Hand Gestures Recognition Using a WearOS Smart Watch and Deep Learning

Mihai Fleșer [1, 2]

[1] SC X2 Mobile Development SRL
37 King Ferdinand Street, 400027,
Cluj-Napoca, Romania,

[2] Technical University of Cluj-Napoca
Cluj-Napoca, Romania
mihaifleser@gmail.com

Teodor Ștefănuț

Technical University of Cluj-Napoca
Cluj-Napoca, Romania
teodor.stefanut@cs.utcluj.ro

ABSTRACT

Nowadays, smartwatches have become cheaper and more affordable, meaning that people can integrate this technology and use them in their everyday life. Fortunately, the manufacturers enabled third party developers to build their own applications in order to greatly improve people's life from everywhere. In this article we will discuss an approach to detect specific arm gestures using WearOS powered smart watches and a custom imported neural network. The main objective is to evaluate the processing capabilities of these new devices when working with neural networks, which are known to produce substantial delays between the moment when receiving the input and producing the output. The initial dataset is composed of three gestures described by the watch's accelerometer and gyroscope, while the final product is an application that enables users to control a music player running on a paired smartphone, using just their hand movement.

Author Keywords

WearOS; Movement Classification; Neural Network; Wearable sensors; Deep Learning; Time-series signal processing.

ACM Classification Keywords

Machine learning

DOI: 10.37789/rochi.2023.1.1.8

INTRODUCTION

We are aiming to enable people to do certain actions by only using their wrist via the smartwatch they are wearing. Since it's very convenient to move your hand when wearing a smartwatch, we considered that it would be very useful to control some functions from the phone by just making gestures with your hand.

The system should be able to perform multiple functions at the same time. The most important feature it must achieve is feeding live sensors data to the model and being able to quickly get the output result, to be responsive and suitable for the fast-paced world we live in. Moreover, the application should be able to quicky and reliably communicate its internal status via Bluetooth to another application installed on the user's phone.

Our motivation is the need to be able to control phone's functions from outside, by making use of the watch's sensors, especially the accelerometer and gyroscope. The possible applications of gesture recognition can be applied to many domains. Sign language recognition, Parkinson's syndrome or fitness exercises are just a few of them, and developing applications available for common people can greatly increase the overall life quality of users.

RELATED WORK

The current existing alternatives of movement detection products running on WearOS are just the ones already provided by the smartwatch's operating system. These solutions can detect basic hand gestures which are used to wake up the watch when the user tilts his hand. However, methods of detecting gestures were researched in the past, with promising results.

In paper [1], the authors developed a pattern recognition system for time-series signals. The scientists analyzed two applications of human gesture recognition: human activity recognition (HAR), and gesture recognition for limb amputations. For the first application, they used raw sensor input like gyroscope and accelerometers signals, while for limb amputations they used surface electromyography (sEMG). Four different deep learning models were used: a one-dimension convolution neural network (1-D CNN), a long-short term memory model (LSTM), a hybrid model containing one convolutional and one neural network model (C-RNN) and, a model containing three convolutional layers and three recurrent neural layers. (3+3 C-RNN). The resulted trained model can then be used on applications ranging from entertainment interfaces to prosthetic arms.

In paper [2], the authors used the accelerometer and gyroscope sensor values to train a deep learning model. The recurrent neural network consists of multiple LSTM layers which receive the data mapped as a time-series. These layers are capable of learning long-term temporal dependencies. The last layer is a softmax that calculates the probabilities that some specific values belong to a specific class.

Article [3] had similar objectives as our project. The scope was to design a neural network that could be easily integrated into an android project. Even if the authors used a smartphone and not a smartwatch, a lot of good ideas and inspiration came to us from their work. The project purpose was to build a model capable of detecting tilt movements done by holding a phone in the hand.

Other papers also presented similar approaches to the ones described above. In paper [4], using deep neural networks with body warn sensors describing the acceleration and rotation, they manage to extract various features further used in human activity recognition.

