CSE483

Computer Vision

SUDOKU SOLVER

PHASE 2

Contents

Methodology

Preprocessing

After getting the tiles from phase one we needed some preprocessing before going to OCR this involved

1. Median filter to remove salt and pepper noise
2. Trying to remove shadows and threshold the image
3. Apply median filter again to the threshold image
4. Remove any collisions with the border (possibly caused by wrong cuts in phase 1)

Technique used to remove the shadows and threshold the image (step 2)

The idea generally is to dilate the image (which should make the digit bigger) adding then blurring to suppress any noise so we should now have an image with the background containing any shadow or discoloration

now take the difference between the original image and the image we obtained the pixels that have close value to each other will be black other than that they will be white (which have high difference)

so, we can say we removed a lot of shadow and noise in the background

now apply adaptive threshold and OTSU threshold and take the ANDing between them there we get our desired image

at last I remove any connected component that touches the border

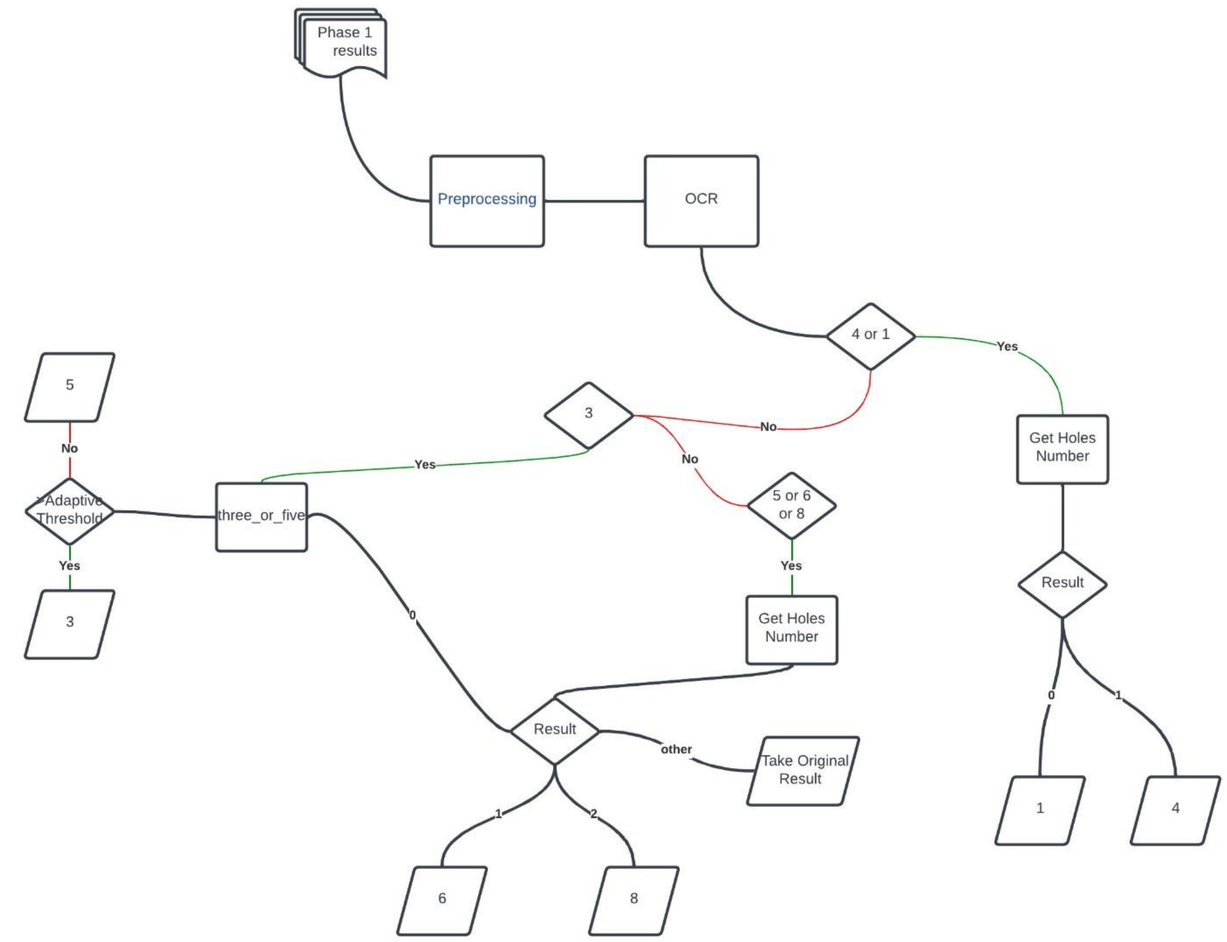
now let's go to OCR

Optical Character Recognition OCR

We'll do a classical approach using morphological operations and "gold" reference digit structuring elements. By seeing how well each tile matches each of the digits, we could detect the value in each tile.

Its like doing some convolution on the image with our SE to find if its pattern is in the image or not

But this method alone wont work good enough as there are patterns of numbers in other numbers like you can find the pattern of the number 1 in number 4 so we needed some post processing explained in the following flowchart

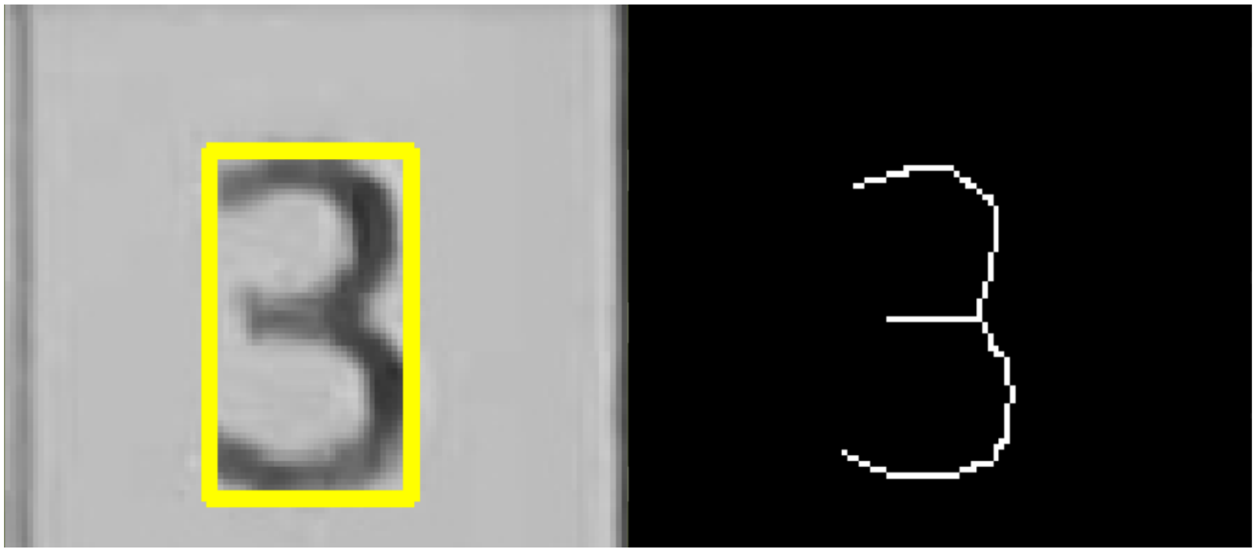


To Get the number of holes we used function called findContours we used it more than once so it's worth mentioning it simply follows the borders of binary image but with marking policy to know which border is the parent of which one

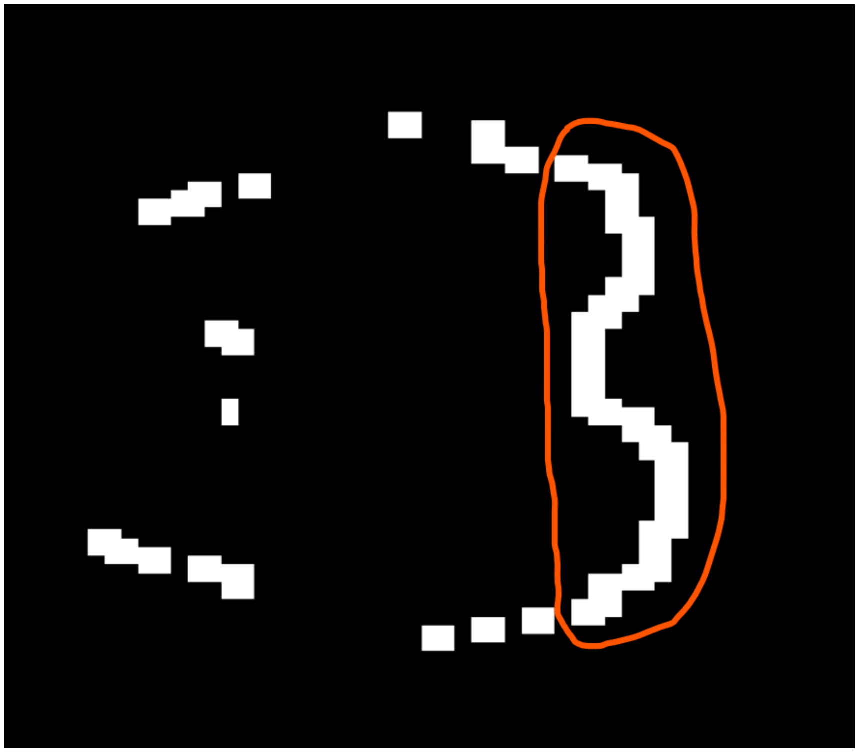
Now let's talk about 3 and 5 as it was very tricky

We want to differentiate between them both so the idea it to apply a vertical lines detector hopefully it will break both 5 and 3 and due to the structure of the 3 that doesn't contain perfect horizontal lines as 5 we can get the max area of the largest broken pieces and hopefully the 3 will be bigger

But before applying perwitt we need to do thinning to not detect the vertical edges multiple times

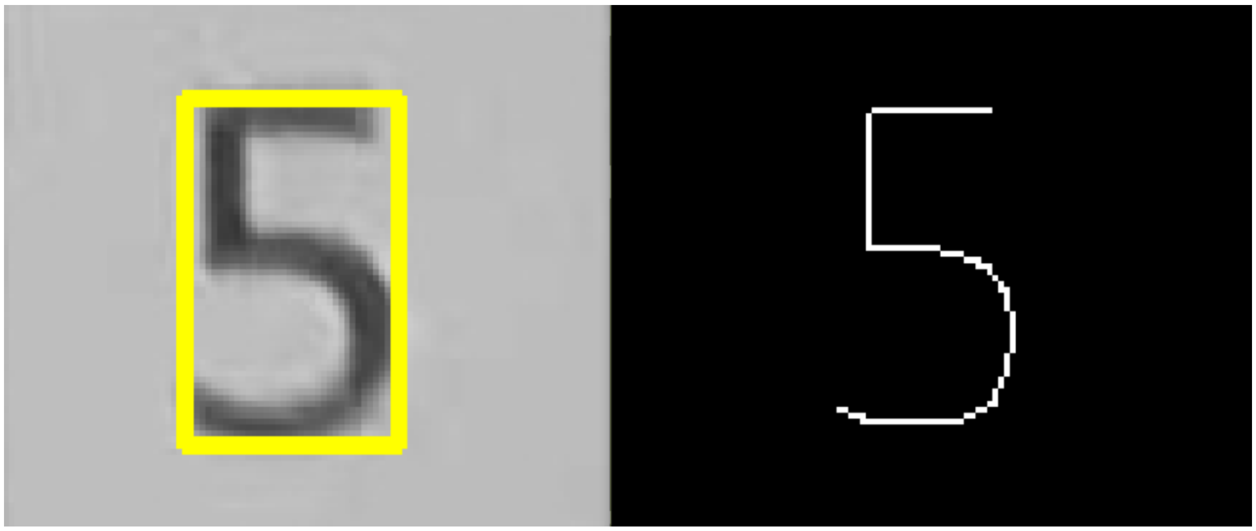


As you can see in this image the result of thinning the 3 now after applying perwitt

