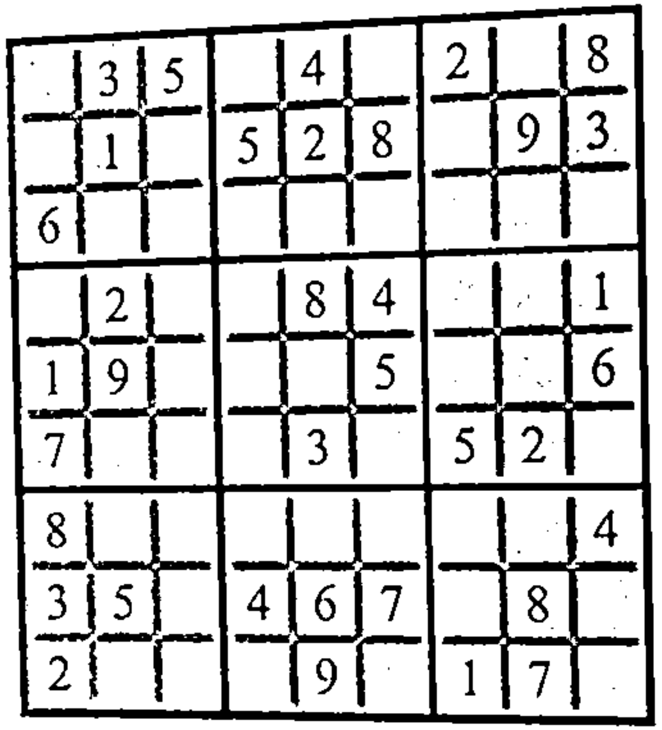
Extracting the Digits(sub steps 2)

- In this step I removed all the bounding grid lines in the extracted sudoku square, the only thing that remains is the bounding square and the digits within.
- The Grayscale extracted sudoku image is first dilated and then eroded using the openCV morphologyEx() function with a 3x3 kernel. This operation is called "Opening" which simply means dilation followed by erosion.
- This step is done because when the image is captured from the webcam, owing to different angles of the sudoku square and illumination, some digits can be broken(or disconnected) when the image is Binary Thresholded in the first step.
- This step makes sure all the digits are intact in their "context"

