Anomaly Detection in WBANs Using CNN-Autoencoders and LSTMs

Kartikeya Dubey(✉) and Chittaranjan Hota

Birla Institute of Technology and Science, Pilani, Hyderabad Campus, Hyderabad 500078, India
{f20200031,hota}@hyderabad.bits-pilani.ac.in

Abstract. Wireless Body Area Networks (WBANs) are wireless networks that consist of microscopic sensors that are either placed on a subject's body or attached to their clothes. These low-power devices track the physiological readings from a subject and relay them to a server over the Internet. Since WBANs find various uses, from remote medical monitoring to early disease detection in the medical field, their readings must be accurate. Anomalies can occur in the data being transmitted for various reasons, such as sensor faults or malicious attacks.

In some instances, however, what is considered an anomaly may not be an anomaly, and the readings are correct. To mitigate such cases, we must detect anomalies promptly. The anomaly detection process developed here consists of point and contextual anomaly detection. Point anomaly detection deals with checking whether, given a set of readings, some of the readings are anomalous in an isolated sense. Contextual anomaly detection then checks whether readings from other sensors can corroborate the reading of our "faulty" sensor. This paper has employed a CNN Autoencoder and an LSTM-based classifier to do this two-step process. Our model shows an accuracy of 94% and a loss of 14%.

1 Introduction

With the advent of microscopic sensors, getting readings of various physiological parameters unobtrusively became possible. A natural progression of this technology is a network of microscopic sensors spread across a person's body. This is the concept behind Wireless Body Area Networks (hereafter referred to as WBANs).

The various sensors placed across the body collect the data. Since these sensors have extremely low power consumption, they cannot transmit the data to a remote location. So, to overcome this, the sensors first send their data to a central unit, which is closer in proximity, which then transmits the data to a remote data centre from where doctors worldwide can access it [1]. It is immediately apparent how WBANs will significantly improve patient care and quality of life. Using data from WBANs, medical professionals can be alerted on any development in their patients without the patients having to be physically present to get readings taken. The data is also valuable because medical professionals can get real-time readings of patients afflicted with various diseases, providing valuable data on the day-to-day dynamics of a disease.

The primary drawbacks of WBANs currently are security concerns and anomalous readings. Due to the sensitive nature of the data, a breach of data integrity would severely violate patients' privacies. Anomalous readings may lead to misdiagnosis or

L. Barolli (Ed.): AINA 2024, LNDECT 201, pp. 187–197, 2024.
https://doi.org/10.1007/978-3-031-57870-0_17

delivery of incorrect medical dosages, which is equally harmful to a patient's health. Our work in this research focuses on this second aspect of WBANs.

There are existing deep-learning algorithms for this task. Autoencoders are a popular technique employed to capture spatially sensitive data. The type of autoencoder, whether CNN-based or LSTM-based, matters as different features ofa the data are prioritized by each. With this in mind, we have proposed a novel CNN-based autoencoder. The output of this autoencoder is given as input to the LSTM, which finally classifies the anomalies.

The rest of this paper follows the following structure: Sect. 2 briefly explains the related works in anomaly detection in WBANs. Section 3 describes the design of the proposed model and its algorithms for point-anomaly and contextual-anomaly detection. Section 4 presents the experimental results and analysis. Section 5 concludes the paper and provides future research directions.

2 Related Work

To approach this problem, M. U. Harun Al Rasyid et al. [2] employ a sliding window. The $(n+1)^{th}$ measurement is predicted using the data from this sliding window. This process is repeated for each physiological reading, and the error of the predicted value is checked against a dynamic threshold. The reading is considered anomalous if most of the physiological readings' errors exceed their respective thresholds. The paper proposes using errors from regressed data to predict erroneous readings. However, it simply considers that if most of the readings are erroneous, the sensors must be erroneous rather than the readings supporting each other being accurate.

Mikel Canizo [3] proposes using multi-head CNN-RNNs to solve the problem. They use CNNs with 1D convolution to convolute each physiological sensor time series. This convoluted data is then fed to the RNN-based classifier that classifies all the sensors simultaneously. The paper then experimentally analyses the results of their model against pre-existing CNN-based models. The model performs well against traditional CNN models. However, the convolution used by this approach does not convolute all the sensor's data together, which could help increase accuracy by introducing an element of cohesion.

