PREZENTARE INTERMEDIARĂ PROIECT GAEBO – MILESTONE 2 (DEMO TEHNIC)

AUTOR

ȘERBAN Alexandru-George (Project Manager)

Facultatea de Automatică și Calculatoare,

Universitatea “Politehnica” București

grupa 344C2, alexandru.serban00@stud.acs.upb.ro

ID Teams: 110133

COMPONENȚA ECHIPEI

+ Project Manager / Manager de proiect: Șerban Alexandru-George 344C2

+ Team Leader / Liderul echipei: Marii Hristofor 344C4

+ Software Developer / Dezvoltator software: Andrei Alin-Ionuț 344C3

+ Software Developer / Dezvoltator software: Timpuriu Mircea 342C5

+ Tester: Popa Marian-Elvis 342C5

NOȚIUNI TEORETICE INTRODUCTIVE

- Binarizarea – proces prin care se convertește o imagine (de obicei cu tonuri de gri) în alb-negru, fiind o metodă de segmentare

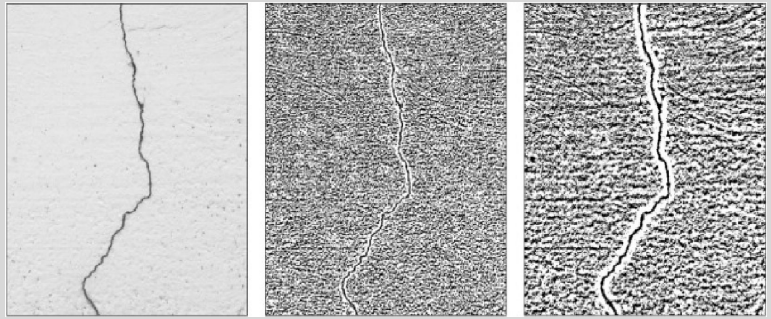
- Pentru determinarea tipului de pixel se ține cont de o valoare de threshold (prag de binarizare)

- Se pot nuanța 2 categorii de pixeli:
  - de fundal
  - ce conțin obiecte relevante (ex: text)

- Binarizare:
  - globală: se aplică un unic prag de binarizare pe întreaga imagine
  - locală: multiple praguri de binarizare pe ferestre diferite extrase din cadrul fișierului

- Performanța ideală (max. posibilă) – ground truth

- Aplicații posibile: preprocesări imagini, scanări medicale, securitate



PROGRES IMPLEMENTARE SOFTWARE ÎN MILESTONE 2

- Soluția software a proiectului GAEBO este implementa în Python 3 și unelte conexe precum Jupyter Notebook

- In cadrul milestone 2, s-a definitivat implementarea binarizarii globale

- Executarea surselor cod presupune obținerea unui arbore care returnează un prag global de binarizare – nodul rădăcină (root), cat mai aproape de ground truth, a cărui frunze sunt thresholduri canonice (ex. Otsu, Kittler)

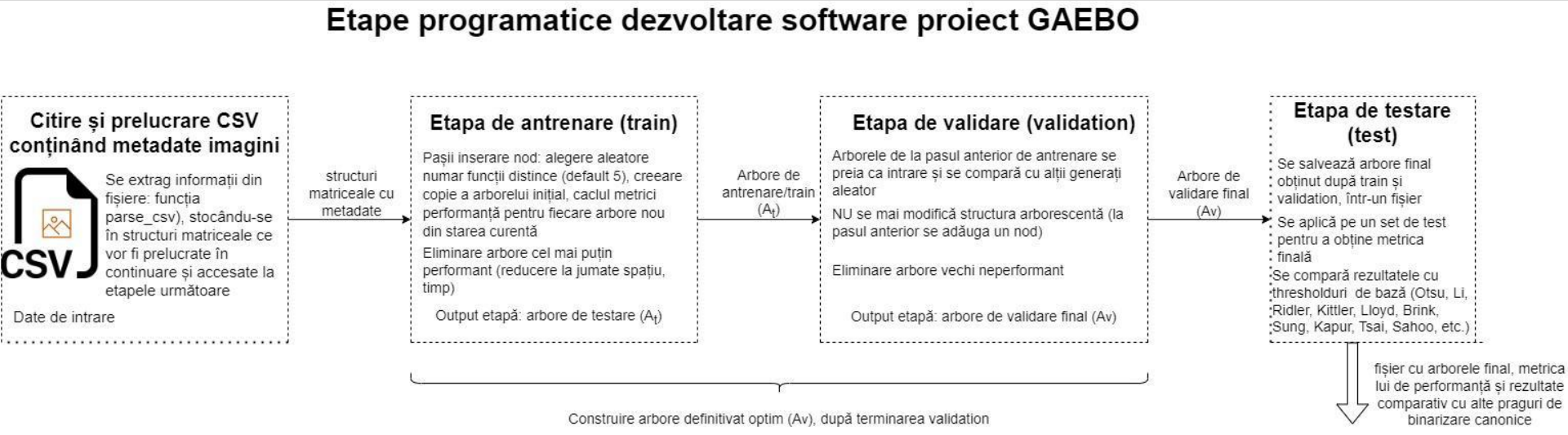
- Se ilustrează un algoritm optim pentru generare arbori globali, a căror noduri de tip rădăcină (root) reprezintă pragurile finale obținute

- F-measure ~70% pentru setul de test (partiționat dataset) la un timp de rulare de aproape 1s

- Alte performanțe arbori ce utilizează praguri de binarizare clasice: Li ~68%, Otsu ~67%, Shanbagh ~50%

ARHITECTURA SOFTWARE ÎN MILESTONE 2 (1)

- Etapele de dezvoltare programatică sunt de antrenare (train), validare (validation) și testare (test), dependente de prelucrarea de metadate asociate unor imagini în setul de fișiere CSV
- Modul de construire a arborelui este generativ, progresiv (poate fi asemuit unui pipeline)
- Introducerea de noduri tip rădăcină (root-uri) este etapizată și dependentă aplicării rezultatului de funcții compuse.
- Comparația de rezultate nu se rezumă doar la momentul inserării de noduri, ci este efectuată și asupra altor arbori generați aleatoriu, în deosebi în cadrul etapei de validare.



ARHITECTURA SOFTWARE ÎN MILESTONE 2 (2)

Surse cod din ierarhia de fișiere (soluția software)

- generator.py, scrie instrucțiunile din programul functions.py, acesta din urmă conținând implementarea unui număr fix (determinist) de funcții care au fost generate aleator

- *functions.py, creat doar dacă nu există deja, utilizându-se 16 proceduri/funcții de bază; conține corpurile funcțiilor ce vor fi folosite pentru calcularea nodurilor

- Global_binarisation.ipynb, notebook-ul Jupyter în care este implementat algoritmul principal de obținere a arborilor de threshold-uri.
  Conține 3 segmente:
  1. *de funcții*, implementează funcțiile necesare rulării
     - **parse_csv** extrage informațiile din fișierele CSV oferite, pe care le stochează în structuri de date matriceale care vor fi prelucrate în continuare;
     - **choose_random_starting_thresholds** generează un subset aleator de praguri (threshold-uri) frunză pentru un arbore;
     - **generate_new_node_in_tree** mecanismul de adăugare a unui nou nod de tip rădăcină (root) în arbore;

ARHITECTURA SOFTWARE ÎN MILESTONE 2 (3)

- **solve** aplică funcțiile din arbore pentru a obține un threshold în root, și returnează f-measure-ul threshold-ului obținut pentru o imagine;
- **solve_over_dataset** și **get_final_metrics** aplică funcția solve pe un întreg set de date, returnând metrica de performanță pe întreg setul de date;
- **train** execută etapa de antrenare;
- **validate** execută etapa de validare;
- **write_results_to_file** formatează output-ul pentru a-l face prezentabil;
- **get_known_threshold_metric** calculează performanțele threshold-urilor consacrate (Otsu, Kittler, etc.) asupra setului de test.

2. *de creare a seturilor de date, unde se citesc datele din fișierele CSV și se împarte setul total în mulțime de antrenament (50%), de validare (20%) și de test (30%)*

3. de determinare a arborilor, în care se aplică etapele de antrenare și validare, scriindu-se cei mai buni arbori în fișiere de output

Arhitectura software proiect de generare arbori pentru evaluarea de binarizări optime (GAEBO)

Soluția de binarizare globală

Citire, prelucrare fișiere CSV

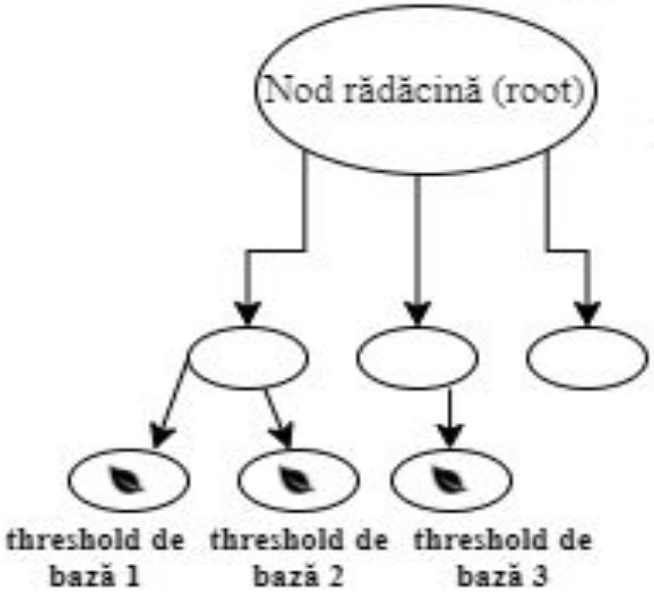
Set fișiere CSV ce conțin informații binarizare pe set imagini (metadate)

Stocare în structuri de date matriceale

Etapa de antrenare (train)

Comparație metrici performanță (între arbori parțiali și aceiași, dar completați prin adăugare nod rădăcină R)

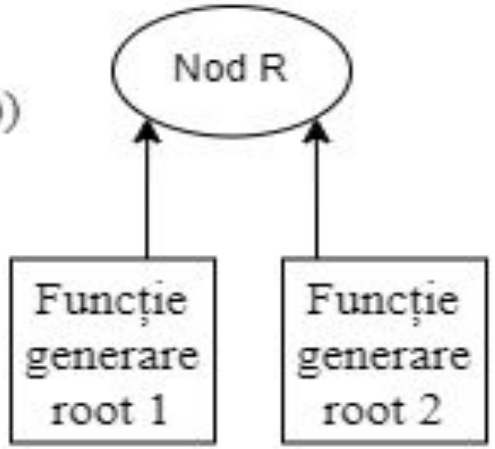
Arbore parțial (A_p)



?

$perf(A_p) > perf(insert(A_p, nod\ R))$

Inserare nod rădăcină R



Calcul metrică performanță pe un arbore

$perf(arb) = avg(F\text{-}measure_{dataset}) - 10 * T_{exec}$

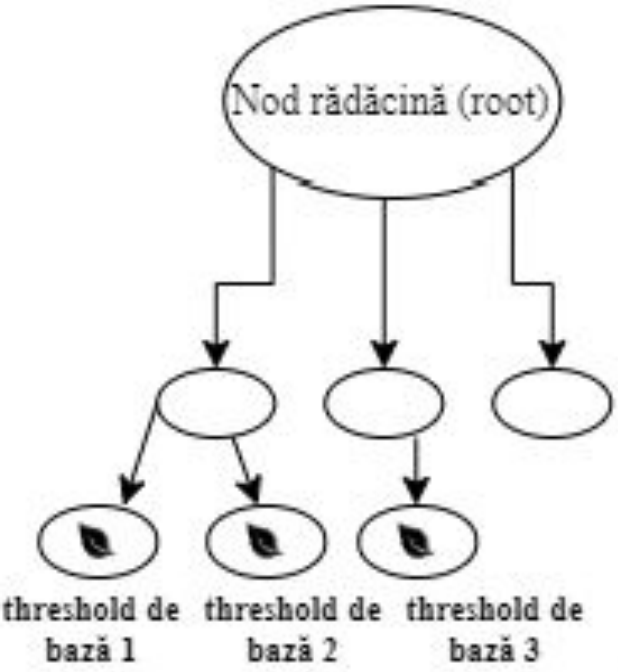
Arbore optim train (A_t)

Etapa de validare (validation)

Arbore optim train (At)

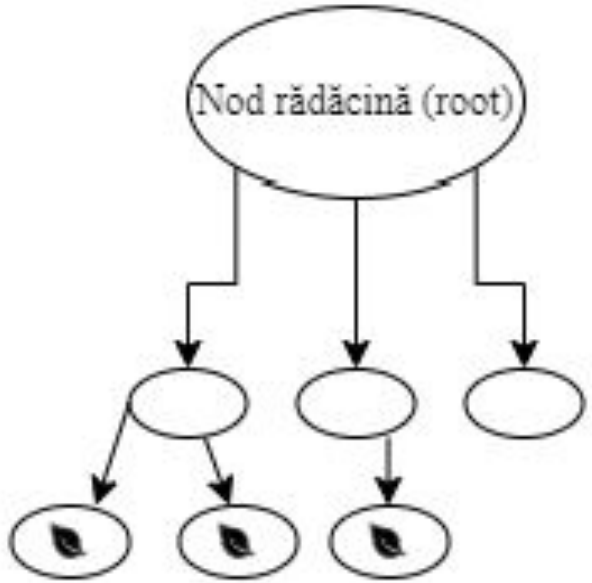
Comparație metrici performanță (între arborii de train și alții generați random)

Arbore generat aleator (Arand)



?

perf(At) > perf(Arand)



Arbore optim validare (Av)

Etapa de testare (test)

Salvare arbore optim de validare (Av) - final obținut algoritmului de optimizare - într-un fișier de ieșire

Extragere informații de performanță (metrică finală)

Performanță: ~70%

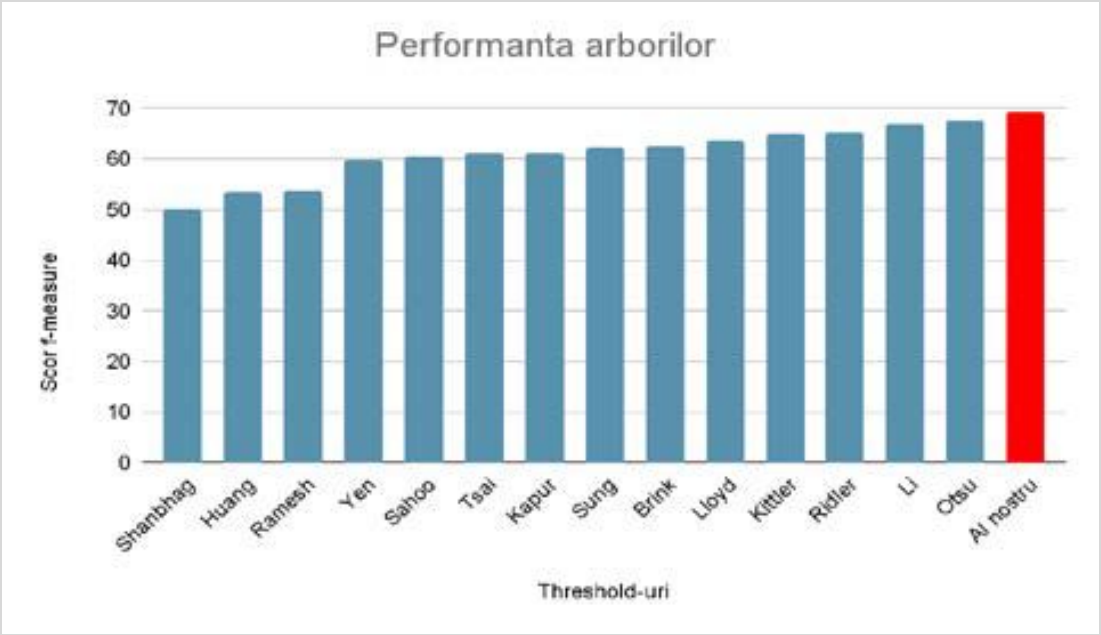
Comparare rezultate cu metrici performanță derivate din tresholduri (praguri de bianarizare) canonice

Li ~68%

Otsu ~67%

Shanbagh ~50%

EVALUARE REZULTATE INTERMEDIARE ALE SOLUȚIEI SOFTWARE



- Comparație performanțe pe arbori obținuți (în stânga)
- f-measure mediu de aproximativ 70%, într-un timp de rulare de aproximativ 1s
- Singurele praguri care se apropie de performanța arborelui sunt threshold-urile Otsu și Li (67-68% performanță), celelalte plasându-se între 50% și 64% performanță.

- Set de date mari → timp de rulare de 5 minute, dar compensat de threshold-ul bun obținut!

PENTRU MILESTONE 3

- Se termină implementarea, descrierea integrată în arhitectura software a binarizării locale

- Rezultă metricile complete de performanță

- Code styling (ex. eliminarea instrucțiuniilor repetitive dpdv logic care sunt scrise pe mai multe linii decât ar fi cazul - comprimare în one-liners unde este cazul - în functions.py)

- Further reduction? La binarizarea globală, numărul de operații de căutare nod potrivit de inserat, a fost cu jumătate mai puțin!