

Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Detectarea emoțiilor umane folosind rețele neurale convoluționale

Lucrare de licență

**Prezentată ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Calculatoare și Tehnologia Informației*
programul de studii *Ingineria Informației***

Conducător științific
Prof. Dr. Ing. Mihai Ciuc

Absolvent
Florin-Daniel Frățilă

Anul 2020

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Detectarea emoțiilor umane folosind rețele neurale convoluționale*, 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 Inginerie Electronică și Telecomunicații/ Calculatoare și Tehnologia Informației, programul de studii *Ingineria Informației* 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 măsurărilor 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, Septembrie 2020.

Absolvent: Florin-Daniel Frățilă



Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Departamentul **EAI**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **FRĂȚILĂ N. Florin-Daniel , 444A**

1. Titlul temei: Detectarea emoțiilor umane folosind rețele neurale convoluționale

2. Descrierea contribuției originale a studentului (în afara părții de documentare) și specificații de proiectare:

Se va proiecta o rețea neurală convoluțională pentru detectarea emoțiilor umane în limbajul Python. În implementarea proiectului se vor folosi librării specifice pentru machine learning și inteligență artificială(PyTorch,TensorFlow), precum și librării pentru achiziționarea și procesarea imaginilor(OpenCV).

Rețeaua neurală va fi antrenată pe diferite baze de date specifice(NovaEmotions,Jack Fee Emotions) în care vor fi stocate atât imagini cu diferite expresii faciale și emoțiile aferente acestora, cât și diferite etichete care vor ajuta rețeaua neurală să clasifice emoția umană asociată imaginii.

La finalul procesului de antrenare și testare, după ce rețeaua neurală va avea o acuratețe corespunzătoare, se va implementa o interfață grafică ce va permite achiziționarea datelor de intrare de la o cameră video și va afișa emoția umană recunoscută de rețea.

3. Resurse folosite la dezvoltarea proiectului:

Python,PyTorch,TensorFlow,OpenCV

4. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline:

RFIA,PI,DEPI,POO

5. Proprietatea intelectuală asupra proiectului aparține: U.P.B.

6. Data înregistrării temei: 2020-03-03 17:56:41

Conducător(i) lucrare,

Prof. dr. ing. Mihai CIUC

Student,

Director departament,

Prof. dr. ing Sever PAȘCA

Decan,

Prof. dr. ing. Mihnea UDREA

Cod Validare: **1b8f472a72**

Cuprins

Lista figurilor	ix
Lista tabelelor	xi
Lista acronimelor	xii
Introducere	1
Motivație	1
Aplicabilitate	1
Obiective	3
Starea artei	3
1. Noțiuni teoretice	9
1.1. Istoric	9
1.2. Analogie biologică	10
1.3. Rețele neurale clasice	11
1.3.1. Modelul neuronal McCulloch-Pitts	11
1.3.2. Perceptronul	12
1.3.3. Perceptronul Multi-Strat (MLP)	14
1.3.4. Funcții de activare	18
1.3.5. Funcția de cost	19
1.4. Rețele neurale convoluționale	20
1.4.1. Stratul de convoluție	23
1.4.2. Straturile de pooling	24
1.4.3. Straturile fully-connected	24
1.4.4. Arhitecturi CNN	24
1.4.5. Transfer Learning	27
2. Mediul de dezvoltare	29
2.1. Python	29
2.2. OpenCV	31
2.3. TensorFlow	32
2.4. CUDA Toolkit și cuDNN	34
2.5. Baza de date	35

3. Implementare software	37
3.1. Dezvoltarea modelului	37
3.1.1. Achiziția și augmentarea datelor	38
3.1.2. Arhitectura rețelei	40
3.2. Antrenarea rețelei	42
3.2.1. Funcția de cost	42
3.2.2. Funcția de optimizare	42
3.2.3. Callbacks	43
3.3. Clasificarea emoției	44
3.3.1. Detectarea feței	44
3.3.2. Procesarea imaginii	45
3.4. Experimente și rezultate	46
3.5. Hiperparametri	47
3.6. Performanța rețelei	48
Concluzii	51
Concluzii generale	51
Dezvoltări ulterioare	51
Bibliografie	53
Anexa A	56
Codul sursă al scriptului de proiectare și antrenare a modelului	56
Codul sursă al aplicației de recunoaștere a emoției umane	61

Lista figurilor

1.	Detecția intervalelor RR în semnalul ECG. Baza de date AMIGOS[1], participant 1, video 10 [2].	3
2.	Detecția peak-urilor în semnalul GSR. Baza de date AMIGOS[1], participant 1, video 10 [2].	4
3.	Acuratețea modelului DCNN ce folosește semnale EEG și ECG pentru recunoșterea emoțiilor, participant 1, video 10[2].	4
4.	Exemple cu cele 7 emoții de bază: Fericit, Supărat, Neutru, Furios, Speriat, Surprins și Dezgustat [3]	5
5.	Evoluția funcției de cost și a acurateții modelului bazat pe arhitectura VGG [3]	6
6.	Modul de început, folosit pentru reducerea dimensionalității[4]	7
7.	Acuratețea de validare obținută în timpul procesului de antrenare[4]	8
8.	Modelul propus recunoaște în timp real emoțiile subiecților[4]	8
1.1.	Structura neuronului[5]	10
1.2.	Modelul neuronal McCulloch-Pitts[6]	11
1.3.	Implementarea porții logice NOR cu ajutorul modelului neuronal McCulloch-Pitts[6]	11
1.4.	Implementarea porții logice NAND cu ajutorul modelului neuronal McCulloch-Pitts[6]	12
1.5.	Simbolul general al neuronului artificial[6]	12
1.6.	Graficul fluxului de semnal al perceptronului[7]	13
1.7.	Reprezentarea suprafeței de decizie între două clase[7]	13
1.8.	Arhitectura rețelei MLP cu două straturi ascunse[7]	15
1.9.	Conexiunile neuronului "j" din stratul intermediar[8]	15
1.10.	Conexiunile neuronului "k" din stratul de ieșire[8]	15
1.11.	Funcția sigmoidă[8]	16
1.12.	Funcția tangentă hiperbolică[8]	16
1.13.	Funcții de activare[9]	18
1.14.	Gradientul negativ (Gradient Descent)[10]	20
1.15.	Exemplu de imagine RGB de dimensiune 4x4x3[11]	21
1.16.	Arhitectură CNN folosită pentru clasificarea cifrelor scrise de mână[11]	22
1.17.	Reprezentarea unui strat convoluțional[12]	23
1.18.	Aplicarea operației de max-pooling	24
1.19.	Arhitectura LeNet[13]	25
1.20.	Arhitectura AlexNet[14]	25

1.21. Arhitectura ZF Net[15]	26
1.22. Arhitectura GoogLeNet[16]	26
1.23. Arhitectura VGGNet[17]	27
1.24. Arhitectura ResNet-34[18]	27
2.1. Detectarea feței cu ajutorul librăriei OpenCV[19]	31
2.2. Grafic de calcul al fluxului de date. Datele de intrare sub formă de tensori sunt procesate de rețea[20]	32
2.3. Framework-ul Caffe rulat pe CPU, GPU și GPU + cuDNN [NVIDIA]	34
2.4. Exemple de imagini din baza de date Fer2013 [21]	35
3.1. Diagrama ilustrând structura generică a arhitecturii de clasificare	38
3.2. Reprezentarea funcției de activare Softmax[22]	42
3.3. Recunoașterea celor 5 emoții de către modelul CNN	47
3.4. Evoluția acurateții pe loturile de test și validare	47
3.5. Evoluția funcției de cost pe loturile de test și validare	48

Lista tabelelor

2.1. Cele mai comune operații făcute cu ajutorul TensorFlow și prescurtările acestora[20]	33
3.1. Performanțele rețelei și funcțiile de optimizare folosite	46

Lista acronimelor

ANN = din engleză: "Artificial Neural Network"

CNN = din engleză: "Convolutional Neural Network"

DCNN = din engleză: "Deep Convolutional Neural Network"

EEG = Electroencefalogramă

ECG = Electrocardiogramă

GSR = din engleză: "Galvanic Skin Response"

QRS = din engleză: "Q wave, R wave, S wave"

AI = din engleză: "Artificial Intelligence"

AU = din engleză: "Action Units"

PCA = algoritm nesupervizat, din engleză: "Principle Component Analysis"

JAFFE = baza de date, din engleză: "The Japanese Female Facial Expression"

CK+ = baza de date, din engleză: "The Extended Cohn-Kanade Dataset"

MLP = Perceptronul Multi-Strat, din engleză: "Multi-Layer Perceptron"

ReLU = funcție de activare, din engleză: "Rectified Linear Unit"

Introducere

Motivație

Această lucrare își propune să studieze sistemele de recunoaștere automată a emoțiilor umane bazate pe inteligență artificială - domeniu ce a capătat foarte multă popularitate în ultimii ani datorită modului în care a venit ca răspuns la nevoile omenirii tot mai mari de a crea sisteme inteligente, capabile să învețe și să ia decizii asemenea unei ființe umane.

Deoarece emoțiile unei persoane pot reprezenta de multe ori o sursă de informație ce poate oferi detalii despre o anumită trăire sau despre felul în care un om reacționează la anumiți stimuli, este foarte important ca acestea să poată fi analizate și determinate de un sistem inteligent într-un mod automat și cât mai precis. Din punctul de vedere al psihologiei, emoțiile au evoluat din procesul de homeostază, acestea constituind un puternic factor ce poate schimba felul în care un om răspunde la anumiți stimuli din mediul înconjurător, reușind totodată să fie și rezultatul unor procese cognitive ce pot oferi o multitudine de informații despre modul în care o ființă umană reacționează atunci când este pus în fața anumitor situații. Întrucât emoțiile pe care le trăim variază de la o persoană la alta și acestea pot oferi informații cheie în foarte multe domenii în care actorii principali sunt oamenii, determinarea și înțelegerea lor cu ajutorul unui sistem automat de recunoaștere a devenit o metoda extrem de utilă și eficientă.

Studiul acestor sisteme inteligente și încercarea de a implementa un astfel de sistem sunt motivate de faptul că există foarte aplicații unde este nevoie de clasificarea cât mai precisă a emoțiilor trăite de anumiți subiecți, iar principalul avantaj al acestei abordări automatizate este faptul că poate gestiona o cantitate foarte mare de date într-un timp mult mai scurt comparativ cu un om.

Aplicabilitate

Aceste tipuri de sisteme inteligente ce folosesc rețele neurale convoluționale au o aplicabilitate vastă în foarte multe domenii. Câteva dintre domeniile în care aceste sisteme pot fi aplicate sunt domeniul medical, domeniul turismului, domeniul marketingului, domeniul educațional precum și multe altele.

Un studiu relevant despre rețele neurale convoluționale și recunoașterea emoțiilor a fost publicat într-un articol din jurnalul academic al celor de la **IEEE Access**. [2]

În ultimii 20 de ani, mai mulți cercetători au evidențiat faptul că ar trebui să fie acordată mai multă atenție domeniului **Affective Computing**, aceștia urmărind cu mare interes comunitățile academice ce își doreau să facă progrese în acest domeniu. [23] Ei au reușit să obțină rezultate impresionante folosind tehnici și algoritmi din zona de deep machine learning, dezvoltând câteva

modele pentru recunoașterea emoțiilor. Unul dintre domeniile în care sistemul automat bazat pe rețele neurale convoluționale a fost folosit cu succes este domeniul turismului. Acest model DCNN analizează diferite date pentru detectarea stării afective a fiecărui turist cum ar fi variația bătăilor inimii, temperatura pielii dar și alte informații ce proveneau de la sistemul nervos periferic și sistemul nervos central, cum ar fi spre exemplu datele dintr-o electrocardiogramă. Tot mai multe studii care s-au făcut în ultimul timp în industria turismului au arătat faptul că aceste sisteme de recunoaștere automată a emoțiilor sunt extrem de benefice.[24] Pentru stabilirea emoției trăită de un turist într-o anumită locație, sistemul colectează date de la diferiți senzori analizând starea subiectului în 3 etape: înainte de a vizita destinația turistică, în timpul vacanței și după ce sejurul a luat sfârșit.[25]

Autorii studiului susțin că în funcție de anvergura destinației turistice, unele agenții vin în sprijinul turiștilor cu sisteme inteligente de tipul DCNN care le recomandă acestora o destinație turistică în funcție de emoțiile acestora. Se precizează totodată că Organizația Mondială a Turismului recunoaște că în ultimul timp tot mai mulți turiști tind să prefere beneficiile emoționale în dezafoarea atracțiilor fizice și a prețului.[26]

Un alt domeniu în care poate fi aplicat un astfel de sistem de recunoaștere automată a emoțiilor este cel al marketingului. Merchandiserii sau chiar reprezentanții magazinelor mari ar putea să aleagă folosirea unui sistem inteligent pentru a clasifica emoțiile clienților. Acest tip de abordare poate reprezenta o soluție viabilă de feedback pentru reprezentanții magazinelor atunci când vine vorba de introducerea unui produs nou sau chiar un semnal de alarmă în cazul unui produs care are cărui vânzări au scăzut. Totodată, există și posibilitatea aplicării acestui sistem de detectare a emoțiilor în momentul în care o persoană testează un anumit produs.

În domeniul medical, există foarte multe aplicații în care un sistem inteligent de tipul DCNN poate reprezenta cheia înțelegerii comportamentului persoanelor cu dificultăți de exprimare. Există foarte multe cazuri de copii cu dizabilități care ar putea fi monitorizați de un astfel de sistem în timp real, iar personalul medical care se ocupa de aceștia să fie anunțat atunci când starea emoțională a acestora se degradează sau se îmbunătățește. Monitorizarea stării emoționale ar putea constitui și un feedback cu privire la modul în care anumiți indivizi reacționează la un anumit tratament. O bază de date realizată printr-un procedeu de tipul **data fusion** între informațiile biometrice de la diferiți senzori, semnalele EEG/ECG și semnalul video ce monitorizează anumiți subiecți, ar putea constitui o bază de antrenare solidă pentru un sistem automat de recunoaștere a emoțiilor bazat pe rețele neurale convoluționale.

De asemenea, un alt rezultat remarcabil din domeniul medical obținut cu ajutorul rețelelor neurale convoluționale este identificarea și recunoașterea anumitor afecțiuni ale sistemului osos, pe baza radiografiilor. De-a lungul timpului, radiografiile au fost interpretate de medici specialiști în radiologie, dar în majoritatea cazurilor, interpretările au fost făcute în moduri mai mult sau mai puțin subiective, acest fapt conducând la dezvoltarea unor sisteme automate de recunoaștere a afecțiunilor bazate pe rețele neurale convoluționale. [27] Aceste sisteme s-au dovedit a fi extrem de folositoare și în determinarea anumitor afecțiuni ale unor organe cum ar fi ficatul, rinichii și plămânii. În majoritatea cazurilor, pentru a dezvolta un sistem capabil să identifice diferite afecțiuni sunt folosite modele pre-antrenate utilizându-se tehnica transferului de cunoștințe. Pe baza radiografiilor și a tomografiilor computerizate, un astfel de sistem inteligent poate detecta și chiar prezice apariția anumitor leziuni cu o precizie impresionantă. [28]

Așadar, aplicabilitatea vastă a rețelelor neurale convoluționale în domeniile ce ne marchează viața de zi cu zi, reprezintă dovada clară a faptului că în viitorul apropiat aceste sisteme inteligente, ce simulează felul în care ființa umană discernă și ia anumite decizii, vor fi existente în majoritatea domeniilor și ne vor ajuta să ne dezvoltăm ca civilizație facilitându-ne totodată accesul spre noi tehnologii.

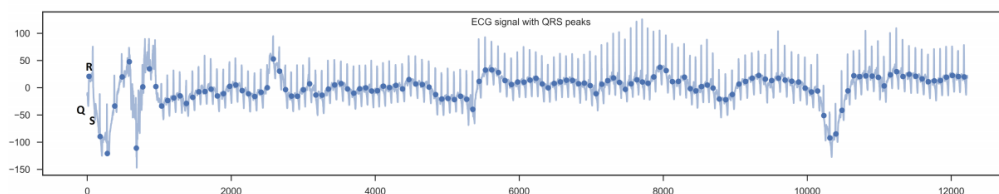


Figura 1: Detectția intervalelor RR în semnalul ECG. Baza de date AMIGOS[1], participant 1, video 10 [2].

Obiective

Obiectivul general al acestei lucrări este proiectarea și dezvoltarea unui sistem de recunoaștere a emoțiilor umane bazat pe rețele neurale convoluționale.

Atingerea acestui obiectiv general poate fi realizată prin îndeplinirea următoarelor obiective individuale:

- **Dezvoltarea unui model de clasificare:** Se va dezvolta un model capabil să clasifice emoția dintr-o imagine de o anumită dimensiune.
- **Obținerea și organizarea setului de antrenare:** Modelul va avea nevoie de baze de date specifice ce conțin imagini cu diferiți subiecți exprimând o anumită emoție.
- **Antrenarea modelului:** Modelul va fi antrenat cu ajutorul imaginilor existente în setul de antrenare. Acuratețea modelului va influența în mod direct felul în care sistemul va recunoaște emoțiile exprimate de subiecții din imagini.
- **Implementarea unor funcții de preprocesare:** Imaginile ce vor urma a fi clasificate de model trebuie să îndeplinească anumite condiții. Acestea vor fi redimensionate și procesate în funcție de cum a fost proiectat stratul de intrare al modelului.
- **Crearea unei interfețe grafice:** Toate aceste module individuale enumerate mai sus vor alcătui într-un final sistemul de recunoaștere a emoțiilor. Se va crea o interfață grafică ce va permite achiziția semnalului video sau pentru încărcarea unei anumite imagini specifice. Sistemul va detecta și va afișa emoția recunoscută.

Starea artei

Datorită aplicabilității vaste și a modului în care inteligența artificială s-a impus ca soluție principală în majoritatea domeniilor, există o sumedenie de implementări ale rețelelor neurale convoluționale care abordează diferite probleme. Există oameni din foarte multe domenii care cercetează rețelele neurale și modul în care acestea funcționează. Creierul uman este capabil să proceseze informații foarte rapid și precis, fapt ce a condus la permanenta dorință de a dezvolta sisteme bazate pe inteligență artificială. Creierul reușește de-a-lungul vieții unui om să învețe foarte multe lucruri și să folosească informația dobândită pentru a învăța să efectueze acțiuni mai complexe. Aceste rețele neurale încearcă să simuleze felul în care funcționează sistemul nervos dar ”au devenit obiect matematic” [8] instruindu-se ”sistematic prin utilizarea unor eșantioane de învățare” [8]. Un studiu relevant în domeniul inteligenței artificiale publicat în jurnalul academic IEEE Access [2] reușește să pună în evidența performanța superioară a rețelelor neurale convoluționale în comparație cu alte sisteme inteligente dezvoltate folosind arhitecturi și tehnologii diferite. Cercetătorii care au efectuat acest studiu comparativ și-au propus să dezvolte un sistem de recunoaștere a emoțiilor plecând de la diferite arhitecturi.

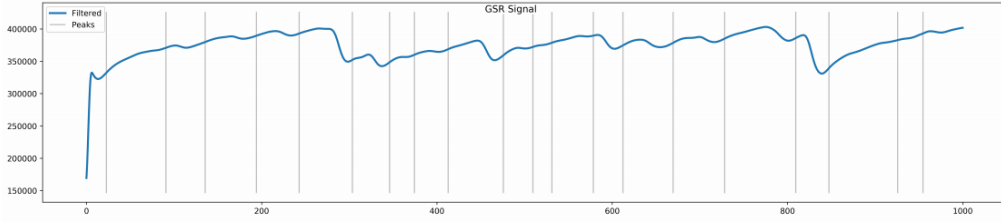


Figura 2: Detectia peak-urilor în semnalul GSR. Baza de date AMIGOS[1], participant 1, video 10 [2].

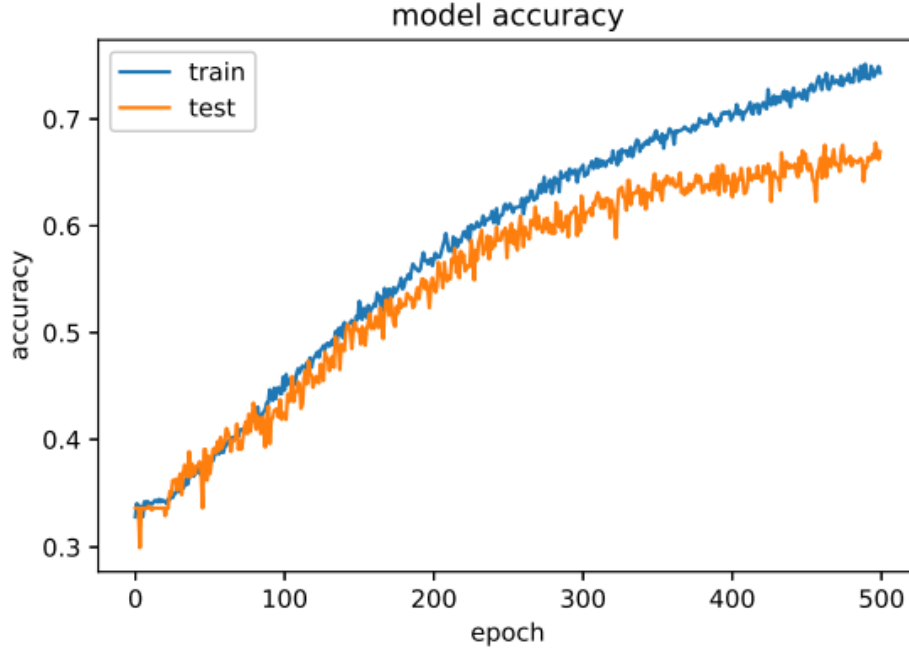


Figura 3: Acuratețea modelului DCNN ce folosește semnale EEG și ECG pentru recunoșterea emoțiilor, participant 1, video 10[2].

Cu ajutorul bazei de date AMIGOS[1], autorii studiului au reușit să realizeze un model DCNN ce prelucrează date psihologice cum ar fi semnalele EEG, ECG și GSR captate de senzori ce culegeau informații de la subiecți ce vizualizau o secvență dintr-un clip video pentru a observa ce emoții trăiesc aceștia. Aceștia au preprocesat datele folosind anumite funcții de detecție a semnalelor QRS, obținând în acest mod intervalele dintre două bătăi succesive ale inimii subiectului, mai exact intervalele RR ale ECG-ului. Asemănător, autorii studiului au identificat și seriile temporale ale varfurilor răspunsului conductanței pielii din semnalele GSR. Un factor important în clasificarea corectă a emoției trăite de subiect, este corelarea caracteristicilor obținute în urma analizei semnalelor ECG și GSR. [2]

În figura 1 se poate observa detecția intervalelor RR dintr-un semnal ECG aparținând unui anumit subiect din baza de date AMIGOS[1]. Caracteristicile extrase se corelează cu cele rezultate în urma analizei semnalului GSR ce poate fi observat în figura 2. Este de menționat faptul că modelul propus a obținut o acuratețe superioară altor modele (aproximativ 70%), folosind 90% din baza de date pentru test și restul de 10% pentru validare. În figura 3 se poate observa creșterea exponențială a acurateții modelului în timpul procesului de antrenare și testare pentru 500 de epoci. [2]

În prezent, datorită aplicabilității vaste, există foarte multe modele ce folosesc rețelele neuronale convoluționale pentru recunoașterea de emoții. Din cauza faptului că implementarea și



Figura 4: Exemple cu cele 7 emoții de bază: Fericit, Supărat, Neutru, Furios, Speriat, Surprins și Dezgustat [3]

antrenarea unui model "from scratch" sunt niște procedee ce necesită mult timp și resurse de calcul mari, cei mai mulți dezvoltători de sisteme ce folosesc CNN-uri aleg să folosească procedura transferului de cunoștințe(sau transfer learning).

