Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator

Anthony J. Dunn[1], Stefano Coniglio[1*], Mohamed ElRefai[2], Paul R. Roberts[2], Benedict W. Wiles[3] and Alain B. Zemkoho[1]

[1]School of Mathematical Sciences, University of Southampton, University Road, Southampton, SO17 1BJ, UK.
[2]Department, University Hospital Southampton, Tremona Road, Southampton, SO16 6YD, UK.
[3]Department, St George's University Hospitals NHS Foundation Trust, Blackshaw Road, London, 610101, UK.

*Corresponding author(s). E-mail(s): s.coniglio@soton.ac.uk;
Contributing authors: ajd1g15@soton.ac.uk;
mohammed.elrefai@uhs.nhs.uk; paul.roberts@uhs.nhs.uk;
ben.wiles@doctors.org.uk; a.b.zemkoho@soton.ac.uk;

Abstract

In cardiology, it is standard for patients suffering from ventricular arrhythmias (the leading cause of sudden cardiac death) belonging to high risk populations to be treated using Subcutaneous Implantable Cardioverter-Defibrillators (S-ICDs). S-ICDs carry a risk of so-called T Wave Over Sensing (TWOS), which can lead to inappropriate shocks with an inherent health risk. For this reason, according to current practice patients' Electrocardiograms (ECGs) are screened over 10 seconds to measure the T:R ratio (the ratio between the amplitudes of the T and R waves), which is a used as a marker for the likelihood of TWOS. Unfortunately, the temporal variability of a patient' T:R ratio can render this short screening procedure unreliable. In

this paper, we propose and investigate a tool based on deep learning for the automatic prediction of the T:R ratios of 10-second ECG segments. In particular, we evaluate different deep learning model architectures, assess a range of stochastic gradient descent based optimization methods for model training, perform hyperparameter tuning, and create ensembles of the best performing models. Our automated T:R ratio detection tool can enable clinicians to provide a more reliable assessment of whether a patient is eligible for S-ICD implantation by allowing for a significantly longer screening window which captures the behavior of the patient's T:R ratio better than the current practice.

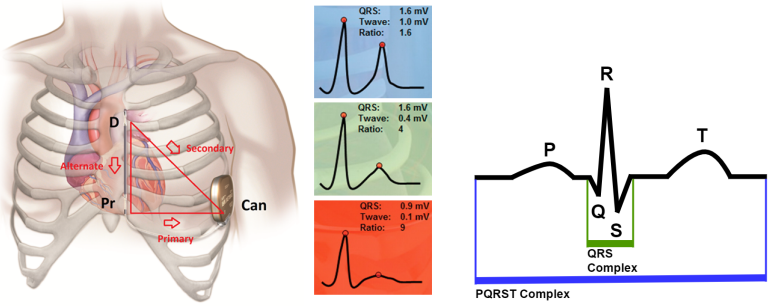
Keywords: deep learning, machine learning, optimization, subcutaneous implantable cardioverter defibrillators

1 Introduction

Sudden Cardiac Death (SCD) is one of the major causes of death worldwide. Most incidences of SCD can be attributed to ventricular arrhythmias [1]. For patients belonging to high risk populations, it is highly recommended that Implantable Cardioverter-Defibrillators (ICDs) be used for preventing ventricular arrhythmias from triggering an SCD [2, 3]. S-ICDs are comprised of a *can* and of a subcutaneous *lead*[1] (shown in Figure 1(a)), which can deliver a shock to treat episodes of ventricular arrhythmias. While S-ICDs have been proven to have equally effective sensing capabilities as other forms of ICD [4], S-ICDs entirely avoid the classical complications other ICDs suffer from by utilizing a totally avascular approach [5]. Unfortunately, the method that is currently used by the S-ICD to detect ventricular arrhythmias is known to be vulnerable to T Wave Over Sensing (TWOS)[6]. If the amplitude of the T and R waves are similar (Figure 1(b) shows a diagram of the PQRST complex—the ECG (Electrocardiogram) of a single heartbeat—comprised of 5 main waveforms, i.e., the P, Q, R, S, and T waves), the T wave can be misinterpreted as a second R wave and this apparent doubling in heart rate can be incorrectly identified by the S-ICD as an episode of ventricular arrhythmia. Such an error can cause the S-ICD to deliver an inappropriate shock, leading to an increased morbidity and mortality [6].

According to current practice, patients are screened before S-ICD implantation by analyzing short, non-invasive, three-lead, surface ECGs to assess their risk of TWOS. This ECG recording spans several PQRST complexes and, for its recording, the electrodes are placed in the locations that the sensing electrodes of an S-ICD would occupy. The *T:R ratio* (the ratio between the T wave's and the R wave's respective amplitudes) is used to predict a patients S-ICD implantation eligibility. Patients with high T:R ratios are deemed to have

[1]Using a single lead, the S-ICD is able to record ECG on 3 vectors—see Figure 1(a). When using a Holter ECG recording to assess patients' S-ICD implantation eligibility, numerous leads are used. We refer to a single lead of a Holter recording as the ECG recording for a single vector.



(a) (b)

Fig. 1 (a) An S-ICD with underlying anatomy showing the can (pulse generator), the sensing electrodes (Pr and D), and the vectors between them. (b) Diagram of a single PQRST complex, comprised of the P, Q, R, S, and T waves. The figure also labels the QRS complex (a section of the PQRST complex featuring the Q, R, and S waves only).

an increased risk of TWOS. As such, these patients are not eligible for S-ICD implantation. IT is current practice to measure the T:R ratio by locating a plastic template (designed to fit PQRST complexes with acceptably low T:R ratios (below 1/3) completely within its boundary) over the PQRST complexes within a recording of the patient's ECG. The patient passes the screening if the T:R ratio of at least one lead is suitably low. Despite this screening process, most of the inappropriate shocks that take place in patients implanted with S-ICDs are still caused by TWOS [5].

Crucially, the T:R ratio of a patient can vary over time due to frequent temporal variations in the R and T waves' respective amplitudes due to multiple factors, such as a variation in electrolyte levels [7–10]. In patients implanted with S-ICDs, these variations commonly go unnoticed as so-called *silent* TWOS episodes. As such, they carry a considerably high risk for a patient to be subjected to inappropriate shocks. The ECG recording currently used for the screening process has a short duration of roughly 10 seconds and, as such, it does not allow for taking the variance of the T:R ratio over an extended period into consideration. It follows that a patient whose T:R ratio is usually low could fail the screening and a patient whose T:R ratio is usually high could to pass it.

In this paper, we outline a methodology for constructing models capable of accurately and reliably predicting the T:R ratio of a given 10-second, single-lead ECG recording. A three-lead, 24-hour ECG recording can then be broken into its constituent 10-second, single-lead ECG signals which can then be passed into our model which would output a predicted T:R ratio for each ECG signal. This process would enable 24-hour automated screening, which, thanks to its significantly increased screening window, would provide far superior insight into the normal variations in the patient's T:R ratio as well as a greater level of detail and reliability than the currently utilized 10-second screening is capable of.

The method we propose is based on deep learning and, in particular, on Convolutional Neural Networks (CNNs). In ECG analysis, CNNs have been used for classifying atrial fibrillation [11, 12], heart attacks [13], and other arrhythmias [14–17], as well as for predicting blood pressure [18]. Differently from these works where the multi-lead ECG is used as input for the CNN, in our model we include the additional step of transforming the 1-dimensional ECG signal of each lead into a 2-dimensional phase-space reconstruction (PSR) image which is then fed as input to the CNN. PSR images are typically used for computer vision tasks which learn to extract relevant features from them but, to the best of our knowledge, have never been used as input to a CNN. As a second element of novelty and differently from most of the literature where CNNs are used for classification (see [19] for an exception), in this work we use them to perform a regression task.

In the literature, machine learning methods have been typically used to classify various Cardiovascular Diseases (CVDs) from ECG data [11, 12, 14, 15, 17, 20–23]. Machine learning has also been used for analyzing ECGs to detect heart attacks and seizures [13, 24] as well as to predict patients' blood pressure [18]. Brain computer interfaces (BCI) have been created using brain ECG analysis for detecting which body part was being used to complete a task [25, 26]. The creation of PSR images from ECG data is a commonly employed technique in ECG analysis. Manual approaches for extracting features from the PSR matrices (using box counting or calculating column and row statistics) have been used for, e.g., predicting CVD [20–23], creating BCIs [25, 26], and detecting facial expressions [27]. Differently from such works, in this paper we input the entirety of the PSR matrix to a (deep learning) model which, during it's training, learns to extract relevant features.. We are not aware of another work where the entire PSR image is used in such a way.

The paper is organized as follows. In Section 2, we first introduce our methodology and the preprocessing techniques we use to de-noise the ECG signals. We then detail the structure of the deep learning layers we use and the architectures of the deep learning models we propose for the task. We then outline the way our models are evaluated, the way we build ensemble models, and the way hyperparameter tuning is carried out (which is key to the performances we achieve). In Section 3, we assess the accuracy of our models and methods by selecting the best training method, tune the corresponding hyperparameters, and create ensemble models. In Section 4, we demonstrate how our best performing models can be incorporated into a clinical screening process for predicting the T:R ratio for a much longer time than what the current practice allows for.

2 Methodology

Predicting T:R ratios by locating T and R waves, measuring their peaks, and calculating the T:R ratio is vulnerable to TWOS when the T wave and R wave have similar characteristics. In this section, we propose a method for

Fig. 2 Flowchart providing an overview of the preprocessing method.

developing deep learning models which are able to predict the T:R ratio of a single-lead ECG segments without explicitly detecting R and T peaks.

