University of Hyderabad & AppliedRoots

Online Diploma in Artificial Intelligence & Machine Learning

Capstone Project - Virtual Fitness Assistant

By
Prince Kumar Singh

Under Guidance of
N. Brahma Reddy

Acknowledgment

This project report is the result of work done as part of the capstone project for the course of Diploma in Artificial Intelligence and Machine Learning - by the University of Hyderabad and AppliedRoots. I am extremely grateful to my project mentor N. Brahma Reddy Sir, who is the co-founder and a very distinguished professional himself. His thorough guidance and valuable feedback at every stage of the project were of immense value and kept us motivated and on track. He was very kind and patient to hear out the issues we faced and without his co-operation, this wouldn't have been possible.

I am also thankful to our chief instructor, Srikanth Varma Chekuri Sir, who was the guiding light from the very beginning. He is no less than an inspiration himself and is the source of all our knowledge and motivation.

I would also like to thank all the teachers and staff of the University of Hyderabad and AppliedRoots, who were behind driving this course through their work and support.

Finally, I would also like to thank my friends and coursemates who helped me from time to time with their encouragement and ideas.

Index

Chapter 1

1.1 Motivation

“Health is Wealth” - is one of the most trivial sayings we learn at school during our childhood, however as we become adults, we get so engrossed in our daily work that we hardly take time to follow any proper exercise routine unless explicitly mentioned by our physician. Sometimes we content ourselves by simply walking, or doing some rigorous household activities. Although the situation has changed since the last decade or so, with the advancement in social media, there has been more awareness about physical training being a part of a normal healthy lifestyle rather than being considered as something done by people from sports or movies. Public gyms have come up, especially in cities and fitness is a topic being greatly discussed and promoted by companies. Having said that, a good chunk of society, who are aware and willing to follow an exercise regime still tend to be hesitant because of the lack of mentors who can guide them efficiently and are affordable and available as per their convenience. As per a report from a reputed news website[1], the cost of a personal trainer, in India, can range from Rs 750 for beginners to about Rs 2,000 an hour for specialized cases. Average being Rs 1,000-1,200 per hour. For someone willing to devote around 1 hour for 5 days a week, the monthly cost can be around 20,000 - 24,000, which is a good amount of money considering the Indian market. So, most people go to public gyms, however, they don't get personalized training as per their body needs and then they lose motivation and finally give up due to lack of progress. Also, in a gym, it is not practical for the instructor to personally monitor each member throughout the session. So we are trying to build a smart system that could help.

With the advent of the Covid-19 pandemic, a whole lot of business sectors were hit very badly. Due to gyms being closed down following the covid-19 norms and protocols, the fitness industry suffered a massive setback. However, as the saying goes - “*along with challenges, come opportunities*”, those who could leverage this time with rapid transformation and digitization, changing their business models along the new normal not only survived but could also become more competitive and trendsetters in their respective fields.

1.2 Introduction

The concept of “*going virtual*” has now come into the fitness industry - like in various others - teaching, social meetups, etc. However, it does have its challenges. It requires some level of technical expertise from the user’s side. Both the trainers and the trainees have to get accustomed to connecting and training over the internet. Also, from the trainer’s side, it is not easy to check and correct everyone over the video, as it is difficult to manually track all the trainees at the same time.

Keeping in mind the above discussion we can say that computer vision and AI are inevitable in the fitness industry now. As smartphones are getting cheaper and more powerful, and breakthroughs in IoT are paving the way for cheap sensors and smart wearables, it is adding more weight to this hypothesis. So, we are trying to build a product that can work to assist any gym instructor. This cannot replace the instructor but can make their work more efficient and organized. The idea is to digitize the routine part of the work like observing poses, keeping counts, making slight corrections, etc, and thus enable the instructor to cater to more people. The user can practice in front of the system and monitor themselves in their own sweet time.

Chapter 2

2.1 Business Case

We are trying to build an application as discussed in the previous section. We try to cover the following exercises:

- Jumping Jacks: "*A jumping jack, also known as a star jump and called a side-straddle hop in the US military, is a physical jumping exercise performed by jumping to a position with the legs spread wide and the hands going overhead, sometimes in a clap, and then returning to a position with the feet together and the arms at the sides.*" - Wikipedia

  Jumping Jacks engage and work muscles like - glutes, quadriceps, hip flexors. They also involve the abdominal and shoulder muscles. Jumping jacks are a kind of plyometric exercise - which works by rapidly stretching and contracting the muscles in quick succession. Some of the health benefits associated with this exercise are - weight management, reduced blood pressure, decreased "bad" fat, increased "good" fat, and maintained balanced sugar levels. Care must be taken while doing this exercise especially if you have any joint issues or muscle injury.

- Crunches: "*The crunch is one of the most popular abdominal exercises. When performed properly, it engages all the abdominal muscles but primarily it works the rectus abdominis muscle and the obliques. It allows both building six-pack abs, and tightening the belly.*" - Wikipedia

  Crunches target the core muscle of the upper body (torso) called rectus abdominis. More commonly called the six-pack muscle. It also targets the sides of the trunk, and muscles in the hips, lower back, and pelvis. These muscles work together to stabilize the body posture. It's a beginner-friendly exercise done to target the abs and can be done in a free-hand way. However, this exercise puts a strain on your back and neck and care should be taken to avoid any injury, especially for older adults.

- Lunges: "*A lunge can refer to any position of the human body where one leg is positioned forward with knee bent and foot flat on the ground while the other leg is positioned behind.*" - Wikipedia

The lunge is a simple yet very effective exercise for the lower body. It is a multi-joint exercise targeting muscles of quads, hamstring, glutes, hips, and calves. It also improves body stability and unilateral movements. The rhythmic contraction and lengthening of the muscle make it very effective. We can increase the complexity by adding some simple gym weights. Some specific points to take care of are - not lunging too far forward, externally rotating knee, taking a stance that is too wide/close.

- Plank: "*The plank is an isometric core strength exercise that involves maintaining a position similar to a push-up for the maximum possible time.*" - Wikipedia

  Plank is one of the best exercises to have strong and stable core muscles, mainly targeting the core and abdominal muscles. It can also be used as a stress test. It is more of a strength-building than cardio exercise. Some common mistakes to avoid are to avoid - arching of the back, sagging of hips, and tilting the head up.

- Squat: "*A squat is a strength exercise in which the trainee lowers their hips from a standing position and then stands back up. During the descent of a squat, the hip and knee joints flex while the ankle joint dorsiflexes; conversely the hip and knee joints extend and the ankle joint plantarflexes when standing up*" - Wikipedia

  The squat is another freehand strength training exercise that works both the upper and lower body. It mainly targets the thigh and the glutes. It strengthens the core and lower body along with reducing the risk of injury and burning calories. There are some variations for adding some variety. Few things to take care of while performing squats are - having a solid base, keeping eyes forward and posture straight, keeping the core engaged.

The key challenge here is to provide a smooth experience to the user, by keeping the application as much real-time as possible. We will try to build the following features in our application:

- Maintain exercise catalog
- Provide real-time, easy to understand audio-visual feedback for pose correction
- Count repetitions and keep time for the user
- Carry out basic analytics and provide insights for user

Chapter 3

3.1 Literature Survey

We will try and understand the related works specific to the problem we are trying to solve.

HUMAN POSE ESTIMATION is one of the main computer vision problems, which must be implemented to build this application. Human pose estimation is the detection and localization of human body parts(limbs, joints, and other body parts as key points) so that we can ensure that the person is following the correct posture for the exercise. Some challenges which make the problem a non-trivial one, are - lack of training data for exercise from various angles, hidden joints and body parts due to heavy clothing, detecting the key points accurately despite occlusion or varied lighting conditions, which is a common scenario in homes.

3 of the most common human body models are [2]:

Skeleton-based models estimate the skeletal structure of the body using a set of joints and body parts like the ankle, knee, shoulder, elbow, wrist, and orientation of limbs.

In *contour-based models*, the torso, and limbs are estimated with the help of boundaries and shapes of a person's silhouette.

Volume-based models represent the body mainly in the form of 3D shapes using geometric meshes and shapes.

Initially, this problem was solved using classical image processing approaches and hand-crafted features, but deep learning and advanced CNN architectures over the last decade have delivered some path-breaking results.

DeepPose[3] from the researchers at Google was one of the exceptional works in this area, based on Deep Neural Networks(DNN) and the advancements in visual classification and object localization. They set up the pose estimation as a DNN-regression problem using convolution layers(from AlexNet). Using DNN helps to capture a larger context from the image and there is no need to manually specify features like graphical methods. Such DNN layers are cascaded to achieve a high precision result, by using the full image to start with initial pose estimation and train the DNN based regressors with subsequent sub-images and achieve better refined joint predictions.

The researchers at OpenPose[4] have worked to achieve a good solution for a multi-person pose estimator. Unlike most of the previous works, which mostly use a top-down strategy of first detecting people in the image and then estimating their pose independently for each of them, they are using a bottom-up strategy of representing the association scores with Part Affinity Fields(PAFs), which are a set of 2D vector fields that encode the location and orientation of the body parts. The top-down strategy is like

a pipeline, it first commits the region where people are occurring in the image, so if some error creeps in then there is no way to go back. Also, it misses any information brought in by spatial dependencies among people at a global level.
Another good work, specific to Hand Key-points detection[5] was brought up by the team from Carnegie Mellon University. They use a multi-camera setup or as they term it - multiview bootstrapping. This ensures a performance boost even in cases there are slight occlusions. Reprojecting the triangulated 3D hand joints help in cases where occlusion is dominant. This also adds to the existing data and thus makes the detector more powerful and accurate iteratively.
We also have yet another promising and powerful top-down approach for Multi-person Pose Estimation[6] from the researchers at Google. Detecting people in the “wild” and then solving pose estimation, makes this solution more rigorous. They propose a stage-wise method - using a Faster RCNN detector in the first stage to predict the location and scale of the bounding boxes which are most likely to contain people. The second stage first predicts the key points of the contained person with some estimation, after which it predicts dense heatmaps and offsets using a full convolution ResNet. The final key points are a result of aggregation using their novel approach. Some other novelties introduced in this paper are - keypoint-based Non-Maximum-Suppression (NMS) and confidence score estimation. They use the publicly available COCO dataset.
Newell et al, in their work, called Stacked Hourglass Networks for HPE[7] continue along the lines of leveraging the power of CNNs to get phenomenal performance without the need of hand-crafted or graphical features. Their novel “stacked hourglass” deep network, captures and consolidates information in a scale-invariant way, using pooling and subsequent upsampling, in continuous blocks one after another, hence combining information from various scales. Also, while training, repeated bidirectional inference and intermediate supervision help to achieve good accuracy.

