Evolutionary Optimization of Residual Neural Network Architectures for Modulation Classification

Erma Perenda*, Sreeraj Rajendran*, Gerome Bovet†, Sofie Pollin* and Mariya Zheleva‡

{erma.perenda, sreeraj.rajendran, sofie.pollin}@esat.kuleuven.be,

gerome.bovet@armasuisse.ch, mzheleva@albany.edu

* WaveCore, ESAT, KU Leuven, †Cyber-Defence Campus, armasuisse Science&Technology, ‡Department of Computer Science, University at Albany - SUNY

***Abstract*—Automatic modulation classification receives significant interest in the context of current and future wireless communication systems. Deep learning emerged as a powerful tool for modulation classification, as it allows for joint discriminative features learning and signal classification. However, the optimization of deep neural network architectures for modulation classification is a manual and time-consuming process that requires profound domain knowledge and much effort. Most state-of-the-art solutions focus mainly on classification accuracy, while optimization of network complexity is neglected. This paper presents a novel bi-objective memetic algorithm, BO-NSMA, to search optimal deep neural network architectures for modulation classification to maximize classification accuracy and minimize network complexity. The experiments show that BO-NSMA, with an initial population of six individuals and only ten generations, finds a deep neural network architecture that outperforms all human-crafted architectures. Furthermore, BO-NSMA discovered the first low-complexity Convolutional neural network architecture, which achieves slightly better performance than costly Recurrent neural network architectures, allowing a 2.9-fold reduction in network complexity with 1.43% performance improvement. Compared to counterparts from network architecture search, BO-NSMA finds the best architecture, which achieves up to 18.73% accuracy gain and up to an 82-fold reduction in network complexity. The results are validated using the Wilcoxon signed-rank test.**

***Index Terms*—Modulation Classification, Deep Learning, Network Architecture Search, Multi-objective Genetic Algorithm.**

I. Introduction

AUTOMATIC Modulation Classification (AMC), an intermediate step between signal detection and demodulation, is an integral part of designing an intelligent transceiver for future wireless communication with critical applications in Dynamic Spectrum Access (DSA) and resource allocation. Furthermore, it is a key enabler for many other spectrum sensing applications such as signal monitoring, intruder detection, jammer identification, and numerous regulatory and defense applications.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519

Research in Automatic Modulation Classification (AMC) has been carried out for more than 40 years, using three main methodological themes: Likelihood-Based (LB), Feature-Based (FB), and Deep Learning (DL). LB methods formulate AMC as a multiple composite hypothesis problem, where the number of hypotheses is equal to the number of target modulations [1]. Although optimal in the Bayesian sense, LB methods require prior knowledge about all signal and channel parameters and suffer from high computational cost [2]. On the other hand, FB methods perform classification by utilizing pre-designed discriminative features at a lower computational cost than LB methods [3,4]. FB methods' performance heavily depends on the discriminative nature and noise robustness of the extracted features, and it is very challenging to pre-design the right features [5,6]. In contrast to FB, DL automatically learns radio features from raw In-phase/Quadrature (I/Q) data and outperforms FB methods [7]–[11].

Due to its ability to jointly learn discriminative features from raw I/Q data and perform signal classification based on them, DL has been widely adopted for AMC. Two streams of DL-based methodology can be distinguished: Recurrent Neural Network (RNN) [7] and Convolutional Neural Network (CNN) [8]–[10]. Due to RNNs' higher computational cost and memory requirements, CNNs have been preferred for classification tasks. Deeper CNN architectures have a vanishing gradient problem, making them unsuitable for complex classification tasks [11] because the network performance degrades with depth. Recently, inspired by RNN, new CNN based models such as Residual Neural Network (ResNet) [12] and Aggregated Residual Transformations for Deep Neural Networks (ResNeXt) [13] have been proposed. These models outperform State-of-the-Art (SoA) CNN models, as shown for the ResNet-based AMC model in [8] and ResNeXt-based AMC model in [11]. ResNet and ResNeXt are modularized architectures where the pre-designed blocks are stacked. Several designs of ResNet blocks [12] and ResNeXt blocks [13] have been proposed. Despite the great successes in using Deep Neural Network (DNN) for AMC, designing efficient and accurate DNN architectures is usually a manual, time-consuming process that requires profound domain knowledge. Moreover, many AMC applications run in real-time and require fast inference. The lower the DNN architecture complexity, the faster the inference will be. The following challenges make this process even more difficult.

