Raport Tehnic - Echipa 4

Buzdugan Mihaela 341 C2
Dan Gabriela 341 C5
Gheorghe Adrian Valentin 341 C4
Negoescu Bianca Maria 341C5
Radu Sabina 341 C4

1. Introducere în domeniu

În general, documentele scanate, inclusiv text, desene și regiuni grafice pot fi considerate documente de tip mixt. În multe aplicații practice, avem nevoie să recunoaștem sau să îmbunătățim în principal conținutul text al documentelor. În astfel de cazuri, este de preferat ca documentele să fie convertite într-o formă binară. Astfel binarizarea documentelor este primul pas în majoritatea sistemelor de analiză a documentelor. Scopul binarizării documentului este de a converti inputul dat, fie el document în tonuri de gri sau color, într-o reprezentare pe două niveluri. Această reprezentare este convenabilă deoarece majoritatea documentelor care apar în practică au o singură culoare pentru text (de exemplu, negru) și o culoare diferită (de exemplu, alb) pentru fundal.

Thresholding-ul devine un instrument simplu, dar eficient de a separa obiectele de fundal. Rezultatul operației de thresholding este o imagine binară a cărei stare va indica obiectele din prim-plan, adică textul tipărit, o legendă etc., în timp ce starea complementară va corespunde fundalului. În funcție de aplicație, prim-planul poate fi reprezentat de nivelul de gri 0, adică negru pentru text, și 1 pentru fundalul hârtiei, adică alb.

Binarizarea imaginii este procesul de separare a valorilor pixelilor în două grupuri, alb ca fundal și negru ca prim plan. Thresholding-ul joacă un rol important în binarizarea imaginilor. Thresholding-ul poate fi clasificat în Thresholding global și Thresholding local.

În imagini cu distribuție uniformă a contrastului de fundal și prim-plan precum imaginile cu documente, Thresholding-ul global este mai potrivit.
Metode de binarizare globală, cum ar fi cel propus de Otsu încearcă să găsească o singură valoare de prag pentru întregul document. Apoi fiecare pixel este atribuit
fie în prim-plan-ul imaginii, fie în fundalul acesteia pe baza valorii sale de gri. Metodele de binarizare globală sunt foarte rapide și dau rezultate bune pentru documentele scanate tipice. Pentru mulți ani, binarizarea documentelor în tonuri de gri s-a bazat pe algoritmii de prag global.

În imaginile degradate, unde există zgomot de fundal considerabil sau există variație de contrast și iluminare, sunt mulți pixeli care nu pot fi ușor clasificați ca pixeli de prim plan sau pixeli de fundal. În astfel de cazuri, binarizarea cu prag local este mai potrivită. Aceste tehnici estimează un prag diferit pentru fiecare pixel conform informațiilor în tonuri de gri ale pixelilor vecini. Tehnicile lui Bernsen, Chow și Kaneko, Eikvil, Mardia și Hainsworth, Niblack, Taxt, Yanowitz și Bruckstein și Sauvola și Pietikainen aparțin acestei categorii. Tehnici hibride: O'Gorman și Liu și Li, care combină informații despre pragurile globale și locale aparțin altei categorii.

2. Arhitectură

In cadrul proiectului se identifica doua tipuri de arhitecturi folosite.

1) Arhitectura ierarhica deschisa
2) Arhitectura centrata pe fluxul datelor: pipeline secvential

Structura proiectului este urmatoarea:

In primul rand, modulul care genereaza executia programului este application.py. Acesta asteapta input din partea utilizatorului pentru a declansa executia unuia dintre modulele: globalImplementation.py sau localImplemenation.py, in functie de inputul introdus. Asadar, application.py are acces la toate modulele aflate sub el, aspect specific arhitecturii ierarhice deschise.

In al doilea rand, proiectul este compus din doua parti principale: partea de implementare globala si partea de implementare locala. Fluxul datelor in cadrul acestor componente este secvential. Implementarile noastre se bazeaza pe etape succesive de transformare a datelor de intrare pentru a produce rezultate intermediare ce vor fi folosite in continuare pentru a ajunge la rezultate finale. Ambele implementari mentionate folosesc aceeasi logica de prelucrare a datelor:

- prima etapa, train, genereaza primul set de rezultate (filteredTrainResults.json / localTestResults.json).
- a doua etapa, validation, foloseste ca input de prelucare rezultatele primei etape si genereaza alt set de rezultate ((filteredValidationResults.json / localValidationResults.json).
- a treia si ultima etapa, test, foloseste ca input de prelucrare rezultatele celei de-a doua etape si genereaza ultimul set de rezultate (bestResults.json / localTestResults.json).

