

RESEARCH ARTICLE

Particle Swarm Optimization and Random Search for Convolutional Neural Architecture Search

KOSMAS DELIGKARIS 

Okinawa Institute of Science and Technology Graduate University, Okinawa 904-0495, Japan

e-mail: kosmas.deligkaris@oist.jp


ABSTRACT Evolutionary and swarm intelligence-based algorithms are commonly used as the search strategy component in Neural Architecture Search (NAS). However, little work has been done to quantify the performance improvements that these nature-inspired metaheuristics offer compared to simpler baseline methods such as random search. This work evaluates the efficacy of Particle Swarm Optimization (PSO) as a NAS strategy for chain-structured Convolutional Neural Networks (CNNs) by conducting thorough and fair comparisons of a PSO-based algorithm (termed evobpso) to 12 alternative methods from the literature as well as random search. A total of 10 benchmark datasets are used for model evaluation, including eight MNIST variations, MNIST-Fashion, and CIFAR-10. The results of this study suggest firstly that evobpso is a competitive NAS algorithm when compared to the literature methods, producing models with the lowest test error rate in three datasets (10.84% in MNIST-RD+BI, 1.62% in MNIST-RB, 5.44% in MNIST-Fashion). Secondly, a statistical comparison of 30 independent executions of evobpso and random search showed that the differences between the mean error rates of the models produced by the two algorithms were rather limited, ranging between 0.02% and 1.9%, but always in favor of evobpso. Therefore, it is concluded that evobpso is a viable NAS strategy able to find top-performing architectures, while random search also merits significant consideration due to its lower complexity and good average performance.

INDEX TERMS Neural architecture search, particle swarm optimization, neuroevolution, convolutional neural networks.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) have been successfully utilized in various computer vision applications. One of the early studies on CNNs was the work of LeCun et al. on hand-written document recognition [1]. Since then, a large number of improved network architectures has been presented, illustrating the interest of the research community and the potential of the methodology [2], [3], [4], [5], [6]. Some of the more successful hand-crafted architectures include LeNet [1], AlexNet [2], ResNet [3], DenseNet [4], Inception [5], and U-Net [6]. All of the above networks were crafted by experts in their field, after substantial trial and error.

The overall design of a neural network can be separated into the macro-architecture (number of layers, connectivity)

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

and micro-architecture (kernel size, pooling layer type). Finding optimal values of the macro- and micro-architecture is time-consuming and necessitates domain-specific knowledge. As a result, there is currently substantial interest in the research community in automating the design of CNNs, a field termed Neural Architecture Search (NAS). NAS is a subfield of automated machine learning and overlaps with hyper-parameter optimization [7], [8]. NAS algorithms are composed of three main components: search space, search strategy, and performance evaluation. Below, a summary of these three components is provided; for more comprehensive reviews the reader is referred to [7] and [9].

The search space defines the set of architectures that the NAS algorithm is able to find [7]. One possible search space is that of sequential chain-structured networks, where each layer receives input from the previous layer and sends its output to the next. More sophisticated search spaces consist of layers whose connectivity can be described as a directed

acyclic graph, with potentially multiple inputs or outputs per layer. Besides the layer connectivity, the hyper-parameters of each layer (e.g., the number of filters in convolutional layers) are also part of the search space.

The search strategy defines the method employed to create the solutions that will be evaluated. Bayesian optimization [10], evolutionary algorithms [11], reinforcement learning [12], and random search [13] have all been successfully utilized. For example, Real et al. showed that image classifiers evolved through reinforcement learning or evolution strategies have higher classification accuracy than random search [14]. Another study used the Adolescent Identity Search Algorithm (AISA) to optimize popular pre-trained convolutional architectures for diabetic retinopathy classification [15]. The results were promising, since AISA performed better than both random hyper-parameters and Bayesian optimization.

Performance evaluation refers to the methodology of assigning a fitness value to each candidate solution (neural network). Several alternatives have been proposed, such as full training, partial training, surrogate models, and progressive training. Full training is the most accurate but also computationally expensive, as each solution needs to be fully trained before evaluation. Due to prohibitive computational costs, several studies have opted for partial training during optimization (down to one epoch), after which the optimum solution is fully trained from scratch for a longer period (e.g., 100+ epochs) [16], [17], [18].

Surrogate models offer an alternative, computationally efficient approach [19]. However, their performance depends on the ability of the surrogate model to make accurate predictions for a large number of architectures based on a small amount of training data, which poses its own challenges. Lastly, progressive training refers to the case where the amount of training data used is initially limited but grows as the search progresses, improving the accuracy of network evaluation [20].

Due to the high computational costs of NAS, evolutionary and swarm intelligence-based algorithms are commonly used as NAS search strategies, as they promise to quickly find the global or local optima of a given problem. More specifically, a variety of algorithms can be found in the NAS literature, for example Particle Swarm Optimization (PSO) [16], [21], genetic algorithms [22], as well as hybrid approaches [23]. Despite their prevalence, however, whether or not evolutionary metaheuristics outperform simpler baseline hyper-parameter optimization methods, such as random search, has not been clarified yet.

The contribution of this work is two-fold. Firstly, a PSO-based NAS algorithm is devised (termed *evobpso*) and its performance evaluated against state-of-the-art methods. Secondly, an objective and statistically thorough comparison to random search is conducted by replacing the search strategy (PSO) with random initialization, while keeping the rest of the algorithm the same. As a result, the

comparison isolates the contribution of PSO (the search strategy component) from the rest of the implementation details (e.g., search space). It is found that *evobpso* is a competitive algorithm with regards to both state-of-the-art approaches as well as random search. However, the performance differences compared to random search were rather limited and dependent on the dataset. The results of this work support the hypothesis that swarm intelligence-based NAS approaches offer tangible but small improvements when compared to random search.

II. RELATED WORK

A. PARTICLE SWARM OPTIMIZATION

PSO is a population-based, iterative, stochastic optimization algorithm originally proposed by Kennedy and Eberhart in 1995 [24]. PSO has attracted a lot of attention in science and engineering due to its good performance, fast convergence, and low number of hyper-parameters [25]. These properties also make it suitable as a search strategy in computationally demanding NAS algorithms [16], [26]. PSO originates from the collective behavior of flocks of birds or schools of fish. PSO simulates animals' social behavior, where all group members work together to accomplish a common goal. Each individual, termed particle, is affected by their personal as well as the group's history.

Algorithmically, each particle keeps track of its own personal best solution as well as the swarm's best solution (termed the global best solution). In each iteration, a particle's new position is updated based on both the personal and global best solutions. The position update formula for PSO is defined as:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (1)$$

where x_i and v_i are the position and velocity of the i -th particle respectively. The velocity is calculated according to the following formula:

$$v_i(t+1) = \omega \times v_i(t) + c_1 \times r_1 \times (p_i(t) - x_i(t)) + c_2 \times r_2 \times (g_i(t) - x_i(t)) \quad (2)$$

where ω , c_1 , and c_2 are hyper-parameters controlling the behavior of the algorithm. The first hyper-parameter ω is termed momentum and affects how much the velocity of the previous iteration will influence the velocity at the current iteration. The second and third parameters, c_1 and c_2 , are termed acceleration coefficients and control the influence of the personal and global best solution respectively. The personal and global best positions are denoted as p_i and g_i respectively. Lastly, r_1 and r_2 are random numbers in $[0, 1]$ aiding in search space exploration.

The Boolean PSO (BPSO) is a variation of the real-valued PSO algorithm. BPSO is based on Boolean algebra and although its parameters closely mimic those of the real-valued PSO, BPSO works with binary variables [27], [28]. In BPSO, the position and velocity update formulas are defined for each

bit d in a particle as follows:

$$x_d(t+1) = x_d(t) \oplus v_d(t+1) \quad (3)$$

where the velocity is calculated according to the following formula:

$$v_d(t+1) = \omega \cdot v_d(t) + c_p \cdot (p_d(t) \oplus x_d(t)) + c_g \cdot (g_d(t) \oplus x_d(t)) \quad (4)$$

In the above formulas, the Boolean *and* (\cdot), *or* ($+$), and *xor* (\oplus) operations are used. The momentum (ω), c_p , and c_g acceleration coefficients are bits assigned stochastically from predefined parameters (Ω , C_p , C_g) in $[0, 1]$ respectively. More specifically, Ω , C_p and C_g define the probability of ω , c_p , and c_g being equal to 1. These three BPSO coefficients define the influence of the previous iteration's velocity, the personal best solution, and the global best solution, similar to the parameters of the real-valued PSO. For more information on the selection of hyper-parameters the reader is referred to the existing studies by Bratton and Kennedy [29], Trelea [30], and Shi and Eberhart [31].

