Universitatea “Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Detecția genului persoanelor folosind rețele neurale adânci

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Electronică și Telecomunicații*
programul de studii de licență *Electronică Aplicată*

Conducător științific
Prof. Dr. Ing. Mihai Ciuc

Absolvent
Adrian-George MURGOCI

2021

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **ELA**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **MURGOCI A. Adrian-George , 445B-ELA**

1. Titlul temei: Detectie genului persoanelor folosind retele neurale adanci

2. Descrierea temei şi a contribuţiei personale a studentului (în afara părţii de documentare):
Se va proiecta in Python, folosind biblioteci dedicate (Tensorflow, Pytorch) o retea neurala capabila sa detecteze genul unei persoane dintr-o fotografie a fetei acesteia. Se vor cauta baze de date publice pentru antrenarea si testarea retelelor. Se vor investiga mai multe configuratii de retele neurale (folosind transfer learning, retele proiectate de la zero etc.).

3. Discipline necesare pt. proiect:
Programarea calculatoarelor, retele neurale si sisteme fuzzy

4. Data înregistrării temei: 2021-01-20 17:02:39

Conducător(i) lucrare,
Prof. dr. ing. Mihai CIUC

Student,
MURGOCI A. Adrian-George

Director departament,
Ş.L. dr. ing Bogdan FLOREA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **215394e727**

Anexa 5

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Detecție genului persoanelor folosind rețele neurale adânci*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică și Telecomunicații*, programul de studii *Electronică Aplicată* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, *22.06.2021*

Absolvent *Adrian-George MURGOCI*

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

DCNN - din engleză: ”Deep Convolutional Neural Network”
DNN - din engleză: ”Deep Neural Networks”
ReLU - din engleză: ” Rectified Linear Unit”
IA - Inteligență artificială
CNN - din engleză: ”Convolutional Neural Network”
API - din engleză: ”Application Programming Interface”

Introducere

Motivație

În cadrul acestei lucrări, am implementat un sistem inteligent artificial de recunoaștere automată a genului unor persoane dintr-o imagine ce conține fața acestora. Acest sistem are la bază o rețea neurală convoluțională ce a fost proiectată de la zero, urmând apoi antrenarea acesteia folosind o bază de date cu fețe umane etichetată cu genul persoanei. Modelul rezultat în urma antrenării a fost salvat și folosit mai departe pentru prezicerea genului persoanelor dintr-o fotografie ce se regăsește într-o bază de date locală sau dintr-un cadru captat în timp real de la o cameră video.

Motivul alegerii acestei teme l-a reprezentat interesul personal în multitudinea de domenii în care inteligența artificială are o aplicativitate din ce în ce mai mare și mai importantă în prezent, datorită puterii de învățare și decizie pe care o oferă, asemenea creierului uman. De asemenea, pasiunea pentru programare precum și dorința de a aprofunda și mai mult noțiunile de limbaj Python dobândite la anumite materii de-a lungul facultății au reprezentat un alt motiv în alegerea acestui proiect de licență.

Obiective

Obiectivul general al lucrării este construirea și dezvoltarea unui sistem de detectare a genului unei persoane bazat pe rețele neurale adânci, care să ofere în final o acuratețe cât mai bună.

Îndeplinirea acestui obiectiv constă în realizarea unor etape intermediare:

- **Obținerea, organizarea și preprocesarea unei baze de date:** Pentru procesul de antrenare al rețelei va fi nevoie de o bază de date etichetată corespunzător, iar imaginile transmise la intrarea rețelei vor trebui să fie de același tip și dimensiune.

- **Proiectarea și construirea modelului:** Se va defini și construi un model capabil să clasifice cât mai corect genul unei persoane.

- **Compilarea și antrenarea modelului:** Se vor defini funcțiile și parametrii specifici pentru procesul de antrenare, iar apoi se va face antrenarea modelului utilizând imaginile din baza de date descrisă anterior.

- **Realizarea unui algoritm pentru detectarea fețelor dintr-o imagine:** Se vor detecta și extrage toate fețele care apar într-o imagine.

- **Afișarea grafică a rezultatelor:** În final, se va aplica modelul antrenat pentru fiecare față extrasă anterior, iar rezultatul prezis de model va fi afișat grafic pe imaginea inițială pentru fiecare persoană în parte.

Starea artei

În prezent, inteligența artificială a devenit un domeniu prosper ce a soluționat rapid problemele dificile din punct de vedere intelectual pentru ființele umane, dar relativ simple pentru computere - probleme care pot fi descrise printr-o listă de reguli formale, matematice [1]. Dorința de copiere a funcționalității creierului uman, ce are o capacitate formidabilă de a acumula informație în timp, utilizând-o apoi pentru a lua diferite decizii, a dus la studierea și dezvoltarea de către mulți cercetători a rețelelor neurale adânci. Sistemele ce folosesc acest tip de rețele au o aplicabilitate mare într-o gamă largă de domenii precum securitate, medicină, educație, social-media și multe altele. Aceste rețele oferă o performanță foarte bună comparativ cu alte tehnologii similare, mai ales când sunt utilizate în realizarea sistemelor de detecție și clasificare de obiecte, lucru demonstrat și de un studiu publicat în jurnalul IEEE Access [2]. Autorii acestui studiu au realizat o arhitectură DCNN compactă pentru recunoașterea genului dintr-o imagine a feței, care atinge o acuratețe de ultimă generație la un cost de calcul foarte redus.

Metoda propusă în acest studiu a fost bazată pe o arhitectură de rețea neurală multifuncțională numită MobileNets [3]. Motivul din spatele acestei alegeri este că această arhitectură atinge o acuratețe ridicată în cazul recunoașterii de obiecte, pastrând totodată o latență scăzută. [2]

Model	Latency (ms)	Accuracy (%)							
		LFW	VGG val.	VGG test	UNISA-1	UNISA-2+SM	IMDB	WIKI	Adience
xception-71	623	98.50	**97.80**	96.17	**97.92**	93.25	80.17	94.97	83.66
xception-150	1363	**98.84**	97.70	**97.02**	**97.92**	**94.72**	**80.76**	95.90	**84.49**
shufflenet-0.5-64	153	97.99	96.52	96.27	93.75	91.11	80.21	94.57	83.14
shufflenet-0.5-112	199	98.46	97.00	96.69	97.66	93.25	80.61	95.44	83.95
shufflenet-0.5-224	342	98.69	97.32	96.84	96.88	94.46	80.64	95.84	84.22
shufflenet-1-224	561	98.72	97.33	96.94	96.35	94.36	80.74	**95.97**	84.27
squeezenet-224	161	98.05	96.52	96.48	95.57	90.48	80.41	94.91	82.83
squeezenet-112	63	97.89	96.30	95.91	95.83	90.16	80.20	94.62	81.80
squeezenet-64	39	97.72	95.84	95.67	94.27	88.76	79.98	94.03	81.59
proposed 64_0.5_4	**38**	97.69	95.88	95.48	91.93	86.38	79.97	93.93	81.61
proposed 64_0.5_6	56	98.08	96.46	96.10	92.97	91.41	80.29	94.79	82.68
proposed 64_0.5_8	71	98.01	96.69	96.34	94.53	92.54	80.35	95.01	82.91
proposed 64_0.5_17	113	98.25	96.70	96.30	96.61	92.06	80.41	94.94	83.23
proposed 96_0.75_4	80	97.83	96.28	95.77	93.49	86.74	80.24	94.44	82.41
proposed 96_0.75_6	104	98.32	96.69	96.45	95.31	91.18	80.51	95.12	83.31
proposed 96_0.75_8	131	98.43	96.94	96.59	97.40	92.24	80.58	95.46	83.56
proposed 96_0.75_17	209	98.55	97.15	96.73	97.40	93.04	80.66	95.67	84.48
proposed 160_0.75_4	115	97.50	95.72	95.30	87.76	82.96	80.11	94.07	81.41
proposed 160_0.75_6	226	98.29	96.81	96.35	95.83	91.00	80.51	95.34	83.12
proposed 160_0.75_8	267	98.18	96.93	96.53	95.05	92.73	80.63	95.40	83.41
proposed 160_0.75_17	341	98.73	97.18	96.86	96.35	93.58	80.74	95.78	84.45

Tabel 1: Evaluarea diferitelor arhitecturi pe diferite baze de date [2]

În tabelul 1 se poate observa o comparație între performanțele obținute cu ajutorul modelelor realizate de autorii acestui studiu și alte modele populare deja existente, performanțe obținute la antrenarea mai multor baze de date: LFW [4], VGGFace [5], MIVIA-Gender [6], IMDB-WIKI [7], Adience [8]. Modelele propuse în acest studiu sunt cele cu numele "*proposed x_y_z*", unde x se referă la dimensiunea imaginilor de intrare, y reprezintă multiplicatorul de lățime iar z este numărul de blocuri. Acuratețea cea mai bună a acestor modele propuse este de 98,73% și a fost obținută folosind baza de date LFW [4]. Pe aceeași bază de date, folosind arhitectura numită Xception [9], s-a obținut o acuratețe puțin mai ridicată (98,84%), însa timpul de procesare a fost unul foarte mare (1363 ms) comparativ cu modelul propus anterior (341 ms). Performanțe similare au fost obținute și în cazul folosirii altor modele bazate pe arhitecturile ShufflenetV2 [10] sau Squeezenet [11].

Un rezultat important de remarcat este că indiferent de arhitectura DCNN utilizată, acuratețea este una foarte bună pentru orice bază de date folosită la antrenare, atingându-se valori cuprinse între 79,98% și 98,84%. Astfel, este afirmat faptul că utilizarea unei arhitecturi de rețea

neurală adâncă într-un sistem de recunoaștere și clasificare a obiectelor duce la rezultate foarte bune în final.

Conceptul de detectare de către o mașinărie a genului unei persoane după fața acesteia poate prezenta un mare interes în multe aplicații din diferite domenii. Un exemplu îl reprezintă robotica socială inteligentă, unde percepția trăsăturilor biometrice moi este utilizată pentru a personaliza conversația și a crește senzația de inteligență percepută de interlocutorul uman. Marketingul digital este o altă aplicație în care recunoașterea genului poate fi utilizată în mod eficient, deoarece permite creșterea eficienței campaniilor publicitare; într-adevăr, în acest scenariu este posibilă înlocuirea conținutului afișat pe un ecran cu unele reclame dinamice, personalizate în funcție de genul persoanei care se uită la ecran. [2] Identificarea unei persoane într-un sistem, interfețe inteligente, supraveghere vizuală sau recunoașterea comportamentului unei persoane sunt alte aplicații în care clasificarea automată a genului are o importanță deosebită. [12]

Așadar, rețelele neurale adânci stau la baza multor sisteme inteligente datorită performanțelor foarte bune ce se obțin cu ajutorul acestora. De aceea, mulți dezvoltatori mari din industrie ce doresc să construiască sisteme și aplicații în care apare nevoia de recunoaștere și clasificare automată a obiectelor preferă sa folosească aceste arhitecturi de rețele.

Capitolul 1
Noțiuni teoretice

1.1 Istoric

Primele informații despre rețelele neurale artificiale se regăsesc în anul 1943, atunci când neurofiziologul Warren McCulloch și tânărul matematician Walter Pitts au dezvoltat primele modele de rețele neurale. Ei au publicat o lucrare numită "*The Logical Calculus of the Ideas Immanent in Nervous Activity*" în care explicau modul posibil de funcționare al neuronilor. În 1949, Donald Hebb a ajuns la concluzia conform căreia căile neuronale sunt întărite de fiecare dată când sunt utilizate, un concept fundamental esențial pentru modul în care oamenii învață. Dupa 9 ani de la această concluzie, Frank Rosenblatt a efectuat o lucrare în care a definit pentru prima dată perceptronul. Perceptronul era văzut ca un dispozitiv electronic construit în conformitate cu principiile biologice, având abilitatea de a învăța.

Rețelele de tip ADALINE (*ADAptive LInear NEuron*) și MADALINE (*Multiple ADAptive LInear NEuron*) au fost dezvoltate în anul 1960 de catre cei doi ingineri Bernard Widrow și Marcian E. Hoff. Metoda de învățare a rețelei ADALINE era diferită față de cea a perceptronului, aceasta utilizând la antrenare algoritmul LMS (*Least-Mean-Square*). Aceste rețele erau folosite la vremea respectivă pentru eliminarea ecoului în timp real de pe liniile de comunicații telefonice.

Datorită interesului minim al societății și lipsei de fonduri financiare, a urmat o perioadă în care studiile asupra sistemelor de rețele neurale nu au mai avansat spectaculos. Totuși, este de menționat anul 1974, în care Paul Werbos a dezvoltat algoritmul de învațare "*back-propagation*", însă importanța acestei descoperiri nu a fost pe deplin apreciată până în anul 1986.

În anul 1987 a avut loc prima conferință deschisă despre rețele neurale numită *IEEE "International Conference on Neural Networks"*. Un an mai târziu, a fost fondat jurnalul INNS *"Neural Networks"*, urmat în 1989 de *"Neural Computation"*. Alte lucrări ce au adus o contribuție importantă în acest domeniu au fost *"A recurrent neural network framework, Long Short-Term Memory (LSTM)"*, publicată în 1997 de Schmidhuber și Hochreiter, și *"Gradient-Based Learning Applied to Document Recognition"*, publicată în 1998 de către Yann LeCun. [13]

În prezent, s-au înregistrat progrese semnificative în domeniul rețelelor neurale - suficient pentru a atrage o mare atenție și a se finanța cercetări ulterioare. Realizări și îmbunătățiri dincolo de aplicațiile comerciale actuale par a fi posibile, oamenii de știință reușind să dezvolte cu timpul rețele neurale din ce în ce mai performante.

1.2 Neuronul

Pentru a putea înțelege conceptele din spatele primului model neuronal și mai departe a rețelelor neurale trebuie facută o analogie din punct de vedere biologic a neuronului. Neuronii reprezintă unitățile funcționale de bază ale sistemului nervos. Aceștia generează semnale electrice numite potențiale de acțiune, ceea ce le permite să transmită rapid informații pe distanțe mari. Creierul uman este alcătuit din aproximativ 100 de miliarde de neuroni. După cum se observă și în figura 1.1, neuronul este alcătuit din 3 componente principale (dendrite, nucleu, axon) și o componentă externă numită sinapsă. Dendritele sunt responsabile pentru obținerea semnalelor

primite din surse exterioare. Nucleul are rolul de a procesa semnalele de intrare și de a decide dacă un neuron ar trebui să declanșeze un semnal de ieșire. Axonul preia acele semnale procesate și le transmite mai departe sub formă de impulsuri electrice către alte celule relevante. În final, prin intermediul sinapselor se realizează conexiunea între terminațiile axonului neuronului curent și dendritele altui neuron. Astfel, o rețea neuronală poate fi definită ca un ansamblu de unități funcționale (neuroni) interconectate. [15]

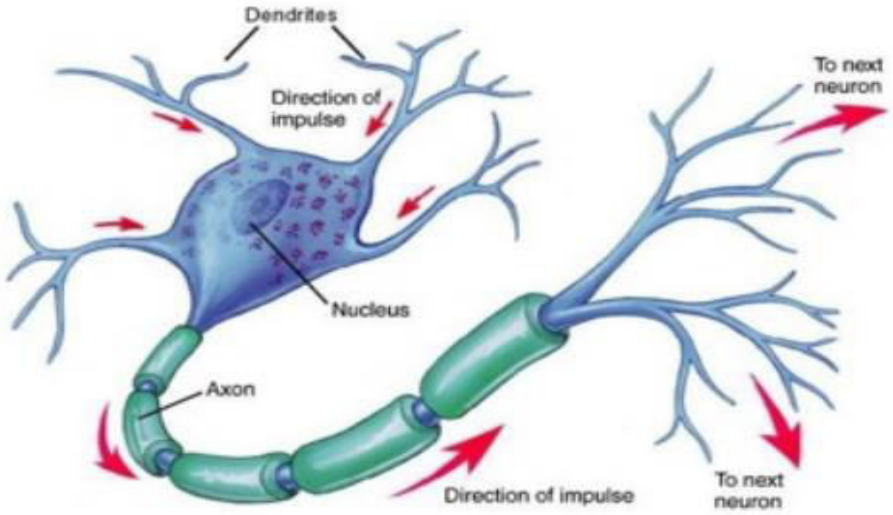


Figura 1.1: Structura neuronului [14]

