UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
DOMENIUL DE LICENŢĂ INFORMATICĂ

LUCRARE DE LICENŢĂ

COORDONATOR ŞTIINŢIFIC
CONF. DR. IONESCU RADU TUDOR

STUDENT
ENACHE ALEXANDRU-MĂDĂLIN

BUCUREŞTI
2018

UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
DOMENIUL DE LICENŢĂ INFORMATICĂ

LUCRARE DE LICENŢĂ

APLICAŢIE WEB PENTRU CĂUTAREA INFORMAŢIILOR DESPRE ACTORI PE BAZĂ DE RECUNOAŞTERE FACIALĂ

COORDONATOR ŞTIINŢIFIC
CONF. DR. IONESCU RADU TUDOR

STUDENT
ENACHE ALEXANDRU-MĂDĂLIN

BUCUREŞTI
2018

Cuprins

Capitolul 1

1. Introducere

În ciuda faptului că inteligenţa artificială s-a dezvoltat într-un pas foarte alert în ultimii ani, nu foarte multe

In această lucrare vom folosi o reţea neuronală convoluțională pentru a clasifica un set de date creat specific pentru această aplicaţie, el conţinând un număr de aproximativ 10600 imagini, grupate în 100 categorii, reprezentând cei mai faimoşi actori în urma cautărilor pe platforma IMDB din anul 2017.

Reţelele neuronale convoluţionale au avut un succes imens în ultima jumătate de deceniu, totul pornind de la reţeaua propusă de Alex Krizhevsky, [1].

Astfel, în capitolul 2 vom prezenta conceptele teoretice care stau la baza realizării recunoaşterii faciale din imagini, vom discuta mai pe larg în ce constau cele 2 reţele propuse de Facebook şi Google, iar apoi vom preciza rezultatele obţinute folosind o reţea preantrenată pe setul de date al nostru, precum şi o comparare cu un algoritm de bază.

În capitolul 3 prezentăm tehnologiile folosite în realizarea aplicaţiei web,

Capitolul 4 este dedicat descrierii aplicaţiei, unde vor fi prezentate cazuri de utilizare, diagrame de activităţi, structura bazei de date.

Capitolul 2

2. Recunoaşterea facială din imagini

2.1. Concepte teoretice

2.1.1. Reţele neuronale convoluţionale

Reţelele neuronale convoluţionale sunt o categorie a reţelelor neuronale care s-au dovedit să aibă un succes foarte mare în clasificarea, recunoaşterea obiectelor şi identificarea persoanelor din imagini după feţe. Faţă de reţelele normale, cele convoluţionale au avut un succes mai mare în aceste domenii datorită faptului că spaţialitatea imaginii este păstrată prin folosirea filtrelor.

Reţelele convoluţionale sunt foarte asemănătoare cu cele neuronale ordinare prin faptul că sunt au la bază neuroni în care sunt reţinute informaţii. Un neuron este o unitate în care se poate stoca informaţie, iar aceasta se înmulţeşte cu o matrice de dimensiune 1xN, rezultând un scalar, care apoi este adunat cu un bias. Formula arată în felul următor:

$$y = \sum_{i=1}^{n} x_i * w_i + b \qquad (2.1)$$

De preferat, această operaţie ar trebui să fie urmată de o funcţie neliniară. Acest lucru este datorat faptului că dacă am înlănţui mai multe funcţii liniare precum cea din formula (2.1), tot acest lanţ ar putea fi înlocuit cu o singură funcţie liniară. De aceea, fiecare funcţie liniară este urmată de una neliniară, astfel evitând cazul precizat. În final, formula va arată astfel:

$$y = f\left(\sum_{i=1}^{n} x_i * w_i + b \right) \qquad (2.2)$$

De-a lungul anilor au fost folosite mai multe funcţii neliniare în acest domeniu, denumite şi funcţii de activare. Prima funcţie pe care o vom prezenta este funcţia **Sigmoid**, notată σ. Aceasta are următoarea formulă matematică şi are următoarea distribuţie în jurul punctului 0.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

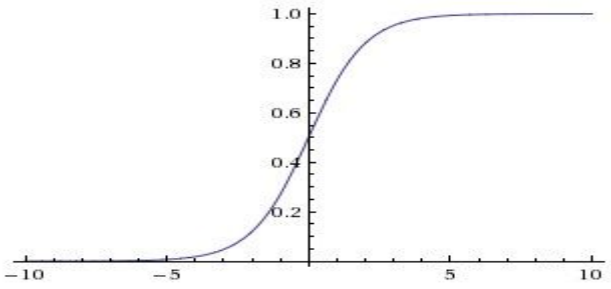

Figura 2.1: Reprezentarea grafică a funcţiei sigmoid

Sursă: CS231n [17]

Funcţia sigmoid a fost folosită în trecut datorită faptului că reuşeste să restrângă variabila în intervalul [0, 1]. În practică, această funcţie este folosită foarte rar deoarece are câteva mari dezavantaje. Primul problemă este aceea că gradientul în jurul punctelor 0 şi 1 este aproape 0. Astfel, în propagarea înapoi a reţelei, semnalul care v-a trece prin neuron va fi foarte mic şi nu vor aparea schimbări prea mari în ponderile acestuia, reţeaua învaţând foarte greu. Acest lucru poate fi evitat parţial prin iniţializarea foarte atentă a ponderilor neuronilor care folosesc sigmoid.

Un alt dezavantaj este acela ca rezultatele funcţiei sigmoid nu sunt centrate în 0. Acest lucru nu este dorit datorită faptului că dacă toate datele de intrare au acelaşi semn, gradienţii în propagarea înapoi vor avea toţi acelaşi semn, fie pozitiv, fie negativ. În plus, funcţia exponenţială este scumpă din punct de vedere computaţional.

Funcţia **Tangentă** este o funcţie de activare care restrânge valorile în intervalul [-1, 1]. Precum funcţia sigmoid, tangenta este computaţional scumpă şi saturează, dar valorile rezultate sunt centrate în 0. De aceea, tangenta este mereu preferată faţă de sigmoid în practică. Un lucru interesat care poate fi observat este că tangenta poate fi formată dintr-o funcţie sigmoid:

$$\tanh(x) = 2 * \sigma(2 * x) + 1 \tag{2.4}$$

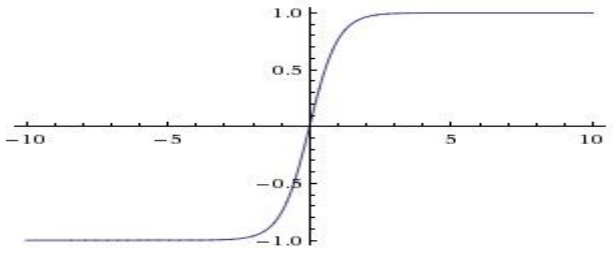

Figura 2.2: Reprezentarea grafică a funcţiei tangentă

Sursă: CS231n [17]

În continuare vom prezenta funcţia de activare **ReLU**. Aceasta a fost propusă pentru a fi folosită în reţelele neuronale de către Alex Krizhevsky în [1]. În această lucrare ne este indicat să folosim ReLU în detrimentul tangentei, acurateţea ramânând aceeaşi, deşi ReLU este de 6 ori mai rapidă. Această are urmatoarea formulă:

$$f(x) = \max(0, x) \tag{2.5}$$

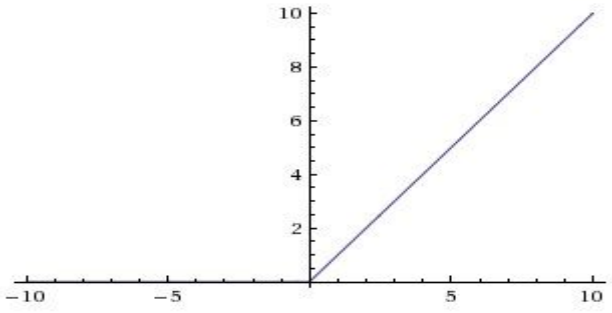

Figura 2.3: Reprezentarea grafică a funcţiei ReLU

Sursă: CS231n [17]

O mare problemă care poate apărea în antrenarea reţelelor neuronale este cazul în care reţeaua învaţă să clasifice foarte bine setul de date oferit la intrare, dar nu se descurcă bine pe unul nou. Pentru a evita acest lucru, ne este propusă o tehnică foarte eficientă şi simplă de ‘adormire’ a neuronilor în [2]. Aceasta ne spune ca în momentul antrenării să ‘adormim’ un neuron cu o probabilitate p. Astfel, la fiecare pas de propagare înainte, doar o parte din neuroni vor participa în acea etapă şi doar ei vor fi actualizaţi la propagarea înapoi, restul fiind setaţi să aibă valoarea 0 în acea rundă.

