**FINAL YEAR PROJECT REPORT**

# POSTURE AND MOOD MONITORING USING DEEP LEARNING

**PROJECT REPORT**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF

**BACHELOR OF TECHNOLOGY**

SUBMITTED BY

MOHD SHAJAR – 20ELB027

MARIYA IFTEKHAR – 20ELB173

UNDER THE GUIDANCE OF

Dr. TAHIRA PARVEEN

DEPARTMENT OF ELECTRONICS ENGINEERING
ZAKIR HUSSAIN COLLEGE OF ENGINEERING AND TECHNOLOGY
ALIGARH MUSLIM UNVIERSITY
ALIGARH, INDIA – 202002
MAY 2024

POSTURE AND MOOD MONITORING USING DEEP LEARNING

## CANDIDATE DECLARATION

We hereby declare that our work submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in the Department of Electronics Engineering of the Aligarh Muslim University, titled as “Posture and Mood Monitoring Using Deep Learning” is a record of our work carried out during the VII Semester from August 2023 to December 2023 and VIII Semester from January 2024 to May 2024 under the guidance of Prof. Tahira Praveen, Professor, Department of Electronics Engineering, AMU.

The matter presented in this report has not been submitted by us for the award of any other degree of this or any other Institute/University.

We have also received a plagiarism report from MA Library and submitted it to our guide, where the similarity index is 13 percent.

Mohd Shajar- 20ELB027

Mariya Iftekhar- 20ELB173

This is to certify that the above statement made by the candidates is true to the best of my knowledge and belief.

Date:May 4,2024

Prof Tahira Praveen
Supervisor

## ACKNOWLEDGEMENT

We would like to start by expressing our sincerest gratitude to our Project supervisor Prof. Tahira Praveen, Professor, Department of Electronics Engineering at the Aligarh Muslim University, for his expertise, guidance, and enthusiastic involvement during our coursework.

We are highly obliged to faculty members of the Electronics Engineering Department because without their valuable insights and constructive opinions during evaluations, our project would not have yielded the significant results and led us to explore a myriad of use cases that we have put forward in this report.

We express our special thanks to our parents for their encouragement, constant moral Support, and to our friends and colleagues for being inquisitive and supportive during the course of this project.

Mohd Shajar- 2OELB027
Mariya Iftekhar-20ELB173

# ABSTRACT

The Posture and Mood Monitoring System is a novel application designed to provide real-time insights into an individual's emotional state and posture habits. Leveraging computer vision and machine learning techniques, the system analyzes live video feed from a camera to detect facial expressions and body postures. Through the integration of advanced algorithms, it accurately identifies emotions such as happiness, sadness, disgust, and neutrality, while also evaluating posture correctness.

The system comprises two main models: the Emotions Detection model and the Posture Tracking model. The Emotions Detection model utilizes a deep learning model (CNN) trained on a diverse facial expressions dataset to recognize facial emotions. Meanwhile, the Posture Tracking model employs “Pose Landmark Detection” (MediaPipe) to assess body alignments and angles, determining whether a user maintains a correct posture or not.

The Posture and Mood Monitoring System is employed as a Streamlit web app which has the following features: real-time feedback on mood and posture, personalized insights for individuals, and alerts for prolonged incorrect postures. By providing users with continuous feedback, the system aims to promote emotional awareness and encourage healthier posture habits, ultimately contributing to overall well-being and productivity.

In this report, we present the design, implementation, and evaluation of the Posture and Mood Monitoring System. We discuss the underlying technologies, the development process, and the system's performance in real-world scenarios. Additionally, we explore potential applications, future enhancements, and the impact of the system on user behavior and lifestyle. Overall, the Posture and Mood Monitoring System represents a promising solution for proactive health management and self-improvement.

This project’s contributions are two fold:

1. Our posture and mood monitoring system efficiently and effectively monitors the mood and posture of individual seated on a chair.

2. The Stream-lit app offers real-time insights into mood and posture, contributing to improved overall well-being when carefully and effectively analyzed.

Keywords

Convolution Neural network, Deep Learning, Facial Expression Recognition, Facial Feature Extraction, Expression Classification, Landmark Detection

# TABLE OF CONTENT

# LIST OF FIGURES

.

# OBJECTIVE

The primary objective of this project is to design and implement an integrated system capable of real-time detection and monitoring of both facial expressions and body postures. Specific goals include:

- Utilizing computer vision and deep learning techniques to capture real-time facial images and body landmarks.
- Extracting key features from facial images and body landmarks to analyze facial expressions and assess posture correctness.
- Employing machine learning models trained on labeled datasets to classify emotional states and identify correct/incorrect body postures using MediaPipe Pose landmark detection.
- Developing a user-friendly interface using Streamlit for seamless interaction with the integrated system.

# CHAPTER 1

# MOOD MONITORING

## 1.1 Introduction

In an era characterized by increasing sedentary lifestyles and growing concerns about mental health, the importance of proactive health monitoring and self-awareness cannot be overstated. The Posture and Mood Monitoring System emerges as a response to these contemporary challenges, offering a comprehensive solution for individuals seeking to enhance their emotional well-being and physical health.

This innovative project combines the power of computer vision, deep learning, and real-time data analysis to provide users with valuable insights into their mood and posture habits. By harnessing the capabilities of modern technology, the system aims to empower individuals to take proactive steps towards improving their overall health and lifestyle.

The project's genesis lies in the recognition of the interconnectedness between emotional states and physical posture. Research has shown that emotions can manifest in subtle facial expressions, while posture can influence mood and confidence levels. With this understanding, the Mood and Posture Monitoring System seeks to bridge the gap between these two domains, offering users a holistic perspective on their well-being.

Through the integration of advanced algorithms and intuitive user interfaces, the system offers real-time feedback and personalized insights tailored to individual users. By leveraging live video feed from a camera, the system can detect and analyze facial expressions to determine the user's emotional state accurately. Simultaneously, pose estimation algorithms track body alignments and angles, providing feedback on posture correctness.

The significance of the Posture and Mood Monitoring System extends beyond individual health monitoring. By promoting self-awareness and encouraging positive behavioral changes, the system has the potential to contribute to broader societal goals, such as reducing the prevalence of sedentary lifestyles and improving mental health outcomes.

In this report, we delve into the design, development, and evaluation of the Posture and Mood Monitoring System. We explore the underlying technologies, discuss implementation challenges and considerations, and present the system's performance in real-world settings. Additionally, we examine potential applications, future directions for research and development, and the broader implications of the system for health and well-being.

## 1.2. Libraries and Framework Used:

**Pandas:** Pandas is a robust data manipulation and analysis package. It provides data structures such as DataFrames and Series, which make structured data easier to handle, clean, and analyse.

**Numpy:** Numpy is a basic tool for scientific computing in Python. It supports massive, multidimensional arrays and matrices, as well as mathematical functions that can be applied to them effectively.

**Keras:** This library is based on TensorFlow and offers an intuitive interface for creating, training, and deploying deep learning models. It offers numerous functionalities such as layers, models, optimizers, and callbacks to create neural networks.

**MediaPipe:** Google created Mediapipe, an open-source platform that provides a comprehensive solution for developing real-time multimodal perceptual pipelines (e.g., hands, face, attitude). It includes pre-trained models, reusable components, and an adaptable architecture for effectively processing and analysing multimedia data such as photos, video streams, and audio. We used the Pose landmark detection model.

**cv2 (OpenCV):** OpenCV stands as a computer vision toolkit, furnishing resources for the analysis of images and videos. It encompasses a diverse array of capabilities, including image editing, identifying features, recognizing objects, and beyond.

**Seaborn:** seaborn is a data visualization library based on matplotlib. It offers a high-level interface for drawing attractive and informative statistical graphics, making it easier to create complex visualizations.

**Matplotlib:** Matplotlib is a plotting library that enables the creation of various plots and visualizations in Python. The pyplot module provides a MATLAB-like interface for creating static, interactive, and publication-quality plots.

**PIL:** Python Imaging Library is a powerful library for image processing tasks in Python. It provides a wide range of functionalities for opening, manipulating, and saving images in various formats. PIL allows users to perform operations such as resizing, cropping, rotating, filtering, and enhancing images with ease.

**Scikit-learn:** Scikit-learn, often abbreviated as sklearn, serves as a widely-used Python library for machine learning. It delivers straightforward and effective resources for mining and analyzing data, leveraging foundational scientific computing packages like NumPy, SciPy, and matplotlib. Offering an extensive selection of supervised and unsupervised learning techniques, including classification, regression, clustering, and dimensionality reduction, Scikit-learn also includes features for preprocessing data, evaluating models, and fine-tuning hyperparameters.