Also, the authors of [5] used convolutional neural networks for recognizing the human activity with the help of body warn sensors. They also used the idea of sliding window to continuously detect new movements, managing to obtain an accuracy of 90%.

All the approaches presented in these papers helped us shape a better idea on how to use Neural Networks together with WatchOS operating system to recognize specific hand gestures. As there is no publicly available dataset that can be used to train the network on recognizing the gestures we want to detect, we had to build it ourselves, on top of the whole detection system.

PROBLEM DEFINITION AND ANALYSIS

The main purpose of our endeavor was to discover a reliable method for combining WearOS devices with neural networks and create a real-time solution for recognizing specific user gestures. Although in the present there are quite a few devices from different manufacturers that are using WearOS, only few applications such as health monitoring and fitness tracking are targeting them.

Our research's objectives were the following:

- Building a solid application architecture, following the most up to date best practices and the most recent technologies in this field.
- Achieving a reliable communication channel between the smartwatch and the phone, while maintain a real time flux of messages between the two of them.
- Recording and storing a complex and big database on the physical device, even though the memory capabilities are quite restrictive.
- Exporting this big data in a third python application, in order to use it to train a fully custom neural network.
- Importing this new neural network into the application and obtaining the output in almost real time.

The problem of having a custom neural network model running on a physical smartwatch is quite complex, because such a network requires significant computing power and resources. Our application demonstrates the fact that users can control different phone functions using gestures while wearing a smartwatch.

For the project demonstration we built a mp3 music player, and the users will be able to control the music using just wrist gestures. However, the potential of this kind of detecting application is huge, as it can be applied to medical or industrial fields, depending on what use-cases are needed. The system is built to easily integrate new data, new gestures and new procedures that shall be needed for the corresponding field of activity.

Product Perspective

Our research is intended to be part of a bigger system that puts to good use the custom gestures detection capabilities of the model. The implementation must be easily customisable and the process of inserting new gesture and recording new data should be as convenient as possible. Even though the main watch application has been created using a stand-alone template, it is intended to be used in conjunction with a phone in order to communicate the corresponding detected action to the overall system. Therefore, the following block diagram can be used to describe the system overall behavior:

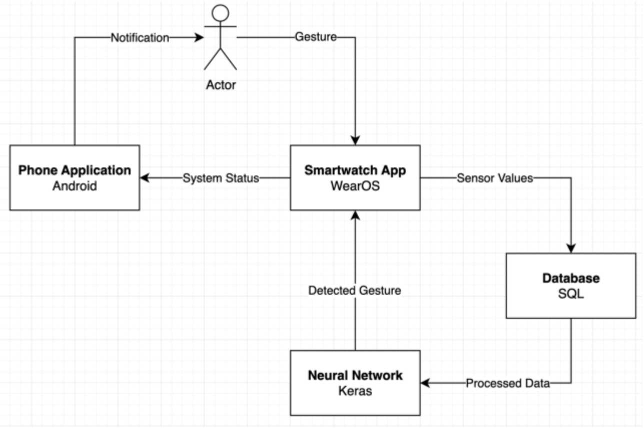

Figure 1. System's perspective

The flow firstly begins with the primary actor, which is also an important stakeholder, mainly concerned with the overall well-being of the whole system. The diagram then follows the flow of data collected from the user and stored inside the smartwatch app.

The next step is the actual way of saving this data inside the local systems database. This database consists of different tables stored locally on device.

After that, the next step is processing this data in order to start training and validating the model. Then, the trained model is imported into the smartwatch application which, in turn, will start feeding the neural network with new data.

In all this time, the system status is continuously communicated to the phone application via Bluetooth, that will send notifications to the users in order to let them know about all the system components and their state.

DATA MODELLING

The following chapter has the objective of analyzing the structure of the application data along with the procedures used when collecting this data and preparing it for training the custom model. This is crucial part of the system, especially because an appropriate database was not publicly available at the time of writing this paper for training the neural network. We made the decision on recording three gestures, as different as possible, because of the following reasons:

- It is easier for the neural network to differentiate between one another, while also facilitating the learning of common features from the gesture of the same type.
- Because we had to record this data manually, the number of recordings will be quite reduced, so the neural model must learn to distinguish the gestures using a modest sized database.

