Final Report Submitted

by Ayush vij

Submission date: 08-Jan-2023 06:38PM (UTC+0530)
Submission ID: 1987641515
File name: Final_Project_Report.docx (1.22M)
Word count: 7255
Character count: 35653

AI Yoga Asana Estimator

A project report submitted in partial fulfilment of the requirement for

Disruptive Technologies-1

(21ECP-102)

By

Group No - 2

S.No.	UID	Name	Responsibility
1	22BAI71060	Sanchit Wadehra	Code, Proposed Framework, Result
2	22BAI71077	Harshita Sharma	Background
3	22BAI71090	Ayush Vij	Abstract,Intro
4	22BAI71063	Aryan Tomar	Conclusion & Future Scope
5	22BAI71077	Mayank	Reference

under the guidance of

Dr. Kiranjot Singh

UIE, Chandigarh University

Table of Contents

List of Figures

List of Tables

Abstract

Yoga is a form of exercise that is beneficial for our body and helps us to relax our body as well as mind. The demand for yoga in a person's life has multiplied as increased people switch to desk jobs. yoga has many mental advantages, but to experience them, one must perform the right asana's, which also benefit physical health. However, according to doctors, if individual will not perform yoga in the correct posture, then it will lead to problems like acute pain, back pain and sometimes cause severe injuries. However, People usually like to do yoga at their home and they always need an instructor to watch the practitioner and correct them by assessing the specific asana they are performing. Whether the posture is appropriate or not. So, this project will help several people to correct their yoga posture using an interactive display via a yoga instructor. This project will correct the posture of a person using Human pose estimation. Human pose estimation will contain the data for posture of body, object identification etc. The project is build using artificial intelligence and machine learning. This yoga AI trainer is developed on android as well as a web-based platform. In this, a person will perform specific yoga posture which he or she wants to perform and based upon the person's movements of joints, the AI trainer will identify the asana and tells the accuracy of the performed asana. Employing the posenet version of tensorflow.js on ml5.js, which enables us to exploit tensorflow.js to its maximum by using the web browser's built-in GPU, this notifies the practitioner whether the asana is accurate or not.

Keywords: PoseNet , p5.js, ml5.js, Artificial intelligence, Deep learning

Chapter 1

Introduction

The word yoga has been derived from the Sanskrit word “YUG” which means to join or to unite. Yoga is the union of the individual soul with the absolute or divine soul. Yoga also means the unification of physical, mental, intellectual, and spiritual aspects of human being. Yoga is the science of development of a person’s consciousness. yoga is a useful exercise yoga improves strength, balance, and flexibility. Yoga is also a one of the best ways for losing weight. Yoga helps you to focus, yoga reduces stress and anxiety. There is different yoga asana which are performed in diverse ways and have different advantages, one should perform yoga every day to remain fit and active as we know healthy body leads to a healthy mind. The main goal of this project based on the machine learning is to build a model who estimate the accuracy of the asana and identify the name of the asana. The libraries like p5.js and ml5.js are also used in the formation of the model. The p5.js library is used for the animation purpose and ml5.js library is used for collecting the data and feeding them into the posenet model. The pose net machine learning model is designed to analyze image and video and estimate the pose of a person in real time, it uses a convolutional neural network (CNN) to analyze the input data and output a set of key points that stand for the joints of the person's body, along with a confidence score for each key point.

Problem Statement

As we know performing yoga in correct way with proper knowledge is especially important as performing yoga in incorrect way and with lack of knowledge can lead to several problems such as muscular pain and soreness. Performing yoga by the guidance of yoga trainer and gym instructor bit expensive even though not affordable to everyone. So, we need a yoga trainer which is accessible and affordable to everyone. The following project proposes the development of a yoga posture coaching system using a web based interactive transfer learning technique. It aims to develop an AI based platform that aids individual in performing yoga in correct posture.

Objectives

1) A web-based solution accessible to all.

2) Sound based system for hands free experience.

3) 360-degree angle approach for user to use it at any angle.

4) Decreasing the dependency on an instructor.

Figure 1: Working Map of the Report

Chapter 2

Background

This project focused on pose estimation, which is the task of using an ML model to determine the pose of a person from an image or video with the help of key body joints, and poseNet to determine the posture of a person asana with the help of key points.PostNet will not recognize who is in the image or video but the algorithm will tell us about the key body joints.PostNet will return a pose object that contains the list of key points from just a webcam and it is trained to recognize 17 key points on the body, including head, neck, shoulders, elbows, wrists, hips, knees, and ankles. Here key points are nothing but human skeleton joints which resemble give us a human skeleton.

The project also focused on neural networks, also known as artificial neural networks (ANN) . These are a subset of machine learning and deep learning algorithms. Their names and structures are inspired by the human brain. A neural network works similarly to neurons. It is a specific mathematical function that collects and classifies information about a specific architecture. And in this, just collects and classifies information about the key joints and a stick figure representation of the body, which helps in predicting the body asanas of a person. A neutral network contains layers of interconnected nodes. Each node is called a multiple linear regression. There is a need for hardware components to create a neural network. In simple words, neural networks reflect the behaviour of the human brain and it allows computer programs to recognize patterns and help in solving common problems in the field of artificial intelligence and machine learning.

The libraries like p5.js library and ml5.js library is used. A p5.js library is a javascript tool that helps in doing fundamentals of coding and tells how to use objects and create your

classes. This library is also used for animation. It makes javascript code easy by providing us with some simpler functions as well as it integrates well.

ml5.js library is a javascript library used with the focus of having access to machine learning algorithms and models in browsers. It is inspired by processing and p5.js.With the help of these libraries, it is possible to make a tool for predicting the yoga postures of a person by collecting the data from a video or a photo and then feeding them into a pose Net model. It also has a wide collection of images, sounds, and text-based models. It provides us access to pre-trained models. We can also create our neural network with ml5.js.It helps us to improve and debug the neural network.

