Yoga Pose Detection

Yoga Pose Detection is based on PoseNet, ml5.js and KNN.

# K- Nearest Neighbours (KNN)

K-NN is a supervised machine learning algorithm mostly used for classification. It classifies a data point based on how its neighbours are classified.

KNN is based on feature similarity, so we can do classification using KNN classifier.

K in KNN is a parameter that refers to the number of nearest neighbours in the majority voting process.

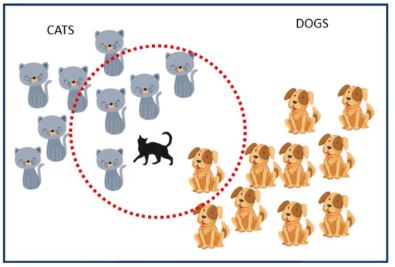
To choose K:

1. Sqrt(n), where n is total number of data points
2. Odd value of K is selected for tie-breaker.

Example

Suppose there is a dataset that contains information regarding cats and dogs. There is a new data point and you need to check if that sample data point is a cat or dog. To do this, you need to list the different features of cats and dogs.

Now, let us consider two features: claw sharpness and ear length.



Here, we have taken K=5. The majority votes from its five nearest neighbours and classifies the data point. The animal will be classified as cat because 4/5 neighbours are cats.

Working of KNN

Step 1 – When implementing an algorithm, you will always need a data set. So, you start by loading the training and the test data.

Step 2 – Choose the nearest data points (the value of K). K can be any integer.

Step 3 – Do the following, for each test data –

3.1 – Use Euclidean distance, Hamming, or Manhattan to calculate the distance between test data and each row of training. The Euclidean method is the most used when calculating distance.

3.2 – Sort data set in ascending order based on the distance value.

3.3 – From the sorted array, choose the top K rows.

3.4 – Based on the most appearing class of these rows, it will assign a class to the test point.

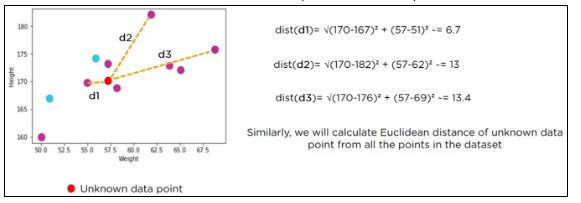
Example

Consider a dataset that contains two variables: height (cm) & weight (kg). Each point is classified as normal or underweight. Based on the above data, we need to classify whether a data point is normal or underweight using the KNN algorithm.

So if our datapoint is (57kg, 170cm)



We calculate the Euclidean distance of the unknown data point with our dataset.





Looking at the table and say K = 3, we can say that (57kg,170cm) should be normal because majority of its neighbours are classified as normal.

# PoseNet

PoseNet is a machine learning model that allows for real-time Human Pose Estimation.

PoseNet can be used to estimate either a single pose or multiple poses, which means that the model can be used to detect only one person in an image/video and can also be used to detect multiple persons in an image/video.

With PoseNet running on [TensorFlow.js](https://js.tensorflow.org/) anyone with a decent webcam-equipped desktop or phone can use this technology right from within a web browser.

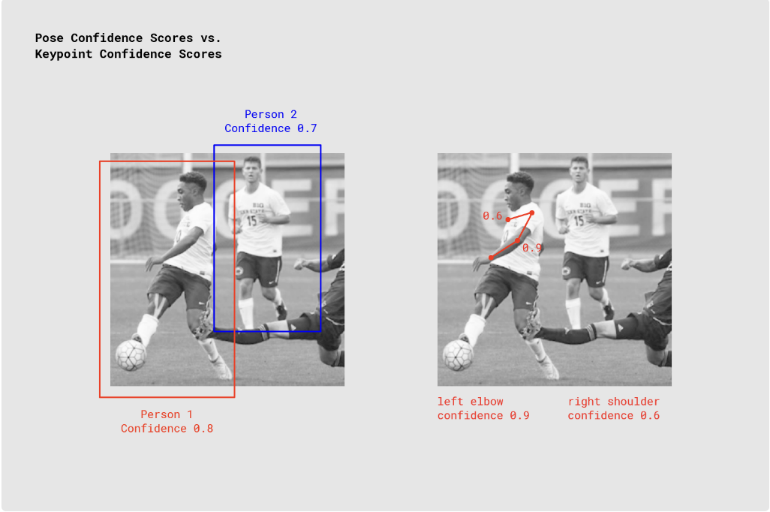
**What is pose-estimation?**

Pose estimation refers to computer vision technique that detect human figures in images and video, so that one could determine, for example, where someone’s elbow (or knee) shows up in an image.

