GENERALCONFEDERATION OF LABOR OF VIETNAM

TON DUC THANG UNIVERSITY FACULTY OF INFORMATION TECHNOLOGY

DIGITAL IMAGE PROCESSING

REPORT ASSIGNMENT

Instructing Lecturer: **Dr. PHAM VAN HUY**

Student's name: ***NGUYEN HUNG GIA AN - 518H0126***

Class: **18H50205**

Course: **22**

HO CHI MINH CITY, 2022

ACKNOWLEDGEMENT

In order to make this report complete and achieve good results, we have received the support and assistance of many teachers and classmates.

With deep affection, sincerity, we express deep gratitude to all individuals and agencies who have helped us in our study and research.

First of all, I would like to express a special appreciation to Ton Duc Thang University's teachers for their conscientious guidance and advices throughout the last semester by gave me their modern outlook and meticulous supervision to carry out the job perfectly.

Especially we would like to send our sincere thanks to Mr.Pham Van Huy has paid attention, help, guide us to complete the report well over the past time.

We would like to express our sincere gratitude to the leadership of Ton Duc Thang University for supporting, helping and facilitating us to complete the report well during the study period.

With limited time and experience, this report can not avoid mistakes. We are looking forward to receiving advice and comments from teachers so that we can improve our awareness, better serve the practical work later.

Thank you!

EVALUATION OF INSTRUCTING LECTURER

Confirmation of the instructor

Ho Chi Minh, ... , 2022

(Sign and write your full name)

The assessment of the teacher marked

Ho Chi Minh, ... , 2022

(Sign and write your full name)

INTELLECTUAL PROPERTY

The following documentation, the content therein and/or the presentation of its information is proprietary to and embodies the confidential processes, designs, technologies and otherwise of team. All copyright, trademarks, trade names, patents, industrial designs, and other intellectual property rights contained herein are, unless otherwise specified, the exclusive property of team.

The ideas, concepts and/or their application, embodied within this documentation remain and constitute items of intellectual property which nevertheless belong to team.

The information (including, but by no means limited to, data, drawings, specification, documentation, software listings, source and/or object code) shall not be disclosed, manipulated, disseminated or otherwise in any manner inconsistent with the nature and/or conditions under which this documentation has been issued.

The information contained herein is believed to be accurate and reliable. Team accepts no responsibility for its use in any way whatsoever. Team shall not be liable for any expenses, damages and/or related costs which may result from the use of the information contained herein.

I. Implement

Step 1: Preprocessing

```
6
7   def pre_process_image(img, skip_dilate=False):
8       """Uses a blurring function, adaptive thresholding and dilation to expose the main features of an
9       proc = cv2.GaussianBlur(img.copy(), (9, 9), 0)
0       proc = cv2.adaptiveThreshold(proc, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
1       proc = cv2.bitwise_not(proc, proc)
2       if not skip_dilate:
3           kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]],np.uint8)
4           proc = cv2.dilate(proc, kernel)
5       return proc
6
```

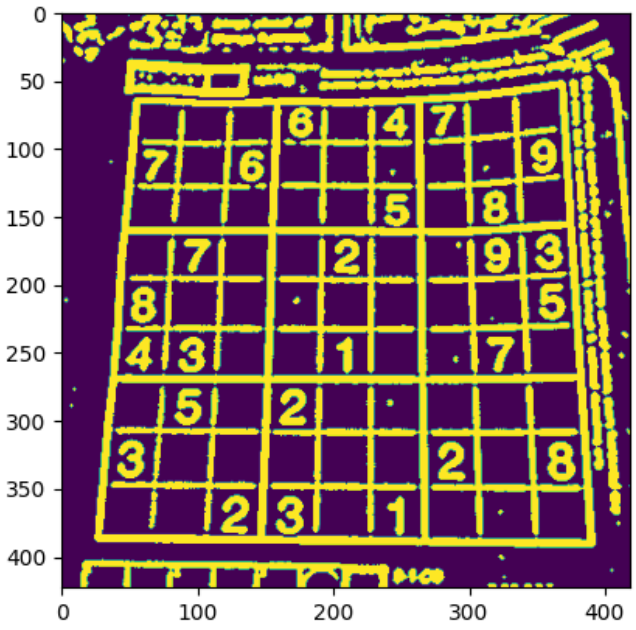
Blur the image a little. This smooths out the noise a bit and makes extracting the grid lines easier.

