

Concepțe și Aplicații în Vederea Artificială

Tema 1

Obiectiv

Scopul acestui proiect este dezvoltarea unui sistem automat pentru a calcula scorul în jocul Mathable.

Pentru realizarea proiectului, am folosit:

- *Python* 3.13.0
- *OpenCV* 4.10.0
- *NumPy* 2.1.2
- *Matplotlib* 3.9.2

Am folosit Jupiter Notebook pentru a scrie codul și pentru a testa funcționalitatea acestuia. Editorul de text folosit a fost Visual Studio Code.

Mențiuni

Soluția lucrează cu alte fișiere prin path-uri relative la folderul în care se află (Folder-ul **/solutie**). Soluția cuprinde o secțiune numită Evaluare, unde se introduce path-ul către setul de testare altături de numărul de jocuri și numărul de mutări ale jocului Mathable care se află în setul de testare.

```
# Folders and files we use
folder_path_input = '../testare'
folder_path_output = '../352_Buca_Mihnea-Vicentiu'

# number of games (add +1 since the loop starts from 1)
nr_games = 4 + 1
nr_turns = 50 + 1
```

Parametrii se pot modifica ca atare pentru a funcționa evaluarea pe setul de testare dorit. Pentru a rula soluția este necesar să se ruleze toate celulele din Jupiter Notebook.

Structura documentației

Documentația se împarte în 3 secțiuni:

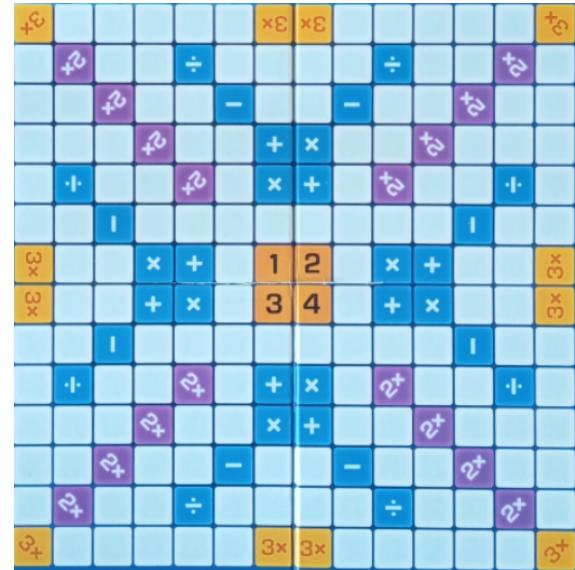
- **Task 1.** – unde se realizează detectarea tablei și extragerea pozițiilor unde se află piesele
- **Task 2.** – unde se realizează recunoașterea cifrelor de pe piesele plasate pe tabla de joc
- **Task 3.** – unde se realizează calculul scorului pentru fiecare joc

Task 1

Pentru a putea calcula scorul în jocul Mathable, trebuie să extragem pozițiile unde se află piesele. Este clar că nu avem nevoie de toată imaginea, ci doar de o parte din aceasta. Prin urmare scopul nostru este ca pentru o imagine dată, să furnizăm zona de interes.



Tabla jocului fară nici o piesă



Partea din imaginea care ne interesează

Detectia tablei de joc

Este suficient de intuitiv că pentru a detecta zona de interes, trebuie să identificăm colțurile tablei de joc.

Pentru a putea detecta colțurile tablei de joc, vom folosi o serie de transformări ale imaginii.

- Convertirea imaginii la tonuri de gri pentru a elimina informațiile de culoare.
- Aplicarea unui filtru median pentru a reduce zgomotul și pentru a păstra marginile clare.
- Aplicarea unui filtru Gaussian pentru o netezire suplimentară și eliminarea artefactelor.
- Erodarea imaginii pentru a elmina detaliile mici și a conecta segmentele apropiate.
- Detectarea marginilor folosind algoritmul Canny Edge Detection.
- Dilatarea marginilor detectate pentru a îmbunătăți conectivitatea și claritatea acestora.
- Găsirea contururilor în imagine folosind metodele de detectie a contururilor.

