

Mathable Vision Solver

Gabroveanu Razvan

December 2, 2024

Contents

1	Introducere	1
2	Extragerea tablei relevante de joc	2
2.1	Eliminarea background-ului	2
2.1.1	Obtinirea si aplicarea mastii binare	2
2.1.2	Croparea tablei de joc	3
2.2	Eliminarea marginii tablei	5
2.2.1	Obtinirea mastii	5
2.2.2	Croparea tablei de joc	6
3	Iterarea prin celulele jocului	7
4	Extragerea template-urilor	8
4.1	Simpla extragere	8
4.2	Binarizare template-uri	9
4.3	Cropare template-uri	10
5	Clasificare pe setul de imagini	12
5.1	Preprocesare	12
5.2	Clasificarea unui patch	13
5.2.1	O prima euristica	13
5.2.2	O a doua euristica	13
5.2.3	Euristica fail-safe	13
5.3	Algoritmul de clasificare	13
5.3.1	Reprezentarea tablei de joc	13
5.3.2	Cautarea piesei noi adaugate	14
6	Calcularea scorului	15

1 Introducere

Urmeaza sa descriu pas cu pas flow-ul proiectului si procesarile aduse pe imagini pana la obtinerea rezultatelor finale.

2 Extragerea tablei relevante de joc

Tabla relevanta de joc este de fapt grid-ul, fara marginea redundanta care este nefolosita.

2.1 Eliminarea background-ului

Scopul este de a elimina detalii inutile din imagine. Avem nevoie de o imagine clara a tablei de joc.

2.1.1 Obtinirea si aplicarea mastii binare

Pentru a elmina background-ul tablei de joc(anume biroul pe care imaginile tablei de joc au fost facute) este necesar sa aplicam o masca asupra imaginii care sa elime culoarea maronie a biroului. Masca binara se obtine prin transformarea pixelilor cu valori intre 0-20 a valorii Hue din imaginea in format HSV in pixeli negri si restul pixelilor vor deveni albi. Valorile 0-20 reprezinta culori de la rosu la culoarea maronie a backgroundului, iar valoarea a fost descoperita prin trial and error. Masca se aplica asupra imaginii folosind operatorul "bitwise and" si rezultatul o sa fie asemantanor urmatorului exemplu:

