Evolving Block-Based Convolutional Neural Networks for Cardiac Arrhythmia Classification

Reuben Kouidri

Under the Supervision of *Henggui Zhang*

School of Physics and Astronomy

University of Manchester

May 2022

Abstract

We design a neural architecture search algorithm with a genetic algorithm search strategy to generate 2D-CNN networks for a multi-class ECG classification problem. Networks were evolved using 4 different crossover mechanisms - using both random chromosome, and random gene transfer, as well as both gene, and genome averaging. We found that genome-averaging yields networks with the highest fitness scores; different algorithm hyperparameters are also investigated. The top-3 generated networks were trained for 200 epochs on data from a 12 lead ECG. Classification accuracies exceeded the baseline model of last semester in all categories except atrial fibrillation, showing that a neural architecture search algorithm with an evolutionary search strategy can be used to generate competitive network architectures for a multi-class ECG classification problem.

I. INTRODUCTION

Cardiovascular disease (CVD) is the leading cause of mortality worldwide, accounting for 37% of deaths in Europe in 2017 alone [1]. The prevalence of CVDs is expected to rise in coming years due to an ageing, and growing global population, putting more strain on healthcare systems around the world [2]. There is, thus, a large interest in optimising healthcare systems to function more efficiently and effectively in the coming years.

Oxygenated blood is pumped around the body due to the contraction of the heart muscle. This contraction relies on the synchronised depolarisation and repolarisation of the cardiac myocardium [3]. Enhanced, or abnormal impulse formation, disruption in the conduction system, and even inflammatory diseases such as COVID-19 can lead to unphysiological, abnormal beating of the heart, called a *cardiac arrhythmia* (CA) [4, 5]. Although not all CAs are malignant [6], some, such as ventricular fibrillation [7], are more serious and can lead to further complications and even death [8]. Identifying CAs as early as possible is important for providing necessary treatment to prevent further deterioration of the patient and loss of life. An electrocardiogram (ECG) is the standard tool used to perform such a diagnostic. However, the procedure is time-consuming and the validity of its interpretation relies heavily on the experience and ability of the analyst performing the test [9], with incorrect diagnostics occurring frequently.

Artificial neural networks (ANNs) are computing systems inspired by biological nervous systems, such as connections in the brain [10, 11]. An ANN is a layered system of interconnected nodes, called neurons, which take inputs from neurons in previous layers, and output a value to neurons in following layers that depends on the total input as well as the specific activation function of the layer (c.f. sec II D). Convolutional neural networks (CNNs) are a subclass of ANN that were first introduced by Yann LeCun et al. in 1989 [12] for the classification of handwritten digits. The simplest type CNN is composed of alternating convolutional and sampling layers [13], which are used to extract features from images; an input image enters the network and is passed forward through each of the layers until the output layer, where the network attempts to classify the image into a set of predefined classes. A typical network contains millions of 'learnable' parameters, which are then updated via the back-propagation algorithm [14], allowing the network to 'learn' patterns in the data [15]. Two-dimensional CNNs have been the gold-standard in image recognition and computer vision tasks ever since the AlexNet [16] breakthrough in ILSVRC-2012, where a 2D-CNN was trained to classify 1.2million high-resolution images into 1000 different classes, beating the second-best submission – that used a traditional support-vector-machine ML approach – by more than 10% [17]. They have since proven to be incredibly successful across a plethora of domains, such as object detection [18], computer vision [19], natural language processing [20], and medicine where it has proved itself to be a valuable asset in radiology [21], covid detection [22], as well as detection of cardiac arrhythmias [23].

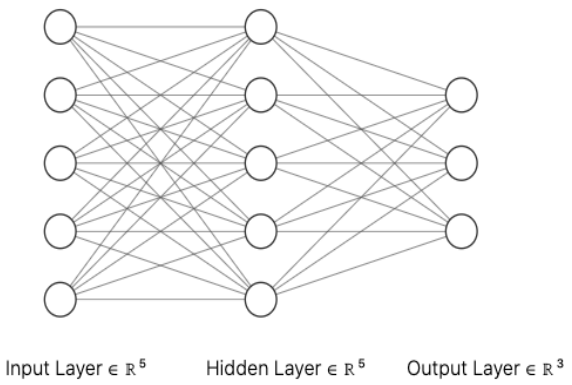

FIG. 1: Basic ANN architecture consisting of 3 dense layers

In this report we first introduce the basic components of 2D-CNNs and explain their significance. We then give a brief description of neural architecture search (NAS) algorithms before introducing the algorithm that we designed and explain both how it works and the ways in which it can be optimised via its own hyperparameters and multi-GPU acceleration methods. We then illustrate the results of the algorithm, and compare these with the baseline, high-performance, handcrafted model that was trained last semester. Finally, I conclude the work carried out and discuss the aspects that can be improved and expanded on in the future.

II. COMPONENTS OF A CONVOLUTIONAL NEURAL NETWORK

A. Convolutional Layers

At the heart of CNNs are convolutional layers. These contain a set of learnable kernels/filters that each get convolved with the input image during the

forward pass and updated by the backpropagation algorithm during the backward pass. During the forward pass, the matrix dot-product is taken between the kernel and the image pixels at every valid location creating a 2D activation map for that specific kernel.
Mathematically, convolution of an $n \times n$ input image with an $m \times m$ kernel is given by [24]

$$\tilde{T}_{i,j} = \sum_{p,q=1}^{m} k_{pq} T_{p+i-1,q+j-1}.$$

During the backwards pass, the kernel weights are updated for every kernel in the set in such a way as to increase activation outputs for certain image features. Kernels that amplify image features in a way that results in higher accuracy will be learnt by the network, whereas those which amplify insignificant features or noise will be diminished.

B. Activation Functions

After the activation map has been created by the convolutional layer, it is passed to a non-linear activation function. The activation function is used by the network to decide what information is useful and should be magnified, and what information should be suppressed to positively enhance the network's output [25]. The added non-linearity allows the network to learn more complicated patterns in the data that would usually not be possible in a simple linear regression model. Common choices of activation function are the sigmoid function, hyperbolic tangent and rectified linear unit (ReLU) and its variations such as PReLU (parametric), leaky-ReLU, and softplus [26]. PReLU is a generalisation to ReLU by including a small, learnable parameter a and is given by $f(x) = max(ax, x)$, where for ReLU $a = 0$. ReLU has the benefit that it does not activate all neurons at the same time and is hence computationally efficient in comparison with saturated activations such as $tanh$ [26]. However, one of the problems with ReLU is that if a neuron is not activated, the gradient of the activation function will always be zero and, hence, we cannot update its weights. Leaky-ReLU and PReLU were introduced to address this issue, and have been shown to consistently improve results [26, 27].

C. Pooling Layers

It is common to introduce pooling layers between the convolutional layers of a network, and after the activation and batch normalisation layers [28].

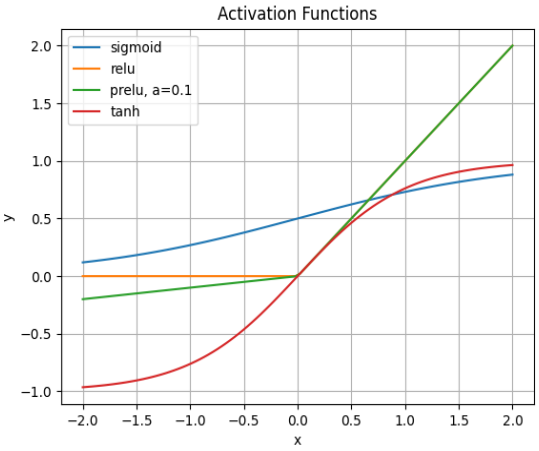

FIG. 2: commonly used activation functions

Pooling layers increase accuracy by shrinking the size of the feature map and inducing translational invariance [29]. Without pooling layers, learnt features would be associated with the specific location in the input image in which they occur. After successive pooling operations, the feature map is shrunk smaller and smaller in size whilst preserving the most important features. Hence, the image features are 'mapped' to a larger proportion of the image, increasingly independent of its original location.