Immense search space: Even for a simple CNN architecture, the search space is very large, as the degrees of freedom include the number of layers, the number of filters per layer, and the filter size. For example, let us consider a simple ResNet-18 network with 18 layers (of which 16 Convolution

(Conv) and 2 pooling layers). A typical architecture optimization task for ResNet-18 would consider 16 layers with $8, 16, 32, 64$, or 128 filters and a filter size of 1 or 3. This creates a large search space of $(5 \times 2)^{16} = 10^{16}$ possible architectures. A random search of such space can take days or weeks. Recently, two heuristic approaches based on Reinforcement Learning (RL) [14] and Genetic Algorithm (GA) [15]–[17] have been widely adopted in computer vision to automate the Network Architecture Search (NAS). While the former uses RL to guide the search, the latter is a population-based metaheuristic reflecting natural selection [18]. RL-based approaches [14] suffer from prohibitively high computational cost and are not readily applicable to Multi-Objective Optimization Problems (MOOPs) [19]. Generally, in the literature, two distinct approaches exist on multi-objective RL: single-policy and multiple-policy. The single-policy approach [20] performs conversion of a MOOP into a Single-Objective Optimization Problem (SOOP) through the specific scalarization methods, which denote the preferences of objectives. Specific weights represent the preference, and they are different in each run. This approach is redundant in both computation and model representation. The most used scalarization method is a linear combination of weighted objectives. In [21] is shown that any system based on a linear combination of the objectives is incapable of producing a good approximation of the Pareto front for problems that exhibit non-convex regions. Many real-world MOOP problems have that nature, and the scalarization method is an inefficient tool to solve such problems. In contrast, the multiple-policy approach [22] must find multiple policies to satisfy trade-offs between objectives. It can be done simultaneously (in a single run) or iteratively (one policy per run). Multi-policy RL methods still have a high computational cost, even in their expanded and optimized version [22]. As these methods explicitly maintain multiple policies, they are difficult to scale up to high-dimensional preference spaces among the objectives. In contrast, GAs are highly efficient for MOOPs [18] since they can obtain a set of solutions in a single run due to their population-based characteristic. NAS has seldom been considered for DL-based AMC [23].

Dataset-centric solutions: Most existing human-crafted AMC DNN architectures were optimized for a single set of modulations [7,9]. Adding new modulation formats and/or changing the input features are highly likely to deteriorate the DNN performance [11]. Thus, new target classes trigger the re-optimization of the architecture. To make this re-optimization tractable, a flexible search space and encoding scheme for GAs are necessary to make them robust to input feature changes. Most of the GAs for NAS neglect this and require a new search space and encoding scheme when the input feature space changes. [16] proposed pre-designed blocks, which fail on input feature changes as shown later. [23] proposed a shallow CNN architecture that fails on complex feature spaces, as shown in [11].

Maximizing classification accuracy while neglecting network complexity: All human-crafted AMC DNN architectures have focused on maximizing the classification accuracy while reducing network complexity has been neglected. However, such the human-crafted DNN architectures might not be optimized in terms of the connections and hyperparameters values. As already mentioned, GA provides a few techniques to solve MOOP, but GAs applied for AMC's NAS are still single-objective driven [23].