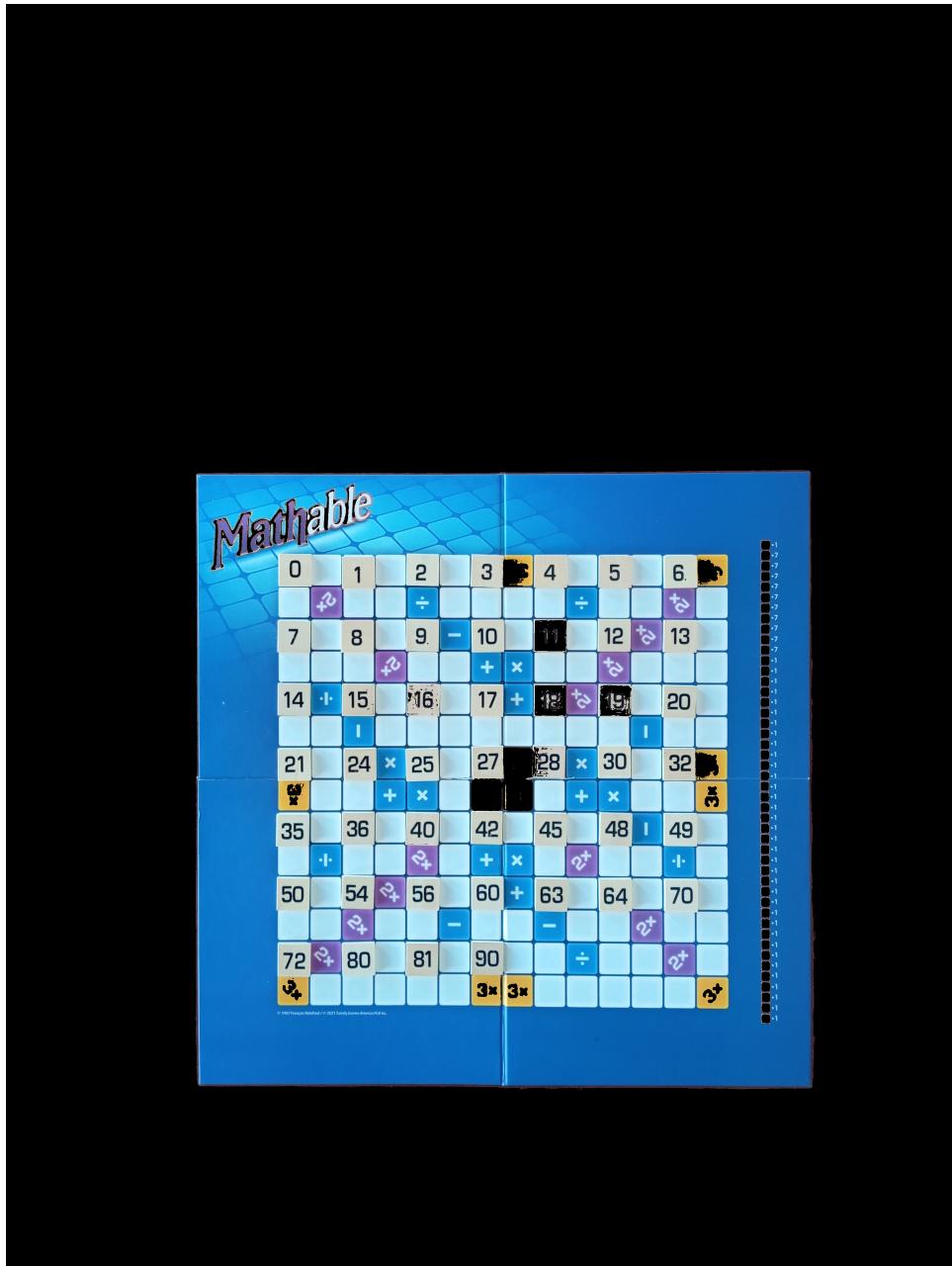


Figure 1: Imaginea mascata.

2.1.2 Croparea tablei de joc

Pentru a croapa imaginea obtinuta anterior este necesar sa obtinem coordonatele colturilor tablei. Pentru a realiza asta vom proceda in felul urmator:

- Se duce imaginea in grayscale.
- Se aplica un filtru median pentru cizelare.
- Se obtin coordonatele marginilor contururilor din imagine folosind functia `cv2.getcontours()`.
- Pentru fiecare contur:

- Se cauta punctele ce reprezinta colturile (cel mai de sus din stanga, cel mai de sus din dreapta, cel mai de jos din dreapta si cel mai de jos din stanga).
 - Se calculeaza aria pentru figura formata de colturile extrase.
 - Daca aria este maxima se salveaza in variabile punctele.
- La final avem punctele conturului cu aria maxima, anume colturile tablei de joc.

