

# **PROIECT DE LICENȚĂ**

**Aplicație web pentru căutarea informațiilor despre actori pe bază de  
recunoaștere facială**

# Cuprins

<b>Introducere</b>	<b>3</b>
<b>Recunoașterea facială din imagini</b>	<b>5</b>
2.1. Concepte teoretice	5
2.1.1. Rețele neuronale convoluționale	5
2.1.2. Histograma gradientilor orientați (HOG)	12
2.1.3. Mașini cu suport vectorial (SVM)	14
2.1.4. Metoda celor mai apropiați k vecini (KNN)	18
2.2. Metode recente pentru recunoașterea facială din imagini	19
2.2.1. FaceNet	19
2.2.2. ResNet	20
2.3. Algoritmul de recunoaștere facială implementat	21
2.4. Rezultate și experimente	24
<b>Dezvoltarea aplicației web</b>	<b>26</b>

## **Capitolul 1**

### **Introducere**

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

În această lucrare vom folosi o rețea neuronală convoluțională pentru a clasifica un set de date creat specific pentru această aplicație, el conținând un număr de aproximativ 10600 imagini, grupate în 100 categorii, reprezentând cei mai faimoși actori în urma căutărilor pe platforma IMDB din anul 2017.

Rețelele neuronale convoluționale au avut un succes imens în ultima jumătate de deceniu, totul pornind de la rețeaua propusă de Alex Krizhevsky, [1].

Astfel, în capitolul 2 vom prezenta conceptele teoretice care stau la baza realizării recunoașterii faciale din imagini, vom discuta mai pe larg în ce constau cele 2 rețele propuse de Facebook și Google, iar apoi vom preciza rezultatele obținute folosind o rețea preantrenată pe setul de date al nostru, precum și o comparație cu un algoritm de bază.

În capitolul 3 prezentăm tehnologiile folosite în realizarea aplicației web,

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



## Capitolul 2

### Recunoașterea facială din imagini

#### 2.1. Concepte teoretice

##### 2.1.1. Rețele neuronale convoluționale

Rețelele neuronale convoluționale sunt o categorie a rețelilor neuronale care s-au dovedit să aibă un succes foarte mare în recunoașterea și clasificarea fețelor și a obiectelor din imagini. Față de rețelele normale, cele convoluționale au avut un succes mai mare în aceste domenii datorită faptului că spațialitatea imaginii este păstrată prin folosirea filtrelor.

Rețelele convoluționale sunt foarte asemănătoare cu cele neuronale ordinare prin faptul că sunt compuse din neuroni în care sunt reținute informații. Fiecare neuron primește ca date de intrare un vector de numere, realizează o operație de înmulțire cu vectorul de ponderi și apoi adună un parametru numit bias. Formula arată în felul următor:

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

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

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

De-a lungul anilor au fost folosite mai multe funcții neliniare, denumite și funcții de activare, în acest domeniu. Prima funcție pe care o vom prezenta este funcția **Sigmoid**, notată  $\sigma$ . Aceasta are următoarea formulă matematică și are următoarea distribuție în jurul punctului 0.

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

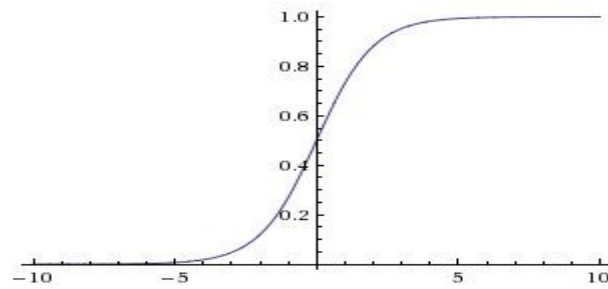


Figura 2.1: Reprezentarea grafică a funcției sigmoid

Funcția sigmoid a fost folosită în trecut datorită faptului că reușeste să restrângă variabila în intervalul  $[0, 1]$ . În practică, această funcție este folosită foarte rar deoarece are câteva mari dezavantaje. Primul problemă este aceea că gradientul în jurul punctelor 0 și 1 este aproape 0. Astfel, în propagarea înapoi a rețelei, semnalul care v-a trece prin neuron va fi foarte mic și nu vor apărea schimbări prea mari în ponderile acestuia, rețeaua învățând foarte greu. De aceea, este necesar să fim foarte atenți când inițializăm ponderile neuronilor care folosesc sigmoid pentru a evita saturarea.

Un alt dezavantaj este acela că rezultatele funcției sigmoid nu sunt centrate în 0. Acest lucru nu este dorit datorită faptului că dacă toate datele de intrare ale unui neuron sunt pozitive, atunci gradientul ponderilor în pasul de propagare înapoi va avea toate valorile ori pozitive, ori negative, depinzând de gradientul total al funcției. În plus, funcția exponențială este scumpă din punct de vedere computațional.

