HIGHLY-EFFICIENT FOG-BASED DEEP LEARNING AAL FALL DETECTION SYSTEM

David Sarabia-Jacome, dasaja@teleco.upv.es
Regel Gonzalez-Usach, regonus@doctor.upv.es
Carlos E. Palau, cpalau@dcom.upv.es
Manuel Esteve mesteve@upv.es
Universitat Politècnica de València, Spain

Abstract

Falls is one of most concerning accidents in aged population due to its high frequency and serious repercussion; thus, quick assistance is critical to avoid serious health consequences. There are several Ambient Assisted Living (AAL) solutions that rely on the technologies of the Internet of Things (IoT), Cloud Computing and Machine Learning (ML). Recently, Deep Learning (DL) have been included for its high potential to improve accuracy on fall detection. Also, the use of fog devices for the ML inference (detecting falls) spares cloud drawback of high network latency, non-appropriate for delay-sensitive applications such as fall detectors. Though, current fall detection systems lack DL inference on the fog, and there is no evidence of it in real environments, nor documentation regarding the complex challenge of the deployment. Since DL requires considerable resources and fog nodes are resource-limited, a very efficient deployment and resource usage is critical. We present an innovative highly-efficient intelligent system based on a fog-cloud computing architecture to timely detect falls using DL technics deployed on resource-constrained devices (fog nodes). We employ a wearable tri-axial accelerometer to collect patient monitoring data. In the fog, we propose a smart-IoT-Gateway architecture to support the remote deployment and management of DL models. We deploy two DL models (LSTM/GRU) employing virtualization to optimize resources and evaluate their performance and inference time. The results prove the effectiveness of our fall system, that provides a more timely and accurate response than traditional fall detector systems, higher efficiency, 98.75% accuracy, lower delay, and service improvement.

Keywords: IoT, big data, fog computing, cloud computing, deep learning, AAL, health, AHA, Fall

1. INTRODUCTION

One of the main concerns in current society is the aging of the population along with the special caring needs required by the elderly. In Europe, it is expected an increase of 45% on the number of people aged over 65 in the next 20 years, overpassing 220 million, while the proportion of elderly on the whole population will keep steeply increasing [1]. Several problems affect the elderly's health and quality of life. Falls are the most frequent accident in the aged population that can involve injuries and hip fracture, in addition to serious health issues derived from a late attention. According to the World Health

Organization (WHO), approximately 30% of the people over 65 suffer accidentally one or more falls per year. Health damages have a very negative impact on the elderly's health, mobility, and quality of life. Typically after a fall, the person affected waits a long time until another person detects the fall and alerts emergency services [2]. A frequent consequence of this long wait is a severe deterioration of the subject's health; it is common that the injured person suffers in addition dehydration, hypothermia, and pneumonia. In those cases, there is a considerable risk of death within the next 6 months as a consequence of these affections. A fall of an aged person not timely assisted has a very negative impact on his or her health, quality of life (QoL), independence and mobility. Thus, a timely detection of this accident is critical to minimize health damages and currently, it represents an important medical concern.

Ambient Assisted Living (AAL) aims to enhance the quality of life of elderly population through the leverage of modern technologies such as the Internet of Things (IoT) [3], addressing problems such as the inattention after a fall. AAL also comprises Active and Healthy Ageing (AHA) [4], a field defined by WHO as "the process of developing and maintaining the functional ability that enables wellbeing in older age" [5]. Fall detection is both an important AAL and AHA application. The use of IoT sensors to monitor (i.e. biomedical sensors, wearables, and multimedia devices) conjointly with an intelligent analysis of these data allow for an accurate fall detection. Currently, there are three main approaches to fall detection: wearable-based, vision-based, and ambiance-based [6]. These systems require big data analysis to extract patterns for fall detection from the massive monitoring data collected from the IoT devices. This analysis is performed through cloud-based big data analytics employing Artificial Intelligence techniques such as Machine Learning (ML) and Deep Learning (DL). On the cloud, ML and DL models for the detection of falls are developed and trained. These models will allow to accurately determine the existence of a fall that requires assistance, eliminating false positives and negatives that degrade the quality of service. ML techniques have been widely employed on the aforementioned main approaches for fall detection. Among the different types of fall detector approaches, DL techniques have been principally explored by vision-based solutions in order to improve the accuracy of fall detection. After the establishment of a model, data from IoT sensors is analyzed on the cloud to resolve if a fall occurs, in all current approaches, due to the huge processing resources that this recognition task requires. Though, the use of cloud computing analytics has drawbacks such as inherent delay, high bandwidth occupancy, and network congestion. Those setbacks make cloud computing inference non-appropriate for delay-sensitive applications on which timely alerts are critical.

To overcome these limitations, fall detection approaches have recently adopted fog computing for performing the inference process. Fog computing extends part of the cloud processing and storage capabilities on networking elements at the edge of the IoT network (i.e. fog nodes such as IoT gateways

or routers) [7]. As far as the processing nodes are adjacent to the IoT sensors, this fact minimizes or even avoids the transfer of massive sensors data through the external network to the online cloud services. As a result, the response time of the AAL system is minimized, providing a more timely reaction detecting fall events -without unexpected delays due to network congestion. Few studies in fall detection have placed ML models previously trained in the cloud on edge devices (smartphones and IoT-gateways) to take advantages of the fog computing. The use of DL models on an IoT-Gateway-based solves setbacks of a smartphone approach (high dependence on battery life and on the commitment of carrying the device, high intrusiveness) but represents a very complex challenge as IoT-gateways are generally implemented using resource-constrained devices (e.g. Raspberry Pi, Odriod). Few studies based on DL models to detect falls (mainly vision-based) affirm to employ predictive analytics on resource-constrained devices but they did not provide information about the deployment nor conducted a practical and experimental deployment of DL models. This gap in the literature, regarding documentation, not only happens on the specific case of fall detection, but on all practical use cases of DL on the fog, to our knowledge. Unlike ML models, DL models require considerable resources to detect fall efficiently without affecting the inference time. Thus, the deployment of DL fall detection models in resource-constrained devices such as fog nodes is a very challenging task. High efficiency in resource usage is critical. Also, it is necessary a previous analysis of its limitations based on a detailed practical and experimental process.

In this paper, we present and discuss an innovative and highly efficient intelligent AAL & AHA system based on fog computing that performs DL inference tasks entirely on the fog. Our system detects falls on the ground through this DL inference process. Our proposed 3-layer fog-cloud architecture supports fall detection service and can be employed with other different AAL services using other different DL or ML models. Our solution uses a non-invasive wearable-based approach to monitor the elderly's daily life activities. The data collected by the IoT sensors are sent in real time to a fog node. As far as the fog nodes have very limited storage and processing resources, and inference and predictive analytics require considerable processing capabilities, our solution applies a very highly-efficient usage of resources through virtualization technologies. In this way, the system presents a highly innovative and efficient architecture for the deployment of DL models entirely on fog nodes. Our system supports the remote deployment and management of DL models on these edge devices for performing inference tasks and provides a real-time fall detection and an alert notification service to caregivers' smartphones. The techniques and mechanisms employed for the efficient deployment of the DL inference model are described and documented, covering this gap in the literature, not only in fall detectors but also in any practical use case of DL fog inference. Finally, we provide a comprehensible evaluation of the effectiveness of our system for fall detection. We prove that it provides enhanced accuracy, efficiency, and a very timely response, better than achieved through a cloud approach.

This paper presents some contributions with the objective to tackle the challenges and limitations of the current literature in the study area. To sum up, our system architecture achieves:

- The enablement of deep learning predictive analytics in the fog by designing a fog node (Smart IoT Gateway) architecture.
- Reduction of the time to detect anomalies by placing the predictive analysis on the fog, close to the IoT devices.
- Higher accuracy for fall detection using deep learning techniques.
- A highly-efficient and automatic deployment through the use of virtualization technologies that solves the complex challenges of resource limitations on the fog for DL inference.
- A timely alarm mechanism in case of fall detection to caregivers' mobile phone through our app.
- A practical and experimental implementation that provides information about the deployment of deep learning techniques on fog nodes, as well as the advantages and limitations of this approach.

This paper is structured as follows: Section 2 reviews the background and current literature in fog computing and AAL systems. Section 3 presents our high-level architecture for overcoming the limitations of cloud approaches and for enabling a highly-efficient deployment of DL models on the fog. In section 4 we present our AAL system for elderly fall detection. In Section 5, we present the experimental results and the evaluation of the system effectiveness. And finally, Section 6 presents our conclusion and future work.

2. BACKGROUND

2.1. Deep Learning

Machine learning (ML) is a branch of computer science that allows for the extraction of knowledge from data (learning) and to make predictions on the behavior of this data through the use of inference models. ML provides a set of algorithms such as decision tree, linear regression, Support Vector Machine (SVM), Artificial Neural Networks (ANN), Bayesian networks, among others, for the creation of models in order to automate make statistical inferences. Particularly, the ANN has gained momentum recently due to the high capacity to learn from complex data patterns. The ANN is composed of artificial neurons which are connected each other composing a layer of neurons, called neural network. The increment of

layers in a neural network has enabled the creation of a new set of algorithms, that is the founding of deep learning [8].

Deep learning (DL) refers to a branch of ML that employs complex neural network architectures composed of more than three layers for building models to detect and predict patterns from unstructured data. Currently, there is a wide variety of DL architectures for extracting data information. In particular, the architectures of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are receiving special attention due to their extensive application for image and speech recognition.

