# Just Dance Everywhere: Using Deep Pose Estimation to Get Your Groove On!

Adi Ojha
University of Texas at Austin
adiojha629@utexas.edu

Kevin Wang
University of Texas at Austin
kevinwang.1839@utexas.edu

Yunong Liu
University of Edinburgh
yunong_l@utexas.edu

Mingxuan Zhao
University of Texas at Austin
ming.zhao@utexas.edu

## Abstract

*On November 17, 2009, the world was taken by storm by Ubisoft's hit game Just Dance. 13 years later, the game is still restricted to TV-connected gaming consoles. That changes this semester. We created a laptop version of the game powered by Deep Pose Estimation models [1]. Our game estimates the joint locations of users and Just Dance dancers and compares their relative position and motion to calculate game scores. This game is run on a laptop with no extra computational resources or an HD video camera. We developed a novel scoring algorithm that allows users to see their scores in real-time, just like the original game.*

## 1. Introduction

Ubisoft's game Just Dance has become a video game classic. In the game, players get to "jam out" to their favorite songs, following the moves of a dancer on screen, and are scored based on how close their movements are.(see Figure 1). This amazing experience has been confined to expensive video game consoles, which use 3D accelerometer data [2] to calculate how close the player is to the dancer's positions. However, with the invention and success of deep pose estimation, we wanted to bring the experience to everyone who has a laptop. Hence the title "Just Dance *Everywhere*."

It was important that our pose estimation be accurate yet, computationally lightweight so it could be run on a laptop. We developed a web application that uses MoveNet, a lightweight Google-made deep pose estimation model, to capture the angle between users' joints. We calculate these angles for Just Dance videos (found on YouTube) and compare the calculated angles to score the user. The application allows the user to play Just Dance, and see their score similar to the original game.

Figure 1. Just Dance. Notice that the dancer's left (reader's right) hand is blue: this means that only the player's right hand will be used for scoring purposes.

We believe our game offers many improvements from Ubisoft's original design. Firstly, this game is not tethered to video game consoles; any one with a laptop should be able to play our game. Secondly, many consoles only track one key point to calculate score. For example, in the Wii and Switch versions of the game, only the position of your right hand is tracked. That's why in Figure 1 the dancer's left (reader's right) hand is highlighted in blue. Therefore, players can get perfect scores by only moving the hand that needs to be scored. Our version will track several human body joints making it harder for players to cheat!

## 2. Related Work

### 2.1. Pose Estimation

Pose estimation is a well researched topic in computer vision. It is typically done by identifying, locating, and tracking human body key points, such as hands, elbows, knees, and even eyes and mouth. The introduction of deep networks for pose estimation was first done in 2014 [3], and since the efficiency and benchmark scores for pose estimation have been drastically improving.

Deep pose estimation methods are classified into two categories: top-down and bottom-up. Top-down methods start by first identifying human bodies and then running pose estimation within those bounding boxes [4]. In contrast, bottom-up methods analyze all pixels of the image or video frame for key points, and groups these points together to make up a person. Examples of this method include [5] and [6].

Top down methods allow researchers to limit the search space to a bounding box, but errors in the human body object detection propagate to the pose detection. In addition, the initial pass to create a bounding box adds to the latency of the model. In contrast, bottom up methods, do not rely on a possibly faulty object detection algorithm and have improved latency due to the lack of initial pass through to find the bounding box.

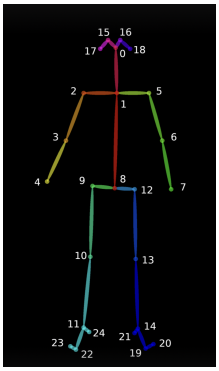


Figure 2. 25 keypoints body from OpenPose [7]

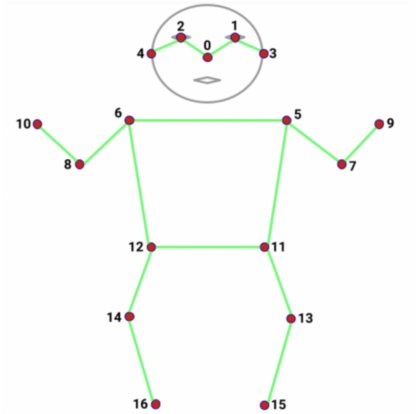


Figure 3. 17 keypoints body from MoveNet [8]

Below we list some state of the art pose estimation models for reference. We will be discussing OpenPose and MoveNet, as these are the models that we choose to use for our application.

- OpenPose - is the first real-time multi-person human pose detection library that is capable of detecting a total of 135 keypoints of the human body, foot, and face. It also supports 3D single-person keypoint detection. When it does 2D keypoint detection, it uses 25-keypoint body/foot keypoint estimation as shown in Figure 2. It is one of the most accurate models, but it also requires significant computational resources [9].
- AlphaPose - is an accurate pose estimator that has high mAP [6][10].
- MoveNet - focuses on making the detection ultra fast. It is able to detect 17 key points of a single person (see Figure 3) faster than 30 FPS [11].
- BlazePose- supports 3D pose detection. It utilizes depth order annotation to reduce depth error [12].

### 2.2. Pose Comparison and Scoring

Ubisoft doesn't release how the game scores players. What is known is that the game gives players feedback on their moves in six levels: Perfect (highest), Super, Good, Nice, Ok and X (lowest); X indicates that the player skipped a move or did not perform it well.

Given the lack of information from Ubisoft, we surveyed gesture recognition literature to identify several ways to compare human dance moves. In [2], the authors use the x,y,z accelerometer data from Wii controllers to build a data-driven Hidden Markov Model to classify various poses (drawing a square, circle, or the letter Z drawn in the air). [13] mentions several data-driven approaches that use supervised labeled data to classify different gestures for human-computer interaction.

Another approach to classifying movements is to compare a movement in question to an already labeled movement. For example, if you have a video of someone dancing the task is to compare this dance video to another one of the people dancing. Metrics such as cosine similarity and Euclidean distance have been used to achieve results [14] [15].

## 3. Methodology

We use the following pipeline for our Just Dance video game:

- We convert YouTube videos of Just Dance to mp4.
- We preprocess these videos using Open Pose [9] to obtain high-fidelity pose estimates on the video every second, and store the key angles in a time indexed csv file.
- At run time, we take the user's webcam feed and calculate poses using Google's MoveNet [12]. This model is less accurate but does not require GPUs as OpenPose does.

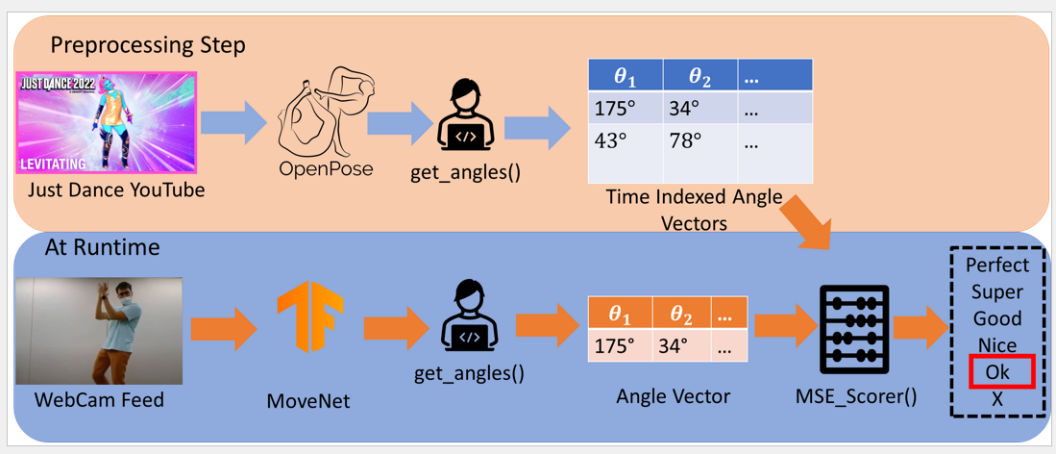


Figure 4. Overview of how we determine scores for the game.

- We compare the angles between the player and dancer's joint in real-time using an MSE metric. We threshold this metric to provide the various scores.

These steps are summarized in Figure 4.

