WearBreathing: Real World Respiratory Rate Monitoring Using Smartwatches

DANIYAL LIAQAT, University of Toronto, Canada and Vector Institute for Artificial Intelligence
MOHAMED ABDALLA, University of Toronto and Vector Institute for Artificial Intelligence
PEGAH ABED-ESFAHANI, University of Toronto
MOSHE GABEL, University of Toronto
TATIANA SON, University Health Network
ROBERT WU, University Health Network and University of Toronto
ANDREA GERSHON, Sunnybrook Health Sciences Centre and University of Toronto
FRANK RUDZICZ, St Michael's Hospital, University of Toronto, Vector Institute for Artificial Intelligence, and Surgical Safety Technologies
EYAL DE LARA, University of Toronto

Respiratory rate is a vital physiological signal that may be useful for a multitude of clinical applications, especially if measured in the wild rather than controlled settings. In-the-wild respiratory rate monitoring is currently done using dedicated chest band sensors, but these devices are specialized, expensive and cumbersome to wear day after day. While recent works have proposed using smartwatch based accelerometer and gyroscope data for respiratory rate monitoring, current methods are unreliable and inaccurate in the presence of motion and have therefore only been applied in controlled or low-motion settings. Thus, measuring respiratory rate in the wild remains a challenge.

We observe that for many applications, having fewer accurate readings is better than having more, less accurate readings. Based on this, we develop WearBreathing, a novel system for respiratory rate monitoring. WearBreathing consists of a machine learning based filter that detects and rejects sensor data that are not suitable for respiratory rate extraction and a convolutional neural network model for extracting respiratory rate from accelerometer and gyroscope data. Using a diverse, out-of-the-lab dataset that we collected, we show that WearBreathing has a 2.5 to 5.8 times lower mean absolute error (MAE) than existing approaches. We show that WearBreathing is tunable and by changing a single threshold value, it can, for example, deliver a reading every 50 seconds with a MAE of 2.05 breaths/min or a reading every 5 minutes with an MAE of 1.09 breaths/min. Finally, we evaluate power consumption and find that with some power saving measures, WearBreathing can run on a smartwatch while providing a full day's worth of battery life.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; **Empirical studies in ubiquitous and mobile computing**; • **Applied computing → Consumer health**;

ACM Reference Format:
Daniyal Liaqat, Mohamed Abdalla, Pegah Abed-Esfahani, Moshe Gabel, Tatiana Son, Robert Wu, Andrea Gershon, Frank Rudzicz, and Eyal de Lara. 2019. WearBreathing: Real World Respiratory Rate Monitoring Using Smartwatches. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article 56 (June 2019), 22 pages. https://doi.org/10.1145/3328927

Corresponding author: Daniyal Liaqat, University of Toronto, Department of Computer Science, 40 St George St., Toronto, ON M5S 2E4, Canada, dliaqat[at]cs.toronto.edu.

2474-9567/2019/6-ART56 $15.00
https://doi.org/10.1145/3328927

1 INTRODUCTION

Continuous monitoring of physiological signals has the potential to revolutionize personalized health care. Respiratory rate is one signal that may be useful for a multitude of clinical applications. Higher respiratory rates are a strong predictor of cardiac arrest [10] and have been linked to negative outcomes in hospital wards and emergency rooms [5]. For example, Fieselmann *et al.* [10] reported that a respiratory rate over 27 breaths/minute was a significant predictor of cardiac arrest. In developing an early warning measure for cardiac arrest, Goldhill *et. al* [11] scored respiratory rates between 9 and 30 in increments of 5 (e.g. $9-14$ or $15-20$), and reported that 21% of ward patients with a respiratory rate between 25 and 29 died in hospital. Cretikos *et al.* [5] recommend that patients with a respiratory rate greater than 24 breaths/minute should be monitored closely and those with respiratory rate greater than 27 should receive immediate medical review.

While these studies highlight the clinical importance of respiratory rate, the implications of continuous respiratory rate monitoring in uncontrolled settings have not been studied because there are no existing devices to facilitate such studies. Though devices such as chest bands that can be worn around the torso to measure respiratory rate have been available for quite a while, they are burdensome to use day after day and the effort of having to use an extra device will dissuade all but the most determined users. Monitoring respiratory rate outside labs and hospitals, on larger populations and over longer periods of time could lead to the development of new detection, monitoring and prediction systems for various conditions. These include not only respiratory conditions such as asthma and chronic obstructive pulmonary disease (COPD), but also non-respiratory conditions such as cardiac arrest, heart failure, panic attacks and anxiety disorders – all of which have shortness of breath as a potential symptom[1]. However, any research in the development of such systems must first devise a way to reliably detect respiratory rate in real world environments.

Our aim is to make respiratory rate monitoring as effortless as possible. This means we need inexpensive devices that are readily available, easy to use and contain sensors, as well as algorithms that can reliably measure the respiratory rate signal. Recent works [14, 16, 32] have identified smartwatches as a strong candidate for respiratory rate monitoring. Smartwatches are relatively inexpensive off-the-shelf devices that contain an assortment of sensors. They serve multiple purposes, making them more appealing for users to wear day after day. Respiratory rate monitoring will likely not even be the primary purpose of using such a smartwatch for most users, just an additional benefit. Furthermore, smartwatches are programmable, commercially-supported software platforms which means that turning a smartwatch into a respiratory rate monitor could be as simple as installing an application.

Existing works that use smartwatches for respiratory rate monitoring [14, 16, 32] make use of the accelerometer and/or gyroscope sensors. Breathing produces subtle, periodic motions that can be measured by the Inertial Measurement Unit (IMU) in the smartwatch. The gist of these approaches is to use the periodic nature of breathing to detect a signal that falls within some frequency that corresponds to the expected breathing rate (e.g. 9-30 breaths/min). However, the biggest challenge to these approaches is that they are highly susceptible to motion artifacts. Existing systems acknowledge this as a limitation and conduct their experiments in controlled or low-motion settings. Our goal is to make respiratory rate monitoring possible in everyday environments and during daily tasks.

Our key insight is that many applications benefit more from a small amount of accurate data rather than a large amount of inaccurate data. For example, an application that monitors the progression of a respiratory disease over time does not necessarily need to know the exact respiratory rate at any given second, but rather needs to keep track of trends over weeks or months. For such applications, sensors need only be accurate some of the time. The challenge then is identifying which sensor readings are accurate.

[1] https://www.mayoclinic.org/symptoms/shortness-of-breath/basics/causes/sym-20050890

Based on this insight, we develop WearBreathing, a two-step system for respiratory rate monitoring. In the first step, a random forest (RF) model acts as a filter and rejects data that will result in an inaccurate respiratory rate reading. Sensor readings that pass the filter are then fed into the second step which uses a Convolutional Neural Network (CNN) model to extract respiratory rate from accelerometer and gyroscope data. Though previous works have also used filters to reject some data [14, 16, 32], our random forest filter is fundamentally different. Previous filters are inflexible: they assume that excessive motion is the sole cause for inaccurate respiratory rate readings, and reject readings if the level of motion surpasses some threshold. Our approach does not make any assumptions about what causes unreliable readings and instead learns what causes the extractor in the second step to be inaccurate.

This idea of a learned filter is the key difference between our work and previous works. In a more abstract sense, previous works only filter the source signal based on the amount of noise present. Our approach takes into account both the original signal and its interaction with the extraction algorithm. Our proposed approach, which includes a method for training the filter to learn the interaction between an extractor and the source signal, could be applied to other sensing tasks and potentially entirely different domains.

We collected data in a one hour long semi-controlled and a three hour long uncontrolled setting, from two groups of participants. By using data from two different groups of participants we highlight the generalizability of our approach. The first group consists of younger, healthy participants and the second of older participants suffering from a lung disease (chronic obstructive pulmonary disease, or COPD). We evaluate WearBreathing on our collected dataset and show that it is able to achieve a mean absolute error (MAE) of 2.05 breaths/min while delivering a respiratory rate reading every 50 seconds, which is a 3.6 times lower MAE than previous work [16, 32]. Moreover, unlike existing approaches, WearBreathing is highly tunable and can be easily configured to trade off reading frequency for accuracy. For example, WearBreathing can deliver a reading on average every 15 seconds with a MAE of 2.73, every minute with MAE 2.17 or every 5 minutes with MAE 1.09. This level of flexibility makes WearBreathing a good match for a wide range of applications. Some applications may require more readings and are willing to accept a lower accuracy while others may be willing to accept less frequent readings, but demand a higher accuracy. This tunability is possible because our random forest filter learns to directly control for accuracy. In previous work, the relationship between what the filter controls and accuracy is not intuitive, making this level of tunability difficult or impossible.

Using a combination of data traces and simulation, we also explore the energy consumption of WearBreathing when deployed on a real smartwatch. We find that under ideal conditions, a duty cycling scheme can provide between 24-42 hours of battery life. Alternatively, applications that do not wish to use duty cycling can opt to use the smartwatch as a continuous recording device and process the data offline, which will provide over 18 hours of battery life.

Our contributions are as follows:

- A demonstration that in wild settings, identifying when data is accurate is an important problem and that existing filters are not well suited to this task.
- A two-step filter/extractor system in which the filter learns the interaction between the extractor and input data.
- A highly tunable random forest filter that allows flexible trade-off between frequency and accuracy by changing a single, easy-to-understand threshold value.
- A novel method of respiratory rate extraction using a CNN that is more accurate than existing approaches.
- The combination of our random forest filter and CNN extractor which, for the first time, enables out-of-the lab respiratory rate monitoring using a smartwatch.
- An evaluation of out-of-the-lab respiratory rate monitoring on two different populations.
- An evaluation of WearBreathing showing that it can be run on a smartwatch with reasonable battery life.