Among them, RNN best adapt to problems involving time-series data or time-sequences data, which are typical in IoT systems. As smart devices generally produce data that follows a sequence at a given time, RNNs are suitable for addressing IoT problems and learning from IoT big data. This type of neural networks has the ability to keep past states in memory thanks to the feedback from previous layers or from the same layer. RNNs are architectures composed of a block of memory cells as hidden layers. The block of memory cells receives a sequence of data X_t, where t is the time-step index. The time-steps refer to a position in the sequence and not to a passage of time in the real world [9]. Fig. 1 shows an RNN architecture with a single block of memories as hidden layers. RNN architectures vary depending on the type of feedback and the structure of their memories. The most popular memory variants are the Long-Short Term Memory (LSTM) and the Gated-Recurrent Unit (GRU). The LSTM memory cells are capable of retaining information for a long period of time [10]. To do this, an LSTM memory cell is composed of three gates (input, output and forget), as shown in Fig. 2. The LSTM version used as a reference is the one proposed by [11], implemented by the following composite function:

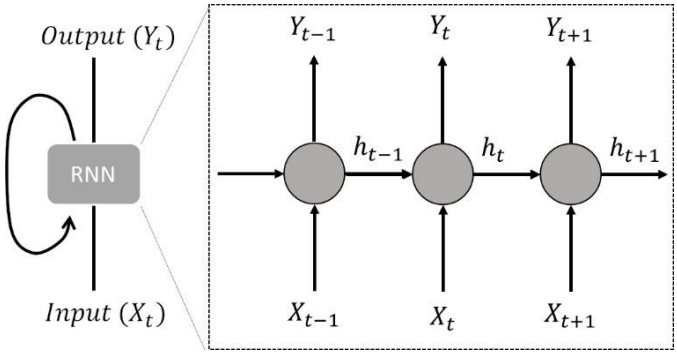

Fig. 1. RNN architecture

$$i_t = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{fx}X_t + W_{fh}h_{t-1} + b_f) \quad (2)$$

$$O_t = \sigma(W_{ox}X_t + W_{oh}h_{t-1} + b_o) \quad (3)$$

$$\tilde{C}_t = tanh(W_{cx}X_t + W_{ch}h_{t-1} + b_c) \quad (4)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (5)$$

$$h_t = O_t * \tanh(C_t) \quad (6)$$

Where *i, f, o, c* are the input gate, forget gate, output gate, and cell activations vectors. The h_{t-1} is the hidden previous cell state, and the weights matrix of each gate (i.e. W_{hi} is the hidden-input gate matrix, W_{fx} is the input-forget gate matrix). Also, the *b* is the bias for each gate. The logistic sigmoid function is represented by σ. Using these functions, the LSTM memory is able to control when new information can enter the memory, when a part of the information is forgotten, and when the information of the cell is used in the result.

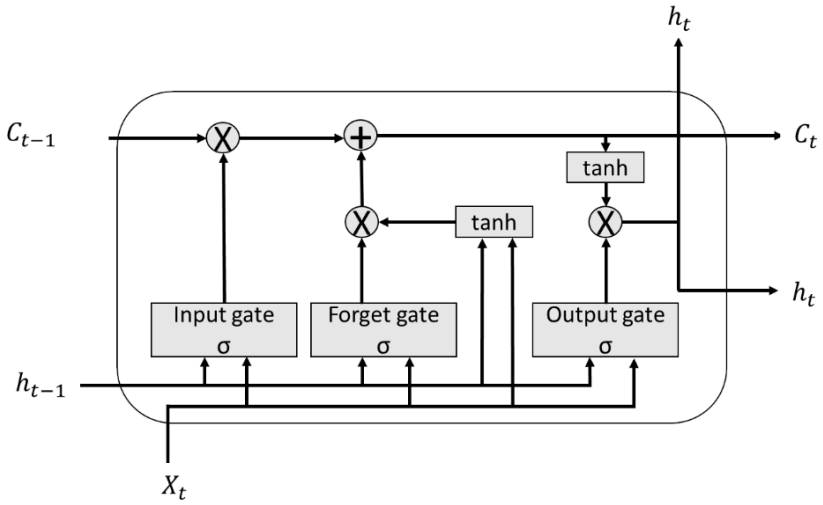

Fig. 2. LSTM cell memory

The GRU architecture unlike the LSTM architecture does not have an internal memory, nor use an exit gate. Instead, the GRU has two gates (update and reset gate) as shown in Fig. 3. For the GRU version used as a reference is the one proposed by [12], implemented by the following composite function:

$$u_t = \sigma(W_{ux}X_t + W_{uh}h_{t-1} + b_u) \quad (7)$$

$$r_t = \sigma(W_{rx}X_t + W_{rh}h_{t-1} + b_r) \quad (8)$$

$$\tilde{h}_t = tanh(W_{cx}X_t + W_{hh}(r_t * h_{t-1}) + b_c) \quad (9)$$

$$h_t = u_t * \tilde{h}_t + (1 - u_t) * h_{t-1}) \quad (10)$$

Where *u, r, h* are the update gate, reset gate, and cell activation vectors. The h_{t-1} is the hidden previous activation and the h_t is the candidate activation. Using these functions, the unit is able to decide how much of the previous unit is updated in its content, and how to add the new one with the previous content.

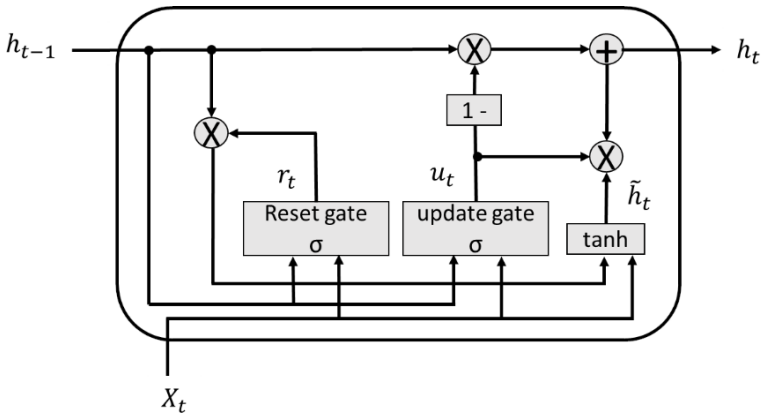

Fig. 3. GRU unit gate

2.2. Related Works

One of the most widely studied problems on AAL and AHA is the detection of elderly's falls on the ground. Falls are the most common accidents in aged population, which require timely medical assistance in case of suffering health damage. Approximately 30% of the people over 65 suffer accidentally one or more falls per year [2]. More alarmingly, on people over 80 years this rate rises up to 50%. In general, falls are related to normal activities of daily living (ADL), a fact that makes prevention difficult. Very often these accidents occur in indoor environments, at the elderly's home. This fact makes difficult the detection and alert to medical services if the affected person loses mobility and is unable to seek help. Frequently assistance is not provided until another person notices the accident, especially in the case of elderly who lives alone. A typical situation after a fall accident is the "long lie" that involves a high risk of serious health complications (i.e. dehydration, pneumonia, and hypothermia among others). In this situation, the aged person waits a long period of time until is assisted, with a subsequent degradation of medical condition that can even lead to death in the following months in many cases [13]. Therefore, a fall of an aged person not timely assisted has a very negative impact his or her health, QoL,

independent living and mobility. Fall detector systems that can alert caregivers or emergency services on time are a social need. The mission of a fall detection system is to detect this event in real time and alert emergency systems and healthcare professionals. In this way, medical assistance is provided as soon as possible, avoiding further health complications derived from a long wait. The main benefits of this system are the avoidance of the long lie risks, preservation of independent life in the aged population and improving their quality of life.

In the current literature, fall detection systems can be classified into three main approaches: video-based, ambiance-based, and wearable-based systems [6]. The video-based fall systems employ cameras to monitor the daily activities. Some proposals such as [14]–[16] have demonstrated that the use of this devices and ML and DL techniques have good accuracy on fall detection. However, this approach has a high cost, limited coverage area, and irrupts with elderly privacy. Overcoming part of these limitations, ambiance-based systems are an alternative approach, which are characterized by sensing environment parameters such as infrared temperature, electromagnetic waves, floor vibrations or sounds [17]–[19]. Despite this approach have many advantages such as low cost, more coverage area, and non-intrusive approach, it presents an important drawback such as a high rate of false alarm (inaccuracy). Finally, as another fall detection approach, wearable-based fall detectors [20]–[22] are characterized by using embedded sensors attached to elderly in order to detect the fall. The proposals used a tri-axial accelerometer and a gyroscope which can be worn in some part of the human body such as a chest, waist, and writs, to sense variations in gravity during the elderly daily activities. Even though the wearable-based has many advantages such as low cost, low weight, and low power consumption, it presents some disadvantages such as dependency on battery life and the oblivion on wearing the device. Recently, the wearable devices have acquired relevant acceptation in the community due to the visible benefits to track health rates and sports activities. The wearable-based approach has been chosen in our proposed system due to its advantages compared to other approaches for the specific case of fall detection.

Despite the different advantages and disadvantages of each approach, the effectiveness of the system relies principally on the methodology employed to detect the falls. In the current literature, a wide range of methodologies has been studied, which is mainly based on thresholds, ML, and DL algorithms. The threshold-based methods using value conditions that are obtained by a previously statistics analysis (mean, standard deviation, kurtosis, among others) of the sensor data (accelerometers and gyroscopes) [23][24]. This methodology has reached low model accuracy (83.96%, 61%, respectively). The limitations of the systems based on thresholds are solved by using ML techniques such as Support Vector Machine (SVM), K-Nearest Neighbors (kNN), Naive Bayes (NB) Stochastic Gradient Descent (SGD), Decision Tree and Gradient Boosting to classification the activities of daily living. For example, [25]

proposed two ML models based on SVM and kNN to detect fall from data coming from a wrist-worn smartwatch, which reached an accuracy of 98% and 96.13%, respectively. Although the accuracy is the most common indicator to evaluate the performance of the fall detector algorithm, there are others indicators such as precision (sensitivity) and specificity that can be used for obtaining a more detailed performance measurement. A system with high sensitivity and low specificity produces many false alarms, while a system with high specificity and low sensitivity can miss emergency events. The second case is very prejudicial in terms of service performance and must be drastically avoided: the system must not ignore any real fall. False positive are preferable to miss real emergency events. Though, false positives have to be avoided at the extent of possible as far as reduces the quality of service and represent an inefficient alert to health services. To improve the specificity and sensitivity in ML models features engineering algorithms has been developed. For example, [26] proposed a short time min-max feature from a tri-axial accelerometer data, which is used in an SVM algorithm to detect the falls. This feature engineering uses a 0.1 seconds sliding window to evaluate the minimum and maximum resultant accelerations during the critical phase fall. Then, these values are introduced in an SVM to detect the fall. By using the indicators of sensitivity and specificity, [27] makes a comparison between the threshold-based and ML algorithms. According to these studies, ML algorithms improve the sensitivity and specificity in the fall detection process. However, the ML algorithms require an extensive feature extraction stage and in some cases a manually feature extraction or feature engineering, thus ML lack flexibility in these terms. In contrast, DL techniques provide automatic feature extraction in addition to higher accuracy and precision on fall detection algorithms. Video-based fall detection systems principally employ DL techniques for differentiating fall and non-fall events, such as the use of CNN which uses convolutional layers to extract features and fully connected layers (multiple neuronal layers) [28], [29]. Another DL approach is the use of RNN to extract a feature from a time-series data (video, wearable or ambiance). For example, [22] proposed an RNN-LSTM model to detect fall from sequential accelerometer data. The LSTM model is composed of two LSTM layer with 200 memory units and two fully-connected layers. This DL approach improved the accuracy, sensitivity, and specificity of the ML models, reaching 98.57%, 96.67%, 100% respectively. Undoubtedly, DL techniques have improved fall detection systems by providing a flexible and automatic feature extraction and enhancing the accuracy of the model, but they present some disadvantages: DL requires a larger amount of data than ML for creating and training prediction models, as well as higher processing and memory resources.

Since ML and DL algorithms require high capabilities to pre-process data and then train the models, cloud computing technologies have been used to support the development of fall systems algorithms. In real environments, the prediction process (e.g. fall detection) is also placed in the cloud to take advantages of the powerful cloud capabilities. However, cloud computing presents the disadvantages

of high latency, high network bandwidth consumption, and probable network congestion, among others, thus it is not appropriate for a delay-sensitive use case such as fall detection. To overcome these drawbacks, the following approaches proposed the physical placement of the fall models closer to the IoT data source: smartphone-based and IoT-gateway-based. The smartphone-based approach consists on the use of a mobile application with the ML and DL models, running over a mobile OS (e.g. Android, iPhone OS, among others), conjointly with the use of the embedded smartphone's accelerometer sensors or an external wearable sensor. This type of approach employs a smartphone as a fog node. For example, [25] proposed the uses of a smartwatch connected to a smartphone using Bluetooth Low Energy (BLE) for detecting falls. On the smartphone runs on Android mobile application with embedded ML models (SVM and KNN), reaching (93.6 and 88% sensitivity. In [30], the authors deployed an SVM, kNN, and a Self-Organizing Map (SOM) models into a smartphone Huawei P8 lite, and they evaluated the hardware resources performance achieving a memory occupancy of 40Mb (SVM, kNN) and 28Mb (SOM) and a response time of 9s (SVM, kNN) and 0.04s (SOM). In this way, the authors demonstrated that the most accurate model is not necessarily the most suitable to run in real-time on a smartphone. The drawbacks of this approach are its high dependence on the smartphone battery life, and the increment of difficult of the system by adding a new device that elderly care about it, made it a more intrusive system. The IoT-gateway-based approach solves these drawbacks by using embedded platforms or resource-constrained devices such as Raspberry Pi, Odroid, PandaBoard ES, among others. These devices are characterized by its resource limitation for processing and storing tasks but are widely used in IoT environment for its easy customization, programmability, flexibility, and low cost. [31] proposed a fall detection system based on fog-cloud computing to enable the deployment of a decision-tree model to a smart IoT gateway (raspberry pi). In this case, the authors tested that the ML models can be deployed to resource-constrained devices (Raspberry Pi) because they are low learning-parameters models. The computational requirements of each classifier depend on the models' parameters, and as far as resources are constrained, a model with fewer parameters can provide a faster detection and better performance than a more accurate one with a larger number of parameters that require higher computational capabilities. Accordingly, the DL models are high learning-parameters models (approximately 1'000.000 parameters in a CNN model) due to their deeply neural networks architectures (more than 3 layers), so they require more capacities such as processing and memory to be deployed. In this case, the deployment of DL models on resource-constrained devices can introduce a significant processing latency: the limitation of resources do not allow the same performance of the inference task as achieved on a cloud-based approach. It is necessary an effective strategy to deploy a DL model on a fog device to take advantage from the benefits of this approach and improve the response time in real environments. The deployment of DL inference models on fog nodes (smart IoT gateways) are not documented in the current literature. In some cases such as

[18], the authors stated the use of a resource-constrained device (Odriod) to support DL inference but they did not bring information about how to perform the deployment nor evaluated the performance of the model. The use of an inference smart IoT gateway solution improved the response time to attend the emergency risk but releases a challenged task to deploy DL models for its limitations of resources.

To the best of our knowledge, this is the first proposal for a highly-efficient deployment of DL fall detection models on a fog node (smart IoT gateway) in a real environment that details this development, proposes a methodology for its application and offers a practical and experimental evaluation of the DL model performance. Our proposal uses a wearable-based approach for its advantages and its benefits demonstrated in smart-health. Also, our proposal uses a DL model to detect falls for its high performance in solving this problem in comparison to ML models. Finally, our proposal uses a smart IoT gateway as the fog device for performing DL inference instead of other options (e.g. smartphones) due to its flexibility, robustness, the possibility to provide multi-attendance and multi-services, and that it barely requires any human maintenance action (unlike smartphones).

3. SYSTEM ARCHITECTURE

The system architecture is designed to support our AAL system that implements a fall detection service. The architecture is based on a fog-cloud approach that employs cloud big data analytics resources for training a DL model and performs DL model inference for detecting falls on a fog device. The fog device employed is a smart IoT gateway due to its non-dependence on battery life, low intrusiveness and lack of the necessity of carrying it, compared with a smartphone approach.

As far as the service must be robust enough to detect true falls, never omitting them, it must present very high accuracy. To do so, the use of the DL approach is chosen as it can bring better performance than ML approach. In addition, this service must have a very timely response time, as far as an early warning of fall events in the elderly is critical to provide assistance on time. For this aim, in the inference process for detecting falls with our trained DL model, data is both processed and temporary stored in the fog instead of the cloud. This substantially reduces the amount of data that must be sent, since fog computing is responsible for processing the data and only sends part of the results to the cloud. In the same way, fog computing reduces the response time executing the detection process on the fog, thus sparing the network transport latency and identifying fall events in lower time.

Our system architecture presents a hierarchical division into 3 layers (device, fog, and cloud). This feature can be appreciated in Fig. 4, that shows an overview of our high-level system architecture. Our proposed 3-layer system architecture is flexible and can easily implement other different AAL and AHA services that require the use of DL models on fog nodes and employ different sensors.

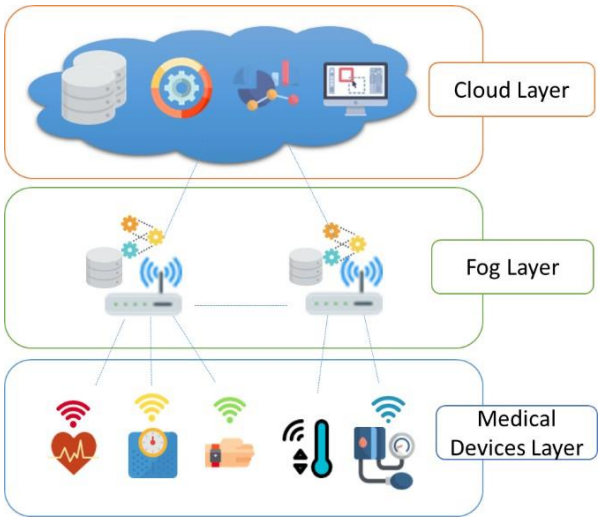

Fig. 4. High-level fog-cloud architecture

3.1. Medical devices Layer

To provide innovative services such as monitoring of elderly's well-being, home environment, and daily life activities, our AAL system proposed uses wearables and other sensors to monitoring medical and environmental parameters in a non-invasive manner. The environmental sensors and wearables compose a sensor network that employs low-power wireless technologies such as Bluetooth, Zigbee, 6LoWPAN, Wi-Fi (Wireless Fidelity) which allow user mobility. Devices require optimized Machine to Machine (M2M) communication protocols to send the monitoring data to the fog nodes.