Besides RL and GA, there are a few alternative approaches to optimize DNN architectures. In [24], a human-crafted DNN is pre-selected. Then greedy criteria-based pruning is applied to reduce the number of trainable parameters achieved by pruning unimportant features per layer. The performance of this method depends mainly on the human-crafted initial architecture. Knowledge distillation was considered for NAS in [25], where DNN architecture compression is done by transferring knowledge from a trained teacher network to a smaller and faster student model. This method has a significantly lower computational cost than GA and may arrive at a sub-optimal solution as it does not explore the entire search space. Knowledge distillation can be combined with RL to expedite the convergence of RL-based NAS, as shown in [26]. For AMC applications, it is very important to have DNN architectures with high classification accuracy while keeping the complexity low. The time consumption for searching for such architecture is not critical, allowing us to apply the GA approach, which might find an architecture close to the global optimum.

This paper proposes a novel AMC algorithm called BO-NSMA (Bi-objective Network Search using Memetic Algorithm) to optimize both classification accuracy and network complexity. Memetic algorithm refers to an extension of GA with Local Search (LS) [18]. The main goal of an LS operator is to push candidate solutions towards more promising areas of the search space, where a finding of the optimum solutions is highly likely. Consequently, a carefully designed LS operator can expedite the convergence rate of GA, as it is shown in this work. Optimization of network complexity considers both the connections and hyper-parameters of variable-length network architecture. The key contributions of this paper are summarized below:

- A first study of the use of multi-objective GA-based NAS for AMC. This paper identifies the key components of the proposed BO-NSMA, such as Fitness Sharing (FS), Local Search (LS), and self-adaptivity of mutation and crossover rates that enable it to find a diverse population close to the Pareto optimal front. Note that the Pareto optimal front is a set of non-dominated individuals referred to as Pareto optimal solutions. There is a (possibly infinite) number of Pareto optimal solutions.

- The impact of the search space and encoding on GA convergence rate and AMC's performance is thoroughly explored.
- It is demonstrated that BO-NSMA can find a diverse population very close to the Pareto optimal front for a small set of 6 individuals. Furthermore, it is shown that BO-NSMA outperforms all human-crafted DNN-based AMC by yielding up to cc. 2% gain in accuracy and up to cc. 5-fold reduction in network complexity.
- The new termination criteria are created to balance the trade-off between search duration and gained performance efficiently.

The remainder of the paper is organized as follows. The overview of State-of-the-Art (SoA) GA methods for NAS are presented in Section II. The problem statement is introduced in Section III. Section IV explains the structure of the proposed GA method for AMC's NAS. The deep performance analysis of the proposed method and its comparison with human-crafted AMC methods is presented in Section V. The time complexity, limitations and potential applications of BO-NSMA are discussed in Section VI. The conclusions are briefly presented in Section VII.

II. Related Work

A detailed overview of existing work in the field of NAS research can be found in [27]. As the selection of the GA approach for NAS is justified in the previous section, a detailed overview of GA-based NAS research is given below.

GAs have been successfully used for NAS in image processing [15,16,28]. By encoding the network architecture as a chromosome or individual, GA methods strive to optimize the weights of the DNN architecture and/or the connections and hyper-parameters of the DNN architecture. The literature in GA for NAS can be categorized into two main streams.

A. *Collaborative combination*

GAs optimize both the connections and hyper-parameters, and weights of the DNN architecture by using single or multiple GAs. However, all the proposed GAs assume a fixed-length network with simple architectures with one hidden layer, such as Feed-Forward Neural Network (FFNN) [29,30]. [29] seeks to optimize only weights, while [30] seeks to optimize number of neurons and their weights. Complex network architectures increase individual representations' complexity and result in a computationally expensive search for the optimal weights. On the other hand, back-propagation algorithms have emerged as an efficient method for weights optimization [31]. Furthermore, intelligent weight initialization can slightly boost back-propagation performance, as shown in [15], where weights initialization values, encoded into individuals as the additional hyperparameters, are optimized over generations.