## 1.3. Emotion Detection Model:

Mood or Emotion detection typically involves three primary stages, as depicted in Figure 1: (i) Face Detection, (ii) Feature Extraction, and (iii) Emotion Classification.

**1.3.1. Face Detection (Pre-processing):** In the initial stage, the system detects the face within an image, isolating the facial components within that region. These components may encompass eyes, eyebrows, nose, and mouth. This pre-processing step focuses on identifying and localizing the face within the given image.

**1.3.2 Feature Extraction:** Following face detection, the system extracts informative features from different facial regions. This stage involves analyzing and extracting specific details or characteristics from various parts of the face, which are considered essential for identifying emotions.

**1.3.3. Emotion Classification:** In the final stage, a classifier is trained using labeled training data. Once trained, this classifier is utilized to assign labels corresponding to specific emotions based on the extracted features. This stage involves using machine learning or pattern recognition techniques to classify the emotions portrayed in the facial expressions captured in the input image.

These three stages form a sequential process, beginning with identifying the face, extracting significant facial features, and culminating in the classification of emotions based on the extracted features. The effectiveness of the overall Facial Emotion Recognition system relies on the accuracy and robustness of each stage in accurately capturing and interpreting facial expressions to recognize emotions.

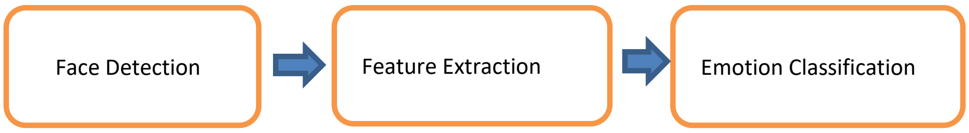


*Figure 1.1. Facial Emotion Classification Stages*

## 1.4. Data Collection and preprocessing:

### 1.4.1. Overview of the Dataset:

The dataset employed in this project was obtained from Kaggle. Specifically centered around facial emotion recognition, this dataset encompasses a compilation of images labeled with diverse emotions, including neutral, disgust, happy and sad as shown in Fig 1.

This dataset encapsulates a varied array of facial expressions, capturing a broad spectrum of emotions commonly encountered in human interactions. Each image within the dataset is meticulously labeled with its corresponding emotion, providing the foundation for supervised learning to train the emotion detection model.

The dataset encompasses distinct emotion categories, including expressions denoting neutrality, disgustness, happiness, and sadness. Additionally, depending on the dataset specifics, there may exist supplementary emotion categories such as surprise, fear, disgust, among others.

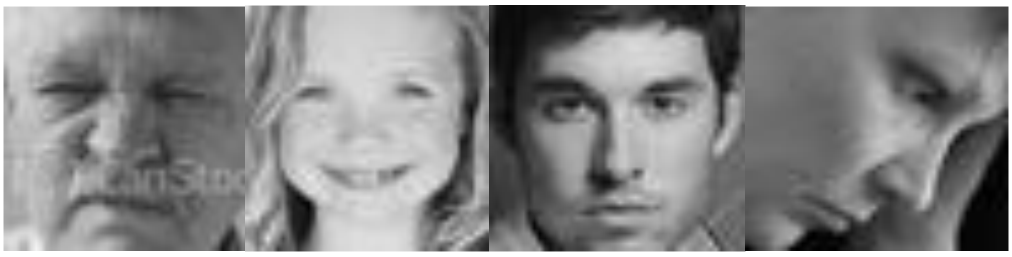


*Figure 1.2: Depicting Emotions Categories*

## 1.5. Methodology

Our model's design is based on Convolutional Neural Networks (CNNs). CNNs are a type of deep neural network that is specifically built for tasks involving visual data, making them very useful in image analysis, recognition, and classification.

As shown in Figure 2, CNNs are composed of three layers: convolutional layers, pooling layers, and fully connected (FC) layers. Figure 3 depicts the fundamental layer architecture of a CNN. It offers a powerful classification system for image identification and verification.

### 1.5.1. Data Preparation:

The emotion dataset used for training and validation was sourced from the Kaggle Emotion Dataset. This dataset comprises images categorized into different emotion classes, including happy, sad, neutral, and disgust. The training data is stored in the directory specified by TRAIN_DIR, while the validation data is stored in TEST_DIR.
To prepare the data for training, a custom function create data frame was implemented to extract image paths and corresponding labels from the specified directories. The extracted data was then organized into pandas Data Frames for ease of handling and manipulation.

### 1.5.2. Feature Extraction:

The feature extraction process involved converting the images into grayscale and reshaping them to the required dimensions for model input. The extract features function was created to iterate over the image paths extracted in the previous step, loading each image using the Python Imaging Library (PIL), converting it to grayscale, and storing the resulting pixel values as features.
The extracted features were then normalized by dividing by 255.0 to ensure consistency in scale across all features.

### 1.5.3. Label Encoding:

The labels associated with the emotion classes were encoded using the Label Encoder from scikit-learn. This step involved converting the categorical labels into numerical representations to facilitate model training. The fit transform method of the Label Encoder was applied to both the training and validation labels to ensure consistency in encoding.

## 1.6. Our CNN Model Architecture:

### 1.6.1 Input Layer:

The input layer, shaped as (48, 48, 1), signifies the entry point for facial image data into the neural network. In the context of grayscale images with a single channel, the specified shape corresponds to the dimensions of the input images.

### 1.6.2. Convolutional Modules (Modules 1, 2, 3):

The Convolutional Modules (Modules 1, 2, and 3) in the Sequential model are fundamental building blocks for feature extraction and hierarchical representation learning in convolutional neural networks (CNNs). Each module comprises a series of layers designed to extract increasingly complex features from the input data. Let's delve into the features of each module:

**Module 1:**

In the first module, a Conv2D layer with 32 filters is applied to the input image. This layer extracts fundamental features such as edges and textures. Batch normalization is then performed to stabilize and accelerate training, followed by the ReLU activation function to introduce non-linearity. Subsequently, max pooling with a 2x2 kernel size is applied to reduce spatial dimensions while retaining essential features. Finally, dropout with a rate of 30% is incorporated to prevent overfitting by randomly deactivating neurons during training.

**Module 2:**

Module 2 continues the feature extraction process with a Conv2D layer employing 64 filters. This layer captures more complex patterns from the input data. Similar to Module 1, batch normalization, ReLU activation, max pooling, and dropout are applied to maintain consistency and regularize the model.

**Module 3:**

The third module further enhances feature abstraction using a Conv2D layer with 128 filters. This layer aims to capture even more abstract and high-level features from the input images. Batch normalization, ReLU activation, max pooling, and dropout mechanisms are once again employed to ensure stable training and prevent overfitting, mirroring the approach taken in the previous modules.

### 1.6.3. Flatten Layers:

Following the convolutional modules, the model transitions to fully connected layers for classification:

Before feeding the extracted features into the dense layers, the Flatten layer reshapes the multi-dimensional output from the convolutional layers into a one-dimensional array, preparing it for input to the dense layers.

### 1.6.4. Dense Layers:

A dense layer with 128 units is added to introduce further non-linearity and abstraction to the extracted features. Batch normalization is applied to stabilize training by normalizing the activations of the previous layer. The ReLU activation function is then used to introduce non-linearity. To prevent overfitting, dropout with a rate of 50% is incorporated, randomly deactivating neurons during training.

### 1.6.5. Output Layers:

Finally, the output layer consists of a dense layer with 4 units, each corresponding to one of the emotion classes. The SoftMax activation function is applied to produce probabilities for each class, enabling the model to make predictions.

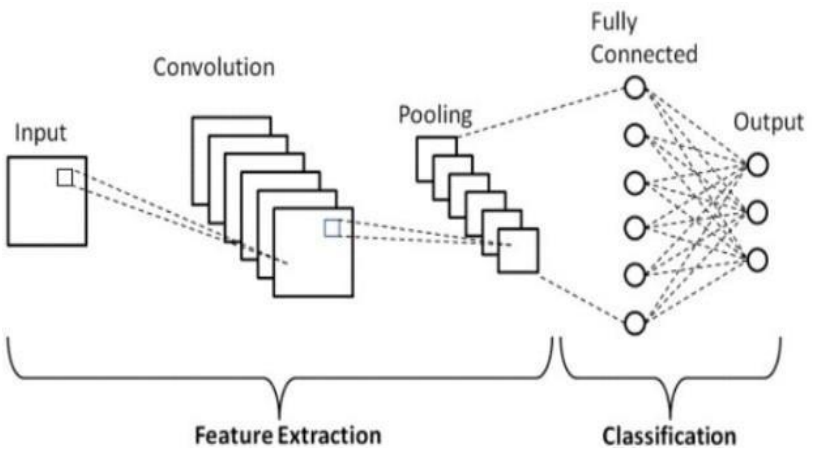


*Figure 1.3: Basic Architecture of Convolutional Neural Networks (CNN)*

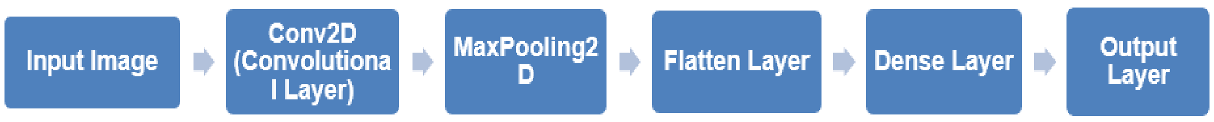


*Figure 1.4: Basic CNN Layers Architecture*

## 1.7. Compilation and Configuration:

The compilation and fitting step in training a deep learning model is a crucial phase where the model is prepared for training and the actual training process is executed.