Mai mulţi neuroni din acelaşi nivel care conţin o funcţie de activare formează un strat. O reţea neuronală este compusă din mai multe astfel de straturi complet conectate între ele şi un strat final care ne va returna răspunsul.

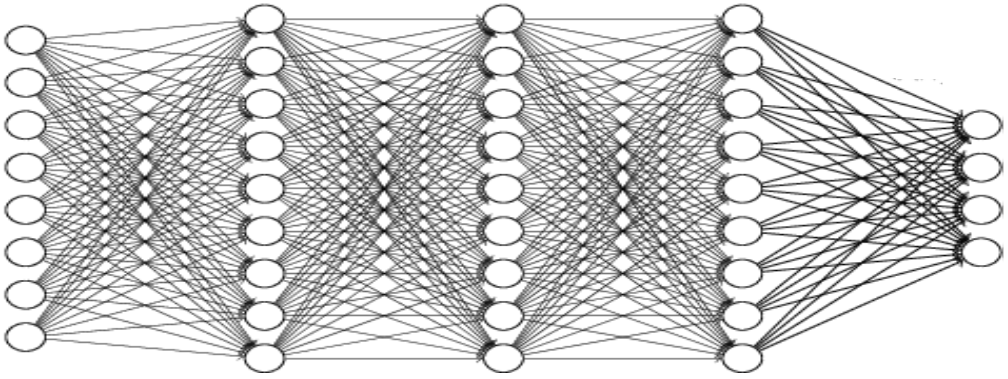

Figura 2.4: Reprezentarea grafică a unei reţele neuronale

Sursă: Google [19]

Ultimul strat al unei reţele este unul de decizie. Aici va avea loc prezicerea asupra cărei clase aparţin datele de intrare. Cea mai folsită funcţie pentru a efectua acest lucru este **Softmax**. Ea primeşte la intrare un vector de dimensiune n şi returnează un vector între 0 şi 1, având suma totală egală cu 1. Astfel, valoarea vectorul de ieşire poate fi gândită ca 'probabilitatea ca un element să aparţină acelei clase'. Funcţia softmax are urmatoarea formulă:

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}} \tag{2.6}$$

În final vom discuta despre funcţia de cost. Această funcţie ne spune cât de bine am aproximat rezultatul final. Astfel, putem vedea cât de aproape am fost de răspunsul corect şi putem modifica valorile neuronilor. Rezultatul acestei funcţii este numită eroarea acelui pas. Cea mai cunoscută funcţie de cost este aceea în care calculăm pătratul diferenţei dintre răspunsul corect şi răspunsul prezis de noi pentru fiecare clasă, iar apoi le adunăm pe toate pentru a obţine eroarea totală. În funcţie de tipul problemei pe care încercăm sa o rezolvăm, pot apărea diferite funcţii de cost.

Valoarea funcţiei de cost este influenţată de ponderile neuronilor. Astfel, pentru minimizarea erorii, ponderile sunt modificate în pasul de propagarea înapoi. Aceste modificări au loc folosind o funcţie de optimizare. În general, această funcţie calculează gradientul unui neuron, adică derivata funcţiei de cost în raport cu ponderile, iar apoi ponderile sunt modificate în direcţia opusă acestui gradient calculat.

În acest punct avem toate componentele necesare pentru a construi o reţea neuronală ordinară. Toate aceste lucruri vor apărea şi în reţelele convoluţionale.

Principala diferenţă între o reţea convoluţională şi una ordinară este faptul că ştim că la intrare vom primi o imagine. Fiecare imagine poate fi gândită ca o matrice de numere cuprinse între 0 şi 255. Un canal este termenul folosit prin care ne referim la o componentă a imaginii. O imagine color are 3 canale (roşu, verde şi albastru), putând fi exprimată ca 3 matrici suprapuse, câte una pentru fiecare canal. Imaginile alb-negru au un singur canal, deci vor fi reprezentate ca o matrice cu componentele între 0 şi 255, de la alb la negru cu cât scazi mai mult valoarea elementului.

O reţea convoluţională este formată dintr-o înşirare de straturi. Totuşi, spre deosebire de reţelele ordinare care folosesc doar straturi complet conectate între ele (fig. 2.4), cele convoluţionale mai introduc 2 tipuri de straturi: convoluţionale şi de unificare.

Stratul convoluţional este cea mai importantă componentă a unei reţele convoluţionale. Convoluţiile au proprietatea de a păstra legătura spaţială dintre pixeli, fiind folosite pătrate de dimensiune mici, numite filtre. Filtrul va porni din partea stânga-sus şi va parcurge imaginea de la stânga la dreapta şi apoi de sus în jos. La fiecare pas, este efectuat produsul dintre filtru şi submatricea acoperită de filtru, iar apoi este efectuată suma componentelor matricei rezultate.

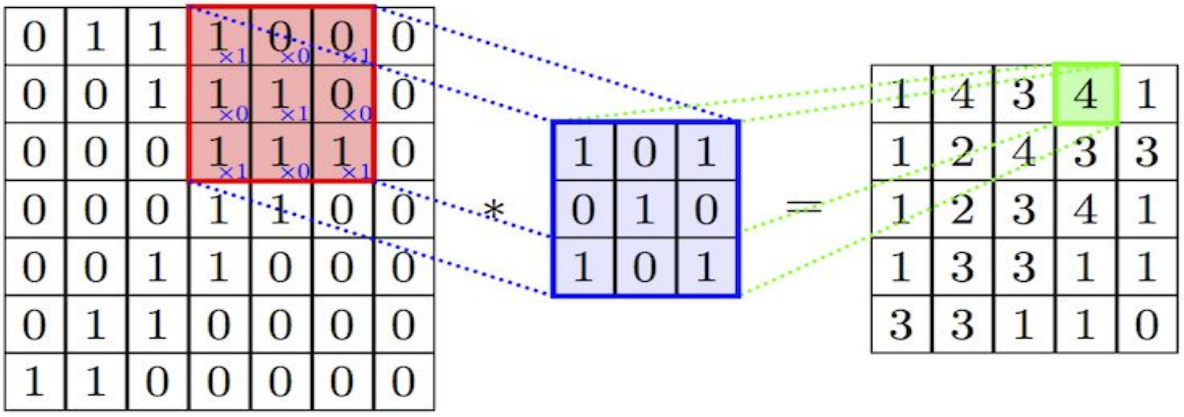

Figura 2.5: Reprezentarea grafică a unui pas de convoluţiei

Sursă: Google [19]

Pentru fiecare strat convoluţional trebuie să decidem ce dimensiune va avea filtru, peste câţi pixeli sa săra la fiecare pas şi de unde să înceapă. Conform [3], filtrele ar trebui să aibă dimensiunea de 3x3, aceasta fiind cea mai mică dimensiune care păstrează noţiunile de centru, sus-jos şi stânga-dreapta. Filtrele de dimensiuni mari ar trebui să fie schimbate cu mai multe filtre de dimensiune 3x3 consecutive. Astfel, putem înlocui un filtru de dimensiune 7x7 cu 3 filtre de dimensiune 3x3 cu deplasarea 1, iar faptul că sunt folosite mai multe filtre va face funcţia de decizie să fie mai discriminativă. Se observă că un strat convoluţional 7x7 are acelaşi câmpul vizual precum cel al 3 convoluţii consecutive 3x3. În final, se poate observa o micşorare al numărului parametrilor folosiţi. Un strat de dimensiune 7x7 ar avea nevoie de 7 x 7 x C x C parametri, C reprezentând numărul de canale, pe când cele 3 straturi 3x3 ar avea nevoie de doar 3 x (3 x 3 x C x C). Astfel, stratul 7x7 foloseşte cu 81% mai mulţi parametri decât cele 3 de dimensiune 3x3, ne spune [3].

Deplasarea reprezintă numărul de pixeli peste care va sări filtrul după fiecare pas. Când deplasarea este de dimensiune 1, atunci filtrul va merge câte un pixel, iar când aceasta este 2, filtrul va sări peste un pixel. Creşterea pasului de deplasare va genera o matrice de trăsături de dimensiune mai mică.

