UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

FaceFinder – recunoaștere facială folosind descriptorul POEM

propusă de

Adriana - Simona Ursachi

Sesiunea: *februarie, 2020*

Coordonator științific

Lect. dr. Anca Ignat

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

FaceFinder – recunoaștere facială folosind descriptorul POEM

Adriana – Simona Ursachi

Sesiunea: februarie, 2020

Coordonator științific

Lect. dr. Anca Ignat

Avizat,

Îndrumător Lucrare de Licenţă

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conţinutului lucrării de licenţă

Subsemnatul(a) ……

domiciliul în ……..

născut(ă) la data de ……., identificat prin CNP ……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …… specializarea ……., promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*FaceFinder – recunoaștere facială folosind descriptorul POEM*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iaşi, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *data*

Absolvent *Prenume Nume*

_____

(semnătura în original)

Cuprins

Introducere

Conceptul de recunoaștere facială a devenit tot mai răspândit în întreaga lume și este utilizat în tot mai multe aplicații din jurul nostru. Un sistem de recunoaștere facială este o tehnologie care ajută la identificarea sau verificarea unei persoane dintr-o imagine sau dintr-un video.

Pentru această lucrare de licență am dezvoltat o aplicație desktop care exemplifică acest concept, și anume identificarea unei persoane dintr-o anumită fotografie. Aplicația pune la dispoziție utilizatorului o interfață prin care acesta poate parcurge și modifica conținutul colecției de imagini, respectiv posibilitatea de a încărca o poză pentru a identifica persoana din imagine. Am ales pentru acest proiect să implementez un algoritm de recunoaștere facială deoarece poate fi ușor încorporat într-o multitudine de aplicații si sisteme, folosit pentru autentificare, securitate și multe alte scopuri.

De-a lungul timpului tehnicile de recunoaștere facială s-au îmbunătățit considerabil, în ultimul deceniu fiind implementate o multitudine de sisteme de recunoaștere facială, fiecare folosind metode și algoritmi diferiți.

Recunoașterea facială se realizează în mai multe etape: detectarea feței din imagine și normalizarea acesteia, reprezentarea caracteristicilor feței prin accentuarea anumitor aspecte și, în cele din urmă, clasificarea, care verifică dacă fața extrasă se potrivește cu o alta, prin compararea caracteristicilor faciale, extrase în pasul anterior, cu o colecție de date existentă. Pentru etapele de detectare și clasificare am folosit în mare parte algoritmii implementați în biblioteca *OpenCV,* iar pentru reprezentare am utilizat descriptorul POEM ("*Patterns of Oriented Edge Magnitudes*" (1) – "*Șabloane de orientări ale magnitudinilor*"), care este punctul central al acestei lucrări.

În algoritmul de recunoaștere facială implementat, pentru fiecare pixel din imagine se construiește o caracteristică POEM. Aceasta este calculată prin acumularea unei histograme locale a orientărilor gradienților ce include informațiile referitoare la pixelii vecini. Odată ce această reprezentare a fost calculată, pentru clasificare am ales să folosesc distanța dintre histogramele de caracteristici, și anume distribuția Chi-Square. Pentru testare am utilizat două

colecții de imagini, una dintre ele fiind baza de date FERET (*Tehnologia de Recunoaștere Facială*) (2).

Din punct de vedere tehnic, această aplicație este realizată în limbajul de programare Java. Pentru dezvoltarea interfeței am folosit platforma JavaFX, o platformă software folosită pentru crearea de aplicații, care pot fi rulate pe o gamă largă de dispozitive, respectiv sisteme de operare. Pentru manipularea imaginilor am folosit OpenCV, o bibliotecă care conține funcții specializate în prelucrarea imaginilor și a surselor video, precum și algoritmi de învățare automată.

Lucrarea este structurată în cinci capitole, fiecare având un număr diferit de subcapitole. În primul capitol voi descrie mai pe larg conceptul de recunoaștere facială, precum și câteva abordări anterioare ale acestuia. În cel de-al doilea se află o detaliere completă a algoritmului de recunoaștere folosit, iar în cel de-al treilea se găsesc funcționalitățile aplicației pe care am dezvoltat-o. În final, în ultimele două capitole este prezentată arhitectura aplicației, respectiv detalii legate de implementare și testarea aplicației.

Contribuții

În realizarea lucrării de licență m-am ocupat de documentare în domeniul recunoașterii faciale, de planificarea arhitecturii aplicației, precum și de determinarea unor tehnologii potrivite pentru a putea fi folosite în dezvoltarea aplicației. De asemenea, am implementat un algoritm de recunoaștere facială, bazat pe construirea unui vector de caracteristici a unei imagini, folosind descriptorul POEM.

Am încercat să găsesc modalități cât mai eficiente de stocare și prelucrare a datelor, colecțiile de imagini având un volum destul de mare de date, timpul de procesare al acestora fiind ridicat. Totodată m-am ocupat de testarea algoritmului si am realizat documentația lucrării.

1 Recunoașterea facială

Recunoașterea facială este un proces de identificare și verificare a unei fețe detectate într-o imagine, sursă video sau orice conținut ce provine de la o cameră, prin potrivirea cu una din colecția de fețe ce sunt cunoscute sistemului. Această potrivire se realizează prin compararea unor trăsături selectate și calculate ale fețelor.

1.1 Descrierea problemei

Această tehnologie este foarte des folosită datorită beneficiilor pe care le aduce. Sistemele care folosesc recunoașterea facială pot fi folosite în scopuri de supraveghere, precum căutarea unor infractori sau chiar a unor persoane dispărute. Pe lângă faptul că recunoașterea facială se utilizează pentru identificarea persoanelor, aceasta poate fi folosită și pentru autentificarea la anumite dispozitive sau aplicații. Un avantaj îl constituie faptul că este mult mai rapidă și mai practică decât alte metode de deblocare a telefonului, și în același timp, șansele ca altă persoană să îți acceseze telefonul sunt aproape inexistente.

Interesul pentru recunoașterea facială fiind mereu în creștere, s-au dezvoltat mulți algoritmi pentru îndeplinirea acestui scop. Există multe abordări pentru această problemă, cele mai principale fiind:

1. Analiza componentelor principale ("*Principal Component Analysis*" - PCA) (3) – reprezintă o procedură statistică eficientă de extragere a trăsăturilor feței folosind transformări ortogonale.
2. Analiza discriminantului linear ("*Linear Discriminant Analysis*" - LDA) (3)– este de asemenea o abordare statistică folosită "*pentru a găsi combinații liniare ale proprietăților ce caracterizează sau separă două sau mai multe clase de obiecte sau evenimente.*" (4)
3. *Elastic Bunch Graph Matching* (EBGM) (3)– este un algoritm pentru recunoașterea obiectelor bazat pe o reprezentare grafică a imaginilor.

1.2 Abordări anterioare

La momentul actual există multe aplicații și sisteme în jurul nostru care folosesc algoritmi de recunoaștere facială. O astfel de aplicație este *Facebook*, fiind una dintre cele mai cunoscute și utilizate rețele de socializare din lume. Aceasta oferă utilizatorilor o setare de recunoaștere facială care construiește un model pentru fiecare persoană pe baza fotografiilor în care este etichetată, acest model fiind folosit în compararea cu alte fotografii sau clipuri video pentru a recunoaște persoana din imagine.

O altă rețea de socializare care folosește acest concept este *Snapchat*. Aplicația detectează fața unei persoane în fotografie și permite realizarea unor modificări ale imaginii, precum schimbarea fețelor între două persoane sau adăugarea unor filtre de amuzament pe fața persoanei.

Recunoașterea facială se mai întâlnește și în anumite servicii de securitate. Statele Unite Ale Americii "*deține unul dintre cele mai mari sisteme de recunoaștere facială din lume cu o bază de date ce conține 117 milioane de cetățeni americani adulți*"[1]. Sistemul este folosit atât pentru anumite investigații, cât și pentru acordarea vizelor cu scopul de a evita accesul teroriștilor sau infractorilor. Autoritățile își iau informațiile necesare pentru baza de date din permisele de conducere, vizele deja acordate sau închisori.

[1] https://en.wikipedia.org/wiki/Facial_recognition_system#United_States

2 Algoritmul de recunoaștere facială

Orice algoritm de recunoaștere facială, indiferent de metodele utilizate, se face prin următorii pași:

1. Detectarea și extragerea feței dintr-o sursă digitală, urmată de normalizarea ei;
2. Extragerea trăsăturilor faciale sub forma unui vector de caracteristici unici ai feței extrase;
3. Compararea rezultatelor obținute la pasul anterior cu informațiile deja cunoscute pentru o clasificare cât mai precisă.

În cele ce urmează voi prezenta mai pe larg pașii enumerați anterior, explicând în particular algoritmul pe care l-am ales pentru acest proiect de licență.

2.1 Detecția feței

Detecția fețelor este tehnologia care se ocupă de identificarea prezenței unor chipuri umane într-o fotografie sau sursă video. De asemenea, această tehnologie determină locația și dimensiunile fețelor, ignorând orice alt detaliu din sursa digitală respectivă, precum copaci, obiecte sau alte corpuri.

Totodată, detecția fețelor poate fi privită și ca un caz general de asociere al unui obiect la o clasă, ce se ocupă cu determinarea dimensiunilor și coordonatelor tuturor obiectelor ce aparțin acelei clase dintr-o imagine dată.

În recunoașterea facială, detecția feței este necesară pentru ca algoritmul folosit să fie aplicat doar pentru partea imaginii ce conține fața, astfel comparându-se la final doar caracteristicile fețelor, lăsând la o parte restul imaginii. Pentru ca recunoașterea facială să fie făcută într-un mod cât mai eficient, și să dea rezultate cât mai bune, trebuie în primul rând ca detecția feței din imagine să fie cât mai precisă. Mai jos, în *Figura 1*, putem observa un exemplu de detecție a feței greșită, deoarece nivelul de încadrare a feței în chenarul respectiv este destul de scăzut, iar în *Figura 2* se poate vedea chenarul perfect centrat pe fața persoanei.

Figura 1 - Detecție greșită

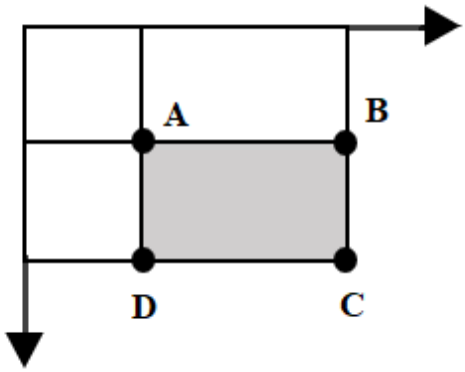
Figura 2 - Detecție corectă

Există o multitudine de tehnici care pot fi folosite pentru localizarea fețelor, fiind împărțite în mai multe categorii în funcție de diferiți factori. Unele dintre cele mai întâlnite tehnici pentru detectarea fețelor sunt:

- "*Găsirea fețelor în imagini cu un fundal controlat;*
- *Găsirea fețelor în funcție de culoare;*
- *Găsirea fețelor în funcție de mișcare;*
- *Combinarea unor tehnici de mai sus;*
- *Găsirea fețelor în medii necondiționate*." (5)

Algoritmii de detectare a feței încadrați în ultima categorie din clasificarea de mai sus sunt probabil cei mai eficienți și au fost utilizați cu succes de-a lungul timpului. Una dintre cele mai cunoscute tehnici folosite este algoritmul Viola-Jones, publicat de Paul Viola și Michael Jones în anul 2001 (6), un algoritm care se folosește de clasificatori de tip cascadă bazați pe caracteristici Haar, caracteristicile imaginii digitale folosite în recunoașterea obiectelor. Am ales să folosesc acest algoritm, care este implementat în librăria OpenCV, pentru că este unul dintre cei mai eficienți din punct de vedere al complexității.

În primul pas al algoritmului se construiește o imagine integrală, care este o imagine cu aceeași dimensiune cu imaginea sursă. Această imagine se obține prin schimbarea valorilor fiecărui pixel din imaginea originală cu o suma care să fie egală atât cu valoarea tuturor pixelilor de deasupra lui, cât și cu toți pixelii din stânga lui. Imaginea integrală reduce semnificativ volumul de operații efectuate, deoarece permite extragerea trăsăturilor dintr-o anumită regiune, nu din toată imaginea.



Astfel, pentru orice regiune dreptunghiulară din imagine, suma se calculează folosind doar patru valori. De exemplu, valoarea dreptunghiului din *Figura 3* este egală cu valoarea calculului D - (B + C) + A.

Figura 3 - Exemplu de dreptunghi

Una dintre contribuțiile cheie realizate în algoritm a fost introducerea unui set de caracteristici simple de utilizat, și anume caracteristicile de tip Haar. Algoritmul analizează o porțiune a imaginii folosind trăsături formate din două sau mai multe dreptunghiuri. Fiecare astfel de trăsătură, și anume cele din *Figura 4*, este calculată ca fiind diferența dintre suma pixelilor aflați în dreptunghiurile albe și suma pixelilor aflați în dreptunghiurile negre.

Figura 4 - Caracteristici Haar (6)

În continuare se folosește un algoritm de clasificare, și anume AdaBoost, ce primește un set de trăsături și un set de antrenament cu imagini pozitive și negative pentru detecția facială.

La finalul acestei etape voi avea o nouă imagine ce va conține doar fața persoanei detectate din imaginea sursă. De asemenea, este nevoie de o metodă de normalizare a imaginii create, pentru ca toate fețele extrase din colecția de date cu care va fi comparată să urmeze același tipar. Eu am optat pentru scalarea fețelor la o anumită dimensiune.

Astfel, pentru un control mai bun și o cunoaștere precisă a datelor ce urmează a fi calculate în pașii următori, am redimensionat fața detectată pentru a avea lățimea egală cu 64 și înălțimea cu 128.

2.2 Reprezentarea caracteristicilor – descriptorul POEM

În domeniul recunoașterii faciale un pas foarte important este reprezentarea caracteristicilor feței, deoarece o reprezentare bună duce la o clasificare destul de ușoară, având rezultate optime. Pentru această etapă am ales să folosesc descriptorul POEM („*Patterns of Oriented Edge Magnitudes*" – „*Șabloane de orientări ale magnitudinilor*")) (1), un descriptor care promite o recunoaștere cât mai complexă și eficientă.

Acest descriptor se folosește de proprietățile pixelilor din imagine, precum orientările și magnitudinile acestora, pentru a calcula un vector de caracteristici. În procesul de extragere a descriptorului POEM al unei fețe se disting trei etape: calculul imaginii gradient și discretizarea orientărilor, calculul acumulării de magnitudini și calculul descriptorului pentru fiecare pixel.

2.2.1 Imaginea gradient și discretizarea orientărilor

Primul pas în extragerea caracteristicii POEM a feței detectate anterior, este calculul gradientului imaginii. Acesta reprezintă o schimbare direcțională a intensității sau culorii într-o imagine. În cazul nostru avem nevoie de calculul gradientului pentru a determina schimbările intensităților în direcția x (orizontală), respectiv în direcția y (verticală). Pentru acest calcul am folosit operatorul Sobel (7), care calculează o aproximare a gradientului unei imaginii cu tonuri gri, în cele două direcții.

Pentru a calcula cele două aproximări menționate mai sus, notate G_x și G_y, se folosesc două kernel-uri de dimensiune 3x3(cele din *Figura 5* și *Figura 6*), unul pentru direcția orizontală si unul pentru cea verticală.

