Evolving Neural Architecture Using One Shot Model

Nilotpal Sinha
National Chiao Tung University
National Yang Ming Chiao Tung University
Hsinchu 30010, Taiwan
nilotpalsinha.cs06g@nctu.edu.tw

Kuan-Wen Chen
National Chiao Tung University
National Yang Ming Chiao Tung University
Hsinchu 30010, Taiwan
kuanwen@cs.nctu.edu.tw

ABSTRACT

Previous evolution based architecture search require high computational resources resulting in large search time. In this work, we propose a novel way of applying a simple genetic algorithm to the neural architecture search problem called EvNAS *(Evolving Neural Architecture using One Shot Model)* which reduces the search time significantly while still achieving better result than previous evolution based methods. The architectures are represented by architecture parameter of one shot model which results in the weight sharing among the given population of architectures and also weight inheritance from one generation to the next generation of architectures. We use the accuracy of partially trained architecture on validation data as a prediction of its fitness to reduce the search time. We also propose a decoding technique for the architecture parameter which is used to divert majority of the gradient information towards the given architecture and is also used for improving the fitness prediction of the given architecture from the one shot model during the search process. EvNAS searches for architecture on CIFAR-10 for 3.83 GPU day on a single GPU with top-1 test error 2.47%, which is then transferred to CIFAR-100 and ImageNet achieving top-1 error 16.37% and top-5 error 7.4% respectively.

CCS CONCEPTS

• **Computing methodologies** → **Bio-inspired approaches**; **Genetic algorithms**.

KEYWORDS

One shot model, genetic algorithm, decoded architecture parameter

ACM Reference Format:
Nilotpal Sinha and Kuan-Wen Chen. 2021. Evolving Neural Architecture Using One Shot Model. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459275

1 INTRODUCTION

Convolutional neural networks have been instrumental in solving various problems in the field of computer vision. However, the network designs were mainly done by humans (like AlexNet [16], ResNet [12], DenseNet [14], VGGNet [29]) on the basis of their intuition and understanding of the specific problem. This has led to the growing interest in the automated search of neural architecture called *Neural Architecture Search* (NAS) [11][36][26]. NAS has shown some promising results in the field of computer vision but most of these methods demand a considerable amount of computational power. For example, obtaining the state-of-the-art architecture for CIFAR-10 required 3150 GPU days of evolution [27] and 1800 GPU days of reinforcement learning (RL) [37]. This is due to the evaluation of the architectures during the search process of the different NAS methods because most NAS methods train each architecture individually for certain number of epochs in order to evaluate its performance on the validation data. Recent works [26] [1] [21] [8] [2] [7] have reduced the search time by sharing weights among the architectures.

In this work, we propose a method called EvNAS (*Evolving Neural Architecture using One Shot Model*) which involves evolving a convolutional neural network architecture with weight sharing among the architectures for image classification task. Instead of training each architecture from scratch for a certain number of epochs in order to estimate its fitness, EvNAS trains a single one shot model and uses it to estimate the fitness of an architecture. The one shot model enables weights sharing among the architectures resulting in 1) child architecture inheriting weights from parents instead of retraining from scratch and 2) architectures in a given generation also sharing weights among them, which lead to reduction in search time. The work is inspired by the representation used for the architectures of the network in DARTS [21], which relaxes the search space to a continuous space in order to use gradient descent for optimizing the architecture. But these gradient based methods [21] [34] [2] are highly dependent on the search space and they tend to overfit to operations in the search space that lead to faster gradient descent. By replacing the architecture search using gradient descent with architecture search using evolution, we leverage the speedup introduced due to the weight sharing among the architectures while resolving the overfitting problem due to the stochastic nature of the evolution based methods. Our experiments (Section 4), show that EvNAS outperforms previous evolution based methods while using minimal computational resources (3.83 GPU day on a single GPU) as compared to previous evolution based methods.

Our contributions can be summarized as follows:

- We introduce an efficient method of applying a simple genetic algorithm to the NAS problem with reduced computational requirements by using one shot model.

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459275

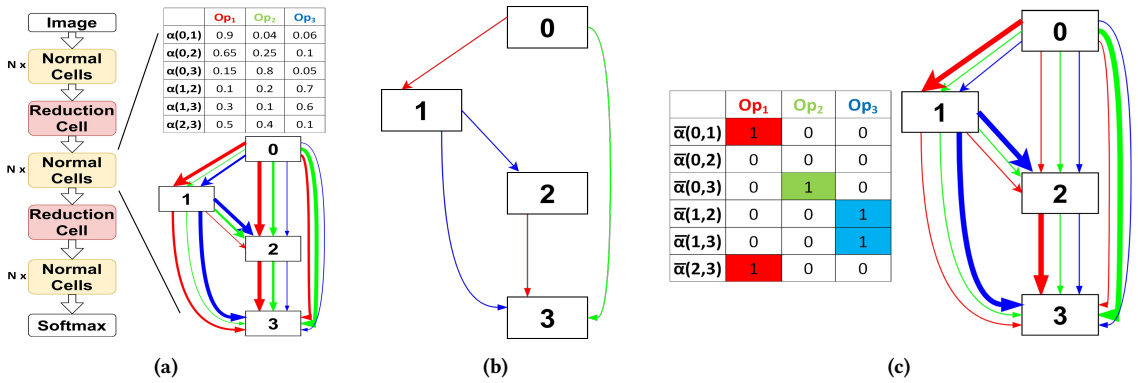


Figure 1: The process of decoding the architecture parameter, α. Better viewed in color mode. Here, we consider three operations in the operation space. (a) One shot model and its representation with arrows between the nodes representing all the operations in the search space, (b) Discrete architecture, $arch_{dis}$, derived from α, (c) Decoded architecture, $\bar{\alpha}$, created using $arch_{dis}$. The thickness of the arrow is proportional to the weight given to an operation.

