Neural Architecture Search Using Covariance Matrix Adaptation Evolution Strategy

Nilotpal Sinha
Kuan-Wen Chen
Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu City, 30010, Taiwan

https://doi.org/10.1162/evco_a_00331

Abstract
Evolution-based neural architecture search methods have shown promising results, but they require high computational resources because these methods involve training each candidate architecture from scratch and then evaluating its fitness, which results in long search time. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) has shown promising results in tuning hyperparameters of neural networks but has not been used for neural architecture search. In this work, we propose a framework called CMANAS which applies the faster convergence property of CMA-ES to the deep neural architecture search problem. Instead of training each individual architecture separately, we used the accuracy of a trained one shot model (OSM) on the validation data as a prediction of the fitness of the architecture, resulting in reduced search time. We also used an architecture-fitness table (AF table) for keeping a record of the already evaluated architecture, thus further reducing the search time. The architectures are modeled using a normal distribution, which is updated using CMA-ES based on the fitness of the sampled population. Experimentally, CMANAS achieves better results than previous evolution-based methods while reducing the search time significantly. The effectiveness of CMANAS is shown on two different search spaces using four datasets: CIFAR-10, CIFAR-100, ImageNet, and ImageNet16-120. All the results show that CMANAS is a viable alternative to previous evolution-based methods and extends the application of CMA-ES to the deep neural architecture search field.

Keywords
Covariance matrix adaptation evolution strategy (CMA-ES), one shot model, neural architecture search, evolution strategies.

1 Introduction
In recent years, convolutional neural networks (CNNs) have been instrumental in solving various computer vision problems. However, the CNN architectures (such as AlexNet, Krizhevsky et al., 2012; ResNet, He et al., 2016; DenseNet, Huang et al., 2017; and VGGNet, Simonyan and Zisserman, 2014) have been designed mainly by humans, relying on their intuition and understanding of specific problem. Neural architecture search (NAS) tries to replace the reliance on human intuition with an automated search of the neural architecture. Recent NAS methods (Elsken et al., 2018b; Zoph and Le, 2016; Pham et al., 2018) have shown promising results in the field of computer vision, but most of these methods consume a huge amount of computational power. Any NAS method (Elsken et al., 2019) has three parts (see Figure 1): search space, search strategy, and performance estimation. The search space typically defines the type of architecture that can be represented in principle. The search strategy defines the process of how
to explore the search space. This typically includes reinforcement learning (RL)-based methods, evolutionary algorithm (EA)-based methods, and gradient-based methods. The performance estimation refers to the process of estimating the performance of a neural architecture. The objective of any NAS method is to find the architecture with high performance using the performance estimation.

Evolutionary algorithm (EA)-based NAS updates a population of architectures on the basis of performance of the architectures from the performance estimation process. Reinforcement learning (RL)-based NAS has a RL agent sampling architecture in the search space, which is updated depending on the performance of the architecture determined the performance estimation process. Both types of methods require huge computational resources, resulting in long search time. For example, the method proposed in Real et al. (2019) required 3,150 GPU days of evolution, and that discussed in Zoph et al. (2018) required 1,800 GPU days of RL search. This is attributed to the performance estimation process (Figure 1) wherein each architecture is trained from scratch for a certain number of epochs in order to evaluate its performance on the validation data. Recently proposed gradient-based methods such as Liu et al. (2019), Dong and Yang (2019b), Xie et al. (2019), Dong and Yang (2019a), and Chen et al. (2019) have reduced the search time by sharing weights among the architectures. However, these gradient-based methods are highly dependent on the given search space and suffer from premature convergence to the local optimum as shown in Chen et al. (2019) and Zela et al. (2020).

EA-based NAS algorithms (Sinha and Chen, 2021) solve the premature convergence problem of the gradient-based methods but require a large number of architecture evaluations to get a feasible solution, which ultimately leads to longer search time. Covariance matrix adaptation evolution strategy (CMA-ES) (Hansen and Ostermeier, 2001; Jin et al., 2002) has shown promising results in tuning hyperparameters of a neural network (Loshchilov and Hutter, 2016), but it required 30 GPUs to accomplish the task. However, CMA-ES has not been used in deep neural architecture search (NAS) problems. In this work, we propose a method called CMANAS (Neural Architecture using Covariance Matrix Adaptation Evolution Strategy, summarized in Figure 2), wherein we use the faster convergence property of CMA-ES in the NAS problem. Here, the neural architecture is represented by a normal distribution (discussed later in Section 3). In every generation, the distribution is first used to sample a population of architectures and then the distribution is updated using CMA-ES on the basis of the performance of the population of architectures. We used a trained one shot model (OSM) (discussed later in Section 3.3) to evaluate each architecture instead of training each architecture from scratch. This resulted in reduced search time because the one shot model shares weight among all the architectures in the search space. We also used an architecture-fitness table (AF table) to maintain a record of the already evaluated architectures, which further resulted in the reduction of the search time.

Our contributions could be summarized as follows:

- We designed a framework of applying the covariance matrix adaptation evolution strategy (CMA-ES) to the NAS problem where the architecture is
Neural Architecture Search using CMA-ES

Figure 2: Illustration of the general framework of CMANAS.

represented by a 2D matrix. The entries in the matrix select an architecture by giving higher weights to that architecture in the search space. The matrix is updated using CMA-ES with more details given in Section 3. Instead of training each architecture in the population from scratch, we used a trained one shot model (OSM) (a supergraph that treats all architectures as subgraphs) for evaluating the performance/fitness of an architecture, resulting in reduced computational requirements.

• We used an architecture-fitness table (AF table) for maintaining the records of the already evaluated architectures in order to skip the process of re-evaluating an already evaluated architecture and thus reducing the search time.

• We also used a NAS benchmark, NAS-Bench-201 (Dong and Yang, 2020), which provides the fitness value of each architecture in the search space. This allowed us to simulate the process of using our method without OSM and guide the search process by training and evaluating each architecture in the population from scratch.

• We also created a visualization of the architecture search performed by CMANAS to get insights into the search process. We found that the search begins with giving equal weights to all architectures in the search space and as the search progresses and converges to an architecture, CMANAS increases the weights to the converged architecture. We also found that the first phase of the search is predominantly an exploration phase wherein CMANAS explores the given search space. This is followed by an exploitation phase (i.e., convergence to an architecture).

The code for our paper can be found here: https://github.com/nightstorm0909/CMANAS.

2 Related Work

Searching the neural architecture automatically by using an algorithm (i.e., NAS) is an alternative to the architectures designed by humans, and in recent years, these NAS methods have attracted increasing interest because of its promise of an automatic and efficient search of architectures specific to a task. Early NAS approaches (Stanley and
Miikkulainen, 2002; Stanley et al., 2009), optimized both the neural architectures and the weights of the network using evolution. However, their usage was limited to shallow networks. Recent NAS methods (Zoph and Le, 2016; Pham et al., 2018; Real et al., 2019; Zoph et al., 2018; Real et al., 2017; Liu, Simonyan, et al., 2018; Xie and Yuille, 2017) perform the architecture search separately while using gradient descent for optimizing the weights of the architecture for its evaluation, which has made the search of deep networks possible. The various NAS methods can be classified into two categories on the basis of the different methods used in the search strategy in Figure 1. These are gradient-based methods and non-gradient-based methods.

**Gradient-Based Methods:** These methods begin with a random neural architecture, which is then updated using the gradient information on the basis of its performance on the validation data. In general, these methods (Liu et al., 2019; Dong and Yang, 2019b; Xie et al., 2019; Dong and Yang, 2019a), relax the discrete architecture search space to a continuous search space by using a one shot model (OSM). The performance of the OSM on the validation data is used for updating the architecture using gradients. As the OSM shares weights among all architectures in the search space, these methods take lesser time in the performance estimation process in Figure 1 and thus shorter search time. However, these methods suffer from the overfitting problem wherein the resultant architecture shows good performance on the validation data but exhibits poor performance on the test data. This can be attributed to its preference for parameterless operations in the search space, as it leads to rapid gradient descent (Chen et al., 2019). Some regularization techniques have been introduced to tackle this problem, such as early stopping (Zela et al., 2020), search space regularization (Chen et al., 2019) and architecture refinement (Chen et al., 2019). In contrast to these gradient-based methods, our method does not suffer from the overfitting problem because of its stochastic nature and does not need any regularization to arrive at a good solution.

