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

Recunoașterea expresiilor faciale similare

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul Electronică, Telecomunicații și Tehnologia Informației
programul de studii de licență Microelectronică, Optoelectronică și Nanotehnologii

Conducător științific
Prof. Dr. Ing. Corneliu FLOREA

Absolvent
Andreea-Alexandra VINTILESCU

Anul 2023

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

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului **VINTILESCU I. Andreea-Alexandra, 442E**

1. Titlul temei: Recunoasterea expresiilor faciale similare

2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):
Se consideră o solutie bazată pe rețele convoluționale adânci pentru recunoașterea expresiilor similare în imagini pe baza de date "Google facial expression comparison dataset". Scopul aplicatiei dezvoltate este ca dându-se o pereche de poze cu expresii faciale, să se precizeze automat dacă cele două sunt similare sau diferite. Se va modifica arhitectura rețelei convolutionale, funcția obiectiv si alți hiperparametri, astfel încât să se optimizeze performanța. Proiectul se implementează în Python, si se folosește libraria Pytorch pentru utilizarea rețelelor convoluționale

3. Discipline necesare pt. proiect:
ISIA, DEPI, POO

4. Data înregistrării temei: 2022-12-06 11:43:29

Conducător(i) lucrare,
Prof. dr. ing. Corneliu FLOREA

Student,
VINTILESCU I. Andreea-Alexandra

Director departament,

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **203b74109a**

Anexa 5

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "Recunoașterea expresiilor faciale similare", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Inginerie Electronică, Telecomunicații și Tehnologia Informației,* programul de studii *Microelectronică, Optoelectronică și Nanotehnologii*, este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

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

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

București, 27.06.2023

Absolvent *Andreea-Alexandra VINTILESCU*

_____

(semnătura în original)

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

ADAM	Adaptive Moment Estimation	Estimarea Momentului Adaptiv
ANN	Artificial Neural Network	Rețele Neuronale Artificiale
BCELoss	Binary Cross-Entropy Loss	Pierderea Binară A Entropiei Încrucișate
CNN	Convolutional Neural Network	Rețele Neuronale Convoluționale
CPU	Central Processing Unit	Unitate Centrală De Procesare
CSV	Comma-Separated Values	Valori Separate prin Virgulă
CUDA	Compute Unified Device	Dispozitiv De Calcul Unificat
cuDNN	Cuda Deep Neural Network	Rețea Neurală Adâncă Cuda
FC	Fully Connected	Strat Conectat Complet
GPU	Graphics Processing Unit	Unitate De Procesare Grafică
ReLU	Rectified Linear Unit	Unitate Liniară Rectificată
SGD	Stochastic Gradient Descent	Coborâre Cu Gradient Stochastic
SNN	Siamese Neural Network	Rețea Neurală Siameză

Introducere

Lucrarea prezentă ilustrează o soluție pentru recunoașterea expresiilor faciale similare într-un set de imagini, scopul aplicației fiind precizarea automată a similitudinii celor două poze, utilizând tehnologii de inteligență artificială.

Analiza de expresii faciale a primit în ultima perioadă tot mai multă atenție datorită numeroaselor sale aplicații în domenii variate. Un prim exemplu ar fi folosirea în domeniul medical pentru detectarea autismului, precum și în prevenirea bolilor psihice sau a depresiei. S-a demonstrat clinic faptul că persoanele diagnosticate cu tulburare de spectru autist pot avea dificultăți în a-și exprima emoțiile prin expresii faciale și chiar în a interpreta expresiile faciale ale altor oameni. Astfel, analiza expresiilor faciale poate fi utilă în identificarea abilităților și comportamentelor care necesită dezvoltare suplimentară. În plus, aceasta poate fi o unealtă utilă și în procesul de intervenție și terapie, întrucât le oferă un feedback asupra comportamentelor lor, ajutându-i să-și dezvolte abilitățile de comunicare și cele sociale.

Un alt exemplu îl reprezintă cel din domeniul securității. Mai exact, analiza expresiilor faciale este utilizată în securitate pentru identificarea persoanelor căutate de autorități sau persoanelor care reprezintă un risc pentru cei din jur. Altfel spus, aceasta poate fi folosită pentru identificarea infractorilor, teroriștilor sau a persoanelor care încalcă legile, cât și în detectarea comportamentelor suspecte precum agresivitatea sau nervozitatea excesivă, stări emoționale ce preced unor atacuri de terorism sau violență în zone publice.

Consider că acest subiect este unul ce merită studiat deoarece utilizarea rețelelor neuronale în privința învățării autonome devine tot mai răspândită în măsura în care acestea sunt capabile să învețe, să acționeze pe baza experiențelor anterioare și să ia o decizie în cazul unor situații neprevăzute. Astfel, abilitatea rețelelor neuronale de a învăța din propria experiență fac această tehnologie să fie tot mai răspândită. Odată cu apariția primului computer, oamenii de știință au luat în calcul posibilitatea realizării unei inteligențe artificiale capabile de a lua decizii pe baza unor reguli bine stabilite, iar mai târziu pe baza experienței.

Din punct de vedere structural, o astfel de tehnologie se aseamănă foarte mult creierului uman ce conține și el, asemeni rețelelor neuronale, o rețea complexă de neuroni interconectați. Neuronii pot forma legături multiple cărora le sunt asociate niște ponderi. Informația necesară va parcurge legăturile realizate, însă numai într-un singur sens. Astfel, fiecare neuron primește o valoare de intrare pe care o înmulțește cu ponderea asociată fiecărei legături, comparându-se mai apoi valoarea obținută cu una de referință. Dacă aceasta este mai mare decât cea de referință, valoarea obținută se transmite mai departe următorului nod drept valoare de intrare, în caz contrar, nu se va transmite. Acest proces se repetă până la parcurgerea ultimului strat prezent în rețeaua neuronală [2].

În acest proiect a fost necesară înțelegerea conceptelor de rețele convoluționale adânci pentru modificarea, antrenarea și testarea modelelor preantrenate puse la dispoziție pe PyTorch. Rezultatele obținute ilustrând importanța acestei abordări și o soluție potrivită situațiilor prezentate mai sus.

Primul pas în realizarea proiectului a fost cel al antrenării unei baze de date de la Google, care a fost, însă, supusă unor mici modificări în privința numărului datelor de intrare. Atenția fiind pusă pe aproximativ 11 000 dublete de expresii faciale din cele 500 000 totale, împreună cu adnotări care specifică dacă cele două poze sunt asemănătoare sau nu. Setul de date inițial, cu adnotările de comparare a expresiilor poate fi descărcat de aici. Cele 11 000 de dublete au fost împărțite, în primă fază, în 80% poze pentru antrenare și 20% destinate validării.

Pe parcursul antrenării, distribuirea pozelor a fost modificată odată cu folosirea, pe rând, a mai multor optimizatori. Într-un final s-au păstrat ponderile de 95% poze de antrenare și 5% poze destinate validării, obținând în cazul de față cele mai bune rezultate.

1. Considerații teoretice

1.1 Machine learning

Machine learning, cunoscută și drept învățarea automată, reprezintă una dintre ramurile inteligenței artificiale ce se concentrează pe dezvoltarea de algoritmi și modele ce permit sistemelor o învățare automată, precum și îmbunătățirea acestora pe baza experiențelor anterioare. Comparativ cu alte tehnologii, algoritmii de învățare automată sunt instruiți să ofere predicții pe baza unui set de date de antrenare. Acest set de date conține exemple în urma cărora algoritmul învață să identifice anumite aspecte și să ofere predicții sau chiar decizii pe baza cunoștințelor dobândite[10].

Un aspect considerabil al tehnologiei machine learning îl reprezintă antrenarea modelelor pe seturi de date destinate învățării. Aceste seturi de date prezintă exemple de intrări dorite antrenării, în cazul lucrării de față fiind reprezentate de perechi de imagini și valori asociate sau etichete ale acestora. Ideea din spatele antrenării modelelor este învățarea acestora în a generaliza și a oferi predicții corecte datelor noi de intrare pe baza a ceea ce ele au învățat din experiențele lor anterioare în antrenare.

În prezent, domeniul de învățare automată este reprezentat de 5 categorii puternic dezvoltate :

- Învățarea supravegheată (Supervised Learning);
- Învățarea nesupravegheată (Unsupervised Learning);
- Reinforcement Learning;
- Deep Learning;
- Deep Reinforcement Learning.

Învățarea supravegheată se bazează pe folosirea de date etichetate. Acest tip de învățare implică furnizarea de exemple de intrare și ieșire corespondente, astfel încât modelul să poată învăța să facă predicții corecte pe baza acestora. Acest tip de învățare este utilizat în multiple aplicații precum a celor de recunoaștere a scrisului de mână [10] .

În comparație cu învățarea supravegheată, învățarea nesupravegheată nu asociază o valoare de ieșire intrării, ci solicită identificarea unor anumite caracteristici comune pentru tot setul de date. În funcție de aplicabilitatea acesteia, specificarea unei valori corecte ieșirii nu poate fi înfăptuită ușor de adnotatori. Acest tip de învățare este utilizat în aplicații pentru detectarea obiectelor, recunoașterea facială, etc [10] .

Reinforcement learning constă în găsirea echilibrului între îndeplinirea soluțiilor găsite și căutarea altor posibile rezolvări. În folosirea acestei ramuri, algoritmului de învățare i se oferă mai multe soluții la întâmplare, ajungând să fie recompensat sau pedepsit pe baza deciziilor făcute [10] .

Astfel, atunci când acesta execută comportamentul dorit este recompensat, iar în cazul unui comportament nedorit - pedepsit, computerul ajungând să diferențieze acțiunile corecte de cele greșite. Acest tip de învățare a fost conceput inițial pentru inteligențe artificiale capabile să joace jocuri [10] .

Deep learning reprezintă un model de învățare inteligentă bazat pe rețele neuronale artificiale (ANN), mai precis pe baza rețelelor neuronale convoluționale (CNN). Acesta se concentrează pe antrenarea rețelelor neuronale artificiale profunde pentru a învăța și a realiza sarcini complexe de prelucrare a datelor. Acest model de învățare se inspiră din funcționarea creierului uman și este folosit, în general, în aplicații de prelucrare a imaginii - viziunea computerizată, recunoașterea vocală, dar și în jocuri [10] .

Deep reinforcement learning este un tip de învățare automată formată din combinarea modelelor de învățare deep learning și reinforcement learning. Împreună, cele două categorii de învățare automată au fost utilizate cu succes în aplicații de robotică, jocuri video, sănătate, finanțe, etc. În prezent această metodă este cercetată de industrii pentru versatilitatea dovedită [10] .

Întrucât tehnicile machine learning sunt diverse și pot reprezenta multiple domenii, acestea produc și provocări în asociere cu utilizarea acestora. Astfel, prelucrarea și gestionarea datelor de antrenare și testare, selectarea algoritmului potrivit, rezolvarea overfitting-ului, precum și interpretarea rezultatelor, pot reprezenta niște aspecte importante ce necesită atenție pentru o implementare corectă și eficientă a tehnicilor machine learning.

În cazul recunoașterii expresiilor faciale similare prin intermediul rețelelor neuronale s-a decis utilizarea tehnicii deep learning, deoarece poate genera soluții optime pentru acest tip de aplicație. Expresiile faciale se pot întâlni într-o infinitate de situații distincte, întrucât față prezintă o cantitate mare de trăsaturi. De aceea abilitatea unui sistem de a percepe, interpreta și acționa asupra acestora, este necesară pentru evaluarea oricărui tip de situație. Împreună cu alte trăsături de învățare bazate pe experiență, metoda deep learning este potrivită pentru scopul prezentei lucrări.
După cum am menționat anterior, deep learning reprezintă una dintre cele cinci ramuri ale învățării automate și se concentrează pe antrenarea rețelelor neuronale precum și pe utilizarea acestora pentru o bună înțelegere și o reușită a extragerii datelor necesare.

Aceste tehnologii de învățare au luat o puternică amploare în ultimii ani și au revoluționat în multiple domenii. Spre exemplu, tehnica deep learning a înregistrat progrese semnificative în domenii precum recunoașterea facială, recunoașterea vocală și multe altele [12] .

În comparație cu celelalte metode, deep learning s-a dovedit a fi mult mai eficientă în rezolvarea problemelor de învățare automată complexe. Una dintre caracteristicile acesteia constă în utilizarea rețelelor neuronale profunde. Se cunoaște faptul că aceste rețele sunt compuse din mai multe straturi ascunse ce permit extragerea caracteristicilor dorite într-o manieră lină, preluarea acestor date fiind făcută treptat. Fiecare strat ascuns reprezentativ rețelei prelucrează datele obținute de la straturile

anterioare și le transmite mai departe, către următorul strat. În această manieră caracteristicile lor fiind prelucrate complex într-un mod ierarhic. Desigur, pentru obținerea unor valori semnificative, este necesară antrenarea unor seturi mari de date etichetate în prealabil și ajustarea iterativă a parametrilor rețelei neuronale utilizând algoritmii de optimizare potriviți [12] .

De asemenea, o altă caracteristică importantă a tehnicii deep learning este reprezentată de utilizarea rețelelor neuronale convoluționale (CNN). Rețelele neuronale convoluționale reprezintă o arhitectură de rețea ce a fost proiectată să învețe și să înțeleagă datele de intrare oferite (imaginile) și să aplice niște filtre și operații de convoluție acestora pentru o identificare mai ușoară a caracteristicilor importante.
Deci, CNN-urile preiau la intrare o imagine și oferă diferitelor aspecte o importanță, cât și o diferență asupra acestora. Rolul lor este de a reduce imaginea într-o formă ușor de procesat, evitând pierderea de caracteristici importante ce ajută la obținerea corectă a predicțiilor [12] .

După cum s-a menționat mai sus, una dintre provocările asociate tehnicii de învățare automată - deep learning este reprezentată de rezolvarea supraantrenării, cunoscută și drept "overfitting". Aceasta reprezintă una dintre comunele probleme ale tehnicii deep learning în care modelul de rețea neuronală devine mult prea specializat pe datele oferite drept antrenare, nereușind astfel o generalizare corectă a datelor noi. Această problemă este întâlnită, deseori, atunci când programul își adaptează excesiv parametrii, eșuând în excluderea zgomotului și a variațiilor semnificative ale acestora. Totuși, pentru evitarea acestei probleme, există mai multe modalități ce pot fi abordate: regularizarea, validarea încrucișată, augmentarea datelor, regularizarea timpurie și tehnica dropout [12] .

- Regularizarea reprezintă una dintre tehnicile ce poate ajuta la controlarea complexităților modelului, precum și la limitarea valorilor extreme ale parametrilor, oferind astfel o reducere a supraantrenării.

- Validarea încrucișată oferă o evaluare mai complexă a performanței modelului prin împărțirea datelor în seturi de antrenare și seturi de validare. Astfel, printr-o evaluare pe multiple seturi de date, această tehnică evită apariția supraantrenării.

- Augmentarea datelor susține extinderea seturilor de date prin aplicarea de multiple transformări precum redimensionarea și rotirea, contribuind astfel în creșterea diversității datelor și prevenirea supraadaptării.

- Regularizarea timpurie constă în monitorizarea performanței modelului și în oprirea prematură a antrenării atunci când performanța se deteriorează, instalându-se problema supraadaptării.

- Tehnica dropout ajută în generalizarea mai bună a modelului prin dezactivarea temporară a unor neuroni în timpul antrenării setului de date.

Pe parcursul anilor, tehnologia deep learning a avansat, în prezent fiind facilitată implementarea și utilizarea rețelelor neuronale profunde. Printre cele mai populare biblioteci și framework-uri, utilizate și în lucrarea de față, se numără și TensorFlow și Pytorch ce ajută la definirea, antrenarea și evaluarea rețelelor neuronale. Vom discuta mai pe larg despre acestea în următorul capitol.

1.2 Rețele neuronale convoluționale

Un CNN reprezintă o arhitectură de rețea cu un algoritm de învățare profundă, utilizat în mod special pentru recunoașterea imaginilor. Astfel, acesta preia la intrare o imagine și poate oferi diferitelor obiecte/aspecte din imagine o importanță, cât și o diferență asupra acestora. Rolul CNN-urilor este de a reduce imaginea într-o formă ușor de procesat, evitând pierderea de caracteristici importante ce ajută la obținerea corectă a predicțiilor. CNN-urile sunt de obicei antrenate utilizând SGD pentru a găsi ponderi ce minimizează anumite funcții de pierdere și pentru a mapa intrările arbitrare la ieșirile vizate cât mai aproape posibil [1] .

Un CNN este format din numeroase straturi precum: straturile de convoluție, straturi de interpolare și straturi complet conectate. Acestea utilizează un algoritm de propagare inversă pentru a învăța automat și ierarhiile spațiale ale datelor.