- We propose a decoding technique for each architecture in the population which diverts a majority of the gradient information to the current architecture during the training phase and is used to calculate the fitness of the architecture from the one shot model during the fitness evaluation phase.
- We propose a crossover operation that is guided by the predicted fitness of the partially trained architectures of the previous generation and does not require keeping track of the ancestors of the parent architectures.
- We achieved remarkable efficiency in the architecture search achieving test error of 2.47% with 3.63M parameters on CIFAR-10 in search time significantly less than previous evolution based methods and showed that the architecture learned by EvNAS is transferable to CIFAR-100 and ImageNet.

2 RELATED WORK

Automated Neural Architecture Search is an alternative to the handcrafted architectures where the machine designs the best suited architecture for a specific problem. Several search methods have been proposed to explore the space of neural architectures, such as evolutionary algorithm (EA) [27][28][19][32], reinforcement learning (RL) [36][37][26], random search[17] and gradient-based methods [20][21][25][2][34]. These can be grouped into two groups: *gradient-based* methods and *non-gradient based* methods.

Gradient Based Methods: In these methods, the neural architecture is directly optimized using the gradient information based on the performance on the validation data. In [20][21], the discrete architecture search space is relaxed to a continuous search space by using a one shot model and the performance of the model on the validation data is used for updating the architecture using gradients. This method reduces the search time significantly but suffers from the overfitting problem wherein the searched architecture performs very well on the validation data but exhibits poor performance on the test data. This is mainly attributed to the preference of parameter-less operations during the search process as it leads to a rapid gradient descent [2]. Many regularizations have been introduced to tackle the problem such as early stopping [34], search space regularization [2] and architecture refinement [2]. Contrary to the gradient based methods, the proposed method does not suffer from the overfitting problem because of the stochastic nature introduced by the mutation operation.

Non-Gradient Based Methods: These methods include reinforcement learning (RL) and evolutionary algorithm (EA). In RL methods, an agent is trained to generate a neural architecture through its action in order to maximize the expected accuracy on the validation data. In [36][37], a recurrent neural network (RNN) is used as an agent which samples neural architectures which are then trained to convergence in order to obtain their accuracies on the validation data. These accuracies are then used to update the weights of RNN by using policy gradient methods. Both of these methods suffered from huge computational requirements. This was improved upon in [26], where all the sampled architectures were forced to share weights by using a single directed acyclic graph (DAG) resulting in the reduction of computational resources. Early approaches based on EA such as [31][30] optimized both the neural architectures and the weights of the network which limited their usage to relatively smaller networks. Then, methods such as [32][27] used evolution to search for the architecture and gradient descent for optimizing the weights of each architecture which made it possible to search for relatively large networks. However, this resulted in huge computational requirements. To speed up the training of each individual architecture, *weight inheritance* was introduced in [28] wherein a child network inherits the parent networks' weights. In this work, we have used both weight inheritance and weight sharing among the architectures to speed up the search process. FairNAS [4], NSGANetV2 [22] has also proposed evolutionary search with weight sharing which has two steps for searching neural architecture where they optimizes the supernet in the first step and then FairNAS performs architecture search using evolutionary method with the trained supernet as the evaluator in the second step while NSGANetV2 uses the weights from trained supernet to initialize weights of an architecture and train it using gradient descent for some epochs to evalauates its fitness during

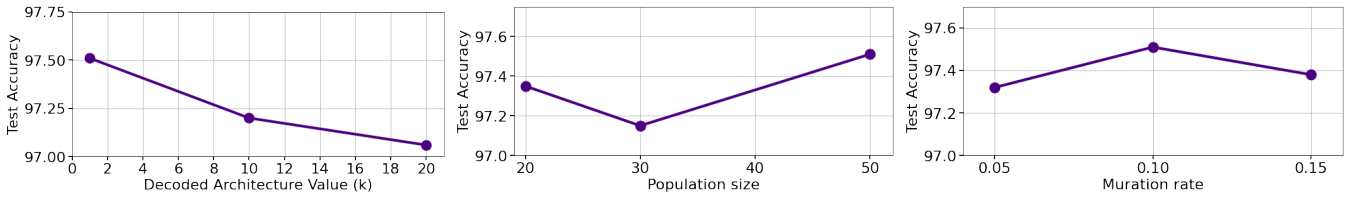


Figure 2: (a) Test accuracy vs Decoded architecture value (k), (b) Test accuracy vs Population size, (c) Test accuracy vs mutation rate. All the tests are done in Search Space S1.

the architecture search. In contrast, our method combines both the training and search process in one single stage.

3 METHODS

This section discusses different parts of the proposed algorithm and its relationship to prior works.

