Open Access

Open Journal of Internet of Things (OJIOT)
Volume 5, Issue 1, 2019
http://www.ronpub.com/ojiot
ISSN 2364-7108

Experimentation and Analysis of Ensemble Deep Learning in IoT Applications

Taylor Mauldin [A], Anne H. Ngu[A], Vangelis Metsis[A], Marc E. Canby[B], Jelena Tešić [A]

[A] Department of Computer Science, Texas State University, 601 University Drive, San Marcos, TX 78666-4684 Texas, USA {trm119, angu, vmetsis, jtesic}@txstate.edu
[B] Department of Computer Science, Rice University, 6100 Main Street, Duncan Hall 3122 Houston, TX 77005-1892 Texas, USA marc.canby@gmail.com

Abstract

This paper presents an experimental study of Ensemble Deep Learning (DL) techniques for the analysis of time series data on IoT devices. We have shown in our earlier work that DL demonstrates superior performance compared to traditional machine learning techniques on fall detection applications due to the fact that important features in time series data can be learned and need not be determined manually by the domain expert. However, DL networks generally require large datasets for training. In the health care domain, 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). Moreover, fall data is also inherently noisy since motions generated by the wrist-worn smartwatch can be mistaken for a fall. This paper explores combing DL (Recurrent Neural Network) with ensemble techniques (Stacking and AdaBoosting) using a fall detection application as a case study. We conducted a series of experiments using two different datasets of simulated falls for training various ensemble models. Our results show that an ensemble of deep learning models combined by the stacking ensemble technique, outperforms a single deep learning model trained on the same data samples, and thus, may be better suited for small-size datasets.

Type of Paper and Keywords

Regular research paper: *ensemble methods, deep learning, recurrent neural network, fall detection, time series, IoT.*

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 entitled "Staying Connected is Crucial to Staying Healthy" (WSJ, June 29, 2015 and "Digital Cures For Senior Loneliness" (WSJ, Feb 23-34, 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

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2019)* in conjunction with the VLDB 2019 conference in Los Angeles, USA. The proceedings of VLIoT@VLDB 2019 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

becoming more prevalent. The urgency for investment in health monitoring IoT technology is also echoed by another Wall Street Journal article (July 21, 2018) entitled "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 [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 [23]. 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, 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 recent 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 [18] 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 compared traditional machine learning (SVM, Naïve Bayes) techniques with deep learning, in particular the Recurrent Neural Network (RNN), for fall detection [14] 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 an online test with five subjects. Only a single deep model was trained and the standard parameter estimation and training procedures are being used in our earlier work. Moreover, the accelerometer data, which is a form of time-series data, was processed using a fixed-size sliding window approach. As pointed out by [7], 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 paper is on the 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, achieved only 86% accuracy, based on our real-world test with five subjects [14], due to the occurrence of false positives which reduced the precision of the detector. The medical field is often averse to a device that only works "most of the time". Therefore, there is definitely room to further improve the precision of our fall detection application without scarifying the recall for the practical real-world deployment of the fall-detection application.

In this work, we investigate the idea of combining popular ensemble techniques such as Stacking and Boosting [29] with RNN to create an ensemble of diverse learners to mitigate small sample dataset. 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 validated the generated ensemble models using two datasets. The first dataset was collected in house using the Microsoft Band 2 smartwatch [5] from 14 subjects ranging from 20 to 60 years old. The second dataset is the open source UniMiB database [15] that contains falls and Activities of Daily Living (ADLs) collected using a smartphone from 30 different subjects with ages ranging from 18-60.

Our results show that an ensemble of RNN models outperforms a single RNN model trained on the experimental data. Furthermore, 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.

The main contributions of this paper are:

- An in-depth study of ensemble techniques paired with deep learning for fall detection on two different fall datasets.
- A demonstration that a stacked ensemble of deep learning models trained on small size datasets yields better classification accuracy than a model trained using a single deep learning network.
- A set of experiments which 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.

The remainder of this paper 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 main experimental results and detailed explanation of those results. Finally, in section 6 we present our conclusion and future work.

2 Related Work

We review two categories of research work that are relevant in this section. The first category is related to small sample deep learning and the second category is deep learning on streaming IoT data for fall detection or human activities detection.