B. *Supportive combination*

In this stream of work, GAs are used to optimize the connections and hyperparameters of the DNN architecture, while the weights are optimized using other algorithms such as the back-propagation [31]. FFNN optimization is proposed in [32], CNN optimization in [15]–[17,33] and RNN optimization in [33,34]. Mostly considered hyperparameters are the number of hidden layers, learning rate, type of optimizer, number of filters, layers' positions, and activation functions. There are a few different research approaches within this stream. They are distinguished by whether they focus on network depth, optimization problem formulation, or the way of DNN architectures building. First, considering the network depth, there are two categories: fixed [17] and variable [15,16,32]–[34] network depth approaches. While the former might waste computational power in cases when the network depth is set to a value higher than optimal, the latter tries to find the optimal network depth and, thus, provides more computationally efficient candidate solutions. Second, considering the optimization problem formulation, there are two categories: single-objective [15,17,33,34] and bi-objective [16,32] optimization approaches. While the former minimizes only classification error or mean squared error, the latter minimizes both the classification error and network complexity. The bi-objective optimization problem is translated into single-objective using the scalarization method in [32] or Pareto Dominance (PD) approach in [16]. The scalarization method is very sensitive to the weighting of the objectives. It may require a large number of iterations to converge to a small part of the Pareto optimal front. Third, considering the way of DNN architectures building, there are two categories: layer stacking [15,17,32]–[34] and block stacking [16] approaches. The layer stacking approach is not preferred for complex classification problems. It requires a deeper network vulnerable to the vanishing gradient problem where gradients become vanishingly small, preventing network training [12]. On the other hand, a careful design of blocks with identity shortcuts makes the network robust to the vanishing gradient problem. These identity shortcuts allow gradient information to pass through the layers, even in deeper networks, making the training independent of network depth. Besides adding the identity shortcuts, the design of blocks highly impacts DNN's performance. For instance, in [16], a few pre-designed blocks are proposed with the same human-designed connection settings. NAS running on such a limited search space might not find an optimal DNN architecture.

In the context of modulation classification, GA methods have been employed to extract and optimize classification features [35]–[38] or to optimize the DNN architecture [23]. However, [23] does not consider the joint optimization of both the connections and the hyperparameters of the AMC's DNN architecture. Moreover, only the simple CNN network architecture has been considered [23]. This paper aims to jointly optimize both the connections and hyperparameters of the AMC's DNN architecture by a novel memetic algorithm that addresses the drawbacks mentioned above of GA methods in image processing.

III. Problem definition

This section gives a mathematical representation of NAS for AMC. The input to AMC is explained as well.

A. Signal model

Before the definition of NAS for AMC, let us introduce the modulated signal as input to the DNN-based modulation classifier. Assume that one active transmitter transmits a signal, $s(t)$, over a dynamic wireless fading channel with an impulse response, h_c. Assuming one antenna at the receiver, the distorted and noise-corrupted received signal, $r(t)$, is given as

$$r(t) = e^{j(\phi_0 - 2\pi\Delta f t)} s(t-\Delta t) \circledast h_c(t) + v(t), \quad (1)$$

where Δt is the timing offset, Δf is the frequency offset, ϕ_0 is the phase offset, and $v(t)$ is Additive White Gaussian Noise (AWGN) with mean 0 and variance $2\sigma_v^2$. It is assumed that the receiver is working at the same center frequency as the transmitter. The received signal, $r(t)$, is sampled in the time domain and N raw I/Q samples are fed to the input of the AMC classifier as its input features. The N raw I/Q samples are referred to as an instance, represented as a matrix with dimensions $2-by-N$, where the first row holds *I* values and the second row holds the corresponding *Q* values. For example, an instance of size $N = 4$ for Binary Phase-Shift Keying (BPSK) modulated signal looks as below

$$[I/Q]_{2x4} = \begin{bmatrix} 0.027 & 0.053 & 0.119 & 0.144 \\ -0.014 & 0.003 & 0.005 & 0.035 \end{bmatrix}. \quad (2)$$

The AMC classifier has a task to correctly select a modulation format from a pool of known N_{mod} candidate modulations.

