Bicycle Fitting
GIP Project, Spring 2021

Rotem Elias, Emmanuel Ferdman

Supervisors: Haitham Fadila, Alon Zvirin, Yaron Honen

Computer Science Department, Technion - Israel Institute of Technology, Haifa

Contents

1 Abstract

Bicycle fitting is a process of adjusting a bicycle for a cyclist. A good bicycle fit is the key to improve cycling. Every cyclist should fit his bicycle to his measurements in order to improve performance, prevent long term injuries and optimize the cycling experience. The bicycle fit creates the connection between the rider's current physical state and what it is they desire to achieve.

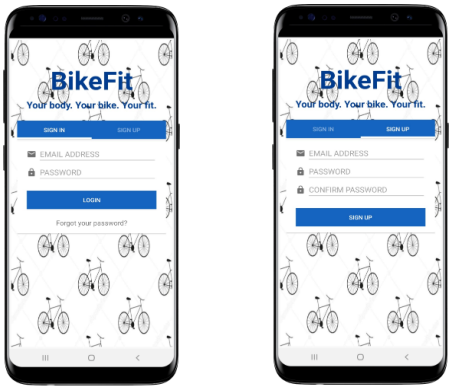
Adjusting the bicycle to fit the individual needs of the cyclist is a hard task. A human being is a complex organism that depends on a wide variety of parameters. One individual could be tall and slim while the other one could be short and muscular. Each such state could affect the right configuration of the bicycle and should be taken into account. Furthermore, the goal of the individual plays an important role in the fit process. Some are interested in finding a position for increasing aerodynamics for their next triathlon, while others are interested in improving their cycling experience by reducing the pain.

Since self bicycle fitting could be inaccurate and lead to injuries, the cyclist could contact an expert for professional help. The level of skill and expertise that the cyclist will receive, depends on the qualifications from the specific shop. The problem with requiring such service is that the process isn't cheap. A single session could cost up to 300$ and usually it takes no less than two hours [1]. While that may seem like not a lot, the cyclist should keep in mind that the body of an athlete changes frequently and that he will have to revisit the service in the near future.

The goal of the project was to create an application that allows to upload a short video of the cyclist riding his bicycle and rank the bicycle configurations. Such ranking should give a good initial indication of the current fitting state. The application is using digital image processing algorithms to detect the rider's body, find the coordinates and calculate the angles. In addition to the height, this information will be used by the software to identify and rate the rider's seating configurations. Since the task of configuration rating, is not an easy task that can be done with predefined set of parameters, we will turn to artificial intelligence for help. We will collect data of riders riding their bicycles and train a prediction model that can rank the user's bicycle configurations. One of the most important factors in a bicycle fit that provides a good cycling experience and prevents injuries are the configurations of the saddle and handlebar. So we will include the information of those two factors as part of the data. Once we have a prediction model, we can use it at the backend of our application.

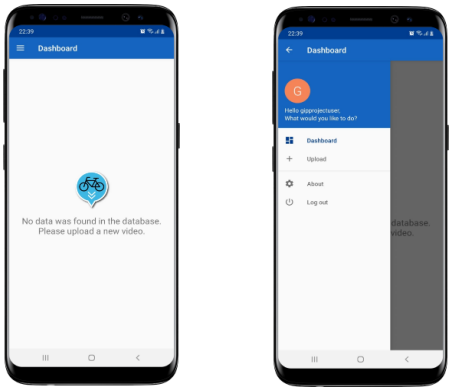
2 Application overview

The opening layout of the application lets the user to create a new account by signing up or to enter into an already existing account by signing in. In the sign up screen, user will have to provide an email address and a password. Such authentication is needed to distinguish between the data of the different users, which will be uploaded into the application by them. Once the form is filled, the application will create a new account using the Firebase Authentication service [2] and then it will automatically login the user into the system. In case the user forgot his password, the application provides an option to restore it using his email address.



The opening layout of the application, allows users to sign in into an existing account or create a new one

Once logged in, the application will redirect the user to the "Dashboard" layout. In the dashboard, user can view all of his uploads and their detection results. For new users, this layout will be empty, until they will upload a new video in the "Upload" layout. The application provides a navigation UI component to navigate between layouts. This feature allows to move into the "Upload" layout.