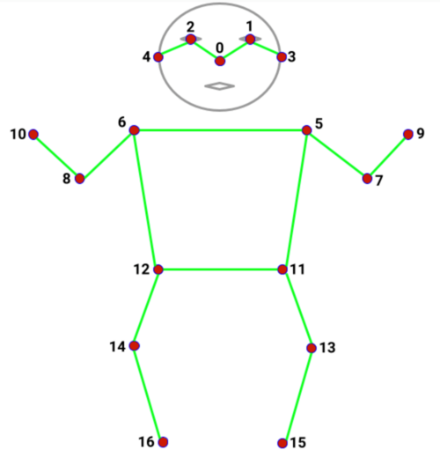
Pose estimation happens in two phases:

1. An input RGB image is fed through a convolutional neural network.

2. A single-pose or multi-pose decoding algorithm is used to decode poses, pose confidence scores, keypoint positions, and keypoint confidence scores from the model outputs.



For each pose, the model returns a list which contains a confidence score of the pose and an array of keypoints. Each pose contains 17 keypoints. The keypoints are (x,y) co-ordinate values. Using these keypoints, we can determine angles of different limb of our body or can use these points in classifier model for human activity detection.



# ml5.js

Built on top of TensorFlow.js, the ml5.js library provides access to machine learning algorithms and models in the web browser with no other external dependency.

ml5.js provides immediate access in the browser to pre-trained models for detecting human poses, generating text, styling an image with another, composing music, pitch detection etc.