We also went through some of the latest works in **HUMAN POSE MATCHING.**
Pradnya et al, in their paper - Match Pose[8] - talk about comparing the detected pose of a person with that of a reference pose, by comparing the angles between the joints, instead of trying to superimpose the skeletal structure. The angles are independent of the length of the skeleton, hence making the solution scale independent. Any part of the body is taken as a base to find the relative position of others, and the angles are calculated using the cosine triangle rule.
Wilms et al. have taken this forward in their work - Human Pose Matching[9]. They take the key points from the pose estimator and then match it with the model pose by calculating affine transformation, after normalizing the input pose by extracting a box around it. Affine transformation searches for a mathematical transformation(rotation, shear, translation, scaling) to fit the predicted pose with the model pose, and then the final score is calculated.

3.2 Evaluation Metrics

We studied some of the prominent evaluation metrics and key performance indicators relevant to this application. Some of these are used directly as cost-function for training pose estimation models whereas others are used in pose matching or in quantifying the exercise quality.

Percentage of Correct Parts(PCP)
A body part is considered correctly detected if the distance between two predicted and actual joint location is less than half of the joint length. It penalizes shorter limbs more since shorter limbs have smaller thresholds.

Percentage of Correct Keypoints(PCK)
A joint is considered correctly detected if the true and predicted points are within a specified threshold distance. PCKh@0.5 means 50% of head bone length, PCK@0.2 means 20% of the torso diameter.

Object Keypoint Similarity is commonly used in COCO keypoints challenge

$$\frac{\sum_i \exp\left(-d_i^2/2s^2k_i^2\right)\delta\left(v_i > 0\right)}{\sum_i \delta\left(v_i > 0\right)}$$

Where d_i is the Euclidean distance between the detected keypoint and the corresponding ground truth, v_i is the visibility flag of the ground truth, s is the object scale, and k_i is a per-keypoint constant that controls falloff.

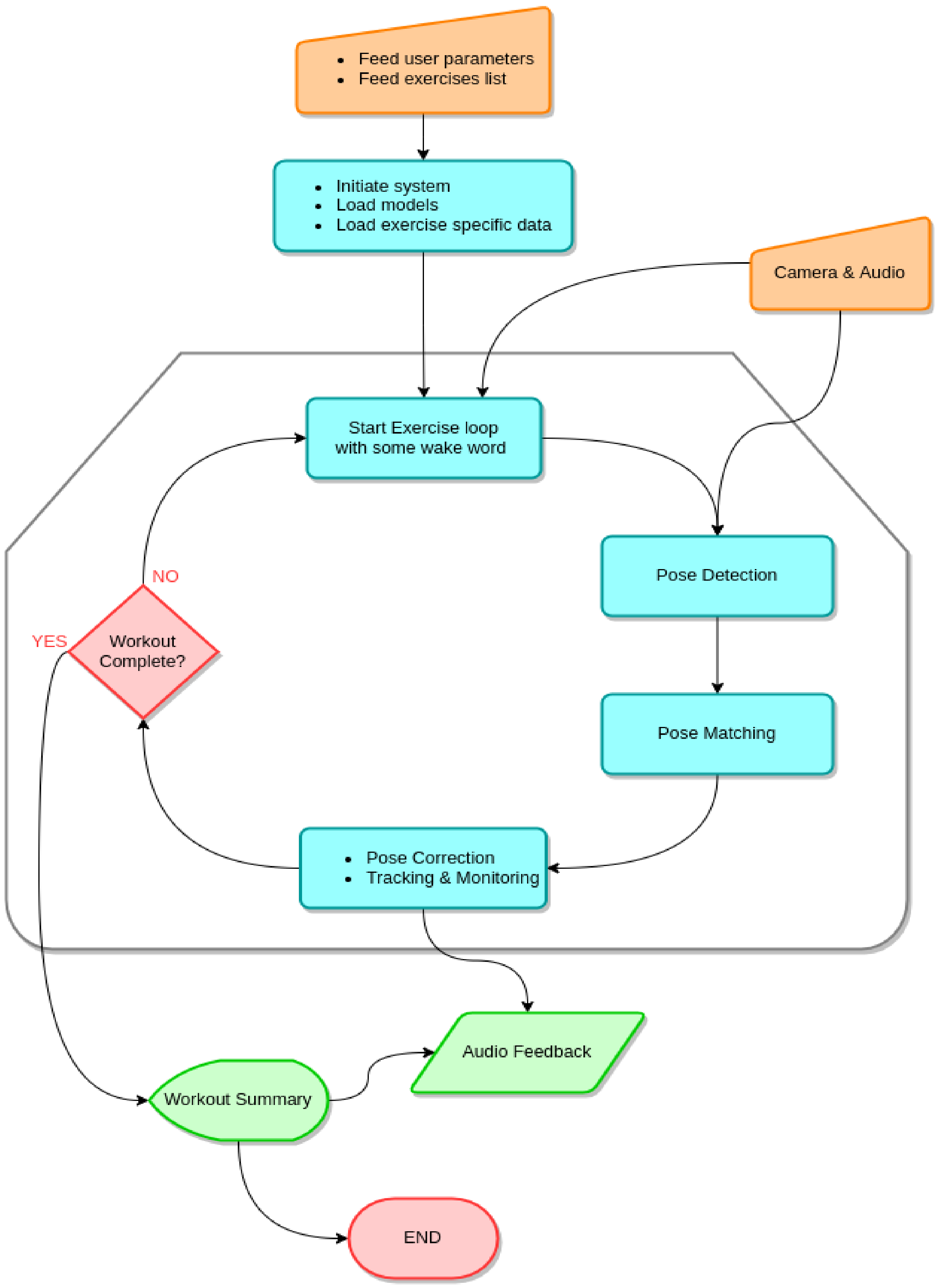
While doing the set of exercises, Work-to-Rest-Ratio measures the complete cycle of repetition within one exercise set, and Repetition-tempo-rate represents the tempo (or speed) at which a user performs a repetition. Having a consistent rate is an essential feature of a healthy workout routine.

We also calculate another metric - Workout Intensity, which is:

$$\frac{exercise\ duration}{exercise\ duration + non{-}exercise\ duration}$$

where duration is measured in terms of the number of frames(not time).

3.3 System Design



Chapter 4

4.1 Application Modules

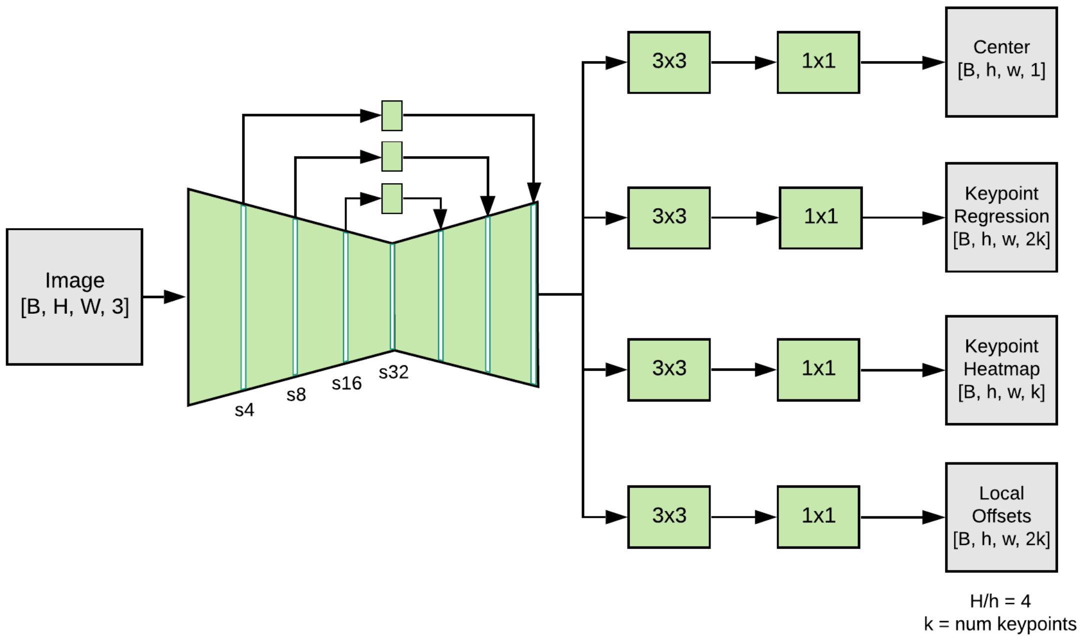
As per our defined architecture, we break the entire system into various main modules. Each will work in coherence with others and cater to specific functionalities for the system as a whole. Our end-to-end system covers all the aspects of the process.

For **User Input**, we use a simple laptop webcam. It will take the video feed and show it live on the UI. Each frame captured is processed further by the other module. We developed an app using Streamlit to work as the UI. The user faces the application from a single perspective and carries on with their exercise. There is no specific requirement such as distance, just that the whole body should fit properly in the video frame.