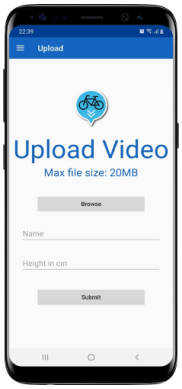


Once logged-in, users can navigate between different layouts like the "Dashboard" layout

Inside the "Upload" layout, user can upload a new short video of himself riding a bike. In order to do so, user will have to click on the "Browse" button, allow the application to access the local storage (by allowing the required permissions) and choosing the video from the opened System Explorer. The maximum size of the video file is 20MB. In case the user tries to upload a file with

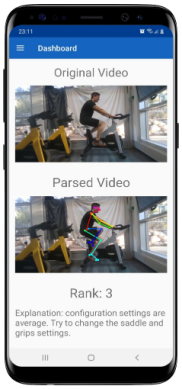
bigger size, the application will not permit it and will display an error message using the Toast widget [3]. Also, the application supports only videos in format AVI or MP4. This means that if the user will try to upload a file in different format, the application will not permit it and will display an error message using the Toast widget.

User will also have to provide a unique name which describes the video in order to distinguish between different uploads. Furthermore, user will have to provide the height of the person filmed in the video, since it's part of the dataset features. Once user clicks on the "Submit" button, the application will upload and store the information to the Drive. At the end of the process, a Toast widget will display the status of the upload (success or failure).



In the "Upload" layout, user can upload a video of himself riding a bike

The backend will be triggered by a new video upload. Once the backend finishes performing the image processing operations on the video and running the model on the dataset, it will upload the results into the Drive. Then, user can navigate back to the dashboard and check the results. After pulling the results from the Drive, the application will display the original video, the parsed video containing the body detection results and the rank of the bicycle configurations that were found in the video. The configurations are ranked between $1 - 5$ and for each such rank, the application will add a message that describes the situation.



In the "Dashboard" layout, user can view the results of his uploaded video

In case the rank is not the highest score (which is 5), user can modify his bicycle configurations and upload a new video to the application.

3 Technologies and equipment

Technologies and platforms that were used as part of the project:

Android Studio

Android Studio is an integrated development environment for Google's Android operating system, designed specifically for Android development [4]. The Android Studio IDE was used to develop the Android application. Gradle is a build automation tool for multi-language software development. It supports different languages. **Gradle** was used by the Android Studio IDE to build the application project, containing source code files both in **Java** and **XML**.

DropBox Drive

The Android application uses the DropBox Drive as the storage for keeping the videos. Once user submits a new video to the application, it will upload it into the Drive using the Java SDK provided by DropBox [5]. Then, the backend will get triggered and pull the video from the Drive. After finishing ranking the video, the backend will upload the parsed video with the results, back to the Drive, using the Python SDK, provided by DropBox [6].

PyCharm

PyCharm is an integrated development environment of JetBrains, mainly designed for the application written in the **Python** language [7]. The backend of the application was written with Python using PyCharm. The backend is a Python utility which connects to the DropBox drive (using the Python SDK) and awaits for events. Once triggered by a new video, the utility will pull it and perform image processing operations on it. The collected data will be used as the dataset for the model. For the model part, the Python utility used those packages: **numpy**, **pandas**, **sklearn** and **matplotlib**.

OpenPose

OpenPose is a Real-time multi-person keypoint detection library for body, face, hands, and foot estimation [8]. OpenPose was used to detect user's body and to find the coordinates in each frame. The OpenPose CLI allows to pass a video in MP4/AVI format. The output of the OpenPose CLI is a directory of JSON file, one for each frame of the video. Each such JSON file, consists of a set of keypoints, whose ordering is related with the UI output.

Technologies and platforms that were tested as part of the development process:

MediaPipe

MediaPipe is a library that offers cross-platform, customizable ML solutions for live and streaming media [9]. One of the MediaPipe's features it to detect person's pose and motion in a video. MediaPipe was used to extract the coordinates of the person in the video and was compared against the OpenPose results.

OpenCV

OpenCV provides a real-time optimized Computer Vision library, tools, and hardware [10]. We used OpenPose to split the videos by frames and use the MediaPipe to detect the user's pose on each frame. We also used OpenCV to draw the body lines and the angles that were calculated.

Equipment that was used as part of the project:

Intel's RealSense depth camera D345

Microsoft's Azure Kinect DK camera

4 The development process

For creating our application, we separated our development process into three main stages:

- Creating a detection system which finds rider's body coordinates and calculates the body angles.
- Creating a bicycle configuration ranking system.
- Creating an application where users can use to upload the video and creating a backend which uses the detection system and the configuration ranking system to rank user's video.

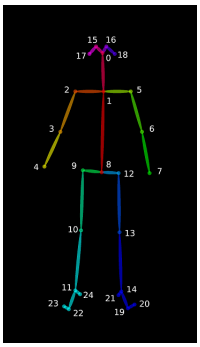
We will go through each stage and describe the decisions we made, the challenges we faced and how we solved them.

