Artificial Intelligence and Machine Learning

Project Report

Semester-IV(Batch-2022)

AI Powered Yoga Pose Detector

Supervised by:

Mr. Sachin Garg

(G-22)

Submitted by:

Lavanya Kapoor

2210991850

Department of Computer Science and Engineering

Chitkara University Institute of Engineering & Technology,

Chitkara University, Punjab

Abstract

Yoga pose detection, a cutting-edge fusion of computer vision and machine learning technologies, represents a significant stride in the realm of health and wellness. This report serves as a comprehensive exploration of yoga pose detection, offering insights into its technological underpinnings, operational mechanisms, multifaceted applications, and looming challenges. By leveraging the prowess of artificial intelligence (AI), yoga pose detection systems endeavour to automatically discern and categorize yoga postures from visual data, thereby revolutionizing the practice and accessibility of yoga for individuals across diverse demographics.

The foundational technologies driving yoga pose detection encompass computer vision and machine learning. Computer vision techniques empower these systems to analyze visual data, such as images or video frames, while machine learning algorithms, notably deep learning models like Convolutional Neural Networks (CNNs), are instrumental in classifying yoga poses based on learned patterns and features. The synergy between these technologies enables the extraction of pertinent information regarding body joints and landmarks, facilitating the accurate identification and classification of yoga postures.

The applications of yoga pose detection are as diverse as they are transformative. Primarily, these systems serve as virtual yoga instructors, furnishing practitioners with instantaneous feedback on posture alignment and form during yoga sessions. Moreover, they facilitate progress monitoring by tracking practitioners' performance over time and identifying areas for improvement. Beyond individual practice, yoga pose detection technology holds promise in healthcare and wellness domains, where it can be integrated into applications aimed at promoting physical fitness, rehabilitation, and stress management.

In conclusion, yoga pose detection stands at the vanguard of technological innovation, poised to reshape the landscape of yoga practice and wellness promotion. Through its amalgamation of computer vision and machine learning technologies, yoga pose detection democratizes access to personalized yoga instruction, fosters holistic well-being, and augurs a future where technology serves as an enabler of physical and mental flourishing.

1.Introduction

1.1 Background

A yoga pose detector utilizes computer vision techniques to automatically identify and recognize yoga poses from images or videos. It begins with preprocessing the input data, which involves enhancing the quality of the image or video frame through resizing, normalization, and noise reduction. The core component is the pose detection model, which analyze the input and detects key points corresponding to body parts. This can be achieved through deep learning-based pose estimation or keypoint detection techniques like OpenPose. Once key points are detected, they are used to classify the yoga pose being performed. Machine learning algorithms are often employed for this classification, trained on labeled datasets of yoga poses.

The output of the detector is the identification of the yoga pose(s) in the input, which can be visualized by overlaying labels or annotations. Integration into various applications, such as fitness apps or virtual yoga classes, allows users to receive real-time feedback and guidance on their yoga practice, aiding in improving their form and technique.

Implementing a yoga pose detector involves several key steps. Initially, you need to collect a dataset of images or videos featuring individuals performing various yoga poses, and annotate each with the corresponding pose label. Then, preprocess the data to enhance quality and normalize conditions. Next, select an appropriate pose detection model or technique and train it on the annotated dataset, optimizing its parameters for yoga pose detection. Integration into an application follows, where the trained model is embedded for real-time inference. Feedback mechanisms, like overlaying pose labels or providing alignment cues, enhance user experience. Rigorous testing and evaluation ensure accuracy and real-time performance across diverse conditions. Continuous refinement based on user feedback and updates in pose detection research is essential. Finally, deploy the finalized yoga pose detector, monitoring its performance and applying necessary updates for ongoing reliability and effectiveness.

1.2 Objectives

1.2.1 Automatic Recognition: The primary objective of yoga pose detection is to automatically recognize and identify various yoga postures performed by individuals. By leveraging computer vision and machine learning techniques, the system aims to accurately discern the specific pose being executed in real-time.

1.2.2 Pose Classification: Another objective is to classify detected yoga poses into predefined categories. This involves training machine learning models on labeled datasets of yoga poses to enable the system to categorize new instances into the appropriate pose categories with a high degree of accuracy.

1.2.3 Real-time Feedback: Yoga pose detection systems aim to provide real-time feedback to practitioners on their posture alignment, form, and execution during yoga sessions. This feedback can help practitioners adjust their poses in-the-moment to achieve better alignment and maximize the benefits of their practice.

1.2.4 Progress Monitoring: Additionally, yoga pose detection facilitates progress monitoring by tracking practitioners' performance over time. By analyzing changes in posture alignment and form across multiple sessions, practitioners can gauge their progress, identify areas for improvement, and set goals to enhance their practice.

1.2.5 Accessibility and Convenience: One of the objectives of yoga pose detection is to make yoga more accessible and convenient for individuals of all skill levels and abilities. By offering virtual yoga instruction and guidance through digital platforms, these systems enable practitioners to engage in yoga practice anytime, anywhere, without the need for in-person instruction.

1.2.6 Integration into Wellness Applications: Yoga pose detection technology aims to integrate seamlessly into various wellness applications, including fitness trackers, health monitoring devices, and rehabilitation programs. By incorporating yoga practice data, these applications can provide personalized recommendations, insights, and interventions to support users' overall health and well-being.

1.3. Significance

The significance of yoga pose detection lies in its transformative impact on the practice of yoga and its broader implications for health, wellness, and technological innovation. By providing personalized instruction and real-time feedback, these systems enhance the learning experience for practitioners of all levels, fostering proper alignment, safety, and confidence in their practice. Moreover, yoga pose detection promotes accessibility and inclusivity by removing barriers to participation, making yoga more accessible to individuals with physical limitations or geographical constraints. Beyond yoga practice, the technology has applications in health monitoring, rehabilitation, and sports performance, driving advancements in interdisciplinary fields such as computer vision, machine learning, and human movement analysis. As research and development efforts continue, yoga pose detection opens up opportunities for innovation, collaboration, and the exploration of new frontiers in technology-enabled wellness solutions.

2.1 Problem Statement

Despite the growing popularity of yoga as a holistic wellness practice, many individuals struggle to achieve proper alignment and form during their yoga sessions, leading to potential risks of injury and reduced effectiveness of the practice. Traditional methods of yoga instruction often rely on in-person guidance from instructors, which may not always be accessible or convenient for all practitioners.

The problem statement revolves around the necessity for innovative solutions, such as yoga pose detection systems, to overcome barriers to effective yoga practice and promote safer, more personalized experiences for practitioners. Addressing this problem holds the potential to revolutionize the way people engage with yoga, fostering greater accessibility, inclusivity, and effectiveness in achieving holistic wellness goals.

2.2 Requirements

Operating System: Windows 10

Python Environment: Python

Libraries and Dependencies:

OpenCV: A versatile library for computer vision tasks, providing robust tools for image processing, feature detection, and video analysis.

MediaPipe: MediaPipe is a powerful library developed by Google that allows for building machine learning pipelines for various types of media processing tasks, such as image and video processing. It provides a wide range of pre-built components for tasks like face detection, hand tracking, pose estimation, object detection, and more.

NumPy: NumPy is a fundamental library for numerical computing in Python. It provides support for arrays, matrices, and a variety of mathematical functions to operate on these data structures efficiently.

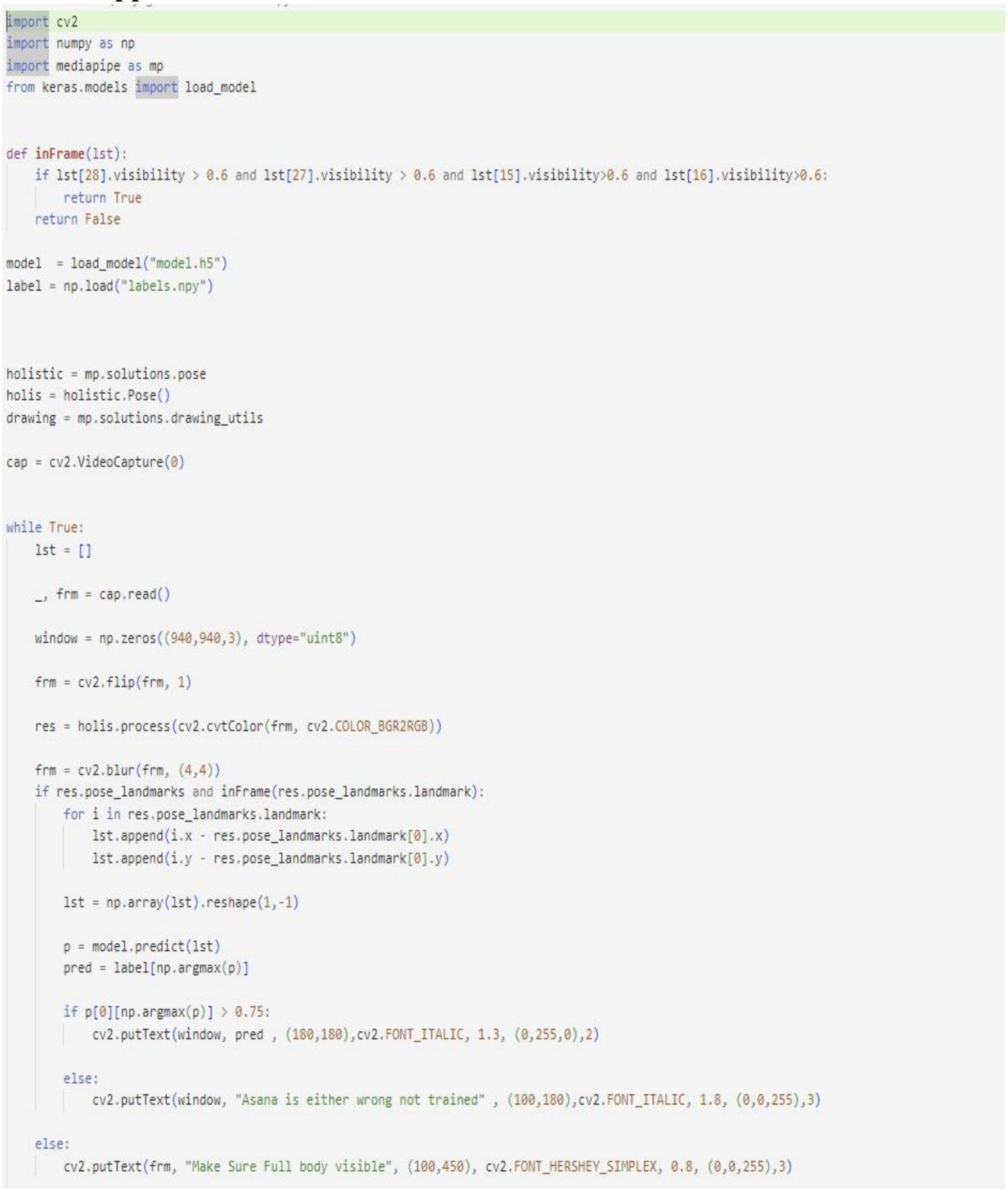
Keras: Keras is an open-source deep learning library written in Python. It's designed to be user-friendly, modular, and extensible, allowing both beginners and experts to quickly build and experiment with neural networks

dlib: A C++ library containing machine learning algorithms and tools, particularly renowned for its implementation of facial landmark detection and face recognition.

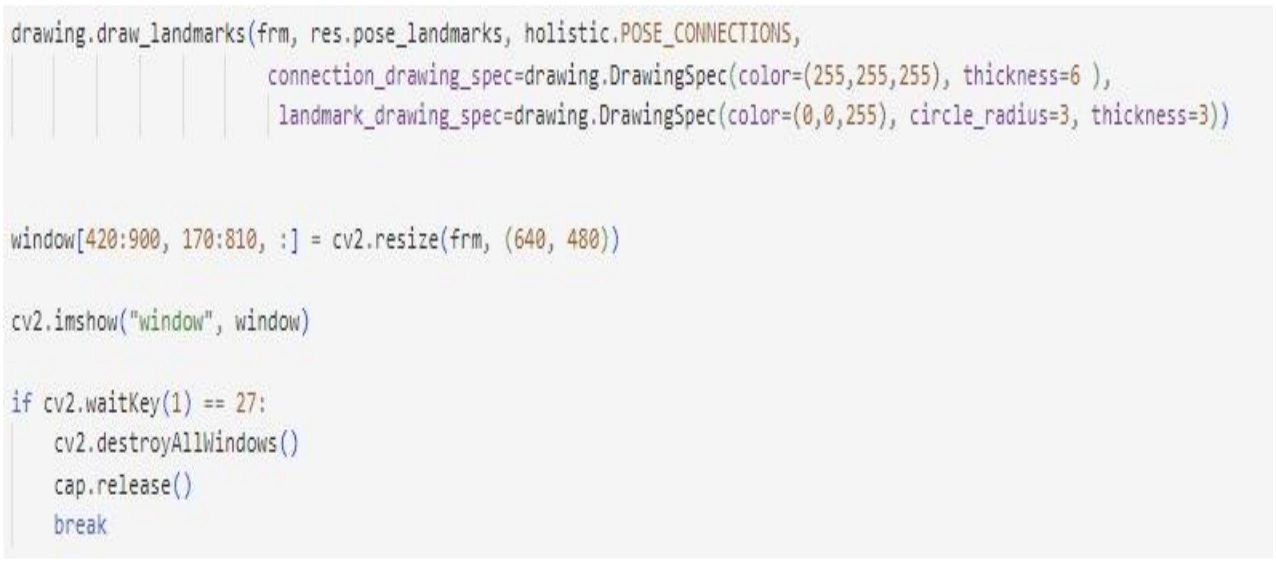
TensorFlow: An open-source machine learning framework developed by Google, widely used for building and training deep neural networks, including those for facial recognition tasks.

PyTorch: Another popular open-source deep learning framework, maintained by Facebook's AI Research lab, known for its flexibility and ease of use in developing neural network models, including those for facial recognition.

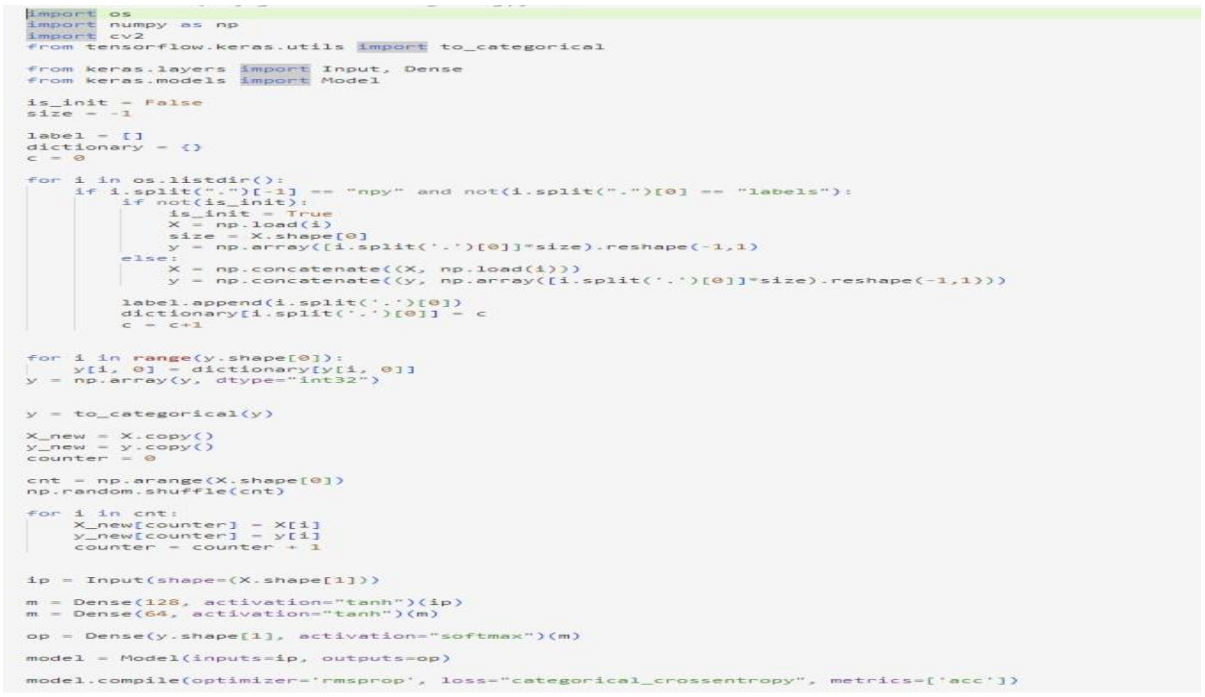
Code Snippet:



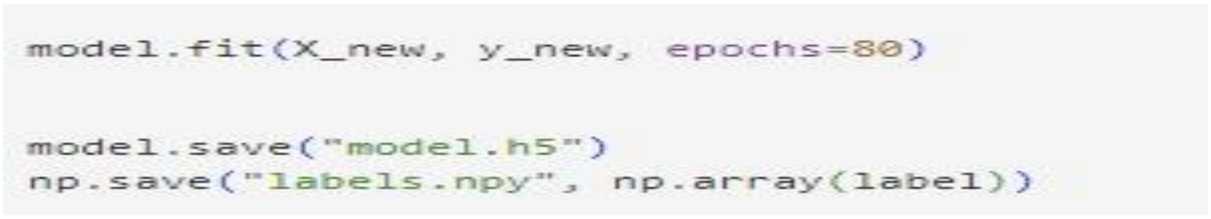
```python
import cv2
import numpy as np
import mediapipe as mp
from keras.models import load_model


def inFrame(lst):
    if lst[28].visibility > 0.6 and lst[27].visibility > 0.6 and lst[15].visibility>0.6 and lst[16].visibility>0.6:
        return True
    return False

model  = load_model("model.h5")
label = np.load("labels.npy")



holistic = mp.solutions.pose
holis = holistic.Pose()
drawing = mp.solutions.drawing_utils

cap = cv2.VideoCapture(0)


while True:
    lst = []

    _, frm = cap.read()

    window = np.zeros((940,940,3), dtype="uint8")

    frm = cv2.flip(frm, 1)

    res = holis.process(cv2.cvtColor(frm, cv2.COLOR_BGR2RGB))

    frm = cv2.blur(frm, (4,4))
    if res.pose_landmarks and inFrame(res.pose_landmarks.landmark):
        for i in res.pose_landmarks.landmark:
            lst.append(i.x - res.pose_landmarks.landmark[0].x)
            lst.append(i.y - res.pose_landmarks.landmark[0].y)

        lst = np.array(lst).reshape(1,-1)

        p = model.predict(lst)
        pred = label[np.argmax(p)]

        if p[0][np.argmax(p)] > 0.75:
            cv2.putText(window, pred , (180,180),cv2.FONT_ITALIC, 1.3, (0,255,0),2)

        else:
            cv2.putText(window, "Asana is either wrong not trained" , (100,180),cv2.FONT_ITALIC, 1.8, (0,0,255),3)

    else:
        cv2.putText(frm, "Make Sure Full body visible", (100,450), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255),3)
```



```
drawing.draw_landmarks(frm, res.pose_landmarks, holistic.POSE_CONNECTIONS,
                        connection_drawing_spec=drawing.DrawingSpec(color=(255,255,255), thickness=6 ),
                         landmark_drawing_spec=drawing.DrawingSpec(color=(0,0,255), circle_radius=3, thickness=3))


window[420:900, 170:810, :] = cv2.resize(frm, (640, 480))

cv2.imshow("window", window)

if cv2.waitKey(1) == 27:
    cv2.destroyAllWindows()
    cap.release()
    break
```

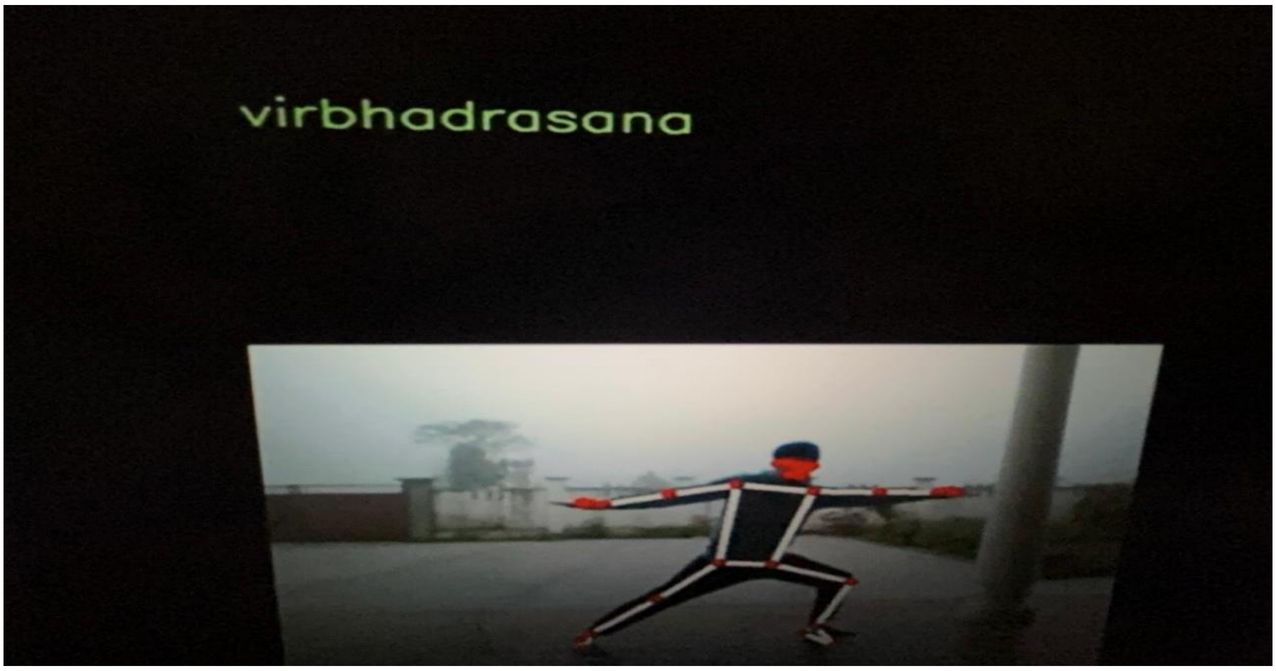


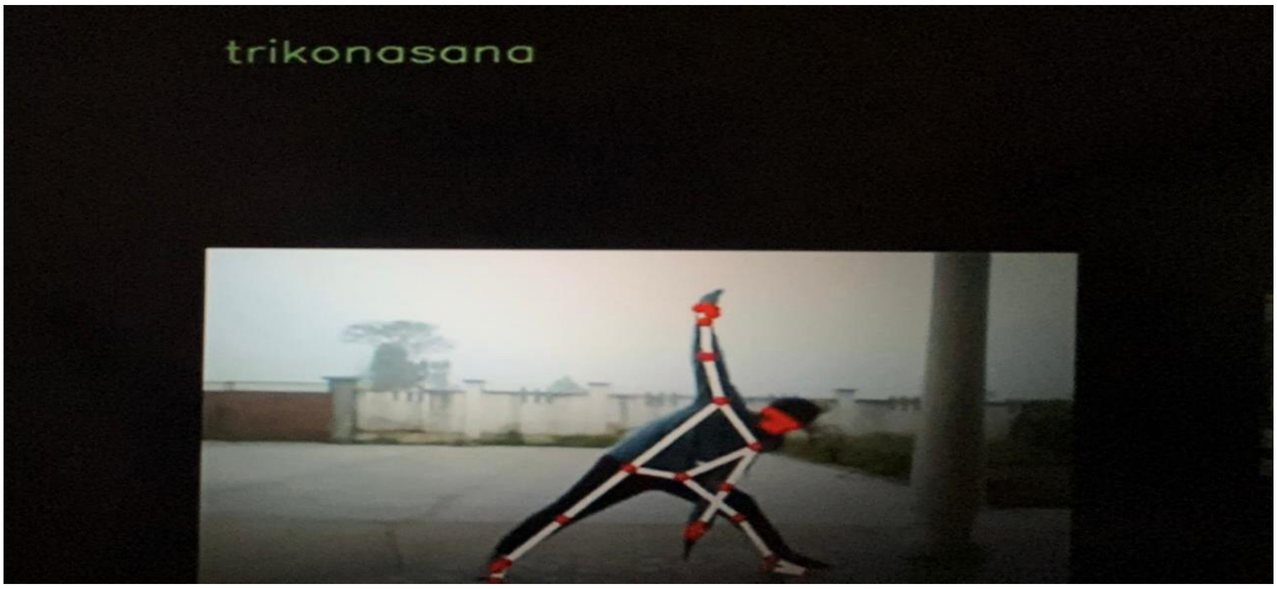
```
import os
import numpy as np
import cv2
from tensorflow.keras.utils import to_categorical

from keras.layers import Input, Dense
from keras.models import Model

is_init = False
size = -1

label = []
dictionary = {}
c = 0

for i in os.listdir():
    if i.split(".")[-1] == "npy" and not(i.split(".")[0] == "labels"):
        if not(is_init):
            is_init = True
            X = np.load(i)
            size = X.shape[0]
            y = np.array([i.split('.')[0]]*size).reshape(-1,1)
        else:
            X = np.concatenate((X, np.load(i)))
            y = np.concatenate((y, np.array([i.split('.')[0]]*size).reshape(-1,1)))

        label.append(i.split('.')[0])
        dictionary[i.split('.')[0]] = c
        c = c+1


for i in range(y.shape[0]):
    y[i, 0] = dictionary[y[i, 0]]
y = np.array(y, dtype="int32")


y = to_categorical(y)

X_new = X.copy()
y_new = y.copy()
counter = 0

cnt = np.arange(X.shape[0])
np.random.shuffle(cnt)

for i in cnt:
    X_new[counter] = X[i]
    y_new[counter] = y[i]
    counter = counter + 1


ip = Input(shape=(X.shape[1]))

m = Dense(128, activation="tanh")(ip)
m = Dense(64, activation="tanh")(m)

op = Dense(y.shape[1], activation="softmax")(m)

model = Model(inputs=ip, outputs=op)

model.compile(optimizer='rmsprop', loss="categorical_crossentropy", metrics=['acc'])
```



```
model.fit(X_new, y_new, epochs=80)


model.save("model.h5")
np.save("labels.npy", np.array(label))
```

Result:





References:

https:// Explore applications on QuickPose.ai (QuickPose.ai)/

https://viso.ai/(viso.ai)/