Avram Petru-Alin
Joandrea Sergiu-Cătălin

Explicarea implementării părții practice

1. Descărcarea bibliotecilor necesare.

```
import shutil
import tkinter
import tkinter as tk
from tkinter import messagebox

import customtkinter
import cv2
import torch
import torchvision.transforms as transforms
from PIL import Image
from customtkinter import *
from facenet_pytorch import MTCNN, InceptionResnetV1
```

- **shutil:** folosit pentru operații cu fișiere, inclusiv funcții pentru a copia, muta și șterge fișiere și directoare.
- **tkinter:** bibliotecă standard în Python pentru a implementa interfețe grafice.
- **messagebox** din **tkinter:** modul specific în Tkinter pentru crearea de casete de mesaje simple.
- **customtkinter:** o versiune actualizaă a bibliotecii tkinter.
- **cv2:**biblioteca OpenCV specializată în procesarea imaginilor și a videoclipurilor.
- **torch:** biblioteca PyTorch folosită pentru sarcini de învățare automată.
- **transforms** din **torchvision:** funcții de transformare a datelor din biblioteca torchvision a PyTorch. Sunt utilizate pentru preprocesarea imaginilor înainte de a le furniza modelelor de învățare automată.
- **Image** din **PIL**: folosit pentru manipularea și salvarea imaginilor.
- **MTCNN**, **InceptionResnetV1** din **facenet_pytorch**: MTCNN este o clasă pentru rețele convoluționale cu mai multe nivele, folosită pentru detectarea fețelor. InceptionResnetV1 este o clasă pentru un model de recunoaștere facială.

Avram Petru-Alin
Joandrea Sergiu-Cătălin

2. Main

```
if __name__ == "__main__":
    device = set_device()
    mtcnn, model = initialize_models(device)
    database_path = 'Images'
    database = load_database_images(database_path)
    database_features = extract_database_features(database, mtcnn, model, device)

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    root = CTk()
    app = FaceRecognitionApp(root, mtcnn, model, database_features, face_cascade, database_path)
    root.mainloop()
```

- **set_device():** configurează dispozitivul pentru modelul de învățare automată.
- **initialize_models(device):** inițializează MTCNN și modelul InceptionResnetV1.
- **load_database_images(database_path):** încarcă imagini din directorul specificat database_path.
- **extract_database_features(database, mtcnn, model, device):** extrage caracteristici din imaginile din baza de date folosind MTCNN și modelul InceptionResnetV1.
- **cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'):** inițializează un clasificator de cascadă Haar pentru detectarea fețelor.
- **CTk():** folosită pentru a crea o fereastră principală a interfeței grafice (GUI) într-o aplicație CustomTkinter.
- **FaceRecognitionApp(root, mtcnn, model, database_features, face_cascade, database_path):** inițializează o instanță a clasei FaceRecognitionApp, transmitând parametrii necesari.
- **root.mainloop():**pornește bucla de evenimente CustomTkinter, permițând GUI-ului să ruleze și să răspundă la interacțiunile utilizatorului.

Avram Petru-Alin
Joandrea Sergiu-Cătălin

3. Funcții auxiliare pentru clasa Aplicație

```
def recognize_face(input_features, database_features):
    recognized_person = ""
    min_distance = float('inf')
    for person, person_features in database_features.items():
        for feature in person_features:
            distance = ((input_features - feature) ** 2).sum()
            if distance < min_distance:
                min_distance = distance
                recognized_person = person
    return recognized_person if min_distance < 1.0 else None


1 usage
def display_recognition_result(frame, faces_cascade, recognized_person):
    for (x, y, w, h) in faces_cascade:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, f"{recognized_person}", (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)


2 usages
def display_unknown_result(frame, faces_cascade):
    for (x, y, w, h) in faces_cascade:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, "Necunoscut", (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
```

- **recognize_face(input_features, database_features):** primește caracteristicile feței detectate (input_features) și caracteristicile fețelor din baza de date (database_features). Compară caracteristicile feței detectate cu cele din baza de date și returnează numele persoanei recunoscute cu cea mai mică distanță Euclidiană, cu condiția ca distanța să fie mai mică decât 1.0. În caz contrar, returnează None.
- **display_recognition_result(frame, faces_cascade, recognized_person):** primește un cadru (frame), o listă de regiuni cu fețe (faces_cascade), și numele persoanei recunoscute (recognized_person). Adaugă un dreptunghi și text la fiecare regiune cu față recunoscută în cadrul pentru a indica numele persoanei recunoscute.
- **display_unknown_result(frame, faces_cascade):** Similar cu display_recognition_result, această funcție adaugă un dreptunghi și text pentru fiecare regiune cu față detectată, dar indică că persoana este necunoscută.