-1	0	+1
-2	0	+2
-1	0	+1

Figura 5 - Kernel-ul pentru schimbările din direcția orizontală (8)

+1	+2	+1
0	0	0
-1	-2	-1

Figura 6 - Kernel-ul pentru schimbările din direcția verticală (8)

Astfel, G_x și G_y sunt calculate prin aplicarea unei operații de convoluție[2] între imaginea sursă și cele două kernel-uri menționate anterior. Convoluția este o operație matematică simplă, care este fundamentală pentru mulți operatori comuni de procesare a imaginilor.

În continuare, G_x și G_y pot fi combinate pentru a calcula magnitudinea absolută a gradientului în fiecare punct, respectiv orientarea acelui gradient. Magnitudinea gradientului se calculează cu ajutorul formulei: $G = \sqrt{G_x^2 + G_y^2}$ (7), iar orientarea gradientului prin formula: $\theta = \tan^{-1}\frac{G_y}{G_x}$ (8).

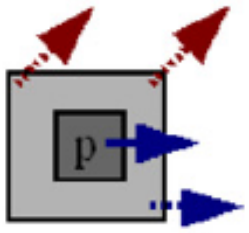
În figura alăturată este calculată imaginea gradient pentru fața extrasă din *Figura 2* din subcapitolul *2.1 Detecția* feței.

Figura 7 - Imagine gradient

Următorul pas este cuantificarea[3] orientărilor gradienților. Astfel, orientarea fiecărui pixel din imagine este discretizată pe intervalul [0, Π], pentru o reprezentare fără semn, sau pe intervalul [0, 2Π], pentru o reprezentare cu semn.

Pentru a realiza această discretizare, se alege un număr de orientări, iar intervalul specificat mai sus se împarte în mai multe subintervale egale, în funcție de acest număr. Apoi, se stabilește cărui subinterval îi aparține orientarea fiecărui pixel. De exemplu, dacă orientarea unui pixel aparține primului subinterval acestuia i se va atribui prima orientare, dacă aparține celui de-al doilea subinterval, i se va atribui cea de-a doua orientare și tot așa pentru fiecare pixel.

[2] Convoluție = întoarcere, răsucire
[3] Cuantificare = stabilire a valorilor discrete(discontinue) pe care le poate lua o mărime fizică



În figura alăturată orientarea discretizată a unui pixel oarecare p este reprezentată de săgeata continuă emisă din pixel.

Figura 8 - Orientările magnitudinilor pixelilor (1)

Prin urmare, la finalul acestei etape, fiecărui pixel i se va atribui un vector ce conține câte două elemente: magnitudinea calculată anterior și orientarea discretizată.

2.2.2 Acumularea magnitudinilor

În cadrul următorului pas informațiile referitoare la pixelii vecini (săgețile discontinue din *Figura 8*) se acumulează prin calculul unei histograme locale de orientări ale gradienților. Astfel, pentru fiecare pixel, "*caracteristica va fi acum un vector cu m elemente, unde m reprezintă numărul de orientări*" (1).

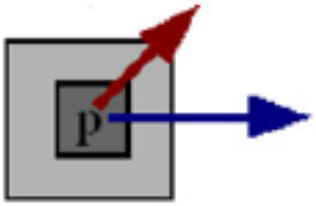


Figura 9 - Magnitudinile acumulate (1)

O histogramă a orientării gradienților ("*Histogram of oriented gradients*" - HOG (9)) este un descriptor de caracteristici care numără aparițiile orientărilor gradienților în anumite porțiuni ale imaginii. Aceste porțiuni ale imaginii se numesc celule, fiind mici regiuni care conțin mai mulți pixeli. Am ales dimensiunea celulelor ca fiind 3x3, adică fiecare celulă să fie alcătuită din nouă pixeli.

Pentru descriptorul POEM, numărul de apariții din calculul histogramei este înlocuit de valorile magnitudinilor pixelilor. Așadar, pentru fiecare pixel din imagine se calculează câte o histogramă acumulând magnitudinile celor opt pixeli vecini din celulă într-un vector, în funcție de orientările pe care le au pixelii. Deci, dacă un vecin al unui pixel oarecare are orientarea cu numărul unu, pe prima poziție a vectorului ce reprezintă histograma pixelului se va adăuga valoarea magnitudinii acelui vecin. În acest mod se completează tot vectorul, adăugând în poziția corespunzătoare orientării, magnitudinea pixelului. În *Figura 10* se află un exemplu de calcul al unei histograme de acest gen pentru un pixel oarecare p.

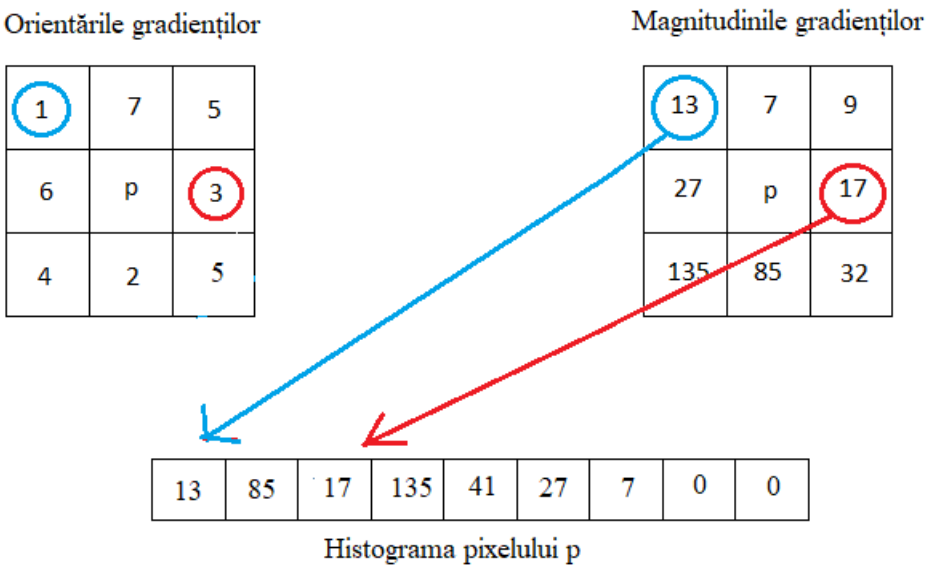


Figura 10 - Modul de calcul al unei histograme pentru un pixel p

2.2.3 Calculul vectorului de caracteristici

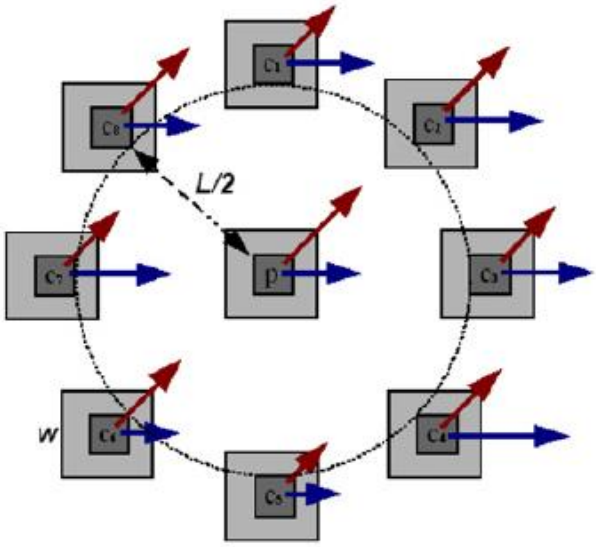


Figura 11 - Calculul similarității lor bazat pe magnitudinile orientate într-un bloc, folosind operatorul LBP (1)

Într-un final, "*magnitudinile acumulate anterior sunt codificate folosind operatorul LBP ("Local Binary Patterns" – Modele locale binare), aplicat pe grupuri mai mari de celule, numite blocuri.*" (1)

LBP este un operator simplu de textură, dar foarte eficient, care etichetează fiecare pixel al unei imagini printr-un număr în reprezentare binară, obținut din informațiile pe care le oferă pixelii vecini. Datorită simplității computaționale și eficienței sale, LBP este utilizat frecvent în

multe aplicații din domeniul viziunii computerizate. În *Figura 12* se află un exemplu de aplicare a acestui operator pentru un pixel oarecare. După cum se poate observa, se compară valorile pixelilor vecini cu valoarea pixelului în cauză și se setează o valoare binară corespunzătoare: dacă valoarea este mai mare se setează cu 1, altfel cu 0. Astfel, se obține un număr binar ce se va transforma în decimal și va reprezenta noua valoare a pixelului.