4.1 Detection system

In order to analyze the rider's configurations and get the necessary parameters, we needed to analyze the video we received from him. The system we developed requires the user to shoot a short video of him riding a bicycle so that the point of view is from the side, about a meter above the floor so the rider's right side can be seen. Also, since it's hard to distinguish between different individuals, the system requires the video to present only one person riding the bicycle. Given such a video, it is possible to use a dedicated tool or a combination of several tools, to perform image processing on the video and detect the rider's body coordinates. Since we want our system to be reliable, we must use a tool that will provide the best results, before moving to the next stage.

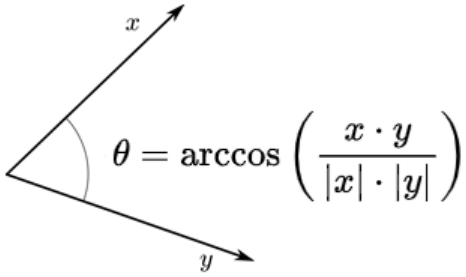
4.1.1 Detection using OpenPose

As our first attempt, we tried using OpenPose, which is an open-source real-time multiple-person detection library that can detect human body, hand, facial, and foot keypoints. The library gets a video as an input and returns the body coordinates of all the people in each frame. In addition, the OpenPose returns the a visual output which is a video that shows all the poses found in the video. We can use this library by passing the video of the user riding his bicycle and getting back the coordinates of his body. The OpenPose library detects 25 different keypoints and it keeps the coordinates of each keypoint in JSON files, one for each frame of the video [11]. Once the OpenPose done with the video, we can iterate over the JSON files and use the body coordinates for different actions such as calculating the angles.



OpenPose's keypoints illustration

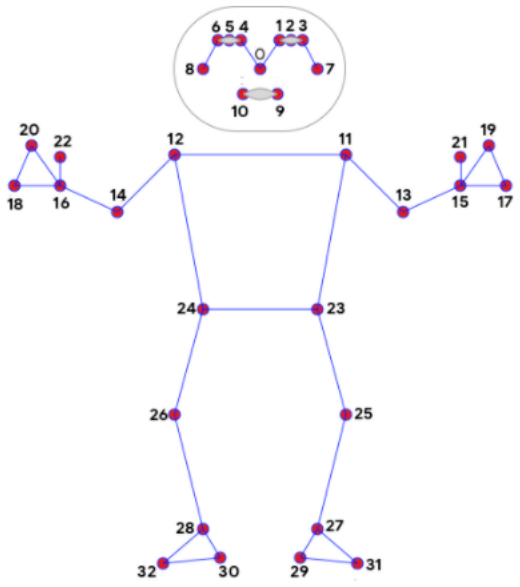
Assuming we have the coordinates of the cyclist's body, we can use them to calculate the angles at certain points that interest us. We can use the dot product between vectors to calculate the angle. The dot product divided by the product of the norm gives the cosine of the angle [12]:



Angle calculation illustration

4.1.2 Detection using MediaPipe

The OpenPose library gave us good results but we wanted to test another utility in order to create a more reliable system. We tried using MediaPipe, which is an open-source library that provides solutions for major Video Intelligence use-cases. One of the MediaPipe's features is to detect and track human pose. We can use this feature to detect and track the user's body, riding his bicycle. We have developed a Python utility which uses MediaPipe to detect the rider's body and track the coordinates by splitting the video into frames. Given the coordinates, we could calculate the angles, the same way we calculated them with OpenPose. Also, we used OpenCV to display the human pose in the video, the same way OpenPose did. Unlike OpenPose which shows 25 keypoints, MediaPipe shows 33 keypoints [13], but most of the added keypoints, such as facial, do not interest us.



