Fitness-Run: Real-Time Squat Detector Game for Fitness Using Computer Vision Concepts

COMP 425 - Computer Vision

Ismail Shafique
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada

Adam Atamnia
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada

***Abstract*—We embarked on a mission to build an endless-runner type game, that would be controlled using physical exercises. Thus, promoting people to become more active in their lives despite their busy schedules. The endless-runner game uses key concepts in Computer Vision, specifically Landmark Detecting and Image Classification. After much consideration towards different types of exercises that can be used as the main control system (when the character jumps over obstacles), the physical movement that was best suited for our game as well as well-coordinated with Computer Vision concepts was indeed the Standard Squat. Our first approach to coding a Squat-Counter, was to first create our own dataset and identifying clear landmarks that are needed to identify squats. We did this by approximately collecting 200 images of different full-lower body poses online (these images were taken from stores like H&M, Forever21, etc.) These images varied within color, pose, noise background, variations of lower garment wear and more variables which we trained through a Convolutional Neural Network model (CNN). These images went through the process of labeling them (manually labeling the right hip, right knee, right ankle, left hip, left knee and left ankle). We concluded that the training a model manually using this technique indeed enhanced deeply our Computer Vision understanding (since we went through each and every step of the process), but the process took an extremely lengthy time. Not to mention that training the model was slightly unsuccessful due to a limited and not diversified enough dataset (we needed more like 2000 images and to also manually label all the landmarks manually as well which was deemed to be too long for a project like this). The results of our model ranged from 50-70% accuracy which was deemed to be too low for use. Moving on, we used an alternate technique which was to import a ready-made pose detection model (fully trained) and this was going to detect the same points as our customed trained model. We were than going to extract the angles between those landmarks to deem whether a squat was successful or not (and then translate that into our game). This method was successful upon completing the code and running approximately 15,000 simulations! We also tried using a completely different technique which was an Image Classification Model (to see if it was better than our custom trained model or MediaPipe's trained model). We manually found and labeled approximately 500 different images and trained them as well.[1] This was deemed successful (more of the results for this specific model will be discussed in the presentation and the final report (due to word-count limit and for the conference paper not to exceed the 6-page limit). Ultimately after comparing the models, we decided to use the MediaPipe model as it yielded the best results. This was than implemented within our game application successfully, after which, we designed the graphics, visual layout, character motions/states within the game.**

I. Reasons for building the Endless-runner

As is well-known across many countries around the world, the rise of obesity has significantly increased over the past couple of decades. More specifically, recent data suggests that 26.6% of Canadians struggle with being overweight, and this data is consistent within both rural and urban communities alike (Public Health Agency of Canada, 2020).

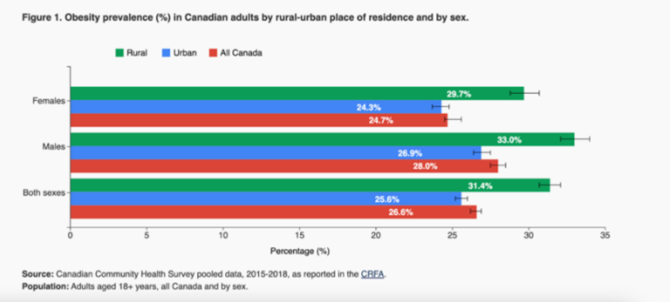


Figure 1. Obesity prevalence (%) in Canadian adults by rural-urban place of residence and by sex.

Source: Canadian Community Health Survey pooled data, 2015-2018, as reported in the CRFA.
Population: Adults aged 18+ years, all Canada and by sex.

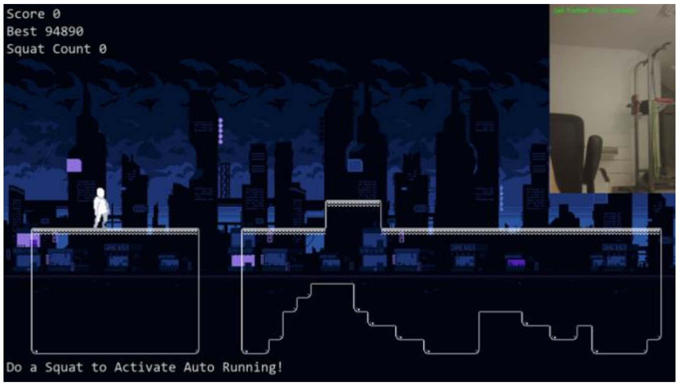
Another study presents how over the years, there has also been a large percentage of adults being classified as physically inactive. The World Health Organization conducted a study where they found approximately an estimated 1.8 billion adults are physically inactive (World Health Organization, 4)! These statistics are extremely concerning and thus, the goal of this project is to build a computer-game application (using various

