ROMÂNIA

MINISTERUL APĂRĂRII NAŢIONALE
ACADEMIA TEHNICĂ MILITARĂ
„Ferdinand I”

FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ
Specializarea: Calculatoare și sisteme informatice pentru apărare și securitate națională

Aplicație pentru extragerea si indexarea conținutului din documente scanate

COORDONATOR ȘTIINȚIFIC:
Lector dr. ing. Stelian SPÎNU

ABSOLVENT:
Sd.Plt. Alin-Romeo TUDOSE

Conține _____ file
Inventariat sub nr. _____
Poziția din indicator: ____
Termen de păstrare:_____

BUCUREȘTI
2021

PAGINĂ ALBĂ

PAGINA ALBA

PAGINĂ ALBĂ

PAGINĂ ALBĂ

PAGINĂ ALBĂ

PAGINĂ ALBĂ

ABSTRACT

In this day and age, the world is going through an imminent digitalization process, which also implies, amongst other factors, the transition from physical to digital files, in a manner that shall efficiently store the data.

A system of this sort is accessible to any public or private institution, as well as individuals whom are unlikely to have a great knowledge as far as computer usage goes.

A handwriting recognition system must fulfil two functions: the pre-processing of images in order to provide the data in a more familiar aspect, from which the features of each character can be easily extracted, and secondly, the classifier, which offers the predictions regarding the identified characters. The current application, apart from the recognition system, provides an efficient method of data storage, using a cutting-edge technology database, and the means of accessing it through a web interface.

The main issue in deep learning systems is the necessity of obtaining high enough accuracy in the recognition of information that it is provided with. For the moment, the best performance for computer vision problems is obtained by implementing convolutional neural networks.

The present paper provides the basic theoretical notions that are involved in the development of an application which generates handwriting recognition from filled-in documents, as well as to index the documents into a database for later access, the process that was pursued in order to implement this application, and, lastly, the obtained results.

REZUMAT

În prezent, lumea trece printr-un proces de digitalizare foarte puternic, ceea ce impune, printre altele, și convertirea documentelor din format fizic în format digital, într-o formă care să stocheze datele eficient.

Un astfel de sistem poate fi folosit de către orice instituție publică sau privată, de către oameni cărora, foarte probabil, le lipsește experiența de folosire la nivel înalt a calculatoarelor.

Un sistem de recunoaștere a scrisului de mână trebuie să îndeplinească două funcții: preprocesarea imaginilor pentru a aduce datele sub o formă cunoscută, din care trăsăturile caracterelor pot fi extrase ușor și clasificatorul, care oferă predicțiile despre caracterele identificate. Aplicația curentă, pe lângă sistemul de recunoaștere, oferă o modalitate eficientă de stocare a datelor, folosind o bază de date, și o metodă de accesare a datelor stocate, printr-o interfață web.

Principala problemă în sistemele de învățare profundă este obținerea unei precizii cât mai mari în recunoașterea informației pusă la dispoziția acestuia. Momentan, cele mai bune performanțe pentru problemele legate de computer vision sunt oferite de rețelele neuronale convoluționale.

Lucrarea descrie principalele noțiuni teoretice implicate în dezvoltarea unei aplicații care realizează recunoașterea scrisului de mână din documente completate, și indexarea lor într-o bază de date pentru accesare ulterioară, precum și traseul parcurs pentru implementarea acestei aplicații, și rezultatele obținute.

Cuprins

ABSTRACT 7
REZUMAT 8
LISTĂ ABREVIERI 11
LISTĂ FIGURI 12
INTRODUCERE 14
1.1. Importanța și actualitatea problematicii abordate 14
1.2. Scopul și obiectivele proiectului 14
1.3. Structura proiectului pe capitole 15
STADIUL CUNOAȘTERII ÎN DOMENIU 16
2.1. Fundamente teoretice 16
2.1.1. Inteligență artificială 16
2.1.2. Machine Learning 16
2.1.3. Deep Learning 17
2.1.4. Rețele neuronale 17
2.1.5. Reprezentarea datelor într-o rețea neuronală 19
2.1.6. Operații cu tensori 19
2.1.8. Rețele Neuronale Convoluționale 23
2.1.9. Operația de convoluție 23
2.1.10. Stratul BatchNormalization [7] 25
2.2. Problema proiectului 26
2.3. Soluții și abordări similare 27
2.3.1. Modelul propus de Alex Graves și Jurgen Schmidhuber, din cadrul Universității Tehnice din Munchen [3]. 27
2.3.2. Modelul propus de Hanu Priya Indiran, din cadrul Universității de Tehnologie Kamaruguru, India [2]. 28

2.3.3. Modelul propus de Arik Poznanski și Lior Wolf, din cadrul Universității Tel Aviv, „The Blavatnik School of Computer Science” [4]. 28
IMPLEMENTAREA APLICAȚIEI 30
3.1. Descrierea proiectului 30
3.2. Modulul de preprocesare 31
3.2.1. Preprocesarea formularelor goale 31
3.2.2. Preprocesarea formularelor completate 35
3.3. Clasificatorul 40
3.3.1. Seturi de date folosite 40
3.3.2. Arhitectura rețelei neuronale 41
3.4. Baza de date 51
3.5. Interfața Web 54
3.5.1. Introducerea unui tip nou de formular 55
3.5.2. Vizualizarea informațiilor despre un formular reținut 58
3.5.3. Recunoașterea scrisului dintr-un document completat 59
3.5.4. Vizualizarea documentelor recunoscute deja 60
REZULTATE OBȚINUTE 61
CONCLUZII ȘI DIRECȚII VIITOARE 62
BIBLIOGRAFIE 64

LISTĂ ABREVIERI

Nr. Crt	Abreviere	Forma completă
1.	IA	Inteligență Artificială
2.	ML	Machine Learning
3.	DL	Deep Learning
4.	CV	Simularea vederii pentru un computer (Computer Vision)
5.	SP	Procesare de Secvențe (Sequence Processing)
6.	CNN	Rețele Neuronale Convoluționale
7.	RNN	Rețele Neuronale Recurente
8.	MDRNN	Rețele Neuronale Recurente Multi-Dimensionale
9.	CTC	Clasificare Temporală Conexionistă
10.	LSTM	Long Short Term Memory
11.	HMM	Model Markov Ascuns
12.	OCR	Recunoașterea Optică a Caracterelor
13.	ReLU	Rectified Linear Unit
14.	SGD	Stochastic Gradient Descent

LISTĂ FIGURI

Figura 1 – SGD pentru o funcție de cost unidimensională (un singur parametru antrenabil) 22
Figura 2 - SGD pentru o funcție de cost bidimensională (doi parametri antrenabili) [1] 22
Figura 3 - Efectuarea convoluției pe o imagine, aplicând un filtru de dimensiune 3x3 24
Figura 4 - Exemplu imagine formular necompletat 33
Figura 5 - Imagine cu formular preprocesat 34
Figura 6 - Rezultatul OCR-ului peste imaginea formularului 34
Figura 7 - Extragere pixeli albaștri 35
Figura 8 - Extragere pixeli roșii 36
Figura 9 - Rezultatul final al preprocesării 37
Figura 10 - Vizualizarea sumelor pe orizontală în imagine 38
Figura 11 - Regiune extrasă din formular 38
Figura 12 – Vizualizarea sumelor pe verticală în regiune 39
Figura 13 - Litere extrase din regiuni 40
Figura 14 - Exemple de imagini de antrenare NIST 40
Figura 15 - Exemple de imagini de antrenare UNIPEN 41
Figura 16 - Modelul dens simplu 43
Figura 17 - Acuratețea modelului dens simplu 43
Figura 18 - Rețea convoluțională mică 44
Figura 19 - Performanțele modelului convoluțional mic 44
Figura 20 - Model convoluțional cu BatchNormalization 45
Figura 21 – Performanțele modelului convoluțional cu BatchNormalization 45
Figura 22 - Modelul convoluțional cu Dropout 45
Figura 23 - Performanțele modelului convoluțional cu Dropout 46
Figura 24 - Modelul final Kaggle 47
Figura 25 - Performanțele modelului Kaggle 47
Figura 26 - Performanțele modelului Kaggle după augmentarea datelor 48
Figura 27 - Acuratețea modelului numeric - 99.74% 49
Figura 28 - Acuratețea modelului text - 99.3% 50
Figura 29 - Acuratețea modelului mixt - 96% 50

Figura 30 - Exemplu obiect .json ce conține informația despre un tip de formular.....52
Figura 31 – Exemplu obiect .json ce conține informația recunoscută dintr-un document.....53
Figura 32 - Adăugarea unui nou formular.....56
Figura 33 – Completarea informațiilor despre un tip nou de formular.....57
Figura 34 - Vizualizarea unui formular.....58
Figura 35 - Recunoașterea scrisului dintr-un document.....59
Figura 36 - Vizualizarea documentelor.....60
Figura 37 - Conținutul recunoscut al unui document.....60

CAPITOLUL I

INTRODUCERE

1.1. Importanța și actualitatea problematicii abordate

În societatea modernă, automatizarea proceselor simple, de zi cu zi, a devenit, încetul cu încetul, un subiect de discuție foarte important. Începând de la micile sisteme de IoT, care preiau și excelează in desfășurarea activităților zilnice, precum încuietorile smart sau termostatele automate, care creează mediul optim, după preferințele fiecărei persoane, până la întregi linii de asamblare industriale, automatizarea tuturor acestor procese a permis ridicarea standardului de viață pentru întreaga omenire, fie că vorbim despre confortul propriu, fie despre capacitatea de producție a fabricilor în diferite industrii.

Una din problemele ce încă persistă în ziua de astăzi, este procesarea diferitelor tipuri de documente, în mod automat, și cât mai rapid. Rapiditatea, ușurința stocării, si eficiența nu sunt termeni asociați cu modalitatea tradițională de gestionare, unde una sau mai multe persoane erau însărcinate cu gestiunea arhivelor de documente. În plus, întreținerea unei persoane care să se ocupe de această gestiune poate fi costisitoare pe termen lung. Așadar, implementarea unei soluții eficiente, automate, care să reducă atât costurile, cât și timpul de răspuns la diferite tipuri de cereri este necesară.

1.2. Scopul și obiectivele proiectului

Proiectul își propune realizarea unei aplicații software ce va realiza în mod automat extragerea informațiilor din documente formatate, completate de mână. Această aplicație, primind ca input o imagine scanată a unui document preformatat, va identifica secțiunile de interes din imagine, și anume, informațiile completate de mână. Odată identificate, aceste secțiuni din imagine vor fi oferite unei mașini cu inteligență artificială, bazată pe tehnici de învățare profundă, ce va identifica textul din imagine la nivel de caracter. Caracterele prezise din imagine vor fi asamblate înapoi in cuvinte, și stocate într-o bază de

date nerelațională, ElasticSearch. Pentru ușurința accesării și configurării formatelor documentelor, datele salvate vor putea fi accesate printr-o interfață web.

1.3. Structura proiectului pe capitole

În capitolul II se definesc noțiunile teoretice necesare pentru înțelegerea funcționării unui clasificator bazat pe rețele neuronale convoluționale, plecând de la noțiunea de rețea neuronală, modul de funcționare, reprezentarea datelor, și convoluția dintr-o rețea neuronală convoluțională. De asemenea, capitolul II conține alte abordări problemei recunoașterii scrisului de mână, lucrări științifice publicate în diverse jurnale.

Capitolul III descrie organizarea aplicației pe componente, și prezintă succint modul de lucru, funcționalitățile și implementarea fiecăreia dintre acestea, făcând și o parcurgere a funcționalităților aplicației din punctul de vedere al utilizatorului.

Capitolul IV prezintă rezultatele și comportamentul aplicației, precum și punctele slabe ale acesteia.

Capitolul V descrie îmbunătățirile ce pot fi aduse aplicației, impactul prevăzut asupra societății, și concluziile proprii legate de dezvoltarea aplicației, și învățăturile acumulate pe parcursul redactării lucrării prezente.

CAPITOLUL II

STADIUL CUNOAȘTERII ÎN DOMENIU

2.1. Fundamente teoretice

În ultimii ani, inteligența artificială a fost un subiect de interes, atât pentru media, cât și pentru noii cercetători în domeniul Științei Calculatoarelor. Inteligența artificială, algoritmi de Machine Learning, chiar si algoritmi de Deep Learning, apar în tot mai multe articole, chiar și în afara publicațiilor științifice. Ne-a fost promis un viitor plin de autoturisme autonome și asistenți virtuali inteligenți, de care ne apropiem, din ce în ce mai rapid.

2.1.1. Inteligență artificială

Inteligența artificială a luat naștere în anii 1950, când o grupare de cercetători în nou-creatul domeniul al științei calculatoarelor, și-au pus următoarea întrebare: *„Ar putea calculatoarele să fie învățate să gândească?”*. Această întrebare este în dezbatere până în ziua de azi. O definiție pentru domeniul IA, ar putea fi următoarea: *„Efortul de a automatiza procese desfășurate, în mod normal, de oameni.”* [1]. Așadar, inteligența artificială este un domeniu cu un grad foarte mare de generalizare, ce cuprinde, dar nu se limitează la domeniile Machine Learning și Deep Learning.

2.1.2. Machine Learning

În Anglia epocii victoriene, Lady Ada Lovelace era o prietenă și colaboratoare a lui Charles Babbage, inventatorul mașinii analitice. Chiar dacă a fost mult înaintea timpului ei, mașina analitică nu a fost niciodată gândită ca un calculator de uz general, atunci când a fost proiectat, în anii 1830 – 1840, deoarece conceptul uzului general al unui calculator nu era încă inventat. A fost doar un mod de a folosi operații mecanice pentru a automatiza anumite calcule

din domeniul analizei matematice, de unde și numele acesteia. Ada Lovelace a făcut atunci remarca despre mașina analitică: *„Maşina Analitică nu are nicio pretenție sa aducă ceva nou. Ea poate să facă tot ce ştim noi să îi spunem să facă. Scopul ei este să ne asiste în găsirea rezultatelor problemelor cu care suntem familiari.”* [1].

Domeniul Machine Learning are la bază această intrebare: *„Poate un calculator să treacă dincolo de granițele a ceea ce știm noi să îi spunem să facă?”*. Așadar, este introdusă o nouă paradigmă de programare. Dacă în programarea clasică, unui calculator îi erau prezentate reguli și date, acesta trebuia să furnizeze rezultate. În domeniul ML, un calculator primește date și rezultate, iar acesta trebuie să furnizeze regulile de obținere a rezultatelor din acele date. Un sistem ML nu este un sistem programat explicit, ci este un sistem antrenat.

2.1.3. Deep Learning

Deep Learning-ul este un subdomeniu al domeniului ML, o nouă abordare asupra învățării reprezentărilor din date disponibile, ce pune accent pe straturi succesive de învățare. Cuvântul „Deep” din Deep Learning nu presupune vreun fel de înțelegere în profunzime a problemelor, ci doar face referire la modelele cu straturi succesive de învățare. Adâncimea modelului reprezintă numărul de straturi succesive de învățare pe care le are un model. Modelele actuale de DL adesea cuprind zeci, chiar sute de astfel de straturi succesive, și fiecare este antrenat automat prin prezentarea datelor si a rezultatelor așteptate pentru acestea [1].

2.1.4. Rețele neuronale

În ultimii 10 ani, cele mai performante sisteme cu inteligență artificială – de exemplu, sistemele de recunoaștere a vorbirii de pe smartphone-uri, sau cel mai recent translator automat de la Google – au rezultat prin aplicarea tehnicilor de Învățare Profundă (DL).

DL este, de fapt, doar un alt nume pentru rețelele neuronale, care au apărut pentru prima dată in 1944, fiind propuse de Warren McCullough și Walter Pitts, doi profesori de la Universitatea din Chicago.

O rețea neuronală este un mecanism prin care un calculator învață să facă anumite sarcini, analizând exemple de antrenare care îi sunt puse la dispoziție, exemplele fiind etichetate de mână anterior. De exemplu, un sistem de recunoaștere a obiectelor poate fi antrenat pe mii de imagini etichetate ce conțin case, mașini, căni de cafea, etc., și va identifica modele în imaginile ce au o corelație puternică cu etichetele lor.

