Tema 1 - CAVA

Extragerea informației vizuale din careuri Sudoku

1. Introducere

Scopul acestui proiect este dezvoltarea unui program care extrage informatii vizuale din imagini in care gasim diverse careuri de puzzle-uri Sudoku.

2. Preprocesarea imaginilor

Pentru extragerea cat mai exacta a informatiilor, am trecut imaginea prim mai multe niveluri de preprocesare:

2.1. Normalizare

Tinand cont de felul in care se reflecta lumina, aceasta ar putea afecta felul in care interpretam imaginea, motiv pentru care am aplicat niste filtre pentru a o putea normaliza, eliminand umbrele sau razele mai puternice (cod preluat din laborator).

```
def normalize_image(img):
    noise = cv.dilate(img, np.ones((7, 7), np.uint8))
    blur = cv.medianBlur(noise, 21)
    res = 255 - cv.absdiff(img, blur)
    no_shadow = cv.normalize(res, None, alpha=0, beta=255, norm_type=cv.NORM_MINMAX)
    return no_shadow
```

2.2. Treshold

Dorim sa transformam forma imaginii astfel incat sa contina un singur canal (grayscale), normalizam (pasul 2.1.), aplicam altre filtre, dintre care cel mai relevant este adaptiveThreshold. (transforma imaginea astfel incat sa se adapteze la doar 2 culori: alb - negru sau 0-255).

```
def filter_image_v2(image):
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    img_norm = normalize_image(image)
    img_gblur = cv.GaussianBlur(img_norm, (9, 9), 0)

    thresh = cv.adaptiveThreshold(img_gblur, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY_INV, 11, 2)
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (2, 2))
    morph = cv.morphologyEx(thresh, cv.MORPH_OPEN, kernel)

    res = cv.dilate(morph, kernel, iterations=1)
    res = cv.erode(res, kernel)

    return res
```

2.3. Gasirea contururilor

Folosind un algoritm prezentat la laborator, am aplicat functia pe imagine pentru a gasi cele 4 colturi ale careului de Sudoku: stanga-sus, dreapta-sus, dreapta-jos, stanga-jos. De retinut este ca fara pasii anteriori, algoritmul de gasire a conturului nu este la fel de exact.

2.4. Devierea/translatarea imaginii

Odata ce au fost gaiste cele 4 margini ale careului, imaginea trebuie translatata, din cauza perspectivei din care a fost facuta fotografia.

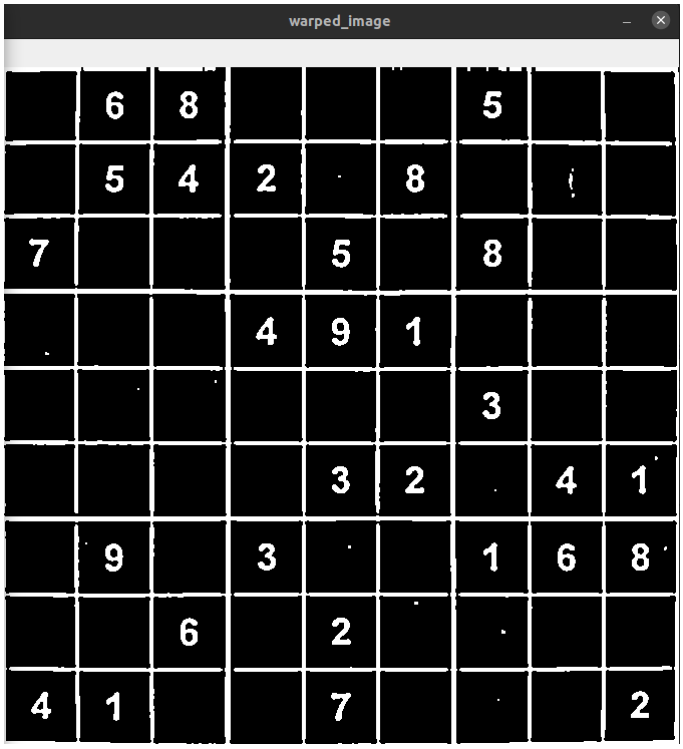
```
def warp_image(corners, image):
    corners = np.array(corners, dtype='float32')
    top_left, top_right, bottom_right, bottom_left = corners

    width = int(max([
        np.linalg.norm(top_right - bottom_right),
```

```
        np.linalg.norm(top_left - bottom_left),
        np.linalg.norm(bottom_right - bottom_left),
        np.linalg.norm(top_left - top_right)
    ]))

    mapping = np.array([[0, 0], [width - 1, 0], [width - 1, width - 1], [0, width - 1]],
dtype='float32')
    matrix = cv.getPerspectiveTransform(corners, mapping)

    return cv.warpPerspective(image, matrix, (width, width))
```

2.5. Rezultatul dupa aplicarea tuturor filtrelor

```
def preprocess_task1(image):
    filtered_img = filter_image_v2(image)
    corners = find_contours(filtered_img)

    warped, _ = warp_image(corners, image)
    warped_processed = filter_image_v2(warped)
    return warped_processed
```

Imaginea rezultata:



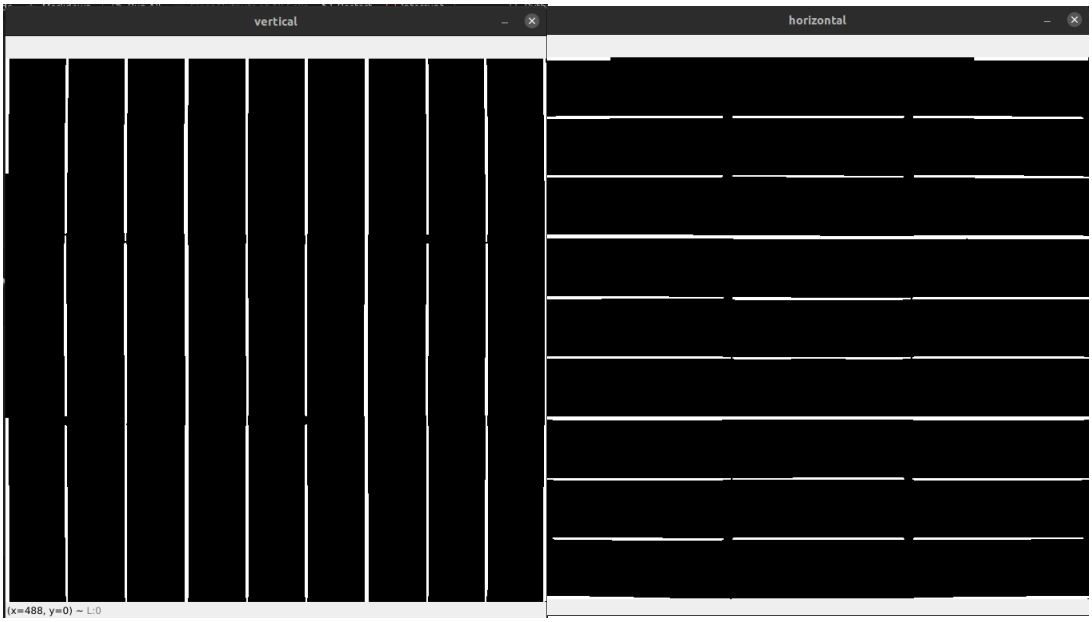
3. Eliminarea liniilor din careu

3.1. Extragerea liniilor verticale si orizontale

Folosing kernelul corespunzator, erodari si dilatari, am extras liniile orizontale si verticale din imagine:

```
def get_grid_line(image, type):
    img_cpy = image.copy()
    if type == 'h':
        s = img_cpy.shape[1]
    else:
        s = img_cpy.shape[0]
    size = s // 10

    if type == 'h':
        kernel = cv.getStructuringElement(cv.MORPH_RECT, (size, 1))
    else:
        kernel = cv.getStructuringElement(cv.MORPH_RECT, (1, size))

```

```
    img_cpy = cv.erode(img_cpy, kernel)
    img_cpy = cv.dilate(img_cpy, kernel)

    return img_cpy
```

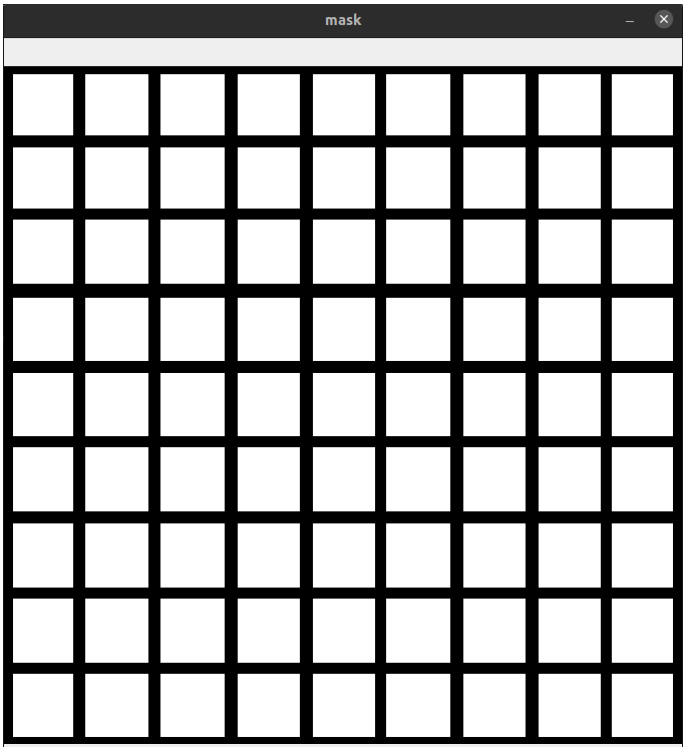


3.2. Transform liniile extrase intr-o masca

Se aduna cele doua imagini obtinute, aplicand ulterior un filtru de dilatare. Folosind transformata Hough pe rezultatul obtinut, voi putea desena linii mai proeminente pe deasupra. De asemenea, voi calcula negarea imaginii, pentru ca liniile sa devina negre.

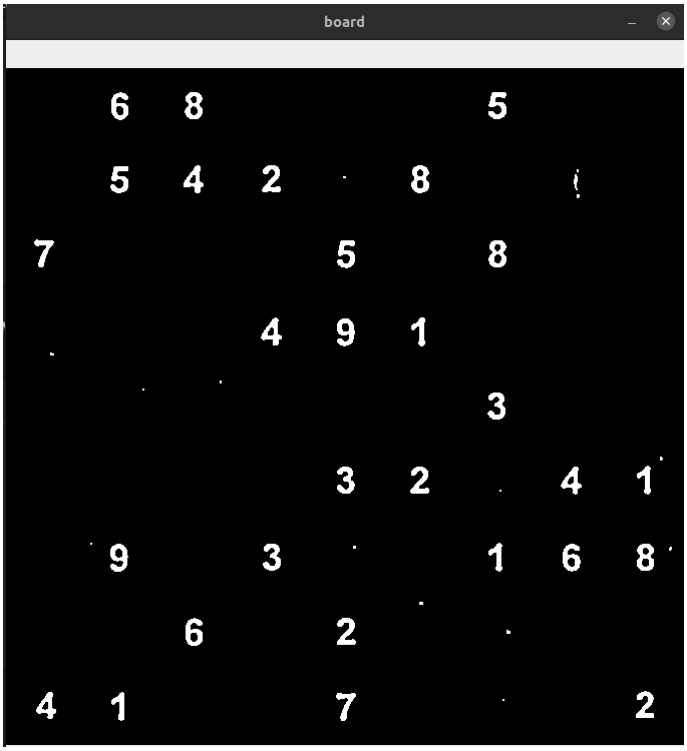
```
def grid_mask(vertical, horizontal):
    grid = cv.add(horizontal, vertical)
    grid = cv.adaptiveThreshold(grid, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 235, 2)
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
    grid = cv.dilate(grid, kernel, iterations=2)

    points = cv.HoughLines(grid, 0.3, np.pi/90, 200)
    lines = draw_grid_lines(grid, points)
    mask = cv.bitwise_not(lines)
    return mask
```

Masca rezultata:



3.3. Rezultatul dupa aplicarea mastii

Avand imaginea rezultata de la 2.3 si masca de la 3.2, se va calcula un bitwise and intre cele 2. Se va obtine urmatoarea imagine:



4. Rezolvare Task 1

Pentru submiterea solutiei am extras fiecare celula din careu, calculand matematic 9 linii si 9 coloane de lungime identica. Pentru fiecare celula extrasa am calculat o medie, iar beneficiind de preprocesarea facuta anterior, observam ca media celulelor ar trebui sa fie foarte mica, aproape de 0.

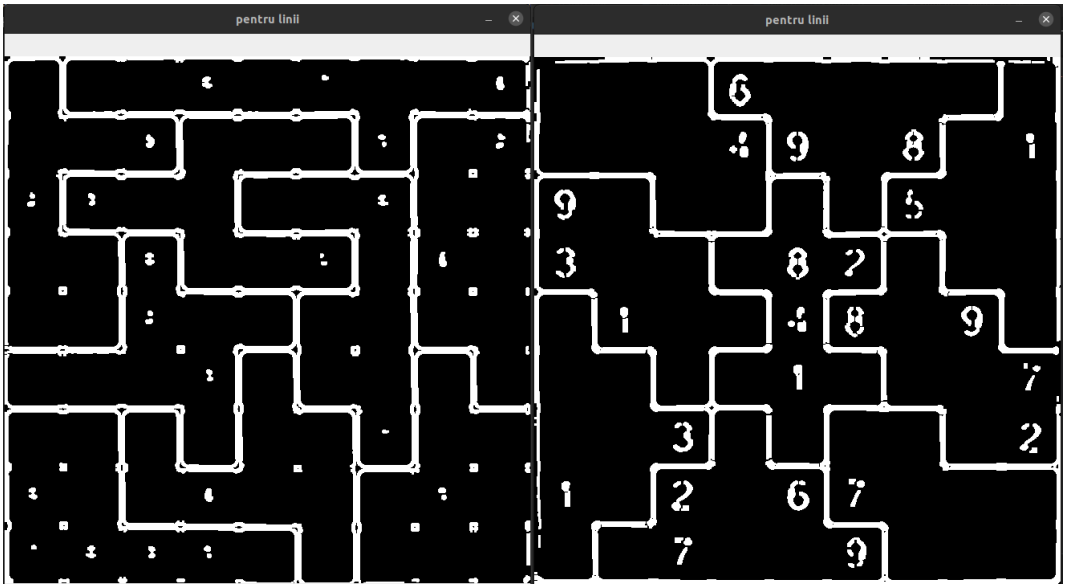
```
def cell_is_empty(cell):
    cell = np.array(cell)
    avg_color_row = np.average(cell, axis=0)
    avg_color = np.average(avg_color_row, axis=0)
    return avg_color <= 6
```

Pentru submisie am notat celula goala cu 0, iar celelalte cu x.

5. Rezolvare Task 2 - Sudoku Jigsaw

Pentru rezolvarea task-ului 2 am folosit aproximativ aceleasi filtre de preprocesare, doar ca am adaptat kernelul astfel incat sa pot evidentia liniile groase (pentru verificarea).

5.1. Extragerea liniilor groase



5.2. Extragerea celulelor - exact ca la Task-ul 1.

5.3. Impartirea pe culori

Impartirea pe culori se poate rezuma la o problema de gasire a componentelor conexe intr-un graf. Raportandu-ne la problema noastra, doua celule adiacente sunt conectate daca intre ele nu exista un "zid". Pentru recunoasterea zidului, am realizat din nou o medie a valorilor de pe linia careului.

Codul pentru gasirea componentelor conexe, pentru care asignez culori in ordine crescatoare.

```
def dfs(self, i, j, color):
    self.visited[i][j] = color
    for [[a_i, a_j], [b_i, b_j]] in self.adj_list:
        if a_i == i and a_j == j and self.visited[b_i][b_j] == -1:
            self.dfs(b_i, b_j, color)

def solve_islands(self):
    color = 1
    for i in range(9):
        for j in range(9):
            if self.visited[i][j] == -1:
                self.dfs(i, j, color)
                color += 1
```

6. Bonus

Pentru rezolvarea bonusului am antrenat pe layerele modelului ResNet50, fara preantrenare.