Concepte și Aplicații în Vederea Artificială
Tema 1 – Extragerea informației vizuale din careuri Sudoku

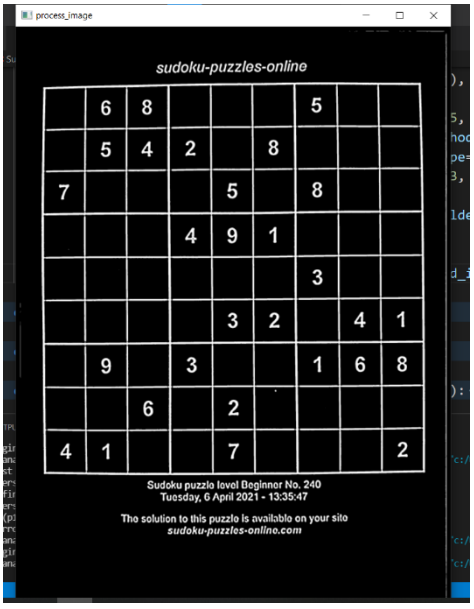
Task 1 – Extragerea configurațiilor din careuri Sudoku Clasic
Fișier: task1.py

opencv-python==4.5.4.60
numpy==1.21.4
scipy==1.7.2
imutils==0.5.4

Pentru o mai bună organizare, funcțiile folosite pentru procesarea unei singure imagini și extragerea sudok-ului sub formă de matrice și scrierea acesteia în fișierul soluție, se află în clasa *SudokuClassic*. Această este instanțiată pentru fiecare imagine de antrenare/testare și apelate metodele acesteia în funcția **apply_for_all(output_folder_path, input_folder_path, name_pattern)**.

În constructorul clasei este salvată imaginea care urmează să fie prelucrată și sunt inițializate datele membre care vor fi folosite. De asemenea, există și o metodă statică **show_an_image(window_name, image)** care a fost folosită pentru debugging și afișează o imagine dată că parametru prin redimensionarea ei la 20% din dimensiunea originală.

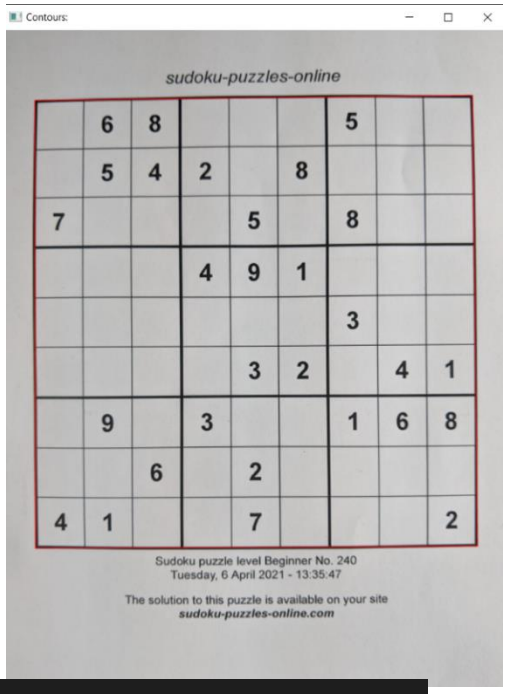
Pașii pentru procesarea imaginilor sunt împărțiți în metode, apelate în ordinea următoare:



1. **process_image(self)**
   Adaugă o bordură de 40px de culoarea marginilor pentru detectarea conturului tablei cu succes și în cazul în care tabla nu este încadrată central, iar una dintre margini lipsește. Convertește imaginea la tonuri de gri(inputul dat fiind de tip RGB), scapă de noise pentru o acuratețe mai mare în recunoașterea contururilor prin intermediul blurului, îndepărtează gradientul din imagine folosind *cv.adaptiveThreshold(...)* și inversează albul cu negrul, deoarece recunoașterea contururilor se face pe culoarea albă.

2. **get_contours(self)**
   Detectează toate contururile și le salvează sub forma unui *np.array* în care fiecare element reprezintă o listă de coordonate x și y, unde x este punctul de start al conturului și y punctul de final a conturului. Acest array va fi sortat în ordinea descrescătoare a punctelor de start.

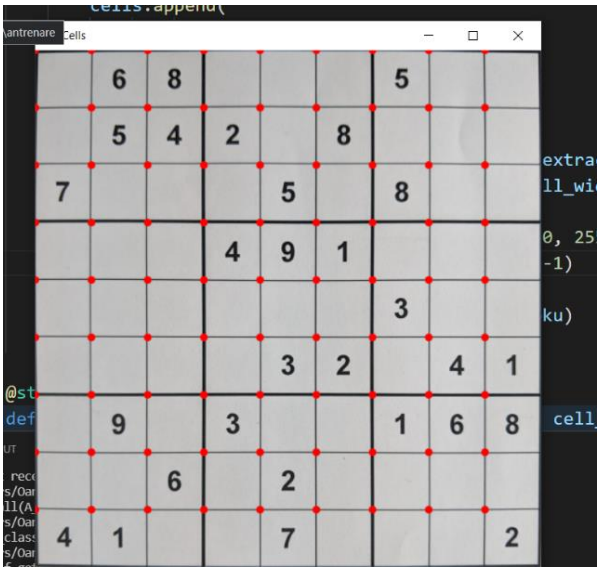
3. **iterate_through_contours(self)**
   Funcția de la punctul anterior poate detecta toate contururile existente în imagine(inclusiv numerele). Pentru a extrage doar conturul mare, pătratic al tablei, calculez pentru fiecare contur detectat poligonul aproximativ determinat. Ca acesta să fie eligibil, trebuie să aibă 4 vârfuri și o arie mai mare ca 500 000 px. Un astfel de rezultat se poate vedea marcat cu roșu pe imaginea din figura alăturată.



