



UNIVERSITATEA DIN
BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ



SPECIALIZAREA INFORMATICĂ

Lucrare de licență

TITLUL LUCRĂRII DE LICENȚĂ

Absolvent

Hadirca Dionisie

Coordonator științific

Titlul și numele profesorului coordonatorului

București, iunie 2021

Rezumat

Odata cu aparitia internetului, Securitatea Sistemelor Informationale a devenit o necesitate de baza a oricarei aplicatii, iar pe parcursul anilor, implementarea acesteia a devenit din ce in ce mai puternica. De la algoritmi de criptare a datelor utilizatorilor, pana la diverse masuri de protectie anti-spam, astfel incat interfata si experienta utilizatorilor sa fie cat mai simpla, iar libertatea programatorilor, cat mai restrictionata. Una dintre aceste masuri de protectie poarta numele de CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart).

Exista mai multe tipuri de astfel de test, cum ar fi imagini cu diferite obiecte care trebuie identificate, imagini cu text ce trebuie transcris, si chiar si CAPTCHA-uri audio. Chiar daca rolul CAPTCHA-urilor era sa previna atacuri cibernetice si atacurile de spam asupra unui website, fiind introduse sub forme de input-form, odata cu trecerea timpului, programatorii au reusit sa creeze niste sisteme de ocolire, dezvoltand software-uri rau-intentionate, cu scopul de a rezolva CAPTCHA-ul intampinat.

In cadrul acestei lucrari, ce poarta denumirea "Breaking CAPTCHA" ne vom axa pe tipul de CAPTCHA bazat pe text si vom observa cum putem cu ajutorul Vederii Artificiale sa construim un program care sparge CAPTCHA-urile respective. Se va explica in detaliu fluxul de lucru, obiectivul programului, algoritmi implementati, si pasii efectuati, astfel incat la final sa obtinem un soft cat mai curat si ideal. Programul va fi implementat cu ajutorul limbajului de programare Python, iar pe langa bibliotecile secundare vor fi utilizate 3 biblioteci principale dezvoltate special pentru lucrul cu Inteligenta Artificiala: NumPy, PyTorch, OpenCV. Seturile de date cu imagini CAPTCHA, descrise in lucrare, vor fi generate din librarii implementate in diverse limbaje, cum ar fi JavaScript, Python, Rust, etc.

Spre final vom incerca sa imitam un scenariu din viata de zi cu zi, si sa punem in practica bot-ul creat, utilizand o aplicatie secundara cu scopul de a observa comportamentul programului, remarca progresele, si de a formula concluzii asupra masurii de protectie, revenind cu o potentiala extensie a studiului efectuat, posibile imbunatatiri ale sistemului anti-CAPTCHA creat, dar si cu o sinteza precum si o serie de recomandari in care vor fi exemplificate tehnicile de protectie care opun o rezistenta mai mare, unor astfel de tipuri de atac.

Abstract

With the advent of the Internet, the Security of Information Systems has become a basic necessity of any application, and over the years, the implementation has become more and more powerful. From user data encryption algorithms, to various anti-spam protection measures, so that the user interface and experience is as simple as possible, and the freedom of programmers, as restricted as possible. One of these protection measures is called CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart).

There are several types of such tests, such as images with different objects to be identified, images with text to be transcribed, and even audio CAPTCHAs. Even if the role of CAPTCHAs was to prevent cyber attacks and spam attacks on a website, being introduced as input-forms, with the passage of time, programmers managed to create some circumvention systems, developing malicious software, in order to solve the encountered CAPTCHA. In this work, which bears the name "Breaking CAPTCHA", we will not focus on the type of CAPTCHA based on text and we will observe how we can, with the help of Artificial Vision, build a program that breaks the respective CAPTCHAs. The workflow, the objective of the program, the algorithms implemented, and the steps performed will be explained in detail, so that in the end we get a software as clean and ideal as possible.

The program will be implemented using the Python programming language, and in addition to the secondary libraries, 3 main libraries developed specifically for working with Artificial Intelligence will be used: NumPy, PyTorch, OpenCV. The data sets with CAPTCHA images, described in the paper, will be generated from libraries implemented in various languages, such as JavaScript, Python, Rust, etc.

Towards the end, we will try to imitate a scenario from everyday life, and put the created bot into practice, using a secondary application in order to observe the behavior of the program, note the progress, and formulate conclusions on the protection measure, coming back with a potential extension of the study carried out, possible improvements of the anti-CAPTCHA system created, but also with a synthesis and a series of recommendations in which the protection techniques that oppose greater resistance, some such types of attack will be exemplified.

Cuprins

1	Introducere	5
2	Preliminarii	7
2.1	Istoric CAPTCHA	7
2.2	Tehnici de Protectie	9
3	Instrumente de lucru si Inteligenta artificiala	11
3.1	Istoric Deep Learning	11
3.2	Instrumente de lucru in Vederea Artificiala	11
3.3	Fluxul de lucru	13
3.4	Google Colab	14
4	Implementarea Aplicatiei, Experimente Realizate	15
4.1	Prezentare - Seturi de date CAPTCHA	15
4.1.1	Lepture-CAPTCHA	17
4.1.2	Samir-CAPTCHA-rs	18
4.2	Preprocesarea imaginilor	19
4.2.1	Preprocesare - Aditonal	23
4.2.2	Analiza spatiului de etichete	23
4.3	Constructia Retelei Neurale Artificiale	26
4.3.1	Plotarea Datelor	29
4.3.2	Aplicatia Secundara	31
4.3.3	Script-ul de atac	32
	Bibliografie	35

Capitolul 1

Introducere

Odata cu dezvoltarea internetului si cresterea numarului de persoane care au acces la platformele web, a fost necesara introducerea unor sisteme care impiedica sau provoaca inconveniente in automatizarea proceselor, cum ar fi creare excesiva de conturi, colectare de date, reclama fortata, cu alte cuvinte, sisteme care implementeaza masuri de protectie impotriva botilor. Astfel a fost introdusa CAPTCHA.

CAPTCHA este prescurtare de la Completely Automated Public Turing Test to Tell Computers and Humans Apart.

Ce este Testul Turing? Testul turing original denumit si "jocul de imitatie", creat de Alan Turing, este un experiment implementat pentru a testa capacitatea masinii de a manifesta un comportament inteligent, echivalent, sau chiar si indiscutibil fata de cel al oamenilor. [7]

Termenul de CAPTCHA a fost inventat in anul 2003 de catre Luis von Ahn, Manuel Blum, Nicholas J.Hopper, si John Langford, iar primul si cel mai comun tip de CAPTCHA a fost inventat prima data in 1997 de catre 2 grupe de oameni de stiinta lucrând in paralel [12]. Mai jos se poate observa prima versiune a CAPTCHA-ului.



Figura 1.1: First Version of CAPTCHA

Dupa cum se poate observa, prima versiune de CAPTCHA este una bazata pe text, dar pe langa aceasta pe parcusul evolutiei au fost dezvoltate si alte tipuri cum ar fi, bazate pe imagini, sau audio. Initial s-a constatat ca CAPTCHA-urile puteau fi folosite in scopuri de securitate deoarece nu se stia, nu era descoperit, sau inventat un program care ar reusi sa rezolve problema intampinata

O vulnerabilitate majora, care nu era considerata la momentul crearii CAPTCHA-urilor datorita nepopularitatii si progreselor minore, era si ramane un alt subdomeniu al informaticii, Inteligenta Artificiala, tinta caruia este sa imite comportamentul uman, sa detecteze, si sa identifice obiecte de interes intr-o imagine, secventa video, etc.

In aceasta lucrare ne vom axa pe CAPTCHA-uri de tip text, de altfel cum am si mentionat mai sus, cel mai comun tip de CAPTCHA, vrem sa evaluam starea curenta a tehnologiilor AI experimentand cu diverse dataseturi de CAPTCHA bazate pe text si sa determinam daca inca mai este valida ipoteza creata, si mai exact, daca CAPTCHA-urile de tip text manifesta o rezistenta puternica impotriva programelor automatizate, boti.

Aplicatia principala este un script de python care ruleaza pe fundal face http-request catre aplicatia secundara, care e o platforma web, si creeaza in continuu conturi pentru platforma respectiva, folosindu-se de AI-ul antrenat pentru rezolvarea CAPTCHA-ului corespunzator

Aplicatia secundara este o aplicatie web, vulnerabila, simpla, creata cu ajutorul framework-urilor Next.js care serveste ca scop, ilustrarea functionalitatii aplicatiei principale

Structura lucrarii este urmatoarea:

Capitolul I: Prezentarea generala a CAPTCHA-urilor. Vor fi prezentate librariile de Captcha utilizate pentru generarea seturilor de date, si folosite pe parcursul experimentelor, vor fi ilustrate tehnicile de protectie care sunt folosite, evolutia generala a lor, si un tabel cu situatia actuala.

Capitolul II: Prezentare generala a Invatarii Automate Adanci. Vor fi descrise bibliotecile folosite, va fi explicat fluxul de lucru in privinta realizarii aplicatiei.

Capitolul III: Experimentele realizate si rezultatele respective. Alegerea generatoarelor de imagini CAPTCHA tinta, urmata de justificarea modalitatii de spargere, proiectarea sistemului AI si compararea acestuia cu SOTA existent.

Capitolul IV: Vor fi structurate sub forma de sinteza concluziile la care am ajuns, problemele pe care le-am intampinat, niste propuneri cu ce se poate imbunatati, cat si recomandari personale in privinta alegerii masurilor de protectie.

In continuare vom vorbi doar despre CAPTCHA bazat pe text, si pentru simplitate ii vom spune doar CAPTCHA

Capitolul 2

Preliminarii

2.1 Istoric CAPTCHA

Primul CAPTCHA (Figura 1) a fost folosit in 1997 de catre motorul de cautare Alta-Vista si a impiedicat boti sa adauge URL-uri (Uniform Resource Locator) la motorul lor web [10]. Putem observa masurile de protectie intreprinse in cazul respectiv, caracterele sunt distorsionate, iar imaginea de fundal are cromatica unui gradient. In trecut doar aceste lucruri erau de ajuns pentru asigurarea protectiei, impotriva programelor rau intentionate, dar datorita progresului in stiinta, bariera a fost ocolita de catre un soft OCR (Optical character recognition) implementat in anii 2000 [10].