Funcția **Tangentă** este o funcție de activare care restrânge valorile în intervalul  $[-1, 1]$ . Precum funcția sigmoid, tangenta saturează și este computațional scumpă, dar valorile rezultate sunt centrate în 0. De aceea, tangenta este mereu preferată față de sigmoid în practică. Un lucru interesant care poate fi observat este că tangenta poate fi formată dintr-o funcție sigmoid:

$$\tanh(x) = 2 * \sigma(2 * x) - 1 \quad (2.4)$$

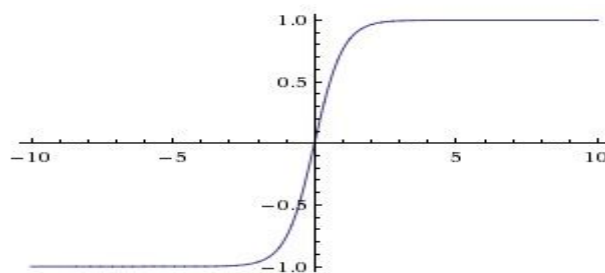


Figura 2.2: Reprezentarea grafică a funcției tangente

În continuare vom prezenta funcția de activare **ReLU**. Aceasta a fost propusă pentru a fi folosită în rețelele neuronale de către Alex Krizhevsky în [1]. În acea lucrare ne este indicat să folosim ReLU în detrimentul tangentei, acuratețea rămânând aceeași, deși funcția este de 6 ori mai rapidă. Această are următoarea formulă:

$$f(x) = \max(0, x) \quad (2.5)$$

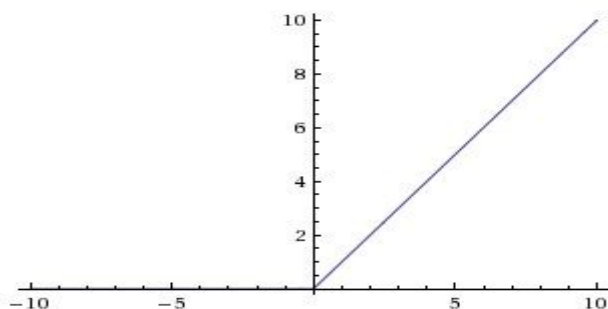


Figura 2.3: Reprezentarea grafică a funcției ReLU

O mare problemă care poate apărea în antrenarea rețelelor neuronale este cazul în care rețeaua învață 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 și 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 și doar ei vor fi actualizați la propagarea înapoi, restul fiind setați să aibă valoarea 0 în acea rundă.

Mai mulți neuroni interconectați care conțin o funcție de activare formează un strat. O rețea neuronală este compusă din mai multe astfel de straturi complet conectate între ele și un strat final care ne va returna răspunsul.

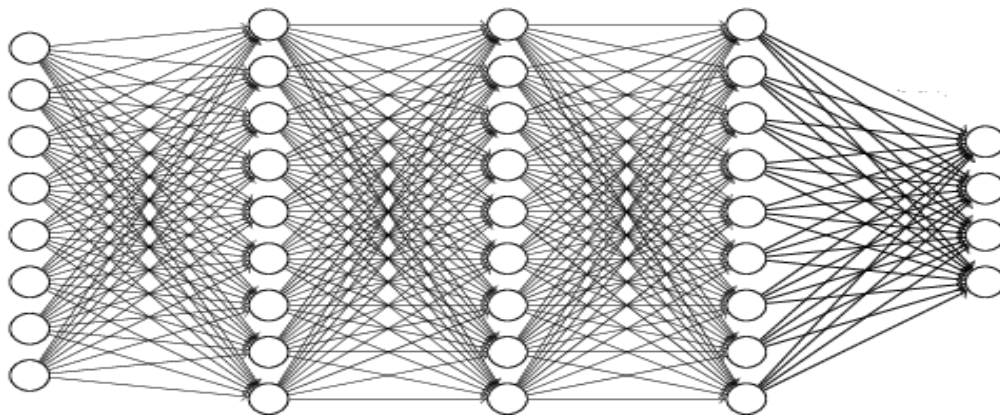


Figura 2.4: Reprezentarea grafică a unei rețele neuronale

Ultimul strat al unei rețele este unul de decizie. Aici va avea loc prezicerea asupra cărei clase aparțin datele de intrare. Cea mai folosită funcție pentru a efectua acest lucru este **Softmax**. Ea primește la intrare un vector de dimensiune  $n$  de numere reale și returnează un vector de numere reale între 0 și 1, având suma totală egală cu 1. Astfel, valoarea vectorului de ieșire poate fi gândită ca 'probabilitatea ca un element să aparțină acelei clase'. Funcția softmax are următoarea formulă:

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

În final vom discuta despre funcția de cost. Această funcție ne spune cât de bine am aproximat rezultatul final. Astfel, putem vedea cât de aproape am fost de răspunsul corect și putem modifica valorile neuronilor. Rezultatul acestei funcții este numită eroarea aceluia pas. Cea mai cunoscută funcție de cost este aceea în care calculăm pătratul diferenței dintre ce am prezis noi și răspunsul corect pentru fiecare clasă, iar apoi le adunăm pe toate pentru a obține eroarea totală. În funcție de tipul problemei pe care încercăm să o rezolvăm, pot apărea diferite funcții de cost.

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

În acest punct avem toate componentele necesare pentru a construi o rețea neuronală ordinară. Toate aceste lucruri vor apărea și în rețelele neuronale convoluționale.



Principala diferență între o rețea convoluțională și una ordinară este faptul că știm că la intrare vom primi o imagine. Fiecare imagine poate fi gândită ca o matrice de numere cuprinse între 0 și 255. Un canal este termenul folosit pentru a ne referi la o anumită componentă a unei imagini. O imagine color are 3 canale (roșu, verde și 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 și 255, 0 reprezentând negru și 255 alb.

O rețea convoluțională este formată dintr-o înșirare de straturi. Totuși, spre deosebire de rețelele ordinare care folosesc doar straturi complet conectate între ele (fig. 2.4), cele convoluționale mai introduc 2 tipuri de straturi: convoluționale și de unificare.

