**import** cv2 *#image and video processing***import** numpy **as** np *#numerical computations,multi-dimensional arrays and matrices***from** keras.models **import** load\_model *#load a pre-trained model from a file***from** keras **import** utils *# image preprocessing, and handling data formats***import** smtplib *# sending emails using the Simple Mail Transfer Protocol***import** ssl*#enabling secure communication over networks (Secure Sockets Layer)***import** pygame*#providing modules for handling graphics***import** threading*#creation, control #for timer***import** tkinter **as** tk*#standard GUI toolkit managing windows, dialogs, buttons, menus***from** tkinter **import** ttk  
**from** PIL **import** Image, ImageTk*#Python-friendly image processing capabilities and displaying images.***import** sys  
  
**class** EyeDetectionApp:  
 **def** \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title(**"Eye Detection Alarm"**)  
 self.video\_source = 0 *# Default camera* self.cap = cv2.VideoCapture(self.video\_source)  
 self.canvas = tk.Canvas(self.root, width=640, height=480)  
 self.create\_widgets()  
  
 *# Load the pre-trained face and eye detection models* self.face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + **'haarcascade\_frontalface\_default.xml'**)*#openCV, for face detection then access the path then xml file contain pre train data* self.eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + **'haarcascade\_eye.xml'**)  
 *# Load the trained eye state model* self.eye\_state\_model = load\_model(**'eye\_state\_model.h5'**)  
  
 self.closed\_eyes\_threshold = 3  
 self.sender\_email = **"rafikian2002@gmail.com"** self.app\_password = **"iifw putm vcwi bbfy"** self.recipient\_email = **"rafiakiranphotos@gmail.com"** self.font = cv2.FONT\_HERSHEY\_SIMPLEX  
 self.font\_scale = 1  
 self.font\_thickness = 2  
 self.font\_color = (255, 255, 0) *# White color in BGR* self.line\_type = cv2.LINE\_AA  
 self.debug = **False** self.timer\_thread = **None  
  
 def** create\_widgets(self):  
 label = ttk.Label(self.root, text=**"Eye Detection Alarm"**, font=(**'Helvetica'**, 16))  
 label.pack(pady=10)  
  
 start\_button = ttk.Button(self.root, text=**"Start Eye Detection"**, command=self.eye\_detection\_loop)  
 start\_button.pack(pady=10)  
  
 stop\_button = ttk.Button(self.root, text=**"Stop Eye Detection"**, command=self.stop\_detection)  
 stop\_button.pack(pady=10)  
  
 debug\_button = ttk.Button(self.root, text=**"Show/Hide Logs"**, command=self.shohHide)  
 debug\_button.pack(pady=10)  
  
 self.canvas.pack()  
  
 **def** trigger\_alarm(self):  
 print(**"Eyes closed for {} seconds. Triggering alarm."**.format(self.closed\_eyes\_threshold))  
 *# Play the alarm* pygame.mixer.init()  
 pygame.mixer.music.load(**"mixkit-classic-alarm-995.wav"**)  
 pygame.mixer.music.play()  
  
 *# Set the flag to indicate that the alarm is playing  
 # Send email notification* self.send\_email\_notification()  
  
 *# Function to send email notification* **def** send\_email\_notification(self):  
 subject = **"Alert: Eyes Closed"** body = **"Eyes closed for {} seconds. Triggering alarm."**.format(self.closed\_eyes\_threshold)  
  
 *# Create a secure SSL context* context = ssl.create\_default\_context()  
  
 *# Compose the email message* message = **f"Subject: {**subject**}\n\n{**body**}"  
  
 try**:  
 *# Connect to the email server and log in* **with** smtplib.SMTP\_SSL(**"smtp.gmail.com"**, 465, context=context) **as** server:  
 server.login(self.sender\_email, self.app\_password)  
  
 *# Send the email* server.sendmail(self.sender\_email, self.recipient\_email, message)  
  
 print(**"Email notification sent successfully!"**)  
  
 **except** Exception **as** e:  
 print(**f"An error occurred while sending email: {**str(e)**}"**)  
  
 **def** eye\_detection\_loop(self):  
 **while True**:  
 ret, frame = self.cap.read()*#capture vid then readd next frame  
  
