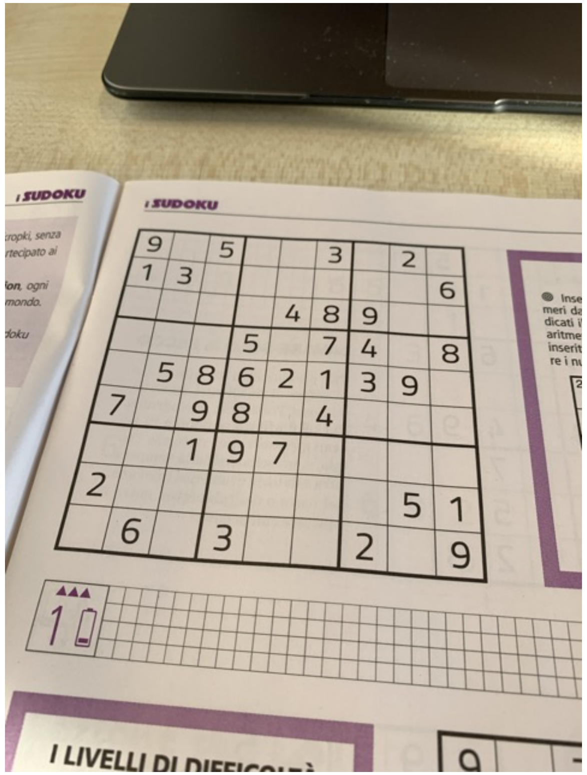
UNIVERSIDADE DE AVEIRO

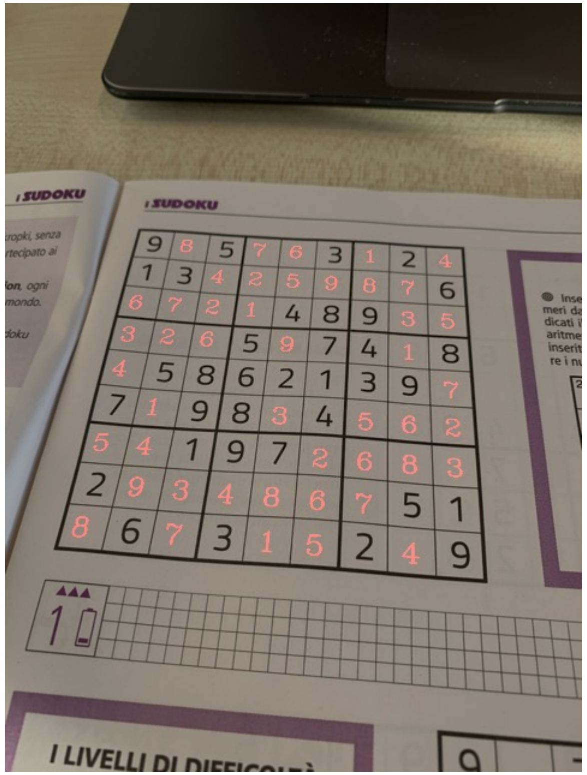
SUDOKU SOLVER

Petrina Stefano - 112056

Goal of the project

The goal of the project is to receive as input an image containing a sudoku, detect it, find a solution and print the missing numbers in the original image.



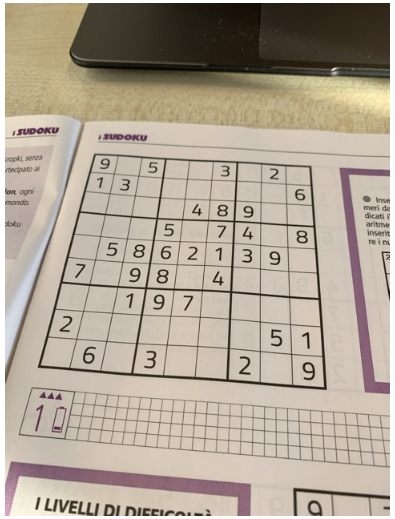


SUMMARY

1. Find the contours of the board
2. Perspective transformation
3. Split the board
4. Recognise the numbers
5. Solve the sudoku
6. Display the result

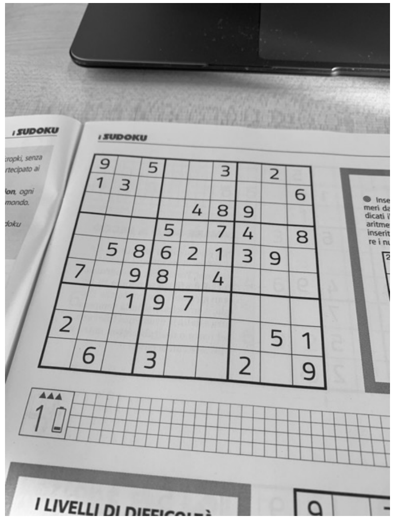
Find the Contours

Import the image



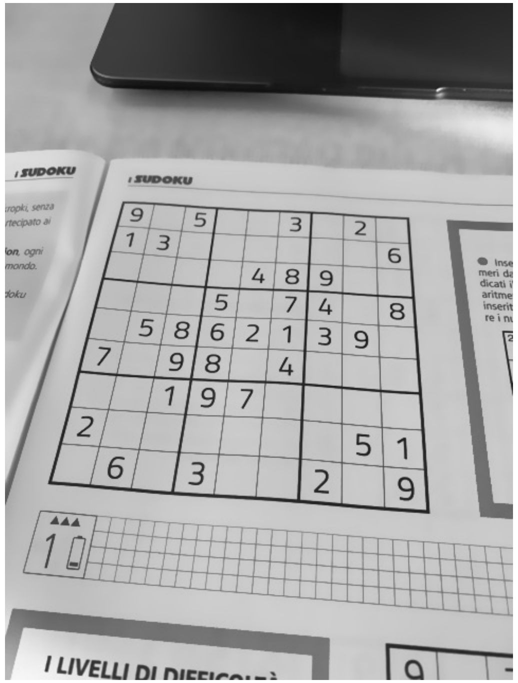
Convert to Grayscale

```
gray=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```



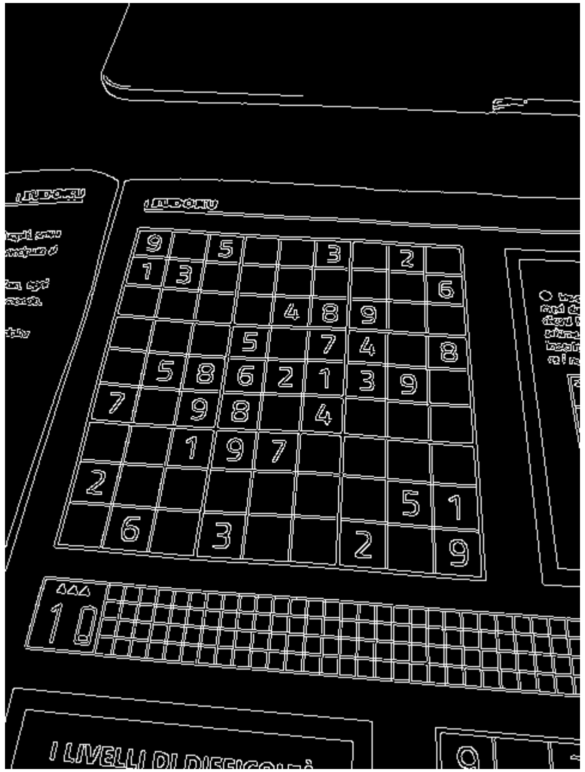
Apply a bilateral filter

```
bfilter = cv2.bilateralFilter(gray, 13, 20, 20)
```



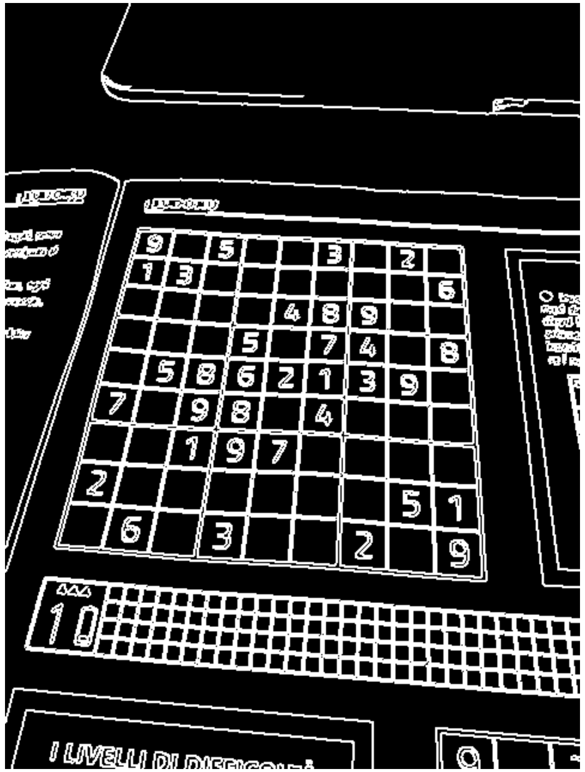
Canny edge detector

```
canny = cv2.Canny(bfilter, 30, 200)
```



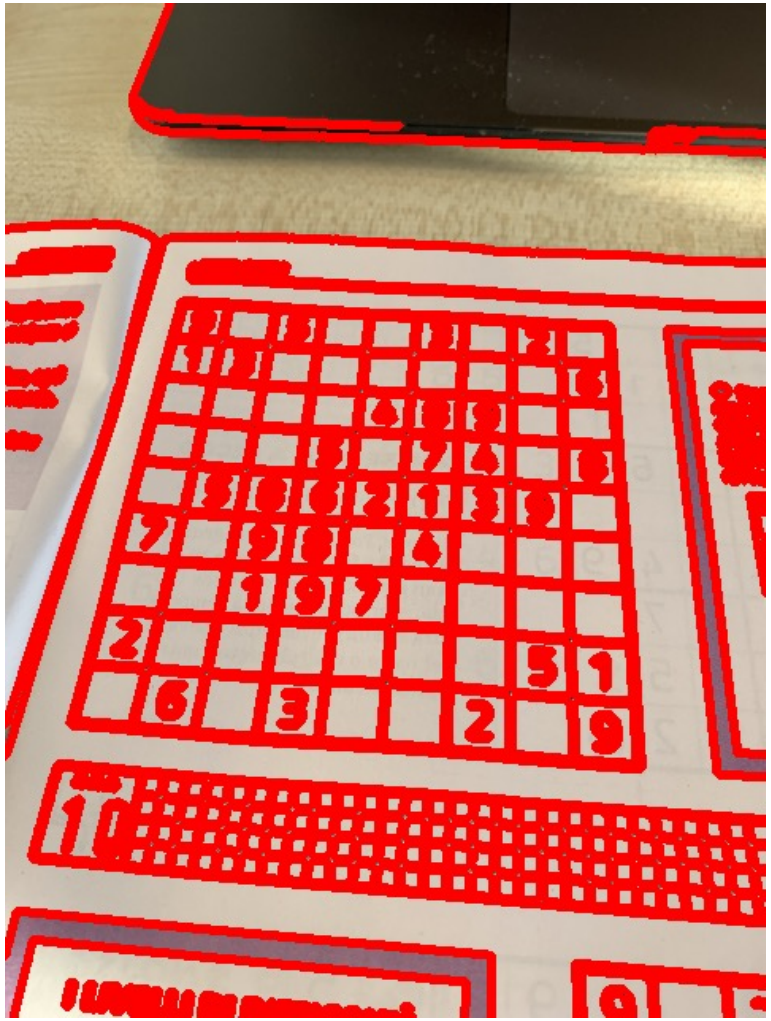
Dilation

```
k=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(2,2))
imageDil = cv2.dilate(canny, k, iterations=1)
```



Find the contours

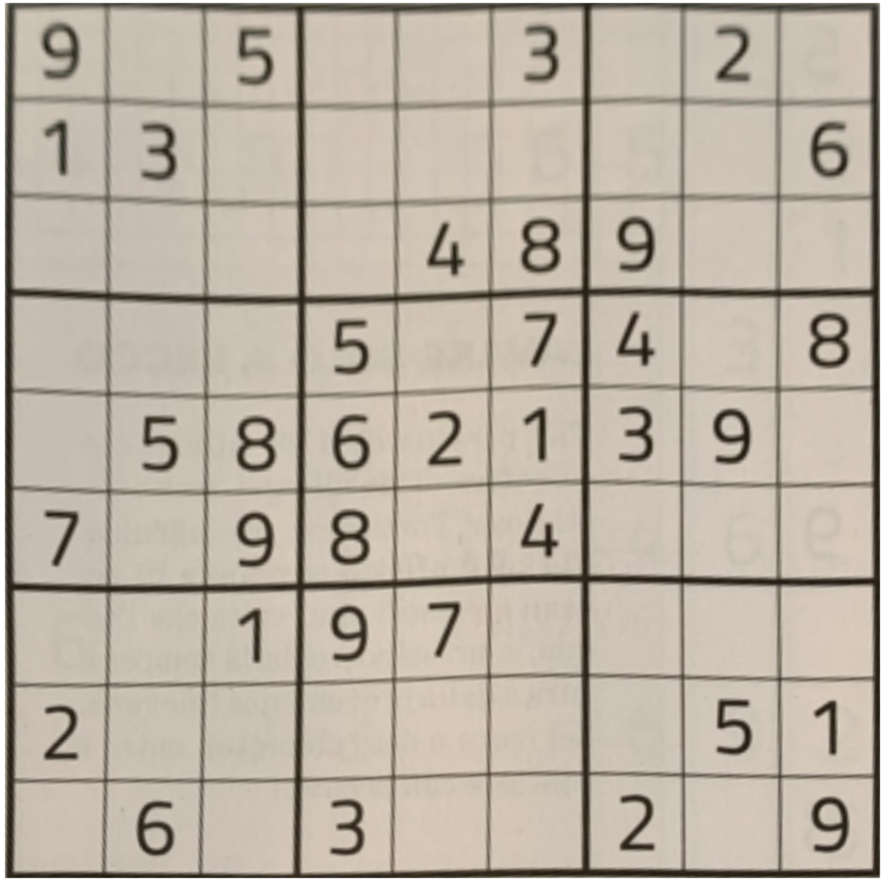
```
keypoints = cv2.findContours(imageDil.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
newimg = cv2.drawContours(img.copy(), contours, -1, (0, 0, 255), 3)
```



Perspective Transformation

After taking the 15 largest contours, I find the rectangular ones and I apply this function.

```
def perspective_img(img, loc):
    '''
    INPUT: image and location of interesting region
    OUTPUT: selected region with a perspective transformation
    '''
    h = 900
    w = 900
    p1 = np.float32([loc[0], loc[3], loc[1], loc[2]])
    p2 = np.float32([[0, 0], [w, 0], [0, h], [w, h]])
    # Apply Perspective Transform Algorithm
    matrix = cv2.getPerspectiveTransform(p1, p2)
    result = cv2.warpPerspective(img, matrix, (w, h))
    return result
```



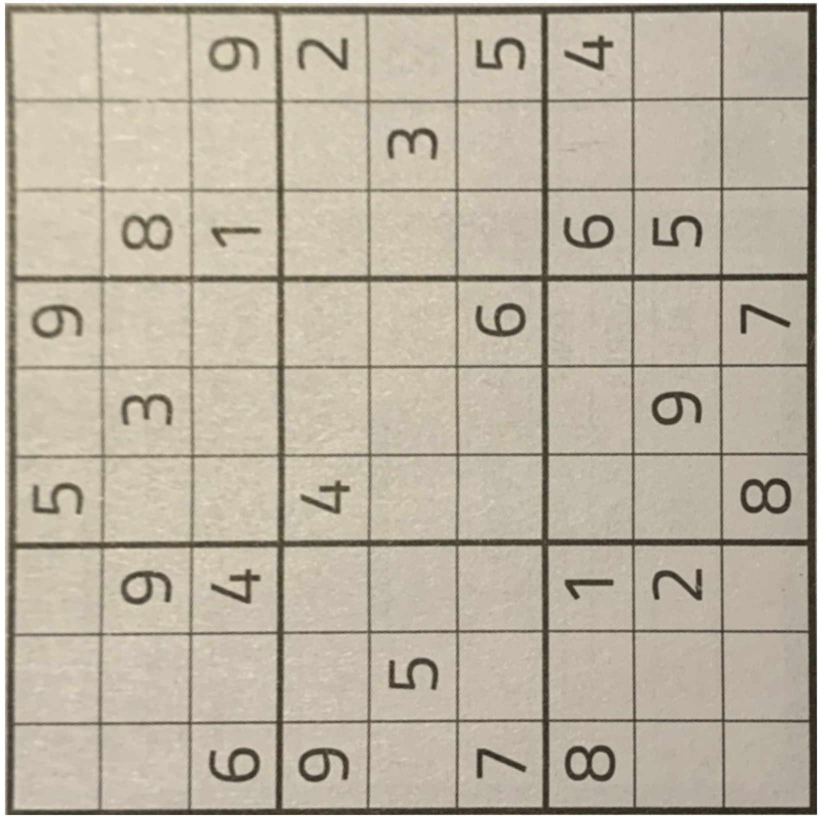
9		5			3		2	
1	3							6
				4	8	9		
			5		7	4		8
	5	8	6	2	1	3	9	
7		9	8		4			
		1	9	7				
2							5	1
	6		3			2		9

Problem

After the transformation,the sudoku could be rotated.

To solve this problem I applied the following lines:

```
cv2.imshow("Press 'r' to rotate by 90 Degrees", result)
k=cv2.waitKey(0)
while k != ord('q’):
   if k==ord('r’):
      result = imutils.rotate(result, 90)
      cv2.imshow("Press 'r' to rotate by 90 Degrees", result)
      k=cv2.waitKey(0)
```



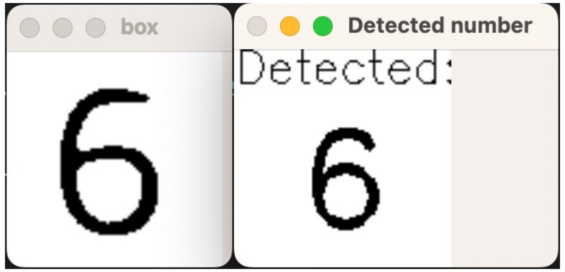
Split boxes

Using this function I divide the sudoku into 81 cells.

```
def find_boxes(board):
'''
INPUT: sudoku board
OUTPUT: 81 elements representing every cell
'''
rows = np.vsplit(board,9) # vertical split
elements = []
for r in rows:
cols = np.hsplit(r,9) # horizontal split for every
row
for cell in cols:
size_cell=cell.shape[0]
cell = cv2.resize(cell,(size_cell,
size_cell))/255.0
elements.append(cell)
return elements
```

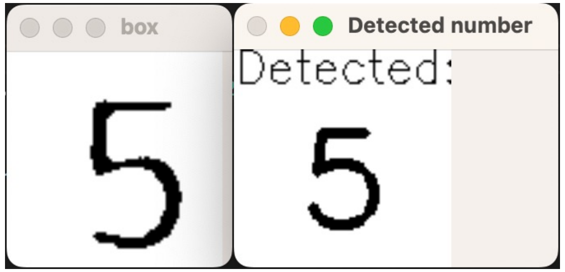
Recognise the numbers

Using the pytesseract library, I try to recognize the numbers.
For the ones detected in the wrong way, I put the right number using the keyboard.



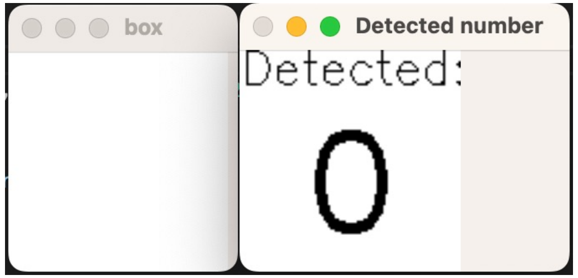
Detected:
6

✓



Detected:
5

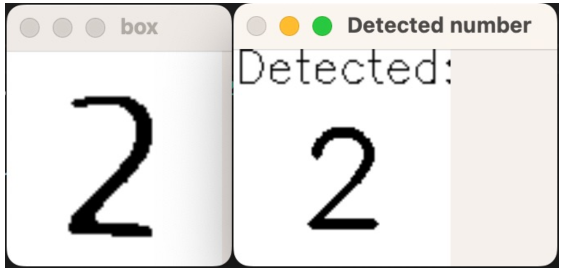
✓



Detected:
0

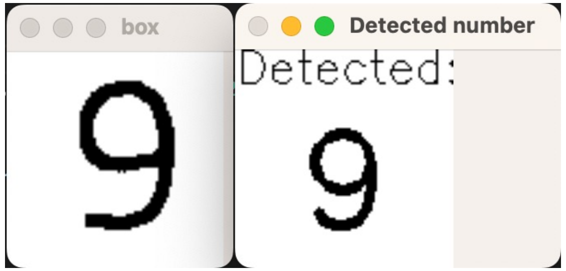
✓

(0 represents an empty cell)



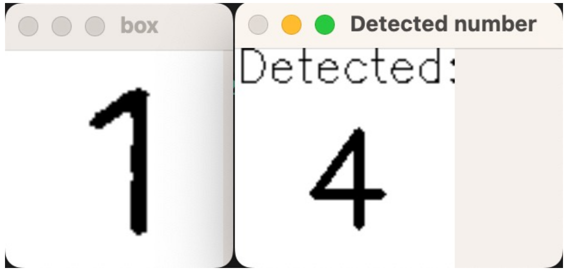
Detected:
2

✓



Detected:
9

✓



Detected:
4

✗

Problem

The detection at the beginning was very inaccurate.

To increase the precision I applied two changes:

1) Crop the image (to remove the black contour that sometimes remains in the image)

```
box2 = box1[10:box1.shape[0]-10, 10:box1.shape[1]-10]
```

2) Threshold (to make the image sharper)

```
_,thresh1 = cv2.threshold(box2,100,255,cv2.THRESH_BINARY)
```

Solve the sudoku

Function used to solve the sudoku:

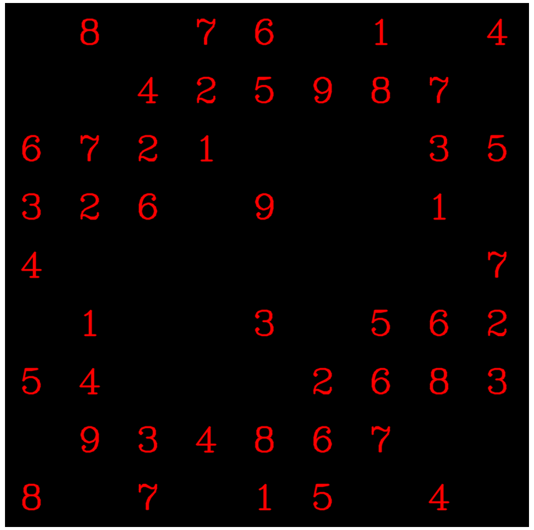
1. `empty_cell(sudoku)` → It finds the first empty cell.
2. `solvable(sudoku, number, position)` → It detects if the sudoku is correct.
3. `solve_sudoku(sudoku)` → It uses the previous two functions to solve the sudoku trying to put a number from 1 to 9 in the empty cells.
4. `get_solved_sudoku(sudoku)` → If the sudoku has been solved, it returns the board.

Display numbers

After getting the solved sudoku, I want to display the numbers in the original image. To do so I make the following steps:

1) Create a blank mask and put in it only the solved digits

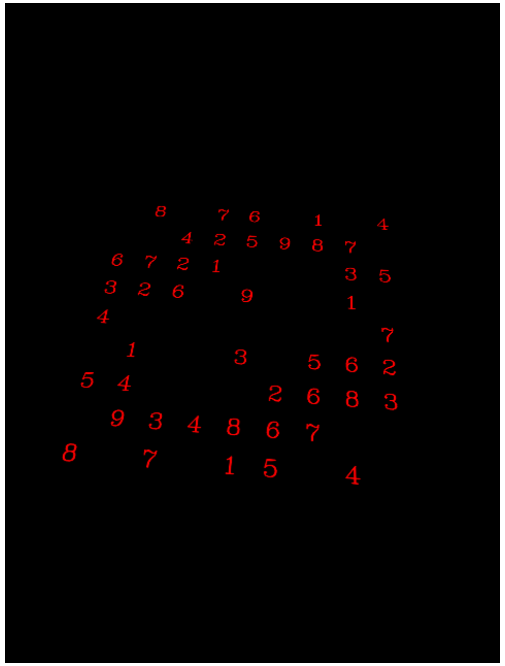
```
mask = np.zeros_like(result)
sudoku_mask = displayNumbers(…) # displays solved numbers
```



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

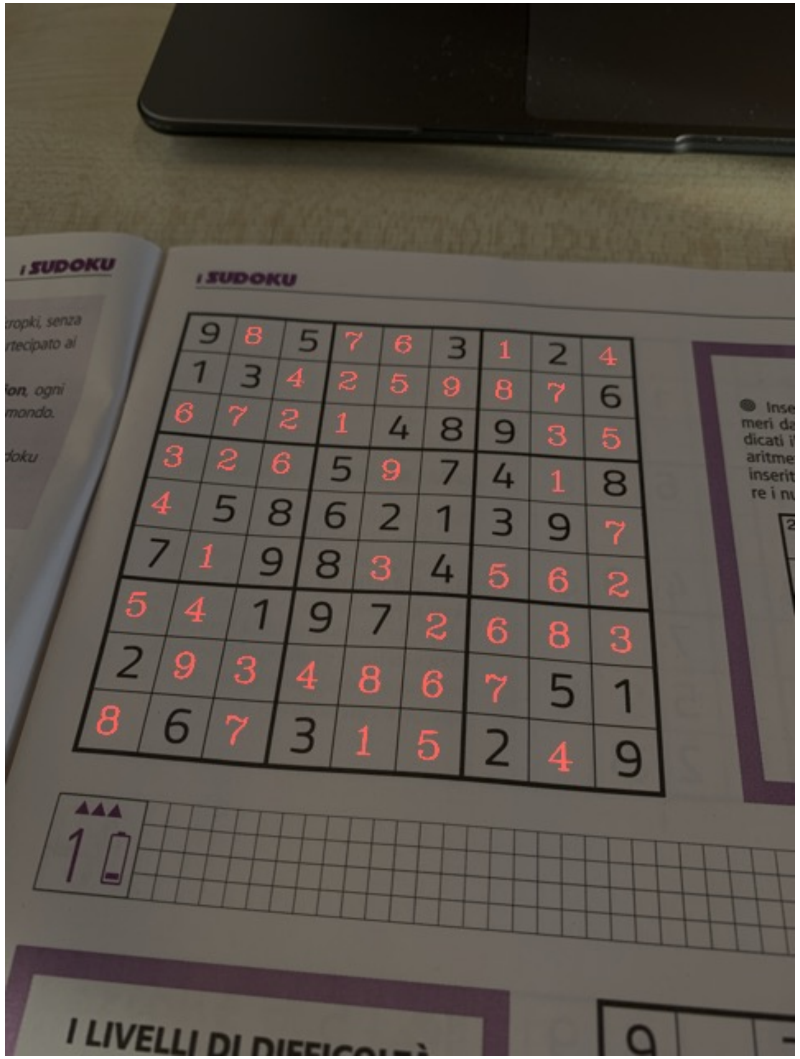
2) Apply the inverse of the perspective transform used before to the mask. In this way the mask fits perfectly the original image

```
inv_mask = get_InvPerspective(img, sudoku_mask, location)
```



3) Combine the original image with the mask obtained in the step 2

```
combined = cv2.addWeighted(img, 0.5, inv_mask, 1, 0)
```



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

P.S. : if the program does not find a solution (it could happen if the a detected number is wrong), it prints ‘Not solved’.

Thanks for the attention