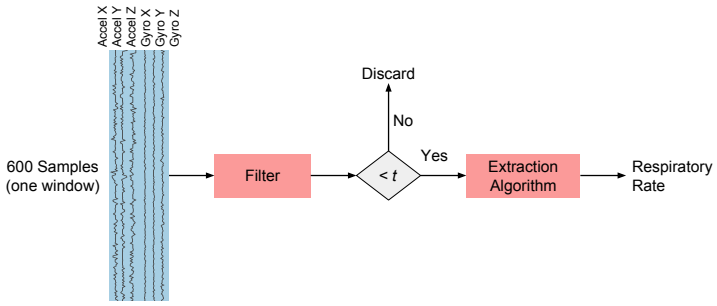

Fig. 1. High level system diagram of WearBreathing.

The rest of this paper is organized as follows. In Section 2 we describe our system for filtering data and extracting respiratory rate. In Section 3, we describe our dataset, existing methods and research questions. Next, in Section 4, we evaluate the performance of WearBreathing and existing methods. This is followed by a review of related works in Section 5 and a discussion of real-world deployment issues, avenues for future research and interesting overlaps with other research fields in Section 6.

2 SYSTEM DESIGN

As mentioned previously, not all data from the accelerometer and gyroscope will result in a reliable respiratory rate reading. The problem we solve is identifying when the respiratory rate reading is reliable. We do this by creating a system consisting of an extractor and a filter. The extractor takes sensor data as input and produces an estimated respiratory rate. The filter takes the same sensor data as input, but instead of predicting respiratory rate, it predicts the error the extractor will have on this input. To determine if a respiratory rate is produced, the predicted error from the filter is compared to a user defined threshold. If the predicted error is below the threshold, the sensor data is passed to the extractor which produces the estimated respiratory rate. This process is illustrated in Figure 1.

Although in our final system the filter is applied before the extractor, during development and training we need to build the extractor first. Therefore, in this section, we first describe our extractor and then the filter. Because of this dependency between the extractor and filter, there are some subtleties in how we train our model, which we describe at the end of this section.

2.1 Extractor

We develop a novel method for extracting respiratory rate from accelerometer and gyroscope signals using a CNN. We employ a CNN model to extract respiratory rate because CNNs excel at detecting patterns in spatial or temporal sequences of multi-channel data and have been used extensively for time series data [15, 17, 34, 35]. While typically, recurrent networks such as LSTMs have been used for time series data, some work has shown that in some cases, simple CNNs can outperform LSTMs. Additionally, CNNs tend to be less computationally expensive than recurrent networks[2], which is critical for a model designed to run on a resource constrained device such as a smartwatch.

In our case, we want the network to identify the breathing signal, which has a distinct pattern, in time series IMU data where the axes of the accelerometer and gyroscope are represented as channels.

[2]https://github.com/baidu-research/DeepBench

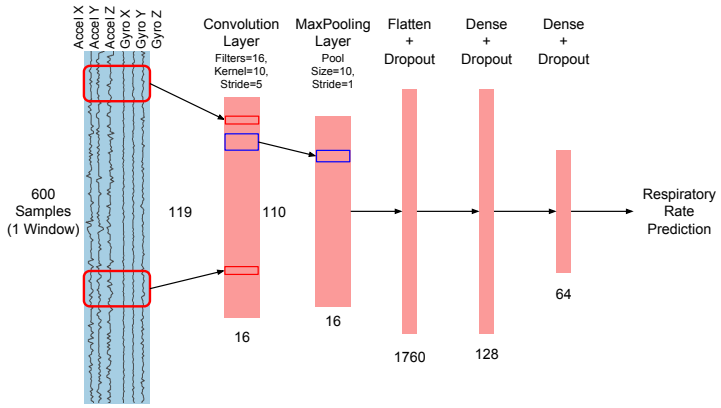

Fig. 2. CNN extractor architecture showing how input data is transformed to produce the estimated respiratory rate.

The architecture for our CNN is shown in Figure 2. The CNN operates on 30 second windows of 6 axes of raw accelerometer and gyroscope data. We use 30 second windows because it is a commonly used window length in existing respiratory rate monitoring work [28, 32]. We also experimented with using the Fourier transform and derivative of each axis as input to the CNN; however, we found that using raw data performed better. We use a shallow CNN composed of a single convolutional layer with a rectified linear unit activation, 16 hidden units, a kernel size of 5 and stride size of 1. Following the convolutional layer is a max pooling layer with a pool-size of 10 and stride of 1. The output of the pooling layer is connected to a dense layer with 128 hidden units that feeds into another dense layer with 64 hidden units. Both dense layers use a rectified linear activation and a 0.2 dropout after each. The second dense layer connects to a single predictive node, again with a rectified linear activation. The network is optimized using *adaptive moment estimation* [22] (Adam), with mean absolute error as the loss function. The network is implemented using the Keras framework [3] and for any unspecified hyper-parameters the Keras default values were used. Treating this as a regression task, our network is trained to predict respiratory rate values. The respiratory rate labels used to train this CNN are obtained from a chest band. Our procedure for collecting labeled data is explained in more detail in Section 3.1.

2.2 Filter

The goal of the filter is to predict the error in the extractor. Previous works operated on the idea that respiratory rate extraction would be inaccurate in the presence of motion. Therefore, they employed simple filters that assumed the amount of motion is directly related to the error. An example of such a filter is one that looks at the average vector magnitude of the x, y and z axes of the previous n accelerometer readings (Equation 1). For our experiments, we use a window size of 30 seconds and sampling rate of 20 Hz, which means that $n = 600$. So for a given window of 600 samples, if the value resulting from Equation 1 exceeds some predetermined threshold t, then this window will not be passed on to the extractor. As we show later, simple filters such as these do not perform well and are not intuitive.

$$accept(window) = \frac{\sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2 + z_i^2}}{n} < t \tag{1}$$

Our random forest regression-based filter is a novel approach to building these kinds of filters. Instead of assuming that motion is the only source of error and estimating the amount of motion using a formula, we

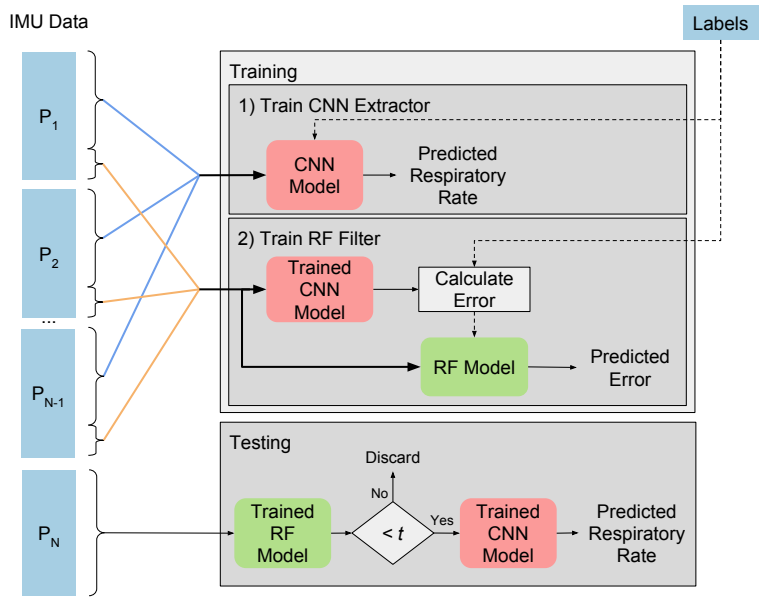

Fig. 3. Training scheme illustrating how we use leave-one-out cross validation and separate CNN and RF training data to improve generalizability.

train a classifier to learn what causes errors; that is, our filter learns the interaction between the input data and extractor. This filter has the same interface as the simple filter in that it takes 600 readings as input from each axis of the 6-axis accelerometer/gyroscope data and outputs a single value that is then compared to a threshold. However, unlike the simple filter, which estimates the amount of motion in a window, our filter predicts the error we can expect if we were to apply our CNN extractor on a given window. As we show later, this results in a filter that is easier to tune since the threshold directly controls error in respiratory rates, which users care about and understand, rather than the motion within a window, which is harder to reason about.

Our random forest takes as input a summary of the accelerometer and gyroscope data. This summary is obtained by first computing 16 aggregate measures for each axis. These measures are the mean, median, minimum, maximum, kurtosis, skew and 10^{th}, 20^{th}, ..., 90^{th} percentiles. This reduces the dimension of each window from 6 vectors of length 600 to 6 vectors of length 16. The resulting 6 vectors are then concatenated into a single vector of length 96. Principal component analysis (PCA) is used to further reduce the vector's dimension to an empirically determined length of 20. The output of PCA is then used as input to a random forest regression model. The PCA projection matrix is computed using the training data only. Our general idea still works without PCA; however, we found that using PCA slightly improved our results.

For labels, we use the error of our CNN-based extractor. That is, for each input, we extract respiratory rate using our trained CNN and take the absolute difference between the predicted respiratory rate and the ground truth respiratory rate as the error. Our random forest model's task is to predict this error.

2.3 Training Scheme

Our random forest and CNN models both require training and testing. To prevent over-fitting and ensure that our results are generalizable, we use a leave-one-out cross validation scheme. That is, for each participant x, we

Fig. 4. Zephyr BioHarness 3.0 used as a ground truth respiratory rate monitor.

train a model using data from all participants except x. This means that when we evaluate a trained model on a participant, the model has never seen data from that participant during training. It also means that all the results we present are averages across participants.

