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

Reţelele neuronale convoluţionale au avut un succes imens în ultima jumătate de deceniu, totul pornind de la reţeaua propusă în 2012 de Alex Krizhevsky, [1]. Deşi prima versiune a reţelelor convoluţionale a apărut înaintea anilor 2000, momentul respectiv a adus în atenţie potenţialul acestora şi a făcut marile companii să investească în acest domeniu. Creşterea a fost una foarte rapidă, lucru văzut prin reuşitele obţinute de aceste reţele an de an în domeniul recunoaşterii în imagini.

În ciuda faptului că inteligenţa artificială s-a dezvoltat într-un pas foarte alert în ultimii ani, nu foarte multe persoane au adăugat componente din acest domeniu în aplicaţiile web. În momentul de faţă dezvoltatorilor le sunt puse la dispoziţie multe surse de informaţii şi modalităţi de a utiliza aceste reţele.

Motivaţia acestei lucrări este de a afla ce acurateţe poate avea un model creat de noi şi de a le da posibilitatea unor persoane să folosească această funcţionalitate de recunoaştere facială pentru aflarea informaţiilor despre actori. Pentru a realiza acest lucru vom folosi o reţea neuronală convoluţională prin care vom 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 din anul 2017 în urma căutărilor pe platforma IMDB.

Astfel, în capitolul 2 vom prezenta conceptele teoretice care stau la baza realizării recunoaşterii faciale din imagini, precizând principalele caracteristici ale reţelelor convoluţionale. În continuare vom face o descriere a descriptorului HOG, acesta fiind folosit pentru a arăta diferenţele de acurateţe între reţele şi alte modele de recunoaştere facială, iar apoi vom prezenta 2 dintre cei mai folosiţi clasificatori folosiţi în inteligenţa artificială.

După aceea vom arăta particularităţile propuse de Google şi Microsoft în două dintre cele mai recente reţele apărute în acest domeniu: FaceNet şi ResNet. Capitolul se va încheia cu prezentarea algoritmului folosit pentru recunoaşterea facială în aplicaţia web şi enunţarea

rezultatelor obţinute pe setul de date nou folosind reţeaua preantrenată şi descriptorul HOG: ResNet & SVM, ResNet & KNN, HOG & SVM şi HOG & KNN.

Capitolul 3 ne oferă o descriere a tehnologiilor folosite în realizarea aplicaţiei web. Pe lângă enumerarea lor, ne sunt prezentate şi particularităţi ale acestora, inclusiv scopul pentru care au fost folosite în aplicaţie.

Capitolul 4 este dedicat descrierii aplicaţiei. Aici vom discuta cum funcţionează aplicaţia, vom arăta ce poate să facă fiecare utilizator în funcţie de rolul său, măsurile luate în vederea protecţiei parolelor împotriva unor atacatori sunt prezentate, precum şi modul în care este verificat dacă un utilizator are dreptul să execute o anumită acţiune.

Capitolul 5 ne prezintă concluziile în vederea numărului de poze necesare pentru fiecare actor astfel încât să obţinem o acurateţe bună, dar şi o listă de posibile dezvoltări ulterioare.

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 rezultate mai bune datorită faptului că spaţialitatea imaginii este păstrată prin folosirea filtrelor în straturile convoluţionale.

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 de stocare, acesta actualizându-se prin înmulţirea unor ponderi cu datele de intrare şi adunarea rezultatului cu o valoare b. Formula arată în felul următor:

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

De preferat, această operaţie ar trebui să fie urmată de o funcţie neliniară. Acest lucru este dorit datorită 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. Fiecare funcţie liniară trebuie urmată de una neliniară pentru a evita cazul precizat. În final formula va arată astfel:

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

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

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

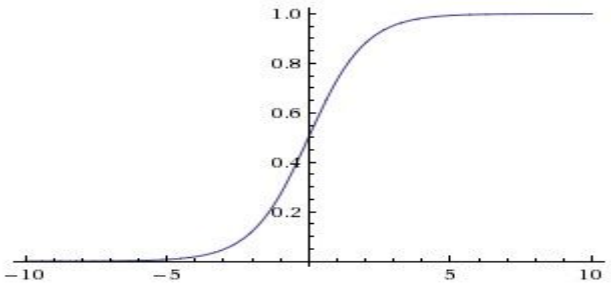


Figura 2.1: Reprezentarea grafică a funcţiei sigmoid

Sursă: CS231n [17]

Funcţia sigmoid a fost folosită în trecut datorită faptului că are codomeniul 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 apărea schimbări prea mari în ponderile acestuia, reţeaua învăţând foarte greu. Acest lucru poate fi evitat parţial prin iniţializarea foarte atentă a ponderilor neuronilor care folosesc sigmoid. [17]

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, [17]. 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]. Tangenta este computaţional scumpă, precum funcţia sigmoid în (2.3), 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:

$$\tanh(x) = 2 * \sigma(2 * x) + 1 \qquad (2.4)$$

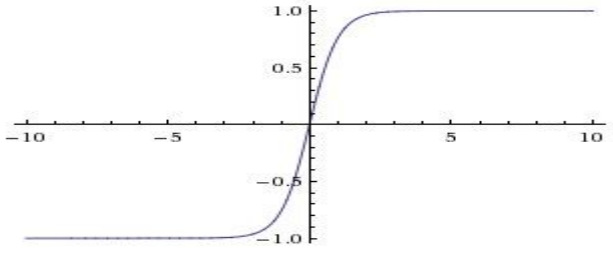


Figura 2.2: Reprezentarea grafică a funcţiei tangentă

Sursă: CS231n [17]

În continuare vom prezenta funcţia de activare **ReLU**. Alex Krizhevsky a fost propus în [1] să folosim această funcţie în reţelele neuronale în detrimentul tangentei, acurateţea rămânând aceeaşi în ciuda faptului că ReLU este mult mai rapidă. Formula şi distribuţia funcţiei apar în continuare:

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

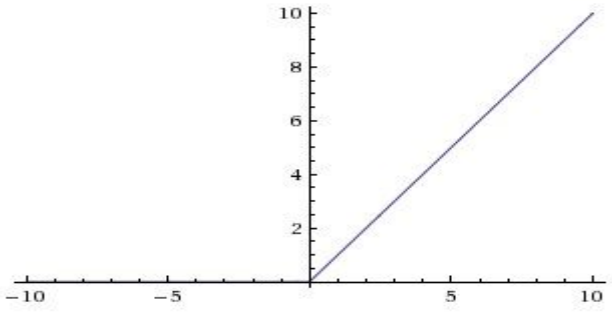


Figura 2.3: Reprezentarea grafică a funcţiei ReLU

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 predicţia.

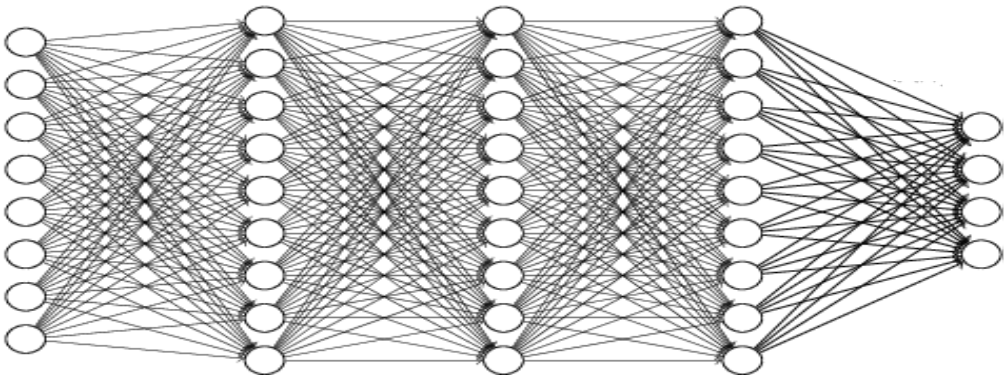


Figura 2.4: Reprezentarea grafică a unei reţele neuronale

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 folosită funcţie pentru a efectua acest lucru este **Softmax**. Ea primeşte la intrare un vector de dimensiune n şi returnează un vector de aceeaşi dimensiune cu valori între 0 şi 1, având suma totală egală cu 1 [15]. Astfel, valoarea vectorul de ieşire poate fi gândită ca 'probabilitatea ca un element să aparţină acelei clase'. Funcţia Softmax are următoarea formulă:

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

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 [15]. În funcţie de tipul problemei pe care încercăm sa o rezolvăm pot fi folosite 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. [15]

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 scade 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 răspunsul este reprezentat de suma componentelor matricei rezultate.