During compilation, we configured the model's architecture and parameters for training. Following parameters are used:
Loss Function: categorical cross entropy is used for multi-class classification task to monitor the loss or cost of the model during training.

**Optimizer:** Adam optimizer is chosen for its adaptive learning rate and momentum properties, which often lead to faster convergence.

**Metrics:** Accuracy, a commonly used metric for classification tasks is used to monitor the model's performance.

In addition to compiling the model, data augmentation techniques are applied to increase the diversity of the training dataset and improve the model's generalization capability. Techniques such as rotation, horizontal and vertical shifts, shear, zoom, and horizontal flipping are applied to the training images using an ImageDataGenerator instance.

Callbacks are also utilized during training to customize the training process and adaptively adjust model parameters. EarlyStopping is employed to monitor the validation accuracy and halt training if no improvement is observed over a 12 number of epochs, helping prevent overfitting. ReduceLROnPlateau reduces the learning rate if validation loss has plateaued, allowing for finer adjustments and faster convergence.

Once the model is compiled and data augmentation techniques and callbacks are set up, the actual training process begins. The model's fit () method is called with the augmented data generator, specifying parameters such as the number of epochs, steps per epoch, and validation data. The model iterates over the training data for 200 epochs, adjusting its weights based on the calculated gradients and minimizing the loss function. The training progress and performance metrics are tracked and recorded in a history object, which is used for analysis and visualization after training is complete.

**Haar Classifier:**

We integrated the Haar Cascade Classifier into our project for its proven accuracy and efficiency in real-time face detection. The pre-trained Haar cascade file, specifically 'haarcascade_frontalface_default.xml,' served as our detector, allowing us to identify and localize faces within the input data.[7]

The Haar Cascade works by training on a large dataset of positive and negative images. Positive images contain the target object (faces), while negative images consist of the background. The classifier learns to discern the unique features of the target object, constructing a set of rules that define its characteristics.

## 1.8. Result

### 1.8.1. Accuracy Curve:

An accuracy curve, also called a training accuracy curve, is a visualization tool used in machine learning.
It depicts how well a model performs on a task, specifically focusing on the proportion of correct predictions the model makes, as training progresses.

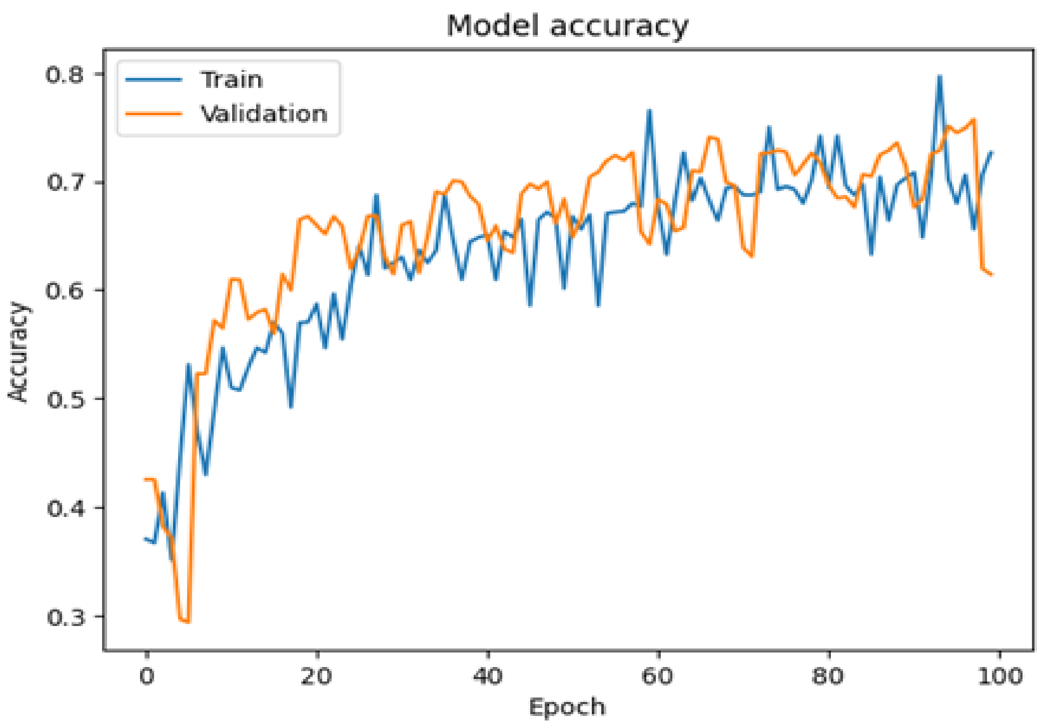


*Figure 1.5. Accuracy Curve of Mood Detection Model*

The accuracy curve exhibits an increasing trend for training accuracy, potentially reaching a plateau, while validation accuracy shows an initial rise followed by a plateau or even a slight decrease. Overall, the training and testing accuracies we achieved are 76% and 77 %.

## 1.9. Confusion Matrix:

A confusion matrix is a performance measuring tool used in machine learning to determine the accuracy of a classification model. It is a table that depicts the performance of a classification algorithm by contrasting actual and predicted classes.

It includes four key metrics: true positives (properly predicted positive cases), true negatives (correctly predicted negative instances), false positives (incorrectly forecasted as positive), and false negatives (incorrectly predicted as negative). The matrix aids in understanding the model's strengths and shortcomings, particularly when discriminating between classes and finding opportunities for improvement in the classification task.

Fig. 4 depicts the confusion matrix of the mood detection model:

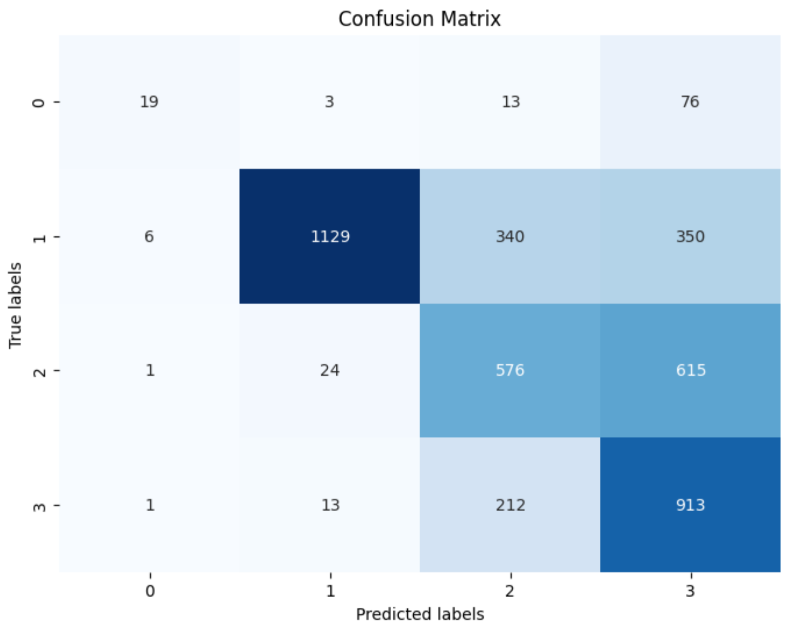


*Figure 1.6: Confusion Matrix of the Mood Detection Model*

- Class 0: The model correctly classified 19 out of 31 samples. There were 13 misclassifications.
- Class 1: The model performed well, correctly classifying 1129 out of 1175 samples. There were 46 misclassifications.
- Class 2: The model correctly classified 576 out of 803 samples. It misclassified 227 samples.
- Class 3: The model achieved high accuracy, correctly classifying 913 out of 950 samples. There were 37 misclassifications.

After extensive training and evaluation, the developed emotion detection model achieved a commendable accuracy of 77 percent. The model demonstrated effective learning and inference capabilities, showcasing its proficiency in predicting human emotions based on facial expressions as shown in fig 1.7.

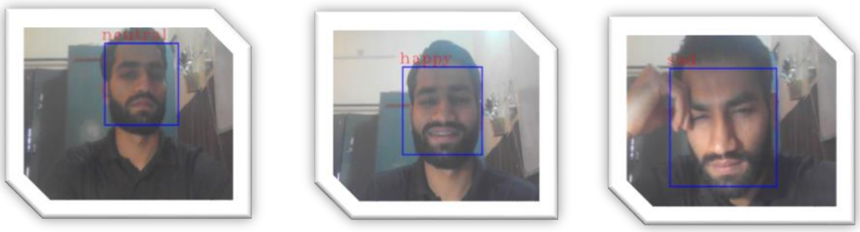


*Figure 1.7: Depicts the predicted emotions.*

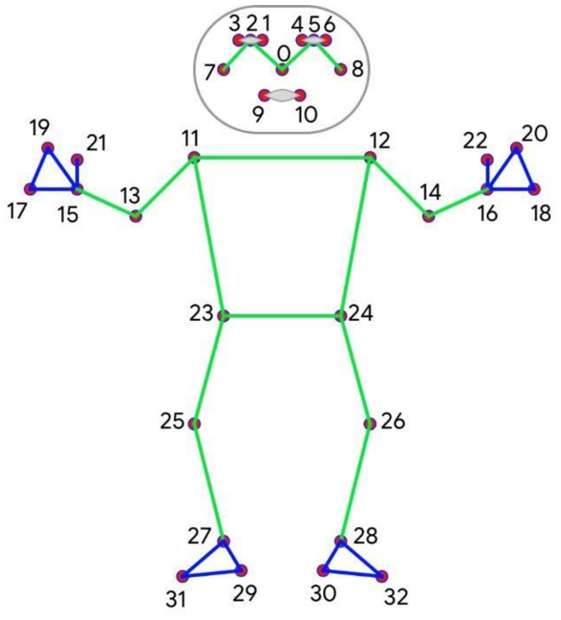
# CHAPTER 2

# POSTURE DETECTION AND MONITORING

## 2.1 Introduction

The “Posture Detection and Monitoring” model utilizes a pre-trained pose estimation model from the MediaPipe to detect and track sitting person posture in real-time.
MediaPipe, developed by Google, is an open-source framework renowned for its versatility in constructing real-time multimodal perceptual pipelines, encompassing features like hands, face, and pose detection. It provides pre-trained models, modular components, and a flexible architecture tailored for the efficient processing and analysis of various multimedia data formats, including images, video streams, and audio. In our project, we leveraged its Pose landmark detection model.
The pose estimation model adeptly identifies 33 distinct points across the human body, as illustrated in Fig. 2.

We have created a module for posture detection using the pre-trained pose estimation model. Within this posture module, detector and tracker functions are defined. The detector initially identifies regions of interest within the image, while the tracker then proceeds to pinpoint the landmarks of interest associated with sitting person.



0. nose
1. right eye inner
2. right eye
3. right eye outer
4. left eye inner
5. left eye
6. left eye outer
7. right ear
8. left ear
9. mouth right
10. mouth left
11. right shoulder
12. left shoulder
13. right elbow
14. left elbow
15. right wrist
16. left wrist
17. right pinky knuckle #1
18. left pinky knuckle #1
19. right index knuclke #1
20. left index knuckle #1
21. right thumb knuckle #2
22. left thumb knuckle #2
23. right hip
24. left hip
25. right knee
26. left knee
27. right ankle
28. left ankle
29. right heel
30. left heel
31. right foot index
32. left foot index

*Figure 2.1: Body posture detection using MediaPipe*

## 2.2. Implementation:

### 2.2.1. pose Estimation Model:

The posture monitoring system relies on a pre-trained pose estimation model provided by MediaPipe. This model is trained on a large dataset of human body images and is capable of accurately detecting key landmarks on the human body, such as joints and body parts.

### 2.2.2. Key Landmarks Detection:

Using the pose estimation model, the system identifies and localizes key landmarks on the human body, including:
Shoulders, Elbows, Nose, Eyes, etc. These landmarks serve as reference points for analyzing the user's posture.

### 2.2.3. Real-time Monitoring:

The model operates in real-time, continuously analyzing video frames captured from a webcam or video stream.
As the user moves or changes their posture, the model dynamically tracks the movement of these key landmarks across consecutive frames.

### 2.2.4. Angle Calculation and Posture Assessment:

By analyzing the spatial relationships between key landmarks, the system calculates various angles associated with different body postures.
These angles are compared against predefined thresholds to determine whether the observed posture is correct or incorrect.

### 2.2.5. Feedback and Visualization:

The system provides real-time feedback to the user, usually through a graphical user interface (GUI) overlay on the video stream.
highlight the user's posture status and any deviations from the desired posture.

### 2.2.6. Posture Time Tracking:

The system keeps track of the duration for which the user maintains correct and incorrect postures.
It records the total time spent in each posture state, allowing users to monitor their posture habits over time.

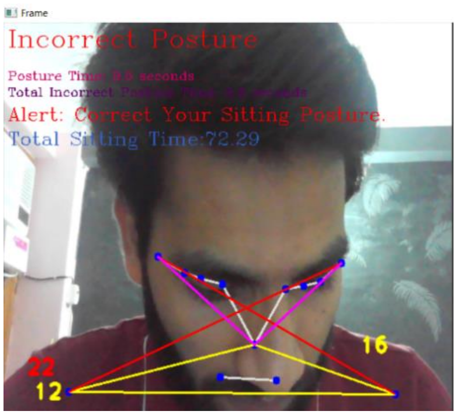
### 2.2.7. Alerts and Reminders:

In addition to real-time feedback, the system may include alerts or reminders to prompt users

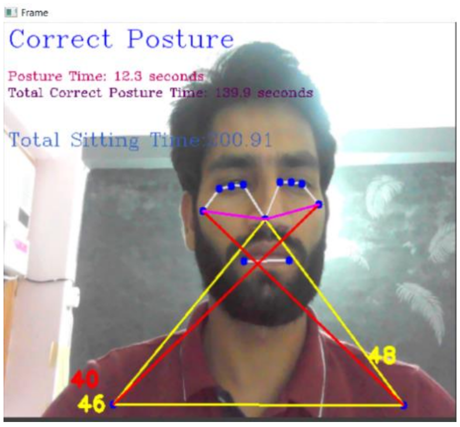
to correct their posture or take breaks at regular intervals.
These alerts can be customizable based on user preferences and requirements.

### 2.2.8. Data Logging and Analysis:

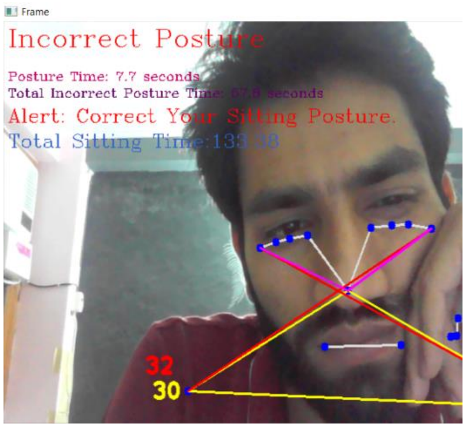
The system may log posture data, including angles, posture status, and duration, for further analysis.
This data can be used to identify trends, patterns, and areas for improvement in the user's posture habits over time.



Incorrect Posture
Posture Time: [illegible] seconds
Total Incorrect [illegible] seconds
Alert: Correct Your Sitting Posture.
Total Sitting Time:72.29



Correct Posture
Posture Time: 12.3 seconds
Total Correct Posture Time: 139.9 seconds
Total Sitting Time:200.91



Incorrect Posture
Posture Time: 7.7 seconds
Total Incorrect Posture Time: [illegible] seconds
Alert: Correct Your Sitting Posture.
Total Sitting Time:133.38

