Documentation:

Business Problem:

According to the Research and Markets report, the digital fitness market size is expected to reach $27.4 billion by 2022
There is a rise in health-conscious people.
Exercises are an integral part of many people's lifestyles. Needless to say, they are beneficial to health. Plus, there is a rise in fitness habits, especially in millennials. (https://tinyurl.com/565pxjts)
The primary problem is that exercises can be very dangerous if performed incorrectly.
The other problems faced by individuals who want to exercise can be multifold.
Given that, they want to exercise regularly, they have to rely on a trainer to teach them how to perform exercises correctly. This condition is subject to other issues:

(a) Going to gym: According to this article (https://tinyurl.com/yc7z9jjr), lack of time and confidence are the main quoted reasons for the people who want to use gym for mental benefits. Can we make a solution which can enable people to work out , at least some exercises, from the comfort of their choice of place?

(b) Hiring Personal Trainer: This is likely to be an expensive option for most of users who want to exercise. Can we build a solution to address this issue of expense?

Domain-specific issues: What separates a correct exercise performance from an incorrect one? It boils down to correct and incorrect poses.
An exercise, generally, is repetitive in nature. A step or a cycle is a count of that exercise. That step is rule-bound. The rule, generally, involves the correct movement of specific parts. For example, in a bicep curl, the forearm needs to be static, and the upper arm needs to slowly go down till the upper arm and forearm are in line, then the forearm is lifted upwards.
There can be number of ways in which given an exercise, ML problem can be defined.

ML Problems: Given, we have a basic idea of domain-specific issues, what does the problem of separating badly performed exercise from a good one boils down to?
How can we track bad poses throughout a video clip of an exercise?
I am using a media pipe pose estimator model , BlaisePose to detect the key points which describe a particular pose. The change in coordinates of these points describe track the movements of these points , which quantifies the pose changes with time.

Once we have track of these coordinates, I have applied some basic heuristics to judge the basic criterias of correctness of an exercise. For example, in bicep curl, the user needs to lift forearm above the 90 degree angle with the upper arm.

Also, we can have data of pose changes of an expert trainer while performing a particular exercise. We can apply dtw(dynamic time warping) similarity between this data and that of the user. We set a realistic threshold, subject to experimentation. And then, can conclude if the video of user's performing an exercise is correct or not.

Pros and cons of the above methods:
The heuristic method
Pros: very simple to implement and the benefit of it is that it can give live analysis.
Con : It is not a data driven method and is very limited and specific in its judgement. For example, it is applicable to a limited set of points and only checks if thos limited set of points satisfy a particular criteria. As long as those limted set of points manage to satisfy the criteria laid down by heuristic method, the analysis will conclude the exercise to be correct.

Dtw similarity:
Pros: It is data driven. That is , the method looks holistically to the movements of the parts and analyse whether it is okay or not.

Cons: To apply it in real time video data is a challenge in itself. As, it is a data driven method, it needs more data points to calculate similarity with already collected data. This can be simply done after a video clip of exercise performance is recorded.

Intuitively, dtw approach is based on finding similarity between a test video and videos already stored.
If the test video comes very similar to the good ones, it should be marked as good while in other cases, marked as bad one.

Dtw similarity: Any two time series can be compared using euclidean distance or other similar distances on a one to one basis on time axis. The amplitude of first time series at time T will be compared with that of second time series at time T. This will result in a very poor comparison and similarity score even if the two series are very similar in shape but out of phase in time.
A tack of movement of coordinates of a specific estimated point of user while performing exercise need not be in same phase as of that of the already recorded data of an expert. And hence , the use of dtw similarity.
Dtw similarity compares amplitude of first series at each time T , with all the amplitudes of the second series at all the times available. The cost matrix is defined with all the amplitudes of one series along a dimension of the cost matrix, and with all the amplitudes of another series along the other dimension of this cost matrix. The cost matrix is filled and the minimum cost path from one end to another is found. The average of which is called dtw distance.

More issues:
Different persons with different heights
Different camera position
threshold infer
Giving more detailed suggestion about what is wrong in the execution
How to process the relevant part of an exercise during continuous video stream

Third approach: Classification

For a given exercise, there can be multiple ways it can go wrong. This depends on domain knowledge to identify the wrong classes.

So, we can use an model for multi-class classification for our video data. The important thing to note is that our data is video and in a video frames are sttacked temporally.
This nature must be kept in mind for the choice of model to be decided later.
Ideal model which comes to mind is RNN based or CNN with 1 dimensional convolutions based models.

I have selected the third approach for my task.
I have narrowed my task to a specific exercise of squat.
So, finally, my task is squat classification on real time data.

Dataset

a. **Source of the dataset**: The dataset contains video clips of my friends and my execution of exercises. Till now, bicep curl only. Also, there are you tube videos which are to be used in future. I have had hard time finding vieo clip of bicep curl, which could be used for my solution. But, for other exercises, the clips are there and I will use them in later phases of experiments
b. **Explanation of each feature and datapoint available**:

Basically, a datapoint means the track of some specific estimated body points with respect to time. The possible useful features are frequency of the movement of points, the variability in the movements, the peak and trough in the series.

c. **Data Size and processing challenges:**

The data-size is not a challenge in itself, at least up till now in the project.

However, processing it poses some challenges. A few of them are:

1. How to detect start of an exercise: A user generally takes some time before start of an exercise. The part of the time series corresponding to inactivity is not likely of any use for our purpose, so we would like to discard that. Also , even after ending exercise, user may take time before he/she decides to end the live recording.

Also, the frequency by which a user performs an exercise is very likely too vary.

For example, when someone starts bicep curl, the speed of the performance may be consistent and normal , but as the muscles tire up and the speed of the performance slows down. Now, the speed of performance of a trainer is likely to much consistent throughout.

Real world challenges and constraints:
a> The solution should be able to give good results in live stream

Also, the solution should be able to point out mistakes and provide suggestions. The quality of pointing out errors and providing suggestions is itself subject to improvement

The approaches possible:
I can collect many data , that is , many video clips of a particular exercise, manually label them , correct and not , under guidance of an expert . Then, pose it as a classification problem too , in addition with dtw similarity.
With respect to classification idea, I can take a query point , can calculate average dtw similaity with a number of correctly performed exercise clips , and use it as a feature . Another feature can be fourier transform. The idea of this feature will be important especially in exercises where speed of execution is important.

Credits
https://arxiv.org/pdf/2006.11718.pdf
https://mobidev.biz/blog/human-pose-estimation-ai-personal-fitness-coach

https://www.infoq.com/articles/human-pose-estimation-ai-powered-fitness-apps/
https://www.readcube.com/articles/10.1093%2Fcybsec%2Ftyaa021
https://medium.com/towards-data-science/just-used-machine-learning-in-my-workout-ff079b8e1939
https://medium.com/walmartglobaltech/time-series-similarity-using-dynamic-time-warping-explained-9d09119e48ec

DATA PREPROCESSING AND IMPORTANT FEATURES

Our task involves tracking real time exercise performance of the user and giving him/her feedback about his/her performance.
This involves capturing the live video of the user for a given window f time and then outputting the feedback.
The data to our solution is video in nature, which is basically a series of images.

How to use this series is an obvious question, we need to tackle.
In other words, what should be our features?

One way is to use images in their raw formats.
A possible solution based on such usage of data is to compare the image of user to pre-stored image of correct execution.
But, in this approach , we will take the whole image as input, which will have many areas in which our interest does not lie. Like, alongwith user, an image will have other objects in background. So, we are not fully focusing on areas of interest which involves only user.

There is more to it. Our area of interest can be deeper than just the part where user is. What we are interested in, is the movement of limbs and joints. So, we need to extract features dealing with the same. How can we do that?

We can make use of pose estimation, here. Pose estimation model will give us coordinates of required joints and other body points and from that, we can derive our required features.

Possible pose estimation models to use:
Movenet, OpenPose, Mediapipe(BlaisePose).
I have experimented with Movenet and Mediapipe both.
I found Mediapipe more appropriate as it tracks more number of body points. It also seemed more accurate by the fact that the rendered limbs in the images while taking live video were less flickering in nature. Also, mediapipe is a 3d model unlike movenet which is a 2d model. Movenet lightning model is faster but even mediapipe does a decent job when speed is the criteria.

Mediapipe is designed in a way that before final pose estimation the images go through intermediate processing, which saves developer a lot of hassle.
It uses BlaisePose as pose estimator.
In the following picture, which shows mediapipe structure, Gate is a very important component.
It acts as a filter. It passes incoming video through only when the landmarks found in the previous frame had low confidence.

In other words, as long as the landmarks found in the previous frame are good enough, they can act as region of interest for the next frame.
So, as long as confidence is high, the landmark detection model doesnot run, which saves time and leads to good speed.

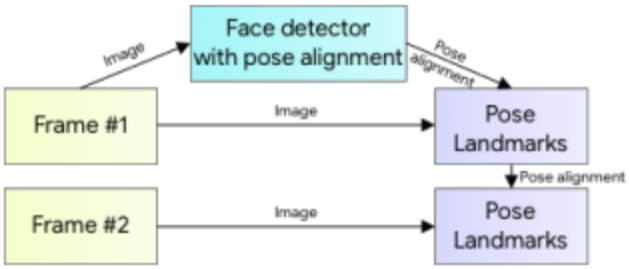


Figure 1. Inference pipeline. See text.

Fig 1:Source : blaze pose research paper

For the very first frame, it runs a simple face detector. By detecting the face, it approximates rough region where the person is.
It also runs a coordinate regression model which detects coordinates of each of the keypoints.
From the second frame, instead of running the coordinate model it tries to estimate how much each keypoint moved with associated confidence level. If the confidence level is not so poor, there will be no need to run co-ordinate regression model

Let me explain in more detail:

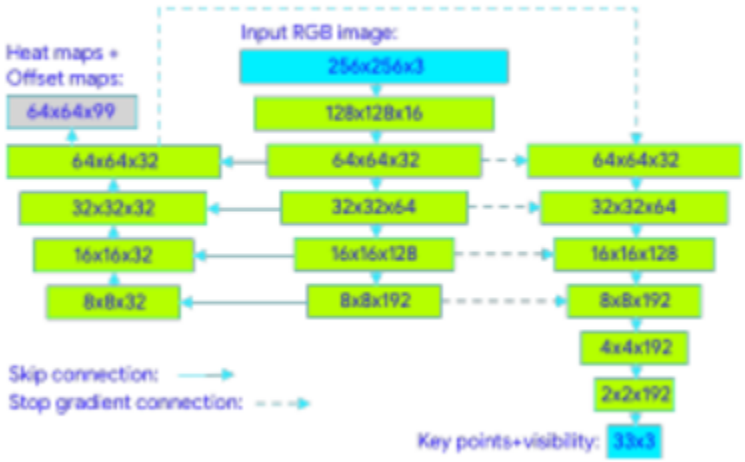


Figure 4. Network architecture. See text for details.

