ACTA IMEKO
ISSN: 2221-870X
September 2023, Volume 12, Number 3, 1 - 11

Use of the *k*th-nearest neighbour and its analysis for fall detection on Systems on a Chip for multiple datasets

Purab Nandi[1], K. R. Anupama[1], Himanish Agarwal[1], Arav Jain[1], Siddharth Paliwal[1]

[1] *Birla Institute of Technology and Science, Pilani, K K Birla Goa Campus, Goa -403726, India*

ABSTRACT
Fall of an elderly person often leads to serious injuries and death. Many falls occur in the home environment, and hence a reliable fall detection system that can raise alarms with minimum latency is a necessity. Wrist-worn accelerometer-based fall detection systems and multiple datasets are available, but no attempt has been made to analyze the accuracy and precision. Wherever the comparison does exist, it has been run on a cloud. No analysis of the models, convergence, and dataset analysis on Systems on a Chip (SoCs) has ever been attempted. In this paper, we attempt to present why Machine Learning (ML) algorithms in their current state cannot be run on existing SoCs.
We have used Snapdragon 410c SoC to do our analytics. In this paper, we have used the *k*th-nearest neighbour to prove that ML cannot be directly run on SoCs. We have looked at the effect of distance metrics and neighbors as well as the effect of feature extraction on the accuracies and the latencies. In this paper, we establish the need for model compression and data pruning for fall detection using ML/Deep Learning algorithms on SoCs. We have done this by analyzing various datasets on varying architectural parameters.

Section: RESEARCH PAPER

Keywords: SoCs; Wearable; IoT; ML; k-NN

Citation: Purab Nandi, K. R. Anupama, Himanish Agarwal, Arav Jain, Siddharth Paliwal, Use of the k-nearest neighbour and its analysis for fall detection on Systems on a Chip for multiple datasets, Acta IMEKO, vol. 12, no. 3, article 51, September 2023, identifier: IMEKO-ACTA-12 (2023)-03-51

Section Editor: Francesco Lamonaca, University of Calabria, Italy

Received February 28, 2023; **In final form** July 17, 2023; **Published** September 2023

Corresponding author: Purab Nandi, e-mail: p20200056@goa.bits-pilani.ac.in

1. INTRODUCTION

Medical and healthcare advancements have increased the average human lifespan to over 80. Geriatric healthcare hence has become vital, and regular monitoring of parameters is required. Currently, visiting, or in-house nursing staff monitor various parameters; such an arrangement is expensive for a significant part of Indian society. The past decade has witnessed substantial advances in IoT (Internet of Things)-based wearable-based health devices and their integration with machine learning and deep learning; remote diagnosis, prognosis, and treatment can be performed using IoT-based medical-grade devices. Hence advances in IoT, embedded systems and ML (Machine Learning) are the catalysts in developing 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 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.

Sensors monitor multiple health conditions and relay data to an intelligent ML-based system that can detect and predict the condition. In this paper, we concentrate on geriatric fall detections. The causes of the falls can be internal or 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, and other ailments. The duration of the fall is also essential. 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 geriatric population. Although numerous studies [3] related to fall detection have been out recently, several challenges still exist. These include

1. The lack of a comprehensive analysis of ML techniques deployed to detect falls,
2. A high number of false positives,
3. The systems that detect and correct such false positives have low accuracy,
4. The inability of the system to detect the duration of the fall.

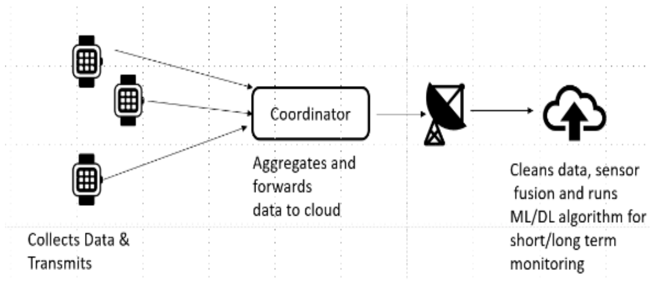
With technological advances in SoCs (System on Chip) and IoT systems, wearable devices have emerged as a leading area of research in geriatric health care; the amount of data collected at homes/hospitals for the elderly is large and complex and making accurate decisions based on multiple parameters is the primary requirement for such safety-critical systems. SoCs are resource-constrained devices 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 selects one of the simplest ML algorithms. This algorithm is validated using varying datasets of different sizes. We have done this to prove that running even a simple ML algorithm such as k-NN (Kth Nearest Neighbour) is impossible when the data set size is considerable. Not only is the latency high but as the size of the dataset 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.

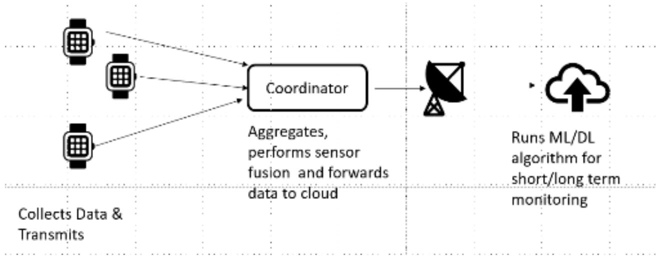
The organization of the rest of the paper is as follows: Section 2 talks about various IoT architectures employed in fall detection, Section 3 gives a brief overview of ML algorithms used for fall detection, Section 4 provides the operational details of k-NN, Section 5 gives the details of the dataset used for analysis, we present our results in Section 6, and Section 7 summarises and concludes this paper.

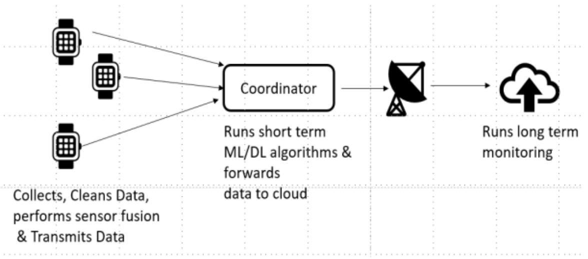
2. ARCHITECTURAL MODELS FOR IOT BASED FALL DETECTION SYSTEMS WITH WEARABLE END DEVICE

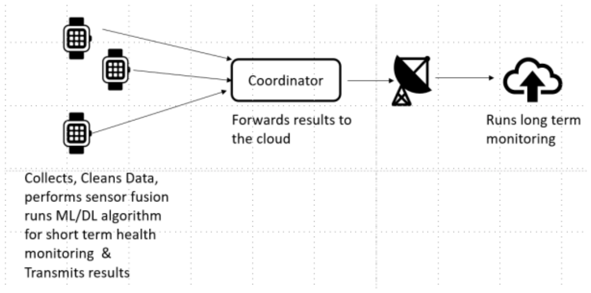
With the growth of SoCs and their integration with IoT systems, wearable healthcare devices are now a focused research area. This section presents four possible IoT architectural models for healthcare applications. The variation in the models is in data gathering, processing and the conversion of data to knowledge.