**Non-Gradient-Based Methods:** These methods include reinforcement learning (RL) methods and evolutionary algorithm (EA) methods. In the RL methods, an agent is used for the generating neural architecture. The agent is then trained to generate architectures in order to maximize its expected accuracy on the validation data (calculated in the performance estimation process in Figure 1). In Zoph and Le (2016) and Zoph et al. (2018), a recurrent neural network (RNN) is used as an agent for sampling the neural architectures. These sampled architectures are then trained from scratch to convergence in order to get their accuracies on the validation data (i.e., performance estimation process in Figure 1). These accuracies are then used for updating the weights of the RNN agent by using policy gradient methods. Because of the huge computational requirement of training the architectures from scratch in the performance estimation process, both of these methods suffered from long search time. This was improved in Pham et al. (2018) by using a single directed acyclic graph (DAG) for sharing the weights among all the sampled architectures, thus resulting in reduced computational resources.

The EA-based NAS methods begin with a population of architectures, each of which is evaluated on the basis of its performance on the validation data (performance estimation process in Figure 1). The population is then evolved on the basis of the performance of the population. Methods such as those proposed in Real et al. (2019) and Xie and Yuille (2017) used gradient descent for optimizing the weights of each architecture in the population from scratch in order to determine their accuracies on the validation data as their fitness during the performance estimation process, resulting in huge computational requirements. In order to speed up the training process, Real et al. (2017)
introduced weight inheritance wherein the architectures in the next generation population inherit the weights of the previous generation population, resulting in bypassing the training from scratch. However, the speed up gained is less, as it still needs to optimize the weights of the architecture. Methods such as that proposed in Sun, Wang, et al. (2019) use a random forest for predicting the performance of the architecture during the performance estimation process, resulting in a high speed up as compared to previous EA methods. However, its performance was far from the state-of-the-art results. In contrast, our method achieved better results than previous EA methods while using significantly less computational resources. CMA-ES has shown good performance in many high-dimensional continuous optimization problems such as fine-tuning the hyperparameters of the CNN (Loshchilov and Hutter, 2016). However, to the best of our knowledge, CMA-ES has not been applied to the NAS problem because of the discrete nature of the problem.

3 Proposed Approach

3.1 Search Space

The choice of the search space can affect the quality of the searched architecture. CMANAS searches for both operations and connections which is in contrast to previous EA-based NAS (Xie and Yuille, 2017; Sun, Wang, et al., 2019; Elsken et al., 2018a; Suganuma et al., 2017) focusing on one facet of the architecture search, for example, connections and/or hyperparameters. This makes our search space more comprehensive. The success of the recent hand-crafted CNN architectures is attributed to their sharing similar characteristics of repeating motifs (He et al., 2016; Huang et al., 2017; Szegedy et al., 2016). Therefore, in Zoph et al. (2018) and Zhong et al. (2018), the researchers proposed to search for such motifs called cells, instead of the whole architecture. In this work, we used this cell-based search space, which has been successfully employed in recent works (Pham et al., 2018; Real et al., 2019; Zoph et al., 2018; Liu et al., 2019; Dong and Yang, 2019b, 2019a; Lu et al., 2020; Liu, Zoph, et al., 2018). As illustrated in Figure 3a, the architecture is created by stacking together cells of two types: normal cells, which preserve the dimensionality of the input with a stride of one, and reduction cells, which reduce the spatial dimension with a stride of two. To construct both types of cells, we used directed acyclic graphs (DAGs) containing n nodes and edge (i, j) between any two nodes representing an operation from the search space with $N_{ops}$ different operations. In this work, we applied our method to two different search spaces: Search space 1 (S1) (Liu et al., 2019) and Search space 2 (S2) (Dong and Yang, 2020).

3.1.1 Search Space 1 (S1)

S1 is similar to that used in Liu et al. (2019), which allows us to compare the performance of our method with other NAS methods. Here, we search for both normal and reduction cells in Figure 3a, where each node $x^{(j)}$ maps two inputs to one output. The two inputs for $x^{(j)}$ in cell k are picked from the outputs from previous nodes $x^{(i)}$ in cell k (i.e., $i < j$), output from previous cell $c_{k-1}$, and output from the previous-to-previous cell $c_{k-2}$.

3.1.2 Search Space 2 (S2)

S2 is a smaller search space with a total of 15,625 architectures in the search space and is similar to that used in NAS-Bench-201 (Dong and Yang, 2020), where we search only for the normal cell in Figure 3a. Here, each node $x^{(j)}$ is connected to the previous node $x^{(i)}$ (i.e., $i < j$). NAS-Bench-201 provides a unified benchmark for almost any up-to-date
NAS algorithms by providing the results of each architecture in the search space on CIFAR-10, CIFAR-100, and ImageNet16-120. It provides an API that can be used to query accuracies on both validation and test sets for all the architectures in the search space. The API provides two types of accuracies for each architecture, that is, accuracy after training the architecture for 12 epochs and 200 epochs. The accuracies of the architectures after 200 epochs are used as the performance measurement of various NAS algorithms. NAS-Bench-201 (Dong and Yang, 2020) (i.e., S2) provides the search results for two types of NAS methods: weight-sharing-based and non-weight-sharing-based. In the weight-sharing-based NAS methods, all the architectures in the search space share their weights to reduce the search time (e.g., Pham et al., 2018; Liu et al., 2019; Dong and Yang, 2019b, 2019a; Li and Talwalkar, 2020). In the non-weight-sharing-based NAS methods (e.g., Real et al., 2019; Bergstra and Bengio, 2012; Williams, 1992; Falkner et al., 2018), the architectures in the search space do not share their weights, and during the architecture search, the performance of each architecture is evaluated on the basis of the accuracy on the validation data after training for 12 epochs, which is provided by the API.

### 3.2 Representation of Architecture

As illustrated in Figure 3b, a cell in the architecture is represented by an architecture parameter, $\alpha$. Each $\alpha$ for a normal cell and a reduction cell is represented by a matrix with columns representing the weights of different operations $Op(.)$s from the operation space $O$ (i.e., the search space of NAS) and rows representing the edge between two nodes. For example, in Figure 3b, $\alpha(0, 1)$ represents the edge between node 0 and node 1 and the entries in the row represent the weights given to the three different operations. The operations shown in Figure 3 are generic operations to provide an overview. So, for
Neural Architecture Search using CMA-ES

\( N_{\text{ops}} \) number of operations in the search space and \( N_{\text{edges}} \) edges between nodes in the cell, \( \alpha \) has \( N_{\text{edges}} \times N_{\text{ops}} \) parameters. \( \alpha \) is modelled with a multivariate normal distribution, \( \mathcal{N}(\mathbf{m}, \mathbf{C}) \), with a mean vector, \( \mathbf{m} \), of size \( N_{\text{edges}} \times N_{\text{ops}} \), representing the parameters in \( \alpha \), and a covariance matrix, \( \mathbf{C} \), of size \( (N_{\text{edges}} \cdot N_{\text{ops}}) \times (N_{\text{edges}} \cdot N_{\text{ops}}) \), which is used for guiding the search process using CMA-ES. So, the representation for the search spaces are as follows:

- **Search space 1 (S1):** Here, each cell has 7 nodes with the first 2 nodes being the output from previous cells and last node as output node, resulting in 14 edges \( (N_{\text{edges}}) \) among them. There are eight operations \( (N_{\text{ops}}) \) considered in S1, which are as follows: 3 \( \times \) 3 and 5 \( \times \) 5 dilated separable convolutions, 3 \( \times \) 3 and 5 \( \times \) 5 separable convolutions, 3 \( \times \) 3 max pooling, 3 \( \times \) 3 average pooling, skip connect, and zero. Therefore, an architecture is represented by two 14 \( \times \) 8 matrices, one each for a normal cell and a reduction cell. The values in these two matrices are modeled with a multivariate normal distribution \( \mathcal{N}(\mathbf{m}, \mathbf{C}) \) with a mean vector, \( \mathbf{m} \), of size 224, and a covariance matrix, \( \mathbf{C} \), of size 224 \( \times \) 224.