Uneori ne va fi nevoie să controlăm dimensiunea matricei de trasături rezultată, iar acest lucru se poate realiza prin adăugarea de zerouri în jurul imaginii înaintea unei convoluţii.

Când construim o reţea convoluţională, trebuie sa avem grijă la dimensiunea ferestrei care va rezulta după un strat convoluţional. Dimensiunea matricei de trăsături rezultate în urma stratului de convoluţie poate fi calculată astfel:

$$D = \frac{W - F + 2 * P}{S} + 1 \qquad (2.7)$$

, unde D reprezintă dimensiunea matricei de trăsături, W dimensiunea înălţimea/lăţimea ferestrei de intrare, F dimensiunea filtrului, 2 * P numărul de linii/coloane de zerouri adăugat, iar S dimensiunea pasului de deplasare.

Stratul de unificare este folosit pentru a reduce dimensiunea matricei de intrare, dar păstrând cea mai importantă informaţie. Conform [4], cea mai folosită metodă de unificare este cea a maximului. Prin aceast procedeu, matricea de trăsături este împărţită în zone şi este ales maximul din fiecare petic.

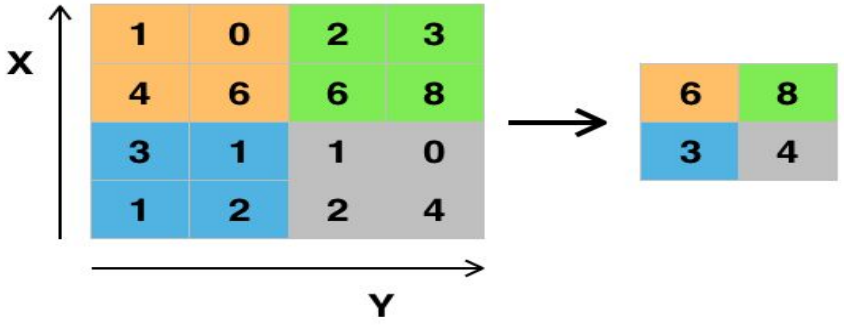

Figura 2.6: Strat de unificare prin extragerea maximului

Sursă: Wikipedia [15]

Există şi alte metode de unificare, precum calcularea mediei sau a sumei. La fel ca în convoluţii, când folosim straturi de unificare trebuie să avem grijă la dimensiunea matricei de trasături rezultate. În [4] ne este dată formula care ne returnează dimensiunea matricei finale, aceasta fiind aceeaşi indiferent de tipul de unificare:

$$D = \frac{W - F}{S} + 1 \quad (2.8)$$

, unde D reprezintă dimensiunea matricei de trăsături, W dimensiunea înălțimea/lățimea ferestrei de intrare, F dimensiunea filtrului, iar S dimensiunea pasului de deplasare.

Se poate observa ca formula 2.8 este un caz particular ar celei 2.7, atunci când P = 0. Acest lucru are sens, deoarece în straturile de unificare nu sunt adăugare zerouri pentru a controla dimensiunea matricei de trăsături.

În final, o rețea neuronală convoluțională va fi compusă din mai multe straturi convoluționale, printre care apar straturi de unificare şi apoi unul sau mai multe straturi complet conectate între ele, încheiate cu un strat de softmax pentru a ne rezulta probabilitățile de apartenenţă pentru fiecare clasă.

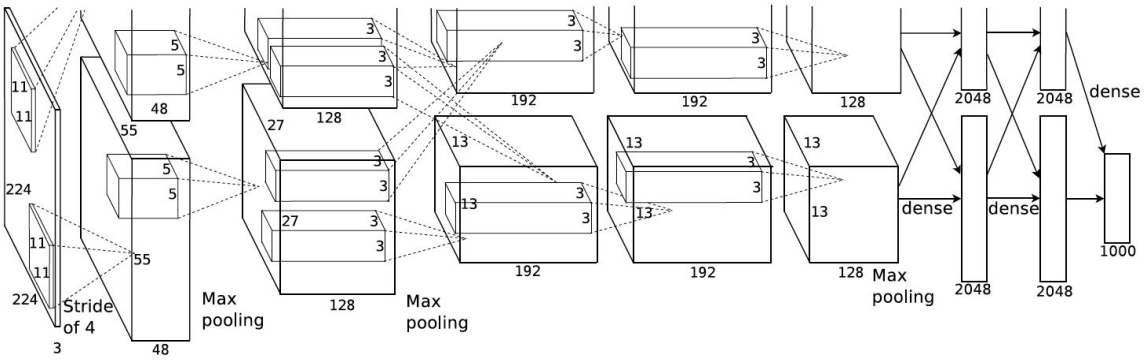

Figura 2.7: Reţeaua AlexNet, propusă de Alex Krizhevsky

Sursă: [1]

Alex Krizhevsky, Ilya Sutskever şi Geoffrey Hinton au şocat lumea în 2012, când au prezentat o reţea neuronală convoluțională care a câştigat concursul ILSVRC, obținând o eroare de doar 15.4% pe primele 5 predicții, față de precedenta de 26.2%. Acesta a fost primul model care a avut un astfel de succes pe setul de date foarte dificil ImageNet. Pornind de la ideile prezentate de cei 3 în [1], an de an au apărut rețele care au bătut recordul precedent. În 2015, Microsoft a ajuns la o eroare pe primele 5 predicții de 3.56% prin reţeaua ResNet, prezentată în [5].

2.1.2. Histograma gradienţilor orientaţi (HOG)

Histograma gradienţilor orientaţi a fost folosită pentru a extrage trăsăturile unui obiect din imagini. Spre deosebire de o reţea neuronală convoluţională care învaţă de una singură cum să extragă trăsături, această metodă are un algoritm predefinit pentru a calcula aceste date de ieşire. Motivul pentru care, în general, reţelele convoluţionale funcţionează mai bine decât descriptorul HOG este acela că reţelele învaţă cum trebuie să extragă trăsăturile pentru a obţine o acurateţe cât mai bună pe un anumit set de date, pe când descriptorul HOG are acelaşi mod de extragere a trăsăturilor, indiferent de parametrii. De aceea, când avem un set de date cu multe imagini, este de preferat să antrenăm o reţea deoarece aceasta va avea o acurateţe mai bună aproape de fiecare dată. Totuşi, dacă avem un set de date mic şi nu avem posibilitatea să creştem numărul de imagini, este posibil ca metoda histogramei gradienţilor orientaţi sa ofere o acurateţe mai bună.

Principala ideea din spatele histogramei gradienţilor orientaţi este aceea că aspectul şi forma locală a unui obiect poate fi descrisă de distribuirea intensităţii gradienţilor sau de orientarea marginilor, [6]. Imaginea este împarţită în regiuni conectate numite celule, iar pixelii din fiecare celulă vor determina o histogramă. Descriptorul este o concatenare a acestor histograme.

Definiţia 2.1: [15] *Gradientul unei imagini este schimbarea direcţiei în intensitate sau culoare dintr-o imagine.*

Detectarea muchiilor dintr-o imagine este una dintre cele mai întâlnite folosiri ale gradienţilor.

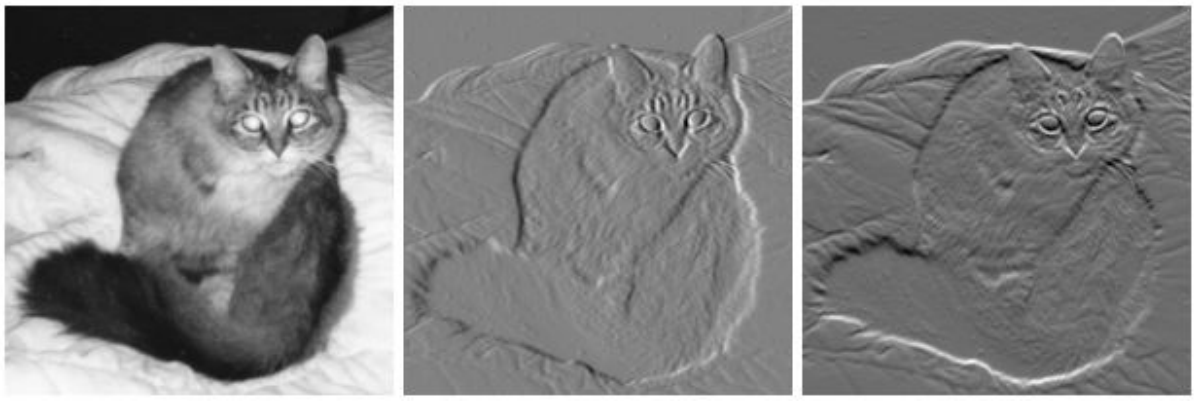

Figura 2.8: Gradienţii unei imagini

Sursă: Wikipedia [15]

În partea stângă avem imaginea alb-negru a unei pisici. În centru avem imaginea gradienţilor în direcţia x, măsurând schimbările orizontale în intensitate. În partea dreaptă avem imaginea gradienţilor în direcţia y, măsurând schimbările verticale în intensitate. Astfel, pixelii gri au o valoare mică a gradienţilor, pe când pixeli albi şi negri au o valoare ridicată.