B. PSO FOR NEURAL ARCHITECTURE SEARCH

One of the first attempts to use PSO for NAS was IPPSO, which used an encoding inspired by network IP addresses [32]. In IPPSO, a specific layer is encoded by a binary string of two bytes, which is then converted to two separate values of one byte (termed an IP address). To accommodate variable-depth networks, disabled layers are utilized. Therefore, the number of dimensions in each particle is constant, while the depth of the encoded network can vary based on the number of disabled layers. IPPSO was evaluated on three common benchmark datasets, the MNIST Basic, the MNIST with Rotated Digits and Background Images, and the Convex sets [33].

Junior and Yen proposed a PSO variant for NAS, up to 20 layers deep, termed psoCNN [16]. It was shown that the psoCNN algorithm could outperform several other algorithms in nine image classification problems, while having competitive run-time performance and quick convergence. A drawback of the psoCNN approach is that for each iteration, the parameter values are being copied directly from the global or personal best. In turn, this limits search space exploration and the quality of the optimum solution found.

Subsequently, Lawrence et al. [17] proposed a method with a new velocity update mechanism that takes into account the value of layer parameters when both layers are convolutional, an improvement upon the earlier psoCNN algorithm. In addition, a group-based encoding was used to ensure the number of pooling layers is valid without additional rules. Eight well-known datasets were used for evaluation, with the results showing state-of-the-art performance.

Singh et al. [34] proposed a PSO-based algorithm with two different optimization levels (MPSO-CNN). The population at the first level optimizes the architecture in terms of the

number of convolutional layers, the number of pooling layers, and the number of fully connected layers. At the same time, the population at the second level optimizes the hyper-parameters (e.g., number of filters) of each architecture. MPSO-CNN utilized a sigmoid-like function to control the inertia weight and maintain a balance between exploration and exploitation. MPSO-CNN was tested on five benchmark datasets including MNIST, CIFAR-10, and CIFAR-100, having improved performance compared to other competitive algorithms and models.

Nistor and Czubala [35] introduced IntelliSwAS, a framework for discovering neural architecture cells. Cells are small modules that can be aggregated in order to create complete network architectures. IntelliSwAS utilizes a machine learning model for predicting the relative quality of a pair of network cells, rather than full training and testing. IntelliSwAS showed high performance on a set of nine standard sets, including CIFAR-10 and MNIST variations.

Elhani et al. [18] proposed a variant of the PSO algorithm called Particle Swarm Optimization without Velocity (pswvCNN), aiming to minimize the computational cost and the convergence time. However, this approach copies layers directly from either the personal or global best solutions, without creating novel layer configurations. Therefore, similar to the algorithm proposed by Junior et al. [16], pswvCNN may suffer from reduced space exploration.

Yuan et al. [20] proposed an autoencoder-based PSO algorithm for efficient neural architecture search (EAEPSO). In EAEPSON, variable-length network architectures are encoded as latent vectors of fixed length, resulting in dimensionality reduction while maintaining discriminative features. Further, a progressive fitness evaluation scheme was employed in order to conserve computational resources. Briefly, particles are trained initially on partial datasets until convergence, followed by additional rounds of training with progressively more training data. EAEPSON achieved an error rate of 2.74% on the CIFAR-10 and 16.17% on the CIFAR-100 dataset respectively.

III. RESEARCH PROBLEMS AND CONTRIBUTIONS

As NAS is computationally expensive, evolutionary and swarm intelligence algorithms, such as PSO and genetic algorithms, can be utilized to efficiently guide the search. On the other hand, the utilization of such population-based metaheuristics increases the complexity of the algorithm as well as the required computational resources.

For example, the presence of layer types with different properties (pooling layer, convolutional layer) prevents the straightforward hybridization of the personal and global best terms (as in (4)). The effects of this discrepancy can be seen in the literature, with some works bypassing hybridization by copying the layers directly [16], [18], and others creating sophisticated rules for hybridization [17]. A disadvantage of the copying strategy is that it severely limits the range of created solutions, while custom hybridization solutions may

need additional modifications when the search space expands to include other, incompatible, layer types.

On the contrary, random search requires no such modifications, and because of its nature it does not suffer from reduced space exploration. Therefore, given the above obstacles in the implementation of population-based metaheuristics, it is important to document and quantify the performance improvements evolutionary strategies offer in comparison to random search. In turn, this will allow practitioners to make well-informed decisions on whether the benefits of implementing an evolutionary NAS strategy outweigh the added complexity costs.

Further, earlier studies showed that state-of-the-art NAS methods failed to substantially beat random search [8], [36] in several popular NAS benchmarks, such as ENAS [37] and DARTS [38]. In light of this result, consideration of the performance of random search is necessary for the proper evaluation of evolutionary NAS algorithms. However, to the best of the author's knowledge, there are no thorough and fair comparisons of PSO to random search. As a result, a literature gap in relation to the performance of PSO and random search has been identified.

The main goal of this work is to perform a critical evaluation of the efficacy of PSO as a search strategy in NAS, by quantifying its performance gains in classification accuracy rates against those of random search. More specifically, the contributions of this work are as follows:

- A novel PSO-based algorithm is described, composed of a binary search space encoding and a modified version of the BPSO algorithm. It is shown that the devised algorithm exhibits competitive performance against alternative methods and models from the literature. An analysis of the algorithm's behavior illustrates the good convergence properties and robustness of the proposed method.
- The proposed method is then compared to random search in nine benchmark datasets. A thorough and fair statistical comparison of the two approaches is ensured by running each algorithm 30 times, using the same experimental parameters otherwise. To the best of the author's knowledge, this work is the first study that objectively and thoroughly quantifies potential performance improvements of PSO compared to baseline methods.

IV. METHODS

A. SEARCH SPACE AND NETWORK ARCHITECTURE ENCODING

The basic architectural scheme used in this work is a chain-structured CNN (Fig. 1), composed of Convolutional, Max Pooling, and Average Pooling layers. At the end of the network, a fully connected layer with the number of neurons equal to the number of classes in the corresponding dataset is always added. In addition, a maximum of two pooling layers can be present in any architecture.

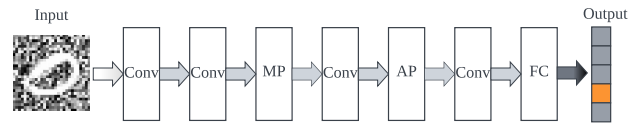


FIGURE 1. Chain-structured CNN architecture used in the current study. **Conv:** Convolutional layer; **MP:** Max Pooling layer; **AP:** Average Pooling layer; **FC:** Fully Connected layer.

TABLE 1. Encoding of a single layer and search ranges.

Bits	Representation	Range of values
8	Convolutional filters size	[1, 256]
3	Convolutional kernel size	[2, 9]
1	Existence of pooling layer 0: True 1: False	[0, 1]
1	Type of pooling layer 0: Max pooling 1: Average pooling	[0, 1]
1	Kernel size of pooling layer 0: 2x2 1: 3x3	[0, 1]

A challenge in optimizing chain-structured architectures with population-based metaheuristics is the incompatibility of the convolutional and pooling layers. Convolutional layers have a corresponding number of filters and kernel size while pooling layers are not associated with a number of filters, but only a pooling kernel size. This complicates layer by layer hybridization of solutions since the semantics of the encoding are different for each layer type.

To deal with the layer incompatibility issue, this work uses an integrated binary encoding scheme, where only convolutional layers are represented explicitly; the existence and properties of a pooling layer are included in the encoding of the previous convolutional layer. More specifically, each convolutional layer is represented by a binary string of 14 bits. Eight bits are used to encode the number of convolutional filters, three bits are used to encode the convolutional kernel size, one bit to encode the existence of a pooling layer after the convolutional layer, one bit is used to control the type of pooling layer (max or average pooling), and one bit to control the kernel size of the pooling layer (2×2 , or 3×3). The definitions and ranges of these parameters are summarized in Table 1.

