An Immune-Inspired Approach to Macro-Level Neural Ensemble Search

Luc Frachon
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
luc.frachon@hw.ac.uk

Wei Pang
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
ORCID iD 0000-0002-1761-6659

George M. Coghill
School of Natural and Computing Sciences
University of Aberdeen
Aberdeen, United Kingdom
ORCID iD 0000-0002-2047-8277

2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504955

***Abstract*—Recent years have seen a renewed interest in evolutionary computation applied to the automatic design of deep neural network architectures, i.e. Neural Architecture Search (NAS). The advantages of evolutionary approaches in NAS include their conceptual simplicity and their flexibility with regards to search space definition and/or optimization objective.**

However, Artificial Immune Systems (AIS) that follow the evolutionary computation paradigm are less explored in NAS. In this research, we aim to leverage their intrinsic and excellent ability to balance performance and population diversity to develop a novel Neural Ensemble Search method, based on the Clonal Selection Algorithm [1]. For more generality, we focus on designing macro-architectures rather than architectural components.

Experiments on popular computer vision benchmarks demonstrate that our method reaches competitive accuracy and efficiency despite minimal augmentation and post-processing. We show that the AIS brings tangible benefits, including maintaining the diversity of solutions, a semantically straightforward implementation, and high efficiency. Moreover, this AIS can exhibit a "secondary response": when presented with a related but more difficult task, the ensemble will perform competently with zero modification to the architectures or the training protocol.

***Index Terms*—Deep Neural Networks, Artificial Immune Systems, Neural Ensemble Search**

I. Introduction

Deep Learning's (DL) achievements are largely due to ingenious architectures painstakingly developed by teams of experts. The number of neural network configurations grows exponentially with the depth of the network. To make the problem tractable, simplifying assumptions are made, such as repeated blocks of operations [2], [3]. Even so, for most research teams, the only feasible option is transfer learning; however, a model trained on data "similar" to the target task is not always available. Hence the growing appeal of Neural Architecture Search (NAS), i.e., methods that automatically design DL architectures. In this section, we first give a brief overview of different methods in NAS, then present some advantages of Artificial Immune Systems (AIS). Finally, we discuss features of micro- and macro-search spaces.

A. NAS Methods

The first work to focus on automating the *architecture* design of *deep* neural nets is [4] by Zoph *et al.*, who trained a reinforcement learning (RL) agent to select and configure layers. They achieved results on par with some of the best handcrafted models, albeit with much postprocessing. However, these results required over 20,000 GPU-days, making the method largely impractical. Around the same time, Real *et al.* [5] achieved similar results in 3,000 GPU-days by using an evolutionary algorithm (EA).

Recent years have seen a proliferation of NAS methods. In addition to RL and EA, other paradigms have been used to guide the search, including Bayesian optimisation [6], [7], one-shot architecture search [8]–[10]. Although efficient, the latter uses restrictive search spaces. Its effectiveness also came under scrutiny [11]–[13], with suggestions that the performance originates more from search space engineering or elaborate training procedures than the search itself. In addition, state-of-the-art (SOTA) methods always employ a great deal of augmentation techniques, postprocessing, or training tricks in order to improve results, which depend heavily on human expertise [13]. This is somewhat at odds with the point of NAS and obfuscates comparisons between methods.

Evolutionary approaches are popular in NAS due to their flexibility, ease of implementation, and efficiency. Examples include genetic algorithms (GA) [5], [14], [15], genetic programming [16], and various swarm intelligence algorithms [17]. However, AIS have been little explored thus far.

An emerging trend is Neural Ensemble Search (NES), i.e. the use of NAS to build ensembles of neural networks [18], [19]. In the latter work (post-dating this paper's preprint), Zaidi *et al.* directly optimize the ensemble performance and show how neural ensembles help dealing with test set corruption. Our focus is different: we want to understand if an AIS brings significant benefits in macro-level NES, and if this approach can be an alternative to conventional NAS.

B. Benefits of Artificial Immune Systems

AIS algorithms were developed in the 1990s and early 2000s for pattern recognition and optimisation problems [1]. They are population-based and their main advantage compared

to many EAs is in their excellent ability to intrinsically balance performance with population diversity, and to search in multimodal problem spaces [1], [20], [21].

These features make AIS natural candidates for NES. Indeed, a diverse population allows us to ensemble models to obtain an economical performance gain, since the models are already available. The use of an AIS had previously been explored for hyper-parameter (HP) optimisation in shallow neural networks, and [21] showed that the AIS enables a more diverse and thus more effective ensemble than other population-based algorithms. This motivates us to investigate AIS as applied to the search of deep neural ensembles.

C. Macro vs. micro-search spaces

A key difficulty in NAS is balancing the size of the search space with its expressivity and potential for innovation. Since [22], most NAS works adopt a micro-search approach: Instead of searching for a complete "macro-architecture", they only design up to two different cells with a small number of components. The candidate architectures are a sequence of repeated cells within a hand-designed skeleton. Micro-architecture search dramatically shrinks the search space. It also helps with transferability to more complex tasks, simply by increasing the channel and cell counts.

