Strain Analysis Based On Eye Blinking

By

Penmetsa Yuva Sai Varma

Aman Varma Rudraraju

Indukuri Harish varma

Mandapaka Venkata Bharath

1. Problem Statement

1.1 Overview:

In today's digital age, excessive screen-time and prolonged focus on electronic devices have become commonplace. However, this behavior often leads to a decrease in the frequency of eye blinking, which can result in eye strain and related discomfort. The lack of regular blinking while engaged in screen-based activities further exacerbates the issue, contributing to dry and irritated eyes, fatigue, and potential long-term vision problems. Additionally, individuals may not position their screens optimally or take necessary breaks, compounding the strain on their eyes. Therefore, there is a pressing need to address this problem and develop a solution that promotes healthy blinking habits and minimizes the risk of eye strain and associated ailments.

1.2 Purpose:

The purpose of this study is to develop an application that monitors eye blinking patterns to alert users and promote healthy blinking habits during prolonged screen usage. By utilizing a neural network model and the integrated webcam of a device, the application captures facial movements and analyzes the frequency of eye blinks. If the blink count deviates significantly from the average value, indicating potential eye strain, the application initiates an alert mechanism. This includes playing an audio message to notify the user and displaying a popup message on the screen to encourage them to take necessary breaks and rest their eyes. The primary objective is to raise awareness about the importance of regular blinking and prevent eye strain-related issues caused by excessive screen-time, ultimately promoting better eye health and reducing the risk of long-term vision problems.

2. Literature Survey

[1] Eye fatigue estimation is a crucial area of research, aiming to identify and address the adverse effects of prolonged visual tasks. One promising approach is the utilization of blink detection techniques based on Eye Aspect Ratio Mapping (EARM). By analyzing the changes in eye shape during blinks, EARM provides a reliable indicator for estimating eye fatigue levels. This literature survey explores existing studies that have employed EARM-based blink detection methods for eye fatigue estimation, highlighting their contributions and limitations. The findings of this survey serve as a foundation for further advancements in this field, offering potential solutions for accurate and non-intrusive eye fatigue assessment.

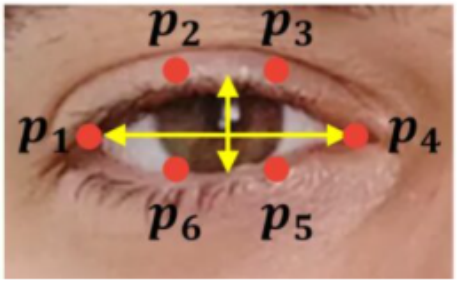
[2] In the paper titled "Eye-blink detection system for human-computer interaction" by Królak and Strumiłło (2012), the authors present a novel approach to eye-blink detection for enhancing human-computer interaction. The study focuses on the development of a system that accurately detects and analyzes eye blinks in real-time. By utilizing computer vision techniques, blink detection algorithms, and machine learning, the proposed system offers a non-intrusive and efficient method for interpreting blink patterns during human-computer interaction. The paper discusses the technical details of the system and highlights its potential applications in areas such as disability assistance, telecommunication, and user interface design. The results demonstrate the system's effectiveness in detecting eye blinks and its potential for improving user experiences in various interactive contexts.

[3] The paper titled "Eye Fatigue Prediction System Using Blink Detection Based on Eye Image" by Kuwahara et al. (2021) proposes an innovative approach to predict eye fatigue by analyzing blink patterns using eye images. The study focuses on developing a system that utilizes computer vision techniques to detect and analyze blinks from captured eye images in real-time. By extracting features such as blink duration and frequency, the system predicts the onset of eye fatigue. The paper highlights the potential of the proposed system in preventing eye strain-related issues and promoting visual health. The results demonstrate the system's accuracy in predicting eye fatigue, suggesting its applicability in various domains such as computer vision, healthcare, and human-computer interaction.