Fiind construită pe modelul creierului uman, o rețea neuronală este formată din mii, sau chiar milioane, de noduri de procesare care sunt puternic interconectate. Majoritatea rețelelor neuronale din ziua de astăzi sunt organizate în straturi de noduri, și sunt *feed-forward*, adică datele trec prin acestea într-o singură direcție. Un nod individual poate fi conectat la mai multe noduri din stratul de dedesubt, de unde primește date, și la mai multe noduri din stratul de deasupra, către care trimite date.

Pentru oricare din nodurile de intrare, un nod asociază un număr numit „pondere”. Când rețeaua este activă, nodul primește date, numere, de la fiecare conexiune, o multiplică cu ponderea asociată acesteia, și le însumează, obținând un singur număr. Dacă acel număr este mai mic decât o valoare de prag, numită „prag de activare”, nodul nu trimite date către stratul următor. Dacă numărul depășește valoarea de prag, nodul se activează, adică trimite numărul obținut către toate conexiunile de ieșire ale acestuia.

Când se începe învățarea unei rețele, toate ponderile și pragurile de activare sunt inițializate cu valori aleatoare. Datele de antrenare sunt încărcate în stratul cel mai de jos, numit și stratul de intrare, și, progresiv, trece prin toate celelalte straturi ale rețelei, până când ajunge la stratul de ieșire. Pe timpul învățării, ponderile și pragurile de activare sunt modificate în mod constant, până când date cu aceeași etichetă vor oferi rezultate similare [6].

2.1.5. Reprezentarea datelor într-o rețea neuronală

În general, toate sistemele de ML folosesc tensori ca structură de date de bază. Tensorii sunt de o importanță deosebită în domeniul inteligenței artificiale, atât de importanți încât framework-ul de DL de la Google a fost denumit după aceștia – *Tensorflow*.

În esență, un tensor este un container pentru date, de obicei, numerice. Așadar un tensor este un container pentru numere. Tensorii pot avea mai multe dimensiuni: 0D – pentru scalari, 1D – pentru vectori, 2D – pentru matrice, 3D, ș.a.m.d..

Tensorii sunt caracterizați de trei atribute cheie:

- Numărul de axe – de exemplu un tensor 3D are 3 axe;
- Forma – dimensiunea tensorului de-a lungul fiecărei axe;
- Tipul de date – se referă la tipul datelor conținute în tensor, de obicei întregi, numere reale, etc.

2.1.6. Operații cu tensori

Așa cum fiecare program poate fi redus la un set mic de operații binare pe date – AND, OR, NOR, etc. – toate transformările efectuate de rețelele neuronale pot fi reduse la un set redus de operații cu tensori, aplicate pe tensori ce conțin date numerice. De exemplu, un strat dens conectat, cel mai utilizat tip de strat din framework-ul *Keras*, având, de obicei, funcția de activare *ReLU*, poate fi interpretat ca o funcție, care primește la intrare un tensor 2D, și returnează un alt tensor 2D:

$$output = relu(dot(W, input) + b)$$

Unde:

- W este un tensor 2D, b este un vector, amândoi atribute ale stratului;
- ReLU este funcția de activare, fiind de fapt *relu(x) = max(x,0)*;

Dot este operația de produs intern (în engleză: dot product), între doi tensori; exemplu: pentru doi tensori 1D, produsul intern este un scalar reprezentat de suma produselor elementelor de pe aceeași poziție din tensori, ($\sum_{i=0}^{n} x_i * y_i$); pentru doi tensori 2D, este echivalent cu produsul matriceal între două matrice.

2.1.7. Procesul de învățare al rețelelor neuronale: Algoritmul Gradient Descent

Așa cum am spus mai sus, un strat dens conectat este echivalent cu aplicarea funcției:

$$output = relu(dot(W, input) + b)$$

În această expresie, W și b sunt tensori ce aparțin stratului de învățare, și sunt numiți parametri antrenabili (W – atributul kernel; b – atributul bias). Acești parametri conțin informația învățată de rețea, fiind expuși la datele etichetate pe care se execută procesul de învățare. Inițial, aceștia sunt inițializați cu valori aleatoare, fără să aibă vreo semantică folositoare. Aceste valori inițiale nu reprezintă decât punctul de plecare pentru rețeaua neuronală. Pentru a îmbunătăți performanțele rețelei, acești parametri trebuie să fie ajustați pas cu pas, pe baza unor criterii. Această ajustare repetată reprezintă, de fapt, procesul de învățare a rețelei.

Învățarea se execută într-o buclă, iar algoritmul parcurge următorii pași:

1. Extrage un set de date *x*, și etichetele corespunzătoare *y*;
2. Obține predicțiile rețelei *y_pred*, pentru datele de intrare *x*;
3. Calculează costul rețelei pentru *x*, o metrică a distanței dintre *y* și *y_pred*;
4. Modifică ponderile rețelei astfel încât costul să fie redus pentru acest set de date;

După mai multe aplicări ale algoritmului peste datele de antrenare, vom rămâne cu un cost foarte mic pentru datele de antrenare, o eroare redusă între *y* si *y_pred*, și putem spune că rețeaua a învățat să mapeze corect intrările către etichetele dorite.

Dacă pașii 1-3 par simpli, și pot fi implementați doar cu noțiunile prezentate până acum, pasul 4 este mai dificil. Cea mai mare dificultate este să știm în ce direcție să modificăm ponderile rețelei, și să știm cu cât anume să le modificăm.

Pentru a găsi răspunsul la aceste probleme trebuie să profităm de faptul că toate operațiile care au loc în interiorul unei rețele neuronale sunt diferențiabile. Fiind dată o funcție diferențiabilă, este posibil, matematic, să fie găsit minimul acesteia. Punctele de minim ale unei funcții se află în punctele în care derivata funcției este 0. Așadar, pentru a găsi minimul funcției trebuie să găsim toate punctele în care derivata acesteia este 0, și să calculam în care dintre aceste puncte funcția are cea mai mică valoare.

Pentru o rețea neuronală, acest lucru înseamnă sa găsim combinația ponderilor pentru care funcția de cost are valoarea minimă; anume să rezolvăm ecuația *gradient(f)(W) = 0* pentru *W*. Dacă, matematic vorbind, este posibil să rezolvăm această ecuație, oricât de mulți parametri ar avea rețeaua, din punct de vedere computațional, acest lucru nu este deloc fezabil. Rețelele neuronale nu au niciodată un număr de parametri mai mic de ordinul miilor, adesea acesta fiind de ordinul zecilor de milioane, pentru arhitecturile complexe. Așadar, putem folosi algoritmul de învățare descris mai sus pentru a găsi minimul, modificând parametrii rețelei, puțin câte puțin, în sensul opus gradientului, pentru a obține mereu o valoare a funcției de cost mai mică. Această metodă se numește *Stochastic Gradient Descent* (Coborârea stohastică a gradientului).

Algoritmul parcurge, astfel, următorii pași:

- Extrage un set de date *x*, și etichetele corespunzătoare *y*;
- Obține predicțiile rețelei *y_pred*, pentru datele de intrare *x*;
- Calculează costul rețelei pentru *x*, o metrică a distanței dintre *y* și *y_pred*;
- Calculează gradientul funcției de cost, în funcție de parametrii rețelei;
- Modifică parametrii în sensul opus gradientului, după formula: *W -= lr * gradient*, astfel reducând valoarea costului cu puțin.
  (lr – pasul de învățare (learning rate), o valoare aleasă arbitrar)

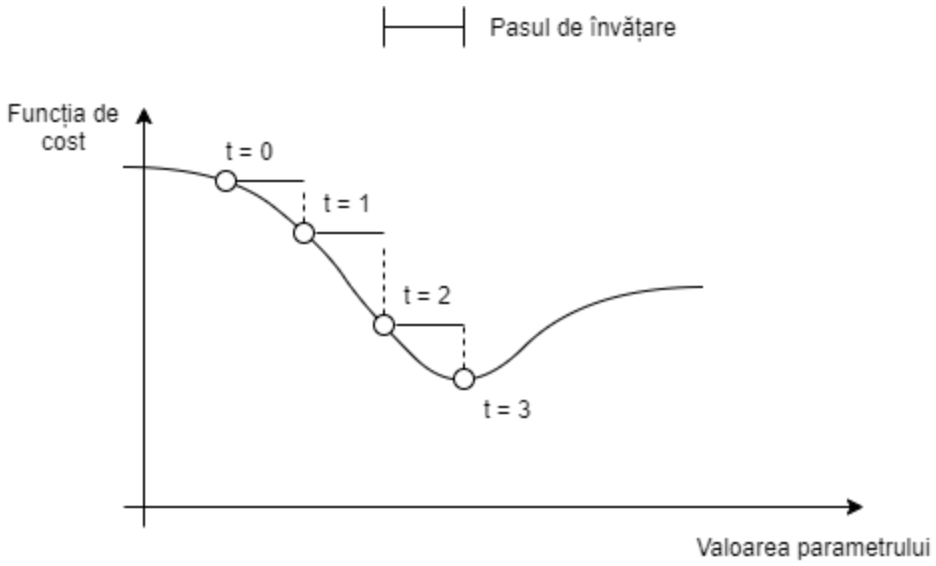


Figura 1 – SGD pentru o funcție de cost unidimensională (un singur parametru antrenabil)

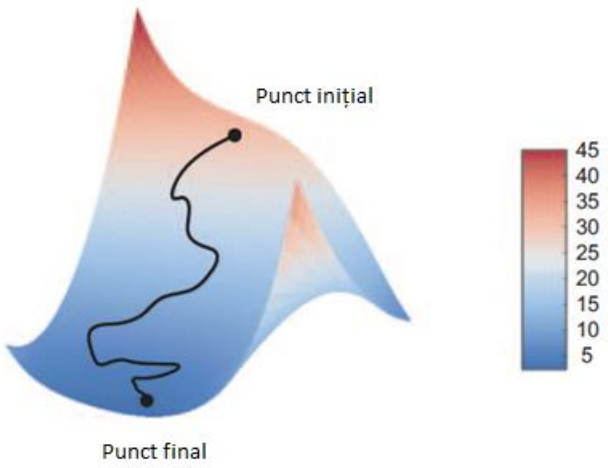


Figura 2 - SGD pentru o funcție de cost bidimensională (doi parametri antrenabili) [1]

După cum se poate observa, este foarte important să fie aleasă o valoare rezonabilă pentru rata de învățare *lr*; o valoare prea mică și algoritmul va avea nevoie de multe iterații pentru a găsi minimul global, sau ar putea rămâne blocat într-un minim local, iar o valoare prea mare ar putea face ca algoritmul să depășească minimul, și să ajungă în cu totul alte locații.

Dacă în imaginile de mai sus se observă SGD pentru într-un spațiu de parametri uni- sau bidimensional, în practică, SGD se aplică pentru spații

multidimensionale, de ordinul miilor, poate chiar a zecilor de milioane, deoarece fiecare parametru antrenabil introduce o nouă dimensiune în spațiu.

De asemenea, există mai multe variante de SGD, diferența între acestea fiind modul în care sunt gestionate modificările anterioare pasului curent, spre deosebire de varianta standard, în care se ține cont doar de valorile curente. Spre exemplu, se introduce inerția în modificarea parametrilor, pentru a rezolva problema vitezei de convergență și a blocării în puncte de minim local (RMSProp, Adagrad) [1].

2.1.8. Rețele Neuronale Convoluționale

Pentru problemele de CV, precum recunoașterea obiectelor în imagini, recunoașterea vorbirii, etc., cea mai bună soluție este folosirea de Rețelele Neuronale Convoluționale. Acestea pot fi folosite pentru probleme de DL, pentru orice tip de date, atâta timp cât datele pot fi reprezentate sub forma unui tensor 3D.

La fel ca rețelele neuronale standard, CNN-urile sunt alcătuite din mai multe straturi de neuroni, pe lângă care folosesc și straturi convoluționale.

2.1.9. Operația de convoluție

Diferența fundamentală între un strat dens și unul convoluțional este aceea că straturile dense învață șabloane la nivel global în spațiul lor de intrare, în timp ce straturile convoluționale învață șabloane local, în cazul imaginilor, șabloane conținute în ferestre bidimensionale mici, de dimensiune 3x3, 5x5, etc.. Așadar, informația învățată de straturile convoluționale este invariantă la translații; după ce învață un șablon în colțul din stânga sus al unei imagini, îl pot recunoaște oriunde în imagine, spre exemplu în colțul din dreapta jos. O rețea ce folosește doar straturi dense, ar trebui să învețe șabloanele în fiecare locație în care apar. De aceea, rețelele convoluționale sunt mult mai eficiente în procesarea datelor.

Încă un lucru important despre straturile convoluționale este acela că pot învăța ierarhii spațiale ale șabloanelor. Un prim strat convoluțional va învăța obiecte mici precum muchii, sau colțuri, un al doilea strat convoluțional va învăța șabloane mai mari, construite din trăsăturile învățate de primul strat, un al treilea va învăța bazat pe trăsăturile învățate de al doilea strat, ș.a.m.d..

Operația de convoluție se bazează pe filtre, matrice cu dimensiuni definite de programator. Filtrarea unei ferestre din imaginea de intrare constă în înmulțirea element cu element între filtru și fereastră, apoi efectuarea mediei între rezultatele obținute. Filtrarea pentru toată imaginea presupune deplasarea ferestrei peste toată imaginea de intrare, rezultatele fiind aranjate sub forma unei noi matrice. De obicei, rețelele convoluționale folosesc mai multe filtre, așadar un strat convoluțional al unei rețele ce primește la intrare o imagine, și aplică peste aceasta *N* filtre, va avea ca ieșire un număr de *N* matrice.

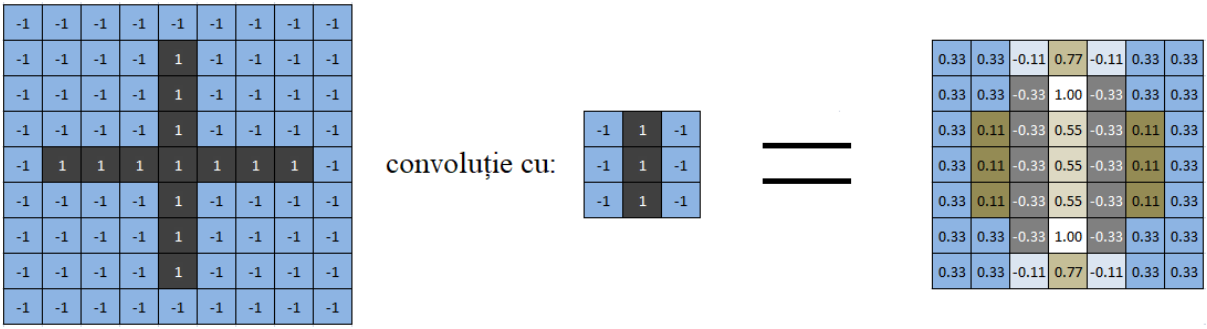


Figura 3 - Efectuarea convoluției pe o imagine, aplicând un filtru de dimensiune 3x3

În figura 3 este prezentat modul în care un CNN interpretează o imagine. Dacă pentru un om, recunoașterea simbolului „+” în imagine este o sarcină banală, pentru un calculator nu este deloc ușor. Efectuarea operației de convoluție pentru imagine este primul pas în recunoașterea simbolului. După cum se poate observa, filtrul obține valorile maxime în porțiunile din imaginea de intrare în care pixelii sunt dispuși la fel ca în filtru.

2.1.10. Stratul BatchNormalization [7]

Stratul BatchNormalization este folosit pentru a normaliza intrările acestuia. Se aplică o transformare peste datele de intrare, pentru a menține media ieșirilor aproape de 0, și deviația standard aproape de 1.

BatchNormalization funcționează diferit pe timpul învățării față de timpul predicției.

