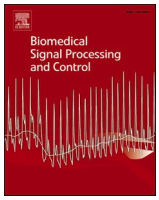
Biomedical Signal Processing and Control 101 (2025) 107155

Contents lists available at ScienceDirect

Biomedical Signal Processing and Control

journal homepage: www.elsevier.com/locate/bspc



Enhancing arrhythmia prediction through an adaptive deep reinforcement learning framework for ECG signal analysis

Mohamed Adel Serhani [a,*], Heba Ismail [b], Hadeel T. El-Kassabi [c], Hamda Al Breiki [d]

[a] Department of Information Systems, College of Computing and Informatics, University of Sharjah, Sharjah, United Arab Emirates
[b] Department of Computer Science and Information Technology (CS-IT), College of Engineering, Abu Dhabi University, Abu Dhabi, United Arab Emirates
[c] Faculty of Applied Sciences & Technology, Humber College Institute of Technology &Advanced Learning, Toronto, Ontario, Canada
[d] College of Technological Innovation, Zayed University, Abu Dhabi, United Arab Emirates

ARTICLE INFO

Keywords:
Agent
Arrhythmia
CNN
DEEP learning
ECG
Heart diseases
Reinforcement learning

ABSTRACT

Heart diseases, particularly arrhythmia, remain a leading global cause of mortality. Electrocardiography (ECG) serves as a vital tool for monitoring and mitigating arrhythmia risks. Continuous proactive monitoring entails processing ECG time series data, offering critical insights to prevent severe outcomes like heart attacks and strokes. While deep learning models are widely employed for large-scale ECG data analysis, their effectiveness requires intricate hyperparameter tuning and architecture adjustments. This study presents an innovative framework incorporating a reinforcement learning-based approach to automatically fine-tune hyperparameters for a Convolutional Neural Network (CNN) model, addressing resource consumption and complexity issues. Our proposed model optimizes hyperparameters and network configurations by maximizing a reward function, overcoming challenges associated with resource-intensive and complex deep learning models. To assess its efficacy, we compare our automated fine-tuning model against both a non-optimized baseline model and a manually fine-tuned model. The findings demonstrate that our model outperforms the alternatives, achieving a higher accuracy of 97.4 %, reduced execution time (0.33 min), and lower mean square error (0.0081). Moreover, a comprehensive evaluation of resource consumption illustrates the lightweight nature of our proposed solution, minimizing overhead and complexity. Our model consistently converges to an optimal configuration, establishing its efficiency in predicting arrhythmia. In conclusion, our reinforcement learning-based hyperparameter tuning approach presents a lightweight and efficient solution for optimizing CNN models in ECG analysis. The results underscore its superiority over non-optimized and manually fine-tuned models, positioning it as a promising method for enhancing arrhythmia prediction accuracy while minimizing resource consumption and complexity.

1. Introduction

The cardiopulmonary system measures electrical impulses that control heartbeat movement via specialized regions of patient's heart [1]. Therefore, distinguishing distinct types of electrical impulses that drive hearts, such as sinus node vs pacemaker rhythms, aids doctors in treating patients accurately since the excessive delay between successive ventricular fibrillations can cause arrhythmias such as atrial fibrillation, which puts patients at risk for stroke, dementia, and other cardiovascular complications [1–3]. However, the visual monitoring of ECG is time-consuming and prone to false detection. For instance, cardiac arrhythmias are irregularities in a patient's heartbeat rate or rhythm caused by abnormal electrical impulses that influence the heart's electrical system [1]. The heart can beat either too fast or slow during arrhythmias or with an irregular rhythm. Their severity can range from benevolent to life-threatening, as the heart may be incapable of pumping sufficient blood to the rest of the body throughout an arrhythmia. Beside, arrhythmia is so common and silent in patients [2,3], numerous studies have long attempted to propose automatic prediction systems that can provide early warning to patients on any arrhythmia-related symptoms to prevent undesirable sudden complications at a later stage [4–7]. Electrocardiography (ECG) signals to play a vital role in these automatic prediction systems. However, ECG signals can be overwhelming, noisy, and hard to analyze.

* Corresponding author.
E-mail addresses: mserhani@sharjah.ac.ae (M.A. Serhani), heba.ismail@adu.ac.ae (H. Ismail), Hadeel.El-Kassabi@humber.ca (H.T. El-Kassabi), Hamda.AlBreiki@zu.ac.ae (H.A. Breiki).

https://doi.org/10.1016/j.bspc.2024.107155
Received 18 March 2024; Received in revised form 16 August 2024; Accepted 2 November 2024
Available online 21 November 2024

Consequently, the need for several preprocessing phases to prepare ECG signals for use in prediction models is accentuated. For instance, denoising and sampling [4–6], heartbeat detection [8], segmentation [9], and feature extraction [10,11] are vital preprocessing phases in almost all arrhythmia prediction systems. Additionally, prediction model fine-tuning is another challenge that arrhythmia prediction systems must overcome. Notably, for deep learning prediction models, several model hyperparameters, configurations, and various structures impact the accuracy of the prediction model and the validity of the produced predictions. Thus, many researchers tested several configurations as part of a manual fine-tuning process until they obtained an optimum configuration with reasonable accuracy [6,4,12]. However, manual fine-tuning is time-consuming and is never optimum as researchers cannot exhaust all possible configurations and structures. Therefore, some researchers have adopted an automatic fine-tuning approach [11,13]. Yet, automatic fine-tuning has been mainly applied to model weights and not to the structure in these research works. Therefore, herein, we propose a fully automated fine-tuning deep-learning-based arrhythmia prediction system. We implement deep reinforcement learning (RL) in the proposed model to automate the fine-tuning process of the model's hyperparameters, weights, and structures. The proposed adaptive arrhythmia prediction model implements an adaptive deep RL module to optimize the selection of the deep learning prediction model's hyperparameters and structures using a list of all possible values for the model's parameters and the network architecture as the input. The proposed model relies on an agent component that automatically fine-tunes these parameters based on a reward evaluation mechanism. Moreover, we provide a formal definition of the proposed model and an empirical evaluation based on the MIT-BIH dataset. Our experiments show that the proposed adaptive prediction model achieves a high accuracy of 97.4 % with a cross-entropy of 0.0081.

The paper is organized as follows. Section 2 thoroughly reviews arrhythmia, preprocessing phases, prediction models, and automatic fine-tuning methods. Section 3 overviews the proposed reinforcement-based model and its main components. Section 4 details the problem formulation based on RL for convolutional neural network (CNN) model optimization-based hyperparameters and network structure. Section 5 describes the experiments, including the environment setup, dataset preprocessing, processing and analytics, experimental design, and results discussion. Finally, Section 6 concludes the paper and presents topics for future research investigations.

2. Background and literature review

2.1. Automatic prediction/detection of cardiac arrhythmias based on ECG signal

Several research studies have long focused on designing approaches for automatically detecting arrhythmias from ECG signals. Typically, the process is carried out through multiple phases to extract representative features from raw ECG signals. Fig. 1 depicts the common phases of the detection process.

First, raw signals are denoised to eliminate interference from other sources of electromagnetic radiation and baseline wanderings stemming from patient movement or respiration that can alter the detected ECG peaks [4]. Several denoising techniques have been used in the reviewed literature including the deployment of a FIR bandpass [14], or Butterworth low pass filters [15]. Meanwhile, other researches use a combination of median and low pass [4], median, low pass, bandpass and adaptive [6], moving average, comb, and high pass [7] filters, Daubechies wavelet filters [5,16,14], or wavelet-based thresholding [17].

Fig. 1. Common process of automatic arrythmia detection using ECG signals.

Second, a common preprocessing step in the majority of the reviewed deep learning models is heartbeat detection. Some models require the detection of the entire heartbeat [6]. On the other hand, other approaches require the detection of only some particular portions of the heartbeat such as the peaks [4], T-waves [18], or the QRS complex [5,7] using Pan-Tompkins algorithm [8].

Third, once the peaks are detected, the ECG signal is segmented into several heartbeats. The heartbeats are either segmented starting from the P-peaks until the previous T-peak [4], centered at the R-peaks [5,9], or using the segmentation software to obtain the QRS onset, T-wave offset times, and the P-wave onset and offset times, if present [19]. Other approaches segmented based on grouping the continuous arrhythmia beat sequences into the appropriate arrhythmia groups [20] or using frame blocking [15].

Finally, after the ECG signal is segmented, several features are extracted to classify a heartbeat into either normal or abnormal. These features include temporal features such as Pre-RR interval, Post-RR interval, Local average R-R interval, and Global average RR interval of a beat [4,14], or superficial features on lower sampling rates to decrease power consumption and hardware complexity [6]. Other approaches utilized transfer learning to extract features. For instance, AlexNet, which requires data conversion from ECG signals to binary images, is used for feature extraction and heartbeat classification [7]. Moreover, discrete wavelet transformation is applied to the input layer for feature extraction [11]. To automate the preprocessing and feature extraction phases, some approaches proposed end-to-end structures of layers to produce feature maps of the ECG signals [12]. Furthermore, some approaches combined several feature extraction techniques including 1D-Local Binary Pattern (1D-LBP), Higher Order Statistics (HOS), Central Moment, Hermite Basis Function (HBF), Discrete Wavelet Transform (DWT), and R-R Intervals [14].

Data quality and resource consumption are major concerns of the arrythmia classification frameworks. Hence, some approaches considered improving data quality through data normalization [9,18]. For instance, Z-score normalization [9] is used to reduce the offset effect and equalize the amplitude of the ECG signals to reduce measurement variations. Moreover, to address resource consumption concerns, normalization [9], compression [13], or input length standardization [20] are performed to speed up the training process. Furthermore, compression is performed to decrease digital storage space consumed and secure patient information [13].

ECG datasets for arrhythmia are imbalanced, which complicates the classification process. To address this challenge, data augmentation and synthesis is used to negate the skewing effects of its data imbalance. Some approaches augment the minority classes by changing the standard deviation and mean of the Z-score of the original ECG data [5]. Other approaches deploy the synthetic minority over-sampling method (SMOTE) [21,22,14], to synthesis data samples for minority groups [18]. Moreover, some approaches proposed applying eight cropping operations on the ECG spectrograms obtained by applying Short-Time Fourier Transform (STFT) [17].

2.2. Deep learning-based models with manual fine-tuning

Using data from the MIT-BIH arrhythmia database, Sannino et al. [4] trained a DNN to classify ECG signals as normal or arrhythmic beats, specifically premature ventricular contractions, supraventricular premature beats, fusions of ventricular and normal beats, or unclassifiable beats. They first preprocessed the data by denoising, peak detection, and signal segmentation. After that, the data is fed into the DNN, which instantly creates a decision-making function after extracting temporal characteristics and classifying them into a signal learning body.

A study by Yıldırım et al. [12] proposed a 1-D CNN that classifies 17 different classes of cardiac arrhythmia. After adjusting the number and

type of layers, a 16-layer CNN that combines feature extraction with the selection and classification processes was deployed. No other preprocessing was performed on the data used from the MIT-BIH arrhythmia database. According to state-of-the-art standards, this 1-D CNN is efficient, fast, and simple to use.

Mathews et al. [6] classified preprocessed single-lead ECG signals using the restricted Boltzmann machine (RBM) and deep belief networks (DBN). They extracted the features from the signals using two methods to produce and evaluate the representativeness of the two feature sets. The features were RR intervals, heartbeat intervals, segmented morphology intervals, and fixed-interval morphology. Feature Set 1 (FS1) comprises RR intervals, heartbeat intervals, and segmented morphologies with a total of 26 features, while Feature Set 2 (FS2) comprises RR intervals and fixed interval morphologies with a total of 22 features. Using deep learning networks, high average recognition accuracies of arrhythmic heartbeats, specifically ventricular ectopic beats and supraventricular ectopic beats, can be achieved at a low sampling rate and using simple features. This reduces the computational cost of training the DBN.