Matematic, gradientul unei imagini este un vector format din derivatele sale parţiale:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}, \tag{2.9}$$

unde $\frac{\partial f}{\partial x}$ este derivata în funcţie de x, adică gradientul pe direcţia x, pe când $\frac{\partial f}{\partial y}$ este derivata în funcţie de y, reprezentând gradientul pe direcţia y.

Direcţia gradientului este dată de:

$$\theta = tanh^{-1}\left[\frac{g_x}{g_y}\right] \tag{2.10}$$

, iar magnitudinea gradientului, reprezentând cât de repede va creşte intensitatea în acea direcţie, poate fi calculată prin formula:

$$G = \sqrt{g_x^2 + g_y^2} \tag{2.11}$$

Una dintre cele mai simple metode de a găsi gradienţii unei imagini este prin folosirea unui filtru Sobel. Pentru găsirea gradientului pe direcţia x, filtrul Sobel va parcurge imaginea precum un filtru dintr-un strat convoluţional al unei reţele. Pentru găsirea gradientului pe direcţia y vom aplica aceeaşi metoda, dar vom folosi transpusa filtrului Sobel. Pentru a obţine gradienţii pe ambele direcţii, este suficient să trecem cu primul filtru pe imaginea iniţială, iar apoi cu al doilea filtru pe imaginea rezultată după aplicarea primului filtru. Imaginea finală va fi folosită în construirea histogramei gradienţilor orientaţi. În acest pas este calculată direcţia şi magnitudinea gradienţilor.

Filtrul Sobel, numit după Irwin Sobel, este un filtru de dimensiune 3x3 şi are urmatoarele componente:

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}.$$

Figura 2.9: Componentele filtrului Sobel pe direcţia x

Sursă: Wikipedia [15]

Alte filtre pot fi folosite pentru a obţine gradienţii unei imagini. În [7], histograma gradienţilor orientaţi a fost folosită pentru a detecta pietonii dintr-o intersecţie. Aici, filtrul care a dat cele mai bune rezultate a fost unul de dimensiune 1x3, având valorile [-1, 0, 1].

După aflarea gradienţilor şi împărţirea imaginii pe celule, urmează să construim histograma în fiecare dintre aceste zone. Matricea direcţiei gradienţilor va conţine valori cuprinse între 0 şi 359, reprezentând măsura unghiului direcţiei gradientului într-un cerc trigonometric. Astfel, histograma va împărţii aceste unghiuri în porţiuni, iar fiecare pixel îşi va aduna valoarea magnitudinii în componenta corespunzătoare direcţiei sale. După ce toţi gradienţii pixelilor unei celule au fost parcurşi, histograma acelei celule va fi completă. Histograma gradienţilor orientaţi ale unei imagini rezultă din concatenarea histogramelor tuturor celulelor acelei imagini.

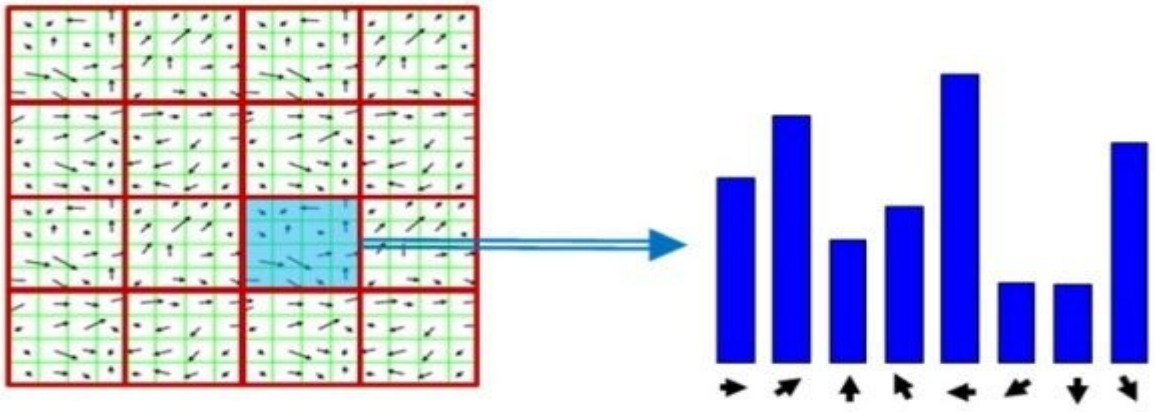

Figura 2.9: Reprezentarea grafică a histogramei unei celule

Sursă: Google [19]

2.1.3. Maşini cu suport vectorial (SVM)

Maşinile cu suport vectorial sunt folosite în învăţarea supervizată. Acestea mapează parametrii de intrare într-un spaţiu de dimensiune foarte mare neliniar. În acest spaţiu, o suprafaţă liniară de decizie este construită. Proprietăţile speciale ale suprafeţei de decizie ne asigură o generalizare mare a maşinilor de învăţare.

Definiţia 2.2: **[15]** *Învăţarea supervizată în inteligenţa artificială este problema învăţării unei funcţii care mapează datele de intrare pe unele de ieşire plecând de la mai multe exemple de intrare-ieşire.*

SVM-ul este printre cele mai bune(şi mulţi cred că este cel mai bun) algoritm de învăţare supervizată standard, [11]. Acest clasificator defineşte un hiperplan pentru a separa clasele diferite.

Definiţia 2.3: **[15]** *În geometrie, un hiperplan este un subspaţiu a cărui dimensiune este mai mică cu 1 decât a spaţiului ambiental.*

Dacă spaţiul este de dimensiune 3, atunci hiperplanele sale au dimensiunea 2, iar dacă spaţiul are dimensiunea 2, atunci hiperplanele sunt drepte de dimensiunea 1.

Definiţia 2.4: **[15]** *Spaţiul ambiental este spaţiul care înconjură un obiect.*

Un algoritm de antrenare SVM crează un sistem care, plecând de la un set de date anotat corespunzător, atribuie unei noi intrări una dintre clase, formând un clasificator liniar.

Un model de maşină cu suport vectorial este o reprezentare a datelor ca puncte în spaţiu, fiind mapate astfel încât cele aparţinând categoriilor diferite să fie despărţite de un gol clar care este cât mai mare posibil. Pe lângă faptul că efectuează clasificări liniare, maşinile cu suport vectorial pot executa clasificări neliniare folosind ceea ce se numeşte trucul nucleului, mapând implicit datele sale de intrare într-un spaţiu al trăsăturilor cu dimensiune foarte mare, [15].

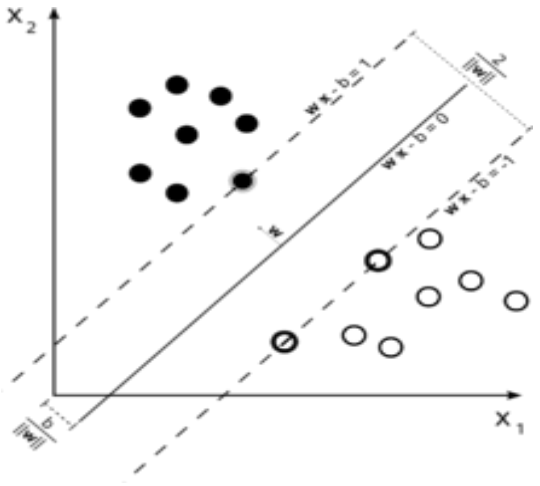

Figura 2.10: Reprezentarea grafică a unui SVM liniar

Sursă: Wikipedia [15]

Dreptele cu linie punctată se numesc vectori suport. Deşi există multe linii care pot separa cele doua categorii din figura (2.10), hiperplanul optim este cel aflat la mijlocul distanţei dintre cei 2 vectori suport şi paralel cu aceştia.

