Application of Neural Networks to Predict Patient-Specific Cellular Parameters in Computational Cardiac Models

Chang Hi Lee

Washington University in St. Louis

McKelvey School of Engineering

Department of Computer Science and Engineering

Application of Neural Networks to Predict Patient-Specific Cellular Parameters in Computational Cardiac Models

by

Chang Hi Lee

A thesis presented to the McKelvey School of Engineering of Washington University in St. Louis in partial fulfillment of the requirements for the degree of Master of Science.

May 2022
St. Louis, Missouri

Table of Contents

List of Tables

List of Figures

Acknowledgments

I wish to acknowledge Dr. Jon Silva for his guidance and patience throughout the time I have worked with him. Without Dr. Silva's generous support, I would not have been able to join the Master's program in Computer Science. His deep expertise in computational cardiology and equally beneficial mentorship were crucial to the development of this study. I also wish to acknowledge Dr. Jonathan Moreno for his instrumental help in developing the cardiac model and computational experiments used in this study, as well as being another valuable mentor to me. Fellow lab members Dr. Kathryn Mangold and Druv Bhagavan were incredible sources of knowledge and also helped to guide my research. A special thanks goes to my Master's Thesis committee members, Dr. Michael Brent and Dr. Neal Patwari, for overseeing my defense.

Chang Hi Lee

Washington University in St. Louis
May 2022

Dedicated to my parents, Han Goo Lee and Jung Mi Kwon, for their encouragement and support across the sea, and to my sister Yeron Lee for cheering me up and taking care of my parents when I could not. Dedicated to my future wife Sun Hye(Sunny) Chung, for sticking with me through all the struggles we faced and always having my back.

I praise God, who I love with all my heart, all my soul, all my strength, and all my mind. I praise Jesus Christ, who redeemed me in my darkest hour through His steadfast love.

ABSTRACT OF THE THESIS

Application of Neural Networks to Predict Patient-Specific Cellular Parameters in Computational Cardiac Models

by

Chang Hi Lee

Master of Science in Computer Science

Washington University in St. Louis, 2022

Research Advisor: Jonathan Silva

Sudden cardiac death(SCD) is a significant cause of death that accounts for more than 180,000 deaths in the US and 4 million deaths worldwide annually. SCDs are mainly caused by irregular heartbeats called arrhythmias, which are caused by abnormal electrical activity within the heart. Precision medicine, in the form of personalized computational cardiac models of a patient's heart, can suggest optimal therapies for cardiac arrhythmias. Prior work has used finite element meshes derived from cardiac MRIs to simulate cardiac electrical activity. In this study, I sought to augment this approach by developing a neural network to learn the parameters that describe patient-specific cellular activity. The learned parameters can be combined with a 3-D cardiac mesh to achieve more accurate simulations. I found that all parameters except one could be predicted with low error using the current approach. Thus, there exists a parameter subset that is well-predicted and could be used for initial approaches to patient-specific modeling.

Chapter 1

Introduction

Sudden cardiac death(SCD) accounts for 25% of 17 million cardiac deaths worldwide every year[5]. Within the United States, SCD causes around 180,000 to 250,000 deaths annually[2]. SCDs are mainly caused by abnormal electrical activities in the heart, which can cause irregular heartbeats called cardiac arrhythmias. An arrhythmic heart can malfunction and fail to pump blood properly, leading to a life-threatening situation for the patient.

Precision medicine, an approach to medical care that takes into account the unique conditions of individual patients, is a rapidly rising trend in the healthcare sector [13, 23]. One way that precision medicine can be applied to the preemptive diagnosis and treatment of cardiac arrhythmias is through constructing patient-specific computational cardiac models. Cardiac models seek to simulate the electrophysiological activities in a patient's heart in order to detect abnormal electric signals that can cause arrhythmias.

Cardiac models first start at the cellular level, simulating the electrophysiological behavior of a single cardiac cell, called a cardiomyocyte. Cardiomyocytes generate electrical activities called action potentials, which are "spikes" of voltages across cell membranes that propagate throughout interconnected cardiomyocytes and cause a flow of current. When the total activities of cardiomyocytes in a person's heart are viewed on the organ scale, the collective action potential propagation can be observed as a "wave" of electrical activity across the heart, which regulates the contraction and expansion of a heart.

Much like how numerous myocyotes are connected together to eventually form the full heart, these single-cell simulations can be linked to form a 3-D simulation conducted at the organ level. The 3-D simulations can model the electrophysiological behavior of a patient's heart, and different experiments can be conducted to test for any abnormal electrical behavior that might occur.

When applied to cardiology, precision medicine can help simulate a more realistic version of the patient's heart and increase chances of arrhythmia detection by creating patient-specific cardiac models that incorporate the unique biological parameters of each patient. Arevalo et al. (2016)[3] developed personalized 3-D cardiac models through creating a 3-D finite element mesh(FEM) from cardiac MRI scans. By conducting cardiac simulations on this personalized heart structure, they demonstrated that the method can predict future arrhythmias better than existing clinical metrics. Giffard-Roisin et al. (2018)[7] used non-invasive data to personalize cardiac models to predict model responses to different electrical stimuli.

This study explored how precision medicine can further be applied to computational cardiac models to improve the level of personalization to a patient. This was achieved in a two-pronged approach.

First, I sought to develop a single-cell computational cardiac model that was personalized to fit cellular characteristics of a patient with hypertrophic cardiomyopathy(HCM), a cardiac disease that occurs frequently in young athletes[14] and is a leading cause of SCD in young people[21]. This cardiac model is capable of capturing abnormal electrophysiological behavior in HCM patients, and can be used in conjunction with the organ-level 3-D simulation approach proposed by Arevalo et al. (2016). Their combined usage in a 3-D cardiac simulation can achieve more accurate simulations that reflect the true electrophysiological behavior of HCM patients' hearts.

Second, I attempted to personalize key cardiac model parameters to match observable electrophysiological data in clinical settings using machine learning, specifically neural networks. A cardiac single-cell model is a system of ordinary differential equations(ODEs) that describe ionic currents that flow across the cell membrane, solving for the action potential(voltage) that results from these currents. However, it is impossible to analytically calculate the ionic current parameters from the action potential because the system of ODEs is an underdetermined system of equations. This means that several different combinations of cardiac model parameters can produce the exact same action potential behavior, and it is impossible to analytically determine which combination is the correct one. In light of this difficulty, machine learning has been proposed as a way to estimate cardiac model parameters.

Sobie et al. (2010)[17] attempted to solve the underdetermined system of ODEs in a cardiac model through least-squares regression, and found that augmenting the action potential data with intracellular calcium concentration($[\mathrm{Ca}^+]_\mathrm{i}$) data could constrain the system of ODEs to make better predictions. Neumann and Mansi (2020)[15] used regression approaches and reinforced learning using Markov decision processes for model parameter estimation. More recently, Jeong and Lim (2021)[10] also found that by using neural networks, they could predict the types of ion channel parameters that were modified with high accuracy.

In this study, I first developed a computational cardiac model that can simulate abnormal electrophysiological behavior of HCM patients' cardiomyocytes that can lead to cardiac arrhythmias. Then, I created a neural network-based machine learning approach to predict cardiac model parameters with a novel data preprocessing and augmentation technique that combines different approaches developed by authors in cardiology and machine learning.

Chapter 2

Background

2.1 Hodgkin-Huxley Model

The first computational model of electrophysiological activities produced by a cell was developed by Hodgkin and Huxley (1952)[9], inspired by the giant squid axon. Their work focused on modeling action potentials produced by the squid axon, which are voltage spikes across the cell membrane of the axon. These action potentials affect the membrane voltages of neighboring axons, and given a high enough spike, these axons are also triggered to produce a similar voltage spike. This chain of action potentials propagates across the network of axons, causing an electrical current to flow through the axons and produce what we know as electrical brain activity.

The Hodgkin-Huxley model is at its core a dynamic system of ordinary differential equations(ODEs) that describe the total ionic membrane current that flows across the cell membrane, solving for the action potential(voltage) produced by the cell as a result of the ionic currents. The basic premise of solving for voltage given the current is founded upon the assumption that the electrophysiologal behavior of the cell can be modeled as a capacitative circuit, where the cell membrane is a capacitor(Equation 2.1). Because the total membrane current can be decomposed into individual ionic currents within the cell, this essentially becomes a system of ODEs where each ODE describes each ionic current within the cell. Note that there is a constant leak current across the membrane, denoted as I_{leak}.

Finally, an ionic current for ion X can be written in a Hodgkin-Huxley formulation. The formulation consists of the ionic current I_X, the membrane capacitance C_m, a channel conductance variable G_X, several "gating" variables m, n, and h between 0 and 1 that represent the "openness" and "closedness" of the channel, and the Nernst potential E_X which is the equilibrium membrane potential(voltage) at which no ionic flow across the membrane occurs. Here, V_m represents the membrane voltage that is being solved for in the system of ODEs and will continuously be updated as the system is solved for every timestep dt. Some examples of Hodgkin-Huxley ionic current formulations are shown in Equations 2.2 and 2.3.

$$I_{tot} = \sum_X I_X = I_{Na} + I_K + I_{leak} = C_m \frac{dV_m}{dt} \tag{2.1}$$

$$I_{Na} = G_{Na} \cdot m^3 \cdot h \cdot (V_m - E_{Na}) \tag{2.2}$$

$$I_K = G_K \cdot n^4 \cdot (V_m - E_K) \tag{2.3}$$

2.2 Computational Cardiac Modeling

Cardiac cells, or cardiomyocytes, also produce electric signals via action potential propagation. The heart's electric activity regulates the physical movements of the heart such that it can beat smoothly and regularly. Computational cardiac models seek to simulate this activity through application of the Hodgkin-Huxley model, and many famous models in the field are based upon the Hodgkin-Huxley formulations of the ionic currents.

There is a large number of ionic currents involved in cardiac activity, and as such the system of ionic current ODEs can become quite complex to solve compared to the basic system introduced above. However, Equations 2.1 through 2.3 still serve as a base to understand ionic equations introduced in the Methods section of this study.

Chapter 3

Methods

3.1 Data Acquisition

3.1.1 Cardiac Model Formulation

The second version of the computational single-cell model developed by Ten-Tusscher et al. (2006)[19] was used as the base model to simulate the electrophysiological activity of cardiomyocytes and generate training/test data. To replicate the behavior of late sodium currents(I_{NaL}) that play a large role in cardiac diseases such as hypertrophic cardiomyopathy(HCM), the ionic equation describing I_{NaL} was manually added into the TT2 model using a Hodgkin-Huxley current formulation described by Coppini et al. (2012)[6]. Equations 3.1 to 3.3 describe their approach.

$$I_{NaL} = G_{NaL} \cdot m_L^3 \cdot h_L \cdot (V_m - E_{Na}) \tag{3.1}$$

$$h_L = \frac{1}{1 + \exp\left[\frac{V_m + 91}{6.1}\right]} \tag{3.2}$$

$$I_{tot} = (I_{Na} + I_{NaL}) + I_{Kr} + I_{Ks} + \ldots = C_m \frac{dV_m}{dt} \tag{3.3}$$

Equation 3.1 shows the Hodgkin-Huxley formulation of the late sodium current, and Equation 3.2 describes a separate equation for the gating variable h_L. Then, Equation 3.3 incorporates I_{NaL} into the system of ODEs that describe the Hodgkin-Huxley model and solve for V_m, the action potential voltage. In Equation 3.3, many other ionic equations besides

I_{NaL} are present, but omitted for clarity. $G_{NaL} = 0.085$ mS/μF is the conductance of the ion channel associated with I_{NaL}, and m_L is identical to the gating variable of m used to calculate the rapid sodium current I_{Na} introduced in the Background section. The gating variables m and m_L are values between 0 and 1. E_{Na} is the sodium Nernst potential(given constant), and $C_m = 0.185$ μF/cm^2 is the membrane capacitance.

A subset of parameters that control ionic currents in the TT2 model are presented in Table 3.1. To offset the model imbalance caused by the addition of I_{NaL}, the TT2 model parameters G_{Kr} and G_{GKs}, and the additional parameter G_{NaL} used by Coppini et al. (2012), were accordingly adjusted to produce physiologically realistic action potentials and intracellular calcium transients. Their modifications are shown in Appendix A.

Then, to capture the level of physiological variability across different individuals, the parameters listed in Table 3.1 were varied by multiplier modifiers using a grid range. A multiplier grid of [$\times$0.1, $\times$0.5, $\times$1.0, $\times$1.5, $\times$2.0] was used for all parameters except G_{NaL}, where a multiplier grid of [$\times$0.1, $\times$0.5, $\times$1.0, $\ldots$, $\times$5.0] was used to capture a larger variability of G_{NaL} in HCM. A starting multiplier value of $\times$0.1 was chosen to avoid total parameter knockouts that could lead to wildly unphysiological behavior. This approach was utilized by Jeong and Lim (2021) to generate training data for their neural network, and provided a means to normalize the wildly differing scales of the parameters while providing a efficient framework to conduct different TT2 simulations.

Table 3.1: Modified Parameter Subset of TT2 Model

Parameter	**Definition**	**Value**
G_{NaL}	Late Na$^+$ Current Conductance	0.034 mS/μF
G_{Kr}	Rapid Rectifier K$^+$ Channel Conductance	0.19125 mS/μF
G_{Ks}	Slow Rectifier K$^+$ Channel Conductance	0.49 mS/μF
G_{to}	Transient Outward K$^+$ Channel Conductance	0.294 mS/μF
G_{K1}	Inward Rectifier K$^+$ Channel Conductance	5.405 mS/μF
G_{CaL}	L-Type Ca$^+$ Channel Conductance	3.98E-5 cm^3/μF·s
k_{NaCa}	Max Na-Ca Exchange Current	1000 μA/μF
V_{maxup}	Max Sarcoplasmic Reticulum Ca$^+$ Uptake Current	6.375E-3 mM/ms

3.1.2 Data Generation and Sampling

Parameter multipliers were sampled in a uniformly random method from their respective grids to form 20,000 unique multiplier combinations. Each multiplier combination was applied to the TT2 model to vary simulations across repeated runs. Each run was conducted under a simulation protocol, where the protocol first simulated 40 "beats" of a cardiomyocyte to run the dynamic ODE system to steady-state using the new parameter multipliers. Then, over an additional two beats, the action potential and intracellular calcium transient behavior of the model were recorded. The frequency of the beats was set to 1 Hz, outputting 2000 ms long timeseries data for both AP and $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data.

The simulations were conducted using the open-source cardiology simulation package openCARP [16], and a single-machine multi-processor system with 48 CPUs was utilized to accelerate data generation. After each simulation, the resulting action potential was analyzed using openCARP's built in AP detection function and any data not showing any APs were discarded. About 900 simulations were discarded to form a total dataset of $\sim 19,000$ simulations.

3.2 Machine Learning Model Design

3.2.1 Input and Output

Following suggestions made by Sobie et al. (2010) to add another factor of differentiation between timeseries AP data, the $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data from each simulation was combined with the AP data to produce a multivariate timeseries(MTS) data type as the input variable. The objective was to predict the parameter multiplier combinations that produced this data. The decision to predict the multipliers instead of the raw parameter values themselves ensured that the loss function would optimize equally for all parameters across a common scale and accurately quantify the overall model prediction performance.

Thus, the machine learning algorithm was formulated as a supervised vector regression model with MTS AP and $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data as the input, the predicted multiplier combination as the output, and the actual multiplier combination as the target vector.

3.2.2 Data Preprocessing and Augmentation

To standardize the AP and $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data, the "default" behaviors of AP and $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ were subtracted over the dataset according to the method prescribed by Jeong and Lim (2021). The "default" data was simply the unmodified TT2 simulation results. The "subtracted" timeseries data were then transformed into 398×398 RGB images using the Gramian Angular Field(GAF) encoding map proposed by Wang and Oates (2014)[20], such that the data could be trained on a convolutional neural network. After encoding, this produced two images(AP and $[\mathrm{Ca}^{+}]_{\mathrm{i}}$) for each simulation. In order to augment the AP data with $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data as suggested by Sobie et al. (2010), the AP image was stacked on top of the $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ image to create a single 796×398 rectangular image, using the method described by Yang et al. (2019)[22]. The resulting dataset consisted of 19,000 796×398 RGB images for the input, and target vectors of length 8 representing the multiplier combinations for each image. A sample approach demonstrating the transformation step is shown in Figure 3.1.

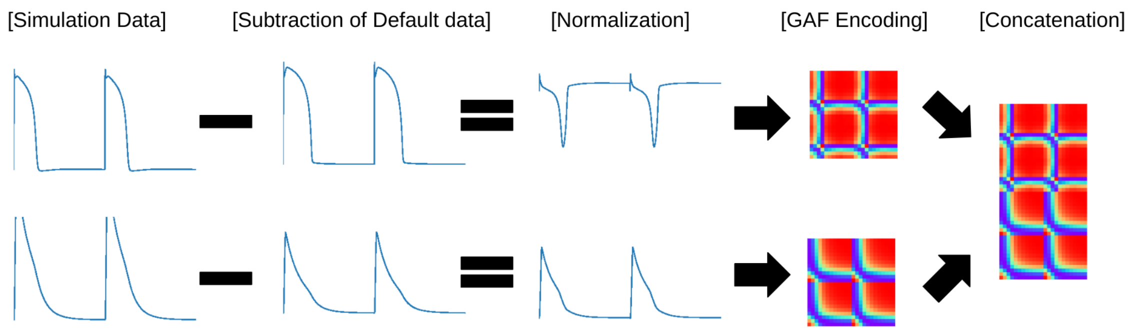

Figure 3.1: Data Preprocessing/Augmentation Step of a Simulation

3.2.3 Neural Network Formulation

A convolutional neural network was used to train with the image data. The architecture consisted of 6 convolutional layers with max pooling layers in between. Each convolutional layer had a padded stride length of 3 with varying filter sizes, and the ReLu activation function was used for every layer. For the max pooling layers, a pooling size of 2 was used. A flattening layer was placed between the last convolutional layer and the output layer to enable dense connections, and a rescaling layer was added after the input layer to normalize the image's RGB values between 0 and 1. For more details, a full diagram of the architecture can be found in Appendix B. The Python packages Tensorflow[1] and Keras[4] were used to implement the convolutional neural network.

3.2.4 Training and Evaluation

The train/test split ratio was set to 7:3 and resulted in 13376 images for training and 5733 images for testing. After randomly shuffling the data, the training and test images were segregated into separate directories to prevent data leakage. Basic hyperparameter tuning was performed with the entire training set by manually adjusting the number of neurons and layers, optimizer type, learning rate, batch size, activation function, convolution filter size, and number of training epochs. Given the manual nature of the hyperparameter tuning, a separate validation set was not constructed in this study. Based on the tuning results, the Adam optimizer and a learning rate of 0.004 was chosen, with a data batch size of 32. The neural network was trained for 200 epochs at the start, and epoch numbers were decreased for consecutive runs if model convergence was observed. To differentiate between random effects and true model performance as much as possible, the network was re-trained multiple times with different weight initializations and stochastic optimization.

The mean-squared error(MSE) metric was chosen as the loss function and monitored during training and evaluation. The evaluation step also tracked the model's predictions on individual values of a benchmark combination in order to observe the prediction error on a more granular level. The benchmark data, which are the simulation results for a TT2 simulation with default settings, are shown in Figure 3.2 with the preprocessing results. The prediction results for the benchmark data were used to reconstruct a TT2 simulation, and the simulation results were compared to the benchmark simulation to observe the differences, discussed later in the Results section.

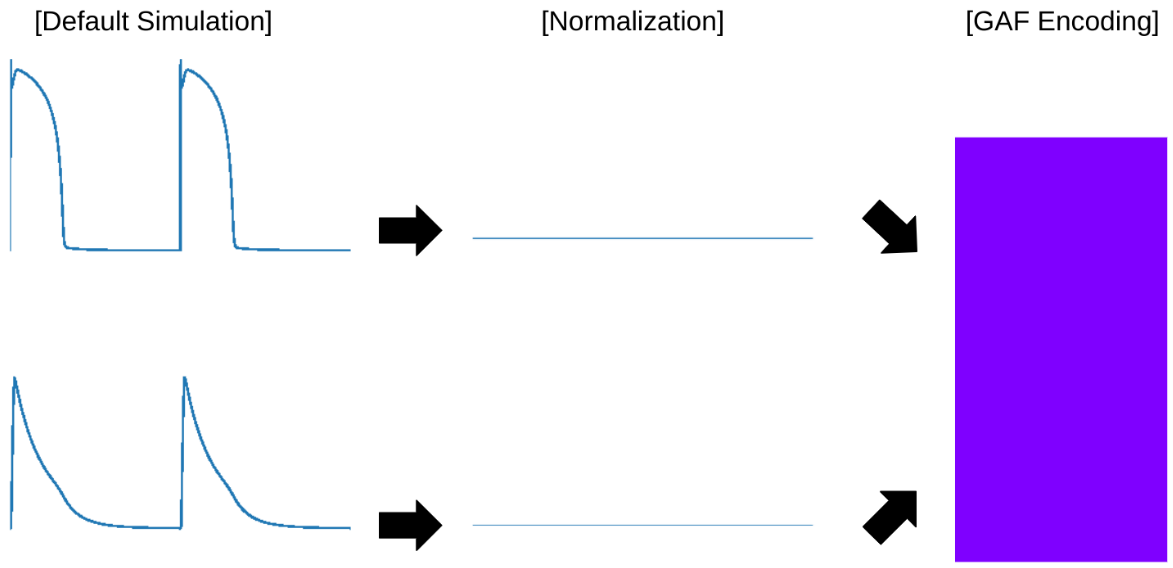

Figure 3.2: Benchmark Data Used for Evaluation

From start to finish, the machine learning workflow consisted of generating unique multiplier combinations for model parameters, conducting the corresponding TT2 simulations, preprocessing the simulation data, training the convolutional neural network, and finally the reconstruction of benchmark data using the predicted values at the end of training. This workflow is visualized in Figure 3.3.

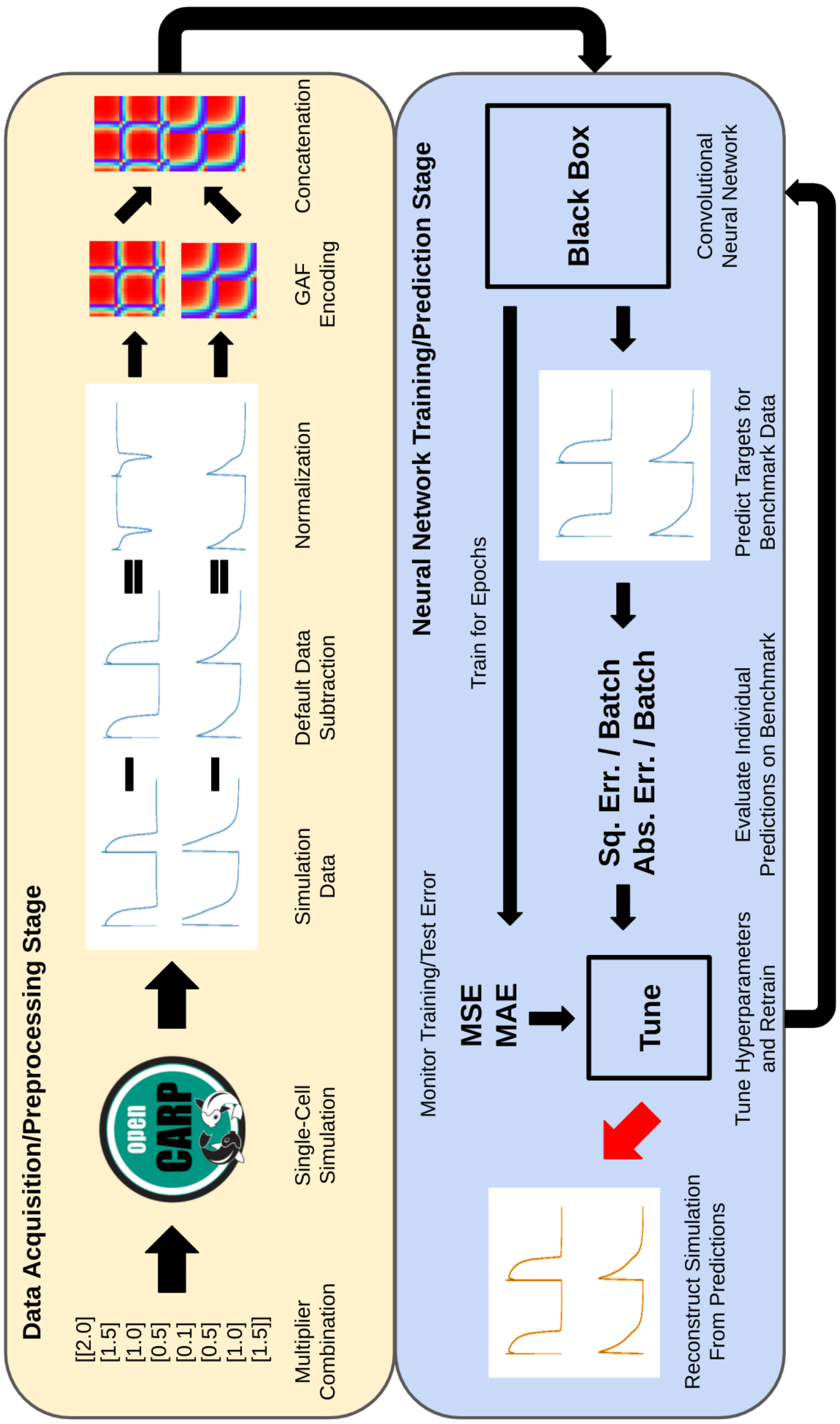
Data Acquisition/Preprocessing Stage
[[2.0] [1.5] [1.0] [0.5] [0.1] [0.5] [1.0] [1.5]]
Multiplier Combination
open CARP
Single-Cell Simulation
Simulation Data
Default Data Subtraction
Normalization
GAF Encoding
Concatenation
Neural Network Training/Prediction Stage
Black Box
Convolutional Neural Network
Predict Targets for Benchmark Data
Train for Epochs
Sq. Err. / Batch
Abs. Err. / Batch
Evaluate Individual Predictions on Benchmark
Monitor Training/Test Error
MSE
MAE
Tune
Tune Hyperparameters and Retrain
Reconstruct Simulation From Predictions

Figure 3.3: Machine Learning Workflow

Chapter 4

Results

4.1 Preliminary Results

Initial attempts to develop and improve the neural network was based on a small dataset of around 5,000 GAF-encoded images. These images originally included multipliers with a value of 0 in order to simulate extremely low levels of these parameters. It was later found that the 0 value multipliers caused the simulation to generate physiologically unrealistic data, negatively influencing the training process and causing the test MSE to remain high. Preliminary training results with this data showed a steady convergence to a training MSE around 0.4 and a test MSE around 2.0 after 100 epochs of training.

Since the MSE had limited interpretability outside the optimization loss, some sample images and corresponding multiplier combinations were chosen to roughly monitor the training prediction errors for each multiplier. This provided meaningful insights into the model performance by showing a breakdown of the underlying contributors to the test MSE. During the initial stage, the first image of the test set was picked as the benchmark data, and the test errors on each value of the corresponding multiplier combination were roughly monitored through screen outputs while training the model.

When observing the predictions of the model on the benchmark data throughout training, a subset of parameters(G_{NaL}, k_{NaCa}, V_{maxup}) were consistently unable to be predicted well and had significantly large prediction errors which impacted the overall test MSE. Moreover, the erroneous predictions steadily remained at their levels, signifying that the model had converged to a stationary point during optimization of the loss function and would not improve further.

Attempts to escape the stationary point using different optimizing functions and learning rates were unsuccessful. The learning rate was increased on Adam, RMSprop, and Nesterov momentum accelerated stochastic gradient descent(SGD) optimizers. Only Adam and Nesterov momentum SGD succeeded in converging with larger learning rates, while RMSprop exhibited exploding gradient behavior and ran into undefined number issues(NaNs). However, even with larger learning rates in Adam and Nesterov momentum SGD, and increased momentum in the Nesterov momentum SGD optimizer, the model consistently failed to escape the convergent region. The Adam optimizer showed slightly faster convergence than Nesterov moementum SGD, and was chosen as the default optimzer for the model.

As the next step, more convolutional layers and max pooling layers were added into the model to increase the complexity of the architecture. This was because the training error plateaued after several epochs and did not decrease over time as expected, which implied that the model was incapable of overfitting to the training data. While the increased complexity resulted in a faster model convergence, the test MSE still converged to similarly large levels close to 2.0, signifying that the model architecture was not the main cause.

The large test MSE slightly decreased after increasing the dataset size to 19,000 images, averaging around 1.5 in multiple runs. At this point, the amount of GPU memory became a limiting factor and led to memory exhaustion if there were too many layers in the model or if the training batch size was too large.

4.2 Final Training and Test Results

A re-evaluation of the dataset found that the 0 multiplier values in the multiplier combinations, which were meant to simulate extremely low values of the corresponding parameters, caused unrealistic physiological behavior within the TT2 model and prevented the neural network from giving accurate predictions on physiologically realistic data. These 0 values were replaced with a multiplier value of 0.1 to simulate low values of these parameters while preventing total knockout. The dataset was regenerated using these new multiplier grids, with a size of around 19,000.

Using this new constrained dataset, the test MSE significantly decreased by approximately 50% from around 1.5 to around 0.75. With the network architecture described in the Methods section, the model converged within one epoch of training and did not deviate with additional epochs. Based on previous experience, the Adam optimizer and a learning rate of 0.004 was chosen, with a data batch size of 32. The plot of the training and test results are shown in Figure 4.1.

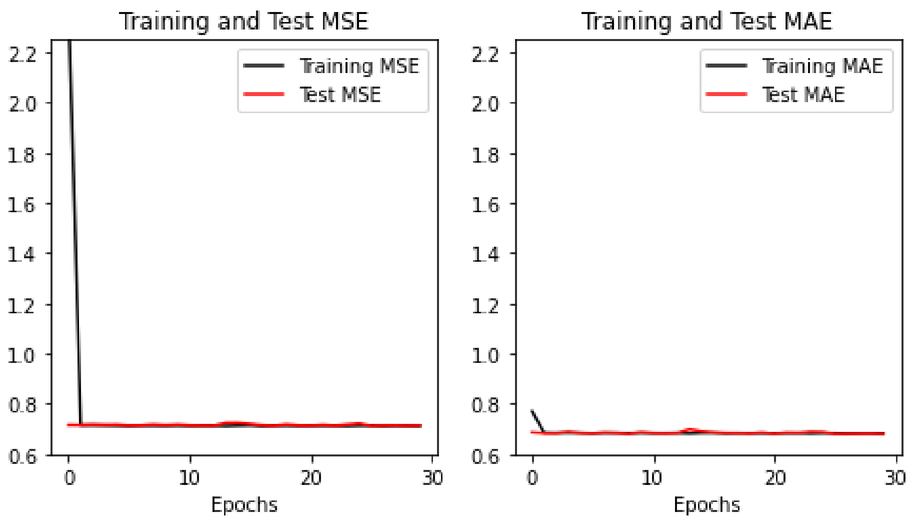

Figure 4.1: Training and Test Error Metrics

Since the dataset was totally regenerated, a new benchmark image needed to be picked from the test data to evaluate the model's performance on each parameter multiplier. Instead of the somewhat rough method of randomly selecting a sample from the test data, one piece of sample data was fixed for evaluation. The ”default” simulation data, a TT2 simulation with ”default”, unchanged values, was used for this purpose. The default data has previously been visualized in Figure 3.2. It was ensured that the ”default” benchmark data was not present in both the training and test sets to prevent any data leakage.

The prediction results at the end of the third epoch for the benchmakr data are listed in tabular format in Table 4.1. The prediction error progression for three epochs on the benchmark data is also shown in Figure 4.2. All parameter multipliers except that of G_{NaL}

were predicted with squared errors on the order of magnitude of 10^{-2} or smaller. The absolute errors were on the same order of magnitude excepting that of G_{NaL}. The G_{NaL} multiplier had square and absolute error magnitudes on the order of 1.

Table 4.1: Prediction Results of Benchmark Data at End of Last(Third) Epoch

Multipliers	**Target**	**Predicted**	**Sq. Err.**	**Abs. Err.**
$\mathbf{G_{NaL}}$	1.00	2.2511	1.5653	1.2511
$\mathbf{G_{Kr}}$	1.00	1.0141	1.9950E-04	0.0141
$\mathbf{G_{Ks}}$	1.00	1.0415	1.7240E-03	0.0415
$\mathbf{G_{to}}$	1.00	0.9169	6.8946E-03	0.0830
$\mathbf{G_{K1}}$	1.00	0.9080	8.4562E-03	0.0919
$\mathbf{G_{CaL}}$	1.00	0.9764	5.5403E-04	0.0235
$\mathbf{k_{NaCa}}$	1.00	0.8437	2.4422E-02	0.1562
$\mathbf{V_{maxup}}$	1.00	0.8997	1.0049E-02	0.1002

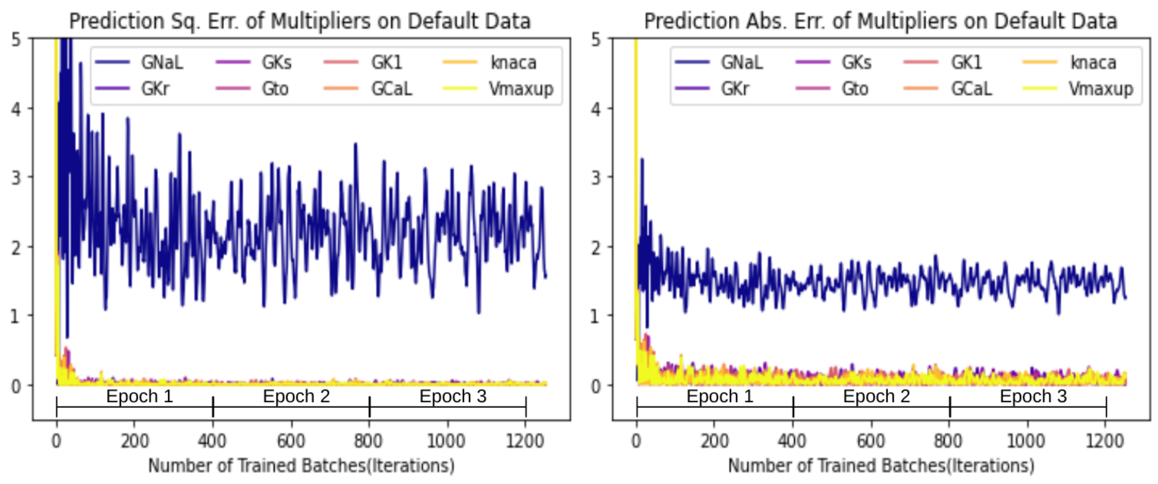

Figure 4.2: Squared and Absolute Error on Default Data Prediction

Reconstructing the benchmark TT2 simulation based on the predicted results shown in Table 4.1 is shown in Figure 4.3. Due to persisting prediction errors in the G_{NaL} multiplier, the reconstructed simulation does not fully match the default simulation.

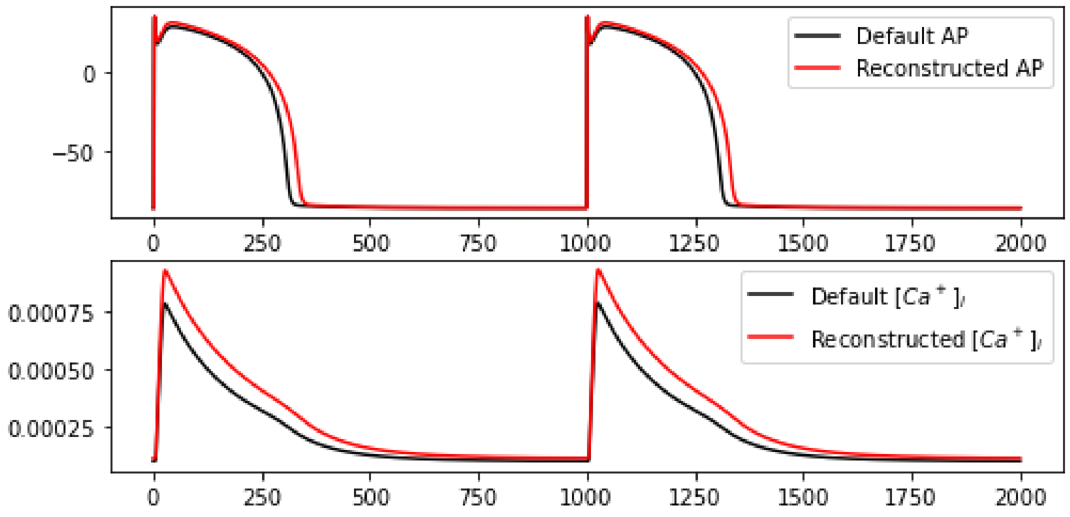

Figure 4.3: Default Data and Reconstructed Data From Model Predictions

Chapter 5

Discussion

The choice to use a neural network to predict ionic channel parameters was founded upon the work done by Jeong and Lim (2021), who used a basic artifical neural network consisting of three densely connected layers to predict the type of ion channel parameter that was modified, given only the AP data as the input. Their data was generated using the parameter multiplier combination approach, and was normalized with the ”subtraction” method of subtracting the default AP data from their experimental data for normalization.

While their approach was successful as a single-variable classification model, converting their classification model into a vector regression model to predict eight continuous values proved to be an entirely different problem. The main issue was that I was trying to solve an underdetermined system of ODEs, meaning that with only AP data as the input, I had no guarantee that my obtained predictions would be a unique solution to the system. To attempt to yield unique predictions, I augmented the AP data with $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data while training. The idea that $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data could sufficiently constrain the system of ODEs to yield a unique solution was suggested by Sobie et al. (2010).

However, this still left the problem of training a neural network with multivariate timeseries data. An initial brute-force approach to concatenate the raw $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data to the AP data for training did not perform well. To overcome this limitation, I borrowed the Gramian Angular Field image encoding method that could encode timeseries data into 2-D RGB images. This enabled me to train my data on a convolutional neural network, as well as easily augment the AP data with $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ data for training.

Re-generating the dataset after replacing the 0 multiplier values proved to be the key factor in improving the prediction performance of the neural network. During the development stage of the model, I discovered that the model architecture and hyperparameters did not influence the performance significantly. The quality of the dataset was the most important contributor to increasing the predictive performance, as is widely acknowledged by the data science community.

An interesting behavior in Figure 4.1 is that the training MSE is quite higher than the training MAE. While truncated on the plot, the training MSE starts around 2.4. One might expect that the MSE should reflect a squared value of the MAE, and find this behavior surprising. However, this is only true when dealing with a scalar value. When calculating MSE and MAE with vectors, the square and mean operations are conducted element-wise, meaning that the resulting MSE will no longer be an exact square of the MAE.

Another behavior that initially seems concerning is the relatively high training error compared to the test error. When training, one will normally observe initially high training and test errors which decrease as the training progresses - it is quite unusual to see a test error that is actually lower than the training error. On further investigation into this effect, I found that the model was actually fitting to the data so rapidly that the training errors converged within one epoch. However, because Tensorflow reports the average of the training errors for each epoch, the training errors look relatively high for Epoch 1, and thus leads to the abnormal-looking plots with higher training errors.

Given the rapid convergence of the model within one epoch shown in Figure 4.1, I felt that more insights could be gained by reporting the prediction performance by each batch within an epoch, instead of reporting by epochs. Figure 4.2 thus shows training progression across three epochs by batch numbers. There were approximately 400 batches in one epoch, which are also labeled in the figure.

One aspect that needs to be kept in mind when viewing Figure 4.2 is that these are prediction results for only a single piece of simulation data, which is the default simulation data. I made the choice to limit the analysis to a single datapoint because of the ability it provided to explore the data in-depth instead of being limited to averaged values. Given the relatively low levels of the test MSE, and that the G_{NaL} multiplier was consistently mispredicted across repeated training of the model with randomized weights and stochastic optimization, I judged that this analysis was representative of the model's performance as a whole.

The inability of the neural network to accurately predict the multiplier for G_{NaL} likely comes from the fact that I used a larger grid multiplier range($[\times 0.1, \times 0.5, \times 1.0, \ldots, \times 5.0]$) compared to the rest of the parameter multiplier grid ranges($[\times 0.1, \times 0.5, \times 1.0, \times 1.5, \times 2.0]$). Since I uniformly sampled from these grids simultaneously, the multiplier grid for G_{NaL} was not thoroughly explored as the grids for the remaining parameters. This means that the model did not have the opportunity to fully learn the behavior of altered G_NaL compared to the other parameters. This is evidenced by the large amount of continuous fluctuations in the error per batch of the G_{NaL} multiplier in Figure 4.2. All parameter multipliers except the G_{NaL} multiplier can be predicted with error levels close to 0 and show relatively little signs of fluctuation. However, while the prediction error is centered around 2.2 for the squared error per batch and 1.5 for the absolute error per batch, the G_{NaL} multiplier prediction continues to fluctuate across training.

Figure 4.3 shows that while the reconstructed data generated with 4.1 does not completely match the true data in magnitude. The mismatch aligns with my expectations of a larger G_{NaL} in the TT2 model, which prolongs the AP duration and also increases $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ levels. Even with the relatively large prediction error for the G_{NaL} multiplier, however, the overall effect in the reconstructed data is smaller than expected, and the morphology of the reconstructed data is well preserved compared to the true data.

Chapter 6

Conclusion

This study sought to establish an inter-disciplinary approach to apply precision medicine to cardiology by proposing a machine learning workflow to personalize single-cell cardiac models. I first developed a new cardiac model based on the work of Coppini et al. (2012) to capture abnormal electrophysiological behavior of HCM cardiomyocytes. Then, I utilized data generation and preprocessing approaches from Jeong and Lim (2021), the data augmentation method with $[\mathrm{Ca}^{+}]_{\mathrm{i}}$ suggested by Sobie et al. (2010), the Gramian Angular Field image encoder developed by Wang and Oates (2014) to transform my data into images, and finally the training methodology suggested by Yang et al. (2019) to simultaneously train a convolutional neural network with multiple input GAF images. Through combining the approaches outlined in these works, I was able to synthesize a novel data preprocessing method for a convolutional neural network and demonstrated that it could predict most parameters in a modified Ten-Tusscher cardiac model.

Immediate future work lies in effectively predicting the elusive G_{NaL} multiplier with a low error level. One immediate solution is to generate more datapoints to compensate for the lack of data exploration in the larger multiplier grid range. Another solution is to introduce constrained optimization into the model to ensure the neural network always predicts parameter multiplier values within a physiologically realistic range. Experimentally obtained parameter values found in literature could be used to establish lower and upper bounds during optimization. The constrained optimization approach seems especially promising, given the significant decrease of the test MSE after preventing 0 value multipliers from occurring within the dataset.

Even after the predictive power of the model is sufficiently increased, the model still needs to be trained with real-world data to have clinical impacts. This process will require navigating around noisy data and active prevention of overfitting to the noise. Various methods such as dropout and L1, L2 regularization can be applied to avoid overfitting. This is not to mention overcoming the difficulty of obtaining actual patient data in the first place. Currently, a cardiac tissue digesting pipeline is being developed to provide a steady flow of human cardiomyocytes for data recording, where a sophisticated microscope can provide simultaneous recordings of action potential and intracellular calcium transient levels for robust data acquisition. Much work remains before truly being able to deploy this machine learning framework in a clinically meaningful setting.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Selcuk Adabag, Russell V. Luepker, Véronique L. Roger, and Bernard J. Gersh. Sudden cardiac death: epidemiology and risk factors. *Nature reviews. Cardiology*, 7(4):216–225, Apr 2010.

[3] Hermenegild J. Arevalo, Fijoy Vadakkumpadan, Eliseo Guallar, Alexander Jebb, Peter Malamas, Katherine C. Wu, and Natalia A. Trayanova. Arrhythmia risk stratification of patients after myocardial infarction using personalized heart models. *Nature Communications*, 7(1):11437, May 2016.

[4] François Chollet et al. Keras, 2015.

[5] Sumeet S. Chugh, Kyndaron Reinier, Carmen Teodorescu, Audrey Evanado, Elizabeth Kehr, Mershed Al Samara, Ronald Mariani, Karen Gunson, and Jonathan Jui. Epidemiology of sudden cardiac death: clinical and research implications. *Progress in cardiovascular diseases*, 51(3):213–228, 2008.

[6] Raffaele Coppini, Cecilia Ferrantini, Lina Yao, Peidong Fan, Martina Del Lungo, Francesca Stillitano, Laura Sartiani, Benedetta Tosi, Silvia Suffredini, Chiara Tesi, Magdi Yacoub, Iacopo Olivotto, Luiz Belardinelli, Corrado Poggesi, Elisabetta Cerbai, and Alessandro Mugelli. Late sodium current inhibition reverses electromechanical dysfunction in human hypertrophic cardiomyopathy. *Circulation*, 127(5):575–584, 2013.

[7] Sophie Giffard-Roisin, Thomas Jackson, Lauren Fovargue, Jack Lee, Hervé Delingette, Reza Razavi, Nicholas Ayache, and Maxime Sermesant. Noninvasive personalization of a cardiac electrophysiology model from body surface potential mapping. *IEEE Transactions on Biomedical Engineering*, 64(9):2206–2218, 2017.

[8] Priya Hays. *Trends in Precision Oncology and Precision Medicine 2.0*, pages 419–480. Springer International Publishing, Cham, 2021.

[9] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[10] Da Un Jeong and Ki Moo Lim. Artificial neural network model for predicting changes in ion channel conductance based on cardiac action potential shapes generated via simulation. *Scientific Reports*, 11(1):7831, Apr 2021.

[11] Kevin B. Johnson, Wei-Qi Wei, Dilhan Weeraratne, Mark E. Frisse, Karl Misulis, Kyu Rhee, Juan Zhao, and Jane L. Snowdon. Precision medicine, ai, and the future of personalized health care. *Clinical and translational science*, 14(1):86–93, Jan 2021.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[13] J. Larry Jameson and Dan L. Longo. Precision medicine—personalized, problematic, and promising. *Obstetrical & Gynecological Survey*, 70(10), 2015.

[14] Ali J. Marian and Eugene Braunwald. Hypertrophic cardiomyopathy. *Circulation Research*, 121(7):749–770, 2017.

[15] Dominik Neumann and Tommaso Mansi. 5 - machine learning methods for robust parameter estimation. In Tommaso Mansi, Tiziano Passerini, and Dorin Comaniciu, editors, *Artificial Intelligence for Computational Modeling of the Heart*, pages 161–181. Academic Press, 2020.

[16] Gernot Plank, Axel Loewe, Aurel Neic, Christoph Augustin, Yung-Lin Huang, Matthias A.F. Gsell, Jorge Sanchez Elias Karabelas, Mark Nothstein, Anton J Prassl, Gunnar Seemann, and Edward J Vigmond. The openCARP simulation environment for cardiac electrophysiology. *Computer Methods and Programs in Biomedicine*, 208:106223, 2021.

[17] Amrita X. Sarkar and Eric A. Sobie. Regression analysis for constraining free parameters in electrophysiological models of cardiac cells. *PLoS computational biology*, 6(9):e1000914–e1000914, Sep 2010.

[18] Nicholas J. Schork. Artificial intelligence and personalized medicine. *Cancer Treatment and Research*, 178:265–283, 2019.

[19] K. H. W. J. ten Tusscher and A. V. Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology-Heart and Circulatory Physiology*, 291(3):H1088–H1100, 2006.

[20] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. 2014.

[21] Adaya Weissler-Snir, Katherine Allan, Kristopher Cunningham, Kim A. Connelly, Douglas S. Lee, Danna A. Spears, Harry Rakowski, and Paul Dorian. Hypertrophic cardiomyopathy&#x2013;related sudden cardiac death in young people in ontario. *Circulation*, 140(21):1706–1716, 2019.

[22] Chao-Lung Yang, Chen-Yi Yang, Zhi-Xuan Chen, and Nai-Wei Lo. Multivariate time series data transformation for convolutional neural network. In *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 188–192, 2019.

[23] Roy C. Ziegelstein. Personomics and precision medicine. *Transactions of the American Clinical and Climatological Association*, 128:160–168, 2017.

Appendices

Appendix A

Table A.1: TT2 Model Parameters and Modifications

Parameter	**Original**	**Changed**
G_{NaL}	0.085 mS/μF (added from Coppini et al. (2012))	0.034 mS/μF
G_{Kr}	0.153 mS/μF	0.19125 mS/μF
G_{Ks}	0.392 mS/μF	0.49 mS/μF

Appendix B

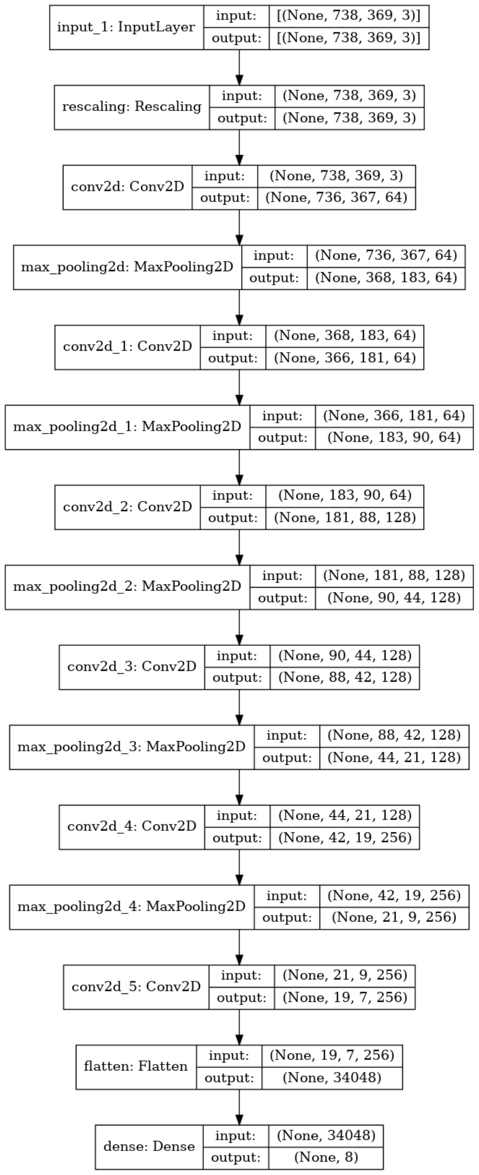
input_1: InputLayer | input: [(None, 738, 369, 3)] | output: [(None, 738, 369, 3)]
rescaling: Rescaling | input: (None, 738, 369, 3) | output: (None, 738, 369, 3)
conv2d: Conv2D | input: (None, 738, 369, 3) | output: (None, 736, 367, 64)
max_pooling2d: MaxPooling2D | input: (None, 736, 367, 64) | output: (None, 368, 183, 64)
conv2d_1: Conv2D | input: (None, 368, 183, 64) | output: (None, 366, 181, 64)
max_pooling2d_1: MaxPooling2D | input: (None, 366, 181, 64) | output: (None, 183, 90, 64)
conv2d_2: Conv2D | input: (None, 183, 90, 64) | output: (None, 181, 88, 128)
max_pooling2d_2: MaxPooling2D | input: (None, 181, 88, 128) | output: (None, 90, 44, 128)
conv2d_3: Conv2D | input: (None, 90, 44, 128) | output: (None, 88, 42, 128)
max_pooling2d_3: MaxPooling2D | input: (None, 88, 42, 128) | output: (None, 44, 21, 128)
conv2d_4: Conv2D | input: (None, 44, 21, 128) | output: (None, 42, 19, 256)
max_pooling2d_4: MaxPooling2D | input: (None, 42, 19, 256) | output: (None, 21, 9, 256)
conv2d_5: Conv2D | input: (None, 21, 9, 256) | output: (None, 19, 7, 256)
flatten: Flatten | input: (None, 19, 7, 256) | output: (None, 34048)
dense: Dense | input: (None, 34048) | output: (None, 8)

Figure B.1: Convolutional Neural Network Architecture