Universitatea POLITEHNICA din București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Recunoasterea emoțiilor folosind rețele neurale adânci

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea

titlului de *Inginer*

în domeniul *Electronică, Telecomunicații și Tehnologia Informației*

programul de studii *Tehnologii şi Sisteme de Telecomunicaţii*

Conducător științific

Prof. Dr. Ing. Corneliu BURILEANU

Ș.l. Dr. Ing. Anamaria RĂDOI

Dr. Ing. Liviu Cristian DUȚU

Absolvent

Nicolae-Cătălin RISTEA

Anul 2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **Tc**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **RISTEA N. Nicolae-Cătălin , 442C**

1. Titlul temei: Recunoașterea emoțiilor folosind rețele neurale adînci

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) și specificații de proiectare:
Lucrarea are drept scop crearea unui sistem automat de recunoaștere a emoțiilor pe baza mișcării mușchilor faciali. Aplicațiile sistemului sunt multiple și au la bază folosirea diverselor mijloace tehnologice în sprijinul individului; de exemplu, tratarea stărilor repetate de nemulțumire, creșterea indicelui de fericire, atingerea unei stări de echilibru. În acest sens, lucrarea de diplomă implică studiul literaturii de specialitate în ceea ce privește legătura dintre mișcările anumitor mușchi faciali și emoția exprimată. Sistemul de recunoaștere va alege dintre un număr de emoții discrete pe cea mai probabilă, în concordanță cu starea subiectului urmărit cu ajutorul unei camere video. Datele (atît cele de antrenare, cît și cele de testare) necesită o pre-procesare, realizată în Python, cu ajutorul bibliotecii TorchVision. Rețeaua neurală va fi implementată în PyTorch și va fi compusă din două parți: extractor de caracteristici și clasificator. Extractorul de caracteristici este compus din straturi de convoluții, blocuri ReLU și MaxPooling. Clasificatorul este format din straturi de neuroni complet conectați și o funcție de tip Sigmoid sau SoftMax. În acest proiect, se vor testa mai multe tipuri de rețele neurale, atît dezvoltate în cadrul proiectului de diplomă, cît și existente în literatura de specialitate pentru a fi efectuate comparații cu rezultatele obținute prin arhitecturile propuse. Sistemul complet va fi implementat în Python și va fi testat atît pe baza de date de referință, cît și pe o bază de date creată de student. De asemenea, se va testa funcționarea sistemului în timp real.

3. Resurse folosite la dezvoltarea proiectului:
Limbaje de programare: Python, C++ Biblioteci: numpy, PyTorch, TorchVision, OpenCV Dispozitive: NVIDIA GPU M4000, CPU

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
DEPI, PDS, POO

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

6. Data înregistrării temei: 2018-11-28 10:15:04

Conducător(i) lucrare,
Ş. L. Dr. Ing. Anamaria RĂDOI
semnătura: .....

Student,
semnătura: .....

BURILEANU Corneliu, FOTONATION SRL
semnătura: .....

Director departament,
Conf. dr. ing. Eduard POPOVICI
semnătura: .....

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

Cod Validare: **5585a343b5**

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul *Recunoasterea emoțiilor folosind rețele neurale adânci*, prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul Inginerie Electronică şi Telecomunicaţii/ Calculatoare şi Tehnologia Informaţiei, programul de studii *Tehnologii şi Sisteme de Telecomunicaţii* 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, Iunie 2019.

Absolvent: Nicolae-Cătălin RISTEA

.....

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

L1 = Normă euclidiană

L2 = Normă pătratică

PAL = Linie alternantă a fazei (engl. *Phase Alternating Line)*

PCA = Analiza componentelor principale (engl. *Principal component analysis*

SVM = Vectori suport (engl. *Support Vectors Machine*)

Capitolul 1

Introducere

Emoțiile sunt răspunsuri psihice de mare intesitate care presupun manifestări expresive, fiziologice și subiective tipice. Ele sunt însoțite de tendințe spre acțiune (apropiere sau îndepărare de obiectul sau ființa care a declanșat emoția), ceea ce explică modificările fiziologice care însoțesc fenomenul emoției, spre exemplu: înroșirea feței, creșterea ritmului cardiac, încordarea mușchilor. Comunicarea interumană se realizează prin diferite mijloace, inclusiv prin emoții. A fi capabil să întelegi emoțiile unui om și trăirile interioare ale acestuia este un factor cheie în creșterea eficienței comunicării.

Mai mult decât atât, cercetările în domeniul roboticii, în special în cel al roboților umanoizi, au început să ia amploare, ceea ce evidențiază un interes în acest domeniu. Devine foarte interesantă și atractivă partea de comunicare om-mașină și modalitățile de realizare a acesteia. De aceea, un modul pentru detecția și înțelegerea emoțiilor este în atenția cercetătorilor. Scopul acestui proiect este de a realiza un modul de recunoaștere a emoțiilor, care poate fi folosit în diverse scopuri.

Interesul pentru acest domeniu a condus la dezvoltarea unui sistem de codificare a mișcărilor faciale [2], cu ajutorul căruia se poate descrie orice mișcare a mușchilor faciali. Așadar, o emoție este o mulțime finită de mișcări faciale ce sunt executate concomitent. În literatura de specialitate se afirmă că există o serie de emoții primare, care pot fi grupate pentru a nuanța mai fin trăirile unei persoane. Vom detalia aceste emoții și modul lor de manifestare, din punct de vedere al expresiei.

Tristețea are următoarele caracteristici: gura este deschisă, colțurile buzelor sunt în jos, colțurile interioare ale sprâncenelor sunt ridicate, privirea este în jos. *Furia* are

următoarele caracteristici: sprâncenele sunt trase în jos, cu colțurile interioare spre nas, cu sprâncenele în jos ochii sunt larg deschiși (pleoapele împing în sprâncenele care sunt trase în jos), buzele sunt împreunate și încordate fără să fie încrețite. *Disprețul* are următoarele caracteristici: pe o singură parte a feței, colțul buzei este tensionat și ușor ridicat. *Dezgustul* are următoarele caracteristici: buza superioară este ridicată la maximum, buza de jos este ridicată la rândul ei (ieșind puțin în afară), nările sunt ridicate și nasul este încrețit. *Surprinderea* are următoarele caracteristici: sprâncenele sunt înălțate și curbate. Această înălțare produce riduri orizontale de-a lungul frunții (deși aceste riduri nu apar la toate persoanele). Ochii sunt larg deschiși, cu pleoapa de deasupra ridicată și cea de jos relaxată (și nu tensionată ca în cazul fricii), gura este deschisă. *Frica* are următoarele caracteristici: pleoapele superioare sunt ridicate, cele inferioare sunt tensionate, ridurile deasupra sprâncenelor sunt în centrul frunții și nu pe toată lungimea frunții (ca la surpriză) gura deschisă și buzele sunt trase orizontal spre urechi (alungirea gurii), sprâncenele ridicate, ochii sunt larg deschiși. *Fericirea* are următoarele caracteristici: colțurile buzelor sunt ridicate, dinții pot fi sau nu expuși, obrajii sunt ridicați, o dezchidere a ochilor mai mică.

Figura 1.1: Exemple de imagini din seturile de date MMI (partea de sus), CKP (partea de jos). Emoțiile de la stânga la dreapta: Furie, Tristețe, Dezgust, Fericire, Frică, Surprindere

Astfel, după o înțelegere mai profundă a emoțiilor, cercetătorii au început să utilizeze o gamă variată de metode pentru a face o clasificare automată. Tehnici precum utilizarea unor particularități definite manual de către ingineri sau folosirea SVM-urilor au fost preferate la început din considerente tehnologice. Odată cu trecerea timpului, bazele de date pentru această sarcină s-au dezvoltat, fiind mult mai reprezentative și cu o gamă

mai largă de persoane și emoții. De asemenea, puterea de calcul, cât și tehnologiile de inteligență artificială au crescut exponențial, lucru favorabil pentru a încerca și dezvolta noi metode. Această evoluție a condus la încercări bazate pe rețele convoluționale. Trecerea la rețele neurale a condus la îmbunătățirea rezultatelor într-un mod deosebit. Acestea au demonstrat rezultate practice extrem de bune, ceea ce dovedește capabilitatea lor de a extrage foarte bine acele trăsături abstracte care definesc o emoție.

Capitolul 2

Studii similare

În această secțiune vor fi prezentate încercările făcute până în prezent de cercetători în domeniul recunoașterii emoțiilor, împreună cu rezultatele și concluziile la care s-au ajuns. Vom prezenta mai ales acele metode bazate pe rețele neurale, deoarece au atins rezultate superioare.

2.1 Abordări clasice

Vom detalia o abordare bazată pe procesarea imaginii și găsirea anumitor particularități ce pot ajuta la clasificarea emoțiilor. Ekman and Friesen au propus o soluție în anul 1995 [3], care avea la bază un PCA. În primul rând, ei au propus o procesare inițială a pozelor, care consta în: egalizare de histogramă, detecție și decupare față subiect. Apoi pozele erau redimensionate la o valoare generică propusă și introduse în rețea. Rețeaua era compusă dintr-un perceptron multistrat, care avea la ieșire 6 neuroni, fiecare neuron a fost asociat biunivoc cu o anumită emoție (fericire, tristețe, frică, furie, surprindere, dezgust).

Rezultatele au fost bune, ceea ce a dus la concluziile că PCA este un algoritm folositor în clasificarea imaginilor și nu este nevoie de transformări adaptive pentru a localiza ochii și gura. De asemenea, dacă păstrăm doar 8 valori din nucleele inițiale ale transformării vom obține o acuratețe de 80% din cea inițială.

2.2 Abordări bazate pe rețele neurale adânci

Sarah Adel Bargal și colegii ei propun un algoritm bazat pe rețele neurale, algoritm ce a fost înscris la competiția "Recunoașterea Emoțiilor în Natură" 2016 [4]. Acest proiect este conceput să primească o secvență video și să returneze emoția pe care subiectul aflat în video o exprimă. Ei folosesc un set de rețele preantrenate (VGG13, VGG16, ResNet256) care ajută la extragerea proprietăților specifice emoțiilor. Fiecare rețea are aceeași secvență la intrare și returnează un vector de caracteristici, care mai apoi este normalizat L2. Cei trei vectori sunt concatenați și introduși într-un modul de codificare statistică, care produce un singur vector de carcteristici. Clasificarea se face cu ajutorul unui SVM. Algoritmul are o acuratețe de 59.42% pe setul de validare, depășind nivelul de referință dat de competiție de 38.81%.

Heechul Jung împreună cu alți cercetători de la Școala de Inginerie Electrică și Institutul de Știință și Tehnologie din Coreea [5] propun un algoritm complex, ce ține cont de mai multe aspecte, nu doar de imaginile propriu-zise. Soluția are la bază rețele convoluționale preantrenate cu ajutorul cărora se extrag trăsături ce ajută la clasificarea emoției. Rețeaua este alcatuită din două structuri aflate în paralel: o latură primește la intrare imagini cu fața subiectului decupată, cea de-a doua latură a rețelei primește puncte asociate feței (pe față sunt marcate 49 de puncte, fiecare corespunzând unei zone a feței, spre exemplu colțul buzei din dreapta). Ieșirile celor două subrețele se concatenează într-un vector de caracteristici cu ajutorul căruia se face clasificarea, prin intermediul unor straturi complet conectate. Rezultatele raportate le depășesc pe cele deja existente, astfel că ei afirmă că au obținut 97.25% pe setul de date CK+, 81.46% pe setul de date Oulu și 70.24% pe setul MMI. În medie, algoritmul lor obține rezultate cu 3% mai bune, ceea ce scoate în evidență că atunci când o rețea primește mai multe informații la intrare, poate avea rezultate mai bune.

O abordare extrem de interesantă este prezentată în articolul "Facial Expression Recognition by De-expression Residue Learning" [6], prezentat la CVPR 2018. Aceștia se bazează

pe studii psihologice și afirmă că emoția este alcatuită dintr-o componentă de expresivitate și o componentă neutră a feței. Ceea ce înseamnă că, dacă rețeaua ar primi la intrare o poză cu subiectul în două ipostaze (neutru și cu o anumită emoție) și ar face diferența lor, ar reuși să clasifice mult mai bine emoțiile. Abordarea lor este extrem de complexă, deoarece este necesară o rețea preantrenată de tip codor-decodor, care să primească o poză cu o expresie oarecare și să returneze o poză cu aceeași persoană, însă cu o expresie neutră. Această rețea, denumită generator, are mai multe straturi de abstractizare și pentru a obține componenta de expresie se face o diferență a informației ce parcurge rețeaua pe diversele ei straturi. Vectorii diferență sunt introduși în straturi complet conectate pentru a face clasificarea emoției. Rezultatele obținute sunt extrem de sugestive 97.30% pe setul de date CK+, 88% pe setul Oulu și 75.23% pe setul MMI. Concluzia este că această abordare ce își are fundamentele în psihologie este pertinentă și obține rezultatele cele mai bune din literatură, până la acel moment. Dificultatea acestei abordări provine din necesitatea unei rețele generator, care poate face transformarea din orice expresie în neutru.

Rezultate impresionante au fost obținute și cu abordarea denumită "DeXpression" [7]. Autorii propun o rețea convoluțională clasică cu o singură intrare, însă elementul de noutate constă în arhitectura acestei rețele. Arhitectura este complexă, alcatuită din mai multe ramuri ce se despart, iar mai apoi se concatenează. Datele de intrare constau în imagini cu fața unui subiect, iar ieșirea este un clasificator ce are un neuron pentru fiecare emoție. Rezultatele lor sunt următoarele: 99.6% pe setul CK+ și 98.63% pe setul MMI. În concluzie, arhitectura complexă propusă reușește să extragă într-un mod foarte eficient caracteristicile necesare pentru determinarea emoției.

Capitolul 3

Baze de date

Tranziția de la recunoașterea emoțiilor faciale în laborator, într-un mediu controlat, la recunoașterea lor în natură, unde nu putem ajusta factorii de mediu în niciun fel și succesul recent al rețelelor neurale adânci în diverse domenii, au condus la dorința de a avea mai multe baze de date cu ajutorul cărora rețelele să aibă rezultate superioare. Cele mai importante două probleme în sistemele actuale de clasificare a emoțiilor sunt următoarele: fenomenul de supraînvățare, cauzat de un număr insuficient de date de antrenare și lipsa variațiilor pentru expresii, spre exemplu pentru o aceeași persoană cu o anumită expresie nu sunt variații de iluminare, poziție a capului sau a camerei.

A avea suficiente date marcate în mod corespunzător pentru antrenarea rețelelor neurale, incluzând o variație mare de subiecți și factori de mediu, este extrem de important în vederea obținerii unor rezultate concludente. În această secțiune vom detalia seturile de date folosite în proiect, atât publice cât și cele proprietare, din punct de vedere al numărului de persoane ce au fost înregistrate, distribuția expresiilor, factorii de mediu, cât și alte informații relevante.

3.1 Setul de date Cohn-Kanade extins (CK+)

Acest set de date este dedicat cercetării în domeniul recunoașterii automate a expresiilor cu ajutorul imaginilor. Există două versiuni ale acestuia: prima a fost publicată în anul 2000, iar cea de-a doua în anul 2009. Prima versiune a inclus 486 de secvențe cu 97 de persoane diferite. Fiecare secvență video începe cu o stare neutră a subiectului și continuă

cu o creștere în intensitate a unei anumite expresii. Expresia cu intensitate maximă este marcată conform Sistemului de Codare a Mișcărilor Faciale [8] și îi este atribuită o emoție.

Versiunea a doua este o variantă îmbunătățită și conține expresii simulate cât și spontane, împreună cu date suplimentare. Față de versiunea anterioară numărul de secvențe a crescut cu 22%, iar numărul de persoane a crescut cu 27% [1]. Secvențele au între 10 și 60 de cadre și captează evoluția de la starea neutră la o anumită expresie, identic ca în versiunea anterioară. De asemenea, o îmbunătățire adusă este faptul că emoția marcată este validată de mai mulți marcatori, astfel având un caracter mai general. În plus, CK+ oferă un rezultat de referință în ceea ce privește recunoașterea emoțiilor, astfel cercetătorii își pot compara rezultatele cu unele oficiale.

Tabela 3.1: Distribuția emoțiilor în setul de date CK+ [1]

Emoție	Furie	Dispreț	Dezgust	Frică	Fericire	Tristețe	Surprindere
Număr apariții	45	18	59	25	69	28	83

Figura 3.1: Exemple de imagini din setul de date CK+. Spre exemplu marcajele pentru această poză sunt următoarele: (a) Dezgust - MM 1+4+15+17, (b) Fericire - MM 6+12+25, (c) Surprindere - MM 1+2+5+25+27, (d) Frică - MM 1+4+7+20, (e) Furie - MM 4+5+15+17, (f) Dispreț - MM 14, (g) Tristețe - MM 1+2+4+15+17, (h) Neutru

3.2 Setul de date Man-Machine Interaction (MMI)

Setul de date este destinat cercetării în domeniul recunoașterii emoțiilor și a fost gândit pentru a fi ușor accesibil corpurilor de cercetare. Acesta conține peste 1500 de exemple,

fie ele statice sau secvențe de imagini, cu fețe în poziție frontală și profil, care exprimă o varietate de expresii faciale.

Imaginile statice sunt colore și au dimensiunile de 720x576 pixeli. Există aproxivativ 600 de imagini cu subiecții în poziție frontală și 140 cu o dublă perspectivă, frontal și profil. Toate secvențele video au fost înregistrate cu o rată de 24 de cadre pe secundă, utilizând o cameră PAL standard. În setul de date sunt aproximativ 30 de exemple filmate în profil și 750 în dublă perspectivă [9]. Secvențele au lungime variabilă în intervalul 40 - 520 cadre și pornesc cu subiecții într-o stare neutră, continuând cu o expresie, iar mai apoi din nou cu starea neutră.

Setul de date include 32 de persoane diferite (studenți și membrii ai corpului de cercetare respectiv) cu o proporție de sexe feminin : masculin de 44% : 56%. Plaja vârstelor este cuprinsă în intervalul 19-62 de ani, iar rasele sunt diverse: European, Asiatic, Sud-American. Subiecții au fost nevoiți să simuleze o serie de 79 de expresii, serie ce include chiar și mișcarea unui singur mușchi facial sau a unei asocieri de mușchi. Ei au fost instruiți de un expert în codificarea mișcărilor faciale pentru a executa corect expresiile dorite. Mai mult decât atât, acest set de date conține imagini mult mai provocatoare din punct de vedere al recunoașterii emoțiilor, deoarece persoanele execută o expresie în mod neuniform, iar multe dintre ele poartă accesorii precum ochelari, iar unii dintre barbați au o barbă proeminentă, ce obturează trăsăturile feței [10].

3.3 Setul de date Emoții în mod multimodal (CREMA-D)

Acest set de date este destinat cercetării în domeniu și conține 7442 de clipuri, de la 91 de actori. Dintre aceștia 48 au fost bărbați și 43 femei cu vârste cuprinse între 20 și 74 de ani, cu o varietate relativ ridicată în ceea ce priveste rasele (african, asiatic, caucazian și latin)[11].

Setul de date este multimodal, ceea ce presupune prezența unei secvențe video împreună cu sunet. Actorii au rostit 12 fraze diferite, cu intonații în concordanță cu emoția dorită

Figura 3.2: Exemple de imagini din setul de date MMI. Emoțiile sunt de la stânga la dreapta: furie, dezgust, frică, fericire, tristețe, surprindere

(furie, dezgust, frică, fericire, neutru, trist) și patru nivele de intensitate (ușor, mediu, ridicat, nespecificat). Pentru a avea o referință corectă a emoțiilor din baza de date, 2443 de participanți au contribuit la clasificarea exemplarelor produse de actori. Astfel peste 95% din filmulețe au fost clasificate de mai mult de 7 persoane diferite.

Autorii acestui set de date au afirmat că acuratețea umană este următoarea: audio 40.9%, vizual 58.2%, multimodal audio împreună cu vizual 63.6%.

3.4 Setul de date FotoNation

Acesta este un set de date proprietar FotoNation [1]. Datele au fost achiziționate în laborator, iar subiecții au fost actori pentru a avea o veridicitate a emoțiilor cât mai bună. De asemenea, scenariile pentru achiziționarea emoțiilor au fost făcute de regizori pentru a face cât mai naturală starea subiectului. Setul conține peste 10,000 de exemple, atât video cât și imagini singulare. În baza de date există multiple unghiuri pentru fiecare emoție, iar achiziția a fost facută în domeniul vizibil și în infraroșu apropiat. Imaginile au fost

[1] https://www.fotonation.com/

marcate de FotoNation, iar la marcare au luat parte trei persoane care au documentat prezența anumitor mișcări faciale, ale emoțiilor și intensității lor. De asemenea, trebuie evidențiat că emoțiile luate în considerare în acest set de date sunt următoarele: neutru, fericire, tristețe, surprindere, dezgust, frică, furie.

Capitolul 4

Fundamente teoretice

4.1 Spectrograma

Spectrograma este o reprezentare vizuală a spectrului de frecvențe a unui semnal pe masură ce acesta variază în timp. Această noțiune este folosită mai ales în muzică, radar, procesare de voce și sunet. Aceasta este reprezentată sub forma unei imagini cu axa timpului pe orizontală, frecvența pe verticală și amplitudinea semnalului cuantificată în nivelul de culoare al imaginii. Analiza se face cu ajutorul transformatei Fourier [12].

Vom defini două tipuri de spectrograme: dacă sunetul se împarte în porțiuni egale, fiecare cu o lungime de aproximativ 3 milisecunde (sau dacă folosim filtre cu lărgimea de bandă de 300Hz) vom numi spectrograma de bandă largă, iar dacă sunetul se împarte în secțiuni de 20 milisecunde (sau dacă folosim filtre cu o lărgime de bandă de 45Hz) vom numi spectrograma de bandă îngustă[13].

4.2 Detecție facială

Primul pas în recunoașterea automată a emoțiilor este decuparea feței subiectului de restul contextului din imagine, astfel încât să selectăm doar informația esențială. Există două mari abordări: detecția în imaginile statice și detecția în timp real. Deși volumul de calcul este mai mare pentru detecția fețelor în timp real, aceasta este mai simplă, comparativ cu imaginile statice, deoarece persoanele sunt într-o continuă mișcare, în timp ce mediul înconjurător este static. Astfel folosind metode de eliminare a componentelor statice din

imagini, putem obține o detecție cu un grad ridicat de acuratețe. În plus, prin această metodă putem detecta fețe și în medii naturale, nu doar în laborator.

În continuare vom detalia un clasificator de tip cascadă bazat pe caracteristici Haar. *Cascadele Haar* (engl. *Haar Cascade*) reprezintă un algoritm pentru detecția obiectelor, ce poate fi folosit cu usurință în diverse scopuri precum detecția fețelor sau detecția numărului de înmatriculare a mașinilor.

Vom numi trăsătură acea valoare care se calculează ca diferența dintre suma pixelilor aflați sub dreptunghiul alb și suma pixelilor aflați sub dreptunghiul negru.

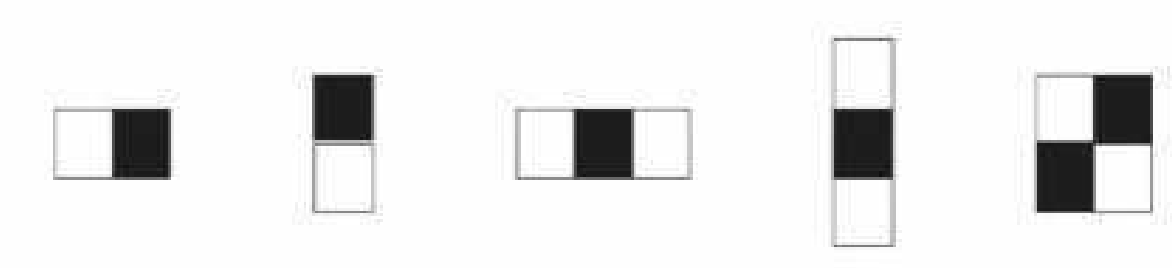

Figura 4.1: Caracteristici Haar folosite de Viola-Jones [1]

În prima etapă a algoritmului, se încarcă două seturi de poze: pozitiv și negativ. Setul pozitiv este alcătuit din imagini faciale ale mai multor persoane (cu diferite unghiuri ale feței, fundal diferit, iluminare diferită), iar setul negativ conține orice imagine ce nu seamană cu o față umană (spre exemplu casă, scaun, mașină,animale). A doua etapă constă în extragerea trăsăturilor în timp ce clasificatorul se antrenează cu imagini integrale și algoritmul Adaboost [14].

Conceptul de *imagine integrală* a fost introdus prima dată de Paul Viola și Michael Jones în anul 2001. Valoare unei imagini integrale la coordonatele (x, y) conține suma pixelilor situați la stânga și deasupra coordonatelor imaginii curente. Astfel, putem calcula o imagine integrală cu ajutorul următoarei formule:

[1] `https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html`

$$intX_{(x,y)} = \sum_{\substack{x' \leq x \\ y' \leq y}} X_{(x',y')} \tag{4.1}$$

Unde intX reprezintă imaginea integrală și X reprezintă imaginea originală. Imaginea integrală reduce volumul de operații necesare în prelucrarea imaginilor. În general, este nevoie de extragerea trăsăturilor dintr-o anumită regiune de interes, nu din toată imaginea, operație care se efectuează mult mai simplu cu ajutorul imaginii integrale.

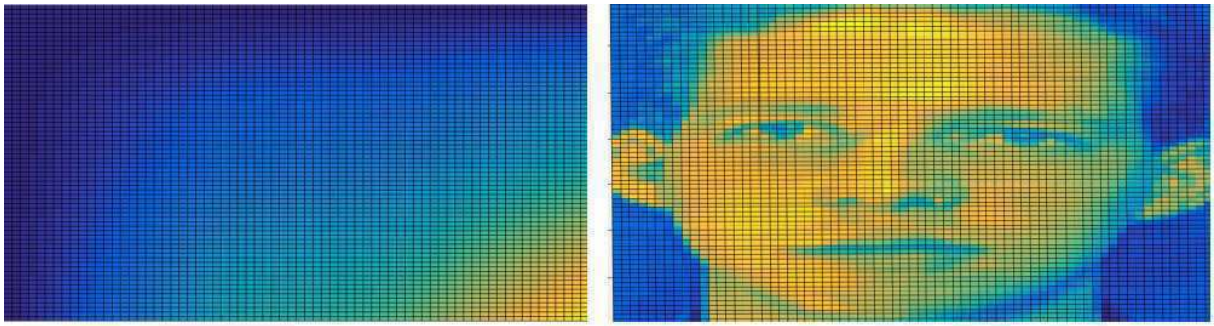

Figura 4.2: În stânga: imagine integrală, în dreapta: imagine originală [15]

4.3 Învățarea mașinilor

În domeniul informaticii, inteligența artificială este inteligența expusă de calculatoare, diferită față de inteligența naturală, particularitate a oamenilor și a unor animale. Informatica definește IA ca fiind orice dispozitiv care își percepe mediul și efectuează acțiuni care maximizează șansa de a-și atinge obiectivele [16]. De asemenea, Kaplan și Haenlein definesc inteligența artificială ca fiind "capacitatea unui sistem de a interpreta corect datele externe, de a învăța din astfel de date și de a folosi ceea ce a învățat pentru a-și atinge obiectivele și sarcinile specifice printr-o adaptare flexibilă" [17].

4.3.1 Rețele neurale adânci

Învățarea ierarhică (engl. *Deep Learning*) este o parte a învățării mașinilor (engl. *Machine Learning*) bazată pe diferite straturi utilizate în rețele neurale artificiale. Procesul de

învățare poate fi: supervizat (atunci când rețelei i se oferă datele de intrare cât și rezultatul dorit pentru fiecare exemplu în parte) sau nesupervizat (atunci când avem la dispoziție doar datele de intrare, fără a avea un rezultat dorit pentru ele).

Învățatul adânc are următoarele caracteristici:

- folosește o cascadare de straturi multiple de procesări neliniare pentru a extrage caracteristici și a efectua transformări. Fiecare strat folosește ca dată de intrare, ieșirea stratului precedent.
- se utilizează în mod supervizat și/sau nesupervizat
- învață multiple moduri de reprezentare a datelor, ce corespund diferitelor straturi de abstractizare (acest lucru se efectuează într-un mod ierarhic)

4.3.2 Rețele neurale convoluționale

Acest tip de abordare are la bază presupunerea că datele de intrare sunt imagini și că informația din ele se poate codifica și concentra prin diverse funcții. Aceste tipuri de rețele au la bază neuroni cu parametrii invatabili (ponderi și deviații), care primesc date la intrare, fac un produs scalar și aplică o funcție neliniară rezultatului. Întreaga rețea se comportă ca o funcție puternic neliniară ce face legatura între intrare (de obicei imagini) și scorurile claselor de ieșire. De asemenea, pe ultimul strat se aplică o funcție ce evaluează pierderile cu ajutorul căreia parametrii rețelei sunt actualizați, astfel având loc procesul de învățare.

Spre deosebire de o rețea neurală obișnuită, straturile de convoluție au neuroni ce sunt structurați pe trei dimenisiuni: lungime, lățime, adâncime (termenul adâncime se referă la a treia dimensiune a unui volum de neuroni, nu la numărul de straturi succesive din arhitectura rețelei).

---

[2] `http://cs231n.github.io/convolutional-networks/`

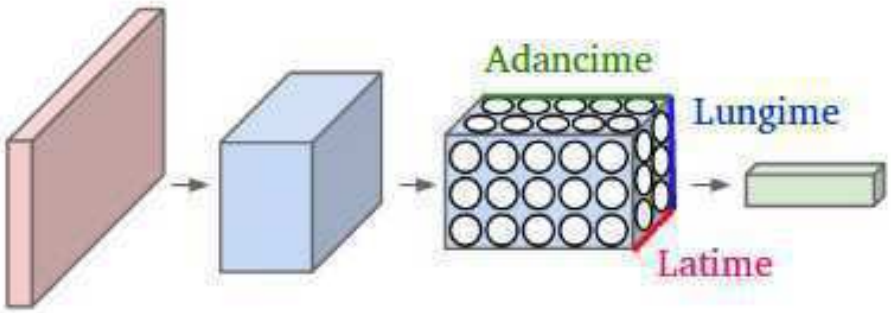

Figura 4.3: Exemplu rețea convoluțională cu neuroni pe trei dimensiuni [2]

O arhitectură simplă de rețea convoluțională cuprinde trei tipuri de straturi: convoluționale, straturi de extragere și straturi complet conectate. Acestea puse în cascadă vor transforma datele de intrare (presupuse imagini) într- o distribuție de probabilități ce cuprinde diverse clase. Se evidențiază că anumite straturi nu conțin parametrii invatabili, ci aplică o funcție prestabilită: straturile de extragere și de activare.

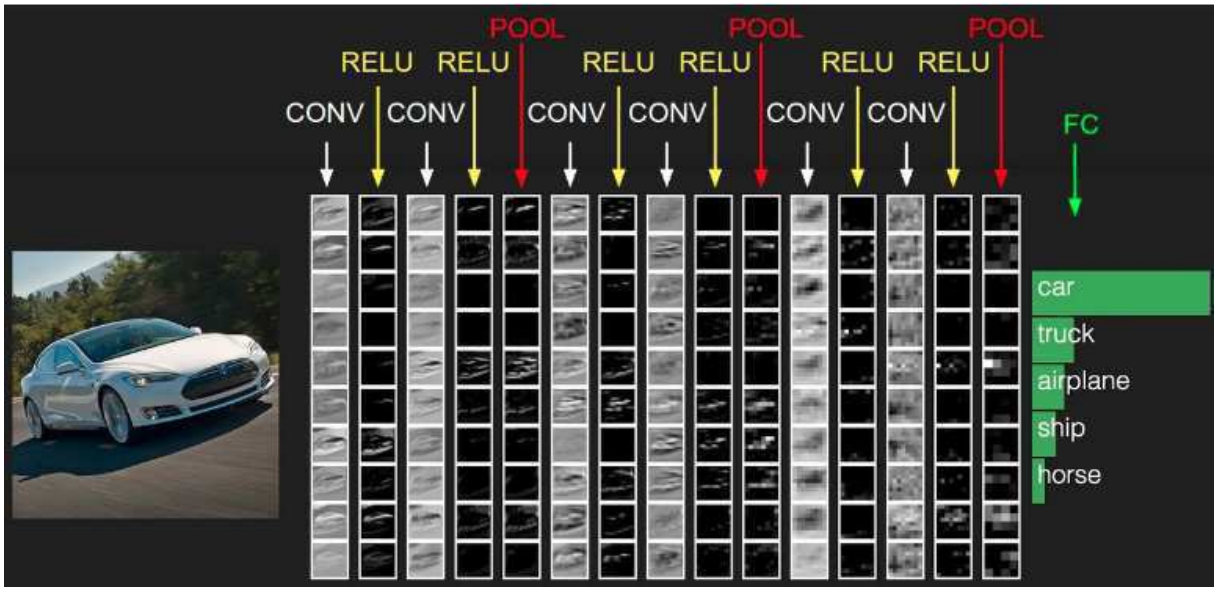

Figura 4.4: Nucleele unei rețele convoluționale antrenate. Volumul inițial de date, sub forma unei imagini cu trei plane, este procesat cu ajutorul rețelei. Se pot observa filtre din ce în ce mai abstracte, odată ce înaintăm în adâncimea rețelei. [3]

Straturile de convoluție sunt elementele definitorii ale acestui tip de rețea, care au rolul de a face cea mai mare parte din procesarea informației. Aceste straturi sunt constituite din filtre cu parametrii invatabili. Fiecare filtru are o dimensiune spațială mică (în lungime și lățime), dar se extinde pe întreg volumul datelor de intrare.

Procesul de obținere al volumului de ieșire este următorul: centrăm filtrele de convoluție pe un pixel din imaginea de intrare, calculăm produsul scalar al parametrilor filtrului cu valorile pixelilor peste care se suprapune și astfel se obține rezultatul corespunzător acelui pixel. Procesul se repetă secvențial pentru toți pixelii din imaginea de intrare. Intuitiv, fiecare nucleu va învăța parametrii, cu ajutorul cărora se va activa atunci când în imagine există un anumit tip de informație, spre exemplu, contururi sau anumite orientări. Astfel, aceste nuclee vor produce un volum de ieșire ce va conține informații mai abstracte.

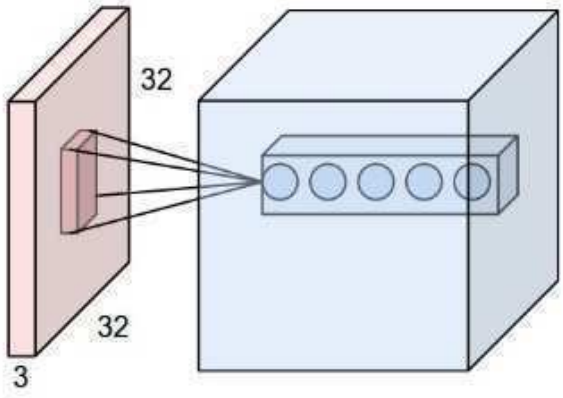

Figura 4.5: Exemplu de convoluție pentru o imagine cu trei canale de intrare.

În continuare vom detalia legătura dintre dimensiunea de intrare și cea de ieșire. Trei hiperparametrii influențează dimensiunea volumului de ieșire: numărul de nuclee, deplasamentul și bordarea cu zero. Adâncimea datelor la ieșire este egală cu **numărul de nuclee** al stratului de convoluție. Fiecare nucleu va învăța lucruri diferite față de celelalte, spre exemplu dacă primul filtru va reacționa la orientări verticale, al doilea poate învăța să reacționeze la cele verticale. **Deplasamentul** se referă la numărul de pixeli cu care este mutat centrul unui nucleu. Dacă acesta este 1, nucleul va fi centrat în fiecare pixel al imaginii, dar dacă acesta este 2 sau 3, nucleul va fi mutat cu 2 respectiv 3 pixeli, ceea ce va produce un volum de ieșire cu dimensiune mai mică. **Bordarea cu zero** este un alt hiperparametru care specifică numărul de rame cu pixeli zero puse imaginii. Acest lucru poate fi folositor atunci când ne dorim să păstrăm dimensiunea imaginii de intrare după convoluții.

De obicei între straturi succesive de convoluție se inserează un **strat de extracție**. Funcția acestuia este de a reduce dimensiunile datelor și astfel de a micșora numărul

de parametrii invatabili necesari, dar și de a extrage și concentra cele mai importante caracteristici. Acest tip de strat este independent de adâncimea datelor de intrare și efectuează operații la nivelul fiecărei hărți. Din punct de vedere al dimensionalității:

- Are nevoie de doi parametrii: dimensiunea nucleului **K** și deplasamentul **S**
- Primește la intrare date cu dimensiunea (lățime, lungime, adâncime)

$$W_1 \times H_1 \times D_1 \tag{4.2}$$

- Generează un volum cu dimensiunea

$$W_2 \times H_2 \times D_2 \tag{4.3}$$

$$W_2 = \frac{W_1 - K}{S} + 1 \tag{4.4}$$

$$H_2 = \frac{H_1 - K}{S} + 1 \tag{4.5}$$

$$D_2 = D_1 \tag{4.6}$$

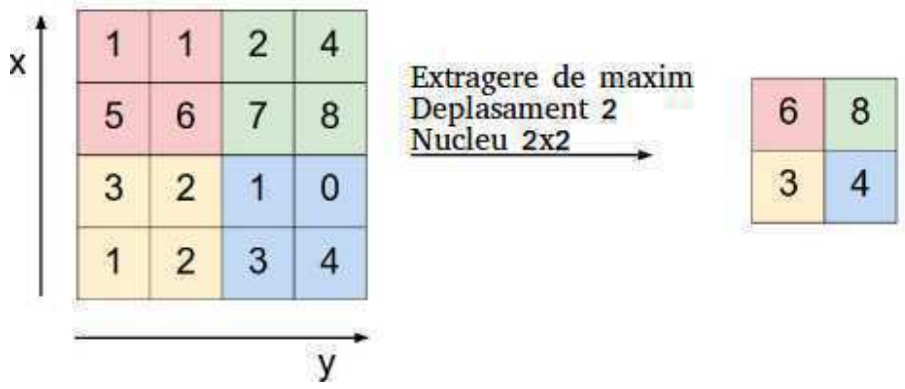

Figura 4.6: Exemplu de extragere de maxim pentru K=2, S=2.

Straturile **complet conectate** sunt poziționate în partea de final a rețelei, deoarece au un număr ridicat de parametrii invatabili, ceea ce înseamnă că este de preferat să avem un număr limitat de date la intrarea lor, pentru a fi eficiente din punct de vedere computațional. Caracteristica lor principală este că au conexiuni cu toți neuronii din

stratul anterior. Acestea pot fi văzute ca o multiplicare de matrice, urmată de o adunare a unei medii.

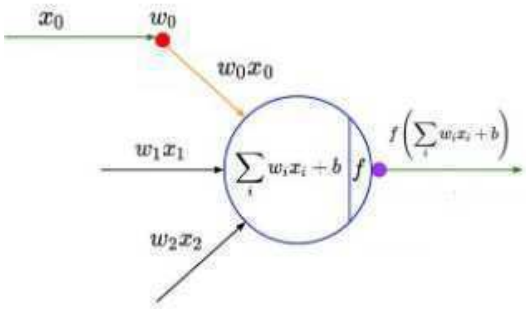

Figura 4.7: Exemplu de neuron din strat complet conectat. X_i sunt datele din stratul anterior, iar f este funcția de activare.

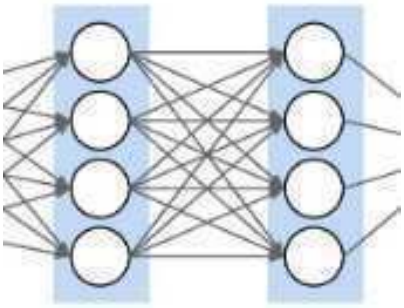

Figura 4.8: Exemplu de interconectare două straturi complet conectate.

Funcțiile de activare cele mai des folosite sunt: rectificare liniară unitară (ReLU engl. *Rectified Linear Unit*) și rectificare liniară unitară parametrică (PReLU engl. *Parametric Rectified Linear Unit*) [18]. Acestea imprimă un caracter puternic neliniar unei rețele, astfel încât aceasta să fie capabilă să realizeze funcții cât mai complexe. Funcțiile matematice sunt următoarele:

$$ReLU : f(x) = max(0, x) \tag{4.7}$$

$$PReLU : f(x) = \begin{cases} \text{x}, & \text{if x} \geq 0, \\ \alpha \cdot \text{x}, & \text{if x} < 0 \end{cases}, \alpha \in R \tag{4.8}$$

4.3.3 Funcții de cost

În domeniul optimizărilor matematice și al teoriei deciziilor, funcția de cost (sau de pierderi) face o asociere între un eveniment de intrare sau valorile mai multor variabile și un număr real ce reprezintă, în mod intuitiv, un "cost" asociat acelui eveniment. O problemă de optimizare presupune minimizarea funcției de cost, în raport cu o mulțime finită de parametrii. În doomeniul statisticii, o funcție de cost se folosește, de obicei, pentru estimarea parametrilor, prin diferența dintre valoarea estimată și cea reală provenită din date.

Intuitiv, funcția de pierderi ajută la evaluarea unui algoritm care modelează anumite date. Dacă predicțiile deviază foarte mult de la rezultatele marcate, valoarea funcției va fi un număr mare. Pe de altă parte, atunci când algoritmul modelează în mod corect datele, funcția va returna o valoare mică, ceea ce sugerează că algoritmul converge.

$$\lim_{x \to\ x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{4.9}$$

Cea mai importantă proprietate a funcției de pierderi este *derivabilitatea*. Aceasta trebuie să fie derivabilă, deoarece pentru a actualiza parametrii modelului, funcția va fi derivată parțial în raport cu fiecare parametru. Pentru ca funcția să fie derivabilă valoarea limitei din relația (9) trebuie să existe și să fie finită.

În mod general, funcțiile de cost pot fi împărțite în două categorii mari, depinzând de natura algoritmului de învățare: funcții de cost pentru *regresii* și pentru *clasificare*. În clasificare se încearcă prezicerea unei categorii dintr-o mulțime finită de posibilități (cu alte cuvinte rezultatul funcției este discret). Pe de altă parte, în sarcinile de regresie se încearcă prezicerea unei valori pe un domeniu continuu.

În continuare vom prezenta exemple de funcții de cost pentru ambele sarcini:

- Abaterea valorii absolute

$$MAE = \sum_{i=1}^{n} \frac{|y_i - \hat{y}|}{n} \tag{4.10}$$

  Se folosește în sarcini de regresie și măsoară media abaterilor absolute ale valorilor prezise față de cele observate. Totuși, această funcție nu ține cont de semnul rezultatului, ceea ce înseamnă că este invarianta cu sensul deplasării erorii.

- Abaterea patratică medie

$$MSE = \sum_{i=1}^{n} \frac{(y_i - \hat{y})^2}{n} \tag{4.11}$$

  Se folosește, de obicei, în sarcini de regresie și măsoară media diferențelor pătratice între valorile observate și cele prezise. La fel ca în cazul MAE, această funcție nu ține cont de sensul deplasării erorii. În plus, valorile ce se indepărtează mult de cele

observate, vor fi penalizate mai tare decât cele care sunt relativ aproape.

- Funcția de pierderi bazată pe entropie

$$CrossEntropyLoss = -(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{4.12}$$

Formula (12) este dată pentru cazul unei clasificări binare. Funcția de pierderi se folosește în sarcini de clasificare și se bazează pe entropia valorilor prezise și observate. Se evidențiază faptul că această funcție penalizează dur rezultatele care au o precizie mare, dar sunt greșite.

4.3.4 Augmentarea datelor

Un aspect foarte important în domeniul învățării mașinilor este setul de date. În realitate, există o mare problemă deoarece datele nu sunt suficiente pentru a obține rezultate bune și mai mult decât atât, costurile de producție sunt foarte ridicate.

Atunci când antrenezi un model, ceea ce se întâmplă practic este modificarea parametrilor invatabili ai modelului, astfel încât să realizeze cât mai bine o sarcină particulară (spre exemplu să clasifice o imagine cu clasa din care face parte). Scopul optimizării este acela de a minimiza funcția de cost, ceea ce se întâmplă atunci când toți parametrii au *valorile potrivite*. În mod natural, atunci când modelul are un număr mare de parametrii este nevoie de un set de date cu exemple proporționale cu numărul de parametrii, pentru a obține o performanță bună. De asemenea, numărul de parametrii invatabili trebuie ales ținând cont de complexitatea sarcinii pe care ne dorim să o rezolvăm.

Soluția cea mai puțin costisitoare a acestei probleme este augmentarea datelor. Asta presupune alterarea minoră a datelor de intrare cu tehnici precum rotații sau translații.

O rețea neurală convoluțională care poate clasifica obiecte chiar dacă au diferite orientări are proprietatea de *invarianta*. Ceea ce înseamnă că, rețeaua este invariantă la tranzlații, punct de vedere diferit, mărime sau iluminare sau o combinare liniară a lor [19]. Această

afirmație stă la baza tehnicii de augmentare a datelor. În realitate seturile de imagini au o variantă mică a scenariilor de achiziție. Ceea ce ne dorim este să adaugăm o mai mare varietate de condiții prin modificare sintetică a datelor.

În continuare vom enumera câteva tehnici clasice de augmentare:

- Oglindire
- Rotație
- Scalare
- Decupare
- Translație
- Adăugare zgomot gaussian alb

Prin tehnicile prezentate mai sus putem mări setul de date cu orice procent dorit de noi (x2, x5, chiar și x30) ceea ce ajută, în mod evident, la obținerea unor rezultate mai bune.

4.3.5 Metode de regularizare

În secțiunea anterioară a fost prezentată o problemă frecventă în antrenarea rețelelor neurale adânci, lipsa datelor. Însă, uneori chiar și cu metodele de augmentare nu putem înlătura această problemă. Literatura de specialitate oferă o altă soluție, *regularizarea*. Aceasta se definește ca o funcție aplicată anumitor straturi ale rețelei care face anumite ponderi zero, după o anumită regulă. În mod intuitiv, aceste metode descurajează rețeaua să învețe lucruri extrem de complexe, evitând astfel fenomenul de supraînvățare.

Dropout-ul este o tehnică gândită să ajute la rezolvarea acestei probleme. Ideea de bază este de a anula în mod aleator neuroni din rețea, împreună cu legăturile ce converg către neuron, în timpul antrenării. Prin termenul anulare ne referim la scoaterea din rețea a acelui neuron, împreună cu conexiunile sale. Cel mai simplu mod de a face asta este

să atribuim neuronilor o probabilitate de anulare p și una de existență *1-p*, iar după o anumită perioadă să se evalueze aceste probabilități și neuronii să fie anulați. [20].

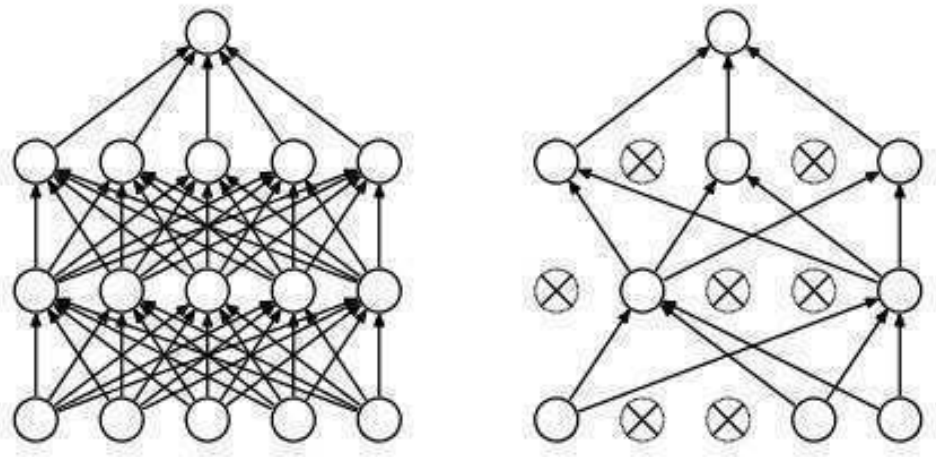

Figura 4.9: Exemplu de aplicare a funcției de dropout. Stânga: înainte de aplicare, dreapta: după aplicare.

O altă tehnică de regularizare este penalizarea ponderilor rețelei (engl. *Weigth decay regularization*). Această metodă oferă rezultate pentru cazurile de supraînvățare, dar este dovedit că uneori ajută la obținerea unei acurateți mai bune pentru rețea [21]. În procesul de învățare, un termen de regularizare este adăugat funcției de pierdere, termen ce este proporțional cu norma (de obicei L1 sau L2) ponderilor rețelei. Acest lucru este folositor atunci când se fac derivatele parțiale în raport cu ponderile, deoarece penalizează ponderile mari, iar pe cele mici le lasă libere pentru a avea variații și implicit a învăța, ceea ce ajută la o mai bună generalizare a datelor de intrare.

Intuitiv, această metodă de regularizare încurajează ponderile rețelei să fie mici, iar pe cele mari le micșorează. Ne dorim acest lucru deoarece cu cât ponderile unei rețele sunt mai mari cu atât instabilitatea rețelei crește, ceea ce sugerează că o modificare mică a datelor de intrare (spre exemplu zgomot) are un efect major în datele de ieșire.

Capitolul 5

Arhitecturi de rețele neurale

În această secțiune vor fi prezentate o serie de arhitecturi pentru rețele neurale adânci. O parte dintre ele sunt existente deja în literatură și vor fi prezentate elementele de noutate aduse de acestea, însă unele dintre ele sunt dezvoltate în cadrul proiectului de licență și sunt adaptate pentru a rezolva problema clasificării emoțiilor. Codurile pentru aceste rețele sunt listate in Anexa 3.

5.1 Arhitecturi existente în literatură

5.1.1 ResNet

Corpul de cercetare Microsoft a propus o arhitectură care își dorește să rezolve problemele existente în antrenarea rețelelor adânci. Elementul inovator este că în straturile de antrenare au adăugat o cale *reziduală* prin care informația de intrare să circule nealterată către ieșire.

Prin acest element arhitectural, cercetătorii de la Microsoft au demonstrat că rețelele sunt mult mai ușor de optimizat și au acuratețe crescută atunci când adâncimea rețelei crește, comparativ cu cele obișnuite, fără latura reziduală. În plus, afirmă că această schimbare poate reduce fenomenul de dispariție sau explozie a gradienților [22].

Cele mai bune rezultate sunt obținute în domeniul recunoașterii imaginilor, unde adâncimea rețelei crește considerabil acuratețea. Această arhitectură a câștigat locul 1 la "Competiția de recunoaștere vizuală la o scară largă ImageNet" (engl.) *ILSVRC*) 2015 pentru sarcina

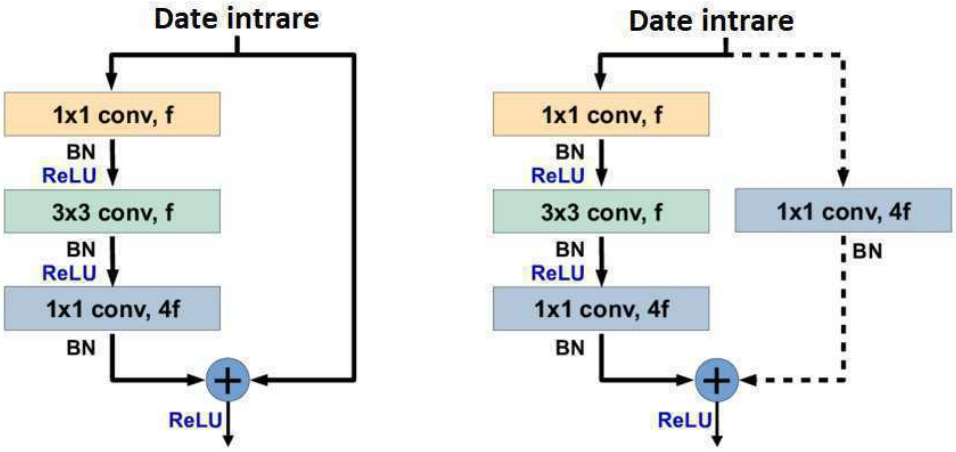

Figura 5.1: Arhitectura bloc rezidual. BN = Normalizare date intrare, ReLU = Funcție de activare (engl. *Rectified linear unit*)

de clasificare.

5.1.2 InceptionV4

Această arhitectură este inspirată din *GoogLeNet* (de asemenea cunoscută sub numele de *Inception*) și își propune să aducă îmbunătățiri, în ceea ce privește rezultatele pentru clasificarea de imagini, față de arhitecturile precedente.

Wei Liu și colegii săi de la Google afirmă că cea mai importantă caracteristică a arhitecturii Inception este aceea că în interiorul rețelei resursele computaționale sunt folosite într-un mod optimizat [23]. Blocurile Inception permit arhitecturi mai adânci și mai ramificate, menținând complexitatea computațională constantă.

InceptionV4 este o variantă îmbunătățită a arhitecturilor Inception precedente prin faptul că sunt folosite trei blocuri diferite Inception (care păstrează dimensiunea datelor de intrare) și de asemenea sunt folosite blocuri de reducere (engl. *Reduction*, care micșorează dimensiunea datelor de intrare).

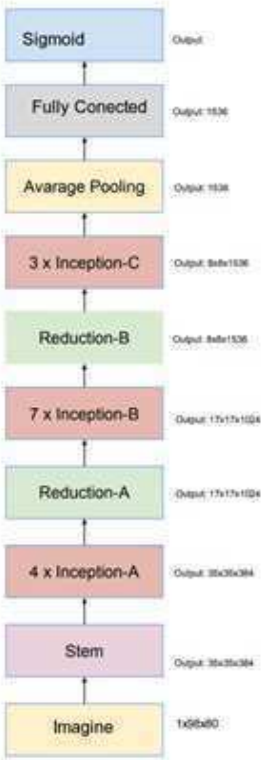

Figura 5.2: Arhitectura rețelei InceptionV4

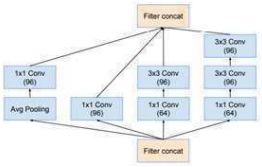

Figura 5.3: Arhitectura blocului InceptionA

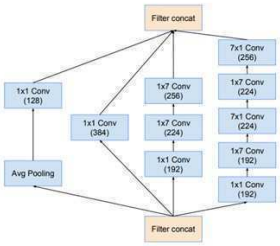

Figura 5.4: Arhitectura blocului InceptionB

5.2 Arhitecturi dezvoltate în cadrul proiectului

5.2.1 Arhitectura 1 (AR1)

Această arhitectură este o rețea convoluțională simplă, cu ajutorul căreia se poate obține un rezultat de referință. Este compusă din elemente clasice: straturi de convoluție, extracție de maxim, funcții de activare, straturi complet conectate și o funcție *Sigmoid* pentru a avea rezultate în intervalul [0, 1].

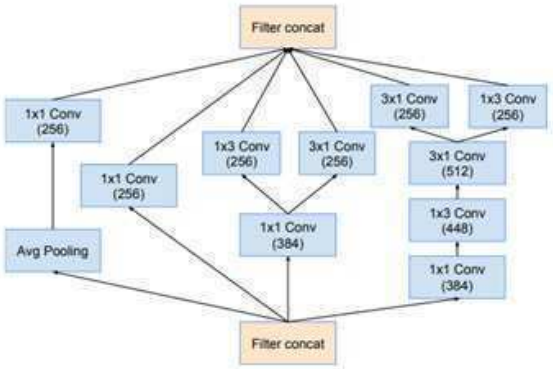

Figura 5.5: Arhitectura blocului InceptionC

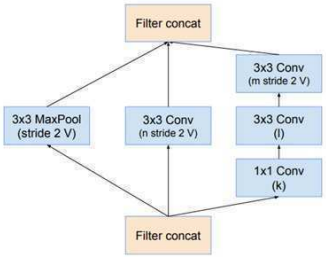

Figura 5.6: Arhitectura blocului ReductionA

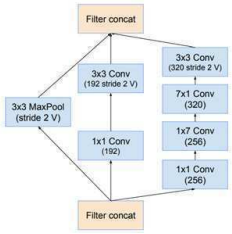

Figura 5.7: Arhitectura blocului ReductionB

$$Sigmoid(x) = \frac{e^x}{1 + e^x} \tag{5.1}$$

5.2.2 Arhitectura 2 (AR2)

Această arhitectură este o rețea convolutională, însă este de remarcat adâncimea rețelei (numărul de straturi consecutive) și adâncimea fiecărui strat din interiorul rețelei (numărul de canale). Acestea au dimensiuni mici, ceea ce înseamană că această rețea este potrivită pentru seturi de date restrânse, cu o variantă mică, în care problema suprânvățării apare frecvent.

Principiul de bază este asemănator cu cel al AR1, iar ultimului strat din rețea îi este aplicat de asemenea o funcție sigmoid.

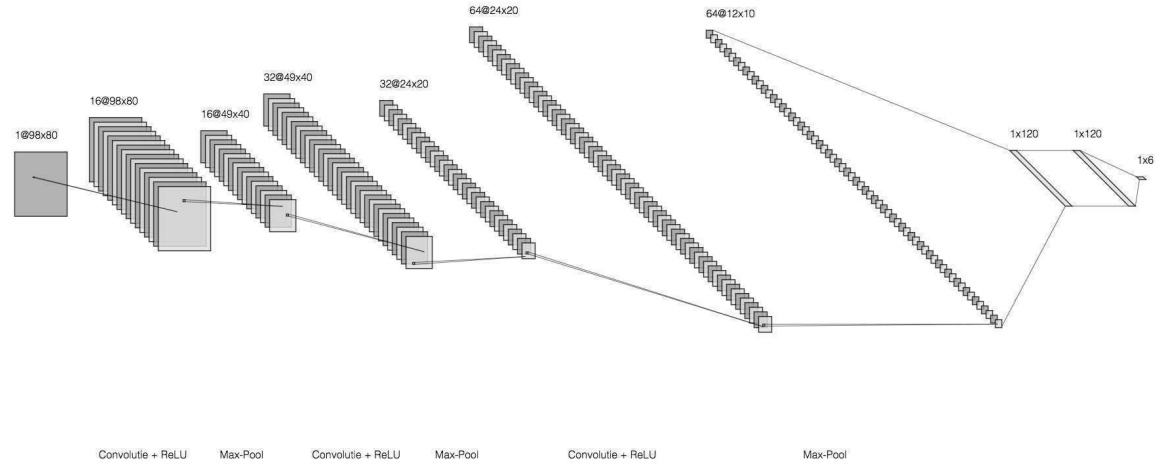

Figura 5.8: Arhitectura rețelei AR1

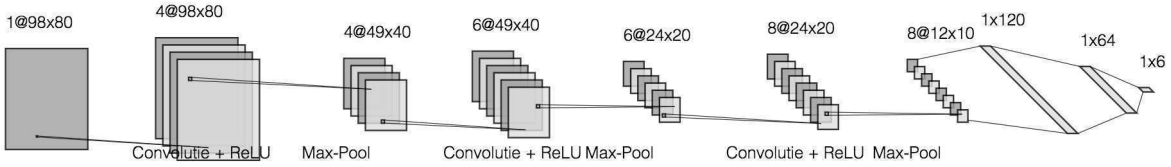

Figura 5.9: Arhitectura rețelei AR2

5.2.3 Arhitectura 3 (AR3)

Această arhitectură se bazează pe convoluții tridimensionale, în care fiecare nucleu este un volum de neuroni. Datele de intrare pentru această rețea trebuie să aibe 4 dimensiuni (adâncime × canale × latime × lungime).

Arhitectura este potrivită pentru date de intrare de tip secvențe temporale, în care trebuie ținut cont atât de conținutul unei imagini, cât și de relațiile dintre imaginile inserate în rețea.

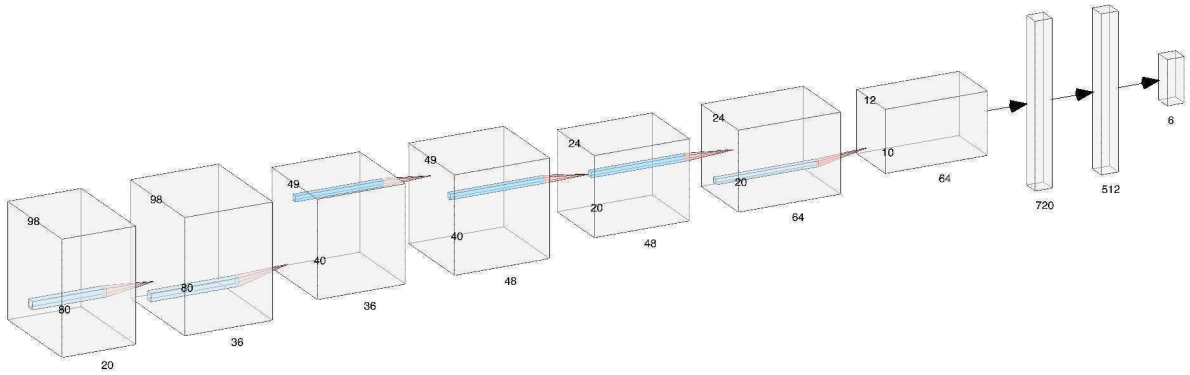

Figura 5.10: Arhitectura rețelei AR3

5.2.4 Arhitectura 4 (AR4)

Această arhitectură este compusă din două părți separate: extractor de particularități și clasificator.

Extractorul de particularități este o rețea similară cu AR1, dar care are un singur strat complet conectat la sfârșitul rețelei și căruia nu i se aplică nicio funcție. Valorile obținute se doresc a fi particularități foarte abstracte care sintetizează foarte bine informația din datele de intrare.

Clasificatorul este un perceptron cu un strat ascuns. S-a ales această arhitectură deoarece poate aproxima orice funcție oricât de complexă [24]. Prin urmare, dacă particularitățile extrase sunt sugestive, această rețea ar trebui să găsească conexiunile dintre ele și să facă o clasificare corespunzătoare între anumite date de intrare și clasele din care fac parte.

5.2.5 Arhitectura 5 (AR5)

Arhitectura a fost concepută pentru sarcini multimodale, precum recunoașterea emoțiilor din date video, care includ atât sunet cât și secvențe de imagini. Intrarea rețelei este compusă din două ramuri separate, în care trebuie inserate date în paralel. Odată cu inferența în rețea, caracteristicile abstracte de pe un anumit strat se concatenează și vor fi folosite ca date de intrare pentru un strat complet conectat.

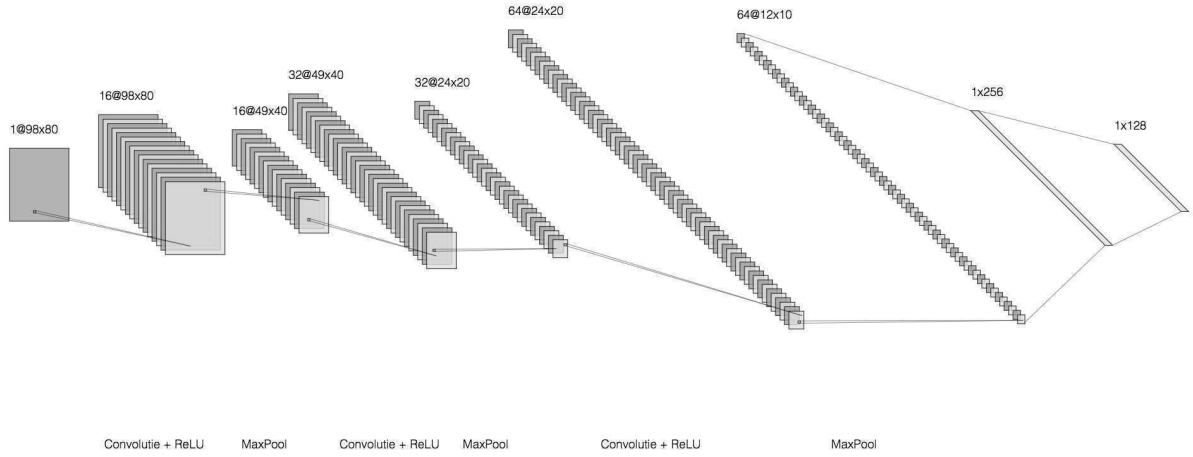

Figura 5.11: Arhitectura rețelei de extragere a particularităților AR4

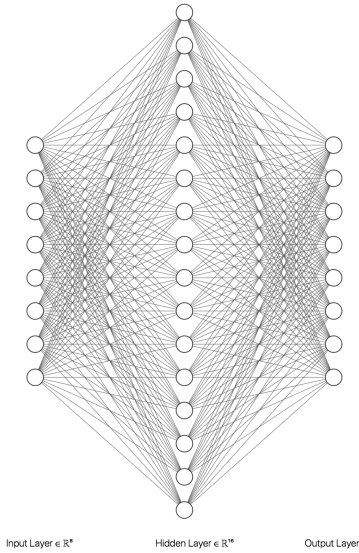

Figura 5.12: Arhitectura rețelei de clasificare AR4

Această arhitectură a fost concepută pentru a putea extrage particularități din două clase diferite de date, iar mai apoi rețeaua să clasifice în funcție de toate caracteristicile.

De asemenea, trebuie remarcat că prin lungimea vectorilor de caracteristici pentru fiecare ramură a rețelei, se poate da o importanță mai mare unei anumite laturi. Astfel, putem configura mai bine rețeaua, dacă datele ce intră pe ramuri separate au importanțe diferite.

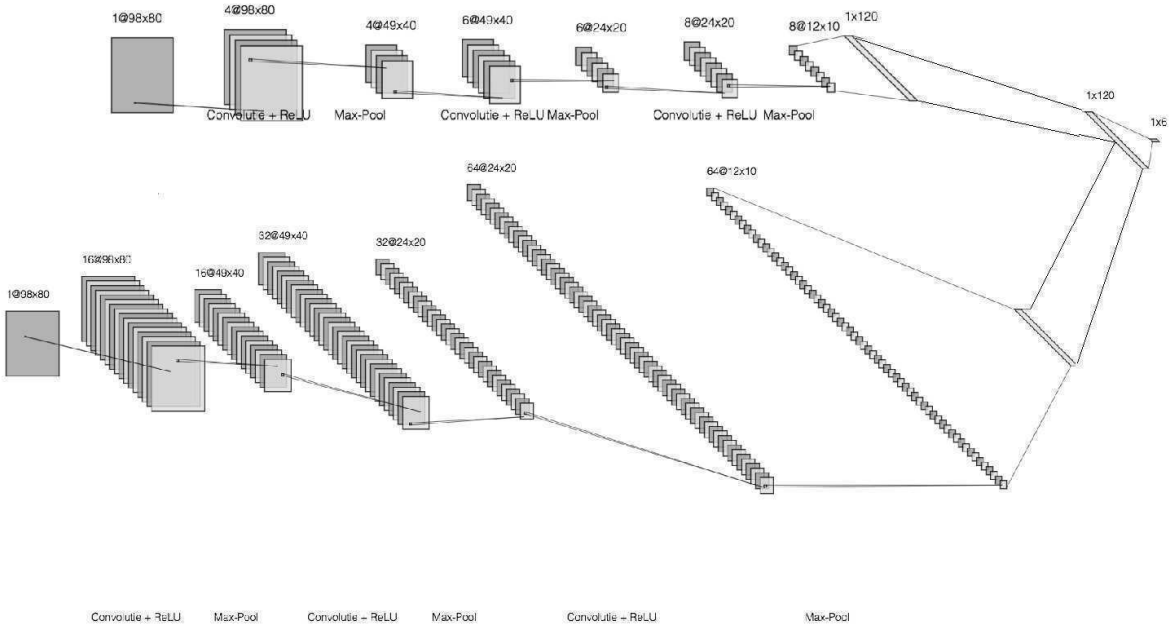

Figura 5.13: Arhitectura rețelei AR5

Capitolul 6

Experimente

În cadrul acestui proiect au fost încercate mai multe abordări, care își au fundamentele în literatura de specialitate, dar au fost aduse și elemente de noutate din diverse puncte de vedere precum: arhitectura sau modul de procesare al datelor. Provocarea a fost să se găsească un model care să dea rezultate bune în scenarii de utilizare reale (cu diferite fundaluri, iluminări sau camere), dar care să ruleze în timp real. În această secțiune vor fi descrise și explicate experimentele făcute, împreună cu rezultatele la care s-au ajuns. Codul pentru experimentele efectuate se regăsește în Anexa 1.

6.1 Experimente de recunoaștere în mod unimodal

Această parte a experimentelor se bazează pe recunoașterea emoțiilor dintr-o singură poză. Rețeaua va avea ca date de intrare o poză cu ajutorul căreia trebuie să se clasifice anumite lucruri.

În cadrul tuturor experimentelor se dorește să se recunoască atât emoția subiectului cât și mișcările faciale ale acestuia. Ceea ce presupune că vectorul de ieșire va fi format din 15 mișcări faciale (acestea au fost definite și adnotate în seturile de date) și 7 emoții (fericire, tristețe, furie, neutru, dezgust, surprindere, frică). Pentru fiecare element descris mai sus se va prezice un număr în intervalul [0, 1], care reprezintă probabilitatea de existentă a elementului în imagine.

Modul de lucru este următorul:

- Se primește o poză cu un subiect oarecare
- Se face o conversie a pozei în formatul alb-negru (engl. *gray scale*)
- Se decupează fața subiectului, se redimensionează la dimensiunea 98x80 pixeli
- Se normalizează cu ajutorul formulei (f - imagine inițială, g - imagine normalizată):

$$g(x, y) = \frac{f(x, y) - 127}{128} \tag{6.1}$$

- Se inserează în rețea
- Se procesează și rețeaua prezice mișcările faciale și emoțiile subiectului

Prima parte a modului de lucru se realizeaza cu codul listat în Anexa 2.

6.1.1 Experimentul 1

În cadrul acestui experiment s-a utilizat o arhitectură de rețea consacrată, ResNet, deoarece s-a dorit să fie un experiment stabil care să ofere rezultate concludente. În plus, datorită rezultatelor remarcabile în sarcini de clasificare a imaginilor, ne așteptăm ca rezultatele să fie comparabile cu cele din literatură.

În cadrul acestui experiment s-a variat o gamă largă de parametrii precum: rata de învățare (în intervalul $10^{-3} \div 10^{-6}$), optimizatorul sau numărul de blocuri reziduale ale rețelei.

