Report of Computer Vision with OpenCV

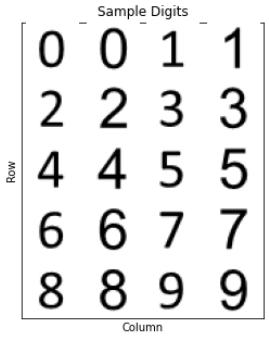
(Project 2, part 3)[1]

Author: ZHUO Xu (卓旭); Email: zhuoxu@seu.edu.cn

All the previous procedures in Part 1 and Part 2 are integrated into functions called *sudoku_part1, sudoku_part2*, which takes a Sudoku Puzzle Photo and gives the 81 extracted cells with empty/fill identifications. We'll start from the cells (not threshed) in Part 3.

7. Digits Recognition

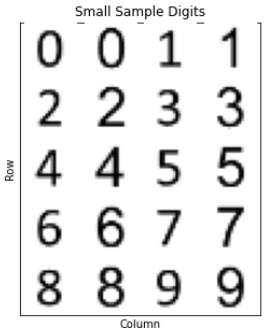
Sample digits of 1~9 with two fonts are read from provided dataset along with their labels.




```
array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9])
```

Figure 1 – Sample Digits and Corresponding Labels (Left-Right, Top-Down).

These digits are resized into 20×20 ones, Displayed as follow.




```
array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9])
```

Figure 2 – Smaller Sample Digits and Corresponding Labels.

Using HOG as feature descriptor, we can compute a feature vector for each digit image. The parameters follow those in the handout. After this, we get a series of HOG Feature Vectors stored in *referenceHOG*. Each vector has 144 dimensions. The dimension is so high that it is difficult to visualize it well, but we can imagine that feature vectors corresponding to a same digit should cluster in the hyperspace.

A function called *compute_NN* is implemented to get the digit with smallest HOG vector from the test input to those references. Euclidean Distance and Nearest Neighbor

[1] Corresponding code: *sudoku_part3.ipynb*; Opensource at github.com/z0gSh1u/rennes1-seu-cv .

are applied as metrics. The target is to calculate the objective function

$$\arg\min_i \sqrt{\|\text{testHOG} - \text{referenceHOG}_i\|_2^2}$$

It returns the index of the reference with minimal distance and the distance itself.

Since reference images are of shape 20×20, extracted cells are also resized into this shape. And reference images have pure white background, so cells are threshed by 255 // 2 = 127 to set the gray background to pure white, and foreground to pure black. The processed cells are shown below. Black cells have no digits inside.

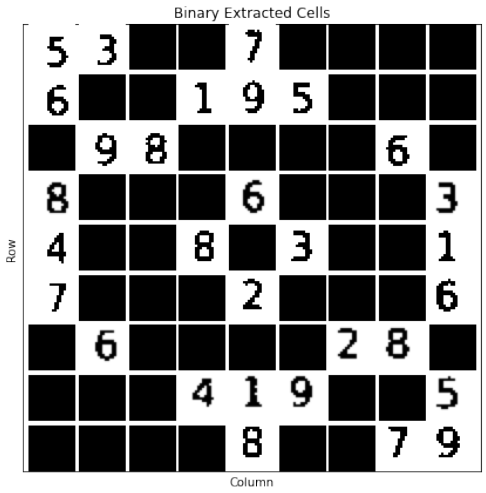


Figure 3 – Threshed and Resized Cells.

Then, a multiscale sliding window approach is implemented in function *slidingWindowDigitRecog* and applied on filled cells. The initial window size is 15×15. Each window is resized to 20×20 and then its HOG is compared with those references to identify what digit it is. Multiscale solutions are more robust to scaling of digits. The recognition result is 100% correct as shown below.

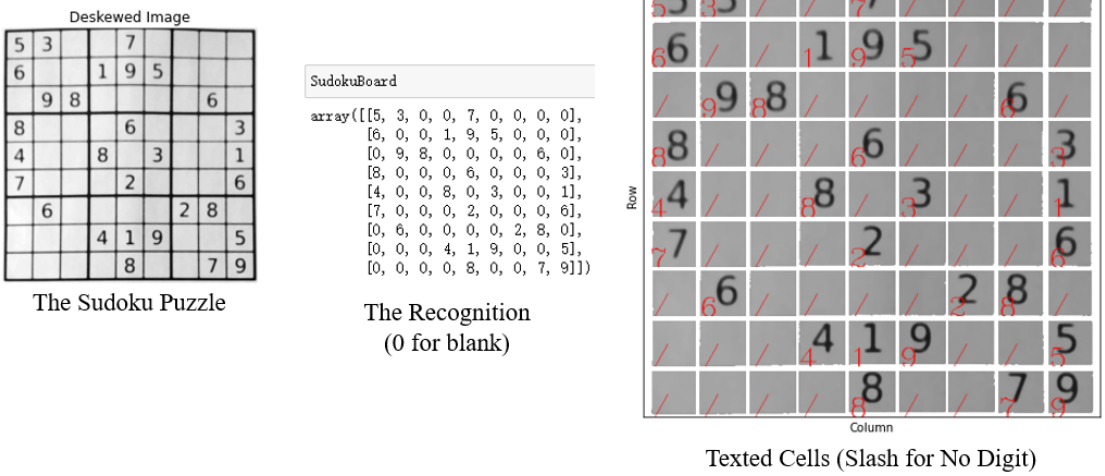


Figure 4 – The Digit Recognition Result.

8. Solve the Sudoku Puzzle

Now, we have the "digital" Sudoku Puzzle. I am not a Sudoku expert. So, I use the

py-sudoku library to solve it. The solution is shown below. It is verified to be correct.

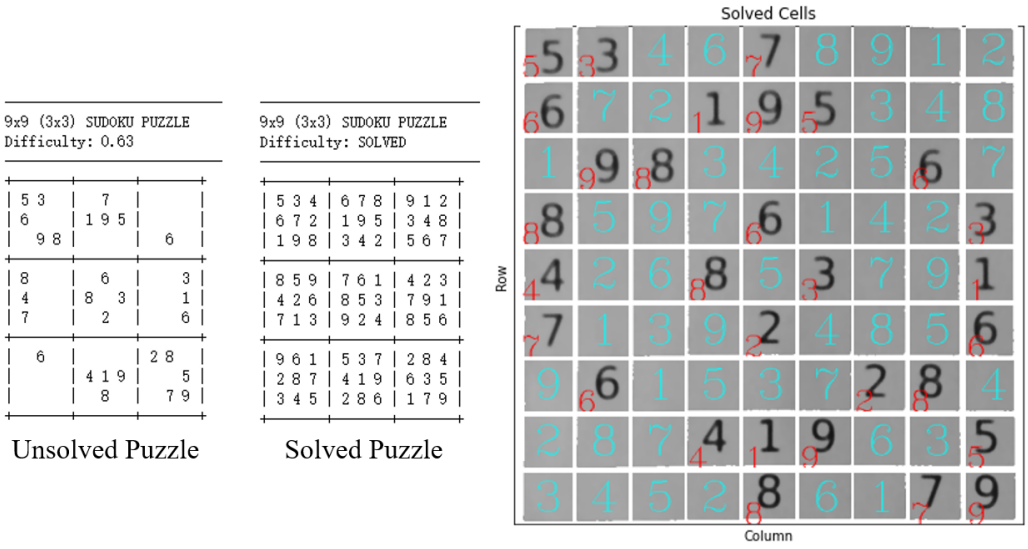


Figure 5 – The Solution.

9. Robustness Analysis[2]

To further evaluate the robustness (generalization ability) of our entire program, an ensembled function is written. Some procedures are revised and enhanced here:

① The original de-skewed image is 721×721. So, all the other de-skewed images are resized into this shape, then it matches our original case and reduce extra parameters.

② The center crop for 80×80 cells into 64×64 ones is unnecessary if we fully remove the grid lines. So, this step is cancelled.

③ The rule to judge empty/filled cells is the count of non-zero pixels in the Adaptively Threshed cell. In Part 2, the parameter C in *cv2.adaptiveThreshold* is set to 0. This will cause the output to have only the contour. We set C to 2 here, so that the result follows the conventional threshold concept.

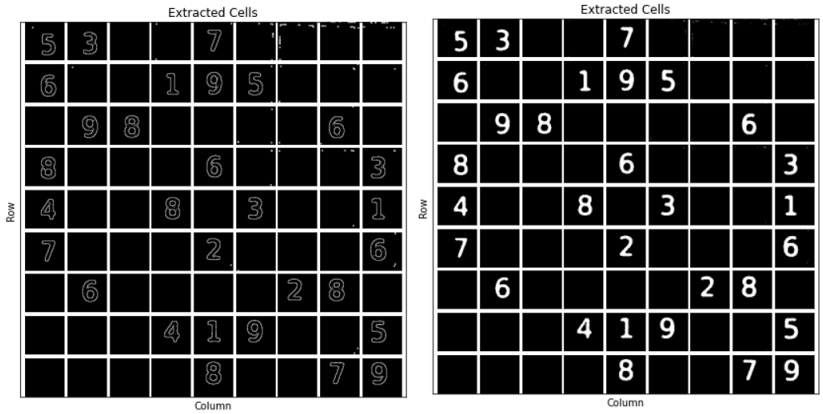


Figure 6 – Adaptive Threshed Cells with Different C.

Then, the rule to identify empty/filled cells is changed to "Cells with more than 2.5%

[2] Corresponding code: *sudoku_ensumble.ipynb*; Opensource at github.com/z0gSh1u/rennes1-seu-cv .

non-zero pixels are filled, otherwise empty". This is more robust than an absolute pixel count.

④ The number 0 in the sample dataset to compute HOGs is ignored, since there will not be 0 in the cells. This avoids some failure cases in number recognition.

After these modifications, the original case still works well as before, and no critical parameters need manually setting. For details, please refer to the source code.

I then searched for some other Sudoku photos on the Internet to test our algorithm. Results are shown below. Original sources of photos are provided.

Photo 1 *(data/extra/extra1.jpg) [https://golsteyn.com/writing/sudoku]*

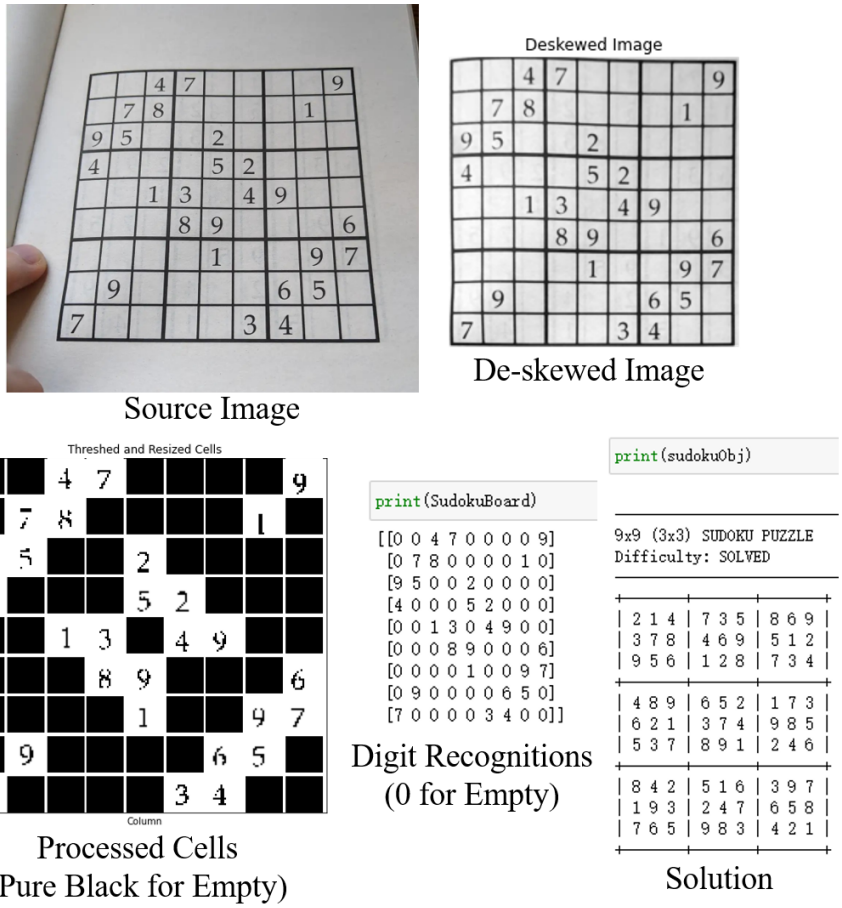


Figure 7 – Results of Extra Photo 1.

The source image is very clean, without complicated external interferences. Under this situation, our algorithm works well and gives all correct results. Notice that digits in processed binary cells are a little blurring and unconnected. We can add a dilation step on each filled cell to enhance the digit.

Photo 2 *(data/extra/extra2.jpg) [https://www.dreamstime.com/royalty-free-stock-images-sudoku-puzzle-image863979]*

(Continued on the next page.)

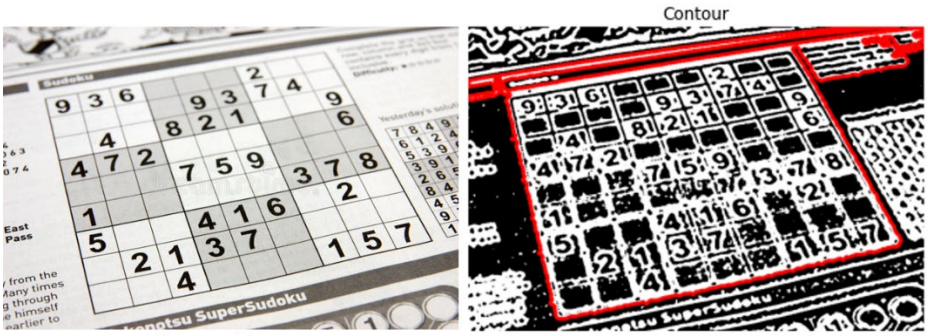


Figure 8 – Results of Extra Photo 2.

This photo is a failure case. External interferences like the title is very close to the puzzle's grid, making the contour finding incorrect.

Expect those points discussed above, the digit recognition procedure may also be improved by using popular Deep Learning-based approaches like LeNet-5 on MNIST.

10. Conclusions

In this project, we developed a computer vision-based Sudoku Solver. It generalizes to some other clean images, while is not very robust for complicated cases. It is challenging to develop a very robust and generic algorithm for engineering practices, since there are many corner cases.