3.1 Representation of Architecture

The proposed algorithm deals with a population of architectures in each generation during the search process. Instead of having a separate model for each architecture in a population [32][27], we used a *one shot model* which treats all the architectures as subgraphs of the supergraph while sharing the weights among all the architectures. The one shot model is composed of repeatable cells which are stacked together to form the convolutional network. The one shot model has two types of convolutional cells: *normal* cell and *reduction* cell. A normal cell uses operations with stride 1 whereas reduction cell uses operations with stride 2. A cell in the one shot model is represented by the parameter, α called *architecture parameter*, which represents the weights of the different operations $op(.)$ in the operation space O (i.e. search space of NAS) between a pair of nodes. The edge between node i and node j can be written as:

$$f^{(i,j)}(x) = \sum_{op \in O} \frac{exp(\alpha_{op}^{i,j})}{\sum_{op' \in O} exp(\alpha_{op'}^{i,j})} op(x). \quad (1)$$

Where $\alpha_{op}^{i,j}$ refers to the weight of the operation op in the operation space O between node i and node j. The architecture is represented by two matrices, one for normal cell and one for reduction cell, where the row represents the edge between two nodes and the column represents the weights of different operations from the operation space as shown in Figure 1(a). Please refer to the original DARTS paper [21] for more technical details. The design choice results in weight sharing among the architectures in a given population of architectures. It also results in weight inheritance from one generation of architectures to the next generation of architectures i.e. the next generation architectures are not trained from scratch but inherit the partially trained weights from the previous generation architectures. All of these ultimately leads to the reduction of the architecture search time using evolution.

3.2 Decoding Architecture Parameter

Architecture parameter, α, gives variable weights to the operations in any particular architecture which results in very noisy estimate of fitness of the architecture. This results in the algorithm performing marginally better than the random algorithm as discussed in Section 4.4. We propose a decoding technique, which is a process of giving equal higher weight to the operations of the actual architecture/subgraph according to the architecture parameter, α and equal smaller weights to the operations of the other architectures. This can be thought of as decoding/mapping the genotype, i.e. α, to the phenotype, i.e. actual architecture [10]. The process has the following two steps:

- For any α, derive the discrete architecture, $arch_{dis}$, from α as shown in Figure 1(b).
- On the basis of the discrete architecture, $arch_{dis}$, create another architecture parameter called *decoded architecture* parameter , $\bar{\alpha}$ (as shown in Figure 1(c)), with the following entries:

$$\bar{\alpha}_{op}^{i,j} = \begin{cases} k, \text{if } op \text{ between node } i \text{ and } j \text{ present in } arch_{dis} \\ 0, \text{otherwise} \end{cases} \quad (2)$$

where k is an integer. The design ensures that the current architecture according to α gets a majority of the gradient information to update its parameters while the rest of the gradient information is distributed equally among all the other architectures to update their parameters. This results in making sure that the weights of an architecture does not get co-dependent with the weights of the other architecture due to the weight sharing nature of the one shot model. It also helps in improving the estimation of the *fitness* of each architecture in the population, as it gives higher equal weight to that particular architecture operations while giving lower equal weights to the other architecture operations. This results in the higher contribution from a particular architecture while very low contribution by other architectures during the fitness evaluation step of that particular architecture from the one shot model which also leads to a consistent estimated fitness for an architecture which might arise from different values of α in a given generation. This is in contrast to the variable architecture contribution, used in the original DARTS paper, wherein an architecture is evaluated using α which results in very noisy estimate of its performance. We empirically found that k = 1 gives a good result and increasing the value of k from 1 tends to deteriorate the accuracy as shown in Figure 2(a).

3.3 Training and Performance Estimation

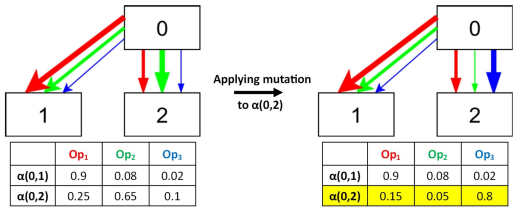
The sharing of the network weights among the architectures in the population, due to the one shot model representation [21], helps in

exchanging information to the next generation population, wherein the architectures of the next generation do not start training from scratch. This can be thought of as child architecture model inheriting the weights of the parent architecture model, also known as *weight inheritance*. Therefore, instead of the full training of each architecture in the population from scratch, EvNAS partially trains the inherited architecture model weights by using the training data. This is done by first copying the *decoded architecture* parameter, $\bar{\alpha}$, in Section 3.2, for the individual architecture in the population to the one shot model and then training the network for a certain number of batches of training examples.

To evaluate the performance of each individual architecture, its *decoded architecture* parameter, $\bar{\alpha}$, from Section 3.2, is first copied to the one shot model. The model is then evaluated on the basis of its accuracy on the validation data, which becomes the *fitness* of the architecture. Note that the fitness value of each architecture is a noisy estimate of its true accuracy on the validation data as the architecture has been trained partially on a certain number of training batches while inheriting its weights from the previous generation.

3.4 Evolutionary Algorithm

The evolutionary algorithm (EA) starts with a population of architectures, which are sampled from a uniform distribution on the interval $[0, 1)$, and it runs for G generations. In each generation, the one shot model is trained on the training data by using the decoded architecture parameter $\bar{\alpha}$ of each individual architecture in the population in a round-robin fashion. Then, the fitness of each individual architecture is estimated using the decoded architecture parameter $\bar{\alpha}$. The population is then evolved using crossover and mutation operations to create the next generation population replacing the previous generation population. The best architecture in each generation does not undergo any modification and is automatically copied to the next generation. This ensures that the algorithm does not forget the best architecture learned thus far and gives an opportunity to old generation architecture to compete against the new generation architecture; this is known as *elitism*. The best architecture is returned after G generations. The entire process is summarized in Algorithm 1.



