university of groningen / faculty of science and engineering

Evolutionary Optimization of activation functions in deep heterogeneous networks

Bachelor's Project Thesis

Gregory Schweickert, s371058, g.a.schweickert@student.rug.nl,
Supervisor: Steven Abreu

Abstract: The effective use of deep neural networks often requires time consuming and expertise-reliant manual design decisions. Existing meta-learning methods have come a long way toward alleviating these requirements and enabling end-to-end learning. However, one important yet often overlooked facet is the selection of activation functions. Although methods exist to optimize this process, the vast majority are only concerned with homogeneous networks. That is, optimizing a single activation function for all hidden units in a network. Conversely, heterogeneous networks employ a variety of activation functions throughout. In this work, an evolutionary search method for discovering well-performing homogeneous and heterogeneous activation function setups is presented. Using random search for baseline comparison, experiments show that the developed directed search method is well-suited for the task. Indeed, hand-engineered deep CNNs tested on CIFAR-10 using ReLU and Swish are outperformed by those using discovered solutions. Furthermore, the explored heterogeneous setups result in better performance than their homogeneous counterparts. Lastly, novel solutions of both types are shown to generalize to CIFAR-100. However, transfer to an up-scaled architecture is relatively less successful. The presented methods offer a promising new approach to meta-learning in the space of deep heterogeneous neural networks.

1 Introduction

Over the last decade, Convolutional Neural Networks (CNNs) have become one of the most established learning models in the field of deep learning [Li et al., 2021, Abiodun et al., 2018, Yamashita et al., 2018]. This rise in popularity has come in response to impressive achievements in many areas of research - especially that of computer vision. Specifically, CNNs have demonstrated near human-level and sometimes superhuman performance in many vision tasks, for example in object recognition [He et al., 2015b, Taigman et al., 2014], image segmentation [Zeng et al., 2017], and pose estimation [Groos et al., 2022]. In fact, much of the success of NNs, in general, can be attributed to the increased availability of computation and data. However, a few developments in the field were key to making efficient use of these resources. Effectively, allowing complex (deep and wide) NNs to be trained effectively [Manessi and Rozza, 2018, He et al., 2015b]. These include, among others, advances in regularization techniques [Laurent et al., 2016], well-designed initialization schemes [Sutskever et al., 2013], extensive data augmentation [Howard, 2014, Krizhevsky et al., 2012], and innovative layer designs [Szegedy et al., 2015, He et al., 2015a].

Another breakthrough that contributed to the resurgence in interest of the scientific community in deep neural networks (DNNs) is the introduction of new activation functions (AFs) [He et al., 2015b, Apicella et al., 2021]. Indeed, the influence of activation functions on the performance of NNs has now been well established [Hagg et al., 2017]. Specifically, studies have shown that different choices of activation functions can result in significant differences in convergence speed and resulting network errors [Kamruzzaman and Aziz, 2002, Efe, 2008].

For clarity, following the definition of Goodfellow et al. [2016], AFs can be described as fixed non-

linear functions, g(z), through which the weighted linear combination of a neuron's input values, $z = W^\intercal x + b$ (where W is the weight matrix, b is the bias, and x is the input vector), is passed before being outputted for use by neurons in subsequent layers. The non-linearity property is a requirement to ensure the universal approximator property of the network, as shown by Cybenko [1989] and Hornik et al. [1989]. Initially, bounded continuous function, or so-called squashing functions, such as the logistic sigmoid function, $g(z) = \sigma(z)$, and the hyperbolic tangent function, $g(z) = \tanh(z)$, were used to reach excellent results in shallow NNs [Goodfellow et al., 2016]. However, due to the saturating across most of their domain, they suffered from the vanishing gradient problem when training DNNs with gradient-based learning [Manessi and Rozza, 2018, Apicella et al., 2021]. That is, in some cases, the partial derivatives of weights with respect to the loss function used for weight updates would become vanishingly small [Bengio et al., 1994]. At worst, this could severely slow down or even halt NN training. As a response, an active area of research aimed at creating novel activation functions was born [Manessi and Rozza, 2018].

One of the first and most notable function to arise from this effort is the Rectified Linear Unit (ReLU) [Glorot et al., 2011],

$$\mathrm{ReLU}(z) = \max\{z, 0\}. \tag{1.1}$$

Due to the simplicity of this non-squashing AF and its effectiveness in training DNNs, it inspired many future variants. These were based on the same basic shape but alleviate some know disadvantages of the ReLU function through slight variations. For example, Leaky ReLU (LReLU). It introduced a non-zero gradient when $z < 0$ to alleviate the "dying" ReLU problem, where the output of a neuron is always zero when large negative biases are learned [Maas et al., 2013]. Or, Softplus,

$$\mathrm{softplus}(z) = \log(1 + \exp(z)), \tag{1.2}$$

which can be interpreted as a smoothed version of the ReLU function - it solves the problem of non-differentiability at $z = 0$ [Dugas et al., 2001]. Beyond these strictly unparameterized examples, a set of tunable activation functions also garnered interest. These permit users to alter the shape of activation functions using parameters. For instance, the exponential linear unit (ELU),

$$\mathrm{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha * (\exp(z) - 1) & \text{otherwise,} \end{cases} \tag{1.3}$$

an AF that keeps the identity for positive arguments but otherwise returns a bounded exponential value, is adjustable by the parameter α [Clevert et al., 2015]. Indeed, as with all of the approaches mentioned above, the onus is on the neural network designer to create, select, and set appropriate AFs for the learning task. Thereby, heavily relying on the expertise of the designers as well as widely used hyperparameter optimization strategies such as a grid search and manual search. [Bergstra and Bengio, 2012] However, manual design of this sort can be tedious, computationally expensive, and unapproachable for the inexperienced [Abreu, 2019]. As such, the stage is set for the automation of this process by incorporating it into the learning paradigm.

A survey on trainable AFs, by Apicella et al. [2021], reveals that the idea has been explored in various forms since the 1990s but that recently, with the resurgence of interest in DNNs, there has been an influx of studies on the topic. Broadly, the authors characterize efforts to this end in one of two ways, methods optimizing parameterized activation functions and those based on ensemble methods. The former refer to the automatic tuning of function shape parameters, whereas the latter refer to approaches that merge different functions. Parametric ReLU,

$$\mathrm{PReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha \times z & \text{otherwise,} \end{cases} \tag{1.4}$$

is a well-known example of a trainable parameterized activation function [He et al., 2015b]. Its learning parameter α, which sets the negative slope of the ReLU function, is learned jointly with the weights and biases during backpropagation. On one hand, these methods introduce a negligible number of tuning parameters, which is positive for minimizing training cost, but on the other, their expressiveness is limited. After all, even an AF with many parameters is substantially bounded to the basic function shape on which it is built. Conversely, ensemble methods combine a set of basis functions (which can themselves be parametrized) to create AFs that are far more expressive. Examples of these

vary, however many rely on linear combinations of one-to-one basis functions [Agostinelli et al., 2014, Sütfeld et al., 2020, Piazza et al., 1992], whilst others include many-to-one functions with set ways to combine the two [Ramachandran et al., 2017, Basirat and Roth, 2018].

Considering the optimization of AFs, it can be done directly during the learning phase [He et al., 2015b, Sütfeld et al., 2020]. However, for methods that are non-differentiable, meta-learning schemes can be employed. That is, an additional optimization loop is added on top of the existing training loop, making meta-learning a nested optimization task. As such, the space of activation functions can be searched through in the same manner as different network topologies [**?**] and hyperparameters [Bergstra et al., 2013] have been traditionally explored. Meta-learning can be achieved using different learning paradigms, including Bayesian optimization [Kandasamy et al., 2018], meta-heuristic searches [Basirat and Roth, 2021, Elsken et al., 2017], reinforcement learning (RL) [Ramachandran et al., 2017] and evolutionary algorithms (EA) [Stanley and Miikkulainen, 2002]. Lately, the latter two methods have been gaining a lot of momentum for impressive results in automatic NN architecture design [Galván and Mooney, 2021]. Indeed, EA methods implemented for this purpose are part of a broader sub-field called neuroevolution (NE). It concerns itself with levering evolutionary mechanisms for a variety of NN related tasks. These include, updating connection weights, architecture design, learning rules, input feature selection, and much more Galván and Mooney [2021]. Indeed, their applicability to many facets of the learning and meta-learning process makes them ideal candidates for enabling end-to-end learning. As a consequence of its versatility with respect to several aspects of NN design and overall promise, NE is a central theme in this work.