Asadar, se identifica arhitectura de pipeline secvential, intrucat execuția unei etape de transformare a datelor se poate iniția numai după ce s-a terminat executia etapei precedente.

3. Tehnologii folosite

Pentru pregătirea mediului de dezvoltare al proiectului software creat cu ajutorul limbajului de programare Python, ne-am folosit de IDE-ul Visual Studio Code. IDE-ul a fost integrat cu GitHub, cu ajutorul extensiei Codespaces, facilitând preluarea și updatarea ultimei versiuni a codului. De asemenea, am utilizat biblioteci ajutătoare precum json (folosim funcții implementate pentru importul și exportul de date în și din fișiere json), time (măsurarea timpului de execuție), csv (parsarea fișierelor de input de tip csv), os (preluarea tuturor fișierelor de input dintr-un folder), dar și itertools (funcția combinations() folosită pentru generarea tuturor combinațiilor de operatori: “+”, ”-”, “*”, “/”).

Specificațiile sistemului

- hardware: CPU: Intel i5-8250U, 1.60GHz base, overclock 1.80GHz, RAM 8GB, 4 Cores.
- software: OS: Windows 10, x86-64

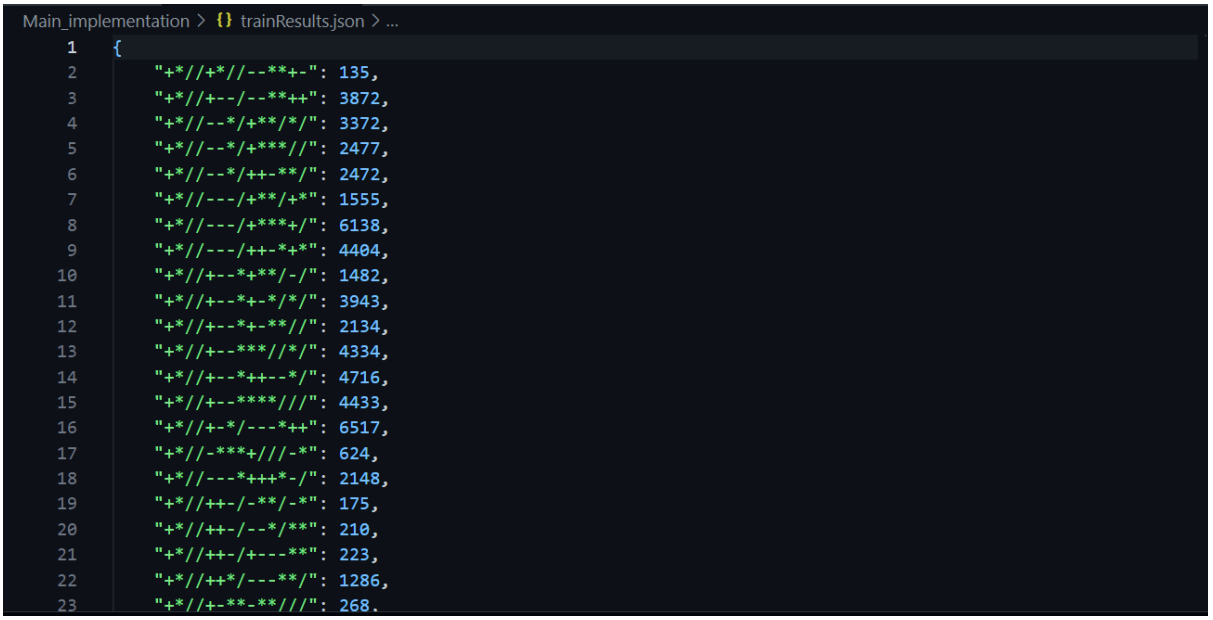
4. Implementarea soluției

4.1 Etapa Globală

Pentru prima parte a proiectului, și anume binarizarea globală, am ales să luăm input-urile ca termeni ai unei funcții. Astfel, am ales o metodă de rezolvare de tip greedy: în primul rând am generat toate combinațiile posibile de 14 operatori, rezultând aproximativ 430.000 de combinații, dintre care și duplicate. După ce am eliminat tot ce era în plus și am pus condiția ca numărul de “+” să fie egal cu numărul de “-” și numărul de “*” să fie egal cu numărul de “\”, pentru a evita cat mai mult posibil apariția unor valori mai mari decat 1 și mai mici decat 0, valori ce nu ne-ar fi de folos. Combinațiile rămase le-am salvat apoi într-o listă pentru a le putea folosi mai ușor.

