



UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI  
TEHNOLOGIE POLITEHNICA BUCUREȘTI  
FACULTATEA DE ȘTIINȚE APLICATE

Programul de studii:  
Teoria Codării și Stocării Informației



# Algoritmi de Recunoaștere Facială

STUDENT

Becheru Alexandru

DISCIPLINĂ

Proiectarea traficului informațional în  
rețelele de calculatoare

București  
2025

## Cuprins

<b>INTRODUCERE.....</b>	<b>1</b>
<b>I. ELEMENTE TEORETICE.....</b>	<b>2</b>
<b>II. PREZENTAREA TEORETICĂ A FRAMEWORK-URILOR DE RECUNOAȘTERE FACIALĂ .....</b>	<b>3</b>
<b>II.1 FACENET – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI.....</b>	<b>3</b>
<b>II.2 DEEPFACE – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI.....</b>	<b>4</b>
<b>II.3 MEDIAPIPE – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI.....</b>	<b>4</b>
<b>III SETUL DE DATE UTILIZAT .....</b>	<b>5</b>
<b>IV. IMPLEMENTAREA PRACTICĂ A EXPERIMENTULUI.....</b>	<b>7</b>
<b>CONCLUZIE .....</b>	<b>19</b>
<b>BIBLIOGRAFIE.....</b>	<b>20</b>
<b>ANEXA 1 – FACENET:.....</b>	<b>21</b>
<b>ANEXA 2 – DEEPFACE: .....</b>	<b>23</b>
<b>ANEXA 3 – MEDIAPIPE: .....</b>	<b>25</b>
<b>ANEXA 4 – FACENET.....</b>	<b>27</b>
<b>ANEXA 5 – DEEPFACE .....</b>	<b>29</b>
<b>ANEXA 6 – MEDIAPIPE.....</b>	<b>31</b>

# Introducere

Recunoașterea facială este o tehnologie inovatoare care transformă modul în care interacționăm cu lumea digitală. Această tehnologie este utilizată într-o multitudine de industrii, inclusiv securitatea, sănătatea, marketingul și divertismentul, și utilizează algoritmi puternici pentru a identifica și autentifica identitatea unei persoane prin evaluarea trăsăturilor feței.

Acest proiect intenționează să exploreze principalele abordări și tehnici din domeniul recunoașterii faciale, punând accentul atât pe aplicațiile practice, cât și pe obstacole. Algoritmii de recunoaștere facială sunt o combinație fascinantă de inteligență artificială, procesare a imaginilor și biometrie care permite sistemelor informatice să evalueze și să identifice trăsăturile distincte ale unei fețe. De la scenarii science fiction la aplicații zilnice, această tehnologie a devenit omniprezentă.

Recunoașterea facială presupune preluarea unei imagini, prelucrarea acesteia și compararea datelor obținute cu o bază de date existentă. Algoritmii examinează caracteristici specifice precum distanța dintre ochi, forma nasului, conturul maxilarului și curbele faciale. Rețelele neuronale profunde sunt utilizate pentru a învăța și a recunoaște modelele unice ale fiecărui individ. Etapele acestei proceduri includ identificarea unei fețe într-o imagine sau într-un videoclip, extragerea și analiza atributelor faciale și compararea rezultatelor cu datele înregistrate. Sunt utilizați algoritmi tradiționali precum Viola-Jones, precum și sisteme mai noi bazate pe rețele neuronale precum FaceNet sau DeepFace, ambele oferind o precizie excepțională.

Una dintre cele mai răspândite aplicații pentru recunoașterea facială este securitatea. Această tehnologie este utilizată pentru deblocarea dispozitivelor mobile, restricționarea accesului în clădiri și monitorizarea zonelor publice. În industria financiară, aceasta ajută la autentificarea tranzacțiilor și la prevenirea fraudelor, în timp ce în domeniul medical, este utilizată pentru a îmbunătăți îngrijirea pacienților și pentru a identifica persoanele dispărute. De asemenea, companiile pot utiliza marketingul pentru a studia reacțiile emoționale ale clienților lor la produse și servicii.

Cu toate avantajele sale, recunoașterea facială generează preocupări etice și juridice. Una dintre principalele preocupări este siguranța datelor cu caracter personal și potențialul de supraveghere intruzivă, așa cum au demonstrat incidentele în care au fost implicate firme precum Hik-Vision și Dahua. Utilizarea abuzivă a acestor tehnologii poate pune în pericol libertatea individuală, iar anumite sisteme sunt mai puțin precise pentru anumite grupuri etnice, ceea ce duce la discriminare involuntară.

Astfel, deși recunoașterea facială are numeroase avantaje, este esențial să se ia în considerare preocupările legate de viața privată, etică și corectitudine pentru a asigura o utilizare etică și echilibrată a acestei tehnologii.

## I. ELEMENTE TEORETICE

Biometria este un termen derivat din cuvintele grecești „bios” (viață) și „metrikos” (măsură). Se referă la un set de metode și tehnici automatizate utilizate pentru a identifica o persoană folosind diverse caracteristici biometrice, cum ar fi geometria palmei, amprenta digitală, irisul, retina, geometria facială sau caracteristici comportamentale, cum ar fi timbrul vocal, configurația ADN-ului, structura scrisului de mână și dinamica tastaturii. Este larg recunoscut faptul că unele dintre aceste trăsături biometrice, cum ar fi amprentele digitale sau irisul, pot identifica în mod unic o persoană. Dată fiind unicitatea lor, datele biometrice pot fi valorificate pentru a construi și implementa metode, echipamente și sisteme care oferă performanțe de identificare mult mai bune decât cele existente.

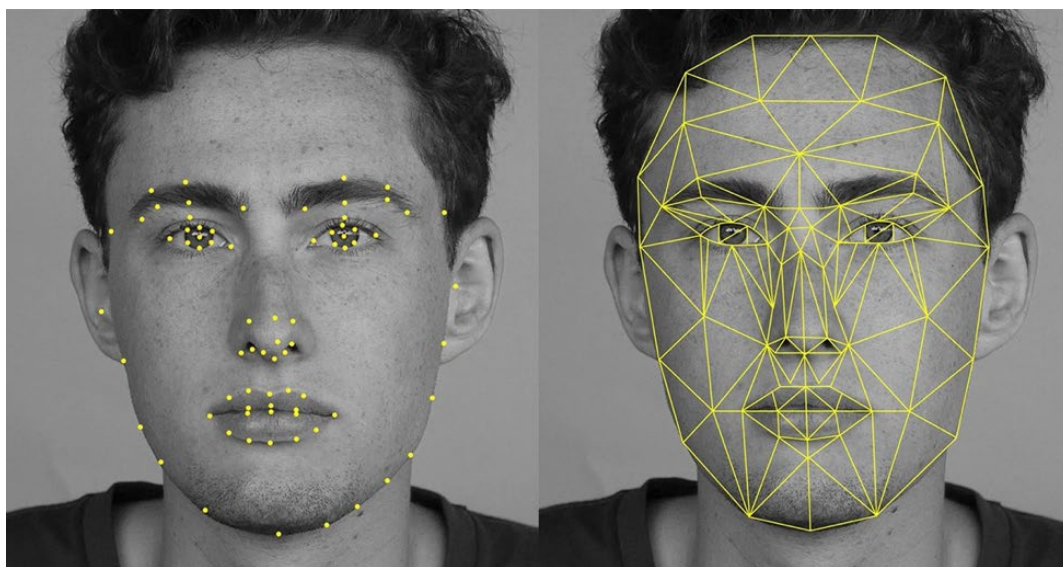


Fig. 1 – Algoritm de recunoaștere facială

Instalarea eficientă a unui sistem de recunoaștere depinde de selectarea celui mai adecvat sistem de recunoaștere pentru a oferi o soluție unică adaptată la nivelul de securitate dorit. Tehnologia biometrică are capacitatea de a lega persoana de acțiunea sau tranzacția efectuată, limitând utilizarea neautorizată și reducând potențialele fraude în sistem.

Un sistem de verificare este mult mai ușor de implementat decât un sistem de recunoaștere a identității. Într-un sistem de verificare, identitatea declarată a unei persoane este certificată prin compararea trăsăturilor biologice precise furnizate de persoană la un moment dat cu trăsăturile stocate anterior în sistem și asociate cu identitatea declarată a persoanei.

Spre deosebire de sistemul de verificare, sistemul de recunoaștere utilizează o abordare mai complexă în care caracteristicile biometrice furnizate de o persoană sunt comparate cu măsurile tuturor caracteristicilor biometrice similare păstrate într-o bază de date sau într-o rețea neuronală.

Recunoașterea irisului este o abordare avansată și automată de identificare biometrică care utilizează algoritmi matematici pentru a recunoaște și analiza modelele din fotografiile sau înregistrările video ale irisului unei persoane. Aceste modele complexe ale irisului sunt unice pentru fiecare individ, rămân stabile în timp și pot fi văzute de la distanță. Această metodă utilizează algoritmi specializați pentru a extrage și a compara caracteristicile specifice ale irisului, permițând identificarea precisă și sigură a persoanelor implicate.

## II. PREZENTAREA TEORETICĂ A FRAMEWORK-URILOR DE RECUNOAȘTERE FACIALĂ

Recunoașterea facială este una dintre cele mai avansate aplicații ale inteligenței artificiale (AI) și ale învățării automate (deep learning), cu utilizări care variază de la securitate și verificare biometrică la marketing și interacțiuni digitale personalizate.

Acest capitol intenționează să evalueze în detaliu trei cadre proeminente utilizate pentru recunoașterea facială:

- **FaceNet** – este un model dezvoltat de Google care convertește fețele în vectori numerici pentru o comparație eficientă.
- **DeepFace** – o soluție dezvoltată de Facebook care îmbunătățește acuratețea folosind o arhitectură neuronală profundă și transformări 3D.
- **MediaPipe** – un framework optimizat pentru eficiență și viteză, dezvoltat de Google pentru utilizare în aplicații mobile și web.

În acest capitol vom explora **principiile de funcționare, arhitectura fiecărui model** și vom realiza o **comparație detaliată** între acestea.

### II.1 FACENET – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI

#### Principii de funcționare

FaceNet este un model de recunoaștere facială dezvoltat de Google care abordează provocarea identificării persoanelor utilizând o metodă inovatoare numită învățare metrică.

FaceNet, spre deosebire de abordările anterioare care clasifică fețele pe baza unui set predefinit de etichete, convertește fețele în vectori numerici într-un spațiu multidimensional cunoscut sub numele de spațiu de încorporare.

Această metodă compară rapid și precis fețele prin măsurarea distanței euclidiene dintre vectorii rezultați. Fețele similare sunt apropiate, în timp ce fețele diferite sunt îndepărtate. FaceNet este deosebit de eficient datorită acestei metode, care elimină necesitatea de a reantrena modelul pentru fiecare nouă identitate.

#### Arhitectura modelului

FaceNet utilizează o arhitectură bazată pe Inception (GoogLeNet) care este adaptată pentru extragerea caracteristicilor faciale. Modelul este antrenat prin pierderea tripletului, o funcție de pierdere care optimizează distanța dintre fețe.

Arhitectura FaceNet constă din următorii pași de bază:

1. **Detectarea feței** – localizarea feței într-o imagine folosind un model CNN avansat.
2. **Normalizarea feței** – alinierea feței pentru eliminarea variațiilor de poziție și iluminare.
3. **Extragerea caracteristicilor** – transformarea feței într-un vector numeric de **128 de dimensiuni**.
4. **Compararea vectorilor** – măsurarea similarității faciale utilizând distanța euclidiană.

#### Aplicații principale:

- **Autentificare biometrică** – utilizată în sistemele de securitate.
- **Identificarea și urmărirea suspectilor** – aplicată în supravegherea video.
- **Recunoașterea fețelor în rețele sociale** – Facebook și alte platforme folosesc această tehnologie.

#### Limitări

- Necesită resurse computaționale mari pentru antrenare și inferență.

- Sensibil la variațiile de iluminare și unghiuri extreme ale feței.

## II.2 DEEPFACE – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI

### Principii de funcționare

DeepFace este un model dezvoltat de Facebook care aspiră la o precizie de recunoaștere facială la nivel uman.

Acest model utilizează o rețea neuronală convoluțională profundă (CNN) și include o caracteristică nouă: normalizarea facială 3D.