4. Clasa aplicației FaceRecognitionApp

```
def __init__(self, root, mtcnn, model, database_features, face_cascade, database_path):
    self.root = root
    self.root.title("Aplicatie pentru recunoastere faciala")
    self.height = root.winfo_screenheight()
    self.width = root.winfo_screenwidth()
    self.root.geometry("%dx%d" % (0.6 * self.width, 0.75 * self.height))
    self.root.configure(fg_color="#E3EFFC")
    self.mtcnn = mtcnn
    self.model = model
    self.database_features = database_features
    self.face_cascade = face_cascade
    self.database_path = database_path
    self.cap = cv2.VideoCapture(0)
    self.snapshot_index = 1
    if not self.cap.isOpened():
        messagebox.showerror("Error", "Nu s-a putut deschide camera. Verificați conexiunea și resursele hardware.")
        self.root.destroy()
        return
    self.frame_3 = CTkFrame(master=root, fg_color="#BFDFFF")
    self.frame_3.grid(row=1, column=1)
    CTkLabel(master=self.frame_3, text="Alege o optiune din stanga", font=("Arial Bold", 20),  justify="left").pack(
        expand=True, pady=(15, 15))

    self.frame_1 = CTkFrame(master=root, fg_color="#BFDFFF")
    self.frame_1.grid(row=0, column=0, rowspan=2, sticky="nsew", padx=15, pady=15)

    self.label1 = CTkLabel(master=self.frame_1, text="Alege:", font=("Arial Bold", 20), justify="left")
    self.label1.place(relx=0.5, rely=0.1, anchor=CENTER)

    self.radio_var = tkinter.StringVar(value="default")
    self.detP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                             command=self.radiobutton_event,
                                             text="Detectie Persoana", value="Detectie Persoana",
                                             fg_color="#091540")
    self.addP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                             command=self.radiobutton_event,
                                             text="Adaugare Persoana", value="Adaugare Persoana",
                                             fg_color="#091540")
    self.delP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                             command=self.radiobutton_event,
                                             text="Stergere Persoana", value="Stergere Persoana",
                                             fg_color="#091540")

    self.detP.place(relx=0.48, rely=0.3, anchor=CENTER)
    self.addP.place(relx=0.5, rely=0.4, anchor=CENTER)
```

- **__init__(self, root, mtcnn, model, database_features, face_cascade, database_path):** constructorul acestei clase inițializează o aplicație CustomTkinter pentru recunoașterea facială. Inițializează diferite elemente ale interfeței grafice, inclusiv cadre, etichete, butoane radio, și alte elemente. Definește metode pentru funcționalitățile aplicației, cum ar fi adăugarea sau ștergerea utilizatorului, și detectarea/recunoașterea fețelor în timp real.

```
1 usage
def detect_faces(self, frame):
    return self.mtcnn(frame)

1 usage
def detect_faces_cascade(self, gray_frame):
    return self.face_cascade.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
```

- **detect_faces(frame):** utilizează modelul MTCNN pentru a detecta fețele într-un cadru și returnează regiunile cu fețele detectate.
- **detect_faces_cascade(gray_frame):** folosește un clasificator Haarcascades pentru a detecta fețele într-un cadru de imagine alb-negru și returnează regiunile cu față detectate.

```
def update_video(self):
    ret, frame = self.cap.read()
    if ret:
        faces_cascade = self.detect_faces_cascade(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY))

        for (x, y, w, h) in faces_cascade:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        if self.recognition_enabled:
            faces = self.detect_faces(frame)
            if faces is not None and len(faces) > 0:
                img_resized = transforms.Resize((480, 640))(Image.fromarray(frame))
                input_features = extract_face_features(img_resized, self.mtcnn, self.model, set_device())

                if input_features is not None:
                    recognized_person = recognize_face(input_features, self.database_features)
                    if recognized_person is not None:
                        display_recognition_result(frame, faces_cascade, recognized_person)
                        self.hide_warning()
                    else:
                        display_unknown_result(frame, faces_cascade)
                        self.hide_warning()

            else:
                display_unknown_result(frame, faces_cascade)
                self.show_warning()
        else:
            self.hide_warning()

        self.display_frame(frame)

    self.root.after(10, self.update_video)
```

- **update_video():** actualizează continuu cadru cu cadru și afișează informații relevante, inclusiv recunoașterea facială.

Avram Petru-Alin
Joandrea Sergiu-Cătălin

```
def display_frame(self, frame):
    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    pil_image = Image.fromarray(rgb_image)
    tk_image = CTkImage(pil_image, size=(0.4 * self.width, 0.4 * self.height))
    self.video_label.img = tk_image
    self.video_label.configure(image=tk_image)
    self.video_label.update_idletasks()

1 usage
def detect_and_recognize(self):
    self.recognition_enabled = not self.recognition_enabled

1 usage
def show_warning(self):
    self.warning_label.pack()

3 usages
def hide_warning(self):
    self.warning_label.pack_forget()
```

- **display_frame(frame):** afișează un cadru în interfața grafică CustomTkinter.
- **detect_and_recognize():** activează sau dezactivează recunoașterea facială în timp real.
- **show_warning() și hide_warning():** afișează sau ascunde avertismente legate de nedetectarea feței în imagine.