Lastly, there is one more important aspect of AFs that must be accounted for, namely their configuration in the NN architecture. In a *homogeneous* setup, every hidden neuron in a NN uses the same activation function. In contrast, a *heterogeneous* NN is one where the AFs used by different hidden neurons may vary. In fact, they need not be optimized per neuron, as per layer [Manessi and Rozza, 2018] and per channel [He et al., 2015b] examples also exist. Overall, the increased expressiveness of heterogeneous setups lends itself to performance gains without changing the topology of the network, thus allowing for the development of more compact networks [Hagg et al., 2017, Turner and Miller, 2014]. Indeed, smaller networks are more parsimonious as they require fewer data samples and are simpler to train [Orr, 1993]. An impressive example of a heterogeneous DNN is the one developed in the original PReLU paper by He et al. [2015b], where they optimized the AFs of a CNN architecture in a channel-wise manner. They were the first to surpass human-level performance on the ImageNet 2012 classification dataset.

The goal of this paper is to further investigate deep heterogeneous networks through neuroevolution. More precisely, an attempt is made to optimize AFs and their configuration in a DNN simultaneously through genetic programming. To the author's best knowledge, the work is unique in applying this approach to DNNs. As such, a novel methodology for finding well-performing deep heterogeneous networks is presented and investigated. Moreover, the transferability of the discovered candidate solutions, with respect to a larger network architecture and a different dataset, is also considered. Lastly, this exercise offers an opportunity for new qualitative insights into effective activation function shapes for heterogeneous network setups.

2 Related Work

Neuroevolution (NE) has historically been a popular approach for neural architecture search [Elsken et al., 2019, Wistuba et al., 2019]. Indeed, a large number of methods have been developed to automatically create suitable network topologies - these often simultaneously evolve connection weights. That said, the majority of NE implementations evolve homogeneous networks, where the AFs of the hidden neurons are all identical. Generally, the methods that do allow for the evolution of heterogeneous networks work in one of two ways.

The first method involves optimizing the selection of the AF of each neuron from a predetermined list of functions. These are usually made up of basic functions, for example, the logistic sigmoid, Gaussian, and hyperbolic tangent function. Typically, as part of a broader NE procedure, the AF used by each neuron is encoded in a genetic

sequence. Then in each generation, through defined genetic operations, e.g. selection, crossover, and mutation, ineffective heterogeneous configurations are eliminated and more competitive ones are generated. NE methods capable of creating heterogeneous networks in this manner include General Neural Networks (GNN) [Liu and Yao, 1996], Cartesian Genetic Programming of Artificial Neural Networks (CGPANN) [Turner and Miller, 2013], a modified Hierarchical Co-evolutionary Genetic Algorithm (HCGA_2) [Weingaertner et al., 2002], and derivations of Neuroevolution of Augmenting Topologies (NEAT) [Stanley et al., 2009, Hagg et al., 2017]. To elaborate on the latter, NEAT is a well-known algorithm for creating parsimonious networks using a constructive approach, whereby a minimal network is expanded slowly by systematically adding neurons and connections. For its overall success and use of "minimal initialization", it has inspired several extensions. One such derivative method, HA-NEAT [Hagg et al., 2017], extends the direct encoding scheme of NEAT to include AFs. As a result, heterogeneous networks are evolved where the AF of each neuron is set to either a step, ReLU, sigmoid, or Gaussian function. These show convergence speeds similar to the best homogeneous networks whilst being smaller and more sparsely connected.

Another method to evolve heterogeneous networks is to use parameterized AFs. As such, the learning task becomes optimizing tuning parameters associated with each individual neuron. Turner and Miller [2014] employed a version of this method to optimize the shape of variable Gaussian and logistic sigmoid functions. More precisely, a single scaling parameter per neuron was additionally encoded in the gene sequences defining the NNs. Note, their work explored two NE strategies, namely a conventional fixed-topology NE algorithm and an extended version of CGPANN. The resulting evolved networks yielded superior performance on several benchmark tasks compared to their homogeneous non-parameterized counterparts. Moreover, these experiments were part of a broader investigation into the effectiveness of evolved heterogeneous networks. They also implemented the above described method, where the selection of the AF from a predetermined list was optimized for each neuron. Similarly, they found results in favor of the heterogeneous solutions. To take it one step further, the authors combined the two approaches. That is, they allowed evolution to select both a parametrized AF from a predetermined set and the associated tuning parameter value for each individual neuron. However, the combination of the two methods was found to be no more beneficial, on average, than when they were used individually. Two possible reasons for this finding were identified, namely an increase in search space dimensionality without an increase in the density of good solutions and the need for more subtle evolutionary algorithm parameter tuning.

A more complex version of the combined methods is put forth by Augusteijn and Harrington [2004]. They allowed the AFs for the individual neurons themselves to be subject to evolution through genetic programming. Indeed, by applying standard genetic operators to function tree structures - a search space of highly expressive mathematical functions was explored. These trees were made up of a rich set of nodes representing either functions, variables, or constants. Despite only using a small network with few neurons, they reflect that their largely unrestricted search space likely explains their mixed results. That said, their work formed an interesting challenge to the status quo of the time - namely the ubiquitous use of sigmoidal functions for hidden units. More recently, Bingham et al. [2020] showed how complex AFs could be evolved for deep learning purposes. In a similar fashion, tree-based structures were used to represent the AFs. A set of unary and binary operators were employed as the nodes. However, in contrast to Augusteijn and Harrington [2004], the number of operators available and the expressiveness of the function trees was restricted significantly. In fact, they used a slightly modified search space from the well-known work of Ramachandran et al. [2017], where novel AFs were discovered by a combination of exhaustive search and reinforcement learning. Their effectiveness was evaluated on several datasets, tasks, and models. The overall best performing evaluated candidate, $g(z) = zsigmoid(z)$, is now popularly known as *Swish*. Crucially however, in both works, the search space was restricted to the discovery of AFs for homogeneous networks.

This paper builds on the research of Bingham et al. [2020], and by extension Ramachandran et al. [2017], to evolve heterogeneous networks. However, to cope with the explosion of the search

space, the AFs are not evolved per neuron as done by Augusteijn and Harrington [2004]. Instead, AFs are optimized with respect to predetermined blocks (i.e. sections) of a DNN. Hereby, the author hopes to demonstrate that it is possible to evolve well-performing deep heterogeneous networks with reasonable computational resources. Furthermore, despite not co-evolving topology or connection weights, this idea can be made to fit within existing holistic NE schemes. Lastly, the model and dataset transferability of the evolved solutions is evaluated following the experimental design of Bingham et al. [2020].

3 Methods

3.1 Search Space

Each activation function is represented as a function tree, where the nodes are either unary operators or binary operators. That is, functions that take one variable or two variables, respectively. Note, as an exception to this description, a unary operator can also be a constant - either 0 or 1. The function trees are structured such that two unary operators always feed into one binary operator. The set of operators used was taken from Bingham et al. [2020], who modified it from the work of Ramachandran et al. [2017] primarily by removing operators/nodes with tuning parameters. Ensuing is the full list of unary and binary functions considered:

- **Unary:** 0, 1, x, x, $|x|$, x^2, x^3, $\sqrt{x}$, e^x, e^{-x^2}, $\log(1+e^x)$, $\log(|x+\epsilon|)$, $\sin(x)$, $\sinh(x)$, $\arcsinh(x)$, $\cos(x)$, $\cosh(x)$, $\tanh(x)$, $\arctanh(x)$, $\max(x,0)$, $\min(x,0)$, $\sigma(x)$, $\mathrm{erf}(x)$, $\mathrm{sinc}(x)$
- **Binary:** x_1+x_2, x_1-x_2, $x_1\times x_2$, $x_1/(x_2+\epsilon)$, $\max(x_1,x_2)$, $\min(x_1,x_2)$

Here, ϵ represents a small positive value to avoid undefined operations, such as division by 0. Further following their work, a "core unit" is defined as *core_unit(x) = binary(unary1(x), unary2(x))*. Note, a core unit takes in two identical scalar inputs, passes each one through a unary operator, and then combines their results using a binary operator that itself outputs a scalar. Hence, the aim of finding an activation function that transforms a single scalar input, x, into a single scalar output is achieved. Unlike previous work however, the depth of the AFs explored will be limited to 1. As such, structures of greater depth, for example, depth-2 AFs in the form of *core_unit(x) = binary(unary1(core_unit1(x)), unary2(core_unit2(x)))*, will not be considered. This design choice was made to restrict the size of the search space, since multiple of these function trees are considered in a heterogeneous set up.

Extending the work of Bingham et al. [2020], let F be the set of all possible function trees that can be created using the above-described nodes and rules. The search space for function trees (AFs) of depth $d \in \mathbb{Z}^+$, S_d, is described as follows,

$$S_s = \{f \in F |\ \mathrm{depth}(f) = d\}. \tag{3.1}$$

Consequently, an example of a (homogeneous) solution of depth-1 is *core_unit1(x)*. Next, in the heterogeneous search space, solutions consist of several core units in series. The degree of *heterogeneity*, h, describes how many individual AFs are represented by a solution. Then we can define a heterogeneous search space, S_d^h, as the h-fold Cartesian product of S_d. For example, the search space for heterogeneous solutions representing 3 AFs of depth-1, can be described as $S_1^3 = S_1 \times S_1 \times S_1$. Thereby, it can expressed in the general form: [*core_unit1(x); core_unit2(x); core_unit3(x)*]. Each core unit (element in the solution) represents the AF used by separate collections of hidden neurons in a NN. More precisely, a neural network is segmented into consecutive blocks (sections) of layers and the heterogeneous solutions are applied respectively. By applying AFs per block, instead of per neuron or per layer, the heterogeneous search space can be significantly reduced. Note, following this new notation, the homogeneous search space can be described as S_1^1. More details regarding the application of various AFs configurations to NNs is given at the end of Section 3.3.2.

The research presented here exclusively concerns itself with the search space of S_1^1 for homogeneous solutions and S_1^3 for heterogeneous solutions.

3.2 Genetic Algorithm

Genetic algorithms (GAs), a subclass of evolutionary algorithms, are a collection of population-based metaheuristics that can be used to solve stochastic

optimization problems. Characteristically, the solution domain is encoded through genetic representation. In this case, each candidate solution is defined as a sequence of genes describing the nodes of the underlying AFs trees (see Figure 3.1). These "individuals" compete and exchange information with one another to complete a given task. Effectively, principles from natural evolution are used to search through complex domains with many local optima. In fact, an advantage over gradient descent methods is that the landscape describing the solution space need not be differentiable [Yao, 1999]. Note, GAs also have a number of known disadvantages such as being slow (computationally expensive) and difficult to implement effectively [Leoshchenko et al., 2019]. Indeed, their performance is highly dependent on various tuning parameters, how the problem is encoded, and the design of the evolutionary operators.

The basic idea behind genetic algorithms is that a large set of candidate solutions, referred to as a population, is evolved over many generations to select for better solutions. Typically, a population is initially randomly generated. The candidate solutions are then evaluated based on a fitness metric, which determines their quality. According to their relative fitness value, a new population is generated from the previous one. More specifically, three different mechanisms are employed to create/select the candidate solutions for the next generation: (a) selection, (b) crossover, and (c) mutation. This process is repeated successively over many generations to converge towards optimal solutions.

The rest of this section will present the approach to evolving activation functions for heterogeneous and homogeneous networks alike. More precisely, it will introduce the implementation details of the genetic operators and overall algorithm used. Note, these have been slightly modified from Bingham et al. [2020] to accommodate for the different problem structure.

3.2.1 Selection

Selection is the predominant evolutionary procedure for exploiting good solutions. In this work, a predetermined number of the best-evaluated (fittest) candidate solutions get passed on to the next generation. They undergo no changes.

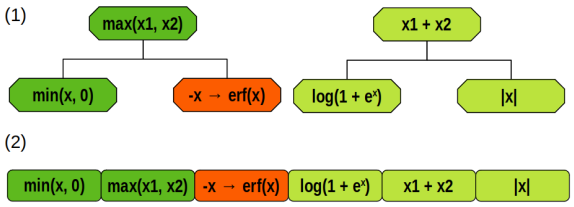


Figure 3.1: Diagram of a heterogeneous candidate solution containing two activation functions (from $S_{1,2}$), represented as (1) function trees and (2) an genetic sequence, undergoing a mutation operation. Specifically, a unary node in the first activation function is changed from $-x$ to $\mathrm{erf}(x)$

3.2.2 Mutation

Mutation drives the exploration of the solution space by randomly changing one gene in a chosen candidate solution. Note, all genes represent a node in one of the underlying AF tree structures. As such, after randomly selecting one gene/node, it is replaced by another node of the same type - binary or unary. A replacement is sampled using a uniform distribution over all the possibilities of matching type. An example of this operation is shown in Figure 3.1.

3.2.3 Crossover

In crossover, two selected parent candidate solutions pass on a proportion of their genes to a single offspring candidate solution. One-point crossover is used for this purpose. Specifically, a single cut-off point, deciding the proportion of genes from each parent, is determined by sampling a discrete uniform distribution. Crucially, the genetic sequence of the resulting offspring is always equal in length to that of the parents. An example of crossover is given in Figure 3.2.

3.2.4 Evolution

To begin the evolutionary process, a population of N random candidate solutions is created. All candidates within this initial pool are unique to force variety in the search. Next, a NN is configured for each individual according to its specifications and trained on a predefined dataset. More precisely,

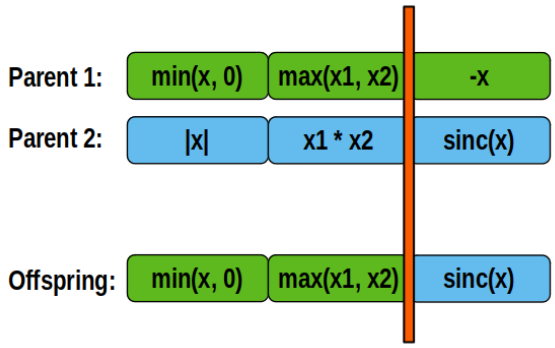


Figure 3.2: Diagram showing a crossover operation involving two homogeneous candidate solutions. The orange line represents the randomly selected crossover point. The offspring is the resulting candidate solution

the AFs represented by the candidate solutions are appropriately implemented in a fixed topology NN and used to learn a classification task's input/output relationship from a training dataset. After training, individuals are tested on a validation dataset and given an evaluation score. This metric can be either validation accuracy or negative validation loss. Once the effectiveness of all individuals has been evaluated, the above evolutionary operators are employed to create a new population for the next generation. First, the top b candidate solutions are selected, according to evaluation scores, to be directly added to the new population. Next, all candidate solutions, given index i, are assigned a fitness score, p_i, equal to the *softmax* of their evaluation score, L_i. As such,

$$p_i = \frac{e^{L_i}}{\sum_{j=1}^{N} e^{L_j}}. \tag{3.2}$$

The *softmax* operation is used to convert the evaluation scores into a probability distribution, such that all individual selection probabilities, p_i, add up to 1. Thus, candidate solutions can be selected with a probability proportional to their relative effectiveness by sampling from the distribution. When evolving the population, $(N - m - b)$ pairs of candidate solutions are chosen in this manner (with replacement) to act as parents for the crossover operation. Then, the $(N - m - b)$ resulting offspring undergo mutation as described above. Note, m is the number of randomly generated candidate solutions added to the new population to spur exploration. These are completely new candidate solutions that are generated by uniform sampling the solution space. Finally, the size of the new population matches that of the previous population. This process is repeated for several generations, hereby leading to the discovery of increasingly better performing AF configurations. The genetic algorithm is run for a set number of iterations (generations), after which the best-evaluated candidate solutions of the entire search are returned. Note, this evolutionary search procedure is identical whether the search space contains homogeneous or heterogeneous solutions.