The starting position is done by keeping the left hand, face up, in front of the user, near the belly. All the gestures should be done in approximately one second. After the starting position is achieved, the following gestures were recorded to be used in the model training:

Up-down motion: This motion is intended to be done in a very simple way, as it just requires the user to raise his hand from the starting position to a point near the chin, then coming back. This movement is especially described by the variation of the acceleration on the Z axis. Thus, the accelerometer is the key sensor that will be taken into consideration when labeling this action.

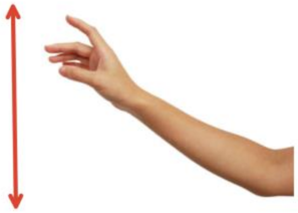

Figure 2. Up-down motion.

Left-right motion: The second movement is the left and right motion. This is again a very simple gesture, and it can be described as moving your hand to the left approximatively 70 degrees, then coming back to the starting position. On contrast with the first movement, this action will engage the X and Y accelerometer axes, consequently making these values determinant in the gesture's recognition.

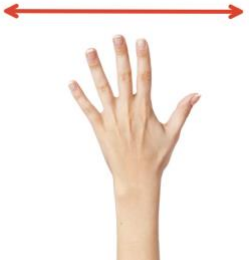

Figure 3. Left-right motion.

Rotate-wrist motion: The third movement is the rotation of the wrist from the starting position about 60 degrees and the coming back. While the first two movements used the accelerometer extensively, this movement is intended to engage the gyroscope.

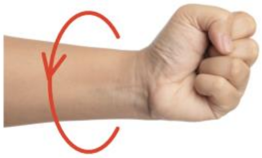

Figure 4. Rotate-wrist motion.

Data structure

The sensors used to record the gestures are just two: the accelerometer and the gyroscope. The values recorded will consist of X, Y, Z accelerometer and gyroscope readings.

Each gesture will be described by a collection of consecutive frames, where each frame will have all these values written above.

A gesture will have the duration of 1 second, with a duration of 10 milliseconds between each data frame, up to a total of 100 frames. The gestures will be saved in the database organized in batches, where each batch represents a gesture of a certain type with 100 consecutive frames.

Each Batch will have a type corresponding to the label assigned for that gesture. Using these values, the data will be further processed and fed to the neural network.

The data will be stored on the local device and will consist of two tables, the Batch and Measurement. The Measurement will store the sensor values. It will also have a reference to a particular gesture, in order to not lose track of the gesture's measurements.

The id of the gesture and the gesture's type is stored in the Batch class. This is the class referenced by the Measurement. Each gesture is then mapped to a batch, and each batch will have 100 measurements corresponding to them. This diagram describes the relation between these entities:

Data collection procedure

The data collection procedure is quite straightforward and simple. The user will have to firstly put the watch on their wrist. After that, they will be able to choose which kind of gesture they want to record. After choosing the type of gesture, the watch will start recording the data.

In order to register only the gesture data, as much as possible, there is a movement intensity threshold that needs to be surpassed before the watch will start recording the gesture. This setup will prevent recording when the user touches the watch to select the gesture type or to initiate the recording. The limit is small, so as soon as a more prominent movement is detected, the watch will start saving the recording.

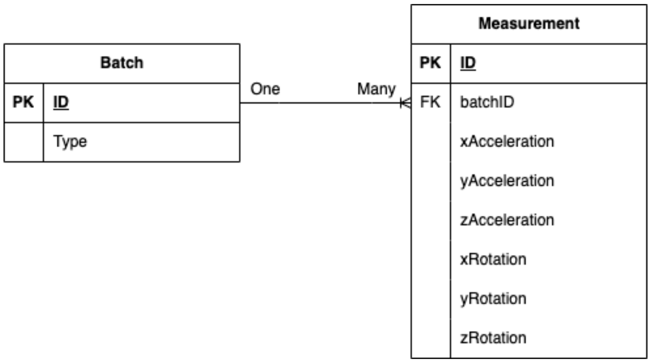

Figure 5. Database Diagram

However, the watch always listens to the sensors readings and keeps track of the last 15 values recorded. When the threshold is triggered, these readings will be automatically introduced in the start of the time series, in order to better monitor the beginning of each gesture.