Învaţarea hiperplanelor într-un SVM liniar este efectuată folosind algebră liniară.

Pornind de la un set de date format din n puncte de forma

$$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$$

, unde y reprezintă clasa din care face parte punctul x (-1 sau 1), iar x este un vector de dimensiune m format din numere reale. Trebuie găsit hiperplanul care desparte cele 2 grupe pentru care distanţa minimă dintre acesta şi oricare punct să fie maximă.

Putem scrie orice hiperplan ca:

$$\vec{w} \cdot \vec{x} - b = 0, \tag{2.12}$$

, unde w este vectorul normal al hiperplanului.

Definiţia 2.5: [15] *În geometrie, vectorul normal este vectorul perpendicular pe un obiect dat.*

Construirea modelului SVM este influenţată de 2 paremetri:

- parametrul de **Regularizare** (notat cu C în libraria Scikit-Learn pentru Python) spune clasificatorului cât de mult îi permiţi să clasifice greşit datele de antrenare. Pentru valori mari ale lui C, optimizatorul va alege un hiperplan de dimensiune mai mică pentru a face mai bună clasificarea punctele de antrenare. Pe de altă parte, o valoare mică a lui C va face optimizatorul să caute un hiperplan de lungime mai mare, chiar dacă acel hiperplan va clasifica greşit câteva puncte, [16]. Problema cu alegerea unui C cu valoare mare este aceea că este posibil ca SVM-ul să nu prezică corect datele de testare. De aceea, trebuie să descoperim valoarea variabilei C în funcţie de fiecare set de date.
- parametrul **Gamma** defineşte care puncte influenţează crearea hiperbolei de separare a claselor. Un gamma mic va face ca şi punctele aflate la o distanţă mai mare de posibila hiperbolă să fie luate în vedere în calcularea soluţiei. Pe de altă parte, un gamma mare va spune să folosim doar punctele din apropiere, [16].

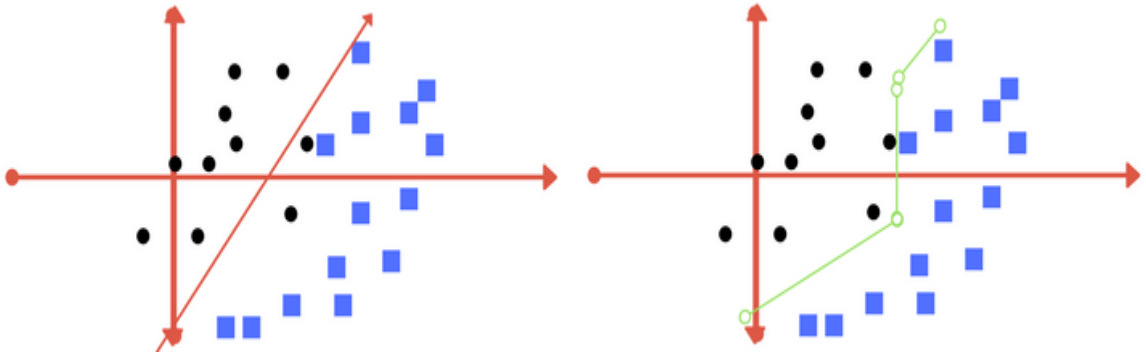

Figura 2.11: În stânga SVM-ul are valoarea parametrului de regularizare mică.

În dreapta SVM-ul are valoarea parametrului de regularizare mare.

Sursă: Mediul [16]

În continuare vom discuta despre margine în maşinile cu suport vectorial. SVM-ul încearcă să obţină o margine cât mai bună.

Definiţia 2.6: **[15]** *În inteligența artificială, marginea unui punct este definită ca distanța dintre acel punct și limita de decizie.*

Definiţia 2.7: **[15]** *Limita de decizie este regiunea în care rezultatul unui clasificator este ambiguu.*

Limita de decizie căutată trebuie să maximizeze marginea pentru a produce un model în ale cărui predicţii avem mai mare încredere că sunt bune. Acest lucru va face clasificatorul să separe exemplele pozitive de cele negative eficient.

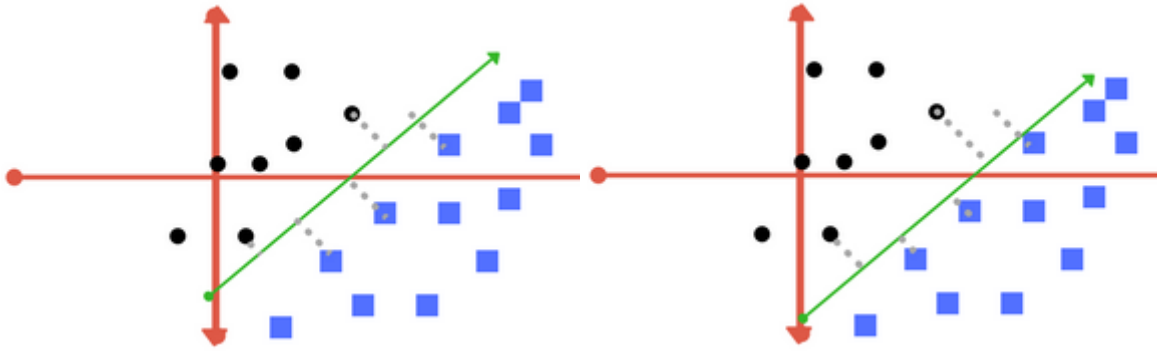

Figura 2.12: În stânga marginea este bună.

În dreapta marginea nu este aleasă bine.

Sursă: Mediul [16]

Marginea din imaginea stângă este bună deoarece aceasta a fost aleasă cât de egal s-a putut față de cele două părți. Pe de altă parte, imaginea din dreapta nu are marginea aleasă corect deoarece hiperbola este mai apropiată de clasa care conţine pătratele albastre.

2.1.4. Metoda celor mai apropiaţi k vecini (KNN)

Deşi a fost introdusa acum mai bine de 60 de ani, metoda celor mai apropiaţi k vecini este încă topic în inteligenţa artificială, [11]. Aceast algoritm este folosit pentru clasificare şi regresie.

În construirea clasificatorului KNN avem nevoie de 3 componenete:

- setul de date pentru antrenare cu clasele corespunzătoare
- metoda prin care se va calcula distanţa între 2 înregistrari. Cea mai folosită metrică în inteligenţa artificială este distanţa Euclidiana, aceasta având următoarea formulă pentru 2 puncte p = $(p_1, p_2, ..., p_n)$ şi q = $(q_1, q_2, ..., q_n)$, [15]

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \qquad (2.13)$$

- variabila k, reprezentând numărul de vecini care sunt luaţi în calcul în etapa de prezicere. Această variabilă este foarte importantă în vederea acurateţii pe care o vom obţine de la clasificatorul KNN. Dacă valoarea este prea mică, atunci algoritmul poate avea probleme în clasificarea setului de testare. În plus, alegerea unui k mic face ca modelul să fie foarte sensibil la date de intrare introduse eronat. Pe de altă parte, dacă alegem un k prea mare putem ajunge la situaţia în care decizie să fie luată de vecinii aflaţi la o distanţă mare faţă de punctul curent, aceştia fiind irelevanţi pentru etapa curentă.

Algoritm

Fie x un punct pentru care trebuie să prezicem clasa în care aparţine. Iniţial, vom calcula distanţele de la x la toate celelalte puncte din setul de antrenare şi vom sorta vectorul rezultat crescător. Dacă k = 1, atunci clasa lui x va fi aceeaşi cu clasa celui mai apropiat vecin. Pentru k > 1 decizia va fi luată de un vot majoritar. În cazul în care votul se termină cu remiză, atunci decizia poate fi luată fie prin alegerea clasei celui mai apropiat punct, fie aleator din clasele aflate la egal, [11].

Un dezavantaj al acestui algoritm simplu este faptul că este destul de costisitor din punct de vedere al timpului de prezicere, metoda trebuind să calculeze de fiecare dată toate distanţele şi să afle care sunt primele k dintre acestea. Totuşi, alţi algoritmi mai rapizi au fost

propuşi de Belur Dasarathy în [12], bazat pe arbori de căutare care partiţionează spaţiul şi ghidează cautarea, precum şi de Farago în [13].