3.3 Experiments and Setup

3.3.1 Datasets

The CIFAR-10 and CIFAR-100 image datasets are used in the experiments. Note, only CIFAR-10 is used for the searches. These are both balanced datasets of 60000 labelled 32×32 3-channel (color) images belonging to 10 and 100 classes, respectively. All classes are mutually exclusive. In both cases, 50000 image training sets and 10000 image test sets are provided [Krizhevsky et al., 2009]. No standard validation sets are given. However, for the searches, an additional validation set is necessary to prevent the reporting of overfitted results. Hereby, the training set is used exclusively for training the models, the validation set is used for computing fitness and selecting the best solutions, and the test set is used to demonstrate the true quality of the discovered candidate solutions. As such, a 5000 image validation set validation is created from the training dataset. The latter is therefore reduced to 45000 examples. Furthermore, a stratified split is used in order to balance the number of examples from each class. Since only the CIFAR-10 dataset undergoes this procedure, the validation set is formed by randomly taking 500 images per class from the training set. No structural changes are made to the test sets to allow for comparison with future work. Lastly, all images are normalized by dividing each pixel value by 255 and their corresponding labels are one-hot-encoded. No data augmentation procedures are employed.

3.3.2 Architectures

In order to restrict the search space of candidate solutions whilst not restricting the depth of the networks which can be optimized - it is necessary to move away from per neuron and per layer conceptions of heterogeneous networks. However, it is useful to utilize structural delimitations when applying several activation functions to a NN. As such, this work concerns itself with block-based architectures, inspired by the VGG networks of Simonyan and Zisserman [2014]. These deep convolutional NN, named after the research group who designed them, have exhibited impressive performance by relying on structural units (blocks) of repeating patterns of small kernel-sized convolutional layers [Khan et al., 2020]. Crucially, for use in this work, those network designs only served as inspiration and were not explicitly copied. Instead, manual designs were conceived. Additionally, supporting the design process with hyperparameter optimization schemes was avoided. The impetus is to minimize the degree to which the fixed NN architectures are optimized for any one function. For example, by avoiding network architectures that have been optimized with ReLU in mind. Hereby, the evaluation and comparison of the effectiveness of various AF configurations can made fairer.

Accordingly, a 2-block 6-layer convolutional NN (VGG-HE-2) and a 4-block 10-layer convolutional NN (VGG-HE-4) were implemented in TensorFlow, with the specifications detailed in Table 3.1 and Table A.1, respectively. The VGG-inspired blocks each contain two convolutional layers followed by max pooling and dropout. After every convolutional layer, batch normalization is performed. The last block in the architecture propagates the activations to a fully connected layer. After which, again, batch normalization and dropout are applied before finally reaching the output layer. The latter is composed of 10 nodes and uses *softmax* activation for the class prediction. Categorical cross-entropy is the loss function selected, as is common in multi-class classification tasks. Furthermore, the models are optimized using stochastic gradient descent with an unchanging learning rate of 0.001 and momentum 0.9. Note, all convolutional layers share the same kernel size of 3×3. Furthermore, the He uniform kernel weight initialization scheme is chosen as it offers more robust training for activation functions that involve non-linear properties [He et al., 2015b]. However, the number of kernels per convolutional layer increases with every block, starting from 32. That is, layers in the first block contain 32 filters, which is doubled to 64 filters in the following block, then again to 128 filters for the following block, and so on. The number of nodes in the first fully connected layer matches the number of filters in the final block. For example, 256 in the case of the first fully connected layer in VGG-HE-4. Similarly, the dropout rate starts at 20% and increases by 10% with each additional block. Namely, for VGG-HE-2, the dropout rate in the first block is set to 20%, 30% in the second, and the non-output fully connected layer is followed by 40% dropout. To speed up computationally expensive experiments, the tensorflow distributed mirrored strategy was implemented on two 11GB Nvidia Geforce RTX 2080 Ti GPUs. Furthermore, a large batch size of 256 per GPU unit was used. Note, we are only interested in relative and not absolute classification performance. Lastly, for any network parameter not explicitly detailed above, the default value provided by the *Keras* library was used (see Chollet et al. [2015]).

Concerning the activation functions used, in the homogeneous experiments, they are all identical except the one used by the output layer (which is always *softmax*). On the other hand, in the heterogeneous experiments, only convolutional layers in the same block share the same activation functions. The fully connected layer, before the output layer, is also given a unique activation function (see Table 3.1).

3.3.3 Searches

To compare homogeneous and heterogeneous networks, solutions of both kinds are implemented on a fixed topology network. Searches are employed to explore good candidate solutions in these domains. Regardless of which is considered, either S_1^1 for homogeneous solutions or S_1^3 for heterogeneous solutions, every candidate solution explored is implemented on the VGG-HE-2 architecture. The end of Section 3.3.2 describes how the candidate solutions are applied to the architecture. Using CIFAR-10, the resulting networks are trained for 50 epochs and then evaluated on the validation set. An early stoppage scheme is used to stop training in case of

VGG-HE-2		
Id	Layers	Properties
conv_b1_l1	2D convolution	$32 \times (3,3)$
	Activation	custom1
	Batch normalization	
conv_b1_l2	2D convolution	$32 \times (3,3)$
	Activation	custom1
	Batch normalization	
	2D Max pooling	$(2,2)$
	Dropout	20%
conv_b2_l3	2D convolution	$64 \times (3,3)$
	Activation	custom2
	Batch normalization	
conv_b2_l4	2D convolution	$64 \times (3,3)$
	Activation	custom2
	Batch normalization	
	2D Max pooling	$(2,2)$
	Dropout	30%
fc_l5	Fully connected	64
	Activation	custom3
	Batch normalization	
	2D Max pooling	$(2,2)$
	Dropout	40%
fc_l6	Fully connected	10
	Activation	softmax

Table 3.1: Description of the 2 block heterogeneous VGG-inspired architecture. The AFs are depicted in generalized format, namely custom1, custom2, and custom3. These refer to the AFs represented by individual candidate solutions in S_1^3.

NaN loss.

To conduct the searches, two strategies are employed, namely evolutionary search as described above and random search as a baseline. Unlike previous work, no exhaustive search was performed in the homogeneous solution space, despite being relatively small with 3,456 possible combinations. The motivation is to mitigate a significant computational exercise that cannot feasibly be managed in common practice. Instead, the genetic algorithm from Bingham et al. [2020] was altered to evolve solutions in this space. Note, the S_1^3 heterogeneous search space contains around 41 billion solutions. To compare the directed search and random search, both are performed in each domain. Every one of these evaluate 750 candidate solutions in total. In addition, a single 1500 candidate solution evolutionary search in the heterogeneous (S_1^3) domain is performed - it is used to gauge the outcome of longer searches. Following, the implementation details of the searches are given.

Random search As performed by Bingham et al. [2020], the random search is split up into "generations" to offer an illustrative baseline to the evolutionary search. Hereby, the 750 random solutions evaluated are split up into 15 groups of 50. Importantly, all evaluated candidates are unique as any duplicates are replaced.

Evolutionary search Under the evolutionary strategy, a population of 50 candidates is evolved ($N = 50$) for either 15 generations (in the case of the 750 candidate searches) or 30 generations (in the case of the 1500 candidate search). Concerning the underlying evolutionary mechanisms, the top 5 solutions of each generation get passed on to the following generation ($b = 5$). The remaining 35 ($= N - m - b$) candidates are generated by means of crossover and mutation as described above. These evolution parameter values, taken from Bingham et al. [2020], were chosen because preliminary trials showed positive results with respect to optimization performance. Furthermore, only loss-based fitness is employed as their findings suggest it has a clear advantage over accuracy-based fitness. A consequence of the latter being more lenient in terms of selection. Briefly, this is explained by the fact that loss, in comparison to accuracy, is not bounded between 0 and 1. Remember, accuracy is the per-

centage of correct classifications and loss indicates the results of categorical cross-entropy with respect to the entire validation set. Indeed, relative differences between the evaluation scores of candidate solutions are significantly larger when the loss measure is used. As such, when considering the *softmax* operation used to determine fitness (see Equation 3.2), a loss-based approach more sharply penalizes poor solutions than its accuracy-based counterpart. Note, the same settings and parameters were used for all search spaces considered.