Totodată, am folosit un dicționar în care am ținut evidența numărului de fișiere pentru care fiecare combinație a avut o valoare F-measure de cel puțin 0.8 (80%). Astfel, am luat fiecare fișier de input și am calculat rezultatul pentru fiecare combinație de operatori și, dacă acesta se apropie de Tresholding-ul ideal, incrementăm frecvența corespunzătoare acesteia din dicționar.

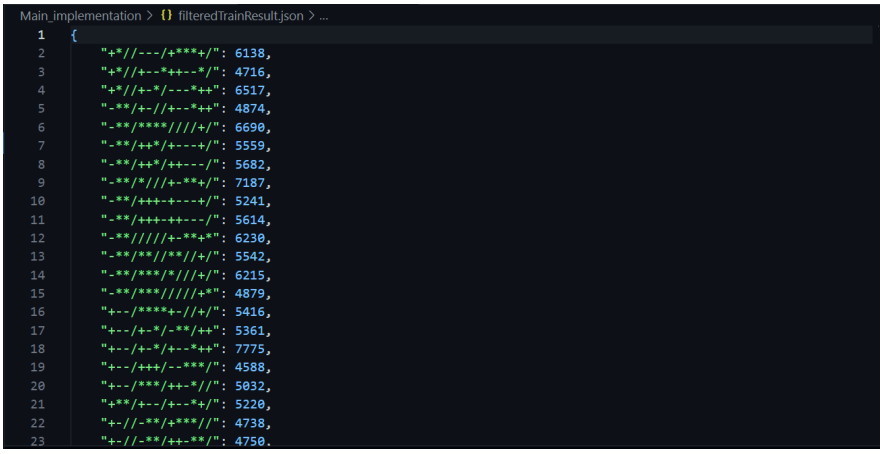
Am împărțit fișierele de input în 3 categorii: 70% din ele pentru partea de train, 25% pentru validation și ultimele 5% pentru partea de testing. Astfel, am rulat procesul descris mai sus pentru partea de training, iar dicționarul rezultat l-am exportat într-un fișier .json.



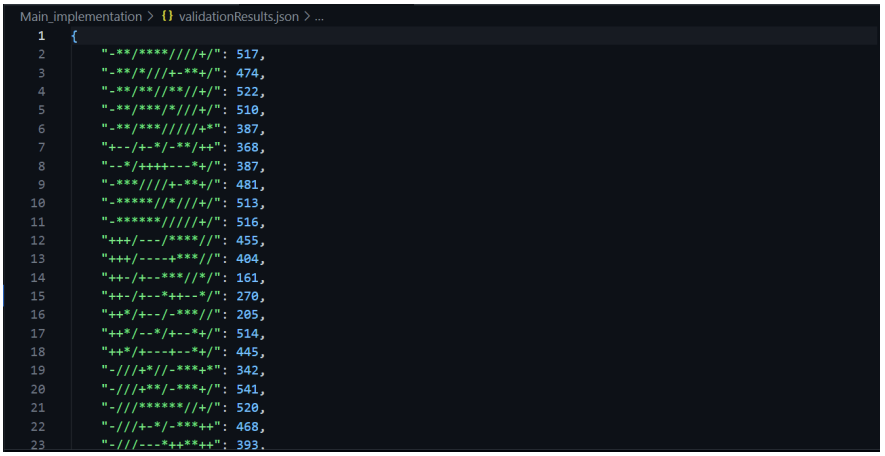
Pentru a trece la următoarea etapă, am ales doar combinațiile care erau cel puțin egale cu 75% din maximul obținut. Același procedeu l-am aplicat și pentru următoarele două

etape, însă scopul a fost ca după fiecare etapă să eliminăm cât mai multe valori și să rămânem doar cu cele mai relevante. Astfel, am obtinut următoarele rezultate parțiale:

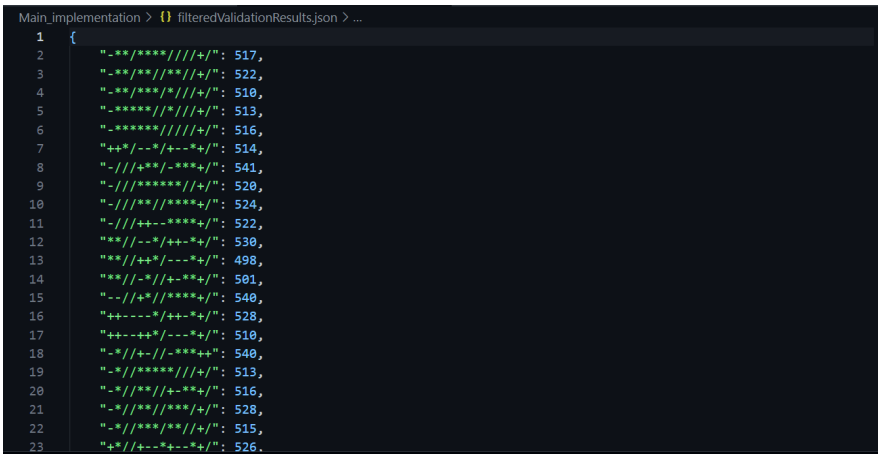
Rezultatele înainte de etapa validation:



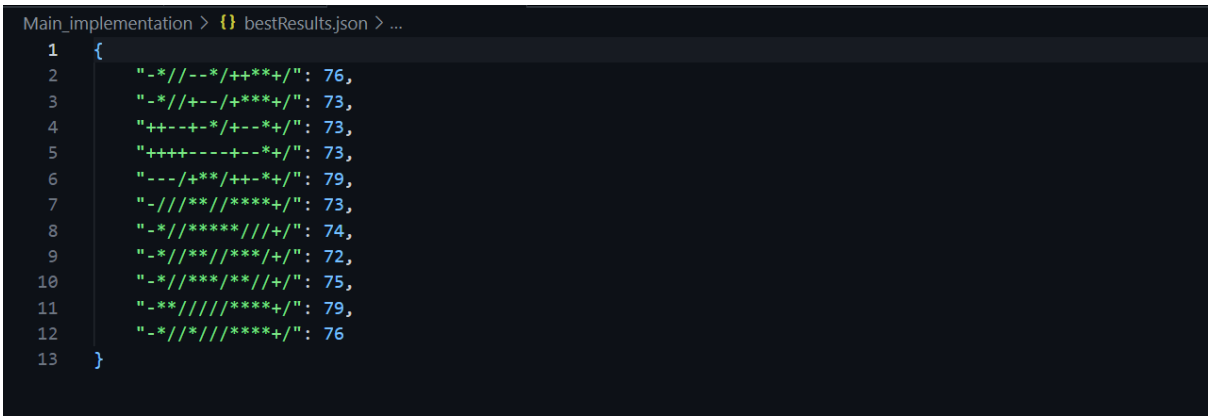
Rezultatele după etapa validation:



Rezultatele înainte de etapa test:



Rezultatele finale ale etapei globale



Astfel, am reușit să ajungem la 11 posibile rezultate finale.

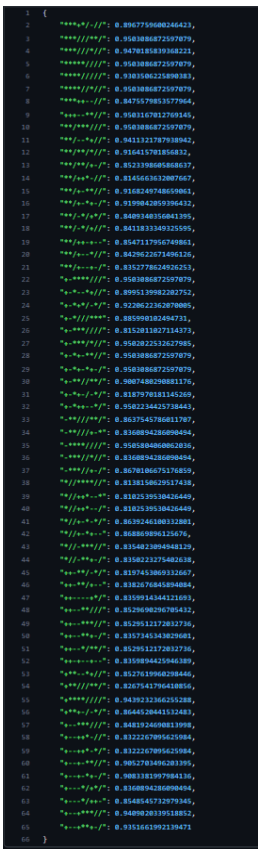
4.2 Etapa Locală

Pentru partea de binarizare locală am reușit să punem baza implementării, ideea fiind bazată tot pe o strategie de tip greedy, similară cu cea de la etapa precedentă. Deși au fost mai puțini operatori, principala problemă pe care am intampinat-o a fost timpul de execuție al programului, îngreunat de numărul imens de linii din fișierele de input.

Testele aferente partii locala sunt la randul lor impartite, precum cele la etapa globala in 3 parti: train, validation si test.

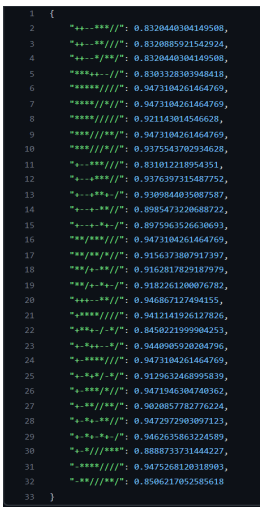
Implementarea abordată are ca prima etapa realizarea combinațiilor de 9 operatori. Apoi, pentru fiecare dintre fișierele selectate pentru etapa de train, vom afla care dintre combinatiile propuse anterior are cel mai mare procentaj de match. Acest procentaj l-am calculat in felul urmator: am luat fiecare combinatie dintre cele propuse si le-am aplicat pe fiecare pixel al imaginii (fiecare linie a fisierului). In urma aplicarii combinatiei pe un pixel, acesta ne-a rezultat ca apartine clasei 0 sau 1. Verificam daca raspunsul primit este acelasi cu cel pe care il asteptam, iar daca se potriveste il marcam ca reusit. La finalul fisierului calculam procentajul de match al combinatiei curente impartind numarul de match-uri ale claselor la numarul total de pixeli. Dupa ce am terminat cu o combinatie, vom lua urmatoarea combinatie si-i vom calcula si acesteia in mod identic procentajul de match. La finalul aflarii procentajelor de match al tuturor combinatiilor pe primul fisier, vom trece la urmatorul si vom recalcula noile procentaje. Toate aceste procentaje sunt contorizate intr-un dictionar in care calculam in mod constant procentajul de match al fiecarei combinatii, bazat pe procentajele obtinute pe fisierele anterioare.

Din etapa de train am selectat doar procentajele de match cel putin egale cu 80%.



Pentru etapa de validation nu vom mai folosi toate combinatiile posibile, ci doar cele care au fost selectate la etapa anterioara, deoarece dorim sa le validam corectitudinea. In cadrul etapei de validation procedam in fel identic precum in etapa de train pentru a calcula procentajele pe noile fisiere selectate pentru validation.

Din etapa de validation am selectat doar procentajele de match cel putin egale cu 83%.



Pentru etapa de test vom folosi doar combinatiile selectate in etapa de validation pentru a testa corectitudinea lor. De asemenea, in aceasta etapa vom calcula in mod identic procentajele pe fisierele selectate pentru test, iar dintre aceste procentaje vom selecta doar

cele cu cel putin 85%. Combinatiile selectate dupa etapa de test sunt considerate de proiectul nostru cele mai potrivite.

Testarea soluției

Aceasta solutie a fost constat testata, in timpul implementarii. Fiecare operatie pe care am realizat-o, am calculat si manual care ar trebui sa fie rezultatele pentru a ne asigura ca mergem pe o cale corecta. Dupa implementarea functiilor principale, am testat pe un numar mai mic de fisiere, pentru a putea tine evidenta rezultatelor si corectitudinii acestor.

La finalul implementarii, fiecare membru al echipei a rulat solutia prezentata in proiect pe fisiere diferite, alese random si a verificat daca fiecare etapa si fiecare functie returneaza rezultatele corecte.

De asemenea, am verificat si in sens invers, adica am verificat daca acele combinatii selectate se potrivesc cat mai bine pe cate un fisier ales aleator dintre cele pe care am rulat solutia.

Din pacate nu am realizat si o testare automata sau unitara. Singura testare pe care am investit-o in proiectul nostru este testarea manuala a intregii echipe.

5. Evaluarea soluției

5.1 Etapa globala

a) Timp de executie

- Etapa *train* contine 13496 fisiere de tip CSV care au fost prelucrate intr-un timp de 441.6 secunde => aproximativ 7.36 minute.
- Etapa *validation* contine 4541 fisiere de tip CSV care au fost prelucrate intr-un timp de 8.22 secunde.

- Etapa *test* contine 1083 fisiere de tip CSV care au fost prelucrate intr-un timp de 0.34 secunde.

=> un timp total de executie pentru etapa globala de cca 450 secunde => 7.5 minute pentru un input de 19120 fisiere csv.

```
Alege implementarea dorita: [global, local]global
Se ruleaza implementarea globala:
Se ruleaza etapa de training:
11886
--- 441.59763193130493 timp de executie ---
==============================
Se ruleaza etapa de validation:
--- 8.224333763122559 timp de executie ---
==============================
Se ruleaza etapa de test:
--- 0.3403291702270508 timp de executie ---
==============================
--- 450.16405296325684 timp de executie total ---
```

b) <u>Memorie folosita</u>

Prima coloana- CPU
A doua coloana - Memory

c) <u>Evaluarea rezultatelor</u>

Cele mai bune combinatii si procentul de potrivire pentru inputul de 19120 fisiere csv:

- "++--+-*/+--*+/" => 50 %
- "++++----+--*+/" => 45%
- "-*//+--/+***+/" => 43%
- "-*//*///****+/" => 42%
- "-**/////****+/" => 41%
- "---/+**/++-*+/" => 40%
- "-*//*****///+/" => 39%
- "-///**//****+/" => 38%
- "-*//***/**//+/" => 35%
- "-*//**//***/+/" => 32%
- "-*//--*/++**+/" => 30%

5.2 Etapa locala

a) <u>Timp de execuție</u>

```
Se ruleaza implementarea locala:
Se ruleaza etapa de training:
^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C^[[C--- 1730.6610615253448 t
imp de executie ---
==============================
Se ruleaza etapa de validation:
--- 3140.4769308567047 timp de executie ---
==============================
Se ruleaza etapa de test:
--- 2554.0018944740295 timp de executie ---
==============================
S-a ajuns la urmatoarele solutii:
('*', '*', '*', '/', '/', '/', '*', '*', '/')
('*', '*', '*', '/', '/', '/', '*', '/', '/')
('*', '*', '*', '*', '*', '/', '/', '/', '/')
('*', '*', '*', '*', '/', '/', '/', '/', '/')
('*', '*', '*', '*', '/', '/', '*', '/', '/')
('+', '+', '+', '-', '-', '*', '*', '/', '/')
('*', '*', '/', '*', '*', '*', '/', '/', '/')
('*', '*', '/', '*', '*', '/', '*', '/', '/')
('*', '*', '/', '+', '-', '*', '*', '/', '/')
('*', '*', '/', '+', '-', '*', '+', '-', '/')
('+', '-', '*', '*', '*', '*', '/', '/', '/')
('+', '-', '*', '+', '*', '/', '-', '*', '/')
('+', '-', '*', '/', '/', '/', '*', '*', '*')
('+', '-', '*', '*', '*', '/', '*', '/', '/')
('+', '-', '*', '+', '-', '*', '*', '/', '/')
('+', '-', '*', '+', '-', '*', '+', '-', '/')
('+', '-', '*', '*', '/', '/', '*', '*', '/')
('+', '-', '*', '+', '+', '-', '-', '*', '/')
('-', '*', '*', '*', '*', '/', '/', '/', '/')
('+', '*', '*', '*', '*', '/', '/', '/', '/')
('+', '-', '-', '+', '-', '*', '*', '/', '/')
('+', '-', '-', '+', '-', '*', '+', '-', '/')
('+', '-', '-', '+', '*', '*', '*', '/', '/')
('+', '-', '-', '+', '*', '*', '+', '-', '/')
--- 7425.141303062439 timp de executie total ---
```

- Etapa *train* contine 950 fisiere de tip CSV care au fost prelucrate intr-un timp de 1730.666 secunde => aproximativ 29 minute.
- Etapa *validation* contine 197 fisiere de tip CSV care au fost prelucrate intr-un timp de 3140 secunde => aproximativ 53 minute.
- Etapa *test* contine 48 fisiere de tip CSV care au fost prelucrate intr-un timp de 2554 secunde => aproximativ 43 minute.

=> un timp total de executie pentru etapa globala de cca 7423 secunde => 123 minute pentru un input de 1195 fisiere csv.

b) <u>Memorie folosită</u>

Name	Status	CPU	Memory	Disk	Network
python3.10		13.2%	552.7 MB	0 MB/s	0 Mbps

c) Evaluarea rezultatelor

Cele mai bune combinatii si procentul de potrivire pentru inputul de 1195 fisiere csv:

- "*****////": 93,87%
- "****//*//": 93,87%
- "****/////": 91,4%
- "***///**/": 93,87%
- "***///*//": 92,95%
- "+--+***//": 92,96%
- "+--+**+-/": 92,34%
- "**/***///": 93,87%
- "**/**/*//": 90,9%
- "**/+-**//": 90,84%
- "**/+-*+-/": 91%
- "+++--**//": 93,83%
- "+****////": 93,29%
- "+-*++--*/": 93,35%
- "+-****///": 93,87%
- "+-***/*//": 93,86%
- "+-*+-**//": 93,87%
- "+-*+-*+-/": 93,77%
- "-****////": 93,89%