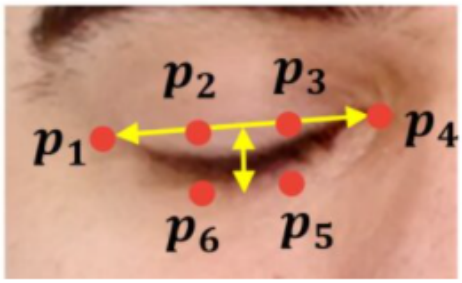
Proposed Solution

In their 2016 study, Real-Time Eye Blink Detection Using Facial Landmarks, Soukupová and ech introduced the eye aspect ratio (EAR), a statistic that the blink detector calculates. The eye aspect ratio is a beautiful algorithm that uses a fairly straightforward computation based on the ratio of distances between the eyes' facial landmarks.

Six (x, y)-coordinates are used to represent each eye, beginning at the left corner and moving clockwise around the remainder of the area:



Open eye will have more EAR



Closed eye will have less EAR

We can observe a relationship between the width and height of these coordinates based on this image. The eye aspect ratio (EAR), which is an equation that captures this relationship, can therefore be derived as follows:

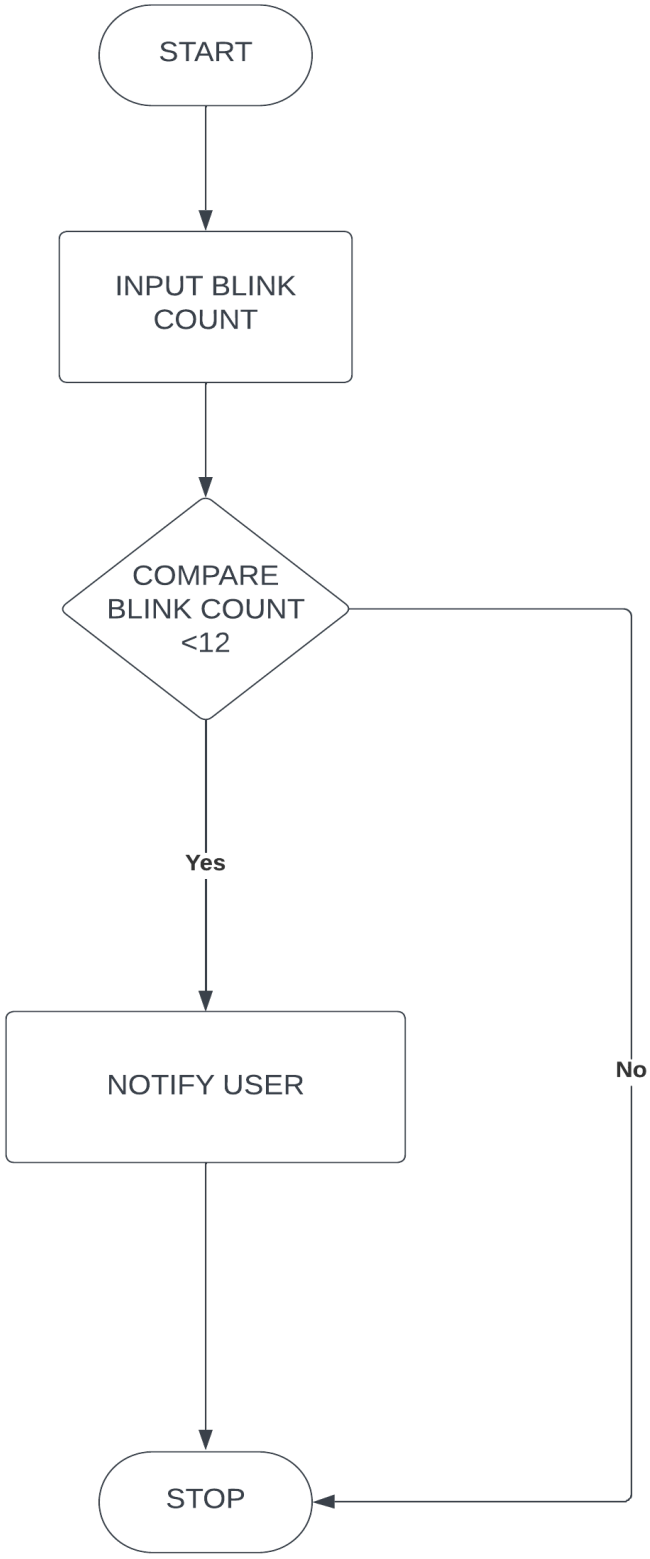
$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

While the eye is open, the aspect ratio is roughly constant, but when a blink occurs, it quickly decreases to zero. The ratio of eye landmark distances can be used instead of image processing methods to assess whether a person is blinking by using this straightforward equation. To make sure that the subject genuinely blink and that their eyes were not closed for an extended period of time, a frame threshold range is employed.