- **Search space 2 (S2):** Here, each cell has 4 nodes with the first node as the input node and last node as the output node, resulting in six edges \( (N_{\text{edges}}) \) among them. The five operations \( (N_{\text{ops}}) \) considered in S2 are as follows: 1 \( \times \) 1 and 3 \( \times \) 3 convolutions, 3 \( \times \) 3 average pooling, skip connect, and zero. Therefore, an architecture is represented by a 6 \( \times \) 5 matrix for the normal cell. The values in the matrix are modeled with a multivariate normal distribution \( \mathcal{N}(\mathbf{m}, \mathbf{C}) \) with a mean vector, \( \mathbf{m} \), of size 30, and a covariance matrix, \( \mathbf{C} \), of size 30 \( \times \) 30.

An architecture is derived from \( \alpha \) for the two search spaces through a mapping process (discussed in Section 3.3).

### 3.3 Performance Estimation

Evaluating an architecture involves training it from scratch for some epochs and then evaluating it on the basis of its performance on validation data, leading to long search time (Real et al., 2019; Zoph et al., 2018). Instead, we use a one shot model (OSM) (Liu et al., 2019), which shares the weights among all architectures by treating all the architectures as the subgraphs of a supergraph. As illustrated in Figure 3c, the OSM uses the architecture parameter, \( \alpha \), by normalizing it using softmax. The directed edge from node \( i \) to node \( j \) is the weighted sum of all \( Op(.) \)s in \( O \) where the \( Op(.) \)s are weighted by the normalized \( \alpha^{(i,j)} \). This can be written as:

\[
f^{(i,j)}(x^{(i)}) = \frac{\sum_{Op \in O} \sum_{Op' \in O} \exp(\alpha^{(i,j)}_{Op}) \cdot Op(x^{(i)})}{\sum_{Op' \in O} \sum_{Op \in O} \exp(\alpha^{(i,j)}_{Op'}) \cdot Op(x^{(i)})} \tag{1}\]

where \( \alpha^{(i,j)}_{op} \) represents the weight of the operation \( Op(.) \) in the operation space \( O \) between node \( i \) and node \( j \). This design choice allows us to skip the individual architecture training from scratch for its evaluation because of the weight-sharing nature of OSM, thus resulting in a significant reduction of search time. The performance of an architecture is calculated by the trained OSM using the validation data. The OSM is trained for a certain number of epochs using Stochastic Gradient Descent (SGD), (Sutskever et al., 2013) with nesterov momentum. The training of the OSM begins with randomly initializing the weights of the OSM. Then for each training batch in an epoch, a uniformly random architecture parameter, \( \alpha \), is sent to the OSM so that no particular subgraph
Algorithm 1: Training one shot model

**Input:** One shot model \( M \), architecture parameter \( \alpha \), training data \( D_{tr} \), total epochs \( N_{epochs} \), initial learning rate \( \eta_{max} \), minimum learning rate \( \eta_{min} \), weight decay \( \lambda \), momentum term weight \( \rho \).

**Output:** Trained one shot model \( M \).

1. \( W \leftarrow \) Randomly initialize weights of \( M \);
2. \( \Delta v \leftarrow 0 \) (Initialize the momentum term);
3. \( B \leftarrow \) Number of training batches;
4. **for** \( \tau \leftarrow 1 \) to \( N_{epochs} \) **do**
   5. \( \eta_{r} \leftarrow \eta_{r} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{\tau}{N_{epochs}} \pi)) \); // cosine annealing
   6. **for** \( t \leftarrow 1, \ldots, B \) **do**
      7. Update \( \alpha \) to random values and send to \( M \);
      8. \( \mathcal{L}(W_{t-1}) \leftarrow \) Cross-entropy loss;
      9. \( g_{t} \leftarrow \nabla_{W}\mathcal{L}(W_{t-1}) + \lambda W_{t-1} \);
     10. \( \Delta v \leftarrow \rho \cdot \Delta v + g_{t} \);
     11. \( g_{t} \leftarrow g_{t-1} + \rho \cdot \Delta v \); // Using nesterov momentum
     12. \( W_{t} \leftarrow W_{t-1} - \eta \cdot g_{t} \);
   **end**
5. **end**

(i.e., architecture) receives most of the gradient updates of the supergraph (i.e., OSM). The algorithm\(^1\) is summarized in Algorithm 1 and its implementation details discussed in Section 4.3.1.

The trained OSM from Algorithm 1 is then used to evaluate an architecture on the basis of its accuracy on the validation data, also known as the *fitness* of the architecture. The process of evaluation follows two steps sequentially (as illustrated in Figure 4):

\(^1\)Based on the implementation of SGD with nesterov momentum provided in PyTorch.
Neural Architecture Search using CMA-ES

Figure 4: Process of evaluating architecture using the trained one shot model with three nodes for S1 and four nodes for S2. **Left:** mapping process wherein the architecture parameter $\alpha$ is mapped to its corresponding architecture. $c_{k-1}$ and $c_{k-2}$ refer to outputs from the previous cell and previous-to-previous cell, respectively. **Right:** discretization process wherein $\tilde{\alpha}$ is created from the derived architecture and is copied to the OSM. The colors of the arrows between any two nodes represent the different operations and the thickness of the arrow is proportional to the weight of the corresponding operation.

- **Mapping process:** Here, the architecture, $A$, is derived from the architecture parameter $\alpha$, on the basis of the search space used.
  - **Search space 1 (S1):** Each node maps two inputs to one output, so for each node, the top two distinct input nodes are chosen from all previous nodes on the basis of the weights of all the operations in the search space. For example, in Figure 4, the bottom 4 rows in $\alpha$ represent all possible connections to node 2 from the previous nodes using all three generic operations. In the mapping process, node 2 is connected to node 0 and node 1 through $Op_2$ operation because they have the top two weights.
  - **Search space 2 (S2):** Each node is connected to all previous nodes, so for each edge between any two nodes, the top operation is chosen on the basis of weights of all the operations in the search space. For example, in Figure 4, the bottom 3 rows in $\alpha$ represent all possible connections to node 3 from previous nodes using all three generic operations. In the mapping process, node 3 is connected to node 0 though $Op_3$, node 1 though $Op_2$ and node 2 though $Op_1$ because they have the highest weights in each row.

Figure 4 illustrates the mapping process with three operations in both S1 and S2 and three nodes in S1 and four nodes in S2.

- **Discretization process:** The derived architecture, $A$, from the mapping process is then used to create a new architecture parameter called discrete architecture.
parameter, $\bar{\alpha}$, with the following entries:

$$\bar{\alpha}_{O_p}^{(i,j)} = \begin{cases} 1, & \text{if } O_p(.) \text{ present in } A \\ 0, & \text{otherwise}, \end{cases}$$

(2)

where $O_p(.)$ represents an operation in the operation space $O$, $\bar{\alpha}^{(i,j)}$ is used to select an operation from $O$ between node $i$ and $j$, and $\bar{\alpha}$ is used to select an architecture in the one shot model. For example, in discretization for S2 in Figure 4, the row $\bar{\alpha}(0, 1)$ represents the three different operations that can be chosen between node 0 and 1. Since $O_{p2}$ is present in the architecture between node 0 and 1 (as seen in the mapping part of Figure 4), only the entry for $O_{p2}$ in the row for $\bar{\alpha}(0, 1)$ is 1. $\bar{\alpha}$ is then sent to the OSM for evaluation of its accuracy on the validation data, which is the fitness of the architecture $A$. This process gives higher equal weight to the architecture $A$ operations while giving lower equal weights to the other architecture operations. This results in the higher contribution from the architecture $A$ and very low contribution by other architectures during the fitness evaluation.

3.4 CMANAS

Covariance matrix adaptation evolution strategy (CMA-ES) (Hansen and Ostermeier, 2001) is a state-of-the-art evolutionary algorithm for continuous blackbox functions (i.e., functions for which only the function values are available for the search points; Hansen, 2016). We used the capability of CMA-ES’s convergence to find a solution using small population size, as compared to other evolutionary methods, for reducing the search time. As illustrated in Figure 2, CMANAS starts with initializing the normal distribution $\mathcal{N}(m, C)$. This distribution is then used to sample a population of architectures (i.e., $\alpha$) with the population size $N_{pop}$ according to the following equation:

$$\alpha^{(g+1)}_{1:N_{pop}} \sim m^{(g)} + \sigma^{(g)} \mathcal{N}(0, C^{(g)}),$$

(3)

where $g = 0, 1, 2, \ldots$ is the generation number and $\sigma$ is the step size. In the evaluation process, we use an architecture-fitness table (AF table) to save the fitness of the already evaluated architecture. For an individual architecture in the population, we first check whether there is an entry present in the AF table for this architecture. If it is already present, then the entry in the AF table is returned. Otherwise, the fitness of the architecture is evaluated using the trained OSM (Section 3.3), and the AF table is updated with it. The normal distribution, $\mathcal{N}(m, C)$, and the step size, $\sigma$, are updated using CMA-ES, which involves sorting the evaluated architectures in the population in decreasing order according to their fitness and using the top-$\mu$ individuals to update the mean, $m$, as follows:

$$m^{(g+1)} \leftarrow \sum_{i=1}^{\mu} w_i \alpha^{(g+1)}_i$$

(4)

where $\mu = N_{\text{pop}}/2$ and this process is also referred to as the selection and recombination part of CMA-ES. The step size, $\sigma$, is increased or decreased by comparing the length of the conjugate evolution path, $p_\sigma$ (sum of the previous successive steps), with the expected
Neural Architecture Search using CMA-ES

length under random selection. Lastly, the covariance matrix, $\mathbf{C}$, is updated using two terms: Rank-$\mu$ update ($\text{Rank}_\mu$) and Rank-1 update ($\text{Rank}_1$) as follows:

$$
\mathbf{C}^{(g+1)} \leftarrow (1 + k)\mathbf{C}^{(g)} + c_1\text{Rank}_1^{(g)} + c_\mu\text{Rank}_\mu^{(g)}
$$

(5)

where $k$, $c_1$, and $c_\mu$ are the hyperparameters of CMA-ES. For a more detailed analysis of the hyperparameters and the updates in CMA-ES, please refer to Hansen (2016). The updated distribution is then resampled for the next generation population of architectures and repeats the cycle for a certain number of generations, $N_{\text{gen}}$. The mean of the normal distribution is returned as the searched architecture, $\mathbf{m}^{(N_{\text{gen}})}$ after $N_{\text{gen}}$ generations. The algorithm is summarized in Algorithm 2. Note that CMA-ES was chosen over

---

**Algorithm 2: CMANAS**

**Input:** Trained one shot model $M$, validation data $\mathcal{D}_{\text{va}}$, AF table, total generations $N_{\text{gen}}$, population size $N_{\text{pop}}$.

**Output:** Searched architecture, $\mathbf{m}^{(N_{\text{gen}})}$.

1. $g \leftarrow 0$ (Initialize the generation counter);
2. Initialize CMA-ES with $\mathbf{m}^{(g)} \leftarrow \mathbf{0}$, $\mathbf{C}^{(g)} \leftarrow \mathbf{I}$ and $\sigma^{(g)}$;
3. while $g \leq N_{\text{gen}}$ do
   4. Sample population, $\mathcal{P}$, of $N_{\text{pop}}$ individuals using Eq. 3;
   5. for each individual architecture $(\mathcal{A})$ in $\mathcal{P}$ do
      6. if $\mathcal{A}$ already present in AF table then
         7. $\text{fitness}(\mathcal{A}) \leftarrow$ entry in the AF table;
      else
         9. $\text{fitness}(\mathcal{A}) \leftarrow$ Evaluate $\mathcal{A}$ using $M$;
         10. Insert fitness$(\mathcal{A})$ to the AF table;
      end
   end
   12. Update $\mathbf{m}^{(g)}$, $\sigma^{(g)}$ and $\mathbf{C}^{(g)}$ using Eq. 4 and Eq. 5;
   13. $g \leftarrow g + 1$;
15. end
other continuous direct search methods, for example, differential evolution, because it has already shown its potential in hyperparameter tuning in deep neural networks.

4 Experiments and Results

4.1 Baselines

In order to illustrate the effectiveness of CMANAS, we compared the architecture returned by CMANAS with the other architectures reported in various peer-reviewed NAS methods. These peer-reviewed NAS methods are broadly classified into five categories: architectures designed by humans (reported as manual), RL-based methods (reported as RL), gradient-based methods (reported as grad. based), EA-based methods (reported as EA), and others. The others include the random search and sequential model-based optimization (SMBO) wherein the architecture is searched in the increasing order of complexity of its structure. The effectiveness of the reported architectures is measured in terms of the classification accuracy, and the computational requirement is measured in terms of search time on a single GPU reported as GPU days/GPU hours.

4.2 Dataset Settings

Both CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009) have 50,000 training images and 10,000 testing images and are classified into 10 classes and 100 classes, respectively. ImageNet (Deng et al., 2009) is a popular benchmark for image classification and contains 1.28 million training images and 50,000 test images, which are classified into 1,000 classes. ImageNet-16-120 (Chrabaszcz et al., 2017) is a down-sampled version of ImageNet wherein the images in the original ImageNet dataset are downsampled to 16 × 16 pixels with 120 classes to construct the ImageNet-16-120 dataset. The settings used for the datasets in S1 are as follows:

- **CIFAR-10**: We split 50,000 training images into two sets of size 25,000 each, with one set acting as the training set and the other set as the validation set.
- **CIFAR-100**: We split 50,000 training images into two sets. One set of size 40,000 images becomes the training set and the other set of size 10,000 images becomes the validation set.

We followed the settings used in Dong and Yang (2020) for the datasets in S2, which are as follows:

- **CIFAR-10**: The same settings as those used for S1 are used here as well.
- **CIFAR-100**: The 50,000 training images remain as the training set, and the 10,000 testing images are split into two sets of size 5,000 each, with one set acting as the validation set and the other set as the test set.
- **ImageNet-16-120**: It has 151,700 training images, 3,000 validation images, and 3,000 test images.

A tabular version of the dataset split for all the tasks in S1 and S2 is provided in the supplementary appendix. The training set is used for training the OSM and the validation set is used for estimating the fitness of the sampled architecture during the search process (Section 3.3).
4.3 Implementation Details

4.3.1 Training Settings

The training process is executed two times in our method, as follows:

- **One shot model (OSM) training**: In general, the OSM suffers from high memory requirements, which makes it difficult to fit it in a single GPU. For S1, we follow Liu et al. (2019) and Li and Talwalkar (2020) and use a smaller OSM, called proxy model, which is created with 8 stacked cells and 16 initial channels for both CIFAR-10 and CIFAR-100 datasets. It is then trained with SGD for 100 epochs on both CIFAR-10 and CIFAR-100 with the same settings, that is, batch size of 96, weight decay \( \lambda = 3 \times 10^{-4} \), cutout (DeVries and Taylor, 2017), initial learning rate \( \eta_{\text{max}} = 0.025 \) (annealed down to 0 by using a cosine schedule without restart; Loshchilov and Hutter, 2017) and momentum \( \rho = 0.9 \). For S2, we do not use a proxy model, as the size of the OSM is sufficiently small to be fitted in a single GPU. So, the OSM in S2 is created by stacking 5 normal cells for all three datasets. For training, we follow the same settings as those used in S1 for CIFAR-10, CIFAR-100, and ImageNet16-120 except batch size of 256.

- **Architecture evaluation**: Here, the discovered architecture, \( A \) (i.e., discovered cells), at the end of the architecture search is trained on the dataset to evaluate its performance for comparison with other NAS methods. For S1, we follow the training settings used in DARTS (Liu et al., 2019). Here, \( A \) is created with 20 stacked cells and 36 initial channels for both CIFAR-10 and CIFAR-100 datasets. It is then trained for 600 epochs on both the datasets with the same settings as the ones used in the OSM training above. Following recent works (Pham et al., 2018; Real et al., 2019; Zoph et al., 2018; Liu et al., 2019; Liu, Zoph, et al., 2018), we use an auxiliary tower with 0.4 as its weights, path dropout probability of 0.2, and cutout (DeVries and Taylor, 2017) for additional enhancements. For ImageNet, \( A \) is created with 14 cells and 48 initial channels in the mobile setting, wherein the input image size is \( 224 \times 224 \) and the number of multiply-add operations in the model is restricted to less than 600M. It is trained on 8 NVIDIA V100 GPUs by following the training settings used in Chen et al. (2019).