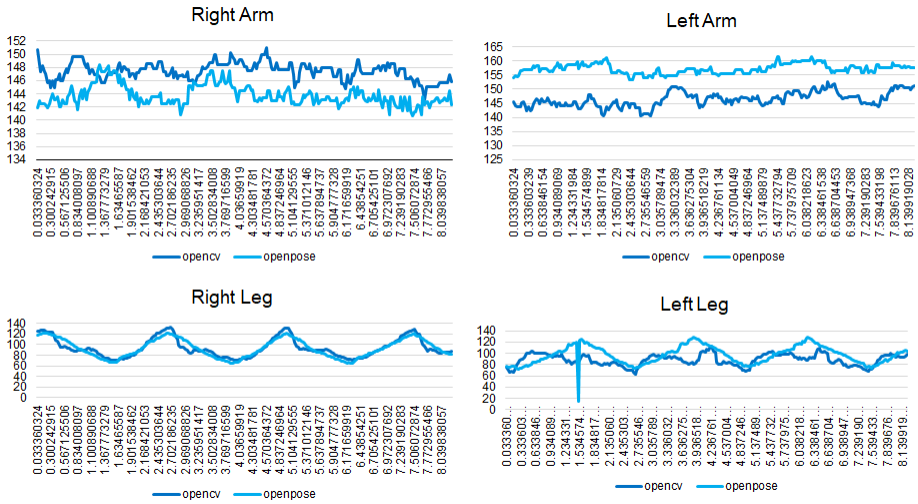
MediaPipe's keypoints illustration

4.1.3 Comparison between OpenPose and MediaPipe

Since we use the same method for OpenPose and MediaPipe to calculate the angles, the best way to compare the two libraries is by comparing the coordinates of the rider body. We created a system that uses the two libraries on five videos of different people riding bikes and compared the results of the coordinates that were obtained. We selected four coordinates - right hand, left hand, right foot and left foot, and compared the results of each one in a graph. The conclusion from all the results was the same and therefore we will present one case. The visual results looked like:

MediaPipe and OpenCV versus OpenPose illustration

We created four graphs so that each graph shows a different coordinate depending on the library we used to find it. Each graph shows the frame number against the coordinate of the keypoint, at that specific frame:



Frame number per specific coordinate illustration

As you can see, for the right leg keypoint, the results are very similar, but for the rest of the keypoints, there are big gaps between the results. Sometimes falls from the side of OpenPose were observed, as in the case of the left leg keypoint, but overall, the results of OpenPose were more accurate and continuous. We also manually compared the results we got by watching the videos and we came to the conclusion that OpenPose is more accurate.

4.2 Configuration ranking system

We have obtained a system that knows how to revive a video of a person riding a bicycle, find the coordinates of the body and calculate the angles. Now we need to understand what is the "proper seating" on a the bicycle and how can we use our new system to detect one.

As our first attempt, we tried to create a set of rules which will indicate if the seating configurations are good or not. Since we are not cycling experts and have no riding experience, we had to do our own research. At first, we tried to read various articles and blogs on the subject. Most of them spoke in general terms and the ideas could not be turned into a set of defined rules. Then we decided to try talking to an expert. We meet with Roi Shifron, the coach of the Technion's bicycle team. He explained to us about the different types of riding and how each type affects the configurations of the bike differently. He also provided us with a number of rules of thumb that can give an indication of whether a person is sitting correctly on the bicycle or not. He also refereed us with Tom Hirshberg, a professional cyclist who also happens to be a graduate student in the Technion's Faculty of Computer Science. With the experience that Tom has with riding and computer science (especially artificial intelligence), she has provided us with some insights on the subject.

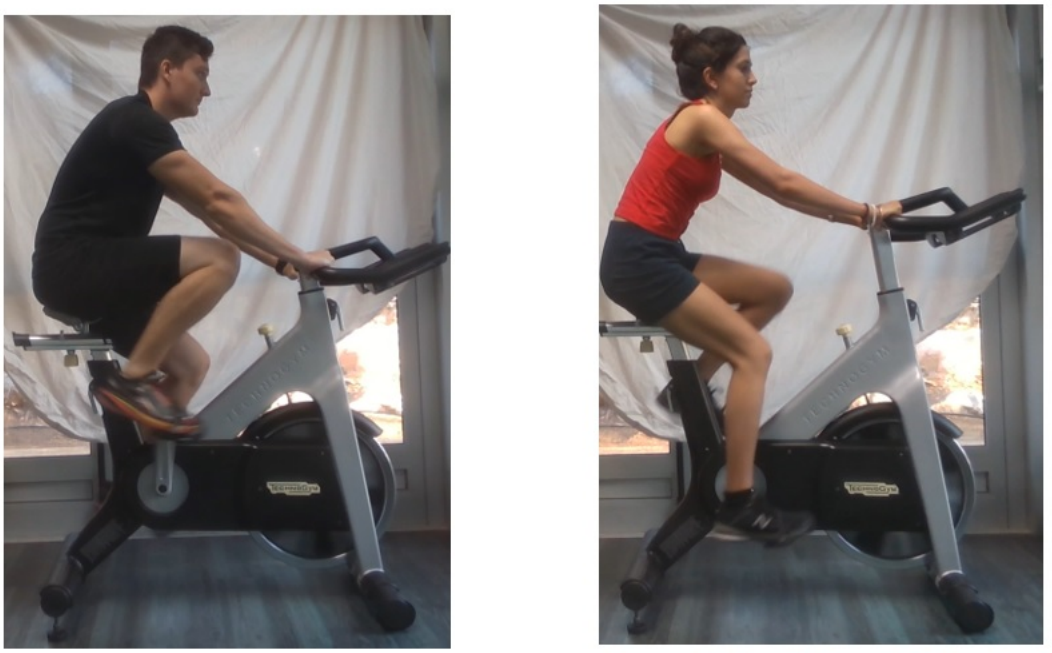
From the conversation with Roi and Tom, we understood a number of important things. The first thing is that we need to decide about what kind of riding, our system is designed to identify. For example, a set of rules for off-road riding would be different from the set of rules for road riding or mountain riding. Both experts suggested using off-road riding which is the most common riding for non-competitive riders. Also, the rider's goal plays an important role in building the set of rules and should be taken into account. Some are interested in improving their riding experience by reducing the amount of pain and some are interested in improving their riding abilities to get better performance in competitions. Both experts suggested building a system for the average rider who is interested in improving his riding experience and is not interested in competitive riding.