Sub steps 2(cont..)

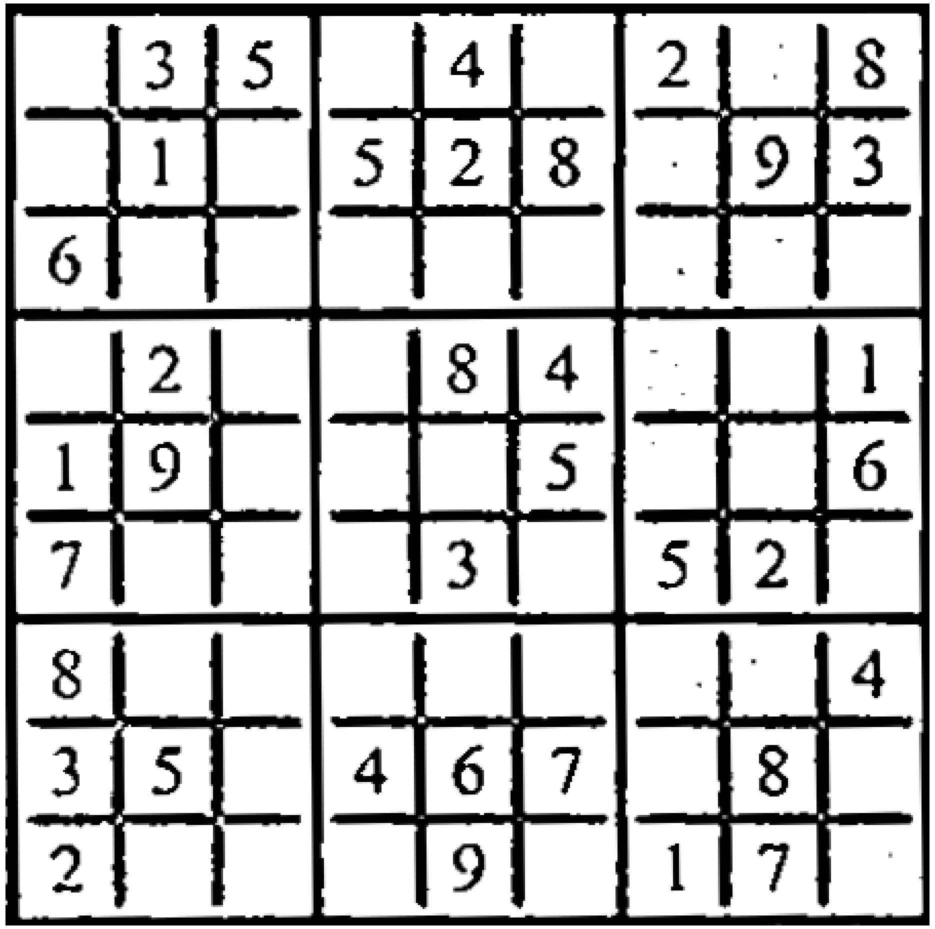
- After the "Opening" operation, we find Houghlines on the sudoku image, using openCV houghlines().
- It returns all the lines on the sudoku image including the four sides of the Sudoku itself.
- Comparing the original image and the image containing the lines, we remove the part from original image where there is line in the lines image, leaving only the digits
- Using the contours we found in the first step, we redraw the square over this new blank image only containing the digits.(Remember the fact that even the sides of square are detected as lines and during the comparison, they are removed as well. Hence, we have to redraw them).