Deep Learning for Human Activity Recognition (HAR) has shown superior results as it demonstrates superior classification results on raw data, and eliminates the need for human crafted features [26]. Deep Long Short Term Memory (LSTM) networks face practical performance limitations when employed in IoT applications: imbalanced dataset, small samples and data quality can significantly degrade the performance of HAR classifier. Training a deep neural networks requires a significant number of iterations before the optimal values of millions of parameters are found. If the network is trained using small dataset, large number of epochs can result in network over-fitting to that specific data sample. One way to overcome this real life limitations is to combine sets of diverse LSTM learners into an ensemble of classifiers as shown in [8]. 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.

A recent approach to solve the small sample learning problem in IoT is shown by the Cost-sensitive Deep Active Learning proposed in [4]. 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 Networks (DNN) which can be configured with any variant of deep models (CNN, RNN-LSTM, RNN-GRU). 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's experimental results demonstrated that the proposed scheme can reduce the amount of labelled samples by 33% to 80%. This system assumes that there is an abundant of unlabelled data which is not true in many IoT applications.

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. However, in digital health domain, transformation of sensor data via rotation, translation and scaling have not been widely adopted [17] due to the fact that 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 [19]. However, there is no real-world validation on the generated fall models. Other approaches for overcoming small dataset problem includes attempt to combine the 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[21, 27].

Application of deep learning to fall detection, in particular the use of Recurrent Neural Networks (RNN's) 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 [25], the authors describe an RNN architecture in which accelerometer signal is fed into 2 Long Short-Term Memory (LSTM) layers, and the output of these layers is passed through 2 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 [13], 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 [16]. The core of their neural network architecture consists of a fully connected layer, which processes the raw data, followed by 2 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 [22], which contains accelerometer data sampled at 200 Hz collected from a sensor attached to the belt buckle. In order to deal with a large imbalance in training data, of which ADL's 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 ADL's.

Our work differs primarily from these two papers that utilizes RNN in that we seek to optimize deep learning model 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 is sampled at a 200 Hz frequency obtained by a specialized equipment; this is a significantly higher than the frequency used by our smartwatch, which samples at 31.25 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 [20], 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 [28] 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 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

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 [14]. We reproduce some of the description here for ease of reference and comprehension.

Our smartwatch-based fall dataset was collected from fourteen volunteers each wearing a MS Band 2 watch[2]. These fourteen subjects were all of good health and were recruited to perform simulated falls and Activities of Daily Living (ADL's). Their ages ranged from 21-60, height ranged from 5 ft to 6.5 ft. and the weight from 100 lbs to 230 lbs. Each subject was told to wear the smartwatch on his/her left hand and performed a pre-determined set of ADL's consisting of: jogging, sitting down, throwing an object, and waving their hands. This initial set of ADL's were chosen based on the fact there are common activities that involved movement of the arms. These datasets were automatically labeled as "ADL" which is in our case we consider as "NotFall". 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 falls. Each subject repeated each type of fall 10 times. The sampling rate was set to 31.25 Hz.

We implemented a data collection service on the smartphone (Nexus 5X, 1.8 GHz, Hexa-core processors with 2G of RAM) that paired with the smartwatch to have a button that, when pressed, labels data as "Fall"

[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

Figure 1: Simulated fall data collection experiment

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.

However, the pressing of the button can introduce errors such as the button is being pressed too late, too early, or too long for a fall activity. To mitigate these errors, we post-processed the collected data to ensure that data points related to the critical phase of a fall were labeled as "Fall". This is done by implementing an R script that will automatically check that for each fall data file, the highest peak of acceleration, and data points before and after that point, were always labeled as "Fall". After this post processing of the collected data, we have a total of 528 fall data samples and 6573 samples of ADL data.

3.1.2 UniMiB SHAR Dataset

The second dataset that we used is the UniMiB SHAR [15]. This is an open source dataset that has 11,771 samples of accelerometer data collected from smartphones (Samsung Galaxy Nexus 19250) for both ADL's 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 ADL's 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 falls and 7579 ADL's in total in this dataset. We processed this dataset and have all the nine type of ADLs labeled as "NotFall". 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. In order 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 fall samples is 710 and ADLs is 486[3].

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 [10] 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 neural networks are biologically-inspired variants of the Multiple Layer Perceptrons (MLPs) [9]. 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 [24].