When the data for the gestures has been recorded the watch was placed on the left-hand. As the accelerometer and gyroscope axis are changing direction when wearing the device on the right hand, recorded data cannot be used to recognize gestures described with the right hand. We have collected a total of approximately 600 instances, 200 for each of the three gestures mentioned above. Each of them was done from the starting position, while sitting still on a chair. All the gestures were recorded on one of the authors.

NEURAL NETWORK MODELLING

This research focuses on using a supervised learning approach, because all the gestures are labeled beforehand, and the goal is to find common patterns in the data in order to be able to consistently classify gestures. To do this, have tried different configurations of neural networks and have chosen one that balances accuracy with processing speed.

The deep learning models are defined by a collection of blocks named layers, which are interconnected and used to find common patterns between the input data. As presented in the book [6], the neural model is inspired by the way human brain functions. However, the easiest definition of a neuron in a neural network can be expressed as a function that maps the given input to the desired output. The neuron receives a set of input data, applies different weights, and tries to obtain the desired output.

Neural networks are essentially functions approximators. This means that if we were given an arbitrary function, the neural networks can represent it, no matter how complex or arbitrary the function may be. The neural networks are also scalable and flexible, because we can stack more layers, provide different types of activation functions and we can tweak neurons parameters in order to fully suit our needs.

The neural network used in this research is a feedforward neural network, also known as a multilayer perceptron [7]. This network consists of several dense layers with a variable number of neurons, where each neuron computes the weighted sum of the inputs and applies an activation function to produce the output.

The sequential model built lets us stack layer after layer on top of each other. The first one is the input layer. Since the input data is an array of 100 elements, each of them having the acceleration and the rotation on X, Y, Z, the shape of the input data is (100, 6). Then, there are two dense layers with 20 and 5 neurons respectively. These neurons have the RELU activation function, detailed in [8]. In summary, this function ensures backpropagation and empowers the model to learn complex data patterns. As mentioned, we have tested with different number of neurons, but in the end this configuration gave the best results. The last two layers are the Flatten layer and another Dense layer. The flatten is used to make the input for the previous layer as a just one array in order to be fed to the last layer. The last layer has a softmax activation function, which outputs the probability of each specific class and is usually used in this kind of multi class problems.

This neural network was implemented with the help of Keras framework. More information on this subject can be found in [9] and [6].

The model was further converted to a tflite file which was imported into the watch application. The watch then feeds sensor data to the model and expects its output. Then, it forwards the detected gesture to the phone application, which based on the interpretation associated with the gesture, will change/pause/resume the music.

APPLICATION DESIGN AND IMPLEMENTATION

Android Platform

Android is an operating system developed by Google for mobile devices such as smartphones and tablets. It is based on a modified version of the Linux kernel. The most important components of any Android application are the activities [10]. Unlike desktop applications, which usually have the entry point the main() method, an Android application have a special main activity which is used as the first screen. Usually, every activity implements a screen, so an application consists of many other activities that transmit information between them and let the user navigate between several screens in order accomplish his task. Every activity has a lifecycle associated with it, which describes the way that activity performs in certain moments in time, depending on the user's action.

WearOS Platform

WearOS is an operating system for smart watches, developed and maintained by Google. Many manufacturers have produced compatible watches, just like in the case with Android phones. They were made in order to benefit of call answering, notification managing, vital signs monitoring or fitness tracking.

The Android platform and the WearOS platform share a lot of common features and lifecycle aspects. The application model is also composed of activities and fragments, both having the same behavior between the two platforms.