Creierul uman dobândește de-alungul vieții foarte multe cunoștințe și abilități, dar pentru a ajunge să fie capabil să gestioneze și să asimileze informații mai complexe, este necesar ca acesta să aibă câteva cunoștințe de bază. Prin transfer learning, o rețea neurală poate fi proiectată folosind cunoștințele dobândite de o altă rețea care a fost antrenată și "deține" mult mai multe informații, acest lucru ajutând-o să asimileze și să învețe să facă alte lucruri noi mult mai rapid. Pentru ca o rețea neurală care a fost specializată pe un anumit task să poată fi folosită cu un alt scop, este nevoie de adăugarea unui clasificator specific task-ului pentru care a fost proiectată noua rețea și de antrenarea acesteia. Toate straturile și ponderile, mai puțin ultimul strat care va fi înlocuit de clasificatorul rețelei noi, vor rămâne neschimbate. Există două proceduri prin care o rețea pre-antrenată poate fi folosită pentru un nou task: fine-tuning și feature extraction. Aceste proceduri sunt explicate în detaliu în Capitolul 1.

O abordare asemănătoare au ales și autorii unui studiu de cercetare realizat în cadrul Universității Purdue, Indiana[3]. Deoarece procesul de antrenare al unei rețele "from scratch" durează foarte mult iar rezultatele sunt de multe ori nesatisfăcătoare, aceștia au ales să aplice procedura de fine tuning rețelei VGG-16, proiectată și antrenată în cadrul Universității Oxford.[3]

Autorii studiului au păstrat majoritatea straturilor de extragere a caracteristicilor intacte, mai exact primele 4 block-uri de convoluție și pooling, reantrenând ultimul strat de convoluție precum și stratul fully-connected, care reprezintă de fapt clasificatorul. În figura 4 se pot observa cele 7 emoții de bază pe care modelul propus este capabil să le recunoască: fericit, trist, furios, surprins, dezgustat, neutru și speriat.

Ca orice alta problema de clasificare, recunoașterea emoțiilor necesită folosirea unor algoritmi de extragere a caracteristicilor și de efectuare a clasificării propriu-zise. În general, pentru a putea clasifica o emoție, este nevoie de extragerea anumitor caracteristici dintr-o imagine și implementarea unui model care este capabil să clasifice pe baza acestor caracteristici. Dezavantajul folosirii altor rețele neurale decât cele convoluționale atunci când inputul rețelei este o imagine este că acestea au nevoie de extragerea anumitor AU-uri din imagine, clasificarea emoției făcându-se pe baza combinațiilor acestor acțiuni individuale.[3] Folosirea unor algoritmi clasici nesupervizați precum PCA, care extrag aceste caracteristici individuale apare atunci când nu mai sunt îndeplinite condițiile de laborator, iar mediul de testare devine unul real. Din cauza faptului că background-ul, lumina și zgomotul vor fi diferite de la o imagine la alta, un clasificator clasic va avea dificultăți în recunoașterea emoțiilor și chiar dacă în condiții optime acesta a obținut un scor bun, în realitate nu se va descurca la fel de bine.

În cazul rețelelor neurale convoluționale, această problemă nu apare deoarece procesul de extragere a caracteristicilor se desfășoară automat iar modelul matematic din spatele acestor rețele, specific tuturor arhitecturilor de tip deep learning, este unul ce combină transformări liniare cu funcții neliniare pentru a remarcă diferențele ce apar în procesul de clasificare. [3]



Figura 5: Evoluția funcției de cost și a acurateții modelului bazat pe arhitectura VGG [3]

Performanța modelului propus de autorii studiului se poate observa în figura 5, aceștia obținând în condiții de laborator o acuratețe pe setul de antrenare de aproximativ 95% și o acuratețe de validare în jur de 81%. Modelul a fost antrenat pe doua baze de date (JAFPE și CK+).

O altă aplicație interesantă a sistemelor de recunoaștere a emoțiilor bazate pe rețele neurale convoluționale este prezentată într-un studiu ce tratează acest subiect și dintr-o perspectivă real-time.[4] Înțelegerea emoțiilor trăite de un om este un lucru esențial în condițiile în care tot mai multe activități din viața noastră sunt marcate de interacțiunea om-robot. În prezent, există foarte mulți așa-zisi roboți personali care ajută oamenii la diverse activități, cum ar fi educarea propriilor copii. După cum am precizat și mai sus, există foarte mulți copii cu problema cum ar fi autismul, iar înțelegerea emoțiilor lor este un lucru esențial în procesul de recuperare. De asemenea, acești roboți asistenți, își pot modela mult mai bine și mai eficient acțiunile dacă sunt echipați cu un sistem de recunoaștere a emoțiilor în timp real.[4] Aplicațiile de tip deep-learning își propun să imite cât mai bine și mai precis felul în care creierul uman funcționează, drept urmare, implementarea unor astfel de sisteme cât mai performante a devenit o necesitate. Datorită modului în care prelucrează imaginile și pot face predicții pe baza acestora, rețelele neurale convoluționale sunt soluția perfectă pentru majoritatea aplicațiilor computer vision. Este de remarcat faptul că în momentul în care roboții vor reuși să înțeleagă în totalitate emoțiile umane, oamenii se vor putea conecta din punct de vedere emoțional cu aceștia, vor avea mult mai multă încredere în ei și drept urmare, aplicațiile bazate pe interacțiunea dintre robot și om se vor bucura de ceva mai multă popularitate și succes. Totodată, casele inteligente vor fi dotate cu tot mai multe gadgeturi și roboți personali care ne vor ajuta și ne vor face să ne desfășurăm activitățile zilnice într-un mod mai ușor și mai relaxant. Înțelegerea emoțiilor umane de către un sistem automat poate ajuta la menținerea unei atmosfere ambientale plăcute. Acești roboți pot gestiona lumina și chiar muzica din casă în funcție de emoția și starea proprietarului. [4] Pentru a putea realiza toate aceste aplicații, este nevoie ca acești roboți personali să recunoască emoțiile în timp real, pentru ca informația obținută în urma predicției să poată fi relevantă. Autorii studiului au implementat un model ținând cont de acest aspect, dar dorindu-se totodată ca modelul să aibă o acuratețe bună pentru a putea reprezenta o soluție fezabilă pentru un sistem de recunoaștere a emoțiilor în timp real. Pentru ca sistemul să poată fi încorporat pe sisteme de tip embedded sau pe roboți, aceștia i-au testat performanțele pe 8 baze de date diferite, mai exact pe bazele de date Fer2013[29], CK și CK+[30], Chicago Face Database[31], JAFPE Dataset[32], FEI face dataset[33], IMFDB[34], TFEID[35] și pe o bază de date custom cu imagini realizate în condiții de laborator.[4] Pentru a reduce complexitatea computațională și pentru a atinge cea mai bună acuratețe, aceștia au folosit diferite tehnici și metode, folosind o tehnică inspirată din așa-zisul inception module.[36]

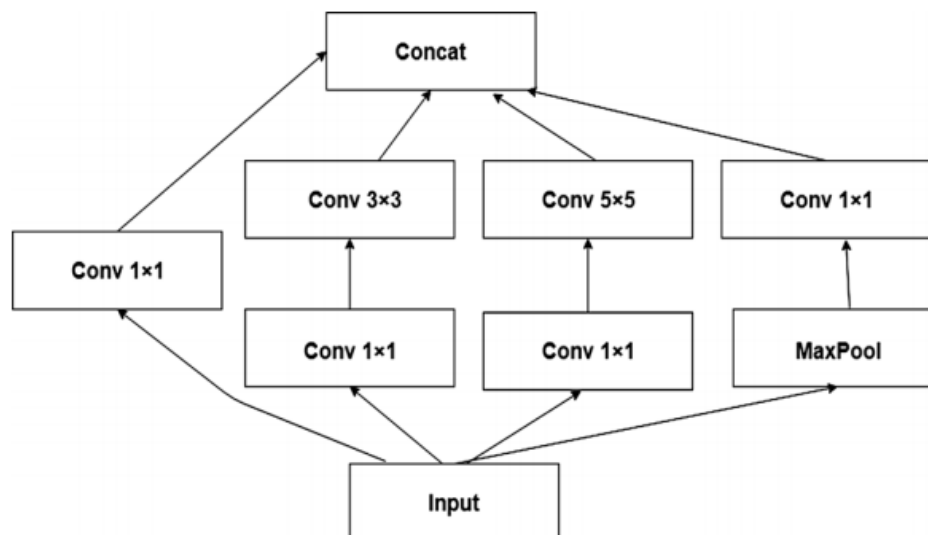


Figura 6: Modul de inceput, folosit pentru reducerea dimensionalității[4]

Modulele de început sunt folosite în arhitecturile bazate pe rețele neurale convoluționale pentru a eficientiza puterea computațională. Aceste module au fost create pentru a reduce puterea de calcul folosită de un sistem atunci când lucrează cu rețele neurale convoluționale, dar și pentru a veni ca o soluție la alte probleme cum ar fi cea a overfitting-ului. Tehnica folosită este relativ simplă și poate fi observată în figura 6. Pe scurt, se iau mai multe filtre kernel de dimensiuni diferite folosite în arhitectura CNN și mai degabră decât să fie folosite secvențial, acestea sunt proiectate să opereze la același nivel.

Autorii studiului susțin că pentru a reduce complexitatea de calcul și a reuși într-un final să obțină un model ce poate detecta emoțiile umane în timp real, s-au inspirat din așa-zisul inception model, descris mai bine în articolul [36]. Ei au implementat convoluții 1×1 , înainte de a folosi convoluții de dimensiuni mult mai mari, reducând astfel dimensionalitatea. În acest fel, numărul înmulțirilor a fost redus de 10 ori, reușind astfel să reducă complexitatea computațională a arhitecturii.

Modelul propus a obținut o acuratețe de aproximativ 64% în procesul de antrenare și o acuratețe de validare de aproximativ 74%. Evoluția acurateții poate fi observată în figura 7. Cea mai mică valoare a funcției de cost este în jur de 0.712. Baza de date pe care acesta a fost antrenat este Fer2013[29], iar durată totală a procesului de antrenare a fost de aproximativ 8 ore, antrenarea modelului efectuându-se pe un server dotat cu o placă video NVIDIA GeForce GTX 680.

În final, autorii susțin că au reușit să obțină un model cu o acuratețe îmbunătățită în comparație cu alte modele existente, arhitectura dezvoltată de aceștia având un număr de parametri cu aproximativ 50% mai mic față de alte arhitecturi existente. Acest lucru și faptul că au folosit tehnici și metode de reducere a puterii de calcul, îi permit sistemului bazat pe rețele neurale convoluționale să recunoască emoții în timp real chiar și atunci când în cadru sunt mai mulți oameni, acest lucru putând fi observat în figura 8. Cercetătorii punctează totodată și faptul că nevoia ca roboții să înțeleagă emoțiile exprimate de om va crește atât timp cât interacțiunile dintre om și robot vor fi din ce în ce mai prezente în viețile noastre. Se precizează în același studiu că pe viitor, vor încerca să folosească acest model sau o variantă îmbunătățită a acestuia pe robotul NaO, aceștia apreciind totodată că roboții vor fi făcuți mult mai sociabili o dată ce aceștia vor putea să reacționează într-o manieră cât mai realistă atunci când interacționează cu omul.[4]

Cea mai bună variantă a unui astfel de robot este Sophia, dezvoltat de compania Hanson

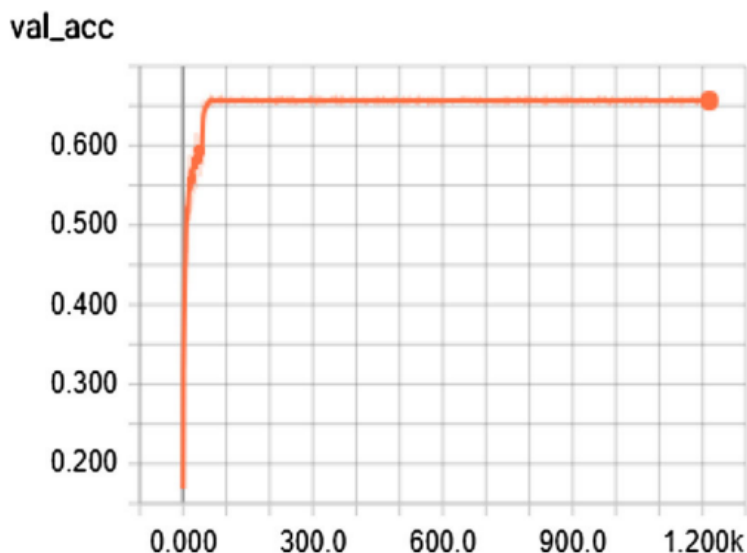


Figura 7: Acuratețea de validare obținută în timpul procesului de antrenare[4]



Figura 8: Modelul propus recunoaște în timp real emoțiile subiecților[4]

Robotics. Aceasta poate înțelege și chiar exprima emoții umane prin intermediul unui sistem bazat pe inteligența artificială și rețele neurale convoluționale. Momentan, acest robot poate fi considerat doar o minune a ingineriei și a tehnologiei actuale, el nefiind disponibil la scară largă, însă pe viitor cu siguranța vor exista astfel de roboți capabili să recunoască emoții umane, făcând în acest fel ca interacțiunea om-robot să devină mult mai dinamică și mai productivă.

Așadar, putem spune că rețelele neurale convoluționale stau la baza multor sisteme inteligente, existând în acest sens foarte multe modele ce folosesc vederea computerizată sau inteligența artificială pentru a rezolva anumite probleme cum ar fi recunoașterea emoțiilor umane. Rețelele neurale convoluționale sunt specializate în mare parte pe probleme de pattern recognition, dar pot reprezenta și un model de bază pentru rețele neurale hybrid cu ajutorul cărora se pot rezolva probleme complexe. Modelul CNN se caracterizează printr-un număr relativ mic de parametri în comparație cu alte rețele neurale, însă performanțele de recunoaștere ale acestuia, în special pe task-uri ce implica folosirea imaginilor, sunt mult mai bune.

În concluzie, trebuie să remarcăm faptul că în prezent există foarte multe aplicații ale CNN-urilor, unele dintre ele arătând performanțe impresionante. Majoritatea sistemelor de recunoaștere a emoțiilor ce folosesc rețele neurale convoluționale sunt antrenate pe imagini grayscale de dimensiune relativ mică, din bazele de date enumerate mai sus, iar performanțele acestora sunt de remarcat. Aceste sisteme bazate pe inteligența artificială reprezintă, cu siguranța, viitorul tehnologiei și pot constitui o soluție fezabilă pentru o varietate de probleme a căror rezolvare încă sta sub semnul întrebării.

Capitolul 1

Noțiuni teoretice

1.1 Istoric

Deși primele informații despre creierul uman datează de sute de ani, informațiile despre conceptul de rețea neurală au apărut în anul 1943, când Warren McCulloch, de profesie neurofiziolog, împreună cu Walter Pitts, de profesie matematician, au scris un articol despre cum ar putea neuronii să funcționeze. Aceștia au elaborat un model primitiv de rețea neurală folosind circuite electrice. După 6 ani de la publicarea acestui studiu, Donald Hebb a adus completări asupra conceptului de neuron în cartea sa, *The Organization of Behavior*, susținând o teorie conform căreia căile neuronale sunt consolidate pe măsura ce sunt folosite. În următorii ani, Nathaniel Rochester și John von Neumann au contribuit fiecare la dezvoltarea inteligenței artificiale, primul dintre aceștia reușind la începutul anilor 1950 să simuleze modul în care funcționează o rețea neurală, iar John von Neumann venind cu o sugestie asupra faptului că funcțiile neuronilor ar putea fi simulate folosind relee telegrafice și tuburi vidate. În 1958, Frank Rosenblatt, de profesie neurobiolog, a reușit să pună bazele perceptronului, cea mai veche rețea neurală folosită și astăzi. Principiul de funcționare era relativ simplu: era folosit un perceptron cu un singur strat care era capabil să clasifice un set de inputuri în una sau două clase, calculând o sumă ponderată a intrărilor, scăzând un prag setat și returnând într-un final două valori posibile ca rezultat. A urmat apoi în 1959 crearea modelelor *Adaline* și *Madaline*, ale cercetătorilor de la Universitatea Stanford, Bernard Widrow și Marcian Hoff, *Madaline* fiind primul model de rețea neurală aplicat pe o problema reală, și anume eliminarea ecourilor de pe liniile telefonice. După o perioadă în care studiile de cercetare în domeniul inteligenței artificiale au fost abandonate din cauza faptului că majoritatea statelor erau nemulțumite de rezultatele obținute, ba chiar unele dintre acestea considerau periculos acest domeniu, în anul 1987, IEEE a susținut o conferință pe tema rețelelor neurale care s-a bucurat de prezența a peste 1800 de participanți. În următorii ani, domeniul inteligenței artificiale a cunoscut o evoluție exponențială, cele mai remarcabile contribuții în acest domeniu fiind aduse atât de Schmidhuber și Hochreiter, cu lucrarea *A recurrent neural network framework, Long Short-Term Memory (LSTM)* publicată în anul 1997, cât și de Yann LeCun, cu lucrarea *Gradient-Based Learning Applied to Document Recognition*, publicată un an mai târziu, în 1998.[37]

În prezent, rețele neurale se bucură de foarte mare popularitate, majoritatea oamenilor de știință încercând să dezvolte an de an arhitecturi din ce în ce mai bune, capabile să rezolve diferite probleme. Rețelele neurale convoluționale, alături de alte arhitecturi de tip *deep learning*, sunt astăzi unele dintre cele mai puternice rețele neurale, arătând performanțe de recunoaștere superioare altor arhitecturi.

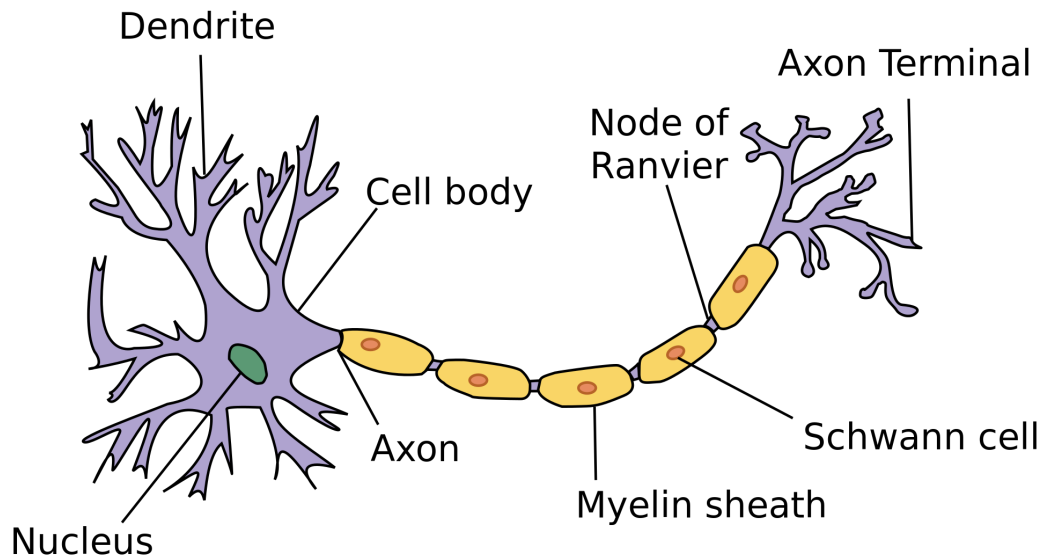


Figura 1.1: Structura neuronului[5]

1.2 Analogie biologică

Din punctul de vedere al biologiei, neuronul este o celulă nervoasă excitabilă electric care comunică cu celelalte celule prin conexiuni specifice numite sinapse. În figura 1.1, se poate observa mai bine structura neuronului și părțile lui componente. Creierul uman este alcătuit din circa 100 de miliarde de astfel de celule nervoase. În funcție de task-ul pe care îl execută, neuronii sunt împărțiți în 3 categorii: neuroni senzoriali, specializați în gestionarea stimulilor precum lumina sau sunetul și trimiterea de semnale către organele senzoriale, neuroni motori care primesc semnale electrice pentru a comanda contracția anumitor mușchi și interneuroni conectați cu alți neuroni din aceeași regiune a creierului. Neuronii comunică între ei prin intermediul unor semnale electrice sub forma unor impulsuri de tensiune de scurtă durată care excită membrana celulei, acest procedeu determinând realizarea unei acțiuni specifice fiecărui bloc de neuroni din creier. Sinapsele reprezintă legăturile dintre neuroni și sunt din punct de vedere structural niște joncțiuni electrochimice aflate pe ramificațiile celulei (dendrite). De obicei, fiecare neuron are foarte multe conexiuni (de ordinul miilor) care sunt reprezentate sub forma de semnale electrice. Prin urmare, fiecare neuron va recepționa aceste semnale primite de la alți neuroni, cu ajutorul corpului celular. Aceste semnale primite de neuronul receptor, sunt apoi procesate și adunate, iar dacă acestea depășesc un anumit prag, neuronul va răspunde printr-un impuls electric. Acest impuls de tensiune este transmis mai departe altor neuroni prin intermediul unei terminații nervoase, numită axon. [38]

Așadar, putem spune că structura unui neuron este alcătuită din următoarele elemente principale: dendrită, corp celular, sinapse și axon. Modelul neuronului artificial este inspirat din această structură, el având o intrare, un corp celular care prelucrează într-un anumit fel informația primită și o ieșire. Între intrare, corpul celular și ieșire există conexiuni care joacă rolul sinapselor din neuronul biologic. Fiecărei astfel de conexiuni îi este asociată o pondere. Tăria sinaptică asociată fiecărei sinapse este înlocuită în modelul neuronului artificial cu aceste ponderi. [39]

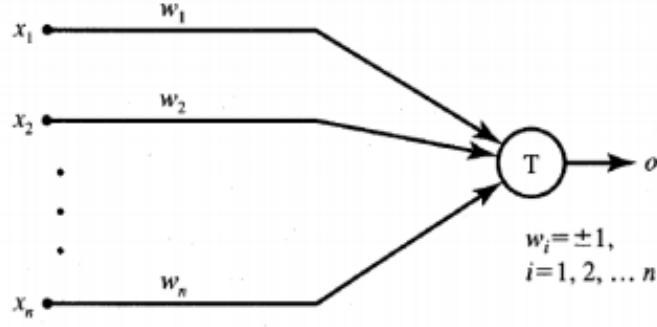


Figura 1.2: Modelul neuronal McCulloch-Pitts[6]

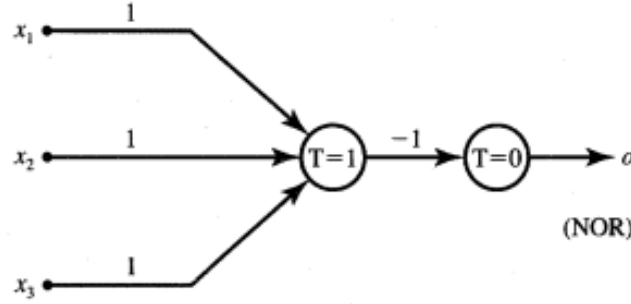


Figura 1.3: Implementarea porții logice NOR cu ajutorul modelului neuronal McCulloch-Pitts[6]

1.3 Rețele neurale clasice

1.3.1 Modelul neuronal McCulloch-Pitts

Acest tip de neuron artificial, inspirat din modelul neuronului biologic, a fost definit de Warren McCulloch și Walter Pitts în anul 1943. În figura 1.2, se poate observa modelul neuronului McCulloch-Pitts. Întrările x_i , $i = 1, 2, \dots, n$, pot avea valoarea 0 sau 1, în funcție de absența sau prezența impulsului de intrare la momentul k . Ieșirea acestui neuron este notată cu o , iar regula de activare este definită astfel:

$$o^{k+1} = \begin{cases} 1 & \text{dacă } \sum_{i=1}^n w_i x_i^k \geq T \\ 0 & \text{dacă } \sum_{i=1}^n w_i x_i^k < T \end{cases} \quad (1.1)$$

Acest model neuronal, pe lângă intrările x_i și ieșirea o , este caracterizat și de ponderile $w_i = 1$ corespunzătoare sinapselor excitatorii, de ponderile $w_i = -1$ corespunzătoare sinapselor inhibitorii și de pragul T a cărui valoare trebuie să fie cel puțin egală cu suma ponderată a valorilor de la intrare pentru ca neuronul să se activeze. Deși acest model neuronal este destul de simplist, acesta poate realiza operații logice de bază cum ar fi NOT, OR și AND dar trebuie avut în vedere ca ponderile și valorile de la intrare să fie selectate corespunzător. Este evident faptul că folosind cele trei operații logice enumerate anterior, mai exact, o combinație între NOT și OR sau între NOT și AND putem obține noi tipuri de porți. În figurile 1.3 și 1.4, se poate observa implementarea porților NOR și NAND cu 3 intrări.