B. VELOCITY AND POSITION UPDATE

Each particle's position is a variable-length array of binary strings. As such, the formulas for the BPSO algorithm can be applied to each bit of the binary strings. However, issues arise in the hybridization of particles due to the variable length of the encoded networks. As such, a "Remove" indicator is defined and used whenever the network lengths of the hybridized particles are not equal.

Fig. 2 shows how two positions are hybridized during the calculation of velocity. Briefly, if both positions have an equal number of layers, then the result is calculated based on the BPSO formula. If the current position is longer than the best position, a "Remove" indicator is added. If the best position

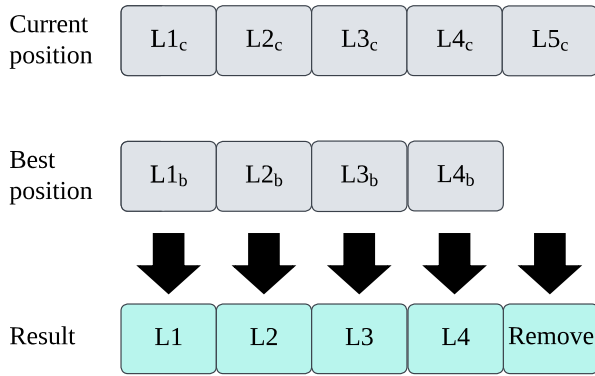


FIGURE 2. Hybridization of positions. Hybridization occurs when the current position is combined with the (personal or global) best position, resulting in the creation of the (personal or global) influence terms.

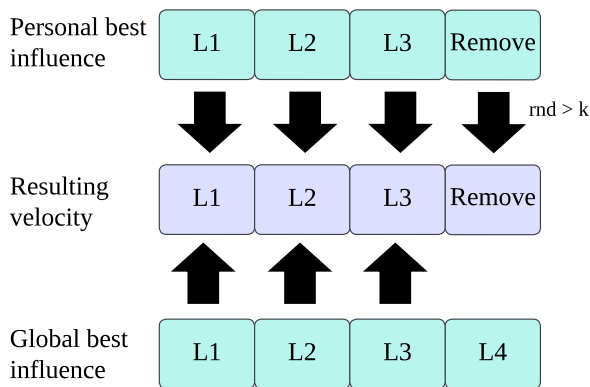


FIGURE 3. Calculation of velocity from the personal and global influence terms. For layers where both the personal and global terms are present the velocity is calculated based on the standard BPSO formula. If one of the terms is “Remove”, a random number is used to select one of the two layers.

is longer than the current position, zero is assumed for the value of the current position and calculations proceed as normal. The hybridization of the particle’s current position with the personal and global best positions results in the personal and global influence terms, which are then used to calculate the velocity as follows (Fig. 3).

Firstly, “Remove” indicators are added to ensure that the sizes of the two terms are equal. Subsequently, the BPSO formula is applied to the layers for which both terms have values. Alternatively, if one of the two terms has a “Remove” indicator, then a random number is used to select the layer from either the personal or global term, with equal probability for either (parameter defined as k).

Once the velocity has been calculated, the position of a particle is updated as follows (Fig. 4). For each layer of the position and velocity, a Boolean *xor* operation is used (as per the standard BPSO formula). However, if the velocity layer is a “Remove” operator no layer is added/calculated.

V. EXPERIMENTAL SETUP

A. ALGORITHM DESCRIPTION

The pseudo-code of the evobpso algorithm is shown in Algorithm 1, while a flowchart is illustrated in Fig 5. Firstly,

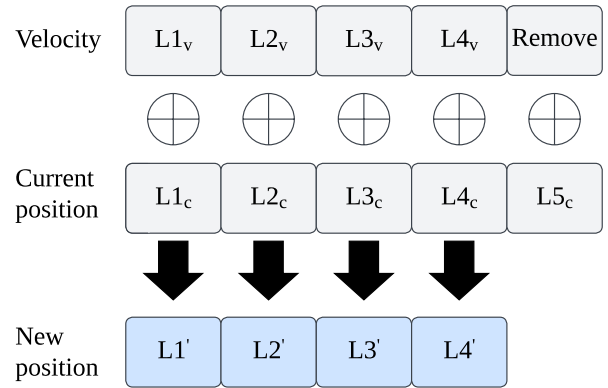


FIGURE 4. Calculation of a particle’s updated position based on the velocity and current position. For all layers, an *xor* operation is used, unless the velocity layer is a “Remove” indicator, in which case no layer is added.

the population is initialized randomly within the search space and the global best particle is identified. Before evaluation, the encoded position of each particle is converted to a deep learning model and the generated model is subsequently trained and evaluated. Ten percent of the training dataset was held out in order to be used as the validation dataset during model training. The fitness value of a particle was calculated as the validation cross-entropy loss. Subsequently, the algorithm evolves the generated architectures by updating the velocities and positions of the particles at each iteration according to the BPSO formula. Once the optimization has reached the maximum number of iterations, the best model found is trained from scratch for a larger number of epochs and evaluated in the unseen test dataset to get the final test accuracy rate. Following, the various parameters used in this work are described:

1) BPSO PARAMETERS

In this work, a population size of 25 was used. The number of iterations was set to 10. Both c_p and c_g were set to 0.5, which places equal importance on the personal and global best solutions. k was also set to 0.5. The momentum term was set to zero.

2) ARCHITECTURE PROPERTIES

A maximum number of 18 convolutional layers was set. The maximum number of pooling layers was restricted to two. For all networks, the convolution stride was set to one, and the activation function was set to ReLU. Padding “same”. Each layer besides the first one was preceded by a dropout layer, with a dropout rate of 0.5 [39]. Batch normalization layers were also utilized [40]. The l2 regularization rate was set to 0.01. For all pooling layers, the stride was set equal to the kernel size.

3) MODEL TRAINING

For all trained models, the batch size was set to 64 and Adam was used as the optimizer [41]. Categorical cross entropy was used as the loss function. The use of early stopping criteria in

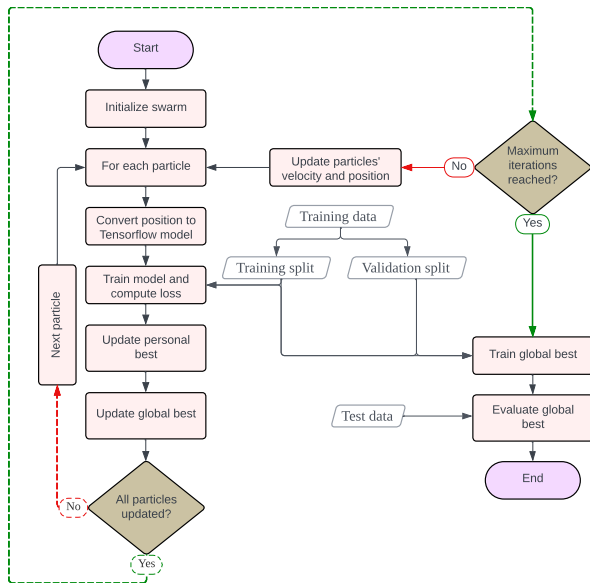


FIGURE 5. Flowchart of the evobpso algorithm. Notice the use of different splits of the datasets for training, validation, and testing.

the optimization and evaluation phases was different. During the optimization phase, each model was trained for up to a maximum of 10 epochs, with two early stopping criteria. Firstly, if the model did not improve for a certain number of consecutive epochs, termed patience, training would stop. Patience was set to two. Secondly, an early stopping criterion based on a population reference curve, originally proposed by Suganuma et al. in [42], was also implemented. In the evaluation phase, the models were trained for a maximum of 150 epochs, with patience set to 30.

4) EQUIPMENT AND SOFTWARE

Experiments were conducted in a cluster environment equipped with Nvidia A100 GPUs. Tensorflow 2 was used as the deep learning library [43].

