UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Sistem de autentificare pe bază de recunoaştere facială

LUCRARE DE DIsertație

prezentată ca cerință parțială pentru obținerea titlului de Master în domeniul Inginerie Electronică şi Telecomunicaţii programul de studii de masterat BIOSINF

Profesor coordonator:
Conf. Dr. Ing. Horia CUCU

Student:
Ariton Adela-Cristina

București
2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Programul de masterat **Tehnologii multimedia în aplicaţii de biometrie şi securitatea informaţiei**

Anexa 2

TEMA LUCRĂRII DE DISERTAȚIE

a masterandului **ARITON P. Adela-Cristina , 421-BIOSINF**

1. Titlul temei: Sistem de autentificare pe bază de recunoaştere facială

2. Descrierea contribuţiei originale a masterandului (în afara părţii de documentare) și specificații de proiectare:
Proiectul presupune proiectarea şi implementarea unui sistem de autentificare pe bază de detecţie şi recunoaştere facială. Sistemul va utiliza un tip de reţea neurală profundă identificată de student (în mod experimental) ca fiind cea mai potrivită pentru această sarcină. Se vor varia: (i) topologia reţelei neurale, (ii) numărul de straturi, (iii) numărul de neuroni pe strat, (iv) tipul straturilor, (v) rata de antrenare şi eventual alţi hyper-parametri ai reţelei neurale. Sistemul va fi antrenat şi evaluat atât pe baza unor corpusuri de imagini publice, cât şi pe baza unor corpusuri proprii.

3. Resurse folosite la dezvoltarea proiectului:
Specializarea Deep Learning de pe Coursera (https://www.coursera.org/specializations/deep-learning), Python, TensorFlow, Keras, C++, Java, corpusuri de imagini adnotate (publice şi proprii).

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
TVRV, APDSV, TACAI, Proiect1, Proiect2

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

6. Data înregistrării temei: 2019-01-18 17:34:19

Conducător(i) lucrare,
Conf. dr. ing. HOCIUCU
semnătura : .....

Student,
semnătura : [redacted]

Responsabil program master,
Prof. dr. ing. Dragoş BURILEANU
semnătura : .....

Decan,
Prof. dr. ing. Cristian NEGRESCU
semnătura: .....

Cod Validare: **67922dc9e2**

DECLARAŢIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul "Sistem de autentificare pe bază de recunoaştere facială", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii „Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Master* in domeniul Inginerie Electronică şi Telecomunicaţii, programul de studii *BIOSINF,* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

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

Bucureşti, *2019* Ariton Adela-Cristina

CUPRINS

LISTĂ DE FIGURI

LISTĂ DE TABELE

Sistem de autentificare pe bază de recunoaştere facială

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIE

Încă din cele mai vechi timpuri, securitatea a fost și va continua să fie una dintre cele mai stringente probleme cu care se confruntă oamenii. Fie că vorbim despre protecția bunurilor, protecția datelor sau protecția persoanelor, putem observa progresele impresionante realizate de-a lungul timpului, cu precădere în ultimele două decenii, în ariile de cercetare și dezvoltare a mecanismelor, dispozitivelor și sistemelor de securitate. Din păcate însă, cu cât sistemele de securitate devin mai complexe și mai precise, cu atât metodele de „păcălire" a acestora devin mai ingenioase.

Direcția în care se concentrează atenția cercetătorilor din domeniul securității, în ultimii ani, este cea care apelează la biometrie în implementarea sistemelor de identificare și autentificare. Biometria este știința care studiază și măsoară acele trăsături ale corpului uman, care sunt unice pentru fiecare individ în parte (iris, trăsături ale feței, amprenta degetelor, amprenta palmei, configurația venelor, geometria mâinii,), precum și anumite semnale ce derivă din acestea (vorbire, semnătură/scris).

Într-o eră a automatizărilor și a lipsei tot mai mari a timplului, se dorește crearea de sisteme și dispozitive care să „gândească” și să ia decizii singure, cât mai precis și într-un timp cât mai scurt, pe baza unor „noțiuni învațate” apriori, fără a mai fi necesară intervenția umană, sau măcar ca intervenția să fie una minimală.

În acest sens, un alt domeniu care a luat amploare în ultimii ani este cel al inteligenței artificiale. Din ce în ce mai multe resurse umane și materiale se investesc în acest domeniu și deja o mulțime de aplicații au la baza algoritmi de machine learning sau rețele neuronale profunde.

1.2 OBIECTIVE

Lucrarea de față își propune atingerea următoarelor obiective:

- studiul noțiunilor de rețea neuronală; rețele neuronale convoluționale
- studiu în domeniul procesării de imagini
- cercetarea domeniului recunoașterii faciale
- familiarizarea cu platforma Tensor flow
- crearea unei rețele neuronale convoluționale simple
- variația hiper parametrilor rețelei și interpretarea rezultatelor
- antrenarea și testarea unei rețele neuronale pre antrenate existente, în scopul recunoașterii faciale
- ajustarea hiper parametrilor rețelei pentru obținerea rezultatelor optime
- alegerea rețelei ce a obținut rezultatele cele mai bune în sarcină de recunoaștere facială
- integrarea rețelei alese într-un sistem de autentificare bazat pe recunoaștere facială

1.3 STRUCTURA LUCRĂRII

Această lucrare poate fi împărțită în 4 părți principale, astfel:

1. Prima parte reprezintă partea teoretică, alcătuită din capitolele 2 și 3 ,în care se vor studia domeniul recunoașterii faciale și cel al învățării profunde, utilizând rețele neuronale profunde. Se vor explica succint cele mai importante noțiuni ce au fost necesare pentru realizarea acestei lucrări.
2. Partea a doua, corespunzătoare capitolului 4, este partea de creare a unei rețele neuronale de complexitate redusă, utilizată în scopul realizării sarcinii de recunoaștere pe o bază de date de imagini publice. Se vor realiza diverse experimente pentru a evalua performanța rețelei și se vor interpreta rezultatele.
3. A treia parte, corespunzătoare capitolului 5, are ca scop implementarea unui model de rețea deja existent și folosit în prezent, pentru majoritatea aplicațiilor de recunoaștere. Modelul se numește Inception-v3, de la GoogLe și deoarece este o rețea pre-antrenată permite implementarea facilă și exploatarea unor informații deja învățate, în scopul îmbunătățirii performanțelor unei sarcini sau pur și simplu pentru studiul comportamentului rețelei la variația anumitor parametri, acest lucru fiind unul dintre obiectivele acestei lucrări.

4. În ultima parte, reprezentată de capitolul 6, se va alege una dintre cele două rețele și se va încerca implementarea acesteia într-un sistem de autentificare bazat pe recunoaștere facială.

CAPITOLUL 2

REȚELE NEURONALE PROFUNDE

2.1 ASPECTE GENERALE

Cererea mare în aria autentificării computerizate a persoanelor, a dus la creșterea interesului spre domeniul biometriei, ca soluție pentru a înlocui parole ce pot fi aflate sau documente care pot fi furate ori falsificate cu ușurință.

Biometria este știința care se ocupă cu studiul și măsurarea unor trăsături ale corpului uman. Pentru a verifica identitatea cuiva, biometria utilizează trăsături precum cele ale irisului, retinei, amprentei digitale, ampretei palmare, feței, dinamicii semnăturii sau vocii. Față de metodele tradiționale de securitate, metodele ce apelează la biometrie au avantajul că nu pot fi furate, falsificate, „împrumutate” sau șterse cu ușurință.

Pentru a putea creea un sistem de autentificare utilizând trăsături biometrice, inetrvine un alt domeniu inovativ, și anume inteligența artificială. Se poate spune că inteligența reprezintă modul în care oamenii percep, interpretează, anticipează și utilizează informații pe care le primesc din exterior. Același lucru este urmărit și de domeniul inteligenței artificiale.

Inteligența artificială înglobează o mare varietate de sub-domenii, de la cele generale (învățare și percepție), la cele specifice (practicarea jocului de șah, demonstarea unor teoreme matematice, conducerea unei mașini pe o stradă aglomerată sau diagnosticarea unor boli).

Neuroștiința este știința care studiază sistemul nervos și în particular creierul. Totuși, modul exact în care acesta permite gândirea, încă este un mare mister pentru neuroștiință. Este deja bine cunoscut faptul că anumite porțiuni localizate din creierul uman sunt responsabile pentru funcții cognitive specifice, iar creierul este alcătuit din celule nervoase numite neuroni.

În prezent, există informații despre maparea între anumite zone ale creierului și părțile corpului pe care le controlează sau de la care primesc stimuli senzorial. Cu toate acestea, încă nu este foarte clar cum alte zone pot prelua alte funcții atunci cand o zonă a fost distrusă din punct de vedere neurologic.

Una din concluziile la care s-a ajuns este aceea că o colecție de simple celule poate duce la gânduri, acțiuni și conștiință.

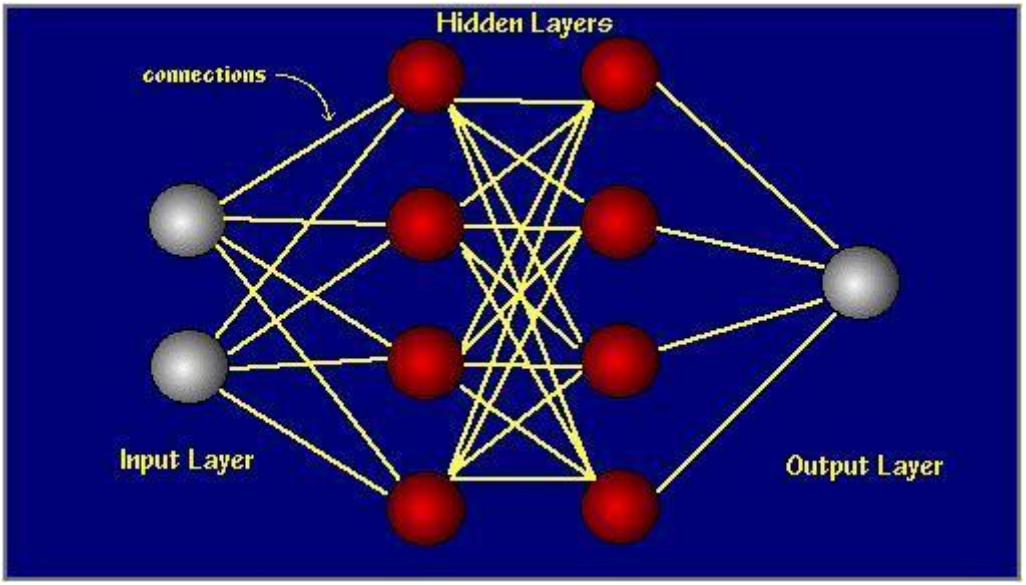
	Calculator	Creierul uman
Unități de calcul	1 CPU, 10^8 porți	10^{11} neuroni
Unități de stocare	10^{10} biți RAM, 10^{12} biți disk	10^{11} neuroni, 10^{14} sinapse
Ciclu de timp	10^{-9} s	10^{-3} s
Bandă	10^9 biți/s	10^{14} biți/s
Actualizări ale neuronului/s	10^6	10^{14}

Tabel 2.1 - Comparație între creierul uman și calculator

2.2 CE ESTE O REȚEA NEURONALĂ PROFUNDĂ?

Dr. Robert Hecht-Nielsen, inventatorul unuia dintre primele neurocalculatoare, a definit rețeaua neuronală ca fiind: „un sistem computațional alcătuit dintr-un număr de elemente de procesare simple, intens interconectate, care procesează informația prin răspunsul dinamic la stimuli externi”..

Rețelele neuronale sunt, de obicei, organizate în straturi. Straturile sunt alcătuite dintr-un număr de „noduri” interconectate care conțin o „funcție de activare”. Modelele sunt prezentate rețelei prin intermediul stratului de intrare, care comunică cu unul sau mai multe straturi ascunse, unde are loc procesarea propriu-zisă, printr-un sistem de conexiuni ponderate. Atunci, straturile ascunse, duc către stratul de ieșire, unde răspunsul este scos, ca în Figura 2.1.

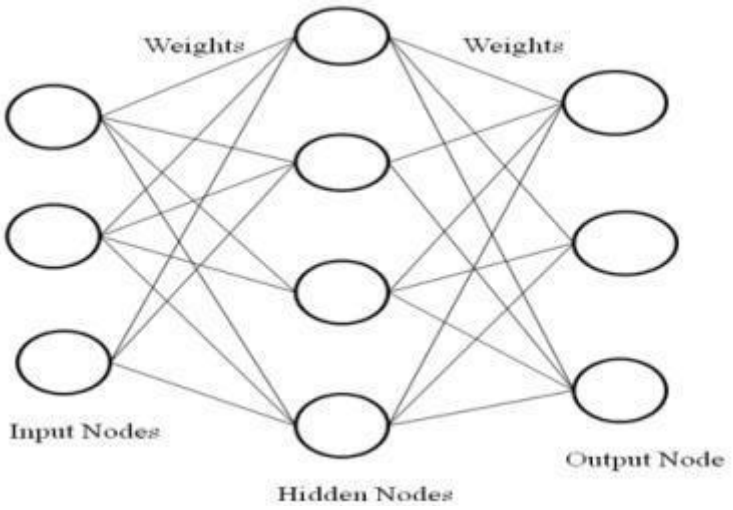


Figură 2.1 - Reprezentarea unei rețele neuronale 1 [2]

Cele mai multe dintre rețelele neuronale conțin o regulă de învățare, care modifică ponderile conexiunilor în concordanță cu modelele de intrare. Într-un fel, rețelele neuronale artificiale învață din exemple, așa cum fac și copiii. Un copil învață să recunoască o floare din exemple de flori.

Rețelele neuronale sunt utilizate pentru capabilitatea lor de a clasifica, acestea realizând predicții atât pentru date cunoscute cât și pentru date necunoscute. Funcționează bine atât pentru seturi de date separate liniar cât și ne-liniar.

Neuronii de intrare iau informația, sub forma unor expresii numerice. Informația este prezentată ca valori de activare, fiecarui nod i se atribuie apoi un număr (cu cât mai mare numărul, cu atât mai mare activarea). Această informație este apoi trimisă în rețea. Bazat pe legăturile dintre neuroni (caracterizate prin ponderi) inhibare sau excitare și funcții de transfer, valoarea de activare este trimisă de la nod la nod. Fiecare nod adună valoarea de activare pe care o primește și modifică apoi valoarea bazat pe funcția lui de transfer. Activarea trece prin rețea, prin straturile ascunse, până când ajunge la neuronii de ieșire.

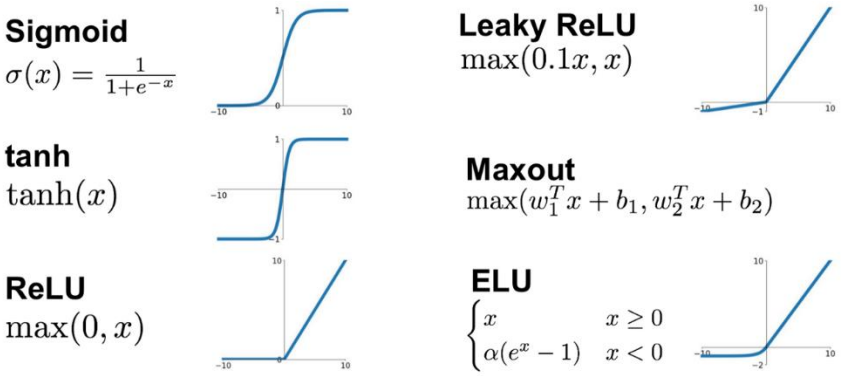


Figură 2.2 - Reprezentarea unei rețele neuronale 2

2.2.1 *Funcţia de activare*

Funcţia de activare este transformarea non-liniară care se aplică unui semnal de intrare. Această ieşire transformată este apoi trimisă către următorul strat de neuroni ca intrare. Este un nod care se pune de obicei la finalul sau între straturile unei reţele neuronale. Această funcţie decide dacă un neuron se va activa sau nu.

Exemplu de funcţii de activare:



Figură 2.3 - Funcţii de activare

Printre cele mai utilizate funcţii de activare, este funcţia ReLU (Rectified Linear Unit). Unul dintre cele mai mari avantaje pe care această funcţie le are peste celelalte funcţii de activare este că nu activează toţi neuronii în acelaşi timp. După cum se poate observa şi în figura de mai sus, funcţia ReLU converteşte toate intrările negative la zero şi neuronul nu se activează. Acest lucru o face eficientă din punct de vedere computational. În practică, ReLU converge de peste cinci ori mai repede decât funcţiile de activare tanh şi sigmoid.

Unul dintre dezavantajele acestei funcţii este că gradientul în zona negativă este zero. Cu gradientul egal cu zero, în timpul propagării înapoi, ponderile nu vor fi ajustate, de aceea uneori se foloseşte Leaky ReLU.

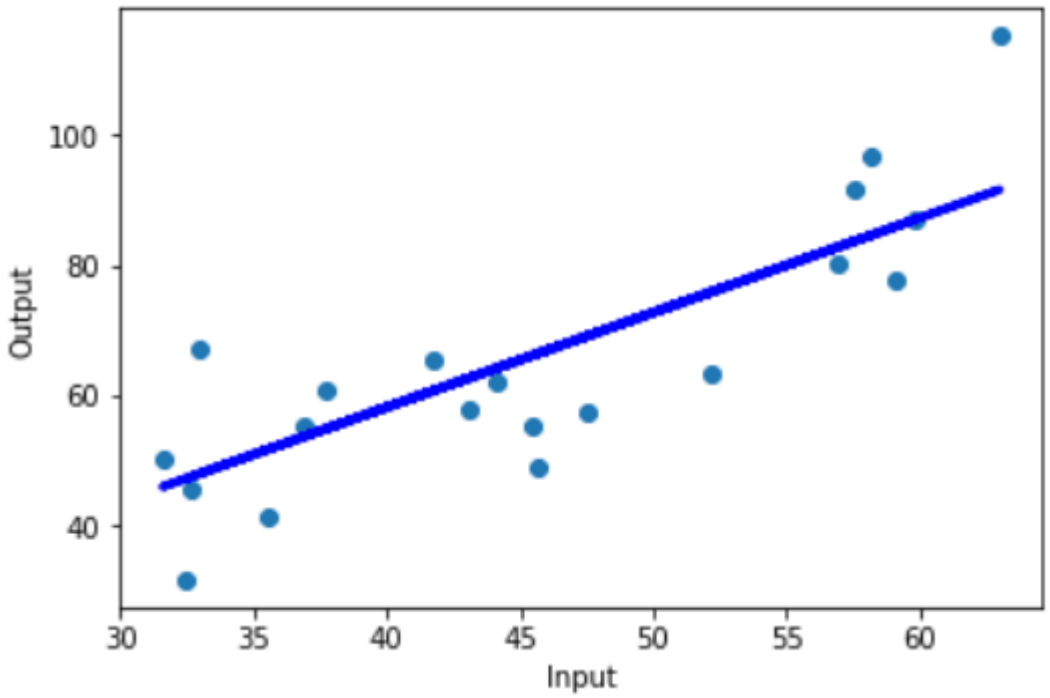
2.2.2 *Regresia liniară simplă*

Regresia liniară simplă este o metodă statistică ce permite studiul relației dintre două variabile continue. Aceasta urmărește relația statistică dintre cele două variabile, și nu pe cea deterministă. Relația dintre cele două variabile ar putea fi numită deterministă, dacă o variabilă ar putea fi exprimată cu precizie prin intermediul celeilalte.

O variabilă, notată cu “x”, este privită ca *predictorul* sau variabila independentă. Cealaltă variabilă, notată cu “y”, este privită ca *răspunsul* sau variabila dependentă.

Regresia liniară simplă, se numește “simplă”, deoarece vizează studiul unei singure variabile predictor. Intuitiv, regresia liniară multiplă vizează studiul a două sau mai multe variabile predictor.

Ideea principală este obținerea unei linii care încadrează cât mai bine datele. Se consideră cea mai bună astfel de linie, cea pentru care eroarea totală de predicție este cât mai mică posibil. Eroarea este distanța dintre punct și linia de regresie. În figura următoare este reprezentat un exemplu de regresie liniară.



Figură 2.4 - Linia regresiei finală pe datele de test [15]

Pentru a prezice răspunsul y_i, se folosește formula

$$\hat{y}_i = b_0 + b_1 x_i ,$$

în urma căreia se produce o eroare de predicție ce se calculează astfel:

$$e_i = y_i - \hat{y}_i$$

O linie care încadrează cel mai bine datele va fi una pentru care cele n erori de predicție (una pentru fiecare punct observat), cât mai mici posibil per total. O modalitate de a atinge acest obiectiv este prin minimizarea sumei erorilor pătratice de predicție, adică găsirea valorilor b_0 și b_1 care minimizează

$$Q = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 .$$

2.2.3 Loss function

Loss function (funcția pierderilor) oferă mai mult decât o reprezentare statică a performanței modelului, aceasta indicând și cum algoritmii utilizați încadrează datele. Majoritatea algoritmilor de machine learning utilizează un fel de loss function în procesul de optimizare sau găsirea celor mai bune ponderi pentru datele utilizate.

Ca un exemplu simplu, putem considera regresia liniară. În regresia liniară a “celor mai mici pătrate”, linia celei mai bune încadrări este determinată prin “Eroarea pătratică medie”. Pentru fiecare set de ponderi pe care modelul îl încearcă, eroarea pătratică medie este calculată peste toate exemplele de intrare. Atunci modelul optimizează funcțiile eroare pătratică medie, sau altfel spus le minimizează, prin intermediul unuialgoritm de optimizare, precum “Gradient Descent”.

2.2.4 Gradient Descent

Gradient Descent este de departe cea mai utilizată strategie de optimizare utilizată în machine learning și învățarea profundă în acest moment. Este utilizat când se antrenează modelele, poate fi

combinat cu orice algoritm și este ușor de înțeles și de implementat. Acesta se bazează pe o funcție convexă și își ajustează iterativ parametrii pentru a minimiza o funcție dată către minimul ei local.

Gradient descent este utilizat pur și simplu pentru a găsi valorile parametrilor unei funcții (coeficienții) care minimizează funcția de cost cât de mult posibil.

Se începe prin definirea valorilor inițiale ale parametrilor și de acolo, gradientul utilizează calculul pentru a ajusta valorile în scopul minimizării funcției de cost.

“Gradientul măsoară cât de mult se schimbă ieșirea unei funcții dacă se variază ușor intrările” – Lex Fridman (MIT). Ne putem gandi la gradient și ca la panta unei funcții. Cu cât este mai mare gradientul, cu atât mai abruptă este panta și cu atît mai rapid poate învăța modelul. Însă dacă panta este zero, modelul încetează să mai învețe. În termeni matematici, gradientul este o derivată parțială în ceea ce privește intrările sale.

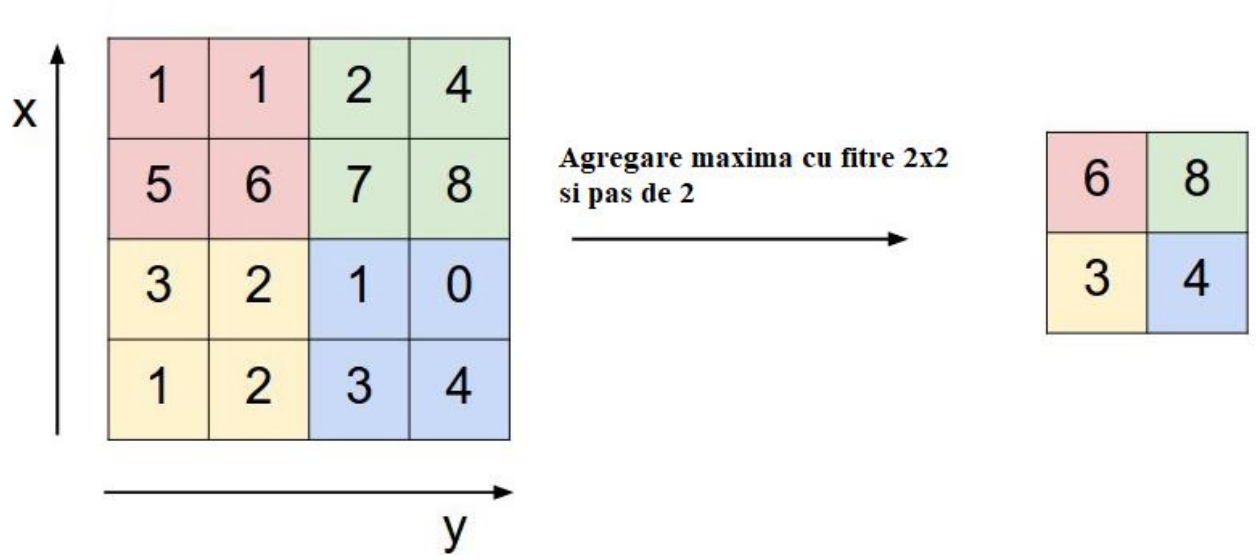
2.2.5 Pooling layer

Stratul de agregare (sau pooling) se poate utiliza între straturile convolutionale are unei reţele neuronale convolutionale. Acest strat reduce numărul de parametri din reţea, controlând overfitting-ul prin reducerea progresivă a dimensiunii spaţiului reţelei.

Au loc două operaţii în acest strat, agregare medie şi agregare maximă.

Agregarea maximă (sau max pooling) după cum indică şi numele va selecta doar valoarea maximă. Acest lucru se realizează prin folosirea unor filtre (ferestre) ce parcurg datele de intrare, şi la fiecare pas, doar valoarea maximă este aleasă, practic se sub-esantioneaza intrearea.

Spre deosebire de stratul de convoluţie, stratul de agregare nu alterează adâncimea reţelei.



Figură 2.5 - Agregare maximă (max pooling)

Straturile dense (fully connected)

În aceste straturi, toţi neuronii sunt conectaţi la toate activările din stratul anterior. Activările lor pot fi astfel calculate cu o înmulţire de matrici urmată de un offset. Aceasta sunt prezente la finalul unei reţele neuronale convolutionale.

2.2.6 Optimizatorul Adam

Este un algoritm de optimizare care poate fi folosit în loc de procedură stohastica de scădere a gradientului pentru a ajusta ponderile reţelei într-un mod iterativ. Beneficiile utilizării acestui algoritm:

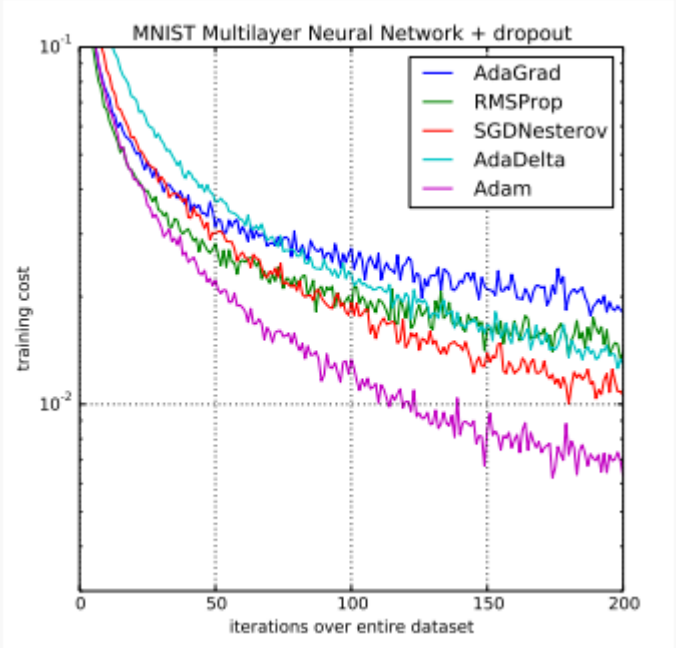
- Uşor de implementat
- Eficient din punct de vedere computational
- Nu necesită multă memorie
- Invariant la re-scalarea diagonală a gradienţilor
- Este foarte potrivit pentru probleme care sunt mari din punct de vedere al datelor sau parametrilor
- Potrivit pentru obiective non-stationare
- Hiper parametrii au o interpretare intuitivă şi de obicei este nevoie de ajustări minore

Din aceste motive este un algoritm popular în probleme de computer vision.

Algoritmii stohastici menţin o singură rată de învăţare pentru toate ajustările de ponderi şi nu se schimbă în timpul antrenării.

Pentru algoritmul de optimizare Adam se menţine o rată de învăţare pentru fiecare pondere din reţea şi separat se adaptează în timpul învăţării.

În imaginea de mai jos se poate observa o comparaţie cu alţi algoritmi de optimizare pentru antrenarea unui perceptron mulţi strat [Adam: A Method for Stochastic Optimization, 2015]:

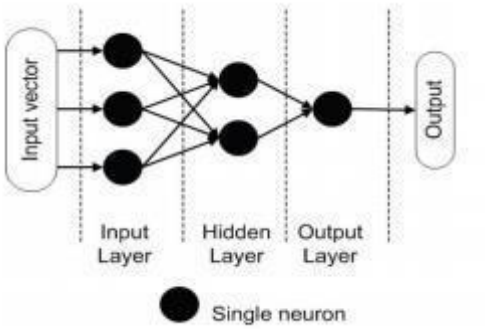


Figură 2.6 -Comparaţie algoritm Adam [14]

2.3 TOPOLOGII UZUALE PENTRU REȚELE NEURONALE:

Rețele feed forward

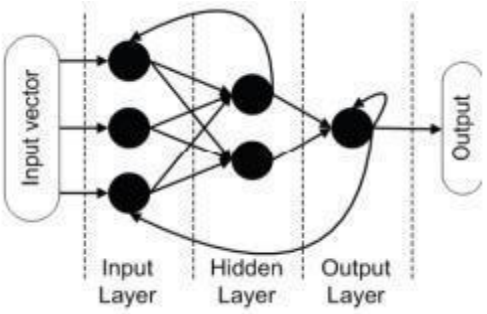
O rețea feed forward este o rețea non-recurentă care conține intrări, ieșiri și straturi ascunse, semnalele pot trece într-o singură direcție. Datele de intrare sunt trimise într-un strat de elemente de procesare care realizează calcule. Fiecare element de procesare își face calculele bazat pe o sumă ponderată a intrării. Valoarea nou calculată devine apoi noua intrare pentru următorul strat. Acest proces continuă până când trece prin toate straturile și determină ieșirea. De obicei se folosește o funcție de transfer de prag pentru a cuantifica ieșirea unui neuron în stratul de ieșire.



Figură 2.7 – Rețea feed forward

În figura de mai sus se poate observa o topologie de rețea neuronală feed forward simplă, denumită și graf aciclic unde informația trece de la intrare la iesire într-o singură direcție.

Rețele recurente



Figură 2.8 – Rețea recurentă

În imaginea de mai sus se poate observa o rețea neuronală recurentă, denumită și graf aciclic, unde informația nu trece într-o singură direcție de la intrare la ieșire, dar și în direcția opusă. Rețelele neuronale artificiale cu asemenea topologie se numesc rețele neuronale recurente, sunt similare cu FNN dar fără limitări în ceea ce privește bucle inverse. În acest caz informația nu este transmisă într-o singură direcție dar este transmisă și înapoi (spre intrarea rețelei). Acest lucru crează o stare interană rețelei care îi permite să aibă un comportament dinamic în timp. Rețelele neuronale recurente își pot folosi memoria internă pentru a procesa orice secvențe de date de intrare.

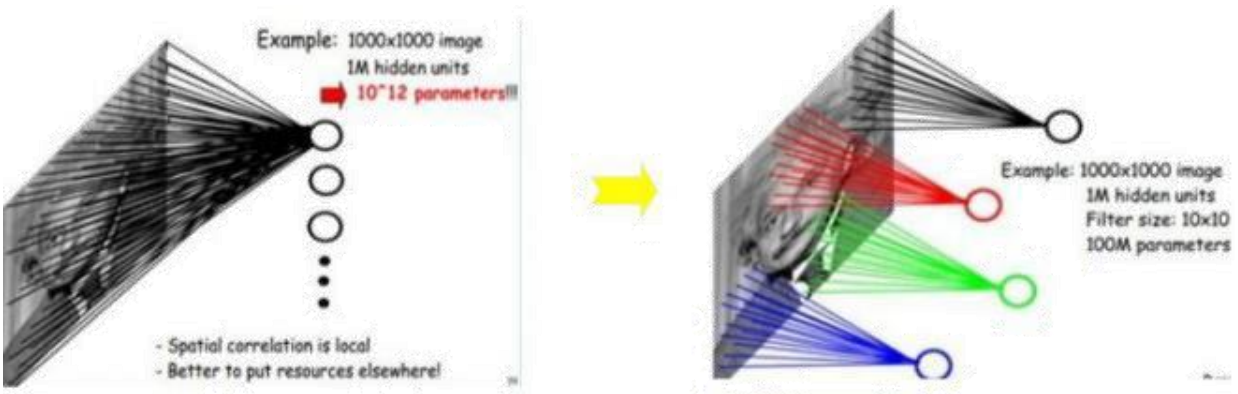
Rețelele neuronale sunt adesea utilizate pentru analiză statistică și modelare de date, unde rolul lor este perceput ca o alternativă la regresia nonliniară standard sau la tehnicile de analiză de tip cluster. Totuși, ele sunt utilizate în probleme care pot fi supervizate în termeni de clasificare sau predicție. Câteva exemple includ recunoașterea de imagini, recunoașterea vorbirii, recunoașterea caracterelor textuale, diagnoza madicală sau predicția indicatorilor piețelor financiare. Acest tip de probleme intră de asemenea în domeniul inteligenței artificiale clasice pentru ca inginerii și programatorii să poată vedea rețelele neuronale ca oferind un fel de calcul distribuit paralel, dar aducând o alternativă la tehnicile algoritmice convenționale, care au predominat în inteligența mașinilor. Termenul de paralelism se referă aici, la faptul că fiecare nod este conceput ca operator independent și concurent (în paralel) cu celelalte, iar informația în rețea este distribuită peste întreg setul de ponderi mai degrabă, decât focusată în câteva locații de memorie, ca în calculatoarele convenționale.

2.4 REȚELE CONVOLUȚIONALE

Deși rețelele neuronale pot fi aplicate în sarcini de computer vision, pentru a obține o performanță a generalizării bună, este benefic ca în arhitectura rețelei să fie introdusă informația cunoscută apriori. Rețelele neuronale convoluționale în recunoașterea de imagini, țintesc utilizarea informației spațiale dintre pixelii unei imagini. Mai mult, acestea se bazează pe convoluția discretă.

Algoritmul rețelei neuronale convoluționale este un perceptron multistrat, care este conceput special pentru a identifica informația unei imagini bi-dimensionale. Are întotdeauna mai multe straturi: startul de intrare, stratul de convoluție, stratul de eșantionare și stratul de ieșire.

Rețeaua neuronală convoluțională nu este ca o mașină boltzmann restrictivă, adică să aibă nevoie să fie înainte și după stratul neuronilor din stratul adiacent la toate conexiunile, în algoritmii rețelelor neuronale convoluționale, fiecare neuron neavând nevoie să „simtă” imaginea globală, ci doar aria locală a imaginii. În plus, fiecare parametru al neuronului este setat la aceeași atribuire de ponderi, și anume fiecare neuron cu aceleași nuclee la deconvoluția imaginii.

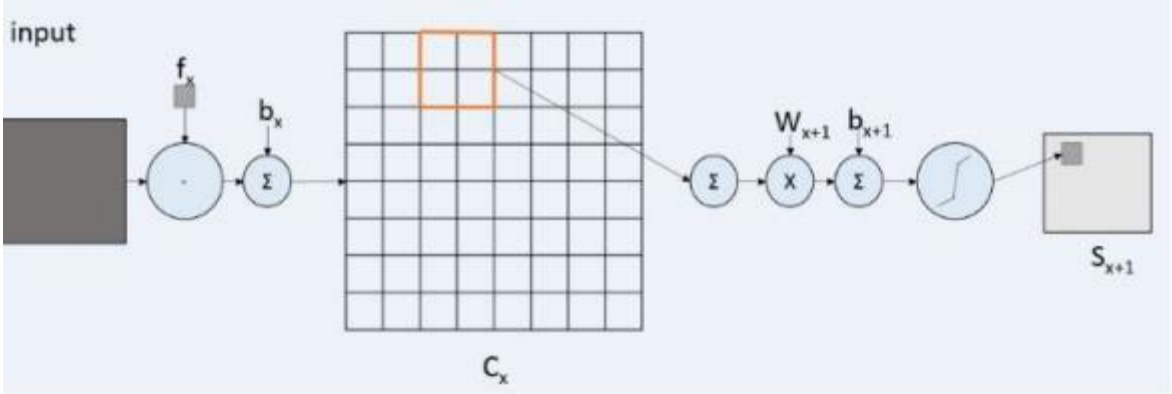


Figură 2.9 - Conexiune totală versus conexiune parțială

Algoritmul rețelei neuronale convoluționale are două procese principale: convoluție și eșantionare.

Procesul de convoluție: utilizează un filtru antrenabil F_x, deconvoluția imaginii de intrare, la care adunând b_x, se poate obține convoluția stratului Cx.

Procesul de eșantionare: n pixeli din fiecare vecinătate devin un pixel, și apoi prin ponderare scalară (W_x + 1) ponderat, la care se adună b_x + 1, iar apoi printr-o funcție de activare rezultă o reducere de n ori hărții trăsăturilor S_{x+1}.



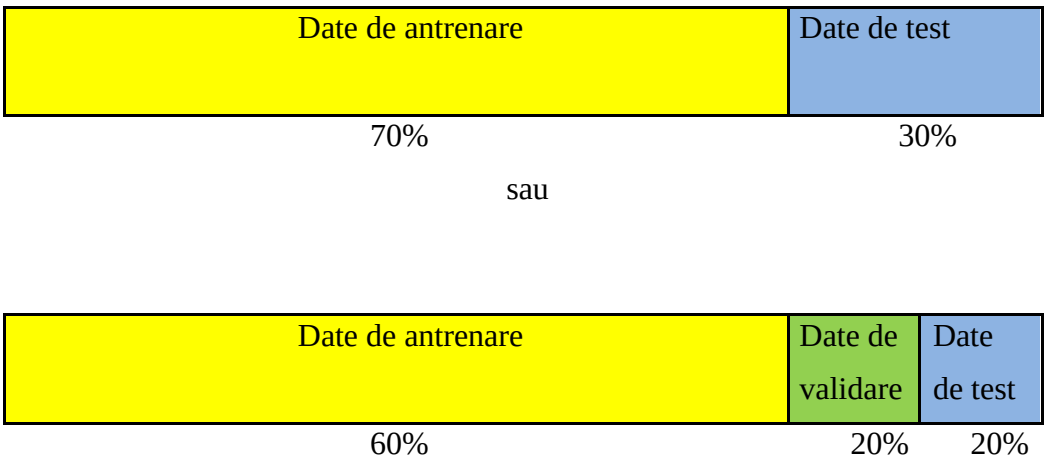
Figură 2.10 - Procesul principal al unei rețele neuronale convoluționale

Tehnologia-cheie a rețelelor neuronale convoluționale este câmpul receptiv local, împărțirea ponderilor, sub eșantionarea în timp sau spațiu , pentru a extrage trăsături și a reduce dimensiunea parametrilor de anternare. Avantajul algoritmilor rețelelor de acest tip este acela că evită extragerea explicită a trăsăturilor și învață implicit din datele de antrenare. Rețeaua poate învăța în paralel, reducându-și astfel complexitatea.

2.5 ÎMPĂRȚIREA SETULUI DE DATE

Pentru crearea unei aplicații ce are la bază crearea unei rețele neuronale profunde, este foarte importantă alegerea seturilor de date, în special al setului de antrenare.

Pentru aplicații unde se dorește obținerea unei acurateți ridicate, este necesar ca aceasta să dispună de cel puțin 10 000 de date. Un exemplu în care aceste date pot fi distribuite, este următorul:



Figură 2.11 – Exemple de împărțire a setului de date

Atunci când se realizează pentru prima data o astfel de aplicație este necesară „ghicirea” următorilor parametri:

- numărul de straturi al rețelei
- numărul de straturi ascunse
- rata de învățare
- funcțiile de activare
- și altele…

CAPITOLUL 3

RECUNOAȘTEREA FACIALĂ

3.1 ASPECTE GENERALE

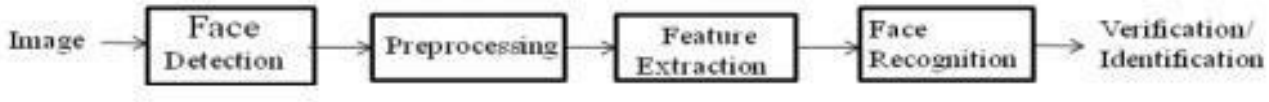
Recunoașterea facială este un domeniu de cercetare foarte popular în computer vision sau în aria de pattern recognition datorită variației expresiilor faciale, posturilor și iluminării, pe scurt, datorită variabilității datelor. Câteva aplicații, de la cele guvernamentale până la cele comerciale, cer industriei să dezvolte sisteme eficiente de recunoaștere automată a fetelor. Deși, mulți cercetători au lucrat în domeniul recunoașterii faciale pentru mulți ani, încă sunt câteva provocări care necesită rezolvare. Diferența în luminozitate a scenei, schimbarea posturii, orientarea, expresia feței sunt doar câteva exemple ale problemelor întâlnite. De asemenea, când baza de date de fețe crește, timpul de recunoaștere devine un aspect important. Recunoașterea feței este una din metodele biometrice care au atât acuratețe ridicată cât și instrusivitate scăzută. Din această cauză, recunoașterea facială a atras atenția cercetătorilor în domenii de la securitate, psihologie și procesare de imagini până la computer vision. Mulți algoritmi au fost propuși pentru recunoașterea facială. Recunoașterea facială analizează caracteristicile feței unei persoane dintr-o imagine dintr-un feed video sau o captură foto.

În această lucrare se vor prezenta diferitele tehnici de contrucție a unor tipuri de rețele neuronale care au fost propuse de mai mulți cercetători în sistemele de recunoaștere facială. În ultimele decenii s-a observat că rețelele neuronale artificiale au fost folosite în mai multe domenii precum pattern recognition, procesare de imagini, diagnoză, etc. Rețelele neuronale artificiale atât predictori cât și modele non liniare dinamice și clasificatori de pattern-uri, au fost sugerate ca posibilă tehnică pentru recunoașterea facială.

3.2 STRUCTURA UNUI SISTEM DE RECUNOAȘTERE FACIALĂ

Orice sistem biometric are patru caracteristici principale care sunt evidențiate în imaginea de mai jos:

- Detecția feței
- Pre-procesarea
- Extragerea caracteristicilor
- Recunoasșerea feței



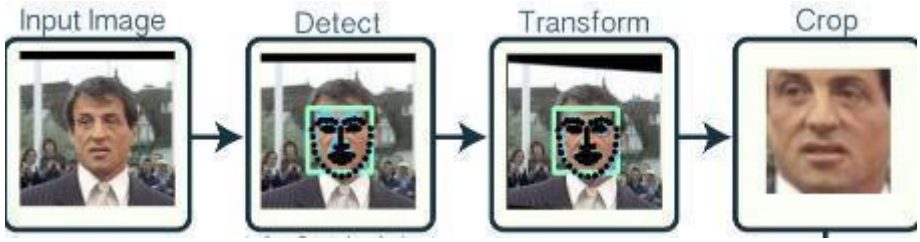
Figură 3.1 - Structura unui sistem de recunoaștere facială

După cum se poate observa în imaginea de mai sus, primul task al sistemului este de a captura imaginea, după care aceasta este trimisă către blocul de detecție al feței.

3.2.1 Detecția feței

Principala funcție a acestui bloc este de a detecta fața dintr-o imagine. Acest proces de detecție verifică dacă imaginea prezentată conține o față sau nu, după detecția feței imaginea este trimisă către blocul de pre-procesare.

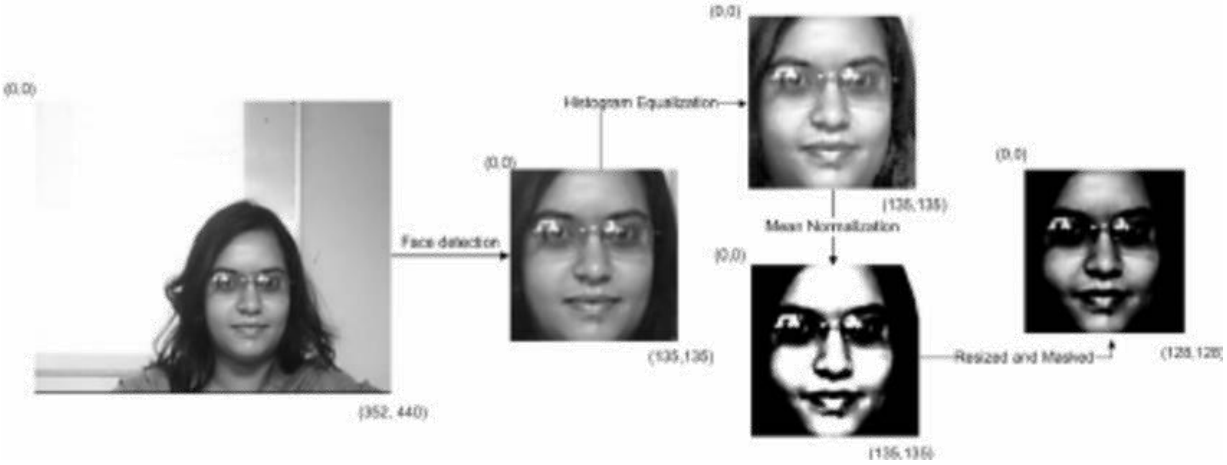
Există o serie de algorimti folosiți pentru detecția feței, majoritatea folosesc segmentarea imaginii pe bază de culoare în combinație cu un sistem de pattern recognition. S-a dovedit eficientă utilizarea rețelelor neuronale adânci chiar și pentru acest tip de sarcină. Majoritatea algoritmilor încearcă separarea caracteristicilor esențiale folosind filtre de procesare de imagini pentru extragerea conturuilor, se caută apoi pattern-uri bazate pe distanțe între segmente pentru a se detecta poziția trăsăturilor esențiale ale feței – poziția ochilor, a nasului și a gurii. După ce toate aceste aspecte au fost determinate, imaginea este decupată pentru a elimina informațiile inutile. Astfel, în blocul de preprocesare ajunge imaginea care conține fața.



Figură 3.2 – Detecția și decuparea feței

3.2.2 Pre procesarea

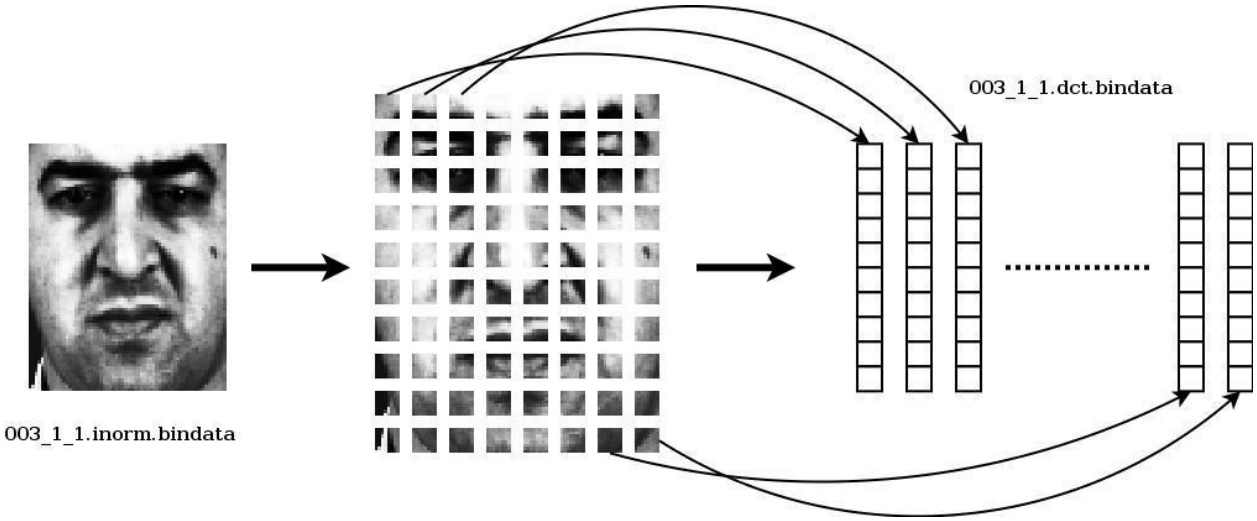
În acest pas, se realizează îmbunătățirea imaginii, pentru a evidenția cât mai bine caracteristicile cele mai importante din imagine, astfel se încearcă eliminarea sau reducerea zgomotului din imagine, se încearcă egalizarea histogramei, eliminarea efectelor de umbrire, etc. Imaginea rezultată va reprezenta un input mult mai bun pentru blocul de extracție a trăsăturilor.



Figură 3.3 – Preprocesarea imaginii

3.2.3 Extragerea trăsăturilor

În acest pas, imaginea procesată este folosită pentru a extrage principalele trăsături faciale prin metode algoritmice. Extracția realizează reducerea cantității de informație necesară, prin cuantificare și vectorizare. Astfel, la ieșirea acestui bloc, imaginea se transformă de obicei într-un vector.



Figură 3.4 – Extragere trăsături

3.2.4 Recunoașterea feței

Odată ce extracția trăsăturilor a fost realizată, se trece în pasul de recunoaștere. În acest pas se realizează o comparație între vectorul de trăsături al imaginii de la intrare și o serie de valori dintr-un model antrenat cu o rețea neuronală. Astfel obiectul cel mai apropiat de acest vector va fi considerat obiectul recunoscut (în funcție de un anumit prag).

Recunoașterea facială poate fi clasificată în două mari clase:

- Metoda bazată pe trăsături locale
- Metoda bazată pe trăsături globale

Fețele umane pot fi caracterizate atât pe o bază locală cât și pe un set de trăsături globale. Trăsăturile globale snt mai usor de capturat dar sunt în general mai puțin discriminative decât trăsăturile locale. În zilele noastre, sistemele de recunoaștere facială, recunosc fețe folosind mai multe imagini ale feței.

Metode de recunoaștere pe bază de trăsături

1. ***EigenFaces:***
   Eigenfaces provine de la vectorii eigen din matematică. Este pur și simplu o aplicație a Analizei Componentelor Principale (PCA) pe o colecție de fețe, pentru a reduce dimensionalitatea reprezentărilor feței. EigenFaces poate fi combinat liniar să aproximeze orice față dată.
2. ***Graph matching:***
   În lucrarea "Face Recognition by Elastic Bunch Graph Matching", un set de filtre Gabor este folosit pentru a construi graful imagine a unei fețe și recunoașterea feței se realizează print-o "potrivire" a grafurilor.

3. ***Rețele neuronale:***
   Recunoașterea de fețe utilizând rețele neuronale poate varia de la procesarea întregii fețe, până la procesarea bazată pe repere faciale. Abordarea recunoașterii întregii fețe presupune deținerea unei cantități mari de imagini diferite cu fețe ale unui individ. În abordarea bazată pe repere faciale, rețelele neuronale detectoare sunt antrenate pe repere faciale precum: ochiul drept, ochiul stâng, etc. Iar detecția finală sau recunoașterea este bazată în mare parte pe relația geometrică dintre repere.
4. ***Învățare profundă:***
   Aria învățării profunde implică învățarea unor trăsături mai abstracte și mai bogate din setul de dae de antrenare înainte de a folosi clasificatorul final. Rețelele convoluționale sunt state-of-the-art în domenii precum sisteme de recunoaștere de obiecte la nivel de categorie, inclusiv sisteme de recunoaștere facială. Publicații recente precum "FaceNet" de la Google și "DeepFace" de la Facebook arată cum este învățarea profundă angajată în recunoașterea facială astăzi.

3D based: Tehnicile 3D implică modelarea 3D a unei fețe din una sau mai multe expuneri foto ale feței. În mod ideal se cere să se obțină o reprezentare dintr-o singură expunere , ca în cazul "DeepFace". Rutinele de procesare ulterioare pot procesa apoi fața dintr-o reprezentare canonică, ceea ce ajută sistemul de recunoaștere să managerieze variații majore ale punctului de observație.

CAPITOLUL 4

CREAREA UNEI REȚELE NEURONALE DE COMPLEXITATE REDUSĂ

4.1 TENSORFLOW

Învățarea profundă este un set de algoritmi inspirați de structura și funcțiile creierului uman, reprezentând un subdomeniu al machine learning-ului.

Tensorflow este al doilea framework de machine learning pe care Google l-a creat și utilizat pentru a proiecta, construe și antrena modele de învățare profundă. Librăria TensorFlow poate fi utilizată pemtru a face calculi numerice, care sunt realizate cu grafuri ale flow-ului de date, în care nodurile reprezintă operații matematice, în timp ce edge-urile reprezintă datele, care sunt de obicei șiruri de date multidimensionale sau tensori, care sunt transmise între aceste edge-

uri. De aici rezultă că numele, “TensorFlow” derivă din operații pe care rețelele neuronale le execută asupra șirurilor de date multidimensionale sau tensorilor.

Pentru a înțelege bine tensorii sunt necesare cunoștințe minime de algebră liniară și calcul vectorial.

Vectori plani

Vectorii pot fi priviți ca niște tipuri speciale de matrici, fiind niște șiruri rectangulare de numere, sau altfel spus, niște colecții ordonate de numere ce sunt văzute deseori ca matrici coloană. Diferența principală între un scalar și un vector este dată de direcție. Lungimea unui vector matematic este un număr, o valoare absolută, în timp ce direcția este relativă, ea fiind măsurată relativ la o direcție de referință , în radiani sau grade.

Vectorii plani sunt cea mai clară construcție de tensori. Aceștia seamănă cu vectorii descriși mai devreme, singura diferență fiind aceea că vectorii plani se găsesc într-un spațiu vectorial.

Tensori

Alături de vectorii plani, co-vectorii și opratorii liniari sunt alte două cazuri care au un lucru în comun: sunt cazuri specifice de tensori. Dacă un vector poate fi caracacterizat ca magnitudini scalare cărora li s-au atribuit o direcție, un tensor estereprezentarea matematică a unei entități fizice ce poate fi caracterizată prin magnitudine și mai multe direcții. Așa cum un scalar poate fi reprezentat printr-un singur număr, iar un vector cu o secvență de 3 numere, într-un spațiu tridimensional, un tensor poate fi reprezentat printr-un șir de 3^R numere, într-un spațiu tridimensional, unde R reprezintă rank-ul tensorului. Mai general, într-un spațiu N-dimensional, un scalar poate fi reprezentat printr-un singur număr, un vector prin N numere, iar un tensor prin N^R numere.

Instalarea TensorFlow-ului

TensorFlow este o librărie open-source intens utilizată în ultimul timp, care include și o librărie matematică destinată programării în aplicații de machine learning și rețele neuronale.

TensorFlow asigură API-uri pentru Python, C++, Java, etc. Pentru a descărca o versiune de TensorFlow care să permită scrierea codului în Python, pe pagina de descărcare https://www.tensorflow.org/install/ se găsesc instrucțiuni pentru a instala TensorFlow utilizând pip, virtualenv sau Docker. Pentru Windows, se poate instala și cu Conda, însă cel mai sigur mod de instalare este prin vizitarea paginii oficiale cu instrucțiuni de intalare https://www.tensorflow.org/install/install_windows .

După ce a fost instalat, este recomandată verificarea că TensorFlow s-a instalat corect. Acest lucru se poate face prin importarea acestuia în spațiul de lucru, sub alias-ul tf (alias utilizat de majoritatea utilizatorilor):

```
import tensorflow as tf
```

Odată instalat, se poate începe scrierea programelor TensorFlow, ce vor fi rulate, în general, ca blocuri. Se poate utiliza însă și Sesiunea Interactivă a TensorFlow-ului, dacă se dorește lucrul mai interactiv cu librăria.

4.2 STRUCTURA REȚELEI

Modelul a fost creat folosind TensorFlow (prezentat în capitolul 4.1) și s-a încercat crearea unei rețele neuronale relativ simple, pentru a putea realiza o comparație cu modelul Inception-v3. Spre deosebire de Inception, se vor antrena toate straturile, nu doar ultimul.

Rețelele neuronale convoluționale sunt în momentul de față state-of-the-art ca arhitectură de model pentru sarcini legate de clasificare bazată pe imagini. CNN-urile aplică o serie de filtre peste datele pixelilor unei imagini pentru a extrage și învăța trăsături de nivel înalt, pe care modelul le va folosi apoi pentru clasificare. CNN-urile conțin 3 componente:

- Straturi convoluționale, care aplică un număr specificat de filtre convoluționale unei imagini. Pentru fiecare sub-regiune a imaginii stratul realizează un set de operații matematice pentru a ajunge la o singură valoare în harta de trăsături de la ieșire. Straturile convoluționale aplică apoi o funcție de activare, de obicei ReLU, la ieșire pentru a introduce în model non –liniarități.

- Straturi de pooling, care sub-eșantionează datele extrase din imagine de straturile convoluționale, pentru a reduce dimensiunile vectorului de trăsături, pentru a reduce timpul de procesare.

- Straturile dense (fully connected) – acestea realizează clasificarea pe trăsăturile extrase de straturile convoluționale și sub-eșantionate de straturile de pooling. Într-un strat dens fiecare nod din strat este conectat la toate nodurile din stratul precedent.

În mod uzual, un CNN este compus dintr-o colecție de module convoluționale care realizează extracția de trăsături. Fiecare modul este format dintr-un strat convoluțional urmat de un strat de pooling. Ultimul strat convoluțional este urmat de unul sau mai multe straturi de pooling.

Ultimul strat dens din rețea conține un singur nod pentru fiecare clasă din model, cu o funcție de activare softmax pentru a genera valori intre 0 și 1 pentru fiecare nod (suma tuturor valorilor softmax este 1). Putem interpreta valorile softmax pentru o imagine ca o măsură ce ne indică clasa de apartenență a acelei imagini.

Pentru arhitectura de față am ales în urma unor experimente modelul următor:

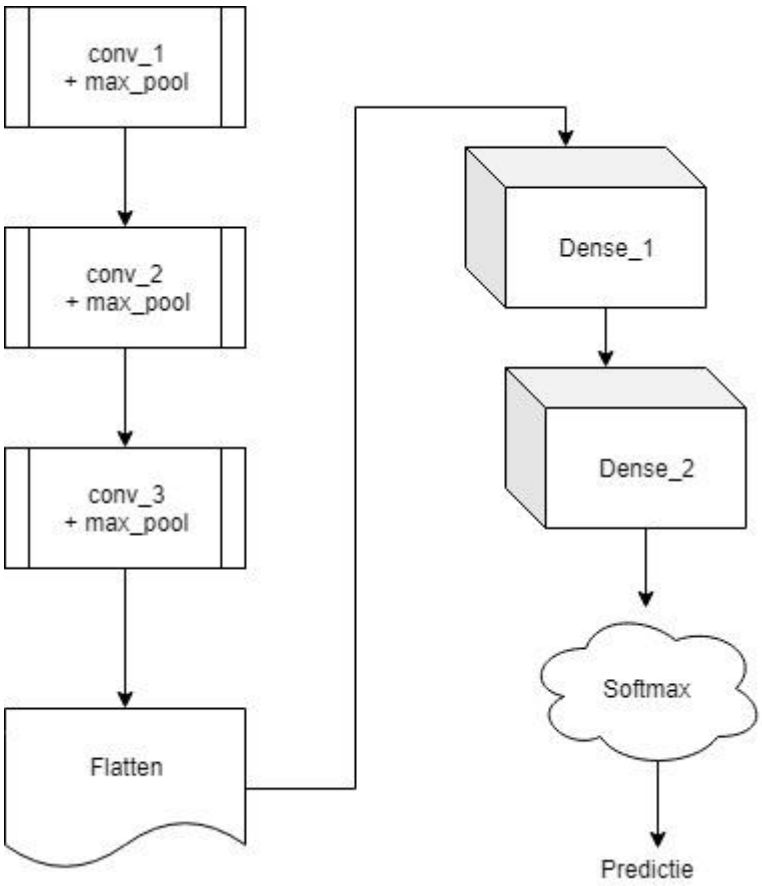
- Se vor folosi 3 straturi convoluționale c1, c2, c3 – primele două straturi vor avea 32 filtre de 3x3 (pixeli), ultimul strat folosind 64 de filtre de 3x3. Funcția de activare folosită va fi ReLU.

- Se va folosi max-pooling după fiecare strat convoluțional

- Un strat de flattening pentru a crea vectorul de caracteristici

- Se vor folosi 2 straturi dense la final

- Softmax pentru a obține predicția



Figură 4.1 – Arhitectura rețelei create

Pentru antrenare s-a încercat crearea unui loader pentru parcurgerea și separarea bazei de date dar și pentru a putea realiza încărcarea în TensorFlow a imaginilor. Pentru această operație s-a folosit OpenCv pentru citirea, re-dimensionarea și procesarea imaginilor. OpenCV poate fi util în cazul bazelor de date de dimensiuni mai mici dar experimental s-a observat că această metodă de încărcare a imaginilor nu este foarte eficientă din punct de vedere al memoriei, astfel pentru baze de date de dimensiuni mari timpul de încarcare pentru antrenare este relativ mare.

Loader-ul permite operații precum modificarea dimensiunii imaginii, modificarea numărului de canale (culori), modificarea dimensiunii unui batch, modificarea raportului de separare a bazei de date antrenare / validare.

Antrenarea se face conform standardului uzual TensorFlow și anume: se trece printr-un număr setat de iterații, în care se face load la batch-ul de antrenare, etichete, batch-ul de validare, folosind loader-ul. După aceea, se pornește o sesiune TensorFlow (session.run) în care se realizează antrenarea rețelei folosind AdamOptimizer cu learning rate pre stabilit, pentru minimizarea costului.

Optimizatorul Adam este un algoritm foarte utilizat și este folosit în locul algoritmilor stohastici bazați pe gradient pentru a seta ponderile rețelei într-un mod iterativ (în timpul antrenării)

După antrenarea și modificarea ponderilor se realizează testarea acurateții de antrenare și validare.

Toate aceste informații sunt salvate în vederea utilizării lor în tensorboard (aplicație care poate oferi o vizualizare grafică a ceea ce se întâmplă cu rețeaua în timpul antrenării).

4.3 BAZA DE DATE FACESCRUB

Un rol important în îmbunătățirile constante aduse în domeniul recunoașterii automate a fețelor și expresiilor faciale, îl au colecțiile de baze de date cu fețe. În prezent, există câteva baze de date utilizate pentru recunoașterea facială, acestea variind în dimensiune, postură, expresii, condiții de iluminat, precum și în numărul de subiecți.

De-a lungul timpului, majoritatea algoritmilor de recunoaștere facială bine cunoscuți, au demonstrat performanțele cu baze de date de la : Laboratoarele AT&T Cambridge, Facial Recognition Technology (FERET), Facial Database from visions Group Essex, Cohn Kande AU-Coded Facial Expression Database (FE) , NIST Mug shot Database, Extended Multi Modal Verification for Teleservices and Security applications (XM2VTS) Database , Baza de date AR Face din Ohio , Baza de date Yale Face , Caltech Faces sau Baza de date Japanese Female Facial Expression (JAFFE).

Baza de date cu imagini publice utilizată pentru acest proiect este FaceScrub.

Aceasta conține imagini cu fețe pentru 530 de persoane, atât femei, cât și bărbați (actori celebri), având aproximativ 200 de poze pentru fiecare persoană. Numărul total și împărțirea acestora sunt prezentate în tabelul următor:

FaceScrub	**Bărbați**	**Femei**	**Total**
#Persoane	265	265	530
#Imagini	55,306	51,557	106,863

Tabel 4.1 Baza de date FaceScrub

Baza de date FaceScrub poate fi considerată una de dimensiuni mari, fiind printre cele mai mari baze de date publice. Aceasta a fost construită prin detectarea fețelor în imaginile returnate de căutările pe Internet ale unor figuri publice, urmată apoi de eliminarea automată a acelora care nu aparțineau persoanelor căutate și de verificarea și curățarea manuală [7] a setului de imagini rezultate. Acestea conțin etichete pentru nume și gen.

Etapele descărcării bazei de date publice FaceScrub:

1. Accesând link-ul [5], se completează online un formular cu câteva date personale, pentru a primi un email cu link-ul către arhiva ce conține URL-urile imaginilor ce vor putea fi descărcate mai apoi. În același email se primește și parola pentru dezarhivare.
2. Cele două fișiere principale obținute în urma dezarhivării (facescrub_actors și facescrub_actresses) conțin câteva coloane text cu informații precum numele, id-ul imaginii și URL-ul de la care se poate descărca. Pentru a descărca imaginile, se utilizează scriptul public python3_download_facescrub.py.
3. În scriptul public python3_download_facescrub.py, singura variabilă care trebuie modificată este „MY_USER_AGENT_STRING”. Această variabilă conține tipul de browser utilizat pentru a descărca imaginile. Pentru a afla această informație, se poate accesa link-ul din referința [6].
4. În linia de comandă, pentru sistemele de operare Windows, se rulează scriptul python3_download_facescrub.py, astfel:
   > py python3_download_facescrub.py input_data data_path , unde input_data este unul dintre cele două fișiere de mai sus (facescrub_actors și facescrub_actresses) sau ambele, iar data_path va fi calea către directorul în care se dorește a fi salvate imaginile.

4.4 HIPERPARAMETRII REȚELEI

O parte dintre hiper parametrii rețelei sunt:

1. Rata de învățare Rata de învățare controlează cât de mult să se actualizeze ponderile în algoritmul de optimizare. Se pot folosi rate de învățare fixe, rate de învățare ce scad gradual sau rate de învățare adaptive.
2. Numărul de epoci

Numărul de epoci reprezintă de câte ori întregul set de antrenare trece prin rețeaua neuronală. Acest număr ar trebui crescut treptat, până când se poate vedea un mic decalaj între eroarea la testare și eroarea la antrenare.

3. Dimensiunea lotului

Mini-loturile sunt preferate de obicei în procesul de învățare al rețelelor convoluționale. un număr cuprins între 16 și 128 poate fi o alegere bună pentru testat.

4. Funcția de activare

Funcția de activare introduce non-linearitate modelului. De obicei, funcția redresoare funcționează bine cu rețele convoluționale. Alte alternative sunt funcțiile sigmoidă, tanh și altele. depinzând de sarcină.

5. Inițializarea ponderilor

De regulă, ponderile se inițializează cu valori mici, alese aleator, însă acestea nu trebuie să fie prea mici, pentru a evita gradientul zero.

6. Deviația standard

Deviația standard este o măsură care are ca rol cuantizarea varianței sau dispersiei unui set de date. În cazul de față, deviația standard va fi radical din varianță.

Pentru a calcula varianța, este necesar ca mai întâi să se calculeze media aritmetică a tuturor elementelor pentru care se dorește la final determinarea deviației standard. Dacă notăm media aritmetică cu μ, atunci varianța va fi media aritmetică a pătratelor diferenței dintre fiecare element și medie:

$$\text{Varianța}=[(x_1-\mu)^2 + (x_2-\mu)^2 + \dots +(x_i-\mu)^2]/N$$

Așadar, deviația standard se va calcula astfel:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

O deviație standard mică indică faptul că datele tind să se apropie de medie (valoarea dorită) a setului de date. O deviație standard mare indică faptul că datele sunt împrăștiate pe un interval mai mare de valori.

4.5 METRICA EVALUĂRII

a). Acuratețea

Acuratețea(notată în continuare cu A), reprezintă raportul dintre numărul predicțiilor corecte și numărul total de predicții făcute de modelul utilizat.

În continuare, se va utiliza notația:

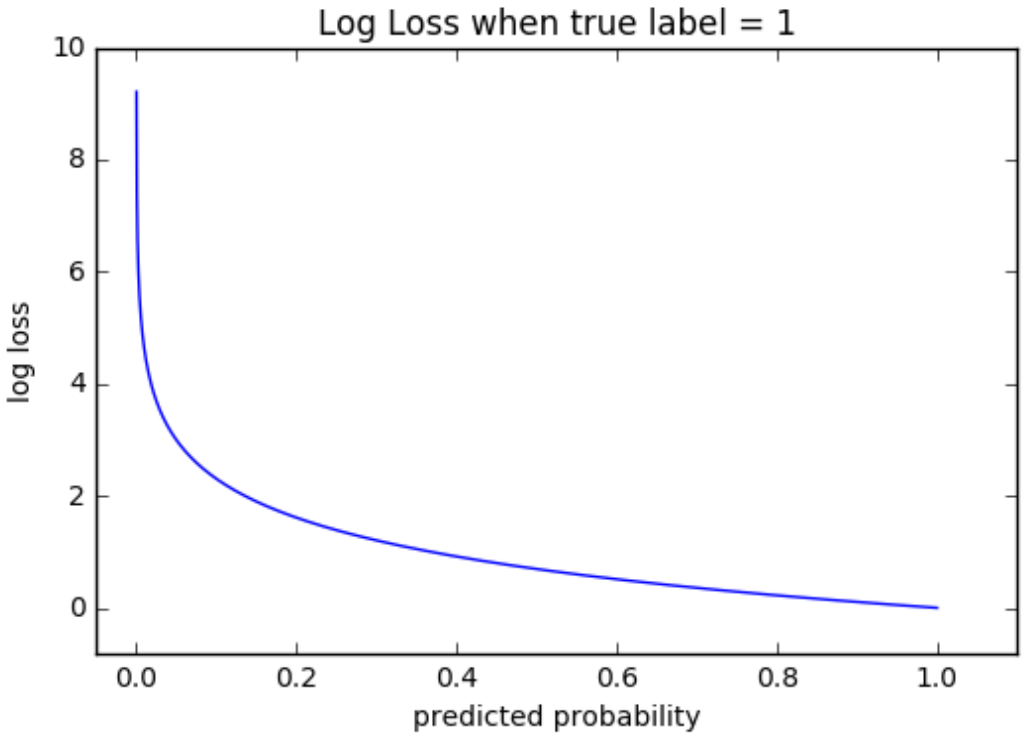
A = Numărul predicțiilor corecte / Numărul total de predicții

Pentru clasificare binară, acuratețea poate fi calculată și în termeni de pozitiv și negativ, după cum urmează:

A = (TP+TN) / (TP+TN+FP+FN),

unde: *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives

b). Pierderea prin entropie (eng. cross-entropy loss sau log loss) , măsoară performanța unui model de clasificare a cărui ieșire este o valoare probabilistică între 0 și 1. Pierderea prin entropie crește cu cât probabilitatea prezisă diverge față de eticheta reală, ceea ce înseamnă că prezicând o probabilitate de 0.01 când eticheta de observare este defapt 1, ar duce la o creștere a valorii pierderii, ceea ce nu se dorește. Un model perfect ar avea o pierdere prin entropie de 0.



Figură 4.2 – Pierderea prin entropie

Graficul de mai sus arată gama de valori posibile ale pierderilor, fiind dată o observație adevărată. Cu cât probabilitatea prezisă se apropie de 1, pierderea prin entropie scade ușor. Însă cu cât probabilitatea prezisă scade, pierderea prin entropie crește rapid.

În clasificarea binară, unde numărul de clase M este egal cu 2, pierderea prin entropie poate fi calculată astfel:

$$-(y\log(p)+(1-y)\log(1-p))$$

Dacă M>2 (clasificare cu mai multe clase), se calculează o pierdere separată pentru fiecare etichetă a clasei per observație și se adună rezultatul.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

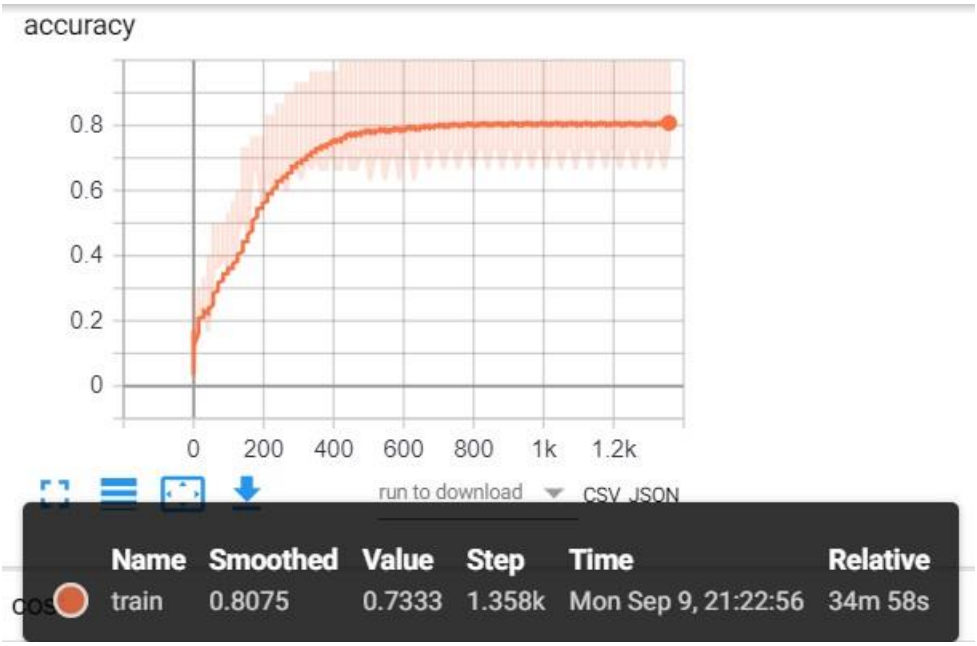
unde:

4.6 REZULTATE EXPERIMENTALE:

4.6.1 Variația ratei de învățare (lr)

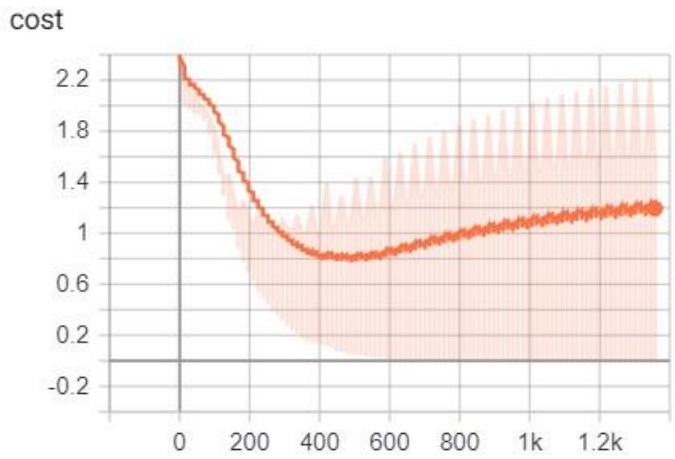
I) lr=0.0001

a. Acuratețea



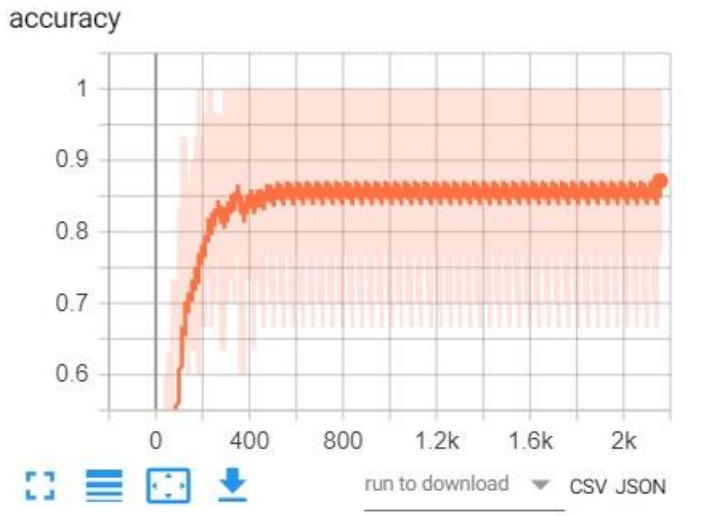
Final test accuracy:72.22 %

b. Pierderea prin entropie



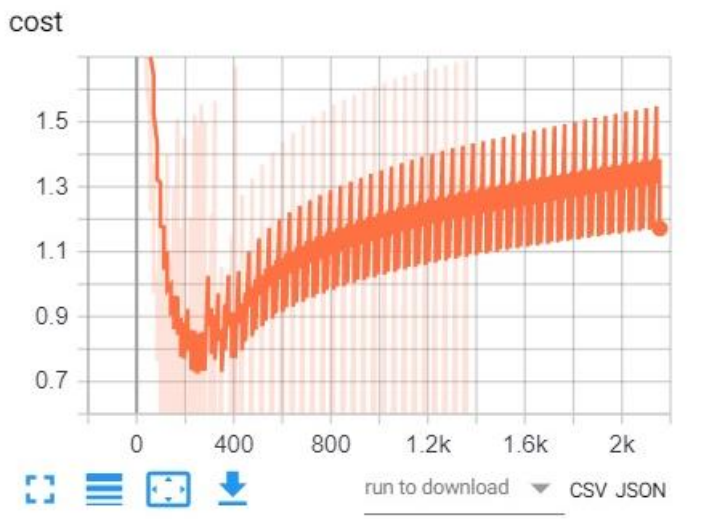
II) lr=0.0005

a. Acuratețea



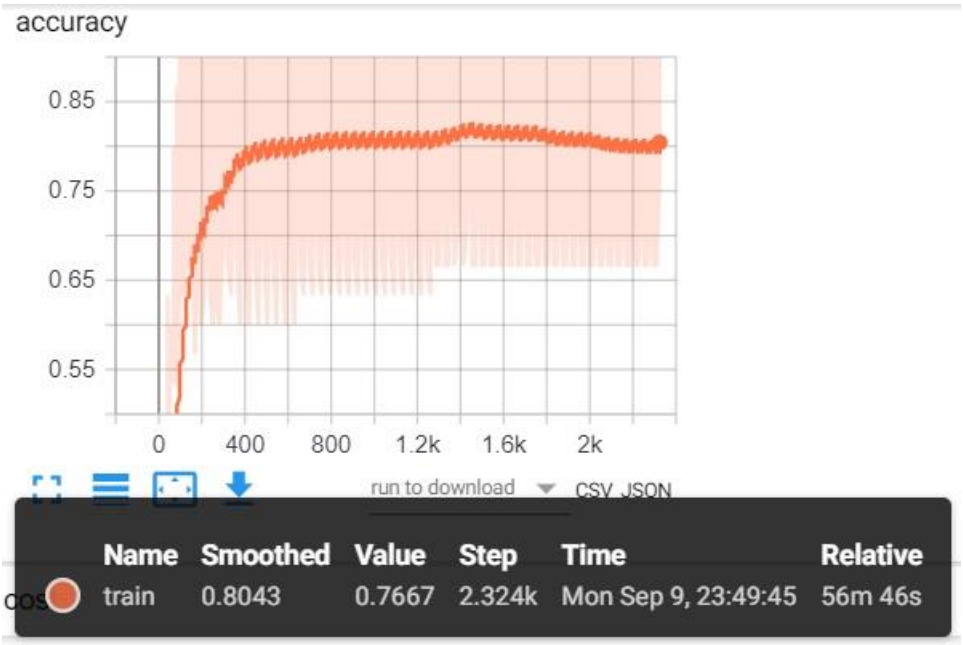
Final test accuracy: 83.33%

b. Pierderea prin entropie



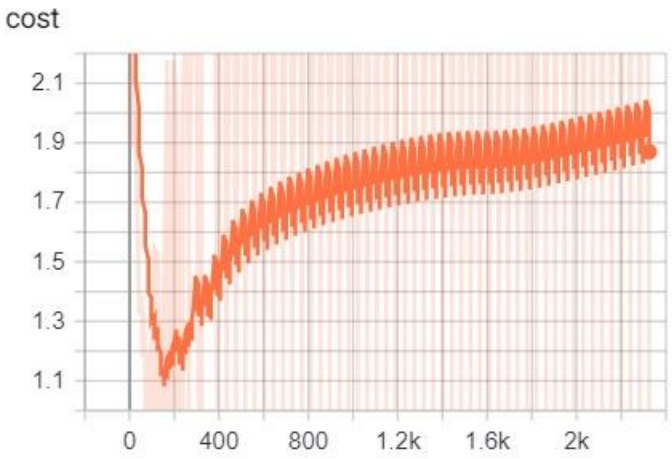
III) lr=0.001

a. Acuratețea



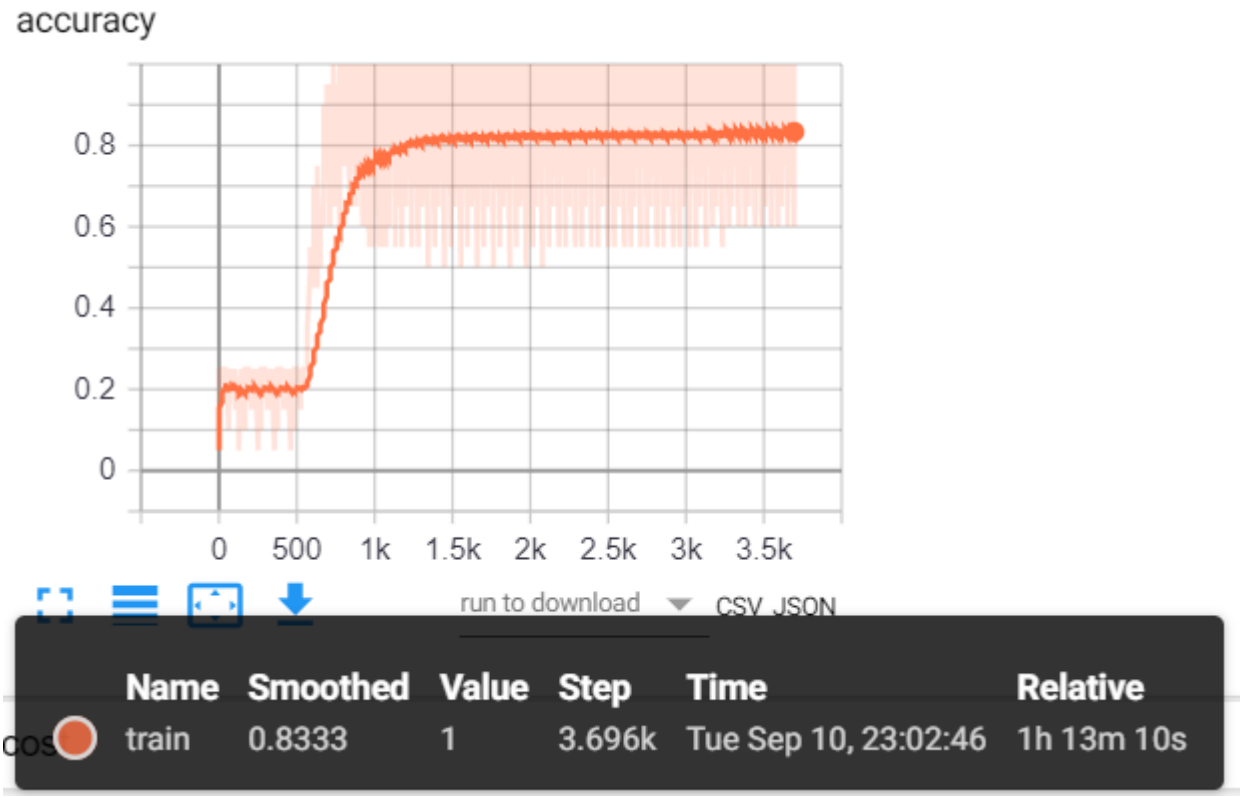
Final test accuracy: 77.77%

b. Pierderea prin entropie



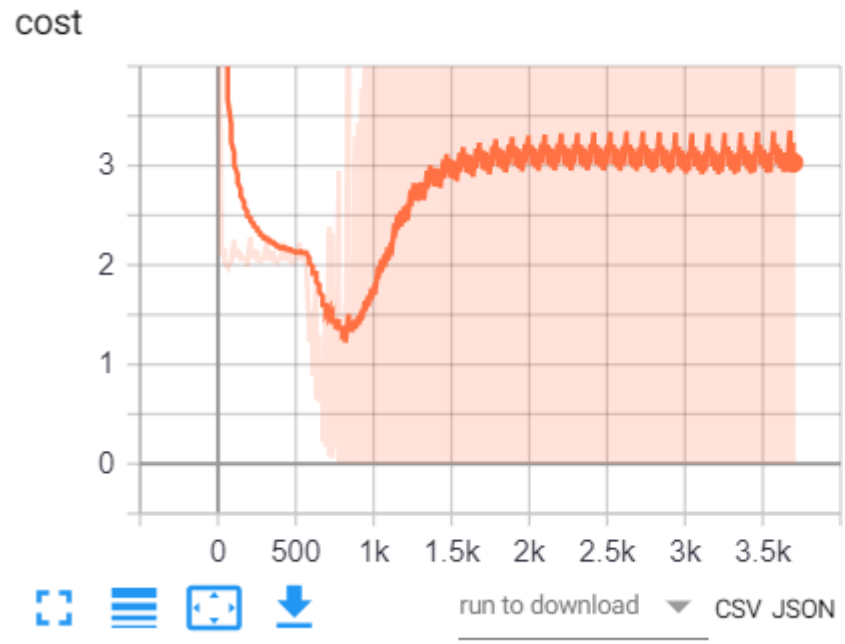
IV) lr=0.005

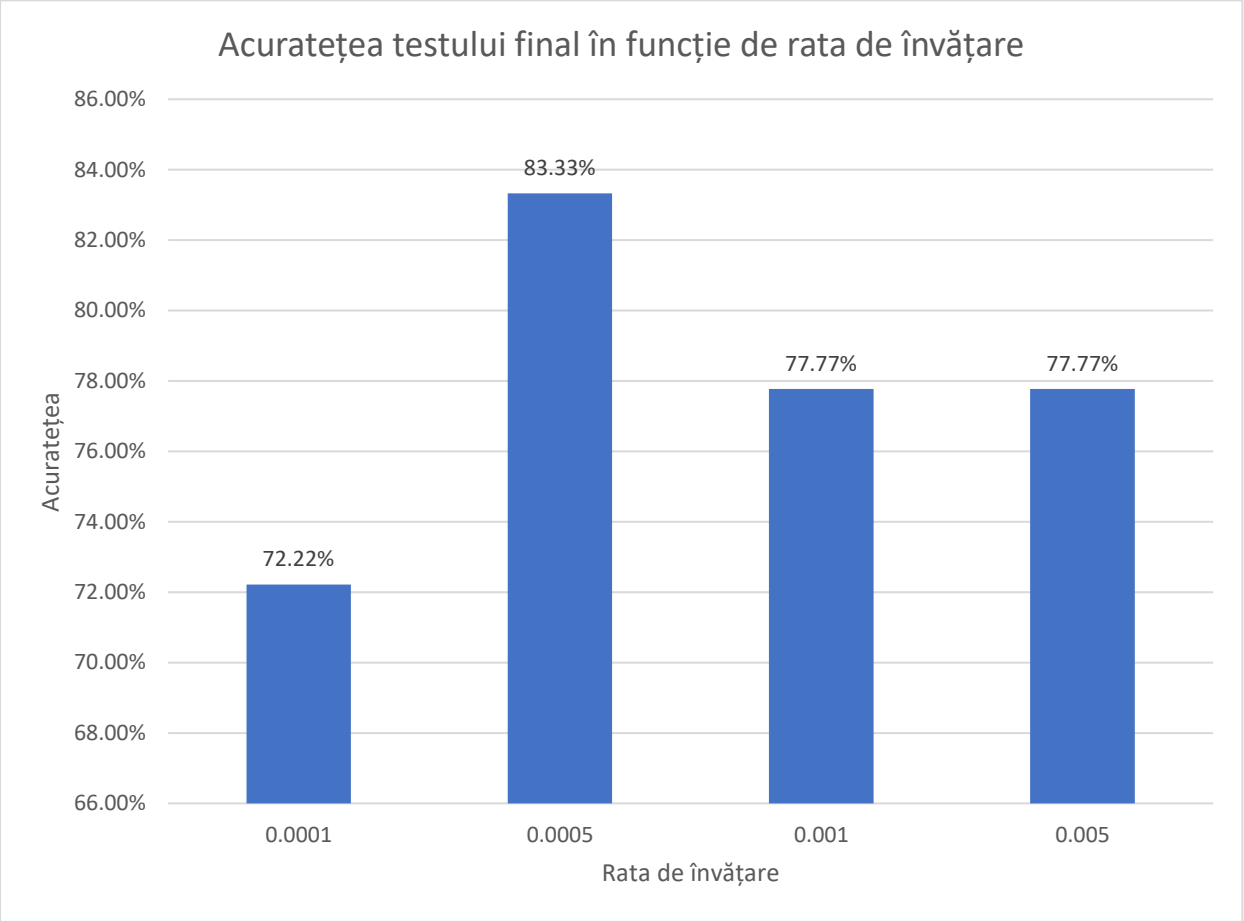
a. Acuratețea



Final test accuracy: 77.77%

b. Pierderea prin entropie





Figură 4.3- Acuratețea testului final în funcție de rata de învățare

CAPITOLUL 5

IMPLEMENTAREA MODELULUI INCEPTION V3 PE BAZA DE DATE FACESCRUB

5.1 TRANSFERUL ÎNVĂȚĂRII

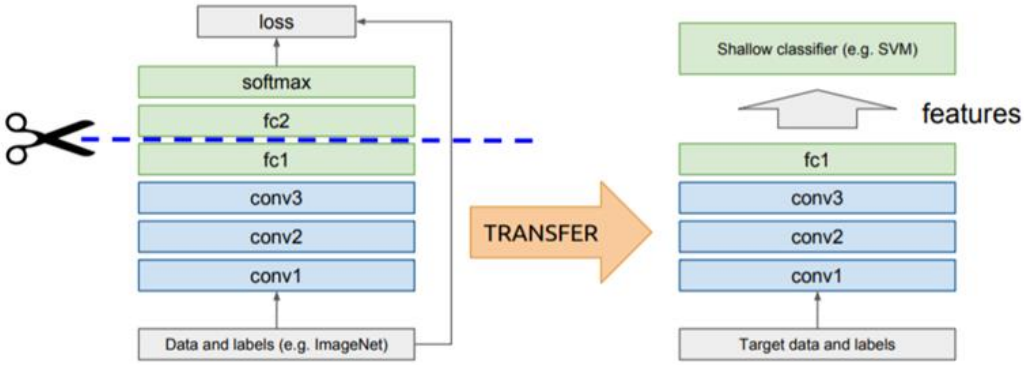
Modele de transfer al învățării profunde

Învățarea profundă a progresat extrem de mult în ultimii ani, acest lucru penrmițându-ne să atingem probleme complexe și să obținem rezultate foarte bune. În continuare, sunt detaliate două dintre cele mai utilizate strategii de transfer al învățării profunde.

Modele pre-antrenate “off-the-shelf” utilizate ca extractori de trăsături

Sistemele de învățare profundă și modelele sunt arhitecturi în straturi, care învață trăsături diferite pe straturi diferite (reprezentări ierarhice ale trăsăturilor stratificate). Aceste straturi sunt conectate apoi la un ultim strat (de obicei un strat fully-connected, în cazul învățării supervizate)

pentru a obține rezultatul final. Această arhitectură stratificată permite utilizarea unei rețele pre-antrenate (precum Inception-v3 sau VGG) fără stratul ei final, ca extractor fix de trăsături pentru alte sarcini.

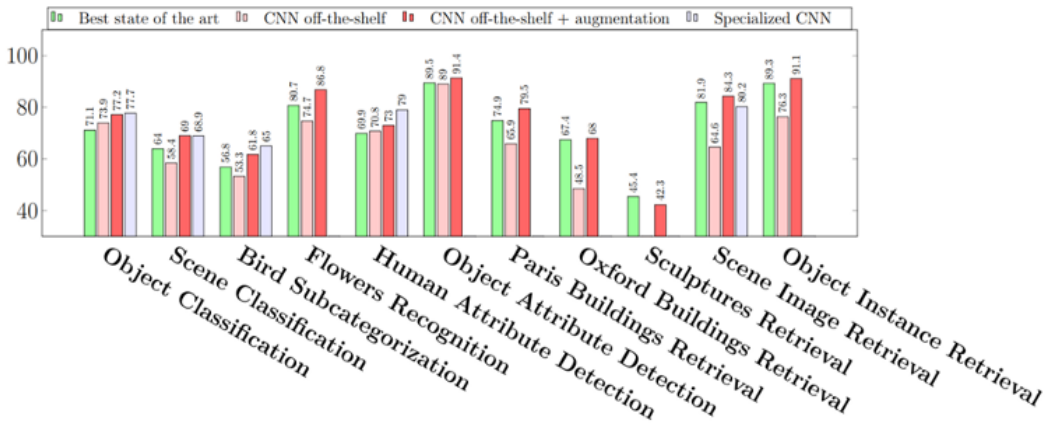


Figură 5.1 – Transferul învățării cu modele de învățare profundă pre antrenate ca extractori de trăsături

Ideea principală aici este să se beneficieze de straturile ponderate ale modelului pentru a extrage trăsăturile fără a actualiza ponderile straturileor modelului în timpul antrenării cu date noi pentru noua sarcină.

De exemplu, dacă utilizăm un model fără stratul lui final de clasificare, ne va ajuta să transformăm imagini dintr-o sarcină dintr-un nou domeniu, într-un vector n-dimensional bazat pe stările lui ascunse , permițându-ne totuși să extragem trăsături dintr-o sarcină dintr-un nou domeniu utilizând informațiile dintr-o sarcină dintr-un domeniu-sursă. Aceasta este una dintre cele mai utilizate metode de a realiza transferul învățării utilizând rețele neuronale profunde.

Această tehnică pare să funcționeze extrem de bine în sarcini din lumea reală , în figura 5.2 fiind prezentată performanța acesteia pentru diferite sarcini bazate pe computer vision.



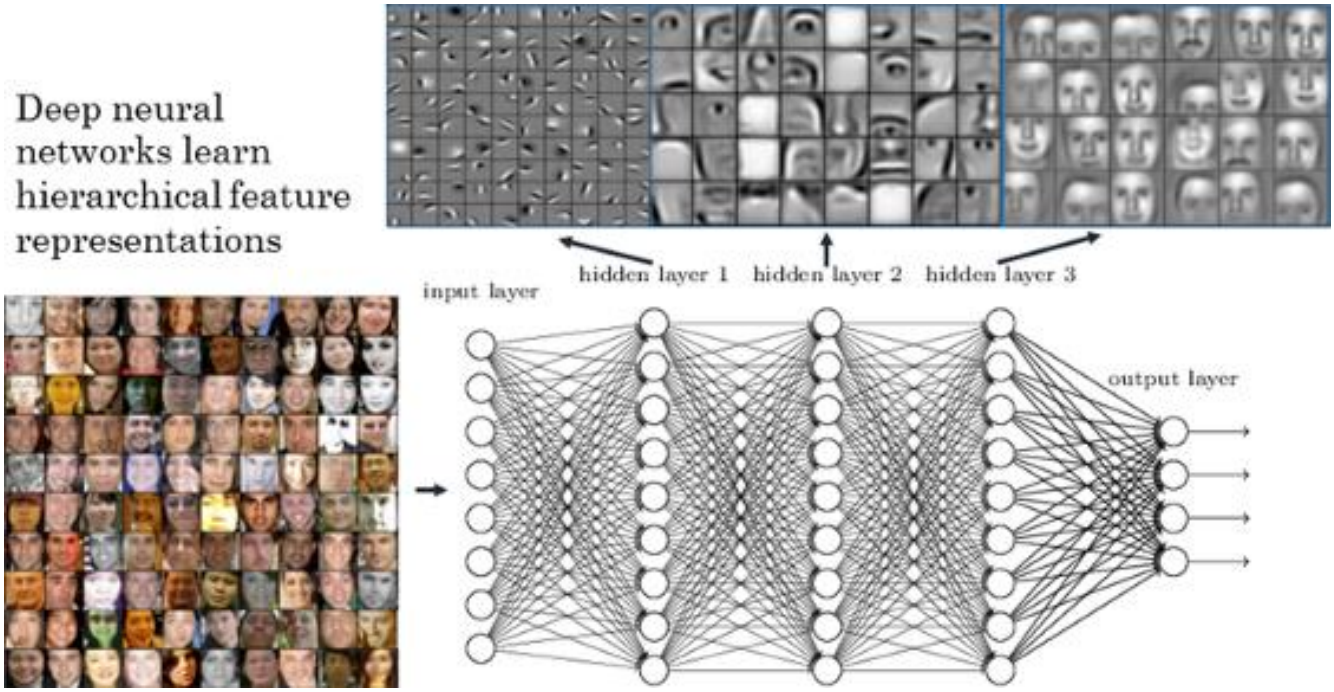
Figură 5.2 – Performanța unor modele pre antrenate versus a unor modele specializate

Performanța modelelor pre-antrenate off-the-shelf versus modelele specializate concentrate pe sarcină se poate observa în 5.2. marcate cu roșu și cu roz, denotă clar faptul că trăsăturile modelelor preantrenate sunt mai performante decât în cazul modelelor foarte specializate pe sarcină.

Reglarea fină a modelelor pre-antrenate off-the-shelf

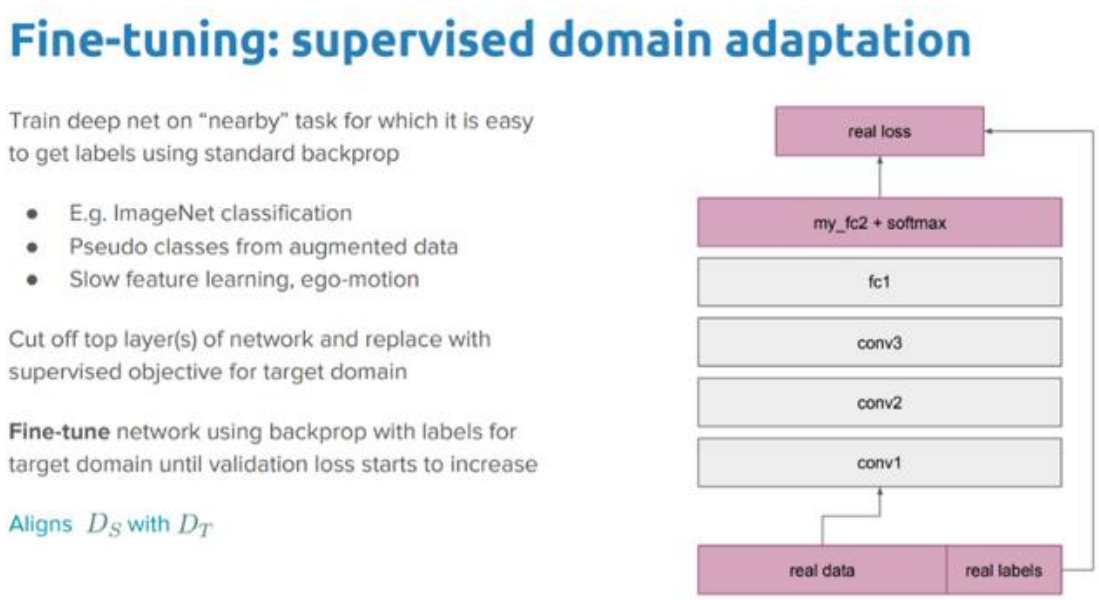
Aceasta este o tehnică ce necesită mai multă implicare, unde nu doar se înlocuiește stratul final (pentru clasificare/regresie), ci se și reantrenează selectiv unele din staturile anterioare. Rețelele neuronale profunde sunt arhitecturi foarte configurabile cu diverși hiperparametri. Straturile inițiale sunt utilizate pentru a capta trăsăturile generale, în timp ce următoarele se axează mai mult pe sarcina care i se dă.

În figura 5.3 este prezentată o sarcină de recunoaștere facială, unde straturile inițiale învață trăsături foarte generale , iar straturile de mai sus învață trăsături foarte specifice sarcinii.



Figură 5.3 – Reprezentarea procesului de învățare a unei rețele pre antrenate

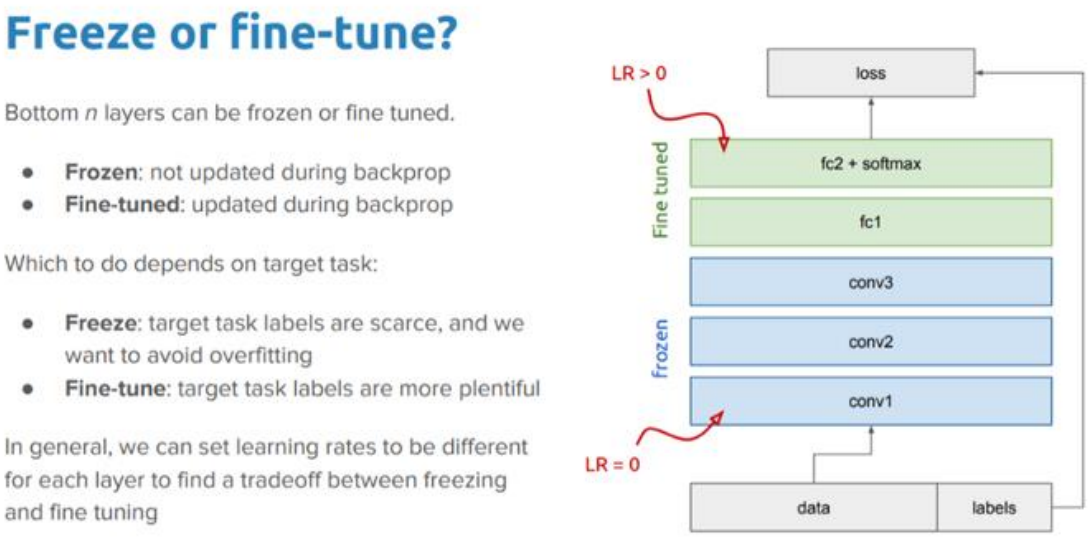
Utilizând aceste tehnici, se pot îngheța anumite straturi (ceea ce se poate traduce în ponderi fixe), în timp ce restul se pot reantrena sau ajusta fin, în funcție de nevoile sarcinii.



Figură 5.4 – Reglare fină: Adaptarea domeniului supervizat

Figura 5.4 încearcă să răspundă la întrebarea "Ar trebui să înghețăm straturile rețelei pentru a le utiliza ca extractori de trăsături sau ar trebui să și ajustăm fin straturi în acest proces?

Înghețare sau ajustare?



Figură 5.5 – Înghețare sau ajustare?

5.2 ARHITECTURA INCEPTION-V3

Inception V3 de la Google este a 3-a versiune dintr-o serie de arhitecturi convolutionale de deep learning.

Inception-v3 are ca scop optimizarea arhitecturii Inception prin creșterea eficienței computaționale și scăderea numărului de parametri.

Inception V3 este o arhitectură ce înglobează o rețea neuronală de învățare profundă cu 42 de straturi, ce poate ajunge la o complexitate similară cu cea a arhitecturii VGGNet, care are de 25 ori mai mulți parametrii. Această optimizare a fost una eficientă și din punctul de vedere al minimizării ratei de eroare, Inception V3 obținând astfel locul 1 la competiția ImageNet Large Scale Visual Recognition (ILSVRC) din anul 2015.

Există 4 versiuni ale arhitecturii Inception: [Arhitectura convoluțională profundă Inception a fost introdusă ca GoogLeNet în (Szegedy et al. 2015a), numită ici **Inception-v1**. Mai târziu, arhitecura Inception a fost rafinată în mai multe moduri, mai întâi prin introducerea **normalizării lotului** (Ioffe și Szegedy 2015) (**Inception-v2**). Mai târziu, prin ideile de factorizare în cea de a treia iterație (Szegedy et al. 2015b) la care ne vom referi prin **Inception-v3** în acest raport] (Tradus din [13]).

5.2.1 Particularități ale arhitecturii Inception-v3, comparativ cu celelalte versiuni ale arhitecturii Inception

1. Factorizarea convoluțiilor

Scopul factorizării convoluțiilor este reducerea numărului de conexiuni/parametrii fără a micșora eficiența rețelei.

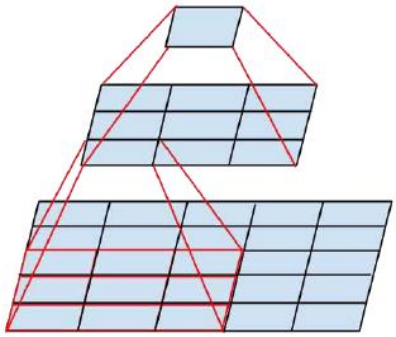
S-au utiliza două tipuri de factorizare:

a. Facotizarea în convoluții mai mici
b. Facotizarea în convoluții asimetrice

Exemplu de **facotizare în convoluții mai mici**:

Se înlocuiește o convoluție 5x5 cu două convoluții 3x3. Utilizând un strat cu un filtru 5x5, numărul parametrilor va fi egal cu : 5x5=25. În schimb, utilizând 2 straturi cu filtre 3x3, numărul parametrlor va fi: 3x3+3x3=18.

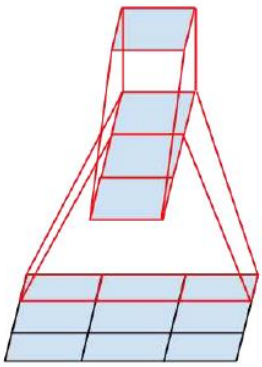
Astfel, numărul parametrilor este reduc cu 28%.



Figură 5.6 – Factorizarea convoluțiilor în convoluții mai mici [14]

Exemplu de factorizare în convoluții asimetrice :

O convoluție 3x1 urmată de o convoluție 1x3 înlocuiește o convoluție 3x3.



Figură 5.7 – Factorizarea convoluțiilor în convoluții asimetrice [14]

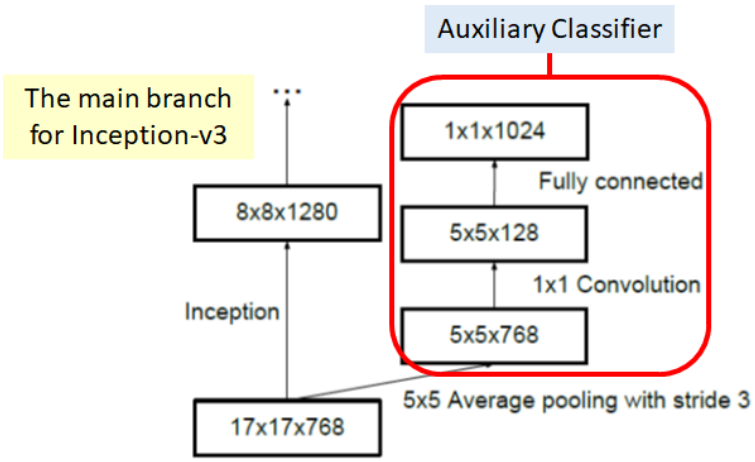
Utilizând un filtru 3x3, numărul de parametrii este egal cu: 3x3=9.

În schimb, utilizând filtrele 3x1 si 1x3 , numărul de parametrii va fi: 3x1+1x3 = 6. Astfel, numărul de parametrii este redus cu 33%.

Cu ajutorul factorizării, numărul parametrilor este redus pentru întreaga rețea, iar probabilitatea să se producă overfitting-ul este mult mai mică, ceea ce înseamnă că rețeaua poate merge mai în adâncime.

2. Clasificatorul auxiliar

Spre deosebire de arhitectura Inception-v1, care utiliza două clasificatoare auxiliare, Inception-v3 utilizează un singur clasificator auxiliar, peste ultimul layer 17x17. Clasificatorul auxiliar este utilizat ca regularizator, spre deosebire de Inception-v1, unde clasificatorii auxiliari erau utilizați pentu a avea o rețea mai profundă.



Figură 5.8 – Clasificator auxiliar acționând ca un regulator [14]

5.2.2 *Starea artei referitoare la arhitecturi de tețele neuronale profunde utilizate în aplicații de recunoaștere facială*

Cu model-singular și multi-decupare, Inception-v3 cu 144 decupări obține o rată de eroare pentru top-5 de 4.2%, ceea ce înseamnă că este mai performantă decât PReLU-Net și Inception-v2 care au fost publicate la finalul anului 2015.

	Rețea	Decupări evaluate	Eroare top-1	Eroare top-5
Inception-v1	GoogLeNet	10	-	9.15%
	GoogLeNet	144	-	7.89%
	VGG	-	24.4%	6.8%
Inception-v2	BN-Inception	-	-	-
	PReLU	10	24.27%	7.38%
	PReLU	-	21.59%	5.71%
Inception-v3 cu diferite	Inception-v3	12	19.47%	4.48%
	Inception-v3	144	**18.77%**	**4.2%**

Tabel 5.1 - Rezultate model-singular multi-decupare

Inception-v1

Inception-v2

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

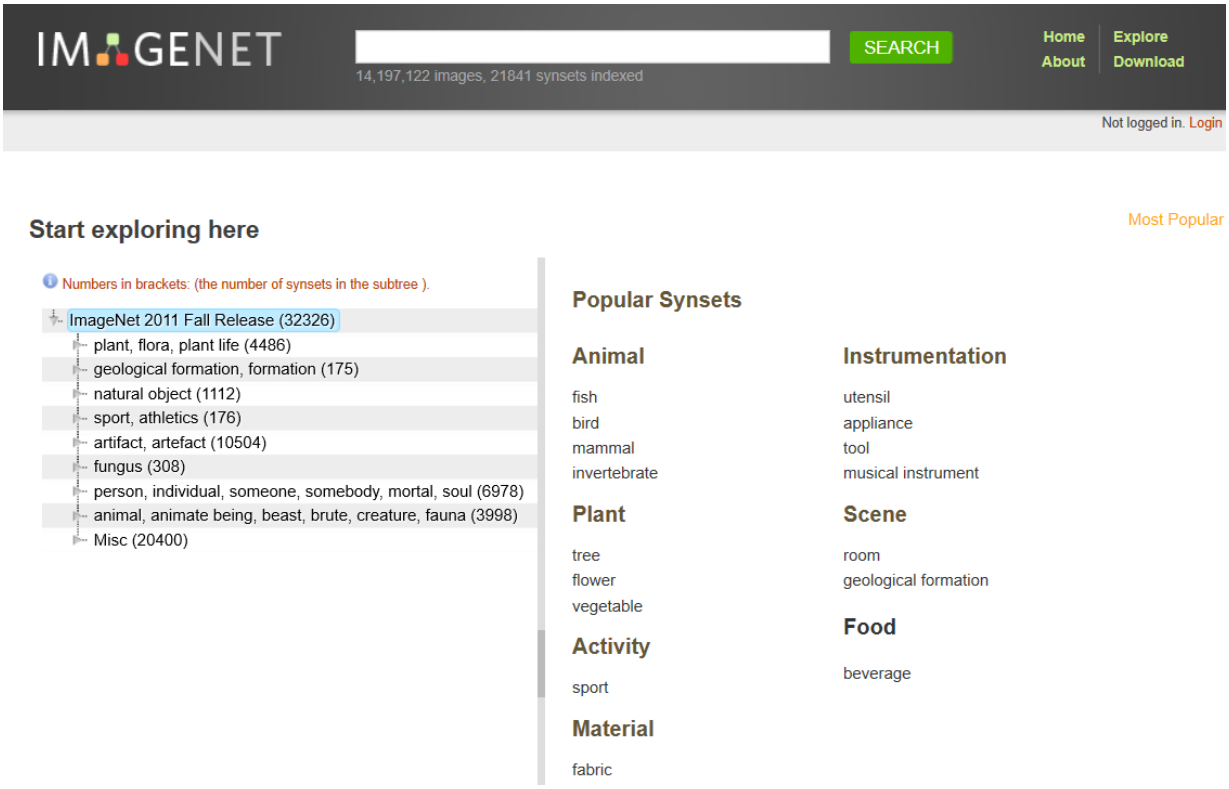
Tabel 5.2 - Rezultate multi-model multi-decupare

Cu multi-model multi-decupări, Inception-v3 cu 144 decupări și 4 modele asamblate, rata de eroare top-5 obținută a fost 3.58%, obținând titlul “1st Runner Up” (pentru clasificare de imagini) în ILSVRC 2015.

Înainte ca modelul să poată fi utilizat pentru a recunoaște imagini, acesta trebuie antrenat. Acest lucru se face de obicei prin învățare supervizată folosind un set mare de imagini etichetate. Deși Inception V3 poate fi antrenat prin diverse seturi de imagini etichetate, ImageNet este o alegere uzuală.

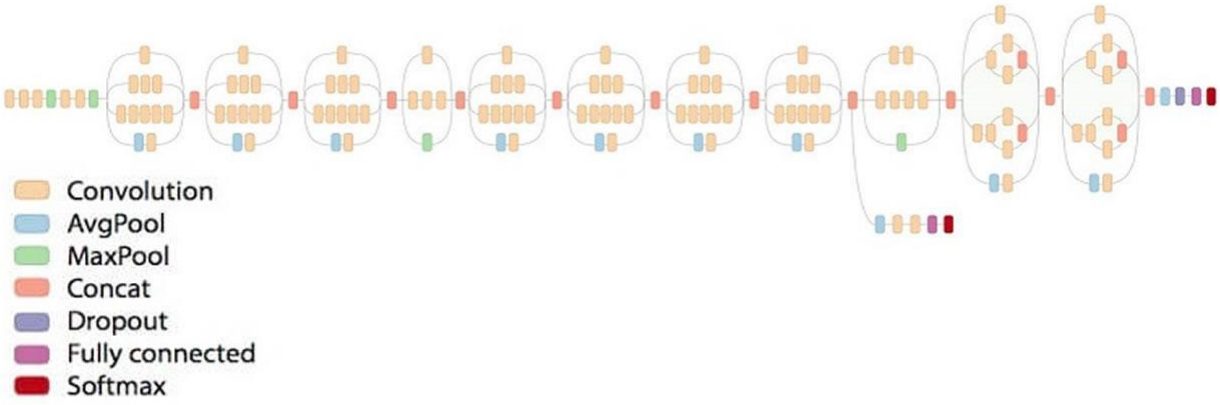
Modelul Inception v3 utilizează pentru etapa de pre-antrenare, setul de imagini IMAGENET. Acesta conține imagini cu persoane, animale (pești, păsări, mamifere, etc) , plante(copaci, flori, legume), activități (sport), materiale textile, mâncare , camere, formațiuni geologice , instrumente și ustensile.

Inception V3 a fost antrenată utilizând un set de date de 1000 de clase din setul original de date ImageNet, care a fost antrenat cu peste 1 milion de imagini de antrenare. În figura de mai jos, se regăsesc cele mai populare clase de obiecte, aparținând setului de date ImageNet.



Figură 5.9 – Structura bazei de date

Transfer learning permite antrenarea stratului final al unui model existent, având ca rezultat nu doar scaderea semnificativă a timpului de rulare, ci și scăderea dimensiunii setului de date necesar. Unul dintre cele mai cunoscute modele ce pot fi utilizate pentru transfer learning este Inception V3. Fiind antrenat la origine pe mai bine de 1 milion de imagini din 1000 de clase, pe mașini puternice, acest model, prin permiterea antrenării stratului final, face posibilă mentinerea informațiilor pe care le-a învațat în timpul antrenării originale și aplicarea acestora pe o baza de date mai mică, ca în cazul acestei lucrări, avand ca rezultat clasificări cu acuratețe foarte mare, fără a fi nevoie de o antrenare extensivă și de putere de calcul foarte mare.



Figură 5.10 – Arhitectura Inception-v3

Transfer learning poate fi extrem de util deoarece informațiile învățate dintr-un domeniu pot fi folosite apoi în alt domeniu, reducând considerabil timpul de rulare, prin eliminarea nevoii de a re-colecta datele de antrenare pentru un anumit domeniu.

Se poate lua o rețea preantrenată de clasificare de imagini care a învățat deja să extragă trăsături relevante și foarte importante din imagini naturale și se poate utiliza ca punct de plecare pentru o nouă sarcină. Majoritatea rețelelor preantrenate sunt antrenate pe un subset al bazei de date ImageNet, care este utilizat în ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Aceste rețele au fost antrenate pe mai mult de un milion de imagini și poate clasifica imagini în 1000 de categorii de obiecte, precum creioane, animale, plante, căni, etc. Utilizând o rețea preantrenată cu transfer de învățare este de obicei mult mai rapid și mai ușor decât să antrenezi o rețea de la început.

Rețelele preantrenate se pot folosi pentru următoarele sarcini:

Clasificare - Se pot aplica rețelele preantrenate direct pe sarcini de clasificare.

Extragere de trăsături – O rețea preantrenată se poate utiliza ca un extractor de trăsături folosind stratul de activări ca trăsături. Aceste activări se pot utiliza ca trăsături pentru a antrena un alt model de machine learning , precum support vector machine (SVM).

Transferul învățării – Se pot lua straturi dintr-o rețea antrenată pe un set de date și ajusta pentru un nou set de date.

5.3 REZULTATE EXPERIMENTALE

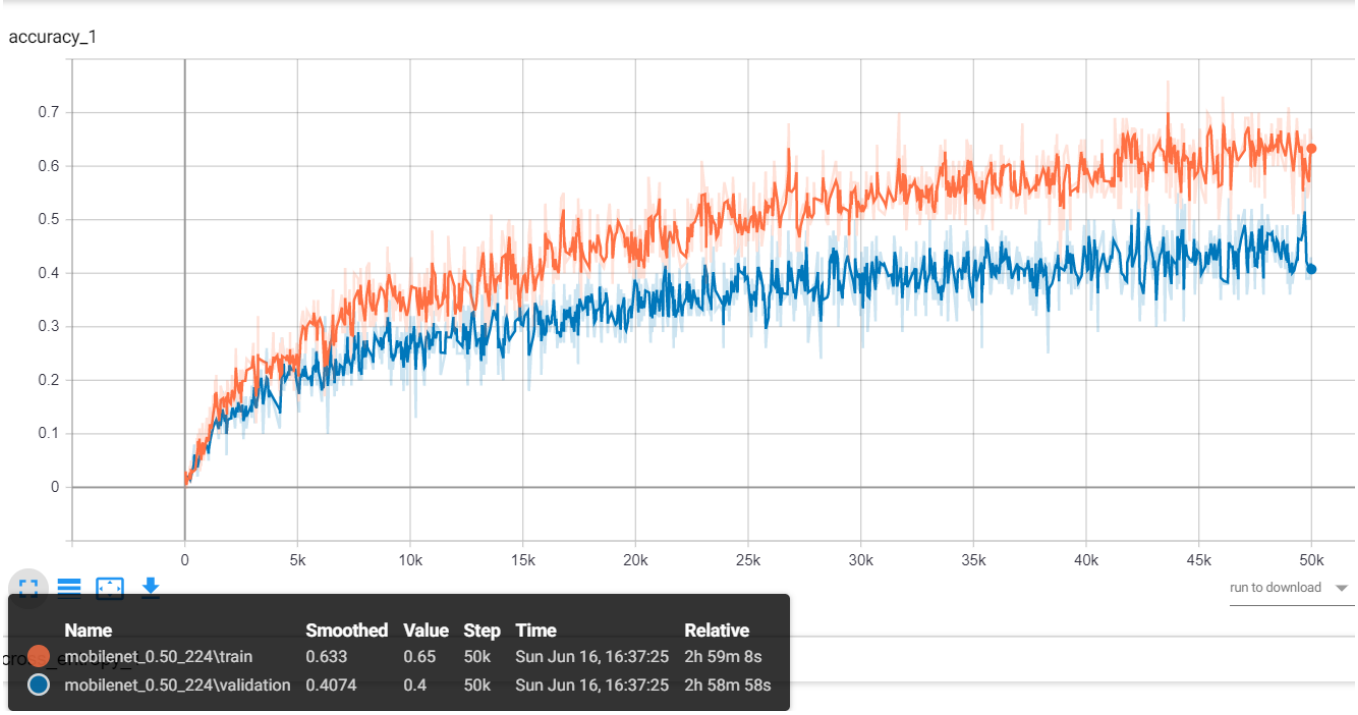
Rezultatele obținute la antrenarea setului de date ScrubFaces cu modelul Inception V3

Ca prim experiment, am variat doar numărul de pași de antrenare, pentru a observa evoluția performanțelor sistemului și stabilirea numărului optim de pași necesari pentru antrenarea pe baza de date ScrubFaces.

5.3.1 Variația numărului de paşi de antrenare

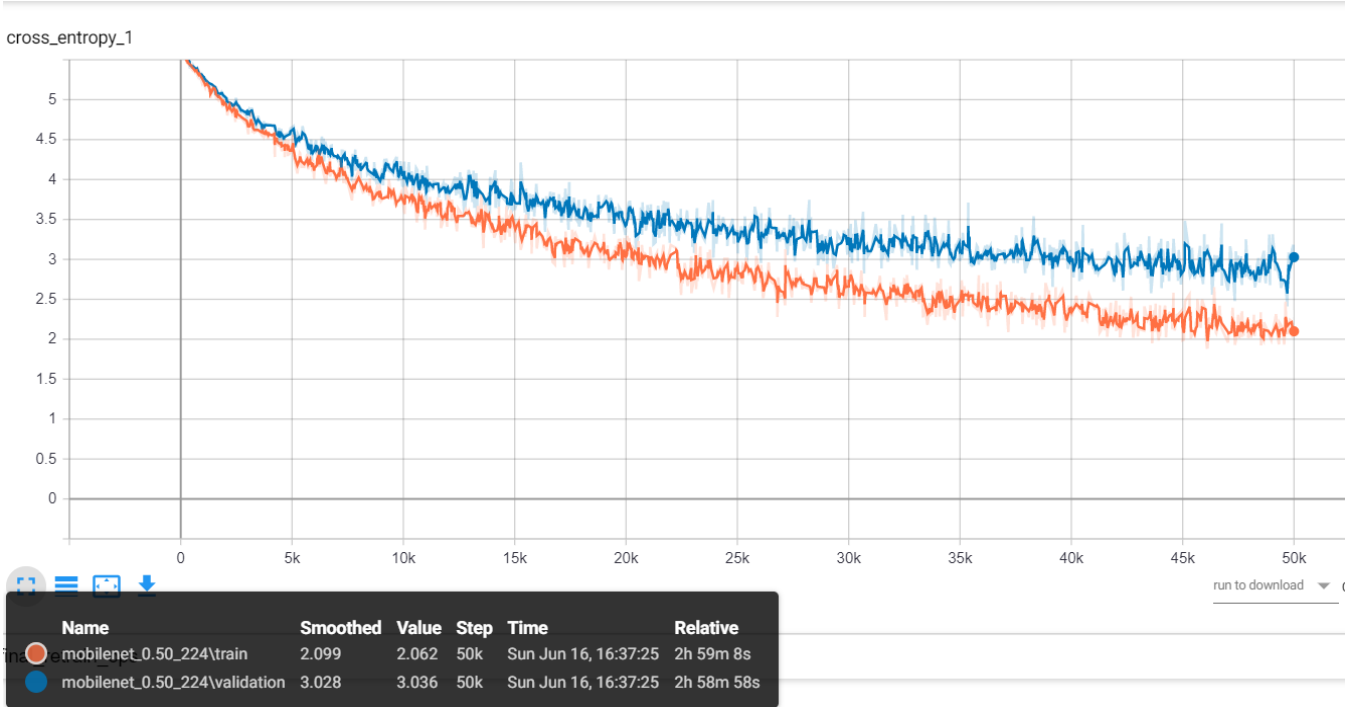
I) N=50.000 pași

c. Acuratețea



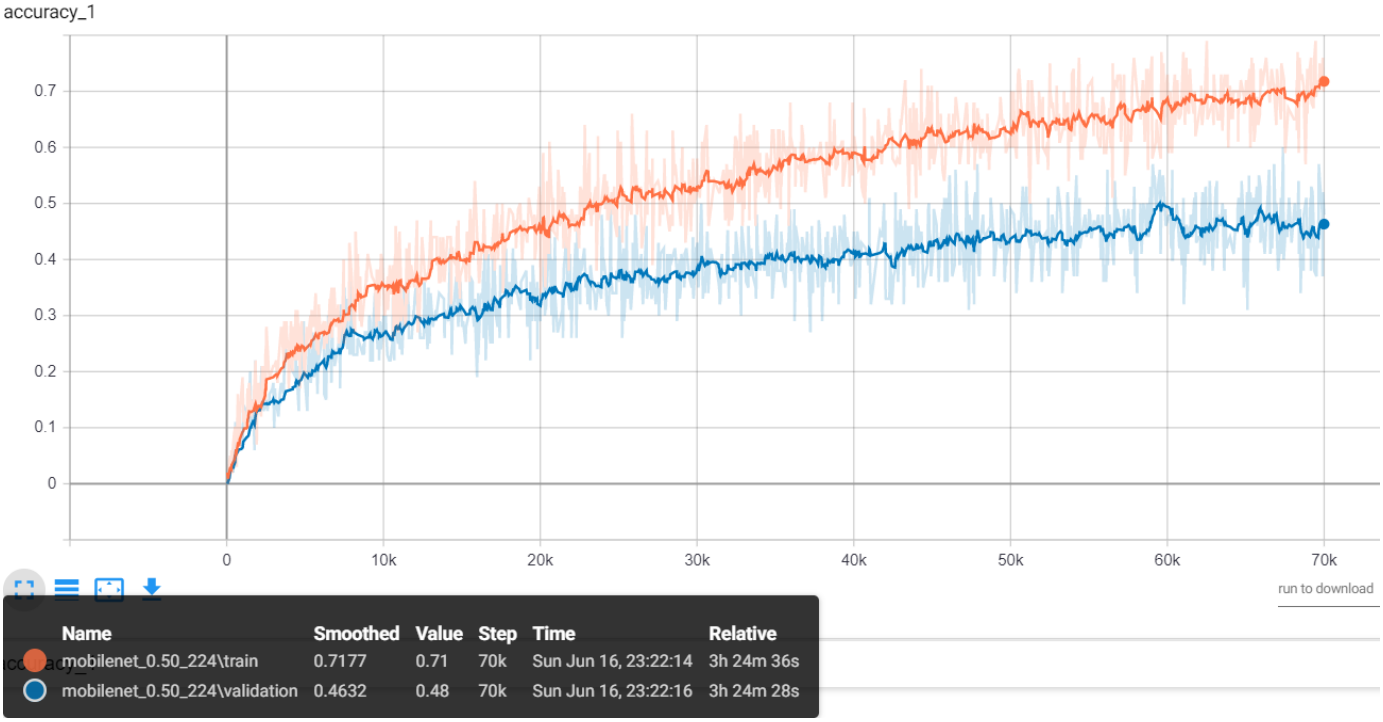
Final test accuracy: 41.3%

b. Pierderea prin entropie



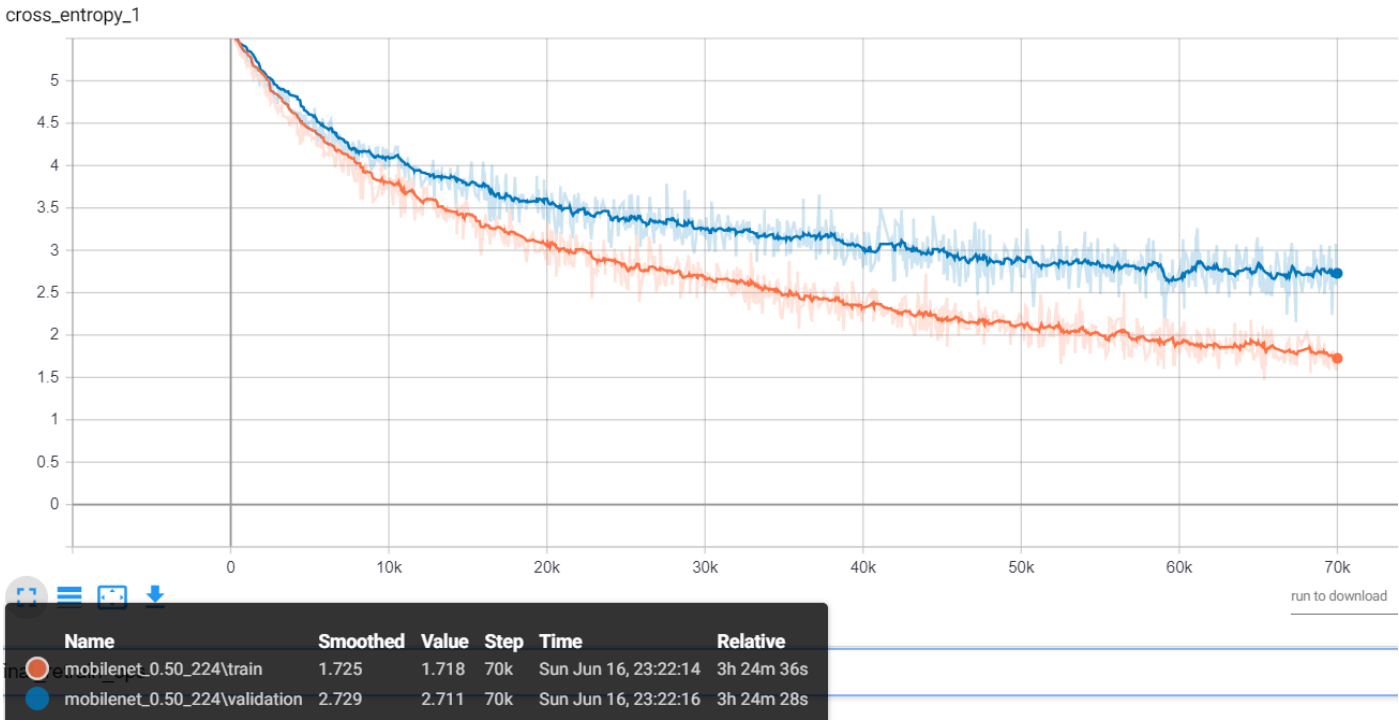
II) N=70.000 pași

a. Acuratețea



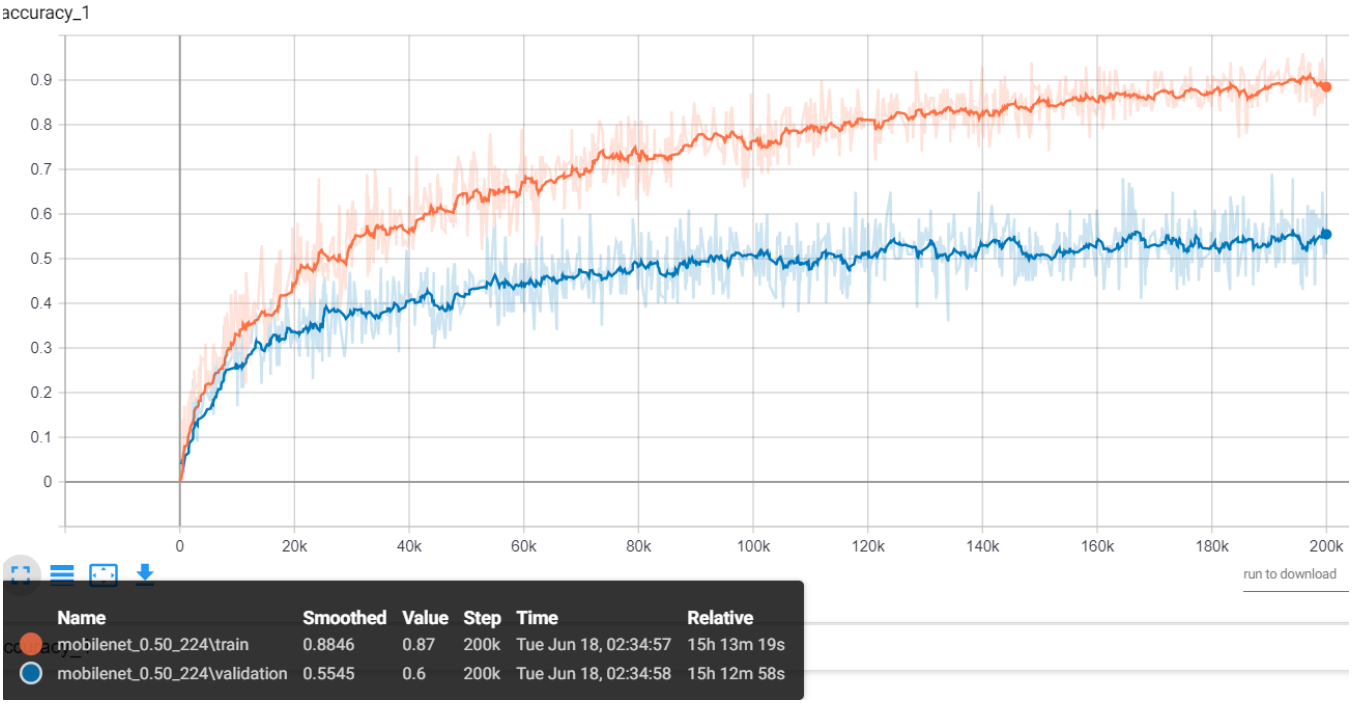
Final test accuracy: 45.1%

b. Pierderea prin entropie



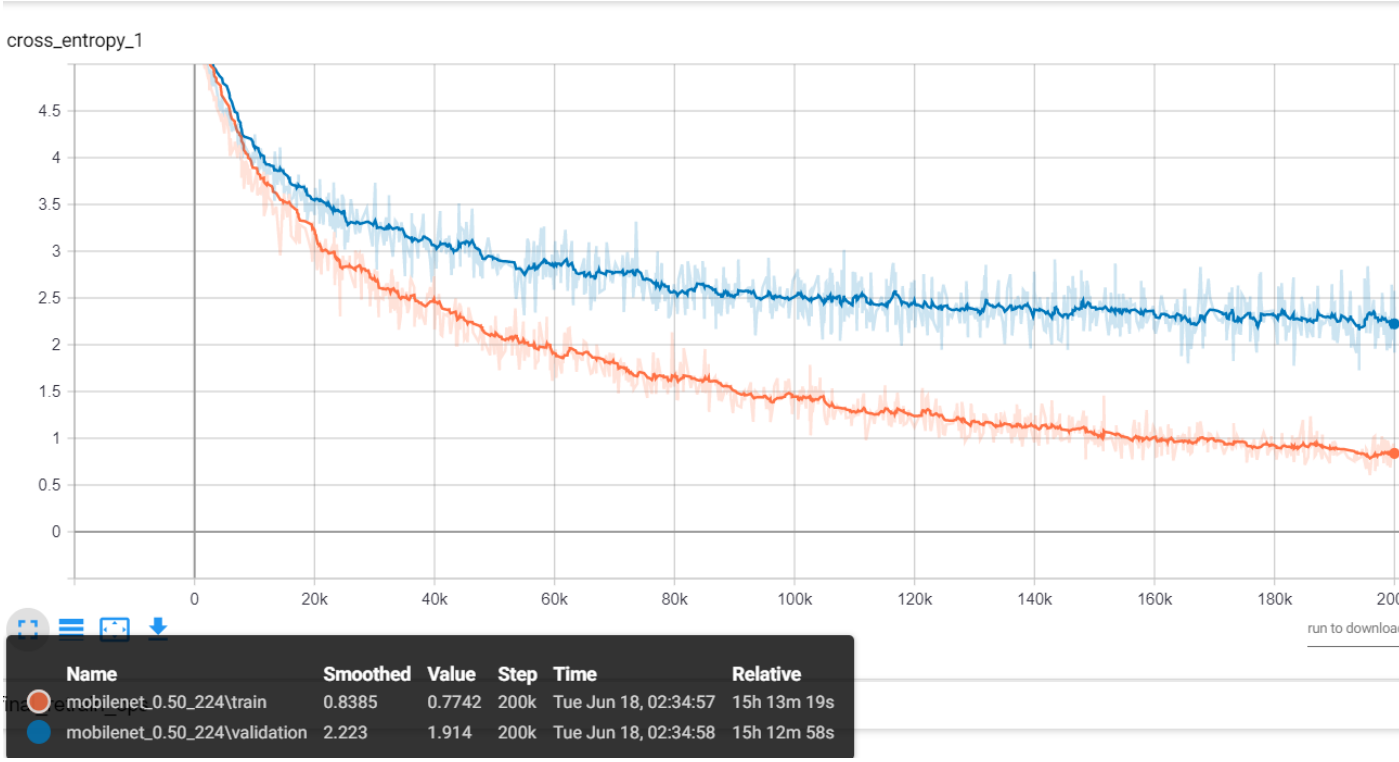
III) N=200.000 pași

a. Acuratețea



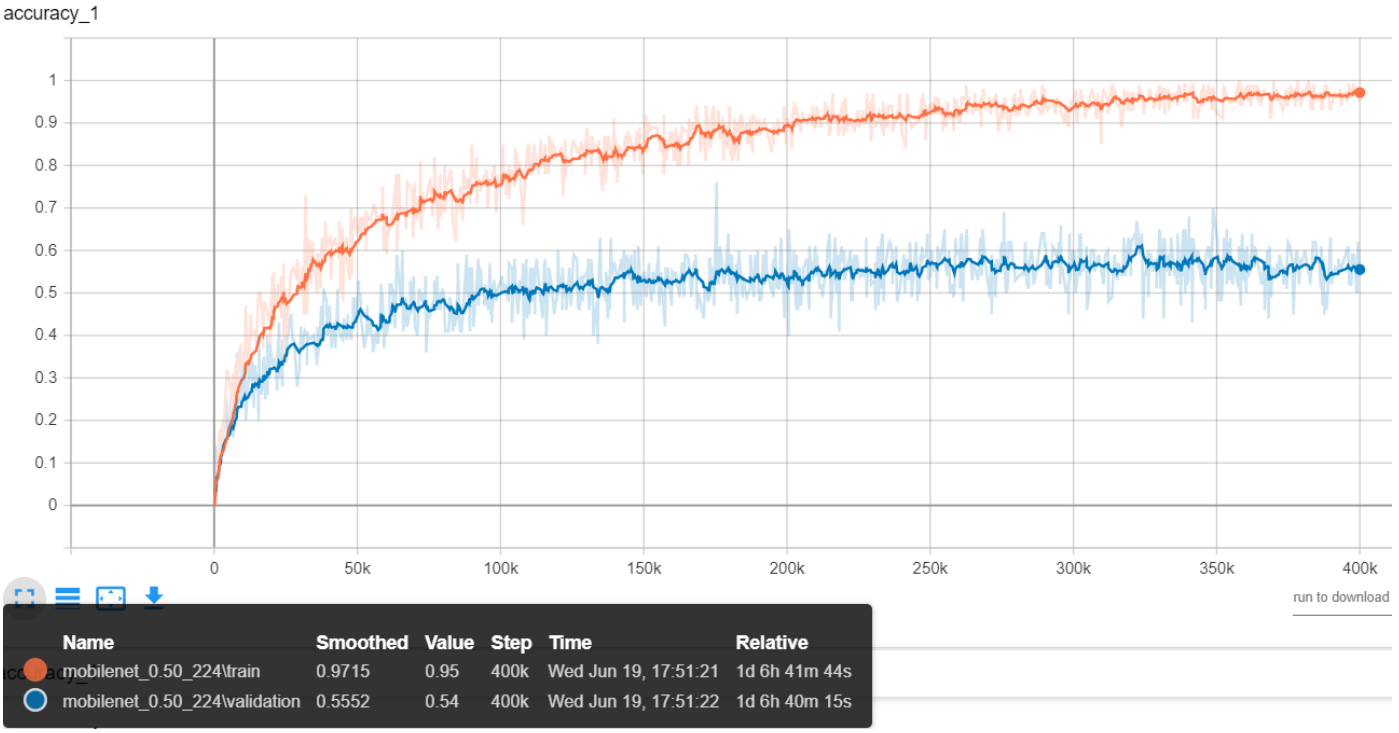
Final test accuracy: 52.7%

b. Pierderea prin entropie



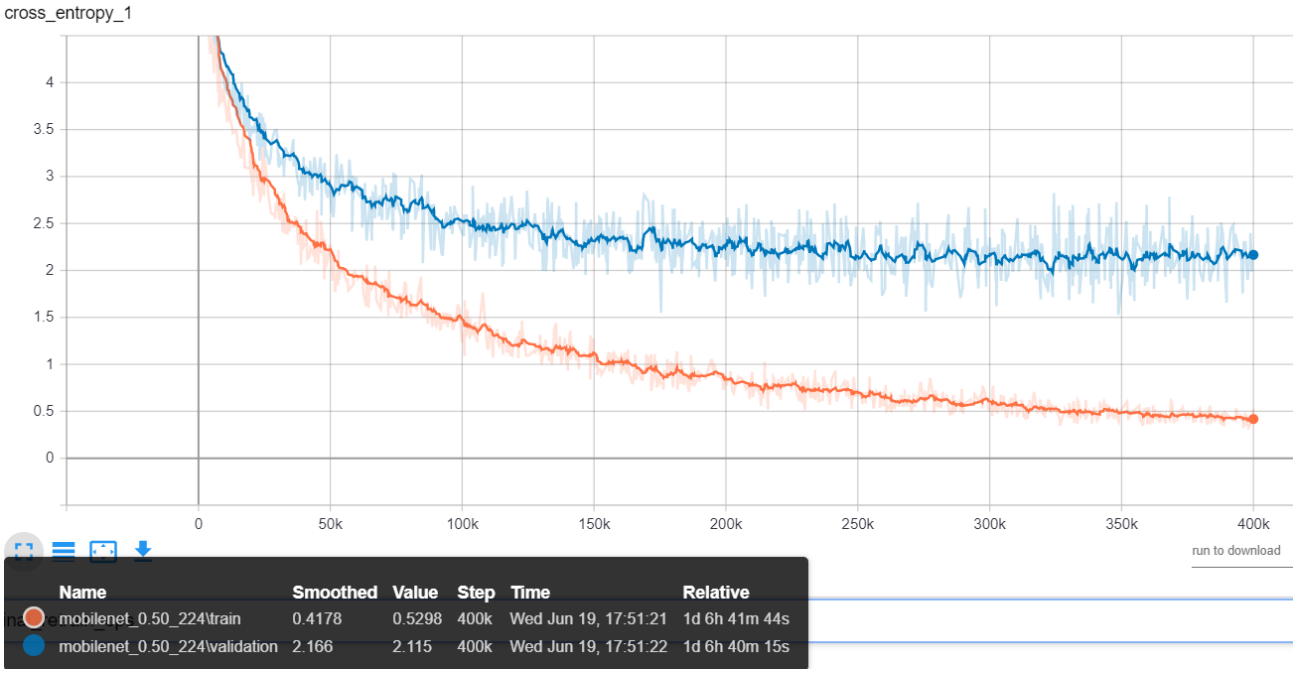
IV) N=400.000 pași

a. Acuratețea



Final test accuracy: 54.5%

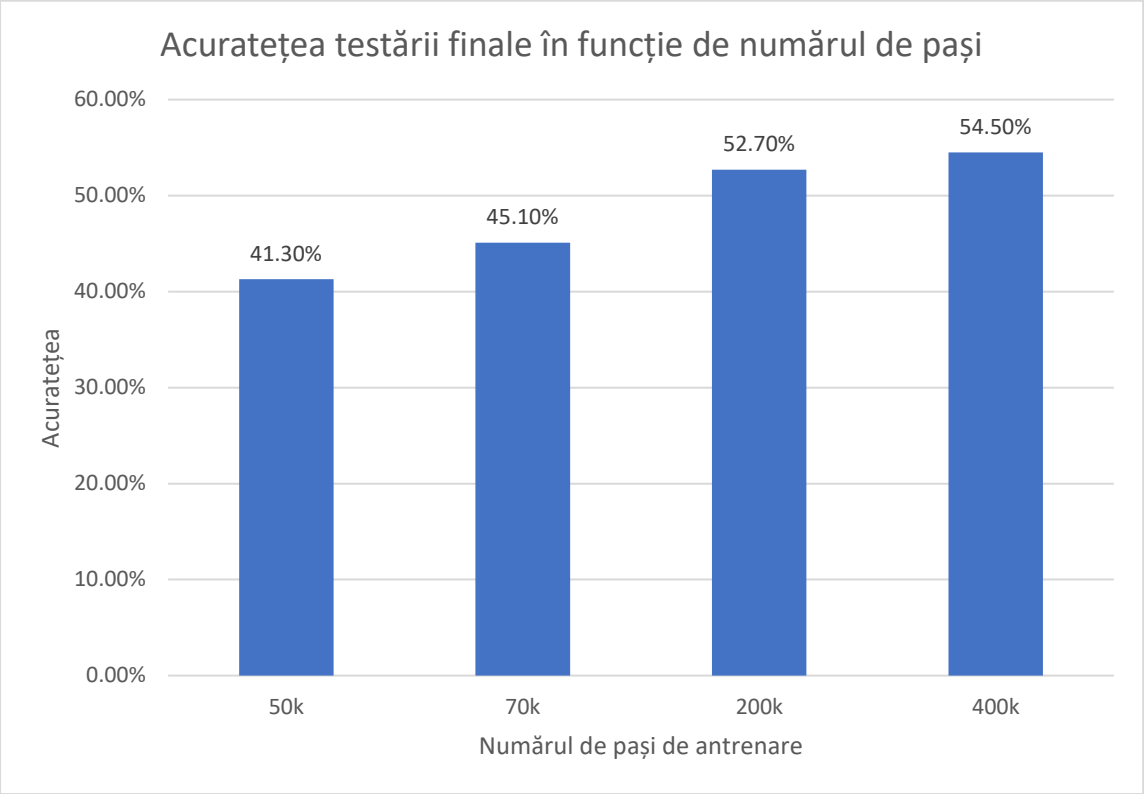
b.Pierderea prin entropie



Se poate observa din graficele de mai sus cum acuratețea atât la antrenare cât și la validare cresc odată cu creșterea numărului de pași. Acuratețea sistemului la testare, de asemenea crește ușor odată cu creșterea numărului de pași de antrenare.

De asemenea, se observă cum pierderea prin entropie scade odată cu creșterea numărului de pași de antrenare și a acurateței, ceea ce este natual.

În continuare, din cauza timpului totuși mare de rulare (aproximativ 24h pentru 400000 de pași), am ales ca celelalte experimente să se efectueze la N=20000 de pași.



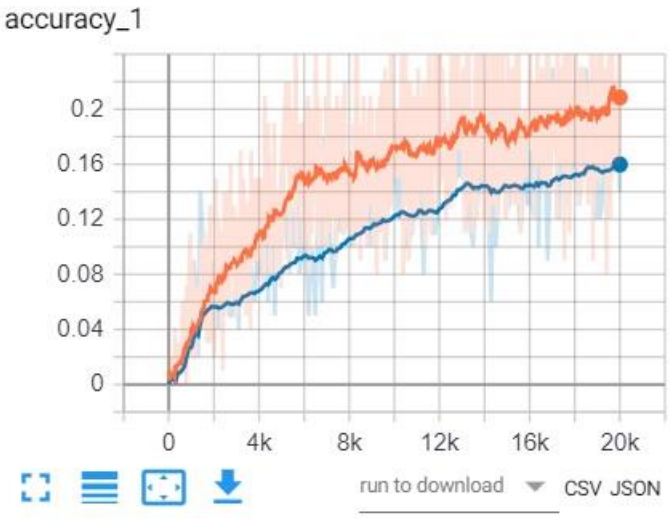
Figură 5.11 - Acuratețea testării finale în funcție de rata de învățare

5.3.2 Variația ratei de învățare (N=20000 de pași)

Pentru reducerea timpului de rulare, s-a ales ca număr de pași de antrenare N=20000, în ciuda rezultatelor slabe obținute în experimentul anterior.

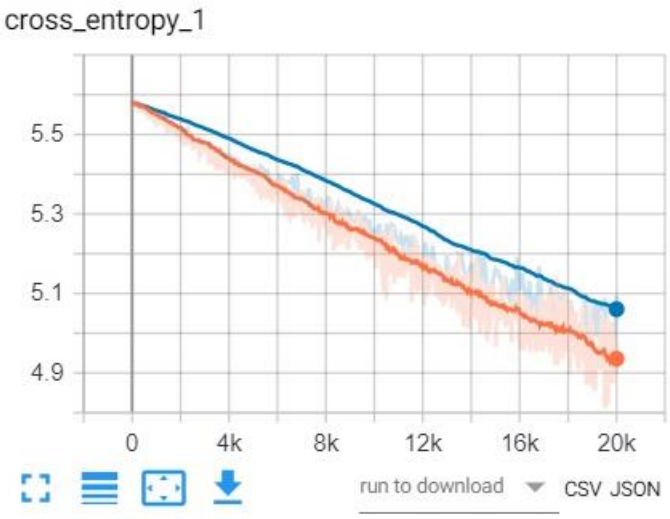
I) learning_rate = 0.001

a. Acuratețea



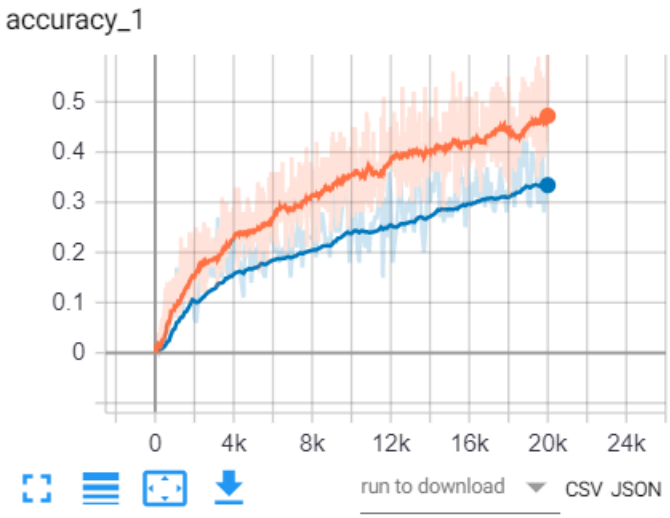
Final test accuracy: 15.2%

b.Pierderea prin entropie



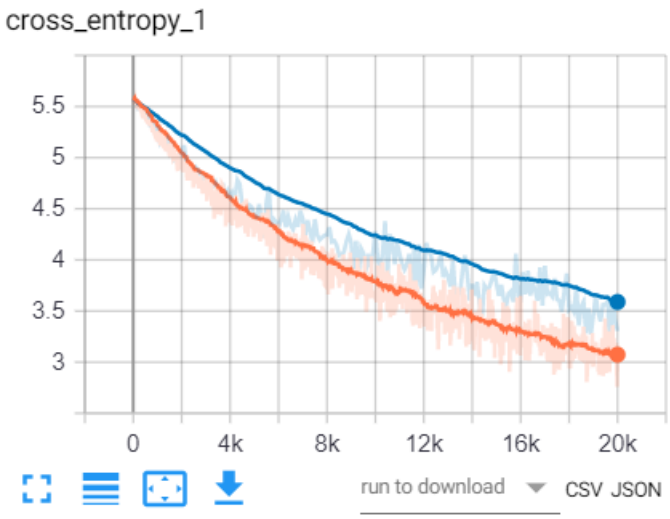
II) learning_rate = 0.01

a. Acuratețea



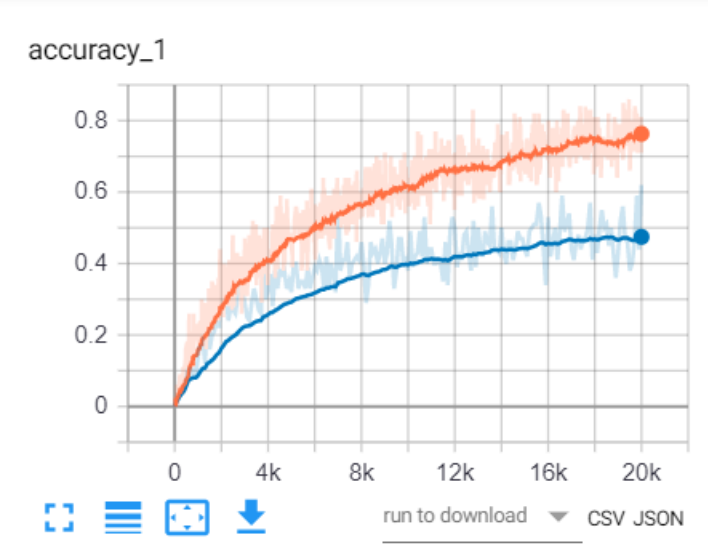
Final test accuracy: 31.7%

b.Pierderea prin entropie



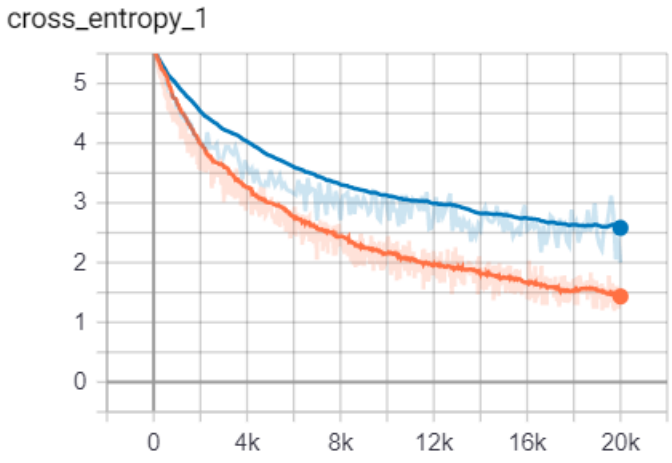
III) learning_rate = 0.05

a. Acuratețea



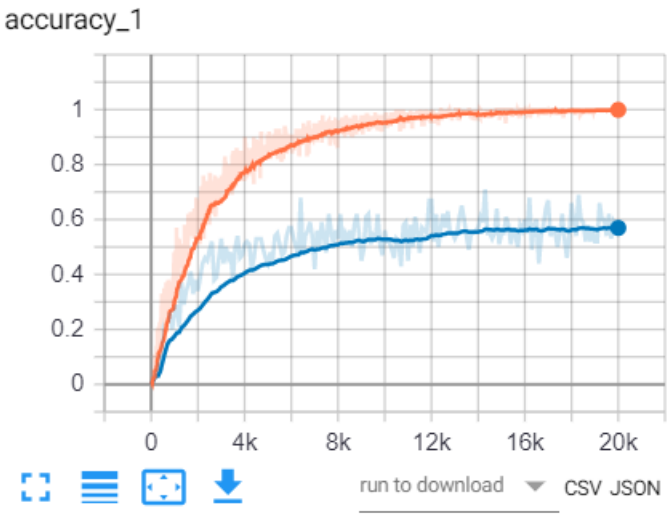
Final test accuracy: 46.5%

b.Pierderea prin entropie



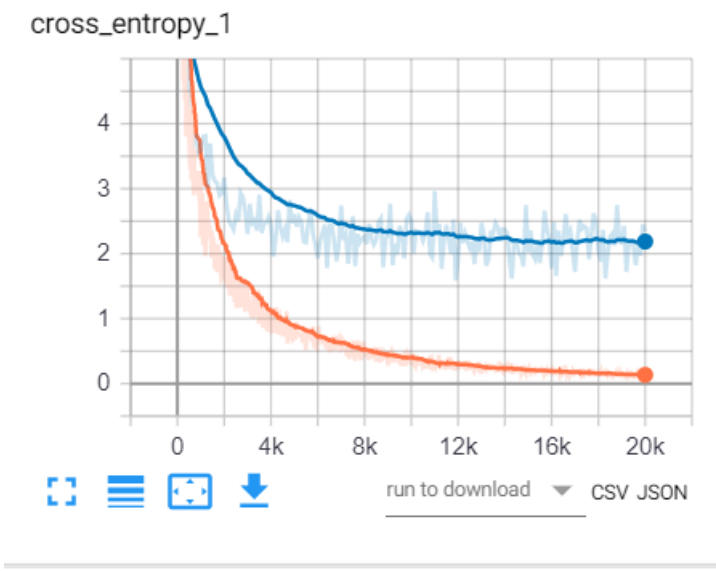
IV) learning_rate = 0.5

a. Acuratețea



Final test accuracy: 55.5%

b.Pierderea prin entropie

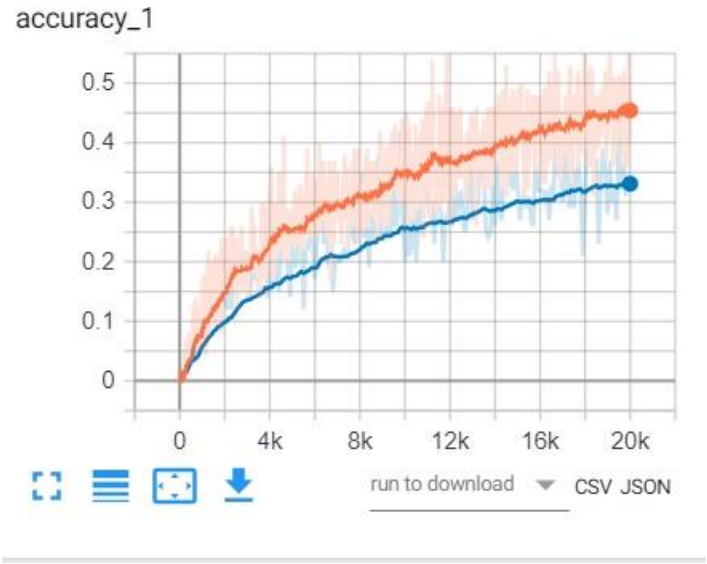


Din graficele de mai sus putem concluziona că numărul de pași a fost insuficient pentru a putea decide care este valoare optimă a ratei de învățare, însă putem observa de asemenea, că deși acurateața avea o direcție ascendentă în toate cele 4 cazuri, în primul caz (lr=0.001), panta este cea mai abruptă, deci cu șanse mai mari de creștere, odată cu creșterea numărului de pași de antrenare.

5.3.3 *Variația dimensiunii batch-ului (N=20000 de pași)*

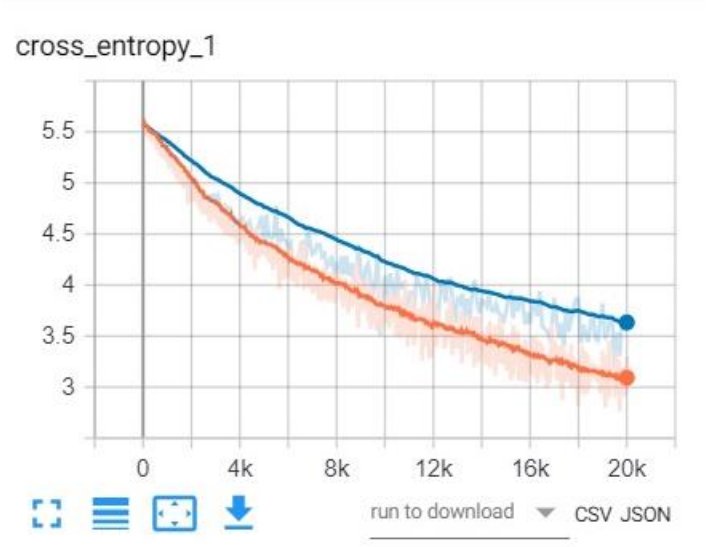
I) batch_size = 50

a. Acuratețea



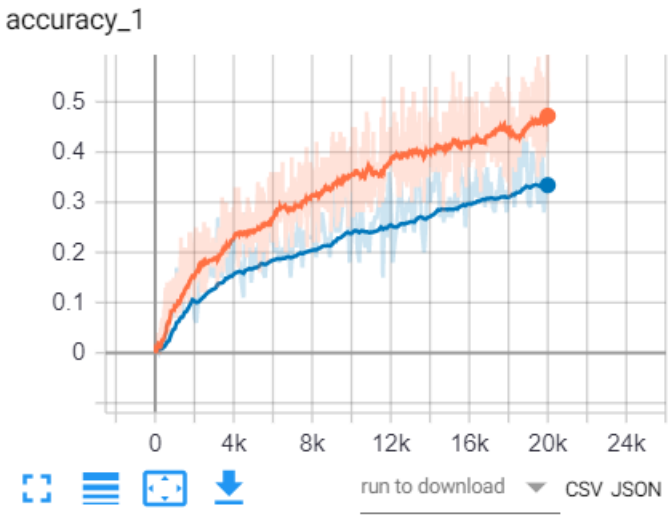
Final test accuracy: 31.5%

b.Pierderea prin entropie



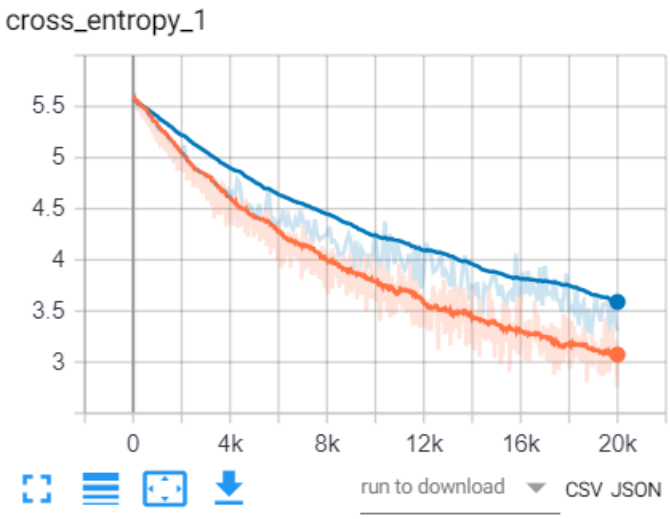
II) batch_size = 100

a. Acuratețea



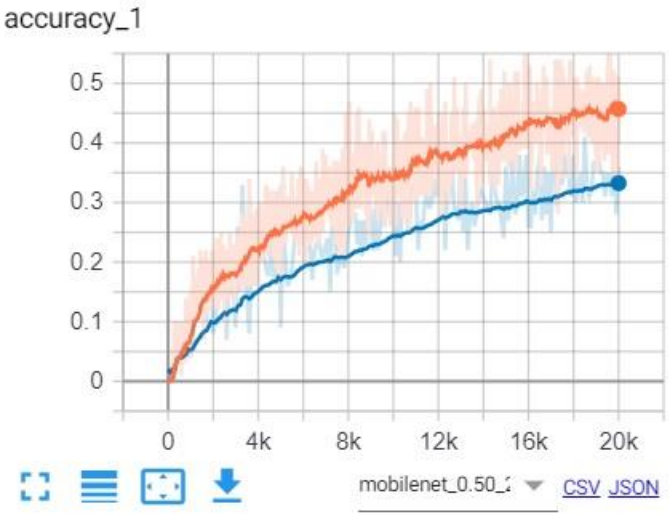
Final test accuracy: 31.7%

b.Pierderea prin entropie



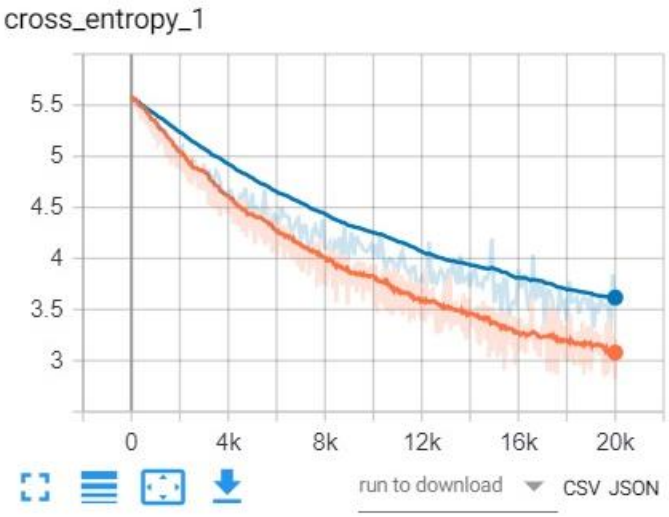
III) batch_size = 200

a. Acuratețea



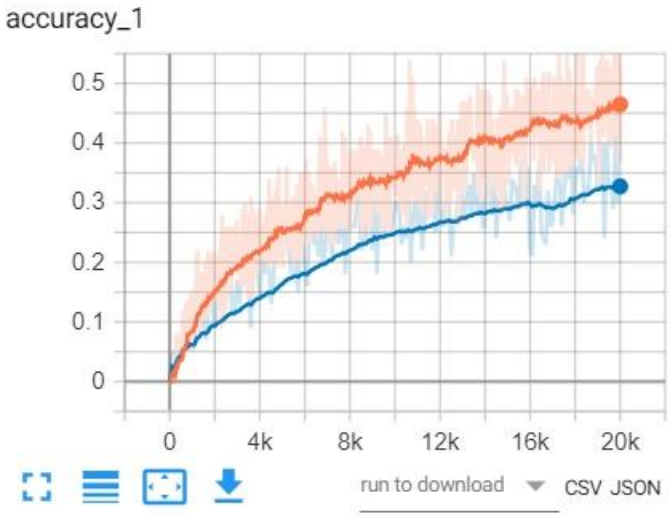
Final test accuracy: 31.8%

b.Pierderea prin entropie

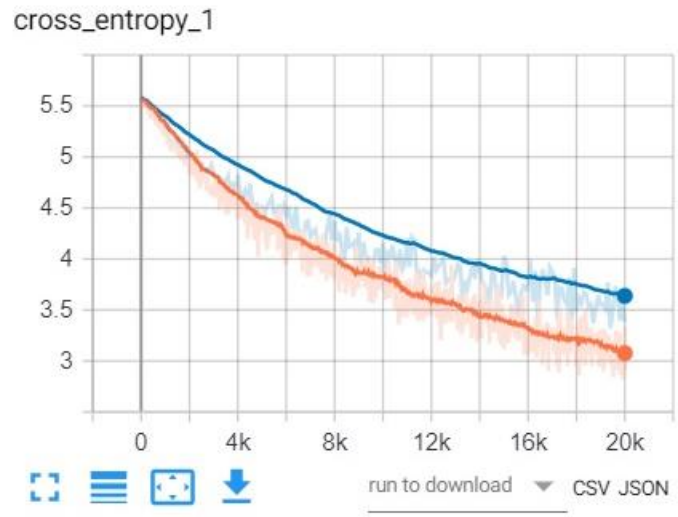


IV) batch_size = 500

a. Acuratețea

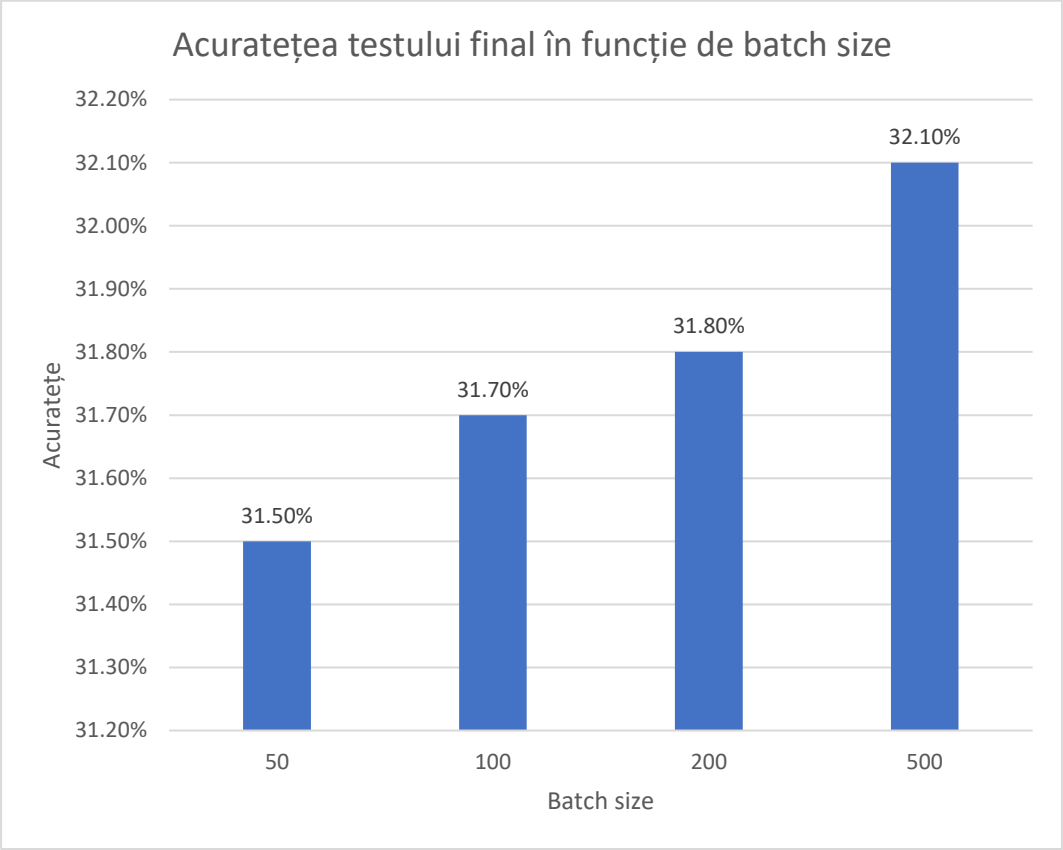


b.Pierderea prin entropie



Final test accuracy: 32.1%

Ca și în cazul anterior, din graficele de mai sus putem concluziona că numărul de pași a fost insuficient pentru a putea decide care este valoare optimă a batch-size-ului.



Figură 5.12 - Acuratețea testării finale în funcție de batch size

CAPITOLUL 6

CREAREA APLICAȚIEI

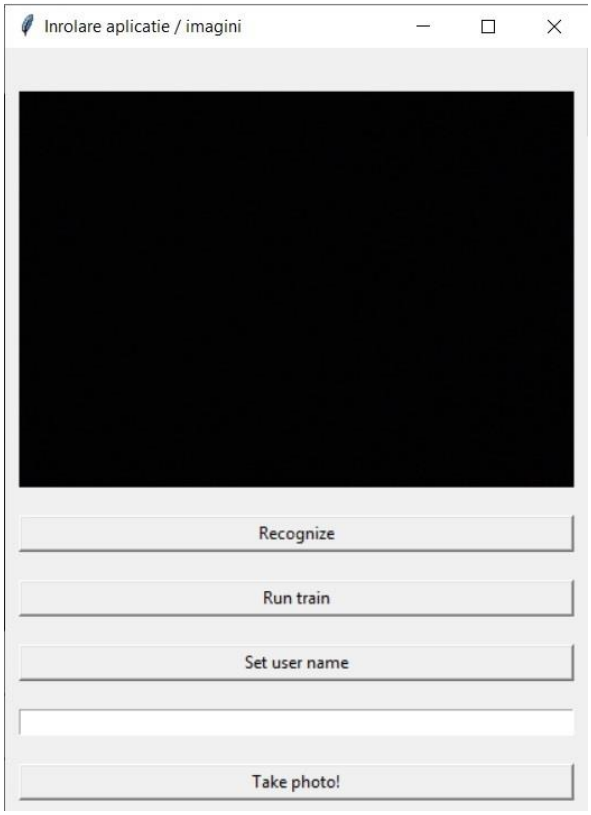
6.1 APLICAȚIA DE ÎNROLARE ȘI AUTENTIFICARE

Aplicația constă în preluarea imaginilor frame cu frame, în timp real, de la camera video a laptopului de lucru. Se va detecta fața din imagine, iar apoi folosind modelul obținut după antrenare, se trece la recunoașterea feței din imagine și afișarea pe ecran a probabilității ca persoana din imagine să fie una dintre cele înregistrate în baza de date. Dacă una dintre probabilități este mai mare de 80%, atunci persoana respectivă se consideră recunoscută de sistem și poate primi acces spre exemplu într-o clădire, sau în alt sistem în care ar putea fi implementată.

Această aplicație este contruită în totalitate în python și are la bază biblioteca tensorflow.

Interfața grafică este una simplă, folosind un set de butoane pentru a determina diferite acțiuni și un display al imaginii capturate de la camera web.

În mod evident o aplicație comercială ar necesita mult mai multe caracteristici si funcții însă nu acesta este scopul acestei lucrări.



Figură 6.1 Intergața grafică a aplicației

Aplicația va juca rol atât de înrolare cât și de autentificare.

Înrolarea se va face prin setarea unui nume de utilizator urmată de apăsarea butonului *Set user name,* acest lucru va determina crearea unui director cu numele utilizatorului și salvarea pozei în interiorul acestuia.

Apoi se va apăsa butonul Take photo pentru a captura imaginea și a o salva în director.

(Baza de date)-

-(Utilizator 1)

-imagine1

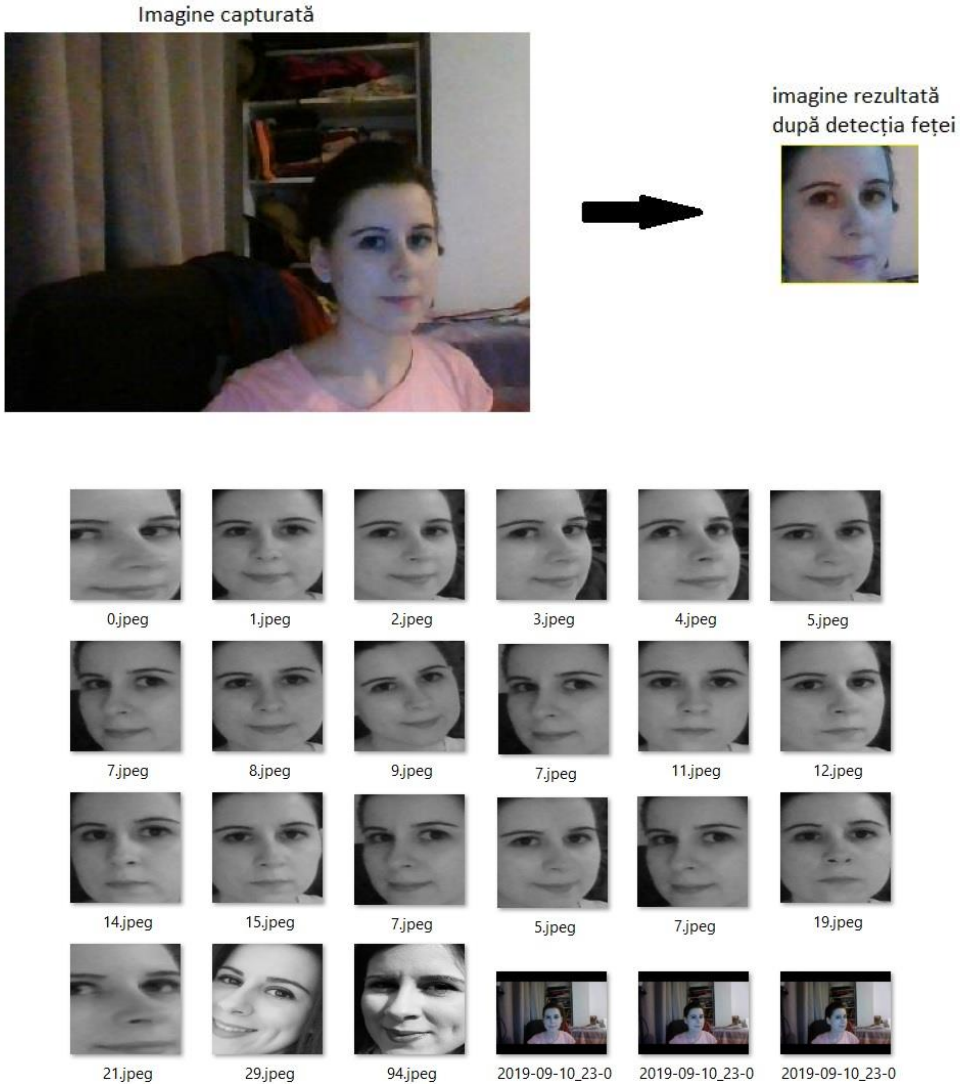
-imagine2

-(Utilizator 2)

-imagine1

-imagine2

Pozele salvate în directoarele personale vor deveni baza de date de antrenare, însă înainte de a putea porni o antrenare este necesară pre-procesarea imaginilor (transformare în imagine de luminanță, modificarea dimensiunilor dar și extragerea feței din imagine folosind un detector de fețe (Haar Cascade))



Figură 6.2 Construirea bazei de date

Pre-procesarea se realizează in mod automat după apăsarea butonului Run train. După ce această etapă este completă se va trece la pasul de antrenare. Antrenarea se realizează în același mod in care a fost prezentat și în capitolele anterioare (folosind aceeași arhitectura proprie de rețea convoluțională).

```
epoch 1 ----- acuratete antrenare:  50.0%, acuratete validare:   0.0%,  cost: 1.175
epoch 2 ----- acuratete antrenare:  50.0%, acuratete validare:   0.0%,  cost: 1.672
epoch 3 ----- acuratete antrenare: 100.0%, acuratete validare:   0.0%,  cost: 1.355
epoch 4 ----- acuratete antrenare: 100.0%, acuratete validare:  50.0%,  cost: 1.236
epoch 5 ----- acuratete antrenare: 100.0%, acuratete validare:  50.0%,  cost: 1.020
epoch 6 ----- acuratete antrenare: 100.0%, acuratete validare:  50.0%,  cost: 0.731
epoch 7 ----- acuratete antrenare: 100.0%, acuratete validare:  50.0%,  cost: 0.430
epoch 8 ----- acuratete antrenare: 100.0%, acuratete validare: 100.0%,  cost: 0.201
epoch 9 ----- acuratete antrenare: 100.0%, acuratete validare: 100.0%,  cost: 0.086
epoch 10 ----- acuratete antrenare: 100.0%, acuratete validare: 100.0%,  cost: 0.039
epoch 11 ----- acuratete antrenare: 100.0%, acuratete validare: 100.0%,  cost: 0.018
epoch 12 ----- acuratete antrenare: 100.0%, acuratete validare: 100.0%,  cost: 0.012
this is it
```

Figură 6.3 Exemplu iterații de antrenare

Numărul de pași de antrenare se poate varia (ca default însă se folosește un break în cazul obținerii unor valori bune de acurateţe în mod repetat).

Modelul se va salva ulterior într-un director împreună cu rezultatele de antrenare și validare.

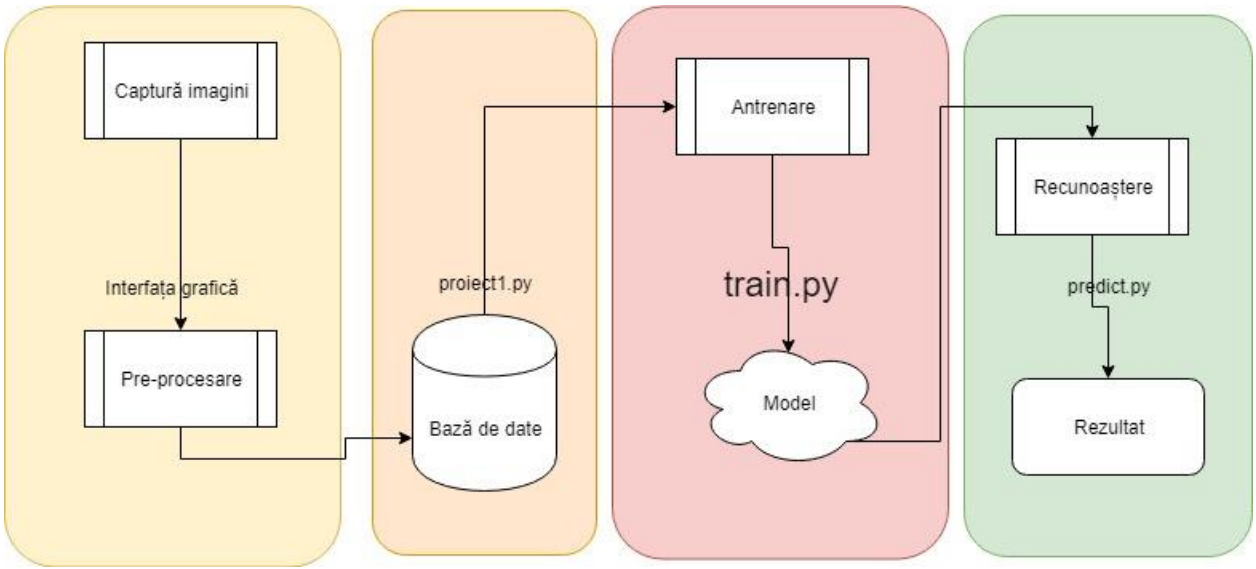
Apăsarea butonului Recognize, va determina restaurarea modelului antrenat anterior și utilizarea imaginii curente ca input în rețea.

Rezultatul va fi apoi afișat împreună cu probabilitatea de adevăr returnată.

```
Predicted is  Adela Ariton probability:  0.8813192
Recognizing...
Predicted is  Adela Ariton probability:  0.9750318
```

Figură 6.4 Afișarea rezultatului

Diagrama de execuție a aplicației arată în felul următor:



Figură 6.5 Diagrama de execuție a aplicației

ANEXE

CODUL SURSĂ AL REȚELEI PROPRII:

Codul sursă al rețelei neuronale:

```
import dataset
import tensorflow as tf
import time
from datetime import timedelta
```

```
import math
import random
import numpy as np
import os

from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(2)




classes = os.listdir('training_data')
num_classes = len(classes)
# set de validare procent
validation_set_value = 0.1
#dim imaginii
img_size = 224
#nr canale
num_channels = 3
train_dataset='training_data'
batch_size = 100

#încărcare date pentru antrenare
data = dataset.read_data_for_training(train_dataset, img_size, classes,
validation_set_value=validation_set_value)

#pornire sesiune tensorflow
session = tf.Session()

#tensorflow file writer pentru a putea salva date pentru tensorboard

train_writer = tf.summary.FileWriter( './logs', session.graph)
#placeholder input
x          =          tf.placeholder(tf.float32,          shape=[None,
img_size,img_size,num_channels], name='x')
```

```
## label-uri si placeholder label
y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)

##Parametrii straturilor convoluţionale
dim_filter_c1 = 3
nr_filters_c1 = 32

dim_filter_c2 = 3
nr_filters_c2 = 32

dim_filter_c3 = 3
nr_filters_c3 = 64

fc_size = 128

def create_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))

def create_biases(size):
    return tf.Variable(tf.constant(0.05, shape=[size]))



def conv(input,
              num_input_channels,
              conv_filter_size,
              num_filters):
    ## Crearea ponderilor ce urmeaza a fi antrenate (se vor folosi valori aleatoare trunchiate ce urmaresc o distribuţie normală de deviaţie standard 0.05)

    weights = create_weights(shape=[conv_filter_size, conv_filter_size, num_input_channels, num_filters])
    ## Bias - valori ce vor fi antrenare - initializate ca constane de valoare 0.05
```

```
    biases = create_biases(num_filters)

    ## Crearea stratului convoluțional
    layer = tf.nn.conv2d(input=input,
                         filter=weights,
                         strides=[1, 1, 1, 1],
                         padding='SAME')

    layer += biases

    ##Folosim max pooling dupa layer-ul convoluțional
    layer = tf.nn.max_pool(value=layer,
                           ksize=[1, 2, 2, 1],
                           strides=[1, 2, 2, 1],
                           padding='SAME')
    ## Ieșirea stratului de pooling va fi trimisă catre funcția de
activare ReLU
    layer = tf.nn.relu(layer)

    return layer



def create_flatten_layer(layer):
    #Shape-ul stratului este dat de [batch_size img_size img_size
num_channels]
    # se poate folosi shape-ul stratului anterior
    layer_shape = layer.get_shape()



    num_features = layer_shape[1:4].num_elements()

    ## stratul va fi flattened pentru a putea extrage vectorul de
caracteristici
    layer = tf.reshape(layer, [-1, num_features])

    return layer
```

```
##Crearea stratului dens
def create_fc_layer(input,
             num_inputs,
             num_outputs,
             use_relu=True):

    #Definirea ponderilor și bias-urilor antrenabile
    weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)

    # Stratul fully connected ia intrarea x și produce wx+b . Din moment
ce acestea sunt matrici, folosim funcția matmul din TensorFlow
    layer = tf.matmul(input, weights) + biases
    if use_relu:
        layer = tf.nn.relu(layer)

    return layer

#Arhitectura finală a rețelei
layer_conv1 = conv(input=x,
               num_input_channels=num_channels,
               conv_filter_size=dim_filter_c1,
               num_filters=nr_filters_c1)
layer_conv2 = conv(input=layer_conv1,
               num_input_channels=nr_filters_c1,
               conv_filter_size=dim_filter_c2,
               num_filters=nr_filters_c2)

layer_conv3= conv(input=layer_conv2,
               num_input_channels=nr_filters_c2,
               conv_filter_size=dim_filter_c3,
               num_filters=nr_filters_c3)

layer_flat = create_flatten_layer(layer_conv3)

layer_fc1 = create_fc_layer(input=layer_flat,
```

```
num_inputs=layer_flat.get_shape()[1:4].num_elements(),
                     num_outputs=fc_size,
                     use_relu=True)

layer_fc2 = create_fc_layer(input=layer_fc1,
                     num_inputs=fc_size,
                     num_outputs=num_classes,
                     use_relu=False)

y_pred = tf.nn.softmax(layer_fc2,name='y_pred')

y_pred_cls = tf.argmax(y_pred, dimension=1)
session.run(tf.global_variables_initializer())
cross_entropy                                                    =
tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2,
                                                    labels=y_true)

cost = tf.reduce_mean(cross_entropy)
tf.summary.scalar('cost' , cost)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-2).minimize(cost)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', accuracy)
val_accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('val_accuracy', val_accuracy)

merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter( 'summaries/train', session.graph)
test_writer = tf.summary.FileWriter('summaries/test')
session.run(tf.global_variables_initializer())



def show_progress(epoch, feed_dict_train, feed_dict_validate, val_loss,
i, merge):
    summary,      acc      =      session.run([merge,      accuracy],
feed_dict=feed_dict_train)
```

```
    summary,     val_acc     =     session.run([merge,     accuracy],
feed_dict=feed_dict_validate)
    train_writer.add_summary(summary, i)
    msg = "Training Epoch {0} --- Training Accuracy: {1:>6.1%},
Validation Accuracy: {2:>6.1%}, Validation Loss: {3:.3f}"
    print(msg.format(epoch + 1, acc, val_acc, val_loss))


total_iterations = 0


saver = tf.train.Saver()
def train(num_iteration):
    global total_iterations


    for i in range(total_iterations,
                   total_iterations + num_iteration):



        x_batch,       y_true_batch,       _,       cls_batch       =
data.train.next_batch(batch_size)
        x_valid_batch,    y_valid_batch,    _,    valid_cls_batch    =
data.valid.next_batch(batch_size)



        feed_dict_tr = {x: x_batch,
                        y_true: y_true_batch}
        feed_dict_val = {x: x_valid_batch,
                         y_true: y_valid_batch}


        session.run(optimizer, feed_dict=feed_dict_tr)


        if i % int(data.train.num_examples/batch_size) == 0:
            summary,    val_loss    =    session.run([merged,    cost],
feed_dict=feed_dict_val)
            train_writer.add_summary(summary, i)
            epoch = int(i / int(data.train.num_examples/batch_size))


            #show_progress(epoch,     feed_dict_tr,     feed_dict_val,
val_loss, i, merge)
```

```
            summary,    acc    =    session.run([merged,    accuracy],
feed_dict=feed_dict_tr)
            train_writer.add_summary(summary, i)
            summary,  val_acc  =  session.run([merged,  val_accuracy],
feed_dict=feed_dict_val)
            train_writer.add_summary(summary, i)
            msg = "Training Epoch {0} --- Training Accuracy: {1:>6.1%},
Validation Accuracy: {2:>6.1%},  Validation Loss: {3:.3f}"
            print(msg.format(epoch + 1, val_acc, val_acc, val_loss))
            saver.save(session, 'faces')



    total_iterations += num_iteration


train(num_iteration=1000)
```

REFERINȚE

[1] Stuart Russell and Peter Norvig, 2010 “Artificial Intelligence A Modern Approach, Third Edition”, ISBN-13: 978-0-13-604259-4

[2] http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[3] Shuangsang Fang, Yuehui Zhao, Peng Wang, Jun Zhang

”Implementation of Training Convolutional Neural Networks Tianyi

Liu”, University of Chinese Academy of Sciences, Beijing, China

[4] https://www.coursera.org/specializations/deep-learning

[5] https://www.cognitoforms.com/ADSC2/FaceScrubDatasetPasswordRequest

[6] https://www.whatismybrowser.com/detect/what-is-my-user-agent

[7] H.-W. Ng, S. Winkler. A data-driven approach to cleaning large face datasets.Proc. IEEE International Conference on Image Processing (ICIP), Paris, France, Oct. 27-30, 2014.

[8] https://www.facefirst.com/blog/brief-history-of-face-recognition-software

[9] http://vintage.winklerbros.net/facescrub.html

[10] Yosinski J., Clune J., Bengio Y. and Lipson H., 2014 „ How transferable are fetures in deep neural networks?” , pages 3320-3328 („Advances in neural information processing systems”).

[11] Huh M., Agrawal P., and Efros A. A., 2016, „What makes Imagenet good for transfer learning?”, arXiv preprint arXiv:1608.08614.

[12] https://www.datacamp.com/community/tutorials/tensorflow-tutorial

[13] Ch. Szegedy, S. Ioffe, V. Vanhoucke. Alexander A. Alemi, 2015, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”, Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

[14] https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

[15] https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86