After finding only the sudoku image.(separated from rest of the content)



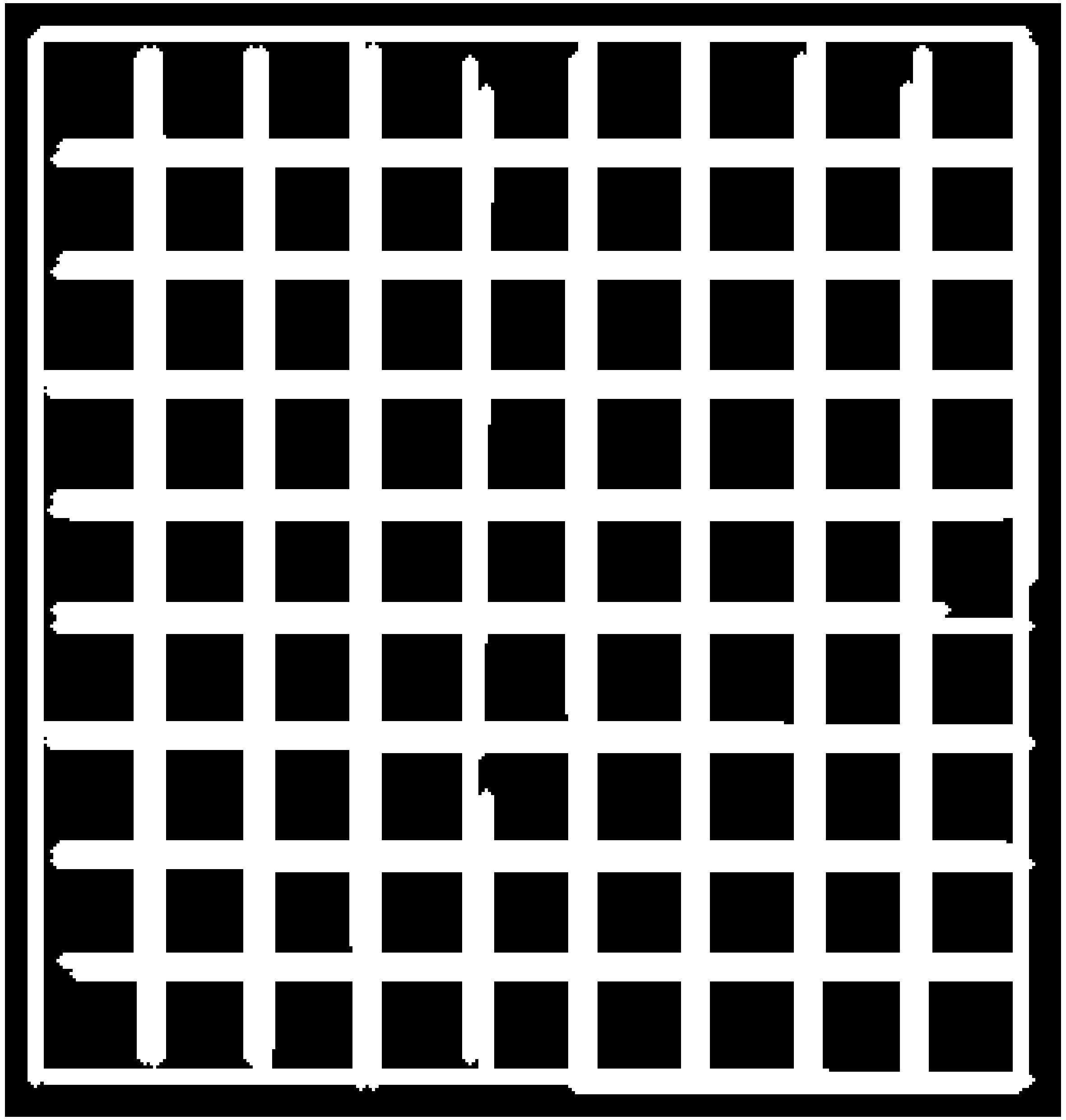
	3	5		4		2		8
	1		5	2	8		9	3
6								
	2			8	4			1
1	9				5			6
7				3		5	2	
8								4
3	5		4	6	7		8	
2				9		1	7	

