Ensemble Deep Learning on Wearables Using Small Datasets

TAYLOR MAULDIN, ANNE H. NGU, and VANGELIS METSIS, Texas State University
MARC E. CANBY, University of Illinois at Urbana-Champaign

This article presents an in-depth experimental study of Ensemble Deep Learning techniques on small datasets for the analysis of time-series data generated by wearable devices. Deep Learning networks generally require large datasets for training. In some health care applications, such as the real-time smartwatch-based fall detection, there are no publicly available, large, annotated datasets that can be used for training, due to the nature of the problem (i.e., a fall is not a common event). We conducted a series of offline experiments using two different datasets of simulated falls for training various ensemble models. Our offline experimental results show that an ensemble of Recurrent Neural Network (RNN) models, combined by the stacking ensemble technique, outperforms a single RNN model trained on the same data samples. Nonetheless, fall detection models trained on simulated falls and activities of daily living performed by test subjects in a controlled environment, suffer from low precision due to high false-positive rates. In this work, through a set of real-world experiments, we demonstrate that the low precision can be mitigated via the collection of false-positive feedback by the end-users. The final Ensemble RNN model, after re-training with real-world user archived data and feedback, achieved a significantly higher precision without reducing much of the recall in a real-world setting.

CCS Concepts: • **Computing methodologies** → **Ensemble methods**; **Neural networks**; • **Human-centered computing** → *Mobile devices*;

Additional Key Words and Phrases: Ensemble methods, deep learning, recurrent neural network, fall detection, time series, wearable, smart health, IoT

ACM Reference format:
Taylor Mauldin, Anne H. Ngu, Vangelis Metsis, and Marc E. Canby. 2020. Ensemble Deep Learning on Wearables Using Small Datasets. *ACM Trans. Comput. Healthcare* 2, 1, Article 5 (December 2020), 30 pages.
https://doi.org/10.1145/3428666

1 INTRODUCTION

Internet of Things (IoT) is a domain that represents the next most exciting technological revolution since the Internet. In the healthcare domain, IoT promises to bring personalized health tracking and monitoring ever closer to consumers. This phenomenon is evidenced in Wall Street Journal articles titled "Staying Connected is Crucial

We thank the National Science Foundation for funding the research under the Research Experiences for Undergraduates site Programs No. CCF-1659807 and No. CNS-1757893, at Texas State University to perform this piece of work, and the infrastructure provided by NSF-CRI Award No. 1305302.
Authors' addresses: T. Mauldin, A. H. Ngu, and V. Metsis, Department of Computer Science, Texas State University, 601 University Drive, San Marcos, Texas, 78666; emails: {trm119, angu, vmetsis}@txstate.edu; M. E. Canby, Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N Goodwin Ave, Urbana, Illinois, 61801; email: marcec2@illinois.edu.

2637-8051/2020/12-ART5 $15.00
https://doi.org/10.1145/3428666

to Staying Healthy" (WSJ, June 29, 2015) and "Digital Cures For Senior Loneliness" (WSJ, Feb. 23, 2019). Modern smartphones and many wearable devices now contain more sensors than ever before. Data from those sensors can be collected more easily and more accurately. In 2014, it is estimated that 46 million people are using IoT-based health and fitness applications. Currently, the predominant IoT-based health applications are in sports and fitness. However, disease management or preventive care applications are becoming more prevalent. The urgency for investment in health monitoring IoT technology is also echoed by another Wall Street Journal article (July 21, 2018) titled "United States is Running Out of CareGivers." By 2020, there will be 56 million people aged 65 and above as compared with 40 million in 2010. In Reference [1], a system called VitalRadio is reported to be able to monitor health metrics such as breathing, heart rate, walking patterns, gait, and emotional state of a person from a distance.

Recently, there has been a surge in the number of real-time preventive care applications such as those detecting falls in elderly patients due to the increasing aging population [28]. Wearable devices, especially smartwatches that pair with smartphones, are increasingly a platform of choice for deploying digital health applications. This is due to the fact that a smartwatch has the benefit of being unobtrusive and comfortable to wear, as it can be seen as the same as wearing a piece of jewelry. The popularity of using a smartwatch paired with a smartphone as a viable platform for deploying digital health applications is further supported by the release of Apple Series 4 smartwatch [2], which has a built-in "hard fall" detection application as well as an ECG monitoring App. Recently, an Android-Wear-based commercial fall detection application called RightMinder [22] was released on Google Play. The number of digital health applications using IoT devices is going to continue to increase in the next few years.

Deep learning (DL) has demonstrated outstanding performance in computer vision, speech recognition and natural language processing applications. Our earlier work [18] compared traditional machine learning techniques (SVM, Naive Bayes) with deep learning, in particular the Recurrent Neural Network (RNN), for fall detection using only acceleration data captured through a wrist-worn watch, and concluded that DL shows superior fall detection performance. We demonstrated the superiority of DL through extensive experiments using three different fall datasets as well as real-world testing with five subjects. A single deep model was trained and the standard parameter tuning and training procedures were followed in our earlier work. Moreover, the accelerometer data, which are a form of time-series data, were processed using a fixed-size sliding window approach. As pointed out by Reference [10], there are a few inherent challenges in applying DL to wearable devices. For example, the collected data could be noisy due to faulty sensor readings. The way data are assigned for training may be limiting as well. For example, if the chosen window size is too small, important signals might fall outside the range; having the window-size too large risks having to process useless input data that is not relevant to a fall. Most importantly, a large training dataset for many wearable health-related applications is almost impossible to obtain, as in the case of fall dataset, since fall is not a common event.

The main focus of this article is on the offline and real-world experimentation and analysis of applying ensemble techniques on deep learning such that it is possible to mitigate the scarcity of data as well as improve the accuracy of prediction via diverse set of learners. The SmartFall fall detection application we reported previously, which was trained on a small dataset of accelerometer data from a smartwatch using RNN, achieved only 86% accuracy, based on our real-world test with five subjects [18], due to the occurrence of high number of false positives, which reduced the precision of the detector.

In this work, we investigate the idea of combining popular ensemble techniques such as Stacking and Boosting [32] with RNN to create an ensemble of diverse learners to mitigate small dataset problem. The rationale of our approach is that training a set of smaller deep learning models and then combining them using an ensemble approach, may perform better than training a single complex deep learning model, which would require large amounts of training data.

In Section 3.3, we discuss our proposed ensemble RNN scheme and present our findings. We validate the generated ensemble models using two datasets. The first dataset was collected in house using the Microsoft Band 2 smartwatch from 14 subjects ranging from 20 to 60 years old. The second dataset is the publicly available

UniMiB database [19] that contains falls and Activities of Daily Living (ADLs) collected using a smartphone from 30 different subjects with ages ranging from 18 to 60.

Our offline experimental results show that long short-term memory (LSTM) RNN models achieve superior performance compared to other DL models, such as one-dimensional convolutional neural networks (1D-CNN) for detecting falls. Furthermore, an ensemble of RNN models outperforms a single RNN model trained on the two datasets. In some situations, we observe that training RNN models on subsets of data, e.g., particular types of falls, and then combining them using an ensemble approach generates better accuracy than training all RNN models on the whole dataset.

For the sake of completeness, we also experimented with traditional ensemble algorithms such as Random Forest and Gradient Boosting classifiers on the same task. These algorithms showed significantly lower performance compared to deep learning-based approaches.

A main obstacle to developing highly accurate models for the task of fall detection is that it is difficult to collect a large number of labelled fall data samples and to anticipate all possible everyday movements that can trigger false-positive alerts, during the data generation process in a lab setting. Instead, it is much simpler to ask a user to wear the watch for a period of time, provide feedback on each generated false positive, and use those samples to re-train the model. In Section 6, we present the design of a real-world experiment involving three subjects, with each subject wearing the watch and using the SmartFall App over a period of time to provide feedback on wrongly classified falls and ADLs.

As documented in Reference [5], there are at least 57 projects that used wearable devices to detect falls in elderly. However, only 7.1% of the projects reported testing their models in real-world setting. Many papers in the literature provide very accurate fall detection results both in terms of recall and precision via offline experiments, but those accuracies are not verified in real-world settings. Our real-world experiments show that models trained with simulated fall and ADL data collected in a lab setting can generate a high volume of false positives in the real-world setting. However, this can be mitigated by collecting and marking those false-positive data samples to re-train the original model. While it is difficult to collect fall data in the real world, it is relatively easy to collect large amounts of ADL data.

In summary, the main contributions of this article are:

- An in-depth study of ensemble techniques combined with deep learning for fall detection on two different fall datasets.
- A set of offline experiments that explore the effect of data heterogeneity on classification performance and the extent to which ensemble deep learners can diversify to better capture that heterogeneity, thus yielding higher classification accuracy compared to a single deep learning model.
- An Android-based SmartFall App that can detect falls using ensemble deep learning model based on only accelerometer data from the wrist-worn watch and that allows users to label falls and non-fall events in real-time.
- The design of real-world experiments that demonstrate the significant reduction of false positives via re-training with user's labeled false-positive samples.
- A new metric based on spikes of accelerometer data for evaluating the precision of the fall detection model independent of user's level of activities in real-world experiments.

The remainder of this article is organized as follows. In Section 2, we review the existing work on small sample deep learning and emphasize on research works that specifically address fall detection using wearable devices. In Section 3, we provide a detailed description of our approach to ensemble deep learning. In Section 4, we present various metrics and the systematic methodology we used for evaluating the quality of a trained model. Then in Section 5, we present our offline experimental results and detailed discussion of those results. In Section 6, we describe the SmartFall App we used for the real-world experiments, present the spikes of accelerometer metric we used for evaluation, and the observations we derived from the real-world experimental results. Finally, In Section 7, we present our conclusion and future work.

2 RELATED WORK

Our literature review focuses on two categories of research work that are relevant to our problem. The first category is related to small sample deep learning, and the second category is deep learning methods applied to streaming time series data for fall detection or human activity recognition.

Deep Learning for Human Activity Recognition (HAR) has shown superior results compared to traditional machine learning as it achieves high classification accuracy on raw data, and eliminates the need for human crafted features [30]. However, deep neural networks face practical performance limitations when employed in IoT applications: imbalanced datasets, small samples, and data quality can significantly degrade the performance of a HAR classifier. Training a deep neural network requires a significant number of iterations before the optimal values of millions of parameters are found. If the network is trained using a small dataset, then a large number of iterations can result in over-fitting. One way to overcome this real-world limitations is to combine sets of diverse LSTM learners into an ensemble of classifiers as shown in Reference [11]. Their method assumes that certain portion of the training data is "problematic." This means there are some low quality data that can negatively impact the performance of the classifier. Since there is no way to tell which section of the input data is problematic in a-priori, a probabilistic selection of subset of data that resembles Bagging is employed. Through repeated random selection of training data, Bagging bootstrapped replicates of good quality training samples. We drew our inspiration from them and focusing on capturing the diversity of data during training and on optimal way to aggregate the different predictions to achieve a meta-classifier with good generalization properties.

A recent approach to solving the small sample learning problem in IoT is shown by the cost-sensitive deep active learning proposed in Reference [6]. Deep active learning interactively requests a portion of the most informative instances to be labelled during the classifier training phase by engaging a human in the loop. The most informative instances are those instances that the classifier is least certain of. The system has a generic double deep neural network (DNN), which can be configured with any variant of deep models (CNN, LSTM-RNN, GRU-RNN). The architecture consists of the primary DNN and an assistant DNN. The role of the primary DNN is to predict the result and the assistant DNN is to predict the cost of misclassification of each sample in the unlabelled data pool. Their experimental results demonstrated that the proposed scheme can reduce the amount of the required labelled samples by 33% to 80%. This approach assumes that there is an abundance of unlabelled data that is not true in many IoT applications such as the fall detection.