To diagnose cardiac arrhythmia using deep learning techniques with as little preprocessing as possible, Swapna et al. [9] employed deep learning networks to create an automated, non-invasive system that can classify ECG signals as normal or abnormal, i.e., arrhythmic. The proposed approach used a combination of deep learning architectures such as CNN, CNN-LSTM, CNN-RNN and CNN-GRU. Their classification system is composed of a specific, fine-tuned sequence of the multi-layered networks and hybrids of the networks, specifically CNN-RNN, CNN-LSTM, and CNN-GRU. Notably, the only preprocessing step taken to prepare the raw data for the classification was heartbeat extraction.

Isin et al. [7] used a two-step process to classify ECG signals as "Normal," "Paced," or "Right Bundle Branch Block". First, the signals are preprocessed by removing noises, detecting their QRS complex, and automatically extracting the features to be analyzed by the AlexNet CNN [23]. As proposed, in the second step, the extracted features from the CNN are run through a simple, three-layer feed-forward neural network trained with a scaled conjugate gradient backpropagation algorithm.

Acharya et al. [5] proposed a convolutional network that classifies a heartbeat as non-ectopic, supraventricular ectopic, fusion, and unknown beats. These are general categories that encompass more specific types of beats. The feature that distinguishes this CNN model from traditional CNNs is its preprocessing, precisely, the synthesis of noise-free ECG imbalanced data that generates balanced data to increase the accuracy. This model was evaluated using two sets of synthesized data. Notably, both datasets perform similarly despite the absence of any denoising in the first set. This is because the algorithm learns the proper filters and automatically eliminates the unwanted noise in the ECG data.

Recently, Li et al. [24] implemented Arrhythmia detection and classification based on deep learning models. The S-shaped reconstruction approach, which they proposed, uses a two-dimensional, deep squeeze-and-excitation residual network (SE-ResNet) to identify heartbeats. The model was trained on a processed ECG dataset. They performed data preprocessing by denoising the initial electrocardiogram (ECG) data, eliminating baseline drift, extracting heartbeats, and balancing the data using a SMOTE approach algorithm. Subsequently, using the advanced S-shaped reconstruction approach to ascertain the connection between distant points in an ECG series, the recovered one-dimensional heartbeat series is converted into a 2-D matrix. Henceforth, 2D heart-based matric is divided using SE-ResNet into five distinct ECG heartbeat categories, i.e. (normal, ventricular ectopic, fusion, supraventricular ectopic, and unknown beats). Although the predictive model showed high performance, the proposed algorithm's temporal complexity has significantly increased.

Pandey and Janghel [25] proposed a Bidirectional long short-term memory (BLSTM) network classifier for the early detection of cardiac arrhythmia. The proposed system employs a deep convolutional encoded feature based on a non-linear compression component for ECG signal segment size. In addition, three different classifiers—the multilayer perceptron, gated recurrent unit (GRU), and unidirectional long short-term memory (ULSTM) network—are developed for performance comparison. The experimental investigations identify and group the five distinct cardiac classes of arrhythmias seen in the MIT-BIH arrhythmia database. Herein, the BLSTM network outperformed the other classifier. However, they did not consider several preprocessing methods, which decreased the quality of training data.

Aphale et al. [16] proposed deploying a pre-trained EfficientNet B7 model for transfer learning with fine-tuning. The denoised ECG data is transformed using Daubechies Wavelet and is converted to 2D images before training the transfer learning model. Both manual and automatic fine-tuning processes are carried out to improve the generalization and performance of the transfer learning model. Network structure including the model's depth, layers, and activation functions are tuned manually. Meanwhile, the model weights are tuned using the Adam optimizer. Following that, 50 % of the layers of the pre-trained network are unfrozen for fin-tuning on a small validation dataset. The proposed network is evaluated on multi-class arrhythmia variations including 17-class, 15-class, 13-class, and 12-class 2-D heartbeat images. Each fine-tuned multi-class experiment is evaluated against the raw model (without fine-tuning). Fine-tuned transfer learning networks yield better results, indicating improved overall performance.

Li et al. [15] proposed utilizing ResNet with the attention-based BiLSTM model for automatic arrhythmia classification. The kernel parameters of the RNN are fine-tuned manually from a pre-defined set, while the learning rate is tuned using the Adam optimizer.

2.3. Deep-learning-based models with automatic fine-tuning

A coded feature-based deep LSTM network was proposed by Yildirim [13] to classify heartbeat data from the MIT-BIH arrhythmia database as normal or abnormal. Using this LSTM, the possible classifiable abnormal classes are atrial premature beats, left bundle branch blocks, right bundle branch blocks, and premature ventricular contractions. To prepare the ECG signals for classification, they are first compressed using a CAE comprising two parts: an encoder and decoder. Next, the CAE extracts features of the heartbeat to be analyzed, including QRS morphology, degree of the intraventricular block, and time interval of the beat to the preceding beat. After compression and feature extraction, the signals are automatically classified using the LSTM network.

2.4. Limitations in existing literature

Despite significant advancements the prediction accuracy of the detection of cardiac arrhythmias using ECG signals, several notable gaps exist in the literature, particularly regarding the predictive models by Aphale et al. [16] and Li et al. [15], one primary shortcoming is the resource-intensive nature of the fine-tuning process of these methods. For instance, Aphale et al. [16] and Li et al. [15] proposed deep learning models that involve extensive manual fine-tuning, including adjustments to network structures, activation functions, and learning rates. These processes, while improving accuracy, require substantial computational resources, making them less practical for real-time or resource-constrained environments. Furthermore, the optimization algorithms employed in some studies often prioritize accuracy at the expense of efficiency, without sufficiently addressing the trade-offs between resource consumption and model performance. For instance, Li et al. [24] noted that although their deep squeeze-and-excitation residual network (SE-ResNet) achieved high predictive performance, the algorithm's increased temporal complexity rendered it resource-intensive, limiting its applicability in settings with constrained computational resources.

Another significant limitation in the existing literature is the focus on binary classification (i.e., normal vs. abnormal) or the emphasis on majority classes of arrhythmia. Many studies, such as those by Sannino

et al. [4] and Yildirim [13]., target only two-class predictions, while others, like the work of Acharya et al. complex [5], focus on distinguishing between a few majority arrhythmia classes. This approach overlooks the importance of accurately predicting minority arrhythmia classes, which are often clinically significant but underrepresented in the datasets.

The proposed method in this study addresses these gaps by introducing a resource-aware approach to fine-tuning predictive models for arrhythmia detection. By leveraging a multi-objective optimization algorithm, the method aims to optimize accuracy while maintaining low resource consumption, making it more suitable for real-time applications in resource-limited environments. Additionally, the proposed approach is designed to handle multiclass predictions, including the identification of minority arrhythmia classes, thereby advancing the state of the art in arrhythmia prediction and offering a more comprehensive and practical solution for real-world deployments.

The novelty of the proposed approach by Oh et al. [20] is the input length standardization of the segments used to train their hybrid CNN-LSTM network. An arrhythmia can occur in single or multiple beats, necessitating the ability to process signals of variable lengths. Once the data were preprocessed, they classified them as normal, left, or right bundle branch block, atrial premature beat, or premature ventricular contraction by a hybrid CNN-LSTM network. No feature extraction was necessary because convolution layers generated the feature maps, and the LSTM layer extracted temporal information about the features. The proposed approach used ten-fold-cross validation to optimize the network and automatically adjusted the layer weights and dropout schemes.

In a study by Yildirim [11], several deep long-short terms (DLSTM) network models were designed and tested on the MIT-BIH arrhythmia database to obtain the best performing classification deep learning network. The four tested DBLSTMs were unidirectional or bidirectional DLSTMs with or without wavelet sequence (WS) layers: DULSTM, DULSTM-WS, DBLSTM, and DBLSTM-WS. The most accurate network was the DBLSTM-WS network, with a layer parameter of 2. Before the rest of the network classified the segmented ECG signals, the WS layer extracted their features by repeatedly filtering until the features containing important information about the input signals were obtained. Then, the data were passed through the LSTM for classification as normal sinus rhythm, left bundle branch blocks, right bundle branch blocks, atrial premature beats, or ventricular premature contractions. Notably, this approach avoided overfitting by using a dropout layer instead of a dropout scheme, similar to Oh et al. [20].

Houssein et al. [14] proposed the implementation IMPA-CNN, which is a CNN with a modified Marine Predators Algorithm (MPA) for automated fine tuning. This model extracts the major features from the raw signals to reduce the time and computational resources. The modified MPA optimizes the learning rate and optimizer type and activation function from pre-defined sets. The proposed model is evaluated on 3 well known datasets.

2.5. Discussion of literature review

As previously discussed, deep learning models are commonly used for arrhythmia prediction, as summarized in Table 1. However, a significant challenge remains in determining the optimum configuration. As illustrated in Table 1, most reviewed research works rely on manual fine-tuning wherein researchers attempt several possible configurations to choose the best performing configuration. However, this approach is exhaustive and costly as it involves intensive human intervention. To address such limitation, some researchers proposed automatic fine-tuning of hyperparameters [20] using Adam optimizer [11,13] or brute force technique [13]. However, research works targeting automatic fine-tuning have a limited number of optimized parameters and mainly focus on weights fine-tuning, while the structure is usually manually fine-tuned. To address this limitation, a fully automated, adaptive prediction model that optimizes the hyperparameters and structure is proposed herein. Details of the proposed system and empirical evaluation are presented in subsequent sections.

Table 1
Research work using deep learning prediction models for predicting arrhythmia.

Work	DL Model	Type of Fine-Tuning	Performance (Accuracy %)
[5]	Deep Convolutional Neural Network (CNN)	Manual (Brute force technique)	Set A (no denoising): 93.47 Set B (with denoising): 93.47
[6]	Deep Belief Network (DBN) and Restricted Boltzmann Machine (RBM)	Manual	**FS1:** −Supraventricular Ectopic Beats (SVEBs) Class: 93.63 −Ventricular Beats (VEBs) Class: 95.57 **FS2:** SVEBs Class: 93.42 − VEBs Class: 95.87
[7]	Transferred Deep Convolutional Neural Network (a CNN called AlexNet) and Back Propagation Neural Network	Manual	N-Fc6: 91.2 N-Fc7: 92.4
[9]	System of CNN, CNN-RNN, CNN-LSTM and CNN-GRU Hybrid Networks	Manual	83.4
[10]	Convolutional Neural Network (CNN)	Manual	97.25
[12]	1 D-Convolutional Neural Network (1 D-CNN)	Manual (Brute-Force technique)	91.33
[4]	Deep Neural Network (DNN)	Manual	99.09
[20]	Hybrid of CNN and LSTM Networks	Automatic and Manual	Dropout Scheme A: 97.88 Dropout Scheme B: 98.10
[16]	Pre-trained EfficientNet B7 model	Manual and Automatic (Adam optimizer)	17-class: 98.69 15-class: 98.09 13-class: 99.23 12-class: 99.17
[15]	ResNet in combination with attention-based bidirectional long short-term memory (BiLSTM)	Manual and Automatic (Adam optimizer)	93.14
[13]	Coded Feature-based Deep Long-Short Term Memory (LSTM) Network	Automatic (Adam optimizer and brute force technique)	−Deep LSTM Network: 99.23 −Deep CAE-LSTM Network: 99.11
[11]	Deep Bidirectional LSTM Network containing wavelet sequence layers (DBLSTM-WS)	Automatic (Adam optimizer)	DULSTM-WS3 (deep unidirectional LSTM): 99.25 DBLSTM-WS2: 99.39 DBLSTM-WS3: 98.85
[14]	Marine Predators algorithm (MPA) and CNN (IMPA-CNN)	Automatic (Marine Predators algorithm (MPA))	MIT-BIH: 99.33 EDB: 99.75 INCART: 99.43
[4]	Convolutional Neural Network (CNN) and RNN “Sequence to sequence” Model	N/A (Fine-tuning is not described)	Inter-patient paradigm used for testing (Intra-patient paradigm used for training ONLY): 99.53
[24]	2-D 19-layer SE-ResNet	N/A (Fine-tuning is not described)	99.61
[25]	Bi-LSTM	N/A (Fine-tuning is not described)	99.52

3. Adaptive reinforcement learning architecture for arrhythmia classification

In this section, we describe our design for an architecture for

arrhythmia classification deep learning model optimization and fine-tuning-based RL, as shown in Fig. 2. The architecture comprises three main modules: 1) ECG preprocessing module, 2) ECG processing module, and 3) adaptive deep RL module Fig. 3. These modules implement a set of processes starting from collection and preprocessing of raw ECG data, ECG feature extraction, and deep learning model optimization to finally classification of Arrhythmia.