Even with the rules of thumb provided for us, it was hard to turn them into something automatic. Experts could not point to a range of specific angle values of the various body points that can give an indication as to whether or not riding configurations are correct. There was no choice but to turn to artificial intelligence to try in solving the task. The idea was to collect data and train a model to make a prediction on a given new video of a cyclist riding his bicycle. Then we could use this model at the backend side of the application. Every time, the beckend will see a new video, he will use the model to rank the bike configurations.

4.2.1 Collecting data

The first task in building a prediction model is to collect data. We used the bikes at our disposal to shoot a number of videos, but very quickly we realized that the task was not simple and required additional attachments. Since nothing holds the bike to the floor, the rider is in motion and the photographer has to move with him. As a result, the recorded video is very blurry and the results of OpenPose detection were not accurate.

Since we do not have a dedicated component designed to hold the bike in place without moving, we decided to use a spinning bike. We arranged a meeting with Hadas Uziel, the Technion's gym trainer, at which we agreed to use the gym's spinning bikes for a small shooting session. During the session, five participants participated. For each participant a video was shot of him riding the spinning bike with different seat and handlebar configurations. Nine videos were shot for each participant and therefore a total of forty-five videos. In addition, for one of the participants, ten additional videos were shot in other configurations. All of the videos were shot using Intel's RealSense camera. In each shooting iteration, we asked the participant to rate from one to five, how comfortable he or she was sitting on the bike. If the participant rated the convenience to be one, then it means that the configurations are not correct at all and if the participant rated the convenience to be five then the configurations are excellent. Also, during the shooting session, we realized that the height of the rider is critical parameter because some of the participants were not able to ride at all for certain configurations. So we decided to add the height of the rider to be part of the final dataset.



The information of the conflagrations illustration

4.2.2 Creating the dataset

Now that we have collected the data, we needed to tag it and create the dataset. The dataset consist of the following set of features:

- All the body coordinates that were detected by OpenPose.
- Some of the body angles that were calculated using the body coordinates.
- The seat and handlebar configurations of the bike.
- The height of the rider.

The label is the ranking of the bike's configurations, which is a value between one to five.

For each frame of each video, the OpenPose tool gave the body coordinates of the rider. We limited each video to 100 frames in order to be consistent. The information in the JSON files that OpenPose produces has been converted to a table where each row represents a specific frame of the video. For each set of body coordinates for a particular frame, we calculated the body angles:

- The angle of the right knee which is calculated by the $(10, 11)$ and $(10, 9)$ keypoints vectors.
- The angle of the right shoulder which is calculated by the $(2, 3)$ and $(8, 1)$ keypoints vectors.
- The angle of the right arm which is calculated by the $(3, 4)$ and $(3, 2)$ keypoints vectors.
- The angle of the right hip which is calculated by the $(9, 10)$ and $(8, 1)$ keypoints vectors.

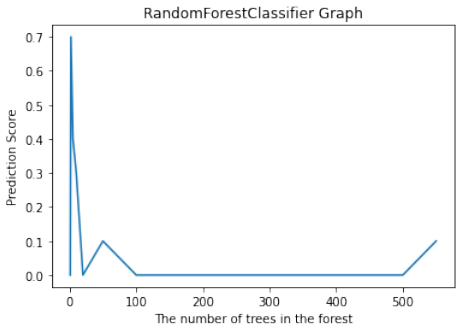
We created three datasets. The first for the training phase, the second for the test phase and the third for the prediction phase. In the training phase we will use a dataset that contains the videos of four participants so that there are a total of 36 videos (nine videos per participant). In the test phase, we will use the nine videos of the additional participant to test the model (which is 20% split of the train dataset). In the prediction phase, we will use the ten videos of the additional participant to test again the model. An important note to understand is that the participant who appeared in the training phase did not appear in the test phase and vice versa.

