GR^{2}AM: Towards Gesture Recording, Recognition, and Application Mapping

Physical Computing Seminar
Summer Semester 2021

Abdulrahman M. Selim
Saarland University
Saarbrücken, Germany

Akash Sinha
Saarland University
Saarbrücken, Germany

Joshua Sonnet
Saarland University
Saarbrücken, Germany

Abstract. This work describes a proof of concept towards creating a simple-to-use software in the scope of Human-Computer Interaction, to create a gesture recognition, and application-mapping system, in hopes of facilitating the interaction with smart computing and IoT devices. In efforts to do so, we created GR^{2}AM, a hybrid-learning based model, with a user-friendly interface, to easily record gestures, either from a predefined set of nine gestures, or create up to six unique custom gestures, map them to trigger certain applications, and process incoming, real-time, data streams. The hybrid-learning model consists of a 1D-CNN and a Random-Forest Classifier, trained on a minimal dataset of 10 samples for each gesture, to produce a real-time accuracy of 82.85%, from a stream of 30 frames of 21 hand landmarks captured using a built-in webcam by using the MediaPipe Python Module. We showed the feasibility of creating such an application using standard equipment, that is easily used by normal users.

Acknowledgment

We want to thank Adwait Sharma for his help with our work and with his broad knowledge in this area, during the seminar.

1 Introduction

According to Merriam-Webster, a gesture is a movement usually of the body or limbs that expresses or emphasizes an idea, sentiment, or attitude [1]. So, with this definition in mind, we can understand that gesture recognition is a method of understanding these ideas, sentiments, or attitudes. Humans are, normally, able to understand common gestures and act upon them, and due to them being easy to perform and understand, a lot of research focuses on trying to make computers understand these gestures, so that we can use them to facilitate using and controlling devices all around us.
The focus of trying to use gestures to control devices increased with the rapid adaptation of smart devices, which are devices that are aware of their environment or surroundings, and are able to perform computations, and connect with other devices for data exchange [2], and the main context under which smart devices operate is called Internet of Things (IoT), which describes a network of devices, each containing different sensors, software programs, and technologies for the purpose of connecting and exchanging data [3], so that you can actually interact with different devices embedded in the surrounding environment by just performing a few gestures.
To build upon this thought process, we saw that nowadays users can use their own biometric data like fingerprints or facial recognition to create customized security methods to access their device so, why cannot users create their own custom hand gestures to use them in communicating with their devices to perform a set of user predefined tasks.
In this report, we are discussing our work in creating GR^{2}AM, a demo, or a proof of concept, of an easy-to-use User Interface (UI) that enables users to record their own hand gestures, either from a predefined set of nine gestures, or create up to six unique custom gestures, using a minimal amount of samples, to train a hybrid learning model, created from a 1D Convolutional Neural Network and a Random Forest Classifier integrated into a hybrid model, then map the recorded gestures to a set of available applications to interact with, with a real-time live classification accuracy of 82.85%.

GitHub The full code base can be found online on GitHub https://github.com/jsonnet/GR2AM

2 Related Work

Gesture recognition is an active research area, so there is a large body of work within this field, and we read a lot of them, to concretely agree on our project idea, but here we will state the most relevant publications directly related to our project. First of these related works, which is also the most relevant, we have to mention Project Prague by Microsoft [4], which is a Software Development Kit (SDK) that enables developers to define a set of gestures to control applications, as it has

similar functionality in a way. However, Project Prague needs a depth camera like Intel RealSense or Microsoft Kinect, in addition to it being an SDK so, normal users might find it a bit difficult to use.
There are other lines of research that implement different gesture recognition methods in controlling home appliances, to facilitate the smart home experience. For example in a publication by Nguyen et al. 2019 [5], they wanted to create a system that utilizes mobile sensors within a smartwatch, smart-phone, and the house itself to control various devices, but they wanted the user to learn and use their own set of predefined gestures, but our approach differs in that we want to make a generic solution, that has a set of predefined gestures, but can also accommodate custom gestures that the user might want to try to use.
In order to see which gestures we should try to test our system on, and include as the predefined set of gestures, we utilized the work from a recent publication by Sharma et al. 2021 [6], in which they test a variety of micro-gestures that can be performed by any of the five fingers, while holding various objects. They performed various gestures, but we chose three to test out, which were tap, flexion or swipe-up, and extension or swipe-down. We chose to follow this work, because they have a concept of gestures that can be performed, uniformly, by all five fingers within one hand, and can later be used while holding objects, so we picked three as a proof of concept for the feasibility of detecting them using our approach.
Lastly, for an easy-to-use UI, we got inspiration from Teachable Machine by Google [7] which is a web-based tool that facilitates exploring the usage of using machine learning algorithms for various applications. The main idea behind their project as stated on their official website, is that it should be accessible to everyone, and to do so, they need a good UI, that is why we followed their main principles when designing our solution.

3 Technical Background

In order to actually recognize these gestures, we used Classical Machine Learning (ML) Algorithms and Neural Networks (NN). So, in this section, we will give a brief overview of the working principles of the algorithms used.

Classical ML Algorithms ML Algorithms are mathematical methods used to understand and recognize patterns available in the data, by learning from a set of predefined data known as a training set [8]. We started by comparing two classical ML algorithms, which are

- Support Vector Machine (SVM) Classifier
- Random Forest (RF) Classifier

SVM The SVM classifier works by trying to find a hyperplane or a border, that separates the different classes while reducing the distance between the classes and the hyperplane [9]. There are different types of hyperplanes or kernels to try, for example a linear kernel as can be seen in Figure 1 or a polynomial kernel.

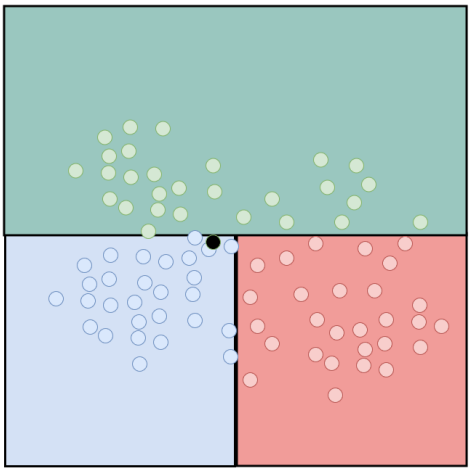

Figure 1: An example of a linear kernel SVM Classifier

RF The building block of the RF classifier is the decision tree, which is a structure built using the extracted features or the data that we calculated in the previous step, where each tree takes a subset of the features and keeps splitting until it reaches a classification, and the random forest consists of multiple decision trees making classifications and the classification that the majority of the trees agree on, is the one used as the predicted label [10]. Figure 2 shows an example of an RF classifier with four trees. The RF classifier can have a few parameters to modify, for example the number of trees in the forest and the maximum depth of each tree, meaning how deep should we keep splitting until we make a decision regarding the classification.