The use of deep convolutional neural networks showed a notable increase in the performance of detecting objects using highly challenging datasets [12]. The two most common implementations of deep neural networks are Convolutional Neural Networks (CNN) [12] and Recurrent Neural Networks (RNN) [6]. CNN is a type of feed-forward artificial neural network which takes fixed size inputs and generates fixed-size outputs. CNNs are ideal for images and video processing. RNNs, unlike feed-forward neural networks, can use their

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

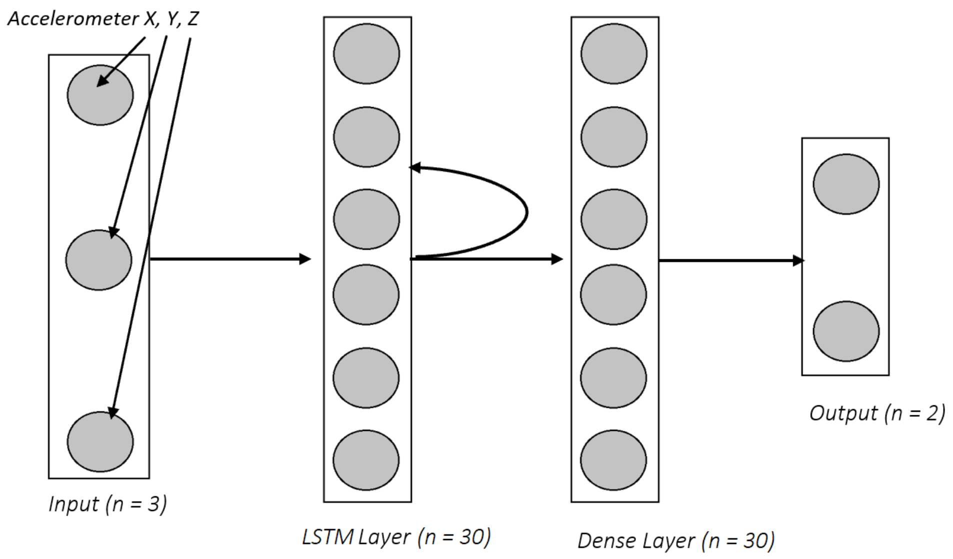


Figure 2: RNN model architecture

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: The model contains an input layer, two hidden layers, and an output layer. 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. The output is a 2-node softmax layer which 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, it will not capture the full scope of the activity. However, if we train it over too many steps, the network may take into account past accelerometer data that is not relevant. We settled on using 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 $\approx$1.12 seconds 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. 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. 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 10 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. Figure 3 outlines this schematic.

3.3 Ensemble Deep Learning

An ensemble is a system that consists of multiple smaller models. The goal of an ensemble is to combine smaller trained models to create a more accurate system [29]. In order to do this, algorithms have been designed that deal with the way these smaller models contribute to the

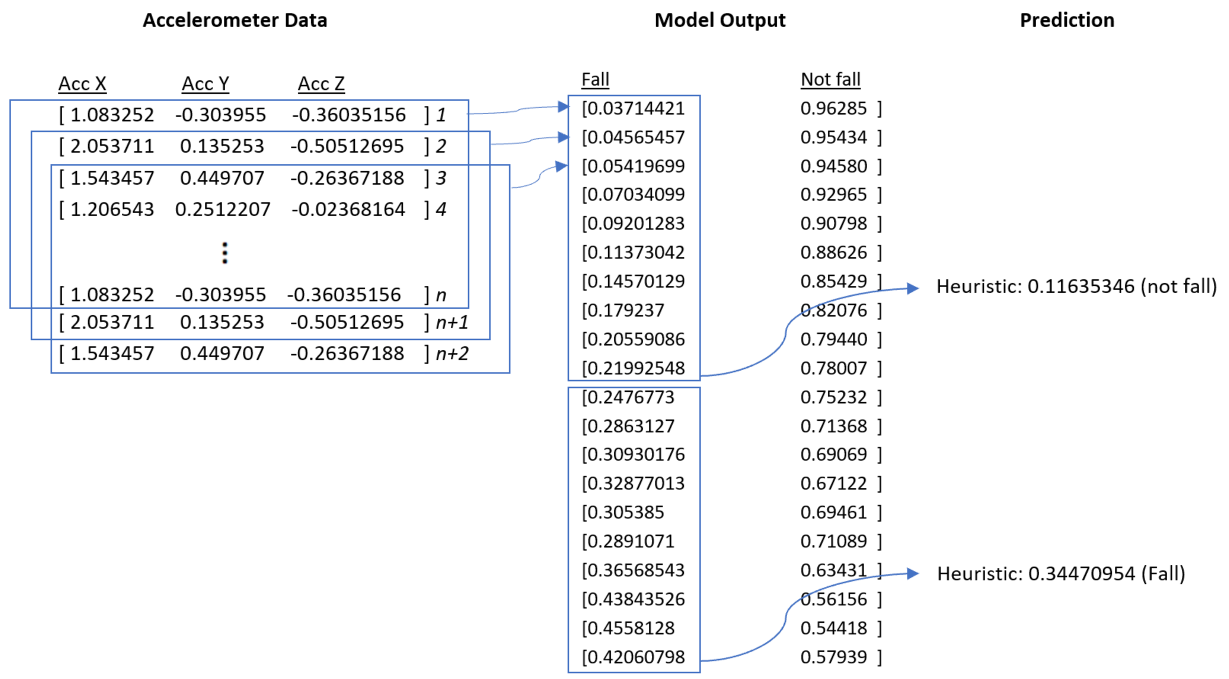


Figure 3: Prediction scheme for deep learning

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 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 in order to produce accurate models. The more complex (more neurons and layers) a neural network is the more data is required to train it. On the other hand, very shallow neural networks, although more easily trained, they cannot capture the full complexity of the problem (i.e. all possible signal patterns generated by different types of falls.). However, an ensemble of such 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.

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. 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

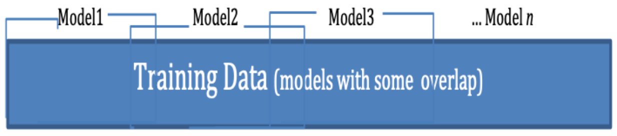


Figure 4: Assignment of training data to each LSTM model with overlap in consecutive order

ways, each model had a chance to contribute something slightly different than the others.

3.4 Boosting

Boosting is a well known ensemble technique used for more than two decades. Boosting has traditionally been used with simpler weak learners such as 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.

3.5 Stacking

Our second ensemble method is a variant of Stacking [29]. 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 [11]. Our experiments with stacking consisted of training multiple LSTMs, 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. 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 1000 rows of data and 10 models of 10% overlap is specified, each LSTM model will get 110 rows.

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

1: 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.}
2: models[0].start_row = 0
3: models[0].end_row = rows_per_model
   {We then calculate the data to assign to subsequent models, except the last one.}
4: for i = 1 to num_models - 1 do
5:     models[i].start = floor(rows_per_model * (1-overlap)) * i
6:     model[i].end = models[i].start + rows_per_model
7: end for {Data is assigned to the last model}
8: models[n-1].start_row = rounded(rows_per_model * (1-overlap)) * (num_models-1)
9: 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.
```

The algorithm for assigning the data is detailed in Algorithm 1. For our meta-algorithm, a list of weights will be initialized with random values. One weight for each model. These weights determine how important each model's prediction is. To determine a final value for each model's weight, we will adjust the weight list by obtaining each model's prediction over the entire training data and determining which values of weights can be multiplied by the model predictions to correctly fit the training data labels. 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.

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

The data is segmented into 'Fall' and 'NonFall' segments of equal size, as described in the Boosting section 3.4 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). An arbitrary data points predicted as 'Fall' and 'NonFall' at various time indices need to map to these segment intervals. When a model makes a 'Fall' prediction at time t_i, the segment containing t_i is marked as a predicted 'Fall'. A segment is considered to be predicted as 'NonFall' when all data points within that segment are classified as 'NonFall'.

Table 1: Evaluation metrics

Measure	Calculation Formula
Recall/Sensit.	$TP/(TP+FN)$
Precision	$TP/(TP+FP)$
Specificity	$TN/(TN+FP)$
Accuracy	$(TP+TN)/(TP+TN+FP+FN)$

5 Offline Experiments

We now present our experiments investigating the effectiveness of ensemble deep learning for fall detection. 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 dataset are simulated from subjects of varying ages, heights and weights.

Experiment Setting: All of our 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 presented the *PR (Precision vs. Recall)* curve for each experiment. 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, in order 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 put 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. The third type of experiments is related to using Stacking as the ensemble technique and demonstrated that Stacking using all available dataset can outperform the best single deep model. The last group of experiment is the validation of the best

Table 2: 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

performing ensemble stacking model using the UniMiB dataset.

5.1 Single Deep Learning Model

In this section, we show the results of various single LSTM network models on our SmartFall dataset. Each model was altered by 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 4 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 a sophisticated model. Our previous work on fall detection [14] indicates that a 20-neuron model is sufficient, and therefore 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 3 times. The average recall and precision over the 3 runs were recorded. Figure 5 shows the PR curve for this experiment and Table 2 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 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.

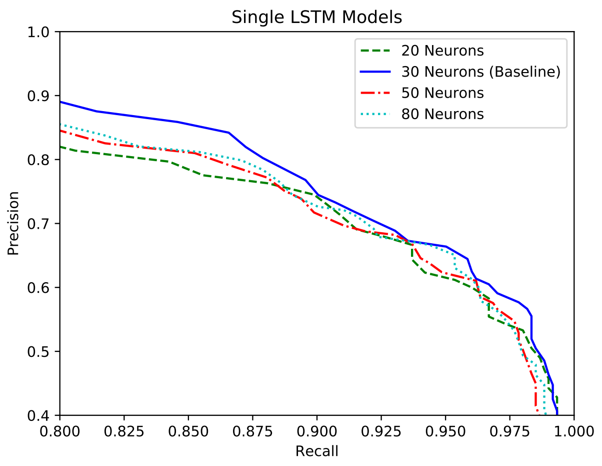


Figure 5: PR curve of single deep model by varying the number of neurons

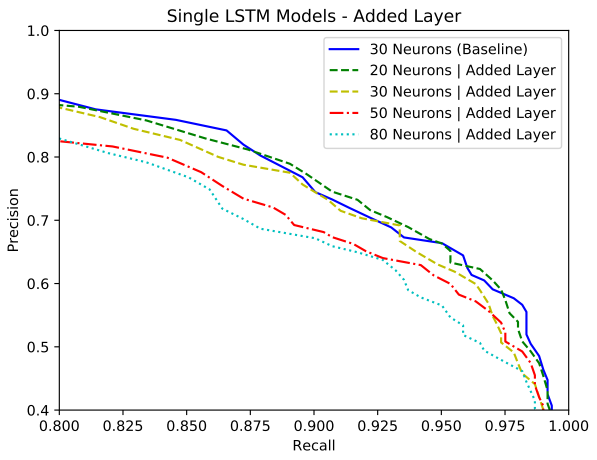


Figure 6: PR curve of single deep model with deep layers and varying number of neurons

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 6 shows the PR curve for this experiment and Table 3 shows the best results achieved by each model.

The results show that additional neurons or layers do no 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.

Table 3: 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 4: 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

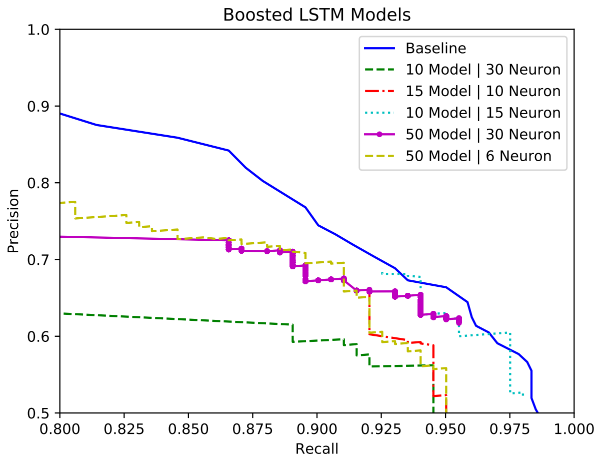


Figure 7: PR curve of boosted LSTM models

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 3.4 above. Each model in the ensemble is a one-layer LSTM with the same number of neurons. We will investigate how an ensemble of LSTM's 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 overall 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 5 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 models in the ensemble to get an idea of the effects of using different numbers of models. The results are shown in Table 4.

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 data. 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 7.

We see in Figure 7 that for almost every value of precision and recall, our baseline model performs best; however, there are 2 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 summary, we do not see any performance gains over the baseline model by using boosting.

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 readjusts 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 2 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 3.5. For the second method, we assign all the training data to every model. Comparing these methods shows if 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.

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 8, and Table 5 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 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

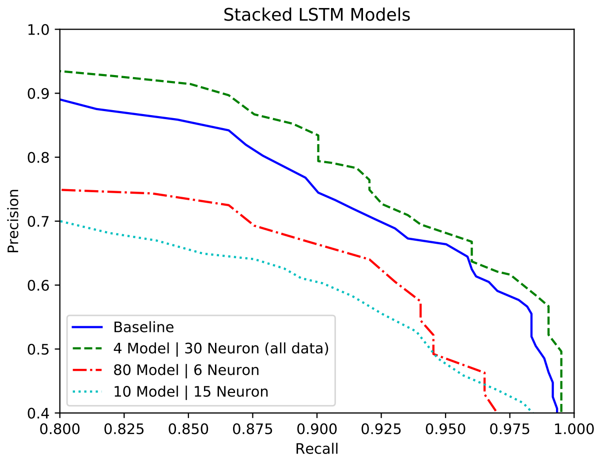


Figure 8: PR curve of stacked LSTM models trained on data subsets

Table 5: Stacking results on SmartFall dataset (Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons)

	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

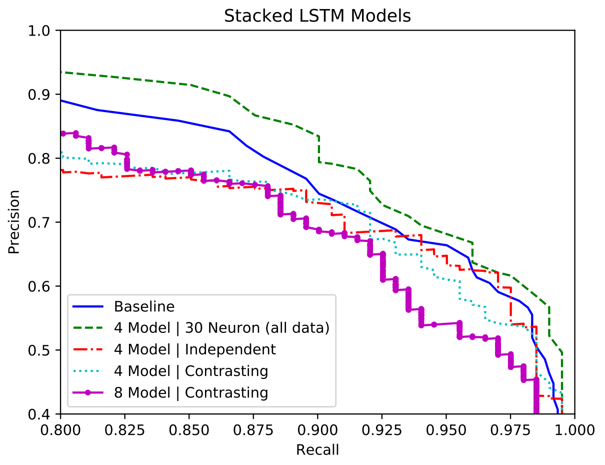


Figure 9: PR curve of stacked LSTM models trained on specific fall types

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 4 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 4 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 9, and Table 6 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.

Table 6: Stacking results on SmartFall dataset subsets (Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind = Independent, Cont. = Contrasting)

	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

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 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 4 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

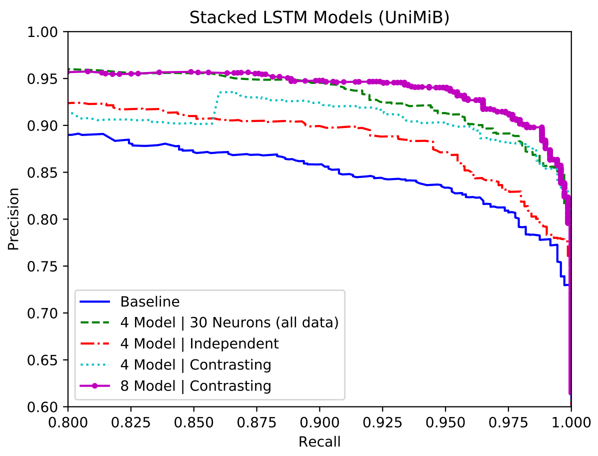


Figure 10: PR curve of stacked LSTM models on UniMiB

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 experiment. Figure 10 shows the PR curve for each model, and Table 7 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 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 Experimental Results

We carried out four groups of experimental studies. Our experiments show that a stacked ensemble of deep 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 deep learning model on small training datasets. Attempts to diversify the learning of our ensemble were made by altering the 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 overfitting. 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 8 provides a summary of the best results achieved by each approach.

Table 7: Stacking results on UniMiB dataset (Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind. = Independent, Cont. = Contrasting)

	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

Table 8: Results of the five best models on SmartFall and UniMiB datasets (Abbreviations: BL = Baseline, ST = Stacking, M = # of models, N = # of Neurons, Ind. = Independent, Cont. = Contrasting)

		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

6 Conclusion and Future Work

Deep neural networks have shown superior performance for fall detection as they eliminate the need for hand crafted features, as shown in our earlier work [14]. However, deep neural networks face practical challenges when used for 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 paper, we aim to mitigate the scarcity of training samples in fall detection IoT application by combining RNN with ensemble techniques such as boosting and stacking. We conducted an extensive set of experiments on two different fall datasets and concluded that the stacking ensemble technique combined with RNN can outperform the single deep RNN model. Boosted LSTM models does 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.

Our immediate future work is to deploy the best performing stacking ensemble deep model (4 Models, 30 Neurons trained on all dataset) on our Fall Detection App and recruit some volunteers to test the performance of the model in real-time. Each volunteer will be asked to wear the watch for a number of days and to perform a series of different type of falls and ADLs each day. Testing in this way lets us evaluate the model's capabilities in a true online situation as well as seeing how the ensemble model performs on specific kinds of falls and ADL's.

Other future work includes the investigation of other small sample learning strategies. This includes data augmentation or 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 Fall Detection 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.

Acknowledgement

We thank the National Science Foundation for funding the research under the Research Experiences for Undergraduates Program (CNS-1358939) at Texas State University to perform this piece of work and the infrastructure provided by a NSF-CRI 1305302 award. We also want to thank Texas State students, in particular Shaun Coyne, who helped with the data collection.

References

[1] F. Adib, H. Mao, Z. Kabelac, D. Katabi, and R. C. Miller, "Smart homes that monitor breathing and heart rate," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 837–846.

[2] apple.com, "Apple watch series 4," http://www.apple.com/apple-watch-series-4/activity/, accessed: 2019-04-18.

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

[4] X. Chen, J. Ji, T. Ji, and P. Li, "Cost-sensitive deep active learning for epileptic seizure detection," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, ser. BCB '18. New York, NY, USA: ACM, 2018, pp. 226–235. [Online]. Available: http://doi.acm.org/10.1145/3233547.3233566

[5] cs.txstate.edu, "Fall data collected using microsoft band smartwatch," http://www.cs.txstate.edu/~hn12/data/SmartFallDataSet, accessed: 2019-04-18.

[6] S. Grossberg, "Recurrent neural networks," *Scholarpedia*, vol. 8, no. 2, p. 1888, 2013.

[7] Y. Guan and T. Plötz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, pp. 11:1–11:28, Jun. 2017.

[8] Y. Guan and T. Plötz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, pp. 11:1–11:28, Jun. 2017.

[9] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[10] A. Janecek, W. Gansterer, M. Demel, and G. Ecker, "On the relationship between feature selection and classification accuracy," in *New Challenges for Feature Selection in Data Mining and Knowledge Discovery*, 2008, pp. 90–105.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer methods and programs in biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[14] T. R. Mauldin, M. E. Canby, V. Metsis, A. H. H. Ngu, and C. C. Rivera, "Smartfall: A smartwatch-based fall detection system using deep learning," *Sensors*, vol. 18, no. 10, 2018.

[15] D. Micucci, M. Mobilio, and P. Napoletano, "Unimib shar: A dataset for human activity recognition using acceleration data from smartphones," *Applied Sciences*, vol. 7, no. 10, 2017.

[16] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, "Online fall detection using recurrent neural networks," *arXiv preprint arXiv:1804.04976*, 2018.

[17] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233 – 261, 2018.

[18] rightminder.com, "Rightminder - fall detection for android smartwatches and android phones," http://www.rightminder.com, accessed: 2018-08-26.

[19] G. L. Santos, P. T. Endo, K. H. d. C. Monteiro, E. d. S. Rocha, I. Silva, and T. Lynn, "Accelerometer-based human fall detection using convolutional neural networks," *Sensors*, vol. 19, no. 7, 2019.

[20] J. A. Santoyo-Ramón, E. Casilari, and J. M. Cano-García, "Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection with supervised learning," *Sensors*, vol. 18, no. 4, 2018.

[21] S. Shin and W. Sung, "Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 2274–2277.

[22] A. Sucerquia, J. D. López, and J. F. Vargas-Bonilla, "Sisfall: A fall and movement dataset," *Sensors*, vol. 17, no. 1, p. 198, 2017.

[23] C. Tacconi, S. Mellone, and L. Chiari, "Smartphone-based applications for investigating falls and mobility," in *2011 5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops*, May 2011, pp. 258–261.

[24] S. Tang and Y. Yuan, "Object detection based on convolutional neural network," 2015.

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

[26] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3 – 11, 2019, deep Learning for Pattern Recognition.

[27] S. Yu, L. Qin, and Q. Yin, "A c-lstm neural network for human activity recognition using wearables," in *2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*, Sep. 2018, pp. 1–6.

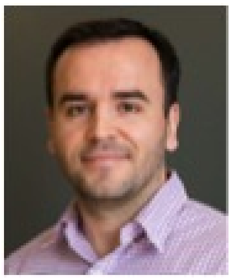
[28] A. T. Özdemir, "An analysis on sensor locations of the human body for wearable fall detection devices: Principles and practice," *Sensors*, vol. 16, no. 8, 2016.

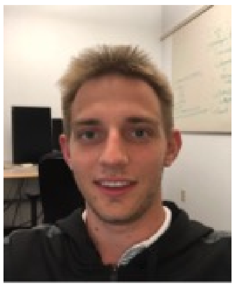
[29] C. Zhang and Y. Ma, *Ensemble machine learning: methods and applications*. Springer, 2012.

Author Biographies

Taylor Mauldin graduated from Texas State University in 2019 with a B.S. in Computer Science. He is set to begin employment as a software engineer for Silicon Labs in June of 2019, working with crystal oscillators, clock generators, clock buffers, and jitter attenuator and PCI Express clock generators and PCI Express buffers products. His interests include machine learning and deep learning, smart healthcare, and signal processing

Anne H.H. Ngu is currently a Professor with the Department of Computer Science at Texas State University. From 1992-2000, she worked as a Senior Lecturer in the School of Computer Science and Engineering, University of New South Wales, Australia. She had held the research scientist/scholar position with Telcordia Technologies, Lawrence Livermore National Laboratory, Microelectonics and Computer Technology (MCC), University of California, Berkeley,Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia and the Tilburg University, The Netherlands. Her main research interests are in Large-scale service and information discovery and integration, Internet of Things, for Smart-health, Scientific workflows, Databases and Software Engineering. She was the recipient of the 2013 NCWIT Undergraduate Research Mentoring Award.


Vangelis Metsis is an Assistant Professor at the Department of Computer Science at Texas State University. He received his Bachelor of Science degree in Computer Science in 2005, from the Department of Informatics of Athens University of Economics and Business in Greece, and his Doctoral degree in 2011 from the Department of Computer Science and Engineering of The University of Texas at Arlington. Dr. Metsis research interests span the areas of Machine Learning, Data Mining and Computer Vision with special focus on applications of Smart Health and Wellbeing, and Pervasive Computing.


Marc E. Canby graduated from Rice University in Spring 2018 with a B.S. in Computer Science, B.A. in Mathematics and Classical Studies, and a Master?s of Statistics. He is currently employed by Microsoft as a software engineer in the telemetry analytics space, and he will be pursuing a Ph.D. in Computer Science at the University of Illinois-Urbana Champaign beginning in August 2019. His research interests are in data mining, natural language processing, and information extraction.

Jelena Tešić is an Assistant Professor at the Department of Computer Science, Texas State University. Dr. Tešić was a senior research scientist at Mayachitra Inc from 2009 to 2017 where she led research projects on scaling multimedia analytics for large image and video repositories within NAVAIR STTR/SBIR, DARPA VIRAT, and IARPA FINDER projects. Prior, Dr. Tešić spent 5 years at IBM T.J. Watson Research Center, Yorktown, NY where she contributed to the top performance of the IBM team in NIST TRECVID benchmark. Dr. Tešić received the Ph.D. in Electrical and Computer Engineering from University of California, Santa Barbara, in 2004. Dr. Tešić research interest is unstructured data science, i.e. machine learning, graph theory, and computer vision advances for large noisy unstructured datasets with focus on DoD and health applications. She has authored over 30 scientific papers, and currently holds 6 U.S. patents.