Warped Image



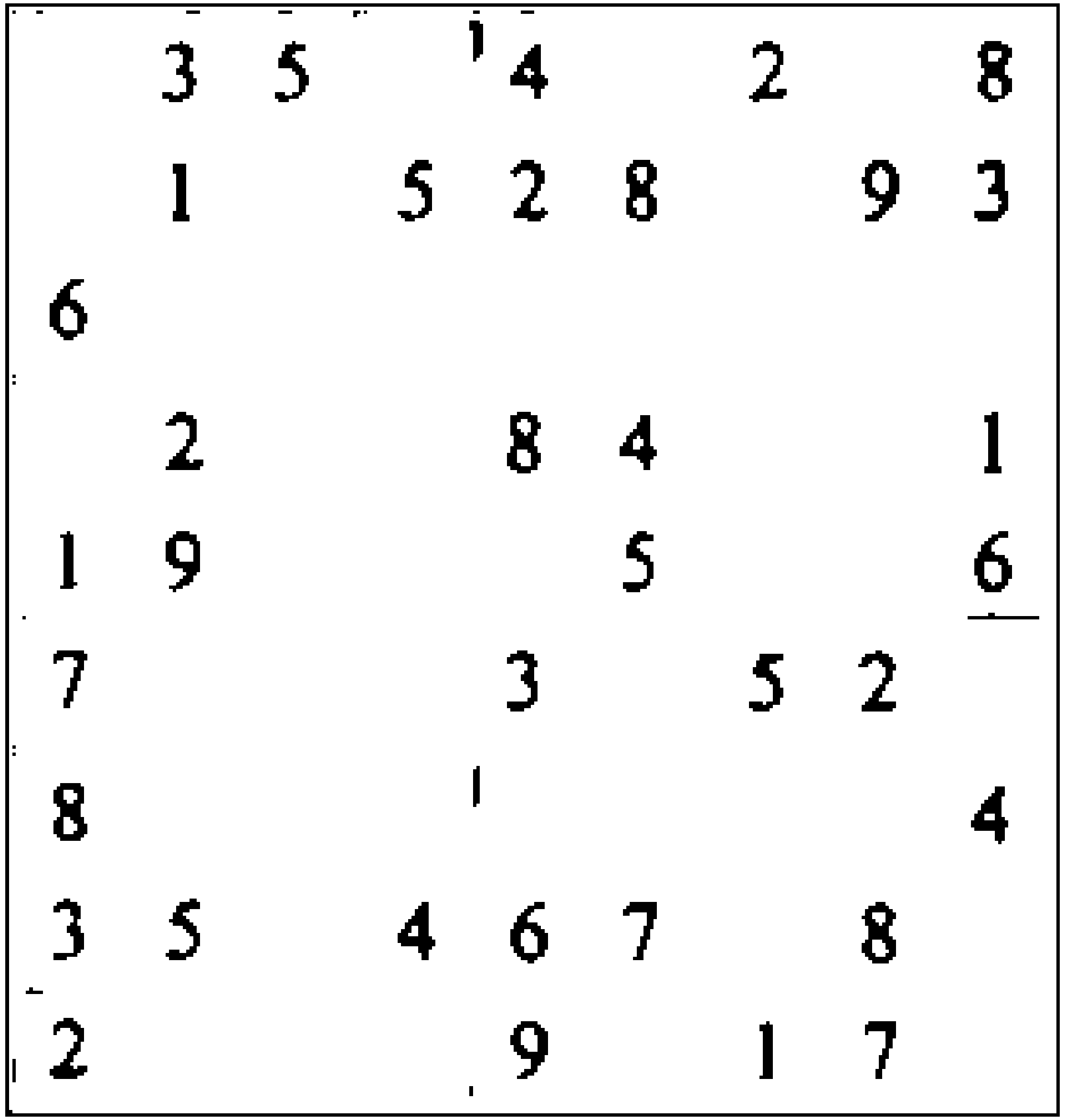
	3	5		4		2		8
	1		5	2	8		9	3
6								
	2			8	4			1
1	9				5			6
7				3		5	2	
8								4
3	5		4	6	7		8	
2				9		1	7	