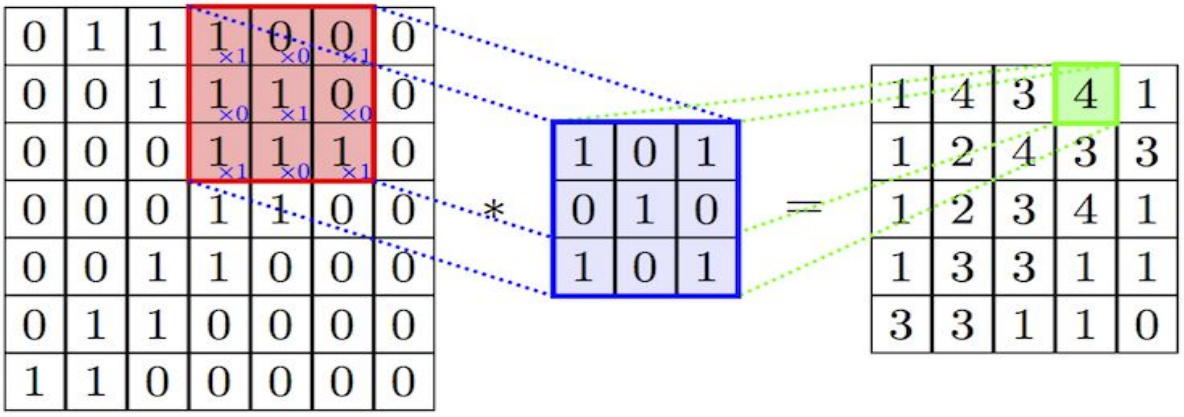


Figura 2.5: Reprezentarea grafică a unui pas de convoluţiei

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. De exemplu, 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 discriminatorie. 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, [3].

Deplasarea reprezintă numărul de pixeli peste care va sări filtrul după fiecare pas. Când deplasarea este de dimensiune 1 filtrul va merge câte un pixel, iar când aceasta este 2 acesta 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 trăsături rezultată, iar acest lucru se poate realiza prin adăugarea de zerouri în jurul imaginii înaintea realizării 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 înălţimea/lăţimea matricei de trăsături, W înălţimea/lăţimea ferestrei de intrare, F înălţimea/lăţimea 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 acest procedeu, matricea de trăsături este împărţită în zone şi este ales maximul din fiecare petic.

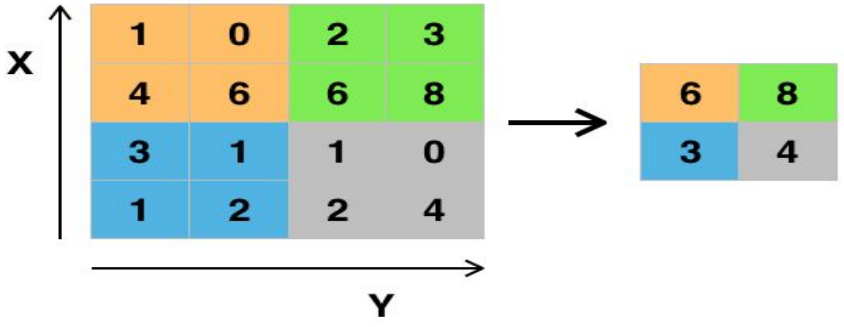


Figura 2.6: Strat de unificare prin extragerea maximului

Sursă: Wikipedia [15]

Există şi alte metode de unificare, precum calcularea mediei sau a sumei. Când folosim straturi de unificare trebuie să avem grijă la dimensiunea matricei rezultate. În [4] ne este dată formula prin care putem calcula dimensiunea matricei finale, aceasta fiind aceeaşi indiferent de tipul de unificare:

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

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

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

O reţea neuronală convoluţională va fi compusă din mai multe straturi convoluţionale, 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 probabilităţile de apartenenţă pentru fiecare clasă.

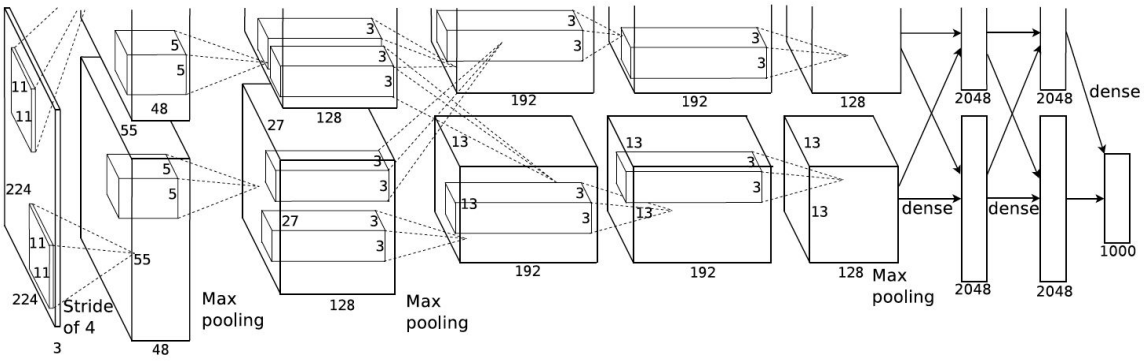


Figura 2.7: Reţeaua AlexNet, propusă de Alex Krizhevsky

Sursă: [1]

Alex Krizhevsky, Ilya Sutskever şi Geoffrey Hinton au şocat lumea în 2012, prezentând o reţea neuronală convoluţională care a câştigat concursul ILSVRC şi care a obţinut o eroare de doar 15.4% pe primele 5 predicţii [15]. Acesta a fost primul model care a avut un astfel de succes pe setul de date foarte dificil ImageNet, precedenta acurateţe fiind de 26.2%. Pornind de la ideile prezentate de cei 3 în [1], an de an au apărut reţele care au bătut recordul precedent. În 2015, Microsoft a ajuns la o eroare pe primele 5 predicţii de 3.56% prin reţeaua ResNet, [5].

Reţelele convoluţionale pot fi folosite la multe lucruri. De exemplu, în domeniul construcţiei de maşini care se conduc singure reţele convoluţionale sunt foarte importante, fiind folosite în detectarea celorlalte maşini şi a liniilor de pe stradă, precum şi a semnelor de circulaţie.

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 deprinde 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. 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 împărţită în regiuni conectate numite celule, iar fiecare celulă va 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.

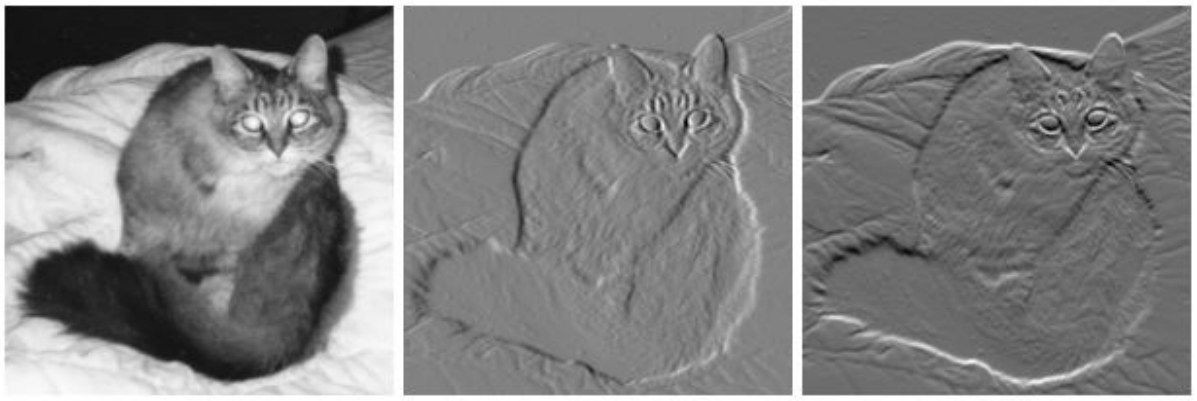


Figura 2.8: Gradienţii unei imagini

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 din ultimele 2 imagini au o valoare mică a gradienţilor, pe când pixeli albi şi negri au o valoare ridicată [17].

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. 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 următoarele 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. Filtrul care a dat cele mai bune rezultate aici 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.

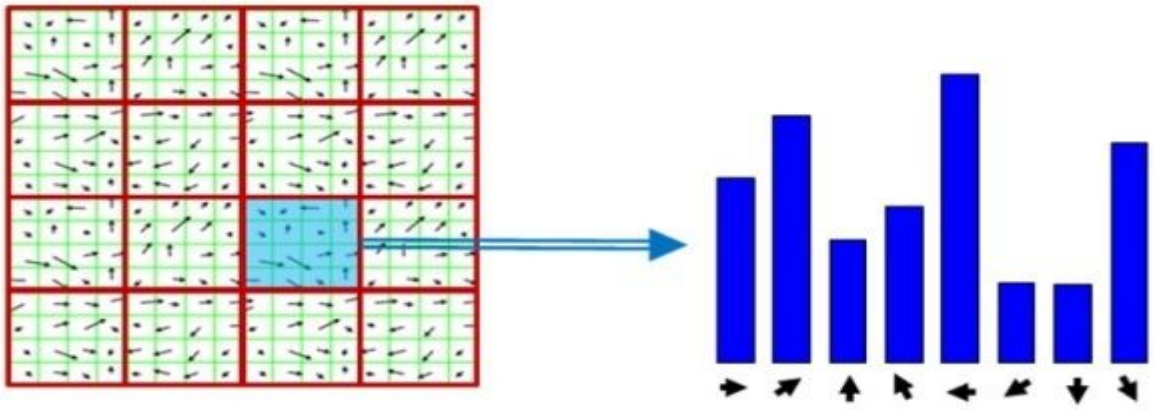


Figura 2.9: Reprezentarea grafică a histogramei unei celule

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. În acest spaţiu, o suprafaţă liniară de decizie este construită.