Primul model matematic capabil să imite în linii mari modelul biologic neuronal a fost dezvoltat în anul 1943 de către cercetătorii Warren McCulloch și Walter Pitts. Neuronul McCulloch-Pitts a cărui arhitectură este prezentată în figura 1.2 este un neuron ce se activează binar. Ieșirea neuronului este notată cu *o* și poate fi exprimată astfel:

$$o = \begin{cases} 0, \ dacă \ \sum_{i=1}^{n} w_i \, x_i < T \\ 1, \ dacă \ \sum_{i=1}^{n} w_i x_i \ \geq T \end{cases} \qquad (1.1)$$

Intrările x_i (i = 1, 2, … , n) pot lua doar valorile 0 sau 1. Neuronul este asociat cu o valoare de prag, ceea ce înseamnă că acesta se activează în cazul în care intrarea este mai mare decât valoarea pragului sau rămâne inactiv pentru o valoare a intrării mai mică decât pragul respectiv. Ponderile neuronului pot avea valoarea -1 corespunzătoare conexiunii (sinapselor) inhibitorii sau valoarea 1 pentru conexiunile excitatorii. Singurele valori ce pot să varieze la acest model neuronal sunt cele de la intrare (x_i), ponderile și pragul având valori bine stabilite. Astfel, se poate spune ca nu există interacțiuni în interiorul modelului, mai exact nu este prezent un algoritm de învățare. Totuși, cu ajutorul acestui model se pot implementa porți logice de bază precum OR, AND și NOT. Folosind combinații între aceste 3 operații de bază se pot obține și alte porți logice cum ar fi poarta NAND, prezentată în figura 1.3, sau poarta NOR. [16]

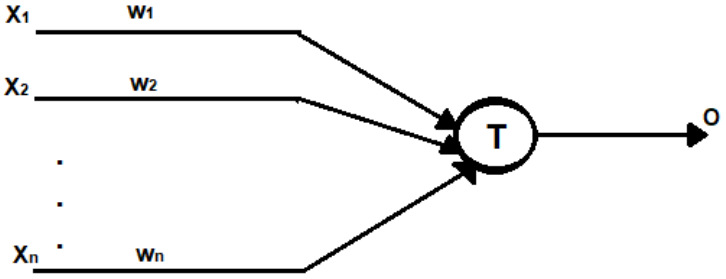


Figura 1.2: Arhitectura neuronului McCulloch-Pitts

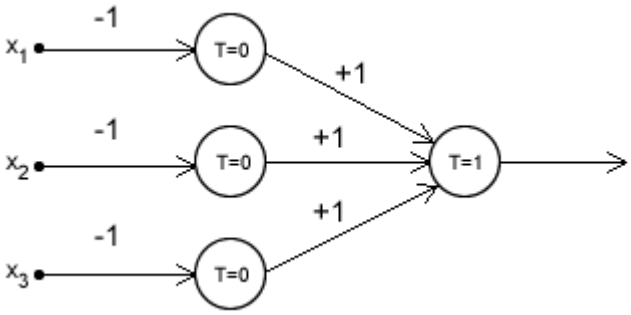


Figura 1.3: Implementarea porții logice NAND folosind modelul neuronal McCulloch-Pitts [17]

1.3 Perceptronul

Perceptronul a fost introdus în anul 1957 de către Frank Rosenblatt și reprezintă cea mai veche formă de rețea neurală. Acesta are la bază modelul neuronal McCulloch-Pitts, însă perceptronul aduce în plus un algoritm de învățare supervizată prin care ponderile se ajustează astfel încât eroarea de clasificare să fie minimă. În figura 1.4 se poate vizualiza arhitectura perceptronului, ieșirea notată cu *y* putând fi exprimată matematic astfel:

$$y = f(w_0 + \sum_{i=1}^{n} w_i x_i) \quad , unde \quad \mathrm{f(x)} = \begin{cases} 0, & dacă\ x < 0 \\ 1, & dacă\ x > 0 \end{cases} \qquad (1.2)$$

Perceptronul este considerat ca fiind cea mai simplă formă a rețelelor neuronale utilizat pentru clasificarea pattern-urilor liniar separabile. Acesta conține un singur neuron cu sinapse ce pot fi ajustate prin modificarea ponderilor și a bias-ului notat cu w_0. Spre deosebire de modelul dezvoltat de McCulloch și Pitts, neuronul nu mai este reprezentat de un prag, locul acestuia fiind luat de o funcție de activare notată *f(x)* în ecuația de mai sus. Această funcție se mai numește și funcția treaptă–unitate. Deoarece perceptronul are un singur neuron, acesta este limitat la clasificarea valorilor de intrare în doar două clase. [16]

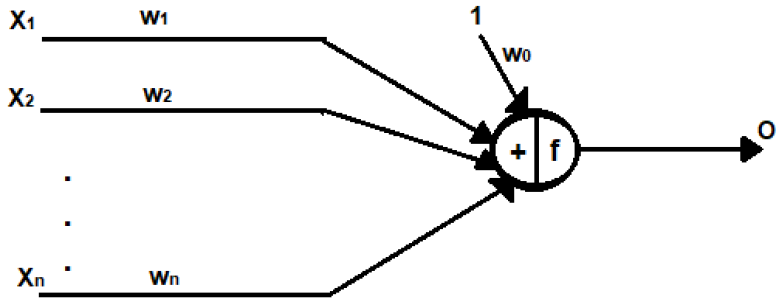


Figura 1.4: Arhitectura perceptronului

Prin acest algoritm, Rosenblatt a demonstrat că dacă două clase sunt liniar separabile, atunci perceptronul converge către o suprafață de decizie de forma unui hiperplan între cele două clase definit de ecuația:

$$w_0 + \sum_{i=1}^{n} w_i x_i = 0 \qquad (1.3)$$

Pentru cazul în care la intrarea perceptronului există doar două valori x_1 și x_2, suprafața de separare între cele două clase va fi reprezentată de o dreapta așa cum se observă și în figura 1.5.

Așadar, este demonstrat faptul că dacă se aplică pentru două seturi de vectori liniari separabili, algoritmul perceptron converge către o soluție stabilă într-un număr finit de pași. [16]

Unul dintre dezavantajele algoritmului perceptron a fost imposibilitatea găsirii unei soluții acceptabile pentru problemele neseparabile liniar precum operația logică XOR. Aceste probleme s-au putut rezolva mai târziu odată cu apariția rețelelor de tip MADALINE.

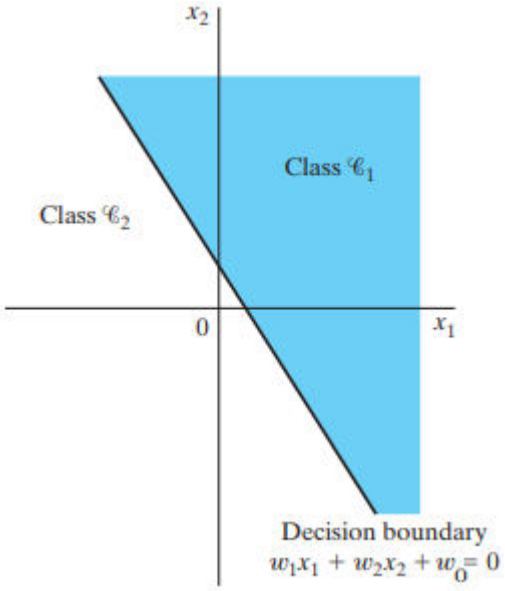


Figura 1.5: Reprezentarea suprafeței de separare între două clase [18]

1.4 Multilayer Perceptron (MLP)

Având la bază modelul perceptronului simplu, arhitectura MLP reprezintă o rețea neurală multistrat a cărei reprezentare grafică poate fi observată în figura 1.6. Această arhitectură este cu mult superioară perceptronului simplu, reușind să ofere o soluție pentru problema clasificării modelelor neliniar separabile. Cele mai importante caracteristici ale rețelelor MLP sunt:

- **Straturile ascunse:** Dupa cum sugerează și numele, o arhitectură MLP poate avea unul sau mai multe straturi intermediare situate între stratul de intrare și cel de ieșire numite straturi ascunse.

- **Conectivitatea:** Unitățile (neuronii) dintre două straturi prezintă o conexiune de tipul „toți-la-toți”, formând astfel din punct de vedere matematic un graf bipartit. [20]

- **Funcția de activare neliniară:** Fiecare neuron al rețelei dispune de de o funcție neliniară care este diferențiabilă. [21]

Cea mai folosită metodă de antrenare a rețelelor MLP este algoritmul *back-propagation*. Acesta constă în două etape generale:

→ **Etapa de *forward*** în care ponderile sinaptice ale rețelei sunt fixate iar semnalul de la intrare este transmis prin rețea din strat în strat până când ajunge la ieșire. [21]

→ **Etapa de *backward*** în care se produce un semnal de eroare rezultat în urma comparării ieșirii obținute cu răspunul real, dorit. Propagarea acestui semnal în rețea se realizează strat cu strat în sens invers, de la ieșire către intrare. Tot în această etapă se aplică ajustări succesive asupra ponderilor rețelei. [21]

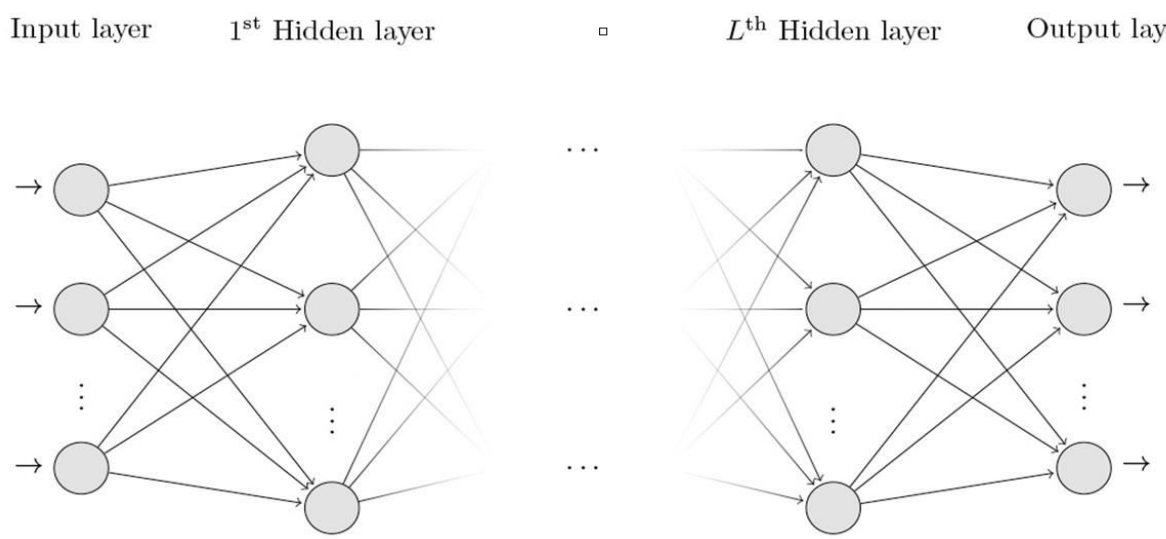


Figura 1.6: Arhitectura rețelei MLP cu un număr de „L" straturi ascunse [19]

Plecând de la informațiile prezentate în subcapitolul anterior, algoritmul *back-propagation* poate fi descris de următorii pași cu mențiunea că de data aceasta *f(x)* va fi o funcție de activare neliniară:

I. La intrarea rețelei se aplică vectorul $X_p = (x_{p1}, \dots, x_{pn})^T$ pentru care se cunoaște vectorul de ieșire ideal $Y_p = (y_{p1}, \dots, y_{pM})^T$, unde *n* reprezintă dimensiunea vectorului de intrare (numărul neuronilor din stratul de intrare) și *M* reprezintă numărul de clase. Inițial, *p* va avea valoarea 1. [22]

II. Se calculează ieșirile neuronilor din straturile intermediare cu ajutorul formulei:

$$i_{pj} = f_j^h\left(\text{net}_{pj}^h\right), \ unde \ \text{net}_{pj}^h = \sum_{i=1}^{n} W_{ji}^h * x_{pi} \ , \ j = 1, \dots, L \qquad (1.4)$$

În formula de mai sus W_{ji}^h reprezintă mulțimea ponderilor aferente stratului intermediar, *L* reprezintă numărul de neuroni din stratul intermediar iar indicele *h* provine de la cuvântul *hidden*. [22]

III. Se calculează ieșirile neuronilor din stratul de ieșire conform formulei:

$$o_{pk} = f_k^o\left(\text{net}_{pk}^o\right), \ unde \ \text{net}_{pk}^o = \sum_{j=1}^{L} W_{kj}^o * i_{pj} \ , \ k = 1, \dots, M \qquad (1.5)$$

În formula de mai sus W_{kj}^o reprezintă mulțimea ponderilor aferente stratului de ieșire iar indicele *o* provine de la cuvântul *output*. [22]

IV. Se calculează erorile pentru neuronii de ieșire pe baza relației: [22]

$$\delta_{pk}^o = \left(y_{pk} - o_{pk}\right) * f_k^o\left(\text{net}_{pk}^o\right), \ k = 1, \dots, M \qquad (1.6)$$

V. Se calculează erorile pentru neuronii din straturile intermediare cu ajutorul formulei: [22]

$$\delta_{pj}^{h} = f_j'^{h}\left(\text{net}_{pj}^{h}\right) * \sum_{k=1}^{M} \delta_{pk}^{o} * W_{kj}^{o} \ , \ j = 1, \dots, L \qquad (1.7)$$

VI. Se ajustează ponderile aferente stratului de ieșire folosind relația: [22]

$$\frac{\partial E_p}{\partial W_{kj}^{o}} = -\left(y_{pk} - o_{pk}\right) * f_k^{o}\left(\text{net}_{pk}^{o}\right) * i_{pj} = \delta_{pk}^{o} * i_{pj} \qquad (1.8)$$

VII. Se ajustează ponderile aferente straturilor intermediare cu ajutorul formulei: [22]

$$W_{ji}^{h}(t+1) = W_{ji}^{h}(t) + \eta * \delta_{pj}^{h} * x_{pi} \ , \ i = 1, \dots, n \qquad (1.9)$$

VIII. Se calculează eroarea datorată vectorului *p* de antrenare folosind relația: [22]

$$E_p = \frac{1}{2} \sum_{k=1}^{M} \left(y_{pk} - o_{pk}\right)^2 \qquad (1.10)$$

IX. În cazul în care nu s-a terminat parcurgerea întregului set de antrenare, se va introduce un nou vector la intrarea rețelei, apoi se va calcula eroarea corespunzătoare epocii curente (prin noțiunea de epocă se înțelege o parcurgere completă a întregului lot de antrenare) cu formula: [22]

$$E = \frac{1}{N} \sum_{P=1}^{N} E_p \qquad (1.11)$$

1.5 Funcții de activare

Rolul funcției de activare într-o rețea neurală este de a defini cum este transformată la ieșire suma ponderată calculată la intrarea unui neuron dintr-un strat al rețelei. [24] Această sumă ponderată, cunoscută și sub numele de potențial de activare, poate fi exprimată în cazul general astfel:

$$\text{net} = w_0 + \sum_{i=1}^{n} w_i x_i \qquad (1.12)$$

Funcția de activare se aplică acestei valori obținute *net*, determinându-se astfel ieșirea din neuronul respectiv:

$$f(\text{net}) = y \qquad (1.13)$$

Câteva dintre cele mai populare funcții de activare folosite în rețelele neurale artificiale sunt prezentate în figura 1.7. În prezent, funcțile de activare neliniare se regăsesc în majoritatea modelelor de rețele neurale adânci performante. Cu ajutorul acestor funcții neliniare se pot păstra cu ușurință valorile rezultate la ieșirea unui neuron într-o limită stabilită, în funcție de caz. Acest lucru este foarte important mai ales în cazul arhitecturilor DNN ce conțin milioane de parametrii. Folosirea unei funcții liniare în acest caz ar duce la creșterea excesivă a valorilor *y* rezultate în ecuația 1.13 și apoi implicit a puterii computaționale necesare antrenării rețelei. [24]