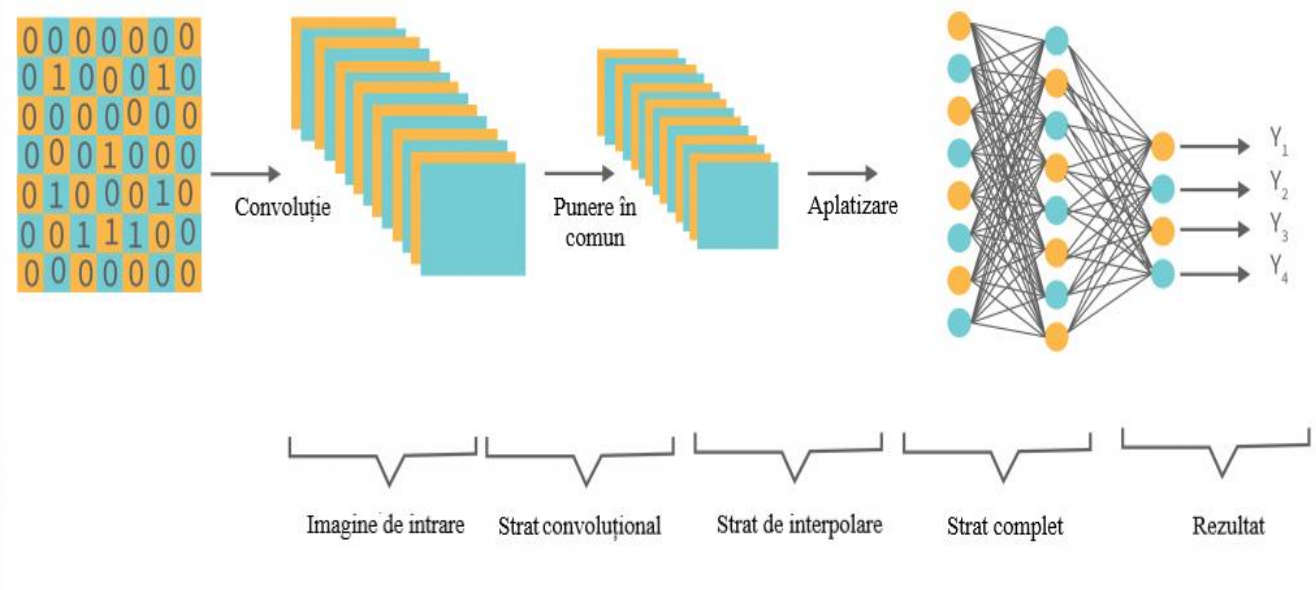


Figura 1.1: Arhitectura tipică a unui CNN; *Sursă: [1]*

Structura unui CNN prezintă o conectivitate similară cu cea a creierului uman în care neuronii sunt dispuși într-un mod specific. Aranjarea neuronilor într-un CNN este asemănătoare poziționării neuronilor în lobul frontal al creierului, zonă responsabilă cu procesarea informațiilor vizuale.
Datorită modului în care neuronii sunt dispuși, acest tip de rețea poate acoperi întregul câmp vizual al unei imagini, evitând astfel problema fragmentării pe care o întâmpină rețelele neuronale tradiționale.

În timp ce rețelele tradiționale trebuie să prelucreze bucăți ale imaginii cu rezoluție mai mică, CNN-urile pot procesa imaginea integral, ceea ce duce la o mai bună calitate a procesării datelor vizuale [2] .

Reamintim faptul că un CNN este format din numeroase straturi, însă prezintă numai un număr de trei straturi principale:

- Straturi convoluționale
- Straturi de interpolare
- Straturi conectate complet

Primul strat din arhitectură este cel convoluțional, care poate fi urmat de alt strat convoluțional sau de un strat de interpolare; ultimul strat fiind reprezentat de stratul conectat complet. Fiecare strat adaugă o complexitate mai mare modelelor și ajută la identificarea porțiunilor mai mari din imagine. Straturile inițiale se concentrează pe caracteristici simple, precum culori, linii, unghiuri, în timp ce straturile ulterioare încep să recunoască elemente mai mari sau chiar forme ale obiectelor. Prin procesarea imaginii, prin intermediul a mai multor straturi, arhitectura poate recunoaște și identifica obiectele din imagine, până când ajunge la rezultatul dorit.

1.3 Straturi convoluționale

Stratul convoluțional reprezintă componenta de bază a rețelelor neuronale convoluționale, rețele ce se ocupă de efectuarea operațiilor de convoluție. Pentru realizarea acestora, stratul convoluțional folosește o matrice numită kernel sau filtru, care efectuează o serie de ajustări orizontale și verticale, astfel încât să acopere întreaga imagine. Dimensiunea kernel-ului este mai mică decât dimensiunea imaginii, dar are o adâncime mai mare, care se extinde pe toate canalele de culoare ale imaginii. Acest lucru ajută la extragerea caracteristicilor importante ale imaginii, cum ar fi marginile sau textura [2] .

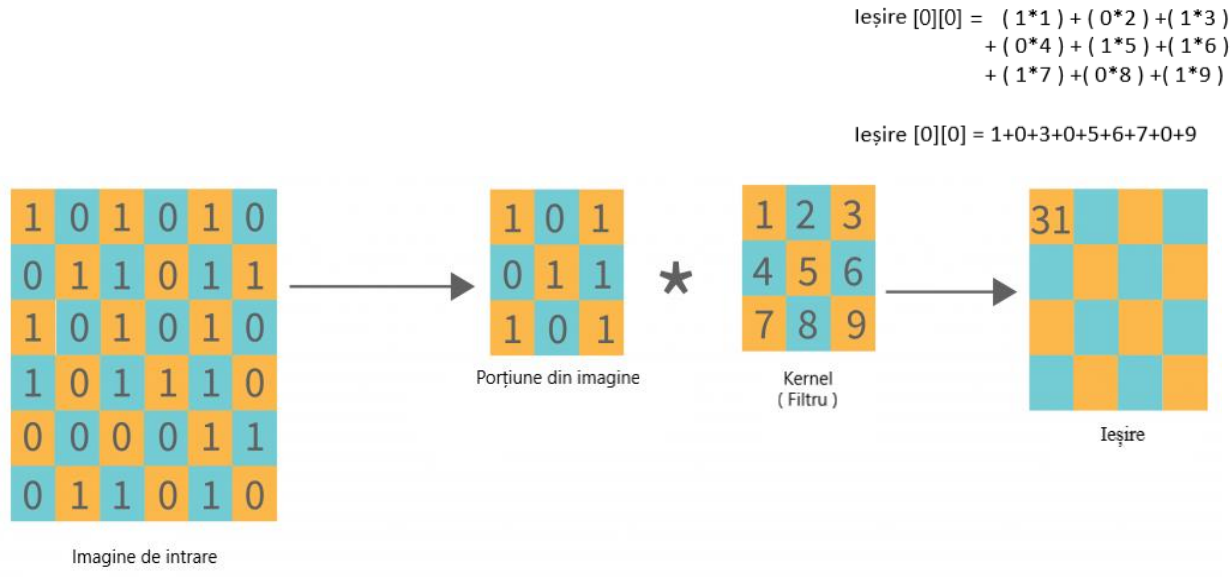


Figura 1.2 : Modul de funcționare al stratului convoluțional; *Sursă: [1]*

După cum se poate observa în figura 1.2, filtrul este aplicat peste o zonă a imaginii unde se va calcula un produs scalar între zona respectivă și filtrul, obținând un rezultat ce va deveni un element dintr-o nouă matrice. Acest proces se repetă până când filtrul se suprapune peste toate zonele din imaginea oferită la intrare, la final rezultând o nouă matrice ce poartă numele de hartă a caracteristicilor sau hartă de activare [3] .

Straturile convoluționale pot fi suprapuse pentru a obține o ierarhie, astfel straturile viitoare pot vedea pixelii straturilor anterioare. Ca și exemplu, pentru lucrarea de față, în cazul analizei de expresii faciale, o primă etapă o reprezintă identificarea feței. Întrucât chipul este reprezentat din mai multe elemente: ochi, nas, gură, urechi etc., fiecare parte separată va forma un strat de nivel inferior, iar combinarea părților reprezintă un nivel superior, creând astfel ierarhia de caracteristici.

În straturile convoluționale, pe lângă operația de convoluție, există și o altă componentă importantă numită funcția de activare neliniară. Aceasta este aplicată asupra ieșirilor operațiilor liniare, după fiecare convoluție. În prezent, funcția de activare neliniară, reprezintă cea mai frecvent utilizată unitate liniară rectificată (ReLU) și are funcția matematică $f(x) = \max(0, x)$ [2] .

1.4 Straturi de interpolare

Stratul de interpolare, cunoscut ca și strat de subeșantionare, este responsabil cu reducerea dimensiunii numărului de parametrii aflați la intrare, reducând astfel cantitatea de putere de calcul necesară procesării datelor. Acesta acționează similar straturilor convoluționale, parcurgând întreaga matrice de intrare, însă filtrul va oferi o altă funcție, numită funcție de agregare.

Cu toate că acest proces, numit și proces de pooling, duce la pierderea unui număr semnificativ de informații, acesta aduce și numeroase beneficii în cazul CNN-urilor. Printre ele se numără reducerea complexității, îmbunătățirea eficienței și limitarea riscului de supraadaptare. Astfel, rețeaua devine mai ușor de antrenat și rulat pe dispozitive cu resurse limitate[1] .

În cadrul straturilor de interpolare, matricea de la ieșire se va popula prin două tipuri de interpolare:

- Interpolare medie
- Interpolare maximă

Cele două metode sunt diferite din punct de vedere al prelucrării imaginilor în rețelele neuronale. În timp ce interpolarea maximă selectează cel mai mare pixel din câmpul receptiv și îl încarcă în matricea de ieșire, interpolarea medie calculează media valorilor din câmpul receptiv și trimite valoarea obținută în matricea de ieșire [1] .

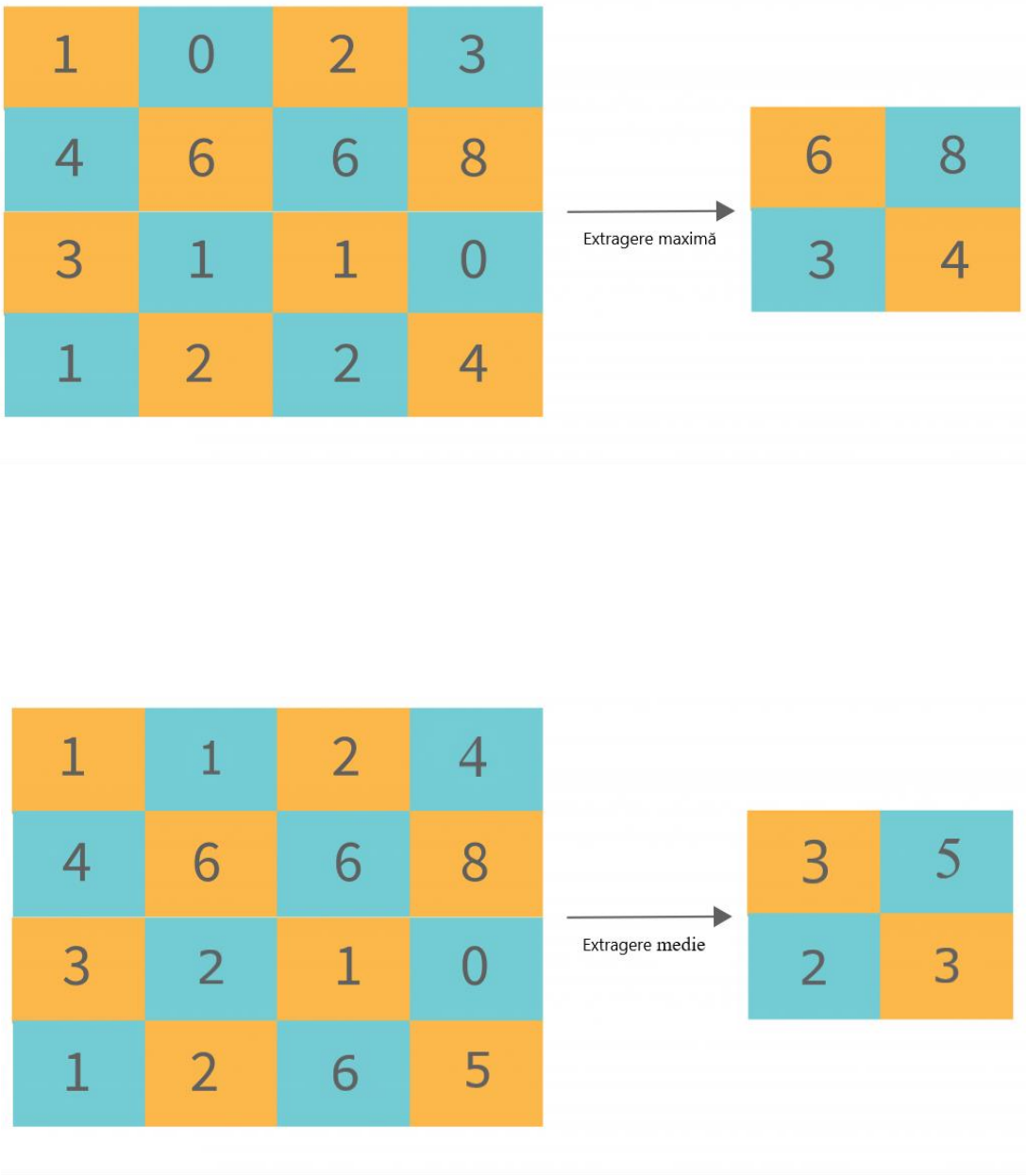


Figura 1.3 : Modul de funcționare al stratului de interpolare prin extragerea maximă, respectiv medie; *Sursă: [1]*

1.5 Straturi conectate complet

Stratul conectat complet poate reprezenta ultimul strat din arhitectura unui CNN, fiind plasat înaintea ieșirii de clasificare. Stratul conectat complet (FC) funcționează prin conectarea fiecărui nod din stratul de ieșire la un nod din stratul anterior și este folosit pentru a aplatiza rezultatele înainte de clasificare. Pot fi adăugate mai multe straturi FC prin care vectorul aplatizat poate fi transmis, permițând operațiilor funcționale matematice să fie efectuate în mod normal [3] .

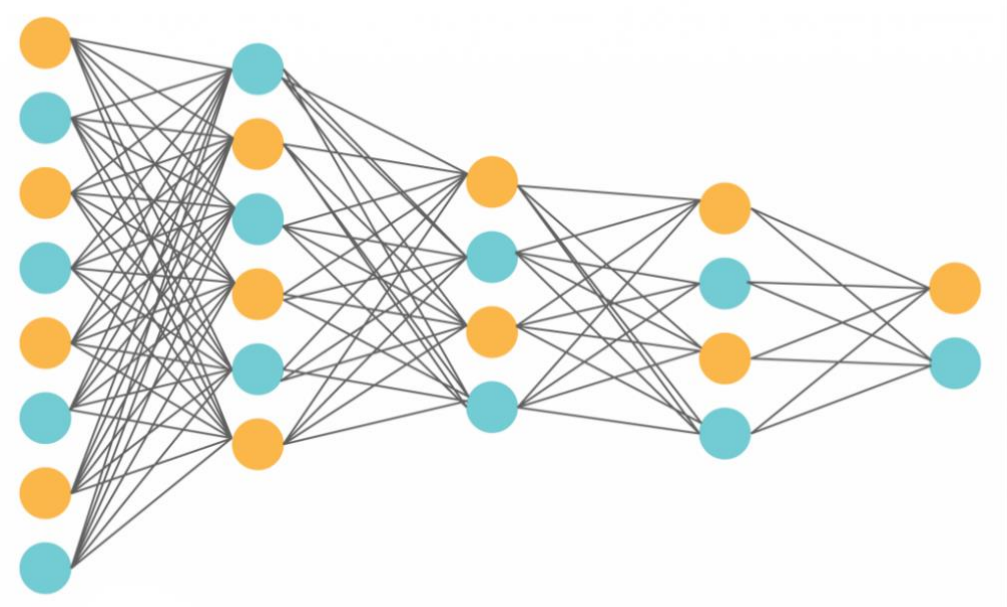


Figura 1.4 : Modul de funcționare al stratului conectat complet; *Sursă: [1]*

Comparativ cu celelalte straturi prezentate, stratul FC folosește o funcție de activare softmax ce normalizează valorile reale de ieșire de la ultimul strat complet conectat la probabilitățile clasei țintă. Astfel, pentru a clasifica intrările în mod corespunzător, valorile variază de la 0 la 1 [1] .

Funcția softmax se calculează folosind formula:

$$\sigma(\vec{z})i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad , i \in (0,\infty) \quad , \quad z = (z_1, z_2, z_3 \ldots . z_k) \in \mathbb{R}^K \qquad (1.5.1)$$

Unde:

- $\vec{z}$ – vectorul de intrare
- e^{z_i} – funcție exponențială standard pentru o valoare din vectorul de la intrare
- **K** – numărul de clase [4]

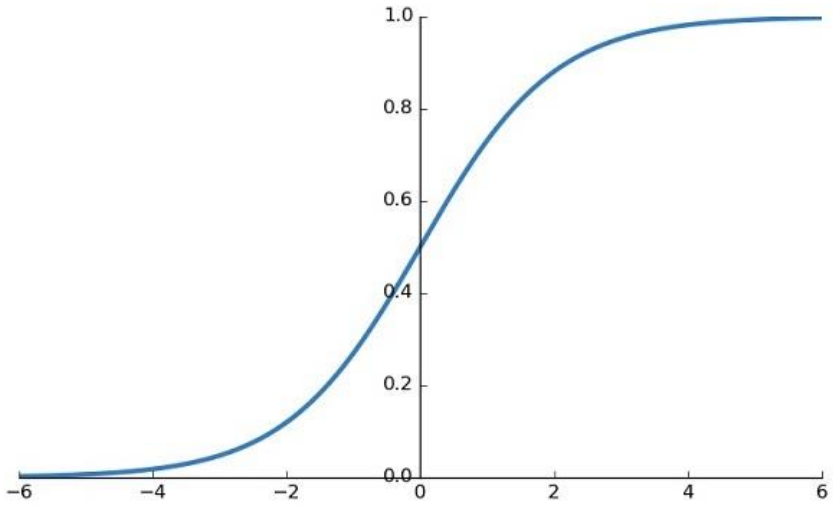


Figura 1.5 : Grafic funcție de activare Softmax; *Sursă [4]*

Cele trei straturi principale din rețeaua neuronală, ilustrate mai sus, prezintă un set unic de parametrii ce pot fi ajustați, îndeplinind o sarcină specifică asupra datelor de intrare.

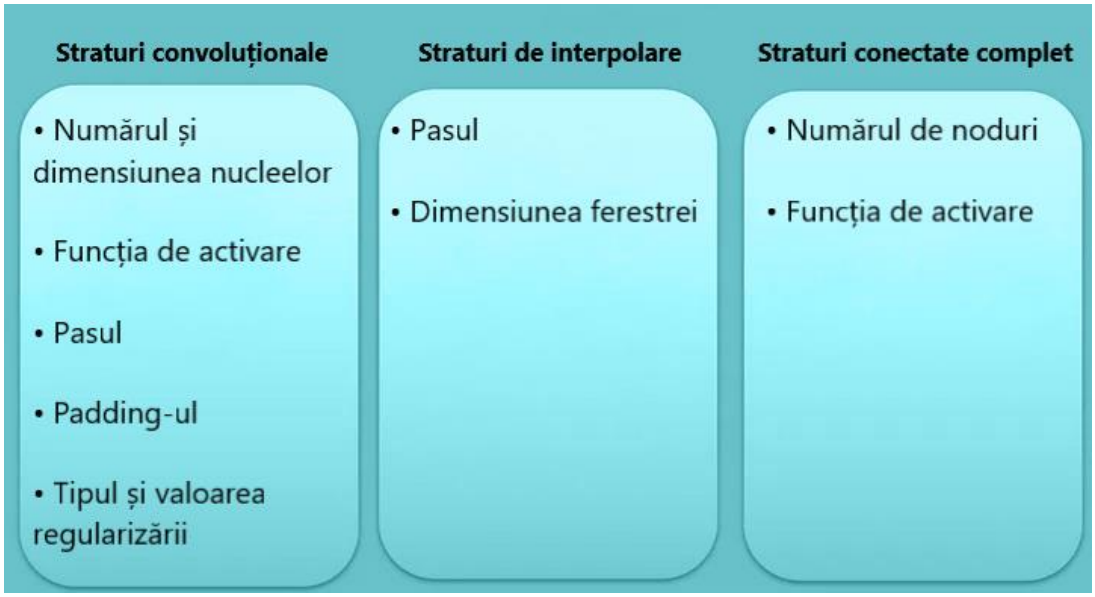


Figura 1.6 : Comparație parametrii straturi ; *Sursă [5]*

1.6 Stratul de normalizare

Stratul de normalizare reprezintă un strat special ce se inserează între două straturi ascunse și are rolul de a prelua ieșirile primului strat ascuns pentru a le normaliza și a le trimite mai departe drept intrări pentru următorul strat. Acesta poate fi plasat atât înainte, cât și după aplicarea funcției de activare, în diferite locații ale rețelei neuronale, în funcție de arhitectura specifică. Strategia de normalizare constă în calculul mediei și a deviației standard a valorilor de intrare din fiecare lot (batch) și normalizarea acestora în funcție de aceste valori.

Drept exemplu, presupunem că datele de intrare sunt reprezentate de mai multe caracteristici $\overline{x_1, x_n}$, fiecare dintre acestea putând lua valori într-un interval diferit. Astfel, pentru fiecare caracteristică, luăm valorile dispuse pe coloană ale acestora din setul de date și calculăm media și varianța, urmând mai apoi să normalizăm valorile folosind formula : $x_i = \frac{x_i - x_{Mean_i}}{StdDev_i}$ [6] .

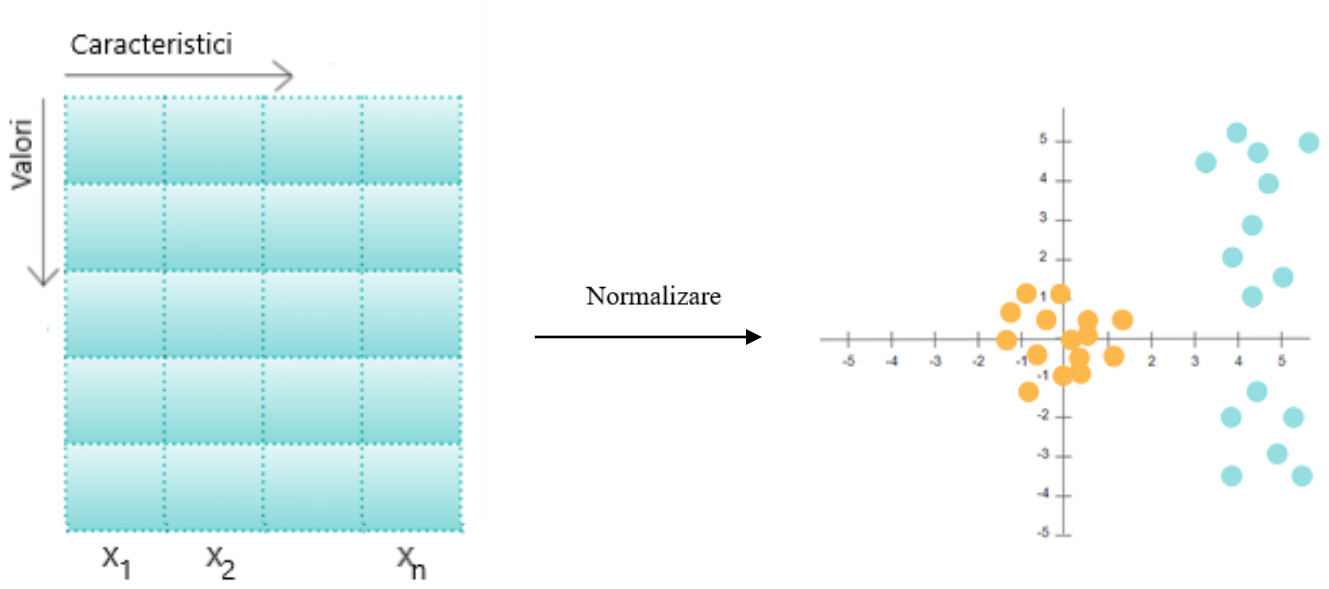


Figura 1.7 : Normalizarea datelor de intrare și efectul acestora ; *Sursa [6]*

Efectul normalizării datelor se poate observa în imaginea de mai jos, unde valorile originale, colorate în albastru, vor fi centrate în jurul valorii de 0, în culoarea portocaliu.
Precum și în cazul straturilor prezentate anterior (convoluționale, de interpolare și cele conectate complet), stratul de normalizare prezintă și el mai mulți parametrii :

- **affine** – valoare booleană în care parametrii affine pot fi învățați dacă aceasta este activată;
- **eps** – valoare necesară pentru stabilitatea numerică;
- **num_features** – numărul de canale ale stratului anterior;
- **track_running_stats** – valoare booleană ce urmărește media și varianța.

1.7 Stratul de abandon

Stratul de abandon reprezintă și el unul din straturile importante și necesare lucrării de față. Se cunoaște faptul că rețelele neuronale de diferite arhitecturi se confruntă cu problema supraajustării, cunoscută sub termenul de "overfitting", în care datorită încercării progresive de a învăța diferite caracteristici, rețelele neuronale învață, uneori, și zgomotul static din setul de date, fapt care ne dezavantajează.
Există mai multe metode pentru evitarea problemei, însă cea mai bună soluție rămâne cea a straturilor de abandon (eng: dropout) [7] .

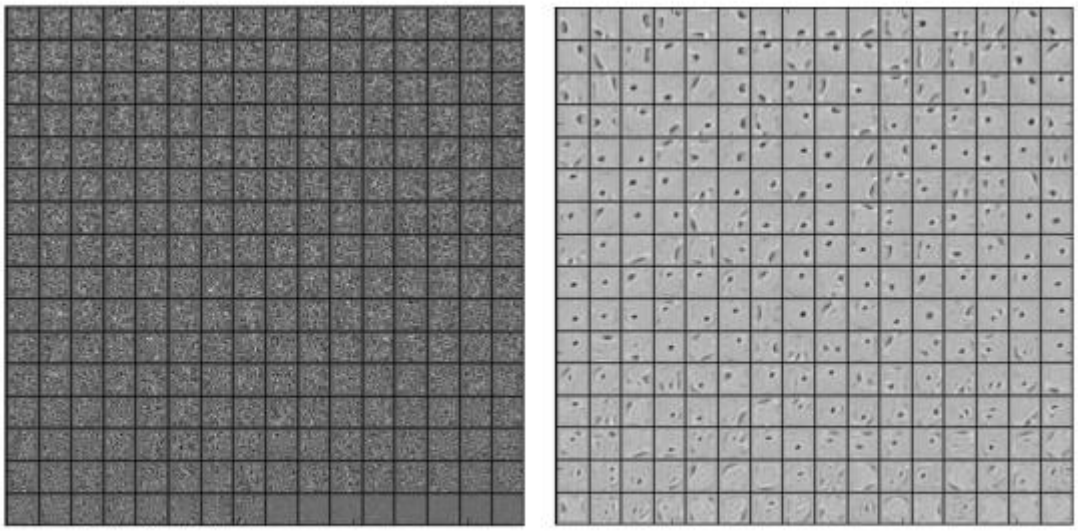
Procesul de Dropout are loc atât în etapa de antrenare, cât și în etapa de testare. Astfel, în timpul antrenării, neuronii sunt dezactivați cu o probabilitate fixată, în mod aleator, pentru fiecare mini-batch de date, rezultând un nou set diferit de neuroni activi pentru fiecare actualizare a ponderilor.Acestea conduc la o medie a ponderilor ponderată. În timpul testării, toate unitățile sunt activate, dar valorile ponderilor sunt ajustate pentru a obține ponderile mediate obținute în timpul antrenării [13] .

Astfel, conform celor menționate mai sus, straturile de abandon renunță la mai multe noduri, fapt ce duce, de asemenea, și la abandonarea temporară a conexiunilor pe care nodurile respective le dețineau, rezultând astfel o nouă arhitectură de rețea din cea inițială. Deci, prin eliminarea aleatorie a câtorva noduri, chiar și cu o probabilitate mică, straturile sunt forțate să-și asume mai multă sau mai puțină responsabilitate pentru intrare, luând o abordare probabilistică.
Deci, unul dintre aspectele importante ale straturilor de abandon constă în alegerea unei valori a probabilității de abandon. Această probabilitate reprezintă șansa ca un neuron să fie dezactivat în timpul etapei de antrenare. Alegerea valorii probabilității poate reprezenta o etapă esențială în procesul de optimizare întrucât alegerea unei probabilități mici conduce la supraadaptarea modelului, în timp ce o probabilitate prea mare poate influența negativ performanța modelului [13] .

De exemplu, în cazul în care straturile ascunse prezintă 1000 de neuroni, deci 1000 de noduri și se dorește renunțarea la acestea cu o probabilitate de 0.5, atunci 500 de neuroni vor fi abandonați aleatoriu la fiecare iterație [7] .

Așadar, datorită lucrării lui Srivastava, Hinton, Krizhevsky, Sutskever și Salakhutdinov din 2014, în care straturile de abandon au fost introduse pentru prima dată într-o referință, tehnica de dropout a devenit una dintre cele mai importante și necesare în dezvoltarea rețelelor neuronale.
Prin aplicarea cu succes a acestei tehnici, datorită faptului că modelul nu se mai poate baza excesiv pe niciun mănunchi de neurni, modelul va deveni mai robust, iar informația va fi distribuită uniform în rețea. [13]

De menționat faptul că în cazul straturilor ascunse, valoare de 0.5 este cea mai optimizată probabilitate.



(a) Without dropout (b) Dropout with $p = 0.5$.

Figura 1.8 : Caracteristica straturilor ascunse fară și cu un abandon de probabilitate 0.5; *Sursa [7]*

După cum se poate observa în imaginea de mai sus, stratul ascuns cu abandon (în care se renunță la 50% din noduri) învață mai multe caracteristici generalizate decât stratul ascuns fără abandon și se evidențiază, de asemenea că straturile de dropout se concentrează mai mult pe generalizare.

1.8 Rețele Siameze

O rețea neuronală siameză (SNN) reprezintă o clasă de arhitecturi de rețele neuronale care poate cuprinde două sau mai multe sub-ramuri ce prezintă configurații cu parametrii identici. Aceste rețele sunt folosite pentru a decide dacă două imagini prezintă caracteristici diferite sau nu și sunt utilizate cel mai mult în sisteme de verificare precum autentificarea biometrică. Aceasta se poate face atât prin intermediul caracteristicilor fizice (recunoașterea facială și a amprentelor digitale), cât și prin intermediul celor comportamentale (semnătura, modul de scriere). Scopul principal al rețelelor siameze este de a învăța o reprezentare a datelor care să identifice similarități și diferențe între două instanțe de date [8] .

Spre exemplu, pentru crearea unei aplicații pentru recunoașterea expresiilor faciale similare utilizând rețele siameze se va proceda în felul următor:

1. Se va colecta un set de imagini ce conține expresii faciale identice și se vor eticheta corespunzător emoției reprezentate în imagine: bucurie, tristețe, furie, etc;
2. Pozele vor fi trecute printr-un proces de prelucrare ce poate cuprinde scalarea, decuparea, normalizarea, ș.a.m.d., pentru a asigura o învățare eficientă;
3. Se va defini arhitectura rețelei siameze;

4. Se va împărți setul de date în perechi de imagini ale aceleiași persoane ce exprimă aceeași emoție;
5. Perechile de imagini vor fi introduse în rețea pentru obținerea de vectori caracteristici fiecărei imagini, măsurându-se mai apoi, printr-o funcție euclidiană (sau oricare alta ce are la bază comparația), măsura de similitudine;
6. Într-un final, cu ajutorul unui alt set de date ce conține imagini noi cu expresii faciale, se va testa și evalua rețeaua. Performanța fiind evaluată în funcție de acuratețea obținută în identificarea expresiilor similare.

Figura 1.9 : Compararea a două imagini pentru stabilirea unui scor de similitudine; *Sursa [8]*

Un alt beneficiu important al rețelelor siameze este capacitatea de a învăța caracteristici comune din datele de intrare. Aceasta înseamnă că rețeaua este capabilă să identifice și să extragă aspecte semnificative pentru ambele date de intrare. Prin această învățare de caracteristici comune, rețelele siameze sunt capabile să generalizeze mai bine și să obțină o performanță mai bună pe date noi, chiar dacă acestea se află într-un număr mai mic.

Cu toate acestea, rețelele siameze nu sunt o soluție universală și pot exista provocări în antrenarea și optimizarea lor. Un aspect critic îl reprezintă alegerea unei funcții de pierdere adecvate, care să evalueze similaritatea între cele două imagini. Selecția greșită a funcției de pierdere poate duce la rezultate nu tocmai fericite sau chiar la dificultăți în învățarea modelelor. De asemenea, gestionarea setului de date și echilibrarea acestuia reprezintă un rol esențial în obținerea performanței optime a rețelei siameze. În plus, optimizarea parametrilor rețelei poate necesită o atenție specială și o abordare diferită în funcție de problema întâmpinată.

2. Implementare

2.1 Biblioteci

Bibliotecile necesare pentru dezvoltarea aplicației sunt următoarele: Pytorch, Torchvision, NumPy.

- Pytorch este o bibliotecă open-source de învățare automată fiind lansată prima dată în 2016 și de atunci devenind una dintre cele mai utilizate biblioteci pentru antrenarea rețelelor neuronale profunde. Acesta oferă unelte flexibile pentru a defini și antrena modele, permițând dezvoltatorilor să își construiască rapid prototipuri de modele, să le testeze și să le optimizeze performanța. Drept exemplu, datorită caracteristicilor acestuia, Pytorch este folosit de marii lideri tehnicieni în industrie precum Tesla și Uber [11] .

- Torchvision este o bibliotecă ce lucrează foarte bine cu biblioteca Pytorch și ajută în antrenarea și evaluarea rețelelor neuronale utilizate în procesarea imaginilor. Aceasta conține modele pre-antrenate pentru a accelera procesul de dezvoltare a aplicațiilor de învățare automată bazate pe imagini și oferă, de asemenea, funcții de transformare, cum ar fi redimensionarea, rotirea, decuparea și normalizarea imaginilor, pentru a ajuta la pregătirea datelor de antrenare [11]

- NumPy reprezintă o bibliotecă Python populară pentru calculul numeric și științific. Aceasta oferă funcții eficiente pentru manipularea de matrice și vectori multidimensionali, care sunt esențiale în construirea și antrenarea de rețele neuronale profunde[11] .

Pe lângă acestea au mai fost folosite și alte biblioteci Python standard precum: os, time, copy, json, random, cv2 și matplotlib utilizate pentru diferite scopuri (sistem de operare, timp, copiere, serializare, generare de numere aleatoare, prelucrarea imaginilor și afișarea datelor).

2.2 Configurare software

Limbajul de programare folosit pentru abordarea temei este Python și poate fi instalat accesând site-ul https://www.python.org/downloads/. De asemenea, mai este necesară instalarea unei aplicații IDE, pentru a ușura gestionarea și pentru a facilita procesul de dezvoltare. IDE-ul folosit în lucrarea de față este PyCharm și poate fi descărcat de pe site-ul https://www.jetbrains.com/pycharm/ .
După descărcarea programelor, este necesară instalarea bibliotecilor, în primă instanță fiind esențială biblioteca Pytorch, restul putând fi instalate pe parcurs. Librăria Pytorch se poate procura de pe site-ul https://pytorch.org/ sau rulând comanda: *pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu113* în terminalul IDE-ului.

2.3 Baza de date

Primul pas pentru realizarea proiectului a fost parcurgerea bazei de date puse la dispoziție de cei de la Google care poate fi descarcată de aici : https://research.google/resources/datasets/google-facial-expression/ . Acest set de date constă în triplete de imagini a feței umane împreună cu adnotări care specifică dacă acestea sunt asemănătoare în ceea ce privește expresia facială. Toate informațiile din baza de date se află într-un fișier CSV, iar fiecărei linii îi corespunde un eșantion de date. Întrucât lucrarea de față prezintă o abordare a numai două imagini, deci dublete de expresii faciale, a fost necesară modificarea acestei baze de date pentru a se mula pe cerințele noastre.

Fiecare eșantion din baza de date inițială constă în triplete de imagini ale feței împreună cu etichetele aferente acestora ce indică care două imagini din triplet formează cea mai asemănătoare pereche de expresii faciale. De exemplu, o etichetă având valoarea 1 ilustrează faptul că a doua și a treia imagine sunt vizual mai asemănătoare în ceea ce privește expresia facială, în comparație cu prima imagine, o valoare de 2 reprezintă faptul că prima și a treia imagine sunt vizual mai asemănătoare, iar o valoare de 3 conduce la faptul că prima imagine este vizual mai asemănătoare cu a doua imagine. De menționat este faptul că această adnotare tripletă utilizată nu este una obișnuită.

În general, un triplu loss prezintă noțiunea de poză ancoră, iar celelalte două poze fiind reprezentate de reprezentări ale cazurilor pozitive și reprezentări ale celor negative. În timp ce, în lucrare, fiecare triplet oferă două adnotări: a doua imagine este mai asemănătoare cu imaginea a treia, decât cu prima imagine, iar imaginea a treia este mai asemănătoare cu a doua imagine decât cu prima imagine. De asemenea, tot în baza de date inițială, o a doua imagine poate fi mai asemănătoare de prima imagine într-un set de date, dar totuși mai asemănătoare de a treia imagine într-un alt set de date.

Deci, pentru o mulare corespunzătoare a datelor pe cerințele lucrării noastre a fost necesară aducerea unor modificări a bazei de date. Așadar, s-a iterat prin toate fișierele generate cu cele trei imagini și s-a șters a treia imagine, astfel dacă clasa veche era 3 acum va fi 1, mai exact cele două poze rămase se aseamănă, iar daca clasa veche era 1 sau 2, noua clasă va deveni 0, deoarece pozele sunt diferite.

Așadar, noile date au fost structurate în noi fișiere CSV atât pentru antrenare, cât și pentru validarea rezultatelor obținute. Modificările au fost aduse numai în privința numărului de poze și al clasei din care acestea fac parte (1 dacă cele două poze sunt vizual asemănătoare, iar 0 dacă cele două poze nu sunt asemănătoare), numărul de evaluatori și de adnotări rămânând cel inițial.

Concluzionând, pe baza celor oferite de baza de date inițială și a modificărilor aduse pentru realizarea unei aplicații care să primească drept date de intrare doar două imagini și să stabilească dacă acestea sunt asemănătoare sau nu, noul CSV prezintă următoarele coloane:

- Adresa URL a imaginii 1
- Coloana din stânga sus a casetei de delimitare a feței din imaginea 1 normalizată prin lățime
- Coloana din dreapta jos a casetei de delimitare a feței din imaginea 1 normalizată prin lățime
- Rândul din stânga sus a casetei de delimitare a feței din imaginea 1 normalizată prin înălțime
- Rândul din dreapta jos a casetei de delimitare a feței din imaginea 1 normalizată prin înălțime
- Adresa URL a imaginii 2
- Coloana din stânga sus a casetei de delimitare a feței din imaginea 2 normalizată prin lățime
- Coloana din dreapta jos a casetei de delimitare a feței din imaginea 2 normalizată prin lățime
- Rândul din stânga sus a casetei de delimitare a feței din imaginea 2 normalizată prin înălțime
- Rândul din dreapta jos a casetei de delimitare a feței din imaginea 2 normalizată prin înălțime
- Clasă dublet
- ID evaluator 1
- Adnotare 1
- ID evaluator 2
- Adnotare 2
- ID evaluator 3
- Adnotare 3
- ID evaluator 4
- Adnotare 4
- ID evaluator 5
- Adnotare 5
- ID evaluator 6
- Adnotare 6

Toate imaginile prezintă una sau mai multe etichete emoționale: amuzament, furie, plictiseală, concentrare, confuzie, jenă, mulțumire, durere, mândrie, tristețe, triumf, simpatie, etc. Acestea au fost oferite în perechi de imagini evaluatorilor care au fost instruiți să se concentreze numai pe expresii faciale ignorând alți factori precum identitatea, genul, etnia, poziția, vârsta, etc. și să ofere o adnotare reprezentată de un număr întreg din mulțimea {0, 1}. O valoare de 1 înseamnă că expresiile de pe cele două fețe sunt vizual asemănătoare între ele, în timp ce o valoare de 0 ilustrează contrariul.
Întrucât baza de date este una destul de mare, oferind perechi de 500 000 poze, pentru antrenarea rețelei am ales numai 11 000 de dublete de poze împărțite în 95% antrenare și 5% validare.

2.4 Prelucrarea datelor

Următorul pas a fost cel de prelucrare al imaginilor. Prin urmare, am realizat un CSV în care am introdus link-urile celor 11.000 perechi de poze (fiecare rând fiind reprezentat de câte un dublet) și care servește în descărcarea și numirea pozelor și un alt CSV în care am introdus coordonatele acestora în număr de 4, necesare pentru decuparea porțiunii reprezentative feței.
Pentru a grăbi procesul de descărcare al pozelor m-am folosit de tehnica multithreading ce a condus și la o îmbunătățire a performanței procesului de prelucrare a imaginilor.

Multithreading reprezintă o tehnică de optimizare în prelucrarea imaginilor prin care rularea a mai multor fire de execuție (thread-uri) este făcută în paralel. Fiecare thread poate prelucra o imagine sau chiar un set de imagini într-o manieră independentă, ceea ce conduce la o prelucrare mai rapidă și mai eficientă a datelor. Această abordare este deosebit de utilă în cazul de față, întrucât baza de date aferentă prezintă un număr mare de imagini și situații care, în mod obișnuit, ar necesita un timp de prelucrare semnificativ mai ridicat[12] .
Astfel, sistemul va avea capacitatea de a desfășura mai multe sarcini într-un mod simultan, dar totuși independent. Acest fenomen poartă denumirea de paralelism. Gradul de paralelism constă în posibilitatea de a procesa date în paralel, fără a depinde de ordinea și secvența acestora.
În contextul de față, cel al prelucrării imaginilor, gradul de paralelism constă în capacitatea de a prelucra mai multe imagini în același timp, prin folosirea mai multor resurse de calcul.Astfel, prin distribuirea sarcinilor de prelucrare pe mai multe fire de execuție, se poate realiza un grad ridicat de paralelism, ceea ce conduce la reducerea timpului total de prelucrare, obținând, de asemenea, și o eficiență crescută[12] .

Astfel, după ce au fost descărcate pozele, le vom aplica o serie de transformări. Ca și în cazul descărcării pozelor, pentru prelucrarea acestora a fost folosită o tehnică ce a constat în executarea mai multor sarcini în același timp, aceasta fiind cunoscută drept multiprocesing. Multiprocesing reprezintă, după cum sugerează și numele, tehnica prin care se utilizează mai multe procesoare în execuția diferitelor sarcini concomitent. În contextul prelucrării de imagini, pentru susținerea temei noastre, multiprocesarea a fost folosită pentru accelerarea antrenării și distribuirea sarcinilor de prelucrare pe mai multe procesoare, conducând astfel la o paralelizare eficientă a calculelor. Așadar, imaginile au fost decupate (folosind funcția RandomResizedCrop), răsucite (folosind funcția RandomHorizontalFlip), transformate în tensori, urmând mai apoi ca valorile pixelilor să fie normalizate. De menționat faptul că acestea au fost dimensionate la 224 x 224 x 3, datorită mărimii impuse de stratul de intrare al arhitecturii.

Mai exact, funcția RandomResizedCrop reprezintă una din transformările comune de imagine utilizate în modelele de rețele neuronale. Aceasta funcționează prin selectarea regiunii aleatorii, sau dorite, precum în cazul de față (prin intermediul coordonatelor oferite din CSV) și decuparea acesteia la dimensiunea dorită. Funcția RandomHorizontalFlip reprezintă, de asemenea, și ea una din

transformările comune utilizate. În schimb, aceasta funcționează prin fliparea/rotirea aleatorie a imaginii. Cele două trasnformări ajută la prevenirea overfitting-ului și la creșterea diversității și a cantității datelor de antrenament, conducând la îmbunătățirea generalizării modelului la setul de date de testare. După prelucrare, imaginile sunt de obicei implementate sub formă de tensori cu 3 sau 4 dimensiuni, iar aceștia sunt normalizați astfel încât să se asigure faptul că valorile pixelilor din imagini sunt într-un interval specificat. Întrucât într-o imagine valorile pixelilor pot varia semnificativ, de la 0 la 255, în cazul imaginilor RGB, prin normalizarea imaginilor, acestea sunt aduse într-un interval standard. Acest procedeu ajutând la îmbunătățirea performanței rețelelor neuronale prin reducerea gradientului de valori și a variației acestora.

În concluzie, prin folosirea tehnicilor de multithreading și multiprocesing s-a dorit accelerarea și optimizarea procesului de prelucrare a imaginilor. Totuși, este important de menționat faptul că implementarea și optimizarea celor două tehnici depind de bibliotecile și framework-urile folosite. Astfel, este necesară trecerea în revistă a specificațiilor oferite de acestea în contextul specific al prelucrarii imaginilor în rețelele neuronale înaintea folosirii acestora.

2.5 Antrenarea rețelei

Întrucât procesul de antrenare durează mult timp datorită setului mare de date, am ales ca dispozitiv de lucru GPU-ul. Antrenarea putea fi realizată și pe CPU, dar acest proces poate fi lent, mai ales pentru seturi de date mari și rețele cu mulți parametri. GPU-urile sunt mai rapide și eficiente în executarea operațiilor matematice, în special datorită capacității lor de a realiza operațiile în paralel. De aceea, GPU-urile sunt utilizate pentru a accelera procesul de antrenare al rețelelor neuronale. Printre companiile importante specializate în dezvoltarea de soluții de calcul, precum și în producția de unități de procesare grafică (GPU-uri) se numără și NVIDIA ce a devenit unul dintre cei mai importanți furnizori pentru calculul de mare performanță și inteligența artificială. Pe lângă GPU-uri, NVIDIA dezvoltă și alte tehnologii, cum ar fi soluții software de optimizare și biblioteci pentru calcul paralel. Printre ele se numără și CUDA, care permite programatorilor să utilizeze puterea de calcul a GPU-urilor în mod eficient și cuDNN ce oferă funcționalități specializate pentru CNN-uri, cum ar fi operațiile de convoluție, normalizarea loturilor, pooling și funcții de activare. Aceasta (cuDNN) este una dintre bibliotecile cheie NVIDIA pentru dezvoltarea de aplicații de inteligență artificială și contribuie la accelerarea inovațiilor în domeniul rețelelor neuronale profunde și al învățării automate.

Astfel, după prelucrarea datelor, am oferit mai multe valori în ceea ce privește dimensiunea subsetului și a ratei de învățare, constatând astfel că cele mai bune rezultate le am folosind o putere a lui 2 pentru dimenisunea subsetului și o rată de învățare de 0.001 .

În ceea ce privește optimizatorii, aceștia reprezintă niște algoritmi utilizați în ajustarea parametrilor în procesul de antrenare al rețelei pentru obținerea unei funcții de pierdere mai mici și a unor rezultate și performanțe mai bune a modelului. Acești algoritmi sunt esențiali în etapa de învățare a programului.

Așadar, printre optimizatorii experimentați în lucrarea de față, se numără Adadelta, Adam, SGD.

• Adadelta se bazează pe memorarea gradientului anterior și ajustarea pasului de învățare pentru fiecare parametru. Acest lucru permite optimizatorului adaptarea la rate de învățare diferite, evitând probleme precum divergența algoritmului în cazul unei rate de învățare prea mari și a încetinirii convergenței în cazul folosirii unei rate de învățare prea mici. De asemenea, acesta este capabil de a se adapta la schimbările în distribuția gradientului pe măsură ce înaintează în procesul de optimizare [12] .

• ADAM sau "Adaptive Moment Estimation" utilizează o estimare a mediei și a variației, adaptându-se în mod eficient la rate de învățare diferite și la schimbările în distribuția gradientului pe măsură ce înaintează în procesul de optimizare. Optimizatorul permite algoritmului să facă față problemelor întâlnite în ceea ce privește valorile prea mici sau prea mari a ratelor de învățare și să atingă convergența (o valoare optimă a funcției de cost) mai rapid [12] .

• SGD sau "Stochastic Gradient Descent" reprezintă o variantă a algoritmului Gradient Descent, utilizat pentru ajustarea parametrilor unui model neuronal prin minimizarea funcției cost. Diferența dintre cele două constă în modul în care gradientul este calculat și de ajustările luate în cazul parametrilor. SGD calculează gradientul pe baza unei submulțimi aleatoare de date de antrenare, comparativ cu Gradient Descent ce utilizează toate datele de antrenare disponibile. Utilizarea unei submulțimi aleatoare de date de antrenare la fiecare iterație permite un calcul mai rapid și o actualizare mai deasă a parametrilor [12] .

Funcția de cost folosită a fost BCELoss -"Binary Cross-Entropy Loss " întrucât lucrarea de față susține o clasificare binară în care există doar două clase posibile: clasa pozitivă în care este ilustrat faptul că cele două imagini oferite transmit o emoție comună și clasa negativă în care cele două imagini nu înfățișează o emoție comună. Aceasta funcționează prin măsurarea discrepanței dintre probabilitatea prezisă și valoarea reală. Mai exact, rețeaua produce o probabilitate prezisă pentru clasa posibilă generată prin intermediul funcției de activare, iar cu ajutorul etichetei reale din setul de antrenare, se calculează valoarea entropiei încrucișate binare între cele două. Valorile generate pentru fiecare exemplu se adună, calculându-se apoi media ce este necesară pentru obținerea unei valori de pierdere agreată întregului set de date.

2.6 Rețele neuronale

Modelul de la care am pornit în lucrarea de față este ResNet18, un model preantrenat cu 18 straturi al CNN-urilor ce face parte din familia de arhitecturi ResNet. Acesta este unul dintre modelele populare și este, de obicei, preantrenat pe seturi de date mari precum ImageNet ce conține un număr mare de poze.

Se credea că un număr mare de straturi conduce la creșterea performanței rețelei, însă pe parcurs s-a demonstrat că nu întotdeauna se aplică acest lucru și că de cele mai multe ori creșterea numărului de straturi poate duce la reducerea acurateții modelului. Astfel, privind evitarea acestor probleme a apărut conceptul de " Skip connections/Residual connections " ce transformă rețelele neuronale în rețele de tip blocuri reziduale. Într-o astfel de conexiune, informația inițială poate sări peste unele straturi intermediare. Astfel, aceasta nu va avea un parcurs liniar, trecând prin toate straturile de convoluție, doar o parte din aceasta fiind păstrată și adăugată la rezultatul straturilor obținute anterior acesteia.
Prin aplicarea acestui concept, informația inițială nu este modificată și pierdută pe parcurs [9] .

În figura de mai jos se poate observa cum funcționează o astfel de conexiune în care se preiau informațiile inițiale stratului convoluțional (n-1) și sunt adăugate stratului convoluțional (n+1), stratul convoluțional n fiind astfel omis.

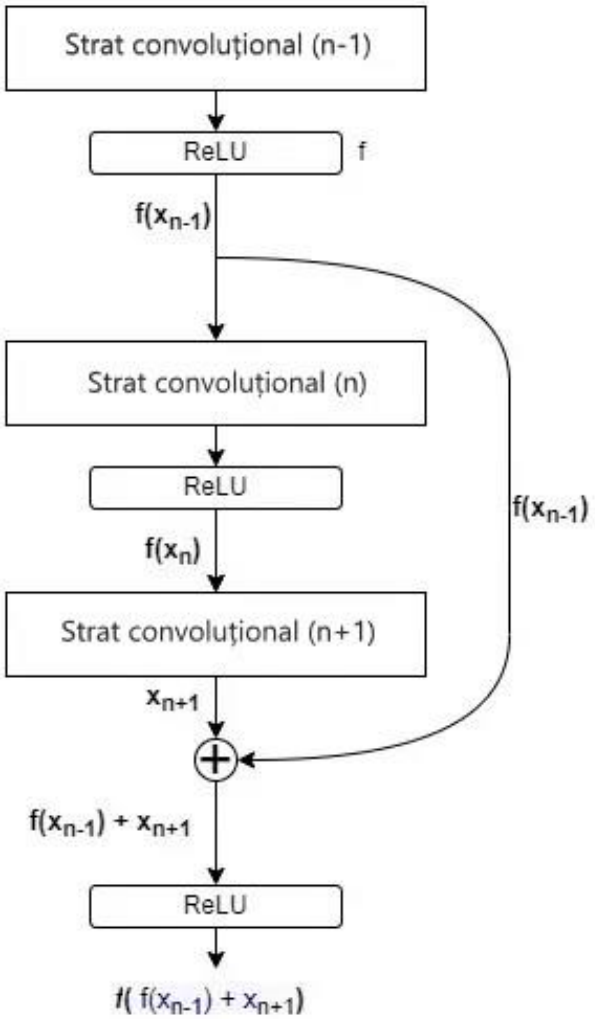


Figura 2.1: Diagrama modului de funcționare a conexiunilor Skip; *Sursă [9]*

O altă tehnică folosită în obținerea unor rezultate mai bune a fost cea a abandonului Această metodă constă în eliminarea temporară a unor noduri cu o probabilitate de abandon p, ducând astfel la crearea unei noi arhitecturi de rețea, evitând generalizarea setului de date și reducând șansa de supraadaptare.

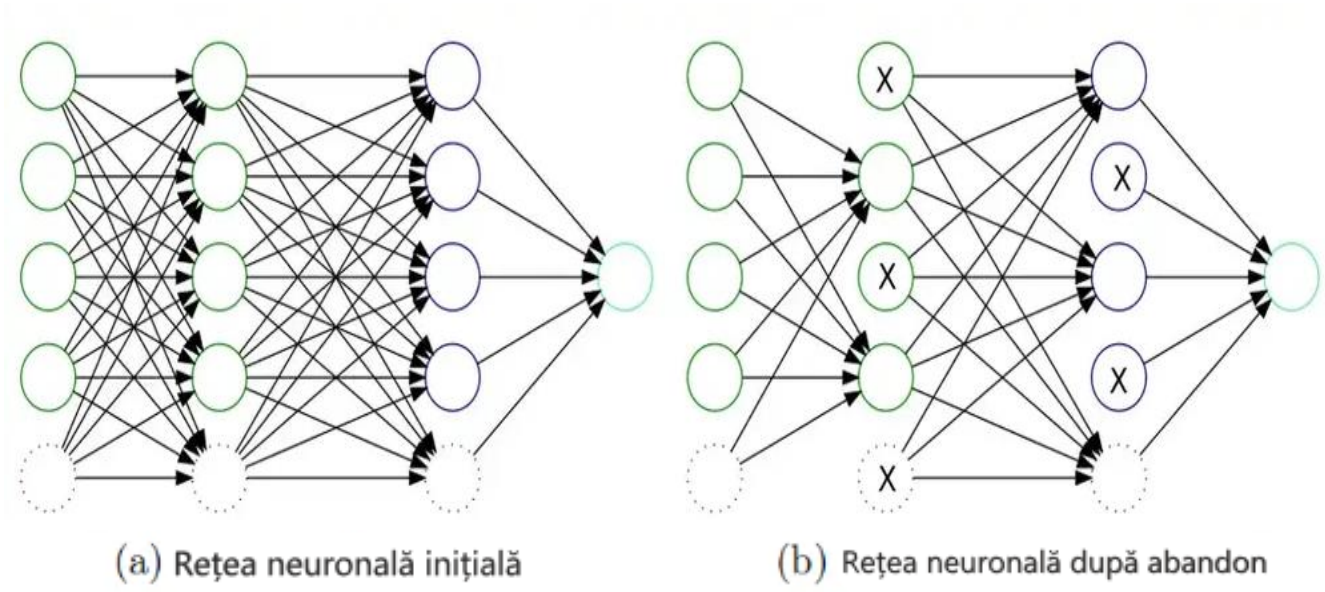


Figura 2.2: Renunțarea temporară a unor noduri prin tehnica abandonului ; *Sursa [7]*

Astfel, la fiecare pas de antrenare, neuronii sunt abandonați pe baza ratei/probabilității de abandon, forțând rețeaua neuronală să aibă o învățare mai robustă și mai generalizată, utilizând o varietate de trasee pentru generarea predicțiilor.

2.7 Probleme întâmpinate

În cadrul acestei lucrări au fost întâmpinate mai multe probleme și provocări. O prima problemă a fost reprezentată de înțelegerea setului de date și adaptarea acestuia conform cerințelor specifice. Întrucât baza de date inițială conținea triplete de imagini, fiecare prezentând etichetele corespunzătoare, în contextul lucrării de față, a fost necesară modificarea setului de date. Conform acestora, modificarea făcută a constatat în renunțarea la a treia imagine din fiecare triplet, deoarece se dorea analiza între două imagini. Pentru stabilirea noilor clase s-a ținut cont de apartenența tripletului inițial. Astfel, dacă un set de date inițial avea drept etichetă clasa 1, ceea ce reprezenta faptul că ultimele două poze din triplet, adică a doua și a treia imagine, sunt vizual mai asemănătoare, prin eliminarea ultimei imagini, s-a stabilit pentru perechea rămasă (prima și a doua imagine) valoarea de 0, întrucât, în mod inițial prima imagine nu se aseamănă cu a doua imagine. Aceeași situație se află și în cazul în care inițial era vorba

de o clasă de triplet având valoarea de 2. În cazul în care clasa de triplet inițială avea valoarea de 3, ceea ce reprezenta faptul că prima și a doua imagine sunt vizual mai asemănătoare decât a treia imagine, cea care într-un final a fost ștearsă, imaginilor păstrate din triplet li s-a atribuit valoarea de 1, întrucât, în mod inițial, în cazul clasei de triplet 3, prima imagine se asemăna cu cea de-a doua imagine.

O a doua problemă întâmpinată a fost legată de obținerea în totalitate a datelor, mai exact a imaginilor. Prin descărcarea acestora pe baza link-ului oferit de setul de date, am constatat că multe dintre acestea nu mai erau disponibile pentru descărcare, iar astfel, a trebuit să renunț la o bună parte de dublete de poze din cadrul CSV-ului.

În ceea ce privește antrenarea și testarea setului de date, o altă problemă a constatat în durata ca timp a finalizării acestora. Inițial, setul de date a fost antrenat prin intermediul IDE-ului PyCharm, însă, datorită solicitărilor prea mari, a fost necesară antrenarea rețelei folosind platforma GoogleColab. Problemele nu s-au încheiat însă nici aici, deoarece din dorința de a avea cât mai multe epoci de antrenare, unde o epocă necesită aproximativ 80 minute, iar GoogleColab permite un timp de rulare de maxim 24 de ore, a fost necesară achiziționarea pachetului Pro, care din păcate nu a avut rezultate excepționale comparativ cu cele inițiale.

3. Rezultate

3.1 Rezultate inițiale

În primă fază am ales un model preantrenat, mai exact mobilenet_v3_large, în care au fost înghețate toate ponderile, antrenarea fiind făcută pe ultimele două straturi fully connected. Astfel, pentru a determina clasa din care fac parte, toate pozele au fost trecute pe rând prin backbone. Rezultatele obținute fiind concatenate, urmând mai apoi antrenarea în care s-a determinat clasa din care fac parte: valoarea de 0 reprezentând emoții diferite, iar cea de 1 de emoții similare.

În acest caz, hiperparametrii folosiți au fost următorii:

- Dimensiune subset: 32
- Funcție de cost : entropia încrucișată
- Număr de epoci : 20
- Optimizator : Adam
- Rata de învățare : 0.001

Pe baza celor de mai sus, a fost obținută o acuratețe de 58 % .

Pentru a mă asigura că am obținut cele mai bune rezultate, am decis să urmez mai multe modele preantrenate. Printre acestea se numără și ResNet18 care a urmat, de asemenea, pașii menționați mai sus. Astfel, pentru acest caz am folosit următorii hiperparametrii:

- Dimensiune subset: 32
- Funcție de cost : BCELoss
- Număr de epoci : 20
- Optimizator : Adam
- Rata de învățare : 0.001

Rezultatul obținut fiind reprezentat de o acuratețe de 65%.

Conform celor menționate mai sus, cele două antrenări se diferențiază prin funcția de cost. Prima antrenare fiind făcută utilizând entropia încruțișată, iar cea de-a doua utilizând BCELoss, prezentând în plus o funcție de activare softmax.
După cum a fost susținut în lucrarea de față, funcția de cost BCELoss este folosită numai în probleme de clasificare binară, în care sunt posibile doar două clase, în timp ce Cross-Entropy Loss este folosită în probleme de clasificare multiplă, cu mai mult de două clase posibile. Utilizabilitatea celor două reprezentând unul dintre motivele pentru care acestea oferă rezultate diferite. Deci, întrucât funcția de

cost BCELoss își îndreaptă atenția asupra clasificării binare, acest lucru o face cea mai potrivită situației prezente.
Un alt motiv ce poate contribui la obținerea celor două rezultate diferite constă în calculul din spatele celor două funcții. Astfel, BCELoss aplică un logaritm binar pe valorile prezise, calculând mai apoi entropia încrucișată între valorile prezise și cele de referință, în timp ce Cross-Entropy Loss aplică un logaritm natural pe valorile prezise, calculând mai apoi media entropiei încurcișate pentru toate valorile prezise și cele oferite drept referință.

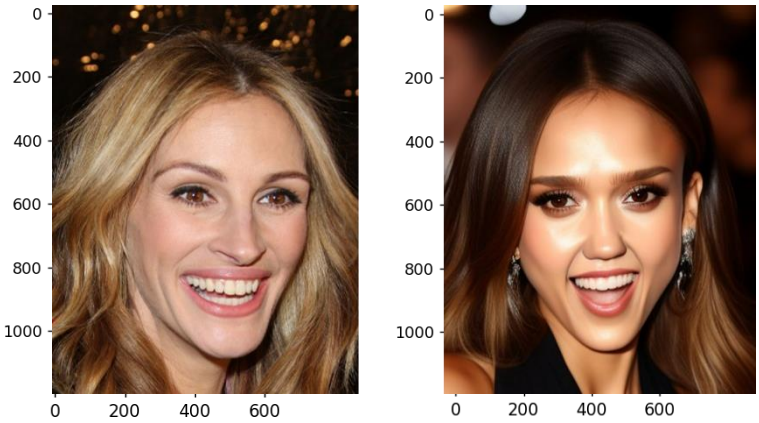
3.2 Rezultate după modificarea funcției de cost

Astfel, conform celor menționate în subcapitolul 3.1, întrucât am obținut un rezultat mult mai bun utilizând backbone-ul ResNet18 și funcția de cost BCELoss, am decis să fac mai multe antrenări modificând dimensiunea subsetului, optimizatorul și rata de învățate. Rezultatele obținute se regăsesc în tabelul întocmit mai jos:

Optimizator	Adadelta	Adadelta	Adadelta	Adam	Adam	Adam	SGD
Dimensiune subset	32	32	64	64	32	32	64
Rata de învățare	0.01	0.001	0.001	0.001	0.001	0.01	0.001
Număr de epoci	20	20	20	20	20	20	20
Acuratețe obținută	63%	63%	64%	65%	65%	36%	63%

Tabel 3.1 Rezultate obținute la antrenări diverse

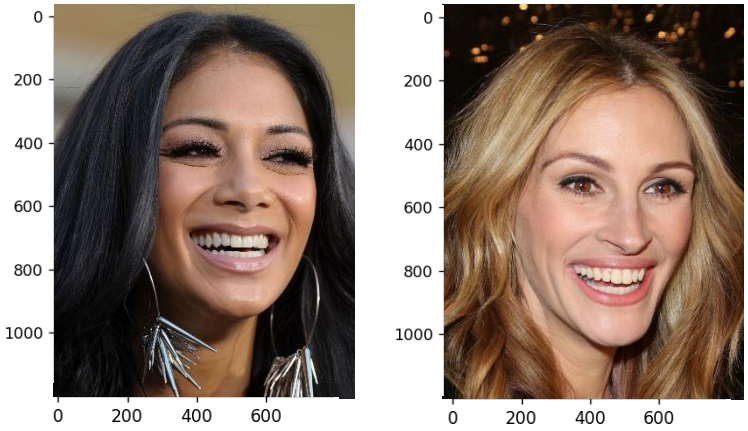
Întrucât acuratețea nu este destul de ridicată, modelul întâmpină câteva probleme în ceea ce privește stabilirea similitudinii pozelor. Spre exemplu, la compararea celor două poze de mai jos, programul menționează faptul că cele două celebrități prezintă aceași emoție, deși în poza din stânga este ilustrată o stare de amuzament, iar în cea din dreapta de surprindere. Acest lucru poate fii datorat faptului că, în multe dintre pozele destinate antrenării, persoanele ce afișează o stare de amuzament sunt adesea surprinse afișând un zâmbet larg, în care dinții sunt la vedere. Cazul de mai jos, ilustrat în figura 3.1, reprezentând unul dintre puținele exemple în care este returnată o predicție greșită, cele mai multe obținând cu succes o predicție favorabilă lucrării.



a – amuzament b – surprindere

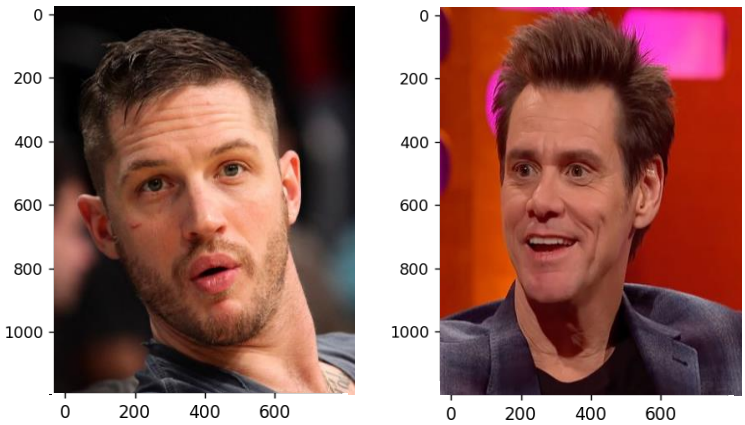
Aceeași emoție

Figura 3.1 : Exemplu predicție greșită



a – amuzament b – amuzament

Aceeași emoție



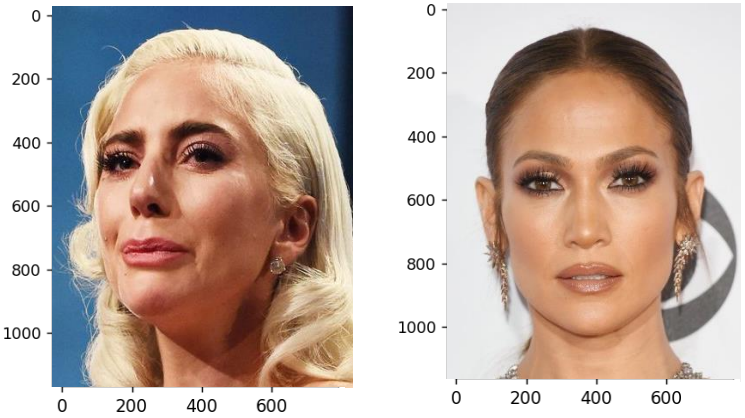
a – surprindere b – surprindere

Aceeași emoție

Figura 3.2: Exemplu predicție corectă între aceleași emoții

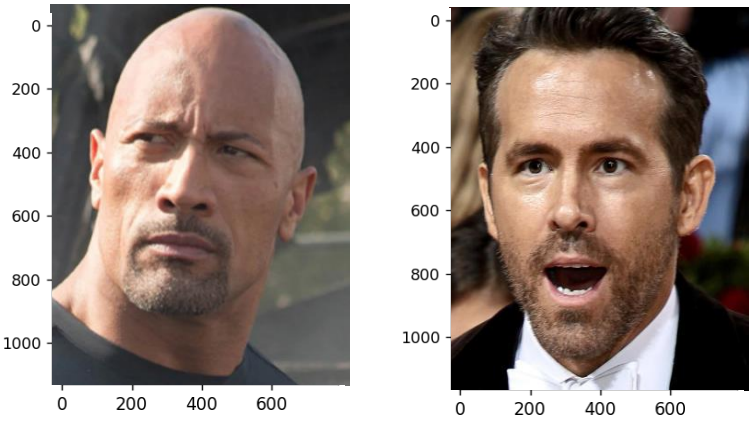
a – amuzament b – concentrare

Emoție diferită



a – tristețe b – neutru

Emoție diferită



a – furie b – surprindere

Emoție diferită

Figura 3.3: Exemplu predicție corectă între emoții diferite

3.3 Colectarea de noi imagini

Pentru a mă asigura de conformitatea funcționării aplicației privind evaluarea similitudinii expresiilor faciale, am apelat la colegii de facultate în privința unor poze cu ei. Am luat respectiva decizie în baza convingerii faptului că în setul de date inițial, utilizat în antrenarea rețelei, nu existau aceste poze.

Imaginile cu aceștia au evidențiat o gamă amplă de emoții precum amuzament, furie, fericire, tristețe, concentrare, neutru ș.a.m.d., contribuind astfel la evaluarea comprehensivă a funcționalității aplicației.

Așadar, pentru evaluarea aplicației au fost furnizate dublete de imagini ale colegilor manifestând aceleași emoții sau chiar diferite și au fost trecute, pe rând, spre comparare, sub forma a mai multor situații. Astfel, au fost comparate atât persoane de același gen, prezentând o emoție diferită sau aceeași emoție, cât și persoane de gen diferit.

În imaginea de mai jos se poate observa cum cele două poze supuse procesului, amândouă înfățișând aceeași emoție – amuzament, au oferit un rezultat pozitiv.

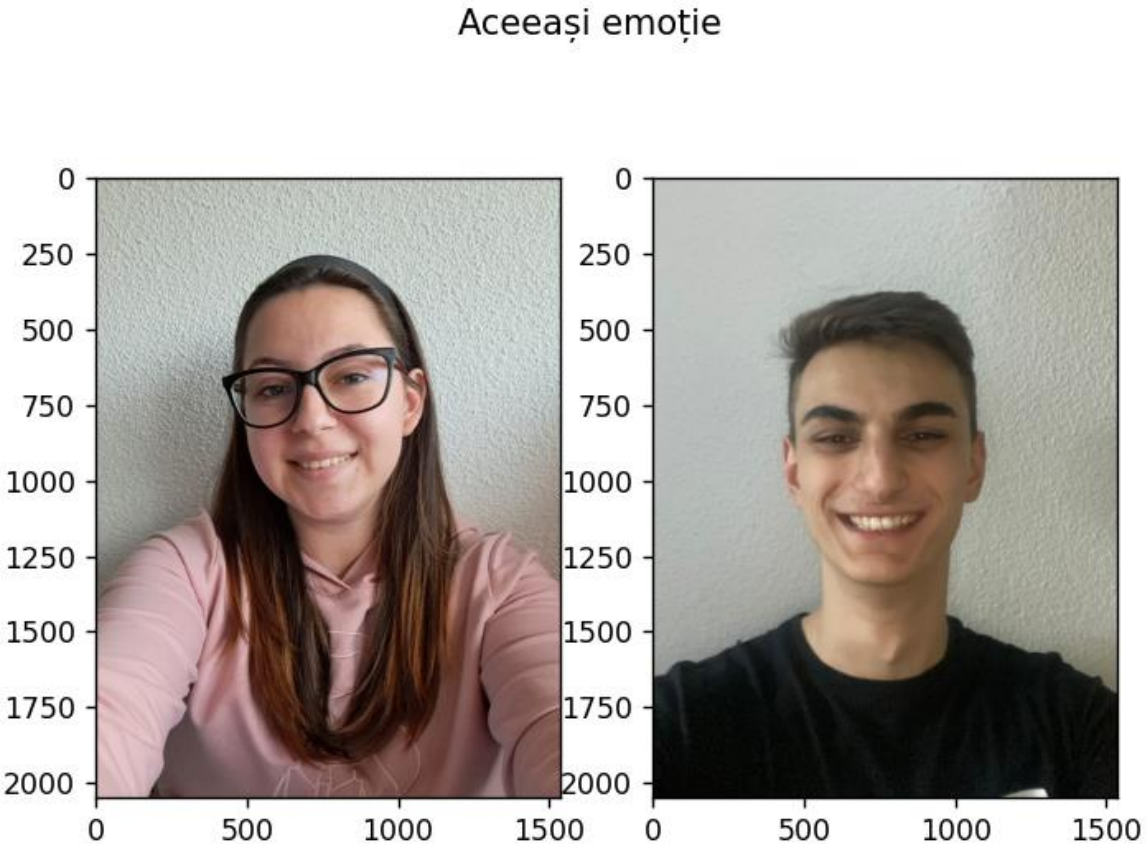


Figura 3.4: Rezultat pozitiv în cazul testării aceleași emoții – amuzament

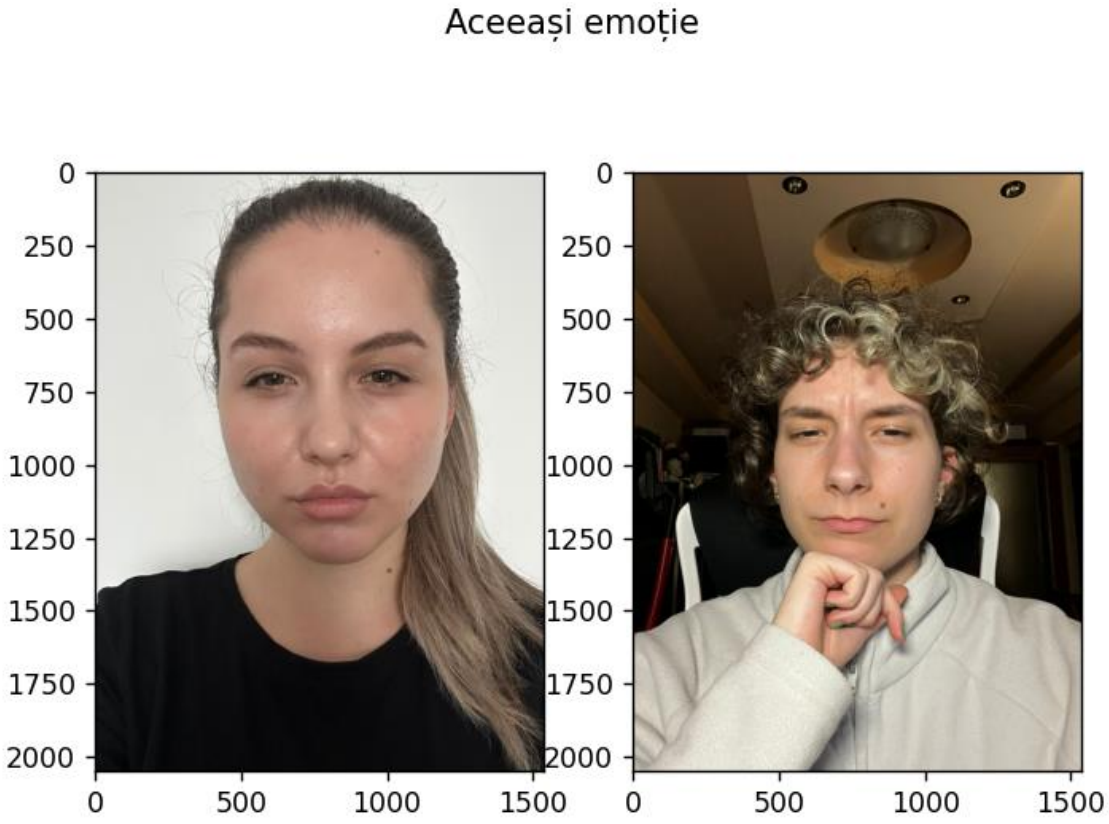


Figura 3.5: Rezultat pozitiv în cazul testării aceleași emoții – concentrare

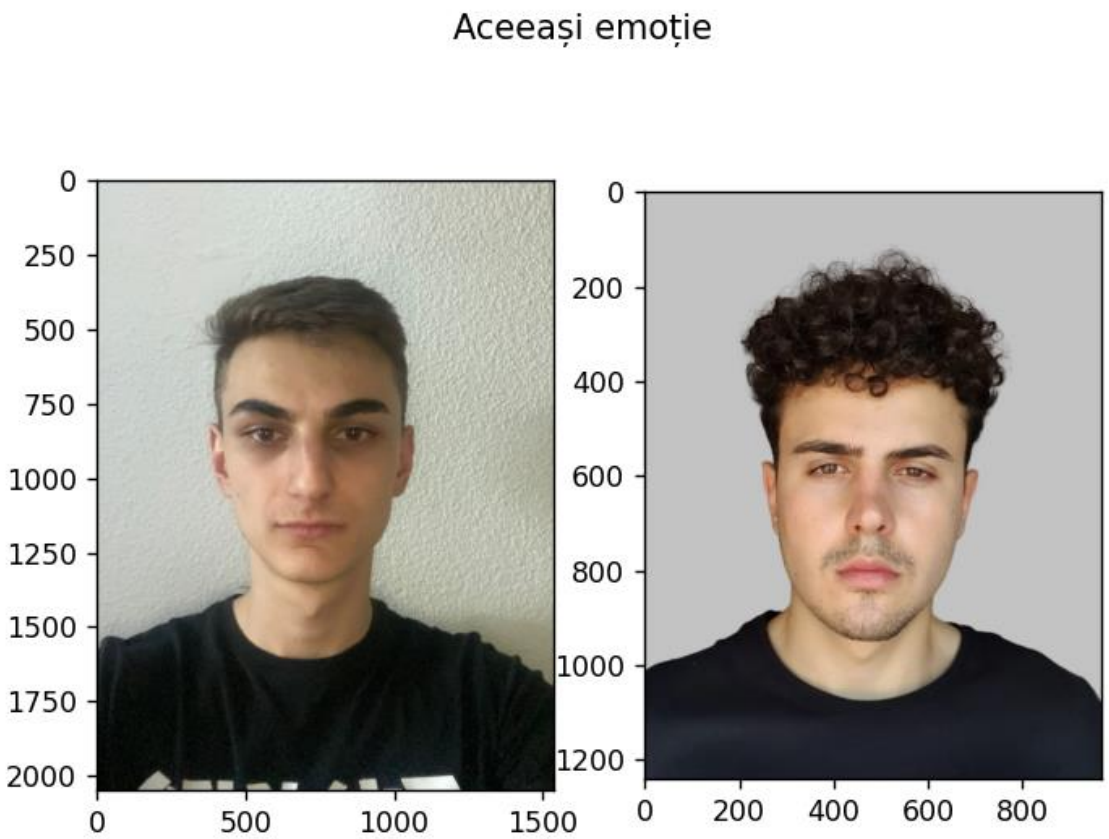


Figura 3.6: Rezultat pozitiv în cazul testării aceleași emoții – neutru

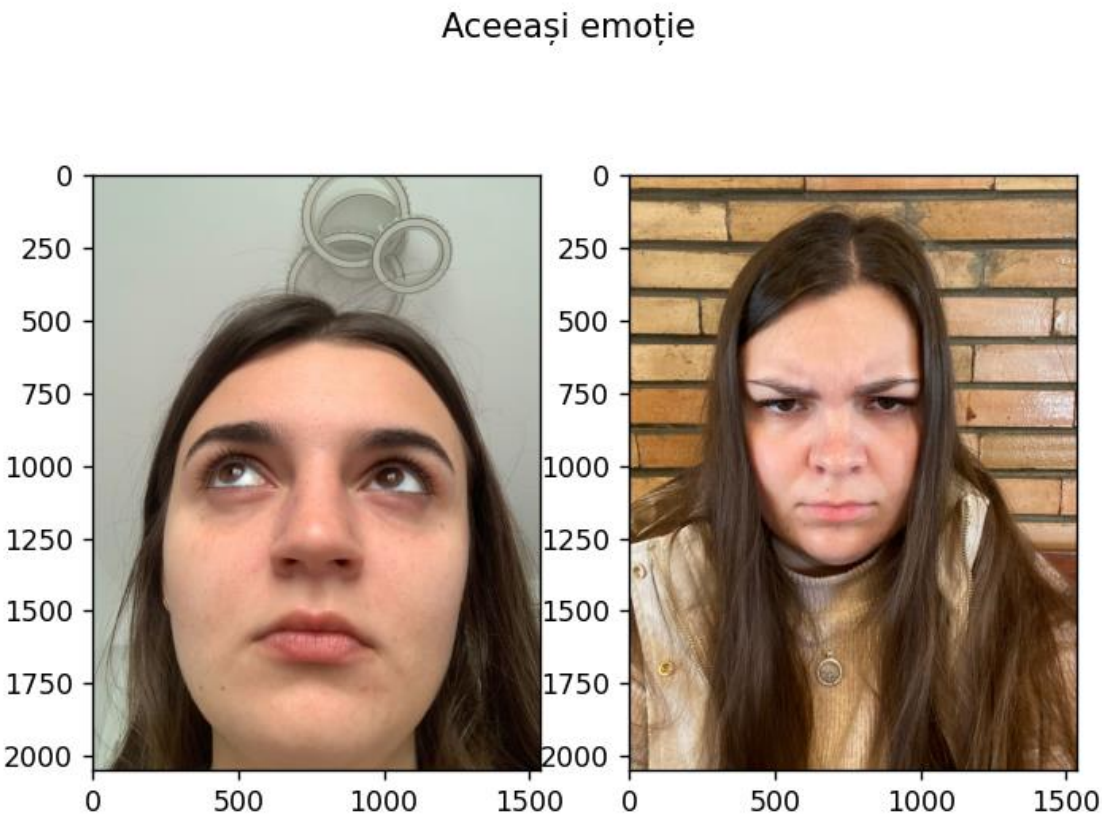


Figura 3.7: Rezultat pozitiv în cazul testării aceleași emoții – furie

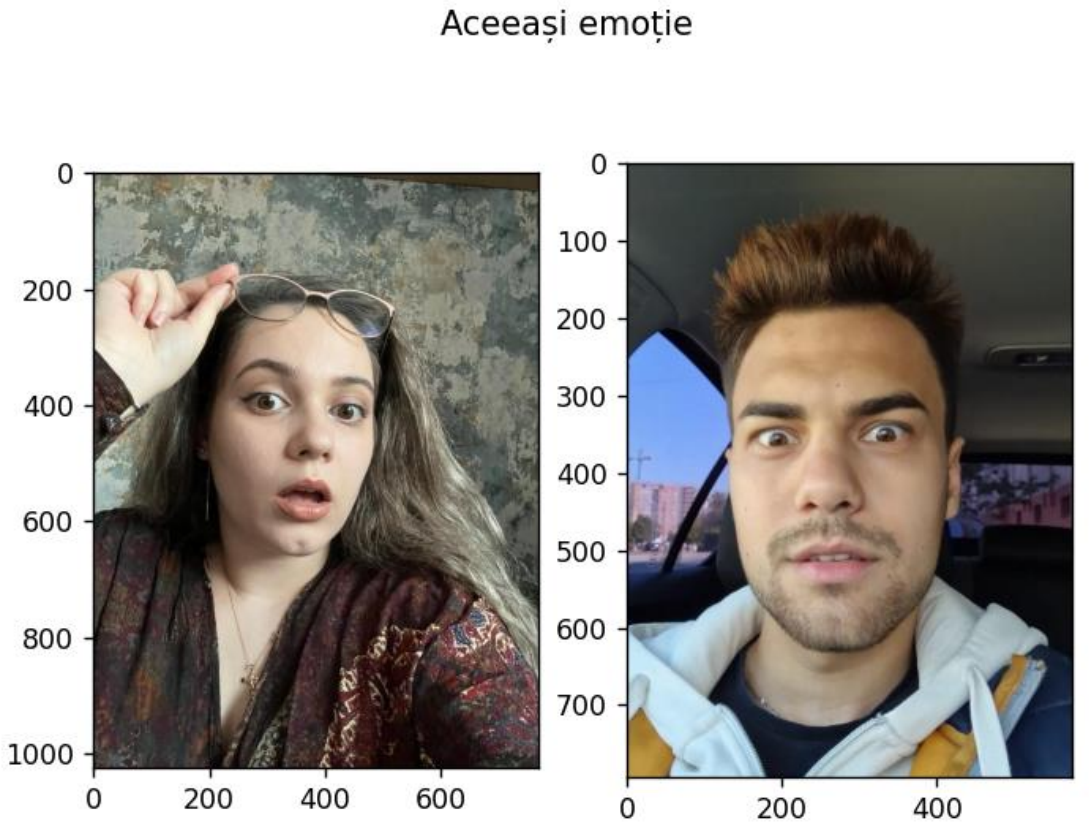


Figura 3.8: Rezultat pozitiv în cazul testării aceleași emoții – surprindere

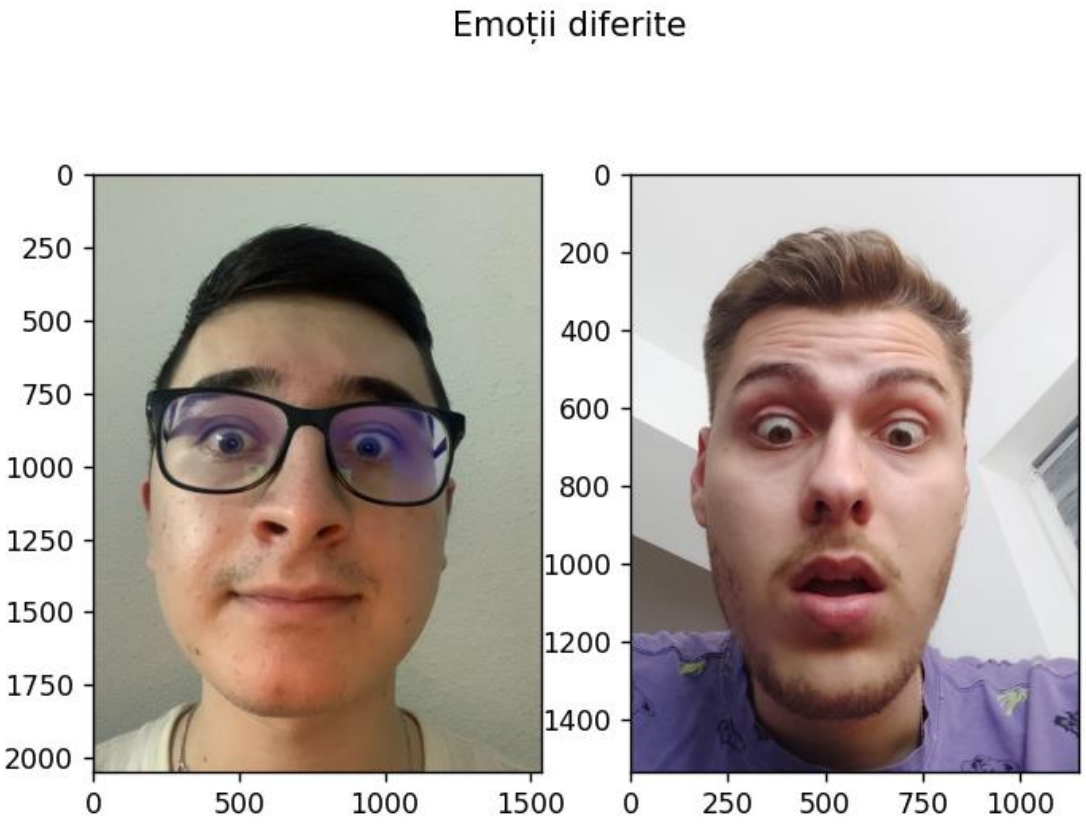


Figura 3.9: Rezultat negativ în cazul testării aceleași emoții – surprindere

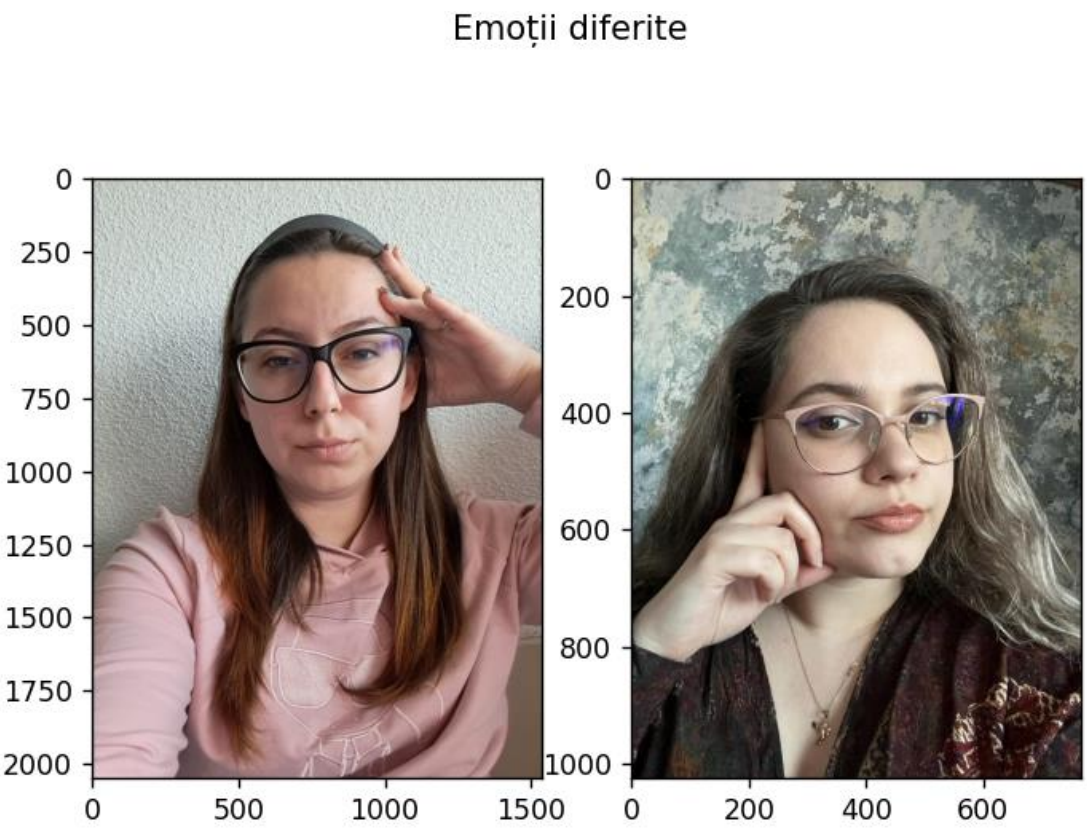


Figura 3.10: Rezultat negativ în cazul testării aceleași emoții – concentrare

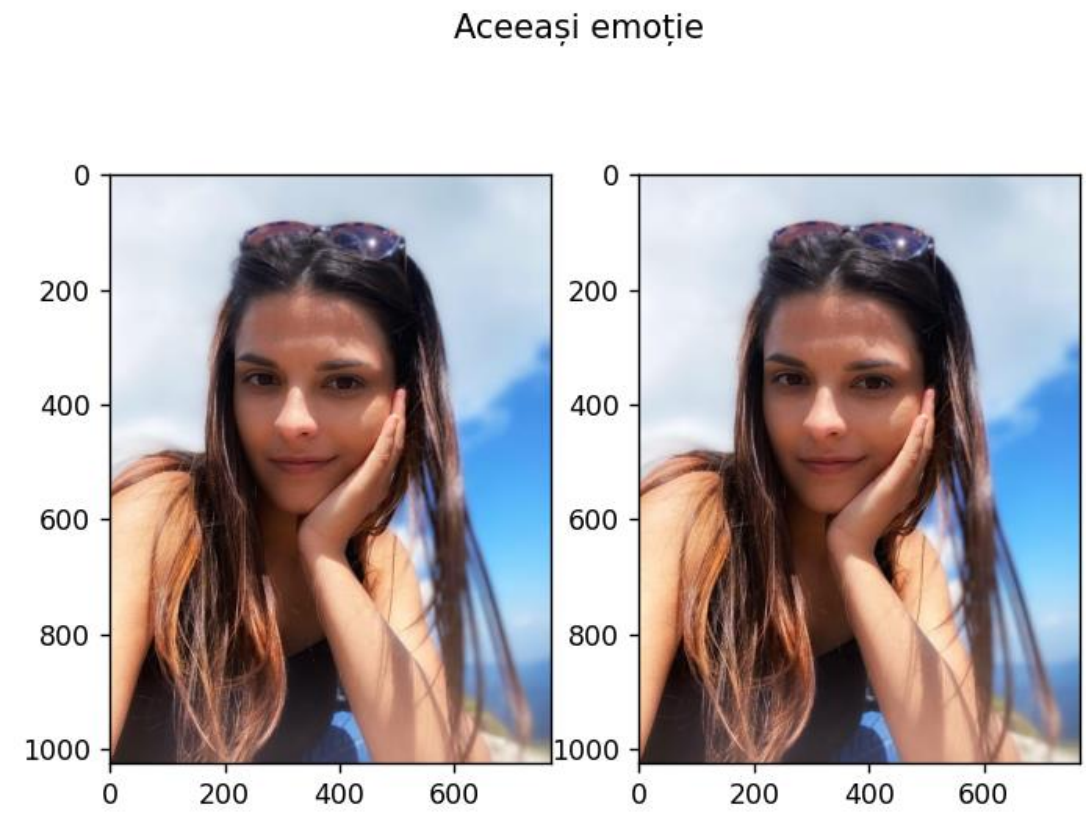


Figura 3.11: Rezultat pozitiv în cazul testării aceleași imagini

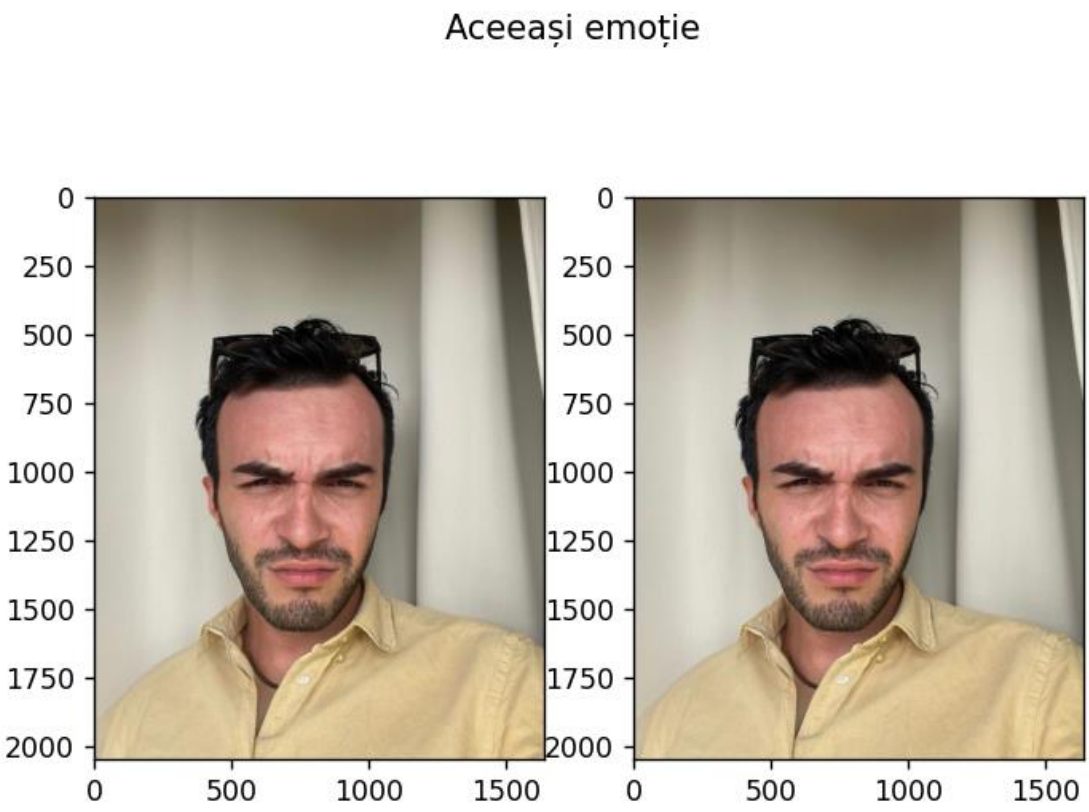


Figura 3.12: Rezultat pozitiv în cazul testării aceleași imagini

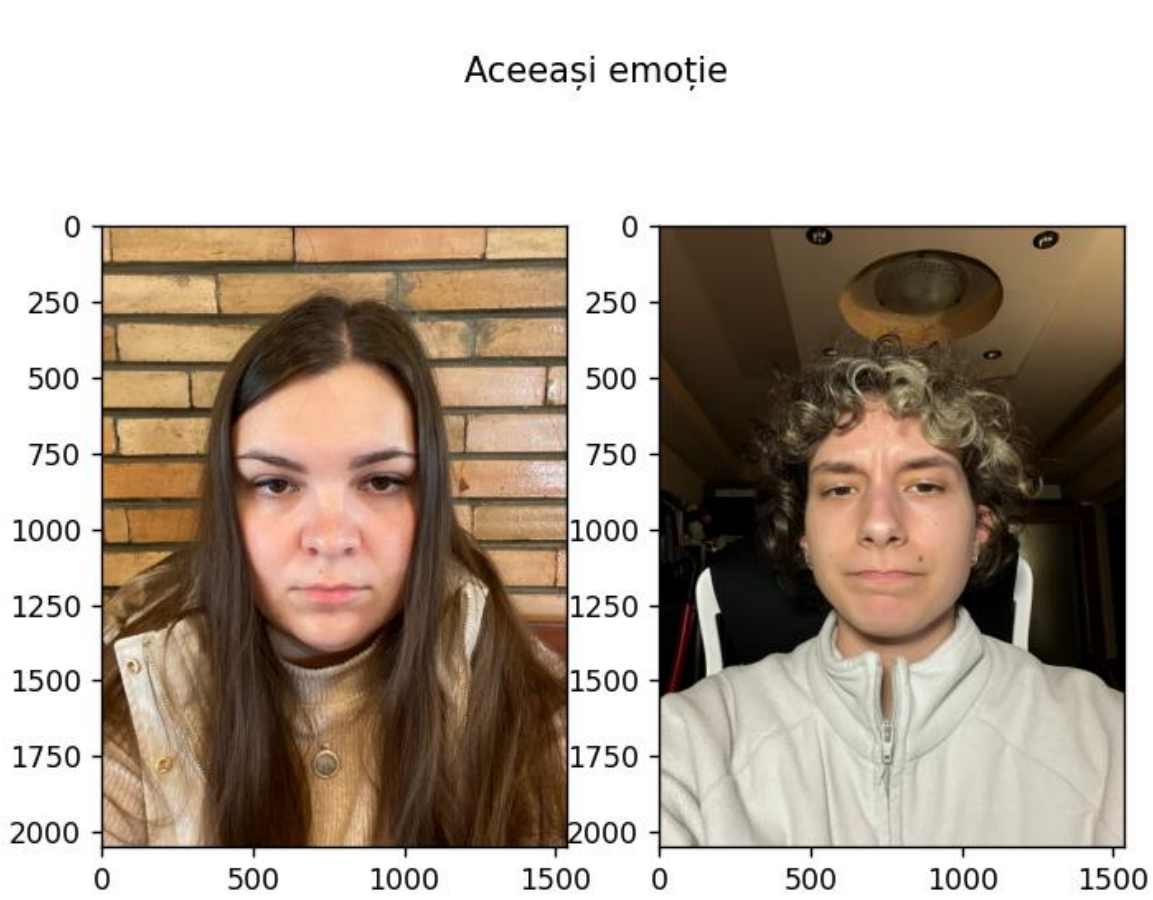


Figura 3.13: Rezultat negativ în cazul testării a două expresii diferite

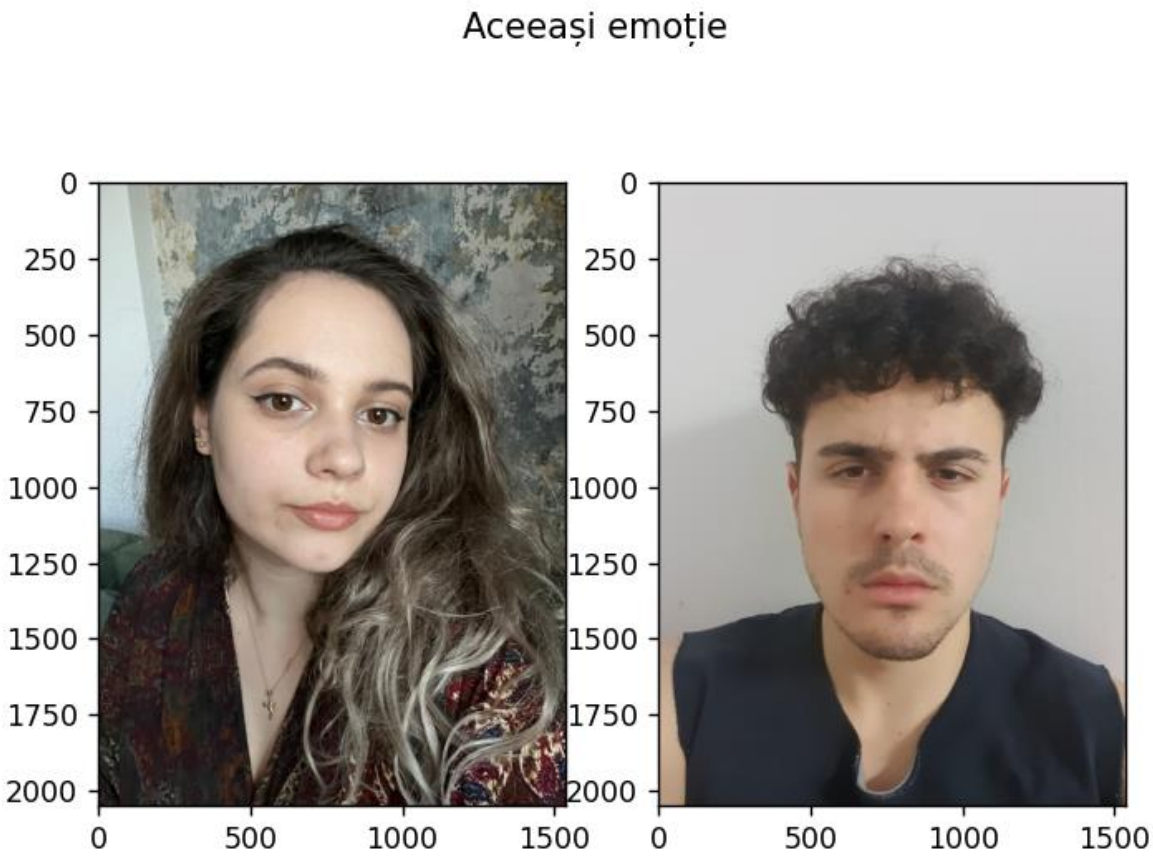


Figura 3.14: Rezult negativ în cazul testării a două expresii diferite

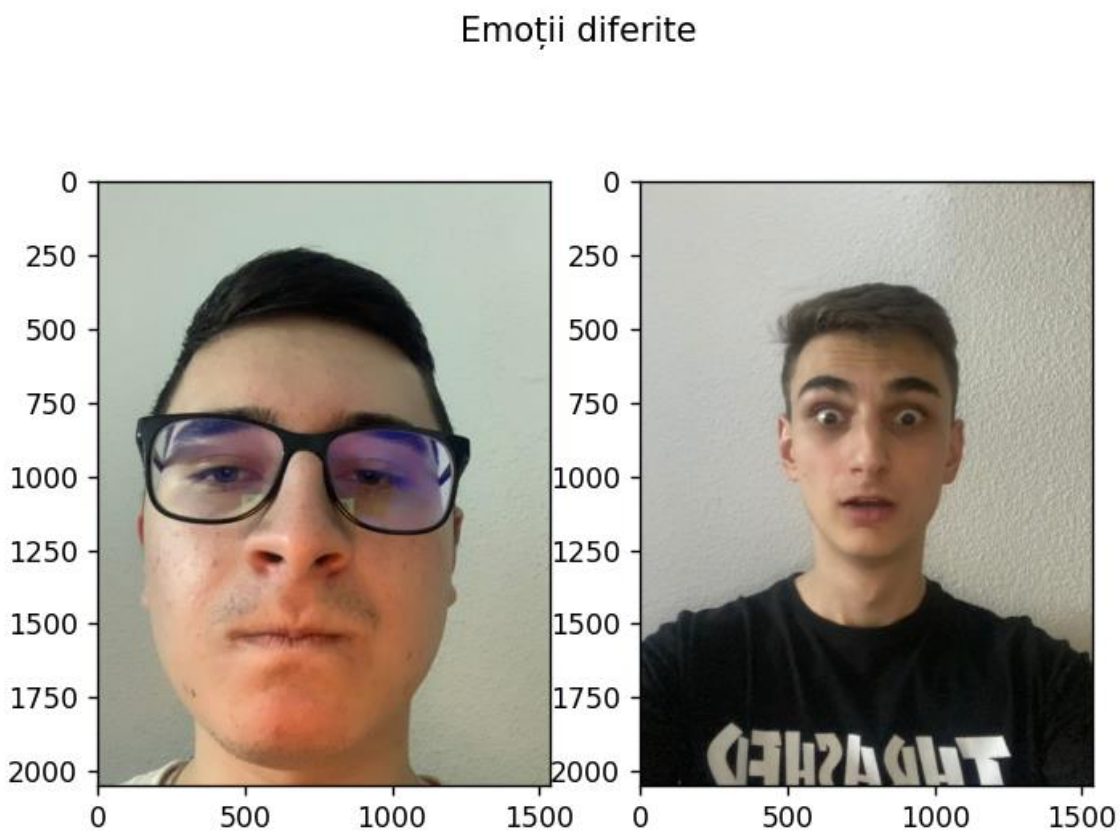


Figura 3.15: Rezultat pozitiv în cazul testării a două expresii diferite

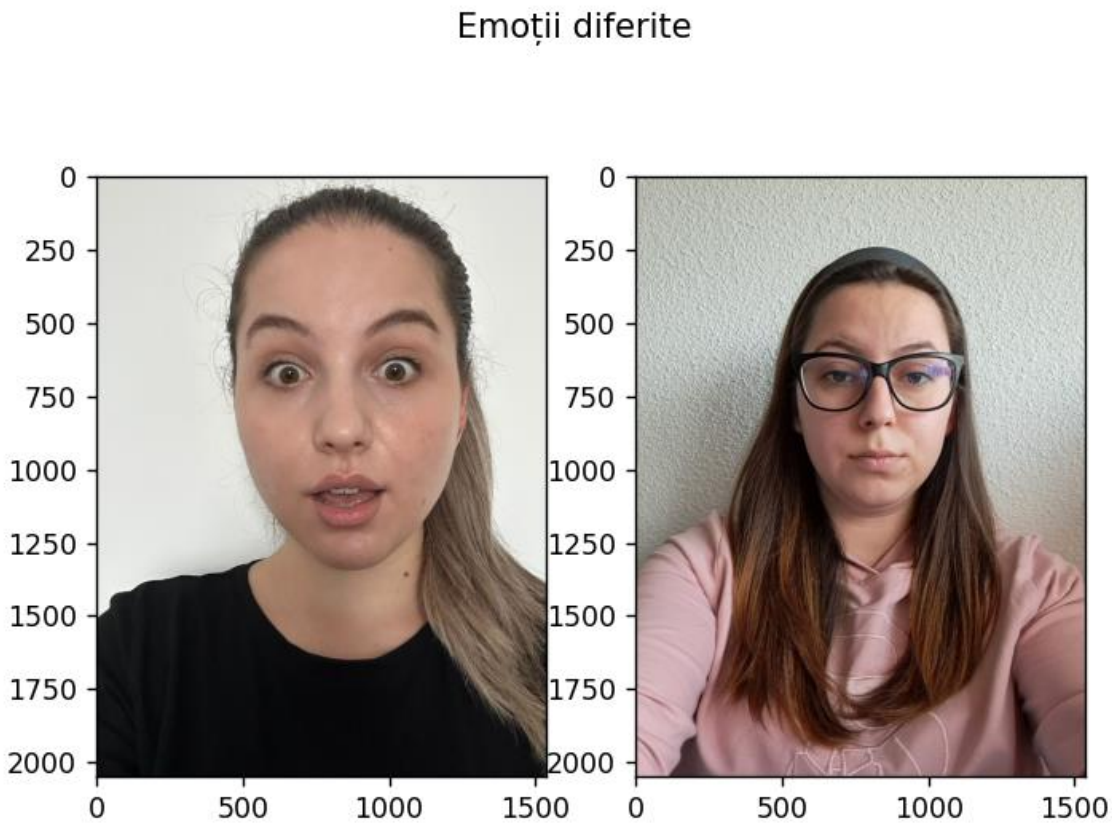


Figura 3.16: Rezultat pozitiv în cazul testării a două expresii diferite

După cum mă așteptam, întrucât acuratețea programului nu este suficient de înaltă, acesta întâmpină mici dificultăți în recunoașterea expresiilor faciale similare. Cu toate acestea, după cum se poate observa în imaginile oferite anterior, aplicația furnizează un rezultat corect în concordanță cu așteptările stabilite.

Unul dintre factorii care poate contribui la discrepanțele dintre funcționarea programului și așteptările dorite, poate fi reprezentat de insuficiența diversității și diferențierii setului de date supus antrenării. După cum se poate observa în figura 3.9, în cazul comparării a două imagini ce reprezintă aceeași stare, aplicația furnizează un rezultat negativ, indicând faptul că cele două prezintă două emoții diferite. Această discrepanță poate fi atribuită setului de antrenare, întrucât persoanele care exprimă surprindere sunt, de cele mai multe ori, ilustrate în imagini având gura deschisă, aspect care lipsește din prima imagine supusă comparației.

4.Concluzii

4.1 Principalele concluzii și rezultate

Scopul prezentei lucrări constă în implementarea unui sistem automat care să determine, pe baza a două fotografii cu expresii faciale, dacă acestea prezintă o emoție similară sau diferită. Acest obiectiv a fost realizat prin intermediul inteligenței artificiale, utilizând tehnica Deep Learning. În urma lucrării practice s-a dezvoltat o rețea neuronală capabilă de recunoașterea expresiilor faciale similare.

Astfel, această temă a condus la explorarea rețelelor neuronale adânci, a modelelor deja existente și a modului lor de funcționare, la modificarea arhitecturii rețelei convoluționale, a funcției obiectiv și a altor hiperparametrii astfel încât să se optimizeze performanța.

Cu toate că rețeaua dezvoltată a obținut rezultatele așteptate în urma antrenării pe un set de 22.000 de fotografii, pentru a utiliza această rețea într-un proiect mai amplu, care implică situații reale, este necesar un antrenament extins, bazat pe un număr mai mare de imagini și o simulare mai avansată, conducând astfel și la necesitatea unor resurse mai bogate.
Așadar, prin intermediul unei puteri mai mari de procesare, conducând astfel și la o perioadă mai îndelungată de antrenare, o astfel de tehnologie poate prezenta o soluție viabilă pentru viitoarele sisteme de recunoaștere a expresiilor faciale.

În opinia mea, acesta reprezintă un subiect important și de actualitate, întrucât utilizarea rețelelor convoluționale este tot mai răspândită și folosită în zilele noastre, oferind un potențial uriaș în domeniul tehnologiei. Aceste rețele permit extragerea unor caracteristici semnificative din imagini și învățarea automată a acestora, conducând astfel la dezvoltarea de noi modele mai precise și mai robuste.

În concluzie, consider că scopul lucrării de față a fost atins, noțiunile necesare acesteia au fost parcurse, înțelese și asimilate cu succes, reușind astfel dezvoltarea unei aplicații ușor de utilizat, cu o acuratețe bună proporțională cu numărul de fotografii supuse antrenării.

4.2 Dezvoltări ulterioare

Unul dintre aspectele remarcabile ale rețelelor convoluționale îl reprezintă flexibilitatea lor, întrucât aceasta permite adaptări ale arhitecturii în vederea obținerii unei învățări automate avansate. În cadrul acestei lucrări, în care sunt utilizate două fotografii drept date de intrare, se pot efectua modificări la nivelul arhitecturii rețelei, incluzând introducerea sau eliminarea anumitor straturi. O altă îmbunătățire semnificativă poate fi reprezentată de utilizarea unui set de date diferit, mai extins, care să ofere o gamă diversificată de imagini cu o acuratețe sporită în redarea expresiilor faciale. Acest aspect nu se referă

neapărat la numărul de imagini, deoarece setul de date inițial, prin intermediul căruia am putut realiza antrenarea, conține 500 de mii de perechi de fotografii, însă calitatea acestora în ceea ce privește reprezentarea expresiilor faciale nu este una extraordinară. Deci, prin colectarea de noi fotografii cu o calitate superioară în ceea ce privește reprezentarea expresiilor faciale, programul poate fi ajutat în obținerea unei acurateți mai bune, rezultatele finale fiind astfel îmbunătățite.

De asemenea, putem aduce modificări și în privința numărului de intrări în program. Astfel, în loc să se utilizeze doar două fotografii pentru comparație, aplicația poate fi adaptată pentru a permite comparația a trei imagini, iar fiecărei posibilă potrivire să îi fie atribuită unei clasă specifice de adnotări. Spre exemplu, adnotările pot lua valori întregi din mulțimea {1, 2, 3}. O valoare de 1 semnifică faptul că expresiile de pe imaginile 2 și 3 din tripletele de poze sunt vizual mai similare în comparație cu expresia facială din prima imagine, o valoare de 2 însemnând că expresiile faciale de pe prima și a treia fotografie din triplet sunt mai similare în comparație cu cea de-a doua imagine, iar o valoare de 3 indicând faptul că expresiile faciale din prima și din a doua imagine sunt considerate vizual mai asemănătoare decât expresia din a treia fotografie. Așadar, mărirea numărului de intrări poate determina o perspectivă mai completă asupra expresiilor faciale și obținerea unei valori mai mari a acurateții în determinarea similarității sau diferenței expresiilor faciale.

Bibliografie

[1] *Typical CNN Architecture*, https://www.interviewbit.com/blog/cnn-architecture/ , accesat la data de: 27.03.2023

[2] *Inside convolutional neural networks*, https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network , accesat la data de 27.03.2023

[3] *Convolutional Layer*, https://www.ibm.com/topics/convolutional-neural-networks , accesat la data de 27.03.2023

[4] *Sigmoid, Softmax and their derivatives*, https://themaverickmeerkat.com/2019-10-23-Softmax/, accesat la data de 23.04.2023

[5] *Simple Introduction to Convolutional Neural Networks*, https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac , accesat la data de 23.04.2023

[6] *Batch Norm Explained Visually — How it works, and why neural networks need it* https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739, accesat la data de 09.05.2023

[7] *Dropout in Neural Networks*, https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9, accesat la data de 09.05.2023

[8] *Siamese Network with application to face similarity,* https://datahacker.rs/019-siamese-network-in-pytorch-with-application-to-face-similarity/ , accesat la data de 25.05.2023

[9] *ResNet — Understand and Implement from scratch* , https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db, accesat la data de 27.05.2023

[10] Machine Learning – Categories of Machine Learning, https://www.tutorialspoint.com/machine_learning/machine_learning_tutorial.pdf , accesat la data de 13.06.2023

[11] Pytorch, https://pytorch.org/about/ , accesat la data de 19.06.2023

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

Anexe

 # Set de date

```
class FacialExpresionDataset(Dataset):
    def __init__(self, image_paths, transform):
        self.image_paths = image_paths
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_filepath = self.image_paths[idx]
        img1 = image_filepath[0][0]
        img2 = image_filepath[0][1]

        image1 = Image.open(img1)
        image2 = Image.open(img2)

        label = image_filepath[1]

        if self.transform:
            image1 = self.transform(image1)
            image2 = self.transform(image2)

        return image1,  image2,  label
```

 # Model

```
class Model(nn.Module):

    def __init__(self):
        super(Model, self).__init__()
        # se înarcă modelul ResNet18 preantrenat și se îngheață ponderile acestuia
        net = models.resnet18(pretrained=True)
        for param in net.parameters():
            param.requires_grad = False
        # se îndepărtează stratul FC de la final
        self.net = nn.Sequential(*list(net.children())[:-1])
        # se adăugă straturi liniare pentru a compara caracteristicile celor două imagini
        self.fc = nn.Sequential(
            nn.Linear(512 * 2, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, 1),
        )

        self.sigmoid = nn.Sigmoid()

    def forward_once(self, x):
        output = self.net(x)
        output = output.view(output.size()[0], -1)
        return output
```

```
    def forward(self, input1):
        # se obțin caracteristicile celor două imagini
        output1 = self.forward_once(input1[0])
        output2 = self.forward_once(input1[1])

        # se concatenează
        output = torch.cat((output1, output2), 1)

        # se trece la straturilor liniare
        output = self.fc(output)

        # se trece la stratul sigmoid
        output = self.sigmoid(output)

        return output
```

 # Buclă antrenare

```
def train(model, device, train_loader, optimizer, epoch, criterion):
    """
    model: modelul declarat de noi anterior
    device: cuda / cpu in functie de disponibilitate
    train_loader: datele de antrenare
    optimizer: optimizatorul folosit
    epoch: numarul de epoci
    criterion: functia de loss

    setam modelul in modul de antrenare, trecem prin datele din setul de antrenare, comparam rezultatul
    cu valoare reala, calculam eroarea si facem backword propagation, acest proces se reia pana sa
    termina toate batch-urile si ulterior se reia in functie de numarul de epoci
    """
    model.train()

    # nu folosim `TripletLoss`, ci`BCELoss`.

    batch_idx = 0
    for images_1, images_2, targets in train_loader:
        batch_idx += 1
        images_1, images_2, targets = images_1.to(device), images_2.to(device), targets.to(device)
        targets = targets.to(torch.float32)
        optimizer.zero_grad()
        outputs = model((images_1, images_2)).squeeze()
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        if batch_idx % 1 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(images_1), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))



model = Model()
model = model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=1, gamma=0.1)
```