Application of KNN for Fall Detection on Qualcomm SoCs

Purab Nandi(✉), Apoorva Bajaj, and K. R. Anupama

Birla Institute of Technology and Science, Pilani, K.K Birla Goa Campus, Goa 403726, India
{p20200056,f20180269,anupkr}@goa.bits-pilani.ac.in

Abstract. A fall of an elderly person often leads to serious injuries, even death. Many falls occur in the home environment, and hence, a reliable fall detection system that can raise alarms immediately is a necessity. Wrist-worn accelerometer-based fall detection systems have been developed, and there are various data sets available, but the accuracy and precision have not been standardized or compared; even where comparison does exist, it has been run on GPUs. No analysis of the workability of the models and the data sets on SoCs has been previously attempted. Though over the last few years, ML and DL algorithms have been increasingly used in fall detection, and there have also been some suggestions for the use of compressed modelling, there are no concrete statistics available to form this conclusion. In this paper, we attempt to understand why ML algorithms cannot run as-is on existing SoCs; We are using Snapdragon 410c to do our analytics as it is primarily used in Biomedical and IoT applications, has low power consumption and small form factor making it ideal for wearables. In this paper, we have used KNN to prove that ML cannot be used directly on SoCs. We are using KNN as it does not have any pre-training period, and it is a very simple algorithm that gives good accuracy. In this paper, we establish the need for model and data compression for fall detection if we must use ML or DL algorithms on SOCs. We have done this with statistical analysis across different data sets.

Keywords: SoC · Wearable · IoT · KNN

1 Introduction

Medical and healthcare advancements have increased the average human lifespan to over 80 years. Geriatric healthcare hence has become vital, and regular monitoring of parameters is required. Currently, visiting or in-house nursing staff do the monitoring of various parameters. This arrangement is expensive for a significant part of society; the past decade has witnessed substantial technological advances in IoT based wearable devices, machine and deep learning; these technologies are now increasingly used in the healthcare domain. IoT-based healthcare includes tracking patients, inventory management of drugs, collection of patient samples, authentication of the patients and staff, and automated data collection and storage. IoT in the healthcare domain enables all these applications due to the availability of various wearable and non-wearable devices. These

S. Spinsante et al. (Eds.): HealthyIoT 2022, LNICST 456, pp. 148–169, 2023.
https://doi.org/10.1007/978-3-031-28663-6_12

devices are used for remote diagnosis, prognosis, and treatment. Hence advances in IoT, embedded systems and ML are the catalysts in the development of geriatric healthcare systems. Such systems are available at a reduced cost for detecting anomalies and raising timely alerts, assistance and care when required. Such a system is essential in a country like India, with a rising population of seniors residing in isolation. Some of the common concerns of the geriatric population include falls, sleep apnea, hiatus hernia, and other respiratory disorders that do not have a surgical solution, and the primary cause is frailty. Medical literature [1] also indicates that these disorders generally compound into life-threatening disorders.

Falls - According to a WHO report, there is an exponential increase in the frequency of falls with frailty and age. The elderly population in nursing homes are more susceptible to falls than those in the community. Falls usually result in recurrent falls. WHO data validates these statistics; It indicates that 40% of the falls among the elderly are recurrent falls.

Sleep Apnea - Sleep apnea is prevalent in adults and a small percentage of juveniles. Subjects suffering from sleep apnea experience periods of shallow breathing while asleep. Periods of shallow breathing or air flow reduction are called a hypopnea. This condition ultimately leads to apnea which causes breathing to stop temporarily. Both these conditions can cause clinical co-morbidity.

Hiatal Hernia -Hiatal hernia is when the stomach bulges up into the chest via an opening in the diaphragm. The muscles that separate the diaphragm from the stomach are called the hiatus. There are mainly two types of hiatus hernia (i) sliding (ii) paraesophageal.

Most elderlies suffer from the former type of hernia, which is inoperable. Hiatus hernia causes a person to exasperate their food which causes a drop in the O_2 level; SpO_2 sensors can detect the drop in O_2 level, which prevents the need for complex and costly diagnostic processes like polysomnography.

Sensors can be used to monitor the above three health conditions, and they can relay data to a smart ML-based system that can detect and predict the condition. In this paper we concentrate on fall-detections, the causes of falls are primarily categorized into internal and external. External causes of falls are due to environmental factors like slippery surfaces. Internal causes include cramps, weakness in the muscular-skeletal structure, vision impairments, chronic disorders, etc. The duration of the fall is also important. According to [2], about 40% of the individuals who fall cannot get up on their own, and about 50% who experience a long fall are likely to die within the next few months. A long-duration fall can also result in localized muscle injury, tissue damage, nerve issues, dehydration, hypothermia, pneumonia and a fear of further falls. These conditions affect the overall health of the elderly. Although numerous studies [3] related to fall detection have been out recently, several challenges still exists. These include:

(a) The lack of a comprehensive analysis of ML techniques deployed to detect falls (b) A high number of false positives (c) The low accuracy of systems that are expected to detect and correct such false positives (d) The inability of the system to detect the duration of the fall.

With technological advances in SoCs and IoT systems, wearable devices have emerged as a leading area of research in geriatric health care; the amount of data collected

at homes/hospitals for the elderly is large and complex and making accurate decisions based on multiple parameters is the primary requirement for such safety-critical systems. SoCs are resource-constrained devices, whether in terms of memory, processing power or energy constraints. Hence, they cannot implement ML algorithms or Deep Neural Networks. Hence a possible solution for this is the use of model compression. This paper picks up one of the simplest ML algorithms and runs them on varying data sets of different sizes to prove that it is impossible to run even a simple ML algorithm such as KNN when the data set size is considerable. Not only is the latency high, but as the size of the data set increases, the SoC fails, indicating the non-availability of the required resources. This paper proves that ML algorithms give inaccurate, non-reliable and high latency results when run on a raw data set. Hence this paper builds a case for using model compression algorithms while using SoCs.

The paper is organized as follows: Sect. 2 talks about various IoT architectures that can be used for fall detection, Sect. 3 gives a brief overview of ML algorithms used for fall detection, Sect. 4 provides the operational details of KNN, while Sect. 5 gives the details of the dataset used for analysis, we present our results in Sect. 6, while Sect. 7 summaries and concludes this paper.

2 Architectural Models for IoT Based Fall Detection Systems with Wearable End Devices

With the growth of SoCs and its integration into IoT systems, wearable healthcare devices are now a focused area of research. This section presents four possible IoT architectural models that can be used for health care. The models vary in terms of how the data is gathered, processed and where the conversion of data to knowledge occurs.

Model A- In this architectural model, the data is collected at regular sampling intervals from the sensors that are interfaced with the wearable devices. The wearable device then transmits the raw data to the coordinator; the co-ordinator then collects the data from multiple wearables, aggregates the data and forwards the data to the cloud. The actual analysis and processing of data take place on the cloud. The sensor fusion algorithms that require combining data from multiple sensors also run on the cloud. The essential features are then extracted from the fused data and converted to health parameters using ML or DL algorithms. In this architectural model, data storage, as well as the processing of data, is done on the cloud. The data processing may be for emergency or long-term monitoring. The end devices are minimalistic and constrained in terms of processing and memory capabilities. The coordinator only acts as an aggregator of data. The focus of this architecture is on developing networking protocols that can deliver the data to the cloud with minimum latency and control overhead.

Model B –Similar to the case of model A, the wearable end device collects the raw data and transmits it to the coordinator. The coordinator aggregates the data and performs multi-sensor fusion on the raw data. The fused data is then sent to the cloud by the coordinator, the cloud performs feature extraction and converts the data to health parameters using ML/DL algorithms. In this model, while network protocols are also important, the architecture of the coordinator also needs to be carefully considered. Since sensor fusion is performed at the coordinator, the coordinator must be a powerful

SoC. The advantage of this model would be that the amount of communication between the coordinator and the cloud is reduced. This reduces the communication bandwidth requirement, but the actual conversion of data to knowledge still happens on the cloud. Model A and B may not be suitable for emergency response due to the latencies involved in communication. If the coordinator cannot connect to the cloud, any falls detected cannot immediately trigger an alarm (Figs. 1, 2, 3 and 4).

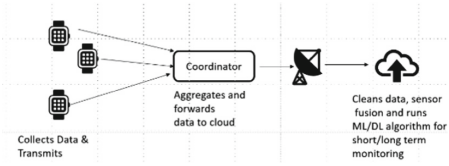

Fig. 1. Model A

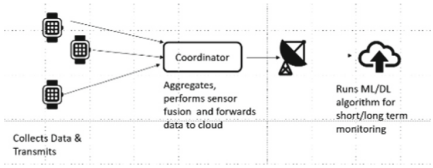

Fig. 2. Model B

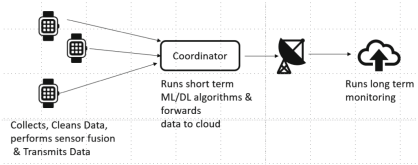

Fig. 3. Model C

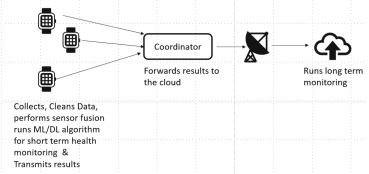

Fig. 4. Model D

Model C- In the case of architectural model C, the wearable end device is more powerful as it not only collects data from multiple sensors but also runs sensor fusion algorithms. The wearable device then transmits these fused data to the coordinator. In many cases, the sensor fusion algorithm that runs on the wearable device is more statistical in nature. The end device in model C must be a powerful microcontroller that can run complex mathematical and thresholding algorithms. Using ML algorithms, the coordinator then converts the fused data to short-term health parameters. In case of any falls or lie-ins detected, the coordinator raises the alarm. The cloud then runs only the ML and DL algorithms that are required for long-term health monitoring and rehabilitation. In the case of model C, the end device needs to have more processing power and at the same time also be able to handle complex network algorithms to transmit the fused data to the coordinator. The latencies in communication between the end device and the coordinator still exist even though the latency in raising the alarm is considerably reduced. Here data processing is done by all elements of the IoT model. While the end device performs sensor fusion, ML/DL algorithms to extract information from the fused data is run at the coordinator while the cloud converts data into long-term health parameters.

Model D – In the case of architectural model D, the wearable device is built using a powerful SoC as it is responsible for collecting data from multiple sensors, fusing it, and running ML and DL algorithms to detect/predict falls. This requires considerable processing power; The wearable device is required to collect and clean the data, perform sensor fusion, extract the required features and then convert the data into information using a complex ML/DL algorithm. Though SoCs have considerably advanced to handle complex biomedical applications, they are still constrained in terms of the amount of

memory available, energy consumed and form factor. As the device is wearable, the form factor must be very less. At the same time, power consumption must also be limited. Heat dissipation is another issue that is common in wearable devices. Running complex processing algorithms will cause the processor to expend more heat. Hence, running ML, DL, or deep neural networks widely used in IoT-based health services is very difficult. Over the last couple of years, research in model compression of ML and DL algorithms has gained traction. The goal of model compression is to achieve a model that is simplified as compared to the original but provides the same level of accuracy as the original algorithms. The advantage of running a reduced model means that fewer or smaller parameters need to be stored in the memory, as not only is the data resident in memory, but the operating system, as well as the code, is also resident in the memory. It is also expected that the processing latency would be reduced, allowing the model to predict in a shorter duration. In model D, the coordinator again acts as a forwarder of information, and the cloud runs long-term health monitoring and rehabilitation algorithms. The advantage of having the SoCs run the compressed algorithms is that alarms can be raised in case of falls even when no network connection is available. This reduces the lie-in period after the fall, reducing complex health situations that might arise due to long lie-in periods.

Model D and Available Wearable Devices in the Market
IoT applications are classified into different levels based on the complexity of the application and the complexity of the elements used to build them. In the case of model A,B and C described in this section, data storage and analytics is not done on the wearable device. Health monitoring systems usually collect a large amount of data from multiple sensors. The size of the data is large and also requires complex data analytics. Hence using the usual classification of IoT systems, the data storage and analytics must be primarily performed on the cloud. While this model works well for long-term health monitoring and rehabilitation, it will not be suitable for emergency services. In our suggested model D, we store some of the data and run the complex data analytics on them to handle an emergency such as falls; hence each end device has a very powerful SoC at its core. We plan to use SoCs such as Qualcomm Snapdragon 410c/820c/wear 4100 series that are built explicitly for biomedical applications. Running ML/DL applications especially requires high processing power. Hence if the analytics for emergency care must be performed on wearable devices, we need to use compressed ML algorithms.

We have reviewed multiple fall detection-based systems [4] available commercially and under theoretical research. In the following subsection, we give a brief overview of such fall detection systems.

Commercially Available Systems and Their Applications
Apple watch SE or series 4 [5] and above can detect hard falls. These services are enabled by default if you are above 55 years of age and are available as an optional application for those between 18 to 55. The "apple watch fall detection app" can help connect users to emergency services while sending messages to their emergency contacts. Apple Watch can detect only hard falls. It uses accelerometer and gyroscope data to detect a fall. It uses the impact acceleration and the resultant wrist trajectory for fall detection. In order to detect falls, it uses thresholding technology on the data, and no ML/DL algorithms

are run on the wearable system. Apple Watch also detects if a person is immobile for 60 s, it then begins a 30 s counter that starts an audio alert. The audio alert keeps getting louder until emergency services press "cancel". Despite the availability of such features, experimental data show the accuracy is only 4.7%, it has a false-negative rate of 95.3% and an interesting point is also that Apple watches are better at detecting forward falls than sideways falls because the wrist movement in sideways fall is equivalent to lying down in bed.

Another smart wearable device available is the "Unali Kanega" watch [6], another wrist-based device available for fall detection. It also makes use of accelerometer data to detect falls. The Unali watch is unique because the user can change the battery while still wearing the watch. This is a useful feature since falls may occur when the user removes his watch to charging.

There is also the Phoenix watch available which has an app called WellB Medical Alert plus that sends out the GPS location of the fallen person as long as he presses the button.

Other than the wearable systems available commercially, there are also applications which can run on the mobile. The summary of the applications and their capability is listed in the table given below. All the applications require that a user either presses a button or uses some form of an audio alert. The application will only provide the GPS location of the person. (Wherever GPS+ is mentioned in the table, it also uses Wi-Fi information to detect the person's position).

Other than these, there are also the popular "fall call lite application" [7] which usually runs on the Watch Operating Systems. Here the user must press a button and call for help when he falls. These applications are not very popular as they require that the person is still conscious and able to raise an alert.

The Wearable Devices Under Research

[8] talks about a smart vest that can monitor respiratory and physical activities. The M-health platform [9] described as part of the "Frail" project has a smart vest, fall sensors, and a smartwatch. The sensing platform aims to address the continuous monitoring of vital signs relevant to frail users and detecting and alerting falls. The smart watch worn by the user acts as the gateway to the platform, gathering data from sensors and receiving events and remainders introduced by caregivers.

Since the smartwatch is responsible for communication with the frail servers, the end devices, the sensing platforms, need only to send the sensor data. Hence this falls under model C of the IoT architecture described in Sect. 2. For fall detection, mainly accelerometer-based devices are used. Also, after the accelerometer detects the fall, the smartwatch expects the wearer to confirm he/she has fallen. If the user confirms the fall, the smartwatch sends the event to the frail servers back and then triggers a preconfigured procedure. The sensor module used is a tri-axial accelerometer, and the processing module is a PIC18 F2431 Microcontroller which uses a thresholding method to detect falls. The sensors are placed as an adhesive patch on the skin of the lower back. Again, this system does not use multiple sensor data or any machine learning algorithm to detect falls.

Table 1. The summary of commercially available wearables

Product	Automaticfall detection	Location capability	Battery life
GreatCall *Lively Mobile Plus*	Yes	GPS	1–3 days
Philips Lifeline *GoSafe 2*	Yes	GPS+	2–3 days
Medical Guardian *Active Guardian* (rebranded version of the Freeus *Belle+*)	Yes	GPS+	up to 5 days
LifeFone *At home, On-the-Go GPS, Voice in Pendant* (rebranded version of the Freeus *Belle+*)	Yes	GPS+	up to 5 days (30 days if no fall detection capability)
LifeFone *At home, On-the-Go GPS* (rebranded version of the MobileHelp *Duo*)	Yes	GPS	1 day (mobile base station) pendant: long
MobileHelp *Duo*	Yes	GPS	1 day (mobile base station) pendant: 18 months
Medical Guardian *Mini Guardian*	Yes	GPS+	up to 5 days

Further research analysis shows that most fall detection systems use Model A, while the rest may use Model B, where sensor fusion is done on the coordinating device. This will be the first attempt to build a wearable SoC device that runs compressed ML\DL algorithms that provide auto alerts for emergency help.

3 Machine Learning

ML [10] is a technique that applies mathematical models to data sets to analyze, classify and convert data into knowledge. There are three types of ML algorithms.

1. Supervised – In supervised learning, the input data is classified apriori using a training data set; any new data is automatically classified into one of the input types, some of the algorithms include KNN [11], Naïve Bayes [12], Decision trees [13], Linear Regression [14], SVM [15].
2. Unsupervised – In unsupervised learning, the ML algorithm recognizes a pattern on its own from a given data set, some of the common algorithms include K-Means clustering [16], Classification rules [17], Hidden Markov model [18], Neural Networks [19].

3. Reinforced -This algorithm allows the system to adapt its behavior based on feedback from the environment.

In the case of fall detection, binary classification is used to convert an activity into a fall or ADL. The diagram given below shows how the ML model is built.

In each category of ML, there are several algorithms, as shown in Fig. 5.

Fig. 5. Schematic of ML Model

For fall detection, several ML algorithms are currently being used. Commonly used algorithms are Logical Regression, Naïve Bayes, SVM, K Nearest Neighbor and Random Forest. Among these algorithms, this paper concentrates on the KNN algorithm. We had earlier run multiple ML algorithms for fall detection. The AUC/ROC curve for them is shown in the figure below.

The algorithms were run on the data set that we had collected. The dataset had over 70k points, of which 70% was used for training; and 30% was used as test data. KNN has a good AUC score of 0.971, performing as well as SVM. We have used KNN to analyze the latencies involved in the ML algorithm as the accuracies of KNN are good, and KNN does not require any pre-training until a query is raised. Hence KNN is the best model to understand the effect of implementing ML on SoCs as there will be no heavy pre-training required, unlike the other algorithms. Though Naïve-Bayes is easier to implement. Its AUC score is only 0.825, which is very low when compared to KNN (Fig. 6).

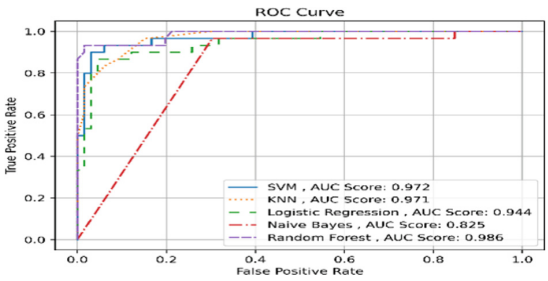

Fig. 6. AUC/ROC curve of ML algorithms for fall detection

4 K Nearest Neighbor Classification Algorithm (KNN)

The KNN algorithm [20] is a general classification algorithm that uses Nearest Neighbor rules to classify data. It compares the input data with K samples with the same class label, finds the nearest one, and classifies the input data accordingly. It differs from the Nearest Neighbor algorithm in that, instead of a single neighbor, it takes K nearest neighbor in the decision-making process. This allows the KNN algorithm to utilize more information for data classification. It also eliminates the process of learning when compared to other classification algorithms. The KNN schematic diagram for classification is shown in the figure below. The decision-making process is straightforward in KNN. The input data is compared to the sample class close to it (Fig. 7).

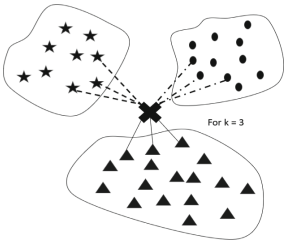

Fig. 7. Classification Schematic for k = 3

The classification is done based on the distance metric, as shown in the figure above. There are multiple methods available to calculate the distance in KNN. The most commonly used methods are (a) Euclidean, (b) Manhattan, (c) Hamming (d) Minkowski [21].

The Euclidean distance is calculated using the formula given below.

Euclidian distance

$$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2} \tag{1}$$

The Manhattan distance is the distance between two points measured along the axis at right angles and uses the following formulae for calculating the distance.

Manhattan distance

$$d = |x_2 - x_1| + |y_2 - y_1| \tag{2}$$

The Hamming distance uses the number of BITS dataset which differ between the two-binary data, the distance between the data is given by

Hamming distance

$$d = \sum |A_i - B_i| \tag{3}$$

The Minkowski distance comes somewhere between Euclidean and Hamming. The Minkowski distance is given by

Minkowski distance

$$d = \sum_{i=1}^{n} (x_i - y_i)^{\frac{1}{p}} \tag{4}$$

5 Dataset and Data Preparation

The size of the data set, the data collection methodology used, the nature of the sensors, and the sampling rate affect the model's output. Research in fall detection has advanced over the last few years due to the increase in the elderly population who live alone. Multiple fall data sets are available, and various data sets [22] have been analyzed in detail. When coming to falls, the sensors used for fall detection can be (a) Biological, (b) IMU, (c) Image (d) Ambient sensors. While the first two fall under the wearable sensor category, the latter are non-wearable. Biological sensors might be heartrate sensors, GSR sensors, SpO_2 sensors or sensors based on the previous health history of the person. IMU sensors include a 3D accelerometer and gyroscope. Image sensors are monochrome/RGB/Thermal cameras/IR cameras, which detect falls. Ambient sensors are usually placed around the entire room occupied by the elderly. This includes radar and acoustic sensors, which are used for detecting falls.

Wearable sensors are preferred because they can follow the elderly through their entire daily routine. Multiple wearable sensors are available, primarily IMU-based sensors, most of which use 3D accelerometers. 80% of the data set available currently have waist-worn accelerometer sensors. Some supplement the waist-worn sensors by placing the sensors on the thigh or leg of the sensor. Few data sets place sensors all over the torso. Wearing sensors on the waist and the torso can be highly uncomfortable for the elderly as these sensors have to be worn for the entire day. In this paper, we focus on wrist-worn sensors as we strongly believe a wrist sensor is more convenient for an elderly user, and the sensor data can constantly be augmented appropriately using mathematical models.

Furthermore, there are multiple fitness bands available in the market that provide IMU and heart rate, blood pressure, and oxygen levels, and these devices can very quickly be adapted for fall detection. Fitness bands come with their own SoCs, so we believe we can adapt ML algorithms to be run locally on the device and the data to be stored locally for short-term health emergencies such as falls. We examined multiple datasets which used 3D accelerometers and looked mainly at data sets where the sensors were worn on the wrist. Among these, Smartwatch, SmartFall and Notch datasets are available publicly, so we used them. We also collected data using ten volunteers wearing a TIC watch (BITS dataset).

Data Collection Methodology for BITS Dataset

We gathered our data using ten volunteers wearing a TIC watch which included a 3-axis accelerometer, 3-axis magnetometer, 3-axis gyroscope and optical heart rate sensor.

Experiments were performed in a controlled environment in 20 different ADL/fall activity simulations, such as walking, running, climbing stairs, abrupt movements, and various types of falls. Using a TIC Watch, data were collected at four samples/second, the maximum possible frequency.

The volunteers were aged between the ages of 20–22 years. Their height ranged from 5′1” to 5′8”, and weight from 40 kg to 75 kg.

Experiments were performed across 20 different ADL/fall activity simulations, such as walking, running, climbing stairs, and abrupt movements. The volunteers simulated the following activities: (a) Walking slowly (b) Walking quickly (c) Jogging (d) Climbing

up and down a flight of stairs (e) Slowly sitting on a chair, waiting a moment, and standing up slowly (f) Quickly sitting on a chair, waiting a moment, and standing up quickly (g) Trying to transition from sitting to standing position but collapsing midway (h)Transitioning from sitting to lying and back, slowly (i) Transitioning from sitting to lying and back, quickly (j) Transitioning from sideways position to one's back while lying down (k) Standing, about to sit down, and getting up (l) Stumbling while walking (m) Slowly jumping without falling (n) Swinging hand (o) Falls – forward, backward, left lateral, right lateral (p) Grabbing while falling (q) Spinning fall.

To collect the data, the TIC watch was programmed to collect the data and send it via the user interface to the system. The data was automatically moved into a.csv file. The user interface of the software had buttons for every type of activity that was performed, so the corresponding button was selected before performing a particular activity. This enabled automatic labelling of the dataset as it got stored.

Samples on IMU and heart rate were collected during 14 ADLs and 6 falls, with each activity/fall, repeated twice. All activities were conducted at the BITS Pilani, K K Birla Goa campus. The falls were simulated in a controlled environment. The anechoic chamber at BITS Pilani, K K Birla Goa campus was used for this purpose. The chamber, 4.5 m $\times$ 2.2 m $\times$ 2.5 m, is padded with NRL USA standard 8093 complying material on all four walls, the floor, and the ceiling. This padding provided the necessary shock absorption capabilities to protect volunteers from harm during the experimentation.

The volunteer's rebound and residual movements after a fall activity were also considered to ensure that the data set also contains post-fall values of the IMU and heart rate parameters. Hence for falls, the data collection was stopped not immediately but a few seconds after the actual event occurred, during which time the volunteers performed post-fall movements such as rolling over and attempting to get up. The data set generated had, in total, over 110,000 lines about the ADLs as mentioned above and falls.

In case of the BITS data set though we collected data using a 3 axes accelerometer, magnetometer and gyroscope as well as an optical heart rate sensor, we used only the data from 3 axes accelerometer to study the accuracies over the other data sets. This changed the number of data points from 110,000 to 47,656.

Data preparation for SmartFall, Smartwatch and Notch

Smartwatch Dataset [23]- Smartwatch had 7 subjects with the age range of 21 to 55 who perform 971 different activities. Each data was collected at a sample rate of 31 Hz. In order to match it with our data set we downscale it to 20 kHz using the python scipy 1.2.3 package overall there were 34,019 data points.

Notch Dataset [24]- Notch data set have only 7 subjects in the ages ranging from 20–35. Overall notch had 10,645 data points; the data set was also collected at 31 Hz which was eventually downscaled to 20 Hz to match our data set.

SmartFall Dataset [25] - Smartfall had more subjects (14) covering a wider age range of 21 to 60, covering 1027 activities and 92,780 data points. Again, as in case of Smartwatch and Notch, we down sample the smartwatch data to 20 Hz.

Fused Dataset - We also fused the data set of SmartFall, Smartwatch and Notch as all 3 of them use the same accelerometer and a sampling rate of 31.25 Hz. This gave us overall 2496 activities extended over 28 subjects in the age range of 20–60. We combined the data set as we wanted to understand the effect of the size of the data set on the performance of the ML algorithm, especially in terms of latency, as we implemented it on the Qualcomm Snapdragon 410c SoC.

A summary of all data sets is shown in the Table 2.

We did not combine the BITS data set in the fused data set as a completely different accelerometer was used to collect data. The data ranges were completely different, and the ML algorithms would have given entirely incorrect results. We did not upscale or downscale the data because no mathematical relationship could be derived from the data sets due to the use of entirely different sensors. Any min-max scale or thresholding would have required that we examine over 100,000 data points to look for similarities between various falls and non-falls events. This would have required a considerable performance and computational complexity tradeoff (Table 1).

Table 2. Summary of datasets used in this paper

Dataset	Activities	Data points	Test subjects	Age
BITS	3000	47,656	10	20–22
Notch	698	10,645	7	20–35
SmartWatch	771	34,019	7	21–55
SmartFall	1027	92,780	14	21–60
Fused	2496	137,444	28	20–60

6 Results and Discussions

To understand the working of ML on an SoC, we choose to experiment with KNN as the ML model. As mentioned in Sect. 4, the primary advantage of KNN is that no pre-training is required until a query is raised. As described in Sect. 5, we worked with multiple data sets such as a Smartwatch, Smartfall and Notch. This was to understand the effect of data collection and results of the data set on the accuracy of the ML model. Furthermore, to understand the limitations of the data set size, that the SoC could work while we used the clean data set from the Smartwatch and Smartfall and Notch all of them down sample to 20 Hz; we also used the data collected by us in both the raw and clean format. Data cleaning or statistical analysis of data is generally performed on the cloud. If we collect data and send it to the cloud for cleaning, the network latency remains; hence, running the ML algorithms on the SoC has no benefit. Hence the cleaning of the data should also happen on the SoC itself. When we use raw data, the SoC is unable to proceed further. More details are given below in this section.While analyzing the various datasets and the effect of running the algorithm on an SoC, we varied the values of k from 1-to 200. In some cases, we have looked at a narrower data window with k values

varying between 17–69 because that is where the maximum variation was observed. We also ran the accuracy for three different distance equations, Euclidian, Minkowski and Manhattan. When we attempted to run the algorithm with Hamming distance, the accuracies were considerably lower. As most of our data is 64-bit floating-point format and there are minimal variations in mantissa values, Hamming does not work well with the kind of data set we are using. We also analyzed the latencies of the various datasets. We ran the algorithms for varying values of k on Apple M1 systems and Qualcomm Snapdragon 410c. We ran the algorithm on the M1 system as a single thread since the SoC also runs the algorithm as a single thread. The apple M1 pro was running at 3.22 Ghz, and the 410C SoC was running at 2.40 GHz.

The plots of our various results are given below:

Figures 8 and 9 give the accuracy and latencies of Smartwatch on M1 pro.

Figures 10 and 11 give the accuracy and latency of Smartwatch on Qualcomm 410c.

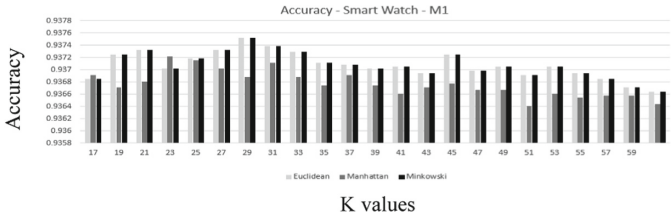

Fig. 8. Accuracy of Smartwatch dataset on Apple M1 pro

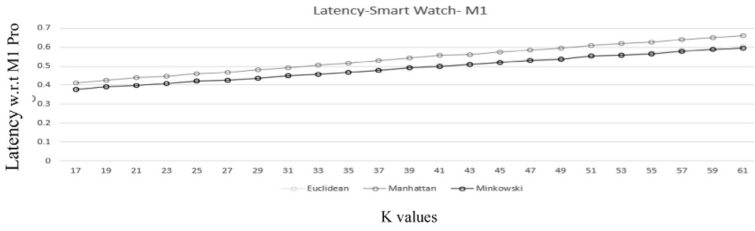

Fig. 9. Latency on Smartwatch dataset on Apple M1 pro

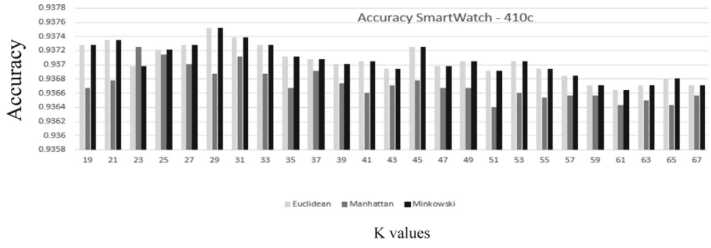

Fig. 10. Accuracy of Smartwatch dataset on Qualcomm 410c

Analysis of Smartwatch Dataset Results

In the case of Smartwatch, there are 771 data taken from 7 subjects over ages 21 to 55. The details are given in Sect. 6. The best results were obtained at a k value of 29 for

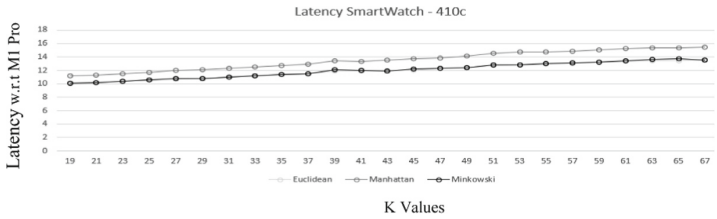

Fig. 11. Latency on Smartwatch dataset on Qualcomm 410c

both Euclidean and Minkowski, while Manhattan gave its best result at a slightly lower value of k = 23. This is a trend that can be observed over various datasets that Manhattan gives higher accuracies at a lower value of k; this is because of the way the distance is calculated in the case of Manhattan, which is the sum of the absolute difference of the cartesian co-ordinates of the data. It can be seen from the latency plots that the SoCs take about 10.5 s in case of the highest accuracy for both Euclidean and Minkowski are about 11.75 s for Manhattan.

The accuracy of Minkowski (93.7519%) and Euclidian (93.7519%) is slightly higher than that of Manhattan (93.7213%). Where as in the case of M1 Pro, the time taken is 0.45 s for all the algorithms. The data sampling rate in case of all our data is 20 Hz, meaning we collect a sample every 0.25 s and if the time taken to run the algorithm is between 10 to 12 s, even at a frequency of 2.4 GHz, and with a simplistic ML algorithm such as KNN, it is not possible to run an ML algorithm realtime on the SoC. The number of data points on Smartwatch is 34,019. Though it covers only 771 activities. The latencies required on the SoC are extremely high.

Analysis of Smartfall Dataset Results

In the case of Smartfall size of the data set is 92,780 data points covering 1027 activities. The best performance for Euclidian and Minkowski is obtained at a k value of 25 and in the case of Manhattan at a k value of 21. As in the previous 2 cases, the best performance of Manhattan (93.5186%) is slightly lower than that of Euclidian (93.5312%) and Minkowski (93.5312%). The latencies, as expected, are higher as the data set size is more extensive. Euclidian and Minkowski's peak performance, the latency incurred by M1 pro is 0.65, and in the case of Manhattan, it is 0.7. The latencies in the case of 410c for Euclidean and Minkowski is 21 s and 23 s for Manhattan.

Figures 12 and 13 give the accuracy and latency of Smartfall on M1 pro.

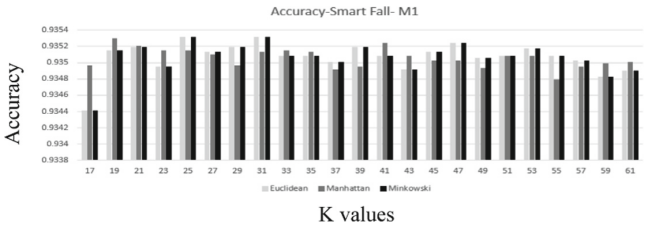

Fig. 12. Accuracy of Smartfall dataset on Apple M1 pro

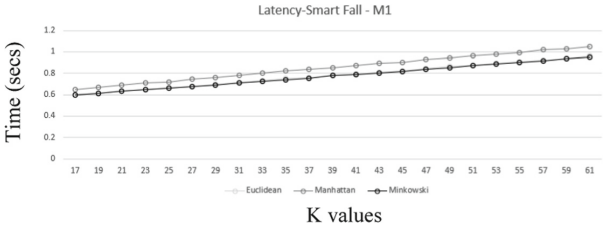

Fig. 13. Latency on Smartfall dataset on Apple M1 pro

Figures 14 and 15 give the accuracy and latency of Smartfall on Qualcomm 410c.