Lines Detected



Finally..

Image only containing digits



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

Recognizing Digits(step 3)

- Now comes the digit recognition part where I have to detect the numbers from images.
- At first, I intended to use pretrained TensorFlow model created on mnist dataset, but it always returned 3 or 4 errors, since the test images were hand-written digits.
- I needed something that could detect the computer-generated fonts

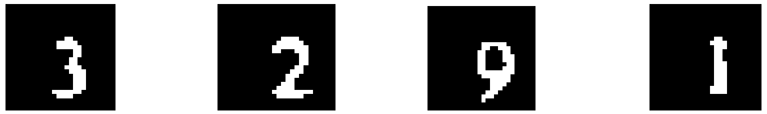
The solution?(part3)

Template Matching

Part 3(cont..)

- Template matching involves matching two image context using sum of squared difference measure. Images with most amount of similarity would have least SSD value, and hence, higher the similarity.
- I created sets of test images from multiple sudoku images, as digits detected wouldn't look the same always(different angles, illumination and many such factors)
- And then after finding my sudoku image and extracting digit images one by one I would template match against train images of digits 0-9 and predict the digit with least SSD measure.

Some sample train images

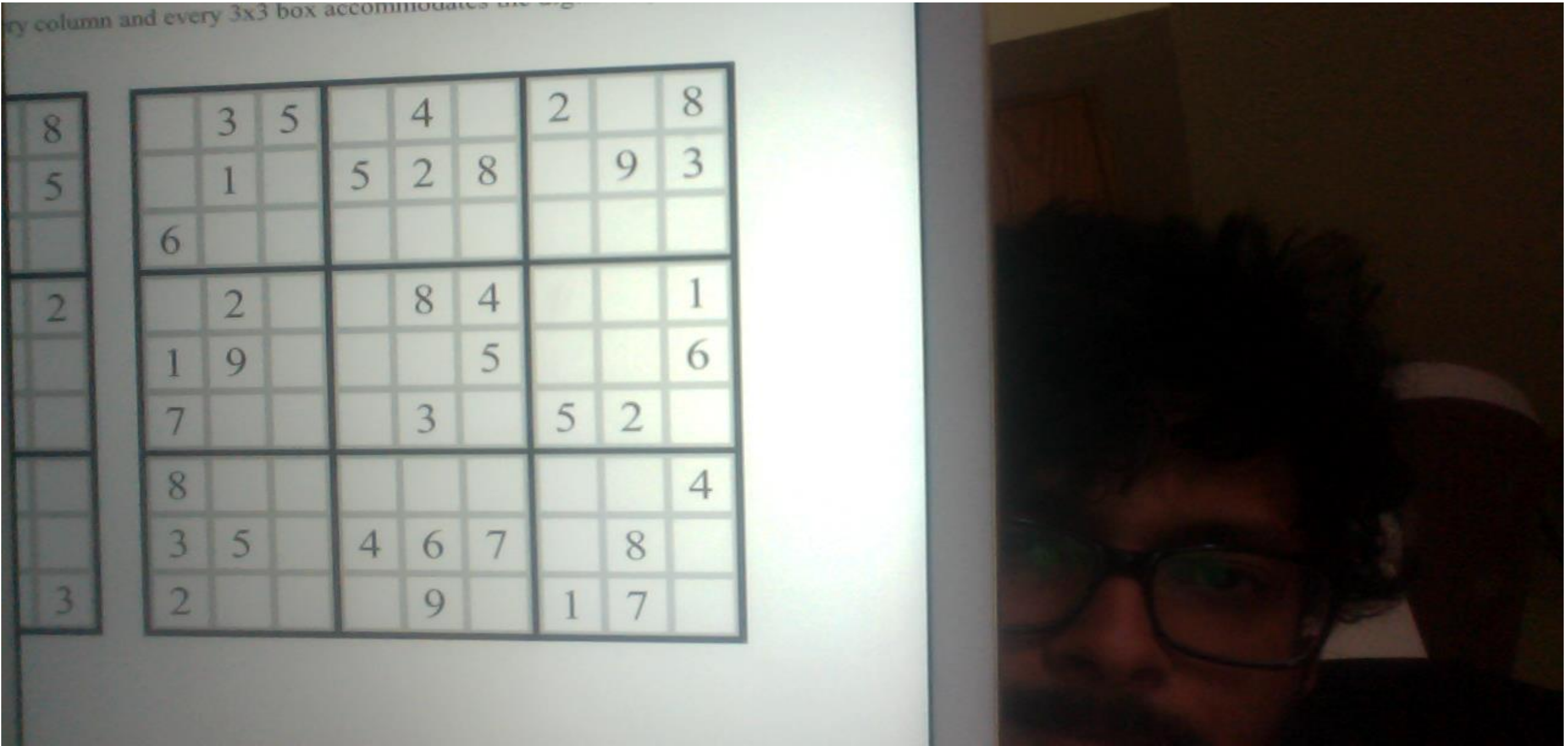


- A care must be taken that our test images are standardized just as the train images to obtain maximum accuracy
- The test images are resized to 26X26 and "Opened" again to retain "content".
- Similarly, in the recognition part the digits are resized again to 26X26 and the Centre of mass of the digit is shifted to a standard point as in the train images.
- Finally, on finding the image with which our test image finds the least SSD, it is given the corresponding number.

Final Step

- Finally the recognized numbers are sent to the solver.py which returns the solved sudoku digits.
- It is a basic backtracking search algorithm with a heuristic.
- The digits are then overlapped over the original initial image.