2.1 ECG signal preprocessing

Before using the ECG data as input to our deep learning models, we preprocess it to calculate the T:R ratios, filter out noise from the ECG signal, and generate PSR images. This process, outlined in Figure 2 [28], is explained in this section.[2]

The data used in this paper consists of 10-second three-lead ECG signals collected from a Holter recording. The vectors used by the leads in this recording are the same vectors which are used by the S-ICD. These three-lead ECG recordings are then broken out into three single-lead ECG signals. We denote each 10-second segment with sampling frequency f by the vector $(x_1, \ldots, x_{10f})$. Each such segment is manually annotated with the positions of the T wave peaks at indexes $\{T_1, \ldots, T_n\}$ and R wave peaks at indexes $\{R_1, \ldots, R_n\}$. Using these annotations, we calculate the average T:R ratio (the dependent variable) as

$$\frac{\sum_{i=1}^{n} x_{T_i}}{\sum_{i=1}^{n} x_{R_i}}.$$

This fraction is negative when the T wave has negative amplitude. While, in clinical use, there is considered to be no difference between a positive and negative T:R ratio and, thus, only the magnitude of the T:R ratio is used (i.e. positive and negative T:R ratios are considered equivalent). However, when analyzing the single lead signals, ECG signals with either positive or negative T waves are significantly different. Therefore, when building models for predicting T:R ratios of single-lead ECG signals, we retain the T:R ratio's sign.

The 10-second, single-lead segments of ECG signal are filtered to remove various forms of noise. Firstly, a one-dimensional Discrete Wavelet Transformation (DWT) is used to correct the baseline drift. We then creates a 9 level decomposition of the ECG signal using the Daubechies 8 (db8) wavelet. The level 9 coefficients are used to reconstruct the wandering baseline of the ECG signal which is then subtracted from the original signal. The resulting signal has a stable baseline of 0. Secondly, the 50Hz power-line noise is suppressed using bandstop filtering. Finally, any remaining high frequency noise is suppressed using a lowpass filter. Figure 3a show an ECG segment with a

[2]The preprocessing techniques we use in this paper to derive PSR images from 10-second single-lead ECG segments were first illustrated in our preliminary work on the clinical impact of this type of tool [29]. For the sake of completeness, we briefly overview these methods here.

significant baseline drift before any filtering. Figure 3b shows the same after filtering has been applied.

TWOS is particularly likely in PQRST complexes with small R waves. To address this, the standard algorithm employed within S-ICDs will search the QRS complex (shown in Figure 1a) for negative Q or S waves with greater amplitude than that of the R wave. Provided that the amplitude of the largest wave in the QRS complex is significantly larger than that of the T wave, the S-ICD will not deliver a shock. For this reason, we are not strictly interested in predicting the T:R ratio but, rather, the ratio between the amplitudes of the wave of greatest amplitude in the QRS complex and the T wave. To account for this in our analysis, after filtering (including baseline drift correction), we search for the peak of greatest magnitude within a narrow region surrounding each R peak label. For simplicity, in the rare case that the largest peak found is not the R peak, these new peaks are assigned as the new R peak despite, in fact, being a Q or an S wave. As R waves are always positive, to ensure that all signals handed to our models have a positive wave labeled as the R wave, when the R peak label is moved to a Q or an S wave, the whole ECG signal is flipped, making these waves positive. Figure 3c gives an example of an ECG segment, which has had filtering applied, with a very small R wave and a large negative wave in the QRS complex. Figure 3d shows the same segment with it's reassigned R peaks after negative QRS flipping.

The one-dimensional time series $x_1, x_2, \ldots, x_n$ representing the filtered and potentially flipped single ECG signal of a single lead is then transformed to a two-dimensional Phase Space Reconstruction (PSR) image. While high-dimension PSR transformations have been used in the field of BCI [25, 26], two-dimensional PSR transformations are more commonly used in the literature on ECG analysis [20, 22, 23, 30]. The transformed time-series is given as

$$B = \begin{bmatrix} \frac{1}{q}x_1 & \frac{1}{q}x_{1+\tau} \\ \frac{1}{q}x_2 & \frac{1}{q}x_{2+\tau} \\ \vdots & \vdots \\ \frac{1}{q}x_{n-\tau} & \frac{1}{q}x_n \end{bmatrix},$$

where $q = \max\left\{|x_| : i = 1, \ldots, n\right\}$ and each scaled point $\frac{1}{q}x_i$ in the time-series signal is mapped to a *phase space vector* comprised of the original scaled point and the scaled point τ readings previous $\left[\frac{1}{q}x_{i-\tau}, \frac{1}{q}x_i)\right]$. The *phase space plot* (the plot of all phase space vectors in $\mathbb{R}^2$, ranging from -1 to $+1$ on each axis) is divided into N^2 squares which we denote by g_{ij} for all $i, j \in \{1, \ldots, N\}$, of size $\frac{2}{N} \times \frac{2}{N}$, with $N \in \mathbb{Z}^+$. C is the phase space matrix with dimension $N \times N$ and is constructed in such a way that, for each $i, j \in N$, the entry c_{ij} is equal to the number of phase space vectors in B that fall within the square area g_{ij}. We construct P by normalizing C such that, for each $i, j \in N$, the element p_{ij} corresponds to the proportion of all of the phase points which fall within g_{ij}.

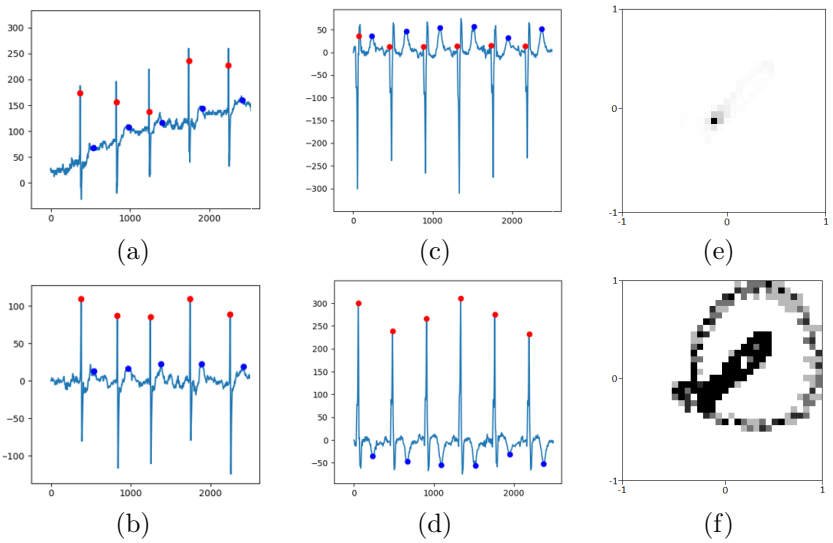


Fig. 3 (a) The first 5 seconds of an unfiltered 10-second ECG segment showing significant baseline drift with R peaks in red and T peaks in blue. (b) The signal shown in Figure 3a post-filtering. (c) The first 5 seconds of a 10-second ECG segment with very small R wave but a large negative wave in the QRS complex with R peaks in red and T peaks in blue. (d) The signal shown in Figure 3c after flipping and relabeling. (e) A 32×32 pixel PSR image created from 10-seconds of single-lead ECG signal. (f) The PSR image shown in Figure 3e darkened for readability.

We calculate P as follows:

$$P = \frac{1}{M}C, \quad M = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}. \tag{1}$$

At this point, the typical approach in the literature is to extract features of the PSR images to use as inputs to numerical models that are used for classifying ECG signals into categories [20, 22, 30]. Differently, in this paper we consider these matrices as $N \times N$ (with $N = 32$) pixel images and rely on deep learning model architectures typically used for computer vision to autonomously perform feature selection. Figure 3e gives an example of one of the PSR images. Figure 3f shows the same PSR image darkened for readability.

2.2 Neural network layers

Here, we briefly overview the 5 types of layer that we use within the architectures of our neural networks (presented in the next subsection).

A *dense layer* consists of a number of neurons, each of which is connected to all neurons in the previous layer. For each neuron of index i, the neuron's outputs z_i is given by

$$z_i = b_i + x \odot w_i,$$

where b is a vector containing a bias for each neuron in the layer, w_i is a vector of weights for the ith neuron containing a weight associated to each element in the input, and $\odot$ is the Hadamard product.

Next, we consider *convolutional layers*. In a standard convolutional layer, 3-dimensional filters, also referred to as *kernels*, are applied to the input at regular intervals (the size of these intervals is referred to as the *stride*) to transform it into a 3-dimensional output comprised of a number of 2-dimensional feature maps. Let k be the index of a feature map. The output for a neuron in row i and column j of the feature map k in a convolutional layer with 3-dimensional input x is given by

$$z_{ijk} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i'j'k'}\, w_{uvk'k} \qquad \text{with} \begin{cases} i' = i\, s_h + u \\ j' = j\, s_w + v, \end{cases}$$

where s_h and s_w are, respectively, the vertical and horizontal strides, f_h, f_w, and $f_{n'}$ are the height and width and depth of this layer's filters (where $f_{n'}$ is equal to the number of feature maps in the previous layer), b_k is the bias applied to the feature map k, and w_k is the matrix of weights defining the 3-dimensional filter used to generate the kth feature map. Hence, $w_{uvk'k}$ is the weight in row u and column v of the two dimensional slice of the filter which connects to feature map k' of the input. For a detailed introduction to CNNs, we refer the reader to [31].

We also consider *batchnorm layers*. Batch normalization, or *batchnorm*, is a tool for reparameterizing neural networks to significantly speed up training and reduce generalization error, allowing us to forgo the use of dropout (the process by which neurons are randomly turned on and off during training) [32]. For regularizing CNNs, batchnorm has been shown to perform better than dropout [33]. It can be applied to any layer and normalizes the distribution across the minibatch for each output in a layer, thus ensuring that throughout training the inputs to the following layer always have the same distribution [33]. Batchnorm is applied within its own layer. The minibatch outputs $z = (z^{(1)}, \ldots, z^{(m^B)})$ of a batch normalization layer with minibatch inputs $X = (X^{(1)}, \ldots, X^{(m^B)})$ are given by

$$\begin{aligned}
\mu^B &= \frac{1}{m^B} \sum_{n=1}^{m^B} X^{(n)} \\
\sigma_B \odot \sigma_B &= \frac{1}{m_B} \sum_{n=1}^{m_B} \left(X^{(n)} - \mu^B\right) \odot \left(X^{(n)} - \mu^B\right) \\
\hat{X}^{(n)} &= \left(X^{(n)} - \mu^B\right) \oslash \left(\sigma^B + \epsilon\right) \\
z^{(n)} &= \gamma \odot \hat{X}^{(n)} + \beta,
\end{aligned}$$

where m^B is the size of the minibatch and $X^{(n)}$ is the matrix containing the output from the previous layer for the nth data point in the minibatch. The outputs $z^{(n)}$, the minibatch mean μ_B, the standard deviation σ_B, and the scaling and shifting parameters γ and β are all matrices with the same shape as the input $X^{(n)}$. The learnable parameters in this layer are the vectors γ and β. The Hadamard product $\odot$ and Hadamard division $\oslash$ are used in these calculations. We apply the activation function in its own layer, after batchnorm is performed (as was done in the original paper [34] which presented batchnorm).

In our *activation layers*, we use the generally recommended activation function rectified linear unit or ReLU [35, 36] given by

$$R(z) = \max(0, z).$$

This function is applied element wise to the output of the layer preceding it.

The last layer we consider is a *skip connection layer*. *Addition skip connections* (as utilized in ResNet [37]) are used within our deep CNNs, whereby the output of one layer is added to the input of a layer located significantly deeper into the network. Models with skip connections have a smoother loss surface [38, 39], which makes optimizing the model parameters easier and thus speeds up training.

2.3 Model architectures

We now discuss the architectures of the deep learning models we use to predict T:R ratios from 32×32 pixel PSR images.

Each model consists of N feature-extraction blocks feeding into a regression block. Figure 4 shows this general structure. Following preliminary experiments [29] where a wide range of model architectures were accessed, we found two model architectures to provide a good level of accuracy: the MLP5 and Complex CNN5 models. Their names refer to the fact that, in these models, the regression block is preceded by 5 Multi Layer Perception (MLP) feature extraction blocks or by 5 Complex Convolutional Neural Network (Complex CNN) feature extraction blocks, respectively.

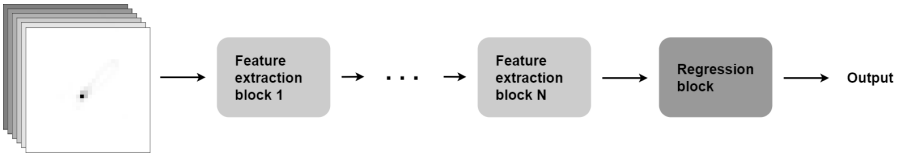


Fig. 4 The general structure of a model which takes PSR images representing 10-second ECG segments as input and outputs predicted T:R ratios.

Let us first consider the MLP5 model shown in Figure 5. Each of the 5 MLP feature extraction blocks has a dense layer of 1024 neurons, a batchnorm layer,

then an activation layer. These layers extract abstract features from the PSR images, and pass them on to the regression block. Before being used as input into the first 1-dimensional (1D) dense layer, the 32×32 pixel PSR images are flattened into a 1D vector. The regression block derives the T:R ratio from the features output from the final feature extraction block. It also utilizes two dense layers containing 256 and 64 neurons, each followed by batchnorm and activation layers before a final fully connected layer with a single neuron which generates the predicted T:R ratio. Further details on the structure of the MLP5 feature extraction blocks and the regression block are given in Tables A1 and A2 in the Appendix.

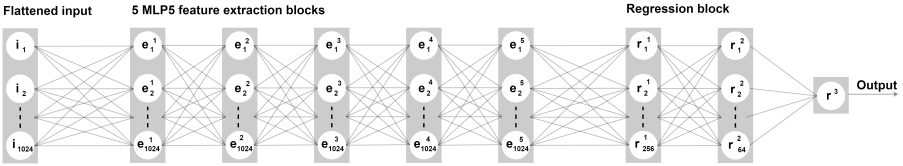


Fig. 5 A diagram of the MLP5 model, consisting of 5 MLP feature extraction blocks and the regression block. The batchnorm and activation layers are not visually represented. The neurons in the flattened input, feature extraction blocks, and regression block are labeled i, e, and r, respectively. Their superscript denotes which layer they belong to, while the subscript refers to their position in that layer.

Let us now focus on the Complex CNN5 model, in which five Complex CNN feature extraction blocks are used to extract features from the PSR image which are then flattened and passed into the regression block. The convolutional layers used within this model aim to exploit the multi-dimensional nature of the PSR image inputs. As convolutional layers take three-dimensional (3D) inputs, the original PSR images are considered to have shape $32 \times 32 \times 1$. Figure 6 shows the first feature extraction block of the Complex CNN5 model. The Complex CNN feature extraction block contains two convolutional layers (each followed by batchnorm and activation layers) with relatively small kernels which feed into a convolutional layer with stride 2 and a larger kernel (followed by batchnorm and activation layers). The first two convolutional layers extract features from the PSR images, while the third reduces the size of the feature maps and increases their number. While pooling layers are typically used to decrease the size of the feature maps, using a convolutional layer to do this allows an additional opportunity to extract more complex features from the outputs of the previous layers. Addition skip connections are also included, which add the input of the block to the output of the first two convolutional layers, speeding up training.[3] Each subsequent Complex CNN feature extraction block has twice as many feature maps as its predecessor had and the

[3]In preliminary experiments [29], a more basic CNN feature extraction block (with a single convolutional layer, no skip connection, and a pooling layer for dimensionality reduction) was evaluated and found to predict T:R ratios with far inferior accuracy.

height and width of these feature maps are half the size of those in the previous Complex CNN feature extraction block. The extracted features output by the 5th block are flattened and input into the regression block. Because of this halving in height and width of the feature maps, the kernel size of the convolutional layers must also decrease. Details on the exact structure of the Complex CNN feature extraction blocks can be found in Table A3 in the Appendix.

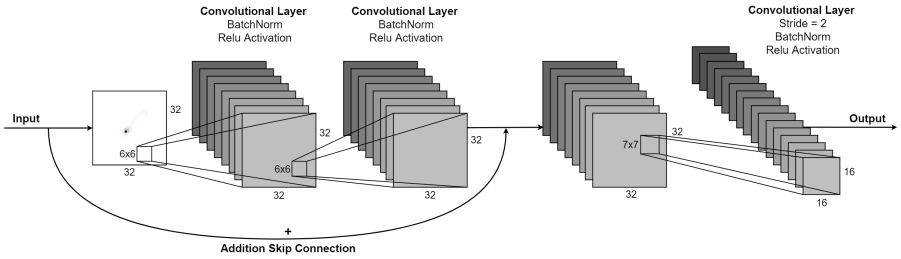


Fig. 6 A diagram of the first feature extraction block of the Complex CNN5 model. The batchnorm and activation layers are not visually represented.

2.4 Model evaluation

For each model architecture, we perform 10 rounds of 10-Fold CV. For each round of 10-fold CV, the data is shuffled before being split into 10 equally sized folds. Then, at each iteration of the 10-fold CV, one fold is reserved as the testing set. A randomly sampled 20% of the remaining 9 folds is used to form the validation which is used to prevent overfitting while training. The other 80% of the non-testing data is used as training set. Figure 7 illustrates our 10-fold CV process as well as the training of an ensemble model (see the next subsection for the introduction of ensemble models).

In this paper, our models are trained for at most 1000 epochs; however, we employ *early stopping* [40] for regularization in model training. It is often observed during training that both the training and validation errors initially decrease and, at some point, the validation error begins to increase while the training error continues to decrease. The error on the validation set serves as a proxy for the generalization error which is used to determine when overfitting has begun. Therefore, in order to achieve the lowest possible validation error, we reset the model parameters to their values before overfitting began and the validation error started increasing. This means that we continue training our model until the validation error has not improved for a given number of epochs, referred to as the *patience* of the optimization algorithm, instead of continuing to train until we reach a local minimum of the training error [41]. In all of our experiments, we use a patience of 200 epochs.

While, during the training of our models, we look to minimize the Mean Squared Error (MSE), when we come to evaluate the performance of our models in predicting T:R ratios of the testing PSR images there are additional

metrics we are interested in. We record the *training time* for each model as, should two models give comparable accuracy, the one with a shorter training time is preferable as it would be less computationally expensive to retrain it if and when new data became available. Similarly, we record the number of epochs the model trained for before early stopping kicked in. The *average prediction time*, i.e., the average amount of time taken for the model to predict each testing PSR image's T:R ratio, is also recorded. A lower average prediction time indicates that predicting T:R ratios in real time would be less computationally expensive and, therefore, the model would be more suitable for incorporation into a device with limited computational resources. It is worth noting here that we only compare the average prediction time of models with different architectures as, after training, the average prediction time for models with the same architecture should be consistent. We assess the accuracy of the models' predictions of the T:R ratios of the testing PSR images using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), defined by

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \qquad \text{MAE} = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}, \tag{2}$$

where, for each $i \in \{1, \ldots, n\}$, y_i is the true T:R ratio, $\hat{y}_i$ is the predicted T:R ratio, and n is the number of PSR images. RMSE, while similar to its squared counterpart the Mean Squared Error (MSE), does not see such a broad distribution of errors and, thus, lends itself to a graphical comparison of errors over multiple rounds of CV. MAE is the most intuitive of these errors and, therefore, it is important when discussing the impact of the prediction of a model.

2.5 Ensemble Models

As a final inclusion to improve the performance of our models, we construct and evaluate ensemble models.

Within each round of CV after the testing set has been excluded, the remaining data is split into 5 non-overlapping folds. 5 *sub-models* are then trained, each using one of these 5 folds as a validation set and the remaining 4 as a training set. The predictions of the ensemble models for the testing set are calculated by taking the average of the predictions of the sub-models. This is illustrated in Figure 7.

By creating an ensemble model, we ensure that each data point which is not used for testing will be used for training by all but one of the sub-models. In contrast, when training a single model, the data points in the validation set are never used for training.

Having determined the best model architecture, the best training algorithm, and the best hyperparameter selection, we will construct and evaluate an ensemble models at each iteration of the 10 rounds of 10-fold CV.

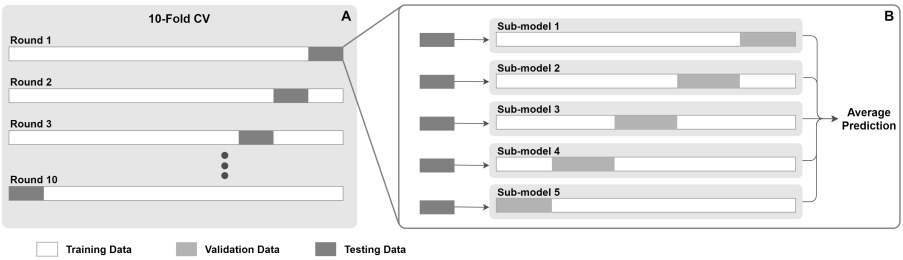


Fig. 7 Part A gives an overview of how training and testing sets are iteratively sampled within a single 10-fold CV. Part B illustrates how within each round of a single 10-fold CV, having reserved a testing set, 5 sub-models are trained with different training and validation sets and their predictions for the testing set are averaged giving the ensemble models prediction. If we consider an ensemble model with only one sub-model, this becomes exactly the same as the 10-fold CV process for training non-ensemble models, described in Section 2.4.

3 Experiments

In this study, we compare the use of various optimization techniques to fine tune our models for accurately predicting T:R ratios of S-ICD implantation candidates.

3.1 Data sets

To train and evaluate our models, we rely on the Southampton General Hospital ECG (SGH-ECG) data set, which was collected by our partners at the Southampton General Hospital for our preliminary study [29]. To construct the SGH-ECG data set, we randomly sampled 10-second ECG segments from 24-hour ECG recordings. A total of 390 segments (sampled at 500 Hz with annotated R and T peaks) were collected. These 24-hour recordings were originally collected for the clinical research study HEART TWO.[4] The dataset comprises a total of 18 patients with an equal number of males and females of a mean age of 53.16 years. 4 of them (22.22%) displayed a structurally-normal heart, 3 (16.67%) an adult congenital heart disease, 3 (16.67%) a hypertrophic cardiomyopathy, and 8 (44.44%) a congestive hart failure. Each of the 10-second ECG segments was used to generate a 32×32 pixel PSR images as well as its corresponding T:R ratio using the preprocessing techniques detailed in Subsection 2.1.

Due to the relatively small size of our data set, at each round of CV we combine the training data with 310 32×32 pixel PSR images with associated T:R ratios derived from the open source ECG Identification (ECG-ID) Database [28], containing 10-second ECG segments with a sampling rate of 500Hz, with R and T peak annotations. In this dataset, 44 and 46 of the participants were male and female respectively and the age of patients ranged from 13 to 75. We do not add this data to the testing and validation sets as

[4]This study was performed with favorable opinion from the REC (17/SC/0623) and with R&D (RHMCAR0528) approval. This study was conducted in accordance with the Research Governance Framework for Health and Social Care (2005), Good Clinical Practice and their relevant updates.

we are only interested in evaluating our the performance of our models for patients with specific heart conditions which the participants of the ECG-ID study did not have.

3.2 Training algorithms

In all of our experiments, training is carried out via minibatch Stochastic Gradient Descent (minibatch SGD), a variant of gradient descent that makes use of the fact that the average gradient on a minibatch of m randomly sampled data points is an unbiased estimator of the gradient on all the data points. Let X and Y be the sets of observations and labels and, for all $i = 1, \ldots, |X|$, let (x_i, y_i) be an observation-label pair. The formulas by which minibatch SGD calculates the unbiased estimate $\hat{g}$ of the gradient g and uses it to update the model parameters θ at each step are $\hat{g} := \frac{1}{m} \sum_{x_i \in \bar{X}} \nabla_\theta L(f(x_i;\ \theta), y_i)$ and $\theta := \theta - \epsilon\hat{g}$, where $\bar{X} \subseteq X$ is the set of m data points in the minibatch and $f(x_i)$ is the output of the model with x_i given as input. Training is carried out for a number of *epochs*. An epoch is comprised of a number of descent steps such that the gradient is calculated on each data point in the training set at least once.

A popular inclusion to the minibatch SGD algorithm is momentum (a technique designed to leverage the fact that, when the gradient is small but consistent across multiple descent steps, we can afford to take larger steps in that direction, thus achieving an accelerating learning [42]). In cases where the gradient is small, SGD can exhibit a slow convergence. When implementing the classical momentum method [43], at each descent step an exponentially decaying history of the previous descent steps is added to the product of the gradient of the loss function and the learning rate. In this paper, we implement *Nesterov momentum*, a variant of momentum wherein, rather than calculating the gradient using the current value of the model parameters θ, the gradient is calculated using the parameters after the momentum has been applied [44].

We consider 3 SGD variants with an adaptive learning rate in which the learning rate is scaled for each individual model parameter: AdaGrad (where the learning rates corresponding to each model parameter are scaled by the inverse of the sum of the squared partial derivatives over all training iterations [45]), RMSprop (an adaptation of AdaGrad designed to converge more effectively in non-convex optimization problems such as the training of multi-layer neural networks by diminishing the learning rate proportionally to the inverse of an exponentially decaying window of squared gradients), and ADAM [46] (in which an exponentially decaying history is kept of both the first and second order moments of the gradient). The formulas by which ADAM updates the model parameters θ at each iteration are $s := \rho_1 r + (1 - \rho_1)g$, $\hat{s} := \frac{s}{1-{\rho_1}^t}$, $r := \rho_2 r + (1-\rho_2) g \odot g$, $\hat{r} := \frac{r}{1-{\rho_2}^t}$, $\Delta\theta := -\epsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}$, and $\theta := \theta + \Delta\theta$, where t is the number of descent steps that the algorithm has been running for, ρ_1 and ρ_2 are additional hyperparameters, and $\odot$ is the Hadamard product.

The principal requirement when initializing the model parameters of a neural network (thereby choosing a starting point for the optimization method) for

training via SGD variants is to *break symmetry* [32]. To achieve this, we initialize the weights randomly using the Glorot normalized initialization method [47] with 0 mean and a variance of $\text{Var}[W] = \frac{2}{n_{\text{in}}+n_{\text{out}}}$, where W is the matrix of weights in a given layer, n_{in} is the number of nodes in the previous (input) layer, and n_{out} is the number of nodes in this layer's output. Differently from the original work, where a uniform distribution is used, in our work we sample the weights from a Normal distribution.

In this paper, we consider the choice of SGD based algorithm used to train the model parameters as a hyperparameter of the model itself which should be selected optimally. We then look to tune the hyperparameters of these SGD based algorithms, namely, batch size and global learning rate. As the 10 rounds of 10-fold CV is very computationally expensive, using gridsearch to search a 3D grid is infeasible. We instead tune the hyperparameters sequentially, first evaluating SGD, SGD with Nesterov Momentum, AdaGrad, RMSProp, RMSprop with momentum, and ADAM to determine the SGD based algorithm which gives the best accuracy with default hyperparameter selections. We then tune its batch size, and finally tune its global learning rate while using the batch size found in the previous step.[5]

3.3 Experiment 1: architecture comparison

Firstly, we compare the two model architectures proposed in Subsection 2.3, i.e., Complex CNN5 and MLP5.

In this experiment, we use ADAM as the minibatch SGD variant for training, as "...Adam works well in practice and compares favorably to other stochastic optimization methods"[46]. We use the generally accepted default selections of batch size 32 [48, 49] and learning rate 0.001 as suggested in the original paper [46]. These experiments are conducted using Tensorflow in Python 3.10.

Figure 8 comprises 5 violin plots of the distributions of 5 performance metrics (detailed in Subsection 2.4) over the 100 total rounds of CV for the MLP5 architecture (shown in blue) and the Complex CNN5 model (shown in red). The MLP5 model completes its training in, on average, 493 seconds. This is significantly quicker than the time of the Complex CNN5, whose average training time is 1262 seconds despite taking roughly the same number of epochs to complete training. This shorter training time is preferable as it would enable to model to be updated more regularly as new data becomes available.

The MLP5 models are found to be able to predict the testing PSR images' T:R ratios much quicker than the Complex CNN5 models, with an average prediction time per PSR image of 0.0075 seconds compared to an average of 0.0119 seconds for the Complex CNN5 one. This shorter prediction time is

[5]In our preliminary experiments [29], we assessed the use of image augmentation schemes. However, we did not find that they significantly improved the models' performance and, hence, we do not evaluate these methods here.

valuable, as it indicates that the MLP5 models would require fewer computational resources to make predictions of the T:R ratio in real time than the Complex CNN5 models would.

In our analysis, our primary concern is accuracy. Here, we consider RMSE and MAE as defined in Equation (2). The MLP5 models achieve an average RMSE of 0.105 and an average MAE of 0.0567 compared to the Complex CNN5 models' average RMSE of 0.111 and MAE of 0.0624. For these reasons, we conclude that the MLP5 architecture leads to better performance as well as to shorter training and prediction times.

In the subsequent experiments, we will look to select the best SGD variant for training these MLP5 models. We will not consider the prediction times in the subsequent analysis, as they should be the same for all models with the same architecture.

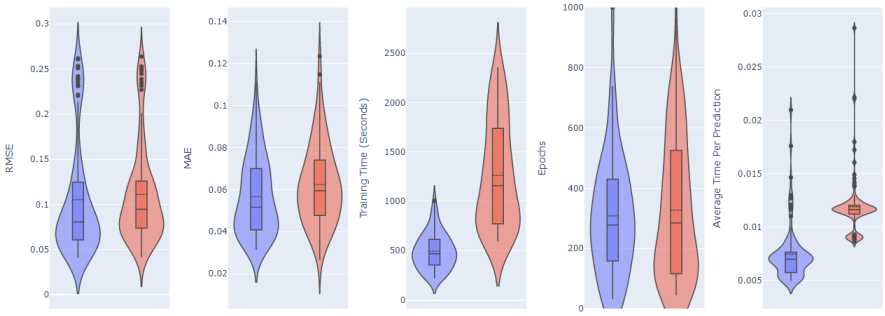


Fig. 8 Violin plots showing the distribution of the RMSE, MAE, training time, epochs, and average prediction time of the 100 MLP5 models (blue) and Complex CNN5 models (red) evaluated in the 10 rounds of 10-fold CV.

3.4 Experiment 2: optimizer comparison

In the previous experiment, we assumed ADAM was the preferred minibatch SGD variant. In this experiment, we test such an assumption by evaluating 6 minibatch SGD variants: SGD, SGD with Nesterov Momentum, AdaGrad, RMSProp, RMSprop with momentum, and ADAM. At this stage, we still use the default batch size and learning rate of, respectively, 32 and 0.001. We perform 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the 6 minibatch SGD variants for a total of 100 models trained using each minibatch SGD variant and evaluated on a distinct testing set.

Figure 9 shows the distribution of the RMSEs across the 100 models for each minibatch SGD variant (left) as well as an expanded version of the same graph (right). The trend shown in these results is that the prediction error is lower for SGD variants which are better suited to the non-convex setting (such is the case when training a deep neural network for T:R ratio prediction from PSR images). MLP5 models trained using ADAM exhibit an average RMSE of 0.105 compared to those trained using basic minibatch SGD, which

have an average RMSE of 0.121. We observe that the training time and the number of epochs before early stopping is generally greater for more complex versions of minibatch SGD such as for ADAM and RMSprop. This can be seen in Figure B1 in Section B of the Appendix.

With this experiment, we have confirmed that ADAM does indeed give the best accuracy (with a default hyperparameter selection) than a number of popular minibatch SGD algorithms and, hence, is the optimization algorithm we will use going forward.

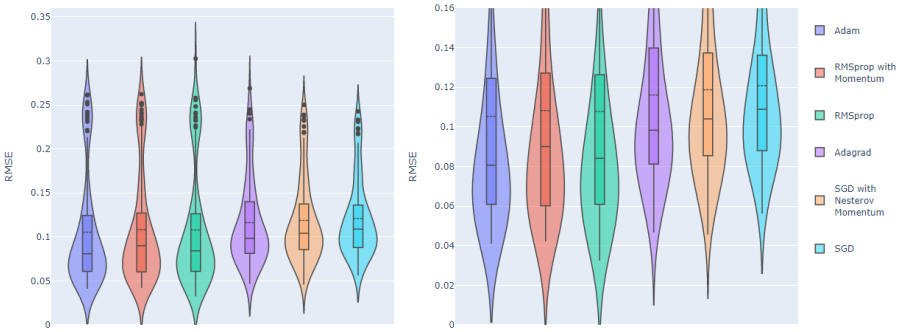


Fig. 9 Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

3.5 Experiment 3: tuning batch size

Batch size is a hyperparameter of all minibatch SGD based algorithms. In previous experiments, we used the generally recommended batch size of 32 [48, 49]. In this experiment, we evaluate batch sizes of 2^n for $n \in \{3, 4, 5, 6, 7, 8\}$ (while still using a learning rate of 0.001) by performing 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the 6 different batch sizes for a total of 100 models trained with each batch size and evaluated on a distinct testing set.

Figure 10 shows the distribution of the RMSEs across the 100 models trained with each batch size (left) as well as an expanded version of the same graph (right). As can be seen, increasing the batch size steadily decreases the RMSE up to a batch size of 128, which achieves an average RMSE of 0.1029, compared to the previously used batch size of 32 which leads to an average RMSE of 0.1053.

Figure B2 in Section B of the Appendix shows that, while the number of epochs before earlystopping increases as the batch size increases, the training time decreases. In one epoch, minibatches are propagated forward though the network and their errors are used to calculate gradients and update the parameters. This is repeated until all data has been seen at least once. This means that a large batch size results in the parameters being updated far fewer times per epoch. This is the reason why, in Figure B2, we see that the models with

a large batch size typically take more epochs to train as each epoch represents fewer descent steps. However, despite training for more epochs, the large reduction in the number of times the model parameters are updated results in a drastic reduction in training time for models trained with larger batch sizes.

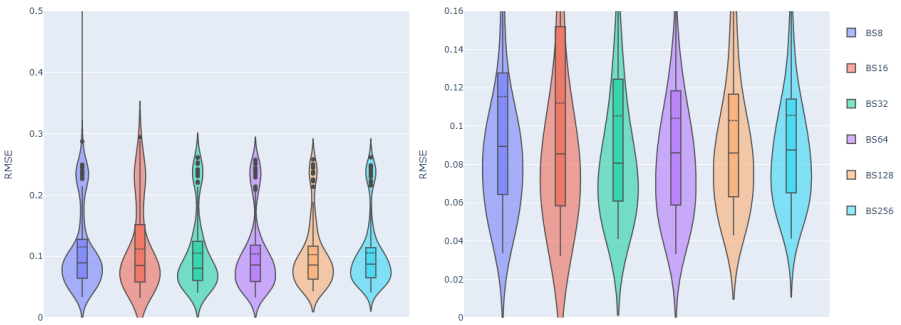


Fig. 10 Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

Using a batch size of 128 leads to the best average RMSE of 0.1029 while also resulting in a significant reduction in average training time compared to using a batch size 32 and, as such, 128 is the batch size we will use going forward.

3.6 Experiment 4: tuning learning rate

ADAM is considered robust to the choice of hyperparameters ρ_1 and ρ_2 and, as such, we only tune the global learning rate ϵ by searching a logarithmic range 10^{-n} for $n \in \{1, 2, 3, 4, 5, 6\}$ while using the best-performing batch size of 128 found in the previous experiment [41]. We evaluate these learning rates by carrying out 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the different learning rates for a total of 100 models trained with each learning rate and evaluated on a distinct testing set.

Figure 10 shows the distribution of the RMSEs across the 100 models trained with each learning rate (left) as well as an expanded version of the same graph (right). Figure 10 does not show the results for models trained with a learning rate of 10^{-5} and 10^{-6}. However, the trend demonstrated in Figure 10 (that, for learning rates less than 0.01, the average RMSE increases as the learning rate decreases) continues for these values. We observe that using a learning rate of 0.01 gives an average RMSE of 0.1020 as opposed to the RMSE of 0.1029 given by the previously used default of 0.001.

Figure B3 in Section B of the Appendix shows that, as we would expect, the number of epochs increases for smaller learning rates and, as the batch size is fixed at 128 in this experiment, the training time is proportional to the number of epochs and so it also increases as the learning rate becomes smaller.

We conclude from this experiment that a MLP5 model trained using ADAM with a batch size of 128 and a learning rate of 0.01 offers improved accuracy while taking less time to train.

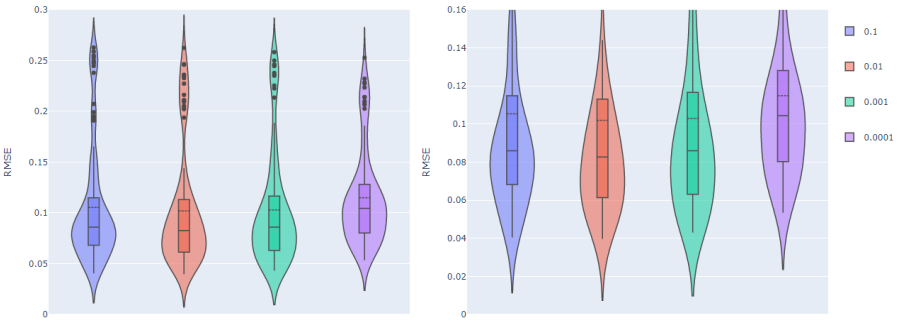


Fig. 11 Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

Figure 12 shows the average training and validation error at each epoch of each of the 100 models trained using ADAM with a batch size of 128 and a learning rate of 0.01 in each of the 100 rounds of CV performed in this experiment. As can be seen, while the validation error is higher than the training error during training, both decrease rapidly before stabilizing and reduce slowly over the rest of the training. For the purposes of this plot where early stopping occurs, the errors are kept constant for the remainder of the 1000 epochs.

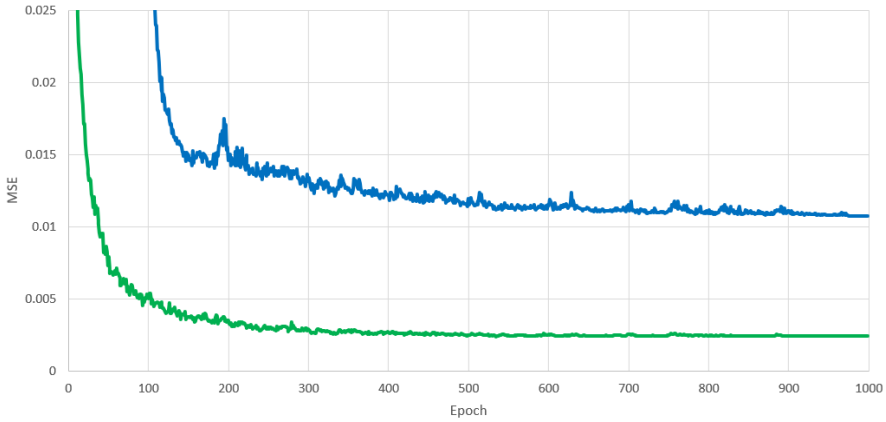


Fig. 12 The average training and validation loss over the 100 CV rounds over 1000 epochs.

3.7 Experiment 5: creating ensemble models

In the previous experiments, we found that the a *tuned* MLP5 model trained using ADAM with a batch size of 128 and a global learning rate of 0.01 gave the best accuracy. In this experiment, we assess the benefits of ensemble models by performing 10 rounds of 10-fold CV and, in each fold, training 5 MLP5 submodels, for a total of 100 ensemble models each containing 5 tuned MLP5 submodels. We compare each of these MLP5 ensemble models to a tuned MLP5 model and to a 'default' MLP5 model trained using ADAM with a batch size of 32 and a learning rate of 0.001 (the naive hyperparameter selections we initially considered to be good defaults).

Figure 10 shows the distribution of the RMSEs of the 100 MLP5 ensemble models on their respective testing sets, compared with those of the tuned MLP5 models and the default MLP5 models (left) as well as to an expanded version of the same graph (right). As can be seen, the default MLP5 models achieve an average RMSE of 0.1052 while the tuned MLP5 models average 0.1024 and the ensemble MLP5 models average 0.0928. This represents a reduction in average RMSE of 12% for ensemble MLP5 models but only 2% for the tuned MLP5 models. If, however, we examine the upper quartile of these distributions, we see that both the ensemble MLP5 models and the tuned MLP5 models offer a more significant improvement over the base MLP5 models. The ensemble MLP5 models' RMSE distribution has an upper quartile of 0.1002 compared to the tuned MLP5 models' 0.1147 and to the base MLP5 models' 0.1259, which is a reduction of 9% for the tuned models and 20% for the ensemble models. We do not examine the number of epochs or the training duration for this experiment as the number of epochs used to train each submodel is expected to be the same as for the tuned MLP5 models and, likewise, the training time for the whole ensemble MLP5 model is expected to be simply 5 times that of a single tuned MLP5 model.

To this point, we have focused on RMSE because it is makes the violin plots we have used more easily interpretable than MSE, and it provides a stronger penalization to large prediction errors than MAE. However, in order to put the performance of our model in context, we will once again examine MAE, as it as very easy to interpret as the average absolute error across each prediction. The default MLP5 models achieve an average MAE of 0.0567 compared to our best performing model, the ensemble MLP5 model, which achieves an average MAE of 0.0461. This represents a substantial reduction in average absolute prediction error (MAE) of 19%.

4 Discussion

In the previous section, we showed that, through optimization algorithm selection, hyperparameter tuning, and creating ensemble models, we were able to reduce the average MAE across the 100 CV rounds to 0.0461, a reduction of 19%, and the mean and upper quartile of the distribution of RMSEs over the 100 CV rounds by 12% and 20% respectively.

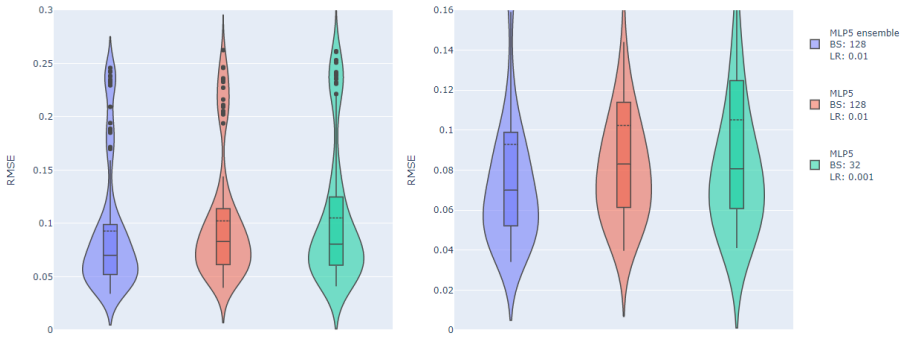


Fig. 13 Left: Violin plots showing the distribution of the RMSEs of the 100 ensemble MLP5 models (blue), tuned MLP5 models (red), and default MLP5 models (red) evaluated in the 10 rounds of 10-fold CV. Right: The same graph resealed for readability.

In this section, we discuss the clinical impact of our model's ability to predict the T:R ratios of 10-second, single-lead, ECG segments. As described in Subsection 2.1, from a clinical perspective the sign of the T:R ratio is irrelevant. Depending on the sign of the T wave, our model is expected to output both positive and negative predicted T:R ratios. For use in clinical analysis, we simply compute the absolute value of the model outputs.

Figure 14 shows the predicted absolute T:R ratios for each PSR image (notice that, during each round of 10-fold CV, each PSR image is used for testing exactly once) generated using the SGH-ECG data set when it was reserved for testing in 10-fold CV, plotted against their true absolute T:R ratios. The green line represents a perfect labeling. As can be seen, the absolute value of the outputs of the ensemble MLP5 model predict the true absolute T:R ratios with very low error.

Having established that this model can be used to accurately and autonomously predict T:R ratios from 10-second ECG segments, we can now use it to efficiently perform screening to assess the normal variations in the patients' T:R ratios across multiple leads over a 24-hour period. 8640 non-overlapping 10-second, single-lead ECG segments are created by breaking down a 24-hour, three-lead ECG recording. The 10-second segments are transformed following the process laid out in Subsection 2.1. This results in 8640 chronologically ordered PSR images. For each lead, a time series of T:R ratios can be generated by inputting the PSR images into the model. As previously mentioned, for clinical use we consider the absolute values of the predicted T:R ratio. These time series of T:R ratios corresponding to each lead of the ECG recording can enable clinicians to easily analyze the normal behaviors of the patient's T:R ratio over a much longer screening window than the 10-second screening process currently used.

Using this method we can create Figure 15, showing the variation of the T:R ratio on each lead over a 24-hour period. For readability, we have smoothed the time series of T:R ratios by plotting a moving average with a window of half an hour.

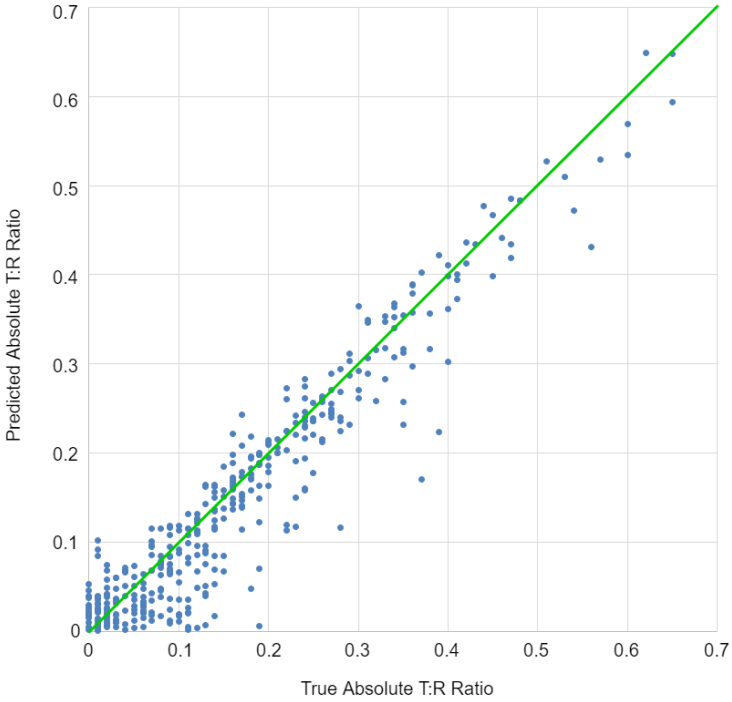


Fig. 14 Graph of the absolute value of true T:R ratios for the SGH-ECG data set plotted against our model's predicted T:R ratios during testing. For readability, 5 outliers have been removed.

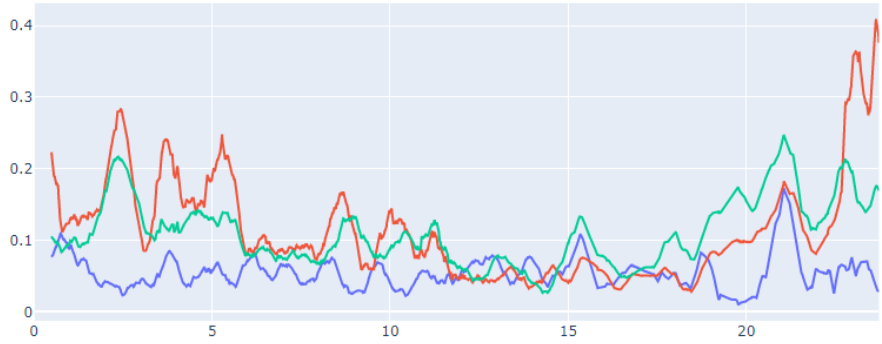


Fig. 15 Graph showing the behavior of the patient's T:R ratio during the 24-hour screening period for the three leads using the same vectors that would be used by an S-ICD.

5 Conclusions

TWOS is the leading cause of inappropriate shocks for S-ICD implanted patients. Due to the fact that the T:R ratio can vary significantly over time, the current screening process of using a 10 seconds ECG recording to estimate a patient's typical T:R ratio does not reliably capture the normal behavior of

the patient's T:R ratio. In this paper, we have developed models which enable autonomous T:R ratio prediction from PSR images of 10-second, single-lead ECG signals with a high degree of accuracy. These models can be incorporated into a tool enabling clinicians to perform 24-hour, automated screenings to assess patients' normal T:R ratio variations with greater detail and reliability.

Following 100 total rounds of CV, we have determined that the MLP5 model is preferable to the Complex CNN5 model as it has a shorter training time, can make predictions faster, and, most importantly, it predicts T:R ratios with higher accuracy (with an average RMSE of 0.105 compared to the Complex CNN5's average RMSE of 0.111).

Creating ensemble models comprised of MLP5 models, trained using the best-performing SGD based optimization method and with a tuned batch size and global learning rate, we were able to greatly reduce the prediction error compared to the models which used default selections for these hyperparameters. We were able to reduce the average MAE across the 100 CV rounds to 0.0461, a reduction of 19%, and the mean and upper quartile of the distribution of RMSEs over the 100 CV rounds by 12% and 20% respectively.

Currently, we are using our proposed screening tool to assess the temporal variations of the T:R ratio in patients belonging to patient groups who are likely to be implanted an S-ICDs. We are also investigating whether the increased detail provided by our tool can allow the cut-off threshold for the screening to be increased above the current value of 1/3, allowing patients who had, under the previous screening process, been determined to be at too high a risk of TWOS to be safely implanted with S-ICDs. Finally, we hope to determine in future research if fluctuations in a patients' T:R ratios can provide indication that a cardiac episode is impending.

Acknowledgments. The work of Anthony J. Dunn is jointly funded by Decision Analysis Services Ltd and EPSRC through the studentship with Reference EP/R513325/1. The work of Alain B. Zemkoho is supported by the EPSRC grant EP/V049038/1. The work of S. Coniglio and Alain B. Zemkoho is supported by The Alan Turing Institute under the EPSRC grants EP/N510129/1 and EP/W037211/1.

References

[1] Adabag, A.S., Luepker, R.V., Roger, V.L., Gersh, B.J.: Sudden cardiac death: epidemiology and risk factors. Nature Reviews Cardiology **7**(4), 216–225 (2010)

[2] Kusumoto, F.M., Bailey, K.R., Chaouki, A.S., Deshmukh, A.J., Gautam, S., Kim, R.J., Kramer, D.B., Lambrakos, L.K., Nasser, N.H., Sorajja, D.: Systematic review for the 2017 aha/acc/hrs guideline for management of patients with ventricular arrhythmias and the prevention of sudden cardiac death: a report of the american college of cardiology/american

heart association task force on clinical practice guidelines and the heart rhythm society. Circulation **138**(13), 392–414 (2018)

[3] Priori, S.G., Blomström-Lundqvist, C., Mazzanti, A., Blom, N., Borggrefe, M., Camm, J., Elliott, P.M., Fitzsimons, D., Hatala, R., Hindricks, G., *et al.*: 2015 esc guidelines for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death. the task force for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death of the european society of cardiology. Giornale italiano di cardiologia (2006) **17**(2), 108–170 (2016)

[4] Boersma, L., Barr, C., Knops, R., Theuns, D., Eckardt, L., Neuzil, P., Scholten, M., Hood, M., Kuschyk, J., Jones, P., *et al.*: Implant and midterm outcomes of the subcutaneous implantable cardioverter-defibrillator registry: the effortless study. Journal of the American College of Cardiology **70**(7), 830–841 (2017)

[5] Knops, R.E., Olde Nordkamp, L.R., Delnoy, P.-P.H., Boersma, L.V., Kuschyk, J., El-Chami, M.F., Bonnemeier, H., Behr, E.R., Brouwer, T.F., Kääb, S., *et al.*: Subcutaneous or transvenous defibrillator therapy. New England Journal of Medicine **383**(6), 526–536 (2020)

[6] van Rees, J.B., Borleffs, C.J.W., de Bie, M.K., Stijnen, T., van Erven, L., Bax, J.J., Schalij, M.J.: Inappropriate implantable cardioverter-defibrillator shocks: incidence, predictors, and impact on mortality. Journal of the American College of Cardiology **57**(5), 556–562 (2011)

[7] Madias, J.E., Bazaz, R., Agarwal, H., Win, M., Medepalli, L.: Anasarca-mediated attenuation of the amplitude of electrocardiogram complexes: a description of a heretofore unrecognized phenomenon. Journal of the American College of Cardiology **38**(3), 756–764 (2001)

[8] Madias, J.E.: Qtc interval in patients with changing edematous states: implications on interpreting repeat qtc interval measurements in patients with anasarca of varying etiology and those undergoing hemodialysis. Pacing and clinical electrophysiology **28**(1), 54–61 (2005)

[9] Fosbøl, E.L., Seibæk, M., Brendorp, B., Torp-Pedersen, C., Køber, L., Investigations, D., *et al.*: Prognostic importance of change in qrs duration over time associated with left ventricular dysfunction in patients with congestive heart failure: the diamond study. Journal of cardiac failure **14**(10), 850–855 (2008)

[10] Assanelli, D., Di Castelnuovo, A., Rago, L., Badilini, F., Vinetti, G., Gianfagna, F., Salvetti, M., Zito, F., Donati, M.B., De Gaetano, G., *et al.*: T-wave axis deviation and left ventricular hypertrophy interaction in diabetes and hypertension. Journal of electrocardiology **46**(6), 487–491

(2013)

[11] Fan, X., Yao, Q., Cai, Y., Miao, F., Sun, F., Li, Y.: Multiscaled fusion of deep convolutional neural networks for screening atrial fibrillation from single lead short ecg recordings. IEEE journal of biomedical and health informatics **22**(6), 1744–1753 (2018)

[12] Pourbabaee, B., Roshtkhari, M.J., Khorasani, K.: Deep convolutional neural networks and learning ecg features for screening paroxysmal atrial fibrillation patients. IEEE Transactions on Systems, Man, and Cybernetics: Systems **48**(12), 2095–2104 (2018)

[13] Liu, W., Zhang, M., Zhang, Y., Liao, Y., Huang, Q., Chang, S., Wang, H., He, J.: Real-time multilead convolutional neural network for myocardial infarction detection. IEEE journal of biomedical and health informatics **22**(5), 1434–1444 (2017)

[14] Kiranyaz, S., Ince, T., Gabbouj, M.: Real-time patient-specific ecg classification by 1-d convolutional neural networks. IEEE Transactions on Biomedical Engineering **63**(3), 664–675 (2015)

[15] Zhang, J., Liu, A., Gao, M., Chen, X., Zhang, X., Chen, X.: Ecg-based multi-class arrhythmia detection using spatio-temporal attention-based convolutional recurrent neural network. Artificial Intelligence in Medicine **106**, 101856 (2020)

[16] Sangaiah, A.K., Arumugam, M., Bian, G.-B.: An intelligent learning approach for improving ecg signal classification and arrhythmia analysis. Artificial intelligence in medicine **103**, 101788 (2020)

[17] Lih, O.S., Jahmunah, V., San, T.R., Ciaccio, E.J., Yamakawa, T., Tanabe, M., Kobayashi, M., Faust, O., Acharya, U.R.: Comprehensive electrocardiographic diagnosis based on deep learning. Artificial intelligence in medicine **103**, 101789 (2020)

[18] Miao, F., Wen, B., Hu, Z., Fortino, G., Wang, X.-P., Liu, Z.-D., Tang, M., Li, Y.: Continuous blood pressure measurement from one-channel electrocardiogram signal using deep-learning techniques. Artificial Intelligence in Medicine **108**, 101919 (2020)

[19] Babu, G.S., Zhao, P., Li, X.-L.: Deep convolutional neural network based regression approach for estimation of remaining useful life. In: International Conference on Database Systems for Advanced Applications, pp. 214–228 (2016). Springer

[20] Vemishetty, N., Gunukula, R.L., Acharyya, A., Puddu, P.E., Das, S., Maharatna, K.: Phase space reconstruction based cvd classifier using

localized features. Scientific reports **9**(1), 1–18 (2019)

[21] Vemishetty, N., Acharyya, A., Das, S., Ayyagari, S., Jana, S., Maharatna, K., Puddu, P.E.: Classification methodology of cvd with localized feature analysis using phase space reconstruction targeting personalized remote health monitoring. In: 2016 Computing in Cardiology Conference (CinC), pp. 437–440 (2016). IEEE

[22] Rocha, T., Paredes, S., De Carvalho, P., Henriques, J., Antunes, M.: Phase space reconstruction approach for ventricular arrhythmias characterization. In: 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 5470–5473 (2008). IEEE

[23] Roberts, F.M., Povinelli, R.J., Ropella, K.M.: Identification of ecg arrhythmias using phase space reconstruction. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 411–423 (2001). Springer

[24] Lee, S.-H., Lim, J.S., Kim, J.-K., Yang, J., Lee, Y.: Classification of normal and epileptic seizure eeg signals using wavelet transform, phase-space reconstruction, and euclidean distance. Computer methods and programs in biomedicine **116**(1), 10–25 (2014)

[25] Djemal, R., Bazyed, A.G., Belwafi, K., Gannouni, S., Kaaniche, W.: Three-class eeg-based motor imagery classification using phase-space reconstruction technique. Brain sciences **6**(3), 36 (2016)

[26] Chen, M., Fang, Y., Zheng, X.: Phase space reconstruction for improving the classification of single trial eeg. Biomedical Signal Processing and Control **11**, 10–16 (2014)

[27] Dawid, A.: Psr-based research of feature extraction from one-second eeg signals: a neural network study. SN Applied Sciences **1**(12), 1–12 (2019)

[28] Lugovaya, T.S.: Biometric human identification based on electrocardiogram. Master's thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University 'LETI', Saint-Petersburg, Russian Federation (2005)

[29] Dunn, A.J., ElRefai, M.H., Roberts, P.R., Coniglio, S., Wiles, B.M., Zemkoho, A.B.: Deep learning methods for screening patients' s-icd implantation eligibility. Artificial Intelligence in Medicine **119**, 102139 (2021)

[30] Krishnan, S.M., Dutt, D.N., Chan, Y., Anantharaman, V.: Phase space analysis for cardiovascular signals. In: Advances in Cardiac Signal Processing, pp. 339–354. Springer, Berlin, Germany (2007)

[31] Wu, J.: Introduction to convolutional neural networks. National Key Lab for Novel Software Technology. Nanjing University. China **5**(23), 495 (2017)

[32] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT press, Cambridge, Massachusetts (2016)

[33] Garbin, C., Zhu, X., Marques, O.: Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications, 1–39 (2020)

[34] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015). PMLR

[35] Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Icml (2010)

[36] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011). JMLR Workshop and Conference Proceedings

[37] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

[38] Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K.W.-D., McWilliams, B.: The shattered gradients problem: If resnets are the answer, then what is the question? In: International Conference on Machine Learning, pp. 342–350 (2017). PMLR

[39] Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. arXiv preprint arXiv:1712.09913 (2017)

[40] Prechelt, L.: Early stopping-but when? In: Neural Networks: Tricks of the Trade, pp. 55–69. Springer, Berlin/Heidelberg, Germany (1998)

[41] Bengio, Y., Goodfellow, I., Courville, A.: Deep Learning vol. 1. MIT press Massachusetts, USA, Cambridge, Massachussets and London, England (2017)

[42] Bishop, C.M., *et al.*: Neural Networks for Pattern Recognition. Oxford university press, Cambridge, Massachussets and London, England (1995)

[43] Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. Ussr computational mathematics and mathematical physics

4(5), 1–17 (1964)

[44] Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning, pp. 1139–1147 (2013). PMLR

[45] Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research **12**(7) (2011)

[46] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[47] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256 (2010). JMLR Workshop and Conference Proceedings

[48] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In: Neural Networks: Tricks of the Trade, pp. 437–478. Springer, Berlin/Heidelberg, Germany (2012)

[49] Masters, D., Luschi, C.: Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612 (2018)

Appendix A Model architectures

In this section, we provide details on the regression and feature-extraction blocks used in our models. Details on the feature extraction block that is used to construct the MLP5 and Complex CNN5 models are reported in Table A1. Details on the feature extraction blocks used for the MLP5 models are reported in Table A2. The structures of the 5 Complex CNN feature extraction blocks used within the Complex CNN5 model are given by Table A3, where n denotes which of the 5 feature extraction blocks we are referring to.

Table A1 Regression Block

Layer	Type	Output size
1	Dense	256
2	BatchNorm	256
3	Activation(Relu)	256
4	Dense	64
5	BatchNorm	64
6	Activation(Relu)	64
7	Dense	1

Table A2 MLP Feature Extraction Block

Layer	Type	Output size
1	Dense	1024
2	BatchNorm	1024
3	Activation(Relu)	1024

Table A3 Complex CNN Feature Extraction Block n

Layer	Type	Output size	Kernel size	Stride
1	Convolutional	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$	$(6-n) \times (6-n)$	1
2	BatchNorm	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$		
3	Activation(Relu)	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$		
4	Convolutional	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$	$(6-n) \times (6-n)$	1
5	BatchNorm	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$		
6	Activation(Relu)	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$		
7	Skip(Input)	$2^{n+2} \times 2^{6-n} \times 2^{6-n}$		
8	Convolutional	$2^{n+3} \times 2^{5-n} \times 2^{5-n}$	$(7-n) \times (7-n)$	2
9	BatchNorm	$2^{n+3} \times 2^{5-n} \times 2^{5-n}$		
10	Activation(Relu)	$2^{n+3} \times 2^{5-n} \times 2^{5-n}$		

Appendix B Experiment graphs

In this section, we provide additional figures from the experiments reported in Section 3. Figure B1 shows the training time and number of epochs before early stopping for the various SGD variants evaluated in Subsection 3.4, Figure B2 shows the training time and number of epochs before early stopping for the various batch sizes evaluated in Subsection 3.5, and Figure B3 shows the training time and number of epochs before early stopping for the various global learning rates evaluated in Subsection 3.6.

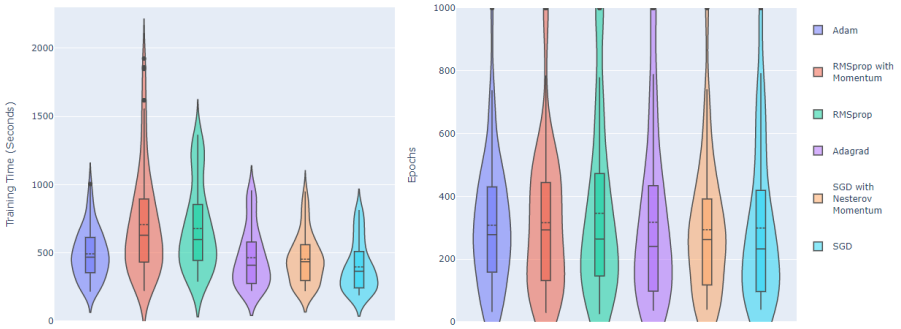


Fig. B1 Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV.

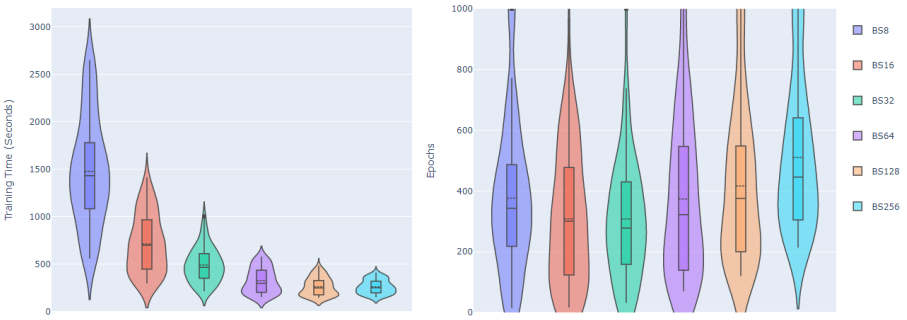


Fig. B2 Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV.

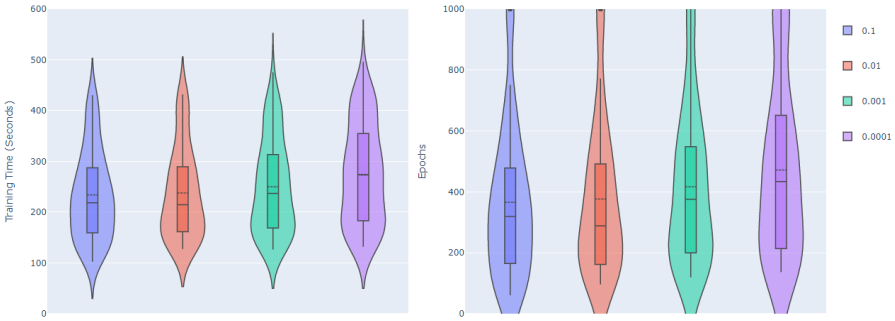


Fig. B3 Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV.