Program de analiză a trăsăturilor morfologice ale feței

Student: Barna Ioana Melisa

Universitatea Tehnică din Cluj-Napoca

28 Noiembrie, 2024

Cuprins

1. Introducere 4

1.1 Context 4

1.2 Obiective 4

1.3 Domeniu 5

1.4 Motivaţie 5

2. Studiu bibliografic 6

2.1 Ce este recunoaşterea trăsăturilor morfologice ale feţei? 6

2.2 Algoritmi de detecţie facială 6

3. Analiza & Design 7

a. Analiza 7

3.1 Propunerea Proiectului 7

3.2 Analiza Funcţionalităţilor 8

3.2.1 Detectarea Trăsăturilor Faciale 8

3.2.2 Preprocesarea Imaginilor 8

3.2.3 Interacţiunea cu Utilizatorul 9

3.2.4 Experienţa Utilizatorului: Feedback Vizual şi Interactiv 9

3.3 Scenarii de Utilizare 9

3.4 Limitări şi Posibile Îmbunătăţiri 10

b. Design 10

3.5 Arhitectura Sistemului 10

3.6 Designul Modulului de Preprocesare 12

3.7 Designul Algoritmului de Detectare a Trăsăturilor 12

3.8 Interfaţa cu Utilizatorul 13

3.9 Fluxul de Date 14

4. Implementare 16
4.1 Iniţializarea Dependinţelor şi Configurarea Mediului 16
4.2 Detectarea Componentelor Faciale 17
4.3 Procesarea Imaginilor din Fişiere 19
4.4 Crearea Interfeţei Grafice cu Tkinter 21

5. Testare & Validare 22
5.1 Testare analiza morfologică din imagini statice 22
5.2 Testare analiza morfologică în timp real 23
5.3 Testare funcţionalitatea interfeţei grafice 24
5.4 Scenarii de Testare 25
5.5 Probleme Identificate şi Soluţii 26
5.5.1 Detectare imprecisă în condiţii de iluminare slabă 26
5.5.2 Performanţă scăzută pe camere de rezoluţie joasă 27
5.5.3 Lag în timpul detecţiei în timp real 27

6. Concluzii 27

7. Bibliografie 28

Introducere

1.1 Context

Programele de analiză a trăsăturilor morfologice ale feţei sunt deja un subiect bine explorat, stând la baza tehnologiei de recunoaştere facială. Un exemplu cunoscut este sistemul "FaceID", pe care îl folosim aproape zilnic, fie pentru deblocarea telefonului mobil, fie pentru interfoanele inteligente. Această tehnologie are un rol esenţial în protejarea informaţiilor noastre personale, fiind o componentă importantă a sistemelor moderne de securitate**.**

Dacă algoritmii de recunoaştere facială nu sunt suficient de precişi, există riscul ca dispozitivele să confunde o altă persoană cu utilizatorul real, ceea ce ar putea compromite spaţiul privat şi expune informaţiile confidenţiale. Din acest motiv, proiectul are o importanţă foarte mare atât pentru studiul tehnic, cât şi pentru societate, având în vedere că este bazat pe inteligenţa artificială, o tehnologie care evoluează constant.

1.2 Obiective

Proiectul va fi realizat in Python, avand in vedere varietatea de librarii pe care acest limbaj de programare il ofera. Ca obiectiv principal, scopul proiectului este de a detecta in timp real trasaturile morfologice ale fetei, cum ar fi nasul, gura si ochii. Inainte de a ajunge la varianta finala, proiectul va avea o etapa de simulare, in care voi selecta imagini iar programul va trebui să identifice trăsăturile menţionate anterior.

Dupa acest prim pas vom trece la imbunatatirea codului, respectiv de a-l face mai interactiv, lucru realizat prin integrarea cu camera laptopului. De asemenea, varianta de simulare nu va dispărea definitiv din cod, utilizatorul având posibilitatea să aleagă între opţiunea de a selecta o imagine sau de a realiza detectarea în timp real folosind camera. Ca si obiective specifice avem:

->implementarea unui algoritmul de detecţie facială folosind biblioteci precum **dlib** şi **OpenCV**, care permit recunoaşterea trăsăturilor faciale din imagini;

->implementarea modelului **shape_predictor_68_face_landmarks.dat**, folosit pentru detectarea precisă a trăsăturilor feţei;

->implementarea unei interfețe grafice simple, folosind **Tkinter**, care să permită utilizatorilor să încarce imagini din sistem și să proceseze automat aceste imagini pentru detectarea componentelor faciale;

1.3 Domeniu

Proiectul de recunoaştere a trăsăturilor morfologice ale feţei se încadrează în domeniul viziunii computerizate, al inteligenţei artificiale şi procesării de imagini. Având în vedere că viziunea computerizată este un domeniu foarte important, putem afirma că acesta se ocupă cu dezvoltarea unor algoritmi care permit computerelor să interpreteze şi să „vadă” imagini.

Pe de altă parte, procesarea de imagini se referă la tehnicile prin care imaginile sunt analizate şi transformate, fiind o etapă esenţială în recunoaşterea trăsăturilor faciale.

1.4 Motivaţie

Acest proiect reprezintă o bază importantă a tehnologiei actuale, motivându-mă să dezvolt această latură şi să explorez ramurile sale diverse. Recunoaşterea trăsăturilor morfologice aduce avantaje în multe domenii, inclusiv în tehnologie, educaţie şi sănătate.

Proiectul poate îmbunătăţi securitatea sistemelor, facilita interacţiuni personalizate în educaţie şi contribui la soluţii inovatoare în domeniul sănătăţii**.**

Studiu bibliographic

2.1 Ce este recunoaşterea trăsăturilor morfologice ale feţei?

Recunoaşterea trăsăturilor morfologice ale feţei se referă la procesul de identificare şi analizare a caracteristicilor fizice ale feţei umane pentru a distinge între diferite persoane. Acest proces implică capturarea imaginii feţei, urmată de analiza geometrică a trăsăturilor, cum ar fi distanţa dintre ochi, forma nasului şi lăţimea feţei.

Aceste date sunt apoi folosite pentru a crea o "semnătură facială" unică, care poate fi comparată cu imagini stocate în baze de date pentru a verifica identitatea unei persoane.

2.2 Algoritmi de detecţie facială

Algoritmii de detecţie facială sunt tehnici utilizate pentru a identifica feţele umane în imagini sau în fluxuri video. Proiectul foloseşte **dlib**, **o bibliotecă populară pentru detecţia şi recunoaşterea feţelor**, împreună cu OpenCV pentru procesarea imaginilor.

Proiectul se bazează pe doi algoritmi principali: detectorul de feţe HOG (Histogram of Oriented Gradients) şi un model de predicţie a punctelor caracteristice.

1. **Detectorul de Feţe HOG** analizează imaginea prin divizarea acesteia în celule şi calculează gradientele de intensitate pentru a identifica contururile şi formele caracteristice ale feţei. Acesta este implementat în cod prin dlib.get_frontal_face_detector().

2. Modelul de Predicţie utilizează un model pre-antrenat, **shape_predictor_68_face_landmarks.dat**, care a fost antrenat să identifice 68 de puncte caracteristice ale feţei, inclusiv contururile ochilor, nasului şi gurii. Acesta este încărcat cu dlib.shape_predictor() şi folosit pentru a estima poziţia acestor puncte pe baza caracteristicilor extrase din imagine.

Analiză & Design

a. Analiză

3.1 Propunerea Proiectului

Varianta finală a programului de recunoaştere a trăsăturilor morfologice va cuprinde caracteristicile descrise mai jos.

Nr.	Funcţionalitate	Descriere
1	Încărcarea imaginii	Aplicaţia permite încărcarea unei imagini din sistemul de fişiere pentru procesare şi detectarea trăsăturilor faciale.
2	Preprocesarea imaginii	Imaginea este preprocesată (redimensionare, conversie la nuanţe de gri) pentru a îmbunătăţi performanţa algoritmilor de detecţie.
3	Captarea fluxului video	A doua versiune a aplicaţiei va avea funcţionalitatea de a utiliza camera, reuşind astfel să evidenţieze în timp real trăsăturile morfologice ale unei persoane.
4	Detectarea trăsăturilor faciale	Algoritmul HOG şi shape_predictor_68_face_landmarks.dat sunt utilizate pentru a detecta şi marca punctele caracteristice ale feţei.
5	Evidenţierea trăsăturilor detectate	Aplicaţia marchează trăsăturile feţei (nas, gură, ochi etc.) direct pe imagine sau pe fluxul video, pentru vizualizare în timp real.
6	Interfaţă grafică pentru utilizator	Utilizatorul poate încărca imagini, activa camera, vizualiza rezultatele detecţiei şi naviga între modurile de funcţionare (imagine şi video).
7	Feedback vizual	Aplicaţia oferă un feedback vizual clar despre trăsăturile detectate, prin conturarea lor în timp real.
8	Funcţionalităţi pentru gestionarea camerei	Aplicaţia permite două tipuri de detectare a trăsăturilor faciale: detectarea din imaginile încărcate de utilizator şi detectarea în timp real, unde trăsăturile sunt identificate direct pe faţa utilizatorului prin cameră.

3.2 Analiza Funcţionalităţilor

3.2.1 Detectarea Trăsăturilor Faciale

Aplicaţia utilizează algoritmi de detecţie facială pentru a identifica şi localiza feţele în imagini. Unul dintre cele mai eficiente metode este metoda **HOG** (**Histogram of Oriented Gradients**), care analizează imaginea pentru a extrage trăsăturile vizuale esenţiale care caracterizează feţele. HOG funcţionează prin divizarea imaginii în celule mici şi calcularea orientării gradientului pixelilor din fiecare celulă. Aceşti gradienti sunt apoi utilizaţi pentru a crea un histogram de direcţii care descrie forma generală a feţei.

Pentru a identifica punctele esenţiale ale feţei, aplicaţia utilizează modelul **shape_predictor_68_face_landmarks.dat**, care **conţine date despre cele 68 de puncte de referinţă esenţiale pentru feţe, cum ar fi colţurile ochilor, nasul şi gura**. Modelul aplică tehnici de învăţare automată pentru a prezice coordonatele acestor puncte pe baza informaţiilor extrase de HOG.

3.2.2 Preprocesarea Imaginilor

Preprocesarea imaginilor este un pas crucial pentru a asigura o detecţie eficientă a trăsăturilor faciale. Aceasta implică următorii paşi:

1. **Conversia în Nuanţe de Gri:** Imaginile sunt convertite în nuanţe de gri pentru a reduce complexitatea datelor şi a elimina informaţiile de culoare care nu sunt necesare pentru detecţia feţei.
2. **Redimensionarea Imaginilor:** Imaginile sunt redimensionate la dimensiuni standardizate pentru a optimiza performanţa algoritmului, astfel încât să fie procesate mai rapid.

Preprocesarea este esenţială pentru:

- **Îmbunătăţirea Preciziei:** Eliminarea zgomotului şi reducerea dimensiunii imaginii ajută la creşterea preciziei algoritmilor de detecţie.
- **Reducerea Zgomotului:** Transformările aplicate ajută la izolarea caracteristicilor relevante ale feţei, facilitând identificarea acestora.

3.2.3 Interacţiunea cu Utilizatorul

Aplicaţia oferă o interfaţă prietenoasă pentru utilizatori, permiţându-le să interacţioneze prin intermediul a două butoane intuitive:

- **Încărcarea unei Imagini:** Utilizatorii pot selecta butonul dedicat pentru a încărca imagini din galeria personală, iar aplicaţia va analiza imaginea pentru a detecta trăsăturile faciale.
- **Activarea Camerei Video:** Alternativ, utilizatorii pot alege butonul pentru activarea camerei dispozitivului, permiţând capturarea imaginilor în timp real, iar aplicaţia va procesa şi analiza fluxul video.

3.2.4 Experienţa Utilizatorului: Feedback Vizual şi Interactiv

Aplicaţia oferă feedback vizual utilizatorului prin conturarea trăsăturilor faciale detectate. Aceasta include:

- **Afişarea Punctelor Esenţiale:** Odată ce trăsăturile sunt detectate, aplicaţia va evidenţia punctele esenţiale ale feţei, oferind un indiciu clar despre zonele analizate.
- **Sistem de Mesaje:** Utilizatorii pot primi mesaje de confirmare sau erori, în funcţie de succesul procesului de detecţie.

3.3 Scenarii de Utilizare

În **primul scenariu, utilizatorul poate încărca o fotografie**, iar **aplicaţia va analiza imaginea pentru a recunoaşte şi evidenţia trăsăturile faciale**. Procesul începe cu încărcarea fotografiei, urmată de preprocesarea acesteia pentru a o pregăti pentru analiză. Aplicaţia va utiliza algoritmi avansaţi pentru a localiza feţele din imagine şi pentru a marca trăsăturile faciale, cum ar fi ochii, nasul şi gura.

În **al doilea scenariu**, detectarea în timp real, **utilizatorul permite aplicaţiei să acceseze camera video** a dispozitivului său. Aceasta **va analiza fluxul video în timp real pentru a detecta trăsăturile faciale**. Aplicaţia va identifica feţele şi va evidenţia trăsăturile faciale în timp real, oferind o interacţiune dinamică. Acest scenariu este ideal pentru diverse aplicaţii, cum ar fi filtrele de pe platformele de socializare, jocurile care folosesc recunoaşterea facială sau chiar aplicaţiile de securitate care identifică utilizatorii.

Astfel, cele două scenarii oferă utilizatorilor experienţe diferite, adaptate la nevoile lor, fie că vor să analizeze o fotografie statică sau să interacţioneze în timp real cu tehnologia.

3.4 Limitări şi Posibile Îmbunătăţiri

Proiectul de recunoaştere a trăsăturilor faciale se confruntă cu mai multe provocări semnificative. Una dintre acestea este reprezentată de **condiţiile de iluminare şi unghiurile de vizualizare**. Algoritmii de detecţie sunt sensibili la variaţiile de iluminare; astfel, în condiţii de lumină slabă sau prea puternică, detaliile feţei pot fi distorsionate, afectând precizia recunoaşterii. De asemenea, unghiurile neobişnuite, cum ar fi feţele văzute din lateral sau din unghiuri atipice, pot îngreuna identificarea trăsăturilor faciale.

