Virtual Yoga Trainer with PoseNet API

Harshraj Mahesh (harshraj.mahesh@sjsu.edu)
Jojo Joseph (jojo.joseph@sjsu.edu)
Shivangi Jain (shivangi.jain@sjsu.edu)
Vanditt Sama (Vanditt.sama@sjsu.edu)

Computer Engineering Department, San Jose State University
San Jose, CA 95192 USA

***Abstract*— This paper covers our findings and learnings while building a Web-app and methodologies associated with it. This project leverages PoseNet API released by Google Creative Labs for Pose detection which requires a combination of computer vision and machine learning techniques to detect human poses in images or video. This paper discusses the development and training of the machine learning model for yoga poses and different components of an application on which the model is deployed. The application is built upon React.js library to maintain consistency and modularity between various application components and utilizes bootstrap for UI elements. Further discussed is connection with Firebase API for authentication and process of storing application/user data into Cloud Firestore.**

***Keywords*— Posenet, yoga-pose, workout, progress-tracker, TensorFlow, React.js, Authentication, Web-app, Machine Learning (ML), Firebase, Application programming Interface (API), Dataset, Matching**

I. Introduction

This is the final report submission of our team project completed during Fall-2019 semester in class of Enterprise software platforms (CMPE-272) at San Jose State University as graduate students.

The core functionality uses a front camera or webcam on a device to detect users pose and matches it to a machine learning model trained using TensorFlow to determine correctness of a yoga pose. The app built around this idea enhances this functionality by providing several yoga poses along with it's advantages and images to guide users during a yoga workout session. Our aim is to provide a complete guided and automated workout routine for the users along with a progress tracker.

II. Preparing Machine Learning Model

A. Pose Estimation

Pose estimation is a technique of detecting positions of body parts and joints in an image or a real time video and combining the data to predict the posture of the user. The main advantage of this API is that it does not require any additional or bespoke hardware such as infrared camera, X-Ray or Ultrasonic imagery. It is because the image processing takes place locally on a web browser, which is why it is also able to perform real-time analysis while also keeping the user data private.

Fig 2.1

The above figure (2.1) depicts how PoseNet represents the identified key parts and joints in a body. For each point, the output returned is the X and Y coordinates of the image that has been detected as a key point and a measure of how confidence the API is about this estimation. The following block is an example of high level output generated by the API.

```
{
  "score": 0.32371445304906,
  "keypoints": [
    { // nose
      "position": {
        "x": 301.42237830162,
        "y": 177.69162777066
      },
      "score": 0.99799561500549
    },
    { // left eye
      "position": {
        "x": 326.05302262306,
        "y": 122.9596464932
      },
      "score": 0.99766051769257
    },
    { // right eye
      "position": {
        "x": 258.72196650505,
        "y": 127.51624706388
      },
      "score": 0.99926537275314
    },
    ...
  ]
}
```

Fig 2.2

To use, Once the library is importing using <npm install @tensorflow-models/posenet>, a simple HTML code snippet will be able to integrate the API into the web browser.

```
<html>
  <body>
    <!-- Load TensorFlow.js -->
    <script src="https://unpkg.com/@tensorflow/tfjs"></script>
    <!-- Load Posenet -->
    <script src="https://unpkg.com/@tensorflow-models/posenet">
    </script>
    <script type="text/javascript">
      posenet.load().then(function(net) {
        // posenet model loaded
      });
    </script>
  </body>
</html>
```

B. PoseNet matching technique

To achieve pose matching and return data from fig 2.2 is returned in the form of image, Weighted matching technique has been utilized. First, a cosine similarity is calculated between the PoseNet data from the training image and the received image.

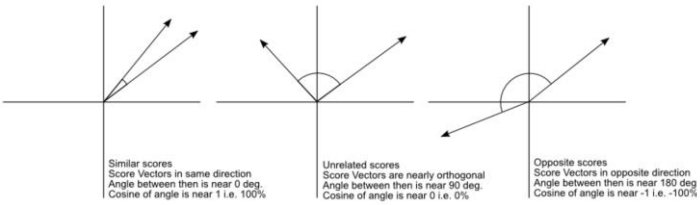

Fig 2.3

In the above figure, if the score vectors are similar, the model can assume that the key points between two images are similar. Following is the high level representation of the comparison operation.

$$D(Fxy, Gxy) = \sqrt{2 * (1 - cosineSimilarity(Fxy, Gxy))}$$

But, the confidence scores for a few key point might be low, this happens because the image set is 2-dimensional and the user could be in an irregular posture.