4.2.3 Creating the model

With ready datasets, artificial intelligence algorithms can now be used to train a prediction model. We used a number of artificial intelligence algorithms for the task. For each algorithm, we created the model, trained and tested it and then saved the results to perform a comparison between the different algorithms. For each such model we tracked two types of scores, the test score and the prediction score, each for a different phase. The test score calculates the accuracy of each frame while the prediction score calculates the accuracy of each video. The test score uses the accuracy method which is a weighting of true positive and true negative results. The prediction score finds the most common rank and compares it with the expected rank.

Different algorithms lead to different results. For each such algorithm, we have chosen a set of constant parameters and additional parameter that we iterate on. For each such iteration, we trained the model and kept the prediction results to present in a graph. We will demonstrate the results of three selected algorithms.

For our first attempt, we used the Random Forest algorithm which combines the output of multiple decision trees to reach a single result. The Random Forest algorithm supports multi-class datasets so there is no need to use the "one-versus-one" (OvO) or "one-versus-rest" (OvR) approaches. When training a Random Forest model, we considered one hyper-parameter - the number of trees in the forest. The parameter that we iterated on, was the value of the number of tress if the forest. The results:

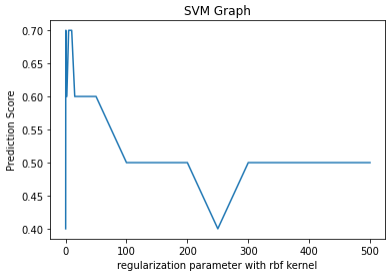


The next algorithm that we used, was the Support Vector Machines (SVM) algorithm. The SVM algorithm is a supervised learning algorithm and is therefore suitable for solving classification problems based on a database of tagged examples. In the SVM algorithm, the training examples are points (x, y) in a plane and the SVM classifier is a separator which creates the largest space between it and the examples closest to it.

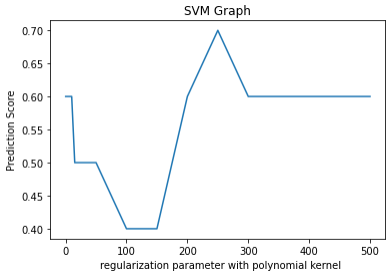
SVMs inherently do binary classification. In binary SVMs, the classifier creates a space between the closest samples of two classes. When a new point is given, the algorithm detects whether it is located inside the line defining the class, or outside it. Since our model requires a multi-class strategy, this kind of SVM is not suitable for the task. Luckily, SVMs can be enhanced to support multi-class models. The Sklearn's SVM implement the "one-versus-one" (OvO) approach for multi-class classification. This approach, splits the multi-class classification dataset into binary classification problems - one dataset for each class versus every other class. In that case, we get $\binom{5}{2} = 10$ classification problems. Each binary classification model may predict one class label and the model with the most votes is predicted by the "one-versus-one" strategy.

We also used the SVM algorithm with different kernels functions - Polynomial, RBF and Linear. When training an SVM with the Radial Basis Function (RBF) kernel, we considered two hyper-parameters - the Regularization parameter and the Gamma parameter. The Regularization parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of the Regularization parameter, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower value of the Regularization parameter, will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. The Gamma parameter defines how far the influence of a single training example reaches. The parameter that we iterated on, was the value of the Regularization parameter as the iteration parameter, with fixed Gamma parameter (like 'scale', 'auto' and others).

For Gamma parameter with the 'scale' option we got:



We also tried the SVM algorithm with the Polynomial kernel function. In that case, we only used the Regularization parameter. The results are:



Next we used the K-Nearest Neighbors (KNN) algorithm. Like the SVM algorithm, the KNN algorithm is a supervised learning algorithm and is therefore suitable for solving classification problems based on a database of tagged examples. The KNN classification is based on the assumption that if two objects are similar, their classifications will also be similar. The idea is that during the training phase, the algorithm stores all training examples and during the prediction phase, the algorithm measures the distance between the object to be classified for each training example and returns the classification of k training examples closest to the object, which is determined by the majority (hence KNN classification is considered as lazy learning). In our case, given an object (dataset of the video), the KNN classifier will find the closest k-neighbors and classify it to the most common class among the neighbors (rank between $[1, 5]$). Since in the training phase, the algorithm only stores the samples and their classification, whereas in the prediction phase, the algorithm measures the similarity between the unclassified object and all the training samples, it means that the training duration time is short, while all the work has to be done in the prediction phase. It follows that for the nearest neighbor algorithm, the training phase is fast but the classification phase is expensive.