*Figure 2.2: Sitting Posture Monitoring*

# CHAPTER 3

# USER INTERFACE / STREAMLIT WEB APPLICATION

## 3.1. Introduction

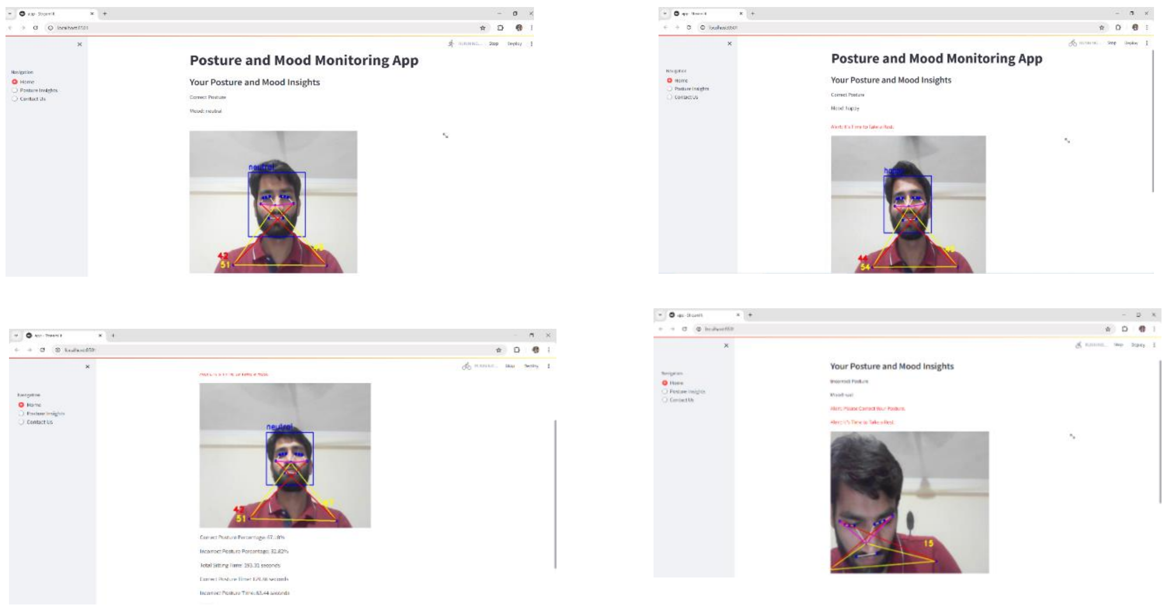
Streamlit is an open-source Python library that allows you to create web applications for machine learning, data science, and other computational tasks with minimal effort. It simplifies the process of building interactive web-based tools, enabling developers to quickly prototype and deploy their machine learning models, data visualizations, and analytical workflows.

Our Streamlit application integrates posture detection and mood monitoring functionalities to provide users with insights into their well-being. By combining deep learning models for both posture and mood monitoring, our app offers real-time feedback on body posture correctness and emotional states.

## 3.2. Home Section:

Posture Detection: Utilizes a pose tracking module to analyze the user's body posture in real-time. The module detects key body landmarks and evaluates the angles between them to determine whether the posture is correct or incorrect.

Mood Detection: Employs an emotion detection module to assess the user's emotional state based on facial expressions captured from the video feed. The module classifies emotions into categories such as happy, sad, neutral and disgust. Fig. 3.1 depicts the home section of the Posture and Mood Monitoring web app which is showing posture and mood insights in the real time.


Posture and Mood Monitoring App

Your Posture and Mood Insights

Correct Posture

Mood: neutral

Posture and Mood Monitoring App

Your Posture and Mood Insights

Correct Posture

Mood: happy

Alert: It's time to take a Rest.

Correct Posture Percentage: 67.18%

Incorrect Posture Percentage: 32.82%

Total Sitting Time: 193.31 seconds

Correct Posture Time: 129.86 seconds

Incorrect Posture Time: 63.44 seconds

Your Posture and Mood Insights

Incorrect Posture

Mood: sad

Alert: Please Correct Your Posture.

Alert: It's Time to Take a Rest.

Fig. 3.1 Posture and Mood Monitoring Web App

## 3.3. Visualization:

Provides visual feedback on both posture mood through interactive widgets and video feed display. Users can observe their posture correctness and current mood status on the interface.

## 3.4. Posture Insights Section:

Pie Chart Visualization: Presents insights into posture correctness over time using a pie chart.The chart illustrates the distribution of correct and incorrect posture time, allowing users to track their posture habits. Fig. 3.2 depicts posture insights section of the web app which is showing the correct and incorrect sitting time percentages through a pie chart.

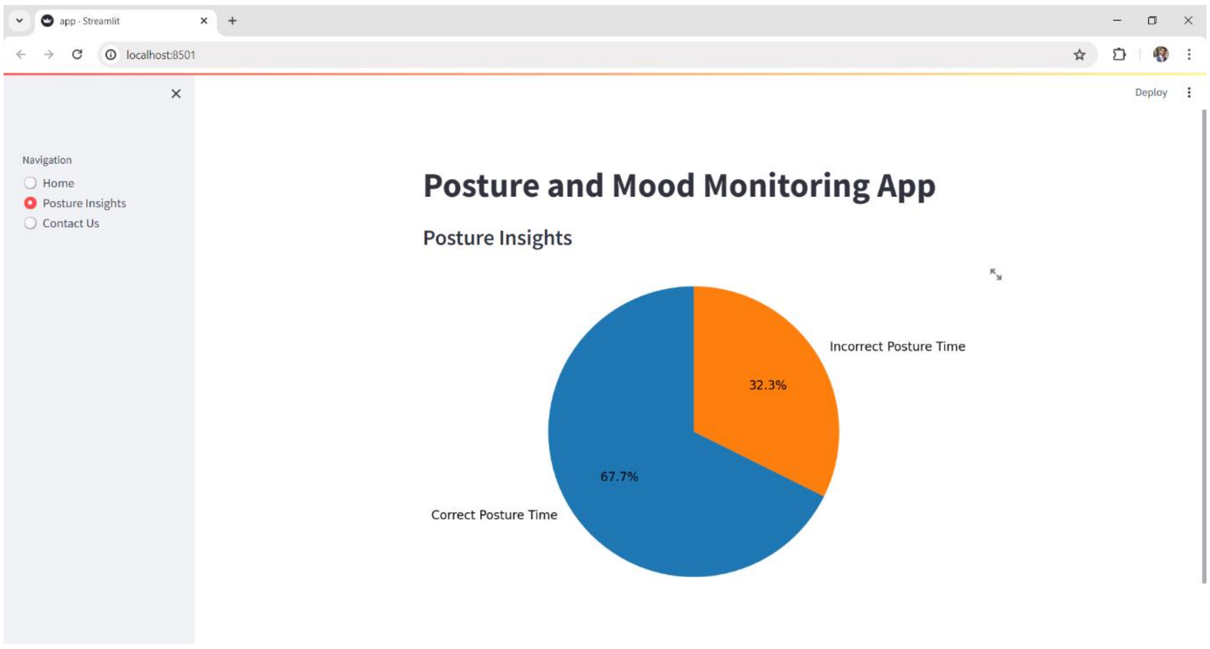


Fig. 3.2 Posture Insights Section

## 3.5. Contact Us Section:

Communication Form: Allows users to reach out to us for inquiries or feedback. Users can enter their name, email, and message in the form and submit it directly from the app as shown in fig. 3.3.

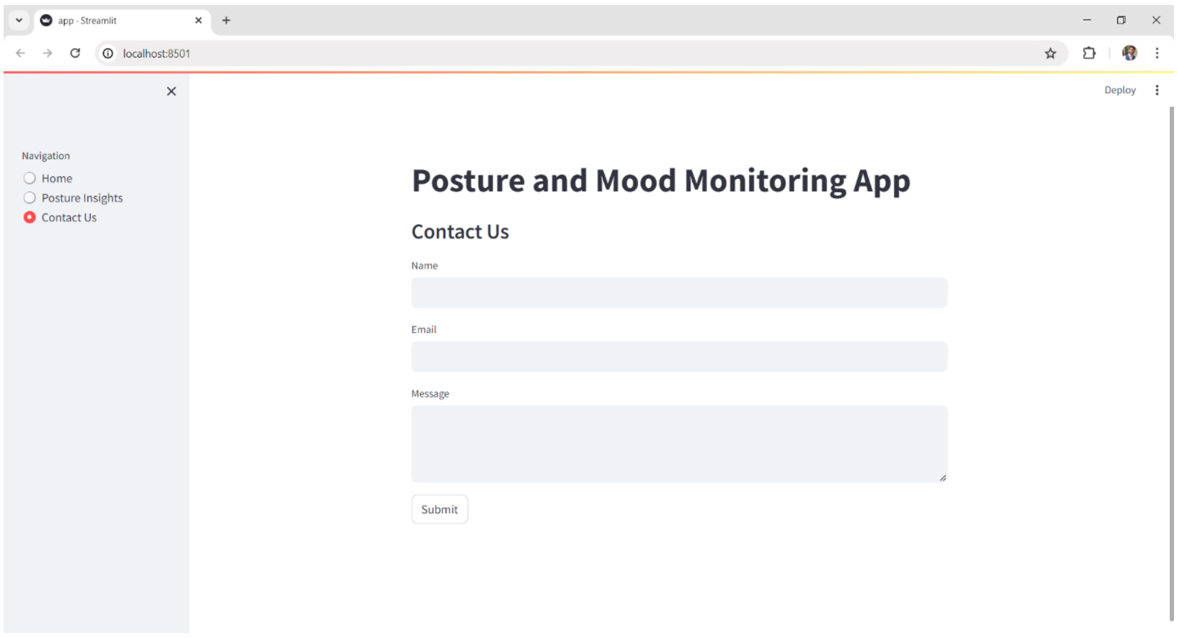


Fig. 3.3 Contact Us Section

# CHAPTER 4

# FUTURE SCOPE

The future scope of the posture and mood monitoring system encompasses several avenues for further development and enhancement to maximize its effectiveness and usability.

## 4.1. Integration of Additional Features:

One potential direction for future development involves integrating additional features to provide users with a more comprehensive monitoring experience. This could include features such as activity tracking, stress level analysis, and sleep pattern monitoring. By expanding the range of monitored parameters, the system can offer users a more holistic view of their overall well-being.

## 4.2. Personalized Feedback and Recommendation:

Incorporating machine learning algorithms to analyze user data over time can enable the system to provide personalized feedback and recommendations tailored to each user's unique needs and goals. By leveraging historical data on posture, mood, and other relevant metrics, the system can offer insights into behavior patterns and suggest personalized strategies for improvement.

## 4.3. Enhanced User Interface and Interaction:

Improvements to the user interface and interaction design can enhance the usability and accessibility of the system. This could involve the development of mobile applications or web-based interfaces with intuitive dashboards, visualization tools, and customizable settings. Additionally, integrating voice commands or gesture-based controls can offer users a more seamless and intuitive interaction experience.

## 4.4. Validation and Clinical Applications:

Conducting further validation studies and clinical trials can help validate the efficacy and reliability of the system for use in healthcare and clinical settings. Collaborating with healthcare professionals and researchers can facilitate the development of evidence-based interventions and treatment plans leveraging the data collected by the system.

## 4.5. Scalability and Deployment:

Optimizing the system for scalability and deployment in diverse environments, such as workplaces, schools, and healthcare facilities, can broaden its impact and reach. This may involve developing cloud-based infrastructure, ensuring compatibility with various hardware configurations, and implementing robust security and privacy measures to safeguard user data.

# CHAPTER 5
# CONCLUSION

Our project on posture and mood monitoring using deep learning has showcased the potential of advanced technology in promoting holistic well-being. By leveraging deep learning techniques and innovative frameworks like MediaPipe, Keras, etc. we have developed a comprehensive system capable of monitoring both physical posture and emotional states in real time.

Through the implementation of pose estimation model, we have successfully tracked and analyzed human skeletal landmarks, providing valuable insights into posture correctness and potential ergonomic issues. Additionally, by integrating mood detection algorithms, we have enabled our system to assess emotional states, contributing to a more nuanced understanding of users' mental well-being.

The integration of posture and mood monitoring models into a streamlit web application not only offers individuals greater awareness of their physical and emotional health but also opens avenues for proactive intervention and personalized feedback through real time posture and mood insights. By harnessing the power of deep learning, our project has the potential to revolutionize how we approach self-care and preventive healthcare practices.

Looking ahead, further refinements and enhancements to our system could include broader deployment in various settings, such as workplaces, schools, and healthcare facilities, to support individuals in optimizing their overall health and productivity. With ongoing advancements in deep learning and related technologies, the future holds promising prospects for the continued evolution and impact of posture and mood monitoring systems on human well-being.

## References

[1] C. Liao, R. Chen and S. Tai, "Emotion stress detection using EEG signal and deep learning technologies," 2018 IEEE International Conference on Applied System Invention (ICASI), p. 90-93, 2018.

[2] M. I. Georgescu, R. T. Ionescu, and M. Popescu, “Local learning with deep and handcrafted features for facial expression recognition,” IEEE Access, vol. 7, 2019.

[3] M. K. Pramerdorfer, Christopher, “Facial expression recognition using convolutional neural networks: state of the art,” arXiv, vol. 1612.02903, 2016.

[4] Jason Brownlee, “A Gentle Introduction to Deep Learning for Face Recognition,” Deep Learning for Computer Vision, 2019.

[5] Cowie R, Douglas-Cowie E, Tsapatsoulis N, Votsis G, Kollias S, Fellenz W, Taylor JG (2001) Emotion recognition in human–computer interaction. IEEE Signal Process Mag 18(1):32.

[6] D. A. Pitaloka, A. Wulandari, T. Basaruddin, and B. Y. Liliana,“Enhancing CNN with Preprocessing Stage in Automatic EmotionRecognition,” Procedia Comput. Sci., vol. 116, pp. 523–529, 2017.

[7] S. Deshmukh, M. Patwardhan, and A. Mahajan, “Survey on Real-Time Facial Expression Recognition Techniques,” IET Biom., pp. 1-9, 2015.

[8] C. J. L. Flores, A. E. G. Cutipa and R. L. Enciso, "Application of convolutional neural networks for static hand gestures recognition under different invariant features," 2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Cusco, 2017, pp. 1-4.

[9] Gao, H., Yüce, A., & Thiran, J.-P. (2014). Detecting emotional stress from facial expressions for driving safety. 2014 IEEE International Conference on Image Processing (ICIP), 5961–5965. https://doi.org/ 10.1109/ICIP.2014.7026203.

[10] Stress Detection Based on Emotion Recognition Using Deep Learning Anjali R1 , J. Babitha2 , Rithika W3 , Ms.Reeja S.L4 1,2,3Associate Software Engineer, Ospyn Technologies, Thiruvananthapuram, India 4Research Scholar, APJ Abdul Kalam KTU, 2349-6002, 2021, pp.109-111.

[11] Garg, S., Saxena, A. & Gupta, R. Yoga pose classification: a CNN and MediaPipe inspired deep learning approach for real-world application. J Ambient Intell Human Comput 14, 16551–16562 (2023). https://doi.org/10.1007/s12652-022-03910-0

[12] Jheanel Estrada, Larry Vea. Computer Users Sitting Posture Classification Using Distinct Feature Points and Small Scale Convolutional Neural Network for Humana Computer Intelligent Interactive System During COVID-19, Journal of Computer Science 19(4):493-513, April 2023.

[13] Bhatlawande, S., & Girgaonkar, I. Elderly Care System for Classification and Recognition of Sitting Posture. In 2022 2nd International Conference on Intelligent Technologies (CONIT) (pp. 1-7). IEEE. June 2022.

[14] Yang, H., & Yang, X. Video Sitting Posture Recognition of Human Skeletal Features Based on Deep Learning. *International Journal of Simulation: Systems, Science & Technology, April* 2022.

[15] Huseyin COSKUN, Human Sitting Postures Classification Based on Angular Features with Fuzzy-Logic Labeling, 5th International Conference on Applied Engineering and Natural Sciences, July 10, 2023