	Op_1	Op_2	Op_3
α(0,1)	0.9	0.08	0.02
α(0,2)	0.25	0.65	0.1

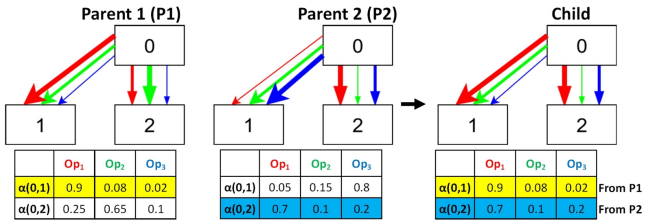
	Op_1	Op_2	Op_3
α(0,1)	0.9	0.08	0.02
α(0,2)	0.15	0.05	0.8

Figure 3: Illustration of mutation operation

Mutation Operation: It refers to a random change to an individual architecture in the population. The algorithm uses the *mutation operator* [10], which decides the probability of changing the architecture parameter, $\alpha^{i,j}$, between node i and node j. This is done by re-sampling $\alpha^{i,j}$ from a uniform distribution on the interval $[0, 1)$ as illustrated in Figure 3.

Algorithm 1 EvNAS

Input: population P, population size N, number of selected individuals in tournament selection T, mutation rate r, total number of training batches B, one shot model M.
for g = 1, 2, ..., G generations **do**
  for i = 1,..., B **do**
    Copy $\bar{\alpha}[i \mod N]$ generated from $\alpha[i \mod N]$ to M and train M on training batch$[i]$;
  end for
  Calculate fitness of each individual architecture, α in P by copying the respective $\bar{\alpha}$ to M;
  Set Elite, $E \leftarrow$ best architecture in P;
  Copy E to the next generation population, P_{next};
  for i = 2,.., N **do**
    Use *Tournament Selection* for selecting 2 parents;
    Use *crossover* to create new child from the 2 parents for P_{next};
    for each edge in *child* **do**
      if $uniformRandom(0, 1) \leq r$ **then**
        Apply *mutation* operation to the edge;
      end if
    end for
  end for
  $P \leftarrow P_{next}$;
end for
return Elite, E



	Op_1	Op_2	Op_3
α(0,1)	0.9	0.08	0.02
α(0,2)	0.25	0.65	0.1

	Op_1	Op_2	Op_3
α(0,1)	0.05	0.15	0.8
α(0,2)	0.7	0.1	0.2

	Op_1	Op_2	Op_3	
α(0,1)	0.9	0.08	0.02	From P1
α(0,2)	0.7	0.1	0.2	From P2

Figure 4: Illustration of crossover operation

Crossover Operation: It is a process of combining parent architectures to create a new child architecture, which may perform better than the parents. EvNAS uses *tournament selection* [10] for the parent selection process to generate the next generation architecture population. In *tournament selection*, a certain number of architectures are randomly selected from the current population and the most fit architecture from the selected group becomes the parent. We get *parent1* and *parent2* on applying tournament selection two times which are then used to create a single child architecture. This is done by copying the architecture parameters, $[\alpha^{i,j}]_{parent1}$ and $[\alpha^{i,j}]_{parent2}$ between node i and node j, from *parent1* and *parent2*, respectively, with a certain probability to the child architecture parameter, $[\alpha^{i,j}]_{child}$ between node i and node j as illustrated in Figure 4. This can be formulated as follows:

$$[\alpha^{i,j}]_{child} = \begin{cases} [\alpha^{i,j}]_{parent1}, \text{with probability } 0.5 \\ [\alpha^{i,j}]_{parent2}, \text{otherwise} \end{cases} \quad (3)$$

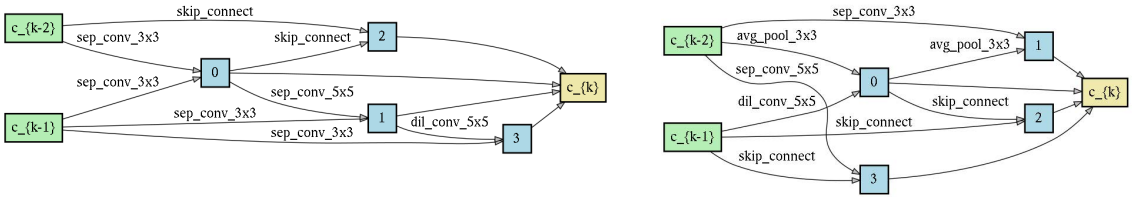


Figure 5: Discovered cell using EvNAS-A (a) Normal Cell (b) Reduction Cell

Table 1: Comparison of EvNAS with other NAS methods on CIFAR-10 and CIFAR-100 datasets for S1 in terms of test error (lower is better). The first block presents the performance of the hand-crafted architecture. The second block presents the performance of other NAS methods, the third block presents the performance of our method and the last block presents the performance of our ablation study. All the architecture search were performed using cutout. Results with † were NOT trained with Cutout [6].