Below we describe each module of the architecture and highlight the features contributing to the final research aim, which is the accurate classification and prediction of various arrhythmia classes.

ECG preprocessing module: The input of this module is a raw ECG signal, which is responsible for cleaning, denoising, and artifact removal from raw ECG recordings (e.g., powerline noise interference). The preprocessing activities are important for improving the signal's quality, readability, and accuracy.

ECG Processing module: This module uses the preprocessed ECG signal as input and processes it to extract main features that will be used later for ECG signal classification. This module encompasses three sequential activities: R-peak detection, hearth beat segmentation, and temporal feature extraction. R-peak detection comprises the analyzing of the ECG signal and detecting R-peaks and annotating these along the ECG recordings, it is widely used to diagnose cardiac disorders and analyze heart-rate variability. However, heartbeat segmentation identifies individual heartbeats and selects a series of samples (segments) centered on the R-peak of each heartbeat. Finally, the temporal feature extraction comprises the computation of each beat's temporal features, which helps maintain useful information about beats for classification. Such features include instance pre-RR and post-RR intervals and amplitude of P-, R-, and T-waves.

Adaptive deep reinforcement learning module: This module is the core of the architecture and is responsible for optimizing the deep learning model to be used for classifying preprocessed and processed ECG signals. It employs the list of hyperparameters, both model and network architecture parameters (see. Table 2), as input to the RL module and relies on an agent component that automatically fine-tunes these parameters based on the reward evaluation mechanism. The selected hyperparameter configuration is provided as input to the deep learning model that reward trains the ECG dataset using the selected configuration and then tests and evaluates the model accuracy using a set of metrics (e.g., cross-entropy). Observations and feedbacks are returned to the RL component, which re-evaluates the rewards and instructs the agent to fine-tune the corresponding hyperparameters. Iterations and loop back between model training and RL components continue until the deep learning model is optimized after the reward value reaches a certain threshold, as defined later.

4. Problem formalization

The manual selection of the optimal set of hyperparameters and neural network architecture parameters is an incremental and repetitive process that is time-consuming and requires exhaustive trials and experimentation as well as error handling to attain the best performance.

This study aims to automatically optimize the performance of CNN with application on ECG classification. Accordingly, herein, we define the CNN performance considering the model accuracy, processing cost, and execution time. An optimum model is one with high accuracy, low processing cost, and reduced execution time. For that purpose, our approach optimizes the set of training hyperparameters and the network-structure design-related hyperparameters called network hyperparameters of the CNN, including topology information.

4.1. Training hyperparameters set

Our first optimization level is the selection of a set of hyperparameters that achieve the best performance according to the above-mentioned performance definition, including the learning rate, momentum, number of epochs, and batch size [26,27]. The learning rate determines the updating speed of the network parameters, which is defined as $lr \in \{0.01 - xyz\}$. Additionally, the momentum m determines the direction of the next step given the previous steps, where $m \in \{0.5 - 0.9\}$. Moreover, the number of epochs is determined by monitoring the loss values during training using the early stopping call back function. Additionally, the batch size bsz is defined as the number of data samples fed to the network at the time of parameter update, where $bsz \in \{32, 64, 128, 256\}$.

4.2. Network structure hyperparameters set

Most of the approaches for optimizing the network structure design of neural networks in literature adopt plain CNN based on general architecture, which comprises convolution, pooling, and fully connected layers [28]. In our model, we employ the following network structure hyperparameters of CNN: the number of hidden units, dropout, network weight Initialization, and activation function [26]. For each layer type, we define the number of hidden units as $nhu > 1$, dropout as d, where $d \in \{20\% - 50\ \%\}$, and network weight initialization nw, which is a uniformly distributed set of weights. We use the most used activation functions AF, such as Rectified Linear Activation (ReLU), Logistic (Sigmoid), or Hyperbolic Tangent (Tanh), and SoftMax activation functions.

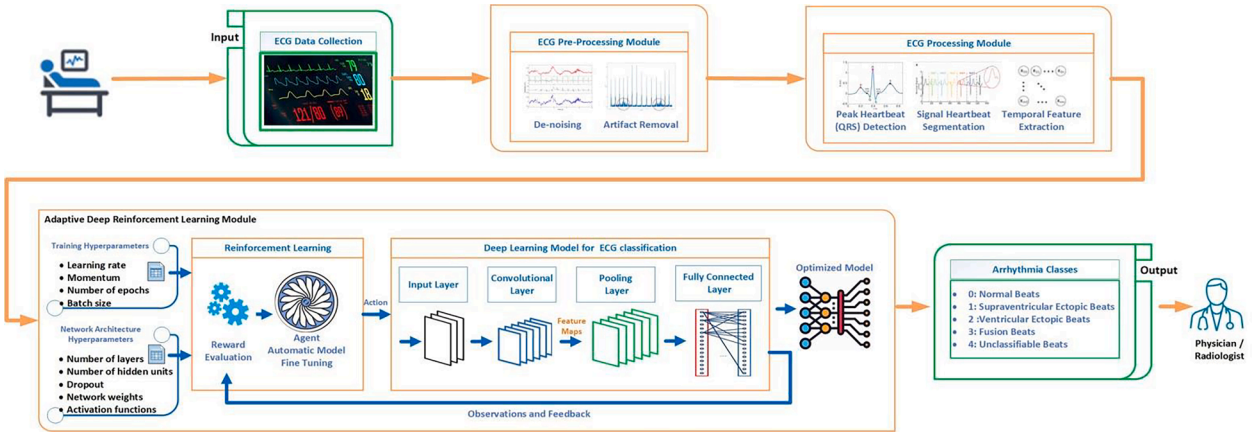


Fig. 2. Adaptive reinforcement learning architecture for arrhythmia-classification-based ECG data.

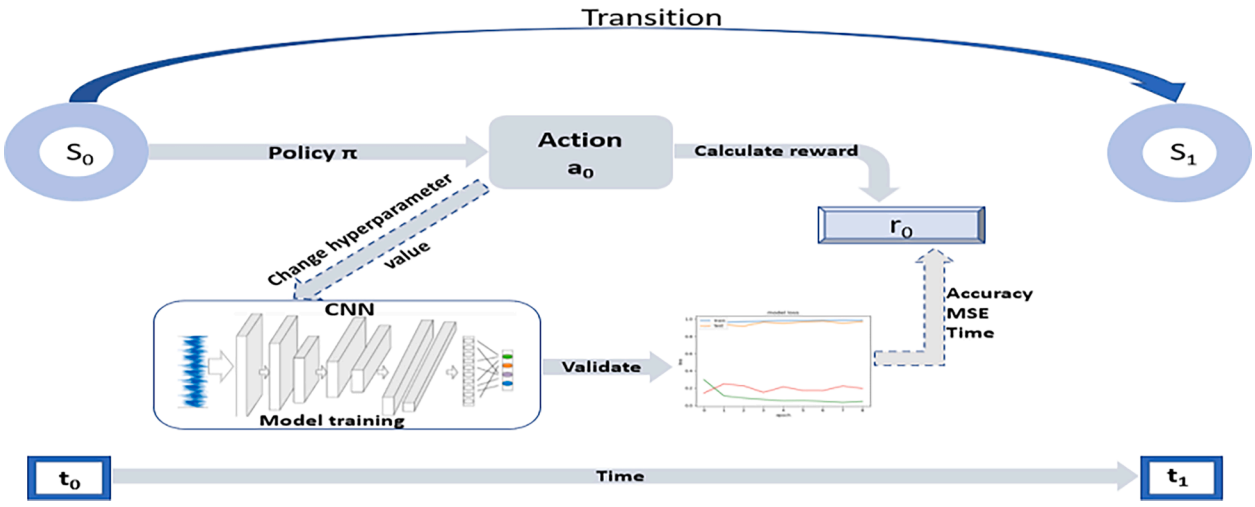


Fig. 3. MDP time step.

Table 2
CNN training and network structure hyperparameters.

	Hyperparameters	Value Ranges
CNN Training	Learning rate (***lr***)	$\boldsymbol{lr} \in \{0.01 - \boldsymbol{xyz}\}$
	Momentum (***m***)	$\boldsymbol{m} \in \{0.5 - 0.9\}$
	Number of epochs (***nep***)	$\boldsymbol{nep} > 0$
	Batch size (***bsz***)	$\boldsymbol{bsz} \in \{32, 64, 128, 256\}$
CNN Network Structure	Number of hidden units (***nhu***)	$\boldsymbol{nhu} > 1$
	Dropout (***d***)	$\boldsymbol{d} \in \{20\% - 50\ \%\}$
	Network weight initialization (***nw***)	$\sum_{i=1}^{n} \boldsymbol{w_i} = 1$
	Activation functions (***AF***)	ReLU, Sigmoid, Tanh, and other SoftMax activation functions

4.3. Reinforcement learning for deep learning (CNN) model optimization

We formally define D to denote a deep learning algorithm with a set of configurations of all hyperparameters ϵ, including discrete and continuous dimension spaces. Let D_μ be D with a set of training and network hyperparameters μ, where $\mu \in \epsilon$. We propose a method that applies RL to solve our deep learning optimization problem as a powerful learning decision-making framework [29]. Conventional RL problems are modeled as a multi-objective Markov decision process (MDP) as a sequential decision process, where the agents are sequentially employed to select the hyperparameters one by one based on a reward system that appreciates when the system performance improves. Generally, a problem with a large input space can be effectively solved by breaking the problem into multiple smaller subproblems to be easily handled, and consequently, the search space is reduced.

The RL-based framework can be described as follows: Each task has a set of n configuration parameters, including the hyperparameters α and the network architecture parameters β. The agent individually chooses the hyperparameters. Then, the deep learning model reflecting the assigned hyperparameters is trained on the training dataset denoted as D_{train}. The agent aims to maximize the expected performance of the deep learning algorithm by maximizing the expected reward. The system performance is measured as defined in the previous section and is used as the reward R to update the agent's internal parameters, which are the training and network structure hyperparameters. Moreover, we measure the accuracy of the deep learning model on the validation dataset, denoted as D_{valid}; the architecture complexity and execution time are both used as a reward to update the agent's parameters of the RL algorithm. Accordingly, the agent learns how to tune the hyperparameters following an iterative process.

We formalize our optimization problem as a multi-objective MDP with a 5-tuple (S, A, P, R, γ), were.

- S is the set of all states $S = \{s_1, s_2, s_3 \ldots s_m\}$. The agent selects one hyperparameter at a time, based on the previous past decisions. Thus, we define a state s at time t as $s_t = \text{Decision}_{t-1}(\mu_{t-1})$, where Decision_{t-1} is the decision of the agent at time $(t-1)$.
- A is the set of all actions $A = \{a_1, a_2, a_3 \ldots a_m\}$. For example, an action could denote choosing a certain value for hyperparameter h to be applied to the model and accordingly reaching a certain state.
- P is the set of probabilities of the transition between states at time t. It is the probability of the transition from s→s′ under action a at time t: $P_a^t(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$

The agent's action follows a policy that defines the state-action value π: S → A [29]. The objective of the agent is to choose a policy that maximizes the expected reward $.\pi(a, s) = \Pr(s_t = s, a_t = a)$

- $\boldsymbol{R}$ is the reward function, which is measured after the transition of **s**→**s′** under action **a** at time **t**. Since we adopt a multi-objective MDP, the reward value signal is defined as a vector comprising the feedback values of multiple objectives rather than a scalar reward in single-objective MDP. Therefore, we define our reward function as a weighted sum of all performance factors, such as accuracy, network architecture complexity, and time of execution formalized as follows:
- We evaluate the model's classification performance using cross-entropy loss. This method is particularly effective because the policy is represented as a probability distribution over actions. Cross-entropy loss helps adjust this distribution to better align with the optimal policy, which is defined by the following formula:

$\text{Loss} = -\sum_t \log\pi(a_t|s_t) \cdot R_t$, where π is the policy, a_t is the action taken, s_t is the state, and R_t is the return reward at time t.