CAPTCHA-urile au devenit mai complexe, diverse nivele de zgomot si distorsii au fost adaugate la imagini, astfel incat OCR-urile populare au devenit ineficiente iar taskul pentru spargerea CAPTCHA-urilor a devenit costisitor pentru partea atacanta. Developerii de CAPTCHA in schimb trebuiau sa aiba grija cu nivelul de anti OCR aplicat imaginilor intrucat uneori se producea o inconvenienta mai mare pentru om decat pentru calculator sa le recunoasca [10].

O versiune mai complexa a primului CAPTCHA a fost implementata in Rusia, pentru platforma web Yandex.com.



Figura 2.1: Improved CAPTCHA

Remarcam un nivel de distorsie mai mare a caracterelor in imagine precum si diferentierea in culori a textului combinat cu imaginea de fundal ce corespunde unei valori

aleatoare intr-un spectru (0 - 255) de la negru la alb. Deasemenea imaginea de fundal este inconsistenta, tehnica care eroneaza si previne atacuri de spargere care utilizeaza anumite euristici precum diferenta neschimbata intre culorile imaginii, aceasta fiind cea mai mare problema a atacatorului - separarea caracterelor de imaginea de fundal. Astfel pentru a avansa in spargerea CAPTCHA-ului este necesara identificarea fonului si setarea fiecarui pixel al sau cu 0 (negru) iar fiecare pixel care apartine unui caracter cu 1 (alb) procedeu care se numeste binarizarea imaginii [10].

In 2005 isi face debutul primul CAPTCHA creat de Google, cu numele reCAPTCHA.

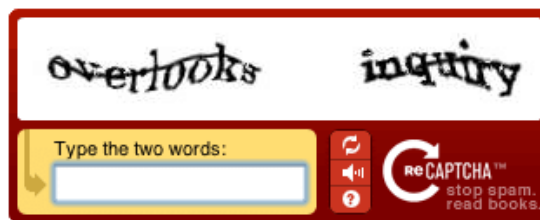


Figura 2.2: Google reCAPTCHA

Ca masuri de securitate, putem remarca multiple cuvinte, forma de valuri a acestora, precum si o linie peste caractere care impiedica segmentarea usoara a lor. reCAPTCHA-ul se deosebeste prin modul in care a fost programat, cuvintele fiind scanate dintr-o carte aleasa de adminului programului, si alese prin ajutorul Crowd-Funding-ului.

Programul reCAPTCHA alegea cuvintele in felul urmator, unul care e lizibil de catre OCR si celalalt nu. Apoi CAPTCHA-urile erau pasate unui grup de utilizatori, care pe rand rezolvau testele respective. Daca primul cuvant era introdus corect, programul presupunea ca si al doilea cuvant este corect, al doilea cuvant fiind pasat urmatorilor utilizatori ca fiind primul in testele noi. Programul mai apoi compara raspunsurile si avea destule dovezi pentru a recunoaste care cuvinte au fost introduse corect.

Astfel programul avea un scop dublu:

- Verificarea utilizatorului - Om/Robot?
- Verificarea cuvantului care nu putea fi citit de catre OCR si digitalizarea acestuia pentru cunoasterea umana. [8]

Ideea Google-ului era sa digitalizeze un numar cat mai mare de carti, ajungand la un numar de 40 milioane in 2019, potrivit wikipedia[1], si de a determina oamenii sa identifice caractere si cuvinte care nu puteau fi identificate de cei mai buni algoritmi de procesare a imaginilor existente la acel moment [8]

In acelasi timp multi programatori construiau diverse tipuri de CAPTCHA implementand multiple tehnici de protectie, care pot fi urmarite in imaginea de mai jos:



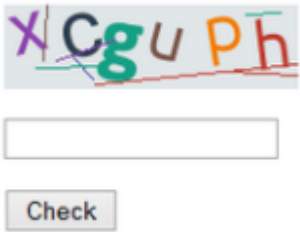

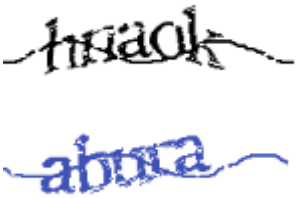

Figura 2.3: Tehnici de Protectie Captcha, Imagine preluata din [13]




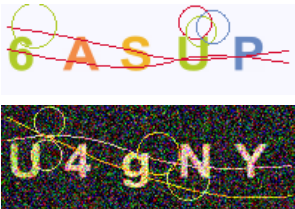
2.2 Tehnici de Protectie

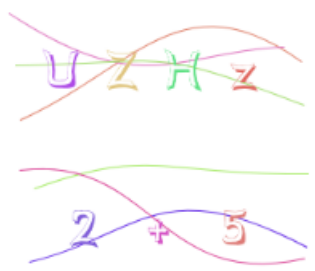
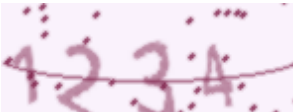
- Distorsiunea Caracterelor - Orice fel de modificare a caracterului cum ar fi rotirea sau deformarea acestora impiedica botii inspre utilizarea potrivii sabloanelor, cu niste imagini ale caracterelor prestabilite, pentru identificarea rapida a caracterelor distorsionate.
- Distorsiunea Imaginii de Fundal - Aplicarea diferitor manipulari asupra fonului, cum ar fi diverse culori, sau imagini inconsistente, acopera o mare vulnerabilitate in spargerea CAPTCHA-ului, problema de baza, si cea mai complicata fiind separarea fonului de caractere.
- Introducerea zgomotului in imagine - Zgomotul in imagine aplicat corect reuseste sa uneasca caracterele si imaginea de fundal astfel acestea raman lizibile pentru om dar foarte greu de distins de catre calculator, iar incercarea eliminarii acestuia, duce catre eliminarea unor componente foarte importante in CAPTCHA, cum ar fi portiuni de caractere sau chiar caractere intregi.

- Utilizarea diferitor fonturi - Stilul, marimea si originea fontului este o aplicatie importanta, fiindca adauga complexitate si provoaca inconveniente botilor in replicarea si identificarea caracterelor .
- Amplasarea aleatoare a caracterelor - Pozitia caracterelor in CAPTCHA, pot cauza erori in programul malitios si euristica utilizata de catre atacator.
- Concatenarea caracterelor - Atacatorul trebuie sa depuna efort inspre proiectarea unui program care separa caracterele concatenate, astfel incat imaginea este segmentata corect.
- Adaugarea elementelor secundare - Elemente secundare cum ar fi puncte, linii, figuri geometrice aleatoare in imagine sunt si ele considerate fiind zgomot, dar e mai putin aleator, si reprezinta o piedica in segmentarea imaginii, detectia precum si recunoasterea caracterelor.
- Lungimea textului - O tehnica simpla dar puternica, se bazeaza pe faptul ca OCR-ul nu va reusi sa identifice corect toate caracterele din CAPTCHA, prin urmare daca lungimea textului este 15, iar OCR-ul a ghicit doar 14 caractere, testul va fi considerat incorect
- Combinatii intre tehnici - Evident combinarea a doua sau mai multe procedee descrise mai sus, sporeste complexitatea testului prin urmare si dificultatea intampinata de algoritmii automatizati este mai mare

In continuare atasez un tabel cu generatoarele de CAPTCHA care au fost analizate si in cadrul carora au fost executate experimentele din lucrarea respectiva. Librariile prezentate au fost construite de alti programatori, si se pot gasi open-source, aferent link-urilor atasate acestora.

Library	Examples	URL Open Source	Anti-Segmentation Method	Anti-recognition Method	Automated Extraction and Saving	Note
Captcha for Laravel 5/6/7		https://github.com/mewebstudio/captcha	Random lines of different colors, Random background details such as background fog	Random fonts, Multiple captcha-recognition methods such as addition of 2 numbers or text copying	Function Captcha::create()	Useful for training segmentation Will have to get clear text values of captcha. PHP
Gregwar captcha		https://github.com/Gregwar/Captcha	Random lines, Various background colors	Character distortion, Different fonts	CaptchaBuilder builder → <i>build</i> builder → <i>save</i>	Basic Captcha May be useful training on character distortion. PHP
trekjs captcha		https://github.com/trekjs/captcha	Text with noise, Random lines	character distortion	Function: run() example on github	Gif Captcha will have to extract images. Javascript
svg- captcha		https://github.com/product/svg-captcha	Almost no methods, Few random lines, Text contains the same color as the background	Random text-color, Random captcha recognition methods such as addition of 2 numbers or text copying	Functions randomText() create()	Generates SVG files will have to convert them to PNG. Javascript

Library	Examples	URL Open Source	Anti-Segmentation Method	Anti-recognition Method	Automated Extraction and Saving	Note
go-captcha		https://github.com/wenlg/go-captcha	Random lines of different colors, Dots, Noise, then another image with different background and same symbols	Hard character detection even for the human eye, in-order character selection, rotated characters	Functions GetCaptcha() Generate()	Hardest captcha in my opinion detection and identification in two places Can be useful to train segmentation. GO language
steambap captcha		https://github.com/steambap/captcha	Random lines, Noise, Text interflows with the background	Different fonts, Character distortion	Function handle()	Tricky to identify characters useful on training with noise and lines. GO language
tarunk04 captcha-generator		https://github.com/tarunk04/Captcha_Generator	Multiple Random lines over the text	Random character color, Rotated characters, Different fonts	Realized using POST method with a user input. Automation of user input	Applying Erozion might be useful to remove the lines and bypass anti-segmentation. PHP
samir's captcha-generator		https://github.com/samirdjelal/captcha-rs	Random lines, Various geometric figures over the text, Random background colors, Noise,	Random captcha recognition methods such as addition of 2 numbers or text copying, character distortion, different colors	Constructor: CaptchaBuilder Properties: captcha.text captcha.base_img	Has multiple levels of captcha, 1-10 increasing difficulties useful for training with noise and filters. Rust

Library	Examples	URL Open Source	Anti-Segmentation Method	Anti-recognition Method	Automated Extraction and Saving	Note
svg-captcha-express		https://github.com/lazarofl/svg-captcha-express	Random lines of different colors, text color interflows with background	Multiple recognition methods, different character colors, random fonts	Functions create(...options) image()	Another SVG - - Generating library seems to be more accurate than the first one. Javascript
lepture captcha		https://github.com/lepture/captcha	Random dots and Lines, different background colors	Character distortion, different fonts, unification of letters	Constructor: ImageCaptcha Methods: write()	Easy to use can contain unified characters. Python

Capitolul 3

Instrumente de lucru si Inteligenta artificiala

3.1 Istoric Deep Learning

Inteligenta Artificiala este un domeniu al informaticii ce si-a luat nastere in 1940 si avea ca scop principal simularea cu o precizie cat mai mare a comportamentului uman. Astfel erau creati si studiati algoritmi de creare a unor programe speciale incat la interactiunea lor cu omul, persoana sa nu isi dea seama daca interactioneaza cu o alta persoana sau cu un robot. Progresul urmeaza in 1958 cand a fost creata prima Retea Neurala Artificiala de catre psihologul Frank Rosenblatt. Reteaua era denumita Perceptron si a fost destinata să modeleze modul în care creierul uman procesa datele vizuale și învață să recunoască obiectele. Alți cercetători au folosit de atunci ANN-uri similare pentru a studia cogniția umană.[6]. Mai apoi in 1967 inspirata din ideea Retelelor Neurale Artificiale (ANN) apare primul algoritm functional, care era in sine un Perceptron cu mai multe straturi, alaturi de algoritmul Stochastic Gradient Descent, cu ajutorul caruia erau calculate ponderile in Reteaua Neurala. De atunci pana in prezent se realizeaza multe descoperiri si ajungem cu o arie de studiu intreaga pe care se focuseaza multi oameni de stiinta, numita Vederea Artificiala.