Since the domain of interest of this work, yoga is not much explored, only a few works found related to this inspect the use of pose net and pose estimation. These studies combine AI (Artificial Intelligence) and yoga training to prevent the injuries that took place during performing yoga asana. These studies make doing yoga a piece of cake for every person.

To design a self-training system Chen et al. [1] introduced yoga activity recognition using a feature-based approach. It uses a Microsoft motion sensor that is Kinect for capturing the body map of the user.

Hua-Tsung Chen proposes a yoga posture recognition system that can recognize the posture performed by the person. In the first step, he used the kinetic technique to get the body map of a specific person and in the next step, he used the star skeleton for fast skeletonization which connects the centroid to other joints of the body. In this system he got 96%accuracy.

Edwin W.trejo[2] introduced a model for the correction of yoga postures. In this, the user will get real-time instructions for pose correction by some experts. In this, they used a recognition algorithm called the AdaBoost algorithm. From this algorithm, they create a database for almost 6 asanas. They got almost 92% accuracy in doing this.

Mohanty et al. [3] have applied image recognition techniques for yoga posture by detecting them from images using a convolutional neural network (CNN). But there is a drawback that this works only for images not for videos.

AI-based yoga posture estimation was proposed by Girija et al. [4] which focused on posture estimation and pose net. This study was an android application that took the video of a person doing asana and using Postnet. They predict your yoga posture's correctness.

Another relevant paper is by Chinnaaiah et al. [5]. In this, pressure nodes are identified by force-sensitive resistors (FSR). The pressure data collected are utilised with an algorithm.

Manisha et al. [6] proposed a dataset called yoga -82 for the classification of human yoga postures and attained the top score for image classification, detecting human yoga postures and correcting them.

There is also a paper from Dongye Lv [7], in which he created a system that trained golf players to make a perfect swing by capturing them in a portable way. They used the Dynamic Bayesian network (DBN) model for capturing the person in the most accurate way. They used kinetic as a capturing device.

Nagalakshmi et al. [8] used 3D landmark points. This model includes KNN, SVM, and other deep learning models. Each model was made on the dataset that was based on a collection of 13 different postures.

Overall, there is a growing body of literature that exploring the use of machine learning and computer vision techniques for predicting yoga asanas.

This research takes a step ahead in the direction that there is a web-based solution for accessibility to all .it gives sound-based system for hands and free experience and gives a technique of 360° angle approach for users to use it at any angle. These researches have some promising results but still there is a need for further development in this era.

Chapter 3

Proposed Framework

In this research, we explored the use of artificial intelligence (AI) techniques to analyse live video feed of a person performing yoga asanas (Yoga asanas are physical postures that are used in the practice of yoga. They are designed to help improve flexibility, strength, and balance, as well as to reduce stress and improve overall physical and mental well-being.) and estimate the correctness of their poses. Our goal was to develop a web-based system that could provide real-time feedback on a person's form and alignment as they perform different yoga poses, with the aim of improving their practice and reducing the risk of injury.

To achieve this goal, we developed a machine learning model that was trained on a video dataset that we generated ourselves. The dataset consisted of video footage of people performing various yoga poses, which we used to train the model to recognize different poses and estimate their correctness.

Our web-based system allows users to input live video feed of themselves performing yoga poses, which is then analysed by the machine learning model and used to estimate the correctness of their poses. The system supplies real-time feedback on a person's form and alignment, allowing them to adjust and improve their practice in real time.

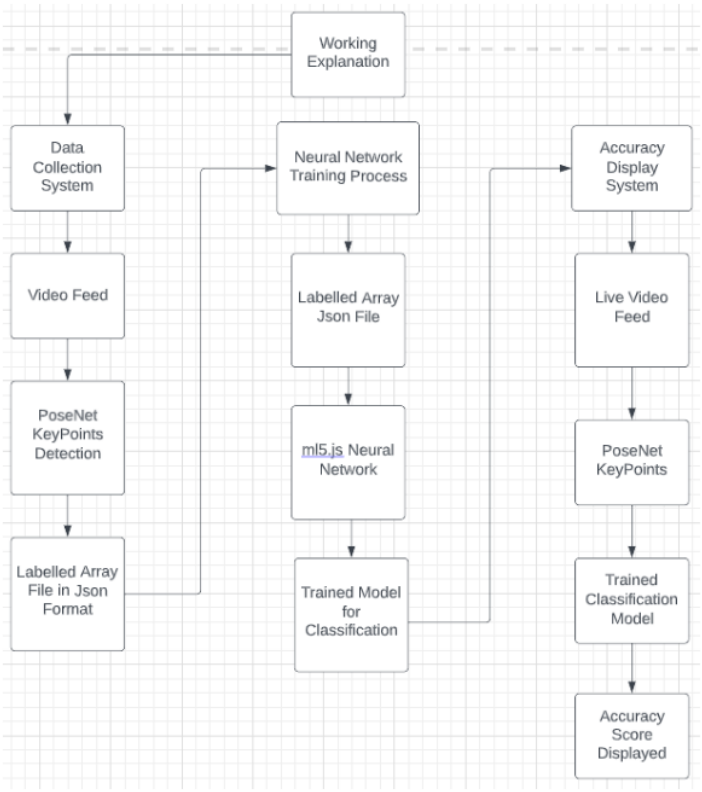

Figure 2: Working Process Explanation

In the following sections, we describe the details of our method, including the process we used to generate and curate our video dataset, the machine learning techniques we employed, and the challenges we faced and the solutions we implemented to address them. Our approach involved training the model on the video dataset, testing it on a separate set of video footage, and iteratively improving its performance through a process of fine-tuning and optimization. We believe that our web-based system has the potential to provide correct and reliable real-time feedback on a person's yoga practice, and we are excited to share the results of our research with the community.

To develop our web-based system for estimating the correctness of yoga poses in real-time, we decided to use the PoseNet machine learning model. As shown in Figure 3, the PoseNet machine learning model is designed to analyse images and video and estimate the pose of a person in real-time. It uses a convolutional neural network (CNN) to analyse the input data and output a set of key points that stand for the joints of the person's body, along with a confidence score for each KeyPoint. The key points can then be used to estimate the pose of the person, by connecting the key points with lines to form a skeleton.