3.2. Fog Computing Layer

The fog computing layer is composed of several fog nodes (smart IoT gateways) that are distributed within the IoT ecosystem of AAL. Each fog node is placed on a different subject's home. The system allows the remote deployment and management of several independent systems that belong each one to a different user that is monitored. These fog nodes are located near the IoT devices, which can therefore connect to them and send IoT data. Unlike the IoT sensors, the fog device is able to store and process these data. The fog node has a series of highly-efficient modules in terms of resource management to optimally exploit the limited resources of the edge device, thanks to optimization through virtualization technologies. These modules cover the functions of communication, storage and processing of IoT device data. Fig. 5 shows the architecture of the fog node. Each module is detailed below:

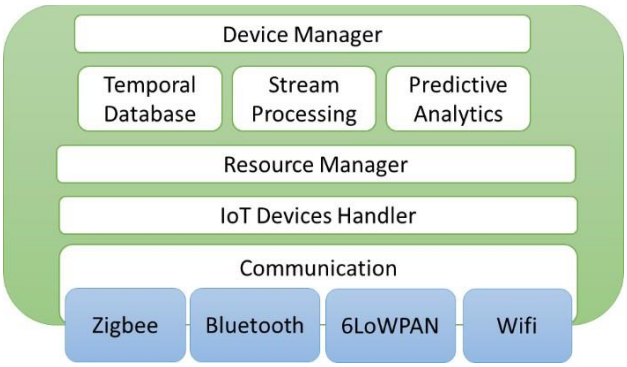

Fig. 5. Fog Node Architecture

3.2.1. Wireless Communication and Interoperability

This module supports communication protocols and wireless technologies to enable the connection of IoT devices. It supports the most commonly used low-power communication technologies and protocols (Bluetooth, Zigbee, 6LoWPAN, Wi-Fi). Additionally, the module provides technical, syntactic and semantic interoperability. Technical interoperability is achieved by providing connectivity to IoT devices, and by performing protocol conversion of the data flow. Messages from the IoT source are encapsulated in the correct technology protocol used to redirect the messages. Syntactic interoperability is provided through format conversion. The content of the message is adapted to the appropriate format to be forwarded to its recipient. Finally, the module provides semantic interoperability to ensure that the content of the message can be interpreted correctly.

3.2.2. Devices Communication Handler

IoT devices use communications protocols to exchange messages with a fog node, allowing bidirectional communication. The module uses an MQTT broker to facilitate M2M communication (between IoT devices and fog nodes). This mechanism also enables the actuators to receive commands from the system to execute an action. Moreover, this module allows to send notifications to health professionals and caregivers to alert them about a risk situation. This mechanism minimizes the critical assistance time in the fall service by sending an alert on time.

3.2.3. Stream Processing

The stream processing module is responsible for transforming the data received from the device communication handler. The module executes a group of operations (aggregations, filtering, compression or fusion data from several sensors) pre-defined by the application to transform the data. The operators are executed within a temporary window in order to reduce the data dimensions. This temporary window

moves over time to transform the data streams. Transformed data are sent to the cloud and, in some applications, to the predictive analytics module.

3.2.4. Predictive Analytics

The predictive analytics module implements Deep Learning-based services for the detection and prediction of risk situations. This module performs the highest consumption of resources in the fog node, thus it is critical an optimal resource-efficient implementation of this block, what represents a major challenge in the creation of our system. The data generated by the IoT devices is a time-sequence data, which is usually exploited through the use of RNN architectures. RNN usually have fewer parameters than a convolutional neural network. Though, the module is capable of supporting CNN-based services with model graph optimizations. The module receives the data transformed by the stream processing through a REST API defined in the services. This executes an inference process using the DL model and emits a result. The result is sent to a temporary storage and in case of detection of a risk situation, a notification is sent to the MQTT broker.

3.2.5. Short Term Store

The short term store module consists of a database which temporarily stores the results obtained from the data processing module and the predictive analytics module. The data stored in this module has a short life time or utility predefined by the application, since the fog node is a device limited in storage capacity and a malfunction of this module can hinder the full proper functioning of the fog node. This module provides mechanisms to generate queries of the data stored in order to obtain a descriptive analysis of the data, which is sent to the cloud to be stored. The database is based on documents to store the JSON formats that are used as the format of the data. The module uses the NoSQL MongoDB database engine to provide the storage service.

3.2.6. Data Manager

The data manager module stores preconfigured queries to be employed in the short-term store. These queries allow obtaining a descriptive analysis of the data stored in the node. The queries results are formatted to the standard HL7 used to manage medical data. Later, the data manager sends the data to the cloud using the Apache Kafka message exchange platform. The fog node acts as Kafka producer to publish the data in the publish/subscribe Kafka broker configured in the cloud layer.

3.2.7. Resource Manager

The fog node has limited resources so it requires a mechanism that allows adequate management of them. Lightweight virtualization technologies have proven to be suitable for this type of device because of their efficient use of resources. Thus, container-based technology allows better management of resources

through the isolation of applications or services at the operating system level, avoiding the virtualization of hardware and drivers. Each module runs in containers isolated from each other but sharing resources. We employ Docker technology instead of other options as it is the most widely used and supported for the deployment of virtual containers. The Docker engine provides APIs to manage each module that allows limiting the resources allocated to each of them.

3.2.8. Device Manager

The device manager module allows remote management of the device. In this case, the module functions as an orchestrator of the activities and the operation of the other modules. The module exploits the exposed Docker engine APIs in order to establish a management of the fog node Docker modules. This module uses a Nodejs server to expose a REST API that handles the fog node used by operators POST, GET, PUT, and DELETE.

3.3. Cloud Computing layer

The cloud computing level consists of a group of servers that compose a cluster, on which are deployed big data analytics platforms that manage data generated by IoT devices. The main functions of this level are to store, process and analyze data, and serve as a back-end of the AAL system. The cloud computing level is composed by the modules shown in Fig. 6.

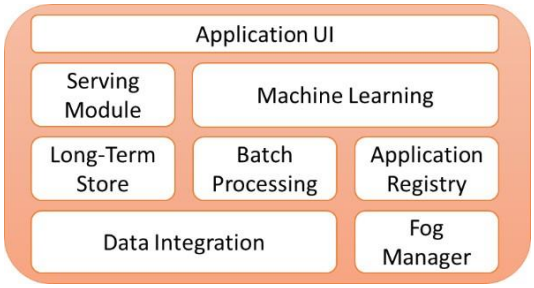

Fig. 6. Cloud layer architecture

3.3.1. Long-Term Store

The long-term store module consists of a distributed file system that stores data from one or several data sources. The main data source is the system fog nodes that send from time to time information reports of the data collected by IoT sensors. In addition to this source, the data from external sources such as Smart City, a health system, among others, can also be stored. In terms of file storage, the file system provides redundancy, scalability, high availability and privacy. Specifically, we employ the Hadoop Distributed File System (HDFS) in our system. To optimize file storage, each file is transformed into a

columnar representation of data using Apache Parquet. The parquet files are stored incrementally in the HDFS system.

3.3.2. Batch Processing

The batch processing module consists of a distributed data processing platform mainly composed of a cluster of servers. The main task of this module is the extraction and processing of the data stored in the long-term store. For this, the processing module employs Apache Spark framework which offers in-memory processing, providing better response times than disk processing. This type of abstraction is known as the Resilient Disk Distributed (RDD). In addition, Apache Spark provides a stack of libraries that facilitate the processing of data (i.e. SparkSQL, DataFrames, and others). These two libraries are used to perform a descriptive data analysis. End users can receive the results of this data analysis through the serving module. From the point of view of DL, these libraries allow performing the data pre-processing required to train and test the dataset.

3.3.3. Data Integration

The data integration module connects the fog nodes with the cloud level. In addition, it allows connection with other data sources such as data generated by a Smart City or data from other health systems (i.e. a system for managing patient records). For the connection with the cloud level and with other data sources, the module uses the publish/subscribe Apache Kafka messaging platform. Apache Kafka enables the exchange of messages through the use of Kafka Brokers (Kafka servers) which have configured topics that serve as message classifiers. Kafka connectors facilitate the connection with other platforms and other services (SQL, HDFS, MQTT, JDBC). The fog nodes have the role of message producers that publish messages associated with a defined topic. On the other hand, the module of a long-term store gets the role of message consumer by subscribing to one or several topics.

3.3.4. Machine Learning

The machine learning module provides a framework and libraries for the development of ML and DL models. The module uses data previously stored in the Long-term store and processed by the batch processing module. Subsequently, the data are exploited using techniques, methods, and algorithms in order to find patterns, correlations and detect anomalies. Models based on RNN and CNN are developed to extract more information from data of the time-sequence type. To do this, the module uses the TensorFlow framework in conjunction with the Deep Learning Keras API model development library. In addition, the module uses the Scikit-learn libraries in a complementary way in the preparation and evaluation of the models. The models are stored in the Long-Term Store module. Another important task is the optimization of the graph models. To do this, the module provides functions and libraries for the optimization of graph models by a graph simplification, quantization, and constant folding techniques.

These optimization techniques are important to reduce the size of a DL model and to improve the performance of the model in fog nodes.

3.3.5. Serving Module

The serving module allows the results of the batch processing of the data to be visible to patients and medical staff. For this aim, the serving module has a database that stores the results of the batch processing and a graphical interface that shows the results of the descriptive analysis of the data. We use a NoSQL Apache Cassandra database for the storage of results. This type of database supports CQL query language. Users can launch pre-configured CQL queries from the graphical interface to be executed from the web server, and visualize the results obtained. The web server and the graphical user interface are developed using Node.js.

3.3.6. Fog Manager

The device manager allows the management of the geo-distributed fog nodes. This module uses a lightweight Docker container manager based on Portainer UI to connect to the REST API of each fog node and manage it remotely. The fog nodes are registered on it, and they can be accessed remotely to deploy and manage applications and services based on DL.

