Extragerea informației vizuale din careuri Sudoku -Desrierea soluției

AVRAM DANIEL

La rularea fisierului *main.py* are loc instanțierea unei clase `Identifier` care conține tot codul de procesare. Funcția `process_image` din cadrul clasei primește ca parametru numărul testului, tipul (`'clasic'` sau `'jigsaw'`), și dacă să testeze cu bonus sau fără. Această funcție este singura interfață a clasei. Restul proceselor au loc "în spate". Unele din aceste sub-procese depind de unii hiper-parametri care sunt declarați în fișierul *config.py*. Sub-procesele sunt apelate în ordinea următoare:

1. Determinarea colțurilor careului.
2. Decuparea careului.
3. Împărțirea in 81 de celule.
4. Procesarea fiecărei celule și determinarea conținutului.
5. Dacă sudoku e de tip jigsaw, determinarea contururilor din interiorul careului.
6. Crearea răspunsului final.

1 Determinarea colțurilor careului

Funcția `__get_corners()` primește ca parametru imaginea (care deja a trecut prin `resize(img, fx=0.2, fy=0.2)`) și trece prin codul oferit de la laborator:

1. Imaginea este transformată în grayscale.
2. Imaginea este blurată cu un blur median cu `ksize=3` pentru a obține o imagine cu mai puțin zgomot.
3. Imaginea obținută este blurată cu un blur gaussian cu deviația standard de 7.
4. Ambele imagini, cea cu blur median si cea cu blur gaussian, sunt combinate într-una singură, având ponderile **1** și **-0.8** respectiv. Ponderea negativă pentru imaginea cu blur gaussian va crea un fundal alb pentru imaginea cu blur median în zonele cu pixeli negri (contururile careului, numerele, textul).

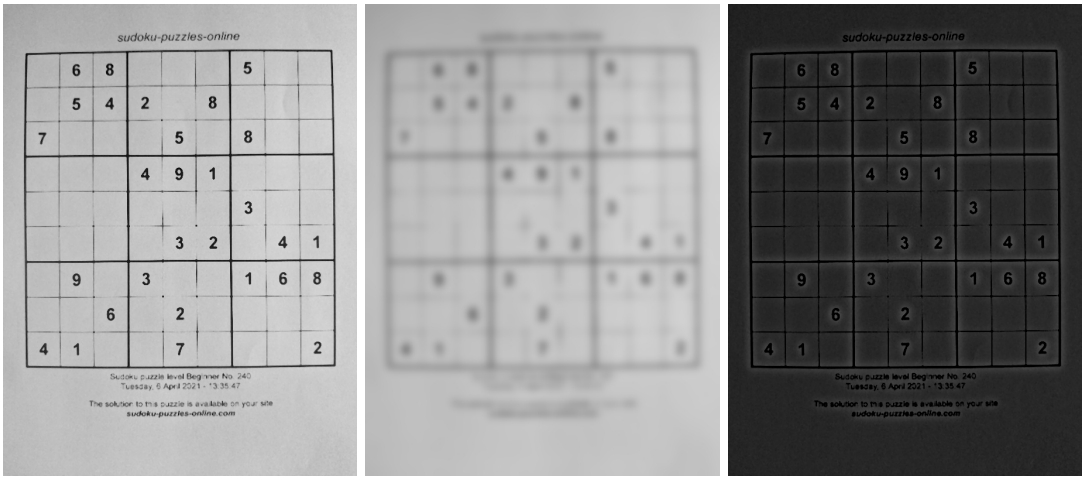

Figure 1: Imaginea cu blur median, gaussian si combinarea acestora (blur gaussian negat).

5. Pe imaginea obținută se aplică un threshold binar de 0 (toți pixelii cu o valoare mai mare de 0 vor fi 255). Vom obține o imagine binară cu contururile careului, cifrelor și a literelor.

6. Pe imaginea obținută se aplică Canny pentru a determina muchiile din imagine, iar aceste muchii sunt utilizate pentru a determina figurile din imaginea cu threshold. Figura care ne interesează este pătratul careului. Dacă în urma transformărilor cu blururile median și gaussian thresholdul a obținut un contur continuu al careului, acesta va fi determinat și de Canny + findContours.

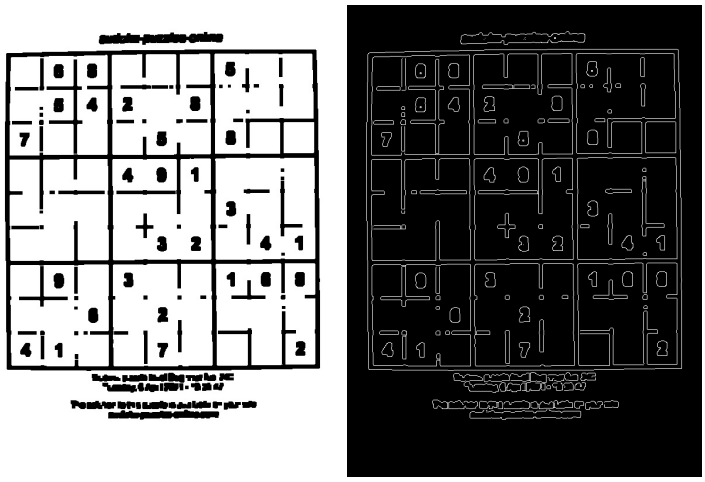

Figure 2: Threshold și marginile obținute cu Canny.

7. Segmentele care construiesc conturul sunt reprezentate prin puncte. Parcurgem fiecare contur (figură), parcurgem punctele din fiecare contur și găsim punctul cel mai din stânga-sus și punctul cel mai din dreapta-jos. Calculăm poziția aproximativă a punctelor stânga-jos și dreapta-sus și găsim punctele care sunt cel mai apropiate de aceste valori în conturul curent. Calculăm aria patrulaterului ai cărui vârfuri am găsit și calculăm aria cu `cv.contourArea`. Forma cu aria cea mai mare trebuie să fie conturul careului (din moment ce e cel mai mare obiect cu muchie continuă din imagine). Păstrăm coordonatele vârfurilor patrulaterului cu aria cea mai mare și le returnăm.

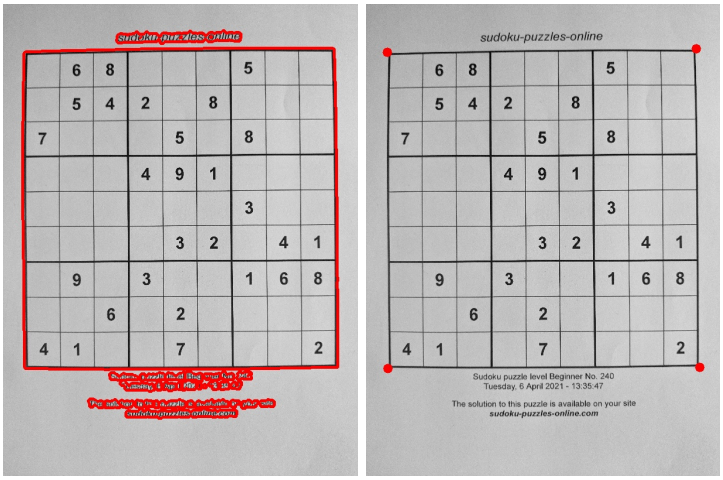

Figure 3: Contururile obținute și vârfurile celei mai mari figuri.

2 Decuparea careului

După ce obținem coordonatele vârfurilor careului, urmează să îl decupăm într-o imagine separată. Pozele făcute, însă, nu oferă o perspectivă "dreaptă" (muchiile careului nu sunt paralele cu marginile pozei sau camera nu se afla perfect paralel cu foaia la momentul fotografierii). Deci vârfurile obținute reprezintă un patrulater oarecare ce trebuie transformat într-un pătrat perfect. Funcția `__crop_image()` va transmuta vârfurile patrulaterului în colțurile unei imagini de *510*510* utilizând o matrice de transmutare obținută cu functia `cv.getPerspectiveTransform()` și va aplica această transmutare imaginii orinale cu funcția `cv.warpPerspective()`. Apoi, imaginii obținute i se aplică un padding de 3 pixeli pentru a scăpa de conturul exterior al careului, astfel încât imaginea finală să fie de dimensiunea *504*504* pentru a putea diviza in *9*9* celule.

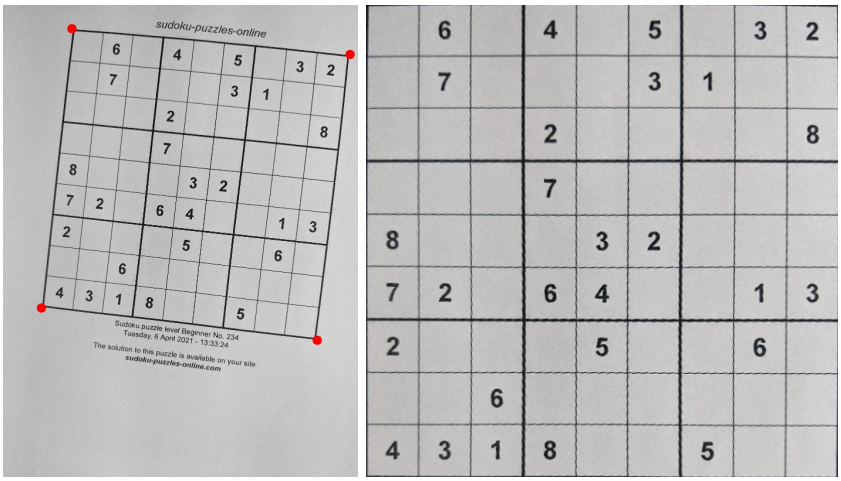

Figure 4: Decuparea prin transformare a unui careu a cărui muchii nu sunt paralele cu marginile pozei.

3 Împărțirea în celule

Pentru a obține celulele, în funcția `__get_results()` vom tăia careul cropat la punctul precedent în porțiuni egale de *56*56* și vom aplica un mic padding de 7 pixeli la careurile jigsaw și 5 pixeli la careurile clasice, pentru a scăpa de contururile interioare.

4 Procesarea celulelor

Pentru a determina interiorul unei celule, în funcția `__get_cells_map()`, mai intâi parcurgem pixelii celulei pentru a măsura valoarea lor medie, și apoi se determină cifra din interior utilizând *template matching*.

4.1 Verificarea dacă celula conține o cifră în interior sau nu

Pentru a verifica daca celula e goală sau nu, în funcția `__get_min_chunk()` aceasta este parcursă de un sliding window de dimensiunea *10*10* pentru a determina "fereastra" cu media cea mai mica. Dacă cea mai mică medie a unei celule e mai mic decât threshold-ul de 150, atunci această celulă conține o zonă de *10*10* care trebuie să aparțină unei cifre (nu poate aparține conturului datorită padding-ului efectuat la pasul precedent).

4.2 Determinarea cifrelor

Dacă în urma verificării s-a determinat că celula curentă conține o cifră, facem *template matching* cu funcția `cv.matchTemplate`. Această funcție încearcă să facă o potrivire între cifra din interiorul careului și un template, returnând gradul de potrivire între aceste două. Din moment ce fontul cifrelor din careurile clasice, careurile jigsaw grayscale și careurile jigsaw colorate diferă, a fost nevoie de extras fiecare cifră din fiecare tip de careu (în total, 30 de template-uri). Pentru fiecare celulă, deja știm tipul careului (clasic sau jigsaw), rămâne doar să determinăm daca e jigsaw colorat sau grayscale. Funcția `__is_colored()` calculează suma pixelilor pe fiecare canal BGR, apoi face diferența între suma cea mai mare și suma cea mai mică. Dacă celula e colorată, atunci această diferență va fi mai mare decât a unei celule grayscale, destul de mare pentru a le trece printr-un threshold. Desigur, parcurgerea fiecarei celule în parte ar lua prea mult timp, de aceea e destul doar de parcurs prima celulă din setul total de celule pentru a alege setul de template-uri respectiv. Înainte de a face potrivirea propriu-zisă, atât celula, cât și template-ul sunt trecute în grayscale și printr-un threshold de 127.

5 Determinarea contururilor din interiorul careului

Acest pas în funcția `__get_contours()` este similar cu determinarea conturului întregului careu: se accentuează imaginea cu blurul median, gaussian și thresholding; Se determină contururile cu `cv.Canny()` și `cv.findContours()`. Singura diferență e în thresholdul aplicat la `cv.thresh()`, din moment ce contururile la jigsaw grayscale sunt mai accentuate decât cele de la jigsaw colorat (tipul careului jigsaw îl știm de la pasul precedent).

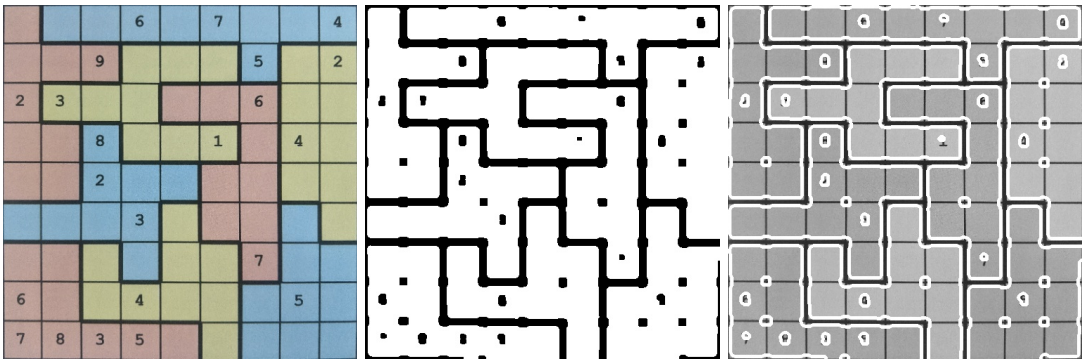

Figure 5: Imaginea cropuită este trecută prin blururi și threshold și apoi se găsesc contururile.

Urmează în funcția `__get_bolds_map()` maparea contururilor din imagine într-un array pe care să-l putem utiliza în cod. Fiecare celulă din imagine are 4 margini care pot fi sau nu conturate accentuat. Fiecare margină (orizontală și verticală) este verificată în modul următor: se ia punctul din centrul marginii (putem determina poziția prin calcule elementare) și se verifică distanța până la cel mai apropiat contur. Pentru fiecare punct, se parcurg toate contururile și se determină distanța cu funcția

`cv.pointPolygonTest()`. Dacă distanța e mai mică decât un threshold (10 pixeli la jigsaw colorat și 7 pixeli la jigsaw grayscale), atunci muchia respectivă punctului trebuie sa fie una accentuată. Cu maparea obținută și o parcurgere BFS obținem cele 9 zone ale careului.