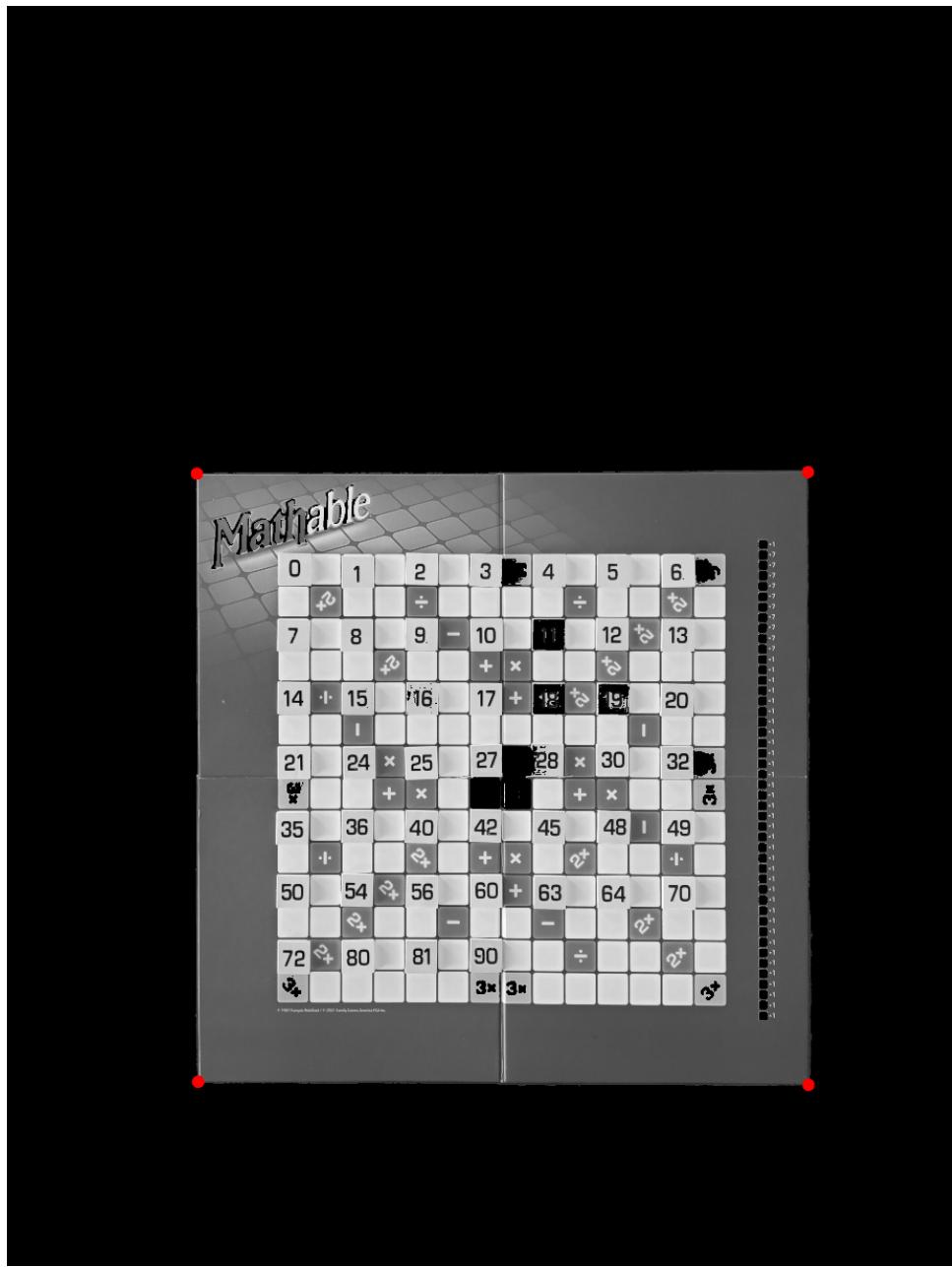


Figure 2: Colturi detectate.

Acum ca avem coordonatele necesare ramane sa cropam imaginea din imaginea initiala, fara masca aplicata. Pentru a standardiza dimensiunea imaginilor rezultate vom stabili inaltimea si latimea si vom folosi functiile **cv2.getPerspectiveTransform()** si **cv2.warpPerspective()** pentru a obtine perspectiva pixelilor colturilor si de a extrage

tabla cu marimile prestabilitate. Marimea noii imagini nu este relevanta deoarece un procedeu asemanator urmeaza sa fie aplicat pe imaginea rezultata.



Figure 3: Colturi detectate.

2.2 Eliminarea marginii tablei

Pentru a avea o perspectiva cat mai buna asupra jocului, avem nevoie doar de matricea acestuia, precum la sudoku. Pentru asta trebuie sa eliminam marginea jocului care este redundanta, folosind o strategie asemanatoarea cu cea de la **eliminarea backgroundului**.

2.2.1 Obtinirea mastii

De data aceasta, nu s-a folosit mascarea folosind valoare Hue din imaginea dusa in HSV, deoarece marginea nu are o culoarea fixa si prezinta detalii ce pot sa nu fie mascate doar prin culoare(simbolul jocului, gridul inclinat din stanga sus, piese din marginea din drapta). Asa ca abordarea de data aceasta este prin hardcodare. De la **eliminarea backgroundului** stim ca fiecare imagine are acum o marime fixa, deci putem hardcoda o masca pe margini. In principiu am observat ca masca are in toate directiile cam 13-14% din latura imaginii. Valorile pot sa difere pentru fiecare directie, obtinandu-se prin trial and error.