1D-CNN Neural Networks are processing systems inspired by how human brains process data, it consists of a network of information processing elements arranged in layers, and working in parallel to understand the data meaningful [11]. One type of neural networks, is a one-dimensional Convolutional Neural Network(1D-CNN). Classification of time series data using 1D-CNN is similar to classical ML approaches, where we extract temporal features of the time series data and classify them. What features to extract depends on the weight of the kernels of the CNN layer, which are learnt as part of the learning process. The extracted features are then forwarded to a fully connected dense NN layer, the job of this layer is to take the features as inputs, and classify them into an output, which, in our case, is a gesture. Figure 3 shows the basic

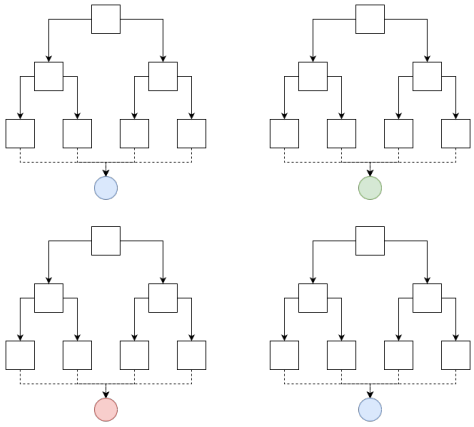

Figure 2: An example of a Random Forest Classifier

intuition behind the classification of time series data, where some relevant features are extracted, and then classified into output classes.

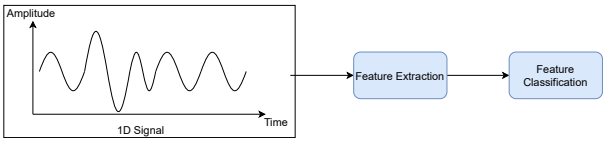

Figure 3: An example of classifying a one dimensional time series data

To understand how 1-Dimensional (1D) CNN work to extract temporal features from time series data, we can imagine that the input signal is of the shape $Input\_Dimension * Sequence\_Length$ where Input_Dimension is the number of features of the input signal, and Sequence_Length is the number of instances where the signal is sampled. In a convolutional layer, a kernel is convoluted across the Input_Dimension of the input signal, as shown in Figure 4.

If we wish to extract just the temporal features and not the spatial features, then the output dimension of such a convolution operation leads to a vector of shape $Input\_Dimension * 1$. Each kernel is a way to extract a particular statistical temporal feature out of the time series data. Therefore, if we wish to extract more temporal features, then the number of kernels has to be increased.

4 Implementation

4.1 Gestures

The cornerstones of the whole work and GR^{2}AM, are the hand gestures that will be used afterwards to control various applications.

4.1.1 Gesture Capturing

One of the important aspects in the work that we wanted to achieve, was to use a normal laptop camera or webcam for capturing the gestures so that the solution can be easily used without the need of buying expensive depth cameras, and nowadays, almost everyone either has a webcam or a camera already embedded within their laptop. In order to achieve that, we used the MediaPipe Hands framework [12], which is an open source and freely available solution by Google, that is able to track and recognize hand and finger positions using normal webcams and producing 21 landmarks in each frame as can be seen in Figure 5 and each landmark holds three coordinates which are **x**: the width, **y**: the height, and **z**: the depth, but the z coordinate represents the depth with the wrist being the origin as can be seen in Figure 6. We used Python Programming Language for our implementation [13]. In order to make use of MediaPipe, we also needed to make use of OpenCV (Open Source Computer Vision Library) which is an open source computer vision and ML software library [14]. All the gestures were captured within a 30-frame box window, due to software limitations from MediaPipe being able to only capture 15-20 frames per second. So, some gestures took less than 30 frames, but the maximum number of frames for the recorded gestures was 30, which amounts to approximately a second and a half. For the gestures that were less than 30 frames, we padded them by adding frames filled with zeros (0, 0, 0) so that all the captured gestures were 30 frames.

4.1.2 Gestures Set

To start, we wanted to try a few simple gestures to see how the overall system should be constructed before trying more complex sets of gestures. We initially started with three gestures, which are tapping with the index finger, tapping with both the index and middle fingers together, and tapping with the index, middle, and ring fingers together.
Afterwards, we started trying more complex sets of gestures, which included performing tapping, swipe up, and swipe down with all five fingers, so we had 15 different gestures. These gestures have similarities between them, and learning models might find difficulties in differentiating between them using a minimal number of training samples, that is why we chose these 15 gestures to train our models on, after the initial set, so that we can test it out under more complex situations. However, afterwards we decided to remove the ring finger and pinky finger gestures, so we ended with nine predefined gestures, which can be seen in Figure 7.
In addition to these nine gestures, we collected some negative class gestures, which are gestures that can occur, but do not trigger any applications, like a still hand or a hand moving fast within a frame and so on. We added these negative classes, because our end application works as a live classifier that keeps receiving a stream of coordinates, and that can cause some problems for example false classifications which can

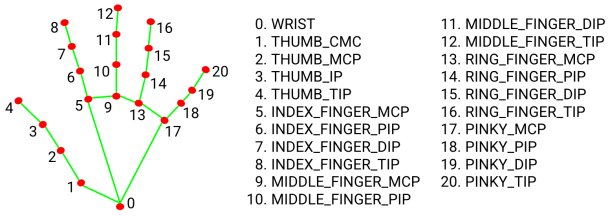

Figure 5: 21 Landmarks from MediaPipe

Source: MediaPipe. [Online]. Available: https://google.github.io/mediapipe/solutions/hands.html

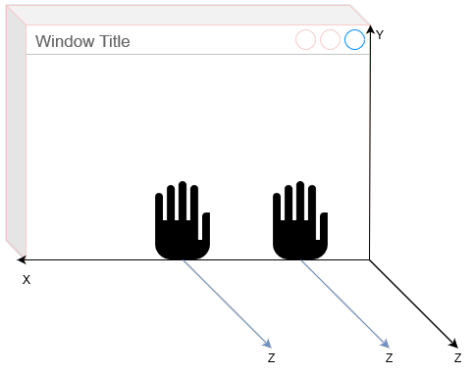

Figure 6: The (x, y, z) axis with respect to the computer screen, and the Z origin being the wrist itself

lead to false triggering of applications.
We collected the gestures from three participants, each performing each gesture ten times, while in some cases more data was recorded to further test the system as a whole.
Before actually sending the data to the learning models, we had to perform a preprocessing step, where we normalize the data, which is discussed in detail, in the following part.

4.2 Data Preprocessing

Before sending the data captured from MediaPipe to the classifier, we needed to perform a normalization step, because the x and y coordinates that MediaPipe captures, are normalized with respect to the computer screen, meaning that the top left corner represents (0, 0) and the bottom right corner represents (1, 1) with the step size depending on the camera resolution. We tried various normalization trials in order to see which one gave us the required results. Initially, we tried to translate the hand coordinates so that the wrist would always be at (0, 0) with respect to the screen. Then, we tried making the first frame being at the origin and all the subsequent frames relative to this frame, so we, in a way, only capture the movements with respect to the first frame. Afterwards, we tried normalizing with respect to the wrist meaning that the wrist became the origin at (0, 0), and the other landmarks were relative to it, and then we subtract the mean so that we end up with values ranging from (-1, 1).
Finally, we reached our final normalization, which works by scaling the hand coordinates according to the palm width, which is the distance between the Metacarpophalangeal (MCP) joints of the index and pinky fingers, then we reference all the coordinates according to the wrist joint of the first frame, which becomes the origin (0, 0, 0), and finally we subtract the mean so that we end up with values ranging from (-1, 1). To summarize or better understand all of this, the process of how a single frame is normalized, is seen in Algorithm 1.

4.3 Learning Models

To actually be able to recognize the different gestures, we needed to try different algorithms to see which ones work out best. We started by testing various algorithms in an offline classification to test the performance, before trying it out in a live classification.

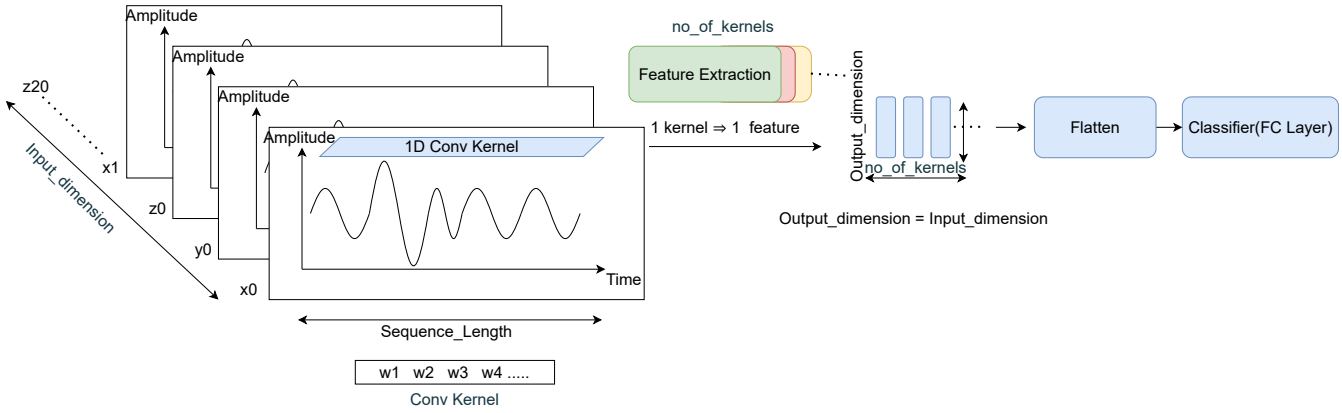

Figure 4: An example of a CNN

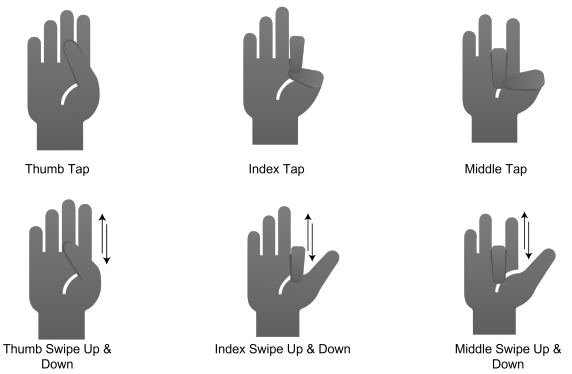

Figure 7: The nine predefined gestures

Algorithm 1 Gesture Normalization

$Base\_Scale = 0.12$
$Frame[i] = ((x[i]0, y[i]0, z[i]0).....(x[i]20, y[i]20, z[i]20))$-
where $i = 0, 1, 2, ...29$
$Landmark[i][X] = x[i]0, x[i]1, x[i]2, ...x[i]20$
$Landmark[i][Y] = y[i]0, y[i]1, y[i]2, ...y[i]20$
$Landmark[i][Z] = z[i]0, z[i]1, z[i]2, ...z[i]20$
$Index\_MCP[i] = (x[i]5, y[i]5, z[i]5)$
$Pinky\_MCP[i] = (x[i]17, y[i]17, z[i]17)$
$Palm\_Width[i] = \sqrt{(x[i]5-x[i]17)^2+(y[i]5-y[i]17)^2+(z[i]5-z[i]17)^2}$
$Scaling\_Factor[i] = \frac{Base\_Scale[i]}{Palm\_Width[i]}$
$Landmark[i][X] = Landmark[i][X] * Scaling\_Factor[i]$
$Landmark[i][Y] = Landmark[i][Y] * Scaling\_Factor[i]$
$Landmark[i][Z] = Landmark[i][Z] * Scaling\_Factor[i]$
$Reference = Landmark[0][0] = (x[0]0, y[0]0, z[0]0)$
$Landmark[i][X] = Landmark[i][X] - Reference[i][X]$
$Landmark[i][Y] = Landmark[i][Y] - Landmark[i][Y]$
$Landmark[i][Z] = Landmark[i][Z] - Reference[i][Z]$
$Landmark[i][X] = Landmark[i][X] - Mean[i][X]$
$Landmark[i][Y] = Landmark[i][Y] - Mean[i][Y]$
$Landmark[i][Z] = Landmark[i][Z] - Mean[i][Z]$

4.3.1 Classical Machine Learning

In the ML approach, two main algorithms were tested, which were SVM and RF. For both algorithms, the approach was similar in that we first extracted some statistical features for the data, split the data into 70 % training and 30 % testing, but in case we had an unequal number of samples, we got the same number of training samples for each gesture and any remaining samples were used for testing, and then we fed the training data to the classifier and tested the fitted classifier on the remaining testing data. An overlook of the general pipeline concept can be seen in Figure 8.

We will start by looking at the feature extraction step and then look deeper into each classifier.

Feature Extraction Figure 9 shows a sample of what a single file containing a gesture looks like, where each row represents a single frame each consisting of 21 landmarks each has 3 coordinates. To extract the features, we extracted them column-wise for each landmark, so that we capture the time information within the features. So, if we extracted N features, we would end up with a $21 \times 3 \times N$ matrix representing the extracted features for each coordinate within each landmark.

The Python modules used to process the data were SciPy, NumPy, and Pandas [15] [16] [17]. The main module we used to extract the features was TSFEL, which is a python package used for feature extraction on time series data [18]. For TSFEL, we extracted multiple statistical features, but ended up using the features seen below.

- Mean
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$
- Median Absolute Deviation
$$MAD = Median(|(x_i - \bar{x})|) \tag{2}$$
- Standard Deviation
$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{3}$$
- Variance
$$Var = \sigma^2 \tag{4}$$
- Kurtosis
$$Kurt = n\frac{\sum_{i=1}^{N}(x_i - \bar{x})^4}{(\sum_{i=1}^{N}(x_i - \bar{x})^2)^2} - 3 \tag{5}$$
- Skewness
$$Skew = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^3}{(N-1) * \sigma^3} \tag{6}$$

Classifiers For the classifiers, we used the Scikit-learn Python package [19].

SVM We chose the SVM algorithm, because as a classifier, it is used in multiple applications in various domains, while producing good results, along with it being used in various gesture recognition systems. As a classifier, it can be used in multi-class classifications with a number of kernels to choose from, which include a linear kernel, polynomial kernel, and radial basis function kernel. We tried all three kernels and the best resulting one was the linear kernel.

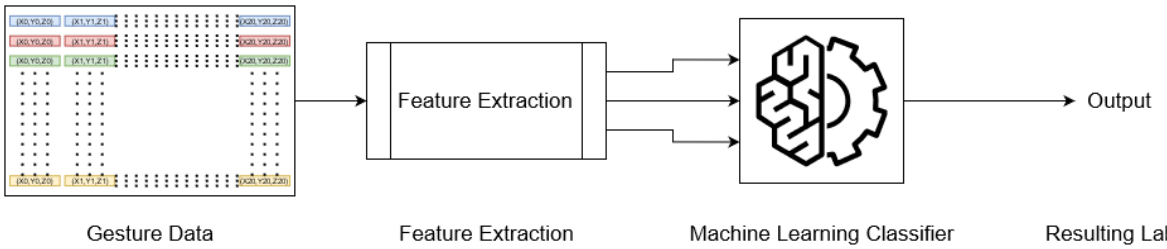

Figure 8: ML Working Principle

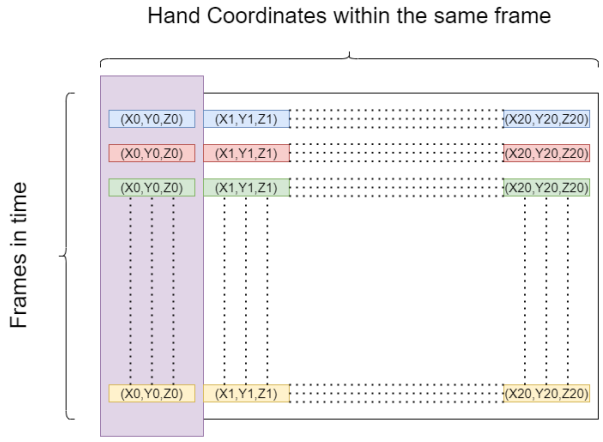

Figure 9: A file sample of a gesture captured using MediaPipe

RF The RF classifier is a versatile classifier, that can be used in multi-class classifications with great success in various application domains, that is why we chose to try it here as well. There are various parameters to tune in an RF model, we mostly looked at the number of trees in the forest and the maximum depth, meaning how deep should the tree go, until when should it keep splitting the data. The depth of each tree, after trying various values, the default produced the best result, which means it keeps splitting, until each leaf is pure, or contains just one sample. On the other hand, the number of trees in the forest, affected the results greatly, and that was the main parameter that affected how both the offline and live classifications performed.

Overall, we ended up using mainly the RF classifier moving forward as it gave us the best performance for all the test participants and even in the live classification.

4.3.2 Neural Networks

NN Architectures For the NN learning model, we tried various approaches, and two of them are discussed below. All the approaches were implemented using Pytorch [20].

GRU and a Fully Connected Layer This approach consisted of Gated recurrent units (GRU) and a Fully Connected Layer. In this approach, the GRU network was used to extract temporal features from the gesture data. The extracted temporal features were then passed to a fully connected dense layer for classification.

1D CNN and a Fully connected layer This approach consists of multiple components. There are two 1D CNN layers, the first later consists of 128 kernels, while the second layer consist of 64 kernels with Tanh as an action activation function after each layer. We applied Batch normalization to the output of convolutional layer and also used a Dropout layer after the convolutional layer for regularization. The features extracted after the second convolutional layer are then passed to a fully connected dense layer, where a Softmax activation is used, which then classifies the extracted feature into one of the gesture classes. The whole approach can be seen in Figure 10.

We ended up using the 1D CNN and a Fully connected layer, because it had better performance than the other approach.

Training the model To train the NN, there were a lot of parameters to adjust, and the following parameters were the ones we ended-up using for the final approach. We used Mini-Batch gradient descent algorithm, with a Batch Size of 16 for training our network. The Mini-batch gradient descent works by splitting the training data into smaller batches that are used to calculate the error and update the model. Also, we used Adam's optimiser, which dynamically adjusts the learning rate for each weight of the neural network. We used the base learning Rate of 0.02 and an Epoch size of 800.