4. **extract(self)**
   Calculează coordonatele colțurilor și dimensiunea pătratului cu tabla de sudoku, folosind suma și diferența maximă/minimă a colțurilor unei imagini și distanța Euclidiană între două puncte. În plus, translatează colțurile pentru cazul în care tabla a fost fotografiată strâmb:

```
# transforms the possibly rotated image by translating the corners
perspective_transform = cv.getPerspectiveTransform(src=corners,
                                                   dst=sudoku_matrix_template)
# self.image is RGB => extracted_sudoku is RGB
extracted_sudoku = cv.warpPerspective(src=self.image,
                                      M=perspective_transform,
                                      dsize=(sudoku_width, sudoku_height))
```

   a. **get_cells_coordinates(self, cell_width, cell_height)**: calculează colțul din stânga sus pentru fiecare celulă și returnează un array (dimesniune 81) cu coordonatele (x,y) ale celulelor, începând de la stânga la dreapata și de sus în jos pe tablă.



b. **get_cells_with_numbers(self, cells, sudoku_contour_warped, cell_height, cell_width)**: metodă statică, în care extrag fiecare celulă și verific dacă există ceva scris în ea prin calcularea valorii medii și compararea acesteia cu un bias de valoare 10. Orice valoare mai mică ca acesta semnifică faptul că, nu se află nimic scris în celulă. De asemenea folosesc un padding de 40px prin care ignor eventualele margin preluate în extragerea celulei.

Extragere celulă

```
cell = extracted_sudoku[coordinate[1] + padding: coordinate[1] + cell_height - padding,
                        coordinate[0] + padding: coordinate[0] + cell_width - padding]
```

Calcul valoare medie

```
threshold = cv.threshold(src=cell_grayed,
                         thresh=145,
                         maxval=255,
                         type=cv.THRESH_BINARY_INV)

mean_value = threshold[1].mean()
```

c. **make_sudoku(self, cells_with_numbers)**: generează matricea sub formă de listă de liste cu elementele ‘x’ sau ‘o’.

La final, array-ul va fi salvat ca soluție sub forma unui fișier *.txt* folosind metoda: **save_to_txt(self, output_path)**.

Task 2 – Extragerea configurațiilor din careuri Sudoku Jigsaw
Fișier: task2.py

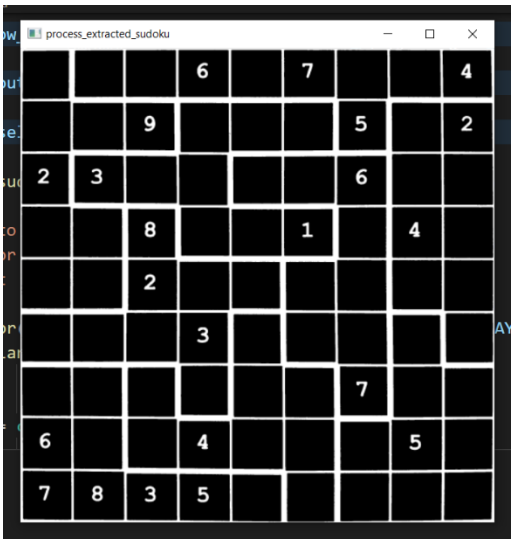
opencv-python==4.5.4.60
numpy==1.21.4
scipy==1.7.2
imutils==0.5.4

Asemenea task-ului 1, pentru o mai bună organizare, funcțiile folosite se află în clasa *SudokuJigsaw*. Pe lângă datele membre ale clasei de la task-ul 1, se mai adaugă un dicționar și 3 liste, iar funcționalitatea acestora va fi detaliată pe parcurs. Funcțiile pentru citire și scriere sunt aceleași ca la task-ul 1.

Primii pași pentru procesarea imaginilor sunt aceiași cu cei de la task-ul 1, iar cei noi urmează să fie detaliați. Algoritmul folosește setul de 10 culori diferite (declarate în dicționarul self.COLORS), pentru colorarea fiecărei zone, fiecărei culori fiindu-i atribuit un număr de la 0 la 9 corespunzător zonei în ordine. Pentru colorarea în totalitate a zonelor se elimină din imagine liniile subțiri care delimitează fiecare celulă în parte, rămânând doar cele groase care delimitează zonele sau tabla.

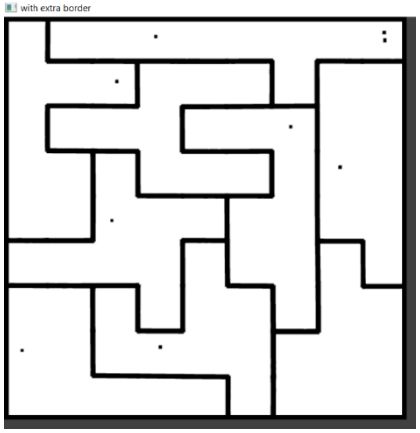
1. **process_raw_image(self)**: aceeași funcționalitate cu *process_image(self)* de la task-ul 1.
2. **get_contours_f1(self)**: aceeași funcționalitate cu *get_contours(self)* de la task-ul 1.
3. **iterate_through_contours_f1(self)**: aceeași funcționalitate cu *iterate_through_contours (self)* de la task-ul 1.

4. **extract(self)**:
   Păstrează de la task-ul 1, calculul coordonatelelor colțurilor, dimensiunea pătratului cu tabla de sudoku și translatarea colțurilor pentru cazul în care tabla a fost fotografiată strâmb. În plus, sunt apelate următoarele metode:
   a. **process_extracted_sudoku(self) :** imaginea cu tabla de sudoku extrasă este în format RGB și necesită prelucrare pentru colorarea individuală a zonelor: grayscale, blur și eliminarea gradientului.

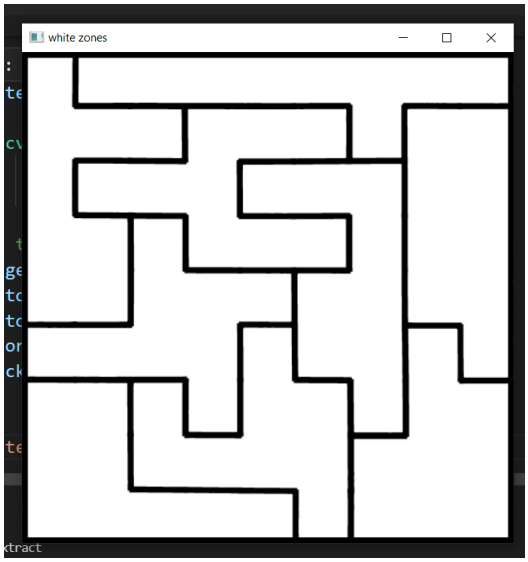


   b. **remove_thin_lines(self)**: elimină contururile subțiri prin *erode & dilate* (erodare și dilatare) folosind funcția din OpenCV *morphologyEx(...)* împreună cu constanta MORPH_OPEN.

   c. **draw_extra_border(self)**: desenează o bordură neagră de 30px pe marginea tablei de joc, pentru a evita eventualele erori create de neomogenitatea acesteia



d. **get_contours_f2(self)**: comportament asemănător metodei *get_contours_f1(self)*, dar fără criteriu de sortare a contururilor.
e. **iterate_through_contours_f2(self)**: comportament asemănător metodei iterate_through_contours_f1(self), numai că folosește un grad de detaliere mai mare (0.00002) pentru o colorare mai precisă a zonelor în alb.



f. **get_cells_coordinates(self, cell_width, cell_height)**: aceeași funcționalitate de la task-ul 1.

g. **remove_mistakes(self, cells_coordinates, contours, cell_width, cell_height)**: preia fiecare celulă determinată de punctul din stânga sus, calculează culoarea medie(îndepărtând o margine de 100px pentru evitarea erorilor) și colorează zona din care face parte (testată cu ajutorul functiei din OpenCV *cv.pointPolygonTest(...)*) cu, culorile din dicționar luate în ordinea dată. Imaginea finală va avea fiecare zonă colorată diferit.

h. **extract_zones_for_cells(self, cells, cell_width, cell_height)**: salvează într-un array (dimensiune 81) zona din care face parte fiecare celulă a tablei. Acest lucru se realizează asemănător raționamentelui metodei anterioare, prin calcularea culorii medii din fiecare celulă și compararea acesteia cu, culoarea corespunzătoare și o marjă de eroare de 5 unități.

```
# calculate the average color for the cell
# for the recognition of the zone
average_color = cv.mean(cell)[:3]

for zone_number in self.COLORS.keys():
    color = self.COLORS[zone_number]
    # calculate the errors
    e1 = abs(color[0] - average_color[0])
    e2 = abs(color[1] - average_color[1])
    e3 = abs(color[2] - average_color[2])

    # small errors < 5
    if e1 <= 5 and e2 <= 5 and e3 <= 5:
        self.zones_for_cells.append(zone_number)
        break
```

i. **get_cells_with_numbers(self, cells, self.extracted_sudoku, cell_height, cell_width)**: aceeași funcționalitate de la task-ul 1.
j. **make_sudoku(self, cells_with_numbers)**: aceeași funcționalitate de la task-ul 1.

Funcția **find_differences(path1, path2)** compară fișierele din două foldere diferite pentru a evalua performanța algoritmului.

Bibliografie:

pyimagesearch.com: OpenCV Sudoku solver and ocr

OpenCV documentation