Another popular technique for overcoming small sample dataset for deep learning is via data augmentation. This technique is commonly used in the field of computer vision in the form of rotation, translation and scaling of the original images. However, in the digital health domain, transformations of sequential sensor data have not been widely adopted [21] due to the fact that a small distortion in medical signals might imply a huge change in medical condition. Recently, works on data augmentation for training more accurate fall detection models on two datasets provided in our earlier work was found in Reference [24]. However, there is no real-world validation on the generated fall models. Other approaches for overcoming small dataset problem include attempts to combine convolutional and recurrent layers, where convolutional layers act as feature extractors and provide abstract representations of the input sensor data in feature maps, and the recurrent layers model the temporal dynamics of the activation of the feature maps [26, 31].

Application of deep learning to fall detection, in particular the use of recurrent neural networks (RNNs) to detect falls has been attempted by researchers; however, to our knowledge, no such work uses solely accelerometer data collected by a smartwatch to detect falls. In Reference [29], the authors describe an RNN architecture in which accelerometer signal is fed into two LSTM layers, and the output of these layers is passed through two feed-forward neural networks. The second of these networks produces a probability that a fall has occurred. The model is trained and evaluated on the URFD dataset [16], which contains accelerometer data taken from a sensor placed on the pelvis, and produces a 95.71% accuracy. The authors also describe a method to obtain additional training data by performing random rotations on the acceleration signal; training a model with this data gives an accuracy of 98.57%.

Another system based on RNN for fall detection using accelerometer data is proposed in Reference [20]. The core of their neural network architecture consists of a fully connected layer, which processes the raw data, followed by two LSTM layers, and ending with another fully connected layer. They also have some normalization and dropout layers in their architecture to optimize the training. The authors train and test their model with the SisFall dataset [27], which contains accelerometer data sampled at 200 Hz collected from a sensor attached to the belt buckle. To deal with a large imbalance in training data, of which ADLs form the vast majority, the authors define a weighted-cross entropy loss function, based on the frequency of each class in the dataset, that they use to train their model. In the end, their model attains a 97.16% accuracy on falls and a 94.14% accuracy on ADLs.

Our work differs primarily from these two papers that utilize RNN in that we seek to optimize deep learning models for time series data collected from a smartwatch. Our fall detection model obtains accelerometer data from an off the shelf smartwatch rather than specialized equipment placed near the center of the body. This presents several challenges not addressed in these papers' methodology. Because of its placement on the wrist, a smartwatch will naturally show more fluctuation in its measurements than a sensor placed on the pelvis or belt buckle. Moreover, the accelerometer data used in Reference [27] is sampled at a 200 Hz frequency obtained by a specialized equipment; this is a significantly higher than the frequency used by commodity off-the-shelf smartwatch, which samples at 30–50 Hz. We also have the additional restriction that the model we develop should not consume so many computational resources that it cannot be run on a smartphone. Thus, while there has been some work done on deep learning for fall detection, we have additional constraints that make these works not directly relevant for our purposes.

In Reference [25], a fall detection system architecture using multiple sensors with four traditional machine learning algorithms (SVM, Naive Bayes, Decision tree, and KNN) was studied. The paper is the first to propose using ANOVA analysis to evaluate the statistical significant of differences observed by varying the number of sensors and the choice of a particular machine learning algorithm for time series data. The main conclusion from this paper is that sensors placed close to the gravity center of the human body (i.e., chest and waist) are the most effective. A similar paper in Reference [34] also conducted a study on the effect of the sensor location on the accuracy of fall detection. They experimented with six different traditional machine learning algorithms including dynamic time warping and artificial neural network. They showed that 99.96% sensitivity can be achieved with a waist sensor location using the KNN algorithm. Our work is focused on using a wrist-worn watch as the only sensor and thus cannot leverage these research results on other sensor locations to improve the accuracy.

In summary, many different deep learning algorithms have been applied to time series data collected from wearable devices with some success. However, no in-depth study in a real-world setting has been conducted on how robust models for time series data can be trained with deep networks using small dataset collected from smartwatches with low sampling frequency.

3 METHODOLOGY

3.1 Datasets

We used two publicly available datasets, that contain simulated falls and ADLs, to evaluate our methods. A brief description of each dataset is given here.

3.1.1 SmartFall Dataset. This section describes how the fall dataset[1] that we used for experiments was collected using a smartwatch. A detailed description is available in our earlier publication [18]. We reproduce some of the description here for ease of reference and comprehension.

Our smartwatch-based fall dataset was collected from 14 volunteers, each wearing a Microsoft Band 2 watch.[2] These 14 subjects were all of good health and were recruited to perform simulated falls and activities of daily living (ADLs). Their ages ranged from 21 to 60, height ranged from 5 to 6.5 ft. and the weight from 100 to

[1]This dataset is available from http://www.cs.txstate.edu/~hn12/data/SmartFallDataSet under the smartFall folder.
[2]Unfortunately, Microsoft announced stopping the support for this product on May 31, 2019.

Fig. 1. Simulated fall data collection experiment.

230 lbs. Each subject was told to wear the smartwatch on their left wrist and performed a pre-determined set of ADLs consisting of: jogging, sitting down, walking, and waving their hands. This initial set of ADLs was chosen based on the fact there are common activities that involved movement of the wrist with varying intensity of acceleration. These data samples were automatically labeled as "ADL," which is in our case we consider as "NonFall." We then asked the same subject to perform four types of falls onto a 12-inch-high mattress on the floor; front, back, left, and right side falls. Each subject repeated each type of fall 10 times. The sampling rate was set to 31.25 Hz, which is the medium sampling rate supported by MS Band watch.

We implemented a data collection service on a smartphone (Nexus 5X, 1.8 GHz, Hexa-core processor with 2G of RAM) that paired with the smartwatch to have a button that, when pressed, labels data as "Fall" and otherwise "NonFall." Data was thus labelled in real-time as it was collected by the researcher holding the smartphone. Figure 1 shows the scene of a fall data collection experiment. After manually inspecting the accelerometer signals and the marked falls for errors, a total of 528 labeled falls were left in the dataset. The average duration of a fall, as marked by the researchers through the smartphone application, was approximately 0.8 s, which corresponds to 25 data points given our sampling rate of 31.25 Hz. For consistency, the collected data were post-processed, and all falls were marked to be 25 points around the peak acceleration value for that particular fall. The rest of the signal was assumed to be ADL data. Segmenting the signal into segments of the same size (25 data points) as the fall segments, we generated a total of 6,572 ADL segments. During our training process, however, we fed the learning algorithms with sequences of 35 data points with overlap, to encompass some data before and after a fall.

3.1.2 UniMiB SHAR Dataset. The second dataset that we used is the UniMiB SHAR [19]. This is a publicly available dataset that has 11,771 data points of accelerometer data collected from smartphones (Samsung Galaxy Nexus 19250) for both ADLs and falls performed by 30 subjects of ages ranging from 18 to 60 years. The subjects are mostly female. The data was collected at a sampling rate of 50 Hz. The user put phones in the left and right pockets of trousers and handclaps are used to signal the beginning and ending of fall. The ADLs data is divided into nine different types and the falls are divided into eight types resulting in a total of 17 different classes. There are 4192 data points marked as falls and 7579 data points of ADLs in total in this dataset. We processed this dataset and have all the nine type of ADLs labeled as "NonFall." We only retain the four fall types that we used in our smartwatch fall dataset and omitted other special fall types such as falls derived from hitting an obstacle, syncope, falling with protection strategies, and falling back while trying to sit on a chair. To keep this dataset comparable with our smaller sample SmartFall dataset, we omit some duplicate activities performed by the same user. This reduces the size of the dataset while maintaining its diversity. The final number of falls is 710 and ADLs is 486.[3]

[3]The post-processed UniMiB dataset is available from http://www.cs.txstate.edu/~hn12/data/UniMiBProcessed.zip.

3.2 Deep Learning Model

One of the disadvantages of traditional machine learning algorithms is the need for a priori feature extraction from the data. Feature extraction and selection are tasks that need to be performed before any learning can occur. The types of features to be extracted have to be manually specified and thus their effectiveness heavily depends on the ingenuity of the researcher. In the time series domain, each signal has different temporal and frequency domain characteristics [3]. This makes feature extraction and selection a complicated task, which can heavily affect the performance of the machine learning model. For example, in Reference [13] the accuracy of the SVM algorithm varies depending on the feature selection method used. In feature-dependent methods, the main difficulty is to extract the appropriate features. In certain types of data, to extract high-quality features we need human-like understanding of the raw data.

Deep learning comes to solve this problem by eliminating the need for separate feature extraction, selection and model training phases. Deep learning refers to the process of machine learning using deep neural networks. Deep learning has shown significant improvements in areas such as image classification and object detection. In early object detection approaches, people extracted features and fed these features to learning algorithms (e.g., SVM) to successfully detect objects of interest (e.g., pedestrians) in the image. However, when these methods were used to detect several classes other than pedestrians, e.g., car, sign or tracks, the accuracy of the model dropped [33].

The use of deep convolutional neural networks showed a notable increase in the performance of detecting objects using highly challenging datasets [15]. The two most common implementations of deep neural networks are Convolutional Neural Networks (CNN) [15] and Recurrent Neural Networks (RNN) [9]. CNN is a type of feed-forward artificial neural network that takes fixed size inputs and generates fixed-size outputs. CNNs are ideal for images and video processing. One-dimensional variants of CNN (1D-CNN) have also been utilized for time series data such as human activity recognition and fall detection [17, 24].

RNNs, unlike feed-forward neural networks, can use their internal memory to process arbitrary sequences of inputs and the ability to capture the temporal dynamics of the input data. RNNs are thus ideal for the analysis of the sequential nature of the data points collected from accelerometers in our fall detection task. A popular variant of the traditional RNN contains units called long short-term memory (LSTM). This architecture helps to capture fall motion related activities over a period of time so that they can be better distinguished from others. We believe it has an advantage over threshold-based algorithms when making a single classification. Many regular activities can briefly trigger high acceleration values, and threshold-based models often have a hard time telling these apart from falls. Looking at many data points at once allows us to make more robust distinctions between activities. Figure 2 displays our model architecture:

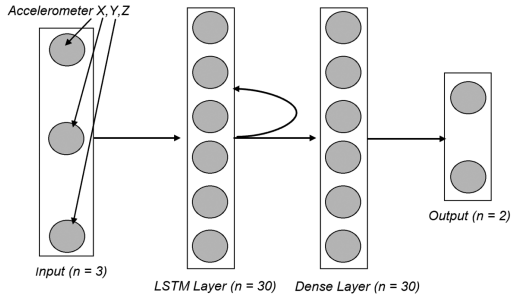

Fig. 2. RNN model architecture.

We experimented with both 1D-CNN and LSTM-based models and compared their performance on the fall detection task. In the following paragraphs, we describe the network architecture of each model employed. As it is evident from our experimental results shown in Figure 5, LSTM outperformed 1D-CNN on the fall detection

task with SmartFall dataset, and thus our subsequent exploration of ensemble deep learning models uses LSTM as the basic building block.