With the noise smoothed out, we can now threshold the image. The image can have varying illumination levels, so a good choice for a thresholding algorithm would be an adaptive threshold. It calculates a threshold level several small windows in the image. This threshold level is calculated using the mean level in the window. So it keeps things illumination independent.

It calculates a mean over a 5x5 window and subtracts 2 from the mean. This is the threshold level for every pixel.

Since we're interested in the borders, and they are black, we invert the image *outerBox*. Then, the borders of the puzzles are white (along with other noise).

This thresholding operation can disconnect certain connected parts (like lines). So dilating the image once will fill up any small "cracks" that might have crept in.



Step 2: Find the biggest connected pixel structure in the image.

```python
def find_largest_feature(inp_img, scan_tl=None, scan_br=None):
    """
    Uses the fact the `floodFill` function returns a bounding box of the area it filled to find the biggest
    connected pixel structure in the image. Fills this structure in white, reducing the rest to black.
    """
    img = inp_img.copy()
    height, width = img.shape[:2]
    max_area = 0
    seed_point = (None, None)
    if scan_tl is None:
        scan_tl = [0, 0]
    if scan_br is None:
        scan_br = [width, height]
    for x in range(scan_tl[0], scan_br[0]):
        for y in range(scan_tl[1], scan_br[1]):
            if img.item(y, x) == 255 and x < width and y < height:
                area = cv2.floodFill(img, None, (x, y), 64)
                if area[0] > max_area:
                    max_area = area[0]
                    seed_point = (x, y)
    for x in range(width):
        for y in range(height):
            if img.item(y, x) == 255 and x < width and y < height:
                cv2.floodFill(img, None, (x, y), 64)

    mask = np.zeros((height + 2, width + 2), np.uint8)
    if all([p is not None for p in seed_point]):
        cv2.floodFill(img, mask, seed_point, 255)
    top, bottom, left, right = height, 0, width, 0
    for x in range(width):
        for y in range(height):
            if img.item(y, x) == 64:
                cv2.floodFill(img, mask, (x, y), 0)
            if img.item(y, x) == 255:
                top = y if y < top else top
                bottom = y if y > bottom else bottom
                left = x if x < left else left
                right = x if x > right else right
    bbox = [[left, top], [right, bottom]]
    return img, np.array(bbox, dtype='float32'), seed_point
```

Here's the technique I use. First, I use the floodfill command. This command returns a bounding rectangle of the pixels it filled. We've assumed the biggest thing in the picture to be the puzzle. So the biggest blob should have be the puzzle. Since it is the biggest, it will have the biggest bounding box as well. So we find the biggest bounding box, and save the location where we did the flood fill.

Step 3: Crop and warps a rectangular

```
def distance_between(p1, p2):
    """Returns the scalar distance between two points"""
    a = p2[0] - p1[0]
    b = p2[1] - p1[1]
    return np.sqrt((a ** 2) + (b ** 2))
```

```
58
59  def crop_and_warp(img, crop_rect):
60      """Crops and warps a rectangular section from an image into a square of similar size."""
61      top_left, top_right, bottom_right, bottom_left = crop_rect[0], crop_rect[1], crop_rect[2], crop_rect[3]
62      src = np.array([top_left, top_right, bottom_right, bottom_left], dtype='float32')
63      side = max([
64          distance_between(bottom_right, top_right),
65          distance_between(top_left, bottom_left),
66          distance_between(bottom_right, bottom_left),
67          distance_between(top_left, top_right)
68      ])
69      dst = np.array([[0, 0], [side - 1, 0], [side - 1, side - 1], [0, side - 1]], dtype='float32')
70      m = cv2.getPerspectiveTransform(src, dst)
71      return cv2.warpPerspective(img, m, (int(side), int(side)))
72
```

Simple code. We calculate the length of each edge. Whenever we find a longer edge, we store its length squared. And finally when we have the longest edge, we do a square root to get its exact length.