În momentul în care verifici un model KNN pe lângă unul SVM, se pot observa următoarele:

- la antrenare KNN-ul termină imediat, neavând de executat nicio operaţie, pe când în SVM trebuiesc create hiperbolele de decizie.
- la testare SVM-ul se mişcă mai repede decât un model KNN pe acelaşi set.
- acurateţea celor 2 modele diferă de la set la set şi în funcţie de parametrii aleşi. Astfel, dacă trebuie să alegi un clasificator, cel mai bine ar fi să verifici mai multe modele şi să vezi pe ce obţii cele mai bune rezultate.

Un lucru interesant care se poate observa într-un model KNN cu k = 1 este acela că limitele de decizie generează o diagrama Voronoi.

Figura 2.13: De la stânga la dreapta: setul de date, reprezentare grafică KNN pentru k = 1, reprezentare grafică KNN pentru k = 5.

Sursă: Wikipedia [15]

În partea stângă avem setul de date colorat diferit în funcţie de clasa de care aparţine. Îmaginea din centru ne prezintă limitele de decizie ale clasificatorului KNN cu k = 1 folosind distanţa euclidiană. În dreapta clasificatorul a fost calculat cu k = 5. Zonele albe au fost cauzate de remiză în calcularea votului majoritar (de exemplu: 2 vecini au clasa rosie, 2 au clasa albastră şi ultimul are clasa verde), [17].

2.2. Metode recente pentru recunoaşterea facială din imagini

2.2.1. FaceNet

Reţeaua convoluţională FaceNet a fost propusă de cei de la Google în 2015. Spre deosebire de reţelele precedente care pentru a calcula acurateţea pe un set de date nou care antrenau un strat clasificator şi apoi foloseau încă un start intermediar pentru a generaliza clasificarea, aceştia s-au gândit la o modalitate total diferită.

Conform [14], ei au construit o reţea care pentru o imagine trimisă prin reţea să returneze un vector de 128 de numere reale. Vectorul rezultat din reţea are următoarea proprietate: distanţa euclidiană dintre el şi vectorul realizat dintr-o altă imagine a aceeaşi persoane este mai mică decat distanţa faţă de vectorul rezultat din imaginea unei persoane diferite. Astfel, imaginile vor fi împărţite în grupuri într-un spaţiu de dimensiunea 128. Acest lucru ne permite să antrenăm un clasificator elementar precum SVM sau KNN şi să obţinem rezultate la fel de bune ca în alte reţele.

Spre deosebire de reţeaua FaceNet care returnează doar 128 octeţi, celelalte reţele, pornind de la AlexNet în 2012 [1] şi ajungând la VGG în 2014 [3], au avut ultimul strat complet interconectat de dimensiune 1000.

Reţeaua FaceNet a fost antrenată folosind o funcţie de cost triplă. La fiecare pas, reţelei i-a fost dat ca date de intrare 2 imagini ale aceleiaşi persoane şi una a altei persoane. Spre deosebire de reţelele precedente a căror funcţii de cost le făceau să înveţe să clasifice corect persoanele din imagini, funcţia de cost triplă are rolul de a face ca vectorul rezultat să aproprie imaginile ale aceleiaşi persoane prin distanţa euclianã. Formula funcţiei de cost triplă este următoarea:

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ . \tag{2.14}$$

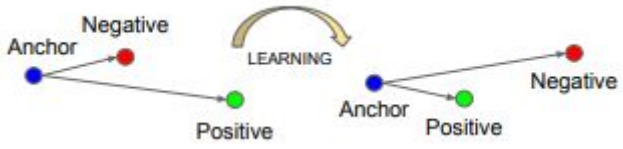

Figura 2.14: Antrenarea în ResNet

După cum se poate observa în figura 2.14, în reţea sunt introduce 3 imagini. Funcţia de cost triplă încearcă să apropie imaginea ancoră de imaginea pozitivă, adică cea a aceleaşi persoane, şi să o depărteze de cea negativă.

2.2.2. ResNet

Pornind de la reţeaua AlexNet care avea doar 8 staturi şi ajungându-se la VGG, aceasta având 16 staturi, s-a pus problema a numărului optim de staturi pe care trebuie să le aibă o reţea. Totuşi, în [5] s-a arătat că pentru a scoate o acurateţe mai bună nu este suficient doar să antrenăm o reţea cu un număr mare de straturi. Aceştia au obţinut o eroare pe setul de antrenare mai mare pentru o reţea cu 56 staturi decât pe una cu 20.

Explicaţia acestui fenomen este următoarea: Deoarece există un număr mare de staturi în acea reţea, gradientul în pasul de propagare înapoi în primele staturi o să aibă o valoare foarte mică, posibil să ajungă chiar la 0, şi reţeaua nu va reuşi să înveţe.

Pentru a evita această situaţie, autorii articolului [5] au propus o învăţare reziduală. Ei au presupus să introducă o funcţie de identitate, numită start rezidual, care va sări unul sau mai multe staturi. Reţeaua în care au fost introduse aceste staturi reziduale nu ar trebui să producă o acurateţe mai slabă decât una fără acestea deoarece am putea folosi doar staturile iniţiale şi astfel prima reţea ar fi identică cu cea de-a doua, rezultând o acurateţe egală.

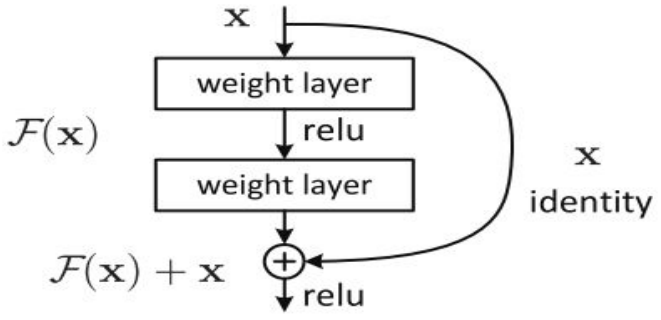

Figura 2.15: Reprezentarea unui strat rezidual

Prin acest procedeu, Microsoft a reuşit să construiască o reţea care de 152 staturi prin care a câştigat ILSVRC 2015, obţinând o eroare pe primele 5 predicţii de 3,56%.

2.3. Algoritmul de recunoaştere facială implementat

În implementarea algoritmului de recunoaştere facială am folosit o reţea preantrenată ResNet pentru a extrage trăsăturile din imagini şi un clasificator simplu precum SVM sau KNN pentru a antrena modelul pe setul de date propriu. Motivul pentru care am fost nevoie să folosesc o reţea preantrenată şi nu am antrenat una de început este acela că setul de date este foarte restrâns, conţinând doar 10600 imagini şi fiind împărţit în 100 clase, iar antrenarea unei noi reţele pe un astfel de set ar produce o acurateţe mai mică decât folosirea uneia deja antrenate.

Algoritmul de recunoaştere a actorilor a fost făcut în limbajul Python. Python este un limbaj de programare gratuit care poate fi rulat pe toate sistemele de operare. Popularitatea acestui limbaj a crescut imens datorită interesului care este acordat inteligenţei artificiale, fiind principalul limbaj folosit în acest domeniu.

Pentru a obţine trăsăturile dintr-o imagine am folosit libraria Face Recognition, [18]. Această librărie este disponibilă pe sistemele de operare macOS şi Linux, dar există şi ghiduri pentru instalarea sa pe Windows, deşi acest lucru nu este suportat oficial. Aici, ne sunt oferite funcţii pentru gasirea feţelor din imagini, existând atât o implementare naivă prin HOG+SVM, precum şi una prin reţele convoluţionale, dar aceasta din urmă are nevoie de mai mult timp să dea un răspuns, deşi acurateţea este mai mare. în plus, ne este oferită o funcţie de extragere a 128 trăsături dintr-o faţă.

Extragerea trăsăturilor se realizează folosind o reţea ResNet antrenată pe milioane de imagini şi care are o acurateţe de 99,38% pe setul de date ‘Labeled Faces in the Wild’, [18]. Pentru antrenare a fost folosită o funcţie de cost triplă precum cea propusă în [14]. Astfel, a fost necesar doar să trecem aceşti vectori de trăsături printr-un clasificator SVM/KNN pentru a obţine rezultate foarte bune.

Pentru clasificatorii SVM şi KNN am folosit librăria Scikit-learn din Python. Aceasta este o librărie disponibilă pe toate sistemele de operare care oferă implementări pentru mai multe domenii ale inteligenţei artificiale precum clasificare, regresie etc.