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

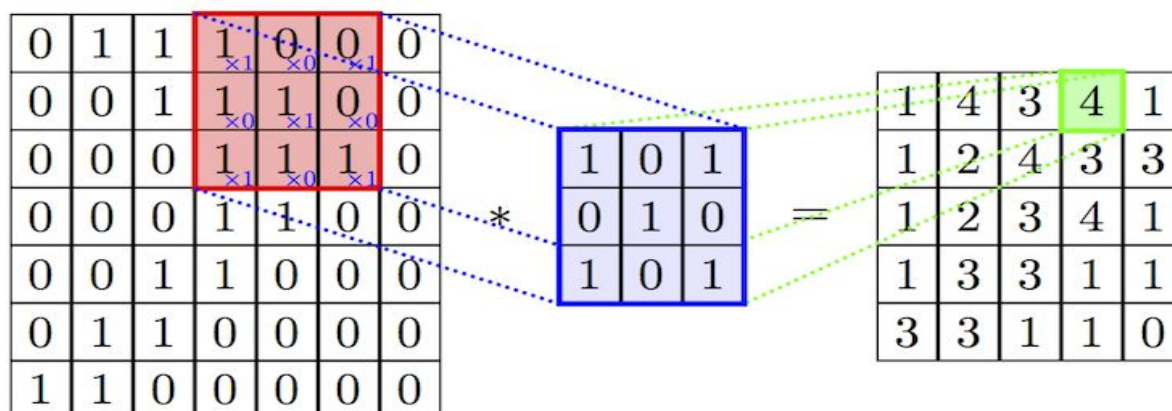


Figura 2.5: Reprezentarea grafică a unui pas de convoluției

Pentru fiecare strat convoluțional trebuie să decidem ce dimensiune va avea filtrul, peste câți pixeli să sara la fiecare pas și de unde să înceapă. Conform [3], filtrele ar trebui să aibă dimensiunea de 3x3, aceasta fiind cea mai mică dimensiune care păstrează noțiunile de stânga-dreapta, sus-jos și centru. 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 funcția de decizie să fie mai discriminativă. În al doilea rând, campul vizual al unui strat

convoluțional 7x7 este același precum cel al 3 convoluții consecutive 3x3. Nu în ultimul rând, un strat de dimensiune 7x7 ar avea nevoie de  $7 \times 7 \times D \times D$  parametri, D reprezentând numărul de canale, pe când cele 3 straturi 3x3 ar avea nevoie de doar  $3 \times (3 \times 3 \times D \times D)$ . Astfel, stratul 7x7 ar avea nevoie de 81% mai mulți parametri decât cele de 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. Creșterea pasului de deplasare va genera o matrice de trăsături de dimensiune mai mică.

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

Când construim o rețea convoluțională, trebuie să avem grijă la dimensiunea ferestrei care va rezulta după un strat convoluțional. Dimensiunea matricei de trăsături rezultate în urma stratului de convoluție poate fi calculată astfel:

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

, unde D reprezintă dimensiunea matricei de trăsături, W dimensiunea înălțimea/lățimea 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 informație. Conform [4], cea mai folosită metodă de unificare este cea a maximului. Prin această procedură, matricea de trăsături este împărțită în zone nesuprapuse și este extras maximul din fiecare petic.

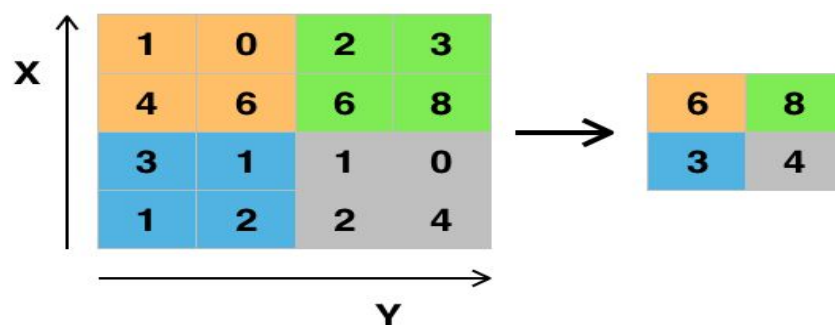


Figura 2.6: Strat de unificare prin extragerea maximului

Există și alte metode de unificare, precum calcularea mediei sau a sumei. La fel ca în convoluții, la straturile de unificare trebuie să avem grijă la dimensiunea matricei de trăsături rezultate. În [4] ne este dată formula care ne returnează dimensiunea matricei finale, aceasta fiind aceeași 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 al celei 2.7, atunci când  $P = 0$ . Acest lucru are sens, deoarece în straturile de unificare nu sunt adăugate 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 și la final unul sau mai multe straturi complet conectate între ele, încheiate cu un strat de softmax pentru a ne rezulta probabilitățile de apartenență pentru fiecare clasă.