O altă limitare este legată de **modelul utilizat**, shape_predictor_68_face_landmarks.dat. Acest model are restricţii în capturarea diversităţii trăsăturilor faciale, iar acurateţea sa poate fi insuficientă în anumite situaţii.

b. Design

3.5 Arhitectura Sistemului

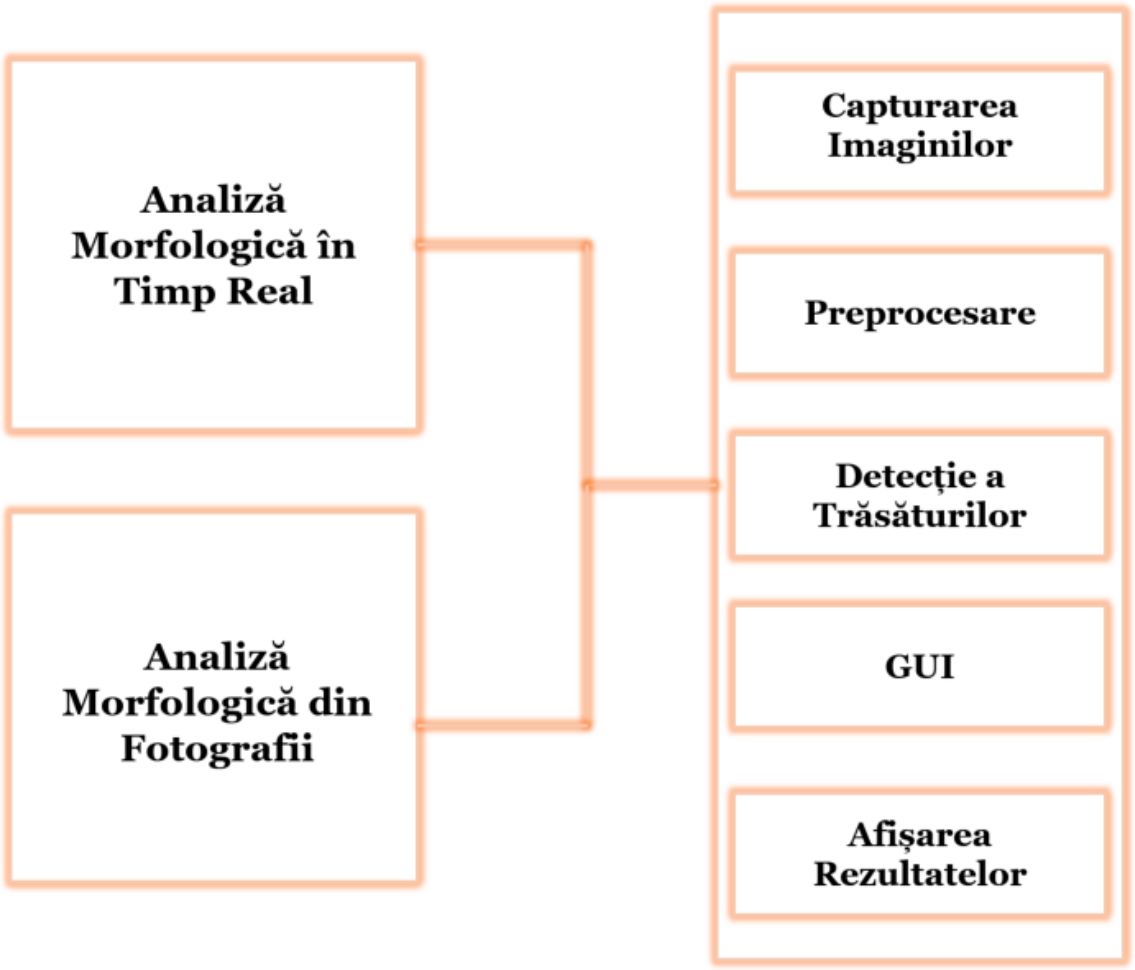
Arhitectura sistemului este organizată pe baza unui **model modular**, în care **fiecare componentă are responsabilităţi specifice**. Aceasta facilitează întreţinerea, scalabilitatea şi uşurinţa în utilizare.

Principalele componente ale sistemului includ:

1. **Modulul de Capturare a Imaginilor**:
    - Responsabil de obţinerea imaginilor de la utilizator (încărcare fişiere sau camera video).
2. **Modulul de Preprocesare**:
    - Aplică tehnici de filtrare şi transformare pentru a pregăti imaginea pentru detecţie.
3. **Modulul de Detecţie a Trăsăturilor**:
    - Utilizează algoritmi avansaţi pentru a detecta trăsăturile faciale.

4. **Interfaţa Grafică cu Utilizatorul (GUI)**:
    - Permite utilizatorului să interacţioneze cu aplicaţia, să încarce imagini şi să vizualizeze rezultatele.
5. **Modulul de Afişare a Rezultatelor**:
    - Afişează rezultatele detectării pe GUI, fie prin suprapunerea de marcaje pe imagini, fie prin evidenţierea trăsăturilor în feedul video.

Toate modulele colaborează într-o arhitectură bazată pe evenimente, unde fiecare componentă transmite datele la momentul adecvat, asigurând un flux de informaţii eficient şi fluid pentru detectarea facială, atât în timp real prin camera video, cât şi pe imagini statice, aşa cum este ilustrat în diagrama de mai jos.



3.6 Designul Modulului de Preprocesare

Modulul de Preprocesare utilizează o combinaţie de algoritmi şi tehnici pentru a transforma imaginea brută într-un format optim pentru detectarea trăsăturilor faciale. Procesul se desfăşoară în mai multe etape esenţiale:

1. **Preluarea Imaginilor**:
    - Imaginea este primită de la Modulul de Capturare, fiind pregătită pentru procesare.
2. **Reducerea Zgomotului**:
    - Se aplică un **filtru Gaussian**, care elimină detaliile inutile din imagine, îmbunătăţind contururile trăsăturilor faciale.
3. **Redimensionare**:
    - Imaginea este ajustată la dimensiuni standardizate, cum ar fi **640x480 pixeli**, pentru a asigura consistenţă şi a optimiza viteza de procesare.
4. **Conversia în Nuanţe de Gri**:
    - Dacă este necesar, imaginea este convertită în tonuri de gri pentru a simplifica analiza, ceea ce reduce complexitatea calculului.
5. **Normalizare**:
    - Valorile pixelilor sunt normalizate pentru a asigura o uniformitate de contrast, sprijinind astfel o detecţie mai precisă a trăsăturilor faciale.

Imaginea preprocesată este apoi pregătită pentru a fi transmisă Modulului de Detecţie a Trăsăturilor, asigurând o bază solidă pentru analiza detaliată.