By not giving equal weight to the data with low confidence, the accuracy can be greatly improved. Google researchers George Papandreou and Tyler Zhu have derived a formula which incorporates this idea:

$$D(F,G) = \underbrace{\frac{1}{\sum_{k=1}^{17} Fc_k}}_{\text{Summation 1}} \times \underbrace{\sum_{k=1}^{17} Fc_k ||Fxy_k - Gxy_k||}_{\text{Summation 2}}$$

In the formula above, F and G are two vectors from the PoseNet data. Fck is the confidence score of the k^{th} keypoint of F. F_{xy} and G_{xy} represent the x and y positions of the k^{th} keypoint for each vector.

III. Developing a React Application

The goal of the web application is to provide the user an interactive way to practice yoga poses and improve the correctness of the posture to make workouts more efficient. On top of the PoseNet front, the app also allows user to keep track of their workout progress by signing in, provides information about different yoga poses along with posture images. An architecture diagram of the application summarizes how the PoseNet model was trained and the interaction with google cloud for user authentication and storage.

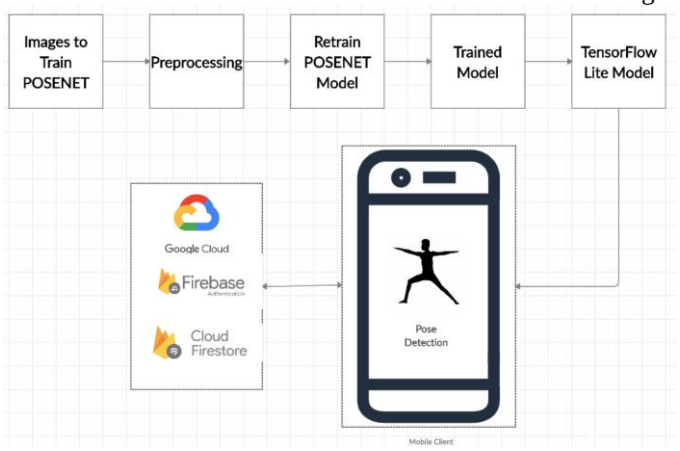

Fig 3.1

Following sections mention the technologies and APIs used and the procedure to implement the same.

A. Front-End

An intuitive front end allows the app to be more user friendly and therefore user interface was designed keeping the focus on content as a priority. React.js was used for front end development as well as client side scripting. React act is an open sourced JavaScript library (Maintained by Facebook) for designing interactive front end for applications. The react-dom provides the ability to only re-render the page content that has changed thus allowing dynamic pages to load faster. Use of React library allowed the application to have multiple components with independent state. Each component is solely responsible to provide it's own functionality hence making the project development process more modular and easy to integrate.

Reactstrap library provides self-contained components which can be imported and added to the React.js code. Various UI elements like buttons, cards, and layout have been designed using reactstrap components.

B. User Authentication using Google Firebase

To personalize user experience and store specific user data, Firebase Authentication provides SDKs and ready-made libraries to implement user authentication into an application. Choosing Firebase for authentication as well as Cloud Firestore as a no SQL database option was a voluntary decision since these services provide native integration with each other. When the user enters an application for the first time, a prompt is generated to allow the user to log in using email or use google

sign in. Once the login is successful, the existing user data is fetched from the Cloud Firestore. If no such data exists, new user data is added to the database.

Certain configuration is required in the react app to ensure successful use of Firebase API before it can be imported and used. Below code snippet describes the config parameters for this implementation.

```
import firebase from 'firebase';
import 'firebase/firestore';

var firebaseConfig = {
    apiKey: "AIzaSyB_6t3K[redacted]_g409uqU35fyWr[redacted]",
    authDomain: "virtualtra[redacted].firebaseapp.com",
    databaseURL: "https://virtualtra[redacted]irebaseio.com",
    projectId: "virtualtra[redacted]",
    storageBucket: "virtualtrai[redacted]",
    messagingSenderId: "1010[redacted]",
    appId: "1:10106744[redacted]1fc92c0f[redacted]",
    measurementId: "G-QSNC[redacted]"
  };

  const fire = firebase.initializeApp(firebaseConfig)
  export default fire;
```

Fig 3.2

The firebase.auth() object can be used to determine if the user is signed in and the user data like email, name etc are stored in the user object.

C. Cloud Firestore

This is a NoSQL cloud storage database which stores data as ‘collection’ of ‘documents’. The documents have a hierarchical structure which allows data storage to be flexible: Data can be stored as values which are mapped to respective fields, or it can contain an object which itself is a collection of similar data. Data can be retrieved at a document level which finds it’s use case while retrieving single user’s data. Or, the complete list of documents can be retrieved to perform operations on the data. An example of such an operation is when information about all yoga poses require to be fetched and displayed to the user.

In the virtual yoga trainer application, two different types of collections have been designed namely ‘poses’ and ‘userData’. The templates of document schema in below images denotes the raw structure of a document belonging to either of the collections along with their descriptions:


```
{
  "poseName": "Pose1",
  "sanskritName": "SanskritPose1",
  "benefits": {
    "0": "Benefit1",
    "1": "Benefit2",
    "2": "Benefir3"
  },
  "desc": "About Pose",
  "imgSrc": "http...",
  "longDesc": {
    "0": "Text1",
    "1": "Text2"
  },
  "difficulty": 2
}
```


Fig 3.3

Figure 3.3 describes the documents in ‘poses’ collection in the Cloud Firestore database. The document ID is generated automatically by the API when a new document is created. Only database administrators can add new documents to the ‘poses’ collection whenever a new pose is added to the application.


```
{
  "email": "john@gmail.com",
  "name": "John",
  "poseHistory": {
    "pose1": "2012-04-23T18:32:54.275Z",
    "pose2": "2013-05-23T19:41:23.567Z",
    "pose3": "2014-06-23T20:14:42.324Z"
  },
  "lastLogin": "2015-06-23T21:15:43.352Z",
  "userImg": "http...",
  "posesCompleted": {
    "0": "Pose1",
    "1": "Pose3"
  },
  "totalUsage": 2548
}
```


Fig 3.4

Figure 3.4 describes the documents in ‘userData’ collection in the Cloud Firestore database. The document ID is generated automatically by the API while inserting new documents into the collection. The email field for each document is unique and new documents are only inserted into the collection after a checking that the user data for a given email address doesn’t already exist. The virtual trainer app uses the Firebase API to add a new user data record whenever a new user signs in. For returning users, this collection fetches a single document to display users progress and other parameters.

IV. Results And Conclusions

This project has been a very important learning experience for all the team members and has helped them get familiar with various open source technologies and Machine Learning. The PoseNet model was successfully integrated with the React application and web app was hosted using the Heroku Cloud Platform. The project source code along supporting files and documentation is available at GitHub: https://github.com/SJSUFall2019-CMPE272/Virtual-Yoga-Trainer

URL for live implementation of the project: http://yogatrainer1.herokuapp.com/dashboard

Acknowledgment

The project team collectively wants to thank our mentor and instructor, Prof. Rakesh Ranjan, Innovation Leader, Director, Emerging Technologies at IBM Data & AI, for directing our efforts in this project for selection of technology stack, market research, design thinking process, development and

deployment of Virtual Yoga Trainer. His guidance, motivation and industry knowledge has been pivotal in making this project a success. We would also like to express gratitude to everyone who directly or indirectly contributed and the Computer Engineering Department at San Jose State University for allowing us the opportunity to work on this project. Lastly we acknowledge with pride that all the group members are pleased with the cohesion, team-work and dependability they exhibited throughout the duration of this assignment.

References

[1] https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5
[2] https://medium.com/tensorflow/move-mirror-an-ai-experiment-with-pose-estimation-in-the-browser-using-tensorflow-js-2f7b769f9b23
[3] https://ai.google/research/pubs/pub47173
[4] https://fribbels.github.io/vptree/writeup
[5] https://reactjs.org/docs/react-dom.html