3.3.7. Application Registry

The module is a repository of images for each application used in the fog nodes, facilitating the deployment of own images that are developed to support systems based on ARMv6 and ARMv7 architecture. After training the DL model, it is added to a Docker image ready to be deployed by using the device manager to the fog nodes. This new Docker image is stored in the repository. In this way, the models and applications of DL are kept under privacy in a particular Docker registry.

4. AAL SYSTEM: IMPLEMENTATION AND DEMONSTRATION

4.1. System Implementation

The implementation of the system covers four stages: implementation of the IoT nodes, of the fog node, of big data management, and finally of the DL model.

4.1.1. IoT Nodes

The system medical device layer consists of IoT nodes that are distributed in the elderly's home and attached at belt-wearable to be worn. The IoT nodes perform the function of monitoring the elderly activity and health, as well as the home environment. The activity monitoring node is a wearable belt prototype, composed of a tri-axial accelerometer sensor ADXL345 ± 16g, 13 bits of analog to digital

converter - ADC, a microcontroller NodeMcu v1.0 V3 (based on ESP8266 and include a WiFi module) and a Power Pack V1.2 (3.7V 3800 mAh battery). The WiFi module allows the connection of this device to the fog node. Meanwhile, the environment monitoring node is responsible for measuring temperature, humidity, and air quality. The IoT node is implemented with DHT11 temperature and humidity sensors and MQ135 air quality sensor. These sensors are connected to an Arduino Uno (ATmega328, 5v, 32KB Flash, 2KB RAM, 1KB EEPROM). An additional ESP8266 WiFi module allows the connection to the fog node. Finally, the elderly's health is monitoring by using a weight scale. For this, the A&D Medical precision body weight scale is used, which uses Bluetooth connection.

4.1.2. Fog Node

The system fog computing layer is composed of a fog node that implements the necessary modules described in the design section of the fog node (3.2). The fog nog employs a Raspberry Pi2 (RPi) model B (QuadCore @ 900MHz ARMv7 Cortex-A7, 1GB RAM, MicroSD 8GB) and the Raspbian Stretch operating system. Additionally, the system uses the Docker engine for the deployment of Docker containers in the fog node. The deployment of Docker containers on ARMv7 architecture devices uses Docker images other than those generally used for deployment on X86-64 architecture servers. For this reason, a series of new images are created for the support of fog node modules based on ARMv7. Four Raspbian Stretch image-based are created for the support of MQTT as part of the device communication handler using Eclipse Mosquitto, storage of data temporal using MongoDB, predictive analysis using TensorFlow and Python Numpy, Matplot, and Flask libraries, and for data processing using the Python library Streamz. In addition to these, the Docker image provided by Portainer UI is used to remotely manage the device and deploy containers and applications based on DL.

4.1.3. Big Data Management

The system cloud layer consists of Big Data platforms and DL frameworks and libraries to implement the modules described in the section of the cloud layer design (section number). Cloud services are implemented using 5 instances of Amazon EC2. Three instances (T2.micro, 1 vCPUs, 2.5 GHz, 1GB RAM, 8GB HDD, Ubuntu 16.04 SO image) are used to deploy the server cluster using YARN. The HDFS and Apache Spark file system is deployed in the cluster. The HDFS file system is configured to have 3 DataNodes and 1 NameNode, with default replication factor. While the Apache Spark 2.6 is configured to have a Master Node and 2 Worker Node. The fourth instance (T2.medium, 2 vCPUs, 2.3 GHz, 4GB RAM, 8GB HDD, Ubuntu 16.04 SO image) configured with TensorFlow 1.8.0 as the backend of Keras API 2.1.6 and with the Python libraries Numpy, Matplot, and Scikit -Learn to support the development of DL models. The fifth instance (T2.micro, 1 vCPUs, 2.5 GHz, 1GB RAM,

8GB HDD, Ubuntu 16.04 SO image) uses Docker engine to isolate services in containers. The services of the device manager, application registry and serving are isolated with their corresponding applications.

4.1.4. Deep Learning Model

The decision of using a DL model, despite the complex challenge of its deployment on the fog, was due to DL techniques provide automatic feature extraction in addition to higher accuracy and robustness on fall detection algorithms. The exploitation of the data is done using an RNN architecture because this type of networks has better performance for time sequence data. To evaluate and make a comparison of functionality and test the efficiency of the models in the fog nodes, two RNN models are trained using the GRU and LSTM approach. The dataset presented by [23] is the one used to train the RNN models. This dataset details the activities of daily living of 23 adults and 15 elderly. The dataset is composed of a total of 4497 text files, of which 2701 belong to ADL such as walking, running, sitting, among others and 1796 belong to falls. Each file collects the data generated by 2 tri-axial accelerometer sensors (ADXL345, MMA8451Q) and 1 gyroscope (ITG3200). The dataset is stored in the proposed HDFS file system in the cloud layer. Subsequently, the data is processed using the Apache Spark platform and its Spark Dataframe library. This library transforms each text file into a table called dataframe. From this dataframe, only the information from the ADXL345 accelerometer sensor is extracted, forming a new dataframe composed of the time-steps corresponding to each axial (x, y, z) of the accelerometer. In addition, the processing identifies the time-steps where the fall occurs to be then labeled with a value of 1, while the time-steps where there is no fall are labeled with the value of 0. The dataframe with the sensor information ADXL345 accelerometer (x, y, z) and labels are stored in a new text file in the HDFS file system. The process is repeated for each file that is part of the dataset. Since the generated files have different time-steps as each activity is recorded during variable periods of time, the size of each file varies in number of time-steps. Thus, the time-steps of the files are in a range of 2999-30000. To obtain a uniform dataset in time-steps size, the size of time-steps is defined in 2999 for each file since this is the minimum that we will find. Thus, files with a size greater than 2999 can be divided into several samples, increasing even more the amount of files that will pass through the model for training. The 2999 time-steps are equal to 15 seconds.

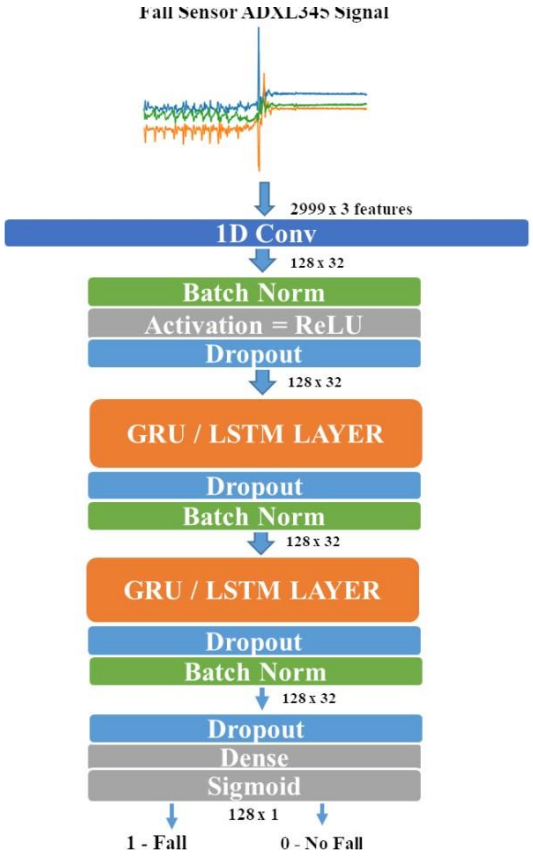

Fig. 7. Fall Detector Deep Learning Model

The DL trained models to detect the events of falls have the architecture shown in Fig. 7. The models have as input 2999x3 dimensions related to the 2999 time-steps for the tri-axial accelerometer (x, y, z). The DL models have the following structure:

- 1-Dimension Convolutional layer: the first stage of the models uses 32 convolutional 1D filters that allow to extract low-level features and reduce the dimensions of the data. The filters have a dimension of 192x1 without stride. As a result, the dimensions are reduced to 128x32, which require less computational capabilities than the input dimensions. A Batch Normalization stage is added to adjust and scale the activations, while a ReLU activation function and a Dropout stage is added to reduce the complexity of the model and avoid overfitting.

- LSTM/GRU layers: the next stages are two layers of LSTM/GRU which read the time sequences from left to right. The first layer is composed of 32 cells/units LSTM/GRU that receives as input a sequences of data X_t, and produces an output at the time-step t = 1,2,3…,128. In the same way, the second LSTM/GRU is composed of 32 cells/units LSTM/GRU that receives as input the sequence results of the first LSTM/GRU layer, and

produces an output at the time-step t = 1,2,3…,128. At the exit of each of these stages, it is necessary to use a Batch Normalization and Dropout stage to address the activations scale and overfitting.

- Dense layer: to regularize the values between 0 and 1 and make the predictions the system uses a dense/fully-connected layer time distributed and a sigmoid activation function. The prediction has a dimension of 128x1, which corresponds to 64 milliseconds. The models use a unidirectional RNN instead of a bidirectional RNN to detect falls almost immediately they occur, and not wait for the end of the sequences to determine if there was a fall. This configuration is particularly chosen to reduce the inference time.

4.2. System Demonstration

4.2.1. Predictive Analytics Services Deployment and DataFlow structure.

After training the model in the cloud, the DL predictive model is isolated in an independent Docker container. This is done by employing a Docker image only for the predictive analytics module. We deploy remotely the container on the fog nodes through the fog manager placed in the cloud, sparing the need to physically access to the edge device to configure it and deploy the application. Our system can be deployed in a single fog node, or several, depending on the number of houses on which the fall detection service is deployed. Each elderly's home is covered by a single IoT gateway. The application deployment in the fog nodes is automatic and remote. Thus, it is not necessary to be physically where the fog node is located to configure it and deploy the application. Furthermore, the set of operations required in the deployment is reduced for the system operator and service developer through the use of the device manager of each fog node and the fog manager in the cloud, making the deployment operations more efficient and simplified. The modules of the fog node and the fall detection service are deployed using the Portainer UI from the cloud layer. Fig. 8 shows the web interface and the containers deployed in the fog node together with the fail detection service.

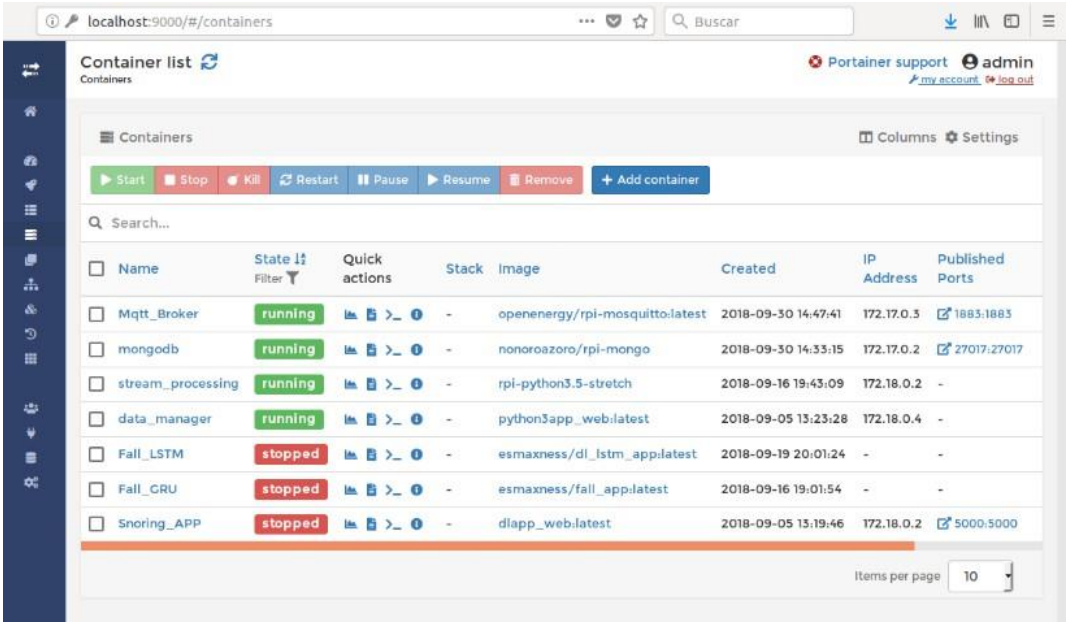

Fig. 8. Web interface manager to deploy remote container

The application deployed in the node fog follows the data flow shown in Fig. 9. The wearable sends monitoring data to the fog node using MQTT transport protocol through a WiFi connection. The device communication handler receives the data and sends it through an MQTT broker using the topic "sensor/accelerometer". The processor module receives this data through an MQTT subscription to the sensor's topic, and processes it using a window configured for the reception of 2999 time-steps. The time-steps received in the window are sent to the predictive analytics module using a POST request. The predictive analytics module contains the fall detection service that exposes a REST API using the Python Flask library. The input data (2999x3) is received through the REST API and it is checked if it has a valid file size. In response to this prediction request, the application informs about the positive or negative detection of a fall with a response in JSON format. In case of fall, the response is sent to the device communication handler using an MQTT client as part of the application employing the topic "prediction/fall". From the MQTT broker it is distributed to the short-term store to store the fall event by subscribing to the respective topic and also a fall alert is issued for medical and care professionals to a mobile application. This mobile application based on MQTT is subscribed to the topic, so caregivers and medical staff can receive an alerts if a fall occurs to a patient, as Fig. 10 shows.

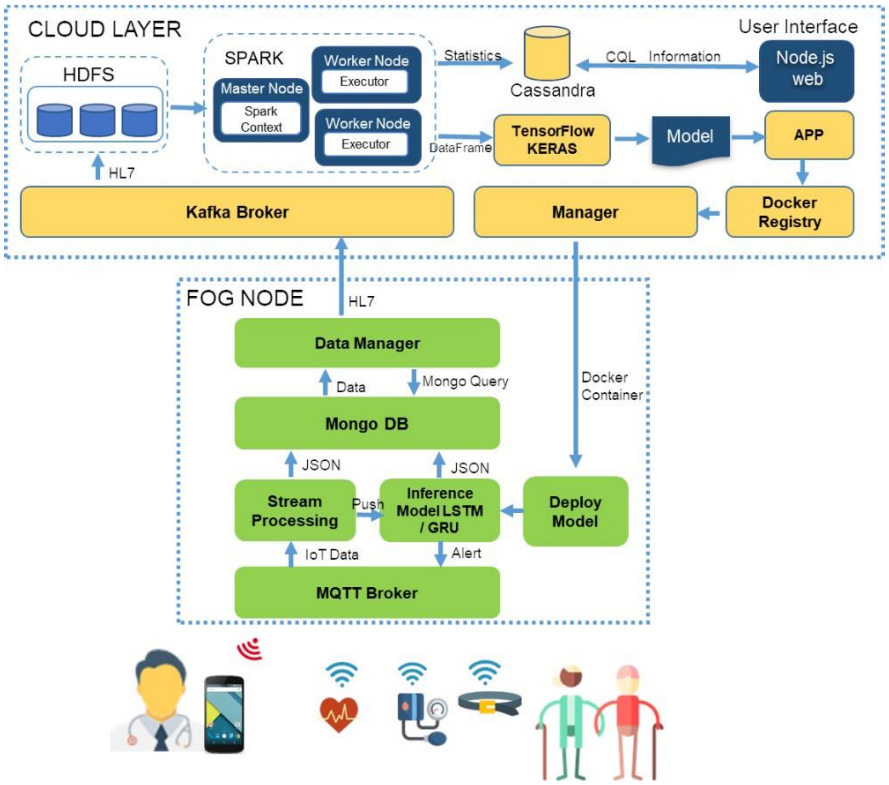

Fig. 9. Workflow pipeline to detect in healthcare architecture

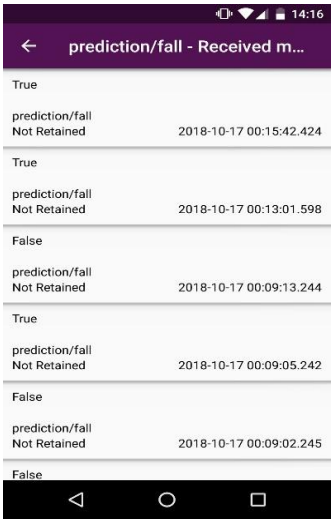

Fig. 10. Fall mobile application

On the other hand, there are environmental sensors that allow measuring the temperature, humidity and air quality. These sensors are essential in an AAL system as considered in this case. In this way, the sensors send the collected data to the MQTT broker of the fog node by using the topic "sensors/environment". The processor module receives this data through an MQTT subscription to the topic of the room sensors and processes them through the use of a sliding window. The window unlike the

processor for predictive analytics moves in time to reduce the amount of data collected. Within each time window, an average value is calculated for both temperature, humidity, and air quality. Those average values are sent to the cloud to be stored and processed using a Kafka topic. In this way, the amount of data that should be sent to the cloud is reduced. In the cloud, the data is received, stored and processed and then presented to the end user. Similarly, the body weight scale is used to sense the body mass indicator (BMI). This device sends data to the MQTT broker when the elderly use the scale. This data is processed (transformed and formatted) to send in JSON format to the Mongo DB. The data manager queries from Mongo DB every week to collect the elderly weight status. This data is formatted using the HL7 format to be sent to the Kafka broker in the cloud by using a topic for each elderly. In the cloud, the data is stored in the HDFS.

5. RESULTS

5.1. Performance of the Predictive Analytics

DL models trained in the cloud layer are evaluated to determine their performance through the calculation of accuracy, sensitivity, and specificity. For that, the confusion matrix is determined by labeling true positives (TP), false positives (FP), true negatives (TN) and False negatives (FN). Fig. 11 shows the confusion matrix for both DL models (LSTM/GRU). The sensitivity is the ability to identify true positives (i.e., real falls), while specificity is the ability to identify true negatives (i.e., real no falls). It is important to have the best trade-off between the two indices, this means a system with few or none false alarms and few or none miss alarms. The calculation of accuracy, sensitivity, and precision is done by using the following equations:

$$sensitivity = \frac{TP}{TP + FN} \quad (11)$$

$$accuracy = \frac{TP + TN}{TP + TN + TP + FN} \quad (12)$$

$$specificity = \frac{TN}{TN + FP} \quad (13)$$

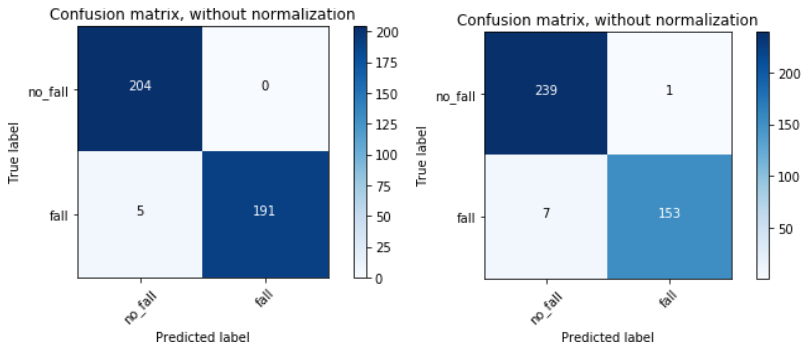

Fig. 11. Confusion matrix LSTM and GRU models

Table I. presents the results of accuracy, sensitivity, and specificity of our proposed system and a comparison to wearable-based related works. In comparison to ML models (SVM, kNN), the DL models (LSTM, GRU) reach better accuracy. An exception is the SOM model that consist of two feed-forward neural networks that reached the similarity accuracy as our LSTM model. On the other hand, our models sensitivity is the higher reached in the comparison. That means our system identifies better true positives (real falls) that other systems (smartphone-based and IoT-Gateway). Finally, the kNN specificity proposed by [25] is the highest reached using a smartphone-based approach. However, the trade-off between the two parameters (sensitivity and specificity) is not the best, so the system can miss emergency events and the system should not ignore any real fall. In this sense, the DL models (LSTM, GRU) provides a better trade-off than ML models. Our approach presents the best sensitivity-specificity trade-off, the higher accuracy with low model's error rate and the higher sensitivity to detect real falls. Also, our models require 128 unit for each RNN layer in comparison to [22] which required 200 units, so our models are highly efficient due to they require less processing and memory capabilities.

Table I. Comparison of Related Works Models Performance

Parameter	**Our**		[25]		[22]	[30]		
	LSTM	**GRU**	**SVM**	**kNN**	**LSTM-Acc**	**kNN**	**SVM**	**SOM**
Accuracy [%]	**98.75**	98.40	98	96.13	95.71	70.6	95.6	98.7
Sensitivity [%]	**97.60**	97.15	80	69	96.67	84.2	82.7	91.2
Specificity [%]	**97.44**	95.63	99	99.2	95.00	41.0	91.9	98.1

Moreover, the models are also evaluated based on the inference time of a model. This indicator refers to the response time of the model and it is calculated from the moment the input data of the model is sent to the prediction service until the model returns a response. This parameter is critical in performance. In case of a risk situation, the system response time should be as low as possible. Table II presents comparatively the inference time of the models executed in the fog node and the cloud. It can be noted that the response time in the fog node is lower than the response time in the cloud. The delay in the

cloud is higher due to the additional network latency of routing data packets from the fog node to the cloud. Also, the response time of the GRU model is lower than the response time of the LSTM because the structure of the GRU unit has 2 gates, unlike the 3 gates of the LSTM. Thus GRU inference requires less mathematical calculations and consequently lower processing time. Next, Table III presents comparatively the inference time of our models and the models presented by [30], to make a comparison between both edge inference approaches (Smartphone-based and Smart-IoT-Gateway). Our models' response time are better than ML models deployed on the smartphone, but the SOM (Neural Network) model provides the lower response time to inference fall. SOM response time is lower because it has fewer models parameters, and the smartphone resources capabilities (Huawei P8 Lite) are higher than our resource-constrained devices (Raspberry Pi 2). Thought the inference time of SOM is fewer, the model is not efficient due to the model performance (sensitivity and specificity) is not optimal, as the Table I shows. To sum up, our DL models reduces the time inference by placing the inference closer to the source.

Table II. Inference time comparison Fog-Cloud layer

Parameter	**Fog Node**		**Cloud Layer**	
	LSTM	*GRU*	*LSTM*	*GRU*
Inference Time [s]	0.886	0.912	2.655	2.389

Table III. Inference time comparison edge inference approaches

Parameter	**Our Smart-IoT-Gateway**		**Smartphone** [30]		
	LSTM	*GRU*	*SVM*	*k-NN*	*SOM*
Inference Time [s]	0.886	0.912	9	9	0.04

5.2. Fog Node Performance

The fog node evaluation is accomplished to obtain information about its performance. Since the fog node is a resource-constrained device, the CPU, memory, network, and power consumption parameters are evaluated for the proposed scenario. These parameters are measured using the methodologies proposed in [32], [33]. The measurements are taken in two different scenarios, without Docker engine and with it. Fig. 12 shows the comparison between both scenarios. The CPU percentage usage for no Docker scenario is higher than Docker scenario. In this case, the resource manager reduces the CPU usage by the Docker engine process. The CPU usage is directly proportional to power consumption, so the power consumption in the no Docker scenario is also higher than Docker scenario. The memory usage is higher in the Docker scenario due to the Docker engine and other Docker APIs occupy memory and MongoDB memory allocation. Also, the LSTM model uses less CPU than GRU model due to the LSTM cell does less complex operations. While the GRU model usage less memory due to the GRU cell has less model's

parameters. The Docker approach requires CPU usage and power consumption, but the memory usage is high. This affects the deployment of more large DL architectures that has more parameters such as complex CNN models.

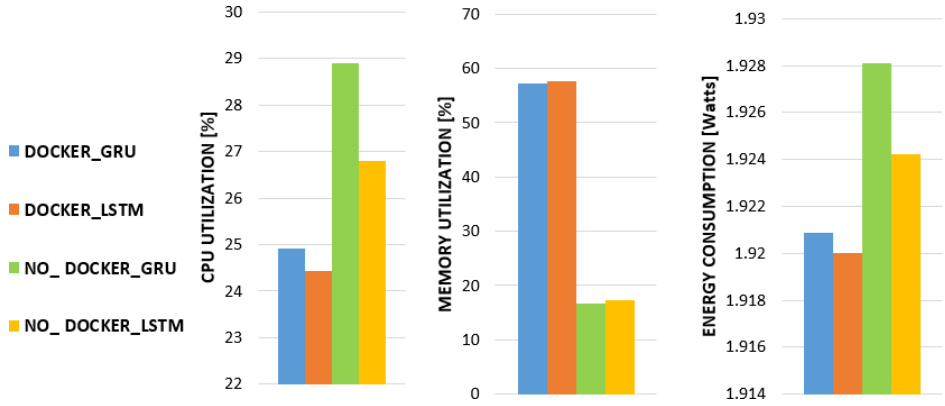

Fig. 12. Fog Node performance.

The delay in the network is the parameter improved by using this architecture. The delay is measured by sending packets from the IoT node to the fog node and the cloud layer, using the iPerf3 tool. The result obtained was a delay of 160ms between the IoT node and the cloud layer, and a latency of 13ms between the IoT node and the fog node. The packets are sent through the internet across multiple routers until reaching the cloud server, and this increases the network latency. As a result, the delay between the IoT nodes to the cloud layer is much higher than the delay between the IoT node and the fog node. It has also been reflected in the total inference time of the model from the cloud shown in Table II which is increased due to this latency. Deploying the model in the fog node reduces the total inference time because the packets have one hop reducing the network.

5.3. Big Data Performance

This evaluation is focused on the performance of the big data platforms during the dataset pre-processing. This stage is necessary for preparing the training dataset to detect health anomalies using DL techniques. We use the dataset described in the cases studies to evaluate the performance. Since the dataset does not oversize 752.7 MB, some text files were duplicate until reaching 2 GB size. This data augmentation allows determining the performance of the big data system in case of dataset growth. The new dataset is split into various batch sizes (64, 128, 256, 512, 1024, 2048 MB). The throughput is calculated by dividing the batch size by the time to take processing it. Fig. 13 shows the throughput of the big data batch processing module. The performance increases despite the dataset growth. This behavior is expected since Apache Spark uses a parallel processing by using the RDD abstraction, and it guarantees dataset scalability in terms of processing performance. Thus, if the dataset increases, the system would

keep a good performance during pre-processing data steps instead of decreases its performance due to the increment of data.

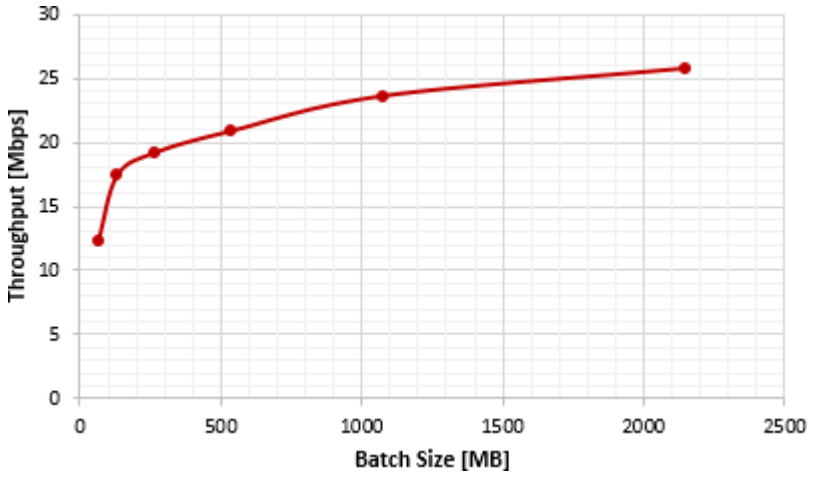

Fig. 13. Performance of big data processing platform

6. CONCLUSION

In this paper, we present an innovative intelligent system to detect falls based on a proposed fog-cloud 3-layer architecture to deploy DL models into fog resource-constrained devices. In this way, two important improvements are achieved. First, the use of a DL model for the detection of falls allows a higher accuracy and efficiency than traditional ML models, due to the high potential of DL for generating complex models and the ability of automatic extraction of features. In this sense, our system has higher efficiency than other DL approaches as far as achieves high accuracy with fewer parameters. Second, our system employs a fog approach for the fall detection process (DL inference), instead of placing the inference process on the cloud. This has the advantages of virtually no network latency, nor extra network congestion delay. Thus, a more timely response is achieved as far as the monitoring data does not need to be transported to the cloud. Few studies based on DL models to detect falls affirm to employ predictive analytics on resource-constrained devices but they did not provide information about the deployment nor conducted a practical and experimental deployment of DL models. To our best knowledge, this is the first documented approach for fall detection on fog devices using DL. Also, it is the first regarding DL on the fog, as this gap of documentation in the literature happens on all practical use cases of DL on fog nodes in real environments. As far as fog devices, in special IoT gateways, are resource-constrained and DL inference process requires considerable resources (higher than ML in general), this approach requires to solve a complex challenge of a very highly-efficient optimization of resources and also a proper choice of

the algorithm and the DL model. To overcome this challenge, we employ a lightweight virtualization architecture capable of supporting multiple application and services. We solved the problem through the use of virtualization technologies that allow a very tight optimization of resource usage: this is a novel solution not employed before in DL on the fog.

Our system employs a wearable-approach in combination with the use of an IoT gateway as a fog node to collect monitoring data of the subject. Wearable sensors present significant advantages for monitoring falls in comparison to other detection approaches. On the other hand, the use of a smart gateway as a fog device has important advantages over a smartphone choice (the other main option), such as non-dependence on battery life, on the commitment of the elderly subject on carrying the device and lower intrusiveness. Also, it is appropriate for seamlessly covering indoor environments -where undetected falls mostly happen. If a fall is detected an alert is immediately sent to caregivers' smartphone through our app, which is an alert service more appropriate than others widely used that require a confirmation through a wearable button, that may not be pushed by the affected person. Regarding the choice of the methodology to detect the fall, we decided to employ a DL approach to benefit from DL advantages. Our choice was an RNN algorithm that is appropriate for sequential data such as IoT monitoring data. We created two DL models based on RNN (LSTM/GRU) that fulfill the resource constraint requirements and provide very high accuracy (98.75% in one model). Both models provide better accuracy and a better trade-off of sensitivity and specificity parameters than the other wearable-based systems.

We demonstrate that due to the placement of the DL model inference in the fog node, the total inference time is reduced almost 2 seconds. Also, we demonstrate that using virtualization technology (specifically with Docker engine), fog node resources (i.e. processing and power consumption) are very efficiently managed, achieving better performance. As a counterpart, the memory occupancy is slightly higher using our virtualization strategy due to MongoDB memory allocation process and Docker unnecessary APIs activation.

Also, we employed Big Data analytics in the cloud layer to manage the amount of data generated by the IoT nodes to create the model to be implemented in the fog, and later on to improve it taking advantage of the fog inference process results, without affecting the ongoing inference process. We demonstrate that the use of Big Data analytics platforms Apache Spark and HDFS improve the performance of data pre-processing previous to the training of DL models, and allow to save the data in a redundant and reliable way. This is relevant in the case a new DL model is trained with more data, as more amount of data must not affect the performance of the pre-processing. Finally, we implemented a mechanism to remotely manage and deploy applications and DL models on fog nodes. In this way, the

application deployment in the fog nodes is automatic, remote and optimally simplified. Thus, it spares the need to be physically placed where the fog node is located in order to configure it and deploy applications in-situ.

In short, the proposed system not only covers the existing gap in literature regarding the deployment of DL models in resource-constrained devices, but also it improves the current ML and DL fall detection models performance, explains and details the deployment of DL models on resource-constrained devices, provides a solution to the complex challenge of management of fog limited resources for supporting multiple services and applications by means of a virtualization strategy, and provides a mechanism that allows a remote automatic deployment and management of models on fog nodes. Results prove the effectiveness of our fall detection service that surpass in accuracy, response time, and other aspects current ML fall detector approaches.

In the future, we will use the system in a nursery in where more smart IoT gateways can connect to each other, augmenting the service area to a whole building. Also, we will optimize the memory consumption which is currently high due to the short-term module. Since our proposed architecture and deployment mechanism is flexible and can be adapted to other health applications, we will deploy other health services using DL models such as arrhythmia detection, apnea detection, among others.

REFERENCES

[1] "World Population Ageing." [Online]. Available: http://www.un.org/esa/population/publications/worldageing19502050/. [Accessed: 23-Sep-2018].

[2] "Falls," *World Health Organization*. [Online]. Available: http://www.who.int/news-room/fact-sheets/detail/falls. [Accessed: 20-Sep-2018].

[3] P. Rashidi and A. Mihailidis, "A Survey on Ambient-Assisted Living Tools for Older Adults," *IEEE J. Biomed. Heal. Informatics*, vol. 17, no. 3, pp. 579–590, May 2013.

[4] J. Bousquet *et al.*, "Operative definition of active and healthy ageing (AHA): Meeting report. Montpellier October 20-21, 2014," *Eur. Geriatr. Med.*, vol. 6, no. 2, pp. 196–200, 2015.

[5] "WHO | What is Healthy Ageing?" [Online]. Available: http://www.who.int/ageing/healthy-ageing/en/. [Accessed: 19-Sep-2018].

[6] X. Yu, "Approaches and principles of fall detection for elderly and patient," in *2008 10th IEEE Intl. Conf. on e-Health Networking, Applications and Service, HEALTHCOM 2008*, 2008, pp. 42–47.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*, 2012, p. 13.

[8] X. Fei *et al.*, "CPS data streams analytics based on machine learning for Cloud and Fog Computing: A survey," *Futur. Gener. Comput. Syst.*, vol. 90, pp. 435–450, 2019.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT PRESS, 2016.

[10] W. Zaremba, "Recurrent Neural Network Regularization," no. 2013, pp. 1–8, 2015.

[11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," pp. 1–9, 2014.

[13] J. Fleming and C. Brayne, "Inability to get up after falling, subsequent time on floor, and summoning help: Prospective cohort study in people over 90," *Bmj*, vol. 337, no. 7681, pp. 1279–1282, 2008.

[14] N. Zerrouki, F. Harrou, Y. Sun, and A. Houacine, “Vision-Based Human Action Classification,” vol. 18, no. 12, pp. 5115–5121, 2018.
[15] L. Panahi and V. Ghods, “Human fall detection using machine vision techniques on RGB–D images,” *Biomed. Signal Process. Control*, vol. 44, pp. 146–153, 2018.
[16] N. Lu, Y. Wu, L. Feng, and J. Song, “Deep Learning for Fall Detection: 3D-CNN Combined with LSTM on Video Kinematic Data,” *IEEE J. Biomed. Heal. Informatics*, vol. 2194, no. c, 2018.
[17] Y. Li, K. C. Ho, and M. Popescu, “A Microphone Array System for Automatic Fall Detection,” vol. 59, no. 2, pp. 1291–1301, 2012.
[18] C. Taramasco *et al.*, “A novel monitoring system for fall detection in older people,” *IEEE Access*, vol. 6, pp. 43563–43574, 2018.
[19] X. Fan, H. Z. B, C. Leung, and Z. Shen, *Fall Detection with Unobtrusive Infrared Array Sensors*, vol. 501. Cham: Springer International Publishing, 2018.
[20] C. Wang *et al.*, “Low-Power Fall Detector Using Triaxial Accelerometry and Barometric Pressure Sensing,” vol. 12, no. 6, pp. 2302–2311, 2016.
[21] S. B. Khojasteh and E. De Cal, “Improving Fall Detection Using an On-Wrist Wearable Accelerometer,” pp. 1–28.
[22] T. Theodoridis, V. Solachidis, N. Vretos, and P. Daras, “Human fall detection from acceleration measurements using a recurrent neural network,” in *IFMBE Proceedings*, 2018, vol. 66, pp. 145–149.
[23] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, “SisFall: A fall and movement dataset,” *Sensors (Switzerland)*, vol. 17, no. 1, 2017.
[24] F. Sposaro and G. Tyson, “iFall : An Android Application for Fall Monitoring and Response,” pp. 6119–6122, 2009.
[25] A. Ngu, Y. Wu, H. Zare, A. P. B, B. Yarbrough, and L. Yao, “Fall Detection Using Smartwatch Sensor Data with Accessor Architecture,” vol. 2, pp. 81–93.
[26] P. Jantaraprim and P. Phukpattaranont, “Fall Detection for the Elderly using a Support Vector Machine,” no. 1, pp. 484–490, 2012.
[27] 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,” *Med. Biol. Eng. Comput.*, vol. 55, no. 1, pp. 45–55, 2017.
[28] M. D. Solbach and J. K. Tsotsos, “Vision-Based Fallen Person Detection for the Elderly,” *Proc. - 2017 IEEE Int. Conf. Comput. Vis. Work. ICCVW 2017*, vol. 2018–Janua, pp. 1433–1442, 2018.
[29] X. Li, T. Pang, W. Liu, and T. Wang, “Fall detection for elderly person care using convolutional neural networks,” *Proc. - 2017 10th Int. Congr. Image Signal Process. Biomed. Eng. Informatics, CISP-BMEI 2017*, vol. 2018–Janua, pp. 1–6, 2018.
[30] V. Carletti, A. Greco, A. Saggese, and M. Vento, “A Smartphone-Based System for Detecting Falls Using Anomaly Detection,” vol. 6978, 2017, pp. 490–499.
[31] D. Yacchirema, J. S. De Puga, C. Palau, and M. Esteve, “Fall detection system for elderly people using IoT and Big Data,” *Procedia Comput. Sci.*, vol. 130, pp. 603–610, 2018.
[32] D. Sarabia-Jacome, A. Rego, S. Sendra, and J. Lloret, “Energy consumption in software defined networks to provide service for mobile users,” *2017 13th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2017*, 2017.
[33] F. Kaup, P. Gottschling, and D. Hausheer, “PowerPi: Measuring and modeling the power consumption of the Raspberry Pi,” *39th Annu. IEEE Conf. Local Comput. Networks*, pp. 236–243, 2014.