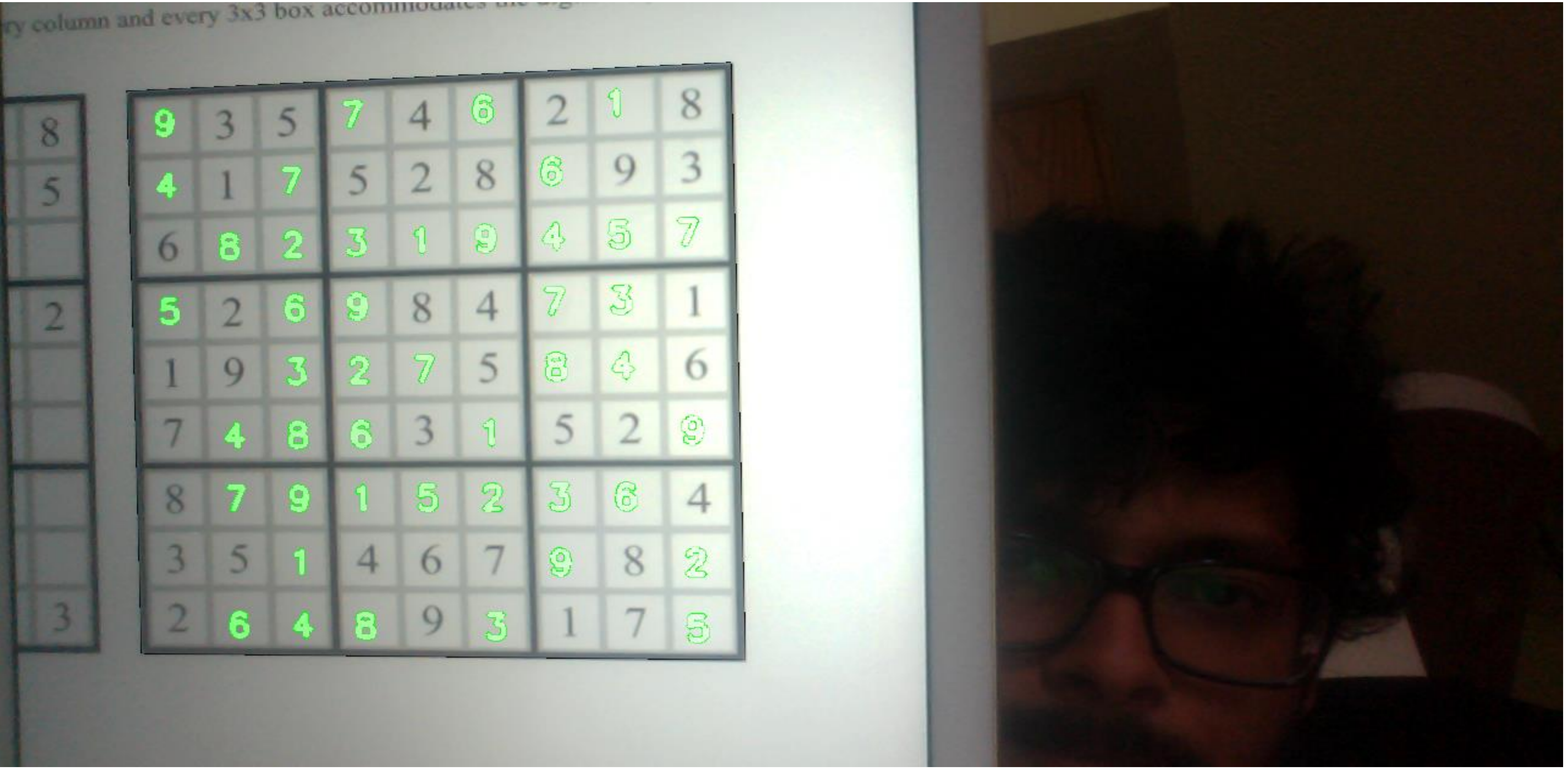
Initial Image



ry column and every 3x3 box accommodates the

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

Final Output



ry column and every 3x3 box accommodates the digi

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

Assumptions, Limitations and Future Work.

- Datsets that could detect all kinds of fonts and styles of digit Images and probably using faster detection models.

- Making sudoku detection more robust considering factors like illumination, rotation angle,distortions etc.

THANK YOU!