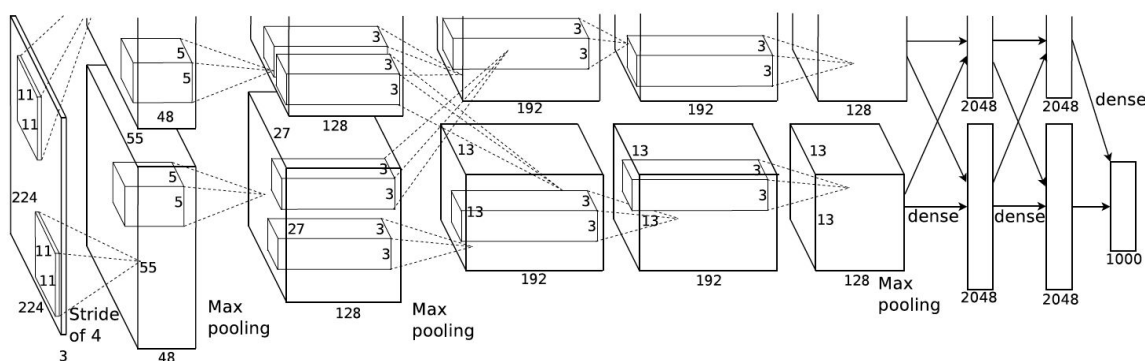


Figura 2.7: Rețeaua AlexNet, propusă de Alex Krizhevsky în [1]

Alex Krizhevsky, Ilya Sutskever și Geoffrey Hinton au șocat lumea în 2012, când au prezentat o rețea neuronală convoluțională care a câștigat concursul ILSVRC, obținând o eroare de doar 15.4% pe primele 5 predicții, față de precedentă 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 rețeaua ResNet, prezentată în [5].

### 2.1.2. Histograma gradientilor orientați (HOG)

Histograma gradientilor orientați a fost folosită pentru a extrage trăsăturile unui obiect din imagini. Spre deosebire de o rețea neuronală convoluțională care învață de una singură cum să extragă trăsături, această metodă are un algoritm predefinit pentru a calcula aceste date de ieșire. Motivul pentru care, în general, rețelele convoluționale funcționează mai bine decât descriptorul HOG este acela că rețelele învață cum trebuie să extragă trăsăturile pentru a obține o acuratețe cât mai bună pe un anumit set de date, pe când descriptorul HOG are același mod de extragere a trăsăturilor, indiferent de date. De aceea, când avem un set de date cu multe imagini, este de preferat să antrenăm o rețea deoarece aceasta va avea acuratețe mai bună aproape de fiecare dată. Totuși, dacă avem un set de date mic și nu avem posibilitatea să creștem numărul de imagini, este posibil ca metoda histogramei gradientilor orientați să ofere o acuratețe mai bună.

Principala idee din spatele histogramei gradientilor orientați este aceea că aspectul și forma locală a unui obiect poate fi descrisă de distribuția intensității gradientilor sau de orientarea marginilor, [6]. Imaginea este împărțită în regiuni conectate numite celule, iar pixelii din fiecare celulă vor determina o histogramă a direcției gradientilor. Descriptorul este o concatenare a acestor histogramme.

**Definiția 2.1:** *Gradientul unei imagini este schimbarea direcției în intensitate sau culoare dintr-o imagine.*

Detectarea muchiilor dintr-o imagine este una dintre cele mai folosite aplicații ale gradientilor.

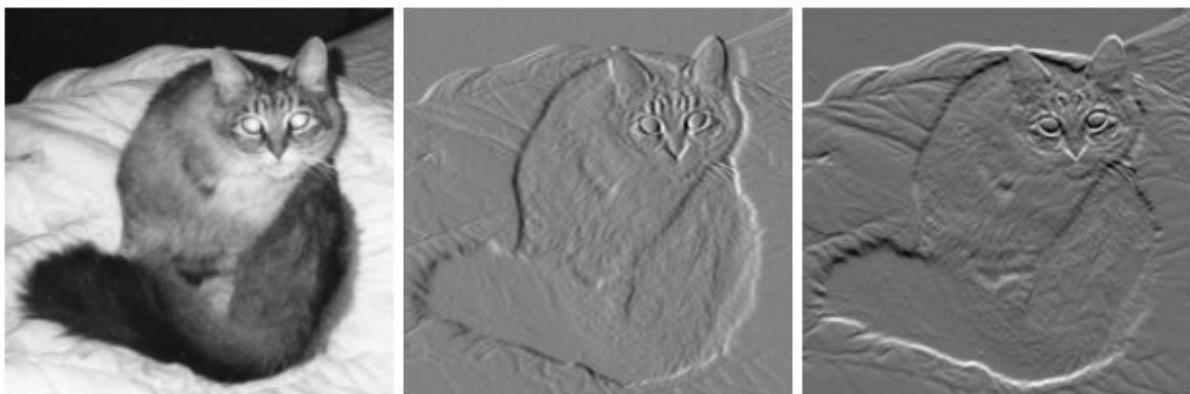


Figura 2.8: Gradientii unei imagini

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

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

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

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

Direcția gradientului este dată de:

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

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

$$G = \sqrt{g_x^2 + g_y^2} \quad (2.11)$$

Una dintre cele mai simple metode de a găsi gradientii unei imagini este prin folosirea unui filtru Sobel. Pentru găsirea gradientului pe direcția x, filtrul Sobel va parcurge imaginea precum un filtru dintr-un strat convoluțional al unei rețele neuronale convoluționale. Pentru găsirea gradientului pe direcția y, vor aplica aceeași metoda, dar vom folosi transpusa filtrului Sobel. Pentru a obține gradientii pe ambele direcții, este suficient să trecem cu primul filtru pe imaginea inițială, iar apoi cu al doilea filtru pe imaginea rezultată după aplicarea primului filtru. Imaginea finală va fi folosită în construirea histogramei gradientilor orientați. În acest pas este calculată direcția și magnitudinea gradientilor.