We also have to be cautious about training the random forest regressor because its labels are dependent on output of the CNN, which could potentially bias the random forest. For example, if a window that's used to train the random forest was previously used to train the CNN, we would expect that the CNN is more accurate in extracting respiratory rate from this window. Therefore, the error that the random forest is attempting to predict would not be a true representation of the error we would expect if the CNN was predicting on unseen data. To avoid this, we hold out a small amount of data (10%) from each participant in the CNN's training set. These data are not used to train the CNN. Once the CNN has been trained, the hold out data is passed through the CNN model and a respiratory rate is predicted. We compute the absolute error for these predictions and use these errors as labels to train the random forest filter. This scheme is illustrated in Figure 3.

3 EXPERIMENTAL SETUP

In this section, we first describe our dataset and how it was collected. Next, we summarize two existing works against which we compare WearBreathing. Finally, we present the research questions we explore in our evaluation of WearBreathing.

3.1 Data Collection

To test and evaluate respiratory rate monitoring in the wild, we collect and make use of a dataset that contains data from a smartwatch and a ground truth device. Collection of this dataset was approved by both the Institutional Review Board of the hospital where we collected data as well as our academic institution.

Our dataset contains data from 14 participants, 7 of which were healthy and 7 had chronic lung disease (3 female and 4 male in both groups, healthy group mean age: 28.4 years, chronic lung disease group mean age: 69.3 years). There is a significant age difference in these groups because COPD occurs most commonly in older adults. All participants were asked to wear an LG Urbane smartwatch on their non-dominant hand. The watch was running Android Wear and a data collection application we developed to collect accelerometer and gyroscope data. Data from both the accelerometer and gyroscope are recorded at 20 Hz because, as shown by BioWatch [16], this is a sufficient sampling frequency to capture the respiratory rate signal which has a frequency between 0.13 Hz and 0.66 Hz. This data is transmitted over Bluetooth to a smartphone. The smartphone simply acts as a relay and uploads the data to a remote server. All analysis and model training is done offline on the remote server. Although our system can support real-time analysis on a mobile device, the purpose of this data collection was

Table 1. Summary of our dataset showing duration of activities along with the mean and range (1^{st} and 99^{th} percentile) of respiratory rate (breaths/min) for each activity according to the ground truth BioHarness.

Activity	Duration	Healthy		COPD	
		Mean (SD)	Range	Mean (SD)	Range
6MWT	6 min	20.9 (4.0)	14 - 29	24.9 (4.5)	13 - 34
Sitting	4 min	20.6 (5.2)	10 - 29	17.7 (3.0)	10 - 24
Walking	4 min	19.7 (4.6)	13 - 30	25.4 (5.4)	16 - 37
Lying	4 min	17.6 (2.5)	14 - 26	15.5 (5.7)	6 - 23
Standing	4 min	15.2 (5.0)	4 - 22	18.1 (4.0)	12 - 23
Eating	3 min	14.9 (2.7)	9 - 20	17.7 (3.7)	12 - 24
Brushing	2 min	16.0 (3.7)	9 - 25	18.5 (4.4)	11 - 24
Uncontrolled	3 hours	17.4 (4.2)	6 - 28	19.9 (4.9)	10 - 29

to collect data that can be used to train our RF filter and CNN extractor. Later, in Section 4.3, we deploy our trained models on real devices to estimate WearBreathing's impact on battery life.

To obtain ground truth data, participants also wore a Zephyr BioHarness 3.0[3], shown in Figure 4, which uses a capacitive pressure sensor to measure expansion and contraction of the chest. The BioHarness has been validated in several studies [13, 20, 21] to be accurate under the conditions we set in our data collection study. It has also been validated for participants with COPD [30]. For each respiratory rate reading from the BioHarness, we consider the preceding 30 seconds of accelerometer and gyroscope data as a window.

The first portion of the study took place in a lab. Participants were asked to complete specific activities in the lab while wearing both the smartwatch and BioHarness. We call this portion semi-controlled because while participants were asked to perform specific activities, there were no restrictions on how to perform the activities. For example, participants were not told how to place or move their arms. They had the freedom to move their arms however they wanted during the study.

The activities performed during the semi-controlled portion, listed in Table 1, are selected because we expect that they are the most frequently occurring activities in daily living. The six-minute walk test (6MWT) was included because it represents the fastest participants are likely to walk in their daily lives and because it is a commonly used test for respiratory conditions [9]. Participants were allowed to take breaks between activities, so the total duration of the semi-controlled portion ranged from 40 to 75 minutes.

The second portion of the experiment was completely uncontrolled. Participants still wore the smartwatch and BioHarness, but were free to go about their day outside the lab. After three hours, the participants could take off the smartwatch and BioHarness.

During our data collection study, we collected over 53 hours of data from our 14 participants resulting in over 144,800 individual respiratory rate measurements from the BioHarness. The mean respiratory rate according to the BioHarness across our entire dataset is 18.31 breaths/min with a standard deviation of 4.72 and a range of 7-29 breaths/min. The range shows the 1^{st} and 99^{th} percentile of observed values. In Table 1, we break down the mean, standard deviation and range of the respiratory rate by group and activity.

3.2 Existing Approaches

To compare WearBreathing, we implement the respiratory rate extraction methods explained in BioWatch [16] and SleepMonitor [32].

[3]https://www.zephyranywhere.com

BioWatch [16] describes an extractor that first performs noise removal on gyroscope data by applying an averaging filter and then a band-pass Butterworth filter of order two with cut-off frequencies of 4 and 11Hz. Using this noise-removed data, the BioWatch extractor obtains three respiratory rate predictions by computing an FFT on each of the three axes and selecting the frequency with the highest amplitude between 0.13 Hz and 0.66 Hz (i.e., between 7.8 to 40 breaths/min). To determine which of the three respiratory rate predictions to use, it again look at the FFT amplitude and select the one from the axis with the greatest amplitude. The BioWatch filter takes the vector magnitude of the derivative of the accelerometer data.

To validate our implementation of the BioWatch algorithm, we collected a small sample of data from two of the authors using the same method described in the BioWatch paper (i.e., sitting very still) and were able to achieve similar results (MAE < 1). Like BioWatch, our study includes sitting, standing and lying down. However, we give very little instruction to participants on how to perform these activities. Therefore, even as participants were sitting or lying down, they were doing so naturally and were not trying to be still. This is evidenced by the fact that in the BioWatch paper, the filter preserved 85.87% of windows when using a threshold of 0.15 [16], which was the *maximum* value the BioWatch authors observed from their filter during their in-lab study. However, when we apply that same derivative magnitude filter on our dataset, we see an *average* value of 3.56; if we were to discard windows from our data where the filter value is greater than the original 0.15 threshold, we would have accepted only 5.14% of windows. This highlights the drastic difference in the amount of motion between data collected out-of-the lab, as in our study, compared to data collected in controlled lab settings, as in previous studies.

We also replicated the algorithm described in SleepMonitor [32]. The SleepMonitor extractor uses a total variational filter (TV filter) [1, 18] to remove both high and low-frequency noise. After noise removal, respiratory rate estimation is performed on each accelerometer axis using an FFT in a manner similar to BioWatch. However, unlike BioWatch which selects respiratory rate from a single axis, SleepMonior merges the respiratory rate from all three axes using a Kalman filter. The filter to reject widows described in SleepMonitor [32] looks at the proportion of accelerometer readings in a window with a vector magnitude greater than $10m/s^2$. The threshold used in SleepMonitor is 5, so that any window where more than 5% of samples have a vector magnitude greater than $10m/s^2$ is rejected.

3.3 Research Questions

The two metrics we consider in our analysis are accuracy (measured as MAE) and frequency (average time between readings). Using these metrics, we ask the following questions to evaluate WearBreathing:

- How does WearBreathing perform and compare to existing methods?
- How does activity affect performance?
- Does WearBreathing work on both the healthy group and the chronic lung disease group?
- Is there agreement between WearBreathing and the chest band ground truth?
- How tunable is WearBreathing?
- Is it feasible to run WearBreathing on a smartwatch?
- What is the battery impact of running WearBreathing on a smartwatch?

4 EVALUATION

In this section, we present an evaluation of WearBreathing on a diverse dataset that includes data from younger, healthy individuals and those with COPD while performing a wide range of activities. Having this diverse dataset allows us to analyze how the participant's activity and group affects WearBreathing's performance. Throughout the entire study, participants wore a smartwatch and a chest band for ground truth data.

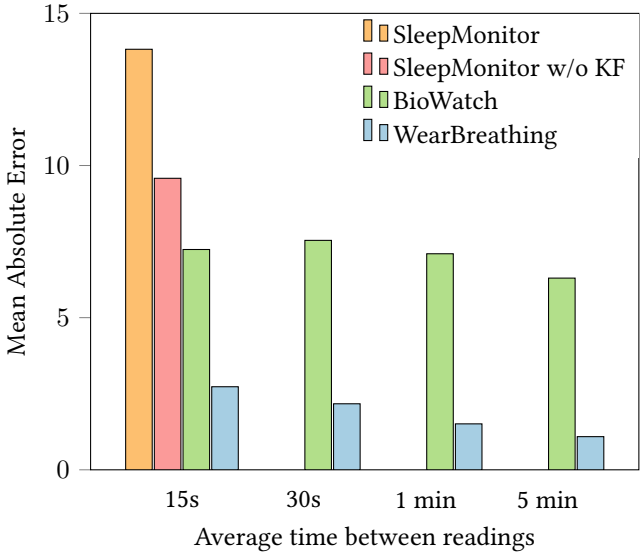

Fig. 5. Mean absolute error for WearBreathing and existing methods when a filter is used to provide readings at various frequencies. SleepMonitor only shown for 15s because there was no threshold that resulted in readings at other intervals.

In addition to accuracy, we explore the feasibility of running WearBreathing in real-time on a smartwatch. Using a combination of data traces and simulation, we show that with duty cycling, WearBreathing can be run in real-time on a modern smartwatch while providing enough battery life to last a full day.

The rest of this section is organized as follows. First, in Section 4.1, we select a few fixed threshold values and compare the performance of WearBreathing against the two existing methods. In Section 4.2, we analyze tunability and characterize how WearBreathing and existing methods perform across their entire threshold domain. Finally, in Section 4.3 we deploy WearBreathing on a smartwatch and analyze runtime and battery life impact.

4.1 System Performance

The performance characteristics of WearBreathing are heavily dependent on the threshold applied to the random forest filter. By using a lower threshold, we are able to increase the accuracy of readings but we receive data less frequently. We highlight this in Figure 5 by setting threshold values to produce readings on average every 15s, 30s, 1 min and 5 mins and showing the mean absolute error at these different frequencies for WearBreathing, BioWatch and SleepMonitor.

We observe that SleepMonitor has a very high error. We suspect this is because of how the Kalman filter described in SleepMonitor relies on exploiting historical readings to boost predictive accuracy. By basing the predicted respiratory rate ($rr_{t|t}$ in the SleepMonitor paper [32]) as a limited change from the posterior/prior respiratory rate in the previous time step ($rr_{t|t-1}$), the Kalman filter is able to reduce the random noise caused by sudden movements. This, however, is predicated upon the assumption that readings are coming in at consistent time intervals. This assumption does not hold in the wild when a filter is used to discard windows because windows are no longer occurring at consistent time intervals. To validate this, we modified the SleepMonitor algorithm to combine the respiratory rate prediction from each axis by taking a simple average instead of using a Kalman filter (SleepMonitor w/o KF in Figure 5) and see that the modified algorithm does indeed perform better on noisy data.

For SleepMonitor, we also observe that there was no threshold we could set that would result in a reading on average every 30 seconds, 1 min or 5 mins. For BioWatch, although we could tune the frequency at which we receive data, we did not see any improvement in the MAE as we decreased the frequency of readings. WearBreathing, on the other hand, in addition to having a dramatically lower MAE, also allows trading frequency for accuracy.

WearBreathing has a significantly lower MAE for all frequencies. When no filter is used, WearBreathing has a MAE of 2.86 compared to BioWatch's 7.01 and SleepMonitor's 9.86, which is a 2.5 and 3.4 times improvement, respectively. When providing a reading every 15*s*, we see a 2.6 and 3.5 times improvement (2.73 MAE for WearBreathing, 7.24 for BioWatch and 9.58 for SleepMonitor). At a reading every 5 minutes, WearBreathing has a 5.8 times lower MAE than BioWatch (1.09 vs. 6.30)

This highlights two points. First, the CNN extractor in WearBreathing by itself has a lower MAE than any existing system. Second, despite the CNN already having a lower MAE, by applying our random forest filter, we can further lower the MAE if we decrease reading frequency. Having this trade-off available makes WearBreathing more applicable to a wider range of applications because now applications can decide whether or not the trade-off is worthwhile based on their requirements.

Conclusion: WearBreathing performs significantly better than existing methods. Using thresholds that result in similar times between readings, WearBreathing's MAE is between 2.5 and 5.8 times lower.

4.1.1 Performance by Activity. As we have shown, the threshold used with our random forest filter greatly affects performance. For fair comparisons, we want to control the frequency at which the three systems provide readings and compare their accuracy. BioWatch's default threshold value of 0.15, provides a reading on average every 50 seconds with a MAE of 7.49. If we set a threshold of 1.05 with WearBreathing, we can achieve the same frequency and a MAE of 2.05 (3.7 times lower). Because SleepMonitor cannot provide a reading on average every 50 seconds, we do not include it in the remaining analysis in Section 4.1. Later on, in Section 4.2, we perform a sensitivity analysis to characterize how all three systems perform across their entire threshold domain.

Using these thresholds, we analyze the MAE and frequency (% of windows accepted) for both WearBreathing and BioWatch during the various activities in our data collection. The results are presented in Table 2. Regardless of activity, WearBreathing has a lower MAE than BioWatch. In some cases, BioWatch does accept more windows than WearBreathing. For example, while standing, BioWatch accepts on average 17.2% of windows compared to WearBreathing, which accepts 7.7% windows. However this increased frequency comes with a much higher MAE (8.2 compared to 1.6). There are also multiple examples of BioWatch not producing any readings during an activity (ex. COPD patients during 6MWT) or only producing readings for one participant (indicated by a missing standard error), but WearBreathing is able to produce readings for multiple participants for all activities.

There can be many reasons why performance is affected by activity. First, as previous works have found, motion makes it harder to isolate the respiratory rate signal [16, 32]. An intuitive example of this can be experienced by trying to measure your pulse while sitting in a moving car or train. However, in our results, we note that high motion activities do not necessarily result in higher error. The 6MWT, for example, likely had the most motion but one of the lowest errors. We think this may be because while the amount of motion is high while walking, the participants wrist is moving in a fairly consistent pattern. So we believe random motion is more detrimental to accuracy than regular, periodic motion. Secondly, while the respiratory signal is generated by the diaphragm and other muscles in the chest and abdomen, we detect this signal at the wrist. Hence it is likely that the participant's posture and body position affects how well the signal propagates from the abdomen/chest to the wrist. For example, if the participant is lying down, having their watch hand on their chest will produce a much stronger signal than if their hand is by their side. Similarly, we expect the signal quality to be affected by whether the participant is sitting, standing or lying down. These factors would have to be carefully considered with traditional, non-machine learning methods. A system that tries to deal with these factors without machine

Table 2. Mean absolute error (SE in brackets) and frequency (% of windows accepted) for BioWatch and WearBreathing broken down by activity and group.

Activity	BioWatch				WearBreathing			
	Healthy		COPD		Healthy		COPD	
	MAE	Freq (%)	MAE	Freq (%)	MAE	Freq (%)	MAE	Freq (%)
6MWT	7.3 (-)	4.5	- (-)	-	0.8 (0.3)	10.0	0.6 (0.3)	11.3
Sitting	5.3 (1.3)	6.8	8.9 (-)	13.8	1.9 (1.5)	16.5	1.2 (0.4)	10.0
Walking	7.5 (-)	27.5	- (-)	-	1.6 (0.8)	20.0	0.7 (0.2)	16.0
Lying	6.8 (0.8)	26.7	12.4 (2.2)	30.3	1.3 (0.6)	7.0	1.3 (0.7)	14.8
Standing	8.2 (3.3)	17.2	7.6 (2.5)	63.5	1.6 (0.7)	7.7	1.3 (0.8)	12.8
Eating	3.2 (1.4)	22.8	7.7 (-)	26.3	1.4 (0.5)	9.2	0.6 (0.2)	17.2
Brushing	5.7 (4.9)	8.5	- (-)	-	2.1 (1.0)	5.2	1.6 (0.6)	18.3
Uncontrolled	6.4 (0.9)	6.3	7.7 (1.2)	15.7	1.5 (0.5)	4.0	2.9 (0.8)	3.0

learning would likely have to implement a posture detection system and then use different signal cleaning and extraction strategies based on body position. Conversely, with our machine learning approach, we are able to provide our system with examples of sensor data and respiratory rates obtained from different body positions and the system learns to extract the signal automatically. The downside is that we lose some interpretability and understanding of why exactly performance is better or worse in some cases.

Conclusion: WearBreathing is able to produce accurate readings across all activities while previous works tend to not generate readings during activities involving more motion.

4.1.2 Performance by Group. Table 2 also shows that BioWatch's performance on the COPD group is lower than on healthy participants. This highlights an important point that is generally well known but worth emphasizing: that systems developed on one population may not generalize to other populations. BioWatch was developed and tested on younger participants without any respiratory conditions, and we see that it works better on healthy participants than on those with COPD. In a real deployment, it may be tempting to read the results of a paper, download or replicate the algorithm and assume you will see the same results on your population. However, as we show, this can result in higher than expected errors. Additionally, without validating on our specific population, we would be wrongly assuming the accuracy of our collected data. Therefore, it is crucial to validate algorithms on the target study population and in the target setting.

One interesting result we see in Table 2 is that using the BioWatch method, respiratory rate is particularly inaccurate (MAE 12.4) for participants with chronic lung disease when they are lying down. This may be because people with COPD have increased airway obstruction and reduced lung volume while lying down [7]. However, because the filter used by BioWatch summarizes the amount of motion in a window, it accepts quite a few windows while patients are lying down, which leads to large errors. Our system on the other hand, is trained on participants with chronic lung disease and is better able to learn and recognize their breathing patterns.

Conclusion: Because WearBreathing is trained using data from healthy participants and participants with COPD, it performs well on both these groups. However, in a real deployment, the performance of WearBreathing (or any system) should be evaluated on the specific population being tested because there can be performance differences between populations.

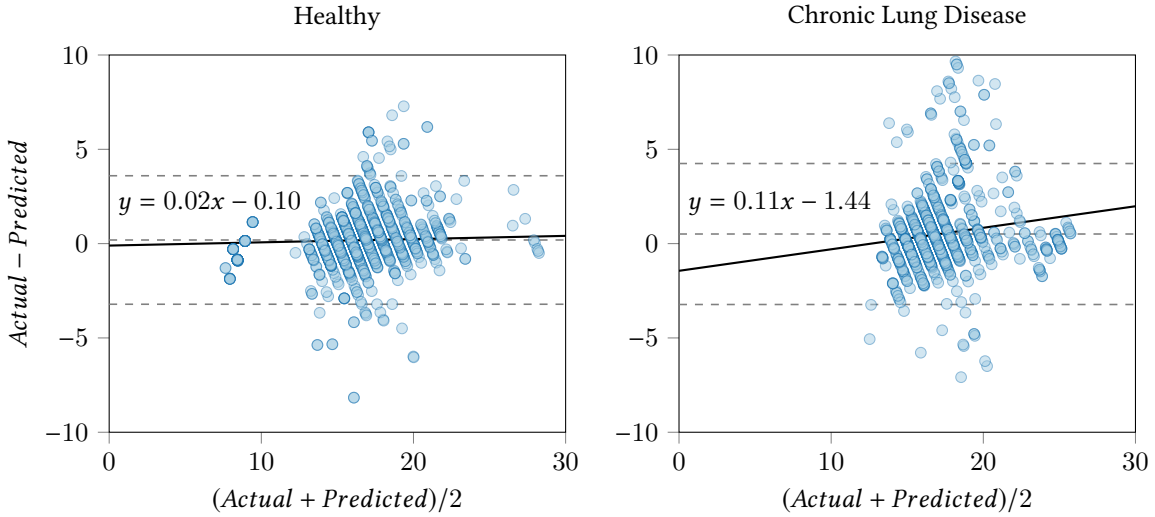

Fig. 6. Bland Altman plot showing agreement between the BioHarness and WearBreathing for both healthy participants and participants with chronic lung disease. For clarity, only a random sample of data points is shown.

4.1.3 Agreement with Ground Truth. To measure agreement between WearBreathing and the ground truth BioHarness, we use a Bland-Altman plot [2]. A Bland-Altman plot helps analyze agreement between two measurement methods. For each pair of measurements, it shows the mean of the two against the difference of the two. This is a useful tool to visualize fixed bias (i.e., a non-zero mean difference), proportional bias (i.e., non-zero slope for the line of best fit) and limits of agreement (i.e., between which two y-values do 95% of data points lie).

The Bland-Altman plot for both the healthy group and the COPD group is shown in Figure 6. For clarity, these graphs only plot a small random sample of data points, however the mean error, limits of agreement and best fit line were computed using all data points.

For the healthy group, we see a fixed bias of 0.19 breaths per minute, proportional bias of 0.02 and limits of agreement between -3.21 and 3.60. This is a fairly low although significant (one sample t-test $p \approx 0$) fixed bias and negligible proportional bias. If desired, the fixed bias can be removed by subtracting 0.19 from all our predicted values. The negligible proportional bias suggests that the error in our prediction is not correlated with the magnitude of the predicted value. Finally, the limits of agreement suggest that 95% of our predicted respiratory rates will be within -3.21 and 3.60 of the BioHarness readings.

For the chronic lung disease group, we see a slightly higher fixed and proportional bias of 0.51 and 0.11, respectively, and limits of agreement between -3.22 and 4.27. While there is slightly less agreement in the chronic lung disease group than the healthy group, both do still show strong agreement.

Conclusion: There is strong agreement between WearBreathing and our ground-truth BioHarness data.

4.2 Tunability

In earlier sections, we selected a few threshold values and presented results for those threshold values. In this section, we explore how the selected threshold affects performance and expand on the black box aspect of our random forest filter. While our random forest filter performs excellently in conjunction with our CNN, because it treats the extractor as a black box, it can also be used with existing respiratory rate extraction methods. By training the random forest to predict the error of BioWatch and SleepMonitor we are able to use our random

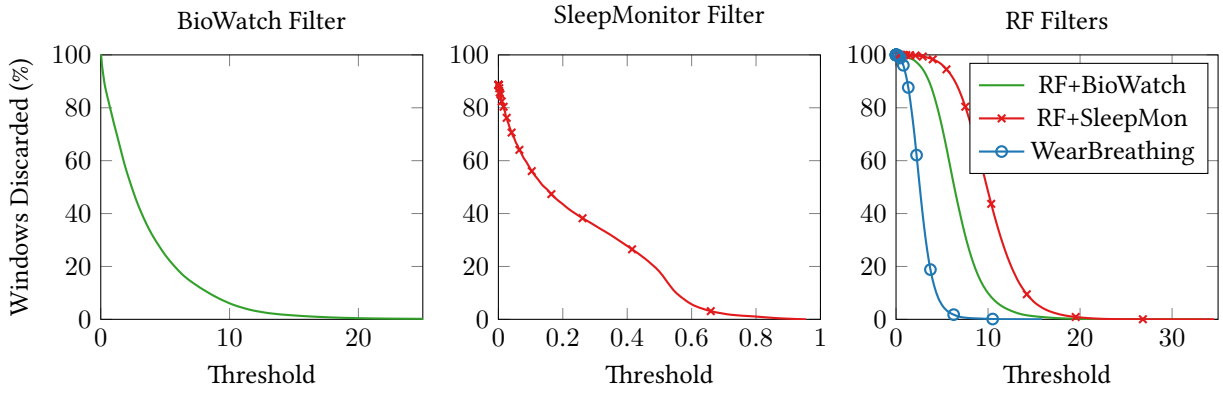

Fig. 7. Proportion of windows discarded by various filters as a function of threshold. RF filter shows three curves depending on which extractor the RF was trained on.

forest filter to improve the performance of these existing methods. In showing that our random forest can be used with existing methods, we highlight that our idea of trading off frequency for accuracy by learning when error will be high is not limited to just our CNN but can be used with other extractors and potentially tasks other than respiratory rate monitoring.

There is a trade-off in selecting a filter threshold. A lower threshold may result in a lower MAE, but at the cost of the number of readings. In Figure 7, we show how the proportion of windows discarded by the filters change as a function of threshold. We observe that with the SleepMonitor filter, even with a threshold of 0, it does not discard 100% of the windows. This is because in each window in our data there are always accelerometer readings where there are no forces acting on the smartwatch besides gravity and therefore the vector magnitude is close to $9.8m/s^2$. This highlights a useful feature that filters should have, which is that they should allow selecting as much or as little data as desired.

Next, we evaluate how the threshold affects the respiratory rate extraction error in an all-vs-all comparison where we apply each filter to each extractor. We vary the threshold for each filter and monitor how that affects the number of windows discarded and the mean absolute error when passing the accepted windows to the extractor. The results for this analysis are shown in Figure 8. We see that simple filters are not smooth functions, which is not a desirable characteristic for tunability since they make it hard to predict how small changes affect accuracy. Our hypothesis for the spikes seen in these filters' curves is that the value these filters are computing is not well-distributed across the threshold domain and there is no direct link between the threshold and filter value. This makes it so that a small change in the filter threshold does not necessarily result in a small change to the number of windows accepted by the filter. Combining our two observations so far, we argue that a good filter should be a smooth function that is capable of discarding anywhere from 0% to 100% of the windows.

Our random forest filter meets both these requirements. When used in conjunction with our CNN, it is much better behaved in that a small increase to the threshold results in a slight increase in number of windows accepted and MAE. This linear property makes the filter a good "turn dial" solution that can be tweaked to the requirements of individual applications. For example, if one wanted to detect respiratory rate with a MAE of 2, they could set the threshold to 1, which would result in a reading roughly every 50 seconds. If one wanted higher accuracy, they could set the threshold to 0.5 which will give an MAE of 1.09 and a reading roughly every 5 minutes. Although the RF predicts error, the threshold value used with the RF filter is simply a value that can

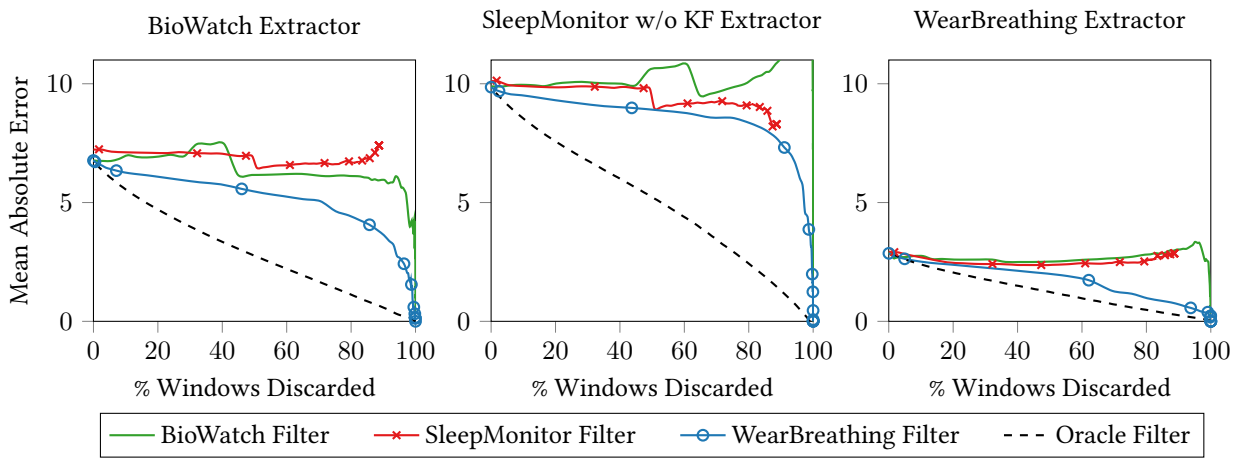

Fig. 8. Mean absolute error vs proportion of windows discarded by filters for different extractors.

be adjusted and not a *guarantee* of accuracy. This is because the RF filter itself is a learned model with its own prediction error. Our RF filter had a 1.20 MAE in predicting the absolute error in our CNN extractor. We also see that while the random forest filter also works for the BioWatch and SleepMonitor extractors, performance is not as strong as with the CNN. The random forest filter has a MAE of 3.48 for the BioWatch extractor and 4.57 for the SleepMonitor extractor.

We also introduce the idea of an *oracle filter*. If the filter's goal is to predict the error in an extractor, the oracle is able to do so with 100% accuracy. Additionally, the oracle knows the extractor's error on all past, present and future windows of data. Therefore, if we use a threshold of 20 with the oracle filter, it only accepts a window if the extractors error on this window is among the lowest 20% of all errors. As we can see in Figure 8, none of the filters perform close to the oracle when applied to the BioWatch or SleepMonitor extractors. This is because both the extraction and filter have some error which compounds to make a poorly performing system. However, for our CNN which has a relatively low error, the random forest filter is much closer to the oracle filter. This again suggests that our RF filter is learning what makes the CNN perform well on a window.

Conclusion: WearBreathing is highly tunable because its threshold value is intuitive and its filter is a smooth function that is capable of rejecting any given proportion of windows.

4.3 Battery Life

Battery life is a critical consideration for any mobile system. After collecting data from participants and training our models, we explore the implication on battery life of running WearBreathing on an actual smartwatch. To do so, we use a combination of experiments and simulation. Our simulation makes use of the IMU data collected from our 14 participants as traces to simulate real-world battery life.

In addition to the IMU traces, we need battery consumption measurements under different modes of operation. These modes are idle, recording IMU, running RF and running CNN. When the watch is idle, its CPU is in a sleep state so no monitoring or processing is occurring. During the RF mode, a wake lock is held and IMU data is collected. The RF mode collects IMU data and runs our random forest filter. Finally, the CNN mode collects sensor data, runs the RF filter but does not use the RF output to determine whether or not the CNN is run.

Instead, both the RF and CNN are run on all windows. For the RF and CNN states, we also need measurements on how long the RF and CNN take to execute on a window of data.

4.3.1 Battery Data Collection. To run our CNN and RF on the smartwatch, we first convert our trained CNN model to TensorFlow Lite[4] (version 1.10.0). The converted model can be loaded on a smartwatch and used by a TensorFlowLite interpreter that we call from our Java app. To run the RF model, we transpile it directly into Java code using sklearn-porter[5]. While the CNN operates on raw sensor data, the RF model requires features extracted from the data. This feature extraction is also done in Java code.

We configure our app to run these models according to the different modes of operation and intermittently record battery levels. We charge six LG Urbane watches, the same ones we used with our participants, to over 90% battery and run our application with the screen off until the battery completely drains. We run each mode of operation on all six watches, and using the initial and final battery level, we calculate the mean and standard deviation of change in battery level per hour. The results for this are shown in Table 3.

Table 3. Battery % change per hour during different modes of operation.

Mode	Battery Δ (%/hour)	
	Mean	SD
Idle	-1.67	2.71
Continuous IMU	-5.42	0.26
Continuous IMU + RF	-22.52	1.47
Continuous IMU + RF + CNN	-22.84	1.42

To measure execution time, we use one watch and run our application for 10 minutes. During these 10 minutes, we time how long it takes to run the RF on 100 windows and divide the result by 100. This helps minimize the overhead of our timing functions. The same procedure is repeated for the CNN. We find that the RF takes on average 26.54ms (± 1.69ms) per window and the CNN takes 32.47ms (± 0.68ms) per window. With a sampling rate of 20Hz we obtain a reading every 50ms, meaning our processing should run in under 50ms to run in real-time. While running both the RF and CNN on all windows would not meet the 50ms requirement, WearBreathing requires running only the RF on all windows. The CNN is run on windows where the RF output is below the threshold. With a threshold of 1.05, less than 20% of windows are accepted by the filter. Therefore, the CNN is run infrequently enough that WearBreathing is able to run in real-time.

4.3.2 Simulation Setup. We use the IMU traces described in Section 3.1, energy consumption measurements and runtime measurements to simulate battery life of WearBreathing. Our simulator, which is written in Python, implements the state machine shown in Figure 9. It starts at $time = 0$ seconds with battery level at 100% and in the RECORD state. While in the RECORD state, the simulator steps through the IMU traces for 30 seconds, until the data window fills up. Once the window is full, it enters the RECORD-AND-RF state. In this state, the RF is run and if the RF output value is below a specified threshold, the CNN is also executed (CNN state).

The time spent in each state depends on the simulation conditions (i.e. RF threshold and duty cycling scheme) and measured values such as the RF and CNN execution times. In our simulation, we randomly sample the execution time for each RF and CNN run from a normal distribution parameterized on our measured mean and standard deviation. Using the time spent in each state ($duration_s$), we update the battery level according

[4]https://www.tensorflow.org/lite/

[5]https://github.com/nok/sklearn-porter

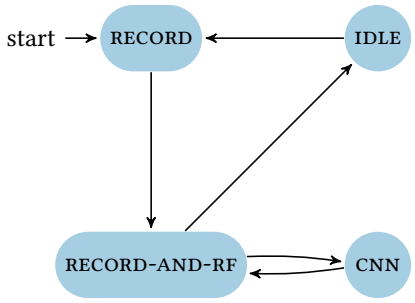

Fig. 9. Battery life simulation state machine.

to Equation 2 where $battery\_consumption_s$ is also drawn from a normal distribution based on the mean and standard deviation of the corresponding mode from 3. The simulator steps through the state machine until the battery level drops to 0% at which point, the time is recorded and the current simulation run ends.

$$battery\_level_{new} = battery\_level_{prev} + duration_s * battery\_consumption_s \quad (2)$$

We also simulate the effect of duty cycling. With duty cycling, data is recorded and processed for some period of time and then the watch is allowed to sleep for some period of time. While this reduces the amount of data collected, it can be a useful way to save energy. We simulate two types of duty cycling; *fixed duty cycle* and *adaptive duty cycle*. In both cases, our simulator cycles between IDLE, RECORD and RECORD-AND-RF; the difference lies in when these transitions occur. In fixed duty cycling, transitions occur periodically. For example, with an 8min/2min duty cycle, the simulator spends 8 minutes in the IDLE state and then transitions to RECORD. The RECORD and RECORD-AND-RF states then collectively run for 2 minutes. In adaptive duty cycling, the IDLE state runs for a fixed period of time, however the RECORD-AND-RF state only runs until n windows where the RF output is below the threshold have occurred.

From previous work [24], we found that transitions between sleep and wake states consume additional power. The sleep to wake transition, which lasts 1 second, consumes 19% more power than the awake state and the wake to sleep transition, also lasting 1 second, consumes 5% more power. In our simulation, we pessimistically assume that each transition takes two seconds and consumes twice as much power as the continuous IMU mode.

Table 4 shows the simulated battery life of the smartwatch under various recording conditions. To validate our simulator, we configure it to perform the modes of operation listed in Table 3, for which we have experimental data. Simulating a continuous IMU recording (simulator always in the RECORD state) yields an expected 18.5 hours of battery life. This is slightly higher than our experimental observation (approximately 17 hours). The small difference in these battery lives is because in our experiments, we charged the devices to anywhere from 90% to 100%, whereas our simulator always begins with 100% battery life. The other two modes produce similar results.

4.3.3 WearBreathing Battery Life. Simulating WearBreathing with data traces from our 14 participants, we estimate that running WearBreathing continuously would result in just over 5 hours of battery life, which is expected because running the RF continuously results in roughly 5.5 hours of battery life and WearBreathing is equivalent to always running the RF and occasionally running the CNN. While five hours of battery life would not be acceptable in a real deployment, using a fixed duty cycle (2 minutes record, then 8 minutes idle), our simulation estimates a battery life of over 21 hours. Using an adaptive duty cycle, where the device sleeps for 8 minutes,

Table 4. Battery life of a smartwatch under various conditions. First three conditions were both experimentally run and simulated and used to test the accuracy of the simulation. Bottom three conditions show WearBreathing under different recording conditions.

Condition	Actual		Simulated	
	Mean	SD	Mean	SD
Continuous IMU	16h 54min	1h 35min	18h 30min	11min
Continuous IMU + run RF	4h 17min	12min	5h 28min	2min
Continuous IMU + run RF and CNN	3h 50min	26min	3h 57min	2min
WearBreathing Continuous			5h 18min	3min
WearBreathing DC (2min/8min)			21h 30min	25min
WearBreathing Adaptive DC ($n = 3$)			1 day 20h 19min	56min

and then wakes up until it obtains three reliable respiratory rate readings, we obtain an estimated battery life of over 42 hours.

While duty cycling allows a full day's worth of battery life, it comes at the cost of missing data while the watch is in a sleep state. Depending on the application, this may or may not be an acceptable trade-off. For applications that need continuous recording of data, there is the option of collecting sensor data continuously, but delaying processing the data. The watch could record data while being worn by a user but wait until being charged to process the data or upload it to a server for processing. This would allow continuous recording, but the measured respiratory rate would not be available immediately. The energy consumption of this offline processing scheme is the same as the Continuous IMU condition shown in Table 4 (17 actual, 18.5 hours simulated), and is also enough to last a full day.