Algoritm

Pasul 1 este reprezentat de citirea setului de date şi creerea celor 2 vectori: datasetX care va conţine trăsăturile şi datasetY care va reţine clasa corespunzătoare

```
for director_clase in os.listdir(director_dataset):
    for locatie_imagine in image_files_in_folder(os.path.join(director_dataset,
director_clase)):
        #incărcăm imaginea
        imagine = face_recognition.load_image_file(locatie_imagine)
        #extragem toate feţele din imagine
        feţe = face_recognition.face_locations(imagine)
        if len(fete) == 1:
            #adăugăm în vector trăsăturile rezultate din reţea
            datasetX.append(face_recognition.face_encodings(imagine,
known_face_locations=feţe)[0])
            datasetY.append(director_clase)
```

La pasul 2 vom împărţi vectorii datasetX şi datasetY în seturi pentru antrenare şi testare. Apoi vom antrena modelul prin funcţia trainSVM.

```
def trainSVM(model_save_path=None):
    # se crează un clasificator SVM cu parametrii daţi
    svm_classifier = svm.SVC(kernel='linear', C = 0.3)
    # se antrenează pe SVM-ul setul de date şi se salveză modelul
    svm_classifier.fit(trainX, trainY)
    if model_save_path is not None:
        with open(model_save_path, 'wb') as f:
            pickle.dump(svm_classifier, f)
    return svm_classifier
```

În final, vom crea funcţia de prezicere şi vom calcula acurateţea.

```
#funcţie care prezice clasa unui vector de trăsături
def predictSVM(faces_encodings):
    return svm_classifier.predict(faces_encodings)[0]
```

```
#calcularea acurateţii
total = 0
nrCorecte = 0
for i in range(len(testX)):
    name = predictSVM([testX[i]])
    total = total + 1
    if name == testY[i]:
        nrCorecte = nrCorecte + 1

print("Corecte KNN: " + str(nrCorecte))
print("Totale KNN: " + str(total))
print("Acc KNN: " + str(nrCorecte/total))
```

În final, am folosit librăria Flask din Python pentru a crea un serviciu REST care va primi de la aplicaţia web locaţia imaginii pe server şi va returna clasa persoanei din imagine, sau şirul vid dacă imaginea conţine mai multe persoane.

2.4. Rezultate şi experimente

Setul de date de care am avut nevoie pentru această aplicaţie a trebuie să fie compus din imagini cu actori, aproximativ 100 imagini pentru fiecare din cei 100 de actori aleşi. Deoarece nu am reuşit să gasesc un astfel de set, am luat decizia să-l construiesc de unul singur, ajungând astfel să testez cât de bine funcţionează clasificarea pe un set nou de imagini. Actorii aleşi au fost luaţi dintr-un top 100 a celor mai populari actori din anul 2017, realizat de platforma IMDB.

Pentru început, a trebuie să găsesc un program care să ma ajute să descarc imagini de pe internet. Am decis să folosesc o extensie de Google Chrome prin care mi se punea la dispoziţie să aleg ce imagini să descarc de pe pagina în care sunt. Prima problemă pe care am avut-o a fost că Google Images îmi imagini în care actorul căutat era între mai multe persoane sau nu era în imagine. După prima fază, din 500 imagini rămâneam cu aproximativ 200.

Aceste imagini erau apoi descărcate şi verificate pentru a elimina duplicatele şi pentru a decupa părţile care nu erau dorite, de exemplu o altă persoană.

În final, am ajuns la un total de 10654 de imagini distribuite uniform în 100 clase. În total, acest proces a durat peste 24 de ore. Consider că, deşi crearea setului de date a fost destul de monotonă, procesul nu este unul deloc dificil.

În continuare, am dorit să aflu cât de bine va reuşi un model format dintr-o reţea preantrenată şi un clasificator simplu să claseze un set nou de date. Setul a fost împărţit astfel: 7654 imagini au fost utilizate în antrenare, iar restul de 3000 au fost folosite pentru a calcula acurateţea programului.

Reţeaua convoluţională aleasă a fost un ResNet antrenat cu o funcţie de cost triplă. Iniţial, am folosit un clasificator SVM liniar şi am obţinut o performanţă de aproximativ 99,66%. Totuşi, prin reducerea parametrului C a clasificatorului la 0.3 am reuşit să obţin o performanţă de 99,7%. Dupa aceea, am verificat şi cu un KNN, acesta reuşit o acurateţe tot de 99,66% pentru k = 1, dar prin modificarea numărului de vecini la 5 am obţinut o acurateţe de 99,73%.

În continuare, am decis să testez şi un algoritm format din histograma gradienţilor orientaţi pentru a extrage atributele feţei şi un clasificator SVM sau KNN. Pentru extragerea histogramei am folosit librăria OpenCV. Aceată librărie este destinată domeniului de viziune a calculatorului, oferind multe implementări de algoritmi elementari.

Folosind clasificatorul SVM şi prin alegerea parametrilor, am ajuns la o acurateţe de 55,25%. Ţinând cont că această metodă a fost introdusă acum mai bine de 10 ani şi că dimensiunea setului de antrenare este una restrânsă, putem trage concluzia că acest model s-a descurcat onorabil, recunoscând corect persoana din imagine cel puţion o dată din două încercări. Totuşi, folosind KNN nu am obţinut un rezultat atât de bun, doar 27,6% pentru numărul de vecini egal cu 11.

În final, putem observa că, prin folosirea unei reţele foarte bine antrenate cu o funcţie de cost triplă, se pot obţine rezultate foarte bune pe un set de date nou cu un număr relativ mic de imagini.

Metodă de extragere a trăsăturilor	Clasificator	Acurateţe
ResNet	SVM	99,7%

ResNet	KNN	99,73%
HOG	SVM	55,25%
HOG	KNN	27,6%

Capitolul 3

3. Dezvoltarea aplicaţiei web

3.1. Tehnologii folosite

Aplicaţia web a fost realizată în **Java**. Java este un limbaj de programare bazat pe clase de nivel înalt făcut de cei de la Oracle în 1995. În prezent, Java a ajuns la versiunea 10, aceasta fiind lansată în Martie 2018.

O diferenţă interesantă intre Java şi C++ este lipsa variabilelor constante. Totuşi, există o modalilate de a obţine o astfel de variabilă folosind cuvintele cheie ‘static final’. Un câmp ‘static’ aparţine clasei şi nu obiectelor, însemnând că variabila constantă va fi

disponibilă fără a crea un obiect. În plus, ‘final’ de obligă să iniţializăm variabila o singură dată. Astfel obţinem o variabilă care va aparţine tuturor obiectelor şi care nu poate fi modificată.

În interiorul unei clase Java pot apărea unele construcţii numite blocuri statice. Acestea sunt folosite pentru a iniţializa variabilele statice şi sunt executate doar la primul obiect creat din acea clasă. Ordinea de executare într-o clasă Java este următoarea: blocul static din supraclasă, blocul static din subclasă, constructorul din supraclasă şi în final constructorul din subclasă.

Fişierele Java au extensia ‘.java’, iar acestea vor fi transformate în fişiere ‘.class’. Acestea din urmă conţin cod Java bytecode, care va fi compilat de maşina virtuală Java. La o singură rulare maşina virtuală Java încarcă mai mult de 20000 clase în memorie.

Java este împărţită în 3 mari componente:

- JavaEE, cuprinzând toate componentele necesare realizării unei aplicaţii desktop: sintaxa, tipurile de date, elementele POO, colecţii, interacţiunea cu baza de date prin JDBC, manipularea fişierelor XML şi JSON, lucrul pe mai multe fire de execuţie, excepţii etc.
- JavaSE, prin care putem crea aplicaţii şi servicii web: HTTP Request/Response, HTTP Session, JSP, Servlet, JSTL, JAX-WS, JAX-RS.
- JavaME, prin care se pot realiza aplicaţii mobile

Aici poate fi precizat şi Android-ul care, deşi este făcut de cei de la Google, are la bază limbajul Java.

Interfaţa Iterable stă la baza erorilor şi excepţiilor limbajului Java. Erorile sunt problemele care apar la maşina virtuală java, acestea rezultând în oprirea execuţiei. Excepţiile sunt fie de tipul celor care trebui trecute în clauza ‘catch’ pentru a oferi o rezolvare în cazul în care apar, fie derivate din clasa RuntimeException (de exemplu NullPointerException).