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

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

Figura 2.9: Componentele filtrului Sobel pe direcția x

Alte filtre pot fi folosite pentru a obține gradientii unei imagini. În [7], histograma gradientilor orientați a fost folosită pentru a detecta pietonii dintr-o intersecție. Aici, filtrul care a dat cele mai bune rezultate a fost unul de dimensiune  $1 \times 3$ , având valorile  $[-1, 0, 1]$ .

După aflarea gradientilor și împărțirea imaginii pe celule, urmează să construim histograma în fiecare dintre aceste zone. Matricea direcției gradientilor va conține valori cuprinse între 0 și 359, reprezentând măsura unghiului dintre direcția gradientului și axa OX în grade. Astfel, histograma va împărți aceste unghiuri în porțiuni, iar fiecare pixel își va aduna valoarea magnitudinii în componenta corespunzătoare direcției sale. După ce toți gradientii pixelilor unei celule au fost parcurși, histograma acelei celule va fi completă. Histograma gradientilor orientați ale unei imagini rezultă din concatenarea histogramelor tuturor celulelor acelei imagini.

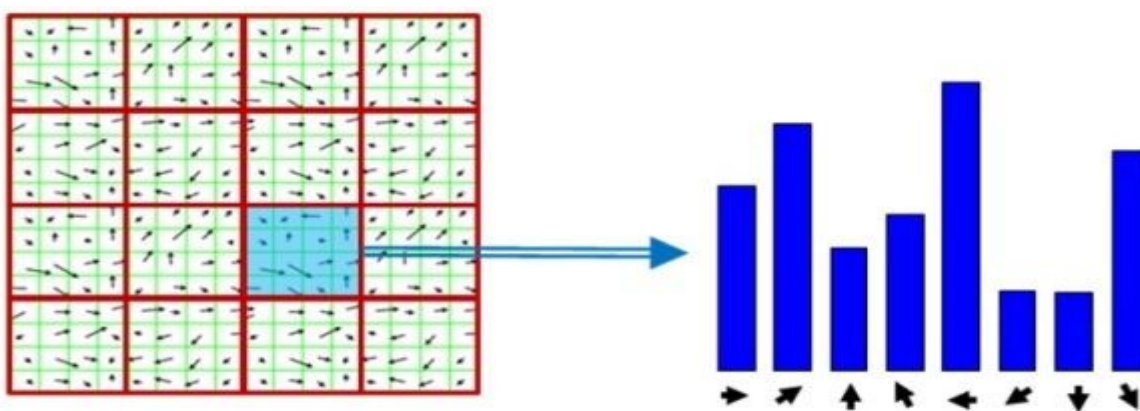


Figura 2.9: Reprezentarea grafică a histogramei unei celule

### 2.1.3. Mașini cu suport vectorial (SVM)

Mașinile cu suport vectorial sunt modele de învățare supervizată folosite în clasificare și regresie. Acestea mapează parametrii de intrare într-un spațiu de dimensiune foarte mare neliniar. În acest spațiu, o suprafață liniară de decizie este construită. Proprietățile speciale ale suprafeței de decizie ne asigură o generalizare mare a mașinilor de învățare.

**Definiția 2.2:** *Învățarea supervizată în inteligența artificială este problema învățării unei funcții care mapează datele de intrare pe unele de ieșire plecând de la mai multe exemple de intrare-ieșire.*

SVM-ul este printre cele mai bune( și mulți cred că este cel mai bun) algoritmi de învățare supervizată standard, [11]. Acest clasificator definește un hiperplan pentru a separa clasele diferite.

**Definiția 2.3:** *În geometrie, un hiperplan este un subspațiu a cărui dimensiune este mai mică cu 1 decât a spațiului ambiental.*

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

**Definiția 2.4:** *Spațiul ambiental este spațiul care înconjură un obiect.*

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

Un model de mașină cu suport vectorial este o reprezentare a datelor ca puncte în spațiu, fiind mapate astfel încât cele aparținând categoriilor diferite să fie despărțite de un gol clar care este cât mai mare posibil. Pe lângă faptul că efectuează clasificări liniare, mașinile cu suport vectorial pot executa clasificări neliniare folosind ceea ce se numește trucul nucleului, mapând implicit datele sale de intrare într-un spațiu al trăsăturilor cu dimensiune foarte mare. wiki

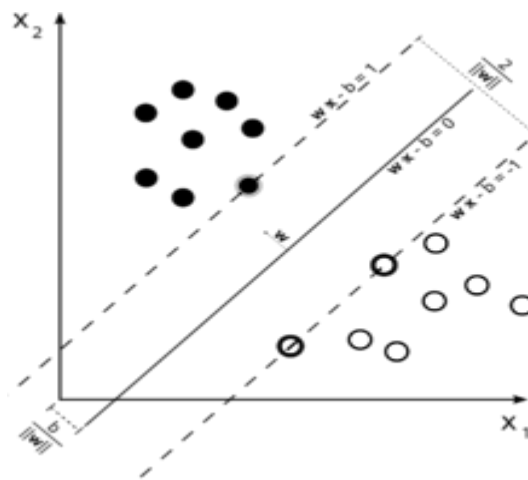


Figura 2.10: Reprezentarea grafică a unui SVM liniar

Dreptele cu linie punctată se numesc vectori suport. Deși există multe linii care pot separa cele două categorii din figura (2.10), hiperplanul optim este cel aflat la mijlocul distanței dintre cei 2 vectori suport și paralel cu aceștia.