Architecture	Test Error (%) C10	C100	Params (M)	Search Time (GPU Days)	Search Method
DenseNet-BC [14]	3.46	17.18	25.6	-	manual
PNAS [18]	3.41	-	3.2	225	SMBO
NASNet-A [37]	2.65	-	3.3	1800	RL
ENAS [26]	2.86	-	4.6	0.45	RL
DARTS [21]	2.76 ± 0.09	17.54	3.3	4	gradient-based
GDAS [8]	2.93	18.38	3.4	0.83	gradient-based
SNAS [33]	2.85 ± 0.02	-	2.8	1.5	gradient-based
SETN [7]	2.69	17.25	4.6	1.8	gradient-based
PDARTS [2]	2.50	16.55	3.4	0.3	gradient-based
AmoebaNet-A [27]	3.34 ± 0.06	18.93	3.2	3150	evolution
Large-scale Evolution† [28]	5.4	-	5.4	2750	evolution
Hierarchical Evolution† [19]	3.75	-	15.7	300	evolution
NSGANetV1-A2 [24]	2.65	17.42	0.9	27	evolution
NSGA-NET [23]	2.75	-	3.3	4	evolution
RSPS[17]	2.86 ± 0.08	-	4.3	2.7	random
EvNAS-A (Ours)	**2.47±0.06**	**16.37**	**3.6**	**3.83**	**evolution**
EvNAS-B (Ours)	2.62 ± 0.06	16.51	3.8	3.83	evolution
EvNAS-C (Ours)	2.63 ± 0.05	16.86	3.4	3.83	evolution
EvNAS-Rand (Ours)	2.84 ± 0.08	-	2.7	0.62	random
EvNAS-ND (Ours)	2.78 ± 0.1	-	3.8	3.83	evolution
EvNAS-NDF (Ours)	2.75 ± 0.09	-	3.1	3.83	evolution
EvNAS-NDT (Ours)	2.67 ± 0.06	-	3.5	3.83	evolution
EvNAS-Mut (Ours)	2.79 ± 0.06	-	3.4	3.83	evolution
EvNAS-Cross (Ours)	2.81 ± 0.08	-	3.2	3.83	evolution

Note that as all the architectures are sub-graph of the super-graph, i.e. the one shot model, so, we do not have to keep track of the ancestors [35][31] in order to apply the crossover operation.

4 EXPERIMENTS AND RESULTS

In this section, we report the performance of EvNAS in terms of a neural architecture search on two different search spaces: 1) Search space 1 (**S1**) and 2) Search space 2 (**S2**). The operations considered for the cells in **S1** and **S2** are given in supplementary. The one shot model is created by stacking *normal* cell with *reduction* cell inserted at 1/3 and 2/3 of the total depth of the model (Figure 1(a)).

We then present an ablation study showing the importance of the proposed *decoded architecture* parameter, $\bar{\alpha}$, crossover and mutation operations during the search process.

Initialization: Each architecture in a population is represented by the architecture parameter, α, which is sampled from a uniform distribution on the interval $[0, 1)$.

4.1 Dataset:

CIFAR-10 and **CIFAR-100** [15] has 50K training images and 10K testing images with images classified into 10 classes and 100 classes respectively. **ImageNet** [5] is well known benchmark for image

Table 2: Comparison of EvNAS with other NAS methods on ImageNet in mobile setting for S1 in terms of test error (lower is better). The first block presents the performance of the hand-crafted architecture. The second block presents the performance of other NAS methods and the last block presents the performance of our method.

Architecture	Test Error (%) top 1	top 5	Params (M)	+× (M)	Search Time (GPU Days)	Search Method
MobileNet [13]	29.4	10.5	4.2	569	-	manual
PNAS [18]	25.8	8.1	5.1	588	225	SMBO
NASNet-A [37]	26.0	8.4	5.3	564	1800	RL
NASNet-B [37]	27.2	8.7	5.3	488	1800	RL
NASNet-C [37]	27.5	9.0	4.9	558	1800	RL
DARTS [21]	26.7	8.7	4.7	574	4	gradient-based
GDAS [8]	26.0	8.5	5.3	581	0.83	gradient-based
SNAS [33]	27.3	9.2	4.3	522	1.5	gradient-based
SETN [7]	25.7	8.0	5.4	599	1.8	gradient-based
PDARTS [2]	24.4	7.4	4.9	557	0.3	gradient-based
AmoebaNet-A [27]	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B [27]	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C [27]	24.3	7.6	6.4	570	3150	evolution
NSGANetV1-A2 [24]	25.5	8.0	4.1	466	27	evolution
FairNAS-A [4]	24.7	7.6	4.6	388	12	evolution
FairNAS-B [4]	24.9	7.7	4.5	345	12	evolution
FairNAS-C [4]	25.3	7.9	4.4	321	12	evolution
EvNAS-A (Ours)	**24.4**	**7.4**	**5.1**	**570**	**3.83**	**evolution**
EvNAS-B (Ours)	24.4	7.4	5.3	599	3.83	evolution
EvNAS-C (Ours)	25.1	7.8	4.9	547	3.83	evolution

classification containing 1K classes with 1.28 million training images and 50K images test images. **ImageNet-16-120** [3] is a downsampled variant of ImageNet where the original ImageNet is downsampled to 16x16 pixels with labels $\in [0, 120]$ to construct ImageNet-16-120 dataset. Settings used for datasets in **S1** are similar to those used in [21] and datasets in **S2** are similar to those used in [9]. These are summarized in the supplementary.