One limitation to our battery life analysis is that the battery measurements used in our simulator were obtained while the smartwatch was still and the screen off. Under normal usage, when the "always-on screen" is disabled, a wrist gesture or screen press can be used to turn on the display temporarily. However in our experiments, the screen was almost never turned on. According to Liu et al. [26], the screen is the most power hungry component of a smartwatch. Therefore, we expect that our simulated battery life is overestimated. However, even if actual battery life is half our predicted values, using either a fixed or adaptive duty cycle should be enough to last a full day's worth of recording.

Conclusion: Full day battery life is possible with WearBreathing using either duty cycling or offline analysis.

5 RELATED WORK

The key difference between WearBreathing and existing works is that both our filter and respiratory rate extraction are automatically learned from data. This is beneficial to our filter because we remove any assumptions about what makes data unreliable. So for example, we do not assume that motion makes data unreliable and therefore try to estimate motion. As we have shown, this results in a better behaved, more intuitive filter. Similarly, for the extractor, we do not manually look for a specific periodic signal in the data. Instead, we let a CNN learn to extract the signal. As demonstrated by the CNN being able to generalize across participants and participant groups, it is able to learn more complex patterns which accurately predict respiratory rate.

Another recent paper, MindfulWatch [14], uses accelerometer and gyroscope data to estimate respiratory rate during meditation. Similar to SleepMonitor, MindfulWatch builds a historical model of respiratory rates. However, since MindfulWatch is designed for meditation where participants hold certain postures for periods of times, it monitors for posture changes and reinitializes a model for each new posture. This idea works well for

meditation because motion occurs for a short period of time followed by a longer period of being still. This gives the Kalman filter or some other historical model time to re-initialize. While this pattern of motion followed by stillness does occur in the wild, we suspect that it is not very common. Additionally, stillness during meditation is very different from being still in the wild. For example, over the course of a day, a person sitting at a desk would be considered still. But this person could be typing or using a phone, which still results in motion at the wrist. We suspect that for these reasons, an approach designed for meditation would not transfer well to wild environments.

While we used the accelerometer and gyroscope on a smartwatch, other works have looked at detecting respiratory rate from a photoplethysmogram (PPG) collected from a pulse oximeter [4, 6, 19]. However, these techniques are also susceptible to motion artifacts [27]. PPG also has the downside of being affected by skin tone [8, 23] and conditions such as anemia [31]. Additionally, while most smartwatches use a pulse oximeter for heart rate monitoring, very few provide access to the raw PPG data and as of Android API version 28, the Android sensor manager documentation does not have an entry for PPG sensors[4]. While most studies examining pulse oximetry use data collected from a fingertip, the forehead or an earlobe, the current interest in smartwatches has brought about a push to bring pulse oximetry to the wrist. For example, the first wrist-based pulse oximeter was approved by the FDA in 2018 [12]. As wrist-based pulse oximetry becomes more developed, respiratory rate from wrist-based PPG data may also be a viable option. This could open doors to sensor fusion techniques that use both IMU and PPG data for even more accurate respiratory rate monitoring.

6 DISCUSSION

The idea of filtering data by throwing out sections that are unreliable is not new. However, we argue that in in-the-wild environments this becomes a much more significant and challenging problem because there are no guarantees about what is happening in the environment, and there are many assumptions embedded within systems. For example, although we tried to make our data collection study as close to real-world as possible by placing very few constraints on participants, we did operate under the assumption that the participants were wearing their smartwatch. In the real-world, users are likely not wearing their smartwatch for large parts of the day (e.g., while sleeping or relaxing at home). In our experience, detecting when the watch is not being worn is not trivial. Applying any sort of detection algorithm to sensor data while the watch is not being worn can result in unusual results. For example, we have observed that even when a smartwatch is not being worn, the heart rate sensor still produces valid heart rate readings. Similarly, BioWatch takes the most periodic signal within a frequency range, so it is possible that it will produce respiratory rate readings even if the watch is not worn. If we wanted to deploy WearBreathing in a real-world experiment, we would have to analyze how it behaves in these kinds of scenarios. Ideally, the filter would reject data from when the watch is not being worn. In our current work, the random forest was only trained with data where the watch was worn so we do not know how it will behave in a real-world deployment. However, this is not an inherent limitation of our result and can be solved with more training data or by implementing another filter to reject data from when the watch is not worn.

In our analysis, we accept all windows where the value generated by the filter is below some threshold. Because the goal of the filters is to select windows that will produce a high accuracy reading, accepting all values below a threshold is a way to control for accuracy. If we set a lower threshold, we are seeking higher accuracy. While we empirically found that 90% of readings occur within 90 seconds of each other, there is no guarantee of how long until a reading is produced (although not receiving readings may itself be a useful signal). Though stochastically receiving readings may be acceptable for some applications, other applications may require consistent, periodic readings. WearBreathing is also able to support these applications. If an application needs a reading every minute, it could buffer filter and extractor outputs for the last minute, and select the best window(s) from the buffer when

[4]https://developer.android.com/reference/android/hardware/Sensor

a reading had to be produced. In this use case, applications also have the freedom to determine how they choose the best window(s). They could, for example, choose the window with the minimum filter value or take an average of the 10% of windows with the lowest filter value.

Furthermore, while we are able to achieve accurate results using a single threshold value for all participants, it may be possible to further improve accuracy by selecting custom thresholds for each user. However, this would require collecting ground truth data from each participant in order to find the optimal threshold. For example, for a study with a small number of participants, the participant on-boarding process could require wearing a ground truth device and a smartwatch for a short period. Then, the filter threshold applied to this user's data is chosen based on the timing/accuracy requirements of the study. However, such an approach would not be suitable for larger studies or crowd-sourced data.

We would also like to point out the importance of validating algorithms on target populations. A system for respiratory rate monitoring is more likely to be useful for people who have some lung disease. However, as we have shown, there can be a difference in accuracy on participants with chronic lung disease and healthy participants. The essence of the issue is that the performance on our test data may not match performance in our actual deployment. One way around this would be to collect a small amount of ground truth data from some or all participants in the real deployment. For example, in a 3 month deployment of 20 participants, it may be feasible to randomly select 5 participants to wear the BioHarness for a few hours. This would give an estimate of how the system is performing in the deployment. While this has overhead, we believe a scheme like this is akin to insurance. It is worth paying this overhead and having an idea of the system's accuracy rather than ending up with hard-acquired and expensive data with little understanding of the accuracy.

Finally, our proposed system makes use of smartwatches, which have limited battery and processing capabilities. We have shown that with duty cycling or offline processing, our approach can run on a smartwatch and provide a full day's worth of battery life. Alternatively, offloading approaches may allow real-time processing with very little overhead and without the need for duty cycling, by running the random forest filter on an auxiliary low-power processor and waking up the main CPU to run the CNN when a good window is detected.

An example of an application leveraging offloading technology is Google's *Now Playing* [33]. This system uses an always-on microphone and digital signal processor (DSP) to listen to the environment. When the DSP detects that music is playing, it wakes up the main processor, which can search a small local database of popular songs or send the information to the cloud to match against a large database. With this three-tier architecture, Now Playing is able to provide always-on song recognition with less than 1% daily battery usage. Similarly, we could run the WearBreathing random forest filter on an always-on, low-power processor. When it accepts a window we can wake up the main processor to either run the CNN on the smartwatch GPU or off-load it to the smartphone or a cloud service.

While programming an integrated DSP is feasible for large companies, it is difficult for most developers and researchers. There are, however, research projects such as LittleRock [29], K2 [25] and Sidewinder [24] that try to make these kinds of systems easier to develop. Sidewinder in particular, proposed providing developers with data processing algorithms as building blocks that would run on the low power processor. The BioWatch and SleepMonitor filters we discussed would be straightforward to set up on a system like Sidewinder. If machine learning models such as random forests were supported, we would be able to run the filter on the low-power processor.

7 CONCLUSION

In this paper, we present WearBreathing, which enables everyday respiratory rate monitoring. The WearBreathing system is composed of two parts. The first is a random forest based filter that is able to detect when input data will result in an accurate respiratory rate. The second is a convolutional neural network based model for

extracting respiratory rate from accelerometer and gyroscope data. The combination of these two models results in a tunable respiratory rate monitor that enables users to trade-off frequency for accuracy. Testing on a diverse, out-of-the-lab dataset, we demonstrate that WearBreathing is able to detect respiratory rate with an MAE of 2.05 breaths/minute while producing a reading on average every 50 seconds, which is 3.6 times better than the previous state of the art. We demonstrate that WearBreathing is highly tunable: users who are more interested in accuracy could opt to receive less, but more accurate data by simply decreasing a single threshold. Finally, we show that a current smartwatch is able to run WearBreathing while providing a full day's worth of battery life. The net result is a system that, for the first time, is able to accurately monitor respiratory rate outside of lab environments using a smartwatch. We hope WearBreathing inspires and enables new research into respiratory rate analysis.

REFERENCES

[1] Álvaro Barbero and Suvrit Sra. 2014. Modular Proximal Optimization for Multidimensional Total-Variation Regularization. *arXiv:1411.0589 [math, stat]* (Nov. 2014). arXiv:math, stat/1411.0589 http://arxiv.org/abs/1411.0589

[2] J. M. Bland and D. G. Altman. 1986. Statistical Methods for Assessing Agreement between Two Methods of Clinical Measurement. *Lancet (London, England)* 1, 8476 (Feb. 1986), 307–310.