Fig2:Source: BlazePose research paper

In this diagram of network structure, we can see an encoder- decoder model with skip connections and the network at the rightmost side is coordinate regression network

Here is a better annotated diagram:

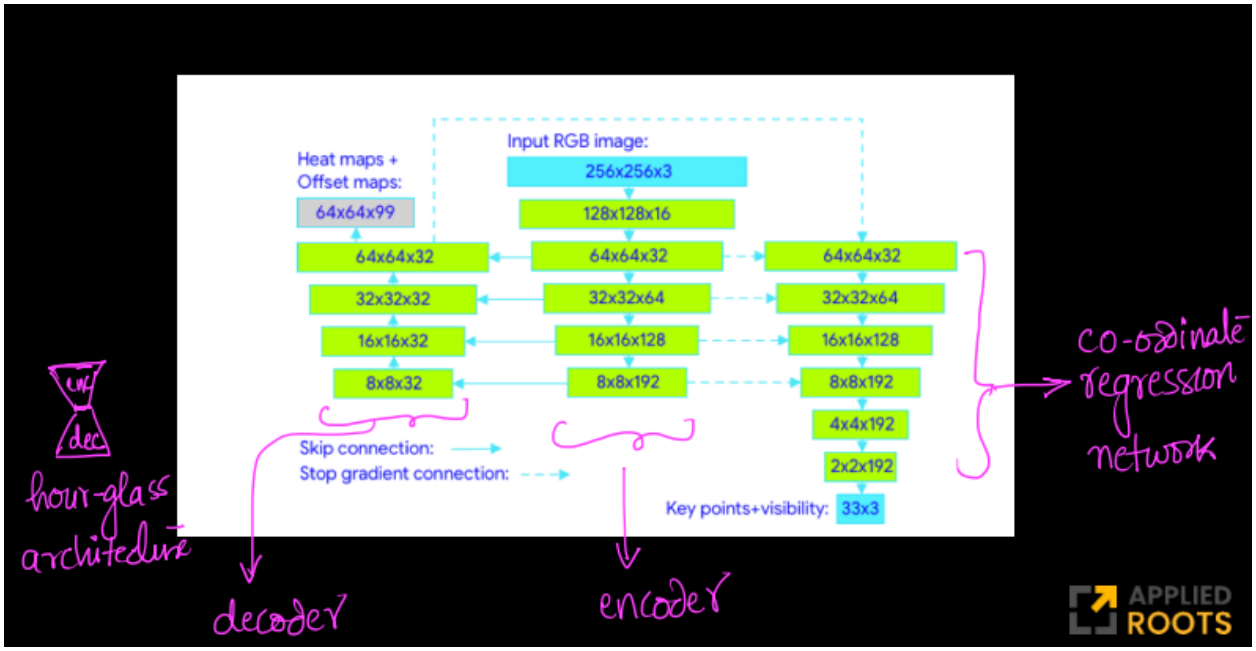


Fig3 :Annotations credit : applied roots

The hour-glass network structure takes input image of 256x256x3 and generates three tensors of size 64x64 for each of the 33 keypoints. This is why the output of decoder is 64x64x99 .
So, for a given keypoint, like left knee,
there are three tensors which can be called heatmap, offset1, offset2.
The heatmap can be understood as the following image:

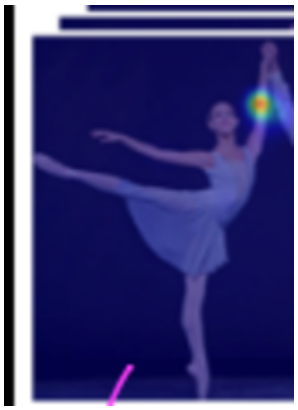


Fig4 :Source: cropped image from PoseNet research paper

That is , for left elbow, the most reddish part is the part with high confidence for the left elbow being present there.

Other two tensors, offset 1 and offset2 informs the network about how much each key point has moved from their coordinates obtained in the previous frame.
So, by heatmap, if the network is informed of low confidence for any of the keypoint, then only for the current frame is the coordinate regression network is run.
In Fig3, that of network architecture of blazepose, notice the stop weight gradients.
What are significance of these?
The signal can flow only forward and not backward through stop weight gradients .
That is, the weights in the coordinate regression network are trained independently of the weights present in hour-glass network.
It means that the regression network uses the features outputted by hour-glass network as inputs.

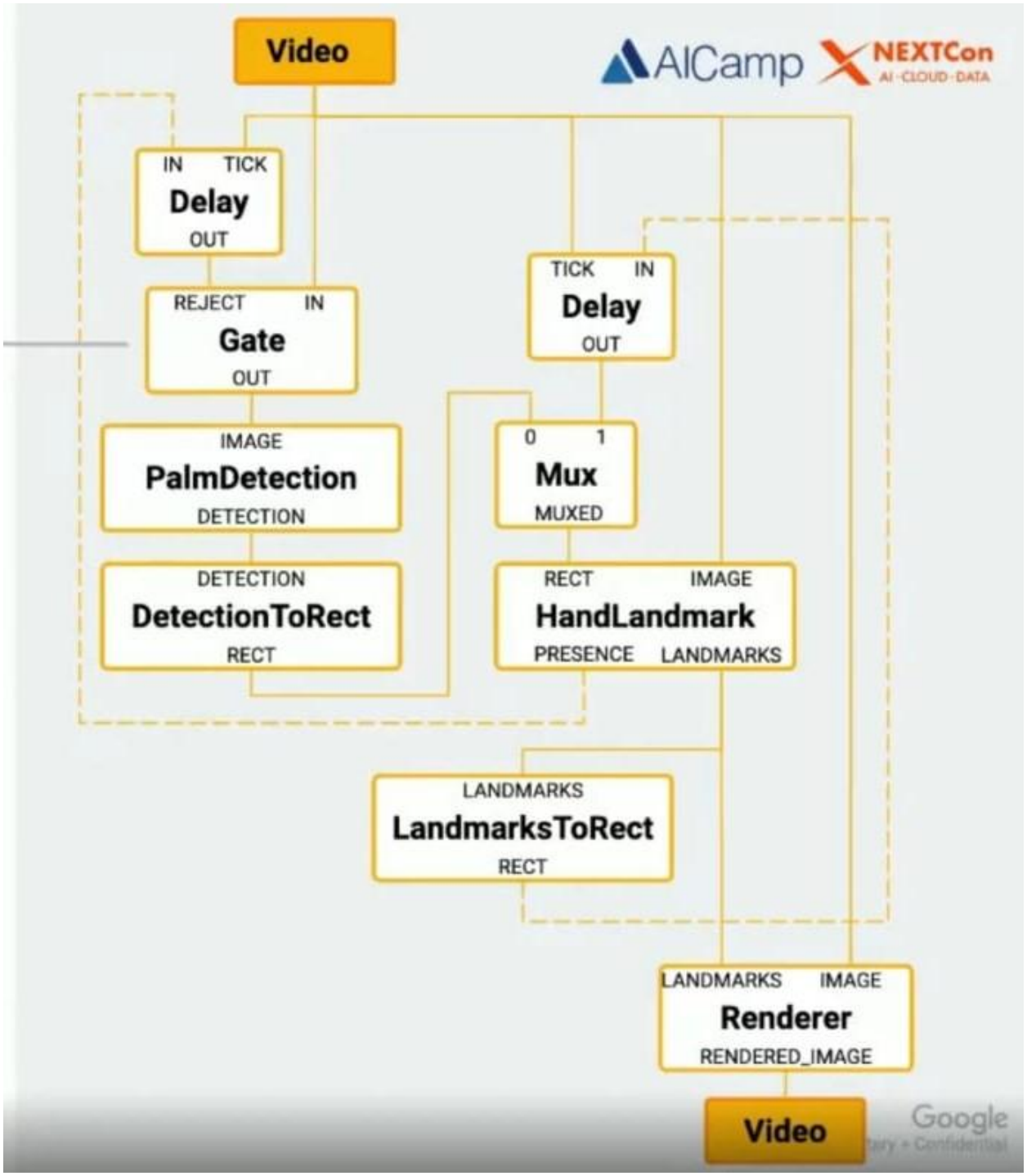


Fig5 : source: youtube:' mediapipe discussion'

For our solution, I am using Mediapipe.

Normalization:

Even using the pose information, subject-specific information such as limb-length , which depend on user's body structure , remain. This is problematic as it will not lead to generalization.
That is if the training data consists of individuals of same body structure , it will test poorly on users with different body structures.

To tackle this problem, we must normalize the coordinates of joints.
It involves centering , with respect to a reference point. The reference point chosen is the midpoint of left hip and right hip. And then, we do scaling, which involves division by a standard length. The standard length selected is body length. The approximate body length is calculated by first calculating torso-distance (the distance between midpoint of hips and midpoint of shoulders) and multiply it with a constant . The constant taken is 2.5

What does division by body length ensure?
It ensures that any distance between two points remain less that 1. That is we converted every distance relative to the body length.

Why did I choose body length to normalize?
There are other standard lengths which can be used to normalize.
But, in my research , I found it was suggestible to use the body length for practical purposes.
Reference: google ai blog

What can be good features given coordinates of different joints?
Angles, distances can be some of features I experimented with.
However, in my research, I came across suggestions to use a distance matrix, in which each row and column corresponds to a specific joint and each cell consists of value of distance between the joints represented by the particular combination of the corresponding row and column.