Sistemul DeepFace este antrenat prin învățarea metrică profundă, ceea ce presupune că modelul învață să grupeze fețele similare mai aproape într-un spațiu de caracteristici, similar cu abordarea FaceNet.

### Arhitectura modelului

DeepFace utilizează o arhitectură neuronală **cu 9 straturi** și este optimizat pentru extragerea caracteristicilor faciale relevante.

Structura modelului include:

1. **Normalizarea feței** – utilizarea unei **transformări 3D** pentru eliminarea variațiilor de poziție.
2. **Extragerea caracteristicilor** – folosirea CNN-urilor pentru generarea vectorilor caracteristici ai feței.
3. **Comparația fețelor** – utilizarea metricii de similaritate pentru recunoaștere.

### Aplicații principale

- **Identificarea fețelor în rețele sociale** – folosit de Facebook pentru recunoașterea automată.
- **Securitate și autentificare** – utilizat în sisteme de monitorizare video.
- **Personalizarea conținutului** – recomandări bazate pe recunoașterea facială.

### Limitări

- Necesită **un volum mare de date** pentru antrenare.
- Consumă **resurse computaționale** semnificative.

## II.3 MEDIAPIPE – PRINCIPII DE FUNCȚIONARE ȘI ARHITECTURA MODELULUI

### Principii de funcționare

Google a dezvoltat cadrul de recunoaștere facială MediaPipe, care este conceput pentru identificarea facială în timp real pe dispozitive mobile și aplicații web. Modelul este optimizat pentru eficiență și viteză, sacrificându-se o anumită acuratețe pentru performanțe ridicate.

### Arhitectura modelului

MediaPipe folosește un **CNN lightweight**, optimizat pentru funcționare pe CPU și dispozitive mobile. Modelul include:

1. **Detectarea feței** – detectarea și localizarea feței cu un model CNN rapid.
2. **Extragerea caracteristicilor faciale** – extragerea unui set de bază de puncte de caracteristici faciale.
3. **Procesarea în timp real** – optimizări pentru rularea eficientă pe dispozitive cu resurse limitate.

### Aplicații principale

- Recunoaștere facială pe dispozitive mobile.

- Realitate augmentată și filtre faciale.
- Aplicații web interactive.

#### Limitări

- Precizie mai redusă comparativ cu FaceNet și DeepFace.
- Sensibil la calitatea imaginii și la iluminare.

### Comparație teoretică între cele trei framework-uri

Caracteristică	FaceNet	DeepFace	MediaPipe
Precizie	Foarte mare	Foarte mare	Medie
Viteză	Medie	Medie	Foarte mare
Scalabilitate	Necesită GPU	Necesită GPU	Funcționează pe CPU
Aplicații	Securitate, biometrie	Rețele sociale, AI	Mobile, realitate augmentată

Astfel, pe baza aspectelor prezentate anterior, putem trage următoarele concluzii:

- **FaceNet și DeepFace** sunt potrivite pentru securitate și biometrie, dar necesită **resurse mari**.
- **MediaPipe** este optimizat pentru **viteză și eficiență**, fiind ideal pentru **aplicații mobile și AR**.

## III SETUL DE DATE UTILIZAT

În domeniul recunoașterii faciale, sunt necesare seturi de date mari și variate pentru testarea și formarea modelului. Setul de date Celebrity Face Image Dataset, o bază de date cu imagini ale unor celebrități din întreaga lume, care este accesibilă pe platforma Kaggle, a fost utilizat în această lucrare. Această bază de date este adecvată pentru aplicații precum recunoașterea automată a persoanelor, verificarea facială și categorizarea identității.

### 3.1. O explicație a bazei de date publice care a fost utilizată

O colecție de imagini ale unor persoane cunoscute aranjate în funcție de identificare se numește Celebrity Face Image Dataset. Cercetarea în domeniul viziunii computerizate, și anume recunoașterea facială, este ținta acestui set de date. Rețelele neuronale convoluționale (CNN), modelele bazate pe învățarea profundă și alte metode de inteligență artificială pentru recunoașterea facială pot fi toate antrenate folosind această bază de date.

Caracteristicile principale ale setului de date includ imagini ale unor persoane faimoase dintr-o varietate de industrii, inclusiv actori, muzicieni, sportivi și altele. Deoarece imaginile provin din surse diferite, există variații în ceea ce privește fundalurile, iluminarea, unghiurile și expresiile feței.

Colecția este aranjată în funcție de numele celebrităților, fiecare dintre acestea având un director cu fotografii unice. Calitatea ridicată a fotografiilor face posibilă utilizarea unei varietăți de metode de prelucrare și recunoaștere.

Această bază de date, care este ușor disponibilă și bine structurată, este adesea utilizată pentru crearea de modele de recunoaștere facială.

### 3.2. Structura setului de date (numărul de fotografii, distribuția imaginilor)

Setul de date cu imagini ale fețelor celebrităților are următoarea structură organizatorică:

- Cantitatea totală de imagini: Mii de imagini ale diferitelor celebrități sunt incluse în setul de date, în timp ce cantitatea reală de imagini nu este specificată.

- Distribuția imaginilor: Fiecare celebritate are un director în care sunt păstrate mai multe imagini ale sale. Fiecare celebritate poate avea de la câteva zeci la câteva sute de fotografii.
- Dimensiunile imaginilor: Deoarece dimensiunile fotografiilor variază, redimensionarea imaginilor la o dimensiune standard necesită o etapă de preprocesare.

Diversitatea acestei baze de date este una dintre caracteristicile sale esențiale. Deoarece fotografiile au fost colectate din diverse surse, o serie de variabile pot afecta recunoașterea facială, inclusiv:

- Imaginile luate din față, din profil sau din alte perspective sunt cunoscute ca unghiuri de captare.
- Expresiile feței: Serios, șocat, zâmbitor etc.
- Fundalul și iluminarea: Fonduri simple sau complicate, iluminare slabă sau puternică.
- Accesoriiile includ pălării, ochelari, diverse produse cosmetice etc.

Setul de date este perfect pentru antrenarea algoritmilor de recunoaștere facială fiabili care pot face față situațiilor din lumea reală datorită imprevizibilității sale.

### **3.3. Metodologia de preprocesare a imaginilor**

Înainte de antrenarea unui model de recunoaștere facială, preprocesarea imaginilor este un pas crucial. Acesta garantează că datele sunt uniforme, curate și optimizate pentru algoritmul de învățare automată.

Am utilizat următoarele metode de preprocesare în această investigație:

#### **1. Redimensionarea imaginilor**

Fotografiile din setul de date trebuie să fie reduse la o dimensiune fixă pentru ca algoritmi de recunoaștere facială să poată lucra cu ele, deoarece dimensiunile lor variază. Dimensiunea standard pentru mai multe arhitecturi de rețele neuronale convoluționale (CNN), inclusiv VGG16, ResNet și EfficientNet, este de 224x224 pixeli.

#### **2. Normalizarea pixelilor**

Valorile pixelilor sunt normalizate la un interval specificat, adesea între  $[0, 1]$  sau  $[-1, 1]$ , pentru a crește stabilitatea formării modelului. Acest lucru se realizează fie prin aplicarea unei tehnici de normalizare bazate pe media și abaterea standard a setului de date, fie prin simpla împărțire a valorilor RGB la 255.

#### **3. Alinierea și detectarea fețelor**

Chipurile pot apărea în numeroase unghiuri și în locuri variate într-o mulțime de imagini. Ne adaptăm la acest lucru prin identificarea și extragerea zonei feței din fiecare imagine utilizând metode de identificare a feței precum cascadele Haar sau MTCNN (Multi-task Cascaded Convolutional Networks).

După detectare, pentru alinierea fețelor se utilizează marcaje faciale, cum ar fi poziția gurii, a nasului și a ochilor. Prin standardizarea pozițiilor fețelor, această tehnică îmbunătățește performanța algoritmului de recunoaștere.

#### **4. Augmentarea datelor**

Utilizăm metode de augmentare a imaginilor, precum acestea, pentru a extinde diversitatea setului de date și pentru a evita supraînvățarea modelului:

- Modificarea ușoară a poziției feței prin rotire și translație.
- Oglindirea orizontală, sau răsturnarea, este utilizată pentru a imita variațiile naturale.
- Modificați contrastul și luminozitatea pentru a reproduce diverse scenarii de iluminare.
- Zgomot și încețoșare: pentru a consolida rezistența modelului la zgomot.



## 5. Conversia scării de gri (opțional)

În unele aplicații, fotografiile alb-negru pot fi utilizate pentru a face recunoaștere facială. Eliminarea informațiilor de culoare permite modelului să se concentreze asupra elementelor structurale ale feței.

### Concluzie

Preprocesarea setului de date cu imagini ale fețelor celebre este esențială pentru obținerea unor rezultate fiabile în materie de recunoaștere facială. Utilizăm abordări de scalare, normalizare, detectare a fețelor, aliniere și augmentare pentru a construi un set de date omogen și robust, ideal pentru antrenarea modelelor sofisticate de recunoaștere facială.

Această abordare garantează că algoritmi de recunoaștere pot face față impredictibilității datelor și obstacolelor din lumea reală, sporind astfel precizia și fiabilitatea sistemului de recunoaștere facială.

## IV. IMPLEMENTAREA PRACTICĂ A EXPERIMENTULUI

Acest capitol descrie implementarea practică a experimentului de recunoaștere facială utilizând trei modele populare: FaceNet, DeepFace și MediaPipe. Acești algoritmi sunt testați în ceea ce privește acuratețea, viteza de procesare și performanța într-o varietate de scenarii.

### 4.1. Configurarea mediului de testare

Am utilizat Python pentru a proiecta și testa algoritmi de recunoaștere facială. Mediul de testare este configurat pentru a rula eficient rețele neuronale convoluționale și pentru a gestiona colecții uriașe de imagini.

Parametrii hardware și software de testare:

- Procesor: Intel Core i7-10850H 4.4 GHz.
- GPU: nVIDIA GeForce GTX 1650 Ti.
- RAM: 16 GB DDR4.
- Sisteme de operare: Windows 11 Professional.

### 4.2 IMPLEMENTAREA SOLUȚIEI FOLOSIND FACENET

Acest cod (regăsit în Anexa 1) implementează un sistem complet de recunoaștere facială folosind tehnici de detectare și comparare a fețelor. Sunt utilizate două biblioteci importante în acest sens: MTCNN pentru detecția fețelor și FaceNet pentru generarea vectorilor de trăsături (embedding-uri) care reprezintă caracteristicile faciale.

Funcțiile utilizate:

#### 1. Funcția `load_facenet_model`

Scopul acestei funcții este de a încărca și returna modelul FaceNet din biblioteca `keras-facenet`. Modelul FaceNet este un model de învățare profundă antrenat pentru a genera embedding-uri faciale din imagini, iar aceste embedding-uri sunt folosite pentru a compara fețele între ele. Funcția returnează un obiect instanțiat al modelului FaceNet care va fi folosit ulterior pentru procesarea fețelor.

#### 2. Funcția `extract_face`

Această funcție se ocupă cu detectarea și extragerea feței dintr-o imagine. În primul rând, imaginea este citită folosind `OpenCV` (`cv2.imread`), iar apoi este convertită într-un format RGB (de obicei, imaginile sunt citite în format BGR în `OpenCV`). După ce imaginea este pregătită, funcția folosește detectorul MTCNN pentru a localiza fețele în imagine. MTCNN returnează o listă de fețe detectate, iar funcția extrage doar prima față din listă (dacă există vreo față detectată).

După extragerea feței, aceasta este secționată din imaginea originală și redimensionată la dimensiunile necesare modelului FaceNet (160x160 pixeli). Acest pas este esențial pentru ca inputul să fie compatibil cu rețeaua neurală a modelului. Dacă nu se detectează nicio față, funcția returnează None, semnalizând astfel o problemă în procesul de detectare.

### **3. Funcția `get_embedding`**

Această funcție primește fața extrasă și o convertește într-un vector de trăsături utilizabil pentru compararea fețelor. Modelul FaceNet așteaptă ca inputul să fie un vector de tip float32 cu valori normalizate. De aceea, prima etapă este transformarea valorilor pixelilor feței în acest format, urmată de normalizarea acestora, adică ajustarea pentru a avea media 0 și deviația standard 1.