B. Problem definition of network architecture search for modulation classification

The Network Architecture Search (NAS) for AMC can be treated as a bi-objective optimization problem where classification accuracy should be maximized, and simultaneously, network complexity in terms of computational cost and memory requirements should be minimized. As this work searches for an optimal CNN architecture, network complexity can be roughly approximated as the total number of trainable parameters of the model. A

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

mathematical representation of considered problem is given below.

Let $\mathcal{Z} \subset \Re^{2\times N}$ be the above mentioned feature space of raw I/Q samples and $\mathcal{Y} = \{1, ..., c\}$ be the label space, where c is the number of considered modulation classes. Thus, the training dataset can be defined as $\mathcal{D} = \{(z_i, y_i)\}_{i=1}^{n}$, where $(z_i, y_i) \in (\mathcal{Z} \times \mathcal{Y})$. A classifier is defined as a function that maps the input feature space to label space, $f : \mathcal{Z} \rightarrow \Re^{c}$. The AMC classifier adopts the *Softmax* output layer with cross-entropy loss for classification. Accordingly, the classification risk, which captures the discriminative nature of features learned by DNN, is given as

$$R_{\mathcal{L}}(f) = \mathbb{E}_{\mathcal{D}}\big[\mathcal{L}(f(z;\theta), y_z)\big] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}\log f_j(z_i;\theta), \quad (3)$$

where θ is set of parameters of the classifier, $\mathcal{L}$ is cross-entropy loss, y_{ij} is the label of instance z_i (represented as j'th element of one-hot encoded label), and f_j denotes the j'th element of the classifier function f. The lower the classification risk, the higher classification accuracy p_c will be. Therefore, the joint connections and hyperparameters optimization of the DNN architectures can be formulated as

$$\begin{aligned} &\text{minimize} \quad F(z) = \big(R_{\mathcal{L}}(f(z)), \#\theta\big) \\ &\text{subject to} \quad z \in \mathcal{Z}, f(z) \in \mathcal{A}, \#\theta > 0, \end{aligned} \quad (4)$$

where $\mathcal{A}$ is the architecture search space and $\#\theta$ is the number of trainable parameters out of all classifier parameters θ. Given $\mathcal{A}$, a goal is to find an optimal architecture $f(z)$ for the classifier with the minimal number of trainable parameters $\#\theta$, such that after training those parameters the architecture can achieve the minimal classification risk, $R_{\mathcal{L}}$.

IV. Methodology

This section introduces the proposed BO-NSMA (Bi-objective Network Search Memetic Algorithm), which flowchart is shown in Fig. 1. BO-NSMA considers the block-level design and utilizes Pareto Dominance (PD) with Fitness Sharing (FS) strategy to solve the bi-objective optimization problem. Simultaneously, in contrast to [16], it allows blocks with different connection settings, using expanded hyperparameters search space. As this work considers a complex network architecture, back-propagation is adopted for the weights optimization with the default Xavier weights initializer [31]. All methods mentioned in Section II use the absolute number of iterations to terminate their GAs. This termination criterion is inefficient and may lead to unnecessary computations. Thus, this work employs the averaged Hausdorff distance to avoid it. Further, the exploration and exploitation are enhanced by self-adaptive mutation and crossover rates. In what follows, the components of the proposed BO-NSMA are explained.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