Such architectures replicate the repetitive patterns of common handcrafted models such as ResNet [2]. While cell-based search may be efficient and produce SOTA results, it has several drawbacks: 1) Lack of innovation in the macro-architecture; 2) the same discovered cell has to work in a wide variety of contexts at different points in the network; 3) requirement for subject-matter expertise that may not always be available, for instance when it comes to manually fixing the network depth; 4) critically, reference [13] shows that human-designed elements play a larger role in the performance than the automated search. In [23], Hu *et al.* show that the performance difference between micro- and macro-search comes predominantly from their starting points: micro-search methods start from more complex candidates that are largely handcrafted, and this advantage carries throughout the search. In contrast, the present work uses a branching macro-architecture space whose initial architectures are not hand-designed. Moreover, it can produce models of arbitrary depth, which is useful when previous knowledge is scarce.

D. Research Questions

Rather than chasing SOTA performance on benchmark datasets, we aim at answering the following questions:

- Can an evolutionary macro-search method yield competitive results inherently, i.e. without resorting to complex pre- and post-processing engineering?
- Does the use of an AIS provide significant benefits in ensembling over established population-based baselines?
- Is such an ensemble able to transfer to harder tasks?

In the remainder of this paper, we first introduce the ImmuNES framework in Section II. In Section III, we detail the search algorithm and its operators. We then present experimental results in Section IV. Finally, in Section V, we discuss our results and suggest future work.

II. The ImmuNES Framework

In this section, we first describe the key ideas incorporated in ImmuNES and how they depend on each other. We then explain how architectures are encoded. Finally, we present the search space.

A. Key concepts

1) Artificial Immune System: AIS are a family of population-based algorithms. One typical example is the Clonal Selection Algorithm (CLONALG) [1], which presents many similarities with GA, but with specific immune operators. The first key difference is that mutation sizes of clones derive from the performance (*affinity*) of the parent, rather than from uniform sampling. This allows the search to be local around good individuals, and wider around poor ones, thus balancing exploration and exploitation. Secondly, GA relies on the crossover operation to improve exploration, while CLONALG simply adds a few randomly-generated individuals. In macro-NAS, this seems more suitable than combining sections from two networks, because layers co-adapt during the search.

2) Neural Network Ensembles: Neural Network Ensembles (NNE) have been studied for a long time, including in the context of convolutional neural networks (CNN) [24]. Their performance essentially depends on two factors: The quality of the models and their error diversity. The latter can be made an explicit optimization objective, such as in [25]. An alternative is to use a search algorithm that is intrinsically able to maintain diversity, such as an AIS. In particular, [21] shows that an AIS optimizing for accuracy only can outperform multi-task optimization for both accuracy and diversity.

3) Features of Evolutionary NAS: EA-based NAS commonly has two important characteristics:

- Diminishing performance returns: the search spends most of its time in a plateau phase where performance gains become increasingly costly;
- Numerous candidate evaluations at every generation. In particular, the last generation typically contain dozens of individuals, which all contain useful information learned over the search.

4) Putting Everything Together: Based on the observations above, and instead of spending many GPU-days striving for small performance gains and discarding all but one members of the final population, we stop the search when performance starts to plateau, and exploit the competent and diverse population found by our AIS to build an ensemble. This reduces the performance gap caused by stopping early. The trade-off, namely a one-off additional cost for final training and slower inference times, is acceptable in many applications, e.g. where predictions are not made in real-time.

B. Representation of Architectures

We represent an architecture as a directed acyclic graph where nodes correspond to layers/blocks and edges to tensors,

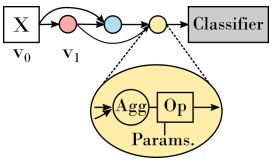

Fig. 1. Graph representation of an architecture with three hidden layers.

TABLE I
SIMPLE SEARCH SPACE (HP: HYPER-PARAMETER)

Operation Type	HP 1 {Values}	HP 2 {Values}	HP 3 {Values}
Conv	Kernel size {1,3,5,7}	BatchNorm {yes/no}	ReLU {yes/no}
DSepConv	Kernel size {1,3,5,7}	BatchNorm {yes/no}	ReLU {yes/no}
Pool	Type {Max, Avg}	Kernel size {3,5}	Ch.multiplier $\{1, 1\frac{1}{3}, 1\frac{2}{3}, 2\}$
Identity	–	–	–

TABLE II
BLOCK SEARCH SPACE (HP: HYPER-PARAMETER)

Operation Type	HP 1 {Possible values}	HP 2 {Possible values}
Resnet Block [2]	Kernel size {3,5}	Downsample {yes/no}
Resnet Bottleneck Block [2]	Kernel size {3,5}	Downsample {yes/no}
Densenet Block [3]	Growth factor {12, 24, 36}	Transition layer {yes/no}
Densenet Bottleneck Block [3]	Growth factor {12, 24, 36}	Transition layer {yes/no}
Inception-Resnet Block A [26]	Kernel size {3, 5}	Bottleneck factor {0.1, 0.4, 0.75}
Inception-Resnet Block B [26]	Kernel size {3, 5}	Bottleneck factor {0.1, 0.4, 0.75}
Pool	Type {Max, Avg}	Kernel size {3,5}
Identity	–	–

see Fig. 1: node v_0 is the input node, which receives data samples. Each subsequent node v_l has at least one incoming edge, which comes from node v_{l-1}. Each hidden layer except v_1 can optionally have a second incoming edge from any node v_{l-k}, $k > 1$, thus creating skip connections [2]. The input node is the only one allowing an outdegree greater than 2, so that all skip connections can always find a source node.