We chose to use PoseNet with JavaScript to develop our web-based system for estimating the correctness of yoga poses in real-time, because it is a well-established model that has been shown to be effective in a variety of applications, and because using JavaScript allowed us to build a system that could be accessed from any device with a web browser.

In the following sections, we describe how we implemented PoseNet with JavaScript, and the steps we took to integrate it into our web-based system for estimating the correctness of yoga poses in real-time. We also discuss the challenges we faced and the solutions we implemented to address them, as well as the performance of our system in terms of accuracy and reliability.

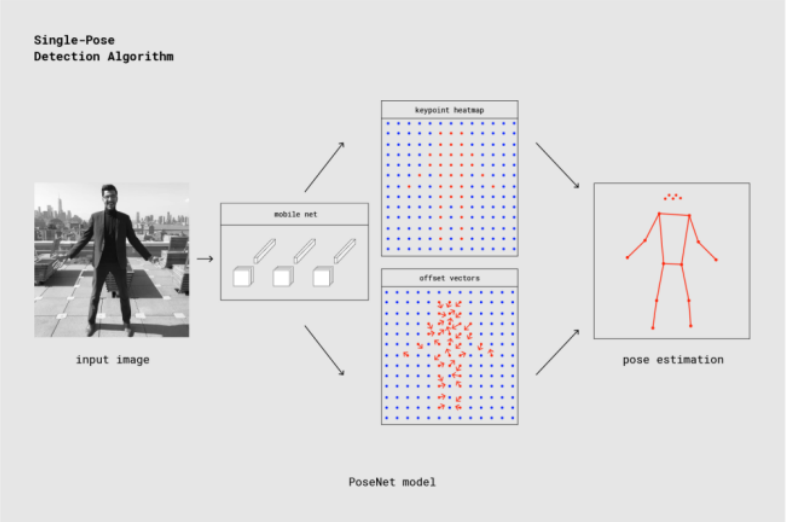

Figure 3: The PoseNet machine learning model for real-time pose estimation

To implement the PoseNet model with JavaScript, we used the ml5.js library. ml5.js is a JavaScript library that is built on top of TensorFlow.js, the JavaScript version of the popular TensorFlow library for machine learning in Python.

We chose to use ml5.js because it is a user-friendly library that makes it easy to use machine learning models in web projects. It provides a simple interface for working with TensorFlow.js, allowing developers to incorporate machine learning techniques into their projects without needing to be experts in machine learning.

In addition to providing access to the PoseNet model, ml5.js also includes several other machine learning models and utilities that can be used in web projects. We found it to be a useful and convenient tool for implementing our system, and we believe it has the potential to be useful in a wide range of other machine learning projects.

To use ml5.js in our web-based system for estimating the correctness of yoga poses in real-time, we followed the steps outlined in the library's documentation. This involved installing the library, loading the PoseNet model, and using the provided API to analyse live video feed and estimate the poses of the person in the video. We also implemented several additional features and optimizations to improve the performance and usability of our system.

In addition to using the ml5.js library to implement the PoseNet model, we also used the p5.js library to make our web-based system visually appealing and easy to use. p5.js is a JavaScript library that is designed for creating interactive graphics, animations, and other visual media.

It provides a wide range of tools and functions for drawing shapes, displaying images and video, and creating interactive elements such as buttons and sliders.

We used p5.js to create a user interface for our system that was visually appealing and easy to use. As shown in Figure 4, We included a live video feed of the person performing the yoga poses, and used p5.js to draw the key points and skeleton that were output by the PoseNet model on top of the video. This allowed the user to see what the model was detecting in real-time, and provided a visual indication of the correctness of their poses.

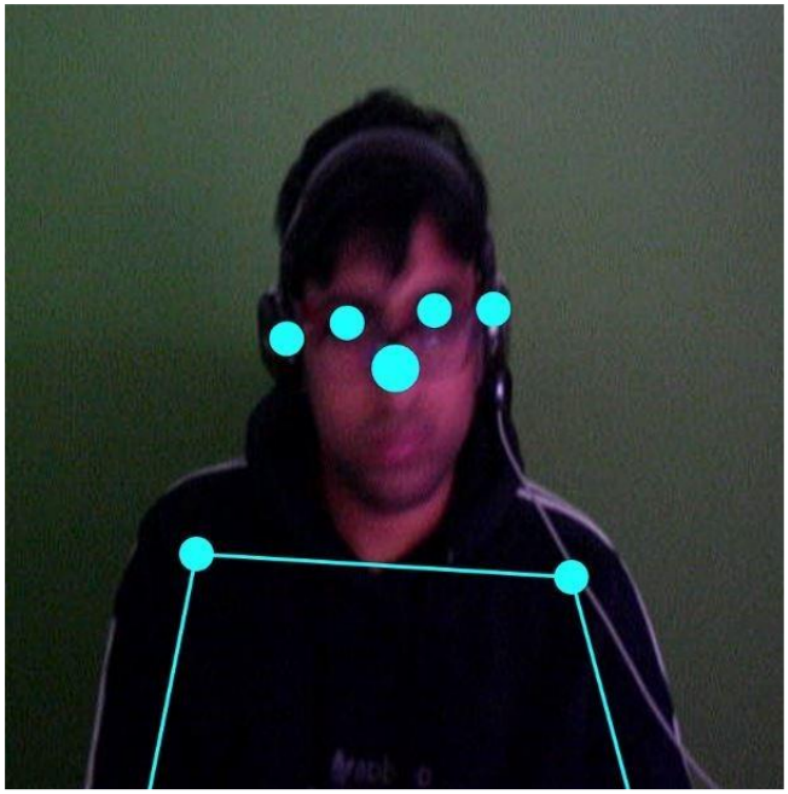

Figure 4: p5.js implemented to show detections made by PoseNet

Overall, we found p5.js to be a powerful and flexible tool for creating visually appealing and interactive web-based applications. It was an integral part of our system, and we believe it has the potential to be useful in a wide range of other projects.