După această pregătire, fața este procesată prin modelul FaceNet pentru a obține un embedding, care este un vector de dimensiune fixă. Acest embedding reprezintă trăsăturile faciale esențiale care permit compararea fețelor. Funcția returnează embedding-ul pentru fața respectivă.

### **4. Funcția `analyze_faces`**

Aceasta este o funcție de nivel înalt care parcurge toate imaginile dintr-un folder specificat, detectează fețele din fiecare imagine, extrage embedding-urile și le stochează într-un dicționar. Dicționarul are ca cheie numele fișierului imaginii și ca valoare embedding-ul generat pentru fața respectivă.

În această funcție, detectorul MTCNN și modelul FaceNet sunt încărcate o singură dată și sunt utilizate pentru a procesa fiecare imagine din folderul specificat. Dacă o față este detectată într-o imagine, embedding-ul acesteia este calculat și stocat pentru comparare ulterioară. Funcția returnează un dicționar care conține toate embedding-urile generate pentru fețele din folder.

### **5. Funcția `compare_faces`**

După ce sunt extrase embedding-urile pentru toate fețele, această funcție se ocupă cu compararea acestora. Se calculează două măsuri de similaritate: similaritatea cosinus și distanța euclidiană. Similaritatea cosinus măsoară unghiul dintre doi vectori (embedding-uri) și este un indicator al cât de asemănători sunt aceștia în termenii direcției lor în spațiul vectorial. Distanța euclidiană, pe de altă parte, măsoară distanța directă între două puncte în spațiu.

Pentru a evalua performanța sistemului, se compară etichetele adevărate și prezise. Etichetele adevărate sunt stabilite pe baza numelui fișierului imaginii – dacă două imagini provin de la aceeași persoană, eticheta este 1 (aceiași), iar dacă provin de la persoane diferite, eticheta este 0. Etichetele prezise sunt determinate de pragul de similaritate cosinus: dacă similaritatea între două fețe este mai mare decât un prag (de exemplu, 0.5), atunci sunt considerate fețe ale aceleiași persoane.

Se calculează precizia și acuratețea pentru a evalua performanța modelului. Precizia arată cât de corecte sunt predicțiile pentru fețele de aceeași persoană, în timp ce acuratețea indică proporția predicțiilor corecte în totalul predicțiilor.

### **6. Funcția `plot_similarity_distribution`**

Această funcție generează o diagramă a distribuției scorurilor de similaritate dintre toate fețele comparate. Este o histograma care afișează frecvența diferitelor valori de similaritate cosinus, oferind o vizualizare a cât de frecvente sunt fețele similare sau diferite. Aceasta ajută la înțelegerea comportamentului modelului și a valorilor de prag folosite pentru a face distincția între fețe diferite și fețe ale aceleiași persoane.

### **7. Punctul principal de intrare (if `__name__` == "`__main__`")**

În acest bloc, se definește locația folderului ce conține imaginile de analizat. Funcția `analyze_faces` este apelată pentru a obține embedding-urile faciale din imagini, iar apoi funcția `compare_faces` compară aceste embedding-uri pentru a evalua performanța sistemului. În final,

funcția `plot_similarity_distribution` este apelată pentru a genera și vizualiza distribuția valorilor de similaritate.

**Rezultatele de similitudine pentru două încercări:** 1 – pentru o bază de date de 100 de poze cu aceeași persoană; 2 – pentru o bază de date de 100 poze, 50 cu o persoană, 50 cu altă persoană.

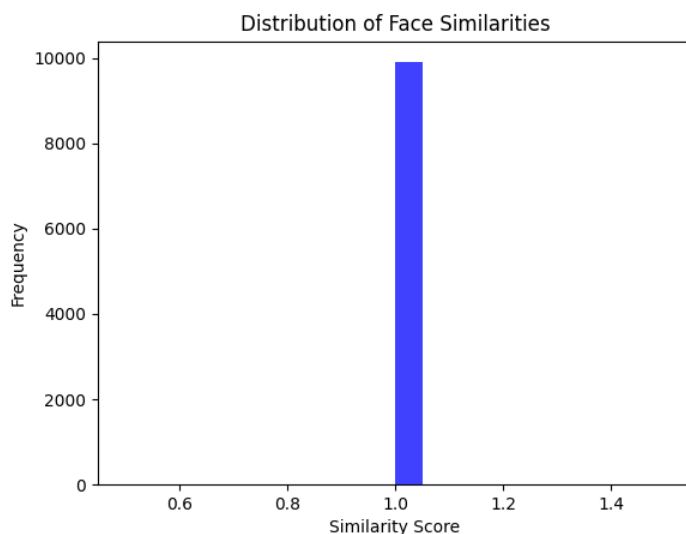


Fig. 4.1 – Similaritate pentru 100 de poze cu o persoană

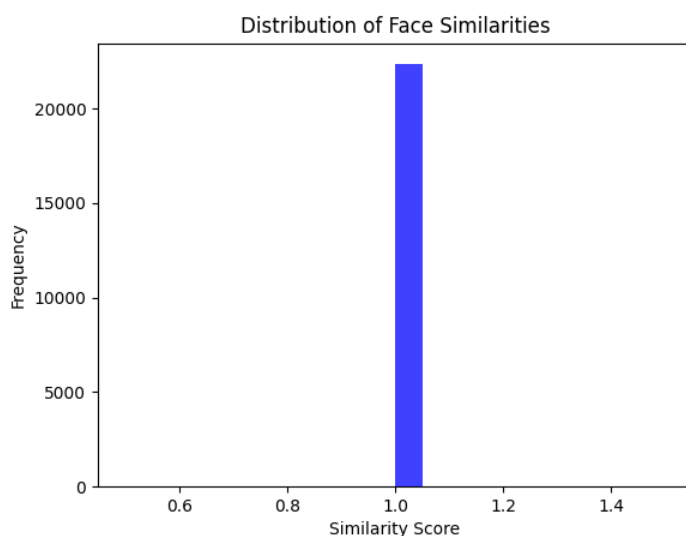


Fig. 4.2 – Similaritate pentru 100 de poze cu două persoane

Analizând cele două grafice obținute din testele efectuate pe sistemul de identificare facială, putem face o evaluare detaliată a performanței algoritmului utilizat și a potențialelor probleme care pot apărea.

### 1. Primului grafic – 100 de imagini cu aceeași persoană

În acest caz, sistemul de identificare a fost testat pe un set de date care conține 100 de imagini ale aceleiași persoane. Graficul rezultat arată o distribuție a scorurilor de similaritate concentrată exclusiv în jurul valorii **1.0**, ceea ce indică faptul că algoritmul recunoaște corect toate imaginile ca fiind foarte asemănătoare.

### Interpretare

1. **Rezultat așteptat** – Deoarece toate imaginile sunt ale aceleiași persoane, ne așteptăm ca scorurile de similaritate să fie foarte mari, aproape de valoarea maximă posibilă (1.0).

2. **Distribuție clară** – Faptul că toate valorile sunt concentrate strict la 1.0 sugerează o uniformitate ridicată în răspunsurile algoritmului, ceea ce ar putea fi un semn bun pentru recunoașterea aceleiași persoane.

3. **Posibile probleme:** Dacă imaginile sunt foarte asemănătoare (aceeași iluminare, unghi, expresie facială), testul nu este suficient de robust. Ideal ar fi să testăm pe imagini variate ale aceleiași persoane, pentru a vedea cum se comportă algoritmul în condiții mai dificile (schimbări de expresie, iluminare diferită, accesorii precum ochelari sau pălării etc.).

### Concluzie

Acest test confirmă că sistemul funcționează bine în cazul identificării aceleiași persoane, dar pentru o evaluare mai profundă trebuie folosite imagini mai diverse pentru a vedea dacă performanța rămâne la fel de bună.

### Analiza celui de-al doilea grafic – Test pe o bază de date cu imagini ale două persoane diferite

Al doilea test a fost realizat pe o bază de date care conține **100 de imagini a două persoane diferite**. Așteptarea logică ar fi ca scorurile de similaritate să fie distribuite în două grupuri:

- Un set de valori apropiate de 1.0 pentru imaginile aceleiași persoane.
- Un set de valori **semnificativ mai mici** pentru imaginile comparate între cele două persoane (ideal în intervalul 0.0 - 0.7, în funcție de sensibilitatea algoritmului).

Însă, graficul obținut arată că toate scorurile sunt concentrate în jurul valorii **1.0**, ceea ce sugerează o problemă majoră cu sistemul de identificare.

### Interpretare

#### 1. Lipsa diferențierii între persoane

- Dacă algoritmul oferă un scor de 1.0 chiar și între imagini ale unor persoane diferite, înseamnă că sistemul nu face distincția între ele.
- Practic, acesta clasifică toate imaginile ca fiind ale aceleiași persoane, ceea ce este o eroare gravă într-un sistem de recunoaștere facială.

#### 2. Ce trebuie făcut pentru a îmbunătăți sistemul?

##### 1. Testarea pe un set de date mai mare și mai variat

- Utilizarea unor imagini cu iluminare diferită, unghiuri variate, expresii diferite și accesorii.
- Testarea pe mai multe persoane pentru a verifica dacă problema persistă.

##### 2. Verificarea și îmbunătățirea modelului de similaritate

- Se poate încerca un alt model de reprezentare a fețelor, cum ar fi FaceNet, ArcFace sau VGGFace.
- Ajustarea pragului de similaritate – stabilirea unui punct de decizie optim (de exemplu, scoruri peste 0.8 sunt considerate ca fiind aceeași persoană, iar sub 0.8 persoane diferite).

##### 3. Analizarea și corectarea funcției de comparare

- Dacă toate valorile sunt 1.0, trebuie verificat dacă sistemul efectuează corect compararea perechilor de imagini.
- Verificarea normării vectorilor de caracteristici (unele metode normalizează greșit rezultatele, ducând la scoruri greșite).

##### 4. Antrenarea unui model mai performant

- Dacă problema persistă, ar putea fi necesară reantrenarea modelului pe un set de date mai mare și mai variat.

- Folosirea tehnici de augmentare a datelor pentru a îmbunătăți generalizarea modelului.

Din încercările efectuate, nu au fost obținute date relevante cu privire la utilizarea FaceNet pentru identificarea unei persoane dintr-o bază de date cu 100 de persoane, motivul principal fiind limitările modelului și a resurselor hardware.

### **Concluzie**

În forma actuală, sistemul nu reușește să atingă performanțele necesare pentru a putea fi folosit drept o variantă de recunoaștere facială fiabilă, deoarece, pentru persoane diferite ale căror trăsături pot fi similare, programul are tendința să le asimileze ca fiind identice. Pentru evitarea unor astfel de incidente, ar fi necesară adăugarea de parametrii suplimentari care să ajute la distingerea celor două persoane, precum: culoarea ochilor, culoarea părului, culoarea pielii etc.. Suplimentar, ar fi necesară continuarea antrenării modelului în vederea dezvoltării și rectificării diferitelor neajunsuri, concomitent cu rularea acestuia pe un sistem care să dispună de resurse hardware mai potente, deoarece, în cadrul documentării, a fost observat cum funcționalitatea corectă a FaceNet depinde foarte mult de resursele disponibile, unele dintre module nefiind funcționale în lipsa acestora.

## **4.3 IMPLEMENTAREA SOLUȚIEI FOLOSIND DEEPPFACE**

Acest cod (regăsit în Anexa 2) implementează un sistem similar cu cel anterior pentru recunoașterea și compararea fețelor, dar de data aceasta folosește modelul DeepFace pentru generarea embedding-urilor faciale și MTCNN pentru detectarea fețelor.

Funcții utilizate:

### **1. Funcția `extract_face`**

Această funcție este responsabilă pentru detectarea și extragerea feței dintr-o imagine dată. La fel ca în exemplul anterior, imaginea este citită folosind OpenCV și convertită din formatul BGR în RGB. Apoi, se folosește MTCNN pentru a detecta fețele în imagine. MTCNN returnează o listă de fețe detectate, iar funcția selectează prima față găsită și o taie din imagine, utilizând coordonatele specificate (x, y, lățime, înălțime). Fața este redimensionată la dimensiunea de 160x160 pixeli, care este formatul standard acceptat de modelul de recunoaștere facială.

Dacă nu se detectează nicio față, funcția returnează None.