We used pre-existing Deep Learning models in my research to identify faces and facial landmarks from photos and video streams. These give back the coordinates of the face features that were utilised to calculate EAR, such as the left eye, right eye, nose, etc. Each minute, blinking rate is recorded.

3. Theoretical Analysis

3.1 Block Diagram



3.2 Hardware/Software requirements

Hardware Requirements

Mac-OS

WebCam access

Minimum disk space of 250mb

Software Requirements

Python

Anaconda/MiniConda

Jupyter Notebook

Flask

PyCharm

HTML

CSS

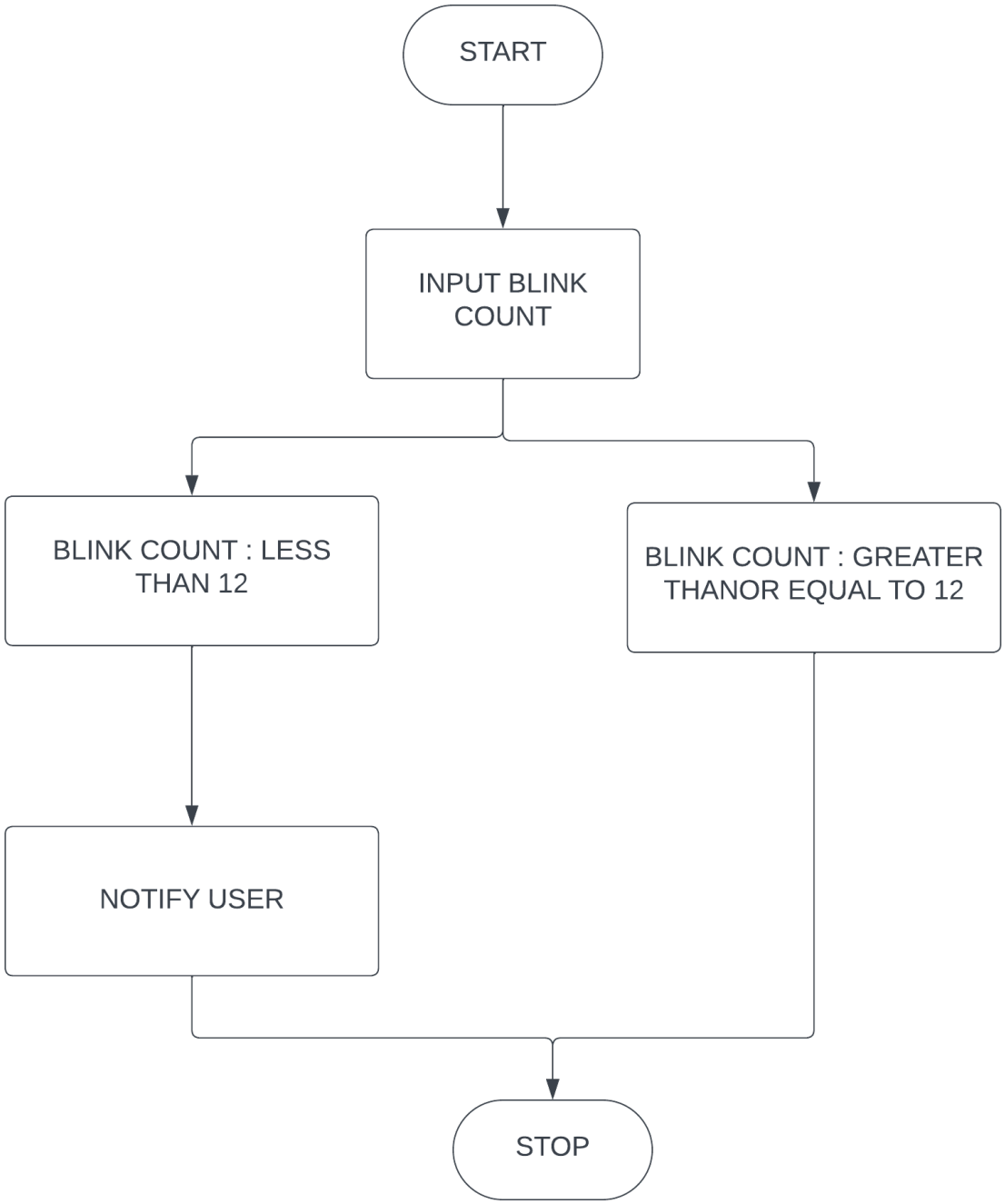
4. Experimental Investigation

Experimental Setup: The experimental setup involves capturing real-time video of the user's face using a webcam or camera. OpenCV is employed to extract the eye region from the video frames. Blink detection algorithms, such as eye aspect ratio (EAR), are applied to analyze the eye movements and detect blinks. The EAR is calculated by measuring the ratios of distances between key points on the eyes, enabling blink detection.