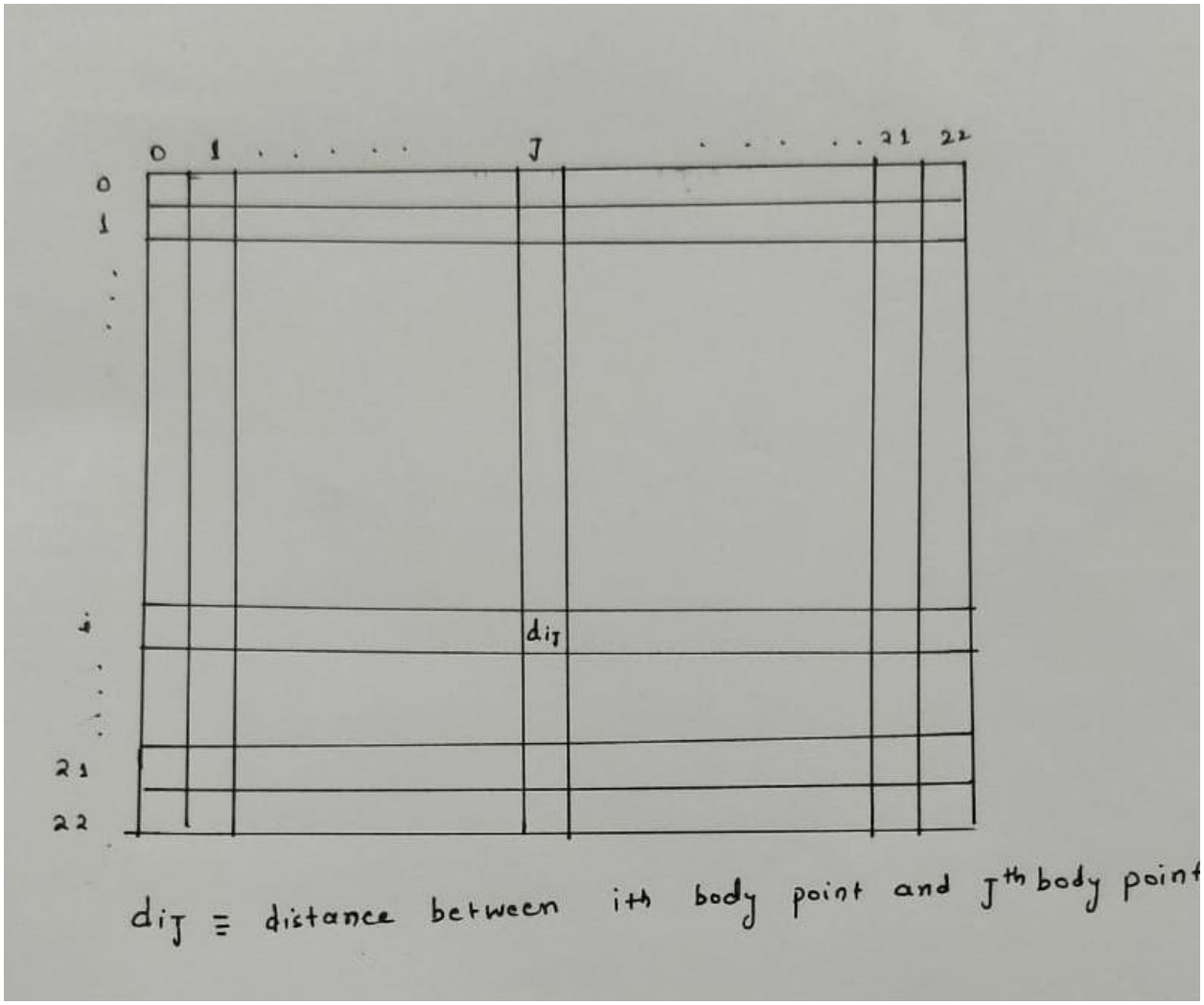


Fig6
This is distance matrix . Rows and columns correspond to the body points tracked.
This matrix is symmetrical in nature with diagonal elements as zero.

Why to use this feature?

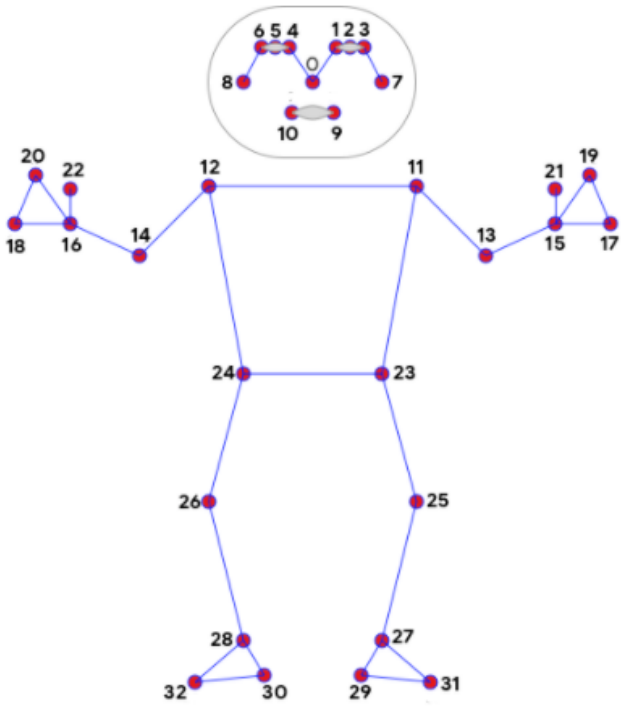
It is better to convert 3D pose information to a distance matrix as it is invariant to global translation, rotation, and has unique representation for each pose.

I am using 23 joints out of total 33 joints' coordinates given by Mediapipe.

The list is so:

1. Nose
2. Right Eye
3. Left Eye

4. Right Ear
5. Left Ear
6. Mouth (towards right)
7. Mouth (towards left)
8. Right Shoulder
9. Left Shoulder
10. Right Elbow
11. Left Elbow
12. Right Wrist
13. Left Wrist
14. Right Hip
15. Left Hip
16. Right Knee
17. Left Knee
18. Right Ankle
19. Left Ankle
20. Right Heel
21. Left Heel
22. Right Foot Index
23. Left Foot Index



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

Source: google.github.io

The distance matrix is symmetric in nature. The useful information can be extracted from either upper triangle or lower triangle.

So, for each frame, we have got a distance matrix and from that distance matrix , we extract the vector consisting of distances in the upper triangular part of distance matrix.

The size of vector is nC2, where n is number of joints.
For our case, n=23, so the size of the distance vector is 253.

Why did I use only 23 joints instead of 33 available. Mediapipe is a general pose estimator and outputs a total of 33 joints' information.
I have not used some of the joints' coordinates.
As my solution deals with a specific exercise, i.e, squats, the most important joints will be of lower body. The estimated points related to fingers are left out.

The extraction of features also depend on the model to choose. I am using resnet 1d, which is generally good at capturing the temporal nature between vectors.
It is noteworthy that for our video data, there is temporal nature between the poses of frames.

DATASET

The data I am using has videos with 300 frames. All videos have the same individual performing squats.
Each video belongs to one of the seven classes.
0: bad back round
1: bad back warp
2: bad head
3: bad innner thigh
4: bad shallow
5: bad toe
6: good

Fig7.Credits: waseda institute

Data collection:
For each video of 300 frames , I have collected 300 corresponding vectors . These vectors stacked up as rows in temporal fashion represent a

particular video. There are 300 rows and 253 columns of this matrix corresponding to a video.

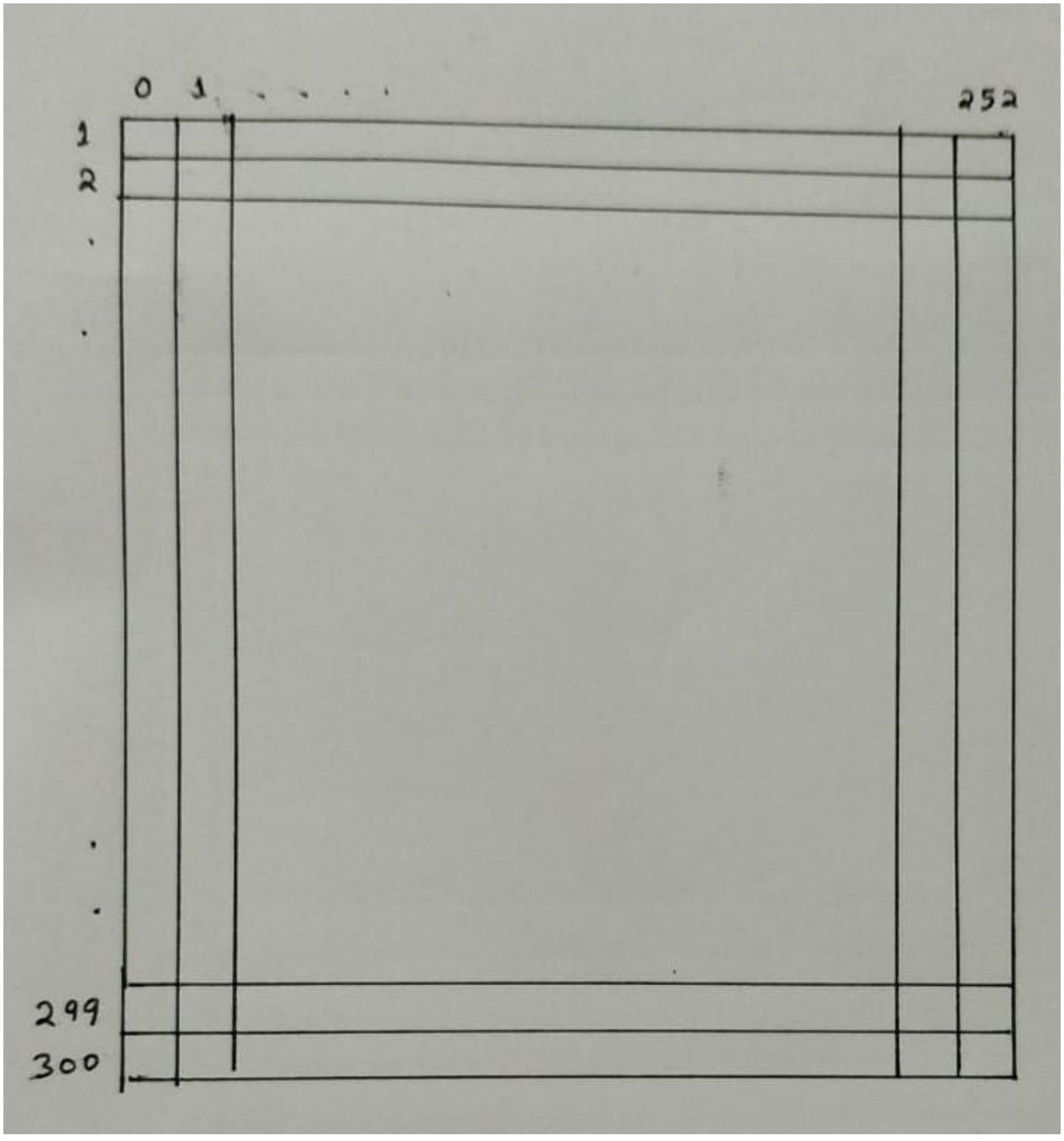


Fig.8
Rows correspond to frames ordered temporally of a video with total 300 frames.
Columns correspond to distances between a unique pair of body points.

There came some videos which had less than 300 frames. Among them , there were some videos which had only 149 frames, which resulted in matrix of 149 rows . Given repetitive nature of exercise, I kept stacking these 149 rows untill the total number of rows reched 300.