The **Frame Processor** module takes each of the extracted frames and processes it sequentially. The first and foremost task was to detect the human body from a given image frame and there the various key points(body points) by pose estimation. We explored various state-of-the-art openly available pose detection packages for this - like MoveNet(17 key points) and MediaPipe(33 key points).

MoveNet[11] is an ultra-fast and reasonably accurate model available on TF Hub. It comes in two flavors - *Lightning* and *Thunder*. Lightning is slightly lighter and helpful for applications demanding high latency, whereas Thunder is intended for applications requiring high accuracy. Both can cater to 30+ FPS on modern computing devices (laptops, smartphones, etc.)
As per the official blog, the MoveNet architecture is built on a bottom-up estimation model. The human body key points are accurately localized using the heatmap approach. The two main components are - a feature extractor and a set of prediction heads. The prediction scheme is loosely based on CenterNet with some optimizations done for speed and accuracy. MobileNetV2 is used as the main feature extractor here. A feature pyramid network here helps to achieve semantically rich feature map output.

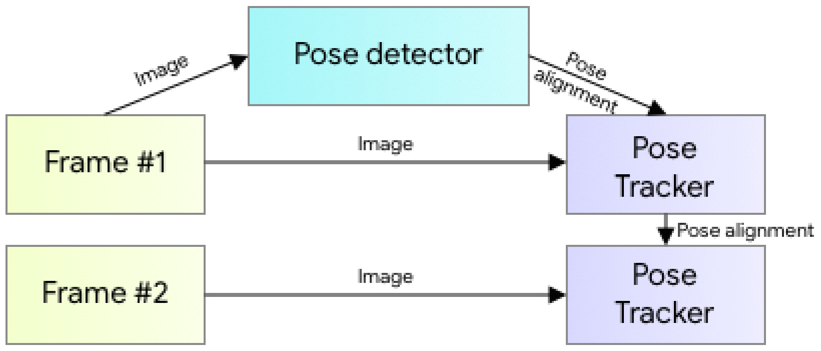
- Person center heatmap: predicts the geometric center of person instances
- Keypoint regression field: predicts a full set of key points for a person, used for grouping key points into instances
- Person keypoint heatmap: predicts the location of all key points, independent of person instances
- 2D per-keypoint offset field: predicts local offsets from each output feature map pixel to the precise sub-pixel location of each keypoint



MoveNet architecture(Source: blog.tensorflow.org)

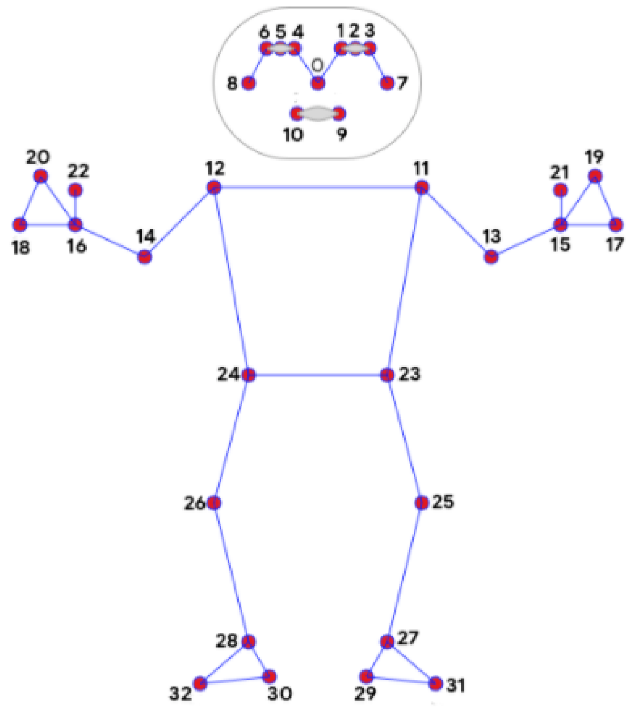
MediaPipe[12] is a cross-platform, open-source framework developed by Google for applying machine learning pipelines over data streams(audio, video, sensor, etc). It has built once deploy anywhere capability, along with ready to use cutting edge ML solutions for various kinds of tasks.

MediaPipe Pose is one of the solutions built for low-latency body pose tracking from RGB images or video frames. It is built on top of BlazePose research. It can infer 33 3-dimensional landmarks(body points) and a background segmentation mask on the whole body.



Pose Estimation Pipeline(Source: ai.googleblog.com)

The solution utilizes a 2-step detector-tracker ML pipeline. The detector part first detects the region-of-interest within the frame for person/pose. Thereafter, the tracker part, predicts the pose landmarks and segmentation masks within the region-of-interest using an ROI-cropped frame as input. For video data, the detector runs for the first frame, and thereafter it's just the tracker for subsequent frames unless the confidence falls below a certain threshold so the detector is run again.



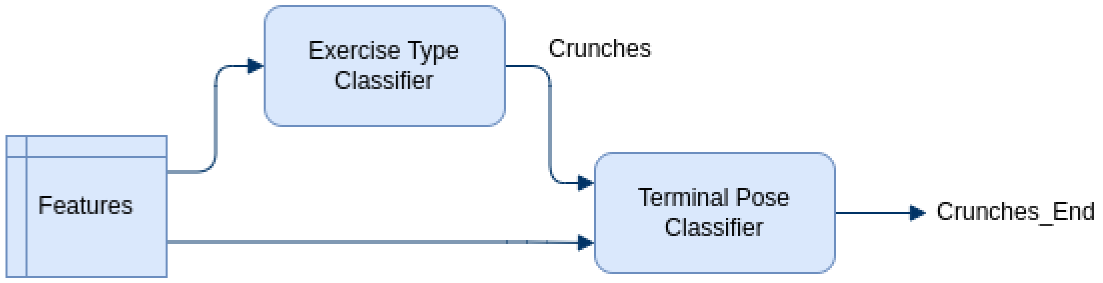
0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Pose Landmarks(Source: google.github.io/mediapipe)

After exploring both the libraries, we fixated on using MediaPipe which is a good blend of speed and accuracy. It also gives more body points in a 3-dimensional format along with their corresponding visibility score. This package is fairly easy to install and use like any other simple python package we use in the application. We also have control to set the tracking and detection confidence thresholds while using the framework, and can also enrich our feature sets with the given visibility score. After pose estimation, we use these key points to featurize the given image frame. We will discuss featurization in detail in the coming sections. Lastly, the frame processor recommends any correction to the user given the detected pose.

The **Pose Classifier** module uses the pre-trained XGBoost classification models and returns the detected exercise type from the given extracted features of the image frame.

The first level of classification tries to detect the correct exercise type. This detected exercise along with the features is further used to drill down to the specific terminal pose of that exercise type. To avoid too much jitter in the classification we also use a sliding window approach - of 3 frames class. This ensures that the predicted class labels are smoothened.



Pose Classifier Module(Source: self)

The **Speech Engine** module is the one responsible for audio output to the user. Any specific message(which can be exercise feedback or any error) can be passed to this module, which converts the text to sound signal and plays for the user.

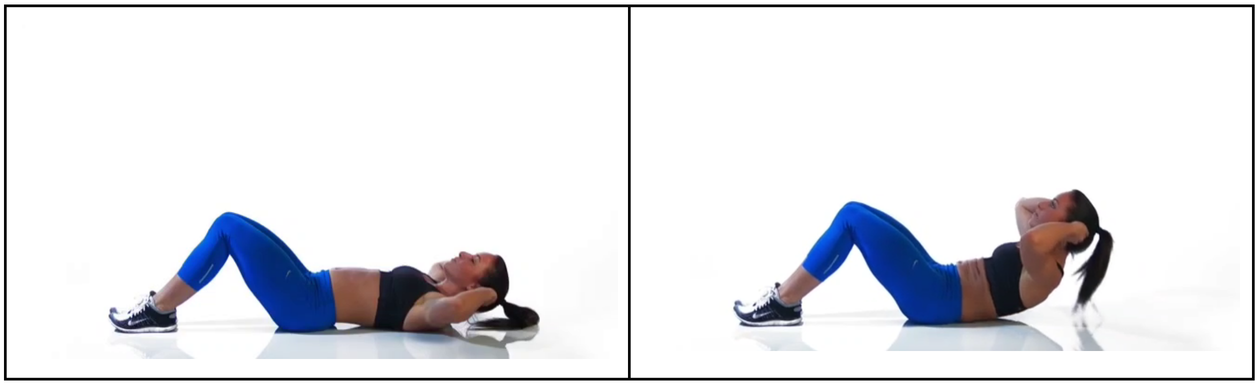
Main Engine is the actual driving module of the entire application. It keeps the application's live stream in front of the user. It extracts frames in RGB format from the stream at an interval of 200ms (to avoid too much congestion) and sends the frames to downstream modules. It gets the predicted labels for the corresponding frames and recommended corrections for the user(if any). It displays the reference gif for the user on the UI for the detected exercise type and maintains the set counts for each of the exercises done by the user.

Apart from these we also have some other utility modules like **Worker** which is a template class for multiprocessing. This enables us to run all the modules parallelly.

