CS523 Computer Vision
Report

Sudoku Grid Detector using OpenCV and Python 3.7

Deniz Gokcin

May 4, 2020

Introduction

This report explains the implementation details of CS523 Computer Vision Assignment 1, which is about detecting rectangles in a given sudoku puzzle image. The pre-processing of the given image will be explained first and will be followed by the logic behind the rectangle extraction. Lastly, I will add my findings and comments. To run the code, there needs to be a directory which contains the .jpg files. The directory and main.py needs to be at the same level.

Pre-Processing

After examining the pictures in the dataset, I decided to pre-process the image to clear some of the noise so that the detector will perform better. Since we do not need the color information, I first downsampled the image by converting it to gray-scale. Once I got a gray-scale image, I applied a Gaussian Blur to remove some noise and make the extraction of the grid easier. After lots of trials and errors, I decided to use a Kernel size of 5 x 5 for Gaussian Blur. Afterwards, I applied a thresholding algorithm. The idea behind thresholding is that if a pixel has a value smaller than a threshold, it is set to 0, otherwise it is set to a maximum value. Since the images in the dataset had different illumination levels, instead of a global thresholding, I used adaptive thresholding, which determines the threshold for a pixel based on a small region around it.

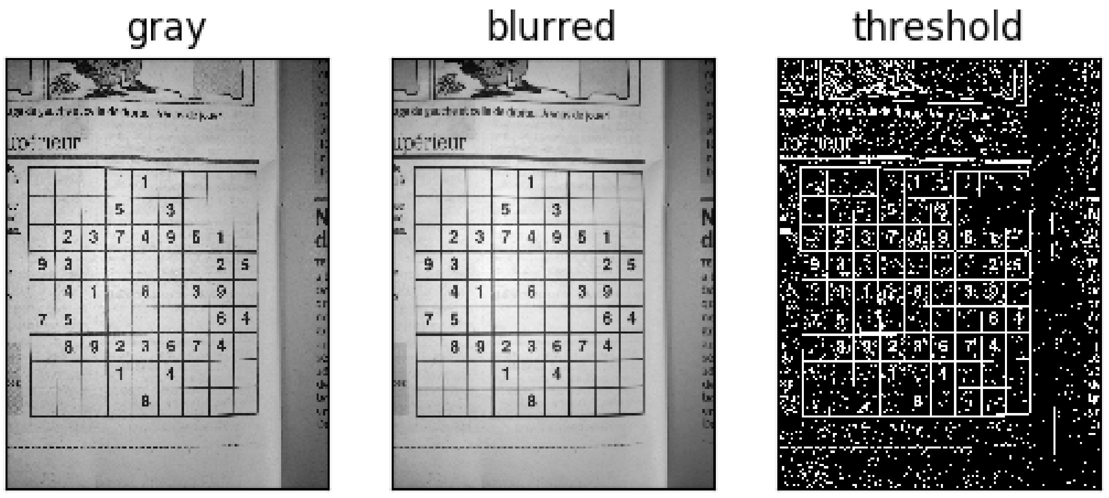

Figure 0.1: Phases of the image during pre-processing

Detecting The Biggest Blob

Now that I have my image ready for rectangle detection, I assumed that the biggest polygon in the image is the sudoku grid. I found all the contours in the image using cv2.findContours method. I sorted the contours with respect to their areas and got the biggest blob.

```python
# Assumption: The biggest contour in the image is the sudoku grid.
        contours = sorted(contours, key=lambda x: cv2.contourArea(x),
                            reverse=True)
        largest_contour = np.squeeze(contours[0])
```

Listing 1: Python example

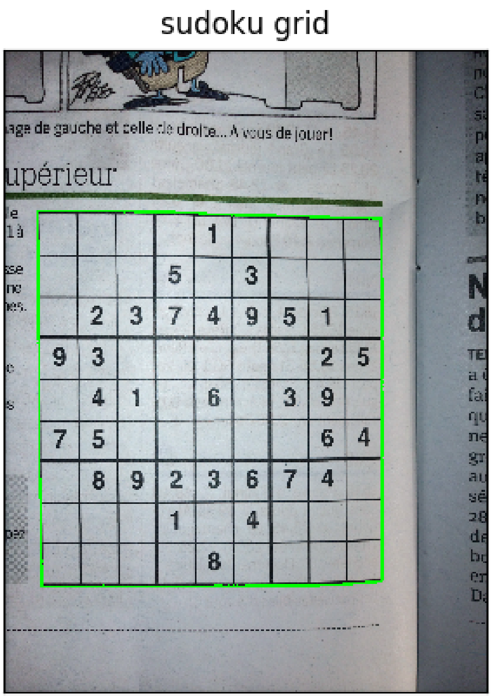

Figure 0.2: Biggest Blob of the image

Detecting The Sudoku Cells

With the detected borders of the Sudoku grid, I created a mask image to only focus on what is inside the Sudoku Grid. I created a new image and cropped it with the help of the mask, so that I have a region of interest. Just like the pre-processing phase, I applied Gaussian Blur and adaptive thresholding, this time to make the image ready for the detection of smaller rectangles. For images with a higher quality, I used a kernel size with 11 x 11, I found out that for images with bad quality smaller sized kernels performed better.