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'.*

Conform definiţiei (2.3) hiperplanul într-un spaţiu de dimensiune 2 este format dintr-o dreaptă sau un lanţ de drepte.

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 notat corespunzător, atribuie unei noi intrări una dintre clase.

Un model de maşină cu suport vectorial poate fi gândit ca o adunare de mai multe puncte într-un spaţiu, fiecare punct reprezentând una dintre datele de intrare. Acestea sunt adăugate astfel încât categoriile diferite să fie distanţate cât mai mult posibil una de cealaltă. Maşinile cu suport vectorial pot face şi clasificări neliniare pe lângă cele liniare folosind aşa-numitul truc al nucleului, trecând datele de intrare într-un spaţiu cu dimensiune foarte mare, [15].

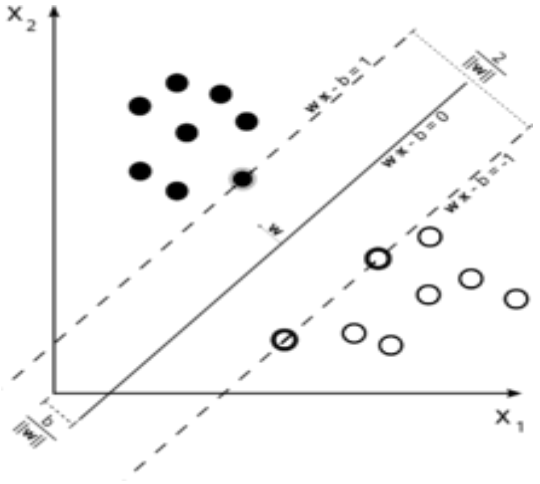


Figura 2.10: Reprezentarea grafică a unui SVM liniar

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.

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, [15].

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

Construirea modelului SVM este influenţată de 2 paremetrii:

- parametrul de **Regularizare** (notat cu C în librăria Scikit-Learn pentru Python) spune clasificatorului cât de mult îi permiţi să clasifice greşit datele de antrenare. Pentru valori mari ale lui C, funcţia de optimizare va alege un hiperplan format dintr-un lanţ de drepte de dimensiuni mici pentru a face mai bună clasificarea punctele de antrenare. Pe de altă parte, o valoare mică a lui C va face funcţia de optimizare să caute un hiperplan format dintr-o singură dreaptă, 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, valoarea mică a lui C dând o generalizare mai mare a delimitărilor. 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, doar punctele din apropiere vor fi folosite dacă valoarea lui gamma este mare, [16].

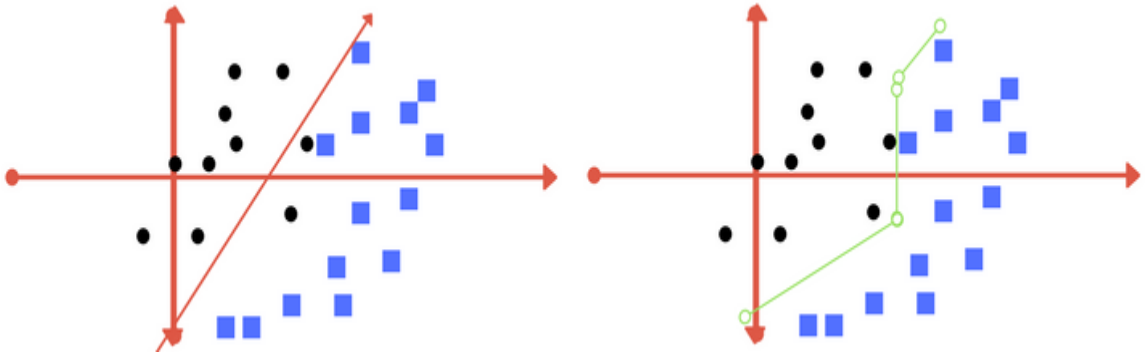


Figura 2.11: În stânga SVM-ul are valoarea parametrului de regularizare mică.

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

Sursă: Mediul [16]

În continuare vom discuta despre noțiunea de margine în maşinile cu suport vectorial. SVM-ul încearcă să obțină 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 sunt bune. Acest lucru va face clasificatorul să separe exemplele pozitive de cele negative eficient.

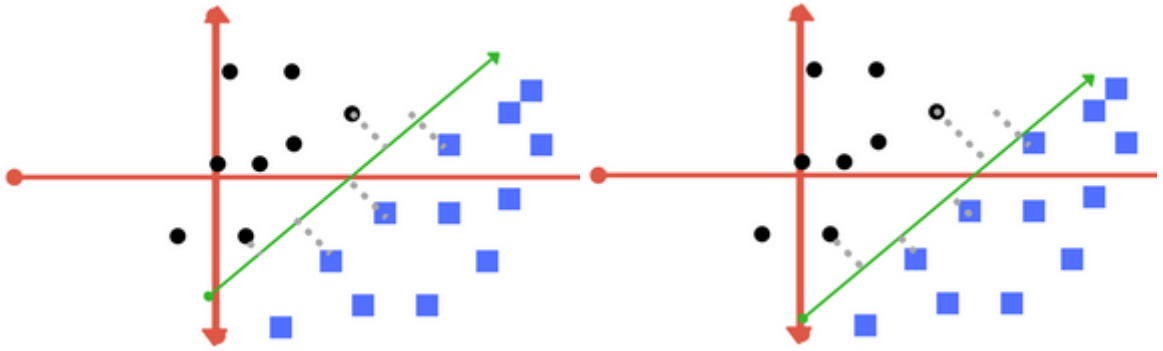


Figura 2.12: În stânga marginea este bună.

În dreapta marginea nu este construită 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 conține 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 un topic de discuţie în inteligenţa artificială, [11]. Acest algoritm este folosit pentru clasificare trăsăturilor sau regresie.

În construirea clasificatorului KNN avem nevoie de 3 componente:

- setul de date pentru antrenare cu clasele corespunzătoare
- metoda prin care se va calcula distanţa între 2 înregistrări. 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. Alegerea unui k mic face ca modelul să fie foarte sensibil la date de intrare introduse eronat. Pe de altă parte, o valoare prea mare a lui k poate duce la situaţia în care decizia este luată de puncte aflate la o distanţă foarte mare de cel căutat. De exemplu, pentru un k = 21 şi un punct X avem 7 vecini din clasa 1 la o distanţă relativ mică şi restul de 14 din clasa 0 la o distanţă mai mare. Deşi X este din clasa 1 şi este înconjurat de clasa corectă în apropiere, prezicerea ne va spune că X este din 0 deoarece numărul vecinilor din 0 este de 2 ori numărul de 1.

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, conform [11], alţi algoritmi mai rapizi au fost propuşi de Farago în [13] şi de Belur Dasarathy în [12], ultimul fiind bazat pe arbori de căutare.

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 trebuie 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 compari mai multe modele şi să verifici rezultatele obţinute.

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

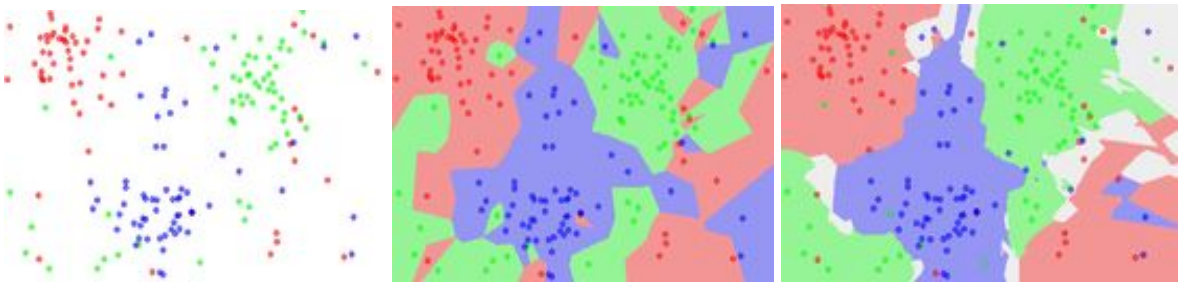


Figura 2.13: De la stânga la dreapta: setul de date, reprezentare grafică KNN pentru k = 1, reprezentare grafică KNN pentru k = 5.