3.3.4 Dataset and Architecture Transfer

The transferability of the well-performing candidate solutions from all the searches with respect to another dataset and architecture is also investigated. Specifically, the top candidates from the CIFAR-10 searches are retrained and evaluated on the CIFAR-100 test set. Note, the VGG-HE-2 architecture remains unchanged. On the other hand, in another experiment testing architecture transferability, these same top search candidates are evaluated on a bigger neural network, namely VGG-HE-4. This is a straightforward transfer exercise when the considered solutions are homogeneous. For heterogeneous solutions, however, AFs are transferred to pairs of blocks. For example, the AF used in the first block of the small VGG-HE-2 architecture is now transferred to the first and second blocks of the VGG-HE-4 architecture. In contrast, the AF of the penultimate fully connected layer is transferred to the equivalent layer in the bigger network. Table A.1 (see Appendix **??**) shows a generalized version of the AF configuration of the VGG-HE-4 architecture after having undergone the heterogeneous transfer exercise.

3.3.5 Testing Strategy

To gauge their true performance, the overall top three candidate solutions from each search undergo a training/testing cycle. Namely, they are trained from scratch for 200 epochs on CIFAR-10 (training and validation dataset) using VGG-HE-2 and are then evaluated on the corresponding unseen test set. Note, all the same learning and architecture parameter values as in the searches are used. Following previous work, this procedure is repeated five times and the median test accuracy is reported [Ramachandran et al., 2017].

The same testing procedure is used to evaluate the transferability of each top candidate solution yielded from the searches (performed exclusively using CIFAR-10 and VGG-HE-2). That is, the transfer methodology is tested using the different architecture and dataset combinations described above. Specifically, applying the training/testing cycles to measure the transfer performance to the different dataset, where the best search candidates are retrained on CIFAR-100 with VGG-HE-2. And to determine transfer performance to the larger NN architecture, where the top search candidate solutions are retrained on CIFAR-10 with VGG-HE-4.

Similar to Bingham et al. [2020], in order to evaluate the significance of the overall top search candidates' test results, they are each retested 20 more times. In addition, both benchmark activation functions undergo the same procedure. As such, mean test accuracies (and 95% confidence intervals) over 20 repeats of the 200 epoch training/testing cycle are computed. In turn, these are used to determine whether there is a significant difference between the mean test accuracies coming from the different AF configurations. Specifically, independent two-sample heteroscedastic (Welch's) t-tests are performed. Unequal variance is assumed because of large differences in the resulting confidence intervals.

Note, all accuracy measures are based on the outcomes of multiple training/testing cycles to mitigate the effects of weight initialization.

4 Results

The purpose of the experiments is threefold. First, to show that heterogeneous setups for deep NNs can outperform their homogeneous counterparts and baseline AFs such as ReLU and Swish. Secondly, with respect to minimizing computational effort, to showcase the benefit of using directed (evolutionary) search in this domain versus random search. Lastly, to investigate the dataset and architecture transferability of the top candidate solutions coming from the searches.

To begin, the outcomes of the evolutionary search method in the homogeneous (S_1^1) and heterogeneous (S_1^3) domains are considered. To evaluate the effectiveness of these directed searches, they

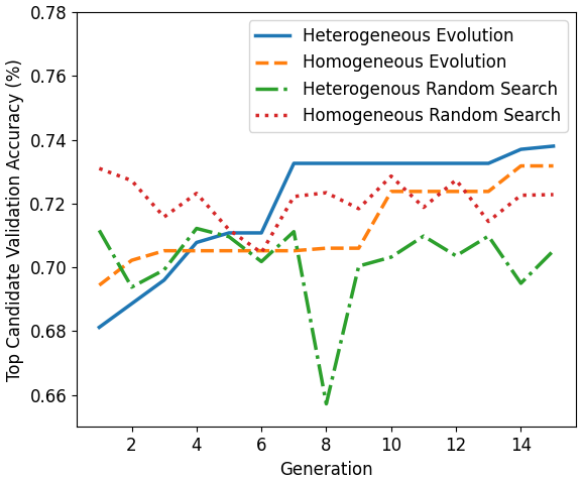


Figure 4.1: Plot of top validation accuracy reached in each generation of two search strategies in the homogeneous (S_1^1) and heterogeneous (S_1^3) solution space. More precisely, both random search and loss-based evolutionary search were used. All solutions are trained with VGG-HE-2 on CIFAR-10 for 50 epochs, then evaluated based on a corresponding validation set. The best validation accuracy reached by a solution in each epoch is plotted for every search. Both evolutionary searches display clear improvement over the generations. Here, the random searches offer an illustrative baseline. Indeed, all randomly sampled homogeneous solutions performed better than the randomly sampled heterogeneous solutions. Note, both evolutionary searches suffer from relatively poor first generation performance, this is due to chance. However, despite starting with the worst performing pool of initial candidate solutions, evolution in the heterogeneous search space yields the overall best performing candidates.

are compared to random searches. Figure 4.1, shows a plot of the top validation accuracy reached by a candidate solutions in the population per generation. That is, the highest accuracy reached on the validation set by any candidate solution in the population after being trained with VGG-HE-2 on CIFAR-10 for 50 epochs. Each line in the plot represents a different search and domain combination: evolutionary search in S_1^1, random search in S_1^1, evolutionary search in S_1^3, and random search in S_1^3. As the generations elapse, there is a clear increasing monotonic trend for both evolutionary searches. In fact, it appears that by the final generation, the best candidate solutions discovered outperform all previously found solutions including those evaluated in the corresponding random searches. As such, these results indicate that our evolutionary search method outperforms random search in both the homogeneous and heterogeneous domain.

Additionally, to gauge whether the pattern of optimization remains consistent, a single run of a 30 generation evolutionary search in the heterogeneous domain was performed. Again, CIFAR-10 and VGG-HE-2 was used to evaluate the candidate solutions. Figure B.1 (see Appendix **??**) shows a plot of the results. Unfortunately, this search fails to discover candidate solutions that clearly outperform the 15 generation evolutionary search in the same domain - despite using identical parameter values. In fact, in both searches, it appears that the rate of optimization slows down significantly after the first 10 generations. That said, although improvement over generations is sometimes slow and inconsistent, a general upward trend is nevertheless visible.

Moreover, returning to Figure 4.1, it is noteworthy that compared to the top validation accuracies reached per "generation" in the random searches, both evolutionary searches suffered from relatively poor performing initial populations. This is fully determined by chance, but may have impacted the outcome of the searches significantly. Another notable aspect of these findings is that random search in the heterogeneous domain consistently outperformed random search in the homogeneous domain across all generations. Although not conclusive, this may suggest that there is a greater proportion of well-performing solutions in the homogeneous search space.

Next, using the training/testing cycles described

TOP 3 CANDIDATES FROM EACH CIFAR-10 SEARCH				MEDIAN TEST ACCURACY AFTER 200 EPOCHS (5 REPEATS)		
				CIFAR-10 & VGG-HE-2	CIFAR-100 & VGG-HE-2	CIFAR-10 & VGG-HE-4
Heterogeneous Evolution (S_1^3)						
1.	$\max(\max(x,0),\lvert x\rvert)$	$\operatorname{arcsinh}(x)+\lvert x\rvert$	$\max(\operatorname{erf}(x),x)$	80.44	**51.02**	76.29
2.	$\max(\max(x,0),(-x))$	$\log(1+e^x)+\lvert x\rvert$	$x\times 1$	**80.63**	50.97	10.00
3.	$\max(\max(x,0),(-x))$	$x+\lvert x\rvert$	$x/(\tanh(x)+\epsilon)$	80.15	50.10	78.42
Heterogeneous Random Search (S_1^3)						
1.	$\lvert x\rvert-(-x)$	$\max(x,\sqrt{x})$	$x+\min(x,0)$	78.78	50.01	79.15
2.	$\lvert x\rvert\times\operatorname{sinc}(x)$	$x-\tanh(x)$	$(1/(1+e^{-x}))/(e^x+\epsilon)$	78.10	45.73	61.74
3.	$\log(1+e^x)\times(1/(1+e^{-x})))$	$\log(1+e^x)-\cos(x)$	$\min(\operatorname{arcsinh}(x),e^x)$	79.70	49.33	78.42
Homogeneous Evolution (S_1^1)						
1.	$\lvert x\rvert-\operatorname{arcsinh}(x)$			80.38	50.07	79.25
2.	$\lvert x\rvert+\operatorname{arcsinh}(x)$			80.34	50.20	79.38
3.	$\lvert x\rvert-\operatorname{erf}(x)$			79.64	49.90	76.19
Homogeneous Random Search (S_1^1)						
1.	$\min(\lvert x\rvert,\max(x,0))$			79.58	49.85	80.27
2.	$\max((-x),\log(\lvert x+\epsilon\rvert))$			**80.63**	49.92	78.18
3.	$\lvert x\rvert+\operatorname{arcsinh}(x)$			79.80	49.67	78.58
Baseline Activation Functions						
	ReLU			79.39	49.45	80.13
	Swish			79.90	49.88	**81.26**

Table 4.1: Listed are the top 3 candidate solutions from each 15 generation search, along with two benchmarks - ReLU and Swish. The best performing solutions are selected from the searches according to their validation accuracy after training with VGG-HE-2 on CIFAR-10 for 50 epochs. The final test accuracy displayed is determined by retraining with VGG-HE-2 on CIFAR-10 for 200 epochs and evaluating performance on an unseen test set. Furthermore, to evaluate transfer performance to a different dataset, the best candidates from the CIFAR-10 searches are retrained with VGG-HE-2 on CIFAR-100 for 200 epochs. Similarly, to determine transfer performance to a larger NN architecture, these same candidate solutions are retrained with VGG-HE-4 on CIFAR-10 for 200 epochs. Here again, classification performance is measured with respect to a corresponding unseen test set. Note, all values listed are the median final test accuracies over 5 repeats. The values made bold represent the highest median final test accuracy reached by a candidate solution for each experiment setup.

Candidate	Mean Accuracy (95% C.I.)
Best of S_1^1	80.17 ($\pm$0.19)
Best of S_1^3	80.58 ($\pm$0.19)
Swish	79.88 ($\pm$0.18)
ReLU	79.31 ($\pm$0.39)

Candidate	t-statistic; p-value	
Best of S_1^1 vs Swish	2.31; 2.62×10^{-2}	(*)
Best of S_1^1 vs ReLU	4.13; 3.11×10^{-4}	(*)
Best of S_1^3 vs Swish	5.65; 1.74×10^{-6}	(*)
Best of S_1^3 vs ReLU	6.14; 1.45×10^{-6}	(*)
Best of S_1^1 vs Best of S_1^3	3.29; 2.19×10^{-3}	(*)

Table 4.2: Mean of final test accuracy (with 95% confidence intervals), evaluated after 200 epochs of training with VGG-HE-2 on CIFAR-10, over 20 repeats. Performed for the best candidate solutions yielding from the S_1^1 (homogeneous) and S_1^3 (heterogeneous) searches - as determined by their median final test accuracy over 5 repeats (see 4.1). These are $\max((-x),\log(\lvert(x+\epsilon)\rvert))$ and $[\max(\max(x,0),(-x));\log(1+e^x)+\lvert x\rvert;x\times 1]$, respectively. Furthermore, ReLU and Swish are also evaluated for comparison. Lastly, independent sample two-tailed heteroscedastic (Welch's) t-tests are performed to determine the significance of found differences between mean final test accuracies. Given the cutoff $p\leq .05$, the significant findings have been marked with (*).

Candidate	Mean Accuracy (95% C.I.)
Best of S_1^1	50.08 ($\pm$0.28)
Best of S_1^3	50.77 ($\pm$0.33)
Swish	50.14 ($\pm$0.28)
ReLU	49.54 ($\pm$0.27)

Candidate	t-statistic; p-value	
Best of S_1^1 vs Swish	0.35; 7.30×10^{-1}	
Best of S_1^1 vs ReLU	2.93; 5.70×10^{-3}	(*)
Best of S_1^3 vs Swish	3.01; 4.71×10^{-3}	(*)
Best of S_1^3 vs ReLU	6.04; 5.91×10^{-7}	(*)
Best of S_1^1 vs Best of S_1^3	3.33; 1.96×10^{-3}	(*)

Table 4.3: Mean of final test accuracy (with 95% confidence intervals), evaluated post 200 epochs of training with VGG-HE-2 on CIFAR-100, over 20 repeats. Performed for the best candidate solutions yielding from the S_1^1 (homogeneous) and S_1^3 (heterogeneous) searches - as determined by their median final test accuracy over 5 repeats (see 4.1). These are $\max((-x), \log(|(x+\epsilon)|))$ and $[\max(\max(x,0),|x|)$; $\text{arcsinh}(x)+|x|$; $\max(\text{erf}(x),x)]$, respectively. Furthermore, ReLU and Swish are also evaluated for comparison. Lastly, independent sample two-tailed heteroscedastic (Welch's) t-tests are performed to determine the significance of found differences between mean final test accuracies. Given the cutoff $p \leq .05$, the significant findings have been marked with (*).

in Section 4, the true performance of the overall top 3 candidate solutions per search is approximated. Figure 4.1 shows these results - including those gauging true transfer performance. Note, the highest median testing accuracies achieved by the various AF setups - per architecture and dataset combination - have been made bold.

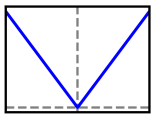
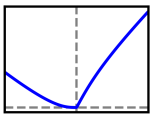
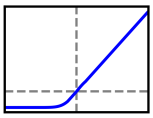
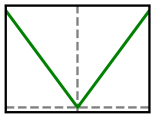
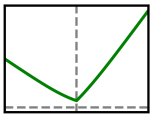
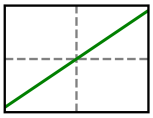
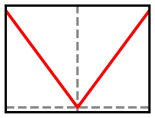
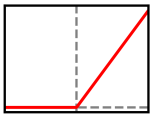
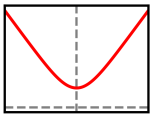
First, consider the non-transfer experiment results. These suggest that the best heterogeneous AF setup discovered, $[\max(\max(x,0),(-x))$; $\log(1+e^x)+|x|$; $x \times 1]$, comes from the evolutionary search. However, a homogeneous solution, $\max((-x), \log(|x+\epsilon|))$, performing equally well was found through random search. Both candidate solutions outperform the baseline AFs - ReLU and Swish. In the dataset transfer experiment, the functions that performed well on CIFAR-10 in the searches successfully generalize to CIFAR-100. In fact, despite no search being performed on CIFAR-100, most top search candidates, whether homogeneous or heterogeneous, outperform the baseline AFs. Specifically, $[\max(\max(x,0),|x|)$; $\text{arcsinh}(x)+|x|$; $\max(\text{erf}(x),x)]$, one of the top candidate solutions coming from evolutionary search in the S_1^3 domain, claimed the highest overall median test accuracy. In contrast, all top search candidates showed relatively poor architecture transfer performance. That is when the architecture was changed to VGG-HE-4. Indeed, not a single tested top search candidate was able to outperform Swish and only one was able to surpass the testing accuracy of ReLU - albeit marginally. Furthermore, one of the evolved heterogeneous solutions, $[\max(\max(x,0),(-x))$; $\log(1+e^x)+|x|$; $x \times 1]$, failed to accommodate for any learning whatsoever.

To evaluate whether the accuracy differences between the best tested search candidates are significant, the training/testing cycle is repeated 20 more times. Table 4.2 presents the results for the CIFAR-10 and VGG-HE-2 combination. Note, this is the same problem setup that was originally used for the searches. Regarding mean test accuracy, the best candidate solution found in S_1^3 (M=80.58, SD=0.40) outperformed the best found in S_1^1 (M = 80.17, SD=0.40) and the best benchmark (M=79.88, SD=0.39) - Swish. The Welsh's t-tests between the best heterogeneous and best homogeneous solution ($t(38) = 3.29$, $p = 2.19 \times 10^{-3}$) as well as that between the best heterogeneous and best benchmark solution ($t(38) = 3.01$, $p = 4.71 \times 10^{-3}$) - confirm the significance of these findings. Likewise, the best homogeneous solution outperforms the best benchmark ($t(38) = 5.65$, $p = 1.74 \times 10^{-6}$). Note, unlike the best tested heterogeneous solution, the best tested homogeneous solution was discovered through random search.

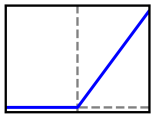
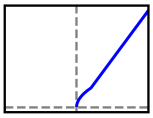
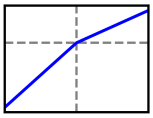
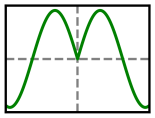
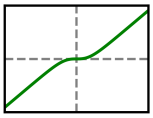
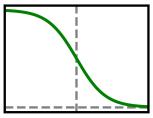
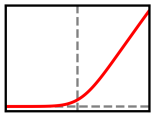
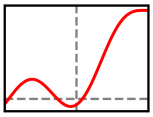
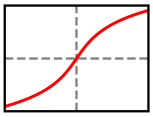
Similarly, to evaluate the significance of the dataset transfer results, 20 repeats of 200 epoch of training followed by testing was performed using the top search candidates that performed best on CIFAR-100 and VGG-HE-2 (see Table 4.1). Here again, the best heterogeneous solution (M = 50.77, SD = 0.71) outperformed the best homogeneous solution (M = 50.08, SD = 0.59) - significantly ($t(37) = 3.33$, $p = 1.96 \times 10^{-3}$). In addition, it reached higher mean accuracy than Swish (M = 50.14, SD = 0.60), the best benchmark ($t(37) = 3.01$, $p = 4.71 \times 10^{-3}$). On the other hand, the best

TOP 3 CANDIDATES FROM EACH CIFAR-10 SEARCH VISUALISED

Heterogeneous Evolution (S_1^3)

● $\max(\max(x, 0), \lvert x\rvert)$	$\operatorname{arcsinh}(x) + \lvert x\rvert$	$\max(\operatorname{erf}(x), x)$
		
● $\max(\max(x, 0), (-x))$	$\log(1 + e^x) + \lvert x\rvert$	$x \times 1$
		
● $\max(\max(x, 0), (-x))$	$x + \lvert x\rvert$	$x/(\tanh(x) + \epsilon)$
		

Heterogeneous Random Search (S_1^3)

● $\lvert x\rvert - (-x)$	$\max(x, \sqrt{x})$	$x + \min(x, 0)$
		
● $\lvert x\rvert \times \operatorname{sinc}(x)$	$x - \tanh(x)$	$(1/(1 + e^{-x}))/(e^x + \epsilon)$
		
● $\log(1 + e^x) \times (1/(1 + e^{-x}))$	$\log(1 + e^x) - \cos(x)$	$\min(\operatorname{arcsinh}(x), e^x)$
		

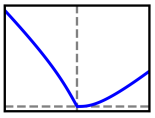
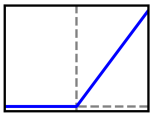
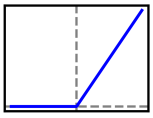
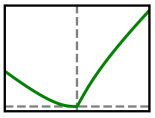
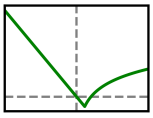
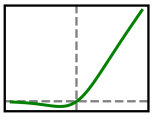
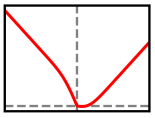
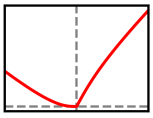
Homogeneous Evolution (S_1^1)	**Homogeneous Random Search (S_1^1)**	**Baseline AFs**
● $\lvert x\rvert - \operatorname{arcsinh}(x)$	● $\min(\lvert x\rvert, \max(x, 0))$	● ReLU
		
● $\lvert x\rvert + \operatorname{arcsinh}(x)$	● $\max((-x), \log(\lvert x + \epsilon\rvert))$	● Swish
		
● $\lvert x\rvert - \operatorname{erf}(x)$	● $\lvert x\rvert + \operatorname{arcsinh}(x)$	
		

Table 4.4: Top 3 candidate solutions from each 15 generation CIFAR-10 search and baseline AFs visualized. Function plots have a domain of $x \in (-5, 5)$, with varying ranges. The dotted lines in each plot represent the axes.

homogeneous solution performed on par with the best benchmark ($t(38) = 0.35$, $p = 7.30 \times 10^{-1}$). In this experiment, both the best homogeneous and heterogeneous solutions came from evolutionary searches. Lastly, no further investigation into architecture transfer was attempted due to the unfavorable preliminary results.

Finally, using Table 4.4, several observations can be made about the AF function shapes of the top candidate solutions. Starting with the top homogeneous solutions, irrespective of the search method used to discover them, they are generally analogous - except for a ReLU equivalent. That is, they are all similarly shaped non-monotonic functions which render most of their domain positive. Note, the gradient on either side of the y-axis differs. The similarity between the top homogeneous candidate solutions, independent of search strategy, may suggest that the global optimum is being approximated. Regarding $min((|x|, max()(x, 0)))$, it is clearly a needlessly complicated formulation of ReLU. As such, this finding motivates the optimization of top candidate solutions post-search to maximize their efficiency.

Furthermore, the evolved heterogeneous solutions are highly similar, it is clear that the directed search is approximating a local optimum. In fact, the first AF in all of these solutions are functionally identical. On the other hand, the heterogeneous candidate solutions yielding from the random search follow no clear pattern. In fact, they are highly unintuitive. Notably, $[|x| - (-x); \max(x, \sqrt{x}); \ x + \min(x, 0)]$. Whereby, the second AF is undefined for negative values due to the square root. The use of this AF is only possible because the AF preceding it (which is functionally identical to ReLU) that maps all of its negative domain to zero. As such, these partially undefined functions can only exist in heterogeneous configurations. Whether these solutions are robust across many trials requires further investigation.

5 Discussion & Future Work

The results presented above clearly show that there exist heterogeneous AF configurations for DNNs that outperform their homogeneous counterparts - including ReLU and Swish. Specifically, heterogeneous solutions which involve applying multiple AFs to a DNN in a block-wise manner. Likewise, in line with the findings of Bingham et al. [2020], the results suggest that it is possible to discover specialized AFs for a homogeneous setup that significantly outperform commonly used AFs. As far as the search method used for doing so, the presented evolutionary (genetic algorithm-based) approach is successful. However, using random search as a point of comparison, it appears that evolutionary search may be better suited to the larger heterogeneous search space with proportionally fewer good solutions. Another noteworthy influence on the effectiveness of evolutionary search is that of population initialization. Indeed, an initial pool of poor candidate solutions can misdirect and prolong the search. Its effects may be tempered by increasing exploration in favor of exploitation. In general, finer tuning of the evolutionary search parameters, with respect to the individual search spaces, is sure to lead to the faster discovery of well-performing candidate solutions. Furthermore, the slowing speed of optimization over the course of the longer evolutionary search suggests that scheduling parameter changes may be beneficial.

Unsurprisingly, AF configurations perform best on the datasets and architectures used to discover them. However, the transfer experiments performed in this work present strong evidence for the generalizability of discovered solutions. Specifically, in the near transfer task to CIFAR-100, the top candidate solutions from the CIFAR-10 searches performed significantly better than ReLU and Swish. This applies to both homogeneous and heterogeneous solutions. On the other hand, the transfer task to a larger NN was comparatively unsuccessful. Note, the method of transfer used is novel and irregular. That said, despite not being able to outperform baseline AFs, most tested candidate solutions still enabled learning and resulted in reasonable performance. As such, there is reason to believe that more successful transfer methodologies could be developed.

Future research may be able to push such generalization much further by evaluating candidate solutions across multiple datasets and architectures. Indeed, a similar approach to the one Ramachandran et al. [2017] used to discover the highly generalizable Swish AF could be employed to find a widely applicable heterogeneous AF configuration. Indeed, beyond reinforcement learning, evolution is

well-positioned to combine the requirements of the multiple tasks into one fitness function. In its simplest form, a solution resulting from such a process may be two AFs that, in combination, outperform AFs used in a homogeneous fashion. For example, by applying the discovered AFs to the hidden units in the first half of a network and the second half, respectively. Clearly, such a solution would be trivial to apply in practice and could potentially offer users an easy was to improve the performance of their NN models. That said, to achieve this feat, greater investigative care must be taken to identify optimal architecture transfer strategies for heterogeneous AF configurations.

6 Conclusion

An evolutionary strategy for discovering homogeneous and heterogeneous AF configurations was presented. It proved to be better performing than random search - especially in the large heterogeneous search space. Indeed, the novel AF configurations resulting from the searches outperform baseline AFs - such as ReLU and Swish. Furthermore, the best tested heterogeneous solutions outperform the best tested homogeneous solutions - supporting past research on the matter. Moreover, the top AF configurations discovered in the searches successfully transfer from CIFAR-10 to CIFAR-100. However, they fail to transfer from VGG-HE-2 to VGG-HE-4. In fact, the solutions tested in the dataset near transfer task outperform ReLU and Swish. Lastly, it has been successfully demonstrated that heterogeneous AF configurations can be discovered for deep neural networks with reasonable computational resources. Indeed, by leveraging the power of evolutionary search in large search spaces, AF configurations of various degrees of heterogeneity can be discovered. Hereby, the scene is set for future research seeking to discover generalizable heterogeneous AF configurations for deep neural networks.

References

Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

Steven Abreu. Automated architecture design for deep neural networks. *arXiv preprint arXiv:1908.10714*, 2019.

Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

Marijke F Augusteijn and Thomas P Harrington. Evolving transfer functions for artificial neural networks. *Neural Computing & Applications*, 13 (1):38–46, 2004.

Mina Basirat and Peter M Roth. The quest for the golden activation function. *arXiv preprint arXiv:1808.00783*, 2018.

Mina Basirat and Peter M Roth. S* relu: Learning piecewise linear activation functions via particle swarm optimization. In *VISIGRAPP (5: VISAPP)*, pages 645–652, 2021.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

Garrett Bingham, William Macke, and Risto Miikkulainen. Evolutionary optimization of deep learning activation functions. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 289–296, 2020.

Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pages 472–478, 2001.

Mehmet Önder Efe. Novel neuronal activation functions for feedforward neural networks. *Neural processing letters*, 28(2):63–79, 2008.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20 (1):1997–2017, 2019.

Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2(6):476–493, 2021.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Daniel Groos, Lars Adde, Ragnhild Støen, Heri Ramampiaro, and Espen AF Ihlen. Towards human-level performance on automatic pose estimation of infant spontaneous movements. *Computerized Medical Imaging and Graphics*, 95: 102012, 2022.

Alexander Hagg, Maximilian Mensing, and Alexander Asteroth. Evolving parsimonious networks by mixing activation functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 425–432, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015b.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Andrew G. Howard. Some improvements on deep convolutional neural network based image classification. *CoRR*, abs/1312.5402, 2014.

Joarder Kamruzzaman and SM Aziz. A note on activation function in multilayer feedforward learning. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 1, pages 519–523. IEEE, 2002.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018.

Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8): 5455–5516, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661. IEEE, 2016.

SD Leoshchenko, AO Oliinyk, SA Subbotin, VA Lytvyn, and VV Shkarupylo. Modification and parallelization of genetic algorithm for synthesis of artificial neural networks. *Radio Electronics, Computer Science, Control*, (4):68–82, 2019.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Yong Liu and Xin Yao. Evolutionary design of artificial neural networks with different nodes. In *Proceedings of IEEE international conference on evolutionary computation*, pages 670–675. IEEE, 1996.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

Franco Manessi and Alessandro Rozza. Learning combinations of activation functions. In *2018 24th international conference on pattern recognition (ICPR)*, pages 61–66. IEEE, 2018.

Mark JL Orr. Regularised centre recruitment in radial basis function networks. In *Centre for Cognitive Science, Edinburgh University*. Citeseer, 1993.

F Piazza, A Uncini, and M Zenobi. Artificial neural networks with adaptive polynomial activation function. 1992.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Leon René Sütfeld, Flemming Brieger, Holger Finger, Sonja Füllhase, and Gordon Pipa. Adaptive blending units: Trainable activation functions for deep neural networks. In *Science and Information Conference*, pages 37–50. Springer, 2020.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

Andrew James Turner and Julian Francis Miller. Cartesian genetic programming encoded artificial neural networks: a comparison using three benchmarks. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1005–1012, 2013.

Andrew James Turner and Julian Francis Miller. Neuroevolution: evolving heterogeneous artificial neural networks. *Evolutionary Intelligence*, 7(3): 135–154, 2014.

Daniel Weingaertner, Victor K Tatai, Ricardo R Gudwin, and Fernando J Von Zuben. Hierarchical evolution of heterogeneous neural networks. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1775–1780. IEEE, 2002.

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Tao Zeng, Bian Wu, and Shuiwang Ji. Deepem3d: approaching human-level performance on 3d anisotropic em image segmentation. *Bioinformatics*, 33(16):2555–2562, 2017.

A Appendix

VGG-HE-4		
Id	Layers	Properties
$conv\_b1\_l1$	2D convolution	$32 \times (3, 3)$
	Activation	custom1
	Batch normalization	
$conv\_b1\_l2$	2D convolution	$32 \times (3, 3)$
	Activation	custom1
	Batch normalization	
	2D Max pooling	$(2, 2)$
	Dropout	20%
$conv\_b2\_l3$	2D convolution	$64 \times (3, 3)$
	Activation	custom1
	Batch normalization	
$conv\_b2\_l4$	2D convolution	$64 \times (3, 3)$
	Activation	custom1
	Batch normalization	
	2D Max pooling	$(2, 2)$
	Dropout	30%
$conv\_b3\_l5$	2D convolution	$128 \times (3, 3)$
	Activation	custom2
	Batch normalization	
$conv\_b3\_l6$	2D convolution	$128 \times (3, 3)$
	Activation	custom2
	Batch normalization	
	2D Max pooling	$(2, 2)$
	Dropout	40%
$conv\_b4\_l7$	2D convolution	$256 \times (3, 3)$
	Activation	custom2
	Batch normalization	
$conv\_b4\_l8$	2D convolution	$256 \times (3, 3)$
	Activation	custom2
	Batch normalization	
	2D Max pooling	$(2, 2)$
	Dropout	50%
fc_l9	Fully connected	256
	Activation	custom3
	Batch normalization	
	2D Max pooling	$(2, 2)$
	Dropout	60%
fc_l10	Fully connected	10
	Activation	softmax

Table A.1: Description of the 4 block heterogeneous VVG (VVG-HE-4) architecture. Note, it has been set up according to the transfer task. That is, each two consecutive VGG blocks use the same activation function, whilst the activation function of the penultimate fully connected layer is the same as the equivalent layer in the smaller network. The AFs are depicted in generalized format, namely custom1, custom2, and custom3. These refer to the AFs represented by individual candidate solutions yielding from $S_{1,3}$

B Appendix

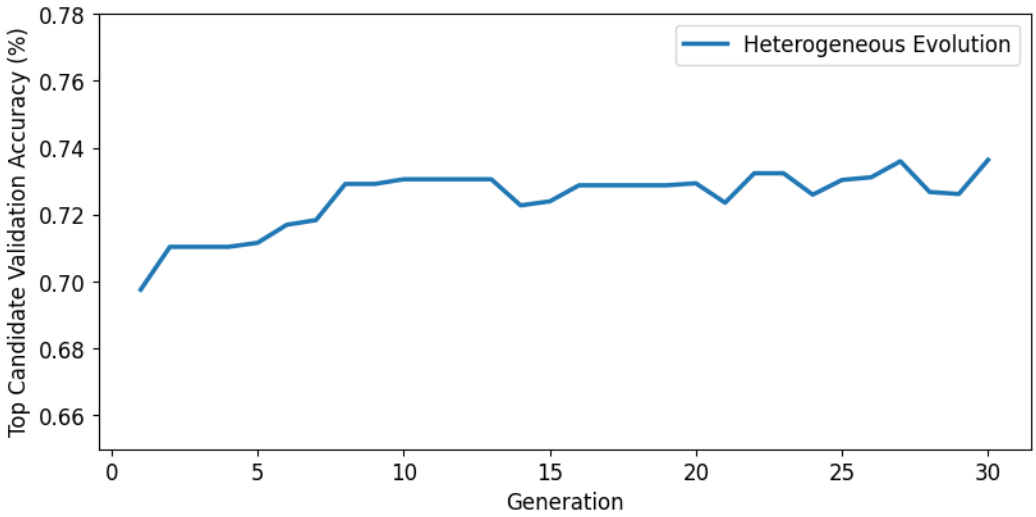


Figure B.1: Plot of top validation accuracy reached by a candidate solution per generation of an evolutionary search in the heterogeneous domain. It was performed for 30 generations and uses the same evolutionary search parameters as the 15 generation searches from Figure 4.1. Similarly, 50 epochs of training using VGG-HE-2 on CIFAR-10 followed by testing on a validation set was used to evaluate the performance of the candidate solutions.