Each node has an *Agg*regation and an *Op*eration. *Aggs* are used to combine tensors from earlier layers. They are *None* when the node has an indegree of 1 but can be *Add* or *Concatenate* when it is 2. *Ops* apply a mathematical transformation to their input (e.g. the ResNet block). As per Tables I and II, each Operation can have several HPs.

C. Search Space

Although human biases are unavoidable in NAS search space design [13], we want macro-architectures to be discovered, with potential for innovation, rather than manually enforced (see Section I-C). We define two search spaces: The "simple" space used in our simpler experiments, and the "block" search space used in the full-scale experiments.

1) Simple Search Space: Our method was initially developed using this search space and the Fashion-MNIST dataset [27]. This task is light-weight, yet challenging enough for performance differences to emerge.

We defined this search space to allow fast iteration during development (see Table I). The chosen operations are typical of conventional CNNs. The Identity *Op* allows the algorithm to remove a layer that was added earlier. *Ch. multiplier* is a multiplicative factor applied to channel counts. The input layer is always a pointwise convolution without batch normalization nor activation function. Its role is to increase the channel count from 1 to 64 to quickly give the network more capacity. The classifier, or network head, is always a sequence of global concatenation pooling, batch normalization, dropout ($p_{drop} = 0.2$) and a fully-connected layer.

2) Block Search Space: Our main experiments were run on CIFAR-10 [28]. Architectures developed for this task traditionally involve a repetition of identical blocks [2], [3], [26]. Almost all recent NAS works replicate this practice and only design these blocks (or 'cells') automatically, rather than the overall architecture. Key features of these macro-architectures (depth, width, etc.) are predefined, which are major human decisions (see Section I-C).

To generate macro-architectures with enough capacity for CIFAR-10, we also build them block by block rather than one operation at a time, but without the enforced repetition of identical cells. To do this while avoiding search space optimization, we create a menu of high-level blocks that are simply chosen *a priori* from the popular deep learning literature. This is similar to [16] but with many more configurations. Unlike all micro- and some macro-search methods [13], we do not pre-define the number of blocks, their position, the location of skip connections, the network's maximum depth[1], or the activation maps' dimensions. Unlike in architectures produced by micro-NAS, skip connections can occur at the macro level. This freedom offers opportunities for interesting architectures to emerge. At the same time, we make the search tractable by adopting progressive search (see Section III-C).

In Table II, *Downsample* indicates whether the block should reduce the spatial size, and increase the channel count, by a factor 2. *Growth factor* is the number of channels that tensors gain as they go through the block. *Transition layer* indicates whether a compression layer is appended to halve the channel count. *Bottleneck factor* is a compression factor that applies to the channel count of the middle branch of the block. We

[1]Although we define a theoretical maximum number of generations, it is never reached in practice due to early stopping.

Neural Ensemble Search by an AIS

Require: N_0*: initial population size,* L_{ini}*: initial # layers,* ρ*: mutation factor,* n_c*: # clones per parent,* n_i*: # random insertions,* n_a*: # augmented networks per parent,* π_1*: outer loop patience,* τ_1*: outer loop threshold,* π_2*: inner loop patience,* τ_2*: inner loop threshold*

```
pop ← MakeRandomArchitectures(N_0)
pop ← MakeAugmentedCopies(pop, n_a) {n_a copies per parent to increase initial beam width}
Evaluate(pop)
counter1 ← 1
while counter1 ≤ π_1 do
    counter2 ← 1
    while counter2 ≤ π_2 do
        clones ← Clone(pop, n_c) {n_c copies per parent}
        clones ← Mutate(clones, ρ)
        Evaluate(clones)
        pop ← SelectNBest(pop ∪ clones, N_0)
        avg_affinity ← ComputeAverageAffinity(pop)
        pop ← MakeRandomArchitectures(N_i)
        Evaluate(pop_{N_0,...,N_0+n_i})
        if avg_affinity improved by less than τ_2 then
            counter2 ← counter2 + 1
        end if
    end while
    pop ← MakeAugmentedCopies(pop, n_a)
    Evaluate(pop_{N+1,...,N(1+n_a)}) {N = N_0 + n_i}
    avg_affinity ← ComputeAverageAffinity(pop)
    if avg_affinity improved by less than τ_1 then
        counter1 ← counter1 + 1
    end if
end while
return pop
```

Fig. 2. Pseudo-code for the search procedure by an AIS

refer the reader to the respective papers for details on these blocks.

The input and classification layers are the same as previously, but with 32 initial channels to reduce the architectures' memory footprint and allow for deeper models.

III. Neural Ensemble Search by an AIS

In this section, we describe our method and its key components. The pseudo-code for the search algorithm is provided in Fig. III. We use "layer" and "block" interchangeably.