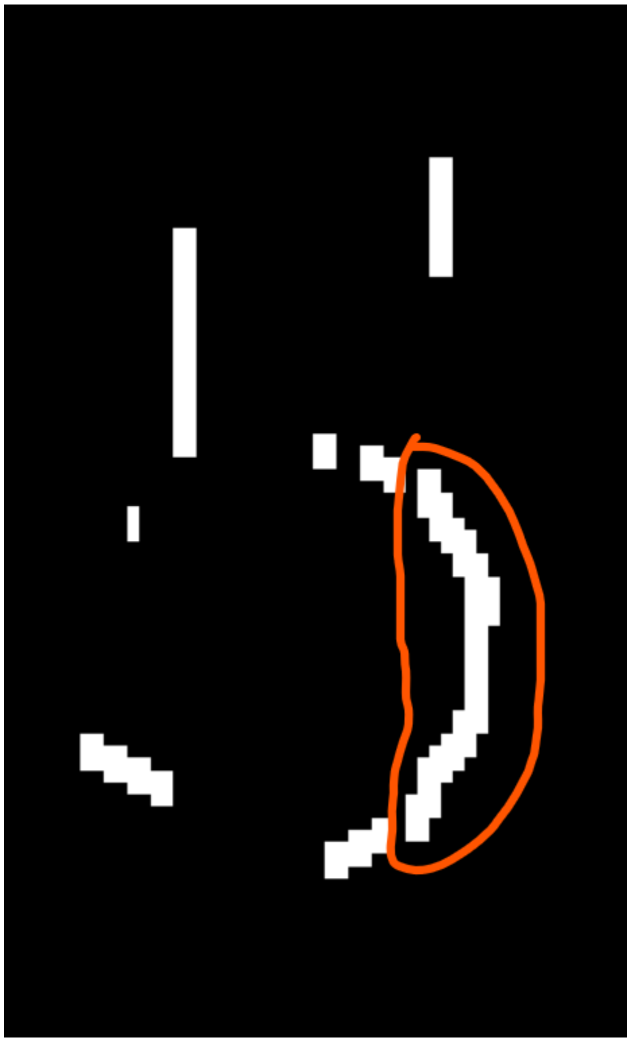


We will calculate the are of the red part as it’s the largest connected component

Making the same thing on 5



Then applying perwitt



We will get the part which is smaller than the 3

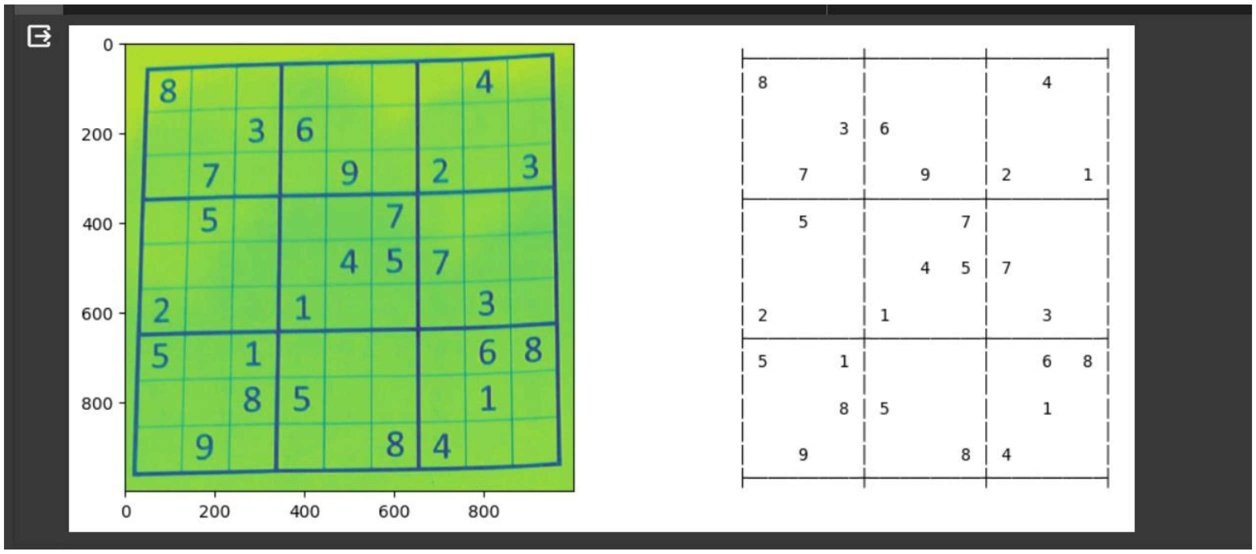
Now we want to make a threshold to determine is it a 3 or 5

This is done by getting all values from digits that are identified as 3 or 5 and calculating the area then using OTSU method we calculate the global optimal threshold which will separate the numbers into two histograms

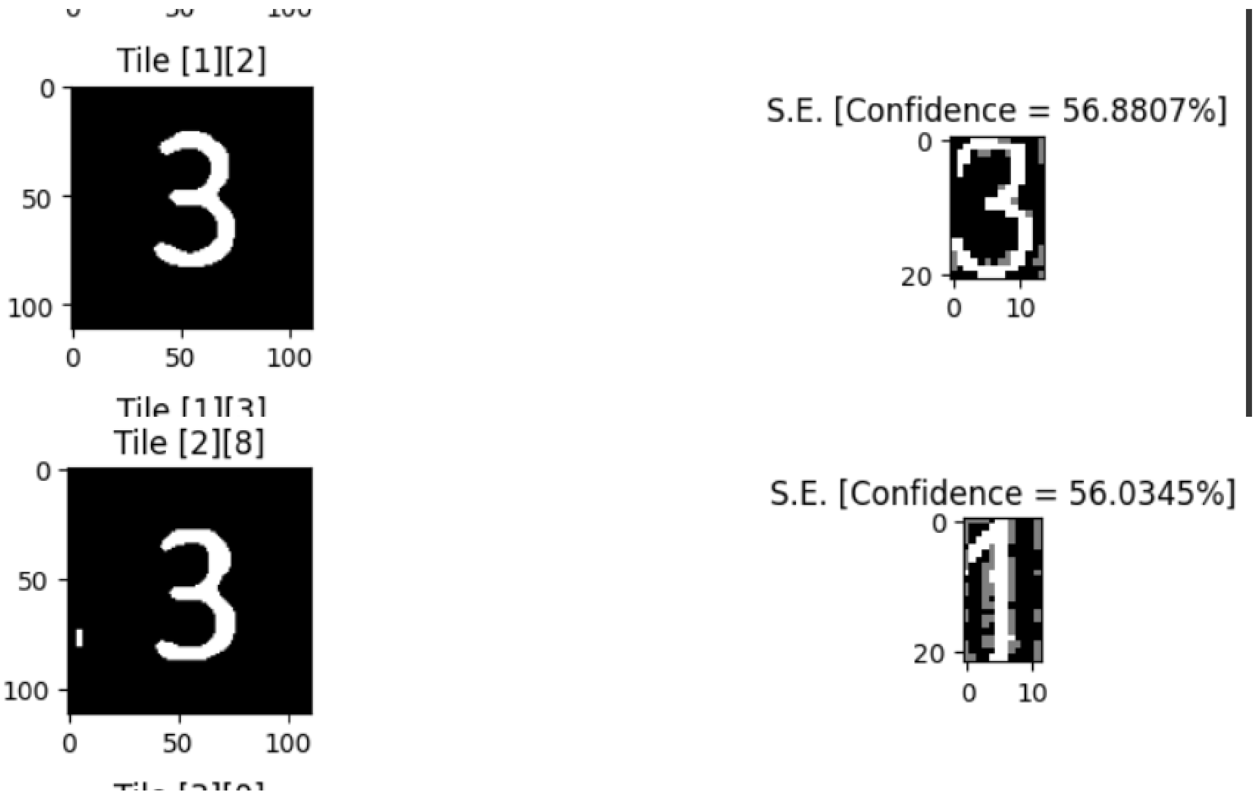
We analyze the results below

Results

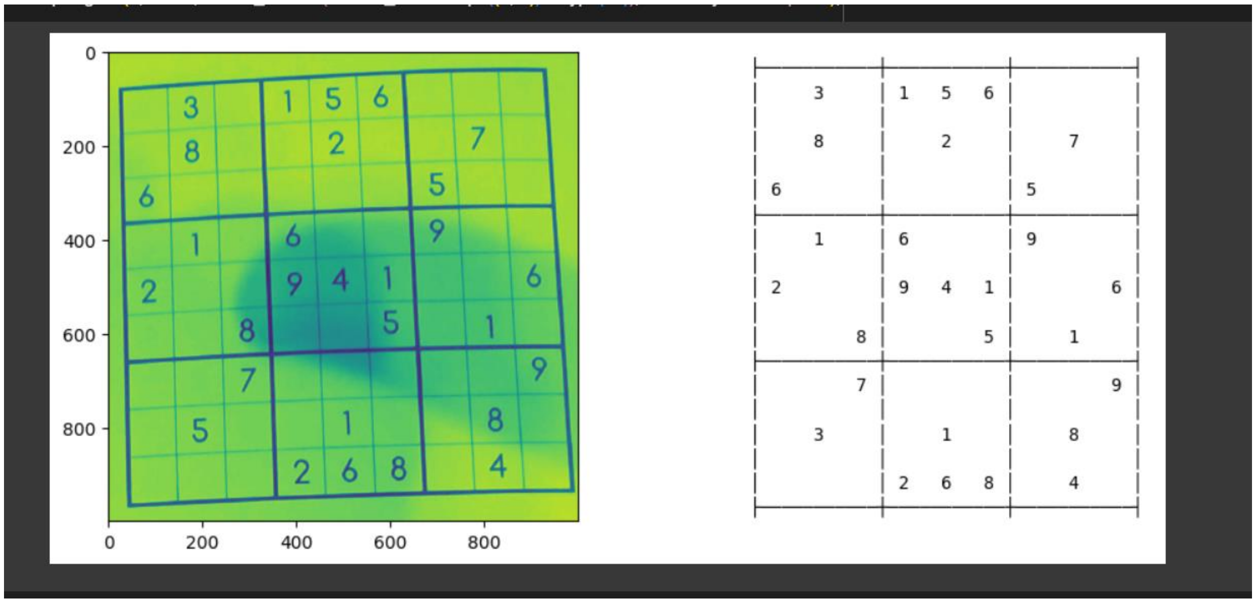
01-Normal



Here all numbers are detected except for 3 and 1 which you can see the difference between the confidence values were too small (also you can see the pattern of one in the right most side of the 3)