Antrenând o arhitectură ResNet18 timp de 100 de epoci s-a ajuns la următoarele rezultate experimentale.

Se poate observa o scădere continuă a funcției de pierdere atât pe datele de antrenare cât și pe cele de testare, ceea ce evidențiază că modelul ales nu a suferit de fenomenul de supraînvățare. Mai mult decât atât, cele două valori ale pierderilor sunt apropiate, ceea ce semnifică o învățare eficientă a rețelei. Pe de altă parte, după o anumită epocă

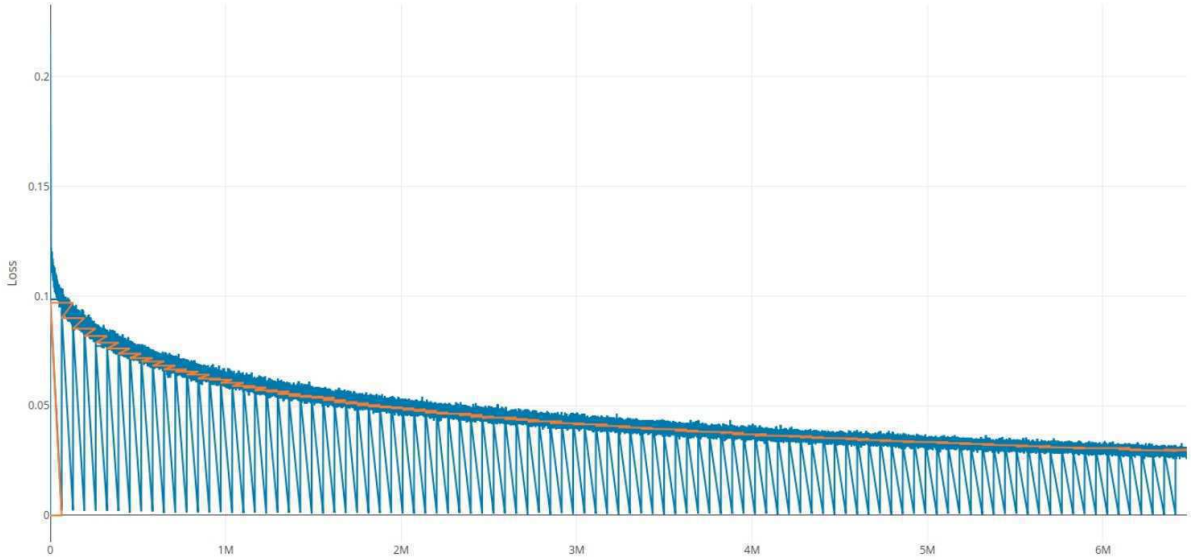

Figura 6.1: Graficul funcției de cost pentru experimentul 1

se constată o saturare a valorii funcției de cost, ce se traduce printr- o incapacitate de a învăța mai mult din datele existente.

Tabela 6.1: Rezultate clasificare emoții obținute pe seturile de date

Set de date	FN1	FN2	FN3	FN4
E1	75.40%	64.79%	58.13%	65.74%
E2	88.59%	65.12%	61.47%	74.92%

Experimentul *E1* presupune predicția tuturor celor 7 emoții, inclusiv neutru, pe când experimentul *E2* presupune predicția emoțiilor fară neutru. Vom prezice neutru atunci când scorurile celorlalte emoții sunt foarte mici. Putem observa că această schimbare a adus o îmunătățire considerabilă pe fiecare set de date. De asemenea, în tabelul 3 putem observa acuratețea fiecărei mișcări faciale în parte (mișcarea facială este codată conform [2]).

Tabela 6.2: Rezultate detecție mișcări faciale

E1	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	82.93%	75.59%	75.68%	87.04%	85.70%	79.08%	99.98%	70.48%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	81.60%	75.37%	77.50%	82.59%	65.14%	90.81%	86.77%	
E2	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	83.86%	75.18%	76.35%	84.80%	84.81%	78.34%	63.77%	77.38%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	69.21%	76.20%	79.11%	58.76%	88.67%	85.59%	87.09%	

În concluzie, rezultatele rețelei pe seturile de date interne FotoNation sunt comparabile cu cele mai bune obținute de ei, însă mai mici în medie cu 4%. De asemenea, s-a observat că o rețea ResNet cu peste 50 de blocuri reziduale nu oferă rezultate în timp real și introduce o latență, mică dar existentă.

6.1.2 Experimentul 2

În acest experiment s-a încercat o altă rețea consacrată din literatură, InceptionV4. Aceasta a fost aleasă, de asemenea, pentru rezultatele dovedite în literatura de specialitate în domeniul recunoașterii obiectelor.

Pentru procesul de antrenare s-au variat parametrii în mod asemănător experimentului 1. După o antrenare de 15 epoci s-au observat următoarele rezultate:

Tabela 6.3: Rezultate clasificare emoții obținute pe seturile de date

Set de date	FN1	FN2	FN3	FN4
E1	68.47%	65.58%	56.44%	52.99%
E2	71.60%	64.79%	54.31%	70.12%

Se observă că valoarea funcției de pierdere scade invers exponențial pentru setul de antrenare și testare, ceea ce ne indică o antrenare fară incidente majore. Comparabil cu experimentul 1, observăm că InceptionV4 reușește să prezică în medie cu o acuratețe mai mare cu 8% decât experimentul anterior, dar are rezultate mai slabe cu 3% în prezicerea emoțiilor. O altă deosebire importantă este timpul de antrenare, mult mai mare în cazul

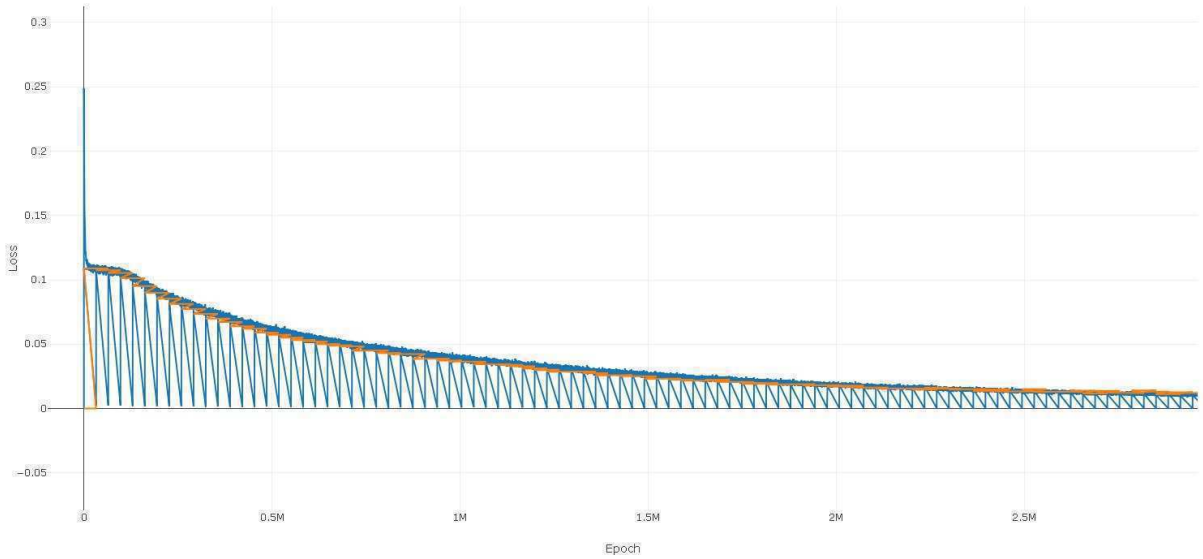

Figura 6.2: Graficul funcției de cost pentru experimentul 2

InceptionV4 (dacă pentru ResNet18 10 epoci durau în medie 16 ore, pentru InceptionV4 10 epoci de antrenare durează aproximativ 36 ore).

Tabela 6.4: Rezultate detecție mișcări faciale

E1	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	92.37%	90.52%	89.61%	94.78%	95.24%	90.07%	99.98%	90.26%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	94.26%	92.51%	93.40%	94.24%	89.30%	97.14%	96.22%	
E2	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	92.30%	89.94%	89.54%	94.42%	95.07%	89.62%	99.98%	89.92%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	94.21%	91.54%	93.28%	93.75%	88.58%	96.74%	95.65%	

În concluzie, această rețea reușește să aibă rezultate superioare în descrierea și clasificarea mișcărilor faciale, dar este mai puțin performantă în clasificarea emoțiilor față de ResNet. Această diferență poate veni din faptul că un bloc Inception este foarte ramificat, deci ar trebui să rețină trăsături cât mai diverse, însă pe parcursul înaintării în rețea, informația de emoție se pierde, deoarece rețeaua se concentrează pe trasături mai subtile.

6.1.3 Experimentul 3

Din experimentele anterioare am concluzionat că arhitectura ResNet ajută la o mai bună clasificare a emoțiilor, iar arhitectura Inception ajută la o clasificare mai bună a mișcărilor faciale. Un pas natural în dezvoltarea acestui proiect este să antrenăm o arhitectură care să se bazeze pe cele menționate. Adică o mixare a celor două, astfel încât să putem beneficia de avantajele fiecăreia. Prin urmare, am ajuns la arhitectura InceptionResnet [25].

În această arhitectură există mai multe tipuri de blocuri: *InceptionA, InceptionB, InceptionC, ReductionA, ReductionB, ReductionC*. Blocurile Inception sunt diferite din punct de vedere al ramificărilor din interior, dar toate au în comun o latură reziduală. Blocurile Reduction sunt folosite pentru a reduce dimensiunea datelor de intrare. Acest lucru este necesar pentru a micșora puterea computațională necesară unei inferențe, dar și pentru a sintetiza și concentra informațiile.

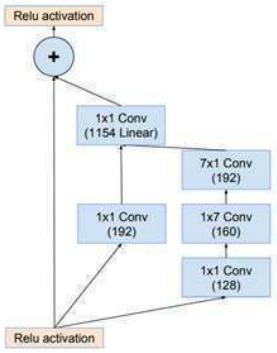

Figura 6.3: Arhitectura blocului Inception-ResnetB

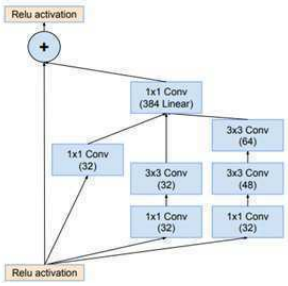

Figura 6.4: Arhitectura blocului Inception-ResnetA

În procesul de antrenare s-a observat o scădere invers exponențială pentru funcția de cost atât pe setul de date de antrenare, cât și pe cel de testare. Rezultatele au confirmat că această arhitectură combină avantajele Inception și ResNet. Acuratețea pentru detecția mișcărilor faciale a crescut, împreună cu acuratețea pentru clasificarea emoțiilor.

Tabela 6.5: Rezultate clasificare emoții obținute pe seturile de date

Set de date	FN1	FN2	FN3	FN4
E1	86.72%	63.95%	48.08%	78.26%
E2	88.19%	67.91%	59.06%	74.92%

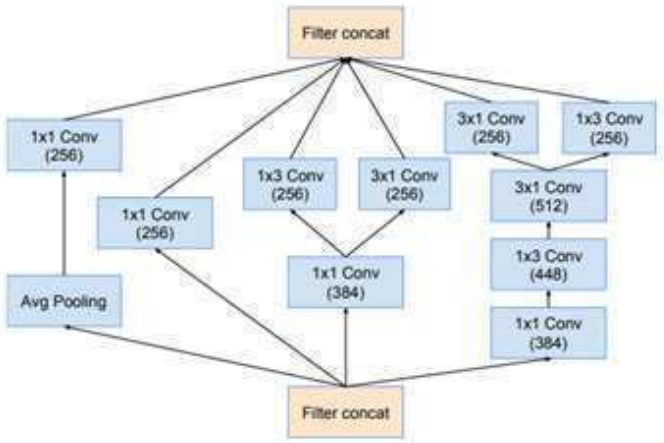

Figura 6.5: Arhitectura blocului InceptionResnetC

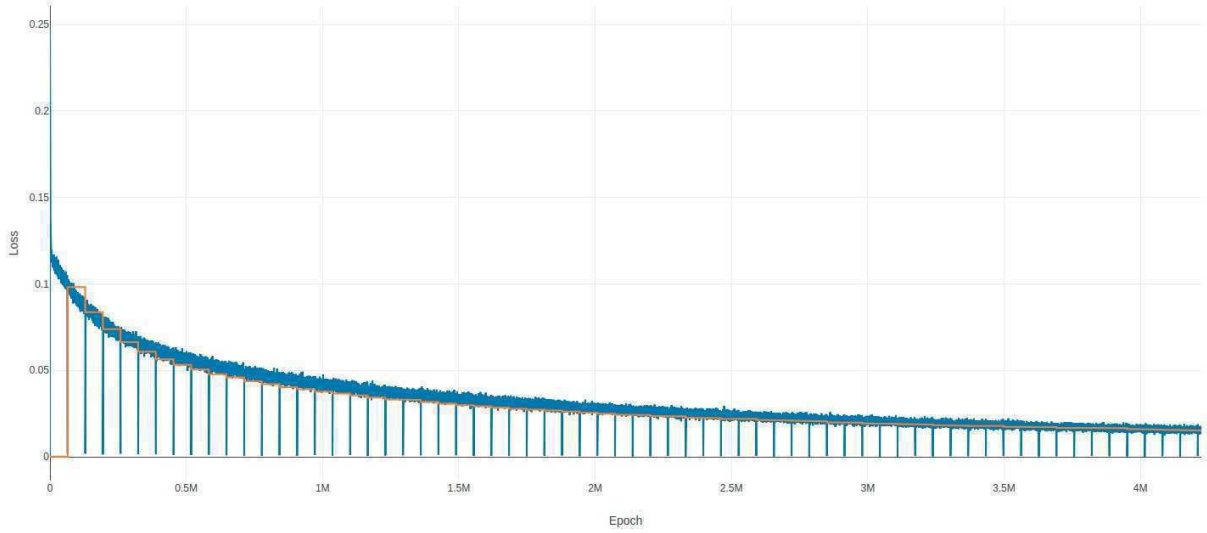

Figura 6.6: Graficul funcției de cost pentru experimentul 3

Tabela 6.6: Rezultate detecție mișcări faciale

E1	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	92.11%	92.11%	89.95%	87.65%	95.59%	93.88%	91.03%	90.35%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	93.11%	91.55%	92.99%	93.92%	88.19%	96.52%	95.66%	
E2	**AU1**	**AU2**	**AU4**	**AU5**	**AU6**	**AU7**	**AU9**	**AU10**
	89.38%	85.42%	84.54%	92.90%	91.42%	85.17%	86.24%	90.81%
	AU12	**AU15**	**AU16**	**AU17**	**AU20**	**AU23**	**AU26**	
	87.74%	89.82%	91.56%	82.77%	95.37%	93.50%	94.19%	

De asemenea, această arhitectură permite o utilizare în timp real, prin alegerea corespunzătoare a numărului de straturi utilizate.

6.1.4 Experimentul 4

În acest experiment vom schimba abordarea utilizată în primele trei. Până acum rețeaua avea ca date de ieșire un vector format din 22 de elemente (mișcările faciale concatenate cu emoțiile), adică prezicea în același timp mișcările faciale și emoțiile. Vom schimba această abordare în următorul mod: vom folosi arhitecturile propuse în primele trei experimente pentru a prezice mișcările faciale (datele de ieșire vor fi formate dintr-un vector cu 15 elemente), iar după aceea vom folosi o rețea de tip perceptron cu un singur strat ascuns pentru a prezice din acele 15 elemente emoția subiectului.

Intuitiv, vrem ca prima rețea să extragă mișcările faciale, iar a doua să învețe o logică de combinare a trăsăturilor astfel încât să prezică emoția, deoarece știm din literatură că o emoție este o mulțime finită de mișcări faciale. Rețelele vor fi antrenate secvențial și vor fi folosite la testare împreună.

Tabela 6.7: Rezultate clasificare emoții obținute pe seturile de date, prin abordarea cu două rețele separate

Set de date	FN1	FN2	FN3	FN4
E1	71.92%	62.72%	54.88%	67.33%
E2	70.33%	67.41%	55.88%	61.13%
E3	87.46%	68.33%	57.41%	76.82%

Procesele de antrenare s-au derulat fără incidente, ceea ce este indicat și din rezultatele obținute. Cea mai importantă observație este aceea că nu avem un rezultat general valabil. Asta conduce la ideea că ambele variante trebuie încercate atunci când ne dorim să obținem cele mai bune rezultate (prezicerea împreună a mișcărilor faciale și a emoțiilor, sau separat).

Rezultatele obținute trebuie comparate cu *E1* din experimentele anterioare. Observăm că pentru arhitectura ResNet prezicerea separată a obținut rezultate mai slabe, în medie

cu 2-3%, în timp ce pentru arhitectura Inception avem rezultate superioare pentru clasificarea emoțiilor cu aproximativ 4%. Pentru arhitectura InceptionResnet rezultatele sunt comparabile, ceea ce sugerează o echivalență între metode.

6.2 Experimente de recunoaștere în mod multimodal

În cadrul acestei secțiuni vom avea o abordare mai complexă a sarcinii de recunoaștere a emoțiilor. Vom încerca să determinăm emoția unui subiect din secvențe video, care cuprind atât imagini, cât și sunet.

Arhitectura folosită este AR5 (descrisă în capitolul anterior), deoarece aceasta este potrivită pentru sarcini multimodale. Se observă că această arhitectură prezintă două laturi pentru datele de intrare. Acestea vor fi folosite în felul următor: brațul mai dens va fi folosit pentru a insera secvența video, iar cel de-al doilea braț va fi folosit pentru a introduce spectrogrma sunetului din video. Astfel, cu ajutorul imaginilor, dar și al sunetului putem avea o percepție mai amplă asupra emoției.

De asemenea, în cadrul acestui experiment se va face o comparație pentru a demonstra dacă sunetul ajută sau nu la clasificarea emoțiilor. Această comparație este compusă din experimentul propriu-zis cu AR5 și un alt experiment cu o arhitectură care nu prezintă două brațe pentru datele de intrare (unde vor fi inserate doar secvențele video).

Secvențele video vor fi introduse în rețea în următorul mod: din numărul total de cadre al unui video vor fi alese 20 egal distanțate, iar acestea vor fi introduse ca număr de canale ale unei imagini. Astfel, datele de intrare vor avea următoarea dimensiune, la modul general:

$$Dim = 1 \times NC \times W \times H \tag{6.2}$$

Unde NC semnifică numărul de cadre ale secvenței, W lățimea unui cadru și H înălțimea unui cadru.În cazul nostru dimensiunile sunt următoarele: $1 \times 20 \times 98 \times 80$.

Baza de date utilizată este *CREMA-D*, iar pentru a testa acuratețea algoritmului am ales

următoarea metodă: am antrenat rețeaua pe toți subiecții în afară de unu și am testat pe acela. Astfel, rețeaua nu a avut date cu subiectul de test la antrenare și putem obține rezultate concludente.

Tabela 6.8: Rezultate clasificare emoții

Set de date	CREMA-D
E1	62.84%
E2	69.42%

Experimentul *E1* semnifică doar utilizarea secvențelor video pentru clasificare, iar în experimentul *E2* au fost folosite atât secvențele video cât și cele audio pentru clasificare.

În acest experiment, o mare problemă a fost fenomenul de supraînvățare, deoarece baza de date este relativ mică și cu variații extrem de reduse (fundalul este același, la fel și metoda și unghiurile de înregistrare). Pentru această problemă au fost încercate diferite tehnici precum: augmentarea datelor, dropout, micșorarea rețelei, constrângerea ponderilor din rețea să aparțină unui anumit interval. Cu toate acestea, problema încă apare la testarea pe diferiți subiecți. De evidențiat este faptul că, funcția de cost este *CrossEntropy*, iar aceasta penalizează mai mult atunci când rețeaua face o predicție greșită cu o acuratețe mare.

O concluzie foarte importantă ce reiese din acest experiment este aceea că, acuratețea rețelei este îmbunătățită cu 6.58% atunci când se introduce și partea de sunet. Asta semnifică faptul că o caracterizare mai bună a contextului (video și sunet) ajută rețeaua să prezică mai bine emoția subiectului.

De remarcat este faptul că autorii bazei de date afirmă că acuratețea umană este următoarea:

Tabela 6.9: Acuratețea umană pe CREMA-D

Mod	Audio	Video	Audio + Video
Acuratețe	40.9%	58.2%	63.6%

În concluzie, rezultatele obținute cu ajutorul rețelei sunt mai bune decât cele obținute de oameni, ceea ce evidențiază puterea algoritmului ales.

Capitolul 7

Aplicație

Pentru a verifica aplicabilitatea reală a modelelor obținute vom implementa o aplicație ce va ajuta la testarea algoritmilor aleși. Astfel, am ales să implementăm o aplicație în Python care va rula pe calculator, indiferent de sistemul de operare.

Aplicația va rula în felul următor: va prelua imaginile de la o cameră conectată la calculator, le va trimite către modulul de preprocesare, care va pregăti poza pentru a fi inserată în rețea, iar mai apoi poza preprocesată va intra în rețea. Rețeaua va intoarce următoarele rezultate: 15 procente pentru mișcările faciale (probabilitatea de existență pentru fiecare) și 7 procente pentru emoții.

Structura aplicației este următoarea:

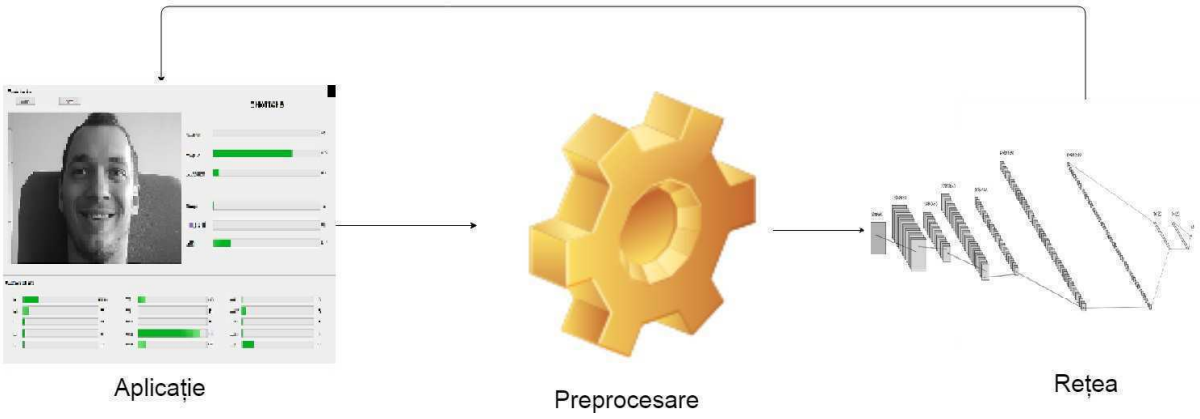

Figura 7.1: Schemă bloc aplicație

Blocul *Aplicație* face referire la interfața utilizatorului. Codul acestei aplicații este listat in Anexa 4. Aceasta afișează imaginile văzute de cameră, mișcările faciale cât și emoțiile subiectului. De asemenea, din interfață putem selecta între două moduri: mod lucru video (în care datele de intrare sunt luate de la cameră) sau mod lucru imagine (în care

o imagine este luată din memoria calculatorului și inserată).

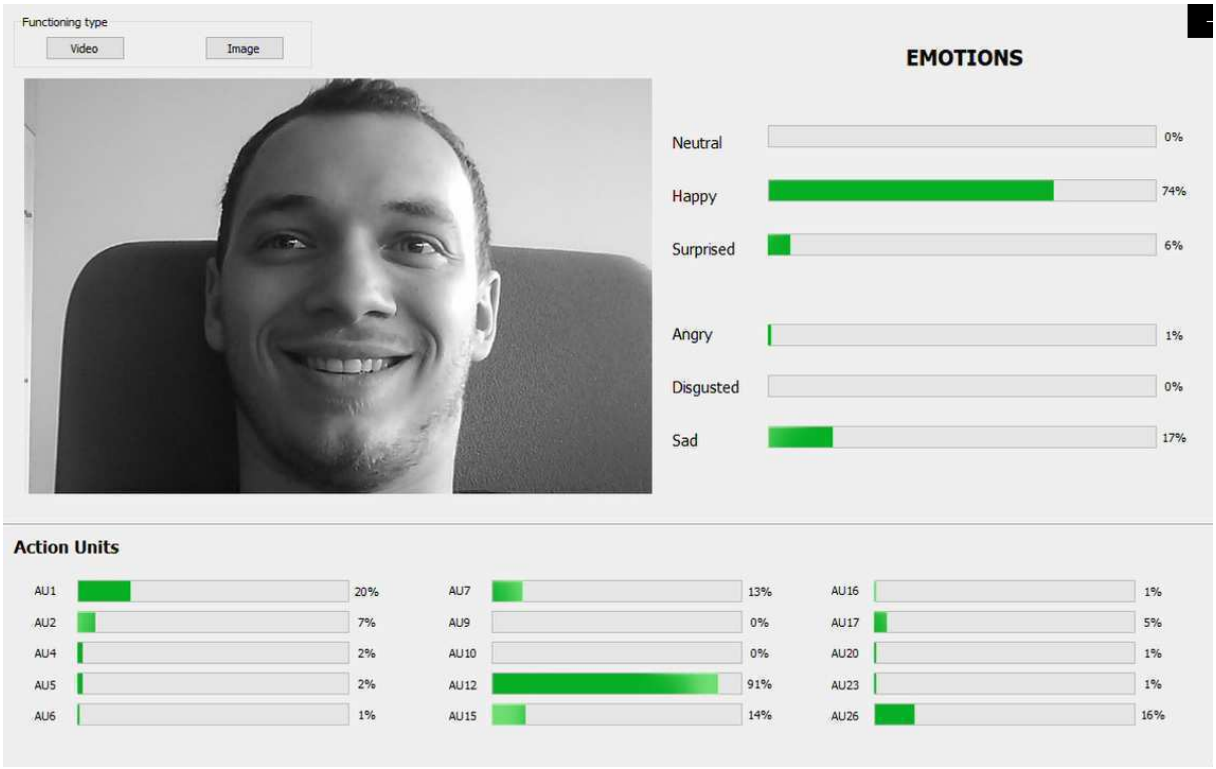

Figura 7.2: Captură foto aplicație

Blocul *Preprocesare* conține algoritmi ce se aplică pozei. În acest modul sunt incluse următoarele operații: detecție facială, decupare facială, redimensionare, egalizare de histogramă, normalizare poză.

Blocul *Rețea* este compus dintr-o clasă ce conține modelul rețelei ales și ajută la inferență cât și la preluarea și modificarea formatului rezultatelor.

Capitolul 8

Concluzii

În cadrul proiectului au fost încercate mai multe experimente, din care se pot extrage concluzii importante în ceea ce privește domeniul de clasificare al emoțiilor.

În primul rând, modul unimodal de recunoaștere al emoțiilor (doar din imagini singulare) are rezultate impresionante, cu acuratețe în medie de peste 70% pe seturi de date validare adnotate de FotoNation. Mai mult decât atât, mișcările faciale ajută la clasificarea emoțiilor într-un procent foarte important. Acest lucru a fost demonstrat în *experimentul 4*, în care o rețea simplă de tip perceptron multistrat are ca date de intrare doar prezența sau absența unor mișcări faciale și totuși reușește să obțină o acuratețe de clasificare a emoțiilor de 71.92% în cazul arhitecturii ResNet și 70.33% în cazul arhitecturii Inception (pe setul de date FN1). Ceea ce sugerează faptul că există o legatură biunivocă între mișcările faciale și emoția subiectului. În plus, prin această metodă de a extrage prima dată mișcările faciale și după emoție putem antrena rețeaua să recunoască un comportament disimulat, deoarece o persoană nu poate avea control total asupra mușchilor faciali. Spre exemplu, dacă un subiect încearcă să mimeze fericirea, acesta cu siguranță va zâmbi și își va ridica pometii, însă o altă mișcare facială specifică zâmbetului este închiderea partială a ochilor, pe care o poate omite în mod involuntar. De asemenea, există și posibilitatea de a face o mișcare facială care nu este specifică zâmbetului și astfel să ne dăm seama că este o încercare de a disimula emoția.

În al doilea rând, din experimentele unimodale am putut observa o diferență în timpul de antrenare și inferență pentru diversele arhitecturi alese. Concluzia este aceea că o arhitectură de tip ResNet este mai rapidă în antrenare și se potrivește mai bine pentru sarcini

care necesită o rulare în timp real, în comparație cu cea Inception. Diferența de timp este aproximativ cu 50% mai mare atât pentru antrenare cât și pentru inferență. Pe de altă parte, atunci când combinăm aceste arhitecturi într- una mixtă, numită InceptionResnet, observăm că timpul necesar procesării se reduce, însă nu scade sub cel al arhitecturii ResNet. Cu toate acestea, am remarcat că Inception, datorită stratificării pe orizontală, reușește să prindă caracteristicile imaginii mai bine decât ResNet (acest lucru s-a observat prin acuratețea obținută de cele două rețele pentru mișcări faciale), însă pentru sarcina globală, ResNet are rezultate mai bune.

O altă concluzie importantă a fost extrasă din experimentele de recunoaștere în mod multimodal. Se observă că atunci când utilizăm doar un video avem o acuratețe mai mică decât atunci când rețeaua primește ca date de intrare video și audio (acuratețea este cu 6.57% mai mare). Același lucru se poate observa când analizăm acuratețea umană pe acest set de date, modul multimodal îmbunătățește clasificarea cu 5.4%. Aceste observații conduc la concluzia că un context cât mai amplu este benefic atunci când dormim să clasificăm emoții. Astfel, având date diferite dar despre același subiect putem clasifica cu o acuratețe mai mare. De asemenea, având mai multe informații despre subiect putem clasifica între comportament disimulat și natural cu o mai bună certitudine.

Bibliografie

[1] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pp. 94–101, IEEE, 2010.

[2] J. F. Cohn, Z. Ambadar, and P. Ekman, "Observer-based measurement of facial expression with the facial action coding system," *The handbook of emotion elicitation and assessment*, pp. 203–221, 2007.

[3] C. Padgett and G. Cottrell, "Identifying emotion in static face images," in *Proceedings of the 2nd Joint Symposium on Neural Computation*, vol. 5, pp. 91–101, 1995.

[4] S. A. Bargal, E. Barsoum, C. C. Ferrer, and C. Zhang, "Emotion recognition in the wild from videos using images," in *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pp. 433–436, ACM, 2016.

[5] H. Jung, S. Lee, J. Yim, S. Park, and J. Kim, "Joint fine-tuning in deep neural networks for facial expression recognition," in *Proceedings of the IEEE international conference on computer vision*, pp. 2983–2991, 2015.

[6] H. Yang, U. Ciftci, and L. Yin, "Facial expression recognition by de-expression residue learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2168–2177, 2018.

[7] P. Burkert, F. Trier, M. Z. Afzal, A. Dengel, and M. Liwicki, "Dexpression: Deep convolutional neural network for expression recognition," *arXiv preprint arXiv:1509.05371*, 2015.

[8] R. Ekman, *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS)*. Oxford University Press, USA, 1997.

[9] M. Pantic, M. Valstar, R. Rademaker, and L. Maat, "Web-based database for facial expression analysis," in *2005 IEEE international conference on multimedia and Expo*, pp. 5–pp, IEEE, 2005.

[10] S. Li and W. Deng, "Deep facial expression recognition: A survey," *arXiv preprint arXiv:1804.08348*, 2018.

[11] H. Cao, D. G. Cooper, M. K. Keutmann, R. C. Gur, A. Nenkova, and R. Verma, "Crema-d: Crowd-sourced emotional multimodal actors dataset," *IEEE transactions on affective computing*, vol. 5, no. 4, pp. 377–390, 2014.

[12] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*, vol. 31999. McGraw-Hill New York, 1986.

[13] B. E. Kingsbury, N. Morgan, and S. Greenberg, "Robust speech recognition using the modulation spectrogram," *Speech communication*, vol. 25, no. 1-3, pp. 117–132, 1998.

[14] R. E. Schapire, "Explaining adaboost," in *Empirical inference*, pp. 37–52, Springer, 2013.

[15] C. B. Sorin-Valentin Geana, Ana-Maria Radoi, "Sistem minimal de autentificare faciala," pp. 19–20, 2017.

[16] G. F. Luger, *Artificial intelligence: structures and strategies for complex problem solving*. Pearson education, 2005.

[17] A. Kaplan and M. Haenlein, "Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence," *Business Horizons*, vol. 62, no. 1, pp. 15–25, 2019.

[18] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[19] M. A. Tanner and W. H. Wong, "The calculation of posterior distributions by data augmentation," *Journal of the American statistical Association*, vol. 82, no. 398, pp. 528–540, 1987.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] G. Zhang, C. Wang, B. Xu, and R. Grosse, "Three mechanisms of weight decay regularization," *arXiv preprint arXiv:1810.12281*, 2018.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[24] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 683–697, 1992.