Strain Analysis: The blink data obtained from the video stream is analyzed to assess eye strain levels. Parameters such as blink rate, duration, and patterns are measured and monitored. By comparing the detected blink parameters against predefined thresholds or user-specific baselines, the system determines if the user is exhibiting signs of eye strain due to reduced blinking. If strain is detected, a real-time notification is triggered to remind the user to blink their eyes and take breaks as needed.

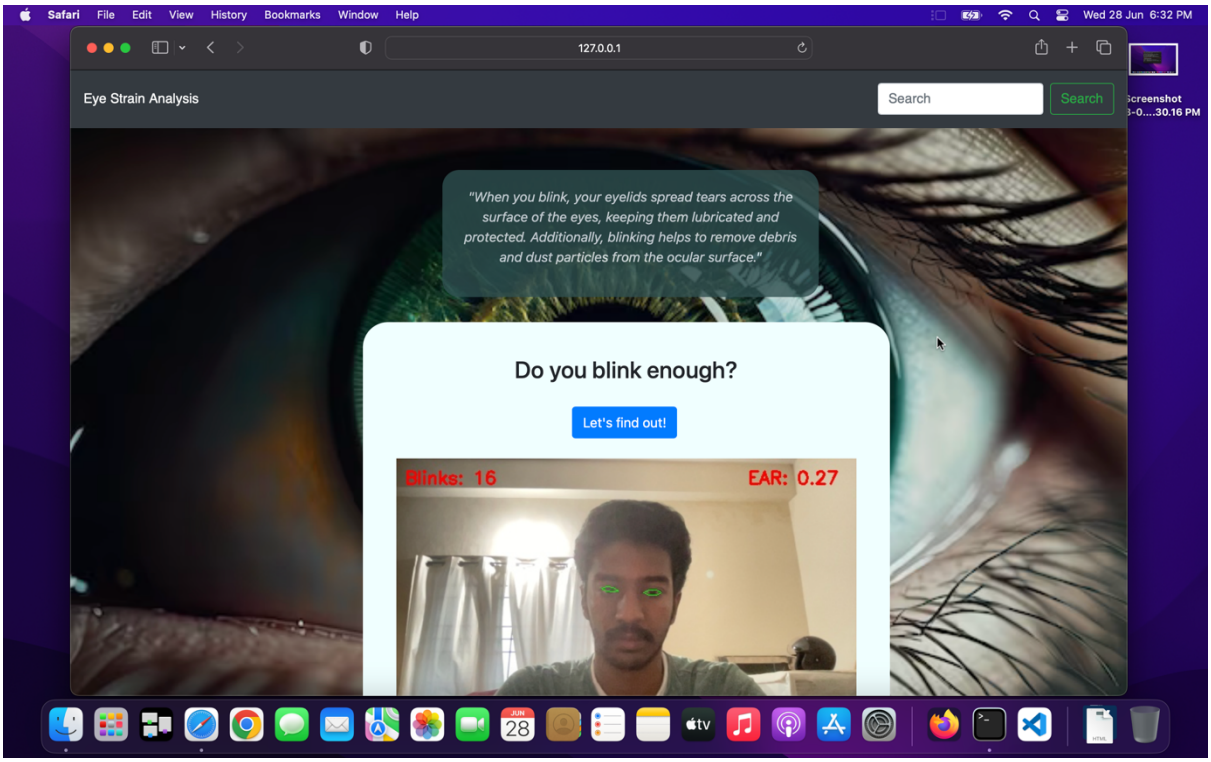
Eye Blink Notification: The system displays a notification on the screen to alert the user when their blink rate falls below the desired threshold. The notification is designed to be non-intrusive yet attention-grabbing, utilizing visual cues such as pop-up messages or overlays. Additionally, an optional audio alert can be played to further prompt the user to blink their eyes and relieve strain.