Ne vom folosi de o serie de funcții din OpenCV pentru a realiza aceste transformări.

```
def extract_table(image):  
    # find the coordinates of the table  
    image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY) # convert to gray  
    image_m.blur = cv.medianBlur(image_gray, 5) # median blur with 5x5 kernel  
    image_g.blur = cv.GaussianBlur(image_m.blur, (5, 5), 0) # gaussian blur with 5x5 kernel
```

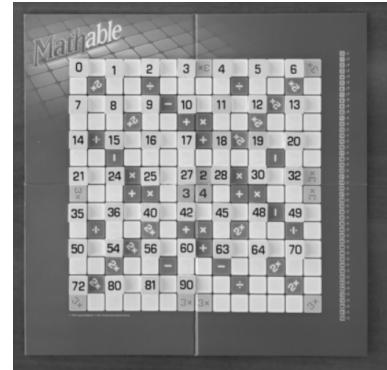
Prelucrarea imaginii arătând astfel:



Tablă + piesele de joc



Filteturui median



Filteturui gaussian

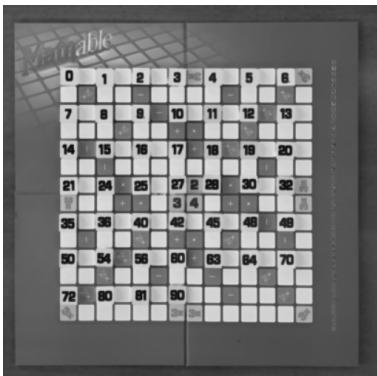
Continuăm cu erodarea imaginii și detectarea marginilor folosind algoritmul Canny Edge Detection.

```

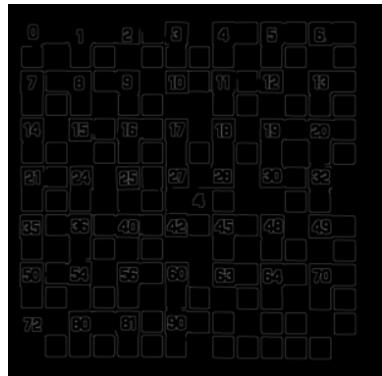
kernel = np.ones((5, 5), np.uint8) # kernel for erode and dilate
thresh = cv.erode(image_g_blur, kernel, iterations=2) # erode to remove small details
# 2 iterations to remove more details

edges = cv.Canny(thresh, 150, 400) # detect edges using Canny Edge Detection
edges = cv.dilate(edges, kernel, iterations=2) # dilate to improve connectivity

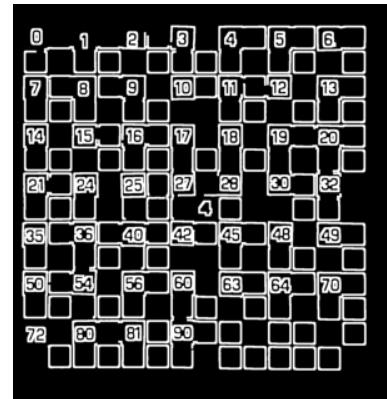
```



Imaginea erodată



Canny Edge Detection



Dilatarea marginilor

Având acest produs intermedian, putem să găsim contururile din imagine.

Vom folosi funcția `cv.findContours` pentru a găsi contururile din imagine, iar apoi vom extrage coordonatele acestora. Odată ce avem coordonatele, putem să extragem ‘maximal bounding box’ ce cuprinde toate contururile pentru a obține coordonatele colțurilor tablei de joc.

```

# find contours
contours, _ = cv.findContours(edges, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
for contour in contours: # find corners of the table
    x, y, w, h = cv.boundingRect(contour)
    top_left = (min(top_left[0], x), min(top_left[1], y))
    top_right = (max(top_right[0], x + w), min(top_right[1], y))
    bottom_left = (min(bottom_left[0], x), max(bottom_left[1], y + h))
    bottom_right = (max(bottom_right[0], x + w), max(bottom_right[1], y + h))

```

Odată extrase coordonatele colțurilor tablei de joc, putem vom schimba perspectiva imaginii pentru a extrage zona de interes.