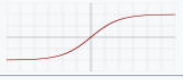
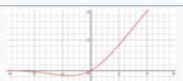
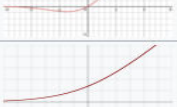
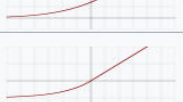
Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$
Identity		x	1
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f(x)(1-f(x))$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$
Softplus[8]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$
Softmax		$f(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$ for $i = 1, 2, \ldots K$	$f'(x) = f(x)(1 - f(x))$
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Sigmoid linear unit (SiLU,[4])		$\frac{x}{1+e^{-x}}$	$\frac{1 + e^{-x} + xe^{-x}}{(1+e^{-x})^2}$

Figura 1.7: Exemple de funcții de activare [23]

Alegerea funcției de activare poate fi influențată de mai mulți factori precum tipul rețelei, numărul final de clase sau caracteristicile bazei de date utilizată la antrenare. Cea mai frecvent folosită funcție pentru straturile ascunse este *ReLU* datorită simplității de implementare și eficienței pe care o are comparativ cu alte funcții precum cea logistică sau tangențială în combaterea problemei numită *vanishing gradients* (anularea gradienților). Marele dezavantaj al acestei funcții este că atunci când valoarea potențialului de activare este negativă, ieșirea neuronului va fi mereu nulă, ceea ce înseamnă că procesul de antrenare al rețelei va fi încetinit. Pentru a rezolva această problemă, a fost dezvoltată funcția numită *Leaky ReLU*. [24]

Pentru stratul de ieșire, în cazul în care clasificarea rețelei este una de tip binar (2 clase), este recomandată folosirea funcției logistice (*sigmoid*). Această funcție preia orice valoare reală la intrare, valoarea ieșirii fiind cuprinsă între 0 și 1. Cu cât valoarea de intrare va fi mai mare cu atât valoarea de ieșire va fi mai aproape de 1 și invers. Pentru un număr multiplu de clase, cea mai optimă alegere este funcția de activare *Softmax*, o versiune mai ”fină” a funcției argmax ce permite o ieșire asemănătoare conceptului ”învingătorul ia tot”. [24]

1.6 Funcția de cost

Funcția de cost, numită în anumite publicații și funcția de pierderi, oferă informații despre cât de bine o rețea neurală antrenează baza de date utilizată. Pe scurt, această funcție primește ca parametrii ieșirea obținută de rețea și ieșirea reală și returnează o valoare numerică (eroare) în urma comparării acestor doi parametrii. Mai exact, dacă ieșirea obținută este foarte departe de cea reală,

această eroare va fi foarte mare, iar dacă cei doi parametrii sunt aproximativ similari, eroare va fi foarte mică. Un exemplu de funcție de cost este eroarea medie pătratică definită matematic prin următoarea relație, unde *N* reprezintă numărul de neuroni din stratul de ieșire:

$$E(y_{pred}, y_{real}) = \frac{1}{2 * N} \sum_{i=1}^{N} (y_{pred} - y_{real})^2 \qquad (1.14)$$

Alegerea unei funcții de cost este necesară pentru calculul erorii în timpul procesului de optimizare a rețelei. Un model de rețea neurală este antrenat utilizând un algoritm de optimizare bazat pe tehnica gradientului negativ (*gradient descent*), ponderile fiind ajustate prin propagarea inversă a erori. [25] Această tehnică presupune găsirea minimului local sau global al unei funcții, ajutând rețeaua să învețe gradientul sau direcția în care ar trebui sa se îndrepte pentru a reduce erorile. După cum se observă și în figura 1.8, la fiecare iterație modelul converge progresiv către minim, ajungând în final la convergență, moment în care se poate spune că s-a atins o performanță de optimizare maximă. Ponderile modelului se modifică după fiecare iterație prin regula de învățare Widrow-Hoff. Această regulă presupune calcularea derivatei în funcție de ponderi și ajustarea acestora în sensul opus al gradientului: [26]

$$\frac{\partial E}{\partial w_{jl}} = \frac{\partial E}{\partial y_i} \times \frac{\partial y_i}{\partial w_{ji}} \qquad (1.15)$$

Modificarea ponderilor se va produce în final astfel:

$$w = w - \eta \frac{\partial E}{\partial w} \ , \ \eta - rata\ de\ învățare \qquad (1.16)$$

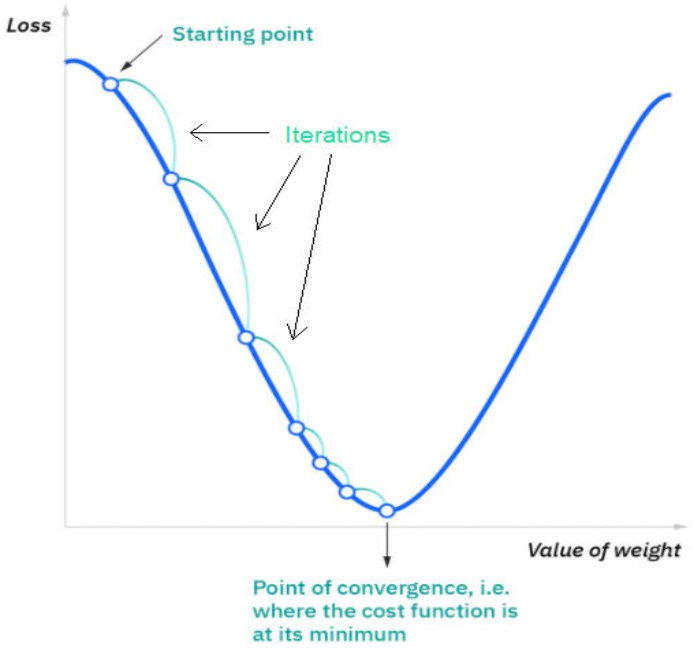


Figura 1.8: Gradientul negativ (*gradient descent*) [27]

În arhitecturile DNN moderne, cele mai populare funcții de optimizare folosite precum și o comparație realizată în cadrul lucrării științifice *"Adam: A Method for Stochastic Optimization"* [28] se pot vizualiza în figura 1.9. Se observă că funcția Adam este cea mai eficientă și performantă dintre acestea.

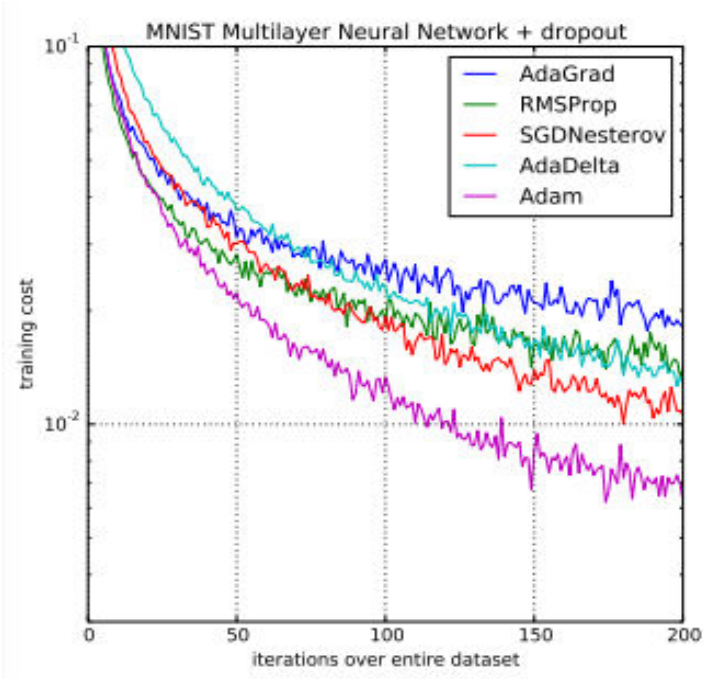


Figura 1.9: Comparație între algoritmi de optimizare antrenând un MLP [28]

1.7 Rețele neurale adânci

Creierul uman, funcțiile sale și modul în care acesta lucrează au reprezentat inspirația necesară dezvoltării domeniului de inteligență artificială [29]. Așadar, în ultimii ani, oamenii de știință au implementat diferite arhitecturi bazate pe învățarea adâncă foarte performante, capabile să clasifice cu o rată mare de succes tot felul de obiecte. Un exemplu de astfel de arhitectură este rețeaua neurală convoluțională (CNN), folosită în mare parte în subdomeniul IA numit viziune computerizată (*computer vision*).

Rețelele neurale convoluționale sunt preferate în aplicații de clasificare a imaginilor. Astfel, în general la intrarea acestor rețele se va afla așa cum se poate vizualiza și în figura 1.10 o matrice de 3 dimensiuni (lățime, înălțime și adâncime) ce conține valorile pixelilor. Dacă imaginea este de tip RGB, adâncimea va avea valoarea 3, acest număr reprezentând planurile de culoare din structura acesteia (roșu, verde și albastru).

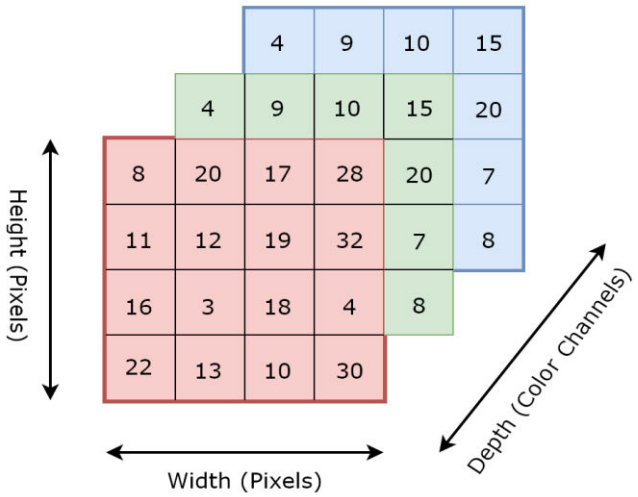


Figura 1.10: Exemplu de reprezentare a unei imagini RGB [30]

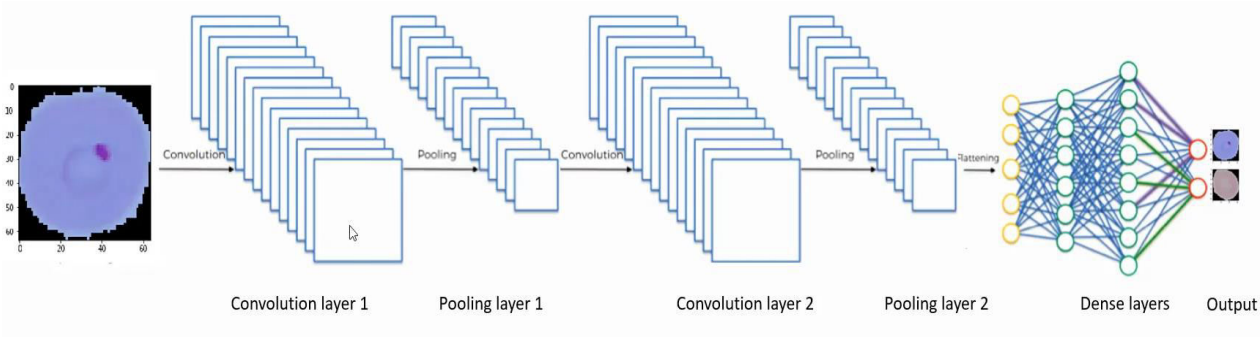


Figura 1.11: Structura unui model CNN pentru clasificarea celulelor de malarie [31]

O arhitectură CNN este alcătuită din mai multe tipuri de straturi: stratul de intrare, stratul de convoluție, stratul de pooling și stratul complet conectat (*fully-connected*). Evident, o rețea poate fi formată din mai multe straturi de convoluție sau pooling. În figura 1.11 se poate observa un exemplu de arhitectură CNN utilizată pentru clasificarea celulelor de malarie.

Fiecare tip de strat are un rol foarte important în procesul de clasificare al rețelei:

- **Stratul de intrare**

  Așa cum am menționat mai devreme, stratul de intrare va primi valorile pixelilor imaginilor ce urmează a fi clasificate sub forma unei matrici cu dimensiuni specifice.

- **Stratul de convoluție**

  După cum sugerează și numele, în acest strat se realizează procesul de convoluție între valorile primite de la stratul de intrare și anumite valori (ponderi) reprezentate sub formă de filtre (*kernels*). În urma realizării acestui proces se obțin mai multe harți de caracteristici (*feature maps*). Așa cum se observă și în figura 1.12, o matrice de dimensiuni prestabilite ce conține valori ale pixelilor este extrasă din imaginea primită de la intrare, urmând apoi aplicarea filtrului respectiv pe bucata extrasă. Este important de menționat că valorile acestor filtre au propietatea de a învăța și a se ajusta în funcție de cum evoluează procesul de antrenare al rețelei. În final, valoarea obținută în acest proces de convoluție va fi transmisă funcției de activare aleasă, determinându-se astfel ieșirea din acest strat. [20]

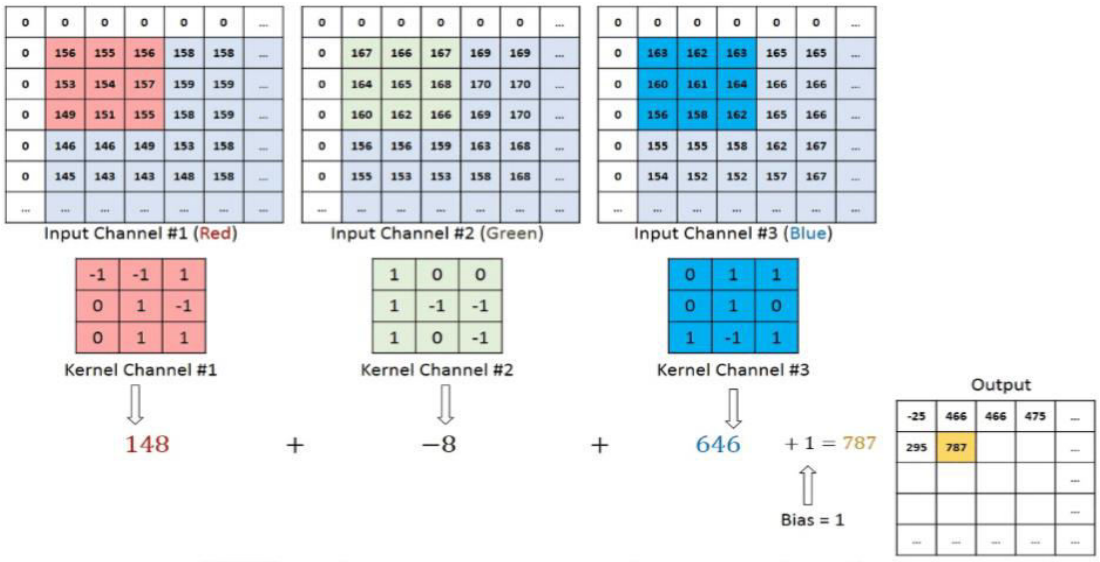


Figura 1.12: Reprezentarea procesului realizat în stratul de convoluție pentru o imagine de tip RGB [32]

- **Stratul de pooling**

În general, acest strat se introduce în arhitectura rețelei după un strat convoluțional. Rolul lui este de a reduce numărul foarte mare de parametrii prezenți în rețea, păstrând în același timp caracteristicile dominante identificate în stratul anterior. Un alt rol al acestui strat aplicat într-o rețea care lucrează cu imagini este de a reduce zgomotul prezent în acestea sau diferite puncte (pixeli) nesemnificative procesului de clasificare. Cea mai des utilizată metodă în acest strat este cea numită *max-pooling*. Această metodă constă în aplicarea unui filtru de dimensiune mică (de obicei 2x2 sau 3x3) ce calculează valoarea maximă a zonei pe care este suprapus. Un exemplu al acestei operații este prezentat în figura 1.13 unde pasul (*stride*) se referă la lungimea deplasării pe care acest filtru o realizează în acel strat de caracteristici (*feature map*).