The 1D-CNN model architecture contains an input layer, two 1D-convolutional layers of 64 filters each and a kernel of size 3, a max pooling layer of size 2, and a fully connected dense layer of 30 neurons before reaching the softmax binary output layer. Dropout is used on the second convolutional layer, and the final dense layer. The dropout rate was configured to be 10% of the neurons in both cases. We found this architecture of the 1D-CNN to be the most effective, and it is similar to other implementations found in the literature for the problem domain of human activity recognition from accelerometer data.

The LSTM model contains an input layer, two hidden layers, and an output layer (see Figure 2, which depicts the netqork architecture we used). The input layer contains 3 nodes for the raw data; the accelerometer x, y, z vectors. It then feeds through our hidden layers: a recurrent layer of size 30 LSTM nodes, and a fully connected dense layer of size 30 nodes. A dropout rate of 10% is used after the LSTM and after the dense layer to reduce over-fitting. The output is a 2-node softmax layer that outputs a predicted probability that a fall has occurred. This model is lightweight relative to many deep learning architectures, and makes inference computation much more efficient for mobile devices. RNNs are traditionally trained with backpropagation through time (BPTT), so it is necessary to specify how many steps n in the past the network should be trained on. This parameter is important, since our falls and activities occur over a period of time. If we train the network on only a few steps in the past, then it will not capture the full scope of the activity. However, if we train it over too many steps, then the network may take into account past accelerometer data that are not relevant. We settled on using time 35 steps for this model. In our case, a "step" corresponds to a single acceleration data point. With a sampling frequency of 31.25 Hz, this means on each prediction the model takes into account ≈1.12 s of data. This is enough time to capture the aspects of a fall without including too much irrelevant data.

Model predictions (i.e., predictions produced by the neural architecture) begin once the number of sensor data points acquired is equal to the number of configured steps, which is the same as the prediction window size of 35 data points. Every model prediction thereafter will only require one additional data point, as the model will slide one data point at a time, reusing all of the previous data points except for the least recent. This means the amount of overlapping between a window and the actual set of samples is $K - 1$, where K is the size of the window. However, before producing a final prediction, we generate a heuristic value based on the probabilities produced by several consecutive model predictions. In essence, we compute the average value of 20 consecutive probabilities, and compare this with a pre-defined threshold value. If the average probability exceeds this threshold, then it is considered a fall prediction. This helps to avoid isolated positive model predictions from triggering a false positive. A window that only captures a partial (e.g., 1/3) sequence of fall data will have a lower probability of being classified as fall, because the average of the 20 predictions will be lower. However, a window that contains more than half of the actual data of a fall event can still be classified as "fall" depending on the threshold value we use. The parameter values of the optimal training window size (i.e 35) and output probability average (i.e., 20) were determined experimentally through grid search. Figure 3 outlines this schematic.

The reader should note that the heuristic applied here to predict fall events is not a universal solution used to classify time-series data. A different heuristic could produce different predictions and consequently have a different accuracy. However, by applying the same heuristic to the output of all our models, we can consistently compare their performance and decide which machine learning model is the most appropriate for the task at hand.

3.3 Ensemble Deep Learning

An ensemble is a system that combines multiple smaller models to create a more accurate system [32]. To do this, algorithms have been designed that deal with the way these smaller models contribute to the final decision. One such algorithm is AdaBoost, which optimizes the contribution of multiple weak classifiers to generate a more accurate strong classifier. Another popular ensemble technique, known as Bagging (short for Bootstrap

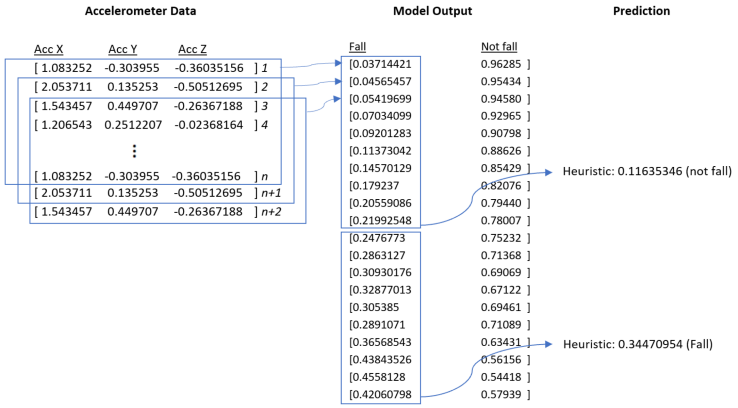

Fig. 3. Prediction scheme for deep learning. "Accelerometer Data" represents the raw input of sensor measurements. The continuous data stream is segmented into fixed size overlapping windows. "Model Output" represents the probabilities of "Fall" and "Not fall" produced by the deep learning model for each input window; "Predition" shows the final decision of our method by applying a heuristic to the sliding window of model output probabilities.

Aggregation) also utilizes a subset of the data in each training iteration. A probabilistic method is used to select a subset of the data so that the ensemble of models are trained on different portions of the training dataset. The resulting ensemble of models provides a final prediction on the data by voting. Finally, Stacking is a technique that resembles Bagging except an additional meta-classifier is trained at the end to combine predictions from the ensemble of classifiers without having any knowledge or effect on the data subsets on which the individual classifiers were trained.

The hypothesis that an ensemble of deep learning models can produce higher accuracy on small datasets than a single deep learning model stems from the fact that deep neural network architectures usually require large amounts of training data to produce accurate models. The more complex (more neurons and layers) a neural network is the more data is required to train it. However, very shallow neural networks, although more easily trained, cannot capture the full complexity of the problem (i.e., all possible signal patterns generated by different types of falls). However, an ensemble of diverse shallow networks may be able to perform better.

In this section, we discuss two ensemble deep learning techniques called Boosting and Stacking. Bagging was also considered, however, the combination of Bagging with RNNs did not produce good results. That is due to the fact that in Bagging, individual RNN models are trained using random subsets of the training data. RNN models trained in random subsets consistently displayed lower performance compared to models trained on the full training set, or subsets containing only particular types of falls in our experiments.

Our goal with ensemble deep learning was to mitigate the issue with false positives, while maintaining the overall high recall we achieved with a single model deep learning. We hypothesized that an ensemble of deep models could perform better than a single deep model. Our support for this comes from the lack of improvement when making model adjustments for a single deep model given our constraints of being unable to obtain large dataset for training. One such adjustment we made was dropout. Dropout is known to be useful for helping deep networks generalize. It removes neurons from the model at random during the training phase, which helps prevent overfitting the training data. However, our single deep model still suffered from sensitivity to false positives. This indicates an underlying pattern in how the models converge on the training data. Ensemble learning was our approach to mitigate this convergence pattern. By combining multiple deep models trained in different ways, each model had a chance to contribute something slightly different than the others.

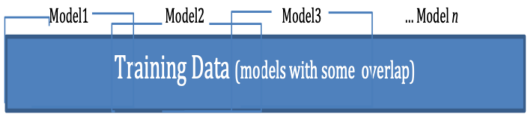

Fig. 4. Assignment of training data to each LSTM model with overlap in consecutive order

3.3.1 Boosted Deep Learning Models. Boosting is a well-known ensemble technique used for more than two decades. Boosting has traditionally been used with simpler weak learners such as shallow Decision Trees. We followed the standard AdaBoost implementation using LSTMs as weak learners. LSTM models are trained iteratively on weighted samples of the data. Data samples on which the models perform poorly on a given iteration are assigned a higher weight in future iterations. This process continues until a pre-determined number of weak classifiers is selected or the training accuracy does not improve anymore. As training proceeds, models (weak classifiers) are weighted based on their training error; higher weighted models contribute more to the output of the overall ensemble during the classification stage.

Since we are dealing with time series data, the samples required for AdaBoost need to take into account the sequential nature of the data. We thus segment the data into fixed-sized intervals, and each interval will be considered a sample marked as "Fall" or "NonFall." The intervals are computed from the data as follows: first, consecutive data points classified as "Fall" are coalesced into intervals (each has length 25). Then, the remaining data (the "NonFall" sections) are segmented into intervals of length 25 and marked as "NonFall." "NonFall" intervals that have length < 25 are ignored.

These intervals are considered the samples supplied to AdaBoost. It should also be noted that when AdaBoost feeds samples into the LSTM sub-models, it provides 400 data points prior to the sample, so that the LSTM has enough history to properly train on the sample.

3.3.2 Stacked Deep Learning Models. Our second ensemble method is a variant of Stacking [32]. This is an approach that takes the predictions of multiple models and trains an additional meta-classifier that learns how to combine these predictions using an Adam optimizer [14]. Our experiments with stacking consist of training multiple LSTMs separately, then training a list of weights that dictate how much each model's prediction contributes to the final prediction. The LSTMs were trained either on a subset of the training data, or on all the training data as described below.

Traditional ensemble methods such as Bagging and Boosting take random samples of the training data to assign to models. Instead, we just take consecutive sections of the data of equal size to assign to each model. This ensures every bit of the training dataset is used. In the case of human activity data, this has the advantage that the temporal nature of the data is retained. Figure 4 shows a schematic diagram of how data is assigned to each LSTM model. If there are 1,000 rows of data and 10 models with 10% overlap is specified, then each LSTM model will be fed with 110 rows of data.

The algorithm for assigning the data is detailed in Algorithm 1.

For our stacking meta-classification algorithm, a list of weights, one for each model, is initialized with random values. These weights determine how important each model's prediction is. To determine a final value for each model's weight, we adjust the weight list by repeatedly obtaining each model's prediction performance over the entire training set and and employing a gradient-based optimization process using Adam optimizer [14] to reduce the total error of the ensemble. Algorithm 2 lists the steps for calculating the final weights of each model.

At prediction time, the final output is calculated as a weighted linear combination of all LSTM models.

3.4 Classic Ensemble Models

Even though previous research has shown that classic machine learning approaches that require manual feature extraction demonstrate inferior performance compared to deep learning models [18], we experimented with two

ALGORITHM 1: Process for assigning data to models.

INPUT: number_of_models, model_overlap, list_of_models, and dataset_size
OUTPUT: subset of data assigned to different models

```
rows_per_model = rounded(dataset_size / ((1 - overlap) * (num_models - 1) + 1)) {We calcuate how many rows each model should have and assign the first model.}
models[0].start_row = 0
models[0].end_row = rows_per_model
{We then calculate the data to assign to subsequent models, except the last one.}
for i = 1 to num_models - 1 do
    models[i].start = floor(rows_per_model * (1-overlap)) * i
    model[i].end = models[i].start + rows_per_model
end for
{Data is assigned to the last model}
models[n-1].start_row = rounded(rows_per_model * (1-overlap)) * (num_models-1)
models[n-1].end_row = dataset_size
```

ALGORITHM 2: Algorithm for meta-classification.

INPUT: Trained LSTM models.
OUTPUT: Model weights.

1: Obtain all prediction outputs for every model over the entire training data.
2: Multiply the list of weights with the list of model outputs, then aggregate the result to get a final weighted output.
3: Compare this weighted output with the actual label of the data.
4: Use gradient descent to adjust the weight list so that the next time it is multiplied by the model outputs, the result will be closer to the actual label.
5: Repeat until error is no longer being improved.

popular ensemble algorithms for the sake of comparison. Specifically, we investigated Random Forest [7] and Gradient Boosting [8], implemented in the Python scikit-learn library. One hundred estimators were used in both algorithms. The fall prediction process is the same as the one used in the deep learning models, except that instead of feeding the raw data to the classifier, we convert each sequence window into a feature vector by manually extracting features from the sequence. The features extracted are the same as the ones previously proposed for SVM and Naive Bayes described in References [12, 18]. As shown in our experimental section (Figure 5), our results agree with previous findings. Deep learning models significantly outperform classic machine learning algorithms on the task of fall detection.