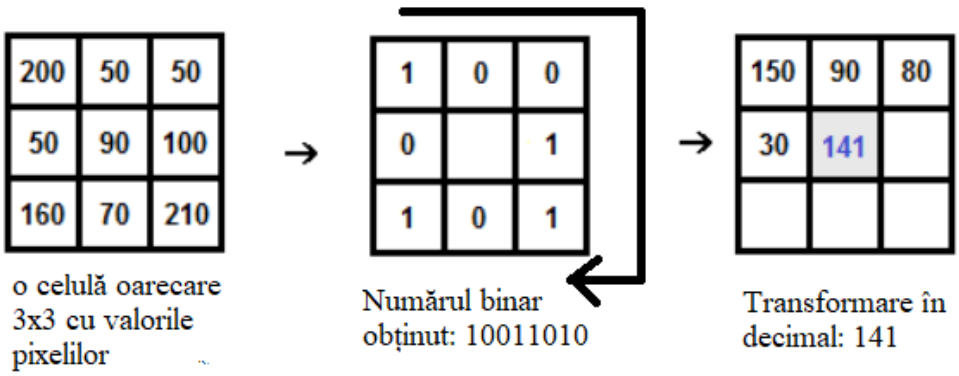


Figura 12 - Modul de aplicare LBP pentru un pixel

Având informațiile pentru fiecare pixel (vectorii cu câte nouă elemente), calculate în etapa precedentă, vom forma nouă imagini care vor conține valorile de pe pozițiile corespunzătoare din acești vectori. Mai exact, prima imagine va conține toate valorile ce se află pe prima poziție din toți vectorii specifici pixelilor, în a două imagine se vor afla valorile din a doua poziție și tot așa.

Apoi se va aplica operatorul LBP fiecărui pixel din cele nouă imagini, așa cum este exemplificat mai sus, obținându-se nouă imagini cu valori LBP. Pentru fiecare imagine de acest gen se va calcula câte o histogramă locală în modul clasic, rezultând nouă vectori. În consecință reprezentarea finală a caracteristicilor feței detectate sunt cei nouă vectori rezultați, ce urmează să fie comparați cu restul caracteristicilor colecției de imagini.

2.3 Clasificarea

Odată ce a fost calculată reprezentarea caracteristicilor imaginii, aceasta trebuie comparată cu colecția de date cunoscută pentru a găsi o potrivire. Ca și metodă de clasificare am

optat pentru cea în care se găsește vecinul cel mai apropiat (1-NN), folosindu-mă de distanța Chi-Square.

Clasificarea vecinului cel mai apropiat ("*k-nearest neighbors*" (8)) este o metodă care se bazează pe calcularea distanțelor dintre datele de testare și toate datele de pregătire pentru a identifica vecinii apropiați și pentru a produce rezultatul de clasificare. Ca și funcție de calcul al distanței dintre două seturi de date am folosit distanța Chi-Square care ajută la compararea histogramelor prin formula: $d(H_1, H_2) = \sum_I \frac{(H_1(I)-H_2(I))^2}{H_1(I)}$ (10), unde H_1 și H_2 sunt histogramele pentru care se calculează distanța, iar $I = \overline{0,n}$; unde *n* este numărul de elemente ale celor două histograme.

Așadar, am însumat distanțele dintre toate cele nouă histograme din reprezentarea caracteristicilor, cu histogramele fiecărei imagini din baza de date, rezultând distanța finală. Cea mai mică distanță găsită constituie potrivirea feței cu persoana din imaginea pentru care s-a calculat acest rezultat.

3 Specificații funcționale

Scopul meu pentru acestă lucrare de licență a fost crearea unei aplicații desktop care să ajute la identificarea persoanelor din fotografii, implementând un algoritm de recunoaștere facială, bazat pe un descriptor eficient. Totodată, am vrut ca aplicația mea să ofere un sistem complex de administrare al colecției de imagini, care să fie utilizat în mod special în scopul de testare al algoritmului.

În continuare voi prezenta funcționalitățile aplicației, precum și opțiunile pe care le are un utilizator al acesteia.

3.1 Fereastra de căutare

Pentru ca utilizatorul să poate parcurge și vizualiza colecția de fișiere în care sunt stocate resursele aplicației am implementat un panou de căutare paginabil care permite accesarea și filtrarea tuturor colecțiilor de fotografii(*Figura 13 - Browse Panel*). De asemenea, aplicația permite mai multe operații asupra acestor seturi de date:

- Crearea unei noi colecții, implică crearea unui nou folder pe disc;
- Ștergerea unei colecții împreună cu tot conținutul acesteia;
- Crearea unei imagini, implică alegerea unei imagini din sistemul de fișiere local;
- Filtrarea tuturor imaginilor după un anumit cuvânt cheie;
- Ștergerea definitivă a unei imagini;
- Accesarea unei colecții și a unei imagini.

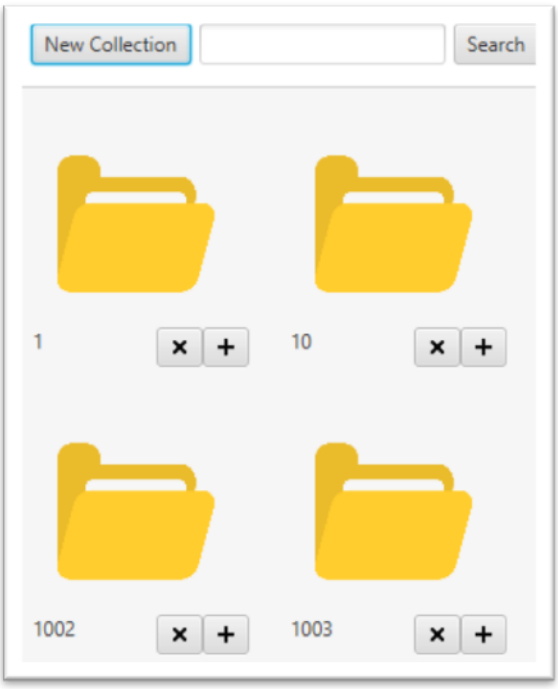


Figura 13 - Browse Panel

3.2 Administrarea imaginilor

Un utilizator al aplicației are posibilitatea de a aplica mai multe operații asupra colecției de imagini prin intermediul metodelor oferite de opțiunea ***Images*** din cadrul meniului din *Figura 14*. Astfel, acest sistem de administrare al imaginilor oferă următoarele posibilități:

- *Resetarea tuturor procesărilor*. Acest lucru se poate face în cazul în care s-a produs o eroare în modul în care imaginea era stocată pe disc sau atunci când algoritmul de recunoaștere facilă este modificat și este dorită o înlocuire totală a tuturor procesărilor.
- *Convertirea tipurilor imaginilor*. Java, și în particular JavaFX, nu are un mod de a reprezenta grafic imagini de tipul *".tif"*. O mare parte a imaginilor folosite în testare erau de acest tip, deci, pentru o bună desfășurare a proceselor de verificare, am optat pentru folosirea unei biblioteci care oferă posibilitatea de a citi și rescrie imagini dintr-un anumit tip în altul. La rularea unei astfel de comenzi, sistemul de fișiere va fi parcurs și fiecare imagine cu extensia ".tif" va fi convertită într-o imagine ".bmp".
- *Actualizarea imaginilor*. În cazul în care s-au adăugat imagini sau colecții noi după deschiderea aplicației, este posibilă o comandă de actualizare. Aceasta va prelua toate resursele noi și le va procesa, adăugându-le în memorie.
- *Reprocesarea imaginilor*. Similar cu actualizarea, această comandă rulează algoritmul de detecție pentru imaginile neprocesate.