[1] Note this part (the Image Classification Model wasn't entirely complete upon writing this IEEE conference report, thus out of honesty we pointed this out. Although it will be included in the presentation and the final report.

concepts in Computer Vision). The main purpose of this computer game would be for the user to lose as many calories as possible (encouraging exercise while at the same time being used for weight-loss to combat obesity). The theme will be based on an endless-runner type game, while the controls will be performing exercise movements (such as performing a 150 degree box-squat which constitutes to a "jump" when going over obstacles). The goal of building this into an endless-runner application is to encourage users to beat personal records (in terms of their fitness objectives) and have a great time playing a highly interactive game! Many people struggle with exercising (as it's physically demanding on the body) while also finding the time where they can relax themselves while playing video games. Thus, our main selling point towards this application is being able to burn calories while at the same time keeping it in a game-type theme.

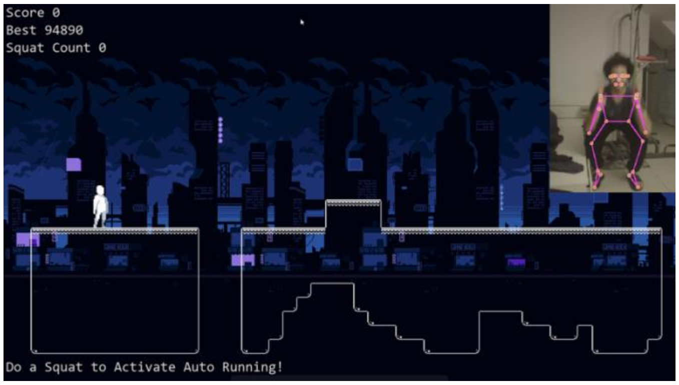
II. Brief Background on How the Game Works

The game that utilizes our squat detector is an 2D auto runner meaning the character runs automatically and the player only has to control the jump to navigate platforms. The game is endless meaning it only ends when the player is eliminated by falling off the screen. We decided to do an auto runner to keep the controls simple allowing a simple squat to be to be the only movement needed to control the game. The game keeps track of the number of squats a player does and a score that describes how far the player can run without falling. The game also saves the highest score to encourage players to improve. In the future we can add a progress bar at the top that shows how close the player is to his previous best score in a way that is more easy to see. We could also add a streak system (that shows how many days in a row the player has worked out) which can encourages players to come back and work out day after day. To build the game we used the "pygame python module" and some art assets (some paid and some free) that we found online. Pygame is useful because it allows us to load sprites (images), draw them on the screen, detect collisions between the sprites, and get user input among other things. At first, when we linked the squat detector and the pygame-code, the performance dropped significantly since they were both running in the same process. To solve this problem, we decided to use separate processes for the game and the squat detector which drastically improved the game speed.



Score 0
Best 94890
Squat Count 0

Do a Squat to Activate Auto Running!



Score 0
Best 94890
Squat Count 0

Do a Squat to Activate Auto Running!

III. Deciding Which Exercise is Best for the Game

Knowing before-hand that we wanted to build an endless-runner game that incorporated some sort of physical exercise as the main control system (mainly the jump mechanism over obstacles), this brought up a valuable question: "which exercise to use?" Upon asking this question we decided to narrow down our exercises to three different possibilities and state the pros/cons. Thus, being able to find the right physical movement for the endless-runner game.

Option 1 - Standard Push-Up: In terms of the pros upon first glance, a standard Push-Up was ideal to be implemented in our game since it was a very standard and well-known exercise. The greater reason which inclined us to possibly use Push-Ups was due to its ease of being implemented with Computer Vision. More specifically, we imagined using a model (or training one) which could detect certain landmarks within the human body. Those landmarks would specifically be the wrist, elbow and shoulder respectively. Using that landmark detection in those certain areas, we can then test and manipulate a variety of different experiments. The main one being, finding the angle in between the ankle, wrist and shoulder which we can then use to determine if someone has indeed performed a Push-Up or not (if the angle between these limbs were 90 degrees or under, the program can record that as a successful Push-Up etc.) We can also use these landmarks to detect different variations of Push-Ups (such as diamond Push-Ups, wide-arm Push-Ups etc.)