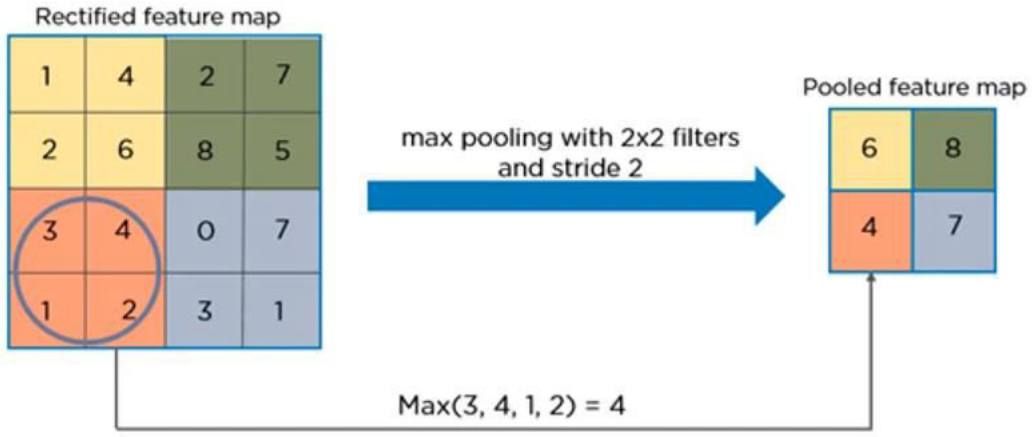


Figura 1.13: Operația de max-pooling [31]

- **Stratul complet conectat**

Acest strat este alcătuit la rândul lui din mai multe straturi. Primul este stratul numit *flatten layer* rezultat prin aplatizarea valorilor de ieșire ale stratului de pooling anterior (figura 1.14). Ultimul strat va fi cel de ieșire, corespunzător claselor modelului, iar între aceste două straturi se pot afla unul sau mai multe straturi dense (*dense layers*) cu propietatea că toți neuronii prezenți aici vor avea o conexiune de tip toți-la-toți (*all-to-all connected*).

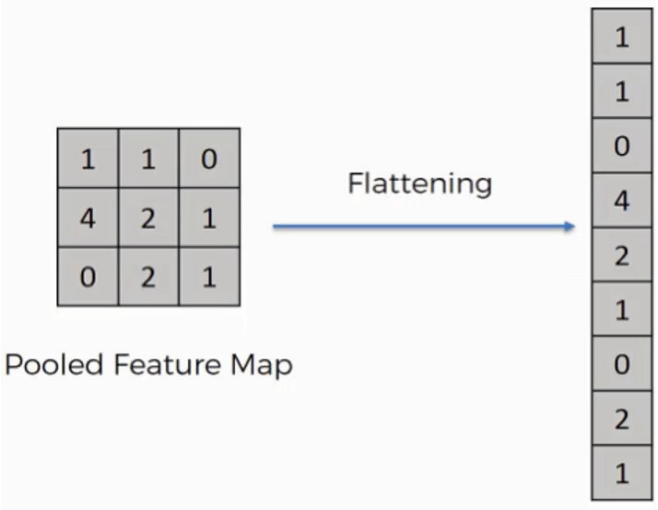


Figura 1.14: Operația de flattening [31]

1.8 Arhitecturi CNN

Cu timpul, cercetătorii au dezvoltat tot mai multe arhitecturi CNN cu scopul de a oferi performanță la o putere computațională cât mai mică. În continuare sunt menționate cronologic câteva dintre aceste arhitecturi populare:

- **LeNet-5**

  Acest model a fost dezvoltat în 1998 de către Yann Lecun cu scopul de a identifica cifrele scrise de mână pentru recunoașterea codului postal. Arhitectura LeNet-5 este reprezentată în figura 1.15 și poate fi privită ca un punct de start în domeniul CNN. [33]

- **AlexNet**

  Arhitectura AlexNet dezvoltată de către Alex Krizhevsky în anul 2012 a adus o popularitate mare domeniului numit *computer vision* și implicit rețelelor neurale adânci. Modelul este prezentat în figura 1.16 și conține aproximativ 60 milioane de parametrii. [33]

- **VGG-16**

  Rețeaua VGG a fost dezvoltată în anul 2014 de către Karen Simonyan și Andrew Zisserman. Aceasta reprezintă o variantă mai complexă față de arhitecturile anterioare, așa cum se observă și în figura 1.17. Numărul de parametrii aproximativ al rețelei este de 138 milioane. [33]

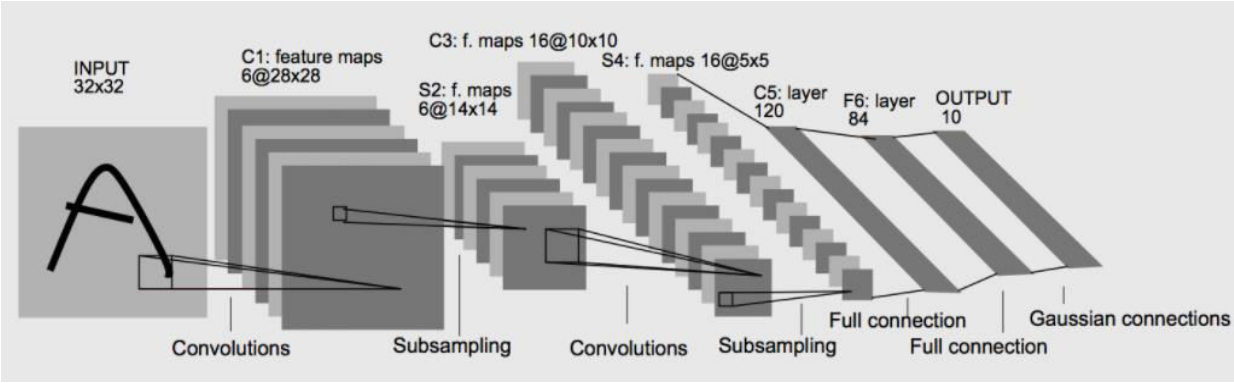


Figura 1.15: Arhitectura LeNet-5 [33]

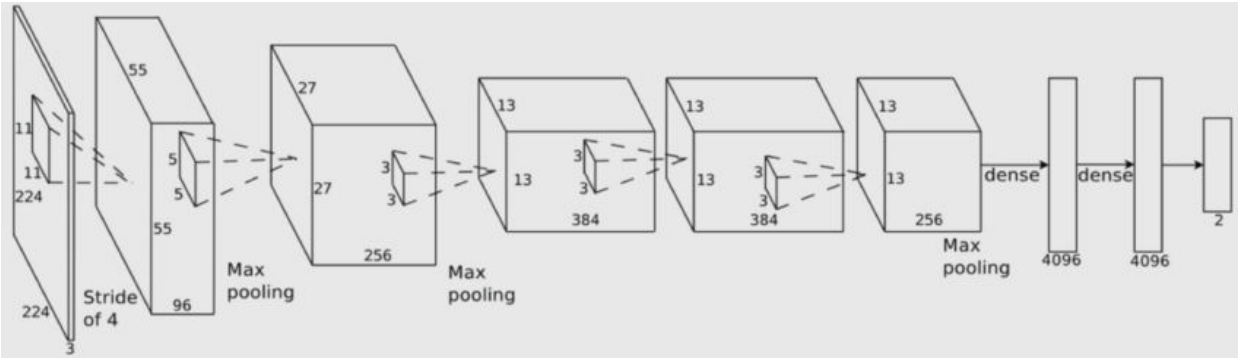


Figura 1.16: Arhitectura AlexNet [33]

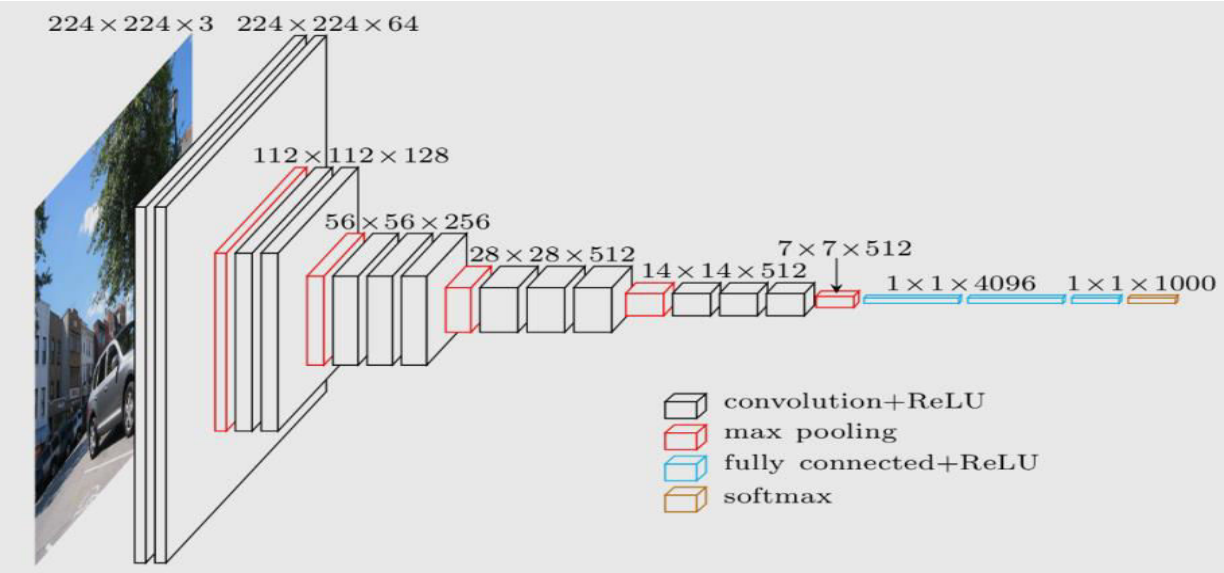


Figura 1.17: Arhitectura VGG [33]

- **Inception-v1 (GoogLeNet)**

Această arhitectură dezvoltată de cercetătorii companiei Google în 2014 a introdus noțiuni noi precum convoluția 1x1 sau modulul *Inception* cu scopul de a minimiza numărul de parametrii al rețelei (4-5 milioane). Comparativ cu arhitecturile anterioare, gradul de adâncime al acestei rețele prezentate în figura 1.18 este mult mai mare. [33]

- **ResNet-50**

Arhitectura ResNet50 prezentată în figura 1.19 a fost dezvoltată în 2015 de către cercetătorii: Kaiming He, Xiangyu Zhang, Shaoqing Ren și Jian Sun. Acest model folosește conceptul de batch normalization și nu conține straturi fully-connected iar performanța obținută cu ajutorul aceste rețele este foarte ridicată. [33]

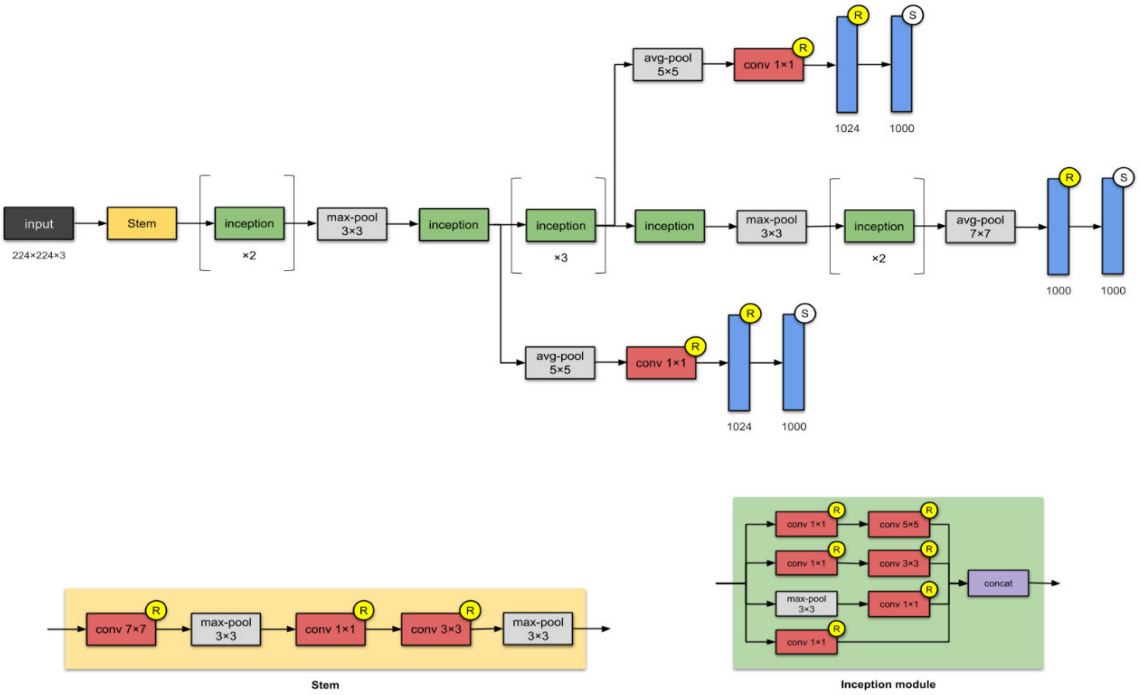


Figura 1.18: Arhitectura GoogLeNet [34]

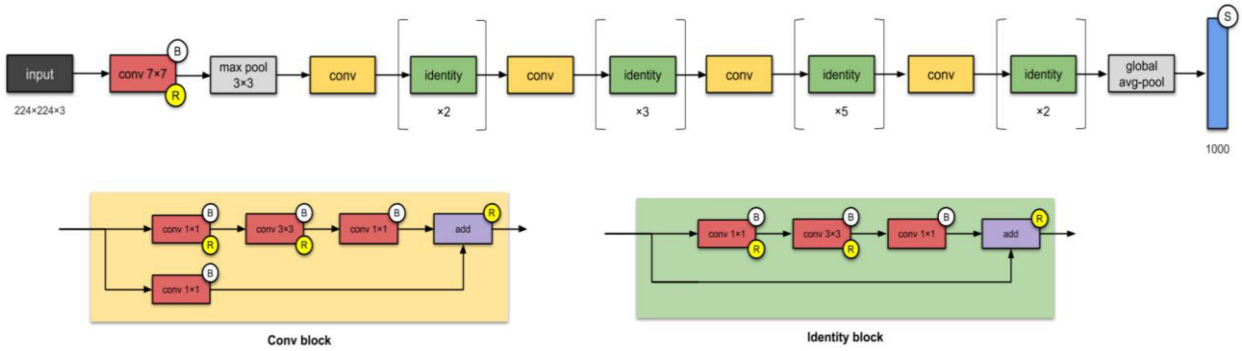


Figura 1.19: Arhitectura ResNet50 [34]

1.9 Transfer Learning

Implementarea și antrenarea unei rețele de la zero poate reprezenta un proces de lungă durată mai ales când baza de date folosită este una imensă. Din fericire, există o metodă numită *transfer learning* (transfer de cunoștințe) ce permite reutilizarea unei rețele antrenată de la zero și pentru alte aplicații. Acest procedeu funcționează deoarece straturile de bază ale unei astfel de rețele sunt aparent extractoare de caracteristici generale. În cadrul metodei de transfer al cunoștinței, se ia o astfel de rețea, se elimină stratul de ieșire și în locul lui se folosește un nou strat cu ponderi specifice aplicației curente. Apoi, în acest context rețelele antrenate de la zero sunt numite rețele pre-antrenate. [20]

Există două moduri de exploatare a unei rețele pre-antrenate:

- **Fixed Feature Extraction:** se înlocuiește și se reantrenează doar stratul de ieșire cu baza de date curentă. Restul rețelei va rămâne fixă, nu se va schimba nimic la ea. Acesta este cel mai simplu și rapid mod de implementare a metodei *transfer learning*. [20]

- **Fine-Tuning:** această modalitate este mai complexă și implică reajustarea întregii rețele, pe lângă reantrenarea stratului final. Procesul durează mai mult decât modul anterior, însă în final se obține o acuratețe de predicție mai mare. [20]

Capitolul 2
Mediul de dezvoltare

În procesul realizării acestui sistem inteligent de detecție a genului unei persoane au fost necesare anumite tehnologii și resurse software ce vor fi descrise în subcapitolele următoare. Codul pentru acest sistem a fost scris în totalitate cu ajutorul limbajului de programare Python, folosind diferite funcții preluate în principiu din două librării populare: TensorFlow și OpenCV. Baza de date folosită este una publică și conține aproximativ 58700 imagini cu fețe clasificate după genul persoanei. Toate operațile necesare acestui proiect au fost realizate pe dispozitivul de lucru personal; astfel am avut la dispoziție următoarele resurse hardware: procesor Intel Core i5-7300HQ, placa video NVIDIA GeForce GTX 1060 6GB și memorie RAM 16GB. În timpul procesului de antrenare al rețelei componenta cea mai utilizată a fost procesorul.

2.1 Python

Python este un limbaj de programare de nivel înalt, open-source și ușor de folosit. El a fost dezvoltat de către Guido van Rossum la începutul anilor 1990 în cadrul unui institut de cercetare din Olanda. Python este un limbaj interactiv și complet obiect-orientat ce oferă caracteristici precum clase, moșteniri și obiecte. Totodată, acest limbaj este considerat unul de tip portabil deoarece poate fi rulat pe platforme hardware și software majore cu puține schimbări în codul sursă. Librăria standard care vine la pachet cu limbajul Python este una extrem de utilă și foarte ușor de înțeles datorită sintaxelor scurte ce se aseamănă cu limba engleză. [35]

Python poate fi utilizat pe orice versiune recentă a sistemelor de operare Windows, MacOS sau Linux. Cele mai folosite medii de dezvoltare existente pe Windows specifice limbajului Python sunt: PyCharm, Spyder și Thonny. În cadrul acestei lucrări am folosit mediul de dezvoltare Spyder cu distribuția Anaconda versiunea 3.8.

Acest limbaj de programare este foarte folosit în realizarea proiectelor ce implică: dezvoltare web, interfețe grafice, programare baze de date, analiză și procesare de imagini, aplicații și jocuri mobile, învățare automată, analiză de date, etc. [35] Motivul pentru care Python a ajuns cel mai popular limbaj în domeniul învățării automate și al viziunii computerizate a fost dezvoltarea unor librării specifice precum PyTorch, TensorFlow, Keras, OpenCV și multe altele.

2.2 TensorFlow

TensorFlow este o librărie open-source dezvoltată în anul 2015 de către Google folosită în realizarea aplicaților și modelelor bazate pe învățarea automată. Datorită nivelului de performanță și flexibilitate oferit, TensorFlow este folosit în cadrul unor mari companii internaționale precum: Intel, PayPal, Twitter și Lenovo. [36]

Unul din conceptele de bază cu care această librărie lucrează este tensorul. Din punct de vedere matematic, un tensor este ”o funcție ce manipulează funcții vectoriale” [20]. În Python, tensorii reprezintă matrici multidimensionale ce pot conține orice tip de date cunoscut în programare (numere întregi, numere reale, șiruri, valori de tip boolean, etc). Tensorul oferit de librăria TensorFlow este într-un fel similar cu matricea din librăria clasică NumPY, de aceea există

și o metodă prin care se pot converti în ambele sensuri aceste structuri de date. [37]

Cele mai importante caracteristici ale unui tensor sunt: [37]

- **Forma:** Lungimea (numărul de elemente) a fiecărei dintre axele tensorului.
- **Rangul:** Numărul de axe ale tensorului. Un scalar are rang 0, un vector are rang 1, o matrice are rangul 2.
- **Axa:** O dimensiune particulară a unui tensor.
- **Mărimea:** Numărul total de obiecte al tensorului.

Operațiile de bază matematice (adunare, scădere, înmulțire, împărțire, modulo) precum și multe alte operații standard ale limbajului Python (max, argmin, greater, etc) pot fi folosite și în cazul tensorilor. Cu alte cuvinte, tensorii pot considerați în unele cazuri vectori, matrici sau chiar scalari. Un exemplu de modalitate de reprezentare grafică a unui tensor poate fi observat în figura 2.1.

Instalarea Tensorflow se poate face cu ușurință în cadrul mediului de dezvoltare Spyder, folosind următoarele comenzi în prompt-ul Anaconda:

```
pip install --upgrade pip
pip install tensorflow
```

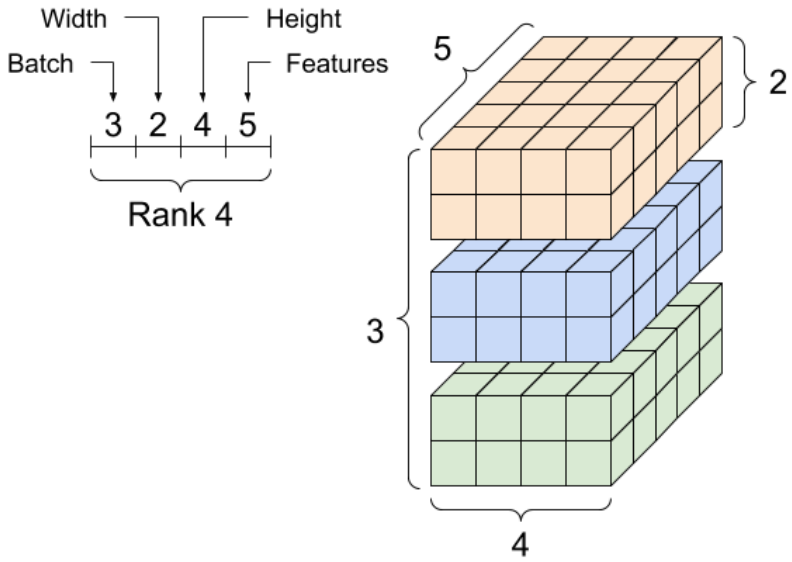


Figura 2.1: Reprezentarea unui tensor de rang 4 având forma [3, 2, 4, 5] [37]

Deoarece pachetul inițial oferit de TensorFlow ”a fost considerat oarecum complicat de înțeles și aplicat” [20], a fost dezvoltat un API mai simplu numit Keras. Așadar, Keras a ajuns în prezent un API de nivel înalt al librăriei TensorFlow folosit în definirea și antrenarea rețelelor neurale. Keras este caracterizat de ”o interfață accesibilă, extrem de productivă pentru rezolvarea problemelor de învățare automată, punând accent pe metodele de învățare adâncă moderne”. [38]

Structurile de date de bază ale API-ului Keras sunt **straturile** (*layers*) și **modelele** (secvențial sau funcțional). [38] Totodată, această librărie dispune și de câteva dintre cele mai bune rețele pre-antrenate care pot fi folosite în realizarea unor noi aplicații utilizând tehnica transferului de cunoștințe. Keras oferă o serie de tipuri de straturi predefinite ce pot fi folosite la construirea rețelei neurale:

- **Conv2D:** Stratul în care se realizează procesul de convoluție.
- **Activasion:** Aplică funcția de activare ieșirii.
- **BatchNormalization:** Normalizează intrarea acestui strat.
- **MaxPooling2D:** Stratul în care se realizează operația de max-pooling.
- **Dropout:** Strat folosit pentru prevenirea conceptului de overfitting.
- **Flatten:** Strat folosit pentru comprimarea intrării.
- **Dense:** Strat dens-conectat al rețelei neurale.

2.3 OpenCV

OpenCV este o librărie open-source dedicată domeniului de viziune computerizată și învățare automată. Lansarea acesteia a avut loc în anul 2000 de către o echipă de cercetători a companiei Intel. OpenCV a fost construit pentru a oferi o infrastructură comună pentru aplicațiile de viziune computerizată și pentru a accelera utilizarea percepției mașinăriilor în produsele comerciale. Această librărie conține peste 2500 de algoritmi optimizați ce pot fi folosiți pentru detectarea și recunoașterea fețelor, identificarea obiectelor, clasificarea acțiunilor umane dintr-un video, urmărirea mișcării ochilor, etc. [39]

OpenCV poate fi folosit împreună cu limbajele de programare: C++, Python, Java sau MATLAB pe orice platformă precum Windows, Linux, MacOS și Android. Instalarea acestei librării se poate face, în cazul mediului de dezvoltare Spyder, printr-o singură comandă tastată în prompt-ul Anaconda:

```
pip install opencv-python
```

În cadrul acestei lucrări, librăria OpenCV a fost folosită în principal pentru achiziționarea semnalului video de la camera web a computerului și, pe scurt, detectarea fețelor dintr-o imagine.

2.4 Baza de date

Baza de date folosită pentru antrenarea modelului de detectare a genului unei persoane conține 58670 imagini colore ce au la intrare în rețea dimensiunea 96 x 96. Aceste imagini au fost împărțite în 2 clase numite barbat (29144 imagini) și femeie (29526 imagini). Avantajul folosirii acestei baze de date este că toate imaginile conțin doar fețe umane, poziționate în diferite unghiuri. În anumite imagini apar și alte elemente care pot acoperi suprafața feței precum ochelari de soare, microfon, părul sau mâna persoanei, etc. În figura 2.2 se pot observa câteva imagini selectate aleator din baza de date inițială.

Această bază de date este una publică iar sursa de la care poate fi obținută este Kaggle[40]. Propietarul bazei de date, Ashutosh Chauhan, susține că datele au fost colectate din diferite surse ale internetului, cea mai mare sursă fiind baza de date IMDB[41]. Acesta a prelucrat și separat datele în 2 clase iar apoi a decupat și salvat fețele în directoarele respective sub formă de imagini cu extensia ”*.jpg*”. [40]

Figura 2.2: Exemple de imagini din baza de date [40]

Capitolul 3
Implementare software

Din punct de vedere software, acest proiect a constat în realizarea a 3 fișiere în care s-au implementat pe rând: construirea și antrenarea modelului (*construire_retea.py*), detectarea genului dintr-o imagine locală (*detectare_poza.py*) și detectarea genului de la camera web a dispozitivului (*detectare_camera_web.py*). Toate aceste resurse se pot vizualiza online[1] iar codul pentru fiecare fișier menționat este prezentat în anexa 1 de la finalul documentului. O organigramă generală și simplificată a întregului proiect se poate observa în figura 3.1.

În acest capitol voi prezenta în ordine cronologică pașii și etapele detaliate care au dus la realizarea sistemului de detecție automată a genului persoanelor.

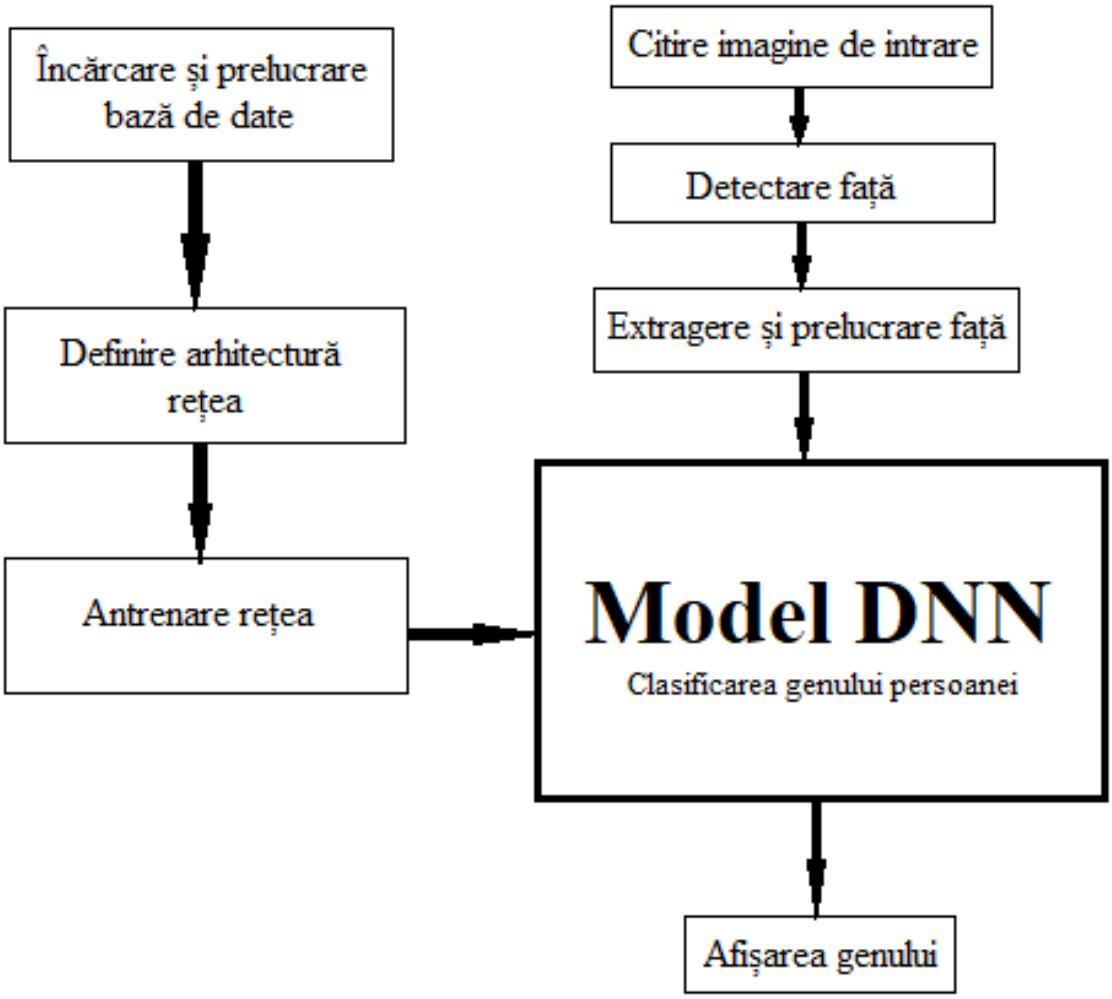


Figura 3.1: Organigrama generală a proiectului

[1] https://github.com/MurgociAdrian/Detectie-genului-persoanelor-folosind-retele-neurale-adanci

3.1 Datele inițiale

Pentru început, în vederea realizării acestui proiect a fost necesară importarea librăriilor: NumPy, os, glob, OpenCV, random, matplotlib, Tensorflow, Keras și sklearn. Dintre acestea, pentru dezvoltarea rețelei au fost folosite funcții preluate doar din librăriile Tensorflow și Keras. După importarea acestor librării, s-au definit următoarele variabile: numărul de epoci (50), dimensiunea lotului de antrenare (64), rata de învățare (0.001), dimensiunea imaginilor și numărul planurilor de culori ale acestora sub formă de tuplu ((96,96,3)), două liste goale în care se vor stoca informațiile relevante din baza de date.

```
# parametrii initiali
epochs = 50
batch_size = 64
l_rate = 0.001
dim_img = (96,96,3)

# cele 2 liste unde o sa stochez imaginile/etichetele din baza de date
data_img = []
etichete = []
```

În continuare voi oferi câteva explicații despre aceste variabile inițiale. Deoarece baza de date folosită la antrenarea rețelei este foarte mare aceasta trebuie secționată în mai multe loturi. Atunci când întreaga bază de date este transmisă prin rețea atât în etapa de *forward* cât și în cea de *backward* o singură dată se poate spune că s-a realizat o epocă. Dimensiunea lotului de antrenare (*batch_size*) reprezintă numărul de imagini de antrenare într-o etapă de *forward/backward*.

După cum se va vedea și în subcapitolul următor, din baza inițială de date au fost alese 80% din imagini pentru procesul de antrenare, asta însemnând mai exact 46936 imagini de antrenare. Așadar, având 46936 imagini de antrenat și valoarea dimensiunii lotului de antrenare setată la 64, va fi nevoie de aproximatiov 733 iterații pentru a se finaliza o epocă, lucru observat și în figura 3.7.

3.2 Încărcarea și prelucrarea datelor

Primul pas realizat în această etapă a fost încărcarea tuturor imaginilor din baza de date în mediul de dezvoltare. Mai exact, am stocat toate *path-urile* imaginilor într-o listă numită *fisiere_img* cu ajutorul funcției *glob* preluată din librăria cu același nume. Acest lucru a fost relativ ușor deoarece imaginile din folderul bazei de date (*"bazadetest"*) sunt împărțite corect în doar două subfoldere numite *"barbat"* și *"femeie"*. A fost nevoie apoi de amestecarea valorilor stocate în această listă. Deoarece funcția de mai devreme a preluat alfabetic mai întâi calea către imaginile din folderul *"barbat"* iar apoi a trecut la folderul *"femeie"*, dacă am fi lăsat la intrarea rețelei ordinea aceasta a imaginilor, ea ar fi învățat mai întâi să clasifice bărbați ajustând ponderile doar pentru acest caz specific, iar apoi ar fi trecut brusc la învățarea celuilalt gen. Acest lucru ar fi influențat în mod negativ acuratețea finală a modelului, de aceea a trebuit realizată această amestecare, pentru a avea o oarecare diversitate la intrarea rețelei.

```
# incarcare imagini din baza de date locala
fisiere_img = [f for f in glob.glob(r'C:\p_licenta\bazadetest' + "/**/*", recursive=True) if not os.path.isdir(f)]
random.shuffle(fisiere_img)
```

A urmat apoi partea de prelucrare și extragere a informației de interes pentru fiecare imagine din baza de date. Această prelucrare a însemnat mai exact citirea imaginii de la calea respectivă, redimensionarea acesteia la valorile definite la început în variabila *dim_img* și convertirea acesteia într-o matrice (ce va conține valorile pixelilor din cele 3 planuri de culori RGB). Redimensionarea fiecărei imagini de forma 96 x 96 a fost necesară deoarece la intrarea rețelei toate imaginile trebuie sa fie uniforme, în caz contrar nu se va obține în final un model bun. Informația de interes este reprezentată de valorile pixelilor imaginii precum și numele folderului în care aceasta se regăsește

(practic eticheta sau clasa din care face parte imaginea). Aceste două informații au fost stocate cu ajutorul funcției *append* în cele două liste numite *data_img* și *etichete* inițializate la început.

În continuare, a urmat convertirea celor două liste de mai sus în matrici. Pentru diminuarea resurselor folosite în timpul rulării programului, am împărțit toate valorile pixelilor la 255. Astfel, matricea *data_img* de dimensiuni (58670, 96, 96, 3) creată acum va conține doar valori cuprinse între 0 și 1, în locul intervalului de valori foarte mare cuprins între 0 și 255.

```
# convertire imagini in matrici si redimensionare + etichetare categorii si apoi salvare in liste
for c_img in fisiere_img:

    img = cv2.imread(c_img)
    img = cv2.resize(img, (dim_img[0],dim_img[1])) # redimensionare 96x96
    img = img_to_array(img)
    data_img.append(img)

    aux_etich = c_img.split(os.path.sep)[-2]
    if aux_etich == "femeie":
        aux_etich = 1
    else:
        aux_etich = 0

    etichete.append([aux_etich]) # [[1], [0], [0], ...]

# preprocesare:  convertesc listele declarate mai sus in matrici)
data_img = np.array(data_img, dtype="float") / 255.0
etichete = np.array(etichete)
```

În final, am împărțit baza de date cu ajutorul funcției *train_test_split* astfel: 80% reprezintă imaginile ce vor fi folosite în procesul de antrenare iar restul de 20% sunt imaginile de test (validare). Apoi, am convertit matricile de etichete rezultate în urma împărțirii bazei de date în matrici binare, folosind următoarele notații: valoarea 0, care făcea referire la clasa *"barbat"*, este echivalentă acum cu elementul [1, 0] din matricea binară, iar valoarea 1 (*"femeie"*) este echivalentă cu elementul [0, 1]. Ultimul pas înaintea construirii modelului a fost folosirea metodei de augmentare a datelor, proces prin care se poate îmbogăți artificial baza de date folosită la antrenare astfel încât rețeaua să nu învețe de două ori exact aceeași imagine. Acest lucru s-a realizat prin folosirea clasei *ImageDataGenerator* preluată din librăria Keras ce primește ca argument doar o valoare de rotație (în grade) pe care o aplică aleator imaginilor și returnează un obiect ce va fi folosit mai târziu la antrenarea rețelei.

```
# divizare baza de date astfel:  20% - img de validare (test) si 80% - img de antrenare (train)
(train_X, test_X, train_Y, test_Y) = train_test_split(data_img, etichete, test_size=0.2, random_state=42)

# convertirea matricilor de etichete in matrici binare folosind conventia:
train_Y = to_categorical(train_Y, num_classes=2) # 0 - barbat va fi [1, 0],  1 - femeie va fi [0, 1]
test_Y = to_categorical(test_Y, num_classes=2)  # [[1, 0], [0, 1], [0, 1], ...]

# augmentare
aug = ImageDataGenerator(rotation_range=25)
```

3.3 Arhitectura rețelei

Arhitectura rețelei a fost creată în interiorul unei funcții normale definite cu numele *build* ce primește ca parametrii dimensiunea imaginilor (lățime, înălțime), numărul planurilor de culori al acestora și numărul de clase de la ieșirea rețelei. Modelul definit este unul de tip secvențial și este alcătuit la modul general din 7 blocuri ce conțin unul sau mai multe straturi de un anumit tip.

```
# definire model
def build(latime, inaltime, nr_canale_culori, clase):   # 96  96  3  2
    model = Sequential()
    inputShape = (latime, inaltime, nr_canale_culori)
```

Primul bloc începe cu un strat de convoluție ce primește la intrare imaginile bazei de date de dimensiuni specifice precizate în variabila *inputShape*. Acest strat conține un număr de 32 de filtre de dimensiuni 3 x 3. Pe lângă acești parametrii, stratul convoluțional mai are setat argumentul *padding* la valoarea *"same"* (același). Acest lucru înseamnă că se va produce o extindere (umplere) a imaginilor de intrare prin adăugarea unor linii/coloane de valoare 0 la marginea fiecărui plan de culoare al imaginii, pentru ca dimensiunea ieșirii din acest strat rezultată în urma procesului de convoluție să fie similară cu cea a intrării. Urmează apoi aplicarea funcției de activare *ReLU* asupra ieșirii primului strat de convoluție și totodată se realizează procesul de normalizare a valorilor (lotului) pentru a aduce un plus de stabilitate și accelerare procesului de învățare. Următorul strat major al acestui bloc este cel în care se realizează operația de *max-pooling* folosind un filtru 3 x 3. Stratul final este cel numit *Dropout*, rolul acestuia fiind în acest caz de a elimina aleator din structura rețelei 25% din numărul de neuroni pentru a preveni conceptul numit overfitting.

```
# Bloc 1
model.add(Conv2D(32, (3,3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=dim_canal))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Dropout(0.25))
```

Al doilea bloc al modelului conține un strat convoluțional, urmat de aplicarea funcției de activare și apoi normalizarea datelor. Singura diferență față de blocul anterior este faptul că acum se folosesc 64 filtre în procesul de convoluție.

```
# Bloc 2
model.add(Conv2D(64, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=dim_canal))
```

Următoarele 3 blocuri sunt similare primelor două blocuri. Mai exact, blocurile 3 și 5 au aceeași structură cu blocul 1 iar blocul 4 este similar cu blocul 2. Diferențele dintre acestea apar doar la numărul de filtre folosite în straturile convoluționale precum și dimensiunea filtrelor utilizate în procesele de convoluție și max-pooling.

```
# Bloc 3
model.add(Conv2D(64, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=dim_canal))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Bloc 4
model.add(Conv2D(128, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=dim_canal))

# Bloc 5
model.add(Conv2D(128, (3,3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=dim_canal))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

Blocul 6 este alcătuit din stratul numit *Flatten* (a cărui rol este de a aplatiza valorile primite de la ultimul strat al blocului anterior) și de un strat dens ce conține 1024 unități (neuroni). Funcția de activare folosită pentru acest strat dens este *ReLU*. Totodată, se realizează din nou o normalizare a lotului iar în final există stratul *Dropout* care de data aceasta primește ca parametru o rată de 50%.

```
# Bloc 6
model.add(Flatten())

model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

Ultimul bloc este reprezentat de stratul de ieșire al rețelei, ce conține în aces caz doar 2 neuroni. Funcția de activare aleasă pentru acest ultim strat este funcția logistică (*sigmoid*), ea fiind cea mai portivită pentru o clasificare de tip binar.

```
# Bloc 7
model.add(Dense(clase))
model.add(Activation("sigmoid"))

return model
```

O reprezentare scurtă precum și câteva informații despre arhitectura modelului returnat de această funcție se poate observa în figura 3.2.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 96, 96, 32)        896
_________________________________________________________________
activation (Activation)      (None, 96, 96, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 96, 96, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              8389632
_________________________________________________________________
activation_5 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 2050
_________________________________________________________________
activation_6 (Activation)    (None, 2)                 0
=================================================================
Total params: 8,675,202
Trainable params: 8,672,322
Non-trainable params: 2,880
```

Figura 3.2: Arhitectura modelului

3.4 Antrenarea rețelei

Înainte de procesul de antrenare a rețelei a fost nevoie de construirea propriu-zisă a modelului. Acest lucru s-a realizat prin apelarea funcției *build* de mai devreme folosind parametrii specifici bazei noastre de date și stocarea rezultatului returnat în variabila *model*. Funcția de optimizare aleasă pentru acest model este Adam. Această funcție contribuie la modul în care ponderile rețelei se vor modifica în timpul procesului numit *back-propagation*. După cum am mai precizat și în alt subcapitol, Adam este cea mai populară și performantă funcție ce poate fi utilizată într-o astfel de arhitectură DNN, oferind multe beneficii din punct de vedere al eficienței computaționale și a memoriei folosite de sistem.

Un ultim pas realizat înainte de antrenare a fost compilarea modelului folosind funcția de optimizare precizată mai sus și o funcție de cost (pierderi) numită *binary_crossentropy*. După cum sugerează și numele, această funcție de cost aleasă este cea mai potrivită pentru o clasificare de tip binar. Antrenarea rețelei a fost realizată cu ajutorul funcției *fit*, iar în final modelul antrenat a fost salvat pentru a putea fi folosit mai departe în aplicația de detecție a genului persoanelor.

```
# construire model
model = build(latime=dim_img[0], inaltime=dim_img[1], nr_canale_culori=dim_img[2], clase=2)

# compilare model
opt = Adam(learning_rate=l_rate, decay=l_rate/epochs) # functia de optimizare
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# antrenare model
H = model.fit(aug.flow(train_X, train_Y, batch_size=batch_size),
                        validation_data=(test_X,test_Y),
                        steps_per_epoch=len(train_X) // batch_size,
                        epochs=epochs, verbose=1)

# salvare model
model.save('detectarea_genului_pers3.h5')
```

3.5 Clasificarea genului (fotografie)

În continuare, folosind modelul salvat anterior și în mare parte funcții ale librăriei OpenCV, a fost implementată aplicația de detectare a genului persoanelor după o imagine cu fața acestora. Mai întâi, am realizat această detecție folosind ca intrare o imagine stocată local pe dispozitiv iar apoi intrarea a reprezentat semnalul video captat de la camera web a dispozitivului. Clasificarea genului a constat în realizarea a 3 etape principale: extragerea fețelor din imaginea dată la intrare, prelucrarea acestei imagini (fețe) pentru a putea fi introdusă în modelul antrenat și afișarea grafică a rezultatului predicției modelului.

3.5.1 Detectarea feței

Detectarea fețelor dintr-o imagine a fost realizată cu ajutorul funcției *detect_face*. Această funcție primește ca parametru imaginea de la intrare sub forma unei matrici de valori întregi pozitive ale pixelilor din cele 3 planuri de culori. Este de menționat faptul ca toate funcțile corespunzătoare librăriei OpenCV folosesc convenția de notare a planurilor de culori în ordinea BGR în loc de RGB. Funcția *detect_face* returnează două liste. Prima listă, care este singura de interes în această aplicație, conține câte un vector de câte 4 valori pentru fiecare față detectată în imagine. Aceste 4 valori reprezintă coordonatele prin care se poate delimita o față detectată. Aceste coordonate, ce pot fi vizualizate și în figura 3.3, au fost explicate și printr-un comentariu aflat înaintea funcției respective în codul sursă. În cazul în care imaginea de la intrare nu conține nicio față, se vor face afișările corespunzătoare acestui caz în partea de jos a imaginii.

```
# functia pentru detectarea fetei: fata - o lista ce contine cele 4 puncte: 2 din coltul stanga jos(X,Y) respectiv 2 din dreapta sus(X,Y)
fata,_ = cv.detect_face(img_in)

# in cazul in care nu s-a gasit nicio fata se fac afisarile corespunzatoare:
if not fata:
    if aux1 == 0:
        aux2 = 0
        img_redim = cv2.resize(img_in, (dim2,dim1))
        cv2.putText(img_redim, 'Nu s-a putut detecta nicio fata!', (10, (dim1-15)), cv2.FONT_HERSHEY_DUPLEX, dim_font_1, (0, 0, 255), 2)
    else:
        cv2.putText(img_in, 'Nu s-a putut detecta nicio fata!', (10, (dim1-15)), cv2.FONT_HERSHEY_DUPLEX, dim_font_1, (0, 0, 255), 2)
```

3.5.2 Prelucrarea imaginii (feței) extrase

Folosind coordonatele oferite de funcția *detect_face* de mai devreme, am extras zona de interes, mai exact porțiunea feței din imaginea inițială, apoi am trasat cu ajutorul unei funcții din librăria OpenCV un dreptunghi verde ce cuprinde această zonă a feței. Pentru a putea fi introdusă în modelul antrenat, imaginea extrasă va trebui să aibă aceleași propietați cu imaginile folosite în procesul de antrenare al rețelei. Așadar, s-a efectuat o redimensionare de 96x96 a imaginii extrase, apoi valorile pixelilor imaginii au fost împărțite la 255 iar în final imaginea a fost convertită într-o matrice.

```
for i, f in enumerate(fata):

    # cele 4 puncte
    (start_X, start_Y) = f[0], f[1]
    (final_X, final_Y) = f[2], f[3]

    # desenare dreptunghi peste fata detectata
    cv2.rectangle(img_in, (start_X,start_Y), (final_X,final_Y), (0, 204, 102), 2)

    # decupare fata detectata
    fata_decupata = np.copy(img_in[start_Y:final_Y,start_X:final_X])

    # preprocesare: (la fel ca la construirea modelului)
    fata_decupata = cv2.resize(fata_decupata, (96,96))
    fata_decupata = fata_decupata.astype("float") / 255.0
    fata_decupata = img_to_array(fata_decupata)
    fata_decupata = np.expand_dims(fata_decupata, axis=0)
```

3.5.3 Detectarea și afișarea genului

Ultima etapă a acestei aplicații de clasificare a constat în aplicarea modelului antrenat asupra imaginii ce conține fața, lucru realizat cu ajutorul metodei *predict*. Așadar, valoarea returnată stocată în variabila *val_incredere* va fi reprezentată sub forma unei matrici cu o coloană și două linii. Pe prima linie se va afla un număr cuprins între 0 și 1 care înmulțit cu 100 va reprezenta procentul prezis de model în favoarea clasei *"barbat"*, similar și pe a doua linie, de data aceasta fiind vorba de clasa *"femeie"*. Ceea ce se va afișa în final va fi clasa a cărui procent prezis de model este cel mai mare. Pe lângă numele clasei, se va afișa și această valoare prezisă de model sub formă de procent cu o singură zecimală. Toate afișările au fost făcute cu ajutorul funcției *putText*.

```
# aplicarea modelului pentru fata curenta
val_incredere = model.predict(fata_decupata)[0]

# se determina clasa corecta in functie de valoarea maxima de mai sus
i = np.argmax(val_incredere)
eticheta = clase[i]

eticheta = "{} - {:.1f}%".format(eticheta, val_incredere[i] * 100)

Y = start_Y - 10 if start_Y - 10 > 10 else start_Y + 10

# functia de scriere a clasei si a valorii in procent de incredere a fetei detectate
cv2.putText(img_in, eticheta, (start_X, Y), cv2.FONT_HERSHEY_SIMPLEX, dim_font_2, (0, 204, 102), val_grosime_font)
```

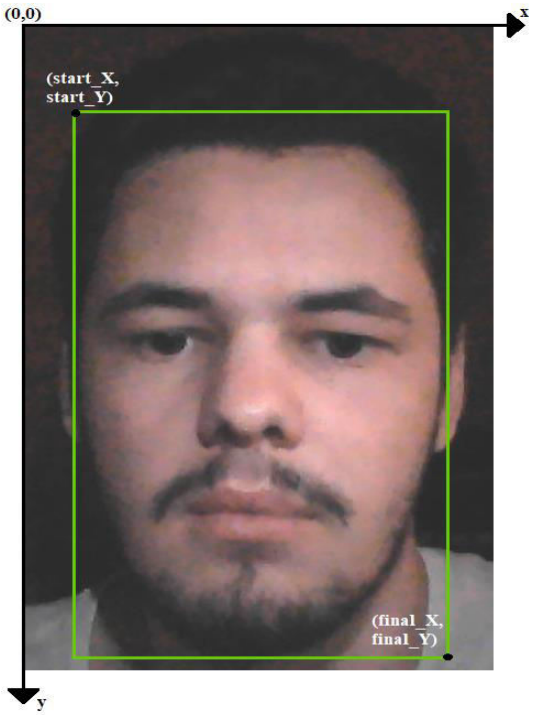


Figura 3.3: Explicație puncte detectare față

3.6 Clasificarea genului (camera web)

Aplicația de clasificare a genului persoanelor folosind camera web este în mare parte similară cu cea explicată anterior. Singura diferență majoră apare doar la partea de început în care se obține imaginea ce va fi transimsă funcției de detectare a feței. Pentru a putea prelua semnalul video este nevoie mai întâi de inițializarea camerei web. Apoi, se extrage pe rând fiecare cadru din acest semnal cu ajutorul funcției *read*. Aceste cadre reprezintă de fapt imaginile ce vor fi transmise ca parametru funcției *detect_face*. Execuția aplicației se oprește la apăsarea tastei ”x”.

```
# initializare camera
webcam = cv2.VideoCapture(0)

clase = ['Barbat','Femeie']

while webcam.isOpened():

    # citire frame (img/s) de la camera
    _, frame = webcam.read()

    fata,_ = cv.detect_face(frame)
```

3.7 Rezultate și performanțe

Performanța obținută cu ajutorul acestei rețele neurale adânci realizată de la zero și antrenată folosind o bază de date de dimensiune medie este una foarte bună. Rezultatele privind evoluția acurateții modelului și a funcției de cost pot fi observate în figurile 3.4, respectiv 3.5.

Acuratețea finală obținută în timpul procesului de antrenare este de 98,51%, această valoare fiind atinsă în 50 de epoci. Se observă ca încă de la început, modelul atinge o acuratețe de aproximativ 96% după primele 6 epoci, urmând ca apoi această valoare să varieze încet până la rezultatul final. Acuratețea în cazul procesului de validare arată foarte bine, în sensul că nu există semne de overfitting sau underfitting prezente pentru acest model. Așadar, acuratețea în cazul validării după 50 de epoci este de 97,65%.

În cazul funcției de cost, se observă o scădere exponențială în prima parte a procesului de antrenare, urmând apoi o porțiune mai lentă în care valoarea acestei funcții descrește. Valoarea finală a funcției de cost este 0,0426 în cazul antrenării și 0,1005 în cazul validării. Aceste rezultate foarte mici sunt implicit considerate foarte bune, ținând cont că funcția de cost folosită *binary_crossentropy* returnează valori doar în intervalul [0,1].

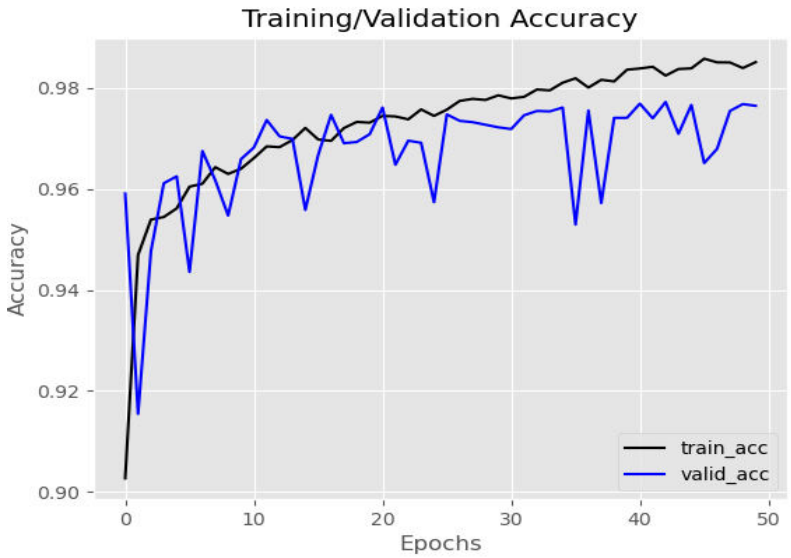


Figura 3.4: Evoluția acurateții în timpul procesului de antrenare/validare

Figura 3.5: Evoluția funcției de cost în timpul procesului de antrenare/validare

Aplicația de detectare a genului persoanelor ce folosește această rețea neurală adâncă clasifică destul de bine imagini reale, însă pot apărea limitări datorită unor factori de mediu precum luminozitatea sau zgomotul de fundal, mai ales în cazul folosirii ca intrare a camerei web. Claritatea și rezoluția fotografiei introdusă în model sunt alți factori ce pot influența negativ decizia de clasificare finală. În cazul unei luminozități extrem de mici sau foarte mari a imaginii, apar probleme în cazul detectării fețelor din imaginea respectivă; așadar, nu se poate furniza nimic modelului în scopul clasificării. Un ultim aspect de menționat este faptul că modelul nu este capabil să clasifice foarte corect persoane de vârste foarte mici sau nou-născuți, datorită insuficienței de imagini cu persoane din această categorie a bazei de date.

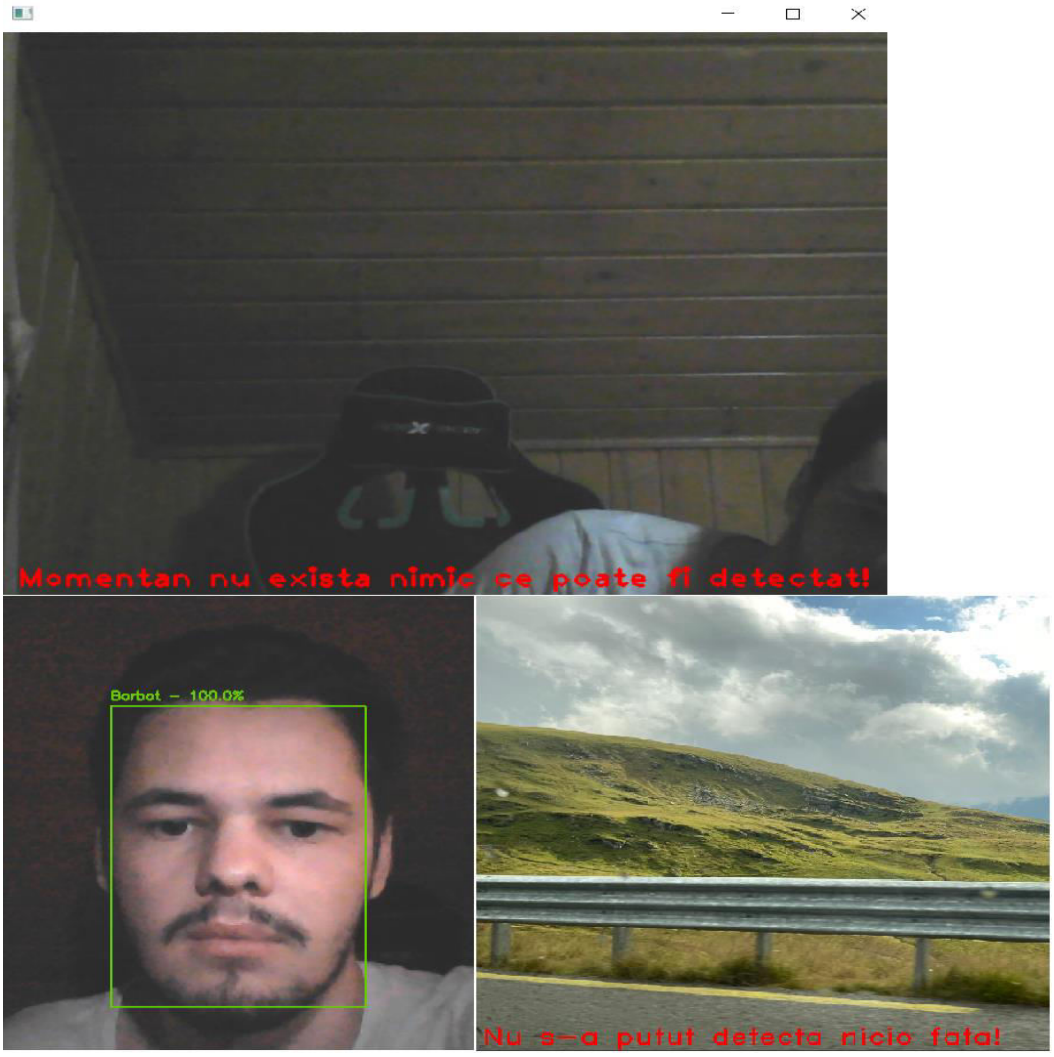


Figura 3.6: Exemple de afișare

Procesul de antrenare al acestei rețele folosind baza de date menționată a durat aproximativ 433 de minute. Înaintea acestui proces, am extras din baza de date doar 10240 de imagini pe care le-am folosit să antrenez aceeași rețea prezentată până acum, singura diferență fiind folosirea mai multor parametrii la metoda de augmentare a datelor. Acuratețea obținută de acest prim model a fost de 98,57% în cazul antrenării și 97,12% în cazul validării. Cu toate acestea, după o serie de teste reale făcute pe o varietate de fotografii, am constatat că modelul antrenat pe întreaga bază de date de 58670 imagini se descurcă mai bine comparativ cu acel prim model. Deci, concluzia ar fi că un factor foarte important care duce la obținerea unui rețele performante este dimensiunea bazei de date folosită la antrenare.

Câteva exemple a modului de afișare a aplicației de detectare a genului persoanelor se pot observa în figura 3.6. Rezultatele grafice din figurile 3.4 și 3.5 au fost obținute cu ajutorul unor funcții din librăria *matplotlib*. Datele cu care s-au realizat aceste reprezentări grafice au fost preluate din timpul procesului de antrenare al modelului. În figura 3.7 se poate observa o captură de ecran ce conține câteva informații din timpul antrenării modelului.

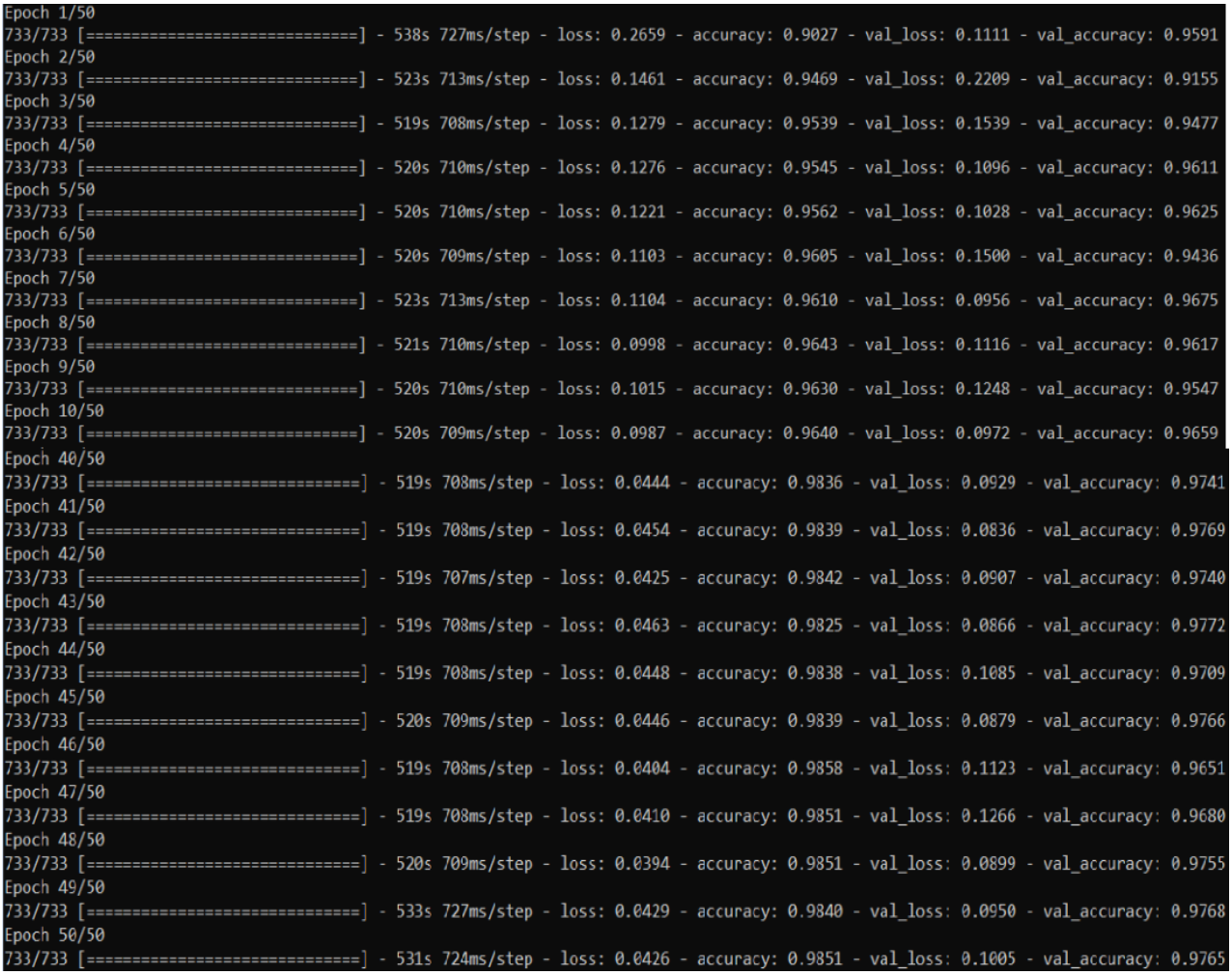


Figura 3.7: Captură de ecran din timpul procesului de antrenare

Concluzii

Concluzii generale

Realizarea acestui proiect de diplomă a reprezentat o oportunitate prin care am putut aprofunda tehnici și metode actuale de învățare automată în acest domeniu bogat al inteligenței artificiale, precum și noțiuni importante ale limbajului Python. Performanța obținută de modelul de detecție a genului unei persoane bazat pe rețele neurale adânci a fost una foarte bună, peste așteptările personale.

Am preferat să implementez acest model de la zero și să nu folosesc metoda populară a transferului de cunoștințe tocmai pentru a înțelege și mai bine ceea ce se întâmplă în spatele acestor algoritmi de învățare automată. Modelul a fost antrenat pe o bază de date medie cu imagini colore de tip RGB, etichetată cu genul persoanei. Un factor important ce a contribuit la acuratețea finală a modelului a fost faptul că imaginile bazei de date conțin o varietate de fețe umane decupate din diferite surse de către autorul acesteia. Modelul antrenat a fost folosit apoi în două aplicații capabile să clasifice cu succes genul unei persoane după o fotografie cu fața acesteia dintr-o imagine locală, respectiv de la camera web a dispozitivului.

În concluzie, rețelele neurale adânci reprezintă cea mai bună alegere în cadrul unei aplicații moderne ce își propune rezolvarea problemelor de detecție și clasificare a imaginilor, datorită performanțelor remarcabile obținute în aceste sisteme reale. Arhitecturile de tip CNN bazate pe conceptul de învățare adâncă sunt considerate cu siguranță viitorul multor domenii ale inteligenței artificiale.

Dezvoltări ulterioare

O posibilă dezvoltare ulterioară a acestui proiect din punct de vedere educațional ar fi folosirea arhitecturii rețelei în mai multe procese de antrenare pe aceeași bază de date, schimbând de fiecare dată un parametru cheie al rețelei. Prin acest parametru, mă refer la funcția de optimizare, funcția de cost, funcția de activare din straturile interne sau funcția de activare de la ieșire. Așadar, comparând rezultatele obținute după fiecare antrenare, se va putea concluziona ce configurație a rețelei va furniza cea mai bună performanță și implicit care este cel mai bun parametru ce contribuie la obținerea acestei performanțe.

O altă dezvoltare ulterioară a aplicației de detecție a genului persoanelor ar putea fi în domeniul de marketing, la personalizarea produselor și serviciilor oferite de o companie în funcție de genul persoanei. Această aplicație ar putea fi integrată pe pagina web principală a unei compani ce se ocupă cu vânzare de haine. Astfel, în urma detecției genului clientului, se vor afișa doar produse specifice categoriei respective.

Bibliografie

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning* , MIT Press, 2016. *Online: http://www.deeplearningbook.org*

[2] Greco, Antonio & Saggese, Alessia & Vento, Mario & Vigilante, Vincenzo, *A Convolutional Neural Network for Gender Recognition Optimizing the Accuracy/Speed Tradeoff* , IEEE Access, PP. 1-1. 10.1109/ACCESS.2020.3008793.
Online: https://www.researchgate.net/publication/342906361_A_Convolutional_Neural_Network_for_Gender_Recognition_Optimizing_the_AccuracySpeed_Tradeoff

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *"Mobilenets: Efficient convolutional neural networks for mobile vision applications"*, arXiv, 2017.

[4] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, *"Labeled facesin the wild: A database forstudying face recognition in unconstrainedenvironments"*, in Workshop on faces in'Real-Life'Images: detection,alignment, and recognition, 2008.

[5] O. M. Parkhi, A. Vedaldi, A. Zisserman et al., *"Deep face recognition"* in BMVC, vol. 1, no. 3, 2015, p. 6.

[6] V. Carletti, A. Greco, A. Saggese, and M. Vento, *"An effective realtime gender recognition system for smart cameras"*, Journal of Ambient Intelligence and Humanized Computing, 2019.

[7] R. Rothe, R. Timofte, L. V. Gool, *"Deep expectation of real and apparent age from a single image without facial landmarks"*, International Journal of Computer Vision (IJCV), July 2016.

[8] E. Eidinger, R. Enbar, and T. Hassner, *"Age and gender estimation of unfiltered faces"*, IEEE Trans. on Information Forensics and Security, vol. 9, no. 12, pp. 2170–2179, 2014.

[9] F. Chollet, *"Xception: Deep learning with depthwise separable convolutions"*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[10] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, *"Shufflenet v2: Practical guidelines for efficient cnn architecture design"*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, *"Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size"*, arXiv preprint arXiv:1602.07360, 2016.

[12] Zaman, Fadhlan. (2020). *Gender classification using custom convolutional neural networks architecture*. International Journal of Electrical and Computer Engineering (IJECE). 10. 5758. 10.11591/ijece.v10i6.pp5758-5771.

[13] Macukow B. (2016) *Neural Networks – State of Art, Brief History, Basic Models and Architecture*. In: Saeed K., Homenda W. (eds) Computer Information Systems and Industrial Management. CISIM 2016. Lecture Notes in Computer Science, vol 9842. Springer, Cham.
Online: https://doi.org/10.1007/978-3-319-45378-1_1

[14] Ilya Kuzovkin, *"Deep Learning - THEORY, HISTORY, STATE OF THE ART & PRACTICAL TOOLS"*, Machine Learning Estonia, 2016
Online: http://www.ikuz.eu/materials/slides/Ilya-Kuzovkin-Deep-Learning-theory-history-state-of-the-art-practical-tools.pdf

[15] Saurabh Mhatre, *"What Is The Relation Between Artificial And Biological Neuron?"*, 2020
Online: https://smhatre59.medium.com/what-is-the-relation-between-artificial-and-biological-neuron-18b05831036

[16] UPB. Laboratorul de rețele neuronale și sisteme fuzzy

[17] Utkarsh Sinha, *"First artificial neurons: The McCulloch-Pitts model"*, AI Shack
Online: https://aishack.in/tutorials/artificial-neurons-mccullochpitts-model

[18] S. Haykin, *"Neural networks and learning machines"*, 3rd ed. Pearson education Upper Saddle River, 2009.

[19] Wilson Castro, Jimy Oblitas, Roberto Santa-Cruz, Himer Avila-George, *"Multilayer perceptron architecture optimization using parallel computing techniques"* , 2017
Online: https://doi.org/10.1371/journal.pone.0189369

[20] Christoph Rasche, *"Computer Vision"* , 2019.

[21] Abdelrahman Elogeel, *"Multilayer Perceptron"*, 2010
Online: https://elogeel.wordpress.com/2010/05/10/multilayer-perceptron-2/

[22] Victor Neagoe, *"Rețele neurale pentru explorarea datelor"* , Matrix Rom, 2018

[23] *"Activation function"*, *Online: https://en.wikipedia.org/wiki/Activation_function*

[24] Jason Brownlee, *"How to Choose an Activation Function for Deep Learning"* , 2021
Online: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[25] Jason Brownlee, *"Loss and Loss Functions for Training Deep Learning Neural Networks"* , 2019
Online: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[26] B. Graham. *Lecture 3: Delta rule*. 2014
Online: http://www.cs.stir.ac.uk/courses/CSC9YF/lectures/ANN/3-DeltaRule.pdf

[27] IBM Cloud Education, *Gradient Descent* , 2020
Online: https://www.ibm.com/cloud/learn/gradient-descent

[28] Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization* , 2015
Online: https://arxiv.org/abs/1412.6980

[29] Anastasia Kyrykovych, *Deep Neural Networks* , 2020
Online: https://www.kdnuggets.com/2020/02/deep-neural-networks.html

[30] *Sursă: https://www.oreilly.com/library/view/machine-learning with/9781789346565/6899a0ec-2cf9-4525-b3bf-56dfafa827e7.xhtml*

[31] *Sursă: https://www.youtube.com/watch?v=LvqzKr-dORQ*

[32] Angel Das, *Convolution Neural Network for Image Processing — Using Keras* , 2020
Online: https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

[33] Jeremy Jordan, *Common architectures in convolutional neural networks.* , 2018
Online: https://www.jeremyjordan.me/convnet-architectures/

[34] Raimi Karim, *Illustrated: 10 CNN Architectures* , 2019
Online: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#81e0

[35] *Sursă: https://www.w3resource.com/python/python-tutorial.php*

[36] *Sursă: https://www.tensorflow.org/*

[37] *Sursă: https://www.tensorflow.org/guide/tensor*

[38] *Sursă: https://keras.io/about/*

[39] *Sursă: https://opencv.org/about/*

[40] *Sursă: https://www.kaggle.com/cashutosh/gender-classification-dataset*

[41] Rasmus Rothe, Radu Timofte, Luc Van Gool, *DEX: Deep EXpectation of apparent age from a single image* , IEEE International Conference on Computer Vision Workshops (ICCVW), 2015

Anexa 1

Codul sursă pentru construirea și antrenarea rețelei