The approach employed by F. A. Khan et al. [4] uses a Markov model. First, the features of the ECG dataset are extracted using a Discrete Wavelet Transform (DWT). The features are divided into feature sequences. The probability for each feature sequence is then calculated, and the system decides if an anomaly has occurred. This model performs well in cases where anomalies make up 10% or even 5% of the dataset. The paper, however, focuses on ECG data. We aim to develop a more generic model for anomaly detection. Once again, the model implemented in this paper is for a single time series data while We aim to find anomalies across multiple time series datasets.

O. I. Provotar [5] uses an LSTM-based autoencoder to detect anomalies. First, they determine the dominant frequency using autocorrelation or Welch's power spectral density estimation. Then, the signal is decomposed into seasonal, trend, and residual. Then, the residual component is used to decide whether an anomaly occurs. This paper, once again, aims to find anomalies in only a single time series of data. We aim to find anomalies across multiple time series datasets.

3 Proposed Model

The proposed anomaly detection process consists of 2 steps: first, the CNN autoencoder reconstructs the input data; if, for any point, the error is greater than a set threshold, then the point is labelled as point anomaly. Then, the encoded data is passed to the LSTM classifier, classifying it as being contextually anomalous or not. Thus, we have proposed a combination of CNN and LSTM models to identify contextual anomalies present in the sensor values.

3.1 Data Preparation

The dataset used for the model's training is the MIMIC-I dataset [7]. The dataset contains detailed medical records for 72 patients. The data we are concerned with are the physiological vitals: the pulse, blood oxygen saturation (SpO_2), heart rate (HR), diastolic arterial blood pressure (ABPdia), systolic arterial blood pressure (ABPsys), mean arterial blood pressure (ABPmean), and the respiration rate (resp). Using the NumPy Python library [8], these 7 metrics were first converted into one 2D array. The values were then normalised to the range [0–1], both inclusive. The correlation matrix was then found for these metrics using the scikit-learn library [13]. Next, this correlated data is fed to the CNN AutoEncoder. For training, any row in the array that contains a 0 value is dropped as a sensor has been removed or is faulty.

3.2 Point Anomaly Detection

Point anomalies are points that are anomalous without considering the contextual readings. A CNN autoencoder is employed to perform this analysis. The training input provided to the autoencoder is the 2D array that contains all the vital readings. The array is split into 7×7 sections since providing square dimensions to the CNN autoencoder allows for easier output formatting.

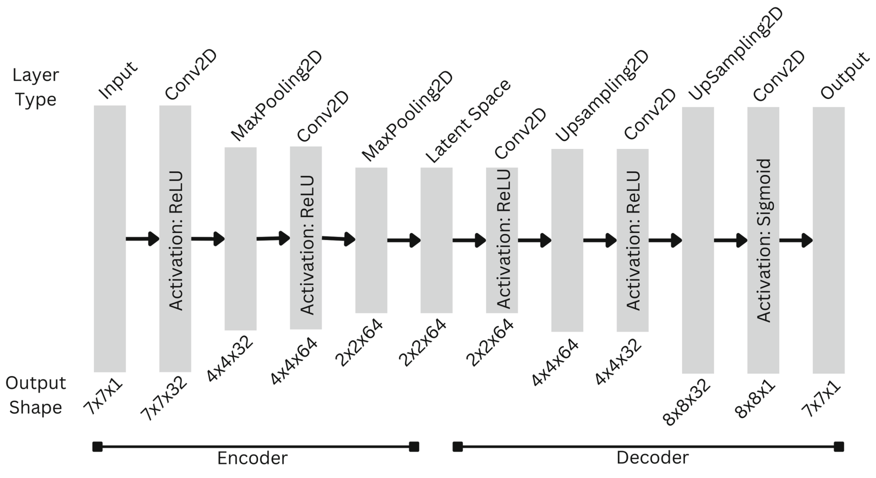

Fig. 1. An Overview of the autoencoder model

The encoder consists of five layers, an input layer and two pairs of Convolutional and MaxPooling Layers. The following five layers make up the decoder section of the model. This architecture can be seen in Fig. 1. The Convolutional layers detect patterns and features in the input, and the MaxPooling layers reduce the spatial dimensions of the input. For the threshold, the standard deviation of the error is considered. i.e., any point where Eq. 1 is satisfied is considered anomalous.

$$\text{value}_{predicted} - \text{value}_{actual} > \text{value}_{standarddeviation} \tag{1}$$

These anomalies are labelled as such and passed on to the next step. The coded form of the input data obtained from the encoder is also passed along as input for the classifier model. The shape of the encoded data for each 7×7 segment is $2 \times 2 \times 64$. Algorithm 1 is an overview of the point anomaly detection process of a list of segments.

Algorithm 1. Point Anomaly Detection

```
1: procedure FindPointAnomalies(readings)
2:     segments ← SplitIntoSegments(readings)
3:     predicted_segments ← PredictWithCNN(segments)
4:     errors ← CalculateErrors(segments, predicted_segments)
5:     mean_error ← CalculateMean(errors)
6:     labels ← []
7:     for $i$ ← 1 to $N$ do
8:         if errors[$i$] > mean_error then
9:             labels.append(True) // True means a point anomaly has occurred
10:        else
11:            labels.append(False) // False means no point anomaly
12:        end if
13:    end for
14: end procedure
```

3.3 Contextual Anomaly Detection

Contextual anomaly detection entails checking whether a point anomaly is an anomaly or a valid value in the context of the other sensor values. To perform this analysis, we have employed an LSTM-based classifier model. We chose an LSTM as our Recurrent Neural Network (RNN) because it can handle long-term dependencies. This is advantageous to us since we are dealing with time series. A simple LSTM cell can be seen in Fig. 2. These cells' internal mechanisms, the gates, control the flow of information, allowing LSTM-based RNNs to capture and "remember" long-term dependencies in sequential data. The memory is stored in C_t, and the hidden state of the cell is stored in H_t where t denotes a time step. Several of these cells are chained together to form our LSTM RNN. The architecture of the LSTM classifier can be seen in Fig. 3.

Since this model takes a labelled dataset as an input for training, we must first label our unlabelled input. Contextual anomaly analysis must be done for each physiological

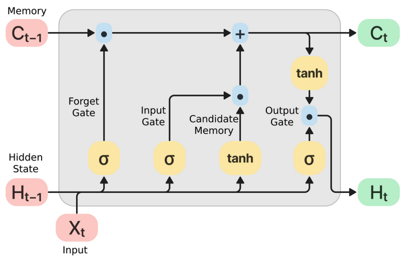

Fig. 2. A single LSTM cell.

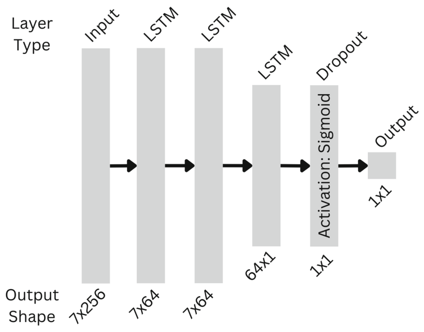

Fig. 3. An Overview of the classifier model

metric separately, as opposed to the point anomaly analysis. To check whether a point is a contextual anomaly, we have checked whether the transformed value of the correlated value of that point is greater than the standard deviation of the same across the entire series. This is done in the following steps:

1. For each point, the correlated value is calculated using the correlation coefficient from the correlation matrix. This is done for each metric, using Eq. 2 as given below.

$$\text{correlated\_value}_i = \sum_{n=1}^{7} \text{CC}_{n,i} \cdot R_n \tag{2}$$

   where $\text{CC}_{n,i}$ is the Correlation coefficient of metric n and metric i.
2. These correlated values are then transformed using Eq. 3

$$\text{anomaly\_score} = \frac{\text{correlated\_value}_i - \text{correlated\_value}_{mean}}{\text{correlated\_value}_{standarddeviation}} \tag{3}$$

3. Thus, any point with an anomaly score greater than the standard deviation of anomaly scores is considered a contextual anomaly.

This is done for all the points first. Note that even if the anomaly_score exceeds the threshold, it is not a contextual anomaly if a point is not already considered a point anomaly. Next, since the points have been compressed from 7×7 to 2×2, if any of the 7 rows is considered a contextual anomaly, the entire 2×2 compression is also labelled a contextual anomaly.

With these as the labels, our classifier is trained, and the LSTM classifier is obtained. Algorithm 2 describes how to train the LSTM model, and algorithm 3 describes how to use it to classify a list of encoded segments.

4 Results

To implement our model, the Tensorflow [10] and keras [11] Python libraries were utilized. These were chosen since they allow for easy modifications and troubleshooting

Algorithm 2. LSTM Model

```
 1: procedure TrainContextualAnomalies(readings)
 2:     segment_length ← 7
 3:     number_of_segments ← $\lfloor$length(readings)/segment_length$\rfloor$
 4:     segments ← []
 5:     encoded_segments ← []
 6:     anomaly_scores ← []
 7:     for $i$ ← 0 to number_of_segments − 1 do
 8:         start_index ← $i$ × segment_length
 9:         end_index ← start_index + segment_length
10:         segment ← readings.get(start_index to end_index)
11:         encoded_segment ← Encoder(segment)
12:         anomaly_score ← Calculate_Anomaly_Score(segment)
13:         segments.append(segment)
14:         encoded_segments.append(encoded_segment)
15:         anomaly_scores.append(anomaly_score)
16:     end for
17:     std_dev ← Standard_Deviation(anomaly_scores)
18:     labels ← Label_Anomalies(scores, std_dev)
19:     LSTM_classifier ← Train_LSTM_Classifier(encoded_segments, labels)
20: end procedure
```

Algorithm 3. Contextual Anomaly Detection

```
 1: procedure FindContextualAnomalies(encoded_segments)
 2:     labels ← []
 3:     for $i$ ← 1 to $N$ do // Loop through all $2 \times 2$ encoded segments
 4:         if Is_Contextual_Anomaly(encoded_segments[$i$]) then
 5:             labels.append(True) // True means a contextual anomaly has occurred
 6:         else
 7:             labels.append(False) // False means no contextual anomaly
 8:         end if
 9:     end for
10: end procedure
```

model creation and training. The scikit-learn library [13] was used to aid in dataset manipulation, such as creating train-test-validate splits and the correlation matrix. The Pandas [12] and Matplotlib [13] libraries were used to create diagrams and graphs.

First, the results of training the autoencoder are studied. To analyse the optimal specifications for training, the model is passed through varying epochs, and certain metrics for that model are graphed alongside it. The metrics chosen are the number of anomalies - while this does not provide much information regarding the model itself, it allows us to pinpoint diminishing returns, MAE - a good representative for an autoencoder output error, and MSE - chosen for the same reason as MAE.

As seen from Fig. 4, 5 and 6, after an initial steep decrease in the MAE and MSE, the models become more or less plateaued. After this point, overfitting begins; to avoid this, the number of epochs in the final model is set to 20.

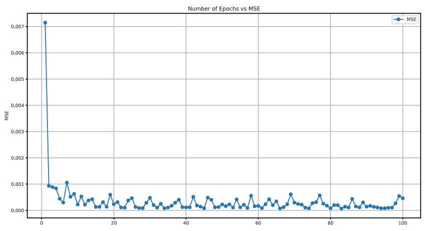

Fig. 4. MSE vs Epochs of training

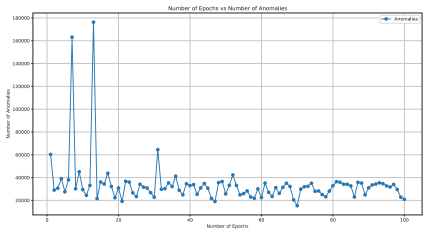

Fig. 5. Anomalies vs Epochs of training

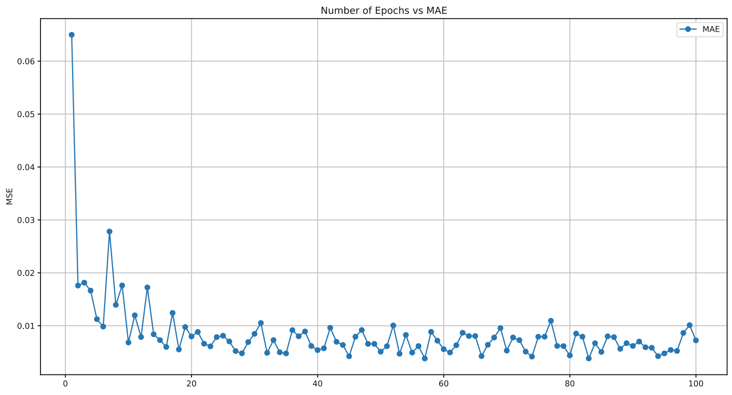

Fig. 6. MAE vs Epochs of training

With these as our parameters - 20 training epochs and a batch size of 128, we train the model. After reconstruction, at any point where the difference between the reconstructed value and the actual value is greater than the standard deviation, we mark it as a point anomaly. Figure 7, 8, 9, 10, 11 and 12 shows the results of the point anomaly detection process. Note that the figures show 500 time steps.

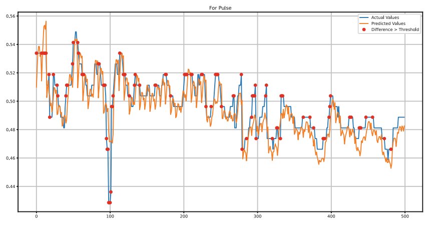

Fig. 7. Reconstructed Pulse

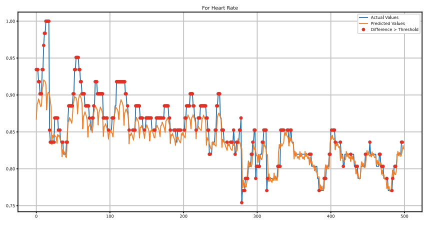

Fig. 8. Reconstructed Heart Rate

The results from Fig. 7, 8, 9, 10, 11 and 12 show that the reconstructions faithfully conform to the original data to a certain extent, and point anomalies can be discovered

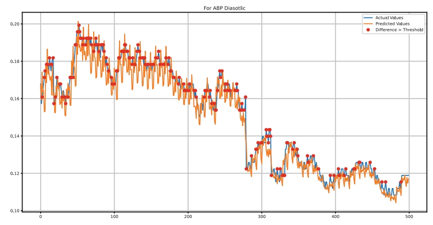

Fig. 9. Reconstructed ABPDia

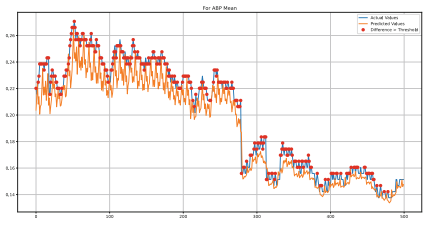

Fig. 10. Reconstructed ABPmean

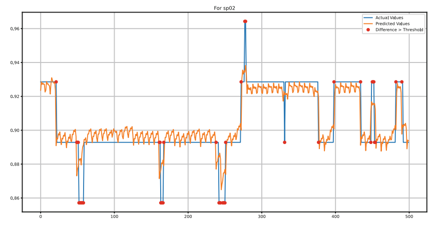

Fig. 11. Reconstructed SpO_2

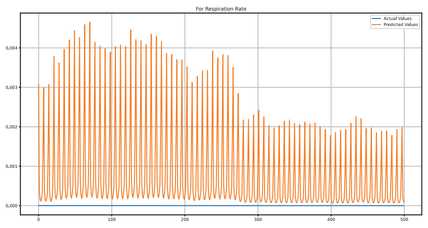

Fig. 12. Reconstructed Resp

using the CNN-based autoencoder. The encoded values from the autoencoder are then fed to the LSTM model as input.

Next, for training the LSTM model, yet again, we first aim to find an optimum number of training epochs. To do this, we use conventional evaluation metrics for classifiers: Accuracy, Precision, Recall, and F1-Score. We trained the model for varying numbers of epochs; the results of this can be seen in Fig. 13, 14, 15 and 16.

As can be seen from the graphs, the metrics all begin plateauing about Epoch 50. So, we use 50 as the number of epochs to train the LSTM model to avoid overfitting.

With our model trained, we can now classify contextual anomalies in our dataset. We must divide the input data into 7×7 sections. Then, these must be encoded into a 2×2 array using the encoder derived from our autoencoder. This is done to take advantage of the inherent feature extraction in autoencoders. These 2×2 arrays can now be classified using the LSTM model. The results of this classification for SpO_2 readings can be found in Fig. 17. Only SpO_2 readings are pictured for the sake of brevity, with other readings showing similar results.

As visible, the overlap between calculated and actual contextual anomalies is quite significant. The accuracy of the final model is 0.9400, and the loss of the model is 0.1457.

Furthermore, the classification report and the confusion matrix of performing testing on our classifier model can be seen in Table 1a and Table 1b. These results were obtained using the scikit-learn library [13].

Table 1. Performance metrics of the LSTM classifier model

(a) Classification Report of the LSTM classifier

Classes	Precision	Recall	F1-Score
Non-Anomalous	0.96	0.97	0.97
Anomalous	0.77	0.69	0.73

(b) Confusion Matrix of the LSTM classifier

		Label		
		Positive	Negative	Total
Prediction	Positive	471	140	611
	Negative	214	5231	5445
	Total	685	5371	6056

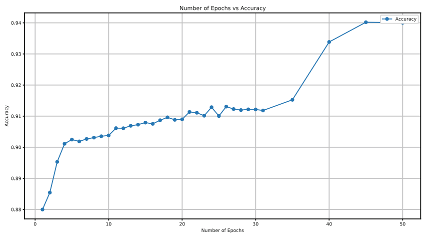

Fig. 13. Accuracy vs Epochs of training

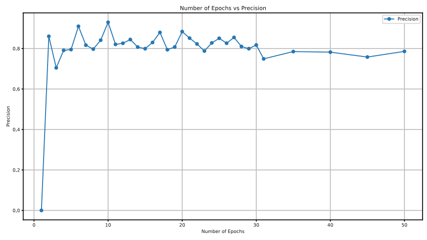

Fig. 14. Precision vs Epochs of training

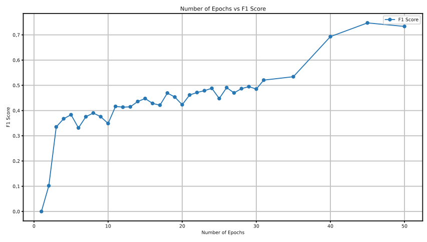

Fig. 15. F1 Score vs Epochs of training

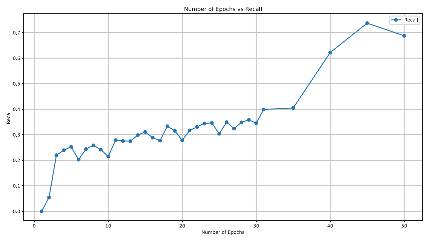

Fig. 16. Recall vs Epochs of training

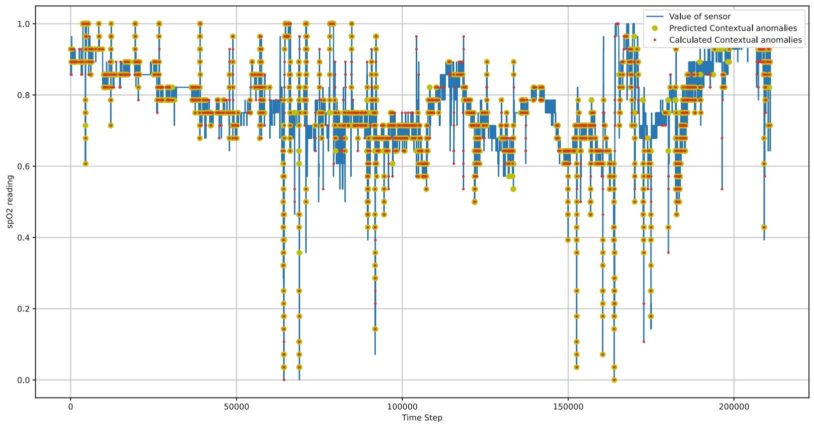

Fig. 17. A plot of classified contextual anomalies overlaid on SpO_2 readings.

5 Conclusions and Future Work

WBANs are a fast-developing field, and widespread usage of these will prove to be invaluable to the medical industry. As such, the detection of anomalous readings of these networks is an important task. This paper demonstrates that the usage of deep learning-based models can be helpful in this task. With our CNN-based autoencoder being able to detect point anomalies using an unlabeled dataset, we can detect and even replace anomalies. The LSTM classifier then further refines these anomalies to be contextually valid or not. Since accurate readings which appear anomalous must be allowed to pass through, the LSTM's task is vital. The model for detecting anomalies in SpO_2 shows an accuracy of 94% and a loss of 14%.

In the future, these models can be further refined with different paradigms to detect contextual anomalies, not just the anomaly score as implemented in this paper. With the use of feature extraction, this can be achieved in several ways. With newer and more complete medical records available as well, other metrics could also be considered beyond the seven used in this paper. Overall, deep-learning-based anomaly detection has enormous potential in the implementation of WBANs.

Acknowledgements. The authors would like to express their enormous gratitude to the BITS BioCyTiH Foundation and to DST, Govt. of India, for their support and funding provided for this research.

Data Availability Statement. The source code for the models described in this research is accessible on GitHub [9]. It has been implemented using TensorFlow [10], Keras [11], scikit-learn [13], and NumPy [8]. The Pandas Python library was used to help in data graphing [12].

References

1. IEEE Standard for Local and metropolitan area networks - Part 15.6: Wireless Body Area Networks. In: IEEE Std 802.15.6-2012, pp. 1–271 (2012). https://doi.org/10.1109/IEEESTD.2012.6161600
2. Harun Al Rasyid, M.U., Setiawan, F., Nadhori, I.U., Sudarsonc, A., Tamami, N.: Anomalous data detection in WBAN measurements. In: 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC), Bali, Indonesia, pp. 303–309 (2018). https://doi.org/10.1109/KCIC.2018.8628522
3. Canizo, M., Triguero, I., Conde, A., Onieva, E.: Multi-head CNN-RNN for multi-time series anomaly detection: an industrial case study. Neurocomputing **363**, 246–260 (2019). https://doi.org/10.1016/j.neucom.2019.07.034
4. Khan, F.A., Haldar, N.A.H., Ali, A., Iftikhar, M., Zia, T.A., Zomaya, A.Y.: A continuous change detection mechanism to identify anomalies in ECG signals for WBAN-based healthcare environments. IEEE Access **5**, 13531–13544 (2017). https://doi.org/10.1109/ACCESS.2017.2714258
5. Provotar, O.I., Linder, Y.M., Veres, M.M.: Unsupervised anomaly detection in time series using LSTM-based autoencoders. In: 2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, pp. 513–517 (2019). https://doi.org/10.1109/ATIT49449.2019.9030505

6. Zhai, J., Zhang, S., Chen, J., He, Q.: Autoencoder and its various variants. In: 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, pp. 415–419 (2018). https://doi.org/10.1109/SMC.2018.00080
7. Goldberger, A., et al.: PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation **101**(23), e215–e220 (2000). https://www.physionet.org/static/published-projects/circulation. Accessed 10 Sept 2023
8. Harris, C.R., et al.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2
9. Kartikeya, D.: Anomaly detection in WBANs. GitHub. https://github.com/kakuking/AnomalyDetectionInWBANs. Accessed 25 Oct 2023
10. TensorFlow Developers, TensorFlow. Zenodo (2023). https://doi.org/10.5281/ZENODO.4724125
11. Chollet, F., et al.: Keras. GitHub (2015). https://github.com/fchollet/keras
12. McKinney, W., et al.: Data structures for statistical computing in Python. In: Proceedings of the 9th Python in Science Conference, vol. 445, pp. 51–56 (2010)
13. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
14. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007)