Ponderile și pragul acestui model de neuron sunt reprezentate de niște valori fixe, astfel încât putem spune ca nu există interacțiuni în interiorul rețelei, singurele elemente care variază fiind reprezentate de valorile semnalului de intrare. Totuși, acest model poate fi considerat un model de bază pentru rețelele neuronale clasice. Rețelele neurale artificiale (ANN) și algoritmi de calcul folosesc tehnici mult mai variate și avansate comparativ cu acest model prezentat.

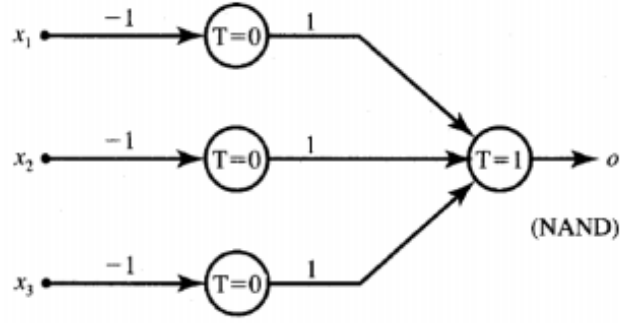


Figura 1.4: Implementarea porții logice NAND cu ajutorul modelului neuronal McCulloch-Pitts[6]

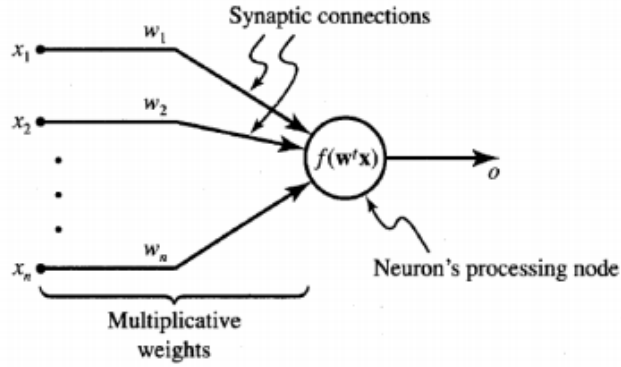


Figura 1.5: Simbolul general al neuronului artificial[6]

Este de menționat faptul că modelul neuronal McCulloch-Pitts stă la baza următoarelor modele și vom considera neuronul din figura 1.5 ca fiind simbolul general al neuronului artificial.

1.3.2 Perceptronul

Perceptronul, a fost introdus de Frank Rosenblatt și reprezintă cea mai simplă și cea mai veche formă de rețea neurală. Acesta are la bază modelul neuronal McCulloch-Pitts și este utilizat pentru clasificarea vectorilor linear separabili (vectori care se afla pe părți opuse ale unui hiperplan). Diferența dintre modelul McCulloch-Pitts și perceptronul definit de Rosenblatt este că în cazul perceptronului există un algoritm de învățare prin care se stabilesc ponderile pe baza unor exemple, în funcție de ce comportament se dorește să adopte acesta. Perceptronul lui Rosenblatt presupunea folosirea unui combinator linear urmat de un limitator hard. Nodul de însumare al acestui model calcula o combinație liniară a intrărilor aplicate pe sinapsele sale, dar avea integrat în el și un semnal bias aplicat din exterior. În continuare, suma rezultată era aplicată la intrarea unui limitator hard. În consecință, ieșirea neuronului este +1, dacă intrarea în limitatorul hard este pozitivă, și -1 dacă este negativă. În figura 1.6, se poate observa că ponderile sinaptice ale acestui model sunt notate cu w_1, w_2, \dots, w_m iar semnalele aplicate la intrarea perceptronului sunt notate cu x_1, x_2, \dots, x_m . Biasul extern aplicat este notat cu b . [7] Din arhitectura modelului, se poate observa ușor că intrarea limitatorului hard, sau *câmpul local indus*, se calculează astfel:

$$v = \sum_{i=1}^m w_i x_i + b \quad (1.2)$$

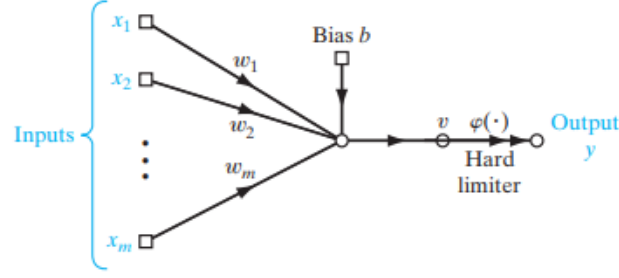


Figura 1.6: Graficul fluxului de semnal al perceptronului[7]

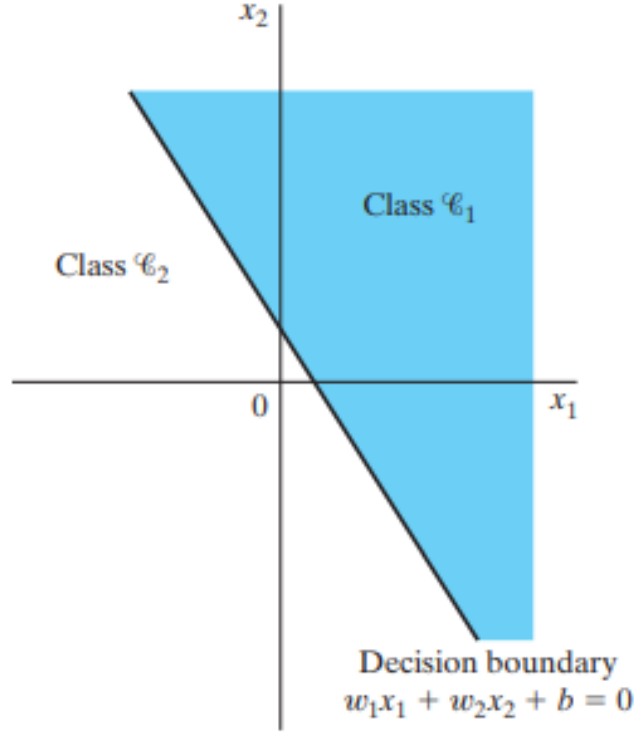


Figura 1.7: Reprezentarea suprafeței de decizie între doua clase[7]

Rosenblatt a demonstrat faptul că dacă vectorii de intrare, folosiți pentru antrenarea modelului, formează două clase liniar separabile, atunci algoritmul de clasificare al perceptronului converge și poziționează suprafața de decizie dintre cele două clase sub forma unui hiperplan definit de următoare ecuație:

$$\sum_{i=1}^m w_i x_i + b = 0 \quad (1.3)$$

În cazul în care la intrarea perceptronului avem două variabile x_1 și x_2 , suprafața de separare între cele două clase va fi reprezentată de o dreaptă, acest lucru putând fi observat în figura 1.7. În acest fel, orice punct care se află deasupra dreptei va aparține clasei c_1 , iar orice punct care se afla sub dreapta care definește suprafața de separare, va aparține clasei c_2 . Rolul utilizării bias-ului extern aplicat este să îndepărteze dreapta de separare de origine. Ponderile perceptronului, w_1, w_2, \dots, w_m , pot fi adaptate pentru o cerință specifică folosind un algoritm iterativ. Pentru realizarea acestei adaptări, este folosită o regulă de corectare a erorilor numită algoritm de convergență al perceptronului.[7]

Pentru a corecta erorile de decizie, se introduce în timpul antrenării o variabilă de intrare și se observă setul de valori obținut la ieșire. În cazul în care ieșirea obținută diferă de ieșirea dorită, ponderile sunt reactualizate după cum a fost descris mai sus. Dacă între ieșirea rețelei și ieșirea ideală nu există nicio diferență, nu se efectuează nicio modificare a ponderilor. [40]

1.3.3 Perceptronul Multi-Strat (MLP)

O altă rețea neurală clasică este arhitectura MLP. În comparație cu modelul clasic al perceptronului care poate fi considerat practic o rețea neurală cu un singur strat, arhitectura MLP folosește după cum o sugerează și numele acesteia, mai multe straturi pentru clasificarea unei anumite intrări. În figura 1.8, se poate observa structura rețelei MLP și modul în care sunt interconectați neuronii. Această arhitectură este net superioară perceptronului clasic care are anumite limitări, în sensul că poate clasifica doar *pattern-uri* ce sunt separabile liniar.[7] Arhitectura MLP a fost dezvoltată pentru depășirea acestor limitări, și este caracterizată prin următoarele particularități:

- **Funcție de activare nonliniară:** Fiecare neuron al arhitecturii dispune de o funcție de activare nonliniară care este *diferențiabilă*.
- **Straturi ascunse:** Rețeaua MLP conține unul sau mai multe straturi (hidden layers) care sunt ascunse atât de ieșire cât și de intrare .
- **Conectivitate:** Rețeaua prezintă un grad ridicat de conectivitate, măsura acestuia fiind influențată în mod direct de ponderile sinaptice ale rețelei.

Aceste caracteristici se fac totodată și răspunzătoare pentru deficitele cunoștințelor noastre în ceea ce privește comportamentul acestor rețele neurale. În primul rând, prezența unei forme distribuite de neliniaritate și gradul ridicat de conectivitate între elementele rețelei fac dificilă înțelegerea arhitecturii din punct de vedere teoretic. În al doilea rând, din cauza prezenței straturilor ascunse, procesul de învățare este foarte greu de supravegheat. O metodă populară de antrenare a arhitecturilor MLP este algoritmul de tip *back-propagation*. [7]

În faza *forward*, ponderile sinaptice ale rețelei sunt fixe și inputul se propagă din layer în layer până când ajunge la ieșire. În faza *backward*, un semnal de eroare este produs după ce ieșirea rețelei este comparată cu ieșirea dorită. Acest semnal de eroare se propagă de această dată înapoi prin rețea, layer cu layer, până când ajunge la input-ul rețelei. De această dată însă, apar modificări succesive ale ponderilor sinaptice. Calculul acestor modificări pentru layer-ul de ieșire poate părea o chestiune simplă, însă pentru straturile ascunse este mult mai complicat. [7]

Pentru clasificarea unui anumit input, este nevoie de definirea ecuațiilor neuronilor din stratul intermediar, împărțite într-o fază liniară α și o fază neliniară β . Conexiunile unui neuron din stratul intermediar pot fi observate în figura 1.9. În mod similar, sunt definite ecuațiile neuronilor din stratul de ieșire, acest proces de prelucrare fiind de asemenea împărțit într-o fază liniară γ și o fază neliniară δ . Conexiunile unui neuron din stratul de ieșire pot fi observate în figura 1.10. [8]

În continuare, înainte de a defini ecuațiile neuronilor din stratul intermediar și stratul de ieșire, se definesc următoarele variabile:

- $X_p = (x_{p1}, \dots, x_{pn})^t$: vectorul care se aplica la intrarea rețelei
- $W_{ji}^h, j = 1, \dots, L; i = 1, \dots, n$: Mulțimea ponderilor aferente stratului intermediar. L reprezintă numărul de neuroni din stratul intermediar și n este numărul neuronilor din stratul de intrare. Indicele superior " h " provine de la "*hidden*".

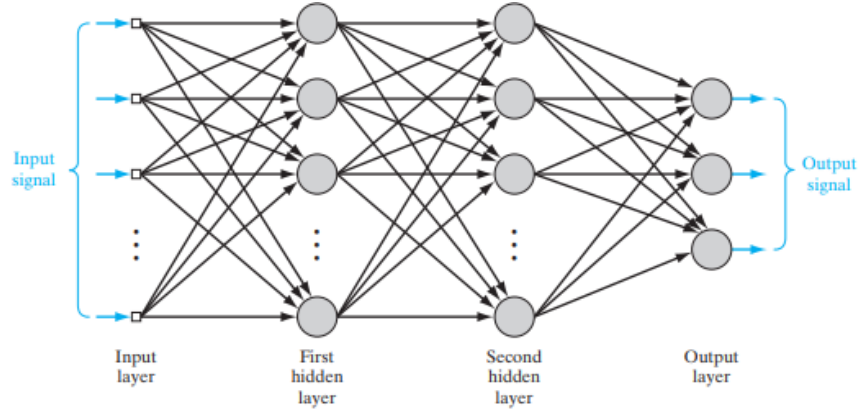


Figura 1.8: Arhitectura rețelei MLP cu doua straturi ascunse[7]

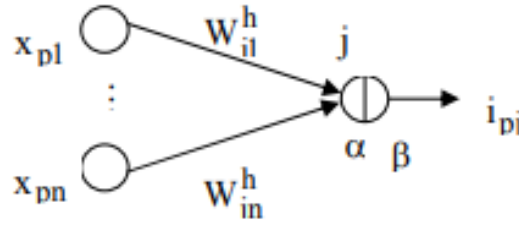


Figura 1.9: Conexiunile neuronului "j" din stratul intermediar[8]

- $W_{kj}^o, k = 1, \dots, M; j = 1, \dots, L$: Mulțimea ponderilor aferente stratului de ieșire. M reprezintă numărul de neuroni ai stratului de ieșire iar o provine de la "output".

În etapa liniară α , vom defini ecuația de activare a neuronului j din stratul intermediar, când la intrare avem vectorul X_p :

$$\text{net}_{pj}^h = \langle W_j^h, X_p \rangle = [W_j^h]^t \cdot X_p, j = 1, \dots, L \quad (1.4)$$

Etapa neliniară β presupune calcularea ieșirii neuronului intermediar cu ajutorul următoarei formule:

$$i_{pj} = f_j^h(\text{net}_{pj}^h), j = 1, \dots, L \quad (1.5)$$

Funcția f_j^h este o funcție neliniară. În figura 1.11, se poate observa o neliniaritate sigmoidală, iar în figura 1.12, o neliniaritate de tip tangenta hiperbolică. [8]

Dupa cum am menționat mai sus, procesul de prelucrare al neuronilor din stratul de ieșire se împarte, de asemenea, într-o etapă liniară și o etapă neliniară.

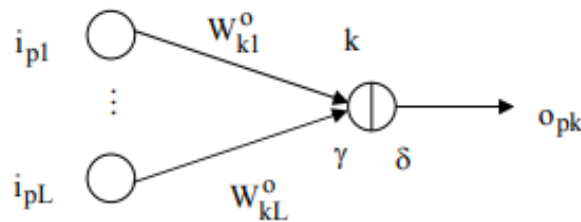


Figura 1.10: Conexiunile neuronului "k" din stratul de ieșire[8]

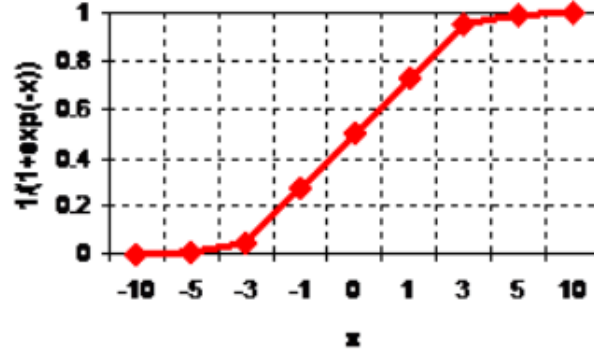


Figura 1.11: Funcția sigmoidă[8]

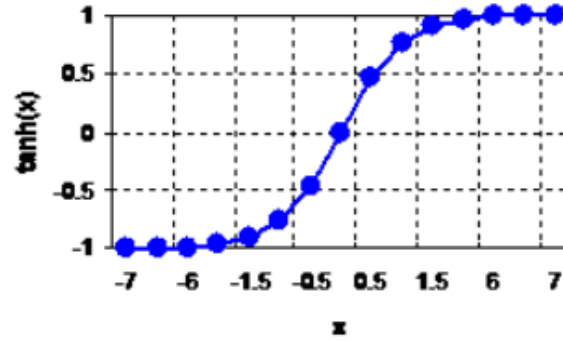


Figura 1.12: Funcția tangentă hiperbolică[8]

În faza liniară γ vom defini ecuația de activare a neuronului k din stratul de ieșire, când la intrare avem vectorul X_p :

$$\text{net}_{pk}^o = \sum_{j=1}^L W_{kj}^o \cdot i_{pj}, k = 1, \dots, M \quad (1.6)$$

În etapa neliniară δ rezultatul neuronului de ieșire k depinde de activarea variabilei net_{ph}^o și se calculează folosind următoarea formulă:

$$o_{pk} = f_k^o(\text{net}_{pk}^o), k = 1, \dots, M \quad (1.7)$$

Perceptronul multi-strat (multinivel) are un algoritm de instruire supervizată, de tip *back-propagation*, pentru a se minimiza eroarea pe lotul de instruire.[8] Acest algoritm de tip asigură o aproximare a traiectoriei calculate în spațiul ponderilor folosind metoda celei mai abrupte coborărări, numită și **gradient descent**. Cu cât rata de învățare η este mai mică, cu atât mai puțin se schimbă ponderile sinaptice ale rețelei de la o iterație la alta, în acest fel traiectoria din spațiul ponderilor fiind mai lină.[7]

Acest algoritm de instruire supervizată este alcătuit din următorii pași:

Pasul 1. Se inițializează $p = 1$. La intrarea rețelei este aplicat vectorul $X_p = (x_{p1}, \dots, x_{pm})^T$ pentru care se cunoaște vectorul de ieșire ideal $Y_p = (y_{p1}, \dots, y_{pM})^T$, unde n reprezintă dimensiunea vectorului de intrare, iar M reprezintă numărul de clase.[8]

Pasul 2. Se calculează activările neuronilor din stratul intermediar folosind următoarea

formulă:[8]

$$\text{net}_{pj}^h = \sum_{i=1}^n W_{ji}^h \cdot x_{pi}, j = 1, \dots, L \quad (1.8)$$

Pasul 3. Se calculează ieșirile neuronilor din stratul intermediar folosind următoarea formulă:[8]

$$i_{pj} = f_j^h(\text{net}_{pj}^h), j = 1, \dots, L \quad (1.9)$$

Pasul 4. Se calculează activările neuronilor din stratul de ieșire pe baza relației:[8]

$$\text{net}_{pk}^o = \sum_{j=1}^L W_{kj}^o \cdot i_{pj}, k = 1, \dots, M \quad (1.10)$$

Pasul 5. Se calculează ieșirile neuronilor din stratul de ieșire conform următoarei formule:[8]

$$o_{pk} = f_k^o(\text{net}_{pk}^o), k = 1, \dots, M \quad (1.11)$$

Pasul 6. Se calculează erorile pentru neuronii de ieșire pe baza relației:[8]

$$\delta_{pk}^o = (y_{pk} - o_{pk}) \cdot f_k^o(\text{net}_{pk}^o), k = 1, \dots, M \quad (1.12)$$

Pasul 7. Se calculează erorile pentru neuronii din stratul intermediar folosind următoarea formulă:[8]

$$\delta_{pj}^h = f_j^h(\text{net}_{pj}^h) \cdot \sum_{k=1}^M \delta_{pk}^o \cdot W_{kj}^o, j = 1, \dots, L \quad (1.13)$$

Pasul 8. Se rafinează ponderile aferente stratului de ieșire folosind următoarea relație:[8]

$$\frac{\partial E_p}{\partial W_{kj}^o} = -(y_{pk} - o_{pk}) \cdot f_k^o(\text{net}_{pk}^o) \cdot i_{pj} = \delta_{pk}^o \cdot i_{pj} \quad (1.14)$$

Pasul 9. Se rafinează ponderile aferente stratului intermediar pe baza relației:[8]

$$W_{ji}^h(t+1) = W_{ji}^h(t) + \eta \cdot \delta_{pj}^h \cdot x_{pi}, j = 1, \dots, L, i = 1, \dots, n \quad (1.15)$$

Pasul 10. Se calculează eroarea datorată vectorului p de instruire cu ajutorul formulei:[8]

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2 \quad (1.16)$$

Pasul 11. Dacă nu s-a terminat de parcurs întregul set de antrenare, se va introduce un nou vector la intrarea rețelei, după care se va calcula eroarea corespunzătoare epocii curente (prin noțiunea de epocă, se înțelege parcurgerea și finalizarea lotului de antrenare) cu ajutorul următoarei formule:[8]

$$E = \frac{1}{N} \sum_{p=1}^N E_p \quad (1.17)$$

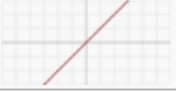



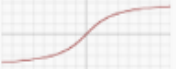

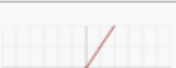


Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figura 1.13: Funcții de activare[9]

1.3.4 Funcții de activare

După cum am discutat în acest capitol, putem concluda asupra faptului că un neuron realizează următoarele task-uri: preia valorile de la intrare, calculează o sumă ponderată ale acestora, adună valoarea provenită de la un semnal extern numit *bias* și într-un final după prelucrarea acestor date, obține o valoare numită potențial de activare. În final, pentru a decide dacă neuronul se va activa și va genera un răspuns este nevoie de folosirea unei anumite funcții de activare aplicată potențialului de activare.