## Appendix

### Mood Detection Model  Training Code:

```
# Import required libraries

import cv2
import PIL
import io
import seaborn as sns
import pandas as pd
import math
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
from tqdm.notebook import tqdm

from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU, Activation
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau

# defining training and testing directories

TRAIN_DIR = "/kaggle/input/emotion-dataset//images/train"
TEST_DIR = "/kaggle/input/emotion-dataset//images/validation"

# create dataframe of images

def createdataframe(dir):
    image_paths = []
    labels = []

    try:
        for label in os.listdir(dir):
            for imagename in os.listdir(os.path.join(dir, label)):
                image_paths.append(os.path.join(dir, label, imagename))
                labels.append(label)
            print(label, "completed")
    except FileNotFoundError:
        print(f"Directory not found: {dir}")

    return image_paths, labels
```

```
# training dataset

train = pd.DataFrame()
train['image'], train['label'] = createdataframe(TRAIN_DIR)

# testing dataset
test = pd.DataFrame()
test['image'], test['label'] = createdataframe(TEST_DIR)

# function to extract features from images

from PIL import Image
def extract_features(images):
    features = []
    for image in tqdm(images):
        img = Image.open(image).convert('L')  # Load image in grayscale
        img = np.array(img)
        features.append(img)
    features = np.array(features)
    features = features.reshape(len(features), 48, 48, 1)
    return features

train_features = extract_features(train['image'])

test_features = extract_features(test['image'])

#normalize the features
x_train = train_features/255.0
x_test = test_features/255.0

from sklearn.preprocessing import LabelEncoder

# label encoding images
le = LabelEncoder()
le.fit(train['label'])
y_train = le.transform(train['label'])
y_test = le.transform(test['label'])

y_train = to_categorical(y_train,num_classes = 4)
y_test = to_categorical(y_test,num_classes = 4)


from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create an ImageDataGenerator instance with augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,      # Random rotation up to 20 degrees
    width_shift_range=0.2,  # Random horizontal shift
    height_shift_range=0.2, # Random vertical shift
    shear_range=0.2,        # Shear intensity
    zoom_range=0.2,         # Random zoom
    horizontal_flip=True,   # Random horizontal flip
    fill_mode='nearest'     # Fill mode for pixels outside the boundaries
)

# Example usage with a batch of images
# Assume x_train is your training data with shape (num_samples, height, width, channels)
# and y_train is your corresponding labels
datagen.fit(x_train)

# Create a generator that yields augmented batches of data
augmented_generator = datagen.flow(x_train, y_train, batch_size=128)

# Callbacks

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    min_delta=0.00005,
    patience=7,
    verbose=1,
    restore_best_weights=True,
)

lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.3,
    patience=7,
    min_lr=1e-6,
    verbose=1,
)

callbacks = [
    early_stopping,
    lr_scheduler,
]

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
BatchNormalization, Activation, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import tensorflow as tf

# Create the Sequential model
model = Sequential()

# Module 1
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(48, 48, 1), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

# Module 2
model.add(Conv2D(64, kernel_size=(3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

# Module 3
model.add(Conv2D(128, kernel_size=(3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

# Flatten
model.add(Flatten())

# Dense layers
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.001),
              metrics=['accuracy'])
# Display the model summary
model.summary()

history  = model.fit(augmented_generator, steps_per_epoch=int(len(x_train) / 128),
epochs=200, validation_data=(x_test, y_test))

model_json = model.to_json()
with open("emotiondetector.json",'w') as json_file:
    json_file.write(model_json)
model.save("emotiondetector.h5")

from tensorflow.keras.models import model_from_json

json_file = open("/kaggle/working/emotiondetector.json", "r")
model_json = json_file.read()
json_file.close()
model = model_from_json(model_json)
```

```
model.load_weights("/kaggle/working/emotiondetector.h5")

# plot confusion matrix

from sklearn.metrics import confusion_matrix
# x_test and y_true are the test data and true labels respectively

# Make predictions using the loaded model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)

y_test = np.argmax(y_test, axis=1)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix")
plt.show()

# plot training and validation accuracy curve

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

**Mood Module:**

```
import cv2
import numpy as np
from keras.models import model_from_json

class Emotions:
    def __init__(self):
        # Load emotion detection model
        json_file = open("emotiondetector2.json", "r")
        model_json = json_file.read()
        json_file.close()
        self.model = model_from_json(model_json)
        self.model.load_weights("emotiondetector2.h5")

        # Load face cascade
                    self.face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

        self.labels = {0: 'disgust', 1: 'happy', 2: 'neutral', 3: 'sad'}

    def detect_emotions(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, 1.3, 5)

        # If no face is detected, return None
        if len(faces) == 0:
            return None, frame

        # Only consider the first face detected
        x, y, w, h = faces[0]
        face_roi = gray[y:y + h, x:x + w]

        # Resize the face ROI for the model
        try:
            face_roi = cv2.resize(face_roi, (48, 48))
        except Exception as e:
            print(str(e))
            return None, frame

        # Normalize the image
        face_roi = face_roi / 255.0

        # Reshape the image to match model input shape
        face_roi = np.expand_dims(face_roi, axis=0)
        face_roi = np.expand_dims(face_roi, axis=-1)

        # Predict emotion
        predicted_class = np.argmax(self.model.predict(face_roi), axis=-1)

        # Get the predicted emotion label
        predicted_label = self.labels[predicted_class[0]]

        # Draw bounding box and emotion label on the frame
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
        cv2.putText(frame, predicted_label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
(255, 0, 0), 2)

        return predicted_label, frame
```

## Posture Module:

```
import mediapipe as mp
import time
import math
import cv2

class PoseTracker:
    def __init__(self, min_detection_confidence=0.4, min_tracking_confidence=0.4):
        self.mp_pose = mp.solutions.pose
        self.pose = self.mp_pose.Pose(
            min_detection_confidence=min_detection_confidence,
            min_tracking_confidence=min_tracking_confidence
        )
        self.mpDraw = mp.solutions.drawing_utils
        self.lmList = []
    def detectPose(self, frame):
        # Convert the BGR image to RGB
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the image and get the pose landmarks
        self.results = self.pose.process(rgb_frame)

        # Check if pose landmarks are available
        if self.results.pose_landmarks:
            # Draw connections between landmarks
            self.mpDraw.draw_landmarks(frame, self.results.pose_landmarks,
self.mp_pose.POSE_CONNECTIONS)

        return frame

    def trackPose(self, frame, draw=True):
        self.lmList = []
        if self.results.pose_landmarks:
            # Get the first face detected
            face = self.results.pose_landmarks
            if face:
                for id, lm in enumerate(face.landmark):
                    h, w, c = frame.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
                    self.lmList.append([id, cx, cy])
                    if draw:
                        cv2.circle(frame, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
        return self.lmList

    def findAngle(self, frame, p1, p2, p3, p4, p5, draw=True):
        # Check if lmList is not empty and has enough elements
        if len(self.lmList) > max(p1, p2, p3, p4, p5):
            # Get the landmarks
            x1, y1 = self.lmList[p1][1:]
            x2, y2 = self.lmList[p2][1:]
            x3, y3 = self.lmList[p3][1:]
            x4, y4 = self.lmList[p4][1:]
            x5, y5 = self.lmList[p5][1:]

            # Calculate the Angle
            angle1 = math.degrees(math.atan2(y2 - y3, x2 - x3) -
                          math.atan2(y2 - y1, x2 - x1))
            angle2 = math.degrees(math.atan2(y1 - y3, x1 - x3) -
                          math.atan2(y2 - y3, x2 - x3))
            angle3 = math.degrees(math.atan2(y2 - y3, x2 - x3) -
                          math.atan2(y2 - y5, x2 - x5))
            angle4 = math.degrees(math.atan2(y4 - y3, x4 - x3) -
                          math.atan2(y2 - y3, x2 - x3))
            angle5 = math.degrees(math.atan2(y4 - y1, x4 - x1) -
                          math.atan2(y5 - y1, x5 - x1))

            angle_list = [angle1, angle2, angle3, angle4, angle5]
            angle_list = [a + 360 if a < 0 else a for a in angle_list]

            # Draw
            if draw:
                cv2.line(frame, (x1, y1), (x3, y3), (0, 255, 255), 2)
                cv2.line(frame, (x2, y2), (x3, y3), (0, 255, 255), 2)
                cv2.line(frame, (x1, y1), (x2, y2), (0, 255, 255), 2)
                cv2.line(frame, (x5, y5), (x2, y2), (0, 0, 255), 2)
                cv2.line(frame, (x4, y4), (x3, y3), (0, 0, 255), 2)
                cv2.line(frame, (x4, y4), (x1, y1), (255, 0, 255), 2)
                cv2.line(frame, (x5, y5), (x1, y1), (255, 0, 255), 2)

                cv2.putText(frame, str(int(angle_list[0])), (x2 - 50, y2+10),
                        cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 255), 3)
                cv2.putText(frame, str(int(angle_list[1])), (x3 - 50, y3-50),
                        cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 255), 3)
                cv2.putText(frame, str(int(angle_list[2])), (x2 - 60, y2-20),
                        cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 3)

            return angle_list
        else:
        # Handle the case where lmList is not populated enough
         return []

def main():
 cap = cv2.VideoCapture(0)
    detector = PoseTracker()

    p_Time = 0  # Initialize ptime outside the loop
    prevPose = None
    ptime = time.time()
    pos_time = 0
    itime = 0  # Initialize total incorrect posture time
    ctime = 0  # Initialize total correct posture time
    ref_time = time.time()  # Record the time when posture changes
    sitting_time = 0

    # Set the desired window width and height
    window_width = 550
    window_height = 480

    while True:
        success, frame = cap.read()
        # Check if the video has reached its end
        if not success:
            # Set the frame position back to the beginning
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            continue

        frame = detector.detectPose(frame)
        lmList = detector.trackPose(frame, draw=True)
        angle_list = detector.findAngle(frame, 0, 12, 11, 8, 7)
        # Resize the window
        frame_resized = cv2.resize(frame, (window_width, window_height))

        # Process posture
        if len(lmList) != 0:
            if angle_list != [0, 0]:
                total_angle_sum = angle_list[0] + angle_list[1]

                if total_angle_sum < 75 or angle_list[0] < 25 or angle_list[1] < 25 or \
                        angle_list[0] > 300 or angle_list[1] > 300:

                    pos = "Incorrect Posture"
                else:
                    pos = "Correct Posture"
                    cv2.putText(frame_resized, pos, (5, 30), cv2.FONT_HERSHEY_COMPLEX, 1,
(255, 0, 0), 1)

