Calculator automat de scor pentru jocul Mathable

Anghel Ioan-Tudor-Alexandru

2nd December 2024

Abstract

Acest proiect are ca scop crearea unui sistem de analiza a tablelor de joc Mathable si urmarirea miscarilor facute de jucatori, pentru a calcula scorurile obtinute de acestia pe parcursul rundelor. Aceasta sarcina a fost impartita in trei parti: detectia unei noi piese plasate intr-o miscare, specificand indexul acesteia, identificarea numarului de pe piesa si intr-un final, calcularea scorului asociat fiecarei runde.

Sistemul creat si descris in aceasta lucrare foloseste tehnici de segmentare a imaginilor, detectare a contururilor si pattern matching pentru a indeplini aceasta sarcina.

Solutia propusa are o performanta de 100% acuratete pe seturile de antrenare si validare, insa nu este una foarte robusta, bazandu-se pe caracterul constant al seturilor de date, in ceea ce priveste perspectiva asupra tablei, luminozitatea si zona din care vine lumina. Asadar, parametrii gasiti prin testare ar putea sa nu fie aplicabili in alte situatii.

1. Introducere

Mathable este un joc similar cu Scrabble, bazandu-se pe ecuatii matematice care trebuie formate pe tabla de joc, in loc de cuvinte.

Jucatorii folosesc o tabla patrata, delimitata de un careu albastru, care contine patrare normale, de culoare albastru deschis, patrate ce constrang jucatorul sa foloseasca numai anumite operatii, de culoare albastru inchis si patrate bonus, acestea fiind mov sau portocalii pentru dublarea, respectiv triplarea scorului. Acestea sunt impartite intr-o grila de 14x14.

Pisele folosite au forma patrata, numerele fiind scrise cu negru, pe un fundal alb. Acestea sunt plasate deasupra, dedesuptul, in dreapta sau in stanga a doua piese cu ajutorul carora se poate forma o ecuatie corecta.

Scorul pe care il obtine un jucator intr-o miscare este egal cu valoarea piesei plasate, inmultita cu numarul de ecuatii pe care il satisface. In cazul in care aceasta este plasata pe un patrat ce contine un multiplicator, scorul este inmultit cu el.

$$Scor_{mutare} = Valoare_{piesa} * Multiplicator_{scor} * Numar_{ecuatiisatisfacute} \quad (1)$$

2. Detectarea si identificarea unei piese nou plasate

Prima parte a problemei se concentreaza pe detectarea piesei plasate in miscarea curenta.

2.1 Detectarea tablei si a zonei de joc

Un prim pas este separarea tablei de fundal. Acest lucru se poate rezolva prin gasirea mastii care corespunde marginii ei si decuparea imaginii pentru a include doar aceasta parte. Tabla trebuie apoi redimensionata, pentru a evita variatii in numarul de pixeli. Zona de joc se obtine in acelasi mod, putand astfel sa calculam coordonatele si dimensiunile patratelor.

O diferenta in detectarea zonei de joc fata de cea a tablei, pe langa aplicarea unei masti diferite, este faptul ca, dupa un anumit numar de runde, mare parte din aceasta va fi acoperita de piese de joc. Acest lucru duce la o segmentare gresita a imaginii, pragurile initiale nefiind potrivite in continuare. Pentru a rezolva aceasta problema, vom aplica doua masti, una pentru patratele de joc si una pentru piesele asezate, folsind reuniunea rezultatelor lor (bitwise or), pentru a aplica la final o functie de dilatare, cu un kernel destul de mare pentru a le uni pe acestea si careurile.

```
cutoutBoard = getBoard(image)
resizedBoard = resizeBoard(cutoutBoard, 1980, 1980)
cellsX, cellsY, cellsW, cellsH = getCells(resizedBoard)
cellW, cellH = cellsW // nCells, cellsH // nCells
```

2.2 Detectarea diferentei dintre miscari

Avand in vedere ca pozitia tablei de joc si perspectiva asupra acesteia nu se modifica considerabil de la o miscare la alta, diferenta principala dintre doua poze consecutive este chiar piesa nou plasata. Pentru a putea extrage cat mai multe informatii relevante, am folosit formatul HSV pentru a calcula aceasta diferenta. Am definit diferentele pe fiecare canal(Hue, Saturation si Value) si am aplicat un threshold binar, cu limitele inferioare si superioare diferite si gasite prin testare. (Fig. 1)

```
hueDifference = cv.absdiff(currentBoardHSV[:,:,0], previousBoardHSV[:,:,0])
_, hueThreshold = cv.threshold(hueDifference, 50, 60, cv.THRESH_BINARY)

satDifference = cv.absdiff(currentBoardHSV[:,:,1], previousBoardHSV[:,:,1])
_, satThreshold = cv.threshold(satDifference, 50, 55, cv.THRESH_BINARY)

valueDifference = cv.absdiff(currentBoardHSV[:,:,2], previousBoardHSV[:,:,2])
_, valueThreshold = cv.threshold(valueDifference, 50, 53, cv.THRESH_BINARY)

combinedDiff = cv.bitwise_or(hueThreshold, satThreshold)
```

```
combinedDiff = cv.bitwise_or(combinedDiff, valueThreshold)
```

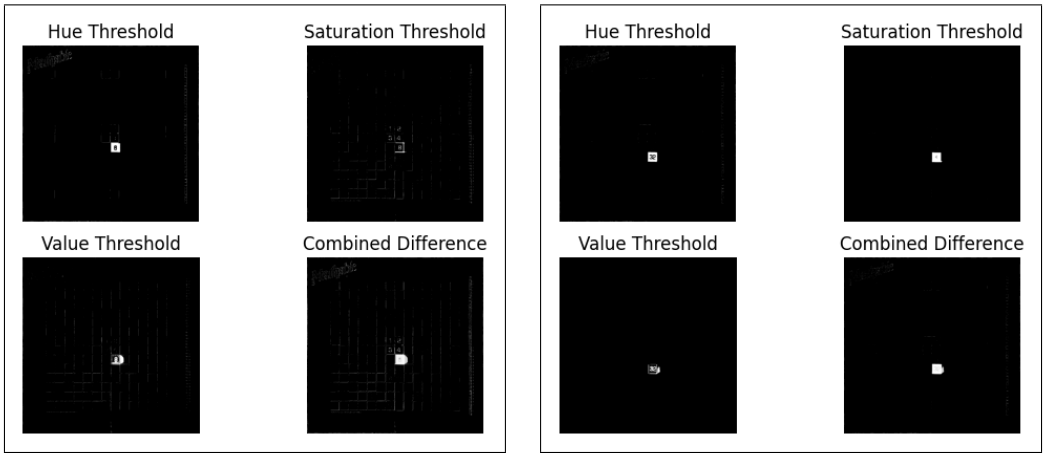

Figure 1: Diferente intre imagini, folosind canalele HSV.

2.3 Influenta canalelor analizate asupra rezultatelor

In urma testelor si vizualizarii rezultatelor, am putut trage anumite concluzii legate de impactul pe care il are fiecare canal asupra segmentarii si cum pot fi prioritizate acestea. Cea mai relevanta diferenta, in majoritatea cazurilor, a fost cea pe canalul Hue, care indica culoarea propriu-zisa si face cel mai usor deosebirea dintre un careu neocupat si unul ocupat (albastru deschis vs bej). Cu toate acestea, in anumite cazuri, rar intalnite, diferenta era una mica sau incompleta. Cel de-al doilea canal, Saturation, care indica cat de multa culoare este intr-un pixel, ofera diferente constante si relevante, indicand corect zona in care au avut loc schimbari. Cu toate acestea, zonele indicate de aceasta sunt mai generale decat cele identificate prin canalul anterior, identificand arii mai mari. Din acest motiv, mai departe am folosit prioritar canalul Hue, utilizand si Saturation in cazurile in care cel anterior nu ne ofera destule informatii.

2.4 Evidentierea zonelor cu energie ridicata din imagine pentru a restrange zona de interes

Desi imaginile plotate par sa ne ofere rezultate stabile, de multe ori, diferentele dintre imagini consecutive nu includ doar piesa nou plasata, ci si variatii ale tablei, luminii sau deplasari nevoite ale celorlalte piese. Acest lucru duce la detectarea unei zone mai mari unde s-a efectuat o posibila mutare, luand in calcul si piese nedorite.

Cum pentru un om este clar, uitandu-se la poza, ca o piesa nou plasata este in zona pixelilor de intensitate si densitate maxima, am decis sa incerc o maximizare a energiei din poza, cu accent pe zonele conectate de pixeli albi.

Am obtinut un rezultat multumitor aplicand o convolutie cu un kernel constant de dimensiuni 10x10. Intrucat operatia a amplificat si zgomotul din imagine, am aplicat o functie de thresholding pentru a il reduce inapoi.

Dupa acest pas, faptul ca sursa de lumina era plasata in zona stanga-sus a devenit un impas, intrucat diferentele in zonele acestea ale colturilor pieselor erau mai mici. Acest lucru ducea la o decupare prea agresiva a elementelor in aceasta zona, putand omite zone de informatie relevanta din acestea.

Pentru a combate aceasta problema, am incercat sa maximizez zonele opuse, de partea dreapta, de energie ridicata, aplicand un kernel cu valori mai ridicate pe jumatatea dreapta a coloanelor. Aceasta operatie a produs efectele dorite, extinzand pixelii albi si in zona stanga.

La finalul acestor operatii, pentru a elimina cat mai mult din zgomotul amplificat, am aplicat un filtru de eroziune de dimensiuni 9x9.

```
maxHueConentration = getMaxEnergyConcentration(hueThreshold)
_, maxHueThresholded = cv.threshold(maxHueConentration, 220, 255,
    cv.THRESH_BINARY)
maxHueThresholded = maximizeRightSideEnergy(maxHueThresholded)

kernel = np.ones((9, 9), np.uint8)
maxHueThresholded = cv.erode(maxHueThresholded, kernel)
```

Am aplicat aceste operatii si pe canalul Saturation, rezultatele acestea fiind folosite doar in cazul in care Hue nu identifica o zona destul de mare de schimbare, care ar fi putut fi potrivita cu o piesa.

2.5 Identificarea piesei nou plasate

Pasii anteriori au produs o masca destul de restransa pentru a include cu o precizie ridicata, doar noua piesa. Asadar, dupa aplicarea unei functii de gasire de contururi pe aceasta, cel de arie maxima o va indica.

Pentru identificarea numarului de pe piesa, vom folosi o functie de pattern matching dintre sabloane extrase cu piesele propriu-zise si zona identificata ca fiind noua piesa. Sablonul care ne ofera similitudine maxima va fi considerat ca fiind piesa plasata.

```
x, y, w, h = cv.boundingRect(largest_contour)

croppedBoard = resizedBoard[y:y+h, x:x+w]

identifiedPiece = matchPiece(croppedBoard)
```

2.6 Identificarea indexului piesei nou plasate

Functia folosita anterior, pentru a identifica valoarea piesei, ne indica si coordonatele punctului stanga-sus al celui mai potrivit sablon.

Mai departe ne folosim de coordonatele de inceput ale tablei de joc, dimensiunile unei casute si coordonatele determinate mai sus pentru a calcula indexul piesei plasate. Am aplicat un offset gasit prin testare, pentru a compensa anumite diferente neprevazute intre imagini.

```
y, x = y + identifiedPiece[1][0], x + identifiedPiece[1][1]
offset = 30
i, j = getCellIndex(
    cellW, cellH, x - cellsX + offset - 40, y - cellsY + offset - 20, nCells
)
```

3. Calcularea scorului unei mutari

Odata identificata o mutare, calcularea scorului presupune urmatorii pasi: luarea in calcul a posibilelor constrangeri de operatii, a posibilelor bonusuri si verificarea numarului de ecuatii satisfacute.

Indecsii ce introduc constrangeri sunt introdusi intr-un dictionar, iar la calcularea scorului, este suficienta verificarea existentei pozitiei in acesta pentru a determina daca o anumita operatie este impusa.

Definim o matrice multiplier, ce contine bonusurile pe indecsii corespunzatori si 1 in rest. Acest lucru duce la o aplicare mai usoara a acestora, pur si simplu inmultind scorul cu valoarea de la pozitia aferenta.

Pentru a verifica numarul de ecuatii satisfacute, vom lua in considerare perechiile de valori plasate in ture anterioare, alaturate noii piese, la dreapta, stanga, deasupra si dedesupt. Operatiile luate in considerare sunt fie indicate de constrangere, fie adunarea, scaderea, inmultirea si impartirea.

Aceste valori sunt inmultite, conform ecuatiei (1), pentru a calcula scorul mutarii.

4. Concluzii

Solutia propusa si descrisa mai sus foloseste tehnici clasice de computer vision, caracterul constant si cunoscut al datelor de intrare oferindu-ne un avantaj. Performanta obtinuta pe seturile de date furnizate se bazeaza pe aceste detalii, si pe valori obtinute prin testare si vizualizarea rezultatelor. Din acest motiv, modelul rezultat nu este unul robust si cel mai probabil nu ar avea o capacitate impresionanta de generalizare. Mai departe, pe langa perfectionarea si inmultirea capabilitatilor modelului, niste imbunatatiri rapide pot consta in verificarea validitatii unei mutari in ceea ce priveste constrangeri de operatie sau de pozitie. Mai mult, acuratetea ar putea fi imbunatatita adaugand verificari de existenta a piesei identificate si luarea in calcul a mai multor sabloane, in cazul in care nu ar mai exista aceasta.

In concluzie, aceasta lucrare prezinta parcursul de dezvoltare al solutiei create, si evidentiaza anumite parti cheie ale acesteia, ce au fost propuse ca rezolvari pentru probleme intalnite prin testare si vizualizarea efectelor operatiilor folosite. Desi pot fi identificate usor imbunatatiri pentru algoritm, utilizarea tehnicilor clasice de computer vision este in continuare o metoda utila, capabila de a genera rezultate consistente si relevante.