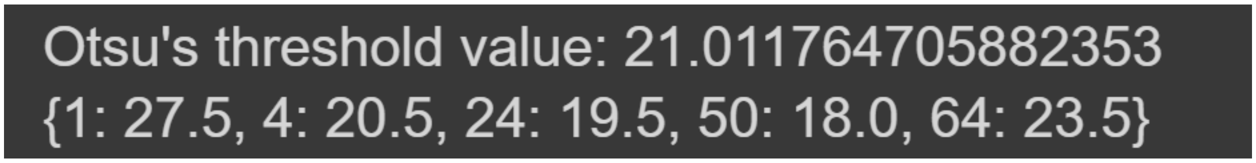


02-TheMightyFinger



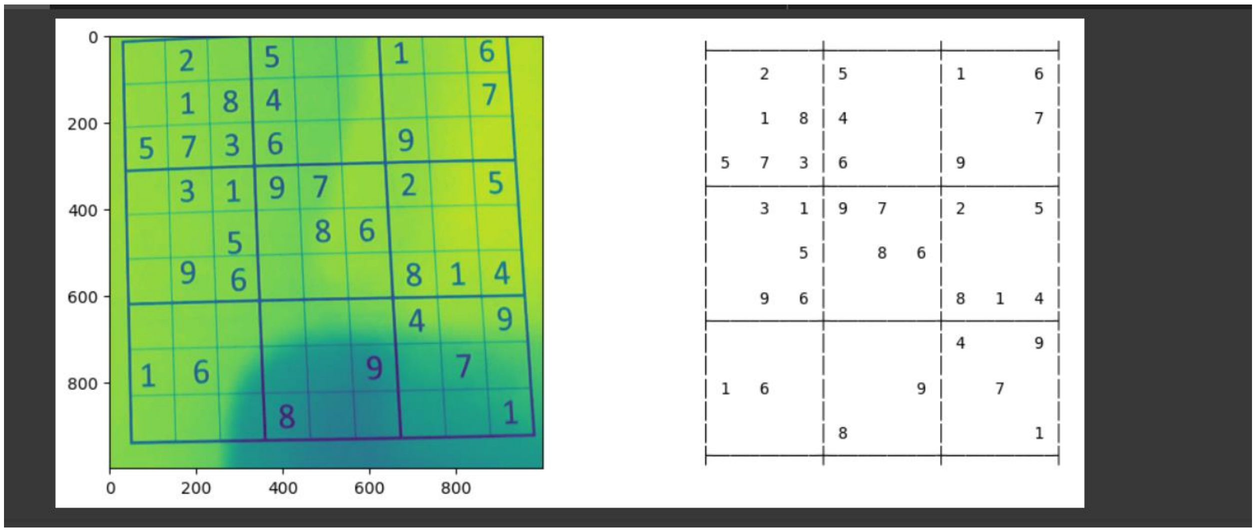
```
+-------+-------+-------+
|   3   | 1 5 6 |       |
|   8   |   2   |   7   |
| 6     |       | 5     |
+-------+-------+-------+
|   1   | 6     | 9     |
| 2     | 9 4 1 |     6 |
|     8 |     5 |   1   |
+-------+-------+-------+
|     7 |       |     9 |
|   3   |   1   |   8   |
|       | 2 6 8 |   4   |
+-------+-------+-------+
```

As this picture contains only one three and more fives than the threes using the optimal global threshold between three and five identified the biggest response from fives as a three



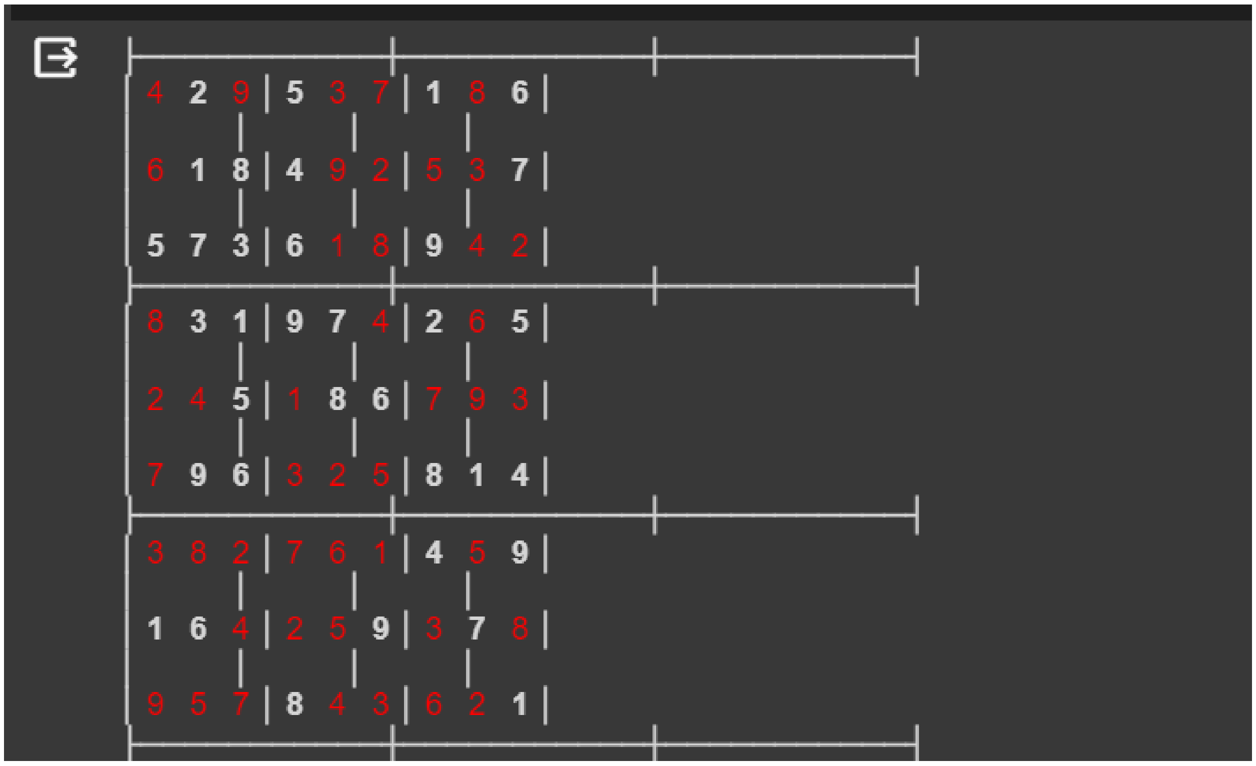
```
Otsu's threshold value: 21.011764705882353
{1: 27.5, 4: 20.5, 24: 19.5, 50: 18.0, 64: 23.5}
```

03-WhereBorder



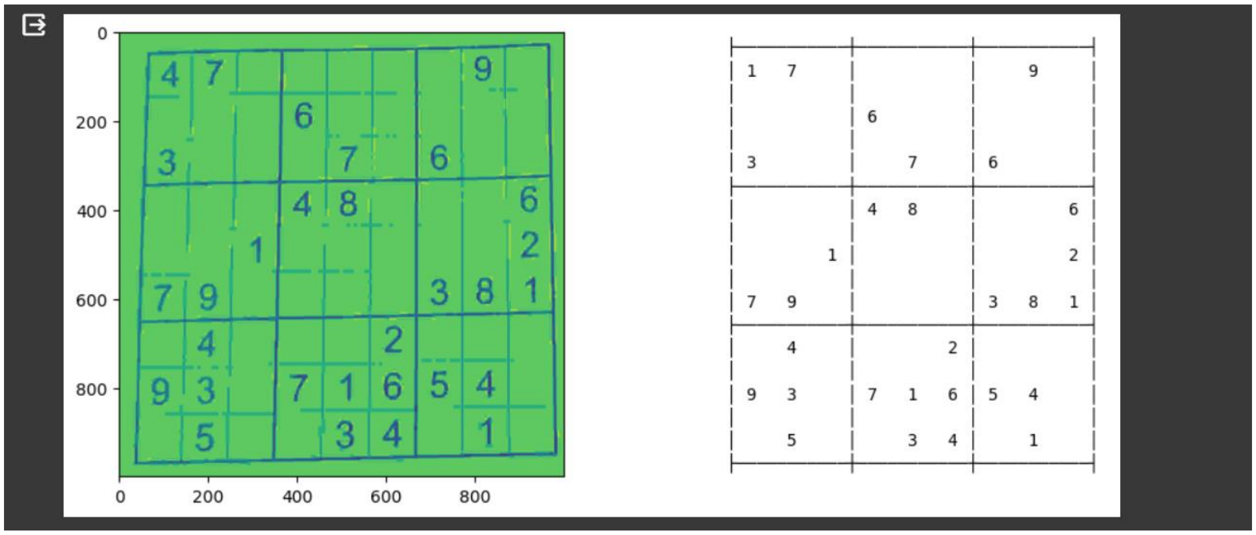
```
+-------+-------+-------+
|   2   | 5     | 1   6 |
|   1 8 | 4     |     7 |
| 5 7 3 | 6     | 9     |
+-------+-------+-------+
|   3 1 | 9 7   | 2   5 |
|     5 |   8 6 |       |
|   9 6 |       | 8 1 4 |
+-------+-------+-------+
|       |       | 4   9 |
| 1 6   |     9 |   7   |
|       | 8     |     1 |
+-------+-------+-------+
```

I just love this image

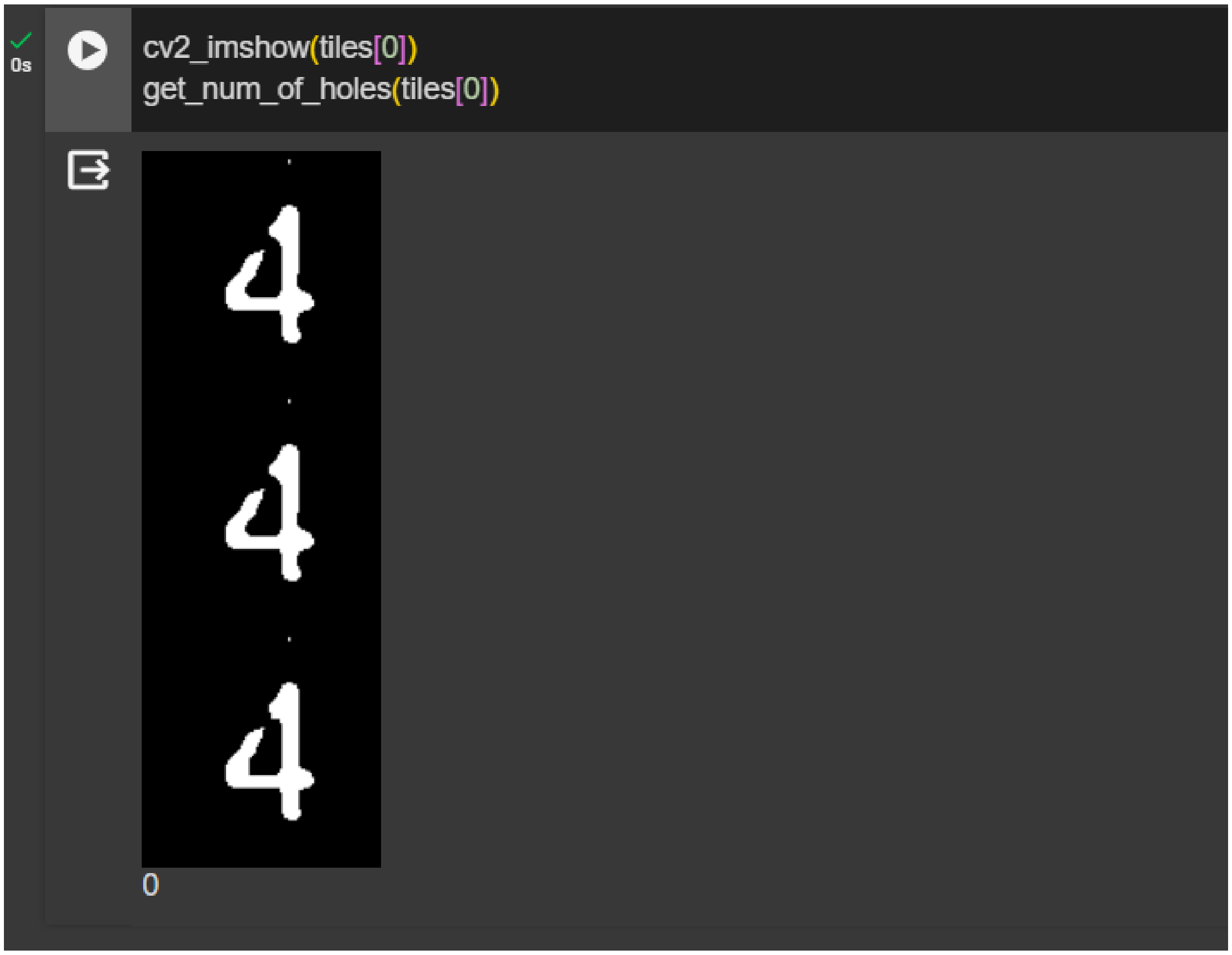


```
 4 2 9 | 5 3 7 | 1 8 6 |
 6 1 8 | 4 9 2 | 5 3 7 |
 5 7 3 | 6 1 8 | 9 4 2 |
 8 3 1 | 9 7 4 | 2 6 5 |
 2 4 5 | 1 8 6 | 7 9 3 |
 7 9 6 | 3 2 5 | 8 1 4 |
 3 8 2 | 7 6 1 | 4 5 9 |
 1 6 4 | 2 5 9 | 3 7 8 |
 9 5 7 | 8 4 3 | 6 2 1 |
```