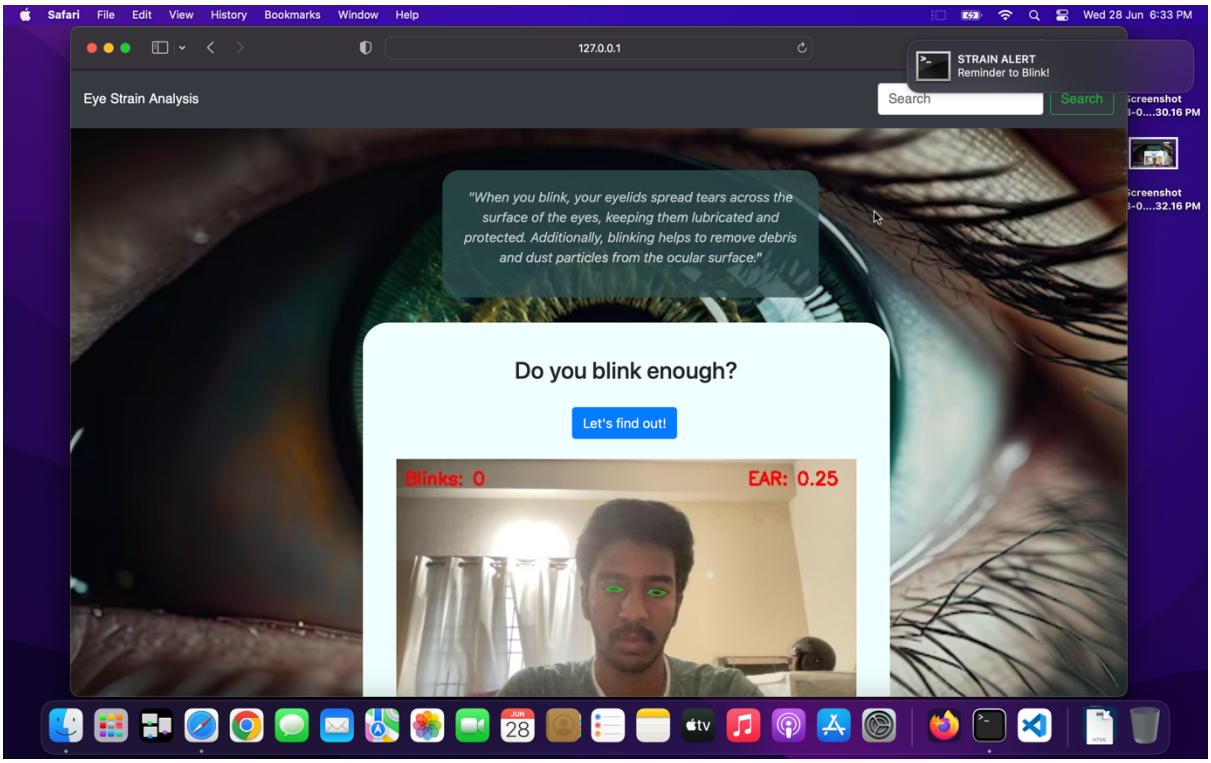
5. Flow Chart


START
INPUT BLINK COUNT
BLINK COUNT : LESS THAN 12
BLINK COUNT : GREATER THANOR EQUAL TO 12
NOTIFY USER
STOP

6. Result


Terminal Shell Edit View Window Help
Wed 28 Jun 6:30 PM
Eye-Strain-Detection — Python webstreaming.py — 80×24
Last login: Wed Jun 28 18:28:58 on console
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Yuvas-Air:~ yuva1808$ cd Downloads
(base) Yuvas-Air:Downloads yuva1808$ cd Externship
(base) Yuvas-Air:Externship yuva1808$ cd Eye-Strain-Detection
(base) Yuvas-Air:Eye-Strain-Detection yuva1808$ conda activate myenv
(myenv) Yuvas-Air:Eye-Strain-Detection yuva1808$ python3 webstreaming.py
* Serving Flask app 'webstreaming'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [28/Jun/2023 18:29:54] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/Jun/2023 18:29:54] "GET /video_feed HTTP/1.1" 200 -
127.0.0.1 - - [28/Jun/2023 18:29:54] "GET /static/images/eye.jpg HTTP/1.1" 304 -
127.0.0.1 - - [28/Jun/2023 18:29:54] "GET /favicon.ico HTTP/1.1" 404 -


Safari File Edit View History Bookmarks Window Help
Wed 28 Jun 6:32 PM
127.0.0.1
Eye Strain Analysis
Search
Search
"When you blink, your eyelids spread tears across the surface of the eyes, keeping them lubricated and protected. Additionally, blinking helps to remove debris and dust particles from the ocular surface."
Do you blink enough?
Let's find out!
Blinks: 16
EAR: 0.27


Safari File Edit View History Bookmarks Window Help
Wed 28 Jun 6:33 PM
127.0.0.1
STRAIN ALERT
Reminder to Blink!
Eye Strain Analysis
Search
Search
"When you blink, your eyelids spread tears across the surface of the eyes, keeping them lubricated and protected. Additionally, blinking helps to remove debris and dust particles from the ocular surface."
Do you blink enough?
Let's find out!
Blinks: 0
EAR: 0.25

7. Advantages and Disadvantages

7.1 Advantages

1) Real-time monitoring: Eye blinking occurs naturally and spontaneously, allowing for real-time monitoring of stain behaviour. This enables continuous analysis and tracking of changes in the stains over time, providing valuable insights into their properties and behaviour.
2) Potential for automation: Eye blinking detection can be combined with computer vision techniques and machine learning algorithms to automate the analysis process. This can improve efficiency, accuracy, and consistency in stain analysis, saving time and effort for researchers or analysts.
3) Cost-effective: Equipment like cameras or optical sensors, which are readily available and inexpensive, can be used to execute eye blinking detection. This makes it a more affordable method of stain examination, especially when compared to other intricate and pricey methods.

7.2 Disadvantages

1) Accuracy limitations: The ability of eye blinking detection to correctly identify and differentiate between various stains could be limited. The accuracy of stain analysis can be impacted by elements like lighting, occlusions, or differences in eye blinking patterns, which may result in false positives or false negatives.
2) Compatibility: This application is currently limited to mac OS , which can be a major disadvantage for windows users.
3) Subject variability: Individual differences in eye problems, ageing, and eye movement patterns can all affect how well eye blinking detection works for a given person. Due to this variation, it may be difficult to standardise the analysis and interpret the findings consistently across various subjects.

8. Applications

1) Computer Vision and Eye Tracking Research: The project can be used as a tool for collecting data on eye blinking patterns and behaviour. Researchers in the field of computer vision and eye tracking can utilize this data to develop and evaluate algorithms, models, and systems related to eye tracking, gaze analysis, or human-computer interaction.

2) Eye Strain and Vision Health: Prolonged periods of staring at digital screens, such as computers or smartphones, can lead to eye strain and discomfort. The project can serve as a reminder system to prompt users to blink their eyes regularly, which helps lubricate the eyes and reduce eye strain. It can be particularly beneficial for individuals who spend long hours working on screens.

3) Dry Eye Syndrome Management: Dry eye syndrome is a common condition characterized by insufficient tear production or poor tear quality, leading to dryness, irritation, and discomfort in the eyes. The project can assist individuals with dry eye syndrome by reminding them to blink more frequently, promoting tear production and relieving the symptoms associated with the condition.

4) Occupational Health and Safety: Certain professions require individuals to perform tasks that involve intense visual concentration, such as air traffic controllers, drivers, or surveillance operators. The project can contribute to occupational health and safety by reminding these professionals to blink regularly, reducing eye fatigue and improving overall visual comfort during their work.

