Cercetare

Optical Character Recognition. Aplicări ale OCR. Sudoku Puzzle Recognition.

A realizat: Beșleaga Sorin

Consultant științific: Miron Raisa

Anul 2022

Cuprins:

I. Introducere

[1.1 Introducere](#Introducere)...............................................................................................................3

[1.2 Scopul cercetării](#Scopul).......................................................................................................3

[1.3 Structura proiectului](#Structura).................................................................................................4

II. Setarea mediului (environment)

[2.1 Mediul de dezvoltare (IDE)](#Mediul)......................................................................................7

[2.2 Necesarul](#Necesarul).................................................................................................................7

III. Partea practică

[3.1 Descriera programului](#Programul)...........................................................................................12

[3.2 Compilare](#Compilare)..............................................................................................................19

IV. Rezultatul

[4.1 Output](#Output)....................................................................................................................21

V. Concluzie

[5.1 Probleme](#Probleme)................................................................................................................27

[5.2 Dezvoltarea proiectului](#Dezproiect)..........................................................................................29

[5.3 Concluzii](#Concluzii)................................................................................................................29

[Bibliografie](#Bibliografie)............................................................................................................................30

I. Introducere

1.1 Introducere

Ideea acestei cercetări este introducerea atât a mea, cât și a celor care vor citi acest document, în principiile de funcționare a procesării imaginei, a distincției anumitor caractere din imagine prin aplicarea unor metode a Inteligenței Artificiale (IA) și de rezolvarea anumitor probleme, dar și aprofundarea în problematica acestui domeniu. Nu am urmat scopul descoperirii, invenării a noi algoritme(principii) sau tehnologii originale și unice, ci doar cel de propria informare și pentru ilustrarea algoritmelor ce stau în spatele prelucrării și procesării imaginei pentru a se aplica IA.

Ce reprezintă Optical Character Recognition (OCR)?

Termenul tradus ca „Recunoașterea optică a caracterelor” reprezintă translatarea mecanică sau electroincă a imaginilor cu scris de mână, tipărit sau printat în text editabil.

OCR este un domeniu de cercetare în recunoașterea modelelor, IA și vederea mecanică. Programele inițiale necesită învățarea caracterelor (exemple ale fiecărui caracter) pentru identificarea unui font specific. În esență, este un domeniu amplu și poate fi utilizat în soluționarea a numeroaselor probleme. Cel mai des este folosit pentru identificarea scrisului uman și procesarea lui în formă de text, mai usoare fiind problemele cu fonturile calculatorului, deoarece sunt prezente mai multe exemple și este mai ușoară pregatirea informației pentru antrenarea modelului, dar și faptul că este invariabil, adica aceeași cifră mereu va fi scrisă la fel, problema principală fiind clasificarea fontului. Deci, OCR crează o punte de trecere între lumea observabilă și cea digitală. Chiar dacă calculatorul modern poate efectua în câteva fracțiuni ale secundei ceea ce omului i-ar lua minute sau chiar ore (calcule, modelare, ilustrații), el nu poate face ceea ce omului îi ia chiar mai putin timp (recunoasterea obiectelor, fețelor, diagnosticul bolilor). De aceea este implicată Inteligența Artificială care prin principiile sale de funcționare oferă posibilitate de apropiere spre modelul uman.

1.2 Scopul cercetării

Cum am reușit deja sa enunț, OCR este un domeniul destul de amplu și pentru cercetarea lui este nevoie de anumite proiecte concrete, adică este nevoie de urmărit toate etapele de lucru cu o astfel de tehnologie în cadrul unei aplicări. Pentru aceasta propun spre considerare scopul acestei lucrări și anume cel de rezolvare a unor exemple de sudoku real.

**Descrierea problemei:**

Se dă un set anumit de imagini, în care sunt reprezentate diferite exemple de sudoku. Imaginile pot fi distorsionate, cu perspectivă modificată, cu zgomote, umbre, incomplete și de complexitate diferită, dar pot fi și clare cu o calitate înaltă a imaginii (totul pentru a se vedea eficiența programei și contrastul între soluții). Ele sunt procesate de program și la final acel program trebuie sa îl rezolve, afisând imaginea finală cu sudoku inițial rezolvat. Același principiu trebuie sa funcționeze și cu videoul. În esență orice videou este schimbarea imaginilor la o anumită frecvență, deci programul ar trebui sa prelucreze aceste imagini în timp real și să afișeze soluția pe videou, programul pentru videou având doar mici modificări la nivel de cod.

**Cum se poate soluționa problema descrisă:**

Pentru a rezolva problema putem importa imaginea de pe o anumită locație sau în cazul videoului de a o citi din fluxul videocamerii, apoi o procesăm și îi modificăm perspectiva, astfel încat ea să fie clară și pentru a se putea distinge cifrele pentru model. Apoi, din imaginea întreagă, procesată, o împărțim în mai multe imagini care cuprind regiunile pătrate cu cifre sau fără. Având în prealabil un model deja antrenat, încărcăm acele imagini și el efectuează predicția. Cu cifrele selectate, le încărcăm în funcție care va rezolva sudoku și el dând soluția, vom afișa doar cifrele care lipsesc din imaginea originală. Cam așa va funcționa algoritmul.

**Concluzie:**

Programa nu are o complexitate înaltă, fiind folosite multe funcții din biblioteci specializate, predefinite, care ușurează semnificativ rezolvarea unor astfel de probleme, și face codul programei mult mai citeț și usor de analizat. Principalele dificultăți fiind întâmpinate în timpul procesării imaginei și recunoașterea cifrelor, deoacere modelul nu este perfect, caci pentru perfecționarea lui este nevoie de o putere de calcul înaltă și cel mai important – timp. Cu toate acestea, scopul lucrării este realizabil și voi descrie detaliat pașii de rezolvare.

1.3 Structura proiectului

Folder-ul <AI\_sudoku\_solver> conține 3 subfoldere: <sudoku solve cercetare>, <sudoku solve sources> și <docs>. Astfel, vom descrie componența fiecărui folder.

* <sudoku solve cercetare> :
  + - folderul <\_\_pychache\_\_> – conține fișierele .pyc create de interpretatorul Python cand fișierele .py sunt importate. Ele conțin „bytecodul compilat” a modulelor/programelor importate asa încât să nu fie nevoie de recompilarea lor. Deci ele sunt transormate în codul cu care calculatorul poate comunica direct.
    - <images> – conține toate imaginile exemple cu sudoku care necesită rezolvate de programă, acest folder poate fi permanent modificat (adăugarea/stergerea anumitor imagini) de aceea nu se va afișa pe vizualizare componența folderului (imaginile ce sunt prezente, putând fi adaugate sau sterse de oricine)
    - Fișierul dugit\_model.h5 – modelul antrenat care va fi folosit pentru recunoașterea cifrelor din imagini.
    - sudoku.py – fișierul principal care va rula programa de rezolvare a sudoku.
    - utils.py – fișierul cu toate funcțiile ajutătoare care se folosesc în fișierul principal, sunt puse aici pentru a fi mai ușor de citit fișierul care rulează programa, oferind și o descriere mai clară, în afara contextului, ce realizează fiecare funcție.
    - sudoku\_solver.py – conține funcția principală de rezolvare a sudoku, astfel algoritmul simplu recursiv pentru rezolvarea oricărui sudoku și alte funcții ajutătoare care sunt necesare pentru găsirea soluției.
  + <sudoku solve sources> :
    - * + <sudoku solve raw> – conține aceleași fișiere ca în folderul <sudoku solve cercetare>, însă fară comentarii explicative a codului în fiecare fișier, pentru a fi mai citeț codul și mai curat.
        + <sudoku solve video> :

digit\_model.h5 – același model antrenat ca și în alte foldere din acest proiect, va fi folosit pentru recunoașterea cifrelor din imagini.

v\_sudoku.py – fișierul sursă, folosit pentru rularea programei care va detecta și va rezolva din videou sudoku, va lucra prin intermediul camerei web, tot algoritmul realizându-se în timp real.

utils.py – la fel ca și in fișierul cu cercetare <sudoku solve cercetare> sau <sudoku solve raw> reprezintă fișierul ajutător cu toate funcțiile-cheie necesare pentru a se realiza programul.

op\_sudoku\_solver.py – conține funcția principală de rezolvare a sudoku, însă spre deosebire de algoritmul recursiv prezentat în fișierul cu aceeași denumire din folderul <sudoku solve cercetare> aici soluția este optimizată și adaptată pentru rezolvarea mai rapidă în condițiile în care viteza este esențială în fluxul de imagini, adică video. Chiar dacă diferențele de performanță a alogritmelor nu este semnificativă (de ordinul secundelor sau milisecundelor), ele oricum se observă în timpul video, mai ales când resursele calculatorului sunt limitate ca în cazul meu.

