PROJECT in Image and Video Processing
EC348

Robust method for Sudoku solving through Image Processing

Submitted by

B S Trinesh Reddy (181EC108)
B Bharath Reddy (181EC110)
V Sai Krishna (181EC153)

VI SEM B.Tech

Under the guidance of

Dr. Shyam lal
Dept of EC, NITK Surathkal

in partial fulfillment for the award of the degree

of

Bachelor of Technology

at

Department of Electronics and Communication

National Institute of Technology Karnataka, Surathkal.

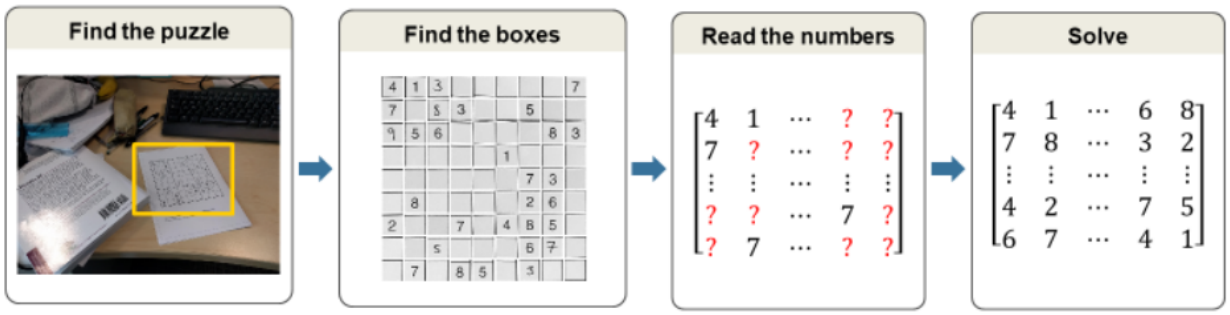
April 2021

Table of Contents

1. Abstract:

The method involves an image-based sudoku solver. This solver is capable of solving a sudoku directly from an image captured from any digital camera. After applying appropriate pre-processing to the acquired image we use contours to recognize the enclosing box of the puzzle. A virtual grid is then created to identify the digit positions. A CNN architecture model is created with MNIST digit dataset and used for digit recognition. The actual solution is computed using a backtracking algorithm.

Experiments conducted on various types of sudoku questions demonstrate the efficiency and robustness of our proposed approaches in real-world scenarios. This algorithm is found to be capable of handling cases of translation, perspective, illumination gradient, scaling, and background clutter.



2. Introduction:

In real life we come across Sudoku puzzles of varying difficulty levels in newspapers and other text and digital media. It is a common leisure activity for a lot of people. However, it is observed that the solution is not always immediately available for verification. In most cases, people have to wait till the next day to check the solutions of the Sudoku they just solved. Hence our motivation for this project was to develop the solution for the puzzle by taking an input image of the sudoku and generating an immediate output image of solved sudoku using a CNN architecture model.

3. Algorithm

This algorithm is designed specifically to solve a Sudoku puzzle containing a 9x9 grid of numbers. The task of the algorithm is divided into two sections.

Firstly, it should accurately locate the grid position in the image while taking care of problems of background clutter, scaling, translation, rotation and perspective skew.

The second part of the algorithm must then locate those positions in the grid where there are numbers and recognize them with precision. A recursive backtracking algorithm will then solve the puzzle and the solved sudoku is displayed as output.

4. Extracting the puzzle grid

a. Smoothing and thresholding

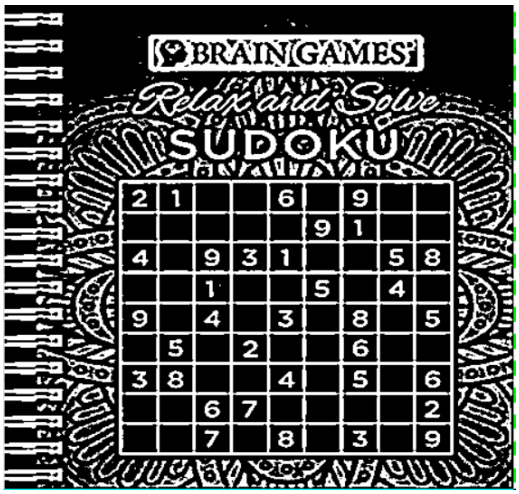
First we need to preprocess our input image.Preprocessing involves converting into gray scale applying gaussian blur and applying adaptive thresholding resulting in robust grid corner extraction.

