A Real-time Patient Monitoring Framework for Fall Detection

by

Dharmitha Ajerla

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
August 2018

Abstract

Fall detection is a major problem in the healthcare department. Elderly people are more prone to falling than others. There are more than 50% of injury-related hospitalizations in people aged over 65. Commercial fall detection devices are expensive and charge a monthly fee to use their service. A more affordable and adaptable system is necessary for retirement homes and clinics. An effective fall detection system would detect fall and send an alarm to the appropriate authority. We propose a framework which uses edge computing i.e. instead of wearable devices sending data to the cloud they will send it to a nearby edge device like a laptop or mobile for real-time analysis. We use cheap wearable sensor devices from MbientLab, an open source streaming engine called Apache Flink for streaming data analytics and a Long Short Term Memory network model (LSTM) for fall classification. The model is trained using a published dataset called MobiAct. Using the trained model we analyze optimal sampling rates, sensor placement and multi stream data correction. Our edge computing framework can perform real-time streaming data analytics to detect falls with an accuracy of 95.8%.

Publications

D[1] D. Ajerla, S. Mahfuz and F. Zulkernine, 2018, BIGMINE 2018, "Fall Detection using Physical Activity Monitoring Data", Proceedings of the 24th ACM SIGKDD BIGMINE 2018 workshop, London, UK, August 2018 (BIGMINE18), 9 pages.

D[2] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, and F. Zulkernine, 2018(In Review), "A Survey of Distributed Data Stream Processing Frameworks", in IEEE Transactions on Knowledge and Data Engineering(TKDE)

D[3] D. Ajerla, S. Mahfuz and F. Zulkernine, 2018(In progress), "A Real-time Patient Monitoring Framework for Fall Detection", Transactions on Large Scale Data and Knowledge centered Systems(TLDKS).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The past few years have seen a rapid increase in people using wearable devices. The use of connected wearable sensor devices is predicted to increase from 325 million in 2015 to 929 million in 2021 [1]. Wrist watch is one of the most popular form of wearable devices. Sensors inside the device measure metrics like the steps taken, stairs climbed, sleep, heartbeat and oxygen levels. Typically the data from a wearable sensor device is sent to a cloud service for analysis and displayed on the dashboard of a connected mobile device. On the cloud the data is synchronized with a data management environment, which is typically provided as a service by the manufacturer of the wearable device. A single accelerometer sensor at 200Hz generates about 2.3GB of data per day. The more sensors or monitoring metrics are added, the more data are generated by the sensors. There is a steady progression towards the era of Internet of Things (IoT) where many such devices will be connected generating tera bytes of data that must be analyzed possibly in real time to provide effective decision support. Usually this data is uploaded and stored in the cloud to do some machine learning analysis. But that leads to wastage of the bandwidth and a decrease in response time.

1.1 Problem Description

According to World Health Organization, a fall is defined as an event in which the person comes to rest onto the ground to any lower level [2]. An estimated 646,000 fatal falls occur each year, making it the second leading cause of death because of unintentional injury, after road traffic injuries below [2]. In all regions of the world, death rates are highest among adults over the age of 60 years. More than 50% of injury-related hospitalizations are seen in people aged over 65 [2]. Consequently, almost 40% of the injury related deaths are from falls in the elderly population [2]. In Canada many retirement and long-term care homes have very high patient to nurse ratio, and falls do not get reported until after some time. Falls also cause hip fracture, another common problem in the elderly population. According to the Ministry of Health and Long Term Care (MOHLTC), the average time taken to treat a minor condition in Kingston General Hospital (KGH) is 2.7 hours [3]. With the rising cost of health care and growing elderly population having chronic diseases, there is an urgent need to shift elderly patient care from the hospital to other patient care facilities such as smart retirement homes. It would facilitate better disease management and ensure wellbeing of all Canadians.

Fall detection and prevention has become an important problem to be solved. External prevention can be done by installing handrails and teaching techniques to avoid falling, but internal fall prevention is trickier. It requires deep research on the neurological activities to analyze which part of the brain depletes their awareness and lowers the reaction time. Also, the gait and posture can be analyzed to correct their balance in case of free fall. Researchers have been working on fall detection for decades. In the 1990s, Personal Emergency Response Systems (PERS) were quite

popular. One of the most common PERS devices is a pendant which is worn on the neck in the form of a chain. A person who needs help would press the pendant and a notification is sent to the caregiver. This eliminates the problem of long waiting time before a person can rise and seek help. However, it is not an optimal solution as Roush et al [4] pointed out that fall often causes fatality in elderly people leaving them disoriented or not conscious enough to act logically such as pressing a button for help. There are also devices available now-a-days which can detect falls automatically and send an alarm to the caregivers or ambulance services. But they are very expensive and require subscription to a monthly service. Alternative solutions are required for Caregiving centers which can detect fall and generate a call for help when needed. A significant amount of research has been done on fall detection using sensors like accelerometers and gyroscopes, which are cheap and included in most of the smart mobile devices available today like the smart phones and tablets. Besides fall detection, there is a dire need for fall prevention systems. Researchers working in this area such as Toshiyo Tamura et al [5] built an innovative system where a wearable air bag inflates automatically when the system predicts a fall 300ms before the fall occurs. Although systems such as these show great progress towards fall prevention, challenges still exist in handling situations like sideward and forward falls. More work is needed for expediting the inflation process of the air bag and in avoiding false alarms.

1.2 Proposed Solution

We propose an edge computing framework which is deployed in close proximity i.e., a maximum of 100ft range from the wireless sensor devices, which collects and performs

preprocessing on the data to only transfer the important data to the cloud. We develop machine learning models to analyze the data and generate notifications in real time to enable call for assistance. We validate our framework for real time fall detection use case scenario by analyzing the accelerometer data collected from the wearable sensor devices. Fig 1.1 shows the general architecture of sensor data processing frameworks.