04-CompressoEspresso



The four here is detected as a one because it was not fully closed so the algorithm read it has 0 holes even after performing closing operation

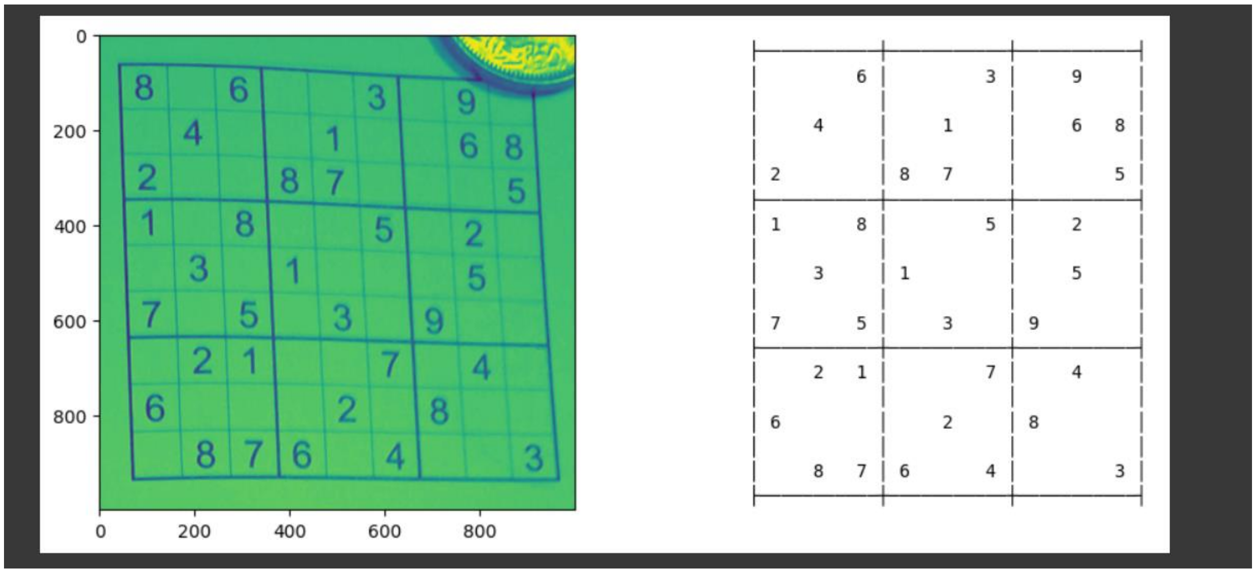


```
cv2_imshow(tiles[0])
get_num_of_holes(tiles[0])
```

```
0
```

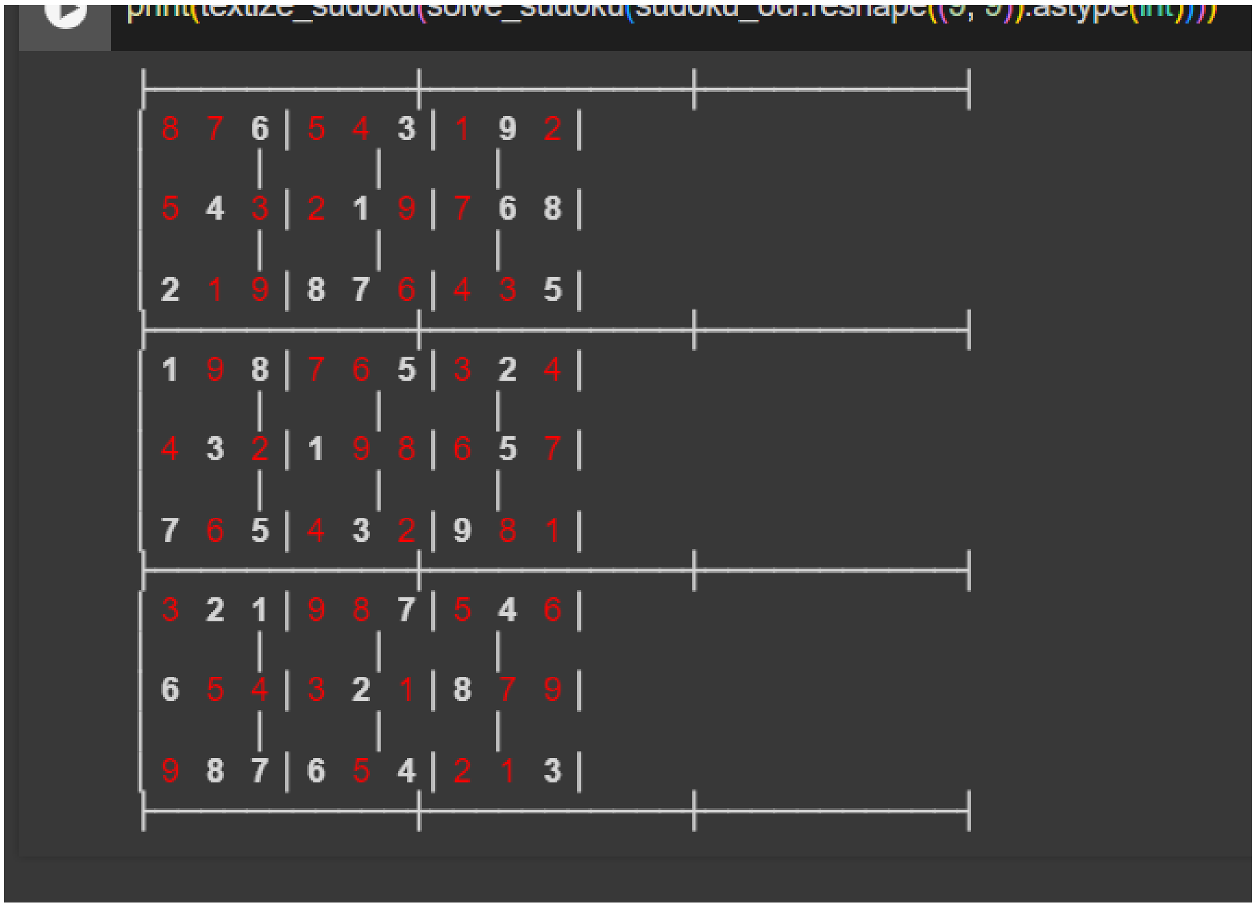
The image has 0 holes after closing

05-Ya3amSheel2elCoinYa3am



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

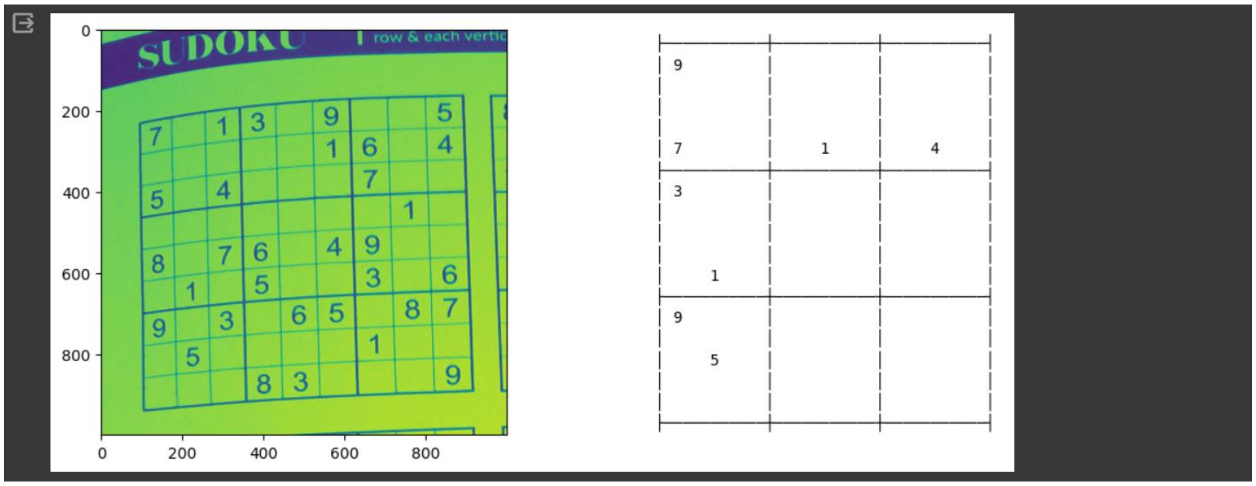
The 8 on the top left corner was removed as a part of preprocessing as I am trying to remove anything collided with the border which was a problem in the first phase during cutting the squares although sudoku still managed to be solved



```
print(textize_sudoku(solve_sudoku(sudoku_ocr.reshape((9, 9)).astype(int))))
```

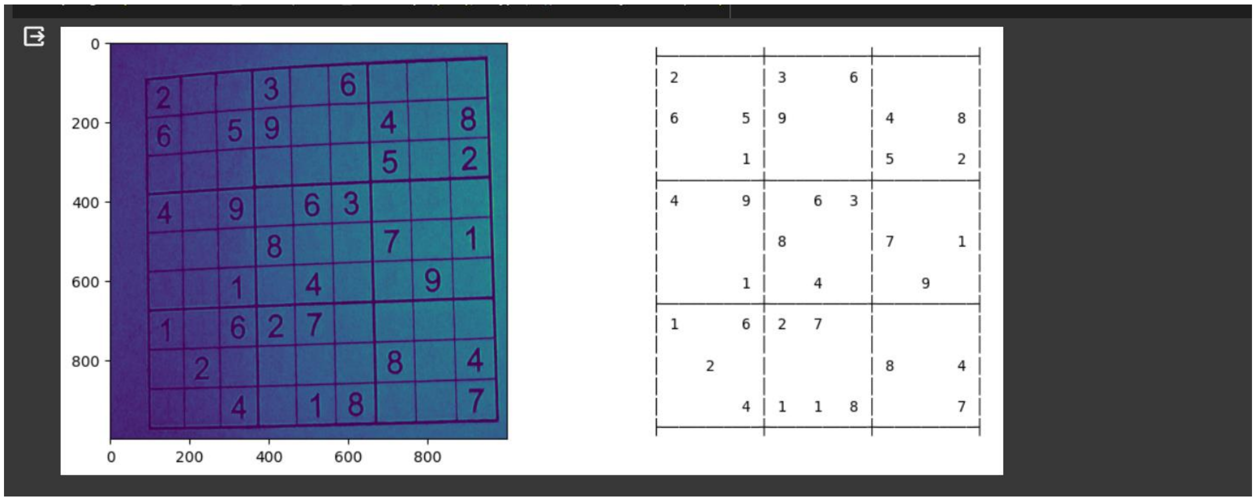
```
| 8 7 6 | 5 4 3 | 1 9 2 |
| 5 4 3 | 2 1 9 | 7 6 8 |
| 2 1 9 | 8 7 6 | 4 3 5 |
| 1 9 8 | 7 6 5 | 3 2 4 |
| 4 3 2 | 1 9 8 | 6 5 7 |
| 7 6 5 | 4 3 2 | 9 8 1 |
| 3 2 1 | 9 8 7 | 5 4 6 |
| 6 5 4 | 3 2 1 | 8 7 9 |
| 9 8 7 | 6 5 4 | 2 1 3 |
```

06-FarAndCurved



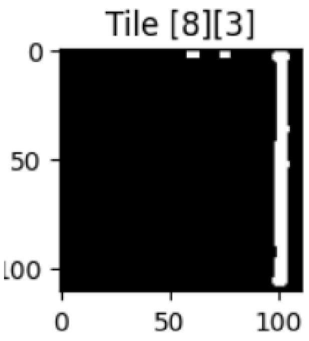
This case failed in phase 1 so nothing to expect here

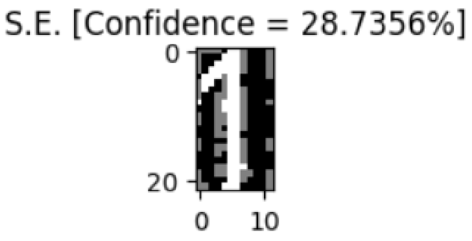
07-2elNoor2ata3



```
|-----------|-----------|-----------|
| 2         | 3       6 |           |
| 6       5 | 9         | 4       8 |
|         1 |           | 5       2 |
|-----------|-----------|-----------|
| 4       9 |     6   3 |           |
|           | 8         | 7       1 |
|         1 |     4     |     9     |
|-----------|-----------|-----------|
| 1       6 | 2   7     |           |
|     2     |           | 8       4 |
|         4 | 1   1   8 |         7 |
|-----------|-----------|-----------|
```

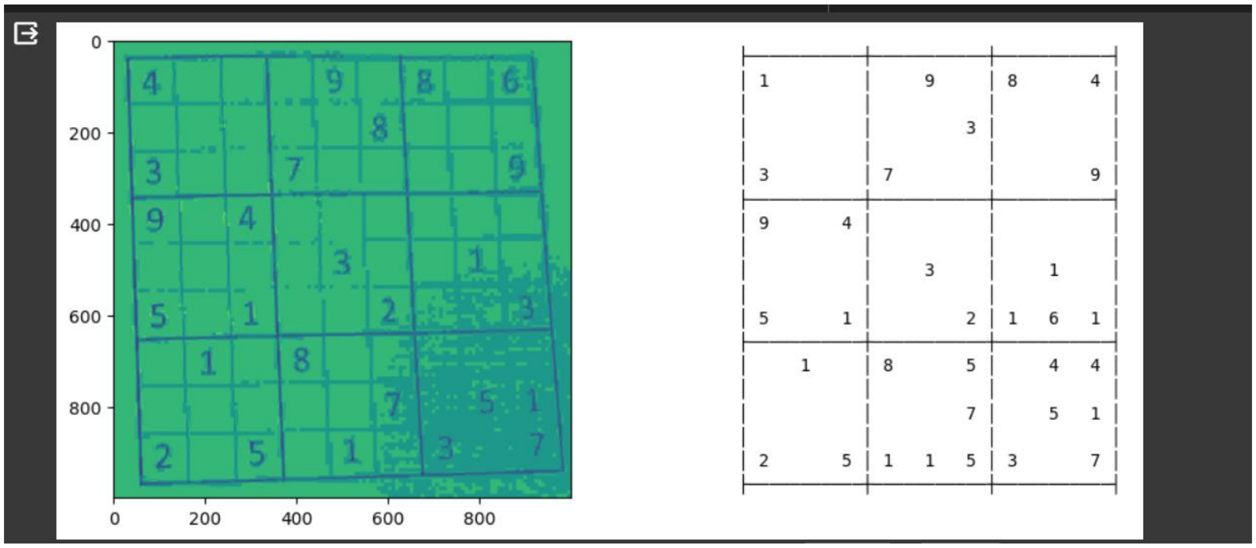
The problem here is a one that got detected even its not there the image below shows why this happened





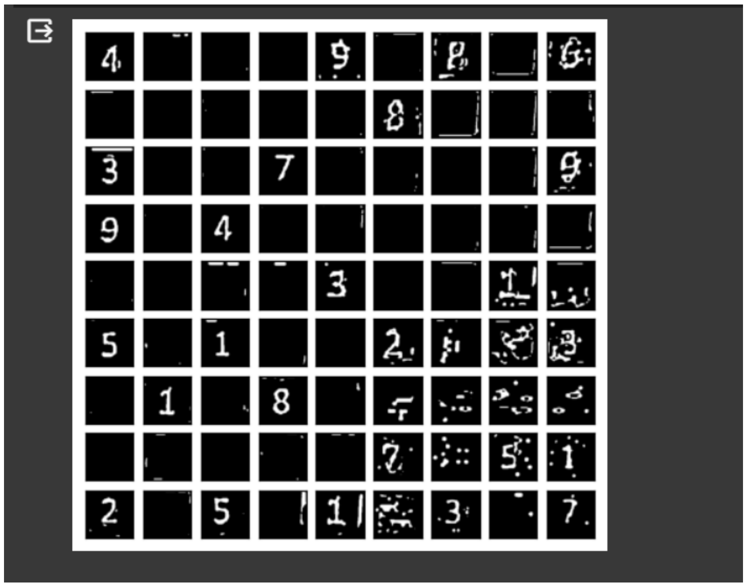
This line should have been removed if it was touching the border, but it wasn't, which caused this problem (dilating would cause other numbers to touch the border that's why this option is not implemented)

08-MeshShayef7agaYa3am

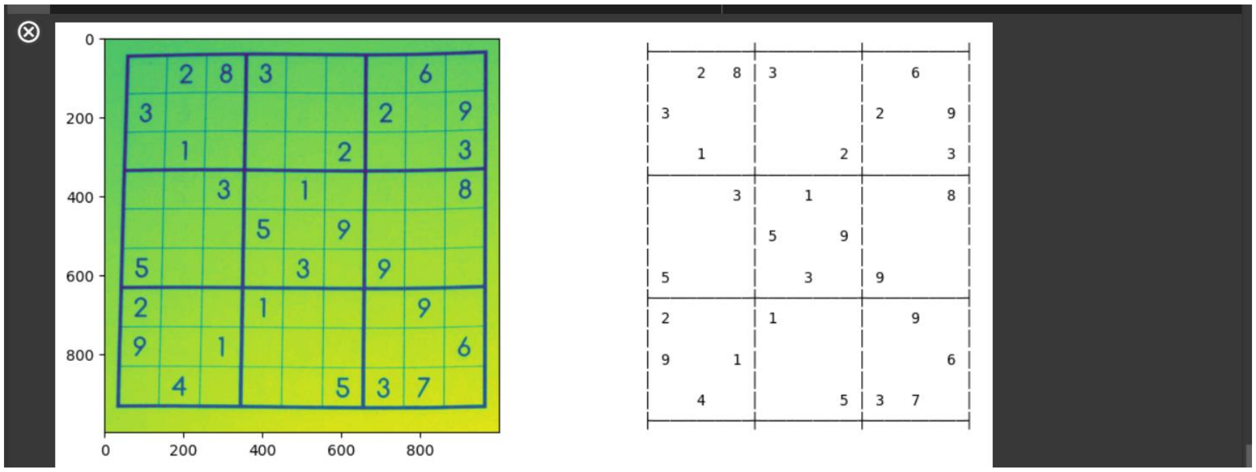


```
+-------+-------+-------+
| 1     |   9   | 8   4 |
|       |     3 |       |
| 3     | 7     |     9 |
+-------+-------+-------+
| 9   4 |       |       |
|       |   3   |   1   |
| 5   1 |     2 | 1 6 1 |
+-------+-------+-------+
|   1   | 8   5 |   4 4 |
|       |     7 |   5 1 |
| 2   5 | 1 1 5 | 3   7 |
+-------+-------+-------+
```

This image was so dark especially the bottom right corner so when we applied our algorithm to remove the shadow it failed so mostly this a failure in preprocessing histogram equalization could be used but it contradicts the idea of our algorithm