În continuare, pentru o mai bună înțelegere a acestui procedeu, vom nota potențialul de activare astfel:

$$v = \text{bias} + \sum_{i=1}^n w_i x_i \quad (1.18)$$

Acestui potențial îi va fi aplicată o funcție de activare, în urma căreia va fi determinată ieșirea neuronului:

$$y = f(v) \quad (1.19)$$

În figura 1.13, pot fi observate cele mai folosite funcții de activare și derivatele acestora. Tot mai mulți dezvoltatori de rețele neurale aleg să folosească în arhitecturile lor funcții de activare neliniare datorită felului în care acestea ajută rețeaua să prelucreze date complexe, să calculeze

și să învețe lucruri complicate oferind totodată și o predicții precise. Alegerea și folosirea unei anumite funcții depinde atât de task-ul pentru care a fost proiectată rețeaua cât și de setul de antrenare pe care îl folosim. Funcția Sigmoid a fost și este utilizată de foarte mult timp, deoarece prezintă avantaje cum ar fi: gradient neted care ajută la prevenirea "salturilor" în ceea ce privește valoarea de la ieșire și normalizarea ieșirii fiecărui neuron datorită variației valorilor de la ieșire între 0 și 1. Această funcție prezintă totuși și dezavantaje cum ar fi instabilitatea în jurul lui 0 și faptul că este costisitoare din punct de vedere computațional.

În ultimul timp, funcția care s-a bucurat de popularitate este **ReLU**, datorită faptului că este foarte eficientă din punct de vedere computațional (îi permite rețelei să converge foarte rapid) și a faptului că deși pare o funcție liniară, aceasta permite folosirea algoritmilor de tip *back-propagation* datorită derivatei sale. Dezavantajul principal al acestei funcții este că atunci când valorile de la intrare se apropie de 0 sau sunt negative, gradientul funcției devine nul, rețeaua nemaiputând practic să realizeze procedeul de *back-propagation* și să învețe. O variantă îmbunătățită a acestei funcții, care rezolvă această problemă, este **Leaky ReLU**.

O altă funcție de activare foarte eficientă și populară este funcția **Softmax**. Această funcție poate lucra cu mai multe clase, normalizând output-ul pentru fiecare clasă între 0 și 1 și împărțindu-l la suma claselor. Este de obicei folosită doar pentru stratul de ieșire, deoarece oferă un vector cu probabilitățile de apartenență la fiecare clasă, clasificând practic input-ul în mai multe categorii.

1.3.5 Funcția de cost

Optimizarea rețelelor neurale este un aspect fundamental atunci când dorim să proiectăm și să antrenăm o anumită arhitectură. Rețelele neurale sunt antrenate folosind tehnici bazate pe algoritmi de tip **gradient descent** și de cele mai multe ori, atunci când proiectăm și configurăm o anumită rețea pentru efectuarea unui task, este nevoie de folosirea unei funcții de cost adecvate. Funcția de cost este folosită pentru a estima performanțele rețelei, cu alte cuvinte, pentru a oferi informații despre cât de bine reușește rețeaua respectivă să aproximeze relația dintre două variabile. În esență, funcția de cost reprezintă diferența dintre valoarea obținută la ieșirea rețelei și valoarea dorită (ideală). Dacă notăm valoarea dorită la ieșire cu o și valoarea obținută la ieșirea rețelei cu y , putem defini generic funcția de cost astfel[41]:

$$E(y, o) = \frac{1}{N} \sum_{i=1}^N \frac{1}{2} (y_i - o_i)^2 \quad (1.20)$$

Pentru a înțelege mai bine cum poate fi optimizată o rețea neurală, trebuie să precizăm faptul că nu putem calcula valorile ideale ale ponderilor deoarece majoritatea rețelelor, în special cele cu învățare profundă (Deep Learning Neural Networks), au foarte multe straturi ascunse, fapt ce conduce practic la imposibilitatea realizării acestui lucru. Există totuși posibilitatea de a rezolva această problemă prin folosirea unor algoritmi ce "navighează" prin rețea și pot schimba aceste ponderi, optimizând rețeaua și obținând în acest fel predicții mai bune.

De obicei, o rețea neurală este antrenată folosind tehnici de optimizare bazate pe algoritmul gradientului negativ, ponderile fiind actualizate folosind un algoritm de tip *back-propagation*. [41] Gradientul negativ este un algoritm eficient de optimizare prin care se poate găsi minimul local sau global al unei funcții. Acest algoritm ajută rețeaua să învețe *gradientul* sau direcția în care ar trebui să se îndrepte pentru a reduce erorile. În acest fel, anumiți parametri vor fi optimizați sau corectați pentru a reduce și mai mult funcția de cost. La fiecare iterație, modelul converge progresiv către minim, unde schimbările sau optimizările parametrilor acestuia produc schimbări foarte mici ale funcției de cost.[10] În figura 1.14, se poate observa fiecare iterație a acestui algoritm, până se ajunge la convergență.

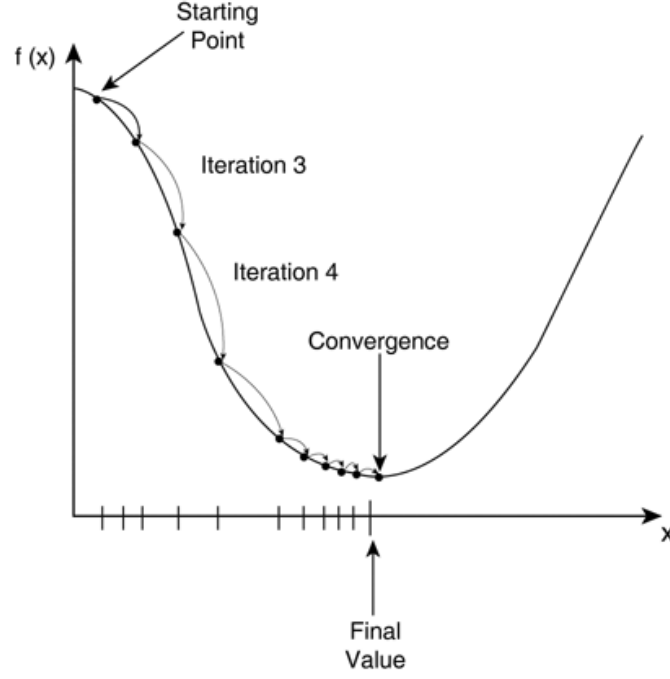


Figura 1.14: Gradientul negativ (Gradient Descent)[10]

Ponderile modelului sunt modificate după fiecare iterație folosind regula Delta enunțată de Widrow-Hoff. Această regulă implică calcularea derivatei în funcție de ponderi și ajustarea acestora în sensul opus al gradientului[42]:

$$\frac{\partial E}{\partial w_{jl}} = \frac{\partial E}{\partial y_i} \times \frac{\partial y_i}{\partial w_{ji}} \quad (1.21)$$

În final, ponderile vor fi ajustate cu următoarea valoare, calculată astfel:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \quad (1.22)$$

unde η reprezintă rata de învățare.

După ce se ajunge la convergență, modelul va fi reușit să-și optimizeze ponderile în așa fel încât să minimizeze funcția de cost. Acest algoritm este esențial pentru procesul de învățare al oricărui model deoarece îi asigură feedback-ul necesar îmbunătățirii bazat pe experiențele anterioare. O alternativă pentru tehnica gradientului negativ ar fi folosirea unei soluții de tipul **brute-force**, în sensul efectuării unei eventuale combinații infinite a parametrilor până când sunt observate îmbunătățiri, lucru care este din motive evidente mult mai ineficient.[10]

1.4 Rețele neurale convoluționale

Domeniul inteligenței artificiale a cunoscut o creștere impresionantă în ultimii ani, ajutând semnificativ la dezvoltarea relației om-mașină. O dată cu această evoluție a rețelelor neurale, au crescut semnificativ și ariile de aplicabilitate ale inteligenței artificiale iar aceste sisteme inteligente sunt astăzi prezente în majoritatea domeniilor cheie. Cercetători, ingineri și oameni de știință lucrează în permanență pentru implementarea a numeroase astfel de rețele neurale care uimesc prin modul în care vin ca soluție la o varietate de probleme. Unul dintre domeniile care s-au dezvoltat extrem de mult datorită inteligenței artificiale este cel al viziunii computerizate (*computer vision*).[11]

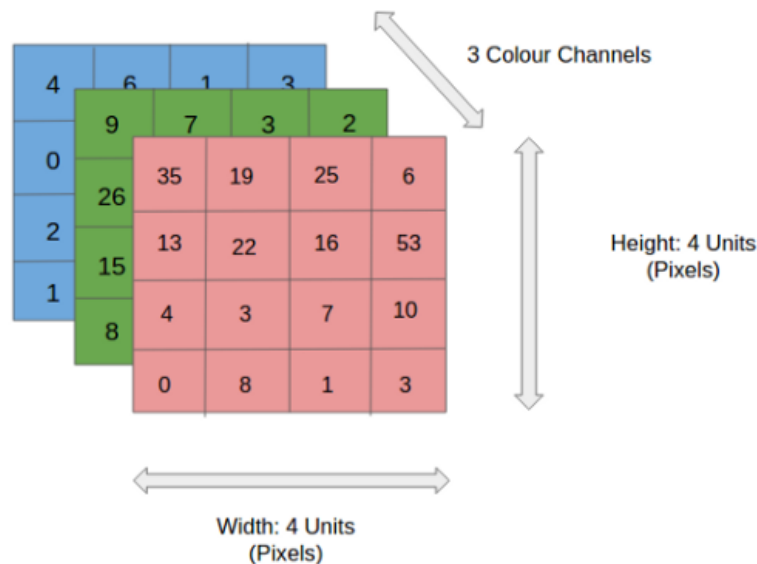


Figura 1.15: Exemplu de imagine RGB de dimensiune $4 \times 4 \times 3$ [11]

Scopul acestui domeniu este să reușească să facă computerele să vadă și să proceseze informațiile la fel cum o facem noi oamenii, să perceapă lumea într-o manieră similară ființei umane și să folosească cunoștințele dobândite pentru a fi capabile să efectueze diferite task-uri cum ar fi: recunoașterea de imagini, clasificarea de imagini și chiar crearea unor sisteme de recreere și recomandare. Progresele în domeniul viziunii computerizate au fost făcute în timp și sunt datorate în mare parte rețelelor neurale convoluționale. [11]

O rețea neurală convoluțională (CNN) este o arhitectură ce are la bază un algoritm de învățare profundă care arată performanțe superioare altor arhitecturi atunci când la intrarea rețelei se află o imagine. Principalul avantaj al acestui algoritm este faptul că se poate concentra pe anumite obiecte/aspecte din imagine, fiind capabil să identifice în mod autonom caracteristicile ce deosebesc o imagine de alta. Un alt avantaj al folosirii acestor rețele este faptul că necesită mult mai puțină pre-procesare în comparație cu alți algoritmi de clasificare. În timp ce majoritatea arhitecturilor folosite pentru clasificarea imaginilor înainte de apariția rețelelor neurale convoluționale foloseau filtre ce necesitau a fi modificate manual, o arhitectura CNN este capabilă să învețe singură cum să-și adapteze filtrele folosite pentru extragerea caracteristicilor.

Rețele neurale convoluționale (CNN) sunt folosite, după cum am precizat mai sus, în aplicațiile ce au la intrare imagini. Comparativ cu alte arhitecturi, neuronii de intrare ai arhitecturilor CNN sunt caracterizați de trei dimensiuni spațiale: înălțime, lungime și adâncime. În figura 1.15, se poate observa o imagine reprezentată RGB, pe 3 canale, fiecare canal constituind unul dintre cele 3 planuri de culoare (roșu, verde, albastru). Deoarece arhitecturile CNN diferă de arhitecturile clasice și din punctul de vedere al conectivității neuronilor, în sensul că neuronii dintr-un anumit strat nu sunt conectați în totalitate cu neuronii stratului precedent, acest lucru permite ca la ieșirea rețelei să avem un strat comprimat de dimensiune $1 \times 1 \times n$ (n fiind numărul de clase), folosind ca input o imagine de $128 \times 128 \times 3$. Este de remarcat faptul că prelucrarea unei imagini la rezoluție 8K (7680×4320) este foarte costisitoare din punct de vedere computațional, iar rețelele neurale convoluționale au rolul de a reduce semnificativ dimensiunea imaginilor până la o formă în care sunt mult mai ușor de procesat, păstrând totodată și caracteristicile distinctive care ajută la clasificarea finală a imaginilor. [11]

Din punct de vedere structural, rețelele neurale convoluționale (CNN) sunt alcătuite din următoarele trei tipuri de straturi (layer): stratul convoluțional, stratul de pooling și stratul complet conectat (fully-connected). Combinația acestor trei tipuri de layer conduce la realizarea unei arhitecturi de tipul CNN.

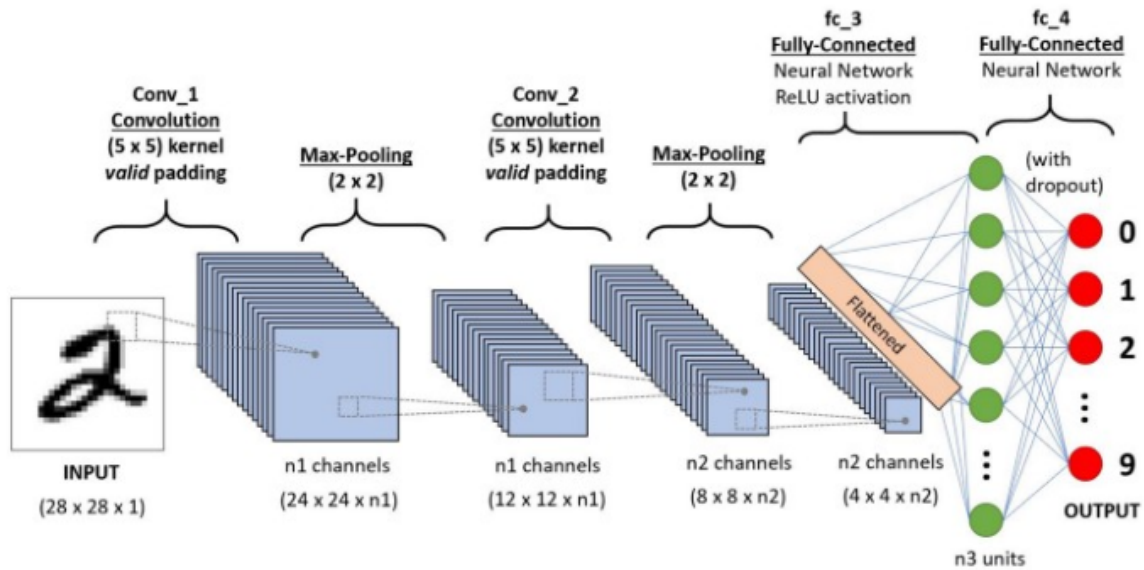


Figura 1.16: Arhitectură CNN folosită pentru clasificarea cifrelor scrise de mână[11]

În figura 1.16, este ilustrată o arhitectura de tipul CNN și secvența în care sunt folosite cele trei straturi enumerate mai sus. Această rețea a fost proiectată pentru recunoașterea cifrelor scrise de mână, un exemplu clasic de înțelegere a modului în care funcționează rețelele neurale convoluționale. Această arhitectură are ca input o imagine pe un singur canal (grayscale) de dimensiune 28x28x1 iar ca output conține layer-ul de top de dimensiune 1x1x10, care este practic clasificatorul.

Procesul de clasificare a imaginii de intrare este împărțit în mai multe etape, fiecare strat jucând un rol foarte important, după cum urmează[12]:

- **Stratul de intrare:** Acest strat conține valorile pixelilor din imaginile ce urmează a fi clasificate.
- **Stratul de convoluție:** Acest strat este responsabil cu realizarea produsului scalar dintre valorile din imaginile de la intrare și anumite ponderi (reprezentate prin filtre). În acest strat se calculează practic ieșirile neuronilor din stratul convoluțional care sunt conectați la intrare. Înainte de a returna un rezultat, potențialul de activare al neuronilor din acest strat este supus funcției de activare ReLU, care va decide care dintre acești neuroni se vor activa.
- **Stratul de pooling:** Acest strat este responsabil de eșantionarea spațiului de intrare, reducând astfel substanțial dimensionalitatea și numărul de parametri.
- **Stratul complet conectat (fully-connected):** Acest strat conține neuroni conectați la doua straturi adiacente, asigurând practic legătura între stratul comprimat (flatten layer) și stratul *softmax* ce conține gradele de apartenență la fiecare clasă ale input-ului.

Toate acestea straturi, conectate în totalitate sau parțial, reușesc să facă rețeaua să înțeleagă caracteristicile distinctive ale fiecărei imagini. Modul în care funcționează fiecare dintre aceste straturi ce alcătuiesc o arhitectură CNN, precum și relația dintre acestea, vor fi discutate în detaliu în subcapitolele următoare.

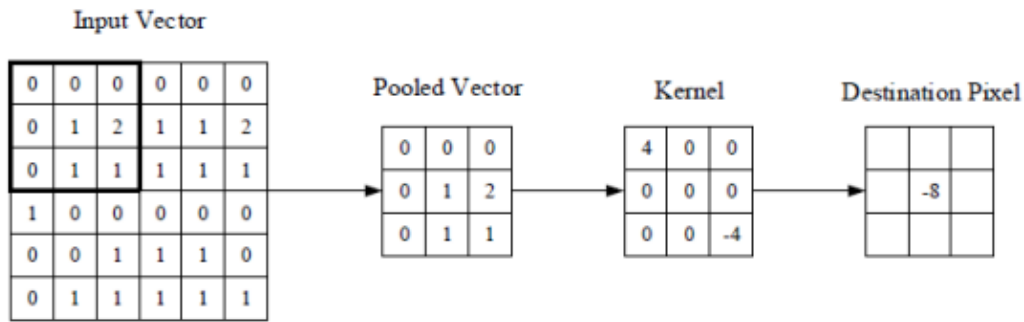


Figura 1.17: Reprezentarea unui strat convoluțional[12]

1.4.1 Stratul de convoluție

Acest strat este definitiv pentru arhitectura CNN, deoarece joacă un rol extrem de important în extragerea caracteristicilor unei anumite imagini. Parametrii acestui strat se focusează pe utilizarea anumitor filtre care au proprietatea că pot învăța să se adapteze în funcție de cum evoluează procesul de instruire al rețelei, în sensul că aceștia își pot schimba ponderile. Aceste filtre au de obicei o dimensiune mică, însă acestea se extind pe întreaga adâncime a intrării. Atunci când datele de la intrare ajung la un strat convoluțional, acesta efectuează operația de convoluție între fiecare filtru și toată suprafața intrării, producând astfel mai mult hărți de caracteristici (*feature maps*) ale imaginii.[12] În figura 1.17, se observă cum elementul din centrul filtrului este plasat pe un anumit pixel din imaginea de intrare, iar valoarea acestuia va fi înlocuită cu rezultatul sumei ponderate dintre el și vecinii săi. După ce s-a parcurs toată suprafața de intrare cu filtrele corespunzătoare unui anumit strat convoluțional, rețeaua va învăța care sunt filtrele care se activează atunci când vad o caracteristică specifică la o anumită poziție a spațiului de intrare. Fiecărui filtru îi va corespunde o hartă de activare, hărți ce vor forma întregul volum de ieșire al stratului convoluțional.

Aceste straturi de convoluție au rolul de a reduce semnificativ complexitatea modelului prin felul în care optimizează ieșirea acestuia. Optimizarea constă în modificarea următorilor parametri: adâncimea, pasul și setarea "zero-padding".

Adâncimea volumului de ieșire produs de straturile convoluționale poate fi setată prin modificarea numărului de neuron din regiunea respectivă. Acest lucru poate fi privit și din perspectiva altor rețele neurale, unde toți neuronii din straturile ascunse sunt conectați direct cu toți ceilalți dinaintea lor. Numărul total al neuronilor rețelei poate fi minimizat prin reducerea acestui hiperparametru, însă există totodată și posibilitatea scăderii performanței de recunoaștere a modelului. [12]

Există totodată și posibilitatea setării pasului cu care este parcurs spațiul de intrare. Spre exemplu, dacă vom seta o valoare a pasului egală cu 1, vom avea un câmp receptiv puternic suprapus care va produce activări extrem de mari. În schimb, dacă vom seta o valoare mai mare a pasului, acest lucru va reduce semnificativ cantitatea de suprapuneri și va avea ca efect reducerea dimensionalității la ieșire.[12] Câmpul receptiv reprezintă dimensiunea dintre intrare și stratul convoluțional.

Setarea "zero-padding" reprezintă procesul prin care se adaugă o bordură vectorului de intrare, fiind o metodă eficientă de a controla dimensionalitatea volumului de ieșire.

Cu toate eforturile care s-au depus de-a lungul timpului pentru reducerea numărului de parametri, se pare că majoritatea modelelor ce folosesc o arhitectură CNN sunt încă extrem de complexe atunci când se folosește ca input o imagine de dimensiune mai mare. Totuși, există câteva metode prin care putem reduce semnificativ numărul de parametri din stratul

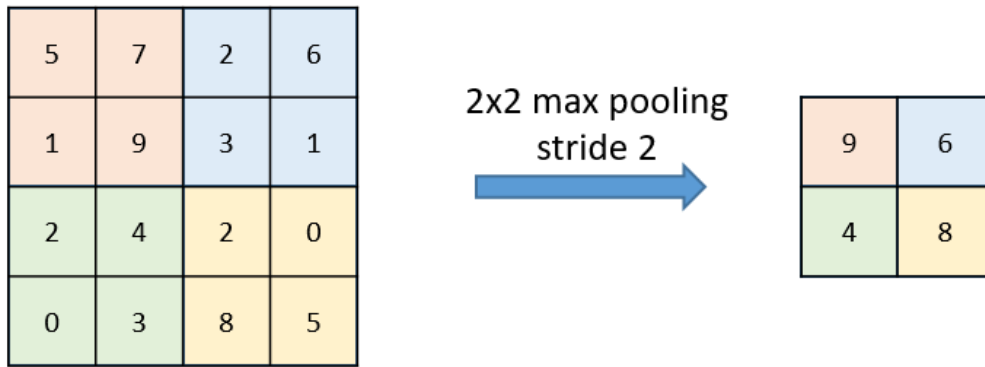


Figura 1.18: Aplicarea operației de max-pooling

convoluțional, însă trebuie să precizăm faptul că mereu va trebui să existe un compromis între performanțe și complexitatea modelelor.

1.4.2 Straturile de pooling

Aceste straturi sunt de obicei introduse între straturi de convoluție succesive. Rolul lor este de a reduce progresiv dimensiunea spațială reprezentată de prelucrările imaginii de la intrare, reducând totodată și numărul de parametri. Acest lucru conduce la reducerea puterii de calcul și la un control mai bun al *overfitting*-ului. Cea mai întâlnită și utilizată formă a acestui filtru este cea de *max-pooling* de dimensiune 2×2 , cu pasul 2, reprezentată și în figura 1.18.