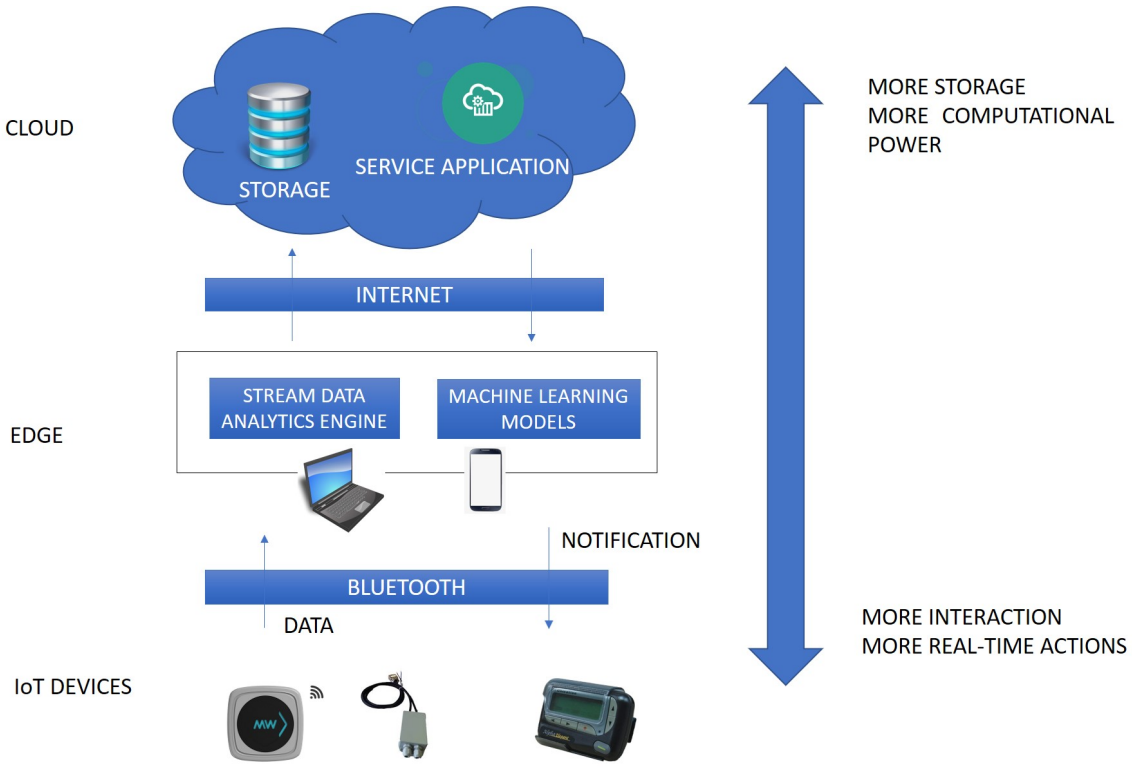


Figure 1.1: Pipeline for streaming and processing wearable sensor data

1.3 Contribution

The research resulted in multiple publications as listed in the beginning of the report, one has been published in a reputed Knowledge Discovery and Data Mining (KDD)

BigMine'18 workshop, one is in progress on the complete work and another journal paper has been submitted to TKDE that used part of this study.

- Study of existing sensor devices and their functionalities.
- Study of state-of-the-art data processing tools
- Development of an artificial neural network model for fall detection, and
- Finally, development of the edge computing framework for real time monitoring of patients for fall detection.

1.4 Organization

The rest of the report is organized as follows. Chapter 2 describes the related work. The fall detection model details is explained in Section 3. Section 4 presents the overview of the framework proposed. Section 5 discusses the results and conclusions are drawn in Section 6.

Chapter 2

Background

The advancement in computational power and big data processing had led to progress in modern day data analytics. There has been great progress in the usage of wearable devices, resulting in research on the acquisition and analysis of data from the IoT or connected devices. Accelerometer and gyroscope are the most commonly used sensors for fall detection with mainly threshold based monitoring and machine learning based predictive analytics approach. In threshold based techniques, a fall is detected when monitored data values exceed pre-defined threshold values. In contrast, machine learning techniques analyzes the data and tries to learn hidden patterns to classify data as fall.

Bourke et al [6] presented a fall detection mechanism where sensors were placed on thigh and trunk. By analyzing the values from these sensors upper and lower fall thresholds were determined. If the resultant value exceeded the upper fall threshold at the trunk, a fall was said to be detected. It was able to detect a fall with 100% specificity. But when tested against the real time data a lot of false positives were observed. In a subsequent study Bourke et al [7] improved the algorithm by monitoring subjects after a fall impact was detected. Though this decreased the number of fall

positives, it still had problems with differentiating between fall and activities like sitting in a car or bus. Kangas et al [8] implemented a similar algorithm by monitoring the posture when a fall impact was detected. The authors inferred that the algorithms performed poorly in detecting backward and lateral falls. But when Bourke et al [9] added features like falling edge time and rising edge time to monitoring algorithms, they performed well with only 1-3 false positives per day. Similarly, Chen et al [10] observed performance improvements using a dot product of acceleration vectors.

Yi He et al [11] used a smartphone placed on the waist for detecting falls and sending a multimedia messaging service to the guardian about the time and location of the user. Median filter was used to smooth raw accelerometer values. Features like signal magnitude area, signal magnitude vector and tilt angle were analyzed against the smoothened values. When the features exceeded a certain threshold a fall was said to be detected.

Quang Viet Vo et al [12] detected fall using a smartphone placed in different places like in hand while messaging or calling, in chest pocket and in pant pocket. A fall was divided into three steps where the first step was when the fall took place, second step when the person hit the ground and third step if the person returned to normal activity or continued to lie down. When a signal exceeded the accelerometer thresholds in the first two steps the orientation data were analyzed for the time between the two steps. If a fall was detected and the magnitudes of acceleration values were constant, orientation data were analyzed for any movement in the third step to decide whether the subject was lying down or continuing his work. This mechanism was tested on five young people, which resulted in 85% accuracy.

Kwapisz et al [13] proposed WSDM (Wireless Sensor Data Mining) to detect fall

using machine-learning models with sensors present in an Android phone. The phone was placed in the front pocket of the subject. The data were collected every 50ms that is 20 samples per second. For feature generation, the data were divided into 10 second segments, each containing 200 readings. Forty-three features were extracted from these raw values. The authors used J48, logistic regression and multilayer perceptron on the features extracted. Among these, multilayer perceptron performed consistently better when compared to others. Ozdemir et al [14] found the multi-layer perceptron neural network model to be most effective for fall detection resulting in around 95% accuracy.

Abbate et al [15] proposed a mechanism to use both threshold and machine learning algorithms to detect falls. The system was implemented using a smartphone and a wearable sensor placed at the waist with a sampling frequency of 50Hz. When a fall was detected using a threshold based technique, it was sent into a classification engine for further analysis. The mechanism also had a notification center using which the user could turn off the false alarms. The false positive data were sent into the classification model again for training. A two layer feed-forward model was used for classification where features were generated from the input signals and fed into the neural network model.The model performed with an accuracy, specificity and sensitivity of 100%.

Khan et al [16] surveyed different machine learning algorithms for fall detection. The authors concluded that Recurrent Neural Networks (RNN) were very well-suited for this problem as the data is usually a sequential time series data. Theodoridis et al [17] built LSTM models using a published dataset called the UR Fall Detection. Two LSTM models, one with simple accelerometer data and another with accelerometer

data rotated at an angle, were compared against a Support Vector Machine model (SVM) and Bourke et al's [6] threshold-based approach. In comparision, LSTM gave better results.

Yacchirema et al [18] used a decision tree based big data model where if a fall was detected at the fog level, data was sent to the cloud for further analysis. Usually the terms edge and fog are used interchangeably. In this case fog depicts a raspberry Pi board which does the near node analytics. The data was collected at a frequency of 200Hz. The decision tree model was trained using an existing historic dataset. The system performed with an accuracy of 91.67% and a precision of 0.937. Similarly we proposed a system where an edge computing framework was used to detect falls in real time. Smart phones were the most widely used devices for fall detection. But as Yi He et al [11] suggested a smart phone placed in the pocket causes too much noise. It is prone to damage in case of a fall as in most of the research it is placed at waist. It is costly to be replaced as well. So, a cheaper alternative to it would be very much preferred. We were using cheap sensors called MetaMotionR from MbientLab. The data was sent from the wearable device to an edge device like a laptop which preprocessed the data using Apache Flink and sends it for further analysis when the magnitude exceeded a predefined threshold. It was analyzed using Tensorflow where a LSTM model was built using a public dataset called MobiAct published by Vavoulas et al [19]. Similar to Vavoulas et al [19], we generated two sets of features, combined them, and then extracted the optimal feature set from them. Different experiments were done to find the best scoring model to incorporate into our framework.