Next, we create source and destination points. The top left point in the source is equivalent to the point (0,0) in the corrected image. And so on.

Then we create a new image and do the undistortion

Step 4: Get digits

The idea is extract the largest component in the square. First step I extract each cells from the image by slicing the Sudoku grid

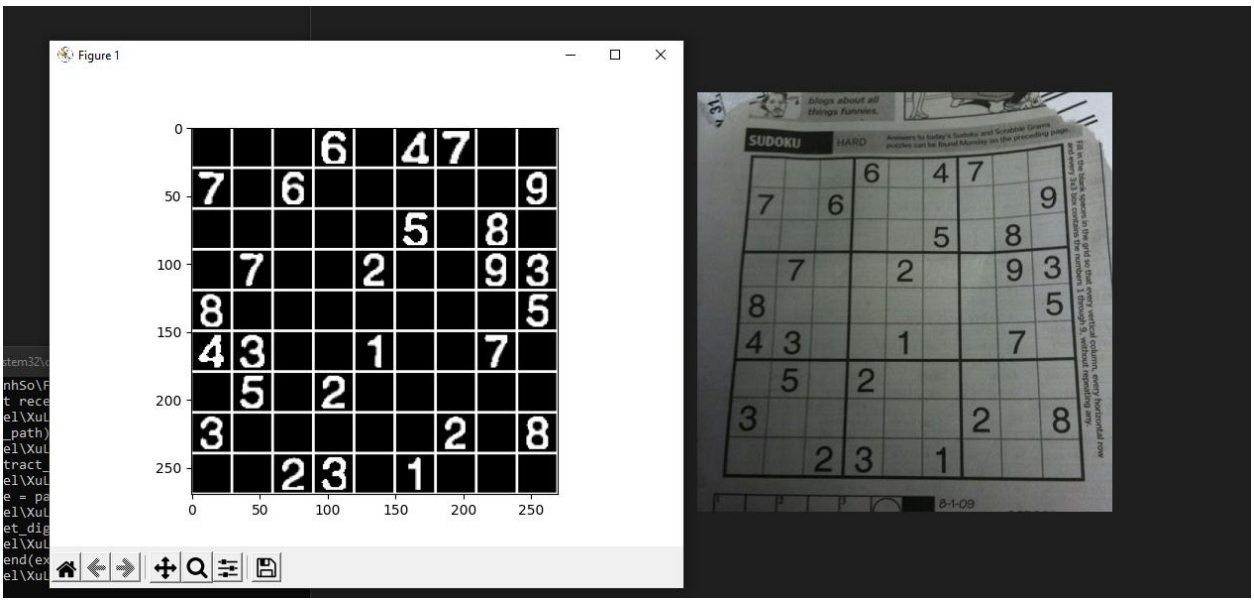
```
def infer_grid(img):
    """Infers 81 cell grid from a square image."""
    squares = []
    side = img.shape[:1]
    side = side[0] / 9
    for j in range(9):
        for i in range(9):
            p1 = (i * side, j * side)
            p2 = ((i + 1) * side, (j + 1) * side)
            squares.append((p1, p2))
    return squares
```

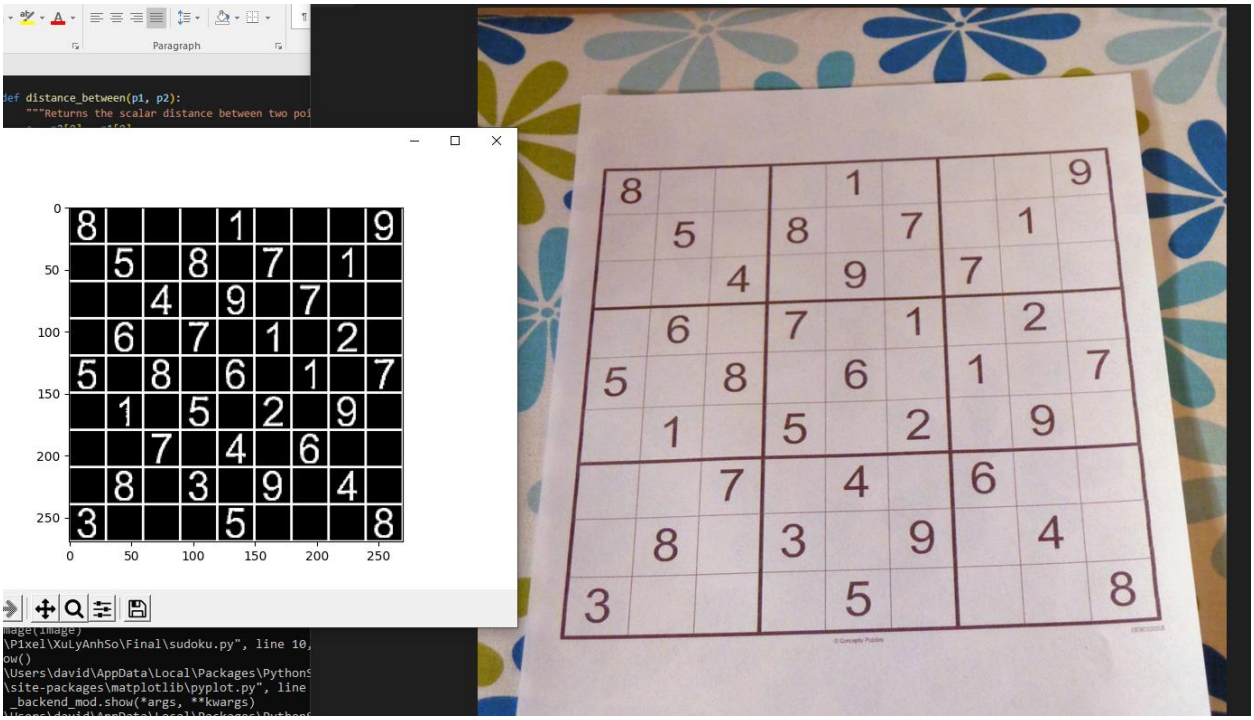
Next, I just need to extract the biggest component in each square and that is the number we need to get

```python
def extract_digit(img, rect, size):
    """Extracts a digit (if one exists) from a Sudoku square."""
    digit = cut_from_rect(img, rect)
    h, w = digit.shape[:2]
    margin = int(np.mean([h, w]) / 2.5)
    _, bbox, seed = find_largest_feature(digit, [margin, margin], [w - margin, h - margin])
    digit = cut_from_rect(digit, bbox)
    w = bbox[1][0] - bbox[0][0]
    h = bbox[1][1] - bbox[0][1]
    if w > 0 and h > 0 and (w * h) > 100 and len(digit) > 0:
        return scale_and_centre(digit, size, 4)
    else:
        return np.zeros((size, size), np.uint8)

def get_digits(img, squares, size):
    """Extracts digits from their cells and builds an array"""
    digits = []
    img = pre_process_image(img.copy(), skip_dilate=True)
#     cv2.imshow('img', img)
    for square in squares:
        digits.append(extract_digit(img, square, size))
    return digits
```

II. Demo




Paragraph
def distance_between(p1, p2):
"""Returns the scalar distance between two poi
0 50 100 150 200 250
\P1xel\XuLyAnhSo\Final\sudoku.py", line 10,
ow()
\Users\david\AppData\Local\Packages\Python
\site-packages\matplotlib\pyplot.py", line
_backend_mod.show(*args, **kwargs)