Acest strat glisează pe fiecare hartă de activare a vectorului de intrare, având rolul de a reduce dimensiunea acestora folosind funcția **MAX**. După cum am precizat și mai sus, majoritatea arhitecturilor CNN folosesc acest filtru în forma reprezentată în figura 1.18, deoarece dimensiunea acestuia permite păstrarea caracteristicilor distinctive ale fiecărei imagini, iar pasul previne suprapunerile ce conduc la activări mari. Procedura este destul de simplă, o dată ce nucleul filtrului a fost suprapus peste o zonă din suprafața de intrare, acesta înlocuiește cei 4 pixeli cu un singur pixel având valoarea maximă, reducând astfel dimensionalitatea cu 25% și păstrând totodată adâncimea și caracteristicile distinctive.

1.4.3 Straturile fully-connected

Straturile fully-connected, sunt după cum o sugerează și numele, straturi complet conectate ale unei arhitecturi CNN. Acestea sunt conțin neuroni care sunt conectați în totalitate atât cu neuronii din stratul precedent (flatten layer) cât și cu neuronii stratului de ieșire care conține gradele de apartenență la fiecare clasă.

1.4.4 Arhitecturi CNN

De-alungul timpului, au fost dezvoltate mai multe arhitecturi CNN care s-au remarcat prin performanțe de recunoaștere superioare arhitecturilor existente până atunci. Primele dintre acestea erau destul de complexe, fiind foarte costisitoare din punct de vedere computațional și necesitând totodată și timpi de antrenare foarte mari. Următoarele arhitecturi CNN s-au remarcat prin faptul că aveau un număr de parametri mai mic și o complexitate mai scăzută, menținând însă o acuratețe remarcabilă a predicțiilor. Cele mai populare dintre aceste arhitecturi sunt[43]:

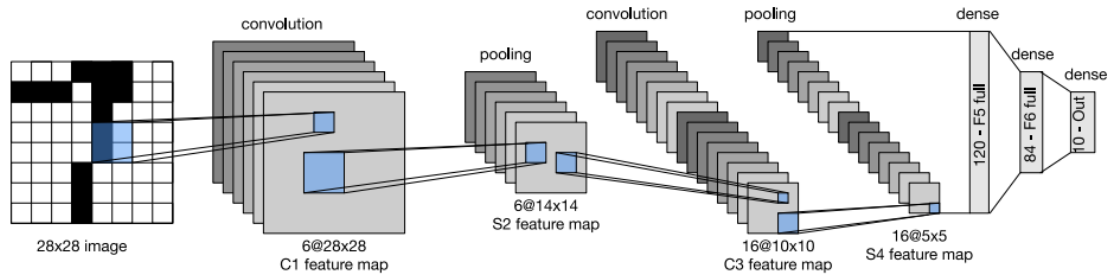


Figura 1.19: Arhitectura LeNet[13]

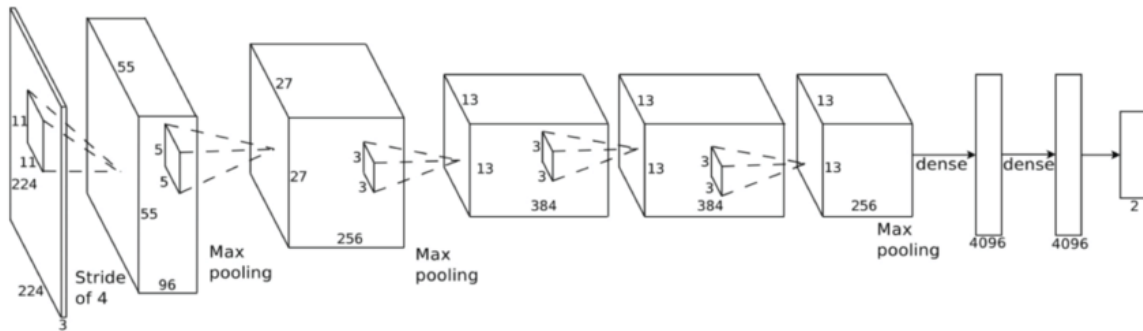


Figura 1.20: Arhitectura AlexNet[14]

- **LeNet:** Această arhitectură a fost dezvoltată de Yann LeCun. Acesta a dezvoltat mai multe aplicații ce foloseau rețele neurale convoluționale de-a lungul anilor 1990, însă arhitectura LeNet este cea mai cunoscută, fiind folosită cu succes la citirea codurilor poștale și la recunoașterea cifrelor. În figura 1.19, este ilustrată această arhitectură.
- **AlexNet:** Această arhitectură a fost dezvoltată de Alex Krizhevsky, Ilya Sutskever și Geoff Hinton, fiind cea care a adus popularitate rețelelor neurale convoluționale și domeniului viziunii computerizate. Această arhitectură seamănă destul de mult cu LeNet însă este mult mai mare, mai complexă și conține mai multe straturi convoluționale suprapuse, în timp ce arhitectura LeNet are doar un strat convoluțional urmat de un strat de pooling. În figura 1.20, poate fi observată această arhitectură.
- **ZF Net:** Această arhitectură a fost dezvoltată de Matthew Zeiler și Rob Fergus, primind acest nume de la inițialele autorilor. Este o variantă îmbunătățită a arhitecturii AlexNet deoarece au fost făcute diferite optimizări ale hiperparametrilor. Dimensiunea straturilor convoluționale de mijloc ale rețelei a fost mărită iar pasul și dimensiunea filtrelor din primul strat au fost micșorate. În figura 1.21, este ilustrată arhitectura acestei rețele neurale convoluționale.
- **GoogLeNet:** Această rețea a fost dezvoltată de Szegedy împreună cu alți oameni de știință de la Google. Principala îmbunătățire adusă de această rețea a fost dezvoltarea unui așa-zis *Inception Module*, care a permis reducerea semnificativă a numărului de parametri (4 milioane, față de 60 de milioane de la AlexNet). Această arhitectură a înlocuit și straturile fully-connected de la ieșirea rețelei cu straturi de **average pooling**, fapt ce a condus la eliminarea unor parametri care aparent nu contau atât de mult. Există în prezent mai multe variante îmbunătățite ale acestei arhitecturi, cea mai recentă dintre ele fiind *Inception-v5*. În figura 1.22, poate fi observată arhitectura acestei rețele.

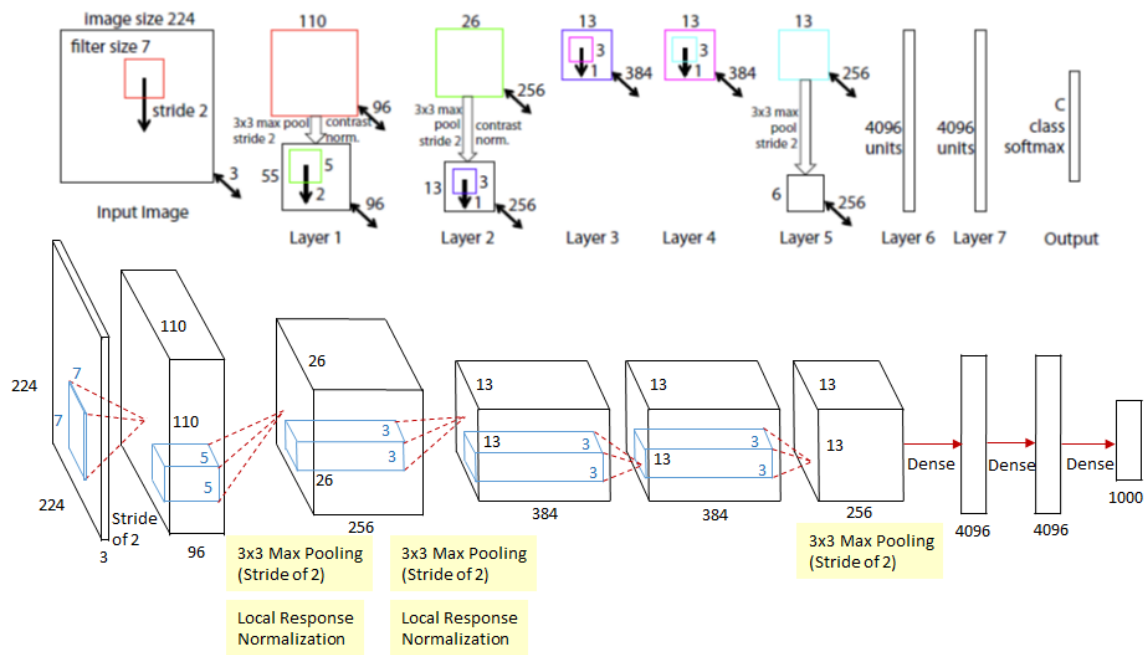


Figura 1.21: Arhitectura ZF Net[15]

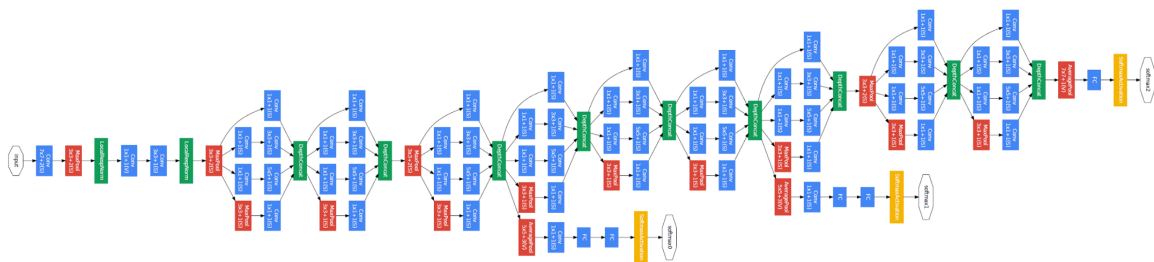


Figura 1.22: Arhitectura GoogLeNet[16]

Straturile convoluționale sunt reprezentate de culoarea albastru, cu roșu straturile de max-pooling și cu verde cele de normalizare.

- **VGGNet:** Această arhitectură a fost dezvoltată de Karen Simonyan și Andrew Zisserman. Principala remarcă făcută după apariția acestei arhitecturi a fost că adâncimea rețelei este un hiperparametru critic pentru performanța acesteia. Modelul final propus de autorii acestei rețele 16 straturi convoluționale complet conectate ceea ce a condus la realizarea unei arhitecturi omogene. Rețeaua realizează convoluții de dimensiune 3×3 și pooling de dimensiune 2×2 . Această rețea este însă destul de costisitoare din punct de vedere computațional deoarece folosește extrem de multă memorie. Numărul de parametri al acestei rețele este de aproxativ 140M, majoritatea aflându-se în primul strat complet conectat. În figura 1.23, este ilustrată arhitectura acestei rețele.
- **ResNet:** Această rețea a fost dezvoltată de Kaiming He și alți cercetători. Aceasta este caracterizată de anumite conexiuni speciale și de folosirea intensivă a normalizării pe batch. Această arhitectură nu conține straturi fully-connected și este printre cele mai bune rețele neurale convoluționale existente. De când a apărut, alți cercetători și entuziaști în domeniu au realizat mai multe modele bazate pe această arhitectură. Arhitectura vine în variante cu 18, 34, 50, 101 și 152 straturi, cu ponderi cuprinse între 46 și 236 MB.

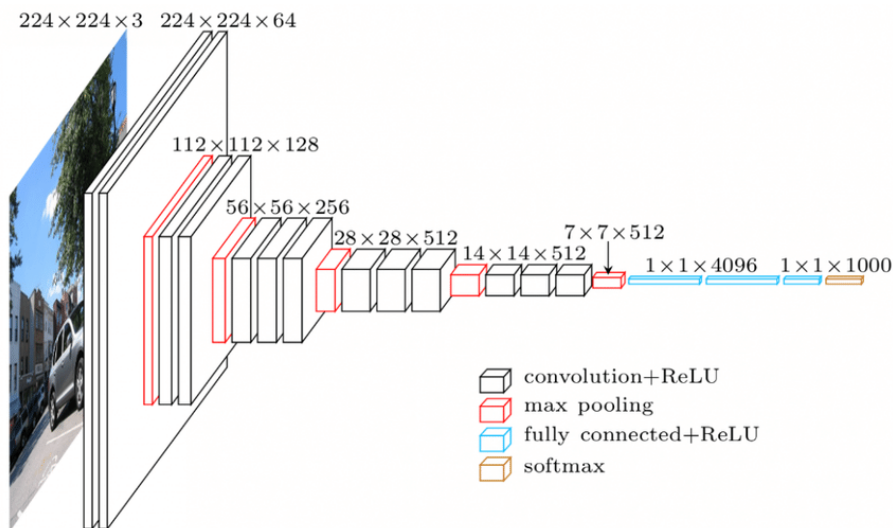


Figura 1.23: Arhitectura VGGNet[17]

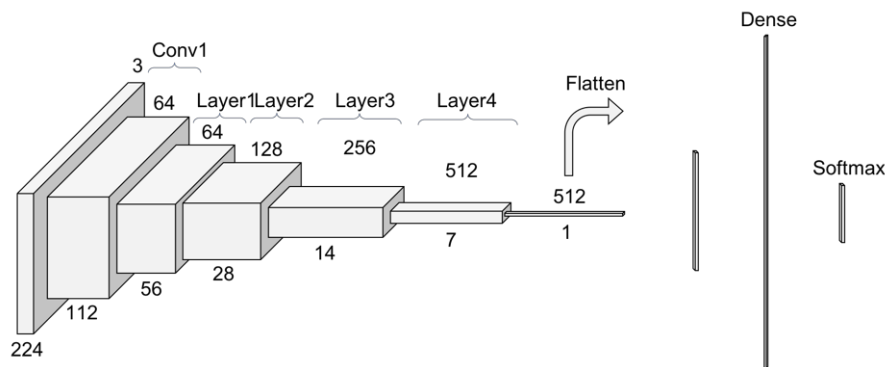


Figura 1.24: Arhitectura ResNet-34[18]

1.4.5 Transfer Learning

De obicei, proiectarea și antrenarea unei rețele de la zero, este un procedeu destul de complicat care durează foarte mult timp. Din fericire, există o tehnică numită **transfer-learning** (transfer de cunoștințe) prin care putem folosi o arhitectură deja existentă, precum cele enumerate în subcapitolul anterior, pentru a crea o nouă rețea capabilă să realizeze alte task-uri. Asemănător omului, care învață lucruri de bază și mai apoi își folosește aceste cunoștințe pentru a realiza alte lucruri mai complexe, cunoștințele rețelelor neurale pre-antrenate pot fi transferate unei noi arhitecturi în scopul realizării unui nou task. Acest lucru funcționează datorită faptului că straturile de bază ale unei rețele antrenate *from-scratch* sunt aparent niște extractoare de caracteristici generale. Pentru realizarea procedurii transferului de cunoștințe, trebuie să luăm o arhitectură deja existentă și să îi înlocuim stratul de ieșire cu nou layer cu ponderi specifice noului task ce se dorește a fi rezolvat.[44]

Aceste rețele antrenate de la zero (*from-scratch*), sunt numite rețele *pre-antrenate* în contextul procedurii transferului de cunoștințe. Pentru a exploata cunoștințele dobândite de o anumită arhitectură cu scopul de a crea o nouă rețea capabilă să rezolve noi task-uri, există două modalități:

- **Fixed Feature Extraction:** Această metodă presupune schimbarea propriu-zisă a stratului de ieșire al rețelei pre-antrenate și înlocuirea acestuia cu un nou strat care va fi re-antrenat folosind imagini dintr-o nouă bază de date. Această procedură se numește

extragere de caracteristici fixă deoarece restul rețelei rămâne neschimbată. Avantajul folosirii acestei metode este faptul că majoritatea arhitecturilor enumerate în subcapitolul anterior au fost antrenate folosind baze de date cu milioane de imagini, fapt ce le-a permis să învețe să-și adapteze straturile inferioare să extragă cele mai bune caracteristici dintr-o imagine. Prin acest procedeu se reduce substanțial timpul de antrenare iar rezultatele sunt remarcabile.

- **Fine-Tuning:** Această metodă este puțin mai complexă și implică ajustarea întregii rețele, pe lângă re-antrenarea ponderilor stratului de ieșire. Procedura durează de obicei mai mult decât cea a extragerii de caracteristici fixă, dar aceasta poate conduce la creșterea acurateții de predicție a modelului. Ponderile rețelei pre-antrenate sunt de obicei cele dobândite în urma antrenării rețelei pe baza de date **ImageNet**. Acest reglaj fin al rețelei implică deblocarea și re-antrenarea straturilor convoluționale și a celor de pooling.

Un aspect extrem de important care trebuie studiat înainte de a aplica procedura transferului de cunoștințe este arhitectura rețelei. De obicei, primele straturi ale unei rețele detectează pattern-uri generale și mai puțin complicate, și în consecință, acestea ar trebui să fie ”înghețate” și folosite cu ponderile originale. Pe măsura ce avansăm în rețea, lucrurile devin mai complicate, iar ponderile tind să se ajusteze pentru a detecta pattern-uri mai complicate, specifice bazei de date folosite pentru noul task.

Așadar, putem aplica procedura transferului de cunoștințe folosind metoda **fixed feature extraction**, iar în funcție de rezultatele pe care le obținem, putem decide dacă vom aplica și metoda **fine-tuning**, caz în care putem să deblocăm și să re-antrenăm și următoarele straturi, începând cu ultimul bloc convoluțional și straturile de pooling.

Capitolul 2

Mediul de dezvoltare

Dezvoltarea unui sistem de recunoaștere a emoțiilor poate părea simplă, însă în practică este destul de complicat să proiectezi o rețea neurală convoluțională capabilă să rezolve acest task. Pentru a atinge acest obiectiv, partea practică va fi alcătuită din mai multe module individuale, fiecare dintre acestea fiind responsabile cu realizarea unor task-uri secundare. Implementarea acestor module va fi detaliată în capitolul următor, însă înainte de acest lucru, vom discuta despre tehnologiile și resursele software necesare realizării acestui proiect. Sistemul inteligent de recunoașterea a emoțiilor a fost scris în totalitate în limbajul de programare Python și folosește diferite librării pentru implementarea anumitor funcții.

Pentru antrenarea modelului, am folosit baza de date Fer2013 datorită dimensiunii imaginilor din aceasta precum și a numărului lor mare. Dat fiind faptul că antrenarea rețelelor neurale convoluționale este foarte costisitoare din punct de vedere computațional, am avut nevoie de un mediu de antrenare cu putere de calcul foarte mare. O soluție la această problemă ar fi folosirea platformei **Google Colab**, care pune la dispoziție dezvoltatorilor de rețele neurale servere cu putere de calcul foarte mare, în mod gratuit, însă cu câteva constrângeri în privința timpilor de antrenare. Deoarece am dorit să dezvolt un sistem de recunoaștere a emoțiilor care să funcționeze în timp real, iar alternativa pentru acest lucru era ca după proiectarea și antrenarea modelului, acesta să fie deployat pe un Cloud și apelat printr-un API, am ales să dezvolt, să antrenez și să folosesc modelul pentru predicții pe computerul personal. Resursele hardware de care am dispus au fost următoarele: procesor AMD Ryzen 7 3750H, placa video NVIDIA GeForce GTX 1660Ti 6GB și memorie ram 8GB. Este de precizat faptul că în antrenarea rețelelor neurale convoluționale, cea mai solicitată și importantă componentă este placa video, deoarece aceasta poate realiza foarte rapid operațiile de convoluție și de prelucrare a imaginilor. Pentru ca antrenarea rețelei pe placa video să poată fi realizată, este nevoie de instalarea unor pachete software specifice plăcii video și rețelelor neurale convoluționale.

În continuare, vom face o trecere în revistă atât a limbajului de programare Python, cât și a tehnologiilor și librăriilor pe baza cărora a fost implementat proiectul.

2.1 Python

Python este un limbaj de programare de nivel înalt folosit pentru dezvoltarea aplicațiilor software dintr-o varietate de domenii. Acesta a fost creat în anul 1991 de către Guido van Rossum, filosofia acestui limbaj fiind aceea de a dezvolta codul unei aplicații mult mai ușor, prin eliminarea caracterelor ce marcau sfârșitul unei linii de cod (de exemplu ";") și înlocuirea

acestora cu spații la începutul fiecărei linii. În acest fel, nu mai trebuie să ținem evidența parantezelor și a acoladelor, orice instrucțiune scrisă cu aceeași identare (același număr de spații raportat la începutul liniei de cod) aparținând practic aceluiaș bloc de instrucțiuni. Sintaxa și abordarea obiect-orientată ale acestui limbaj sunt foarte apreciate de programatori deoarece le permite acestor să dezvolte cod logic și clar pentru proiecte de orice dimensiune.[45]

Din punctul de vedere al compatibilității, Python funcționează pe toate platformele și este foarte ușor de instalat. Datorită librăriei standard, care este considerată una dintre cele mai bune librării ce vin la pachet cu un limbaj de programare de nivel înalt, precum și a simplității sintaxei care este caracterizată în principal de cuvinte din limba engleza, acest limbaj de programare este astăzi unul dintre cele mai folosite limbaje, fiind totodată și cel preferat de cei care lucrează în domeniul inteligenței artificiale.

Python este utilizat ca limbaj de programare chiar și pe platforma despre care am discutat anterior, Google Colab. Librăriile TensorFlow și Keras dezvoltate de cei de la Google, precum și alte librării folosite în domeniul inteligenței artificiale cum ar fi OpenCV sau Numpy, sunt cele care au făcut limbajul Python să devină astăzi cel mai folosit limbaj în domeniul învățării automate și al vederii computerizate.

Acest limbaj de programare este extrem de util în domeniul științei datorită scalabilității care permite atât dezvoltarea proiectelor mici cât și dezvoltarea proiectelor de anvergură. Librăriile care vin ca suport pentru acest limbaj conțin peste 200 000 de pachete, ale căror funcționalități pot fi folosite în realizarea de proiecte ce implică[46]:

- **Automatizări**
- **Analiza de date**
- **Baze de date**
- **Crearea interfețelor grafice**
- **Învățare automată**
- **Analiza și procesare de imagini**
- **Aplicații web**
- **Aplicații mobile**

Instalarea limbajului de programare Python poate fi realizată relativ ușor pe toate platformele consultând documentația¹. Dacă utilizați Windows sau MacOS, Python poate fi instalat folosind **Anaconda** respectiv **Homebrew**, iar dacă folosiți Linux, acesta poate fi instalat foarte ușor folosind următoarele comenzi în terminal:

```
1 $ sudo apt-get update
2 $ sudo apt-get install python3.6
```

În continuare, vom discuta despre librăriile folosite în dezvoltarea proiectului dar și despre alte resurse software ce necesită a fi instalate pentru a putea antrena modelul pe GPU.

¹<https://wiki.python.org/moin/BeginnersGuide/Download>

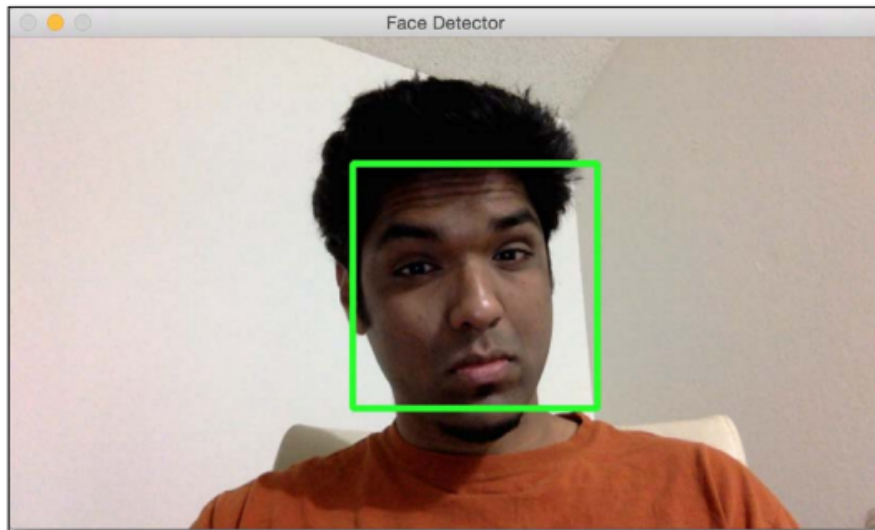


Figura 2.1: Detectarea feței cu ajutorul librăriei OpenCV[19]

2.2 OpenCV

OpenCV este o librărie open-source, ce poate fi folosită pe toate sistemele de operare pentru a dezvolta module software pentru aplicații și experimente din domeniul viziunii computerizate. Aceasta librărie dispune de toate uneltele software necesare pentru a implementa interfețe grafice și funcții de achiziție, procesare și afisare a imaginilor.[19]

Viziunea computerizată se bucură în ultimii ani de mare succes, fiind prezentă în aproape toate tehnologiile și sistemele software moderne. Librăria OpenCV pentru Python este folosită pentru a dezvolta și rula algoritmi de viziune computerizată foarte puternici, ce pot fi rulați în aplicații de procesare a imaginilor real-time. O dată cu dezvoltarea computerelor și cu creșterea puterii computaționale, progresele aduse de această librărie în domeniul viziunii computerizate s-au făcut simțite în majoritatea domeniilor, tot mai multe companii dezvoltatoare de produse software în această direcție alegând să dezvolte pe Cloud tehnologii implementate cu ajutorul acestei librării. [19]

În dezvoltarea proiectului de diplomă, OpenCV a fost folosit pentru realizarea mai multor module ale sistemului de recunoaștere a emoțiilor umane. Unul dintre cele mai importante module, este cel de achiziționare a semnalului video de la camera web a computerului pe care este rulată aplicația. Acest modul detectează automat fața subiectului din poza, fiind capabil să detecteze chiar și mai multe fețe într-un singur frame. Funcția de detectare a feței este implementată cu ajutorul acestei librării. În figura , este ilustrat un exemplu în care se detectează fața unui subiect și este încadrată într-un dreptunghi verde, aceste funcții fiind implementate în librăria OpenCV.[19]

Această librărie poate fi instalată foarte ușor o dată ce limbajul Python a fost instalat². Comanda (într-un environment Python) prin care se poate instala pachetul de bază al librării OpenCV este următoarea:

```
1 pip3 install opencv-python
```

În capitolul următor, vor fi descrise implementările tuturor modulelor software ce folosesc funcții din librăria OpenCV. Aceste funcții au rolul de pre-procesare a imaginilor, pentru a maximiza performanțele rețelei neurale asigurându-i acesteia imagini concise.

²<https://docs.opencv.org/>

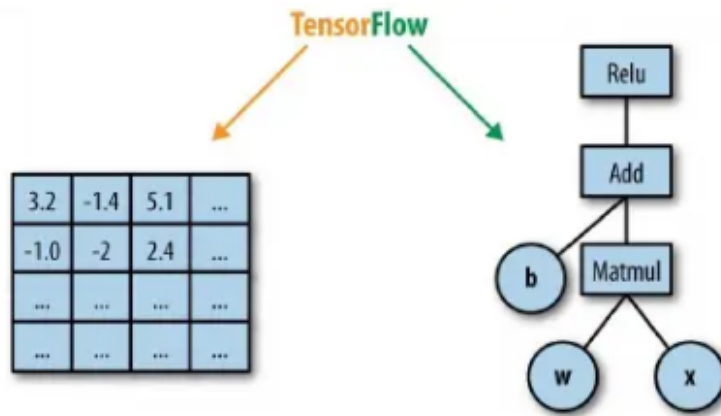


Figura 2.2: Grafic de calcul al fluxului de date. Datele de intrare sub formă de tensori sunt procesate de rețea[20]

2.3 TensorFlow

TensorFlow este o bibliotecă open-source dezvoltată de cei de la Google cu ajutorul căreia se pot implementa rețele neurale cu învățare profundă. În ultimii ani, rețele neurale cu învățare profundă au devenit cele mai folosite datorită rezultatelor obținute în diverse domenii. Aceste rețele sunt folosite pentru rezolvarea unor probleme foarte complexe, necesitând totodată foarte multe date pentru a învăța, dar și unelte software capabile să simplifice implementarea neuronilor și a algoritmilor folosiți de acestea. Având în vedere că în zilele noastre domeniul software se bucură de foarte multe tehnologii și framework-uri open-source, folosirea acestei biblioteci a devenit un lucru esențial pentru oricine este implicat în dezvoltarea sistemelor inteligente cu învățare automată.

Ideea de bază a acestei biblioteci este folosirea tensorilor. Tensorul este din punct de vedere matematic, un obiect geometric cu ajutorul căruia se pot asocia multiliniar vectori, scalari și alți tensori cu un tensor rezultat. Vectorii și scalarii reprezintă cei mai simpli tensori, fiind folosiți în fizică, matematică și în majoritatea aplicațiilor ingineresti. Practic, tensorii sunt hărți multilinare ce asociază spații vectoriale cu numere reale. [47]

Pentru o mai bună înțelegere a faptului că scalarii, vectorii și matricele sunt forme simple ale unui tensor, vom arăta funcțiile de asociere ale spațiilor vectoriale de definiție cu imaginile acestora:

- **Scalar:** $f : \mathbb{R} \rightarrow \mathbb{R}, f(e_1) = c$
- **Vector:** $f : \mathbb{R}^n \rightarrow \mathbb{R}, f(e_i) = v_i$
- **Matrice:** $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}, f(e_i, e_j) = A_{ij}$

În figura 2.2, este ilustrat un grafic de calcul ce definește generic felul în care această bibliotecă lucrează cu datele într-o rețea neurală. TensorFlow este în principal gândită ca o bibliotecă care interfațează implementarea algoritmilor de învățare profundă guvernați de arhitecturi de tipul **deep learning**.

TensorFlow este o bibliotecă portabilă, făcând astfel ca acești algoritmi și aceste rețele neurale să poată fi implementate în majoritatea mediilor de dezvoltare, indiferent de ce sistem de operare sau platforma hardware folosim. Există chiar și posibilitatea de a antrena pe Cloud o rețea dezvoltată cu ajutorul acestui framework, modelul rezultat putând fi folosit distribuit pe un cluster alcătuit din mai multe sisteme și computere. Modelele pot fi deployate pentru a face predicții atât pe platforme mobile, ce rulează folosind sisteme iOS și Android, cât și pe

Operator TensorFlow	Prescurtare	Descriere
<code>tf.add()</code>	$a + b$	Adaugă a și b, element cu element
<code>tf.multiply()</code>	$a * b$	Înmulțește a și b, element cu element
<code>tf.subtract()</code>	a / b	Împarte a și b, în stilul Python-ului
<code>tf.divide()</code>	$a - b$	Scade a și b, element cu element
<code>tf.mod()</code>	$a \% b$	Calculează modulo, element cu element
<code>tf.greater()</code>	$a > b$	Returnează tabelul de adevăr, element cu element
<code>tf.negative()</code>	$\sim a$	Returnează valoarea negativă a fiecărui element din a

Tabela 2.1: Cele mai comune operații făcute cu ajutorul TensorFlow și prescurtările acestora[20]

sisteme de tipul Raspberry Pi. TensorFlow a fost dezvoltat în C++, însă framework-ul cel mai folosit de oamenii de știință care dezvoltă rețele neurale este cel din Python, deoarece cel din C++ este destul de limitat, fiind totodată suficient pentru cei care dezvoltă astfel de arhitecturi pentru sisteme embedded. [20]

Tensorii creați cu ajutorul acestui framework pot fi utilizați pentru a crea un așa-zis graf computațional, prin care se pot aplica operații acestora, rezultatul fiind tot un tensor obținut în urma aplicării tuturor operațiilor din care este alcătuit graful respectiv. Câteva dintre aceste operații care pot fi aplicate tensorilor, pot fi observate în tabelul 2.1.

Deși operațiile cu acești tensori sunt folositoare, în sensul că pot ajuta funcțiile mai avansate ale acestui framework să realizeze prelucrări la nivel de tensor, piesa de rezistență a nucleului librăriei TensorFlow este **Keras**. Librăria Keras este un API de nivel înalt folosit pentru dezvoltarea și antrenarea rețelelor neurale, fiind capabilă să ruleze pe diferite framework-uri folosite la dezvoltarea de sisteme cu învățare automată, printre acestea numărându-se și TensorFlow.

Keras oferă dezvoltatorilor de rețele neurale mai multe beneficii, permițându-le acestora să înceapă proiectarea unei rețele de la zero, folosind un model de tipul secvențial sau funcțional. De asemenea, această librărie dispune și de câteva dintre cele mai bune rețele pre-antrenate care pot fi folosite pentru a dezvolta un model capabil să rezolve un nou task, folosind procedura transferului de cunoștințe. Pe lângă aceste modele, Keras permite importarea și folosirea unor layere deja implementate, care pot fi adăugate modelului final, specificând parametrii acestora. Câteva dintre aceste layere pe care Keras le oferă sunt următoarele:

- **Conv2D**: Layer folosit pentru realizarea operației de convoluție 2D
- **MaxPool2D**: Layer folosit pentru realizarea operației de Max Pooling 2D
- **Dense**: Layer folosit pentru aplicarea unei funcții de activare
- **Flatten**: Layer folosit pentru comprimarea inputului
- **Dropout**: Layer folosit în timpul antrenării pentru prevenirea **overfitting**-ului

Toate funcțiile, layerele și modele disponibile în librăria Keras sunt descrise în documentația framework-ului³. O dată dezvoltat modelul, acesta poate fi compilat alegând o funcție de optimizare și una de cost. Dacă modelul a fost proiectat corect, acesta poate fi antrenat, Keras oferind posibilitatea salvării celei mai bune variante a acestuia.

³<https://keras.io/api/>

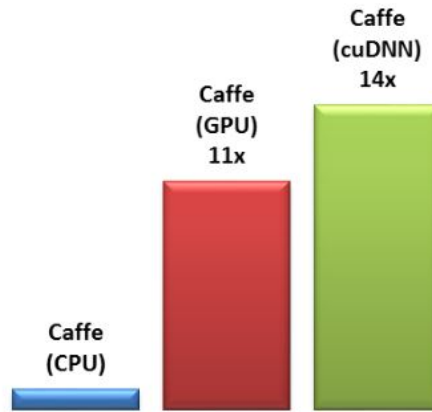


Figura 2.3: Framework-ul CAFFE rulat pe CPU, GPU și GPU + cuDNN [NVIDIA]

Librăria TensorFlow include atât Keras cât și alte funcții și baze de date ce pot fi folosite la dezvoltarea unei rețele neurale convoluționale. Instalarea TensorFlow se poate face foarte ușor într-un environment Python, folosind următoarea comandă:

```
1 $ pip install tensorflow
```

Așadar, TensorFlow și Keras sunt două dintre cele mai puternice librării folosite la dezvoltarea rețelelor neurale și a algoritmilor de învățare automată. Cu o gamă largă de funcții, rețele pre-antrenate și chiar baze de date folosite pentru antrenarea modelelor, TensorFlow este o librărie cu ajutorul căreia se pot dezvolta arhitecturi cu performanțe de recunoaștere remarcabile.

2.4 CUDA Toolkit și cuDNN

CUDA Toolkit este un pachet software oferit de cei de la NVIDIA, folosit pentru crearea unui mediu de dezvoltare a unor aplicații ce folosesc placa video. Cu ajutorul acestui toolkit, se pot dezvolta și optimiza aplicații software și sisteme inteligente ce folosesc învățarea automată.

Pachetul software permite dezvoltarea de algoritmi în C++ și include foarte multe librării și unelte de optimizare și depanare. Pentru a folosi acest toolkit, este nevoie de deținerea unui computer cu placa video dedicată NVIDIA și instalarea driverului corespunzător acesteia.

În dezvoltarea proiectului din lucrarea curentă, acest pachet software a fost necesar pentru a putea antrena și rula arhitectura CNN pe placa video a computerului personal, reducând astfel semnificativ timpii de antrenare și de predicție. Pentru a putea realiza acest lucru, pe lângă instalarea framework-ului TensorFlow și a toolkit-ului CUDA, a fost nevoie și de instalarea unei librării adiționale numită **cuDNN**, ținând cont de anumite aspecte legate de compatibilitate. Această librărie este prima librărie dedicată rețelelor neurale convoluționale dezvoltată de cei de la NVIDIA, îmbunătățind dramatic timpii de antrenare și performanțele placilor video în timpul realizării algoritmilor de convoluție.

Așadar, pentru antrenarea unei arhitecturi CNN, folosirea CUDA Toolkit împreună cu librăria cuDNN reprezintă soluția ideală în cazul în care ne dorim să antrenăm un model pe un computer cu o placa video dedicată NVIDIA. Instalarea versiunilor corespunzătoare ale acestor pachete software împreună cu librăria TensorFlow se poate realiza relativ ușor pentru un utilizator experimentat folosind tutorialul disponibil pe pagina celor de la TensorFlow⁴.

⁴<https://www.tensorflow.org/install/gpu>



Figura 2.4: Exemple de imagini din baza de date Fer2013 [21]

2.5 Baza de date

Baza de date **Fer2013** a fost folosită pentru antrenarea modelului de recunoaștere a emoțiilor umane și conține imagini grayscale cu diferite fețe, de dimensiune 48×48 . Avantajul folosirii acestei baze de date este că majoritatea imaginilor conțin doar fețe, cantitatea de informație din acestea fiind aproximativ egală indiferent de poziționarea feței în poză.

Setul de antrenare conține 35.685 de imagini etichetate, acestea fiind împărțite pe foldere denumite în funcție de emoția prezentă în imaginile respective. Aceste imagini corespund unui număr de 7 clase reprezentând o anumită emoție, după cum urmează: fericit, supărat, neutru, furios, speriat, surprins și dezgustat. Prin procesul de **data augmentation**, putem mări numărul imaginilor disponibile, creând noi imagini bazate pe cele din baza de date, prin adăugarea de zgomot sau rotirea acestora după un anumit plan.

În figura 2.4, se pot observa câteva imagini din baza de date Fer2013. Imaginile ilustrate cochetază cu ideea aplicării modelului final în situații reale, majoritatea dintre acestea fiind caracterizate de condiții reale de lumină, nivele diferite ale intensității emoției exprimate și chiar de prezența mâinilor sau a altor obiecte.

Alt avantaj al folosirii acestei baze de date și implicit crearea unui model cu stratul de intrare de dimensiune $48 \times 48 \times 1$ este faptul că putem aborda chestiunea recunoașterii automate de emoții umane dintr-o perspectivă real-time. Dimensiunea relativ mică a imaginii de la intrarea rețelei facilitează clasificarea acesteia, drept urmare predicția va fi făcută de rețea într-un timp mult mai mic, iar puterea de calcul necesară nu va fi atât de mare.

Pe de altă parte, este nevoie de un compromis între performanțele de recunoaștere în timp real și acuratețea predicțiilor. Dimensiunea relativ mică a acestor imagini, limitează modelul din punctul de vedere al acurateții, acest lucru fiind cauzat de faptul că aceste imagini nu conțin atât de multă informație, fiind totodată și grayscale. Drept urmare, este evident faptul că un model nu va putea să facă diferența atât de bine între două emoții cu pattern-uri similare cum ar fi supărat și nervos sau supărat și neutru, mai ales având în vedere faptul că fizionomia diferă de la o persoană la alta.

Această bază de date a fost creată de Pierre-Luc Carrier și Aaron Courville, aceștia folosind datele pentru un proiect de cercetare. În prezent, baza de date este publică și poate fi folosită de orice entuziast în domeniul învățării automate pentru crearea de noi sisteme bazate pe inteligență artificială. Una din sursele de unde acest set de date poate fi obținut este Kaggle⁵.

⁵<https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge>
data

Capitolul 3

Implementare software

Implementarea software a proiectului de diplomă a fost dezvoltată cu ajutorul tehnologiilor menționate în capitolul anterior, fiind împărțită în trei părți: proiectarea și dezvoltarea arhitecturii, antrenarea rețelei neurale convoluționale și realizarea unei interfețe pentru achiziția semnalului video de la camera web în vederea clasificării emoției detectate de model.

În figura 3.1, se poate observa structura generică a arhitecturii modelului folosit pentru recunoașterea emoțiilor umane. Baza de date folosită pentru antrenarea modelului este Fer2013, structura și particularitățile acestui set de antrenare fiind detaliate, de asemenea, în capitolul anterior. Se poate observa totodată și flow-ul datelor de la intrare, care înainte de a fi folosite ca input pentru rețea, vor trece prin câteva etape simple de preprocesare, cum ar fi convertirea imaginii în imagine cu nuanțe de gri și redimensionarea acesteia.

În continuare, va fi prezentată implementarea software a modelului și vor fi detaliate toate funcțiile folosite.

3.1 Dezvoltarea modelului

Modelul a fost dezvoltat folosind exclusiv funcții din TensorFlow și Keras. Înainte de a începe dezvoltarea propriu-zisă, a fost nevoie să importăm următoarele pachete cu funcții, obiecte și modele pre-definite:

```
from __future__ import print_function
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
    BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import os
```

Următorul pas făcut în configurația modelului, a fost adăugarea unor instrucțiuni prin care vom seta câteva variabile cum ar fi dimensiunea lotului de antrenare, numărul claselor și dimensiunea imaginilor ce vor fi furnizate rețelei. Deoarece pe parcursul dezvoltării și antrenării modelului, au apărut probleme de recunoaștere din cauza faptului că modelul confunda anumite emoții, am ales să folosesc doar 5 emoții de bază, după cum urmează: nervos, fericit, neutru, supărat și surprins.

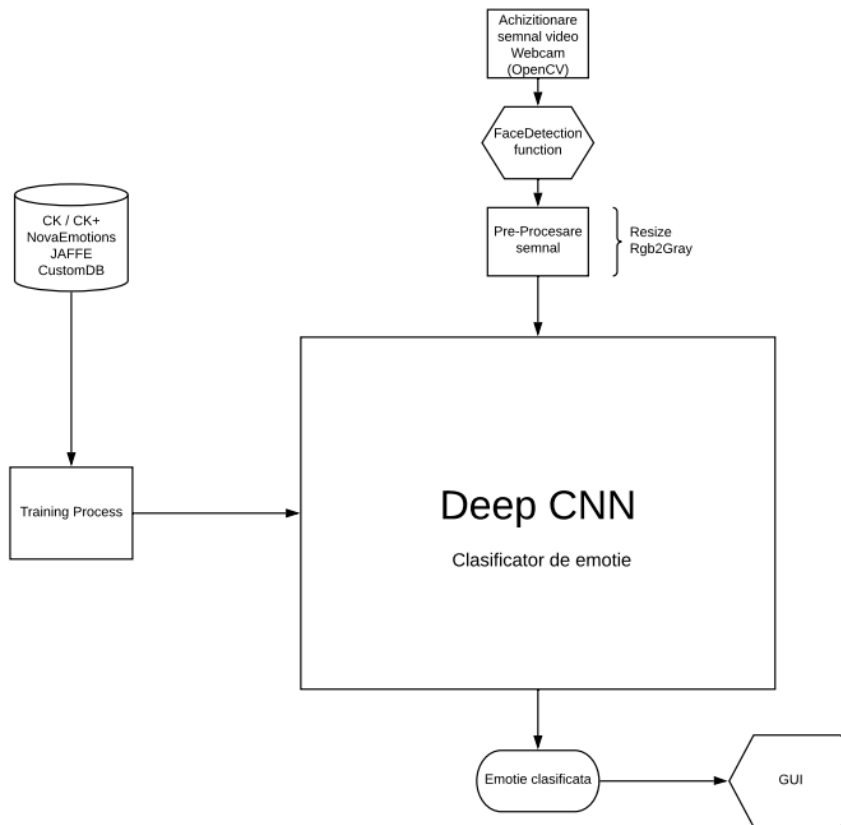


Figura 3.1: Diagrama ilustrând structura generică a arhitecturii de clasificare

Dimensiunea lotului de antrenare este de 32, această valoare fiind aleasă empiric, dar și în funcție de dimensiunea setului de antrenare, puterea de calcul disponibilă și influența acestui parametru în acuratețea de predicție a modelului final. Aceste lucruri vor fi discutate ulterior într-o analiză comparativă a rezultatelor obținute.

3.1.1 Achiziția și augmentarea datelor

Această parte a implementării software este relativ simplă o dată ce am reușit să stocăm imaginile din baza de date pe stația locală (majoritatea surselor oferă imaginile din bazele de date în format `.csv`, acest lucru fiind uneori problematic). O dată stocate pe disc, fluxul de date dintre director și model va fi realizat cu ajutorul unui obiect numit `ImageDataGenerator`, disponibil în pachetul de preprocesare a imaginilor de la Keras.

Obiectul amintit mai sus, oferă pe lângă funcția de încărcare a datelor din director, și posibilitatea augmentării acestora. Augmentarea datelor este procesul prin care putem să adăugăm mai multe date la setul de antrenare disponibil, îmbunătățind în acest fel performanțele modelului. Pentru a putea dezvolta un model robust, acesta trebuie să învețe conceptul de invarianță translațională, cu alte cuvinte să poată recunoaște pattern-urile de interes indiferent de poziția acestora în imagine. De asemenea, pe lângă faptul că putem schimba orientarea unei imagini prin aplicarea unor răsturnări orizontale sau verticale, cu ajutorul aceluiași obiect putem schimba luminile ce caracterizează o imagine, putem face zoom sau chiar să rotim după un anumit unghi imaginea respectivă. Parametrii acestui obiect pot fi aleși în orice combinație posibilă, însă cel mai important aspect este alegerea unui parametru ce va asigura rețelei învățarea conceptului de invarianță translațională. Baza de date a fost împărțită în două părți, după cum urmează: 90% din date au fost folosite pentru antrenare, iar 10% din acestea au fost folosite pentru validare.

În continuare, este prezentată implementarea obiectului `ImageDataGenerator` ce asigură augmentarea datelor:

```
#path-urile catre directoarele ce contin imaginile de test si de validare
train_data_dir = '/home/dani/Licenta/datasets/fer2013/train'
validation_data_dir = '/home/dani/Licenta/datasets/fer2013/validation'
```

```
# data augmentation pentru o baza de date mai mare
```

```
def augmentare():
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=30,
        shear_range=0.2,
        zoom_range=0.2,
        width_shift_range=0.3,
        height_shift_range=0.3,
        horizontal_flip=True,
        fill_mode='nearest')
    return train_datagen

def normare():
    validation_datagen = ImageDataGenerator(rescale=1./255)
    return validation_datagen
```

Funcția de augmentare are rolul de a genera un obiect cu ajutorul căruia imaginile din baza de date vor fi modificate pentru a îmbunătăți performanțele modelului. Aceasta va genera totodată imagini noi care vor fi normate, asemenea celor din setul de validare.

Pentru a genera setul de antrenare și cel de validare, a fost implementată o funcție ce folosește obiectul creat anterior pentru achiziția și augmentarea datelor. Modelul dezvoltat în Keras are nevoie de un obiect special pentru stocarea datelor de antrenare, obiect creat cu ajutorul funcției `flow_from_directory()` ce asigură fluxul de date de la directoarele ce conțin imaginile către acesta.

În continuare, este prezentată implementarea funcțiilor ce asigură achiziția imaginilor de intrare pentru rețeaua neurală convoluțională:

```
def generateTrainSet():
    train_datagen = augmentare()
    train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        color_mode='grayscale',
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True)
    return train_generator

def generateValidationSet():
    validation_datagen = normare()
    validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
```

```

        color_mode='grayscale',
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True)
    return validation_generator

```

Pe lângă aplicarea procedeelelor de augmentare și normare a imaginilor disponibile în baza de date Fer2013, funcțiile de mai sus au și rolul de a asigura dimensiunea și formatul datelor în funcție de setarea anumitor parametri. Drept urmare, imaginile vor fi grayscale, de dimensiune 48×48 , lotul va avea dimensiunea setată la începutul scriptului iar etichetarea imaginilor se va face pe categorii în funcție de denumirea folderului în care se afla acestea. Imaginile vor fi achiziționate în mod aleator din directoarele indicate, urmând a fi stocate în obiectele `train_generator` și `validation_generator`.

3.1.2 Arhitectura rețelei

Arhitectura modelului de recunoaștere a emoțiilor umane este una secvențială, fiind alcătuită din 7 blocuri principale, ce cuprind straturi convoluționale, de pooling, de normalizare a lotului și straturi complet conectate ce asigură extragerea caracteristicilor și clasificarea propriu-zisă a imaginii.

Primul bloc al arhitecturii este un bloc convoluțional, conținând neuroni de intrare proiectați pentru dimensiunea imaginilor din setul de antrenare. Acest bloc conține două straturi convoluționale cu un număr de 32 de filtre de dimensiune 3×3 și aplică totodată setarea *zero-padding* înainte de realizarea convoluțiilor dintre imaginea de la intrare și filtre. Pentru inițializarea ponderilor filtrelor din cele două straturi convoluționale ale acestui bloc, am folosit setarea `he_normal`, ce asigură o distribuție normală a acestor valori ce urmează a fi actualizate pe măsură ce rețeaua neurală învață. Actualizarea ponderilor se va face bazat pe funcția de cost și un algoritm de tipul *back-propagation*. Funcția de activare folosită pentru ieșirea celor două straturi convoluționale este ELU (Exponential Linear Unit), datorită convergenței și a rezultatelor obținute în urma folosirii acesteia. Acest bloc conține și două straturi de normalizare a lotului, fiecare dintre acestea precedând straturile convoluționale și asigurând în acest fel îmbunătățirea vitezei, performanței și stabilității arhitecturii prin normalizarea activărilor și realizarea unor operații de recentrare și rescalare a ieșirilor unui anumit strat. La ieșirea acestui bloc se află un strat de `MaxPooling2D` de dimensiune 2×2 și un strat de `Dropout` responsabil cu "ignorarea" în mod aleator a unui număr de neuroni pentru prevenirea overfitting-ului.

În continuare, este prezentată implementarea primului bloc convoluțional și a straturilor componente ale acestuia, din funcția `buildModel()`, responsabilă cu proiectarea arhitecturii modelului de recunoaștere a emoțiilor:

```

model = Sequential()

# Bloc convolutional (1)

model.add(Conv2D(32, (3, 3), padding='same', kernel_initializer='he_normal',
input_shape=(img_rows, img_cols, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), padding='same', kernel_initializer='he_normal',
input_shape=(img_rows, img_cols, 1)))
model.add(Activation('elu'))

```

```
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

Următoarele trei blocuri sunt asemănătoare ca structura, având în principal aceleași straturi în componența lor. Diferența este că pe măsură ce avansăm în arhitectură, se dublează numărul filtrelor straturilor convoluționale. Primul bloc dispune de straturi convoluționale cu un număr de 32 de filtre și este responsabil cu atribuirea datelor către neuronii de intrare și cu extragerea caracteristicilor generale ale imaginilor. Următoarele trei blocuri dispun de straturi convoluționale cu un număr de 64, 128, respectiv 256 de filtre, de dimensiune 3×3 , responsabile cu extragerea caracteristicilor distinctive ale imaginilor. Straturile de pooling ce intră în componența acestor trei blocuri sunt, de asemenea, de tipul `MaxPooling2D` având filtre de dimensiune 2×2 .

Blocul numărul 5 conține un strat de tip `Flatten` și un strat de tip `Dense`, complet conectate, responsabile cu aplatizarea (comprimarea) ieșirii ultimului strat convoluțional respectiv cu asigurarea conexiunilor între neuronii din straturile ascunse ale blocurilor convoluționale și neuronii din următorul bloc ce conține doar straturi de tip `fully-connected`. Dimensiunea volumului de ieșire este de 64 în ambele straturi complet conectate din blocurile 5 și 6 iar ca funcție de activare este folosită tot funcția ELU. Sunt adăugate și straturi de `Dropout` cu un procentaj de 50% pentru prevenirea overfitting-ului.

În continuare, este prezentată implementarea ultimelor trei blocuri ale arhitecturii:

```
# Bloc flatten (5)

model.add(Flatten())
model.add(Dense(64, kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Bloc fully connected (6)

model.add(Dense(64, kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Bloc fully connected clasificator (7)

model.add(Dense(nr_clase, kernel_initializer='he_normal'))
model.add(Activation('softmax'))
```

Dimensiunea volumului de ieșire al ultimului strat complet conectat va fi egală cu numărul claselor, mai exact 5. Ponderile acestui strat vor fi de asemenea inițializate folosind o distribuție normală, ele fiind actualizate pe baza funcțiilor de cost și de optimizare cu ajutorul unui algoritm de tip *back-propagation*, pe măsură ce rețeaua se antrenează și învață. Funcția de activare folosită pentru neuronii de ieșire ai ultimului strat este `softmax`, ea fiind folosită în majoritatea arhitecturilor de rețele neurale responsabile cu clasificarea intrării în mai multe clase. Această funcție este ilustrată în figura 3.2.

Implementarea software completă a arhitecturii modelului de recunoaștere a emoțiilor umane este realizată cu ajutorul funcției `buildModel()` care poate fi analizată în detaliu consultând anexele acestei lucrări.

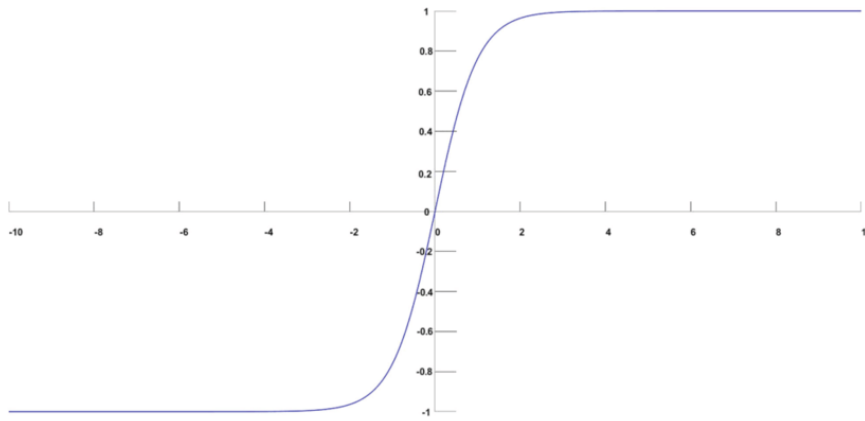


Figura 3.2: Reprezentarea funcției de activare Softmax[22]

3.2 Antrenarea rețelei

După implementarea funcțiilor de achiziție și augmentare a datelor, precum și a funcției de construire a arhitecturii rețelei neurale convoluționale, următorul pas în implementarea software a proiectului este dezvoltarea unei funcții cu ajutorul căreia modelul poate primi datele și se poate antrena.

În mare parte, crearea unei funcții cu ajutorul căreia modelul poate începe procesul de antrenare este relativ simplă, însă înainte de începerea procesului de antrenare propriu-zis, trebuie ca modelul de recunoaștere a emoțiilor umane să fie compilat, alegând o funcție de optimizare, o funcție de cost și, opțional, felul în care dorim să observăm performanțele de recunoaștere ale modelului.

3.2.1 Funcția de cost

Funcția de cost aleasă pentru modelul propus se numește **categorical_crossentropy**, fiind inclusă în librăria Keras. Această funcție de cost este o funcție clasică, des întâlnită în arhitecturile de rețele neurale, ce permite calcularea pierderii dintre etichetele datelor și predicțiile făcute de model (ieșirea dorită comparativ cu ieșirea reală).

3.2.2 Funcția de optimizare

Funcția de optimizare aleasă pentru modelul propus se numește **Adam** și este, de asemenea, inclusă în librăria Keras. Această funcție de optimizare este folosită în algoritmul de tip *back-propagation* de modificare a ponderilor rețelei, fiind sensibil diferită față de abordările bazate pe algoritmi ce folosesc tehnica gradientului negativ. În timp ce algoritmi clasici, bazați pe **Gradient Descent**, mențin neschimbată valoarea ratei de învățare pe tot parcursul procesului de antrenare, algoritmul de optimizare **Adam** calculează media gradientului și a gradientului pătrat în timp ce rețeaua învață, rata de învățare putând fi modificată în funcție de rezultatele obținute. Adam este un algoritm popular de optimizare a rețelelor neurale cu învățare profundă datorită faptului că dă rezultate foarte bune într-un timp relativ scurt, comparativ cu alte funcții de optimizare clasice.

În continuare, este prezentată implementarea funcției de compilare a modelului propus, alegându-se funcția de cost amintită mai sus, iar pentru funcția de optimizare Adam, a fost aleasă o rată de învățare inițială $\eta = 0.001$:

```
def compileModel(model):  
    model.compile(loss='categorical_crossentropy',  
                  optimizer=Adam(lr=0.001),  
                  metrics=['accuracy'])
```

3.2.3 Callbacks

După implementarea funcției de compilare a modelului, următorul pas în implementarea software a proiectului a fost implementarea unei funcții de generare a așa-ziselor `callback-uri`. Aceste `callback-uri` au fost implementate cu ajutorul unor funcții disponibile în librăria Keras și au rolul de a optimiza procesul de antrenare al rețelei. Primul callback setat este `ModelCheckpoint`, callback prin parametrii căruia putem alege ca modelul să fie salvat sub o anumită denumire sau să fie salvate doar ponderile cele mai bune. Cel de-al doilea callback, se numește `EarlyStopping` și este folositor la stoparea procesului de antrenare al rețelei, atunci când nu s-au produs îmbunătățiri vreme de un anumit număr de epoci setat. Ultimul callback generat de funcție se numește `ReduceLROnPlateau` și cu ajutorul acestuia putem scădea rata de învățare cu un anumit factor setat dacă nu s-au produs îmbunătățiri ale performanței rețelei vreme de un număr de epoci setat. Ultimele două callback-uri amintite, monitorizează valoarea funcției de cost pentru a decide dacă s-au produs sau nu îmbunătățiri de la o epocă la alta.

În continuare, este prezentată implementarea funcției `generate_callbacks()`, responsabilă cu generarea callback-urilor amintite mai sus:

```
def generate_callbacks(modelName, patience_es, patience_lr):  
    checkpoint = ModelCheckpoint(modelName,  
                                 monitor='val_loss',  
                                 mode='min',  
                                 save_best_only=True,  
                                 verbose=1)  
  
    earlystop = EarlyStopping(monitor='val_loss',  
                              min_delta=0,  
                              patience=patience_es,  
                              verbose=1,  
                              restore_best_weights=True)  
  
    reduce_lr = ReduceLROnPlateau(monitor='val_loss',  
                                  factor=0.2,  
                                  patience=patience_lr,  
                                  verbose=1,  
                                  min_delta=0.0001)  
  
    callbacks = [earlystop, checkpoint, reduce_lr]  
  
    return callbacks
```

O dată implementate toate aceste funcții amintite până acum, procesul de antrenare al rețelei devine o chestiune simplă. Antrenarea rețelei neurale convoluționale poate începe o dată ce modelul a fost compilat, iar imaginile de testare și validare din baza de date au fost încărcate încărcate în obiecte specifice funcțiilor de achiziție și augmentare a datelor. Numărul imaginilor din setul de antrenare și validare este afișat în consola o dată ce obiectele de stocare a datelor au fost generate. Aceste numere trebuie stocate în două variabile care vor fi pasate ca argument funcției responsabilă cu antrenarea modelului, în vederea calculării numărului de

pași pe epocă în timpul antrenării și al numărului de pași de validare. Numărul de epoci poate fi setat manual și pasat ca argument aceleiași funcții.

În continuare, este prezentată implementarea funcției de antrenare a modelului:

```
def antrenareModel(model, train_set, validation_set, nb_train_samples,
nb_validation_samples, epochs, callbacks):
    history = model.fit_generator(
        train_set,
        steps_per_epoch=nb_train_samples // batch_size,
        epochs=epochs,
        callbacks=callbacks,
        validation_data=validation_set,
        validation_steps=nb_validation_samples // batch_size)

    return history
```

După finalizarea procesului de antrenare al modelului, cea mai bună variantă a acestuia este salvată în directorul local, urmând a fi folosită pentru predicții în eventualele aplicații de recunoaștere a emoțiilor umane.

3.3 Clasificarea emoției

Pentru a putea clasifica o anumită emoție folosind modelul implementat și antrenat cu ajutorul funcțiilor enumerate în acest capitol, a fost creată o interfață în OpenCV pentru achiziția semnalului video de la camera web a stației de lucru. Modelul poate fi încărcat relativ ușor din directorul în care a fost salvat, prin folosirea unei alte funcții disponibilă în librăria Keras, și anume `load_model`.

Clasificarea emoției presupune detectarea feței din streamul video și procesarea acesteia pentru a ajunge la dimensiunea și formatul imaginilor cu ajutorul cărora a fost antrenată rețeaua neurală convoluțională.

3.3.1 Detectarea feței

Detectarea feței din imaginile provenite din semnalul video de la camera web a fost realizată cu ajutorul unei funcții disponibilă în librăria OpenCV. Această funcție este un clasificator de tip *Cascade* ce permite încărcarea unei funcții din familia *Haar Cascade*. Această funcție open-source vine în format `.xml` și este pasată ca argument funcției de clasificare din OpenCV.

Implementarea clasificatorului ce ajută la detectarea feței dintr-o anumită imagine este o chestiune destul de simplă o dată ce funcția `haarcascade_frontalface_default.xml` a fost stocată în directorul curent:

```
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

Următorul pas este să pornim achiziția semnalului video de la camera web cu ajutorul unei funcții din OpenCV:

```
video_input = cv2.VideoCapture(0)
```

O dată ce semnalul este achiziționat, acesta trebuie eșantionat în frame-uri pentru a face mai ușoară prelucrarea imaginii și detectarea feței. Frame-urile sunt citite cu ajutorul funcției `read()`, acestea fiind ulterior convertite în imagini cu nuanțe de gri și pasate ca argument

funcției `detectMultiScale()`. Această funcție poate primi ca parametri atât imaginea cât și un factor de scalare pentru reducerea dimensiunii imaginii, existând totodată și posibilitatea introducerii unui număr referitor la numărul de vecini pe care un dreptunghi candidat ar trebui să-i aibă pentru a fi păstrat. Am determinat empiric că valorile pentru care funcția detectează destul de bine fețele sunt cuprinse în intervalul $[3, 6]$.

În continuare, este prezentată o parte a scriptului responsabilă cu detecția feței din frame-urile obținute după eșantionarea semnalului video:

```
ret, frame = video_input.read()

gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_classifier.detectMultiScale(gray_frame, 1.3, 6)
```

Variabila `faces` va fi o listă în care vor fi stocate coordonatele feței, mai exact coordonatele punctului de început, lățimea și lungimea ce marchează o față dintr-un anumit frame. Folosind valorile stocate în această variabilă, zona de interes din frame (cea care conține o față) este marcată cu un dreptunghi și totodată extrasă și stocată într-o nouă variabilă care va fi folosită după procesare ca intrare pentru modelul de recunoaștere a emoțiilor umane.

3.3.2 Procesarea imaginii

Procesarea imaginii este relativ simplă, o dată ce funcția OpenCV a reușit să detecteze o față într-un frame. Zona de interes ce conține fața este stocată într-o nouă imagine ce va fi redimensionată la 48×48 pentru a putea fi folosită ca input pentru rețeaua neurală convoluțională.

După stocarea zonei ce conține fața într-o imagine, valorile pixelilor din noua imagine sunt normalizate. Folosind funcția `img_to_array` disponibilă în pachetul de procesare al imaginilor al librăriei Keras, imaginea obținută în urma aplicării clasificatorului de fețe și a celorlalte funcții de procesare și redimensionare, este convertită într-un vector ce va conține valorile tuturor celor 2304 pixeli ai imaginii. Această conversie este făcută pentru ca imaginea ce conține fața unui subiect să poată fi oferită ca intrare modelului în vederea clasificării acesteia.

În continuare, este prezentată cea mai importantă parte a scriptului de detecție a emoției dintr-un stream video, cu ajutorul modelului de recunoaștere a emoțiilor umane proiectat anterior și a funcției OpenCV de detectare a feței:

```
for (x, y, w, h) in faces:
    #Incadram cu un dreptunghi de culoare verde fata detectata in imagine
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    #Selectam regiunea dorita din imagine si o convertim in imagine gri de
    dimensiune 48x48
    roi_gray = cv2.cvtColor(frame[y:y+h,x:x+w], cv2.COLOR_BGR2GRAY)
    roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)

    #Daca se detecteaza vreo fata in inputul video, atunci o convertim in array
    pentru a putea fi clasificata
    if np.sum([roi_gray])!=0:
        roi = roi_gray.astype('float')/255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi,axis=0)
```

Funcția de optimizare	Acuratețe	Loss
Adam	60.25%	1.2215
RMSProp	59.42%	1.2893
SGD	58.33%	1.3015

Tabela 3.1: Performanțele rețelei și funcțiile de optimizare folosite

```
#Modelul face o predictie, iar in functie de predictia facuta se adauga in
frame emotia recunoscuta
```

```
preds = emotion_classifier.predict(roi)[0]
label = class_labels[preds.argmax()]
label_position = (x, y)
cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 2,
(0, 255, 0), 3)
```

Modelul rezultat în urma antrenării, funcția de detectare a feței, arhitectura modelului propus precum și toate celelalte resurse software implementate în cadrul acestui proiect sunt disponibile online.

3.4 Experimente și rezultate

În urma implementării și antrenării modelului de recunoaștere a emoțiilor umane au fost realizate diferite teste cu scopul de a determina empiric configurația rețelei care maximizează calitatea rezultatelor obținute.

Configurația rețelei constă în principal, pe lângă modificarea hiperparametrilor, alegerea unor funcții de cost, optimizare și activare specifice atât modului în care a fost proiectată arhitectura cât și bazei de date folosite pentru antrenarea modelului.

În timp ce funcția de cost nu a jucat un rol atât de important, rezultatele fiind în mare parte asemănătoare indiferent de funcția de cost folosită, funcția de activare și cea de optimizare se reflectă destul de puternic în performanțele rețelei. Funcția de cost folosită a fost `categorical_crossentropy`, alternativa acesteia fiind folosirea funcției `sparse_categorical_crossentropy`, singura diferență dintre acestea fiind legată de formatul datelor folosite în antrenarea rețelei.

Funcția de activare folosită în interiorul arhitecturii a fost `ELU` (*Exponential Linear Unit*) iar cea folosită pentru stratul de ieșire a fost funcția de activare `Softmax`, aceasta din urmă fiind de departe cea mai folosită funcție de activare pentru modelele de clasificare a datelor în clase multiple. Funcția de activare `ELU` a dat cele mai bune rezultate în comparație cu `ReLU` (*Rectified Linear Unit*) și funcția sigmoidă. Funcția de activare `ReLU` conduce la pierderea unor neuroni și la o actualizare nepotrivită a ponderilor care cauzează neactivarea acestora. Această problemă este cunoscută sub numele de *"the dying ReLU problem"* și poate fi rezolvată prin folosirea funcției de activare `Leaky ReLU`, însă din cauza liniarității, această funcție nu poate fi folosită în rezolvarea task-urilor complexe.

Optimizarea modelului pentru recunoașterea emoțiilor umane a fost făcută folosind funcția de optimizare `Adam`. În tabela 3.1, se poate observa diferența dintre funcțiile de optimizare `Adam`, `RMSProp` și `SGD`. Funcția de optimizare `Adam` conduce la obținerea celor mai bune rezultate, această funcție fiind folosită în majoritatea arhitecturilor de rețele neurale.

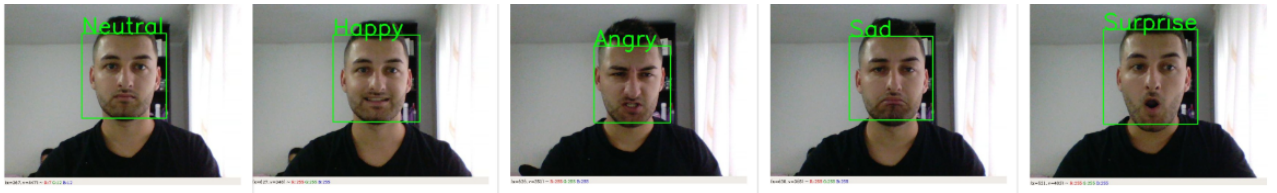


Figura 3.3: Recunoașterea celor 5 emoții de către modelul CNN

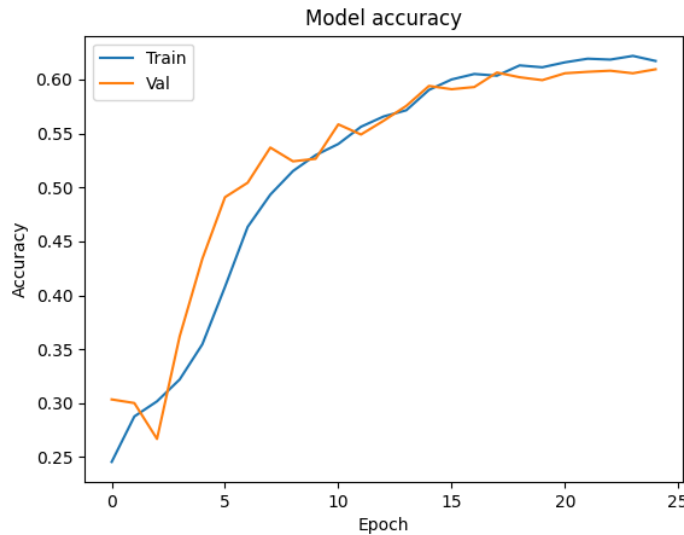


Figura 3.4: Evoluția acurateții pe loturile de test și validare

Rezultatele obținute cu ajutorul aplicației ce folosește modelul propus de rețea neurală convoluțională pot fi observate în figura 3.3. Modelul reușește să clasifice toate emoțiile, însă rezultatele sunt puternic influențate de fizionomia subiecților și de condițiile reale. Lumina și fundalul pot afecta uneori funcționarea aplicației în sensul că funcția de detectare a feței nu dă cele mai bune rezultate, în lipsa unei imagini modelul neavând practic ce predicție să facă.

3.5 Hiperparametri

De cele mai multe ori, rezultatele obținute sunt influențate în mod direct de hiperparametrii rețelei neurale, determinarea celei mai bune combinații de hiperparametri realizându-se în majoritatea cazurilor empiric. În principal, hiperparametrii ce țin de învățarea caracteristicilor spațiale cum ar fi adâncimea rețelei, numărul și dimensiunea filtrelor sau chiar straturile ce alcătuiesc o anumită arhitectură sunt aleși atât în funcție de task-ul ce se dorește a fi rezolvat de rețea cât și de structura și dimensionalitatea imaginilor din setul de antrenare.

Dimensiunea lotului de antrenare (*batch size*) definește numărul de sample-uri care sunt procesate și propagate prin rețea înainte ca modelul să-și actualizeze ponderile. Acest hiperparametru poate fi determinat empiric, cele mai întâlnite valori fiind de 32, 64 și 128. Pentru modelul propus în această lucrare, dimensiunea lotului de antrenare a fost 32, deoarece această este valoarea optimă pentru care se obține atât un timp de antrenare relativ scurt, cât și o acuratețe a predicțiilor satisfăcătoare. Avantajul folosirii acestei dimensiuni a lotului de antrenare este că în acest fel se utilizează destul de puțină memorie, acest lucru fiind extrem de util în cazurile în care setul de antrenare depășește memoria stației de lucru. Totodată, rețeaua se antrenează mult mai rapid în cazul folosirii unui *mini-batch* (dimensiune relativ redusă a lotului de antrenare), acest lucru datorându-se faptului că ponderile sunt actualizate mai des.

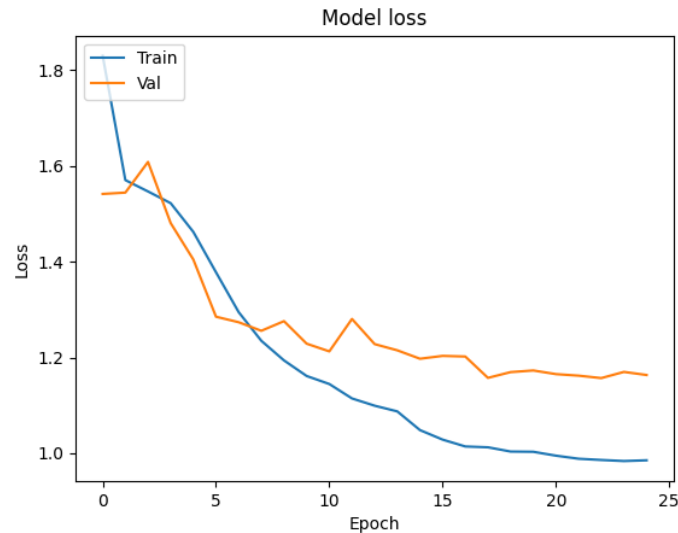


Figura 3.5: Evoluția funcției de cost pe loturile de test și validare

Rata de învățare este alt hiperparamtru ce influențează în mod direct rezultatele și performanța rețelei neurale. Cu ajutorul acestui hiperparamtru, putem controla în ce măsură vom modifica modelul raportându-ne la eroarea estimată de funcția de cost atunci când ponderile rețelei sunt actualizate. De asemenea, acest hiperparamtru este determinat de cele mai multe ori empiric, ținându-se cont și de arhitectura rețelei. O rată de învățare prea mică va conduce la un timp de antrenare foarte mare, existând totodată și riscul ca modelul să rămână blocat și să nu mai apară îmbunătățiri de la o epocă la alta. Pe de altă parte, dacă se alege o valoare prea mare pentru rata de învățare, modelul va învăța mult prea repede iar ponderile rezultate vor fi departe de valorile optime. Pentru modelul propus, am ales o rată de învățare inițială $\eta = 0.001$, această valoare putând fi micșorată dacă nu a apărut nicio îmbunătățire a performanței rețelei după un număr setat de epoci. Modificarea ratei de învățare pe parcursul antrenării poate fi realizată cu ajutorul unei funcții descrise anterior (din librăria Keras) și este cea mai bună soluție atunci când dorim ca rețeaua să fie capabilă să învețe lucruri complexe într-un mod eficient, deoarece putem porni de la o rată de învățare optimă pe care o putem micșora ulterior în funcție de evoluția modelului.

3.6 Performanța rețelei

Performanța rețelei neurale convoluționale este multumitoare având în vedere faptul că modelul a fost dezvoltat de la zero, iar baza de date folosită pentru antrenare conține imagini de dimensiune relativ mică. Acuratețea de predicție obținută de modelul propus este aproximativ 61%.

În figura 3.4, se poate observa evoluția acurateții de predicție în timpul procesului de antrenare. Acesta a ajuns relativ rapid la acuratețea de 60.25%, fapt ce se datorează atât funcției de optimizare **Adam** cât și a callback-uri care au ajutat modelul să ajungă la aceasta performanță în doar 25 de epoci.

În ceea ce privește funcția de cost, aceasta a scăzut destul de rapid, însă valoarea finală a acesteia pe setul de validare este totuși destul de ridicată. În figura 3.5, este ilustrată reprezentarea grafică a funcției de cost pe lotul de validare și cel de antrenare. Funcția scade exponențial și atinge o valoare apropiată de cea finală după primele 10 epoci, urmând ca după să varieze ușor până spre valoarea finală aproximativ egală cu 1.22.

Cu toate că performanțele rețelei mai pot fi îmbunătățite, aceasta se descurcă foarte bine în a clasifica emoții real-time, atunci când mediul pe care este rulată aplicația de recunoaștere a emoțiilor dispune de o putere de calcul decentă. Rezultatele obținute în urma testării aplicației ce folosește rețeaua neurală convoluțională de recunoaștere a emoțiilor depind în principal de mediul în care este captat semnalul video. Dacă lumina este prea intensă sau prea scăzută, aplicația are dificultăți în a identifica fața și drept urmare, modelului nu îi pot fi furnizate imagini în vederea clasificării acestora. Problema principală a modelului atunci când ne referim strict la performanțele de recunoaștere, este incapacitatea acestuia de a distinge uneori între emoțiile supărat și nervos. Acest lucru este cauzat de faptul că cele două emoții seamănă foarte mult, iar imaginile de dimensiune mică nu conțin atât de multă informație încât rețeaua să poate face diferența între aceste două emoții. Trebuie conchis totodată și asupra faptului că rețeaua nu prezintă o acuratețe a predicțiilor foarte ridicată, acest lucru reflectându-se direct în performanțele acesteia.

Toate resursele software implementate în cadrul acestui proiect de diplomă pot fi consultate online¹.

¹<https://github.com/danifratila97/Detectarea-emoțiilor-umane-folosind-rețele-neurale-convolutionale/>

Concluzii

Concluzii generale

Implementarea proiectului de diplomă a reprezentat o ocazie prin care am putut aprofunda mai mult tehnicile actuale folosite în domeniul inteligenței artificiale și al învățării automate. Modelul de recunoaștere a emoțiilor umane bazat pe rețele neurale convoluționale arată o performanță de recunoaștere foarte bună și funcționează în timp real folosind o putere computațională relativ mică, acest fapt datorându-se în principal dimensionalității volumului de intrare.

Arhitectura a fost dezvoltată de la zero, folosind o bază de date cu imagini grayscale de dimensiune relativ mică, acest fapt conducând la obținerea unui scor de recunoaștere mulțumitor, acuratețea predicțiilor fiind mai mică comparativ cu ceea ce îmi propusesem să obțin. Alternativa ar fi fost dezvoltarea unui model folosind tehnica transferului de cunoștințe, însă majoritatea rețelelor pre-antrenate funcționează cu imagini RGB, de dimensiune mai mare. Artificiul prin care am putut folosi imaginile grayscale din baza de date Fer2013 a fost să ”repet” imaginea pe toate cele 3 canale și să o ofer ca input unei rețele pre-antrenate cu imagini RGB, dar rezultatele nu au fost cele mai bune.

În concluzie, putem spune ca rețelele neurale convoluționale sunt unele dintre cele mai bune rețele cu învățare profundă, performanțele acestora fiind remarcate în majoritatea domeniilor. Implementarea unui model robust de recunoaștere a emoțiilor umane este, cu siguranța, una dintre cele mai fascinante aplicații ale CNN-urilor, chestiunea înțelegerii emoțiilor de către computer fiind un lucru fascinant într-o lume în care tot mai multe domenii sunt marcate de interacțiunea om-mașină.

Dezvoltări ulterioare

Domeniul inteligenței artificiale este unul cu o aplicabilitate vastă care ne va marca cu siguranță viitorul prin aplicațiile și tehnologiile care vor fi dezvoltate cu ajutorul rețelelor neurale cu învățare profundă.

O posibilă dezvoltare ulterioară bazată pe acest proiect ar fi implementarea unei rețele neurale convoluționale folosind tehnica transferului de cunoștințe. În acest sens, ar trebui construit un set de antrenare propriu, folosind imagini RGB de dimensiune mai mare. Această eventuală arhitectură ar putea obține o acuratețe a predicțiilor destul de mare și ar putea fi baza unui model robust de recunoaștere a emoțiilor umane.

O altă aplicație interesantă a rețelelor neurale convoluționale ar putea fi evaluarea unor anumite servicii din aplicațiile existente pe piață prin realizarea unei fotografii la finalizarea unei anumite interacțiuni între firmă și client. Imaginea ar putea fi clasificată de un model de

recunoaștere a emoțiilor, iar în funcție de emoția detectată, serviciului să-i fie acordat automat un review sau o notă. Acest lucru ar fi cu siguranță mult mai util și mai plăcut decât actualele metode prin care putem acorda o recenzie unui anumit produs sau serviciu.

Bibliografie

- [1] J. A. Miranda, M. Abadi, N. Sebe, and I. Patras. Amigos: A dataset for affect, personality and mood research on individuals and groups (pdf). *Queen Mary University of London and Università Degli Studi of Trento*, 2017.
- [2] Luz Santamaria-Grandos, Mario Munoz-Organero, Gustavo Ramirez-Gonzalez, Enas Abdulhay, and N. Arunkumar. Using deep convolutional neural network for emotion detection on a physiological signals dataset (amigos). *IEEE Access*, octombrie 2018.
- [3] Chieh-En Li Lanqing Zhao. Emotion recognition using convolutional neural networks. *Purdue Undergraduate Research Conference*, 2019.
- [4] Shruti Jaiswal and G. C. Nandi. Robust real-time emotion detection system using cnn architecture. *Neural Computing and Applications*, octombrie 2019.
- [5] UPB. Laboratorul de analiza și prelucrarea imaginilor.
- [6] J. M. Zurada. Introduction to artificial neural systems. *West publishing company St. Paul*, 1992.
- [7] S. Haykin. Neural networks and learning machines, 3rd ed. *Pearson education Upper Saddle River*, 2009.
- [8] Victor Neagoe. Rețele neurale pentru explorarea datelor. *Matrix Rom*, 2018.
- [9] S. Sharma. Activation functions in neural networks. *Online: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>*, septembrie 2017.
- [10] C. McDonald. Machine learning fundamentals (i): Cost functions and gradient descent. *Online: <https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220>*, noiembrie 2017.
- [11] S. Saha. A comprehensive guide to convolutional neural networks — the eli5 way. *Online: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>*, decembrie 2018.
- [12] R. Nash și K. O'Shea. An introduction to convolutional neural networks. *arXiv preprints arXiv:1511.08458*, 2015.
- [13] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. iulie 2020.

-
- [14] Anibal Pedraza, Jaime Gallego, Samuel Lopez, and Lucia Gonzalez. Glomerulus classification with convolutional neural networks. *Annual Conference on Medical Image Understanding and Analysis*, iunie 2017.
 - [15] Sik-Ho Tsang. Review: Zfnet — winner of ilsvrc 2013 (image classification). *Online: <https://medium.com/coinmonks/paper-review-of-zfnet-the-winner-of-ilsvrc-2013-image-classification-d1a5a0c45103>*, august 2018.
 - [16] Sik-Ho Tsang. Review: Googlenet (inception v1)— winner of ilsvrc 2014 (image classification). *Online: <https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvrc-2014-image-classification-c2b3565a64e7>*, august 2018.
 - [17] Manolis Loukadakis, José Cano, and Michael O’Boyle. Accelerating deep neural networks on low power heterogeneous architectures. ianuarie 2018.
 - [18] P. Ruiz. Understanding and visualizing resnets. *Online: <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>*, octombrie 2018.
 - [19] Joseph Howse, Prateek Joshi, and Michael Beyeler. *OpenCV: Computer Vision Projects with Python*. octombrie 2016.
 - [20] Tom Hope, S. Yehezkel, and Itay Lieder. Learning tensorflow: A guide to building deep learning systems. *O’Reilly Media*, august 2017.
 - [21] Christopher Pramerdorfer and Martin Kampel. Facial expression recognition using convolutional neural networks: State of the art. 12 2016.
 - [22] Shen Leixian, Qingyun Zhang, Guoxu Cao, and He Xu. *Fall Detection System Based on Deep Learning and Image Processing in Cloud Environment*, pages 590–598. 01 2019.
 - [23] R. W. Picard. Affective computing. *MIT Press*, 1997.
 - [24] D. Buhalis and A. Amaranggana. “smart tourism destinations enhancing tourism experience through personalisation of services. *Information and Communication Technologies in Tourism*, 2015.
 - [25] M. Makikawa, N. Shiozawa, and S. Okada. “chapter 7.1 - fundamentals of wearable sensors for the monitoring of physical and physiological changes in daily life,” in wearable sensors. *Oxford: Academic Press*, 2014.
 - [26] R. Carter. A practical guide to tourism destination management. *Ed. UNWTO publications*, 2007.
 - [27] Koichiro Yasaka, Hiroyuki Akai, Akira Kunimatsu, Shigeru Kiryu, and Osamu Abe. Deep learning with convolutional neural network in radiology. *Japanese Journal of Radiology*, 2018.
 - [28] Shelly Soffer, Avi Ben-Cohen, Orit Shimon, Michal Marianne Amitai, Hayit Greenspan, and Eyal Klang. Convolutional neural networks for radiologic images: A radiologist’s guide. *Online <https://doi.org/10.1148/radiol.2018180547>*, ianuarie 2019.
 - [29] Pierre-Luc Carrier. Challenges in representation learning: facial expression recognition challenge. *AC*, 2013.

- [30] Lucey P. The extended cohn-kanade dataset(ck +): a complete dataset for action unit and emotion-specified expression. *IEEE computer society conference on computer vision and pattern recognition—workshops*, 2010.
- [31] Ma C. Wittenbrink. The chicago face database: a free stimulus set of faces and norming data. *Behav Res Methods* 1122–1135:47, 2015.
- [32] Lyons MJ, Akamatsu S, Kamachi M, and Gyoba J. Coding facial expressions with gabor wavelets. *3rd IEEE international conference on automatic face and gesture recognition*, 200–205, 1998.
- [33] Thomaz CE. Fei face database.
- [34] Setty S. Indian movie face database: A benchmark for face recognition under wide variations. *2013 fourth national conference on computer vision, pattern recognition, image processing and graphics (NCVPRIPG)*, 2013.
- [35] Li-Fen C. Facial expression image database. *Brain Mapping Laboratory, Institute of Brain Science, National Yang-Ming University, Taipei, Taiwan*, 2007.
- [36] Szegedy C. Going deeper with convolutions. *arXiv e-prints*, 2014.
- [37] Kate Strachnyi. Brief history of neural networks. *Analytics Vidhya, Medium*, ianuarie 2019.
- [38] Rutecki PA. Neuronal excitability: voltage-dependent currents and synaptic transmission. *Journal of Clinical Neurophysiology*, aprilie 1992.
- [39] R. Rojas. Neural networks: a systematic introduction. 1996.
- [40] R. Fuller. The perception learning rule - tutorial. *Online: <http://uni-obuda.hu/users/fuller.robert/perception.pdf>*, august 2011.
- [41] J. Brownlee. Loss and loss functions for training deep learning neural networks. *Machine Learning Mastery, Online: <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/>*, ianurie 2018.
- [42] B. Graham. Lecture 3: Delta rule. *Online: <http://www.cs.stir.ac.uk/courses/CSC9YF/lectures/ANN/3-DeltaRule.pdf>*, 2014.
- [43] Stanford University. Convolutional neural networks for visual recognition. *Online: <https://cs231n.github.io/convolutional-networks/>*.
- [44] Christoph Rasche. *Computer Vision*. 10 2019.
- [45] Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. 2012.
- [46] Erik Debill. Module counts. noiembrie 2019.
- [47] B. Ramsundar. Tensorflow tutorial. *Stanford Lectures Online: <https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf>*.

Anexa A

Codul sursă al scriptului de proiectare și antrenare a modelului

```
from __future__ import print_function
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
    BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt

import os

os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

# 5 clase - 'Angry' , 'Happy' , 'Neutral' , 'Sad' , 'Surprised'
nr_clase = 5
class_names = ['Angry', 'Happy', 'Neutral', 'Sad', 'Surprise']
img_rows, img_cols = 48, 48
batch_size = 32

#path-urile catre directoarele ce contin imaginile de test si de validare
train_data_dir = '/home/dani/Licenta/datasets/fer2013/train'
validation_data_dir = '/home/dani/Licenta/datasets/fer2013/validation'

# data augmentation pentru o baza de date mai mare

def augmentare():
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=30,
        shear_range=0.2,
```

```

        zoom_range=0.2,
        width_shift_range=0.3,
        height_shift_range=0.3,
        horizontal_flip=True,
        fill_mode='nearest')
    return train_datagen

def normare():
    validation_datagen = ImageDataGenerator(rescale=1. / 255)
    return validation_datagen

def generateTrainSet():
    train_datagen = augmentare()
    train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        color_mode='grayscale',
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True)
    return train_generator

def generateValidationSet():
    validation_datagen = normare()
    validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        color_mode='grayscale',
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True)
    return validation_generator

def buildModel():
    model = Sequential()

    # Bloc convolutional (1)

    model.add(Conv2D(32, (3, 3), padding='same', kernel_initializer='he_normal',
input_shape=(img_rows, img_cols, 1)))
    model.add(Activation('elu'))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), padding='same', kernel_initializer='he_normal',
input_shape=(img_rows, img_cols, 1)))
    model.add(Activation('elu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

```

Bloc convolutional (2)

```
model.add(Conv2D(64, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

Bloc convolutional (3)

```
model.add(Conv2D(128, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

Bloc convolutional (4)

```
model.add(Conv2D(256, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding='same', kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

Bloc flatten (5)

```
model.add(Flatten())
model.add(Dense(64, kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

Bloc fully connected (6)

```
model.add(Dense(64, kernel_initializer='he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

Bloc fully connected clasificator (7)

```
model.add(Dense(nr_clase, kernel_initializer='he_normal'))
model.add(Activation('softmax'))
```

```

return model

from tensorflow.keras.optimizers import RMSprop, SGD, Adam, Adadelta
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
    ReduceLROnPlateau

def generate_callbacks(modelName, patience_es, patience_lr):
    checkpoint = ModelCheckpoint(modelName,
                                monitor='val_loss',
                                mode='min',
                                save_best_only=True,
                                verbose=1)

    earlystop = EarlyStopping(monitor='val_loss',
                              min_delta=0,
                              patience=patience_es,
                              verbose=1,
                              restore_best_weights=True)

    reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                  factor=0.2,
                                  patience=patience_lr,
                                  verbose=1,
                                  min_delta=0.0001)

    callbacks = [earlystop, checkpoint, reduce_lr]

    return callbacks

def compileModel(model):
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(lr=0.001),
                  metrics=['accuracy'])

def antrenareModel(model, train_set, validation_set, nb_train_samples,
nb_validation_samples, epochs, callbacks):
    history = model.fit_generator(
        train_set,
        steps_per_epoch=nb_train_samples // batch_size,
        epochs=epochs,
        callbacks=callbacks,
        validation_data=validation_set,
        validation_steps=nb_validation_samples // batch_size)

    return history

def plotAccuracy(history):
    plt.plot(history.history['accuracy'])

```

```

plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

def plotLoss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

def printModelArchitecture(model, filename):
    plot_model(
        model,
        to_file=filename,
        show_shapes=True,
        show_layer_names=True,
        expand_nested=False,
        dpi=196,
    )

def main():
    nb_train_samples = 24256
    nb_validation_samples = 3006
    epochs = 25

    train_set = generateTrainSet()
    validation_set = generateValidationSet()
    callbacks = generate_callbacks('Emotion_train_model2.h5', 9, 3)

    print(emotion_model.summary())

    compileModel(emotion_model)

    history = antrenareModel(emotion_model,
                             train_set,
                             validation_set,
                             nb_train_samples,
                             nb_validation_samples,
                             epochs,
                             callbacks)

    plotAccuracy(history)
    plotLoss(history)
    printModelArchitecture(emotion_model, "model.png")

```



```
if __name__ == "__main__":
    main()
```

Codul sursă al aplicației de recunoaștere a emoției umane

```
from tensorflow.keras.models import load_model
from time import sleep
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing import image
import cv2
import numpy as np
import os

os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

emotion_classifier = load_model('/home/dani/PycharmProjects/Licenta/
    Emotion_train_model2.h5')
face_classifier = cv2.CascadeClassifier('/home/dani/Licenta/functions/
    haarcascade_frontalface_default.xml')

class_labels = ['Angry', 'Happy', 'Neutral', 'Sad', 'Surprise']

video_input = cv2.VideoCapture(0)

while True:
    # Luam doar un singur frame din inputul video
    ret, frame = video_input.read()
    labels = []
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray_frame, 1.3, 6)

    for (x, y, w, h) in faces:
        #Incadram cu un dreptunghi de culoare verde fata detectata in imagine
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        #Selectam regiunea dorita din imagine si o convertim in imagine gri de
        dimensiune 48x48
        roi_gray = cv2.cvtColor(frame[y:y+h,x:x+w], cv2.COLOR_BGR2GRAY)
        roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)

        #Daca se detecteaza vreo fata in inputul video, atunci o convertim in array
        pentru a putea fi clasificata
        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

            #Modelul face o predictie, iar in functie de predictia facuta se adauga in
            frame emotia recunoscuta

            preds = emotion_classifier.predict(roi)[0]
```

```
        label = class_labels[preds.argmax()]
        label_position = (x, y)
        cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 2,
(0, 255, 0), 3)
    else:
        cv2.putText(frame, 'Nu detectez nicio fata', (20, 60), cv2.
FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3)
    cv2.imshow('Emotion Detector', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_input.release()
cv2.destroyAllWindows()
```