5) Rehabilitation and Therapy: Some individuals may experience difficulties with eye blinking due to various conditions or after certain medical procedures. Your project can be utilized as a therapeutic tool to encourage and monitor eye blinking frequency, assisting in the rehabilitation process, and helping individuals regain normal blinking patterns.

6) Gaming and Virtual Reality (VR): In gaming and VR applications, where users are often engrossed in immersive virtual environments for extended periods, our project can remind gamers or VR users to blink regularly. This can help prevent dry eyes and minimize eye strain caused by prolonged visual focus.

7) Education and Training: Eye blinking detection and reminder systems can be incorporated into educational settings, particularly for children and students. By encouraging regular blinking, it can promote eye health, reduce visual fatigue, and improve concentration during study or classroom activities.

9. Conclusion

In conclusion, our project focuses on analysing strain and promoting eye blinking through detection and reminders, holds significant potential in various domains. By leveraging eye blinking patterns, it offers advantages such as non-invasiveness, cost-effectiveness, real-time monitoring, versatility, and automation potential. However, it also presents certain limitations, including accuracy challenges, subject variability, interpretation complexity and the influence of external factors.

Despite these limitations, our project's applications are diverse and impactful. It can contribute to computer vision and eye tracking research, aid in managing eye strain and promoting vision health, assist individuals with dry eye syndrome, enhance occupational health and safety, facilitate rehabilitation and therapy, benefit gaming and virtual reality experiences, and improve educational settings.

Overall, our project has the potential to positively impact eye health, enhance user experiences, and contribute to scientific research in the field of eye tracking and strain analysis. Continued development and improvement can lead to valuable applications and benefits across various domains, making it a valuable endeavour.

10. Future Scope

1) The user interface is still being improved.
2) Currently, due to various library limitations, this is only compatible with MacOS. It can have a potential cross-platform in future.
3) The visualisations will allow users to see insights about their blinking patterns.

4) We can further explore the potential of leveraging eye blinking data for health monitoring and analysis. By correlating eye blinking patterns with various health conditions or indicators, such as fatigue, stress levels, or cognitive load, it could provide valuable insights for healthcare professionals, researchers, or individuals in self-monitoring their well-being.

11. Bibliography

[1] Eye fatigue estimation using blink detection based on Eye Aspect Ratio Mapping(EARM)

https://doi.org/10.1016/j.cogr.2022.01.003

[2] Królak, A., Strumiłło, P. Eye-blink detection system for human–computer interaction. *Univ Access Inf Soc* **11**, 409–419 (2012).

https://doi.org/10.1007/s10209-011-0256-6

[3] A. Kuwahara, R. Hirakawa, H. Kawano, K. Nakashi and Y. Nakatoh, "Eye Fatigue Prediction System Using Blink Detection Based on Eye Image," *2021 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, 2021, pp. 1-3, doi: 10.1109/ICCE50685.2021.9427681.

12. Appendix

Source Code:

BlinkDetection_Module :

```
# Import all the necessary packages required
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import imutils
import time
import dlib
import cv2
import datetime
from time import sleep

def distance_ratio(eye):
        # Compute the euclidean distances b/w the two sets of vertical (x, y)-coordinates
        A = dist.euclidean(eye[1], eye[5])
        B = dist.euclidean(eye[2], eye[4])

        # Compute the euclidean distance b/w the horizontal (x, y)-coordinates
        C = dist.euclidean(eye[0], eye[3])

        # Now Compute Eye aspect ratio
        eye_ratio = (A + B) / (2.0 * C)

        return eye_ratio

# Here Initialize the dlib's face detector and Create the facial landmark predictor

face_detect = dlib.get_frontal_face_detector()
```

```
predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')
eye_ratio = 0

def calculate_eye_ratio(frame, gray):

        # Grabbing the indexes of the facial landmarks for the left and right eye

        (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
        (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

        points = face_detect(gray, 0)

        # Now loop over the face detections
        for r in points:
                # Determine the facial landmarks and convert the (x, y)-coordinates to a NumPy array
                shape = predictor(gray, r)
                shape = face_utils.shape_to_np(shape)

                l_eye = shape[lStart:lEnd]
                r_eye = shape[rStart:rEnd]
                l_eye_ratio = distance_ratio(l_eye)
                r_eye_ratio = distance_ratio(r_eye)

                # Calculate the average of eye aspect ratio for both eyes
                eye_ratio = (l_eye_ratio + r_eye_ratio) / 2.0

                # Finally Compute the convex hull for the left and right eye and visualize each of the
eyes
                l_eyeHull = cv2.convexHull(l_eye)
                rightEyeHull = cv2.convexHull(r_eye)
                cv2.drawContours(frame, [l_eyeHull], -1, (0, 255, 0), 1)
```

```
                cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

                return eye_ratio
```