4.2 Search Space 1 (S1)

Search space (S1) is the one used in DARTS [21] where we search for both normal and reduction cells in Figure 1(a). Here, each cell has seven nodes with first two nodes being the output from previous cells and last node as output node, resulting in 14 edges among them. There are 8 operation in S1 , so each architecture is represented by two 14x8 matrices, one each for normal cell and reduction cell.

Search Process: We perform the architecture search process on CIFAR-10 in **S1**, which is divided into two stages as was done in [21][17]. In *stage 1*, we perform the search process for both normal and reduction cells on CIFAR-10 by using four different seeds; this can be thought of as the searching stage of the algorithm. In *stage 2*, the architecture found in each trial of stage 1 is evaluated by retraining a larger network created using the same cell blocks discovered in stage 1 for 600 epochs from scratch on CIFAR-10. Next, we choose the best performing architecture among the four trials, making it the final architecture searched by the algorithm. To compare with other architecture search methods, we evaluate the final architecture found from stage 2 by training the network from scratch with ten different seeds for 600 epochs.

Architecture Search Settings: The training setting mainly follows the setup proposed by DARTS [21]. Because of the high memory requirements of the one shot model, a smaller network, called *proxy network* [17], with 8 stacked cells and 16 initial channels is used during the architecture search process, i.e. *stage 1*. For deriving the discrete architecture, $arch_{dis}$, each node in the discrete architecture is connected to two nodes among the previous nodes selected via the top-2 operations according to the architecture parameter α. For training the one shot model, we use similar settings as DARTS [21] and are summarized in supplementary. For our evolutionary algorithm, we use a population size of 50 in each generation, 0.1 as the *mutation rate* and 10 architectures are chosen randomly during the tournament selection. The search process runs for 50 generations on a single GPU, NVIDIA 2080 Ti, where each trial takes 0.95 day to complete and thus, taking 3.83 days to complete stage 1. Number of generations was chosen to match the number of epochs in DARTS [21]. Population size was chosen based on the experiments where we ran our method for population size of 20, 30, 50 with tournament size chosen as one-fifth of the population size, as shown in Figure 2(b). We did not go beyond 50 population size as we wanted to have search time similar to that of DARTS. Mutation rate was chosen based on the experiments where we ran our method for *mutation rate* of 0.05, 0.1, 0.15, as shown in Figure 2(c). All our architecture search in Table 1 are done with cutout [6].

Architecture Evaluation: A larger network, called *proxyless network* [17], with 20 stacked cells and 36 initial channels is used during the selection and evaluation stage. The proxyless network is trained using the settings in DARTS [21], summarized in supplementary. The architecture discovered in the search process on

Table 3: Comparison of EvNAS with other weight sharing based NAS methods on NAS-Bench-201 (S2) [9] with mean ± std. accuracies on CIFAR-10, CIFAR-100, ImageNet16-120 from 3 runs each (higher is better). Optimal refers to the best architecture accuracy for each dataset. Search times are given for a CIFAR-10 search on a single GPU.

Method	Search	CIFAR-10		CIFAR-100		ImageNet-16-120		Search
	(seconds)	*validation*	*test*	*validation*	*test*	*validation*	*test*	**Method**
RSPS [17]	7587.12	84.16 ± 1.69	87.66 ± 1.69	59.00 ± 4.60	58.33 ± 4.64	31.56 ± 3.28	31.14 ± 3.88	random
DARTS-V1 [21]	10889.87	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00	gradient-based
DARTS-V2 [21]	29901.67	39.77 ± 0.00	54.30 ± 0.00	15.03 ± 0.00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00	gradient-based
GDAS [8]	28925.91	90.00 ± 0.21	93.51 ± 0.13	71.14 ± 0.27	70.61 ± 0.26	41.70 ± 1.26	41.84 ± 0.90	gradient-based
SETN[7]	31009.81	82.25 ± 5.17	86.19 ± 4.63	56.86 ± 7.59	56.87 ± 7.77	32.54 ± 3.63	31.90 ± 4.07	gradient-based
ENAS [26]	13314.51	39.77±0.00	54.30 ± 0.00	15.03 ± 00	15.61 ± 0.00	16.43 ± 0.00	16.32 ± 0.00	RL
EvNAS (Ours)	**22444.78**	**88.98±1.40**	**92.18±1.11**	**66.35±2.59**	**66.74±3.08**	**39.61±0.72**	**39.00±0.44**	**evolution**
Optimal	N/A	91.61	94.37	73.49	73.51	46.77	47.31	N/A

CIFAR-10 is then trained on both CIFAR-100 dataset [15] and ImageNet dataset. For CIFAR-100, the same settings as CIFAR-10 is used to train and evaluate the proxyless network on the CIFAR-100 dataset [15]. For ImageNet, we train a network with 14 cells and 48 initial channels in the mobile setting, where the size of the input images is 224 x 224 and the number of multiply-add operations in the model is restricted to less than 600M. We follow the training settings used by PDARTS [2] on 8 NVIDIA V100 GPUs.

Search Results and Transferability to CIFAR-100 and ImageNet: We perform the architecture search on CIFAR-10 three times with different random number seeds and their results are provided in Table 1 as EvNAS-A, EvNAS-B and EvNAS-C, which are then transferred to CIFAR-100. The cells discovered during EvNAS-A are shown in Figure 5 and those discovered by EvNAS-B and EvNAS-C are given in the supplementary. EvNAS-A evaluates 10K architectures during the architecture search with search time significantly less than the previous evolution based methods [27] [28] [19] while achieving better result than them. Following [21][2], the discovered architecture from CIFAR-10 is then evaluated on the ImageNet dataset and the results are provided in Table 2. The result shows that the cell discovered by EvNAS on CIFAR-10 can be successfully transferred to the ImageNet, achieving a top-5 error of 7.4%. Notably, EvNAS is able to achieve better result than previous state-of-the-art evolution based methods [27] [4] while using significantly less computational resources.

4.3 Search Space 2 (S2)

Search space (S2) is a smaller search space and is the one used in NAS-Bench-201 [9], where we only search for normal cell in Figure 1(a). NAS-Bench-201 provides a unified benchmark for almost any up-to-date NAS algorithm by providing results of each architecture in the search space on CIFAR-10, CIFAR-100 and ImageNet-16-12. Here, each cell has four nodes with first node as input node and last node as output node, resulting in 6 edges among them. these are 5 operations in S2, so each architecture is represented by one 6x5 matrix for the normal cell.

Search Process: We apply EvNAS with the same setting as the one used for S1 using a single GPU, NVIDIA 2080 Ti . Following [9], we run EvNAS three times (i.e. the stage 1 of EvNAS) on three datasets: CIFAR-10, CIFAR-100 and ImageNet-16-120 and report the statistics of the test and validation accuracies for each dataset.

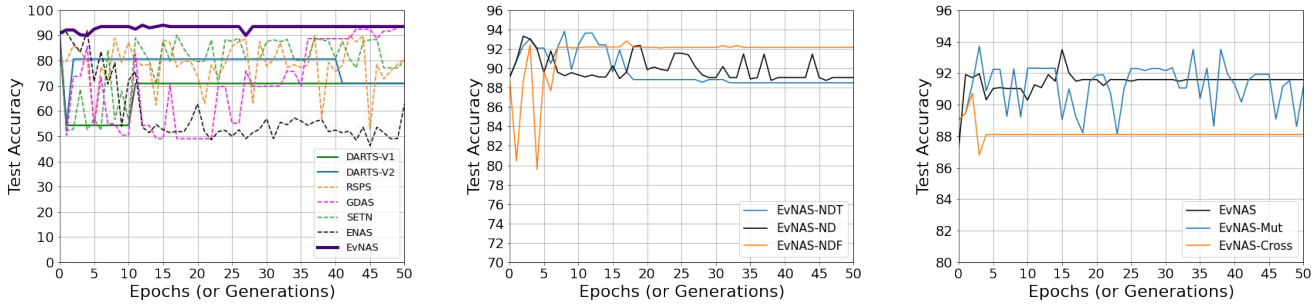
Search Results: The search results of applying EvNAS on S2, evaluated on CIFAR-10, CIFAR-100, ImageNet-16-120 are provided in Table 3. We found that EvNAS outperformed all weight sharing based methods except GDAS [8]. But GDAS performs worse when the search space increases as can be seen in Table 1 and Table 2. In Figure 6 (a), we show the performance of the architecture derived from each algorithm in Table 3 for every epoch. We found that EvNAS converges to an optimal architecture much faster and does not get stuck in sub-optimal architectures like DARTS due to mutation.

4.4 Ablation Studies

To discover the effect of the *decoded architecture* parameter, $\bar{\alpha}$, and the crossover and mutation operations during the search process, we conduct the following architecture searches in both Search Space 1 (**S1**) and Search Space 2 (**S2**) on CIFAR-10: without *decoded architecture* parameter, $\bar{\alpha}$, with *crossover only*, with *mutation only* and without *crossover and mutation*. The search results for **S1** are provided in Table 1. To understand the dynamics during the architecture search, we also plot the test accuracy of derived architecture at each epoch for **S2** in Figure 6 (b)(c).

Without *Crossover and Mutation*: Here, a population of 50 architectures are randomly changed after every generation and in the last generation, the architecture with the best performance on the validation set is chosen as the best found architecture. Thus, the search process evaluates only 200 architectures to come up with the best architecture. On performing the search in **S1** (listed as *EvNAS-Rand* in Table 1), we found that *EvNAS-Rand* behaves as a *random search* and shows similar results to those reported in RSPS [17].

Without *Decoded Architecture* Parameter, $\bar{\alpha}$: Here, we conduct three architecture searches where a population of 50 architectures are modified through both crossover and mutation operations without using the decoded architecture parameter, $\bar{\alpha}$, (i) during the training (*EvNAS-NDT*), (ii) during the fitness evaluation of each individual architecture in the population (*EvNAS-NDF*) and (iii) during both training and fitness evaluation (*EvNAS-ND*). From Table 1, we found that both *EvNAS-NDT* and *EvNAS-NDF* perform better than *EvNAS-ND*, showing that the decoded architecture parameter, $\bar{\alpha}$, plays an important role during the architecture search. From Figure 6 (b), we found that *EvNAS-ND* shows some randomness



(a) (b) (c)

Figure 6: Test accuracy of derived architecture evaluated on CIFAR-10 at each epoch for S2. (a) Comparison of EvNAS with other weight sharing based NAS methods, (b) Comparison of EvNAS with decoded architecture parameter, $\bar{\alpha}$, during different phases of the architecture search, (c) Comparison of EvNAS with crossover and mutation only.

due to the noise introduced in the fitness estimation. *EvNAS-NDT* smooths out the randomness in *EvNAS-ND* due to the presence of $\bar{\alpha}$ during fitness estimation. *EvNAS-NDF* shows that $\bar{\alpha}$ during training is responsible for the fast convergence of EvNAS.