4 EVALUATION METHODOLOGY

We aim to train a fall detection model with a high Recall or Sensitivity. A missed fall is represented in our evaluation experiments as a False Negative (FN). We also do not want to have too many "false alarms," which in our evaluation is represented as False Positives (FP), and thus, we want to achieve a high Precision and Specificity. Table 1 shows the various metrics we used for evaluation and how they are computed. Note that True Positives (TP) is the number of correctly detected falls. The number of True Negatives (TN) is not of particular interest to this application as negative instances represent non-falls and, in practice, they greatly outnumber the number of positive instances.

Since we are dealing with time series data, evaluation of the model needs to account for the continuous nature of the data. Evaluation should also be independent of the prediction method, which allows models using different

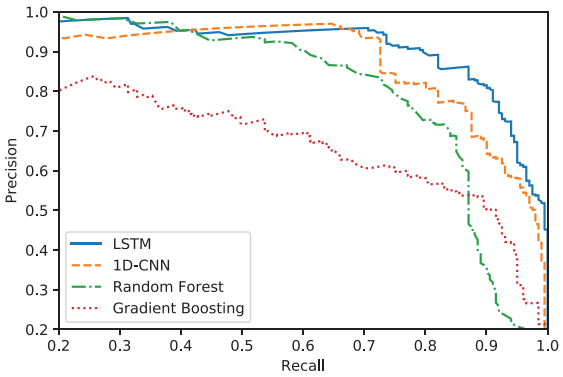

Fig. 5. PR curve comparing single models of different classification algorithms.

Table 1. Evaluation Metrics

Measure	Calculation formula
Recall/Sensitivity	$TP/(TP + FN)$
Precision	$TP/(TP + FP)$
Specificity	$TN/(TN + FP)$
Accuracy	$(TP + TN)/(TP + TN + FP + FN)$

TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative.

Table 2. Example Segments

Segment	Label
(0,24)	ADL
(24,49)	ADL
(71,95)	Fall
(96,120)	ADL

algorithms to be compared in a consistent way. Thus, the evaluation method we describe does not make assumptions about how the the prediction is accomplished, other than that it predicts "Fall" and "ADL" at arbitrary time points. Our evaluation method is described in details in the following paragraph.

The test data is segmented into "Fall" and "ADL" segments of equal size of 25 data points. This corresponds to the duration of a typical fall, which is around 0.8 s. How the data is segmented is described in the Boosting section 3.3.1 above. Each of these segments will be considered a data sample (instance) when calculating the error metrics above (i.e., each segment will be classified as a TP, FP, FN, or TN). A data point predicted as "Fall" and "ADL" at various time indices needs to map to these segments. When a model makes a "Fall" prediction at t_i, the segment containing t_i is marked as a predicted "Fall." A segment is considered to be predicted as "ADL" when all data points within that segment are labelled as "ADL." If a segment does not have any indices corresponding to the model's predictions, then the next prediction after the segment's endpoint is used (the idea being that this would have encompassed the segment). For example, if we have the segments as in Table 2 and the model predicts the following values at indices shown in Table 3, then using our evaluation will result in the classification shown in Table 4:

Table 3. Example of Predicted Values at Various Indices

10-Fall, 20-ADL, 50-ADL, 60-ADL, 80-ADL, 90-ADL, 100-Fall, 120-ADL

Table 4. Classification Results

Segment	Actual	Predicted	Reason	Classification
(0,24)	ADL	Fall	Fall was predicted at index 10.	FP
(25,49)	ADL	ADL	No predictions in segment, first prediction following segment at index 50 is ADL.	TN
(71,95)	Fall	ADL	All predictions in segment are ADL.	FN
(96,120)	ADL	Fall	Fall was predicted at index 100.	FP

Note that this evaluation method is sensitive to the assumption that 25 data points represent a real-world fall duration.

5 OFFLINE EXPERIMENTS

We now present our experiments investigating the effectiveness of ensemble deep learning for fall detection, and compare it against single (non-ensemble) models. In the following, we first describe our experimental setting and then present and discuss the experimental results.

The experiments were conducted over two datasets (SmartFall, UniMiB), described in Section 3.1. The fall and ADL data in these two datasets are simulated from subjects of varying ages, heights and weights. Note that the way the data were collected in the two datasets are different. The SmartFall dataset was collected using a smartwatch attached to the wrist, while the UniMiB dataset was collected using a smartphone attached to the body. We train and evaluate a fall detection model on each dataset. At the time of writing this article, there is no other public smartwatch dataset that can be used to test a global fall detection model for a smartwatch. In Reference [4], the authors performed an evaluation of fall detection models using 14 different fall datasets and concluded that it is not possible to train a global fall detection model by combining fall data collected differently.

Offline Experiment Setting: All of our offline experiments were conducted on a Dell Precision 7820 Tower, 256 GB RAM, and one GPU (GeForce GTX 1080). Our evaluation metrics and method were described in Section 4. Both recall and precision values are in the range between 0 and 1. The higher value of the measure indicates the more effective model.

We also present a *PR (Precision vs. Recall)* curve for each experiment to show the trade-off between precision and recall values at different thresholds. We opted for a PR curve instead of the most common *ROC curve (True -positive Rate (TPR) vs. False-positive Rate (FPR))* due to the fact that FPR = FP/(FP+TN) is affected by the number of True Negative (TN) predictions. In a fall detection application the number of negative samples ("NonFalls") is much larger than the number of positive samples ("Falls"), thus FPR will always be small even in the presence of a high number of false positives.

Furthermore, we include a weighted F1-score (F_β), using the equation below, to combine precision and recall performance into one number:

$$F_\beta = (1 + \beta^2) * \frac{recall * precision}{(recall + \beta^2 * precision)}. \quad (1)$$

We set β to 3 for our calculations as this accurately reflects the higher emphasis we place on recall.

We grouped the experiments into four types. The first type of experiments is related to a single deep model and various optimization/adjustments performed on that. The second type is ensemble deep learning using AdaBoost.

Table 5. Single LSTM Models on SmartFall Dataset

	LSTM 20N	LSTM 30N	LSTM 50N	LSTM 80N
Precision	0.58	0.58	0.61	0.65
Recall	0.97	0.98	0.96	0.95
Weighted F1	0.907	0.915	0.909	0.911

The third type of experiments uses Stacking as the ensemble technique and demonstrates that Stacking using all available datasets can outperform the best single deep model. The last group of experiment is the validation of the best performing ensemble stacking model using the UniMiB dataset.

5.1 Single Deep Learning Model

In this section, we first compare the result of classic ensemble learning with single LSTM model. We performed experiments using popular classic ensemble learning techniques such as Random Forest (RF) and Gradient Boosting (GB). We then performed experiments with 1D-CNN. Figure 5 shows a performance comparison for the different models. Our results show that the single LSTM significantly outperforms both RF and GB, while also exhibiting slightly stronger performance than 1D-CNN in the majority of the PR curve range.

The inferior performance of the RF and GB models, compared to the deep learning models, can be explained by the fact that the manually extracted features cannot capture the full spectrum of the signal properties when a raw signal window is converted into a static feature vector. The 1D-CNN model performs much better than the classic ensemble models, however, it still yields a slightly lower overall performance compared to the LSTM model. A 1D-CNN can effectively model the pattern of fall events, however, an LSTM network has the added advantage that it can take into consideration the temporal dynamics that are commonly present in most fall events, including what happened immediately before and after the fall. Based on the initial performance comparison of different classification algorithms, we will focus our attention on different variations of the best performer among them (i.e., LSTM).

In the following paragraphs, we focus on LSTM experiments. We first performed an experiment on LSTM model by altering either the number of neurons per layer, or the number of LSTM layers. The results for this experiment are important, since they indicate if basic network adjustments are all that is required to improve small sample learning in IoT applications. To better understand the effects of these alterations, we divide this experiment into two parts. The first part examines the result of changing just the number of neurons per layer. This will show how much performance gain, if any, is obtained from additional neurons. The second part is concerned with the effects of an additional LSTM layer. Models used in this part of the experiment still differ in neurons, but they have an additional recurrent layer. The model that produces the best results in this section we will use as our baseline model; the model to which all ensemble approaches will be compared to.

We first look at four LSTM models of various sizes. Each model consists of 20, 30, 50, or 80 neurons per layer. For selecting these values, we first look for a lower neuron count that could still produce an accurate model. Our previous work on fall detection [18], indicates that a network with as low as 20 neurons can effectively model a fall event, and therefore it is a good choice for the smallest model. We start with 20 neurons and increase this number progressively; up to four times as many neurons. This provides a large enough range to help model the relationship between performance and neurons per layer. Each model was trained and tested on the SmartFall dataset three times. The average recall and precision over the three runs were recorded. Figure 6 shows the PR curve for this experiment and Table 5 shows the best results achieved by each model.

The recall for each model is close until they reach a precision of ≈0.8. The results favor the 30-neuron model after this point. These performances are still comparable enough that random factors such as dropout and weight

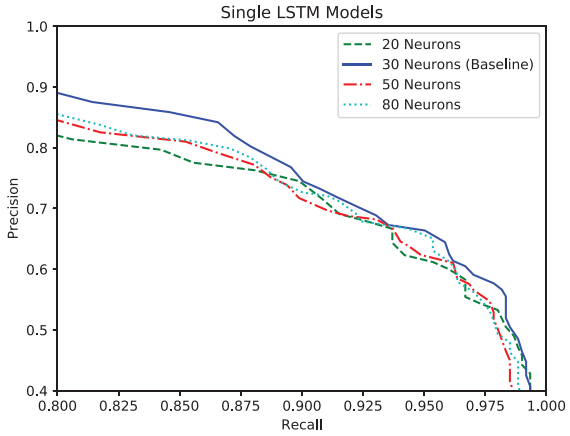

Fig. 6. PR curve of single deep model by varying the number of neurons.

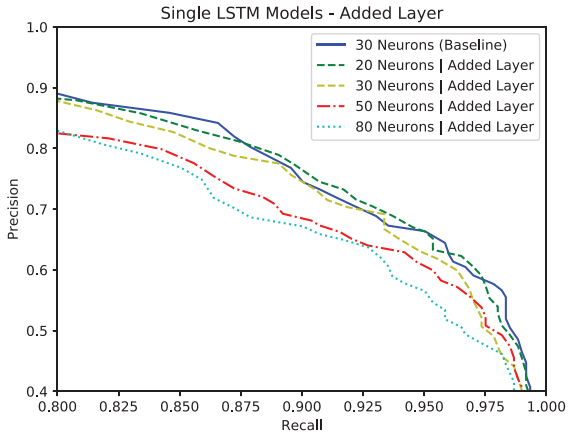

Fig. 7. PR curve of single deep model with deep layers and varying number of neurons.

initialization may explain the difference in results. Regardless of the impact of those factors, we can be confident that the performance does not improve with additional neurons.

We then look at 4 LSTM models of various sizes, and an additional recurrent layer. This layer is inserted directly after the first LSTM layer. We still use model sizes of 20, 30, 50, and 80 neurons for this experiment. Models are again trained and tested on the SmartFall dataset 3 times, and the average recall and precision were recorded. Figure 7 shows the PR curve for this experiment and Table 6 shows the best results achieved by each model.

The results show that additional neurons or layers do not offer any obvious improvements in performance. In fact, in most cases, creating a more complex network hurts the performance, possibly due to over-fitting issues. Given that the 30-neuron model requires less computation to train its parameters, we will use it as our baseline model for future comparisons.

5.2 Ensemble Deep Model with Boosting

Next, we consider the potential performance gain from an ensemble of deep models learning optimized using AdaBoost, as described in Section 3.3.1 above. Each model in the ensemble is a one-layer LSTM with the same

Table 6. Single LSTM Models on SmartFall Dataset Additional recurrent layer

	LSTM 30N	LSTM 20N,2L	LSTM 30N,2L	LSTM 50N,2L	LSTM 80N,2L
Precision	0.58	0.61	0.60	0.56	0.57
Recall	0.98	0.97	0.96	0.97	0.95
Weighted F1	0.915	0.915	0.908	0.902	0.890

Table 7. Boosted LSTM Models on SmartFall Dataset

	Precision	Recall	F1
Baseline	0.5768	0.9784	0.9147
10 Model, 15 Neuron	0.6049	0.9751	0.9189
10 Model, 30 Neuron	0.5621	0.9453	0.8850
15 Model, 10 Neuron	0.5882	0.9453	0.8912
50 Model, 6 Neuron	0.5585	0.9502	0.8880
50 Model, 30 Neuron	0.6234	0.9552	0.9069

number of neurons. We investigate how an ensemble of LSTMs compares to a single LSTM, as well as how the number of models in the ensemble and the number of neurons in each model affects the performance of the boosted classifier.

We choose as our baseline the best model from the previous section—that is, a single LSTM model with 30 neurons. We then run five boosted ensembles of LSTM's with various numbers of models and numbers of neurons. Each ensemble uses 200 samples per sub-model, selected according to the AdaBoost algorithm. Because our previous experiments suggest that 30 neurons is sufficient, we focus on LSTM's with at most 30 neurons. We use between 10 and 50 LSTM models in the ensemble to get an idea of the effects of using different numbers of models. The results are shown in Table 7.

We see that the results between the different ensembles are very similar: all models have a precision around 0.6 and a recall around 0.95. Interestingly, the baseline model has the best recall, suggesting that the boosted models overfit the training data. This behavior is not surprising, as Adaboost models are known to quickly overfit when the weak learners used are actually not so weak, as is the case with using LSTM networks as weak learners. The F1 scores are all very close; the ensemble containing 10 models and 15 neurons per LSTM barely beats the baseline at a score just under 0.92. We also plotted the PR curves, which are shown in in Figure 8.

We see in figure 8 that for almost every value of precision and recall, our baseline model performs best; however, there are two sections (recall around 0.94 and 0.97) where the precision of the 10-model ensemble with 15 neurons is slightly higher than the baseline. The precision of the two 50-model ensembles is very close up until a recall of 0.925, at which point the 30 neuron model considerably outperforms the 6-neuron model. An opposite trend is seen with the two 10-model ensembles, in which case the 15-neuron ensemble outperforms the 30-model ensemble at values of recall > 0.925. This suggests that no conclusion can be made from these experiments about how the number of neurons in the LSTM's affects the result of the overall classifier. We also see that when we keep the number of neurons in each LSTM constant at 30, the 50-model ensemble has significantly higher precision for all values of recall than the 10-model ensemble. This suggests that larger ensembles can yield better results; however, this conclusion is marred by the fact that the baseline model, with only one LSTM, considerably outperforms both these models in the majority of the PR curve range. In summary, we do not see any performance gains over the baseline model by using boosting.

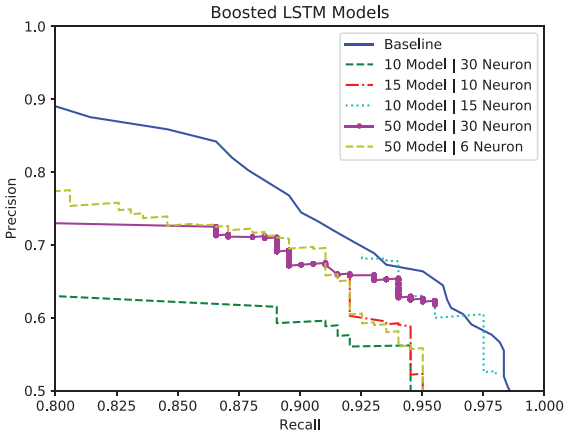

Fig. 8. PR curve of boosted LSTM models.

We believe that boosting fails to work on our dataset, because it is overfitting to the training data. Given that AdaBoost assigns samples randomly with replacement and re-adjusts the assignment probabilities with every iteration, it is very possible that some fall samples are not seen at all during training. This can be especially devastating, since falls are rare in comparison to ADL's, meaning that the models have a much higher chance of seeing ADL's.

Finally, it may be that the phenomenon we are trying to model (falling) is too complex to be broken into small models that see only a small fraction of the data. Given that each component model only sees 200 disparate data samples, it is possible that each model in the ensemble does not have enough predictive power to properly identify falls. This could explain why the ensemble performs worse than the single LSTM model, which has the opportunity to learn complex patterns in the training data by seeing a large stream of continuous data. Thus, we conclude that boosting is not a successful method for fall prediction.

5.3 Ensemble Deep Model with Stacking

This section explores the results of stacking LSTM models on the SmartFall dataset. Each stacked ensemble was altered by the number of models, and the number of neurons per model. We also divide these experiments based on how the training data is assigned to each model. The first part looks at two methods of assigning training data. For the first method, consecutive subsets of the training data are assigned to each model, as described in the Stacking Section 3.3.2. For the second method, we assign all the training data to every model. Comparing these methods shows that training the models on subsets of the data, as opposed to the whole training set, creates the most diverse and effective ensembles. The second experiment involves training each model on a specific type of fall data. For example, an ensemble may include a model trained on left falls, a model trained on back falls, and a model trained on a combination of these falls. ADL data is evenly distributed across such models. This method is isolated into its own experiment so we can compare the effects of training on different combinations of fall types.

We begin our evaluation of the stacked ensemble models by plotting a loss curve of the optimization process. Figure 9 shows the validation loss curves obtained during the training process of the single deep learning models (LSTM and 1D-CNN), discussed in previous sections, as well as a stacked ensemble of four LSTM models. It is evident that the ensemble loss is much lower than the single models. The ensemble loss starts at about the same level as the single LSTM model but it quickly converges to a lower value, as the weights of the stacked models are optimized using adaptive gradient descent. This is a positive indication towards better prediction accuracy. Upon closer examination, the success of the ensemble model appears to stem from the fact that it eliminates

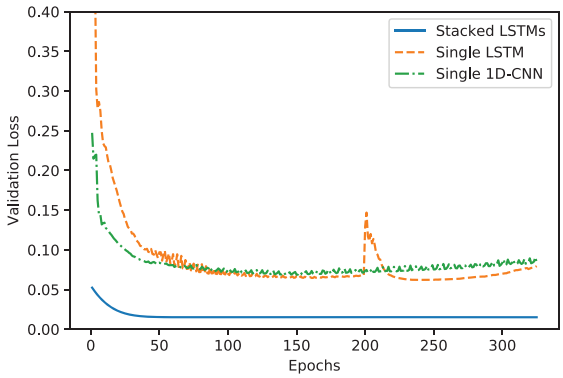

Fig. 9. Comparison of validation loss curves of single deep learning models against a stacked ensemble of LSTM models.

some of the false-positive predictions that the single models produced. Individual LSTM models tend to agree on the real falls, thus maintaining the number of true positives, but they disagree on the false predictions, which allows the ensemble weighted voting scheme to produce fewer false positives.

Subsequently, we perform various experiments with different variations of LSTM models. The ensembles vary in the number of LSTM models used, number of layers in each LSTM model, and number of neurons in each layer.

In summary, the different types of experiments we performed are noted as follows:

- **Baseline**: Single LSTM model with 30 neurons per layer.
- **# Neurons | Added Layer**: Single LSTM model with specified number of neurons and additional LSTM layer.
- **# Model | # Neuron**: Stacked LSTM models with specified number of models and neurons. Each model trained on subsets of the data.
- **# Model | # Neuron (all data)**: Stacked LSTM models with specified number of models and neurons. Each model trained on all the data.
- **# Model | Independent**: Stacked LSTM models of specified number of models and neurons. Each model trained with one unique type of fall.
- **# Model | Contrasting**: Stacked LSTM models of specified number of models and neurons. Each model trained with a combination of two different types of falls.

We present our first experiment of ensembles trained on either ordered subsets of data, or the entire training set. The ensembles are structured as follows: A 10-model, 15-neuron per model ensemble with each model trained on subsets. An 80-model, 6-neuron ensemble with each model trained on subsets. A 4-model, 30-neuron ensemble with each model trained on the whole training set. We believe using these ensembles will help us understand different parts of the spectrum; from a high number of models with low neuron count, to a low number of models with high neuron count. The experimental results of single deep models suggest there is no reason to go above 30 neurons. The PR curve for this experiment is shown in Figure 10, and Table 8 shows the best results achieved by each model.

A large performance gap can be seen between the ensembles trained on subsections, and the ensembles trained on all training data. The subsection ensembles, the 10-model, 15-neuron and 80-model, 6-neuron ensembles, are likely suffering from similar issues that boosting experienced. Although the stacking meta-algorithm may help prevent overfit models from being weighted too high, the datasets' class imbalance may cause models overfit to ADL data to receive high weight. The 4-model, 30-neuron ensemble results show the benefit of each model

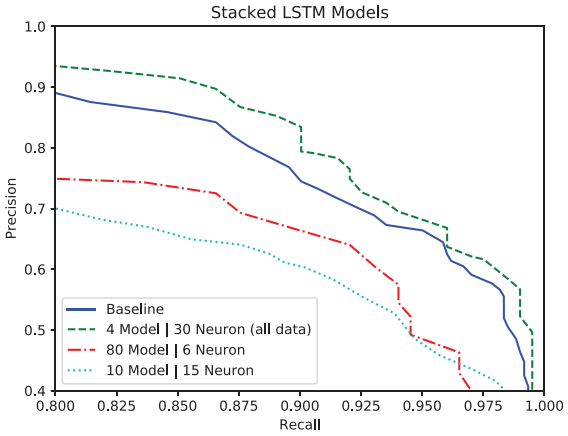

Fig. 10. PR Curve of stacked LSTM models trained on data subsets.

Table 8. Stacking Results on SmartFall Dataset

	BL	ST 4M,30N	ST 80M,6N	ST 10M,15N
Precision	0.58	0.58	0.57	0.53
Recall	0.98	0.99	0.94	0.94
Weighted F1	0.915	0.922	0.884	0.871

Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons.

having a good understanding of the data. Training each model with the whole training set helps assure that each model can correctly deal with the class imbalance and be properly weighted by the meta-algorithm.

Next, we look at models trained on specific fall types. Each ensemble consists of a low number of models, and 30 neurons. We structure the ensembles this way so that we can expand upon the good results obtained previously by the 4-model, 30-neuron ensemble. While training models on subsets of the data may not produce better results than training on all of the data, it is possible that training models on specific fall types will. Three types of ensembles were selected for this purpose. The first one consists of four models, where each is trained on one unique fall type. We hypothesize that models specialized in one fall type may not have to suffer performance tradeoffs by being forced to converge on all fall types. The next ensemble also consists of four models. Each model is trained on two contrasting fall types. Contrasting types refer to the difference between back/front falls and left/right falls. In total, one model is trained on left and back falls, one is trained on left and front falls, one is trained on right and back falls, and the last is trained on right and front falls. This can potentially help keep the models diverse and generalized. The last ensemble is an extension of the previous. It consists of 8 models, where the models in the previous ensemble are simply repeated twice. We do this because dropout and weight initialization provide more opportunities for these models to be more diverse, thus creating more models that can each contribute a useful output. The PR curve for this experiment is shown in Figure 11, and Table 9 shows the best results achieved by each model. We also include the the 4-model, 30-neuron ensemble so we can compare with the best result from the previous experiment.