- We also evaluate the accuracy Acc_t for the model in state s' at time step t as follows:

$$Acc_t = \frac{true\ positives}{total\ observations}$$

- T is the total time CNN takes to train and test the model to produce a result for all time steps from $t = 0$ to time step $t = n$, where n is the total number of time steps. It is defined as $T = \sum_{t=0}^{n} e_t$, where e_t is the execution time at time step t.

- The reward at time step t is a scalar that is defined as the weighted normalized value and is calculated using the following reward function: $r_t = \sum_{i=1}^{n} w_i * o_i$, s.t. $\sum_{j=1}^{n} w_j = 1$, $o_i \in \{Accuracy, T, logloss\}$. The weight w_i is assigned for each attribute o_i contributing to the reward evaluation based on the experimental evaluation. The details of the weight values will be elaborated in the experimentation section.
- γ is a discount factor, and $\gamma \in [0, 1]$. It is used to balance between the current and future rewards by determining how much weight is to be assigned to the long-term reward compared to the instant reward. When this value is closer to zero, the agent maximizes the immediate reward. However, when it approaches one, the agent assigns more weight to the current reward.

The below figure depicts the MDP at each time step t. The probability of the transition to s_{t+1} at time t is defined by $Pr(s_t, a_t, s_{t+1})$ using policy $\pi(s_t, a_t)$, and the reward is given by the reward function $r(s_t, a_t, s_{t+1})$. The action a_0 involves first training the CNN models using the updated hyperparameter values and then evaluating the model using a set of performance metrics.

4.4. Training with reinforce

We define the state-value function Q as the expected return at a certain state s after transition by action a under policy π. It measures how good the new state of the model is, which is state s at time step t.

$$Q_\pi(s, a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s,\ a_0 = a\right]$$

Algoithm 1 Multi-objective Reinforcement Learning for ECG classification

INPUT:
- *hpList* {List of hyperparameters},
- *hpValues* {List of hyperparameters value ranges},
- D_{train} {Training dataset},
- D_{valid} {Validation dataset},
- δ {*Threshold value for accepted Reward*}

OUTPUT:
- optHP List {Optimized hyperparameters List}

```
procedure OptimizeCNN(hpList, hpValues, D_train, D_valid)
  Reward_max ← 0
  while Reward_max < δ do
    optHPList ← defaultHP
    M ← initializeCNNModel(defaultHP)
    for all hp ∈ hpList do
      for all v ∈ hpValues[hp]) do
        hpSelectedValues ← updateHPValues(v)
        M ← updateCNNModel(hpSelectedValues)
          t_start ← getCurrentTime()
          trainCNNModel(M, D_train)
          t_end ← getCurrentTime()
          Ttrain ← tend − tstart
          ACC, MSE← evaluateCNNModel(M, D_valid)
          w_t,w_a,w_m ← getRewardObjWeights()
          Reward ← w_t×T_train+w_a×ACC+w_m×MS E
          if (Reward > Reward_max) then
        Reward_max ← Reward
        optHPList ← updateHPValues(v)
        end if
        end for
    end for
    end while
    return optHPList
end procedure
```

The algorithm selects the policy that maximizes the cumulative reward as follows:

$\pi^*(s) \in argmax_a Q^*(s, a)$, where $Q^*(s, a)$ is the optimum state-action value. We adopt Q-learning [30] in our algorithm as the value-based (and off-policy) method to achieve the optimum policy by iteratively estimating the optimal value obeying the Bellman equation:

$$Q^*(s, a) = E_\pi\left[\sum_{t=0}^{\infty} r + \gamma \max(Q^*(s', a'))\,|\, a' = \pi(s')\right]$$

Algorithm 1 describes the details of the RL-based CNN optimization. The following are taken as the input: list of hyperparameters, list of value ranges for each hyperparameter, training dataset, validation dataset, and threshold for the accepted reward δ value. In each iteration of the algorithm, the agent selects one of the hyperparameters and loops over its available range of values and then selects one value to construct a new CNN model. The newly created model is evaluated using the testing and validation datasets and a new reward R value is determined. The new reward value r_t is compared to r_{t-1}, which is the reward calculated at time $t-1$. If the reward value converges toward the threshold value, the new CNN model configurations are stored to be used in the upcoming iterations; otherwise, the model is discarded and another hyperparameter value is selected in the next iteration. The algorithm keeps iterating until the reward R value exceeds the threshold δ; hence, the corresponding CNN model is accepted and adopted as the optimized model.

5. Experiments

This section describes the experimentations developed to assess various aspects of the proposed adaptive reinforcement model. The main objectives are to prove that the proposed model for the automatic optimization of hyperparameters achieves better performance (accuracy, time, and cross-entropy) compared to the baseline models and identify hyperparameters that might significantly affect the model's performance. Therefore, we developed four scenarios: the first scenario builds a baseline model based on manual hyperparameter optimization, whereas the second scenario implements our proposed automatic model optimization. These two scenarios aim to compare the performance of our automatic optimization to the baseline model. In the third scenario, we rank the hyperparameters in terms of importance and effectiveness in increasing the model's accuracy. Finally, our fourth scenario measures the overhead of our proposed model in terms of resource utilization. The following subsections detail our experiments, including the environment setup, the dataset employed, the experimental design, and scenarios and finally discuss the obtained results.

5.1. Environment setup

This section describes the environment setup employed in our experiments. We implemented and tested the CNN model over a system with the following configurations:

- Processor: Intel(R) Core (TM) i7-10750H CPU @ 2.60 GHz 2.59 GHz
- RAM: 16.0 GB
- System type: 64-bit operating system, x64-based processor
- GPU: NVIDIA GeForce RTX 2070 (8 GB RAM)

5.2. Dataset preprocessing, analysis, and processing setup

We used the MIT-BIH arrhythmia database [31]. This database includes 48 two-channel ambulatory ECG recordings from 47 subjects at the BIH Arrhythmia Laboratory between 1975 and 1979. Each of these ECG recordings is about a 30-min long. The sampling rate for digitizing the recordings was 360 samples/s/channel. Each record was annotated by two or more cardiologists, and the disagreements were resolved. Finally, computer-readable reference annotations were obtained for each beat [32]. WFDB Python Package was used to read the MIT-BIH arrhythmia database. The package includes a library of tools for reading, writing, and processing WFDB signals and annotations.

The study focused on nine classes of Arrhythmia, which are:

1. Right bundle branch block beats
2. Left bundle branch block beats
3. Premature ventricular contractions
4. Atrial premature beats
5. Fusions of ventricular
6. Normal beats
7. Paced beats
8. Unknown/unclassifiable beats
9. Supraventricular premature beats

1) Data preprocessing (Denoising: ECG preprocessing Module)

To minimize the noise in ECG recordings, such as powerline noise interference and baseline wandering, BioSPPy toolbox was used. BioSPPy is a toolbox for biosignal processing that is written in Python to provide an easy starting point into biosignal processing. One of the main methods of BioSppy is filtering biosignals. BioSPPy is offering a suite of algorithms tailored for analyzing various physiological signals such as electrocardiography (ECG), electrodermal activity (EDA), and electromyography (EMG), among others. It simplifies the extraction of pertinent information from biosignals, empowering researchers and developers to focus on their analytical tasks. BioSPPy encompasses multiple modules and functionalities, including signal processing and feature extraction such as heart rate variability (HRV), EDA analysis, and EMG analysis. As an open-source library, it can be effortlessly installed using Python package managers like pip or conda. Its modular architecture and intuitive API cater to both novices and seasoned researchers in the field of biosignal processing [33].

ECG signals might exhibit powerline noise interference and might have a DC offset resulting from the acquisition device, and signals can also have the influence of breathing in the variability of R-peak amplitudes. To minimize the effects of these artifacts and extract a bunch of features, module “signals.ecg” in BioSPPy encompasses functions for electrocardiogram signal importing, and signal pre-processing. For the pre-processing, the module filters interfering components from the signal, such as muscle activity, 50 Hz power noise and baseline wandering [34].

Figs. 4, and 5 show a sample raw, and filtered ECG signal, respectively.

2) Data analysis

The category-wise data distribution shown in Fig. 6 depicts that the F, Q, and S categories comprise only few records, and hence, we did not include these classes in the output. Subsequently, we performed data resampling using the upscaling technique to adjust the sample distribution. Fig. 7 depicts the rebalanced dataset used for training, considering 6 classes out of 9.

3) ECG processing Module

R-peak detection

To detect R-peaks, we relied on the annotation files provided by the MIT-BIH arrhythmia database. In these annotation files, the location of beats annotations generally appears with sufficient accuracy at the R-wave peak.

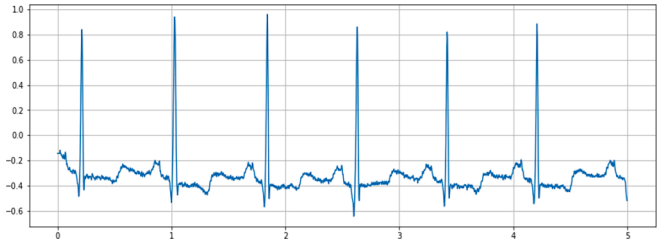


Fig. 4. Sample of an unfiltered ECG signal.

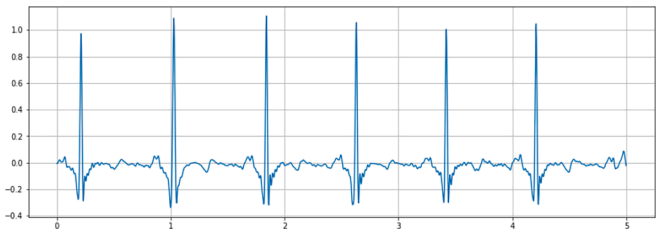


Fig. 5. Sample of a filtered ECG signal using BioSPPy toolbox.

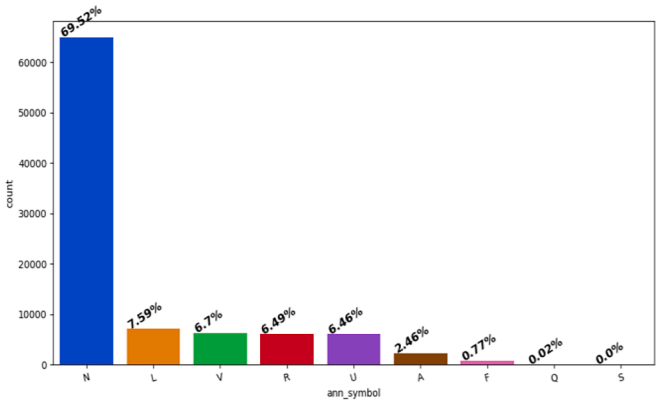


Fig. 6. Category-wise data distribution. N: Normal beats. L: Left bundle branch block beats. V: Premature ventricular contractions. R: Right bundle branch block beats. U: Unknown/unclassifiable beats. A: Atrial premature beats. F: Fusions of ventricular. Q: Paced beats. S: Supraventricular premature beats.

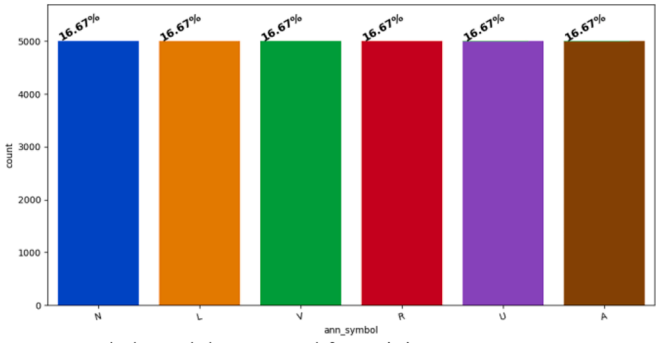


Fig. 7. Rebalanced dataset used for training.

Heartbeat Segmentation

Since each heartbeat in the MIT-BIH arrhythmia database is annotated at its R peak, Swapna et al. extracted heartbeats based on the R peaks, where they defined a window of size 1 s to select a series of samples centered on the R peak of each heartbeat. At the end of their segmentation process, they obtained 361 samples for each heartbeat [9]. Additionally, Yildirim et al. segmented ECG based on the R peak, where they selected 99 samples before the R peak, and 160 samples after the R peak. Each beat included 260 samples [13]. We followed the same method used by Swapna et al. and Yildirim et al. [9 13] to segment ECG signals based on the R peak, and we included 260 samples per beat.

Temporal feature extraction

According to Sannino et al. [4], ECG segmentation can cause loss of information about the ECG waveform. Computing each beat’s temporal features will help retain useful information about the beats for

classification. We computed the following temporal features for each beat:

- Pre-RR interval: defined as the RR-interval between a given heartbeat and the previous heartbeat
- Post-RR interval: defined as the RR-interval between a given heartbeat and the following heartbeat
- Local average RR interval: defined as the average of 10 RR-intervals within a sliding window covering the past 10 s. Here, we assumed one beat per second.
- Global average RR interval: defined as the average of 10 RR-intervals within a sliding window covering the previous 5 min. Here, we assumed one beat per second.

Fig. 8 illustrates the plots for six different ECG classes that were generated based on our ECG processed data. Each of these plots represents a single random beat for a specific ECG class, with each beat comprising 260 samples (as described in the Heartbeat Segmentation section). The ECG classes represented in the figure are based on the data from the annotation files provided by the MIT-BIH arrhythmia database.

5.3. Experimental design: evaluation metrics, scenario's description, and implementation details

1) Model Evaluation Metrics

For our model evaluation, we adopted the cross entropy evaluation metric [36] which is described in the model optimization section. Moreover, we applied different evaluation criteria, such as accuracy, for each class to measure the performance of our experimented classification models [37]. Accordingly, we determine how well our classification models perform. The following formulas detail the evaluation metrics:

$$Accuracy = \frac{truepositives + truenegatives}{truepositives + truenegatives + falsepositives + falsenegatives} = \frac{correctlyclassifiedsignalsasclass_i}{totalactualclass_isignals}$$

Additionally, we adopted additional measurements to further validate our model, including the time to train the model, resource utilization in terms of memory consumption, and CPU utilization. For the time to train the model, we measure the actual time taken to train the data by each model, and we use this measure to compare the models.

2) Scenario's Description

The following four main scenarios are used to evaluate the proposed model:

Scenario — 1: Prediction results without model optimization (manual model optimization) baseline model. In this scenario, we use the ECG dataset and build the CNN model with manual tuning for training and setting network hyperparameters. We use two configuration schemes for the baseline model: the first configuration uses basic hyperparameters and the second configuration uses simple manual tuning of a few hyperparameters.

Scenario — 2: Prediction results with automatic optimization model. In this scenario, we use the same dataset employed in the first scenario and apply our RL-based model to tune both the training and network hyperparameters. We evaluate the model's performance in terms of accuracy, training time, and cross-entropy and then compare the results to those obtained in scenario 1.

Scenario — 3: Rank the hyperparameters in terms of importance and effectiveness. In this scenario, we consider various hyperparameters tunings, and we evaluate the baseline model to individually assess the effect of each parameter and rank the most influential parameters based on the model accuracy. We use different values for the following hyperparameters: learning rate, number of layers, dropout, batch size, number of epochs, and activation function.

Scenario – 4: Model overhead evaluation in terms of processing cost and resource consumption. In this scenario, we assess the cost of the proposed automatic hyperparameters fine-tuning in terms of CPU and memory utilization compared to the benchmarking scenario 1. We report and compare the results to verify whether the processing cost and consumed resources of the proposed model are insignificant compared to those of the baseline model.

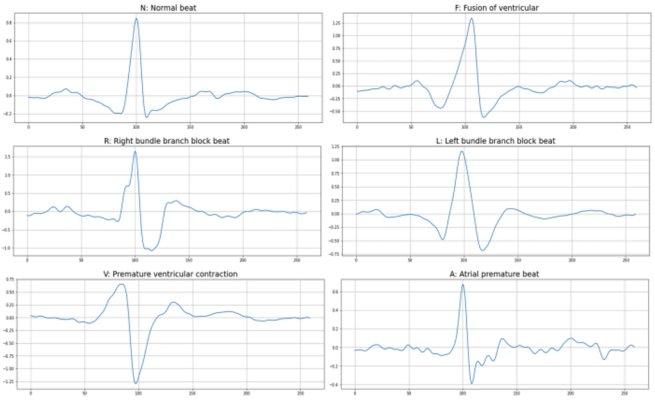


Fig. 8. Sample ECG generated classes.

3) RL Implementation Details

This section describes the RL implementation with respect to the hyperparameter classification using MDP, the actions, and Q-Table initialization as well as the resulting reward table.

MDP for Hyperparameter classification: The Q-Learning algorithm utilizes a Q Table; the latter is updated based on the Bellman equation, as detailed in section 8. The values in the Q Table are used to determine the best action, while Q defines how helpful an action that an agent takes in a state is, and it is computed using the state action-value function.

Actions and Q Table Initialization: Actions represent the change of hyperparameter values from the selected list based on the policy to calculate the reward. The rows and columns count is equal to the current parameter options count, and the Q-Table values are initialized to zeros.

Reward Table: The reward table is used to record the calculated reward corresponding to each action taken by the agent. Additionally, the values in this table are initialized to zeros. If the action reward is more than the threshold value, the table is updated with the actual reward; otherwise, it is updated with − 1.

Fig. 9 displays a sample Q-Table, where the highest value in the "Action" column indicates the best action, the agent can take to obtain the highest reward. For each state depicted in this table, the highest Q-value is in the 4th column (learning rate = 0.005). Hence, we conclude that irrespective of the current state, the action that affords the maximum reward is the change of the learning rate to 0.005. Accordingly, all the hyperparameter values leading to the best performance are derived using the Q-Table using the same procedure.

6. Results and discussion

Following the above experimental design, herein, we conduct a set of experimentations using the above-described scenarios, and we report and discuss the obtained results.

1) Adaptive deep RL module evaluation (Scenario 1, 2)

We detail here the implementation scenarios used to evaluate our proposed RL model optimization in terms of accuracy, cross-entropy,

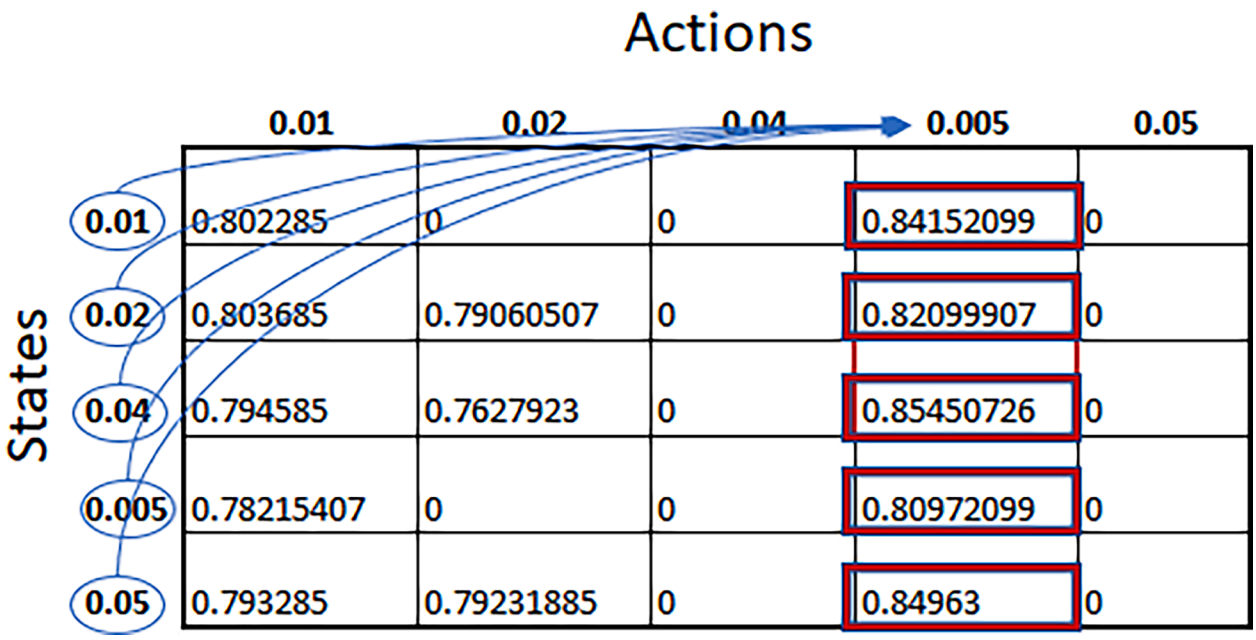


Fig. 9. Q-Table for learning rate.

and training time measured against two other benchmarking models. This experiment is represented by scenarios 1 (A, B) and 2, which are detailed in section 9.3. We first draw the general CNN model architecture as depicted in Fig. 10, and then, we detail the CNN model configuration for each scenario. Finally, we report the obtained results.

Scenario 1A- Prediction results with baseline model: Fig. 11 (a) shows the CNN model architecture used in scenario 1A. We evaluate the prediction results without model optimization. In this scenario, the time series ECG data of the 1-D array is passed to the input layer. The output for the model is a six-element vector containing the probability of a given sample belonging to each of the six arrhythmia classes. The model is defined as having three 1-D CNN layers followed by a pooling layer for dimensionality reduction and a dropout layer for regularization. After the CNN and dropout, the learned features are flattened to one long vector and passed through two fully connected layers. Then, the output layer is used to make a prediction. For this model, we employ a standard configuration of parallel feature maps (filters) and a kernel size for the CNN layers as follows: CNN layer 1 => 32, 3, CNN layer 1 => 64, 3, and CNN layer 1 => 128, 5, where each number represents the filters and kernel size, respectively.

Scenario 1B — Prediction results with manual HP tuning: Fig. 11 (b) shows the CNN model architecture used in scenario 1B. In this model, we reduce one CNN layer from the previous model and the current configurations are as follows: CNN layer 1 =>32, 3 and CNN layer 1 => 32, 3, where each number represents the filters and kernel size, respectively.

Scenario 2 — Prediction results with automatic optimization model: The CNN layers selected in the manual tuning were found to be the best among the selected configurations in automated tuning. The model specifications for each scenario are shown in Table 3.

Analysis of the confusion matrix for scenarios 1A and 1B in Fig. 12 (a), and 12(b) shows that the most incorrect predictions are afforded for the class "N," where the model is predicted as "V." Additionally, incorrect classification is afforded for the class "N" that is predicted as

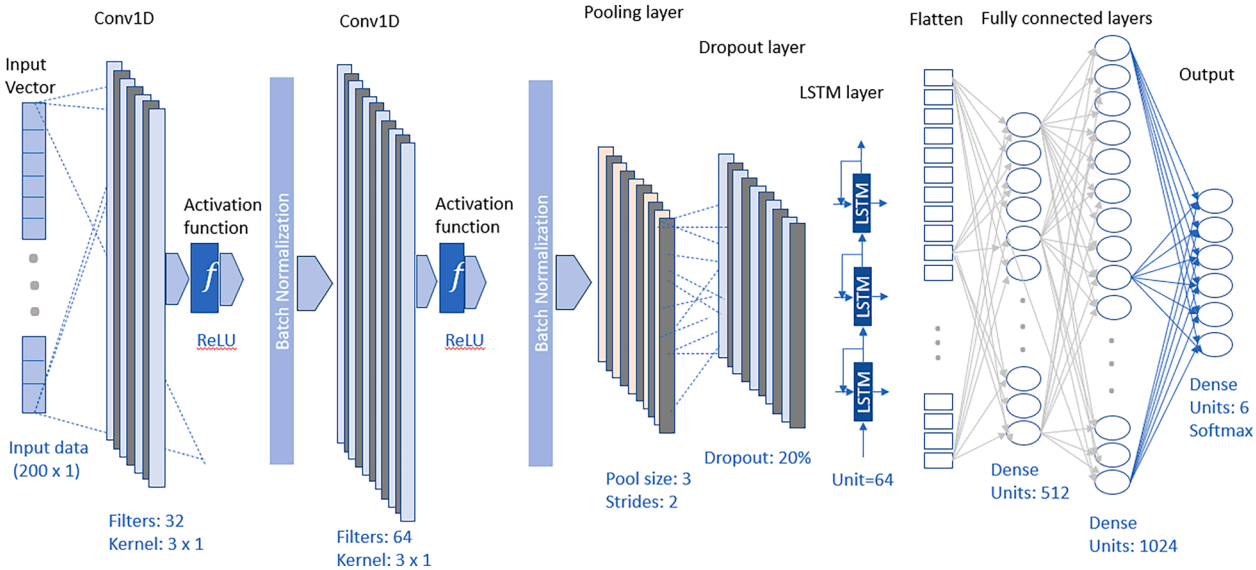
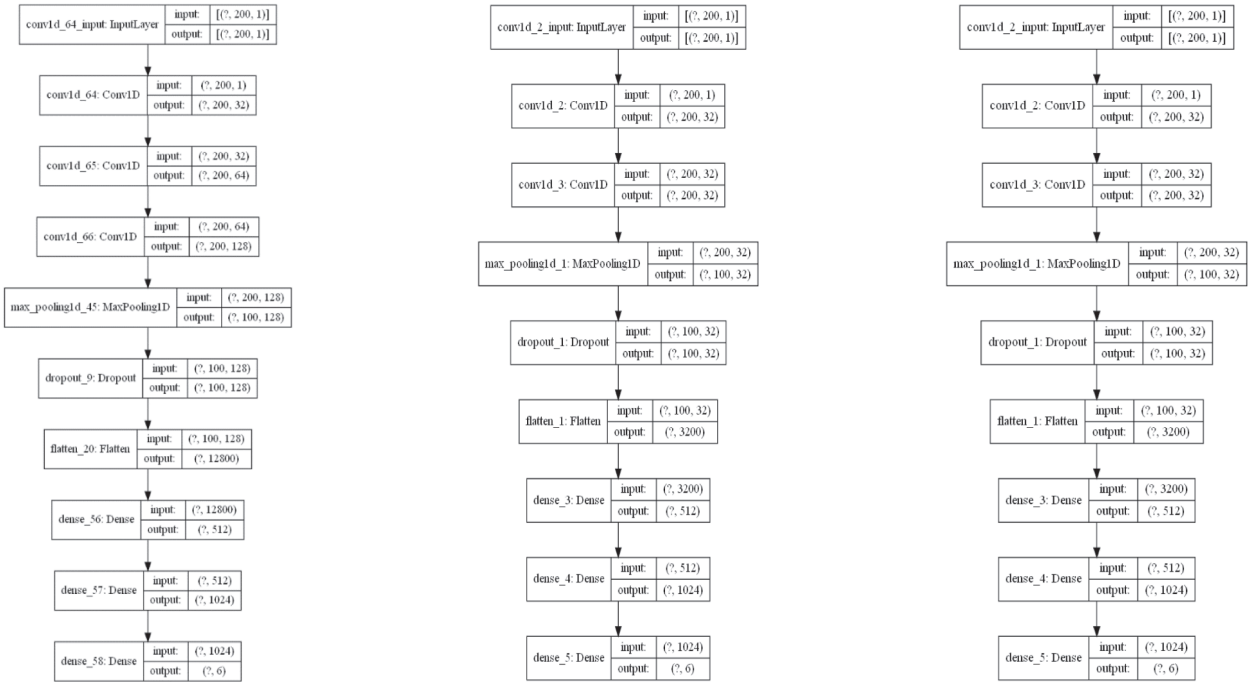


Fig. 10. General CNN model architecture.



a) Baseline CNN model

b) CNN model - manual tuning

c) CNN model – automated tuning

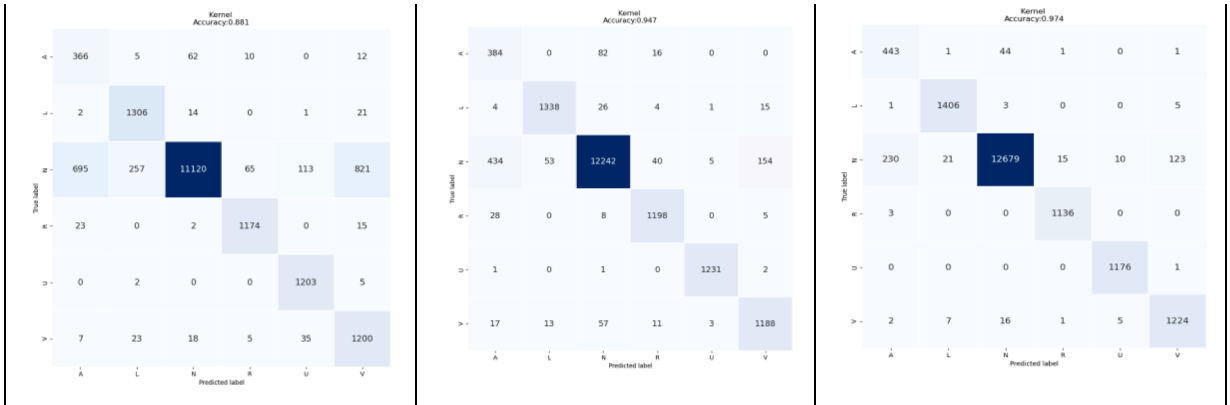
Fig. 11. CNN model architecture details for each scenario.

Table 3
Model specification for each scenario.

	Scenario 1A	Scenario 1B	Scenario 2
Number of layers	9 (6 trainable)	8 (5 trainable)	8 (5 trainable)
Learning rate	0.01	0.01	0.002
Batch size	10	15	64
Dropout	0.5	0.5	0.2
Number of epochs	20	20	15
Activation function	ReLu	ReLu	SoftPlus

"A," followed by the wrong predictions as "L" and "U." We aim to minimize this error by tuning the hyperparameters used in the CNN model. However, the confusion matrix from the automated tuning CNN model shown in Fig. 12(c) depicts an improved outcome.

Table 4 depicts a class-wise comparison of accuracy gain for the six arrhythmia classes based on the above confusion matrices. For Class A, the manual tuning of the hyperparameters does not improve the prediction accuracy, while the automatic tuned model resulted in ~ 10 % improvement in accuracy. The majority of the wrong prediction is afforded for class A, predicted as "N." All the models predict class L almost correctly with an accuracy exceeding 96 %, but the automatic tuned model outperformed the other two models with an accuracy above 99 %. Most of the samples belong to class N, where the prediction using



a) Baseline CNN model confusion matrix

b) CNN model-manual tuning confusion matrix

c) CNN model after automated tuning confusion matrix

Fig. 12. CNN model confusion matrix for all scenarios.

Table 4
Class-wise comparison of accuracy gain with hyperparameter tuning.

Atrial premature beat	CNN model	Sample size	True prediction	False prediction	% accuracy
	Baseline	455	366	89	80.44
	Manual	482	384	98	79.67
	Automated	489	443	46	**90.59**
Left bundle branch block beat	Baseline	1344	2306	38	97.17
	Manual	1388	1338	50	96.4
	Automated	1415	1406	9	**99.36**
Normal beat	Baseline	13,071	11,120	1951	85.07
	Manual	12,928	12,242	686	94.69
	Automated	13,078	12,679	399	**96.95**
Right bundle branch block beat	Baseline	1214	1174	40	96.71
	Manual	1239	1198	41	96.69
	Automated	1139	1136	3	**99.74**
Unknown/unclassifiable beats	Baseline	1210	1203	7	99.42
	Manual	1235	1231	4	99.68
	Automated	1177	1176	1	**99.92**
Premature ventricular contraction	Baseline	1288	1200	88	93.17
	Manual	1289	1188	101	92.16
	Automated	1255	1224	31	**97.53**

the baseline model exhibits poor performance, with an accuracy of ~ 85 %. Manual tuning resulted in better accuracy of above 94 %, but the automated model gave an accuracy of ~ 97 %. For class R, the baseline model and manual tuning model observed similar accuracies of ~ 97 %, but the automatically tuned model again affords better accuracy of approximately 100 %. All models afford good prediction results for class U, with accuracy above 99 %.

For class V, the manual tuning of the hyperparameters is not effective, whereas automatically tuned model gained an accuracy of above 97.5 %. Overall, compared to the other two models, the automatically tuned model well tunes the hyperparameters with good prediction accuracy for all the classes.

Figs. 16–19 summarize and compare the results obtained for each of the above-described scenarios. Fig. 16 shows that the optimized model accuracy exceeds that of the baseline and manually optimized models. Fig. 17 depicts the sample accuracy value for each scenario; clearly, the accuracy of the optimized model (scenario 2) is higher (97.4) than that of the other two models.

Figs. 18 and 19 report the model execution time and cross-entropy observed for each scenario. Similarly, the adaptive optimized model exhibits the lowest execution time of 0.33 min compared to 3.66 min for the baseline model. Moreover, the adaptive optimized model shows the least error of 0.0081 compared to 0.0336 for the baseline model.

Effective, whereas automatically tuned model gained an accuracy of above 97.5 %. Overall, compared to the other two models, the automatically tuned model well tunes the hyperparameters with good prediction accuracy for all the classes.

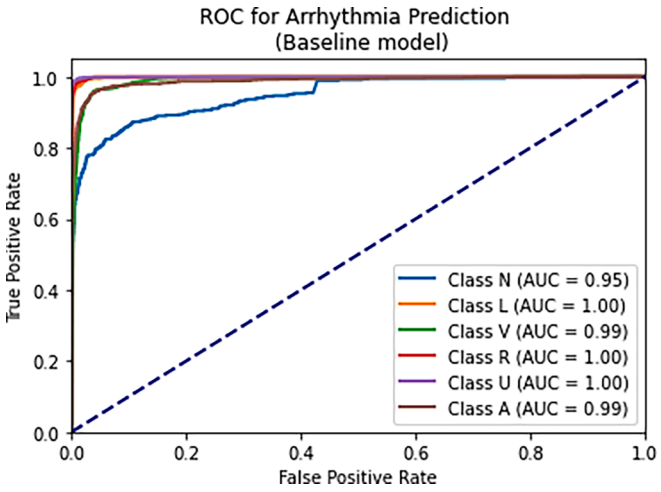


Fig. 13. ROC for arrhythmia Prediction (Baseline model).

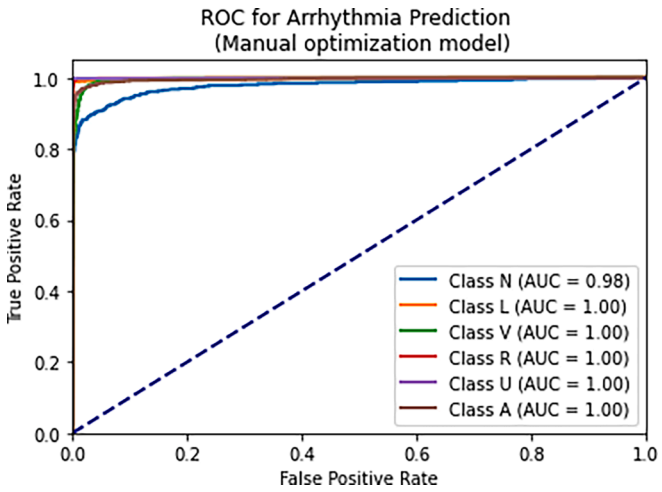


Fig. 14. ROC for arrhythmia Prediction (Manual optimization model).

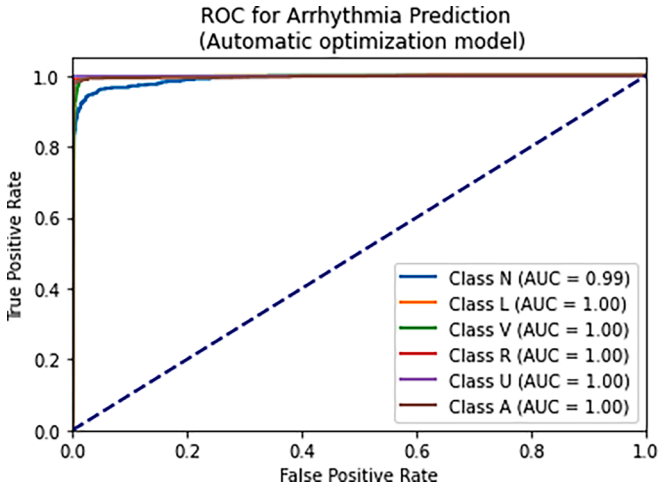


Fig. 15. ROC for Arrhythmia Prediction (Automatic optimization model).

We draw the area under the curve for baseline, manual and automatic optimization of arrythmia prediction as depicted in Figs. 13, 14, and 15. From the three figures automatic optimization model show the highest accuracy of the test, the sensitivity and specificity are 1 for five

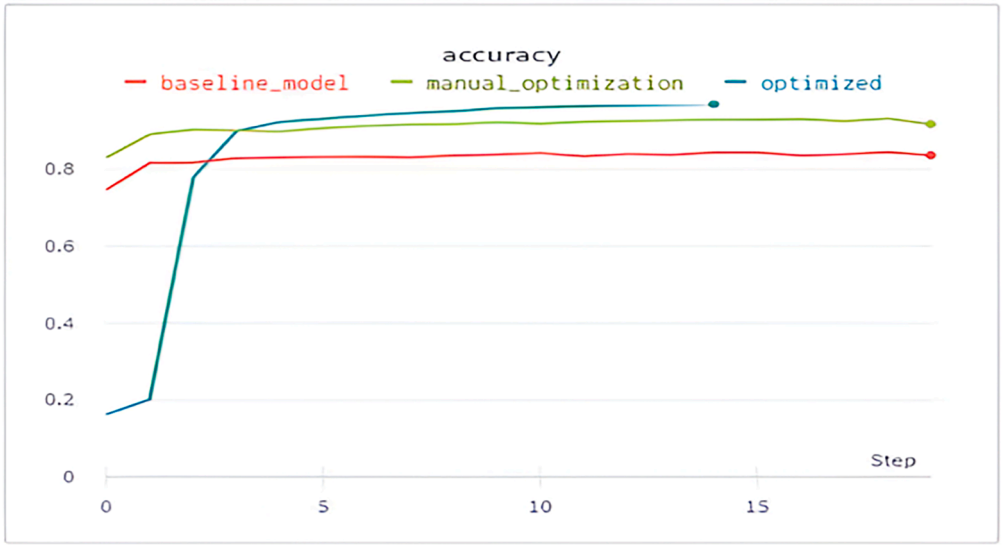


Fig. 16. Accuracy comparison between different scenarios.

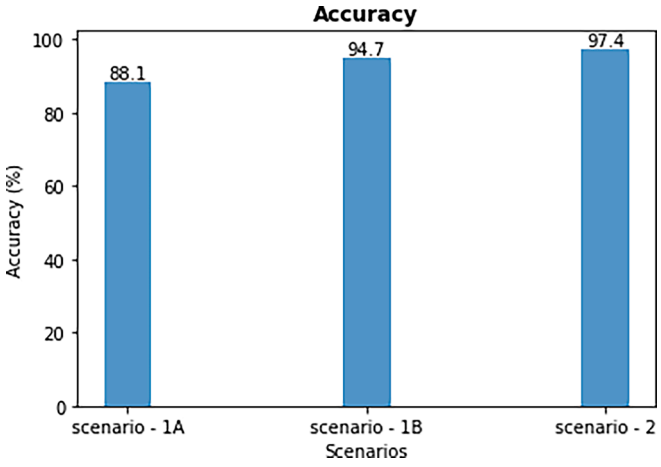


Fig. 17. Sample accuracy value is taken for each scenario.

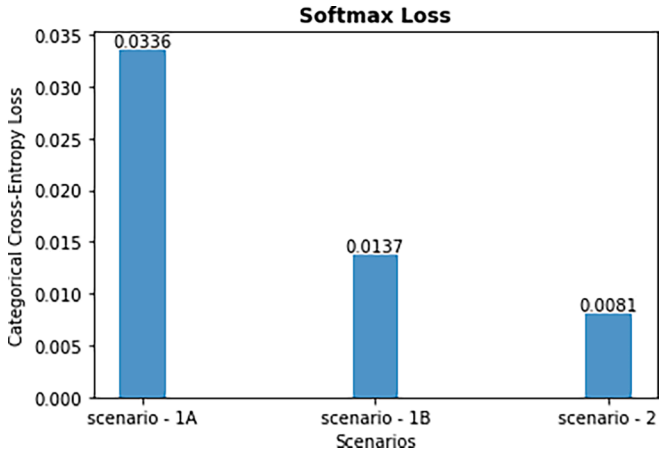


Fig. 19. A typical cross-entropy loss observed from each scenario.

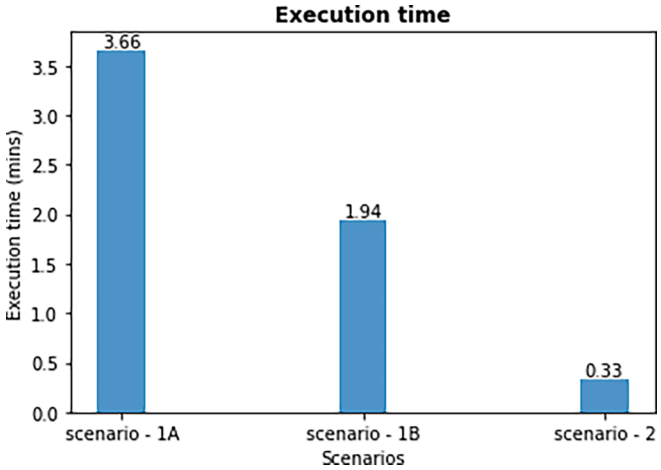


Fig. 18. Sample execution time monitored for each scenario.

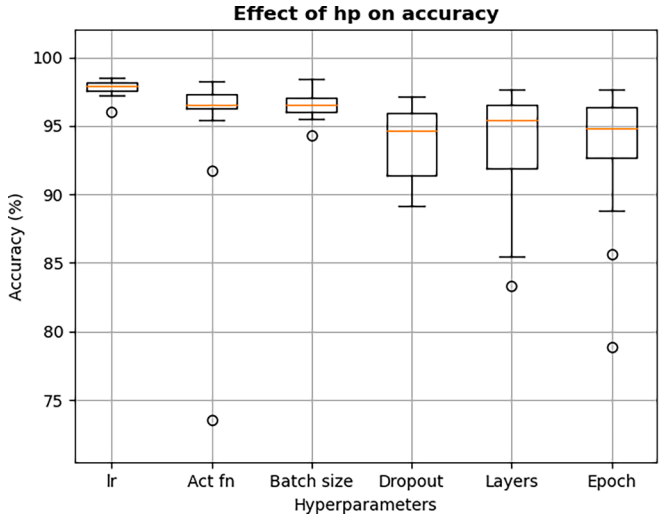


Fig. 20. Accuracy dependency plot for different hyperparameters.

arrhythmia classes.

For class V, the manual tuning of the hyperparameters is not.

2) Effect of hyperparameter on model performance (Scenario 3)

The boxplot in Fig. 20 depicts the CNN model accuracy variations monitored for each of the independently tested hyperparameters. All the values have been taken from hyperparameter tuning using the RL method. Experiments indicate the hyperparameters impact the model accuracy in the following order (severe to mild): 1. Learning rate, 2. Batch size, 3. Activation function, 4. Number of layers, 5. Number of epochs, 6. Dropout.

The boxplot in Fig. 21 depicts the CNN model accuracy variation monitored for a model with 5,6,7, and 8 layers. Increased number of layers and epochs cause model overfitting, which affords the best training accuracy but poor validation accuracy.

1) Resource consumption evaluation (Scenario 4)

Here, we assess the cost of the proposed automatic hyperparameter fine-tuning in terms of GPU, memory, and disk utilization for each model configuration. The results depicted in Fig. 22 show that our optimized CNN model exhibits the least GPU utilization in all the three scenarios, with a large batch size (64) configuration. Additionally, the GPU memory allocation exhibits consistency in all three scenarios by utilizing 100 % of the required memory for the model training as Keras (TensorFlow) allocates all GPU memory by default. Moreover, the training of all three models is well within the ideal GPU temperatures between 65 °C and 85 °C under load. The figure also shows insignificant disk space utilization as a typical CNN model does not use considerable disk space.

Fig. 23 shows the amount of available memory and the memory in use (non-swap), denoting the RAM space utilization and availability during the training of the CNN model. Memory utilization increases with the batch size. The system memory utilization shows little system memory usage for the CNN training because the data are directly hosted on the GPU memory. The system memory usage depicted in the figure includes the memory used for the training process, the initially allocated memory space. Finally, the figure shows little workload on the CPU during the training.

2) Algorithm time complexity

This section evaluates the time complexity of our proposed multi-objective RL algorithm using the big O notation. This provides an indication about the algorithmic efficiency, which consequently impacts the execution time. The performance of Algorithm 1, depicted in section 4 (D), depends on the number of trainings hyperparameters T and network structure hyperparameters N as well as the maximum interval size n_{max} of the selected value ranges of each hyperparameter.

Let each hyperparameter have a set of values $V_i \in [a_i..b_i]$, which is a closed interval representing the set of all integer numbers greater or equal to a_i and less than b_i, where $i \in [1..(\mathrm{T} + \mathrm{N})]$ is the hyperparameter number. Let the number of values between a_i and b_i be n_i, then $n_{max} = n_{max} = \mathrm{Max}_{i \in [1..(\mathrm{T}+\mathrm{N})]} n_i$..

Moreover, the algorithm time complexity is subject to the number of iterations ω that are to be executed until the reward value converges to the acceptable threshold, which adds to the time complexity. Therefore, the time complexity of the CNN optimization algorithm is of the order of:

$$O\left((\mathrm{T} + \mathrm{N}) * n_{max} * \omega\right)$$

Because the number of hyperparameters and interval size for a given problem can be considered as constant values, the overall time complexity should only consider the time complexity of all the CNN models evaluated during each stage of the optimization before converging to the accepted Reward R value. Usually, the time complexity for the CNN models depends on the number of layers, the number of hidden neurons, and the input size.

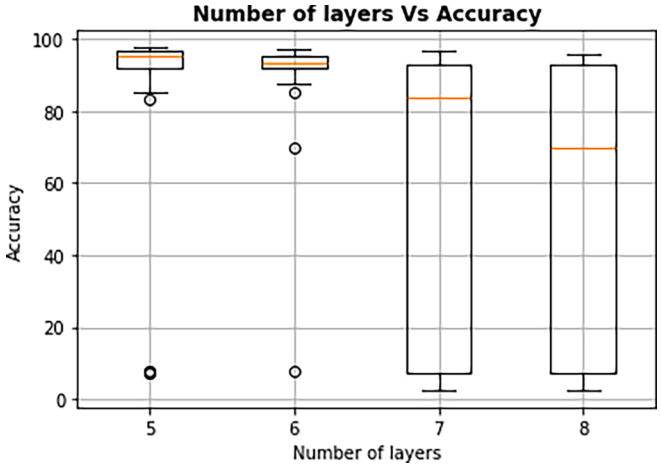


Fig. 21. Accuracy dependency plot for different number of layers.

7. Conclusion

Monitoring-based ECG analysis has been extensively used for diagnosing various cardiovascular diseases and preventing implications associated with these diseases, such as strokes and hard attacks. Deep learning models demonstrated their abilities to accurately process, classify, and analyze huge and time series datasets and generate relevant insights and recommendations. However, to realize the full potential of these deep learning models, their architectures need to be properly designed and their hyperparameters parameters need to be carefully selected and tuned. Therefore, this study proposed an architecture that automatically fine-tunes hyperparameters of the CNN models and network architecture based on RL. The proposed model automatically fine-tunes hyperparameters and performs iterations until the optimum configuration is reached that maximizes the reward function, subsequently achieving high accuracy. The proposed model tackles some of the inherent issues related to automatic fine-tuning of CNN models, such as resources consumption and complexity.

We evaluated our proposed model using a set of scenarios that considered the following: 1) baseline with no model optimization, 2) manual model fine-tuning, and 3) automatic model fine-tuning. We compared the results of these experiments using three evaluation performance metrics: accuracy, cross-entropy, and execution time. The obtained results prove that automatic fine-tuning affords the highest accuracy (97.4 %), least execution time (0.33 min), and lowest cross-entropy (0.0081). Additionally, we assessed the impact of each hyperparameter on the model accuracy, and the results show that the learning rate has the highest influence and dropout has the least influence on the model accuracy. Furthermore, we evaluated and compared the resource consumption of the three models. The results prove that our proposed solution is lightweight, is less complex, consumes the least resources, and the overhead is low. Finally, we evaluated our proposed multi-objective RL algorithm overhead in terms of time complexity and determined that it does not significantly affect the overall time complexity of the optimized CNN Model. In future studies, we plan to apply our RL-based model to other deep learning models, such as LSTM and Recurrent Neural Network RNN, and compare the results.

CRediT authorship contribution statement

Mohamed Adel Serhani: Writing – original draft, Supervision,

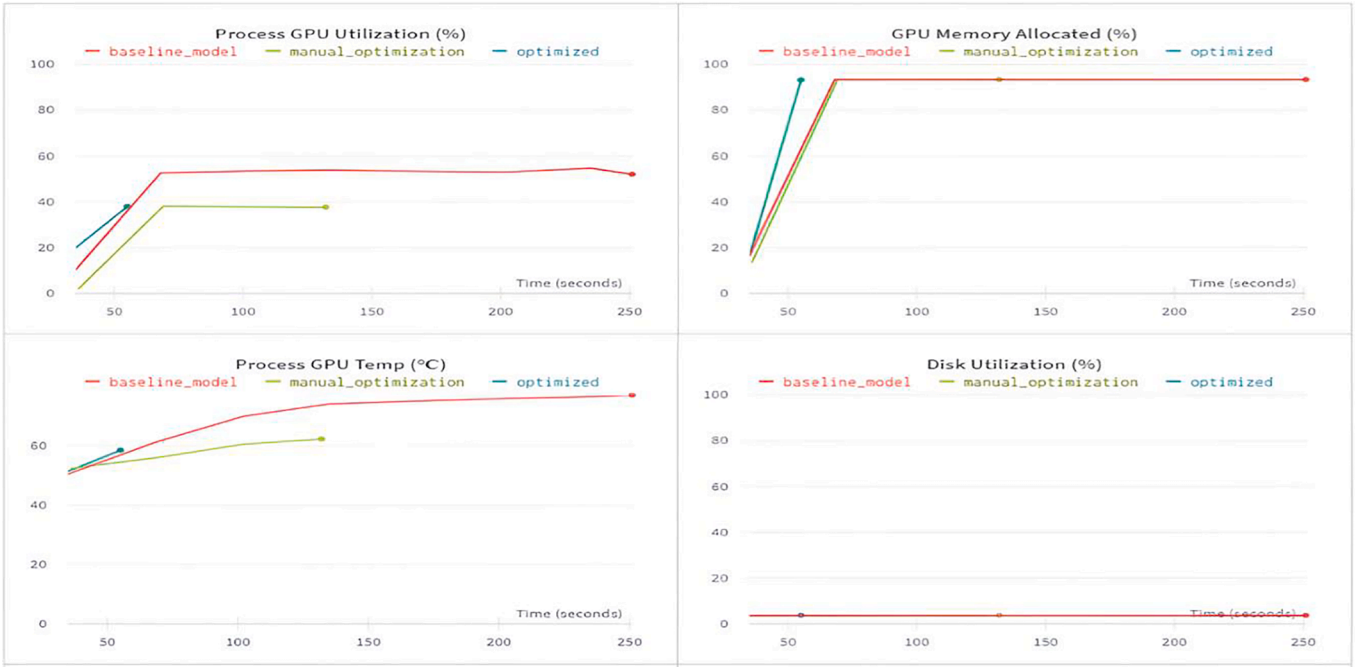


Fig. 22. Resource consumption (GPU, Memory, Disk) for each scenario.

Fig. 23. Resource consumption (Memory available, Memory in use, and CPU utilization) for each scenario.

Resources, Project administration, Methodology, Investigation, Conceptualization. **Heba Ismail:** Writing – review & editing, Visualization, Software, Investigation, Data curation, Conceptualization. **Hadeel T. El-Kassabi:** Writing – review & editing, Methodology, Investigation, Formal analysis. **Hamda Al Breiki:** Writing – review & editing, Visualization, Investigation, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We would like to thank Mr. Madhu Chaliyan for his contribution to the project implementation and experimentation.

Data availability

The authors do not have permission to share data.

References

[1] "Arrhythmia | NHLBI, NIH." .

[2] P.E. Dilaveris, H.L. Kennedy, Silent atrial fibrillation: epidemiology, diagnosis, and clinical impact, Clin. Cardiol. 40 (6) (Jun. 2017) 413, https://doi.org/10.1002/CLC.22667.
[3] A. Barbarossa, F. Guerra, A. Capucci, Silent atrial fibrillation: a critical review, J. Atr. Fibrillation 7 (3) (Oct. 2014) 39–44, https://doi.org/10.4022/JAFIB.1138.
[4] G. Sannino, G. De Pietro, A deep learning approach for ECG-based heartbeat classification for arrhythmia detection, Futur. Gener. Comput. Syst. 86 (2018) 446–455, https://doi.org/10.1016/j.future.2018.03.057.
[5] U.R. Acharya, et al., A deep convolutional neural network model to classify heartbeats, Comput. Biol. Med. 89 (2017) 389–396.
[6] S.M. Mathews, C. Kambhamettu, K.E. Barner, A novel application of deep learning for single-lead ECG classification, Comput. Biol. Med. 99 (2018) 53–62.
[7] A. Isin, S. Ozdalili, Cardiac arrhythmia detection using deep learning, Proc. Comput. Sci. 120 (2017) 268–275, https://doi.org/10.1016/j.procs.2017.11.238.
[8] J. Pan, W.J. Tompkins, A real-time QRS detection algorithm, IEEE Trans. Biomed. Eng. 3 (1985) 230–236.
[9] G. Swapna, K.P. Soman, R. Vinayakumar, Automated detection of cardiac arrhythmia using deep learning techniques, Proc. Comput. Sci. 132 (2018) 1192–1201.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Adv. Neural Inf. Process. Syst., vol. 25, 2012.
[11] Ö. Yildirim, A novel wavelet sequences based on deep bidirectional LSTM network model for ECG signal classification, Comput. Biol. Med. 96 (2018) 189–202, https://doi.org/10.1016/j.compbiomed.2018.03.016.
[12] Ö. Yıldırım, P. Pławiak, R.S. Tan, U.R. Acharya, Arrhythmia detection using deep convolutional neural network with long duration ECG signals, Comput. Biol. Med. 102 (August) (2018) 411–420, https://doi.org/10.1016/j.compbiomed.2018.09.009.
[13] O. Yildirim, U.B. Baloglu, R.S. Tan, E.J. Ciaccio, U.R. Acharya, A new approach for arrhythmia classification using deep coded features and LSTM networks, Comput. Methods Programs Biomed. 176 (2019) 121–133, https://doi.org/10.1016/j.cmpb.2019.05.004.
[14] E.H. Houssein, M. Hassaballah, I.E. Ibrahim, D.S. AbdElminaam, Y.M. Wazery, An automatic arrhythmia classification model based on improved Marine Predators Algorithm and Convolutions Neural Networks, Expert Syst. Appl. 187 (Jan. 2022) 115936, https://doi.org/10.1016/j.eswa.2021.115936.
[15] Z. Li, H. Zhang, Automatic detection for multi-labeled cardiac arrhythmia based on frame blocking preprocessing and residual networks, Front. Cardiovasc. Med. 8 (Mar. 2021) 135, https://doi.org/10.3389/fcvm.2021.616585.
[16] S. Aphale, A. Jha, E. John, "High accuracy arrhythmia classification using transfer learning with fine-tuning," in *2022 IEEE 13th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2022*, 2022, pp. 480–487, doi: 10.1109/UEMCON54665.2022.9965693.
[17] A. Ullah, S.M. Anwar, M. Bilal, R.M. Mehmood, Classification of arrhythmia by using deep learning with 2-D ECG spectral image representation, Remote Sens. 12 (10) (2020) 1685.
[18] S. Mousavi, F. Afghah, Inter-and intra-patient ecg heartbeat classification for arrhythmia detection: a sequence to sequence deep learning approach, in: In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 1308–1312.
[19] P. Laguna, R. Jané, P. Caminal, Automatic detection of wave boundaries in multilead ECG signals: Validation with the CSE database, Comput. Biomed. Res. 27 (1) (Feb. 1994) 45–60, https://doi.org/10.1006/cbmr.1994.1006.
[20] S.L. Oh, E.Y.K. Ng, R.S. Tan, U.R. Acharya, Automated diagnosis of arrhythmia using combination of CNN and LSTM techniques with variable length heart beats, Comput. Biol. Med. 102 (June) (2018) 278–287, https://doi.org/10.1016/j.compbiomed.2018.06.002.
[21] X. Zhai, C. Tin, Automated ECG classification using dual heartbeat coupling based on convolutional neural network, IEEE Access 6 (2018) 27465–27472.
[22] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," 2012.
[24] X. Li, F. Zhang, Z. Sun, D. Li, X. Kong, Y. Zhang, Automatic heartbeat classification using S-shaped reconstruction and a squeeze-and-excitation residual network, Comput. Biol. Med. 140 (Jan. 2022) 105108, https://doi.org/10.1016/J.COMPBIOMED.2021.105108.
[25] S.K. Pandey, R.R. Janghel, Automated detection of arrhythmia from electrocardiogram signal based on new convolutional encoded features with bidirectional long short-term memory network classifier, Phys. Eng. Sci. Med. 44 (1) (Mar. 2021) 173–182, https://doi.org/10.1007/S13246-020-00965-1.
[26] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: theory and practice, Neurocomputing 415 (2020) 295–316.
[27] N. Mboga, C. Persello, J. R. Bergado, and A. Stein, "Detection of informal settlements from VHR images using convolutional neural networks," *Remote Sens.*, vol. 9, no. 11, 2017, doi: 10.3390/rs9111106.
[28] Y. Sun, B. Xue, M. Zhang, G.G. Yen, J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification, IEEE Trans. Cybern. 50 (9) (2020) 3840–3854, https://doi.org/10.1109/TCYB.2020.2983860.
[29] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. G. Bellemare, R. Munos, "The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning," *arXiv*, pp. 1–18, 2017.
[30] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. (1992) 279–292.
[31] G.B. Moody, R.G. Mark, The impact of the MIT-BIH arrhythmia database, IEEE Eng. Med. Biol. Mag. 20 (3) (2001) 45–50, https://doi.org/10.1109/51.932724.
[32] A.L. Goldberger, et al., PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals, Circulation (2000), https://doi.org/10.1161/01.cir.101.23.e215.
[33] H. Pishro-Nik, Introduction to probability, statistics, and random processes. Kappa Research LLC, 2016.
[34] D. M. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," *arXiv Prepr. arXiv2010.16061*, 2020.