Webstreaming.py :

```
import threading
import datetime
import imutils
import time
import cv2
from Blink_detection import blink_detection
import Notifier
from imutils.video import VideoStream
from flask import Response
from flask import Flask
from flask import render_template



app = Flask(__name__)

# Firstly Initialize the output frame and a lock is used to ensure thread-safe exchanges of the output frames
out_frame = None
lock = threading.Lock()

# Now keep track of the time
time_difference = datetime.datetime.now()

# The eye aspect ratio to indicate that the blink and range of consecutive frames the eye must be below the threshold value
EYE_AR_THRESH = 0.22
min_frames = 2
```

```
max_frames = 5

# After initializing frames , Start the video stream thread
vs = VideoStream(src=0).start()
fileStream = False
time.sleep(1.0)

@app.route("/")
def index():
	return render_template("blink.html")


def detect_blinks():

	global vs, out_frame, lock

	# In detect blinks function , Initialize the frame count and the total number of blinks
	count = 0
	blinks = 0

	# Now loop over frames from the video stream
	while True:

		if fileStream and not vs.more():
			break

		# Read frame from the threaded video file stream  the read function and then use resize and convert to grayscale
		frame = vs.read()
		frame = imutils.resize(frame, width=550)
		gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
            if(time_interval_passed()):
                if(blinks < 12):
                    Notifier.notify()
                blinks = 0

            #Calculating Eye Aspect Ratio using calculate ear function
            EAR = blink_detection.calculate_ear(frame, gray)

            if EAR is not None :

                # If the eye aspect ratio is below the blink threshold , increase the count
                if EAR < EYE_AR_THRESH:
                    count += 1

                # Now if the eye aspect ratio is not below the blink threshold
                else:
                    # if the eyes were closed for a sufficient range of frames then
increase the total number of blinks by one
                    if count >= min_frames and count <= max_frames:
                        blinks += 1

                    # Here we reset the eye frame count to zero
                    count = 0

                cv2.putText(frame, "Blinks: {}".format(blinks), (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
                cv2.putText(frame, "EAR: {:.2f}".format(EAR), (420, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

            # Acquire the lock
```

```
                with lock:
                        out_frame = frame.copy()


def time_interval_passed():

        global time_difference

        # Calculate difference of time interval since the last iteration
        time_x = datetime.datetime.now() - time_difference
        sec = time_x.total_seconds()

        # Check if one minute is passed
        if(sec >= 60):
                time_difference = datetime.datetime.now()
                return True


def generate():

        global out_frame, lock

        # Now loop over the frames from output stream
        while True:
                # Here, wait until the lock is acquired and check if the output frame is available,
otherwise skip the iteration
                with lock:
                        if out_frame is None:
                                continue

                        # Encode the frame into JPEG format
```

```
                (flag, encodedImage) = cv2.imencode(".jpg", out_frame)

                # Here ensure if the frame was successfully encoded
                if not flag:
                        continue

            # Derive the output frame in the byte format
            yield(b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
                bytearray(encodedImage) + b'\r\n')


@app.route("/video_feed")
def video_feed():
        # Return the response
        return Response(generate(),
                mimetype = "multipart/x-mixed-replace; boundary=frame")

# Here, check if this is the main thread of execution
if __name__ == '__main__':

        # Finally start a thread that will perform the blink detection
        t = threading.Thread(target=detect_blinks)
        t.daemon = True
        t.start()
        app.run(debug=True, threaded=True, use_reloader=False)


cv2.destroyAllWindows()
vs.stop()
```

Demo Video Link:

https://drive.google.com/file/d/1GDYlj4mCptPwWrq8bXPzXC75X4EDmP0n/view?usp=sharing