3.7 Designul Algoritmului de Detectare a Trăsăturilor

Algoritmul de Detectare a Trăsăturilor are rolul de a identifica şi localiza trăsăturile faciale (ochi, nas, gură) în imaginea preprocesată. Procesul se desfăşoară în următorul flux:

1. **Primirea imaginii preprocesate**:
    - Imaginea prelucrată este trimisă de la Modulul de Preprocesare.
2. **Detectarea feţei**:
    - Se utilizează metoda **HOG (Histogram of Oriented Gradients)**. Această tehnică analizează gradientele de orientare din imagine pentru a

identifica regiunile ce conţin feţe. HOG este eficient în capturarea formei şi structurii feţei.

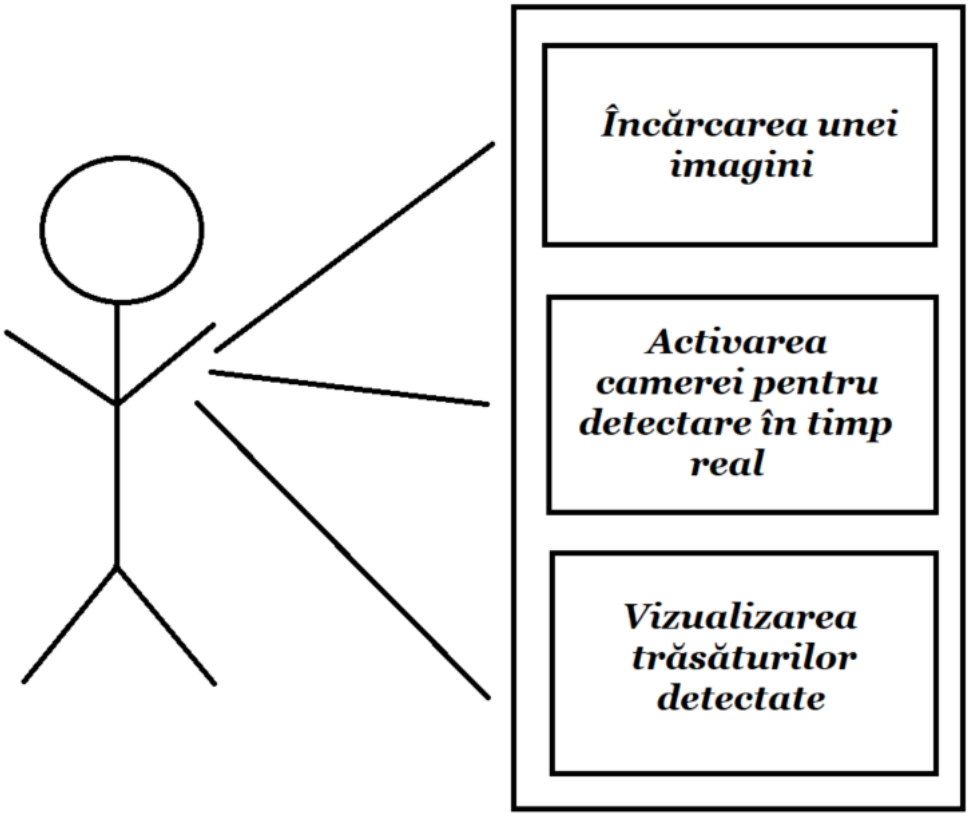
3. **Detectarea trăsăturilor**:
    - Se aplică **Shape Predictor**, cum ar fi modelul pre-antrenat de 68 de puncte de la Dlib. Acest model utilizează coordonatele detectate ale feţei pentru a identifica trăsăturile faciale specifice. Prin învăţarea dintr-un set mare de imagini etichetate, predictorul poate localiza precis punctele cheie ale feţei, cum ar fi colţurile ochilor, vârful nasului şi marginile buzelor.
4. **Marcarea trăsăturilor**:
    - Coordonatele detectate sunt utilizate pentru a crea un overlay pe imagine, evidenţiind trăsăturile identificate prin cercuri sau alte forme vizuale. Aceasta permite utilizatorului să vizualizeze clar trăsăturile faciale marcate.
5. **Iesire**:
    - Rezultatul, care include coordonatele trăsăturilor şi imaginea cu overlay, este trimis înapoi la interfaţa grafică (GUI) pentru a fi afişat utilizatorului.

3.8 Interfaţa cu Utilizatorul

Interfaţa cu utilizatorul este concepută să fie intuitivă şi prietenoasă. Aceasta include:

- **Pagini Principale**: O pagină principală cu două butoane mari, unul pentru simulare (analiză morfologică din fotografii) şi altul pentru detectare în timp real.
- **Afişarea rezultatelor**: Rezultatele detectării sunt afişate ca un overlay (suprapunere) pe imaginea originală, evidenţiind trăsăturile identificate.

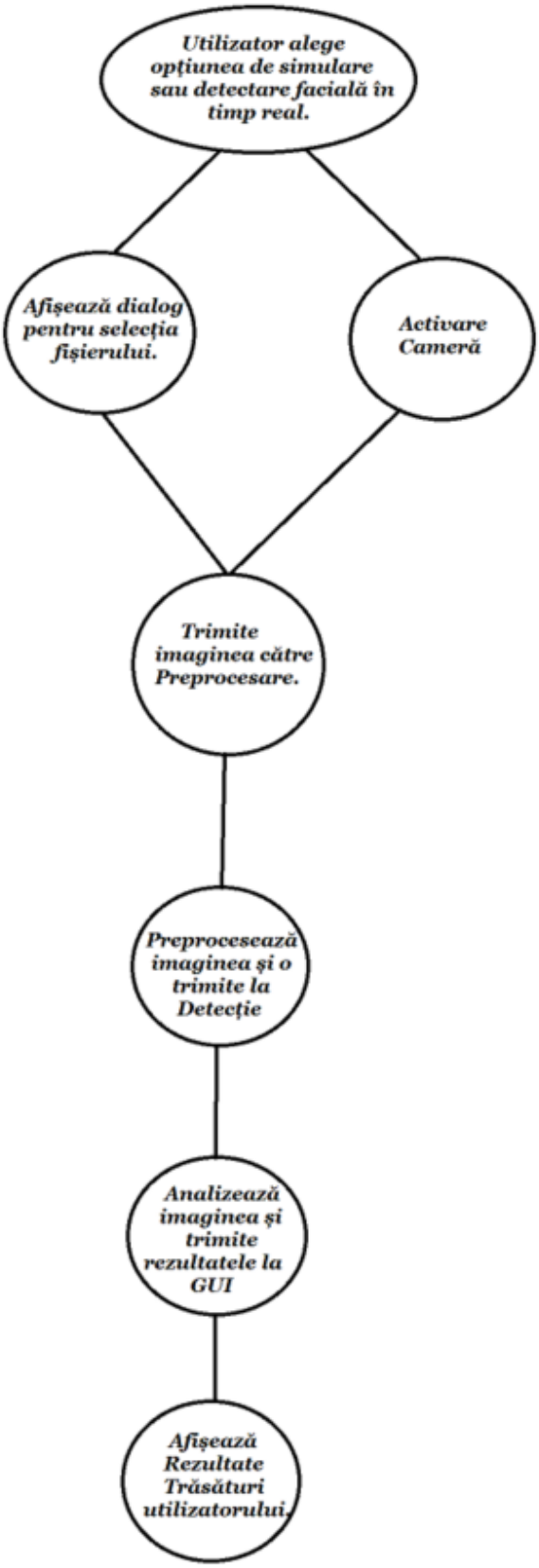
Descrierea diagramei use case pentru interfaţa cu utilizatorul include actorul principal, utilizatorul, care interacţionează cu aplicaţia. **Utilizatorul** are posibilitatea de a **încărca o imagine de pe dispozitiv**, de a **activa camera video** pentru detecţia în timp real a trăsăturilor faciale şi de a **vizualiza trăsăturile faciale evidenţiate pe imaginea procesată**. Săgeţile din diagramă ilustrează modul în care utilizatorul iniţiază aceste acţiuni, evidenţiind astfel fluxul de activitate al aplicaţiei.