Chapter 3

Fall Detection using Machine Learning Model

We developed a machine learning models based on a published dataset "MobiAct" to detect fall as explained in this chapter. Later we incorporated best performing model in our edge computing framework which is described in the next chapter.

3.1 MobiAct Data

We used MobiAct, a publicly available dataset developed by the Biomedical Informatics and Health Laboratory of the Technological Educational Institute of Crete for training the neural network model [19]. MobiAct is based on a previously published MobiFall dataset [20] created for fall detection. The data was collected using accelerometer, gyroscope and orientation sensors present in a smartphone by placing the device in the pocket of the experimental subject. The orientation sensor was software based, which used data from accelerometer and geomagnetic field sensor. By enabling the parameter SENSOR-DELAY-FASTEST, data was collected at the highest sampling rate possible. The MobiAct dataset contains labeled information about four kinds of simulated falls from fifty-four subjects, and nine kinds of daily activities from fifty subjects. Each activity in the dataset contains time stamp, raw

accelerometer values, raw gyroscope values and orientation data. In our experiments, we used a subset of the data containing only two kinds of daily activities and four kinds of falls. The daily activities denote the two non-fall classes i.e., standing and lying. The four kinds of falls are as follows:

- FOL: forward-lying (fall forward from standing, use of hands to dampen fall)
- FKL: front-knees-lying (fall forward from standing, first impact on knees)
- SDL: sideward-lying (fall sideward from standing, bending legs)
- BSC: back-sitting-chair (fall backward while trying to sit on a chair)

3.2 Data Preprocessing

We preprocessed both the datasets using the following operations in the order listed: Scaling, Feature Extraction, Normalization of feature values, Balancing dataset and Feature Selection.

3.2.1 Scaling

The model generated from the existing data is used to classify real-time streaming data. So to maintain compatibility between the two data sources MobiAct dataset and collected MbientLab dataset are rescaled to the range of (+1g,-1g) before extracting the features. It is done using the rescale function in Matlab.

3.2.2 Feature Extraction

The raw accelerometer values in the dataset are used to extract features for fall detection. Vavoulas et al. [19] stated the importance of optimal features to obtain better

performance. They created two feature sets, A and B. Feature set A is generated from their previous work [21] and feature set B is generated from the WSDM study [13]. We investigated the feature generation methods used in these papers, and based on that, devised an algorithm to generate feature set A and B for our work. Since we are not focusing on the classification of different types of non-fall activities, some of the features in the datasets were ignored to reduce the complexity. Based on the procedure followed by Aziz et al [22], we segmented the raw data into 200 blocks and then generated the feature sets for each block. After feature generation, we combined feature sets A and B, and derived an optimal feature set using a feature selection algorithm. Feature set A and B were calculated using Matlab R2017b.

Feature set A. A total of 54 features were generated in feature set A. For each axis (x, y, z) of the acceleration, 21 features were calculated from the mean, median, standard deviation (STD), skew, kurtosis, minimum and maximum. Using the absolute values of each axis (x, y, z) of the acceleration, another 21 features were calculated from the mean, median, STD, skew, kurtosis, minimum, and maximum. Slope was calculated using Eq 3.1, one for the given axes values (x, y, z) and another for the absolute axes values.

$$Slope = \sqrt{(max_x - min_x)^2 + (max_y - min_y)^2 + (max_z - min_z)^2} \tag{3.1}$$

Four other features were calculated using mean, STD, skew, and kurtosis of the tilt angle (TAi) between the gravitational vector and the y-axis using Eq 3.2.

$$TA_i = \arcsin y_i / \sqrt{(x_i)^2 + (y_i)^2 + (z_i)^2} \tag{3.2}$$

where i denotes the sequence of sample.

Using the magnitude of the acceleration vector, 6 features were calculated from the mean, standard deviation, minimum, maximum, difference between maximum and minimum, and zero crossing rate. Magnitude was calculated using Eq 3.3.

$$Magnitude = \sqrt{(x_i)^2 + (y_i)^2 + (z_i)^2} \tag{3.3}$$

where i denotes the sequence of samples.

Feature set B. A total of 10 features were generated in this set. For each of the three axes (x, y, z), mean, standard deviation, and the average absolute difference were calculated making a total of 9 features. Also, the average resultant acceleration of all the three axes was generated using Eq. 3.4.

$$Averageresultantacceleration = (1/n) * (\sum_i \sqrt{(x_i)^2 + (y_i)^2 + (z_i)^2}) \tag{3.4}$$

where i denotes the sequence of samples.

Combined Feature Set. The feature sets A and B were merged to generate the dataset with the combined features. After excluding the repeated features in feature set A and B, the combined feature set was reduced from 64 to 58 features. We had 7670 samples in the dataset. After feature extraction, each sample had one of 6 classification values and 58 extracted features. Class values included 4 kinds of falls and 2 kinds of non-falls as defined in the MobiAct dataset. The two non-fall classes denote standing (STD) and lying (LYI) positions.

3.2.3 Normalization

Next the extracted features were normalized using Matlab R2017b with the min-max scaling formula given in Eq. 3.5.

$$X = (X - Min(feature))/(Max(feature) - Min(feature)) \tag{3.5}$$

3.2.4 Balancing the dataset

The dataset containing the normalized feature values was very unbalanced containing 5,830 non-fall and 1,840 fall data. We balanced the dataset using Matlab R2017b as follows. The different categories of fall data were merged into a single classification of fall data while the two categories of non-fall data were merged into a single classification of non-fall data. Then the fall data was oversampled to create 2,000 samples and the non-fall data was under-sampled to create 2,000 samples to create a combined balanced dataset containing 4,000 samples.

3.2.5 Feature Selection

ReliefF algorithm is one of the filter based feature selection techniques which achieves superior performance in many applications. So, the Relief-F algorithm of the Matlab R2017b feature selection library was used to select the top ten most important features from the combined feature set [23].

3.3 Experimental Setup

3.3.1 System Requirements

All the experiments were run on a system with the following minimum hardware and software requirements:

Platform: Spyder, Matlab R2017b, IntelliJ, Microsoft Visual Basic Studio

OS: Windows 10 64 bit

RAM: 8GB

CPU/GPU:1.60GHz/AMD Radeon HD 7600G

Data size on disk: 210 MB

Hard Disk: 1TB

3.3.2 Experiements and Validation

To validate our models, we performed six experiments with different datasets as listed below. The experimental setup is shown in Fig. 3.1.

Experiment 1: Combined features and multiclass labels.

The combined feature set of 58 features was used as input and 6 classes were identified as the output. The six classes are 0 and 1 for standing and lying, and 2, 3, 4, and 5 for the four fall types FOL, FKL, SDL and BSC as explained in Section 3.1.

Experiment 2: Combined features and 2 class labels.

A combined set of 58 features were used with 2 class labels which are standing and lying as no-fall, and the second one indicating a fall.

Experiment 3: Selected features and multiclass labels.

Here we used a subset of selected features instead of the full feature set with the multiclass label as we did for experiment 1.

Experiment 4: Selected features and 2 class labels.

We used a subset of selected features with the 2-class labels like experiment 2.

Experiment 5: Balanced dataset, combined features and binary labels.

We used the balanced dataset and the combined features in these experiments but with a binary class label as output indicating fall or no-fall.

Experiment 6: Balanced dataset, selected features and binary labels.

In these experiments we used the balanced dataset with the subset of selected features and a binary class label as output indicating fall or no-fall.

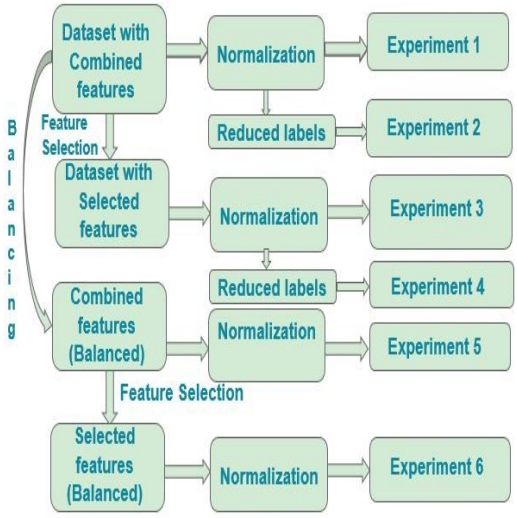


Figure 3.1: The Experiment Setup

The validation of the experiments is performed using the following measures: Precision measures the number of true positives among the positive calls as shown in Eq 3.6. Recall measures the amount of positives that are correctly identified as shown in Eq 3.7. Accuracy measures the amount of predictions that are correct among all the predictions as shown in Eq 3.8.

$$Precision = \frac{numberoftruepositives}{numberoftruepositives + numberoffalsepositives} \tag{3.6}$$

$$Recall = \frac{numberoftruepositives}{numberoftruepositives + numberoffalsenegatives} \tag{3.7}$$

$$Accuracy = \frac{numberofcorrectpredictions}{totalnumberofpredictions} \tag{3.8}$$

3.4 Model Specification

The LSTM is a variant of the RNN that is capable of learning by remembering order dependence in sequence prediction problems [24]. Typically, a LSTM unit consists of a cell, input gate, forget gate and output gate which enable it to model memory and remember information for a long period of time compared to the general recurrent networks. They also eliminate the problem of vanishing gradient usually present in recurrent neural networks. LSTMs are known to work very well for human action recognition.

The LSTM network was built in Python using Keras with Tensorflow in the back-end. It uses softmax activation function, Adam optimizer, mean squared error and categorical cross entropy as the loss function for binary and multi-class models respectively. Each experiment had the feature values at the input layer, 30 LSTM units as hidden layers and number of expected outcome labels at the output layer. Fig. 3.2 depicts the performance of the system against the number of nodes in the LSTM layer. From Fig. 3.2 we can observe that at 30 nodes the network gives the best results, and as the number of nodes increases further, performance degrades due to overfitting. Additionally, a dropout layer of 0.30 is used to avoid over-training of

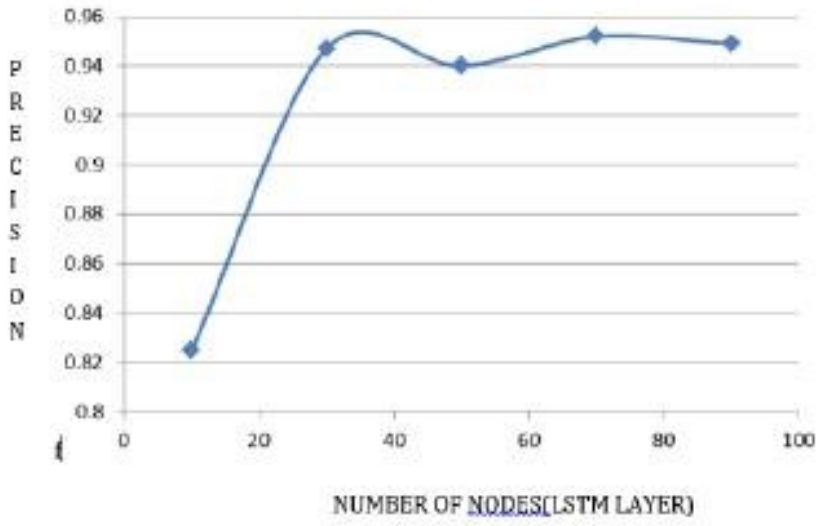


Figure 3.2: Performance (Precision) of the model against the number of nodes in LSTM layer

the models. Each activity in the given dataset was recorded for 10 seconds. So the data was fed into the LSTM accordingly i.e. we used a time step of 10 for all the models. All the output labels were one-hot encoded. One-hot encoding converts the categorical data to a binary sequence for training purposes.

3.5 Model Performance

We built LSTM models for the experiments listed in Section 4.4.4 and validated them. The results are shown in Table 3.1. The best performances were obtained for experiments 2 and 4 for combined and selected features giving an accuracy of 95.22% and 92.18% respectively. Experiments 1 and 3 were able to detect fall but were not able to categorize the different classes of fall which led to low precision and recall values. All the falls were classified as the same fall type. For the same reason

Experiment	Precision	Recall	Overall Accuracy(%)
1	0.792	0.802	80.14
2	0.96	0.951	95.22
3	0.773	0.776	77.52
4	0.939	0.922	92.18
5	0.813	0.797	79.7
6	0.706	0.65	65.08

Table 3.1: : Performance of six experiments using LSTM model.

experiments 2 and 4 did well as these only detected fall or no fall. The accuracies of experiments 5 and 6 with balanced dataset were low because the sequence information was lost in the process of creating the balanced dataset. LSTM models perform better than other models for time series data due to the ability to remember the sequence. In creating the balanced dataset, all the fall and non-fall data were randomly re-sampled. As a result, the sequence information was lost causing LSTM models to perform poorly in the experiments.

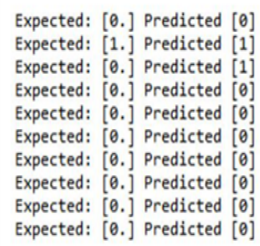


Figure 3.3: Line 3 shows that fall is classified one second later than when expected

From the analysis of the outputs of experiments 2 and 4, it was observed that a lot of false positives occurred because the model was classifying fall one second early or one second later than expected. This is shown in line 3 output in Fig. 3.3. This kind of false positives and false negatives can be ignored because the data is collected at millisecond level. Considering correct predictions made one second earlier or one

Figure 3.4: Confusion Matrix of experiment 4 (a) after ignoring one second delays (b) without ignoring one second delays

Experiment	Precision	Recall	Overall Accuracy(%)
1	0.792	0.802	80.14
2	0.96	0.951	95.22
3	0.773	0.776	77.52
4	0.939	0.922	92.18

Table 3.2: : Performance of LSTM model ignoring one sec delays.

second later as acceptable, we get a very high accuracy of 99% i.e., if a fall is expected at time t and the model detected a fall at time t+1 or t-1, it is considered a true positive.

Fig. 3.4 shows the before and after confusion matrix of experiment 4. Table 3.2 shows the updated performance of the LSTM model for the four experiments where experiments 2 and 4 reach about 99% accuracy with combined features. Experiments 1 and 3 show a smaller improvement for multi-class fall detection compared to the fall or no fall classification as done by experiments 2 and 4 for the same reasons explained before. Table 3.2 does not show the results of experiments 5 and 6 for the balanced dataset as time delay adjustments cannot be made for non-sequential data.

Chapter 4

Edge Computing Framework: An Overview

We present an edge computing framework for fall detection. Real time data from wearable sensor devices are retrieved by an edge device and sent to Apache Flink for pre-processing and Tensorflow for advanced analysis.

4.1 Overview of Framework

We developed a real-time fall detection system using cheap wearable sensor devices. The system applies our fall detection models to real-time streaming data collected from wearable sensors using an edge computing platform as shown in Fig. 4.1. Our system can be used by the healthcare professionals at the clinics, hospitals and retirement homes and can be extended with other machine learning models to monitor activity, sleep, or tracking dementia patients in real-time. The framework consists of:

- Sensors: Multiple sensors can be used to retrieve data from the patients.
- Edge computing node: A node at the edge computing platform retrieves data from the sensors and enables basic data pre-processing and analytics in order to reduce the size of data to be uploaded to a remote server or on the cloud.

- Stream engine: A streaming data processing engine at the edge computing node retrieves the data and performs real-time analysis such as filtering, noise removal, feature extraction, and summarization as necessary.

- Data store: Cleaned and preprocessed data coming out of the stream engine are stored temporarily on the edge platform.

- Machine learning algorithms: Further analysis of the data are enabled by a pre-trained fall detection model deployed on the edge platform to detect fall data.

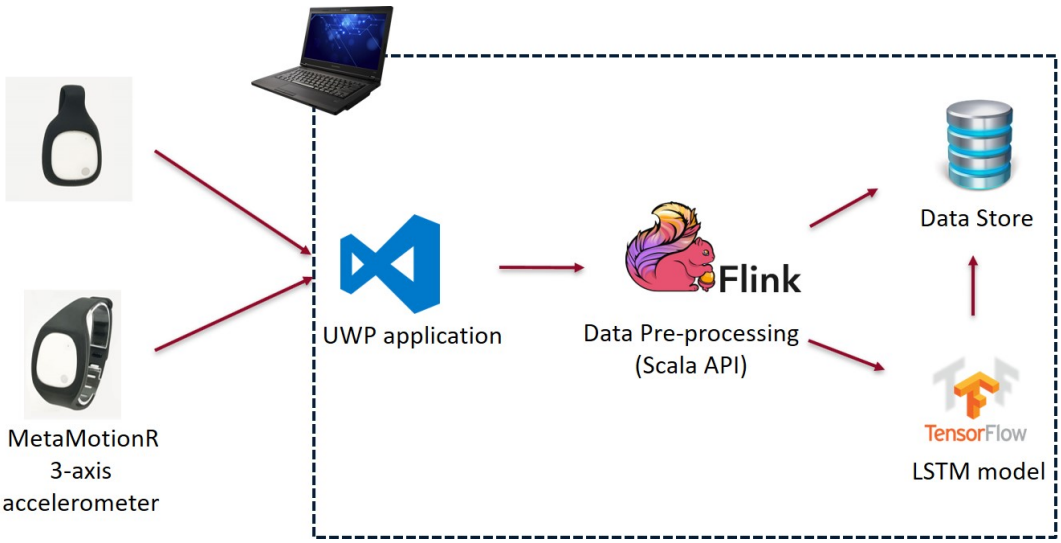


Figure 4.1: An Edge Computing framework for detecting falls

4.2 Components

Our framework consists of five main components: Wearable device, Edge device, Apache Flink, Data store and Tensorflow. We developed this pipeline guided by our

study of fall detection system model in Chapter 3. It was built so that the streaming data is analyzed as it comes and is stored as a csv file temporarily on the edge device.

4.2.1 Wearable Device

The number of units of wearable device shipments were forecasted to be nearly 400 million in 2018 alone [1]. A huge variety of wearable sensor devices are available today. Table 4.1 shows some of the affordable options. All the details were collected from the retail sites of the device manufacturers. Most of the devices only store the data at minute or second level like Garmin and battery life is very less for some of the devices like Teslasz. Among all the devices MetaMotion R from MbientLab was selected because developmental support, battery life and availability of raw sensor data. The device consisted of 8MB of flash memory and allows 10 axes of motion sensing (3-axis Accelerometer + 3-axis Gyroscope + 3-axis Magnetometer + Altimeter/Barometer). The board is powered by a rechargeable 100mAH Lipo battery or replaceable CR2032 coin cell battery. It also has a LED, GPIOS, and a push button switch on the board. The board is based on the nRF52 SOC from Nordic built around an ARM Cortex M4F CPU and a Bluetooth Low Energy 4.2 with 2.4GHz. The sensor data can be stored in the off board memory and downloaded onto a personal device any time. The data is available in the form of a CSV file or as an array of JSON objects. The weight of the device is 0.2 Oz and it is water resistant. For our use case scenario of fall detection we used only the 3-axis accelerometer data. It is a 16 bit accelerometer with values ranging from (-2g,+2g) to (-16g,+16g) with a noise density of 180 $\mu g/\sqrt{Hz}$g/Hz. The connectivity of the device is upto 100ft. It can log data at 1-400Hz and stream data at 1-200Hz [25].

Name	Cost	Sensor Position	Upload data	Waterproof	Battery
Misfit Flash	29.99	Custom	Cloud	Yes	6 months
MiBand 1S	26	Watch	Bluetooth	Yes	10 days
Up move	19	Clip	Bluetooth	Yes	6 months
Vivofit 3	48.95	Watch	Bluetooth	Yes	1 year
MetaMotion R	58	Custom	Bluetooth	Yes	7 days
Arespark	28.13	Watch	Bluetooth	Yes	7 days
Teslasz	16.99	Watch	Bluetooth	Yes	1 day
Fanmis	21.69	Watch	Bluetooth	No	6 Days

Table 4.1: : List of wearable sensor devices.

Figure 4.2: MetaMotionR sensors from mbientLab

4.2.2 Edge Device

An edge computing mechanism brings the computation to the edge of the network so that real time decisions can be taken. It can be done through edge devices like laptops or mobile phones which are closer to the IoT devices. In our framework we use a laptop as it is non-obstructive, mobile and had good computing power to run neural network based analytics. MbientLab provides SDKs to develop applications on the edge device so that sensor data can be collected at our desired rate. We developed

a Universal Windows Platform application(UWP app) in C sharp, using the template provided by MbientLab, to stream raw X,Y,Z-axis of the accelerometer data to the streaming engine. Using the SDK the sampling frequency of the accelerometer was set to 50Hz with values ranging from (-2g,+2g). The UWP application can be deployed on any windows operated machine. Fig 4.3 shows the screenshot of an universal windows application displaying all the available wearable devices. When a desired device is clicked, the application initializes the board and its configuration. After completing the initialization user has the option to start or stop the accelerometer. When the start button is clicked a socket connection is opened and the application starts sending raw accelerometer data to Apache Flink via a socket connection.

Figure 4.3: Universal windows platform application for connecting to wearable devices

4.2.3 Apache Flink

An ideal big data processing platform for real time analysis should be open source and able to analyze streaming data. Some of the platforms that satisfy this criteria

are Spark, Apex, StreamAnalytix, Flink. Among the available big data platforms we chose Apache Flink because of its ability to perform stateful computations on bounded and unbounded data. It can perform computations on different types of data in memory at any scale. Flink can process multiple trillions of events and multiple terrabytes of data each day [26]. It is also easy to deploy as all the communication happens through REST calls. It can run 24/7 without any inconsistencies which is great for streaming applications. Flink provides different features to maintain the consistency like checkpoints, end-to-end exactly once and high availability setup [26]. It provides three levels of abstraction through its API: Process function, Datastream level and SQL/Table level. For our application we use scala API at the datastream level to pre-process the data.

4.2.4 Tensorflow

Tensorflow is an open source library which offers customizable training and layers for the machine learning models. It is one of the libraries that offers advanced machine learning techniques. In our pipeline we used this library to build a long short term memory network which classified whether the activity is a fall or not. We performed six different experiments on the model among which the best one was used for scoring in our data processing pipeline.

4.3 Implementation Details

The components in our proposed framework were integrated by using Bluetooth LE or sockets. The data was sent from the wearable devices to the edge device through

a Bluetooth LE connection. The data would arrive at the edge device at a specified sampling rate and value range. When the accelerometer was started a socket connection was opened and the raw accelerometer values were streamed to Apache Flink which accepted the data in the form of datastreams. Data pre-processing such as data parsing and magnitude calculation were done. When magnitude of axes exceeded a pre-defined threshold value, a 10 second window of data was sent to the Tensorflow via sockets. From the analysis in Chapter 3, it can be inferred that model in experiment 2 gave the best performance. So this model was saved and integrated into our proposed pipeline. Tensorflow extracted the features from the raw values and fed them into this LSTM model. The neural network model then classified if the sent activity was a fall or not. This pipeline was also used to analyze sampling rates, sensor positions and multi stream correction.

4.4 Data Collection using MbientLab Sensors

The data were collected using an accelerometer present in the MbientLab device, MetaMotionR. Experimental subjects helped in collecting the data to validate the data processing pipeline. The subjects were explained about the four types of falls. A yoga mat was provided to avoid any unnecessary injuries. All the data collected went through the same pre-processing steps mentioned in Section 3.1 before they were fed into the model.

The data collected and code for execution of the pipeline are available in Github [1].

[1] https://github.com/Dharmitha/StreamDataMbient

4.5 Summary

In this chapter we present an overview of the proposed edge computing framework where data is collected from wearable sensor devices using an application deployed on edge device like laptop. The application sends the data to Apache Flink for data pre-processing. When magnitude of the axes exceeds a pre-defined threshold value, a 10 second window of data is sent to Tensorflow which contains a pre-saved LSTM model for fall classification. In Chapter 5 the validation tests for the functionality of the framework are discussed.

Chapter 5

Experimentation and Validation of the Framework

We collected data with different configurations to learn about the best possible parameters. The details of the experimental subjects is given in Table 5.1. The final dataset had a total of 64 falls which were annotated by observing the magnitude graph. Using the data processing pipeline, the streaming data was classified into fall and non-fall activities when fall-like activity was detected. To validate the performance of the pipeline the following experiments were done.

Experiment 1: Different sampling rates with the same sensor position

Four types of fall data were collected at frequency 12.5Hz, 25Hz, 50Hz, 100Hz, 200Hz. The device was placed at the waist all through the experiment.

Experiment 2: Varying sensor position with static sampling frequency

Four types of fall data were collected at the best frequency as learned from Experiment 1 by placing the device at different parts of the body like right calf, left side of the chest, right wrist, side waist and right thigh. Sensor is placed at the middle of calf and thigh. Fig. 5.1 shows different sensor positions on the body.

Experiment 3: Two wearable devices with static sampling frequency and sensor position

Experiment	Subjects	Age	Height	Weight	Falls Collected(%)
1	1	22	168	65	20
2	1	22	168	65	20
3	6	22-40	150-171	55-73	24

Table 5.1: : Details of the Experimental Subjects.

Sensor positions and frequencies were selected based on the results of experiments 1 and 2. Four types of fall data were collected.

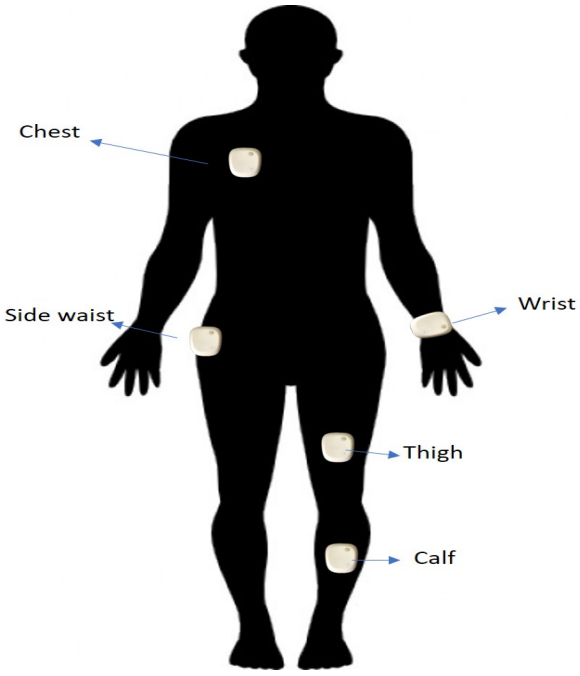


Figure 5.1: Sensor positions at different parts of the body

We analyzed different parameters as listed in section 4.5. Fig 5.2 shows the performance of the LSTM model across different sampling rates. The sampling rate of 50Hz performed the best among all the other frequencies. From our analysis the frequencies below 50Hz were not able to recall all the falls while the frequencies above it performed with similar accuracy. Using a reduced frequency of 50Hz instead of 200Hz, we reduced the amount of data that needed to be stored per day from 2.3GB

to 0.31GB. Nevertheless the data reduction also had a disadvantage of losing information so a combination system should be used as Ren et al [27] suggested. They introduced an energy efficient mechanism where data was usually collected at 50Hz and changed to 200Hz when a possible fall was detected.

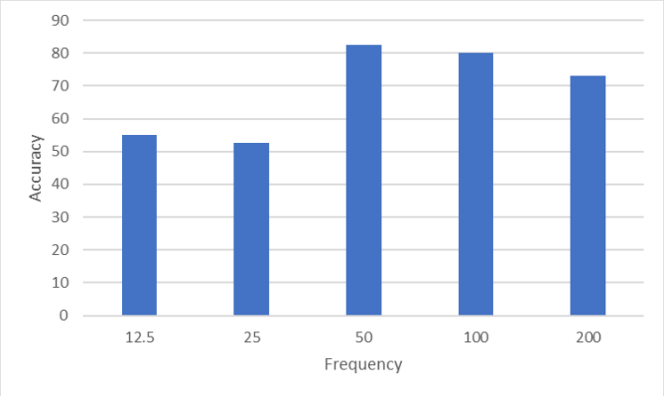


Figure 5.2: Accuracy of the model at different sampling rates

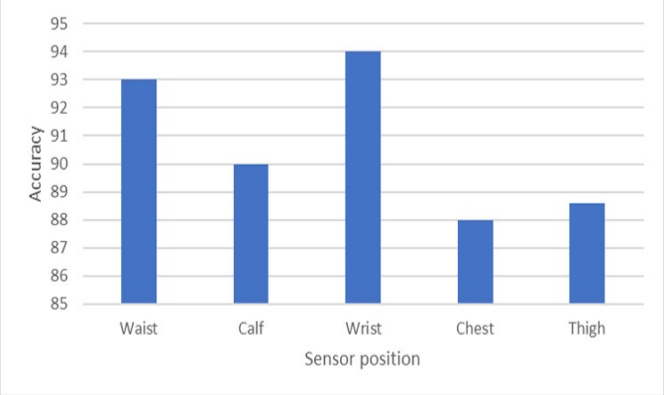


Figure 5.3: Accuracy of the model against the sensor position

Fig. 5.3 shows the performance of the model when sensor is placed at different parts of the body. It could be inferred from the figure that a sensor placed at the wrist position gives the highest accuracy with waist and calf following closely. The waist was found to be the steadiest part of the body with the least noise in the data.

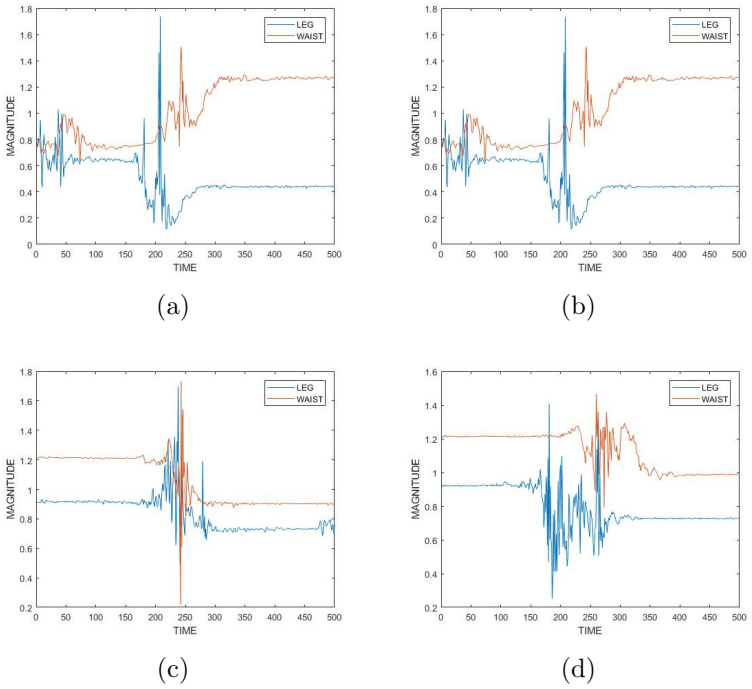
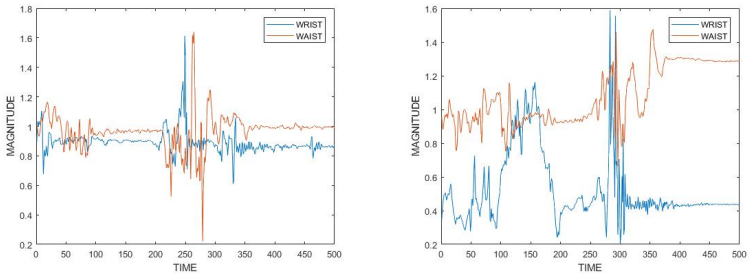


Figure 5.4: Magnitude of fall when waist+leg(calf) combination is used: (a) FOL; (b) FOK; (c) BSC; and, (d) SDL.

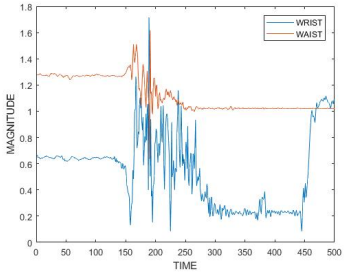
Thigh had higher rate of noise in the data. We took the first three best performances and formed a sensor combination of waist + leg(calf) and waist + wrist. Data was collected from 3 subjects wearing the first combination of sensors and other 3 wearing the second combination of sensors. Fig 5.4 and 5.5 shows the magnitude of falls when waist+leg(calf) and waist+wrist combinations are used respectively. It can be observed that in almost all the cases leg observed fall before waist. Such information could be useful for fall prevention techniques in future. While wrist mostly detected fall at the same time as waist it gave better distinction between falls and other activities when combined with waist.

The performance of the sensor combination models are shown in Table 5.2. Since the model was trained using MobiAct dataset, from the table, we observe that the

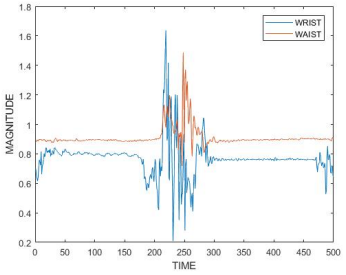


(a)

(b)



(c)



(d)

Figure 5.5: Magnitude of fall when waist+wrist combination is used: (a) FOL; (b) FOK; (c) BSC; and, (d) SDL.

Experiment	Precision	Recall	Overall Accuracy(%)
MobiAct	0.99	0.99	99.62
Waist (50Hz)	0.75	0.64	85.2
Waist + Leg(Calf)	0.95	0.94	94.1
Waist+ Wrist	0.96	0.96	95.8

Table 5.2: : Performance of LSTM model across sensor combinations.

model works great when tested on it. When a real time fall is tested with a sensor placed at the waist at 50Hz, the model was able to detect fall with a precision of 0.75, recall 0.64 and accuracy 85.2%. In the case of multi stream data, both cases were able to detect fall with approximately similar precision and recall after considering one second delay as explained in Chapter 3. The highest performance was obtained with the waist and wrist combination giving 0.96 precision, 0.96 recall and 95.8%

Work	Technique	Performance(%)
Chia Yeh et al [28]	k-NN, SVM	k-NN ACC - 96.2%
Colon et al [29]	Threshold	ACC: 81.3%
Vallabh et al [30]	k-NN, ANN, SVM, NB, LSM, k-NN	SP- 83.78% SE-90.7%
Ajerla et al [D3]	LSTM	ACC- 95.8%

Table 5.3: : Performance comparison of relevant work

accuracy. From the results it can be concluded that multi-stream data can be used to detect falls with high accuracy. Table 5.3 shows the performance comparison of relevant work.

Chapter 6

Conclusion

Fall detection is one of the crucial mechanisms that can enable better care particularly for the elderly people who are more prone to fall to get emergency assistance. We have built an edge computing framework to detect fall automatically using real time monitoring with cheap wearable sensors. Our LSTM model performed the best when correct predictions one second before or after are considered as true positives and when only fall or no-fall are detected. A medium sampling rate of 50Hz and wearable device placed at waist were found to be the optimal settings for fall detection using our selected device. Additionally the use of multi-stream data lead to better performance as well. By training models on existing published datasets and then refining the models proves efficient and can be used more as transfer learning for other use case scenarios. Our future work is focussed on incorporating this framework at the clinics, hospitals and retirement homes, and to test its performance for real patients. In future our system can be expanded by incorporating models to analyze other biometrics like heartbeat of the subject before and after falling, and for tracking patient's activities.

Bibliography

[1] "https://www.statista.com/global-connected-wearable-devices/."

[2] "http://www.who.in."

[3] "http://www.ontariowaittimes.com."

[4] R. E. Roush, T. A. Teasdale, J. N. Murphy, and M. S. Kirk, "Impact of a personal emergency response system on hospital utilization by community-residing elders.," *Southern medical journal*, vol. 88, pp. 917–22, sep 1995.

[5] T. Tamura, T. Yoshimura, M. Sekine, M. Uchida, and O. Tanaka, "A Wearable Airbag to Prevent Fall Injuries," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, pp. 910–914, nov 2009.

[6] A. K. Bourke, J. V. O'Brien, and G. M. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait and Posture*, vol. 26, no. 2, pp. 194–199, 2007.

[7] J. Bourke, Alan K. , van de Ven, Pepijn W.J., Chaya, Amy E., OLaighin, Gearoid M., Nelson, "Testing of a long-term fall detection system incorporated into a custom vest for the elderly," in *2008 30th Annual International Conference of*

the IEEE Engineering in Medicine and Biology Society, pp. 2844–2847, IEEE, aug 2008.

[8] M. Kangas, A. Konttila, P. Lindgren, I. Winblad, and T. Jämsä, "Comparison of low-complexity fall detection algorithms for body attached accelerometers," *Gait & Posture*, vol. 28, pp. 285–291, aug 2008.

[9] A. Bourke, P. van de Ven, M. Gamble, R. O'Connor, K. Murphy, E. Bogan, E. McQuade, P. Finucane, G. ÓLaighin, and J. Nelson, "Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities," *Journal of Biomechanics*, vol. 43, pp. 3051–3057, nov 2010.

[10] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy, "Wearable Sensors for Reliable Fall Detection," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp. 3551–3554, IEEE, 2005.

[11] Y. He, Y. Li, and S. Bao, "Fall detection by built-in tri-accelerometer of smartphone," in *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*, pp. 184–187, Jan 2012.

[12] Q. V. Vo, G. Lee, and D. Choi, "Fall detection based on movement and smart phone technology," in *2012 IEEE RIVF International Conference on Computing Communication Technologies, Research, Innovation, and Vision for the Future*, pp. 1–4, Feb 2012.

[13] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, pp. 74–82, Mar. 2011.

[14] A. Özdemir and B. Barshan, "Detecting Falls with Wearable Sensors Using Machine Learning Techniques," *Sensors*, vol. 14, pp. 10691–10708, jun 2014.

[15] S. Abbate, M. Avvenuti, F. Bonatesta, G. Cola, P. Corsini, and A. Vecchio, "A smartphone-based fall detection system," *Pervasive and Mobile Computing*, vol. 8, no. 6, pp. 883 – 899, 2012. Special Issue on Pervasive Healthcare.

[16] S. S. Khan and J. Hoey, "Review of Fall Detection Techniques: A Data Availability Perspective," 2016.

[17] T. Theodoridis, V. Solachidis, N. Vretos, and P. Daras, "Human fall detection from acceleration measurements using a recurrent neural network," in *IFMBE Proceedings Volume 66*, vol. 66, pp. 145–149, 2017.

[18] D. Yacchirema, J. S. de Puga, C. Palau, and M. Esteve, "Fall detection system for elderly people using iot and big data," *Procedia Computer Science*, vol. 130, pp. 603 – 610, 2018. The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018) / The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018) / Affiliated Workshops.

[19] G. Vavoulas, C. Chatzaki, T. Malliotakis, M. Pediaditis, and M. Tsiknakis, "The MobiAct Dataset: Recognition of Activities of Daily Living using Smartphones," *International Conference on Information and Communication Technologies for Ageing Well and e-Health (ICT4AWE)*, no. Ict4awe, pp. 143–151, 2016.

[20] G. Vavoulas, M. Pediaditis, E. G. Spanakis, and M. Tsiknakis, "The MobiFall dataset: An initial evaluation of fall detection algorithms using smartphones,"

in *13th IEEE International Conference on BioInformatics and BioEngineering*, pp. 1–4, IEEE, nov 2013.

[21] G. Vavoulas, M. Pediaditis, C. Chatzaki, E. G. Spanakis, and M. Tsiknakis, "The mobifall dataset: Fall detection and classification with a smartphone," *IJMSTR*, vol. 2, pp. 44–56, 2014.

[22] O. Aziz, M. Musngi, E. J. Park, G. Mori, and S. N. Robinovitch, "A comparison of accuracy of fall detection algorithms (threshold-based vs. machine learning) using waist-mounted tri-axial accelerometer signals from a comprehensive set of falls and non-fall trials," *Medical & Biological Engineering & Computing*, vol. 55, pp. 45–55, jan 2017.

[23] Z. Wang, Y. Zhang, Z. Chen, H. Yang, Y. Sun, J. Kang, Y. Yang, and X. Liang, "Application of relieff algorithm to selecting feature sets for classification of high resolution remote sensing image," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 755–758, July 2016.

[24] S. Hochreiter and J. Urgen Schmidhuber, "LONG SHORT-TERM MEMORY," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] "mbientlab.com."

[26] "flink.apache.com."

[27] L. Ren, W. Shi, Z. Yu, and Z. Liu, "Real-time energy-efficient fall detection based on ssr energy efficiency strategy," *IJSNet*, vol. 20, pp. 243–251, 2016.

[28] C.-Y. Hsieh, C.-N. Huang, K.-C. Liu, W.-C. Chu, and C.-T. Chan, "A machine learning approach to fall detection algorithm using wearable sensor," in *2016*

International Conference on Advanced Materials for Science and Engineering (ICAMSE), pp. 707–710, IEEE, nov 2016.

[29] L. N. V. Coln, Y. DeLaHoz, and M. Labrador, "Human fall detection with smartphones," in *2014 IEEE Latin-America Conference on Communications (LATINCOM)*, pp. 1–7, Nov 2014.

[30] P. Vallabh, R. Malekian, N. Ye, and D. C. Bogatinoska, "Fall detection using machine learning algorithms," *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–9, 2016.