4.2 Data Collection

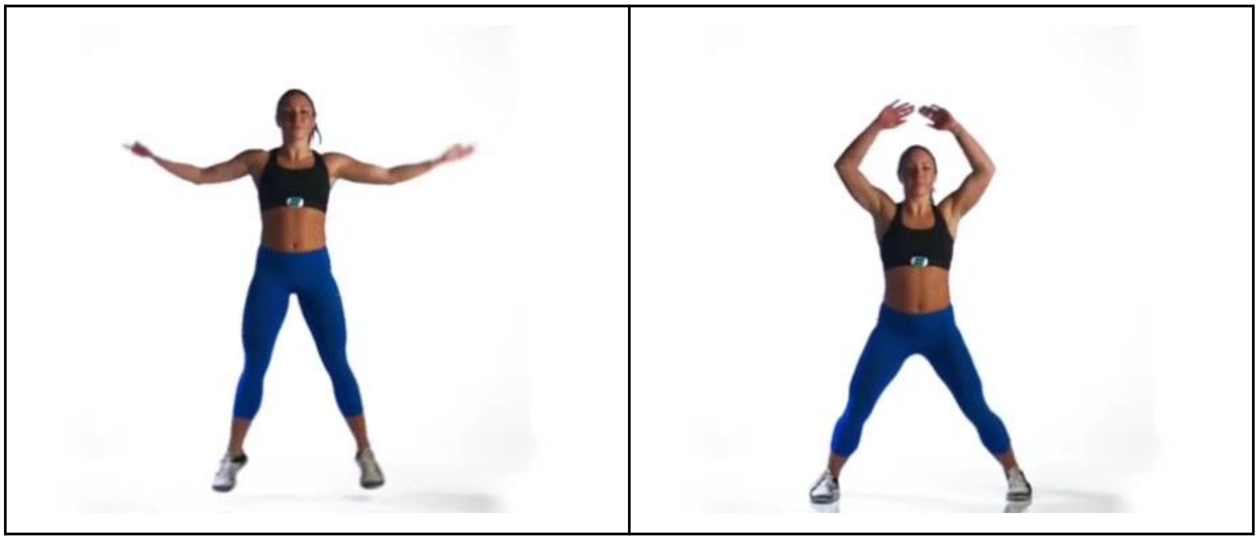
Needless to say, collecting good quality and ample training data is one of the essential prerequisites for any machine learning problem. We needed data to train our exercise type classifier and exercise pose classifier. We divided all the exercises we are covering into 2 terminal poses - *start* and *end*. The idea is - any exercise begins with the start pose and transitions to the end pose. We tried to find any open source images for the exercise types we are training for, but no properly labeled training data was available. So we searched on youtube and internet articles for images of users doing these exercises from the front angle and collected screenshots. Some of the samples are as below:

Crunches

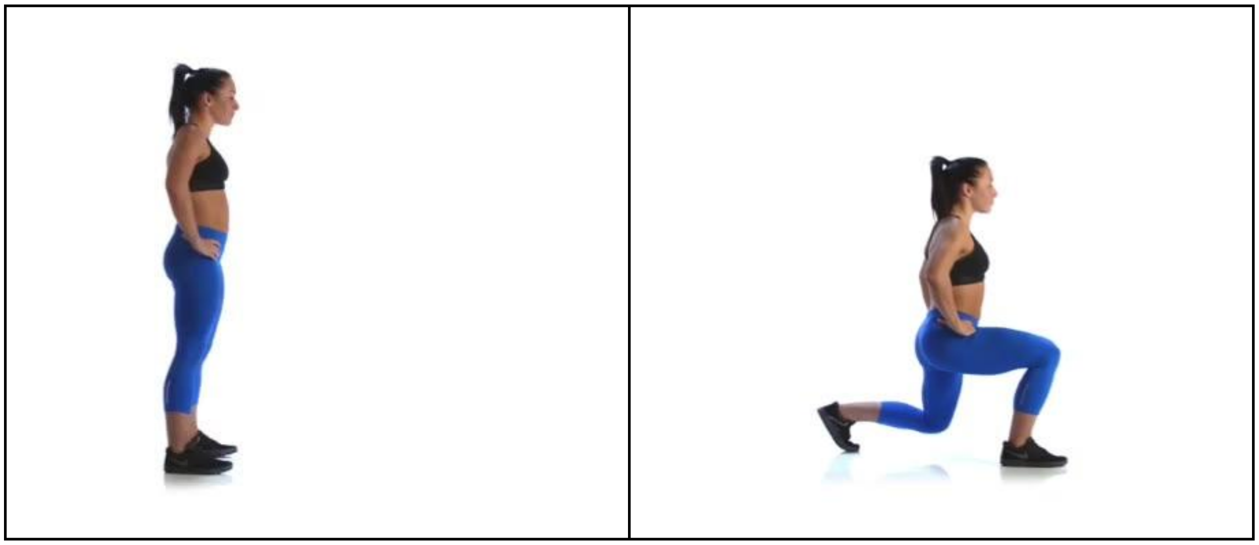


Crunches_Start - Crunches_End

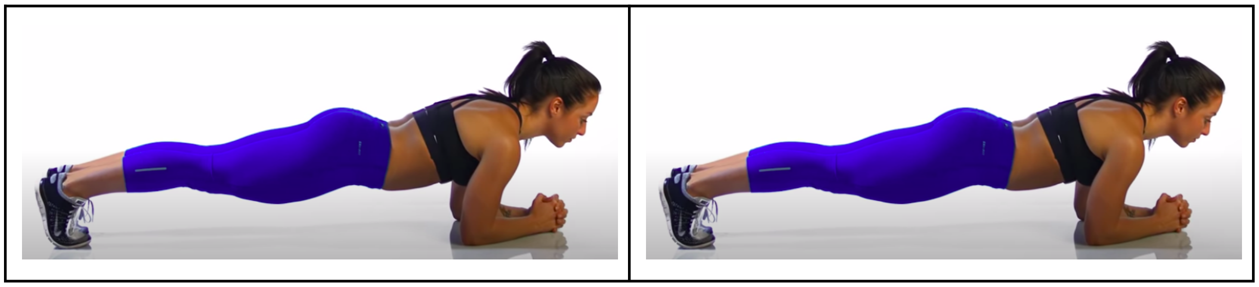
Jumping Jacks



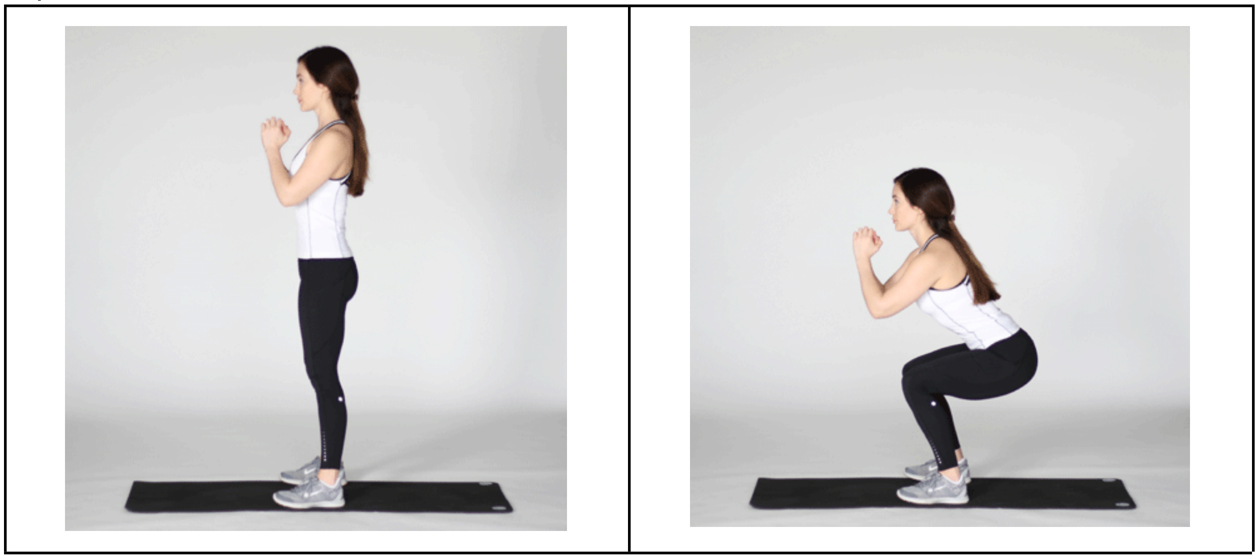
Jumping-Jacks_Start - Jumping-Jacks_End


Lunges_Start - Lunges_End

Planks

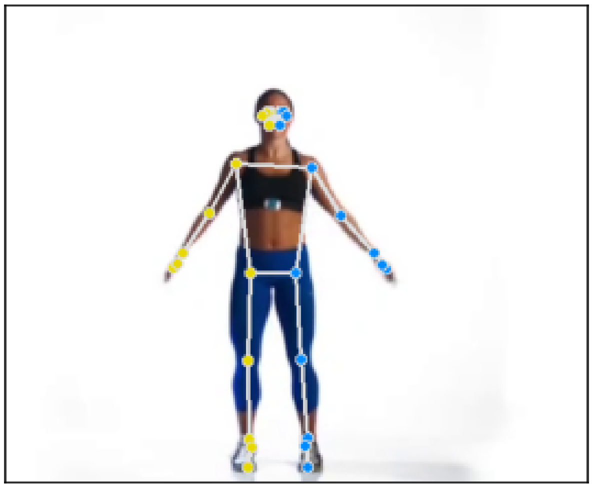

Planks_Start - Planks_End

Squats


Squats_Start - Squats_End

4.3 Featurization

As discussed in the previous sections we have to convert the RGB frames extracted from the video stream into feature vectors which we can feed into the classifier module. The output from the MediaPipe framework after plotting the joint connections is as below:



MediaPipe Output

Firstly we discard some of the body points from the MediaPipe output, which are too small compared to the body size and might add to more noise rather than information. We consider only these specific body points:

Nose	
Left Shoulder	Right Shoulder
Left Hip	Right Hip
Left Elbow	Right Elbow
Left Wrist	Right Wrist
Left Knee	Right Knee
Left Ankle	Right Ankle
Left Foot Index	Right Foot Index

Body Points taken from MediaPipe

We also calculate some more body points using the points we already got as mentioned above. They are:

- neck = (left shoulder + right shoulder) x 0.5
- torso length = (dist between neck-left hip + dist between neck-right hip) x 0.5
- mid hips = (left hip + right hip) x 0.5
- core = (neck + mid hips) x 0.5

We experimented with various kinds of featurization techniques using all the body points and their visibility score given by the MediaPipe library - like calculating distances between body points(joints), calculating the angles between the important joint connections, visibility of different body points, etc. We found that the best results come from using a combination of angles and distances. So these are the features we finalized:

<u>Distance features between body points(Euclidean distance in a 2-D plane)</u>

1-joint distance

- Core - Nose
- Core - Left Elbow
- Core - Right Elbow
- Core - Left Wrist
- Core - Right Wrist
- Core - Left Knee
- Core - Right Knee
- Core - Left Ankle
- Core - Right Ankle

2-joint distance

- Left Shoulder - Left Wrist
- Right Shoulder - Right Wrist
- Left Hip - Left Elbow
- Right Hip - Right Elbow
- Left Shoulder - Left Knee
- Right Shoulder - Right Knee
- Left Hip - Left Ankle
- Right Hip - Right Ankle
- Left Knee - Left Foot Index
- Right Knee - Right Foot Index

Cross-joint distance

- Left Wrist - Right Wrist
- Left Elbow - Right Elbow
- Left Shoulder - Right Shoulder
- Left Hip - Right Hip
- Left Knee - Right Knee

We normalize the distances by torso length. Also, there were some other features that we removed due to high collinearity as found in the experiments.