3.9 Fluxul de Date

Fluxul de date începe cu **capturarea imaginii**, în care utilizatorul poate încărca o imagine sau activa camera video. Imaginea capturată este apoi trimisă la Modulul de Preprocesare, unde sunt **aplicate diverse filtre şi transformări** pentru a optimiza imaginea.

Odată preprocesată, imaginea este transmisă la Modulul de Detecţie, care **analizează imaginea** pentru a identifica feţele şi trăsăturile faciale. Coordonatele trăsăturilor detectate sunt generate şi trimise înapoi la interfaţa grafică, unde **rezultatele sunt afişate** pe ecran, cu marcaje pe imaginea originală sau pe feedul video.


Utilizator alege opțiunea de simulare sau detectare facială în timp real.
Afișează dialog pentru selecția fișierului.
Activare Cameră
Trimite imaginea către Preprocesare.
Preprocesează imaginea și o trimite la Detecție
Analizează imaginea și trimite rezultatele la GUI
Afișează Rezultate Trăsături utilizatorului.

Implementare

4.1 Iniţializarea Dependinţelor şi Configurarea Mediului

În această parte a codului, sunt importate şi configurate diverse biblioteci care vor fi utilizate pe parcursul întregii aplicaţii. Iată cum funcţionează fiecare componentă:

```
import cv2

import dlib

import tkinter as tk

from tkinter import filedialog, Toplevel

from PIL import Image, ImageTk
```

OpenCV(**cv2**) este o bibliotecă populară pentru procesarea imaginii şi viziune computerizată. Este folosită în acest cod pentru **manipularea şi analiza imaginilor, inclusiv pentru detectarea feţei, conversia imaginilor** în diverse formate şi redimensionarea acestora.

Dlib(**dlib**) este o altă bibliotecă de viziune computerizată care **include funcţionalităţi pentru detectarea feţelor şi pentru predicţia punctelor caracteristice ale feţei**. În acest cod, dlib este folosit pentru a detecta feţele dintr-o imagine şi pentru a localiza punctele caracteristice (ochi, nas, gură) pe feţele detectate.

Tkinter(**tkinter**) este biblioteca **de interfaţă grafică standard** în Python, care este folosită pentru a crea GUI-uri. Este folosită pentru a construi interfaţa aplicaţiei, inclusiv pentru interacţiunea cu utilizatorul şi butoanele pentru funcţionalităţi.

Filedialog şi Toplevel din Tkinter sunt module folosite pentru a crea interfete de dialog şi pentru a deschide altele noi**. Filedialog permite utilizatorului să selecteze fişiere de pe sistemul lor de operare**, iar **Toplevel este folosit pentru a crea interfete noi în aplicaţie**.

Pillow(**PIL**) este o bibliotecă care **permite lucrul cu imagini** în Python. În acest caz, este **utilizată pentru a manipula şi a afisa imagini în interfaţa Tkinter**. **ImageTk este folosit pentru a converti obiecte PIL.Image** într-un format care poate fi folosit în Tkinter.

4.2 Detectarea Componentelor Faciale

Detectarea componentelor faciale implică identificarea trăsăturilor feţei, cum ar fi ochii, gura şi nasul, utilizând tehnici de viziune computerizată. În aplicaţie, acest proces se realizează cu ajutorul bibliotecii **dlib** şi a unui model pre-antrenat, **shape_predictor_68_face_landmarks.dat**, care identifică 68 de puncte cheie pe faţă.

Codul conţine şi implementează mai multe tehnici de procesare a imaginilor, precum **HOG, Shape Predictor, Filtrul Gaussian, Conversia în Nuanţe de Gri, Redimensionarea şi Normalizarea**, pentru a optimiza detectarea şi recunoaşterea feţelor. Aceste tehnici sunt folosite în diferite moduri specific, astfel:

```
# Functia de detectare a punctelor caracteristice ale fetei din imagine
def detect_facial_features(image):  2 usages
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # """  Conversia în Nuante de Gri """

    gray = gray / 255.0 # """ Normalizare """

    gray = cv2.GaussianBlur(gray,  ksize: (5, 5),  sigmaX: 0) # """ Filtru Gaussian """

    faces = detector(gray) # """ HOG (Histogram of Oriented Gradients) """

    for face in faces:
        landmarks = predictor(gray, face)  # """ Shape Predictor """
        for n in range(36, 48):  # Ochi
            x = landmarks.part(n).x
            y = landmarks.part(n).y
            cv2.circle(image,  center: (x, y),  radius: 2,  color: (0, 255, 0), -1)
        for n in range(48, 68):  # Gură
            x = landmarks.part(n).x
            y = landmarks.part(n).y
            cv2.circle(image,  center: (x, y),  radius: 2,  color: (0, 255, 0), -1)
        for n in range(27, 36):  # Nas
            x = landmarks.part(n).x
            y = landmarks.part(n).y
            cv2.circle(image,  center: (x, y),  radius: 2,  color: (0, 255, 0), -1)

    return image
```

-> Am implementat **HOG** utilizând funcţiile din biblioteca dlib pentru detectarea feţelor. În codul nostru, HOG este apelat automat atunci când folosim funcţia detector(gray):

faces = detector(gray)

Acesta utilizează **HOG combinat cu un SVM (Support Vector Machine)** pentru a clasifica regiunile din imaginea în tonuri de gri ca fiind feţe sau nu.

-> Codul foloseşte **Shape Predictor** pentru a **identifica trăsăturile faciale** (precum ochii, nasul şi gura) şi pentru **a obţine punctele cheie ale feţei**. Acest lucru este realizat prin încărcarea unui model pre-antrenat şi utilizarea acestuia pentru a determina punctele de referinţă:

```
landmarks = predictor(gray, face)
```

-> **filtrul Gaussian** aplicat **reduce zgomotul şi netezeşte detaliile fine din imaginea în tonuri de gri**. Astfel, ajută algoritmul de detectare a feţei să identifice mai uşor contururile esenţiale şi să fie mai precis în recunoaşterea trăsăturilor faciale.

```
gray = cv2.GaussianBlur(gray, (5, 5), 0)
```

-> Codul conţine şi o **conversie a imaginii în nuanţe de gri** pentru a simplifica procesarea, eliminând informaţiile de culoare, care nu sunt necesare în detectarea feţei.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Prin aceasta**, imaginea este convertită din formatul BGR în format grayscale, reducând complexitatea datelor ce urmează a fi procesate**.

-> Am inclus o metodă de **redimensionare** a imaginii pentru a ajusta dimensiunea acesteia, astfel încât să fie mai uşor de procesat, mai ales în cazul aplicaţiilor cu performanţă limitată. (redimensionarea se realizează într-o funcţie care are rolul de a încărca şi afişa imaginea într-o fereastră nouă)

```
img = cv2.resize(img, new_size)
```

Dimensiunea imaginii este **ajustată conform raportului lăţime/înălţime şi a dimensiunii maxime stabilite (480px), facilitând afişarea şi procesarea imaginii**.

-> **Normalizarea** în acest caz **scalează valorile pixelilor în intervalul [0, 1], făcând imaginea mai uniformă** şi facilitând procesarea acesteia de către algoritmi, care lucrează mai eficient cu valori normalizate.

```
gray = gray / 255.0
```

4.3 Procesarea Imaginilor din Fişiere

În acest capitol, voi detalia procesul de încărcare şi prelucrare a imaginilor din fişiere folosind bibliotecile **OpenCV** şi **Tkinter**. Am implementat o aplicaţie care permite utilizatorilor să selecteze o imagine din fişiere, să aplice analiza trăsăturilor faciale şi să vizualizeze rezultatele într-o fereastră separată.

```
# Funcția pentru încărcarea și afișarea imaginii într-o fereastră nouă
def analyze_image():  1 usage
    file_path = filedialog.askopenfilename()
    if file_path:
        img = cv2.imread(file_path)

        # Procesăm imaginea
        h, w = img.shape[:2]
        max_size = 480
        if h > w:
            ratio = max_size / h
            new_size = (int(w * ratio), max_size)
        else:
            ratio = max_size / w
            new_size = (max_size, int(h * ratio))

        img = cv2.resize(img, new_size) # """ Redimensionarea """
        img = detect_facial_features(img)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Conversia la format compatibil cu Tkinter
        img = Image.fromarray(img)
        img = ImageTk.PhotoImage(img)

        # Creează o fereastră nouă pentru afișarea imaginii
        new_window = Toplevel(root)
        new_window.title("Imagine Procesată")
        new_window.geometry("600x500")
        new_window.configure(bg="#f0f4f7")

        # Afișează imaginea în noua fereastră
        panel = tk.Label(new_window, image=img, bg="#f0f4f7")
        panel.image = img  # Stocăm referința pentru a preveni ștergerea din memorie
        panel.pack(pady=10)

        # Buton de "Înapoi" pentru a închide fereastra secundară
        back_button = tk.Button(new_window, text="Înapoi", command=new_window.destroy, font=("Arial", 12), bg="#4a90e2", fg="white")
        back_button.pack(pady=10)
```

Am scris funcţia **analyze_image()** pentru a permite utilizatorilor să încarce imagini din fişiere şi să aplice procesarea pentru detectarea trăsăturilor faciale. Paşii esenţiali ai implementării sunt:

a). **Încărcarea imaginii**: folosesc un dialog de fişier pentru a permite utilizatorului să selecteze imaginea dorită. Funcţia **filedialog.askopenfilename()** este folosită pentru a **deschide un dialog în care utilizatorul poate alege fişierul de imagine**.

```
file_path = filedialog.askopenfilename()
if file_path:
    img = cv2.imread(file_path)
```

b). **Redimensionarea imaginii**: Am implementat redimensionarea imaginii astfel încât latura cea mai mare să fie limitată la 480px. Acest pas ajută la optimizarea procesării, în special pentru imagini mari.

```
# Procesăm imaginea
h, w = img.shape[:2]
max_size = 480
if h > w:
   ratio = max_size / h
   new_size = (int(w * ratio), max_size)
else:
   ratio = max_size / w
   new_size = (max_size, int(h * ratio))

img = cv2.resize(img, new_size) # """ Redimensionarea """
img = detect_facial_features(img)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

c). **Detectarea trăsăturilor faciale**: Aplicaţia **procesează imaginea** selectată pentru a detecta feţele şi trăsăturile faciale (ochi, nas, gură). Am utilizat funcţia **detect_facial_features()** pentru a marca aceste trăsături, care a fost deja detaliată în capitolul anterior.

```
img = detect_facial_features(img)
```

d). **Conversia şi afişarea**: După procesare, **imaginea este convertită într-un format compatibil cu Tkinter** şi afişată într-o fereastră nouă. Am folosit biblioteca **PIL** (Pillow) pentru a face conversia din formatul OpenCV în formatul necesar pentru Tkinter.

```
img = Image.fromarray(img)
img = ImageTk.PhotoImage(img)
new_window = Toplevel(root)
new_window.title("Imagine Procesată")
new_window.geometry("600x500")
new_window.configure(bg="#f0f4f7")

panel = tk.Label(new_window, image=img, bg="#f0f4f7")
panel.image = img

panel.pack(pady=10)
back_button = tk.Button(new_window, text="Înapoi",
command=new_window.destroy, font=("Arial", 12), bg="#4a90e2",
fg="white")
back_button.pack(pady=10)
```

4.4 Crearea Interfeței Grafice cu Tkinter

Am creat o interfață grafică simplă folosind Tkinter, care permite utilizatorilor să interacționeze cu aplicația pentru analiza trăsăturilor faciale.

Fereastra principală este configurată cu **dimensiuni fixe**, un fundal personalizat și un **icon pentru aplicație**. Butoanele din interfață sunt centrate pentru o utilizare ușoară și permit utilizatorilor să aleagă între analiza imaginii din fișiere sau analiza în timp real folosind webcam-ul.

După încărcarea și procesarea imaginii, rezultatele sunt afișate într-o fereastră secundară, oferind utilizatorilor posibilitatea de a vizualiza trăsăturile faciale detectate. Aplicația rulează continuu prin **root.mainloop()**, care permite interacțiunea utilizatorului cu interfața.

```
# Setează interfața grafică folosind Tkinter
root = tk.Tk()
root.title("Detectare Componente Faciale")

# Setează dimensiunea ferestrei
root.geometry("500x350")
root.minsize( width: 500,  height: 350)

# Setează icon-ul ferestrei (înlocuiește "icon.ico" cu numele fișierului tău de icon)
root.iconbitmap("face-recognition.ico")

# Setează culoarea de fundal
root.configure(bg="#f0f4f7")

# Creează un container pentru centrare
frame_center = tk.Frame(root, bg="#f0f4f7")
frame_center.pack(expand=True)

# Creează butoane pentru funcționalitățile aplicației, centralizate
btn_image_analysis = tk.Button(
    frame_center,
    text="Analiză Morfologică din Fotografii",
    command=analyze_image,
    font=("Arial", 12),
    width=30,
    height=2,
    bg="#4a90e2",
    fg="white",
    activebackground="#357ABD",
    activeforeground="white"
)
btn_image_analysis.pack(pady=10)

btn_real_time_analysis = tk.Button(
    frame_center,
    text="Analiză Morfologică în Timp Real",
    command=real_time_analysis,
    font=("Arial", 12),
    width=30,
    height=2,
    bg="#4a90e2",
    fg="white",
    activebackground="#357ABD",
    activeforeground="white"
)
btn_real_time_analysis.pack(pady=10)