# Steps to run the repository locally

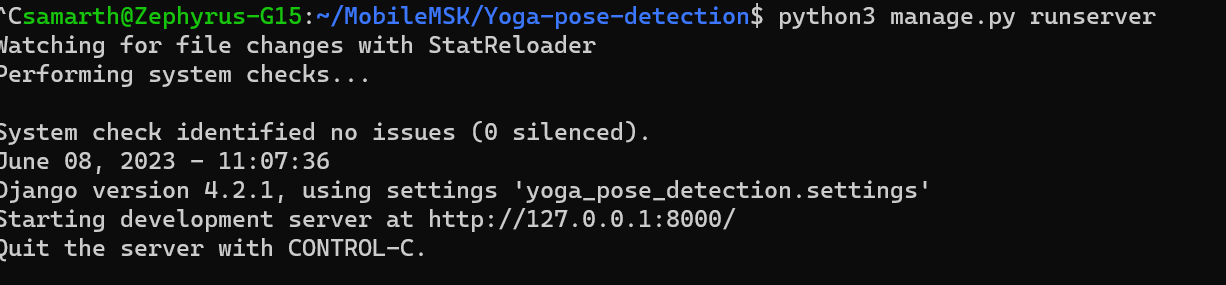
Fork the original repository

git clone <https://github.com/LegendTime1/Yoga-pose-detection.git>

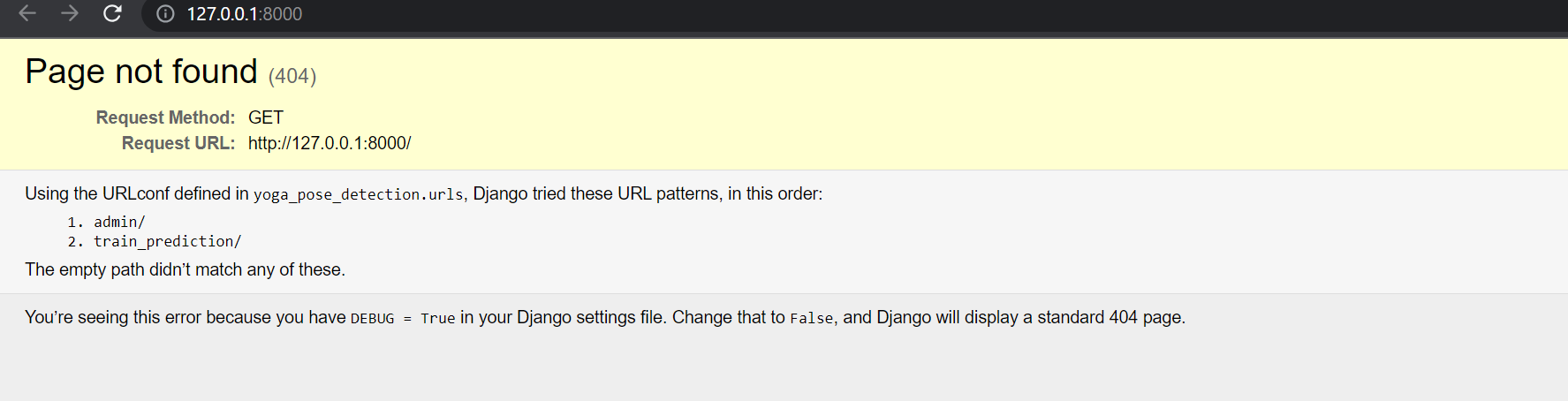
Ensuring system dependencies:

* Python >= 3.6
* pip install numpy
* pip install opencv-python
* pip install Pillow==2.2.2
* pip install django