[25] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Anexa 1

```
import config
import torch
import os
from loadAuDB import EmotionsData
import config as cfg
import numpy as np


class Trainer():
    def __init__(self, network, criterion, optimizer, visualizer, experiment_name):
        self.network = network
        self.train_dataloader = None
        self.valid_dataloader = None
        self.test_dataloader = None
        self.criterion = criterion
        self.optimizer = optimizer
        self.visualizer = visualizer
        self.experiment_name = experiment_name


    def train_epoch(self, epoch):
        running_loss = 0.0
        for idx, (face, labels, confidences) in enumerate(self.train_dataloader, 0):
            labels = labels[:,:-cfg.emotions_size]
            # confidences = confidences[:,:-cfg.emotions_size]
```

```
            if len(face) != cfg.batch_size:
                continue

            self.network.train()
            if config.use_cuda:
                face = face.cuda().float().unsqueeze(1)
                labels = labels.cuda().float()

            self.optimizer.zero_grad()
            predictions = self.network(face)

            loss = self.criterion(predictions, labels)
            loss.backward()
            self.optimizer.step()

            running_loss += loss.item()
            if idx % config.print_loss == 0:
                running_loss = running_loss / config.print_loss
                self.visualizer.update_statistics(idx + len(self.train_dataloader) *
                    epoch, running_loss, 0.0)
                running_loss = 0.0

    # The precision for every AUs is calculated:
    # the number of AUs corectly determined ( even if the label is 0 or 1) / the total
        number of labels
    def eval_net(self, epoch):
        self.network.eval()

        running_eval_loss = 0.0
```

```
        well_marked_AUs = np.zeros(cfg.action_units_size)

        for i, (face_, labels_, confidences_) in enumerate(self.valid_dataloader, 0):
            labels_ = labels_[:, :-cfg.emotions_size]
            if len(face_) != cfg.batch_size:
                continue
            if config.use_cuda:
                face_ = face_.cuda().float().unsqueeze(1)
                labels_ = labels_.cuda().float()

            predictions_ = self.network(face_)
            eval_loss = self.criterion(predictions_, labels_)
            running_eval_loss += eval_loss.item()

            well_marked_AUs = well_marked_AUs + sum(np.array(torch.lt(torch.abs(
                torch.add(labels_, -predictions_)), cfg.DELTA_AUS).cpu()))

        running_eval_loss = running_eval_loss / len(self.valid_dataloader)
        self.visualizer.update_statistics(len(self.train_dataloader) * (epoch + 1), 0,
            running_eval_loss)

        prediction_precision = well_marked_AUs / len(self.valid_dataloader.dataset)
        print('Precision for every AU: ')
        print(prediction_precision)

    def train(self, epoch):
        try:
            os.mkdir(config.path_to_save_models + self.experiment_name)
        except FileExistsError:
            print("Director already exists! It will be overwritten!")
```

```
    for i in range(0, epoch + 1):
        print('Training on epoch ' + str(i))
        self.train_epoch(i)

        if i % config.eval_net_epoch == 0:
            self.eval_net(i)

        if i % config.save_net_epochs == 0:
            self.save_net_state(i)

def save_net_state(self, epoch):
    torch.save(self.network, config.path_to_save_models + self.experiment_name
        + '/Model_epoch_' + str(epoch) + '.pkl')

def test_net(self, network):
    network.eval()
    running_eval_loss = 0.0

    for idx, (face, labels, confidences) in enumerate(self.test_dataloader, 0):
        labels = labels[:,:-cfg.emotions_size]
        if len(face) != cfg.batch_size:
            continue
        if config.use_cuda:
            face = face.cuda().float().unsqueeze(1)
            labels = labels.cuda().float()

        predictions = network(face)
        loss = self.criterion(predictions, labels)
        running_eval_loss += loss.item()
```

```
        running_eval_loss = running_eval_loss / len(self.test_dataloader)
        print('L2 loss on evaluation dataset is: ' + str(running_eval_loss))

    def load_data(self):
        # Pytorch datasets
        aus_train_dataset = EmotionsData(cfg.train_folders, cfg.dataset_params)
        aus_test_dataset = EmotionsData(cfg.test_folders, cfg.dataset_params)

        train_size = int(cfg.train_validation_ratio * aus_train_dataset.__len__())
        valid_size = aus_train_dataset.__len__() - train_size

        train_dataset, valid_dataset = torch.utils.data.dataset.random_split(
            aus_train_dataset, [train_size, valid_size])
        # Data Loader (Input Pipeline)
        self.train_dataloader = torch.utils.data.DataLoader(dataset=train_dataset,
            batch_size=cfg.batch_size, shuffle=True,num_workers=8)
        self.valid_dataloader = torch.utils.data.DataLoader(dataset=valid_dataset,
            batch_size=cfg.batch_size, shuffle=False, num_workers=8)

        self.test_loader = torch.utils.data.DataLoader(dataset=aus_test_dataset,
            batch_size=cfg.batch_size, shuffle=False, num_workers=8)

        print(train_size)
        print(valid_size)
```

Anexa 2

```
import glob
import os

import cv2
import numpy as np
import torch
from torch.utils.data.dataset import Dataset
from scipy import signal
from scipy.io import wavfile

import config as cfg



class EmotionsData(Dataset):
    def __init__(self, folder_list, dataset_params, test=False, preLoad=False):
        self.__data = []

        self.dataset_params = dataset_params
        self.isPreLoaded = preLoad
        # self.fgbg = cv2.bgsegm.createBackgroundSubtractorMOG()

        self.zero_i = 0
        self.low_i = 0
        self.medium_i = 0
```

```
        self.high_i = 0
        self.emotions_counts = np.zeros(self.dataset_params['emotions_size'] + 1)
        self.forTest = test
        print("Using DB for test = ", self.forTest)

        print("isPreLoaded = ", self.isPreLoaded)
        for (sequence, num_aug) in folder_list:
            if sequence.find('crema_eval') == -1:
                audio_path = sequence.replace('crema', 'AudioWAV') + '.wav'
            else:
                audio_path = sequence.replace('crema_eval', 'AudioWAV') + '.wav'
            augmentations = glob.glob(sequence + "/*.raw")[0:num_aug]
            for raw_file in augmentations:
                label = sequence.split('/')[-1].split('_')[-2]
                lbl = cfg.crema_emotion_dic[label]
                self.__data.append([raw_file, lbl, audio_path])

    # Override to give PyTorch access to any image on the dataset
    def __getitem__(self, index):
        if self.isPreLoaded:
            example = self.__data[index]
        else:
            raw_file = self.__data[index][0]
            images = self.get_crops(raw_file)
            spectrogram = self.get_spectrogram(self.__data[index][2])

        x_temp = images
        # x = None
        # x = np.concatenate((x_temp[0], x_temp[int(len(x_temp) / 2)], x_temp[len(
            x_temp) - 1]), axis=0)
```

```
SAMPLES_NUMBER = 15
step = int(len(x_temp) / SAMPLES_NUMBER)

max_seq = len(x_temp) % SAMPLES_NUMBER + 1

x = x_temp[0]
for i in range(1, SAMPLES_NUMBER):
    x = np.concatenate((x, x_temp[i * step]), axis=0)

x = np.expand_dims(x, axis=0)

for k in range(1, max_seq):
    xt = x_temp[k]
    for j in range(1, SAMPLES_NUMBER):
        xt = np.concatenate((xt, x_temp[k + j * step]), axis=0)
    xt = np.expand_dims(xt, axis=0)

    x = np.concatenate((x, xt), axis=0)
img = torch.from_numpy(x)

label = torch.from_numpy(np.array(self.__data[index][1]) )

spectrogram = torch.from_numpy(spectrogram)

spectrogram_batch = spectrogram.unsqueeze(0)
label_batch = label.unsqueeze(0).unsqueeze(0)
for i in range(0, len(img)-1):
    label_batch = torch.cat((label_batch, label.unsqueeze(0).unsqueeze(0)), 0)
```

```
            spectrogram_batch = torch.cat((spectrogram_batch,spectrogram.unsqueeze
                (0)), 0)

        return img, label_batch, spectrogram_batch

    # Override to give PyTorch size of dataset
    def __len__(self):
        return len(self.__data)

    def get_crops(self, raw_path):
        cropSize = self.dataset_params['crops_shape']
        numCrops = int(os.path.getsize(raw_path) / cropSize[1] / cropSize[2])

        crops = np.fromfile(raw_path, dtype=np.uint8)
        crops.shape = [numCrops, cropSize[1], cropSize[2]]

        # new_crops = add_crops_heatmaps(crops, self.mask)

        new_crops = np.array(crops, dtype=float)

        # optical_flow = self.get_OpticalFlow(new_crops)

        new_crops /= 255.0
        new_crops -= 0.5

        new_crops.shape = [numCrops, 1, cropSize[1], cropSize[2]]
        return new_crops

    def get_OpticalFlow(self, images):
        images_new = images.astype(np.uint8)
```

```
        accum_image = np.zeros(cfg.input_size, np.uint8)
        thresh = 2
        maxValue = 5
        for i in range(0, len(images_new)):
            fgmask = self.fgbg.apply(images_new[i])
            ret, th1 = cv2.threshold(fgmask, thresh, maxValue, cv2.
                THRESH_BINARY)
            accum_image = cv2.add(accum_image, th1)
        return accum_image


    def get_spectrogram(self, song_path):
        sample_rate, samples = wavfile.read(song_path)

        noise = np.random.normal(0, 1, samples.shape)
        samples = samples + noise

        frequencies, times, spectrogram = signal.spectrogram(samples, sample_rate)
        np_img = np.array(spectrogram)
        np_img = np.flipud(np_img)
        np_img = np_img / np.amax(np.absolute(np_img))

        np_img_res = cv2.resize(np_img, dsize=cfg.spectrogram_shape, interpolation=
            cv2.INTER_CUBIC)
        return np_img_res
```

Anexa 3

```
import torch
import torch.nn as nn
import config as cfg
import torch.utils.model_zoo as model_zoo
import os
import sys

__all__ = ['InceptionV4']


class BasicConv2d(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride, padding=0):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes, out_planes,
                              kernel_size=kernel_size, stride=stride,
                              padding=padding, bias=False) # verify bias false
        self.bn = nn.BatchNorm2d(out_planes,
                                 eps=0.001, # value found in tensorflow
                                 momentum=0.1, # default pytorch value
                                 affine=True)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(x)
```

```
        x = self.bn(x)
        x = self.relu(x)
        return x


class Mixed_3a(nn.Module):

    def __init__(self):
        super(Mixed_3a, self).__init__()
        self.maxpool = nn.MaxPool2d(3, stride=2)
        self.conv = BasicConv2d(64, 96, kernel_size=3, stride=2)

    def forward(self, x):
        x0 = self.maxpool(x)
        x1 = self.conv(x)
        out = torch.cat((x0, x1), 1)
        return out


class Mixed_4a(nn.Module):

    def __init__(self):
        super(Mixed_4a, self).__init__()

        self.branch0 = nn.Sequential(
            BasicConv2d(160, 64, kernel_size=1, stride=1),
            BasicConv2d(64, 96, kernel_size=3, stride=1)
        )

        self.branch1 = nn.Sequential(
```

```
            BasicConv2d(160, 64, kernel_size=1, stride=1),
            BasicConv2d(64, 64, kernel_size=(1,7), stride=1, padding=(0,3)),
            BasicConv2d(64, 64, kernel_size=(7,1), stride=1, padding=(3,0)),
            BasicConv2d(64, 96, kernel_size=(3,3), stride=1)
        )

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        out = torch.cat((x0, x1), 1)
        return out


class Mixed_5a(nn.Module):

    def __init__(self):
        super(Mixed_5a, self).__init__()
        self.conv = BasicConv2d(192, 192, kernel_size=3, stride=2)
        self.maxpool = nn.MaxPool2d(3, stride=2)

    def forward(self, x):
        x0 = self.conv(x)
        x1 = self.maxpool(x)
        out = torch.cat((x0, x1), 1)
        return out


class Inception_A(nn.Module):

    def __init__(self):
```

```
        super(Inception_A, self).__init__()
        self.branch0 = BasicConv2d(384, 96, kernel_size=1, stride=1)

        self.branch1 = nn.Sequential(
            BasicConv2d(384, 64, kernel_size=1, stride=1),
            BasicConv2d(64, 96, kernel_size=3, stride=1, padding=1)
        )

        self.branch2 = nn.Sequential(
            BasicConv2d(384, 64, kernel_size=1, stride=1),
            BasicConv2d(64, 96, kernel_size=3, stride=1, padding=1),
            BasicConv2d(96, 96, kernel_size=3, stride=1, padding=1)
        )

        self.branch3 = nn.Sequential(
            nn.AvgPool2d(3, stride=1, padding=1, count_include_pad=False),
            BasicConv2d(384, 96, kernel_size=1, stride=1)
        )

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        x3 = self.branch3(x)
        out = torch.cat((x0, x1, x2, x3), 1)
        return out


class Reduction_A(nn.Module):
```

```
    def __init__(self):
        super(Reduction_A, self).__init__()
        self.branch0 = BasicConv2d(384, 384, kernel_size=3, stride=2)

        self.branch1 = nn.Sequential(
            BasicConv2d(384, 192, kernel_size=1, stride=1),
            BasicConv2d(192, 224, kernel_size=3, stride=1, padding=1),
            BasicConv2d(224, 256, kernel_size=3, stride=2)
        )

        self.branch2 = nn.MaxPool2d(3, stride=2)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        out = torch.cat((x0, x1, x2), 1)
        return out


class Inception_B(nn.Module):

    def __init__(self):
        super(Inception_B, self).__init__()
        self.branch0 = BasicConv2d(1024, 384, kernel_size=1, stride=1)

        self.branch1 = nn.Sequential(
            BasicConv2d(1024, 192, kernel_size=1, stride=1),
            BasicConv2d(192, 224, kernel_size=(1,7), stride=1, padding=(0,3)),
            BasicConv2d(224, 256, kernel_size=(7,1), stride=1, padding=(3,0))
```

```
        )

        self.branch2 = nn.Sequential(
            BasicConv2d(1024, 192, kernel_size=1, stride=1),
            BasicConv2d(192, 192, kernel_size=(7,1), stride=1, padding=(3,0)),
            BasicConv2d(192, 224, kernel_size=(1,7), stride=1, padding=(0,3)),
            BasicConv2d(224, 224, kernel_size=(7,1), stride=1, padding=(3,0)),
            BasicConv2d(224, 256, kernel_size=(1,7), stride=1, padding=(0,3))
        )

        self.branch3 = nn.Sequential(
            nn.AvgPool2d(3, stride=1, padding=1, count_include_pad=False),
            BasicConv2d(1024, 128, kernel_size=1, stride=1)
        )

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        x3 = self.branch3(x)
        out = torch.cat((x0, x1, x2, x3), 1)
        return out


class Reduction_B(nn.Module):

    def __init__(self):
        super(Reduction_B, self).__init__()

        self.branch0 = nn.Sequential(
```

```
            BasicConv2d(1024, 192, kernel_size=1, stride=1),
            BasicConv2d(192, 192, kernel_size=3, stride=2)
        )

        self.branch1 = nn.Sequential(
            BasicConv2d(1024, 256, kernel_size=1, stride=1),
            BasicConv2d(256, 256, kernel_size=(1,7), stride=1, padding=(0,3)),
            BasicConv2d(256, 320, kernel_size=(7,1), stride=1, padding=(3,0)),
            BasicConv2d(320, 320, kernel_size=3, stride=2)
        )

        self.branch2 = nn.MaxPool2d(3, stride=2)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        out = torch.cat((x0, x1, x2), 1)
        return out


class Inception_C(nn.Module):

    def __init__(self):
        super(Inception_C, self).__init__()

        self.branch0 = BasicConv2d(1536, 256, kernel_size=1, stride=1)

        self.branch1_0 = BasicConv2d(1536, 384, kernel_size=1, stride=1)
```

```
        self.branch1_1a = BasicConv2d(384, 256, kernel_size=(1,3), stride=1, padding
            =(0,1))
        self.branch1_1b = BasicConv2d(384, 256, kernel_size=(3,1), stride=1, padding
            =(1,0))

        self.branch2_0 = BasicConv2d(1536, 384, kernel_size=1, stride=1)
        self.branch2_1 = BasicConv2d(384, 448, kernel_size=(3,1), stride=1, padding
            =(1,0))
        self.branch2_2 = BasicConv2d(448, 512, kernel_size=(1,3), stride=1, padding
            =(0,1))
        self.branch2_3a = BasicConv2d(512, 256, kernel_size=(1,3), stride=1, padding
            =(0,1))
        self.branch2_3b = BasicConv2d(512, 256, kernel_size=(3,1), stride=1, padding
            =(1,0))

        self.branch3 = nn.Sequential(
            nn.AvgPool2d(3, stride=1, padding=1, count_include_pad=False),
            BasicConv2d(1536, 256, kernel_size=1, stride=1)
        )

def forward(self, x):
    x0 = self.branch0(x)

    x1_0 = self.branch1_0(x)
    x1_1a = self.branch1_1a(x1_0)
    x1_1b = self.branch1_1b(x1_0)
    x1 = torch.cat((x1_1a, x1_1b), 1)

    x2_0 = self.branch2_0(x)
    x2_1 = self.branch2_1(x2_0)
```

```
        x2_2 = self.branch2_2(x2_1)
        x2_3a = self.branch2_3a(x2_2)
        x2_3b = self.branch2_3b(x2_2)
        x2 = torch.cat((x2_3a, x2_3b), 1)

        x3 = self.branch3(x)

        out = torch.cat((x0, x1, x2, x3), 1)
        return out


class InceptionV4(nn.Module):
    def __init__(self, num_classes=22):
        super(InceptionV4, self).__init__()
        # Special attributs
        self.input_space = None
        self.input_size = cfg.crops_shape
        self.mean = None
        self.std = None
        # Modules
        self.features = nn.Sequential(
            BasicConv2d(1, 32, kernel_size=3, stride=2),
            BasicConv2d(32, 32, kernel_size=3, stride=1),
            BasicConv2d(32, 64, kernel_size=3, stride=1, padding=1),
            Mixed_3a(),
            Mixed_4a(),
            Mixed_5a(),
            Inception_A(),
            Inception_A(),
            Inception_A(),
```

```
            Inception_A(),
            Reduction_A(), # Mixed_6a
            Inception_B(),
            Inception_B(),
            Inception_B(),
            Inception_B(),
            Inception_B(),
            Inception_B(),
            Inception_B(),
            Reduction_B(), # Mixed_7a
            Inception_C(),
            Inception_C(),
            Inception_C()
        )
        self.avg_pool = nn.AvgPool2d(8, count_include_pad=False)
        self.last_linear  = nn.Linear(1536, num_classes)
        self.sigmoid = nn.Sigmoid()

    def logits(self, features):
        x = self.avg_pool(features)
        x = x.view(x.size(0), -1)
        x = self.last_linear(x)
        return x

    def forward(self, input):
        x = self.features(input)
        x = self.logits(x)
        return self.sigmoid(x)
```

Anexa 4

```
from PyQt5.QtCore import *
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtGui import QPixmap, QImage
from GUI_scripts.ProcessThreads import Worker
import config as cfg


class Ui_MainWindow(object):
    def __init__(self):
        self.IMAGE_MODE = 0
        self.VIDEO_MODE = 1
        self.threadpool = QThreadPool()
        self.videoWorker = None
        self.imageWorker = None


    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.setFixedSize(1143, 727)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.image_screen = QtWidgets.QLabel(self.centralwidget)
        self.image_screen.setGeometry(QtCore.QRect(20, 70, 591, 391))
        self.image_screen.setText("")
        self.image_screen.setObjectName("image_screen")
```

```
self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget.setGeometry(QtCore.QRect(70, 540, 291, 141))
self.AU_verticalLayout1.setContentsMargins(0, 0, 0, 0)
self.AU_verticalLayout1.setObjectName("AU_verticalLayout1")
self.AU_0 = QtWidgets.QProgressBar(self.verticalLayoutWidget)
self.AU_0.setProperty("value", 24)
self.AU_0.setObjectName("AU_0")
self.AU_verticalLayout1.addWidget(self.AU_0)
self.AU_1 = QtWidgets.QProgressBar(self.verticalLayoutWidget)
self.AU_1.setProperty("value", 24)
self.AU_1.setObjectName("AU_1")
self.AU_verticalLayout1.addWidget(self.AU_1)
self.AU_2 = QtWidgets.QProgressBar(self.verticalLayoutWidget)
self.AU_2.setProperty("value", 24)
self.AU_2.setObjectName("AU_2")
self.AU_verticalLayout1.addWidget(self.AU_2)
self.AU_3 = QtWidgets.QProgressBar(self.verticalLayoutWidget)
self.AU_3.setProperty("value", 24)
self.AU_3.setObjectName("AU_3")
self.AU_verticalLayout1.addWidget(self.AU_3)
self.AU_4 = QtWidgets.QProgressBar(self.verticalLayoutWidget)
self.AU_4.setProperty("value", 24)
self.AU_4.setObjectName("AU_4")
self.AU_verticalLayout1.addWidget(self.AU_4)
self.imageButton = QtWidgets.QPushButton(self.function_type_groupBox)
self.imageButton.setGeometry(QtCore.QRect(180, 20, 75, 23))
self.imageButton.setObjectName("imageButton")
self.videoButton = QtWidgets.QPushButton(self.function_type_groupBox)
self.videoButton.setGeometry(QtCore.QRect(30, 20, 75, 23))
self.videoButton.setObjectName("videoButton")
```

```
self.line = QtWidgets.QFrame(self.centralwidget)
self.line.setGeometry(QtCore.QRect(-3, 480, 1151, 20))
self.line.setFrameShape(QtWidgets.QFrame.HLine)
self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
self.line.setObjectName("line")
self.verticalLayoutWidget_2 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_2.setGeometry(QtCore.QRect(720, 90, 401, 171))
self.verticalLayoutWidget_2.setObjectName("verticalLayoutWidget_2")
self.EMO_L_verticalLayout = QtWidgets.QVBoxLayout(self.
    verticalLayoutWidget_2)
self.EMO_L_verticalLayout.setContentsMargins(0, 0, 0, 0)
self.EMO_L_verticalLayout.setObjectName("EMO_L_verticalLayout")
self.neutral_emo_bar = QtWidgets.QProgressBar(self.verticalLayoutWidget_2)
self.neutral_emo_bar.setProperty("value", 24)
self.neutral_emo_bar.setObjectName("neutral_emo_bar")
self.EMO_L_verticalLayout.addWidget(self.neutral_emo_bar)
self.happy_emo_bar = QtWidgets.QProgressBar(self.verticalLayoutWidget_2)
self.happy_emo_bar.setProperty("value", 24)
self.happy_emo_bar.setObjectName("happy_emo_bar")
self.EMO_L_verticalLayout.addWidget(self.happy_emo_bar)
self.surprised_emo_bar = QtWidgets.QProgressBar(self.verticalLayoutWidget_2
    )
self.surprised_emo_bar.setProperty("value", 24)
self.surprised_emo_bar.setObjectName("surprised_emo_bar")
self.EMO_L_verticalLayout.addWidget(self.surprised_emo_bar)
self.verticalLayoutWidget_3 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_3.setGeometry(QtCore.QRect(720, 280, 401, 161))
self.verticalLayoutWidget_3.setObjectName("verticalLayoutWidget_3")
self.EMO_R_verticalLayout = QtWidgets.QVBoxLayout(self.
    verticalLayoutWidget_3)
```

```
self.EMO_R_verticalLayout.setContentsMargins(0, 0, 0, 0)
self.EMO_R_verticalLayout.setObjectName("EMO_R_verticalLayout")
self.angry_emo_bar = QtWidgets.QProgressBar(self.verticalLayoutWidget_3)
self.angry_emo_bar.setProperty("value", 24)
self.angry_emo_bar.setObjectName("angry_emo_bar")
self.EMO_R_verticalLayout.addWidget(self.angry_emo_bar)
self.disgusted_emo_bar = QtWidgets.QProgressBar(self.
    verticalLayoutWidget_3)
self.disgusted_emo_bar.setProperty("value", 24)
self.disgusted_emo_bar.setObjectName("disgusted_emo_bar")
self.sad_emo_bar.setProperty("value", 24)
self.sad_emo_bar.setObjectName("sad_emo_bar")
self.EMO_R_verticalLayout.addWidget(self.sad_emo_bar)
self.emotii_label = QtWidgets.QLabel(self.centralwidget)
self.emotii_label.setGeometry(QtCore.QRect(850, 30, 111, 41))
self.emotii_label.setObjectName("emotii_label")
self.au_label = QtWidgets.QLabel(self.centralwidget)
self.au_label.setGeometry(QtCore.QRect(10, 490, 111, 41))
self.au_label.setObjectName("au_label")
self.label_netural = QtWidgets.QLabel(self.centralwidget)
self.label_netural.setGeometry(QtCore.QRect(630, 120, 51, 21))
self.label_netural.setObjectName("label_netural")
self.label_happy = QtWidgets.QLabel(self.centralwidget)
self.label_happy.setGeometry(QtCore.QRect(630, 170, 51, 21))
self.label_happy.setObjectName("label_happy")
self.label_surprised = QtWidgets.QLabel(self.centralwidget)
self.label_surprised.setGeometry(QtCore.QRect(630, 220, 61, 21))
self.label_surprised.setObjectName("label_surprised")
self.label_angry = QtWidgets.QLabel(self.centralwidget)
self.label_angry.setGeometry(QtCore.QRect(630, 300, 51, 21))
```

```
self.label_angry.setObjectName("label_angry")
self.label_sad = QtWidgets.QLabel(self.centralwidget)
self.label_sad.setGeometry(QtCore.QRect(630, 400, 61, 21))
self.label_sad.setObjectName("label_sad")
self.verticalLayoutWidget_4 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_4.setGeometry(QtCore.QRect(460, 540, 271, 141))
self.verticalLayoutWidget_4.setObjectName("verticalLayoutWidget_4")
self.AU_verticalLayout_3 = QtWidgets.QVBoxLayout(self.
    verticalLayoutWidget_4)
self.AU_verticalLayout_3.setContentsMargins(0, 0, 0, 0)
self.AU_verticalLayout_3.setObjectName("AU_verticalLayout_3")
self.AU_5 = QtWidgets.QProgressBar(self.verticalLayoutWidget_4)
self.AU_5.setProperty("value", 24)
self.AU_5.setObjectName("AU_5")
self.AU_verticalLayout_3.addWidget(self.AU_5)
self.AU_6 = QtWidgets.QProgressBar(self.verticalLayoutWidget_4)
self.AU_6.setProperty("value", 24)
self.AU_6.setObjectName("AU_6")
self.AU_verticalLayout_3.addWidget(self.AU_6)
self.AU_7 = QtWidgets.QProgressBar(self.verticalLayoutWidget_4)
self.AU_7.setProperty("value", 24)
self.AU_7.setObjectName("AU_7")
self.AU_verticalLayout_3.addWidget(self.AU_7)
self.AU_8 = QtWidgets.QProgressBar(self.verticalLayoutWidget_4)
self.AU_8.setProperty("value", 24)
self.AU_8.setObjectName("AU_8")
self.AU_verticalLayout_3.addWidget(self.AU_8)
self.AU_9 = QtWidgets.QProgressBar(self.verticalLayoutWidget_4)
self.AU_9.setProperty("value", 24)
self.AU_9.setObjectName("AU_9")
```

```
self.AU_verticalLayout_3.addWidget(self.AU_9)
self.verticalLayoutWidget_5 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_5.setGeometry(QtCore.QRect(820, 540, 281, 141))
self.verticalLayoutWidget_5.setObjectName("verticalLayoutWidget_5")
self.AU_verticalLayout_4 = QtWidgets.QVBoxLayout(self.
    verticalLayoutWidget_5)
self.AU_verticalLayout_4.setContentsMargins(0, 0, 0, 0)
self.AU_verticalLayout_4.setObjectName("AU_verticalLayout_4")
self.AU_10 = QtWidgets.QProgressBar(self.verticalLayoutWidget_5)
self.AU_10.setProperty("value", 24)
self.AU_10.setObjectName("AU_20")
self.AU_verticalLayout_4.addWidget(self.AU_10)
self.AU_11 = QtWidgets.QProgressBar(self.verticalLayoutWidget_5)
self.AU_11.setProperty("value", 24)
self.AU_11.setObjectName("AU_21")
self.AU_verticalLayout_4.addWidget(self.AU_11)
self.AU_12 = QtWidgets.QProgressBar(self.verticalLayoutWidget_5)
self.AU_12.setProperty("value", 24)
self.AU_12.setObjectName("AU_22")
self.AU_verticalLayout_4.addWidget(self.AU_12)
self.AU_13 = QtWidgets.QProgressBar(self.verticalLayoutWidget_5)
self.AU_13.setProperty("value", 24)
self.AU_13.setObjectName("AU_23")
self.AU_verticalLayout_4.addWidget(self.AU_13)
self.AU_14 = QtWidgets.QProgressBar(self.verticalLayoutWidget_5)
self.AU_14.setProperty("value", 24)
self.AU_14.setObjectName("AU_24")
self.AU_verticalLayout_4.addWidget(self.AU_14)
self.verticalLayoutWidget_6 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_6.setGeometry(QtCore.QRect(30, 540, 31, 141))
```

```
self.verticalLayoutWidget_6.setObjectName("verticalLayoutWidget_6")
self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_6)
self.verticalLayout.setContentsMargins(0, 0, 0, 0)
self.verticalLayout.setObjectName("verticalLayout")
self.labelAU0 = QtWidgets.QLabel(self.verticalLayoutWidget_6)
self.labelAU0.setObjectName("labelAU0")
self.verticalLayout.addWidget(self.labelAU0)
self.labelAU1 = QtWidgets.QLabel(self.verticalLayoutWidget_6)
self.labelAU1.setObjectName("labelAU1")
self.verticalLayout.addWidget(self.labelAU1)
self.labelAU2 = QtWidgets.QLabel(self.verticalLayoutWidget_6)
self.labelAU2.setObjectName("labelAU2")
self.verticalLayout.addWidget(self.labelAU2)
self.labelAU3 = QtWidgets.QLabel(self.verticalLayoutWidget_6)
self.labelAU3.setObjectName("labelAU3")
self.verticalLayout.addWidget(self.labelAU3)
self.labelAU4 = QtWidgets.QLabel(self.verticalLayoutWidget_6)
self.labelAU4.setObjectName("labelAU4")
self.verticalLayout.addWidget(self.labelAU4)
self.verticalLayoutWidget_7 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_7.setGeometry(QtCore.QRect(420, 540, 31, 141))
self.verticalLayoutWidget_7.setObjectName("verticalLayoutWidget_7")
self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_7)
self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.labelAU5 = QtWidgets.QLabel(self.verticalLayoutWidget_7)
self.labelAU5.setObjectName("labelAU5")
self.verticalLayout_2.addWidget(self.labelAU5)
self.labelAU6 = QtWidgets.QLabel(self.verticalLayoutWidget_7)
self.labelAU6.setObjectName("labelAU6")
```

```
self.verticalLayout_2.addWidget(self.labelAU6)
self.labelAU7 = QtWidgets.QLabel(self.verticalLayoutWidget_7)
self.labelAU7.setObjectName("labelAU7")
self.verticalLayout_2.addWidget(self.labelAU7)
self.labelAU8 = QtWidgets.QLabel(self.verticalLayoutWidget_7)
self.labelAU8.setObjectName("labelAU8")
self.verticalLayout_2.addWidget(self.labelAU8)
self.labelAU9 = QtWidgets.QLabel(self.verticalLayoutWidget_7)
self.labelAU9.setObjectName("labelAU9")
self.verticalLayout_2.addWidget(self.labelAU9)
self.verticalLayoutWidget_8 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_8.setGeometry(QtCore.QRect(780, 540, 31, 141))
self.verticalLayoutWidget_8.setObjectName("verticalLayoutWidget_8")
self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_8)
self.verticalLayout_3.setContentsMargins(0, 0, 0, 0)
self.verticalLayout_3.setObjectName("verticalLayout_3")
self.labelAU10 = QtWidgets.QLabel(self.verticalLayoutWidget_8)
self.labelAU10.setObjectName("labelAU10")
self.verticalLayout_3.addWidget(self.labelAU10)
self.labelAU11 = QtWidgets.QLabel(self.verticalLayoutWidget_8)
self.labelAU11.setObjectName("labelAU11")
self.verticalLayout_3.addWidget(self.labelAU11)
self.labelAU12 = QtWidgets.QLabel(self.verticalLayoutWidget_8)
self.labelAU12.setObjectName("labelAU12")
self.verticalLayout_3.addWidget(self.labelAU12)
self.labelAU13 = QtWidgets.QLabel(self.verticalLayoutWidget_8)
self.labelAU13.setObjectName("labelAU13")
self.verticalLayout_3.addWidget(self.labelAU13)
self.labelAU14 = QtWidgets.QLabel(self.verticalLayoutWidget_8)
self.labelAU14.setObjectName("labelAU14")
```

```
        self.verticalLayout_3.addWidget(self.labelAU14)
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.function_type_groupBox.setTitle(_translate("MainWindow", "Functioning
            type"))
        self.imageButton.setText(_translate("MainWindow", "Image"))
        self.videoButton.setText(_translate("MainWindow", "Video"))
        self.emotii_label.setText(_translate("MainWindow",
                                             "<html><head/><body><p><span
                                                 style=\" font-size:15pt; font-weight
                                                 :600;\">EMOTIONS</span></p
                                                 ></body></html>"))
        self.au_label.setText(_translate("MainWindow",
                                         "<html><head/><body><p><b><span
                                             style=\" font-size:12pt;\">Action Units
                                             </span></b></p></body></html>")
                                             )
        self.label_sad.setText(_translate("MainWindow",
                                          "<html><head/><body><p><span style
                                              =\" font-size:11pt;\">Sad</span></p
                                              ></body></html>"))
```

```
        self.labelAU0.setText(_translate("MainWindow", "AU1"))
        self.labelAU1.setText(_translate("MainWindow", "AU2"))
        self.labelAU2.setText(_translate("MainWindow", "AU4"))
        self.labelAU3.setText(_translate("MainWindow", "AU5"))
        self.labelAU4.setText(_translate("MainWindow", "AU6"))
        self.labelAU5.setText(_translate("MainWindow", "AU7"))
        self.labelAU6.setText(_translate("MainWindow", "AU9"))
        self.labelAU7.setText(_translate("MainWindow", "AU10"))
        self.labelAU8.setText(_translate("MainWindow", "AU12"))
        self.labelAU9.setText(_translate("MainWindow", "AU15"))
        self.labelAU10.setText(_translate("MainWindow", "AU16"))
        self.labelAU11.setText(_translate("MainWindow", "AU17"))
        self.labelAU12.setText(_translate("MainWindow", "AU20"))
        self.labelAU13.setText(_translate("MainWindow", "AU23"))
        self.labelAU14.setText(_translate("MainWindow", "AU26"))

        self.videoButton.clicked.connect(self.start_video_mode)
        self.imageButton.clicked.connect(self.start_image_mode)

        self.start_video_mode()

    def update_AUs(self, au_scores):
        self.AU_0.setProperty("value", au_scores[0])
        self.AU_1.setProperty("value", au_scores[1])
        self.AU_2.setProperty("value", au_scores[2])
        self.AU_3.setProperty("value", au_scores[3])
        self.AU_4.setProperty("value", au_scores[4])
        self.AU_5.setProperty("value", au_scores[5])
        self.AU_6.setProperty("value", au_scores[6])
```

```
        self.AU_7.setProperty("value", au_scores[7])
        self.AU_8.setProperty("value", au_scores[8])
        self.AU_9.setProperty("value", au_scores[9])
        self.AU_10.setProperty("value", au_scores[10])
        self.AU_11.setProperty("value", au_scores[11])
        self.AU_12.setProperty("value", au_scores[12])
        self.AU_13.setProperty("value", au_scores[13])
        self.AU_14.setProperty("value", au_scores[14])

    def update_EMOs(self, emos_scores):
        self.neutral_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['
            neutral']])
        self.happy_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['happy'
            ]])
        self.surprised_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['
            surprised']])
        self.angry_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['angry'
            ]])
        self.disgusted_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['
            disgusted']])
        self.sad_emo_bar.setProperty("value", emos_scores[cfg.emotions_dict['sad']])

    def update_image(self, image):
        qimage = QImage(image, image.shape[1], image.shape[0], QImage.
            Format_Grayscale8)

        pixmap = QPixmap(qimage)
        self.image_screen.setPixmap(pixmap)

    def start_video_mode(self):
```

```
        #self.imageWorker.shutdown_flag.set()
        self.videoWorker = Worker(self.VIDEO_MODE)
        self.videoWorker.signals.AUs_intensity.connect(self.update_AUs)
        self.videoWorker.signals.image.connect(self.update_image)
        self.videoWorker.signals.EMOs_intensity.connect(self.update_EMOs)

        # Execute
        self.threadpool.start(self.videoWorker)

    def start_image_mode(self):
        options = QtWidgets.QFileDialog.Options()
        options |= QtWidgets.QFileDialog.DontUseNativeDialog
        fileName, _ = QtWidgets.QFileDialog.getOpenFileName(
            None,
            "QFileDialog.getOpenFileName()",
            "",
            "Files (*)",
            options=options)

        if fileName.endswith((".png", ".jpg", ".jpeg")):
            self.videoWorker.shutdown_flag.set()

            self.imageWorker = Worker(self.IMAGE_MODE, fileName)
            self.imageWorker.signals.AUs_intensity.connect(self.update_AUs)
            self.imageWorker.signals.image.connect(self.update_image)
            self.imageWorker.signals.EMOs_intensity.connect(self.update_EMOs)

            # Execute
            self.threadpool.start(self.imageWorker)
```