A. Overview

The search is conducted by an AIS, starting from a population of N_0 small networks with random layers. To kick-start the process, we first make multiple copies of these candidates and *augment* them, i.e. append a random layer to each (see Section III-C). Parents and offspring are trained, then evaluated on a validation set (see Section III-D). Their validation accuracy represents their *affinity* with the task.

Subsequently, at every generation, each network yields n_c clones. These clones undergo mutations to their connections, *Aggs* and *Ops*. The resulting architectures are evaluated and the best N_0 from the pool of parents and clones are retained to form the next generation. The mean affinity of the population is then computed. If it does not improve by more than a threshold τ_2 within a patience time of π_2 generations, the population goes through *augmentation* (random appended block) again. This progressive search mechanism allows the AIS to generate minimal networks for each task. At every generation, one or two random architectures (with a depth equal to the current average depth of the population) are also inserted into the population to increase exploration. All layers can be subject to mutation at any time, which prevents the population from being locked in the sub-region of the search space defined by the previous layers.

The whole process then restarts from the cloning and mutation step. The search terminates when π_1 consecutive augmentation phases do not yield a mean affinity improvement of more than τ_1. For simplicity, we set $\pi_1 = \pi_2$, $\tau_1 = \tau_2$. By adjusting these values, one can make the search stop earlier or later in the learning curve. In our main experiments, we used a threshold of 0.3% within two generations, corresponding to a relatively steep part of the learning curve, and we also observed that tuning this value (within reason) was not critical to performance.

B. Cloning, Mutation, and Weight Inheritance

In AIS, the magnitude of mutations depends on the affinity of the clone's parent. In our case, because earlier layers have already had multiple opportunities to mutate, we assign a linearly larger mutation variance to more recent layers, i.e. closer to the network's head. In practice, given a clone with L layers, the mutation rate α is computed as $\alpha = e^{-\rho\,\mathrm{f_p}}$, with $\mathrm{f_p}$ being the affinity of the parent architecture and ρ a meta-parameter. Then for the l^{th} layer, the strength of the mutation μ is randomly sampled as follows:

$$\mu \sim \mathcal{N}(0, \sigma^2) \text{ where } \sigma = \alpha \frac{l+1}{L}. \tag{1}$$

Similar to [5], we store a continuous value in the range $[0, 1]$ for each HP and discretize it on the fly when decoding it into architectural features. This allows even small values of μ to add up over time and trigger architectural mutations. Discretization simply involves spliting the interval into as many bins as there are possible values for the discrete HP.

For each clone, we first attempt to mutate connections, then each of the node *Agg*, then their *Op* type, and finally their *Op* HPs. All these perturbations follow the mutation strength equation (1), and each step is only performed if the previous one has not led to a change in the discretized values. This ensures that only small changes are made at every mutation round, thus maintaining consistency between the parents' and clones' performance. We ran a validation experiment in the simple search space that showed Spearman rank correlation values between a parent and its clones' mean affinity that range between 0.53 at depth 3 and 0.70 at depth 9.

All clones inherit their parents' weights in unchanged layers. In the other layers, we apply He initialisation [29].

C. Architecture Augmentation

We grow architectures through progressive search: we periodically clone all networks and insert one random layer into each clone just before the network's head, with a random (optional) skip connection (see Fig. 3). The original networks

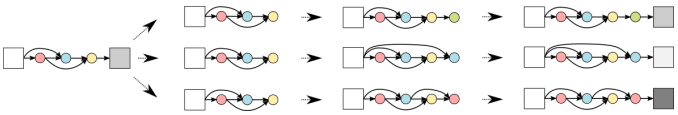

Fig. 3. Architecture augmentation strategy for progressive search.

remain in the population so that unhelpful augmentations can be ignored. As with mutations, clones inherit the weights learned by their parents in unchanged layers, and these weights can be further trained. This method, akin to transfer learning with fine-tuning, improves the time and data efficiency of the new networks' training process. In a second validation experiment, we found rank correlation values of 0.66 between pre- and post-augmentation affinities, with shallow networks benefiting the most, and no catastrophic loss in performance.

D. Evaluation During Search

As the evaluation of each architecture is a major bottleneck in evolutionary NAS, we improve efficiency through partial evaluation. Each candidate is trained on only 20% of the training set with an aggressive early stopping policy: training stops if the validation accuracy does not improve by more than 0.3% within 2 epochs. In the smaller-scale setup, these values were found to yield good performance in less than 0.5 GPU-day (see Section IV-A). Each time the validation accuracy improves, we stash the network's weights, so that when early stopping kicks in (or the maximum allowed number of epochs is reached), these pretrained weights can then be used as a starting point for future clones. The best validation accuracy is recorded as the affinity score. A third validation experiment found high correlations between partial and full evaluation results, between 0.91 at depth 3 and 0.65 at depth 9. The latter correlation value would probably be increased by training large networks for longer in partial evaluation.

