Descrierea Solutiei

Brihac Andrei
Grupa 333

Codul poate fi gasit si la adresa: https://github.com/andrei-brihac/sudoku-cv

Detaliile despre librariile necesare si schimbarea fisierelor de intrare si iesire sunt in **README.md.** Proiectul poate fi rulat **executand src/main.py** intr-un **IDE** (eu am folosit **VSCode**) sau folosind comanda: ***python3 src/main.py***

Descrierea fiecarei metode este si in fisierul **src/sudoku.py** in docstring-urile functiilor. *(cred ca e mai usor de vizualizat in cod)*

Task 1 – sudoku clasic:

Primul obstacol este **extragerea imaginii cu careul de sudoku** din imaginea mare. Trebuie luat in calcul si faptul ca imaginea poate fi **rotita** la stanga sau la dreapta. Pentru a realiza acest lucru am facut urmatoarele procedee:

1. Am **procesat imaginea originala** astfel incat sa obtin o imagine cu mai putin zgomot si invarianta la schimbari de luminozitate in poza. Am aplicat:
- un **filtru Gaussian** de 9x9 pentru zgomot
- **adaptiveThreshold** pentru ‘normalizare’, deoarece este mai folositor pentru imagini cu luminozitati diferite in multiple zone
- **dilatare** cu un kernel de 3x3 pentru a evidentia si mai mult marginile

2. Utilizand functia **findContours** cu imaginea procesata am determinat punctele de pe **conturul zonei cu cea mai mare arie**, adica chiar **careul de sudoku**.

3. Am aflat punctele din acest contur care reprezinta cele **4 colturi** ale careului sudoku astfel:
- am determinat in functie de centrul imaginii **pozitia** in care se afla un punct din contur (**4 cadrane** – dreapta-jos, dreapta-sus, stanga-jos, stanga-sus)
- am retinut cate un punct din fiecare cadran, acelea care sunt mai **aproape** de colturile imaginii (folosind **distanta Manhattan**)
- punctele rezultate sunt chiar fiecare dintre **colturile careului sudoku**

4. Am folosit cele 4 colturi ale careului si functia **warpPerspective** pentru a **taia si roti** imaginea careului cu sudoku.

Al doilea obstacol este **aflarea starii celulelor** careului. Un careu sudoku are **9x9 celule**, deci este foarte usor de **impartit folosind numpy**. Dupa impartire, pentru a determina starea celulelor aplic 2 pasi:

1. Aplic o **taiere a celulei** cu o patrime pe fiecare margine pentru a **elimina orice linie ramasa** din careul de sudoku.

2. In celula ramane doar cifra si poate putin zgomot. Putem sa obtinem **media pixelilor** cu functia **numpy.mean** si sa o **comparam** cu o **valoare fixa** pentru care suntem **siguri** ca in imagine se afla o cifra. Aceasta valoare fixa poate fi obtinuta prin **a experimenta** cu afisarea valorilor mediei celulelor si **a observa** ce valori are pentru celule cu cifre. Imaginile celulelor din algoritmul meu sunt **imagini binare** cu cifre albe pe fundal negru si mi-a iesit **media 20**.

Task 2 – sudoku jigsaw:

Am folosit extragerea careului de la task-ul anterior.

Pentru a **identifica zonele** am aplicat urmatorii pasi:

1. Am **procesat imaginea** folosind:
- un **bilateralFilter de 9x9 si sigma 50**, deoarece este mai bun pentru a retine caracteristicile marginilor.
- **adaptiveThreshold**, la fel ca la primul task
- **eroziune cu un kernel de 2x2 de 3 ori**, ceea ce ii **elimina marginile subtiri**, dar pastreaza marginile groase
- **dilatare cu un kernel de 2x2** pentru a face marginile ramase mai proeminente.

2. Pentru a **eticheta fiecare celula** ma folosesc de **vecinii** ei.
- **vecinii** sunt definiti ca fiind celulele care **nu au o margine** intre ele si celula originala (**tai marginile celulei** si verific cu **numpy.mean** daca e linie separatoare)
- practic, este metoda descrisa in curs (cea cu imaginea cu soparla) in care **celula curenta preia numarul zonei celulelor vecine**
- in cazul in care vecinii au **etichete diferite** inseamna ca cele doua zone sunt **echivalente**
- **o zona noua** apare atunci cand niciunul dintre vecini nu este etichetat
- la sfarsit, **inlocuiesc etichetele echivalente** si aplic un **algoritm de fill** pentru a **renumerota zonele de la 1 la 9**

Solutia mea pentru task 2 merge pe 50% din datele de intrare deoarece procesarea imaginii si determinarea vecinilor nu sunt ideale. (linii lipsa datorita eroziunii, taiere necorespunzatoare etc.). Acest lucru se observa datorita faptului ca etichetele au valori mai mari de 9, adica ies mai multe zone decat ar trebui.