IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 1 May 2022, accepted 6 June 2022, date of publication 21 June 2022, date of current version 30 June 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3184291

IGWO-SS: Improved Grey Wolf Optimization Based on Synaptic Saliency for Fast Neural Architecture Search in Computer Vision

SHIFAT E. ARMAN AND SHAMIM AHMED DEOWAN
Department of Robotics and Mechatronics Engineering, University of Dhaka, Dhaka 1000, Bangladesh
Corresponding author: Shamim Ahmed Deowan (shamimdeowan.rme@du.ac.bd)

This work was supported by the University of Dhaka, Dhaka, Bangladesh.

ABSTRACT Neural Architecture Search (NAS) is the process of automating the design of neural network architectures for a given task. Although NAS automates the process of finding suitable neural network architectures for a specific task, the existing NAS algorithms are immensely time-consuming. The main bottleneck in NAS algorithms is the training time for each architecture. This study proposes an Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS), which is much faster than the existing NAS algorithms and provides better final performance. The IGWO-SS algorithm skips training the less promising architectures by creating a relative rank between the architectures based on synaptic saliency. The architectures that are lower in rank are considered less promising than those that are higher in rank. Since the calculation of synaptic saliency is a very fast process, a significant amount of time is saved by skipping training of less promising architectures. This study involves extensive experiments assessing synaptic saliency's effectiveness in improving NAS. The experimental results indicate that the synaptic saliency of an untrained neural network positively correlates with its final accuracy. Hence, it can be used to identify untrained promising neural networks. The experimental results also suggest that the IGWO-SS algorithm is almost 10x faster and achieves better final performance than five other bio-inspired algorithms. The IGWO-SS algorithm achieves higher mean accuracy than state-of-the-art NAS algorithms, including - REA, RS, RL, BOHB, DARTSV1, DARTSV2, GDAS, SETN, and ENAS. We hope our work will make NAS more accessible and useful to researchers by reducing the time and resources required to perform NAS.

INDEX TERMS Neural architecture search, NAS, grey wolf optimization, GWO, AutoML, deep learning.

I. INTRODUCTION

Artificial Intelligence (AI) has experienced a paradigm shift with the emergence of deep learning. Before the emergence of deep learning, researchers used handcrafted features to build classifiers. This manual extraction of features was time-consuming, inefficient, and tedious. Since deep learning emerged, it was no longer necessary to manually extract features from data, as these networks possess the incredible ability to extract significant features on their own without human intervention. The transition from manual to automatic feature extraction was a significant step forward for the entire community. Unfortunately, although the problem of manual feature extraction was solved, a new challenge arose: the manual design of neural network architectures.

Although various powerful architectures like VGG [1], Inception [2], Xception [3],Mobilenets [4], Shufflenet [5], ResNext [6], Polynet [7], Fractalnet [8] were designed manually, the manual design of neural networks is very inefficient and expensive. Moreover, the same neural network architecture does not work well for all tasks. With the change in tasks, the architecture also needs to be changed. With the increase in the task's difficulty, finding the best architecture suitable for the task becomes extremely difficult. To mitigate this difficulty, the machine learning community tried to developed algorithms that can automatically find the best architecture. It gave rise to a new field of research - Neural Architecture Search or, in short, NAS.

Neural Architecture Search (NAS) is a field of research associated with automating neural network architecture design using different optimization algorithms. According to Elsken *et al.* [9], NAS methods have three dimensions - search space, search strategy, and performance estimation

The associate editor coordinating the review of this manuscript and approving it for publication was Hualong Yu.

strategy. Search space is the space in which the architecture will be searched. The search space consists of all possible architecture candidates. The searching procedure is done following a specific search strategy. The search strategy aims to find the best neural network architecture for a particular task, from a predefined search space, within a specific time or resource budget. The performance estimation strategy determines how the performance of a neural network will be estimated. The most straightforward performance estimation strategy is to calculate the validation performance after full training of a neural network. A less accurate performance estimation is also possible from a partially trained neural network.

The field of neural architecture search exploded when Zoph & Le [10] achieved state-of-the-art results on the CIFAR-10 [11], and the Penn Treebank [12] dataset by using Reinforcement Learning (RL). Despite having tremendous success in terms of predictive performance, the NAS algorithm proposed by Zoph & Le was extremely slow. It took them 28 days and 800 GPUs to find an architecture that achieved state-of-the-art results on the CIFAR-10 dataset. Such a vast amount of time and resources is only available to a few research groups. Independent researchers and small research groups do not have access to enough resources, which will seriously hamper the democratization of AI.

Various techniques were subsequently introduced to accelerate the architecture search process. One such technique involves learning stackable cells instead of learning the entire network. The idea was to learn cells for CIFAR-10 and then use these learned cells to build a large network for ImageNet classification. However, the proposed technique still required four days on 500 GPUs [13]. Naturally, the question arises: Why are these NAS methods so slow? The main reason why the NAS methods are slow is that, during the optimization process, numerous neural networks have to be trained to find the right architecture, and training each of these neural networks takes a considerable amount of time. However, when the search space is very large, most neural network architectures are less promising or less likely to perform well. In order to identify these less promising architectures, several techniques were introduced [14]–[17]. However, these techniques mostly require partial training of numerous neural networks. From the information of partially trained neural networks, these methods try to predict the final performance of the neural network.

Existing works on NAS mainly focus on Reinforcement Learning (RL) [10], [13], [18], [19], Sequential Model Based Optimization (SMBO) [20], [21], or Gradient Optimization (GO) [22]–[24] based strategies. The use of metaheuristic algorithms to perform NAS is largely unexplored, although these algorithms are known to provide good solutions within a short period. This study explored the use of meta-heuristic algorithms for NAS. We proposed an Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS) to perform fast and efficient NAS. The IGWO-SS algorithm leverages the idea of synaptic saliency to identify promising neural networks from a set of untrained neural networks. The algorithm skips training of less promising neural networks, thereby reducing the enormous computational cost of NAS.

The main contributions of this paper are as follows:

1) **Efficacy of Synaptic Saliency to Improve NAS:** We performed several experiments evaluating the efficacy of synaptic saliency to improve NAS. The experimental results suggest that synaptic saliency of untrained neural networks positively correlates with accuracy, model size, and FLOPS of the networks. Hence, it can be used to identify untrained promising neural networks that will make it an effective tool to improve NAS.
2) **Identification of a Limitation of Synaptic Saliency to Improve NAS:** We identified one specific limitation of using synaptic saliency - synaptic saliency of larger networks tends to be bigger. Hence, during the identification of promising architectures from a larger set of architectures, architectures that are larger in size, are most likely to be preferred.
3) **Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS) for Fast NAS:** This study presents an Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS) for fast Neural Architecture Search. The IGWO-SS algorithm leverages synaptic saliency to identify promising neural network architectures and skip training of less promising neural network architectures. It provides better final performance than the state-of-the-art NAS algorithms and several metaheuristic algorithms. Moreover, it requires training almost $20x$ fewer models than the standard GWO algorithm and $10x$ fewer models than other metaheuristics algorithms to achieve similar or better performance.

The rest of the paper is organized as follows: The related works are described in Section II. In Section III, the Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS) is presented. The experimental results and discussions are provided in Section IV and Section V. Finally, the paper is concluded in Section VI.

II. RELATED WORKS

Earlier works of neural architecture search can be dated back to 1990s neuroevolution [25], [26]. Neuroevolution is the idea of creating artificial neural networks using an evolutionary algorithm. In 2002, a software called rapid miner was developed that could do a grid search for entire chains of operators [27]. In 2009, Particle Swarm Model Selection (PSMS) was proposed that used Particle Swarm Optimization (PSO) for Full Model Selection (FMS) for classification tasks [28]. Bayesian optimization became popular after 2010 with the introduction of Sequential Model-Based Algorithm Configuration (SMAC) [29], Spearmint [29] and Tree Parzen Estimator (TPE) [30]. Various Bayesian optimization-based automated machine learning frameworks like Auto-WEKA [31], Auto-sklearn [32]–[34] and hyperopt-sklearn [35], [36] were introduced, which made the use of

Bayesian optimization easier. In 2013, James Bergstra [37] achieved the state-of-the-art performance on three computer vision problems datasets - Labeled Faces in the Wild (LFW) [38], Pubfig83 [39], and CIFAR-10 [11] using TPE. In 2016, Tree-Based Pipeline Optimization Tool (TPOT) was introduced that could make model selection based on an evolutionary algorithm [40], [41]. In 2016, AutoNet became the first AutoML tool to win a competition dataset against human experts [42].

The field of neural architecture search exploded when Zoph & Le [10] achieved state-of-the-art results on the CIFAR-10 [11] and the Penn Treebank [12] dataset by using reinforcement learning to perform NAS. However, the overall architecture search process took 28 days, and 800 GPUs were used throughout the entire period. Later on, various techniques were proposed to speed up the architecture search process. The authors in [13] proposed learning of stackable cells instead of learning entire networks, which will make the overall architecture search process much faster. In the same year, Pham *et al.* [43] used weight sharing between candidate networks so that joint training can be done. It reduced the time required to perform neural architecture search to half a day on a single GPU.

In order to make NAS faster, several performance estimation strategies were introduced. These strategies can be divided into two groups: performance estimation during training and before training. Performance prediction during training is done by performing a reduced-computation training. The idea is to reduce the amount of computation and time by reducing the number of epochs, using a mini-batch of data instead of the whole dataset, or reducing the input's resolution. Several works tried to extrapolate the final performance of neural networks from partially trained neural networks and then stop or completely discard training of the less promising neural networks based on the extrapolated final performance [14]–[17]. Zhou *et al.* [44] found that, while performing reduced computation training, more samples and fewer epochs is better than fewer samples and more epochs. They also found that reducing the channel of the neural network is much more reliable than reducing the resolution of the input. Based on these observations, they proposed an algorithm called Economical evolutionary-based NAS (EcoNAS), which is 400 times faster than evolutionary-based algorithms [45]. In 2017, Deng *et al.* [46] proposed an approach called peephole that predicts the performance of the model before training based on the model architecture. However, the major drawback of peephole is that - for a new dataset, many architectures had to be trained before peephole can make a prediction. Unlike peephole, TAPAS [47] not only considers model architectures but also uses Dataset Characterization Number (DCN), which is used to rank datasets based on difficulty. It performs better than both peephole [46] and LCE [15] in terms of mean squared error (MSE), Kendall's Tau (*Tau*) and coefficient of determination (R^2). In neural network pruning, weights can be pruned using synaptic saliency. Three notable synaptic saliency found in the literature are - Single-Shot Network Pruning (SNIP) [48], Gradient Signal Preservation (GRASP) [49], and Synaptic Flow Pruning (SYNFLOW) [50]. Abdelfatteh *et al.* [51] used different zero-cost proxies, including synaptic saliency, to perform lightweight NAS. They were able to achieve better predictive performance faster using these zero-cost proxies. Mellor *et al.* [52] proposed an algorithm that performs NAS by looking at the correlation matrices of the Jacobian of the data when it is passed through the network before training. The data is passed through the network, and the Jacobian is calculated. If it is very correlated, then the network is bad. If it is uncorrelated, then the network is good.

This study proposed an Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS). The synaptic saliency is used to rank neural networks based on how promising they seem before training. Only the more promising neural networks are trained while the rest are discarded, saving up time and drastically improving performance.

III. METHODOLOGY

This section presents our proposed NAS algorithm, Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS), for fast and efficient NAS. At first, we discuss the search space - mainly the macro skeleton of each architecture candidate and the cells, which are the building blocks of the macro skeleton. Then we discuss the datasets and training pipeline. Finally, we present the IGWO-SS algorithm that leverages synaptic saliency to identify promising neural network architectures from a large set of candidate architectures.

A. MODULAR SEARCH SPACE

In order to avoid the challenges associated with a global search space, a modular search space is used in this experiments. We used the NAS-Bench-201 [53] search space. The NAS-Bench-201 extends the NAS-Bench-101 [54] with multiple datasets, a different search space, and more information. The search space is essentially a fixed cell-based one with 15,625 possible architectures. The training pipelines while training these architectures were kept the same. The loss and accuracy of the neural networks on train, validation, and test set for CIFAR-10 [11], CIFAR-100 [11], and ImageNet-16-120 [55] after training are stored in this benchmark.

1) MACRO SKELETON OF ARCHITECTURE CANDIDATES

The macro skeleton of each architecture in NAS-Bench-201 is shown in Figure 1. An image is taken as input by the neural network architecture. It then goes through a 3×3 convolution layer followed by a batch normalization layer. Each architecture's macro skeleton comprises three groups of cells linked by residual blocks. The cells are stacked N times each. The structure of the cell varies from architecture to architecture, however, the macro skeleton is kept the same.

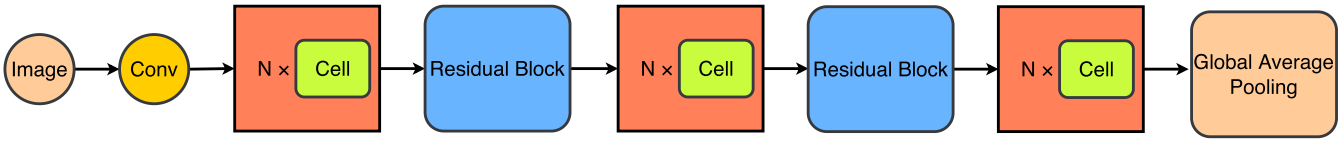
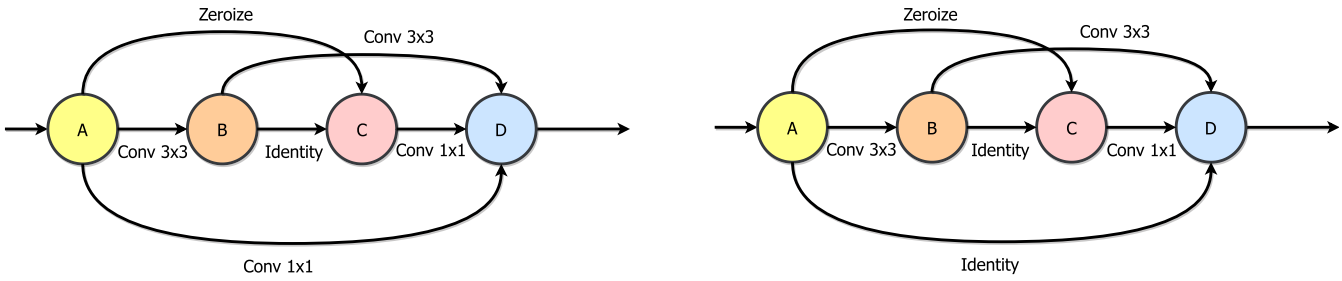


FIGURE 1. Macro-skeleton of architecture candidates.



(a) Example Cell 1 (b) Example Cell 2

FIGURE 2. Example of two unique cells.

The residual blocks have a stride of two, which downsamples the feature map. A global average pooling layer is used to transform the feature map into a feature vector. The final prediction is made using a fully connected layer having softmax activation.

2) CELLS: BUILDING BLOCKS OF THE MACRO-SKELETON

Each cell in the macro-skeleton comprises $V = 4$ nodes. Examples of two unique cells are shown in Figure 2. The nodes are connected by edges, where each edge represents a specific operation. There are 6 edges in total in a cell. The list of operations is predefined. There are five operations $L = 5$. These operations are: 1) 1×1 convolution, 2) 3×3 convolution, 3) 3×3 average pooling, 4) Skip connection, and 5) Zeroize. The zeroize operation indicates that the edge is dropped. Changing these operations allows the creation of different unique cells, giving rise to unique architecture candidates. There are $5^6 = 15,625$ possible architecture candidates in total.

3) DECISION SPACE

The decision space consists of all possible neural network architectures on NAS-Bench-201. Before starting the optimization process, the operations are encoded into integers. Each operation and their encoded integer values are shown in Table 1. There are 6 edges in a cell of each architecture candidate. Each of these edges can take one specific operation. A cell can be represented as a vector of length 6. Each entry in the vector represents a distinct edge, and the value of that entry represents the operation performed at that edge. Any architecture configuration of the NAS-Bench-201 search space can be created using this vector.

TABLE 1. Operation encoding.

Operations	Encoded Value
None	0
Skip connection	1
Normalized Convolution $1x1$	2
Normalized Convolution $3x3$	3
Average Pooling $3x3$	4

B. DATASET

The three most commonly used vision datasets are used throughout the experiments to check the efficacy of the algorithms. These datasets are - CIFAR-10 [11], CIFAR-100 [11], and ImageNet-16-120 [55]. The datasets are split into a train, validation, and test set.

1) CIFAR-10

The CIFAR-10 dataset comprises images of 10 classes. The total number of images in the dataset is 60K, where each image is of resolution 32×32. The training set contains 50K images of these 10 classes, with 5K images per class. The test set contains 10K images, with 1K images per class.

2) CIFAR-100

Unlike CIFAR-10, CIFAR-100 comprises images of 100 classes. The training set contains 50K images, whereas the test set contains 10K images.

3) ImageNet-16-120

The ImageNet-16-120 is a downsampled version of the original ImageNet dataset. According to [56], the downsampling of ImageNet reduces computational cost but provides similar

results. The ImageNet-16-120 dataset comprises 151.7K, 3K, and 3K images on training, validation, and test sets. The resolution of each downsampled image in ImageNet-16-120 is 16×16. In total, there are 120 classes in the dataset.

C. TRAINING PIPELINE

The training pipeline used in NAS-Bench-201 is similar to [13], [57], [57]. Nesterov momentum Stochastic gradient descent (SGD) is used as an optimizer during the training process, which has better convergence than normal SGD. To calculate loss, a categorical cross-entropy loss function is used. Each architecture is trained for 200 epochs. The weight decay was set to 0.0005, and cosine annealing [57] was used to decay the learning rate from 0.1 to 0. Random crop, random flip, and normalization were used as augmentation strategies.

D. IMPROVED GREY WOLF OPTIMIZATION BASED ON SYNAPTIC SALIENCY (IGWO-SS)

This section of the paper presents the IGWO-SS algorithm for NAS. Unlike the Standard GWO algorithm, in the IGWO-SS algorithm, both the initialization and new population generation are done by leveraging synaptic saliency. The synaptic saliency is used to create a rank between the architectures. The architectures that are higher in rank are considered more promising, and those that are lower in rank are considered less promising. The IGWO-SS algorithm only focuses on the more promising architectures. It trains the more promising architectures and discards the rest. Since the calculation of synaptic saliency is a very fast process, the ranking can be done in a very short amount of time. Overall, the algorithm requires training much fewer models to achieve similar or even better result than other algorithms, making it extremely fast. Before going deeper into the algorithm, the method of identifying promising neural network architectures by leveraging synaptic saliency will be discussed. After that, the proposed IGWO-SS algorithm and how it works alongside the components of NAS will be explained.

1) RATIONALE BEHIND USING SYNAPTIC SALIENCY

The performance of an untrained neural network can be estimated using different techniques like Multi Layer Perceptron (MLP) [58], Gaussian Process (GP) [59], Sparse GP [60], Random Forest (RF) [61], XGBoost [61], Bayesian Optimization based techniques [62]–[64] etc. However, the challenges with using these techniques is high initialization time. Numerous neural networks have to be trained, and their accuracy will have to be stored in a temporary dataset to train these predictors to predict the performance of an untrained model. Besides, these predictors need to be updated in real-time to achieve more accurate performance predictions. It adds further cost to the search process. Methods like Learning Curve Extrapolation (LCE) [15], [16] require zero initialization time but a significant amount of query time since each neural network architecture needs to be trained partially to be able to predict the final performance. In this study, synaptic saliency is used, which requires no initialization time and almost zero query time. The synaptic saliency of an untrained neural network can be calculated just by doing a forward and backward propagation of a mini-batch of data through the neural network [51]. Some synaptic saliencies can even be calculated without any data [50]. Since the query time for synaptic saliency is negligibly short and it requires no initialization time, it does not add extra time to the optimization process.

2) CALCULATION OF SYNAPTIC SALIENCY OF A NEURAL NETWORK

The calculation of synaptic saliency of a neural network is performed in four stages. Each step has subtle differences depending on which synaptic saliency is calculated. The steps are described below in detail.

Step 1: Forward Propagate a Mini-batch of Data or Ones Vector Through the Neural Network

The first step to calculate the synaptic saliency of a neural network is to forward propagate a mini-batch of data or an all-ones matrix through the neural network. In order to calculate SNIP [48] or GRASP [49] synaptic saliency, a mini-batch of data has to be forward propagated whereas, to calculate SYNFLOW [50] synaptic saliency, an all-ones matrix having a dimension equal to the dimension of the input image has to be forward propagated. The forward propagation of a mini-batch of data yields a prediction vector.

Step 2: Calculate Loss

Calculation of loss for calculating SYNFLOW synaptic saliency is different from the calculation of loss for SNIP or GRASP synaptic saliency. Loss calculation for SNIP or GRASP synaptic saliency can be done using simple Mean Squared Error (MSE).

$$J(\theta) = \frac{1}{M} \sum_{i=1}^{M} \left(y_i - \hat{y}_i\right)^2 \tag{1}$$

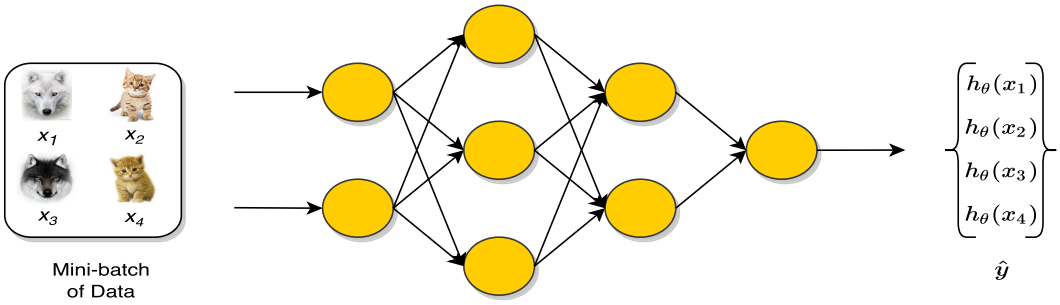
Here, $J(\theta)$ is the loss function. M is the number of forward propagated samples. y_i is the true label of i-th sample, and $\hat{y}_i$ is the predicted label of i-th sample.

The prediction vector is obtained by:

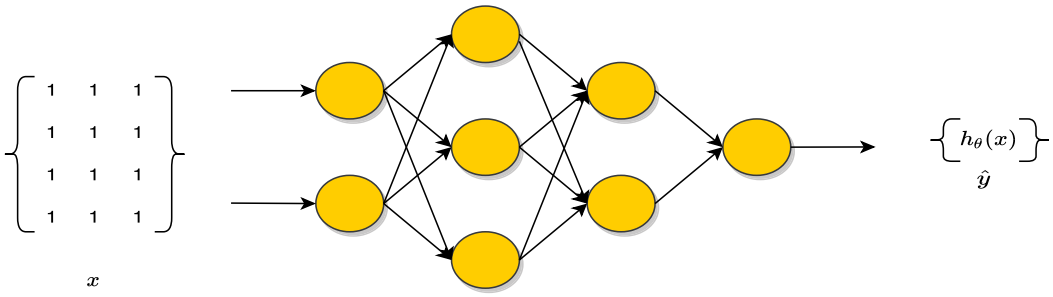
$$\hat{y} = h_\theta(x) \tag{2}$$

Here, $h_\theta(x)$ represents the hypothesis function parameterized by θ that maps between the inputs and the outputs of the neural network. MSE is not used when calculating SYNFLOW synaptic saliency since during calculation of SYNFLOW, a mini-batch of data is not forward propagated; rather, an all-ones matrix is forward propagated. Instead of calculating MSE, a new type of loss called synaptic loss is calculated. Synaptic loss is calculated by multiplying the absolute value of all the parameters of a neural network.

$$\mathcal{R}_{\mathrm{SF}} = \mathbb{1}^T \left(\prod_{l=1}^{L} \left| \theta^{[l]} \right| \right) \mathbb{1} \tag{3}$$

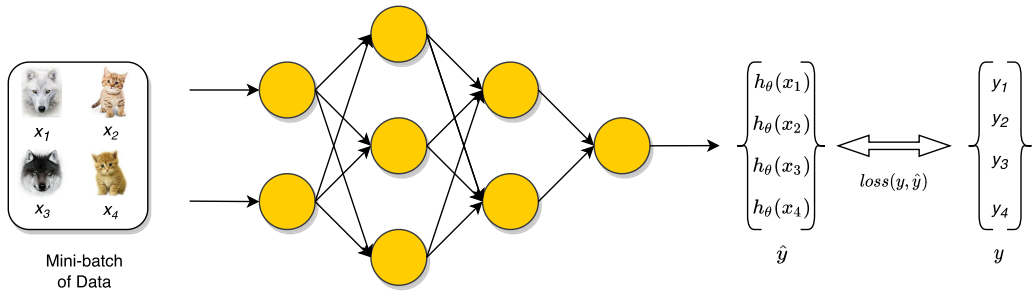


(a) Forward propagation of a mini-batch of data for calculating SNIP & GRASP synaptic saliency

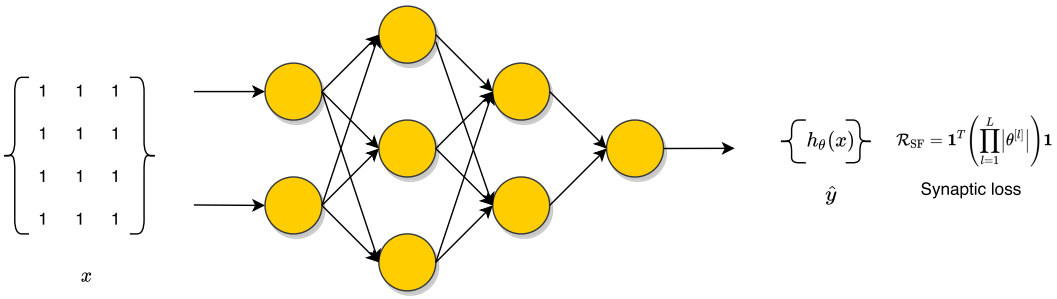


(b) Forward propagation of an all-ones matrix for calculating SYNFLOW synaptic saliency

FIGURE 3. Forward propagation.



(a) Calculation of loss for calculating SNIP & GRASP synaptic saliency



(b) Calculation of synaptic loss for calculating SYNFLOW synaptic saliency

FIGURE 4. Calculation of loss.

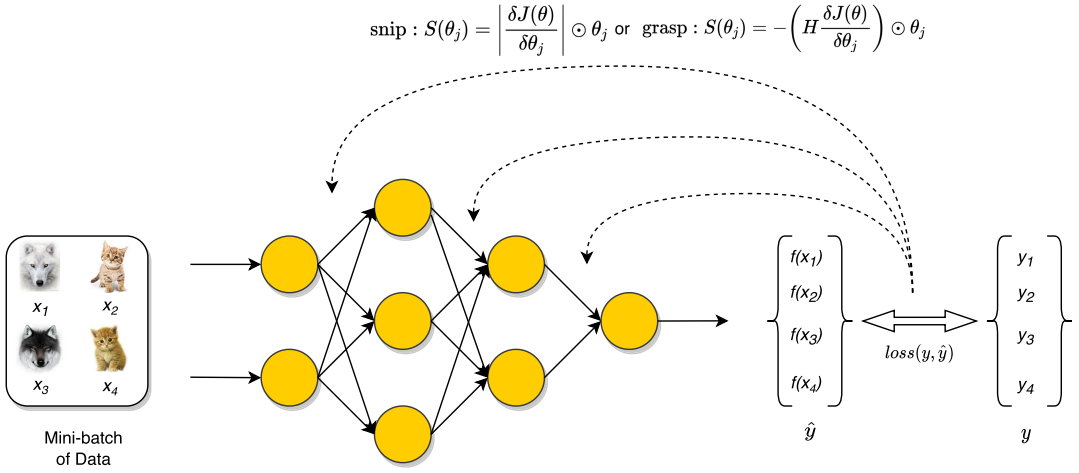
Here, $\mathcal{R}_{SF}$ is the synaptic loss, $\mathbb{1}$ is the all-ones matrix, $\theta^{[l]}$ is a vector containing the parameters or weights of layer l.

Step 3: Backpropagation of Loss to Calculate Synaptic Saliency of Each Parameter in the Network

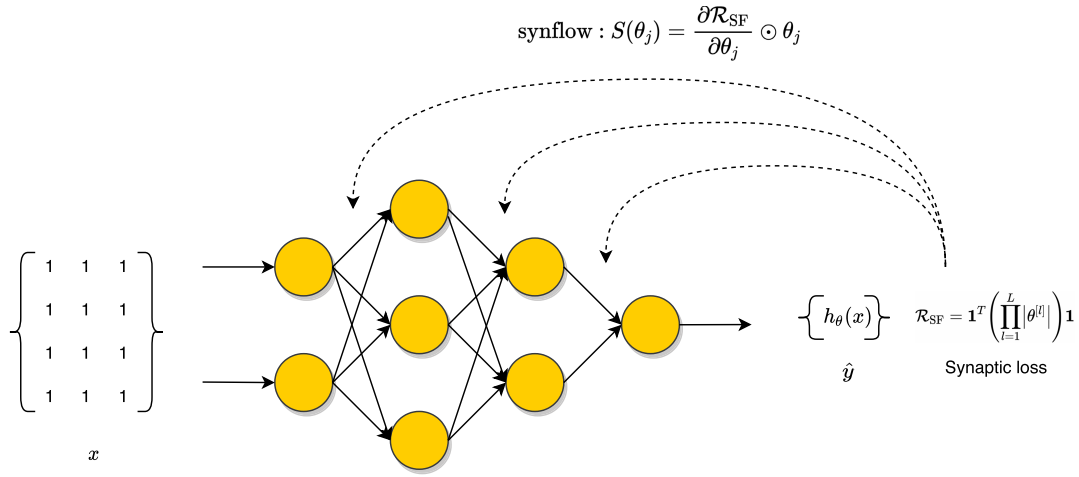
While training a neural network, backpropagation is done to update the parameters of the neural network. It is also called gradient update since each parameter is updated using the gradient of the loss with respect to that parameter.

$$\theta_j \longleftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{4}$$

Here, θ_j is the weight of j-th parameter, α is the learning rate, $J(\theta)$ is the loss, and $\frac{\partial}{\partial \theta_j} J(\theta)$ is the gradient of loss with respect to θ_j.



(a) Backpropagation of loss to calculate SNIP or GRASP synaptic saliency of each parameter in the Network



(b) Backpropagation of loss to calculate SYNFLOW synaptic saliency of each parameter in the Network

FIGURE 5. **Backpropagation of loss to calculate synaptic saliency of each parameter in the Network.**

Unlike standard backpropagation, weight is not updated while calculating synaptic saliency. Instead of updating the weight, the gradient of each weight is used to calculate the synaptic saliency for that weight. SNIP [48] synaptic saliency for a parameter is calculated by taking the Hadamard product of the absolute value of the gradient of the loss with respect to that parameter and the parameter itself.

$$\text{snip} : S(\theta_j) = \left|\frac{\delta J(\theta)}{\delta \theta_j}\right| \odot \theta_j \tag{5}$$

Here, $J(\theta)$ is the loss, $\frac{\delta J(\theta)}{\delta \theta_j}$ is the gradient of loss with respect to the parameter θ_j.

GRASP [49] synaptic saliency of a parameter is calculated by taking the hadamard product of the gradient of the loss and Hessian with the parameter itself.

$$\text{grasp} : S(\theta_j) = -\left(H \frac{\delta J(\theta)}{\delta \theta_j}\right) \odot \theta_j \tag{6}$$

Here, H represents the Hessian matrix that captures the dependencies between different weights of the neural network.

The SYNFLOW [50] synaptic saliency of a parameter is calculated by taking hadamard product of the gradient of the synaptic loss and the parameter itself.

$$\text{synflow} : S(\theta_j) = \frac{\partial \mathcal{R}_{\mathrm{SF}}}{\partial \theta_j} \odot \theta_j \tag{7}$$

Here, $\mathcal{R}_{\mathrm{SF}}$ is the synaptic loss, $\frac{\partial \mathcal{R}_{\mathrm{SF}}}{\partial \theta}$ is the gradient of the synaptic loss.

The visualization of the overall process of calculation of synaptic saliency for each parameter of a neural network is shown in Figure 3, Figure 4, and Figure 5.

Step 4: Synaptic Saliency of an Entire Neural Network

After calculating synaptic saliency for each network parameter, they are summed to calculate the synaptic saliency

of the entire neural network [51].

$$\mathcal{S}_{net} = \sum_{j}^{N} \mathcal{S}(\theta_j) \tag{8}$$

Here, S_{net} represents the synaptic saliency of a neural network, N is the number of parameters in the neural network, and θ_j is the j-th parameter of the neural network.

3) IDENTIFICATION OF PROMISING NEURAL NETWORK ARCHITECTURES LEVERAGING SYNAPTIC SALIENCY

This section will discuss how synaptic saliency can be used to identify promising architectures. The IGWO-SS algorithm uses synaptic saliency at two stages: initialization and new population generation. Top n architectures are selected from N random architectures during initialization by leveraging synaptic saliency. During new population generation, top n architectures are selected from N newly generated architecture by leveraging synaptic saliency. The synaptic saliency is used to estimate the performance of untrained neural networks. At first, the synaptic saliency of N neural networks are calculated. Then the architectures are ranked based on the magnitude of the calculated synaptic saliency. The architectures that are higher in rank are considered more promising than the architectures that are lower in rank. Only top n architectures are selected from the rank, and the rest $(N - n)$ are discarded. Since the calculation time of synaptic saliency is negligible, promising architectures are identified from a set of untrained architectures within seconds.

A simplified example showing the process of identification of promising architectures leveraging synaptic saliency is shown in Figure 6. It is observed from Figure 6 that - the process starts with a set of architectures. In the example shown in Figure 6, the process starts with $N = 5$ architectures. The architectures are either generated randomly or by the NAS algorithm. After the generation of $N = 5$ architectures, their synaptic saliency is calculated. Then the architectures are ranked based on the calculated synaptic saliency. The architectures with higher synaptic saliency are higher in rank, whereas those with lower synaptic saliency are lower. After creating the rank of $N = 5$ architectures, the top $n = 2$ architectures are identified as the promising architectures that will go into the next step of the algorithm, and the rest of the architectures are discarded.

The algorithm for the identification of promising architectures leveraging synaptic saliency is presented in Algorithm 1. The algorithm takes as input the number of neural network architectures n and N where n is equal to the population size of the NAS algorithm and N is set by the decision-maker. $n < N$ since n most promising architectures will be selected from larger a set of candidate architectures N. The algorithm also takes as an input $\boldsymbol{X}$, which is a vector containing the configurations of N architectures generated randomly or generated by the NAS algorithm. The algorithm returns the configurations of top n architectures $\boldsymbol{x}$ from the configurations of N architectures X. In line 1 of the algorithm, a set of neural network architectures $\boldsymbol{X}$ is obtained randomly or from the NAS algorithm. In line 2 of the algorithm, the synaptic saliency of each of the N neural network architectures is calculated. The architectures are ranked based on calculated synaptic saliency in line 3. In line 4, the top n architectures from N neural network architectures are selected based on the rank. The configurations of top n architectures are returned in line 5.

Algorithm 1: Get-Population-Using-Synaptic-Saliency

Input: Number of neural networks n and N (n<N), Set of neural network architectures generated randomly or generated by the NAS algorithm, $\boldsymbol{X}$

Output: $\boldsymbol{x}$

1 Obtain a set of neural network architectures $\boldsymbol{X}(i = 1, 2, \ldots, N)$
2 Calculate synaptic saliency of N obtained neural networks
3 Rank the neural networks based on synaptic saliency
4 Select top n neural networks from N neural networks based on synaptic saliency as the selected population of more promising architectures $\boldsymbol{x}(i = 1, 2, \ldots, n)$
5 return $\boldsymbol{x}$

4) STEPS IN THE IGWO-SS ALGORITHM

This section of the paper presents the IGWO-SS algorithm for NAS. Unlike the Standard GWO algorithm, in the IGWO-SS algorithm, both the initialization and new population generation are done by leveraging synaptic saliency. The synaptic saliency is used to create a rank between the architectures. Architectures that are higher in rank are considered more promising than those lower in rank. The IGWO-SS algorithm only focuses on the more promising architectures. It trains the more promising architectures and discards the rest. Since the calculation of synaptic saliency is a speedy process, the ranking can be done in a short period. Overall, the algorithm requires training much fewer models to achieve similar or even better accuracy than other algorithms, making it fast and reliable.

The flowchart for the IGWO-SS algorithm is presented in Figure 7. Each step of the IGWO-SS algorithm is described in detail below.

Initialization using Synaptic Saliency

During initialization, the positions of N grey wolves are initialized randomly. Each grey wolf corresponds to a specific neural network architecture. n grey wolves are selected from the N grey wolves using the following 3 steps -

1) Calculate synaptic saliency of N grey wolves or neural networks.
2) Rank the grey wolves or neural networks based on calculated synaptic saliency.
3) Select top n grey wolves based on synaptic saliency as the initial population of grey wolves.

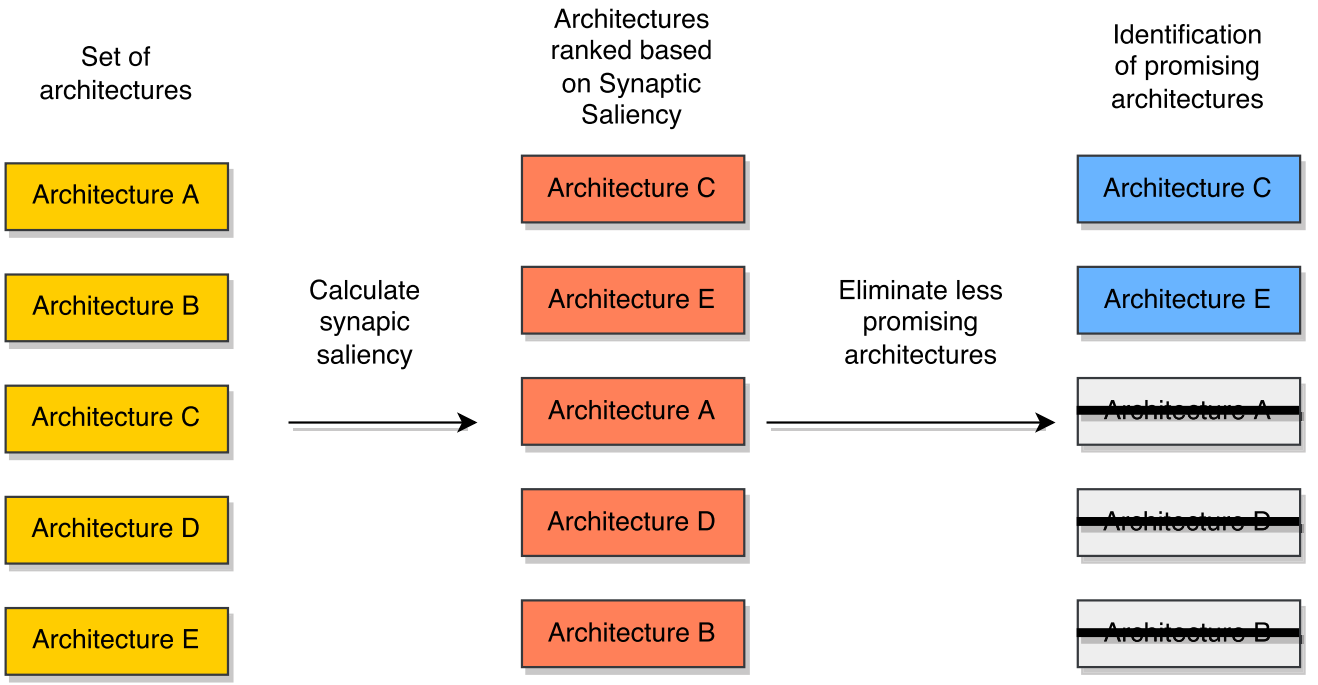


FIGURE 6. Identification of promising architectures leveraging synaptic saliency.

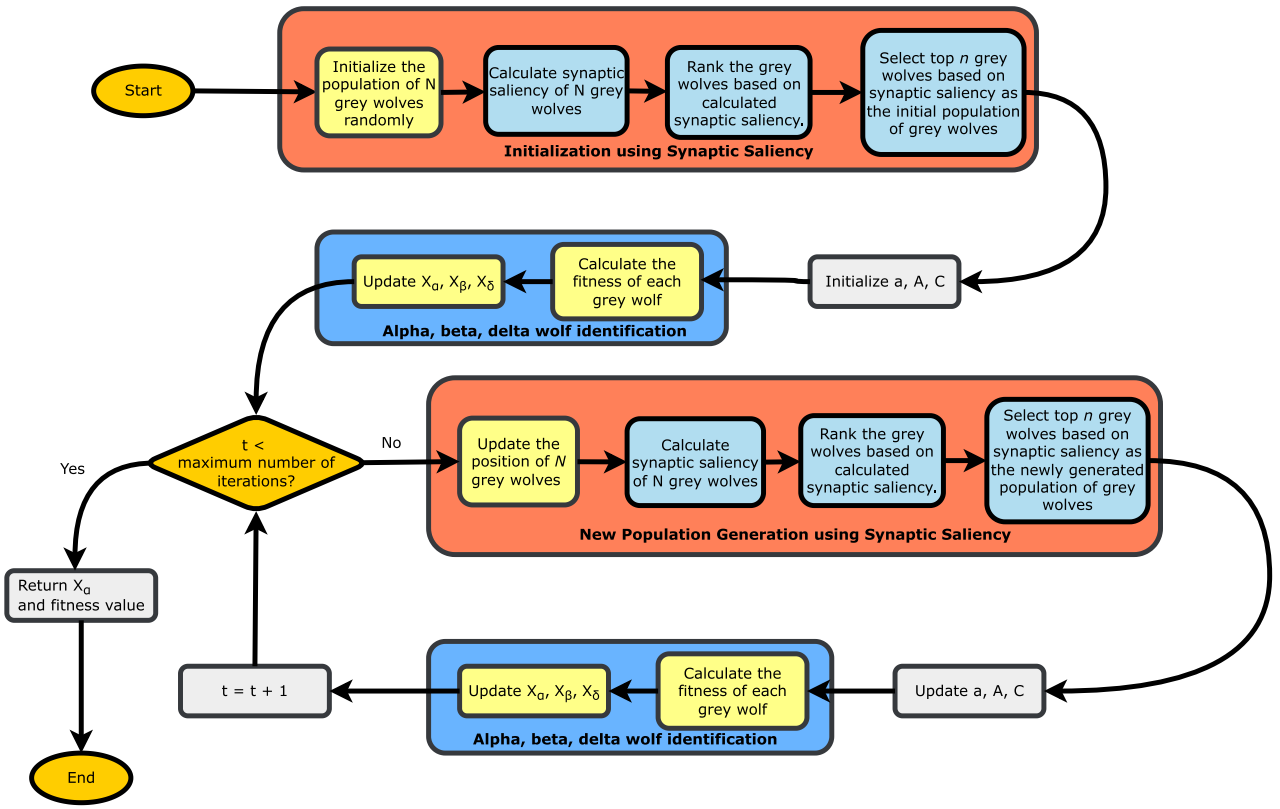


FIGURE 7. Flowchart for Improved Grey Wolf Optimization Algorithm Using Synaptic Saliency (IGWO-SS).

This n grey wolves act as the initial population of the IGWO-SS algorithm.

Alpha, Beta and Delta Wolf Identification from the Initial Population

After obtaining the initial population using synaptic saliency, the simulation parameters $\boldsymbol{a}$, $\boldsymbol{A}$, and $\boldsymbol{C}$ are initialized. The following equations are used to calculate $\boldsymbol{A}$ and $\boldsymbol{C}$:

$$\boldsymbol{A} = 2\boldsymbol{a} \cdot \boldsymbol{r}_1 - \boldsymbol{a} \tag{9}$$

$$\boldsymbol{C} = 2 \cdot \boldsymbol{r}_2 \tag{10}$$

Here, $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are two vectors that can take any random values between 0 and 1. $\boldsymbol{a}$ is a vector whose components decrease linearly from 2 to 0 during iterations.

Then the fitness of the initial population of grey wolves is calculated to identify the α, β, and δ wolves. The fitness can be the loss or accuracy of the neural network. In the search process, the α, β, and δ wolves guide the search process by guiding all other wolves. After each iteration, the wolf that is closest to the prey is set to be the α wolf, the wolf that is the second closest to the prey is set to be the β wolf and the wolf that is the third closest to the prey is set to be the δ wolf. If the loss is used as fitness, the grey wolf or neural network having the lowest loss is the α wolf, the grey wolf or neural network having the second-lowest loss is the β wolf, and the grey wolf or neural network having the third-lowest loss is the δ wolf. These α, β, and δ wolves will guide the search process.

New Population Generation using Synaptic Saliency

In order to generate positions of new wolves, at first, the distance of the current wolf from α, β, and δ wolf are calculated.

$$\left.\begin{aligned} \boldsymbol{D}_\alpha &= |\boldsymbol{C}_1 \cdot \boldsymbol{X}_\alpha - \boldsymbol{X}| \\ \boldsymbol{D}_\beta &= |\boldsymbol{C}_2 \cdot \boldsymbol{X}_\beta - \boldsymbol{X}| \\ \boldsymbol{D}_\delta &= |\boldsymbol{C}_3 \cdot \boldsymbol{X}_\delta - \boldsymbol{X}| \end{aligned}\right\} \tag{11}$$

Here, $\boldsymbol{D}_\alpha$ is the distance of the current wolf from the α wolf, $\boldsymbol{D}_\beta$ is the distance of the current wolf from the β wolf, and $\boldsymbol{D}_\delta$ is the distance of the current wolf from the δ wolf.

Using these distances, three new positions can be obtained for each wolf, which are then averaged to get the new position for the next iteration.

$$\left.\begin{aligned} \boldsymbol{X}_1 &= \boldsymbol{X}_\alpha - \boldsymbol{A}_1 \cdot (\boldsymbol{D}_\alpha) \\ \boldsymbol{X}_2 &= \boldsymbol{X}_\beta - \boldsymbol{A}_2 \cdot (\boldsymbol{D}_\beta) \\ \boldsymbol{X}_3 &= \boldsymbol{X}_\delta - \boldsymbol{A}_3 \cdot (\boldsymbol{D}_\delta) \end{aligned}\right\} \tag{12}$$

$$\boldsymbol{X}(t+1) = \frac{\boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3}{3} \tag{13}$$

During the generation of new grey wolves, instead of generating only n numbers of grey wolves, N numbers of grey wolves are generated ($n < N$). Then n promising grey wolves are selected from these N grey wolves using the following 3 steps -

1) Calculate synaptic saliency for N grey wolves or neural networks.
2) Rank the grey wolves or neural networks based on calculated synaptic saliency.
3) Select top n grey wolves from N grey wolves based on synaptic saliency as the newly generated population.

Alpha, Beta and Delta Wolf Identification from the New Generated Population

After obtaining the newly generated population of grey wolves or neural networks using synaptic saliency, simulation parameters $\boldsymbol{a}$, $\boldsymbol{A}$, and $\boldsymbol{C}$ are updated. Then the α, β, and δ wolves are identified again by calculating the fitness of each grey wolf.

Termination Condition

The algorithm continues to run until the maximum number of iterations is reached. When the maximum number of iterations is reached, the position and fitness value of the α wolf is returned.

Algorithm 2: Improved Grey Wolf Optimization based on Synaptic Saliency (IGWO-SS)

Input: Number of grey wolves n and N (n<N), Maximum number of iterations T
Output: $\boldsymbol{X}_\alpha$, $fitness(\boldsymbol{X}_\alpha)$

```
1  Initialize N grey wolves population X
2  X = GET-POPULATION-USING-SYNAPTIC-SALIENCY(n,N,X)
3  Initialize a, A and C
4  Calculate the fitness of each grey wolf
5  X_α = position of the best grey wolf
6  X_β = position of the second-best grey wolf
7  X_δ = position of the third-best grey wolf
8  while t ≤ T do
9      Update the positions of N grey wolves X
10     X = GET-POPULATION-USING-SYNAPTIC-SALIENCY(n,N,X)
11     Update a, A and C
12     Calculate the fitness of each grey wolf
13     Update X_α, X_β, X_δ
14     t = t + 1
15 return X_α, fitness(X_α)
```

5) THE IGWO-SS ALGORITHM

The IGWO-SS algorithm is presented in Algorithm 2. The algorithm takes as input the number of grey wolves n and N. The N number of grey wolves act as the proxy agents from which n the number of grey wolves are selected. The algorithm also takes the maximum number of iterations T as input. The algorithm returns as output the position of the α wolf, $\boldsymbol{X}_\alpha$ and the fitness of the α wolf, $fitness(\boldsymbol{X}_\alpha)$.

In line 1 of the algorithm, a population of N grey wolves is initialized. In line 2 of the algorithm, the GET-POPULATION-USING-SYNAPTIC-SALIENCY (n,N,$\boldsymbol{X}$) algorithm is called to obtain the configuration of n promising neural networks from the configuration of N neural networks. The simulation parameters $\boldsymbol{a}$, $\boldsymbol{A}$, and $\boldsymbol{C}$ are initialized in line 3. The fitnesses of the wolves are calculated by training n neural networks in line 4. From line 5-7, α, β, and δ wolves are identified. A while loop starts in line 8, which runs till the maximum number of iterations T is reached. As seen in line 9, the positions of N grey wolves are updated during each iteration. Then the GET-POPULATION-USING-SYNAPTIC-SALIENCY(n,N,$\boldsymbol{X}$) algorithm is again called to obtain the configurations of n promising neural networks

from the configurations of N neural networks. The simulation parameters $\boldsymbol{a}$, $\boldsymbol{A}$, and $\boldsymbol{C}$ are initialized in line 11. In line 12, the fitness of each grey wolf is calculated, and in line 13, the α, β, and δ wolves are identified. After T iterations, the position (configuration) of the α wolf and the fitness value (accuracy) is returned.

6) COMPONENTS OF NAS FOR THE IGWO-SS ALGORITHM

The flowchart of this study is presented in Figure 8. According to Elsken *et al.* [9], NAS methods have three components - search space, search strategy, and performance estimation strategy. In our experiments, the NAS-Bench-201 search space [53] is used as the search space and the IGWO-SS algorithm is used as the search strategy to perform NAS. Three datasets are used to validate the performance of the IGWO-SS algorithm - CIFAR-10, CIFAR-100, and ImageNet-16-120. To estimate the performance of the neural networks, the performance data from NAS-Bench-201 obtained by full training of the neural networks and the relative ranking of neural network architectures based on synaptic saliency are used. The performance of the IGWO-SS algorithm is compared with several bioinspired algorithm that includes - Particle Swarm Optimization (PSO) [65], Bat Algorithm (BA) [66], Whale Optimization Algorithm (WOA) [67], Differential evolution (DE) [68], and Genetic Algorithm (GA) [69] and the state-of-the-art algorithms for NAS that includes - REA [45], RS [70], RL [71], BOHB [72], DARTSV1 [73], DARTSV2 [73], GDAS [74], SETN [75], and ENAS [43].

IV. EXPERIMENTAL RESULTS

The experimental results section is divided into two parts. In the first part, the result of different experiments performed to evaluate the efficacy of synaptic saliency to improve NAS are presented. In the second part, different experiments to evaluate the effectiveness of the IGWO-SS algorithm are presented.

A. EXPERIMENTS EVALUATING THE EFFICACY OF SYNAPTIC SALIENCY TO IMPROVE NAS

In this section, the results of different experiments that were performed to evaluate the efficacy of synaptic saliency to improve NAS are presented. In order to calculate the correlations, 15,625 neural network architectures from the NAS-Bench-201 were used.

1) CORRELATION BETWEEN SYNAPTIC SALIENCY AND ACCURACY

The correlation between synaptic saliency and accuracy of neural networks is presented in Table 2. Kendall's Tau, Pearson, and Spearman correlation coefficients for SNIP, GRASP, and SYNFLOW synaptic saliency with accuracy for CIFAR-10, CIFAR-100, and ImageNet16-120 are calculated. It is observed from the table that SYNFLOW synaptic saliency is more correlated with accuracy compared to SNIP and GRASP on all three datasets. Hence, it is more reasonable to use SYNFLOW synaptic saliency to identify untrained promising neural networks.

TABLE 2. Correlation between synaptic saliency and accuracy.

Dataset	Synaptic Saliency	Correlation Coefficient		
		Kendall's Tau	Pearson	Spearman
CIFAR-10	SNIP	0.43	0.03	0.60
	GRASP	0.35	0.00	0.51
	SYNFLOW	0.54	0.16	0.74
CIFAR-100	SNIP	0.47	0.04	0.64
	GRASP	0.39	-0.01	0.55
	SYNFLOW	0.57	0.19	0.76
ImageNet16-120	SNIP	0.43	0.05	0.58
	GRASP	0.39	-0.01	0.55
	SYNFLOW	0.56	0.16	0.75

TABLE 3. Correlation between synaptic saliency and model size.

Dataset	Synaptic Saliency	Correlation Coefficient		
		Kendall's Tau	Pearson	Spearman
CIFAR-10	SNIP	0.47	0.18	0.63
	GRASP	0.36	0.01	0.50
	SYNFLOW	0.59	0.33	0.75
CIFAR-100	SNIP	0.47	0.18	0.63
	GRASP	0.36	-0.00	0.50
	SYNFLOW	0.59	0.33	0.75
ImageNet16-120	SNIP	0.46	0.18	0.62
	GRASP	0.38	0.02	0.53
	SYNFLOW	0.58	0.31	0.75

TABLE 4. Correlation between synaptic saliency and FLOPS.

Dataset	Synaptic Saliency	Correlation Coefficient		
		Kendall's Tau	Pearson	Spearman
CIFAR-10	SNIP	0.47	0.18	0.63
	GRASP	0.36	0.01	0.50
	SYNFLOW	0.59	0.33	0.75
CIFAR-100	SNIP	0.47	0.18	0.63
	GRASP	0.36	-0.00	0.50
	SYNFLOW	0.59	0.33	0.75
ImageNet16-120	SNIP	0.46	0.18	0.62
	GRASP	0.38	0.02	0.53
	SYNFLOW	0.58	0.31	0.75

2) CORRELATION BETWEEN SYNAPTIC SALIENCY AND MODEL SIZE

In order to find out if there is any relationship between synaptic saliency and the size of the neural network or model size, the correlation between synaptic saliency and model size is also calculated. Table 3 confirms that there is a correlation between synaptic saliency and model size. The models which are larger have high synaptic saliency. As a result, while ranking models based on synaptic saliency, the larger models might be preferred over the smaller ones.

3) CORRELATION BETWEEN SYNAPTIC SALIENCY AND FLOPS

The correlation between synaptic saliency and FLOPS is presented in Table 4. It is observed from Table 4 that the

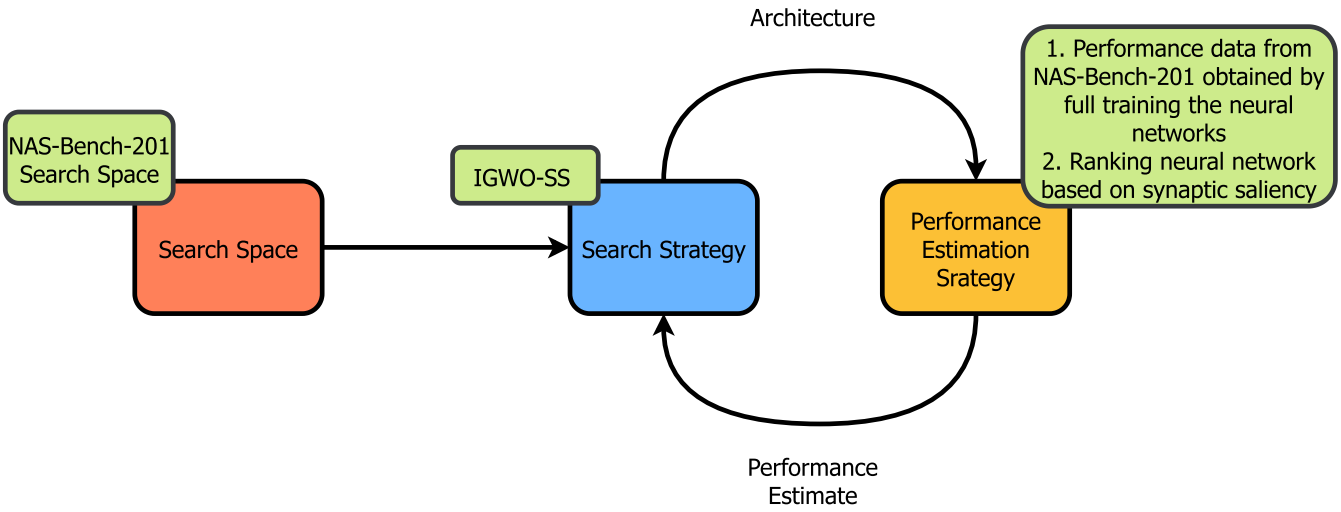


FIGURE 8. Flowchart of this study.

correlations between synaptic saliency and FLOPS are similar to the correlation between synaptic saliency and model size. It indicates a correlation between model size and FLOPS.

4) COMPARISON OF SYNAPTIC SALIENCY WITH OTHER PERFORMANCE PREDICTORS

Spearman correlation coefficient with accuracy of three synaptic saliency along with some other performance predictors from the literatures that includes Fisher [51], [76], Jacob covariance [52], Grad norm [51] on NAS-Bench-201 are shown in Figure 9. The result achieved is similar to [51]. It is observed from the figure that SYNFLOW has the highest correlation with accuracy. Jacob covariance has the second-highest correlation. The correlation of SNIP and Grad norm is quite similar. Fisher has the lowest correlation.

B. EXPERIMENTS EVALUATING THE PERFORMANCE OF THE IGWO-SS ALGORITHM FOR NAS

In this section, different experiments that are performed to evaluate the performance of the proposed IGWO-SS algorithm for NAS are presented. The experimental setting, the comparison of standard GWO algorithm and IGWO-SS algorithm with different synaptic saliency, the comparison of IGWO-SS algorithm with other bio-inspired and state-of-the-art NAS algorithms are presented in this section of the paper.

1) EXPERIMENTAL SETUP

The simulation parameters for the IGWO-SS algorithm are presented in Table 5. During the experiments, the number of search agents (n) varied between 10, 20, and 30. Based on the number of search agents, 100, 200, or 300 models were trained. The number of proxy agents (N) was set to 1000. Each search agent and proxy agent represents an architecture. The number of proxy agents is higher than the number of search agents ($N > n$). Instead of training N neural networks during the optimization, n neural networks are trained. These n neural networks are selected from N neural networks based on synaptic saliency. The maximum number of iterations for each algorithm was set to 10. Each algorithm was run 50 times, and a graph is plotted, which shows the mean and standard deviation of the best test accuracy obtained after training a certain number of models for all runs. The mean and standard deviation of the best neural network model for each algorithm in 50 runs are also presented in different tables.

TABLE 5. Simulation parameters for experiments.

Parameter	Value
Number of search agents (n)	10, 20, 30
Number of proxy agents (N)	1000
Maximum number of trained models	100, 200, 300
Number of Iterations	10
Number of Runs	50

2) IGWO-SS ALGORITHM WITH DIFFERENT SYNAPTIC SALIENCY

In this section of the paper, the performance of the IGWO-SS algorithm with three different synaptic saliency - SNIP, GRASP, and SYNFLOW are presented. The synaptic saliency are used to rank between N neural network architectures based on how promising they seem. Based on this rank, the top n architectures are selected for training, and the rest are discarded. The performances of the IGWO-SS algorithm with different synaptic saliency are shown in Figure 10.

The number of search agents (n) was set to 10, 20, or 30. Based on the number of search agents, the maximum number of trained models was 100, 200, or 300. The average best test accuracy vs. the number of models trained are plotted in

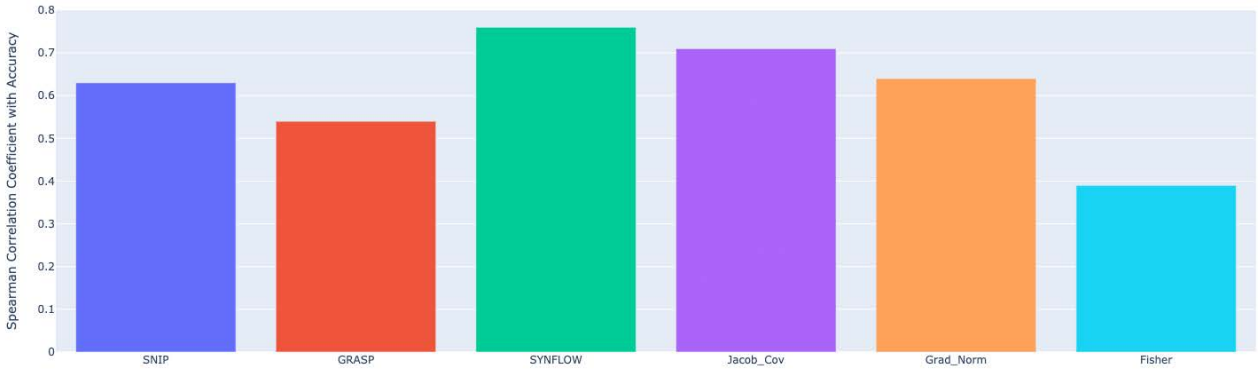


FIGURE 9. Comparison of synaptic saliency with other performance predictors.

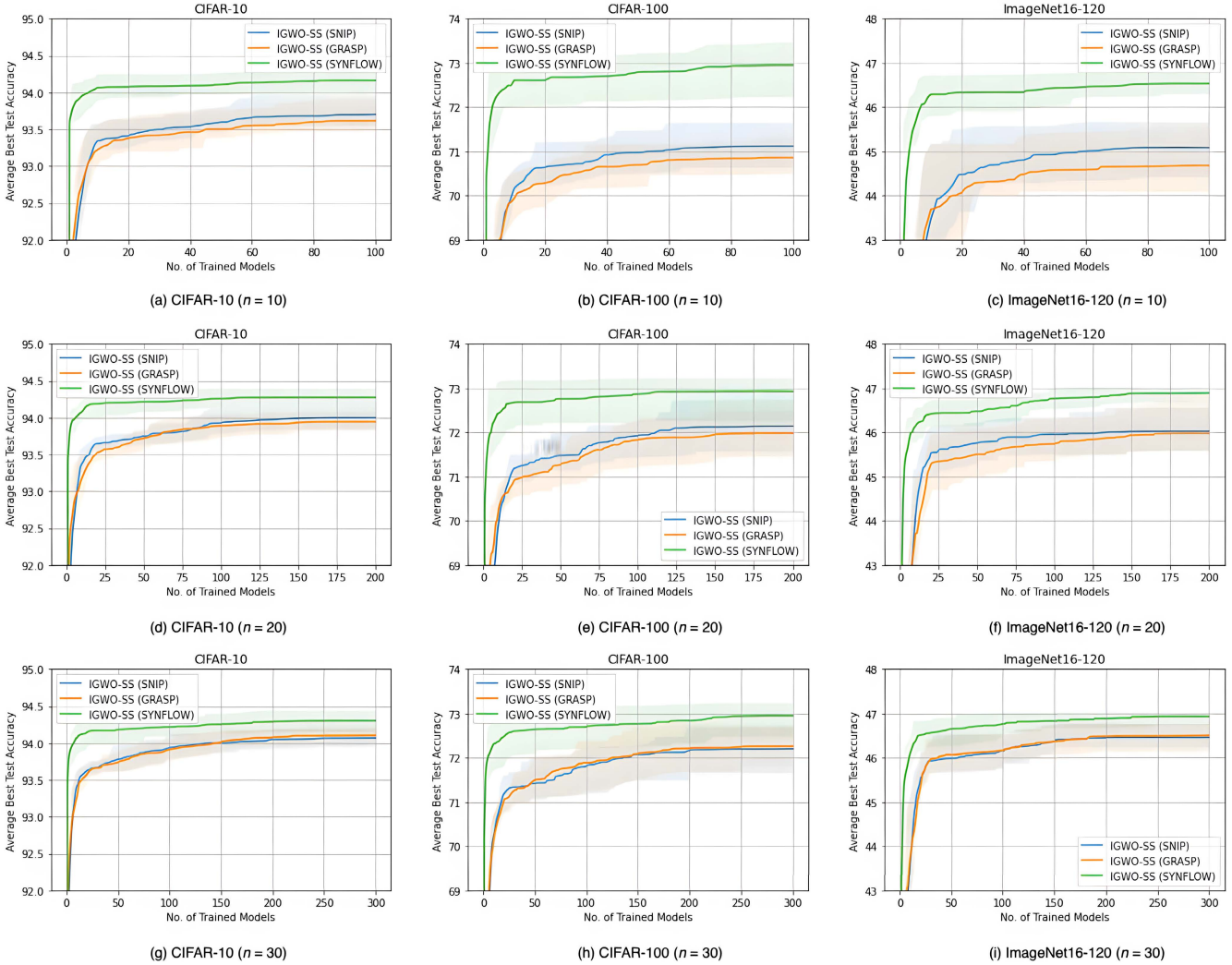


FIGURE 10. IGWO-SS algorithm with different synaptic saliency.

the graph. It is observed from the graph that, for any number of search agents and all datasets, the IGWO-SS algorithm with SYNFLOW outperforms the IGWO-SS algorithm with SNIP or GRASP synaptic saliency. It is also observed from the figure that, initially, the curve for IGWO-SS with SYNFLOW is very steep, which means the algorithm achieves

TABLE 6. Mean and standard deviation of test accuracy for IGWO-SS algorithm with different synaptic saliency for different numbers of search agents.

No. of Search Agents	Method	Test Accuracy (%)		
		CIFAR-10	CIFAR-100	ImageNet16-120
10	IGWO-SS (SNIP)	93.70±0.30	71.11±0.75	45.08±0.90
	IGWO-SS (GRASP)	93.61±0.39	70.85±0.72	44.67±1.30
	IGWO-SS (SYNFLOW)	**94.16±0.20**	**72.94±0.81**	**46.53±0.45**
20	IGWO-SS (SNIP)	94.00±0.20	72.14±0.64	46.02±0.64
	IGWO-SS (GRASP)	93.94±0.22	71.97±0.70	45.97±0.72
	IGWO-SS (SYNFLOW)	**94.27±0.17**	**72.92±0.59**	**46.89±0.41**
30	IGWO-SS (SNIP)	94.06±0.16	72.19±0.74	46.45±0.47
	IGWO-SS (GRASP)	94.10±0.18	72.26±0.69	46.50±0.47
	IGWO-SS (SYNFLOW)	**94.30±0.18**	**72.95±0.70**	**46.92±0.46**

high accuracy very fast by training only a few models. The final converged curve for IGWO-SS with SYNFLOW is also higher compared to the other two, which means, after the search process is finished, the IGWO-SS with SYNFLOW found architecture with better performance compared to the other two.

The mean and standard deviation of accuracy for the IGWO-SS algorithm with different synaptic saliency for different numbers of search agents are presented in Table 6.

It is observed from the table that when the number of search agents (n) is 10 or 20, IGWO-SS with SNIP performs better than IGWO-SS with GRASP in terms of mean accuracy in all three datasets. When the number of search agents (n) is 30, IGWO-SS with GRASP performs better than IGWO-SS with SNIP. With the increase in the number of search agents, IGWO-SS with GRASP begins to perform better, however, the IGWO-SS with SYNFLOW outperforms the other two for any number of search agents.

3) ABLATION STUDY: COMPARISON OF PERFORMANCE OF THE IGWO-SS WITH STANDARD GWO

In this section, the performance of the IGWO-SS algorithm is compared with the performance of the Standard GWO algorithm. Unlike the Standard GWO algorithm, which trains all neural networks, the IGWO-SS algorithm uses synaptic saliency to identify the more promising neural networks and only trains those neural networks. It speeds up the overall search process as the training of numerous less promising neural networks is skipped.

In Figure 11, the comparison between IGWO-SS with SYNFLOW and Standard GWO algorithm is presented in terms of the average best test accuracy obtained after training a certain number of models for 10, 20, and 30 search agents. It is observed from the figure that the IGWO-SS algorithm achieves better final average best test accuracy compared to the Standard GWO algorithm for any number of search agents. The IGWO-SS algorithm also achieves higher average best test accuracy by training fewer models than the Standard GWO algorithm. As the model's training is the most expensive part, and the IGWO-SS algorithm requires training much fewer models than the GWO algorithm to achieve similar or even better performance, it is, therefore, much faster. It is observed from Figure 11 that -

- When the number of models trained is 100, the IGWO-SS algorithm only requires training about 5 neural networks to achieve better accuracy than the Standard GWO algorithm after training 100 neural networks for CIFAR-10, CIFAR-100, and ImageNet16-120 dataset.
- When the number of models trained is 200, the IGWO-SS algorithm only requires training about 10 neural networks to achieve better accuracy than the Standard GWO algorithm after training 200 neural networks for CIFAR-10, CIFAR-100, and ImageNet16-120 dataset.
- When the number of models trained is 300, the IGWO-SS algorithm only requires training about 15 neural networks to achieve better accuracy than the GWO algorithm after training 300 neural networks for CIFAR-10, CIFAR-100, and ImageNet16-120 dataset.

The IGWO-SS algorithm is almost 20X faster compared to the Standard GWO algorithm, considering the fact that the calculation of synaptic saliency is very fast and assuming that all models require the same amount of time to train.

TABLE 7. Mean and standard deviation of test accuracy for Standard GWO and IGWO-SS algorithm with different numbers of search agents.

No. of Search Agents	Method	Test Accuracy (%)		
		CIFAR-10	CIFAR-100	ImageNet16-120
10	GWO	93.62±0.31	70.84±0.99	44.69±1.10
	IGWO-SS	**94.16±0.20**	**72.94±0.81**	**46.53±0.45**
20	GWO	93.94±0.28	71.68±0.85	45.73±0.96
	IGWO-SS	**94.27±0.17**	**72.92±0.59**	**46.89±0.41**
30	GWO	93.99±0.21	71.92±0.85	46.03±0.68
	IGWO-SS	**94.30±0.18**	**72.95±0.70**	**46.92±0.46**

The mean and standard deviation of the accuracy of the best model found using the Standard GWO algorithm and the IGWO-SS algorithm on three datasets - CIFAR-10, CIFAR-100, and ImageNet16-120 are shown in Table 7. Each algorithm was run 50 times. The test accuracy of the best neural network is stored during each run. Then the mean and standard deviation of accuracy of the 50 best neural network models found during 50 runs of each algorithm is calculated. From Table 7 it is observed that the mean accuracy of the best model obtained by IGWO-SS with SYNFLOW synaptic saliency is more on all three datasets compared to the Standard GWO algorithm. In the case of the Standard GWO algorithm, the mean accuracy increases with the increase of the number of trained models for all three datasets. However, in the case of the IGWO-SS algorithm, the mean accuracy increases for the CIFAR-10 and ImageNet16-120 datasets. For the CIFAR-100 dataset, the mean accuracy decreases first and then increases again. It is also observed that with the increase of search agents or the number of models trained, the difference between mean accuracies for the IGWO-SS algorithm and the GWO algorithm decreases, however, the IGWO-SS still out-

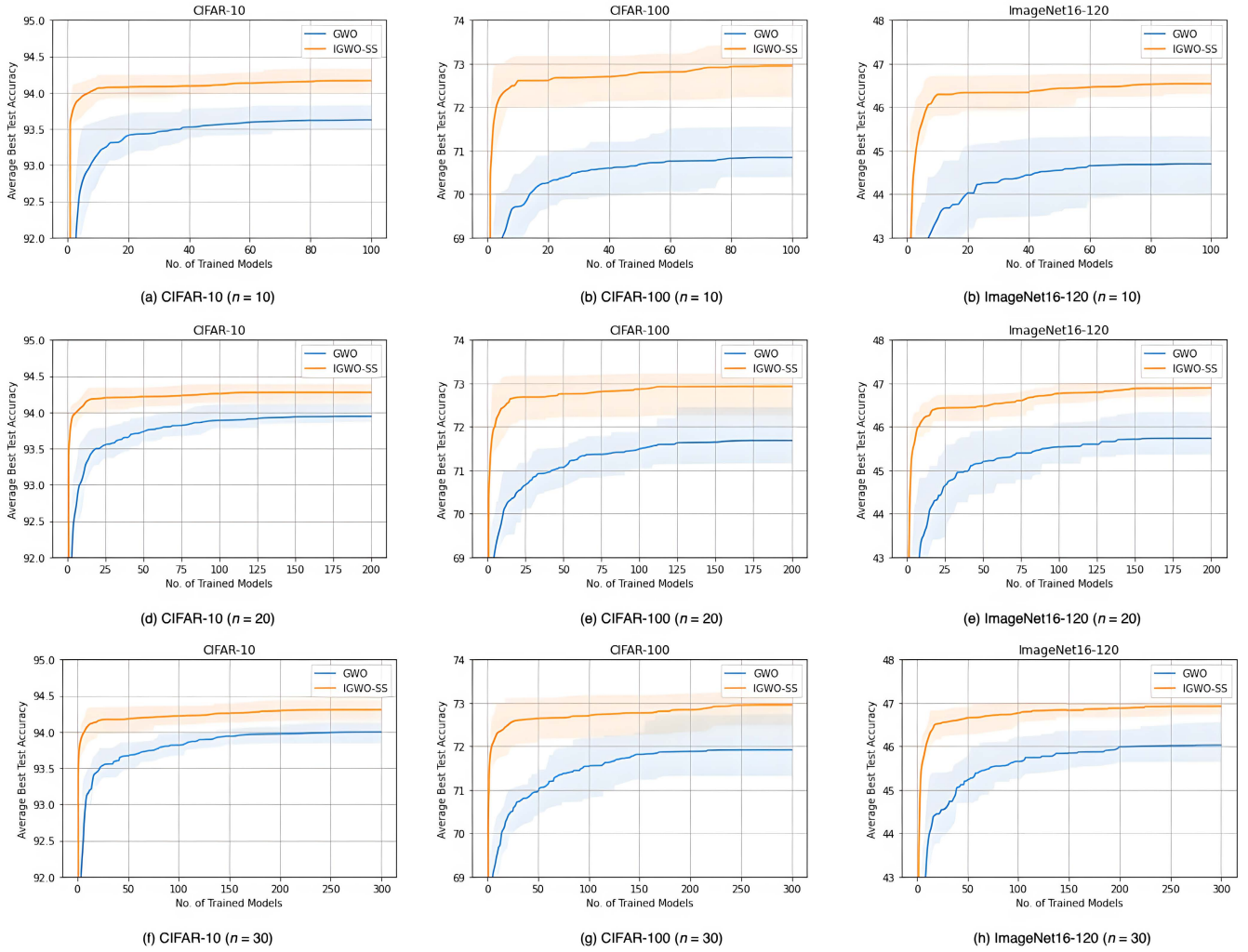


FIGURE 11. Comparison between IGWO-SS algorithm with Standard GWO algorithm.

performs the GWO algorithm. As more and more models are trained, the difference will further decrease as more regions of the search space will be explored. Nevertheless, training numerous models is very expensive and time-consuming. The IGWO-SS algorithm achieves performance similar to the Standard GWO algorithm after training 300 models by training only a few models. Hence, it is much faster.

4) COMPARISON OF THE IGWO-SS ALGORITHM WITH OTHER BIO-INSPIRED ALGORITHMS

In this section of the paper, the comparison of the IGWO-SS algorithm with other bio-inspired algorithms is presented. The IGWO-SS algorithm is compared with five bio-inspired algorithms - PSO, BA, WOA, DE, and GA. The algorithms are compared on three datasets - CIFAR-10, CIFAR-100, ImageNet16-120. Each algorithm used the same number of search agents - 10, 20, or 30. The maximum number of trained models for each algorithm was - 100, 200, or 300. The algorithms were run 50 times.

Figure 12 presents the average best test accuracy obtained by the algorithms after training a certain number of models. It is observed from Figure 12 that the IGWO-SS algorithm outperforms the other algorithms in all three datasets. The IGWO-SS algorithm requires training much fewer models than other algorithms to achieve a similar or even better result. When the number of search agents is less, the IGWO-SS algorithm outperforms the other algorithms by a significant margin. With the increase of search agents, the other algorithms perform slightly better, however the IGWO-SS algorithm still outperforms those. Among the other algorithms, PSO performs well when the number of search agents is more. However, when the number of search agents is less, it does not perform well. GA works well when the number of search agents is less. However, when the number of search agents is more, the other algorithms outperform GA.

The main takeaway from these graphs is how efficient the IGWO-SS algorithm is compared to the other algorithms. The algorithm manages to find a very well-performing neural network, even before training 20 neural networks. It requires

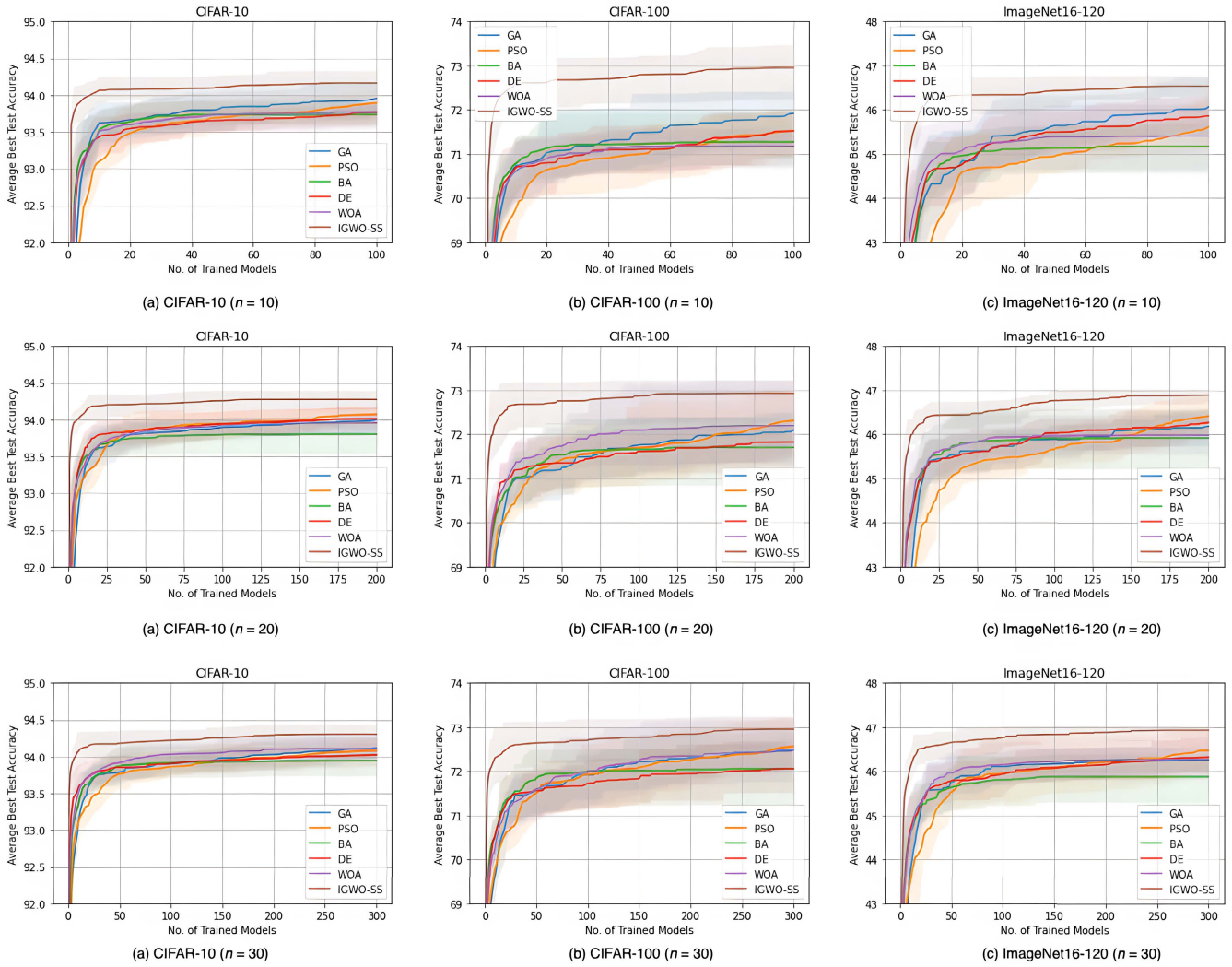


FIGURE 12. Comparison of IGWO-SS algorithm with other bioinspired algorithms.

training about 10x fewer models to achieve a similar or better performing neural network than the other algorithms.

The mean and standard deviation of test accuracy for different bioinspired algorithms with different numbers of search agents are presented in Table 8. The table shows that the mean accuracy of the final neural network model obtained using IGWO-SS algorithm is higher than the other algorithms in all three datasets for any number of search agents. For different numbers of agents and different datasets, GA and PSO interchangeably came in second.

- When the number of search agents is 10, GA achieves the second-highest final mean accuracy for all three datasets. The mean accuracies achieved by the IGWO-SS algorithm are 0.21%, 1.03%, and 0.47% more than the mean accuracies achieved by the GA for CIFAR-10, CIFAR-100, and ImageNet16-120, respectively.
- When the number of search agents is 20, PSO achieves the second-highest final mean accuracy for all three datasets. The mean accuracies achieved by the IGWO-SS algorithm are 0.2%, 0.61%, and 0.48% more than PSO for CIFAR-10, CIFAR-100, and ImageNet16-120, respectively.
- When the number of search agents is 30, GA achieves the second-highest mean accuracy for CIFAR-10 and ImageNet16-120, and PSO achieves the second-highest mean accuracy for the CIFAR-100 dataset. The mean accuracies achieved by the IGWO-SS algorithm are 0.19% and 0.46% more than the mean accuracies achieved by GA for the CIFAR-10 and ImageNet16-120 datasets and 0.39% more than the mean accuracies achieved by PSO for the CIFAR-100 dataset.

The mean accuracies achieved by the BA, DE, and WOA were always less than the mean accuracies achieved by IGWO-SS, PSO, and GA. The IGWO-SS algorithm outperformed all other algorithms and required training 10x fewer models compared to the other algorithms.

5) COMPARISON OF IGWO-SS ALGORITHM WITH OTHER STATE-OF-THE-ART NAS ALGORITHMS

The IGWO-SS algorithm is compared to state-of-the-art NAS algorithms that includes - RL [71], RS [70], BOHB [72],

TABLE 8. Mean and standard deviation of test accuracy for different bio-inspired algorithms with different numbers of search agents.

No. of Search Agents	Method	Test Accuracy (%)		
		CIFAR-10	CIFAR-100	ImageNet16-120
10	GA	93.95±0.30	71.91±0.74	46.06±0.75
	PSO	93.88±0.30	71.52±0.96	45.61±0.98
	BA	93.73±0.36	71.27±0.94	45.16±1.00
	DE	93.76±0.28	71.51±0.75	45.85±0.67
	WOA	93.77±0.35	71.17±0.91	45.40±1.13
	IGWO-SS	**94.16±0.20**	**72.94±0.81**	**46.53±0.45**
20	GA	94.00±0.24	72.09±0.79	46.18±0.60
	PSO	94.07±0.24	72.31±0.98	46.41±0.67
	BA	93.80±0.30	71.70±1.04	45.92±0.90
	DE	94.00±0.24	71.82±0.70	46.27±0.56
	WOA	93.95±0.25	72.19±0.99	45.98±0.77
	IGWO-SS	**94.27±0.17**	**72.92±0.59**	**46.89±0.41**
30	GA	94.11±0.21	72.48±0.69	46.26±0.52
	PSO	94.07±0.25	72.56±0.87	46.46±0.79
	BA	93.94±0.27	72.05±0.91	45.87±0.76
	DE	94.02±0.19	72.05±0.73	46.31±0.52
	WOA	93.98±0.30	72.47±1.01	46.26±0.57
	IGWO-SS	**94.30±0.18**	**72.95±0.70**	**46.92±0.46**

TABLE 9. Comparison with the state-of-the-art algorithms.

Method	Test Accuracy (%)		
	CIFAR-10	CIFAR-100	ImageNet16-120
RL [71]	93.85±0.37	71.71±1.09	45.24±1.18
RS [70]	93.70±0.36	71.04±1.07	44.57±1.25
BOHB [72]	93.61±0.52	70.85±1.28	44.42±1.49
GDAS [74]	93.61±0.09	70.70±0.30	41.71±0.98
DARTSV1 [73]	54.30±0.00	15.61±0.00	16.32±0.00
DARTSV2 [73]	54.30±0.00	15.61±0.00	16.32±0.00
SETN [75]	87.64±0.00	59.05±0.24	32.52±0.21
REA [45]	93.92±0.30	71.84±0.99	45.54±1.03
ENAS [43]	53.89±0.58	13.96±2.33	14.84±2.10
IGWO-SS	**94.16±0.20**	**72.94±0.81**	**46.53±0.45**

The mean and standard deviation of test accuracy of RL, RS, BOHB, GDAS, DARTSV1, DARTSV2, SETN, REA, ENAS are obtained from [53].

GDAS [74], DARTSV1 [73], DARTSV2 [73], SETN [75], REA [45], ENAS [43]. These algorithms are used as benchmark on NAS-Bench-201 [53]. The mean and standard deviation of test accuracy of these algorithms are obtained from [53]. The comparison of presented in Table 9. The algorithms were run multiple times, and the mean and standard deviation of test accuracy were calculated after completing the search. It is observed from the table that the IGWO-SS algorithm achieves higher predictive performance compared to these algorithms.

V. DISCUSSION

In this paper, we presented our proposed IGWO-SS algorithm for NAS. The IGWO-SS algorithm leverages the concept of synaptic saliency to explore the more promising parts of the search space instead of focusing on the less promising part, making the algorithm more efficient and fast.

We performed several experiments to determine the efficacy of synaptic saliency in improving NAS. The experimental results indicate that synaptic saliency positively correlates with accuracy, model size, and FLOPS. The positive correlation with neural network or model size is a drawback of synaptic saliency. However, it can still be effective in identifying promising neural network architectures from a set of untrained neural network architectures, since it positively correlates with predictive performance or accuracy. The SYNFLOW synaptic saliency has the highest correlation with accuracy, suggesting it might be better than the other two in identifying more promising architectures.

Our experimental results indicate that the IGWO-SS with SYNFLOW synaptic saliency works better than the IGWO-SS algorithm with SNIP or GRASP synaptic saliency. The IGWO-SS with SYNFLOW finds better neural network architectures by training fewer neural networks than the IGWO-SS with SNIP or GRASP. The IGWO-SS algorithm also works better and faster compared to the Standard GWO algorithm. The reason is that, in the Standard GWO algorithm, many less promising neural networks are needed to be trained. However, in the IGWO-SS algorithm, synaptic saliency is used to identify the more promising neural networks, and only those neural networks are trained, and the rest are discarded. As a result, the search process becomes faster, and better neural network architectures are found more quickly. The IGWO-SS algorithm was also compared with five bioinspired algorithms - PSO [65], BA [66], WOA [67], DE [68], and GA [77]. The experimental results suggest that the IGWO-SS algorithm requires training $10x$ fewer models and provides better final performance compared to these algorithms. The IGWO-SS algorithm was also compared with several state-of-the-art NAS algorithms that include - RL [71], RS [70], BOHB [72], GDAS [74], DARTSV1 [73], DARTSV2 [73], SETN [75], REA [45], ENAS [43]. The results indicate that the IGWO-SS algorithm achieves higher predictive performance compared to these algorithms.

VI. CONCLUSION

Although NAS can automatically find suitable neural network architectures for a specific task, the existing NAS algorithms are incredibly time-consuming. This study proposed an IGWO-SS algorithm for NAS, much faster than the existing NAS algorithms. The IGWO-SS algorithm leverages the idea of synaptic saliency to rank neural network architectures based on how promising they are. The algorithm only trains the more promising architectures and discards the rest based on this rank. It saves a significant amount of time since the training of many less promising architectures is skipped, and the search is more focused on the more promising part of the search space. The IGWO-SS algorithm requires training almost $10x$ fewer models than other bio-inspired algorithms, including Standard GWO, PSO, GA, DE, WOA, and DE, to achieve similar or even better results. The final performance of the IGWO-SS algorithm is also much better compared to these algorithms. The IGWO-SS algorithm also achieves better performance than the state-of-the-art NAS algorithms, including RL, RS, BOHB, GDAS, DARTSV1, DARTSV2, SETN, REA ENAS. The IGWO-SS

algorithm can perform NAS on a single GPU to get a very good architecture within a short period, making it suitable for anyone with time or resource constraints. One limitation of synaptic saliency used in the IGWO-SS algorithm is its positive correlation with model size. The future work should include developing a multi-objective framework for NAS leveraging synaptic saliency that addresses the limitation of synaptic saliency. Future work should also include identifying the efficacy of the IGWO-SS algorithm in other tasks, e.g., Natural Language Processing (NLP), Automatic Speech Recognition (ASR), etc. In this work, we mainly focused on using synaptic saliency from the pruning literature to improve the GWO algorithm. Future work may include other performance estimation techniques outside of pruning literature to improve different NAS algorithms.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the University of Dhaka for funding the project.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[2] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1–7.
[3] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[5] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.
[6] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1492–1500.
[7] X. Zhang, Z. Li, C. C. Loy, and D. Lin, "PolyNet: A pursuit of structural diversity in very deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 718–726.
[8] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," 2016, *arXiv:1605.07648*.
[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.
[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.
[12] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn treebank," in *Special Issue on Using Large Corpora: II* (Computational Linguistics), vol. 19, no. 2. Cambridge, MA, USA: MIT Press, Jun. 1993.
[13] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[14] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw Bayesian optimization," 2014, *arXiv:1406.3896*.
[15] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. joint Conf. Artif. Intell.*, 2015, pp. 1–9.
[16] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with Bayesian neural networks," in *Proc. ICLR*, Toulon, France, 2017.
[17] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017, *arXiv:1705.10823*.
[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*.
[19] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 1–8.
[20] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.
[21] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 517–531.
[22] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," 2018, *arXiv:1812.09926*.
[23] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 544–560.
[24] D. Wang, C. Gong, M. Li, Q. Liu, and V. Chandra, "AlphaNet: Improved training of supernets with alpha-divergence," 2021, *arXiv:2102.07954*.
[25] N. Richards, D. E. Moriarty, and R. Miikkulainen, "Evolving neural networks to play go," *Applied Intelligence*, vol. 8, no. 1, pp. 85–96, 1998.
[26] D. E. Moriarty and R. Miikkulainen, "Forming neural networks through efficient and adaptive coevolution," *Evol. Comput.*, vol. 5, no. 4, pp. 373–399, 1997.
[27] O. Rittho, R. Klinkenberg, S. Fischer, I. Mierswa, and S. Felske, "Yale: Yet another learning environment," in *Proc. LLWA Tagungsband der GI-Workshop-Woche Lernen-Lehren-Wissen-Adaptivität*, no. 76, 2001, pp. 84–92.
[28] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," *J. Mach. Learn. Res.*, vol. 10, no. 2, pp. 1–36, 2009.
[29] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. Learn. Intell. Optim.* Berlin, Germany: Springer, 2011, pp. 507–523.
[30] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. 25th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 24, 2011, pp. 1–9.
[31] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. Cham, Switzerland: Springer, Aug. 2013, pp. 847–855.
[32] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: Efficient and robust automated machine learning, part of the springer series on challenges in machine learning book series (SSCML)," Tech. Rep., 2019.
[33] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: Efficient and robust automated machine learning," in *Automated Machine Learning*, vol. 28. Cham, Switzerland: Springer, May 2019, pp. 113–134.
[34] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: Hands-free AutoML via meta-learning," 2020, *arXiv:2007.04074*.
[35] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn," in *Proc. ICML workshop AutoML*, vol. 9, 2014, p. 50.
[36] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A Python library for model selection and hyperparameter optimization," *Comput. Sci. Discovery*, vol. 8, no. 1, p. 014008, 2015.
[37] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 115–123.
[38] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Proc. Workshop Faces Real-Life Images, Detection, Alignment, Recognit.*, 2008, pp. 1–15.
[39] D. Cox and N. Pinto, "Beyond simple features: A large-scale feature search approach to unconstrained face recognition," in *Proc. Face Gesture*, Mar. 2011, pp. 8–15.
[40] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, "Automating biomedical data science through tree-based pipeline optimization," in *Proc. Eur. Conf. Appl. Evol. Comput.* Cham, Switzerland: Springer, 2016, pp. 123–137.
[41] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in *Proc. Workshop Autom. Mach. Learn.*, 2016, pp. 66–74.
[42] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Proc. Workshop Autom. Mach. Learn.*, 2016, pp. 58–65.

[43] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[44] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "EcoNAS: Finding proxies for economical neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11396–11404.
[45] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.
[46] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," 2017, *arXiv:1712.03351*.
[47] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C. I. Malossi, "Tapas: Train-less accuracy predictor for architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, 2019, pp. 3927–3934.
[48] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," 2018, *arXiv:1810.02340*.
[49] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," 2020, *arXiv:2002.07376*.
[50] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020, *arXiv:2006.05467*.
[51] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," 2021, *arXiv:2101.08134*.
[52] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," 2020.
[53] X. Dong and Y. Yang, "NAS-bench-201: Extending the scope of reproducible neural architecture search," 2020, *arXiv:2001.00326*.
[54] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[55] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and A. C. Berg, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[56] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*.
[57] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," 2016, *arXiv:1608.03983*.
[58] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian optimization with neural architectures for neural architecture search," 2019, *arXiv:1910.11858*.
[59] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Berlin, Germany: Springer, 2003, pp. 63–71.
[60] M. Bauer, M. van der Wilk, and C. E. Rasmussen, "Understanding probabilistic sparse Gaussian process approximations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1533–1541.
[61] A. Zela, J. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter, "Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks," 2020, *arXiv:2008.09777*.
[62] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*, vol. 4, no. 4. New York, NY, USA: Springer, p. 738. [Online]. Available: https://link.springer.com/book/9780387310732
[63] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with BONAS," 2019, *arXiv:1911.09336*.
[64] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, "Bayesian optimization with robust Bayesian neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4141–4149.
[65] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE ICNN*, vol. 4. Nov./Dec. 1995, pp. 1942–1948.
[66] X. S. Yang, "Bat algorithm for multi-objective optimisation," *Int. J. Bio-Inspired Comput.*, vol. 3, no. 5, pp. 267–274, 2011.
[67] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, Feb. 2016.
[68] J. Lampinen and R. Storn, "Differential evolution," in *New Optimization Techniques in Engineering*. Berlin, Germany: Springer, 2004, pp. 123–166.
[69] S. Mirjalili, "Genetic algorithm," in *Evolutionary Algorithms and Neural Networks*. Cham, Switzerland: Springer, 2019, pp. 43–55.
[70] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 1–25, 2012.
[71] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.
[72] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1437–1446.
[73] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.
[74] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1761–1770.
[75] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3681–3690.
[76] J. Turner, E. J. Crowley, M. O'Boyle, A. Storkey, and G. Gray, "BlockSwap: Fisher-guided block substitution for network compression on a budget," 2019, *arXiv:1906.04113*.
[77] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statist. Comput.*, vol. 4, no. 2, pp. 87–112, Jun. 1994.

SHIFAT E. ARMAN received the B.Sc. and M.Sc. degrees from the Department of Robotics and Mechatronics Engineering, University of Dhaka. He joined BRAC University, as a Faculty Member, in 2022. Before that, he worked as a Research Assistant at the University of Dhaka and a Visiting Research Assistant at Bangabandhu Sheikh Mujibur Rahman Agricultural University (BSMRAU). His research interests include automated machine learning (AutoML), neural architecture search (NAS), computer vision (CV), Bayesian optimization (BO), model-based, model-free, meta-heuristic optimization, medical image analysis, and artificial intelligence in finance and agriculture.

SHAMIM AHMED DEOWAN received the bachelor's degree in mechanical engineering from the Islamic University of Technology (IUT), the master's degree in sensor systems technology (mechatronics) from the Karlsruhe University of Applied Sciences, Germany, and the Ph.D. degree in chemical and materials engineering from the University of Calabria, Italy. He is currently an Assistant Professor and the Chairperson of the Department of Robotics and Mechatronics Engineering, University of Dhaka. Furthermore, he has an international patent in nanotechnology. He has 20 international journal publications, seven books/book chapters, and 30 international conference publications. He has done research on different topics, such as medical robotics, process engineering, and control and automation. He is a member of the Alumni Association of German Universities Bangladesh (AAGUB), European Membrane Society (EMS), American Society of Mechanical Engineers (ASME), and Institution of Engineers, Bangladesh (IEB).