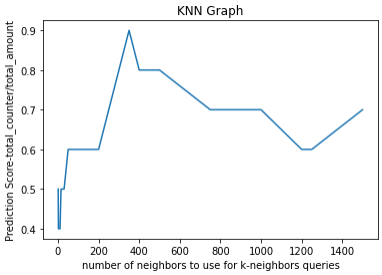
In the KNN classifier, the distance in the feature space is measured and therefore an appropriate distance function which is used by the classifier, is critical for obtaining qualitative results of the prediction. In our case, we selected the distance metric to be the Euclidean distance function. Given a set of features $F = \{f_1, \ldots, f_n\}$, the Euclidean distance is defined as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} \left(d_i \left(f_i(x), f_i(y)\right)\right)^2}$$

Where, for all $i \in [1, n]$, the d_i function is defined by $d_i(x, y) = |x - y|$. We used this distance function because in our case the properties are numeric.

When dealing with learning algorithms that calculate the distance between two points, for example by the Euclidean distance, it is important to check whether there is a need to perform Feature Scaling techniques, in order to normalize the features. The values of the data vary greatly and in some machine learning algorithms, objective functions will not work properly without normalization. If one feature has a wide range of values, the distance will be subject to that particular feature. Therefore, the range of all features should be normalized so that each feature contributes proportionally to the final distance. In our case, we used the MinMax Normalization technique.

The parameter that we iterated on, was the value of K (number of neighbors). We got to 90% prediction success:



The best results were obtained for the KNN algorithm with the parameter $k = 350$ so we will use this model for the prediction.

4.3 Application development

4.3.1 Frontend development

We have built an application for Android written in Java which uses Gradle to build the project. The development process was made with Android Studio IDE. We first developed the login layout which allows users to create a new account or access an existing one. The need for creating a new account for each user is designed to separate between data of the users. We of course do not want a situation to happen where one user has access to another user's data, so we must separate between the users. While creating a new account, each user have to provide an email address and a password. The authentication process is made by the Firebase Authentication service. For that to happen, we had to register our application in the Firebase website. Each time, a user creates a new account, a mail is sent to the email address that was provide, in order to confirm the account. Also, Firebase allows user to recover his password using the email address, in case he forgot it.

Once the user is logged in to the application, he is being redirect to the dashboard layout. We designed the dashboard layout to support two different states, depending on the amount of videos in the drive. One state displays the videos and their results and another state displays a message, saying that there are no videos in the drive. While the application is fetching for data from the drive, it displays a progress circle. We used the DropBox drive for the storage of the videos and the DropBox Java SDK to fetch data from the drive. We had to register out application in the DropBox website to obtain an access token which will be used for accessing the drive using the SDK.

User can navigate between different using a special component that was added. The navigation component is a side drawer, similar to the one of the Gmail application. It allows the user to move between the "dashboard", "upload", "about" and "exit" layouts. By using the navigation component, user can move to the "upload" layout, where he can upload a new short video of him riding a bicycle. In addition to the video, the user needs to provide his height and a unique name that will represent the video in order to separate it from the other videos that he already uploaded. The application supports videos only in AVI or MP4 format and so if the user will try to upload a video in another format, the application will display an error in a Toast widget. In addition, the application does not allow uploading videos larger than 20 MB, since there is no reason for a five-second video to be larger than this limit. Once the user provides the information, the application stores it in the drive using the DropBox Java SDK.

As soon as the backend has finished handling the video and recognizes the level of convenience of the bike's configuration in the video, it will upload it back to drive. If we go back to the dashboard layout, then we can see the original video, the video with the user's skeleton and the result of the prediction. If necessary, the user will have to change the configurations of the bicycle and repeat the process.

4.3.2 Backend development

The backend is a cron, written in Python, which waits for events to happen in the DropBox drive using the DropBox Python SDK. It listens to changes in the drive and in case of new video, it will fetch it. Since the application uploads user's video with his information, to the drive, the backend can fetch a new video from the drive. Once the video is downloaded, it will run OpenPose to detect the user's body coordinates. Using the body coordinates that were found in the JSON output files, the backend can calculate the body angles. As soon as all the information is available, the backend composes the dataset. Previously, we have obtained a model that, given a dataset of a video, can make a prediction of how correct the rider's configurations are. The backend can use this model to perform prediction on the dataset that it constructed. Once the model have ranked the user's bicycle configurations, it uploads the video with the user's skeleton and the result of the prediction, back to the drive. The application can now fetch this information and present it to the user in the dashboard layout.

An important thing to note here is that the backend can be ran on any machine, but the recommendation is to use a machine with a GPU so that the detection process performed by the OpenPose tool will be fast and the user will not have to wait long for the results.

5 Future Work

Splitting the rank label

Instead of using a single label in a dataset for a general rank, it is possible to set a label for each configuration. For example, one label can be set for the configuration of the seat and another label for the configuration of the handlebar. In this way, it is possible to explain more about what the user needs to do in order to improve his bicycle's configurations.

Collecting more data

Our training dataset consists of four different participants and the testing dataset consists of one participant. Collecting more data will allow to train a better model and get better prediction results. In addition, the group of participants should be diverse so that it contains people with different physique such as tall, thin, etc.

Use of actual bicycles and not spinning bicycles

The project began with an attempt to find convenient configurations for bicycles. After consulting with number of experts, we realized that off-road bikes are the most common. We tried to shoot videos of riders on off-road bikes but very quickly realized that it is very difficult to shoot them, without a dedicated component that holds the rider in place. Therefore, it was decided to use a spinning bicycle in the Technion's gym. The infrastructure is already ready and it is possible to shoot videos of riders on off-road bikes (using the dedicated component) and train the model.

Migrating from the drive to database

We have used the DropBox drive as the storage for the videos and the user's information, such as his height. A more correct design is to configure a database where information about the video and other details like user's height will be stored. The application will use this database to store and fetch the information and the DropBox drive will be used for keeping the actual videos only. You can also use external storage platforms (such as Amazon's S3) to keep the videos.

Improving the UI

Currently, the UI of the application is really basic and does not follow any predefined set of design principal. An attempt should be made to make the application follow a set of design principal like of Google's Material Design.

Publishing the application

Currently, the application is only available by direct download of the APK. Since the application is not fully production-ready (due to previous reasons), we didn't release it to the Google Store. In the future, it is possible to upload the application to the Google Store, once it's fully ready.

6 Conclusion

We have built an applications that allows users to upload videos of themselves riding their bicycles and get a good indication of whether their configurations are correct. This applications allows users to configure their bikes in order to get the most suitable configuration set for them to reduce pain and enhance the riding experience.

In this project we showed a Proof of Concept (PoC) of the bike fitting process on a simple spinning bicycle using Machine Learning algorithms. Our work is focused on determining whether this idea could be turned into a reality and this study proves that it's possible. We have built an initial infrastructure that can be expanded and used for follow-up projects that are interested in trying to solve the problem in a more generic way. For example, the follow-up projects could collect more data of riders and enhance the infrastructure to support off-road bicycles, instead of spinning bicycles. Also, the project can be adapted to a similar application that is designed for gym training, rowing, etc.

Aside from our achievements, the project granted us an opportunity to learn many new and interesting things in the computer science study and develop useful skills. We got our first experience with image processing development, by using dedicated utilities and libraries such as OpenPose, MediaPipe and OpenCV. Also, we learned important concepts in the world of artificial intelligence, by using machine learning algorithms that helped us solve the task we faced. In addition, we got to experience developing Android-based applications, build a network between our application and the backend and use Firebase for the authentication. Overall the project allowed us to expend our knowledge in diverse fields and a useful experience that could serve us greatly in the future.

7 Acknowledgments

We want to thank Tom Hirshberg, Roi Shifron, Hadas Uziel and all the other participants for their kind help and their contribution to the success of the project.

8 References

1. **Bike fitting cost and duration**
   Link: https://www.evolvebikes.com/articles/bike-fitting-pg85.htm
2. **Firebase Authentication service**
   Like: https://firebase.google.com/docs/auth
3. **Android Toast widget**
   Link: https://developer.android.com/reference/android/widget/Toast
4. **Android Studio documentation page**
   Link: https://developer.android.com/studio
5. **DropBox Drive Java SDK**
   Link: https://www.dropbox.com/developers/documentation/java
6. **DropBox Drive Python SDK**
   Link: https://www.dropbox.com/developers/documentation/python
7. **PyCharm documentation page**
   Link: https://www.jetbrains.com/pycharm
8. **OpenPose documentation page**
   Link: https://cmu-perceptual-computing-lab.github.io/openpose
9. **MediaPipe documentation page**
   Link: https://google.github.io/mediapipe
10. **OpenCV documentation home page**
    Link: https://docs.opencv.org
11. **OpenPose Keypoints-UI Mapping**
    Link: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/md_doc_02_output.html
12. **Dot product calculation explanation on Wikipedia**
    Link: https://en.wikipedia.org/wiki/Dot_product#Geometric_definition
13. **MediaPipe Keypoints-UI Mapping**
    Link: https://google.github.io/mediapipe/solutions/pose