With *Mutation Only*: Here, a population of 50 architectures are modified only through a mutation operation with 0.1 as the *mutation rate* while using the decoded architecture parameter. On performing the search in **S1** (listed as *EvNAS-Mut* in Table 1), we found that *EvNAS-Mut* performs slightly better than that of the *random search* even though mutation is a random process. From Figure 6 (c), we find that *mutation* behaves more like a random process and is the exploration part of EvNAS which does not let EvNAS get stuck in local optima.

With *Crossover Only*: Here, a population of 50 architectures are modified only through a crossover operation only while using the decoded architecture parameter. On performing the search in **S1** (listed as *EvNAS-Cross* in Table 1), we found that *EvNAS-Cross* performs slightly better than that of the *random search*. From Figure 6 (c), we find that *crossover* is the exploitation part of EvNAS and is prone to getting stuck in local optima. This can be attributed to the selection pressure [10] introduced due to the tournament selection. This improvement of *EvNAS-Cross* and *EvNAS-Mut* over the *random search* in **S1** can be attributed to *elitism*, which does not let the algorithm forget the best architecture learned thus far.

4.5 Discussion on Evolutionary Search vs Gradient Based Search

The gradient based methods are highly dependent on the search space and they tend to overfit to operations that lead to faster gradient descent which is the *skip-connect* operation due to its parameter-less nature, leading to higher number of *skip-connect* in the final discovered cell [34] [9] [2]. PDARTS [2] uses a regularization method to restrict the number of *skip-connect* to a specific number in the final normal cell for the search space S1. PDARTS emperically found that the optimal number of *skip-connect* in the normal cell is 2, which reduces the search space resulting in faster search time as compared to the original DARTS [21], which is a gradient based method without any regularization applied to the search space S1. This optimal number of *skip-connect* is a search space dependent value and thus, the same PDARTS method cannot be applied to the search space S2. Notice that without such regularization of restricting the number of skip-connect to 2 in S1, the gradient based methods, e.g. DARTS, only provides similar search time but worse performance than ours in both S1 and S2 due to the overfitting problem. In contrast, EvNAS does not have to worry about the overfitting problem due to its stochastic nature and so it is not dependent on the search space. EvNAS arrives at this optimal solution without any regularization being applied to the search space as can be seen in the discovered normal cells in Figure 5(a) and all the figures in the supplementary.

5 CONCLUSIONS AND FUTURE DIRECTIONS

This paper presented an efficient method of applying a simple genetic algorithm to the neural architecture search problem which involves weight sharing among the individual architectures and weight inheritance using a one shot model. This results in following practical benefits: 1) the use of one shot model reduces the computational requirements as compared to other EA based methods, 2) it solves the overfitting problem introduced in the gradient-based methods with the use of mutation. The paper also introduces a decoding method for the architecture parameter which is used to improve the fitness estimation of a partially trained individual architecture from the one shot model. Experimentally, EvNAS reduces the search time of evolution based architecture search significantly while achieving better results on CIFAR-10, CIFAR-100 and ImageNet datasets than previous evolutionary algorithms. The proposed method also outperforms most weight sharing based NAS methods in NAS-Bench-201 benchmark for NAS algorithms. All these show that the proposed method is search space agnostic.

ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Technology of Taiwan (MOST 108-2221-E-009-067-MY3 and MOST 110-2634-F-009-018-). Furthermore, we are grateful to the National Center for High-performance Computing for computer time and facilities.

REFERENCES

[1] Gabriel Bender. 2019. Understanding and simplifying one-shot architecture search. (2019).

[2] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2019. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*. 1294–1303.

[3] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819* (2017).

[4] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. 2019. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845* (2019).

[5] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[6] Terrance DeVries and Graham W Taylor. 2017. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552* (2017).

[7] Xuanyi Dong and Yi Yang. 2019. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision*. 3681–3690.

[8] Xuanyi Dong and Yi Yang. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*. 1761–1770.

[9] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJxyZkBKDr

[10] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.

[15] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

[17] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*. PMLR, 367–377.

[18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Fei-Fei Li, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[19] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJQRKzbA-

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1eYHoC5FX

[22] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*. Springer, 35–51.

[23] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 419–427.

[24] Zhichao Lu, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. Multi-objective evolutionary design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation* (2020).

[25] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Advances in Neural Information Processing Systems*. 7816–7827.

[26] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 4095–4104. http://proceedings.mlr.press/v80/pham18a.html

[27] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4780–4789.

[28] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2902–2911.

[29] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

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

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

[32] Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*. 1379–1388.

[33] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rylqooRqK7

[34] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2020. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1gDNyrKDS

[35] Hui Zhu, Zhulin An, Chuanguang Yang, Kaiqiang Xu, Erhu Zhao, and Yongjun Xu. 2019. EENA: efficient evolution of neural architecture. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 0–0.

[36] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[37] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.