Cercetare

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

A realizat: Beșleaga Sorin

Consultant științific: Miron Raisa

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...................................................................................

IV. Rezultatul

4.1 Output......................................................................................

V. Concluzie

5.1 Probleme........................................................................................

5.2 Dezvoltarea proiectului.................................................................

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

│

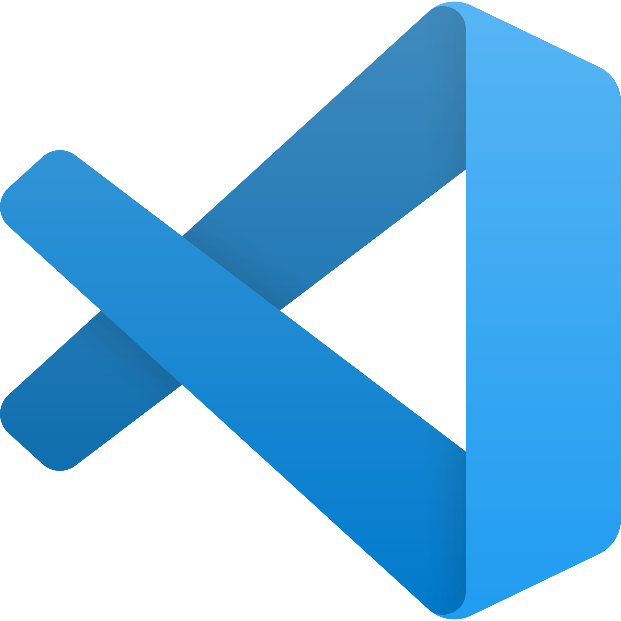
└───\_\_pycache\_\_

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

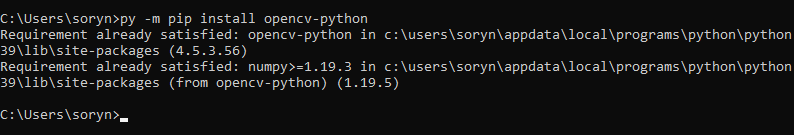
Pentru a lucra asupra proiectului și pentru a programa este nevoie de anumite dependențe precum: opencv, tensorflow, numpy etc. Instalarea și importanța lor va fi expusă mai detaliat în secvența ce urmează. Astfel, avantajul folosirii limbajului Python asupra altor limbaje (C++ care de asemenea susține 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.

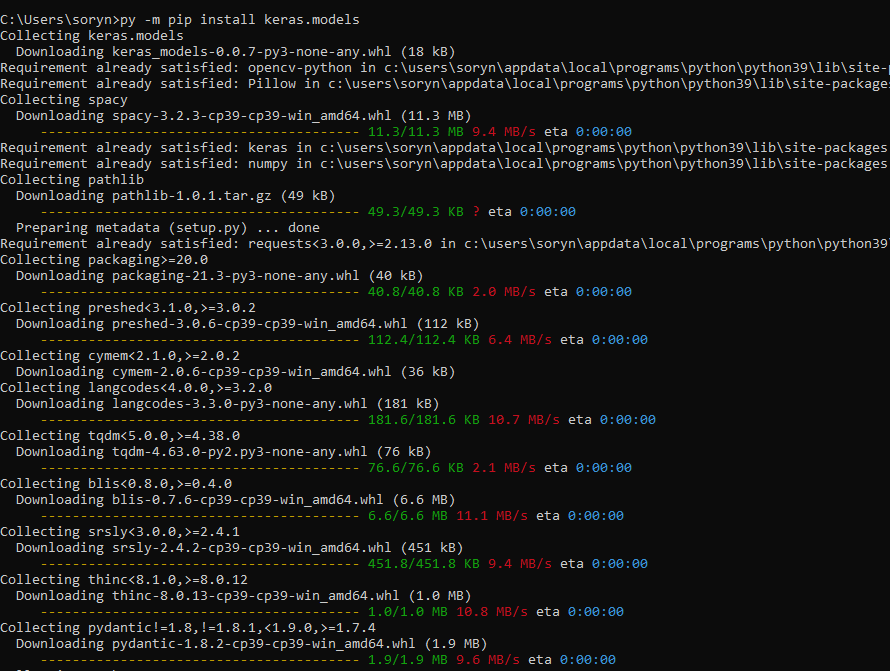
**Instalarea keras.models:**

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:

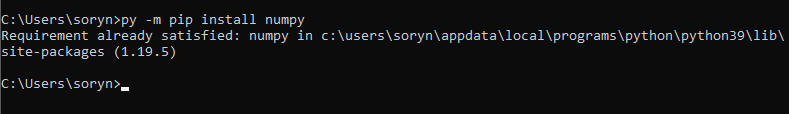


**Instalarea NumPy:**

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.

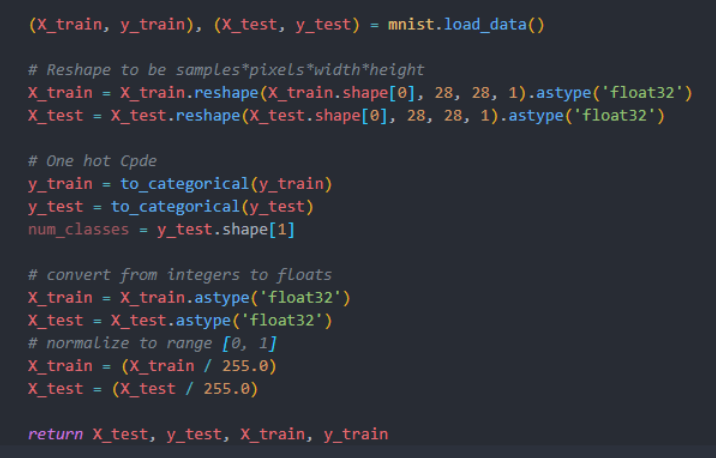
Pentru instalarea NumPy, folosim comanda:

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.

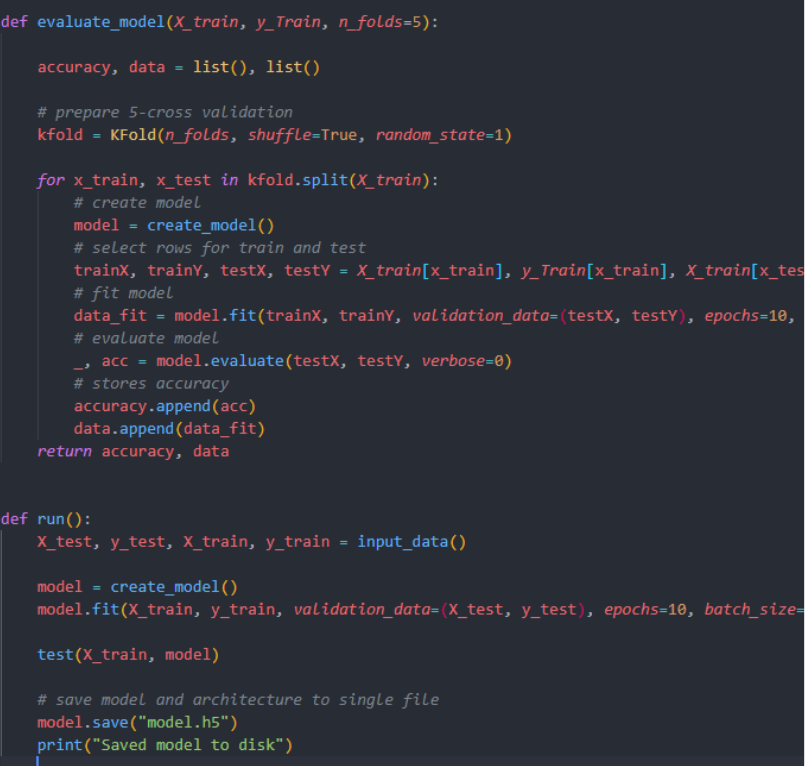
Pentru modelul de machine learning, a fost folosit baza de date MNIST.

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.

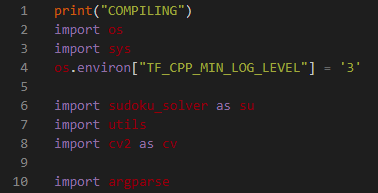
Astfel, modelul antrenat poate atinge acuratețea până la 100 %, aceasta fiind oarecum un rezultat relativ.

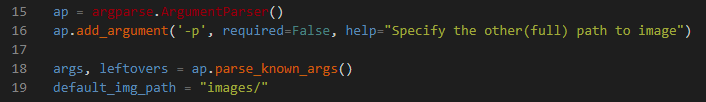
III. Partea practică

3.1 Descrierea programului

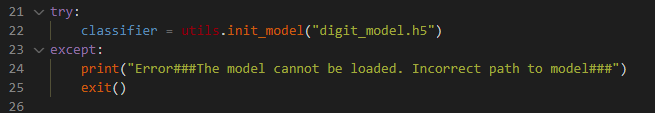
În această parte a cercetării voi descrie mai detaliat programul care corespunde alogritmului expus în partea cu scopul lucrării. În principal va fi descris fișierul care va rula întreaga programă și anume sudoku.py, dar vor mai fi devieri și spre alte fișiere care conțin funcțiile de care are nevoie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

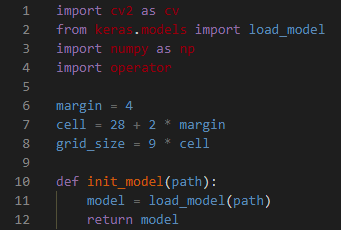
sudoku.py :

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.

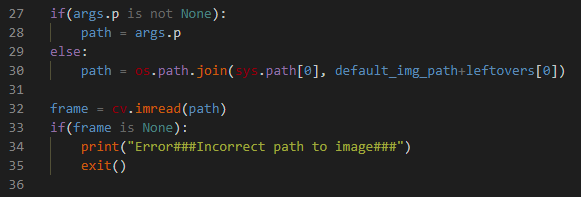
În linia 22, se încearcă încărcarea modelului digit\_model.h5, prin funcția init\_model, în caz de eroare se va afișa mesajul corespunzător în linia 24 și se va ieși din program.

 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.

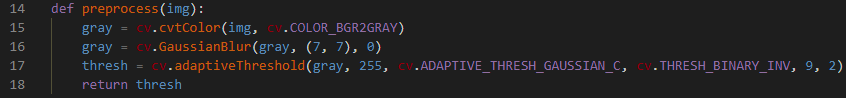
Liniile 6-8 sunt necesare pentru a seta offsetul (marginile), mărimile celulei în sudoku, și mărimea întregii grile. Aceste mărimi au fost folosite pentru antrenarea modelului de aceea ele pot fi considerate constante. 28 este mărimea necesară pentru importarea imaginii în model, care recunoaște imaginile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

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

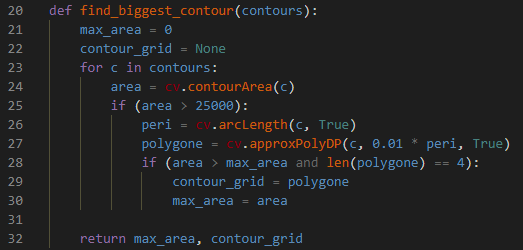
 sudoku.py :

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 :