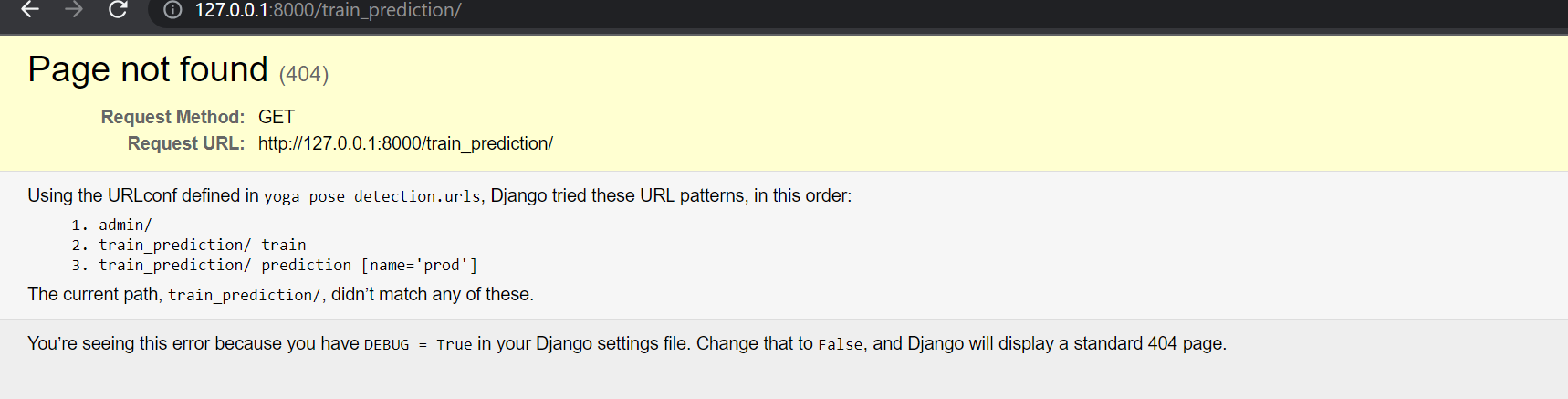
python3 manage.py runserver



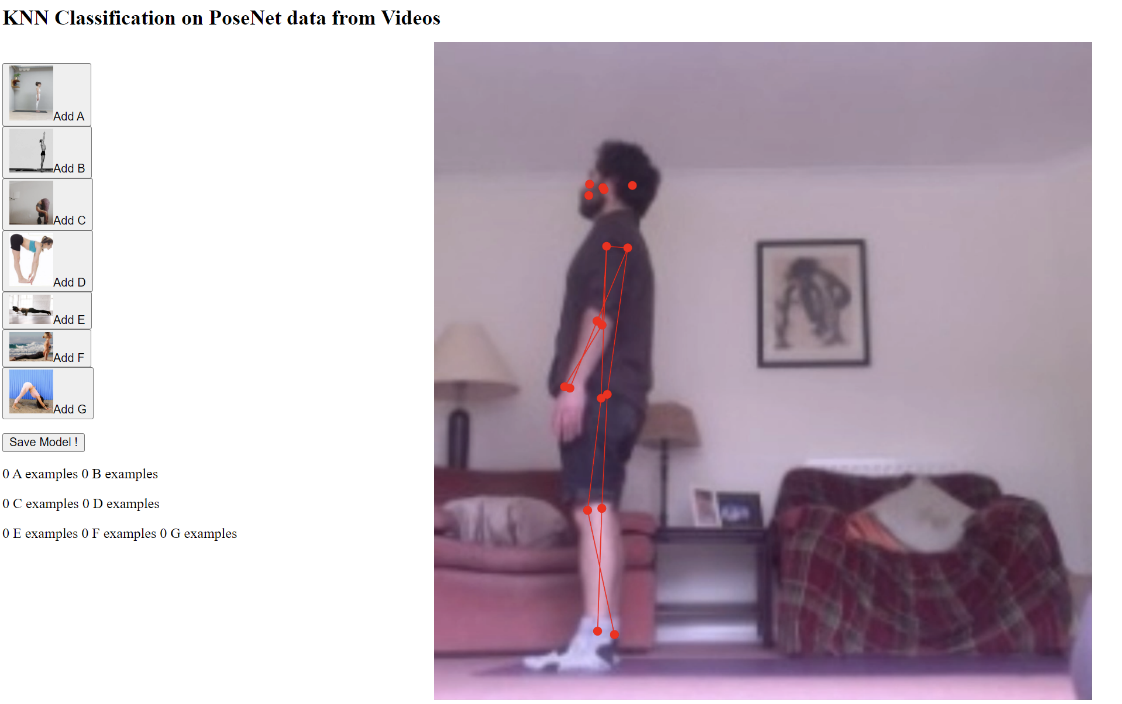
Going on to the url



Changing the url to http://127.0.0.1:8000/train\_prediction/



And going to <http://127.0.0.1:8000/train_prediction/train>



The model can be saved by clicking on save model! button

For prediction, go to <http://127.0.0.1:8000/train_prediction/prediction>

It will access webcam and start predicting yoga asanas.

# Explanation

Resize.py: Python script for resizing images using OpenCV. It reads images from the src\_path directory, resizes them to a size of 750x750 pixels using a function called resize\_image(), and saves the resized images to the dst\_path directory.

Video.py: Python script for generating a video from the resized images.

Pose.js (pred) contains all the backend logic for pose estimation and training. The code is using the K-Nearest Neighbors (KNN) classification algorithm for pose prediction. It makes use of the ml5.js library, specifically the ml5.KNNClassifier() class, to create and train the KNN classifier model. The model is trained using labeled examples of poses, where each pose is associated with a label (e.g., 'A', 'B', 'C', etc.). The classifier learns the patterns and relationships between the pose keypoints and their corresponding labels during the training process. Once the model is trained, it can classify new poses by extracting the pose keypoints from the detected pose and passing them to the knnClassifier.classify() function. The classifier predicts the label of the pose based on its nearest neighbors in the training data. The confidence scores for each label are obtained from the classification result using the result.confidencesByLabel property. The label with the highest confidence score is considered the predicted pose label.

The trained model is loaded using knnClassifier.load(pred\_json, classifierReady), where pred\_json is the JSON data containing the trained model's information.

A detailed explanation is added as comments in the pose.js file.

# References

https://blog.tensorflow.org/2018/05/real-time-human-pose-estimation-in.html

https://github.com/tensorflow/tfjs-models/tree/master/pose-detection

https://github.com/tensorflow/tfjs-models/tree/master/posenet

https://github.com/tensorflow/tfjs-models/tree/master/pose-detection/src/posenet

https://learn.ml5js.org/#/reference/posenet