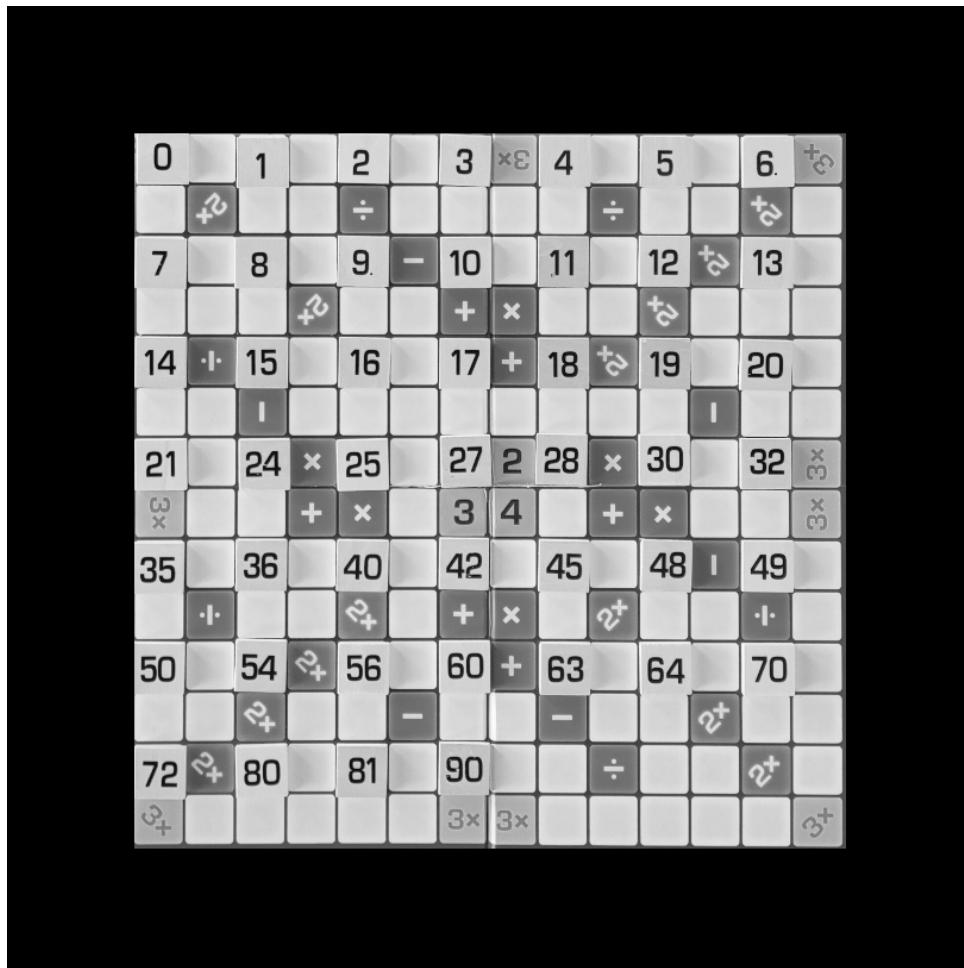


Figure 4: Tabla cu marginea mascata.

2.2.2 Croparea tablei de joc

Pentru a croapa noua imagine se va folosi exact acelasi procedeu descris la **croparea backgroundului**, cu o mica ajustare: dimensiunea imaginii cropare o sa fie in functie de marimea dorita a unei celule de pe tabla de joc. In implementarea mea fiecare celula ar trebui sa fie de 150x150 pixeli, deoarece template-urile pentru clasificarea fiecarei celule vor fi obtinute din aceste imagini si este necesar sa aiba aceeasi dimensiune.

0	1	2	3	$\times 3$	4	5	6	$+ 3$
7	8	9	-	10	11	12	$\div 2$	13
14	\cdot	15	16	17	$+ 3$	$\div 2$	19	20
	-						-	
21	24	\times	25	27	2	28	\times	30
$\times 3$		$+ \times$		3	4	$+ \times$		$\times 3$
35	36	40	42	45	48	-	49	
\cdot		$\div 2$	$+ \times$	$\div 2$	$\div 2$		\cdot	
50	54	$\div 2$	56	60	$+ 3$	63	64	70
	$\div 2$		-		-		$\div 2$	
72	$\div 2$	80	81	90		\div	$\div 2$	$\div 3$
$\div 3$				$\times 3$	$\times 3$			$\div 3$

Figure 5: Matricea tablei.

3 Iterarea prin celulele jocului

In acest moment stim ca avem imagini de 2100x2100(14x14 celule a cate 150x150 pixeli fiecare) reprezentand matricea jocului. Pentru a itera si marca fiecare celula vom crea doua liste de linii verticale si orizontale, prin salvarea coordonatelor de inceput si sfarsit a fiecarei linii.

- Coordonate linii orizontale: $[(0, i), (14 \cdot \text{cell_size} - 1, i)]$, unde $i = n \cdot \text{cell_size}$, $n \in \{0, 1, 2, \dots, 14\}$.
- Coordonate linii verticale: $[(i, 0), (i, 14 \cdot \text{cell_size} - 1)]$, unde $i = n \cdot \text{cell_size}$, $n \in \{0, 1, 2, \dots, 14\}$.

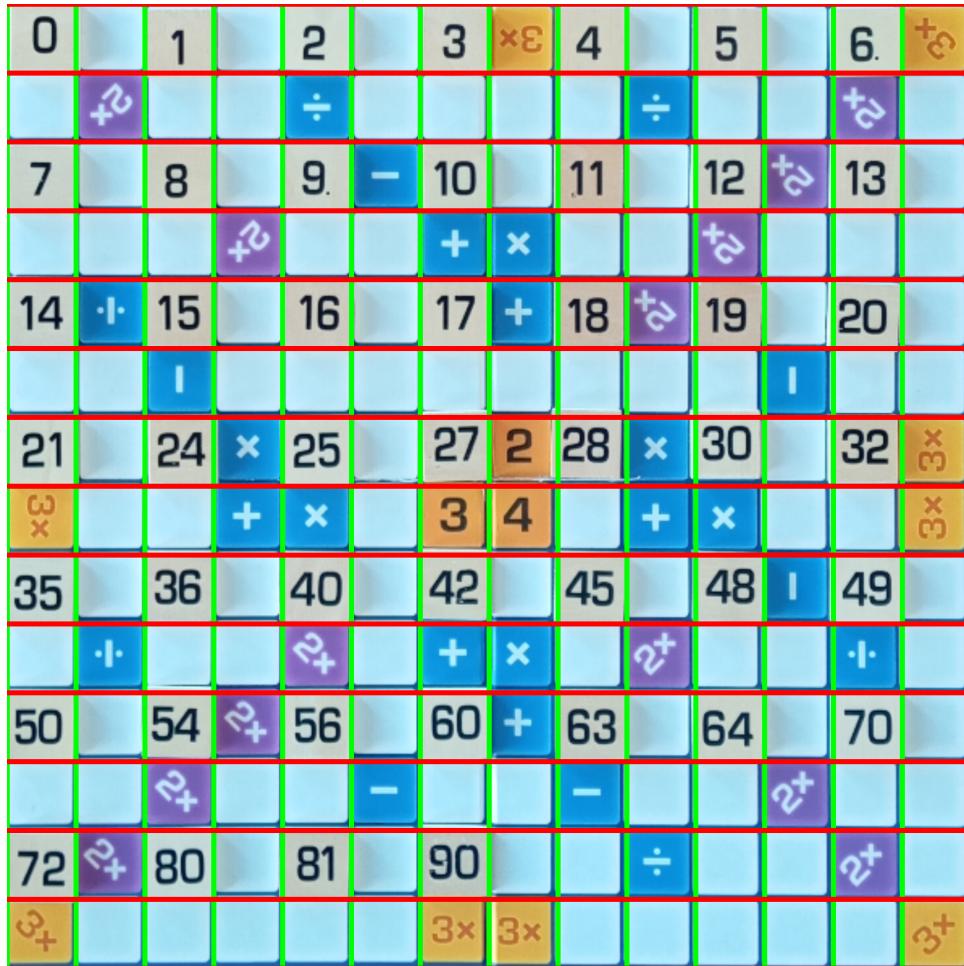


Figure 6: Linii trasate pe grid.

4 Extragerea template-urilor

Inainte de a incepe clasificarea pieselor dintr-o imagine este nevoie sa avem pregatite template-uri cu numerele de pe piese care sa fie potrivite pentru a fi folosite cu functia cv2.matchTemplate().

4.1 Simpla extragere

Template-urile folosite sunt extrase din imaginea auxiliara obtinuta in urma **croparii matricei jocului**. Pur si simplu se itereaza prin coordonatele liniilor verticale si orizontale calculate anterior si se extrage patch-ul curent reprezentand o celula. De asemenea, se aplica o cropare mica; un offset, in caz ca unele celule nu sunt perfect centrate si pot sa apară in imagine marginile jocului.