La învățare, stratul normalizează ieșirile folosind media si deviația standard a setului curent de date. Așadar, pentru fiecare canal al ieșirii din straturile convoluționale, stratul returnează:

$$\gamma * \frac{batch - mean(batch)}{\sqrt{var(batch) + \varepsilon}} + \beta$$

Unde:

- γ – factor de scalare antrenabil, inițializat cu 1;
- ε – constantă cu valori mici, inițializabilă din constructorul stratului;
- β – factor de translație antrenabil, inițializat cu 0;
- batch – setul de date de intrare;
- mean(batch) – media setului de date de intrare;
- var(batch) – deviația standard a setului de date de intrare;

La predicție, stratul normalizează ieșirile folosind o medie și deviație standard egală cu valorile mediane ale mediilor și deviațiilor standard ale seturilor de date peste care s-a efectuat învățarea. Așadar, va returna:

$$\gamma * \frac{batch - moving\_mean}{\sqrt{moving\_var + \varepsilon}} + \beta$$

Unde *moving_mean* și *moving_var* sunt variabile modificate la fiecare etapă de învățare, conform expresiilor:

- $moving\_mean = moving\_mean * momentum + mean(batch) * (1 - momentum)$;

- $moving\_var = moving\_var * momentum + var(batch) * (1 - momentum)$.

2.2. Problema proiectului

Traducerea automată a imaginilor ce conțin scris de mână în seturi de date ce pot fi recunoscute de un computer este un proces destul de dificil. În cadrul acestei teme apar probleme complicate, precum problema segmentării sau problema dicționarului. Pentru a găsi o soluție acestei probleme, este necesar să folosim una din tehnologiile recente descoperite în domeniul DL, și anume Computer Vision (Simularea vederii pentru un computer).

Pentru rezolvarea problemelor de Computer Vision, se folosesc, în majoritatea cazurilor, rețele neuronale convoluționale. Acestea introduc operația de convoluție, în straturile lor. Diferența fundamentală între un strat „Dense" și un strat convoluțional este următoarea: Un strat dens învață trăsături reprezentative în întreg spațiul său de intrare, in timp ce un strat convoluțional învață aceste trăsături local, în cazul imaginilor, in mici ferestre bidimensionale, de obicei de dimensiuni 3 x 3 sau 5 x 5.

Această caracteristică conferă unui CNN două proprietăți interesante:

1. Trăsăturile învățate de acestea sunt invariante la translatări. După ce a fost învățată o trăsătura în colțul din dreapta, jos al unei imagini, acesta poate fi recunoscut în orice altă parte a imaginii, de exemplu în stânga, sus.
2. Pot învăța ierarhii spațiale ale trăsăturilor. Un prim strat convoluțional va învăța trăsături mici, locale, precum colțuri, sau muchii, un al doilea strat va învăța trăsături mai generale, alcătuite din trăsăturile învățate de primul strat, și așa mai departe. Acest lucru permite CNN-urilor să învețe trăsături din ce în ce mai complexe și concepte vizuale abstracte.

2.3. Soluții și abordări similare

2.3.1. Modelul propus de Alex Graves și Jurgen Schmidhuber, din cadrul Universității Tehnice din Munchen [3].

Cei doi cercetători propun un model aparte. În loc să trateze problemele CV si SP separat, precum modelele HMM, folosite pentru transcrieri, aceștia, folosindu-se de două descoperiri recente în domeniul rețelelor neuronale, RNN multidimensionale, și CTC, introduc un model antrenat offline pentru recunoaștere a scrisului de mână, care acceptă la intrare informație sub forma de pixeli de imagine.

Pentru MDRNN se folosesc straturi LSTM. Un strat LSTM este format din celule de memorie conectate in mod recurent, ale căror activări sunt controlate de 3 porți: poarta de intrare, poarta de uitare, și poarta de ieșire. Aceste porți permit celulelor de memorie să rețină, să șteargă și să acceseze informații reținute anterior, obținând recunoașteri dependente de un context destul de larg.

Forma standard a straturilor LSTM, este unidimensionala, deoarece fiecare celulă de memorie conține o singură conexiune recurentă, controlată de o singură poartă de uitare. Totuși, aceste straturi se pot extinde la n dimensiuni, folosind n conexiuni recurente, una pentru fiecare stare anterioară de-a lungul fiecărei dimensiuni, cu *n* porți de uitare.

CTC este un strat de ieșire creat special pentru probleme de procesare de secvențe folosind RNN-uri. Spre deosebire de alte straturi de ieșire, stratul CTC nu necesită date de antrenare pre-segmentate, și nici post-procesarea ieșirilor, pentru a obține datele recunoscute. Acesta antrenează direct rețeaua să estimeze probabilitățile condiționale ale etichetelor posibile prezente in secvențele de intrare.

Acest model nu are nevoie de niciun fel de preprocesare specifică fiecărui alfabet, deci poate fi folosit neschimbat pentru orice limbă. Dovezile generalității si a eficienței acestuia sunt asigurate de datele puse la dispoziție de o competiție recentă de recunoaștere a caracterelor arabice, unde a obținut o acuratețe de 91.4%, (câștigătorul concursului a obținut 87.2%), chiar dacă niciunul din autori nu cunosc alfabetul arabic.

2.3.2. Modelul propus de Hanu Priya Indiran, din cadrul Universității de Tehnologie Kamaruguru, India [2].

Proiectul lui Indiran dorește să clasifice un cuvânt individual, astfel încât textul scris de mână să poată fi translatat într-o forma digitală, și demonstrează folosirea rețelelor neuronale pentru a proiecta un sistem de recunoaștere a caracterelor din alfabetul englez. Acest sistem preia la intrare imaginile binarizate cu literele ce trebuie să fie recunoscute. Imaginile sunt introduse într-un sistem de extragere de trăsături, a cărui ieșire este luat ca intrare în rețeaua neuronală. Indiran abordează două metode pentru rezolvare: clasificarea cuvintelor în mod direct, dar și segmentarea acestora și recunoașterea la nivel de caracter.

Pentru prima metodă, diferite arhitecturi de CNN sunt folosite pentru a antrena un model cu acuratețe ridicată, pentru costuri computaționale relativ mici. Dezavantajul acestei metode este însă faptul ca este dependentă de un dicționar de cuvinte inițial, pentru a putea clasifica intrările.

Pentru cea de-a doua metodă, un RNN cu straturi LSTM, sunt folosite, împreună cu convoluția, pentru a crea delimitări pentru fiecare caracter, în imaginile de intrare. După ce au fost extrase imaginile pentru fiecare caracter din imagine, acestea sunt date către un CNN pentru clasificare. În urma clasificării, se reconstruiește fiecare cuvânt, în funcție de rezultatele obținute. Performanțele vorbesc de la sine, obținându-se o eroare de 1% în recunoașterea cuvintelor. Dacă se folosesc straturi de dimensiuni foarte mari pentru CNN, eroarea de recunoaștere poate scădea pana la 0.2%.

2.3.3. Modelul propus de Arik Poznanski și Lior Wolf, din cadrul Universității Tel Aviv, „The Blavatnik School of Computer Science” [4].

Proiectul celor doi profesori adoptă o metoda relativ aparte. Aceștia folosesc un CNN pentru a estima, plecând de la imaginea unui cuvânt scris de mână, frecvențele grupărilor de n caractere ce apar în cuvânt. Frecvențele monogramelor, digramelor și trigramelor, sunt folosite pentru a asocia profilul

estimat al cuvântului de recunoscut, cu profilul real al cuvintelor dintr-un dicționar de dimensiuni mari. Din nou, metoda cu dicționarul nu este o metodă facilă, pentru tema proiectului, din cauza necesității recunoașterii datelor care nu se afla neapărat într-un dicționar, precum nume, prenume, adrese e-mail, etc..

Modelul este bazat pe o arhitectura de tip VGG, formată din straturi convoluționale mici, de dimensiune (3 x 3). Rețeaua are un total de 12 straturi, 9 straturi convoluționale, si 3 straturi dense. Performanțele acestui model sunt destul de ridicate, obținându-se o acuratețe de aproximativ 95%, pe mai multe seturi de date, printre care se numără și seturile IAM, SVT și RIMES.

CAPITOLUL III

IMPLEMENTAREA APLICAȚIEI

3.1. Descrierea proiectului

Având o aplicație ce îmbină tehnologii din mai multe câmpuri de cunoaștere din domeniul Computer Science, am hotărât că o abordare modulară a implementării este cea mai potrivită metodă.

Astfel, se pot distinge 4 mari componente, și anume:

- Modulul de preprocesare*(python)*;
- Clasificatorul*(python)*;
- Baza de date*(ElasticSearch)*;
- Interfața web*(PHP + CSS + JavaScript + MySQL)*.

Folosind rețeaua neuronală, putem realiza recunoașterea textului prin două metode: recunoașterea la nivel de caracter, și recunoașterea la nivel de cuvânt.

Abordările bazate pe recunoașterea la nivel de caracter, împart, mai întâi, cuvântul în caractere sau subcaractere. Caracterele scrise de mână fără constrângeri sunt adesea conectate, sau chiar se suprapun cu caracterele învecinate, ceea ce face dificil de spus unde se termină un caracter, și începe următorul. În asta constă problema segmentării, acest tip de abordări fiind susceptibil la erori de segmentare.

Abordările bazate pe recunoașterea la nivel de cuvânt nu utilizează segmentare, ci recunosc cuvintele ca entități. Astfel, intervine problema dicționarului, deoarece aceste sisteme au nevoie de cel putin un model, sau șablon pentru fiecare cuvânt recunoscut. Aceasta abordare nu poate fi implementată în proiect, deoarece, de obicei, în formulare sunt prezente nume de persoane, adrese, numere de telefon, sau alte informații pentru care nu se poate genera un șablon eficient.

Pentru ușurarea implementării, și evitarea întâmpinării problemelor precum problema dicționarului, sau problema segmentării caracterelor scrise de mână, am impus, fără a restrânge generalitatea aplicației, ca formularele ce vor fi

recunoscute să fie completate doar cu majuscule, cifre, sau simboluri, introduse, fiecare caracter într-o căsuță separată. Astfel, impunem ca recunoașterea textului scris de mână să fie făcută la nivel de caracter, iar problema segmentării caracterelor dispare.

De asemenea, pentru a putea îmbunătăți calitatea recunoașterii, fara a introduce inconveniențe prea mari, am hotărât ca, pentru recunoașterea unui formular, să fie introdusă, mai întâi, o poză cu formularul necompletat, pentru a putea extrage informații suplimentare despre imaginile ce urmează a fi recunoscute, fără ca acestea să fie influențate de scrisul de mână.

În secțiunile următoare, vom parcurge, în detaliu, fiecare dintre aceste componente, prezentând părțile componente, si funcționalitățile implementate.

3.2. Modulul de preprocesare

Preprocesarea este utilizată, în principal, pentru reducerea zgomotului prezent în datele de intrare ale clasificatoarelor, îmbunătățind astfel performanțele sistemelor. În aplicațiile de acest tip, modulul de preprocesare este de o importanță deosebită, deoarece de el depind în mod direct performanțele sistemului, acesta având sarcina de a extrage caracteristicile, și a le pune într-o formă care să faciliteze recunoașterea cât mai corectă a datelor de către clasificator.

În aplicația prezentată, modulul de preprocesare este folosit în două scenarii, procesarea formularelor completate, și procesarea formularelor goale, ce urmează a fi recunoscute.

3.2.1. Preprocesarea formularelor goale

Pentru recunoașterea oricărui document completat, este necesar ca, mai întâi, să fie introdusă o imagine cu formulatul necompletat, pentru a obține informații despre document ce vor facilita extragerea și recunoașterea ulterioară a caracterelor. Această acțiune trebuie efectuată o singură data, pentru fiecare

tip de formular, recunoașterile ulterioare folosind informațiile extrase de prima dată.

Modulul de preprocesare, în această situație, are rolul de a construi un document *.json* ce va conține informații despre formular, precum mărimea originală a formularului, titlul, o scurtă descriere, textul tipărit pe formular, și numele și tipul fiecărui câmp de completat, pentru selectarea modelului de rețea neuronală folosit la clasificare.

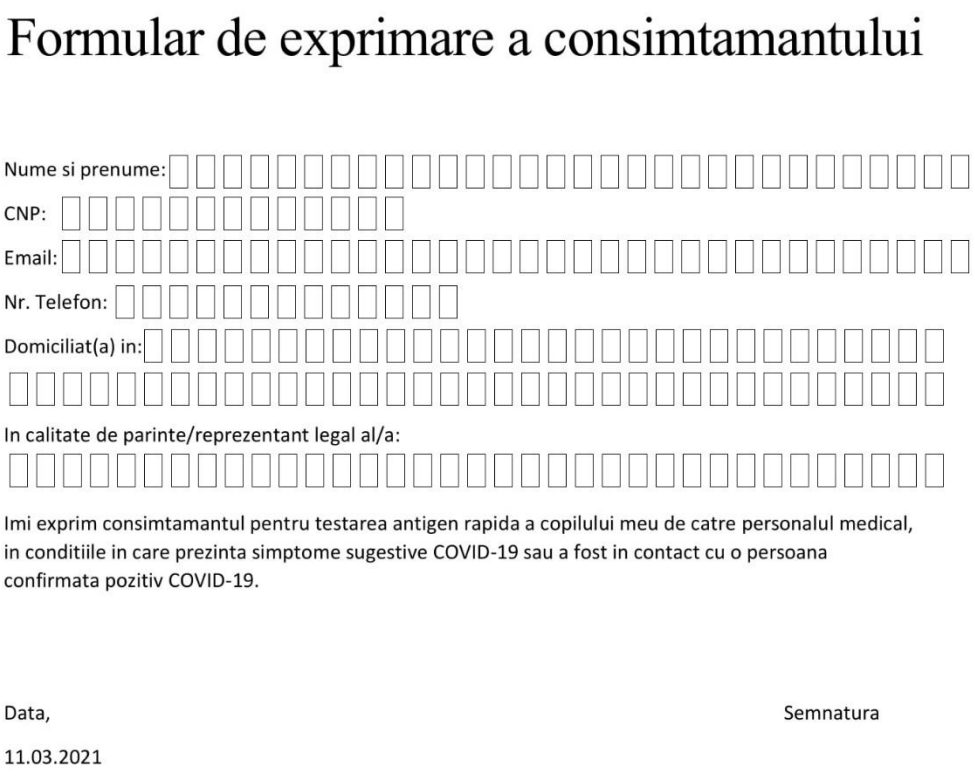
Toate caracteristicile prezentate mai sus, în afară de textul tipărit, sunt introduse de către utilizator în interfața web a aplicației, și trimise către modulul de preprocesare, pentru a fi îmbinate la final, deoarece se pot introduce rapid si fără inconveniențe. Introducerea textului tipărit, este o cu totul altă situație, deoarece aceste formulare pot conține volume mari de text, iar introducerea acestuia poate fi deranjantă. De aceea am hotărât ca extragerea textului să se facă în mod automat folosind tehnologii de OCR.

Pentru acest lucru, am folosit *Pytesseract*, un utilitar de OCR, deținut si dezvoltat de Google, fiind un framework peste motorul Google Tesseract-OCR [5]. Astfel, cu ajutorul acestui utilitar, realizăm recunoașterea și extragerea textului tipărit.

Formularele nu conțin, însă, doar text tipărit, ci conțin și căsuțele de completat. Acestea pot să inducă în eroare motorul de OCR, așadar, trebuie aplicată o preprocesare pentru a le elimina. În cazul acesta, preprocesarea constă în convertirea imaginii din color în tonuri de gri, și aplicarea peste aceasta a unui filtru median cu fereastra de dimensiune 3. Acest filtru este de ajuns pentru a elimina căsuțele din imagine, și orice alt zgomot ce poate apărea în imagine.

În urma extragerii informațiilor, se creează obiectul *.json,* care este mai apoi introdus în baza de date ElasticSearch pentru a fi folosit ulterior la recunoașterea formularelor completate.

Un exemplu de poză cu un formular gol:



Formular de exprimare a consimtamantului

Nume si prenume:

CNP:

Email:

Nr. Telefon:

Domiciliat(a) in:

In calitate de parinte/reprezentant legal al/a:

Imi exprim consimtamantul pentru testarea antigen rapida a copilului meu de catre personalul medical, in conditiile in care prezinta simptome sugestive COVID-19 sau a fost in contact cu o persoana confirmata pozitiv COVID-19.