which would increase versatility to our game. Another advantage to using this exercise was the fact that its movement is quite slow, thus being easily detectable for various Computer Vision Landmark detection systems. Although there were many positive aspects when it came to the Push-Up, after further examination this exercise had many problems that were initially overlooked. Amongst them was the fact that a Push-Up was more of a strength exercise rather than being cardio-oriented (focus on building muscle rather than fitness-oriented which focuses on burning calories). This is further supported when studying how many calories are burnt, which is approximately 7 calories per minute (Push Ups Calories Burned Calculator, 1) (these numbers are approximate considering the various variables that come into play, this is assuming a person who is 150 pounds). This number is relatively low when it comes to a fat-burning exercise, and thus this was definitely a con when it came to our specific game. It's also well-known that the average person (considering that a vast majority of the population is inactive in the first place) cannot perform more than 10 Push-Ups consecutively! Thus, the Fitness-Run game would only last 15-20 seconds per run which is much too short for duration. Since the Push-Up is an exercise which is performed in a plank position on the ground, as well as your face and entire frontal body facing the floor, this posed many problems to our game design. Most notably, not being able to see the screen when playing the game (since your eyes are constantly staring downwards). This presented a large problem as the user will not be able to even see the obstacles in the first place, and thus, this exercise was quickly discarded from being used as the jump mechanism in our game.

Option 2 - Jumping-Jack: This was another strong consideration for us in terms of a possible exercise for jumping over obstacles in the game. That being said, this exercise too had many advantages which inclined us to possibly use this within our application. The first strength was the fact that Jumping-Jacks are a very beneficial exercise when it comes to cardio and fitness. The average person (even if they're very inactive) can perform much more Jumping-Jacks (comfortably) than Push-Ups since it focuses more on body balance rather than strength. Jumping-Jacks also burn around 9 calories per minute which is an increase when comparing to how many calories are burnt from Push-Ups in a minute (Calories Burned from Jumping jacks vigorous, 1) (these numbers are approximate considering the various variables that come into play, this is assuming a person who is 150 pounds). Another up-side to Jumping-Jacks was the fact that they are an exercise that can be performed while staring at a computer screen (unlike push-ups where you are constantly staring at the floor), thus, being aware of the actual gameplay. Despite these positive aspects we ultimately decided to discard Jumping-Jacks at the end due to a number of reasons which were not compatible with our Computer Vision design and the theme of our game. The largest reason for moving on from this exercise was the fact that performing a Jumping-Jack is a relatively quick-moving action. Not only this, but you have multiple key joints within the body that are moving all at one time in this fast motion. This was a realization that came to us after trying to code the Landmark Detection Model with Computer Vision. We found that fast movements often disrupted the pose detection landmarks (since the model constantly had to recalculate because of the movements being increasingly fast. Another obstacle that we faced (which was another reason for us to abandon this exercise from the game) was due to scalability issues. A Jumping-Jack needs the entire body to be visible from the camera-lens, thus involving more key points to be detected by the Computer Vision Model. This was an issue that was very difficult to avoid, and it hindered the performance of our application if we were to use it (in combination to the fact that these points will be moving extremely fast considering how Jumping-Jacks are performed quickly).

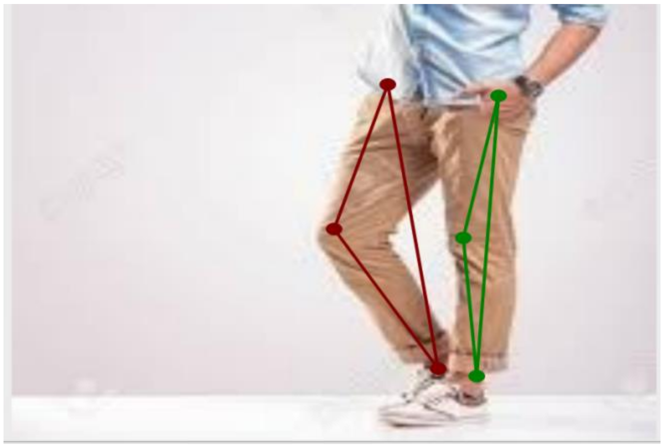
Option 3 (picked exercise) Bodyweight Squat: This was ultimately the exercise that we ended up choosing for our game (specifically when the character jumps over obstacles). The Bodyweight Squat was ideal for our needs in the game, and it functioned well with Landmark Detection. More specifically, it was easy for any model that we used (whether a pre-trained model or a model that we trained ourselves using CNN) to detect certain landmarks within the squat since its detection points are very apparent within the human body. For example, identifying body parts like the ankle and knee were very simple since they usually protrude even when wearing pants, thus making it simple to detect these key features using a Computer Vision Model. Another point being that a squat is a relatively slow exercise to perform and thus, most models will not have to constantly calibrate to re-find certain points (as for example the ankle is always in a constant position, while the knees and hip are moving in a slow motion). This is vital for accuracy of the model since this is something we struggled with when trying to test out exercises that involved faster ranges of motion which ultimately caused Landmark Detection to be difficult to identify smoothly. Continuing upon the last point, we realized that it would be very easy to detect a correctly performed squat since we can simply calculate the angle (between the hip, knee and ankle) to identify a standard squat (we can set the values to whatever we want, whether it be 90 degrees or under 150 degrees etc.). Finally, bodyweight squats were excellent when it came to the number of calories burnt per minute, it's estimated that around 10 calories can be burnt per minute (if done with high intensity and assuming the person weighs 150 pounds) (Leventon, 1). After seeing the benefits this exercise brings both in terms of the Computer Vision aspect and game-performance, we decided to move forward with this action (and for it to be our main control system for the jumping of the character).

Calories Burned per Minute (150-Pound Male)

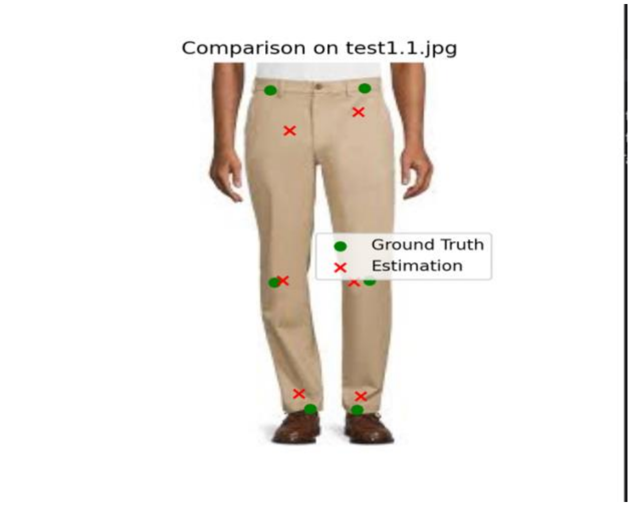
Exercise	Estimated Calories/Minute
Push-ups	7 calories
Squats	**10 calories**
Jumping Jacks	9 calories

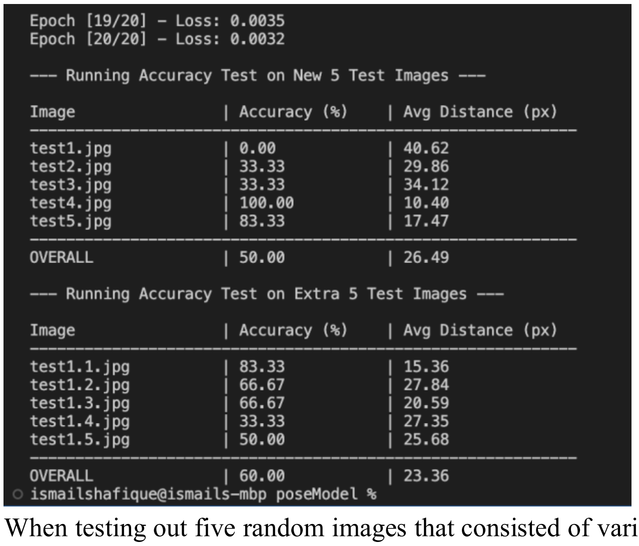
IV. Training a Model Ourselves with CNN

In terms of building the squat-counter for our Computer Vision Project, we examined a variety of different methods that we can use to accomplish our goal. Amongst the earliest of them, was to train our own Model to detect squats. The first step we took to accomplish this aspiring task was to build a custom dataset. While doing much research online for ready-made datasets, we were unable to find anything that suited our needs. Thus, we decided to collect as many images as possible, specifically lower-body photos that displayed the hips, ankles and knees of a person. We started collecting these images from various websites that ranged from fitness apps, clothing stores, blogs etc.). After collecting approximately 200 images manually, we than needed to take these images and label the key landmarks that we wanted to identify. More specifically the right hip, right knee, right ankle, left hip, left knee and left ankle. Labelling these points correctly was extremely important in training our model and thus we downloaded a Python extension called LabelMe which labels keypoints (the landmarks mentioned above) that want to be identified in .jpg images. It then converts these .jpg images into JSON files (which will be later used for NumPy data array set, this will be discussed further later). One of the advantages to this process was the fact that it really made us understand the process it takes to build a dataset from scratch, and to then manually label all six landmark locations. It definitely took an extremely long process to these steps, which may have been a possible disadvantage to this idea of building an entirely new dataset.



The next step was to take these now edited images, and to convert them to the appropriate JSON files. The reason behind this was simply because a machine is able to read a JSON file (rather than a .jpg image). This step is also vital since you would need to extract the data from the JSON file (in terms of the coordinates of the six Landmark Detection sites) and be able to store them in a NumPy data set array (for the CNN to later train with these data values). For more clarity, the JSON files would contain the x and y coordinates of each of the six landmarks (meaning there would be twelve data points per image). After completing this, we than decided to enter these coordinates into a NumPy data array set. This was extremely important since this is the data that we would use to train the CNN model with. Upon discovering this, we decided to then train our CNN model, and ran to exactly 20 epochs (we stopped at 20 epochs since between epochs 15-19, the loss was extremely minimal, and the model wasn't improving drastically after it passed through these cycles of the CNN). The way the CNN and epoch system works in simple terms is that it takes your dataset and splits it into batches (for example batches of 15 images each) and it uses these grouping of images to make estimates (in terms of our case pose detection). One epoch is essentially one pass through the entire dataset (so if we have 150 images, ten batches of checking would equal to one epoch, considering it takes 15 photos per group). As it goes through epoch after epoch, the loss should be less, meaning that the model is improving better and better in terms of finding the landmarks. *Note that the loss is found using squared error means, thus loss 0.0068 is equivalent to the square-root of 0.0068 (meaning about 8.25% away from the ground value). If you wanted to know how many pixels away this was, you would than multiply 0.0825 by 128 pixels (considering if the image was 128x128px). Once this process was over, we than moved on to testing our trained model. We decided to import five test case images to see how well the model would be able to identify the keypoints. These five test images were pre-labeled (with the same python extension used previously "LabelMe") and then the model would have to estimate where those keypoints were located. Upon doing this, we would then measure how far away the guess was from the right answer (we made a threshold of 25 pixels, meaning that the guessed landmark would be deemed correct within this range). After doing all these steps we noticed that are results were very staggered when it came to training our own custom manual using CNN. As is shown in the figure below, the model was great at estimating the landmarks when it came to a normal lower-body standing pose (note the X's are the estimates from the trained model while the green dots are the actual manually entered points). When testing five random images using this particular pose, we received an accuracy of 60.0% (using a threshold of 25 pixels), with a mean distance to the keypoints be 23.36 pixels away from the ground values. These results were not the best since we were hoping for a number closer to 90-95% (these types of numbers can be used in our actual game application).

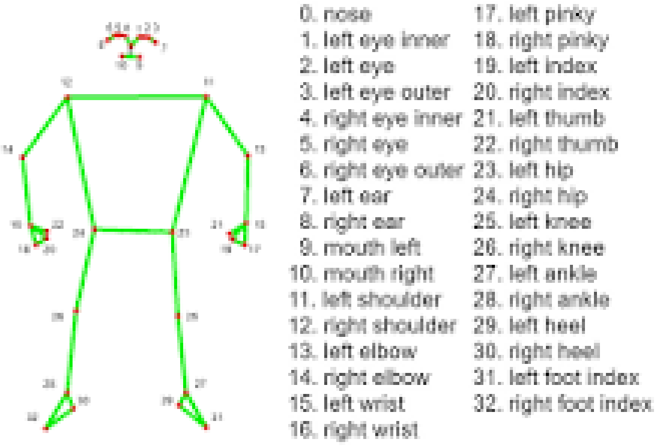



```
Epoch [19/20] - Loss: 0.0035
Epoch [20/20] - Loss: 0.0032

--- Running Accuracy Test on New 5 Test Images ---

Image                  | Accuracy (%)     | Avg Distance (px)
---------------------------------------------------------------
test1.jpg              | 0.00             | 40.62
test2.jpg              | 33.33            | 29.86
test3.jpg              | 33.33            | 34.12
test4.jpg              | 100.00           | 10.40
test5.jpg              | 83.33            | 17.47
---------------------------------------------------------------
OVERALL                | 50.00            | 26.49

--- Running Accuracy Test on Extra 5 Test Images ---

Image                  | Accuracy (%)     | Avg Distance (px)
---------------------------------------------------------------
test1.1.jpg            | 83.33            | 15.36
test1.2.jpg            | 66.67            | 27.84
test1.3.jpg            | 66.67            | 20.59
test1.4.jpg            | 33.33            | 27.35
test1.5.jpg            | 50.00            | 25.68
---------------------------------------------------------------
OVERALL                | 60.00            | 23.36
○ ismailshafique@ismails-mbp poseModel %
```

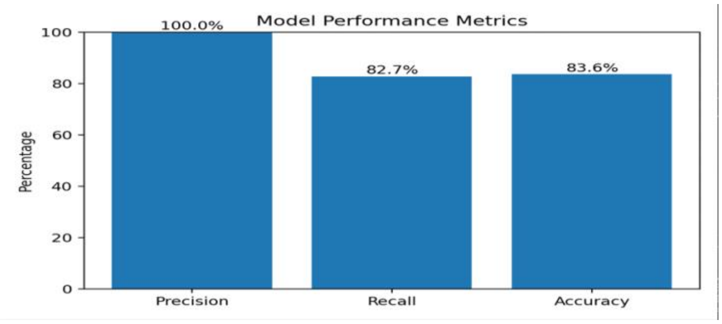
When testing out five random images that consisted of various strange forms of poses, this is when our model did extremely poorly (as shown in the image above). The average when it came to accuracy was 50% and the average distance away each landmark was to its associated ground value was 26.49 pixels! These values for our model were not that impressive and we suspected many reasons as to why the model failed in this section. Firstly, of the approximately 200 image data set that we manually selected from a variety of different websites online, we found that many of the images that we selected were of the standing still lower-body pose (the one that received a better accuracy of 60.0%). As compared to the other test images which consisted of varied poses. We concluded that our model needed more of a variety of images under this domain for it to be trained to observe such images as well (and thus this was amongst the reasons we received a 50% accuracy). Another thing that we noticed was the fact that our data set consisted mostly of people wearing slim fit attire (as this is the trend amongst the photos we obtained from the store websites). Thus, when we gave a test image to the model which consisted of a person wearing very thick and wide pants, the model wasn't able to estimate the keypoints of the knees particularly since they were more covered within the image. Another Computer Vision concept comes to mind was the introduction of occlusion which blocks certain body parts due to a person standing in a position that blocks a landmark. This also affected our trained model deeply and thus we saw many incorrect findings. Again, the way to avoid this from happening was to train more photos that contained some of the keypoints to be under occlusion. After considering all the steps we took to train this model manually, we learnt many Computer Vision concepts on the way. This process was beneficial in the sense that we understood how each step and phase is done for a model to correctly estimate landmarks within the human body. On the other hand, we realized how this particular design may be something which is too difficult to undertake under an undergraduate university setting, since we would need to manually find and label around 2000-3000 images (since the diversification like mentioned above was something very important to training CNN's) rather than approximately 200 which is what we did which undertook a large portion of our time. Thus, we decided to move on from this model and pursue one's whose accuracy was higher.

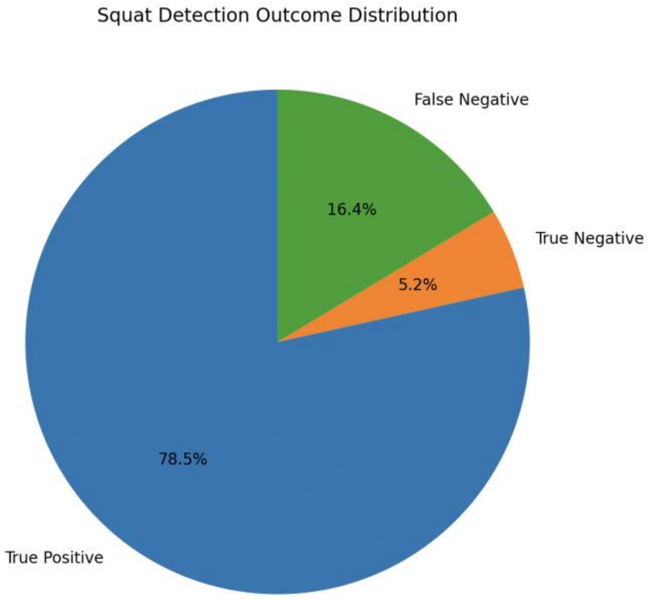
V. Using a Pre-Trained Model: MediaPipe

Knowing that the main objective that we were looking for when it came to building our game (which was to have a model that can detect squats in an accurate manner), we decided to pick a pose detection model that was made by Google. The reason behind picking this model was for numerous reasons. Amongst the first of them was the fact that it was a Landmark Detection Model that focused on locating exactly 33 different keypoints within the human body. Amongst them, was our desired keypoints that we were targeting in detecting squats (the right hip, right knee, right ankle, left hip, left knee and left ankle). The idea behind using MediaPipe was to first extract these keypoints (the ones mentioned in the sentence before), and to than find the angle between these three points (on each side so six).



An advantage to this model was being able to be more precise in terms of squat detection (this will be discussed further during the presentation regarding the third model we considered which was Image Classification within Computer Vision). What is meant by precision is the fact that a squat can be deemed correct depending on the angle of the knee in proportion to hip and ankle. For example, within the code we decided at one point to make our angle to be 150 degrees or lower for a correctly performed squat (to than later register a jump in the game). This number can be altered later on after tests which makes this model advantageous due to its versatility of simply changing the angle for it to be considered a correctly performed squat. This cannot be done with the third model (which will be discussed later) since with Image Classification, we are deeming a list of squats to be accurate without taking into account how many degrees the person has bent his knees and these well-known metrics which are used. Simply, because Image Classification looks at an image as a whole rather than estimating Key Landmark Detection of points and being able to extract those values. Thus, MediaPipe seemed appeasing in that sense as well, since altering the degrees for a valid squat can change the intensity of the game (lower squats of 90 degrees and below are harder to perform than higher squats of 150 degrees). Another benefit to using the MediaPipe model was the fact that it can track these landmarks even if the screen is live (doesn't need to be a still screen). This was important for our project since the gameplay was going to be taking place with a live camera feed, and thus, live landmark detection needed to be present. Something which was explained previously in Section III (Deciding Which Exercise was Best for The Game), was the fact that even with MediaPipe supporting live Landmark Detection with a running video-feed, its performance was deemed highly inaccurate from the start when performing a physical movement in a fast speed. Exercises like Jumping Jacks were discarded for this reason since the model couldn't detect fast enough the landmark points as they changed constantly (and had to look at 12-15 different key landmarks within the human body instead of 6). Its calibration wasn't up to speed and thus, MediaPipe was only going to be the model used for the squat exercise in our endless-runner game. Another advantage to using MediaPipe as a model was the fact that it recorded 18 points (x, y and z) in comparison to the model that we trained (which only recorded coordinates in a 2-D field). The x and y angles were vital for reasons already stated previously, but the z direction was extremely important as well. Knowing the depth of the image was important since some users when playing the game squat facing to the side of the camera (which in this case you would only need the x and y coordinates), while players that faced the camera when squatting need to use the z angle as well. Since you're facing the camera when playing the game, many times the hips of a person might lean far back in comparison to his back, thus hiding the hips. This makes an occlusion (this Computer Vision term is previously explained in the report) phase when trying to detect the landmarks within the human body. Thus, the z direction comes into play here, since it would estimate the depth and hip placement, despite the knees blocking them (if a person chooses to play the game facing straight on to the camera instead of the side). This was another subtle difference between the model that we trained versus using an already trained model form Google. Another thing that we took into consideration when it came to choosing MediaPipe was the fact that it used a dataset that was extremely diversified as well as containing approximately 30,000 different images (unlike our small dataset). In terms of coding the MediaPipe model into our code, this was fairly simple and only involved minimal lines of code for extracting the landmark values (since importing MediaPipe gave us access to quick functions for extraction), and then to manipulate them in order to find the corresponding angles were simple as well and involved basic arithmetic. In terms of accuracy to the model we used (when comparing it to various data simulations) we found that the MediaPipe model that we used was successful.





As is shown in the graphs, after 15,000 simulations of data, we found the precision to be 100%, the recall to be 82.7% and the accuracy to be 83.6% (this will be delved into deeper later in this section). To test out the model and its metrics, the 15,000 simulations were divided within the following categories. Firstly, we have set 14,250 squats to be valid technique (being 95% of the 15,000 simulations). Then we have 5% of the squats which are incorrect (being 750 squats, and we use the variable bad_squat_rate = 0.05 to do this to our simulation data). Thus, we want to know how many are correctly identified in the appropriate category. We also use this part of the data:

```
if random.random() < dropout_rate:
    continue
```

With dropout_rate = 0.025, about 2.5% of frames are skipped, just like when a system fails to detect joints in some frames (we added this since this does happen when a squat is performed to fast, so we added it to our data simulation set). We also added this if there is a lag in the camera and it doesn't pick up the speed of the person performing the squat, so the frame rate gets compromised by 2.5%. This was also added within our simulation as camera failures or deficiencies due happen (not everyone has an amazing camera frame rate etc.). Also knee angles will not be perfect (you have part of your clothes covering the angle, or the landmark detection could be off by a couple of pixels etc. thus, we added some random added value to the angle (to see if the model will still be able to tell if the angle may appear to be a few micro degrees off). We did this by added the code segment on the bottom:

```
45 + random.uniform(-5, 5) – for a deep squat
110 + random.uniform(-10, 10) – for a higher elevation squat
```

When it came to precision as shown in the graph, of the squats that the model said were good, all of them were indeed good, thus giving us 100% accuracy. This came to know surprise since the model has been trained by over 30,000 images and is made by Google. When it comes to recall, of all the actual good squats that there were in the simulation set, the model was only able to identify 82.7% which is still a considerably high number given the very harsh variables that we put into the simulation (for example the camera lagging by reducing the frame rate at points, etc.). Lastly in terms of accuracy, of all good and bad squats that were correctly identified, the number came out to be 83.6% which was also better than expected considering the variables that were put into place in our dataset. We can say though through observation (from testing the Squat-Counter manually as well, that it is highly correct in detecting squats to the point that it is extremely rare for it to miss one). When testing out the game for approximately an hour, we have gone through one or two squats that we seemed doubtful as to whether the model should've recorded it or not. Thus, the real-life test (outside of the simulation) also passes the test (this part can be tested out in the presentation if the TA/Professor wants to).

Ground Truth	**Model Estimation**	**Outcome**	**Explanation**
Good Squat	Detected	True Positive (TP)	Model correctly detected a good squat
Good Squat	Not Detected	False Negative (FN)	Model missed a good squat
Bad Squat	Detected	False Positive (FP)	Model incorrectly detected a bad squat as good
Bad Squat	Not Detected	True Negative (TN)	Model correctly ignored a bad squat

VI. Final Step: Programming the Endless-Runner

To create the game, we looked up how to create a game in python and followed some tutorials on YouTube. Here is a general overview of what we did. Firstly, we created the main game loop and made it so the program terminates when the user presses the ESC key. The role of the game loop is to updates the position of the game objects (like the players and the platforms) based on certain logic and user input then draw the game objects on the screen. Then we created the player object, and we bought some art assets to represent the player visually and to add animations to the player. We added movement to the player by moving the position of the player left or right (based on user input) by a certain number of pixels (the speed) every iteration of the game loop. When the user initiates a jump, we move up the vertical position of the player by a certain number of pixels (the jump velocity) every iteration of the game loop minus the gravity to make sure that after a while the player falls back down. We created the platform images on a tile editor online end imported them into our game. Pygame offers some convenient functions to detect if there is a collision between sprites (images) which allowed us to add collision detection between the player and the platforms so that he doesn't fall thought them. We added some background art that we also bought and made the different piece of the background move slightly when the player moves to give a parallax effect. We added a score counter that represents how far the player can go without being eliminated and we save the highest score in a text file. Finally, we made the player move to the right automatically and made the squat detector activate the jump. We had to run the squat detector in a separate process to avoid lag.

References

[1] *"Obesity in Rural and Urban Canada."* Public Health Agency of Canada, 4 Nov. 2020, health-infobase.canada.ca/datalab/canadian-risk-factor-atlas-obesity-blog.html. Accessed 29 Mar. 2025. (Ignore Images and Ads)

[2] World Health Organization. "Physical Activity." *World Health Organization*, 26 June 2024, www.who.int/news-room/fact-sheets/detail/physical-activity. Accessed 30 Mar. 2025. (Ignore Images and Ads)

[3] Leventon, Scott. "Calories Burned Squats Calculator." *Fitness Volt*, updated by Tom Miller, CSCS, 5 Sept. 2023, https://fitnessvolt.com/squats-calories-burned/. Accessed 30 Mar. 2025 (Ignore Images and Ads)

[4] "Push Ups Calories Burned Calculator." *BizCalcs.com*, www.bizcalcs.com/push-ups-calories-burned/. Accessed 29 Mar. 2025 (Ignore Images and Ads)

[5] "Calories Burned From Jumping jacks, vigorous." *MyFitnessPal*, www.myfitnesspal.com/exercise/calories-burned/jumping-jacks-vigorous-421094. Accessed 29 Mar. 2025 (Ignore Images and Ads)

[6] Maltsev, Anton. "How to Improve Mediapipe Skeletons Recognition." *Medium*, 9 May 2022, https://medium.com/@zlodeibaal/how-to-improve-mediapipe-skeletons-recognition-7c3009774dd4 Accessed 31 Mar. 2025 (Ignore Images and Ads)