Now before we move ahead, we need to understand that in p5.js library, the draw function and the setup function are two principal functions that are used to control the execution of a sketch, which is a program that uses the p5.js library to create interactive graphics, animations, and other visual media.

The setup function is a function that is called once when a sketch starts running. It is typically used to perform initialization tasks, such as creating canvas elements, setting up global variables, and loading external resources. The setup function does not return a value, and it does not take any arguments.

The draw function is a function that is called repeatedly by p5.js, at a rate determined by the framerate variable. It is typically used to update the display of a sketch, such as by drawing shapes, displaying images and video, and updating the positions of elements. The draw function does not return a value, and it does not take any arguments.

The setup function is called once when a sketch starts running, while the draw function is called repeatedly at a rate determined by the framerate variable. Both functions are important components of p5.js sketches, and are used to control the behaviour and appearance of the sketch.

Now as it is shown in Figure 5, to implement p5.js on PoseNet at first, we used 'createCanvas' and 'createCapture' command in the setup function both of which are part of the p5.js library, which is a JavaScript library for creating interactive graphics, animations, and other visual media. These functions are used to create elements in an HTML page that can be used to display visual content, such as images and video.

The 'createCanvas' function creates an element in the HTML page that can be used to draw 2D graphics using the p5.js library.

The 'createCapture' function creates an element in the HTML page that can be used to display a live video feed from a camera or other video source. It takes one argument: an object that specifies the options for the video capture, such as the video source (e.g., a webcam or file) and the dimensions of the video.

```
createCanvas(640, 480);
video = createCapture(VIDEO);
video.hide();
```

Figure 5: Live feed from the camera being generated

'video.hide()' has been used in Figure 5 because 'createCapture' creates a copy of the live feed outside of the canvas and here 'hide ()' has been used to hide the extra output on the webpage.

Now to feed the live video into the canvas we use the following code in the draw function of the code as shown in Figure 6.

```
image(video, 0, 0);
//background(200);

translate(video.width, 0);
scale(-1, 1);
image(video, 0, 0, video.width, video.height);
```

Figure 6: Live feed from the camera being displayed in the Canvas

In Figure 6, the code is from the draw function in the p5.js library, which is a function that is called repeatedly to draw the contents of a canvas element on a web page.

The image function is used to draw an image to the canvas. The first image function call draws the video to the canvas at the default position (0, 0). The translate function is used to move the canvas origin to the position (video.width, 0). The scale function is used to flip the canvas horizontally. The second image function call draws the video to the canvas at the position (0, 0), which is the same position as the first image function call, but since the canvas has been flipped horizontally, the video is drawn in reverse. This creates the effect of a mirror image of the video being displayed on the canvas.

The image function takes two or three arguments. The first argument is the image to be drawn, the second argument is the x-coordinate of the image, and the third argument is the y-coordinate of the image. The image function also has an optional fourth argument, which is the width of the image, and an optional fifth argument, which is the height of the image. If the width and height arguments are not provided, the image is drawn at its original size.

In this code, the video variable is the source of the image that is being drawn to the canvas. The video variable is a reference to a <video> element in the HTML document, which is being used to capture live video from a webcam or other video input device. The video.width and video.height properties are used to set the width and height of the canvas to match the dimensions of the video.

As shown in the Figure 7, we load the PoseNet model from the ml5.js library to be applied on the live video feed coming in on the canvas in the setup function

```
poseNet = ml5.poseNet(video, modelLoaded);
poseNet.on("pose", gotPoses);
```

Figure 7: PoseNet being loaded onto the live feed

The poseNet variable is an instance of the PoseNet model, which is a machine learning model that is used to perform pose estimation. The PoseNet model is created using the ml5.poseNet function, which takes a video element and a callback function as arguments. The modelLoaded function is the callback function that is called when the PoseNet model is ready to use.

The poseNet.on function is used to register an event listener for the "pose" event, which is triggered whenever the PoseNet model detects a pose in the video. The gotPoses function is the event listener that is called when the "pose" event is triggered.

But this would just load the PoseNet model to work on the live feed and to return the key points detected by it in the process now to show the detected key points and the skeleton formed using them we use the functionality of p5.js to draw onto the canvas in the draw function as shown in the Figure 8.

```
if (pose) {
  let eyeR = pose.rightEye;
  let eyeL = pose.leftEye;
  let d = dist(eyeR.x, eyeR.y, eyeL.x, eyeL.y);
  fill(0, 255, 255);
  ellipse(pose.nose.x, pose.nose.y, d / 2);
  fill(0, 255, 255);
  ellipse(pose.rightWrist.x, pose.rightWrist.y, 16);
  ellipse(pose.leftWrist.x, pose.leftWrist.y, 16);

  for (let i = 0; i < pose.keypoints.length; i++) {
    let x = pose.keypoints[i].position.x;
    let y = pose.keypoints[i].position.y;
    fill(0, 255, 255);
    ellipse(x, y, 16, 16);
  }

  for (let i = 0; i < skeleton.length; i++) {
    let a = skeleton[i][0];
    let b = skeleton[i][1];
    strokeWeight(2);
    stroke(0, 255, 255);
    line(a.position.x, a.position.y, b.position.x, b.position.y);
  }
```

Figure 8: Skeleton being drawn onto the detected key points by PoseNet

The code in Figure 8 is part of a function that is used to draw the keypoints and skeleton of a pose detected by the Posenet machine learning model to a canvas element on a web page.

The if (pose) statement checks if a pose has been detected by the Posenet model. If a pose has been detected, the code inside the if statement is executed. If a pose has not been detected, the code inside the if statement is skipped.

The pose variable is an object that contains information about the pose detected by the Posenet model. The pose object has several properties, including rightEye, leftEye, nose, rightWrist, leftWrist, and keypoints.