### 3.1. Pose Estimation & Joint Detection

We use two deep pose estimation models for our application: OpenPose and Google's MoveNet model which are bottom-up methods. Our initial design was to run pose estimation on the user's webcam and the Just Dance YouTube video at the same time. This proved to be infeasible since MoveNet, our fastest model, was not able to accurately predict poses in Just Dance videos (see Figure 6). OpenPose, however, performed better as shown in Figure 5.

We settled on OpenPose because it's a high accuracy model that works well on videos. It uses the temporal relationship between video frames to estimate poses, adjusting joint estimates to account for motion blur. For this reason, we choose to use OpenPose to calculate the poses for the Just Dance videos, giving us a accurate estimate of the ground truth poses.

MoveNet, while less accurate has lower latency. It uses MobileNetV2 as a feature extractor and simultaneously predicts keypoint locations, and corrects for those estimations for a single person [16]. OpenPose, on the other hand, can detect multiple poses in an image or video frame. Since MoveNet was built for mobile applications we choose it to run on our website to estimate the pose of the person. As shown in Figure 7, the model is accurate when applied to a person, except when it comes to estimating face features.

### 3.2. Pose Matching and Scoring

We wanted to score the users on the same scale as Just Dance (Perfect, Super, Good etc.) Our system needed to compare the poses of the player and the Just Dance video

Figure 5. Openpose Results on a Just Dance video.

Figure 6. MoveNet Results on a Just Dance video. There are many errors in the estimation, leading us to not use this model for processing Just Dance videos.

dancer. Our models will output the x and y coordinates of each joint.

One way to compare poses is to directly compare the (x,y) coordinates generated for each frame of the video and the player. This would not give an accurate score because the pixel locations do not correspond to actual distances in 3D space. Also, since different people have different body sizes, with different limb length ratios, comparing the euclidean distance between joints is not feasible even if we are able to get actual distances. Moreover, the camera used

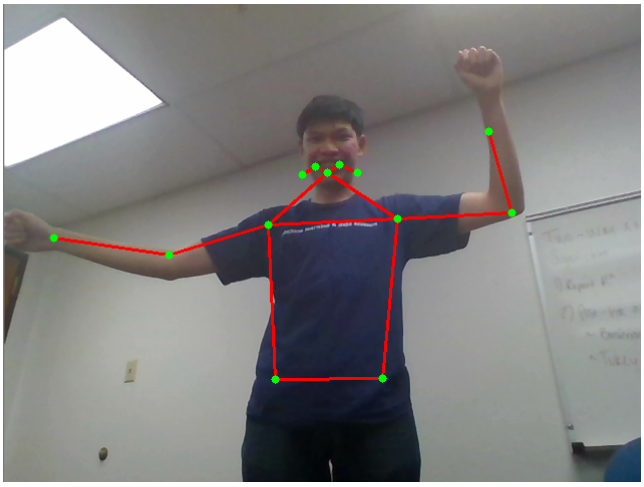

Figure 7. MoveNet Results on a real person. Although the face feature estimates are incorrect, the estimation for the arms and body are accurate.

to shoot the Just Dance video, and the user's cameras have different and unknown camera models that cannot be easily computed.

Our solution was to calculate the angle between the user's joints and the dancer's joints. We outline our process below and will justify each bullet point in the proceeding paragraphs.

- For the shoulders, and elbow joints, calculate the angles formed. In the below formula $P_1, P_2, P_3$ represents the three points needed to form an angle. Using the arc tangent formula we could find the angle between $P_3$ and $P_1$ with respect to the origin and similar for $P_2$ and $P_1$. Then subtract the two angles to find the final angle for the three points.

  $Angle = atan2(P_3.y - P_1.y, P_3.x - P_1.x) - atan2(P_2.y - P_1.y, P_2.x - P_1.x)$;

- Get the Mean Square Error of angles corresponding to the 4 joints listed above.

- If any of the joints are not present, add a penalty score of $0.25 * (180)^2$ to the MSE score.

- Run this MSE value through the equation $1 - tanh(alpha * MSE)$ , where alpha is a hyperparameter here that we used to control the difficulty of the game (i.e. lower alpha will give user a score generously and higher alpha will give user a score strictly), we set alpha = 0.0005. This value was determined through parameter tuning.

- Obtain a score by this value score using Table 1.

- Perform this operation every one second during the dance.

| Difference In Angle | Threshold | Score |
|---|---|---|
| $0° - 10°$ | 1 | Perfect |
| $10° - 20°$ | 0.9 | Super |
| $20° - 30°$ | 0.8 | Good |
| $30° - 40°$ | 0.6 | Nice |
| $40° - 50°$ | 0.3 | Ok |
| $50°+$ | 0.15 | X |

Table 1. Thresholds used for scoring. For example, if $1 - tanh(0.0005 * MSE)$ is 0.75, the score will be Good since $0.8 \geq 0.75 \leq 0.6$, and 0.8 corresponds to Good. Additionally, we can interpret this score as the user having an average angle difference of $20° - 30°$.

The first question is why we chose to use angles. We wanted a feature that could be easily captured from both videos and give signals to how close the player was to the dancer's moves. We chose joint angles because these are invariant to the size of the player and Just Dance dancer. We should note that the angle between joints changes as the player's orientation changes. So for scoring purposes, we assume that the player will face the webcam with their shoulders parallel to their computer screen.

Second, we only collected angles for the 4 joints from the upper body; right and left shoulders and elbows. This is due to the limitations of laptop cameras; the user would have to be far away from the laptop for their whole body to show in the frame. Therefore, we only scored the upper body since this is more likely to be in the frame. The reason why we don't use the angles between the neck and shoulder is that MoveNet doesn't have neck angles. We used the mean square error of these angles because we want the difference between the angles to have an impact on the score's numeric value. In our proposal we suggested using cosine similarity, with the vectors being formed from the joint angles. We abandoned this idea since the magnitude of the joint angles matter. For example, if our player's angle vector is $\langle 10°, 10°, 10°, 10° \rangle$ and the Just Dance dancer's vector is $\langle 60°, 60°, 60°, 60° \rangle$, the cosine similarity between two vectors would be 1 but the player shouldn't be getting a high score. In contrast, with mean square error the magnitude of joint angles matter. Additionally, MSE is more interpretable.We can interpret larger MSE values as corresponding to large angle differences between the player and Just Dance dancer.

Thirdly, we penalized the user if one of the 4 key points is not detected in the frame. This prevents the user from cheating; i.e. they could only show their right elbow on the screen. The value of the penalty for each missing joint is $0.25 * (180)^2$ and was determined empirically.

Fourthly, we also wanted to penalize large-angle differences by giving large-angle differences a lower score rather than a linear penalty. To achieve this, we use the hyper-

bolic tangent (tanh) function. It helped us to threshold our MSE values by mapping our range from $[0, \infty)$ (the range of MSE) to $[0, 1]$. We wanted a big angle difference to correspond to a high score; since MSE increases as the player does worse and worse, we do $1 - tanh(0.0005 * MSE)$. The constant 0.0005 was determined to make the thresholding easier. In Figure 8, we showed this scoring function vs. angle difference. We see that as the angle difference increases the value of the function decreases drastically. This makes it easy for us to penalize large-angle differences and determine thresholds seen in Table 1.

Finally, we perform this operation every second so that the player has to perform the dance well for the duration of the whole video. We choose 1 second as opposed to scoring each frame because we didn't want to penalize the player for being confused about dance moves. This also lowers the computational load on the laptop, as we have to run MoveNet fewer times.

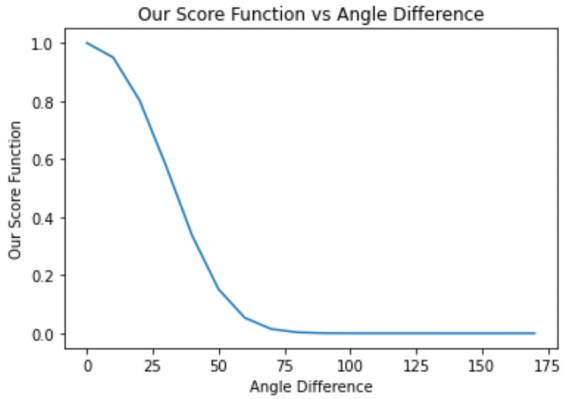


Figure 8. $1 - tanh(0.0005 * MSE)$ vs. Angle Difference. The sharp drop in values as angle difference increases makes it easier find thresholds for "Perfect", "Super", etc.

### 3.3. Overview of User Interface

Our user interface, as shown in Figure 9, displayed a Just Dance video and the user's score. In the website's backend server, our MoveNet read the laptop's web camera data, detect key points (joints) from the data, and score the user based on key points previously collected on the Just Dance video. These actions happened every one second. Since the website and MoveNet both require access to the laptop camera, we created a thread to run MoveNet and update the video frame from the camera. Then this frame is sent to the website for display. For more details, please refer to Figure 10. Just Dance videos we used were found and scraped from YouTube. The example used for the figures in this paper is here [17].

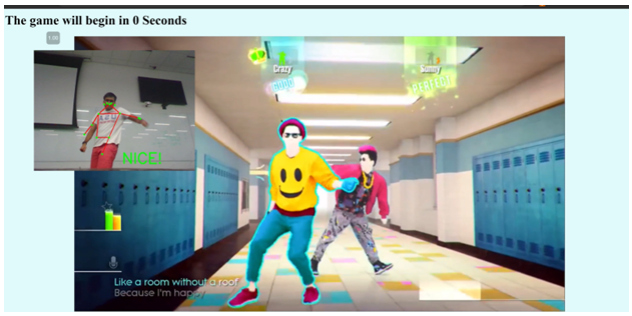


Figure 9. This is what our web application looks like. The Just Dance video is displayed in the middle of the screen and the player's webcam is shown in the top left along with their score.

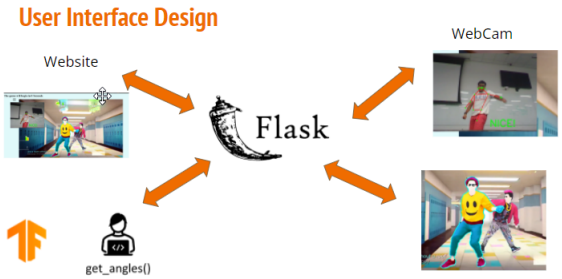


Figure 10. Our website manages three data sources: the webcam stream (top right), the Just Dance video (bottom right), and the MoveNet algorithm/Scoring function (bottom left). When the website is run, our Flask backend sends the Just Dance Video to the Website (top left) to render. It sends the webcam stream frame to the MoveNet function, gets the output score, displays in on the frame and sends it to the website.

## 4. Results

In this sections we include several screenshots on author Adi Ojha playing the Just Dance video game. These screenshots illustrate three major findings:

- Delay Issues: The player needs to anticipate dance moves in order to get a good score.
- Overall, the system works: The player receives good scores for good moves, and bad scores for nonsense moves.

We will now illustrate each of these points, one by one.

### 4.1. Delay Issues

The issue with scoring every one second is that there is a time delay between the player seeing a dance pose and moving their body into that pose. The player needs to time their moves to happen at the exact same moment as the Just Dance dancer in order to get a high score. Consider the frames shown in Figure 11. The player sees a dance gesture at frame t, and trys to mimic it after they see it. As a result, at frame t+1 the player has correctly done the move yet,


Frame t Frame t+1

Figure 11. The issue of delaying. At frame t, the player sees the dancer make a gesture. At t+1, they copy that gesture; however, if t+1 is used for scoring, the player will be compared to the t+1 gesture in the Just Dance video which may be different from the gesture at time t. The player has to anticipate the move at the scoring frame in order to be scored accurately.

since this is a scoring frame, we run pose estimation and compare the angles to gesture at t+1. Since this new gesture is different from the gesture at frame t, the player gets a score of "X". (Note, the "Good" in frame t, is from the previous scoring frame, which was not frame t.)

How can the player get a good score? They must anticipate the move, and instead of following the dancer on screen. In other words, the player must perform moves at the exact same time the dancer on screen does them. This makes the game considerably harder.

### 4.2. Bad Moves are scored as poor, Good Moves are scored as good.

Despite the issues with delays, the scoring method does accurately score the player. When the player does not follow along it gives them an "X" as shown in Figure 12. When the player follows along they get a good score as shown in Figure 9.

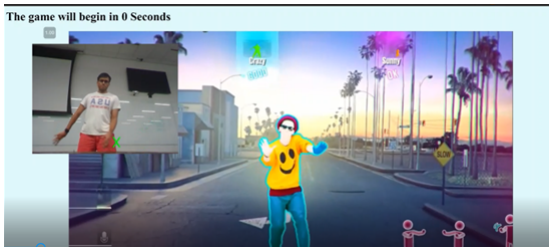


Figure 12. In this frame, the player is not following the dancer on screen, and as a result, he gets a score of X.

## 5. Conclusion and Future Work

With this project, we have established the structure of the scoring algorithm as well as the feasibility of playing Just Dance in real-time on a laptop.

There are some basic improvements that can be made to improve the user experience. For example, score the player less frequently can allow the player to anticipate what moves will be coming up reducing the delay error discussed in the results section.

Additionally, we recognize that dancing is not just about forming the right poses; it's also about how one moves between those poses. Using optical flow to measure if the player and Just Dance dancer are moving in the same way would allow us to capture the latter aspect. That is, our current algorithm only captures the spatial aspect of dancing (poses), and using optical flow would allow us to capture the temporal aspect of dancing (moves).

Thirdly, it would be beneficial if we could run OpenPose on the player. This would allow for more high fidelity scoring as the player and the video they would be following would be processed with the same model. Developers have made a version of the OpenPose that can run on iOS devices as shown in this GitHub repository [18]. This codebase along with the fact that the newest iPhones have 4 core GPUs, means that it is possible that we could run OpenPose on an iPhone version of this game[19]. In fact, from a business perspective, it is beneficial to make a mobile version of the game as this allows us to reach a broader audience.

In order to make the game more attractive, there will need to be a larger number of prepossessed dances. While the dance used for our project was processed from a JustDance video with OpenPose, allowing users to upload their own dances and process them with MoveNet, the same model used for the player, may allow for more accurate comparisons and scoring to be made between the user and the video.

With the popularity of TikTok, there is seemingly no shortage of those who are interested in dancing and watching people dance. The ability to tap into this massive market through devices that users already have access to would be a promising next step.

## References


[1] Elisha Odemakinde. Human pose estimation with deep learning - ultimate overview in 2022, 2022. Last accessed Feb 28, 2022.

[2] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, TEI '08, page 11–14, New York, NY, USA, 2008. Association for Computing Machinery.

[3] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014.

[4] Thong Duy Nguyen and Milan Kresovic. A survey of top-down approaches for human pose estimation, Feb 2022.

[5] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[6] Yuliang Xiu, Jiefeng Li, Haoyu Wang, Yinghong Fang, and Cewu Lu. Pose Flow: Efficient online pose tracking. In *BMVC*, 2018.

[7] Cmu-perceptual-computing-lab/openpose. `https://github.com/CMU-Perceptual-Computing-Lab/openpose`, 2022.

[8] ahmedsabie. tensorflow/tfjs-models. `https://github.com/tensorflow/tfjs-models/tree/master/pose-detection`, 2022.

[9] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[10] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. Rmpe: Regional multi-person pose estimation. In *ICCV*, 2017.

[11] Ronny Votel and Na Li. Next-generation pose detection with movenet and tensorflow.js, May 2021.

[12] Google. 3d pose detection with mediapipe blazepose ghum and tensorflow.js, 2021.

[13] Siddharth S. Rautaray and Anupam Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. In *Artificial Intelligence Review*, pages 1–54, 2015.

[14] Krishna Raj. Human pose comparison and action scoring using deep learning, opencv, and python, 2020.

[15] Google. Pose classification options, 2021. https://developers.google.com/ml-kit/vision/pose-detection/classifying-poseskotlin.

[16] TensorFlow Community. Next-generation pose detection with movenet and tensorflow.js, 2021.

[17] XCageGame. Just dance 2015 - happy pharrell williams gameplay - 5 stars rating [ hd ], 2015. https://www.youtube.com/watch?v=G74_o_43_RQ.

[18] infocom tpo. Swift open pose, 2021. https://github.com/infocom-tpo/SwiftOpenPose.

[19] Juli Clover. iphone 13 pro offers significantly improved gpu performance compared to iphone 12 pro.