# Rulează interfața grafică
root.mainloop()
```

Testare & Validare

5.1 Testare analiza morfologică din imagini statice

Testarea funcţionalităţii de analiză morfologică din imagini statice a fost realizată pentru a evalua performanţa detectării punctelor caracteristice faciale în fotografii preîncărcate.

- Metodologie:

a. Au fost utilizate imagini cu dimensiuni diferite, incluzând diverse fundaluri şi niveluri de iluminare.

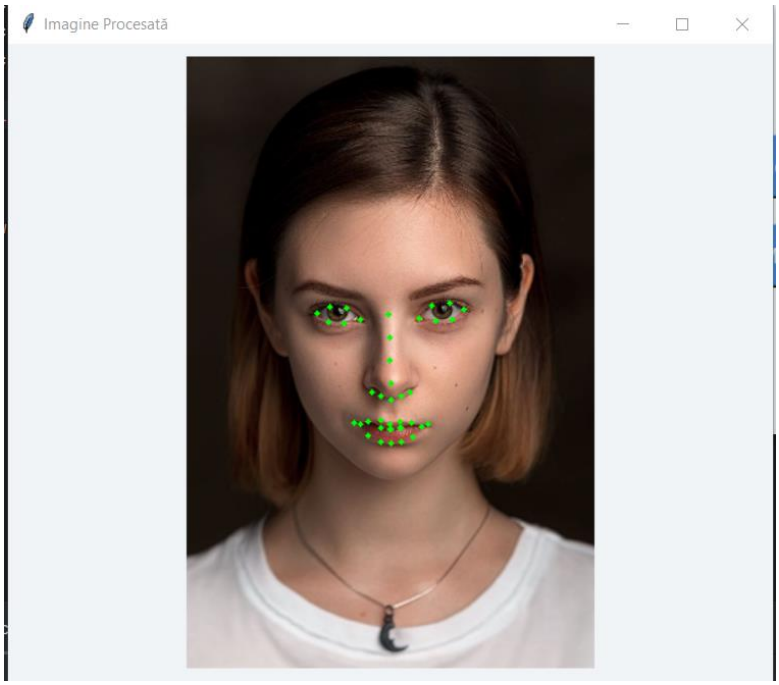
b. S-a verificat dacă punctele sunt detectate corect pentru ochi, nas şi gură.

c. Redimensionarea imaginilor la dimensiuni mai mici (e.g., 480 px pe latura mai mare) a fost implementată pentru a îmbunătăţi viteza de procesare.

- Rezultate:

a. În imagini bine luminate şi cu subiecţi poziţionaţi frontal, detectarea punctelor caracteristice a fost precisă.

b. Algoritmul a funcţionat corespunzător pentru feţele situate în centrul imaginii, dar a avut performanţe mai slabe pentru feţele aflate la margini.



5.2 Testare analiza morfologică în timp real

Testarea în timp real a analizat performanţa aplicaţiei în detectarea şi urmărirea punctelor caracteristice faciale utilizând camera web.

- Metodologie:

a. Scenariile de testare au inclus: faţă frontală, iluminare uniform, unghiuri diferite ale feţei, iluminare slabă sau surse de lumină din spatele subiectului.

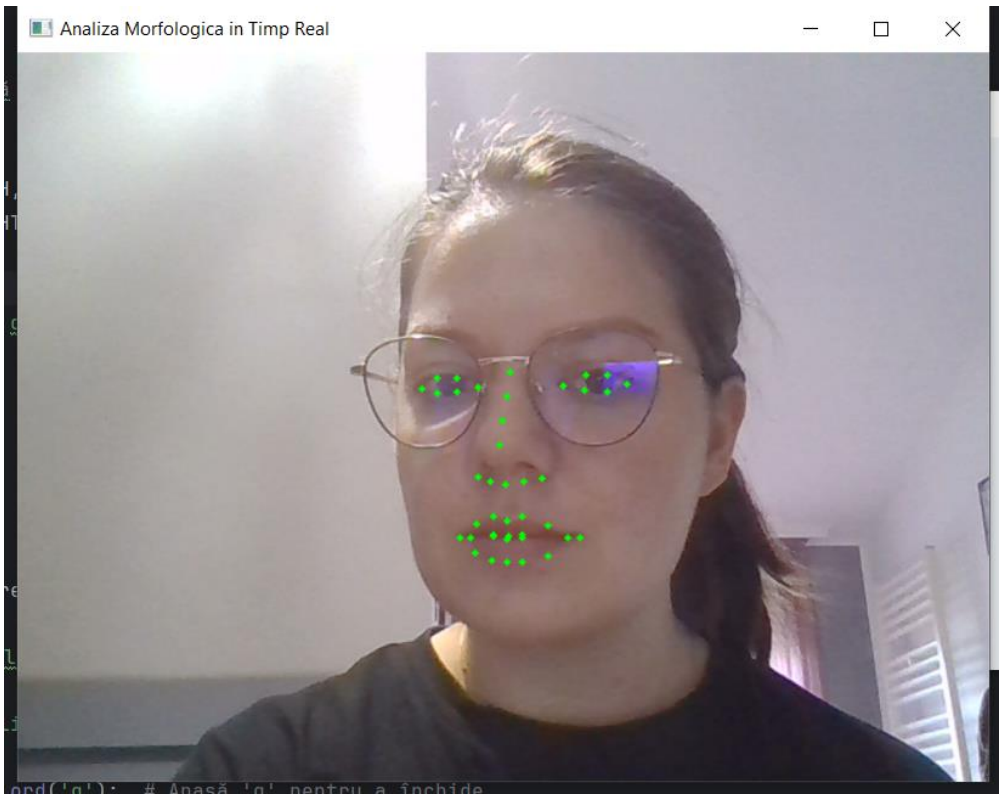
b. Detecţia a fost evaluată prin urmărirea punctelor faciale (ochi, gură, nas) în timp real.

- Rezultate:

a. În condiţii de iluminare bună, punctele au fost detectate precis şi urmărirea s-a realizat fără întreruperi.

b. În condiţii de iluminare scăzută sau în cazul mişcărilor rapide ale feţei, precizia detectării a scăzut (camera fiind blurata).

c. Rata cadrelor (FPS) a fost influenţată de complexitatea algoritmului şi de rezoluţia camerei.



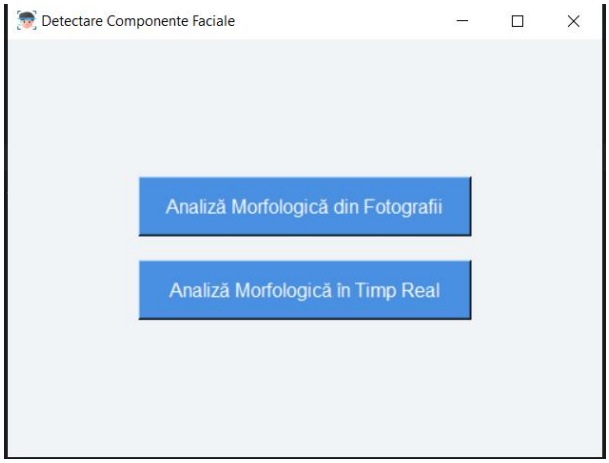
5.3 Testare funcţionalitatea interfeţei grafice

Testarea interfeţei grafice s-a concentrat pe verificarea funcţionării celor două butoane principale din aplicaţie, respectiv *Analiză Morfologică din Fotografii* şi *Analiză Morfologică în Timp Real*.

A fost testată afişarea corectă a ferestrei principale şi a celor două butoane, precum şi comportamentul acestora la interacţiune.

Scenarii de testare:

- Selectarea butonului pentru analiza imaginilor şi încărcarea unei imagini dintr-un dialog de fişiere.
- Selectarea butonului pentru analiza în timp real şi verificarea activării camerei.
- Închiderea ferestrelor nou create şi revenirea la meniul principal.



Rezultate:

- La apăsarea butonului *Analiză Morfologică din Fotografii*, aplicaţia a deschis corect dialogul pentru încărcarea imaginilor.
- La selectarea unei imagini valide, a fost creată o fereastră secundară pentru afişarea imaginii procesate.
- În cazul în care nu a fost selectată nicio imagine, aplicaţia a gestionat situaţia prin mesaje de eroare corespunzătoare.
- La apăsarea butonului *Analiză Morfologică în Timp Real*, camera s-a activat corect, iar punctele faciale au fost detectate în fluxul video.

5.4 Scenarii de Testare

5.4.1 **Detectare în condiţii ideale**

Imagine clară sau subiect în faţa camerei, bine iluminat, cu faţa poziţionată frontal.

Rezultate:

- Detecţia punctelor caracteristice a fost precisă şi stabilă.
- Timpul de procesare pentru imagini a fost redus (sub 1 secundă pentru imagini statice).

5.4.2 **Detectare în condiţii dificile**

Iluminare scăzută, unghiuri variate ale feţei (parţial întoarsă), mişcări rapide.

Rezultate:

- Precizia detectării a scăzut, cu erori frecvente în plasarea punctelor (mai ales în zona gurii şi a ochilor).
- În unele cadre video, detecţia a eşuat complet.

5.4.3 **Funcţionalitatea interfeţei**

Selectarea butoanelor din interfaţa principală:

- "Analiză Morfologică din Fotografii".
- "Analiză Morfologică în Timp Real".
- Gestionarea ferestrelor secundare create de fiecare funcţionalitate.

5.5 Probleme Identificate şi Soluţii

5.5.1 **Detectare imprecisă în condiţii de iluminare slabă**

Problemă: În imagini sau cadre video cu iluminare slabă, punctele caracteristice ale feţei nu sunt plasate corect, ceea ce afectează precizia analizei morfologice. Acest lucru este cauzat de dificultatea algoritmului de a identifica trăsăturile feţei în zone cu contraste reduse.

Soluţie: Pentru a îmbunătăţi rezultatele, utilizarea unui cadru mai luminos reprezintă o soluţie simplă şi eficientă. Asigurarea unui iluminat adecvat în spaţiul de captură facilitează algoritmului detectarea corectă a detaliilor faciale, eliminând astfel problemele cauzate de contrastul redus.

5.5.2 **Performanţă scăzută pe camere de rezoluţie joasă**

Problemă: La utilizarea camerelor cu rezoluţii mici, cum ar fi 320x240 px, detectarea punctelor caracteristice devine lentă şi inexactă. Acest lucru poate duce la o experienţă slabă pentru utilizator.

Soluţie: Configurarea rezoluţiei camerei la valori mai mari, cum ar fi 640x480 px sau mai sus, poate creşte calitatea şi precizia detectării. De asemenea, introducerea unei opţiuni de optimizare a algoritmului pentru camere mai slabe ar permite o performanţă acceptabilă chiar şi pe dispozitive cu resurse limitate.

5.5.3 **Lag în timpul detecţiei în timp real**

Problemă: Aplicaţia prezintă un lag vizibil în timpul detecţiei punctelor faciale în fluxul video în timp real, mai ales pe sisteme cu performanţe medii sau pe calculatoare mai vechi. Acest lag poate afecta experienţa utilizatorului, făcând aplicaţia mai puţin intuitivă şi eficientă.

Soluţie: Reducerea dimensiunii kernel-ului utilizat în filtrul Gaussian de la 5x5 la 3x3 poate accelera procesarea imaginii. În plus, optimizarea algoritmului de detecţie prin limitarea acestuia la punctele strict necesare (e.g., ochi, nas, gură) ar reduce timpul de calcul, îmbunătăţind performanţa în timp real.

Concluzii

Proiectul de detectare morfologică a componentelor faciale a demonstrat **eficienţa tehnologiilor de procesare a imaginii în identificarea trăsăturilor faciale esenţiale**, atât **în imagini statice**, cât şi în **timp real**. Prin utilizarea algoritmilor avansaţi şi a unei interfeţe intuitive, aplicaţia permite utilizatorilor să exploreze rapid şi uşor caracteristicile faciale.

Deşi testarea a evidenţiat provocări precum iluminarea slabă sau rezoluţiile joase ale camerei, soluţiile implementate – cum ar fi optimizarea algoritmilor şi ajustările de lumină – au îmbunătăţit performanţa. Proiectul oferă o **bază solidă pentru extinderi viitoare**, având potenţial pentru utilizări în domenii precum securitatea, medicina sau interacţiunile digitale.

În concluzie, **aplicaţia combină simplitatea utilizării cu rezultate precise**, fiind un exemplu practic al aplicării procesării imaginii în rezolvarea problemelor din viaţa reală.

Bibliografie

https://www.coursera.org/articles/what-is-facial-recognition

https://en.wikipedia.org/wiki/Facial_recognition_system

https://www.scienceabc.com/innovation/facial-recognition-works.html

https://builtin.com/articles/facial-recognition-technology-explained

http://dlib.net/

15 Efficient Face Recognition Algorithms And Techniques - RankRed

What is OpenCV Library? - GeeksforGeeks

How to Draw 5 Types of Architectural Diagrams | Lucidchart Blog

Histogram of Oriented Gradients explained using OpenCV

Computer Vision (Part 25)-Gaussian Filter | by Coursesteach | Medium

TkInter - Python Wiki

Pillow (PIL Fork) 11.0.0 documentation

dlib/dlib/image_processing at master · davisking/dlib

opencv/opencv: Open Source Computer Vision Library

Ajay-B-Kumar/Face-Recognition-GUI-app-python: A simple Tkinter GUI application that utilizes OpenCV and Dlib libraries to detect and recognize faces in real-time. The system is capable of performing multiple tasks, including face detection, face recognition, and identification of unknown faces.

Quantifying How Lighting and Focus Affect Face Recognition Performance | NIST

MDPI - Publisher of Open Access Journals