1. *Model A* – In this architectural model, the data is collected at regular sampling intervals from the sensors of the wearable devices, which forwards the data to the coordinator. The coordinator then collects the data from multiple wearables and transfers the data to the cloud. The data analytics using ML/DL (Deep Learning) algorithms is performed on the cloud. The end devices have constrained processing and memory capabilities in such an architecture. The coordinator only acts as a data forwarder. Figure 1 gives the schematic diagram of Model A. The research focus of this architecture is usually on developing networking protocols that can transmit the data to the cloud with minimum loss and latency with low control overheads.
2. *Model B* – In this architectural model, the wearable end device collects the raw data and transmits it to the coordinator. The coordinator not only forwards the data to the cloud but performs sensor fusion prior to forwarding the data. The cloud then uses the fused data to extract the required features and converts the data to health decisions using ML/DL algorithms. Figure 2 gives the schematic diagram of Model B. In this model, the coordinator architecture is as important as the network protocols; preferably, SoCs are used as coordinators.
3. *Model C* – In the case of architectural model C, powerful end devices collect data from multiple sensors, run sensor fusion algorithms then forward the data to the coordinator. The coordinator then runs ML/DL algorithms for data analytics on the fused data. The data analytics performed on the coordinator are for short-term health monitoring. In contrast, the analytics run on the cloud to which the co-ordinator forwards the data are for long-term health monitoring. The end devices are powerful enough to run sensor fusion algorithms. Figure 3 gives the schematic diagram of


Coordinator
Aggregates and forwards data to cloud
Cleans data, sensor fusion and runs ML/DL algorithm for short/long term monitoring
Collects Data & Transmits
Figure 1. IoT architectural Model A.


Coordinator
Aggregates, performs sensor fusion and forwards data to cloud
Runs ML/DL algorithm for short/long term monitoring
Collects Data & Transmits
Figure 2. IoT architectural Model B.


Coordinator
Runs short term ML/DL algorithms & forwards data to cloud
Runs long term monitoring
Collects, Cleans Data, performs sensor fusion & Transmits Data
Figure 3. IoT architectural Model C.


Coordinator
Forwards results to the cloud
Runs long term monitoring
Collects, Cleans Data, performs sensor fusion runs ML/DL algorithm for short term health monitoring & Transmits results
Figure 4. IoT architectural Model D.

Model C. The server-class coordinators still run the ML/DL algorithms. Hence, latencies will still be involved in making short-term health decisions due to the latency in transmitting data between the end device and the coordinator.

4. *Model D* – Figure 4 gives the schematic diagram of Model D. In the case of architectural model D, the wearable device is built using a powerful SoC as it collects data from multiple sensors, fuses it, and runs ML and DL algorithms to detect/predict falls. The wearable device, in this case, 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 the amount of memory available, energy consumed and form factor. As the device is wearable, the form factor must be significantly 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 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 simplified model compared to the original with the same level of accuracy as the original algorithms. The advantage of running a reduced model is 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 and the code are also resident in the memory. The processing latency is also expected to 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. Bad connectivity reduces the lie-in period after the fall, reducing complex health situations that might arise due to long lie-in periods.

2.1. 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 the models 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 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 is unsuitable 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 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 briefly overview such fall detection systems.

2.2. Commercially available systems and their applications

Apple Watch SE or series 4 [5] and above can detect hard falls. For people above 55 years of age, these services are enabled automatically. 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 impact acceleration and the resultant wrist trajectory for fall detection. 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 seconds; it then begins a 30 second 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 per cent; it has a false-negative rate of 95.3 per cent, 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 for fall detection. It also makes use of accelerometer data to detect falls. The Unali watch is unique because the user can charge the battery while still wearing the watch. This feature is helpful since falls may occur when the user removes his watch to charge.
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 when 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).

There is 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 rare as they require that the person be still conscious and can raise an alert.

2.3. 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 smartWatch worn by the user acts as the gateway to the platform, gathering data from sensors and receiving events and reminders introduced by caregivers. Since the SmartWatch is responsible for communication with the frail servers, the end devices and the sensing platforms only need to send the sensor data.

Hence, this falls under model C of the IoT architecture described in section 2. Table 1 gives a summary of a few commercially available wearable devices which are used for Fall

Table 1. The summary of commercially available wearables.

Product	Automatic fall 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

Detection. For fall detection, mainly accelerometer-based devices are used. Also, after the accelerometer detects the fall, the SmartWatch expects the wearer to confirm that he/she has fallen. If the user confirms the fall, SmartWatch returns the event to the frail servers and triggers a preconfigured procedure.

The sensor module used is a tri-axial accelerometer, and the processing module is a PIC 18F2431 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 machine learning algorithms to detect falls. Further research 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. The research that we are doing 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 analyse, classify and convert data into knowledge. There are three types of ML algorithms.

Supervised Learning: In supervised learning, the input data is classified a priori using a training data set; any new data is automatically classified into one of the input types, some of the algorithms include k-NN [11], Naïve Bayes [12], Decision trees [13], Linear Regression [14], Support Vector Machine (SVM) [15].

Unsupervised: In unsupervised learning, the ML algorithm recognizes a pattern on its own from a given data set; some of the standard algorithms include K-Means clustering [16], Classification rules [17], Hidden Markov model [18], Neural Networks [19].

Reinforced: This algorithm allows the system to adapt its behaviour based on feedback from the environment.

In the case of fall detection, binary classification is used to classify an activity into a fall or Activities of Daily Living (ADL). The diagram given below shows how the ML model is built. The data is collected and cleaned for incorrect sensor readings, and the statistical features are extracted before training.

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

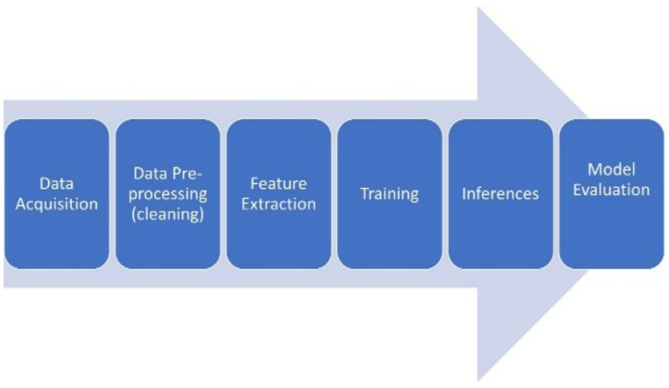


Figure 5. Schematic of ML Model.