For the frames which had only a two or three frames less than 300, I simply appended the last occurring vector till the total number of rows reached 300.
Also, at the time the solution is used, one specific feature has been used extensively. That feature is angle formed by hip , knee and ankle.
This is used to determine, if a person is standing or is in movement.
One simple way to determine this using the angle above discussed is to check if for a given frame this angle is greater than a threshold such as 150 degrees. If true, then treat the person in the frame as standing.
The problem is that it is difficult to map one frame's angle information to the real time event of standing or movement. We need to define a window in which multiple frames' information gets stored for a certain period and we base our decision on statistic related to this window. That is for a given window of some frames, we will check our condition for the average in this window.
If it is to mark a person as standing or not standing , the above method would suffice. However, I needed a solution which can give information about start of exercise.
When camera opens and live feed starts, the user will not start exercising immediately. Initially , he/she will be standing in front of camera and it will be sometime after when he/she starts bending the knees. There should be some method which could give right estimate of whether the user has started decreasing the angle formed by thigh, knee, ankle.
One method is to use Optical Flow to detect motion.
If we can use optical flow to determine whether the user has started moving downwards then we can make the application aware of the same.

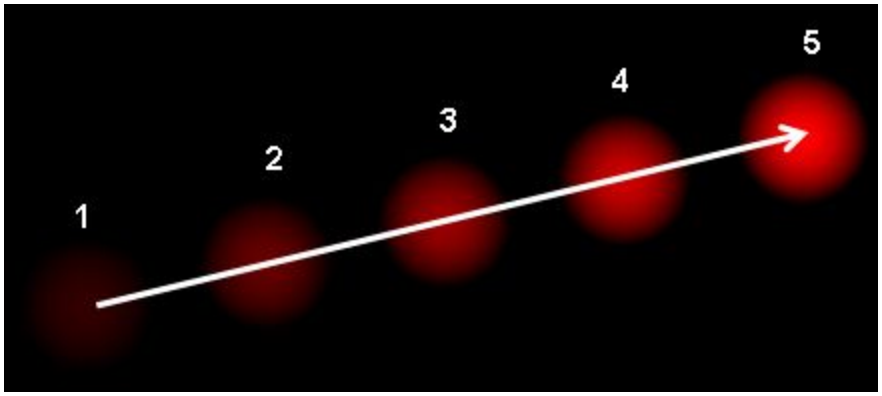


Fig9. credits: wikepidea

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.

The reason I dropped this approach:

It was relatively a complex approach.

Its latency was not as good to work with my problem.

I needed a very simple yet reasonably well solution.

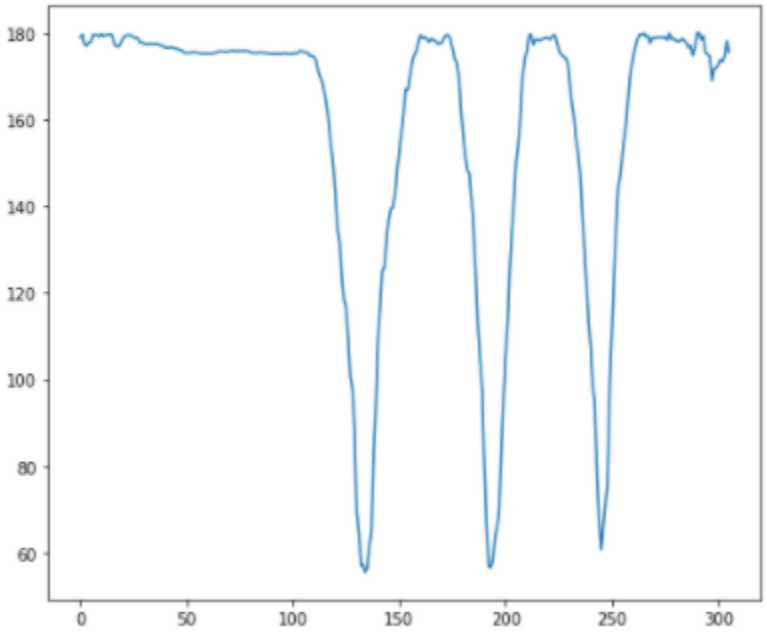


Fig10. knee angle tracked with window length 5.

This curve refers to tracking of angle formed by left hip, left knee and left ankle with smoothening by taking window of length 5. Note the relatively straight line in the beginning. How does that part differentiates from the rest of the part when motion starts?
A simple answer came out to be slope of the curve.
When the exercise starts , slope starts decreasing .

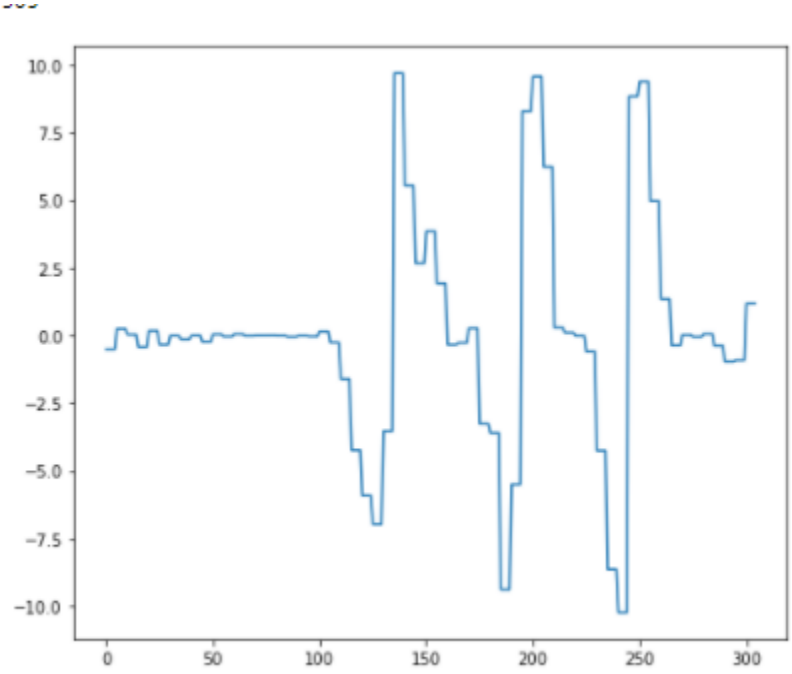


Fig11.Slope tracked in above curve

So, I came up with the method to record this angle for every frame till the window size, then using linear regression to estimate a line which should ideally pass through most of these values in a given window and then, to analyse slope of this line. Initially, when the person will be standing , the angles will be more of less of same values and when the person starts bending, the slope of the line estimated for all the angles in a window will be highly negative. At this point, we can inform our system to start noticing coming frames for classification as the user must have began exercising. To make it more robust, we can also use the condition of angle being greater that 150 degrees for initial phase and notify the user in case it is not so. And then start keeping tracks and using linear regression over windows.

Notice that the above logic will also mark the user as standing at undesired stages. Notice the crests in the curve above. With a small window the above logic will mark those parts as the user being standing.
What am I doing to deal with this problem?
I am using the above method only once, i.e for the first instance the user will bend knees and start exercising . After that I am keeping track of same angle for a larger window and using the error of straight line fit over that window.
More detailed explanation of the method:
What if the user suddenly stops exercising after performing some reps. The solution must know to again start avoiding coming frames after user has abruptly stop squatting and to restart observation of valid frames once the exercise begins.
For this purpose also, I have used linear regression , but in different fashion. This problem is different as in this case, user was initially exercising and suddenly stops. The solution is to define a window and keep track of the knee-angle and fit a line for it, and analyse the error of the fitted line. With slightly larger window ,(the size of window for this purpose is taken as 30, which is after some trials with different window sizes) , drastic difference were observed for the error while movement and error while standing. This happens because during movement, the angle doesnot change in linear fashion and so results in high error when a straight line is fit. A threshold can be easily used to differentiate between these two states.

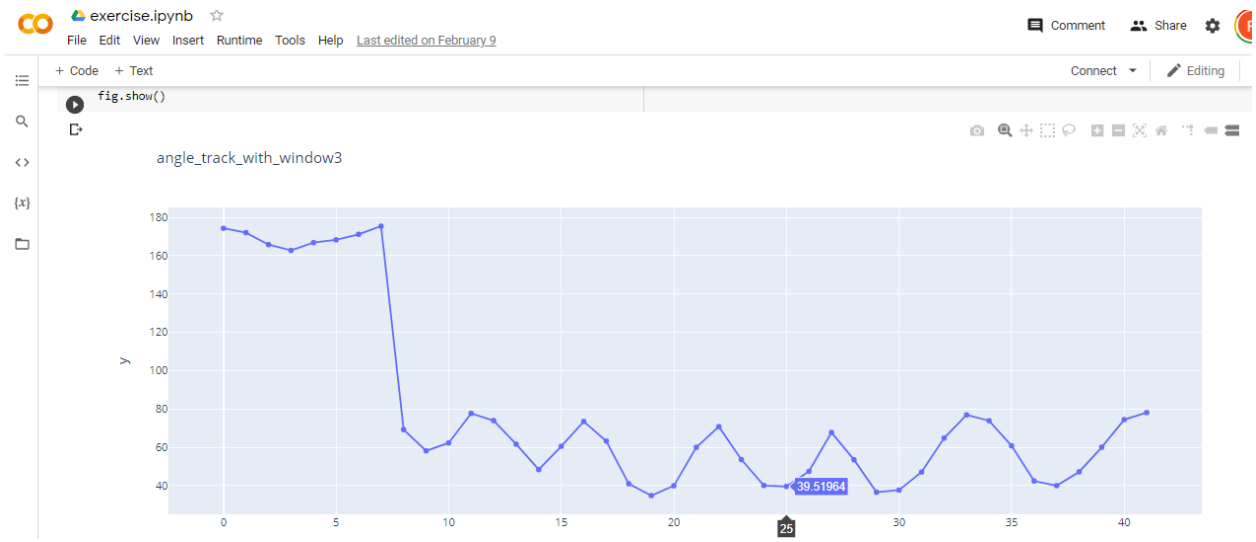


Fig12.Window approach leads to smoothening by ignoring some of the outliers and make it robust to base our decision on.

The reasons of using Mediapipe as pose estimator:

1. 3D : Mediapipe outputs 3dimesnional coordinates of points , unlike other pose estimators as Movenet or PoseNet
2. It is fairly fast: I have tested it. It works seamlessly on live data. It works around 30 frames per second.
3. It is robust and reliable: As mediapipe is already used in some reputable products such as YouTube , and Google Cloud Vision, it can be relied upon to give useful outputs in our case too.

I also experimented with Movenet as pose estimator, but I found movenet better as its accuracy was better and speed was sufficient.

MODELING:

For modeling, I have used resnet1d 34.
As far my research goes, Keras has no pretrained model for resnet , so I had to implement a ResNet 1 d 34 .