Analysis of Notch Dataset Results

The Notch data set has 10,645 data points covering overall 698 activities. This is by far the smallest data set that we have used. The accuracy of the Notch data set is at $k = 35$ for both Euclidean and Minkowski; in the case of Manhattan, the maximum accuracy is observed at $k = 31$. The latencies of the Notch dataset, in the case of both Apple M1 processor and 410c, are slightly lesser as the dataset size is also lesser. In the case of Euclidean and Minkowski at the peak performance, the latency on the M1 processor for the Notch data set is at 0.4 s, and for Manhattan, it is at 0.45 s.

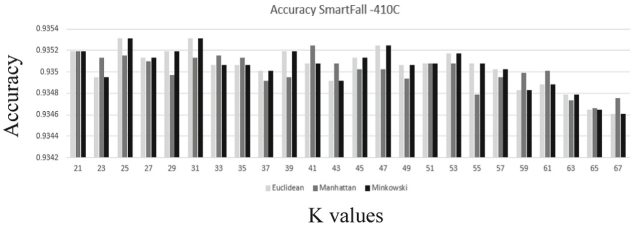

Fig. 14. Accuracy of Smartfall dataset on Qualcomm 410c

While in the case of 410c at the peak performance, the latencies are 10 s for Minkowski and Euclidian and 12 s for Manhattan. In the case of the Notch data set as well, the accuracy of Manhattan (91.8365%) is slightly lesser than that of Euclidean (91.8485%) and Minkowski (91.8485%).

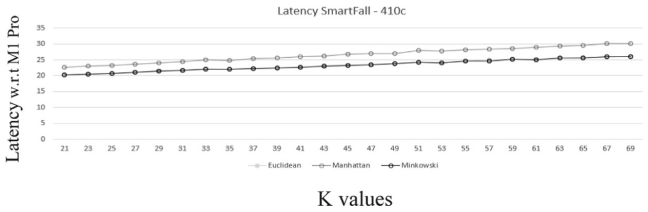

Fig. 15. Latency on Smartfall dataset on Qualcomm 410c

Figures 16 and 17 give the accuracy and latency of Notch on Apple M1 pro.

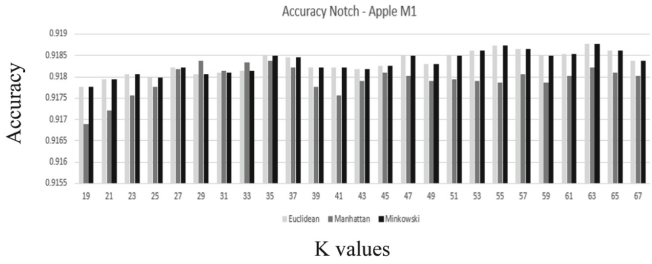

Fig. 16. Accuracy of Notch dataset on Apple M1 pro

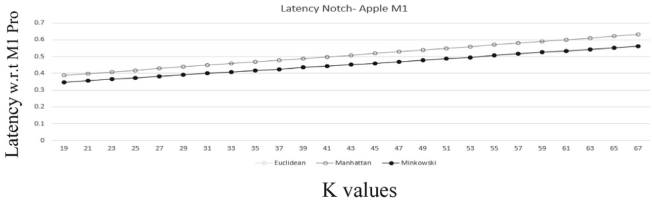

Fig. 17. Latency on Notch dataset on Apple M1 pro

Figures 18 and 19 give the accuracy and latency of Notch on Qualcomm 410c.

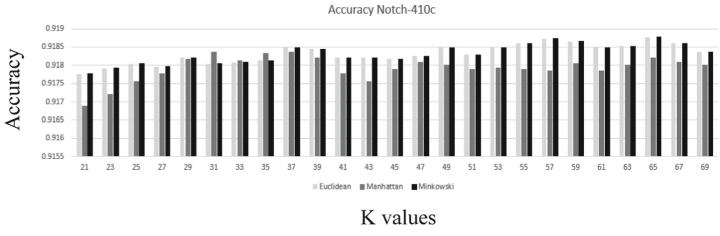

Fig. 18. Accuracy of Notch dataset on Qualcomm 410c

Analysis of Fused Dataset

As described in Sect. 5, we fused the data of Smartfall, Smartwatch and Notch generating 137,444 for 2,496 activities. The fused data set gave the best accuracy for Minkowski and Euclidean at a very high value of k = 49, where as the best results for Manhattan was at a k value of 25, it can be seen from this result that the number of data points have a bigger effect while using Minkowski and Euclidean methods when compared to Manhattan.

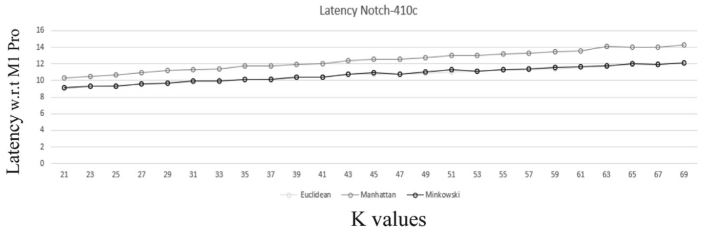

Fig. 19. Latency on Notch dataset on Qualcomm 410c

The accuracy again for the combined data set was slightly higher for Minkowski (92.9587%) and Euclidean(92.9587%)as compare to Manhattan(92.9526%). The latencies are extremely high as expected when compared to previous non fused data sets. For M1 pro, for the peak value of accuracy is at 0.75s and in case of Minkowski and Euclidean and for Manhattan the peak accuracy is observed at 0.8 s.

In case of 410c the latency is approximately 23 s in case of Minkowski and Euclidean and 24 s in case of Manhattan.

Figures 20 and 21 give the accuracy and latency of fused data-set on Apple M1 pro.

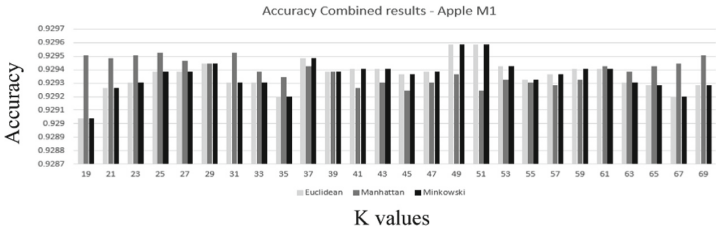

Fig. 20. Accuracy of combined dataset on Apple M1 Pro

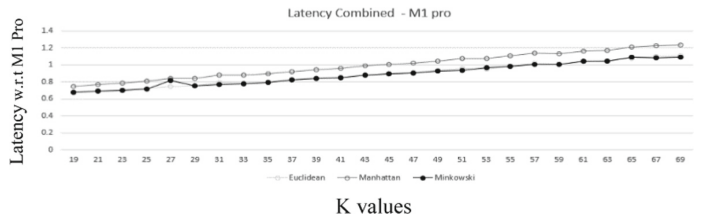

Fig. 21. Latency on combined dataset on Apple M1 Pro

Figures 22 and 23 give the accuracy and latency of combined dataset on Qualcomm 410c.

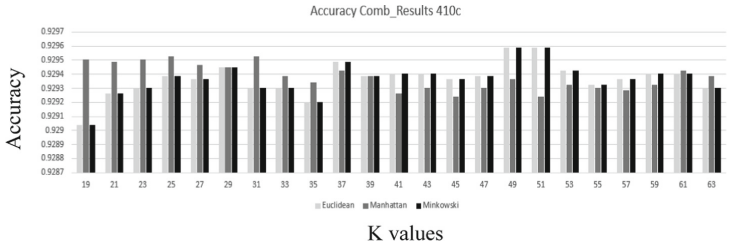

Fig. 22. Accuracy of combined dataset on Qualcomm 410c

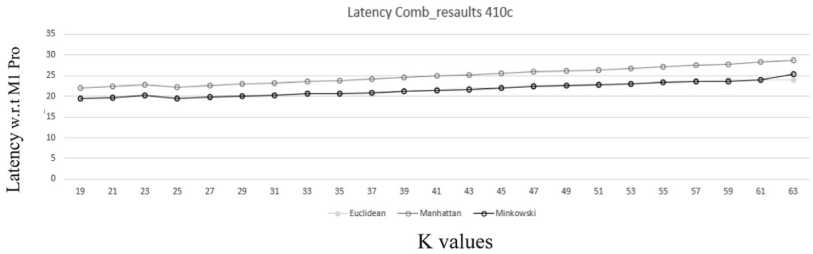

Fig. 23. Latency of combined dataset on Qualcomm 410c

BITS Data Set

In the BITS data set, we have 47,656 data points covering 20 activities. The peak accuracy of the BITS data set is obtained at 27 for all 3. The difference here is in the case of Manhattan, we get a better performance of 85.4701% compared to Minkowski (82.906%) and Euclidian (82.906%). The latencies obtained in the case of the BITS data set are slightly lesser, with the latency being 0.002 s at k = 27 for Manhattan and Minkowski, whereas it is at 0.003s for Euclidean. On Apple M1 processor. In the case of Euclidian and Minkowski, the peak latency is 0.044 s at peak performance for Euclidian and Minkowski and 0.048 for Manhattan. It can also be seen that the latency values do not increase with an increase in k. One of the reasons we could get from the data set was that the age of the subjects was almost the same, varying between 20–22 years, with none having any history of illness or falls. Hence the accuracies are lower, and latencies are lesser. As described in Sect. 5, the BITS data set, other than the accelerometer data, also has gyroscope data, heart rate, and a few other parameters. When KNN was used with all these sensors, we obtained a peak accuracy of approx. 91%, which is comparable with other data sets. The accuracy of results also depends upon the accuracy of the accelerometer used, details of which are provided in Sect. 5. Also, as all the subjects were in the same age range, therefore there is very little difference in acceleration values in terms of fall and non-fall values.

Figure 24 and 25 give the accuracy and latency of BITS dataset on Apple M1 pro
Figures 26 and 27 give the accuracy and latency of BITS dataset on Qualcomm 410c.

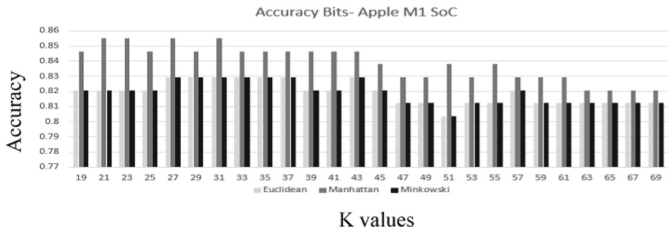

Fig. 24. Accuracy of BITS dataset on Apple M1pro

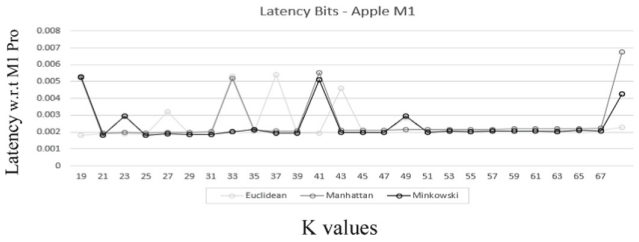

Fig. 25. Latency on BITS dataset on Apple M1 pro

Looking at the varying data sets, we can draw the following conclusions: (i) The accuracy increases with the size of training data (ii) The latencies are very high as the amount of data collected is more (iii) Statistical analysis must be performed before running the ML algorithm on the dataset. (iv) It is impossible to run ML algorithms on the SoC.

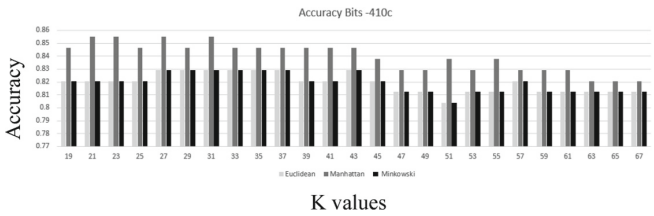

Fig. 26. Accuracy of BITS dataset on Qualcomm 410c

KNN, also considered an algorithm that does not require much pre-training, is one of the simplest algorithms to give accuracies above 90%, even with 10,000 data points requiring latencies more significant than 8 s. Hence there is a need to run compressed ML algorithms on a compressed data set. (v) When we try running the algorithm on raw data set without statistical analysis, 410c repeatedly kills the process as it cannot handle large data.

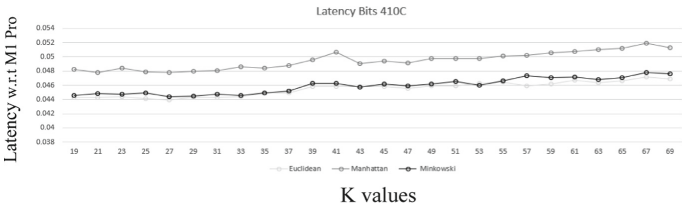

Fig. 27. Latency on BITS dataset on Qualcomm 410c

7 Summary and Conclusions

Deep learning is currently being used for fall detection. DL techniques applied on the Smartwatch, Smartfall and Notch data set gave accuracies in the range of 98.2%, 99.6% and 99%, respectively, while using CNN [26]. Some papers [27] claim RNN can be implemented on STM 32, which is built using ARM cortex M4 and the deep learning algorithms take approximately 1 s for execution. However, the pre-trained and the data set size used was only 22,321 data points. Some data were alert or pre-fall data, and the rest were fall data. The model was pre-trained to detect between pre-falls and falls, giving an accuracy of about 75%. The paper concentrates on reducing energy and power consumption rather than improving accuracy. When we ran the ML algorithm on STM 32 with the BITS data set, which has around 3000 activities and 47,656 data points, we got an accuracy of 77%. This shows that ML or DL algorithms do not converge well on microcontrollers. That is why we have moved on to SoCs.