For fall detection, several ML algorithms are currently being used. Commonly used algorithms are Logical Regression, Naïve Bayes, SVM, k Nearest Neighbour and Random Forest. Among these algorithms, this paper concentrates on the k-NN algorithm. We had earlier run multiple ML algorithms for fall detection. The AUC (Area Under Curve)/ROC (Receiver Operating Characteristic) curve for them is shown in the Figure 6.

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. k-NN has a good AUC score of 0.971, performing as well as SVM. We have used k-NN to analyse the latencies involved in an ML algorithm as the accuracies of k-NN are good, and k-NN does not require any pre-training until a query is raised. Hence, k-NN is the best model to understand the effect of implementing ML on SoCs, as no heavy pre-training is required, unlike the other algorithms. While Naïve-Bayes is easier to implement, its AUC score is only 0.825, which is very low compared to k-NN.

4. K-NEAREST NEIGHBOR CLASSIFICATION ALGORITHM

The problem presented in this paper consists of an ML algorithm that classifies features extracted into fall and non-fall. We wanted to use a classification technique with good accuracy and minimum complexity as we plan to implement the ML algorithm on the End device. In this paper, we have used k-NN [20]. The k-NN algorithm does not require a choice of a specific classification model or feature selection; it only requires a suitable metric to evaluate the distance between features. The k-NN algorithm depends upon a key parameter[21], a priori fixed k; the value of *k* is a critical parameter for the algorithm as it is the

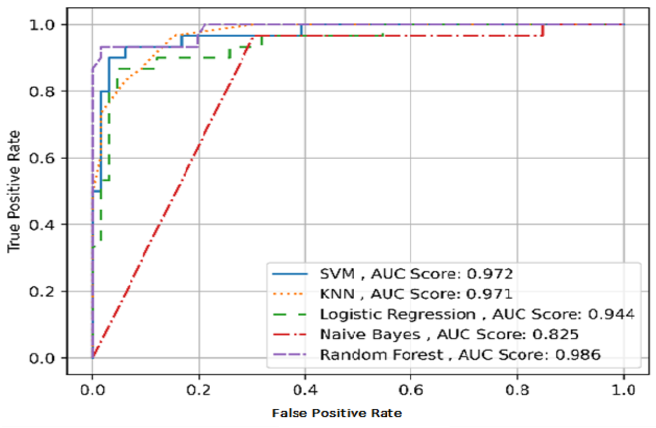


Figure 6. AUC/ROC for Fall detection.

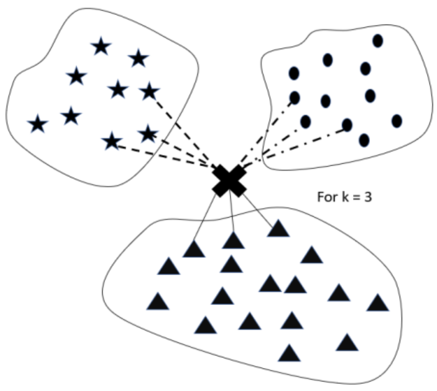


Figure 7. Classification Schematic of k-NN algorithm (k = 3).

primary source of variation in accuracy. A further parameter of the algorithm is the selection of the distance metric between the points such as Euclidean, Manhattan, Minkowski, and Hamming. The k-NN schematic diagram used for classification for a k value of 3 is shown in Figure 7.

Three distance metrics used in this paper are

1. Euclidean
2. Manhattan
3. Minkowski [21].

The Euclidean distance is calculated using the formula given below

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

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

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

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

$$d = \{|x_2 - x_1|^p + |y_2 - y_1|^p)^{1/p}\,. \tag{3}$$

The Hamming distance is a special case of Minkowski distance, the Hamming distance between two strings of equal length results as the number of positions at which the corresponding symbols (0/1) are different.

The Hamming distance equation is given by

$$d = \sum |A_i - B_i|\,. \tag{4}$$

The effect of the parameter p on the behaviour of the k-NN algorithm while using Minkowski distance was also analysed. According to previous research [22], distance plays a pivotal role in studying disease patterns. We tried varying values of p, starting from 0.5 to 12, and we found that the maximum accuracy was in the range of 1.2 to 1.5. So, we ran the k-NN algorithm using the Minkowski distance estimator to find the ideal combination of p and k values. The detailed description is described in the results section.

5. DATASET AND DATA PREPERATION

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 [23] 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—the first two fall under the wearable sensor category. The latter are non-wearable. Biological sensors might be heart rate sensors, GSR sensors, SpO2 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 system includes radar and acoustic sensors, which are used for detecting falls.

Wearable sensors are preferred because they can follow the elderly through their daily routine. Multiple wearable sensors, primarily IMU-based sensors, are available, 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. This paper focuses on wrist-worn sensors as we strongly believe a wrist sensor is more convenient for an elderly user. The sensor data can constantly be augmented appropriately using mathematical models.

Furthermore, multiple fitness bands in the market 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 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. SmartWatch, SmartFall and Notch datasets are available publicly, so we used them. We also collected data using ten volunteers wearing a TicWatch (BITS dataset).

Data collection methodology for BITS dataset

We gathered our data using ten volunteers wearing a TicWatch, including 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 TicWatch, data was 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 ft 1 in to 5 ft 8 in, and their weight was 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, backwards, 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 software's user interface had buttons for every type of activity performed, so the corresponding button was selected before performing a particular activity. The button enabled automatic labelling of the dataset as it got stored.

IMU and heart rate samples 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 × 2.2 m × 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 any harm during the experimentation.

The volunteer's rebound and residual movements after a fall activity were also considered to ensure that the data set contains post-fall 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 that 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 the case of the BITS data set though we collected data using 3-axis accelerometer, magnetometer, and gyroscope as well as an optical heart rate sensor, we used only the data from the 3-axis accelerometer to study the accuracies over the other data sets. Since we filtered out the other sensor and only used a 3-axis accelerometer, the data points were reduced from 110,000 to 47,656.

Data preparation for SmartFall, SmartWatch and Notch

SmartWatch dataset [24] – SmartWatch had 7 subjects aged 21 to 55 who performed 971 different activities. Each datapoint was collected at a sample rate of 31 Hz. To match it with our data set, we downscaled it to 20 kHz using the Python SciPy 1.2.3 package; overall, there were 34,019 data points.

Notch dataset [25] – The Notch data set has only 7 subjects aged 20-35. Overall, Notch had 10,645 data points; the dataset was also collected at 31 Hz, which was eventually downscaled to 20 Hz to match our data set.

SmartFall dataset [26] – 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 the case of SmartWatch and Notch, we downsample 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. The fusion gave us an overall 2496 activities extended over 28 subjects aged 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.

Feature Extraction – Rather than using the raw data, we extracted certain features, primarily statistical parameters such as maximum, minimum, mean, standard deviation, kurtosis, skew, and variance.

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 incorrect results. We did not upscale or downscale the data because no mathematical relationship could be derived from the data sets due to using 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 trade-off.

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

Table 2. Summary of datasets used in this paper.

Data set	Activities	Data points (Raw)	Data Points (feature extracted)	Test Subjects	Age
BITS	3000	1,249,050	47,656	10	20-22
Notch	698	10,645	218	7	20-35
SmartWatch	771	34,019	367	7	21-55
Smart Fall	1027	92,780	182	14	21-60
Fused	2496	137,444	765	28	20-60

6. RESULTS AND DISCUSSION

To understand the working of the ML algorithm on our SoC, we used the k-NN classifier, one of the simplest classifiers, yet its performance competes with the most complex classifiers. The core of this classifier depends mainly on measuring the distance or similarity between the test and the train samples. The main aim of k-NN is to find the nearest neighbour of the query. Hence distance is the primary parameter in k-NN, and the method used to calculate the distances also has a huge impact on the accuracy of the predictions. We have used three of the distance methods, a. Manhattan, b. Euclidean and c. Minkowski. As described in section 5, we worked with multiple datasets; SmartWatch, SmartFall, Notch and BITS dataset. This was to understand the effects of data collection and the impact of the dataset size and features on the accuracy of the ML model. Furthermore, to understand the limitations of the dataset sizes that the SoC could handle, we used raw and cleaned data from a SmartWatch, SmartFall, Notch and BITS dataset, all downsampled to 20 Hz. Feature extraction essentially involved the calculation of statistical parameters as described in Section 5. Statistical analysis of the data is usually done on the cloud. Since we plan to run the ML Algorithm on the SoC itself, it is not logical to do the feature extraction on the cloud and transmit them back to the SoC. Hence the feature extraction happens on the SoC itself.

We ran the algorithms on varying computing architectures.

1. Apple M1 pro running at a frequency of 3.2 GHz
2. Intel i9 12th gen operating at 5.2 GHz
3. Qualcomm 410c working at 2.4 GHz.

We tried random data splits of 80-20, 70-30 and 60-40 while testing on M1 pro and Intel i9 architectures. We also ran the ML algorithms on both the raw dataset and the feature extracted dataset. In this section, for better understandability, we only present the results of the 70-30 splits of the feature-extracted data in graphical form. The rest of the results is presented as part of the discussion.

All graphs plotted are of k vs accuracy with a p-value mentioned on the graph for different values of k.

Figure 8 gives the accuracy of the feature extracted notch dataset. It can be observed from Figure 8 that the Manhattan distance gives the best accuracy at 96.12 %, and the lowest accuracy is that of Euclidean at 95.61 %. We ran the Minkowski distance-based k-NN for varying values of p, and we observed

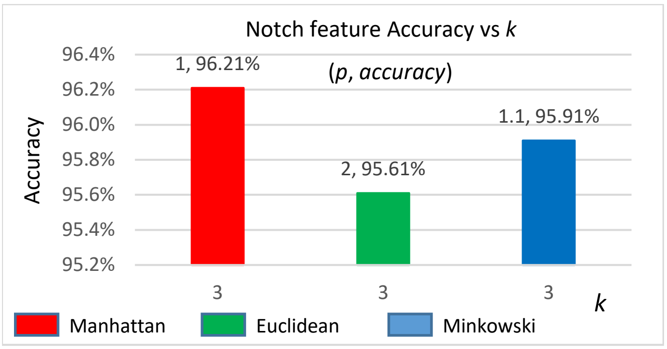


Figure 8. Accuracies of Notch Features vs *k*.

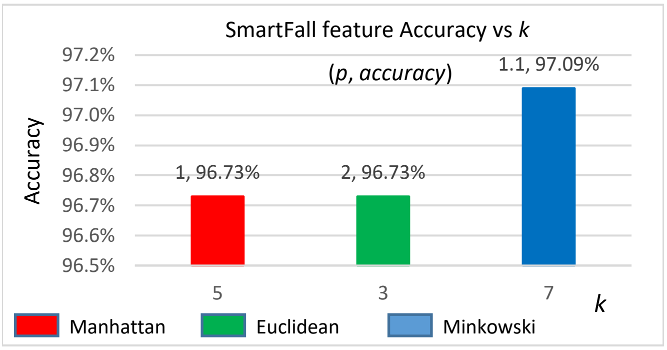


Figure 9. Accuracies of SmartFall Features vs *k*.

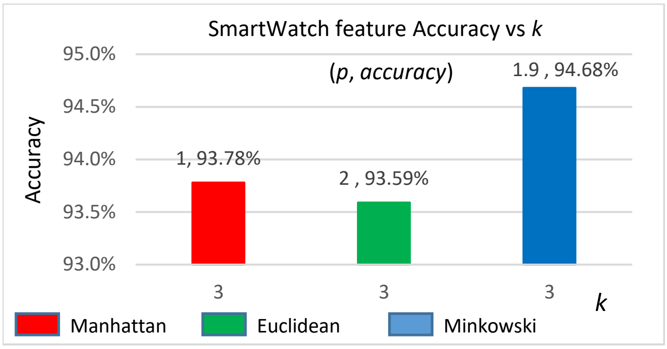


Figure 10. Accuracies of SmartWatch Features vs *k*.

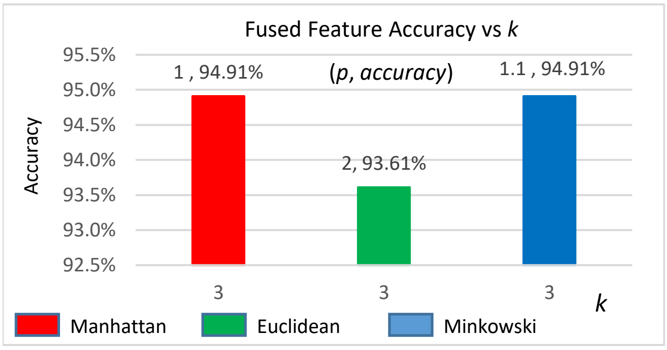


Figure 11. Accuracies of Fused Features vs *k*.

that we got the best accuracy at a *p*-value of 1.1. The accuracy obtained for Minkowski was 95.91 %. This is close to the accuracy of Manhattan since the *p*-value of Manhattan is 1 and Minkowski is 1.1. The variation in accuracy between them is just 0.3 %.

When raw data was used, the accuracies for all the distance metrics was 91.82 % but for different values of k. In the case of Manhattan, the *k*-value was found to be 19, Euclidean 17 and Minkowski 15, with a *p*-value of 1.5. This shows that feature extraction gives good accuracies even with small and noisy datasets. The best accuracies were obtained for Manhattan in the case of Notch. Manhattan usually gives better results when the dimensionality of the data is large, especially the impact can be observed with small datasets. In the case of Notch, the number of data points is 218 while the number of features extracted is 28, indicating high dimensionality [27]; shows that when the dimensionality is high for small and noisy data, Manhattan distance is preferred. For dimensionality of 20 or higher, Manhattan distance provides significantly higher contrast relative to Euclidean distance. This can also be seen in the results because Minkowski gives an accuracy of 95.91 % at a *p*-value of 1.1, making it closer to the Manhattan distance as compared to the Euclidean distance. With the increase in data size, as we move on to SmartWatch and SmartFall as can be observed from Figure 9 and Figure 10 respectively, the performance of Minkowski is slightly better; with an accuracy of 97.09 % for a *p*-value of 1.1 for SmartFall, and 94.68 % for a *p*-value of 1.6 for SmartWatch. For the Fused dataset as can be observed in Figure 11, the accuracy for Minkowski and Manhattan are the same at 94.91 %. Again, the *p*-value of Minkowski is 1.1, making it closer to Manhattan than Euclidean.

In case of SmartWatch, Figure 10, there is a drop in accuracy even though the number of features extracted are more than that of Notch because the user age demographics for Notch is in the range of 21-35, while SmartWatch has a wider range of 21-55. Hence, the drop in accuracy as there might not be enough volunteers across the age range. When we gathered data for the second time, in the age range of 21-50, we used 41 users as we were looking for equal representation across the ages. Again, information is not available regarding the height and the weight of the users but assuming a wide range of 21-55, it is obvious that there would have huge variations in these parameters. Variation in height and weight has huge impact on accelerometer data. Hence, there have been very less training instances considering the wide user demographics, thus causing a slight fall in the accuracy of results.

In case of SmartFall dataset, the user age demographic is between 21-60 but the number of users was 14 and they performed more than 1000 activities. Hence, there was no dearth in the training data. In case of the Fused dataset, Figure 11, the accuracies are at 94.91 % because we combined three completely different datasets together with varying ADL and fall activities. The website for SmartWatch, SmartFall , and Notch does not provide the details for these activities, so it is possible that completely different activities were done for the three datasets.

k-NN tries to find similarities and differences to classify an activity as Fall or ADL but considering that the activities for the three datasets may have been completely different, it would not have been possible to just rely on the distances to get the correct prediction.

In case of the raw data, the accuracies are consistently lower with extremely high *k*-values. For feature extracted data, *k* was consistent at 3. In case of raw data, the *k*-value varied from 15 to 61. The increase in *k*-values is consistent with the increase in data points. There is not much difference between the *k*-values required for the three distances, that is Manhattan, Euclidean and Minkowski. If Manhattan had a *k*-value of 59, then Euclidean had 61, but there was a constant increase in the *k*-value with the number of datapoints. In case of Notch, it was in the range of 31 to 33. In case of SmartFall, it was in the range of 43 to 45. In case

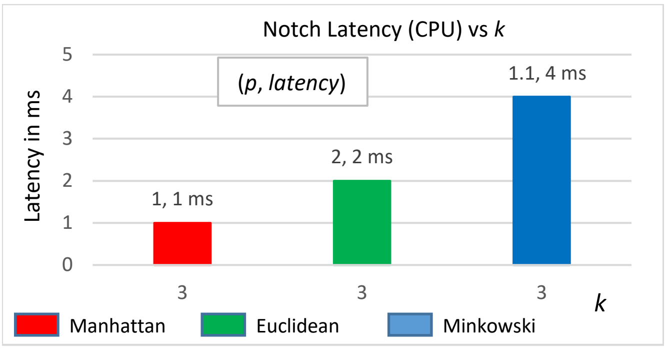


Figure 12. CPU Latency of Notch Features vs *k*.

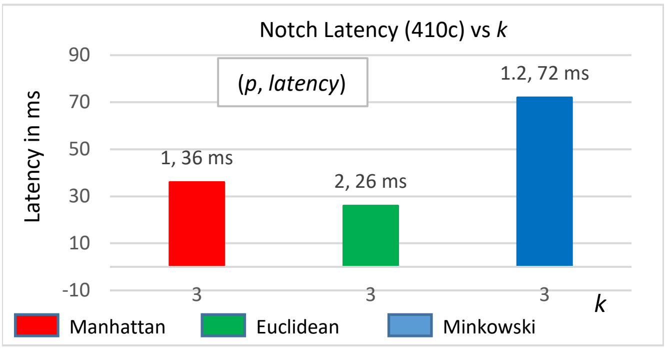


Figure 13. 410c Latency of Notch Features vs *k*.

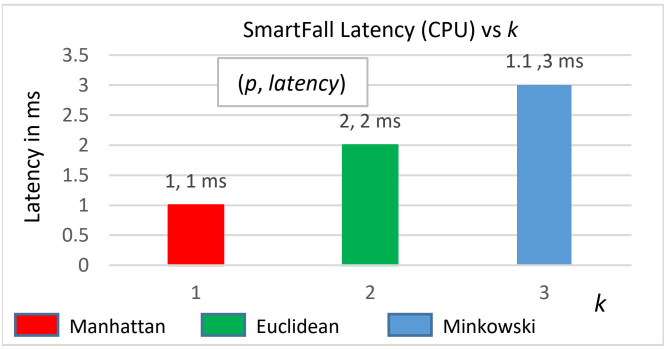


Figure 14. CPU Latency of SmartFall Features vs *k*.

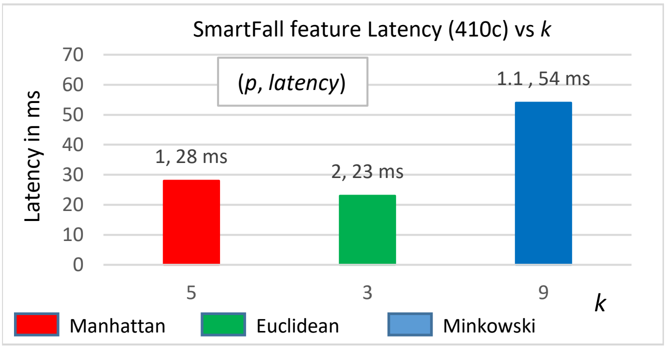


Figure 15. 410c Latency of SmartFall Features vs *k*.

of SmartWatch, it was in the range of 35 to 37. In case of Fused data, it was in the range of 59 to 61.

We had also done the 80-20 data split and 60-40 data split though it has not been plotted here, for which the accuracies were lesser than that of the 70-30 data split by 1.5 % in almost all cases. In case of feature extracted dataset, the k value continues to remain at 3, in case of the raw dataset though there was no significant drop in accuracy, there was variation in the k value. The k value was between 15 to 17 for notch in case of 80-20 data split, 33 - 37 in case of 70-30 data split, and 31 – 33 in case of 60 – 40 data split. In case of SmartWatch for the 80-20 data split the k values were between 17-19 and 35-37 for 70-30 split, and 27-29 for 60-40 data split. Again, there was no significant variation in accuracy. In case of SmartFall for the 80-20 data split, the values were between 27-33 and 43-45 in case of 70- 30 split and 23 - 29 for 60-40 split. In case of Fused it was 59-61 for all the data splits. Again, there was no variation in accuracies.

The increase in k value can be attributed to the fact that there is large amount of data in the raw format, and these are values that are sensed continuously at an interval of 1 s/20 = 0.05 s. Hence, values will be similar, therefore the k values which produce good accuracies would be higher. The downside of having high k values is that the latency increases with k.

We are more concerned with the latencies of the 410c. In case of Notch , the latencies of the CPU, as can be observed from Figure- 12 , they were in the range of 1 – 4 ms. In case of 410c, it varied between 26 ms to 72 ms.

In case of 410c, as seen in Figure 13 the latency was minimum in case of the Euclidean distance, but irrespective of the difference the latency among the distances, it is quite high because our actual sampling rate was 1 ms, later we down sampled it to 50 ms to maintain uniformity between the three datasets.

The Euclidean distance metric corresponds to the L2-norm of a difference between vectors and vector spaces. The cosine similarity is proportional to the dot product of two vectors and inversely proportional to the product of their magnitudes. Hence calculation of Euclidean distance incurs lesser latency. As the size of the dataset increases the latency also increases as can be observed from the graphs. In case of SmartFall it ranges between 1 ms to 3 ms in case of CPU as seen in Figure 14, and 23-54 ms in case of the SoC as seen in Figure 15. In the case of SmartWatch the latencies were observed in the range of 3 ms to 9 ms in case of CPU as seen in Figure 16 and between 35 to 170 ms in case of the SoC as can be observed in Figure 17. In the case of Fused dataset the latencies were observed in the range of 6 ms to 19 ms in case of CPU as can be observed in Figure 18 and between 70 – 340 ms in case of SoC as can be seen in Figure 19. In the Fused dataset we combined the data from Notch, SmartFall and SmartWatch. The latencies were greater than 70 ms, in which time 35 samples would have already been taken from the users hence, as the dataset becomes larger, raw data cannot be used with SoCs, even feature extracted data cannot be used on SoCs. Hence, feature pruning or feature selection would be required.

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 k = 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 Euclidean (82.906 %). The latencies obtained in the case of the BITS data set are slightly lesser, with the latency being 2 ms at k = 27 for Manhattan and Minkowski, whereas it is at 3 ms for Euclidean on the Apple M1 processor. In the case of Euclidean and Minkowski, the peak latency is 44 ms at peak performance for Euclidean and Minkowski and 0.048 ms 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

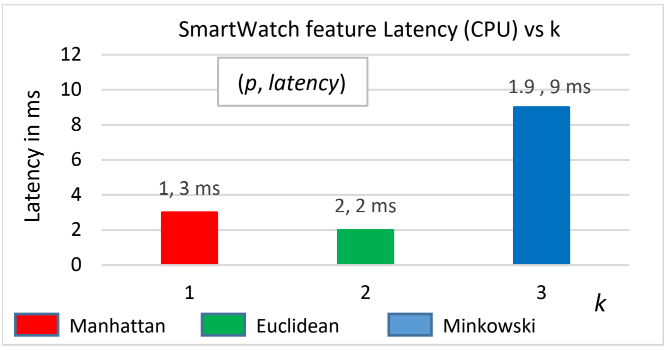


Figure 16. CPU Latency of SmartWatch Features vs *k*.

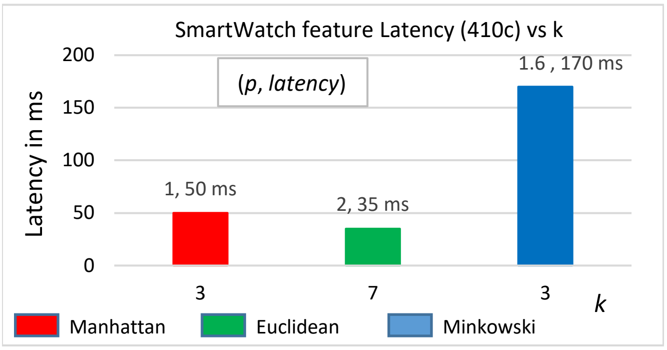


Figure 17. 410c Latency of SmartWatch Features vs *k*.

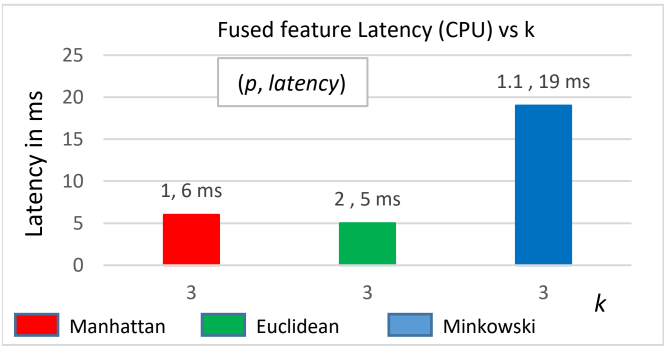


Figure 18. CPU Latency of Fused Features vs *k*.

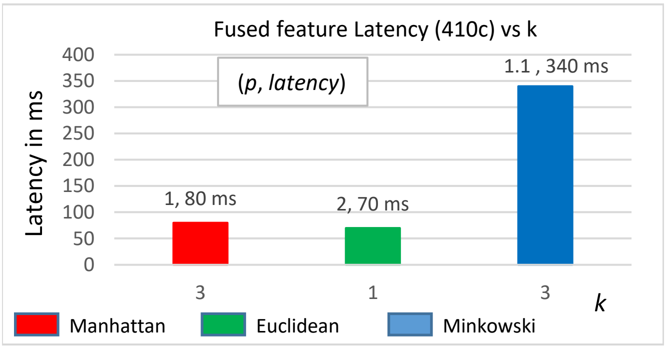


Figure 19. 410c Latency of Fused Features vs *k*.

having any history of illness or falls. Hence the accuracies are lower, and latencies are lesser. As described in Section 5, besides the accelerometer data, the BITS data set also has gyroscope data, heart rate, and a few other parameters. When k-NN 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 accelerometer's accuracy, details of which are provided in Section 5. Also, as all the subjects were in the same age range, there is very little difference in acceleration values regarding fall and non-fall values.

Figure 20 and Figure 21 give the accuracy and latency of the BITS dataset on Apple M1 pro.

Figure 22 and Figure 23 give the accuracy and latency of BITS dataset on Qualcomm 410c.

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. k-NN, 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, compressed ML algorithms are needed on a compressed data set. (v) When we try running the algorithm on raw data sets without statistical analysis, 410c repeatedly kills the process as it cannot handle large data.

7. SUMMARY AND CONCLUSIONS

Currently, Deep Learning techniques are being used for Fall detection. DL techniques used on SmartWatch, SmartFall and Notch datasets gave accuracies in the range of 98.2 %, 99.6 % and 99 %, respectively, when using CNN (Convolution neural networks) [28] on the cloud; some papers [29] implement RNN on STM-32 and DL algorithms only take 1 s for execution. However, the pre-trained and the dataset size used had only 22 thousand points. Some data were alert or pre-fall alert, and the rest was fall. The model was pre-trained to detect pre-falls and falls and gave an accuracy of 75 %. When we tried to reproduce the results by running ML algorithms on STM-32 with the BITS dataset, which had around 3,000 activities and 47,656 data points, we got an accuracy of 77 %. Neither 75 % nor 77 % are good accuracies, especially considering the prediction is for safety-critical Geriatrics applications.

Hence, we have opted to use high-performance advanced architecture SoCs. Though SoCs provide accuracies equivalent to advanced processors such as Intel i9 and Apple M1 pro, we could not run the training and testing for more than ten epochs. In the case of the Fused dataset for ten epochs, the SoC took approximately 4 hr to complete. These were just accelerometer data and one of the simpler algorithms, which was k-NN. We

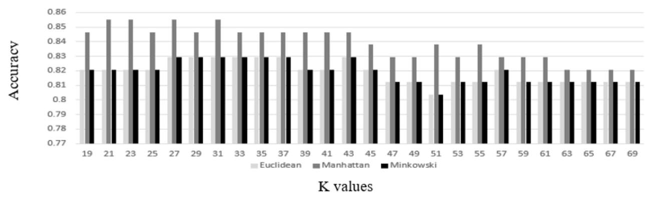


Figure 20. Accuracy vs *k* BITS data M1 pro.

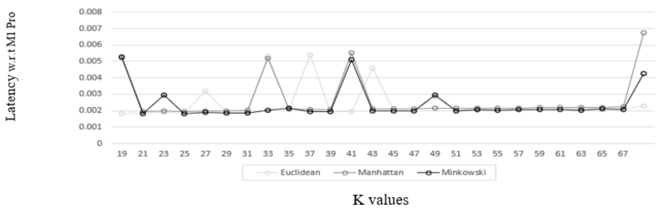


Figure 21. Latency vs *k* BITS data M1pro.

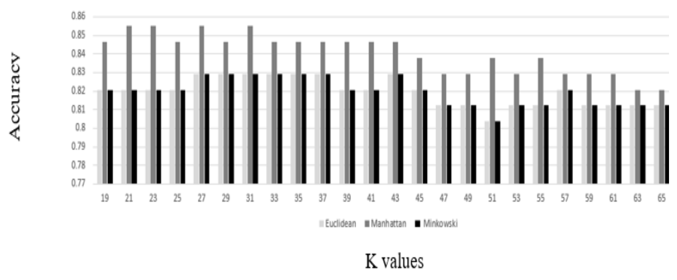


Figure 22. Accuracy vs *k* BITS data 410c.

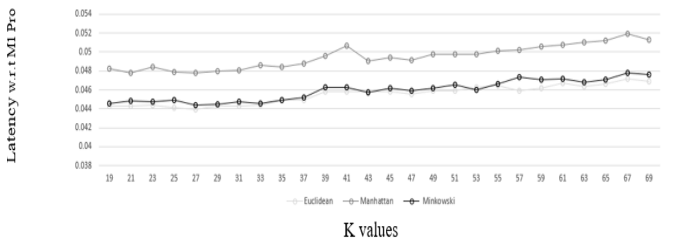


Figure 23. Latency vs *k* BITS data 410c.

obtained an accuracy of around 91 %. Our results also indicate that accuracy increases with the data size; from the results, we can also conclude that when features are extracted, the performance improves. We need to run both the feature extraction and the ML algorithm on the SoC. The results we have shown only include the data loading and testing time but not the time taken for feature extraction. The time for feature extraction may be high depending on the number of features that must be extracted, and the sensors used. When multiple sensors are used, the amount of data increases. In our research, we intend to use data from the 3-axis accelerometer, Gyroscope, and magnetometer, as well as data from heart rate sensors, SpO2 and galvanic skin sensor, which would mean that the number of dimensions would be 94. Hence, the time taken for the algorithm to perform feature extraction and predict the outcome will be extremely high. Therefore, it becomes necessary that if we need to use an SoC-based system, the features need to be pruned as well as the Ensemble ML algorithm must be compressed. When we tried running Ensemble techniques on data gathered from 41 users, we got an accuracy of over 96 %. However, the size of the dataset and features extracted were very large, and the latencies incurred for feature extraction were several minutes. In our future work, we propose to prune the extracted features and use compressed Ensemble techniques that can run on the SoCs with minimum features.

REFERENCES

[1] United Nations, World Population Ageing: 1950-2050. Online [Accessed 12 September 2023] http://globalag.igc.org/ruralaging/world/ageingo.htm

[2] M. E. Tinetti, Clinical practice: preventing falls in elderly persons, The New England Journal of Medicine, 348 (1) (2003), pp. 42-49. DOI: 10.1056/nejmcp020719

[3] J. M. Hausdorff, D. A. Rios, H. K. Edelberg, Gait variability and fall risk in community-living older adults: a 1-year prospective study, Archives of Physical Medicine and Rehabilitation, 82 (8) (2001), pp. 1050-1056. DOI: 10.1053/apmr.2001.24893

[4] N. Thakur, C. Y. Han, A Study of Fall Detection in Assisted Living: Identifying and Improving the Optimal Machine Learning Method. J. Sens. Actuator Netw. 2021, 10, 39. DOI: 10.3390/jsan10030039

[5] Apple Inc., Apple watch user guide. Online [Accessed 12 September 2023] https://support.apple.com/it-it/guide/watch/welcome/watchos

[6] UnaliWear Inc., Unali Kanega watch user guide. Online [Accessed 12 September 2023] https://unaliwear.com

[7] FallCall Solutions, LLC., Fall call lite application documentation. Online [Accessed 12 September 2023] https://www.fallcall.com/apps/FallCall-Lite

[8] D. Naranjo-Hernández, A. Talaminos-Barroso, J. Reina-Tosina, L. M. Roa, G. Barbarov-Rostan, P. Cejudo-Ramos, E. Márquez-Martín, F. Ortega-Ruiz, Smart Vest for Respiratory Rate Monitoring of COPD Patients Based on Non-Contact Capacitive Sensing, Sensors (Basel). 2018;18(7):2144. Published 2018 Jul 3. DOI: 10.3390/s18072144

[9] J. Calvillo-Arbizu, D. Naranjo-Hernández, G. Barbarov-Rostán, A. Talaminos-Barroso, L. M. Roa-Romero, J. Reina-Tosina, A Sensor-Based mHealth Platform for Remote Monitoring and Intervention of Frailty Patients at Home, Int J Environ Res Public Health. 2021;18(21):11730. Published 2021 Nov 8. DOI: 10.3390/ijerph182111730

[10] Anita Ramachandran, Anupama Karuppiah, A Survey on Recent Advances in Wearable Fall Detection Systems, BioMed Research Int., vol. 2020, Article ID 2167160, 17 pages, 2020. DOI: 10.1155/2020/2167160

[11] E. Alpaydin, Voting over Multiple Condensed Nearest Neighbors. Artificial Intelligence Review, 11, 1997, pp. 115–132. DOI: 10.1023/A:1006563312922

[12] K. Vembandasamy, R. Sasipriya, E. Deepa, Heart Diseases Detection Using Naive Bayes Algorithm, IJISET - International Journal of Innovative Science, Engineering & Technology, vol. 2 issue 9, September 2015. ISSN 2348 – 7968. Online [Accessed 12 September 2023] https://ijiset.com/vol2/v2s9/IJISET_V2_I9_54.pdf

[13] D. W. Hosmer, S. L. Lemeshow, Applied Logistic Regression. 2nd ed. Hoboken, NJ: Wiley-Interscience, 2000.

[14] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees (1st ed.), 1984, Routledge. DOI: 10.1201/9781315139470

[15] Liu SH, Cheng WC. Fall detection with the support vector machine during scripted and continuous unscripted activities. Sensors (Basel). 2012;12(9):12301-16. DOI: 10.3390/s120912301

[16] Young-Hoon Nho, Jong Gwan Lim, Dong-Soo Kwon, Cluster-Analysis-Based User-Adaptive Fall Detection Using Fusion of Heart Rate Sensor and Accelerometer in a Wearable Device. IEEE Access, vol. 8, 2020, pp. 40389-40401. DOI: 10.1109/ACCESS.2020.2969453

[17] Kirstine Rosenbeck Gøeg, Ronald Cornet, Stig Kjær Andersen, Clustering clinical models from local electronic health records based on semantic similarity, Journal of Biomedical Informatics, Volume 54, 2015, Pages 294-304, ISSN 1532-0464. DOI: 10.1016/j.jbi.2014.12.015

[18] B. J. Yoon, Hidden Markov Models and their Applications in Biological Sequence Analysis. Current Genomics. 2009 Sep;10(6), pp. 402-415. DOI: 10.2174/138920209789177575

[19] G. Wang, Z. Liu, Q. Li, Fall Detection with Neural Networks, 2019 IEEE Int. Flexible Electronics Technology Conf. (IFETC), 2019, pp. 1-7. DOI: 10.1109/IFETC46817.2019.9073718

[20] W. Xing, Y. Bei, Medical Health Big Data Classification Based on k-NN Classification Algorithm, in IEEE Access, vol. 8, 2020, pp. 28808-28819. DOI: 10.1109/ACCESS.2019.2955754

[21] Andrea Proietti, Massimo Panella, Fabio Leccese, Emiliano Svezia, Dust detection and analysis in museum environment based on

pattern recognition, Measurement, vol. 66, 2015, pp. 62-72 DOI: 10.1016/j.measurement.2015.01.019

[22] Rizwan Shahid, Stefania Bertazzon, Merril Knudtson, William Ghali, Comparison of distance measures in spatial analytical modeling for health service planning. BMC health services research. 9, 2009, 200.
DOI: 10.1186/1472-6963-9-200.

[23] D. Kraft, K. Srinivasan, G. Bieber, Deep Learning Based Fall Detection Algorithms for Embedded Systems, SmartWatches and IoT Devices Using Accelerometers. Technologies 2020, 8, 72. DOI: 10.3390/technologies8040072

[24] SmartWatch dataset. Online [Accessed 12 September 2023] https://userweb.cs.txstate.edu/~hn12/data/SmartFall Data Set / SmartWatch/

[25] Notch dataset. Online [Accessed 12 September 2023] https://userweb.cs.txstate.edu/~hn12/data/SmartFallDataSet/ Notch/Notch_Dataset_Wrist/

[26] SmartFall dataset. Online [Accessed 12 September 2023] https://userweb.cs.txstate.edu/~hn12/data/SmartFall DataSet /SmartFall/

[27] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the Surprising Behavior of Distance Metrics in High Dimensional Space. In: Van den Bussche, J., Vianu, V. (eds) Database Theory — ICDT 2001. ICDT 2001. Lecture Notes in Computer Science, vol 1973. Springer, Berlin, Heidelberg.
DOI: 10.1007/3-540-44503-X_27

[28] G. L. Santos, P. T. Endo, K. H. d. C. Monteiro, E. d. S. Rocha, I. Silva, T. Lynn, Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks. Sensors 2019, 19, 1644. DOI: 10.3390/s19071644

[29] Mohammed Farsi, Application of ensemble RNN deep neural network to the fall detection through IoT environment, Alexandria Engineering Journal, vol. 60, Issue 1, 2021, pp. 199-211.
DOI: 10.1016/j.aej.2020.06.056