 # Convert the frame to grayscale for face and eye detection* gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 *# Perform face detection* faces = self.face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)*# specifying how much the image size is reduced at each image scale  
 # how many neighboring rectangles need to confirm a face before it's accepted as a detection* **if** len(faces) < 1:  
 img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 *# img = gray* img = Image.fromarray(img)  
 img = ImageTk.PhotoImage(image=img)  
 self.canvas.img = img *# Keep a reference to avoid garbage collection* self.canvas.create\_image(0, 0, anchor=tk.NW, image=img)  
 self.root.update()  
 **continue** (x, y, w, h) = faces[0]  
 *# Iterate through detected faces  
 # Draw a rectangle around the detected face* **if** self.debug:  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 *# Extract the region of interest (ROI) for eye detection* roi\_gray = gray[y:y + h, x:x + w]  
 roi\_color = frame[y:y + h, x:x + w]  
  
 *# Perform eye detection within the face region* eyes = self.eye\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.05, minNeighbors=6)  
  
 *# Iterate through detected eyes* **for** (ex, ey, ew, eh) **in** eyes:  
 *# Draw a rectangle around the detected eye* **if** self.debug:  
 cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)  
  
 roi\_eye = roi\_color[ey:ey + eh, ex:ex + ew]*#specific area of a color image (roi\_color) where a detected eye is located  
  
 # Resize the ROI to (130, 130)  
 # resized\_roi\_eye = cv2.resize(roi\_eye, (130, 130))* resized\_roi\_eye = cv2.resize(roi\_eye, (130, 130), interpolation=cv2.INTER\_NEAREST)*# interpolation method to use when resizing the image* img\_array = utils.img\_to\_array(resized\_roi\_eye)  
 img\_array = np.expand\_dims(img\_array, axis=0)*#adds a new layer around the img\_array data* predictions = self.eye\_state\_model.predict(img\_array)  
  
 *# Interpret the model's predictions* **if** predictions[0][0] > predictions[0][1]:  
 **if** self.debug:  
 text = **"Close Eyes"** text\_size = cv2.getTextSize(text, self.font, self.font\_scale, self.font\_thickness)[0]  
 text\_position = ((roi\_color.shape[1] - text\_size[0]) // 2, (roi\_color.shape[0] + text\_size[1]) // 2)  
 cv2.putText(frame, text, text\_position, self.font, self.font\_scale, self.font\_color,  
 self.font\_thickness, self.line\_type)  
 **if** self.timer\_thread **and** self.timer\_thread.is\_alive():  
 print(**"thread already running"**)  
 **else**:  
 *# Start a new timer thread* self.timer\_thread = threading.Timer(self.closed\_eyes\_threshold, self.trigger\_alarm)  
 self.timer\_thread.start()  
  
 **else**:  
 **if** self.debug:  
 text = **"Open Eyes"** text\_size = cv2.getTextSize(text, self.font, self.font\_scale, self.font\_thickness)[0]  
 text\_position = ((roi\_color.shape[1] - text\_size[0]) // 2, (roi\_color.shape[0] + text\_size[1]) // 2)  
 **try**:  
 pygame.mixer.music.stop()  
 **except**:  
 print()  
 **if** self.debug:  
 cv2.putText(frame, text, text\_position, self.font, self.font\_scale, self.font\_color,  
 self.font\_thickness, self.line\_type)  
 **try**:  
 **if** self.timer\_thread **and** self.timer\_thread.is\_alive():  
 self.timer\_thread.cancel()  
 **except** Exception **as** e:  
 print(**"thread cancel exception"**)  
 *# print the raw prediction scores  
 # print("Raw Scores:", predictions)  
  
 # Display the resulting frame in the Tkinter window* img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 img = Image.fromarray(img)  
 img = ImageTk.PhotoImage(image=img)  
 self.canvas.img = img *# Keep a reference to avoid garbage collection* self.canvas.create\_image(0, 0, anchor=tk.NW, image=img)  
  
 self.root.update()  
  
 **def** stop\_detection(self) -> object:  
 *# if self.timer\_thread and self.timer\_thread.is\_alive():  
 # self.timer\_thread.join() # Wait for the thread to finish* self.cap.release()  
 **try**:  
 pygame.mixer.music.stop()  
 **except**:  
 print()  
 cv2.destroyAllWindows()  
 self.root.destroy()  
 sys.exit()  
  
  
 **def** shohHide(self) -> object:  
 *# if self.timer\_thread and self.timer\_thread.is\_alive():  
 # self.timer\_thread.join() # Wait for the thread to finish* self.debug = **not** self.debug  
  
  
**def** main():  
 root = tk.Tk()  
 app = EyeDetectionApp(root)  
 root.mainloop()  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()