All the above trainings were performed on a single Nvidia RTX 3090 GPU except the one on ImageNet. The number of epochs for training the OSM was chosen to be 100 as we found that the performance of the architecture search increases from 50 epochs to 100 epochs and then deteriorates upon further increase in the number of training epochs because of the overfitting of the OSM (as shown in Figure 5).

4.3.2 Architecture Search Settings

The multivariate normal distribution, \( \mathcal{N}(\mathbf{m}, \mathbf{C}) \), that is used to model the architecture parameter, \( \alpha \), is initialized with its mean, \( \mathbf{m} \), equal to the zero vector. This results in the assignment of equal weights to all the operations in all the edges because of the normalization by softmax (Eq. 1). As recommended in Hansen (2016), we initialize the covariance matrix, \( \mathbf{C} \), with an identity matrix and the population size, \( N_{\text{pop}} \), to \( 4 + \lceil 3 \times \ln(n) \rceil \), where \( n \) is the size of the mean, \( \mathbf{m} \). Therefore, for S1, \( N_{\text{pop}} = 20 \), and for S2, \( N_{\text{pop}} = 14 \). The other hyperparameters of CMA-ES are initialized to their default values as per the recommendation in Hansen (2016), which are summarized in the supplementary
Figure 5: Top-1 accuracy of architecture searches performed with OSM trained for 50, 100, 200, and 300 training epochs in S1 on CIFAR-10 dataset. The number after CMANAS specifies the number of training epochs for which OSM was trained.

Figure 6: Plot of validation accuracy/fitness of architecture given by the mean, $m$, of the normal distribution, $\mathcal{N}(m, C)$, at each generation in (a) S1 on CIFAR-10 (denoted as CMANAS-C10) and CIFAR-100 (denoted as CMANAS-C100); and (b) S2 on CIFAR-10 (denoted as CMANAS-C10), CIFAR-100 (denoted as CMANAS-C100), and ImageNet16-120 (denoted as CMANAS-IMG16). The plots show all three runs in both S1 and S2.

appendix. We ran CMANAS for 100 generations, as we observed that the algorithm converged well before 100 generations for both S1 and S2 (as shown in Figure 6). All the architecture searches were performed on a single Nvidia RTX 3090 GPU. All the codes were implemented using the deep learning framework PyTorch (Paszke et al., 2019).

4.4 Results

4.4.1 Search Space 1 (S1)

We performed three architecture searches on CIFAR-10 with different random number seeds; their results are provided in Table 1 as CMANAS-C10A, CMANAS-C10B, and CMANAS-C10C. We also performed another three architecture searches on
Table 1: Comparison of CMANAS with other NAS methods in S1 in terms of test accuracy (higher is better) on CIFAR-10.

| Architecture                  | Top-1 Acc. (%) | Params (M) | GPU Days | # Arch. Evaluated | Search Method |
|-------------------------------|----------------|------------|----------|-------------------|---------------|
| ResNet (He et al., 2016)      | 95.39          | 1.7        | -        | -                 | manual        |
| DenseNet-BC (Huang et al., 2017) | 96.54          | 25.6       | -        | -                 | manual        |
| ShuffleNet (Zhang et al., 2018) | 90.87          | 1.06       | -        | -                 | manual        |
| PNAS (Liu, Zoph, et al., 2018) | 96.59          | 3.2        | 225      | -                 | SMBO          |
| RSPS (Li and Talwalkar, 2020) | 97.14          | 4.3        | 2.7      | -                 | random        |
| NASNet-A (Zoph et al., 2018)  | 97.35          | 3.3        | 1800     | -                 | RL            |
| ENAS (Pham et al., 2018)      | 97.14          | 4.6        | 0.45     | -                 | RL            |
| DARTS (Liu et al., 2019)      | 97.24          | 3.3        | 4        | -                 | grad. based   |
| GDAS (Dong and Yang, 2019b)   | 97.07          | 3.4        | 0.83     | -                 | grad. based   |
| SNAS (Xie et al., 2019)       | 97.15          | 2.8        | 1.5      | -                 | grad. based   |
| SETN (Dong and Yang, 2019a)   | 97.31          | 4.6        | 1.8      | -                 | grad. based   |
| AmoebaNet-A (Real et al., 2019) | 96.66          | 3.2        | 3150     | 20,000            | EA            |
| Large-scale Evo. (Real et al., 2017) | 94.60          | 5.4        | 2750     | -                 | EA            |
| Hierarchical Evo. (Liu, Simonyan, et al., 2018) | 96.25          | 15.7       | 300      | 7,000             | EA            |
| CNN-GA (Sun et al., 2020)     | 96.78          | 2.9        | 35       | 400               | EA            |
| CGP-CNN (Suganuma et al., 2017) | 94.02          | 1.7        | 27       | 600               | EA            |
| AE-CNN (Sun, Xue, et al., 2019) | 95.7           | 2.0        | 27       | 400               | EA            |
| AE-CNN+E2EPP (Sun, Wang, et al., 2019) | 94.70          | 4.3        | 7        | 400               | EA            |
| EvNAS (Sinha and Chen, 2021)  | 97.37          | 3.4        | 3.83     | 10,000            | EA            |
| SI-ENAS (Zhang et al., 2020)  | 95.93          | -          | 1.8      | -                 | EA            |
| CMANAS-C10A                   | 97.44          | 3.8        | 0.45     | 1,021             | EA            |
| CMANAS-C10B                   | 97.35          | 3.2        | 0.45     | 1,040             | EA            |
| CMANAS-C10C                   | 97.35          | 3.3        | 0.45     | 1,052             | EA            |
| CMANAS-C10rand                | 97.11          | 3.11       | 0.66     | 2,000             | random        |

CIFAR-100 with different random number seeds; their results are provided in Table 2 as CMANAS-C100A, CMANAS-C100B, and CMANAS-C100C. The results show that the cells discovered by CMANAS on CIFAR-10 and CIFAR-100 achieve better results than those by human-designed, RL-based, gradient-based, and EA-based methods while using significantly less computational time. We compared the computation time spent (or search cost), in GPU days, of CMANAS with that for the other RL and EA-based NAS methods (as shown in Figure 8a). GPU days for any NAS method is calculated by multiplying the number of GPUs used in the NAS method by the execution time (reported in units of days). A single run of CMANAS on CIFAR-10 and CIFAR-100 took 0.45 and 0.6 GPU days (including the training time of the OSM and the architecture search time using the trained OSM on the dataset), respectively. For comparison with other NAS methods, the search cost on CIFAR-10 for CMANAS was used, as it is the most common search cost used in most of the NAS methods. From Figure 8a, we observe that CMANAS is able to achieve better results than the previous evolution-based methods AmoebaNet (Real et al., 2019), Large-scale Evolution (Real et al., 2017), and Hierarchical...
Table 2: Comparison of CMANAS with other NAS methods in S1 in terms of test accuracy (higher is better) on CIFAR-100.

| Architecture                  | Top-1 Acc. (%) | Params (M) | GPU Days | Search Method |
|-------------------------------|----------------|------------|----------|---------------|
| ResNet (He et al., 2016)      | 77.90          | 1.7        | -        | manual        |
| DenseNet-BC (Huang et al., 2017) | 82.82          | 25.6       | -        | manual        |
| ShuffleNet (Zhang et al., 2018) | 77.14          | 1.06       | -        | manual        |
| PNAS (Liu, Zoph, et al., 2018) | 80.47          | 3.2        | 225      | SMBO          |
| MetaQNN (Baker et al., 2017)  | 72.86          | 11.2       | 90       | RL            |
| ENAS (Pham et al., 2018)      | 80.57          | 4.6        | 0.45     | RL            |
| DARTS (Liu et al., 2019)      | 82.46          | 3.3        | 4        | grad. based   |
| GDAS (Dong and Yang, 2019b)   | 81.62          | 3.4        | 0.83     | grad. based   |
| SETN (Dong and Yang, 2019a)   | 82.75          | 4.6        | 1.8      | grad. based   |
| AmoebaNet-A (Real et al., 2019) | 81.07          | 3.2        | 3150     | EA            |
| Large-scale Evo. (Real et al., 2017) | 77.00          | 40.4       | 2750     | EA            |
| CNN-GA (Sun et al., 2020)     | 79.47          | 4.1        | 40       | EA            |
| AE-CNN (Sun, Xue, et al., 2019) | 79.15          | 5.4        | 36       | EA            |
| Genetic CNN (Xie and Yuille, 2017) | 70.95          | -          | 17       | EA            |
| AE-CNN+E2EPP (Sun, Wang, et al., 2019) | 77.98          | 20.9       | 10       | EA            |
| EvNAS (Sinha and Chen, 2021)  | 83.14          | 3.4        | 3.83     | EA            |
| SI-ENAS (Zhang et al., 2020)  | 81.36          | -          | 1.8      | EA            |

|               |                |            |          |               |
| CMANAS-C100A  | 83.24          | 3.4        | 0.60     | EA            |
| CMANAS-C100B  | 83.09          | 3.47       | 0.63     | EA            |
| CMANAS-C100C  | 82.73          | 2.97       | 0.62     | EA            |
| CMANAS-C100rand | 82.35          | 3.17       | 0.67     | random        |

Evolution (Liu, Simonyan, et al., 2018) while using between ten and a thousand times fewer GPU days.

We also provide the number of architectures evaluated during the search process for the EA-based methods in Table 1 (reported as “# Arch. Evaluated”), as it is the bottleneck that increases the computational cost (measured in GPU days). From Table 1, we can see that the accuracy of the searched architecture increases with the increase in the number of the evaluated architectures, AmoebaNet-A (Real et al., 2019) and Hierarchical Evolution (Liu, Simonyan, et al., 2018). In comparison, our method gives better results while evaluating significantly fewer architectures, which results in significant reduction of the search time. Also, note that methods like CNN-GA and CGP-CNN evaluate fewer architectures as compared to our method but have larger search time as compared to our method. This is because methods like CNN-GA and CGP-CNN train each architecture from scratch for evaluating each architecture, whereas our method trains the one shot model once and uses it to evaluate an architecture. The top cells discovered by CMANAS on CIFAR-10 and CIFAR-100 (i.e., CMANAS-C10A and CMANAS-C100A) are shown in Figure 7. The cells discovered by the other runs of CMANAS on CIFAR-10 and CIFAR-100 are provided in the supplementary appendix.
Neural Architecture Search using CMA-ES

Figure 7: Normal cell on the left side and reduction cell on the right side. (a) Cells discovered by CMANAS-C10A. (b) Cells discovered by CMANAS-C100A.

Figure 8: (a) Search cost comparison of CMANAS with the other EA-based and RL-based NAS algorithms. (b) Search cost comparison of CMANAS with the other NAS algorithms that use OSM in S2 using our setup (i.e., RTX 3090 Ti).

We followed Pham et al. (2018), Real et al. (2019), Zoph et al. (2018), Liu et al. (2019), and Liu, Zoph, et al. (2018) to compare the transfer capability of CMANAS with that of the other NAS methods, wherein the discovered architecture on a dataset was transferred to another dataset (i.e., ImageNet) by retraining the architecture from scratch on the new dataset. The best discovered architectures from the architecture search on CIFAR-10 and CIFAR-100 (i.e., CMANAS-C10A and CMANAS-C100A) were then evaluated on the ImageNet dataset in mobile setting, and the results are provided in Table 3. The results show that the cells discovered by CMANAS on CIFAR-10 and CIFAR-100 can be successfully transferred to ImageNet, achieving better results than those of human-designed, RL-based, gradient-based, and EA-based methods while using significantly less computational time.
Table 3: Comparison of CMANAS with other NAS methods in S1 in terms of test accuracy (higher is better) on ImageNet.

| Architecture                  | Test Accuracy (%) | Params + (M) | GPU Days | Search Method |
|-------------------------------|-------------------|--------------|----------|---------------|
|                               | top 1             | top 5        |          |               |
| MobileNet-V2 (Sandler et al., 2018) | 72.0              | 91.0         | 3.4      | 300           | manual        |
| PNAS (Liu, Zoph, et al., 2018)  | 74.2              | 91.9         | 5.1      | 588           | 225           | SMBO          |
| NASNet-A (Zoph et al., 2018)   | 74.0              | 91.6         | 5.3      | 564           | 1800           | RL            |
| NASNet-B (Zoph et al., 2018)   | 72.8              | 91.3         | 5.3      | 488           | 1800           | RL            |
| NASNet-C (Zoph et al., 2018)   | 72.5              | 91.0         | 4.9      | 558           | 1800           | RL            |
| DARTS (Liu et al., 2019)       | 73.3              | 91.3         | 4.7      | 574           | 4              | grad. based   |
| GDAS (Dong and Yang, 2019b)    | 74.0              | 91.5         | 5.3      | 581           | 0.83           | grad. based   |
| SNAS (Xie et al., 2019)        | 72.7              | 90.8         | 4.3      | 522           | 1.5            | grad. based   |
| SETN (Dong and Yang, 2019a)    | 74.3              | 92.0         | 5.4      | 599           | 1.8            | grad. based   |
| AmoebaNet-A (Real et al., 2019)| 74.5              | 92.0         | 5.1      | 555           | 3150           | EA            |
| AmoebaNet-B (Real et al., 2019)| 74.0              | 91.5         | 5.3      | 555           | 3150           | EA            |
| AmoebaNet-C (Real et al., 2019)| 75.7              | 92.4         | 6.4      | 570           | 3150           | EA            |
| NSGANetV1-A2 (Lu et al., 2020) | 74.5              | 92.0         | 4.1      | 466           | 27             | EA            |
| EvNAS (Sinha and Chen, 2021)   | 74.9              | 92.2         | 4.9      | 547           | 3.83           | EA            |
| CMANAS-C10A                    | 75.3              | 92.6         | 5.3      | 589           | 0.45           | EA            |
| CMANAS-C100A                   | 74.8              | 92.2         | 4.8      | 531           | 0.60           | EA            |

4.4.2 Search Space 2 (S2)

We performed architecture search on the CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets for both the types of NAS methods given in S2 (Dong and Yang, 2020):

- **Weight-sharing-based NAS**: Here, the architecture evaluation in CMANAS used the trained OSM (as discussed in Section 3.3). Following Dong and Yang (2020), we performed the architecture search three times on all three datasets and compared the results with those of other weight-sharing-based NAS methods because of the weight-sharing nature of the OSM. The results are reported as CMANAS in Table 4. In order to make a fair comparison in terms of search time, we re-ran all the methods using OSM on our setup, that is, RTX 3090, for the search space S2 and compare it with our method in Figure 8b. From the figure, we find that CMANAS is able to find good solutions with less search cost as compared to other methods using OSM. In Figure 9, we compare the progression of the search of the weight-sharing-based CMANAS with other NAS methods and found that CMANAS converges to a good solution at a much faster rate.

- **Non-weight-sharing-based NAS**: Here, the fitness of the architecture was evaluated to be the accuracy on the validation data after training for 12 epochs from scratch, which is provided by the API in S2. This allows us to simulate the process of using CMANAS without the OSM, wherein each architecture in the population is trained from scratch for 12 epochs and then evaluated on
Table 4: Comparison of CMANAS with other NAS methods on NAS-Bench-201 (i.e., S2) (Dong and Yang, 2020) with mean ± std. accuracies on CIFAR-10, CIFAR-100, and ImageNet16-120 (higher is better). The first block compares CMANAS with other weight-sharing-based NAS methods. The second block compares CMANAS with other non-weight-sharing-based NAS methods. Optimal in the third block refers to the best architecture accuracy for each dataset. Search times are given for a CIFAR-10 search on a single GPU.

| Method                  | Search (seconds) | CIFAR-10 validation | CIFAR-10 test | CIFAR-100 validation | CIFAR-100 test | ImageNet-16-120 validation | ImageNet-16-120 test | Search Method |
|-------------------------|------------------|---------------------|---------------|----------------------|---------------|---------------------------|---------------------|----------------|
| RSPS (Li and Talwalkar, 2020) | 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 (Liu et al., 2019)      | 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        | grad. based    |
| DARTS-V2 (Liu et al., 2019)      | 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        | grad. based    |
| GDAS (Dong and Yang, 2019b)      | 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        | grad. based    |
| SETN (Dong and Yang, 2019a)      | 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        | grad. based    |
| ENAS (Pham et al., 2018)         | 13314.51         | 39.77 ± 0.00        | 54.30 ± 0.00  | 15.03 ± 0.00          | 15.61 ± 0.00  | 16.43 ± 0.00               | 16.32 ± 0.00        | RL             |
| CMANAS                              | 13896            | 89.06 ± 0.4         | 92.05 ± 0.26  | 67.43 ± 0.42          | 67.81 ± 0.15  | 39.54 ± 0.91               | 39.77 ± 0.57        | EA             |
| AmoebaNet (Real et al., 2019)     | 0.02             | 91.19 ± 0.31        | 93.92 ± 0.30  | 71.81 ± 1.12          | 71.84 ± 0.99  | 45.15 ± 0.89               | 45.54 ± 1.03        | EA             |
| RS (Bergstra and Bengio, 2012)    | 0.01             | 90.93 ± 0.36        | 93.70 ± 0.36  | 70.93 ± 1.09          | 71.04 ± 1.07  | 44.45 ± 1.10               | 44.57 ± 1.25        | random         |
| REINFORCE (Williams, 1992)        | 0.12             | 91.09 ± 0.37        | 93.85 ± 0.37  | 71.61 ± 1.12          | 71.71 ± 1.09  | 45.05 ± 1.02               | 45.24 ± 1.18        | RL             |
| BOHB (Falkner et al., 2018)       | 3.59             | 90.82 ± 0.53        | 93.61 ± 0.52  | 70.74 ± 1.29          | 70.85 ± 1.28  | 44.26 ± 1.36               | 44.42 ± 1.49        | grad. based    |
| CMANAS-h12-Ep25                  | 3.64             | 91.23 ± 0.40        | 94.00 ± 0.39  | 72.16 ± 1.19          | 72.11 ± 1.10  | 45.69 ± 0.84               | 45.7 ± 0.79         | EA             |
| CMANAS-h12-Ep100                  | 7.12             | 91.28 ± 0.35        | 94.06 ± 0.34  | 72.40 ± 0.96          | 72.26 ± 0.84  | 45.74 ± 0.86               | 45.69 ± 0.88        | EA             |
| ResNet                              | N/A              | 90.83               | 93.97         | 70.42                | 70.86         | 44.53                     | 43.63               | manual         |
| Optimal                             | N/A              | 91.61               | 94.37         | 73.49                | 73.51         | 46.77                     | 47.31               | N/A            |
N. Sinha and K. W. Chen

Figure 9: Comparison of the weight-sharing-based CMANAS with (a) gradient-based methods, (b) RL method, and (c) random search in terms of the test accuracy of the derived architecture evaluated on CIFAR-10 at each generation for the search space S2.

The results show that CMANAS outperforms most of the weight-sharing NAS methods except GDAS (Dong and Yang, 2019b). However, GDAS performs worse when the size of the search space increases, as can be seen for S1 in Table 1. CMANAS also dominates the non-weight-sharing-based NAS methods. Notably, the non-weight-sharing-based CMANAS (i.e., CMANAS-h12-Ep25) also dominates the weight-sharing-based CMANAS, which shows that the trained OSM provides a noisy estimate of the fitness/performance of an architecture.

4.4.3 CMANAS vs Gradient-Based Methods
In Figure 9a, we compare the progression of the search of the weight-sharing-based CMANAS with that of the other gradient-based NAS methods. The gradient-based method, like DARTS (Liu et al., 2019), suffers from the overfitting problem, wherein it converges to operations that give faster gradient descent, that is, skip-connect operation due to its parameter-less nature, as reported in Chen et al. (2019), Zela et al. (2020), and Dong and Yang (2020). This leads to higher number of skip-connect in the final discovered cell, a local optimum (as shown in Figure 9a). To resolve the overfitting problem, Chen et al. (2019) used a regularization method of restricting the number of skip-connect to a specific number in the final normal cell for the search space S1. This optimal number of skip-connect is a search-space dependent value and thus, the same method cannot be applied to the search space S2. In contrast, CMANAS does not have to worry about the overfitting problem due to its stochastic nature and does not need any regularization specific to a search space, thus making it independent of the search space.

5 Further Analysis
In the following sections, we will analyze the different aspects of CMANAS.
5.1 Visualizing the Architecture Search

For analyzing the search, we visualize the search process by using the following techniques:

- We plotted the number of unique architectures sampled in every generation for both weight-sharing-based CMANAS and non-weight-sharing-based NAS (i.e., CMANAS-h12-Ep100) in S2 for CIFAR-10 and the weight-sharing-based CMANAS in S1 for both CIFAR-10 and CIFAR-100, as shown in Figure 10. From the figure, we made the following observations:

  1. The number of unique architectures sampled in the beginning part of the architecture search was equal to the population size, $N_{pop}$. This part could be considered as the exploration phase of CMANAS, wherein the algorithm is exploring the search space by sampling unique architectures.

  2. As the search progressed, the number of unique sampled architectures decreased because of the dominance of some architecture solutions in the population. This part could be considered the exploitation phase of CMANAS, wherein the algorithm keeps on sampling the already evaluated good solution.

  3. The non-weight-sharing-based CMANAS (i.e., CMANAS-h12-Ep100) converged considerably faster to a better solution than the weight-sharing-based CMANAS in S2. This showed that the trained OSM provided a noisy estimate of the fitness of an architecture.

  4. Because of the bigger size of S1 than that of S2, the exploration phase of CMANAS was larger in S1 than in S2.

Figure 10: Number of unique architectures sampled in every generation averaged over all the runs. (a) Non-weight-sharing-based CMANAS for S2 on CIFAR-10; (b) Weight-sharing-based CMANAS for S2 on CIFAR-10; (c) Weight-sharing-based CMANAS for S1 on CIFAR-10; (d) Weight-sharing-based CMANAS for S1 on CIFAR-100.
As the architecture parameter, $\alpha$, is modeled using the mean of the normal distribution $\mathcal{N}(m, C)$, we can visualize the progression of $\alpha$ by visualizing the mean, $m$, throughout the architecture search process. In Figure 11, the mean, $m$, of the distribution in a generation is visualized using a bar plot, wherein each bar in the plot represents the edge between two nodes in the cell (x-axis). All the operations in the search space are represented by different colors, and the weight associated with any operation between two nodes (i.e., $\alpha$) is represented by the width of the color associated with that operation in the bar for that specific edge. From the figure, we observed that the search began with equal weights to all the operations in all the edges at generation 0, and as the search progressed, CMANAS changes the weights according to the fitness estimation of the architectures in the population. As the search converged to an architecture, CMANAS increased the weights of the operations of this architecture. Figure 11 shows the search progression for both weight-sharing-based CMANAS and non-weight-sharing-based CMANAS for S2 on CIFAR-10 dataset only. For CIFAR-100 and ImageNet16-120 datasets, please refer to the supplementary appendix.

5.2 Observation on Sampled Architectures

The API provided in the search space, S2, (i.e., NAS-Bench-201, Dong and Yang, 2020) allows a faster way to analyze a NAS algorithm. To discover the patterns associated with the sampled architectures, we plotted the frequency of all the operations in the search space S2 for the top 20 sampled architectures (in terms of their estimated fitness) in both the non-weight-sharing-based CMANAS (i.e., CMANAS-h12-Ep100) and the
Neural Architecture Search using CMA-ES

Figure 12: Frequency of all operations in S2 for top 20 architectures in S2, (i.e., NAS-Bench-201) labeled as “Top NAS201,” top 20 sampled architectures of non-weight-sharing-based CMANAS, labeled as “CMANAS-h12-Ep100,” and top 20 sampled architectures of weight-sharing-based CMANAS, labeled as “CMANAS.”

weight-sharing-based CMANAS as shown in Figure 12. They were compared with the frequency of all the operations in the search space S2 for the top 20 architectures (in terms of the test accuracy after 200 epochs), and we observed the following:

- The operation $3 \times 3$ convolutions dominated the top 20 architectures of S2, which was also seen in the non-weight-sharing-based CMANAS, but the weight-sharing-based CMANAS showed a marginal dominance.

- The frequency of operation $3 \times 3$ average pooling was very low in the top 20 architectures of S2 and was also seen in the non-weight-sharing-based CMANAS, but the weight-sharing-based CMANAS showed higher frequency.

All these observations show that the trained OSM provides a noisy estimate of the fitness of an architecture and with a better estimator, CMANAS can yield better results.

5.3 Ablation

Comparison with Random Search: Here, we do not the update the normal distribution, $N(m, C)$, after estimating the fitness of the individual architectures in the population (i.e., no “update distribution” block as shown in Figure 2). The sampled architecture with the best fitness is returned as the searched architecture after 100 generations. This is essentially a random search and is reported in Table 1 for CIFAR-10 as CMANAS-C10rand and in Table 2 for CIFAR-100 as CMANAS-C100rand. We found that CMANAS-C10rand shows similar results to those reported in RSPS (Li and Talwalkar, 2020) (random search with parameter sharing). We also found that the CMANAS took less time than the random search for both datasets while outperforming the random search in both the datasets. The reason for the random search taking longer than the CMANAS is due to the higher number of architectures evaluated during the search process (Table 1). Also, in Figure 9c, we compare the progression of the search of the weight-sharing-based CMANAS with that of the random search (Li and Talwalkar, 2020) in S2 and found that a mere random search in S2 cannot provide a good solution.

Effectiveness of the AF Table: To illustrate the effectiveness of the AF table used during the evaluation of the fitness of the architecture, we plotted the total number of architectures evaluated during the CMANAS search with and without the AF table as shown in Figure 13, and made the following observations:
Figure 13: Total number of architectures evaluated in a single run of weight-sharing-based CMANAS on (a) S1 for CIFAR-10 and CIFAR-100; (b) S2 for CIFAR-10, CIFAR-100, and ImageNet16-120.

Table 5: Search time of CMANAS with and without the AF table in both S1 and S2.

|          | S1                  | S2                  |
|----------|---------------------|---------------------|
|          | CIFAR-10            | CIFAR-100           |
|          | Search time in GPU days | Search time in seconds |
| Without AF table | 0.66 | 0.67 | 21807 | 27648 | 14220 |
| With AF table   | 0.45 | 0.62 | 13824 | 25992 | 13536 |

- On average, the use of the AF table reduces the architecture evaluation by 50% for S1 and 90% for S2 (as shown in Figure 13). This results in reducing the search time for CMANAS in both the search spaces on all the datasets and has been summarized in Table 5.

- Because of the smaller size of S2 than that of S1, CMANAS requires a smaller number of architecture evaluations in order to converge on an optimal solution in S2 than in S1.

- The speedup is much bigger for CIFAR-10 as compared to the other datasets in both S1 and S2 because of the bigger size of the validation data used for CIFAR-10 (Section 4.2), which results in longer fitness evaluation time.

6 Conclusion and Future Work

The goal of this paper was to develop a framework for utilizing the faster convergence property of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and extending its applicability to the NAS problem while using significantly less computational time than the previous evolution-based NAS methods. This was achieved by using a trained one shot model (OSM) for evaluating the architectures in the population, which allowed us to skip the training of each individual architecture from scratch.
Neural Architecture Search using CMA-ES

for its fitness evaluation. We applied our method (CMANAS) to two different search spaces to show its effectiveness in generalizing to any cell-based search space, that is, search-space agnostic. Experimentally, CMANAS reduced the architecture search time significantly by factors of ten to a thousand, while achieving better results on CIFAR-10, CIFAR-100, and ImageNet datasets than the previous evolutionary algorithms. The reduction in search time was achieved due to the use of one shot model, AF table, and the ability of CMA-ES to converge to a solution using a smaller number of architecture evaluations. CMANAS also solves the overfitting problem present in the gradient-based NAS methods because of its stochastic nature. We also created a visualization of the architecture search performed by CMANAS and found that the search begins with giving equal weights to all architectures in the search space and increases the weights of the converged architecture as the search progresses. We also analyzed the search process and found that the first part of the architecture search in the CMANAS acts as the exploration phase, wherein it explores the search space, and the later part acts as the exploitation phase, wherein the search converges to an architecture. A possible future direction to improve the performance of the algorithm is to use a better fitness estimator, as we found that a better estimator would allow CMANAS to achieve its full potential.

Acknowledgments

This work was supported in part by the National Science and Technology Council of Taiwan (111-2628-E-A49-003-MY2 and 111-2634-F-A49-010-). Furthermore, we are grateful to the National Center for High-Performance Computing for computer time and facilities.

References

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2017). Designing neural network architectures using reinforcement learning. International Conference on Learning Representations.

Bergstra, J., and Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(2).

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In Proceedings of the IEEE International Conference on Computer Vision, pp. 1294–1303.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of ImageNet as an alternative to the CIFAR datasets. Retrieved from arXiv:1707.08819.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F.-F. (2009). ImageNet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255.

DeVries, T., and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. Retrieved from arXiv:1708.04552.

Dong, X., and Yang, Y. (2019a). One-shot neural architecture search via self-evaluated template network. In Proceedings of the IEEE International Conference on Computer Vision, pp. 3681–3690.

Dong, X., and Yang, Y. (2019b). Searching for a robust neural architecture in four GPU hours. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1761–1770.

Dong, X., and Yang, Y. (2020). NAS-Bench-201: Extending the scope of reproducible neural architecture search. In International Conference on Learning Representations. Retrieved from arXiv:2001.00326.
N. Sinha and K. W. Chen

Elsken, T., Metzen, J. H., and Hutter, F. (2018a). Efficient multi-objective neural architecture search via Lamarckian evolution. Retrieved from arXiv:1804.09081.

Elsken, T., Metzen, J. H., and Hutter, F. (2018b). Neural architecture search: A survey. Retrieved from arXiv:1808.05377.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pp. 1437–1446.

Hansen, N. (2016). The CMA evolution strategy: A tutorial. Retrieved from arXiv:1604.00772.

Hansen, N., and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195. 10.1162/106365601750190398

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708.

Jin, Y., Olhofer, M., and Sendhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494. 10.1109/TEVC.2002.800884

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Li, L., and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Li, F.-F., Yuille, A., Huang, J., and Murphy, K. (2018). Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, pp. 19–34.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2018). Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*. Retrieved from arXiv:1711.00436.

Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*. Retrieved from arXiv:1806.09055.

Loshchilov, I., and Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. Retrieved from arXiv:1604.07269.

Loshchilov, I., and Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the 5th International Conference on Learning Representations*. 10.48550/arXiv.1608.03983

Lu, Z., Whalen, J., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W., and Boddeti, V. N. (2020). Multi-objective evolutionary design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*. Retrieved from arXiv:1912.01369.

 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H.
Larochelle, A. Beygelzimer, F. d’Alché-Bue, E. Fox, and R. Garnett (Eds.), *Advances in neural information processing systems, 32*, pp. 8024–8035. Curran Associates.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, pp. 4095–4104.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 4780–4789. 10.1609/aaai.v33i01.33014780

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, pp. 2902–2911.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520.

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. Retrieved from arXiv:1409.1556.

Sinha, N., and Chen, K.-W. (2021). Evolving neural architecture using one shot model. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 910–918.

Stanley, K. O., D’Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212. 10.1162/artl.2009.15.2.15202

Stanley, K. O., and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127. 10.1162/106365602320169811

Suganuma, M., Shirakawa, S., and Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 497–504.

Sun, Y., Wang, H., Xue, B., Jin, Y., Yen, G. G., and Zhang, M. (2019). Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation*, 24(2):350–364. 10.1109/TEVC.2019.2924461

Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2019). Completely automated CNN architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4):1242–1254. 10.1109/TNNLS.2019.2919608

Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Lv, J. (2020). Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854. 10.1109/TCYB.2020.2983860

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pp. 1139–1147.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256.

Xie, L., and Yuille, A. (2017). Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1379–1388.

Xie, S., Zheng, H., Liu, C., and Lin, L. (2019). SNAS: Stochastic neural architecture search. In *International Conference on Learning Representations*. Retrieved from arXiv:1812.09926.
Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020). Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*. Retrieved from arXiv:1909.09656.

Zhang, H., Jin, Y., Cheng, R., and Hao, K. (2020). Sampled training and node inheritance for fast evolutionary neural architecture search. Retrieved from arXiv:2003.11613.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856.

Zhong, Z., Yan, J., Wu, W., Shao, J., and Liu, C.-L. (2018). Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2423–2432.

Zoph, B., and Le, Q. V. (2016). Neural architecture search with reinforcement learning. Retrieved from arXiv:1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710.