The performance between these ensembles is closer than the previous experiment. However, the 4 model, 30 neuron ensemble still gives the best results overall. This suggests that simply training each model on all the data creates the most powerful ensemble. Comparing only ensembles trained on specific fall types, training one

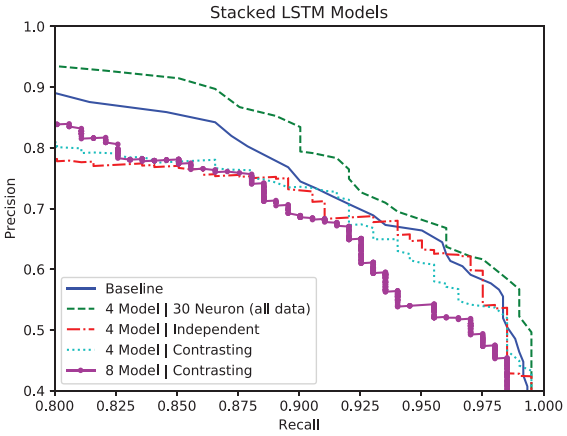

Fig. 11. PR curve of stacked LSTM models trained on specific fall types.

Table 9. Stacking Results on SmartFall Dataset Subsets

	BL	ST 4M,30N	ST 4M, Ind.	ST 4M, Cont.	ST 8M, Cont.
Precision	0.58	0.58	0.62	0.54	0.52
Recall	0.98	0.99	0.97	0.98	0.97
Weight F1	0.915	0.922	0.919	0.905	0.892

Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind = Independent, Cont. = Contrasting.

fall type per model provided the best results by a small margin. An indication that having each model target one fall type produces more useful models.

Across the experiments in this section, the 4-model, 30-neuron stacked ensemble produced the best results and outperformed the baseline. Models in this ensemble were each trained on the whole training set. It is likely that its results over baseline can be explained by the fact that each of the four models were as strong as a baseline model, but converged slightly differently than the rest. The meta-algorithm was able to combine these differences in a useful way.

5.4 Ensemble Deep Model with Stacking on UniMiB

This section takes the best performing models on our SmartFall dataset, and tests them on the UniMiB dataset. UniMiB is larger than the SmartFall dataset and has a more balanced ratio of falls to ADLs. Also, the acceleration data in the UniMiB dataset were collected through smartphones attached to body of the participants as opposed to smartwatch attached to the wrist, thus making it an "easier" dataset for fall detection.

Given the differences in the datasets, it is reasonable to believe there will be differences in how the models perform on them. We will be examining the results of the models on this dataset in a relative fashion to the SmartFall dataset. As it would not make sense to directly compare metrics such as recall and precision across these two unique datasets, we are more concerned with how the models do against each other. For this experiment, we use the same set of ensembles used in the previous experiment with stacking. Our baseline, a 4 model, 30 neuron ensemble trained with the whole training set, and the ensembles trained with specific fall types. These models were selected for this experiment based on their ability to reach high precision while maintaining high recall, as well as their weighted F1-score. Our main goal for this experiment is to either reinforce the results we obtained on our SmartFall dataset, or reveal that some models work better with different datasets. We present our final

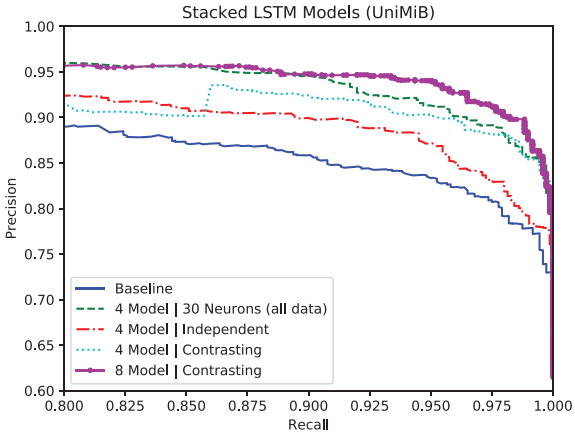

Fig. 12. PR curve of stacked LSTM models on UniMiB.

Table 10. Stacking Results on UniMiB Dataset

	BL	ST 4M,30N	ST 4M, Ind.	ST 4M, Cont.	ST 8M, Cont.
Precision	0.77	0.85	0.78	0.82	0.86
Recall	0.99	0.99	0.99	1.0	0.99
Weight F1	0.967	0.980	0.971	0.979	0.980

Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind. = Independent, Cont. = Contrasting.

experiment. Figure 12 shows the PR curve for each model, and Table 10 shows the best results produced by each model.

There are clear differences between how the models perform on the two datasets. While the baseline had previously performed the second best on the SmartFall dataset, it is now at the bottom of the pack. We also see that the relative performance of the models trained with specific fall types has changed. Whereas the independent model performed the best of the three on the SmartFall dataset, it performed the worst of the three on UniMiB. The 8-model ensemble of contrasting fall types has one of the best performances, comparable with only the 4-model, 30-neuron ensemble. Again, we see the 4-model, 30-neuron ensemble with one of the best performances of the group. This reinforces our confidence in stacked ensembles where each model is trained with the whole training set. It appears that this is the only type of ensemble that is not sensitive to the datasets. While the models trained on specific fall types performed better than baseline, due to the better performance by baseline on the SmartFall dataset, we cannot confidently claim that this is due to the training method and not just the increase in models. Overall, we can be confident that the methods surrounding the stacked 4-model, 30-neuron ensemble produce the best results.

5.5 Discussion on on Offline Experimental Results

We carried out multiple groups of experimental studies. Our initial experiments on classic ensemble techniques (RF, GB), 1D-CNN, and a single LSTM model demonstrates superiority of LSTM model.

Our subsequent experiments show that a stacked ensemble of LSTM models can perform better than a single LSTM model. Using a combination of model outputs appears to improve the subtle learning gaps of a single LSTM model on small training datasets. Attempts to diversify the learning of our ensemble were made by altering the

Table 11. Results of the Five Best Models on SmartFall and UniMiB Datasets

		BL	ST 4M,30N	ST 4M, Ind.	ST 4M, Cont.	ST 8M, Cont.
SmartFall	Precision	0.58	0.58	0.62	0.54	0.52
	Recall	0.98	0.99	0.97	0.98	0.97
	Weight F1	0.915	**0.922**	0.919	0.905	0.892
UniMiB	Precision	0.77	0.85	0.78	0.82	0.86
	Recall	0.99	0.99	0.99	1.0	0.99
	Weight F1	0.967	**0.980**	0.971	0.979	0.980

Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind. = Independent, Cont. = Contrasting.

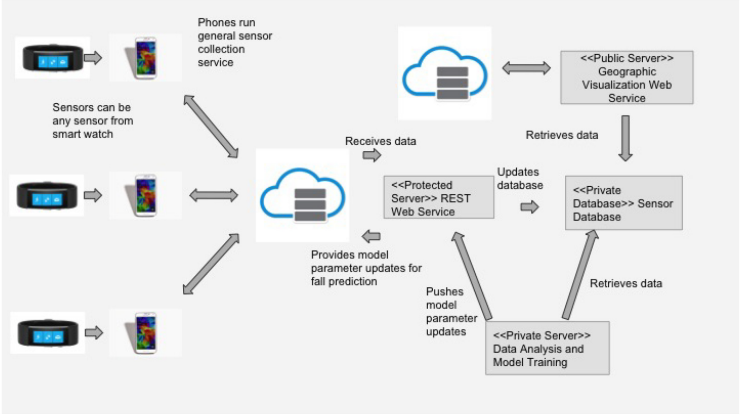

Fig. 13. Architecture of SmartFall system.

structure and training data of each model. In regards to the ensembles that did not perform better than baseline, we believe that there were key fallbacks to some of the approaches that limited the ensembles' performances. For our boosting approach, one fallback was the potential for a boosted LSTM to overfit the samples it trains on. Since the weight of a model is determined by its training error, overfit models exhibit small training errors and thus provided a large contribution to the ensemble. These high-weighted models were often not suited to generalize well on new data. Our stacking approach also experienced over-fitting. However, the issue boosting experienced with its weighted models was mitigated in our stacking approach by the meta-algorithm. The meta algorithm, a list of linear weights for the models, was often able to make sense of the models' outputs well enough to generalize to the testing data. This list of weights is trained only after the ensemble has completed training for all models. Doing it this way can allow more generalized models to be rewarded with a higher weight, as opposed to a boosted model, whose weight is purely based off a select number of samples. Overall, we have shown that a stacked ensemble produces better results than any single LSTM or other ensemble approach we tested. Table 11 provides a summary of the best results achieved by each approach.

6 REAL-WORLD EXPERIMENTS

6.1 Fall Detection System Architecture and App

Figure 13 shows an overview of the architecture of our SmartFall application that we deployed for real-world experimentation. It is a three-layered architecture with the smartwatch on the edge and the smartphone in the

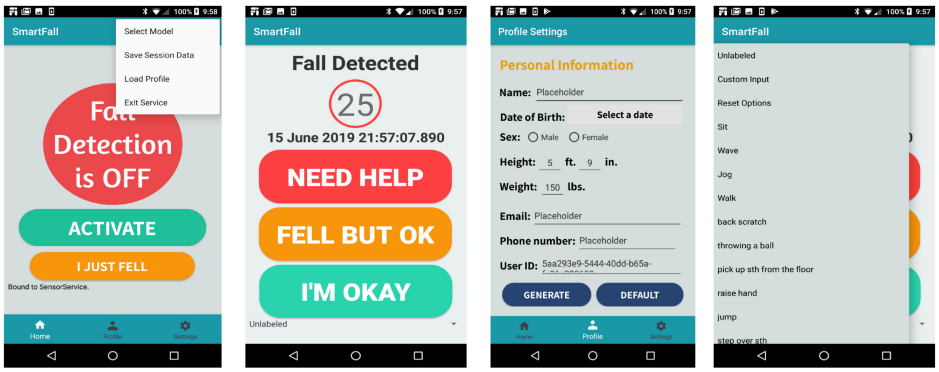

Fig. 14. SmartFall user interface.

middle layer that runs the application. Our system is structured such that the sensed data from the smartwatch can be stored locally to preserve privacy and is in close proximity to the program that processes and analyzes the data in real-time. However, due to the limited storage capacity of the smartphone, we also provide the option of periodically remove the sensed data or transfer the sensor data (with consent from user) to a server securely for continuous refinement of the fall detection model and for the long term archival. The server, which is situated in the inner most layer also serves as the heavy-duty computational platform, which consists of multiple services including a web server to host applications that can visualize aggregated sensor data for public health education, a sensor database for archiving and visualizing sensed data from the smartwatch of the user who has given the consent and machine learning services for analysis of the archived data for continuous refinement of the fall detection model.

The Microsoft Band 2 was chosen as the wrist worn device over other options in this prototyping phase due to the variety of sensors it supports and the low cost of acquiring the watch. At the time of this writing, Microsoft has pulled the support for Microsoft Band 2; however, the watch can still be used so long as we do not need to do a factory reset. We are also in the process of porting the system to run on a Huawei watch, which is Android Wear compatible. The Nexus 5X smartphone was chosen to run our SmartFall application and receive sensor data from the smartwatch via a low-power bluetooth communication protocol. This Nexus smartphone has a 1.8 GHz hexa-core processor and 2 gigabytes of RAM memory. This proved sufficient for real time computation of the features, and for making the predictions, using models that were pre-trained offline.

6.2 Archiving Service and UI Design

We have implemented an archiving service that can be configured with a protocol where a participating user (with consent) can transmit sensed data in three-minute chunks to a designated server via a WiFi connection periodically. The archiving service is critical for the fine-tuning of the fall detection model via feedbacks from users in our real-world experiments. The UI design for this version of SmartFall App also includes buttons specifically for labelling false-positive, true-positive, and false-negative data samples in real time from users. Those labelled archived data samples serve as additional data for re-training our Ensemble RNN model. Figure 14 shows four views of the user interface (UI) of the SmartFall application. The screen on the left shows the home screen UI for the application and the second screen shows the UI when a fall is detected. We followed the best practices advocated in Reference [23] for the design of the UI for the elderly. The three main principles we adopted were strict color scheme with high contrasts, legible and big fonts, simple description of the system to engage them to use it.

We will briefly highlight some of the key features of SmartFall app. The home screen (leftmost screen in Figure 14) launches the SmartFall App when the user presses the "ACTIVATE" button. The user must set up a profile and load the profile before the App can be activated. The "I JUST FELL" button on this screen is designed to collect and label false-negative data samples, i.e., falls that were not detected by the Appendix A user can terminate the application by selecting the "Exit Service" menu choice on the upper right corner. This will archive all the sensed data to the designated server if consent was already given and close the application. When a fall is detected, the second screen of Figure 14 will be displayed with a programmable sound and a timestamp. The user is shown three buttons for interaction. The "NEED HELP" button will send a text message to the caregiver and also save and label the sensed data samples as true positives. The "FELL BUT OK" button will save the sensed data during that prediction interval as true positives without notifying the caregiver. The "I'M OKAY" button will save these data as false positives. If a fall is detected and the user did not interact with any of these three buttons, then after a specified duration (e.g., 25 s), the system assumes that the user might be hurt or unconscious and an alert message will be sent to the caregiver automatically. The third UI screen is for the one time initialization of the user profile before the application can be launched. This UI includes setting up the contact details of the caregiver. Note that minimal personal data is collected and all those data are stored locally in the phone. The automatically generated user-id is used by the system to differentiate different user's data on the server. During data archiving, only this user-id and the selected sensed data such as the accelerometer data are sent to the server.

When saving the data samples as false positives (pressing the "I'M OKAY" button), it will be helpful to also label what type of activity triggers the false-positive event for ease of diagnostics. The rightmost UI screen is being launched to enable the user to tag that activity. If the activity is not already in the tagged list, then user can select the "Custom input" menu item and enter a new activity, which will be added to the list.

6.3 Implementation of EnsembleRNN Model on SmartFall App

We used the four stacked LSTM models (ST, 4M, 30N) trained on the entire SmartFall dataset as discussed in Section 5.3 as the first ensemble model we want to test in real-world experiments. We first trained the four LSTM models and the meta-algorithm for combining the models' outputs. Training for each LSTM model completes after a predetermined number of epochs. This method produced more diverse models as opposed to cutting off the training when the network reaches its lowest loss. It was often the case that the networks, which are the same in structure, shared a similar learning peak despite differences in training provided by dropout and weight initialization. Having networks finish their training with different losses ensured that they did not learn the same information. The trained networks are then used to predict the outputs over the training data, so that the meta-algorithm, a list of weights, can be trained (via gradient descent) to determine the best weight for each model. We define the best weights to be the ones that provide the lowest overall loss on the training set. These trained weights for each LSTM model are saved and ported to SmartFall App. A frozen version of each trained LSTM model is obtained through TensorFlow library and ported to the SmartFall app as well. The SmartFall app is then able to read and run these frozen model files using TensorFlow's java library.

When running the model on the Android device using the SmartFall App, accelerometer data samples are continuously sensed by the smartwatch and sent to each of the four LSTM models for prediction. The prediction outputs are then run through the ensemble's meta-algorithm as described in Algorithm 2, which is simply four floating point values that weight the outputs of their respective networks, and are then aggregated to make a single output. Since we are dealing with time series data, we find that the value of a single output is not very useful. Instead, we utilize a heuristic, where consecutive outputs from the ensemble are averaged. We determined that 20 consecutive outputs to be averaged provided a range long enough to reduce false positives but also maintained good recall. The 20 aggregated prediction output is then compared to a pre-set threshold. If the final prediction breaches the threshold, then the model predicts that a fall has occurred. We determined this threshold

to be 0.3 by performing an exhaustive search of different threshold values over the training data to see which threshold, compared against the ensemble's predictions on the data, provided the best precision and recall.

6.4 Real-world Experiments Set Up

The offline experiments show that the best performing model is the four stacked ensemble RNN models with 30 neurons and trained on all the SmartFall dataset. At a probability threshold value of 0.5, this model has almost perfect recall, but has only around 60% precision. In practical terms, based on the precision formula shown in Table 1, a 60% precision means that for every fall that was correctly detected, about 0.66 false positives were generated. Our SmartFall data contained a total of 528 falls, which means about 348 false positives could be generated overall in our cross validation experiments. This number is not very meaningful as an evaluation criterion of how many false positives someone would get in real-world use. That is because the ratio of falls that exists in the training dataset compared to the total size of the dataset, including the ADLs, is much higher than what one would get in real-world use. While it is possible to perform an offline evaluation on live recorded data, the real-world experiments allow us not only to assess the fall detection for real-life everyday activities, but more importantly, it helps to mitigate false positives via user's feedback. It is not possible to simulate all real-world ADL activities in a lab setting, for the same users who performed the simulated falls, without real-world experiments.

The real-world experiments were designed to minimize inconvenience to the user who participated in the experiments while still capturing as much information as possible about the user's activities that can be used to improve the model. We recruited three volunteers for real-world experiments. Each volunteer was told to carry the smartphone running our SmartFall application, and wore the Microsoft SmartBand watch on his/her left wrist paired with the phone for as many hours as convenient in a day. Within a day, each volunteer was asked to at least perform 20 falls on a mattress (five of each: back, front, left, and right) and four prescribed ADLs (sitting down, walking, jogging, and hand waving). A spreadsheet was given to each volunteer to record the correctly detected (TP) and missed falls (FN), as well as possible false alerts (FP).

Besides those prescribed falls and ADLs, users were otherwise asked to go about their day normally with the stipulation that they stay above a baseline of activity (e.g., not just sleeping or sitting on couch for the whole day). The archiving service on SmartFall would continuously save all the sensed accelerometer data with associated timestamps when the App was running. In the event of a fall being detected by the SmartFall App, if it was a true fall, the user would label it by pressing "FELL BUT OK" or "NEED HELP" on the UI. This would create a meta record that saved the timestamps of when the event was detected, and labelled that as true-positive event. If there was a false positive, then the user would press the "I'M OKAY" button and optionally provide a label that can describe the activity occurring at the time of the prompt. A false-positive meta record would be created. If a fall went undetected, then the user would label the missed fall by selecting the "I JUST FELL" button. A meta-record for false negative will be created. At the end of the testing period, the meta records were used to extract the original accelerometer data associated with each type of event and use those as additional training data. The prescribed activities that were mandatory for each user to perform allowed us to have a standard way to compare user's overall results while their normal daily activities served as a pragmatic test of the real-world use of our SmartFall App.

6.4.1 Custom False-positive Metric. It is expected that activities that produce high acceleration values will generate more false positives than sedentary activities. To evaluate the number of false positives that occur for each user while controlling for different activity levels, we introduce a new custom metric for measuring false positives. The new metric takes into consideration the number of acceleration "spikes" that occurred during a time period and uses those as normalization factor for the false-positive rate.

We consider a "spike" to occur when the magnitude of acceleration for a user exceeds the double of their average acceleration. The average acceleration is computed by processing all the acceleration data archived

Table 12. First Ensemble RNN Model Trained on offline SmartFall Dataset

	User1	User2	User3	Total
Hours worn	19.8	12.7	0.57	33.07
Number of False Positives	300	253	7	560
Number of Spikes	599	454	14	1,067
Normalized Precision	0.5	0.44	0.5	**0.48**

Table 13. Second Ensemble RNN Model Trained with Additional False-positives Samples

	User1	User2	User3	Total
Hours worn	10.24	1.7	2.95	14.89
Number of False Positives	6	27	3	36
Number of Spikes	295	177	127	599
Normalized Precision	0.98	0.86	0.98	**0.94**

during the testing period. When a spike is detected, we ignore the following 16 sample points (approximately half a second with 32 Hz sample frequency) before examining another spike. This ensures that a single jump in acceleration magnitude, which could generate consecutive high sample points, is not read as multiple spikes. The total number of false positives a particular model detects will be compared against the total number of spikes (noted as $\#spikes$) a user emitted to give our Normalized Precision (NP) value:

$$NP = \frac{\#spikes - FP}{\#spikes}. \tag{2}$$

Spikes are a good metric to use in our case, because there is a direct correlation between the user's activity level and the number of false positives. This will be used in the discussion of our results to compare the models we tested.

6.5 Discussion on Real-world Experimental Results

We present and discuss the results of two ensemble RNN models tested in a real-world setting. The first model was trained only on our SmartFall offline dataset, consisting of falls and ADLs performed by 14 users in an experimental setting discussed in Section 5. The second model was trained with the same dataset, as well as the additional false-positive samples gathered from the three volunteers in real world from running the SmartFall App using the first ensemble RNN model.

These two models were compared using two metrics: Recall and Normalized False-positive rate. Recall, the rate of falls correctly predicted, is necessary to determine how well the models do at detecting falls, and Normalized False-positive rate is necessary to determine the precision of the model, i.e., how often the model correctly interprets non-fall movements as non-falls. We also examine how the models do on a prescribed set of ADLs. This is important in comparing how the models do on specific ADLs vs real-world movements, which includes a far wider spectrum of activities. We first present how each model performed on real-world movements.

The important metric for these results is the final Normalized Precision of each model. Since the percentage of spikes detected vastly improved from the first model to the second model, we can conclude that there is a strong performance boost to non-fall prediction (precision) when training with additional false positives.

To conclude that the second ensemble RNN model was an overall improvement, however, the recall for each model must be examined. If the second model could not maintain an acceptable rate of fall detection, then the cost

Table 14. Recall for the First Ensemble RNN Model Trained on Offline SmartFall Dataset

	User1	User2	User3	Total
Falls detected	19	18	20	57
Fall performed	20	20	20	60
Recall	0.95	0.90	1.0	**0.95**

Table 15. Recall for the Second Ensemble RNN Model Trained with Additional False-positive Samples

	User1	User2	User3	User4	User5	Total
Falls detected	17	18	19	17	18	89
Fall performed	20	20	20	20	20	100
Recall	0.85	0.90	0.95	0.85	0.9	**0.89**

of improving non-fall prediction may be too large. The following tables 12, 13, 14, and 15 show the performance of the two models on falls.

Even with significantly fewer false positives, the second model still maintained around 90% on recall. Note that there is a different in number of users recruited across these two models. We recruited additional volunteers to test the recall for the second model to further validate that our recall has not suffered. Fewer users were used to evaluate the first model for recall, since that model was already evaluated in the offline experiments.

These initial results revealed that an effective system needed to be trained on a broader range of activities. After incorporating data collected from normal daily activities for around 33 h and retraining models on our more diverse dataset, the system maintained a high fall detection rate and greatly reduced its false-positive rate. This establishes that a system trained on only falls and some prescribed ADLs in a laboratory setting is insufficient to create a usable system in the real world. There are a wide variety of ADLs that will be missed when training models on a dataset generated by performing a prescribed set of activities. Moreover, it is difficult to anticipate all possible ADLs and cater to them in the training dataset.

We learnt from our real-world experiments that a bootstrapped approach of retraining models on false positives creates a more robust system without sacrificing much on the rate at which the models detect falls. This means that a model can be calibrated to achieve high performance by retraining itself with user feedback on false positives, all without requiring additional falls from users. End users of this system could calibrate it in a day by telling the model when its predictions are false positives, and letting it retrain overnight.

Our real-world experiments indicate that a stacked four-LSTM ensemble trained on small dataset performs well in a real-world context when retrained on its own false positives. They also indicate that analyzing models in a real-world context, instead of from a discrete set of recorded activities, is an important and overlooked aspect that should be studied more.

7 CONCLUSION AND FUTURE WORK

Deep neural networks have shown superior performance for fall detection as they eliminate the need for extraction of hand crafted features, as shown in our earlier work [18]. However, deep neural networks face practical challenges when used on IoT applications, which in general tend to have limited training samples and imbalanced datasets. Training a deep neural network with a small dataset tends to produce overfit models. Getting a large amount of labeled data is costly or even impossible in many IoT applications. In this work, we were able to mitigate the scarcity of training samples in fall detection using wearable devices by combining RNN with ensemble techniques such as boosting and stacking. We conducted an extensive set of experiments on two dif-

ferent fall datasets and concluded that the stacking ensemble technique combined with RNN can outperform the single deep RNN model. Boosted LSTM models did not provide any performance gain over the single deep model using our training dataset. We attribute that to the random assignment of subset of data to each boosted LSTM model and the higher chance for the boosted LSTM model to overfit the small samples selected for the training.

As documented in Reference [5], only 7.1% of the reported fall detection projects tested their models in a real-world setting. We wanted to validate the performance of our best performing stacking ensemble deep model in the real-world setting. This involved porting the offline fall detection model to the SmartFall App and re-designing the App to include an archiving service and an intuitive UI such that it could be used by the users to label false-positive, true-positive, and false-negative events in real-time. We are the first to report testing the fall detection model in this way.

Two important implications were drawn from our real-world experimental results, which will guide our future research. The first implication is that testing fall detection models trained on a specific set of falls vs. ADLs is not a good indicator of how the model will perform in the real world. Our offline experiments in the past, including the offline experiments conducted with the first model discussed in Section 5.5, have shown good performance on specific fall and ADL data. Despite this, the movements users undergo in the real world still triggered many false positives. This is an important implication to fall detection research as a whole, as we find most research deals with training and testing the models on specific ADLs. The second implication is that differences in activity levels across users may significantly affect the number of false positives generated. For example, activity levels may be much lower in elderly compared to younger users. For that reason, we introduced metric using the number of acceleration spikes generated by each user during a time period to objectively measure the performance of fall detection models in real-world, while controlling for different lifestyles and activity levels.

In summary, we are the first to use ensemble RNN for solving small sample learning using time-series datasets. We performed a comprehensive set of experiments using two different datasets to validate the performance of ensemble RNN models. We devised a consistent methodology for evaluating the model that is independent of how the prediction is accomplished. We demonstrated the necessity of real-world experiments in this problem domain. We proposed and implemented a practical real-world experiment to mitigate the false-positive problem and proposed a new metric called spike score that can measure the false alarms in a consistent manner despite the different level of activities carried out by different users.

Our future work includes the investigation of other small sample learning strategies. This includes data augmentation and personalization. For example, the initial ensemble model can be coupled with a personalized model as more data is collected from a specific person via user feedback from the SmartFall App. A personalized fall detection model can put more weight on training on specific types of ADLs that can be mistaken for falls, tailored to a particular person's physical characteristics and lifestyle.

ACKNOWLEDGMENTS

We thank various Texas State students, in particular Shuan Coyne, for helping with the fall data collection process and running of classical RF and GB experiments.

REFERENCES

[1] Fadel Adib, Hongzi Mao, Zachary Kabelac, Dina Katabi, and Robert C. Miller. 2015. Smart homes that monitor breathing and heart rate. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15)*. ACM, New York, NY, 837–846. DOI: https://doi.org/10.1145/2702123.2702200

[2] Apple Inc. 2019. Apple Watch Series 4. Retrieved from http://www.apple.com/apple-watch-series-4/activity/.

[3] Joseph D. Bronzino. 2006. Biomedical signals: Origin and dynamic characteristics; frequency-domain analysis. In *Medical Devices and Systems*. CRC Press, 27–48.

[4] Eduardo Casilari, Raul Lora-Rivera, and Francisco Garcia-Lagos. 2020. A study on the application of convolutional neural networks to fall detection evaluated with multiple public datasets. *Sensors* 20, 5 (2020). DOI: https://doi.org/10.3390/s20051466

[5] S. Chaudhuri, H. Thompson, and G. Demiris. 2014. Fall detection devices and their use with older adults: A systematic review. *J. Geriatric Phys. Ther.* 37, 4 (2014), 178–196.

[6] Xuhui Chen, Jinlong Ji, Tianxi Ji, and Pan Li. 2018. Cost-sensitive deep active learning for epileptic seizure detection. In *Proceedings of the ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB'18)*. ACM, New York, NY, 226–235. DOI: https://doi.org/10.1145/3233547.3233566

[7] Adele Cutler, D. Richard Cutler, and John R. Stevens. 2012. *Random Forests*. Springer, Boston, MA, 157–175. DOI: https://doi.org/10.1007/978-1-4419-9326-7_5

[8] Jerome H. Friedman. 2002. Stochastic gradient boosting. *Comput. Stat. Data Anal.* 38, 4 (2002), 367–378.

[9] Stephen Grossberg. 2013. Recurrent neural networks. *Scholarpedia* 8, 2 (2013), 1888.

[10] Yu Guan and Thomas Plötz. 2017. Ensembles of deep LSTM learners for activity recognition using wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 11 (June 2017), 28 pages. DOI: https://doi.org/10.1145/3090076

[11] Yu Guan and Thomas Plötz. 2017. Ensembles of deep LSTM learners for activity recognition using wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 11 (June 2017), 28 pages. DOI: https://doi.org/10.1145/3090076

[12] Ngu Anne Hee, Wu Yeahuay, Zare Habil, Polican Andrew, Yarbrough Brock, and Yao Lina. 2017. Fall detection using smartwatchsensor data with accessor architecture. In *Proceedings of the International Conference for Smart Health (ICSH'17)*.

[13] Andreas Janecek, Wilfried Gansterer, Michael Demel, and Gerhard Ecker. 2008. On the relationship between feature selection and classification accuracy. In *New Challenges for Feature Selection in Data Mining and Knowledge Discovery*. 90–105.

[14] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Retrieved from https://Arxiv:1412.6980.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. MIT Press, 1097–1105.

[16] Bogdan Kwolek and Michal Kepski. 2014. Human fall detection on embedded platform using depth maps and wireless accelerometer. *Comput. Methods Programs Biomed.* 117, 3 (2014), 489–501.

[17] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho. 2017. Human activity recognition from accelerometer data using convolutional neural network. In *Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp'17)*. IEEE, 131–134.

[18] Taylor R. Mauldin, Marc E. Canby, Vangelis Metsis, Anne H. H. Ngu, and Coralys Cubero Rivera. 2018. SmartFall: A smartwatch-based fall detection system using deep learning. *Sensors* 18, 10 (2018). DOI: https://doi.org/10.3390/s18103363

[19] Daniela Micucci, Marco Mobilio, and Paolo Napoletano. 2017. UniMiB SHAR: A dataset for human activity recognition using acceleration data from smartphones. *Appl. Sci.* 7, 10 (2017). DOI: https://doi.org/10.3390/app7101101

[20] Mirto Musci, Daniele De Martini, Nicola Blago, Tullio Facchinetti, and Marco Piastra. 2018. Online fall detection using recurrent neural networks. Retrieved from https://Arxiv:1804.04976.

[21] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-garadi, and Uzoma Rita Alo. 2018. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Syst. Appl.* 105 (2018), 233–261. DOI: https://doi.org/10.1016/j.eswa.2018.03.056

[22] RightMinder. 2018. RightMinder—Fall Detection for Android Smartwatches and Android Phones. Retrieved from http://www.rightminder.com.

[23] H. M. Salman, W. F. Wan Ahmad, and S. Sulaiman. 2018. Usability evaluation of the smartphone user interface in supporting elderly users from experts? perspective. *IEEE Access* 6 (2018), 22578–22591. DOI: https://doi.org/10.1109/ACCESS.2018.2827358

[24] Guto Leoni Santos, Patricia Takako Endo, Kayo Henrique de Carvalho Monteiro, Elisson da Silva Rocha, Ivanovitch Silva, and Theo Lynn. 2019. Accelerometer-based human fall detection using convolutional neural networks. *Sensors* 19, 7 (2019). DOI: https://doi.org/10.3390/s19071644

[25] Jose Antonio Santoyo-Ramon, Eduardo Casilari, and Jose Manuel Cano-Garcia. 2018. Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection with supervised learning. *Sensors* 18, 4 (2018). DOI: https://doi.org/10.3390/s18041155

[26] S. Shin and W. Sung. 2016. Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'16)*. 2274–2277. DOI: https://doi.org/10.1109/ISCAS.2016.7539037

[27] Angela Sucerquia, José David López, and Jesús Francisco Vargas-Bonilla. 2017. SisFall: A fall and movement dataset. *Sensors* 17, 1 (2017), 198.

[28] C. Tacconi, S. Mellone, and L. Chiari. 2011. Smartphone-based applications for investigating falls and mobility. In *Proceedings of the 5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops*. 258–261. DOI: https://doi.org/10.4108/icst.pervasivehealth.2011.246060

[29] T. Theodoridis, V. Solachidis, N. Vretos, and P. Daras. 2018. Human fall detection from acceleration measurements using a recurrent neural network. In *Precision Medicine Powered by pHealth and Connected Health*. Springer, 145–149.

[30] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. 2019. Deep learning for sensor-based activity recognition: A survey. *Pattern Recogn. Lett.* 119 (2019), 3–11. DOI: https://doi.org/10.1016/j.patrec.2018.02.010 Deep Learning for Pattern Recognition.

[31] S. Yu, L. Qin, and Q. Yin. 2018. A C-LSTM neural network for human activity recognition using wearables. In *Proceedings of the International Symposium in Sensing and Instrumentation in IoT Era (ISSI'18)*. 1–6. DOI: https://doi.org/10.1109/ISSI.2018.8538129

[32] Cha Zhang and Yunqian Ma. 2012. *Ensemble Machine Learning: Methods and Applications*. Springer.
[33] Wang Zhiqiang and Liu Jun. 2017. A review of object detection based on convolutional neural network. In *2017 36th Chinese Control Conference (CCC'17)*. IEEE, 11104–11109.
[34] Ahmet Turan Ozdemir. 2016. An analysis on sensor locations of the human body for wearable fall detection devices: Principles and practice. *Sensors* 16, 8 (2016). DOI: https://doi.org/10.3390/s16081161

Received July 2019; revised June 2020; accepted September 2020