3.2 Instrumente de lucru in Vederea Artificiala

Python este limbajul de programare de baza in rezolvarea problemelor de Inteligenta Artificiala, de altfel si algoritmi descriși in aceasta lucrare sunt implementati in Python. Datorita lizibilitatii codului dar si contributiei a unui grup enorm de oameni, au fost create si mai apoi optimizate multe librarii ce ofera simplitate in lucrul cu concepte complexe.

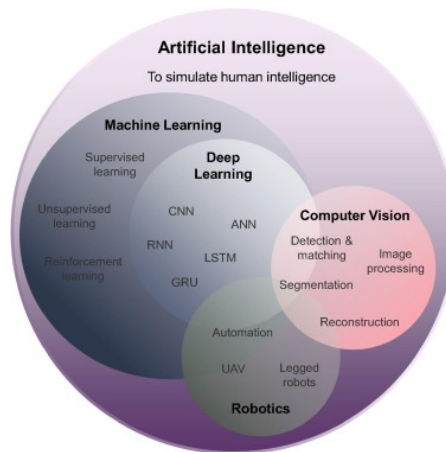


Figura 3.1: Ilustrare Deep Learning

- NumPy - este pachetul fundamental pentru calculul științific în Python. Este o bibliotecă Python care oferă obiecte și matrici multidimensionale, diverse obiecte derivate (cum ar fi matrice și array-uri mascate) și o varietate de operații rapide pe matrice, inclusiv matematice, logice, manipulare a formelor, sortare, selectare, I/O, transformate Fourier discrete, algebră liniară de bază, operații statistice de bază, simulare aleatorie și multe altele.[4]

Deasemenea un avantaj în utilizarea bibliotecii NumPy este că aceasta este implementată în limbajul de programare C/C++, astfel încât programele care utilizează NumPy rulează mult mai rapid decât cele care utilizează funcțiile și sintaxa obișnuită din Python.

Mai jos sunt descrise unele funcții care aparțin NumPy și au fost utilizate în construcția aplicației:

- `np.array(list)`: transformă listele de orice tip în array-uri numpy - util când avem nevoie să manipulăm imaginea în anumite feluri
- `np.transpose(array)`: permite modificarea/transpunerea dimensiunilor matricii
- `np.ones(size)` / `np.zeros(size)`: permite crearea matricilor care conțin pe toate pozițiile fie valoarea 1, fie valoarea 0 - utile pentru eliminarea sau adăugarea elementelor în imaginea de bază.
- OpenCV - Biblioteca open-source care include câteva sute de algoritmi pentru Computer Vision. Pachetul este divizat în mai multe secțiuni, partea pe care se focusează în lucrarea respectivă fiind **Image Processing**. Ca și în NumPy algoritmi sunt și ei implementați și optimizați în C/C++.

Mai jos sunt descrise unele funcții care aparțin OpenCV și au fost utilizate în construcția aplicației:

- `cv.imshow(image)` - funcție permite afișarea pe ecran a imaginii, este utilă în perioada debugging-ului.

- `cv.cvtColor(image, ...other parameters)` - functie care manipuleaza cu spatiul de culori al imaginii. Util cand e necesara eliminarea celor 3 dimensiuni RGB si se convertește imaginea in grayscale.
- `cv.bitwise_not(image)` - functie care inverseaza culorile imaginii, utila cand vrem sa separam obiectele de interes de fundal, si vrem sa fim consistenti in procesul dezvoltarii
- `cv.erode(image)` / `cv.dilate(image)` - functii extrem de importante utilizate in pre-procesarea imaginii, permit eliminarea sau accentuarea detaliilor in imagini.
- `cv.contours(image)` - functie cu ajutorul carora se extrag contururile intr-o imagine, este utila in procesul segmentarii si descoperirii coordonatelor caracterelor
- `cv.boundingRect(contours)` / `cv.rectangle(coordinates, image)` - functii cu ajutorul carora se extrag si se deseneaza sub forma de dreptunghi caracterele care au fost segmentate.
- PyTorch - Biblioteca care implementeaza algoritmi de deep learning, permite construirea retelelor neurale intr-un mod cat mai usor si lizibil pentru programator, este optimizata pentru lucrul cu tensorii folosind GPU-uri si CPU-uri. Mai jos atasez functionalitati utile care au fost folosite in constructia aplicatiei:
 - `torch.nn.Conv2d()` - Constructor pentru amplasarea in retea a unui strat convolutional
 - `torch.nn.Linear()` - Constructor pentru amplasarea in retea a unui strat linear
 - `torch.nn.ReLU()` - Rectified Linear Unit, Functie de activare pentru introducerea non-liniaritatii

3.3 Fluxul de lucru

Orice aplicatie bine implementata este construita in asa mod astfel incat sa restrictioneze, cat se poate de mult actiunile utilizatorului. Acesta insa, nu este scutit de automatizarea unor procese care la prima vedere ar parea complet legale, cum ar fi colectarea de date, sau chiar crearea a mai multor conturi pentru un singur website. Mai jos va fi descris procesul de gandire si implementare a automatizarii crearii conturilor pe o pagina web care contine sistemul de protectie de tip CAPTCHA.

Pentru inceput partea atacanta trebuie sa colecteze informatii despre cum arata pagina web, si in prealabil sa verifice daca CAPTCHA-urile prezente sunt asemanatoare intre ele sau apartin unor generatoare diferite. Odata ce avem destule informatii despre tinta respective putem incepe sa pregatim date pentru antrenarea sistemului de inteligenta artificiala. Daca generatorul de CAPTCHA este open-source acesta poate fi gasit usor, spre exemplu cu ajutorul functionalitatii "search image with google" care este foarte util in cautarea imaginilor cu sabloane asemanatoare, sau pe diverse platforme, cum ar fi github.

Daca insa generatorul nu este open-source, o varianta ar fi automatizarea procesului de extragere a imaginii din webpage, cum ar fi un script care reincarca pagina web, preia CAPTCHA-ul generat si il salveaza intr-un folder in local, apoi se adnoteaza manual setul de date, proces care ar putea sa dureze un pic mai mult. Cu setul de date pregatit, putem sa trecem la antrenarea sistemului de inteligenta artificiala. CAPTCHA-ul trebuie studiat, si depistate vulnerabilitati in masurile de protectie intreprinse, astfel incat spre final sa fie creata o functie care va putea separa caracterele (obiectele de interes) de imaginea de fundal. Functia respectiva o vom numi `preprocess_image()`. Odata ce avem separate caracterele de imaginea de fundal, incepem sa construim Reteaua Neurala care va fi antrenata. Din moment ce lucram cu imagini si trebuie sa depistam asemanari intre sabloane, evident ca vom utiliza straturile convolutionale, astfel vom avea o retea neurala convolutionala. In faza antrenarii, se va experimenta cu mai multi optimizatori, diversi parametri, si se va alege cea mai optima, se va salva intr-un folder in local. Dupa ce avem modelul salvat, urmeaza crearea scriptului de atac care parseaza pagina web tinta si completeaza toate casetele de tip input, cu valori prestabilite, sau generate random, extrage CAPTCHA, o rezolva si apasa pe butonul submit pentru completarea actiunii, dupa care reia actiunea.

Deoarece nu este etic sa cream boti astfel incat sa provocam atacuri de tip spam impotriva oricarui website existent, aplicatia principala va fi testata pe aplicatia secundara deasemenea implementata pentru aceasta lucrare.

3.4 Google Colab

De asemenea in cadrul acestei lucrari vom lucra in environmentul Google Colab. Colab este un produs de la Google Research. Colab permite oricui să scrie și să execute cod python arbitrar prin browser și este deosebit de potrivit pentru machine learning, analiza datelor și educație. Din punct de vedere tehnic, Colab este un serviciu de Jupyter-notebook găzduit care nu necesită configurare pentru a fi utilizat, oferind în același timp acces gratuit la resursele de calcul, inclusiv GPU-urile, si este gratuit.

Capitolul 4

Implementarea Aplicatiei, Experimente Realizate

4.1 Prezentare - Seturi de date CAPTCHA

Intrucat stim deja cum va arata aplicatia tinta, vom trece direct la extragerea si generarea dataset-urilor precum si prezentarea in detaliu a acestora.

SVG-Captcha

SVG-Captcha este o librerie CAPTCHA open-source, implementata pentru javascript, si dupa cum sugereaza numele, CAPTCHA-ul respectiv in momentul generarii este encodat in format SVG.

Ce este formatul SVG? - Scalable Vector Graphics (SVG) este un format de fișier vector web-friendly. Spre deosebire de fișierele raster bazate pe pixeli, cum ar fi JPEG, fișierele vectoriale stochează imagini prin formule matematice bazate pe puncte și linii pe o grilă.[9]

Codul utilizat pentru generare este urmatorul:

```
1 var svgCaptcha = require('svg-captcha');
2 const fs = require('fs')
3
4 for (let i = 0; i < 10000; i++) {
5   var captcha = svgCaptcha.create();
6   fs.writeFile(`../captchas/svg_captcha/images/
7     ${captcha.text}.txt`, captcha.data, (err) => {
8     if (err) throw err
9   })
10 }
```

Listing 4.1: SVG-CAPTCHA generation code

Se importa modulele `svg-captcha` (pentru CAPTCHA) si `fs` (pentru manipularea cu fisierele), dupa care se itereaza de un numar prestabilit de ori, iar la fiecare iteratie, se apeleaza functia `svgCaptcha.create()`, dupa care salvam obiectul generat intr-un folder sub forma de fisier `txt`, iar numele fisierului va fi insusi eticheta / textul din imagine. Odata ce avem imaginile stocate intr-un folder, acestea trebuie transformate in format `png` sau `jpg`, pentru a lucra cu ele sub forma de matrici. Pentru a face asta folosim doua tool-uri implementate pentru python, software-ul `ImageMagick` (gama de software gratuită, open-source, utilizată pentru editarea și manipularea imaginilor digitale. Poate fi folosit pentru a crea, edita, compune sau converti imagini bitmap și acceptă o gamă largă de formate de fișiere, inclusiv JPEG, PNG, GIF, TIFF și PDF. [5]) si biblioteca `Wand` (bibliotecă de funcții `ctypes` care realizeaza conexiunea simplă între `ImageMagick` pentru Python.).

```

1 from wand.image import Image
2 for idx in range(len(image_files)):
3     fn = image_files[idx]
4     ny = Image(filename = fn)
5     fn = fn[:-4]
6     ny_convert = ny.convert('png')
7     ny_convert.save(filename = f'./{formatted_svg_path}/{fn.rsplit("/", 1)
    [-1]}.png')
```

Listing 4.2: Cod conversie SVG-CAPTCHA

Se itereaza prin folderul cu imagini, la fiecare iteratie se creaza un obiect de tip `wand.Image` cu fisierul propriu, se realizeaza conversia, dupa care se salveaza imaginea in formatul dorit in alt folder. Odata ce avem imaginile in formatul potrivit, dorim sa le mutam in environmentul cu care o sa lucram in continuare, google colab. Deoarece google colab este o platforma online, informatiile cu care lucram vor fi stocate in cloud. Pentru a nu consuma mult spatiu si pentru a fi cat mai concisi, vom transporta imaginile CAPTCHA sub forma de fisier `.npy`.

Ce reprezinta stocarea in Cloud? - Stocarea în cloud este un model de stocare a datelor pe computer în care datele digitale sunt stocate în pool-uri logice, despre care se spune că se află pe "cloud". Spațiul de stocare fizic se întinde pe mai multe servere, iar mediul fizic este de obicei deținut și gestionat de o companie de găzduire.

```

1 captcha_array = []
2 captcha_labels = []
3
4 for captcha_file in captcha_files:
5     image = cv.imread(captcha_file)
6     captcha_array.append(image)
7     captcha_labels.append(captcha_file[captcha_file.rfind('/')+1:]
8     .split('.')[0])
9
```

```

10 np.save("svg_formated_test_array",captcha_array)
11 np.save("svg_formated_test_labels",captcha_labels)

```

Listing 4.3: Conversie SVG in PNG



Figura 4.1: Exemple SVG CAPTCHA

4.1.1 Lepture-CAPTCHA

Lepture-CAPTCHA este libraria default CAPTCHA implementata pentru python, si are constructori pentru generare de CAPTCHA atat audio, cat si sub forma de imagini.

```

1 from captcha.image import ImageCaptcha
2 import random
3 import string
4
5 image = ImageCaptcha()
6 for i in range(1,11,1):
7     label = ''.join(random.choices(string.ascii_lowercase + '0123456789',
8                                   k=4))
9     data = image.generate(label)
10    image.write(label, f'./captchas/lepture_captcha/images/{label}.png')

```

Listing 4.4: Lepture-CAPTCHA generation code

Label-ul / Textul din imagine este generat cu ajutorul libreriei *random* implementate pentru python, si functia `random.choices()` care primeste ca parametri o structura de tip lista din care se aleg caracterele, iar `k` este parametrul ce denota lungimea listei returnate, prin urmare in urma rularii scriptului respectiv vom obtine string-uri a cate 4 caractere ce contin litere mari, litere mici si cifre. Acelasi principiu ca la SVG, doar ca de data asta primim imaginile in format .png direct, deci procesul de creare este mult mai usor, urmeaza sa stocam imaginile generate intr-un numpy array, si sa le transportam in memoria storage din cloud.



Figura 4.2: Exemple Lepture-CAPTCHA

4.1.2 Samir-CAPTCHA-rs

Samir's CAPTCHA-rs este o librerie de generare CAPTCHA implementata pentru limbajul de programare Rust

```
1 let captcha = CaptchaBuilder::new()
2     .length(length)
3     .width(200)
4     .height(150)
5     .dark_mode(mode)
6     .complexity(complexity) // min: 1, max: 10
7     .compression(compression) // min: 1, max: 99
8     .build();
9
10 image_file.write_all(captcha.to_base64().as_bytes()).expect("unable to
    write data to image file");
```

Listing 4.5: Samir-CAPTCHA-rs generation code

Odata ce cream obiectul de tip `CaptchaBuilder`, aplicam functiile care construiesc imaginea finala:

length - Reprezinta lungimea CAPTCHA-ului generat

width / height - Dimensiunile Imaginii generate

dark_mode - In dependenta de valoarea booleana true sau false va genera imaginea cu fundal negru sau alb

complexity - Reprezinta nivelul de distorsiune aplicat CAPTCHA-ului, intr-o gama de nivele de la 1 la 10

compression - Reprezinta nivelul de comprimare a zgomotului din Imagine, intr-o gama de nivele de la 1 la 99

Imaginea Captcha este encodata in format base64 si este stocata in fisiere de tip txt. Urmatorul pas este sa transformam imaginea din base64 in format obisnuit - png sau jpg/jpeg. Pentru a converti imaginea o sa folosim urmatoarea functie:

```
1 def base64_to_image(base64_string):
2     def pil_to_cv(image):
3         if image.mode != 'RGB':
4             image = image.convert('RGB')
5         cv_image = cv.cvtColor(np.array(image), cv.COLOR_RGB2BGR)
6         return cv_image
7     if base64_string.startswith('data:image'):
8         base64_string = base64_string.split(',')[1]
9     image_data = base64.b64decode(base64_string)
10    image = Image.open(BytesIO(image_data))
11    return pil_to_cv(image)
```

Listing 4.6: Samir-CAPTCHA-rs generation code

Funcția respectivă utilizează pachetul `pillow`, implementat pentru manipularea imaginilor, care oferă funcționalități asemănătoare cu pachetul `OpenCV`, și bibliotecă `base64`, datorită căreia putem decoda imaginea obținută în faza generării. Astfel, se decodează imaginea, și se stochează într-o variabilă ca obiect de tip `pillow.Image`, după care o convertim în `numpy` array, pentru a putea efectua operații din biblioteca `OpenCV` și returnăm imaginea finală.

4.2 Preprocesarea imaginilor

Preprocesarea imaginii are o importanță enormă în perspectivă, și rezultatele finale, de aceea în mod ideal vrem să separăm perfect elementele de distorsiune din imagine, astfel încât să rămânem doar cu zonele de interes computate corect. Dat fiind faptul că avem mai multe generatoare de captcha, evident că va trebui reimplementată funcția de preprocesare pentru toate dintre ele. În cadrul experimentelor a fost implementată preprocesarea imaginilor pentru cele 3 CAPTCHA, descrise mai sus.

Înainte de a discuta despre funcția respectivă, voi prezenta operațiile morfologice sarcina cărora este să proceseze imaginea bazat pe forma obiectului de interes. Metodele care stau la baza acestor operații sunt eroziunea și dilatarea imaginii.

cv.dilate - Operația constă în convoluția unei imagini A cu un nucleu (B), care poate avea orice formă sau dimensiune, de obicei un pătrat sau un cerc. Nucleul B are un punct de ancorare definit, fiind de obicei centrul nucleului. Pe măsură ce nucleul B este scanat peste imagine, se calculează valoarea maximă a pixelului suprapus pe B și se înlocuiește pixelul imaginii din poziția punctului de ancorare cu acea valoare maximă. Operația respectivă intensifică regiunile luminoase ale imaginii exemplu fiind imaginea următoare: [2]



Figura 4.3: Operația de dilatare, Imagini preluate din[2]

cv.erode - Operația respectivă este inversa dilatării astfel încât calculează un minim local pe aria nucleului dat. Pe măsură ce nucleul B este scanat peste imagine, calculăm valoarea minimă a pixelului suprapus de B și înlocuim pixelul imaginii de sub punctul de

ancorare cu acea valoare minima. Impactul se poate observa in urmatoarea imagine: [2]



Figura 4.4: Operatia de eroziune, Imagini preluate din[2]

Inceput cu SVG-Captcha, pentru ca are numarul minim de zgomot, fiind o linie curba ce trece prin caractere, scopul nostru este sa eliminam linia si sa identificam zonele in care se afla caracterele. Mai jos sunt atasate codul cu functia optimizata, si imaginea asupra careia se va prezenta algoritmul din lucrare

```
1 def preprocess_image(img):
2     gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
3     thresh = cv.threshold(gray,1, 255, cv.THRESH_BINARY_INV|cv.THRESH_OTSU)
4     [1]
5     kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))
6     morph_img = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel)
7     kernel = np.ones((3, 3), np.uint8)
8     morph_img = cv.erode(morph_img, kernel)
9     kernel = np.ones((2, 2), np.uint8)
10    morph_img = cv.dilate(morph_img, kernel)
11    return thresh
```

Listing 4.7: SVG-CAPTCHA image-preprocessing



Figura 4.5: Exemplu SVG CAPTCHA

Pentru inceput, se converteste imaginea in format grayscale cu ajutorul functiei `cv.cvtColor()`, deoarece multe functii se asteapta sa primeasca ca parametru de intrare o imagine in format grayscale, dar si reducerea dimensionalitatii, ofera o simplitate sporita in lucrul cu imaginile.

Dupa ce am convertit imaginea in grayscale, se observa ca fiecare CAPTCHA, din biblioteca SVG are fundalul alb, iar caracterele sunt negre. Pentru viitor, vom seta mereu pixelii caracterelor, la valoarea maxima, iar pixelii fundalului la valoarea minima, astfel in cazul de fata se aplica functia `cv.threshold(gray,1, 255, cv.THRESH_BINARY_INV or cv.THRESH_OTSU)[1]` unde primul argument este imaginea sursă, care ar trebui să

fie o imagine în grayscale, al doilea argument este valoarea pragului care este folosită pentru a clasifica valorile pixelilor, al treilea argument este valoarea maximă care este atribuită valorilor pixelilor care depășesc pragul, iar al patrulea argument reprezintă tipul de prag aplicat imaginii. În cazul nostru se folosește `cv.THRESH_BINARY_INV` care atribuie valoarea maximă pixelilor care sunt sub prag, și valoarea minimă pixelilor care sunt peste prag, combinat cu `THRESH_OTSU` care utilizează algoritmul Otsu pentru stabilirea exactă a unui prag optim, astfel se obține o imagine binară cu valoarea maximă atribuită pixelilor din prim-plan și valoarea minimă atribuită pixelilor de fundal. [2]



Figura 4.6: Aplicarea `cv.threshold`

Observăm că în imaginea rezultată, regiunea aflată în interiorul caracterelor de asemenea este setată pe valoarea minimă, ceea ce ar putea provoca inconveniente, deoarece în momentul eroziunii odată cu eliminarea liniei curbe riscăm să pierdem detalii importante din caracter, fără posibilitatea de a le restabili, de aceea ne vom concentra în umplerea spațiului respectiv cu pixeli de culoare albă.

După aplicarea `cv.threshold`, se construiește cu ajutorul funcției `cv.getStructuringElement()` nucleul care va fi pasat în continuare funcțiilor precum `cv.erode()` sau `cv.dilate()`. Funcția primește ca parametri, forma nucleului, dimensiunea și punctul de ancorare care este centrul nucleului în mod implicit. În cazul de față vom avea un nucleu de formă dreptunghică (`cv.MORPH_RECT`) cu dimensiunile 5 X 5, deci un pătrat.

După construcția nucleului, vom utiliza funcția `cv.morphologyEx()` care combină operațiile de dilatare și eroziune. Funcția primește ca parametri imaginea, asupra căreia se produc modificări, în cazul nostru imaginea obținută în urma aplicării `cv.threshold()`, tipul de morfologie aplicată, unde `cv.MORPH_CLOSE` care reprezintă dilatarea urmată de eroziune, și este utilă când vrem să închidem găuri mici din interiorul obiectului de interes, și nucleul pe care îl aplicăm. După aplicarea funcției obținem următoarea imagine rezultat:



Figura 4.7: Aplicarea `cv.morphologyEx()`

După cum putem observa, regiunile corespunzătoare caracterelor sunt pline, deci putem să eliminăm linia cu ajutorul funcției `cv.erode()` fără riscul de a elimina părți importante din caracter.

Deoarece nu avem nevoie de un nucleu complex vom utiliza functia `np.ones()` care primeste ca parametru un tuplu ce reprezinta dimensiunile unei matrici, si un tip de date corespunzator, si returneaza o matrice care contine pe fiecare pozitie valoarea 1. Cream un nucleu de dimensiune 3x3 dupa care aplicam eroziunea si obtinem urmatoarea imagine:



Figura 4.8: Aplicarea `cv.erode()`

Dupa aplicarea eroziunii, dimensiunile caracterelor sunt un pic mai mici decat in imaginea initiala, de aceea vom aplica o dilatare cu un nucleu mai mic asupra imaginii obtinute, ca sa putem obtine contururile cat mai precise a zonelor ce contin caractere:



Figura 4.9: Aplicarea `cv.dilate()`

Mai jos sunt atasate procesele prin care au trecut seturile de date Samir-CAPTCHA-rs si Lepture-CAPTCHA:

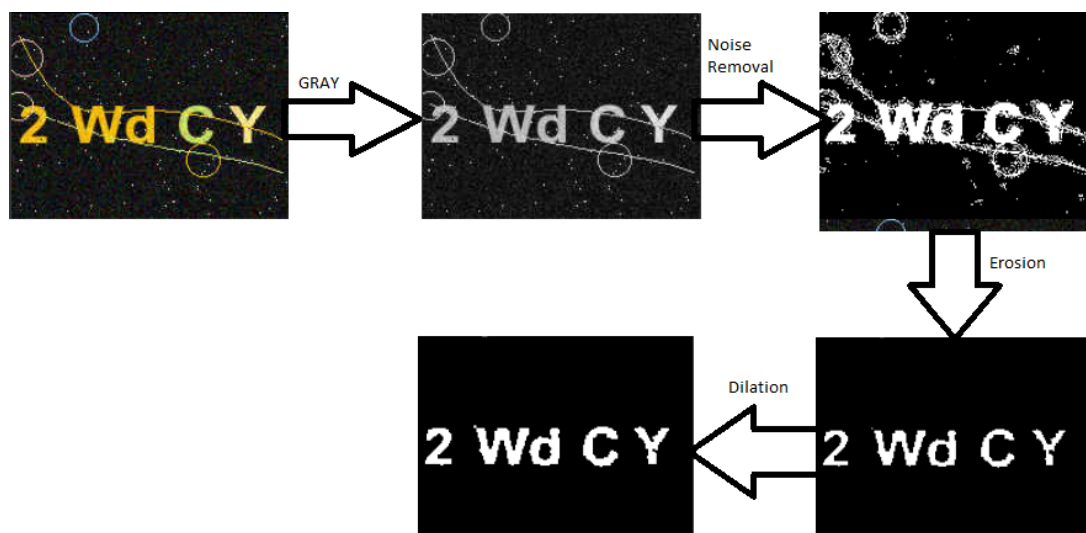


Figura 4.10: Preprocesarea Setului Samir-CAPTCHA-rs

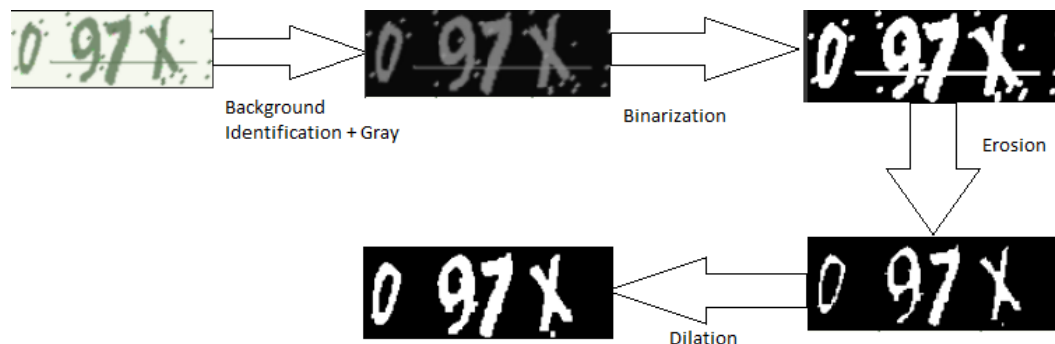


Figura 4.11: Preprocesarea Setului Lepture-CAPTCHA

4.2.1 Preprocesare - Additional

Pentru celelalte doua seturi, observam ca caracterele din Imagine nu mai sunt vulnerabile in cazul in care folosim functia erode pe intreaga imagine, prin urmare nu mai trebuie sa aplicam trucul cu MORPH_CLOSE utilizat pentru SVG.

In schimb ambele imagini contin mai multe elemente de zgomot. In cazul respectiv, incepem prin identificarea fundalului. Odata ce separam fundalul de caractere, se transforma imaginea in format grayscale si se aplica functia *cv.fastNlMeansDenoising()* ce are la baza algoritmul de Non-local means Denoising [3] ce primeste ca parametri imaginea respectiva, destinatia, parametrul *h* - reguleaza filtru (Valoarea *h* mare elimina perfect zgomotul, dar elimina și detaliile imaginii, valoarea *h* mai mică păstrează detaliile, dar păstrează și zgomotul), dimensiunea în pixeli a patch-ului șablonului care este utilizat pentru a calcula ponderile si dimensiunea în pixeli a ferestrei care este utilizată pentru a calcula media ponderată pentru un anumit pixel.[2]

4.2.2 Analiza spatiului de etichete

Luand in considerare ca fiecare imagine CAPTCHA este formata din combinarea mai multor caractere, iar alfabetul este format din litere majuscule, minuscule si cifre, avem un total de 62 de caractere distincte. Fiecare caracter este generat aleator deci obtinem spatiul urmator de valori pentru etichetele CAPTCHA-ului:

$$S \leq 62^n$$

unde *n* este numarul maxim de caractere pe care il poate avea un CAPTCHA. Evident ca vrem sa reducem numarul de etichete pe care ar trebui sa le returneze sistemul de inteligenta artificiala. Observam ca in imaginea noastra, caracterele sunt separate intre ele ceea ce ar face mai usor extragerea acestora, de aceea metoda pe care incercam sa o implementam este extragerea si antrenarea modelului pe fiecare caracter in parte.

Dupa optimizarea functiei de preprocesare, o vom aplica intregului set de date si vom

trece la pasul urmasor - obtinerea conturului caracterelor din imagine . Functionalitatea mentionata poate fi urmarita in urmatoarea secventa de cod:

```
1 def extract_letters(image):
2     def cmp(x):
3         return x[1]
4
5     def expand2square(pil_img, background_color):
6         width, height = pil_img.size
7         if width == height:
8             return pil_img
9         elif width > height:
10            result = Image.new(pil_img.mode, (width, width), background_color)
11            result.paste(pil_img, (0, (width - height) // 2))
12            return result
13        else:
14            result = Image.new(pil_img.mode, (height, height), background_color
15                               )
16            result.paste(pil_img, ((height - width) // 2, 0))
17            return result
18
19    def contours_to_rectangles(contours):
20        letter_image_regions = []
21        for contour in contours:
22            (x,y,w,h) = cv2.boundingRect(contour)
23            if w*h < 100 :
24                continue
25            if w/h >= 2:
26                thirdof_width = int(w/3)
27                letter_image_regions.append((x,y,thirdof_width,h))
28                letter_image_regions.append((x+thirdof_width,y,thirdof_width,h))
29                letter_image_regions.append((x+(2*thirdof_width),y,thirdof_width,
30                h))
31            elif w/h > 1.6:
32                half_width = int(w/2)
33                letter_image_regions.append((x,y,half_width,h))
34                letter_image_regions.append((x+half_width,y,half_width,h))
35
36            letter_image_regions.append((x,y,w,h))
37
38        return letter_image_regions
39
40    image_without_noise = preprocess_image(image)
41    contours = cv.findContours(image_without_noise.copy(),cv.RETR_EXTERNAL,cv
    .CHAIN_APPROX_SIMPLE)[0]
42
43    letter_image_regions = contours_to_rectangles(contours)
44    letter_image_regions.sort(key=cmp)
```

```

42
43 coppied_image = image.copy()
44 characters = []
45
46 for rect in letter_image_regions[:4]:
47     x,y,w,h = rect
48     ch_cropped = image[y:y+h,x:x+w]
49     pillowed = Image.fromarray(np.uint8(ch_cropped))
50     size=(50,50)
51     pillowed = expand2square(pillowed, (255, 255, 255)).resize(size)
52     processed_ch = np.array(pillowed)
53     characters.append(processed_ch)
54 return characters

```

Listing 4.8: Cod extragere caractere din imagine

Vor fi implementate 3 functii ajutatoare:

- `cmp(x)` - In timpul implementarii am observat ca la momentul extragerii caracterelor din imagine, acestea sunt returnate intr-o lista, in ordine aleatoare. Noi in schimb le vom sorta dupa cum sunt aranjate de l-a stanga la dreapta, adica dupa coordonata coloanelor
- `expand2square()` - functie preluata din `imaginglib.py` [**expand2square**] adauga pixeli imaginii, astfel incat sa aiba forma patratica
- `contours_to_rectangles()` - functie care primeste ca parametru o lista cu contururile depistate in imagine si le transforma in dreptunghiuri care le marginesc

Astfel algoritmul executa in felul urmator:

- Se preproceseaza imaginea
- Se preiau contururile cu ajutorul functiei `cv.findContours()` din biblioteca OpenCV care primeste ca parametru o imagine, metoda de extragere, care in cazul nostru este `cv.RETR_EXTERNAL` si preia doar contururile externe, si metoda de aproximare, care in cazul nostru este `cv.CHAIN_APPROX_SIMPLE` care comprimă segmentele orizontale, verticale și diagonale și lasă doar punctele finale ale acestora.
- Se transforma contururile obtinute in dreptunghiuri, dupa se cropeaza imaginea conform coordonatelor dreptunghiurilor obtinute, caracterului obtinut i se aplica functia `expand2square`, dupa care imaginii finale i se aplica o redimensionare catre 50 x 50
- La final se returneaza o lista cu toate caracterele obtinute in urma rularii algoritmului.

Odata ce am extras caracterele din toate imaginile, se vor crea foldere speciale pentru fiecare caracter in care acestea vor fi stocate, pastram imaginile la etichetele respective, si pregatim datele pentru antrenarea modelului.



Figura 4.12: Extragerea contururilor

4.3 Constructia Retei Neurale Artificiale

Odata ce avem toate datele pregatite, vom trece la constructia retelei neurale artificiale, si dat fiind faptul ca lucram cu imagini, vom utiliza straturi convolutionale. Straturile convolutionale sunt proiectate pentru a identifica pattern-uri asemanatoare in imaginile cu aceleasi etichete. Astfel Reteaua construita pe parcursul implementarii aplicatiei este urmatoarea:

```

1  class ConvNet(nn.Module):
2  def __init__(self, out_size):
3      super().__init__()
4      self.conv1 = nn.Conv2d(in_channels=3,out_channels=8,kernel_size=3,
5      stride=1)
6      self.conv2 = nn.Conv2d(in_channels=8,out_channels=32,kernel_size=3,
7      stride=1)
8      #infeatures = (imwidth - kernel_size - kernel_size) ^ 2 deoarece
9      imwidth = imheight
10     self.fc1=nn.Linear(in_features=11*11*32,out_features=128)
11     self.fc2=nn.Linear(in_features=128,out_features=64)
12     self.fc3=nn.Linear(in_features=64,out_features=out_size)
13     self.activation_fn=nn.ReLU()
14     self.pool1 = nn.MaxPool2d(kernel_size=2,stride=2)
15
16 def forward(self, x):
17     x = self.activation_fn(self.conv1(x))
18     x = self.pool1(x)
19     x = self.activation_fn(self.conv2(x))
20     x = self.pool1(x)
21     x = x.view(x.shape[0],-1)
22     x = self.activation_fn(self.fc1(x))
23     x = self.activation_fn(self.fc2(x))
24     x = self.fc3(x)
25     return x

```

```
25 net = ConvNet(out_size=num_cls)
```

Listing 4.9: Cod Retea Neurala

Arhitectura se aranjeaza in felul urmator:

Imaginile de dimensiune 50 X 50 in format RGB (contin 3 canale) sunt trecute prin primul strat convolutional, scanate de un nucleu de dimensiuni 3 X 3 cu pas de o celula, transformate in tensori de dimensiune 48 x 48 a cate 8 canale, rezultatul obtinut mai apoi este trecut prin functia de activare ReLU.

Peste tensorii obtinuti se aplica operatia de MaxPooling cu pasul 2, astfel obtinem tensori de dimensiune 24 x 24 a cate 8 canale

Se aplica cel de-al doilea strat convolutional scanati de un nucleu de dimensiuni 3 x 3 cu pas de o celula, si se obtin tensori de marimea 22 x 22 rezultatul obtinut mai apoi iarasi este trecut prin aceeasi secventa de operatii: ReLU + MaxPooling cu pasul 2.

Dupa aplicarea straturilor convolutionale vom avea tensori de dimensiunea 11 x 11 a cate 32 de canale. Se transforma rezultatul in tensori unidimensionali folosind functia `x.view(x.shape[0], -1)`, dupa care se trece prin 3 straturi lineare complete, in urma carora se obtin predictiile respective, unde fiecare nod din predictiile finale reprezinta scorul imaginii fata de fiecare caracter in parte. Ca sa obtinem caracterul predictiei aferente, preluam pozitia pe care se afla cel mai mare scor in predictiile generate care de altfel este si pozitia caracterului in lista cu toate etichetele.

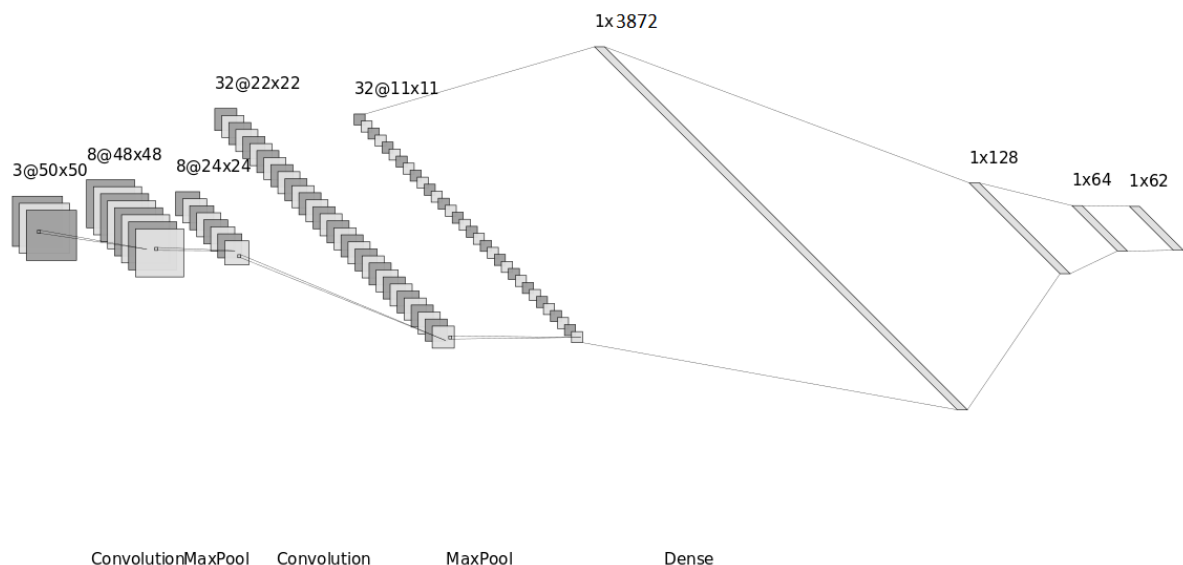


Figura 4.13: Imagine Retea Neurala

Codul pentru antrenarea Retelei este urmatorul:

```
1 num_cls = len(characters_dict)
2 lr = 0.001
```

```

3 momentum = 0.7
4 num_epochs = 10
5 batch_size = 64
6
7 # create a SGD optimizer
8 optimizer = torch.optim.SGD(net.parameters(), lr=lr, momentum=momentum)
9
10 # set up loss function
11 loss_criterion = nn.CrossEntropyLoss()
12
13 train_losses = []
14 train_accuracies = []
15 val_losses = []
16 val_accuracies = []
17 for epoch in range(1, num_epochs+1):
18     train_loss, train_accuracy = train_epoch(net, train_dataloader,
19                                             loss_criterion, optimizer, device)
20     val_loss, val_accuracy = eval_epoch(net, validation_dataloader,
21                                       loss_criterion, device)
22     train_losses.append(train_loss)
23     val_losses.append(val_loss)
24     train_accuracies.append(train_accuracy)
25     val_accuracies.append(val_accuracy)
26     print('\nEpoch %d'%(epoch))
27     print('train loss: %10.8f, accuracy: %10.8f'%(train_loss, train_accuracy)
28           )
29     print('val loss: %10.8f, accuracy: %10.8f'%(val_loss, val_accuracy))

```

Listing 4.10: Cod Antrenare Retea Neurala

Pentru optimizarea ponderilor se utilizeaza metodologia SGD (Stochastic Gradient Descent) cu ajutorului criteriului pentru pierdere `CrossEntropyLoss()`. Se foloseste momentumul = 0.7 pentru a depasi portiumile plate unde ajungem intr-un minim local cu scopul de a descoperi mai adanc minimul global, de altfel, cel care ne intereseaza pe noi.

Stochastic Gradient Descent este o metodă iterativă pentru optimizarea unei funcții obiective diferențiabile sau subdiferențiabile. Poate fi privită ca o aproximare stocastică a optimizării coborârii gradientului, deoarece înlocuiește gradientul real (calculat din întregul set de date) cu o estimare a acestuia (calculată dintr-un subset de date selectat aleatoriu). În special în problemele de optimizare dimensională, acest lucru reduce sarcina de calcul foarte mare, realizând iterații mai rapide în schimbul unei rate de convergență mai scăzute.[sgd_citation]

Cross Entropy Loss - Fiecare probabilitate de clasă prezenta este comparată cu rezultatul dorit de clasă reală 0 sau 1 și se calculează un scor/pierdere care penalizează probabilitatea în funcție de cât de departe este aceasta de valoarea reală așteptată. Penalizarea este de natură logaritmică, dând un scor mare pentru diferențe mari aproape

de 1 și un scor mic pentru diferențe mici care tind spre 0.[[cross_entropy_citation](#)]

Intrucat functia de extragere a caracterelor functioneaza datorita unei euristici, afe-rente amplasarii caracterelor in imagine, ea este utila doar pentru spargerea CAPTCHA-urilor asemanatoare cu SVG-CAPTCHA si Samir-CAPTCHA-rs. Functia nu va extrage corect caracterele din imaginile Lepture-CAPTCHA, prin urmare vom antrena si testa pentru celelalte doua seturi de date, revenind apoi cu concluziile si impeditiunile intam-pinate.

4.3.1 Plotarea Datelor

In urma antrenarii rețelei respective pentru 10 epoci obținem următoarele rezultate:

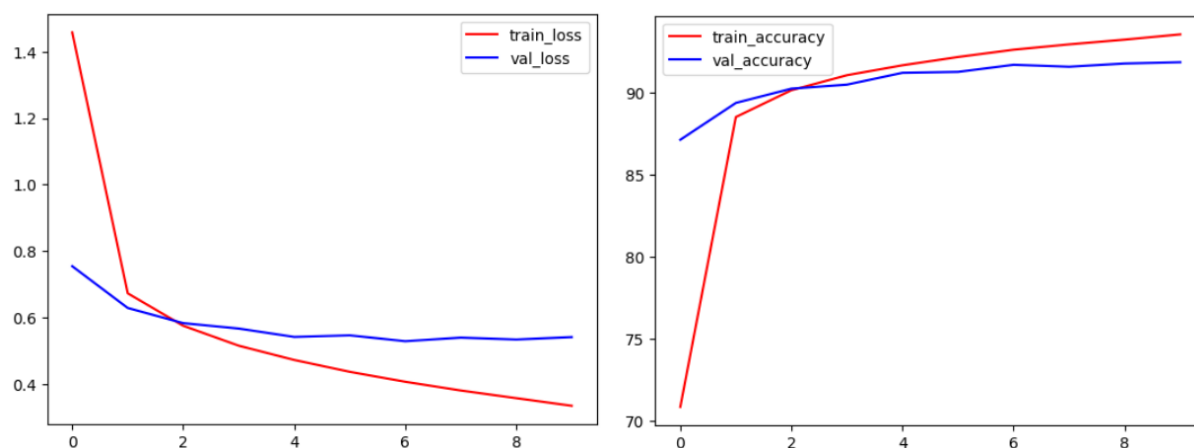


Figura 4.14: SVG CAPTCHA plot

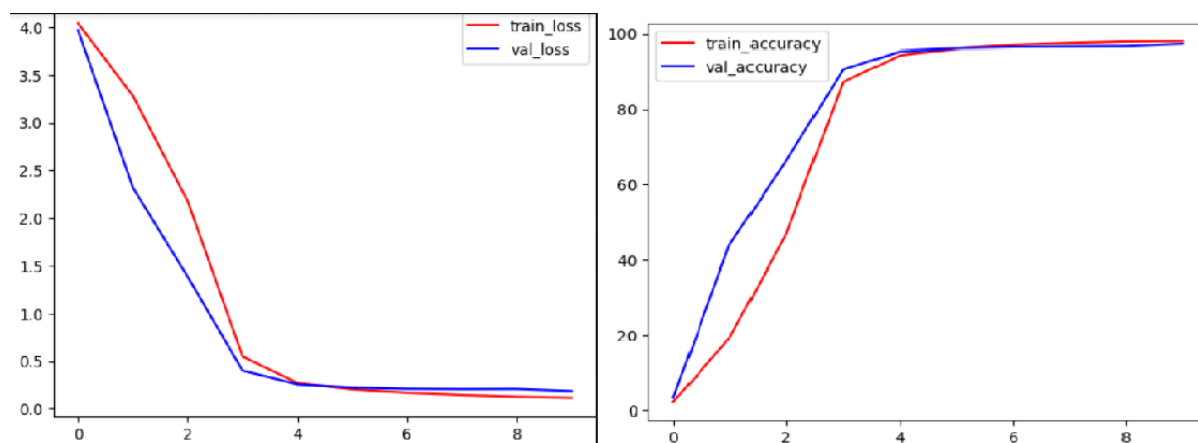


Figura 4.15: Samir-CAPTCHA-rs plot

Reteaua atinge apogeul de aproximativ 92% pentru datele de validare, ceea ce este un rezultat destul de bun. Este timpul sa experimentam cu Imaginile CAPTCHA propriuzise pe care le avem pregatite pentru testare.

Ultima functie care trebuie implementata pentru a finaliza clasa noastra se numeste *solve_captcha()* iar codul ei este prezentat in secventa urmatoare:

```
1 def solve_captcha(captcha_image):
2     characters = extract_letters(captcha_image)
3     for i in range(len(characters)):
4         characters[i] = np.transpose(characters[i], (2, 0, 1))
5     characters = torch.from_numpy(np.array(characters, dtype=np.float32))
6     label = ''
7     predictions = net.forward(characters)
8     for pred in predictions:
9         let_idx = torch.argmax(pred)
10        label+=keys[let_idx]
11    return label
12
13 predicted_labels = []
14 for captcha in test_captcha_array:
15     label = solve_captcha(captcha)
16     predicted_labels.append(label)
17
18 print(accuracy_score(test_captcha_labels, predicted_labels))
```

Listing 4.11: Cod Spargere Captcha

Ideea este simpla. Se preia CAPTCHA-ul ce trebuie rezolvat, si se segmenteaza pe caractere. Caracterele segmentate se trec prin retea si se concateneaza predictiile obtinute intr-o eticheta totala, care va reprezenta textul aflat in CAPTCHA

In urma procesarii a 1000 de imagini CAPTCHA obtinem o acuratete de aproximativ 80% pentru Setul de date SVG-CAPTCHA, si aproximativ 90% pentru setul de date Samir-CAPTCHA-rs, ceea ce era un rezultat asteptabil.

Acum ca avem toate functionalitatile si algoritmi pregatiti, se va construi o clasa care se va numi CaptchaBreaker care le v-a incorpora in intregime, astfel incat sa ne fie mai usor sa lucram cu sistemul. Odata creata clasa, instantiem un obiect si implementam functiile respective pentru tipul de CAPTCHA dorit, dupa care antrenam reseaua neurala. La final salvam modelul cu ajutorul libreriei pickle, pachet care implementeaza protocoale binare pentru serializarea si deserializarea unei structuri de obiecte Python.[11]

```
1 breaker_filename = "breaker.pkl"
2 with open(breaker_filename, 'wb') as f:
3     pickle.dump(breaker, f)
4
5 with open(breaker_filename, 'rb') as f:
6     breaker = pickle.load(f)
```

Listing 4.12: Cod Spargere Captcha

4.3.2 Aplicatia Secundara

Aplicatia secundara a fost creata in scopul testarii si demonstrarii fluxului de lucru a unei persoane care intentioneaza sa faca atacuri de spam asupra unei pagini web. Astfel este o aplicatie web realizata cu ajutorul Framework-ului next.js, si ilustreaza un form de inregistrare ce contine o CAPTCHA.

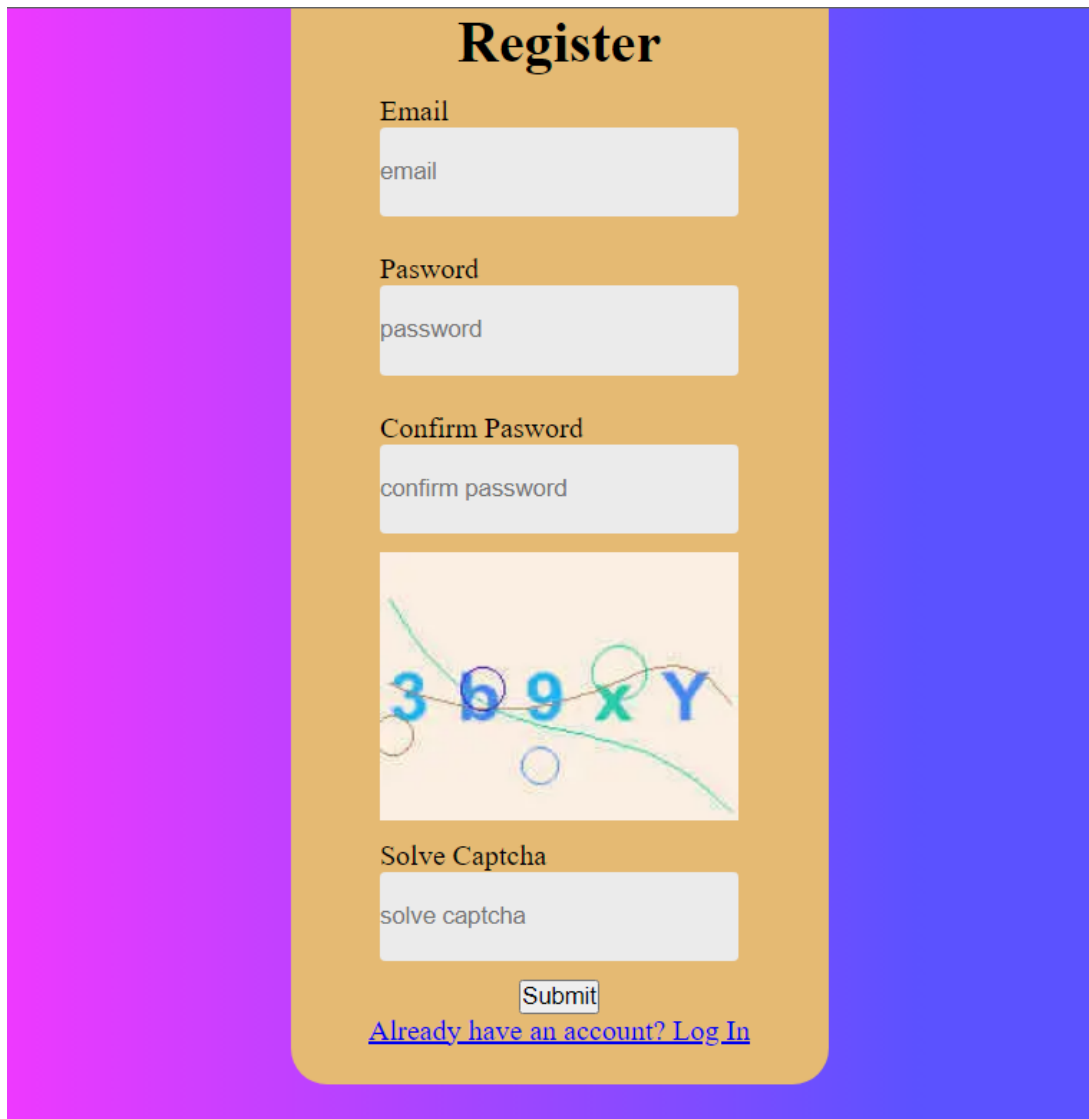
The image shows a web registration form titled "Register" in a bold, black, serif font. The form is set against a light orange background and is framed by a purple-to-blue gradient border. It contains four input fields: "Email" with placeholder text "email", "Pasword" with placeholder text "password", "Confirm Pasword" with placeholder text "confirm password", and "Solve Captcha" with placeholder text "solve captcha". Below the "Solve Captcha" field is a "Submit" button. At the bottom of the form, there is a link that reads "Already have an account? Log In". The CAPTCHA image itself displays the characters "3 6 9 x Y" in various colors and sizes, overlaid with a green wavy line and several blue circles.

Figura 4.16: Captura de ecran a aplicatiei secundare

Atacatorul trebuie sa insereze informatii prestabilite sau generate random in casutele cu email, parola si confirmare a parolii, si respectiv sa insereze textul continut in captcha. Pentru a realiza acest lucru trebuie identificate in codul html etichetele aferente casutelor, creat un script care obtine datele ce vor fi inserate, si utilizat modelul pentru a sparge CAPTCHA-ul respectiv.

4.3.3 Script-ul de atac

Pentru ca scopul nostru final este sa demonstram realizarea cu succes a unui atac care sparge CAPTCHA-ul vom omite celelalte masuri de siguranta de conduita buna care trebuie sa persiste in fiecare aplicatie web.

```
1 import requests
2 from bs4 import BeautifulSoup
3 from io import BytesIO
4 from selenium import webdriver
5 import random
6 import string
7
8 def generate_random_email():
9     domains = ['gmail.com', 'yahoo.com', 'hotmail.com', 'example.com']
10    domain = random.choice(domains)
11    username = ''.join(random.choices(string.ascii_letters + string.digits,
12                                     k=10))
13    email = f'{username}@{domain}'
14    return email
15
16 def generate_random_password():
17    password = ''.join(random.choices(string.ascii_letters + string.digits, k
18                                     =10))
19    return password
20
21 # Generate a random email address
22 random_email = generate_random_email()
23 random_password = generate_random_password()
24
25 webpage = r"http://localhost:3000/register" #
26
27 driver = webdriver.Chrome()
28 driver.get(webpage)
29
30 sbbox = driver.find_element("name", "email")
31 sbbox.send_keys(random_email)
32
33 sbbox = driver.find_element("name", "password")
34 sbbox.send_keys(random_password)
35
36 sbbox = driver.find_element("name", "confirmPassword")
37 sbbox.send_keys(random_password)
38
39 breaker_filename = "breaker.pkl"
```

```

40 with open(breaker_filename, 'rb') as f:
41     breaker = pickle.load(f)
42
43 target_page = "http://localhost:3000/"
44 url = "http://localhost:3000/register"
45 html = requests.get(url).text
46 doc = BeautifulSoup(html, "html.parser")
47 captcha_html = doc.find('img')
48 captcha_source = captcha_html['src']
49 target_code = target_page + captcha_source
50 response = requests.get(target_code)
51 captcha = Image.open(BytesIO(response.content))
52 np_captcha = np.array(captcha)
53
54 label = breaker.solve_captcha(np_captcha)
55
56 sbbox = driver.find_element("name", "captchaTest")
57 sbbox.send_keys(label)

```

Listing 4.13: Cod Script de atac

Pentru inceput se studiaza pagina web tinta si se descopera etichetele casutelor cu input, in cazul nostru arata in felul urmator:

```

1 <div className="input-wrapper">
2     <label htmlFor="username">Email</label>
3     <input type="text" id="email" name="email" className="input-place"
placeholder="email" onChange={this.handleChange}/>
4 </div>

```

Listing 4.14: Cod container Input

Se observa ca casutele de input respective au in atributul *name*: "email", "password", "confirmPassword" si "captchaTest".

Librariile de care vom avea nevoie sunt

- requests care permite sa trimitem requesturi HTTP catre o aplicatie web foarte usor.
- BeautifulSoup, care formateaza si usureaza procesul de lucru cu cod HTML in Python.
- BytesIO, pentru a decoda resurse extrase.
- selenium care este o librarie utilizata in crearea testelor automate.
- random + string pentru generarea elementelor aleatoare.

Se creaza doua functii care genereaza email-uri si parole aleatoare. Intrucat rulam aplicatia secundara pe acelasi calculator domeniul sau va fi *http://localhost:3000/register* dupa care se instantiaza aplicatia automata `webdriver.Chrome()` si se deschide browserul.

Preluam casutele de input cu ajutorul functiei `find_element` care primeste ca parametri: 1. numele atributului dupa care cautam, si 2. valoarea acordata atributului respectiv, in cazul nostru fiind cele mentionate mai sus. Se inregistreaza in casute valorile generate aleator.

Urmeaza sa obtinem imaginea CAPTCHA, de obicei nu este mereu aceeaasi implementare, mai jos este explicat cum am procedat eu in aplicatia lucrarii. Deserializam obiectul salvat mai devreme, pentru rezolvarea CAPTCHA-ului. Se face un `get-request` catre domeniul pentru a obtine in intregime codul html al paginii, dupa care se formateaza cu ajutorul libreriei `BeautifulSoup`. Obtinem codul HTML potrivit CAPTCHA-ului, dupa care preluam ce se afla in atributul `'src'` - sursa imaginii. Vom concatena domeniul de baza si ce am obtinut din manipularile efectuate, astfel incat sa obtinem imaginea. Se decodeaza si mai apoi instantiaza CAPTCHA-ul respectiv ca obiect de tip `PIL` pentru a fi transformat in `numpy.array`. Obtinem eticheta CAPTCHA-ului si o trimitem cu ajutorul programului nostru automat catre casuta input.

Dupa rularea scriptului aplicatia secundara arata in felul urmator:



Figura 4.17: Captura de ecran a aplicatiei secundare dupa rularea scriptului

Bibliografie

- [1] „15 years of Google Books”, în (2020), URL: [%5Curl%7Bhttps://www.blog.google/products/search/15-years-google-books/%7D](https://www.blog.google/products/search/15-years-google-books/).
- [2] G. Bradski, „The OpenCV Library”, în *Dr. Dobb's Journal of Software Tools* (2000).
- [3] Antoni Buades, Bartomeu Coll și Jean-Michel Morel, „Non-Local Means Denoising”, în *Image Processing On Line* 1 (2011), https://doi.org/10.5201/ipol.2011.bcm_nlm, pp. 208–212.
- [4] Charles R. Harris, K. Jarrod Millman și et al. Stéfan J. van der Walt, „Array programming with NumPy”, în *Nature* 585.7825 (Sept. 2020), pp. 357–362, DOI: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2), URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [5] ImageMagick Studio LLC, „ImageMagick”, versiunea 7.0.10, în (4 Ian. 2023), URL: <https://imagemagick.org>.
- [6] Alexx Kay, „History and Evolution of CAPTCHA”, în (2001), URL: [%5Curl%7Bhttps://www.computerworld.com/article/2591759/artificial-neural-networks.html%7D](https://www.computerworld.com/article/2591759/artificial-neural-networks.html).
- [7] Graham Oppy și David Dowe, „The Turing Test”, în (2021), ed. de Edward N. Zalta.
- [8] Kishan Pandey, „History and Evolution of CAPTCHA”, în (2022), URL: [%5Curl%7Bhttps://www.masaischool.com/blog/history-evolution-of-captcha/%7D](https://www.masaischool.com/blog/history-evolution-of-captcha/%7D).
- [9] „SVG Files”, în (ne-mentionat), URL: [%5Curl%7Bhttps://www.adobe.com/creativecloud/file-types/image/vector/svg-file.html/#:~:text=frequently%5C%20asked%5C%20questions-,What%5C%20is%5C%20an%5C%20SVG%5C%20file%5C%3F,and%5C%20lines%5C%20on%5C%20a%5C%20grid.%7D](https://www.adobe.com/creativecloud/file-types/image/vector/svg-file.html/#:~:text=frequently%5C%20asked%5C%20questions-,What%5C%20is%5C%20an%5C%20SVG%5C%20file%5C%3F,and%5C%20lines%5C%20on%5C%20a%5C%20grid.%7D).
- [10] „Text-based CAPTCHA in 2022”, în (2022), URL: [%5Curl%7Bhttps://habr.com/en/articles/670520/%7D](https://habr.com/en/articles/670520/%7D).
- [11] Guido Van Rossum și Fred L. Drake, *Python 3 Reference Manual*, Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.
- [12] Wikipedia, „CAPTCHA”, în (2020), URL: [%5Curl%7Bhttps://en.wikipedia.org/wiki/CAPTCHA%7D](https://en.wikipedia.org/wiki/CAPTCHA) (accesat în 13.6.2023).

- [13] Seyhmus Yilmaz, Sultan Zavrak și Hüseyin Bodur, „Distinguishing Humans from Automated Programs by a novel Audio-based CAPTCHA”, în *International Journal of Computer Applications* 132 (Dec. 2015), pp. 17–21, DOI: [10.5120/ijca2015907575](https://doi.org/10.5120/ijca2015907575).