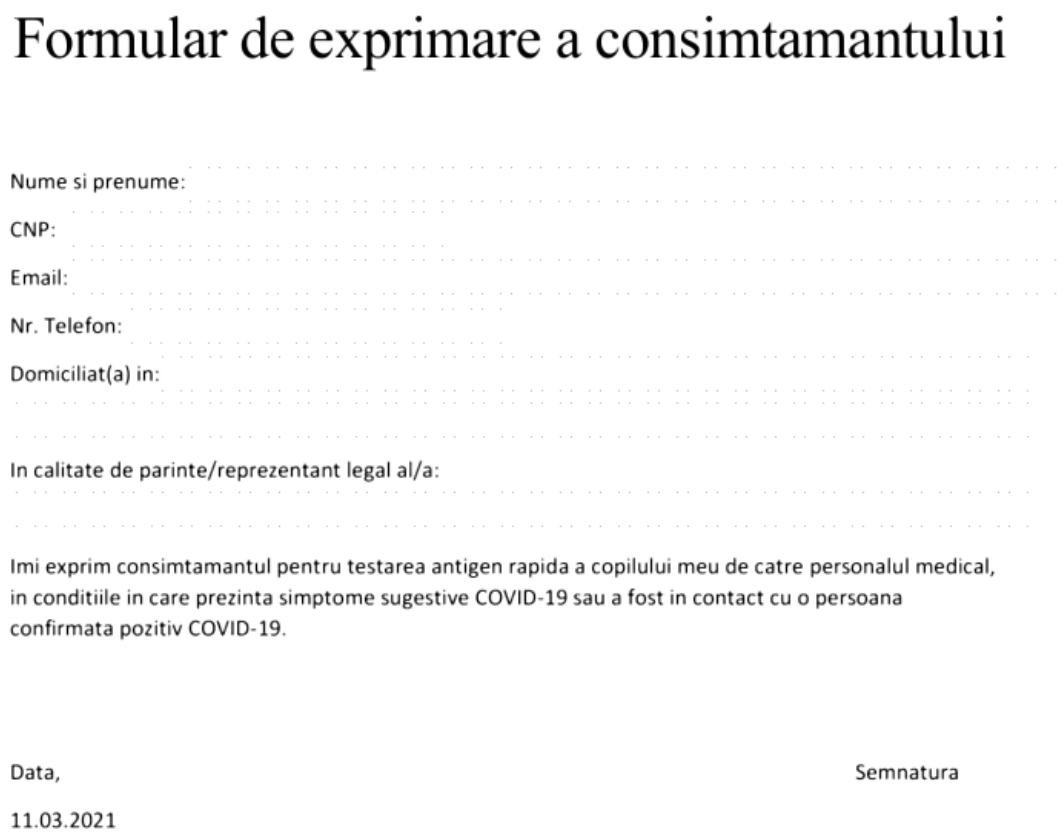
Data,

Semnatura

11.03.2021

Figura 4 - Exemplu imagine formular necompletat

În urma preprocesării parțiale, în vederea maximizării rezultatelor OCR-ului, imaginea arată astfel:


Formular de exprimare a consimtamantului

Nume si prenume:

CNP:

Email:

Nr. Telefon:

Domiciliat(a) in:

In calitate de parinte/reprezentant legal al/a:

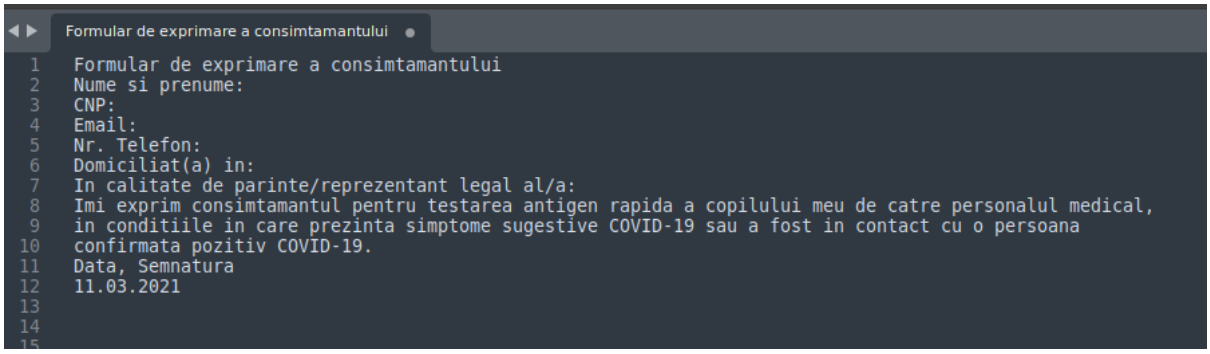
Imi exprim consimtamantul pentru testarea antigen rapida a copilului meu de catre personalul medical, in conditiile in care prezinta simptome sugestive COVID-19 sau a fost in contact cu o persoana confirmata pozitiv COVID-19.

Data, Semnatura

11.03.2021

Figura 5 - Imagine cu formular preprocesat

După această etapă, se execută OCR-ul peste imaginea obținută, și rezultatul este:


Formular de exprimare a consimtamantului

```
1  Formular de exprimare a consimtamantului
2  Nume si prenume:
3  CNP:
4  Email:
5  Nr. Telefon:
6  Domiciliat(a) in:
7  In calitate de parinte/reprezentant legal al/a:
8  Imi exprim consimtamantul pentru testarea antigen rapida a copilului meu de catre personalul medical,
9  in conditiile in care prezinta simptome sugestive COVID-19 sau a fost in contact cu o persoana
10 confirmata pozitiv COVID-19.
11 Data, Semnatura
12 11.03.2021
13
14
15
```

Figura 6 - Rezultatul OCR-ului peste imaginea formularului

3.2.2. Preprocesarea formularelor completate

Dacă preprocesarea formularelor goale a fost o sarcină relativ simplă, putând fi folosite utilitare pentru părțile complicate, și intervenția utilizatorului pentru părțile simple, preprocesarea formularelor completate este cu totul altfel.

Modulul de preprocesare primește imaginea cu documentul dorit și obiectul *.json* al tipului de formular adăugat anterior, și are sarcina de a extrage imagini ce conțin fiecare caracter ce urmează a fi recunoscut. Pentru a realiza această sarcină, au fost parcurși pașii următori:

3.2.2.1. Extragerea pixelilor ce conțin informația scrisă de mână

Pentru a diferenția cu ușurință textul scris de mână de textul de tipar prezent în formular și pentru a face o extragere a caracterelor cât mai bună, trebuie să fie impus următorul lucru: formularul trebuie să fie completat cu o altă culoare decât culoarea textului tipărit, negru. Aici apare întrebarea „Ce culoare ar fi potrivită pentru completare?”. Răspunsul evident ar putea să pară că este albastru, însă, după cum se poate vedea în imaginea de mai jos, pentru documente scanate, unde scanarea nu este ideală, negrul poate fi confundat foarte ușor cu albastru închis, și să fie interpretat ca fiind scris de mână.

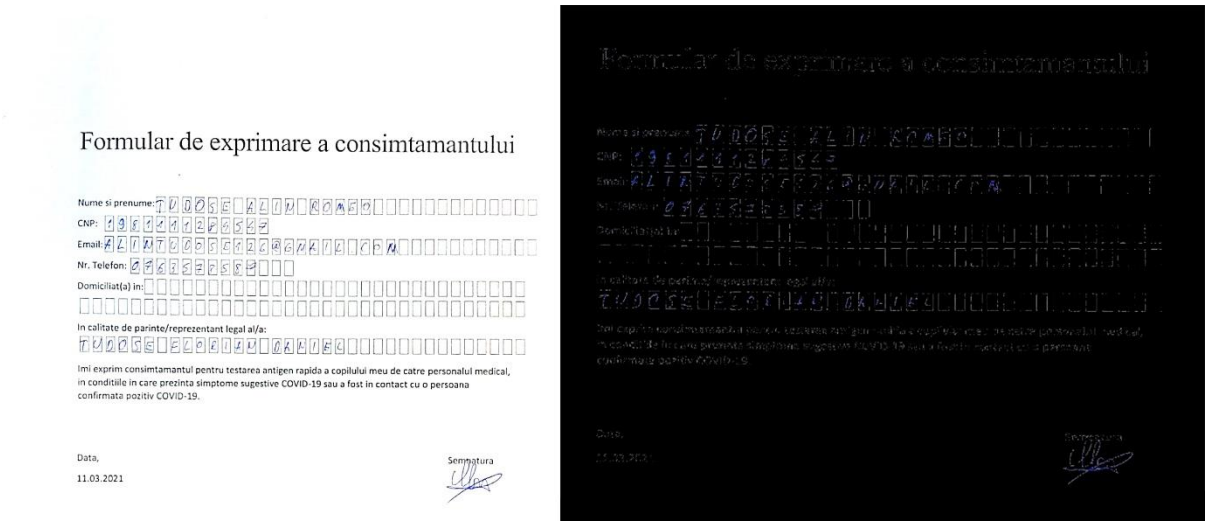


Figura 7 - Extragere pixeli albaștri

Prin urmare, am ales să folosim pentru completare culoarea roșu, fiind ușor diferențiabilă de negru în imagini, și, în același timp, fiind o culoare utilizată în scris. Putem observa diferențele majore în separarea culorilor în imaginea următoare.

Figura 8 - Extragere pixeli roșii

Separarea culorilor se face folosind biblioteca *opencv-python*, cu ajutorul căreia putem extrage dintr-o imagine doar pixelii ce au valorile cuprinse într-un interval setat de noi. Rezultatele din imaginea precedentă au fost obținute pentru valorile RGB (0,0,100) pentru minim, și (100,100,255) pentru maxim.

După extragerea pixelilor roșii, au loc următoarele operații:

- Aplicarea unui filtru median, de dimensiune 3, pentru eliminarea eventualului zgomot prezent în imagine, datorită scanării cu defecte a imaginii, sau prezența altor artefacte;
- Conversia imaginii extrase în tonuri de gri, pentru interpretarea ulterioară a imaginii;
- Efectuarea binarizării tip OTSU.

Figura 9 - Rezultatul final al preprocesării

3.2.2.2. Extragerea regiunilor de interes

Pe baza imaginii obținute anterior, următorul pas este selectarea din imagine a regiunilor ce conțin informația completată. Aici se vede utilitatea convertirii imaginii în tonuri de gri, si binarizarea. În urma binarizării, pixelii din imagine vor avea doar două valori: 255 (alb) – pentru zonele ce conțin scris de mână, și 0 (negru) – pentru zonele de fundal. Astfel, extragerea zonelor de interes devine o problemă banală.

Înainte de extragerea propriu-zisă, folosind obiectul *.json* obținut la adăugarea formularului necompletat în aplicație, se redimensionează imaginea introdusă la dimensiunile formularului original, și se extrag dimensiunile căsuțelor pentru caractere, spațiul dintre acestea, și numele și tipul câmpurilor prezente.

Imaginile sunt reprezentate în memorie ca un tablou tridimensional, de dimensiuni *înălțime* x *lățime* x 3, pentru imaginile color, și un tablou bidimensional, de dimensiuni *înălțime* x *lățime*, pentru imaginile în tonuri de gri. Cum, în urma operațiilor anterioare, se obține o imagine binarizată în tonuri de gri, iar zonele ce conțin scris de mână au valoarea 255, putem extrage zonele de interes folosind valoarea sumei pixelilor pe orizontală pentru fiecare linie din imagine. Astfel, daca pe o linie avem suma pixelilor 0, înseamnă că pe acea

linie nu există scris de mână, iar linia va fi ignorată. Însă, dacă pe o linie avem suma mai mare decât 0, înseamnă ca pe o anumită poziție de pe acea linie este conținut scris de mână. Considerând o regiune de interes ca fiind un șir consecutiv de linii ce au suma valorilor mai mare ca 0, se poate separa cu ușurință textul completat pentru fiecare câmp, de restul imaginii. Pentru a obține rezultate mai bune, și eliminarea spațiului inutil din zonele extrase, se trunchiază linia din imagine, păstrându-se înainte de primul, și după ultimul caracter, o zona de lungime egală cu spațiul dintre căsuțe.

Un exemplu se poate vedea în figura 10, zonele extrase fiind cele marcate cu galben, coloana din stânga reprezentând valoarea sumei pe orizontală.

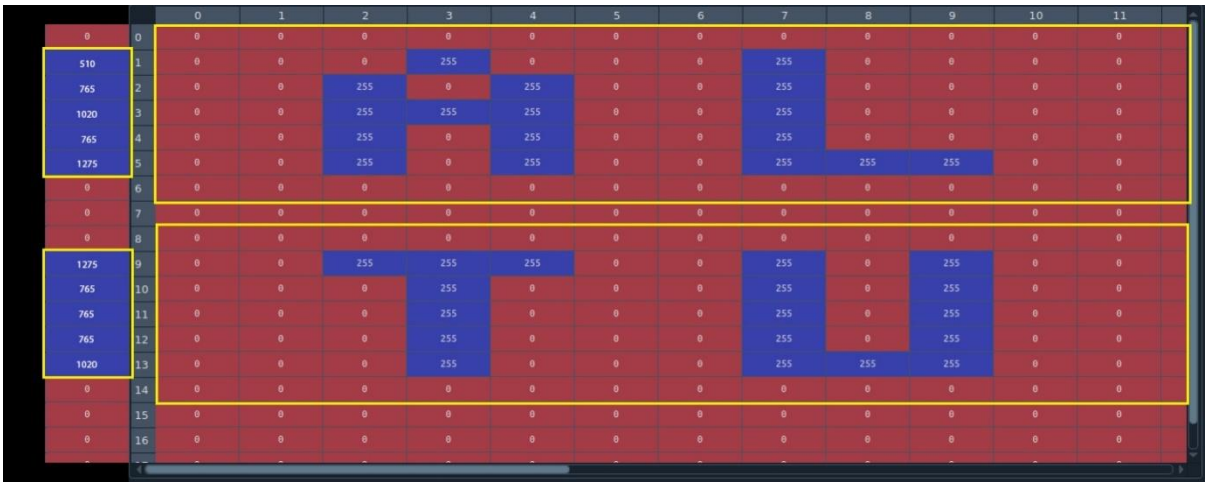


Figura 10 - Vizualizarea sumelor pe orizontală în imagine

Figura 11 - Regiune extrasă din formular

În urma acestei operații se obțin mai multe imagini, fiecare conținând secvența de caractere scrisă de mână, pentru un anumit câmp. Este foarte important să se rețină fiecare regiune cărui câmp aparține, pentru a face reconstrucția ulterioară a documentului cât mai ușoară.

3.2.2.3. Segmentarea caracterelor în regiunile de interes

Aplicând un raționament asemănător cu cel anterior, fiecare caracter din regiunile extrase este separat de caracterele vecine printr-o zona goală. Acest lucru se datorează faptului că am impus completarea formularelor cu majuscule, fiecare caracter având propria căsuță. Chiar dacă, pentru aplicație, căsuțele pentru caractere sunt ignorate, acestea obligă utilizatorul să despartă caracterele la momentul completării, evitând, astfel, problemele complexe de segmentare.

Totuși, aici trebuie să se țină cont de pozițiile relative ale caracterelor, pentru a putea forma cuvinte, și pentru a elimina spațiul nenecesar dintre caractere.

De data aceasta, segmentarea caracterelor se va face efectuând suma pe verticală. Se vor extrage coloanele ce au suma diferită de 0, fiecare caracter fiind reprezentat de un șir consecutiv de coloane cu suma diferită de 0. Daca distanța dintre două caractere consecutive depășește lățimea unei căsuțe, se consideră că pe acea poziție este un spațiu, și poziția va fi memorată pentru reconstrucția ulterioară a cuvintelor.

Figura 12 – Vizualizarea sumelor pe verticală în regiune

Se extrage apoi, din regiune, fiecare zonă ce aparține unui caracter scris, fiind totuși asociate, în continuare, câmpului din formular, pentru a nu se pierde informația despre apartenența caracterelor unui anumit câmp.