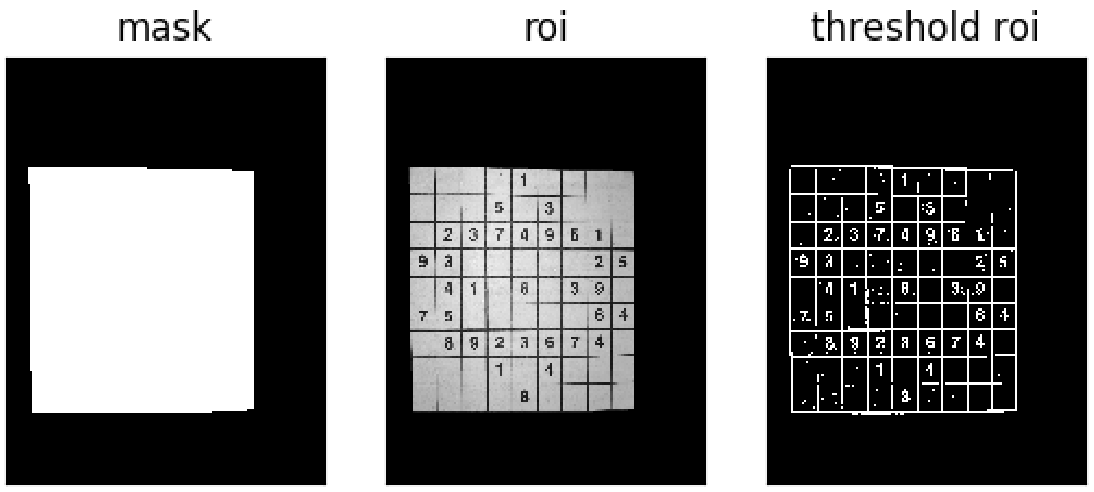

Figure 0.3: Phases of the image after applying the mask

Now, I have the image ready for the detection of the sudoku cells. Again, after some trial and error, I figured out that if the area of a contour is bigger than 800, it is a cell in the sudoku grid. If a smaller threshold is used, one may end up with not just the cells.

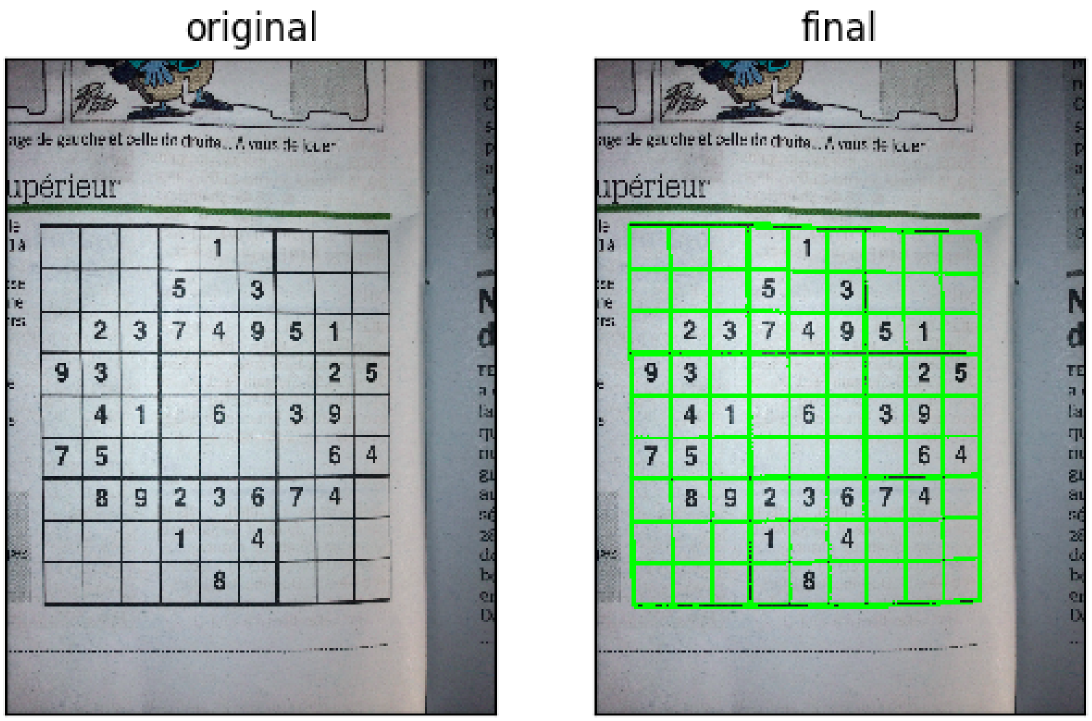

Figure 0.4: Processed image

Findings & Comments

There were 40 images in the v2_dataset. My algorithm successfully detected all the cells in the sudoku grid in 17 of the images, only the bounding rectangle with some missing cells in 8 images and detected only some of the cells in 16 images. I believe my results are not bad considering the quality of the images in the dataset. The algorithm can also I also tried using HoughLines for the detection of the lines but I was not satisfied with the overall results and ended up using processing the contours.

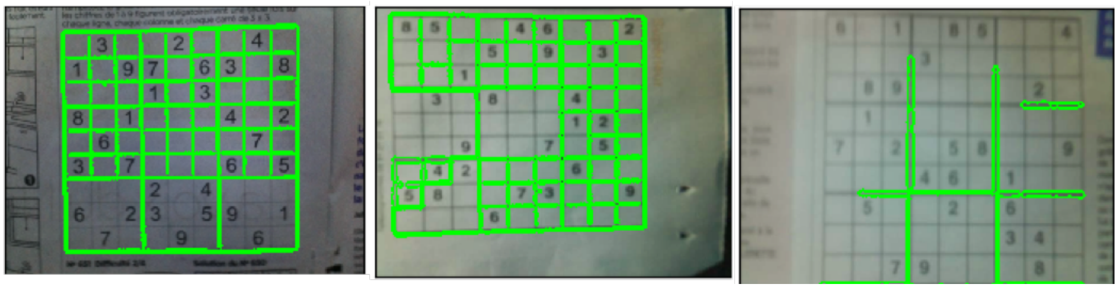

Figure 0.5: Unsuccessful Results