Even in the case of SoCs though the accuracy is better, it is not possible to run uncleaned test data on the model, as we had seen with our raw test data, where the process was repeatedly killed before we applied statistical methods to derive the mean, median, max and min values after which the IMU data gave us an accuracy of around 85% while using KNN. When heart rate and gyroscope data were added to the dataset, KNN gave us an approximate accuracy of 91%. This was all achieved using Python Scikit tools. Our results show that the accuracy increases as the size of the data set increases, even with an algorithm such as KNN. If we use Model D as suggested in Sect. 2, where the ML algorithms for Fall detection are going to run on the SoC itself, it also becomes necessary to clean data, and perform sensor fusion on the SoC itself. This will take high latency on the SoC and also consume an equal amount of power. Hence, data and model compression techniques are needed if we use an extensive data set while using ensemble ML or RNN techniques. We prefer RNN as a DL technique because RNN works well with time-series data. In this paper, we have demonstrated that using SoCs to run ML algorithms is not possible. Even when running KNN, we had observed latencies in terms of seconds; hence, we need to use compressed models, and hence we have firmly established in this paper, that there is a need to use compressed models for wrist-based model D smart bands/watches. In this paper, we chose to use accelerometer data as most of the available smartwatches and smartphones are already equipped with 3-axes accelerometers. In our future work, we propose to augment this data set with biological parameters such as heart rate, SpO_2 and Galvanic skin response sensors employing RNN-based model compression algorithms.

References

1. World Population Ageing: 1950–2050. https://www.agc.org/ruralaging/world/ageingo.html
2. Tinetti, M.E.: Clinical practice: preventing falls in elderly persons. N. Engl. J. Med. **348**(1), 42–49 (2003)
3. Hausdorff, J.M., Rios, D.A., Edelberg, H.K.: Gait variability and fall risk in community-living older adults: a 1-year prospective study. Arch. Phys. Med. Rehabil. **82**(8), 1050–1056 (2001)
4. Thakur, N., Han, C.Y.: A study of fall detection in assisted living: identifying and improving the optimal machine learning method. J. Sens. Actuator Netw. **10**, 39 (2021). https://doi.org/10.3390/jsan10030039
5. Apple watch user guide. https://documents.4rgos.it
6. Kanega, U.: watch user guide. https://unaliwear.com
7. Fall call lite application documentation. https://www.fallcall.com/apps/FallCall-Lite
8. Naranjo-Hernández, D., Talaminos-Barroso, A., Reina-Tosina, J., et al.: Smart vest for respiratory rate monitoring of COPD patients based on non-contact capacitive sensing. Sens. (Basel). **18**(7), 2144 (2018). Published 3 Jul 2018. https://doi.org/10.3390/s18072144
9. Calvillo-Arbizu, J., Naranjo-Hernández, D., Barbarov-Rostán, G., Talaminos-Barroso, A., Roa-Romero, L.M., Reina-Tosina, J.: A sensor-based mhealth platform for remote monitoring and intervention of frailty patients at home. Int. J. Environ. Res. Public Health. **18**(21), 11730 (2021). Published 8 Nov 2021. https://doi.org/10.3390/ijerph182111730
10. Ramachandran, A., Karuppiah, A.: A survey on recent advances in wearable fall detection systems. BioMed. Res. Int. 2167160, 17 (2020). https://doi.org/10.1155/2020/2167160
11. Alpaydin, E.: Voting over multiple condensed nearest neighbors. Artif. Intell. Rev. 115–132 (1997)
12. Vembandasamy, K., Sasipriya, R., Deepa, E.: Heart diseases detection using Naive Bayes algorithm. IJISET – Int. J. Innov. Sci. Eng. Tech. **2**(9) (2015). ISSN 2348 – 7968
13. Hosmer, D.W., Lemeshow, S.L.: Applied Logistic Regression, 2nd edn. Wiley-Interscience, Hoboken, NJ (2000)
14. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees (1st ed.). Routledge (1984). https://doi.org/10.1201/9781315139470
15. Liu, S.H., Cheng, W.C.: Fall detection with the support vector machine during scripted and continuous unscripted activities. Sens. (Basel) **12**(9), 12301–12316 (2012). https://doi.org/10.3390/s120912301. Epub 2012 Sep 7. PMID: 23112713; PMCID: PMC3478840
16. Nho, Y.H., Lim, J.G., Kwon, D.S.: Cluster-analysis-based user-adaptive fall detection using fusion of heart rate sensor and accelerometer in a wearable device. IEEE Access, 1 (2020). https://doi.org/10.1109/ACCESS.2020.2969453
17. Gøeg, K.R., Cornet, R., Andersen, S.K.: Clustering clinical models from local electronic health records based on semantic similarity. J. Biomed. Inform. **54**, 294–304 (2015). https://doi.org/10.1016/j.jbi.2014.12.015ISSN 1532–0464
18. Yoon, B.J.: Hidden Markov models and their applications in biological sequence analysis. Curr. Genomics **10**(6), 402–415 (2009). https://doi.org/10.2174/138920209789177575. PMID:20190955;PMCID:PMC2766791
19. Wang, G., Liu, Z., Li, Q.: Fall detection with neural networks. In: 2019 IEEE International Flexible Electronics Technology Conference (IFETC), pp. 1–7 (2019). https://doi.org/10.1109/IFETC46817.2019.9073718
20. Xing, W., Bei, Y.: Medical health big data classification based on KNN classification algorithm. IEEE Access **8**, 28808–28819 (2020). https://doi.org/10.1109/ACCESS.2019.2955754
21. Chomboon, K., Chujai, P., Teerarassammee, P., Kerdprasop, K., Kerdprasop, N.: An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm, pp. 280–285 (2015). https://doi.org/10.12792/iciae2015.051

22. Kraft, D., Srinivasan, K., Bieber, G.: Deep learning based fall detection algorithms for embedded systems, smartwatches, and IoT devices using accelerometers. Technologies **8**, 72 (2020). https://doi.org/10.3390/technologies8040072
23. SmartWatch dataset. https://userweb.cs.txstate.edu/~hn12/data/SmartFallDataSet/SmartWatch/
24. Notch dataset. https://userweb.cs.txstate.edu/~hn12/data/SmartFallDataSet/notch/Notch_Dataset_Wrist/
25. SmartFall dataset. https://userweb.cs.txstate.edu/~hn12/data/SmartFallDataSet/SmartFall/
26. Santos, G.L., Endo, P.T., Monteiro, K.H.D.C., Rocha, E.D.S., Silva, I., Lynn, T.: Accelerometer-based human fall detection using convolutional neural networks. Sensors **19**, 1644 (2019). https://doi.org/10.3390/s19071644
27. Farsi, M.: Application of ensemble RNN deep neural network to the fall detection through IoT environment. Alexandria Eng. J. **60**(1), 199–211 (2021). SSN 1110–0168, https://doi.org/10.1016/j.aej.2020.06.056