                # Calculate posture time based on the difference between the current time and time
when posture was detected
                pos_time = time.time() - ptime

                if pos == "Correct Posture":
                    ctime += (time.time() - ref_time)
                    ref_time = time.time()
                    cv2.putText(frame_resized, "Total Correct Posture Time: " + str(round(ctime, 1))
+ " seconds", (5, 90), cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 100), 1)
                else:
                    itime += (time.time() - ref_time)
                    ref_time = time.time()
                    cv2.putText(frame_resized, "Total Incorrect Posture Time: " + str(round(itime,
1)) + " seconds", (5, 90), cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 100), 1)

                # Display posture time on the frame
                cv2.putText(frame_resized, "Posture Time: " + str(round(pos_time, 1)) + "
seconds", (5, 70), cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 200), 1)
                if pos_time > 5 and pos == "Incorrect Posture":
                    cv2.putText(frame_resized, "Alert: Correct Your Sitting Posture.", (5,120),
cv2.FONT_HERSHEY_COMPLEX,0.75,(0,0,255),1)
```

```
        #Pomodore timer
        sitting_time = itime + ctime
        cv2.putText(frame_resized, "Total Sitting Time:" + str(round(sitting_time,2)), (5,150),
cv2.FONT_HERSHEY_COMPLEX,0.75,(200,100,50),1)
        if sitting_time > 500:
            cv2.putText(frame_resized, "It's rest time." + str(round(sitting_time,2)), (5,200),
cv2.FONT_HERSHEY_COMPLEX,0.75,(200,100,50),1)
        # Check if lmList is not empty and if the first keypoint exists
        if lmList and len(lmList) > 0 and lmList[0] is None:
            sitting_time = 0
            itime =0
            ctime =0
        #frame rate
        c_Time = time.time()
        fps = 1 / (c_Time - p_Time)
        p_Time = c_Time

        #cv2.putText(frame_resized, str(int(fps)), (70, 50), cv2.FONT_HERSHEY_PLAIN, 3,
(255, 0, 0), 3)
        cv2.imshow("Frame", frame_resized)

        key = cv2.waitKey(1)
        if key == 27:  # Press 'Esc' to exit
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

## Streamlit Web App Code:

```
import streamlit as st
import time
import cv2
from posture_module import PoseTracker
from mood_module import Emotions
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Declare global variables for time-related measurements
st.session_state.setdefault('ctime', 0)
st.session_state.setdefault('itime', 0)
st.session_state.setdefault('pos_time', 0)
st.session_state.setdefault('ref_time', time.time())
st.session_state.setdefault('ref_time2', time.time())
st.session_state.setdefault('p_Time', 0)
st.session_state.setdefault('sitting_time', 0)
st.session_state.setdefault('pos', None)
st.session_state.setdefault('frames', None)
```

```
def home_section():
    st.subheader('Your Posture and Mood Insights')

    # Initialize pose tracker
    detector = PoseTracker()

    # Initialize emotions detector
    emotions_detector = Emotions()

    # Set the desired window width and height
    window_width = 500
    window_height = 450

    # Initialize variables
    p_Time = st.session_state.p_Time
    pos = st.session_state.pos
    pos_time = st.session_state.pos_time

    # Create a container for the blocks
    container = st.container()

    # Create containers for display
    posture_container = st.empty()
    mood_container = st.empty()
    alert_block1 = st.empty()
    alert_block2 = st.empty()
    video_container = st.empty()
    correct_percentage_container = st.empty()
    incorrect_percentage_container = st.empty()
    total_sitting_time = st.empty()
    correct_posture_time = st.empty()
    incorrect_posture_time = st.empty()
    ipose_time_container = st.empty()  # Changed variable name to avoid confusion

    cap = cv2.VideoCapture(0)

    while True:
        success, frame = cap.read()

        if not success:
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            continue

        # Detect mood
        prediction_label, frame = emotions_detector.detect_emotions(frame)

        # Update mood container
        if prediction_label:
            mood_container.write(f'Mood: {prediction_label}')

        # Detect posture
        frame = detector.detectPose(frame)
        lmList = detector.trackPose(frame, draw=True)
        angle_list = detector.findAngle(frame, 0, 12, 11, 8, 7)

        # Resize frame
        frame_resized = cv2.resize(frame, (window_width, window_height))

        # Display video feed
        video_container.image(frame_resized, channels='BGR', width=480)

        # Process posture
        if len(lmList) != 0:
            if angle_list != [0, 0]:
                total_angle_sum = angle_list[0] + angle_list[1]
                if total_angle_sum < 75 or angle_list[0] < 25 or angle_list[1] < 25 or \
                        angle_list[0] > 300 or angle_list[1] > 300:
                    pos = "Incorrect Posture"
                else:
                    pos = "Correct Posture"

                posture_container.write(pos)

                if pos == "Correct Posture":
                    st.session_state.ctime += (time.time() - st.session_state.ref_time)
                else:
                    st.session_state.itime += (time.time() - st.session_state.ref_time)

                st.session_state.ref_time = time.time()

        st.session_state.pos_time += time.time() - st.session_state.ref_time2
        ipose_time = round(st.session_state.pos_time, 2)  # Correctly assigning ipose_time here
        ipose_time_container.write(ipose_time)  # Writing ipose_time to container
        st.session_state.ref_time2 = time.time()
        if pos == "Correct Posture":
            st.session_state.pos_time = 0

        # Incorrect posture message
        incorrect_posture_message = ""
        if st.session_state.pos_time > 5 and pos == "Incorrect Posture":
            incorrect_posture_message = "<span style='color:red'>Alert: Please Correct Your
Posture.</span>"
        if pos == "Correct Posture":
            incorrect_posture_message = ""
        alert_block1.write(incorrect_posture_message, unsafe_allow_html=True)

        # Display correct and incorrect posture time
          correct_posture_time.write(f'Correct Posture Time: {round(st.session_state.ctime, 2)}
seconds')

        incorrect_posture_time.write(f'Incorrect Posture Time: {round(st.session_state.itime, 2)}
seconds')

        # Calculate total sitting time based on updated values
        sitting_time = st.session_state.itime + st.session_state.ctime
        total_sitting_time.write(f'Total Sitting Time: {round(sitting_time, 2)} seconds')

        # Calculate percentages
        if sitting_time != 0:
            correct_percentage = (st.session_state.ctime / sitting_time) * 100
                              correct_percentage_container.write(f'Correct   Posture   Percentage:
{correct_percentage:.2f}%')
        if sitting_time != 0:
            incorrect_percentage = (st.session_state.itime / sitting_time) * 100
                          incorrect_percentage_container.write(f'Incorrect   Posture   Percentage:
{incorrect_percentage:.2f}%')

        # Rest alert
        rest_time_message = ""
        if sitting_time > 20:
          rest_time_message   = "<span style='color:red'>Alert: It's Time to Take a Rest.</span>"
        alert_block2.write(rest_time_message, unsafe_allow_html=True)

        if lmList is None or len(lmList) == 0:
            sitting_time = 0
            st.session_state.itime = 0
            st.session_state.ctime = 0
            rest_time_message = ""
        alert_block2.write(rest_time_message, unsafe_allow_html=True)

        # Display correct and incorrect time
        ctime = round(st.session_state.ctime, 2)
        itime = round(st.session_state.itime, 2)

        cv2.waitKey(1)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    return frame_resized

def posture_insights(ctime, itime):
    st.subheader('Posture Insights')

    # Create a pie chart for correct and incorrect posture time
    posture_data = {'Category': ['Correct Posture Time', 'Incorrect Posture Time'],
                    'Time': [ctime, itime]}  # You may need to adjust the data source based on your
implementation

    # Create a DataFrame from the posture data
    df = pd.DataFrame(posture_data)

    # Plot the posture insights using a pie chart
    fig, ax = plt.subplots()
    ax.pie(df['Time'], labels=df['Category'], autopct='%1.1f%%', startangle=90)
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

    # Display the pie chart in the Streamlit app
    st.pyplot(fig)

def contact_us_section():
    st.subheader('Contact Us')

    # Add form for reaching out
    name = st.text_input('Name')
    email = st.text_input('Email')
    message = st.text_area('Message')
    submit_button = st.button('Submit')

    if submit_button:
        # Process the form submission
        # You can add your logic here to send the message or store it in a database
        st.success('Message Sent!')

def main():
    st.title('Posture and Mood Monitoring App')

    # Add a navigation bar
    nav_selection = st.sidebar.radio('Navigation', ['Home', 'Posture Insights', 'Contact Us'])

    # Display sections based on navigation selection
    if nav_selection == 'Home':
        home_section()
    elif nav_selection == 'Contact Us':
        contact_us_section()
    elif nav_selection == 'Posture Insights':
        posture_insights(st.session_state.ctime, st.session_state.itime)

if __name__ == '__main__':
    main()
```