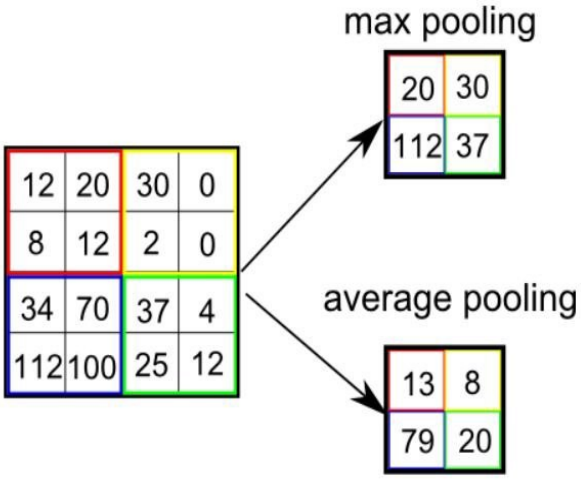

FIG. 3: max pooling operation. Image taken from [30]

D. Dense Layers

Dense layers, also called fully connected layers, have connections to all activation outputs in the

previous layer. Each connection is assigned a weight, and the total activation input of a neuron in a fully connected layer is the sum of all weighted activation outputs from the previous layers, plus any biases

$$x_i^l = b_i^l + \sum_{j=1}^{N_{l-1}} W_{ij}^{l-1} y_j^{l-1}$$

where W^{l-1} is the weight tensor of layer $l-1$, b_i^l is the i^{th} component of the bias vector of the l^{th} layer, and $y_i^l = \phi(x_k^l)$ is the activation output from the i^{th} neuron in the l^{th} layer.

E. Batch Normalisation

When training a CNN, the distribution of each layer's inputs changes due to updating the previous layer's parameters. This slows down training by forcing smaller learning rates and requiring carefully-initiated parameters. This phenomenon is defined as internal covariant shift [31]. Batch normalisation (BN) normalises the inputs to each layer in order to to control this shift, allowing for higher learning rates. BN is computed for each input image as follows: firstly, the batch mean and variance are computed, and each pixel value in the input image is normalised to

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}},$$

where ϵ - a small constant - is added to avoid possible division by zero. Next, a pair of learnable parameters γ and β are used to modify this expression, giving the output of the BN layer as

$$y = \gamma\hat{x} + \beta.$$

The addition of learnable parameters allow the network to optimise the normalisation of layers instead of using a fixed normalisation.

F. Dropout

The purpose of dropout layers is to introduce regularisation into the network [32] to reduce overfitting. They reduce overfitting by randomly dropping neurons during the network's training phase, forcing the network to learn features from an incomplete representation of the data and preventing reliance on specific pathways or activations. This can be illustrated in fig.1 by removing some of the solid lines connecting the neurons. During inference, to allow the network to give its max performance all neurons are activated [33].

G. Composite Modules

There are many ways to improve upon already-performing CNNs. Using the layers previously defined, and drawing from successful CNN architectures [16, 28, 34–37], one can create composite modules with higher complexity. For example, it is common to follow a dense layer with a ReLU activation function and dropout layer. Such a construction will be called a DenseBlock in this report. Similarly, it is common to follow a convolution operation by a ReLU activation function and batch normalisation [28, 38]. We defined this as a basic *convolutional block* (CB)

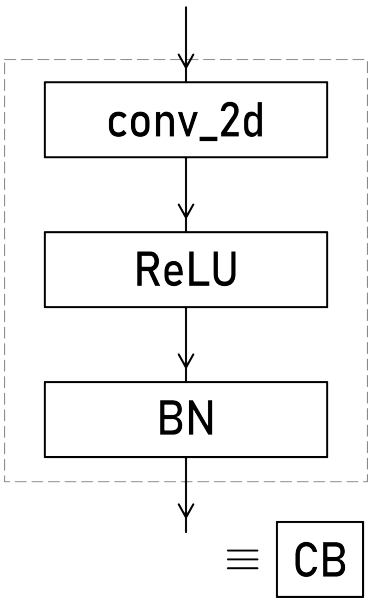

FIG. 4: conv block

Another method of improving networks is with the addition of attention modules (AMs) such as a channel attention module (CAM), spatial attention module (SAM), or convolutional block attention module (CBAM - CAM and SAM's marriage) and it's variants [34, 35]. Attention mechanisms are inspired by the ability to selectively focus the vision on the most relevant image features whilst simultaneously filtering out any irrelevant noise [34, 39]. AMs are light add-ons that do not have a large effect on model complexity, yet can potentially increase classification scores by a significant margin. Indeed, last semester it was noted that with the addition of a CAM module after each of the double-convolutional blocks (convolutional blocks with 2 convolutional layers before the ReLU activation and BN) a performance increase of 20% was achieved over the same network without CAMs. During the forward pass, the input tensor is copied,

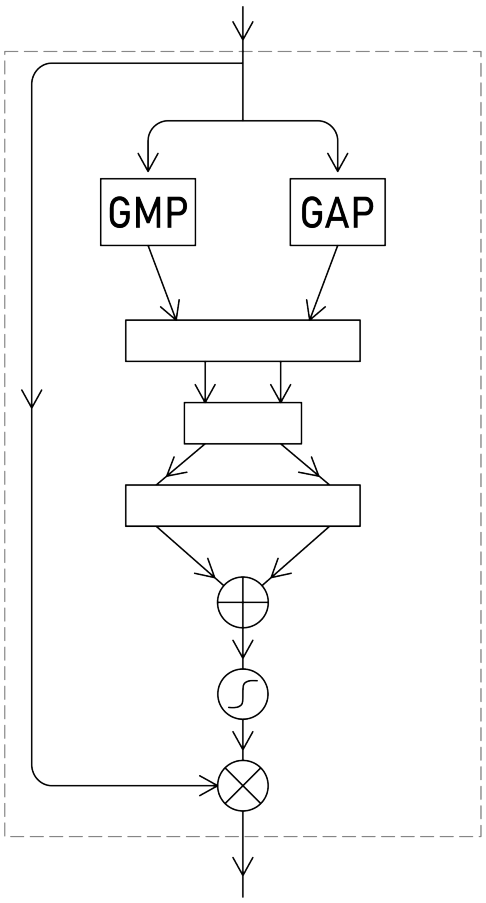

FIG. 5: Channel Attention Module/CAM: The input tensor is copied, with each copy passing separately through a global-max pooling operation (GMP) and a global-average pooling operation (GAP). They then passes through a shared multi-layer perceptron before being recombined in an element-wise addition, passing through a sigmoid activation, and finally recombining with the original input tensor by matrix multiplication.

with one copy passing through a global-max pooling operation (GMP), and the second copy passing through a global-average pooling operation (GAP). They then passes through a shared multi-layer perceptron before being recombined by element-wise addition, pass through a sigmoid activation, and finally recombine with the original input tensor by matrix multiplication.

We refer the reader to the first semester report section 2.5.1, and reference [34] for a more detailed discussion of the CBAM module.