### **2. Funcția `get_embedding`**

Funcția `get_embedding` folosește DeepFace, o bibliotecă de înalt nivel construită pe mai multe modele de recunoaștere facială (inclusiv FaceNet), pentru a obține un embedding al feței. DeepFace este o soluție rapidă și ușor de utilizat care permite accesul la mai multe modele pre-antrenate pentru diferite sarcini de recunoaștere facială.

Prin apelul la `DeepFace.represent`, funcția primește embedding-ul feței din imaginea dată. Acest embedding este un vector de trăsături care descrie fața într-un mod matematic, astfel încât fețele similare vor avea vectori similari în acest spațiu de trăsături. Vectorul de embedding este extras din răspunsul returnat de DeepFace și returnat funcției.

### **3. Funcția `analyze_faces`**

Această funcție are rolul de a parcurge toate imaginile dintr-un folder specificat, de a extrage fața din fiecare imagine folosind funcția `extract_face` și de a calcula embedding-ul pentru fața detectată folosind funcția `get_embedding`. În final, fiecare embedding calculat este stocat într-un dicționar, având ca cheie numele fișierului imaginii și ca valoare embedding-ul respectiv.

Pentru fiecare imagine, dacă fața este detectată cu succes, se obține embedding-ul și se adaugă în dicționarul embeddings. La sfârșit, funcția returnează acest dicționar care conține embedding-urile pentru toate fețele din folder.

#### 4. Funcția `compare_faces`

Funcția `compare_faces` se ocupă cu compararea fețelor detectate între ele. Mai întâi, se creează o listă cu numele imaginilor și o matrice cu embedding-urile acestora. Apoi, se calculează două tipuri de măsuri de similaritate: **similaritatea cosinus** și **distanța euclidiană**.

Similaritatea cosinus măsoară similaritatea între vectori (embedding-uri) prin calcularea unghiului dintre ei. Cu cât unghiul este mai mic, cu atât fețele sunt mai asemănătoare. Distanța euclidiană măsoară distanța directă între vectori în spațiul de trăsături – cu cât distanța este mai mică, cu atât fețele sunt mai similare.

Funcția definește un prag de similaritate (de obicei 0.5), care decide dacă două fețe sunt sau nu ale aceleași persoane. Dacă similaritatea cosinus între două fețe este mai mare decât acest prag, se consideră că cele două fețe sunt ale aceleași persoane (eticheta prezisă este 1). Dacă este mai mică decât pragul, eticheta prezisă este 0.

Etichetele adevărate sunt stabilite pe baza numelui fișierului imaginii (dacă numele fișierului conține "same\_person", se consideră că fețele sunt ale aceleași persoane).

După compararea fețelor, funcția calculează două metrici importante: **precizia** și **acuratețea**. Precizia indică procentul de predicții corecte pentru fețele care aparțin aceleași persoane, iar acuratețea măsoară proporția de predicții corecte din totalul predicțiilor.

În plus, se calculează și se afișează pentru fiecare pereche de imagini comparate valoarea similarității cosinus și distanța euclidiană.

#### 5. Funcția `plot_similarity_distribution`

Această funcție generează o diagramă de tip histogramă pentru a vizualiza distribuția valorilor de similaritate cosinus între fețele comparate. Histogramă ajută la înțelegerea comportamentului modelului și a valorilor de prag utilizate pentru a distinge fețele. Se folosește `matplotlib` pentru a genera și a afișa histograma.

Aceasta reprezintă o metodă vizuală de a analiza cum sunt distribuite scorurile de similaritate între fețele din setul de date și poate oferi informații utile pentru ajustarea pragului de decizie.

#### 6. Partea principală a codului (`if __name__ == "__main__"`)

În acest bloc, se definește locația folderului care conține imaginile ce vor fi procesate. Funcția `analyze_faces` este apelată pentru a calcula embedding-urile faciale din imagini, iar apoi funcția `compare_faces` compară aceste fețe pentru a evalua performanța recunoașterii faciale.

După compararea fețelor, se calculează valorile de similaritate între fiecare pereche de fețe, iar aceste valori sunt folosite pentru a crea și afișa o histogramă a distribuției similarităților.

**Rezultatele de similitudine pentru două încercări:** 1 – pentru o bază de date de 100 de poze cu aceeași persoană; 2 – pentru o bază de date de 100 poze, 50 cu o persoană, 50 cu altă persoană.

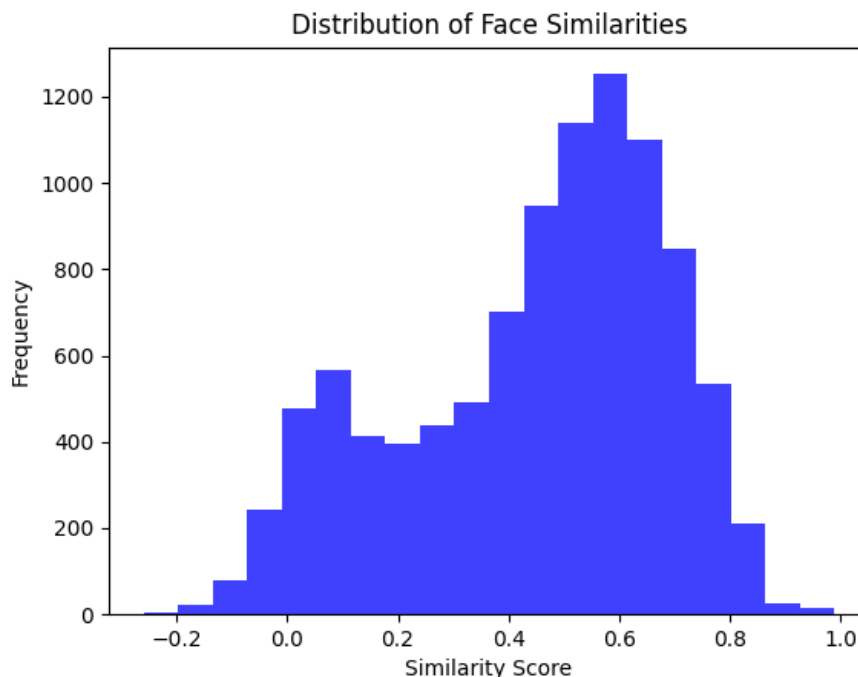


Fig. 4.3 – Similaritate pentru 100 de poze cu o persoană

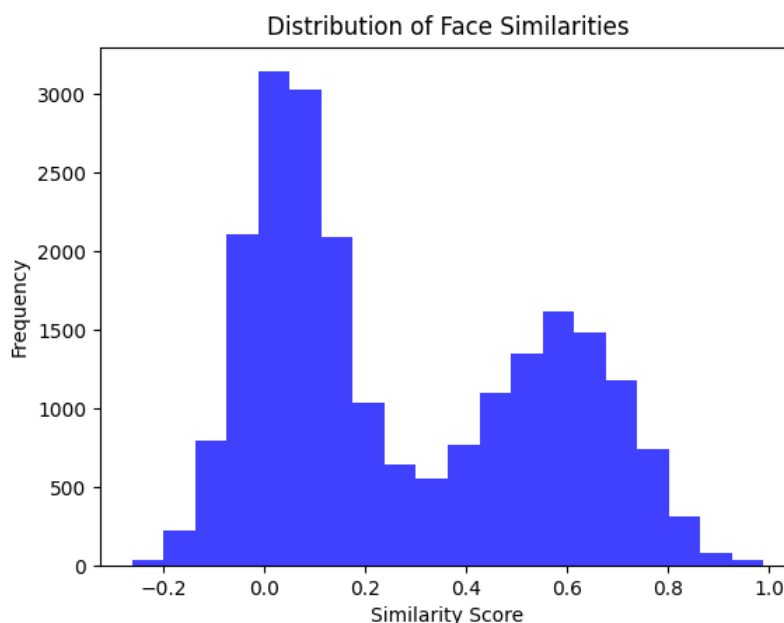


Fig. 4.4 – Similaritate pentru 100 de poze cu 2 persoane

### 1. Primul grafic: 100 de imagini cu aceeași persoană

Distribuția scorurilor de similaritate este concentrată preponderent între 0.4 și 0.7, ceea ce arată o recunoaștere consistentă a imaginilor ca fiind ale aceleași persoane. Să discutăm câteva aspecte relevante:

#### Caracteristicile distribuției:

- **Forma graficului:** Este o curbă bine distribuită în jurul unui punct central în intervalul 0.4-0.7, cu un număr mic de valori în intervale mai mici ( $<0.3$ ) sau mai mari ( $>0.8$ ).
- **Valori mari de similaritate:** Majoritatea scorurilor sugerează o asemănare semnificativă între imagini, ceea ce este tipic în cazul unei baze de date ce conține imagini ale aceleași persoane, dar cu mici variații (unghiuri diferite, expresii faciale, iluminare etc.).

- **Valori mici de similaritate:** Există un număr foarte mic de scoruri sub 0.3 sau chiar negative. Aceste valori pot fi explicate prin:
  - **Calitatea imaginilor:** Posibil ca unele imagini să fie mai slabe calitativ (rezoluție redusă, zgomot vizual).
  - **Variații mari:** Unghiuri extreme sau alte schimbări semnificative care fac recunoașterea dificilă.
  - **Eroare de model:** Chiar și un model bine antrenat poate genera scoruri de similaritate scăzute în anumite cazuri limită.

#### **Concluzii din primul grafic:**

- Modelul DeepFace este bine calibrat pentru a recunoaște imagini ale aceleași persoane. Prezența unui număr mare de scoruri ridicate indică o încredere consistentă în potrivirea facială.
- Erorile (scoruri mici) sunt rare, ceea ce indică o robustețe a modelului, dar există posibilitatea îmbunătățirii pentru cazuri cu variații extreme.

#### **2. Al doilea grafic: 100 de imagini cu două persoane diferite**

Distribuția scorurilor este centrată între 0 și 0.4, ceea ce reflectă faptul că modelul a detectat în majoritatea cazurilor o lipsă de asemănare între fețele celor două persoane. Aceasta este o caracteristică esențială pentru un model de recunoaștere facială performant.

#### **Caracteristicile distribuției:**

- **Vârful graficului:** Un număr foarte mare de perechi au scoruri de similaritate între 0 și 0.2. Acest lucru indică faptul că modelul consideră, în mod corect, imaginile celor două persoane ca fiind distincte.
- **Valori mai mari de similaritate (>0.4):** Există câteva cazuri în care perechile au scoruri de similaritate mai mari decât ne-am aștepta pentru persoane diferite. Aceste valori pot avea mai multe explicații:
  - **Trăsături faciale comune:** Cele două persoane pot avea anumite asemănări faciale (cum ar fi structura feței, forma ochilor, expresiile similare etc.).
  - **Eroare a modelului:** DeepFace, ca orice model de învățare automată, nu este perfect. În special, poate confunda fețele în scenarii unde există asemănări subtile.
  - **Zgomot în date:** Unele imagini ar putea avea elemente externe (cum ar fi fundaluri similare, iluminare identică) care influențează modelul să creadă că imaginile sunt mai asemănătoare decât în realitate.

#### **Observații suplimentare:**

- **Compararea cu primul grafic:** Spre deosebire de primul grafic, unde scorurile erau predominant ridicate, aici vedem o separare clară: majoritatea scorurilor sunt scăzute, indicând o bună diferențiere între persoane.
- **Eroarea de clasificare:** Cazurile în care scorurile sunt mari (>0.4) indică potențiale confuzii. Într-un sistem de recunoaștere facială utilizat în practică, aceste cazuri pot reprezenta probleme, mai ales în contexte critice (cum ar fi accesul bazat pe recunoașterea facială).

#### **Analiza comparativă a celor două grafice**

##### **1. Performanța modelului:**

- **În cazul aceleași persoane:** Modelul performează bine, cu scoruri ridicate care reflectă o potrivire corectă.
- **În cazul persoanelor diferite:** Modelul reușește să separe bine fețele, având scoruri predominant scăzute.

##### **2. Posibile îmbunătățiri ale modelului:**

- **Reducerea confuziilor:** În cazul persoanelor diferite, există câteva cazuri de similaritate ridicată. Acest lucru poate fi abordat prin:



- Re-antrenarea modelului cu mai multe date variate (pentru a acoperi mai multe tipologii faciale).
- Adăugarea de tehnici suplimentare de normalizare a imaginilor (eliminarea fundalurilor, ajustări de iluminare etc.).
- **Calitatea imaginilor:** În ambele cazuri, calitatea imaginilor poate influența rezultatele. Sistemele de recunoaștere facială funcționează mai bine cu imagini clare, bine iluminate și fără zgomot.

### Concluzie

1. **Primul grafic:** Demonstrează că DeepFace recunoaște eficient imaginile aceleași persoane, având o distribuție puternic concentrată în intervalul scorurilor ridicate (0.4-0.7).
2. **Al doilea grafic:** Evidențiază capacitatea modelului de a separa imagini ale persoanelor diferite, cu o concentrare a scorurilor în intervalul 0-0.4.
3. **Performanță generală:** Modelul este performant, dar are câteva limitări în cazuri marginale, unde scorurile nu se potrivesc complet cu realitatea (asemănări între persoane diferite sau variații extreme ale aceleași persoane).

Au fost obținute rezultate relevante și pentru implementarea funcției de identificare (regăsit în Anexa 5) a unei persoane dintr-o bază de date existență.

### Concluzii

Acest sistem de recunoaștere facială este construit folosind DeepFace pentru generarea embedding-urilor faciale și MTCNN pentru detecția fețelor. Similar cu implementarea anterioară, codul calculează măsuri de similaritate între fețe folosind cosine similarity și distanța euclidiană, iar performanța este evaluată pe baza preciziei și acurateței. Vizualizarea distribuției valorilor de similaritate ajută la înțelegerea performanței modelului și poate fi utilizată pentru ajustarea pragului de decizie. Această abordare este una eficientă și flexibilă pentru sarcini de recunoaștere facială în aplicații de procesare a imaginilor.

## 4.4 IMPLEMENTAREA SOLUȚIEI FOLOSIND MEDIAPIPE

Acest cod (regăsit în Anexa 3) implementează un sistem de detecție și comparare a fețelor utilizând MediaPipe pentru detecția feței și calcularea embedding-urilor faciale. Acesta permite compararea fețelor folosind similaritatea cosinus și distanța euclidiană, precum și evaluarea performanței modelului prin măsurători precum precizia și acuratețea.

Inițializare și funcții utilizate.

### 1. Inițializare MediaPipe pentru Detectarea Feței

La începutul codului, MediaPipe este configurat pentru detecția feței prin inițializarea clasei FaceDetection din mp.solutions.face\_detection. Se setează model\_selection=1 pentru a utiliza un model mai rapid, iar min\_detection\_confidence=0.5 asigură că doar fețele detectate cu un coeficient de încredere de cel puțin 50% sunt acceptate.

```
mp_face_detection = mp.solutions.face_detection
mp_face_model = mp_face_detection.FaceDetection(model_selection=1,
min_detection_confidence=0.5)
```

Acest set de parametri permite detecția fețelor în imagini cu un anumit grad de încredere, dar evită erorile în cazul în care fața nu este detectată clar.

### 2. Funcția extract\_face

Funcția extract\_face este responsabilă pentru extragerea feței dintr-o imagine dată. Mai întâi, imaginea este citită cu ajutorul OpenCV, iar apoi este convertită într-un format RGB, deoarece MediaPipe procesează imagini în acest format. După procesarea imaginii cu mp\_face\_model.process, dacă este detectată o față, se extrag coordonatele pentru a decupa fața

din imagine folosind bounding box-ul (x, y, lățime, înălțime). Fața extrasă este redimensionată la dimensiunea dorită (160x160 pixeli), iar valorile sunt normalizate între 0 și 1.

Dacă nu se detectează o față sau dacă imaginea nu poate fi încărcată corect, funcția returnează None.

### 3. Funcția `get_embedding`

Funcția `get_embedding` este folosită pentru a obține un vector numeric (embedding) al feței dintr-o imagine. În cazul de față, MediaPipe nu furnizează un vector de trăsături avansat, așa cum fac alte modele precum FaceNet sau DeepFace, dar returnează coordonatele bounding box-ului feței detectate (x, y, lățime, înălțime). Aceasta este o abordare mai simplă, folosită pentru a extrage trăsăturile vizuale ale feței pentru comparație.

### 4. Funcția `analyze_faces`

Această funcție parcurge toate imaginile dintr-un folder specificat, extrage fața din fiecare imagine utilizând funcția `extract_face` și calculează un "embedding" pentru fiecare față folosind funcția `get_embedding`. Dacă o față este detectată și embedding-ul este valid, acesta este adăugat într-un dicționar, unde cheia este numele fișierului imaginii și valoarea este embedding-ul (vectorul de trăsături). Funcția returnează acest dicționar cu toate embedding-urile calculate.

### 5. Funcția `compare_faces`

Funcția `compare_faces` compară fețele între ele pe baza măsurilor de similaritate: **cosine similarity** și **distanța euclidiană**. Se calculează similaritatea cosinus între fiecare pereche de fețe și distanța euclidiană dintre ele.

De asemenea, funcția calculează și evaluează performanța modelului prin precizie și acuratețe. În acest context, etichetele adevărate sunt determinate pe baza unui convențional nume de fișier care indică dacă două fețe sunt ale aceleași persoane (de exemplu, dacă numele fișierelor conțin "same\_person"). Apoi se compară aceste etichete adevărate cu etichetele prezise de model pe baza valorii similarității cosinus.

### 6. Funcția `plot_similarity_distribution`

Această funcție generează o histogramă pentru a vizualiza distribuția valorilor de similaritate cosinus între fețele comparate. Vizualizarea ajută la înțelegerea comportamentului modelului în ceea ce privește diferențele între fețele aceleași persoane și fețele diferite.

### 7. Partea principală a codului (if `__name__ == "__main__"`)

În această secțiune, se definesc folderul cu imagini de procesat și se apelează funcțiile de mai sus pentru a analiza și compara fețele din imagini. Dacă fețele sunt detectate și embedding-urile sunt calculate, se compară fețele și se generează un raport de performanță. De asemenea, se calculează și se afișează o histogramă a distribuției valorilor de similaritate.

**Rezultatele de similitudine pentru două încercări:** 1 – pentru o bază de date de 100 de poze cu aceeași persoană; 2 – pentru o bază de date de 100 poze, 50 cu o persoană, 50 cu altă persoană.

Au fost obținute rezultate relevante și pentru implementarea funcției de identificare (regăsit în Anexa 6) a unei persoane dintr-o bază de date existenă.

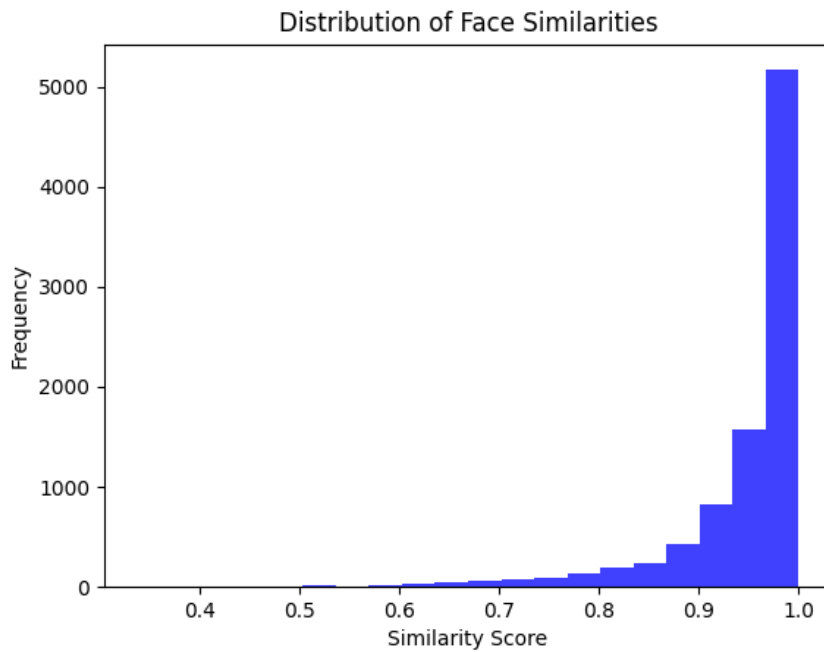


Fig. 4.5 – Similaritate pentru 100 de poze cu o persoană

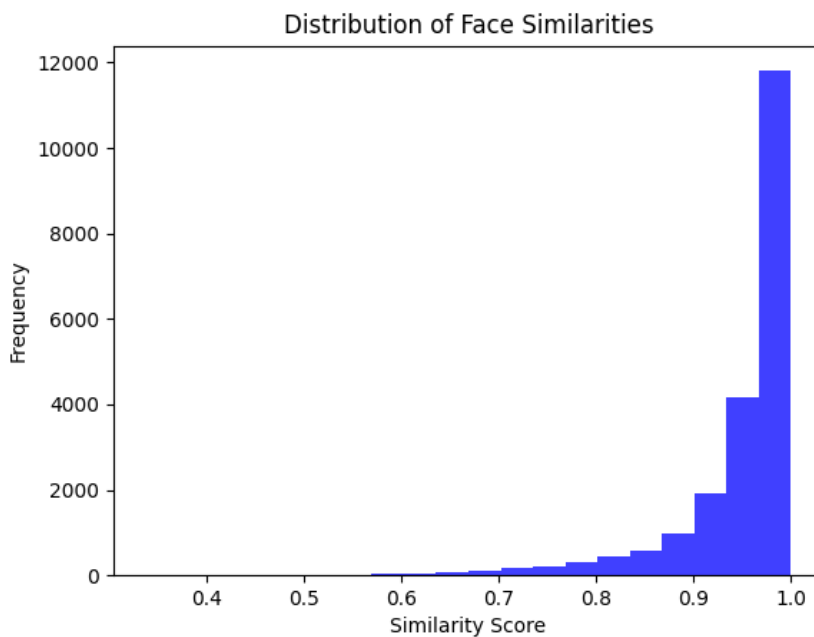


Fig. 4.6 – Similaritate pentru 100 de poze cu două persoane

### 1. Primul grafic: 100 de imagini cu aceeași persoană

#### Caracteristicile distribuției:

- Majoritatea scorurilor de similaritate sunt concentrate între **0.9 și 1.0**, cu un vârf foarte ridicat la **aproximativ 1.0**.
- Există foarte puține perechi de imagini care au scoruri sub **0.8**, ceea ce indică faptul că MediaPipe este capabil să recunoască aceeași persoană chiar și în condiții diferite.
- Forma graficului sugerează că modelul este foarte **robust** la variații precum iluminarea, expresiile faciale sau unghiurile feței.

### Interpretare:

- Algoritmul MediaPipe identifică eficient aceeași persoană în imagini, oferind scoruri foarte mari de similaritate.
- Dacă acest sistem ar fi folosit într-o aplicație de recunoaștere facială, un prag de similaritate de 0.9+ ar putea fi suficient pentru identificarea precisă a unei persoane.
- Numărul foarte mic de scoruri sub 0.8 sugerează că există unele cazuri în care MediaPipe nu reușește să potrivească perfect imaginea, dar acestea sunt excepții.

## 2. Analiza celei de-a doua imagini (100 imagini cu 2 persoane diferite)

### Caracteristicile distribuției:

- Distribuția este încă puternic skewed spre 1.0, ceea ce înseamnă că multe perechi de imagini sunt considerate foarte similare.
- Există însă mai multe valori sub 0.9 comparativ cu prima imagine, ceea ce indică faptul că MediaPipe a reușit să facă diferența între anumite imagini ale celor două persoane.
- Totuși, faptul că multe scoruri sunt peste 0.9 sugerează că există cazuri în care modelul confundă fețele celor două persoane.

### Interpretare:

- Dacă cele două persoane au trăsături faciale similare, modelul ar putea avea dificultăți în a le diferenția, explicând scorurile ridicate de similaritate.
- MediaPipe nu este perfect pentru identificarea persoanelor distincte, mai ales dacă nu sunt diferențe clare în trăsăturile lor.
- Dacă acest sistem ar fi folosit într-un scenariu real (de exemplu, autentificare biometrică), un prag mai ridicat ar fi necesar pentru a evita confuziile.
- Ar fi util să analizăm care sunt cazurile unde similaritatea este mare – poate că imaginile cu unghiuri similare sau expresii neutre sunt mai ușor de confundat.