Preprocessing code

```
#### 1 - Preprocessing Image
def preProcess(img):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # CONVERT IMAGE TO GRAY SCALE
    imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)  # ADD GAUSSIAN BLUR
    imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, 1, 1, 11, 2)  # APPLY ADAPTIVE THRESHOLD
    return imgThreshold
```

Original image

Preprocessed image

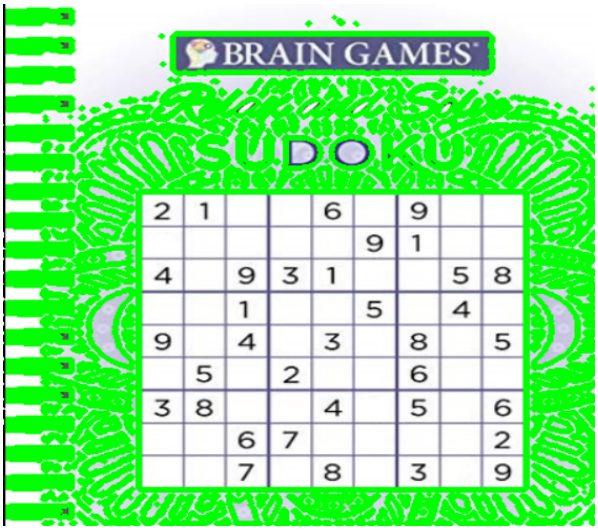


b. Identifying all the contours

Image contouring is the process of identifying structural outlines of objects in an image which can help us identify the shape of the object. We draw contours on the image using opencv **drawContours** function.

Code for drawing contours

```
contours, hierarchy = cv2.findContours(imgThreshold, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # FIND ALL CONTOURS
cv2.drawContours(imgContours, contours, -1, (0, 255, 0), 3) # DRAW ALL DETECTED CONTOURS
```

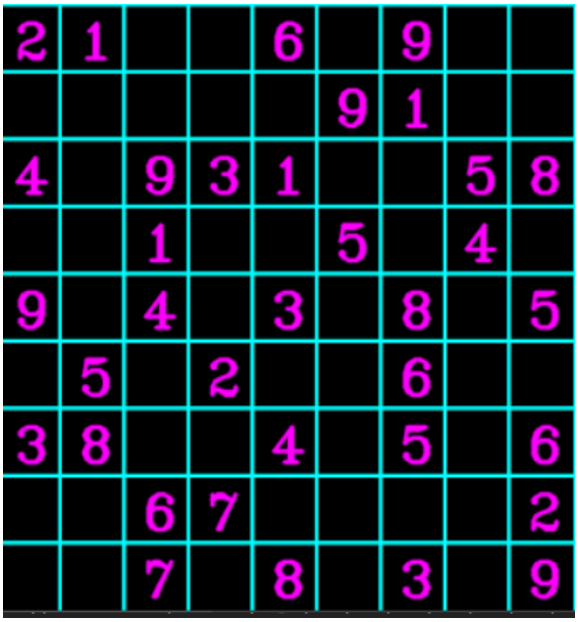


All the identified contours are drawn in 'green lines' in the above image.

c. Extracting the sudoku grid

We assume that the biggest area contour with 4 sides in the input image is our sudoku puzzle. After identifying the sudoku puzzle in our image we mark the 4 corner points.

Corner points of the new image are marked in the above input image as shown in the above figure and isolated the puzzle from the input image and stored as a new image as shown in below figure.



This image is used for identifying the numbers in the sudoku and solving the sudoku by backtracking algorithm.

5. Recognition of digits

a. Extracting the given digits form grid

We know our sudoku grid consists of smaller squares which are 9 x 9=81 in number.Each smaller grid has a number in range [1-9] or empty cell.We perform horizontal and vertical split on our extracted sudoku puzzle with factor 9 and we get 9 x 9=81 new images.These 81 images we fed as inputs to our CNN model to recognize the digits.

Code for splitting grid into 81 smaller images

```
#### 4 - TO SPLIT THE IMAGE INTO 81 DIFFRENT IMAGES
def splitBoxes(img):
    rows = np.vsplit(img,9)
    boxes=[]
    for r in rows:
        cols= np.hsplit(r,9)
        for box in cols:
            boxes.append(box)
    return boxes
```

b. CNN Model

Convolutional operations have been prominent for many years in image recognition fields. The basic idea is to use a filter or a kernel to slide over the image and produce a feature map as an output. Taking the example of a 4x4 image, we can use a 3x3 filter to convolve over the image and produce a 2x2 feature map.

This can mathematically be seen as:

$$h_{i,j} = \sum_{k=1}^{m} \sum_{l=1}^{m} w_{k,l} x_{i+k-1,j+1}$$

where h is the convolved feature map, w is the convolution kernel, x is the input image and m is the width and height of the kernel.

The advantage of using convolutions is that the nodes in a layer only have to be connected to a specific region of nodes in the adjacent layer.

There are several different activation functions that are useful for deep learning. The backpropagation will have much different outcomes depending on our choice of activation function.

We trained our model on a data set containing images of numbers from 1-9.

Our model has 4 convolution layers, 2 dropout layers, 2 dense layers, 2 max pool layers and 1flatten layer.

Conv layer: A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input image.

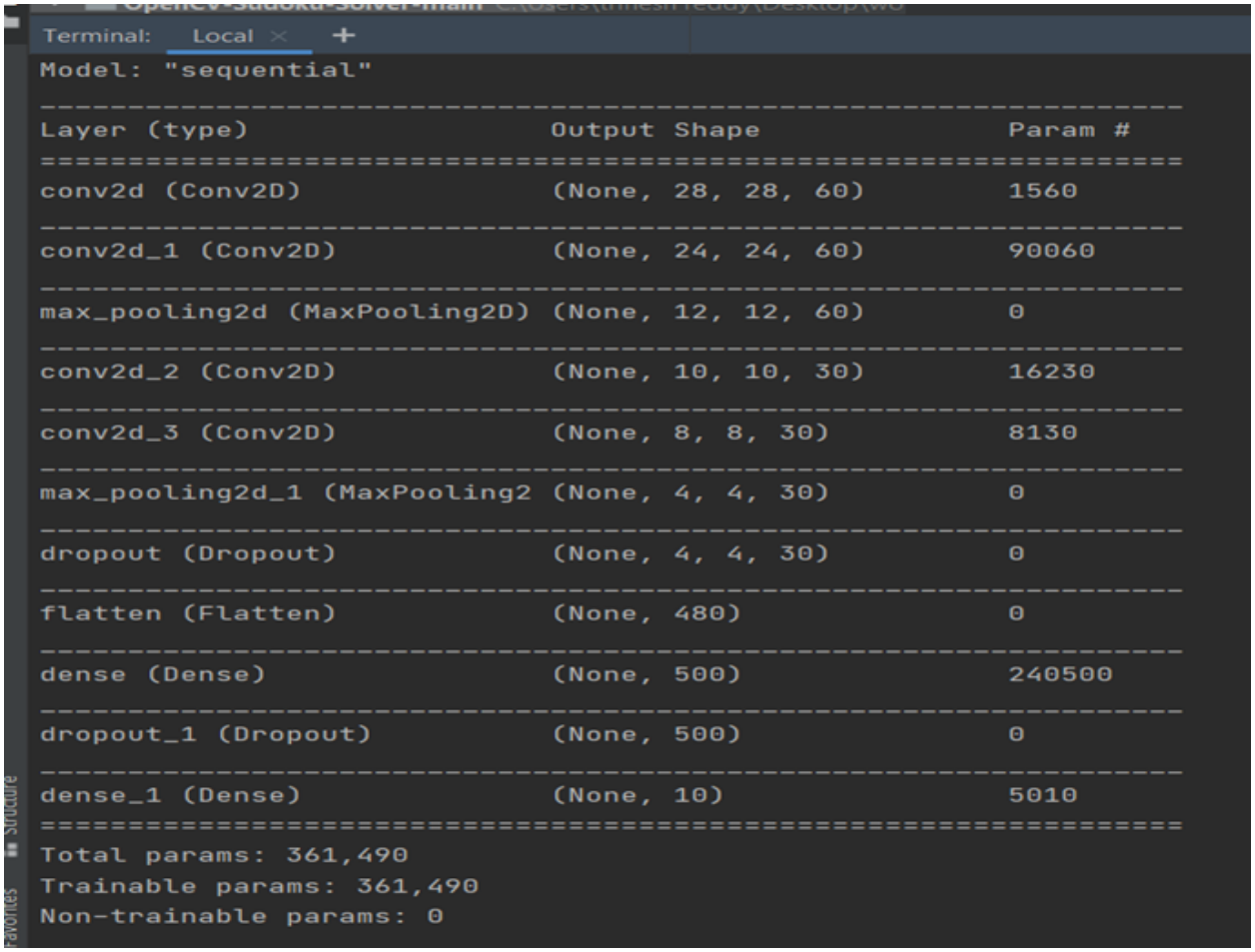
Dropout layer: The purpose of the dropout layer is to remove the noise that may be present in the input of neurons. These are used to reduce overfitting in neural network

Max pool layer: The pooling layer is used for applying max pooling operations on temporal data. It reduces computational cost by reducing the number of parameters to learn as well as overfitting.

Flatten layer: Flattening involves transforming the entire pooled feature map matrix into a single column which is then fed to the neural network for processing.

We got test accuracy=0.9916 and test score=0.0226

Model summary



Training the model

```
Epoch 1/10
130/130 [==============================] - 28s 219ms/step - loss: 1.4480 - accuracy: 0.4873 - val_loss: 0.1795 - val_accuracy: 0.9416
Epoch 2/10
130/130 [==============================] - 29s 220ms/step - loss: 0.5263 - accuracy: 0.8290 - val_loss: 0.0811 - val_accuracy: 0.9785
Epoch 3/10
130/130 [==============================] - 30s 228ms/step - loss: 0.3287 - accuracy: 0.8909 - val_loss: 0.0632 - val_accuracy: 0.9803
Epoch 4/10
130/130 [==============================] - 29s 225ms/step - loss: 0.2557 - accuracy: 0.9186 - val_loss: 0.0680 - val_accuracy: 0.9803
Epoch 5/10
130/130 [==============================] - 31s 235ms/step - loss: 0.2017 - accuracy: 0.9374 - val_loss: 0.0344 - val_accuracy: 0.9908
Epoch 6/10
130/130 [==============================] - 30s 233ms/step - loss: 0.1841 - accuracy: 0.9414 - val_loss: 0.0372 - val_accuracy: 0.9883
Epoch 7/10
130/130 [==============================] - 31s 235ms/step - loss: 0.1576 - accuracy: 0.9502 - val_loss: 0.0305 - val_accuracy: 0.9920
Epoch 8/10
130/130 [==============================] - 29s 226ms/step - loss: 0.1376 - accuracy: 0.9592 - val_loss: 0.0302 - val_accuracy: 0.9932
Epoch 9/10
130/130 [==============================] - 30s 227ms/step - loss: 0.1473 - accuracy: 0.9557 - val_loss: 0.0374 - val_accuracy: 0.9865
Epoch 10/10
130/130 [==============================] - 30s 227ms/step - loss: 0.1191 - accuracy: 0.9637 - val_loss: 0.0216 - val_accuracy: 0.9920
Test Score =  0.022608548402786255
Test Accuracy = 0.9916338324546814
```