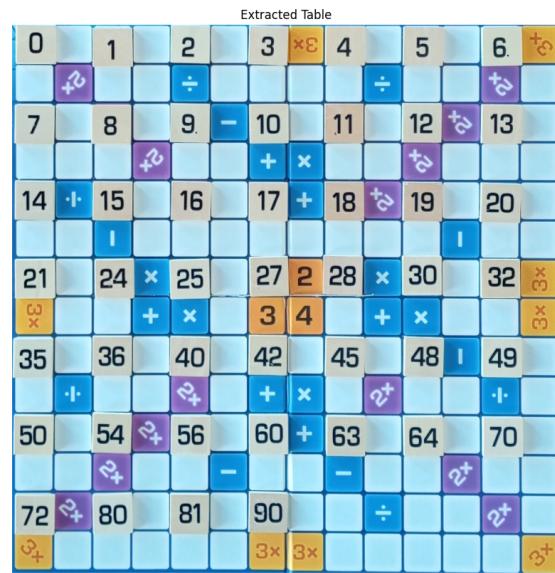
Vom folosi funcția `cv.getPerspectiveTransform` pentru a obține matricea de transformare și apoi funcția `cv.warpPerspective` pentru a obține imaginea finală.

```
# transform the perspective of the image
width, height = 910, 910
bounding_box = np.array([top_left, top_right, bottom_left, bottom_right], dtype="float32")
destination_points = np.array([[0, 0], [width, 0], [0, height], [width, height]], dtype="float32")
M = cv.getPerspectiveTransform(bounding_box, destination_points)
result = cv.warpPerspective(image, M, (width, height))
return result
```

Obținem astfel imaginea finală care reprezintă zona de interes.



Colțurile de interes ale tablei



Zona de interes

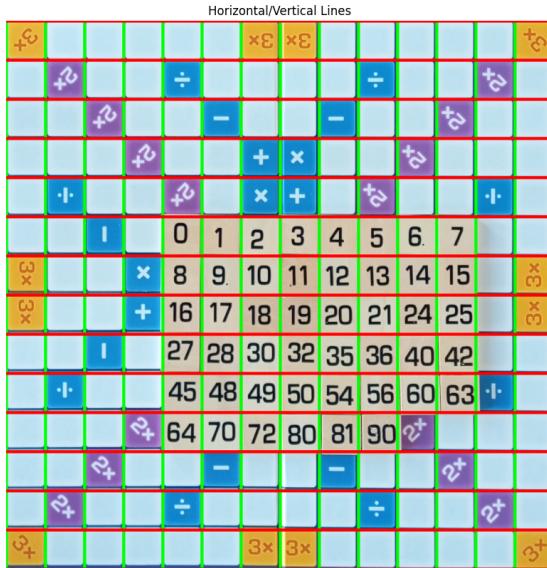
Odată ce am obținut zona de interes, putem trece la extragerea pozițiilor unde se află piesele de joc.

Extragerea pozițiilor pieselor de joc

Pe transformarea curentă a imaginii, vom delimita tabla de joc într-o grilă de 14 orizontale și verticale linii egal distanțate relativ la dimensiunea imaginii (910×910 anume fiecare careu are 65 de pixeli).

De asemenea pentru fiecare celulă din grilă, le delimităm după 5 categorii de culori: alb (**W**), galben (**Y**), albastru (**B**), violet (**V**) și spectrul ocupat dacă avem o piesă pusă pește.

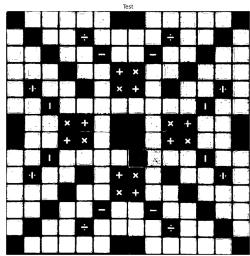
Vom folosi această grilă pentru a extrage pozițiile unde se află piesele de joc.



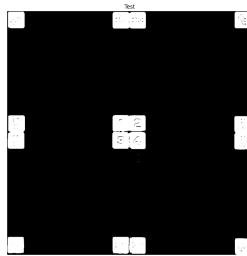
Grilă de 14×14

Pentru a vedea unde plasăm o nouă piesă de joc, ne vom folosi de mutarea curentă și cea precedentă (o imagine cu tabla goală dacă suntem la prima mutare), detectând zona unde se produce cel mai mare contrast. Mai întâi, vom supune zona de interes unor transformări pentru a putea detecta contrastul.