However, since the display is considerably smaller and the resources are quite limited, the platform developers encourage prioritizing the usage of activities rather than fragments. Moreover, there are also some other principles and advice that should be taken into consideration when developing an application for the smartwatch platform [11].

The Android and WearOS applications are built using native technologies, frameworks and Kotlin as the main programming language. The software architectural pattern used is Model–View–ViewModel, also known as MVVM (see Figure 6).

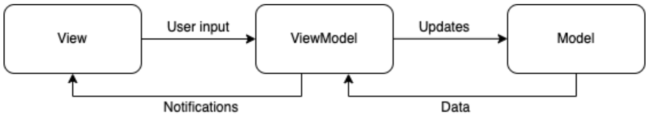

Figure 6. MVVM

In MVVM, each component has its own separate responsibilities:

The Model is used to encapsulate the application data and provide methods to access, edit or delete it. This model describes directly the base entities used through the application, together with the data access objects which can make operations on these entities. The Model does not have any knowledge of the View or the ViewModel, being completely independent.

The View is responsible with the layouts that are used to display the User-Interface. The framework used to draw the UI elements on the screen is Jetpack Compose, the latest method of describing UI components promoted by Google. The view is used to provide user input to the ViewModel, while receiving data from the ViewModel and presenting it to the user.

The ViewModel is the intermediary layer between the two presented above. It manipulates the data provided by the Model and forwards it to the View for display. It also handles the user input given by the View and updates the Model accordingly to the user needs.

Data transfer

There are three main aspects regarding the flow of data. The first problem is exporting the database containing all the recorded sensors values from the watch and use it for training the neural network. The second relates to importing the trained neural network into the watch and building a compatible model file for this. The third aspect focuses on the data transfer between the watch and the phone, in order to properly communicate the detected gestures.

As in this stage we have focused more on the second and third points, we have chosen to manually transfer the data from the watch to the environment for training the neural network. The data has been exported from the device in CSV files, using the database inspector from Android Studio.

For integrating the trained neural network model inside the application, we have converted the built Keras model into a TensorFlow Lite model, because this format is compatible with Kotlin mobile applications. TesorFlow Lite is the lightweight version of the TensorFlow framework specifically developed for devices with limited resources, like mobile phones. For the conversion we have followed the online documentation describing how can a Keras model be transformed into a TF Lite model [12]. The author of [13] also presented a method of building a neural network and the steps required in integrating this model in an Android application. After successfully converting the model, the **tflite** file has been imported into the android project, taking inspiration from the method presented in [14]. The file was added to the assets folder of the application and could be used with the help of the TensorFlow Lite framework.

For establishing the Bluetooth communication channel, I used the Message Client API [15] provided by Android Wear. This works by sending one-way messages between devices. Each connected device in the Bluetooth network is referred as a node, the API sending messages to those nodes.

For finding where to send the message, each node in the network must advertise itself. This is known as capability advertising. The nodes capabilities are represented by an array of strings in the application manifest. For sending a message, an application must find the correct node with his corresponding capability. After identifying the required node, the current application will be able to send messages. In order to receive messages, the other application must have a listener associated that waits for messages to be detected. When a message is received, the application will be able to further process its information.

TESTING AND VALIDATION

Model Performance on Training Data

The training was done in 10 epochs, the following graph describing its accuracy:

As we can see, the results are good, the model achieving an accuracy of 100%. This accuracy is achieved both on the training data and the validation data. As expected, the validation accuracy follows the training accuracy, meaning the model didn't suffer of overfitting. However, we need to keep in mind that this accuracy does not tell us all about the model. This is calculated with the highest value of probability, meaning that if we would have, for example, the accuracy [0.25, 0.35, 0.40] and the correct label is the third, then this result will be taken as a good result, even though the model is only 40% sure that the gesture is the third one.

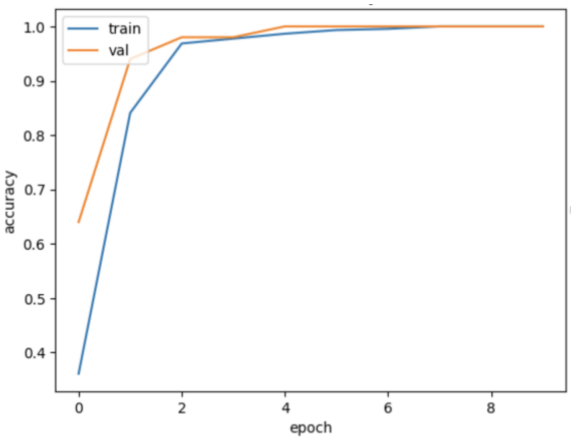

Figure 7. Model Accuracy

Another good metric that measures this error is the loss graph:

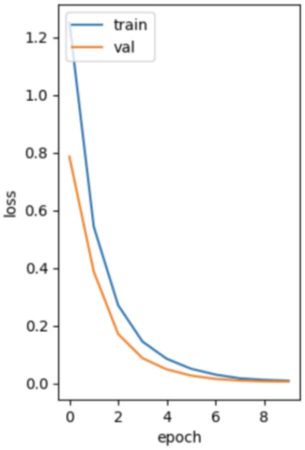

Figure 8. Model Loss

This loss is calculated with the negative logarithm of the predicted value, penalizing the instances with low probability. As we can see, the model loss decreases with the epoch, and almost reaches 0 at the end of the 10th epoch.

Model Performance in Real World

In the following phase, we have done some tests in real world. For the first test, we have empirically verified the probability threshold where the model begins to have problems in recognizing gestures. For that, 20 gestures of each type have been performed for each of the specific threshold values, and the result recorded. In the following graph the Y axis is the overall accuracy, while on the X axis are the threshold values which have been tested. At each step, any gesture which had a probability value under that given threshold will be label as "Unrecognized Gesture".

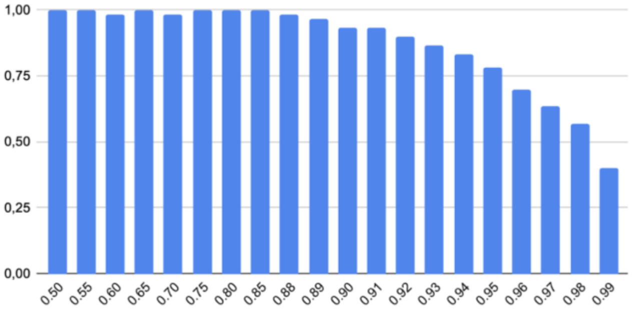

Figure 9. Threshold Accuracy

After this testing, we have decided to set the threshold at 0.80. We consider this to be better than just taking the gesture classified by the neural network with the highest probability, because it can better filter out the random movements that does not actually fit in any of the validated gestures. Our main aim here has been to find a balance point between the risk of not recognizing a valid gesture and the one of considering any user movement as one of the validated gestures.

The next test performed was focused on testing the performance of the neural network on a different user than the one who recorded the training data. In our case, all the training data was taken from one person, so it's interesting in seeing the model performance on someone else. Consequently, we had another user do 100 gestures of each type from the sitting position. In Table 1 you can see the achieved results in terms of percent of the correctly recognized gestures:

Up-Down	Left -Right	Rotation	Overall
74	82	86	80,6

Table 1. Other person accuracy

As expected, the accuracy drops, but the model is still capable of giving the correct result in most cases.

For the last test, we have verified the model performance when moving. As all the training and testing data has been recorded while sitting, we wanted to see how the application behaves when the sensors are influenced by everyday movement. For this test, the same user who recorded the training data has performed 100 gestures of each type while walking. This were the results:

Up-Down	Left -Right	Rotation	Overall
74	89	88	83,6

Table 2. Accuracy while walking

In conclusion, the custom trained model used with a probability threshold of 80% for gesture recognition confidence, manages to correctly identify the gestures made by a new user with an accuracy of 80,6%, and to recognize the gestures while walking with an accuracy of 83,6%.

It's also worth mentioning that the output is given almost instantly, and in less than half a second the message is sent to the phone, which means that the application is overall responsive enough to not produce any kind of confusion.

CONCLUSION

The idea of detecting and interpreting wrist gestures is not new, but there are only a few papers that research this kind of interaction, and we think that there are a lot of opportunities in this field that wait to be discovered. Having an application that can be easily integrated and used on common affordable devices will accelerate the research done on this subject, without needing any other special equipment or knowledge.

The work presented in this article demonstrates an approach on using neural networks together with WearOS powered devices, to recognize specific hand-made gestures. We have highlighted all the necessary steps required to create such a functionality from scratch: (1) record data for training, (2) transfer the data to the training environment, (3) integrate the neural network into the watch application, and (4) use the output of the network in a real-life scenario. In our view, the proposed solution can be easily extended with new gestures and adapted to many other different use scenarios.

According to the limited testing that we have performed so far, proposed approach proves to be resilient to context changes. The 80% accuracy when a new person interacts with the system or when the user is performing different simultaneous movements is not an overwhelming result but gives us a good indication on the improvement capabilities. Having a larger and more variate dataset for training will, most likely, improve these results significantly.

Next, we will focus on recording data from a larger number of users, train the neural network and re-evaluate the performance while recording not only the percentage of correctly identified gestures but also the false positives generated by usual user activity.

ACKNOWLEDGMENT

This research has been partially supported by the CLOUDUT Project, cofunded by the European Fund of Regional Development through the Competitiveness Operational Programme 2014-2020, contract no. 235/2020.

REFERENCES

1. Xie Baao; Li Baihua; Harland Andy, „Movement and gesture recognition using deep learning and wearable-sensor technology," în *International Conference on Artificial Intelligence and Pattern Recognition 2018*
2. Carfì Alessandro; Motolese Carola; Bruno Barbara; Mastrogiovanni Fulvio, Online Human Gesture Recognition using Recurrent Neural Networks and Wearable Sensors, 2018.
3. R. Yanchyshyn, Motion Gesture Detection Using Tensorflow on Android, https://lembergsolutions.com/blog/motion-gesture-detection-using-tensorflow-android.
4. Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, Lisha Hu, „Deep Learning for Sensor-based Activity Recognition: A Survey," 2017.
5. Fernando Moya Rueda, René Grzeszick, Gernot A. Fink, Sascha Feldhorst, Michael Ten Hompel, „Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors," *Sensor-Based Activity Recognition and Interaction*, 2018.
6. J. Loy, Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects, Packt Publishing Ltd, 2019.
7. Tamouridou, A.A.; Pantazi, X.E.; Alexandridis, T.; Lagopodi, A.; Kontouris, G.; Moshou, D. Spectral Identification of Disease in Weeds Using Multilayer Perceptron with Automatic Relevance Determination. Sensors 2018, 18, 2770.
8. P. Baheti, Activation Functions in Neural Networks, https://www.v7labs.com/blog/neural-networks-activation-functions#:~:text=An%20Activation%20Function%20decides%20whether,prediction%20using%20simpler%20mathematical%20operations.
9. Vasilev Ivan; Slater Daniel; Spacagna Gianmario; Roelants Peter; Zocca Valentino, Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow, Packt Publishing Ltd, 2019.
10. Google, Introduction to activities, https://developer.android.com/guide/components/activities/intro-activities.
11. Renju Liu, Renju Liu, „Understanding the Characteristics of Android Wear OS," Purdue ECE, 2016.

12. TensorFLow, TFLiteConverter, https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter.

13. A. Shukla, Train ML Model and Build Android Application Using TensorFlow Lite & Keras, https://medium.com/geekculture/train-ml-model-and-build-android-application-using-tensorflow-lite-keras-6bf23d07309a.

14. V. Valouch, From Keras to Android with TensorFlow Lite, https://medium.com/@vvalouch/from-keras-to-android-with-tensorflow-lite-7581368aa23e

15. Google, Message Client, https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageClient.