**main.py**

import cv2

import numpy as np

import dlib

from imutils import face\_utils

import threading

import time

from playsound import playsound

import tkinter as tk

from PIL import Image, ImageTk

cap = cv2.VideoCapture(0)\*u7u

# Initialize the face detector and landmark detector

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor(r"C:\Users\pradeep\Desktop\Drowsiness Detection\shape\_predictor\_face\_landmarks.dat")

# Status marking for current state

sleep = 0

drowsy = 0

active = 0

status = ""

color = (0, 0, 0)

sleep\_start\_time = None

# Flag to control the detection loop

detection\_running = False

# Function to compute Euclidean distance between two points

def compute(ptA, ptB):

return np.linalg.norm(ptA - ptB)

# Function to detect if eyes are blinking

def blinked(a, b, c, d, e, f):

up = compute(b, d) + compute(c, e)

down = compute(a, f)

ratio = up / (2.0 \* down)

if ratio > 0.25:

return 2

elif ratio > 0.21 and ratio <= 0.25

return 1

else:

return 0

# Function to play the buzzer sound

def play\_buzzer():

playsound(r"C:\Users\pradeep\Desktop\Drowsiness Detection\buzzer.mp3")

# Function to update the video frame in the GUI

def update\_frame():

global detection\_running, status, color, sleep, drowsy, active, sleep\_start\_time

if detection\_running:

\_, frame = cap.read()

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = detector(gray)

for face in faces:

x1 = face.left()

y1 = face.top()

x2 = face.right()

y2 = face.bottom()

face\_frame = frame.copy()

cv2.rectangle(face\_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

landmarks = predictor(gray, face)

landmarks = face\_utils.shape\_to\_np(landmarks)

left\_blink = blinked(landmarks[36], landmarks[37],

landmarks[38], landmarks[41]landmarks[40],landmarks[39])

right\_blink = blinked(landmarks[42], landmarks[43],

landmarks[44], landmarks[47], landmarks[46],

if left\_blink == 0 or right\_blink == 0:

sleep += 1

drowsy = 0

active = 0

if sleep\_start\_time is None:

sleep\_start\_time = time.time()

if sleep > 6:

status = "SLEEPING !!!"

color = (255, 0, 0)

if time.time() - sleep\_start\_time >= 7:

threading.Thread(target=play\_buzzer).start()

elif left\_blink == 1 or right\_blink == 1:

sleep = 0

active = 0

drowsy += 1

sleep\_start\_time = None

if drowsy > 6:

status = "Drowsy! Warning!"

color = (0, 0, 255)

else:

drowsy = 0

sleep = 0

active += 1

sleep\_start\_time = None

if active > 6:

status = "Active :)"

color = (0, 255, 0)

# Convert the frame to a format tkinter can display

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(frame)

imgtk = ImageTk.PhotoImage(image)

video\_label.imgtk = imgtk

video\_label.configure(image=imgtk)

status\_label.config(text=status, fg=color\_to\_hex(color)

# Call this function again after 10 ms

root.after(10, update\_frame)

# Function to start the detection

def start\_detection():

global detection\_running

if not detection\_running:

detection\_running = True

# Function to stop the detection

def stop\_detection():

global detection\_running

detection\_running = False

# Function to exit the application

def exit\_application():

stop\_detection() # Ensure detection is stopped before exiting

root.destroy()

# Function to convert color tuple to hex

def color\_to\_hex(color):

return "#{:02x}{:02x}{:02x}".format(color[0], color[1], color[2])

# GUI Setup

root = tk.Tk()

root.title("Drowsiness Detection")

# Set the window to full screen

root.attributes('-fullscreen', True)

root.configure(bg="#1c1c1c")

# Title label

title\_label = tk.Label(root, text="Drowsiness Detection System", font=("Helvetica", 24, "bold"), bg="#1c1c1c",

fg="#ffffff")

title\_label.pack(pady=20)

# Frame to hold the video feed

video\_frame = tk.Frame(root, bg="#1c1c1c")

video\_frame.pack(pady=20)

# Label to display the video feed

video\_label = tk.Label(video\_frame)

video\_label.pack()

# Status label

status\_label = tk.Label(root, text="Status: Not started", font=("Helvetica", 20), bg="#1c1c1c", fg="#d3d3d3")

status\_label.pack(pady=20)

# Frame for buttons1

button\_frame = tk.Frame(root, bg="#1c1c1c")

button\_frame.pack(pady=20)

# Start button

start\_button = tk.Button(button\_frame, text="Start Detection", command=start\_detection, font=("Helvetica", 16),

bg="#28a745", fg="#ffffff", width=20)

start\_button.grid(row=0, column=0, padx=20)

stop\_button = tk.Button(button\_frame, text="Stop Detection", command=stop\_detection, font=("Helvetica", 16),

bg="#dc3545", fg="#ffffff", width=20)

stop\_button.grid(row=0, column=1, padx=20)

exit\_button = tk.Button(button\_frame, text="Exit", command=exit\_application, font=("Helvetica", 16), bg="#343a40",

fg="#ffffff", width=20)

exit\_button.grid(row=1, column=0, columnspan=2, pady=20)

# Start updating the frame

update\_frame()

root.mainloop()