<\_\_pychache\_\_> – la fel ca și în celelalte foldere conține fișierele binare compilate pentru a fi ulterior folosite direct de programele ce importa modulele la baza căror sunt interpretate pentru a se compila mai rapid codul care folosește acele dependențe (module)

* + <docs> – folderul conține toate documentele necesare pentru informare cu proiectul și pentru gestionare a programei.

După ce a fost descrisă toată arhitectura proiectului pentru a fi ulterior mai înteles unele nuanțe în cod legate de locația anumitor fișiere, vă prezint vizualizarea unix a structurii descrise anterior (unix file tree):

comanda introdusă in cmd: tree /f (după ce am selectat drumul complet spre folderul de care am nevoie, în cazul dat spre <AI\_sudoku\_solver>)

AI\_sudoku\_solver

├───docs

│ Cercetare\_OCR.docx

│ Sinteza\_OCR.docx

│

├───sudoku solve cercetare

│ │ digit\_model.h5

│ │ sudoku.py

│ │ sudoku\_solver.py

│ │ test.py

│ │ utils.py

│ │

│ ├───images

│ │

│ └───\_\_pycache\_\_

│ sudoku\_solver.cpython-39.pyc

│ utils.cpython-39.pyc

│

└───sudoku solve sources

├───sudoku solve raw

│ │ digit\_model.h5

│ │ sudoku.py

│ │ sudoku\_solver.py

│ │ utils.py

│ │

│ ├───images

│ │

│ └───\_\_pycache\_\_

│ sudoku\_solver.cpython-39.pyc

│ utils.cpython-39.pyc

│

└───sudoku solve video

│ digit\_model.h5

│ v\_sudoku.py

│ op\_sudoku\_solver.py

│ utils.py

│

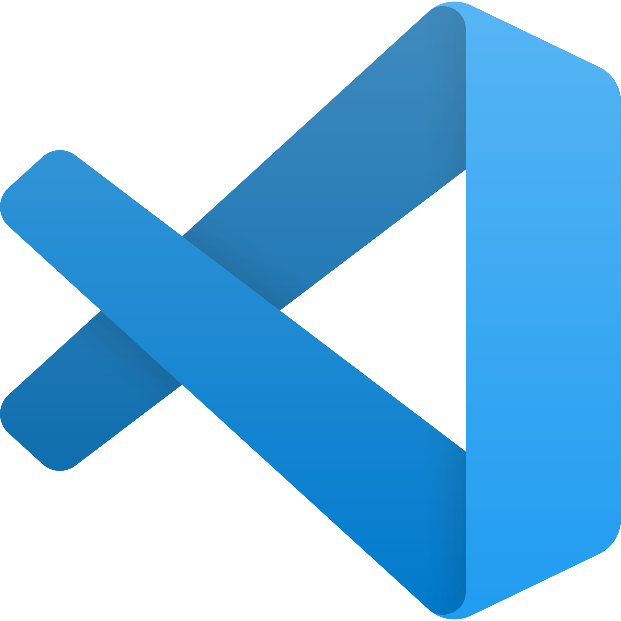
└───\_\_pycache\_\_

op\_sudoku\_solver.cpython-39.pyc

utils.cpython-39.pyc

II. Setarea mediului (environment)

2.1 Mediul de dezvoltare (IDE)

 Mediul de dezvoltare sau IDE (Integrated Development environment) este un software (programă) bogată în diverse unelte și particularități care sunt necesare pentru redactarea comodă, eficientă și rapidă a codului, adică a oricarei alte programe (software). Sunt o mulțime de IDE-uri, cu avantajele și dezavantajele sale. De asemenea, un rol important joaca și gusturile fiecărui în parte. Pentru realizarea acestui proiect și în practica zilnică, cand este vorba de redactarea unui cod în mai multe fișiere și în diverse limbaje, eu prefer să folosesc ***Visual Studio Code***.

În opinia personală el are un șir de avantaje:

* Ușurința în descărcare și setare (folosire)
* Poate fi complet personalizat (oferă o mulțime de setări pentru personalizare)
* Poate fi folosit pentru mai multe limbaje de programare, prin descărcarea anumitor suplimente
* Usor în folosire, deoarece este un IDE destul de ușor, este user-friendly și conține strict necesarul pentru o redactare eficientă
* Este destul de popular, și se poate mai ușor de găsit vreo eroare pe internet, legată de setarea compilatorului, etc.

Din neajunsul am observat doar greutăți în lucrul cu niște proiecte într-adevări mari, cu o mulțime de fișiere și dependențe, neajunsul anumitor unelte care sunt specifice anumitui limbaj de programare și greutăți la compilarea în ferestre aparte, nu în terminalul incorporat.

Astfel, pentru a modifica și folosi proiectul, este nevoie de importat folderul întreg cu proiectul în Visual Studio Code, prin File -> Add folder to Workspace… . Iar prin panela din stânga File explorer putem vizualiza și schimba fișierul de lucru, lansarea programei realizându-se pe butonul specific în partea dreaptă sus.

Pentru a se cunoaște mai detaliat cu această programă puteți accesa linkul: <https://code.visualstudio.com/docs/introvideos/basics> care este destinat începătorilor.

2.2 Necesarul

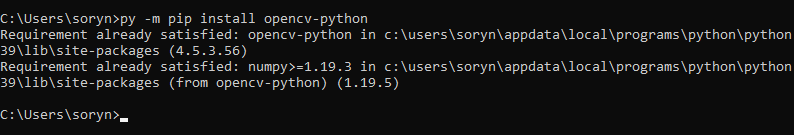
Pentru a lucra asupra proiectului și pentru a programa este nevoie de anumite dependențe precum: opencv, tensorflow, numpy etc. Instalarea și importanța lor va fi expusă mai detaliat în secvența ce urmează. Astfel, avantajul folosirii limbajului Python asupra altor limbaje (C++ care de asemenea susține toate aceste librării și are mecanisme prin care poate opera cu aceste dependențe la fel ca și Python sau chiar mai rapid, însă codul devine mult mai complicat) este instalarea bibliotecilor și importarea lor, anume sistemul de instalarea a librăriilor (numite packages), în opinia mea, este unul din cele mai comode, fiind factorul decisiv în alegerea limbajului.

Comanda generală pentru instalarea oricărui package în Python este: python -m pip install <package> sau py -m pip install <pachage>. Această comanda se introduce în CMD (Command Line Prompt) pe sistemul de operare Windows.

**Instalarea OpenCV:**

OpenCV (Vedere computerizată cu sursa deschisă) este o bibliotecă de funcții informatice specializată pe vedere computerizată în timpul real. Elaborată inițial de Intel, a fost dezvoltată ulterior de Willow Garage, apoi de Itseez (achiziționată mai târziu de Intel). Biblioteca este de tip multiplatformă și este gratuită pentru utilizare sub licența BSD pentru sursă deschisă. OpenCV suportă ansamblurile software de învățare profundă TensorFlow, Torch/PyTorch și Caffe.

Pentru instalarea bibliotecii OpenCV în Python, folosim comanda:

py -m pip install opencv-python

Deoarece la mine deja este instalată această bibliotecă, se afișează mesajul ca este îndeplinită comanda.

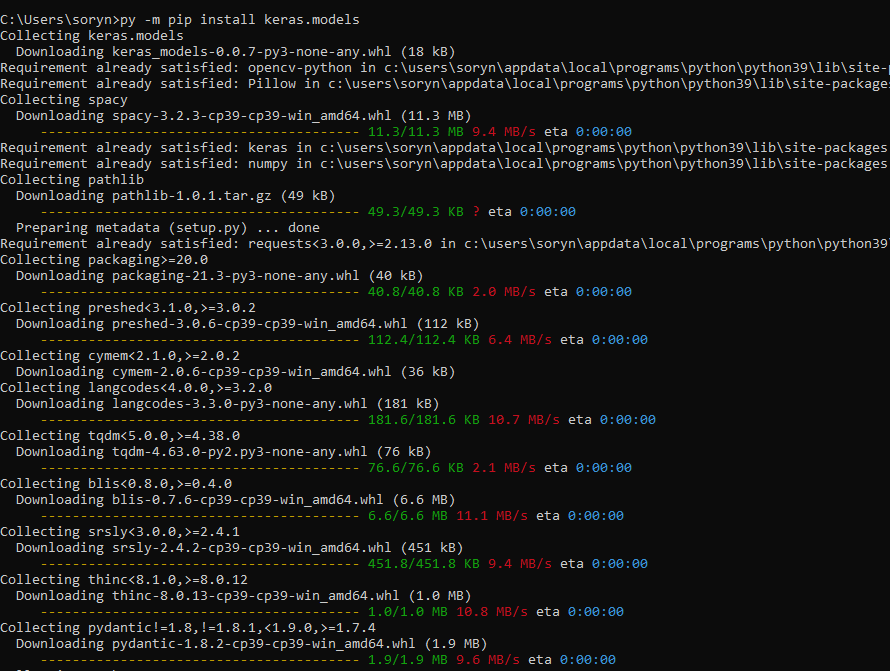
**Instalarea keras.models:**

Keras este o bibliotecă open-source care oferă o interfață Python pentru ANN (Artificial Neural Networks). Keras este implementată ca o interfață pentru biblioteca TensorFlow. Este făcută pentru diverse experimente cu rețele neuronale adânci (deep neural networks), focusându-se pe user-friendly, modularitate și extensibilitate. Este utilizată în Machine Learning și pentru dezvoltarea proiectelor IA.

Pentru instalarea keras.models (un modul din librărila Keras), folosim comanda:

py -m pip install keras.models

Mai jos este atașată o parte din rezultatul aplicării comandei:

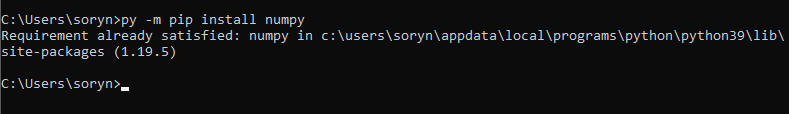


**Instalarea NumPy:**

NumPy este o bibliotecă pentru limbajul de programare Python, care adaugă suport pentru tablouri și matrice mari, multi-dimensionale, pe lângă colețiilor mari de funcții metematice pentru a opera cu aceste tablouri. NumPy este un proiect open-source și are o mulțime de contributori.

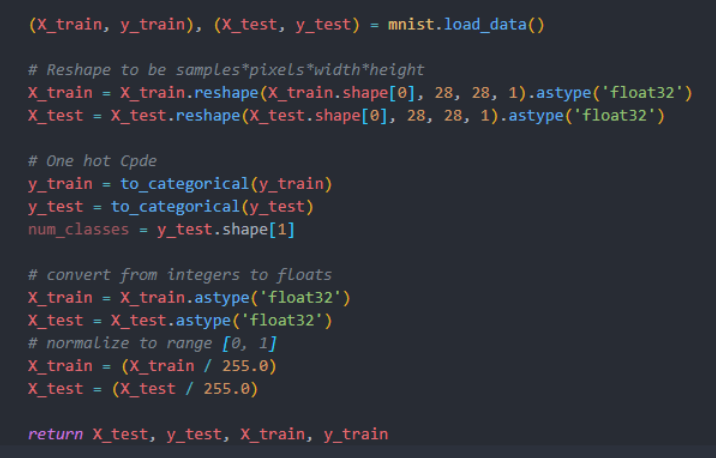
Pentru instalarea NumPy, folosim comanda:

py -m pip install numpy

\*Biblioteca este deja instalată pe calculatorul meu, de aceea nu se vede log-ul complet.

Următorul pas este încărcarea modelului. O remarcă importantă trebuie de facut asupra modelului de antrenare: El a fost preluat de pe [github](https://github.com/), deci a fost compilat și creat pe alt calculator. Cauza acestei decizii este insuficiența de resurse ale calculatorului personal și timpul îndelungat necesar pentru antrenare, care de obicei nu rezulta cu succes pe mașina mea locală, de acceea am optat pentru preluarea modelului, cu particularitățile de setarea lui.

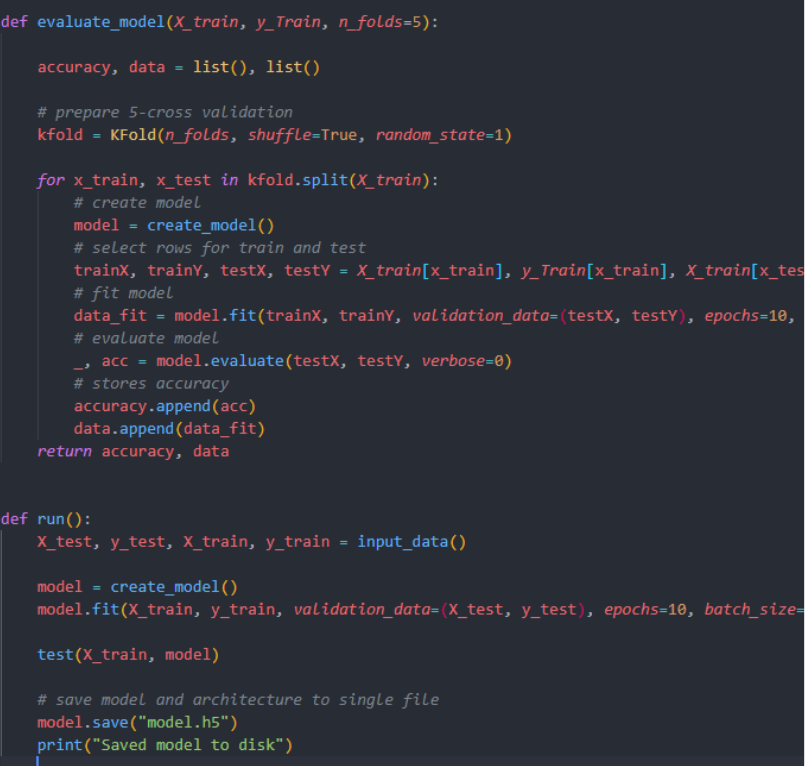
Pentru modelul de machine learning, a fost folosit baza de date MNIST.

Codul folosit pentru a incărca baza de date MNIST:

  
Baza de date MNIST (Modified National Institute of Standards and Technology) este o bază largă de date, care constă din diverse cifre scrise manual. Baza de date conține 60 000 imagini de antrenare și 10 000 imagini de testare. Jumate din setul de testare și antrenare a fost luat din datasetul de antrenare NIST, în timp ce altă jumate a setului de antrenare și testare a fost luată din datasetul de testare NIST.

Deci, în funcția create\_model prin load\_data a fost încărcată baza de date și s-a divizat in seturi de testare și antrenare. După au fost modificate într-un array de 28 pe 28 (reshape), căci baza de date are așa setări pentru exemplele de intrare și pentru efectuarea predicțiilor. Funcția to\_categorical transformă vectorul în matrice binară (numărul din vector fiind indexul unde se va plasa cifra 1 și restul va fi 0). shape[0] – reprezintă numarul de rânduri din array, shape[1] – numarul de coloane. Mai departe se convertesc din integer in float, după se trece de la intervalul de [0, 255] la [0, 1].

Apoi, a fost antrenat modelul și salvat într-un folder, așa încât să nu fie nevoie de încărcat de fiecare dată când va rula programa.

Codul folosit pentru antrenarea și salvarea modelului machine learning:

În funcția evaluate\_model se creează listele acurateței și a datelor, apoi prin KFold se realizează cross-validation (oferă indici de testare/antrenare pentru împărțirea datelor în seturi de testare/antrenare. Se împart în k fold-uri (pliuri) consecutive). Fiecare fold este apoi folosit o dată ca validare în timp ce restul k-1 fold-uri formează seturi de antrenare. Următorul pas este crearea modelului prin create\_model, selectarea rândurilor pentru testare și antrenare, prin fit modelul este antrenat (se antrenează prin procesarea modelului, adică se găsesc coeficienții ecuației specificate prin algoritm, putând apoi fi făcute predicțiile în problemele de clasificare). Prin evaluate, modelul este evaluat prin seturile de testare predestinate evaluării eficienței modelului rezultant, fiind returnată acuratețea lui. Ultima funcție run, îmbină restul funcțiilor deja descrise, și anume de creare a modelului, antrenarea, testarea și salvarea lui.

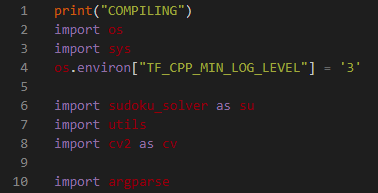
Astfel, modelul antrenat poate atinge acuratețea până la 100 %, aceasta fiind oarecum un rezultat relativ.

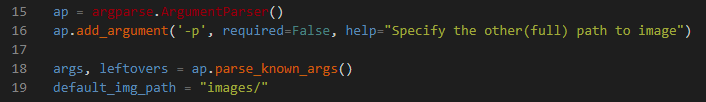
III. Partea practică

3.1 Descrierea programului

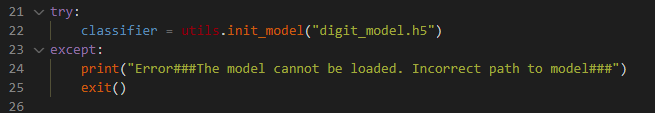
În această parte a cercetării voi descrie mai detaliat programul care corespunde alogritmului expus în partea cu scopul lucrării. În principal va fi descris fișierul care va rula întreaga programă și anume sudoku.py, dar vor mai fi devieri și spre alte fișiere care conțin funcțiile de care are nevoie sudoku.py pentru a funcționa, adică, paralel vor fi descrise și funcțiile din fișierele utils.py și sudoku\_solver.py.

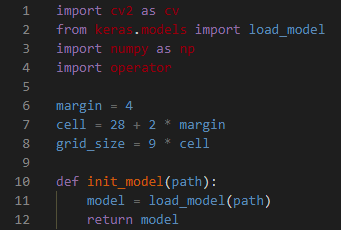
sudoku.py :

Prima linie denotă începului compilării și se importă principalele biblioteci, funcțiile căror vor fi folosite ulterior. Mai precis, os, biblioteca care interacționează cu sistemul de operare, sys – la fel interacționează la nivel local cu sistemul, sudoku\_solver, utils – fișierele care conțin funcțiile de rezolvare și procesare a imaginii, argparse – biblioteca care ajută cu procesarea argumentelor inserate în CMD.

În linia 4 se setează log-urile (informațiile adăugătoare în consolă despre procesul de compilare a funcțiilor din librărie) librăriei TensorFlow, adică se setează sa nu să se afișize acele loguri pentru a fi mai clare datele de ieșire.

Următoarele linii setează clasa de separare a argumentelor în consolă (CMD) (linia 15), apoi se adaugă un argument care nu este obligatoriu, prin care se oferă posibilitatea de a indica drumul comple spre imaginea aflată în altă locație (16) în caz că nu a fost adăugată în folderul implicit cu toate imaginile care se accesează dacă nu a fost specificat argumentul (19), iar în linia 18, se separă argumentele introduse (args) și ceea ce nu se atribuie la nici un argument.

În linia 22, se încearcă încărcarea modelului digit\_model.h5, prin funcția init\_model, în caz de eroare se va afișa mesajul corespunzător în linia 24 și se va ieși din program.

 utils :

În acest fișier sunt importate principalele librării de procesare a imaginii cv2, de lucru cu modelele - keras.models și de operare cu masive (tablouri) numpy, iar operator permite lucrul cu listele.

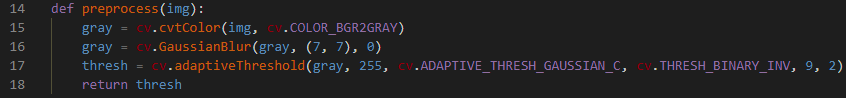
Liniile 6-8 sunt necesare pentru a seta offsetul (marginile), mărimile celulei în sudoku, și mărimea întregii grile. Aceste mărimi au fost folosite pentru antrenarea modelului de aceea ele pot fi considerate constante. 28 este mărimea necesară pentru importarea imaginii în model, care recunoaște imaginile 28x28 pixeli.

Funcția init\_model (10) încarcă modelul în program prin drumul spre locația lui path și returnează rezultatul la apelant.

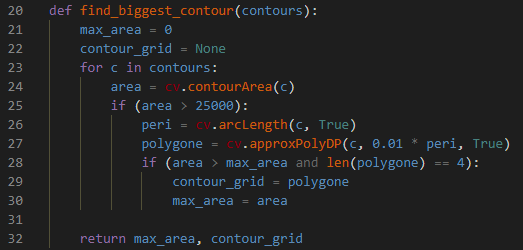
 sudoku.py :

Dacă a fost introdus argumentul pentru specificarea completă a drumului spre imagine atunci variabilei path i se atribuie acest drum, în caz contrar se va lua imaginea specificată din folderul implicit <images> (30). Linia 32 citește imaginea specificată și se verifică apoi dacă a fost citită corect.

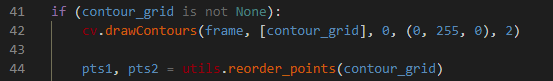
Linia 37 se preprocesează imaginea, se discretizează pentru a putea fi prelucrată în matrice de cifre, apoi se selectează toate contururile externe din imagine (38) pentru a putea fi găsită grila cu sudoku, în final se găsește cel mai mare pătrat (care ar trebui să reprezinte grila) și aria lui (39).

 utils.py :

Ca preprocesare se întelege, convertirea imaginei din RGB într-o imagine cu un singur canal (GRAY), apoi se aplică blurarea gaussiană, pentru a înlătura toate zgomotele mici de pe imagine și pentru a fi discretizată mai corect. Apoi se aplică tresholding-ul adaptiv prin care se realizează această discretizare unde regiunile primesc valori {0, 1} în dependentă de intensitatea pixelului.

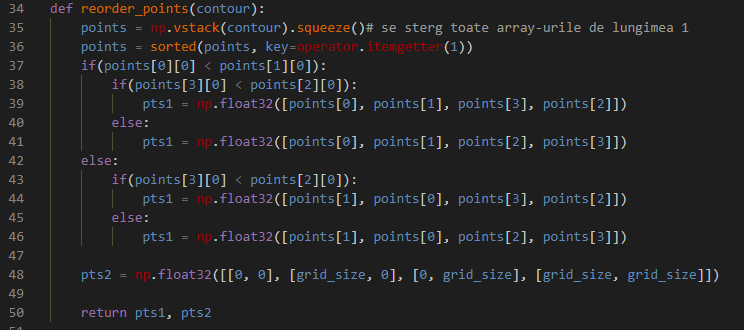
Această funție găseste cel mai mare contur din imagine și funcționează în felul următor: variabila contours este un vector de vectori deci se parcurge în buclă, i se calculează ara prin funcția predefinită din OpenCV apoi se aplică filtru care verifică dacă aria este suficient de mare pentru a fi vizată, apoi se i calculează perimetrul pentru a putea fi aplicată aproximarea DP care reduce poligonul la o formă mai simplă prin „taierea” partilor redundante. Pasul următor (28) este verificarea condiției să fie aria maximă din imagine și ca poligonul extras să fie dreptunghi. În acest fel se parcurge vectorul contours și se extrage conturul maxim.

sudoku.py :

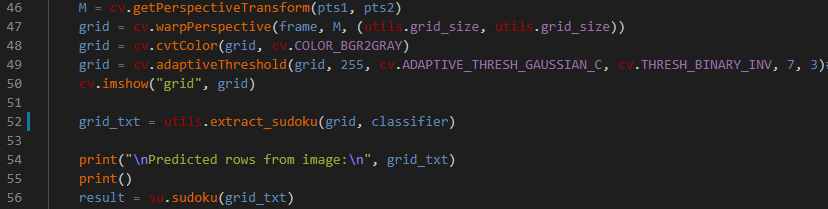
În acestă secvență pe imaginea originală se desenează conturul (42) selectat cu culoarea verde și grosimea 2. Apoi se selectează punctele pentru perspectivă, ele sunt rearanjate pentru a se respecta o anumită ordine (44). pts1 – reprezintă coordonatele punctelor rearanjate din imaginea originală, pts2 – punctele folosite pentru perspectivă, adică pentru noua imagine.

utils.py :

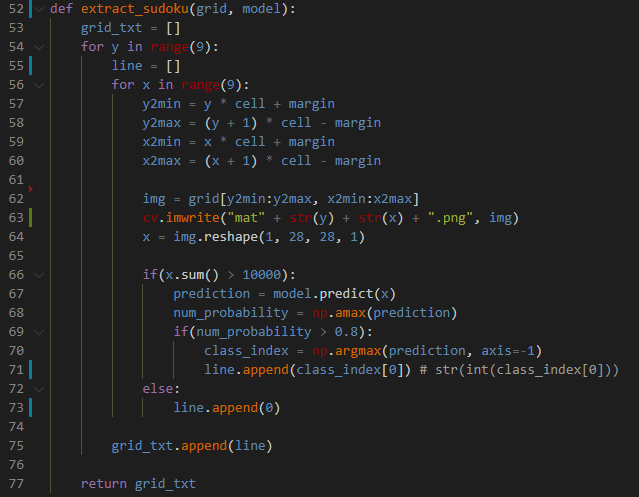
În funcția reorder\_points, se rearanjează ordinea punctelor colțurilor patratului cu sudoku. În linia 35 se aliniază vertical toate vectorile contururilor și se elimină toate celea care conțin dintr-un singur element, adică care nu reprezintă un poligon. Apoi se realizează sortarea punctelor din contur după al doilea parametru din array (coordonata y) (linia 36). itemgetter(1) extrage din obiect indexul a lui al doilea element, pentru ca sortarea să se facă după dânsul. Următorul pas este rearanjarea celor 4 coordonate de la stânga la dreapta, de jos și sus (se clasifică deja după x, deoarece vectorii au fost deja sortați după y). Aceasta este îndeplinit în secvența 37-47. pts2 (48) deja vor fi punctele noi, adică coordonatele noi pentru imagine, care va avea originea în (0, 0) – matrice cu mărimea grid\_size x grid\_size, în final această imagine va fi folosită pentru a schimba perspectiva imaginii originale (privirea de sus asupra imaginii), putând, astfel, fi folosită la procesare și recunoasterea cifrelor din imagine.

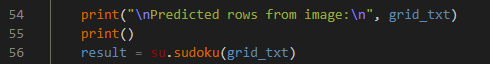


sudoku.py :

Linia 46 face matricea de perspectivă pentru a putea fi aplicată la imaginea originală, apoi (linia 47) ea este aplicată, se convertește în imaginea cu un singur canal (48), apoi se aplică threshold-ul (discretizarea iamginii) (49). Linia 50 afișează imaginea cu perspectiva modificată. Iar linia 52 extrge din imagine fiecare linie a sudoku prin identificarea cifrelor, linia 56 rezolvă sudoku prin matricea de stringuri care conține cifrele prezise.

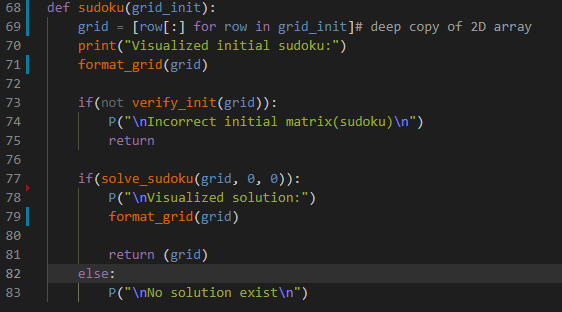
utils.py :

Funcția extract\_sudoku extrage cifrele din imagine. Se aplică tarnsformarile asupra unei celulde din grilă în rândurile 57-60, conform modelului antrenat, apoi imaginea extrasă după aceste coordonate este salvată în calculator pentru a fi apoi vizualizată și sa se mai depistateze alte greseli posibile (funcția imwrite). Apoi imaginea extrasă este remodificată pentru a se putea încadra în model pentru a fi prezisă, prin reshape. Aplicăm un filtru (linia 66) prin care vedem dacă intensitatea pixelor din imagine depășeste o limită (10 000), un pixel din imaginea treshold are intensitatea maximă 255. Următorul pas este extragerea predicției, prin funcția predefinită predict (67), dar totodată și probabilitatea maximă (care apoximativ mereu e 100%) prin funcția amax. Verificăm dacă este o probabilitate decentă și extragem indexul din matricea de probabilități, unde indexul la probabilitate arată cifra ce corespunde la această probabilitate (axis=-1 indică extragerea de pe ultima axă din shape). Linia 71 doar adaugă acest număr în tablou, dacă nu s-a găsit nici un număr se va pune 0. Linia 75 atașează la întreaga matrice de sudoku extras la matricea 2x2.

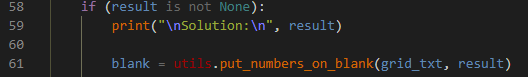
Printăm matricea extrasă, apoi rezolvăm prin funția sudoku definită de noi.

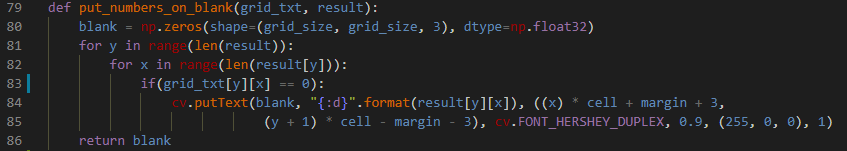
sudoku\_solver.py :

Acest fișier rezolvp sudoku prin cea mai primitivă metodă, și anume cea cu backtracking cunoscută de toți. Voi prezenta pe scurt ce este inclus în acest fișier, deoarece este destul de ușor.

Funcția principală care rezolvă este sudoku, care primește ca parametru matricea extrasă. Linia 69 efectuează deep copy (copiere totală) a matricei inițiale pentru a nu fi modificată cea originală (grid\_init). Apoi matricea este afișată într-o formă mai comodă de vizualizat, în care cifrele care nu sunt prezente (0) sunt reprezentate ca puncte (.). Linia 73 verifică dacă matricea initială a fost extrasă corect, prin funcția definită de noi verify\_init care returnează true dacă nu este depistată nici o greselă. Ea verifică dacă numărul este repetat pe aceeași linie orizontală și verticală în matrice și în pătratul 3x3 din grilă. Linia 77 rezolvă sudoku și scrie rezultatul în variabila grid prin metoda clasică de backtarcking, returnează true dacă este rezolvabilă și păstrează rezultatul în acea variabilă cu care se apelează, apoi se afișează formatat pentru a fi mai ușor vizualizată și se returnează la apelant.

sudoku.py :

Verificăm dacă este vreun rezultat, dacă nu atunci se va afișa imaginea originală cu contur, fără nici o cifră indicată. Printăm soluția (linia 59), apoi creăm o imagine nouă unde plasăm doar cifrele rezolvate, prin funcția put\_numbers\_on\_blank.

 utils.py :

În această funcție se crează o imagine nouă cu cifrele rezolvate înscrise. Linia 80 crează matricea goală de marimea grilei. Pentru fiecare cifră din matricea cu cifrele extrase și rezolvate se găsește locul unde în matricea extrasă nu este nici o cifră (0) și plasează cifra din matricea celor rezolvate, prin putText (funcție predifinită a bibliotecii OpenCV).

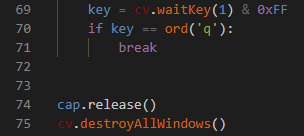
sudoku.py :

Ultimul pas în programă este adăugarea celor 2 imagini (cu cifrele rezolvate puse pe imagine și imaginea originală). Astfel, linia 63 transformă în perspectiva originală a imaginii din pts1 în pts2. Discretizăm imaginea (la micșorarea parametrului thresh se măreste aria unde se aplică threshold-ul) (linia 68). Se creează masca textului care trebuie inserat și inversul măștii. Setăm inversul măștii discretizate (70). Apoi inversăm imaginea (intunecăm regiunea unde trebuie plasate cifrele din imaginea originală pentru a amplasarea cifrelor) (72). Se ia doar regiunea cu cifrele din blankP (doar regiunea care au threshold-ul setat 1) (73). Se pune img2\_fg (imaginea cu cifrele selectate) pe imaginea originală unde regiunea cifrelor este întunecată, prin funcția predefinită add (74). În final, afișăm imaginea cu cifrele completate pe ecran (76).

Deosebirile principale între prelucrarea imaginei selectiv și prelucrarea imaginilor în flux, cum este videoul, este că în videou imaginea se prelucrează constant într-un ciclu infinit, iar acum voi ilustra câteva din aceste deosebiri.

v\_sudoku.py :

Linia 11 declară clasa VideoCapture(0), care este destinată citirii videoului, 0 e indicile dispozitivului de pe care se va realiza citirea, dacă vor fi unite mai multe videocameri în serie, este nevoie de acest indice, însă implicit pentru unicul dispozitiv e 0. Linia 12 definește o variabilă care va fi folosită pentru a reduce numărul necesar de operații pe o imagine din videou. Adică el va determina când să se rezolve sudoku extras, adică o singură dată cand este prezent același sudoku (este prezent conturul extras).

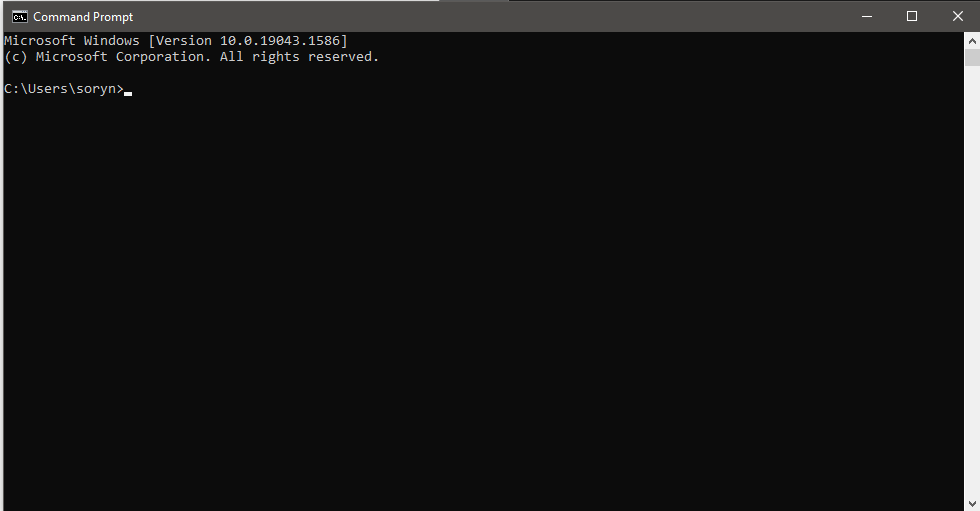
Lini 14 inițiază ciclul infinit, în care cu ajutorul funcției read() se va citi imaginea din videoul și se va procesa ca o imagine simplă (frame), iar ret reprezintă valoarea returnată, dacă imaginea nu va putea fi citită se va returna False.

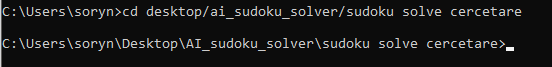
waitKey așteaptă pentru a fi presată o tastă 1 milisecundă, apoi o păstrează în variabila key, iar dacă această tastă e „q” atunci ciclul se încheie (70). În final se eliberează (șterge din memorie) clasa alocată și se închid toate ferestrele active (fereastra cu grila și principală) (74-75).

3.2 Compilare

Pentru a vedea cum funcționează programa este nevoie să o compilăm. În esență, compilarea unei programe este rularea ei, astfel codul scris de noi este transformat în cod-mașină (un set de instrucțiuni întelese de mașină), adică limbajul nativ al calculatorului, iar, ulterior, aceste instructiuni sunt îndeplinite, realizându-se programa noastră.

Pentru a raliza compilarea (cum noi am setat în programă), deschidem CMD la calculatorul pe sistemul de operare Windows (prin căutare manuală sau taparea în meniul sistemului).

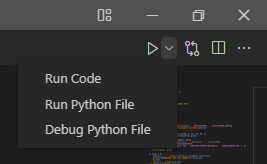
Pentru a compila programa, este nevoie să indicăm locația fișierului care conține programa. În cazul meu va fi: cd desktop/ai\_sudoku\_solver/sudoku solve cercetare. cd reprezintă comanda de a accesa un anumit folder, așadar, se accesează folderul desktop apoi cel unde se conține cercetarea și folderul unde este fișierul.

Mai departe deja trebuie doar de rulat programa. Comanda pentru execuția programelor python e fie py, fie python. Astfel, la mine aceasta va fi: py sudoku.py, în care fișierul indicat este cel principal care se va executa. Apoi indic direct imaginea din folderul images, care să se prelucreze.

Comanda completă va arăta în felul următor: py sudoku.py sudoku-app.jpg

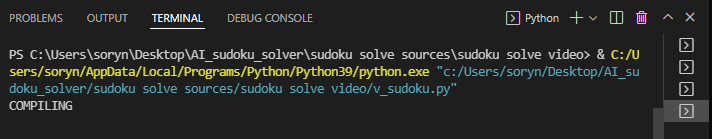
Însă dacă este nevoie de indicat o altă imagine din altă locație, putem pune argumentul –p. Pentru simplitate voi indica aceeași imagine din altă locație. Comanda în cazul meu va fi:

py sudoku.py -p C:\Users\soryn\Desktop\sudoku-app.jpg, unde imaginea sudoku-app.jpg este pusă în folderul desktop, iar argumentul –p este indicat pentru a indica alt drum spre această imagine.

 Iar pentru videou este nevoie de compilat codul în interiorul IDE-ului, căci Visual Studio Code ne permite aceasta, pentru ca toate modulele să funcționeze în mod normal. Pentru aceasta este nevoie de apăsat pe butonul din dreapta sus, în formă de săgeată îndreptată spre dreapta. Putem deodată apăsa pe dânsa sau să selectăm să se compileze anume acest fișier python prin Run Python File.

În consolă ar trebui să se afișeze următorul mesaj:

Dacă nu este afișat nici un mesaj, înseamnă – compilarea s-a realizat cu succes și ar trebui să se pornească camera și să se arate fereastra cu videoul de pe videocameră pe ecran.



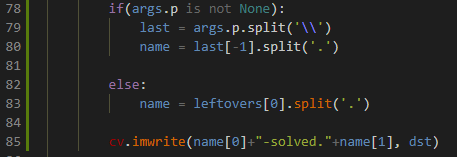
IV. Rezultatul

4.1 Output

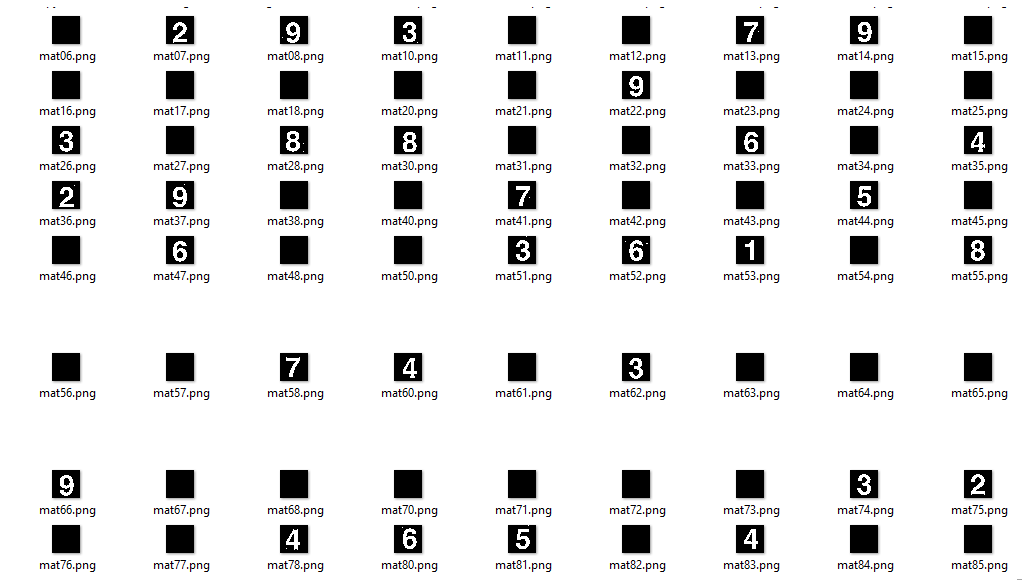
În acest compartiment, voi arăta rezultatul final la ambele programe, pentru cea de prelucrarea a imaginii și videoului. Dacă ceva nu funcționează corect sau nu se compilează, apar erori în cod, aceasta se datorează fie folosirii incorecte a codului, fie învechirii anumitor funcții/proceduri folosite în biblioteca keras/OpenCV/numpy, care nu mai sunt compatibile cu acest cod, sau au căpătat altă întrebuințare. Pentru a soluționa problema este necesar de consultat documentația fiecărei funcții pentru a vedea cu care a fost modificată sau de a accesa internetul pentru a găsi răspunsul pentru că așa situații sunt imprevizibile.

Pentru a vedea ulterior imaginea rezolvată fără a compila programa de mai multe ori, putem să o salvăm prin funția write din librăria OpenCV. Pentru aceasta adăugăm următoarele linii în programul nostru:

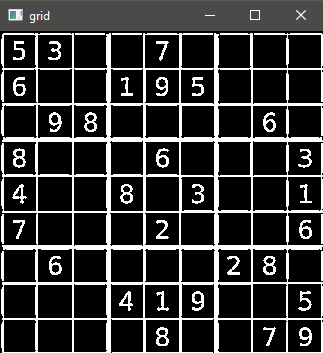
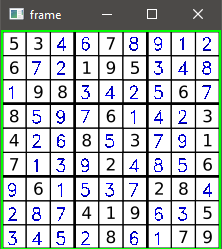
sudoku.py :

Acest cod verifică dacă este prezent drumul spre altă imagine (78) și separă toate folderele din drum (79), apoi extrage doar denumirea fișierului și îl separă în numele fișierului și tipul lui de date (80). Dacă nu este indicat argumentul atunci doar se selectează numele fișierului și tipul lui (83) și se salvează imaginea rezultantă în folderul unde se află programa care a fost rulată.

Rezultatul compilării:

În folderul cu programă vor apărea 81 de imagini de mărimea 28x28 și îs anume acele imagini extrase din grila cu threshold, adică imaginile care au fost încărcate în model pentru a fi recunoscute cifrele. Numerele din numele imaginii indică indexul rândului și coloanei din grila extrasă.

Iar acea grila din care se extrag toate numerele și este o perspectivă modificată a imaginii cu sudoku, care cuprinde doar pătratul cu cifre. Un model de grilă de la o imagine poate fi:

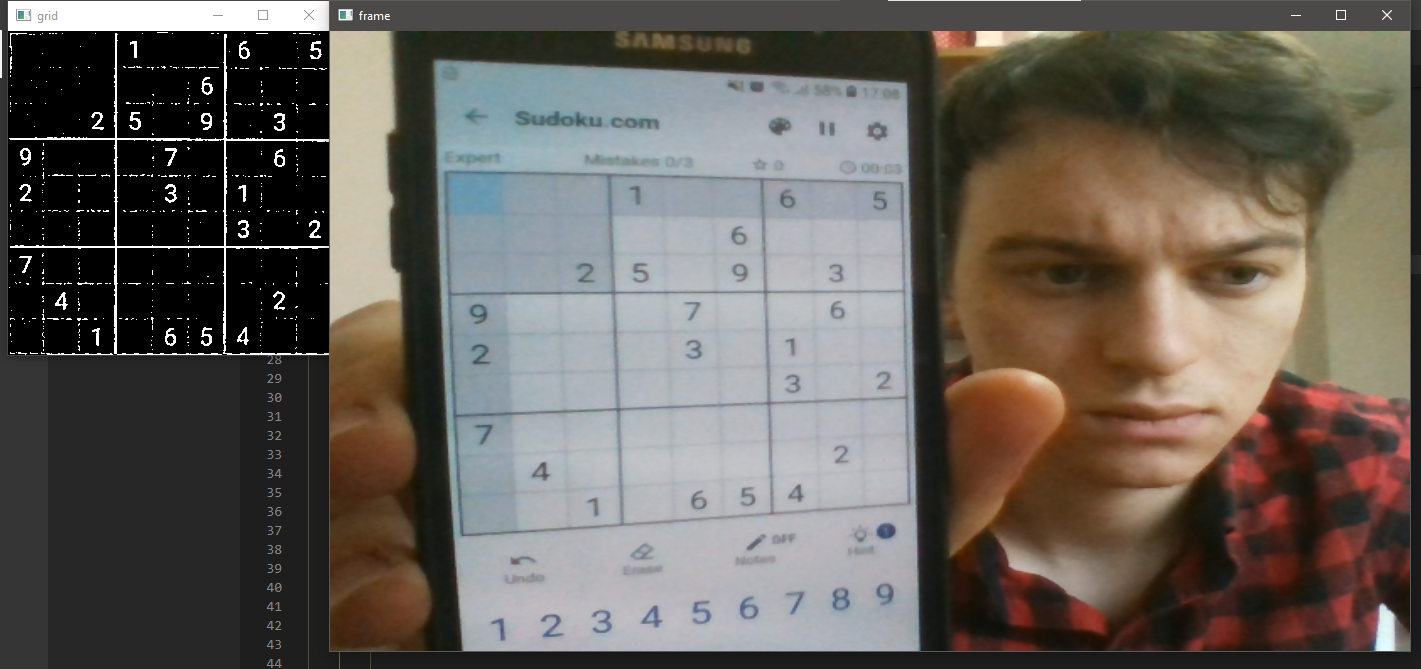


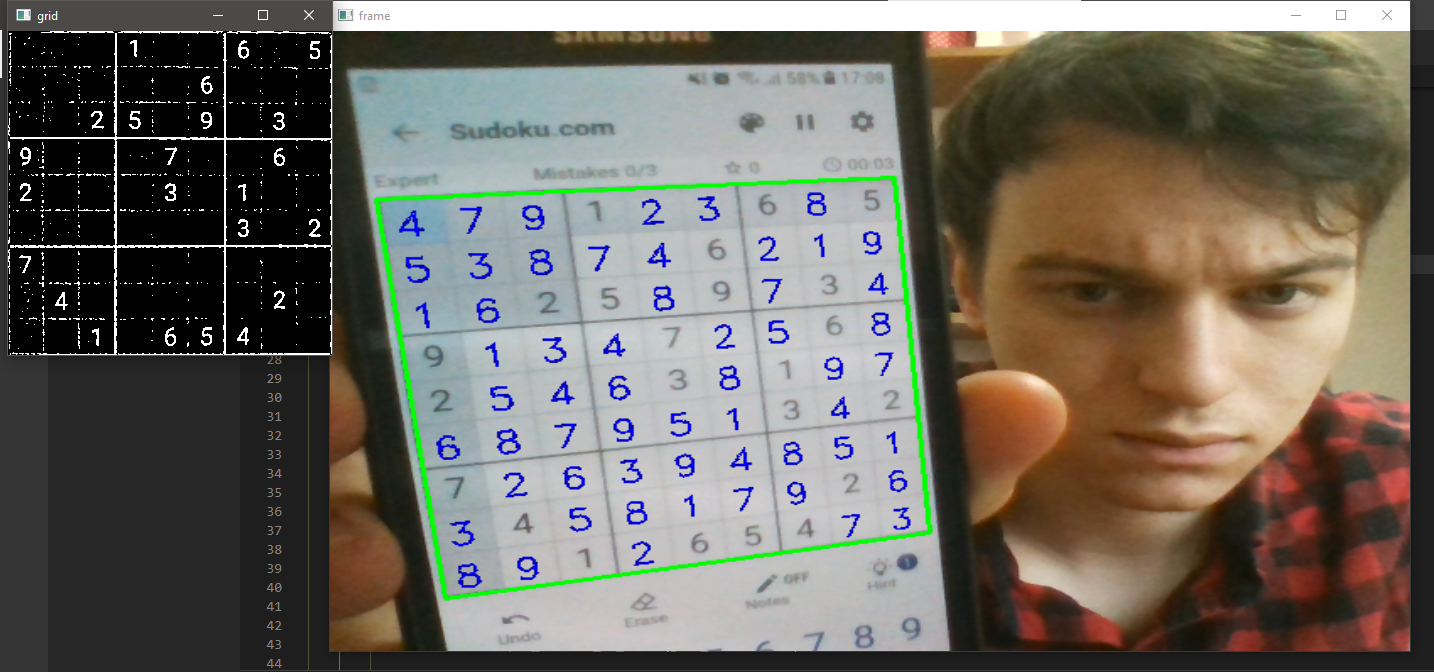
Câteva din rezultatul procesării imaginilor:

|  |  |
| --- | --- |
| **Imaginea inițială** | **Imaginea finală** |
| C:\Users\soryn\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sudoku1.jpg | C:\Users\soryn\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sudoku1-solved.jpg |
| C:\Users\soryn\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sudoku-app.jpg | C:\Users\soryn\AppData\Local\Microsoft\Windows\INetCache\Content.Word\sudoku-app-solved.jpg |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Ultimele 2 exemple nu arată nici un rezultat final, aceasta este condiționat de problemele care vor fi discutate în secțiunea viitoare, și anume zgomote prea intense sau perspectiva prea diformată, prin care nu se poate extrage calitativ pătratul sudoku, iar din cauza zgomotelor care nu s-au eliminat în timpul preprocesării imaginii, cifrele nu pot fi recunoscute, imaginile încărcate în model fiind imprecise, dând, astfel, rezultate ironate. Chiar dacă în penultima imagine se vede clar cifrele, s-ar părea să fie într-o calitate bună, însă în realitate la etapa procesării se poate observa calitatea ei, în realitate zgomotele de pe imagine demonstrează cât de deformată este, iar vizualul poate fi uneori amăgitor. În aceasta și constă cea mai mare problemă, anume faptul că noi putem percepe cifrele și caracterele chiar și în medii destul de deformate, pe când modelul trebuie să fie suficient de antrenat, ceea ce depășeste scopul acestei cercetări.

Rezultatul procesării video:

Imaginea înaintea începerii procesării:

Imaginea după procesare:

A fost ales un sudoku de cea mai înaltă greutate, iar procesarea video se realizează destul de lent.

V. Concluzie

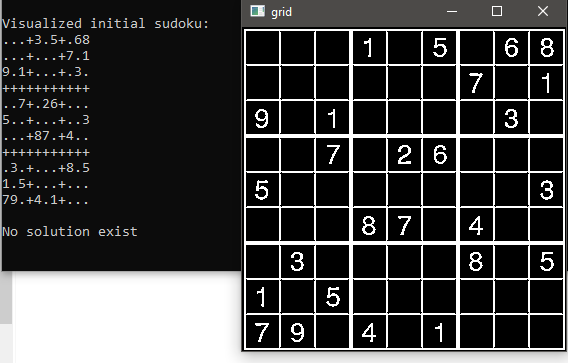
5.1 Probleme

La realizarea acestui proiect am întâmpinat mai multe greutăți, nu cele de programarea care, în realitate, au fost cele mai simple, ci la algoritmi și imagini. Astfel, voi descrie principalele probleme și soluțiile posibile pentru a fi rezolvate:

Problema predicției cifrelor:

Modelul care este în acest proiect a fost antrenat pe prea puține date, care au fost destul de clare, iar când se încărcau imagini în model pentru a fi prezise, care erau destul de calitative, oricum se comiteau erori, deoarece cât de calitative nu ar fi imaginea, aceasta nu a fost suficient, sau are un factor de asemănare cu alt număr destul de mare. Această problemă ține de ordinul modelului și a datelor.

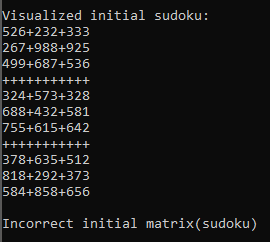
Soluție:

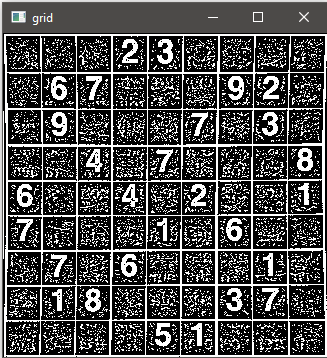
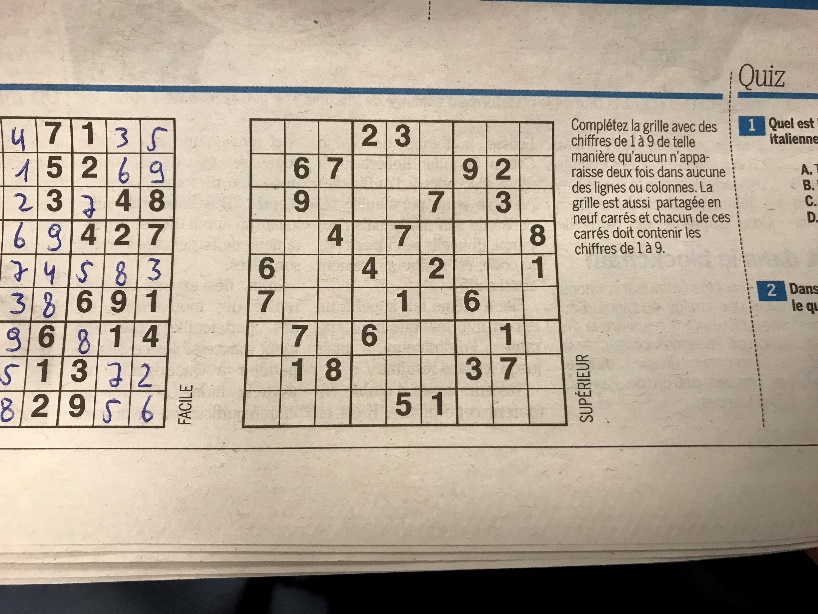
Antrenarea unui nou model mai calitativ, care va avea nevoie de resurse de timp și de o bază de date suficient de mare pentru a prelucra mai multe variante posibile, plus la aceasta se adaugă resursele calculatorului care va trebui să antreneze modelul.

Aici se poate observa cum prima cifră din sudoku (rândul 1 coloana 4) este interpretată ca cifra „3” pe când în realitate e cifra „1”, din cauza acestei erori minore, întregul sudoku nu poate fi rezolvat pentru că cifrele aici contează cel mai mult.

Problema imaginilor:

Imaginile introduse pot fi de proastă calitate, pot fi prea tare distorsionate, cu perspective prea modificate, prea blurate, sau cu prezenta altor detalii care ar influența rezultatul extragerii sudoku. Chiar dacă imaginea pare a fi clară, ea oricum la etapa prepocesării poate arăta neajunsurile. Un astfel de exemplu poate fi prezentat prin următoarea imagine:



Acest sudoku din cauza zgomotelor nu se poate rezolva, deoarece fiecare pătrat unde este prezent zgomote este interpretat ca o cifră separată. Chiar dacă cifrele din pătratele completate îs indicate corect, celelalte pur și simplu sunt prezise incorect.

Soluție:

Pentru a rezolva această problemă este doar necesar de selectat imagini mai corecte, dacă ele sunt luate de pe internet, sau de a le face cât mai calitativ, de obicei o fotografie la lumina zilei cu orice cameră este suficient. Iar restul deja depinde complet de modelul antrenat și de algoritmul de selecție.

Problema resurselor calculatorului:

Această problemă constă în simplul fapt că pentru a reduce din timpul compilației la procesarea imaginii sau cel mai important pentru prelucrarea mai operativă a sudoku-ului din video sunt necesare de resurse de calcul mai avansate. Astfel, pentru ca videoul să nu să se oprească și să ruleze cât mai plan fără freez-uri este nevoie de un procesor cât mai bun și de dorit un video card (daca e laptop cu video card integrat atunci procesorul să fie mai mult sau cel puțin intel i5 ultimile generații). Dacă se va opera doar cu procesorul atunci se va descărca biblioteca tensorflow pentru CPU, iar daca video card-ul e destul de bun, atunci cu tensorflow pentru GPU. Aceste librării au nevoie de resurse mari ale calculatorului, chiar dacă le gestionează destul de eficient.

Soluție:

Soluția este destul de simplă, trebuie de investit și de achiziționat o tehnică mai nouă. Aceasta nu numai că va aduce avantaj în viteza de calcul și procesare, dar și va accelera antrenarea modelului ce, de obicei, consumă mult timp și resurse.

5.2 Dezvoltarea proiectului

Prin rezolvarea problemelor prezentate anterior, se poate de îmbunătățit proiectul astfel încât să devină ceva mai mult. Sunt mai multe posibilități de a-l dezvolta și voi prezenta câteva din ele care sunt destul de realizabile, dar care, iarăși, depășesc scopul acestei cercetări:

* Dezvoltarea unei aplicații mobile sau desktop care a-r rezolva un sudoku prin camera video, fie prin capturarea imaginii, salvării, rezolvării și printării rezultatului. Aplicația mobilă s-ar folosi fie de imaginea din galerie, fie de fotografia realizată din aplicație, sau prin video care, de asemenea, va recunoaște figura pătrată a sudoku din fluxul de imagini. Dacă se realizează o aplicație desktop, e mai ușor de a face o aplicație în care se vor putea încărca diverse poze cu sudoku pentru a fi rezolvate, sau care va porni camera. Aceste aplicații ar fi inutile în cazul când plăcerea pentru joc constă în rezolvarea desinestătătoare, însă pot fi necesare pentru a fi văzut rezultatul corect sau folosit pentru scopuri educative. Realizând acest tip de aplicații se poate de învățat o multime de alte tehnologii precum Mobile Development, App Development, librării care sunt folosite pentru Graphic User Interface (GUI), ce este destul de întrebuințat în orice loc de muncă pentru un developer.
* Un alt tip de dezvoltare ar fi ceva mai modest. S-ar putea de îmbunătăți procesul de indicare a imaginii care este nevoie de a fi procesată, căci lucrul cu consola este destul de anevoios pentru cei care niciodată nu s-au întâlnit cu așa tip de aplicații. Pentru aceasta s-ar putea de făcut o aplicație care ar folosi tehnologia drag and drop, la imaginile care trebuie procesate. Se poate de folosit librăria de operare cu GUI, Tkinter, Qt etc.
* Un alt tip de dezvoltare ar fi bazarea doar pe procesarea video, folosirea unui algoritm mai bun de rezolvare a sudoku, pentru că viteza este cel mai important. În privința această, se poate de trecut întreaga aplicație cu procesare video pe limbajul C/C++, cu aceleași librării tensorflow, programarea va fi mai grea, însă videoul se va procesa semnificativ mai rapid. Axarea pe videou are mai multe beneficii.

5.3 Concluzii

În timpul realizării acestul proiect am aflat destul de multe lucruri noi, mai ales cum se operează cu librăriile pentru inteligența artificială și de procesare a masivelor de date. Totodată, consider că am realizat complet scopul și obiectivele propuse, reazlizând partea practică și cel mai important înțelegând complet procesul de lucru al unor astfel de aplicații și a altor ce sunt bazate pe prelucrarea imaginilor și folosirii IA.

Bibliografie

1. <https://github.com/1nfinityLoop/Sudoku-Solver-AI>

2.<https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2020-2021/MakalahUAS/Makalah-UAS-IF4073-2021%20(11).pdf>