During the search, progress is monitored by computing the mean population affinity at every generation. We tried more complex methods to explicitly enforce diversity as in [25], as well as a mechanism to filter out similar architectures as in Opt-AINet [20], an immune network approach. However, the CLONALG's intrinsic ability to maintain diversity proved just as effective, so we retained the simpler approach.

E. Neural Ensemble Building

Once the search terminates, we load the best generation seen and thus have N_0 partially trained architectures. The decision about how many individuals should be part of the ensemble involves a trade-off between accuracy and final training time. The architectures developed for CIFAR-10 tend to be fairly expensive to train (a few GPU-hours each), and the accuracy improves with diminishing returns as more networks are added. The results presented in this paper use the 5 best architectures (ranked by affinity), in order to spend around 1 GPU-day in the final retraining phase.[2] We fully train them without changing the regularization and augmentation protocols compared to partial training, except that we no longer enable early stopping (see details in Table IV).

The NNE's class predictions are obtained by *weighted soft majority vote*. Given a data sample $\mathbf{X}_i$ and a ensemble comprising N_c neural nets g_j with affinity scores f_j ($j \in \{1, \ldots, N_\mathrm{c}\}$), we get a collection of k-dimensional probability vectors $g_j(\mathbf{X}_i)$, where k is the number of classes in our classification task. We then compute the sum $G(\mathbf{X}_i)$ of these vectors weighted by the normalized affinities of the corresponding architectures. Finally, the NNE's prediction is given by $\operatorname{argmax} G(\mathbf{X}_i)$.

[2]We note that few papers report the time spent in final training, which can sometimes [10] include very expensive techniques such as AutoAugment [30].

IV. Experiments and Results

In this section, we first discuss meta-parameter choices. We then compare our main experimental results to existing NAS methods, both using the simpler task (simple search space) and the more complex one. We also confirm that our method significantly outperforms random search, which addresses one of the main criticisms against many NAS methods [11]–[13]. A comparison with GA showcases the advantages of AIS. Finally, we demonstrate that despite not using repeated cells, our method allows task transfer.

A. Meta-Parameter Selection

Yang *et al.* [13] point out that many papers neglect to explain how meta-parameters were chosen, even though they contribute heavily to the final performance. We describe our meta-parameter selection process below, and detail the values used in our experiments in Table III:

1) Search parameters: Meta-parameters were chosen using the simpler task setting (Fashion-MNIST), to allow for faster iteration. Population size was set to 12, a value close to the 10 used in the original CLONALG paper [1], and with a high number of divisors (useful in our implementation). The numbers of mutated and augmented clones were chosen to achieve a search time of around 0.5 GPU-day. The random insertion rate and mutation factor follow values used in [1] (10% of the initial population size, $\rho = 5$). We observed low sensitivity to early stopping patience π and threshold τ: halving τ only changed the final accuracy by 0.1%. We then applied the following rule-of-thumb changes when moving to CIFAR-10: more augmented copies to explore the larger cardinality of the block search space; lower threshold τ to account for the slower progress; halved mutation variance when perturbing operation types to reflect the higher solution density in the mutation interval [0, 1] (more operations).

2) Partial and final evaluation parameters: We restricted data augmentation to the most common techniques in NAS: pad-crop, horizontal flip, Cutout [31]. Regularization is limited to weight decay and dropout in the classifier layer. Our final training protocol is similar to partial evaluations, except that we reduce the initial learning rate by a factor 3, increase the training time, use the full training set, and add warm restarts as is common in cosine learning rate annealing [32]. [13] shows that the addition of DropPath [33], auxiliary towers [34], and

TABLE III
META-PARAMETER SETTINGS

Search space:	Simple	Block
Search		
Initial population size N	12	12
Initial number of random layers	3	6
Mutated / augmented clones per parent	3 / 3	3 / 5
Random insertions each generation	1	1
Mutation factor ρ	5.0	5.0
Patience π / threshold τ	2 / 0.0075	2 / 0.005
Partial Evaluations		
Training set size (20% of data)	12000	10000
Validation set size	12000	10000
Batch size	128	64
Data augmentation	Pad-crop	Pad-crop, Flip, Cutout
Adam initial learning rate	0.1	0.05
Learning rate annealing	Cosine	Cosine
Weight decay	10^{-5}	10^{-5}
Early stopping patience / threshold	2 / 0.005	2 / 0.003
Max. epochs	15	30
Final Training [1]		
Ensemble size	12	5
Training set size	60000	50000
Adam initial learning rate	$\frac{0.1}{3}$	$\frac{0.05}{3}$
Restart after epoch(s)	25	30, 90
Number of epochs	50	210

[1] Unmentioned meta-parameters are identical to partial evaluation.

TABLE IV
EXPERIMENTAL RESULTS ON FASHION-MNIST

Method	Reported	Test accuracy %	GPU-days
XNAS [10] [1]	Best	96.36	0.3
DeepSwarm [17] *	Best	93.56	1.2
NES-RE ens. of 30, no shift [19] [3, †]	Mean±CI	93.0±0.1	NA
Auto-Keras [35]	Best	92.56	0.5
NASH [36] [2]	Best	91.95	0.5
Gradient Evolution [37] *	Best (Med.)	91.36 (90.58)	NA
ImmuNES (ours) †	Best ($\mu \pm \sigma$)	94.61 (94.37±0.15)	0.4

[1] Micro-search, architecture transferred from search on CIFAR-10, strong regularization and augmentation [2] As implemented in [35] [3] No augmentation.
* Evolutionary method † Evolutionary method with ensembling.

even AutoAugment [30] can bring an improvement in top-1 accuracy of around nearly 2% on ResNet-50 [2], which is more than the contribution of most NAS algorithms over random search. As we are not chasing SOTA top-1 performance, we choose not to apply these techniques to give a more accurate representation of our method's contribution.

B. Neural Ensemble Search Experiments

We present experimental results in the simpler search space on Fashion-MNIST, and in the full-scale setting (block search space, CIFAR-10). All experiments were run on NVidia RTX 2080Ti GPUs using the PyTorch framework.

1) Results – Smaller-Scale Setting: The combination of the simple search space and Fashion-MNIST task is intended as a development and proof-of-concept setting. Nevertheless, we obtain interesting results that show the search-space-agnostic nature of our algorithm.

The results over five runs are reported in Table IV with comparisons with representative works (note that the number of papers reporting results on Fashion-MNIST is fairly limited). XNAS [10] outperforms our method, but its architecture is the result of a cell-based search on CIFAR-10 and its final training procedure involves auxiliary towers and AutoAugment [30], whose time requirement is potentially considerable. Among the other methods, insofar as results involving different pre-processing and training protocols can be compared, ImmuNES is highly competitive with a similar search time. The gain from ensembling is around 0.8%pt over the best individual model. Final retraining of all 12 models takes about 9 GPU-hours.

TABLE V
EXPERIMENTAL RESULTS ON CIFAR-10, BY MAIN PARADIGM.

Method	Reported	Test accuracy %	GPU-days
Hand-crafted			
ResNet-110 [2]	Best (Mean±std)	93.57 (93.39±0.16)	–
Micro, non evolutionary			
XNAS-Large [10] [p,r,h,o]	Best	98.40	0.3
NASNet-A 28M [22] [p,r]	Best	97.60	1800
BANANAS [7] [h]	Best	97.36	12
NASNet-A 3.3M [22] [p,r]	Best	97.35	1800
DARTS 2nd Order [9] [p,r,h,o]	$\mu \pm \sigma$	97.17±0.06	4
ENAS Micro + Cutout [h,o]	Best	97.11	0.5
Micro, evolutionary			
NSGANetV1-A4 [38] [r]	Best	97.98	27
AmoebaNet-B 34M [15] [p,r]	$\mu \pm \sigma$	97.87±0.04	4500 (TPU)
AmoebaNet-B 2.8M [15] [p,r]	$\mu \pm \sigma$	97.45±0.05	4500 (TPU)
LEMONADE II [39] [r]	Best	96.60	56
DPP-Net [40] [p,h]	Best	95.64	56
NES-RE ensemble of 30, no shift [19] [n]	Mean±CI	90.8±0.1	NA
Macro, non evolutionary			
NAS depth 39, extra filters [4] [p]	Best	96.35	22400
ENAS Macro [8] [o]	Best	95.77	0.3
NASH snapshot ensemble [36]	Mean	95.30	2
NASH single model [36]	Mean	94.80	1
NAS depth 15 [4] [p]	Best	94.50	22400
Macro, evolutionary			
CNN-GA Cutout [41]	Best	96.78	35
RandGrow No-DropPath [23] [r,h]	Mean	96.62	6
LEMONADE I [39] [r]	Mean	96.50	56
Large-Scale Evo. ensemble [5]	Best	95.60	3000
CGP-CNN [16] [h]	Best ($\mu \pm \sigma$)	94.99 (93.90±0.89)	30
Large-Scale Evo. [5]	Best ($\mu \pm \sigma$)	94.60 (94.10±0.40)	3000
ImmuNES (ours)	Best ($\mu \pm \sigma$)	95.62 (94.97±0.50)	14

[p] Significant post-processing [r] Strong augmentation/regularization [n] No augmentation
[h] Complex hand-designed initial architecture [o] One-shot search with weight-sharing

2) Results – Full Scale Setting: We complete three runs of ImmuNES on the CIFAR-10 task, using the block search space. Table V compares our results with influential and diverse works in NAS. The variety of training procedures, augmentation and post-processing methods, and search space definitions, makes direct comparisons difficult. To try and bring some clarity, we categorize the works along two axes: micro/macro-search, and evolutionary/other algorithm. For each method, we also characterize the amount of manual post-processing (e.g. meta-parameter search, depth or width increase), augmentation/regularization (e.g. drop-path, auxiliary towers, AutoAugment), and complexity of the starting architectures. Two trends seem to emerge: First, accuracy is closely related to the amount of manual intervention and expertise involved. Micro-search generally outperforms macro-search, possibly due their large hand-designed macro-architectures [23]. Secondly, one-shot NAS (XNAS, ENAS, DARTS) is faster than evolutionary NAS (see concerns in Section I-A).

Our method uses straightforward search space definitions, no post-processing, limited regularization, and random ini-

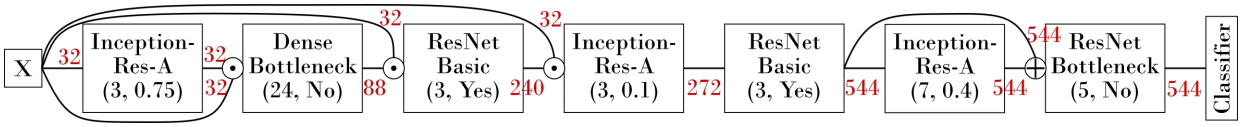

Fig. 4. Best individual architecture on CIFAR-10 (94.5%pt test accuracy), with (HP 1, HP 2) values for each node. $\odot$: Concatenate, $\oplus$: Add. The numbers outside boxes represent channel counts.

tial architectures. We believe these choices to be important for clarity and accessibility even if they sacrifice some performance. Nevertheless, we achieve a reasonable performance/efficiency balance, particularly among evolutionary macro-search methods. Notably, we outperform Large Scale Evolution [5] with two orders of magnitude less GPU-days. CNN-GA achieves slightly higher accuracy without sophisticated manual tricks, but at twice the computational cost.

The final retraining (5 architectures) takes approximately 1 GPU-day and inference 2ms per sample. Ensembling consistently brings a gain of 1.1-1.5%pt in classification accuracy, which, based on observations of the learning curve, would only have been possible with at least 8-10 more GPU-days.

Fig. 4 shows the best architecture found by ImmuNES, whose individual test accuracy is 94.50%. This architecture is noticeably shallower than popular human-designed models. Interestingly, it uses skip connections with concatenation between blocks in the early layers to quickly increase capacity, then goes to a more linear structure.

C. Comparison to Random-Search-Based NES

Recent papers [11]–[13] suggest that random search (RS) on a well-engineered search space might perform similarly to some popular NAS methods. Like [19], we pitch ImmuNES against a RS NES in the smaller-scale setting, generating random network architectures, then training and ensembling them as in ImmuNES. We run RS five times at equivalent evaluation budgets as ImmuNES. We pay close attention to the sampling of the number of layers. ImmuNES starts from 3 layers and its best-found models have 8. To be fair to both methods, the range should not be too wide (which would penalise RS), nor should RS benefit directly from ImmuNES's discovery of the optimal model depth. We, therefore, let RS sample uniformly from the range $\{3 \dots 12\}$. RS only achieves 93.37±0.35% test accuracy for the ensemble (best: 93.79%), significantly worse than our AIS (one-tailed $p < 0.001$). Its best models have 10 layers, against 8 for ImmuNES.

D. Comparison to Genetic Algorithm

In [5], Real *et al.* implement a GA without crossover. Unlike CLONALG, it samples mutation strengths from a uniform distribution and selects candidates by pairwise tournament selection. They include results from an ensemble of networks found by GA, with a 1% gain in accuracy on CIFAR-10.

We take our ImmuNES framework and simply replace the AIS search algorithm with this GA. We run it five times under the same evaluation budget in the smaller-scale setting. The final ensemble's mean test accuracy is 93.95±0.4% with a best result of 94.41%, significantly below ImmuNES (one-tailed $p = 0.039$). While the mean accuracy of the best individual model is very close (93.39% for ImmuNES vs. 93.31% for GA), the gain from ensembling is higher in ImmuNES (0.8%pt vs. 0.6%pt), suggesting that the AIS promotes better diversity.

E. Transferability from CIFAR-10 to CINIC-10

Micro-NAS solutions can transfer to harder tasks by manually increasing the number of cells or their channel count. However, in macro-NAS, adding layers manually is not an option, and when the channel count is determined by the algorithm (as in [5], [16]), it also does not make sense to manually increase it. Therefore, such methods are rarely evaluated for transferability.

However, we hypothesize that the multiple nature of ImmuNES's solutions will help them transfer to a harder task *without any modification*. To push the biological analogy further, this ability could be seen as a secondary response of the antibody population to a related, but different, antigen.

To reuse the architectures directly, we choose CINIC-10 [42], a drop-in replacement to CIFAR-10[3], but significantly harder (see cited paper). We take the best ensemble found on CIFAR-10 and train it on CINIC-10 with the exact same training procedure, including the number of epochs. We obtain a test accuracy of 88.72%, with a 1.36%pt gain from ensembling. Note that the harder task would justify more augmentation, training, or networks in the ensemble. As it stands, this result ranks between VGG-16 [43] (87.77%) and ResNet-18 [2] (90.27%), two popular hand-crafted models. Importantly, the ranking of the architectures is entirely different between the two tasks. Therefore, reusing the best model found on CIFAR-10 would have led to worse results: the population of "antibodies" improved the solution's transferability.