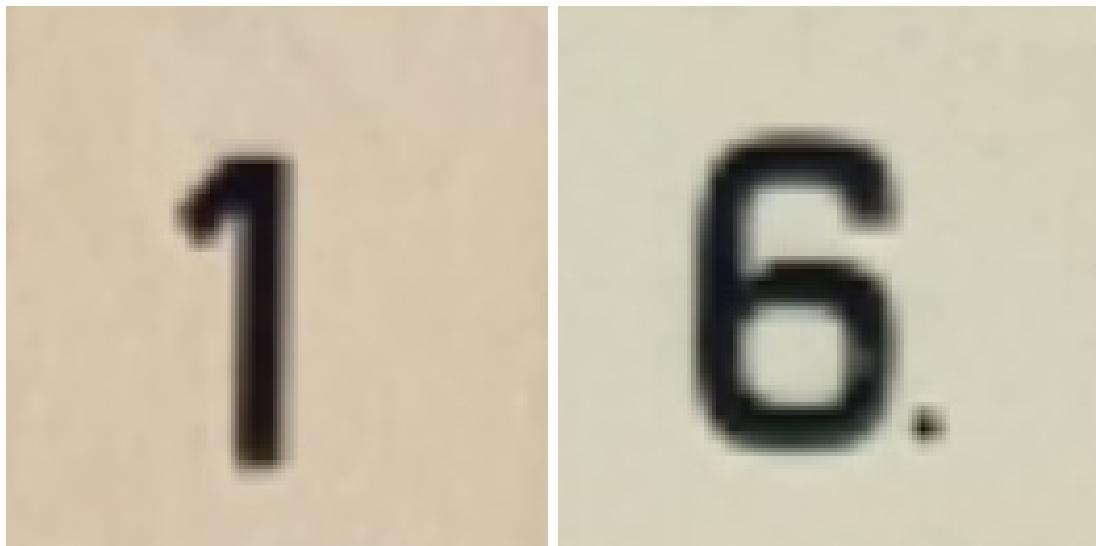


Figure 7: 1

Figure 8: 6

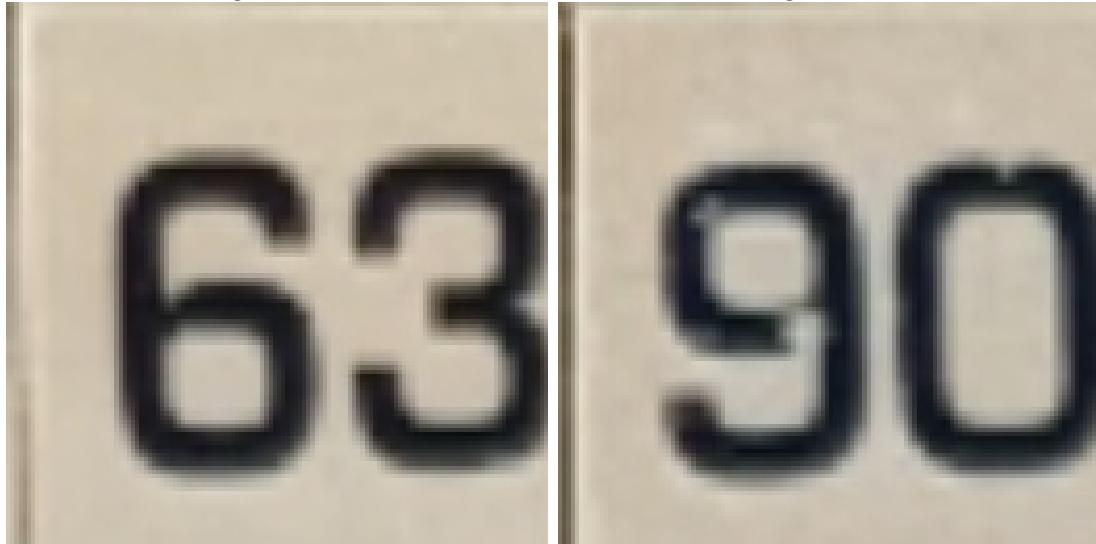


Figure 9: 63

Figure 10: 90

4.2 Binarizare template-uri

Pentru a putea exclude problemele cauzate de luminozitate si diferențe de nuanta din imagini cand o sa facem matching-ul este nevoie sa binarizam imaginile. Fundalul imaginilor va deveni negru, iar cifrele vor deveni albe. Pentru asta se vor duce template-urile extrase in grayscale si se va folosi functia `cv2.threshold()` pentru binarizare. Pragul l-am ales in functie de trial and error.



Figure 11: 1

Figure 12: 6

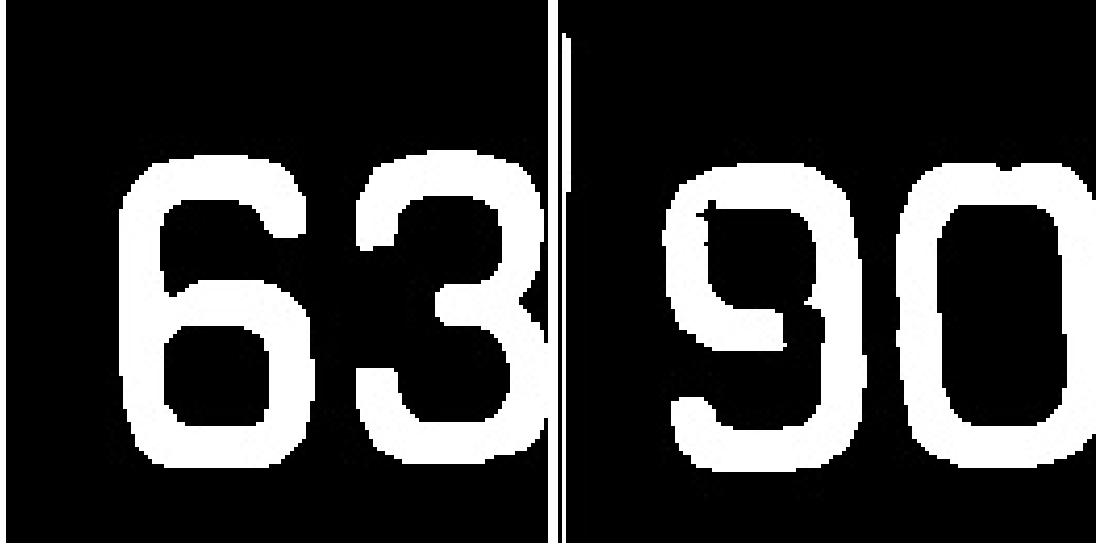


Figure 13: 63

Figure 14: 90

Nota: Se poate observa cum in imaginea 90 mai apare inca marginea stanga si 6 are un cluster de pixeli albi in dreapta. Aceasta problema se va rezolva in pasii urmatori.

4.3 Cropare template-uri

Pentru a mari sansa de match este nevoie de a elimina informatia redundanta din template-uri, anume cat mai mult din marginile negre. Pentru asta am folosit un algoritm simplu care imi elibera toate liniile si coloanele din ambele directii care au numai pixeli negri.



Figure 16: 6

Figure 15: 1



Figure 17: 63

Figure 18: 90

Nota: Din cauza unor margini albe metoda asta nu a functionat pe toate template-urile. Pentru a cropa imaginile euseate am folosit functia **cv2.selectROI()** pentru a cropa manual fiecare imagine.

5 Clasificare pe setul de imagini

Acum ca avem la dispozitie template-uri binare cropate si tabla de joc clara putem sa incepem sa implementam algoritmul de clasificare imagine cu imagine pe data set-ul de 50 de imagini.

5.1 Preprocesare

Imaginiile cu tabla de joc obtinute anterior sunt in format RGB, iar template-urile sunt in format binar. Pentru a face matching trebuie sa binarizam imaginile cu tabla. Se duc imaginile in grayscale si apoi se aplica ca mai devreme functia **cv2.threshold()**. Pragul de data aceasta este ales astfel incat sa subtieze cat mai mult marginile tablei, dar fara sa se piarda forma cifrelor.

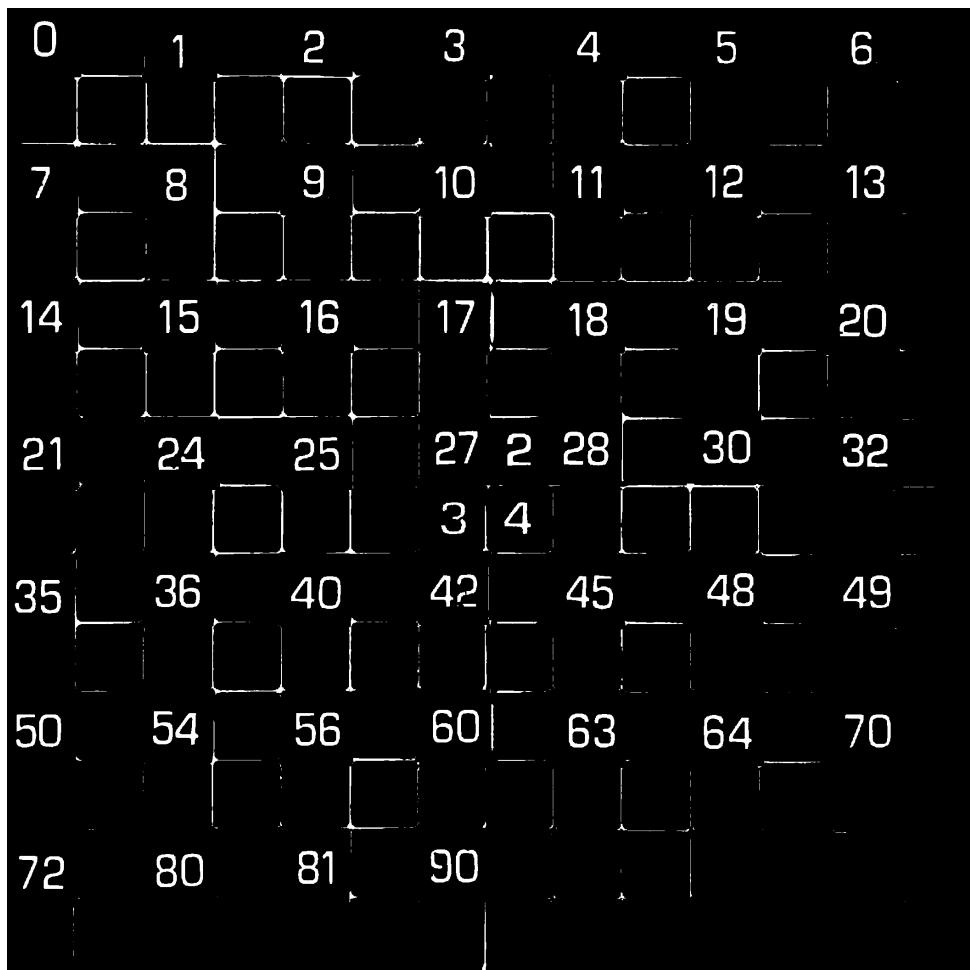


Figure 19: Tabla de joc in format binar.