### Compararea celor două seturi de date

Caracteristică	Set 1 (1 persoană)	Set 2 (2 persoane)
Concentrare scoruri	0.9 - 1.0	0.85 - 1.0
Număr mare de scoruri sub 0.9?	Foarte puține	Mai multe decât în Set 1
Confuzii între persoane?	Nu există (1 singură persoană)	Da, în unele cazuri
Recomandare prag de similaritate	>0.9 pentru identificare precisă	Prag mai ridicat (>0.95) pentru a reduce erorile

### Concluzii și recomandări:

#### 1. Pentru recunoașterea aceleiași persoane (prima imagine)

- MediaPipe funcționează foarte bine, având scoruri ridicate și consistență bună.
- Este robust la variații (expresii, iluminare, unghiuri diferite).
- Pragul de similaritate >0.9 ar fi suficient pentru identificare precisă.

#### 2. Pentru identificarea între două persoane diferite (a doua imagine)

- Există o suprapunere de scoruri, ceea ce sugerează că MediaPipe poate confunda fețele similare.
- Pentru o aplicație reală, ar fi nevoie de un prag mai strict (de ex. >0.95) sau de metode suplimentare pentru a distinge persoanele (ex. embeddings mai avansate).
- Dacă MediaPipe este folosit într-un sistem de autentificare, este posibil să fie nevoie de alte metode complementare (ex. rețele neuronale mai avansate, recunoaștere multi-factor).

## Concluzie

Compararea rezultatelor obținute pentru FaceNet, DeepFace și MediaPipe evidențiază diferențe semnificative între aceste framework-uri de recunoaștere facială, fiecare având avantaje și limitări distincte în funcție de contextul de utilizare.

**FaceNet** s-a remarcat prin acuratețea ridicată a recunoașterii faciale, utilizând o metodă bazată pe învățare metrică și reprezentarea fețelor sub formă de vectori numerici în spațiul de încorporare. Datorită funcției de pierdere triplet, acest model reușește să optimizeze distanța dintre fețele diferite și să apropie fețele similare, oferind o separabilitate clară între identități. În testele efectuate, FaceNet a oferit rezultate solide, menținând un echilibru bun între precizie și viteză de procesare, deși necesită resurse computaționale semnificative pentru inferență și antrenare. În cazul bazelor de date care conțin imagini ale aceleiași persoane, modelul a reușit să păstreze un grad ridicat de similaritate, însă în scenariul ce presupunea diferențierea între persoane distincte, apare necesitatea adăugării unor filtre suplimentare (ex. culoare ochilor, părului, pielii etc.) astfel încât să fie evitate erorile de similaritate între persoane care au trăsături similare din punct de vedere al distanței euclidiene dintre embedding-uri. Totodată, datorită limitărilor resurselor de procesare disponibile, anumite module nu au putut funcționa în maniera dorită.

**DeepFace**, pe de altă parte, utilizează o arhitectură neuronală profundă optimizată pentru recunoașterea facială și introduce un avantaj important prin utilizarea normalizării faciale 3D, ceea ce îi permite să fie mai robust la variațiile de iluminare și poziție ale feței. Deși performanțele sale sunt comparabile cu cele ale FaceNet, DeepFace necesită un volum mare de date pentru antrenare și resurse computaționale semnificative. În testele realizate, DeepFace a demonstrat o capacitate bună de recunoaștere a aceleiași persoane, însă a avut o distribuție mai largă a scorurilor de similaritate comparativ cu FaceNet, ceea ce indică o variabilitate mai mare în răspunsurile sale. În cazul diferențierii între două persoane, modelul a oferit o separare adecvată, dar a avut și câteva confuzii, în special în cazurile în care trăsăturile faciale ale subiecților erau relativ asemănătoare. Astfel, DeepFace este o soluție puternică, dar poate necesita ajustarea pragurilor de similaritate pentru a îmbunătăți diferențierea identităților.

**MediaPipe**, spre deosebire de celelalte două framework-uri, este optimizat pentru viteză și eficiență computațională, fiind ideal pentru aplicații în timp real pe dispozitive mobile sau web. Acest model utilizează un CNN ușor pentru detectarea feței și extragerea unui set minimal de caracteristici faciale, ceea ce îi permite să funcționeze rapid chiar și pe hardware cu resurse limitate. În testele realizate, MediaPipe a avut un comportament excelent în ceea ce privește viteza de procesare, însă precizia sa a fost mai redusă comparativ cu FaceNet și DeepFace. Pentru setul de imagini care conținea aceeași persoană, MediaPipe a oferit scoruri de similaritate ridicate, dar a prezentat o variație mai mare a acestor scoruri, ceea ce sugerează o sensibilitate la factori externi precum iluminarea sau unghiul feței. În cazul în care imaginile aparțineau unor persoane diferite, MediaPipe a întâmpinat dificultăți în diferențierea acestora, având mai multe cazuri în care similaritatea a fost supraestimată, ceea ce indică o predispoziție mai mare la erori false pozitive. Această limitare face ca MediaPipe să nu fie cea mai potrivită alegere pentru aplicații care necesită un nivel ridicat de securitate și acuratețe, însă este o soluție viabilă pentru scenarii care prioritizează eficiența și procesarea rapidă, precum realitatea augmentată, filtre faciale sau aplicații de detectare a expresiilor faciale.

În concluzie, alegerea framework-ului optim depinde de cerințele specifice ale aplicației. Dacă obiectivul principal este acuratețea și fiabilitatea în recunoașterea facială, FaceNet și DeepFace sunt cele mai potrivite opțiuni, deși presupun un consum mai mare de resurse. Dacă, în schimb, viteza și eficiența sunt prioritare, iar precizia poate fi compromisă în favoarea unei implementări mai rapide, MediaPipe reprezintă o soluție excelentă pentru aplicații mobile, AR și scenarii care nu necesită o diferențiere strictă între identități.

## Bibliografie

- [1] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1701-1708).
- [2] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). *FaceNet: A Unified Embedding for Face Recognition and Clustering*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 815-823).
- [3] Lugaresi, C., Tang, J., Nash, H., et al. (2019). *MediaPipe: A Framework for Building Perception Pipelines*.
- [4] Serengil, S. I. (2020). *DeepFace: A Lightweight Face Recognition and Facial Attribute Analysis*.
- [5] Sefik, S. I. (2022). *Deep Face Detection with MediaPipe*.
- [6] Blackcoffer Insights. (2024). *Face Recognition with Deepfills Framework – Deepface*.
- [7] Viso.ai. (2023). *DeepFace - Most Popular Deep Face Recognition in 2024 (Guide)*.
- [8] Vishesh1412. (n.d.). *Celebrity Face Image Dataset*. Disponibil la: <https://www.kaggle.com/datasets/vishesh1412/celebrity-face-image-dataset>
- [9] [https://ai.google.dev/edge/mediapipe/solutions/vision/face\\_detector/python](https://ai.google.dev/edge/mediapipe/solutions/vision/face_detector/python)
- [10] <https://www.techtarget.com/searchenterpriseai/definition/face-detection>
- [11] <https://www.mdpi.com/2079-9292/10/19/2354>
- [12] <https://medium.com/pythons-gurus/what-is-the-best-face-detector-ab650d8c1225>

## ANEXA 1 – FACENET:

```
main.py x
1 import os
2 import numpy as np
3 import cv2
4 from keras_facenet import FaceNet
5 from mtcnn import MTCNN
6 from PIL import Image
7 from sklearn.metrics.pairwise import cosine_similarity
8 import matplotlib.pyplot as plt
9 from scipy.spatial.distance import euclidean
10 from sklearn.metrics import accuracy_score, precision_score
11
12
13 def load_facenet_model(): 1 usage
14     # Initializăm modelul FaceNet din keras-facenet
15     model = FaceNet()
16     return model
17
18
19 def extract_face(image_path, detector, required_size=(160, 160)): 1 usage
20     image = cv2.imread(image_path)
21     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
22     results = detector.detect_faces(image)
23     if results:
24         x, y, width, height = results[0]['box']
25         x, y = abs(x), abs(y)
26         face = image[y:y + height, x:x + width]
27         image = Image.fromarray(face)
28         image = image.resize(required_size)
29         return np.asarray(image)
30     return None
```

```
32
33 def get_embedding(model, face_pixels): 1 usage
34     # Keras-Facenet asteaptă un input de tip float32 pentru face_pixels
35     face_pixels = face_pixels.astype('float32')
36     mean, std = face_pixels.mean(), face_pixels.std()
37     face_pixels = (face_pixels - mean) / std
38     embedding = model.embeddings(np.expand_dims(face_pixels, axis=0)) # Obținem embedding-ul
39     return embedding[0] # Returnăm embedding-ul pentru un singur chip
40
41
42 def analyze_faces(image_folder): 1 usage
43     detector = MTCNN()
44     model = load_facenet_model()
45     embeddings = {}
46
47     for image_name in os.listdir(image_folder):
48         image_path = os.path.join(image_folder, image_name)
49         face = extract_face(image_path, detector)
50         if face is not None:
51             embedding = get_embedding(model, face)
52             embeddings[image_name] = embedding
53
54     return embeddings
55
56
57 def compare_faces(embeddings): 1 usage
58     names = list(embeddings.keys())
59     vectors = np.array(list(embeddings.values()))
60     cosine_similarities = cosine_similarity(vectors)
61     euclidean_distances = [[euclidean(vectors[i], vectors[j]) for j in range(len(names))] for i in range(len(names))]
62
```

```

63     # Modificăm logică pentru a obține etichetele adevărate și prezise
64     labels_true = []
65     labels_pred = []
66
67     similarity_threshold = 0.5 # Adjustable threshold for cosine similarity
68
69     # Comparăm fiecare pereche de fațe
70     for i in range(len(names)):
71         for j in range(i + 1, len(names)):
72             # Determinăm eticheta adevărată pe baza numelui fișierului
73             label_true = 1 if "same_person" in names[i] and "same_person" in names[j] else 0
74             labels_true.append(label_true)
75
76             # Comparăm similaritatea dintre cele două fațe
77             similarity = cosine_similarities[i][j]
78             label_pred = 1 if similarity > similarity_threshold else 0
79             labels_pred.append(label_pred)
80
81     # Verificăm dacă lungimile listelor sunt corecte
82     if len(labels_true) != len(labels_pred):
83         print(f"Error: Mismatched lengths! {len(labels_true)} != {len(labels_pred)}")
84         return

```

```

57 def compare_faces(embeddings): 1 usage
58
59     precision = precision_score(labels_true, labels_pred, zero_division=1)
60     accuracy = accuracy_score(labels_true, labels_pred)
61     error_rate = 1 - accuracy
62
63     print(f"Precision: {precision:.2f}")
64     print(f"Accuracy: {accuracy:.2f}")
65     print(f"Error Rate: {error_rate:.2f}")
66
67     for i in range(len(names)):
68         for j in range(i + 1, len(names)):
69             print(f"Cosine Similarity between {names[i]} and {names[j]}: {cosine_similarities[i][j]:.2f}")
70             print(f"Euclidean Distance between {names[i]} and {names[j]}: {euclidean_distances[i][j]:.2f}")
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 def plot_similarity_distribution(similarities): 1 usage
101     plt.hist(similarities, bins=20, alpha=0.75, color='blue')
102     plt.xlabel("Similarity Score")
103     plt.ylabel("Frequency")
104     plt.title("Distribution of Face Similarities")
105     plt.show()
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



## ANEXA 2 – DEEPFACE:

```
main.py x
1  import os
2  import numpy as np
3  import cv2
4  from deepface import DeepFace
5  from mtcnn import MTCNN
6  from PIL import Image
7  from sklearn.metrics.pairwise import cosine_similarity
8  import matplotlib.pyplot as plt
9  from scipy.spatial.distance import euclidean
10 from sklearn.metrics import accuracy_score, precision_score
11
12 def extract_face(image_path, detector, required_size=(160, 160)): 1 usage
13     image = cv2.imread(image_path)
14     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
15     results = detector.detect_faces(image)
16     if results:
17         x, y, width, height = results[0]['box']
18         x, y = abs(x), abs(y)
19         face = image[y:y + height, x:x + width]
20         image = Image.fromarray(face)
21         image = image.resize(required_size)
22         return np.asarray(image)
23     return None
24
25 def get_embedding(image_path): 1 usage
26     embedding = DeepFace.represent(img_path=image_path, model_name="Facenet", enforce_detection=False)
27     return np.array(embedding[0]['embedding'])
28
29 def analyze_faces(image_folder): 1 usage
30     detector = MTCNN()
31     embeddings = {}
32
33     for image_name in os.listdir(image_folder):
34         image_path = os.path.join(image_folder, image_name)
35         face = extract_face(image_path, detector)
36         if face is not None:
37             embedding = get_embedding(image_path)
38             embeddings[image_name] = embedding
39
40     return embeddings
41
42 def compare_faces(embeddings): 1 usage
43     names = list(embeddings.keys())
44     vectors = np.array(list(embeddings.values()))
45     cosine_similarities = cosine_similarity(vectors)
46     euclidean_distances = [[euclidean(vectors[i], vectors[j]) for j in range(len(names))] for i in range(len(names))]
47
48     labels_true = []
49     labels_pred = []
50     similarity_threshold = 0.5
51
52     for i in range(len(names)):
53         for j in range(i + 1, len(names)):
54             label_true = 1 if "same_person" in names[i] and "same_person" in names[j] else 0
55             labels_true.append(label_true)
56
57             similarity = cosine_similarities[i][j]
58             label_pred = 1 if similarity > similarity_threshold else 0
59             labels_pred.append(label_pred)
```

```

42 def compare_faces(embeddings): 1 usage
43
44
45
46
47
48
49
50
51 precision = precision_score(labels_true, labels_pred, zero_division=1)
52 accuracy = accuracy_score(labels_true, labels_pred)
53 error_rate = 1 - accuracy
54
55 print(f"Precision: {precision:.2f}")
56 print(f"Accuracy: {accuracy:.2f}")
57 print(f"Error Rate: {error_rate:.2f}")
58
59 for i in range(len(names)):
60     for j in range(i + 1, len(names)):
61         print(f"Cosine Similarity between {names[i]} and {names[j]}: {cosine_similarities[i][j]:.2f}")
62         print(f"Euclidean Distance between {names[i]} and {names[j]}: {euclidean_distances[i][j]:.2f}")
63
64
65
66
67
68
69
70
71
72
73
74 def plot_similarity_distribution(similarities): 1 usage
75 plt.hist(similarities, bins=20, alpha=0.75, color='blue')
76 plt.xlabel("Similarity Score")
77 plt.ylabel("Frequency")
78 plt.title("Distribution of Face Similarities")
79 plt.show()
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
```

## ANEXA 3 – MEDIAPIPE:

```
main.py ×
1 import os
2 import numpy as np
3 import cv2
4 import mediapipe as mp
5 from PIL import Image
6 from sklearn.metrics.pairwise import cosine_similarity
7 import matplotlib.pyplot as plt
8 from scipy.spatial.distance import euclidean
9 from sklearn.metrics import accuracy_score, precision_score
10
11 # Initializare MediaPipe Face Detection
12 mp_face_detection = mp.solutions.face_detection
13 mp_face_model = mp_face_detection.FaceDetection(model_selection=1, min_detection_confidence=0.5)
14
15
16 def extract_face(image_path, required_size=(160, 160)):
17     """Extrage fața din imagine și o redimensionează."""
18     image = cv2.imread(image_path)
19     if image is None:
20         return None # Verificare dacă imaginea a fost încărcată corect
21     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
22
23     results = mp_face_model.process(image)
24     if results.detections:
25         for detection in results.detections:
26             bbox = detection.location_data.relative_bounding_box
27             h, w, _ = image.shape
28             x, y, width, height = (int(bbox.xmin * w), int(bbox.ymin * h), int(bbox.width * w), int(bbox.height * h))
29             x, y = max(x, 0), max(y, 0)
30             face = image[y:y + height, x:x + width]
31
```

```
main.py ×
16 def extract_face(image_path, required_size=(160, 160)):
32     if face.size == 0:
33         return None # Evităm erori în cazul unei decupări incorecte
34
35     image_resized = Image.fromarray(face).resize(required_size)
36     return np.asarray(image_resized, dtype=np.float32) / 255.0 # Normalizare între 0 și 1
37
38     return None
39
40
41 def get_embedding(image_path): 1 usage
42     """Obține vectorul de trăsături al fetei folosind MediaPipe."""
43     image = cv2.imread(image_path)
44     if image is None:
45         return None
46
47     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
48     results = mp_face_model.process(image_rgb)
49
50     if results.detections:
51         for detection in results.detections:
52             bbox = detection.location_data.relative_bounding_box
53             return np.array([bbox.xmin, bbox.ymin, bbox.width, bbox.height], dtype=np.float32) # Vector numeric fix
54     return None
55
```

```

57 def analyze_faces(image_folder): 1 usage
58     embeddings = {}
59
60     for image_name in os.listdir(image_folder):
61         image_path = os.path.join(image_folder, image_name)
62         embedding = get_embedding(image_path)
63
64         if embedding is not None:
65             embeddings[image_name] = embedding
66
67     return embeddings
68
69
70 def compare_faces(embeddings): 1 usage
71     if len(embeddings) < 2:
72         print("Nu sunt suficiente imagini pentru comparatie.")
73         return
74
75     names = list(embeddings.keys())
76     vectors = np.array(list(embeddings.values()), dtype=np.float32) # Convertire sigură
77
78     cosine_similarities = cosine_similarity(vectors)
79     euclidean_distances = np.array(
80         [[euclidean(vectors[i], vectors[j]) for j in range(len(names))] for i in range(len(names))])
81
82     labels_true, labels_pred = [], []
83     similarity_threshold = 0.5

```

```

70 def compare_faces(embeddings): 1 usage
84
85     for i in range(len(names)):
86         for j in range(i + 1, len(names)):
87             label_true = 1 if "same_person" in names[i] and "same_person" in names[j] else 0
88             labels_true.append(label_true)
89
90             similarity = cosine_similarities[i][j]
91             label_pred = 1 if similarity > similarity_threshold else 0
92             labels_pred.append(label_pred)
93
94     precision = precision_score(labels_true, labels_pred, zero_division=1)
95     accuracy = accuracy_score(labels_true, labels_pred)
96     error_rate = 1 - accuracy
97
98     print(f"Precision: {precision:.2f}")
99     print(f"Accuracy: {accuracy:.2f}")
100    print(f"Error Rate: {error_rate:.2f}")
101
102    for i in range(len(names)):
103        for j in range(i + 1, len(names)):
104            print(f"Cosine Similarity between {names[i]} and {names[j]}: {cosine_similarities[i][j]:.2f}")
105            print(f"Euclidean Distance between {names[i]} and {names[j]}: {euclidean_distances[i][j]:.2f}")
106
107
108 def plot_similarity_distribution(similarities): 1 usage
109     """Afisează distribuția similarităților într-un histogramă."""
110     plt.hist(similarities, bins=20, alpha=0.75, color='blue')
111     plt.xlabel("Similarity Score")
112     plt.ylabel("Frequency")
113     plt.title("Distribution of Face Similarities")
114     plt.show()

```

## ANEXA 4 – FACENET

```
1 import os
2 import numpy as np
3 import cv2
4 from keras_facenet import FaceNet
5 from mtcnn import MTCNN
6 from PIL import Image
7 from sklearn.metrics.pairwise import cosine_similarity
8 import matplotlib.pyplot as plt
9
10
11 def load_facenet_model():
12     return FaceNet()
13
14
15 def extract_face(image_path, detector, required_size=(160, 160)):
16     image = cv2.imread(image_path)
17     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
18     results = detector.detect_faces(image)
19     if results:
20         x, y, width, height = results[0]['box']
21         x, y = abs(x), abs(y)
22         face = image[y:y + height, x:x + width]
23         image = Image.fromarray(face)
24         image = image.resize(required_size)
25         return np.asarray(image)
26     return None
27
28
29 def get_embedding(model, face_pixels):
30     face_pixels = face_pixels.astype('float32')
31     mean, std = face_pixels.mean(), face_pixels.std()
32     face_pixels = (face_pixels - mean) / std
33     embedding = model.embeddings(np.expand_dims(face_pixels, axis=0))
34     return embedding[0]
35
36
37 def analyze_faces(image_folder):
38     detector = MTCNN()
39     model = load_facenet_model()
40     embeddings = {}
41
42     for image_name in os.listdir(image_folder):
43         image_path = os.path.join(image_folder, image_name)
44         face = extract_face(image_path, detector)
45         if face is not None:
46             embedding = get_embedding(model, face)
47             embeddings[image_name] = embedding
48
49     return embeddings
50
51
52 def find_matching_faces(input_image_path, embeddings, model, threshold=0.5):
53     detector = MTCNN()
54     face = extract_face(input_image_path, detector)
55     if face is None:
56         print("No face detected in input image.")
57         return []
58
59     input_embedding = get_embedding(model, face)
60     matching_faces = []
61
62     for name, emb in embeddings.items():
63         similarity = cosine_similarity([input_embedding], [emb])[0][0]
64         if similarity > threshold:
65             matching_faces.append((name, similarity))
66
67     return sorted(matching_faces, key=lambda x: x[1], reverse=True)
68
69
70 if __name__ == "__main__":
71     image_folder = "C:/Users/alex/PycharmProjects/PythonProject5/Images"
72     test_image_path = "C:/Users/alex/PycharmProjects/PythonProject5/Test/test_image.jpeg"
```

```

73
74     model = load_facenet_model()
75     embeddings = analyze_faces(image_folder)
76     matches = find_matching_faces(test_image_path, embeddings, model)
77
78     if matches:
79         print("Matching faces:")
80         for name, score in matches:
81             print(f"{name}: {score:.2f}")
82     else:
83         print("No matching faces found.")

```

```

C:\Users\alexa\PycharmProjects\PythonProject5\.venv\Scripts\python.exe C:\Users\alexa\PycharmProjects\PythonProject5\main.py
2025-02-09 12:28:25.316655: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2025-02-09 12:28:25.317020: I tensorflow/stream_executor/cuda/cuda_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2025-02-09 12:28:28.081405: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library nvcuda.dll
2025-02-09 12:28:29.047917: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with properties:
pciBusID: 0000:01:00.0 name: NVIDIA GeForce GTX 1650 Ti computeCapability: 7.5
coreClock: 1.4856Hz coreCount: 16 deviceMemorySize: 4.00618 deviceMemoryBandwidth: 178.84618/s
2025-02-09 12:28:29.048786: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2025-02-09 12:28:29.049403: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cublas64_11.dll'; dlerror: cublas64_11.dll not found
2025-02-09 12:28:29.050003: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cublasLt64_11.dll'; dlerror: cublasLt64_11.dll not found
2025-02-09 12:28:29.050690: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cufft64_10.dll'; dlerror: cufft64_10.dll not found
2025-02-09 12:28:29.051432: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'curand64_10.dll'; dlerror: curand64_10.dll not found
2025-02-09 12:28:29.052048: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cusolver64_11.dll'; dlerror: cusolver64_11.dll not found
2025-02-09 12:28:29.052667: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cusparse64_11.dll'; dlerror: cusparse64_11.dll not found
2025-02-09 12:28:29.053274: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudnn64_8.dll'; dlerror: cudnn64_8.dll not found
2025-02-09 12:28:29.053445: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1766] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed.
Skipping registering GPU devices...
2025-02-09 12:28:29.054012: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the followi
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-02-09 12:28:29.055197: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect StreamExecutor with strength 1 edge matrix:
2025-02-09 12:28:29.055358: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]
2025-02-09 12:28:33.359716: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered 2)
WARNING:tensorflow:5 out of the last 1566 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001BC8199C67B> triggered tf.function retracing. Tracing is e
WARNING:tensorflow:6 out of the last 1567 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001BC81727168> triggered tf.function retracing. Tracing is e

```



## ANEXA 5 – DEEPPFACE