<u>Angle features between 3-body points(Cosine angles in a 2-D plane)</u>

We calculate the cosine angle formed by 2 body points with a 3rd point. The angles are calculated in the range of 0-360° so that the ones symmetrical on the y-axis can be differentiated.

- Angle left elbow - neck - right elbow
- Angle left knee - mid hips - right knee
- Angle nose - neck - mid hips
- Angle nose - core - ground

Lastly, we also normalize the angles by 360.

<u>Visibility features of left and right body profiles</u>

We also use the visibility score given by the MediaPipe framework for each body point to find the visibility score for the left and right body profiles and use them as features.

Left Profile

- (left shoulder.visibility + left hip.visibility) x 0.5
- (left hip.visibility + left knee.visibility) x 0.5

Right Profile

- (right shoulder.visibility + right hip.visibility) x 0.5
- (right hip.visibility + right knee.visibility) x 0.5

The whole feature set is produced by concatenating all these constituent features horizontally.

4.4 Training Data

Initially, we started with 5 pictures of each exercise pose, taken from similar angles but with slight differences. So, we have 50 pictures for 5 types of exercises. However, in our experiments, we found that since the number of data points is extremely small, almost all the classifiers had the problem of overfitting. Also, since the training data only had snapshots of youtube fitness videos, which means, all of them were nearly perfect poses. So, while testing on real users the classifier could not pick up slightly wrong poses and gave wrong outputs.

Hence we use the concept of synthetic data generation by adding noise. We add both positive and negative samples by adding slight or more noise in some of the selected features only. To create a positive sample we select half the number of features and add slight noise to them. If too many features are changed it might alter the entire pose. The step-wise algorithm to multiply the training data is as below:

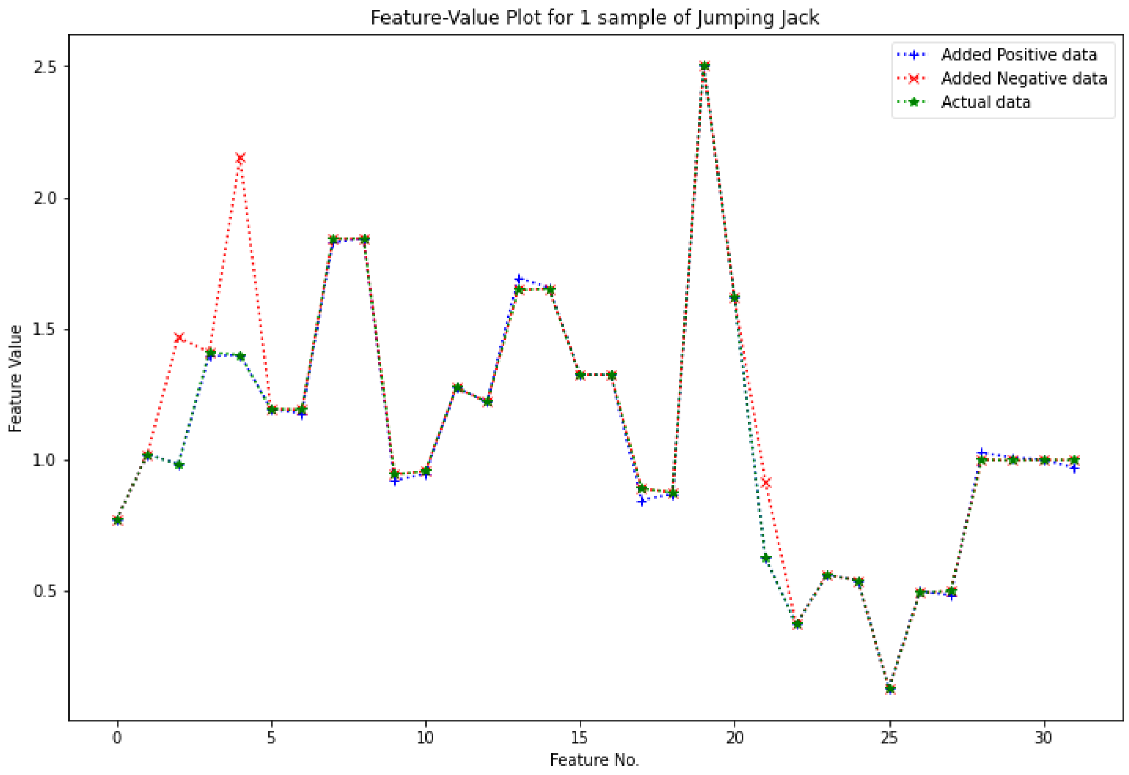
```
multiply data by noise(data, target_n, to_append):
  1. size_mu:= 50% of the total number of features in given data
  2. create target_n-1 sample sizes by drawing samples from a normal
     distribution having:
     mean:= size_mu, standard deviation:= 1.5
  3. for every size in sample_sizes from step 2:
      a. size = max(1, size)
      b. generate random noise for all the features from a normal
         distribution having:
         mean:= 0, standard deviation:= 0.025
      c. randomly chose the 'size' number of features and add noise
         to them
      d. if any of the angles features become negative we simply
         complement it, or if any of the distance features become
         negative we cap it to alpha=0.00001
  4. if to_append is not null, we append its value to the end of the
     data list and return it
```

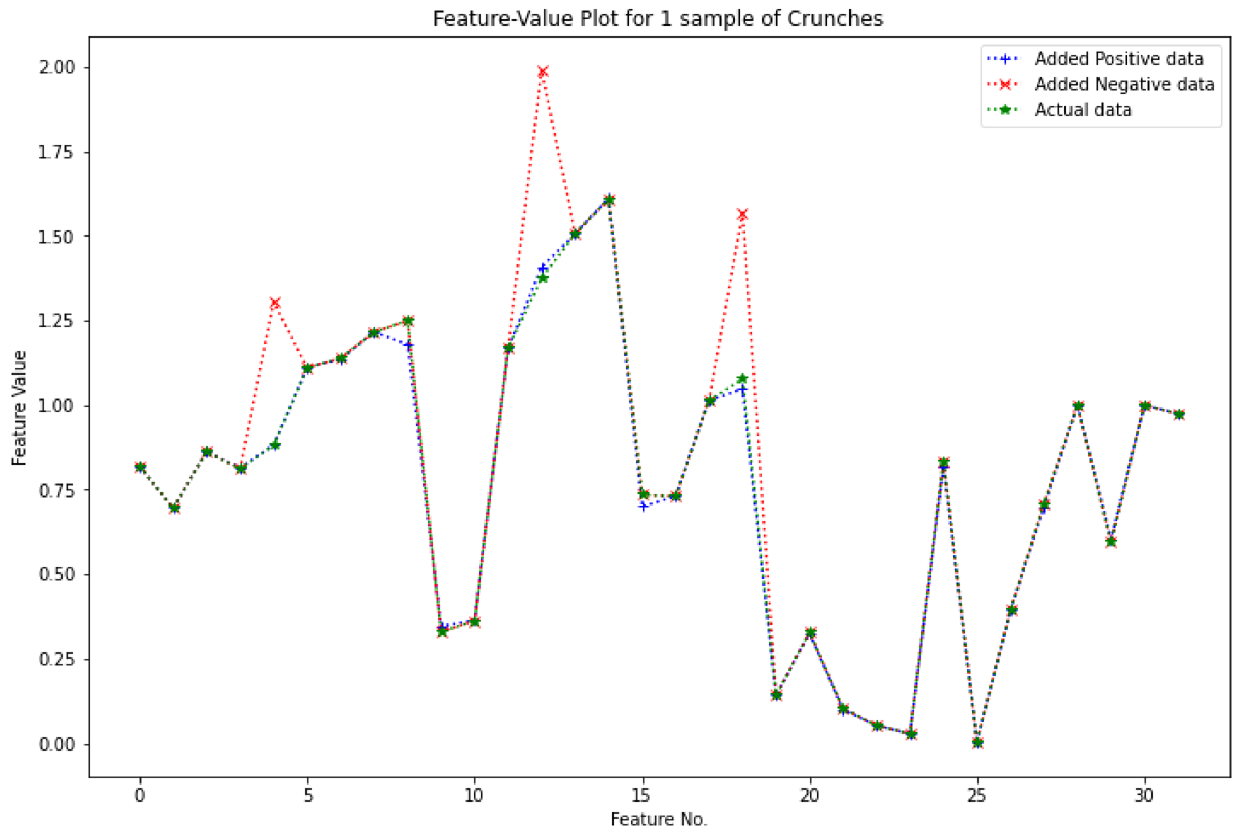
If we only have data related to the exercise classes, then in the real environment, the classifier will always categorize all frames into some exercise class. For example: even if the user is simply standing, the classifier might predict “squat_start” with maybe lesser probability. So we must have data samples that are NOT one of the defined classes. For generating negative samples for a particular feature pose, we choose a small number of features but increase the amount of noise for those features. This is based on our theoretical understanding that even if a particular feature in the pose differs to a large extent then that should be considered a different pose from the original.

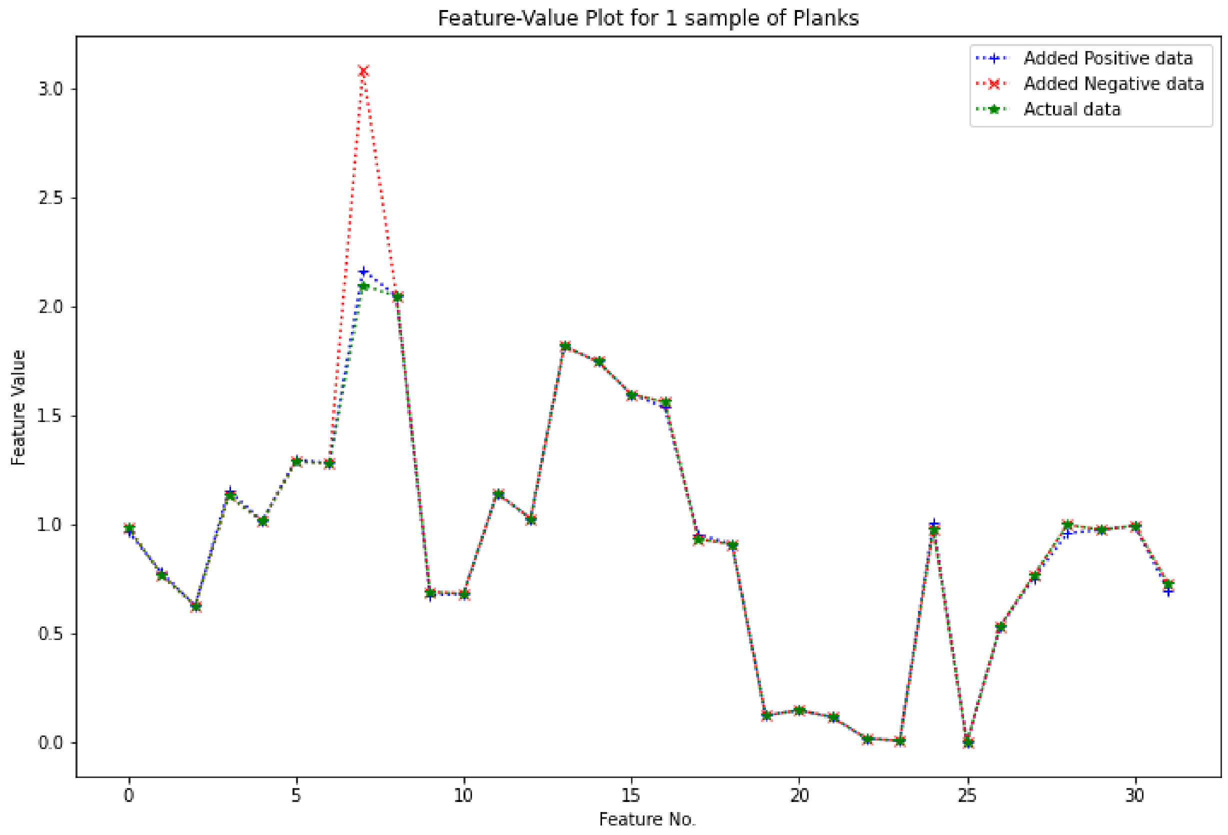
```
negative sampling by noise(data, target_n, to_append):
  1. size_mu:= 5% of the total number of features in given data
  2. create target_n sample sizes by drawing samples from a normal
     distribution having:
     mean:= size_mu, standard deviation:= 1.5
  3. for every size in sample_sizes from step 2:
       a. size = max(1, size)
       b. generate random noise for all the features from a normal
          distribution having:
          mean:= 0.5, standard deviation:= 0.05
       c. randomly chose the 'size' number of features and add noise
          to them
       d. if any of the angles features become negative we simply
          complement it, or if any of the distance features become
          negative we cap it to alpha=0.00001
  4. if to_append is not null, we append its value to the end of the
     data list and return it
```

<u>50 datapoints converted to 3500 datapoints after both algorithms</u>

Lets us look at some examples graphically




Feature-Value Plot for 1 sample of Crunches
Added Positive data
Added Negative data
Actual data
Feature Value
Feature No.
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30


Feature-Value Plot for 1 sample of Planks
Added Positive data
Added Negative data
Actual data
Feature Value
Feature No.
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30

Chapter 5

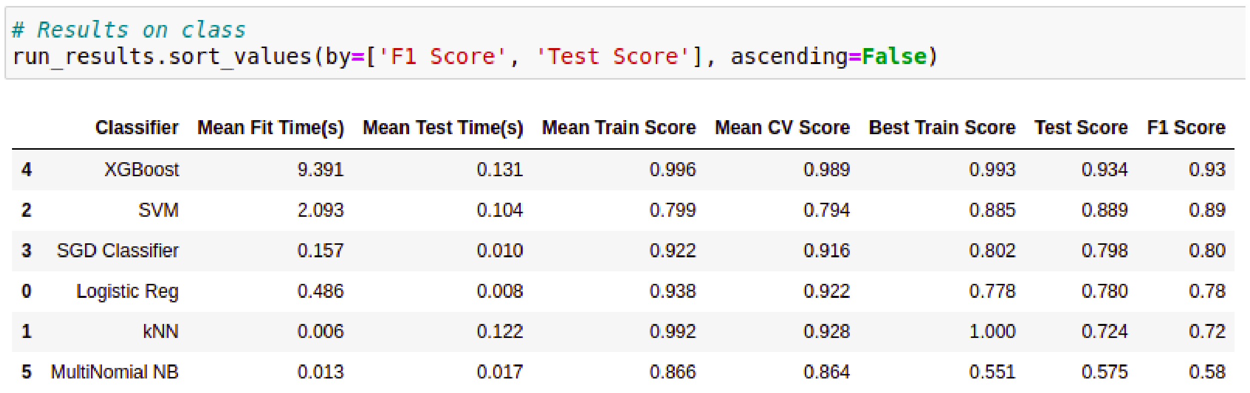
5.1 Model Training & Tuning

k-Nearest Boosted Pose Model

The requirement of the application is such that we need to have a classifier that is of extremely low latency. Also, we don’t have a large amount of well-labelled training data, so the possibility of using any deep learning method (which can do automated feature extraction) does not seem to be feasible. So we continue our experiments on the data collated using classical ML techniques. We explore various classification algorithms like - Logistic Regression, k-Nearest Neighbour, Support Vector Machine, SGD Classifier, Multinomial Naive Bayes, and XGBoost.

Classifier Level 1

Our first classifier is the one that predicts the type of exercise, given a set of features. We run all the classification algorithms and get the below results:



```
# Results on class
run_results.sort_values(by=['F1 Score', 'Test Score'], ascending=False)
```

	Classifier	Mean Fit Time(s)	Mean Test Time(s)	Mean Train Score	Mean CV Score	Best Train Score	Test Score	F1 Score
4	XGBoost	9.391	0.131	0.996	0.989	0.993	0.934	0.93
2	SVM	2.093	0.104	0.799	0.794	0.885	0.889	0.89
3	SGD Classifier	0.157	0.010	0.922	0.916	0.802	0.798	0.80
0	Logistic Reg	0.486	0.008	0.938	0.922	0.778	0.780	0.78
1	kNN	0.006	0.122	0.992	0.928	1.000	0.724	0.72
5	MultiNomial NB	0.013	0.017	0.866	0.864	0.551	0.575	0.58

We can clearly see, going by F1 score metrics and Test Score that XGBoost is clearly the best classifier for exercise type. Although it takes the most amount of prediction time compared among all of them, the absolute time taken is around 100ms which is acceptable, given the good accuracy it is providing. This is intuitive that a decision tree-based algorithm is likely to perform well on this kind of hand-curated features.

```python
print("Classification Report for the best params : ")
print(classification_report(y_test, y_pred, target_names=subclas_encoder.classes_))
```

```
Classification Report for the best params : 
               precision    recall  f1-score   support

     crunches       0.93      0.96      0.94       150
jumping_jacks       0.97      0.99      0.98       150
       lunges       0.93      0.97      0.95       150
       planks       0.92      0.97      0.95       150
       random       0.93      0.83      0.88       300
       squats       0.93      0.98      0.95       150

     accuracy                           0.93      1050
    macro avg       0.94      0.95      0.94      1050
 weighted avg       0.93      0.93      0.93      1050
```

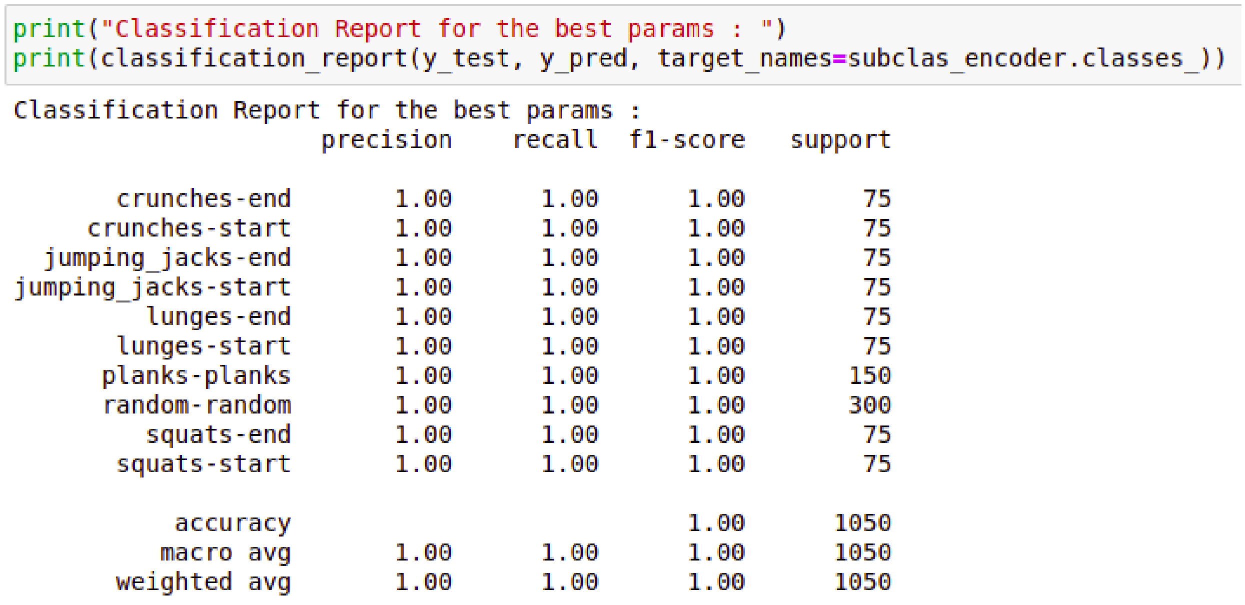
Classifier Level 2

Once the first classifier predicts the type of exercise, the prediction itself is added as a feature and sent to the next level of classification for specifying the pose of that exercise type. We again repeated our experiments on all kinds of classifiers and got the below results.

```python
# Results on class
run_results.sort_values(by=['F1 Score', 'Test Score'], ascending=False)
```

	Classifier	Mean Fit Time(s)	Mean Test Time(s)	Mean Train Score	Mean CV Score	Best Train Score	Test Score	F1 Score
1	kNN	0.007	0.121	1.000	1.000	1.000	1.000	1.00
2	SVM	2.284	0.112	0.814	0.814	1.000	1.000	1.00
4	XGBoost	4.729	0.125	1.000	1.000	1.000	0.990	1.00
0	Logistic Reg	0.726	0.013	1.000	1.000	0.998	0.991	0.99
3	SGD Classifier	0.108	0.014	0.992	0.992	0.984	0.974	0.97
5	MultiNomial NB	0.020	0.016	0.977	0.978	0.719	0.730	0.73

It is interesting to note that for the 2nd level of classification, almost all the classifiers are able to perform well barring Multinomial Naive Bayes. This is because the 1st level classification result added as a feature proves to be extremely helpful. Based on the F1 Score and Test Score kNN and SVM are the best classifiers but we choose kNN as it also has a better Train and CV score.



```
print("Classification Report for the best params : ")
print(classification_report(y_test, y_pred, target_names=subclas_encoder.classes_))
```

```
Classification Report for the best params :
                     precision    recall  f1-score   support

       crunches-end       1.00      1.00      1.00        75
     crunches-start       1.00      1.00      1.00        75
   jumping_jacks-end       1.00      1.00      1.00        75
 jumping_jacks-start       1.00      1.00      1.00        75
         lunges-end       1.00      1.00      1.00        75
       lunges-start       1.00      1.00      1.00        75
      planks-planks       1.00      1.00      1.00       150
      random-random       1.00      1.00      1.00       300
         squats-end       1.00      1.00      1.00        75
       squats-start       1.00      1.00      1.00        75

           accuracy                           1.00      1050
          macro avg       1.00      1.00      1.00      1050
       weighted avg       1.00      1.00      1.00      1050
```

While using the classifiers in the actual application we faced an issue. Since we were working with a simple laptop cam with low FPS, sometimes the extracted frame was not good and MediaPipe could not extract features out of those frames. This also happened if the user was too fast and the picture came blurry. This caused jitter as some of the frames had no exercise type or pose class associated with them. To avoid this issue we use a moving window approach. Our theoretical assumption is that any pose is not likely to be drastically different from the just previous poses in a matter of milliseconds, so we can associate that frame with the pose having the maximum majority in the window size. We have chosen a window size of 3.

5.2 Pose Correction

Once we get the exercise type and the specific pose of that exercise, the next thing is to check if the user is doing anything wrong specific to that pose. To solve this problem we use the body points coordinates extracted using the MediaPipe framework and our calculated features. We then employ some rules specific to the detected pose - based on a study we conducted about what are the key things to be taken care of while the body is in that specific pose while doing that exercise.

For the start pose of jumping jacks, we check if the width between feet is comparable to the shoulder's width and the angle made by the arms should be between 120 and 180. For the end pose of this exercise, we ensure that the arms are well above the head and feet are between 1.5 to 3 times shoulder-width apart.

When we are doing crunches, we ensure that the user starts with a relaxed position lying on the ground and ends with their head slightly raised from the neck and core.

While doing lunges, we check if the user has their core straight all the time and when they are ending the pose, their legs should be almost at right angles. Also, their knees should not cross the toe-line.

For planks, it is important for the person doing the exercise to have a straight body throughout. Right from head to heels.

For squats, it is important to ensure that the head is straight and the knees are not crossing the toe-line while lowering the body.

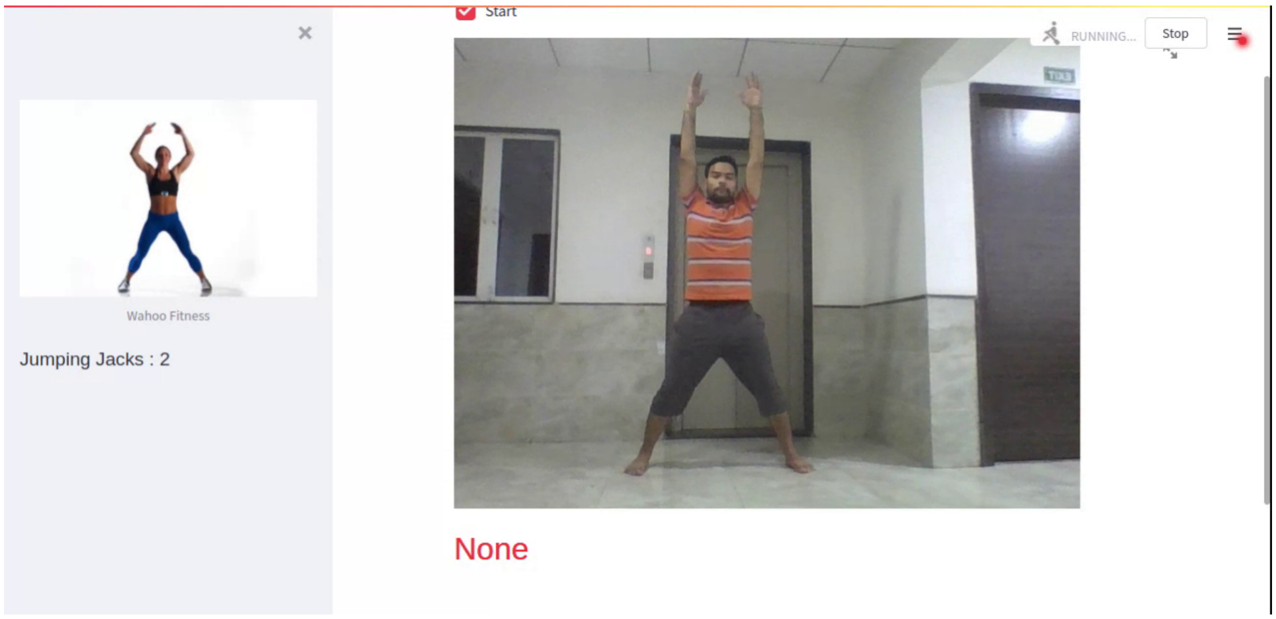
Once we figure out any major deviation from the set rules we inform the user to correct the posture.

Chapter 6

6.1 Application Deployment

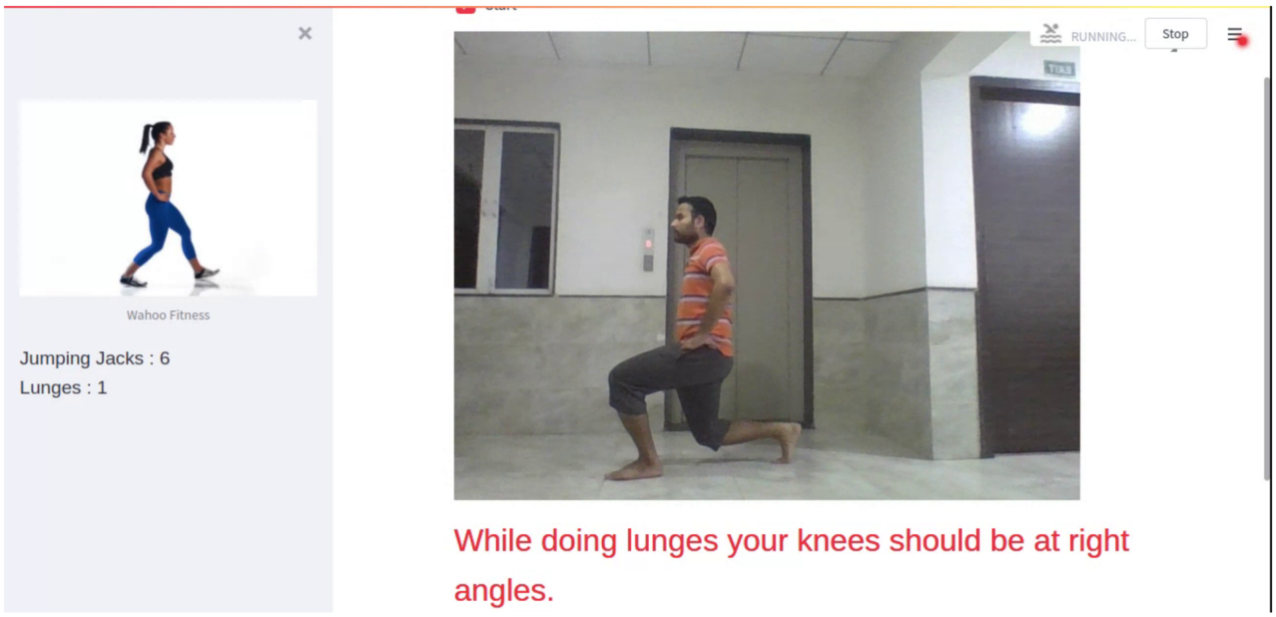
We finally decide to codify our theoretical findings and develop the application in a form that can be easily deployed on any system and can serve the end-user. After exploring various packages we finally decided to use **Streamlit**. The best thing about using streamlit is that it helps to easily convert our core python app into a working web app without too much work required for the front-end.