Results of prediction of different numbers

```
Testing the image 0
[[1.0000000e+00 1.7089628e-19 1.8427077e-16 7.3141769e-23 3.4651921e-21
  2.2178010e-23 5.4006723e-15 1.8935554e-19 1.2510687e-13 1.3854414e-16]]
```

```
Testing the image 1
[[1.02986824e-29 1.00000000e+00 3.21418172e-21 1.62200898e-30
  2.77476006e-24 9.87798319e-33 4.11990119e-29 6.93501476e-25
  3.49730728e-26 1.84104428e-29]]
```

```
Testing the image 2
[[6.2030947e-34 4.6873611e-20 1.0000000e+00 1.1350791e-28 3.5021605e-37
  0.0000000e+00 0.0000000e+00 7.5219355e-35 2.5757921e-27 2.4058719e-34]]
```

```
Testing the image 3
[[7.0863892e-37 2.7907501e-29 8.2870574e-33 1.0000000e+00 0.0000000e+00
  1.6464293e-20 6.9269053e-38 3.2007195e-23 2.0130339e-21 9.5094957e-26]]
```

```
Testing the image 4
[[3.2328275e-26 1.6638194e-25 0.0000000e+00 1.0597946e-37 1.0000000e+00
  8.3922033e-32 5.6540367e-20 0.0000000e+00 3.1709915e-24 3.9313761e-27]]
```

```
Testing the image 5
[[1.3192820e-31 7.7835270e-30 1.5025367e-36 4.9165517e-19 5.0543777e-37
  1.0000000e+00 2.0713404e-21 7.5861713e-34 1.2956656e-32 7.0251745e-32]]
```

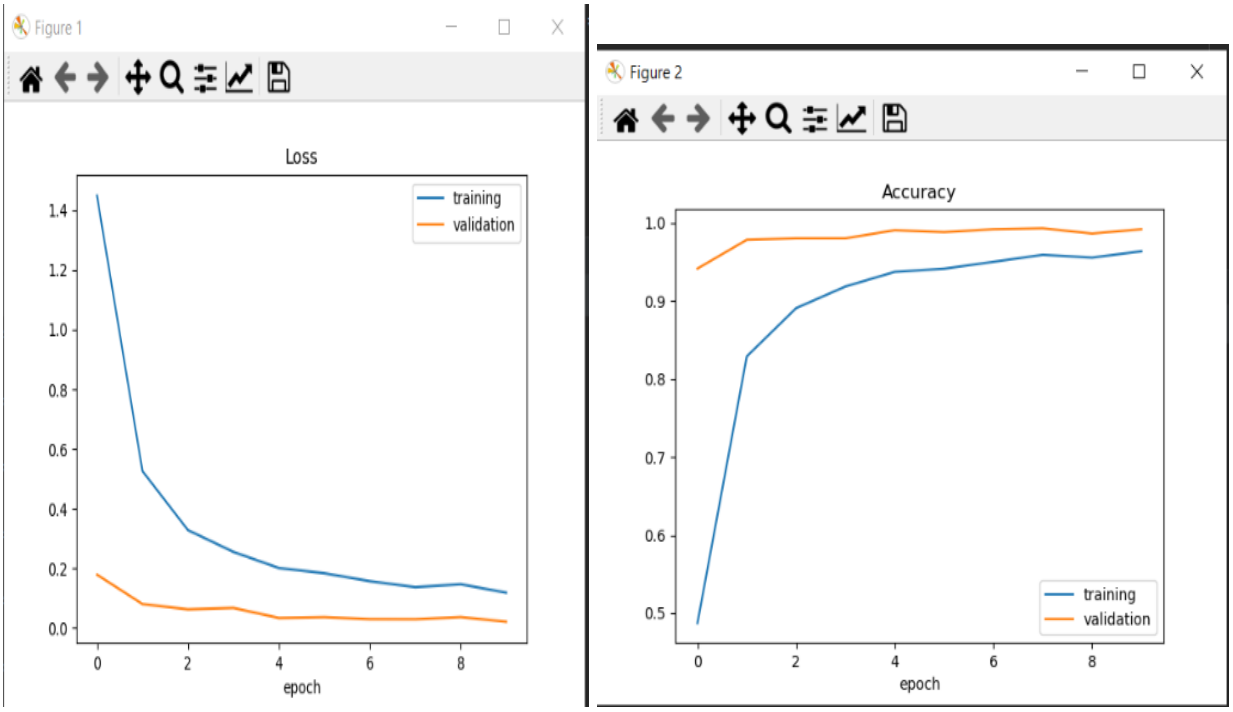
```
Testing the image 6
[[1.1423896e-21 1.4306522e-32 0.0000000e+00 4.0790288e-34 3.6606593e-27
  6.8409653e-28 1.0000000e+00 0.0000000e+00 1.2421252e-22 0.0000000e+00]]
```

```
Testing the image 7
[[3.1961919e-31 6.2352394e-27 3.4255323e-33 2.1991300e-25 0.0000000e+00
  8.5260993e-36 0.0000000e+00 1.0000000e+00 1.7586566e-34 2.3588647e-31]]
```

```
Testing the image 8
[[0.0000000e+00 3.7001836e-33 0.0000000e+00 8.6745417e-23 1.4481808e-37
  2.8878302e-29 1.2499713e-32 1.9402461e-38 1.0000000e+00 6.0577625e-33]]
```

```
Testing the image 9
[[4.0427733e-22 4.4234320e-37 5.6874219e-25 8.8842142e-31 1.1321273e-25
  1.4981638e-29 0.0000000e+00 5.0141952e-25 9.3804201e-30 1.0000000e+00]]
```

Loss and Accuracy graphs



6. Solving the sudoku

After identifying each digit in smaller grid through our CNN model we store them in new 9 x 9 matrix.We now apply backtracking sudoku solving algorithm to solve the puzzle

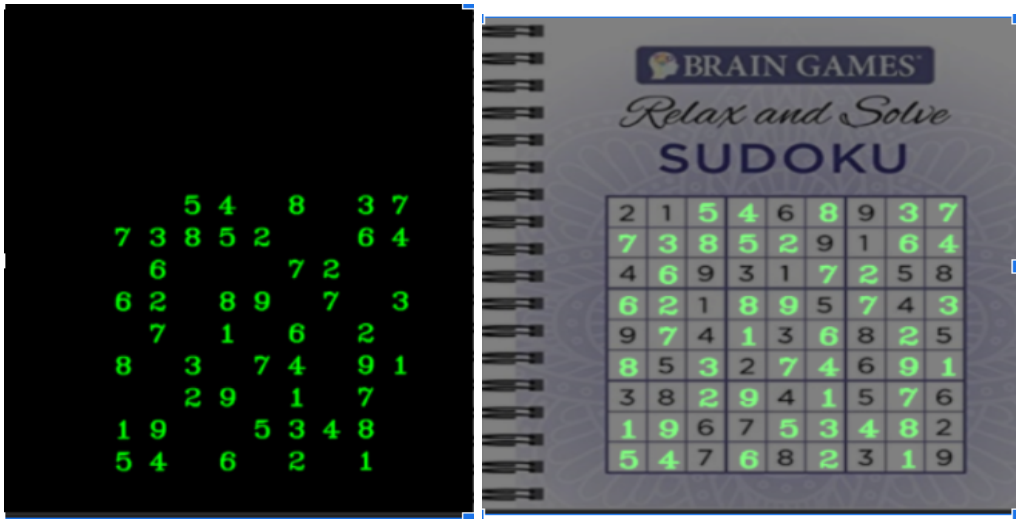
Algorithm for sudoku solving:

I. We store sudoku in a 9 x 9 matrix.
II. User defined function “issafe()” which tells us if it is feasible to put a given digit in a given empty slot.
III. We loop through the matrix row wise, if we find an empty slot we break out of loop.
IV. In an empty slot we start putting digits from (1-9) if it is safe to put a digit in that slot we put it and loop again through the matrix until we find the next empty slot.
V. If at any slot it is not feasible to insert any digit in (1-9) we backtrack and see for the other solution.
VI. If a given puzzle is valid, then it is guaranteed to obtain a solution.

7. **Overlaying our solution on original image:**

   To overlap the sudoku solution on our original image first we need to store the positions of 4 corner points of the sudoku grid in original image.These points is stored in variable **biggest** as shown in code.By using opencv **warpPerspective** function we will be able to put the solved digits on blank image which is same size as original image and at same positions where numbers are missing in original sudoku image. Now with help of opencv **addWeighted** function we will be able to overlay our image which we got from wrapPerspective function on original image.

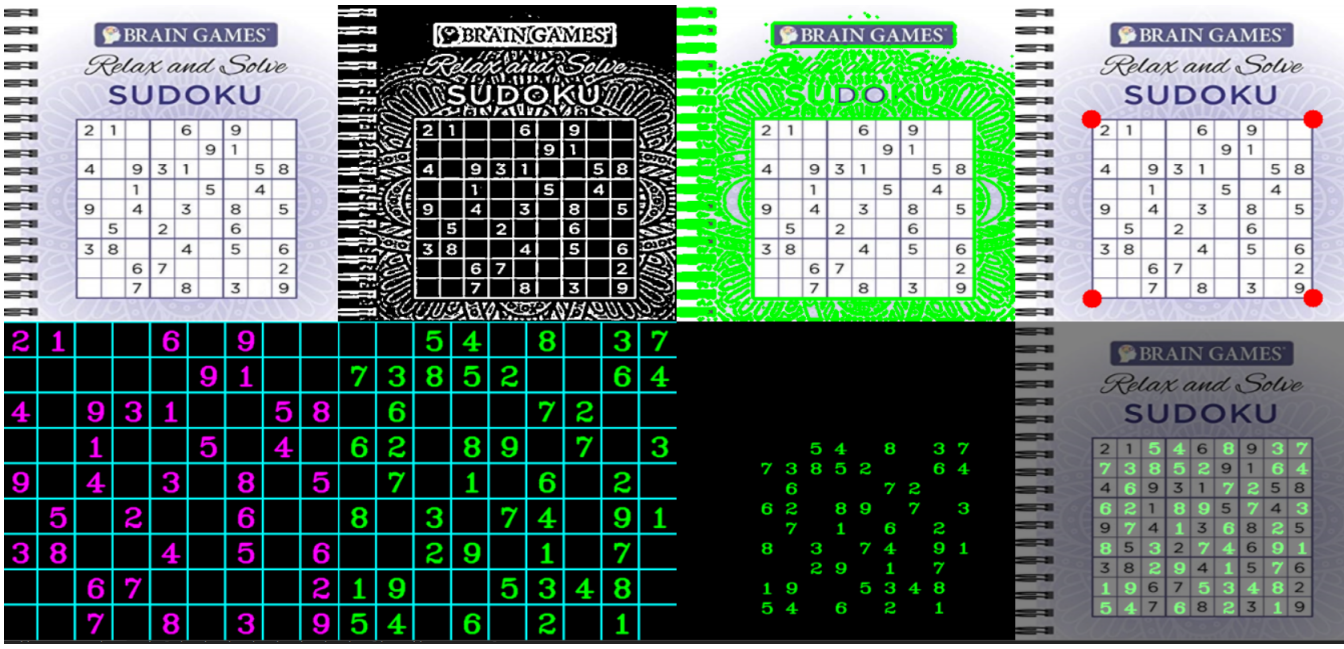
a)WrapPerspective output image **b)Final overlayed image**



Code for overlaying solved digits on original image

```
# #### 6. OVERLAY SOLUTION
pts2 = np.float32(biggest) # PREPARE POINTS FOR WARP
pts1 = np.float32([[0, 0],[widthImg, 0], [0, heightImg],[widthImg, heightImg]]) # PREPARE POINTS FOR WARP
matrix = cv2.getPerspectiveTransform(pts1, pts2)  # GER
imgInvWarpColored = img.copy()
imgInvWarpColored = cv2.warpPerspective(imgSolvedDigits, matrix, (widthImg, heightImg))
inv_perspective = cv2.addWeighted(imgInvWarpColored, 1, img, 0.5, 1)
```

8.Summary



9.limitations:

Our algorithm is not robust to blurring, drastic occlusions and also if any of the four corners of the Sudoku grid are not present in the captured image.The biggest contour with 4 sides must be sudoku grid otherwise we get wrong answers.If any of the digit is wrongly predicted then we will not get the results.So we have to detect the digits with high accuracy.

10. Conclusion

We present a Smart Sudoku Solver that can solve unsolved Sudoku images with a small amount of perspective. Also illumination changes across the images are taken care of.Since the scale of the image also varies from image to image, our algorithm efficiently manages these problems.

11.References

- https://towardsdatascience.com/building-a-sudoku-solving-application-with-computer-vision-and-backtracking-19668d0a1e2
- https://www.pyimagesearch.com/2020/08/10/opencv-sudoku-solver-and-ocr/
- https://www.techwithtim.net/tutorials/python-programming/sudoku-solver-backtracking/
- https://ieeexplore.ieee.org/document/7414762