Sursă: Wikipedia [15]

În partea stângă avem setul de date colorat diferit în funcţie de clasa de care aparţine. Imaginea 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 roşie, 2 albaştrii şi unul 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 foloseau 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 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 decât distanţa faţă de vectorul rezultat din imaginea unei persoane diferite. 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 comparabile cu 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 euclidiană. Formula funcţiei de cost triplă este următoarea:

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

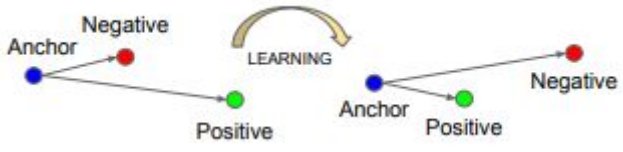


Figura 2.14: Antrenarea în FaceNet

Conform figurii 2.14 reţeaua primeşte câte 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 numărului optim de staturi pe care trebuie să le aibă o reţea. 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, iar reţeaua nu va reuşi să înveţe.

Pentru a evita această situaţie, autorii articolului [5] au propus o învăţare reziduală. Ei au decis să introducă o funcţie de identitate (numit strat 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.

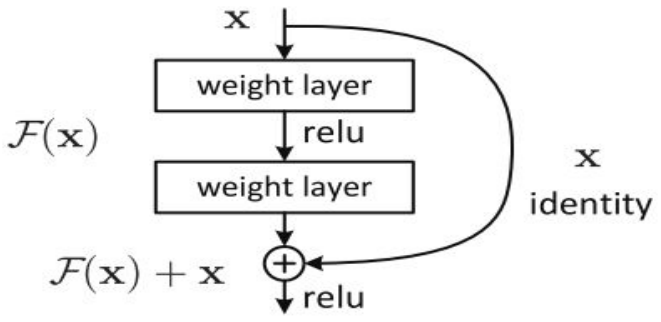


Figura 2.15: Reprezentarea unui strat rezidual

Prin antrenarea unei reţele cu astfel de straturi au observat că a învăţat mult mai bine decât cea care nu le conţinea. Acest lucru se datorează faptului că reţeaua poate decide acum dacă va antrena acele straturi sau va sări peste ele.

Prin acest procedeu Microsoft a reuşit să construiască o reţea 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 nevoit 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 sistem nu este suportat oficial. Aici, ne sunt oferite funcţii pentru găsirea feţelor din imagini, existând atât o implementare naivă prin HOG+SVM, precum şi una prin reţele convoluţionale. 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]. 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 mulţi algoritmi din domeniul inteligenţei artificiale.

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. Toate imaginile

din setul de date vor fi parcurse, iar pentru fiecare imagine vom extrage feţele prezente şi vom trece aceste feţe prin reţea pentru a obţine trăsăturile.

```
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, iar 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 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))
```

Librăria Flask din Python a fost folosită 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 este incorectă.

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 actori aleşi. Deoarece nu am reuşit să găsesc 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 arăta şi imagini în care actorul căutat era între mai multe persoane sau nu era în imagine. Din 500 rezultate rămâneam cu aproximativ 200 după prima fază. 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. 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%. După aceea am verificat şi cu un KNN, acesta reuşind 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 simplu. 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 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 puţin 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.

Putem observa că prin folosirea unei reţele foarte bine antrenate cu o funcţie de cost triplă se obţin rezultate foarte bune pe un set de date nou cu un număr relativ mic de imagini, pe când HOG nu s-a descurcat prea bine.

Metodă de extragere a trăsăturilor	Clasificator	Acurateţe
ResNet (Conv)	SVM	99,7%
ResNet (Conv)	KNN	99,73%
HOG	SVM	55,25%
HOG	KNN	27,6%

Tabelul acurateţii obţinute în funcţie de metodă

Capitolul 3

3. Tehnologii folosite în dezvoltarea aplicaţiei web

3.1. Java

Aplicaţia web a fost realizată în **Java**. Java este un limbaj de programare de nivel înalt bazat pe clase 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 între Java şi C++ este lipsa variabilelor constante. Totuşi, există o modalitate 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’ ne 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: sintaxă, tipurile de date, elementele POO, colecţii, interacţiunea cu baza de date prin JDBC, manipularea fişierelor XML, 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 Throwable stă la baza erorilor şi excepţiilor limbajului Java. Erorile (clasa Error care implementează Throwable) sunt problemele care apar la maşina virtuală Java, acestea rezultând în oprirea execuţiei. Excepţiile (clasa Exception care implementeză Throwable) 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).

În Java nu este nevoie să eliberăm memoria obiectelor create, acest lucru fiind realizare de maşina virtuală printr-un program demon care eliberează memoria folosită de obiectele care nu mai sunt folosite. Există 2 metode de a cere maşinii virtuale să ruleze acest program de eliberare al memoriei: 'System.gc(); ' şi 'Runtime.getRuntime().gc(); '. Cu toate acestea nu există nici o modalitate să ne asigurăm ca maşina virtuală a ascultat de cererile noastre. Pot apărea situaţii în care trebuie să executăm anumite operaţii la distrugerea unui obiect. Un astfel de exemplu este închiderea conexiunii cu baza de date sau închiderea unui fişier deschis de constructorul clasei. Acest lucru se poate realiza suprascriind funcţia 'finalize()' din clasa Object, cea mai importantă clasă din Java.

3.2. Maven

Maven este un program realizat de Apache prin care dezvoltatorii Java pot introduce uşor 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 install), valida şi compila codul, curăţa artefactele vechi şi lansa aplicaţia pe un server, [20]. În plus, proiectul ne este împărţit 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 returna rezultatul corect chiar dacă 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.

3.3. Spring

Spring-ul este o aplicaţie care are la bază principiul de proiectare numit ‘Inversiunea controlului’. Acest principiu ne spune că dezvoltatorul ar trebui să fie atent doar la părţile specifice ale unei aplicaţii, iar părţile comune să fie făcute de aceste aplicaţii precum Spring.

Fie o clasă Brad creată în Java. Pentru a obţine un obiect cu proprietăţi din acestă clasă este necesar să creăm unul nou folosind ‘new’ şi să atribuim aceste proprietăţi. Pe de altă parte, folosind modelul de programare al fabrici putem obţine acest obiect apelând o metodă prin care trimitem proprietăţile. Acest fenomen se numeşte căutarea dependinţei. Totuşi, prin injectarea dependinţei obiectul este trimis direct. Acest lucru poate fi implementat în constructori, în funcţiile de setare şi în proprietăţile claselor.

Pentru implementarea injectării dependinţelor prin constructori clasa care le conţine trebuie să fie adnotată cu ‘@Configuration’. Această adnotare îi spune Spring-ului că acea clasă conţine un bob. Un bob Spring este creat şi administrat de containerul Spring, aceste boabe fiind compuse folosind unele fişiere de configurare oferite de către programator containerului, de exemplu folosind formatul XML. Crearea unui bob pentru clasa Brad se face în felul următor: <bean id=”brad” class=”Brad” />. Adnotarea ‘@Bean’ este folosită pentru a defini că o metodă este un bob. Dacă nu este specificat un nume personalizat, numele ales va fi numele metodei. Pentru realizarea injectării dependinţei pe funcţiile de setare se foloseşte atributul ‘property’ în XML: <bean id=”brad” class=”Brad”> <property name=”nrRamuri” ref=”nrRamuri1” /> </bean>. Injectarea directă în proprietăţile claselor se face folosind adnotarea ‘@Autowired’.

Spring foloseşte adnotări pentru afla tipul clasei, metodei sau proprietăţii. Astfel, toate clasele controlor din Spring trebuie adnotate cu ‘@Controller’. Adnotările ‘@RequestMapping’ sunt folosite pentru a mapa o adresă pe o anumită clasă sau metodă. Dacă clasa are o adnotare pe adresa ‘/url1’ şi o metodă din interiorul ei ‘/url2’, atunci metoda va mapa adresa ‘/url1/url2’ în final. ‘@RequestParam’ este folosit pentru a lega un parametru al cererii web de o variabilă a funcţiei. Adnotarea ‘@ModelAtribute’ este una dintre cele mai

importante în Spring. Prezenţa ei ne permită să legăm parametrii unei cereri de tip POST de o clasă fără să parsăm fiecare câmp individual. Acest fenomen este cunoscut ca legarea datelor în Spring.

Adnotarea specifică claselor de servicii este '@Service', ele cuprinzând toată logica de business a aplicaţiei (de exemplu funcţiile de validare). Acestea sunt principalele metode pe care trebuiesc făcute teste unitare, nu funcţiile care fac modificări pe baza de date. '@Repository' este folosită pentru clasele de acces la baza de date, iar '@Component' este o adnotare generică pentru clasele administrate de Spring, dar care nu sunt controlere, servicii sau clase care fac legătura cu baza de date.

3.4. Oracle

Oracle este un sistem de gestiune al bazelor de date creat de firma cu acelaşi nume. Această bază de date este de tip SQL.

Definiţia 3.2: *O baza de date reprezintă un mod prin care putem stoca şi realiza operaţii cu milioane de înregistrări.*

Înregistrările bazei de date sunt reţinute în tabele. Un tabel reprezintă o relaţie între mai multe mulţimi. O tabelă cuprinde o listă a câmpurilor, aici fiind specificate numele, tipul, dimensiunea şi valoarea implicită unde este cazul. Pe lângă aceasta tabela conţine şi o listă de constrângeri ale datelor salvate precum unicitatea sau completarea obligatorie a câmpului. Fiecare tabelă trebuie să aibă o cheie primară, adică un atribut sau o combinaţie de atribute unică şi diferită de nul pe acel tabel.

Oracle permite folosirea unui limbaj de tip SQL prin care putem declara şi manipula datele. Pe lângă acesta ne este pus la dispoziţie limbajul PL/SQL prin care putem folosi metode pentru a realiza operaţii mai complexe pe baza de date precum parcurgerea datelor selectate folosind un cursor sau construcţia 'pentru'. O particularitate a limbajului PL/SQL este aceea că putem crea unele funcţii care să fie rulate automat înainte sau după executarea uneia sau mai multe operaţii de manipulare (creare, actualizare sau ştergere). Procedurile sunt de 2 feluri: vizibile doar în interiorul blocului PL/SQL sau globale, cele din urmă putând fi apelate folosind funcţia 'call'.

3.5. MyBatis

MyBatis este un program folosit pentru a stabili legătura dintre aplicaţia Java şi baza de date Oracle. Acesta foloseşte fişiere cu extensia '.xml' pentru a scrie implementarea funcţiilor din interfeţele pe care le mapează. Apelarea metodelor interfeţelor se face creând un obiect cu adnotarea '@Autowired'.

Librăria ne pune la dispoziţie o modalitate de evitare a scrierii de mai multe ori a aceluiaşi cod folosind tagul <sql>, acesta putând fi folosit pentru a enumera toate câmpurile dorite atât în citirea tuturor datelor din tabel, cât şi la recuperarea unei anumite intrări după identificator.

Structura 'dacă' poate fi folosită direct în fişierul '.xml'. Prin această structură putem construi interogări dinamice a bazei de date. Acest lucru este foarte avantajos când vrem să facem o retragere de date folosind un filtru cu foarte multe componente. Folosind condiţiile 'dacă' putem scrie aceste interogări foarte uşor.

MyBatis-ul ne oferă posibilitatea să mapăm coloanele interogării bazei de date pe un obiect Java, spunându-i coloanei de ce proprietate să fie legată. Dacă o coloană nu este specificată, aceasta se va lega de proprietatea cu acelaşi nume.

3.6. HTML

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 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 la finalul 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 după încărcarea ferestrei în cazul în care internetul are o viteză mică.

JSP-ul este o tehnologie care ne ajută să creăm 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 folosind ‘dacă’, ‘alege dintre’ şi ‘pentru fiecare’. În plus, ne sunt puse la dispoziţie posibilităţi să formatăm uşor lucruri precum data, timpul şi numere.

3.7. CSS

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 div-ul cu identificatorul Y). Există 3 posibilităţi de a adăuga 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 între tagurile ‘<style>...</style>’. Acest lucru ne permite să ne referim la mai multe elemente prin clase sau prin tip în modul următor: ‘.’ urmat de numele clasei este folosit pentru a identifica acea 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 decizia de a pune stilul într-un fişier separat şi de a-l include în HTML printr-un import.

Pe lângă posibilitatea de a modifica poziţie, dimensiunea, culoarea şi restul proprietăţilor unor obiecte, CSS-ul ne pune la dispoziţie posibilitatea de a efectua tranziţii ale obiectelor, animaţii sau transformări 2D şi 3D la anumite acţiuni precum apăsarea unui buton de pe mouse folosind tagul ‘active’ sau planarea asupra componentei prin tagul ‘hover’.

3.8. JavaScript + JQuery + Ajax

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 prelucrează tot codul de la început, pe când la interpretare este aleasă prima line, este compilată şi executată, iar apoi se trece la linia următoarea. La fel ca şi CSS-ul, JavaScript-ul poate fi trecut atât în componenta ‘head’ a fişierului, cât şi într-un fişier separată care apoi este importată în HTML pentru a respecta principiul separării conceptelor. De aceea vom avea 3 fişiere separate pentru interfaţa unei pagini web, fiecare având funcţia sa proprie.

JQuery este o librărie de JavaScript folosită pentru a scrie cod mai rapid. Aceasta ne pune le dispoziţie posibilitatea de a folosi selectorii CSS pentru a alege elementele, putând astfel să facem o selecţie mai complicată de elemente mult mai uşor decât dacă am fi avut doar JavaScript. În plus codul scris este mai scurt, scăzând timpul de dezvoltare şi fiind mai uşor de urmărit. De exemplu, pentru a căuta un element după identificatorul ‘actor’ în JavaScript ar trebui să scriem document.getElementById(‘actor’), pe când în JQuery acelaşi lucru se poate obţine apelând $(‘#actor’).

Spre deosebire de JavaScript care lucrează direct cu proprietăţile obiectelor, JQuery se foloseşte de funcţii pentru a face acelaşi lucru: pentru găsirea informaţiei se apelează funcţiile fără parametrii, iar pentru setare se trimite valoarea de atribuire apelând metodele cu parametru. Deşi sunt diferite, funcţiile JQuery sunt asemănătoare cu proprietăţile corespunzătoare în JavaScript: corespondenta proprietăţii ‘value’ din JavaScript este funcţia ‘val()’ din JQuery, ‘innerHTML’ corespunde cu ‘html()’ etc. Conversia de la un obiect JavaScript la unul JQuery se face simplu folosind $: objQ = $(objS).

AJAX este folosit pentru a face cereri la server fără a reîncărca pagina web. La noi, AJAX a fost folosit pentru apela serviciul REST de actualizare a filmelor favorite ale utilizatorului conectat. Cererile la server prin AJAX pot fi făcute folosind atât JavaScript, cât şi JQuery. Pentru a trimite parametrii la serviciu se poate folosi câmpul ‘data’ pentru cereri de tip POST, iar parametrii pentru cererile GET se pot trece la finalul adresei serviciului prin adăugarea unui ‘?’ şi înşirarea parametrilor în felul următor ‘cheie=valoare’, fiind separaţi de caracterul ‘&’. Principala diferenţă între POST şi GET este că GET-ul trece parametrii la finalul adresei, iar POST-ul îi salvează în interior. De aceea, conectarea prin parola sau alte operaţii care conţin informaţii cu caracter personal trebuie să fie executate folosind POST pentru a nu permite nimănui să vadă parola în clar în adresă.

Recent, dezvoltatorii web au preferat să folosească Angular sau Vue în detrimentul JavaScript-ului pur şi JQuery-ului, acestea oferind o mai bună comunicare cu serviciile web

decât cele din urmă. Construirea unei aplicaţii care foloseşte doar servicii web este atractivă pentru programatori datorită faptului că face posibilă comunicarea între limbaje total diferite. Spre exemplu, în aplicaţia noastră am creat un serviciu Rest în Flask care primea ca parametru locaţia imaginii şi returna serverului Java numele actorului din imagine. Astfel, la crearea interfeţei putem alege ce dorim să folosim, nefiind obligaţi de limbajul în care a fost scris serverul.

Capitolul 4

4. Descrierea aplicaţiei

Aplicaţia web a fost creată cu scopul de a pune la dispoziţie un mod de a recunoaşte un grup de actori din imagini şi de a afla informaţii despre aceştia.

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, clasele 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.
- InfoActRecognition. Parte separată a aplicaţie care pune la dispoziţie un serviciu web Rest realizat în librăria Flask din Python prin care are loc comunicarea dintre aplicaţia Java şi funcţia de recunoaştere.

Baza de date a fost realizată în Oracle, iar comunicarea între aceasta şi serverul web a fost făcută folosind MyBatis. Informaţiile prin care ne conectăm la bază au fost reţinute într-un fişier de configurare, din care a fost creat mai târziu un Spring Bean pentru a recupera aceste informaţii. Diagramă relaţională a bazei este prezentată în continuare.

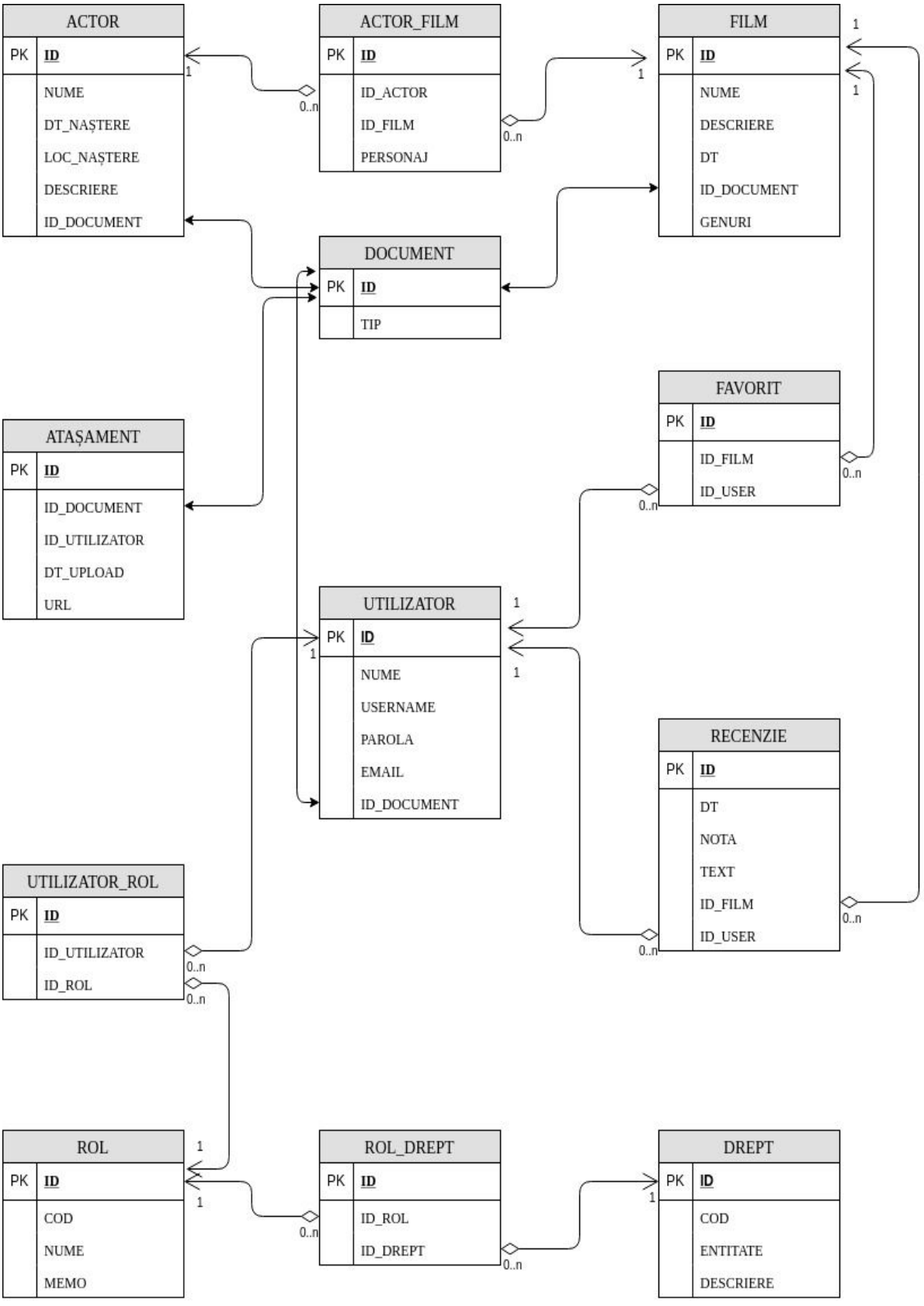


Diagrama relațională a bazei de date

Coloana ID în fiecare tabelă este generată folosind o secvenţă care se incrementează cu 1. Acest lucru ne oferă un identificator unic pentru fiecare intrare care apoi poate fi folosit să ne referim la acel obiect fără a transmite toate informaţiile, ci doar identificatorul. De obicei acesta este folosit în aplicaţiile web pentru a exprima o acţiune pe acel obiect trecând-ul ca parametru în cererea către server. Pentru recuperarea informaţiilor este suficient doar identificatorul deoarece le putem citi tot obiectul din baza de date prin acesta.

Tabelul ACTOR a fost folosit pentru a reţine informaţiile actorilor. Câmpul ID_DOCUMENT crează o relaţie de 1 la 1 cu tabela DOCUMENT, aceasta fiind folosită la afişarea imaginilor actorilor. Între tabelele ACTOR şi FILM există o relaţie ‘many to many’ deoarece actorul poate apărea în mai multe filme, iar un film poate avea mai mulţi actori. Aceasta este folosită pentru a ne spune ce actor a jucat în ce film, precum şi personajul pe care l-a interpretat. Tabelul FILM reţine datele filmelor introduse de moderator.

Tabelul UTILIZATOR reţine informaţiilor utilizatorilor înregistraţi în aplicaţie. Parola este salvată după trecerea prin funcţia SHA-256 pentru a asigura protecţia datelor în cazul în care un atacator reuşeşte să afle informaţiile reţinute în bază. ATAŞAMENT-ul ne păstrează informaţii despre imaginile încărcate de moderatori pentru a fi afişate în paginile actorilor şi filmelor corespunzătoare. Aici sunt păstrate informaţii despre cine şi când a încărcat o imagine, precum şi numele acesteia în fişierul de pe server. Tabela FAVORIT ne spune dacă un utilizator a introdus un film la favorit, astfel dând posibilitatea fiecărei persoane să îşi salveze un film pentru a-l urmări mai târziu. RECENZIE este folosit pentru a permite utilizatorilor să adauge o recenzie unui film. Folosind coloana NOTĂ din această tabelă putem crea o diagramă a notelor date de utilizatori unui film.

ROL-ul reprezintă rolurile utilizatorilor din aplicaţie, precum moderator, admin etc. Tabela UTILIZATOR-ROL ne permite să asigurăm unui utilizator mai multe roluri. DREPT şi ROL_DREPT ne permite să adăugăm drepturi de citire/scriere pe entităţi pentru fiecare rol.

Folosirea tabelelor de roluri şi drepturi ne oferă posibilitatea să adăugam direct din aplicaţie noi roluri şi să atribuim drepturi acestora. Într-o aplicaţie de dimensiuni mari acest lucru este obligatoriu deoarece mereu pot apărea funcţionalităţi speciale unor tipuri de utilizatori, iar prin acest mod ne este foarte uşor să rezolvăm partea de acces.

În continuare o să vedem diagrama de activităţi a aplicaţiei, iar apoi vom discuta mai pe larg ce funcţionalităţi avem.

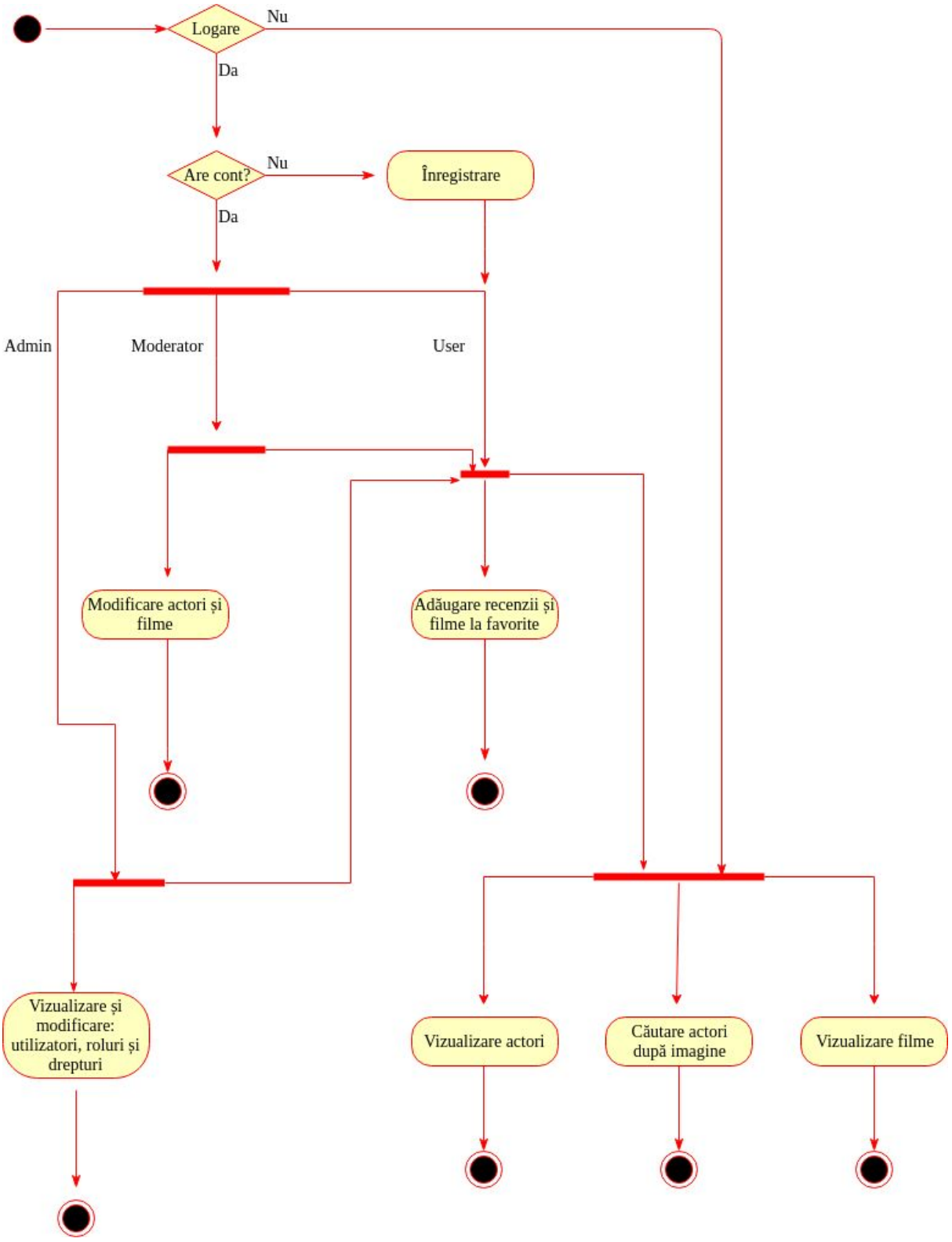


Diagrama de activităţi a aplicaţiei

În momentul deschiderii aplicaţiei, pe prima pagină apar câteva informaţii despre cum poate fi folosită şi scopul acesteia. În partea dreaptă sus ne este pus la dispoziţie un buton de logare. Dacă dorim să ne conectăm în aplicaţie, la acest punct ne apare un ecran în care putem introduce numele şi parola. Lângă butonul de 'Autentificare' există şi butonul de 'Înregistrare'. La crearea unui cont ne sunt cerute următoarele informaţii: username-ul şi parola prin care ne putem conecta în viitor, numele şi adresa de email. Username-ul şi email-ul trebuie să fie unice pentru fiecare utilizator din baza de date, de aceea există un verificator care te va anunţa dacă acestea sunt deja folosite. În momentul înregistrării utilizatorul primeşte drepturile unui utilizator normal, urmând ca un moderator să îi acord un alt rol dacă este cazul. În momentul logării în aplicaţie este creată un HttpSession prin care vom reţine utilizatorul. Sesiunea expiră după 30 de minute, deconectându-i pe cei care nu au executat nici o cerere la server în acest interval. Expirarea sesiunii este un mod simplu de a creşte securitatea aplicaţiei.

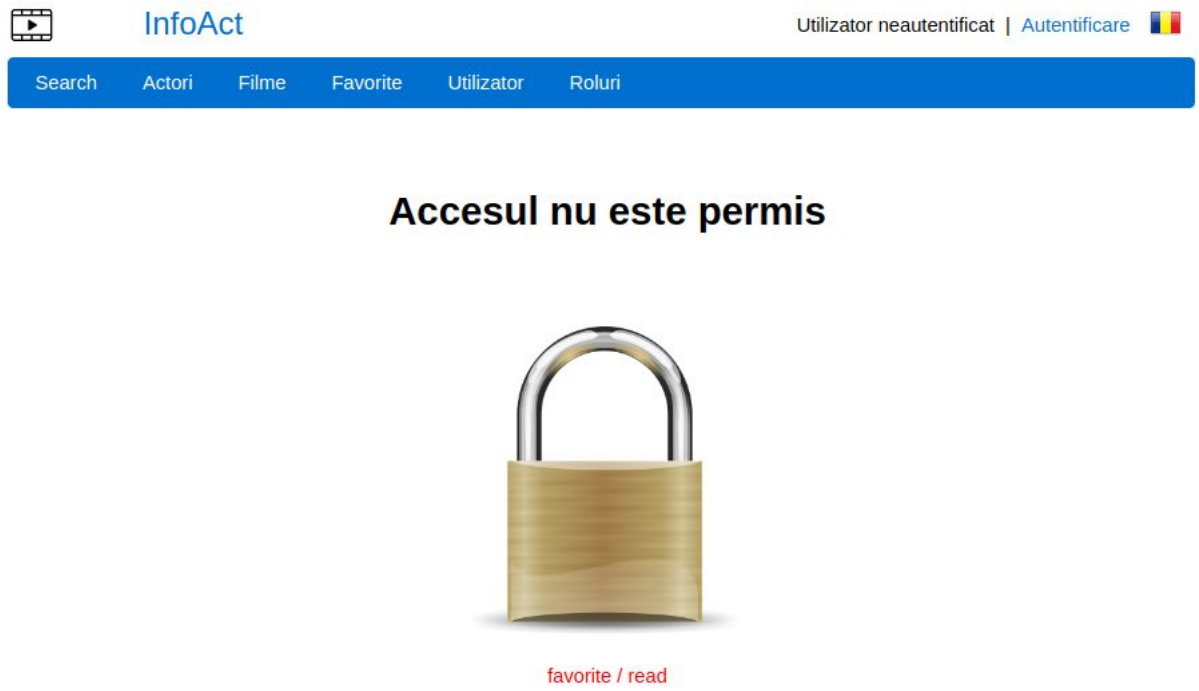
Pentru securitatea informaţiilor utilizatorului, parolele din baza de date sunt ascunse folosind funcţia de 'hash' SHA-256. Această funcţie este urmaşa lui SHA-1, cea din urmă fiind declarată nesigură deoarece cei de la Google au găsit o coliziune generată de acest algoritm, [10]. Găsirea unei ciocniri ne spune că un atacator se poate conecta chiar dacă nu ştie parola corectă, ci folosind celălalt cuvânt din pereche. SHA-256 este mai sigură decât SHA-1 datorită dimensiunii mai mare a numărului de biţi, 256 contra 160.

În momentul de faţă aplicaţia este gândită să cuprindă 4 tipuri de utilizatori: utilizator neconectat, utilizator înregistrat, moderator şi admin. Astfel, un utilizator neconectat poate doar să caute un actor din imagine, să vadă informaţii despre acesta şi despre filmele aflate în baza de date. Pe lângă toate cele prezentate, un utilizator înregistrat în aplicaţie are posibilitatea să salveze filmele favorite şi să lase recenzii cu opinia. Moderatorul are dreptul şi obligaţia de a modifica conţinutul paginilor, iar adminul de a modifica drepturile şi rolurile utilizatorilor.

Deoarece roluri au fost create dinamic, este foarte uşor să adăugăm unul nou. Un admin trebuie doar să intre în pagina de roluri, să adauge nume şi să îi atribuie drepturile dorite. Fiecare ecran are doua drepturi, 'read' şi 'write', care pot fi date sau revocate foarte uşor folosind acest panou.

Funcţiile din clasele controlor verificată drepturile utilizatorului conectat. De exemplu, pentru a vedea lista filmelor favorite, prima linie executată în controlor este o

funcţie care verifică dacă utilizatorul conectat în sesiune are dreptul de ‘read’ în ecranul ‘favorite’. Drepturile unui utilizator sunt citite din baza de date folosind legături între tabele pornind de la UTILIZATOR şi ajungând la tabela DREPT. Dacă nu este găsit dreptul de ‘read’ în ecranul ‘favorite’, atunci este aruncată o excepţie EroareAutorizare care va fi prinsă de Spring şi ne va fi întoarsă următoarea pagină:



Ecranul de ‘EroareAutorizare’ al aplicaţiei

În plus, aici apare motivul pentru care nu a fost lăsat să acceseze pagina, adică lipsa dreptului de ‘favorite / read’. Adminul poate decide să dea drepturi unui grup de utilizatori să facă o acţiune sau să le blocheze accesul.

Fiecărui utilizator al aplicaţiei îi este permis să încarce o imagine pe server pentru a afla dacă un actor existent în baza de date a fost identificat. Încărcarea imaginilor pe server a fost realizată printr-o cerere de tip POST ‘multipart’. Imaginea este transferată în părţi de câte 1024 octeţi. Serverul primeşte cererea şi încarcă imaginea într-un folder după ce numele este schimbat într-un UUID.

Definiţia 4.1: *Un UUID este un identificator generat cu dimensiunea de 128 biţi cu o probabilitate atât de apropiată de zero de a genera o coliziune încât este considerată neglijabilă.*

Locaţia folderului este stabilită într-un fişier cu extensia ‘.properties’ din care a fost creat un Spring Bean. După încărcarea imaginii în folder, serverul Java (port 8080) se

conectează la serverul Python (port 8081), trimiţându-i locaţia imaginii prin una din cele 2 servicii REST, prima folosind un model CNN+SVM pentru recunoaştere, pe când cea de-a doua un algoritm CNN+KNN. Modul de recunoaştere este stabilit înainte de către dezvoltator, apelând una din cele 2 metode puse la dispoziţie. Ambele modele de recunoaştere încarcă imaginea primită şi extrage feţele din imagine. Dacă numărul feţelor extrase este diferit de 1, este returnat ‘’, iar serverul Java va spune că imaginea introdusă nu este corectă.

Dacă numărul feţelor este 1, atunci sunt extrase cele 128 trăsături folosind librăria Face Recognition, iar apoi acest vector este trecut printr-un clasificator SVM sau KNN, în funcţie de serviciul apelat. Clasificatorul ne returnează numele actorului din imagine, iar serverul Java va căuta actorul cu acel nume în bază de date şi îl va trimite utilizatorului. Pentru a evita situaţia în care există 2 actori cu acelaşi nume, ar fi de preferat să antrenăm un model care întoarce identificatorul actorului respectiv din baza de date. În cazul acestei antrenări trebuie să avem mare grijă în momentul în care ştergem un anumit actor din oricare motiv (eroare de soft, greşeală a unui moderator), deoarece la reintroducerea acestuia trebuind să-i schimbăm câmpul ‘id’ din baza de date sau să antrenăm din nou reţeaua.

Ecranul de recunoaştere a actorului din imagine

Imaginea introdusă de utilizator apare în partea stângă. Aceasta este o imagine a lui Hugh Jackman din filmul Logan descărcată de pe Google . În partea dreaptă apare imaginea actorului luată de pe server şi jos ne este spus numele acestuia, putând de aici să mergem pe pagina sa pentru a afla mai multe informaţii. Pentru a obţine rezultate bune în căutare, imaginea feţei actorului ar trebui să aibă o dimensiune de cel puţin 50x50, dar 90x90 ar fi recomandat pentru a pune cu adevărat în valoare reţeaua convoluţională. Fiecare faţă care trece prin model este redimensionată înainte de a intra în reţea, dar redimensionarea de la o rezoluţie mică la una mai mare face ca imaginea să piardă din claritate, rezultând în rezultate mai proaste ale reţelei. Feţele care au fost folosite pentru a antrena reţeaua au avut o rezoluţie de cel puţin 50x50.

În ecranul de ‘Actori’, primul lucru pe care îl poate vedea un utilizator este o listare a actorilor ordonaţi în ordinea alfabetică după nume. Fiecărui actor îi este precizat numele, o mică descriere şi o imagine a acestuia. Pentru a afla mai multe informaţii, utilizatorul trebuie să intre pe pagina personală a actorului, acest lucru fiind realizat printr-o singură apăsare a mouse-lui pe nume. Toate rezultatele sunt paginate, fiind citite din baza de date doar cel mult 10 rezultate pentru a nu creşte timpul de răspuns al aplicaţiei. Pentru a afla mai multe rezultate, utilizatorul poate să folosească filtrul aflat în partea de sus a paginii pentru a căuta un actor după nume, precum şi meniul de navigare prin pagini aflat în partea de jos. În plus, moderatorii au posibilitatea de a intra pe pagina de adăugare a unui nou actor de aici.

În pagina pagina personală a unui actor se regăsesc mai multe informaţii publice despre acesta precum: numele, o mică descriere, data şi locul naşterii. În final apar o serie de imagini ale acelui actor încărcate pe site şi o listă cu personajele pe care le-a jucat în filme. Acestea din urmă pot fi modificate de un moderator fără a trebui să şteargă actorul şi să-l introducă din nou. Funcţiile de ştergere, adăugare şi editare a unui actor sunt puse la dispoziţie doar moderatorilor, aceştia trebuind să aibă grijă ca pagina să aibă un conţinut la curent. Orice alt utilizator care va încerca să folosească aceste funcţii va primi ca răspuns un mesaj de eroare.

Pagina de listare a filmelor este asemănătoare cu cea a actorilor. În ecranul de detaliu al unui film apar, pe lângă detaliile şi actorii participanţi, o diagramă a notelor date în recenzii, o listă a recenziilor şi o steluţă prin care putem adăuga un film la favorite. Spre deosebire de celelalte cereri la server, pentru adăugarea unui film la favorit am folosit o

funcţie JavaScript cu Ajax pentru a apela un serviciu RESTful în care are loc adăugarea sau a scoaterea un film de la favorit pentru un utilizator. Un serviciu asemănător este folosit şi pentru a adăuga filme în care a participat un actor sau actori care au jucat într-un film. Ecranul de favorite este asemănător cu cel de filme, singura diferenţă fiind lista filmelor care apar.

Orice utilizator conectat are dreptul să scrie o recenzie unui film în care îşi poate exprima părerea despre acesta şi îi poate acorda o notă. Adăugarea recenziilor ajută utilizatorii noi să aleagă un film pe care nu l-au văzut până atunci. Utilizatorul are dreptul să-şi şteargă recenzia, dar nu poate adăuga mai mult de una pentru acelaşi film. Moderatorul are dreptul de a şterge orice recenzie dacă este de părere că are un conţinut neadecvat precum cele în care sunt descrise scene din filmul respectiv, acestea deranjând persoanele care nu l-au văzut până atunci.

Diagrama notelor unui film

Diagrama notelor a fost realizată folosind librăria JavaScript Chart.js [21]. Deoarece această metodă a avut nevoie de un obiect JSON pentru crearea acestei diagrame, în funcţia de vizualizare din clasa de control specifică filmelor a trebuit adăugată această operaţie. Toate notele au fost trecute într-o listă, care apoi a fost transformată într-o listă JSON

folosind dependința Jackson și a fost trimisă în partea de interfață. JavaScript-ul a preluat această listă și a desenat o diagramă de tip bară albastră peste un obiect HTML Canvas.

Capitolul 5

5. Concluzii şi dezvoltări ulterioare

Am arătat că este posibil să obţinem un model de recunoaştere facială cu o acurateţe foarte mare pornind de la un set de date construit special pentru această aplicaţie. Folosirea unei reţele convoluţionale antrenate pe un alt set de date folosind o funcţie de cost triplă şi trecerea rezultatului printr-un clasificator elementar precum SVM sau KNN au dat rezultate foarte bune în antrenarea unui model diferit. În plus, am observat că sunt suficiente doar 70 poze pe actor pentru a obţine acurateţea de 99,73%, restul de 30 fiind folosite pentru testare.

Primul lucru care trebuie îmbunătăţit în dezvoltările ulterioare este numărul relativ mic de actori pe care aplicaţia îi poate recunoaşte. Am arătat că pentru a obţine o acurateţe bună ne sunt suficiente cel puţin 70 pe poze pe individ. Găsirea acestui număr de poze pentru o persoană celebră este destul de simplă, deşi toată munca care trebuie depusă pentru a adăuga un număr mare de actori este destul de consumabilă din punct de vedere al timpului şi monotonă. În plus, adăugarea unei prag pe care nu ar trebui să îl depăşească distanţa dintre imaginea propusă şi persoana prezisă îi oferă modelului posibilitatea de a spune dacă actorul introdus pentru a fi recunoscută apare în setul de antrenare sau nu. Decât o predicţie proastă este mai bine să-i spunem utilizatorului că nu putem recunoaşte acel actor. Totuşi, valoarea acestui prag trebuie găsită astfel încât acurateţea reţelei să nu scadă prea mult, introducerea unei noi clase (persoană necunoscută) influenţând şi predicţiile pe setul testat în trecut.

Un alt lucra care ar putea fi luat în calcul este numărul relativ mic de funcţionalităţi ale aplicaţiei web. Pe viitor se pot implementa lucruri care pot care să asigure o mai bună comunicare între utilizatori, acest lucru fiind atrăgător pentru cei care doresc să discute cu mai multe persoane pe seama unor filme. În acestă direcţie se poate crea un forum în care utilizatorii pot pune întrebări, raporta eventuale probleme ale aplicaţiei sau discuta anumite subiecte. În plus, funcţia de editare a profilului, adăugarea pozelor de avatar şi a descrierilor, precum şi posibilitatea de exportare a listei filmelor favorite oferă utilizatorilor posibilitatea să petreacă mai mult timp pe aplicaţie şi să distribuie uşor lista preferinţelor altor persoane.

Funcţia de favorit a filmelor poate fi împărţită în 2: filme care doresc să le văd în viitor şi filme pe care le-am văzut şi vreau să le am salvate în cazul în care vreau să le revăd

în viitor. Un alt ecran care ar trebui adăugat este unul în care utilizatorul poate vedea o listă a recenziilor date de el.

Nu în ultimul rând, o versiune mobilă a aplicaţiei ar putea creşte numărul utilizatorilor cu mult. Un exemplu este acela în care observi o reclamă la un televizor şi vrei să afli cine este persoana aceea. Atunci ai posibilitatea să faci o poză cu camera video a telefonului şi să o încarci în aplicaţie pentru a afla identitatea actorului.

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] https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

[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

[20] https://maven.apache.org

[21] https://www.chartjs.org/