We also develop our application modules to function separately as different processes(multi-processing) so that they work coherently while communicating with each other using python Queues.



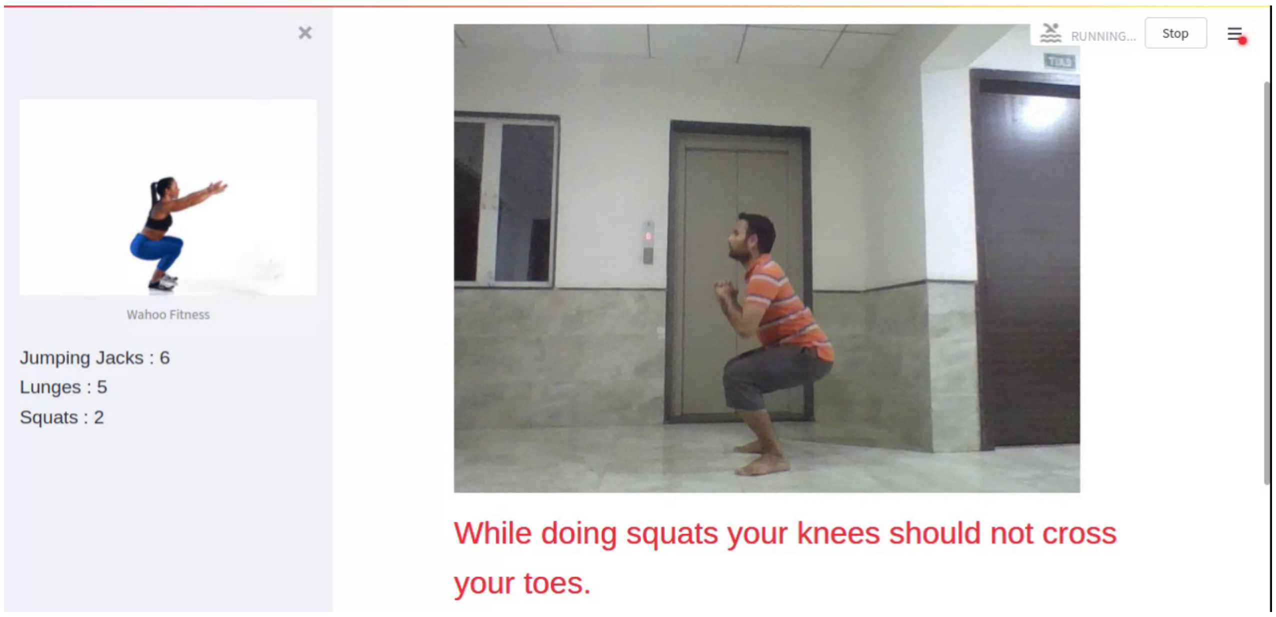
Jumping Jacks : 2

None



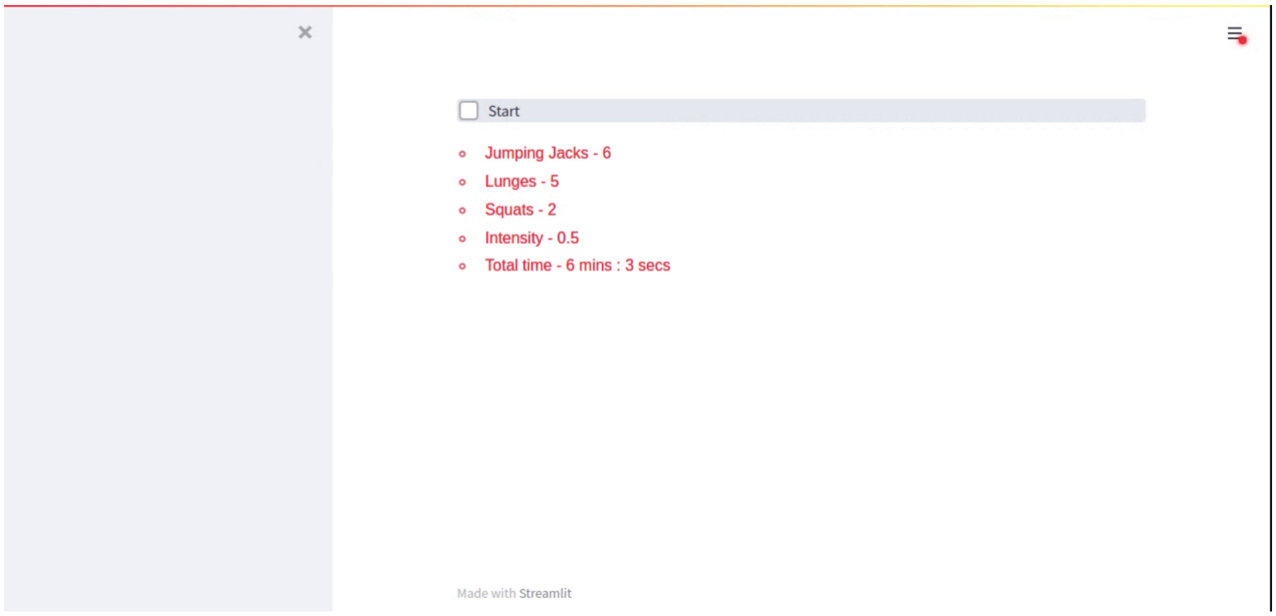
Jumping Jacks : 6
Lunges : 1

While doing lunges your knees should be at right angles.



Jumping Jacks : 6
Lunges : 5
Squats : 2

While doing squats your knees should not cross your toes.



- Jumping Jacks - 6
- Lunges - 5
- Squats - 2
- Intensity - 0.5
- Total time - 6 mins : 3 secs

6.2 Conclusion and Future Work

In this report we worked to build an application that can work as a virtual fitness trainer for any naive user. We worked to collect and collate data from scratch and also applied our augmentation techniques using random normal distributed noise over the data points. We also applied our k-Nearest Boosted Pose model and achieved a fairly high amount of accuracy on the test data. We employed a simple rule-based approach for solving the problem of pose correction by leveraging the classification and feature extraction module. Finally, we deployed a working application using the Streamlit package for the end-user.

We were also able to identify some areas of improvement in our work. Firstly, MediaPipe proves to be the main bottleneck for our application. Since our system is just CPU-based MediaPipe takes a little less than 200ms to process a single image frame. So we have to take frames at some specific intervals and drop the middle ones - this is leading to information loss. If we process the frames too much then after some time the MediaPipe queue gets overloaded and the real-time effect of the application is lost. The user carries on some exercise which is counted by the engine much later.
Also, our classifiers are trained on highly tailor-made data using augmentation. It will definitely help if we get better-curated actual data.
Another approach we could take with increased data is to make the pose correction based on some heuristics or ML-based.

6.3 References

1. Devgan, Kavita. "The lowdown on personal trainers". *Business Standard* (2013). https://www.business-standard.com/article/beyond-business/the-lowdown-on-personal-trainers-110071700021_1.html
2. Tatariants, Maksym. "Human Pose Estimation Technology 2021 Guide" *Mobidev*. (2020). https://mobidev.biz/blog/human-pose-estimation-ai-personal-fitness-coach
3. Toshev, Alexander & Szegedy, Christian. "DeepPose: Human Pose Estimation via Deep Neural Networks". *IEEE Conference on Computer Vision and Pattern Recognition* (2014).
4. Cao, Zhe & Martinez, Gines & Simon, Tomas & Wei, Shih-En & Sheikh, Yaser. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).
5. Simon, Tomas & Joo, Hanbyul & Matthews, Iain & Sheikh, Yaser. “Hand Keypoint Detection in Single Images Using Multiview Bootstrapping”, *IEEE Conference on Computer Vision and Pattern Recognition* (2017).
6. Papandreou, George & Zhu, Tyler & Kanazawa, Nori & Toshev, Alexander & Tompson, Jonathan & Bregler, Christoph & Murphy, Kevin. "Towards Accurate Multi-person Pose Estimation in the Wild". *IEEE Conference on Computer Vision and Pattern Recognition* (2017).
7. Newell, Alejandro, Kaiyu Yang and Jia Deng. “Stacked Hourglass Networks for Human Pose Estimation.” *ECCV* (2016).
8. Borkar, Pradnya Krishnanath, Marilyn Mathew Pulinthitha, and Ashwini Pansare. “Match Pose - A System for Comparing Poses.” *International journal of engineering research and technology 8* (2019)
9. Wilms, Jochen, Gil Beckers, Timothy Callemein, Luc Geurts, and Toon Goedemé. "Human Pose Matching." *Dortmund International Research Conference* (2018)
10. Chandra Babu, Sudharshan. "A 2019 guide to Human Pose Estimation with Deep Learning" *Nanonets*. (2019). https://nanonets.com/blog/human-pose-estimation-2d-guide/
11. Votel, Ronny and Na Li. “Next-Generation Pose Detection with MoveNet and TensorFlow.js” *Tensorflow.* (2021). https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html
12. MediaPipe Pose. https://google.github.io/mediapipe/solutions/pose.html