5.2 Clasificarea unui patch

Proiectul contine o functie care foloseste diverse "euristici" pentru a avea o clasificare cat mai precisa. In principiu ideea naiva a clasificarii unui patch ar fi sa iteram prin toate templateurile, sa aplicam functia `cv2.matchTemplate()` intre patch si template si la final sa alegem patch-ul cu corelatia cea mai mare. Aceasta abordare de multe ori esueaza deoarece pentru un numar format din doua cifre, un template ce reprezinta o cifra poate sa obtina o corelatie mult mai mare fata de template-ul corect. De exemplu, pentru un patch cu 19 template-urile cu cifrele 1 sau 9 pot sa obtina o corelatie mai mare fata de template-ul 19.

5.2.1 O prima euristica

O solutie pe care o folosesc este ca mai intai sa iterez prin toate template-urile ce reprezinta numere formate din doua cifre si sa le testez pe ele primele. Daca corelatia este suficient de mare(0.75 in cazul meu), algoritmul se opreste si considera ca acela este template-ul corect. Alegerea pragul al corelatiei este, evident, ales prin trial and error. Daca niciun template de doua cifre nu coreleaza suficient de mult cu patch-ul curent, se incercă si template-urile ce reprezinta o cifra.

5.2.2 O a doua euristica

O imbunatatire la **metoda precedenta** care poate sa creasca acuratetea este sa exploatam logica jocului. O celula poate sa ia doar anumite valori din toate piesele. Aceste valori sunt calculate prin folosirea operatorilor `+, -, *, :, ;` intre cei doi vecini de sus, jos, stanga si dreapta. Astfel indicat ar fi ca functia de clasificare al patchului sa vina cu o lista de valori posibile pe care le poate lua celula curenta. Din lista aceasta se ignora valoare carora nu le corespunde o piesa si clasificarea apoi se foloseste metoda de la **punctul precedent** doar pe aceste valori.

5.2.3 Euristica fail-safe

Daca se intampla cumva sa esueze imbunatatirea de la **punctul anterior**, iar patch-ul curent inca nu este clasificat, se va folosi metoda de acum doua puncte.

5.3 Algoritmul de clasificare

5.3.1 Reprezentarea tablei de joc

Prima data se incepe cu o tabla goala de joc de 14x14 cu valori setate la -1. La fiecare imagine ne vom folosi de aceasta matrice pentru a stii care piese au fost deja clasificate si care piesa este noua, cum doar una este adaugata per poza. Valorile default din mijlocul tablei se regasesc si ele in matrice, deoarece, dupa cum s-a vazut din imaginile din exemple, acestea pot sa fie confundate cu piese.

Figure 20: Configuratia tablei de joc.

5.3.2 Cautarea piesei noi adaugate

Conform **coordonatelor liniilor orizontale si verticale** se itereaza prin celulele neclasificate folosindu-ne de reprezentarea tablei de la **punctul anterior**. Daca patch-ul curent al celului curent contine pixeli albi in mijlocul acestuia inseamna ca acesta trebuie clasificat. Se calculeaza valorile posibile pe care le poate lua celula si se foloseste **functia de clasificare al unui patch** pentru a-i gasi valoarea potrivita. In continuare se marcheaza valoarea patch-ului in matricea tablei si se trece la urmatoarea imagine.

```

[[ -1 -1 -1 -1 -1 -1 -1 18 -1 5 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 10 -1 2 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 8 -1 3 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 80 -1 6 -1 48 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 10 -1 9 -1 36 -1 -1]
 [-1 -1 -1 3 14 11 3 8 5 3 15 12 27 -1]
 [-1 -1 -1 -1 -1 -1 1 2 3 6 9 3 -1 -1]
 [-1 -1 -1 -1 -1 7 3 4 7 9 63 -1 -1 -1]
 [-1 -1 -1 -1 8 10 2 8 10 -1 54 -1 -1 -1]
 [-1 -1 -1 5 8 3 5 32 17 49 -1 -1 -1 -1]
 [-1 -1 -1 -1 0 -1 -1 40 7 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 70 -1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1 -1 -1 60 -1 -1 -1 -1 -1]]
```

Figure 21: Configuratia tablei de joc la finalul procesarii data set-ului.

6 Calcularea scorului

Calcularea scorului fiecarui jucator este un task relativ simplu dupa ce restul au fost implementate; se realizeaza simultan cu clasificarea fiecarei imagini. La fiecare piesa noua adaugata pe tabla de joc, se verifica cate operatii se pot face in fiecare directie pentru a obtine valoarea noua adaugata(scorel multiplicativ). De asemenea, se tine cont de fiecare constangere de pe tabla printr-o matrice hardcodata cu fiecare constangere. La fiecare imagine noua se calculeaza scorul obtinut si se adauga la totalul curent al jucatorului. Cand celalalt jucator isi incepe turul, datele sunt salvate in fisier si scorul este resetat.

Figure 22: Constrangerile hardcodate.