3.3 Procesarea imaginilor

Prin optarea pentru opțiunea ***Process*** din cadrul meniului din *Figura 14*, utilizatorului îi este oferită posibilitatea de a încărca o fotografie din sistemul dispozitivului pentru identificarea persoanei din aceasta. În momentul procesării unei imagini, se aplică algoritmul de recunoaștere pe aceasta, iar rezultatul obținut se compară cu datele existente stocate în memorie pentru a se găsi o potrivire. Dacă această potrivire s-a găsit, utilizatorul va primi ca și rezultat numele colecției din care face parte acea persoană.

4 Arhitectura aplicației

Aplicația este realizată cu ajutorul platformei JavaFX, fiind formată din mai multe module. În acest capitol voi prezenta felul în care am structurat aplicația desktop, ierarhizarea elementelor în funcție de nevoile și funcționalitățile ei, precum și modul de stocare al datelor și al colecției de imagini.

4.1 Meniul aplicației

Meniul aplicației este situat în partea superioară a ferestrei. În JavaFX, pentru a crea un meniu personalizabil este necesară extinderea clasei *MenuBar*. Aceasta controlează ierarhizarea componentelor compuse, modul în care un meniu este așezat în fereastră și felul în care acesta interacționează cu datele de intrare ale utilizatorului.

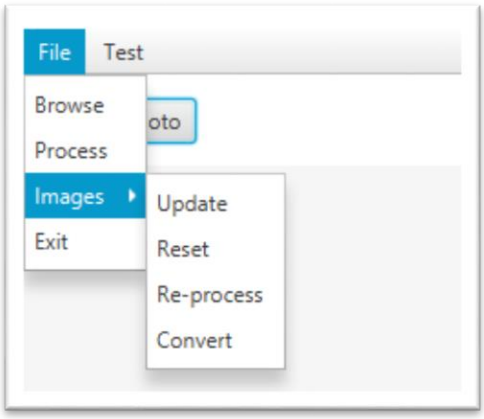


Figura 14 - Meniul aplicației

Un meniu este format din două tipuri de componente: *Menu* și *MenuItem*. *MenuItem* reprezintă o opțiune finală pe care o poate alege un utilizator, iar *Menu* este o componentă compusă, formată din mai multe entități de tip *MenuItem* sau chiar alte entități *Menu*. Astfel putem realiza o ierarhizare specifică a meniului bazată pe prioritatea și apartenența tuturor funcționalităților aplicației. Ierarhia pe care am ales-o pentru aplicație este exemplificată în *Figura 15*.

Meniul este format din două componente de tip *Menu*: ***File*** și ***Test***, pe care le putem observa în *Figura 14*. Opțiunea ***File*** conține comenzi specifice pentru administrarea aplicației:

- ***Browse*** – permite navigarea la pagina unde putem parcurge colecția de imagini și căuta o anumită imagine;

- ***Process*** – face trecerea la pagina de procesare, unde există posibilitatea de încărcare a unei imagini și obținere a unui rezultat prin aplicarea algoritmului de recunoaștere facială;
- ***Images*** – administrarea colecției de imagini, oferind comenzile de actualizare, resetare, reprocesare și convertire explicate în subcapitolul *3.2 Administrarea* imaginilor din capitolul anterior.

Cea de-a doua componentă, ***Test***, reprezintă un portal către funcționalitatea de testare a aplicației.

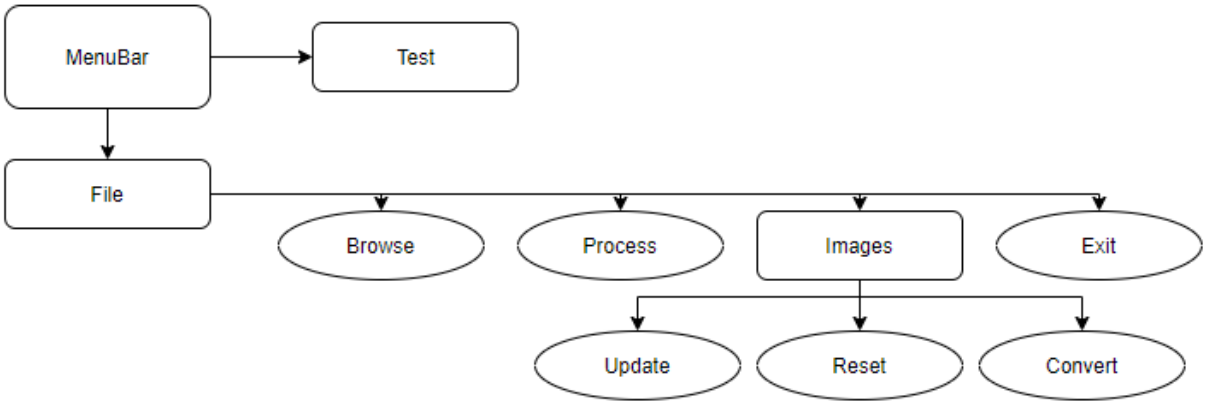


Figura 15 - Ierarhia meniului

4.2 Structura aplicației

Aplicația este structurată pe mai multe straturi ierarhizate după gradul de dependență. Astfel, nivelul cel mai înalt este reprezentat de clasele ce constituie *display*-ul propriu-zis al aplicației și anume interfața. În JavaFX acest lucru este realizat prin extinderea unor clase specifice structurării și reprezentării de date precum:

- *VBox* - utilizat pentru a structura mai multe date pe axa verticală;
- *HBox* - similar *VBox* însă se folosește pe axa orizontală;
- *SplitPane* – utilizat pentru a reprezenta date într-o fereastră ce prezintă un separator central;
- *ScrollPane* – un container ce permite parcurgerea datelor.

Nivelul intermediar este realizat din mai multe clase ce descriu *business*-ul aplicației, precum entitățile ce implementează algoritmul de recunoaștere facială, administratorul de imagini, precum și cel pentru firele de execuție. De asemenea conține și alte clase ce fac legătura între nivelul superior și cel inferior, dar și entități ce ajută la navigarea prin diferite ferestre ale aplicației.

Ultimul nivel, cel inferior, este alcătuit din entitățile de bază ale aplicației, precum reprezentări abstracte ale resurselor (imaginilor și colecțiilor de imagini) și o multitudine de entități ce prezintă comportamente cu uz universal. Acest strat reprezintă temelia tuturor funcționalităților implementate în aplicație, atât pe partea de *business* cât și pe partea de reprezentare a datelor.

În *Figura 16* sunt reprezentate toate cele trei nivele descrise mai sus.

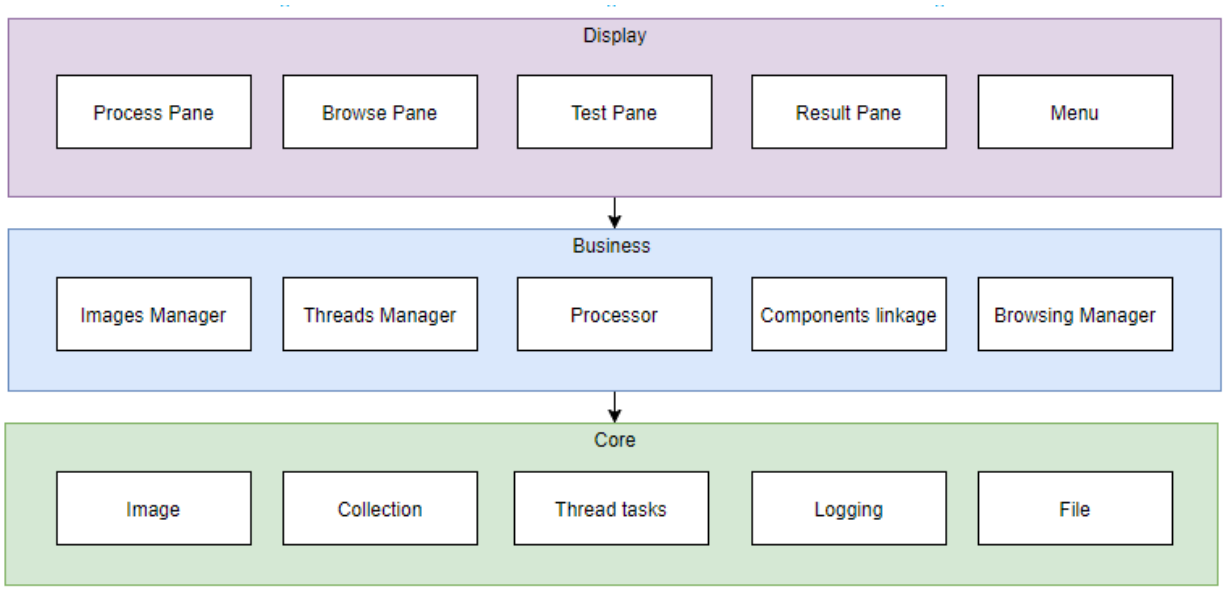


Figura 16 - Structura aplicației pe nivele

4.3 Stocarea datelor.

Resursele aplicației sunt reprezentate de o multitudine de imagini. Acestea sunt stocate strategic într-un sistem de fișiere care ne permite accesul rapid la datele stocate în fișiere cât și administrarea acestora. Fiecare imagine face parte dintr-o colecție, un folder specific unei singure persoane, ce poate cuprinde un număr mare de imagini capturate în ipostaze diferite.

Aplicația se folosește de o serie de funcționalități înglobate într-un *ImageManager* care se ocupă de toate operațiile atribuite imaginilor si colecțiilor. Această clasă este folosită încă de la inițializarea instanței de aplicație pentru a încărca în memorie caracteristicile tuturor fotografiilor aflate în colecție. Astfel, în momentul procesării unei noi fotografii, avem toate datele la dispoziție, fără să mai fie nevoie de o nouă procesare a tuturor resurselor. Așadar, timpul de procesare este scăzut considerabil.

```
public class ImageManager {
    private File root = new File("../Photos/");
    private ArrayList<Collection> collections = new ArrayList<Collection>();
    private App app;

    public ImageManager(App app) {
        this.app = app;
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        initialize();
    }
```

Figura 17 - Clasa ImageManager

Implementarea acestui model de stocare a datelor este detaliată pe larg în subcapitolul *5.2.1 Management-ul* datelor din capitolul următor.

5 Implementare și testare

5.1 Tehnologii folosite

Aplicația este implementată în limbajul de programare Java, un limbaj de programare orientat-obiect de nivel înalt. De asemenea, am folosit bibliotecile OpenCV, pentru lucrul cu imagini, și JavaFX pentru dezvoltarea interfeței grafice. Am implementat mai multe clase care se ocupă de diferite sarcini din cadrul aplicației, și am realizat testarea algoritmului pentru două seturi de date.

5.1.1 OpenCV

OpenCV este o bibliotecă de funcții și algoritmi specializați în domeniul viziunii computerizate. Librăria a fost implementată pentru a putea fi folosită în cadrul mai multor sisteme de operare, precum Windows, Linux sau Mac OS X, și are interfețe pentru limbaje de programare ca și C/C++, Python, Java.

Figura 18 - Logo OpenCV

Scopul librăriei este de a oferi utilizatorilor o modalitate de procesare a imaginilor ușor de folosit, care să ajute la dezvoltarea rapidă a aplicațiilor complexe. Pe lângă funcțiile de manipulare a imaginilor, biblioteca conține și un set complet de funcții și algoritmi din domeniul învățării automate.

În proiectul meu am utilizat OpenCV pentru manipularea și prelucrarea imaginilor. Am folosit funcții pentru citirea și redimensionarea unei imagini, pentru calculul imaginii gradient, a magnitudinilor și orientărilor acestora, precum multe altele. De asemenea, am utilizat cascadele Haar și LBP, implementate în librărie, pentru detecția facială si funcțiile de calcul și comparare a histogramelor. Mai exact, am folosit această bibliotecă pentru implementarea clasei ce se ocupă cu procesul de recunoaștere facială. Această clasă se numește *Processor* și conține implementarea algoritmul descris în capitolul *2 Algoritmul de recunoaștere* facială.

În *Figura 19* se află modul în care am făcut pasul de detecție facială cu ajutorul clasificatorului în cascadă oferit de OpenCV.

```
// Instantierea clasificatorului CascadeClassifier
String xmlFile = "C:\\opencv\\build\\etc\\lbpcascades\\lbpcascade_frontalface.xml";
CascadeClassifier classifier = new CascadeClassifier(xmlFile);

// Detectarea fetei in imaginea sursa
MatOfRect faceDetections = new MatOfRect();
classifier.detectMultiScale(src, faceDetections);

// Extragerea fetei detectate si redimensionarea imaginii
Mat faceImage = new Mat();
Mat resizeimage = new Mat();
for (Rect rect : faceDetections.toArray()) {
    faceImage = src.submat(rect);
    Size sz = new Size(64, 128);
    Imgproc.resize(faceImage, resizeimage, sz);
}
```

Figura 19 - Exemplu de utilizare OpenCV în aplicație

5.1.2 JavaFX

JavaFX este o tehnologie folosită pentru crearea aplicațiilor desktop care poate opera pe diverse sisteme de operare și dispozitive. Aceasta pune la dispoziție dezvoltatorilor un set de pachete grafice și media pentru design-ul, crearea și testarea aplicațiilor de acest gen. Totodată, fiind un API Java, biblioteca oferă posibilitatea utilizării oricărei alte biblioteci și API-uri Java native.

JavaFX oferă o multitudine de colecții de componente pentru crearea de meniuri, butoane, etichete și alte controale care pot fi folosite pentru o dezvoltare ușoară a unei interfețe. Am folosit această platformă pentru construirea interfeței grafice a aplicației, modul în care am folosit componentele bibliotecii găsindu-se în capitolul *4 Arhitectura* aplicației.

5.2 Detalii de implementare

5.2.1 Management-ul datelor

Aplicația are nevoie de un sistem flexibil și global de control al accesului și management-ului resurselor principale, acestea fiind reprezentările caracteristicilor imaginilor din sistemul de fișiere. Un astfel de sistem trebuie să dețină următoarele capabilități:

- Un mod de a procesa și încărca toate rezultatele în memorie;
- Posibilitatea de a accesa orice fotografie în orice moment;

- Posibilitatea de a interoga o imagine pentru a afla colecția din care face parte.

În momentul pornirii aplicației, se creează și un fir de execuție separat de firul de execuție principal. Acesta se ocupă de inițializarea întregului sistem de administrare al imaginilor. Parcurge întregul sistem de fișiere cu imagini, desemnat ca *storage* al aplicației, iar pentru fiecare folder găsit, creează o entitate de tipul *Collection (Figura 20)*.

```
public class Collection {
    private File root;
    private ArrayList<CollectionImage> images;
    private String name;
```

Figura 20 - Modul de definire al entității Collection

Fiecare entitate de tip *Collection* este la rândul ei inițializată iterativ, astfel adresa fizică, care este stocată în variabila *root*, este interogată cu privire la imaginile conținute în directorul respectiv. Fiecare astfel de imagine este asociată la o nouă entitate de tipul *CollectionImage*.

```
public void initialize() {
    for (final File fileEntry : root.listFiles()) {
        CollectionImage image = new CollectionImage(fileEntry.getName(), name);
        image.initialize();
        images.add(image);
    }
}
```

Figura 21 - Procesul de inițializare

Similar, fiecare imagine, sau entitate *CollectionImage*, necesită o inițializare. Aceasta constă în verificarea existenței unui rezultat al procesării imagini respective, mai exact dacă are vectorul de caracteristici calculat. În cazul în care un astfel de rezultat lipsește, se apelează funcția din clasa *Processor*, pentru a executa algoritmul de extragere a caracteristicilor pentru imaginea respectivă. Odată obținut, rezultatul este stocat în memorie.

5.2.2 Management-ul proceselor paralele

Procesarea imaginilor este o operație destul de costisitoare ca și timp, având în vedere cantitatea mare de resurse ce trebuie procesate, timpul total de procesare al unei colecții de dimensiuni mari într-un mod iterativ fiind de ordinul orelor. Totuși, este un proces complet izolat de alte operații, așadar dispunerea procesărilor în paralel a tuturor fotografiilor a fost o prioritate încă de la început.

Inițial am încercat o abordare mai simplă, iterarea unui număr fix de colecții și crearea unui număr egal de fire de execuție. Fiecare fir de execuție avea drept sarcină procesarea unei colecții. După predarea tuturor sarcinilor, fiecare fir de execuție era așteptat, întrucât nu puteam risca crearea unui număr mai mare de *thread*-uri, supraîncărcarea procesorului fiind un risc. După așteptarea fiecărui fir de execuție, iteram prin alt număr fix de colecții care erau repartizate *thread*-urilor. Această rutină era repetată până la terminarea parcurgerii tuturor colecțiilor. În urma acestui proces am observat îmbunătățiri mari în ceea ce privește timpul de execuție al inițializării, totuși era loc de mai bine.

Am observat că anumite colecții erau mai ușor de procesat decât altele, având un număr mai mic de imagini, sau o calitate mai slabă a imaginilor. Algoritmul descris anterior categorisea mai multe colecții într-un singur loc, firele de execuție utilizate fiind forțate să aștepte până la terminarea întregului set de colecții pentru a trece mai departe. Astfel, *thread*-urile care terminau mai repede nu puteau continua separate.

Astfel, am ajuns la ideea de *ThreadManager*, o clasă care se ocupă cu administrarea și sincronizarea firelor de execuție. Astfel, fiecare fir de execuție (*Worker*) are o serie clară de pași pe care să îi urmeze:

- Cere o resursă de la manager;
- Procesează resursa respectivă;
- Trimite rezultatul către manager;
- Verifică dacă mai există resurse ce trebuie procesate. În cazul în care nu mai există resurse, *Worker*-ul se oprește, altfel, revine la primul pas.

Folosind această tehnică nici un fir de execuție nu este suspendat de către un altul, îmbunătățind drastic timpul de execuție.

```
@Override
public void run() {
    while (true) {
        File file = manager.getFile();
        if (file == null) {
            return;
        }
        Collection collection = new Collection(file.getName());
        collection.initialize();
        manager.addCollection(collection);
    }
}
```

Figura 22 - Metoda care se ocupă de administrarea firelor de execuție

În *Figura 22* se află metoda care se ocupă de pașii descriși anterior, iar în *Figura 23* este reprezentat modul de lucru al clasei *ThreadManager*, precum etapele pe care le urmează.

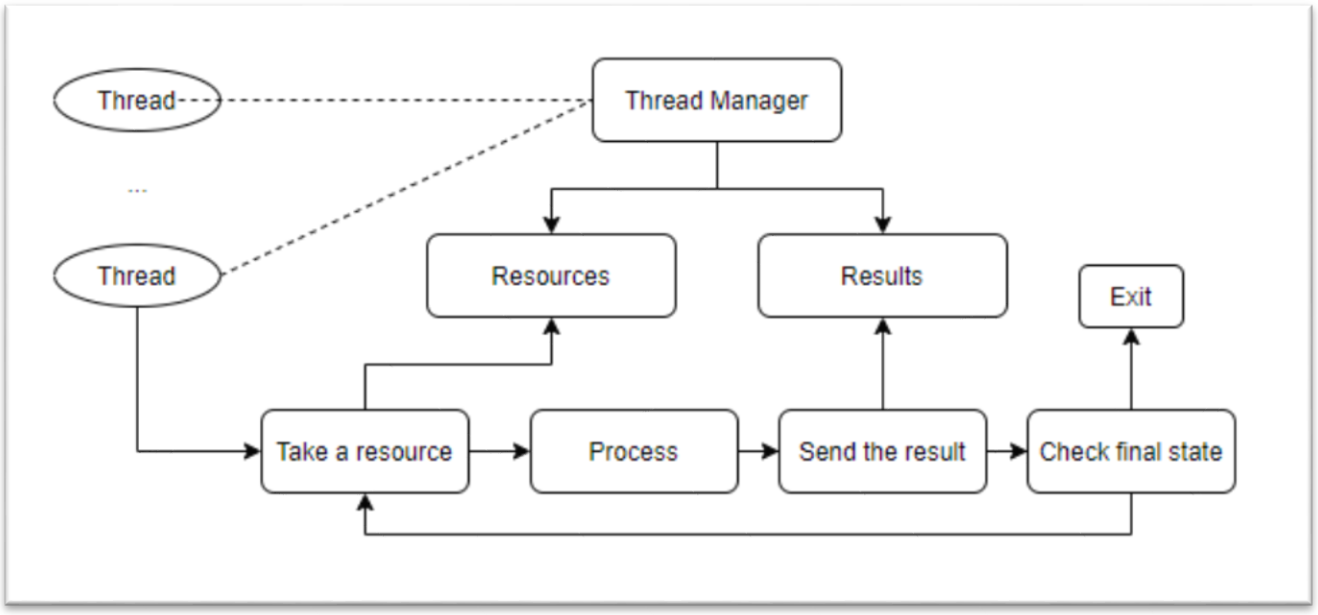


Figura 23 - Modul de lucru al sistemului de administrare al firelor de execuție

5.3 Testarea

Pentru partea de testare al algoritmului de recunoaștere facială am folosit două colecții de imagini, una cu imagini având o calitate mai ridicată, respectiv cunoscuta colecție FERET. De asemenea, pentru a crea niște statistici, precum și pentru îmbunătățirea algoritmului am aplicat o tehnică de validare.

5.3.1 Colecţiile de imagini

Prima colecție conține imaginile a 152 de persoane, fiecare persoană având câte 20 de fotografii în ipostaze diferite. Aceste fotografii au fost făcute cu un fundal verde în spate, distanța dintre aparatul de fotografiat și persoană nefiind foarte mare. Astfel, algoritmul va avea rezultate destul bune pentru acest tip de imagini. Un exemplu de imagine din această colecție se află în *Figura* 2 din cel de-al doilea capitol.

Ce-a de-a doua colecție, FERET, conține fotografiile a aproximativ 1200 de persoane. În acest caz, numărul de fotografii al fiecărei persoane variază, fiecare persoană având minim două fotografii în ipostaze diferite. De exemplu, dacă introducem fotografia din *Figura* 24, una în care persoana are și ochelari de soare, vom obține că s-a găsit o potrivire în directorul 3.

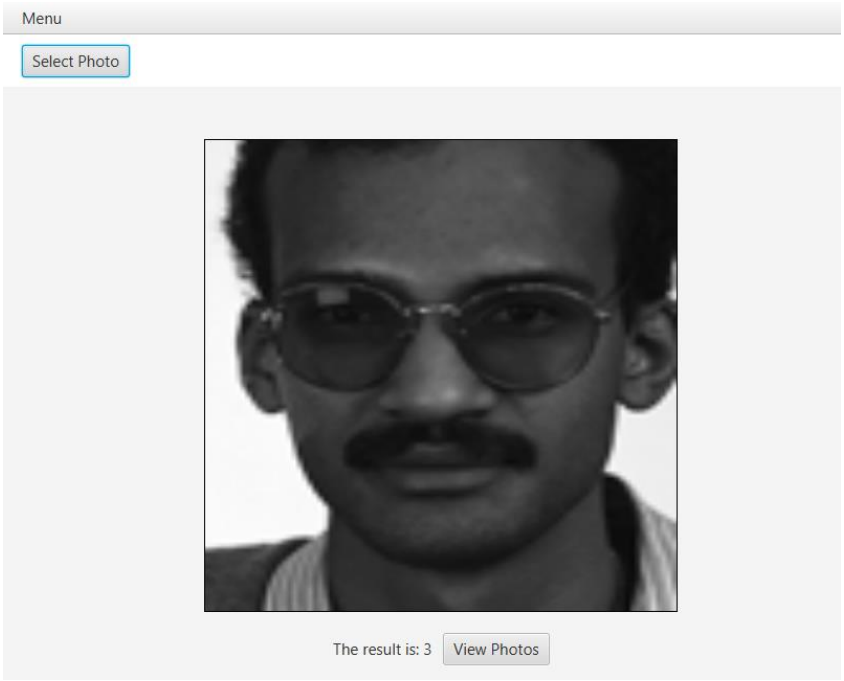


Figura 24 - Procesarea unei imagini pentru a vedea din ce colecție face parte

Dacă vrem să vedem pentru ce colecție s-a găsit potrivirea, alegem opțiunea *View Photos* și vom fi redirecționați la colecția cu numărul 3, și anume cea din *Figura 25*.

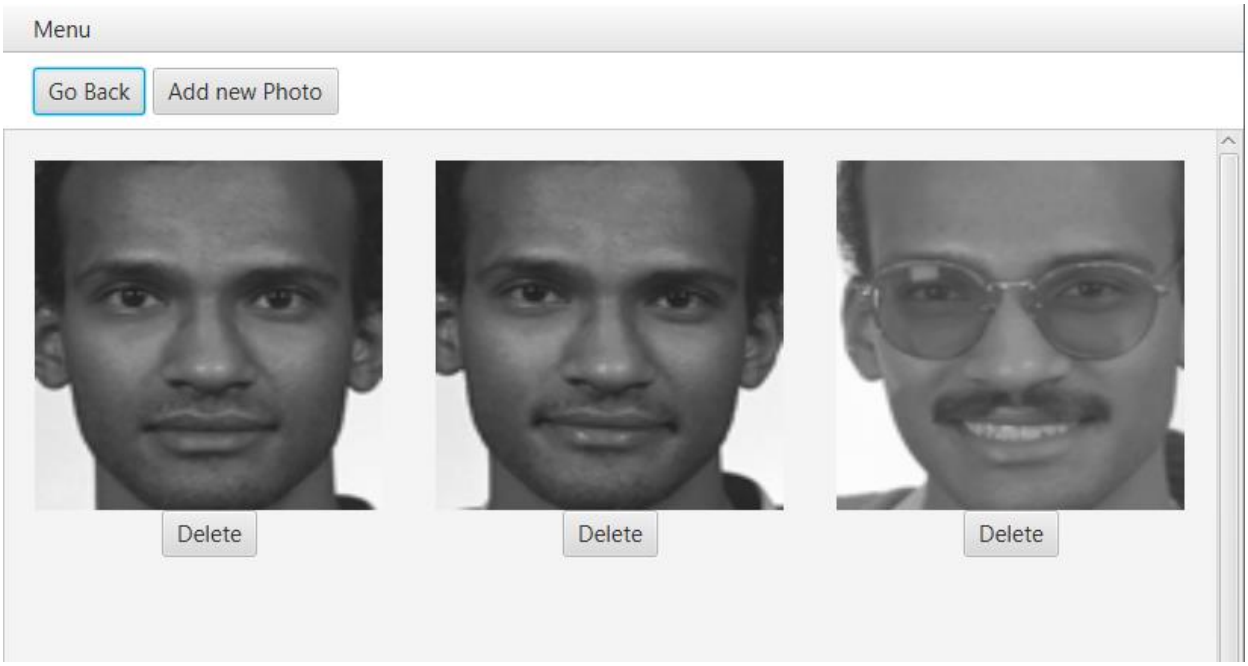


Figura 25 - Directorul în care s-a găsit potrivirea

5.3.2 Statistici

Pentru îmbunătățirea parametrilor folosiți în cadrul algoritmului de recunoaștere facială am folosit tehnica *Leave One Out Cross Validation* ("*validare încrucișată Leave One Out*"), iar pentru testare am folosit prima colecție de imagini. Ca și parametrii din cadrul algoritmului, am verificat teste pentru numărul de orientări discretizate ale magnitudinilor și pentru dimensiunea histogramei finale calculate.

Astfel, din fiecare pereche de imagini din colecție, am scos pe rând câte o fotografie pe care am comparat-o cu restul rămase, pentru diferite valori ale parametrilor și am calculat procente pentru acuratețe și gradul de certitudine. Acuratețea este procentul pentru care algoritmul a clasificat corect instanțele, iar gradul de certitudine reprezintă cât de bine s-a făcut clasificarea.

În tabelele de mai jos se află rezultatele obținute pentru diferite valori ale parametrilor menționați mai sus.

Dimensiunea histogramei	Acuratețea	Gradul de certitudine
15	92%	78%
20	92%	69%
25	92%	60%
50	93%	16%

Se poate observa că pentru o dimensiune mai mare a histogramei acuratețea este mai bună, dar valoarea gradului de certitudine este foarte scăzut. Prin urmare, voi folosi în algoritm o valoare care nu obține o diferență așa de mare precum cea din ultima linie a tabelului.

În cazul numărului de orientări se poate observa că diferențele nu sunt foarte mari. Acuratețea crește odată cu numărul de orientări, dar cu cât acesta este mai mare, gradul de certitudine scade. Pentru că diferențele nu sunt atât de mari ca în cazul anterior, voi rămâne cu valoarea numărului de orientări ales inițial, adică 9.

Numărul de orientări	Acuratețea	Gradul de certitudine
6	91%	75%
7	92%	70%
9	92%	60%

Concluzii

În concluzie, consider că am dezvoltat o aplicație utilă care folosește un algoritm de recunoaștere facială complex și eficient, ce poate fi încadrat ușor în cadrul multor altor aplicații. Totodată, aplicația pune la dispoziție o interfață ușor de utilizat, care oferă utilizatorului posibilitatea de a manipula și modifica colecția de imagini.

Recunoașterea facială este un domeniu foarte popular, iar interesul pentru acesta este în continuă creștere. Acest lucru se datorează beneficiilor pe care le aduce în cadrul dezvoltării de aplicații și sisteme. Poate fi folosită pentru identificarea de persoane, în aplicații de securitate, pentru găsirea unor persoane dispărute sau identificarea unor infractori, sau în cadrul rețelelor de socializare, pentru divertisment. De asemenea, poate fi folosită și pentru autentificarea la anumite dispozitive sau aplicații, fiind una dintre cele mai sigure metode.

Ca și îmbunătățiri și direcții viitoare, aplicația ar putea fi extinsă și pentru utilizarea pe dispozitivele mobile. Astfel, s-ar putea adăuga opțiunea de a încărca și verifica poze făcute direct cu camera telefonului. Acest lucru este posibil, deoarece OpenCV pune la dispoziție funcții care permit lucrul și comunicarea cu o cameră digitală. De asemenea, algoritmul s-ar putea extinde și pentru identificarea persoanelor din surse video.

Nu în ultimul rând, aș vrea să testez algoritmul pe o bază de date mai complexă și anume LFW(*Labeled Faces in the Wild*), pentru a vedea performanța algoritmului. Această colecție conține foarte multe fotografii în ipostaze diferite, ce pun în dificultate identificarea propriu-zisă.

Bibliografie

1. Ngoc-Son Vu, HannahM.Dee, AliceCaplier. **Face recognition using the POEM descriptor.** 2011.

2. https://en.wikipedia.org/wiki/FERET_database.

3. http://www.face-rec.org/algorithms/.

4. https://en.wikipedia.org/wiki/Linear_discriminant_analysis.

5. https://facedetection.com/algorithms/.

6. **Rapid Object Detection using a Boosted Cascade of Simple Features.** P. Viola, M. Jones. 2001.

7. https://en.wikipedia.org/wiki/Sobel_operator.

8. JOHN C. RUSS, F. BRENT NEAL. **The Image Processing Handbook.** 2016.

9. Tomasi, Carlo. **Histograms of Oriented Gradients.**

10. Gary Bradski, Adrian Kaehler. **Learning OpenCV.** 2008.