[3] Chollet, François and others. 2015. Keras. https://keras.io

[4] K. H. Chon, S. Dash, and K. Ju. 2009. Estimation of Respiratory Rate From Photoplethysmogram Data Using Time–Frequency Spectral Estimation. *IEEE Transactions on Biomedical Engineering* 56, 8 (Aug. 2009), 2054–2063. https://doi.org/10.1109/TBME.2009.2019766

[5] Michelle A. Cretikos, Rinaldo Bellomo, Ken Hillman, Jack Chen, Simon Finfer, and Arthas Flabouris. 2008. Respiratory Rate: The Neglected Vital Sign. *The Medical Journal of Australia* 188, 11 (June 2008), 657–659. https://www.mja.com.au/journal/2008/188/11/respiratory-rate-neglected-vital-sign

[6] Parastoo Dehkordi, Ainara Garde, Behnam Molavi, J. Mark Ansermino, and Guy A. Dumont. 2018. Extracting Instantaneous Respiratory Rate From Multiple Photoplethysmogram Respiratory-Induced Variations. *Frontiers in Physiology* 9 (2018), 948. https://doi.org/10.3389/fphys.2018.00948

[7] Loubna Eltayara, Heberto Ghezzo, and Joseph Milic-Emili. 2001. Orthopnea and Tidal Expiratory Flow Limitation in Patients With Stable COPD. *Chest* 119, 1 (Jan. 2001), 99–104. https://doi.org/10.1378/chest.119.1.99

[8] J. R. Emery. 1987. Skin Pigmentation as an Influence on the Accuracy of Pulse Oximetry. *Journal of Perinatology: Official Journal of the California Perinatal Association* 7, 4 (1987), 329–330.

[9] Paul L. Enright. 2003. The Six-Minute Walk Test. *Respiratory Care* 48, 8 (Aug. 2003), 783–785. http://rc.rcjournal.com/content/48/8/783

[10] J. F. Fieselmann, M. S. Hendryx, C. M. Helms, and D. S. Wakefield. 1993. Respiratory Rate Predicts Cardiopulmonary Arrest for Internal Medicine Inpatients. *Journal of General Internal Medicine* 8, 7 (July 1993), 354–360.

[11] D. R. Goldhill, A. F. McNarry, G. Mandersloot, and A. McGinley. 2005. A Physiologically-Based Early Warning Score for Ward Patients: The Association between Score and Outcome. *Anaesthesia* 60, 6 (June 2005), 547–553. https://doi.org/10.1111/j.1365-2044.2005.04186.x

[12] A. Guber, G. Epstein Shochet, S. Kohn-Bouzaglou, and D. Shitrit. 2018. Oxitone 1000: A First FDA Cleared Wrist-Sensor Pulse Oximeter for Continuous Patient Monitoring. In *D51. PHYSIOLOGY AND PHYSICAL ACTIVITY IN PULMONARY REHABILITATION*. American Thoracic Society, A7061–A7061. https://doi.org/10.1164/ajrccm-conference.2018.197.1_MeetingAbstracts.A7061

[13] Jono Hailstone and Andrew E. Kilding. 2011. Reliability and Validity of the Zephyr™ BioHarness™ to Measure Respiratory Responses to Exercise. *Measurement in Physical Education and Exercise Science* 15, 4 (Oct. 2011), 293–300. https://doi.org/10.1080/1091367X.2011.615671

[14] Tian Hao, Chongguang Bi, Guoliang Xing, Roxane Chan, and Linlin Tu. 2017. MindfulWatch: A Smartwatch-Based System For Real-Time Respiration Monitoring During Meditation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (Sept. 2017), 1–19. https://doi.org/10.1145/3130922

[15] Nima Hatami, Yann Gavet, and Johan Debayle. 2018. Classification of Time-Series Images Using Deep Convolutional Neural Networks. In *Tenth International Conference on Machine Vision (ICMV 2017)*, Vol. 10696. International Society for Optics and Photonics, 106960Y. https://doi.org/10.1117/12.2309486

[16] Javier Hernandez, Daniel McDuff, and Rosalind W. Picard. 2015. BioWatch: Estimation of Heart and Breathing Rates from Wrist Motions. In *Proceedings of the 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth '15)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 169–176. http://dl.acm.org/citation.cfm?id=2826165.2826190

[17] Andrey Ignatov. 2018. Real-Time Human Activity Recognition from Accelerometer Data Using Convolutional Neural Networks. *Applied Soft Computing* 62 (Jan. 2018), 915–922. https://doi.org/10.1016/j.asoc.2017.09.027

[18] Álvaro Barbero Jiménez and Suvrit Sra. 2011. Fast Newton-Type Methods for Total Variation Regularization. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, Lise Getoor and Tobias Scheffer (Eds.). Omnipress, Bellevue, Washington, USA, 313–320.

[19] A. Johansson. 2003. Neural Network for Photoplethysmographic Respiratory Rate Monitoring. *Medical and Biological Engineering and Computing* 41, 3 (May 2003), 242–248. https://doi.org/10.1007/BF02348427

[20] James A. Johnstone, Paul A. Ford, Gerwyn Hughes, Tim Watson, and Andrew T. Garrett. 2012. Bioharness™ Multivariable Monitoring Device: Part. I: Validity. *Journal of Sports Science & Medicine* 11, 3 (Sept. 2012), 400–408. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3737934/

[21] James A. Johnstone, Paul A. Ford, Gerwyn Hughes, Tim Watson, and Andrew T. Garrett. 2012. Bioharness™ Multivariable Monitoring Device: Part. II: Reliability. *Journal of Sports Science & Medicine* 11, 3 (Sept. 2012), 409–417. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3737936/

[22] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv:cs/1412.6980 http://arxiv.org/abs/1412.6980

[23] K. H. Lee, K. P. Hui, W. C. Tan, and T. K. Lim. 1993. Factors Influencing Pulse Oximetry as Compared to Functional Arterial Saturation in Multi-Ethnic Singapore. *Singapore Medical Journal* 34, 5 (Oct. 1993), 385–387.

[24] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldana. 2016. Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 205–215. https://doi.org/10.1145/2872362.2872398

[25] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. 2015. K2: A Mobile Operating System for Heterogeneous Coherence Domains. *ACM Trans. Comput. Syst.* 33, 2 (June 2015), 4:1–4:27. https://doi.org/10.1145/2699676

[26] Xing Liu, Tianyu Chen, Feng Qian, Zhixiu Guo, Felix Xiaozhu Lin, Xiaofeng Wang, and Kai Chen. 2017. Characterizing Smartwatch Usage in the Wild. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '17*. ACM Press, Niagara Falls, New York, USA, 385–398. https://doi.org/10.1145/3081333.3081351

[27] K. Venu Madhav, M. Raghu Ram, E. Hari Krishna, Nagarjuna Reddy Komalla, and K. Ashoka Reddy. 2013. Robust Extraction of Respiratory Activity From PPG Signals Using Modified MSPCA. *IEEE Transactions on Instrumentation and Measurement* 62, 5 (May 2013), 1094–1106. https://doi.org/10.1109/TIM.2012.2232393

[28] Neal Patwari, Joey Wilson, Sai Ananthanarayanan, Sneha K Kasera, and Dwayne R Westenskow. 2014. Monitoring Breathing via Signal Strength in Wireless Networks. *IEEE Transactions on Mobile Computing* 13, 8 (2014), 1774–1786.

[29] Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu. 2011. LittleRock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones. *IEEE Pervasive Computing* 10 (2011), 12–15.

[30] Noah Rubio, Richard A Parker, Ellen M Drost, Hilary Pinnock, Christopher J Weir, Janet Hanley, Leandro C Mantoani, William MacNee, Brian McKinstry, and Roberto A Rabinovich. 2017. Home Monitoring of Breathing Rate in People with Chronic Obstructive Pulmonary Disease: Observational Study of Feasibility, Acceptability, and Change after Exacerbation. *International Journal of Chronic Obstructive Pulmonary Disease* Volume 12 (April 2017), 1221–1231. https://doi.org/10.2147/COPD.S120706

[31] John W. Severinghaus and Shin O. Koh. 1990. Effect of Anemia on Pulse Oximeter Accuracy at Low Saturation. *Journal of Clinical Monitoring* 6, 2 (April 1990), 85–88. https://doi.org/10.1007/BF02828282

[32] Xiao Sun, Li Qiu, Yibo Wu, Yeming Tang, and Guohong Cao. 2017. SleepMonitor: Monitoring Respiratory Rate and Body Position During Sleep Using Smartwatch. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (Sept. 2017), 1–22. https://doi.org/10.1145/3130969

[33] Blaise Agüera y Arcas, Beat Gfeller, Ruiqi Guo, Kevin Kilgour, Sanjiv Kumar, James Lyon, Julian Odell, Marvin Ritter, Dominik Roblek, Matthew Sharifi, and Mihajlo Velimirović. 2017. Now Playing: Continuous Low-Power Music Recognition. *arXiv:1711.10958 [cs, eess]* (Nov. 2017). arXiv:cs, eess/1711.10958 http://arxiv.org/abs/1711.10958

[34] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. 2015. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, Buenos Aires, Argentina, 3995–4001. http://dl.acm.org/citation.cfm?id=2832747.2832806

[35] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. 2017. Convolutional Neural Networks for Time Series Classification. *Journal of Systems Engineering and Electronics* 28, 1 (Feb. 2017), 162–169. https://doi.org/10.21629/JSEE.2017.01.18

Received November 2018; revised February 2019; accepted April 2019