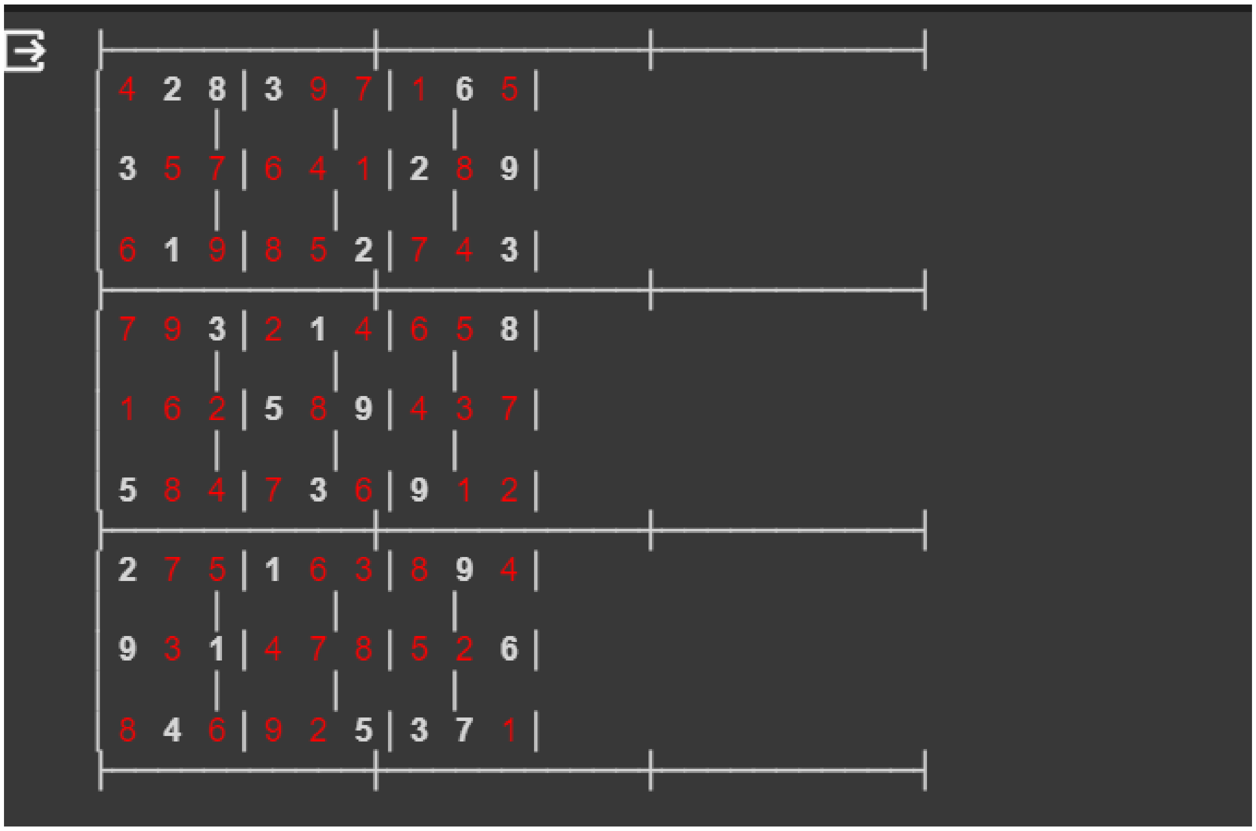


09-Normal2



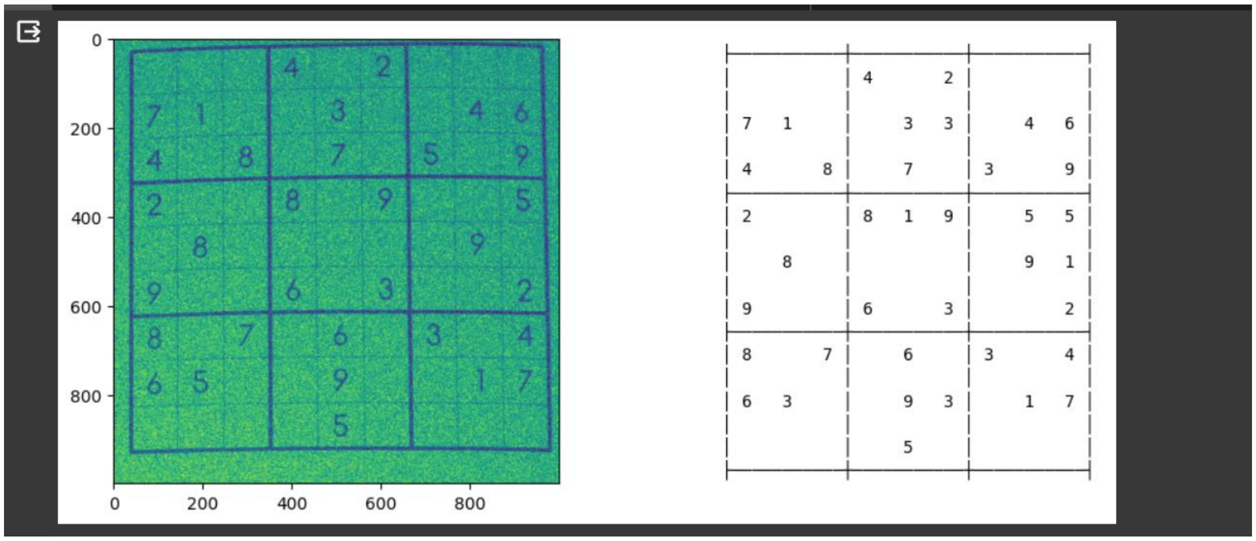
```
|-------|-------|-------|
|   2 8 | 3     |   6   |
| 3     |       | 2   9 |
|   1   |     2 |     3 |
|-------|-------|-------|
|     3 |   1   |     8 |
|       | 5   9 |       |
| 5     |   3   | 9     |
|-------|-------|-------|
| 2     | 1     |   9   |
| 9   1 |       |     6 |
|   4   |     5 | 3 7   |
|-------|-------|-------|
```

This is my favorite



```
|-------|-------|-------|
| 4 2 8 | 3 9 7 | 1 6 5 |
| 3 5 7 | 6 4 1 | 2 8 9 |
| 6 1 9 | 8 5 2 | 7 4 3 |
|-------|-------|-------|
| 7 9 3 | 2 1 4 | 6 5 8 |
| 1 6 2 | 5 8 9 | 4 3 7 |
| 5 8 4 | 7 3 6 | 9 1 2 |
|-------|-------|-------|
| 2 7 5 | 1 6 3 | 8 9 4 |
| 9 3 1 | 4 7 8 | 5 2 6 |
| 8 4 6 | 9 2 5 | 3 7 1 |
|-------|-------|-------|
```

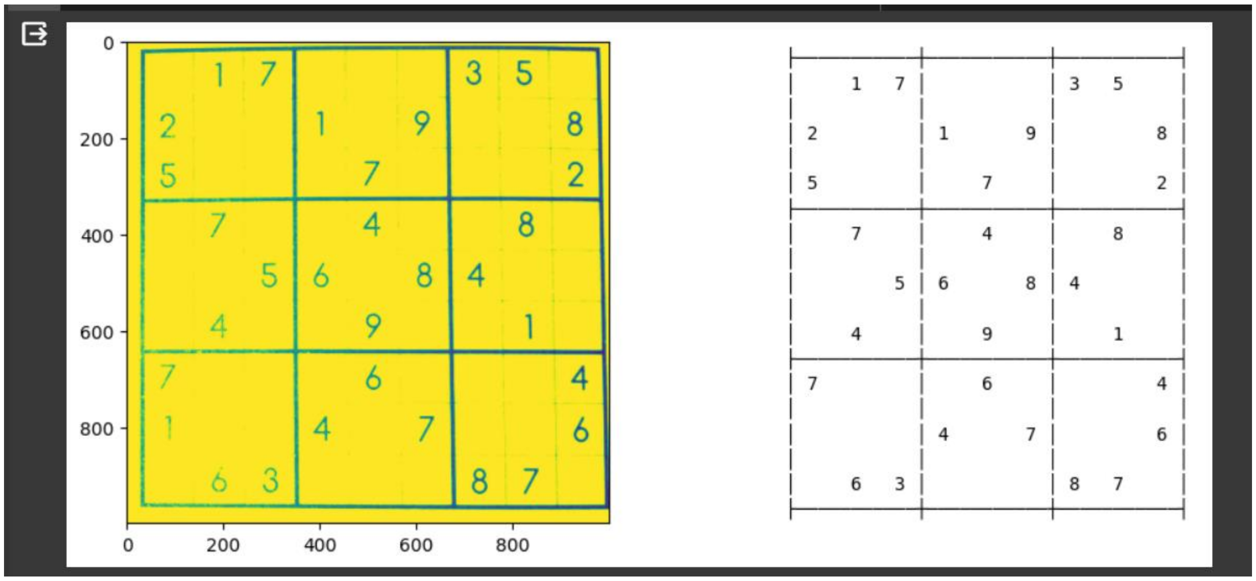
10-Mal7wFelfel



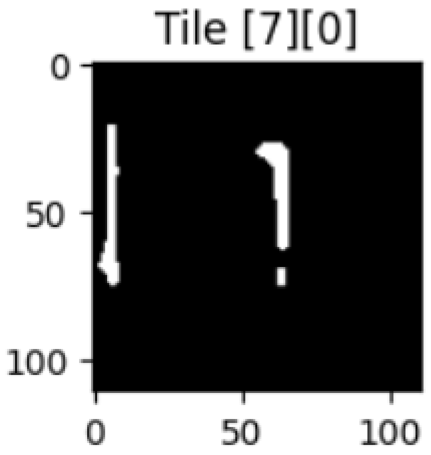
```
|       | 4   2 |       |
| 7 1   |   3 3 |   4 6 |
| 4   8 |   7   | 3   9 |
|-------+-------+-------|
| 2     | 8 1 9 |   5 5 |
|   8   |       |   9 1 |
| 9     | 6   3 |     2 |
|-------+-------+-------|
| 8   7 |   6   | 3   4 |
| 6 3   |   9 3 |   1 7 |
|       |   5   |       |
```

Although median filter was applied two times the noise resulted in detecting numbers that aren’t there and changing the number 5

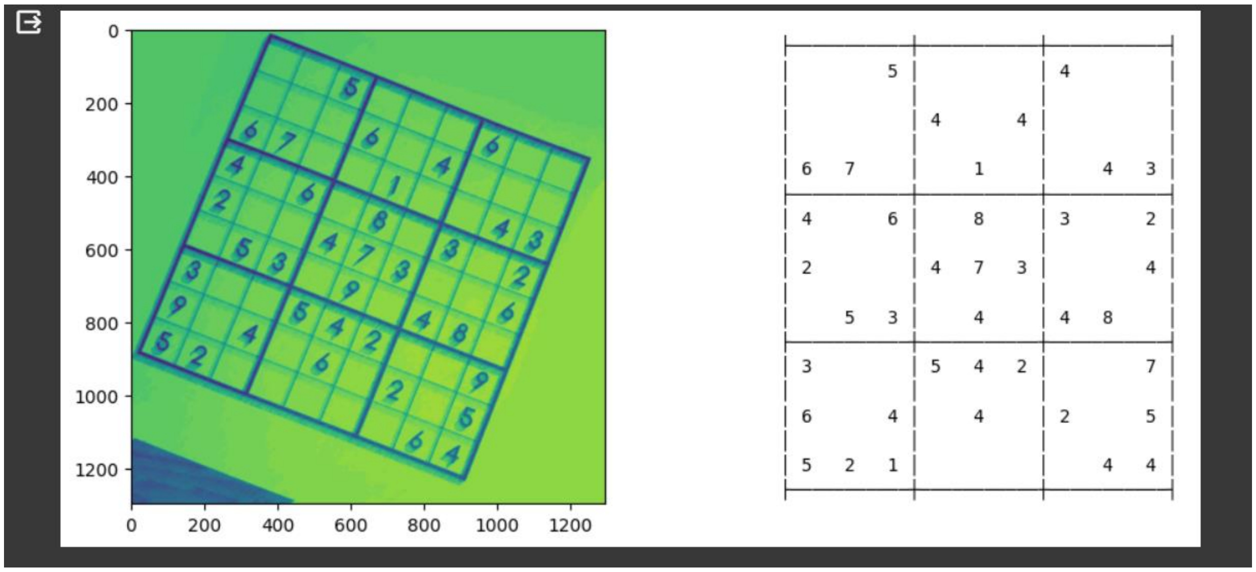
11-FlashBang



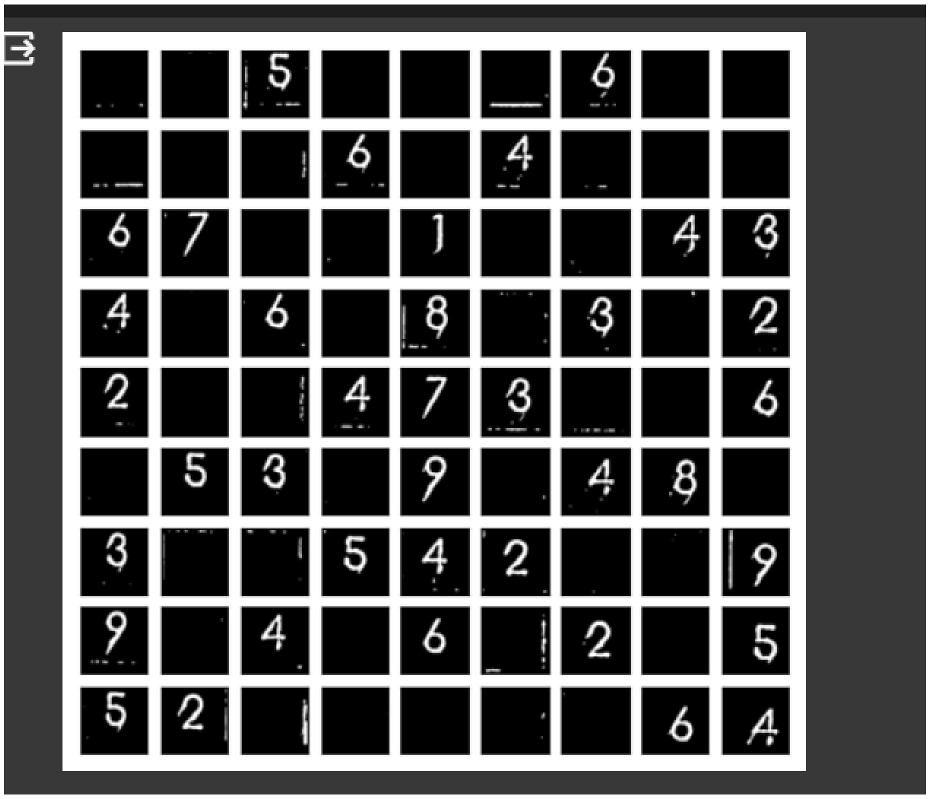
The one at tile 7,0 didn't get detected as you can see the one is not connected as it should even after closing the image making it much shorter than a normal one would do,While all other numbers are detected