The rightEye and leftEye properties are objects that contain information about the position of the right and left eyes, respectively. The nose property is an object that contains information about the position of the nose. The rightWrist and leftWrist properties are objects that contain information about the position of the right and left wrists, respectively. The keypoints property is an array of objects that contains information about the position of all the keypoints detected by the Posenet model.

The skeleton variable is an array of arrays that contains information about the connections between the keypoints detected by the Posenet model. Each inner array contains two objects, which represent the keypoints that are connected by a line.

The dist function is used to calculate the distance between the right eye and the left eye. The distance is calculated using the Pythagorean theorem. The distance is then used to set the size of the ellipse that is drawn at the position of the nose.

The fill function is used to set the color of the shapes that are drawn to the canvas. The ellipse function is used to draw an ellipse to the canvas. The ellipse function takes four arguments: the x-coordinate and y-coordinate of the center of the ellipse, and the width and height of the ellipse.

The for loop is used to iterate over the keypoints array. For each keypoint in the array, an ellipse is drawn at the position

In our project, the process of estimating the accuracy of yoga poses performed by a user involved the following three systems:

1) Data collection system: This system used the PoseNet model to take an image as input and perform "pose estimation," or to try to identify key points on a human body in the image. These key points were represented as xy coordinates, and there was a total of 17 of them. The xy coordinates identified by PoseNet were then used as input to the ML5 machine learning model, which classified the pose based on the coordinates into a label representing a specific yoga asana. Additional asana labels could be created to classify the poses in the images.

2) Neural network training system: This system used the labelled data generated by the data collection system to train the neural network to recognize the relationships between the input data (xy coordinates of key points) and the correct poses (labels representing specific yoga asanas).

3) Results display system: This system used the PoseNet model to analyse live video feed of the user performing the poses, and used the trained neural network to make predictions about the correctness of the poses in real-time. The results were displayed to the user in a visually appealing and interactive way, using the p5.js library.

Overall, these three systems worked together to allow us to estimate the accuracy of yoga poses performed by a user in real-time, using a combination of machine learning techniques and interactive visualizations.

Data Collection System:

```
let options = {
  inputs: 34,
  outputs: 3,
  task: "classification",
  debug: true,
};
brain = ml5.neuralNetwork(options);
```

Figure 9: Neural Network model being generated

As shown in Figure 9 the code creates an object called options that contains several properties, and then uses this object to create a neural network using the ml5.neuralNetwork function.

The inputs property of the options object specifies the number of input neurons in the neural network. In this case, the value of inputs is 34, which means that the neural network has 34 input neurons.

The outputs property of the options object specifies the number of output neurons in the neural network. In this case, the value of outputs is 3, which means that the neural network has 3 output neurons.

The task property of the options object specifies the type of task that the neural network will be used for. In this case, the value of task is "classification", which means that the neural network will be used for classification tasks.

The debug property of the options object is a Boolean value that specifies whether to enable debugging output for the neural network. In this case, the value of debug is true, which means that debugging output will be enabled.

Finally, the ml5.neuralNetwork function is called with the options object as an argument, and the resulting neural network is stored in the brain variable. This creates a neural network with the specified number of input and output neurons, and sets it up for use in classification tasks.

```
function keyPressed() {
  if (key == "s") {
    brain.saveData("data");
  }
  if (key == "c") {
    targetLabel = p;
    console.log(targetLabel);
    setTimeout(function () {
      console.log("collecting");
      state = "collecting";
      setTimeout(function () {
        console.log("not collecting");
        state = "waiting";
      }, t2 * 1000);
    }, t1 * 1000);
  } else {
    console.log("Pressed the C key to collect data");
  }
}
```

Figure 10: KeyPressed function to collect data for Asana

The keyPressed function in the Figure 10 is a function that is called whenever a key is pressed on the keyboard. It contains two if statements that check the value of the key variable, which represents the key that was pressed.

If the value of key is "s", then the brain.saveData function is called, with the string "data" as an argument. This causes the data collected by the neural network to be saved to a file with the name "data".

If the value of key is "c", then the value of the targetLabel variable is set to the value of the p variable, and the value of the state variable is set to "collecting". This causes the neural network to start collecting data. After a certain amount of time (specified by the t1 variable), the value of the state variable is set to "waiting", which causes the neural network to stop collecting data.

If the value of key is anything other than "s" or "c", then a message is printed to the console indicating that the user should press the C key to collect data.

Overall, the keyPressed function in this code allows the user to control the data collection process by pressing keys on the keyboard, and provides a way to save the collected data to a file.

```
function gotPoses(poses) {
  //console.log(poses);

  if (poses.length > 0) {
    pose = poses[0].pose;
    skeleton = poses[0].skeleton;

    if (state == "collecting") {
      let inputs = [];
      for (let i = 0; i < pose.keypoints.length; i++) {
        let x = pose.keypoints[i].position.x;
        let y = pose.keypoints[i].position.y;
        //let score = pose.keyPoints[i].score;
        inputs.push(x);
        inputs.push(y);
        //inputs.push(score);
      }
      let target = [targetLabel];

      brain.addData(inputs, target);
    }
  }
}

function modelLoaded() {
  console.log("poseNet ready");
}
```

Figure 11: gotPoses and modelLoaded functions used in Figure 7

As shown in Figure 11 the gotPoses function is a function that is called whenever the PoseNet model detects a pose in the video. The poses argument is an array of objects that contain information about the poses detected in the video.

The function begins by checking if the poses array is non-empty, which indicates that at least one pose has been detected. If this is the case, the first element of the poses array is stored in the pose and skeleton variables, which contain information about the pose and the skeleton of the person in the pose.

The function then checks if the value of the state variable is "collecting". If this is the case, the function begins a loop that iterates over the keypoints property of the pose object. The keypoints property is an array that contains information about the keypoints (or points of interest) on the body of the person in the pose.

For each keypoint in the keypoints array, the function extracts the x and y coordinates of the keypoint, and pushes these coordinates onto the inputs array. The inputs array will eventually be used as input data for the neural network.

Finally, the function creates an array called target that contains the value of the targetLabel variable. The target array will be used as the output data for the neural network.

The modelLoaded function is a callback function that is called when the PoseNet model is ready to use. It simply logs a message to the console to indicate that the model is ready.

```
let p = prompt("Enter a name of the asan :- ");
let t1 = prompt(
  "Enter the amoutn of Time(sec) you want between pressing the key and starting collecting tha data :- "
);
let t2 = prompt(
  "Enter the amount of Time(sec) for which you want to collect data :- "
);

let state = "waiting";
```

Figure 12: User inputs needed in the system

In Figure 12 the p variable is a string that is assigned the value of the prompt function, which displays a dialog box that prompts the user to enter a name of an asana. The user's input is stored in the p variable.

The t1 variable is a number that is assigned the value of the prompt function, which displays a dialog box that prompts the user to enter the amount of time in seconds that they want to wait between pressing a key and starting to collect data. The user's input is stored in the t1 variable.

The t2 variable is a number that is assigned the value of the prompt function, which displays a dialog box that prompts the user to enter the amount of time in seconds for which they want to collect data. The user's input is stored in the t2 variable.

The state variable is a string that is assigned the value "waiting". The state variable is used to control the data collection process. When the value of the state variable is "waiting", the data collection process is not active. When the value of the state variable is "collecting", the data collection process is active. The state variable is used in the gotPoses function to determine whether to collect data for the neural network.

The values of the p, t1, and t2 variables are used in the keyPressed function, which is a function that is called whenever a key is pressed on the keyboard. The keyPressed function uses the values of the p, t1, and t2 variables to set the value of the targetLabel variable, which is used to label the data that is collected for the neural network. The keyPressed function also uses the values of the t1 and t2 variables to determine how long to wait before starting and stopping the data collection process.

Neural network training system:

```
let options = {
  inputs: 34,
  outputs: 3,
  task: "classification",
  debug: true,
};
brain = ml5.neuralNetwork(options);
brain.loadData("all_collected_poses.json", dataReady);
```

Figure 13: To load a labelled dataset from a file

In Figure 13 the let options block defines an object that contains the options for the neural network. The inputs property specifies the number of input nodes in the neural network, which is 34 in this case. The outputs property specifies the number of output nodes in the neural network, which is 3 in this case. The task property specifies the task that the neural network is being trained for, which is "classification" in this case. The debug property is set to true to enable debugging.

The brain variable is used to hold the neural network. The ml5.neuralNetwork function is used to create a new neural network object with the specified options.

The brain.loadData function is used to load a dataset of labelled poses from a file. The dataset is stored in a file called "all_collected_poses.json". The dataReady function is passed as an argument to the loadData function, and it is called when the data has been loaded and is ready to be used.

```
function dataReady() {
  brain.normalizeData();
  brain.train({ epochs: 500 }, finished);
}

function finished() {
  console.log("model trained");
  brain.save();
}
```

Figure 14: Saving the model by training on data

In Figure 14 the dataReady function is used to normalize the data in the dataset and then train the neural network on the data. The normalizeData function is used to normalize the data so that it falls within a specific range. The train function is used to train the neural network on the data. The epochs property specifies the number of training iterations that the neural network will go through. The finished function is passed as an argument to the train function, and it is called when the training is finished.

The finished function is used to save the trained neural network to a file. The save function is used to save the neural network to a file.

Results Display System:

```
let o = prompt("Please enter the asana you want help with :- ");
```

Figure 15: User input for asana taken

In Figure 15 the code is asking the user to input a name of a yoga asana for which they want to get help with. It does this by using the prompt () function, which displays a dialog box in the browser with a text input field and two buttons: "OK" and "Cancel". The user can enter the name of the asana in the text input field and then click "OK" to submit their input. The input value will be stored in the o variable. This input will be used later in the code to classify the pose and display the result to the user.

```
let options = {
  inputs: 34,
  outputs: 3,
  task: "classification",
  debug: true,
};
brain = ml5.neuralNetwork(options);
const modelInfo = {
  model: "500_epoch_all/model.json",
  metadata: "500_epoch_all/model_meta.json",
  weights: "500_epoch_all/model.weights.bin",
};
brain.load(modelInfo, brainLoaded);
//brain.loadData("daw.json", dataReady);
```

Figure 16: Loading the trained Neural Network

In Figure 16 let options block defines an object that contains the options for the neural network. The inputs property specifies the number of input nodes in the neural network, which is 34 in this case. The outputs property specifies the number of output nodes in the neural network, which is 3 in this case. The task property specifies the task that the neural network is being trained for, which is "classification" in this case. The debug property is set to true to enable debugging.

The brain variable is used to hold the neural network. The ml5.neuralNetwork function is used to create a new neural network object with the specified options.

The modelInfo object contains information about the location of the trained neural network model. The model property specifies the location of the model file, the metadata property specifies the location of the metadata file, and the weights property specifies the location of the weights file.

The brain.load function is used to load the trained neural network from the specified files. The modelInfo object is passed as an argument to the load function, and the brainLoaded function is passed as a callback function that is called when the neural network has finished loading.

```
function brainLoaded() {
  console.log("Pose Classification Ready!");
  classifyPose();
}

function classifyPose() {
  if (pose) {
    let inputs = [];
    for (let i = 0; i < pose.keypoints.length; i++) {
      let x = pose.keypoints[i].position.x;
      let y = pose.keypoints[i].position.y;
      //let score = pose.keyPoints[i].score;
      inputs.push(x);
      inputs.push(y);
      //inputs.push(score);
    }

    brain.classify(inputs, gotResult);
  } else {
    setTimeout(classifyPose, 100);
  }
}
```

Figure 17: Inputs feeded into the Trained model

classifyPose in Figure 17 is designed to classify the pose of a person based on the input of a set of keypoints. It first checks if there is a pose available, and if so, it creates an array called inputs to hold the keypoint positions. It then iterates through the keypoints and pushes the x and y positions into the inputs array.

After collecting all the keypoint positions, the function calls the classify method of the brain object with the inputs array and the gotResult function as arguments. The classify method of a neural network object is used to classify an input using the trained model. It takes an array of inputs and a callback function as arguments, and calls the callback function with the result

of the classification. In this case, the gotResult function will be called with the result of the classification of the pose based on the keypoint positions.

```
function gotResult(error, results) {
  selectedValue = o;
  //selectedValue='uttana shishoasana';
  //if(selectedValue != 0){}
  k = selectedValue;
  for (i = 0; i < 20; i++) {
    if (results[i].label == k) {
      selectedLabel = i;
    }
  }
  //}

  //k=selectedLabel;
  console.log(results[selectedLabel].confidence);

  // Set the value of the range element to the confidence value
  document.getElementById("myRange").value = results[selectedLabel].confidence;

  // Update the label with the confidence value
  document.getElementById("rangeLabel").innerHTML =
    results[selectedLabel].confidence;

  // Check if the confidence value is greater than 0.5
  if (results[selectedLabel].confidence > 0.5) {
    // Create a new Audio object
    var audio = new Audio();

    // Set the audio file to play
    audio.src = "ring_sound.mp3";

    // Set the volume to full (1)
    audio.volume = 1;

    // Play the audio file
    audio.play();
  }

  /*
  if (results[0].label == a) {
    a = results[0].label;
  } else {
    console.log(results);
    console.log(results[0].label);
    a = results[0].label;
  }
  */
  classifyPose();
}
```

Figure 18: Classification Result

In Figure 18 function that is used to process the results of a classification performed by a neural network. The function takes in two arguments: error and results. error is an error object that is returned if there was an issue with the classification, and results is an array of objects that represent the classification results.

The function starts by setting the value of a global variable selectedValue to a default value o. It then iterates through the results array and checks if any of the labels match a predefined value k. If a match is found, the function sets the value of selectedLabel to the index of the matching label in the results array.

The function then logs the confidence value for the label represented by selectedLabel to the console, and sets the value of a range element in the HTML document to that confidence value. It also updates the label for the range element to display the confidence value.

If the confidence value is greater than 0.5, the function creates a new Audio object and sets its source file to ring_sound.mp3. It then sets the volume to full and plays the audio file.

Finally, the function calls itself (classifyPose()) to continue classifying poses.

Chapter 4

Results

In this project we made a display system which used the PoseNet model to analyse live video feed of the user performing the poses, and used the trained neural network to make predictions about the correctness of the poses in real-time. The results were displayed to the user in a visually appealing and interactive way, using the p5.js library.

1) The system receives the input image and the corresponding pose estimation data from the pose estimation system.
2) The system uses the pose estimation data to calculate the accuracy of the pose performed by the user. This accuracy value can be calculated using a variety of different metrics, such as the distance between the detected keypoints and their corresponding ground truth positions, or the degree to which the detected pose matches the desired pose.
3) The system displays the input image and the calculated accuracy value to the user, using a visual interface such as a graph or a text label.

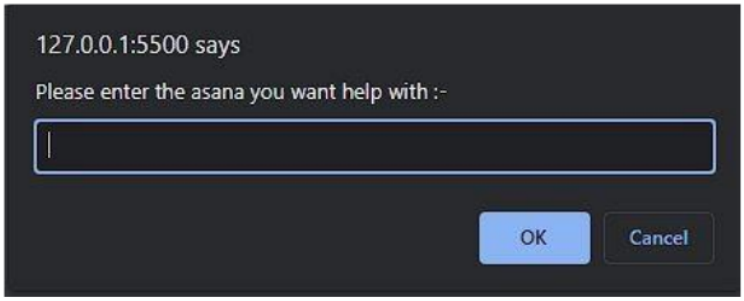

Figure 19: Input from User

In Figure 19 it shows the prompt message for the user where the user fills in the name of the Asana whose estimation the user wants with his/her Yoga Session after which the screen loads into the following as shown in Figure 20

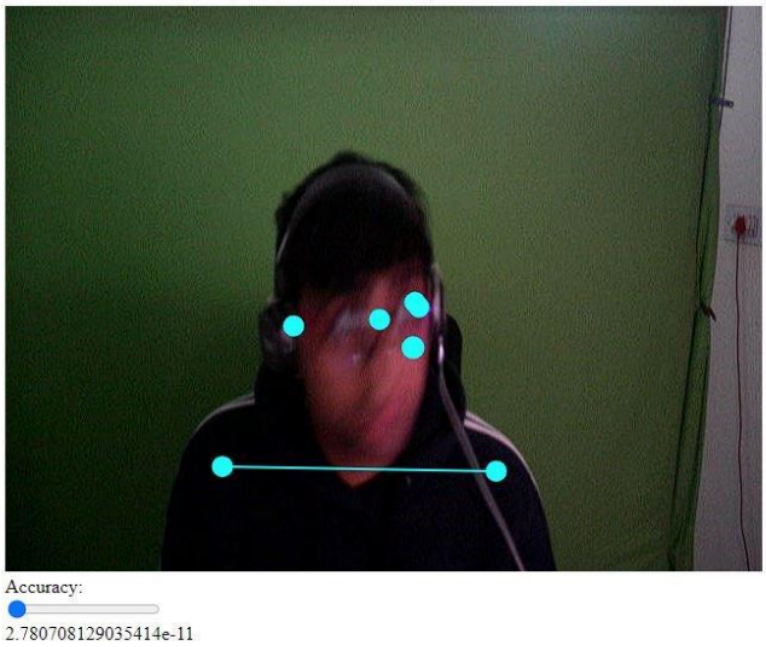

Figure 20: Program showing accuracy of the asana

The program also plays a ringing sound once the accuracy of the asana is more than 50% according to it and keeps on playing it until the user holds the asana to help the user in having a hands-free experience and user could get the feedback easily.