V. Conclusion

With ImmuNES, we show that an AIS is able to competently perform NES at the macro-architecture level, when provided with relevant efficiency improvements. It is able to reach competitive results on two very different search spaces without complex pre- and post-processing engineering or training techniques. We also show that the AIS brings significant benefits over two baselines when it comes to ensembling. Finally, and unlike most macro-architecture NAS method, the multiple nature of ImmuNES' solutions allows them to naturally transfer to a harder task. Even though our experiments do not reach the SOTA on CIFAR-10, we believe this approach to be

[3]If a different number of classes was required, we would only have had to modify the fully connected layer in each of the classifier blocks.

promising as a future alternative to micro-architecture search methods, without many of their drawbacks. Future research directions include further improving our method's efficiency by implementing performance prediction to guide the search, and exploring more advanced AIS algorithms.

REFERENCES

[1] L. N. de Castro and F. J. Von Zuben, "Learning and Optimization Using the Clonal Selection Principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.

[2] 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*, 2016, pp. 770–778.

[3] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[4] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *International Conference on Learning Representations*, 2017.

[5] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. JMLR.org, 2017, pp. 2902–2911.

[6] J. Wang, J. Xu, and X. Wang, "Combination of Hyperband and Bayesian Optimization for Hyperparameter Optimization in Deep Learning," *arXiv preprint arXiv:1801.01596*, 2018.

[7] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search," *arXiv preprint arXiv:1910.11858*, 2019.

[8] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," in *International Conference on Machine Learning*, 2018, pp. 4092–4101.

[9] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," *arXiv preprint arXiv:1806.09055*, 2018.

[10] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "XNAS: Neural Architecture Search with Expert Advice," in *Advances in Neural Information Processing Systems*, 2019, pp. 1977–1987.

[11] L. Li and A. Talwalkar, "Random Search and Reproducibility for Neural Architecture Search," *arXiv preprint arXiv:1902.07638*, 2019.

[12] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the Search Phase of Neural Architecture Search," *arXiv preprint arXiv:1902.08142*, 2019.

[13] A. Yang, P. M. Esperança, and F. M. Carlucci, "NAS Evaluation is Frustratingly Hard," *arXiv preprint arXiv:1912.12522*, 2019.

[14] L. Xie and A. Yuille, "Genetic CNN," *arXiv preprint arXiv:1703.01513*, 2017.

[15] 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, 2019, pp. 4780–4789.

[16] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of Deep Convolutional Neural Networks using Cartesian Genetic Programming," *Evolutionary Computation*, vol. 28, no. 1, pp. 141–163, 2020.

[17] E. Byla and W. Pang, "DeepSwarm: Optimising Convolutional Neural Networks using Swarm Intelligence," *arXiv preprint arXiv:1905.07350*, 2019.

[18] V. Macko, C. Weill, H. Mazzawi, and J. Gonzalvo, "Improving Neural Architecture Search Image Classifiers via Ensemble Learning," *arXiv preprint arXiv:1903.06236*, 2019.

[19] S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. W. Teh, "Neural Ensemble Search for Performant and Calibrated Predictions," *arXiv preprint arXiv:2006.08573*, 2020.

[20] L. N. de Castro and J. Timmis, "An Artificial Immune Network for Multimodal Function Optimization," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1, 2002, pp. 699–704.

[21] R. Pasti, L. N. de Castro, G. P. Coelho, and F. J. V. Zuben, "Neural Network Ensembles: Immune-Inspired Approaches to the Diversity of Components," *Natural Computing*, vol. 9, no. 3, pp. 625–653, 09/01 2010, iD: Pasti2010. [Online]. Available: https://doi.org/10.1007/s11047-009-9124-1

[22] 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*, 2018, pp. 8697–8710.

[23] H. Hu, J. Langford, R. Caruana, E. Horvitz, and D. Dey, "Macro Neural Architecture Search Revisited," in *2nd Workshop on Meta-Learning at NeurIPS*, 2018.

[24] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional Neural Network Committees for Handwritten Character Classification," in *International Conference on Document Analysis and Recognition (ICDAR), 2011*, 2011, pp. 1135–1139.

[25] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter Optimization for Convolutional Neural Network Committees Based on Evolutionary Algorithms," in *Proceedings of the 24th IEEE International Conference on Image Processing*, 2018, pp. 3924–3928.

[26] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." in *AAAI*, vol. 4, 2017, p. 12.

[27] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[28] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," *Technical Report, University of Toronto*, vol. 1, no. 4, 2009.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-level Performance on ImageNet Classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[30] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning Augmentation Policies from Data," *arXiv preprint arXiv:1805.09501*, 2018.

[31] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[32] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[33] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-Deep Neural Networks without Residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[34] 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*, 2015, pp. 1–9.

[35] H. Jin, Q. Song, and X. Hu, "Auto-Keras: An Efficient Neural Architecture Search System," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1946–1956.

[36] T. Elsken, J. H. Metzen, and F. Hutter, "Simple and Efficient Architecture Search for Convolutional Neural Networks," *arXiv preprint arXiv:1711.04528*, 2017.

[37] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "Gradient Based Evolution to Optimize the Structure of Convolutional Neural Networks," in *25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 3438–3442.

[38] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2020.

[39] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[40] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware Progressive Search for Pareto-optimal Neural Architectures," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[41] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020.

[42] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "CINIC-10 is not ImageNet or CIFAR-10," *arXiv preprint arXiv:1810.03505*, 2018.

[43] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.