4.3.3 Hybrid Model

After trying the ML and NN each by itself, we tried a Hybrid Model, which utilizes both approaches to provide the result. In Algorithm 2, there is a pseudocode describing the process, it basically depends on the confidence of each model in its classification result, we found that the 1D-CNN model performed accurately if it had a confidence of more than approximately 80%, while the RF classifier performed accurately when it had a confidence of more than approximately 15%, so after fine-tuning the different values to perform rather accurately

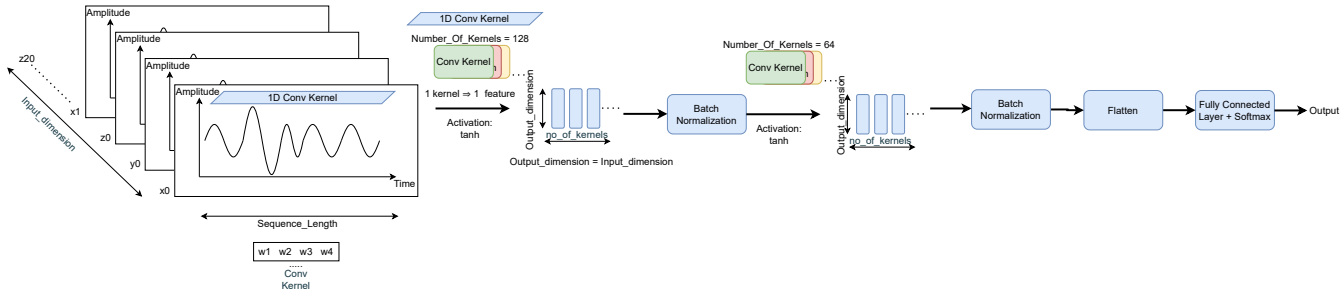

Figure 10: 1D CNN and a Fully connected layer NN

across the available participants, we used a confidence threshold of 82% for the 1D-CNN and 17% for the RF classifier.

Algorithm 2 Hybrid Model Output

```
if NN Confidence > 82% then
    return NN Output
else if ML Confidence > 17% then
    return ML Output
else
    return Negative Class
end if
```

4.4 Live Stream

A major part of GR^{2}AM was to actually enable live classifications, meaning that the participant after recording the data, and generating a learning model, they should be able to use the model to classify gestures during a live video capture. In order to do so, we had to go through multiple steps. All the steps work in parallel, each in a separate multiprocess, while communicating with each other via queues. We start by capturing the stream using the same combination of MediaPipe and OpenCV, but instead of saving the data into a file, we have a buffer that collects 30 frames, which is our window size, but we created an overlap between the windows because a gesture might actually occur in between windows. So, we applied a 60% overlap which means that each window holds 18 frames from the previous window and 12 new frames and so on. We chose that overlap value after rigorous testing to evaluate different thresholds. Before passing the data forwards to the hybrid model, we perform the normalization step, after which the data is passed by another queue to the hybrid model to get the prediction result back, and then forwards the prediction to perform or activate the mapped application. For the ML classifier, the TSFEL feature extraction took some time, which produced a noticeable lag between performing a gesture and getting a result back. In order to overcome this, we used the features extracted by TSFEL, but written in Pandas and NumPy, which noticeably improved the delay between performing the gesture and getting the classification.

5 Evaluation

Procedure progress For our evaluation we used data from three participants. For the initial three gestures, we were able to correctly classify all the test data correctly for all the participants, this produced an accuracy of 100%. After the initial results with the simplified gestures, we started using the set of 15 gestures. We had to exclude the gestures from one of the participant because MediaPipe was unable to detect the hand due to poor lighting conditions. For this set of gestures, we went through various stages of implementations and trials. Initially just recording the data, and padding them with the largest frame size, meaning that if the largest gesture had 60 frames, we padded all the gestures with zero-filled frames until all of them were 60 frames long. For this approach, we were able to achieve an RF accuracy of 95.56% for participant 1 (P1), and 88.67% for participant 2 (P2). We then started applying a predefined fixed window size of 30 frames, which caused P1 to get an RF accuracy of 75.5% and a 1D-CNN accuracy of 81.63%, and P2 to get an RF accuracy of 95.56% and a 1D-CNN accuracy of 95.63%.
P1 therefore recorded a new set of gestures that within a shorter period to fit the 30-frame window, but both P1 and P2 added negative classes to their gestures, which produced a 1D-CNN accuracy of 92% for P1, and 94% for P2. The results of the live classification at this point were an issue, so we reduced the number of gestures by removing the Ring and Pinky gestures, due to their difficulty, for all the participants, to perform which caused false classifications. After doing so, we achieved, for offline classifications, for P1 an RF accuracy of 97.22% and a 1D-CNN accuracy of 94.44%, and for P2 an RF accuracy of 91.67% and a 1D-CNN accuracy of 100%, as seen in Figure 11.

Final result After the last results, we recorded a video where we perform the gestures five times each, and then feed the video to the classifier to classify them in real-time and to facilitate the evaluation on our side. With the model trained on seven samples from each gesture, we achieved an HL ac-

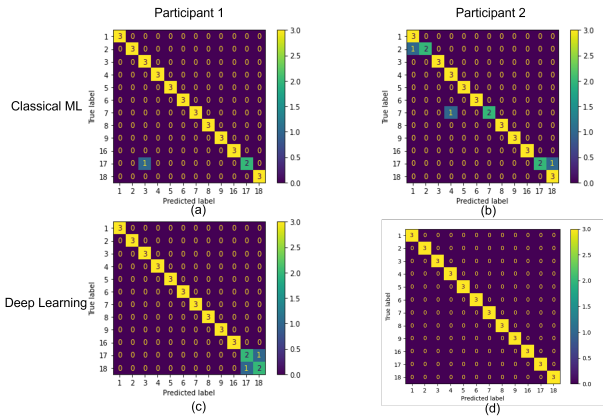

Figure 11: Confusion Matrix for offline classifications of the final learning models

curacy of 51.1%. Then we increased the training dataset to include 14 samples from each gesture, which produced an ML accuracy of 55.6% and an NN accuracy of 62.2%, but we then removed the Index swipe-up and Middle swipe-down gestures from the calculations and were able to achieve the accuracies of 51% for NN, 71% for ML and 82.85% for the HL.

6 User Interface Design

6.1 CMD UI

During the development process, we started with a command-line standalone UI, which was able to record gestures, generate a learning model, and do live classifications with the labels of the recognised gestures constantly being displayed in real-time. The recording feature was implemented to be controlled by the keyboard, where you press “Space bar” to record, “n” for the next capture, “r” for a redo, and “q” to switch to the next gesture in line to be recorded. We added to the camera feed shown on the screen while recording the gesture, the functionality to show a red exclamation mark, if the current recording was a lot longer than the 30-frame window size. In conjunction with the Web UI, the static UI was mainly implemented to be used for on device demonstrations by using different command-line arguments, to start the individual modes, or set the path to a certain model_folder and start the client-side software, without starting the web based UI.

6.2 Web UI

For the end product, we developed a web-based UI. We used Flask [21] for the web backend framework, Bootstrap [22] for the frontend framework, Jinja [23] a template engine, CSS [24] for styling and Javascript [25] for all interactive part of the Web UI, during development of the UI.

Login The web UI starts off with the login screen as seen in Figure 12, where different users can be created by choosing a username, or log in if they had already created a user with the same name before. We added the login functionality because the gestures, gesture-application mappings, and learning models are saved, for each user, independently.

Main page Once logged in, the user is presented with an overview page of their already recorded gestures, available applications, and the already created gesture-application mappings. They can also start the application for live classification, as well as be able to log out, all this is seen in Figure 13. There are also notification messages shown to the user, to indicate whether a specific task was completed correctly, or whether an error occurred, to provide them with feedback.

Adding new gestures The user can add new gestures by clicking the plus symbol in the left most column, which will in turn lead them to the second page, which can be seen in Figure 14. On this page, the user is presented with three columns.The left column, shows a list of available gestures, which includes a set of predefined gestures that they can record, along with the possibility to record up to six custom gestures, which the user can give a custom name, the middle column shows a preview of the user’s camera and controls once the recording is enabled, and the right column shows the recorded gestures. Each recorded gesture can then be deleted by clicking the red “X” symbol next to each gesture. Furthermore, each predefined gesture has a preview animation, that will show how the gesture is intended to look like. Once a gesture has been selected from the left column, the user will hit the confirm button, and their camera will be enabled, then they can click the red “record button” and perform the gesture and once finished, they click the blue “next button” to get ready to perform the next trial, or the yellow “redo button” in case of a mistake while performing a certain trial. A progress bar located at the top of the middle column indicates the percentage of collected trials. After reaching 100% the site automatically reloads, and the camera gets disabled and ready for the next gesture. In the case of the site reloading while recording a gesture, or the user leaving before performing enough trials for a certain gesture, this gesture will be highlighted and coloured in red to make it visible to the user, when the user selects it to continue recording, the progress bar will indicate the already recorded samples in its counts. Afterwards, when the user finishes recording the gestures they want, they can click on the generate model button, which starts training the model, then a resulting confusion matrix, pops-up after the models are trained successfully.

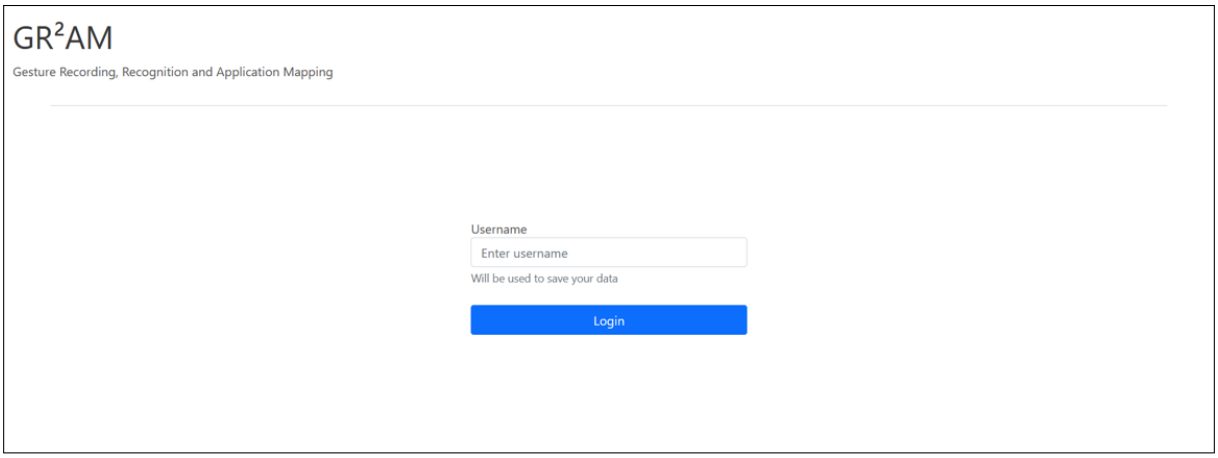

Figure 12: Web UI login Page

Mappings Finally, the user can go back to the main page and create their own set of gesture-application mappings with the newly recorded gestures. This is done by selecting the gesture as well as the application to map it to and submit the mapping. To unmap a gesture, it is enough to only select the gesture without an application. For all the UI parts, if a button is disabled, its functionality not clear to the user, there are tooltips to point the user in the right direction on what to do, or what the button functionality is.

7 Applications

As previously stated, the end goal is to use our application to control various applications and smart devices like lamps, outlets, playing or pausing some media, or controlling a variety of other applications. So, once the user records a gesture, and generates the learning model, they map the gesture to an application which will be triggered once the gesture is recognized by the live classification. As part of the current implementation, we modelled the system as a client-server software, where all the processing is done on the server-side. So, when the user performs a certain gesture, it will be processed by the server-side, which will then send the command back to the client-side to control certain elements of the user's PC. A variety of applications can be triggered currently, for example, increasing and decreasing the PC volume, playing and pausing media, and skipping and rewinding media. An overview of the whole process, from performing the gesture till the application is triggered, can be seen in Figure 15.

8 Feasibility of Graspable Objects

As a way to test the possibility of using our application to detect gestures while holding different objects, we tried to see if MediaPipe would be able to detect the hand while holding different object types, as can be seen in Figure 16. The different objects follow the types of objects discussed by Sharma et al. 2019 [26]. We noticed, that due to MediaPipe needing at least an outline of the upper part of a hand to recogise it as one, it had trouble identifying the hand when holding big objects, where most of the hand would be covered. So when holding a big box up, with the hand being mostly behind, it could not detect anything. With a small piggy bank, it tried to recognise the hand, but failed by mistaken the wrinkles in the hand for the fingers, similarly when holding it from the top, it could not fully figure out where the hidden fingers would be, therefore failing the part of the recognition. On the other hand small objects where just the palm of the hand would be covered, or where a clear outline of the four upper fingers would be recognisable, succeeded.

9 Discussion

Limited sample size With the current implementation, the application is able to work with an acceptable accuracy, given the fact that only 10 samples are used for each gesture to train the model. The number of samples is a limitation, because the current learning models require more data, to perform significantly better, but the downside of increasing the number of samples is that it can overwhelm the user and 10 samples produce acceptable results given that fact.

Limited model size In addition to the number of samples, not being able to add more than 18 gestures in total – resulting in only six custom gestures – is also a limitation of the current learning models. However, this is not due to the model, but due to the fact that the parameters are tuned for 18 gestures, and the model would have to be constantly retuned if we were to add more than this maximum number of gestures.

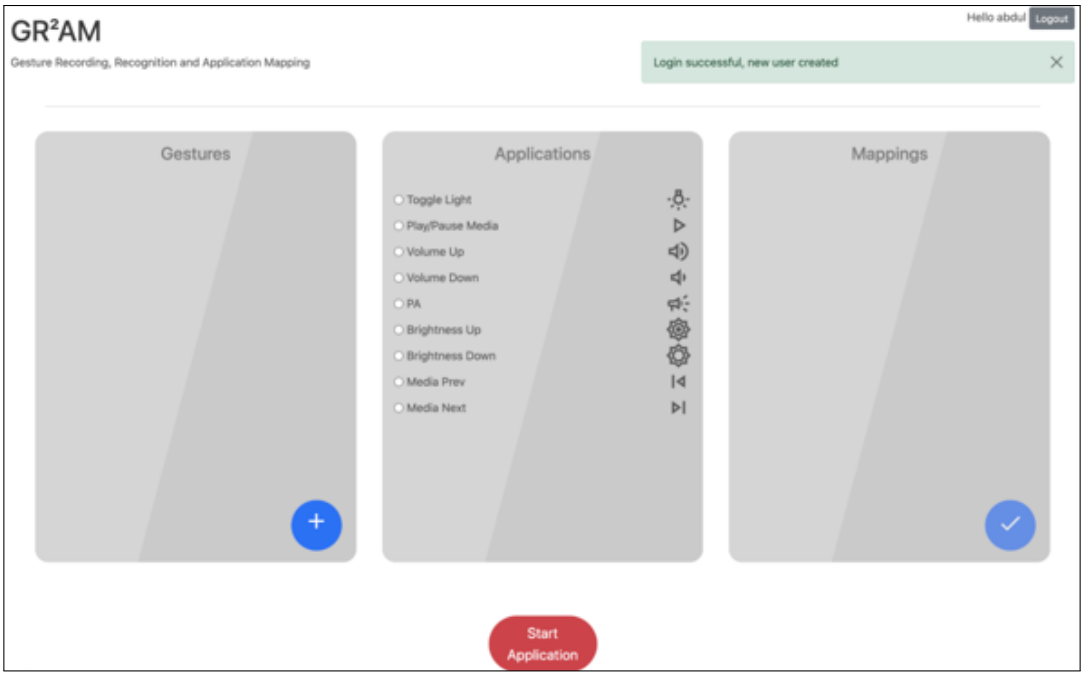

Figure 13: Web UI First Page

User dependent model Further, we noticed that people perform the same gestures, slightly differently from one another, which then reflects in the classifier's ability to correctly detect the gestures, even after scaling and normalising. With this in mind, after we found this reflected in the initial results from the testing phase, we opted for a user dependent model, which might have added a bit of extra workload on the user, but improved the overall performance.

Security risks As previously stated, one of our goals was to use a normal off-the-shelf 2D camera, not a depth camera. However, they come with certain security and privacy concerns. In our application, we only actually record the user's hand landmark coordinates, so no image or video feed is actually being recorded, so even if the recorded data of multiple users were to be leaked, they will not be able to distinguish which user did which gesture, in normal circumstances, achieving k-Anonymity. With that said, the hardware on one hand can be susceptible to security risks, such as being intercepted by an attacker capturing the feed, but on the other hand a far more likely approach would be a remote attacker, that would intercept the stream right on the device itself from a software point, so before it is being processed by GR^{2}AM.

10 Future Work

For the future work on how to improve the current version, there are a number of steps, that can be taken to do so. For example, trying different Deep learning models, which might be better than the current 1D CNN, to improve the overall accuracy, with the minimal amount of training data for each gesture. The ability to add custom applications, is also a needed feature, so the whole application can cater to each user's own needs, more personally. As stated, the current models are tuned to have a maximum of 18 gestures, so finding ways to make the number of possible gestures increase, is also a proper way to improve the overall end product.

Finally, as a futuristic look, we thought that the use of single board computers, that hold a camera, can make the application easier to use and place, while adding a user-recognition system, that automatically recognizes which user is currently performing the gestures, to automatically switch to their costume gestures and gesture-application mappings.

11 Conclusion

We have shown that it is possible to create a simple and easy to use UI, that is able to use a normal 2D camera connected to a computer, to record and recognize costume gestures, to easily control various applications, on a computer, or a smart device, using a minimal number of recordings. But there are certain limitations we have seen to MediaPipe, which uses an ML model trained to recognise a hand based on camera images, as well as limitations to our own learning models. For the former one, while recording training data, we noticed the importance of sufficiently good lighting, especially as webcams have trouble with dynamic range in image processing, which makes it hard for MediaPipe to recognise the hand

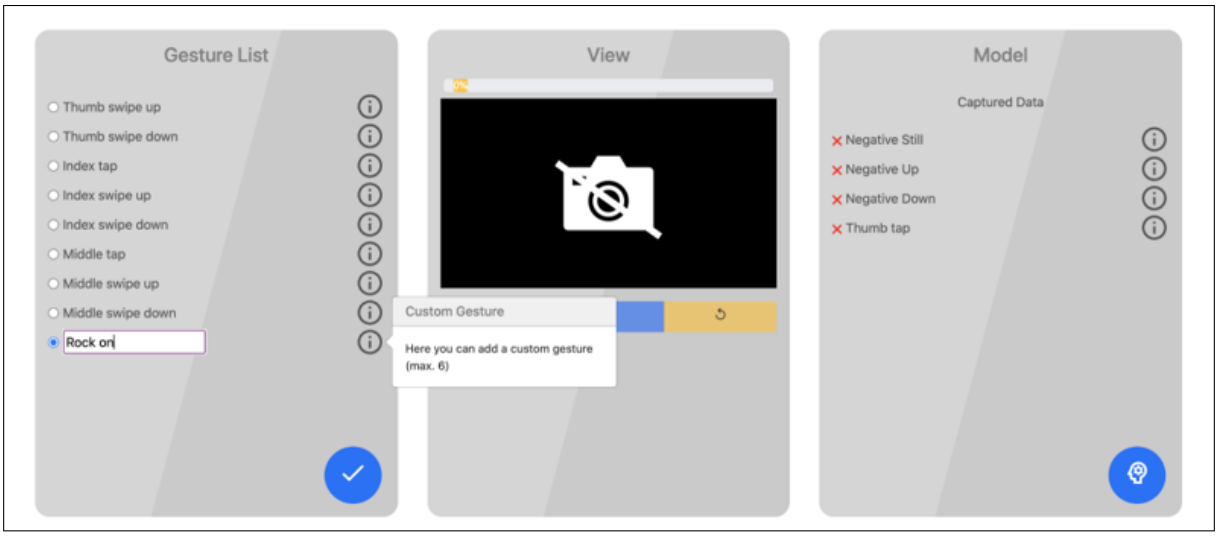

Figure 14: Web UI Second Page

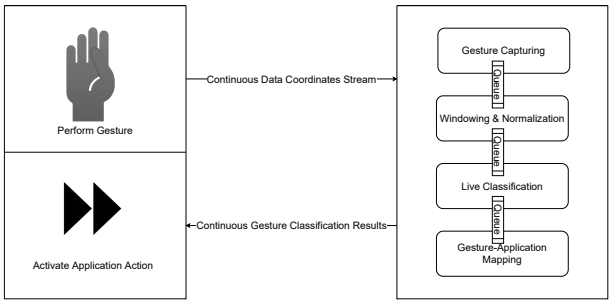

Figure 15: Processing Pipeline Architecture Overview

based on the data it is trained on. In regard to our own models, we noticed that these models work better when they receive a greater variety of training data, especially when recording the samples, it is easy to repeat the gesture without a wide variance, which in return is detrimental for the live classification, where each gesture is done in a continuous motion. Therefore, we looked for a middle ground between having a minimal amount of data and the need to still be user-friendly. In the end, we still achieved a real-time live classification accuracy of 82.85%, with one participant recording all gestures in a normal manner, so without paying great attention to optimal variance and lighting. The offline accuracies of the models, in comparison, were much higher due to the nature of the collected training data, where we know the exact start and finish times of each gesture, the controllability of the collection process itself, and of no noise being produced by sudden hand movements. This proof of concept proves the feasibility of creating GR^{2}AM, a user-friendly UI capable of performing live gesture-application mappings which can be customised to each user independently.

References

[1] Gesture. 2021. In Merriam-Webster.com. Retrieved August 7, 2021, from https://www.merriam-webster.com/dictionary/gesture.

[2] Silverio-Fernández, M., Renukappa, S. & Suresh, S. What is a smart device? - a conceptualisation within the paradigm of the internet of things. Vis. in Eng. 6, 3 (2018). https://doi.org/10.1186/s40327-018-0063-8

[3] What is IoT? [Blog post]. Retrieved from https://www.oracle.com/internet-of-things/what-is-iot/

[4] Microsoft Project Prague - Hand Gestures SDK https://www.microsoft.com/en-us/research/project/project-prague/

[5] Nguyen, Trong Khanh & Ha, Bui & Pham, Cuong. (2019). Recognizing Hand Gestures for Controlling Home Appliances with Mobile Sensors. 10.1109/KSE.2019.8919419.

[6] Adwait Sharma, Michael A. Hedderich, Divyanshu Bhardwaj, Bruno Fruchard, Jess McIntosh, Aditya Shekhar Nittala, Dietrich Klakow, Daniel Ashbrook, and Jürgen Steimle. 2021. SoloFinger: Robust Microgestures while Grasping Everyday Objects. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Article 744, 1–15. DOI:https://doi.org/10.1145/3411764.3445197

[7] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In Extended

Figure 16: (A) Example of Successful hand detection while holding objects, (B) Example of Failure to detect the hand while holding objects

Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20). Association for Computing Machinery, New York, NY, USA, 1–8. DOI:https://doi.org/10.1145/3334480.3382839

[8] Kamalakannan Palanichamy, Chapter 19 - Integrative Omic Analysis of Neuroblastoma, Editor(s): Loo Keat Wei, In Translational Epigenetics, Computational Epigenetics and Diseases, Academic Press, Volume 9, 2019, Pages 311-326, ISSN 25425358, ISBN 9780128145135, https://doi.org/10.1016/B978-0-12-814513-5.00019-2.

[9] Khadija El Bouchefry, Rafael S. de Souza, Chapter 12 - Learning in Big Data: Introduction to Machine Learning, Editor(s): Petr Škoda, Fathalrahman Adam, Knowledge Discovery in Big Data from Astronomy and Earth Observation, Elsevier, 2020, Pages 225-249, ISBN 9780128191545, https://doi.org/10.1016/B978-0-12-819154-5.00023-0.

[10] S. Suthaharan, Chapter 6 - A Cognitive Random Forest: An Intra- and Intercognitive Computing for Big Data Classification Under Cune Condition, Editor(s): Venkat N. Gudivada, Vijay V. Raghavan, Venu Govindaraju, C.R. Rao, Handbook of Statistics, Elsevier, Volume 35, 2016, Pages 207-227, ISSN 0169-7161, ISBN 9780444637444, https://doi.org/10.1016/bs.host.2016.07.006.

[11] Yi-Ping Phoebe Chen, Elena P. Ivanova, Feng Wang, Paolo Carloni, 9.15 - Bioinformatics, Editor(s): Hung-Wen (Ben) Liu, Lew Mander, Comprehensive Natural Products II, Elsevier, 2010, Pages 569-593, ISBN 9780080453828, https://doi.org/10.1016/B978-008045382-8.00729-2.

[12] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C. L., and Grundmann, M. (2020). Mediapipe hands: On-device real-time hand tracking. arXiv preprint arXiv:2006.10214.

[13] Van Rossum, G., and Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

[14] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[15] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J.

Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

[16] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke and Travis E. Oliphant. Array programming with NumPy, Nature, 585, 357–362 (2020), DOI:10.1038/s41586-020-2649-2

[17] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010), (http://conference.scipy.org/proceedings/scipy2010/mckinney.html)

[18] Barandas, Marília and Folgado, Duarte, et al. “TSFEL: Time Series Feature Extraction Library.” SoftwareX 11 (2020). https://doi.org/10.1016/j.softx.2020.100456

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011) (https://jmlr.org/papers/v12/pedregosa11a.html)

[20] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[21] Grinberg, M. (2018). Flask web development: developing web applications with python. "O’Reilly Media, Inc."

[22] Bootstrap HTML, CSS, and JavaScript framework. Online. https://getbootstrap.com/

[23] The Pallets Projects. Jinja templating engine. Online. https://palletsprojects.com/p/jinja/

[24] Duckett, J. (2011). HTML & CSS: design and build websites (Vol. 15). Wiley Indianapolis, IN.

[25] Flanagan, D. (2006). JavaScript: the definitive guide. "O’Reilly Media, Inc."

[26] Adwait Sharma, Joan Sol Roo, and Jürgen Steimle. 2019. Grasping Microgestures: Eliciting Single-hand Microgestures for Handheld Objects. In CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK. ACM, NewYork,NY, USA, 13 pages. https://doi.org/10.1145/3290605. 3300632