For Example, in the Figure 21 the user was initially asked to enter the asana with which the user wanted help and once the user filled in “vriksasana” the screen showed the accuracy as 90%

Figure 21: Accurate Prediction of Yoga Asana

Chapter 5

Conclusion and Future Scope

Nowadays, most of the people have become health conscious and are taking care of their health and fitness. It is important to be healthy physically as well as mentally. So, for that one of the best ways is doing yoga and asanas. Well doing asanas is not an easy task, even many people do it in the wrong way which can lead to problems like acute pain, back pain and sometimes cause severe injuries. People always need an instructor to correct them.

So, our study will help several people to know how much an individual is doing right using an interactive display. This helps to correct the posture of a person using Human pose estimation. Human pose estimation contains the data for the posture of the body, object identification, etc. This system is made using artificial intelligence and machine learning.

Our AI Yoga Asanas Estimator will help people to do yoga propesrly and to be fit physically and mentally.

Current shortcomings which could be overcome in future:

1. This will let the person know how much right or wrong the person is doing but it will not tell what the person is doing wrong.
2. This will not even tell the whole correct process to do any posture.
3. This is unable to tell how to correct the posture.

All these can be fixed in the future but it requires data in very detail, and for that, we would need a variety of people so that we can have as wide a range of data as possible, and then it would be more accurate.

References

1) Chen, HT., He, YZ. & Hsu, CC. Computer-assisted yoga training system. *Multimed Tools Appl* **77**, 23969–23991 (2018). https://doi.org/10.1007/s11042-018-5721-2

2) E. W. Trejo and P. Yuan, "Recognition of Yoga poses through an interactive system with Kinect based on confidence value," 2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM), 2018, pp. 606-611, doi: 10.1109/ICARM.2018.8610726.

3) Mohanty, A., Ahmed, A., Goswami, T., Das, A., Vaishnavi, P., Sahay, R.R. (2017). Robust Pose Recognition Using Deep Learning. In: Raman, B., Kumar, S., Roy, P., Sen, D. (eds) Proceedings of International Conference on Computer Vision and Image Processing. Advances in Intelligent Systems and Computing, vol 460. Springer, Singapore. https://doi.org/10.1007/978-981-10-2107-7_9

4) Chiddarwar, Girija Gireesh, Abhishek Ranjane, Mugdha Chindhe, Rachana Deodhar, and Palash Gangamwar. "AI-based yoga pose estimation for android application." Int J Inn Scien Res Tech 5 (2020): 1070-1073.

5) Anusha M, Dubey S, Raju PS, Pasha IA (2019) Real-time yoga activity with assistance of embedded based smart yoga mat. In: 2019 2nd International Conference on Innovations in Electronics, Signal Processing and Communication, https://doi.org/10.1109/IESPC.2019.8902371

6) Verma, M., Kumawat, S., Nakashima, Y., & Raman, S. (2020). Yoga-82: a new dataset for fine-grained classification of human poses. In Proceedings of the

IEEE/CVF conference on computer vision and pattern recognition workshops (pp. 1038-1039).

7) Dong-yue, L., Zhi-pei, H., Guan-hong, T., Neng-hai, Y., & Jian-kang, W. (2015). Dynamic Bayesian network model based golf swing 3D reconstruction using simple depth imaging device. 电子与信息学报 37(9), 2076-2081.

8) Nagalakshmi, Chirumamilla, and Snehasis Mukherjee. "Classification of Yoga Asanas from a Single Image by Learning the 3D View of Human Poses." Digital Techniques for Heritage Presentation and Preservation. Springer, Cham, 2021. 37-49.

Final Report Submitted

ORIGINALITY REPORT

9%	6%	5%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source	2%
2	www.coursehero.com Internet Source	2%
3	www.slideshare.net Internet Source	1%
4	ijarsct.co.in Internet Source	<1%
5	Youngpil Cha. "Satisfaction assessment of multi-objective schedules using neural fuzzy methodology", International Journal of Production Research, 1/20/2003 Publication	<1%
6	ntnuopen.ntnu.no Internet Source	<1%
7	Submitted to University of Huddersfield Student Paper	<1%
8	uwspace.uwaterloo.ca Internet Source	<1%

9	"Advanced Computing Technologies and Applications", Springer Science and Business Media LLC, 2020 Publication	<1%
10	Submitted to Open University Malaysia Student Paper	<1%
11	"Proceedings of Fifth International Congress on Information and Communication Technology", Springer Science and Business Media LLC, 2021 Publication	<1%
12	Carla Delmarre, Marie-Anne Resmond, Frédéric Kuznik, Christian Obrecht, Bao Chen, Kévyn Johannes. "Artificial Neural Network Simulation of Energetic Performance for Sorption Thermal Energy Storage Reactors", Energies, 2021 Publication	<1%
13	Submitted to Colorado State University, Global Campus Student Paper	<1%
14	hubpages.com Internet Source	<1%
15	personalhealthcare.info Internet Source	<1%

16	Shakti Kinger, Abhishek Desai, Sarvarth Patil, Hrishikesh Sinalkar, Nachiket Deore. "Deep Learning Based Yoga Pose Classification", 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON), 2022 Publication	<1 %
17	Submitted to University of Southern Queensland Student Paper	<1 %
18	Wu Liu, Qian Bao, Yu Sun, Tao Mei. "Recent Advances of Monocular 2D and 3D Human Pose Estimation: A Deep Learning Perspective", ACM Computing Surveys, 2022 Publication	<1 %
19	www.ijirset.com Internet Source	<1 %
20	"Digital Techniques for Heritage Presentation and Preservation", Springer Science and Business Media LLC, 2021 Publication	<1 %
21	Chhaihuoy Long, Eunhye Jo, Yunyoung Nam. "Development of a yoga posture coaching system using an interactive display based on transfer learning", The Journal of Supercomputing, 2021 Publication	<1 %

22	P. M. Trivailo. "The inverse determination of aerodynamic loading from structural response data using neural networks", Inverse Problems in Science and Engineering, 6/1/2006 Publication	<1 %
23	Santosh Kumar Yadav, Amitojdeep Singh, Abhishek Gupta, Jagdish Lal Raheja. "Real-time Yoga recognition using deep learning", Neural Computing and Applications, 2019 Publication	<1 %

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		