```
import numpy as np
import os
import glob
import cv2
import random
import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation,
BatchNormalization, Flatten, Dropout, Dense
from tensorflow.keras import backend as K
from sklearn.model_selection import train_test_split


# parametrii initiali
epochs = 50
batch_size = 64
l_rate = 0.001
dim_img = (96,96,3)

# cele 2 liste unde o sa stochez imaginile/etichetele din baza de date
data_img = []
etichete = []

# incarcare imagini din baza de date locala
fisiere_img = [f for f in glob.glob(r'C:\p_licenta\bazadetest' + "/**/*",
recursive=True) if not os.path.isdir(f)]
random.shuffle(fisiere_img)

# convertire imagini in matrici si redimensionare + etichetare categorii si apoi
salvare in liste
for c_img in fisiere_img:

    img = cv2.imread(c_img)
    img = cv2.resize(img, (dim_img[0],dim_img[1])) # redimensionare 96x96
    img = img_to_array(img)
    data_img.append(img)

    aux_etich = c_img.split(os.path.sep)[-2]
    if aux_etich == "femeie":
        aux_etich = 1
    else:
        aux_etich = 0

    etichete.append([aux_etich]) # [[1], [0], [0], ...]
```

```
# preprocesare:  convertesc listele declarate mai sus in matrici)
data_img = np.array(data_img, dtype="float") / 255.0
etichete = np.array(etichete)

# divizare baza de date astfel:  20% - img de validare (test) si 80% - img de
antrenare (train)
(train_X, test_X, train_Y, test_Y) = train_test_split(data_img, etichete,
test_size=0.2, random_state=42)

# convertirea matricilor de etichete in matrici binare folosind conventia:
train_Y = to_categorical(train_Y, num_classes=2) # 0 - barbat va fi [1, 0],  1 -
femeie va fi [0, 1]
test_Y = to_categorical(test_Y, num_classes=2)  # [[1, 0], [0, 1], [0, 1], ...]

# augmentare
aug = ImageDataGenerator(rotation_range=25)

# definire model
def build(latime, inaltime, nr_canale_culori, clase):   # 96  96  3  2
    model = Sequential()
    inputShape = (latime, inaltime, nr_canale_culori)
    dim_canal = -1

    if K.image_data_format() == "channels_first": # returneaza un string, fie
'channels_first' sau 'channels_last'
        inputShape = (nr_canale_culori, inaltime, latime)
        dim_canal = 1 # utilizat la normalizare, iar channel se refera la
nr_canale_culori

    # Bloc 1
    model.add(Conv2D(32, (3,3), padding="same", input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=dim_canal))
    model.add(MaxPooling2D(pool_size=(3,3)))
    model.add(Dropout(0.25))

    # Bloc 2
    model.add(Conv2D(64, (3,3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=dim_canal))

    # Bloc 3
    model.add(Conv2D(64, (3,3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=dim_canal))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))

    # Bloc 4
    model.add(Conv2D(128, (3,3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=dim_canal))

    # Bloc 5
    model.add(Conv2D(128, (3,3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=dim_canal))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))

    # Bloc 6
    model.add(Flatten())
```

```
    model.add(Dense(1024))
    model.add(Activation("relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))

    # Bloc 7
    model.add(Dense(clase))
    model.add(Activation("sigmoid"))

    return model


# construire model
model = build(latime=dim_img[0], inaltime=dim_img[1],
nr_canale_culori=dim_img[2], clase=2)

# print(model.summary())

# compilare model
opt = Adam(learning_rate=l_rate, decay=l_rate/epochs) # functia de optimizare
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# antrenare model
H = model.fit(aug.flow(train_X, train_Y, batch_size=batch_size),
                        validation_data=(test_X,test_Y),
                        steps_per_epoch=len(train_X) // batch_size,
                        epochs=epochs, verbose=1)

# salvare model
model.save('detectarea_genului_pers3.h5')


# grafice pentru training si validation accuracy+loss in functie de epochs
plt.style.use("ggplot")

plt.figure()
plt.plot(np.arange(0,epochs), H.history['accuracy'], label="train_acc",
color='k')
plt.plot(np.arange(0,epochs), H.history['val_accuracy'], label="valid_acc",
color='b')
plt.title("Training/Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.savefig('grafic31.png')

plt.figure()
plt.plot(np.arange(0,epochs), H.history['loss'], label="train_loss", color='y')
plt.plot(np.arange(0,epochs), H.history['val_loss'], label="valid_loss",
color='r')
plt.title("Training/Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.savefig('grafic32.png')
```

Codul sursă pentru detectarea genului unei/unor persoane dintr-o fotografie

```
import numpy as np
import cv2
```

```
import cvlib as cv

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array


model = load_model('detectarea_genului_pers3.h5')

clase = ['Barbat','Femeie']

# citire imagine
img_in = cv2.imread(r'D:/FACULTATE/LICENTA/855937.jpg') # dim (y,x,3)

dim1 = img_in.shape[0]  # y
dim2 = img_in.shape[1]  # x
aux1 = 1
aux2 = 1
aux_font = 1
dim_font_1 = 1.7
dim_font_2 = 0.6
val_grosime_font = 2

# in cazul in care imaginea de la intrare are dimensiuni mai mari decat
1920x1010 se va face redimensionare pentru afisare:
if (dim1 > 1010):
    aux1 = 0
    dim1 = 1010
if (dim2 > 1920):
    aux1 = 0
    aux_font = 0
    dim2 = 1920

# conditie pentru imagini cu rezolutii foarte mici pentru afisarea textului in
limitele acesteia
if (dim2 < 900):
    dim_font_1 = 0.7

# conditie pentru imagini cu rezolutii foarte mari: se cresc valorile care tin
de dim font pentru ca textul sa fie vizibil
if (aux_font == 0):
    dim_font_2 = 1
    val_grosime_font = 3

# functia pentru detectarea fetei: fata - o lista ce contine cele 4 puncte: 2
din coltul stanga jos(X,Y) respectiv 2 din dreapta sus(X,Y)
fata,_ = cv.detect_face(img_in)

# in cazul in care nu s-a gasit nicio fata se fac afisarile corespunzatoare:
if not fata:
    if aux1 == 0:
        aux2 = 0
        img_redim = cv2.resize(img_in, (dim2,dim1))
        cv2.putText(img_redim, 'Nu s-a putut detecta nicio fata!', (10, (dim1-
15)), cv2.FONT_HERSHEY_DUPLEX, dim_font_1, (0, 0, 255), 2) # (B,G,R)
    else:
        cv2.putText(img_in, 'Nu s-a putut detecta nicio fata!', (10, (dim1-15)),
cv2.FONT_HERSHEY_DUPLEX, dim_font_1, (0, 0, 255), 2)

for i, f in enumerate(fata):

    # cele 4 puncte
    (start_X, start_Y) = f[0], f[1]
    (final_X, final_Y) = f[2], f[3]
```

```
    # desenare dreptunghi peste fata detectata
    cv2.rectangle(img_in, (start_X,start_Y), (final_X,final_Y), (0, 204, 102),
2)

    # decupare fata detectata
    fata_decupata = np.copy(img_in[start_Y:final_Y,start_X:final_X])

    if (fata_decupata.shape[0]) < 10 or (fata_decupata.shape[1]) < 10:
        continue

    # preprocesare: (la fel ca la construirea modelului)
    fata_decupata = cv2.resize(fata_decupata, (96,96))
    fata_decupata = fata_decupata.astype("float") / 255.0
    fata_decupata = img_to_array(fata_decupata)
    fata_decupata = np.expand_dims(fata_decupata, axis=0)

    # aplicarea modelului pentru fata curenta
    val_incredere = model.predict(fata_decupata)[0]

    # se determina clasa corecta in functie de valoarea maxima de mai sus
    i = np.argmax(val_incredere)
    eticheta = clase[i]

    eticheta = "{} - {:.1f}%".format(eticheta, val_incredere[i] * 100)

    Y = start_Y - 10 if start_Y - 10 > 10 else start_Y + 10

    # functia de scriere a clasei si a valorii in procent de incredere a fetei
detectate
    cv2.putText(img_in, eticheta, (start_X, Y), cv2.FONT_HERSHEY_SIMPLEX,
dim_font_2, (0, 204, 102), val_grosime_font)


# afisare
if ((aux2 == 0) & (aux1 == 0)):
    cv2.imshow(" ", img_redim)
if ((aux1 == 0) & (aux2 == 1)):
    img_redim = cv2.resize(img_in, (dim2,dim1))
    cv2.imshow(" ", img_redim)
if (aux1+aux2) == 2:
    cv2.imshow(" ", img_in)
```

Codul sursă pentru detectarea genului unei/unor persoane folosind camera web

```
import numpy as np
import cv2
import cvlib as cv

from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array


model = load_model('detectarea_genului_pers3.h5')

# initializare camera
webcam = cv2.VideoCapture(0)

clase = ['Barbat','Femeie']

while webcam.isOpened():
```

```
    # citire frame (img/s) de la camera
    _, frame = webcam.read()

    fata,_ = cv.detect_face(frame)

    if not fata:
        cv2.putText(frame, 'Momentan nu exista nimic ce poate fi detectat!',
(12, 472), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 0, 255), 2)

    for i, f in enumerate(fata):

        (start_X, start_Y) = f[0], f[1]
        (final_X, final_Y) = f[2], f[3]

        cv2.rectangle(frame, (start_X,start_Y), (final_X,final_Y), (0, 204,
102), 2)

        fata_decupata = np.copy(frame[start_Y:final_Y,start_X:final_X])

        if (fata_decupata.shape[0]) < 10 or (fata_decupata.shape[1]) < 10:
            continue

        fata_decupata = cv2.resize(fata_decupata, (96,96))
        fata_decupata = fata_decupata.astype("float") / 255.0
        fata_decupata = img_to_array(fata_decupata)
        fata_decupata = np.expand_dims(fata_decupata, axis=0)

        val_incredere = model.predict(fata_decupata)[0]

        i = np.argmax(val_incredere)
        eticheta = clase[i]

        eticheta = "{} - {:.1f}%".format(eticheta, val_incredere[i] * 100)

        Y = start_Y - 10 if start_Y - 10 > 10 else start_Y + 10

        cv2.putText(frame, eticheta, (start_X, Y), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 204, 102), 2)

    cv2.imshow(" ", frame)

    # la apasarea tastei "x" se opreste executia programului
    if cv2.waitKey(1) & 0xFF == ord('x'):
        break

# eliberare resurse
webcam.release()
cv2.destroyAllWindows()
```