Maven este un program realizat de Apache prin care dezvolatorii Java pot introduce usor dependinţe în proiectele lor. Aceste ne oferă o serie de comenzi prin care putem: instala pachetele local pentru a le putea folosi în alte proiecte (‘mvn clean’), valida şi compila codul, curăţa artefactele vechi şi lansa aplicaţia pe un server. În plus, proiectul ne este împarţie în ‘main’, unde vom păstra pachetele şi tot ce ţine de aplicaţie şi în ‘test’, unde vor apărea testele unitare. Acestea pot fi scrise în JUnit şi vor fi verificate la fiecare comanda ‘mvn clean install’, aceasta realizând succesiv cele 2 comenzi copil.

Definiţia 3.1: *Testele unitare sunt o serie de verificări realizate de programatori prin care ne asigurăm că o anumită funcţie va executa funcţia dorită iniţial, chiar dacă de-a lungul timpului implementarea acesteia s-a schimbat.*

Testele unitare ne permit să găsim mult mai uşor eventuale erori care pot apărea la anumite modificări ale funcţiilor existente. Se spune că o aplicaţie realizată corect ar trebui să aibă teste unitare pe mai mult de 80% din funcţii.

Spring

Oracle

MyBatis

HTML este un limbaj folosit pentru aranjarea elementelor într-o pagina web. Acesta stă la baza oricărei aplicaţii web, fiind componenta de bază a acestor aplicaţii. Acesta ne propune o mulţine de o mare diversitate de elemente din care putem alege în funcţie de ce avem de realizat. Distribuţia elementelor unui fişier HTML este una ierarhică. Un document HTML are 2 componente principale: cap(head) şi corp(body). Capul este folosit, în principal, pentru a specifica titlul paginii şi pentru a încărca fişierele de stil şi de script. Totuşi, în ultimul timp tot mai multe persoane au început să adauge fişierele de script în interiorul componentei de corp. Acest lucru face ca pagina să se încarce mai repede deoarece compilarea scriptului are loc după ce aceasta s-a afişat. Cu toate acestea, compilarea scriptului la final poate face probleme mari în cazul funcţiilor care se execută dupa încărcarea ferestrei în cazul în care internetul are o viteză mică.

JSP-ul este o tehnologie care ne ajută să creem pagini web dinamice. Acesta ne oferă posibilitatea să scriem cod Java direct în paginile HTML. JSTL este o librărie care ne oferă suport pentru realizarea unor sarcini structurale precum structurile ‘dacă’, ‘cât timp’ şi ‘pentru fiecare’. În plus, ne sunt oferite posibilităţi să formatăm uşor lucruri precum data, timpul şi numere.

Limbajul CSS este folosit pentru stilizarea elementelor HTML. Acesta ne oferă posibilitatea să alegem elementele după: identificatorul unic al elementului, clasele acestora sau tipul de element HTML. În plus, ne este pus la dispoziţie şi posibilitatea de a alege mai multe tipuri de elemente, precum şi de a alege elemente într-un mod ierarhic (de exemplu, toate paragrafele cu clasa X din divul cu identificatorul Y). Există 3 posibilităţi de a adauga stil unei pagini HTML. Prima posibilitate este de a-l adăuga în componenta ‘style’ a tuturor elementelor HTML. Deşi acest lucru nu este recomandat pentru proprietăţile similare ale mai

multor elemente, consider că este util să adaugăm aici acele caracteristici individuale elementului deoarece sunt mult mai uşor de găsit şi modificat decât să cautam acel element după un identificator unic într-un fişier de stilizare. A doua posibilitate de folosite este prin introducerea în componenta ‘head’ al paginii web. Acest lucru ne permite să ne referim la mai multe elemente prin identificatorii de clase sau de tip în modul urmator: ‘.’ este folosit pentru clasă, iar tipul nu este precedat de nimic. Totuşi, nu ne dorim să avem stilul în paginile HTML în procesul de dezvoltare al aplicaţie. De aceea, pornind de la principiul de separare al conceptele, s-a luat decizie de a pune stilul într-un fişier separat şi de al include în HTML printr-un import.

JavaScript este cel mai folosit limbaj de programare. Spre deosebire de limbajele Java, C, C++ care trebuiesc compilate, acesta este un limbaj de interpretare. În compilare, compilatorul compilează tot codul de la început, pe când la interpretare este aleasă prima line, este compilată şi executată, iar apoi este trecut la urmatoarea linie. La fel ca şi CSS-ul, JavaScript-ul poate fi trecut atât în componenta ‘head’ a fişierului, cât şi într-o pagină separată care apoi este importată în HTML pentru a respecta principiul separării conceptelor. Astfel, vom avea 3 fişiere, fiecare cu funcţia sa proprie.

3.2. Descrierea aplicaţiei

Aplicaţia este formată din 4 module care comunică folosind Maven:

- InfoAct. Aici sunt păstrate scripturile bazei de date, diagrama acesteia, o listă cu abrevieri folosită în instrucţiunile SQL pentru tabele şi orice alte fişiere care nu conţin cod.
- InfoActApp. Cea mai importantă funcţie a acestui modul este executarea operaţiunilor pe baza de date. În directorul ‘resources’ oferit de Maven sunt salvate fişierele ‘.xml’ ale MyBatis-ului, aici fiind scrise instrucţiunile de manipulare ale datelor din bază. Pe de altă parte, directorul ‘src’ conţine interfeţele pe care sunt mapate fişierele din MyBatis, validatoarele prin care se verifică dacă informaţia care urmează să fie introdusă în baza de date este una corectă şi serviciile, acestea cuprinzând o serie de funcţii care sunt folosite de aplicaţie.

- InfoActCommon. Aici sunt păstrate obiectele de domeniu şi cele care vor fi folosite în filtrarea rezultatelor după anumite criterii.
- InfoActWeb. Partea principală a aplicaţiei web. Aici sunt păstrate controalele, serviciile Web (realizate folosind Jax-RS), fişierele de stilizarea, JavaScript şi HTML, precum şi fişierele de configurare ale Spring-ului.

  Încărcarea imaginilor pe server a fost realizată folosind un port multipart Spring.

6. Bibliografie

[1] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks*. Proceedings of NIPS, pp. 1106–1114, 2012.

[2] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. *Dropout: a simple way to prevent neural networks from overfitting.* J. Machine Learning Res. 15, 1929–1958, 2014.

[3] Simonyan, K. & Zisserman, A. *Very deep convolutional networks for large-scale image recognition*. In Proc. International Conference on Learning Representations http://arxiv.org/abs/1409.1556, 2014.

[4] Vincent Dumoulin & Francesco Visin. *A guide to convolution arithmetic for deep learning.* arXiv:1603.07285, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385, 2015.

[6] G. Tsai. *Histogram of oriented gradients*. University of Michigan, 2010.

[7] Navneet Dalal, Bill Triggs. *Histograms of Oriented Gradients for Human Detection*. Cordelia Schmid and Stefano Soatto and Carlo Tomasi. International Conference on Computer Vision & Pattern Recognition (CVPR '05), Jun 2005, San Diego, United States. IEEE Computer Society, 1, pp.886–893, 2005, . <10.1109/CVPR.2005.177>.

[8] F. Suard, A. Rakotomamonjy, A. Bensrhair, and A. Broggi. P*edestrian detection using infrared images and histograms of oriented gradients*. In IEEE Intelligent Vehicles Symposium, 2006.

[9] CORINNA CORTES & VLADIMIR VAPNIK. *Support-Vector Networks*. 1995 Kluwer Academic Publishers, Boston, 1995.

[10]

[11] Radu Tudor Ionescu & Marius Popescu. *Knowledge transfer between computer vision and text mining*. *Similarity-based Learning Approaches.* Springer, 2016.

[12] Dasarathy BV. *Nearest neighbor (NN) norms: pattern classification techniques.* IEEE Computer Society Press, Los Alamitos, 1991.

[13] Faragó A, Linder T, Lugosi G. *Fast nearest-neighbor search in dissimilarity spaces*. IEEE Trans Pattern Anal Mach Intell 15(9):957–962, 1993.

[14] Florian Schroff, Dmitry Kalenichenko, James Philbin. *FaceNet: A Unified Embedding for Face Recognition and Clustering*. arXiv:1503.03832, 2015.

[15] https://en.wikipedia.org

[16] https://medium.com/

[17] http://cs231n.stanford.edu/

[18] https://pypi.org/project/face_recognition/

[19] http://www.google.ro