REAL-TIME HUMAN ACTIVITY RECOGNITION AND INDOOR POSITIONING SYSTEM FOR THE ELDERLY

ECEN495

Submitted by

Amr Alfayoumy – 19105223

Rawan Youssif – 18101552

Shahd Gamal – 19200136

Nagham Nessim – 18100013

Supervised by

Dr. Tawfik Ismail

Table of Contents

Acknowledgement

The authors would like to extend their heartfelt gratitude to the faculty of the School of Engineering and Applied Sciences at Nile University for their unwavering support and guidance throughout the research process. The department has provided a stimulating and supportive environment that has allowed us to flourish and reach new heights in our academic pursuits. We would like to express special appreciation to Dr. Tawfik Ismail, Director of the Wireless Intelligent Networks Center (WINC) at Nile University, and Dr. Ahmed Madian, Director of the Electronics and Computer Engineering Program at Nile University, for their invaluable contributions to this research. Their expertise and wealth of knowledge in the field have been instrumental in shaping the direction and outcomes of this study. Their insightful guidance, constructive feedback, and encouragement have been a constant source of motivation for us. Their passion for teaching and research is truly inspiring, and we are honored to have had the opportunity to work with them. Additionally, we would like to acknowledge the administrative staff at Nile University who have provided us with the necessary resources and support to carry out this research. Their dedication to ensuring the smooth running of the department has been greatly appreciated and has allowed us to focus on our work. In conclusion, we are grateful to everyone who has played a role in making this research a success and we hope that it will make a meaningful contribution to the field of engineering.

Dedication

This research is dedicated to those who have supported and inspired us throughout our academic and professional journeys. We would like to express our deep gratitude to our mentors, colleagues, friends, and families who have encouraged us and provided us with the resources and motivation to pursue our passions and achieve our goals. Without their support, this research would not have been possible. Their wisdom, guidance, and unwavering belief in us have provided us with the strength and inspiration to persevere and overcome any obstacles that arose during the research process. We would also like to acknowledge and pay tribute to those who have made significant contributions to the field of study that this research is built upon. Their ground-breaking work has paved the way for novel discoveries and advancements, and their legacy continues to inspire us to push the boundaries of knowledge and make a positive impact on the world. In conclusion, this research is a testament to the power of collaboration, perseverance, and the pursuit of knowledge. We dedicate this work to all those who have supported us and inspired us to keep learning and making a difference.

List of Figures

List of Tables

Summary

The global elderly population has increased dramatically as the number of individuals aged 60 and above reached 962 million, more than a two-fold increase from the 382 million recorded in 1980. By 2050, this number is projected to double once more, surpassing 2.1 billion individuals. As people age, their physical activity and capacity to perform daily tasks decrease, impacting both their mental and physical health. Moreover, a large percentage of elderly people living in residential care settings have dementia or other cognitive impairments. Due to the limited number of staff members compared to the number of residents in care facilities, using technology can enhance the care provided. The ability to track elderly patients with cognitive impairment or dementia can prevent wandering and getting lost. Previous research has focused on applying machine learning and deep learning models to recognize the activities of healthy and younger populations, but there has been a lack of attention given to the recognition of activities performed by the elderly. This study aims to provide assistance to elderly people by integrating Human Activity Recognition (HAR) and Indoor Positioning, monitoring patients' activities in different indoor and outdoor environments in real-time, as well as simultaneously locating their positions in indoor environments. We propose a solution that incorporates artificial intelligence, particularly deep learning models, and is based on sensor readings collected from a smartwatch. This method aims to detect five classes of activities: walking, sitting, laying down, going upstairs, and going downstairs. Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Long-Short Term Memory (LSTM), and CNN-LSTM model are implemented and their performances are comprehensively compared. Upon evaluation, the CNN-LSTM model outperformed all the other HAR models, achieving an F1-score of 98.95%. As for the Indoor Positioning System (IPS), it is based on RSSI measurements of a BLE beacon and was implemented using machine learning classifiers including k-Nearest Neighbor (kNN), Support Vector Machine (SVM) with linear, polynomial, and RBF kernels, NuSVC, Random Forest, Decision Tree, Gradient Boosting, Gaussian Naïve Bayes, Linear Discriminant Analysis, and Quadratic Discriminant Analysis. A Voting Classifier was implemented using a majority 'hard' voting of the five best-performing classifiers. Cross-validation using a shuffle-split method (n = 10 times) was employed for dividing the dataset into training and testing subsets with an 80:20 split ratio. The Random Forest classifier achieved the highest mean F1-score of 84.12%, whereas the Voting Classifier achieved the second highest mean F1-score at 83.88%.

Chapter 1: Introduction

1. Introduction

In recent times, the field of Human Activity Recognition (HAR) has become increasingly popular in research. The use of sensors, accelerometers, and advancements in technology such as computer vision, machine learning, artificial intelligence, and IoT have enabled the development of numerous applications that can recognize, detect, and categorize human behavior. These applications are often designed to be low-cost and energy-efficient, utilizing information gathered from various smartphone sensors and wearable devices, such as accelerometer sensors and gyroscope sensors, as well as factors such as time and location. When integrated with other technologies, such as the Internet of Things, it can be applied in various areas such as healthcare, sports, and industry [1].

According to the World Health Organization, a significant portion of the human population, approximately 1.3 billion individuals, experience significant disability [2]. There is a lack of adequate resources to meet the needs of people with disabilities, one of which is the requirement for a constant companion to oversee their activities [3],[4]. To ensure the safety and protection of people with disabilities from potential harm or accidents, 24-hour supervision is necessary [5]. In 2017, there were 962 million individuals aged 60 and above worldwide, more than double the 382 million people in that age range in 1980. It is projected that by 2050, the number of elderly individuals will exceed 2.1 billion, with two-thirds of them residing in developing countries where their population is increasing at a faster rate than in industrialized countries [6]. It is also expected that by 2050, nearly 8 out of 10 of the world's elderly population will be living in underdeveloped countries. The percentage of people aged 60 and over who live independently, either alone or with a spouse, varied greatly among the 143 countries or territories with available data, ranging from 2.3% in Afghanistan to 93.4% in The Netherlands [7]. As people age, their physical activity and ability to perform everyday tasks decrease, which can negatively impact both their physical and mental health. While there have been many studies using machine learning and deep learning methods to recognize human activities, there are very few that focus specifically on the recognition of elderly people's activities.

The importance of being physically active for older individuals in order to decrease their risk of developing comorbidities and to increase their quality of life [8] is well-known. Monitoring Physical Activity (PA) under real-world conditions can assist in determining if the recommended levels of PA are being met and can be used for feedback. Additionally, older individuals tend to increase their activity levels when monitoring PA, likely because monitoring increases motivation to achieve PA goals [9]. An activity classification method with high accuracy is crucial for providing appropriate feedback to older individuals. While many classification algorithms have been proposed for sensor-based activity recognition, most studies have focused on younger adults. Due to potential differences in movement patterns between the two age groups, algorithms trained on young individuals may not be as accurate when applied to data from older individuals [10]. Therefore, there is a need to develop a classifier specifically tailored for older individuals. A small number of studies have

specifically focused on older individuals using inertial sensors [11]–[14]. These studies utilized a combination of accelerometry, gyroscopes, and magnetometers, which were processed using neural networks, support vector machines, decision trees, or random forests to achieve accuracies of 82-93% [11]–[14]. These studies indicate that accurate activity classification in older individuals is possible.

In recent years, there has been an increased interest in using artificial intelligence for human activity recognition (HAR) due to its self-learning nature and robust classification models [15]. Various studies have been conducted using machine learning and deep learning techniques for HAR [16], [17], but only a few have focused on developing a framework specifically for elderly people. Kaixuan Chen et al. [18] provided an overview of the challenges and opportunities in the use of deep learning techniques for HAR. In order to maximize performance, it is important to utilize multimodal features, which can help to distinguish between different types of sensor data [19]. Attention-based mechanisms have been used to identify the most important and distinctive modalities in HAR [20]. Chen et al. [21] used multiple agents to focus on modalities related to sub-motions, which improved performance compared to other methods, but it has not yet been validated on a dataset of the elderly.

Furthermore, as people with dementia require more support for their daily activities, many move to assisted living facilities. "Residential care" is a broad term used to describe these types of environments, which are intended to help older adults maintain their independence. A large percentage of people living in residential care settings in developed countries have dementia or cognitive impairment [22], [23]. Due to the limited number of staff members compared to the number of residents in these facilities, using technology can potentially help to fill in resource gaps and enhance the care provided. Knowing the location of an elderly patient is crucial for a number of reasons. One of the main reasons is safety. Elderly patients are at a higher risk of accidents and injuries, such as falls, and being able to quickly locate and respond to an emergency can greatly improve the chances of a positive outcome. Additionally, for patients with cognitive impairment or dementia, the ability to track their location can prevent wandering and getting lost, which can be a major concern for both patients and their caregivers. The use of real-time location systems (RTLSs) significantly reduces the frequency and duration of wandering incidents in older adults with dementia, and also improves the safety of these individuals.

Real-time locating systems (RTLS), also known as indoor positioning or location systems, are technologies primarily used to track individuals and equipment in indoor environments in real or near-real time. They can be incorporated into nurse call or safety systems and consist of a wearable device that is worn by an individual, a number of receiver devices embedded in the environment, and software to visualize location data on a facility map [24], [25]. The wearable device, such as a tag or bracelet, contains a sensor that communicates with the receiver devices, which are connected to a wireless network, to continuously track people or items in real time. Non-wearable sensor technologies, such as near-field radio-frequency ID (RFID) tags or passive infrared (IR) sensors, can also be deployed as RTLS in clinical settings to detect the movement of individuals passing

through doors or at a room-level scale. For higher accuracy, wearable systems such as Bluetooth or Ultra-wideband (UWB) are used when the intention is to collect within-room movement patterns at a regular sampling rate over time. RTLS provides a vast amount of data on an individual's location over time and can characterize movement through a well-defined target environment. RTLS installations have been studied in a wide variety of healthcare settings, such as monitoring residents/patients and staff movements, as well as assets such as surgical equipment [26]–[29].

Context-aware computing, smart connection with existing networks, and cost-effective low-power wireless technologies are essential components of the Internet of Things (IoT) idea. One of the most recent IoT innovations, Bluetooth Low Energy (BLE), is particularly well-suited for ultra-low power sensors that use small batteries. Bluetooth is a wireless technology standard for transmitting data across short distances. It is also a commonly used technology in wireless personal area networks and is common among mobile devices [30]. Due to its low cost and ease of deployment, Bluetooth has been employed for indoor localization [31], [32]. BLE is an effective replacement for indoor positioning systems (IPS) that provides good accuracy and easy setup [33]. Indoor positioning using BLE technology is becoming increasingly important in the supervision of elderly patients, allowing caregivers to track the location of patients within a facility, such as a hospital or assisted living facility, in real time. This helps to ensure the safety and well-being of elderly patients, as it allows caregivers to quickly respond to any potential issues, including patients falling, wandering, or becoming lost or unconscious. Furthermore, BLE indoor positioning can also be used to monitor the activity levels of elderly patients and track their movements over time, providing valuable information for healthcare professionals to assess their physical and mental well-being. Overall, indoor positioning using BLE technology can be a powerful tool for improving the care and supervision of elderly patients, providing caregivers with valuable information to help keep patients safe and healthy.

This study aims to provide assistance to elderly people by monitoring their activities in different indoor and outdoor environments in real-time using data collected from a smartwatch, as well as simultaneously locating their positions in indoor environments in real-time using BLE beacons. Smartwatches have been commonly used to monitor human activities, and the data used in this study include routine activities such as sitting, walking, going up and down stairs, and lying down. This study aims to develop a human-activity-monitoring and localization application for elderly people, and for this purpose, it is important to use human activity recognition (HAR) and Indoor Positioning System (IPS) algorithms that have high accuracy, precision, and recall. Several deep learning approaches for activity recognition are implemented and evaluated. This study aims to determine the most effective approach for developing a HAR and IPS framework for elderly people. For HAR, we compare the performance of deep learning approaches such as the Artificial Neural Network (ANN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and CNN-LSTM. The Long Short-Term Memory Network, which is a variation of recurrent neural network (RNN), is particularly well suited for handling temporal sequences. As for IPS, several supervised machine-learning algorithms for indoor localization are implemented and evaluated. The performance of these algorithms is evaluated using

several metrics, including confusion matrices, precision, recall, accuracy, F1-score, and computation speed. A comprehensive comparison of the algorithms is presented to give a better understanding of their performance. Finally, a web application is developed to exploit the models' capabilities in recognizing human activities and detecting the location of an individual in real time.

2. Objectives

A Human Activity Recognition (HAR) system will be integrated with an Indoor Positioning System (IPS) to monitor elderly or ill persons and take action if abnormal behavior is detected, or harmful occurrence takes place. Thus, the objectives for this project are as follows:

- Creating a dataset of sensor data of 5 classes: sitting, walking, lying down, going upstairs, and going downstairs.
- Detecting the person's activities in real-time, such as whether they are sitting, walking, lying down, going upstairs, or going downstairs, using a wearable device connected to a database.
- Comparing various deep learning techniques, including Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and CNN-LSTM.
- Creating a BLE-based Indoor Positioning System (IPS) to locate the monitored elderly person inside a building in real time.
- Comparing several supervised machine learning algorithms and their performance in detecting the location of the elderly individual in an indoor environment.
- Integrating the HAR and IPS systems into one framework.

3. Project Impacts

The following is a list of the proposed project's impacts:

- Introducing a novel framework integrating Human Activity Recognition and Indoor Positioning
- Novel HAR dataset of 19 sensor features covering 5 activities: sitting, walking, lying down, going upstairs, and going downstairs
- Real-time activity recognition and indoor positioning via a secure Web App
- Comprehensive analysis and performance comparison of DL models: ANN, CNN, LTSM, and CNN-LSTM for HAR
- Comprehensive comparison of the performance of 11 machine learning classifiers for indoor positioning using BLE beacon's RSSI values

Chapter 2: Literature Survey

1. Human Activity Recognition

The study of human activity recognition (HAR) makes use of two types of sensors: ambient sensors and wearable sensors. Ambient sensors are fixed in a specific location, and wearable sensors are worn by the user. One common ambient sensor used for HAR is a video camera, which uses computer vision techniques to identify human activity from videos or images [34]. With advancements in sensing technology, a depth camera can now capture 3D data in real-time, leading to further research in activity recognition from 3D data [35]. In addition, recent developments in wireless technology have led to the use of wireless signals for activity recognition, as the presence and movement of humans can affect the channel state information (CSI) of wireless signals [36]. This has sparked many recent research efforts in CSI-based activity recognition [36], [37]. These ambient sensor-based methods do not require a device to be worn by the user, making them ideal for security purposes and interactive applications. However, the drawback of these methods is that they require predetermined infrastructures such as video and Wi-Fi devices, which can be difficult to attach to individuals during daily activities.

The use of wearable sensors for human activity recognition became necessary due to the limitations of ambient sensors. The first activity recognition using wearable sensors, which utilized an accelerometer for posture and motion detection, was proposed in the 1990s [38]. The accelerometer is the most commonly used sensor for activity recognition, as noted in multiple studies [39], [40]. A review of techniques for physical human activity recognition utilizing wearable inertial sensor data is presented by Attal et al. [41]. Lara and Labrador also conducted a survey of the current state of the art in activity recognition using wearable sensors [42]. With advancements in sensor technology, smartphones now have various sensors including accelerometers, gyroscopes, magnetometers, and barometers which can be combined for activity recognition. Shoaib et al. proposed an activity recognition method using smartphone motion sensors [43]. In comparison to ambient sensor-based methods, wearable sensors are not limited by coverage and can capture continuous activities.

Traditional machine learning methods for activity recognition usually involve feature extraction and classification. The performance of these methods heavily relies on the extracted features, which requires significant human effort and is a major challenge. Features used for activity recognition include time domain features (e.g. mean, standard deviation, variance, interquartile range), frequency domain features (e.g. Fourier Transform, Discrete Cosine Transform), and others (Principal Component Analysis, Linear Discriminant Analysis, Autoregressive Model) [42]. For activity classification, there are numerous classic methods such as decision trees [44], Bayesian [45], neural networks [46], k-nearest neighbors [47], regression methods [48], support vector machines [49], and Markov models [50].

A concept for a human activity recognition system built on the Android platform was proposed by Usharani J et al. [51]. They developed a program that facilitated online training and classification and used accelerometer

data for classification. They improved the performance, accuracy, and execution time of the k-NN classifier on the Android platform by using the clustered k-NN technique. They came to the conclusion that the device types and capabilities had an impact on the classification times as well. A real-time HAR framework for live prediction of human physical motions based on smartphone inertial sensors was also proposed by Meysam, Vakili, et al. [52]. Six cumulative learning algorithms were utilized to evaluate the system's performance on a total of 20 distinct tasks, and all of them were contrasted with cutting-edge HAR techniques like Decision Trees (DTs), AdaBoost, etc. The best accuracy was provided by incremental k-NN and Naïve Bayes, both of which had a 95% accuracy. Using a machine learning technique, Jirapond Muangprathub et al. [53] devised a new elderly person tracking system. The k-NN model with a k value of 5 was utilized in this study, and it was able to detect older people's real-time activity with the best accuracy of 96.4%. Additionally, they developed a system that allows an old person to view the information in a geographical manner and, in an emergency, utilize a messaging device to seek assistance.

In recent years, deep learning has gained significant attention and has been implemented in various fields, including human activity recognition. Researchers have applied convolutional neural networks (convnets) to human activity recognition using sensor signals, but these methods have limitations. For example, Zheng et al. used a one-layer convolutional network which resulted in low accuracy [54]. Ronao and Cho proposed a deep learning method that classified six types of activities, but it only used accelerometers and magnetometers which made it difficult to differentiate vertical activities such as going upstairs and downstairs [55]. Gu et al. proposed a smartphone-based activity recognition method that utilized four types of sensor data, including an accelerometer, gyroscope, magnetometer, and barometer, but it only used a two-layer structure that did not fully utilize the non-linear capabilities of deep learning, resulting in performance defects [56]. Ravi et al. proposed a deep learning method for human activity recognition using low-power devices, but it relied on a manual feature selection strategy to extract features that required significant human intervention [57]. A CNN was suggested by Baoding Zhou et al. [58] for the detection of indoor human activities. Based on data acquired by cellphones' accelerometers, magnetometers, gyroscopes, and barometers, a total of nine separate activities were identified. The proposed approach was successful in achieving a remarkable accuracy of 98%. Combinations of CNN and LSTM were employed by Yashi Nan et al. [59] to identify the activities of elderly people. A multi-channel CNN-LSTM model was able to attain the greatest accuracy of 81.1% out of all the combinations.

In [58], a deep learning-based approach was developed to recognize indoor activities by utilizing data from multiple sensors on a smartphone. A new type of CNN was designed specifically to analyze one-dimensional sensor data and automatically learn relevant features. Results of experiments indicate that the proposed method has an accuracy rate of 98.33% for identifying nine different activities, including standing still, walking, going upstairs, using an elevator, using an escalator, going downstairs, turning, and more. The significance of this research is the development of an efficient deep-learning framework for recognizing indoor activities, which can be applied in indoor localization. Additionally, a large database of pedestrian activity data, consisting of

over 6 GB of information from accelerometers, magnetometers, gyroscopes, and barometers collected from various types of smartphones, was created.

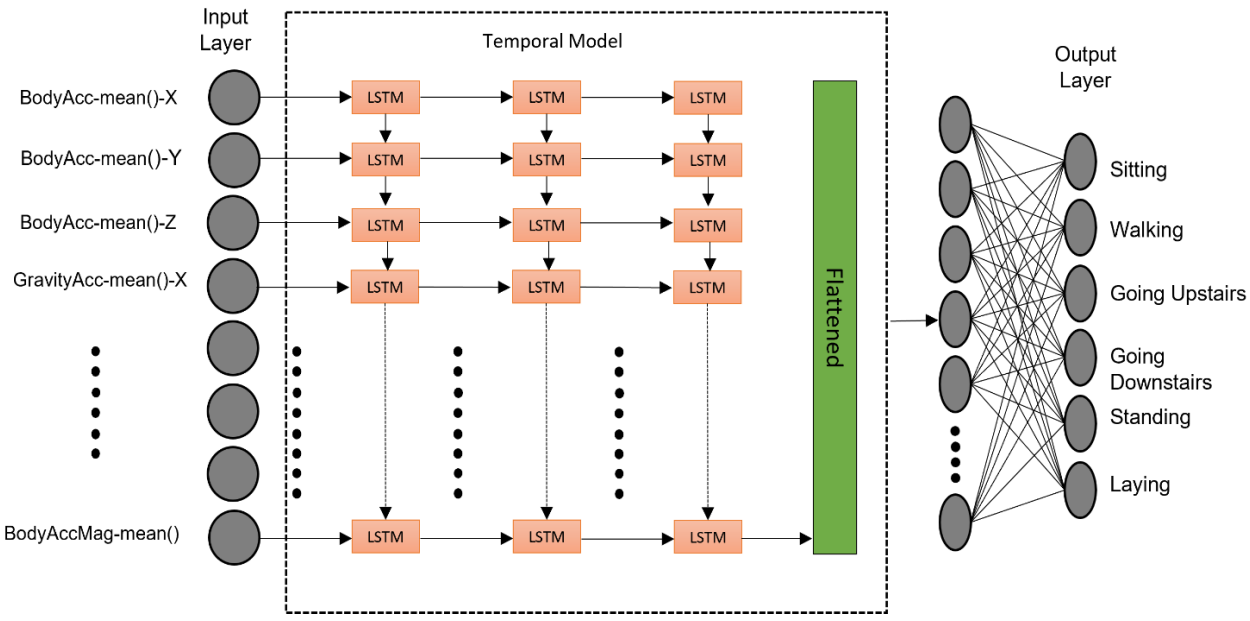

Figure 1: Architecture of Hayat et al.'s [60] LSTM model for human activity recognition

Hayat et al.'s study [60] used a dataset from the University of California, Irvine (UCI) for its research. The data was collected from a group of participants aged between 19 and 48 who all had smartphones. The accelerometer and gyroscope of the phones were used to capture the acceleration and angular velocity of the data at a sample rate of 50Hz. To process the signals, a median filter, and a notch filter with a drop frequency of 20Hz was applied. The dataset had 561 features and 10,299 samples. The data was divided into two sections, with 80% being used for training and 20% being used for testing. This means that there were 8240 training data and 2059 testing data. The dataset includes six different categories of user activity: sitting, walking, going upstairs, going downstairs, standing, and lying. Additionally, the system was cross-validated on another dataset called "Activity recognition with healthy older people using a battery-less wearable sensor dataset," which was collected from 14 participants aged between 66 and 86 in two clinical rooms. This dataset had 75,128 samples and was categorized into four different user activities: sitting on the bed (16,406 instances), sitting on the chair (4911 instances), lying on the bed (51,520 instances), and ambulating (walking or standing within the room) (2291 instances). Limited pre-processing was done to enhance the models' generalization. The only pre-processing technique used was Principal Component Analysis (PCA) for dimensionality reduction for optimal processing. PCA is used to compress the dataset into a lower-dimensional feature space while preserving important information.

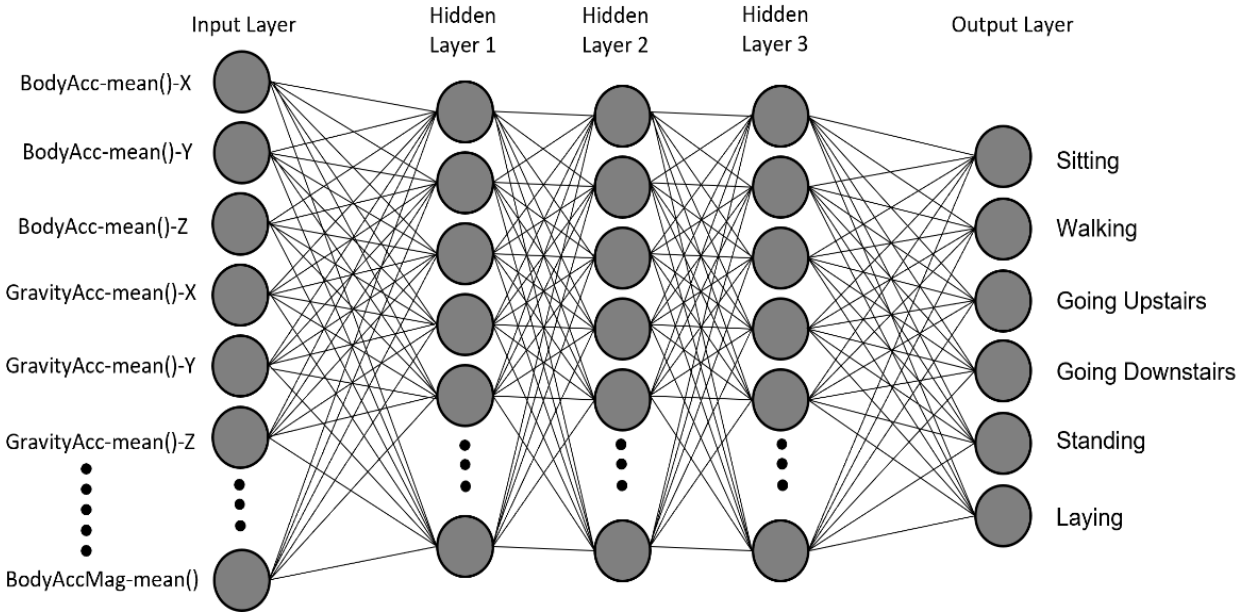

Figure 2: Architecture of Hayat et al.'s [60] artificial feed-forward neural network with 3 hidden layers

In Hayat et al.'s work [60], the experimental results of various conventional machine learning and deep learning methods for activity detection in elderly individuals are presented and discussed. To ensure stability, each experiment was repeated 10 times due to the random assignment of initial weights and parameters. The average results are presented in the study for consistency. The study found that the Long Short-Term Memory (LSTM) approach, shown in Figure 1, was the most successful among all the methods tested, with the Artificial Neural Network (ANN), shown in Figure 2, having the best accuracy in classifying "sitting" for both 2-fold and 10-fold cross-validation. Deep learning methods, such as ANN and LSTM, performed better than conventional machine learning methods in terms of accuracy. LSTM had the highest overall accuracy at 95.05%. Additionally, it was observed that the model had difficulty separating "going upstairs" and "going downstairs" activities, as they were often misclassified. Results showed that the LSTM classifier using 10-fold cross-validation had the best precision, recall, accuracy, and F1-score values at 92.87%, 85.32%, 95.05%, and 88.94%, respectively, however, the Support Vector Machine (SVM) had the best processing time at 0.08 and 0.42 min in both 2-fold and 10-fold cross-validation. The study also tested the proposed method on another dataset and found that it performed well even on an imbalanced dataset. The study concludes that deep learning methods are more suitable for human activity recognition of elderly people, but if processing time is a concern, machine learning methods, particularly SVM, may be a better option.

In the field of activity recognition, Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) are frequently used among deep learning techniques [14], [61], [62]. Studies have shown that CNNs can achieve high accuracy levels, such as in [63] where a 1D CNN model was proposed for simple physical activity classification using triaxial acceleration data with six activity classes and achieved 93% accuracy. In [61], a multichannel CNN model was created to process data from multiple inertial measurement units in parallel channels. Additionally, combining LSTM with a CNN, as in [64], can improve results by six percent

compared to a CNN alone. Various neural network architectures have been developed and have achieved state-of-the-art results, however, there is a lack of direct comparison of their performance for activity recognition. Additionally, it is worth investigating how these deep learning algorithms perform with older individuals.

In [59], the researchers evaluated deep learning techniques, including CNN and LSTM, by using only accelerometry data from 53 older individuals (average age of 83.8 ± 3.8 years, 38 of them were male) performing various activities under controlled conditions. The activities were grouped into categories such as lying, sitting, standing, transitioning, walking, going upstairs, and going downstairs. The researchers tested four different models, including a 1D CNN, a multichannel CNN, a CNN-LSTM, and a multichannel CNN-LSTM. They evaluated the models based on their accuracy and computational efficiency. The results showed that the multichannel CNN-LSTM model had the highest accuracy of 81.1%, and it was able to classify the activities with an acceptable level of complexity. Specifically, the accuracy of the model was 67.0% for lying, 70.7% for sitting, 88.4% for standing, 78.2% for transitions, 88.7% for walking, 65.7% for walking downstairs, and 68.7% for walking upstairs. The study concluded that the multichannel CNN-LSTM model can be used for recognizing the activities of older people using smartphone data.

2. Indoor Positioning System

Propagation-based and fingerprint-based methods for indoor situating have been primarily used in Wi-Fi-based area frameworks [65]. Triangulation or trilateration, shown in Figure 3, is used in propagation-based techniques to identify transmitters in known directions. In [66], Jayakody et al. use an indoor propagation model of sorts to estimate the distance between the smartphone and visible transmitters, and they then use these distances to estimate the location of the smartphone. Due to the dynamic nature of signals and signal attenuation, they are extremely inaccuracy prone while operating in an interior setting. However, area fingerprint coordinating approaches assess the place coherently, the framework gathers large amounts of data, and a series of explorations and investigations are carried out to offer a reference to the location. Both of these tools proactively take use of the proliferation of modern, intelligent smartphones and their rapidly increasing hardware capabilities [67].

Indoor Positioning Systems can also be based on RSSI and utilizing BLE technology [68]. Compared to alternative methods like Wi-Fi, GSM, or GPS, the system is more energy efficient. Because these BLE tags are simple to set up and may operate for several months on a single small battery, deployment is rapid. A BLE device can perform in several modes depending on the functionality that is needed. The neighbor-finding process has three modes: advertising, scanning, and initiating [68]. It uses 40 channels of 2 MHz each to operate in the 2402-2480 MHz frequency range. One of the main explanations for why BLE uses such low energy is the central-peripheral interaction. A peripheral device (such as a BLE tag) may simply broadcast its information while nearby central devices (such as smartphones) can collect it. BLE often functions in a central and periphery role using a client/server approach. One or more servers can be accessed by a client after connecting [69]. Either the master or the slave roles are used by BLE. A slave can only be linked to one master,

but a master can handle several simultaneous connections with a variety of slave devices. In order to find slaves, the master monitors certain ad channels [70]. Data is sent as connection events, where the slave and master wake up and exchange frames to keep the system synchronized. Both gadgets are in sleep mode for the remainder of the time. Only when requested, the BLE device communicates [71].

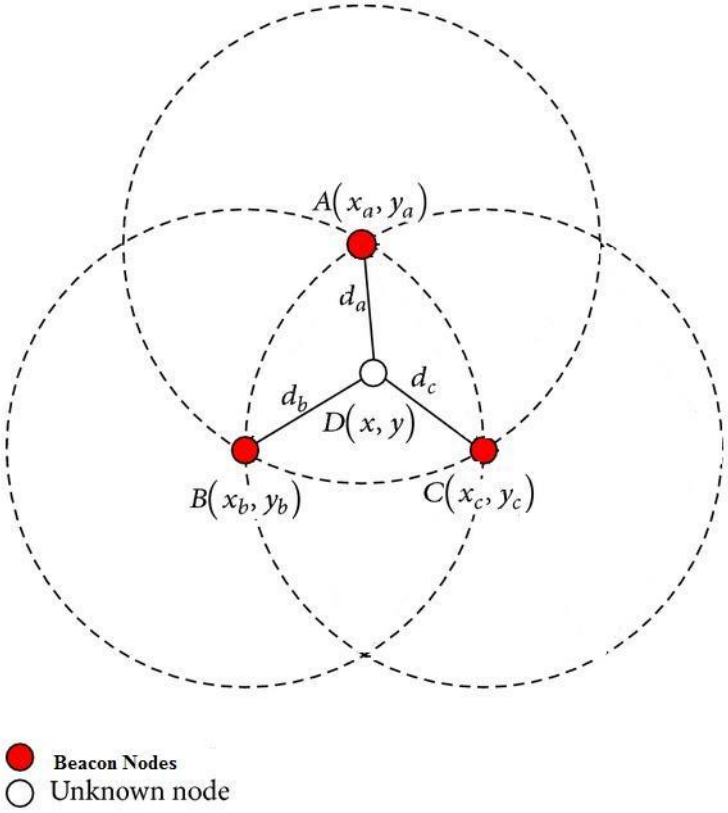

Figure 3: Trilateration algorithm [66]

Cabarkapa et al. [72] give a comparative analysis of modern BLE indoor positioning solutions taking into mind the categorization, comparison, and variety concerns of characteristics that are necessary for building new indoor positioning techniques. Accuracy, RF signal coverage, availability, power consumption, and lowest costs for interior installations are requirements for various application scenarios. It is necessary to construct and maintain an RSSI-based BLE infrastructure with a number of optimizations to increase positioning accuracy if the IPS demands greater precision (1.5m) for 80% of the time [72]. According to Cabarkapa et al. [72], BLE IPS systems mostly employ three techniques: Trilateration, BLE Fingerprinting, and BLE Proximity Detection.

Trilateration is a location estimation technique with a trigonometric foundation. This method makes use of distance measurements to at least three well-known reference sites (beacon nodes or APS) [73]. The distance is determined by the RSSI that the RPs received. For the purpose of determining their position, RSSI data are converted into distances using the RF propagation model [74]. Due to the indoor RF propagation channel's considerable variability, this technique does not provide the most precise estimate, but it does not need any

configuration or calibration procedures. The trilateration equations can be solved using a variety of mathematical techniques. Three techniques are covered in [75]: centroid placement, three-border positioning, and least square estimation (LSE). BLE Fingerprinting (scene analysis) is the process of fingerprinting is an RSSI-based scene analysis in which characteristics (or fingerprints) of nearby reference points are first gathered at each site in the areas of interest, followed by the creation of a fingerprint database[76]. During the training phase, RSSI fingerprints are collected at various reference points and stored in a database. This process is repeated for multiple locations and the resulting set of fingerprints is referred to as a radio map [76]. In the determination phase, a mobile user takes measurements of RSSI values at an unknown location, then compares these measurements to the fingerprints in the training database using a positioning algorithm to determine their likely location. BLE Proximity Detection is using proximity-based IPS based on its proximity to other BLE devices or beacons. Information about the symbolic relative position is provided by proximity algorithms. The most straightforward proximity-based strategy is to choose the node's location with the strongest signal [77]. In order to construct a relative ordering of nodes based on their proximity to the target device, information regarding RF signal strength is needed. Push notifications are sent to the user's smartphone or tablet when the BLE beacon is in close proximity to the user [78]. This approach does not reveal the device's precise location. This technique necessitates a widespread deployment of BLE beacons in order to attain the best accuracy. The implementation of proximity approaches is very straightforward and does not need calibration [79].

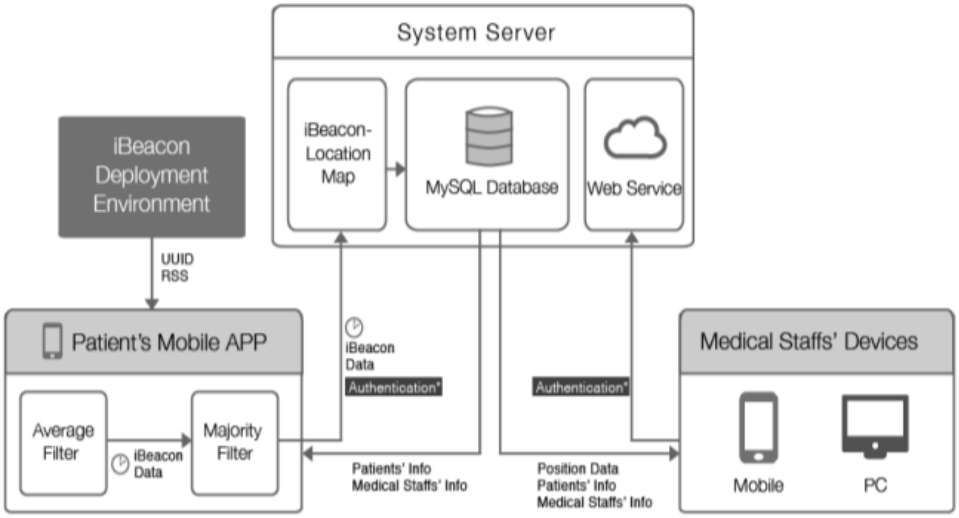

Figure 4: Lin et al. [69] system architecture

Lin et al. [69] implement a mobile-based indoor positioning system using mobile applications (APP) with the iBeacon solution based on Bluetooth Low Energy (BLE) technology while using the Received Signal Strength (RSS) based localization method to estimate the locations of the patients. The four main parts of the system architecture, shown in Figure 4, are the iBeacon deployment, patient mobile applications, system server side, and monitoring side. These four elements combine medical information about patients, information about their locations, and data about the medical staff. The system utilizes a mobile application for patients, which can be downloaded after registration in the emergency room. The app allows patients to access their medical records

and information about their doctors and nurses from the system server, which is displayed on the patient's mobile phone. Additionally, the app automatically detects and selects the nearest iBeacon based on Bluetooth signals received in the background. The information about the selected iBeacon is then uploaded to the system server, and all data is transmitted securely through the HTTPS protocol. The server-side application is responsible for managing data storage, retrieval, and mapping. It serves as a connection between the patient and monitoring sides and maps the estimated nearest beacons sent from the patient side to the corresponding locations. The process of identifying locations is carried out on the server side. The server stores a mapping table of beacons and locations for the purpose of determining the actual locations. The final component is the monitoring side, which allows medical staff to view the location information of patients through a web browser or mobile device. The location classification accuracy of this positioning algorithm was 97.22% [69]. The estimated error for predictions is within 5 meters of the neighboring area, which was considered acceptable for tracking patient locations.

Chapter 3: Project Approach and Methodology

1. Statement of Project

We employ machine learning and deep learning to establish a framework for Human Activity Recognition (HAR) and Indoor Positioning System (IPS) for elderly people. First, to build the HAR part of the framework (Figure 5), sensor data of everyday activities like sitting, walking, lying down, going upstairs, and going downstairs will be collected into a dataset and used to train Deep Learning (DL) models. Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Long Short-Term Memory Networks (LSTM) is the Deep Learning algorithms that will be used to recognize human activities in this project. The wearable device sensors will be used to capture and store different measurements such as acceleration, angular velocity, pitch, roll, yaw, etc. The features will be obtained from the dataset, and the samples will be analyzed. Next, the distribution of the dataset will be analyzed and visualized. The data will then be split into two sub-datasets, with 80% as training data and 20% as testing data.

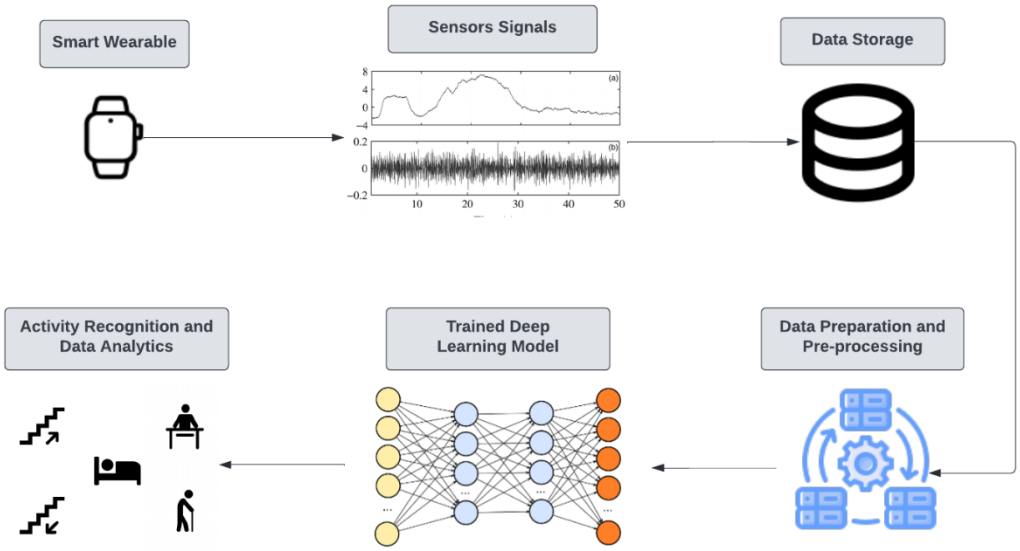

Figure 5: HAR overview

In addition, an Indoor Positioning System (IPS) part of the framework (Figure 6) will be designed to find the position of the monitored elderly person inside a building using Bluetooth Low Energy (BLE) beacons, ESP32 detectors, and Machine Learning models. The BLE signals emitted from the device worn by the monitored person are detected by the ESP32 BLE receivers located inside the building, which measure the Received Signal Strength Indicator (RSSI) of the BLE beacon and send the data to the server. Finally, a Python script will be fetching the most recent measurements (RSSI values) from the ESP32 detectors and then predict the location of the person using trained machine-learning models.

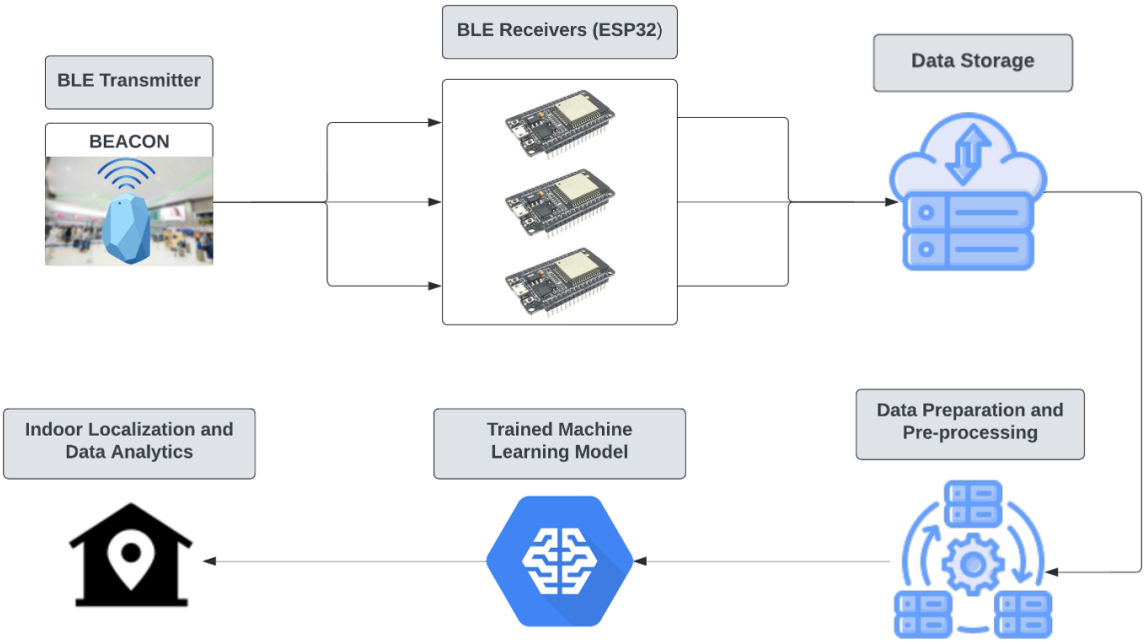

Figure 6: IPS overview

The training environment used in this work is the Google Compute Engine backend (GPU) available through Google Colab. The hardware specifications of the training environment are shown in Table 1.

Table 1: Training environment

Parameter	Specification
CPU Model Name	Intel(R) Xeon(R)
CPU Freq.	2.30GHz
No. CPU Cores	2
CPU Family	Haswell
Available RAM	12GB
Disk Space	25GB
GPU	Nvidia K80
GPU Memory	12GB
GPU Memory Clock	0.82GHz
Performance	4.1 TFLOPS
Support Mixed Precision	No
GPU Release Year	2014

2. Detailed Project Approach and Methodology

A. Human Activity Recognition

The critical building blocks for human activity recognition are shown in the figure below. Raw data from the wearable device will be gathered, and different cleaning and preprocessing techniques will be applied to the dataset. Following that, features from the dataset will be extracted, and the model will be trained using those features. Human activity will be recognized in the final step. The building blocks of the human activity recognition process are illustrated in Figure 7.

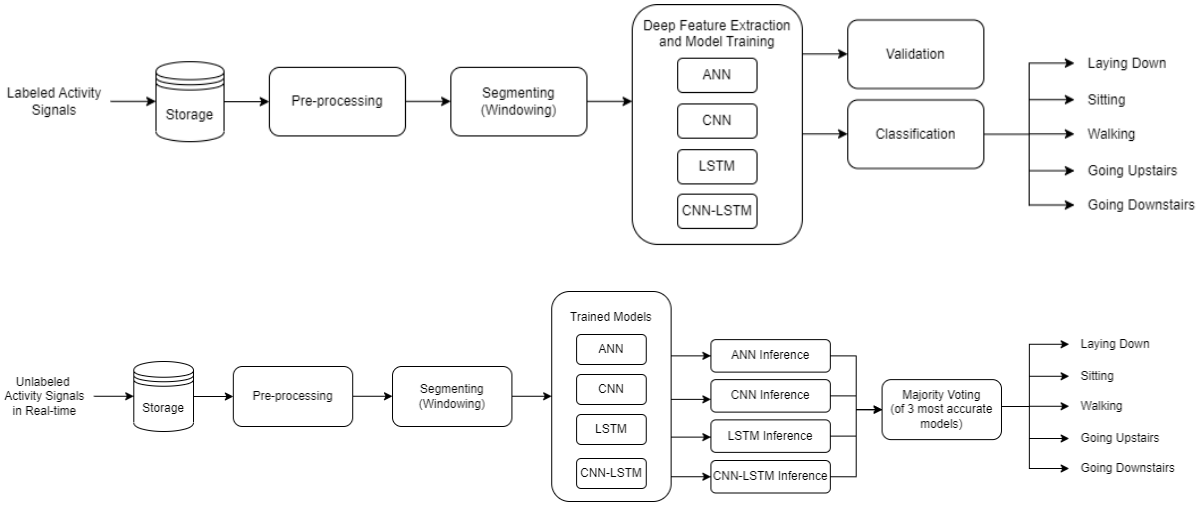

Figure 7: HAR training phase (top) and testing phase (bottom)

i. Hardware Components

a) Wearable Device

The wearable device chosen in this project is the Apple Watch Series 4 smartwatch. The Apple Watch Series 4 smartwatch is a smartwatch developed by Apple Inc. It was announced on September 12, 2018, and was released on September 21, 2018. It features a larger display, an electrical heart sensor, and improved health-tracking capabilities. It runs on watchOS 9. The Apple Watch Series 4 has a Retina display with Force Touch technology, which allows the watch to distinguish between a tap and a press. The watch also has several features for communication, including the ability to make and receive phone calls, send and receive text messages, and send and receive digital touch [80].

The Apple Watch Series 4 (Figure 8) uses several communication protocols to facilitate communication with other devices. The watch uses Bluetooth to connect to other devices, such as the iPhone, for communication and data transfer. This allows the watch to receive notifications, make phone calls, and send messages. The watch also uses Wi-Fi to connect to the internet, allowing it to access online services and communicate with other devices. The watch has an integrated GPS sensor, which allows it to track location and provide location-based services. Some models of the watch also have cellular connectivity, which allows them to make and

receive calls and texts, even when the iPhone is not nearby. All these protocols work together to allow the Apple Watch Series 4 to provide a wide range of features and services to the user.

Figure 8: Apple Watch Series 4

The following are the hardware technical specifications of the Apple Watch Series 4 smartwatch [80]:

- Chip: S4 SiP with 64-bit dual-core processor; W3 wireless chip
- Connectivity: LTE and UMTS2, GPS, Wi-Fi 802.11b/g/n 2.4GHz, and Bluetooth 5.0
- Memory: 16 GB storage memory, 1 GB RAM
- Battery: Built-in rechargeable lithium-ion 292 mAh (1.12 Wh) battery (Up to 18 hours)
- Operating System: watchOS 9.1

ii. Data Collection

The sensor data is collected from the smartwatch using the SensorLog app. SensorLog is a tool that allows developers to access sensor data from iPhone, iPad, and Apple Watch devices. This data can be saved as a file in either CSV or JSON format, or it can be sent as an HTTP request. The purpose of collecting and reading sensor data through SensorLog is to make it visible and accessible to the user for their own usage. The provider of SensorLog does not use the data collected through the tool for any other purposes than providing it to the user. Any data that is recorded by SensorLog on the device will be stored until the user chooses to delete it, either through SensorLog's own log file deletion function or by uninstalling the app from their device. Data that is sent via HTTP to a server specified by the user is under the control and responsibility of the user [81].

The first logging option allows for simultaneous logging of all selectable sensor data in the background with a maximum frequency of 50Hz. Additionally, individual sensors can be logged at a higher frequency of up to 100Hz. This option also supports both streaming and HTTP requests, but only in client mode. The second logging option is for logging duration greater than 1 hour, in which case the background logging only supports accelerometer data with a maximum frequency of 50Hz. However, in the foreground, all sensor data can be logged with a frequency of up to 100Hz. Streaming and HTTP requests are only available when the app is in the foreground. Data can also be transmitted via HTTP(S) GET/POST request to a REST API in JSON format (POST) or form-URL encoded (POST and GET) with a maximum frequency of 10Hz [81]. The app was

configured to transmit sensor data via HTTPS request to a REST API in JSON format (POST) at the maximum frequency of 10Hz to the following Firebase Realtime Database REST endpoint URL: https://sensorlog-6c0d5-default-rtdb.europe-west1.firebasedatabase.app/readings.json

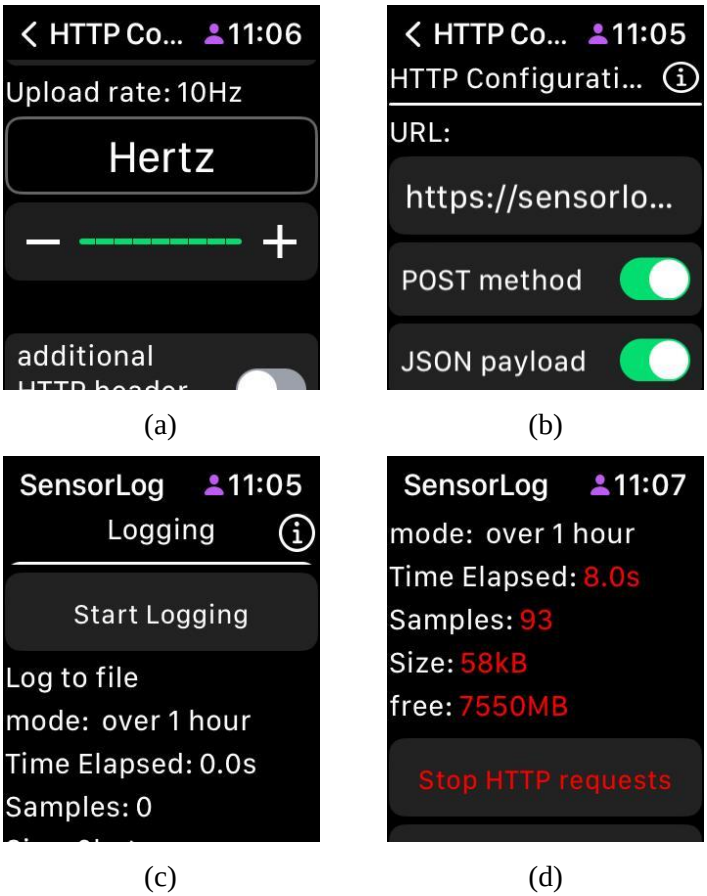

Figure 9: (a) Upload rate selection. (b) Selection of REST endpoint, HTTP(S) request method, and format. (c) SensorLog interface before the start of logging. (d) Logging/HTTP(S) requests in progress.

Depending on how the user has configured the app, SensorLog will collect certain data from the connected Apple Watch and store it on the device, and/or distribute it to a user-specified server via HTTP request as configured by the user [81]. The specific data that SensorLog was configured to collect and subsequently store or send via HTTP requests are listed below. The aforementioned steps for configuring SensorLog to collect sensor data from the smartwatch are depicted in Figure 9. The following is a list of all the sensor data collected from the smartwatch. Note that the pedometer data was dropped later due to causing the DL models to perform poorly, which will be discussed in detail in the Data Cleaning section of this paper.

- General Logging Info:
    - loggingTime(txt)
- Raw Accelerometer Data:
    - accelerometerAccelerationX(G)
    - accelerometerAccelerationY(G)
    - accelerometerAccelerationZ(G)

- Unbiased Data User Acceleration, Gravity, Heading, Rotation:
    - motionYaw(rad)
    - motionRoll(rad)
    - motionPitch(rad)
    - motionRotationRateX(rad/s)
    - motionRotationRateY(rad/s)
    - motionRotationRateZ(rad/s)
    - motionUserAccelerationX(G)
    - motionUserAccelerationY(G)
    - motionUserAccelerationZ(G)
    - motionQuaternionX(R)
    - motionQuaternionY(R)
    - motionQuaternionZ(R)
    - motionQuaternionW(R)
    - motionGravityX(G)
    - motionGravityY(G)
    - motionGravityZ(G)
- Steps, Distance, Pace, Cadence, Floors:
    - pedometerNumberofSteps(N)
    - pedometerAverageActivePace(s/m)
    - pedometerCurrentPace(s/m)
    - pedometerCurrentCadence(steps/s)
    - pedometerDistance(m)
    - pedometerFloorAscended(N)
    - pedometerFloorDescended(N)
- Device ID (set by user):
    - deviceID(txt)
- Data Labelling:
    - label(N)

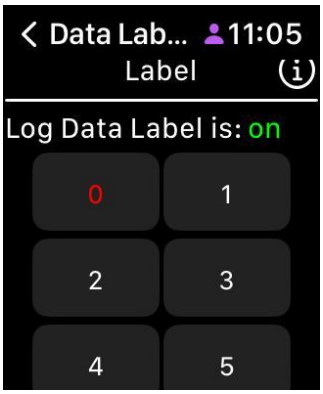

Figure 10: Data labeling using SensorLog

This data is collected from four subjects as they performed some day-to-day human activities such as—walking, sitting, laying down, ascending, and descending stairs for a specific period of time. In all cases, data is collected at a frequency of 10 samples per second, which is one record every 100 milliseconds (10Hz). Labeling training data using SensorLog (Figure 10) was determined to be as follows: 0 for 'Laying down', 1 for 'Sitting', 3 for 'Walking', 4 for 'Upstairs', and 5 for 'Downstairs', with the label "2" left unused. Figure 11 shows the Pandas dataframe of all collected data before cleaning and preprocessing.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 521541 entries, 0 to 521540
Data columns (total 39 columns):
 #   Column                                    Non-Null Count   Dtype
---  ------                                    --------------   -----
 0   loggingTime(txt)                          521541 non-null  object
 1   accelerometerTimestamp_sinceReboot(s)     521541 non-null  float64
 2   accelerometerAccelerationX(G)             521541 non-null  float64
 3   accelerometerAccelerationY(G)             521541 non-null  float64
 4   accelerometerAccelerationZ(G)             521541 non-null  float64
 5   motionTimestamp_sinceReboot(s)            521374 non-null  float64
 6   motionYaw(rad)                            521374 non-null  float64
 7   motionRoll(rad)                           521374 non-null  float64
 8   motionPitch(rad)                          521374 non-null  float64
 9   motionRotationRateX(rad/s)                521374 non-null  float64
 10  motionRotationRateY(rad/s)                521374 non-null  float64
 11  motionRotationRateZ(rad/s)                521374 non-null  float64
 12  motionUserAccelerationX(G)                521374 non-null  float64
 13  motionUserAccelerationY(G)                521374 non-null  float64
 14  motionUserAccelerationZ(G)                521374 non-null  float64
 15  motionAttitudeReferenceFrame(txt)         521374 non-null  object
 16  motionQuaternionX(R)                      521374 non-null  float64
 17  motionQuaternionY(R)                      521374 non-null  float64
 18  motionQuaternionZ(R)                      521374 non-null  float64
 19  motionQuaternionW(R)                      521374 non-null  float64
 20  motionGravityX(G)                         521374 non-null  float64
 21  motionGravityY(G)                         521374 non-null  float64
 22  motionGravityZ(G)                         521374 non-null  float64
 23  motionMagneticFieldX(µT)                  521374 non-null  float64
 24  motionMagneticFieldY(µT)                  521374 non-null  float64
 25  motionMagneticFieldZ(µT)                  521374 non-null  float64
 26  motionMagneticFieldCalibrationAccuracy(Z) 521374 non-null  float64
 27  pedometerStartDate(txt)                   482290 non-null  object
 28  pedometerNumberofSteps(N)                 521374 non-null  float64
 29  pedometerAverageActivePace(s/m)           521374 non-null  float64
 30  pedometerCurrentPace(s/m)                 521374 non-null  float64
 31  pedometerCurrentCadence(steps/s)          521374 non-null  float64
 32  pedometerDistance(m)                      521374 non-null  float64
 33  pedometerFloorAscended(N)                 521374 non-null  float64
 34  pedometerFloorDescended(N)                521374 non-null  float64
 35  pedometerEndDate(txt)                     482189 non-null  object
 36  batteryState(N)                           521374 non-null  float64
 37  batteryLevel(R)                           521374 non-null  float64
 38  label                                     521374 non-null  float64
dtypes: float64(35), object(4)
memory usage: 155.2+ MB
```

Figure 11: Dataset info before data cleaning

The dataset has 39 columns – including 'label' and 'timestamp'. 'label' is the name of the activity, 'timestamp' is the Unix timestamp, and the rest are the sensor readings and related information of a given sensor at a given instance of time. Our target variable is 'label' which will be predicted by our deep learning models for new data.

iii. Data Cleaning

First, unnecessary columns that are irrelevant to the activity recognition task are dropped from the dataset. This reduces the number of columns in the dataset from 39 to 27 columns (Figure 12). The timestamp datatype is then changed from a string to a DateTime object.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 514795 entries, 4668 to 510897
Data columns (total 27 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   accelerometerAccelerationX  514795 non-null  float64
 1   accelerometerAccelerationY  514795 non-null  float64
 2   accelerometerAccelerationZ  514795 non-null  float64
 3   motionYaw                   514628 non-null  float64
 4   motionRoll                  514628 non-null  float64
 5   motionPitch                 514628 non-null  float64
 6   motionRotationRateX         514628 non-null  float64
 7   motionRotationRateY         514628 non-null  float64
 8   motionRotationRateZ         514628 non-null  float64
 9   motionUserAccelerationX     514628 non-null  float64
 10  motionUserAccelerationY     514628 non-null  float64
 11  motionUserAccelerationZ     514628 non-null  float64
 12  motionQuaternionX           514628 non-null  float64
 13  motionQuaternionY           514628 non-null  float64
 14  motionQuaternionZ           514628 non-null  float64
 15  motionQuaternionW           514628 non-null  float64
 16  motionGravityX              514628 non-null  float64
 17  motionGravityY              514628 non-null  float64
 18  motionGravityZ              514628 non-null  float64
 19  pedometerNumberOfSteps      514628 non-null  float64
 20  pedometerCurrentPace        514628 non-null  float64
 21  pedometerCurrentCadence     514628 non-null  float64
 22  pedometerDistance           514628 non-null  float64
 23  pedometerFloorsAscended     514628 non-null  float64
 24  pedometerFloorsDescended    514628 non-null  float64
 25  label                       514628 non-null  object
 26  timestamp                   514795 non-null  datetime64[ns, pytz.FixedOffset(120)]
dtypes: datetime64[ns, pytz.FixedOffset(120)](1), float64(25), object(1)
memory usage: 110.0+ MB
```

Figure 12: Dataset info after dropping unnecessary columns

Next, we use the dropna() function in Pandas to remove the missing values. The following is the function syntax [82]:

```
dataframe.dropna(axis, how, thresh, subset, inplace)
```

- axis – 0 or 'index', 1 or 'column' (default 0) – This makes it easier to identify and then delete the rows or columns that contain missing values. If a value of 1 is supplied to this parameter instead of the usual value of 0, which is missing values from rows are removed, then missing values from columns are also removed.
- how – 'any' or 'all' (default – any) – This option provides criteria for the removal of missing values. If "any" is supplied, a row will be eliminated even if just one value is missing, but if "all," all values must be missing for a row or column to be erased.
- thresh – int(optional) – Thresh parameter is used to specify the count of Non-NA values that should be present in the passed dataframe.
- subset – array-like(optional) – This subset is a part of the dataframe considered along the other axis from where null values are removed.
- inplace – boolean – The inplace determines whether or not the operations carried out on the dataframe are permanent. The modifications brought about by the operations become permanent if true is

transmitted to it; otherwise, they do not. False is the default setting. The final output will be a dataframe with NA values removed.

Using this, we identify and remove 167 rows with missing values. Next, we use Pandas' drop_duplicates() function to remove duplicate rows from the dataframe. The following is the function syntax [83]:

```
dataframe.drop_duplicates(subset, keep, inplace)
```

- keep: {first, last, False}, default 'first' – Which duplicates should be maintained in the dataframe is decided by this. If 'first' is given, then any duplicates aside from the first are removed. Similarly, if 'last' is given, all duplicates aside from the last are removed. All duplicates are removed if false is given. The default value of 'first' is used in this work.

747 duplicated rows are identified and removed. Afterward, Panda's nunique() function is used to count the number of unique entries in each column of the dataframe [84]. It is useful in situations where the number of categories is unknown beforehand. It does not take any parameters and returns the number of unique entries in the requested columns as shown in Figure 13.

```
timestamp                  513881
motionYaw                  513143
motionRoll                 513093
motionPitch                513075
motionQuaternionZ          513075
motionQuaternionW          513030
motionQuaternionX          512978
motionQuaternionY          512944
motionRotationRateZ        506947
motionRotationRateX        506128
motionGravityX             505496
motionGravityY             504800
motionRotationRateY        503556
motionGravityZ             488345
motionUserAccelerationX    471156
motionUserAccelerationY    463356
motionUserAccelerationZ    371136
accelerometerAccelerationX 107061
accelerometerAccelerationZ  90008
accelerometerAccelerationY  83299
pedometerDistance            3308
pedometerNumberOfSteps       2582
pedometerCurrentPace         1735
pedometerCurrentCadence      1685
pedometerFloorsAscended        41
pedometerFloorsDescended       41
label                           5
dtype: int64
```

Figure 13: Number of unique values of each column in the dataset

Subsequently, we apply a selection-based approach for dimensionality reduction, which involves removing certain features or attributes from the dataset. The choice of which attributes to eliminate can be determined through experimentation and evaluating the model's performance or by examining the characteristics of the attributes such as the number of unique values, quality of measurements, or percentage of missing values, or through statistical tests that measure correlation. Using Panda's nunique() function, it was found that the pedometer sensor values change at a much slower frequency than the other sensor values, and hence have a relatively low number of unique values. When a feature has mostly the same value or has very little variation, it can make it difficult for the model to learn from it. Thus, we decide to remove all pedometer features,

reducing the dimensionality from 25 features to 19 features. Figure 14 shows the 19 features (index 0 to 18) and the target variable ‘label’, as well as the size of the dataset.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 513881 entries, 4668 to 510897
Data columns (total 20 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   accelerometerAccelerationX  513881 non-null  float64
 1   accelerometerAccelerationY  513881 non-null  float64
 2   accelerometerAccelerationZ  513881 non-null  float64
 3   motionYaw                   513881 non-null  float64
 4   motionRoll                  513881 non-null  float64
 5   motionPitch                 513881 non-null  float64
 6   motionRotationRateX         513881 non-null  float64
 7   motionRotationRateY         513881 non-null  float64
 8   motionRotationRateZ         513881 non-null  float64
 9   motionUserAccelerationX     513881 non-null  float64
 10  motionUserAccelerationY     513881 non-null  float64
 11  motionUserAccelerationZ     513881 non-null  float64
 12  motionQuaternionX           513881 non-null  float64
 13  motionQuaternionY           513881 non-null  float64
 14  motionQuaternionZ           513881 non-null  float64
 15  motionQuaternionW           513881 non-null  float64
 16  motionGravityX              513881 non-null  float64
 17  motionGravityY              513881 non-null  float64
 18  motionGravityZ              513881 non-null  float64
 19  label                       513881 non-null  object 
dtypes: float64(19), object(1)
memory usage: 82.3+ MB
```

Figure 14: Dataset info after data cleaning

iv. Exploratory Data Analysis

In this section, we explore the dataset after cleaning. Table 2 shows the number of samples collected for each activity. The graph (Figure 15) is a visualization of the table, showing the distribution of the samples across different activities. From the graph, it can be observed that there is a significant imbalance in the distribution of the class labels, with the majority of the samples being labeled as 'Sitting' and 'Laying Down'. Conversely, the activities 'Upstairs' and 'Downstairs' have the least representation in the dataset.

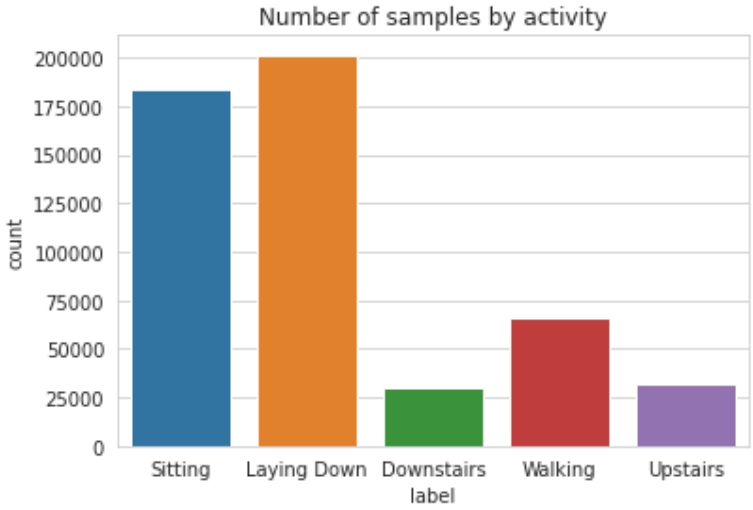

Figure 15: Number of samples for each activity in the dataset

Table 2: Number of samples for each activity in the dataset

Activity	Number of samples
Laying Down	201,521
Sitting	183,940
Walking	65,951
Upstairs	32,362
Downstairs	30,107

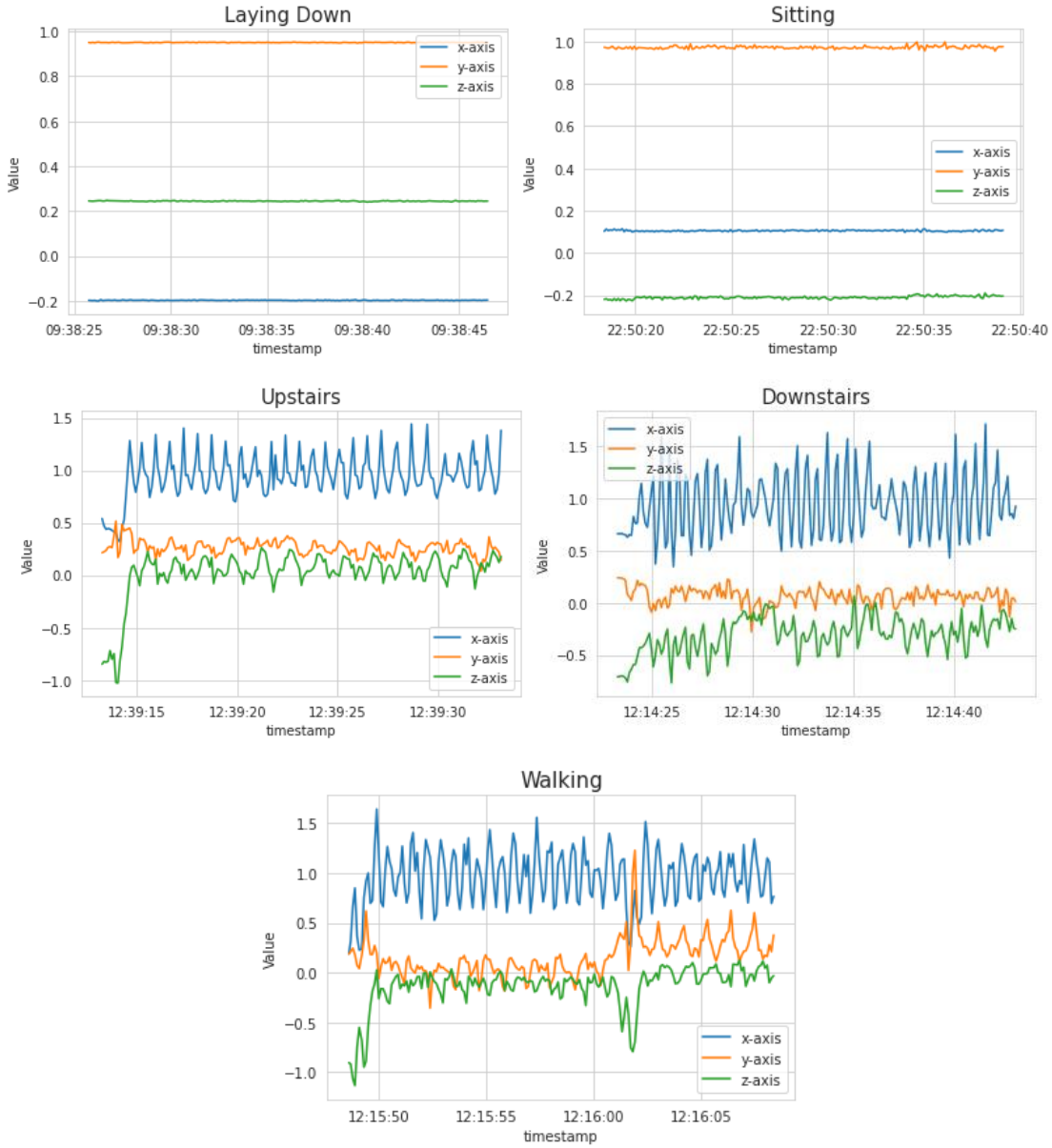

Figure 16: Accelerometer sensors readings in the x, y, and z axes over time for each activity

Figure 16 visualizes the signal values of the accelerometer sensor in the x, y, and z axes over time for each activity. It can be observed how the signals behave differently for each activity. For this visualization, a subset

of 200 samples was considered, which is equivalent to 20 seconds of activity (as the data was collected at a frequency of 10Hz). It is noticeable that the signal exhibits periodic behavior for activities such as 'Walking', 'Upstairs', and 'Downstairs', while it shows very little movement for stationary activities like 'Sitting' and 'Laying Down'. These signals are modeled as time-series data.

A pair-plot is used to visualize the correlation of each feature pair in a dataset against the class distribution, with the identical feature pairs rendered for the diagonal plots which represent the class distribution for that pair. A pair plot helps us comprehend and rapidly analyze the correlation matrix (Pearson) of the dataset since it clearly illustrates the correlation of each feature pair. Plotting "pairwise relationships in a dataset" can be done using Seaborn's pair plot function (seaborn.pairplot) [85]. In terms of level, Seaborn builds on Matplotlib and "provides a high-level interface for producing appealing and informative statistical visuals." To ensure that the class distribution makes sense in terms of correlation, the various feature pair plots are selected to be scatter plots. The histogram can effectively depict the density distribution of the classes, and thus is selected as the diagonal's plot type. A pair-plot of all the continuous data in the dataset is shown in Figure 17.

The linear correlation between two characteristics is measured by the Pearson (product-moment) correlation coefficient. It is the proportion of x and y's covariance to the sum of their standard deviations. It's known as Pearson's r and is frequently represented by the letter r. the value can be mathematically expressed with this equation:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \quad (1)$$

Where, r is Pearson's Correlation Coefficient, x_i and y_i are variables samples, $\bar{x}$ is the mean of values in the x variable, and $\bar{y}$ is the mean of values in the y variable. The following details concerning the Pearson correlation coefficient are crucial [85]:

- The Pearson correlation coefficient is a real number that can be any value between $-1 \leq r \leq 1$, inclusive.
- A positive correlation between x and y is shown by a value of r > 0. When r is less than zero, there is a negative correlation between x and y.
- The scenario where x and y have a perfect positive linear connection is represented by the maximum value of r = 1. To put it another way, greater x values match larger y values and vice versa. Likewise, the scenario when there is a perfect negative linear connection between x and y corresponds to the smallest value of r = - 1. In other words, lower y values and greater x values are equivalent.
- The circumstance when x and y don't have a linear connection is represented by the value r = 0.

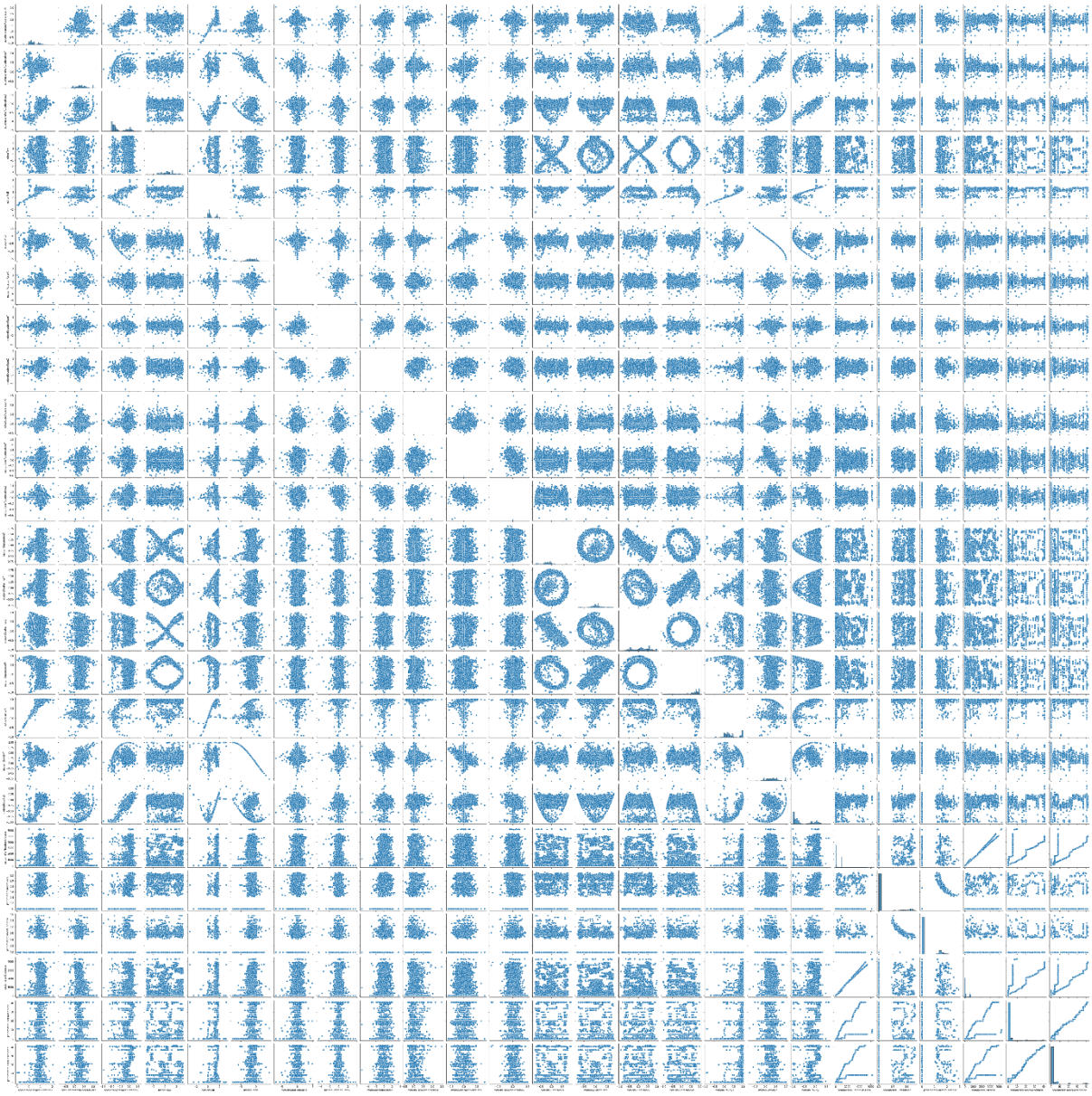

Figure 17: Pair plot of all continuous features in the dataset

In essence, better correlation and a closer affinity to a linear function are indicated by a bigger absolute value of r. A weaker correlation is indicated by a lower absolute value of r. To determine the pairwise Pearson's correlation coefficient of all columns in a dataframe, Pandas' dataframe.corr() function is used [85]. The function will disregard any columns with non-numeric data types [86]. Figure 18 is a heatmap of the pairwise correlation coefficients. The highest positive correlation coefficients can be seen in dark red, while the highest negative correlation coefficients are in dark blue. The correlation coefficients that are close to '0' can be seen in light gray color.

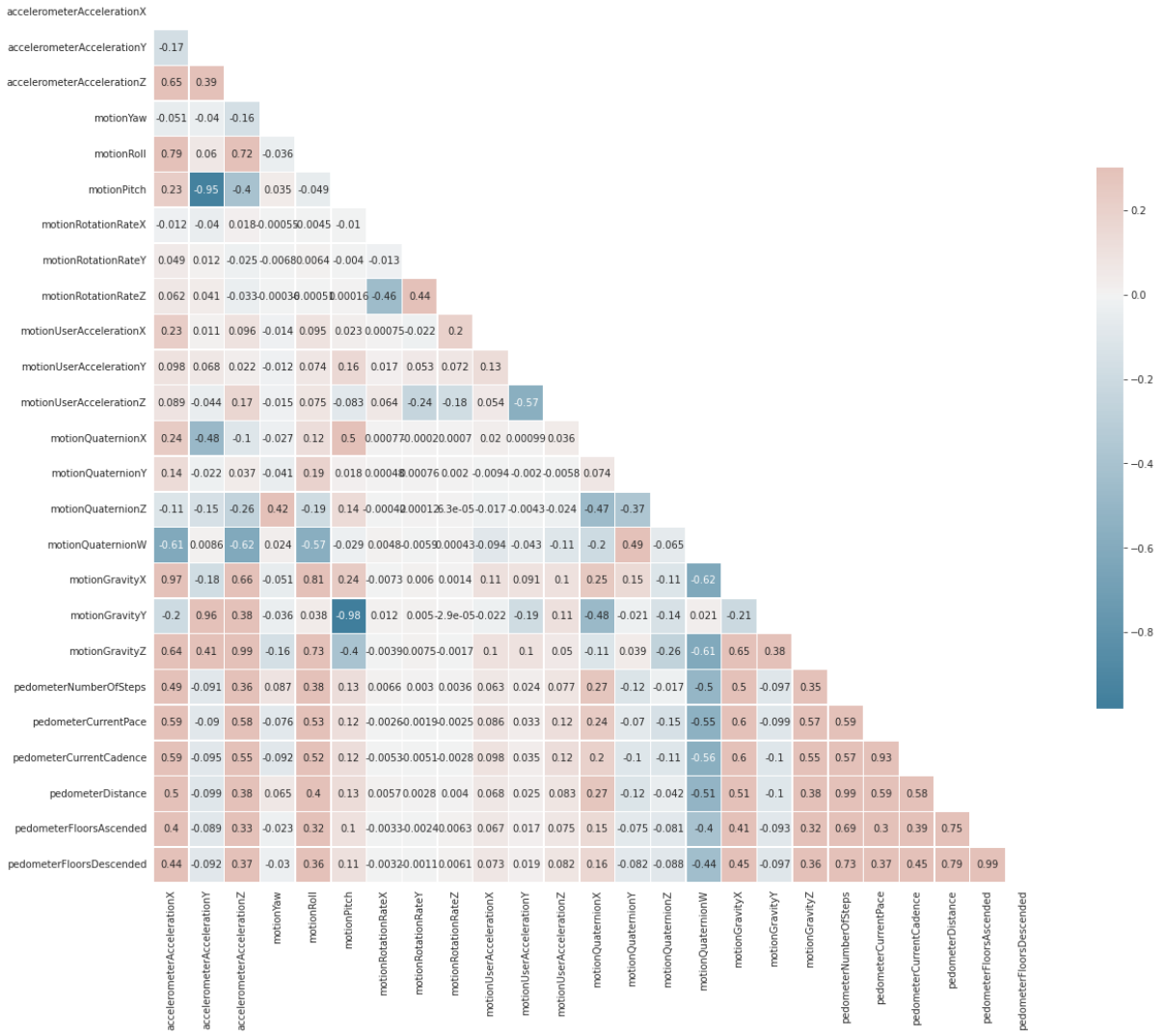

Figure 18: Heatmap of the pairwise correlation coefficients of dataset features

To understand the heatmap better, the absolute values of all the correlation coefficients are sorted to display the pairs of features with the highest correlation, whether it is a negative correlation or a positive one. Figure 19 shows the 20 pairs with the highest absolute values of correlation coefficients, sorted in ascending order. It is found that the pedometer features, which were decided to be removed earlier due to low variation, were all redundant due to high values of correlation among them. As for motionGravityX, motionGravityY, and motionGravityZ features, they expectedly show high absolute correlation with accelerometerAccelerationX, accelerometerAccelerationY, and accelerometerAccelerationZ, respectively. Normally, if two variables are highly correlated, keeping only one will help reduce dimensionality without much loss of information [87]. However, further analysis based on the models' performance during the experimentation phase, as well as the feature importance analysis that will be discussed in this paper, showed that these variables are significant to the models' learning, and dropping them results in marginally poorer performance. Therefore, motionGravityX, motionGravityY, and motionGravityZ were kept as features in the dataset.

		Correlation
motionGravityZ	accelerometerAccelerationX	0.641896
accelerometerAccelerationZ	accelerometerAccelerationX	0.646934
motionGravityZ	motionGravityX	0.653346
motionGravityX	accelerometerAccelerationZ	0.656651
pedometerFloorsAscended	pedometerNumberOfSteps	0.690548
motionRoll	accelerometerAccelerationZ	0.723743
motionGravityZ	motionRoll	0.725860
pedometerFloorsDescended	pedometerNumberOfSteps	0.731438
pedometerFloorsAscended	pedometerDistance	0.749221
motionRoll	accelerometerAccelerationX	0.787758
pedometerFloorsDescended	pedometerDistance	0.789155
motionGravityX	motionRoll	0.808672
pedometerCurrentCadence	pedometerCurrentPace	0.927314
motionPitch	accelerometerAccelerationY	0.947062
motionGravityY	accelerometerAccelerationY	0.956401
motionGravityX	accelerometerAccelerationX	0.970598
motionGravityY	motionPitch	0.981442
motionGravityZ	accelerometerAccelerationZ	0.988893
pedometerFloorsDescended	pedometerFloorsAscended	0.992823
pedometerDistance	pedometerNumberOfSteps	0.994598

Figure 19: Pairs of features with the highest absolute correlation coefficients in ascending order

v. Data Pre-processing

a) Windowing (Segmenting)

In this part, generate fixed-length sequences or windows are generated using the collected samples. Each sequence that is generated will contain 200 records, equivalent to 20 seconds of activity (as the data collection frequency is 10Hz). It is worth noting that the windows created have an 80% overlap. This is done because the activity is continuous and having an overlap in the windows ensures that each subsequent window carries some information from the previous window. Table 3 shows the number of generated sequences for each activity in the dataset.

Table 3: Number of segments for each activity

Activity	Segment Count
Laying Down	5,038
Sitting	4,597
Walking	1,648
Upstairs	811
Downstairs	749

b) Data Splitting

Data splitting is a technique used to divide a dataset into two separate groups: one for training and the other for testing. The purpose of this method is to evaluate the performance of statistical and machine-learning models [88]. The most commonly used data splitting method is random subsampling, where a random portion of the data is selected for testing, and the remaining data is used for training. The proportion of data used for training and testing can vary, with 80:20 being a common ratio, but other ratios such as 70:30, 60:40, and even 50:50 are also used in practice [88]. There is no clear guidance on the best ratio of training and testing data for a given dataset. In this work, we use the 80:20 random subsampling ratio, where 20% of the data will be used for testing the performance of the model. The 80% portion of the data is further split into an 80% training set and a 20% validation set. Thus, the original dataset was partitioned into three separate sets: the training set, validation set, and test set. The proportion of data allotted to each set was 64%, 16%, and 20%, respectively. The validation set is used in evaluating the performance of the model during the training process, whereas the test set was used to derive all the final results, which are presented in Chapter 4.

c) Data Standardization

Standardizing a dataset is rescaling the distribution of values to make the mean of observed values 0 and the standard deviation 1. This can be compared to centering the data or removing the mean value. When the data comprises input values with different scales, standardization can be helpful and even necessary in some machine learning and deep learning methods. Standardization is based on the presumption that the data fits a bell-shaped Gaussian distribution with a consistent mean and standard deviation. The statistical mean and standard deviation of the attribute values are calculated, the mean is subtracted from each value, and the result is divided by the standard deviation. The output of this procedure, known as standardizing a statistical variable, is a set of values with a mean of 0 and a standard deviation of 1 [89]. It is a must to be able to precisely estimate the mean and standard deviation of observable quantities in order to standardize. Centering is the process of removing the mean from the data, while scaling is the division of the standard deviation. Thus, the technique is occasionally referred to as "center scaling". The standard score of sample x is calculated as [90]:

$$z = \frac{x - u}{s} \tag{2}$$

where u is the mean of the training samples, and s is the standard deviation of the training samples. The mean and standard deviation estimates of a dataset can be more robust to new data than the minimum and maximum values. Here, we standardize our dataset using the scikit-learn object StandardScaler [90]. By calculating the pertinent statistics on the samples in the training set, centering and scaling are applied independently to each feature. Then, for applying to new data using the “transform” function, the StandardScaler object containing the scaling parameters is stored as a .bin file. Figure 20 shows the distribution of each feature in the dataset

before applying data standardization, and after where their distributions represent bell-shaped Gaussian distributions.

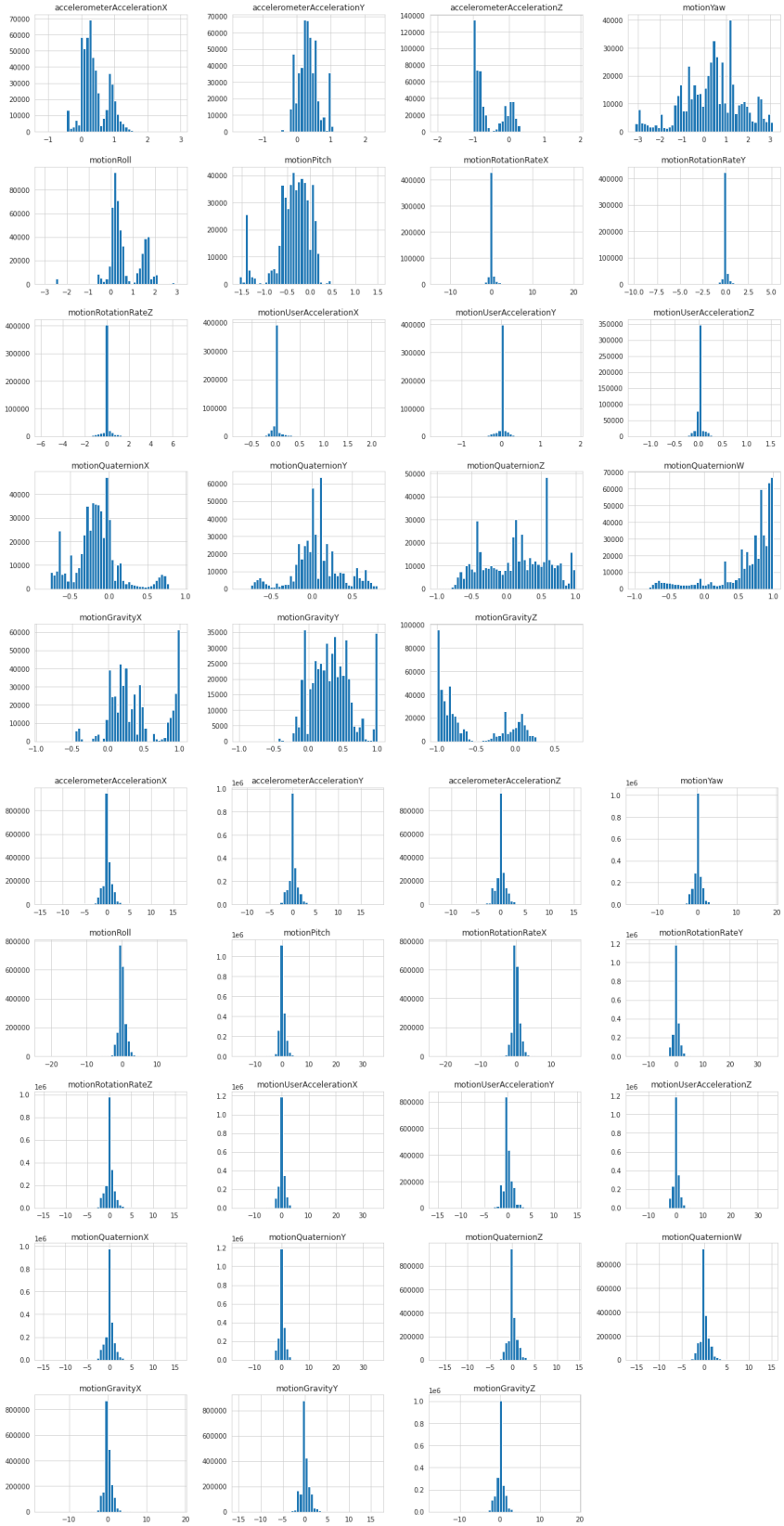

Figure 20: Distribution of each feature in the dataset before applying data standardization (top) and after (bottom)

vi. Deep Learning Models

a) Artificial Neural Network (ANN)

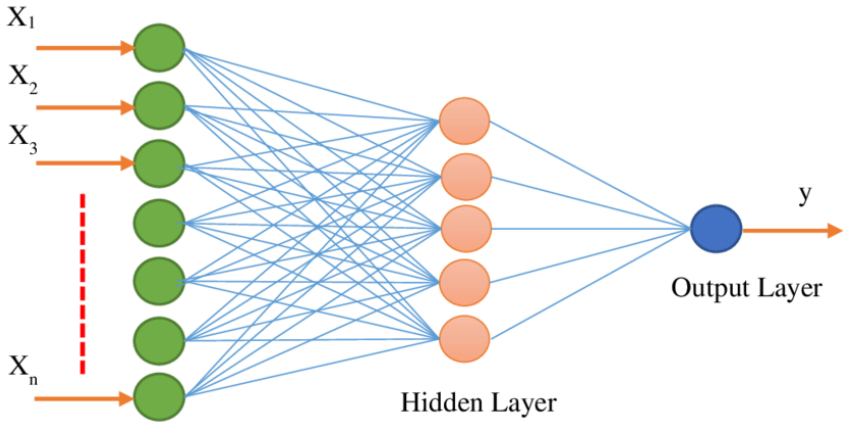

Figure 21: ANN structure

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functions of the human brain [91]. ANNs are a type of machine learning algorithm that consists of interconnected nodes, known as artificial neurons, which are arranged into layers. These layers receive inputs and apply transformations to produce outputs. The objective of an ANN is to learn and generalize patterns in data and use this information to make predictions or classify new inputs. One can think of a single perceptron (or neuron) as a Logistic Regression. An artificial neural network, or ANN, is made up of several perceptrons and neurons at each layer. Because inputs are exclusively processed in the forward direction, ANN is sometimes referred to as a feed-forward neural network – information travels from one layer to another without touching a node twice. The neurons in an ANN are connected to each other through directed edges, or weights, which represent the strength of the connection between two neurons. The outputs of neurons are passed from one layer to another, and the values of the weights are updated during training to minimize an objective function, such as the Mean Squared Error (MSE). The ANN is trained on a large dataset and the weights are updated in such a way that the ANN can make accurate predictions for new inputs.

ANN is composed of three layers: input, hidden layer, and output layer [92]. The structure of an ANN is shown in Figure 21 [93]. Inputs are received by the input layer, processed by the hidden layer, and then the output layer generates the output. In essence, each layer makes an effort to learn specific weights. A feedforward network consists of three key components: the input layer, hidden layers, and the output layer. The input layer receives the inputs of the problem, while the hidden layers determine and represent the relationship between the inputs and outputs through the use of synaptic weights. The output layer then emits the solutions to the problem. The architecture of a feedforward network is modeled with three basic elements: synaptic weights, which characterize the set of synapses, a linear combiner for summing input signals, and an activation function for limiting the amplitude of the neuron's output to a finite value. The activation function's input can be amplified by incorporating a bias term [92].

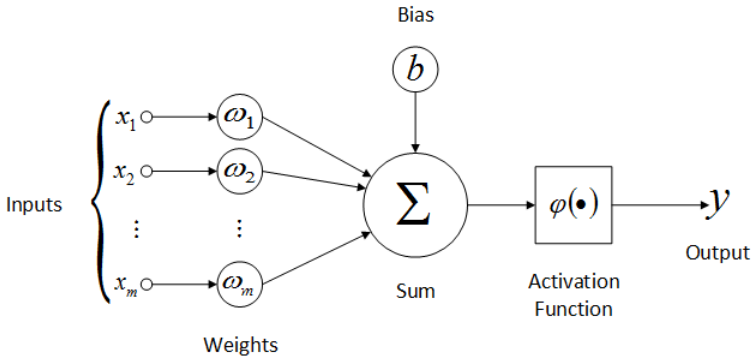

Figure 22: Artificial neuron model

In an artificial neuron, the weight serves as the numerical representation of a synapse. Negative weights indicate inhibitory connections while positive weights denote excitatory connections. The activity of the neuron cell is represented by the sum of all inputs, which are modified by the corresponding weights. This resulting computation is referred to as a linear combination. An activation function then regulates the magnitude of the output. The activation function defines the range in which the output is acceptable, which is typically between 0 and 1, or -1 and 1. This process is described in Figure 22 [94].

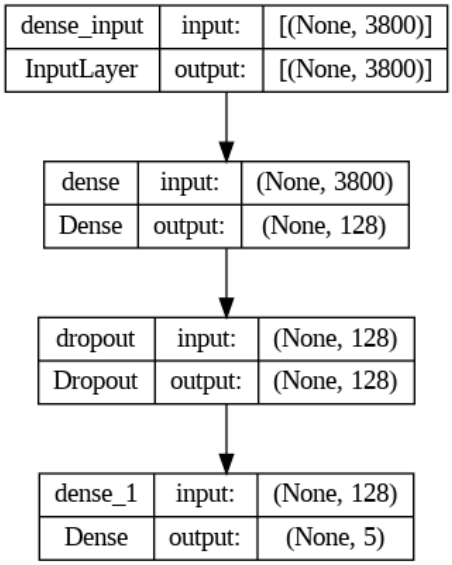

Figure 23: Proposed ANN model architecture

The proposed ANN model, shown in Figure 23, consists of a dense (fully connected) layer, a dropout layer, and an output layer. The complexity of the model is determined by the number of neurons in each layer. If there are fewer neurons, the network lacks the ability to properly train and process information, resulting in a low recognition rate. However, as the number of neurons increases, the recognition rate improves but the training ability of the network decreases. In the proposed ANN model, the number of neurons was tested between 32 and 128, and the recognition rate and time were evaluated. It was found that the recognition rate was low when using 32 neurons but increased when using 128 neurons. The recognition time also increased with the increase in the number of neurons. To prioritize recognition rate over recognition time, 128 neurons

were chosen for the dense layer. Due to its speed and computational efficiency, the activation function used for this layer is the ReLU activation function.

Dropout is a regularization technique used to prevent overfitting in a network by randomly disconnecting input and recurrent connections to hidden units during training. This leads to a better model performance by reducing the reliance on specific connections. Dropout can also be applied to the input connections of the hidden nodes, meaning that for a certain probability, the input data to each hidden layer will be ignored during node activation and weight updates. This can be done in TensorFlow by adding a dropout layer, with a range of 0 (no dropout) to 1 (no connection). A Dropout rate of 20% is often used as a balance between maintaining model accuracy and avoiding overfitting. In the proposed ANN model, the dropout rate is set to 0.20, which is the fraction of input units to be dropped. Figure 24 demonstrates the use of dropout in ANN [95].

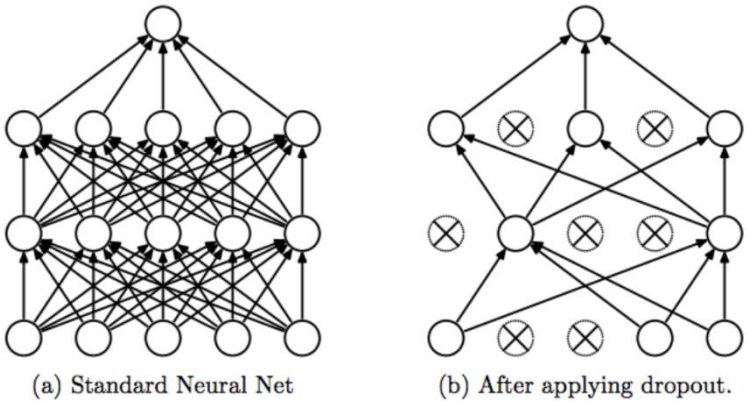

Figure 24: Dropout in Neural Networks

The final layer is the activation layer. Finally, the features are passed through a SoftMax activation function, which produces the activity features that are used for activity classification. Table 4 shows the ANN model summary.

Table 4: Proposed ANN model summary

Layer (Type)	Output Shape	Param #
Dense	(None, 128)	486,528
Dropout	(None, 128)	0
Dense	(None, 5)	645
Total params		487,173
Trainable params		487,173
Non-trainable params		0

b) Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) play a significant role in computer vision and are characterized by their unique hidden layer structure. These hidden layers include Convolutional Layers, Pooling Layers, and Fully Connected Layers, where the convolution and pooling functions serve as activation functions, as shown in Figure 25 [96].

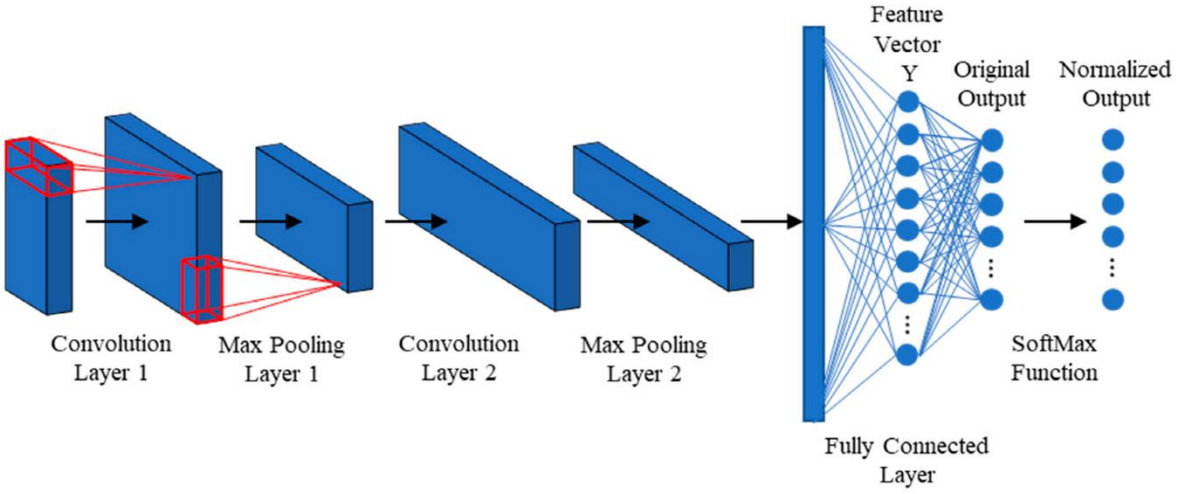

Figure 25: CNN structure

The primary computational responsibility in Convolutional Neural Networks (CNNs) lies with the Convolutional Layers (CONV layers), which are equipped with a set of filters (kernels) with adjustable weights. The depth of the filters and inputs is identical. Specifically, the forward propagation in a 1D CONV layer can be represented as follows [96]:

$$g_i = f\left[\sum_{n=1}^{N} \text{conv1}D\left(w_{i,n}, a_n\right) + b_i\right] \tag{3}$$

where g_i, is the calculation result of the ith filter; a is the input data of size $1 \times N_a \times N$, w_i is the weight matrix of the ith filter of size $1 \times N_w \times N$; b_i is the bias of the ith filter and f is the activation function.

The pooling layers reduce the size of the feature maps obtained from the CONV layers, with Max pooling being the most frequently utilized approach, which is expressed as follows [96]:

$$p_i(j) = \max_{(j-1)\times m < k \leq j \times m} (a_i(k)) \tag{4}$$

where $p_i(j)$ is the jth element of the ith feature map output by the pooling layer, and $a_i(k)$ is the kth element of the ith feature map input into the pooling layer. The size of the pooling layer filter is $1 \times m$. The final classification of the feature maps extracted by the Convolutional and Pooling Layers is performed by the Fully Connected Layers to produce the original output data. The normalization of the output data is carried out using

the SoftMax function, which calculates the probability distribution of the input samples across different categories.

Convolutional Neural Network (CNN) models were initially designed for image classification tasks, where the model learns to extract features from two-dimensional inputs. This same concept can also be applied to one-dimensional sequences of data, such as acceleration and gyroscopic data for human activity recognition [97], [98]. In this process, the model learns to extract relevant features from sequences of observations and map these internal features to different types of activities. One of the key advantages of using CNNs for sequence classification is that they can directly learn from raw time series data, without requiring domain expertise to manually engineer input features. Ideally, this approach leads to a model that has comparable performance to models fitted with engineered features. In Conv1D, the kernel slides along a single dimension, making it suitable for activity recognition tasks from accelerometer data. In this type of data, there are two dimensions - time steps and values of acceleration in three axes. Figure 26 illustrates the movement of the kernel along the time axis in accelerometer data. Each row represents the time series acceleration for a particular axis. The kernel can only move in one direction, along the time axis [99].

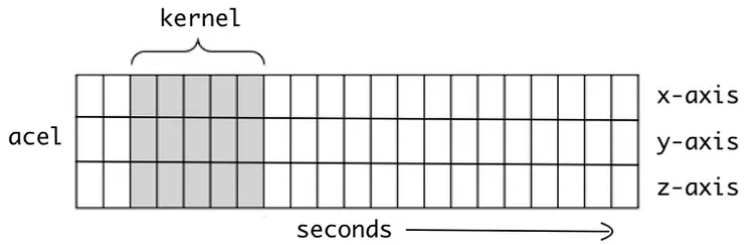

Figure 26: Movement of the kernel along the time axis

In the proposed CNN model, two 1D Convolutional Neural Network (CNN) layers are utilized, followed by a dropout layer for regularization and a pooling layer. It is a common practice to utilize two CNN layers in order to increase the chances of the model effectively learning features from the input data. Since CNNs have a tendency to learn quickly, a 50% dropout layer is implemented to slow down the learning process, ultimately leading to a better final model. The pooling layer consolidates the learned features by reducing their size to half of the original size, thereby retaining only the most crucial elements. The learned features are then flattened into a single vector, which is passed through a fully connected layer before reaching the output layer. The fully connected layer serves as an intermediary between the learned features and the output, allowing for the interpretation of the learned features before making a prediction. The architecture of the model is shown in Figure 27.

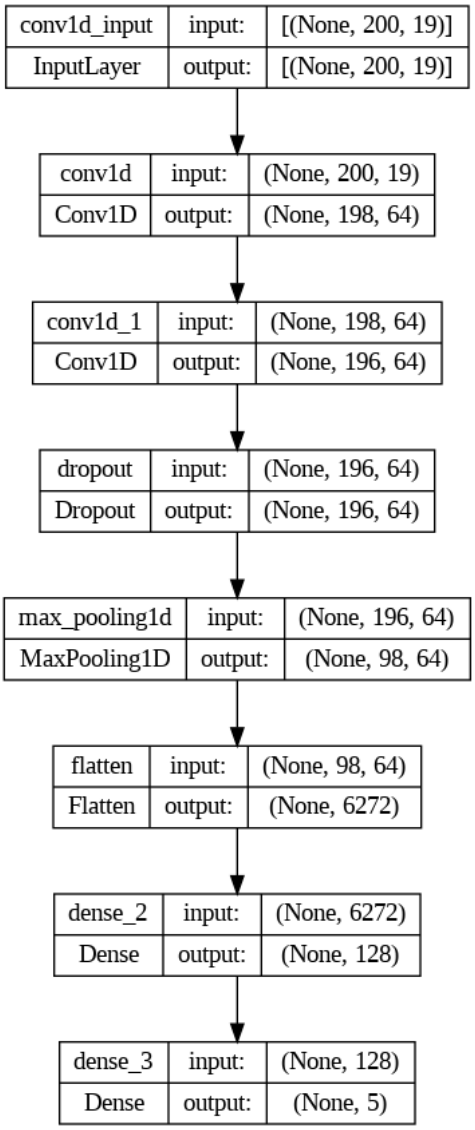

Figure 27: Proposed CNN model architecture

For this model, a standard configuration of 64 parallel feature maps and a kernel size of 3 is used. The feature maps represent the number of times the input is interpreted, whereas the kernel size refers to the number of input time steps considered as the input sequence is processed onto the feature maps. The optimization of the network is achieved through the use of the efficient Adam version of Stochastic Gradient Descent. Given the nature of the multi-class classification problem, the categorical cross-entropy loss function is employed. Additionally, the ReLU activation function is used here in the two Conv1D layers as well as the first dense layer due to its speed and computational efficiency. The model is trained for a fixed number of 200 epochs with Early Stopping, using a batch size of 64 samples, where 64 windows of data are exposed to the model before the weights are updated. Table 5 shows the CNN model summary.

Table 5: Proposed CNN model summary

Layer (Type)	Output Shape	Param #
Conv1D	(None, 198, 64)	3,712
Conv1D	(None, 196, 64)	12,352
Dropout	(None, 196, 64)	0
MaxPooling1D	(None, 98, 64)	0
Flatten	(None, 6272)	0
Dense	(None, 128)	802,944
Dense	(None, 5)	645
Total params		819,653
Trainable params		819,653
Non-trainable params		0

c) Long Short-Term Memory (LSTM)

Recurrent neural networks, or RNNs, have gained significant popularity in various fields of research that involve sequential data, such as text, audio, and video. However, traditional RNNs that use sigma or tanh cells have a limitation in their ability to effectively learn and retain important information from input data when there is a large gap in the input sequence. To address this issue, researchers introduced gate functions into the cell structure of RNNs, resulting in the development of the long short-term memory (LSTM) model. The LSTM model is able to handle the problem of long-term dependencies in input sequences much more effectively than traditional RNNs. Since its introduction, the LSTM has been widely used in deep learning research and has been responsible for achieving many exciting results in the field [100]. As a result, the LSTM has become a central focus in the field of deep learning.

Recurrent neural networks have units or blocks called long short-term memory (LSTM). Certain artificial memory techniques are made to be used by recurrent neural networks, which can aid these artificial intelligence programs in more accurately mimicking human reasoning. Long short-term memory blocks are used by the recurrent neural network to offer context for how the program receives inputs and produces outputs. A complex structure, the long short-term memory block has several parts, including weighted inputs, activation functions, inputs from earlier blocks, and final outputs. Because the algorithm uses a structure built on short-term memory processes to build longer-term memory, the unit is known as a long short-term memory block [100].

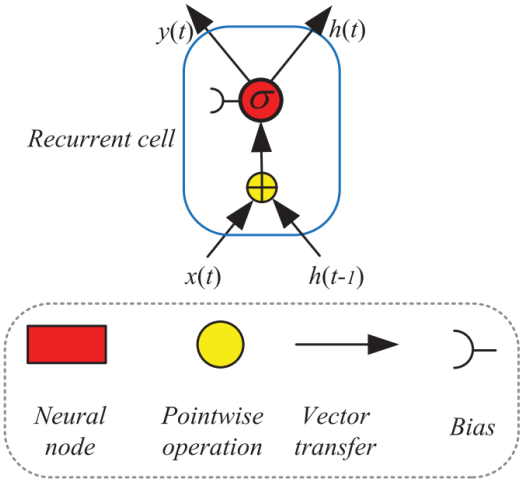

Figure 28: Standard recurrent sigma cell schematic

Recurrent Neural Networks (RNNs) are a type of neural network that incorporate feedback connections, allowing for the states of the recurrent layers or hidden layers to be affected by both past states and current input. These recurrent layers can be organized in different architectures, which is what primarily distinguishes one RNN from another. The types of recurrent cells and inner connections used in the network can also impact its capacity and capabilities. Figure 28 shows the standard recurrent sigma cell schematic [100]. The standard recurrent sigma cell can be mathematically expressed as follows:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$$
$$y_t = h_t \quad (5)$$

where x_t, h_t, and y_t denote the input, the recurrent information, and the output of the cell at time t, respectively; W_h and W_x are the weights; and b is the bias.

Standard recurrent cells, such as sigma cells and tanh cells, are often used in RNNs and have had some success in certain problems. However, RNNs that only utilize standard recurrent cells struggle with handling long-term dependencies. As the gap between related inputs increases, it becomes increasingly difficult for the network to learn the connection information. Research by Hochreiter [101] and Bengio et al. [102] has identified that the fundamental problem with long-term dependencies in RNNs is that error signals flowing backward in time tend to either become too large or disappear entirely. Hochreiter and Schmidhuber [101] introduced a solution in 1997 to address the issue of "long-term dependencies" in recurrent cells by incorporating a "gate" within the cell, known as the LSTM cell. This improved the ability of the standard recurrent cell to retain information over time. Since this initial proposal, many researchers have further developed and popularized the LSTM cell, including modifications such as LSTM without a forget gate, LSTM with a forget gate, and LSTM with a peephole connection. Typically, when referring to an LSTM cell, it is assumed to include a forget gate [100].

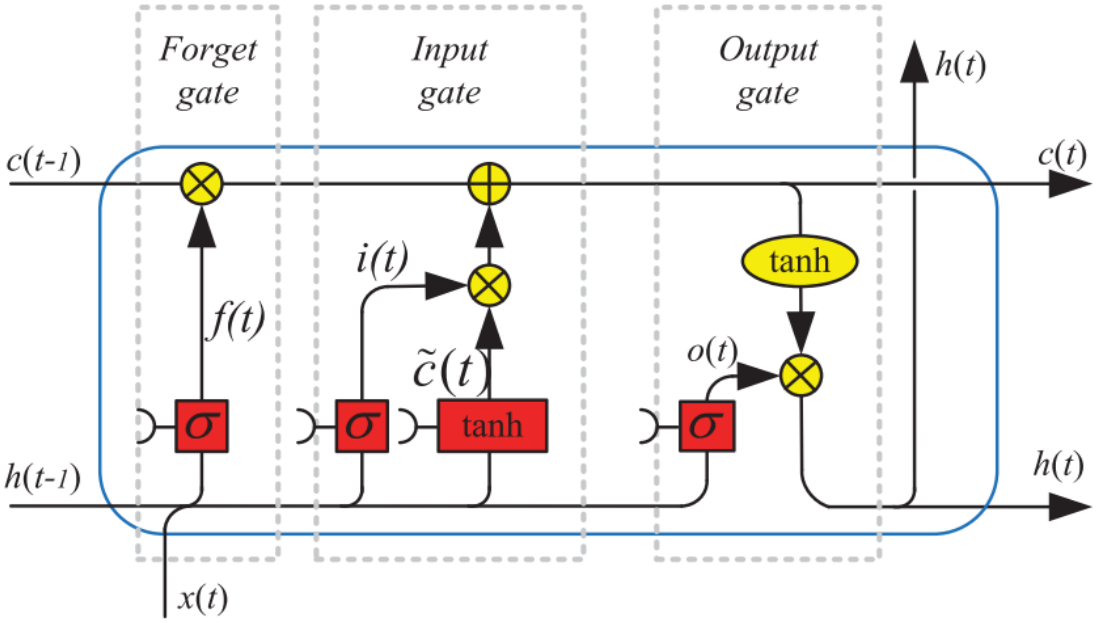

Figure 29: Schematic of LSTM cell incorporating a forget gate

In 2000, Gers, Schmidhuber, and Cummins made a modification to the original Long Short-Term Memory (LSTM) by incorporating a forget gate into the LSTM cell [103]. Figure 29 illustrates the internal connections of this modified LSTM cell, and from these connections, the LSTM cell can be mathematically represented as follows [100]:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$
$$h_t = o_t \cdot \tanh(c_t) \quad (6)$$

where c_t denotes the cell state of LSTM, and the operator '$\cdot$' denotes the pointwise multiplication of two vectors.

The input gate in an LSTM cell is responsible for determining which of the input values should be used to update the memory. This is accomplished through the use of two different functions: the sigmoid function and the tanh function. The sigmoid function is used to determine whether to allow a value of 0 or 1 through the gate, while the tanh function is used to assign a weight to the data provided. This weight is then used to determine the importance of the data on a scale of -1 to 1. The following are the mathematical expressions of the input gate [100]:

$$i_t = \sigma(W_{ih} h_{t-1} + W_{ix} x_t + b_i)$$
$$\tilde{c}_t = \tanh(W_{\tilde{c}h} h_{t-1} + W_{\tilde{c}x} x_t + b_{\tilde{c}}) \quad (7)$$

where W_i, $W_{\tilde{c}}$, and W_o are the weights.

The forget gate, on the other hand, is responsible for finding the details that should be removed from the block. This is determined by a sigmoid function, which looks at the preceding state (h_{t-1}) and the content input (x_t) for each number in the cell state (c_{t-1}). The following are the mathematical expression of the forget gate [100]:

$$f_t = \sigma\left(W_{fh}h_{t-1} + W_{fx}x_t + b_f\right) \tag{8}$$

The forget gate plays a crucial role in the LSTM cell by determining which information will be discarded from the cell state. When the value of the forget gate (f_t) is 1, it keeps the information, while a value of 0 means it discards all the information. Research by Jozefowicz et al. in 2015 [104] found that increasing the bias of the forget gate (b_f) often leads to improved performance of the LSTM network.

Finally, the output gate is used to determine the output of the block. This is done by using both the block's input and memory to make this determination. The sigmoid function is used to determine whether to allow a value of 0 or 1 through the gate, while the tanh function is used to assign a weight to the values provided. This weight is then used to determine the relevance of the values on a scale of -1 to 1 and is multiplied by the sigmoid output to produce the final output. The following is the mathematical expression of the output gate [100]:

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \tag{9}$$

LSTMs are designed to work with sequence data. By adding layers to an LSTM, it allows for the greater abstraction of input observations over time, meaning the input data can be broken down into chunks or represented at different time scales. Stacked LSTMs, also known as deep LSTMs, were first introduced in a study on speech recognition and were able to beat a benchmark on a difficult standard problem [105]. This technique is now commonly used for tackling complex sequence prediction issues. In the aforementioned study, it was found that the depth of the network was more important than the number of memory cells in a given layer for achieving good results [105]. A stacked LSTM architecture is made up of multiple LSTM layers, with each layer providing a sequence output rather than a single value output to the layer below it. To stack LSTM layers, the configuration of the prior layer must be altered to output the same shape of the input array as input for the subsequent layer. This can be achieved by setting the return_sequences argument on the layer to "True" instead of the default "False". Our proposed LSTM model (Figure 30) consists of two LSTM layers. The first LSTM layer provides a sequence output that is fed as input to the second LSTM layer. The second LSTM layer produces an array of data that includes information about time and activity. This array is then passed through a fully connected layer, which helps to identify global activity characteristics.

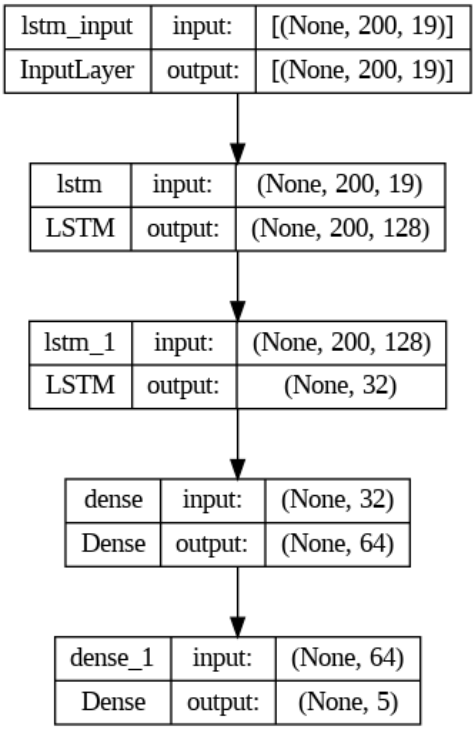

Figure 30: Proposed LSTM model architecture

As mentioned earlier, the complexity of the model is determined by the number of neurons in each layer. In the proposed LSTM model, the number of neurons was tested between 32 and 128, and the recognition rate and time were evaluated. To prioritize the recognition rate without significantly increasing recognition time, 128 neurons were chosen for the first LSTM layer and 32 for the second LSTM layer. Dropout, as mentioned earlier, is a regularization technique used to prevent overfitting in a network by randomly disconnecting input and recurrent connections to LSTM units during training. It is recommended to add a Dropout layer in combination with each LSTM layer to prevent overfitting. In this work, the dropout rate for each of the two LSTM layers is set to 0.20, which is the fraction of input units to be dropped.

Table 6: Proposed LSTM model summary

Layer (Type)	Output Shape	Param #
LSTM	(None, 200, 128)	75,776
LSTM	(None, 32)	20,608
Dense	(None, 64)	2,112
Dense	(None, 5)	325
Total params		98,821
Trainable params		98,821
Non-trainable params		0

After the LSTM layers have processed the input data and made predictions toward the desired output, the shape of the output needs to be reduced to match the desired output. The final layer that is added is the activation layer. This layer can technically be included in the dense layer, but there is a benefit to keeping them separate. In some cases, it may be useful to retrieve the reduced output of the density layer of the model. The dense layer (fully connected layer) is set to have 64 neurons with ReLU activation. Finally, the features are passed through a SoftMax activation function, which produces the activity features that are used for activity classification. Table 6 shows the LSTM model summary.

d) CNN-Long Short-Term Memory (CNN-LSTM)

The CNN LSTM is a specialized version of the LSTM architecture that is specifically designed to handle sequence prediction problems with inputs that have a spatial structure, such as images or videos. This is in contrast to the standard LSTM which is not well-suited for this type of input. The CNN LSTM architecture combines the features of both CNN and LSTM, using CNN layers to extract features from input data, and LSTMs to handle sequence prediction. This architecture is specifically designed to handle visual time series prediction problems, such as generating textual descriptions of activities shown in a sequence of images or videos. CNN LSTMs are well suited for activity recognition and other vision tasks that involve sequential inputs and outputs, as they are deep both spatially and temporally, and can be applied to a wide range of problems [106].

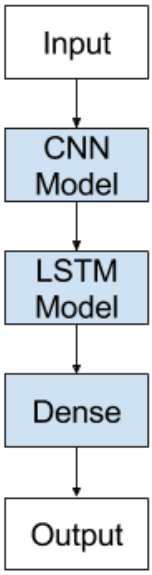

Figure 31: CNN-LSTM architecture

The CNN LSTM architecture, also known as the LRCN model – short for Long-term Recurrent Convolutional Network – uses a CNN as a front end to an LSTM. A CNN-LSTM architecture can be created by adding CNN layers at the beginning, followed by LSTM layers, and a Dense layer at the output, as shown in Figure 31. This architecture can be thought of as consisting of two separate models: the CNN Model, which extracts features from the input, and the LSTM Model, which interprets these features across time steps [107].

In a CNN, features are extracted from the input signal through the use of convolution operations performed on the signal and kernels. Activation functions are used to detect features captured by the kernels, and ReLU is a common choice to neutralize negative values to zero. The pooling layer receives local characteristics from the convolution layer, performs a local sampling operation, and reduces the size of the network. Together, the pooling and convolution layers create a deep structure that can automatically extract important information from the input signal, such as activity data. An LSTM layer is a type of recurrent neural network that has feedback connections, called cells, which can remember information from previous time intervals in a sequence. It has three gates, input, output, and forget gate, which control the flow of information in and out of the cell. This type of layer is often combined with a CNN to improve the accuracy of recognizing transition activities [107]. To prevent the rapid change of information, the training speed needs to be decreased, which can be achieved by using a learning rate. The features obtained are then presented to the SoftMax activation function to obtain the activity features for activity classification. The goal is to have the final output features that are robust and consistent, which allows for accurate activity classification [108].

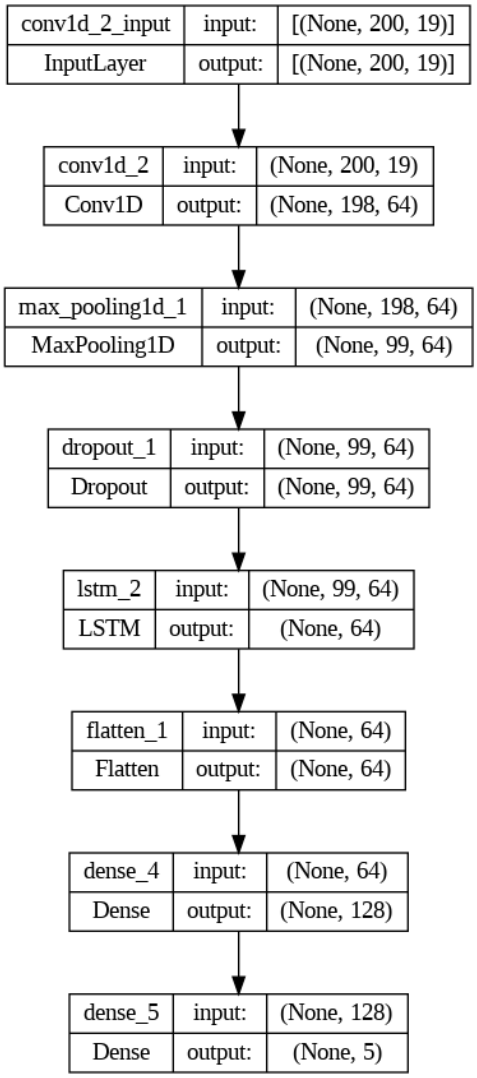

Figure 32: Proposed CNN-LSTM model architecture

In the proposed CNN-LSTM model shown in Figure 32, a standard configuration of 64 parallel feature maps and a kernel size of 3 is used for the Conv1D layer. The convolution layer is followed by a pooling layer. The pooling layer consolidates the learned features by reducing their size to half of the original size. The pooling layer is followed by a 50% dropout layer. Next is the LSTM layer, which is set to have 64 neurons and a 20% dropout rate. The learned features are then flattened into a single vector, which is passed through a fully connected layer before reaching the output layer. The dense layer (fully connected layer) is set to have 128 neurons with ReLU activation. Finally, the features are passed through a SoftMax activation function, which produces the activity features that are used for activity classification. Table 7 shows the CNN-LSTM model summary.

Table 7: Proposed CNN-LSTM model summary

Layer (Type)	Output Shape	Param #
Conv1D	(None, 198, 64)	3,712
MaxPooling1D	(None, 99, 64)	0
Dropout	(None, 99, 64)	0
LSTM	(None, 64)	33,024
Flatten	(None, 128)	0
Dense	(None, 128)	8,320
Dense	(None, 5)	645
Total params		45,701
Trainable params		45,701
Non-trainable params		0

vii. Hyperparameter Tuning

a) Learning Rate

The learning rate is a crucial hyperparameter that determines the magnitude of adjustments made to the model based on the estimated error during each weight update. The process of learning and training the deep learning model can be complex because the information at the input of each layer can change based on the parameters of the previous layer. To mitigate this issue, the training speed is decreased by adjusting the learning rate. The learning rate parameter determines the size of the steps taken by the model based on the estimated error of the model. Choosing a specific learning rate is crucial, as a value that is too low can cause the training process to take a long time, and a value that is too high can cause the model to learn in an unstable manner and converge too quickly to a suboptimal solution [109]. In this study, the learning rate for all models is set to 0.0001, prioritizing model learning over training time.

b) Batch Size

The batch size is a variable that determines the number of training samples that will be processed before the model's internal parameters are updated during gradient descent. It determines how many training samples are used in each iteration. Due to a large amount of data, it is not computationally efficient to use all of the data in one go, so the data is divided into smaller groups called batches. In this work, the batch size was experimented with and a batch size of 64 was found to be optimal for all HAR models.

c) Epochs

The number of epochs refers to the number of times the entire training dataset is passed through during gradient descent. A single epoch refers to a complete iteration of the training dataset, providing each sample with an opportunity to update the internal model parameters. An epoch can consist of one or multiple batches, with the simplest example being the batch gradient descent learning algorithm, where one epoch consists of one batch. The number of epochs is usually large, ranging from hundreds to thousands, to allow the learning algorithm to continue until the error from the model has been significantly reduced. In literature, the number of epochs is set to values such as 10, 100, 500, 1000, and larger [110]. In this work, the number of epochs is set to 200 with each epoch consisting of 64 batches for all HAR models.

Early Stopping

The number of training epochs in a deep learning model can significantly impact the final outcome. Too many epochs can result in overfitting of the training data, while too few can result in an underfitting model. To address this issue, early stopping is a method that enables the specification of a large number of training epochs, with the ability to halt the training process once the model performance stops improving on a validation dataset. Keras supports early stopping through the use of the EarlyStopping callback [111]. This callback allows the user to monitor a specific performance measure and terminate training when a specified trigger is reached. The "monitor" argument is used to specify the performance metric to be monitored, with a default setting of 'val_loss' for the loss on the validation dataset. The "mode" argument also needs to be specified, indicating whether the objective is to maximize or minimize the chosen performance measure. By default, the mode is set to 'auto' to minimize loss or maximize accuracy. It is important to note that the first instance of no improvement in the model performance may not necessarily be the optimal time to stop training. In some cases, the model may temporarily worsen before significantly improving. To account for this, the "patience" argument can be set to specify the number of epochs to wait for improvement before stopping training [111]. In this work, the "patience" argument is set to 20 epochs. To determine the epoch at which training was stopped, the "verbose" argument is set to 1, which will print the epoch number once training is halted.

d) Activation Functions

ReLU

The activation function plays a crucial role in determining whether a neuron should be activated or not. It does this by taking the weighted sum of the inputs and adding bias to it. The main purpose of the activation function is to introduce non-linearity into the output of a neuron. This non-linearity allows the neural network to learn more complex and nuanced patterns in the data, making it more accurate in its predictions. The rectified linear activation function, also known as ReLU, is a commonly used function for hidden layers in deep neural networks. This function is popular because it is easy to implement and is effective in overcoming the limitations of other previously popular activation functions such as Sigmoid and Tanh. ReLU is less likely to experience issues such as "dead" or saturated units, and it is also less likely to encounter vanishing gradients, which can prevent deep models from being trained. The ReLU function is mathematically defined as [112]:

$$y = \begin{cases} \max(0, x) \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases} \tag{10}$$

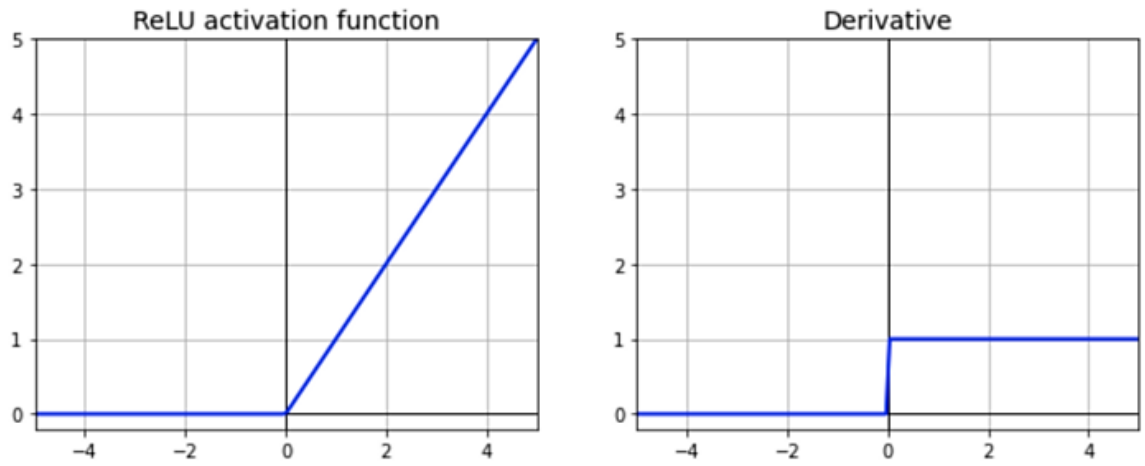

Figure 33: ReLU function graph (left) and its derivative (right)

To introduce non-linearity, the ReLU function is divided into two linear halves (Figure 33). When a function's slope is not constant, it is considered non-linear. The ReLU function is non-linear at zero, but the slope is either 0 (for negative inputs) or 1 (for positive inputs). Since its derivative is zero for any negative input, the ReLU function is not differentiable. Additionally, the output of ReLU has no maximum value, which helps the Gradient Descent algorithm. It is noteworthy that such a basic function can perform so well in deep neural networks, in comparison to Sigmoid and Tanh [113]. In contrast to the tanh or sigmoid functions that require a costly exponential calculation, the ReLU function can be implemented by simply applying a threshold to the activation at zero. Therefore, it is computationally more efficient and faster to train models using the ReLU activation function [112]. This makes it a popular choice for deep neural network architectures, especially when dealing with large datasets or real-time applications.

SoftMax

Another activation function used in this work is the SoftMax function. The SoftMax function is a function that converts a vector of K real numbers into a vector of K real numbers that add up to 1. The input values can range from positive, negative, zero, or greater than one, but the SoftMax function converts them into probabilities that fall between 0 and 1. When one of the inputs is small or negative, the SoftMax function turns it into a small probability, and when an input is large, it turns it into a large probability, yet it will always stay between 0 and 1 [114]. The SoftMax formula is as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{11}$$

K is the number of classes in the multi-class classifier. The SoftMax function takes in an input vector composed of elements (z_0, ... z_K), where each z_i value is an element of the input vector and can take on any real value, whether it be positive, negative, or zero. The standard exponential function e^{z_i} is applied to each element of the input vector, resulting in positive values greater than 0, yet fixed in the range (0, 1) which is what is required for a probability. The normalization term, found in the denominator of the formula, ensures that all output values of the function will sum to 1 and be within the range of (0, 1), making it a valid probability distribution [114].

The SoftMax function is also known as the softargmax function or multi-class logistic regression. This is because it is a generalization of logistic regression that can be used for multi-class classification and its formula is similar to the sigmoid function used in logistic regression [114]. However, it should be noted that the SoftMax function can only be used in a classifier when the classes are mutually exclusive. Many multi-layer neural networks have a final layer that outputs real-valued scores that are not scaled and may be difficult to work with. In such cases, the SoftMax function is very useful as it converts the scores into a normalized probability distribution that can be displayed to users or used as input for other systems [114]. It is therefore common, as done in this work, to include a SoftMax function as the last layer of a neural network.

e) Loss Function

Categorical Cross-Entropy

Cross-entropy is a widely used loss function in machine learning. It is a measure taken from the field of information theory that builds upon the concept of entropy and calculates the difference between two probability distributions. It is closely related to, but different from, KL divergence, which calculates the relative entropy between two probability distributions. Cross-entropy, on the other hand, can be thought of as calculating the total entropy between the distributions. Entropy is the amount of information required to transmit a randomly selected event from a probability distribution. A distribution with events that have a high probability of occurring has a low entropy, whereas a distribution where events that have an equal probability has a higher entropy. Cross-entropy builds upon the concept of entropy from information theory and calculates

the amount of information required to represent or transmit an average event from one distribution compared to another distribution [115]. The cross-categorical entropy can be mathematically expressed as [116]:

$$CCE = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\vec{p}_i[y_i]\right) \tag{12}$$

where N is the number of examples, y_i is the target class index, and $\vec{p}_i$ is the neural network output (probability distribution).

Categorical Cross-Entropy is used with multi-class classification, where each class is given a distinct integer value, and the goal values are in the range of {0, 1, 3, ..., n}. It is the preferred loss function mathematically within the maximum likelihood inference paradigm. The loss function should be assessed first, and only altered if necessary. For each class in the task, the average difference between the actual and predicted probability distributions will be calculated as a score by cross-entropy. A perfect cross-entropy value is zero once the score is minimized [117]. Employing the cross-entropy error function, as opposed to the sum-of-squares, for a classification problem results in both faster training and enhanced generalization [118].

The choice of activation function depends on the specific application. In this case, we have multiple classes but only one can be present at a time. For this type of problem, the SoftMax activation function is often the best choice because it allows us to interpret the outputs as probabilities. The loss function and activation function are often chosen together. When using the SoftMax activation function, it is common to use cross-entropy as the loss function, or more specifically categorical cross-entropy in this case since we are dealing with a multiclassification problem. These two functions work well together because the cross-entropy function cancels out the plateaus at each end of the SoftMax function, which speeds up the learning process.

f) Optimizers

Adam

Adam optimization algorithm is a stochastic gradient descent extension that has lately gained more popularity for use in computer vision and natural language processing. In place of the conventional stochastic gradient descent method, Adam is an optimization technique that may be used to iteratively update network weights depending on training data. Adam was introduced by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in 2015 [119]. The authors list the following advantages of using Adam on non-convex optimization problems: simple to put into action; effective in computation; memory needs are minimal; rescaling of the gradients diagonally is not affected; effective for issues with vast amounts of data or parameters; suitable for non-stationary goals; fit for issues involving extremely noisy or sparse gradients, and hyperparameters often require minimal adjustments and may be interpreted intuitively.

Compared to classical stochastic gradient descent, Adam is unique. For all weight updates, stochastic gradient descent maintains a constant learning rate (referred to as alpha), which does not fluctuate throughout training. As learning progresses, a learning rate is maintained and independently adjusted for each network weight (parameter). Using estimations of the first and second moments of the gradients, the approach calculates unique adaptive learning rates for various parameters. According to the authors [119], Adam combines the benefits of two more stochastic gradient descent modifications. Specifically: the adaptive gradient algorithm (AdaGrad): this method increases performance in situations with sparse gradients by maintaining a per-parameter learning rate; and Root Mean Square Propagation (RMSProp): this method additionally retains per-parameter learning rates that are customized in accordance with the weight gradients' average recent magnitudes. The algorithm performs effectively on online and non-stationary issues (e.g., noisy) according to this.

Adam is aware of the advantages of RMSProp and AdaGrad. Adam uses the average of the second moments of the gradients in addition to the average of the first moments, which is how RMSProp adjusts the parameter learning rates (the uncentered variance). The constants beta1 and beta2 regulate the decay rates of these moving averages, and the method specifically creates an exponential moving average of the gradient and the squared gradient. Moment estimations are biased towards zero as a result of the starting value of the moving averages and beta1 and beta2 values that are near 1.0 (recommended). By first computing the biased estimates and then the bias-corrected estimates, this bias is eliminated.

Adam is a well-known deep learning method because it produces good results quickly. Empirical findings show that Adam performs admirably in real-world applications and compares favorably to other stochastic optimization techniques. Adam was practically proven in the original study to show that convergence satisfies the predictions of the theoretical approach. It can tackle real-world deep learning issues effectively using huge models and datasets [119]. Adam is now advised as the default approach to employ in practice and frequently performs somewhat better than RMSProp. Adam’s configuration parameters are alpha, beta1, beta2, and epsilon. alpha, also known as the step size or learning rate, is the frequency with which weights are updated. Prior to the rate being modified, initial learning proceeds more quickly for larger numbers. Smaller values drastically reduce learning during training. beta1 is an estimate of the initial moment's exponential decay rate, and beta2 is the rate of exponential decline for the estimations from the second instant. epsilon is extremely low in order to avoid any division by zero during implementation.

In this work, the Adam optimizer algorithm is used for the CNN, LSTM, and CNN-LSTM models. The Adam study [119] proposes that alpha=0.001, beta1=0.9, beta2=0.999, and epsilon=10E-8 are good default parameters for the evaluated machine learning issues, hence, these parameters will be used in this work.

AdaMax

It is possible to think of Adam as updating weights in a manner that is inversely proportional to the scaled L2 norm (squared) of earlier gradients. This is expanded by AdaMax to include the alleged infinite norm (max)

of prior gradients. The infinite norm computation displays a stable behavior [120]. In the AdaMax optimization algorithm, the computation of the infinite norm and the exponential moving average is performed. The exponential decay rates of the first moment and the exponentially weighted infinity norm are controlled by the beta parameters, beta1, and beta2. The epsilon parameter of the Adam optimization algorithm is also used in AdaMax.

The base learning rate plays a crucial role in determining the steps of the AdaMax optimizer with respect to the loss. It determines the initial learning rate of the optimizer before any additional techniques, such as momentum or damping, are employed. Finding the optimal balance between the two learning rates is essential to achieve a minimum in a reasonable amount of time while avoiding the risk of overshooting the minimum. A commonly used approach for determining the appropriate base learning rate is to start with a value of 0.1 and gradually decrease it by a factor of 10 until the results no longer improve [121]. In order to effectively model the relationship between inputs and outputs, it is important to ensure that the neural network has a suitable level of complexity. This can be achieved by selecting weights that differ significantly from one another. However, excessive complexity should be avoided to prevent overfitting and ensure the ability to generalize. Another parameter, weight decay, is a method to mitigate overfitting by adding a term to the loss function that penalizes the distance between weights, typically in the form of L2 normalization. This encourages the optimizer to reduce both the loss and the separation between weights.

In this work, the AdaMax optimization algorithm is used only for the ANN model, which proved to be an improvement from using the Adam optimization algorithm during the experimentation phase. The configuration parameters of AdaMax are set to their default values as follows: learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, and weight_decay=None [122].

Table 8 summarizes the models used in this work for the Human Activity Recognition task and the hyperparameters of each model.

Table 8: HAR models hyperparameters

Model	Hyperparameter		Value
ANN	# of dense layers		2
	Neurons	Dense Layer 1	128
		Dense Layer 2	5
	Dropout		0.2
	Optimization algorithm		AdaMax
	Learning rate		0.0001
	Batch size		64
	Training epochs		42 (ES)
CNN	# of convolutional layers		2
	Kernel size	Conv1D Layer 1	3
		Conv1D Layer 2	3
	Feature maps	Conv1D Layer 1	64

		Conv1D Layer 2	64
	Dropout		0.5
	Pooling size		2
	Neurons	Dense Layer 1	128
		Dense Layer 2	5
	Optimization algorithm		Adam
	Learning rate		0.0001
	Batch size		64
	Training epochs		33 (ES)
LSTM	# of LSTM layers		2
	Neurons	LSTM Layer 1	128
		LSTM Layer 2	32
		Dense Layer 1	64
		Dense Layer 2	5
	Dropout	LSTM Layer 1	0.2
		LSTM Layer 2	0.2
	Optimization algorithm		Adam
	Learning rate		0.0001
	Batch size		64
	Training epochs		102 (ES)
CNN-LSTM	# of convolutional layers		1
	# of LSTM layers		1
	Kernel size		3
	Feature maps		64
	Pooling size		2
	Dropout	Dropout Layer	0.5
		LSTM Layer	0.2
	Neurons	LSTM Layer	64
		Dense Layer 1	128
		Dense Layer 2	5
	Optimization algorithm		Adam
	Learning rate		0.0001
	Batch size		64
	Training epochs		72 (ES)

B. Indoor Positioning System

i. Hardware Components

a) ESP32

The ESP32 (Figure 34) is a single-chip Wi-Fi and Bluetooth combo device that has been engineered with low-power 40 nm TSMC technology [123]. Its design is aimed at delivering the best possible power and RF

performance, with a focus on robustness, versatility, and reliability in a wide range of applications and power conditions. The ESP32 has been specifically developed for mobile, wearable electronics, and Internet of Things (IoT) applications. It boasts advanced low-power features, including fine-grained clock gating, multiple power modes, and dynamic power scaling [123]. Additionally, the output of the power amplifier is adjustable, which enhances the trade-off between communication range, data rate, and power consumption.

Figure 34: ESP32 MCU

The ESP32 is a highly integrated solution for Wi-Fi and Bluetooth IoT applications. The chip includes an antenna switch, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules, which results in a minimal Printed Circuit Board (PCB) area [123]. The ESP32 uses CMOS technology for its single-chip, fully integrated radio, and baseband, and it also integrates advanced calibration circuits that can compensate for external circuit imperfections or adapt to changing external conditions. The ESP32 implements a TCP/IP protocol and a complete 802.11 b/g/n Wi-Fi MAC protocol [123]. Power management is designed to require minimal host interaction, reducing the active-duty period. The ESP32 chip also integrates a Bluetooth link controller and Bluetooth baseband, which are responsible for executing baseband protocols and other low-level link routines, such as modulation/demodulation, packet processing, bitstream processing, frequency hopping, and more. The following are some of the most important specifications of the ESP32 chip [123]:

- Core: ESP32-D0WDR2-V3 Dual Core
- Core clock maximum frequency: 240 MHZ
- Memory: 4 MB Flash, 448 KB ROM, 520 KB SRAM
- Operating Temperature: -40º Celsius ~ +85º Celsius
- Dimensions (mm): 35.6 x 34.4 x 3.5
- Antenna: On-board dual PCB antennas
- Bluetooth: Compliant with Bluetooth v4.2 BR/EDR and Bluetooth LE specifications, +9 dBm transmitting power, Simultaneous advertising, and scanning
- Wi-Fi: 802.11b/g/n, 802.11n (2.4 GHz), up to 150 Mbps

Bluetooth Low Energy (BLE) is a wireless communication protocol designed for low-power devices. It operates in the 2.4 GHz ISM band and uses a technique called frequency hopping to reduce interference with other wireless devices. BLE devices can be in one of two roles: a central device or a peripheral device. Central devices, such as smartphones, initiate connections with peripheral devices, like a fitness tracker. Once connected, the central device can read and write data to the peripheral device's attributes, which are similar to variables. BLE devices can also broadcast advertising packets, which contain information such as the device name and available services. This allows other BLE devices to discover and connect to the advertising device without the need for prior pairing. BLE consumes very less power, hence it's commonly used in IoT devices and wearables which need to run for a long time on a small battery. The following are the specifications of the BLE communication protocol [124]:

- Connection speed: 2 Mbps
- Network data rate: 2 Mbps
- Frequency: 2.4 GHz
- Compatibility: Bluetooth 5 is fully compatible with Bluetooth 4.x. while supporting the whole main functionality of previous protocol versions (1.x., 2.x., 3.x.)
- Message Capacity: 255 bytes which give 200-230 bytes for actual data payload
- Indoor Range: 40-50 meters with obstacles, 200-2500 meters in line of sight

According to Espressif Systems, ESP32 draws a current of 30-68 mA during modem sleep, 95-100 mA during receiving BT/BLE, and 180 mA during Wi-Fi 802.11n transmission [123]. The average power consumption has been calculated to be approximately 60-80 mAh since the BLE and Wi-Fi are only receiving/transmitting for 10 seconds every 30-second window.

b) BLE Beacon

Bluetooth Low Energy (BLE) is a wireless technology that is used for short-range data transmission. As the name suggests, BLE is designed to consume less energy and cost less than regular Bluetooth while having a similar communication range. BLE communication is done by sending small packets of data at regular intervals through radio waves. The iBeacon standard dictates that these packets are broadcasted every 100ms. BLE beacons (Figure 35) are the hardware of BLE devices that powers the broadcast signal transmission. Devices that are close to a BLE beacon may conduct activities thanks to beacons, which are the hardware that makes this possible. The movement and position of a BLE device may be traced using BLE beacons. Also, make it possible for location-based activities to be started when a BLE device enters a certain place. The use of BLE beacons enables BLE tracking. Advantages of iBeacon over GPS include that iBeacon is more accurate, faster, consumes less power, and can provide indoor navigation [125].

Figure 35: Bluecharm BLE beacon

Using radio waves, Bluetooth beacons communicate by sending data packets that are picked up by a suitable receiving device. These data packets might be standalone or function as triggers for other events like push notifications, app activities, and prompts on the receiving device. BLE operates on a separate set of channels while using the same frequency range as Bluetooth Classic (2.400-2.4835 GHz ISM band) [126]. BLE has three main ad channels, which speeds up device connections and cuts down on listening device scanning time. BLE employs direct sequence spread spectrum or frequency hopping using digital modulation methods to combat concerns with narrowband interference. The theoretical maximum radius of a Bluetooth beacon is less than 100m. Additionally, the delay from a disconnected state might be up to 6ms. The beacon's real range and reaction time depend on the operation it has been designed to do. Most BLE uses are short-range, including indoor wayfinding (using the standard 1M PHY). Most Bluetooth beacons have a reliable transmission range of up to 30 meters when there are no physical barriers in their path. The typical operational range depends on the transmit power, and the battery usage rises as the range does. Table 9 shows the battery life (in months) for each of the transmission power and interval configurations.

Table 9: Battery life in months for each of the transmission power and interval configurations

Battery Life in Months - Effect of Interval (ms) and TX (dBm) settings (Based on beacon being configured as non-connectable)								
	+4 dBm	0 dBm	-4 dBm	-8 dBm	-12 dBm	-16 dBm	-20 dBm	-40 dBm
100	1.2	1.6	1.8	1.8	1.9	1.9	1.9	2.0
200	2.5	3.2	3.5	3.7	3.7	3.8	3.9	4.0
300	3.7	4.8	5.3	5.5	5.6	5.7	5.8	5.9
400	5.0	6.4	7.0	7.3	7.5	7.6	7.8	7.9
500	6.2	8.0	8.8	9.2	9.3	9.5	9.7	9.9
600	7.5	9.6	10.6	11.0	11.2	11.4	11.7	11.9
700	8.7	11.2	12.3	12.8	13.1	13.3	13.6	13.9
800	10.0	12.8	14.1	14.6	14.9	15.2	15.5	15.9
900	11.2	14.4	15.8	16.5	16.8	17.1	17.5	17.8
1000	12.5	16.0	17.6	18.3	18.7	19.0	19.4	19.8
1500	18.7	24.0	26.4	27.5	28.0	28.6	29.1	29.7
2000	25.0	32.0	35.2	36.6	37.3	38.1	38.8	39.6

Data packets are sent using beacons. For iBeacons and Eddystone, these packets include somewhat different elements. However, beacons typically just provide an ID and some location data, together with a component that shows the beacon's state (e.g., temperature, battery status, etc.) iBeacon, AltBeacon, URIBeacon, and Eddystone are just a few of the several BLE beacon protocols in use. Apple uses the BLE beacon system

known as iBeacon to facilitate payments and provide in-store and on-site deals. Radius Networks' open-source alternative to iBeacon is called AltBeacon. The BLE beacon protocol used in this project is iBeacon. Apple created the iBeacon protocol, which enables devices to look for beacon signals within range and display the content that matches. Through BLE networks, Beaconstac SDK makes it simple to implement location analytics and proximity marketing. NearBee SDK may be used for iOS to effortlessly initiate rich beacon notifications [125].

An iBeacon-enabled beacon emits a serial ID number. Each data packet in an iBeacon's standardized BLE advertising format provides the following 4 pieces of information (Figure 36) [127]:

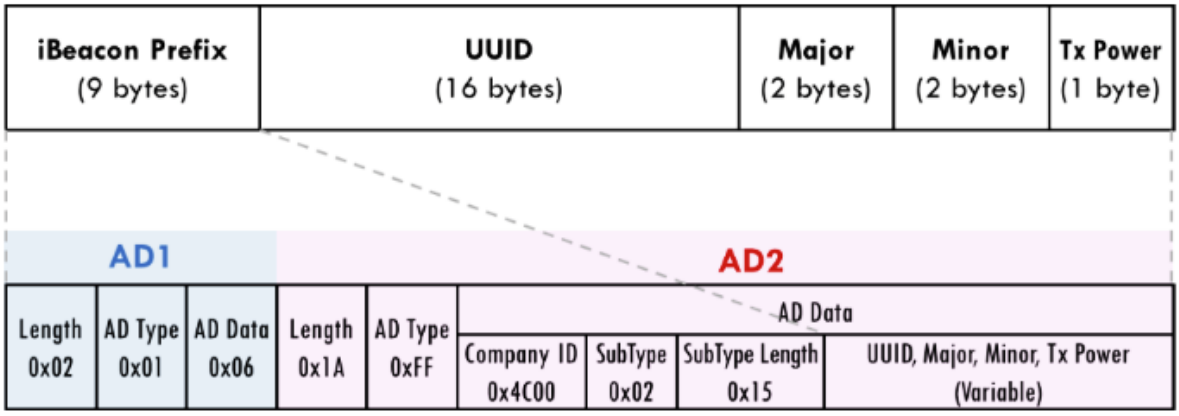

Figure 36: Packet format of iBeacon

1) Unique Universal Identifier (UUID): A beacon's general information. For instance, the name of the person or company that owns the beacon.
2) Major: The geographical data of the beacon. For instance, this beacon is used in the aisles of store #9.
3) Minor: More specific or minute details. For instance, a specific region of the major's floor.
4) Tx power: is the measure of the signal's amplitude at a precise distance of 1 meter from the device. Tx can be used to calculate device proximity from the beacon.

Each ESP32 device is configured to scan for the BLE beacon using its MAC address and read the RSSI value of this BLE beacon. RSSI stands for Received Signal Strength Indicator. It is a measure of the strength of the signal being emitted by a beacon, as received by a device such as a smartphone or an ESP32 device. The strength of the signal is affected by the distance between the device and the beacon and the power at which the beacon is broadcasting. At maximum broadcasting power, the RSSI can range from -26 (when the device is very close to the beacon) to -100 (when the device is about 40-50 meters away from the beacon). RSSI is used to estimate the distance between the device and the beacon by comparing it to another value defined by the iBeacon standard called Measured Power. However, due to external factors that can impact radio waves such as absorption, interference, or diffraction, the RSSI tends to fluctuate. The further away the device is from the beacon, the more unstable the RSSI becomes. Measured Power is a fixed value that is calibrated at the factory and it represents the expected Received Signal Strength Indicator (RSSI) when a device is one meter away

from the beacon. By combining the read-only constant ‘Measured Power’ with the RSSI, we can estimate the distance between the device and the beacon.

In this project, the Bluecharm BLE beacon, shown in Figure 35, is used. The following are its specifications:

- Transmission: iBeacon, Eddystone URL, and/or Eddystone UID format. May be configured to broadcast battery strength.
- MAC address: DD:34:02:06:76:84
- BLE version: 4.0 and 5.0
- Battery size: CR2032 Lithium Coin Battery (included)
- Battery life: 16 months @ 1 second interval & 0dBm
- Includes 3M double-back tape for affixing to a surface
- Size: 36mm x 36mm x 5.75mm
- Weight: 0.28 ounces including battery
- Waterproofing: no
- Password: yes
- Wide TX Power range: select from -40, -20, -16, -12, -8, -4, 0, +4dBm
- Interval: adjustable from 100-10,000 milliseconds including optimal precise numbers
- On/Off switch: yes
- Motion Sensor: yes
- Chip: nRF52810
- Transmit distance: 0.2-90 meters (BLE 5.0); 0.2-50 meters (BLE 4.0)

Figure 37 shows the configuration of the BLE beacon used in this work. It is important to note that with the interval set to 600ms and the TX power set to +4dBm, the BLE beacon’s battery is expected to last for 7.5 months, according to Table 9, which can then be replaced with a new one.

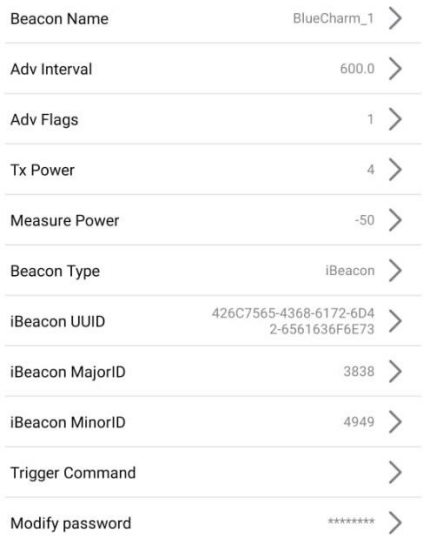

Figure 37: Configuration of Bluecharm BLE beacon

ii. Data Collection

Three ESP32 devices have been placed in separate rooms with the aim of scanning the BLE beacon and recording the received signal strength indicator (RSSI) value. These ESP32 devices are capable of detecting the BLE beacon and determining its proximity based on the strength of the received signal. The dimension and area of each room, also shown in Figure 38, are as follows:

- Living room:
    - Length: 590 cm
    - Width: 296 cm
    - Area: 17.47 m^2
- Bedroom:
    - Length: 350 cm
    - Width: 268 cm
    - Area: 9.36 m^2
- Bathroom:
    - Length: 312 cm
    - Width: 268 cm
    - Area: 8.35 m^2

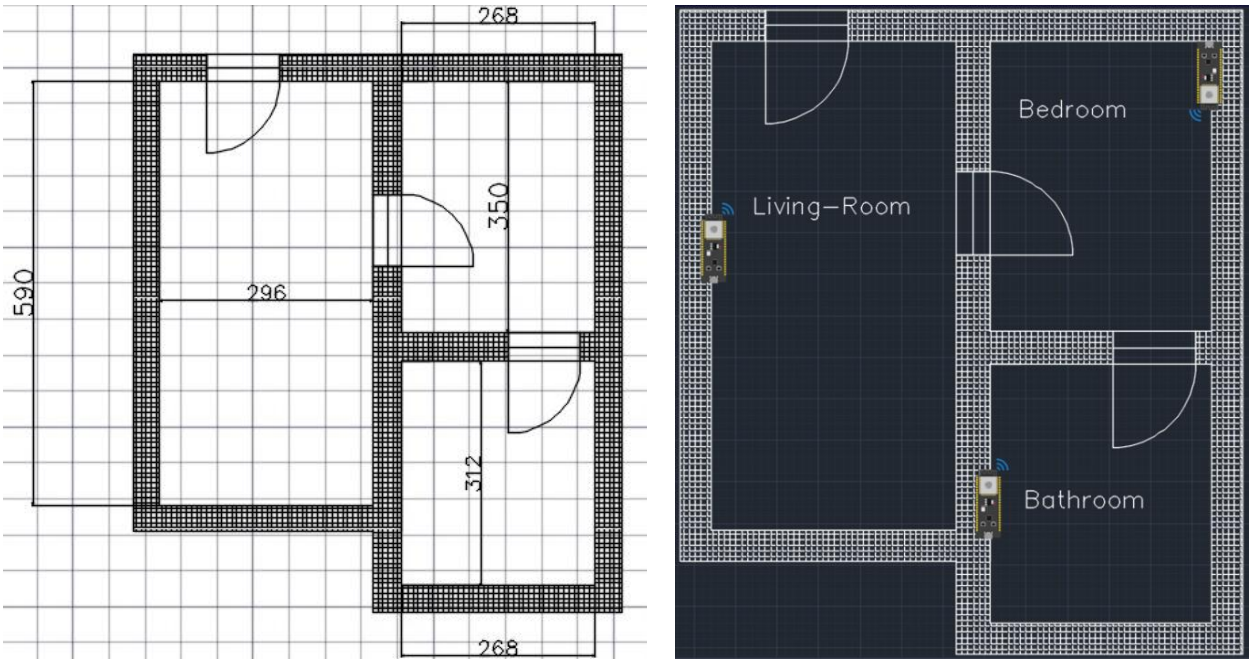

Figure 38: Floor plan of the three partitions (rooms) and the locations of the ESP32s

The ESP32 devices are programmed to scan for the BLE beacon using its MAC address and record the RSSI value in Firebase Realtime Database (RTDB) every 30 seconds. During the training phase, the BLE beacon is placed in different locations in each of the three rooms and the RSSI values from the three ESP32s are stored.

To achieve this, the ESP32 program is coded to perform a number of tasks. To begin, the necessary libraries are imported. The WiFi.h library is utilized to connect to a Wi-Fi network and establish an internet connection, while the Firebase_ESP_Client.h library enables communication between the board and Firebase (Figure 39). The necessary BLE libraries: BLEDevice.h, BLEUtils.h, BLEScan.h, and BLEAdvertisedDevice.h are also imported. Subsequently, the Wi-Fi network's SSID and password, Firebase API key, database path, database URL, and BLE beacon's MAC address are specified.

In the setup() function, the board is connected to the network using the provided SSID and password of the Wi-Fi network. The API key is assigned to the Web API Key of the Firebase project, and the database URL is to the Firebase RTDB URL. Next, an anonymous sign-in is done using the signUp() method with the last two arguments left empty. Upon successful sign-in, the signupOK variable is altered to indicate a successful outcome. In the loop() function, data is periodically transmitted to the database, given that the sign-in was successful and all necessary elements are in place.

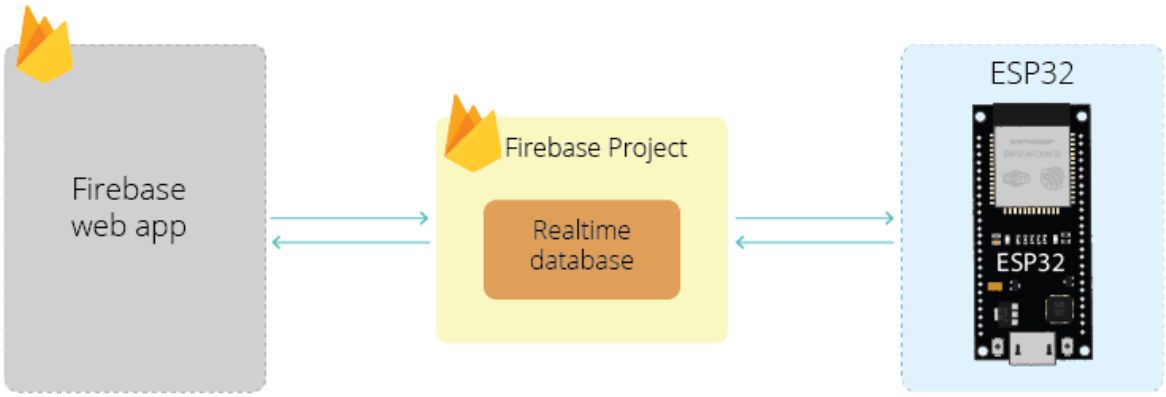

Figure 39: ESP32 communication with Firebase RTDB

The ESP32 uses a Bluetooth Low Energy (BLE) module to scan for nearby devices that are broadcasting BLE signals. This module allows the ESP32 to search for BLE devices and retrieve information about them, including the device's MAC address and the strength of the received signal (RSSI). The code implements the ESP32 as a Bluetooth Low Energy (BLE) device and conducts scans for nearby devices periodically at an interval of 10 seconds. If the BLE beacon is found, that is if the MAC address of one of the found devices matches the MAC address of the BLE beacon, the RSSI value of this device is stored along with the timestamp in the Firebase RTDB. In case the BLE beacon is not among the found devices at an instant, a default RSSI value of '-110' will be stored.

The setJSON function is used to store the acquired RSSI and timestamp information as JSON data at a designated node for this specific ESP32 device within the Firebase RTDB. The function returns a Boolean value that indicates the success of the operation. This is dependent on the fulfillment of two conditions: that the server returns an HTTP status code of 200, and that there is consistency in the data types between the request and response. Figure 40 shows the ESP32 output throughout each step using Arduino Serial Monitor. It is worth noting that the average bandwidth consumption of the ESP32 is approximately 3-3.5 megabytes per hour.

```
17:06:01.902 -> Connecting to Wi-Fi.......
17:06:04.013 -> Connected with IP: 172.20.10.5
17:06:04.013 ->
17:06:06.634 -> Authentication success.
17:06:06.634 -> Scanning...
17:06:06.972 -> time: 1669129567
17:06:08.993 -> Advertised Device: Name: BlueCharm_1, Address: dd:34:02:06:76:84, manufacturer data: 4c000215426c7565436861726d426561636f6e730efe1355ce RSSI: -80
17:06:18.077 -> Devices found: 1
17:06:18.077 -> Scan done!
17:06:18.405 -> Set json... Done
```

Figure 40: ESP32 output using Arduino Serial Monitor

For each of the three rooms, the RSSI values from each of the ESP32s are retrieved from the Firebase RTDB using a Python script and stored as a CSV file where each row represents the RSSI values recorded in one instant. The dataset has three columns, each column represents the RSSI values recorded by one of the ESP32s named 'esp1', 'esp2', and 'esp3', with the label of this room, added as a column to later be fed as training data to the machine learning models, as shown in Figure 41.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 842 entries, 0 to 841
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   esp1      842 non-null    int64
 1   esp2      842 non-null    int64
 2   esp3      842 non-null    int64
 3   location  842 non-null    object
dtypes: int64(3), object(1)
memory usage: 26.4+ KB
```

Figure 41: Dataset of the beacon's RSSI values from three ESP32s

iii. Data Pre-processing

Upon inspection, the dataset is found to contain no null values. The records in which the three ESP32s recorded an RSSI value of '-110' were removed from the dataset, as such records indicate that the BLE beacon was either turned off or out of range. Next, the same data preprocessing techniques discussed earlier in the HAR Data Pre-processing section are applied. Data standardization using StandardScaler is applied to the dataset of RSSI values. Then, for applying to new data using the "transform" function, the StandardScaler object containing the scaling parameters is stored as a .bin file.

Table 10: Number of samples for each room

Location	Number of samples
Living Room	313
Bathroom	287
Bedroom	242

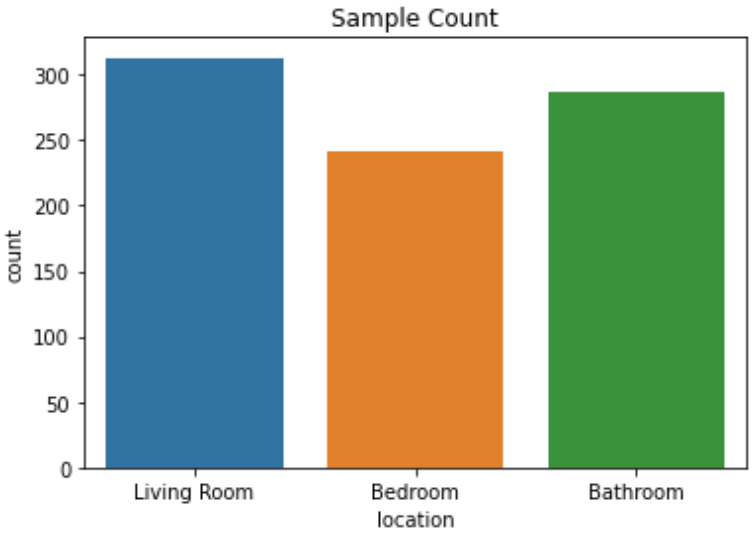

Figure 42: Number of samples for each room

As for data splitting, the dataset is split using the 80:20 random subsampling ratio, where 80% of the data will be randomly selected to be used for training the machine learning models, and the remaining 20% used for testing the performance of the models. Table 10 shows the number of samples collected from each room. The

graph in Figure 42 is a visualization of the table, showing the distribution of the samples across the three classes.

iv. Machine Learning Models

a) k-Nearest Neighbor (kNN)

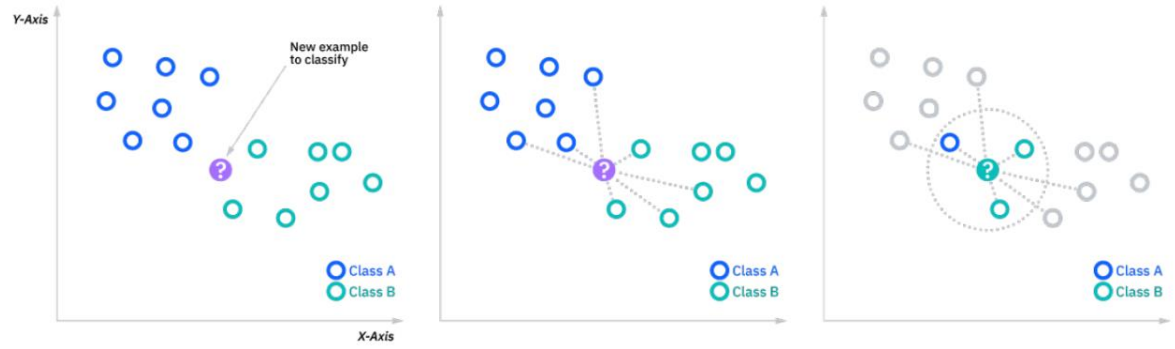

Figure 43: kNN algorithm

kNN classifier categorizes unlabeled observations by putting them into the same category as the labeled samples that are the most comparable to them. For both the training dataset and the test dataset, observational characteristics are collected [128]. A majority vote is utilized to determine the class label for classification problems; the label that is most frequently displayed around a particular data point is used. It is important to note that the KNN method belongs to a group of "lazy learning" models, which just store training datasets rather than going through a training phase. This also implies that all computing happens at the time of classification or prediction. It is also known as an instance-based or memory-based learning approach since it stores all of its training data entirely in memory [129].

The distance between the query point and the other data points must be determined in order to determine which data points are closest to a specific query point. These distance measurements aids in the creation of decision borders, which divide query points into several zones. The most widely used distance metric is called the Euclidean distance, and it can only be employed with real-valued vectors. The straight line between the query location and the other point being measured is calculated using the formula [129]:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \quad (13)$$

The Minkowski distance is a generalized form of the Euclidean and Manhattan distance metrics, characterized by a tunable parameter, p, in the formula [129]:

$$\text{Minkowski Distance } = \left(\sum_{i=1}^{n} |x_i - y_i|\right)^{1/p} \tag{14}$$

The value used in this work is $p = 2$, which results in the Euclidean distance.

b) Naïve Bayes (NB)

Naïve Bayes is one of the simplest algorithms to apply to a dataset. As the name suggests, the algorithm assumes that all variables in the dataset are "naïve," meaning they are not correlated with each other. Naïve Bayes is a very popular classification algorithm used primarily to maintain baseline accuracy in data sets. The concept behind the algorithm lies in the Bayes Theorem that is mathematically expressed, given class variable y and dependent feature vector x_1 through x_n, as [130]:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)} \tag{15}$$

where $P(x_1, \dots, x_n)$ and $P(y)$ are the probabilities of $x_1, \dots, x_n$ and y independently of each other; $P(y|x_1, \dots, x_n)$ is the conditional probability that event y will occur given that $x_1, \dots, x_n$ have occurred, also called the posterior probability; and $P(x_1, \dots, x_n \mid y)$ is the conditional probability that events $x_1, \dots, x_n$ will occur given that y has occurred.

We can use Bayes' theorem to find the probability that y will occur given that $x_1, \dots, x_n$ has occurred. where $x_1, \dots, x_n$ is evidence and y is the hypothesis. The assumption here is that the predictors/features are independent. The presence of one particular trait does not affect other traits. Therefore, it is called naïve. In the process of modeling, the optimization of finding the hypothesis with the highest posterior probability is referred to as the Maximum A Posteriori (MAP) method [131].

The Gaussian Naïve Bayes algorithm for classification used in this work is implemented using sklearn's GaussianNB function. This algorithm operates under the assumption that the likelihood of the features follows a Gaussian distribution:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{16}$$

Where the parameters σ_y and μ_y are estimated using maximum likelihood [130].

c) Support Vector Machine (SVM)

Support Vector Machine, or SVM, is used to solve Classification and Regression problems. However, it is largely employed in Machine Learning Classification issues. The SVM algorithm's objective is to establish the best line or decision boundary that can divide n-dimensional space into classes, allowing quick classification of new data points in the future. A hyperplane is the optimal decision boundary. SVM selects the extreme vectors and points that aid in the creation of the hyperplane. Support vectors refer to these extreme cases as shown in Figure 44 [132].

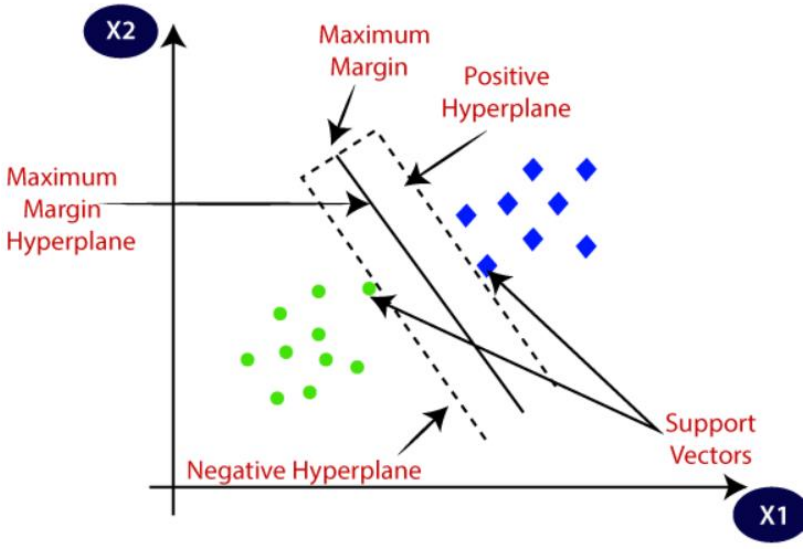

Figure 44: SVM hyperplane and support vectors

By determining the optimum hyperplane and optimizing the distance between points, it divides the data into many groups. A particularly effective tool for navigating high-dimensional spaces is the kernel function. The kernel function is a mathematical operation that transforms a low-dimensional input space into a higher-dimensional space (Figure 45). Kernel functions offer the ability to perform linear discriminants on nonlinear manifolds, which may result in more accuracy and robustness than simple linear models. This is accomplished by mapping the data into a new feature space and the data will be linearly separable in this space. This implies that a hyperplane that separates the data can be found using an SVM [133].

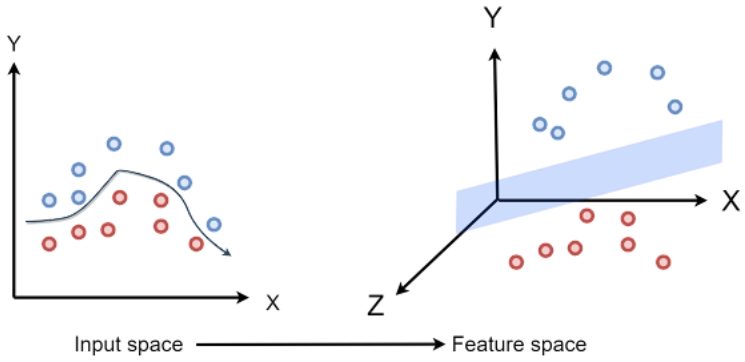

Figure 45: Kernel functions transform input space into a higher-dimensional feature space

Two important parameters of support vector machines are C and gamma. For each incorrectly classified data point, the C parameter applies a penalty. If C is low, choosing a decision boundary with a high margin comes at the expense of more misclassifications because the penalty for incorrectly classified points is low. SVM attempts to reduce the number of incorrectly classified examples due to the high penalty when C is big, which leads to a decision boundary with a narrower margin. Not all examples of misclassification receive the same penalty. It is directly proportional to the distance to the decision boundary. As for the Gamma parameter, low gamma values suggest a wide similarity radius, which causes more points to be grouped together. To be included in the same group (or class) with high gamma values, the points must be quite close to one another. As a result, models with very high gamma values are more likely to overfit [134]. It is worth noting that the C parameter is applicable to all SVM kernels, whereas the Gamma parameter is only applicable to the RBF, Polynomial, and Sigmoid kernels.

The SVC, NuSVC, and LinearSVC algorithms are all capable of performing binary and multi-class classification on a dataset. SVC and NuSVC are similar in nature, but accept slightly different parameters and have distinct mathematical formulations [135].

LinearSVC

The term "linearly separable data" refers to data that can be divided into two groups using only a single straight line [136]. LinearSVC is a faster implementation of Support Vector Classification specifically for linear kernels and does not accept the kernel parameter as it is assumed to be linear. In cases where the training data is linearly separable, two parallel hyperplanes can be selected to separate the two classes of data with the greatest distance between them. The area bounded by these two hyperplanes is referred to as the "margin" and the maximum-margin hyperplane is the hyperplane that lies in the middle of them. When the data is not linearly separable, the hinge loss function is utilized in the SVM algorithm. The linear kernel is defined as $\langle x, x' \rangle$ [135].

Polynomial Kernel

An SVM kernel known as a polynomial kernel maps the data into a higher-dimensional space using a polynomial function. This is accomplished by taking the dot product of the polynomial function in the new space and the original space's data points. A polynomial function is used to map the data into a higher-dimensional space in a polynomial kernel for SVM. The polynomial function in the new space is then taken as the dot product of the data points in the original space. In SVM classification applications when the data cannot be separated linearly, the polynomial kernel is frequently utilized. The polynomial kernel can occasionally locate a hyperplane that divides the classes by mapping the data into a higher-dimensional space. For d-degree polynomials, the polynomial kernel is defined as [135]:

$$(\gamma \langle x, x' \rangle + r)^d \tag{17}$$

where d is specified by parameter degree, r by coef0 [137].

RBF Kernel

The Radial Basis Function (RBF) kernel is commonly utilized for Support Vector Machine (SVM) training. When using this kernel, two parameters need to be carefully considered: C and gamma. C, a parameter that is applicable to all SVM kernels, balances the trade-off between the misclassification of training samples and the simplicity of the decision boundary. A low value of C results in a smooth decision boundary, while a high value of C seeks to classify all training samples accurately. On the other hand, gamma determines the impact a single training sample has on the model. The higher the value of gamma, the closer other samples must be to be affected by it. The RBF kernel is defined as [135]:

$$\exp\left(-\gamma\|x - x'\|^2\right) \tag{18}$$

where γ is specified by parameter gamma and has to be greater than 0 [137].

NuSVC

The ν-SVC formulation [138] is a reparameterization of the C-SVC and is mathematically equivalent. A parameter ν is introduced instead of C, which controls the number of support vectors and margin errors. ν serves as an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors, thus $\nu \in (0,1]$. A margin error corresponds to a sample that is either misclassified or correctly classified but does not fall beyond the margin boundary.

Figure 46 shows an example of the four SVM classifiers applied to the Iris dataset [139].

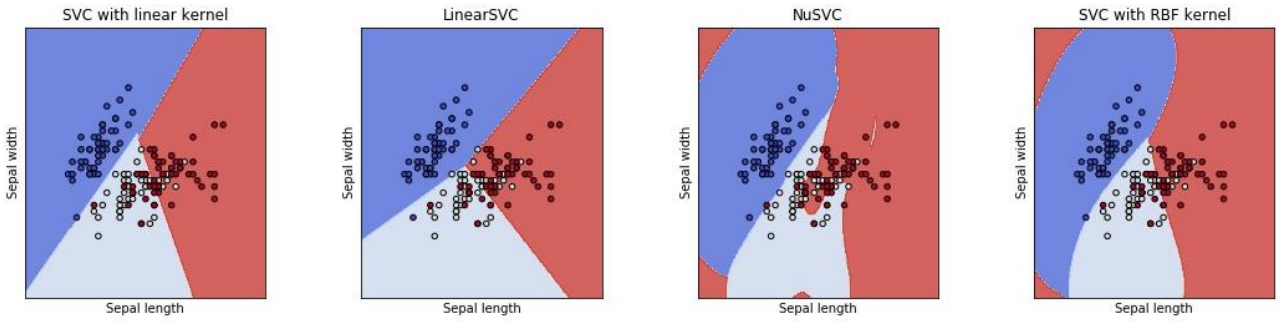

Figure 46: Decision boundaries of the four SVM classifiers applied to the Iris dataset

d) Decision Tree (DT)

A decision tree, as shown in Figure 47 is like a flowchart where each inner node represents a test of a feature, and each leaf node represents a class label (the decision made after all features have been computed) and a branch. The path from the root to the leaf represents a classification rule. The three different sorts of nodes in a decision tree are illustrated in Figure 47 [140]. A decision tree consists of three main components: decision nodes, leaf nodes, and a root node. A training dataset is divided into branches by a decision tree algorithm, which then separates those branches further. This process keeps going until a leaf node is reached. It is

impossible to further separate the leaf node. The attributes that are utilized to predict the outcome are represented by the nodes in the decision tree. Links to the leaves are provided by decision nodes.

Decision trees are built through an algorithmic approach that identifies how to split a data set based on various criteria. It is one of the most widely used practical methods of supervised learning. Decision trees are a nonparametric supervised learning technique used for both classification and regression tasks. A tree model in which the target variable can take on a discrete set of values is called a classification tree.

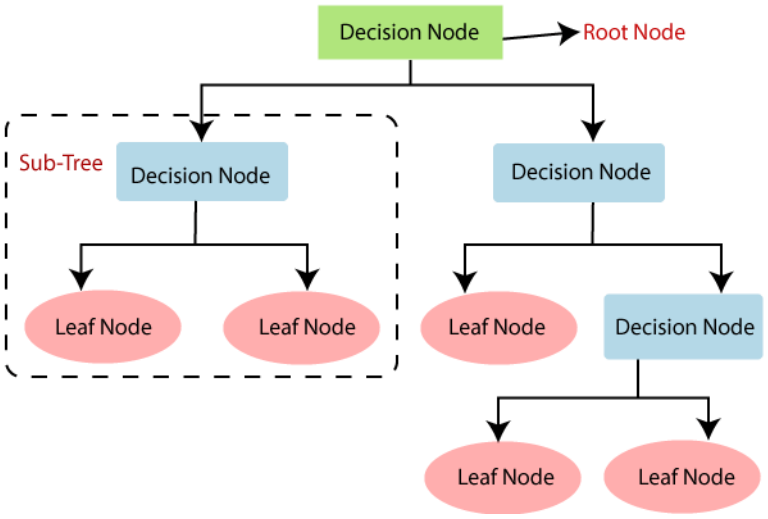

Figure 47: Decision tree algorithm

Information gain is a crucial component in determining which feature to split on at each step during the tree-building process. The goal is to create a simple tree structure; therefore, it is imperative to choose the split that results in the purest child nodes. A widely utilized measure of purity is known as information. The information value quantifies the extent to which a feature provides information about the class for each node of the tree. The split that yields the highest information gain is selected as the initial split and the process continues until all child nodes are pure or until the information gain reaches zero.

In information theory, entropy measures the uncertainty or impurity within a group of observations. This metric plays a crucial role in determining how a decision tree splits data. A clearer depiction of set purity can be seen in the accompanying image. For a dataset containing N classes, entropy can be calculated using the formula [141]:

$$E = -\sum_{i=1}^{N} p_i log_2 p_i \quad (19)$$

Information gain serves as a measure of the amount of information a feature contributes to a class. This metric is instrumental in determining the sequence of attributes in the nodes of a decision tree. The main node is

referred to as the parent node, while sub-nodes are designated as child nodes. Information gain enables us to evaluate the quality of node splitting in a decision tree. It can be calculated using the following formula [141]:

$$Gain = E_{parent} - E_{children} \quad (20)$$

The term "$Gain$" refers to information gain. "E_{parent}" represents the entropy of the parent node, while "$E_{children}$" is the average entropy of the child nodes. The greater the entropy removal, the higher the information gain, making a split more favorable. Hence, the attribute with the highest information gain from a set is selected as the parent or root node.

Decision Tree hyperparameters include the following [142]:

- min_samples_split – Minimum number of samples a node must possess before splitting.
- min_samples_leaf – Minimum number of samples a leaf node must possess.
- min_weight_fraction_leaf – Minimum fraction of the sum total of weights required to be at a leaf node.
- max_leaf_nodes – Maximum number of leaf nodes a decision tree can have.
- max_features – Maximum number of features that are taken into the account for splitting each node.

e) Random Forest Classifier (RFC)

A random forest is a machine-learning method for solving classification and regression issues. It makes use of ensemble learning, which is considered a method for solving complicated issues by combining a number of classifiers. It consists of many decision trees (Figure 48) which create a "forest" that is trained via bagging or bootstrap aggregation [143]. The accuracy of machine learning algorithms is increased by bagging, an ensemble meta-algorithm. Based on the predictions made by the decision trees, the mechanism of operation determines the outcome. It makes predictions by averaging or averaging out the results from different trees. The accuracy of the result grows as the number of trees increases. Additionally, it eradicates a decision tree algorithm's drawback. It improves precision and lowers dataset overfitting. Features of a Random Forest Algorithm include higher accuracy than a decision tree algorithm, effective ways of handling missing data, producing a reasonable prediction without hyper-parameter tuning, and solving the issue of overfitting in decision trees.

Decision trees are the building blocks of a random forest algorithm. For a more precise prediction, Random Forest produces numerous decision trees that are then combined. The Random Forest model is based on the idea that several uncorrelated models (the various decision trees) work significantly better together than they do separately. Each tree provides a classification or a "vote" when using Random Forest for categorization. The classification with the most "votes" is chosen by the forest. When performing regression with Random Forest, the forest selects the mean of all tree outputs [144]. The fact that there is little to no correlation between the component models—specifically, between the decision trees that make up the larger Random Forest

model—is crucial in this situation. The bulk of the decision trees will be accurate, shifting the overall result in the right direction even while some of them may make mistakes.

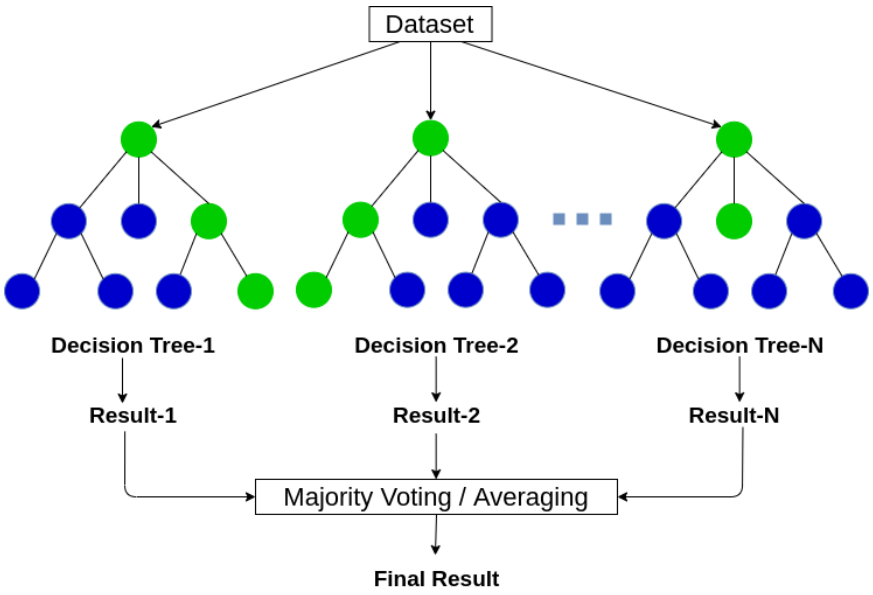

Figure 48: Random forest algorithm

In random forest, the hyperparameters are the number of trees, number of features, type of trees (such as GBM or M5), maximum depth of the tree, minimum number of samples required to be at a leaf node, and minimum number of samples required to split an internal node [145]. It is usually trained using the bagging method. The "bagging" approach utilizes a Bootstrap Aggregation ensemble machine learning technique. An ensemble technique combines predictions from various machine learning algorithms to provide predictions that are more precise than those from a single model [143].

f) Gradient Boosting Classifier (GBC)

The Gradient Boosting Classifier utilizes a differentiable loss function, which can either be a standardized function like a logarithmic loss for classification or squared errors for regression, or a custom-defined loss function. The classifier consists of two important components: a weak learner and an additive component. Decision trees act as the weak learner, and regression trees are used to produce real values. Over time, new trees are added to the model to correct errors in predictions while the existing trees remain unchanged as shown in Figure 49 [146]. The parameters of the trees are optimized through a process resembling gradient descent in order to reduce the error, and this process continues until a specified number of trees have been added or the loss has fallen below a predetermined threshold [147].

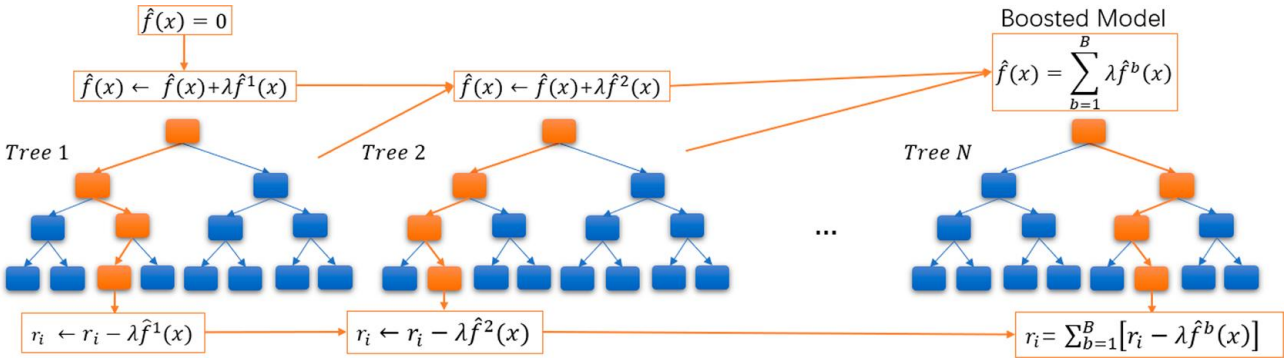

Figure 49: Gradient boosting tree algorithm

Gradient Tree Boosting, also known as Gradient Boosted Decision Trees (GBDT), is a method of boosting that can be applied to an arbitrary differentiable loss function [148]. GBDT is a widely utilized and accurate algorithm for solving both regression and classification problems. The Gradient Boosting Classifier is capable of handling both binary and multi-class classification problems. The number of weak learners, or regression trees, can be specified through the parameter n_estimators, while the size of each tree can be controlled through either the max_depth parameter for tree depth or the max_leaf_nodes parameter for the number of leaf nodes. Additionally, the learning_rate hyper-parameter, in the range (0.0, 1.0], plays a crucial role in controlling overfitting via shrinkage [149].

g) Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a method used in statistics and other fields to determine a linear combination of features that distinguish or separate various classes of objects or events. This linear combination can then be utilized as a linear classifier or, more frequently, as a means of reducing the number of dimensions before further classification. LDA is strongly associated with Analysis of Variance (ANOVA) and regression analysis, which both aim to express a dependent variable as a linear combination of other features or measurements. However, ANOVA employs categorical independent variables and a continuous dependent variable, while discriminant analysis uses continuous independent variables and a categorical dependent variable, namely the class label [150].

The original data matrix must be projected into a lower dimensions space in order to achieve the LDA technique's objective. There are three actions that need to be conducted to accomplish this. The first step is calculating the separability between distinct classes (the distance between the means of different classes), also known as the between-class variance or between-class matrix. The second step is calculating the within-class variance or within-class matrix, also known as the distance between the mean and the samples of each class. The third step is building a lower dimensional space that optimizes between-class variance and minimizes within-class variance [151].

h) Quadratic Discriminant Analysis (QDA)

The observations from each class are assumed to be normally distributed in this approach, which is similar to LDA, but it does not assume that each class has the same covariance matrix. LDA and QDA vary primarily in that QDA is a considerably more flexible classifier than LDA because LDA assumes that each class has a covariance matrix. This implies that it has low variance by default, which indicates that it will perform similarly across various training datasets. The disadvantage is that LDA might have a large bias if the presumption that all classes have the same covariance is false. Alternatively, QDA posits that every class has a unique covariance matrix [152]. Figure 50 shows decision boundaries for Linear Discriminant Analysis and Quadratic Discriminant Analysis [153].

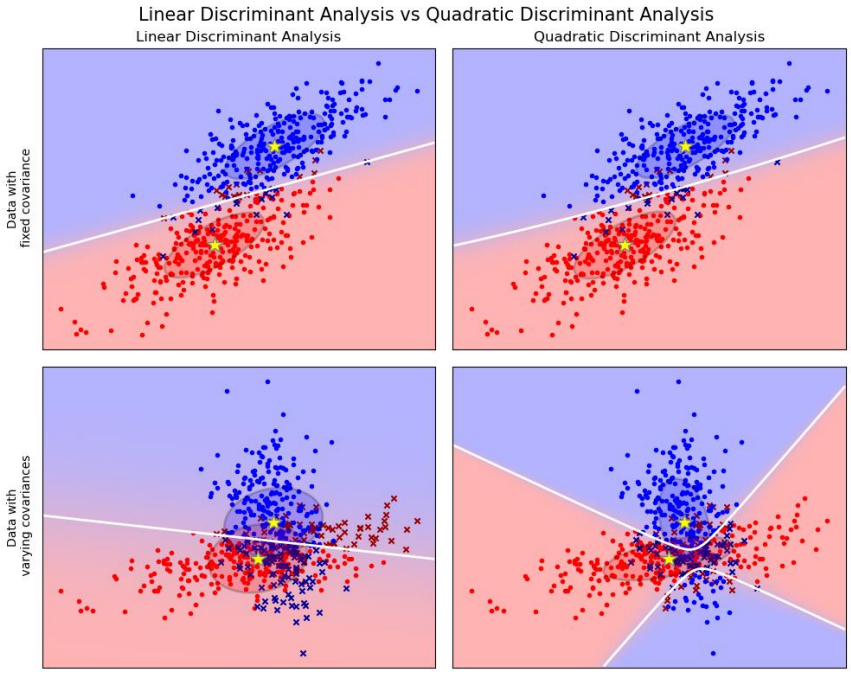

Figure 50: Comparison of LDA vs. QDA decision boundaries

i) Voting Classifier

A voting classifier is a type of machine learning estimator that develops a number of base models or estimators and makes predictions based on averaging their results. Voting for each estimator output can be integrated with the aggregating criteria. The Voting Classifier integrates multiple classifiers in order to produce a final prediction. This approach can be useful when multiple models perform similarly well, and their individual limitations can be balanced out by combining them. There are two categories of voting criteria: Hard Voting, where voting is calculated on the predicted output class; and Soft Voting, where voting is calculated on the predicted probability of the output class. In this work, hard voting (majority voting) is employed, where the class label assigned to a sample is the one that is predicted by the majority of the individual classifiers. In the event of a tie, the Voting Classifier determines the final class label using the ascending sort order of the individual classifiers [149]. The process is demonstrated in Figure 51 [154].

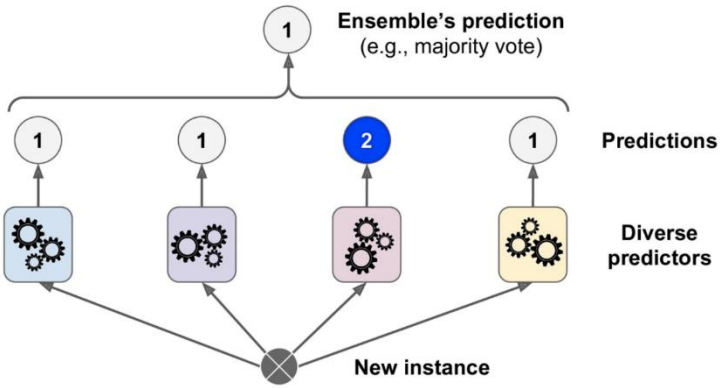

Figure 51: Voting classifier algorithm

v. Hyperparameters

Table 11 summarizes the machine learning models used in this work and the hyperparameters of each model.

Table 11: IPS classifiers hyperparameters

Model Name	Hyperparameters
KNeighborsClassifier	'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 27, 'p': 2, 'weights': 'uniform'
GaussianNB	'priors': None, 'var_smoothing': 1e-09
LinearSVC	'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True, 'intercept_scaling': 1, 'loss': 'squared_hinge', 'max_iter': 1000, 'multi_class': 'ovr', 'penalty': 'l2', 'random_state': None, 'tol': 0.0001, 'verbose': 0
SVC_poly	'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'poly', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False
SVC_rbf	'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False
NuSVC	'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'nu': 0.5,

	'probability': True, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False
DecisionTreeClassifier	'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': None, 'splitter': 'best'
RandomForestClassifier	'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False
GradientBoostingClassifier	'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'loss': 'deviance', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_iter_no_change': None, 'random_state': None, 'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False
LinearDiscriminantAnalysis	'covariance_estimator': None, 'n_components': None, 'priors': None, 'shrinkage': None, 'solver': 'svd', 'store_covariance': False, 'tol': 0.0001
QuadraticDiscriminantAnalysis	'priors': None, 'reg_param': 0.0, 'store_covariance': False, 'tol': 0.0001
VotingClassifier	'estimators': [('1', RandomForestClassifier()), ('2', GradientBoostingClassifier()), ('3', DecisionTreeClassifier()), ('4', LinearDiscriminantAnalysis()), ('5', QuadraticDiscriminantAnalysis())], 'flatten_transform': True, 'n_jobs': None, 'verbose': False, 'voting': 'hard', 'weights': [2, 2, 1, 1, 1]

C. Web App

i. Firebase Database

Firebase is a platform that offers a variety of tools and technologies to help developers create mobile and web applications. Its most well-known services include Firebase Analytics, Firebase Cloud Messaging, Firebase Auth, Realtime Database, Firebase Storage, and Firebase Hosting. The Firebase Realtime Database (RTDB) is a NoSQL cloud-based storage solution that allows for the real-time syncing of data across all connected clients. This means that any changes made to the database by one client will be immediately reflected on all other connected clients. The database is also cross-platform, meaning that it can be used with apps developed for different operating systems, such as Apple and Android, and the SDKs for these platforms automatically receive updates with the latest data. Firebase RTDB is a NoSQL database, optimized for fast data access, which makes it ideal for real-time applications with high traffic and millions of users. It also allows for offline functionality, so even when a device is disconnected, real-time events will still occur, providing a seamless user experience. When the device reconnects, the database automatically synchronizes any local data changes with any remote updates that occurred while offline, resolving any conflicts automatically. The database also has a robust security system, called Security Rules, which allows for flexible, expression-based rules to define data structure and access permissions. Integrating with Firebase Authentication allows developers to specify who has access to what data and how they can access it [155].

An API key is a unique identifier that is utilized to direct requests to a specific Firebase project when interacting with Firebase and Google services. Unlike the traditional method of using API keys to restrict access to backend resources, in Firebase services, API keys are not used for this purpose. Instead, Firebase Security Rules are used to control access to resources by users, and App Check is used to control access to resources by apps. One Firebase project can have multiple API keys; however, each API key can only be linked to one Firebase project. Firebase provides a default API key for all the apps within a project, regardless of the platform (iOS, Android, or web). For web applications, this key can be found in the Firebase config object under the field "apiKey". This key serves as a unique identifier for the Firebase project when utilizing Firebase or Google services. It is used to associate API requests with the project for tracking usage and billing purposes. Additionally, it can be used to access public data by passing the key as a query parameter in a REST API call. The Firebase SDK automatically looks for the API key in the Firebase config file/object when making API calls [156].

The Firebase Realtime Database can also be used as a REST endpoint by appending ".json" to the end of the URL and sending a request from an HTTPS client. HTTPS is required because Firebase only responds to encrypted traffic to ensure the safety of the data. A REST API, or Representational State Transfer API, is an application programming interface that follows the constraints of the REST architectural style and allows for communication with RESTful web services. When a client makes a request through a RESTful API, it retrieves

a representation of the state of the resource being requested. This representation can be in various formats such as JSON, HTML, XLT, Python, PHP, or plain text, with JSON being the most commonly used because it is language-agnostic and can be read by both humans and machines. In Firebase, to achieve the same result as the JavaScript push() method, a POST request can be issued. A successful request will be indicated by a 200 OK HTTP status code and the response will contain the child name of the new data specified in the POST request [157].

ii. Gmail API

The Gmail API is a way for developers to access and manage Gmail accounts programmatically. It can be used for various purposes such as reading and extracting email data, sending automated or programmatic messages, migrating email accounts, organizing, and sorting messages, and standardizing email signatures across an organization. The API allows for two ways to send an email, by directly using the messages.send method or by sending it from a draft using the drafts.send method. The process of sending an email involves creating the email content and encoding it as a base64url string, creating a new message resource and setting its raw property to the encoded string, and then calling the appropriate method to send the message. The exact steps may vary depending on the chosen client library and programming language. The Gmail API has specific requirements for the format of the emails it handles, which must be MIME-compliant and encoded as base64url strings. There are various tools and libraries available in different programming languages that can assist in creating and encoding emails in this format [158].

Google APIs, including Gmail API, utilize the OAuth 2.0 protocol for verifying a user's identity and granting access to certain resources. This protocol is widely used and supported by Google for various types of applications, including web servers, client-side applications, installed applications, and limited-input device applications. To start using OAuth 2.0 with Google, developers need to obtain OAuth 2.0 client credentials from the Google API Console. After obtaining the credentials, the client application will request an access token from the Google Authorization Server. Once the token is received, the client application extracts it and sends it to the Google API they want to access. This process is demonstrated in Figure 52 [159].

In order to access a Google API using OAuth 2.0, there are five main steps that must be followed. The first step is to obtain OAuth 2.0 credentials from the Google API Console, such as a client ID and client secret, which are known to both Google and the application. The specific values required will depend on the type of application being built. Next, an access token must be obtained from the Google Authorization Server, which grants access to the API. This access token can grant varying levels of access to multiple APIs, and the level of access is controlled by the "scope" parameter. This parameter is included in the request for an access token. It is required that the user logs in with their Google account for certain requests. The user is then asked if they will grant the permissions that the application is requesting. This process is called user consent [160].

Next, when the user grants permission, the Google Authorization Server sends an access token to the application, along with a list of scopes of access that has been granted by the token. If the user does not grant permission, an error is returned. It's best to request scopes incrementally when access is required, instead of asking for them all at once. The application should compare the scopes received in the access token to the scopes required for its features and functionality. Any features that can't function without the required scope should be disabled. The scope requested may not match the scope received, even if the user granted all the requested scopes. It is important to refer to the documentation of each API for the required scopes for access. Some APIs may map multiple scope string values to a single scope of access, returning the same scope string for all values that were requested. After an application has obtained an access token through the OAuth 2.0 process, it is sent to a Google API in an HTTP Authorization request header. It is important to note that the access token is only valid for the specific set of operations and resources for which it was requested and must be refreshed if the application needs access to a Google API beyond the lifetime of a single token. The Google OAuth 2.0 endpoint is compatible with various programming languages and frameworks such as PHP, Java, Python, Ruby, and ASP.NET [160].

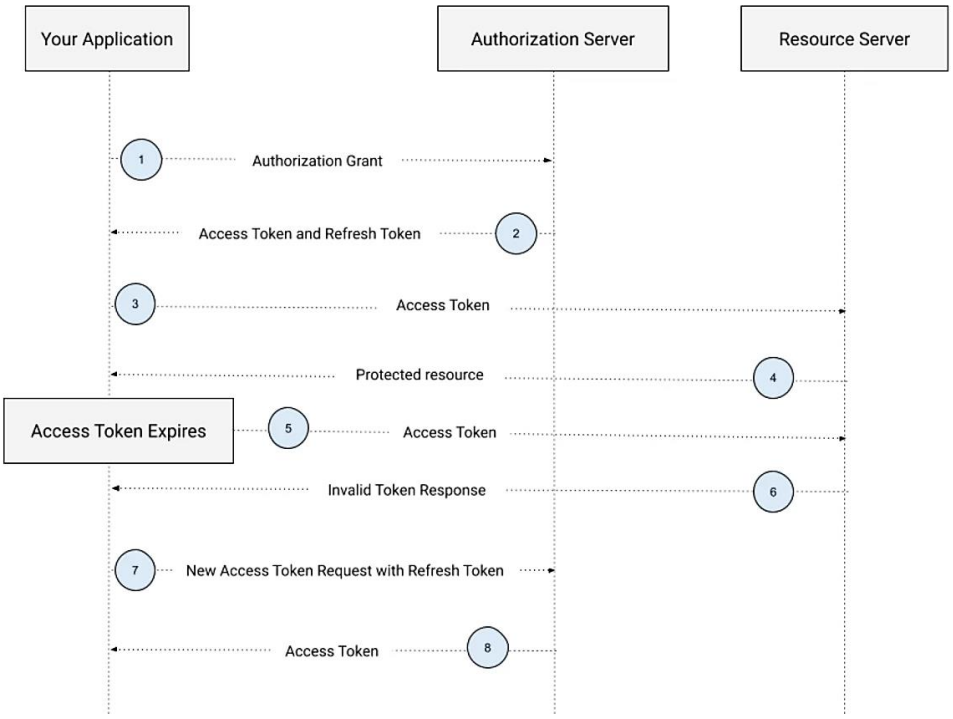

Figure 52: OAuth 2.0 protocol flow

The process of accessing a protected resource, as shown in Figure 52, begins when the client is seeking access to a protected resource and must first authenticate with the authorization server. This is achieved by presenting an authorization grant, which serves as a credential that represents the resource owner's authorization. The authorization server then verifies the validity of the authorization grant and authenticates the client to ensure it is authorized. If the grant is found to be valid, the authorization server issues an access token and a refresh token to the client. It is crucial for the client to securely store the refresh token for future use. The client can

then access the protected data from the resource server by presenting the issued access token. The resource server validates the access token and, if found to be valid, returns the requested data. However, if the access token has expired, the resource server will reject the request and return an error message indicating an invalid token. Upon receiving this response, the client can request a new access token by presenting the stored refresh token to the authorization server. The authorization server then uses the refresh token to issue a new access token. The size of the tokens can vary, with authorization codes being 256 bytes, access tokens at 2048 bytes, and refresh tokens at 512 bytes. Access tokens from Google Cloud's Security Token Service API are similar to OAuth 2.0 access tokens from Google API but have different size limits [160].

It is important to consider that a granted refresh token may no longer be valid for a number of reasons, such as if the user revokes access to the app, the refresh token hasn't been used in six months, the user changes their password and the refresh token contains Gmail scopes, the user account reaches a maximum number of granted refresh tokens, or the user belongs to a Google Cloud Platform organization with session control policies in place. In case the Google Cloud Platform project with an OAuth consent screen configured for an external user type and a publishing status of "Testing," then it is issued a refresh token expiring in 7 days. Additionally, there is a limit of 100 refresh tokens per Google Account per OAuth 2.0 client ID and a larger limit on the total number of refresh tokens a user or service account can have across all clients. To avoid reaching these limits, developers can limit the number of clients authorized per Google Account or create additional users with administrative privileges to authorize some of the clients [160].

iii. Streamlit

Streamlit is considered a valuable tool in academia for its ability to easily create web applications directly from Python [161]. Despite being relatively new, with its first beta release dating back to April 2019, research teams from around the world have begun to adopt the framework to showcase the results of their projects. Today, there are many publications that mention Streamlit as the visualization framework of choice, spanning a wide range of fields, including health [162], computer science [163], economics [164], and civil engineering [165], among others.

The Streamlit framework provides a platform for creating interactive web applications by combining a frontend single-page application (SPA) and a backend server. The framework starts a web server, serving a frontend app whose contents are generated based on a Python script written by the user. The frontend app communicates continuously with the web server, triggering the execution of the Python code on the server whenever events occur. This server-client architecture allows for the creation of interactive web apps solely through the implementation of logic in the server-side Python code.

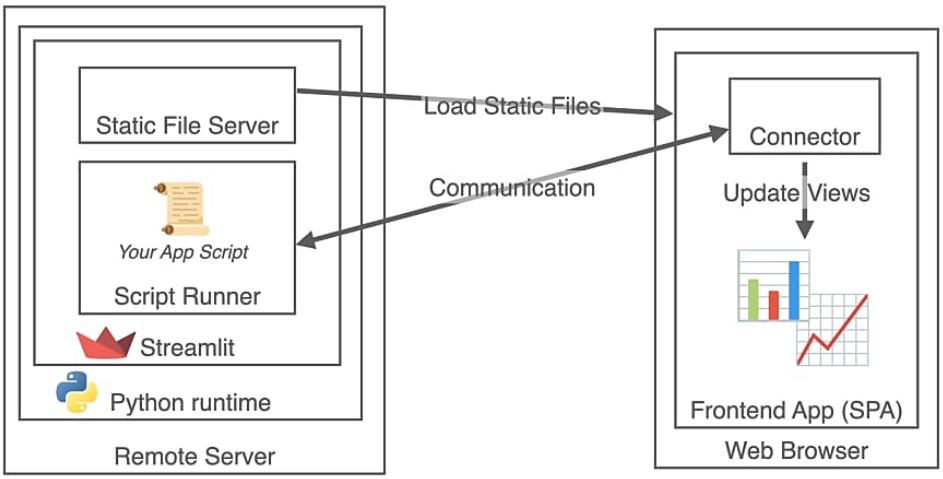

Figure 53: Streamlit framework

Streamlit is a web framework that differs from others in that it modifies the actual Document Object Model (DOM) and its state on the server side to render the final web document that is sent to the browser. This approach does not raise any security concerns under normal usage. However, if the code is poorly written and user input is not properly sanitized, it may be vulnerable to Remote Code Execution (RCE) attacks, similar to the way PHP can be vulnerable. To start a Streamlit web application, the user must run the binary file (streamlit.exe on Windows or stream-lit.sh on MacOS or Linux) using the default Python interpreter against the target document. This will initialize the application's configuration, including secrets, settings, themes, and the Delta Generator (DG). The DG acts as a middleman between the Python script and the ReactJS web application that is served by Streamlit. The DG is responsible for efficiently transferring HTML components to be rendered on the client side and then retrieving their state. This process is demonstrated in Figure 54 [166].

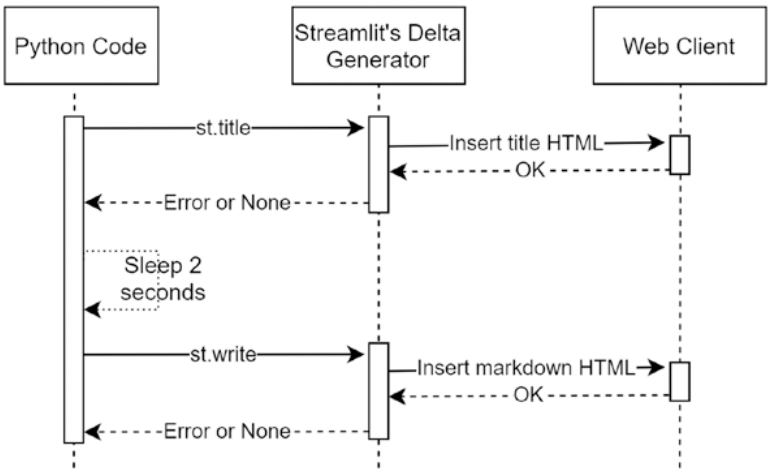

Figure 54: Streamlit's Delta Generator (DG)

The initial render starts at the beginning of the Python document and ends at the last line. However, subsequent renders do not start from the beginning of the file, but rather from the component that was interacted with by the user or that had its state changed. Each new render of a component is queued in the DG, which will later replace it with a new HTML snippet or add it to the final Document Object Model (DOM) among other

rendered HTML components. Streamlit components are rendered individually to avoid negatively impacting the user's experience with a blank page if the renders take too long. Such delays can occur from extensive ongoing computations, waiting for API responses, or even sleep functions [166]. To make it easier to understand, Streamlit is referred to as simply inserting HTML into the client's browser. However, it actually utilizes ReactJS's virtual DOM to insert elements and manage their state. By understanding this fact and looking at Streamlit's source code, it can be seen that Streamlit uses built-in ReactJS components to create a fully functional JavaScript web application with the help of Python.

With Streamlit Cloud, developers can easily connect their GitHub repository and then deploy the application with just one click. Streamlit automatically provisions the application with all necessary dependencies and updates it every time a new source code is pushed. There is no need for any further input from the developer. Additionally, if a developer needs more than one private application, additional computing resources, or enterprise-level features, they can upgrade to Streamlit's premium packages. One of the advantages of using Streamlit Cloud is the use of Secrets, the ability to securely store private data on Streamlit's servers and easily access it in the application [166]. This can be particularly useful for storing sensitive information such as user credentials, database connection strings, API keys, and other passwords. By storing this data on Streamlit's servers, we do not have to include it in plain text form in the code, which is a security best practice.

Streamlit Authenticator is used for user authentication in Streamlit applications. It is a simple process that involves importing the module and calling it to verify the credentials of predefined users. To set up the authentication, a YAML configuration file must be created, and the credentials of the users (names, usernames, and plain text passwords) must be defined. Additionally, a name, random key, and number of days for the expiry of a JSON Web Token (JWT) cookie that will be stored on the client's browser must be entered to enable password-less reauthentication. The Hasher module is then used to convert the plain text passwords into hashed passwords [167].

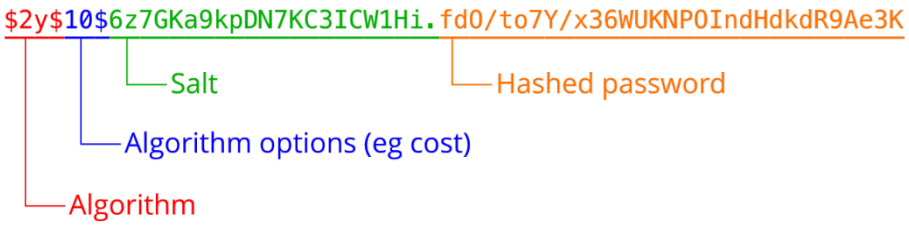

Figure 55: Bcrypt hashing

There are multiple methods for hashing a password, such as MD5, SHA256, SHA512, and others. However, the most commonly used algorithm by modern systems is Bcrypt. Bcrypt is used by default in Linux environments to protect users' passwords. Adding extra bytes to a password, known as adding a salt (Figure 55), produces a completely different hash. This helps in cases where a user reuses the same password across multiple websites and one of them has been breached. This makes it harder for attackers to determine if a user has reused the same password and it makes it harder to break the hash. However, this technique will not be

effective if the attacker knows the hashing salt and how it is applied. Bcrypt addresses this issue by introducing a cryptographic method to store randomly generated salts within the hash. This makes it possible to check if a Bcrypt hash is generated from plain text using a function from an abstracted Bcrypt library [166].

In order for the server to trust a request, the authentication process must verify a specific identifier, known as an "authentication token" or "token" for short. This token is issued by the server and must be verified. Tokens can take the form of custom session IDs or JSON Web Tokens (JWTs). In this case, we will use JWTs as they do not require the server to store them, making the process stateless. JWTs consist of three main parts that are encoded in base64 and separated by a period. The first part contains information about the payload signing mechanism, the second holds the raw payload, and the third contains a password-protected signature of the payload using the same hashing mechanism as the first part. When a new request is made, the headers will be checked for a token. The token's payload will be signed and the signature will be compared to the token's signature. If they match, it confirms that the token was issued by the server. For added security, the token will have an expiration date (30 days from the time of issuance by default) to prevent attackers from stealing legitimate tokens [166].

The returned name and authentication status can then be used to allow the verified user to access any restricted content. Additionally, an optional logout button can be added to the main body or sidebar. To access the persistent name, authentication status, and username variables, they can be retrieved through Streamlit's session state using st.session_state["name"], st.session_state["authentication_status"], and st.session_state["username"]. This allows for the use of Streamlit Authenticator to authenticate users across multiple pages.

iv. Unified Modeling Language (UML) Diagrams

a) Sequence Diagram (Nodes)

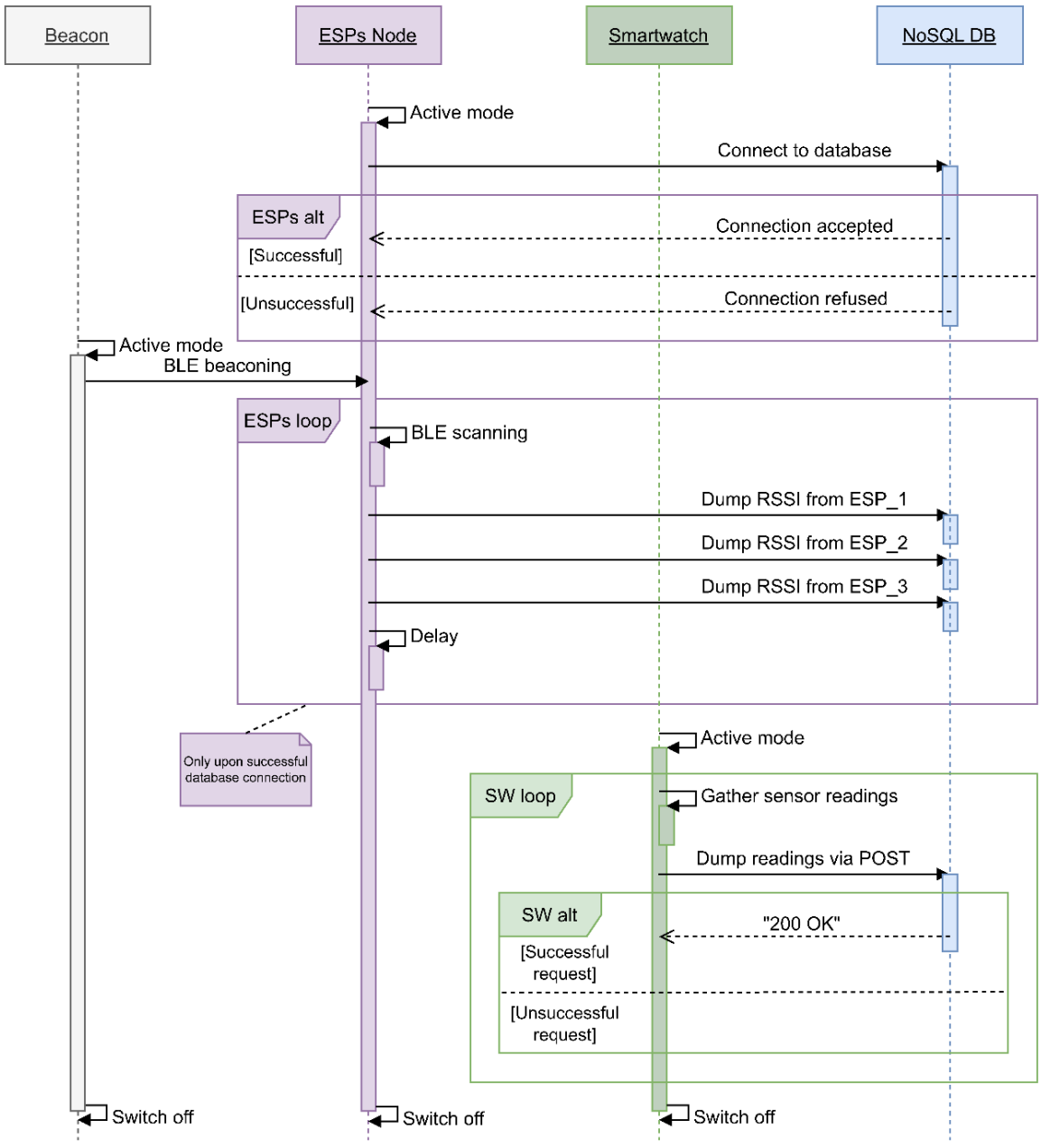

Figure 56: Sequence diagram of the interaction between the BLE beacon, ESP32s, smartwatch, and Firebase RTDB

b) Sequence Diagram (Web App)

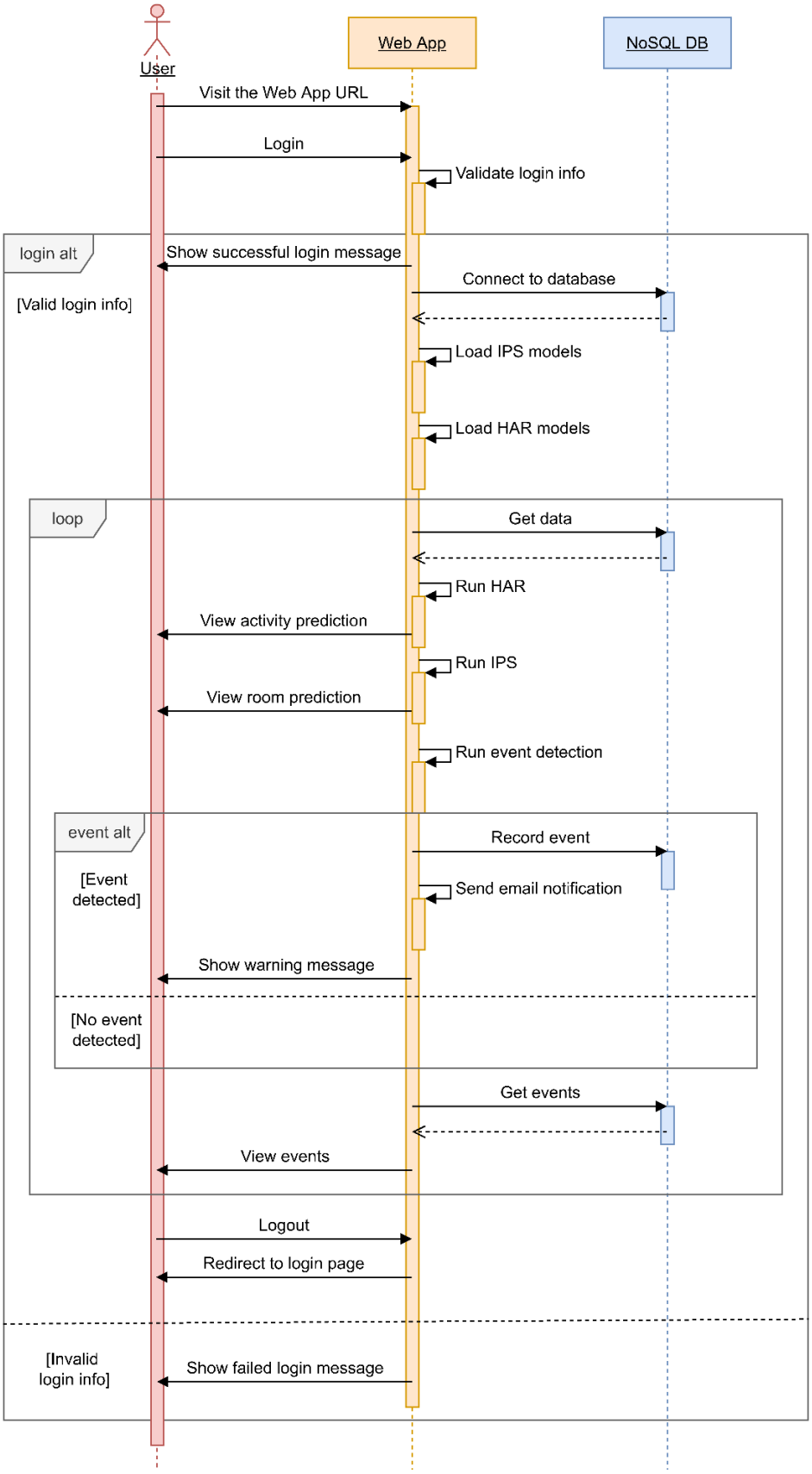

Figure 57: Sequence diagram of the interaction between the user, the web app, and Firebase RTDB

c) Use Case Diagram (Web App)

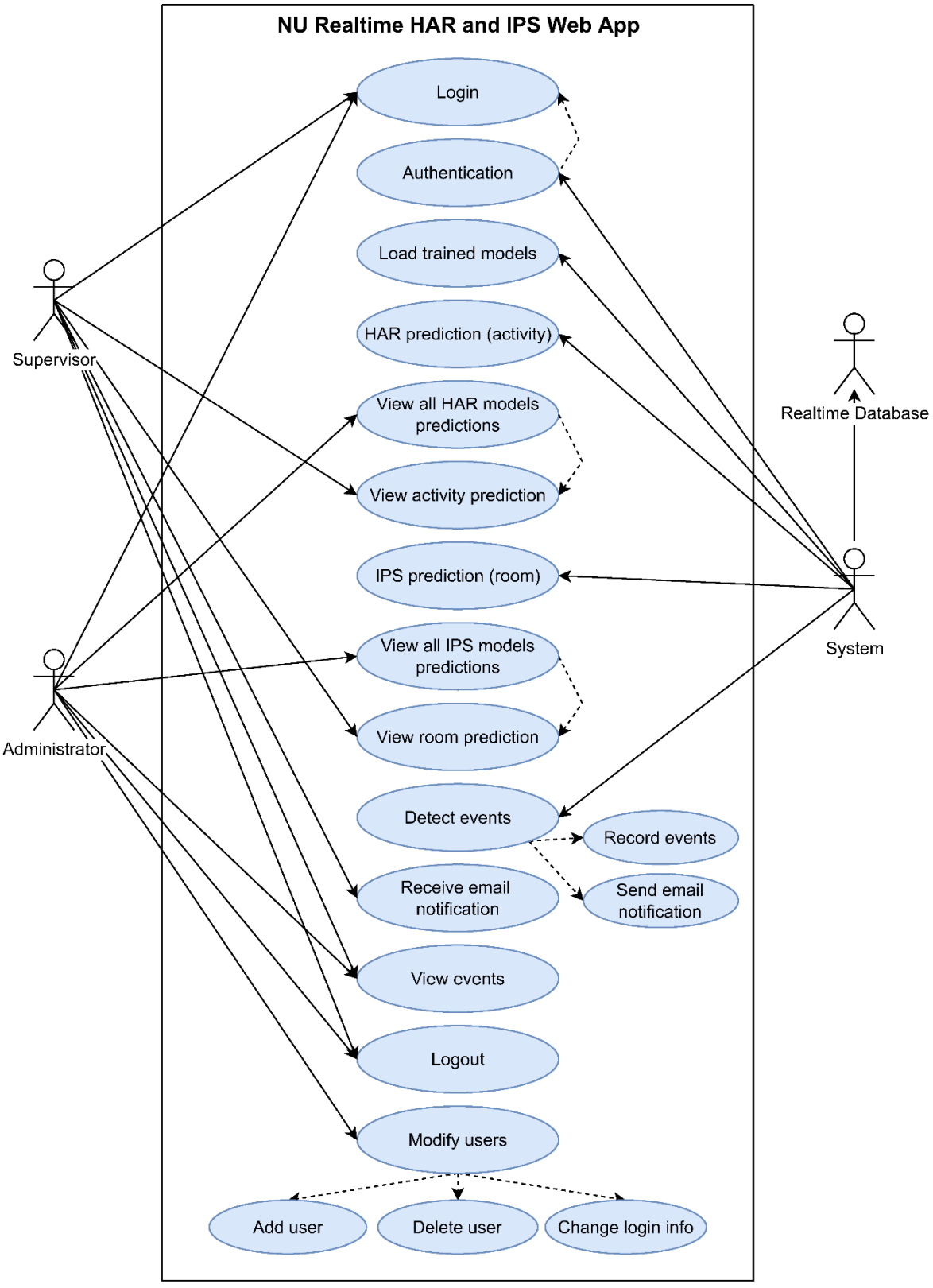

Figure 58: Use case diagram showing each actor and their use-cases

d) Activity Diagram (Web App)

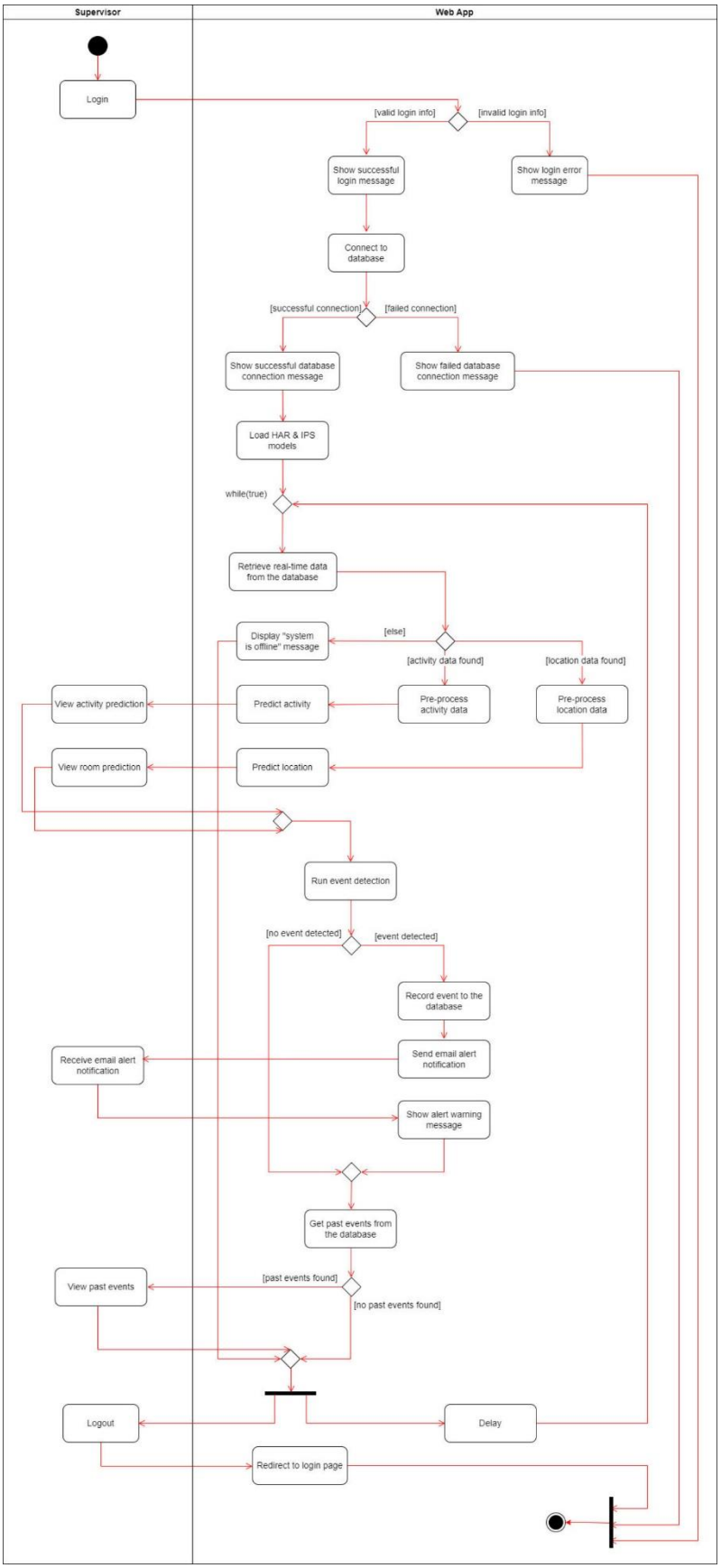

Figure 59: Activity diagram depicting the operation of the system

Chapter 4: Results and Outcomes

1. Human Activity Recognition

In this section, the findings of the experiment involving the application of deep learning methods for the detection of elderly activities are presented and analyzed. Figure 60 presents the confusion matrix for each model, which serves as a representation of the performance of the classification algorithms. The columns in the matrix are used to depict the instances predicted to belong to a specific class, while the rows are used to depict the instances that actually belong to the said class. Upon inspecting the results, it can be concluded that the CNN-LSTM model yielded the most favorable results compared to the actual values, while the ANN model yielded the least favorable results.

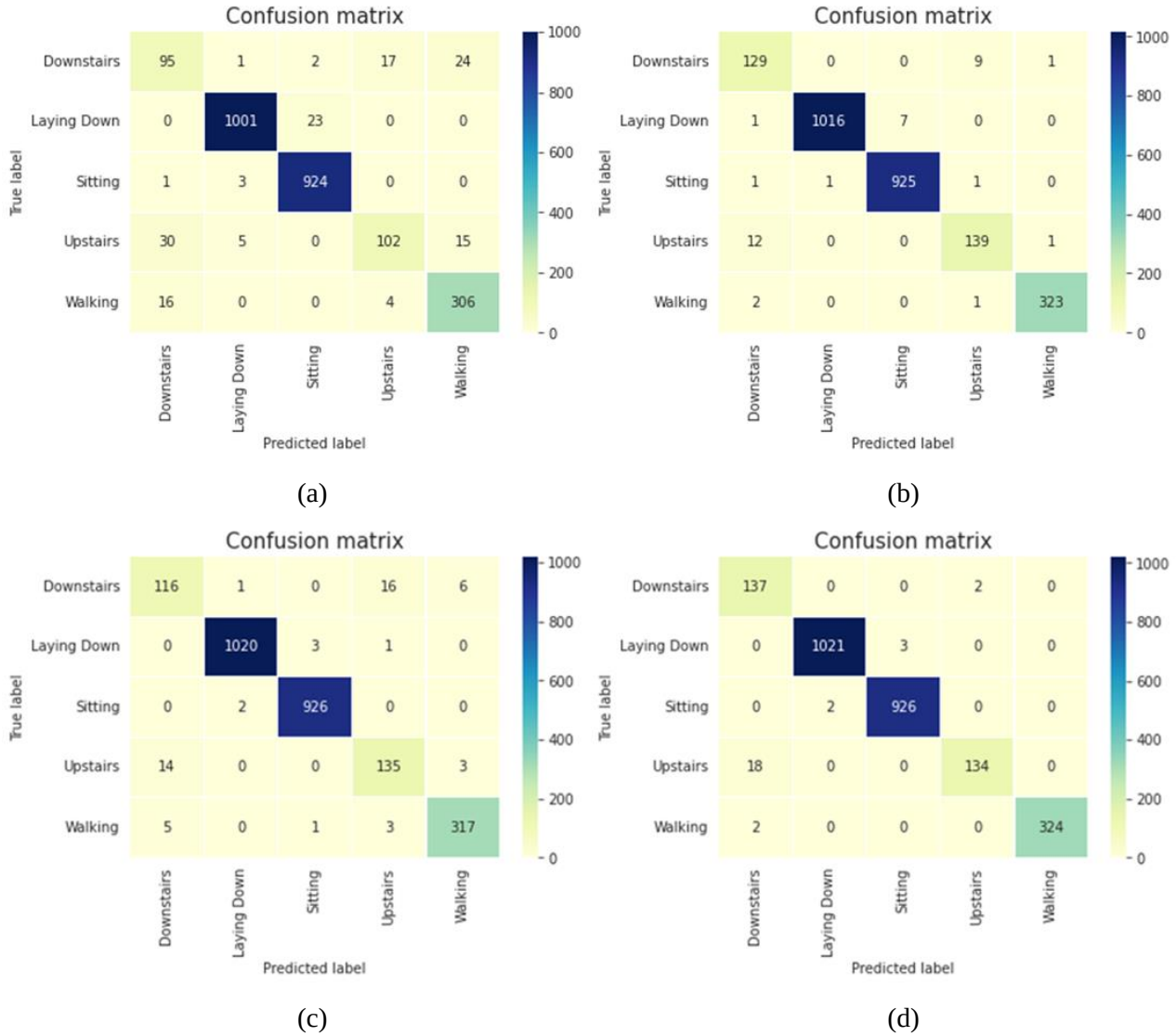

Figure 60: Confusion matrix of (a) ANN, (b) CNN, (c) LSTM, (d) CNN-LSTM

The accuracy and loss curves (training and validation) for the four deep learning models were examined, as shown in Figure 61. The results showed that the ANN achieved a validation loss of 0.2213 and a test accuracy of 94.51%, while the CNN model recorded a training loss of 0.0388 and a test accuracy of 98.56%.

Additionally, the LSTM model achieved a validation loss of 0.0749 and a test accuracy of 97.86%. However, the CNN-LSTM model achieved the lowest validation loss and highest test accuracy, which are 0.0370 and 98.95%, respectively. The validation loss was higher for all models but as the number of epochs progressed, a drastic reduction in the loss was observed, leading to the training and validation curves becoming nearly close. The training was terminated for both models as a result of the Early Stopping criteria, which was based on the completion of a continuous 20-epoch cycle without a decrease in the validation loss. The convergence of the training and validation losses at the end of the epoch cycle in the CNN, LSTM, and CNN-LSTM models indicates that these models were performing effectively. However, the ANN model's validation loss did not converge to the training loss, which may indicate that the model is underfitting as it is not complex enough to capture the patterns in the data.

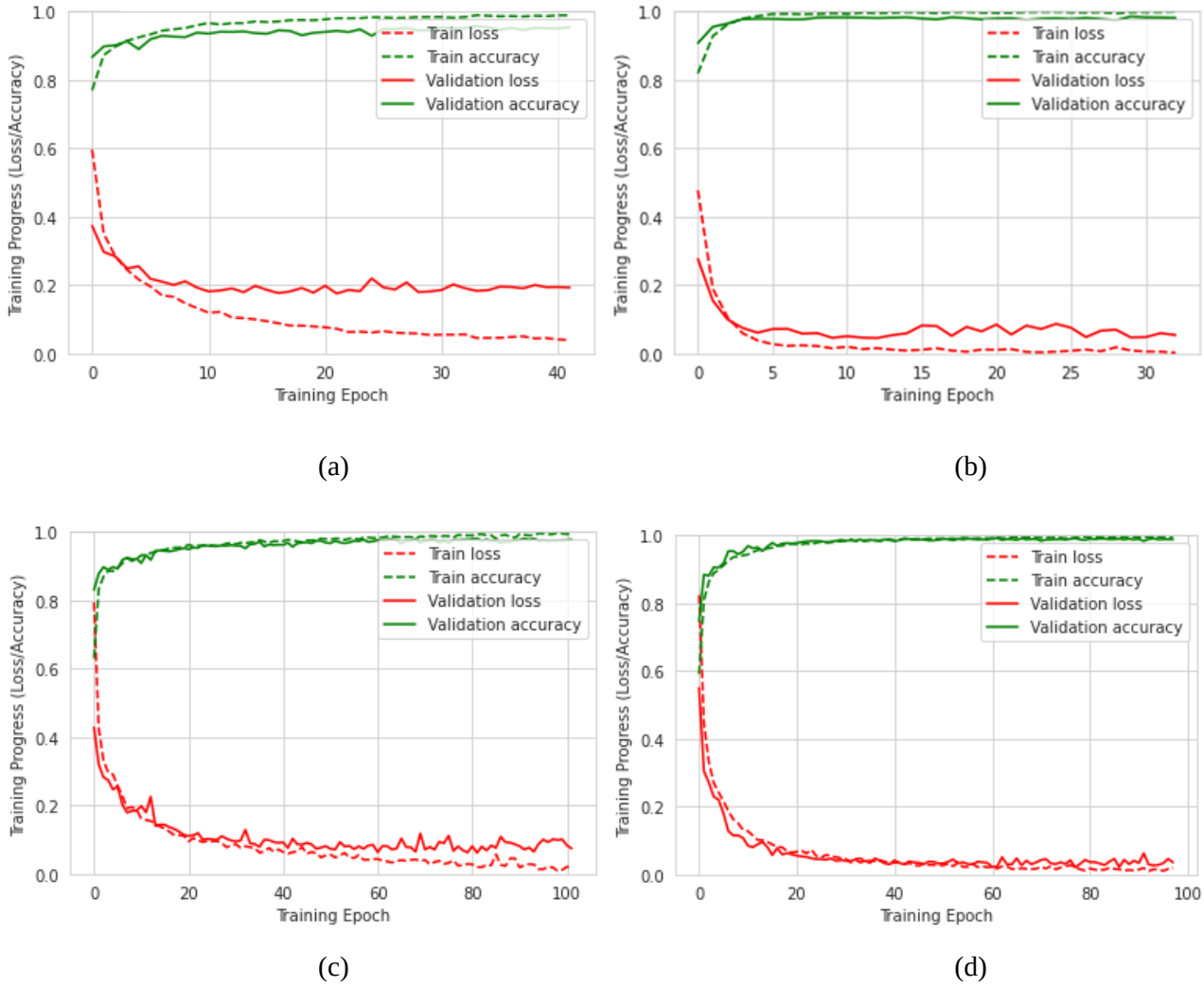

Figure 61: Accuracy and loss curves of (a) ANN, (b) CNN, (c) LSTM, (d) CNN-LSTM

In this research, the basic parameters of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) were analyzed to determine the dependent parameters of precision, recall, accuracy, and F1-score. The mathematical calculation of these parameters was conducted to evaluate the results [48]. TP refers to the number of correct predictions of the correct activity, while TN represents the number of correct

predictions of the incorrect activity. Conversely, FP indicates the number of incorrect predictions of the correct activity and FN represents the number of incorrect predictions of the incorrect activity. The precision, recall, accuracy, and F1 score were determined through the calculation of the aforementioned parameters.

$$\text{Precision} = \frac{TP}{(TP + FP)}$$
$$\text{Recall} = \frac{TP}{(TP + FN)}$$
$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$
$$\text{F1-score} = \frac{(2TP)}{(2TP + FP + FN)}$$

(21)

Moreover, Cohen's Kappa statistic and ROC AUC will be calculated. The computation of Cohen's Kappa is a statistical metric that assesses the inter-annotator agreement in a classification task. This score is calculated as the ratio of the observed agreement between two annotators to the expected agreement that would occur under random labeling. The formula for this calculation is represented as [168]:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

(22)

Where p_o represents the empirical probability of agreement between the annotators on the label assigned to a sample, and p_e represents the expected agreement that would occur if both annotators were to randomly assign labels. The estimation of p_e is performed through the use of a per-annotator empirical prior over the class labels. The Kappa statistic is a measure that ranges from -1 to +1, where the highest value indicates complete agreement, and a value of zero or below signifies an agreement that occurs by chance.

The area under the ROC curve (AUC ROC) is a measure of how well the classifier can distinguish between positive and negative classes but can be extended to multiclass classification problems using the One vs. Rest (OvR) technique. An AUC ROC of 1.0 represents a perfect classifier, while an AUC ROC of 0.5 represents a random classifier [169].

In addition to the evaluation of these parameters, the processing time was also considered in this study due to the complex nature of the dataset. The processing time of the algorithms was measured to gauge their efficiency.

Table 12: HAR performance metrics

	Accuracy	Loss	Precision	Recall	F1-score	Cohen's Kappa	ROC AUC	Prediction Time
LSTM	97.86%	0.0749	97.85%	97.86%	97.85%	0.9689	0.9866	1.4973
CNN	98.56%	0.0388	98.58%	98.56%	98.57%	0.9791	0.9910	0.2407
CNN-LSTM	98.95%	0.0370	99.03%	98.95%	98.95%	0.9847	0.9934	0.3529
ANN	94.51%	0.2213	94.46%	94.51%	94.42%	0.9201	0.9657	0.2158

Table 13: HAR per-class F1-score

	Laying down	Sitting	Walking	Upstairs	Downstairs
LSTM	100%	100%	97%	88%	85%
CNN	100%	99%	99%	92%	91%
CNN-LSTM	100%	100%	100%	93%	93%
ANN	98%	98%	91%	74%	68%

As evident from Table 13, the CNN-LSTM model outperformed the other models in correctly classifying the activity for all five activities. On the other hand, the ANN model demonstrated the lowest accuracy in classifying all activities. The highest accuracy was achieved by the CNN-LSTM model, with a value of 98.95%. The analysis of Figure 61 and Table 13 revealed that the majority of misclassified records were attributed to the activities of "going upstairs" and "going downstairs", which are similar in nature and hence posed challenges for the models to differentiate between them. Figure 62 visualizes the accuracy, precision, recall, F1-score, and ROC AUC values of all HAR models. The CNN-LSTM classifier exhibited the best accuracy, precision, recall, F1-score, Cohen's Kappa, and ROC AUC values of 98.95%, 99.03%, 98.95%, 98.95%, 0.9847, 0.9934, respectively. However, the CNN model showed the best prediction time among CNN, LSTM, and CNN-LSTM models, with 0.2407 seconds. In conclusion, the proposed CNN-LSTM method demonstrated superior overall performance compared to the other deep learning models. Figure 63 visualizes the loss and prediction time of all HAR models.

Despite the highly imbalanced nature of the dataset, the proposed system demonstrated exceptional performance. Thus, it can be concluded that the proposed method is capable of detecting the activity of elderly people. The study found that CNN-LSTM is more appropriate for the Human Activity Recognition (HAR) of elderly people compared to other deep learning architectures. However, if processing time is a crucial factor, the CNN model is faster than the CNN-LSTM model and offers a comparable performance to that of the CNN-LSTM model.

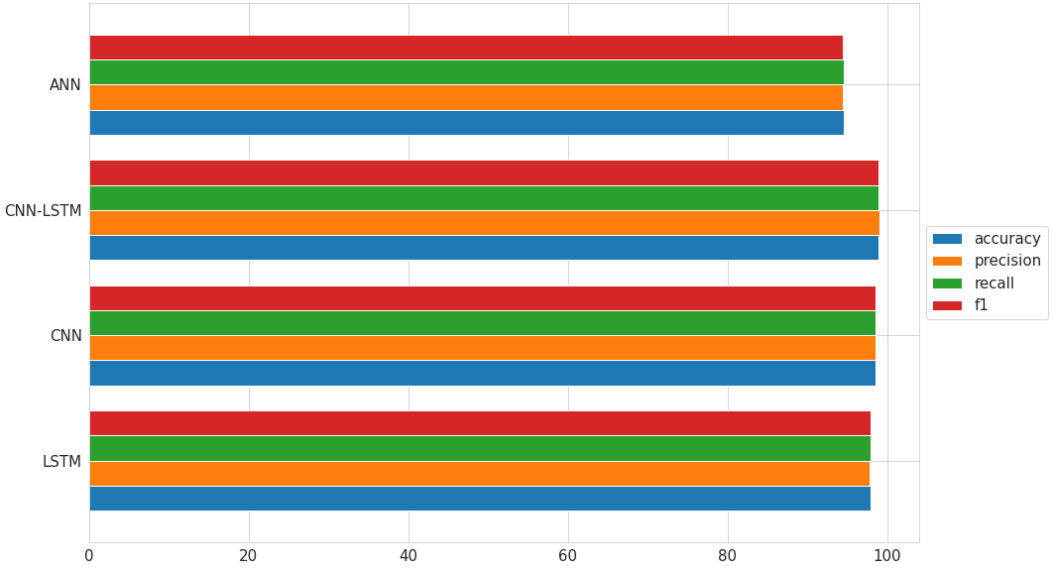

Figure 62: Accuracy, precision, recall, and F1-score metrics of the HAR models

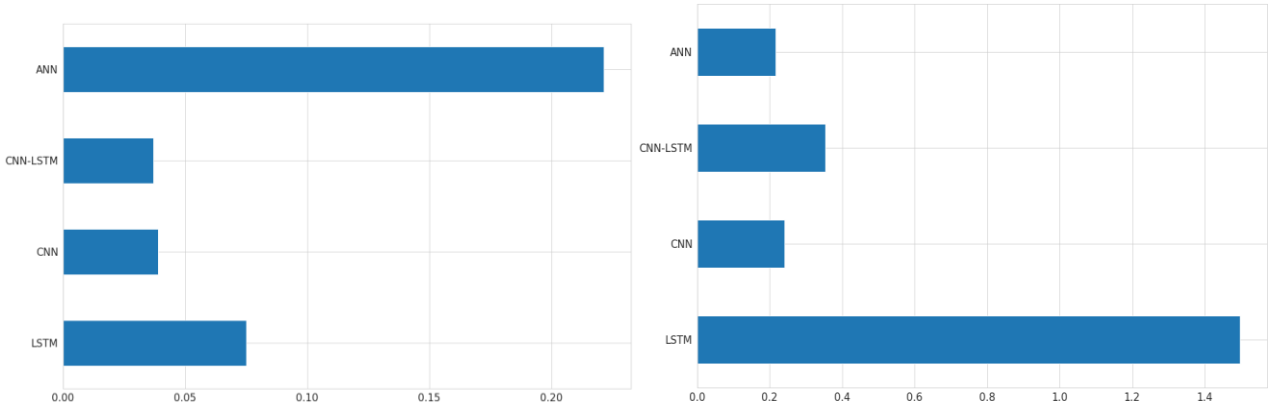

Figure 63: Validation loss (left) and prediction time (right) of the HAR models

2. Indoor Positioning System

In this section, the findings of the experiment involving the application of machine learning classifiers for the detection of a patient's location in a residential care facility are presented and analyzed. Figure 64 presents the confusion matrix for each classifier, which serves as a representation of the performance of the classification algorithms. The columns in the matrix are used to depict the instances predicted to belong to a specific class, while the rows are used to depict the instances that actually belong to the said class. Upon inspecting the confusion matrices, it can be concluded that the Random Forest classifier achieved the most favorable results compared to the true values.

Table 14: IPS performance metrics

Classifier	Train Time Mean	Test Time Mean	CV Acc.	CV STD	Test Acc. Mean	Precisi-on Mean	Recall Mean	F1-score Mean
RandomForestClassifier	0.1610	0.0195	83.61%	0.0257	84.20%	84.48%	84.13%	84.12%
VotingClassifier	0.5237	0.0274	83.31%	0.0181	83.91%	83.98%	84.03%	83.88%
GradientBoostingClassifier	0.3584	0.0075	82.72%	0.0300	83.61%	83.71%	83.73%	83.52%
KNeighborsClassifier	0.0032	0.0121	79.59%	0.0257	80.24%	80.77%	80.48%	80.36%
SVC_rbf	0.0113	0.0089	78.64%	0.0219	79.88%	80.17%	79.98%	79.96%
SVC_poly	0.9735	0.0077	79.82%	0.0259	79.65%	79.99%	79.84%	79.71%
NuSVC	0.0854	0.0105	78.82%	0.0233	79.59%	80.43%	79.54%	79.58%
DecisionTreeClassifier	0.0036	0.0055	79.94%	0.0290	79.29%	79.29%	79.61%	79.29%
LinearDiscriminantAnalysis	0.0038	0.0051	80.24%	0.0182	79.17%	79.84%	78.93%	79.06%
QuadraticDiscriminantAnalysis	0.0033	0.0055	80.06%	0.0218	79.05%	79.77%	79.12%	79.04%
GaussianNB	0.0032	0.0057	73.49%	0.0268	72.90%	74.12%	72.44%	72.77%
LinearSVC	0.0590	0.0063	68.93%	0.0747	66.75%	77.55%	68.48%	65.87%

In our study, we utilized the random cross-validation, also known as the shuffle-split cross-validation technique, to mitigate the bias of our models. This method is similar to the k-fold cross-validation approach where the data is divided into k-folds, with one-fold designated as the test set and the remaining (k − 1) folds serving as the training set. The difference between the k-fold cross-validation and the shuffle split cross-validation lies in the data shuffling after each iteration. In the k-fold cross-validation method, the data is not shuffled, while in the shuffle-split cross-validation method, the data is shuffled before being split into training and test sets. Our data was shuffled and split (n = 10) times, with random sampling on each iteration. The predicted values obtained from the test and training sets in each iteration were not utilized in the development of the model, but instead were appended to a list and the error was calculated over the entire dataset. The

shuffle split cross-validation method, with a split of 20% for the test set and 80% for the training set, resulted in the Random Forest classifier having the highest cross-validation accuracy of 83.61%, but also with a relatively high standard deviation of 0.257. However, the Random Forest and Gradient Boosting classifiers are faster than the Voting classifier when it comes to testing time. Thus, if processing time is a crucial factor, the RFC or GBC would be more appropriate than the Voting Classifier while not losing performance.

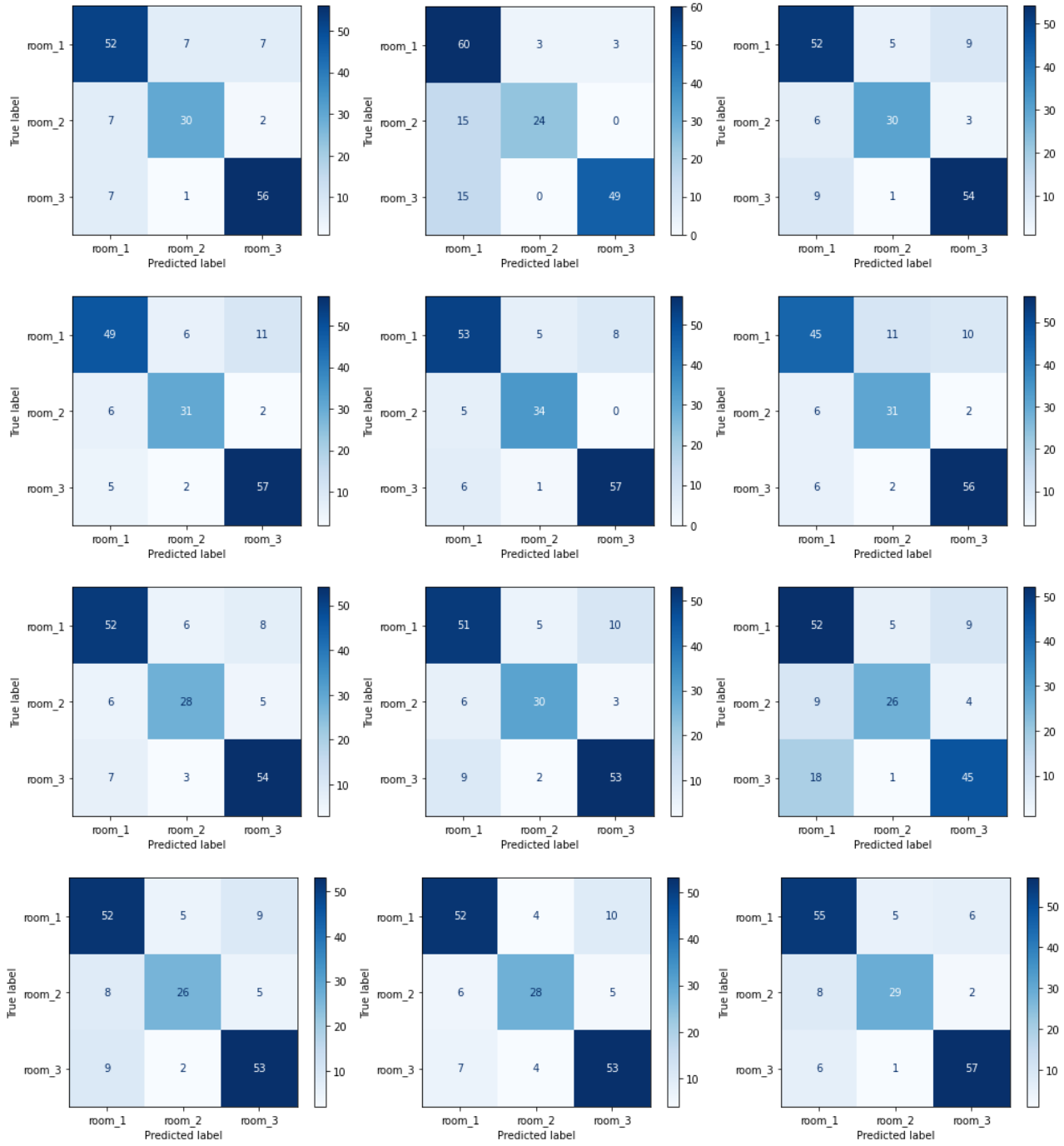

Figure 64: Confusion matrices of the 12 IPS models (from left to right): kNN, SVC (Poly), SVC (RBF), RFC, GBC, DT, LinearSVC, NuSVC, NB, LDA, QDA, Voting

(Room names were changed from ‘room_1’, ‘room_2’, and ‘room_3’ to 'Living Room', 'Bedroom', and 'Bathroom', respectively.)

Table 14 shows the performance metrics of the IPS models. The five best-performing classifiers with a good tradeoff between high cross-validation accuracy and low standard deviation were chosen to be Random Forest, Gradient Boosting, Decision Tree, Linear Discriminant Analysis, and Quadratic Discriminant Analysis. A Voting Classifier was implemented using a majority 'hard' voting of these five classifiers. The Voting Classifier was able to achieve the lowest cross-validation standard deviation of all IPS models at just 0.0181, as well as the second-highest cross-validation accuracy at 83.31% and second-highest F1-score mean at 83.88%, coming second to Random Forest.

Figure 65 visualizes the training time and test time of each of the 12 classifiers, and Figure 66 shows the cross-validation score mean of each of the 12 classifiers.

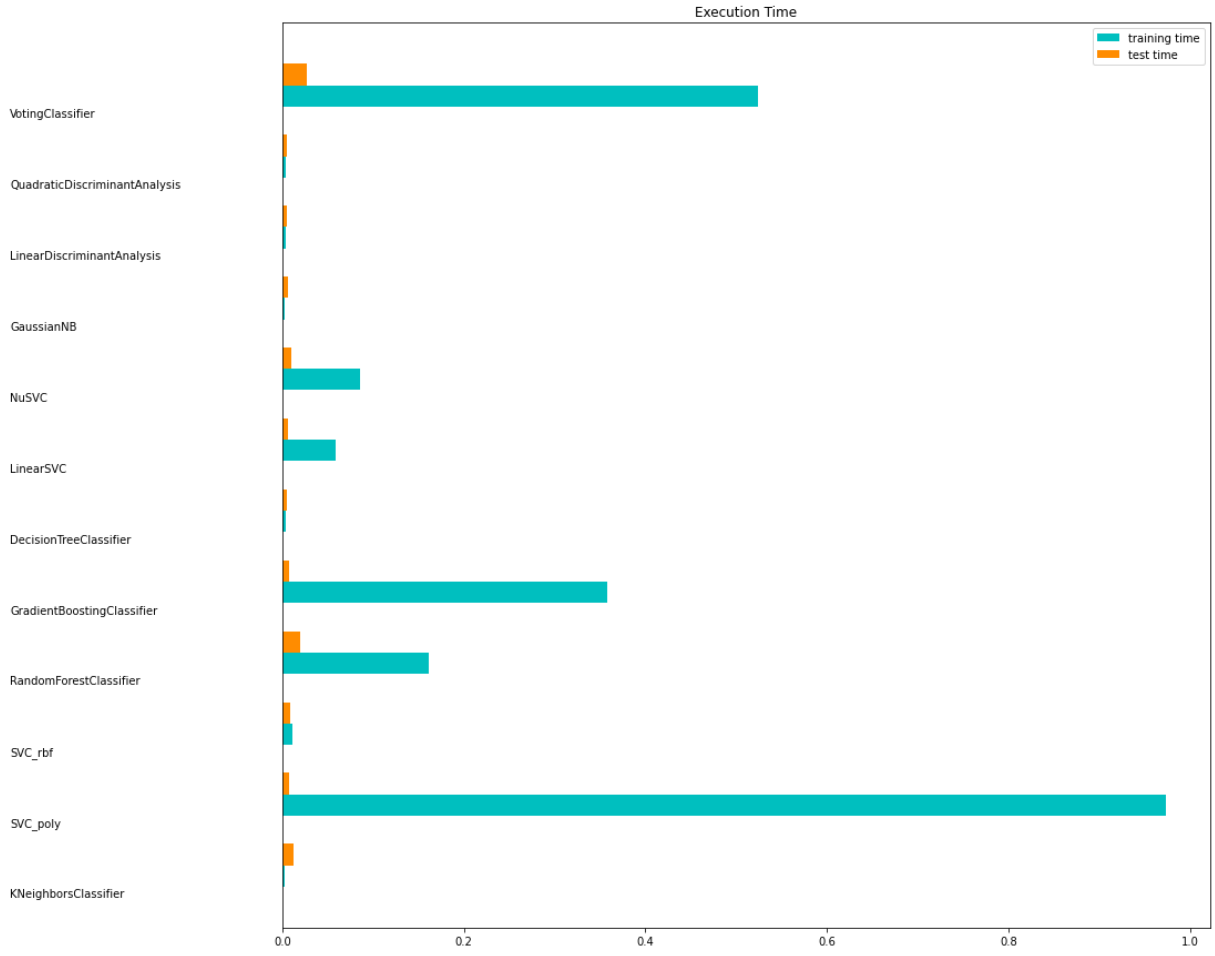

Figure 65: IPS classifiers' training time and test time

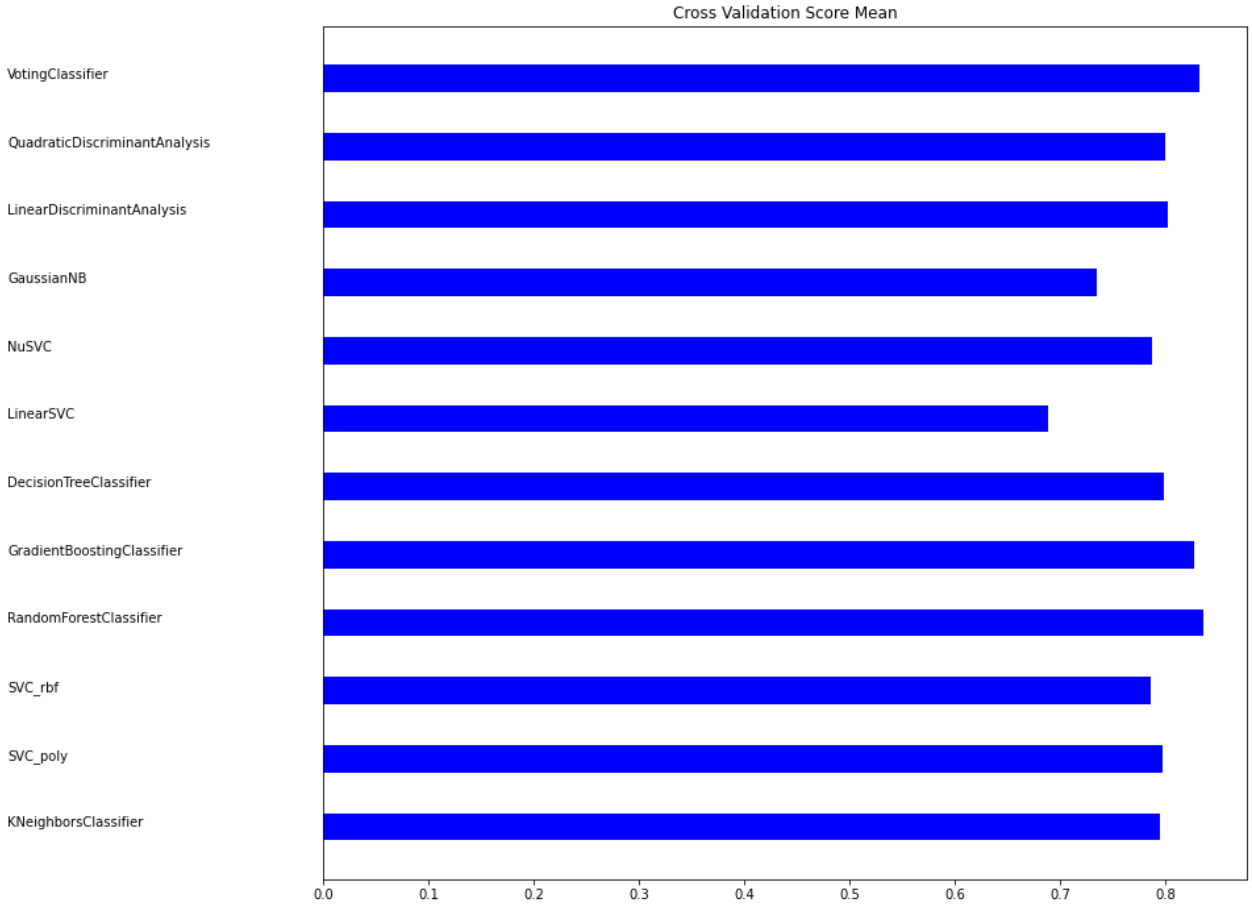

Figure 66: IPS classifiers' cross-validation accuracy mean

3. Real-time HAR and IPS via Web App

This section showcases the real-time activity recognition and indoor positioning Web app and its features. The following figures are screenshots taken from the Web app as it is up and running in real-time.

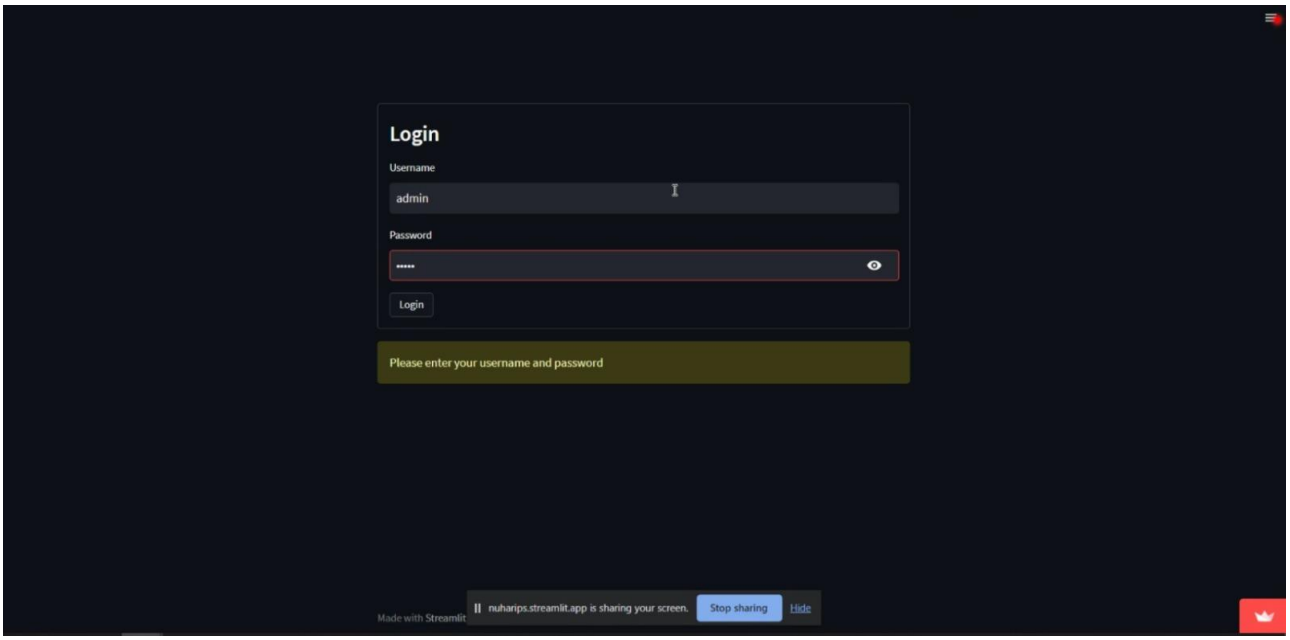

Figure 67: Admin login

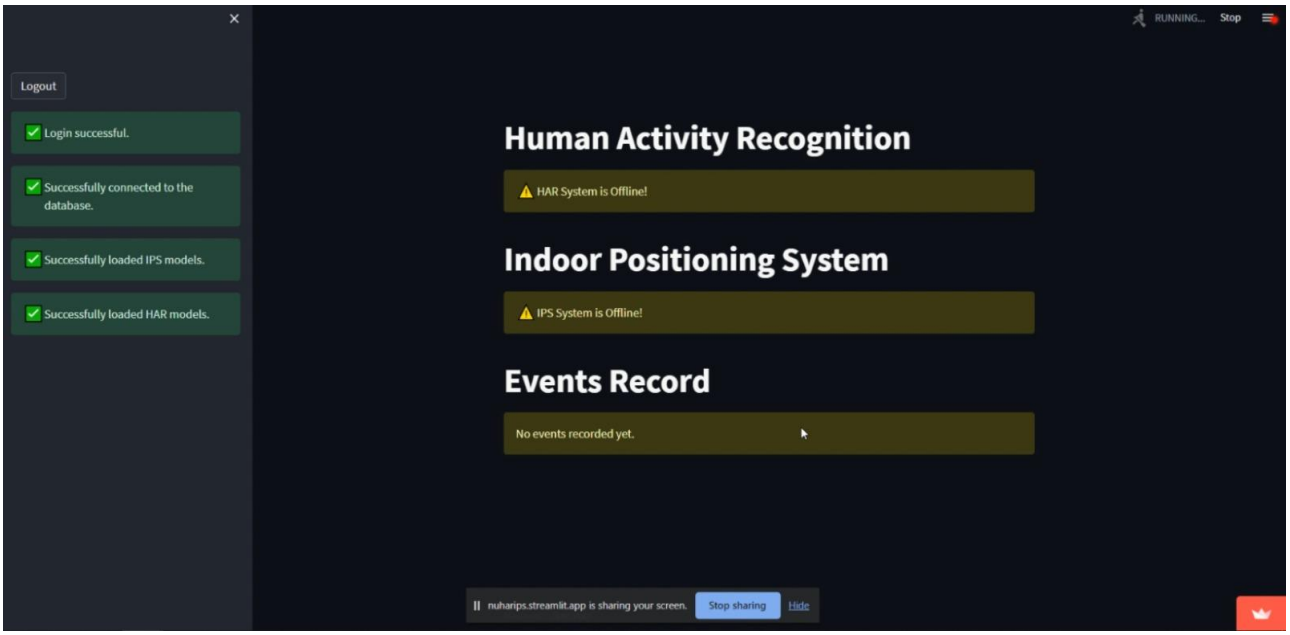

Figure 68: A system is offline when its components are unreachable/offline

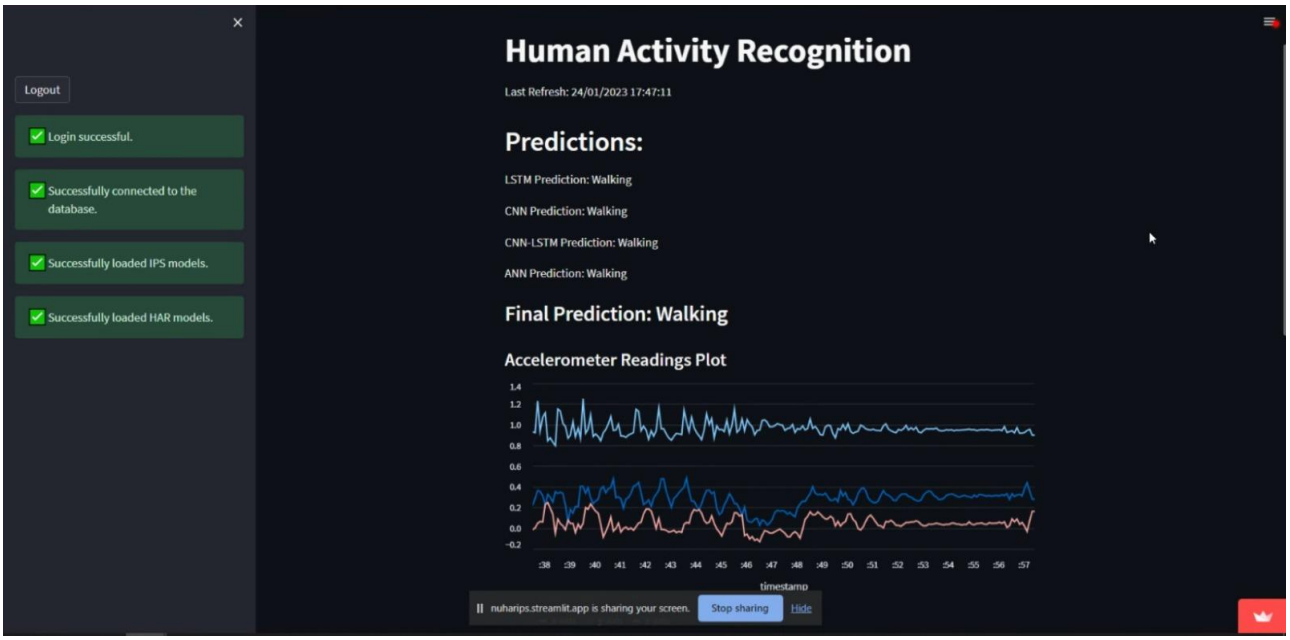

Figure 69: Admin dashboard includes the prediction of each HAR model

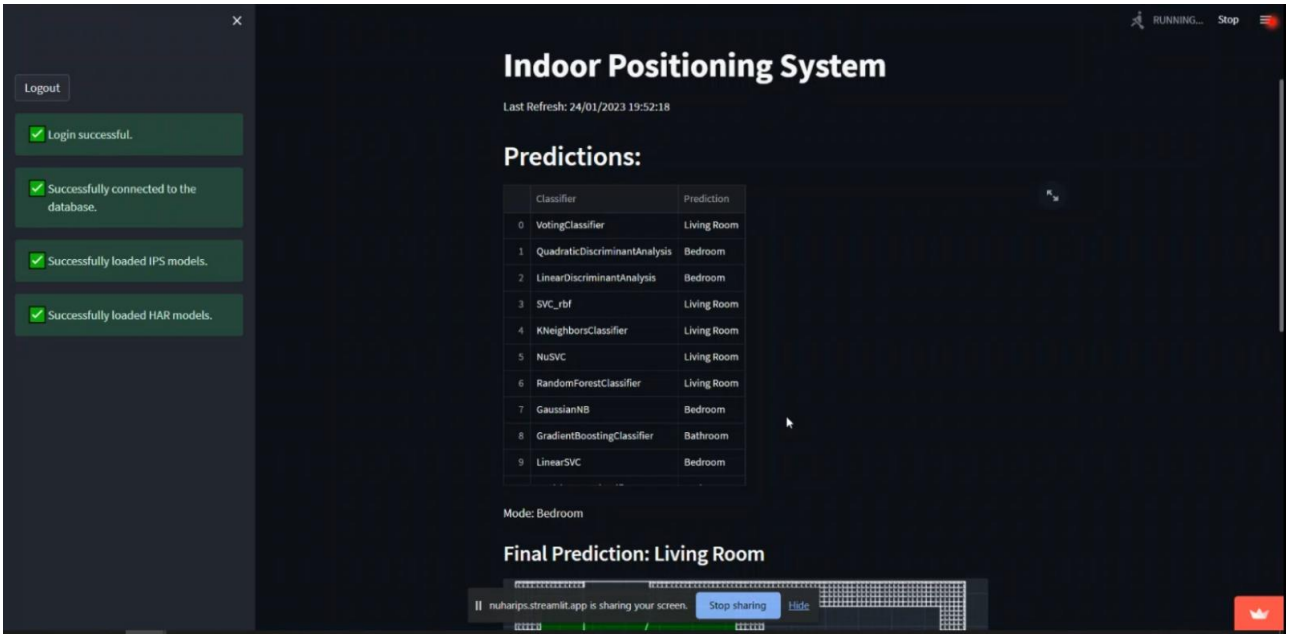

Figure 70: Admin dashboard includes the prediction of each IPS classifier

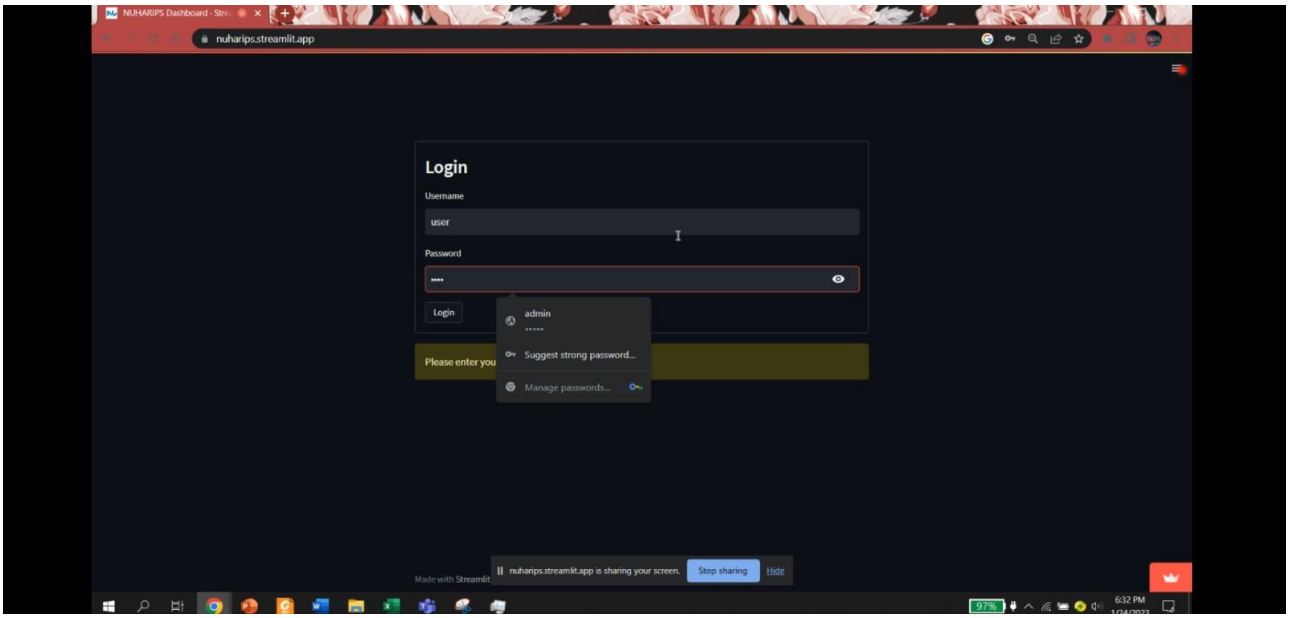

Figure 71: User login

Figure 72: User dashboard includes the final HAR predictions

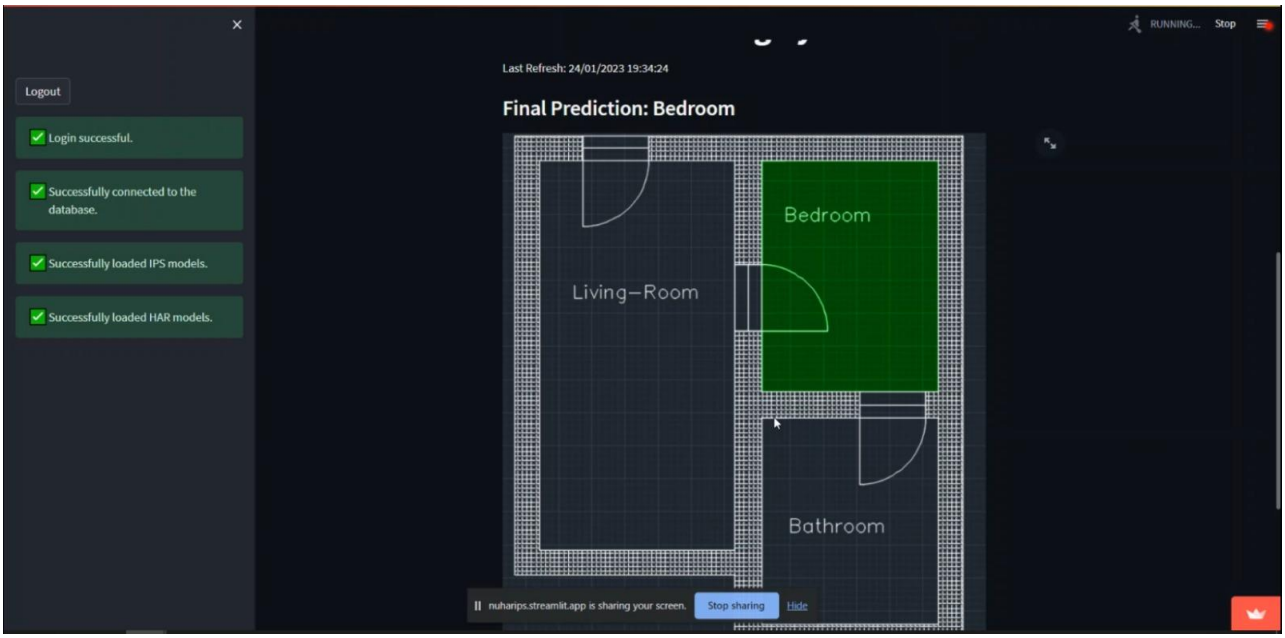

Figure 73: User dashboard includes the final IPS predictions

The web app is able to detect two types of events:

- Inconsistent event: e.g., laying down in the bathroom.
- Alarming activity: location and activity have not changed for a period of time (configurable setting)

Figure 74: Event detection and warning message

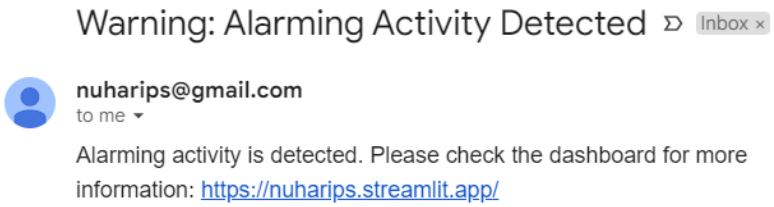

Figure 75: Email alert notification

Chapter 5: Conclusions and Future Work

In conclusion, this research aimed to propose a solution that combines artificial intelligence and sensor readings to identify five different activities, including walking, sitting, laying down, ascending stairs, and descending stairs. To solve the problem, various deep learning models such as Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Long-Short Term Memory (LSTM), and CNN-LSTM were employed and their performance was evaluated. The results indicated that the CNN-LSTM model performed the best among all the models, with a remarkable F1-score of 98.95%. This outcome emphasizes the potential of deep learning and sensor readings in detecting human activities and validates the effectiveness of the CNN-LSTM model for this specific problem.

Furthermore, an Indoor Positioning System (IPS) was implemented using the Received Signal Strength Indicator (RSSI) measurements of a Bluetooth Low Energy (BLE) beacon. A range of machine learning classifiers was employed in this study, including k-Nearest Neighbor (kNN), Support Vector Machine (SVM) with linear, polynomial, and RBF kernels, NuSVC, Random Forest, Decision Tree, Gradient Boosting, Gaussian Naive Bayes, Linear Discriminant Analysis, and Quadratic Discriminant Analysis. The evaluation results revealed that the Random Forest classifier achieved the highest mean F1-score of 84.12%, and the Voting Classifier obtained the second-highest mean F1-score of 83.88%. These findings demonstrate the potential of utilizing machine learning algorithms and RSSI measurements of BLE beacons for accurate indoor positioning.

In summary, the proposed solution demonstrated the potential of incorporating artificial intelligence and sensor readings to detect human activities and accurately determine indoor positioning. The research results suggest that deep learning models are more suitable for detecting activities, while machine learning algorithms are more effective for indoor positioning. The proposed solution could be applied to a range of real-world applications, including wearable devices, smart homes, and healthcare systems. However, further research is needed to improve the efficiency and accuracy of the solution and validate its effectiveness in real-world scenarios.

The limitations of the proposed solution and future work include:

- The current research used a limited dataset collected from a single smartwatch, which may not fully reflect the variability of human activities and movements. Future research should aim to collect more diverse and larger datasets to improve the generalizability of the proposed solution.
- The study used a limited number of subjects to collect the data, which may limit the generalization of the models to a larger population. In order to further generalize the models, it is important to collect data from a larger and more diverse sample of subjects, including individuals of different ages, genders, and health conditions. This will help to ensure that the models are robust and can be applied to a larger population.

- The proposed solution only aimed to detect five classes of activities, which may not cover the full range of human activities. Future research could aim to extend the proposed solution to detect a wider range of activities and incorporate more complex movements.
- While the proposed solution demonstrated promising results, there is still room for improvement in terms of the optimization of the algorithms used. Future work could focus on fine-tuning the algorithms and incorporating advanced deep-learning techniques to improve the performance of the solution.
- The proposed solution was developed as a standalone system and has not been integrated with other systems such as wearable devices, smart homes, or health care systems. Future work could aim to integrate the solution with these systems to demonstrate its full potential and evaluate its impact on real-world applications.
- The use of RSSI values of a BLE beacon for indoor positioning has some limitations that make it unreliable, such as interference of other signals in the environment (e.g., Wi-Fi or Bluetooth signals). This can result in inaccurate readings of the RSSI values and thus, unreliable indoor positioning results. Another limitation is the variation of the RSSI values over time. The strength of a radio signal can vary significantly over time due to changes in the environment, which can result in fluctuations in the RSSI values, making it difficult to accurately determine the position of the user. Additionally, the range of the BLE beacon is limited, which can also contribute to the unreliability of indoor positioning. Future work may include the use of RSSI smoothing, signal filtering, Trilateration, and Kalman filtering to improve the accuracy of the indoor positioning results.

The strengths of the proposed solution include the following:

- The proposed solution incorporates the use of artificial intelligence, specifically deep learning models, to accurately detect human activities. The CNN-LSTM model was shown to outperform other deep learning models, demonstrating the potential of incorporating it in human activity recognition.
- The proposed solution comprehensively compared the performance of four different deep learning models (ANN, CNN, LSTM, and CNN-LSTM) to determine the best model for the HAR of elderly people. This multi-model comparison allowed for a more thorough evaluation of the solution and demonstrated its robustness.
- The proposed solution achieved a high F1-score of 98.95% for the activity detection task, demonstrating its high accuracy in detecting five classes of activities.
- The solution was designed to perform activity detection and indoor positioning in real time, making it suitable for various real-world applications that require quick and accurate results. The use of sensor readings from a smartwatch allows for continuous monitoring of the user's activities, and the deep

learning models were designed to process the data and make accurate predictions in real time, providing quick and precise results. This real-time nature of the solution is crucial for applications that require immediate and accurate detection of human activities, such as healthcare and safety systems.

- The proposed solution not only aimed to detect human activities but also to determine the indoor location of the user. The integration of the activity detection and indoor positioning systems showed the potential of the solution to be used in various real-world applications.
- The proposed solution used a voting classifier to improve the accuracy of the indoor positioning task, demonstrating the potential of combining multiple classifiers to improve performance. The voting classifier achieved a mean F1-score of 83.88%, demonstrating its effectiveness.

In conclusion, this research aimed to develop a system for monitoring the daily activities and indoor positioning of elderly people in order to ensure their health and well-being. The proposed system utilized various deep learning and conventional machine learning methods to identify the activities of elderly individuals and determine their indoor location. The proposed system has the potential to assist older individuals in leading independent lives, fill in resource gaps and understaffing problems in residential care facilities, and enhance the care provided. It can also be applied in various other fields where activity monitoring and indoor positioning are essential.

References

[1] P. Paul and T. George, "An effective approach for human activity recognition on smartphone," *2015 IEEE International Conference on Engineering and Technology (ICETECH)*, pp. 1–3, Mar. 2015, doi: 10.1109/ICETECH.2015.7275024.

[2] World Health Organization, "International Classification of Functioning, Disability and Health (ICF)," *World Health Organization*, 2022. https://icd.who.int/browse11/l-m/en (accessed Jan. 29, 2023).

[3] Du, Lim, and Tan, "A Novel Human Activity Recognition and Prediction in Smart Home Based on Interaction," *Sensors*, vol. 19, no. 20, p. 4474, Oct. 2019, doi: 10.3390/s19204474.

[4] I. A. Bustoni, I. Hidayatulloh, A. Sn, and N. G. Augoestin, "Multidimensional Earcon Interaction Design for The Blind: a Proposal and Evaluation," in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia, Nov. 2018, pp. 384–388. doi: 10.1109/ISRITI.2018.8864487.

[5] S. Chernbumroong, S. Cang, A. Atkins, and H. Yu, "Elderly activities recognition and classification for applications in assisted living," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1662–1674, Apr. 2013, doi: 10.1016/j.eswa.2012.09.004.

[6] World Health Organization, "Ageing and health," *World Health Organization*, 2022. https://www.who.int/news-room/fact-sheets/detail/ageing-and-health (accessed Jan. 30, 2023).

[7] M. Karim, Md. Nuruzzaman Haque, Md. Abdul Goni, Md. Nure Alam Siddiqi, and Md. Mehedi Hasan, "Methods of Measuring Quality of Life: Theoretical Aspects and Empirical Evidence from Older Persons at Rajshahi City in Bangladesh," *SS*, vol. 9, no. 6, p. 253, 2020, doi: 10.11648/j.ss.20200906.16.

[8] D. Taylor, "Physical activity is medicine for older adults," *Postgrad Med J*, vol. 90, no. 1059, pp. 26–32, Jan. 2014, doi: 10.1136/postgradmedj-2012-131366.

[9] D. V. Gaz, T. M. Rieck, and N. W. Peterson, "Activity Tracking and Improved Health Outcomes," in *Fitness Medicine*, H. Sozen, Ed. InTech, 2016. doi: 10.5772/65240.

[10] M. B. Del Rosario *et al.*, "A comparison of activity classification in younger and older cohorts using a smartphone," *Physiol. Meas.*, vol. 35, no. 11, pp. 2269–2286, Nov. 2014, doi: 10.1088/0967-3334/35/11/2269.

[11] M. Awais, L. Chiari, E. A. F. Ihlen, J. L. Helbostad, and L. Palmerini, "Physical Activity Classification for Elderly People in Free-Living Conditions," *IEEE J. Biomed. Health Inform.*, vol. 23, no. 1, pp. 197–207, Jan. 2019, doi: 10.1109/JBHI.2018.2820179.

[12] A. Papagiannaki *et al.*, "Recognizing Physical Activity of Older People from Wearable Sensors and Inconsistent Data," *Sensors*, vol. 19, no. 4, p. 880, Feb. 2019, doi: 10.3390/s19040880.

[13] W. Sansrimahachai and M. Toahchoodee, "Mobile-phone based immobility tracking system for elderly care," in *2016 IEEE Region 10 Conference (TENCON)*, Singapore, Nov. 2016, pp. 3550–3553. doi: 10.1109/TENCON.2016.7848718.

[14] F. Tahavori *et al.*, "Physical activity recognition of elderly people and people with parkinson's (PwP) during standard mobility tests using wearable sensors," in *2017 International Smart Cities Conference (ISC2)*, Wuxi, China, Sep. 2017, pp. 1–4. doi: 10.1109/ISC2.2017.8090858.

[15] T. H. C. Nguyen, J. C. Nebel, and F. Florez-Revuelta, "Recognition of Activities of Daily Living with Egocentric Vision: A Review," *Sensors*, vol. 16, no. 1, p. 72, Jan. 2016, doi: 10.3390/s16010072.

[16] G. Cheng, Y. Wan, A. N. Saudagar, K. Namuduri, and B. P. Buckles, "Advances in Human Action Recognition: A Survey." arXiv, Jan. 23, 2015. Accessed: Jan. 16, 2023. [Online]. Available: http://arxiv.org/abs/1501.05964

[17] C. Dhiman and D. K. Vishwakarma, "A review of state-of-the-art techniques for abnormal human activity recognition," *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 21–45, Jan. 2019, doi: 10.1016/j.engappai.2018.08.014.

[18] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu, "Deep Learning for Sensor-based Human Activity Recognition: Overview, Challenges, and Opportunities," *ACM Comput. Surv.*, vol. 54, no. 4, pp. 1–40, May 2022, doi: 10.1145/3447744.

[19] K. Chen, L. Yao, D. Zhang, X. Wang, X. Chang, and F. Nie, "A Semisupervised Recurrent Convolutional Attention Model for Human Activity Recognition," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 31, no. 5, pp. 1747–1756, May 2020, doi: 10.1109/TNNLS.2019.2927224.

[20] M. Zeng *et al.*, "Understanding and improving recurrent networks for human activity recognition by continuous attention," in *Proceedings of the 2018 ACM International Symposium on Wearable Computers*, Singapore Singapore, Oct. 2018, pp. 56–63. doi: 10.1145/3267242.3267286.
[21] K. Chen, L. Yao, D. Zhang, B. Guo, and Z. Yu, "Multi-agent Attentional Activity Recognition," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Macao, China, Aug. 2019, pp. 1344–1350. doi: 10.24963/ijcai.2019/186.
[22] P. D. Sloane *et al.*, "The Public Health Impact of Alzheimer's Disease, 2000–2050: Potential Implication of Treatment Advances," *Annu. Rev. Public Health*, vol. 23, no. 1, pp. 213–231, May 2002, doi: 10.1146/annurev.publhealth.23.100901.140525.
[23] S. Zimmerman, P. D. Sloane, E. Heck, K. Maslow, and R. Schulz, "Introduction: Dementia Care and Quality of Life in Assisted Living and Nursing Homes," *The Gerontologist*, vol. 45, no. suppl_1, pp. 5–7, Oct. 2005, doi: 10.1093/geront/45.suppl_1.5.
[24] L. C. Shum *et al.*, "Indoor Location Data for Tracking Human Behaviours: A Scoping Review," *Sensors*, vol. 22, no. 3, p. 1220, Feb. 2022, doi: 10.3390/s22031220.
[25] M. N. Kamel Boulos and G. Berry, "Real-time locating systems (RTLS) in healthcare: a condensed primer," *Int J Health Geogr*, vol. 11, no. 1, p. 25, 2012, doi: 10.1186/1476-072X-11-25.
[26] A. Akl, B. Taati, and A. Mihailidis, "Autonomous Unobtrusive Detection of Mild Cognitive Impairment in Older Adults," *IEEE Trans. Biomed. Eng.*, vol. 62, no. 5, pp. 1383–1394, May 2015, doi: 10.1109/TBME.2015.2389149.
[27] C. P. Jansen, M. Diegelmann, E. L. Schnabel, H. W. Wahl, and K. Hauer, "Life-space and movement behavior in nursing home residents: results of a new sensor-based assessment and associated factors," *BMC Geriatr*, vol. 17, no. 1, p. 36, Dec. 2017, doi: 10.1186/s12877-017-0430-7.
[28] M. E. Bowen and M. Rowe, "Intraindividual Changes in Ambulation Associated With Falls in a Population of Vulnerable Older Adults in Long-Term Care," *Archives of Physical Medicine and Rehabilitation*, vol. 97, no. 11, pp. 1963–1968, Nov. 2016, doi: 10.1016/j.apmr.2016.05.013.
[29] M. E. Bowen, J. Crenshaw, and S. J. Stanhope, "Balance ability and cognitive impairment influence sustained walking in an assisted living facility," *Archives of Gerontology and Geriatrics*, vol. 77, pp. 133–141, Jul. 2018, doi: 10.1016/j.archger.2018.05.004.
[30] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surv. Tutorials*, vol. 11, no. 1, pp. 13–32, 2009, doi: 10.1109/SURV.2009.090103.
[31] M. Altini, D. Brunelli, E. Farella, and L. Benini, "Bluetooth indoor localization with multiple neural networks," in *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*, Modena, Italy, 2010, pp. 295–300. doi: 10.1109/ISWPC.2010.5483748.
[32] Sheng Zhou and J. K. Pollard, "Position Measurement using Bluetooth," *IEEE Trans. Consumer Electron.*, vol. 52, no. 2, pp. 555–558, May 2006, doi: 10.1109/TCE.2006.1649679.
[33] K. Townsend, R. Davidson, Akiba, and C. Cufí, *Getting started with Bluetooth low energy: tools and techniques for low-power networking*, Revised First Edition. Sebastopol, CA: O'Reilly, 2014.
[34] R. Poppe, "A survey on vision-based human action recognition," *Image and Vision Computing*, vol. 28, no. 6, pp. 976–990, Jun. 2010, doi: 10.1016/j.imavis.2009.11.014.
[35] J. K. Aggarwal and L. Xia, "Human activity recognition from 3D data: A review," *Pattern Recognition Letters*, vol. 48, pp. 70–80, Oct. 2014, doi: 10.1016/j.patrec.2014.04.011.
[36] Y. Wang, K. Wu, and L. M. Ni, "WiFall: Device-Free Fall Detection by Wireless Networks," *IEEE Trans. on Mobile Comput.*, vol. 16, no. 2, pp. 581–594, Feb. 2017, doi: 10.1109/TMC.2016.2557792.
[37] W. Wang, A. X. Liu, M. Shahzad, K. Ling, and S. Lu, "Device-Free Human Activity Recognition Using Commercial WiFi Devices," *IEEE J. Select. Areas Commun.*, vol. 35, no. 5, pp. 1118–1131, May 2017, doi: 10.1109/JSAC.2017.2679658.
[38] F. Foerster, M. Smeja, and J. Fahrenberg, "Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring," *Computers in Human Behavior*, vol. 15, no. 5, pp. 571–583, Sep. 1999, doi: 10.1016/S0747-5632(99)00037-0.
[39] A. M. Khan, Young-Koo Lee, S. Y. Lee, and Tae-Seong Kim, "A Triaxial Accelerometer-Based Physical-Activity Recognition via Augmented-Signal Features and a Hierarchical Recognizer," *IEEE Trans. Inform. Technol. Biomed.*, vol. 14, no. 5, pp. 1166–1172, Sep. 2010, doi: 10.1109/TITB.2010.2051955.
[40] N. Ravi, N. Dandekar, P. Mysore, and M. Littman, "Activity Recognition from Accelerometer Data," presented at the AAAI Conference on Artificial Intelligence, Jul. 2005. Accessed: Jan. 16, 2023.

[Online]. Available: https://www.semanticscholar.org/paper/Activity-Recognition-from-Accelerometer-Data-Ravi-Dandekar/bbde9ef4a4b4da0620b14a25c5a4a3d6bd4780e5
[41] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, and Y. Amirat, "Physical Human Activity Recognition Using Wearable Sensors," *Sensors*, vol. 15, no. 12, pp. 31314–31338, Dec. 2015, doi: 10.3390/s151229858.
[42] O. D. Lara and M. A. Labrador, "A Survey on Human Activity Recognition using Wearable Sensors," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013, doi: 10.1109/SURV.2012.110112.00192.
[43] M. Shoaib, S. Bosch, O. Incel, H. Scholten, and P. Havinga, "Fusion of Smartphone Motion Sensors for Physical Activity Recognition," *Sensors*, vol. 14, no. 6, pp. 10146–10176, Jun. 2014, doi: 10.3390/s140610146.
[44] L. Bao and S. S. Intille, "Activity Recognition from User-Annotated Acceleration Data," Berlin, Heidelberg, 2004, vol. 3001, pp. 1–17. doi: 10.1007/978-3-540-24646-6_1.
[45] scar D. Lara and M. A. Labrador, "A mobile platform for real-time human activity recognition," in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2012, pp. 667–671. doi: 10.1109/CCNC.2012.6181018.
[46] A. Sharma, Y.-D. Lee, and W.-Y. Chung, "High Accuracy Human Activity Monitoring Using Neural Network," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, Busan, Korea, Nov. 2008, pp. 430–435. doi: 10.1109/ICCIT.2008.394.
[47] W. Wu, S. Dasgupta, E. E. Ramirez, C. Peterson, and G. J. Norman, "Classification Accuracies of Physical Activities Using Smartphone Motion Sensors," *J Med Internet Res*, vol. 14, no. 5, p. e130, Oct. 2012, doi: 10.2196/jmir.2208.
[48] D. Riboni and C. Bettini, "COSAR: hybrid reasoning for context-aware activity recognition," *Pers Ubiquit Comput*, vol. 15, no. 3, pp. 271–289, Mar. 2011, doi: 10.1007/s00779-010-0331-7.
[49] Z.-Y. He and L.-W. Jin, "Activity recognition from acceleration data using AR model representation and SVM," in *2008 International Conference on Machine Learning and Cybernetics*, Kunming, China, Jul. 2008, pp. 2245–2250. doi: 10.1109/ICMLC.2008.4620779.
[50] L. T. Vinh *et al.*, "Semi-Markov conditional random fields for accelerometer-based activity recognition," *Appl Intell*, vol. 35, no. 2, pp. 226–241, Oct. 2011, doi: 10.1007/s10489-010-0216-5.
[51] J. Usharani and U. Sakthivel, "Human Activity Recognition using Android Smartphone," 2010. https://www.semanticscholar.org/paper/Human-Activity-Recognition-using-Android-Smartphone-Usharani-Sakthivel/dc3f3024d192562a2916bd81194509ef612236b3 (accessed Jul. 11, 2022).
[52] M. Vakili and M. Rezaei, "Incremental Learning Techniques for Online Human Activity Recognition." arXiv, Sep. 20, 2021. doi: 10.48550/arXiv.2109.09435.
[53] J. Muangprathub, A. Sriwichian, A. Wanichsombat, S. Kajornkasirat, P. Nillaor, and V. Boonjing, "A Novel Elderly Tracking System Using Machine Learning to Classify Signals from Mobile and Wearable Sensors," *International Journal of Environmental Research and Public Health*, vol. 18, no. 23, Art. no. 23, Jan. 2021, doi: 10.3390/ijerph182312652.
[54] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks," in *Web-Age Information Management*, vol. 8485, F. Li, G. Li, S. Hwang, B. Yao, and Z. Zhang, Eds. Cham: Springer International Publishing, 2014, pp. 298–310. doi: 10.1007/978-3-319-08010-9_33.
[55] C. A. Ronao and S.-B. Cho, "Human activity recognition with smartphone sensors using deep learning neural networks," *Expert Systems with Applications*, vol. 59, pp. 235–244, Oct. 2016, doi: 10.1016/j.eswa.2016.04.032.
[56] F. Gu, K. Khoshelham, S. Valaee, J. Shang, and R. Zhang, "Locomotion Activity Recognition Using Stacked Denoising Autoencoders," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2085–2093, Jun. 2018, doi: 10.1109/JIOT.2018.2823084.
[57] D. Ravi, C. Wong, B. Lo, and G.-Z. Yang, "Deep learning for human activity recognition: A resource efficient implementation on low-power devices," in *2016 IEEE 13th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, San Francisco, CA, USA, Jun. 2016, pp. 71–76. doi: 10.1109/BSN.2016.7516235.
[58] B. Zhou, J. Yang, and Q. Li, "Smartphone-Based Activity Recognition for Indoor Localization Using a Convolutional Neural Network," *Sensors*, vol. 19, no. 3, Art. no. 3, Jan. 2019, doi: 10.3390/s19030621.

[59] Y. Nan, N. H. Lovell, S. J. Redmond, K. Wang, K. Delbaere, and K. S. van Schooten, "Deep Learning for Activity Recognition in Older People Using a Pocket-Worn Smartphone," *Sensors*, vol. 20, no. 24, p. 7195, Dec. 2020, doi: 10.3390/s20247195.
[60] A. Hayat, F. Morgado-Dias, B. Bhuyan, and R. Tomar, "Human Activity Recognition for Elderly People Using Machine and Deep Learning Approaches," *Information*, vol. 13, no. 6, p. 275, May 2022, doi: 10.3390/info13060275.
[61] F. Moya Rueda, R. Grzeszick, G. Fink, S. Feldhorst, and M. ten Hompel, "Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors," *Informatics*, vol. 5, no. 2, p. 26, May 2018, doi: 10.3390/informatics5020026.
[62] K. S. van Schooten, S. M. Rispens, P. J. M. Elders, P. Lips, J. H. van Dieën, and M. Pijnappels, "Assessing Physical Activity in Older Adults: Required Days of Trunk Accelerometer Measurements for Reliable Estimation," *Journal of Aging and Physical Activity*, vol. 23, no. 1, pp. 9–17, Jan. 2015, doi: 10.1123/JAPA.2013-0103.
[63] A. Ignatov, "Real-time human activity recognition from accelerometer data using Convolutional Neural Networks," *Applied Soft Computing*, vol. 62, pp. 915–922, Jan. 2018, doi: 10.1016/j.asoc.2017.09.027.
[64] F. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, p. 115, Jan. 2016, doi: 10.3390/s16010115.
[65] A. Thaljaoui, T. Val, N. Nasri, and D. Brulin, "BLE localization using RSSI measurements and iRingLA," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, Mar. 2015, pp. 2178–2183. doi: 10.1109/ICIT.2015.7125418.
[66] C. Jayakody, S. Lokuliyana, D. Chathurangi, and D. Vithana, "Indoor Positioning: Novel Approach for Bluetooth Networks using RSSI Smoothing," *International Journal of Computer Applications*, vol. 137, pp. 26–32, Mar. 2016, doi: 10.5120/ijca2016909028.
[67] D. Čabarkapa, I. Grujić, and P. Pavlović, "Comparative analysis of the Bluetooth Low-Energy indoor positioning systems," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (ᵀᴱᴸSIKS)*, Oct. 2015, pp. 76–79. doi: 10.1109/ᵀᴱᴸSKS.2015.7357741.
[68] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012, doi: 10.3390/s120911734.
[69] X.-Y. Lin, T.-W. Ho, C.-C. Fang, Z.-S. Yen, B.-J. Yang, and F. Lai, "A mobile indoor positioning system based on iBeacon technology," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, Milan, Aug. 2015, vol. 2015, pp. 4970–4973. doi: 10.1109/EMBC.2015.7319507.
[70] M. Ji, J. Kim, J. Jeon, and Y. Cho, "Analysis of positioning accuracy corresponding to the number of BLE beacons in indoor positioning system," in *2015 17th International Conference on Advanced Communication Technology (ICACT)*, Jul. 2015, pp. 92–95. doi: 10.1109/ICACT.2015.7224764.
[71] S. Yamaguchi, D. Arai, T. Ogishi, and S. Ano, "Short paper: Experimental study of long-term operation of BLE tags for realizing indoor location based service," pp. 136–138, Mar. 2015, doi: 10.1109/ICIN.2015.7073819.
[72] D. Cabarkapa, I. Grujic, and P. Pavlovic, "Comparative analysis of the Bluetooth Low-Energy indoor positioning systems," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, Nis, Serbia, Oct. 2015, pp. 76–79. doi: 10.1109/TELSKS.2015.7357741.
[73] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems," *IEEE Trans. Syst., Man, Cybern. C*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007, doi: 10.1109/TSMCC.2007.905750.
[74] N. Cinefra, "An adaptive indoor positioning system based on Bluetooth Low Energy RSSI," Politecnico di Milano, 2014. [Online]. Available: http://hdl.handle.net/10589/92284
[75] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and L. Cuthbert, "Bluetooth positioning using RSSI and triangulation methods," in *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, Jan. 2013, pp. 837–842. doi: 10.1109/CCNC.2013.6488558.

[76] W. M. Yeung and J. K. Ng, "An Enhanced Wireless LAN Positioning Algorithm based on the Fingerprint Approach," in *TENCON 2006 - 2006 IEEE Region 10 Conference*, Hong Kong, China, 2006, pp. 1–4. doi: 10.1109/TENCON.2006.343696.
[77] S. S. Chawathe, "Beacon Placement for Indoor Localization using Bluetooth," in *2008 11th International IEEE Conference on Intelligent Transportation Systems*, Beijing, China, Oct. 2008, pp. 980–985. doi: 10.1109/ITSC.2008.4732690.
[78] P. H. Kindt, D. Yunge, R. Diemer, and S. Chakraborty, "Energy Modeling for the Bluetooth Low Energy Protocol," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 2, pp. 1–32, Mar. 2020, doi: 10.1145/3379339.
[79] B. Dawes and K.-W. Chin, "A comparison of deterministic and probabilistic methods for indoor localization," *Journal of Systems and Software*, vol. 84, no. 3, pp. 442–451, Mar. 2011, doi: 10.1016/j.jss.2010.11.888.
[80] "Apple Watch Series 4 - Technical Specifications," *Apple*, Mar. 24, 2021. https://support.apple.com/kb/SP778?locale=en_GB&viewlocale=en_US (accessed Jan. 19, 2023).
[81] Dr. B. Thomas, "SensorLog," *SensorLog*, Mar. 04, 2022. http://sensorlog.berndthomas.net/ (accessed Jan. 20, 2023).
[82] "pandas.DataFrame.dropna," *pandas 1.5.3 documentation*. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html (accessed Jan. 30, 2023).
[83] "pandas.DataFrame.drop_duplicates," *pandas 1.5.3 documentation*. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html (accessed Jan. 30, 2023).
[84] "pandas.DataFrame.nunique," *pandas 1.5.3 documentation*. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.nunique.html (accessed Jan. 30, 2023).
[85] R. Python, "NumPy, SciPy, and Pandas: Correlation With Python – Real Python." https://realpython.com/numpy-scipy-pandas-correlation-python/ (accessed Jan. 20, 2023).
[86] J. Magiya, "Pearson Coefficient of Correlation with Python," *Medium*, Nov. 23, 2019. https://levelup.gitconnected.com/pearson-coefficient-of-correlation-using-pandas-ca68ce678c04 (accessed Jan. 20, 2023).
[87] Y. Chen, "How to do Feature Selection/Dimension Reduction?," *Medium*, Jan. 13, 2019. https://yanlinc.medium.com/how-to-do-feature-selection-dimension-reduction-883c844aaaf6 (accessed Jan. 20, 2023).
[88] V. R. Joseph, "Optimal ratio for data splitting," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 15, no. 4, pp. 531–538, 2022, doi: 10.1002/sam.11583.
[89] P. Horky, A. Prokes, and P. Hubacek, "Unsupervised time series pattern recognition for purpose of electronic surveillance," in *2022 24th International Microwave and Radar Conference (MIKON)*, Gdansk, Poland, Sep. 2022, pp. 1–5. doi: 10.23919/MIKON54314.2022.9924999.
[90] "StandardScaler—Sklearn Preprocessing," *Scikit-Learn*. https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (accessed Jan. 20, 2023).
[91] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, Jun. 2000, doi: 10.1016/S0731-7085(99)00272-1.
[92] A. D. Dongare, R. R. Kharde, and A. D. Kachare, "Introduction to artificial neural network," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 2, no. 1, pp. 189–194, 2012.
[93] M. Şen, O. Cakar, and M. Aydin, *Estimation of Natural Frequencies and Buckling Critical Loads of Uniform and Sandwich Cantilever Beams by Using Artificial Neural Networks (ANN)*. 2017.
[94] R. De Oliveira *et al.*, "A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges," *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, vol. 16, pp. 628–645, Sep. 2017, doi: 10.1590/2179-10742017v16i3854.
[95] A. Shekar, C. Rebelo de Sá, H. Ferreira, and C. Soares, "Building robust prediction models for defective sensor data using Artificial Neural Networks," Apr. 2018.
[96] T. Liu, H. Xu, M. Ragulskis, M. Cao, and W. Ostachowicz, "A Data-Driven Damage Identification Framework Based on Transmissibility Function Datasets and One-Dimensional Convolutional Neural Networks: Verification on a Structural Health Monitoring Benchmark Structure," *Sensors*, vol. 20, no. 4, Art. no. 4, Jan. 2020, doi: 10.3390/s20041059.

[97] S.-M. Lee, S. M. Yoon, and H. Cho, "Human activity recognition from accelerometer data using Convolutional Neural Network," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb. 2017, pp. 131–134. doi: 10.1109/BIGCOMP.2017.7881728.
[98] K. Banjarey, S. P. Sahu, and D. K. Dewangan, "Human Activity Recognition Using 1D Convolutional Neural Network," in *Sentimental Analysis and Deep Learning*, Singapore, 2022, pp. 691–702. doi: 10.1007/978-981-16-5157-1_54.
[99] S. Verma, "Understanding 1D and 3D Convolution Neural Network | Keras," *Medium*, Apr. 05, 2022. https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610 (accessed Jan. 31, 2023).
[100] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, doi: 10.1162/neco_a_01199.
[101] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
[102] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: 10.1109/72.279181.
[103] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000, doi: 10.1162/089976600300015015.
[104] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," in *Proceedings of the 32nd International Conference on Machine Learning*, Jun. 2015, pp. 2342–2350. Accessed: Jan. 30, 2023. [Online]. Available: https://proceedings.mlr.press/v37/jozefowicz15.html
[105] A. Graves, A. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks." arXiv, Mar. 22, 2013. Accessed: Jan. 30, 2023. [Online]. Available: http://arxiv.org/abs/1303.5778
[106] J. Donahue *et al.*, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description." arXiv, May 31, 2016. Accessed: Jan. 31, 2023. [Online]. Available: http://arxiv.org/abs/1411.4389
[107] H. Wang *et al.*, "Wearable Sensor-Based Human Activity Recognition Using Hybrid Deep Learning Techniques," *Security and Communication Networks*, vol. 2020, pp. 1–12, Jul. 2020, doi: 10.1155/2020/2132138.
[108] K. S. Krishna and S. Paneerselvam, "An Implementation of Hybrid CNN-LSTM Model for Human Activity Recognition," in *Advances in Electrical and Computer Technologies*, Singapore, 2022, pp. 813–825. doi: 10.1007/978-981-19-1111-8_63.
[109] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, Illustrated edition. Cambridge, Massachusetts: The MIT Press, 2016.
[110] O. A. Montesinos López, A. Montesinos López, and J. Crossa, "Artificial Neural Networks and Deep Learning for Genomic Prediction of Continuous Outcomes," in *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, O. A. Montesinos López, A. Montesinos López, and J. Crossa, Eds. Cham: Springer International Publishing, 2022, pp. 427–476. doi: 10.1007/978-3-030-89010-0_11.
[111] Keras Team, "Keras documentation: EarlyStopping." https://keras.io/api/callbacks/early_stopping/ (accessed Jan. 30, 2023).
[112] W. Di, A. Bhardwaj, and J. Wei, *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing, 2018.
[113] H. Mahmood, "Softmax Function, Simplified," *Medium*, Nov. 26, 2018. https://towardsdatascience.com/softmax-function-simplified-714068bf8156 (accessed Jan. 30, 2023).
[114] "Softmax Function," *DeepAI*, May 17, 2019. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (accessed Jan. 30, 2023).
[115] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, Illustrated edition. Cambridge, MA: The MIT Press, 2012.
[116] K. Kobs, M. Steininger, A. Zehe, F. Lautenschlager, and A. Hotho, "SimLoss: Class Similarities in Cross Entropy." arXiv, Mar. 06, 2020. doi: 10.48550/arXiv.2003.03182.
[117] J. Brownlee, "How to Choose Loss Functions When Training Deep Learning Neural Networks," *MachineLearningMastery.com*, Jan. 29, 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (accessed Jan. 30, 2023).

[118] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
[119] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. doi: 10.48550/arXiv.1412.6980.
[120] "AdaMax," *Hasty.ai*. https://hasty.ai/docs/mp-wiki/solvers-optimizers/adamax (accessed Jan. 30, 2023).
[121] "Base Learning Rate," *Hasty.ai*. https://hasty.ai/docs/mp-wiki/solvers-optimizers/base-learning-rate (accessed Jan. 30, 2023).
[122] Keras Team, "Keras documentation: Adamax." https://keras.io/api/optimizers/adamax/ (accessed Jan. 30, 2023).
[123] "ESP32 Datasheet," *Espressif Systems*. https://www.espressif.com/en/support/documents/technical-documents (accessed Jan. 31, 2023).
[124] J. Yang, C. Poellabauer, P. Mitra, and C. Neubecker, "Beyond beaconing: Emerging applications and challenges of BLE," *Ad Hoc Networks*, vol. 97, p. 102015, Feb. 2020, doi: 10.1016/j.adhoc.2019.102015.
[125] Beaconstac, "BLE Eddystone beacon vs iBeacon? Beacon communication basics.," *Beaconstac*. https://www.beaconstac.com/ibeacon-and-eddystone (accessed Jan. 20, 2023).
[126] J. Seo, K. Cho, W. Cho, G. Park, and K. Han, "A discovery scheme based on carrier sensing in self-organizing Bluetooth Low Energy networks," *Journal of Network and Computer Applications*, vol. 65, pp. 72–83, Apr. 2016, doi: 10.1016/j.jnca.2015.09.015.
[127] H.-T. Chen, P.-Y. Lin, and C.-Y. Lin, "A Smart Roadside Parking System Using Bluetooth Low Energy Beacons," in *Web, Artificial Intelligence and Network Applications*, vol. 927, L. Barolli, M. Takizawa, F. Xhafa, and T. Enokido, Eds. Cham: Springer International Publishing, 2019, pp. 471–480. doi: 10.1007/978-3-030-15035-8_44.
[128] Z. Zhang, "Introduction to machine learning: k-nearest neighbors," *2016*, vol. 4, no. 11, p. 218, 2016.
[129] "What is the k-nearest neighbors algorithm?," *IBM*. https://www.ibm.com/topics/knn (accessed Jan. 31, 2023).
[130] "Naive Bayes," *scikit-learn*. https://scikit-learn/stable/modules/naive_bayes.html (accessed Feb. 01, 2023).
[131] T. M. Mitchell, *Machine Learning*, 1st edition. New York: McGraw-Hill, 1997.
[132] A. Christopher, "Support Vector Machine," *Medium*, Feb. 25, 2021. https://medium.datadriveninvestor.com/support-vector-machine-a37d1684d7b1 (accessed Feb. 01, 2023).
[133] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, New York, NY, USA, Jul. 1992, pp. 144–152. doi: 10.1145/130385.130401.
[134] S. Yıldırım, "Hyperparameter Tuning for Support Vector Machines — C and Gamma Parameters," *Medium*, Jun. 01, 2020. https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167 (accessed Jan. 31, 2023).
[135] "Support Vector Machines," *scikit-learn*. https://scikit-learn/stable/modules/svm.html (accessed Feb. 01, 2023).
[136] R. Gholami and N. Fakhari, "Chapter 27 - Support Vector Machine: Principles, Parameters, and Applications," in *Handbook of Neural Computation*, P. Samui, S. Sekhar, and V. E. Balas, Eds. Academic Press, 2017, pp. 515–535. doi: 10.1016/B978-0-12-811318-9.00027-2.
[137] "sklearn.svm.SVC," *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html (accessed Feb. 01, 2023).
[138] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New Support Vector Algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207–1245, May 2000, doi: 10.1162/089976600300015565.
[139] "Plot different SVM classifiers in the iris dataset," *scikit-learn*. https://scikit-learn/stable/auto_examples/svm/plot_iris_svc.html (accessed Feb. 06, 2023).
[140] Onesmus Mbaabu, "Introduction to Random Forest in Machine Learning," *Engineering Education (EngEd) Program | Section*, Dec. 11, 2020. https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/ (accessed Jan. 31, 2023).
[141] C. Ayuya, "Entropy and Information Gain to Build Decision Trees in Machine Learning," *Engineering Education (EngEd) Program | Section*, Jul. 03, 2021. https://www.section.io/engineering-education/entropy-information-gain-machine-learning/ (accessed Feb. 01, 2023).

[142] "sklearn.tree.DecisionTreeClassifier," *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (accessed Feb. 01, 2023).
[143] T.-H. Lee, A. Ullah, and R. Wang, "Bootstrap Aggregating and Random Forest," in *Macroeconomic Forecasting in the Era of Big Data: Theory and Practice*, P. Fuleky, Ed. Cham: Springer International Publishing, 2020, pp. 389–429. doi: 10.1007/978-3-030-31150-6_13.
[144] T. Yiu, "Understanding Random Forest," *Medium*, Sep. 29, 2021. https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (accessed Jan. 31, 2023).
[145] "sklearn.ensemble.RandomForestClassifier," *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Jan. 31, 2023).
[146] Z. Jiao, P. Hu, H. Xu, and Q. Wang, "Machine Learning and Deep Learning in Chemical Health and Safety: A Systematic Review of Techniques and Applications," *ACS Chemical Health & Safety*, vol. 27, pp. 316–334, Nov. 2020, doi: 10.1021/acs.chas.0c00075.
[147] D. Nelson, "Gradient Boosting Classifiers in Python with Scikit-Learn," *Stack Abuse*, Jul. 17, 2019. https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/ (accessed Feb. 01, 2023).
[148] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001, doi: 10.1214/aos/1013203451.
[149] "Ensemble methods," *scikit-learn*. https://scikit-learn/stable/modules/ensemble.html (accessed Feb. 01, 2023).
[150] D. Wetcher-Hendricks, *Analyzing quantitative data: an introduction for social researchers*. Hoboken, N.J: Wiley, 2011.
[151] A. Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien, "Linear discriminant analysis: A detailed tutorial," *AIC*, vol. 30, no. 2, pp. 169–190, May 2017, doi: 10.3233/AIC-170729.
[152] A. Araveeporn, "Comparing the Linear and Quadratic Discriminant Analysis of Diabetes Disease Classification Based on Data Multicollinearity," *International Journal of Mathematics and Mathematical Sciences*, vol. 2022, pp. 1–11, Sep. 2022, doi: 10.1155/2022/7829795.
[153] "Linear and Quadratic Discriminant Analysis," *scikit-learn*. https://scikit-learn/stable/modules/lda_qda.html (accessed Feb. 01, 2023).
[154] Sanjay.M, "Performance boosting with Voting-Classifier," *Analytics Vidhya*, Jan. 14, 2020. https://medium.com/analytics-vidhya/performance-boosting-with-voting-classifier-ea69313a367c (accessed Feb. 01, 2023).
[155] "Firebase Realtime Database," *Firebase Documentation*. https://firebase.google.com/docs/database (accessed Feb. 01, 2023).
[156] "Learn about using and managing API keys for Firebase," *Firebase Documentation*. https://firebase.google.com/docs/projects/api-keys (accessed Feb. 01, 2023).
[157] "Firebase Database REST API," *Firebase Documentation*. https://firebase.google.com/docs/reference/rest/database (accessed Feb. 01, 2023).
[158] "Gmail API Overview," *Google Developers*. https://developers.google.com/gmail/api/guides (accessed Feb. 01, 2023).
[159] R. Restrepo, "OAuth 2.0 Refresh Token Best Practices," *Blog • Stateful*, Apr. 13, 2022. https://stateful.com/blog/oauth-refresh-token-best-practices (accessed Feb. 01, 2023).
[160] "Using OAuth 2.0 to Access Google APIs," *Google Developers*. https://developers.google.com/identity/protocols/oauth2 (accessed Feb. 01, 2023).
[161] A. A. Sutchenkov and A. I. Tikhonov, "Active investigation and publishing of calculation web based applications for studying process," *J. Phys.: Conf. Ser.*, vol. 1691, no. 1, p. 012096, Nov. 2020, doi: 10.1088/1742-6596/1691/1/012096.
[162] D. Karade and V. Karade, "AIDrugApp: artificial intelligence-based Web-App for virtual screening of inhibitors against SARS-COV-2," *Journal of Experimental & Theoretical Artificial Intelligence*, pp. 1–49, Apr. 2022, doi: 10.1080/0952813X.2022.2058619.
[163] A. R. Kashyap and M.-Y. Kan, "SciWING -- A Software Toolkit for Scientific Document Processing." arXiv, Oct. 23, 2020. doi: 10.48550/arXiv.2004.03807.
[164] S. F. N. Islam, A. Sholahuddin, and A. S. Abdullah, "Extreme gradient boosting (XGBoost) method in making forecasting application and analysis of USD exchange rates against rupiah," *J. Phys.: Conf. Ser.*, vol. 1722, no. 1, p. 012016, Jan. 2021, doi: 10.1088/1742-6596/1722/1/012016.

[165] A. Aboah, M. Boeding, and Y. Adu-Gyamfi, *Mobile Sensing for Multipurpose Applications in Transportation*. 2021.
[166] M. Khorasani, M. Abdou, and J. Hernández Fernández, *Web Application Development with Streamlit: Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework*. Berkeley, CA: Apress, 2022. doi: 10.1007/978-1-4842-8111-6.
[167] "Streamlit-Authenticator, Part 1: Adding an authentication component to your app," *Streamlit*, Dec. 06, 2022. https://blog.streamlit.io/streamlit-authenticator-part-1-adding-an-authentication-component-to-your-app/ (accessed Feb. 01, 2023).
[168] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, Apr. 1960, doi: 10.1177/001316446002000104.
[169] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006, doi: 10.1016/j.patrec.2005.10.010.