Since more frequently than not, AMs modules are not used in isolation, but instead follow on from convolutions, it makes sense to combine these with the convolutional blocks, creating a modified resblock [28]. Indeed, this is how He et al. integrated their CBAM into the resblock of ResNet, achieving

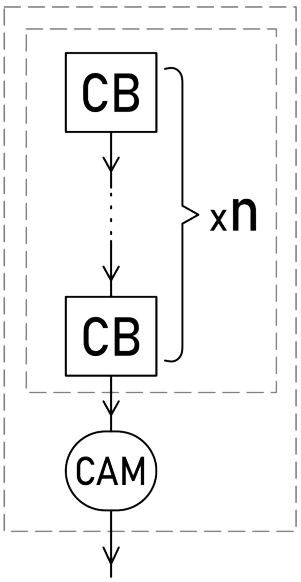

FIG. 6: The construction of a modified resblock, dubbed simply 'ResBlock' in this report. The type of ResBlock is determined by the number of convolutional blocks that precede the CAM module.

state-of-the-art image classification results [34].

III. NEURAL ARCHITECTURE SEARCH AND GENETIC ALGORITHMS

Every successful neural network relies on a well-designed architecture [40]. Yet, optimal network architecture for a given problem is unknown à-priori and infeasible to calculate [41]. As we have seen, 2D-CNNs can have many different components and non-trivial links between layers, including skip-connections and composite modules, that can be arranged in any imaginable way. Our understanding of the efficacy of different network architectures is limited, and relies heavily on trial and error combined with expert knowledge and experience [42]. This process is often labourous and time consuming, which has motivated the use of automated architecture search to discover them automatically [43].

The process of automating neural architecture design is called neural architecture search (NAS), and has 3 main attributes: a predefined search space, a search strategy, and a performance estimation strategy [44]. The search space is the set of all possible network architectures than can be reached, the search strategy provides a way to explore these possibilities, and the performance estimation provides a way to assess the performance of each network. One of the simplest search

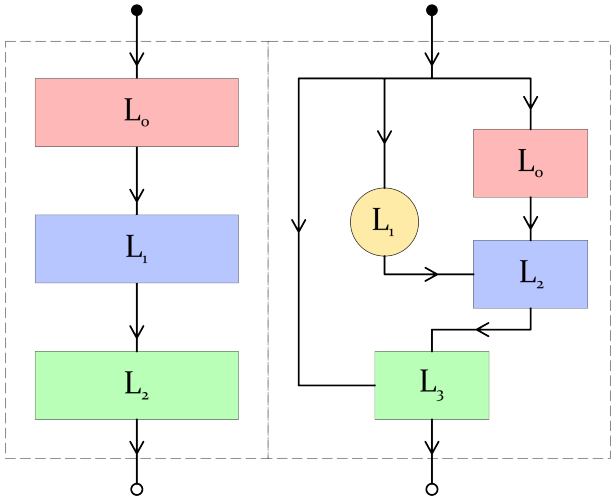

FIG. 7: A sequential vs. a more complex search space

spaces is the space of sequential neural networks, $\mathcal{S}$. Here, the networks are composed of n layers, $L_0, L_1, ..., L_{n-1}$ stacked one after the other, where the output of L_{i-1} is the input of L_i for $1 \leqslant i \leqslant n$. Indeed, any model in the search space can be expressed as a simple composition of layers: $\mathcal{S} \ni S = L_n \circ L_{n-1} \circ ... \circ L_0$. The search-space is then parameterised by [44]:

- the number of layers,
- the operation of each layer on it's input, ϕ_i, e.g. 2D-convolution, max-pooling, batch-normalisation
- the hyperparameters corresponding to each layer's operation for example, the number of convolutional kernels, the sizes of kernels and their strides, the number of neurons in dense layers, etc.

More complex search spaces incorporate design elements inspired by already-performing model architectures, such as using skip connections [45, 46], forming multi-branch networks [44] that vastly increase the size of the search space. Other search spaces use pre-defined blocks [47] which have several advantages, including a reduction in the number of degrees of freedom and size of the search space. This is of particular interest to those with limited computational resources, as it has the advantage of speeding up convergence. However, this also limits the chances of discovering novel architectures. Given that NAS algorithms are notoriously computationally expensive and often require thousands of GPU-hours [46], this may be the only feasible option for those without extensive resources.

A search strategy is a way of exploring the search space of a NAS algorithm. Popular search strategies have framed the search process as a reinforcement-learning problem [46, 48, 49], where the goal is to continuously improve a system – dubbed an 'agent' – in response to feedback from the 'environment' [24]. Feedback is given in terms of a reward signal which measures the performance of the model with respect to a predetermined criterion, and the system is updated with the goal of maximising the long-term reward [50, 51]. In NAS algorithms, the reward signal is usually the validation accuracy of the model on an unseen subset of data [46, 48, 52].
Another search strategy is the use of neuro-evolution/genetic algorithms to explore the search space [53] and have proved to be successful in numerous cases, even resulting in state-of-the-art networks [54, 55].
Genetic algorithms (GAs) are a group of population-based optimisation algorithms, based on Darwin's theory of survival of the fittest and the biological process of genetic evolution [56, 57]. The main components of a GA are the [58]:

- population initiation
- fitness function
- selection mechanism
- crossover operator
- mutation operator.

In the context of NAS, a population of networks is initiated by randomly sampling the search space, with hyperparameters taking values at random from an allowed set - the gene pool. The population is assessed for fitness, for example by training each individual and evaluating it's accuracy on a subset of unseen data. A selection pressure is determined by fitness; only a certain proportion of the fittest individuals survive and are allowed to crossover their genes. Different GAs can have different selection mechanisms, such as *tournament selection* [43, 55], and *roulette wheel* selection [59, 60]. In tournament selection, k individuals are randomly selected from the population. Then, the fittest individual is chosen with probability p, the second fittest with probability $p(1-p)$, the i^{th}-fittest with probability $p(1-p)^i$, etc. and the selected individuals are used in forming the next generation. Roulette wheel selection is another operator used to select individuals to form the next generation. Each individual is assigned a probability, p_i, for selection based on their fitness score

given by

$$p_i = \frac{f_i}{\sum_j^N f_j},$$

meaning that the fittest individuals have a higher probability of selection than those with lower fitness scores. Another method that can be used for selection is a cut-off, where only a certain, fixed proportion of the population, determined by fitness, survive, and parents are selected at random from this pool.

Besides selection methods, different GAs also deploy different crossover and mutation operators which decide on how the genetic information is passed from one generation to the next, forming arguably the most important part of any GA. Crossover operators act locally on a single individual by either altering the hyperparameters, adding and removing layers or skip connections, or any more complicated scenarios. Some of the most popular crossover operators are k-point operators: when $k = 1$, this operator selects at random a single location on the genome, or locus on the chromosome, and all genetic information beyond this point will be exchanged between the two parents [58]. When $k \geqslant 1$, multiple locations are selected and genetic information between the crossover points is swapped. For example, if loci 1 and 4 are selected on a chromosome, genes at loci 2 and 3 will be swapped. Another popular operator, and one of the operators used in this report, is k-point *uniform crossover*. This crossover method randomly selects k locations, and swaps the genetic information of the two parents. We investigate both with exchanging chromosomes, and individual genes.

The third part of any NAS algorithm is the performance estimation strategy. After the generation of an architecture by the search strategy, it needs to be evaluated for performance. This usually means training it and testing it's classification accuracy on a subset of unseen data. Therefore, a GA combines the selection strategy with the evaluation strategy in a single object.

In this research, we design a sequential, block-based search space with only 4 distinct block types. The search space is then simply parameterised by the number of blocks stacked in a row, the type of block, and each block's hyperparameters. We opted for this search space due to simplicity and limited GPU resources necessary to explore more unconstrained search spaces. Additionally, due to the previous success of using a similar architecture [24], we opted for a search space with the potential ability to converge within as few generations as possible. With this in mind, we aim to automatically generate high-performance network architectures that perform competitively in the classification of cardiac arrhythmias.

IV. THE GENETIC ALGORITHM

In the case of our algorithm, the search space is the set of possible network configurations with each unique configuration specified by a single species, the space is searched via crossover and mutation operators, and the performance estimation strategy is the evaluation of trained individuals on a subset of data.
Drawing inspiration from high-performance and state-of-the-art image classifiers such as Inception [36], ResNet [28], and it's modifications [34], and from recent competitive block-based NAS algorithms [48], we opt for a block-based search space with 3 different types of feature-extractor block, and a single type of dense block. The feature extractors consist of a single CAM integrated within a ResBlock as in section II G. The three types of block have either 1, 2, or 3 convolutional blocks stacked in a row before the integrated CAM module, and each dense block consists of a dense layer, followed by ReLU and a dropout layer with dropout factor 0.5. The hyperparameters of the feature extractors are then just the number and size of the convolutional filters in each conv block, the size of the max pool kernel, and the reduction ratio of the CAM module. The number of neurons in the input and output of the MLP within the CAM module is predetermined by the previous network features, and so is not a tuneable hyperparameter.

A. Initiation

In our algorithm, a species is entirely determined by its genome which we define as the ordered set of chromosomes with genes distinguished by location with NoneType values. An individual in the population is identified by its unique genotype which we define as the ordered set of chromosomes with genes distinguished by location with *specific* values. In code, to each species we associate a list, or 'ID', $[(r_{t_1}, ..., r_{t_n}), x]$ where r_{t_i} and x take integer values from 1 to 3. The first element of the ID (a tuple) corresponds to the set of ResBlock chromosomes of type, t_i. The type of RB chromosome specifies the number of Conv2D layers preceding the attention

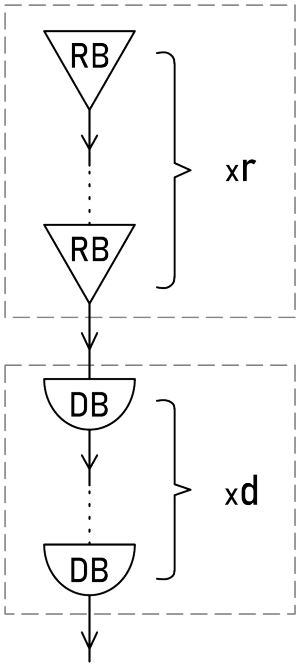

FIG. 8: Our search space composed of 'heavy' blocks allows for simple design and quick convergence, but limits freedom of exploration and discovery of novel architectures. 'RB' denotes a ResBlock as described in sec II G, and 'DB' denotes a DenseBlock

module. The second element of the species ID, x, denotes the number of dense layers that appear after feature extraction. For example, a species ID $[(2,1),2]$ corresponds to a genome [(RB2, RB1), D1, D2] where RB1 and RB2 contain 1 and 2 conv layer(s) before the attention module respectively, and $D1, D2$ encode dense layers. All individuals in this species have chromosomes of these types in these specific locations, but the genes on these chromosomes are initiated at random and allowed to evolve.

For a given species, a population of 20 individuals is initiated, each with a unique genotype. The values that the genes take are specific to the type of gene – meaning the type of layer that it encodes – and are selected at random from their corresponding gene pools. The gene pools are designed to give a large enough search space to explore many different network architectures, but small enough to reduce the time taken for convergence. Convolution and pooling operations both reduce the dimension size of the input image, so their max values are relatively small in comparison to the allowed number of kernels for instance. Maxpool operations have an even greater effect on dimensionality reduction since they use a stride equal to their size, hence the max value is smaller than that of the convolutional kernels. After the random generation of a genotype, it is validated by checking that the phenotype (model constructed from the genotype) will not reduce the dimensions of the input image, 128×128, to zero. If it does, the genotype is regenerated until a valid one is found. For multiple dense genes, an additional constraint is placed such that the number of neurons in each following layer is less than half the number of neurons in the preceding layer. This follows a rule of thumb that the number of neurons should decrease, not increase, towards the output layer.

Gene Pool	Min	Max
No. Conv2D filters	16	128
Conv2D kernel size	2	20
Neurons per FC layer	9	10000
Attention kernel size	2	8
MaxPool kernel size	2	12
R-ratio (attention)	2	10

TABLE I: The population gene pool

Each RB and DB chromosome encode its respective block. A RB chromosome has a sequence of genes corresponding to each of the convolutional, and maxpool kernel sizes, and one corresponding to the number of in planes, out planes, and reduction ratio. A DB chromosome contains only 2 genes, encoding the number of in features and out features of the layer, but with extra DNA acting as an on and off switch controlling dropout and ReLU layers inside the block. For example, an RB1-chromosome has 6 genes, whereas an RB2-chromosome has 9 genes. Each gene is identified by its locus on it's corresponding chromosome and is used in the algorithm to construct the models.

B. Evaluation

For each generation, the genotypes need to be expressed as networks (phenotypes) and evaluated to obtain a fitness score. Each genome in the population is passed to a function that takes each chromosome and expresses it as a custom subclass of 'torch.nn.Module', and appended to create a 'torch.nn.Sequential' module. The number of 'in

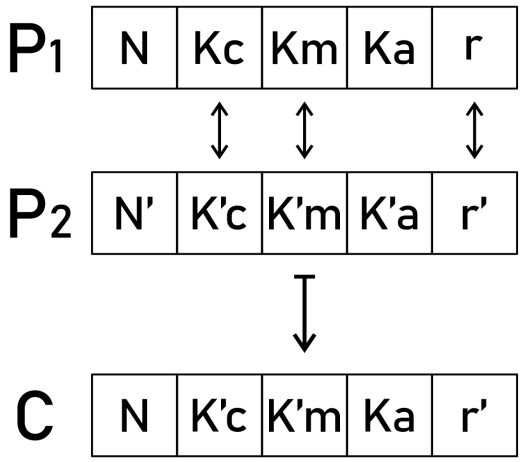

FIG. 9: RGT: genes on one of the parent chromosomes, P1, are selected at random and swapped with the same genes on the second parent's chromosome, P2, to produce a chromosome, C, inherited by the child. This is shown for an RB chromosome where N, K_c, K_m, K_a, r correspond to the number of filters, convolutional-, maxpool-, and attention-kernel sizes, and reduction ratio respectively