B. DATASETS

A total of 10 common image classification datasets are used in this work. These are the MNIST [1], MNIST-RD, MNIST-BI, MNIST-RB, MNIST-RD+BI, Rectangles, Rectangles-I, Convex [33], as well as MNIST-Fashion [44] and CIFAR-10 [45] datasets. A summary of these datasets and their properties is given in Table 2. The selected datasets were used on several PSO-related NAS studies that are discussed here. As a result, it was considered appropriate to use these datasets in order to test the proposed algorithm. All datasets were retrieved online,¹ besides CIFAR-10 which was retrieved from Tensorflow.²

The MNIST dataset was originally created by LeCun et al. [1], containing images of handwritten digits, and is frequently used as a benchmark in image classification studies. The MNIST dataset contains a total of 70,000 images, with 60,000

used for training and 10,000 used for testing. The images are black and white with a size of 28×28 pixels.

Based on the original MNIST dataset, the additional variations (MNIST-RD, MNIST-BI, MNIST-RB, MNIST-RD+BI) pose significantly more difficult challenges to image classification algorithms. More specifically, MNIST-RD contains the MNIST images with digits randomly rotated. MNIST-BI contains the MNIST digits with various background images, while MNIST-RB contains the MNIST digits with random background. Lastly, MNIST-RD+BI contains both rotated digits and background images. These datasets are composed of 12,000 training and 50,000 test images.

The Rectangles dataset contains black and white images of rectangles, of variable sizes. Note, in these images, only the outline is white; the inside of the rectangle is the same color as the background. The Rectangles-I dataset is similar to the Rectangles dataset but with images replacing either the background or the inside of the rectangle. For both Rectangles datasets, the output of the classification model represents which side is longer, the width or the height. Both datasets contain 12,000 training images and 50,000 test images.

The Convex dataset contains images of random geometric shapes, classified as either convex or not. The Convex dataset contains 8,000 training samples and 50,000 test samples. The MNIST-Fashion dataset contains different types of apparel, including dresses, sandals, bags, and t-shirts. The images are 28×28 pixels, grayscale, with a total of 10 classes. This dataset is commonly used for benchmarking computer vision algorithms, as it is more challenging than MNIST. For example, a random forest classifier can achieve up to 97% test accuracy in the MNIST dataset, but only about 87% test accuracy in the MNIST-Fashion dataset [44].

Lastly, the CIFAR-10 dataset contains 50,000 training and 10,000 test color images. The resolution of the images in the CIFAR-10 dataset is 32×32 pixels. The CIFAR-10 dataset is a well-accepted benchmark for image classification.

C. PEER COMPETITORS' MODELS

The performance of the proposed algorithm was compared to a total of 12 peer competitors' models, including both hand-crafted and evolved neural networks. The evolved models include methods where the search strategy is based on PSO as well as methods based on genetic algorithms. Thus, both evolutionary and swarm intelligence-based approaches have been considered when comparing the performance of evobpso to state-of-the-art algorithms.

LeNet-5 is a hand-crafted architecture, and a variant of the LeNet model, proposed by LeCun [1]. LeNet-5 is composed of 22 convolutional layers, two average pooling layers, and three fully connected layers. SVMrbf and SVMpoly are based on Support Vector Machines with Gaussian and polynomial kernels respectively [33]. RandNet-2 and LDANet-2 are simple deep learning networks utilizing random cascaded filters or linear discriminant analysis [46].

¹<http://www-labs.iro.umontreal.ca/lisa/icml2007/data/>

²<https://www.tensorflow.org/datasets>

Algorithm 1 The Devised Neural Architecture Search Model

```

Input: Swarm size ( $N$ ), number of iterations ( $iter_{max}$ ), training data ( $X$ ), testing data ( $Z$ ), epochs for training during
optimization ( $e_{train}$ ), epochs for training during final evaluation ( $e_{test}$ ), hyper-parameters ( $hparams$ ).
Output: The best model found together with its test accuracy.
 $S = \{P_1, P_2, \dots, P_N\} \leftarrow InitializeSwarm(N, hparams)$ 
/* Find and initialize global best */
 $gbest \leftarrow InitializeGbest(S)$ 
for  $iter \leftarrow 1$  to  $iter_{max}$  do
  for  $i \leftarrow 1$  to  $N$  do
     $P_{i,velocity} \leftarrow UpdateVelocity(P_i, gbest, hparams)$ 
     $P_{i,position} \leftarrow UpdatePosition(P_i, hparams)$ 
     $P_{i,fitness} \leftarrow ComputeLoss(P_i, X, e_{train})$ 
    /* Update personal best */
    if  $P_{i,fitness} < P_{pbest,fitness}$  then
       $P_{pbest,fitness} \leftarrow P_{i,fitness}$ 
       $P_{pbest,position} \leftarrow P_{i,position}$ 
    end
    /* Update global best */
    if  $P_{i,fitness} < gbest_{fitness}$  then
       $gbest_{fitness} \leftarrow P_{i,fitness}$ 
       $gbest_{position} \leftarrow P_{i,position}$ 
    end
  end
end
/* Train and evaluate best model on test dataset */
 $gbest_{test\_accuracy} \leftarrow EvaluateGbest(gbest, X, Z, e_{test})$ 
return  $gbest, gbest_{test\_accuracy}$ 

```

TABLE 2. Datasets used in this study. Ten percent of the training data was held-out during training as validation split.

Dataset name	Image size	Number of classes	Training/Test size
MNIST	28x28x1	10	60,000/10,000
MNIST-RD	28x28x1	10	12,000/50,000
MNIST-BI	28x28x1	10	12,000/50,000
MNIST-RB	28x28x1	10	12,000/50,000
MNIST-RD+BI	28x28x1	10	12,000/50,000
Rectangles	28x28x1	2	12,000/50,000
Rectangles-I	28x28x1	2	12,000/50,000
Convex	28x28x1	2	8,000/50,000
MNIST-Fashion	28x28x1	10	60,000/10,000
CIFAR-10	32x32x3	10	50,000/10,000

IPPSO [32], psoCNN [16], PSO-based [17], pswvCNN [18], and IntelliSwAS [35] are all swarm intelligence algorithms based on PSO and are described in detail in section II-B. On the other hand, evoCNN [22] and SEECNN [47] are evolutionary NAS methods based on genetic algorithms.

For the MNIST-Fashion dataset, additional models are utilized, such as SqueezeNet-200 [48], VGG16 [49], and GoogleNet [5]. From the population-based algorithms, evoCNN [22], psoCNN [16], sosCNN [50], and pswvCNN [18] report results for the MNIST-Fashion dataset and as such they are included here as well.

VI. RESULTS**A. COMPARISON AGAINST EXISTING ALGORITHMS**

Firstly, the test error rates of the models produced by the PSO-based algorithm (termed evobpso) were compared to 12 competitor algorithms and models from the literature. The statistics shown in Table 3 are based on 10 independent runs, which is equal or lower to the number of runs used in the alternative approaches, ensuring a fair comparison [16], [17], [18], [32], [35]. In addition, the population size was set to 25, which is in the middle of the 20-30 range commonly used in the literature [16], [17], [18], [32].

In the MNIST dataset, the best model produced by the proposed algorithm exhibited a test error rate of 0.36%, which is very close to the best value among all competing approaches (0.30%, obtained by pswvCNN). The resulting model surpassed evoCNN (1.18%), IntelliSwAS (0.38%), as well as SEECNN (0.79%).

In the MNIST-BI dataset, the best model produced by evobpso had a test error rate of 2.33%. This places it fourth, behind pswvCNN (2.27%), PSO-based (2.20%), and psoCNN (1.90%). Although this is not the best result among the competitor approaches, the proposed algorithm produced a model with a lower test error rate than seven of the 12 alternative approaches,

TABLE 3. Test error rates of evobpsop compared to alternative approaches and models. Statistics are calculated based on 10 independent executions. The results of the evobpsop algorithm are shown in bold.

Model/Dataset	MNIST	MNIST-BI	MNIST-RD+BI	MNIST-RD	MNIST-RB	Convex	Rectangles-I	Rectangles
LeNet-5 [1]	0.95% (+)	-	-	-	-	-	-	-
SVMrbf [33]	3.03% (+)	22.61% (+)	32.62% (+)	10.38% (+)	14.58% (+)	19.13% (+)	2.15% (+)	24.04% (+)
SVMpoly [33]	3.69% (+)	24.01% (+)	37.59% (+)	13.61% (+)	16.62% (+)	19.82% (+)	2.15% (+)	24.05% (+)
RandNet-2 [46]	0.63% (+)	11.65% (+)	43.69% (+)	8.47% (+)	13.47% (+)	5.45% (+)	17.00% (+)	0.09% (+)
LDANet-2 [46]	0.62% (+)	12.42% (+)	38.54% (+)	7.52% (+)	6.81% (+)	7.22% (+)	16.20% (+)	0.14% (+)
IPPSO [32]	1.13% (+)	-	34.50% (+)	-	-	8.48% (+)	-	-
evoCNN [22]	1.18% (+)	4.53% (+)	35.03% (+)	5.22% (+)	2.80% (+)	4.82% (+)	5.03% (+)	0.01% (-)
psoCNN [16]	0.32% (-)	1.90% (-)	14.28% (+)	3.58% (+)	1.79% (+)	1.70% (+)	2.22% (+)	0.03% (-)
PSO-based [17]	0.35% (-)	2.20% (-)	11.61% (+)	3.23% (+)	1.80% (+)	1.36% (-)	1.01% (+)	0% (-)
IntelliSwAS [35]	0.38% (+)	3.53% (+)	15.74% (+)	4.46% (+)	2.55% (+)	1.20% (-)	2.22% (+)	0% (-)
SEECNN [47]	0.79% (+)	4.06% (+)	17.92% (+)	4.33% (+)	2.44% (+)	3.79% (+)	2.18% (+)	0% (-)
pswvCNN [18]	0.30% (-)	2.27% (-)	11.47% (+)	2.74% (-)	1.67% (+)	1.35% (-)	0.72% (-)	0.01% (-)
evobpsop (best)	0.36%	2.33%	10.84%	3.03%	1.62%	1.52%	0.89%	0.07%
evobpsop (mean)	0.46%	2.79%	13.72%	3.74%	2.08%	3.58%	1.55%	0.60%
evobpsop (std)	0.08%	0.26%	1.62%	0.54%	0.26%	2.73%	0.72%	0.65%

which ranged from 3.53% (IntelliSwAS) to 24.01% (SVMpoly).

Subsequently, evobpsop produced the best model for the MNIST-RD+BI dataset, with a test error rate of 10.84%. This result is by far the best result obtained in this dataset, with the next best model having a test error rate of 11.47% (pswvCNN). The PSO-based model had a test error rate of 11.61%, while the rest of the models had test error rates higher than 14%. The variability in the test errors and overall lower performance compared to the base MNIST dataset is not surprising, given that the MNIST-RD+BI dataset is the most challenging among the MNIST-based group.

In the MNIST-RD dataset the proposed algorithm came second with a test error rate of 3.03%. The best error rate among all models compared was produced by pswvCNN (2.74%). The rest of the models did not perform as well, with evoCNN producing a model with a test error rate of 5.22%, and IntelliSwAS a model with a test error rate of 4.46%.

In the MNIST-RB dataset, evobpsop produced the best model among all compared approaches, with a test error rate of 1.62%. On the contrary, pswvCNN produced the second-best model with a test error rate of 1.67%, psoCNN had a test error rate of 1.79%, while IntelliSwAS had a test error rate of 2.55%.

In the Convex dataset, the proposed algorithm came fourth, with a test error rate of 1.52%. The best test error was obtained by IntelliSwAS (1.20%), while pswvCNN came second, with a test error rate of 1.35%. In the Rectangles-I dataset, the proposed algorithm came second, with a test error rate of 0.89%, just behind pswvCNN (0.72%). Lastly, in the Rectangles dataset, the proposed algorithm exhibited an almost perfect test error rate of 0.07%. However, many alternative algorithms also performed very well in this dataset, with IntelliSwAS, PSO-based, and SEECNN achieving 0%.

Table 4 shows the performance of the proposed algorithm against 10 alternative algorithms and models in the MNIST-Fashion dataset. It can be seen that the proposed algorithm produced the model with the lowest error rate

TABLE 4. Test error rates of evobpsop for the MNIST-Fashion dataset. Statistics are calculated on a total of 10 independent executions. The results of the evobpsop algorithm are shown in bold. Data were collected from [18].

Model	Test error (%)	# Parameters
2C1P2F+Dropout [22]	8.4% (+)	3.27M
2C1P [22]	7.5% (+)	100K
SqueezeNet-200 [48]	10.0% (+)	500K
AlexNet [2]	10.01% (+)	62.3M
VGG16 [49]	6.5% (+)	26M
GoogleNet [5]	6.3% (+)	23M
evoCNN [22]	5.47% (+)	6.68M
psoCNN [16]	5.53% (+)	2.32M
sosCNN [50]	5.68% (+)	2.30M
pswvCNN [18]	5.44% (=)	5.15M
evobpsop (best)	5.44%	3.17M
evobpsop (mean)	6.23%	5.71M

among all competitive approaches, with only pswvCNN being able to match the final result (5.44%). However, the model produced by evobpsop has about 2 million parameters fewer than the model found by pswvCNN (3.17M vs 5.15M).

B. COMPARISON TO RANDOM SEARCH

The NAS algorithm presented in this work is composed of various components, including the search space, the search space encoding, and the PSO-based search strategy. Each of these factors has contributed, to a certain extent, to the performance of the algorithm in the benchmark tests. Therefore, to evaluate specifically the impact of the proposed search strategy, a fair comparison to random search was conducted.

The comparison to random search was conducted by replacing the position update formula with random initialization, while keeping all other parameters unchanged. Further, to evaluate the generalization capabilities of the algorithm in a wide variety of datasets while considering time constraints, the base MNIST dataset was replaced with the CIFAR-10 dataset. In addition, to allow for a proper statistical comparison, each algorithm was run for a total of 30 executions.

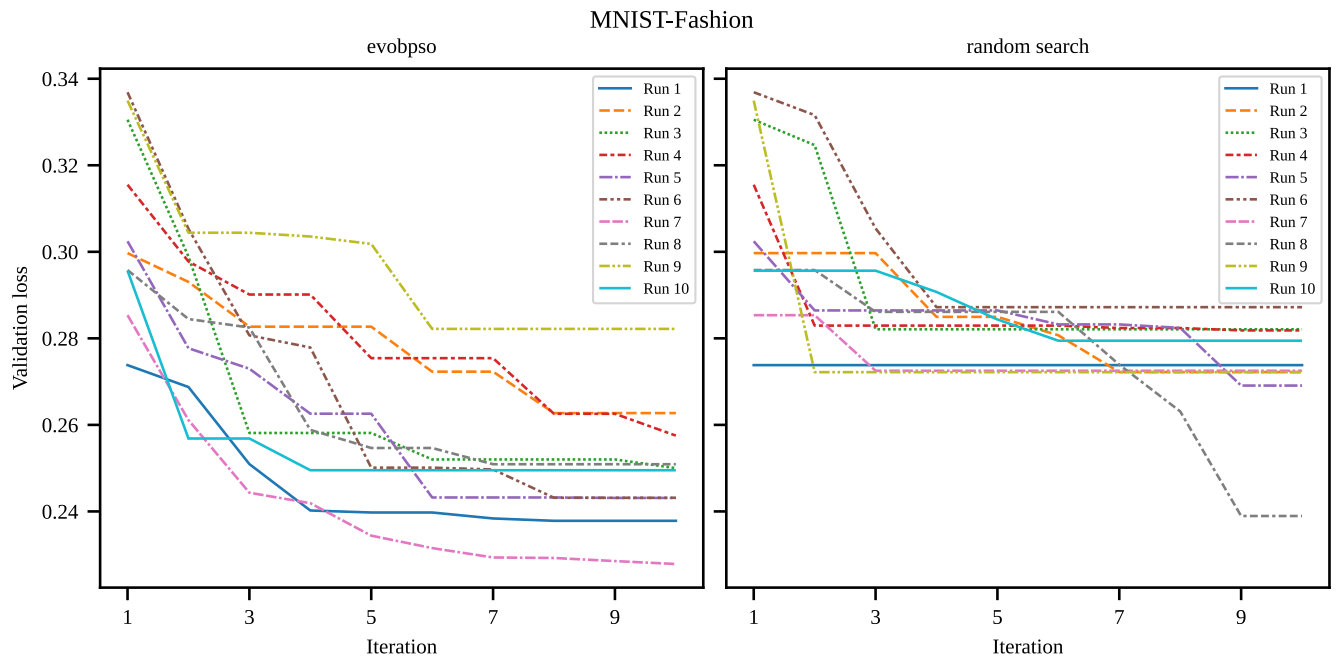


FIGURE 6. Convergence of the evobps algorithm and random search in the MNIST-Fashion dataset for the first 10 runs. Each line shows the fitness value of the global best particle for each iteration.

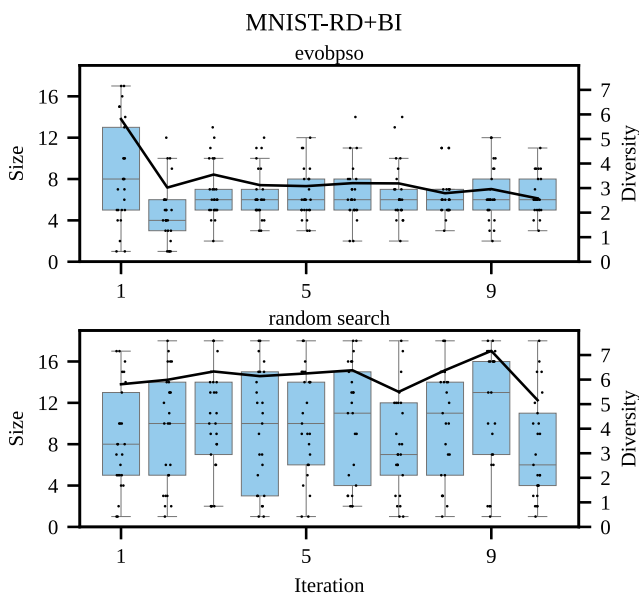


FIGURE 7. Evolution of particle size (network length, left axis) and population diversity (right axis) for evobps and random search, for the MNIST-RD+BI dataset. Particle size is shown as boxplots, while the diversity is shown as the thick black line.

Besides analyzing the accuracy rates, the behavior of evobps in relation to random search was also investigated. The motivation for this analysis was to evidence the potential of evobps as an evolutionary algorithm and identify any shortcomings, as lack of convergence or fitness improvement may indicate an inefficient algorithm. The subsequent sections present the algorithm behavior first followed by the performance analysis.

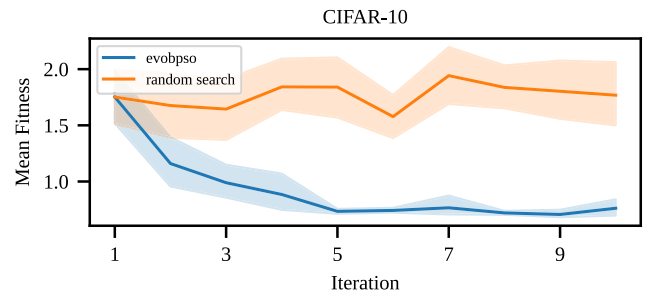


FIGURE 8. Evolution of mean population fitness for evobps and random search, for the CIFAR-10 dataset. Shaded areas represent the 95% confidence intervals.

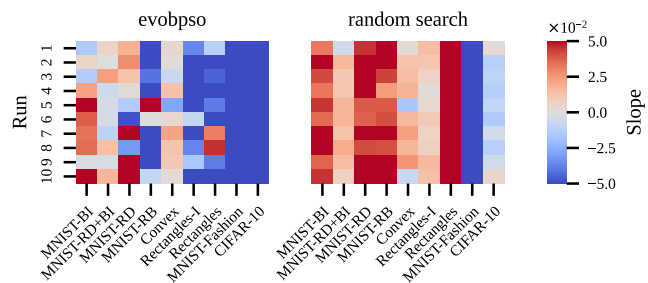


FIGURE 9. Slope of the mean population fitness evolution curves for all datasets. The slope was calculated based on a linear regression to the mean population fitness per iteration.

1) ALGORITHM BEHAVIOR

Fig. 6 shows the convergence of the global best solution for the first 10 runs in the MNIST-Fashion dataset for evobps and random search. It can be seen that evobps manages to reduce the validation loss, quickly in the first iterations and more slowly later on. On the contrary, random search exhibits

TABLE 5. Test error rates of the evobps algorithm compared to random search. Performance improvement is calculated as the difference of the evobps and random search means. Statistics are based on a total of 30 executions. Bold letters highlight the best model.

Algorithm/Dataset	MNIST-BI	MNIST-RD+BI	MNIST-RD	MNIST-RB	Convex	Rectangles-I	Rectangles	MNIST-Fashion	CIFAR-10
evobps (best)	2.16%	9.63%	3.03%	1.62%	1.45%	0.80%	0.07%	5.44%	12.79%
evobps (mean)	2.77%	13.91%	3.89%	2.01%	3.44%	1.70%	0.68%	6.37%	15.20%
evobps (std)	0.33%	2.39%	0.50%	0.22%	1.92%	0.70%	0.55%	0.56%	1.66%
random search (best)	2.33%	10.81%	3.44%	1.68%	1.81%	0.96%	0.14%	5.71%	12.85%
random search (mean)	3.06%	15.81%	4.06%	2.12%	3.46%	2.59%	1.55%	6.42%	15.40%
random search (std)	0.47%	3.21%	0.41%	0.22%	1.52%	0.98%	4.89%	0.43%	1.91%
performance improvement	0.29%	1.9%	0.17%	0.11%	0.02%	0.89%	0.87%	0.05%	0.20%
statistic value	107.50	88.50	160.00	140.00	231.50	80.50	217.00	194.00	224.00
p-value	0.009	0.002	0.140	0.094	1.0	0.001	0.761	0.611	0.871

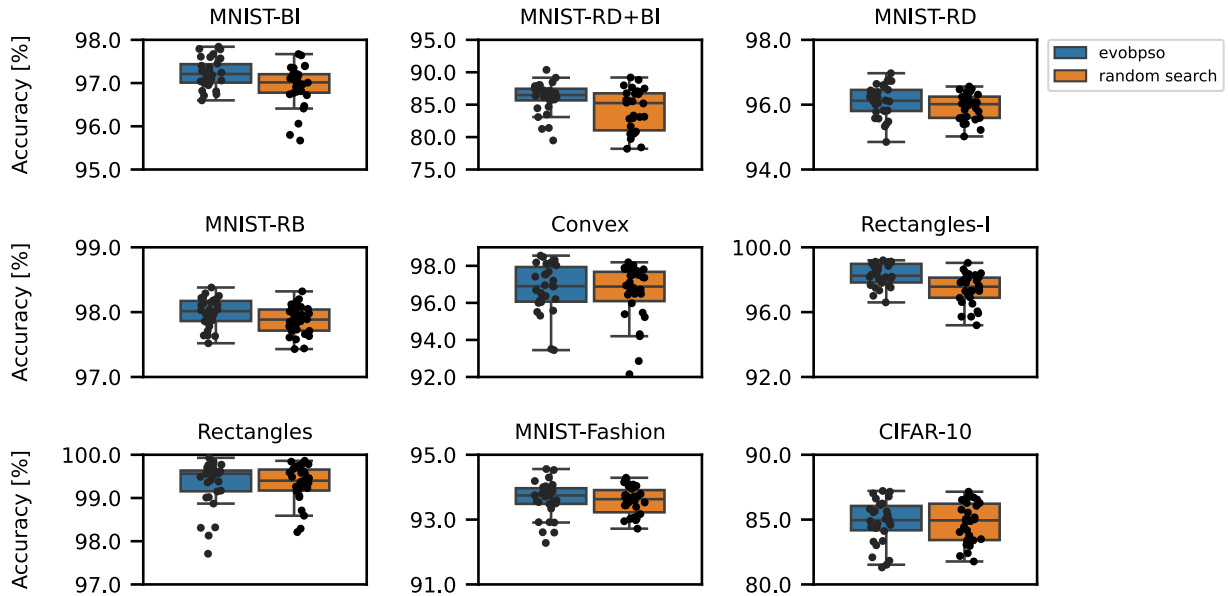


FIGURE 10. Test accuracy rates of evobps and random search in a set of nine benchmark datasets. Individual points in each boxplot mark the accuracy rate of each of the 30 runs.

only sporadic improvements in the validation loss during the optimization phase.

Fig. 7 shows the evolution of the particle size (boxplots) and diversity of the population (thick black line) during one individual execution. The size of each particle is the number of convolutional layers of the model, while the diversity represents the normalized cumulative Hamming distance among all pairs of particles. It can be seen that the evobps algorithm can converge to a few optimum configurations, while random search exhibits more exploratory behavior, as expected.

Fig. 8 shows the mean population fitness during optimization of the CIFAR-10 dataset (for one specific execution). The behavior of evobps is drastically different to that of random search. More specifically, evobps is able to gradually reduce the mean population fitness, indicating the potential of the approach as a NAS optimization algorithm.

Lastly, to gain a more complete understanding of the behavior of evobps in all different datasets, linear regression of the mean population fitness was conducted and the slope value was retrieved. A negative slope would indicate progressive reduction of the mean population fitness, evidencing the algorithm’s consistent performance in a variety of datasets.

Fig. 9 shows the slope values for the mean population fitness linear regression, for the first 10 executions (runs). Firstly, it can be seen that the slopes for random search are mostly around zero. For evobps, most of the slope values were found to be negative (e.g., Rectangles-I, CIFAR-10), but there were also cases where the algorithm struggled to improve the mean fitness (e.g., MNIST-BI).

2) PERFORMANCE

The test error rates of evobps and random search are shown in Table 5, while Fig. 10 shows the accuracy rates graphically. For all nine datasets, the best model generated by evobps had lower test error rates compared to the models created by random search. For example, in the MNIST-RD+BI dataset, evobps obtained a model with a test error rate of 9.63%, while the best model found by random search had a test error rate of 10.81%, a difference of 1.18%. In addition, the average error rate obtained from the models of the evobps algorithm was lower than random search for all datasets.

The performance improvement (PI) of the PSO-based approach compared to random search was calculated as the difference between the mean error rate of evobps and random search (Table 5). The largest PI was 1.9%, observed

in the MNIST-RD+BI dataset, while the second largest PI was 0.89% in the Rectangles-I dataset. In the rest of the datasets, the PI ranged from 0.02% (Convex) to 0.87% (Rectangles).

To identify if the differences between the PSO-based algorithm and random search were statistically significant, a two-sided signed-rank Wilcoxon test was conducted (Table 5). For three out of the nine datasets (MNIST-BI, MNIST-RD+BI & Rectangles-I) the Wilcoxon test resulted in p values smaller than 0.05, indicating that there was enough evidence to reject the null hypothesis of the two algorithms having equal performance. However, for the other six datasets the test could not find statistically significant differences between the two approaches.

VII. CONCLUSION

This work investigated the performance of PSO as a search strategy in NAS, applied to image classification with chain-structured CNNs. Briefly, a PSO-based algorithm was devised, composed of a new binary encoding (Table 1), as well as a modified version of BPSO, and compared to alternative approaches as well as random search. The binary encoding used merges the representation of the pooling layers to that of the convolutional layers, negating the need for additional rules during position hybridization. The BPSO algorithm was adapted in a way to ensure hybridization of the personal and global terms, a feature that is not present in competitor approaches [16], [18].

The results of this work indicate that, overall, evobpso can create competitive models compared to the literature. More specifically, evobpso generated architectures which had the lowest test error rate in three out of the nine datasets utilized (MNIST-RD+BI, MNIST-RB, MNIST-Fashion), while ranking second in two other datasets (MNIST-RD, Rectangles-I). For example, in the MNIST-RD+BI dataset evobpso had a test error rate of 10.84%, while the second best test error rate was 11.47%, achieved by pswvCNN [18]. The test error rate of evobpso in the MNIST-Fashion dataset was 5.44%, equal to that of pswvCNN.

An analysis of the population evolution indicates that evobpso is able to converge to an optimum area of the search space after a few iterations. As can be seen in Fig. 7, after 10 iterations the majority of the particles represent networks with five to eight convolutional neural layers. This is in stark contrast to random search, where the population consistently covers the full search space between one and 18 convolutional layers.

In order to critically evaluate the convergence and limitations of the devised algorithm, the evolution of the mean population fitness for all datasets was investigated (Fig. 8 and Fig. 9). It can be seen that the algorithm has the capacity to iteratively reduce the mean population fitness, indicated by a negative slope in Fig. 9, although it struggles in some datasets (e.g., MNIST-BI). This may explain why evobpso did not produce the best result in all datasets in comparison to the alternative approaches. Therefore, further optimizing the

algorithm to achieve better results in a variety of datasets is one of the future research directions of this work.

Replacing the search strategy with random search resulted in relatively competitive models, frequently with very similar average performance to that of evobpso. Statistically significant differences were found in three out of the nine datasets tested. On the other hand, the best model produced by the evobpso algorithm was consistently better than the best model produced by random search (Table 5).

Further investigation showed that evobpso offered a performance improvement of up to about 1.9% compared to random search (MNIST-RD+BI dataset), although for the majority of the datasets the improvement rate was between 0.02% and 0.89% (Table 5). For the CIFAR-10 dataset, a popular benchmark dataset for image classification, the performance improvement was only 0.2%.

The results presented here highlight that differences in the behavior of the population are not necessarily enough evidence to prove the superiority of an evolutionary NAS algorithm against random search. As shown in representative examples in Fig. 7 and Fig. 8, the population in evobpso successfully converges towards a few good parameter values, with a gradually decreasing mean population fitness. Even though random search does not exhibit any of these properties, it still manages to find very competitive solutions. Thus, for future studies, it is recommended that in addition to the behavior of the algorithm, the final test accuracy rates against baseline measures are reported.

Due to the stochastic nature of the algorithm and CNN training process, the computational complexity cannot be calculated analytically. However, it is expected that the computational complexity of the evobpso algorithm is similar to other, population-based NAS methods presented here [16] [18], [21]. For all such stochastic evolutionary NAS approaches, the computational complexity is dependent on the sum of the computations for updating the particles' positions and the computations for training the deep learning model [51]. Thus, the major factors affecting the runtime are the population size, the number of iterations, the complexity of the deep learning models, and the number of epochs used for training. On the contrary, the random search algorithm does not involve calculations for updating the particles' velocities or positions. As a result, the computational complexity of random search is lower than that of an evolutionary NAS algorithm.

The experimental runtime was also recorded to provide an estimation of the computational cost of the algorithms. In a total of 10 executions, the average running time of the evobpso algorithm in the MNIST-BI dataset was 1.51 hours, with each individual run duration ranging from 1.1 to 1.77 hours. On the contrary, random search had an average running time of 1.44 hours, with a maximum run duration of 1.51 and a minimum of 1.33 hours. In the MNIST-RD+BI dataset, the average run duration of evobpso was 1.31 hours, while that of random search was 1.71 hours. In the current implementation, the most computationally demanding part

of the algorithm is the fitness function evaluation (i.e., model training). For a thorough comparative survey on more computationally efficient approaches, the reader is referred to [52].

Given that the implementation of the evolutionary search strategies is time-consuming and increases the complexity of the algorithm, knowing the expected performance improvement compared to random search is an important consideration for deep learning and NAS practitioners. The current analysis concludes that evobpso, and potentially other PSO-based algorithms, are able to create top architectures for image classification. However, random search is also a viable strategy in NAS due to its good performance and lower complexity.

In this study, PSO was selected as representative of swarm intelligence algorithms due to its good convergence properties and popularity in the NAS field [16], [17], [18], [21], [26], [53]. Besides PSO, other evolutionary and swarm intelligence algorithms have been successfully applied in NAS, such as genetic algorithms [22], [47] and ant colony optimization [54]. However, due to the lack of comparative studies it is currently unclear if any particular algorithm is a better NAS strategy than others, and under what conditions. A recent study found that PSO performs better than ant colony optimization in image classification [53], although results for other approaches are lacking. Therefore, more comparative studies of the behavior and performance of evolutionary and swarm intelligence algorithms in NAS are needed in order to identify the most promising ones.

One limitation of the current study is that only a single set of hyper-parameters was used. The motivation for the selected hyper-parameters' values was to be as close as possible to the algorithms that evobpso was compared with [16], [17], [18]. It is possible that different hyper-parameter values will result in worse or better performance of evobpso. A future research direction of this work is the implementation of self-adaptive NAS algorithms, which would negate the need to manually adjust the hyper-parameters.

In addition, only a single-objective optimization problem was considered, in terms of minimizing the accuracy of image classification. A future research direction is the exploration of multi-objective NAS algorithms aiming to minimize the number of parameters of the model while maximizing performance. Besides, there are various possible search spaces that the current work was not able to cover and will be explored in the future, such as cell-based architectures and vision transformers.

ACKNOWLEDGMENT

The author would like to thank Prof. Greg Stephens for helpful discussions and support for this project. This work was supported by the Scientific Computing and Data Analysis Section of Research Support Division at Okinawa Institute of Science and Technology (OIST).

REFERENCES

- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2017, pp. 2261–2269, doi: 10.1109/CVPR.2017.243.
- [5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [6] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," 2015, *arXiv:1505.04597*.
- [7] T. Elsken, A. Zela, J. H. Metzger, B. Staffler, T. Brox, A. Valada, and F. Hutter, "Neural architecture search for dense prediction tasks in computer vision," 2022, *arXiv:2202.07242*.
- [8] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019, *arXiv:1902.07638*.
- [9] G. Kyriakides and K. Margaritis, "An introduction to neural architecture search for convolutional networks," 2020, *arXiv:2005.11074*.
- [10] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, 2013, pp. I–115–I–123.
- [11] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019, doi: 10.1038/s42256-018-0006-z.
- [12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
- [13] K. Yu, C. Scuti, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019, *arXiv:1902.08142*.
- [14] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," 2018, *arXiv:1802.01548*.
- [15] E. Akkuş, U. Bal, F. Ö. Koçoğlu, and S. Beyhan, "Hyperparameter optimization of pre-trained convolutional neural networks using adolescent identity search algorithm," *Neural Comput. Appl.*, vol. 36, pp. 1523–1537, Nov. 2023. [Online]. Available: <https://link.springer.com/10.1007/s00521-023-09121-8>
- [16] F. E. Fernandes Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019.
- [17] T. Lawrence, L. Zhang, C. P. Lim, and E.-J. Phillips, "Particle swarm optimization for automatically evolving convolutional neural networks for image classification," *IEEE Access*, vol. 9, pp. 14369–14386, 2021.
- [18] D. Elhani, A. C. Megherbi, A. Zitouni, F. Dornaika, S. Sbaa, and A. Taleb-Ahmed, "Optimizing convolutional neural networks architecture using a modified particle swarm optimization for image classification," *Expert Syst. Appl.*, vol. 229, Nov. 2023, Art. no. 120411.
- [19] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017, *arXiv:1705.10823*.
- [20] G. Yuan, B. Wang, B. Xue, and M. Zhang, "Particle swarm optimization for efficiently evolving deep convolutional neural networks using an autoencoder-based encoding strategy," *IEEE Trans. Evol. Comput.*, early access, Feb. 15, 2023, doi: 10.1109/TEVC.2023.3245322.
- [21] T. Lawrence, L. Zhang, K. Rogage, and C. P. Lim, "Evolving deep architecture generation with residual connections for image classification using particle swarm optimization," *Sensors*, vol. 21, no. 23, p. 7936, Nov. 2021.
- [22] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
- [23] A. Ghosh, N. D. Jana, S. Das, and R. Mallipeddi, "Two-phase evolutionary convolutional neural network architecture search for medical image classification," *IEEE Access*, vol. 11, pp. 115280–115305, 2023.

- [24] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE ICNN*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.
- [25] T. M. Shami, A. A. El-Saleh, M. Alswaiti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, "Particle swarm optimization: A comprehensive survey," *IEEE Access*, vol. 10, pp. 10031–10061, 2022.
- [26] A. Sahu, S. K. Panigrahi, and S. Pattnaik, "Fast convergence particle swarm optimization for functions optimization," *Proc. Technol.*, vol. 4, pp. 319–324, Jan. 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2212017312003271>
- [27] A. Marandi, F. Afshinmanesh, M. Shahabadi, and F. Bahrami, "Boolean particle swarm optimization and its application to the design of a dual-band dual-polarized planar antenna," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 3212–3218.
- [28] K. V. Deligkaris, Z. D. Zaharis, D. G. Kampitaki, S. K. Goudos, I. T. Rekanos, and M. N. Spasos, "Thinned planar array design using Boolean PSO with velocity mutation," *IEEE Trans. Magn.*, vol. 45, no. 3, pp. 1490–1493, Mar. 2009.
- [29] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. IEEE Swarm Intell. Symp.*, Apr. 2007, pp. 120–127. [Online]. Available: <http://ieeexplore.ieee.org/document/4223164/>
- [30] Y. Shi and R. C. Eberhart, "Parameter selection algorithm: Convergence analysis and parameter selection," *Inf. Process. Lett.*, vol. 85, no. 6, pp. 317–325, Mar. 2003. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0020019002004477>
- [31] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Evolutionary Programming VII (Lecture Notes in Computer Science)*, vol. 1447, Berlin, Germany: Springer, 1998, pp. 591–600. [Online]. Available: <http://link.springer.com/10.1007/BFb0040810>
- [32] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
- [33] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. 24th Int. Conf. Mach. Learn.* New York, NY, USA: ACM, Jun. 2007, pp. 473–480, doi: [10.1145/1273496.1273556](https://doi.org/10.1145/1273496.1273556).
- [34] P. Singh, S. Chaudhury, and B. K. Panigrahi, "Hybrid MPSO-CNN: Multi-level particle swarm optimized hyperparameters of convolutional neural network," *Swarm Evol. Comput.*, vol. 63, Jun. 2021, Art. no. 100863.
- [35] S. C. Nistor and G. Czibula, "IntelliSwAS: Optimizing deep neural network architectures using a particle swarm-based approach," *Expert Syst. Appl.*, vol. 187, Jan. 2022, Art. no. 115945.
- [36] A. Yang, P. M. Esperança, and F. M. Carlucci, "NAS evaluation is frustratingly hard," 2019, *arXiv:1912.12522*.
- [37] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, in Proceedings of Machine Learning Research, vol. 80, J. Dy and A. Krause, Eds., Jul. 2018, pp. 4095–4104. [Online]. Available: <https://proceedings.mlr.press/v80/pham18a.html>
- [38] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–13. [Online]. Available: <https://openreview.net/pdf?id=S1eYHoC5FX>
- [39] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.
- [40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, in Proceedings of Machine Learning Research, Lille, France, 2015, pp. 448–456. [Online]. Available: <https://proceedings.mlr.press/v37/ioffe15.html>
- [41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
- [42] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using Cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, Mar. 2020.
- [43] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.
- [44] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
- [45] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, USA, 2009.
- [46] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: A simple deep learning baseline for image classification?" *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5017–5032, Dec. 2015.
- [47] X. He, Y. Wang, X. Wang, W. Huang, S. Zhao, and X. Chen, "Simple-encoded evolving convolutional neural network and its application to skin disease image classification," *Swarm Evol. Comput.*, vol. 67, Dec. 2021, Art. no. 100955.
- [48] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size," 2016, *arXiv:1602.07360*.
- [49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [50] F. Miao, L. Yao, and X. Zhao, "Evolving convolutional neural networks by symbiotic organisms search algorithm for image classification," *Appl. Soft Comput.*, vol. 109, Sep. 2021, Art. no. 107537.
- [51] M. S. Sohail, M. O. B. Saeed, S. Z. Rizvi, M. Shoaib, and A. U. H. Sheikh, "Low-complexity particle swarm optimization for time-critical applications," 2014, *arXiv:1401.0546*.
- [52] S. Liu, H. Zhang, and Y. Jin, "A survey on computationally efficient neural architecture search," *J. Autom. Intell.*, vol. 1, no. 1, Dec. 2022, Art. no. 100002.
- [53] S. Lankford and D. Grimes, "Neural architecture search using particle swarm and ant colony optimization," 2024, *arXiv:2403.03781*.
- [54] S. Lankford, "Open-source neural architecture search with ensemble and pre-trained networks," *Int. J. Model. Optim.*, vol. 11, pp. 33–41, May 2021.



KOSMAS DELIGKARIS received the M.S. degree in electrical engineering from University of Twente, The Netherlands, in 2011, and the Ph.D. degree in frontier biosciences from Osaka University, Japan, in 2015.

He is currently a Computer Vision Engineer with Okinawa Institute of Science and Technology, Okinawa, Japan. His research interests include artificial and living neural networks, computer vision, and evolutionary intelligence.

• • •