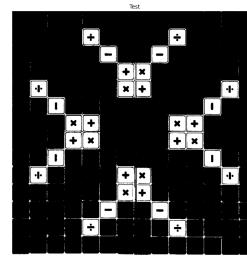
- Conversia imaginii în spațiul de culoare **HSV** pentru a facilita filtrarea bazată pe culoare.
- Se definește o gamă de culori specifice pentru fiecare categorie de interes:
 - Pentru alb (**W**): valori de culoare joase între [90, 0, 175] și [100, 125, 255].
 - Pentru galben (**Y**): valori de culoare joase între [0, 75, 0] și [90, 231, 255].
 - Pentru albastru (**B**): valori de culoare joase între [90, 210, 0] și [255, 255, 255].
 - Pentru violet (**V**): valori de culoare joase între [110, 0, 0] și [255, 255, 255].
- Se aplică aceste intervale pentru a filtra zonele de interes pe baza culorii, utilizând masca generată în spațiul HSV.



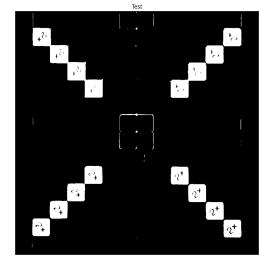
Filtru pt. Alb



Filtru pt. Galben



Filtru pt. Albastru



Filtru pt. Violet

Putem observa spre exemplu că piesa de joc este plasată peste o celulă de culoare albă pe poziția $(8, 7)$. Tot ce rămâne este să selectăm toate cele 14×14 celule să aplicăm filtrare corespunzătoare și folosind o funcție de detectare a contrastului maxim, dintre cele două imagini, să extragem poziția piesei de joc. Vom folosi o funcție de computare al contrastului creat, în cazul meu *MSE* (Mean Squared Error).

Posibile optimizări în detectare piesei

O posibilă optimizare ar fi verificare pozițiilor posibile unde putem plasa o piesă de joc, din moment ce, conform regulilor jocului, avem un spațiu limitat de plasare a pieselor.

Vom ține minte un set de poziții posibile unde putem plasa o piesă de joc și vom verifica doar aceste poziții, în momentul în care detectăm o piesă de joc, răspândim setul de poziții posibile (sus, jos, stânga, dreapta), pentru următoarea mutare.

Task 2

Pentru a putea calcula scorul în jocul Mathable, trebuie să recunoaștem cifrele de pe piesele de joc.

Din moment ce lucrăm cu imaginile unui board game, piesele de joc vor avea numere generice care își păstrează ‘forma’ și ‘culoarea’.

Prin urmare vom aplica un algoritm de template matching pentru a recunoaște cifrele de pe piesele de joc.

Extragere dataset

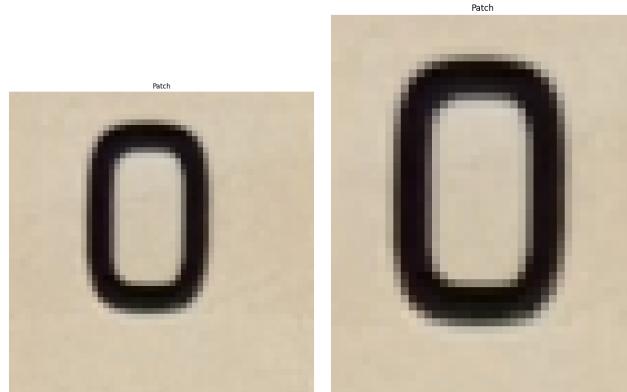
Pentru a putea rula algoritmul de template matching, vom avea nevoie de un dataset de imagini cu cifrele posibile jocului.

Ne vom folosi de setul dat ca antremanet unde la fiecare pozitie făcută vom recunoaște numărul aflat pe piesă.

Pentru a putea decupa numărul consistent de pe piesă, vom folosi o serie de transformări ale imaginii.

- Conversia imaginii în tonuri de gri;
- Aplicarea unui filtru median pentru a reduce zgomotul;
- Aplicarea unui filtru Gaussian pentru a netezi imaginea;
- Aplicarea unei operații de eroziune pentru a elibera zgomotul;
- Detectarea marginilor folosind algoritmul Canny
- Găsirea contururilor în imagine
- Găsirea celui mai mare contur și decuparea imaginii la acea regiune

Astfel după o astă transformare vom obține o imagine centralizată pe numărul de pe piesă.



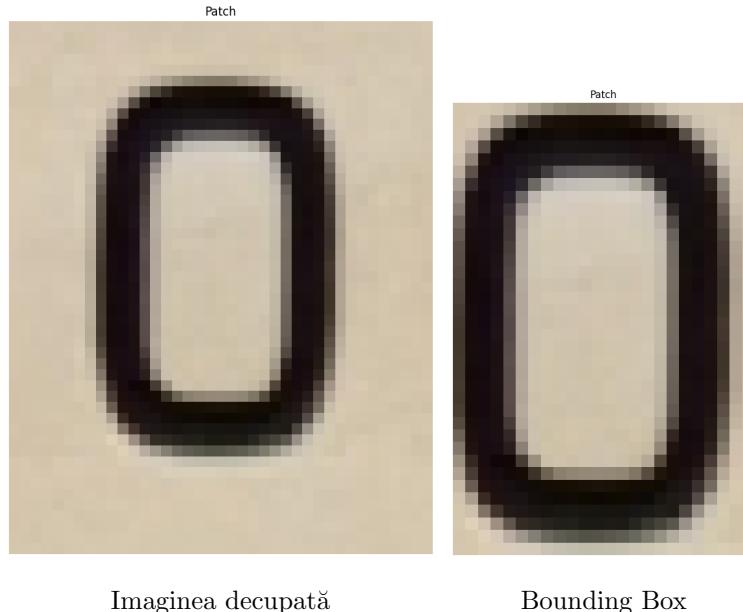
Imaginea originală

Imaginea decupată

Odată ce am obținut imaginea decupată, vom să restrângem din nou bounding boxul, pentru a avea cât mai puțin zgomot în imagine.

Din moment ce background-ul este alb, vom convertează imaginea la grayscale și vom aplica Canny Edge Detection pentru a detecta marginile numărului.

Vom folosi aceste margini pentru a găsi contururile și pentru a extrage bounding boxul care conține numărul, astfel restrângem bounding boxul la cea mai importantă regiune a imaginii.

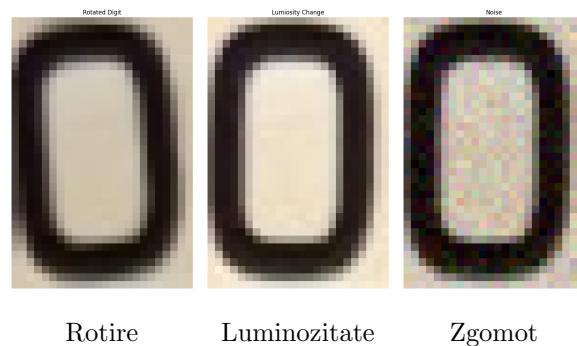


Aplicăm aceste transformări pentru fiecare imagine din datasetul nostru, astfel obținem un set de imagini cu numerele decupate.

Optimizare Dataset

Odată ce am obținut pentru fiecare număr în parte imaginiile decupate, putem să mai aplicăm o serie de transformări pentru a optimiza datasetul nostru.

- Rotirea imaginii cu un unghi mic θ pentru a obține mai multe variante ale aceluiași număr, din moment ce piesele nu sunt puse perfect pe tabla de joc.
- Adăugarea de zgomot în imagine
- Schimbare de luminozitate



```

# rotate the image
image_center = tuple(np.array(img.shape[1::-1]) / 2)
rot_mat = cv.getRotationMatrix2D(image_center, i, 1)
rotated = cv.warpAffine(img, rot_mat, img.shape[1::-1],
flags=cv.INTER_LINEAR, borderMode=cv.BORDER_REPLICATE)

# add noise to the image
img_aug = cv.add(img, np.random.normal(0, 10, img.shape), dtype=cv.CV_8UC3)

# change the brightness of the image
img_aug = cv.convertScaleAbs(img, alpha=alpha, beta=beta)

```

Astfel obținem un dataset variat care ne va ajuta să recunoaștem cifrele de pe piesele de joc.

Template Matching

Template Matching este o tehnică de recunoaștere a obiectelor în imagini, care constă în compararea unei imagini cu un şablon.

În cazul nostru, vom folosi această tehnică pentru a recunoaște cifrele de pe piesele de joc.

Vom folosi funcția *cv.matchTemplate* pentru a compara imaginea cu şablonul pentru a găsi numărul cu care se potrivește cel mai bine şablonul.

Observații

Din moment ce extragere nu produce imagini simetrice pentru a putea compara vom da resize imaginii şablon spre dimensiunea imaginii extrase, pentru a putea compara cele două imagini.

Vom aplica un simplu filtru de thresholding (binar + otsu) pentru a putea compara cele două imagini.

Funcția de calculare a scorului este *MSE* (Mean Squared Error).

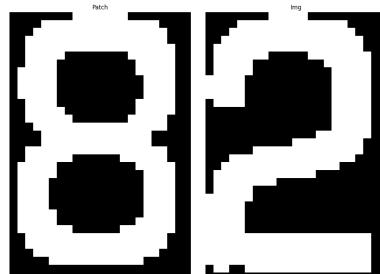
```

score = cv.matchTemplate(patch, img, cv.TM_SQDIFF_NORMED)
score = np.max(score)

if score < best_score:
    best_score = score
    best_match = digit

```

Astfel, vom încerca să recunoaștem cifrele de pe piesele de joc folosind un algoritm de template matching.



Comparare a două imagini 8 și 2

Optimizare Template Matching

Din moment ce avem un număr limitat de piese, mai exact pentru toate piese cu numere de la 1 la 10 avem 7 exemplare, iar pentru restul numerelor doar unul, putem să ținem cont de această informație înainte de a aplica template matching pe un număcar care nu mai poate fi plasat pe tabla de joc.

Din moment ce jocul este unul matematic, putem să ne folosim de acest lucru pentru vedea dacă un număr are sens să fie plasat pe tabla de joc verificând dacă există cel puțin 2 piese în stânga, dreapta, sus sau jos de poziția unde vrem să plasăm piesa, iar în caz că există vedem dacă există o ecuație validă (adunare, scădere, înmulțire, împărțire) între aceste numere care să dea rezultatul şablonului pe care vrem să îl folosim în comparație.

Astfel folsind aceste optimizări reduce numărul de comparații și sporim acuratețea algoritmului nostru.

Task 3

Acest task este cel mai simplu din cele trei, deoarece avem toate informațiile necesare pentru a calcula scorul. Scorul se calculează în funcție de poziția pieselor de joc și de numerele de pe acestea.

Scorul se calculează astfel:

- Notăm cu nr numărul de vecini care formează o ecuație validă, egală cu numărul curent de pe piesă.
- Dacă ne aflăm pe un pătrat de constrângere (albastru), notăm cu nr numărul de vecini care formează o ecuație descrisă de pătrat egală cu numărul curent de pe piesă.
- Dacă ne aflăm pe un pătrat alb sau albastru, scorul se calculează ca fiind $nr \times piesa$.
- Dacă ne aflăm pe un pătrat violet, scorul se calculează ca fiind $2 \times nr \times piesa$.
- Dacă ne aflăm pe un pătrat galben, scorul se calculează ca fiind $3 \times nr \times piesa$.

```
def find_all_eq_score(x, y, curr_candidate):  
    cnt = 0  
    for special_sign in ['/','-','+', '*']:  
        if table_information[y][x] == special_sign:  
            for direction in ['left', 'right', 'down', 'up']:  
                if check_operation(x, y, curr_candidate, direction, special_sign):  
                    cnt += 1  
    return cnt * curr_candidate  
  
for direction in ['left', 'right', 'down', 'up']:  
    for operation in ['/','-','+', '*']:  
        if check_operation(x, y, curr_candidate, direction, operation):  
            cnt += 1  
            break  
  
multiplier = 1  
if table_information[y][x] == 'Y':  
    multiplier = 3  
if table_information[y][x] == 'V':  
    multiplier = 2  
  
return cnt * curr_candidate * multiplier
```

Rămâne doar să aplicăm aceste reguli pentru fiecare tură a jucător, să calculăm scorul final și să afișăm răspunsurile pentru fiecare task.



Bucă Mihnea-Vicențiu

Anul 3 Informatică

Grupa 352