```
1 import os
2 import numpy as np
3 import cv2
4 from deepface import DeepFace
5 from mtcnn import MTCNN
6 from PIL import Image
7 from sklearn.metrics.pairwise import cosine_similarity
8 import matplotlib.pyplot as plt
9
10
11 def extract_face(image_path, detector, required_size=(160, 160)):
12     image = cv2.imread(image_path)
13     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
14     results = detector.detect_faces(image)
15     if results:
16         x, y, width, height = results[0]['box']
17         x, y = abs(x), abs(y)
18         face = image[y:y + height, x:x + width]
19         image = Image.fromarray(face)
20         image = image.resize(required_size)
21         return np.asarray(image)
22     return None
23
24
25 def get_embedding(image_path):
26     embedding = DeepFace.represent(img_path=image_path, model_name="Facenet", enforce_detection=False)
27     return np.array(embedding[0]['embedding'])
28
29
30 def analyze_faces(image_folder):
31     detector = MTCNN()
32     embeddings = {}
33
34     for image_name in os.listdir(image_folder):
35         image_path = os.path.join(image_folder, image_name)
36         face = extract_face(image_path, detector)
37         if face is not None:
38             embedding = get_embedding(image_path)
39
40             embedding = get_embedding(image_path)
41             embeddings[image_name] = embedding
42
43     return embeddings
44
45
46 def find_matching_faces(input_image_path, embeddings, threshold=0.5):
47     input_embedding = get_embedding(input_image_path)
48     matching_faces = []
49
50     for name, emb in embeddings.items():
51         similarity = cosine_similarity([input_embedding], [emb])[0][0]
52         if similarity > threshold:
53             matching_faces.append((name, similarity))
54
55     return sorted(matching_faces, key=lambda x: x[1], reverse=True)
56
57
58 if __name__ == "__main__":
59     image_folder = "C:/Users/alex/PycharmProjects/PythonProject6/Images"
60     test_image_path = "C:/Users/alex/PycharmProjects/PythonProject6/Test/test_image.jpeg"
61
62     embeddings = analyze_faces(image_folder)
63     matches = find_matching_faces(test_image_path, embeddings)
64
65     if matches:
66         print("Matching faces:")
67         for name, score in matches:
68             print(f"{name}: {score:.2f}")
69     else:
70         print("No matching faces found.")
```

```
Run main x
2/2 [=====] - 0s 4ms/step
1/1 [=====] - 0s 19ms/step
Matching faces:
085_b44c8d35.jpg: 0.89
065_343bfe69.jpg: 0.84
086_9b44c502.jpg: 0.80
064_9ac818ed.jpg: 0.80
058_4ed85318.jpg: 0.79
063_fe99146b.jpg: 0.78
096_84100579.jpg: 0.77
053_8075590e.jpg: 0.77
055_51880515.jpg: 0.73
068_72330c63.jpg: 0.73
081_725684cb.jpg: 0.73
080_f5386479.jpg: 0.69
067_fb66d8ac.jpg: 0.69
054_d500397d.jpg: 0.68
088_ff699567.jpg: 0.66
091_5307a177.jpg: 0.65
077_a908452a.jpg: 0.65
056_f2f25510.jpg: 0.64
059_01c01e72.jpg: 0.59
099_0d04ca3f.jpg: 0.59
070_6192068e.jpg: 0.58
078_adefb117.jpg: 0.55
097_833dba65.jpg: 0.55
083_f284dc2b.jpg: 0.54

Process finished with exit code 0
```



## ANEXA 6 – MEDIAPIPE

```
1 import os
2 import numpy as np
3 import cv2
4 import mediapipe as mp
5 from PIL import Image
6 from sklearn.metrics.pairwise import cosine_similarity
7
8 # Inițializare MediaPipe Face Detection
9 mp_face_detection = mp.solutions.face_detection
10 mp_face_model = mp_face_detection.FaceDetection(model_selection=1, min_detection_confidence=0.5)
11
12
13 def extract_face(image_path, required_size=(160, 160)):
14     """Extrage fața din imagine și o redimensionează."""
15     image = cv2.imread(image_path)
16     if image is None:
17         return None # Imaginea nu a fost încărcată corect
18
19     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
20     results = mp_face_model.process(image)
21
22     if results.detections:
23         for detection in results.detections:
24             bbox = detection.location_data.relative_bounding_box
25             h, w, _ = image.shape
26             x, y, width, height = (int(bbox.xmin * w), int(bbox.ymin * h), int(bbox.width * w), int(bbox.height * h))
27             x, y = max(x, 0), max(y, 0)
28             face = image[y:y + height, x:x + width]
29
30             if face.size == 0:
31                 return None # Evităm erori în cazul unei decupări incorecte
32
33             image_resized = Image.fromarray(face).resize(required_size)
34             return np.asarray(image_resized, dtype=np.float32) / 255.0 # Normalizare între 0 și 1
35
36     return None
```

```
39 def get_embedding(image_path):
40     """Obține vectorul de trăsături al feței folosind MediaPipe."""
41     image = cv2.imread(image_path)
42     if image is None:
43         return None
44
45     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
46     results = mp_face_model.process(image_rgb)
47
48     if results.detections:
49         for detection in results.detections:
50             bbox = detection.location_data.relative_bounding_box
51             return np.array([bbox.xmin, bbox.ymin, bbox.width, bbox.height], dtype=np.float32) # Vector numeric fix
52     return None
53
54
55 def analyze_faces(image_folder):
56     """Analizează toate fețele dintr-un folder și returnează embeddings."""
57     if not os.path.exists(image_folder):
58         print(f"Folderul specificat nu există: {image_folder}")
59         return {}
60
61     embeddings = {}
62
63     for image_name in os.listdir(image_folder):
64         image_path = os.path.join(image_folder, image_name)
65         embedding = get_embedding(image_path)
66
67         if embedding is not None:
68             embeddings[image_name] = embedding
69
70     return embeddings
71
72
73 def match_face(target_image, database_embeddings, threshold=0.5):
74     """Compară o imagine de referință cu baza de date și returnează cele mai bune potriviri."""
75     target_embedding = get_embedding(target_image)
```

```

77     if target_embedding is None:
78         print("Nu s-a putut extrage fața din imaginea de referință.")
79         return []
80
81     similarities = {}
82
83     for name, emb in database_embeddings.items():
84         similarity = cosine_similarity([target_embedding], [emb])[0][0]
85         if similarity >= threshold: # Doar imaginile peste un anumit prag sunt afișate
86             similarities[name] = similarity
87
88     # Sortare descrescătoare după similaritate
89     sorted_matches = sorted(similarities.items(), key=lambda x: x[1], reverse=True)
90
91     print("\nImagini potrivite:")
92     for name, sim in sorted_matches:
93         print(f"{name}: Similaritate {sim:.2f}")
94
95     return sorted_matches
96
97
98 if __name__ == "__main__":
99     database_folder = "C:/Users/alex/PycharmProjects/PythonProject5/Images"
100     target_image_path = "C:/Users/alex/PycharmProjects/PythonProject5/Test/test_image.jpeg"
101
102     print("Analizăm baza de date...")
103     database_embeddings = analyze_faces(database_folder)
104
105     if database_embeddings:
106         print(f"Comparăm {target_image_path} cu baza de date...")
107         matches = match_face(target_image_path, database_embeddings)
108
109         if not matches:
110             print("Nu s-au găsit potriviri peste pragul setat.")
111         else:
112             print("Nu s-au găsit fețe valide în baza de date.")
113

```

```

Run main x
Imagini potrivite:
096_9ff8b67f.jpg: Similaritate 1.00
061_d844ec74.jpg: Similaritate 1.00
062_92c7c5ef.jpg: Similaritate 1.00
068_b8b844e2.jpg: Similaritate 0.99
068_72330c63.jpg: Similaritate 0.99
092_8f204d71.jpg: Similaritate 0.99
071_6e010995.jpg: Similaritate 0.99
063_10c723ae.jpg: Similaritate 0.99
097_51a924c1.jpg: Similaritate 0.99
096_32a79dde.jpg: Similaritate 0.99
062_2539f58f.jpg: Similaritate 0.99
074_a86aab43.jpg: Similaritate 0.99
054_5a9566eb.jpg: Similaritate 0.99
059_d09c7676.jpg: Similaritate 0.99
061_9885f065.jpg: Similaritate 0.99
089_16077370.jpg: Similaritate 0.99
084_43b55cdf.jpg: Similaritate 0.99
085_c957f9b7.jpg: Similaritate 0.99
060_f5a38fe0.jpg: Similaritate 0.99
070_4de04405.jpg: Similaritate 0.99
094_7858a9ff.jpg: Similaritate 0.99
070_6192068e.jpg: Similaritate 0.99
056_ce82b1f.jpg: Similaritate 0.99
099_61c7b659.jpg: Similaritate 0.98
083_f284dc2b.jpg: Similaritate 0.98
062_81d4fb18.jpg: Similaritate 0.98
058_4ed85318.jpg: Similaritate 0.98
066_0bdb9ac3.jpg: Similaritate 0.98
086_c7665b8f.jpg: Similaritate 0.98

```

```

Run main x
086_c7665b8f.jpg: Similaritate 0.98
096_84100579.jpg: Similaritate 0.98
100_6cee7c73.jpg: Similaritate 0.98
064_9ac818ed.jpg: Similaritate 0.98
064_1b7da2f6.jpg: Similaritate 0.98
053_8075590e.jpg: Similaritate 0.98
063_c6f8603d.jpg: Similaritate 0.98
077_a908452a.jpg: Similaritate 0.98
080_f5386479.jpg: Similaritate 0.98
053_6ad198c0.jpg: Similaritate 0.98
098_f4f39b7c.jpg: Similaritate 0.98
057_d2d179a5.jpg: Similaritate 0.98
083_9436dbb0.jpg: Similaritate 0.98
057_8572457a.jpg: Similaritate 0.98
056_64fde18d.jpg: Similaritate 0.97
068_712660e4.jpg: Similaritate 0.97
088_ff699567.jpg: Similaritate 0.97
073_f39658d0.jpg: Similaritate 0.97
088_f6c2c5d2.jpg: Similaritate 0.96
078_adeffb17.jpg: Similaritate 0.96
065_86617a5e.jpg: Similaritate 0.96
055_30dc3531.jpg: Similaritate 0.96
090_f67e981f.jpg: Similaritate 0.96
060_19a80cff.jpg: Similaritate 0.96
083_a685594b.jpg: Similaritate 0.96
070_257cc94f.jpg: Similaritate 0.96
055_51880515.jpg: Similaritate 0.96
054_d500397d.jpg: Similaritate 0.96
061_b36635a2.jpg: Similaritate 0.95
074_a880665b.jpg: Similaritate 0.95
057_11a0b23b.jpg: Similaritate 0.94

```

```

Run main x
057_b1a0b23b.jpg: Similaritate 0.94
065_016dc8e2.jpg: Similaritate 0.94
056_f2f25510.jpg: Similaritate 0.94
085_b44c8d35.jpg: Similaritate 0.94
080_40110c00.jpg: Similaritate 0.94
059_01c01e72.jpg: Similaritate 0.94
077_9bcf9b8c.jpg: Similaritate 0.93
053_e8b9dd3f.jpg: Similaritate 0.93
067_fb66d8ac.jpg: Similaritate 0.93
086_8ee9e9d0.jpg: Similaritate 0.93
055_9508128d.jpg: Similaritate 0.93
099_b7d16411.jpg: Similaritate 0.93
082_d9295121.jpg: Similaritate 0.91
086_9b44c502.jpg: Similaritate 0.91
059_089c6dcd.jpg: Similaritate 0.90
097_833dba65.jpg: Similaritate 0.90
065_343bfe69.jpg: Similaritate 0.90
064_61385762.jpg: Similaritate 0.90
058_7f95baf4.jpg: Similaritate 0.90
060_58b1413c.jpg: Similaritate 0.89
081_725684cb.jpg: Similaritate 0.89
099_0d04ca3f.jpg: Similaritate 0.88
094_087829ab.jpg: Similaritate 0.88
094_b043e45c.jpg: Similaritate 0.88
073_9d58f7ef.jpg: Similaritate 0.87
081_7e097924.jpg: Similaritate 0.86
072_5372b2f8.jpg: Similaritate 0.85
088_fb4a5e01.jpg: Similaritate 0.83
089_2539e4d9.jpg: Similaritate 0.82
093_4876af22.jpg: Similaritate 0.81
091_c6186767.jpg: Similaritate 0.81

```

```
091_cc106747.jpg: Similaritate 0.81  
063_fe99146b.jpg: Similaritate 0.78  
093_4027b1e2.jpg: Similaritate 0.77  
078_d36c2be3.jpg: Similaritate 0.74  
091_930fe784.jpg: Similaritate 0.73  
058_1b5b5422.jpg: Similaritate 0.68  
054_ca0da4ca.jpg: Similaritate 0.65  
067_fb02357d.jpg: Similaritate 0.62  
091_5307a177.jpg: Similaritate 0.62
```

```
Process finished with exit code 0
```