Why Resnet1d?
Our input data is a matrix where rows correspond to relative time and each column corresponds to a distance between a specific pair of points given by pose estimator.
To run classification on this data, I am using resnet with one dimensional convolutions. One dimensional convolutions capture temporal connections between information of frames. Note that information corresponding to a frame is a row in the final output of size 300x253. I have used 1 dimensional convolutions in an NLP task where words are represented in vector form and there is order in which each word occurs in a given sentence. Drawing my motivation from some research papers and analogy of the task with previously done assignment , I decided to use 1 dimensional convolutions for the task of video classification. Given a proven track record of good performance of resnet, I decided to use a simple resnet34 as the architecture for my task.
The goal of training a neural network is to model a target function f(x). If we add the input x to the output of the network, the network will be forced to model g(x) = f(x) - x rather than f(x). This is called residual learning.
So, if weights of some intermediate layer fails to learn useful value and outputs zero , then the skip connection will make the output same as its input like an identity function, ensuring that the learning doesnot ceases.
The skip connections ensure that the input signal can easily make its way across the whole network. The deep residual network can be seen as a stack of residual layers, where each residual layer is a small neural network with a skip connection.
Credits of my knowledge about Resnet:
Applied roots,
O’reily hands on Machine learning
Yannic Kilcher : you tuber
Research paper => https://arxiv.org/pdf/1512.03385.pdf

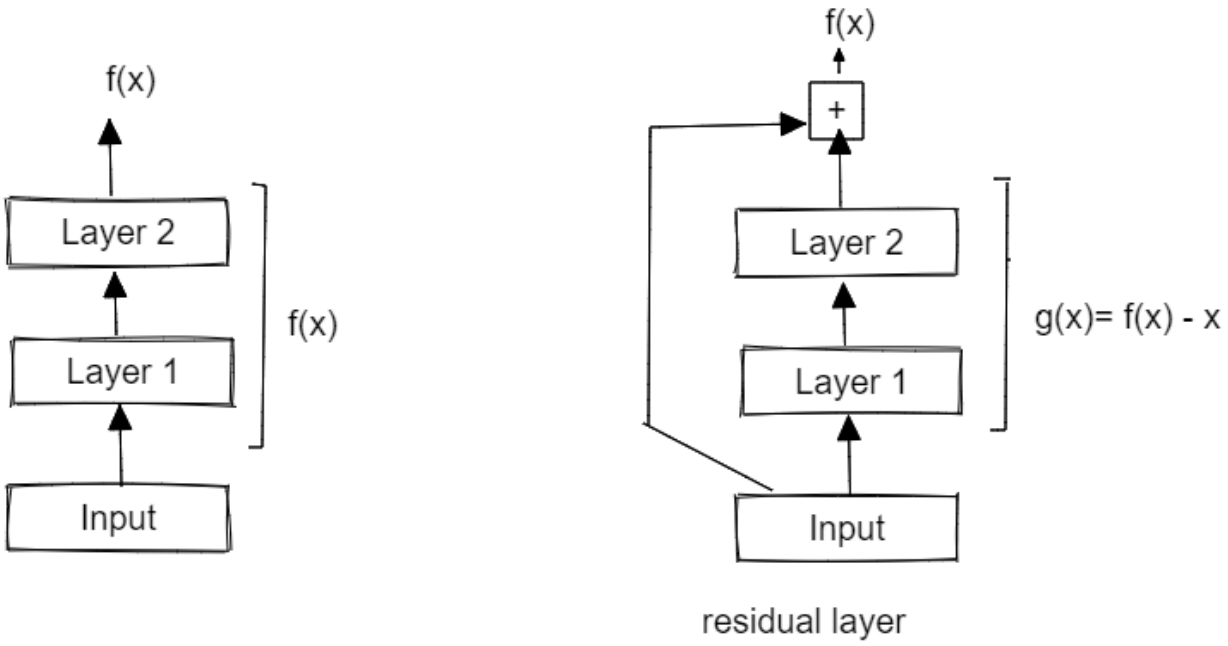


Fig 13.Credits: O’reily hands on machine learning

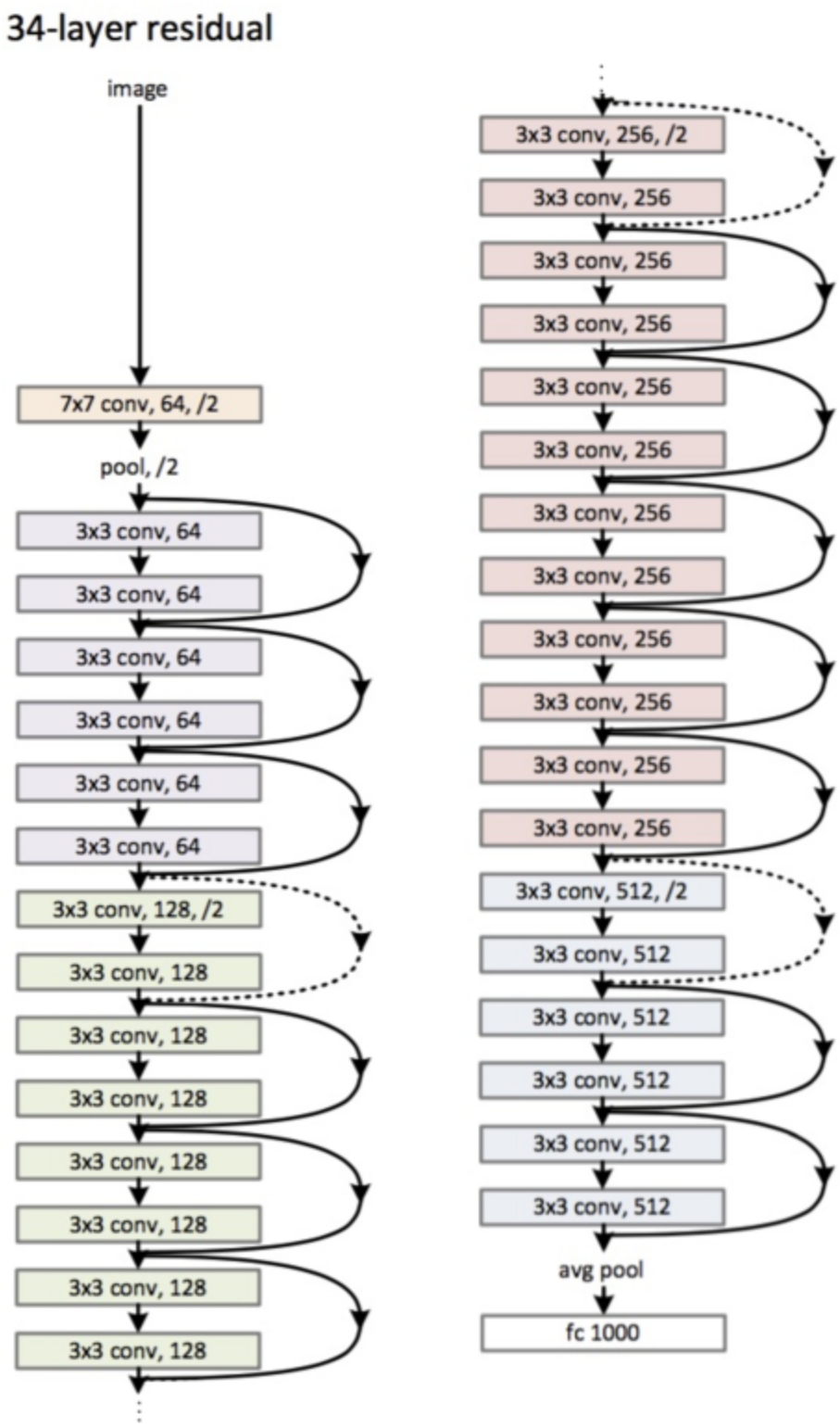


Fig.14. Resnet34 with 2d convolutions.
Source: analytics Vidhya

The changes I had to do in the above architecture was to change the input size to 300 x 253 (300 frames and 253 unique distances) and to replace the 2d convolutions with 1d in the ResLayer.

Most of my ideas are motivated from this research paper.
https://tinyurl.com/2p9uzmuc

I also implemented the architecture proposed by them which was like this:

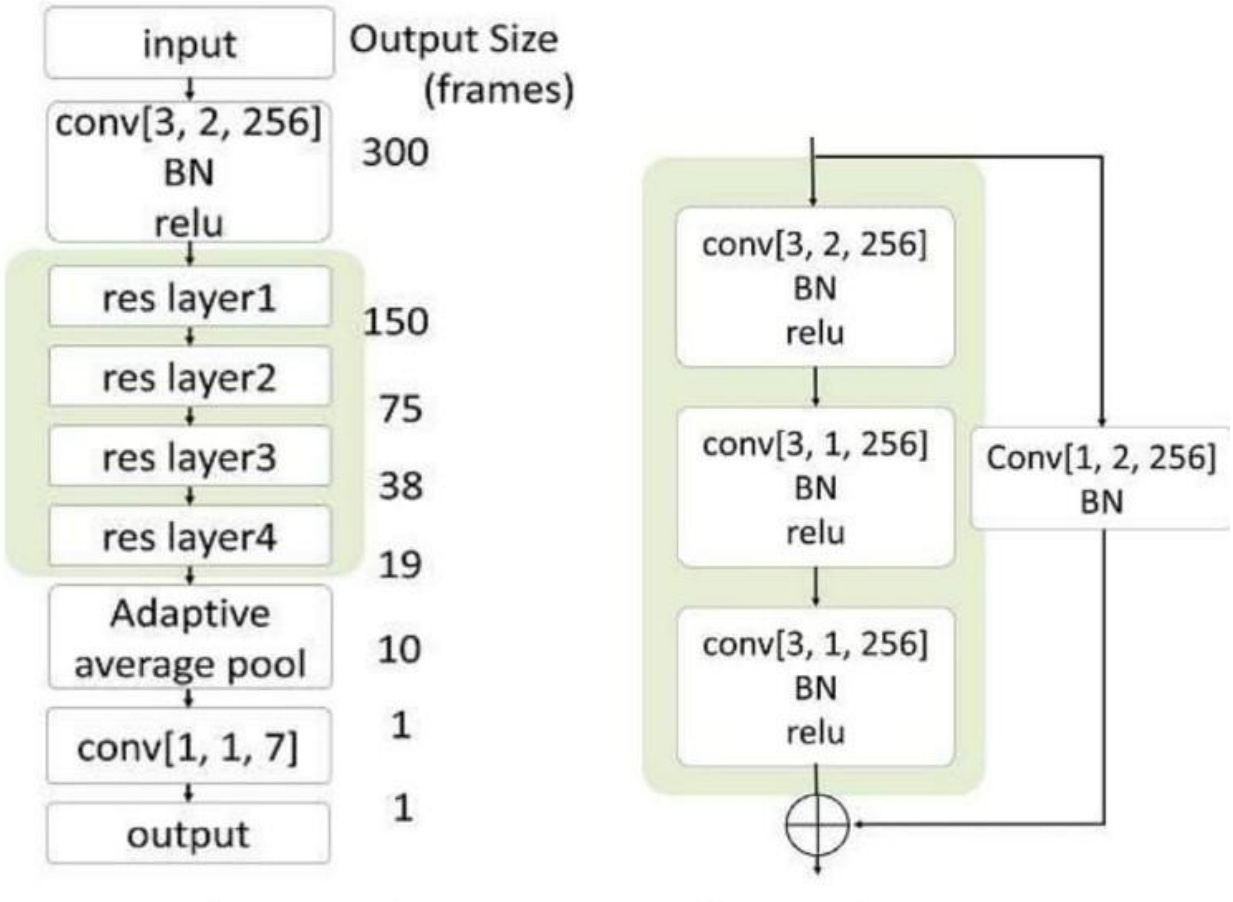


Fig.15.

Earlier when I was using Movenet (https://tinyurl.com/yc6pj3jp) for pose extraction, I used this architecture for video based squat classification. Since I didnot want to copy their proposed model , later, I decided to simply implement Resnet34 . The hope is that it will start processing the frames

with temporal nature and will generate useful features amenable for classification.

Data Augmentation.
My model's test accuracy is coming near 68% with crossentropy loss as 0.8427 and training accuracy came out to be 77 % with cross entropy loss as 0.598
The model seems to be overfitting. How can we genearalise better?
One way to tackle this problem is to augment data .
Right now, I have worked with only 2054 video clips and there were 7 classes. Data was not imbalanced, and so average clips per class happens to be close to 293.
I proportionately divided my data to form training part, validation part and test part.
One way to augment video data is to slow down its speed by upsampling. That is given two frames in a video, we need to generate an intermediate frame in which the information represents an intermediate state between the states represented in the given two frames.
The above explanation is merely an example and not in-depth explanation of upsampling.
This way, with same frame rate, the video will appear to proceed slower.
Contrary to above method, there can be approach of downsampling.
To explain it in a simple manner, in downsampling, we will get rid of some of the intermediate frames in orderly manner and so the video resulted will appear to proceed faster.

The problem happened was that in my case, the video size representation in array form was too big and augmenting that kept crashing my RAM.
So, it is still a work in progress.
There is a very helpful library to achieve above methods of augmentation.
https://github.com/okankop/vidaug

I found this library very late in my research.
This is the library I am using to augment my data.

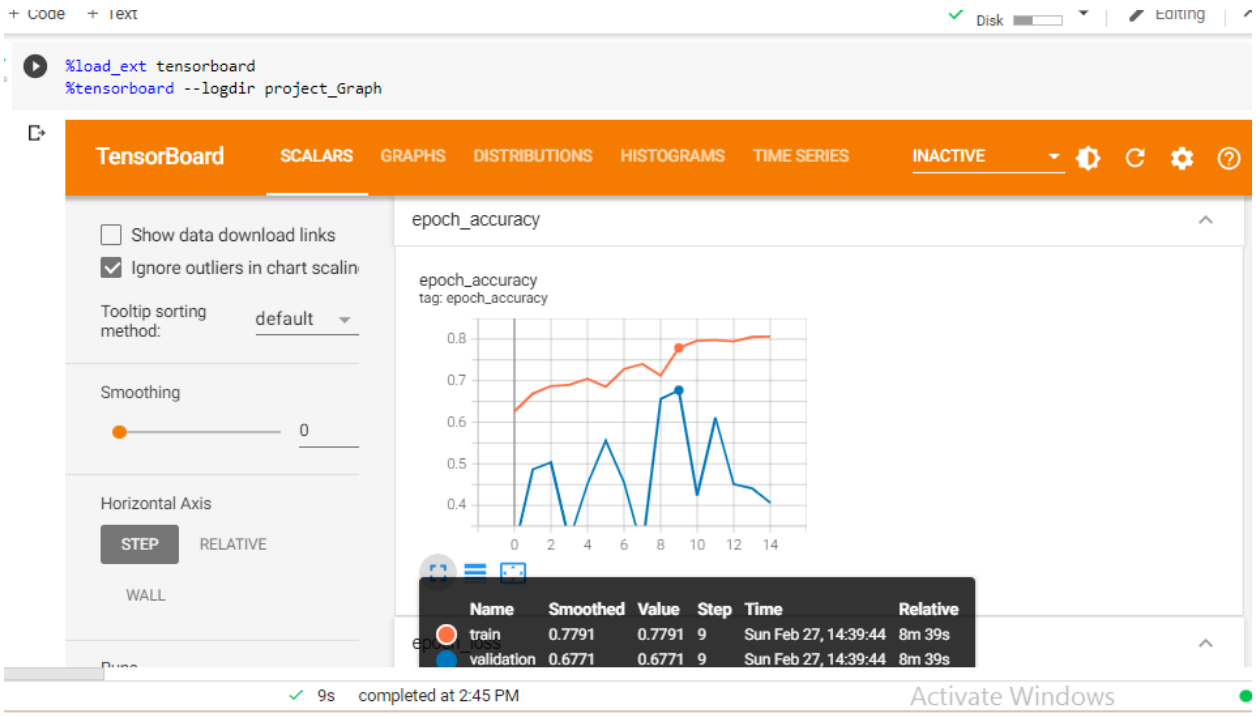


Fig16: accuracy vs epochs of my resnet34 model for squat classification

The final model used corresponds to the particular accuracy highlighted in the curve of accuracy vs epochs in the above screenshot.

It would have been better if I used CSVLogger as checkpoint and used weights corresponding to slightly less validation accuracy which had less training accuracy as well.

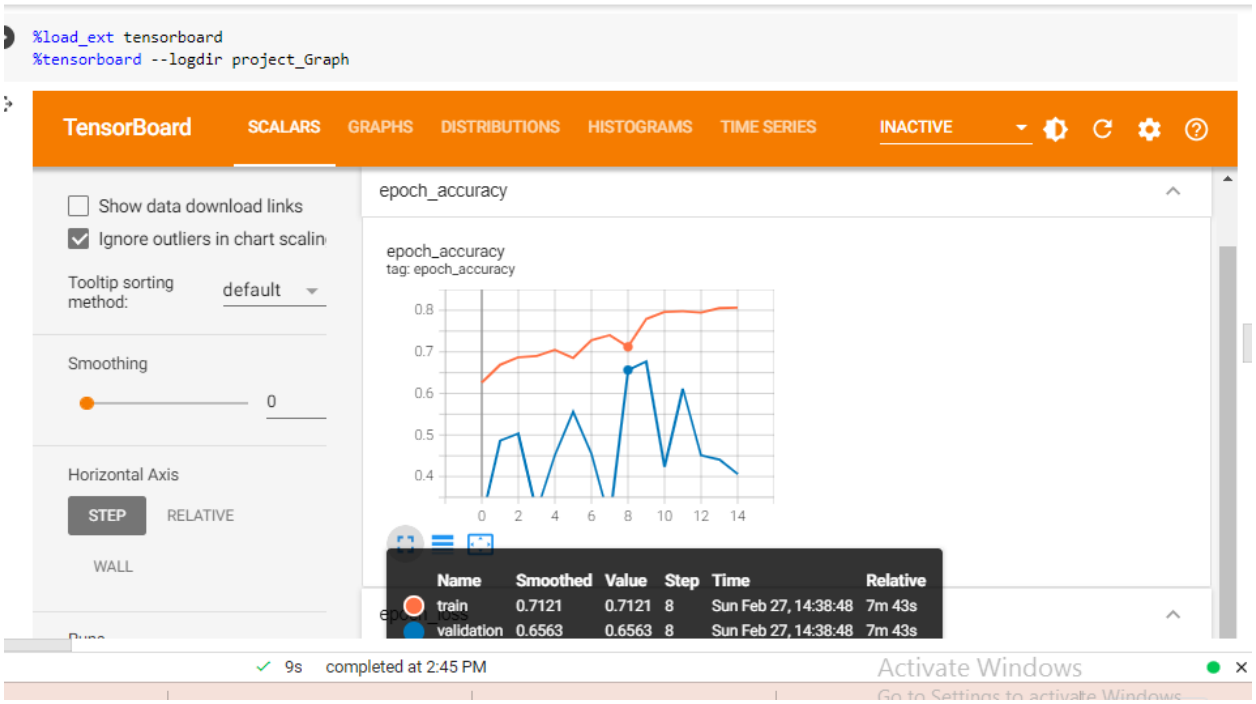


Fig 17

The loss curve vs epochs curve is as follows:

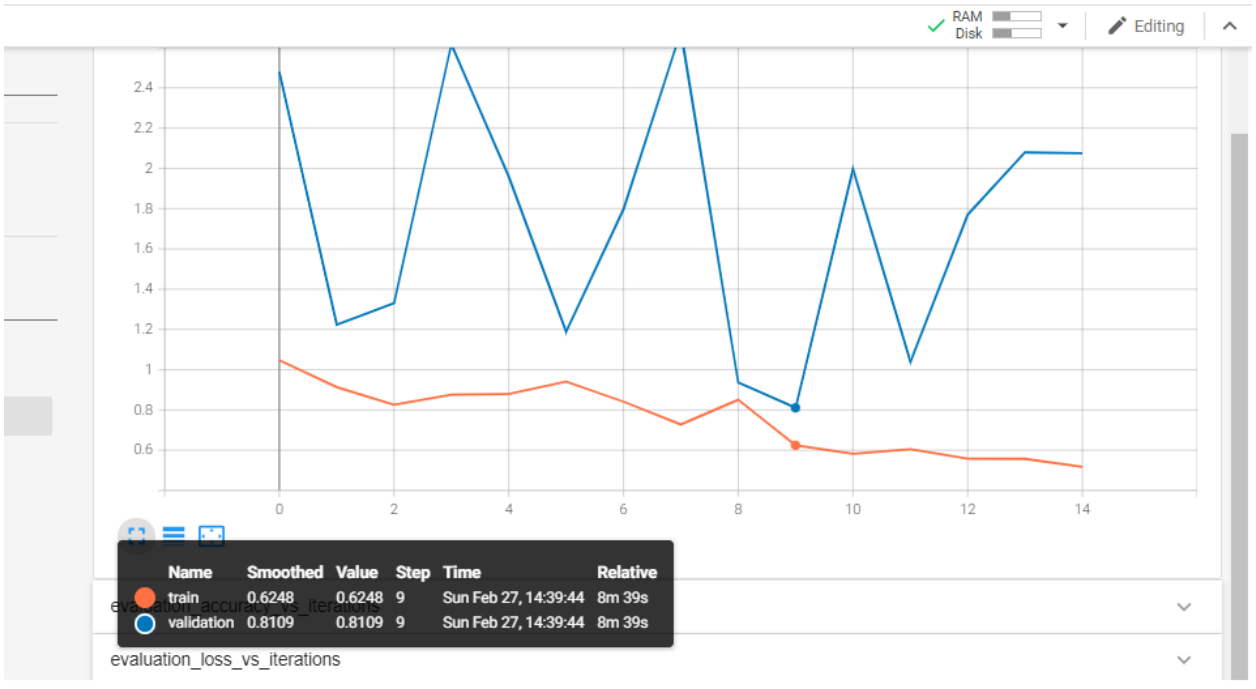


Fig18: accuracy vs epochs of my resnet34 model for squat classification

For classification tasks, confusion matrix is very helpful

Let us look at the confusion matrix related to performance of the final model on test data

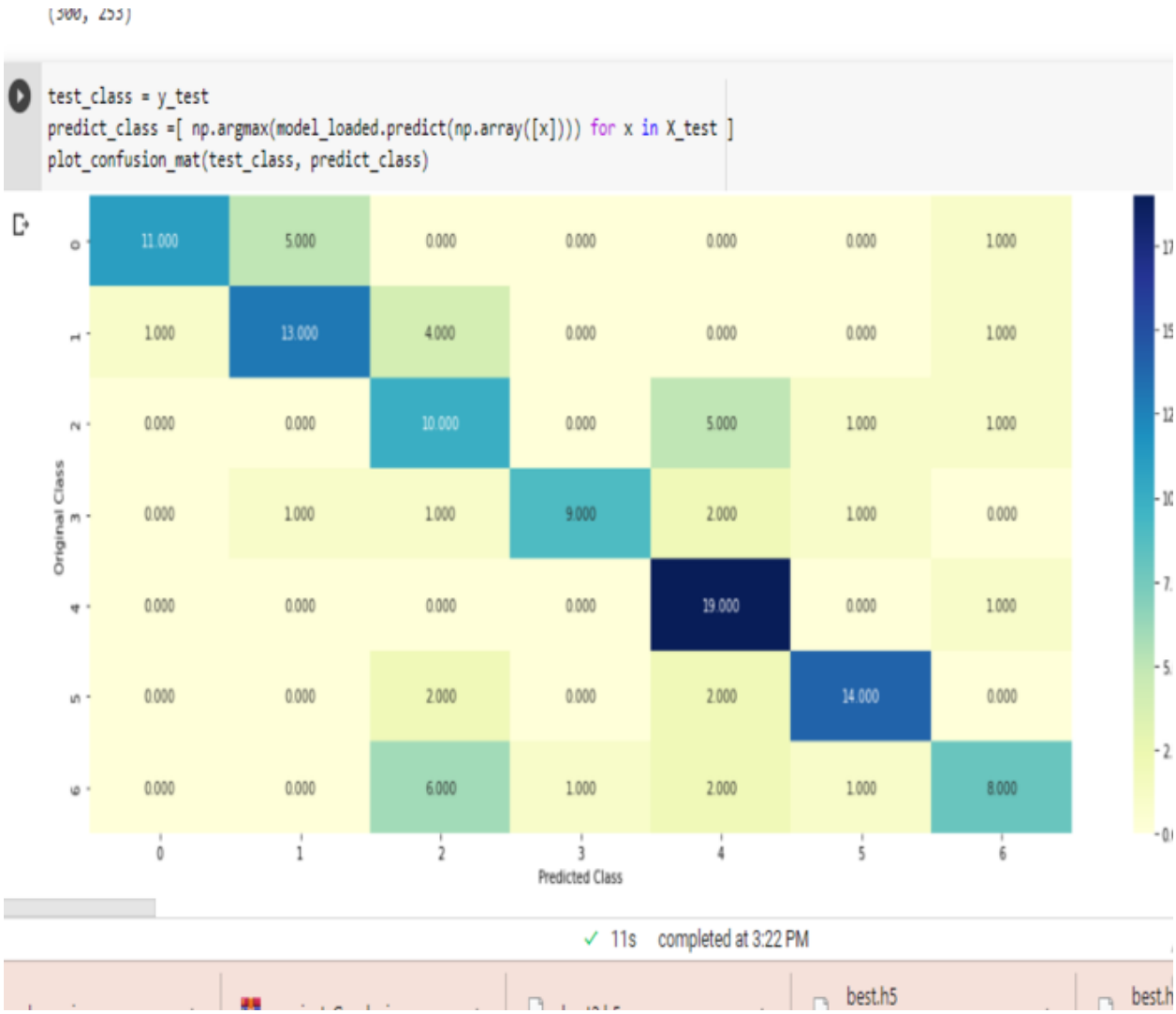


Fig19

{0:'bad_back_round' , 1: 'bad_back_warp', 2: 'bad_head', 3: 'bad_innner_thigh', 4:'bad_shallow', 5:'bad_toe',6: 'good'}

We can see the model confuses between class 0 and class 1, as it predicts 5 points belonging to class 0 as class 1.
We can perhaps see the reason as both class0 and class1 has issues related to the back.
We can see that the precision for class6 is okay, however recall is not that good for class6.
[class6 means “good performance of squat in the given video”]

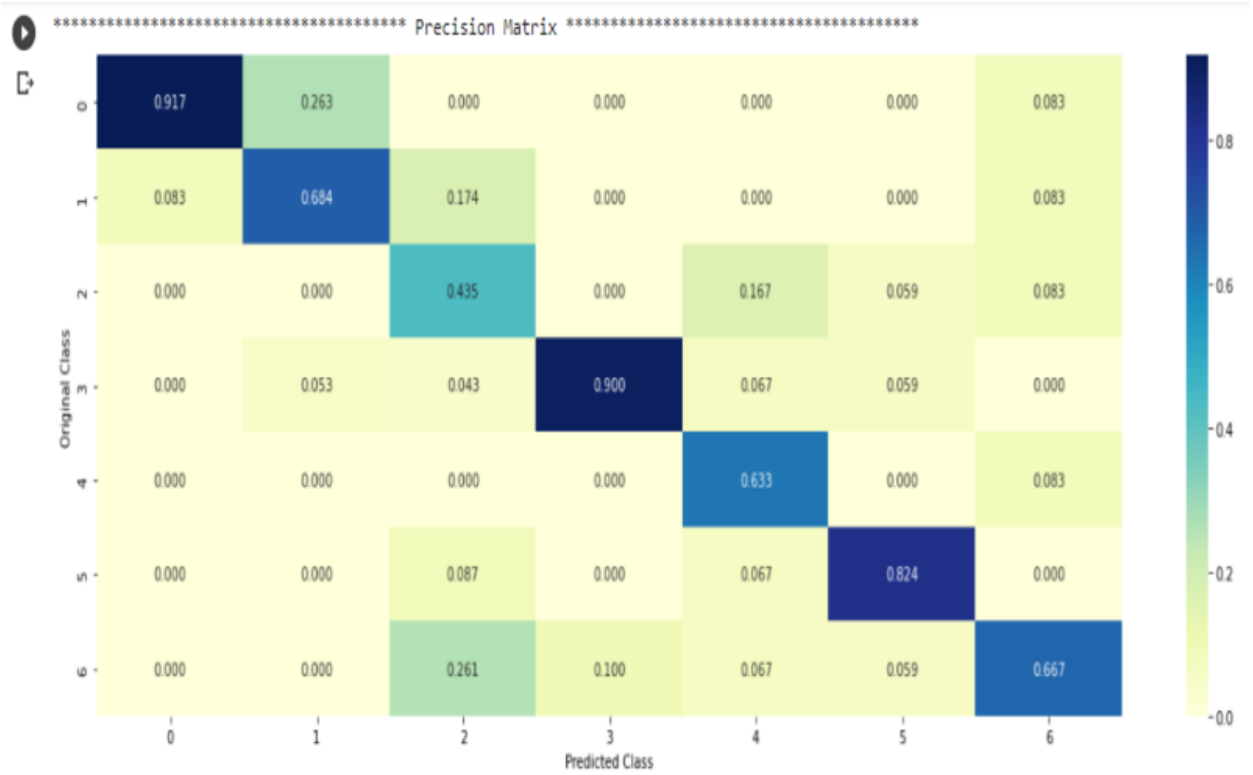


Fig20
The values are in percentage expressed as ratio.

We observe that precision is good for most of the classes.
Exceptions can be class2 and class4

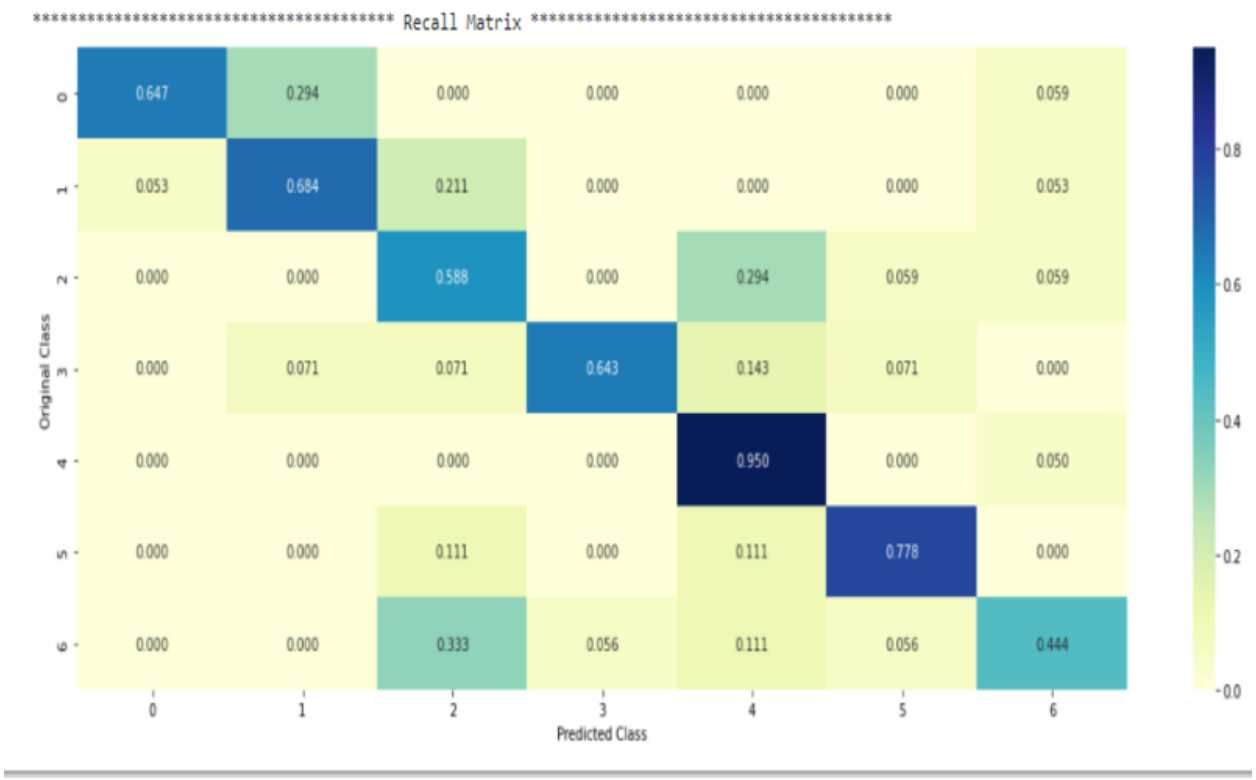


Fig21

We see only one impressive value for recall here and that is for class4.

Confusion matrix on train data:

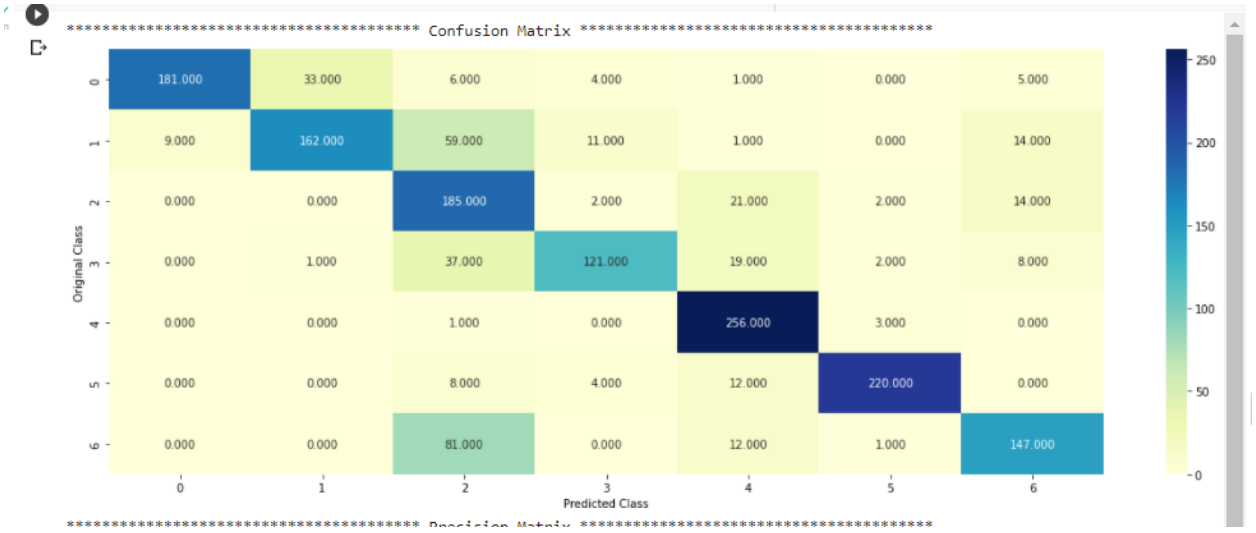


Fig22

Precison matrix on train data:

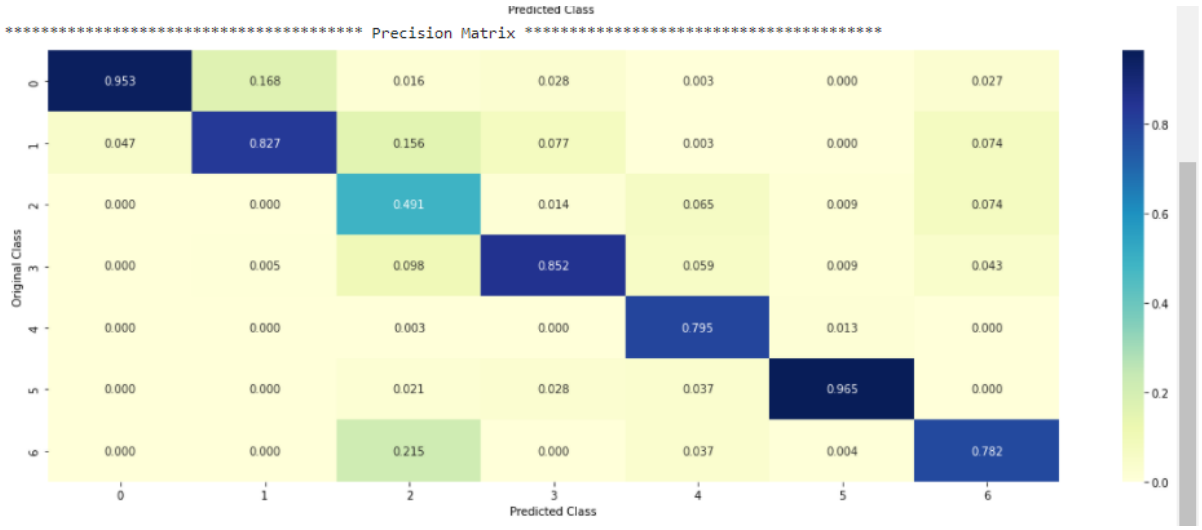


Fig23

Recall matrix on train data:

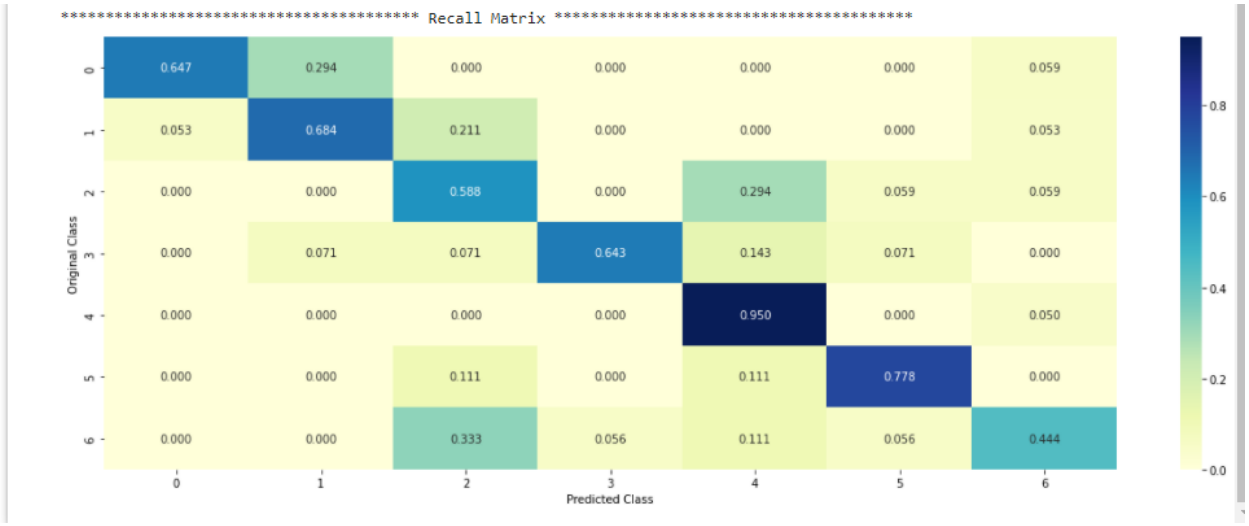


Fig24

We can observe same trends in performance of the model on train data as that on test data.

That is precision for class6 being okay but it recall getting poor.

The model confuses between class0 and class1 as it classifies some of the data belonging to class0 as class1.

A practical problem:

When testing in real time, giving feedback can be problematic.

Possible approaches to give feedback are:

1.Displaying the feedback on screen
2.Giving feedback in audio format

The problem I encountered:
The code of audio/visual feedback will arrive in middle of execution and so , it will appear only for so less time that it is of no practical importance.

The solution is to use daemon thread to play audio feedback.
Daemon threads are always going to run in the background that provides supports to main or non-daemon threads.
The daemon thread does not block the main thread from exiting and continues to run in the background
Credits : realpython

Basic Structure of my solution

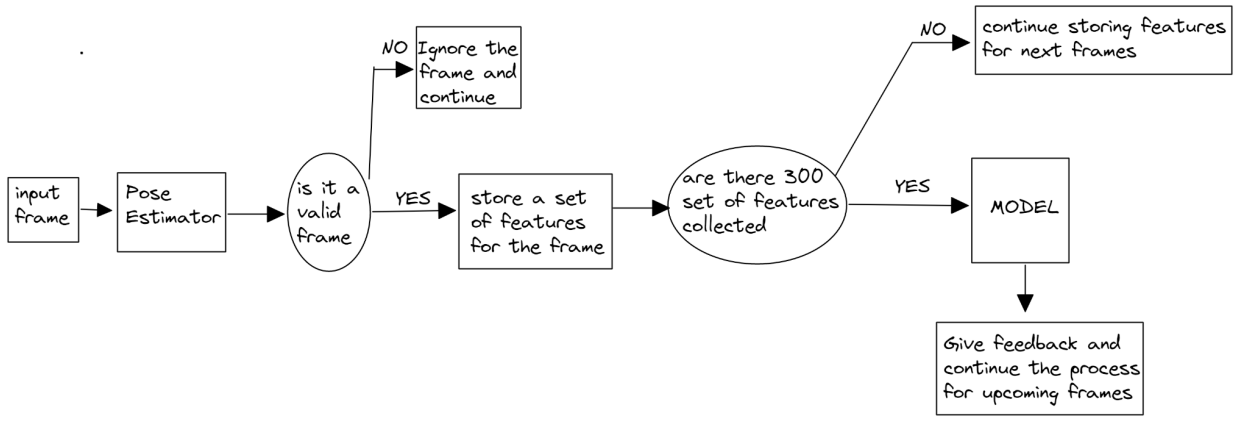


Fig25

Other ideas and Future Work

We should also try RNN based architecture on our data given its temporal nature.
A possible improvement will be to construct a solution which is more dynamic, which is not fixed for a given number of frames. If an error occurs, the solution should immediately point the error right away.

My current solution can be altered in a way to achieve fast result to reduce latency to an extent. That is if I only collect features for the first 100 frames and repeat them to obtain a required sized input tensor, then latency cann be improved.
Will repeating be logical. I have no concrete evidence to support it, but given the repetitive nature of the exercise, I would argue that repetition would not be a bad idea.
However, it still bounds the solution to a fixed sized input.
The aim should be to provide as dynamic a solution as a human trainer.

Credits:

Applied roots
https://arxiv.org/pdf/1512.03385.pdf
https://google.github.io/mediapipe/
http://hi.cs.waseda.ac.jp/~ogata/Dataset.html
https://openaccess.thecvf.com/content_CVPRW_2019/papers/CVSports/Ogata_Temporal_Distance_Matrices_for_Squat_Classification_CVPRW_2019_paper.pdf
https://medium.com/towards-data-science/just-used-machine-learning-in-my-workout-ff079b8e1939
https://pypi.org/project/playsound/