12-BrokenPrinter

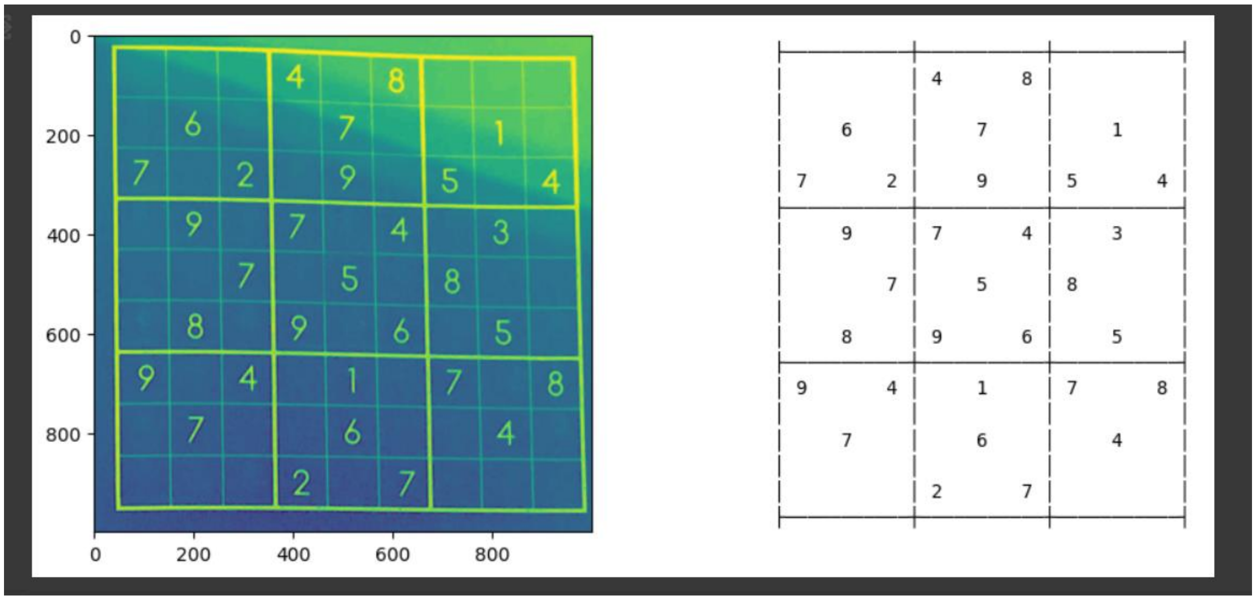


```
|       5 |       | 4     |
|         | 4   4 |       |
| 6 7     |   1   |   4 3 |
| 4     6 |   8   | 3   2 |
| 2       | 4 7 3 |     4 |
|   5   3 |   4   | 4 8   |
| 3       | 5 4 2 |     7 |
| 6     4 |   4   | 2   5 |
| 5 2   1 |       |   4 4 |
```



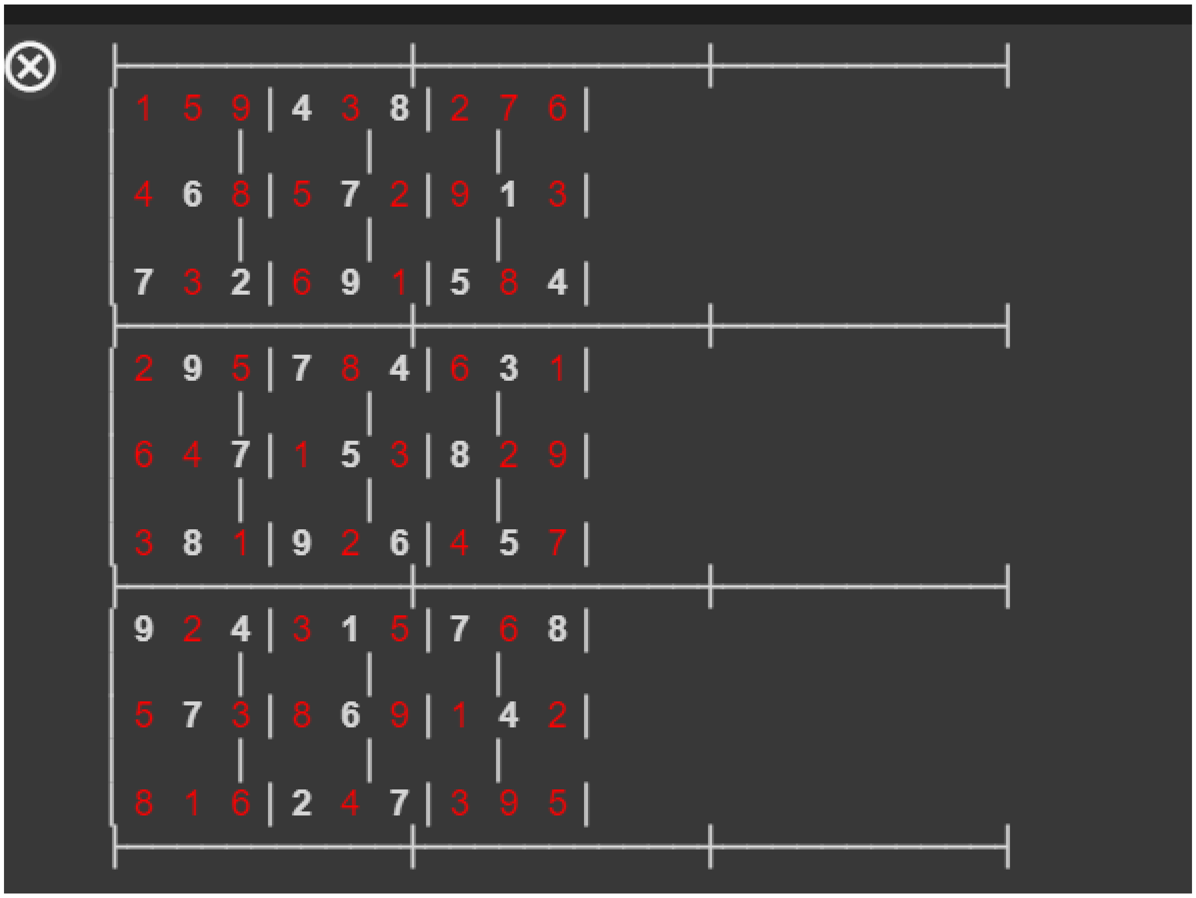
A lot of numbers here are detected as fours especially 9 and 6 probably this is because the line that stretched in both numbers matches perfectly with the diagonal line of the four (maybe adding another set of templates would make this better) also there is 9 detected as 7 this line that is stretched made all things worse

13-DarkMode



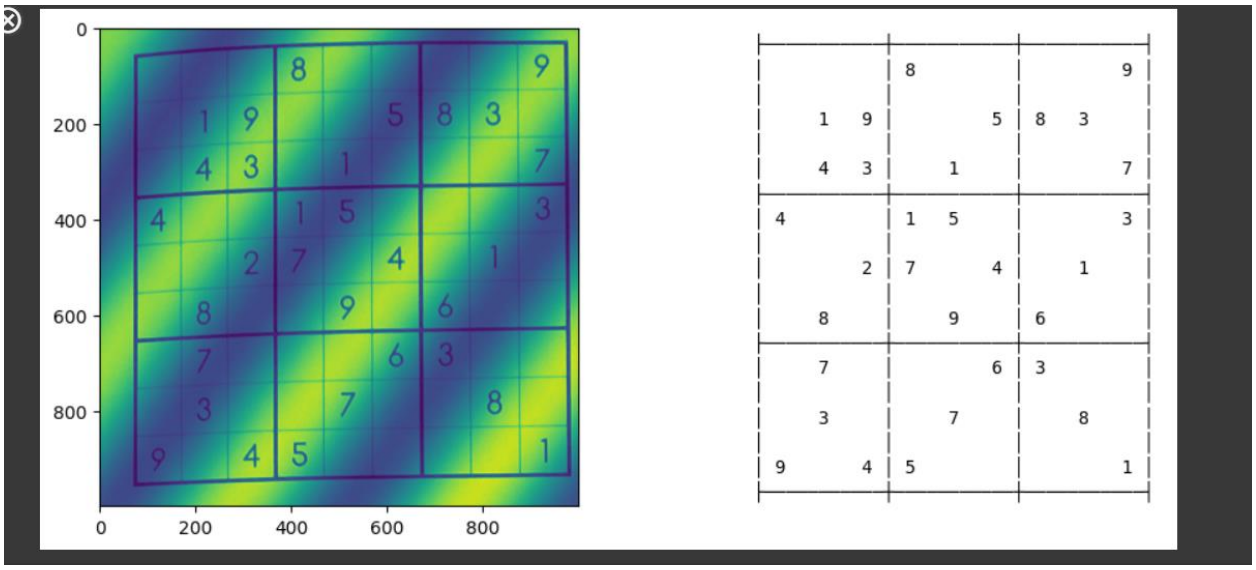
```
+-----------+-----------+-----------+
|           | 4       8 |           |
|     6     |     7     |     1     |
| 7       2 |     9     | 5       4 |
+-----------+-----------+-----------+
|     9     | 7       4 |     3     |
|         7 |     5     | 8         |
|     8     | 9       6 |     5     |
+-----------+-----------+-----------+
| 9       4 |     1     | 7       8 |
|     7     |     6     |     4     |
|           | 2       7 |           |
+-----------+-----------+-----------+
```

Dark mode is the best



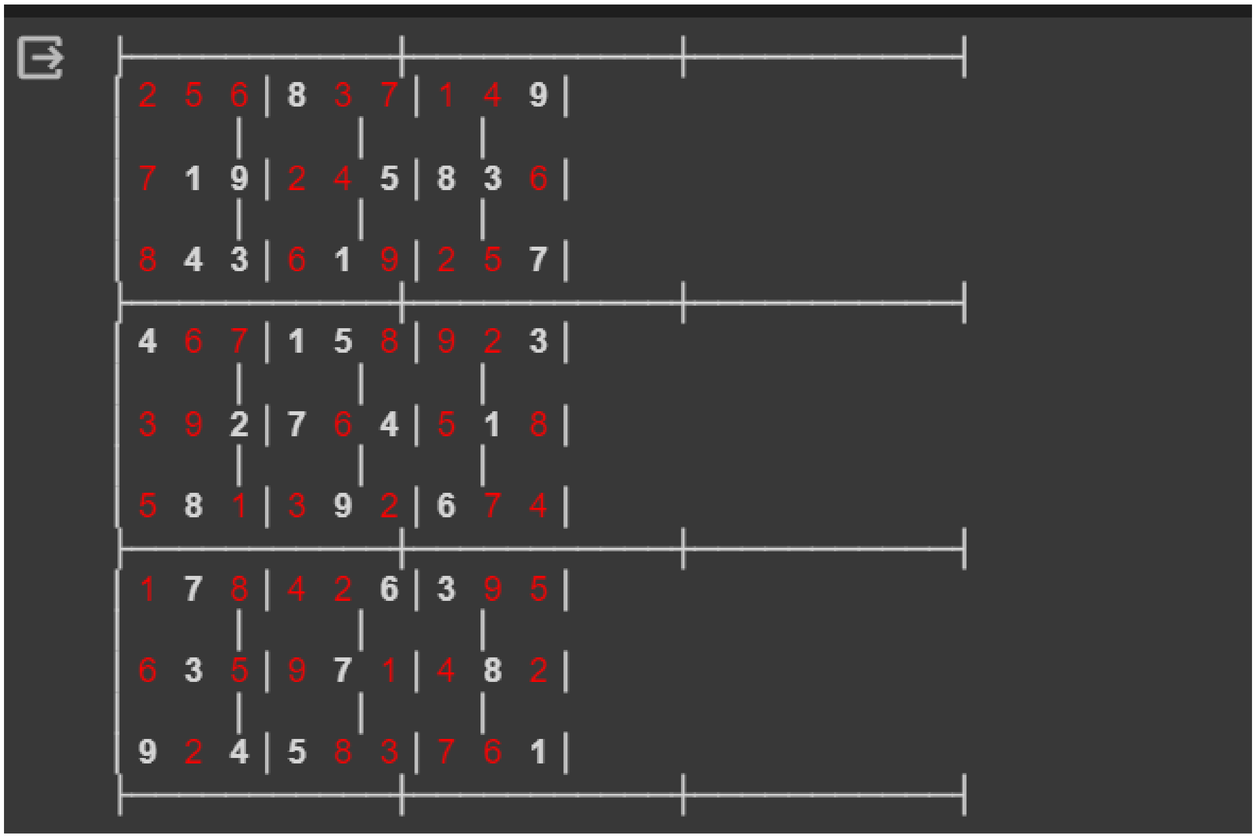
```
+-------------------+-------------------+-------------------+
| 1 5 9 | 4 3 8 | 2 7 6 |
| 4 6 8 | 5 7 2 | 9 1 3 |
| 7 3 2 | 6 9 1 | 5 8 4 |
+-------------------+-------------------+-------------------+
| 2 9 5 | 7 8 4 | 6 3 1 |
| 6 4 7 | 1 5 3 | 8 2 9 |
| 3 8 1 | 9 2 6 | 4 5 7 |
+-------------------+-------------------+-------------------+
| 9 2 4 | 3 1 5 | 7 6 8 |
| 5 7 3 | 8 6 9 | 1 4 2 |
| 8 1 6 | 2 4 7 | 3 9 5 |
+-------------------+-------------------+-------------------+
```

14-Sine



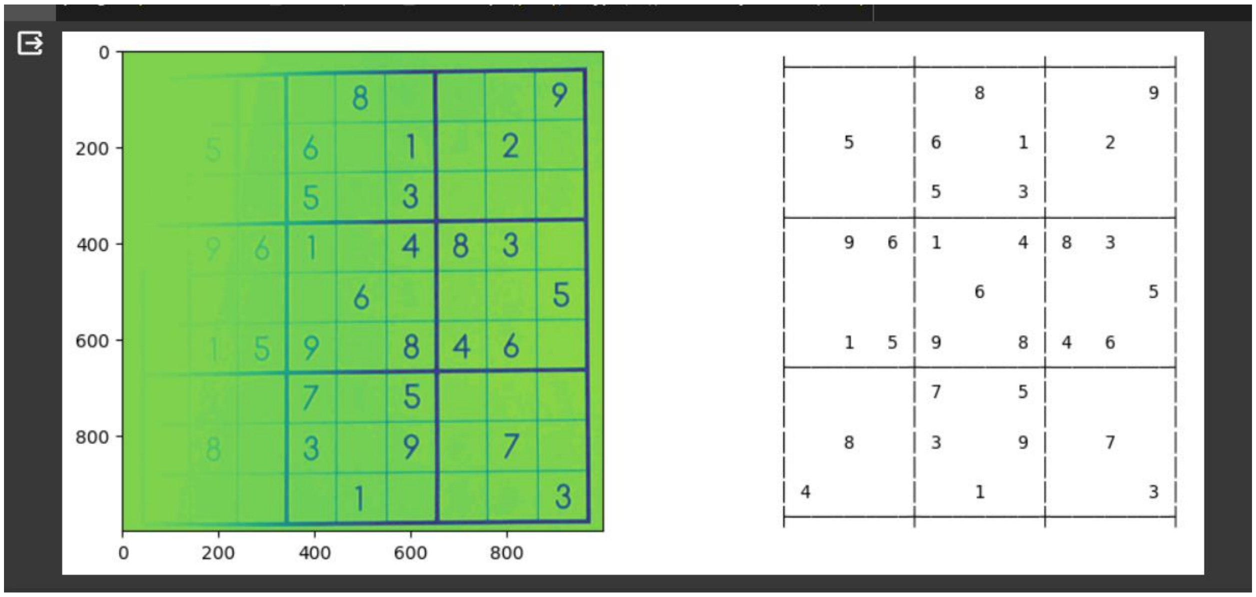
```
      |8     |    9
  1 9 |    5 |8 3
  4 3 |  1   |    7
------+------+------
4     |1 5   |    3
    2 |7   4 |  1
  8   |  9   |6
------+------+------
  7   |    6 |3
  3   |  7   |  8
9   4 |5     |    1
```

I think I got a new favorite



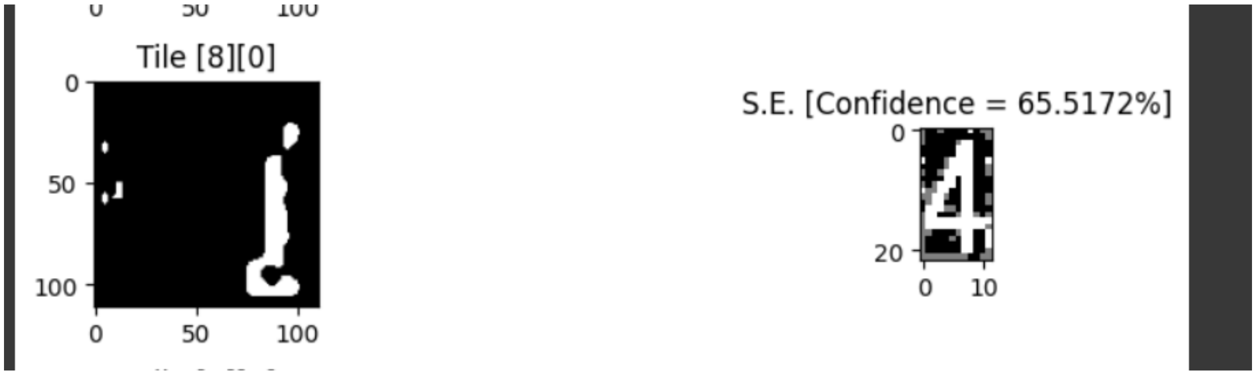
```
2 5 6 | 8 3 7 | 1 4 9 |
7 1 9 | 2 4 5 | 8 3 6 |
8 4 3 | 6 1 9 | 2 5 7 |
4 6 7 | 1 5 8 | 9 2 3 |
3 9 2 | 7 6 4 | 5 1 8 |
5 8 1 | 3 9 2 | 6 7 4 |
1 7 8 | 4 2 6 | 3 9 5 |
6 3 5 | 9 7 1 | 4 8 2 |
9 2 4 | 5 8 3 | 7 6 1 |
```

15-GoneWithTheWind

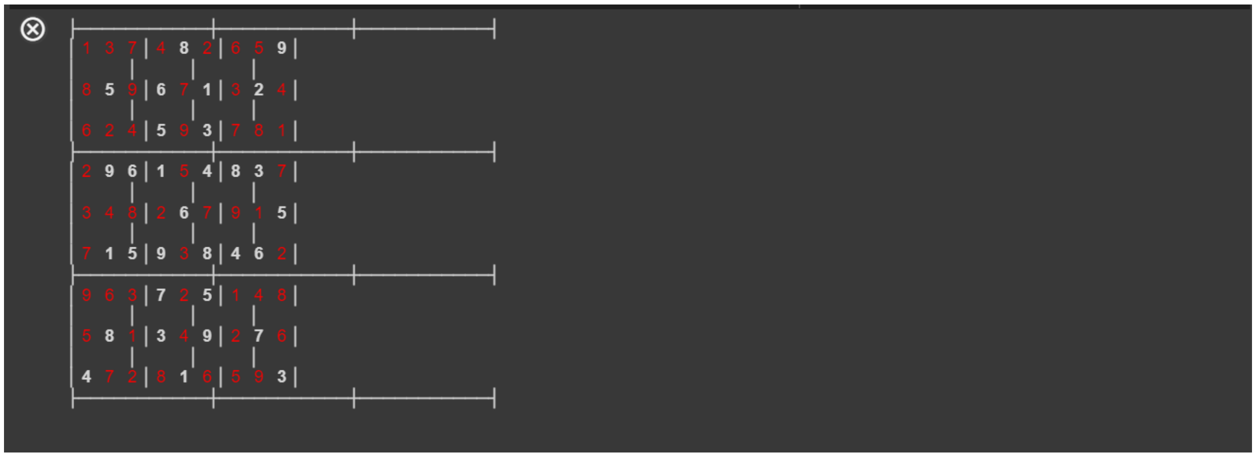


```
|       |   8   |     9 |
|   5   | 6   1 |   2   |
|       | 5   3 |       |
---------------------------
|   9 6 | 1   4 | 8 3   |
|       |   6   |     5 |
|   1 5 | 9   8 | 4 6   |
---------------------------
|       | 7   5 |       |
|   8   | 3   9 |   7   |
| 4     |   1   |     3 |
```

It was perfect but the bottom left corner must ruin it

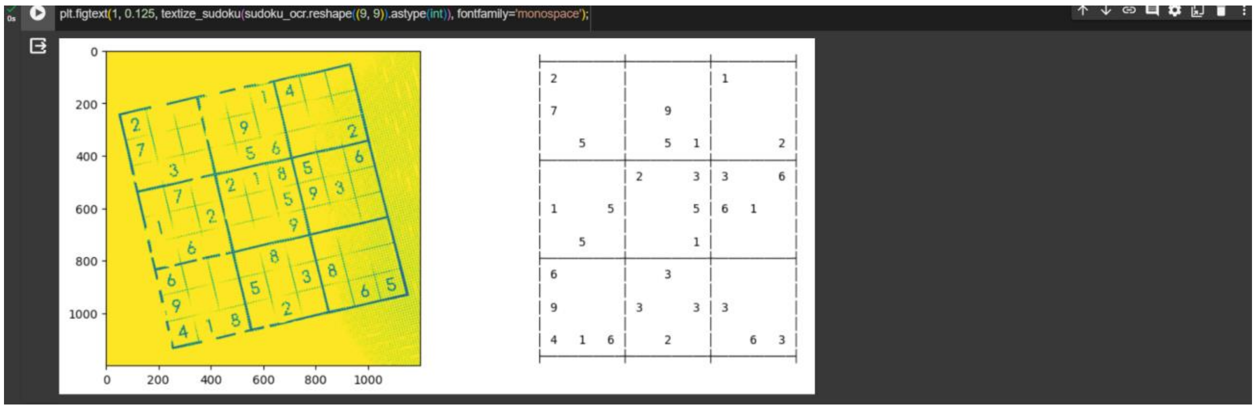


We got this part due to trying to remove shadow, as a part of image is brighter than the other so our algorithm detected that part that slightly brighter as an foreground therefore threshed all the other pixels to 0 and made it white, although this four is detected the puzzle was solved



```
| 1 3 7 | 4 8 2 | 6 5 9 |
| 8 5 9 | 6 7 1 | 3 2 4 |
| 6 2 4 | 5 9 3 | 7 8 1 |
-------------------------
| 2 9 6 | 1 5 4 | 8 3 7 |
| 3 4 8 | 2 6 7 | 9 1 5 |
| 7 1 5 | 9 3 8 | 4 6 2 |
-------------------------
| 9 6 3 | 7 2 5 | 1 4 8 |
| 5 8 1 | 3 4 9 | 2 7 6 |
| 4 7 2 | 8 1 6 | 5 9 3 |
```

16-SomethingWentTerriblyWrongHere



```
plt.figtext(1, 0.125, textize_sudoku(sudoku_ocr.reshape((9, 9)).astype(int)), fontfamily='monospace');
```

```
+-------+-------+-------+
| 2     |       | 1     |
| 7     |   9   |       |
|   5   |   5 1 |     2 |
+-------+-------+-------+
|       | 2   3 | 3   6 |
| 1   5 |     5 | 6 1   |
|   5   |     1 |       |
+-------+-------+-------+
| 6     |   3   |       |
| 9     | 3   3 | 3     |
| 4 1 6 |   2   |   6 3 |
+-------+-------+-------+
```

The digits that are disjoint messed the whole process up (some of digits which were closed are detected right)

References

https://www.sciencedirect.com/science/article/abs/pii/0734189X85900167

https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv

https://stackoverflow.com/questions/44047819/increase-image-brightness-without-overflow/44054699#44054699