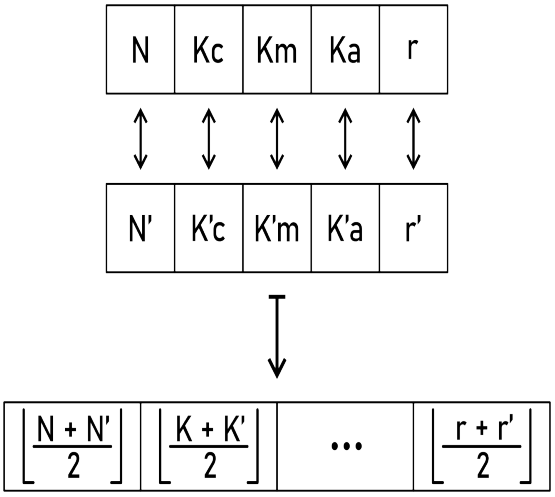

FIG. 10: MGT: the values of each gene are averaged producing a 'mean' chromosome inherited by the child

planes' of the first convolutional layer, and 'out features' of the last dense layer are set to 1 and 9 respectively, reflecting a single image entering the network and there being 9 classification classes. The genetic information is then synchronised in two different ways: inter-, and intra-chromosome synchronisation. Since the networks are sequential, the number of out planes in one RB must equal the number of in planes of the following RB, independent of the RB type. Similarly, the output features of each DB are identical to the input features of the following layer and are adjusted accordingly. This defined as inter-chromosome synchronisation. For RBs consisting of 2 or more CBs, the number of out planes of a given CB must equal the number of in planes of the following CB. This is defined as intra-chromosome synchronisation since it is the genes within the same chromosome that are synchronised. After the RBs, the output is flattened giving a 1D tensor of length n where

$$n = pd^2$$

for p filters in the last convolutional layer. The input to the first DB is thus set to n. Each DB is created with dropout parameter $p = 0.5$, except for the last layer which is always fully connected.

The loss function is effectively a distance function. It tells you how far off your predictions are from the true values. The accuracy achieved by a network tell you how many incorrect classifications the network made on the data. The loss and accuracy are not necessarily correlated. For example, a low loss and low accuracy means that lots of incorrect predictions were made, but the errors in each were small, i.e. they were 'close' to the true values, whereas a large loss and high accuracy would mean few incorrect predictions were made, but the errors in those were large. The fitness criterion was therefore chosen to be the percentage of correct predictions as this better reflects the desired classification accuracy.
Each network is trained using an 80/20 train-validation split of 500 ECG samples for 35 epochs. Preliminary runs with fewer training epochs than this, for example 20 epochs, did not work as SGD optimises the weights in a stochastic manor and results in variations in validation scores for identical networks. Fewer epochs correspond to larger variance, and if too high it is impossible to accurately distinguish between models of similar, yet different performances.
The fitness of the genome is then set equal to the classification score on the validation set.

C. Crossover

Once every genome in the population has been assigned a fitness score, the individuals are ranked and those fit enough for reproduction (set by the corresponding algorithm hyperparameter) are selected for crossover. The *elite* genomes are cloned once to prevent their genes exiting the pool, and a second time with added mutations.
Two genomes are then selected at random from this pool and their DNA are crossed over, producing a 'child' genome. We trial 4 different

crossover methods: genome averaging (GNA), random- and random-mean-chromosome-transfer (RCT and RMCT) as well as random-gene transfer (RGT). GNA simply calculates the average values of the genes for all loci on all chromosomes; RCT randomly selects chromosomes on one parent and swaps them with the corresponding chromosome of the second parent; RMCT randomly selects the chromosomes as above but instead calculates the mean value of the genes on the *selected* chromosomes, and RGT randomly selects chromosomes then, again, randomly selects genes on these chromosomes to swap between the parents. The crossover methods were designed to give an insight into how the 'coarseness' effects the evolution. For example, RCT corresponds to large changes in genetic material whereas RGT provides more fine-tuning.

During mutations, all genes are mutated except those encoding kernel sizes. Changes in kernel size have a large impact on the network and the dimensionality reduction of an input image as it undergoes forward propagation and so these mutations correspond to large phenotypic differences. However, if the population stagnates changes to these genes may be necessary to escape local optima. Since the mutations are random, the average change to a given gene tends to zero as the number of mutations increases. We thus chose to add in mutations to kernel genes if the population stagnated for more than 5 generations. The number 5 was chosen to give the population enough time to sufficiently explore local optima without wasting too much time.

After crossover, genes are mutated with probability given by the algorithm's configuration setting. This process is repeated until the entire population, except the elite individuals and their clones, have been replaced by their children. The process is repeated for the required number of generations.

D. Algorithm Hyperparameters

The algorithm is configured with a set of hyperparameters controlling the mutation rate and magnitudes of mutations, survival threshold, elitism, number of individuals in the population. These have a defining effect on the efficacy of the algorithm and need to be tested. For example, a survival threshold too high would lead to undesirable mixing of traits and lower fitness scores, whereas too low would lead to lack of diversity in the population and therefore would not explore the whole search space. Mutations allow for exploration of local neighborhoods around a given point in the search space and need to be the right magnitude. Mutations too large will not remain local, and those too small will have unnoticeable effect due to low sensitivity of networks to small changes in the majority of their components. Mutations were applied with probability 1 to the fittest models as we wanted to explore the search space around these solutions every time, and applied with a lower probability, to the rest of the population.

If a genome is selected for mutation, all of its genes are mutated by a small amount. The sizes of the mutations also needs to be set in advance and can be investigated if resources and time allow, although we did not have time to do this. As the crossover performs large changes to the genetic material the mutation sizes should be small with the intent of fine-tuning networks and providing a means to investigate a local neighborhood. Elitism refers to the number of individuals in the population that are protected during the reproduction stage. These models are exempt from crossover so their genes are not lost when evolving to the next generation. To increase the proportion of fit models in the population, I cloned the top-2 models and induced random mutations into their genes. The survival threshold is the proportion of individuals that are allowed to breed and cross over their genes with other individuals in the population. If this number is too close to 1, individuals with low fitness scores (undesirable traits) will be able to produce offspring, mostly with undesirable traits. As a rule of thumb, a low survival threshold is preferred, but not so low that the breeding population is small as this will quickly result in low genetic diversity, which is also undesirable.

In this version of the algorithm, each species is optimised in isolation from the others. This means that there is no inter-species competition or interference. This has the advantage that each species is given a fair chance of evolving - removing the possibility that, due to a small population size and unfortunate random initiation, a species with potentially-high performance traits could become extinct before having time to find these genotypes. However, it has the downfall of being slower to run since any species that are indeed innately unfit for classification of cardiac arrhythmias will continue to evolve instead of being out-competed by better-adapted species and becoming extinct.

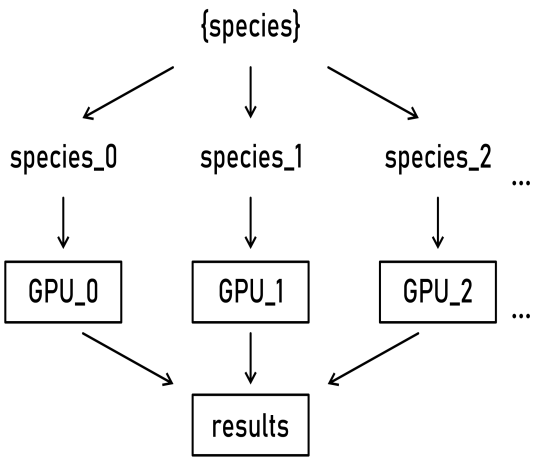

FIG. 11: GPU acceleration for multiple species

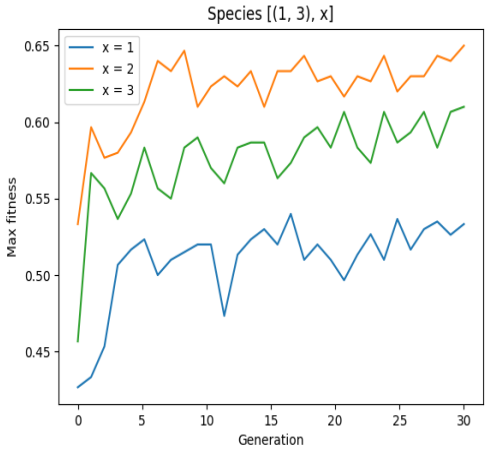

FIG. 12: Evolution of species $[(1, 3), x]$ for $x = 1, 2, 3$ showing clearly that 2 dense blocks after the feature extraction layers yields highest performance

E. Acceleration Methods

Training a NN involves 3 main steps: the forward pass, which allows calculation of the loss, the backward pass to compute gradients, and the optimiser step to update the network parameters [61]. This framework is naturally favoured by data parallelism, where an application creates several identical copies of a model and it's optimiser, each running on a subset of data and performing the forward and backward passes independently [62]. Before the optimiser step, gradients are synced across all models, making sure that all replica models are updated with the identical set of gradients and hence remain synchronised during all iterations [62]. Different acceleration methods and GPU combinations were tried: a single NVIDIA-A100 GPU was ran against 2 NVIDIA-V100 GPUs on a small test set of data. The 2 V100s took 85s for each generation to complete training, whereas the A100 took 55s. 8 CPU cores were used to load the data to and from a single V100 GPU, and 12 CPU cores were used for each A100 GPU. PyTorch's DataParallel (DP) and DistributedDataParallel (DDP) were both tried. DP allows single-process training across multiple threads whereas DDP supports multiprocess training where a separate process is created for each GPU. This proved to be significantly faster as it avoids overhead associated with python's global interpreter lock. NCCL backend was used for best-in-class performance on CUDA tensors [63].

The independent evolution of species in the algorithm is an *embarrassingly-parallel* problem, meaning perfect scaling on multiple GPUs.

V. RESULTS

In this section we present the results of our algorithm hyperparameter testing, species search, and network hyperparameter testing. We present the fittest genotype generated by the algorithm and construct it's phenotype, giving our interpretation of the results.

The 9 arrhythmia types in the ECG are: atrial fibrillation (AF), type 1 atrioventricular block (I-AVB), left- and right bundle branch block (LBBB, RBBB), premature atrial contraction (PAC), premature ventricular contraction (PVC), ST-segment depression STD, and ST-segment elevation (STE) [24]. More information on the arrhythmia types and the dataset can be found in the semester 1 report.

A. Species

The algorithm was run with 12 populations of different species, each allowed to evolve as described in IV B. The 12 species were split up into 5 sets, and split across the 4 available A100 GPUs and 2 V100 GPUs. The 2 V100s were used in parallel to evolve a single species instead of independently to provide more memory and make the speed comparable to the A100s. We found that species containing 2 dense blocks after feature extraction performed better than equivalent models with 1 or 3 dense blocks, independent of the network structure preceding them. Similarly, we found that species with only two RBs and a total number of 4 conv blocks outperformed all other network architectures. We

list the 3 most well-adapted species for classifying the 9 classes of cardiac arrhythmias on the validation set in table II alongside the fittest individual from that population, their corresponding genome, and fitness score.

Top-3 Species

Species Type	Genome	Fitness
[(1, 3), 2]	[RB1, RB3, D ,D]	0.53
[(3, 1), 2]	[RB3, RB1, D, D]	0.55
[(2, 2), 2]	[RB2, RB2, D, D]	0.55

TABLE II: The 3 most well-adapted species for multiclass ECG classification. All three species performed equivalently during preliminary runs.

B. Crossover Operators

When testing the crossover operators, we fixed the species [(2, 2), 2] as a reference and proceeded to explore the 4 different crossover methods outlined in section IV C. We let the populations evolve over 20 generations as before using a survival threshold of 0.25 and mutation rate 0. We found that the crossover method that yielded the fittest individual was GNA, and the crossover method that yielded the lowest fitness scores was RCT. RCT corresponds to the crudest crossover and failed to maintain the individual's traits.

C. Algorithm Hyperparameters

Next, we fixed the crossover method as GNA and investigated the effect of varying the mutation rate on the evolution of the population, taking values of 0, 0.25, and 1. The large differences were to be sure that the effects would be measurable. We used the same species as before, [(2, 2), 2], and let the population evolve over 20 generations. We found that a mutation rate of 0.25 achieved a higher max-fitness score after 20 generations than a mutation rate of 0 or 1. A mutation rate of 0 meant that local optima, if found, could not be explored meaning less fine-tuning of the networks. A mutation rate of 1 mutated the genes of all individuals, including the fittest. This had the effect of not preserving the fittest individuals in the population, sometimes having a positive impact on the phenotypes, but other times having the opposite effect.

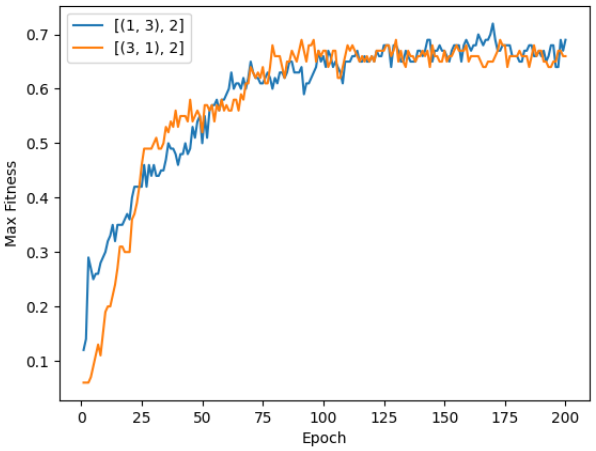

FIG. 13: Training species [(1, 3),2] and [(3, 1), 2] showing comparable validation scores

Hyperparameter	Value	Max Fitness
Mutation Rate	0	0.49
	0.25	0.53
	1	0.51
Survival Threshold	0.25	0.54
	0.5	0.49
	1	0.48

TABLE III: The GA hyperparameters and the maximum fitness scores achieved

Again, fixing the mutation rate to be 0.25, we varied the survival threshold, allowing it to take values of 0.25, 0.5, and 1. We found that the survival threshold yielding highest max-fitness was 0.25. This shows that if the survival threshold is too high, individuals with unsuitable genotypes make it into the selection pool for breeding and produce undesirable offspring for the classification task. However, reducing the survival threshold ensured that only the fittest survived to pass on their genes, ultimately resulting in a population that got fitter as it evolved. The difference in the max fitness scores using 0.5 and 0.1 were insignificant, and further investigation would be needed to resolve them.

D. The Model Architecture

Using the best algorithm hyperparameters from the tests above alongside the GNA crossover operator, we re-ran the algorithm for a further 20 generations

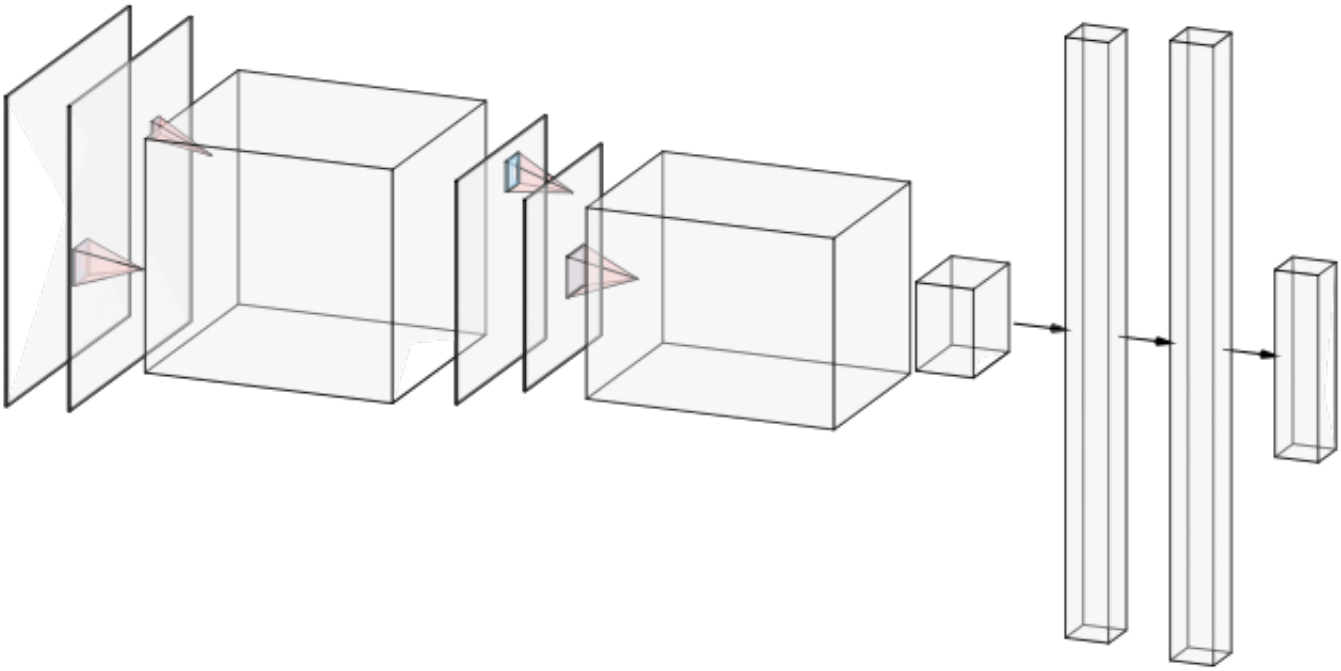

FIG. 14: The architecture of our network: 2 ResBlocks in sequence, each with 2 convolutional layers before the CAM module, depicted by the thick blocks. The output layers consist of two DBs before the final output of size 9.

for the three most well-adapted species: [(1, 3), 2], [(3, 1), 2], and [(2, 2), 2]. We then selected the fittest individual from each population, trained them on the full dataset, and tested on the same set of 300 ECG samples that was used in the previous study [24].

F1 scores were used to assess the network and draw comparisons with the baseline network, and are defined for the i^{th} arrhythmia class as

$$F_{1i} = \frac{2N_{ii}}{\sum_j (N_{ij} + N_{ji})}$$

where SR, AF, I-AVB, LBBB, RBBB, PAC, PVC, STD, and STE are represented by classes 1 to 9 respectively.
An overall $F1$ score is given by the average of all the $F1$ scores,

$$F_1 = \frac{1}{9} \sum_{i=1}^{9} F_{1i}.$$

$F1$ scores are given to the following super-types: blocks

$$F_{block} = \frac{2(N_{33} + N_{44} + N_{55})}{\sum_j (N_{3j} + N_{j3} + N_{4j} + N_{j4} + N_{5j} + N_{j5})},$$

Species	F1	Faf	Fblock	Fpc	Fst	Fitness
[(1, 3), 2]	0.752	0.791	0.895	0.759	0.646	0.69
[(3, 1), 2]	0.785	0.820	**0.902**	0.713	0.737	0.66
[(2, 2), 2]	**0.895**	0.930	0.897	0.800	**0.819**	0.66
reference model	0.778	**0.945**	0.893	0.851	0.747	N/A

TABLE IV: The scores of the fittest individual from each of the top-3 species against the reference model used last semester

ST-segment changes (ST)

$$F_{ST} = \frac{2(N_{88} + N_{99})}{\sum_j (N_{8j} + N_{j8} + N_{9j} + N_{j9})},$$

and premature contractions (PC)

$$F_{PC} = \frac{2(N_{66} + N_{77})}{\sum_j (N_{6j} + N_{j6} + N_{7j} + N_{j7})}.$$

Chromosome	Gene	Value
RB2	in planes 0	1
	out planes 0	93
	out planes 1	53
	conv kernel size 0	10
	conv kernel size 1	3
	maxpool kernel size	3
	reduction ratio	13
RB2	in planes 0	53
	out planes 0	111
	out planes 1	73
	conv kernel size 0	7
	conv kernel size 1	14
	maxpool kernel size	5
	reduction ratio	4
DB	in features	1168
	out features	1461
DB	in features	1461
	out features	9

TABLE V: The genotype of the overall-fittest individual that was found, belonging to species [(2, 2), 2].

VI. CONCLUSION

A NAS algorithm was developed from scratch using a sequential, block-based search space and a genetic algorithm search strategy. Four different crossover operators were tested and we found that GNA yielded networks that were most well-adapted for classifying cardiac arrhythmias. We investigated how the mutation rate and survival threshold of the genetic algorithm effected the evolution and fitness of the individuals, concluding that a lower survival threshold of 0.25, and moderate mutation rate also of 0.25 worked best, although further investigation would be needed to fine tune these values. We evolved 12 different species and retrained the top-3 for a further 20 generations using the optimal hyperparameter settings. The best single individual from each species was extracted and trained on the full dataset for a total of 200 epochs before being scored on the same test set of 300 ECG samples as last semester. The species that performed the best overall at classifying the 9 cardiac arrhythmias had ID [(2, 2) 2], corresponding to 2 type-2-ResBlocks followed by 2 dense blocks. However, we found that species [(3, 1), 2] scored the best on Fblock classification. These networks outperformed the network from last semester in all classification tasks other than the classification of atrial fibrillation, showing that a NAS algorithm combined with a genetic algorithm search strategy can successfully be used to generate competitive CNNs for the task of arrhythmia classification.

VII. DISCUSSION AND FUTURE WORK

Unfortunately, due to time limitations we were not able to explore a wider range of algorithm hyperparameters, only testing 3 mutation rates and survival thresholds. We were also limited by the number of individuals per population since 20 individuals took nearly 24 hours to evolve for 20 generations on a single A100 GPU. Ideally, we would liked to have used a larger population size and more generations, and to reach a plateau in fig.12 suggesting that the best architecture has been reached. Indeed, fig.12 suggests that 20 generations was not enough to find the best architecture, and that is left as a future task. Given that an architecture was found after so few generations is, however, a promising result, and demonstrates clearly the benefits of a block-based search space. Additional crossover methods were discussed but not implemented; working solo, there was not enough time to write the algorithm, as well as implement and test more crossover operators. In fact, an asexual reproduction mechanism, that uses only genetic mutations of the fittest individuals but of larger magnitude, was coded but there was not enough time to test thoroughly. This, again, would be a good pickup point for continuation. In the interest of execution time, some methods have been proposed to speed up NAS algorithms such as [47] using a block-search algorithm to find the best *single block* in terms of performance and then manually building a network by stacking multiple of these identical blocks together. This makes it much quicker to train and tune a model with far fewer parameters and results have shown that the performance of a single block transfers to the entire network. Additionally, parameter sharing between models with identical components would be suited to the type of search space developed in this work, and would massively boost the performance. Another direction of future work would be to remove restrictions on the search space and allow for more natural evolution, instead of being dictated by heavy ResBlocks. This would allow for the discovery of novel architectures with potentially even higher performance than those found in this work. Historically, genetic algorithms were used to update network parameters before back-propagation, which is far superior and is now used ubiquitously. This motivates an exciting possibility in future work of designing a back-propagation-based algorithm for updating a network's hyperparameters using a custom loss function and parameter sharing.

[1] "Causes of death statistics." https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Causes_of_death_statistics. Accessed: 2022-05-15.

[2] D. E. Bloom, E. Cafiero, E. Jané-Llopis, S. Abrahams-Gessel, L. R. Bloom, S. Fathima, A. B. Feigl, T. Gaziano, A. Hamandi, M. Mowafi, *et al.*, "The global economic burden of noncommunicable diseases," tech. rep., Program on the Global Demography of Aging, 2012.

[3] G. Tse, "Mechanisms of cardiac arrhythmias," *Journal of Arrhythmia*, vol. 32, no. 2, pp. 75–81, 2016.

[4] C. Antzelevitch and A. Burashnikov, "Overview of basic mechanisms of cardiac arrhythmia," *Cardiac electrophysiology clinics*, vol. 3, pp. 23–45, 03 2011.

[5] P. E. Lazzerini, F. Laghi-Pasini, M. Boutjdir, and P. L. Capecchi, "Inflammatory cytokines and cardiac arrhythmias: the lesson from COVID-19," *Nature Reviews Immunology*, pp. 1–3, 2022.

[6] M. Zehender, T. Meinertz, J. Keul, and H. Just, "Ecg variants and cardiac arrhythmias in athletes: Clinical relevance and prognostic importance," *American Heart Journal*, vol. 119, no. 6, pp. 1378–1391, 1990.

[7] H. C. Doval, D. R. Nul, H. O. Grancelli, S. D. Varini, S. Soifer, G. Corrado, S. Dubner, O. Scapin, and S. V. Perrone, "Nonsustained ventricular tachycardia in severe heart failure: independent marker of increased mortality due to sudden death," *Circulation*, vol. 94, no. 12, pp. 3198–3203, 1996.

[8] J. Kjekshus, "Arrhythmias and mortality in congestive heart failure," *The American Journal of Cardiology*, vol. 65, no. 19, pp. 42–48, 1990. A Symposium: The Renin-Angiotensin System-Tissue Specific Issues.

[9] D. A. Cook, S.-Y. Oh, and M. V. Pusic, "Accuracy of physicians' electrocardiogram interpretations," *JAMA Internal Medicine*, vol. 180, p. 1461, 11 2020.

[10] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[11] "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[13] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical Systems and Signal Processing*, vol. 151, 2021. https://doi.org/10.1016/j.ymssp.2020.107398.

[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 12 1989.

[15] M. Leong, D. Prasad, Y. T. Lee, and F. Lin, "Semi-cnn architecture for effective spatio-temporal learning in action recognition," *Applied Sciences*, vol. 10, p. 557, 01 2020.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[17] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[19] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis Lectures on Computer Vision*, vol. 8, no. 1, pp. 1–207, 2018.

[20] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[21] R. Yamashita, M. Nishio, R. Do, and et al., "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, 03 2018.

[22] L. Wang, Z. Q. Lin, and A. Wong, "COVID-Net: a tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images," *Scientific Reports*, vol. 10, 2020. https://www.nature.com/articles/s41598-020-76550-z.pdf.

[23] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Medicine*, vol. 25, no. 1, pp. 65–69, 2019. https://europepmc.org/articles/pmc6784839 pdf=render.

[24] Reuben Kouidri, "Multi-Class ECG Classification using Continuous Wavelet Transforms."

[25] X. Yang, "An overview of the attention mechanisms in computer vision," *Journal of Physics: Conference Series*, vol. 1693, p. 012173, 12 2020.

[26] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level per-

formance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[29] J. Yang, K. Yu, and T. Huang, "Supervised translation-invariant sparse coding," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3517–3524, IEEE, 2010.

[30] "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way." `https://towardsdatascience.com/a_comprehensive_guide_to_convolutional_neural_networks_the_eli5_way_3bd2b1164a53`. Accessed: 2022-05-15.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[32] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial intelligence review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[34] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[35] J. Wang, X. Qiao, C. Liu, X. Wang, Y. Liu, L. Yao, and H. Zhang, "Automated ecg classification using a non-local convolutional block attention module," *Computer Methods and Programs in Biomedicine*, vol. 203, p. 106006, 2021.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[37] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 169–175, IEEE, 2018.

[38] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[39] "Learning to combine foveal glimpses with a third-order boltzmann machine — proceedings of the 23rd international conference on neural information processing systems - volume 1, pages 1243-1251," 12 2010.

[40] F. Mattioli, D. Caetano, A. Cardoso, E. Naves, and E. Lamounier, "An experiment on the use of genetic algorithms for topology selection in deep learning," *Journal of Electrical and Computer Engineering*, vol. 2019, pp. 1–12, 01 2019.

[41] L. Mallory and A. Sevkli, "Optimizing neural network architecture through a coarse genetic algorithm," *J. Comput. Sci. Coll.*, vol. 36, p. 71–82, oct 2020.

[42] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[43] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, pp. 4780–4789, 2019.

[44] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[45] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[46] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[47] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[48] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

[49] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[50] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends in Cognitive Sciences*, vol. 23, no. 5, pp. 408–422, 2019.

[51] S. Mirjalili and V. Mirjalili, *PYTHON MACHINE LEARNING - THIRD EDITION : machine learning and deep learning with python, scikit... -learn, and tensorflow 2.* Packt Publishing Limited, 2019.

[52] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018.

[53] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 06 2002.

[54] I. S. Polonskaia, I. R. Aliev, and N. O. Nikitin, "Automated evolutionary design of cnn classifiers

for object recognition on satellite images," *Procedia Computer Science*, vol. 193, pp. 210–219, 2021.

[55] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[56] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary computation*, vol. 4, no. 1, pp. 1–32, 1996.

[57] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[58] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.

[59] F. Yu, X. Fu, H. Li, and G. Dong, "Improved roulette wheel selection-based genetic algorithm for tsp," in *2016 International Conference on Network and Information Systems for Computers (ICNISC)*, pp. 151–154, 2016.

[60] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.

[61] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for backpropagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, pp. 21–28, 1988.

[62] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[63] "WRITING DISTRIBUTED APPLICATIONS WITH PYTORCH." `https://pytorch.org/tutorials/intermediate/dist_tuto.html`. Accessed: 2022-05-15.

VIII. APPENDIX

Species Type	Genome	Fitness
[(1, 1), 1]	[RB1, RB1, D1]	0.31
[(1, 1), 2]	[RB1, RB1, D1, D2]	0.38
[(1, 1), 3]	[RB1, RB1, D1, D2, D3]	0.34
[(1, 2), 2]	[RB1, RB2, D1, D2]	0.46
[(1, 2), 3]	[RB1, RB2, D1, D2 ,D3]	0.39
[(2, 1), 2]	[RB2, RB1, D1, D2]	0.48
[(2, 1), 3]	[RB2, RB1, D1, D2 ,D3]	0.44
[(1, 3), 2]	[RB1, RB3, D1, D2]	0.53
[(1, 3), 3]	[RB1, RB3, D1, D2, D3]	0.47
[(3, 1), 2]	[RB3, RB1, D1, D2]	0.55
[(3, 1), 3]	[RB3, RB1, D1, D2, D3]	0.50
[(2, 2), 2]	[RB2, RB2, D1, D2]	0.55
[(2, 2), 3]	[RB1, RB3, D1, D2, D3]	0.51
[(2, 2, 2), 2]	[RB2, RB2, RB2, D1, D2]	0.49

TABLE VI: Species that were tested