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

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

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

, 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. Vrem să aflăm hiperplanul care împarte grupul punctelor pentru care  $y = -1$  de grupul punctelor cu  $y = 1$  astfel încât distanța dintre hiperplanul ales și cel mai apropiat punct  $x$  din oricare dintre cele 2 grupe este maximă.

Orice hiperplan poate fi scris ca un set de puncte satisfăcând următoarea relație:

$$\vec{w} \cdot \vec{x} - b = 0, \quad (2.12)$$

, unde  $w$  este vectorul normal al hiperplanului.

**Definiția 2.5:** În geometrie, vectorul normal este vectorul perpendicular pe un obiect dat.

Construirea modelului SVM este influențată de 2 parametri:

- parametrul de **Regularizare** (notat cu  $C$  în biblioteca Scikit-Learn pentru Python) spune clasificatorului cât de mult vrei să te ferești de clasificarea greșită a fiecărei date de antrenare. Pentru valori mari ale lui  $C$ , optimizatorul va alege un hiperplan de dimensiune mai mică dacă acesta va face o treabă mai bună în a clasifica punctele de antrenare corect. 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 greșit câteva puncte. 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 funcție de fiecare set de date. medium
- parametrul **Gamma** definește care puncte influențează crearea hiperbolei de separare a claselor. Un gamma mic va face ca și punctele aflate la o distanță mai mare de posibila hiperbolă să fie luate în vedere în calcularea soluției. Pe de altă parte, un gamma mare va spune să fie folosite doar punctele din apropiere. medium



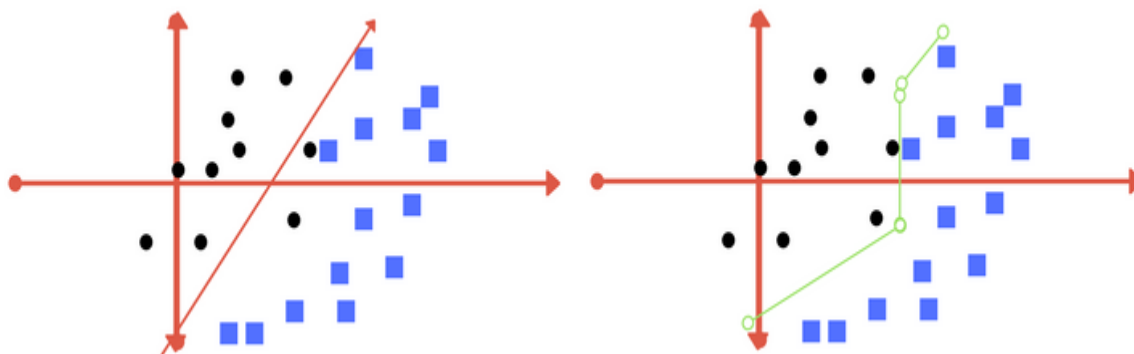


Figura 2.11: În stânga SVM-ul are valoarea parametrului de regularizare mică.

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

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

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

**Definiția 2.7:** *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 predicții avem mai mare încredere că sunt bune. Acest lucru va face clasificatorul să separe exemplele pozitive de cele negative eficient.

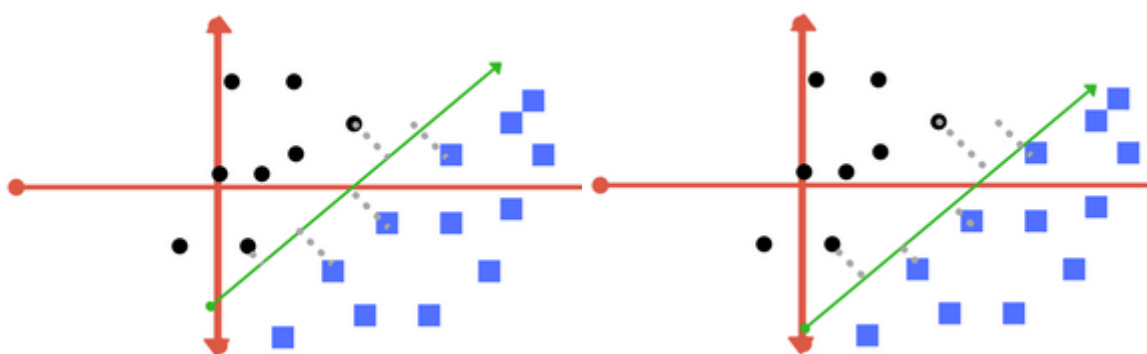


Figura 2.12: În stânga marginea este bună.

În dreapta marginea nu este aleasă bine.

Marginea din imaginea stângă este bună deoarece aceasta a fost aleasă cât de echidistant 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 clasei care conține pătratele albastre.

#### 2.1.4. Metoda celor mai apropiați k vecini (KNN)

Deși a fost introdusă acum mai bine de 60 de ani, metoda celor mai apropiați k vecini este un topic activ în inteligența artificială, susținând faptul că metodele simple sunt mereu atractive, [11]. Acest algoritm este folosit pentru clasificare și regresie.

În construirea clasificatorului KNN avem nevoie de 3 componente:

- setul de date pentru antrenare cu clasele corespunzătoare
- metoda prin care se va calcula distanța între 2 înregistrări. Cea mai folosită metrică în inteligența artificială este distanța Euclidiană, aceasta având următoarea formulă pentru 2 puncte  $p = (p_1, p_2, \dots, p_n)$  și  $q = (q_1, q_2, \dots, q_n)$

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

- variabila k, reprezentând numărul de vecini care sunt considerați în prezicere. Această variabilă este foarte importantă în vederea acurateții pe care o vom obține de la clasificatorul KNN. Dacă valoarea este prea mică, atunci acesta poate avea probleme să clasifice corect setul de testare. În plus, alegerea unui k mic face ca modelul să fie foarte sensibil la presupuse date de intrare eronate. Pe de altă parte, dacă alegem un k prea mare putem ajunge la situația în care decizie să fie luată de vecinii aflați la o distanță mare față de punctul curent, aceștia fiind irelevanți pentru etapa curentă.

##### Algoritm

Fie x un punct pentru care trebuie să prezicem clasa în care aparține. Inițial, vom calcula distanțele de la x la toate celelalte puncte din setul de antrenare și vom sorta vectorul rezultat crescător. Dacă  $k = 1$ , atunci clasa lui x va fi aceeași 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 distanțele și să afle care sunt primele k dintre acestea. Totuși, alți algoritmi mai rapizi au fost

propuși de Belur Dasarathy în [12], bazat pe arbori de căutare multidimensionali care partiționează spațiul și ghidează cautarea, precum și de Farago în [13].

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  $k = 5$ .

În partea stângă avem setul de date colorat diferit în funcție de clasa de care aparține. Îmaginea din centru ne prezintă limitele de decizie ale clasificatorului KNN cu  $k = 1$  folosind distanța 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 roșie, 2 au clasa albastră și ultimul are clasa verde).

## 2.2. Metode recente pentru recunoașterea facială din imagini

### 2.2.1. FaceNet

Rețeaua convoluțională FaceNet a fost propusă de cei de la Google în 2015. Spre deosebire de rețelele precedente care pentru a calcula acuratețea pe un set de date nou care antrenau un strat clasificator și apoi foloseau încă un start intermediar pentru a generaliza clasificarea, aceștia s-au gândit la o modalitate total diferită.

Conform [14], ei au construit o rețea care pentru o imagine trimisă prin rețea să returneze un vector de 128 de numere reale. Vectorul rezultat din rețea are următoarea proprietate: distanța euclidiană dintre el și vectorul realizat dintr-o altă imagine a aceeași persoane este mai mică decât distanța față de vectorul rezultat din imaginea unei persoane diferite. Astfel, imaginile vor fi împărțite în grupuri într-un spațiu de dimensiunea 128. Acest lucru ne permite să antrenăm un clasificator elementar precum SVM sau KNN și să obținem rezultate la fel de bune ca în alte rețele.

Spre deosebire de rețeaua FaceNet care returnează doar 128 octeți, celelalte rețele, pornind de la AlexNet în 2012 [1] și ajungând la VGG în 2014 [3], au avut ultimul strat complet interconectat de dimensiune 1000.

Rețeaua FaceNet a fost antrenată folosind o funcție de cost triplă. La fiecare pas, rețelei i-a fost dat ca date de intrare 2 imagini ale aceleiași persoane și una a altei persoane. Spre deosebire de rețelele precedente a căror funcții de cost le făceau să învețe să clasifice corect persoanele din imagini, funcția de cost triplă are rolul de a face ca vectorul rezultat să aproprie imaginile ale aceleiași persoane prin distanța euclidiană. Formula funcției de cost triplă este următoarea:

$$\sum_i^N \left[ \|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \quad (2.14)$$



Figura 2.14: Antrenarea în ResNet

După cum se poate observa în figura 2.14, în rețea sunt introduse 3 imagini. Funcția de cost triplă încearcă să apropie imaginea ancoră de imaginea pozitivă, adică cea a aceleiași persoane, și să o depărteze de cea negativă.

### 2.2.2. ResNet

Pornind de la rețeaua AlexNet care avea doar 8 straturi și ajungându-se la VGG, aceasta având 16 straturi, s-a pus problema a numărului optim de straturi pe care trebuie să le aibă o rețea. Totuși, în [5] s-a arătat că pentru a scoate o acuratețe mai bună nu este suficient doar să antrenăm o rețea cu un număr mare de straturi. Aceștia au obținut o eroare pe setul de antrenare mai mare pentru o rețea cu 56 straturi decât pe una cu 20.

Explicația acestui fenomen este următoarea: Deoarece există un număr mare de straturi în acea rețea, gradientul în pasul de propagare înapoi în primele straturi o să aibă o valoare foarte mică, posibil să ajungă chiar la 0, și rețeaua nu va reuși să învețe.

Pentru a evita această situație, autorii articolului [5] au propus o învățare reziduală. Ei au presupus să introducă o funcție de identitate, numită start rezidual, care va sări unul sau

mai multe staturi. Rețeaua în care au fost introduse aceste staturi reziduale nu ar trebui să producă o acuratețe mai slabă decât una fără acestea deoarece am putea folosi doar staturile inițiale și astfel prima rețea ar fi identică cu cea de-a doua, rezultând o acuratețe egală.

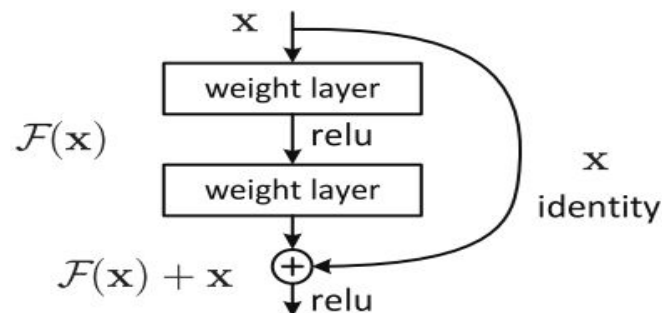


Figura 2.15: Reprezentarea unui strat rezidual

Prin acest procedeu, Microsoft a reușit să construiască o rețea care de 152 staturi prin care a câștigat ILSVRC 2015, obținând o eroare pe primele 5 predicții de 3,56%.

### 2.3. Algoritmul de recunoaștere facială implementat

În implementarea algoritmului de recunoaștere facială am folosit o rețea preantrenată ResNet pentru a extrage trăsăturile din imagini și 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 rețea preantrenată și nu am antrenat una de început este acela că setul de date este foarte restrâns, conținând doar 10600 imagini și fiind împărțit în 100 clase, iar antrenarea unei noi rețele pe un astfel de set ar produce o acuratețe mai mică decât folosirea uneia deja antrenate.

Algoritmul de recunoaștere 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 inteligenței artificiale, fiind principalul limbaj folosit în acest domeniu.

Pentru a obține trăsăturile dintr-o imagine am folosit biblioteca Face Recognition, [18]. Această librărie este disponibilă pe sistemele de operare macOS și Linux, dar există și ghiduri pentru instalarea sa pe Windows, deși acest lucru nu este suportat oficial. Aici, ne sunt oferite funcții pentru găsirea fețelor din imagini, existând atât o implementare naivă prin HOG+SVM, precum și una prin rețele convoluționale, dar aceasta din urmă are nevoie de mai

mult timp să dea un răspuns, deși acuratețea este mai mare. În plus, ne este oferită o funcție de extragere a 128 trăsături dintr-o față.

Extragerea trăsăturilor se realizează folosind o rețea ResNet antrenată pe milioane de imagini și care are o acuratețe de 99,38% pe setul de date 'Labeled Faces in the Wild', [18]. Pentru antrenare a fost folosită o funcție de cost triplă precum cea propusă în [14]. Astfel, a fost necesar doar să trecem acești vectori de trăsături printr-un clasificator SVM/KNN pentru a obține rezultate foarte bune.

Pentru clasificatorii SVM și 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 inteligenței artificiale precum clasificare, regresie etc.

### **Algoritm**

Pasul 1 este reprezentat de citirea setului de date și creerea celor 2 vectori: datasetX care va conține trăsăturile și datasetY care va reține 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)):
        #încărcăm imaginea
        imagine = face_recognition.load_image_file(locatie_imagine)
        #extragem toate fețele din imagine
        fețe = face_recognition.face_locations(imagine)
        if len(fețe) == 1:
            #adăugăm în vector trăsăturile rezultate din rețea
            datasetX.append(face_recognition.face_encodings(imagine,
known_face_locations=fețe)[0])
            datasetY.append(director_clase)
```

La pasul 2 vom împărți vectorii datasetX și datasetY în seturi pentru antrenare și testare. Apoi vom antrena modelul prin funcția trainSVM.

```
def trainSVM(model_save_path=None):
    # se crează un clasificator SVM cu parametrii dați
```

```

svm_classifier = svm.SVC(kernel='linear', C = 0.3)
# se antrenează pe SVM-ul setul de date și se salvează 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 funcția de prezicere și vom calcula acuratețea.

```

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

```

```

#calcularea acurateții
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 aplicația web locația imaginii pe server și va returna clasa persoanei din imagine, sau șirul vid dacă imaginea conține mai multe persoane.

## 2.4. Rezultate și experimente

Setul de date de care am avut nevoie pentru această aplicație a trebuie să fie compus din imagini cu actori, aproximativ 100 imagini pentru fiecare din cei 100 de actori aleși. Deoarece nu am reușit să gasesc un astfel de set, am luat decizia să-l construiesc de unul singur, ajungând astfel să testez cât de bine funcționează clasificarea pe un set nou de imagini. Actorii aleși au fost luați 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 mă ajute să descarc imagini de pe internet. Am decis să folosesc o extensie de Google Chrome prin care mi se pune la dispoziție 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 și verificate pentru a elimina duplicatele și pentru a decupa părțile 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, deși crearea setului de date a fost destul de monotona, procesul nu este unul deloc dificil.

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

Rețeaua convoluțională aleasă a fost un ResNet antrenat cu o funcție de cost triplă. Inițial, am folosit un clasificator SVM liniar și am obținut o performanță de aproximativ 99,66%. Totuși, prin reducerea parametrului C a clasificatorului la 0.3 am reușit să obțin o performanță de 99,7%. După aceea, am verificat și cu un KNN, acesta reușit o acuratețe tot de 99,66% pentru  $k = 1$ , dar prin modificarea numărului de vecini la 5 am obținut o acuratețe de 99,73%.

În continuare, am decis să testez și un algoritm format din histograma gradientilor orientați pentru a extrage attributele feței și un clasificator SVM sau KNN. Pentru extragerea



histogramei am folosit librăria OpenCV. Această librărie este destinată domeniului de viziune a calculatorului, oferind multe implementări de algoritmi elementari.

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

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

Metodă de extragere a trăsăturilor	Clasificator	Acuratețe
ResNet	SVM	99,7%
ResNet	KNN	99,73%
HOG	SVM	55,25%
HOG	KNN	27,6%

## **Capitolul 3**

### **Dezvoltarea aplicației web**

## 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. *Pedestrian 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/](https://pypi.org/project/face_recognition/)