În urma acestei operații, obținem câte o imagine pentru fiecare caracter completat de mână din formular. Aceste imagini vor fi oferite ca input

clasificatorului, pentru a obține textul recunoscut, făcându-se reconstrucția digitală a documentului.

Figura 13 - Litere extrase din regiuni

3.3. Clasificatorul

Clasificatorul este constituit dintr-o rețea neuronală convoluțională. Întrucât NN-urile au nevoie de un set de date de antrenare cât mai mare pentru a obține performanțe ridicate, au fost folosite, pentru procesul de învățare, bazele de date *NIST Special Database 19*, și *UNIPEN*.

3.3.1. Seturi de date folosite

- **NIST Special Database 19** [8]**:**

Acest set de date conține întregul corpus de materiale de antrenare pentru recunoașterea formularelor completate de mână și recunoaștere de caractere. Conține imagini cu formulare completate de la 3600 de persoane diferite, aproximativ 810.000 de imagini cu caractere separate de formularele din care au fost extrase, etichete pentru acestea și utilitare software pentru gestionarea acestora.

Am ales să folosesc această bază de date datorită cantității foarte mare de imagini ce pot fi folosite pentru antrenarea clasificatorului, fiind cea mai mare colecție de imagini propusă de NIST pentru procesarea documentelor scrise de mână și OCR.

Figura 14 - Exemple de imagini de antrenare NIST

- **UNIPEN** [9]**:**

Varianta originală a setului de date este folosit pentru recunoașterea de tip online a caracterelor scrise de mână. Diferența între recunoașterea de tip online și cea de tip offline, folosită în proiectul lucrării, este aceea că în recunoașterea de tip online, avem acces la mișcările mâinii, a pixului, sau a oricărui alt dispozitiv de intrare, la momentul scrierii, pe lângă imaginea propriu-zisă a caracterelor scrise. La recunoașterea de tip offline avem acces doar la imaginile cu caracterele deja scrise.

Totuși, Jorge Sueiras în lucrarea lui „*Using Synthetic Character Database for Training Deep Learning Models Applied to Offline Handwritten Character Recognition*” [9] a creat varianta pentru recunoaștere offline a acestei baze de date, variantă care este folosită în procesul de învățare a clasificatorului. În total, setul conține 62.382 imagini ale caracterelor scrise de mână.

Seturile de date nu au fost folosite în conjuncție, ci doar a fost făcută o comparație a performanțelor, cu mențiunea că, în cazul antrenării cu setul de date de la NIST, întrucât acest set nu conține decât imagini cu litere mari și mici și cifre, pentru recunoașterea caracterelor speciale „.”, „_”, și „@” au fost folosite tot imaginile din setul de date UNIPEN.

Figura 15 - Exemple de imagini de antrenare UNIPEN

3.3.2. Arhitectura rețelei neuronale

Un lucru foarte important ce trebuie învățat în domeniul Învățării Profunde este acela că nu există nicio metodă de a alege arhitectura perfectă pentru rezolvarea unei anumite sarcini. Arhitecturile rețelelor neuronale trebuie să fie mereu alese în mod empiric, adică, modificarea acestora pas cu pas,

adăugare/ștergere de straturi, modificare număr de neuroni pentru straturi, până când se obțin performanțe maxime.

Astfel, plecând de la o arhitectură de rețea foarte simplă, treptat, se ajunge la una complexă, ce aduce rezultate foarte bune pentru clasificare.

Pentru următoarele arhitecturi prezentate a fost folosit la antrenare setul de date de la NIST, setul fiind împărțit în proporție de 90% pentru învățare, și 10% pentru validare.

De asemenea, pe timpul antrenării s-au folosit următoarele callback-uri:

- Early Stopping, pentru a monitoriza costul de validare. În momentul în care valoarea funcției de cost nu mai scade, după un număr mare de epoci, apare procesul de supraînvățare, iar capacitatea de generalizare a modelului scade drastic. Astfel, acest callback, oprește antrenarea modelului odată ce valoarea funcției de cost nu a mai scăzut după un anumit număr de epoci. În proiect, se oprește antrenarea dacă timp de 10 epoci funcția de cost nu a mai scăzut.
- Model Checkpoint, pentru salvarea modelelor intermediare. Pe parcursul antrenării, se urmărește acuratețea pe setul de date de validare, iar acest callback salvează într-un fișier de tip *.h5* modelul cu cea mai bună acuratețe.
- ReduceLROnPlateau. Pe parcursul antrenării, în spațiul definit de parametrii antrenabili ai modelului este posibil să întâmpinăm un „platou” de valori, adică valorile funcției de cost se mențin constante, sau variază foarte puțin la variații mari ale parametrilor. Astfel, pentru o rată de învățare mare a modelului, este posibil să se treacă peste punctele cu valori minime. Acest callback tratează această problemă, reducând rata de învățare cu un factor de *0.1* la detectarea unui platou.

I. Rețele dense

Pentru început, am proiectat o rețea minimală, doar cu straturi dense, pentru observarea performanțelor. De obicei, pentru rețele dense, la recunoașterea caracterelor scrise de mână, se obține o acuratețe a recunoașterii de aproximativ 90%.

Modelul are următoarea arhitectură:

```
1   model = Sequential()
2   model.add(Dense(512, activation='relu'))
3   model.add(Dense(39, activation='softmax'))
```

Figura 16 - Modelul dens simplu

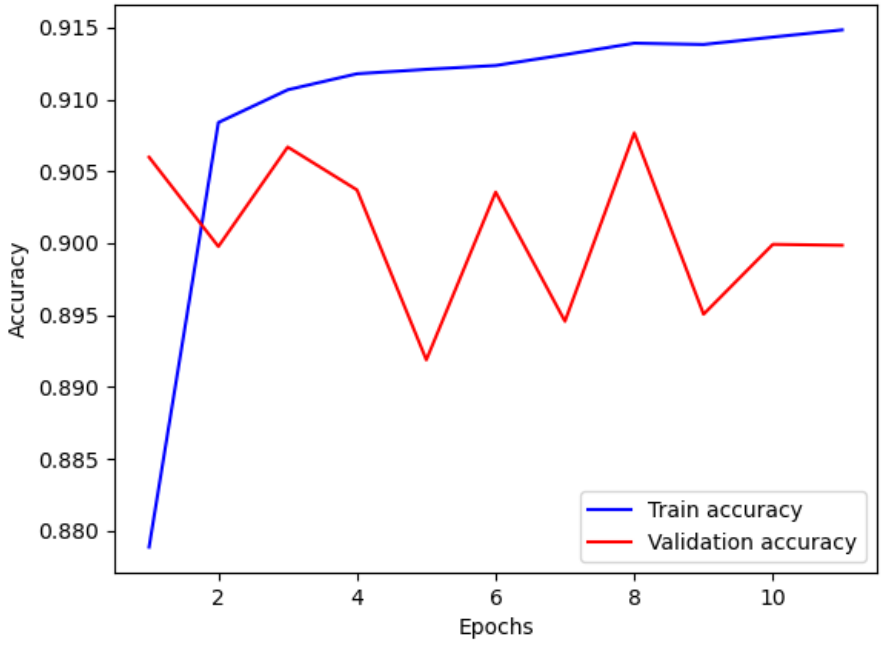


Figura 17 - Acuratețea modelului dens simplu

După cum se poate vedea în figurile 16 și 17, rețeaua începe să supraînvețe deja de la a doua epocă, obținându-se o valoare maximă a acurateței la validare de aproximativ 90%, care nu este deloc o valoare bună scopul proiectului.

II. Rețea convoluțională mică

Următorul model introduce o arhitectură de rețea mică, cu doar 2 straturi convoluționale, și unul dens.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),strides=1,activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu', strides=1))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(39, activation='softmax'))

```

Figura 18 - Rețea convoluțională mică

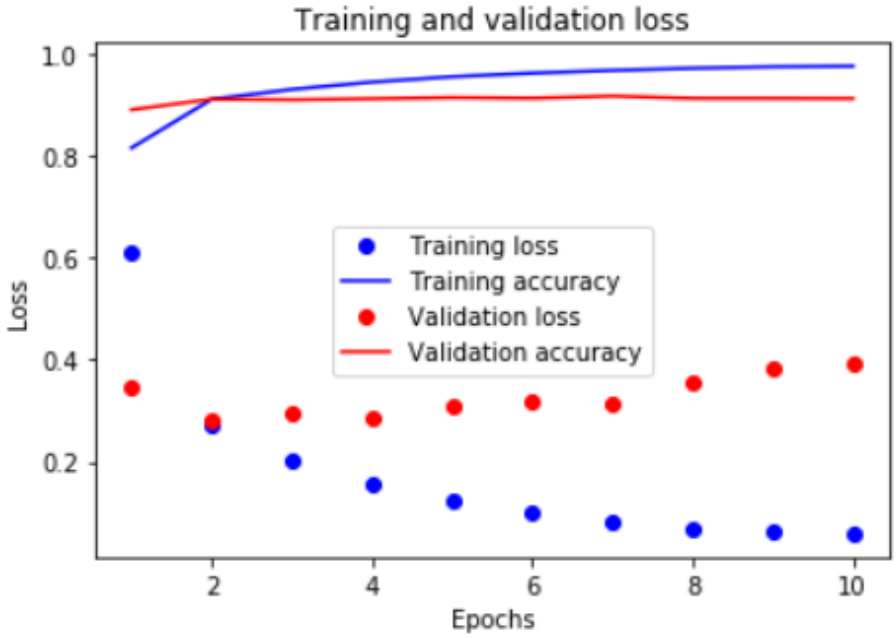


Figura 19 - Performanțele modelului convoluțional mic

Validarea modelelor trebuie făcută în două etape. Prima dată, modelul trebuie să obțină o acuratețe ridicată pe datele de antrenare. În acest caz, obținem o acuratețe pe antrenare undeva peste 95%, așadar modelul trece acest test. A doua verificare trebuie făcută pe setul de date de validare. Dacă acuratețea de antrenare și cea de validare converg către valori diferite înseamnă că modelul nu poate generaliza. Acest lucru se întâmplă și cu modelul curent.

III. Rețea convoluțională cu BatchNormalization

Acest model este la fel ca cel anterior, cu excepția straturilor BatchNormalization folosite după fiecare strat convoluțional.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), strides=1,activation='relu', input_shape = (32, 32, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', strides=1))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(39, activation='softmax'))
```

Figura 20 - Model convoluțional cu BatchNormalization

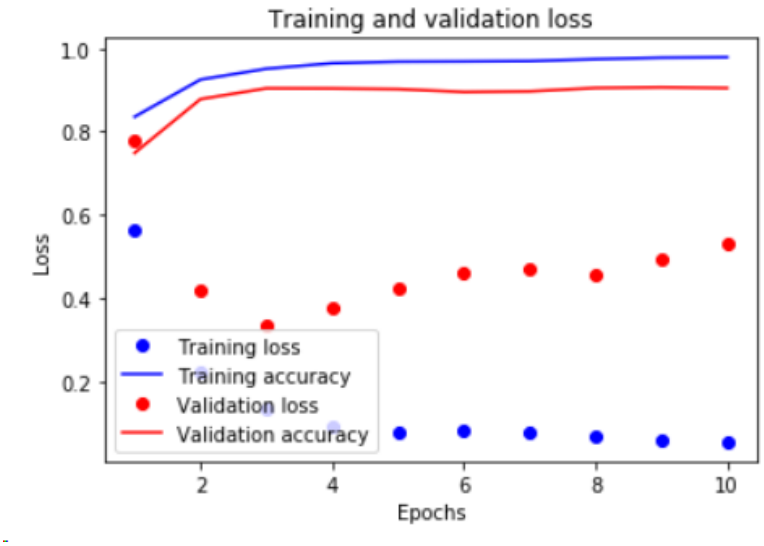


Figura 21 – Performanțele modelului convoluțional cu BatchNormalization

Modelul încă nu converge către aceleași valori la antrenare și validare. Trebuie, deci, să fie introduse straturi Dropout în arhitectură.

IV. Rețea convoluțională cu Dropout

Straturile Dropout setează valori aleatoare din input cu valoarea 0, eliminând astfel cantități de informație, împiedicând supraînvățarea. Modelul poate astfel învăța să generalizeze mai ușor, făcând posibilă convergența rețelei.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), strides=1,activation='relu', input_shape = (32, 32, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', strides=1))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dense(39, activation='softmax'))
```

Figura 22 - Modelul convoluțional cu Dropout

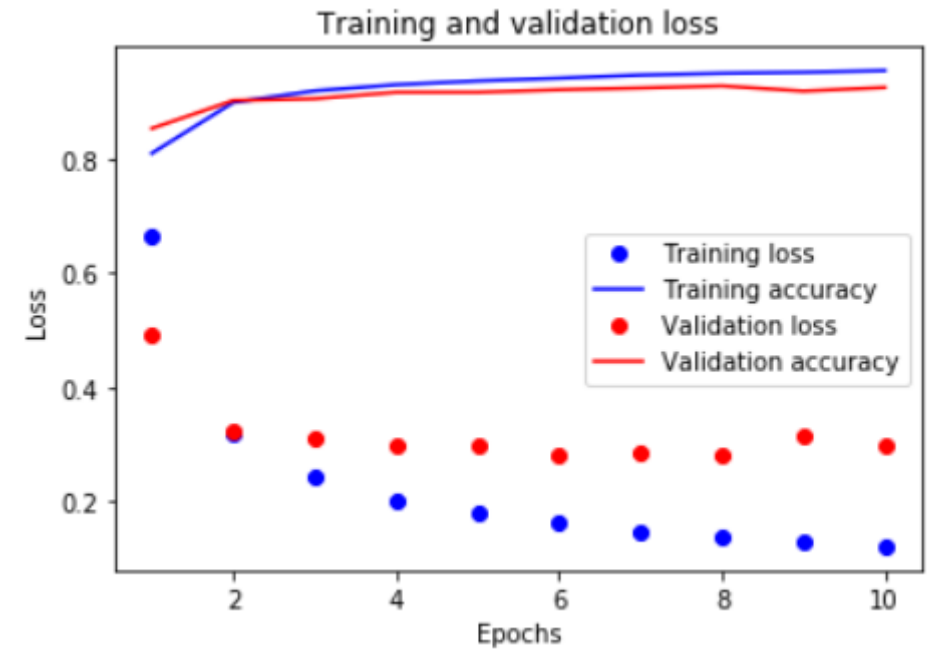


Figura 23 - Performanțele modelului convoluțional cu Dropout

Chiar dacă actualul model are o convergență mai bună, încă se pot obține rezultate mai bune. Este posibil ca diferența de performanță între acuratețe și validare să fie din vina unei rețele prea simple.

V. Modelul final

Această arhitectură de rețea a fost propusă de Chris Deotte în 2020, în competiția Kaggle, obținând o acuratețe de 99.75% [11]. În proiectul lui, Deotte folosește această arhitectură pentru a antrena în paralel 15 rețele, și folosește predicția cumulativă a acestora. În proiect, sunt implementate trei astfel de rețele, folosite individual.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(32, 32, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=5, strides=2, padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Conv2D(128, kernel_size=4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(39, activation='softmax'))
```

Figura 24 - Modelul final Kaggle

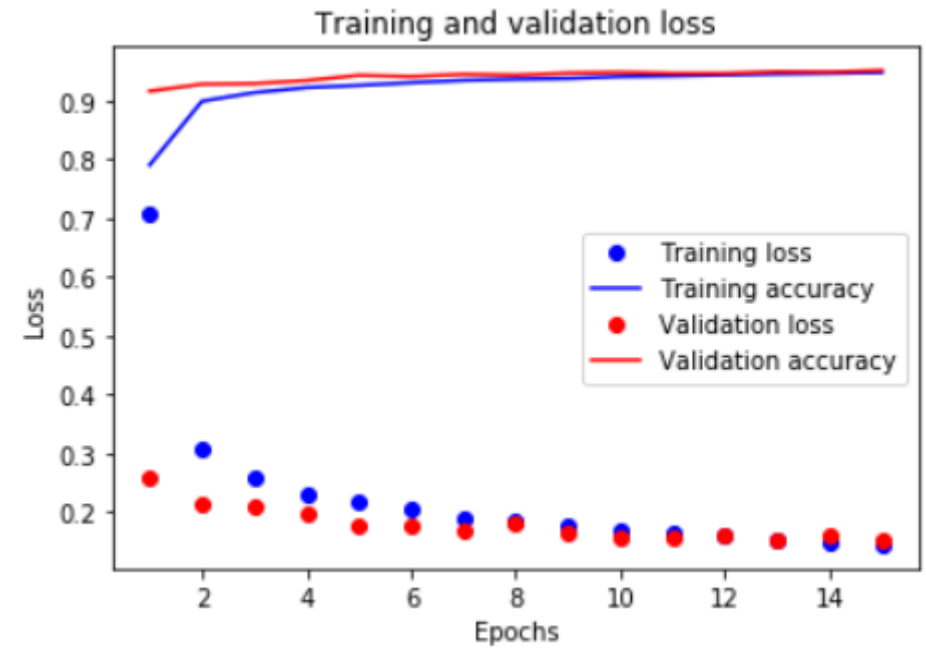


Figura 25 - Performanțele modelului Kaggle

Performanțele modelului vorbesc de la sine. Se mai poate face totuși o îmbunătățire, pentru a oferi o capacitate mai mare de generalizare, și anume augmentarea datelor de intrare.

Augmentarea datelor este un proces prin care se extinde artificial setul de date de antrenare, obținând date modificate din cele deja existente. Augmentarea se poate face atât pe imagini, cât și pe text, audio, sau orice alt tip.

Scrisul de mână suferă variații mari de la om la om, de la unghiul de înclinare a literelor, până la dimensiunea acestora. Din acest motiv, a fost implementat augmentarea pe setul de date de antrenare, îmbunătățind per total modelul.

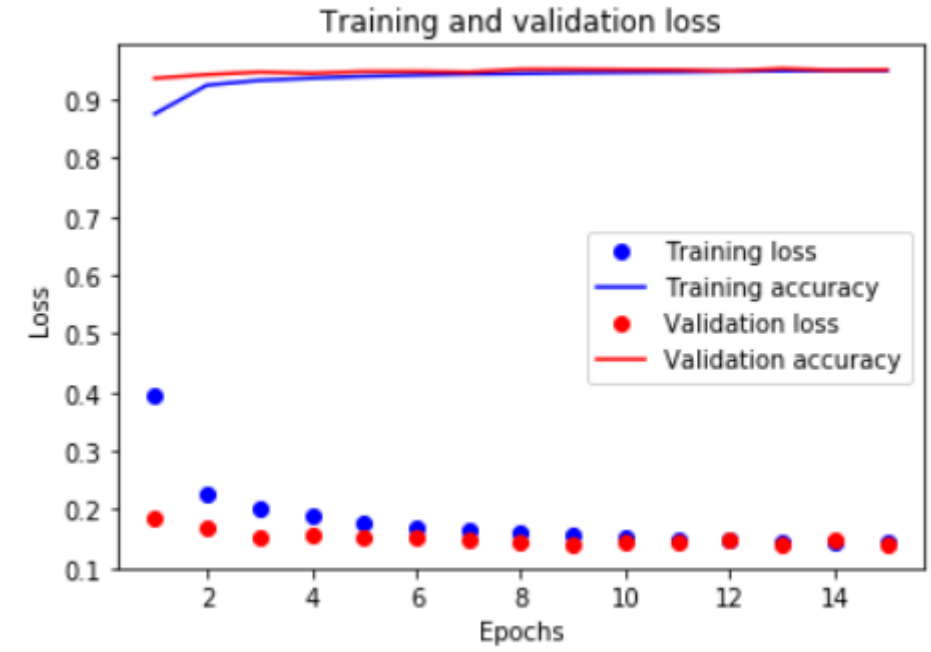


Figura 26 - Performanțele modelului Kaggle după augmentarea datelor

VI. Separarea modelelor

În practică, multe dintre caracterele alfabetului român, cifrele arabe și simboluri seamănă între ele. De exemplu, chiar și pentru oameni, este greu să diferențieze de exemplu între litera „I”, „l”, sau cifra „1”, dacă nu există context. Se poate spune ca și unui calculator ii este, cel puțin tot atât de dificil. Pentru a-i oferi acest context rețelei neuronale, am hotărât să împărțim câmpurile din formulare în trei tipuri posibile:

- Câmp text: Câmp în care pot apărea doar litere;
- Câmp numeric: Câmp în care pot apărea doar cifre;
- Câmp mixt: Câmp în care pot apărea atât cifre cât și simboluri;

Tipul pentru fiecare câmp va fi selectat de către utilizator la introducerea formularului necompletat în aplicație.

Pentru simplitate, simbolurile adăugate pentru recunoaștere sunt doar „.”, „_” și „@”. Extinderea setului de date cu alte simboluri se face relativ simplu, doar adăugând poze cu simbolul dorit în ierarhia de directoare de antrenare, în directorul *Special* al fiecărui set de date, sub un director nou creat cu denumirea codului ASCII în hexazecimal al simbolului dorit.

Din acest motiv, am hotărât să fie antrenate în paralel trei modele cu aceeași arhitectură, un model pentru recunoașterea exclusiv a cifrelor, un model pentru recunoașterea exclusiv a literelor, și un model pentru recunoașterea combinată și a simbolurilor. Diferența apare totuși la numărul de unități ale stratului de ieșire, datorită diferenței numărului de clase posibile – 10 pentru modelul numeric, 26 pentru modelul text, și 39 pentru modelul mixt.

În final, rezultatele sunt următoarele:

- Pentru modelul numeric:

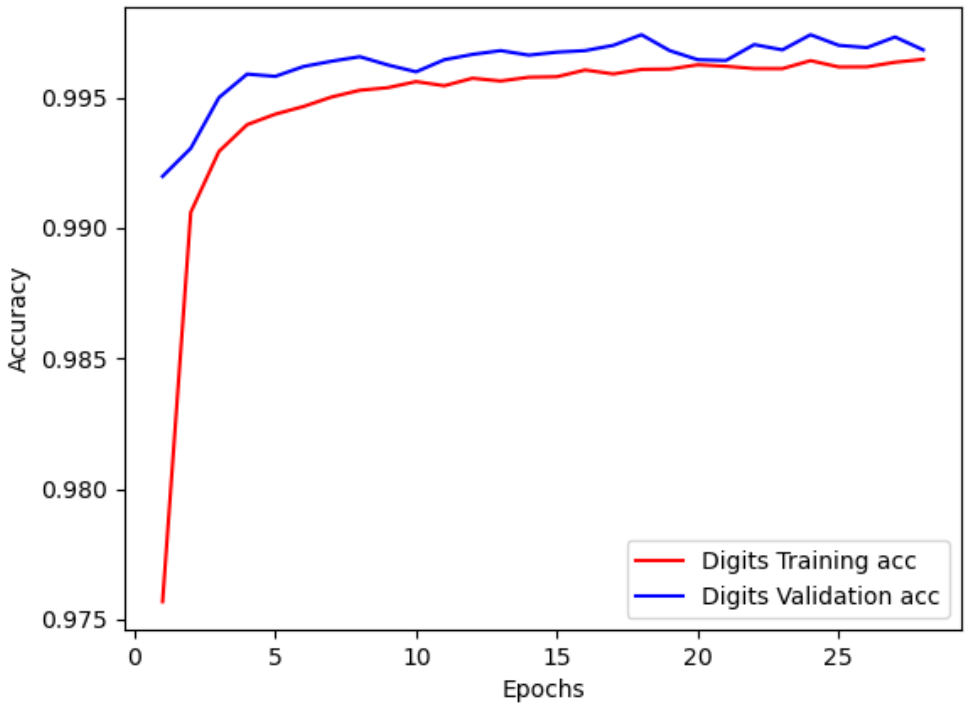


Figura 27 - Acuratețea modelului numeric - 99.74%

- Pentru modelul text:

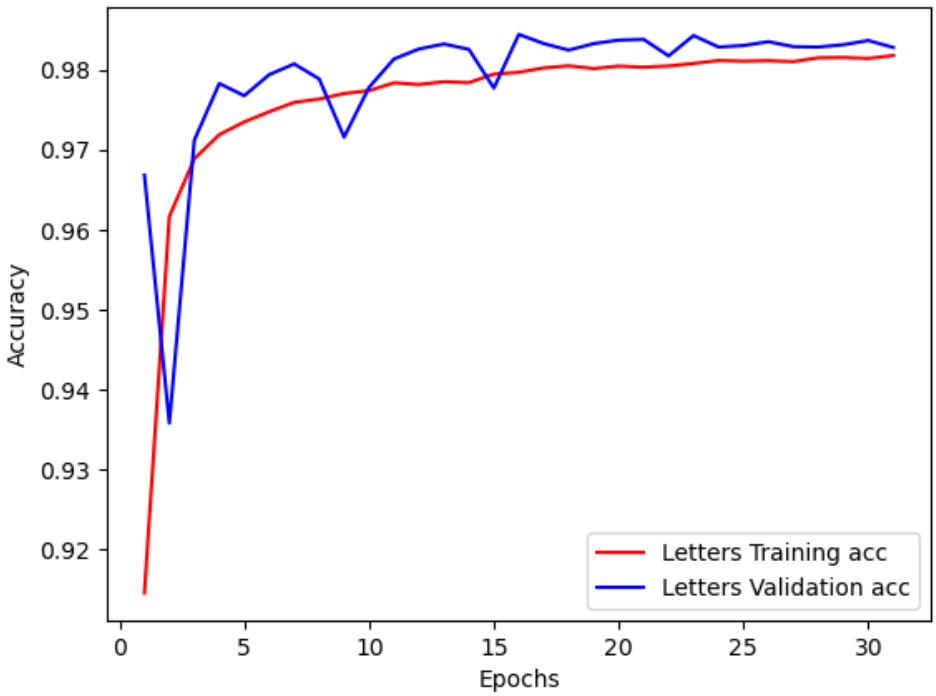


Figura 28 - Acuratețea modelului text - 99.3%

- Pentru modelul mixt:

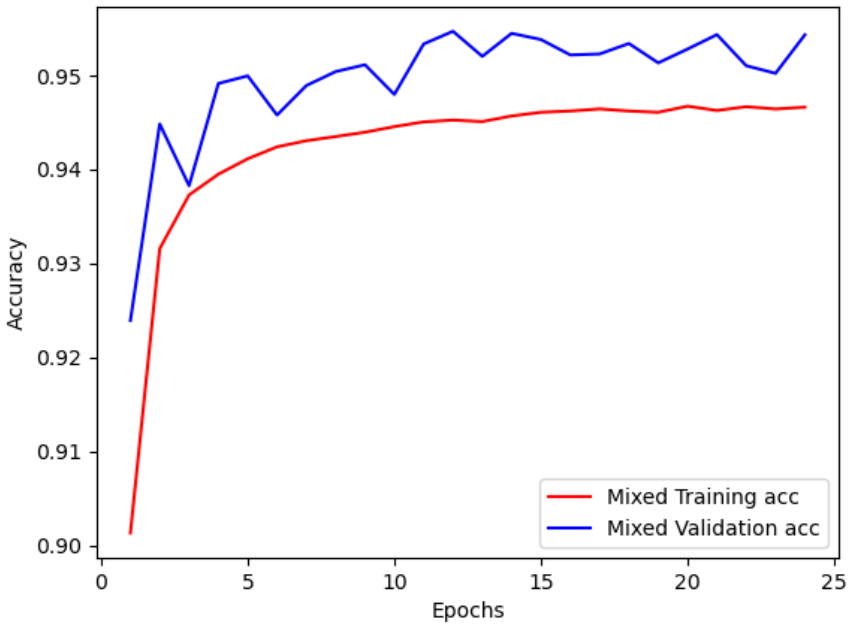


Figura 29 - Acuratețea modelului mixt - 96%

Variantele finale ale modelelor antrenate sunt salvate fiecare în fișierul propriu *.h5*, și va fi folosit la recunoașterea caracterelor în momentul introducerii unui formular completat nou.

3.4. Baza de date

Baza de date a aplicației este construită folosind utilitarul *Elasticsearch*. Elasticsearch este un motor analitic de căutare, peste o bază de date NoSQL, indexând în mod eficient toate datele care sunt puse la dispoziție. Astfel, Elasticearch pune la dispoziție modalități de căutare aproape în timp real, datorită indecșilor, și suportă următoarele tipuri de date:

- Text structurat;
- Text nestructurat;
- Date numerice;
- Date geospațiale.

În baza de date se stochează documente sub forma unor obiecte *.json*. Fiecare document este o colecție de câmpuri, fiecare având un anumit tip, și o anumită valoare. Deoarece Elasticsearch, este o bază de date de tip NoSQL, datele nu sunt organizate în tabele, ci sunt organizate pe indecși. Indecșii implementați de Elasticsearch, numiți indecși inversați, sunt niște structuri de date care suportă căutări pe text foarte rapide, având timpi de răspuns sub o secundă. Pentru date numerice sau geospațiale, nu se folosesc indecși inversați, ci arbori BKD.

Vom folosi această bază de date pentru a stoca informația despre formularele pe care dorim să le recunoaștem, și pentru a stoca datele recunoscute din documentele completate. Ierarhia bazei de date este următoarea:

- Un index principal, numit *format*, în care se reține informația despre fiecare formular, precum numărul de câmpuri, dimensiunea originală a imaginii scanate a acestuia, textul printat pe document, și diverse dimensiuni folosite pentru preprocesare;

```
{
  "_index": "format",
  "_type": "_doc",
  "_id": "5",
  "_version": 1,
  "_seq_no": 24,
  "_primary_term": 26,
  "found": true,
  "_source": {
    "id": "5",
    "Title": "Test",
    "Description": "de data asta testam",
    "Size": {
      "width": "1700",
      "height": "2200"
    },
    "Dimensions": {
      "CharBox": {
        "width": "22",
        "height": "47"
      },
      "BetweenChars": {
        "width": "14"
      }
    },
    "Zones": {
      "Zone": [
        {
          "Field": "Nume",
          "Type": "Text",
          "Coords": {
            "xmin": "0",
            "ymin": "0",
            "xmax": "0",
            "ymax": "0"
          }
        },
        {
          "Field": "CNP",
          "Type": "Numerical",
          "Coords": {
            "xmin": "0",
            "ymin": "0",
            "xmax": "0",
            "ymax": "0"
          }
        },
        {
          "Field": "Email",
          "Type": "Mixed"
```


Figura 30 - Exemplu obiect .json ce conține informația despre un tip de formular

- Indecși secundari, câte unul pentru fiecare tip de formular, ce reține informația recunoscută din documentele completate, având numele *format*X, unde X este id-ul formularului pentru care s-a făcut recunoașterea, luat din indexul principal *format*.


```
{
  "_index": "format5",
  "_type": "_doc",
  "_id": "I_Cs2HkBpbira2ABhTf7",
  "_version": 1,
  "_seq_no": 7,
  "_primary_term": 50,
  "found": true,
  "_source": {
    "Fields": {
      "Nume": "TUDOSE ALIN  ROMEO",
      "CNP": "1981111284547",
      "Email": "AL1NTU0OSE126@6MAIL.COM",
      "Telefon": "0763578387",
      "Adresa": "SLATINA 0LT ALE. TIPO6RAFULUI",
      "Adresa(cont)": "NR. 2 BL. FA25 SC. B AP. 4",
      "Nume Copil": "TUDOSE FL0RIAN OANIEL"
    }
  }
}
```


Figura 31 – Exemplu obiect .json ce conține informația recunoscută dintr-un document

Cursul de acțiune este următorul:

1. Se introduce în indexul principal noul tip de formular pentru care se dorește recunoașterea; să presupunem ca se inserează cu id-ul 1.
2. Dacă introducerea s-a efectuat cu succes, se creează indexul secundar cu denumirea *format1* unde se vor stoca informațiile recunoscute din documente completate.
3. Folosind modulul de preprocesare și clasificatorul se construiește documentul *.json* cu datele din document.
4. Se introduce în indexul *format1* documentul *.json*.

După parcurgerea acestor pași, documentul este stocat în baza de date și este disponibil pentru accesarea acestuia din interfața web.

3.5. Interfața Web

Aplicațiile desktop au reprezentat modalitatea principală de partajare a programelor software dintotdeauna, și pot fi găsite pe orice sistem, oferind orice tip de serviciu pentru rezolvarea diferitelor probleme.

În ultimele două decenii, totuși, o altă opțiune a apărut pentru utilizatorii calculatoarelor, și anume aplicațiile web. Aplicațiile web rulează în browserul preferat al utilizatorilor, funcționând ca o aplicație desktop, dar fără să impună utilizatorului să instaleze diverse programe pe stația proprie. Acestea sunt instalate într-un server web la distanță, și rezolvă cererile utilizatorilor prin acesta. Există multe avantaje pe care aplicațiile web le oferă, printre care putem enumera: mai puține cerințe de sistem, suport multi-user relativ simplu, independența de sistemele de operare, etc..

Din aceste motive, am ales ca proiectul să gestioneze nevoile utilizatorilor într-o aplicație web.

Aplicația va rula într-un server web la distanță, și va fi implementată folosind *HTML*, *JavaScript* și *PHP* [13], iar pentru gestiunea utilizatorilor, am folosit o bază de date separată *MySQL*. Pentru construirea interactivă a paginilor web și stilizarea acestora am folosit framework-ul *Bootstrap v5*, fiind cel mai popular framework pentru dezvoltarea site-urilor web interactive, oferind compatibilitate și cu browserele de pe dispozitive mobile.

Deoarece serverul web este implementat în PHP, iar baza de date Elasticsearch, modulul de preprocesare, și clasificatorul sunt implementate în python, a trebuit să fie găsită o soluție pentru interacțiunea dintre acestea. Utilizatorii framework-ului Keras sunt familiarizați cu multitudinea de mesaje de informare, sau de alertă, care sunt afișate la inițializarea și rularea programelor, și nu pot fi controlate, așadar utilizarea funcției *exec(command, output, result_code)* din PHP, care execută o comandă în terminal și întoarce în parametri acesteia conținutul afișat de program, și codul de întoarcere din program al acestuia, nu este o opțiune, din cauza imposibilității controlării textului.

Din acest motiv, am hotărât că o soluție mai ușor de gestionat ar fi folosirea mecanismului de *sockets*. Un socket este o structură dintr-un nod al unei rețele de calculatoare, folosită pentru a trimite și recepționa date din rețea.

Așadar, backend-ul aplicației va fi format din două entități ce schimbă mesaje între ele: serverul web, ce procesează cererile de la utilizatori, și la nevoie trimite cereri formatate către serverul de python, care tratează cererile de la server web, și apelează, la nevoie, funcționalitățile Elasticsearch, ale modulului de preprocesare, sau ale clasificatorului. Pentru posibilitatea deservirii mai multor utilizatori la un moment dat, serverul python este implementat multi-threaded, astfel, pentru fiecare cerere primită de la serverul web, se creează un nou fir de execuție în care este tratată cererea.

Pentru accesul la funcționalitățile oferite de aplicație, utilizatorii au nevoie de un cont, care se poate crea ușor pe pagina de înregistrare a site-ului. În urma înregistrării, și a autentificării, utilizatorul este întâmpinat de pagina de pornire, de unde poate începe adăugarea tipurilor de formulare, și recunoașterea documentelor completate.

Navigarea între funcționalitățile site-ului se poate face folosind bara de navigare din partea superioară a paginii.

3.5.1. Introducerea unui tip nou de formular

Pentru adăugarea unui nou tip de formular, mergem pe secțiunea „Formats” din navigare, unde vom fi întâmpinați de pagina de vizualizare a formularelor.

În această pagină putem vedea toate tipurile de formulare deja introduse. Pentru a adăuga un nou tip de formular, facem click pe butonul „Add a new format”.

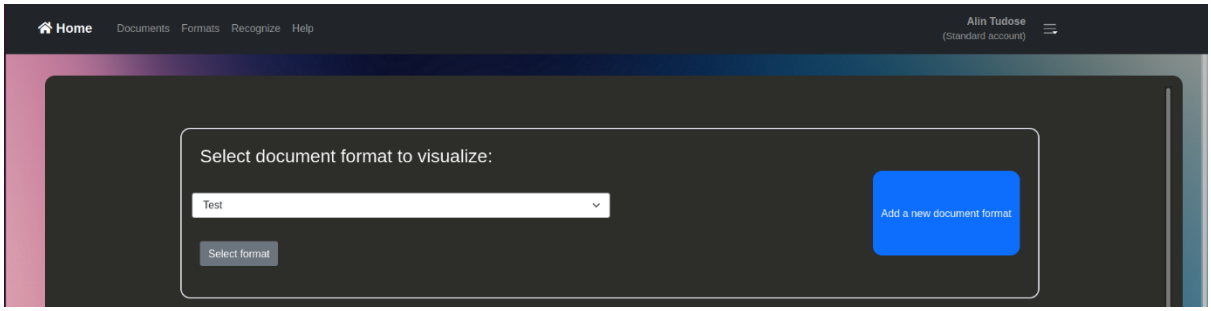


Figura 32 - Adăugarea unui nou formular

Următorul pas este încărcarea unei poze cu un formular necompletat, și completarea informațiilor despre acesta. Pentru obținerea dimensiunilor căsuțelor de caractere, și distanța dintre acestea, se pot introduce valorile exacte în pixeli, dacă se cunosc, sau se poate efectua calculul în interfața web, făcând click pe butoanele de sub fiecare câmp.

Pentru măsurarea dimensiunilor căsuțelor, în urma efectuării clickului pe butonul de măsurare, pagina așteaptă să apară două click-uri pe imagine. Aceste două click-uri trebuie să fie în două colțuri opuse ale unei singure căsuțe de caracter. Făcând o simplă scădere în modul pe coordonatele cursorului la momentele clickurilor, se obțin înalțimea și lățimea căsuței afișate. Aceste dimensiuni sunt valabile, însă, pentru dimensiunea imaginii din pagina web, care poate varia în funcție de dimensiunile ecranului. Din acest motiv, valorile obținute anterior sunt scalate la dimensiunea reală a imaginii încărcate.

Pentru măsurarea spațiului dintre căsuțe, procedeul este același, cu mențiunea că poziționarea cursorului în momentul efectuării celor două click-uri, trebuie acum să fie peste 2 colțuri adiacente a două căsuțe consecutive.

Câmpurile formularelor trebuie completate cu numele și tipul acestora – numeric, text, sau mixt – în funcție de caz. Este de o deosebită importanță ca informațiile completate despre formular să fie corecte, pentru a obține rezultate relevante în urma recunoașterii.

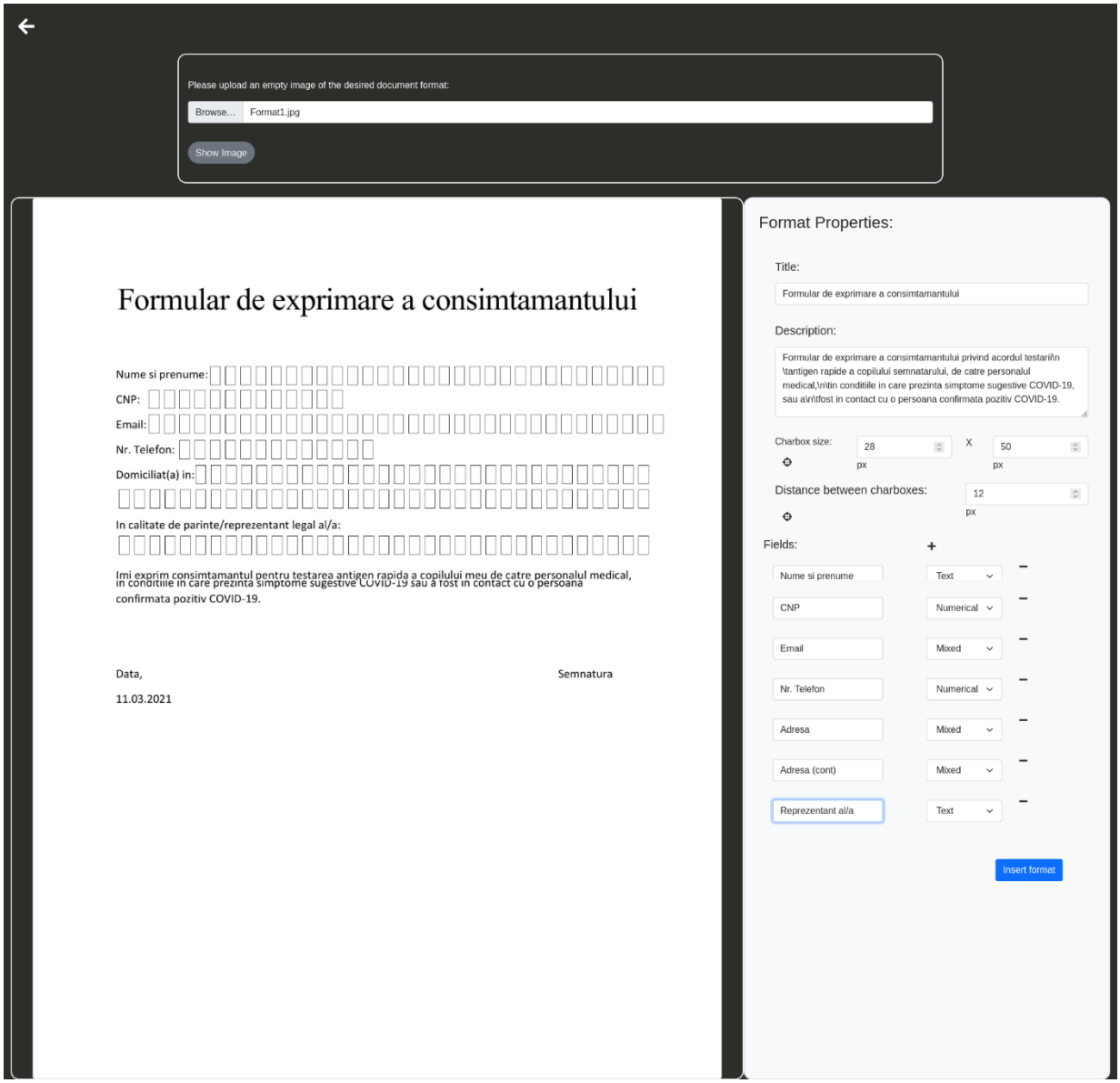


Figura 33 – Completarea informațiilor despre un tip nou de formular

În urma completării informațiilor, facem click pe butonul „Insert format”, și va fi afișat un mesaj cu statusul inserării în baza de date a formularului. Dacă inserarea s-a efectuat cu succes, putem vizualiza formularul și proprietățile acestuia, și să recunoaștem documente completate de acest tip.

3.5.2. Vizualizarea informațiilor despre un formular reținut

Vizualizarea informațiilor despre un formular este utilă în momentul în care vrem să vedem ce tipuri de date urmează să fie completate în documentele de acest tip, sau doar pentru a verifica integritatea acestuia.

Pentru a vizualiza formularul, întrocându-ne la pagina din figura 32, selectăm titlul formularului dorit, și facem click pe „Select format”.

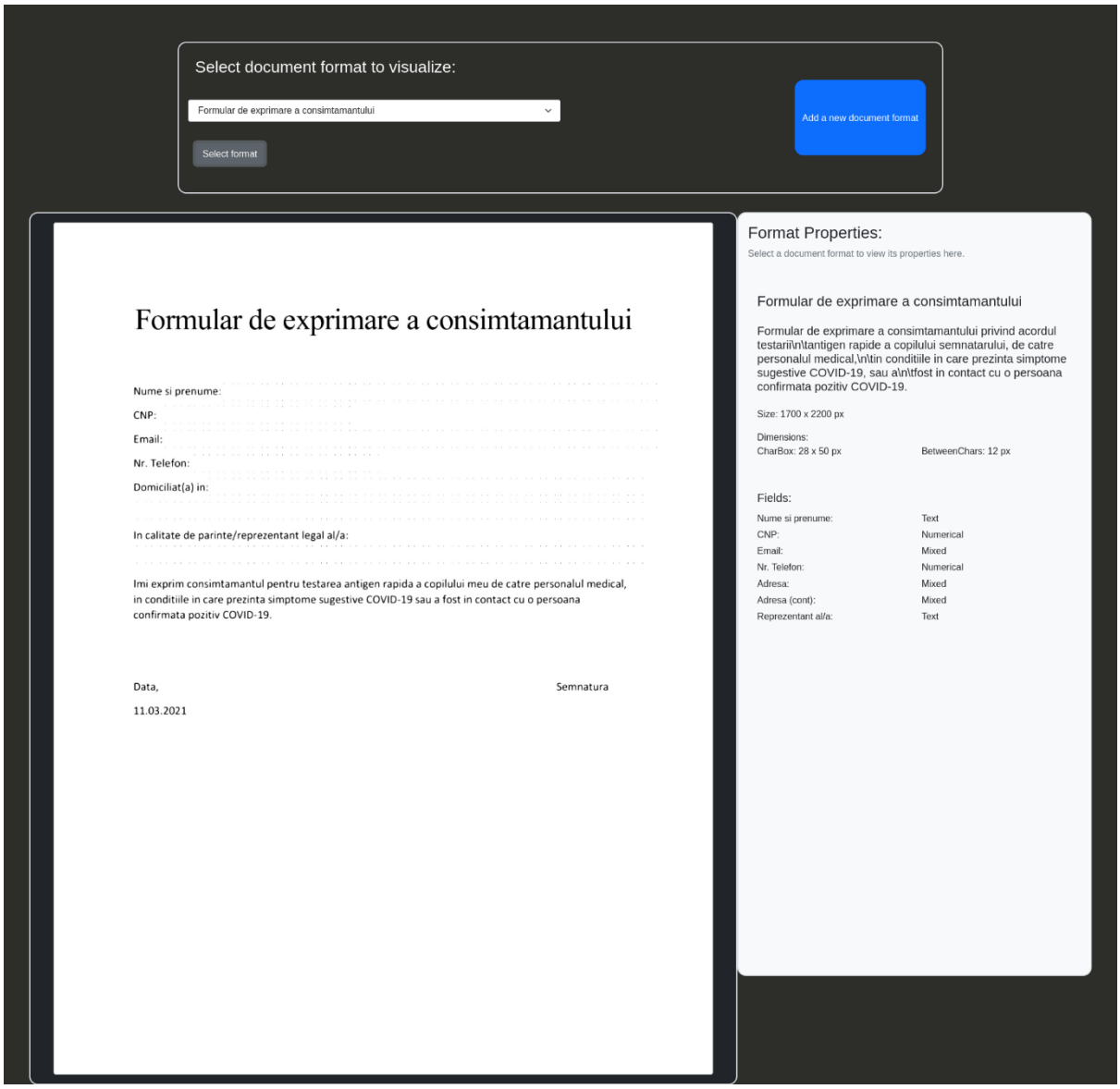


Figura 34 - Vizualizarea unui formular

3.5.3. Recunoașterea scrisului dintr-un document completat

Pentru recunoașterea unui document, mergem la secțiunea „Recognize” din bara de navigație. Procesul de recunoaștere este mai simplu decât cel de introducere a unui formular, aici fiind nevoie doar de încărcarea imaginii scanate a documentului completat, și selectarea tipului de formular. Utilizatorul poate alege dacă să îi fie afișate rezultatele recunoașterii în pagină, folosind switch-ul din dreapta paginii.

Recunoașterea este pornită făcând click pe butonul „Start recognizing”, iar după un timp necesar finalizării execuției, se afișează statusul operațiunii, și, dacă au fost solicitate, rezultatele recunoașterii.

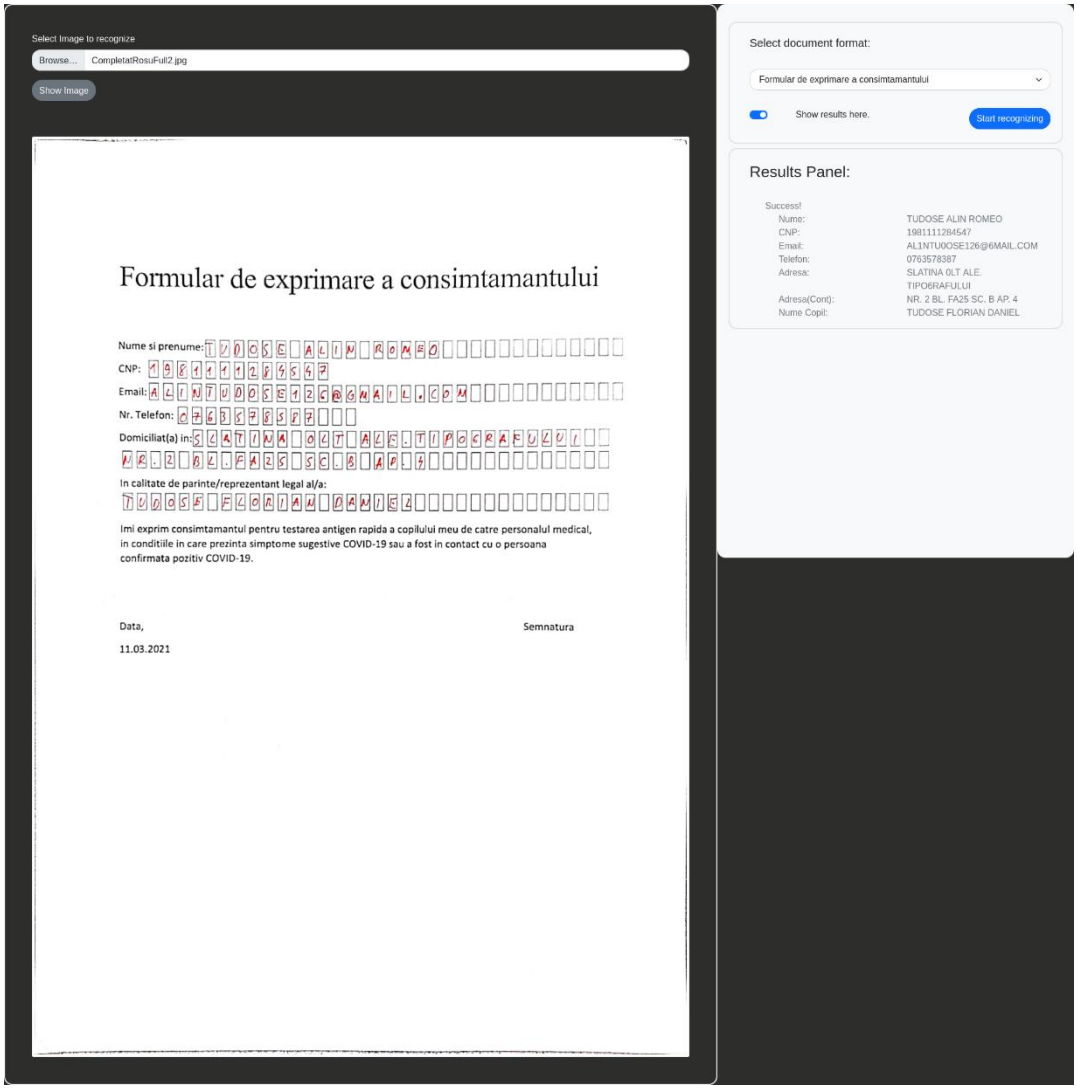


Figura 35 - Recunoașterea scrisului dintr-un document

3.5.4. Vizualizarea documentelor recunoscute deja

Vizualizarea documentelor se poate face în secțiunea „Documents” din bara de navigare. Documentele sunt afișate după formatul de care aparțin, și id-ul cu care au fost inserate în baza de date.

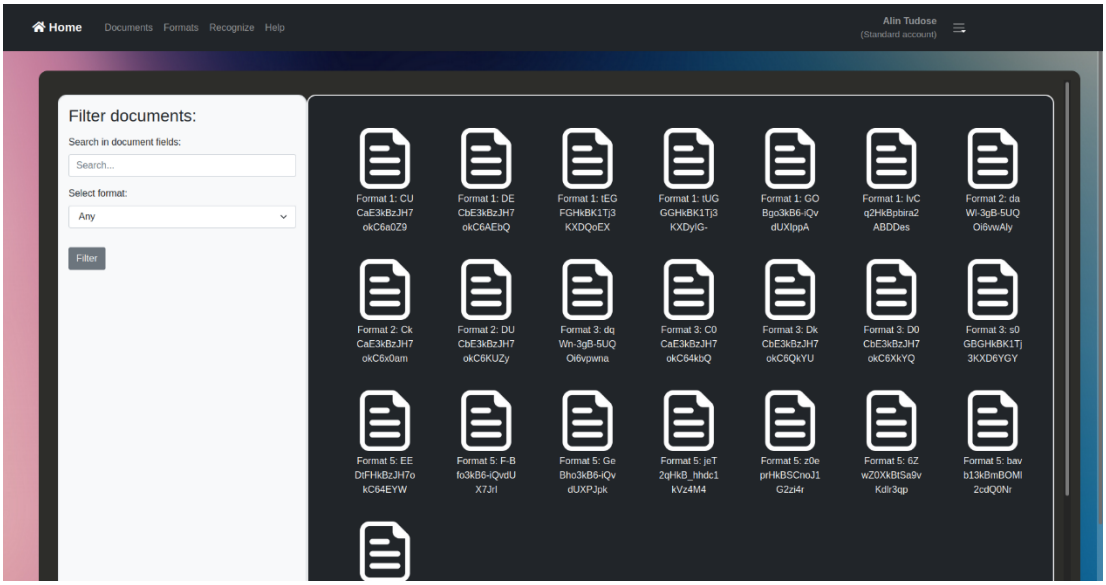


Figura 36 - Vizualizarea documentelor

În urma efectuării unui dublu click pe orice document dorit, se afișează conținutul și scrisul recunoscut.

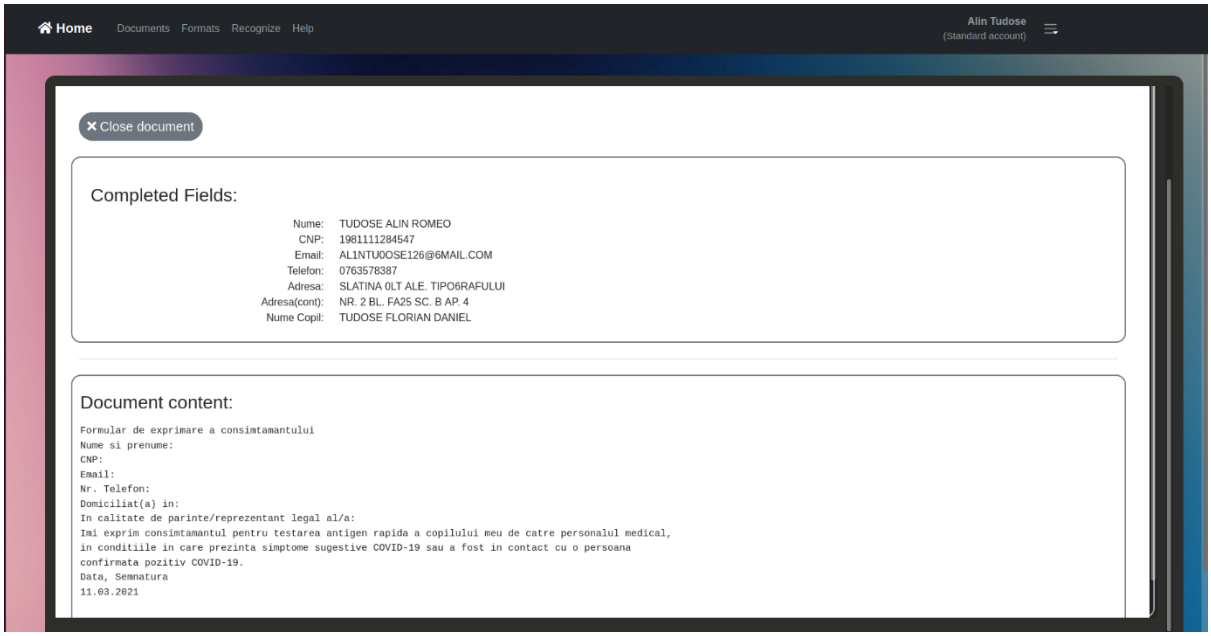


Figura 37 - Conținutul recunoscut al unui document

CAPITOLUL IV

REZULTATE OBȚINUTE

Pentru testarea aplicației au fost folosiți 5 subiecți de test, ce au urmat acțiunile descrise în secțiunea 3.5 a lucrării. Fiecare subiect a introdus un tip de formular ales, a completat un document și a introdus în aplicație imaginea scanată a acestuia.

Performanțele generale ale aplicației sunt mulțumitoare, obținând în medie o acuratețe de 90% în recunoașterea caracterelor. Pentru justificarea erorilor de clasificare, trebuie să se țină cont de două lucruri. Primul este asemănarea simbolurilor scrise de mână, de exemplu, asemănările dintre „S” și „5”, „@”, „O”, „Q” și „0”, „1” și „I”, pot fi foarte mari pentru scrisul anumitor persoane. Dacă și unor oameni le este greu să identifice corect aceste caractere, nu putem avea pretenția ca o mașină inferioară inteligenței umane să se descurce mai bine. Al doilea motiv pentru care se obține diferența de acuratețe dintre clasificator și aplicație în sine este extragerea caracterelor din imaginea scanată. Predicțiile clasificatorului depind în foarte mare măsură de forma datelor, iar dacă modulul de preprocesare nu reușește să extragă datele sub o formă care să fie asemănătoare datelor de antrenare, clasificatorul nu va reuși să recunoască în mod corect caracterele, iar performanțele au de suferit. Majoritatea erorilor de recunoaștere din aplicație provin din incapacitatea modulului de preprocesare de a extrage caracterele corespunzător.

CAPITOLUL V

CONCLUZII ȘI DIRECȚII VIITOARE

În prezent, România trece printr-un proces foarte puternic de digitalizare, pentru alinierea la nivelul de tehnologie a celorlalte țări europene. Considerând și contextul pandemiei, dincolo de problemele de ordin medical, aceasta a avut un impact pozitiv asupra pieței locale din punctul de vedere al digitalizării, accelerând procese, dar și evidențiind carențe, atât la nivelul companiilor cât și al instituțiilor statului și al individului.

Restricțiile impuse de criza sanitară au forțat toți agenții economici să adopte rapid soluții de comunicare la distanță, măsuri de securitate cibernetică, să dezvolte sistemele de plată online, dar și să proceseze diferite documente completate direct online, sau olografic și apoi scanate. Procesarea și extragerea informației din acestea se poate dovedi o activitate înceată și anevoioasă pentru angajați, de aceea, automatizarea și accelerarea procesului poate scuti timp și resurse financiare importante.

De asemenea, pentru digitalizare nu este nevoie doar de procesarea informației dintr-un anumit punct în timp înainte, ci trebuie digitalizată și informația deja existentă. Pentru documente cu cantități mari de informație, introducerea documentelor, manual, în format digital, poate dura foarte mult.

Făcând abstracție de contextul pandemiei, putem efectua un calcul rapid pentru impactul aplicației asupra profitului. Să facem urmatoarele estimări: unui angajat îi trebuie 5 minute în medie să introducă un document în baza de date. Pentru un program de muncă de 8 ore zilnic, presupunând că se muncește eficient 6 dintre acestea, înseamnă 72 de documente introduse într-o zi de muncă pentru fiecare angajat. Așadar, pentru a procesa 1000 de documente într-o zi, avem nevoie de 14 angajați.

Folosind aplicația, durează aproximativ cinci minute pentru introducerea unui formular în baza de date, și aproximativ 20 de secunde pentru extragerea informațiilor dintr-un document completat. Eficiența, din punct de vedere al timpului consumat, este foarte vizibilă, numărul de documente care pot fi procesate în 6 ore de muncă crescând la aproximativ 1200-1500, în funcție de numărul de formulare ce trebuie introduse înainte de recunoaștere, pentru

fiecare angajat. Așadar, folosind aplicația, acum, o singură persoană, poate face munca a mai mult de 14 angajați care lucrează manual. Pentru acelasi volum de muncă, ceilalți 13 angajați nu mai sunt necesari, deci se economiseste echivalentul a 13 salarii, să presupunem că sunt minimul pe economie de 2.300 de lei brut. Pentru fiecare lună, se economisesc 29.900 de lei.

Așadar, aplicația implementată rezolvă atât problema muncii anevoioase pentru utilizatori, dar aduce și un plus de productivitate și profit beneficiarilor.

Pentru dezvoltarea ulterioară a aplicației pot fi urmărite două direcții: optimizarea modulului de preprocesare, și dezvoltarea clasificatorului.

Clasificatorul actual, are performanțe foarte ridicate, obținând o acuratețe de 99.7%, 99%, respectiv 96%. Totuși, aceste performanțe sunt irelevante dacă imaginile caracterelor extrase din documente nu respectă într-o anumită măsură datele de antrenare ale clasificatorului. Pentru acest lucru, modulul de preprocesare poate fi optimizat să extragă informația într-un mod mai eficient, și mai complet. Versiunea actuală a acestuia, este condiționat de culoarea roșie a textului completat, iar porțiunile subțiri din scris este posibil să fie eliminate din imagine, ca zgomot. De asemenea, o dimensionare mai eficientă care să scaleze coerent literele, fără a le distorsiona ar putea oferi rezultate mai bune.

Pentru clasificator, ar putea fi adoptată implementarea lui Chris Deotte, cu mai multe modele CNN antrenate în paralel, și folosind pe post de predicții, răspunsul cel mai comun. Această abordare ar putea îmbunătăți calitatea răspunsurilor, având un grad de generalizare mai mare.

Personal, consider că aplicația dezvoltată are aplicabilitate ridicată în domeniul actual, oferind soluții cu tehnologii de ultimă oră, ușor de implementat și administrat.

BIBLIOGRAFIE

1. F. Chollet, *Deep Learning with Python*, Manning Publications, Shelter Island, NY, Nov. 2017.
2. H. P. Indiran, *Handwritten Character Recognition using Convolutional Neural Networks in Python with Keras*, IEEE, Asian Journal of Convergence In Technology, Tamil Nadu, India, Vol. 5 Is. III, 2020.
3. A. Graves, J. Schmidhuber, *Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks*, NIPS, München, Germania, 2008.
4. A. Poznanski, L. Wolf, *CNN-N-Gram for HandwritingWord Recognition*, IEEE, Tel Aviv, Israel, 2016.
5. Python Software Foundation , Google Tesseract-OCR, Accesibil: https://pypi.org/project/pytesseract/, Accesat la 07.06.2021 18:20.
6. Massachusetts Institute of Technology, Explained: Neural Networks, Accesibil: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414, Accesat la 11.06.2021 13:00.
7. Keras. (2016, June 18). *BatchNormalization layer*. Preluat pe March 20, 2021, de pe Keras: https://keras.io/api/layers/normalization_layers/batch_normalization/
8. National Institude of Standards and Technology, *NIST Special Database 19*, Accesibil: https://www.nist.gov/srd/nist-special-database-19, Accesat la 19.02.2021.
9. J. Sueiras, et al.,*Using Synthetic Character Database for Training Deep Learning Models Applied to Offline Handwritten Character Recognition*, Proc. Intl. Conf.Intelligent Systems Design and Applications (ISDA), Springer, 2016.
10. Medium, *EMNIST handwritten character recognition with Deep Learning*, Accesibil: https://medium.com/@mrkardostamas/emnist-handwritten-character-recognition-with-deep-learning-b5d61ac1aab7, Accesat la 19.02.2021.

11. Kaggle, *25 Million Images! [0.99757] MNIST*, Accesibil: https://www.kaggle.com/cdeotte/25-million-images-0-99757-mnist, Accesat la 21.02.2021.
12. Elastic, *Elasticsearch Guide [7.13]*, Accesibil: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html, Accesat la 03.03.2021.
13. PHP, *PHP: Documentation*, Accesibil: https://www.php.net/docs.php, Accesat la 10.03.2021.