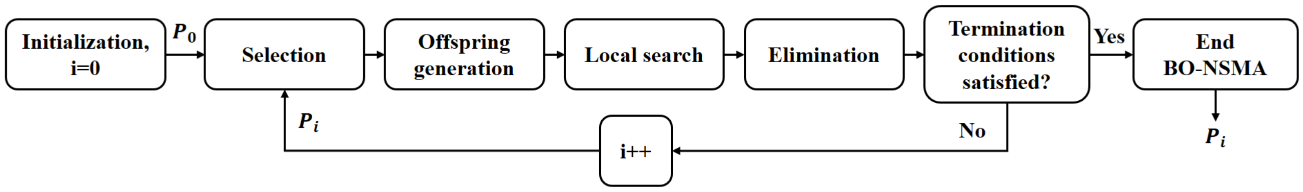

Figure 1: BO-NSMA Overview

A. Search space and encoding

Inspired by the ResNet [12] and ResNeXt [13] architectures, the CNN-based network architecture is designed as a serial fusion of the number of blocks followed by a global pooling layer and several dense layers. Each block is defined as a parallel fusion of w branches with d Conv layers, whose outputs are first concatenated and then merged with the *Identity* branch, as shown in Fig. 2. With the probability of p_{pool}, each block is followed by a pooling layer. Different variants of the merge function, including *Multiply, Add*, and *Concat* are considered. The *Multiply* and *Add* are element-wise operations, while the *Concat* is done along the column axis (axis=1). Traditionally, CNNs capture the spatial properties of the underlying signal as classification features. However, these spatial properties are inherently sensitive to noisy conditions and may suffer significant performance deterioration [11]. This work proposes to address this problem by expanding the search space of the merge function, which might enable the extraction of new features that capture cumulants-like signal properties. To enforce the dimensionality reduction of features space with network depth, the adding of one Conv layer either before the Conv branches and *Identity* branch (Fig. 2(a)) or in the *Identity* branch (Fig. 2(b)) is employed. The first block in the network has a width equal to 1, depth equal to 1, and no *Identity* branch, while for the other blocks, width and depth are randomly chosen from the following ranges: $w \in [1, w_{max}]; d \in [1, d_{max}]$. As dense layers require a higher number of trainable parameters, they are added with a probability of p_{dense}. GA seeks the optimal number of blocks, the number of dense layers, and each block's optimal depth and width. In addition, GA seeks the optimal hyperparameter values for each architecture layer. The search space for hyperparameters is given in Table I. An individual's genotype is given as a list of several blocks,

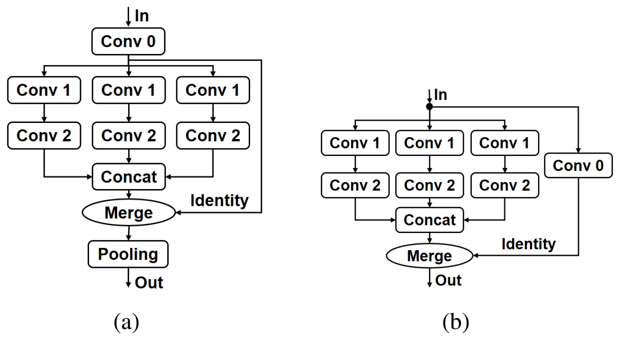

Figure 2: Block structure examples, $w = 3, d = 2$ with (a) Dim. reduction before the Conv and *Identity* branches, and followed by a pooling layer; (b) Dim. reduction in the *Identity* branch, and without a pooling layer.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

Table I. Hyperparameters encoded into individuals

Unit	Hyperparameters	Search space
Network	No. of blocks No. of dense layers	$[1, 10]$ $[0, 4]$
Block	width depth Merge function Dim. reduction Pooling	$[1, 32]$ $[1, 4]$ $\{Add, Multiply, Concat\}$ $\{Before, After\}$ $\{Yes, No\}$
Conv	Filters Kernel size Activation	$\{4, 8, 16, 32, 64, 128\}$ $\{1, 3, 5, 7\}$ $\{relu, selu, tanh, linear\}$
Pooling	Type Kernel size	$\{Max, Average\}$ $\{2\}$
Global pooling	Type	$\{Average, Flatten\}$
Dense	Units Activation	$[32, 256]$ $\{relu, selu, tanh, linear\}$

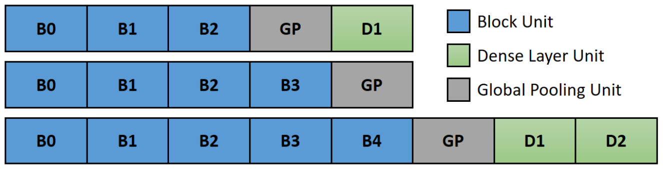

Figure 3: Examples of individual's genotypes.

Table II. BO-NSMA input parameters

Name	Notation	Default Value
Population size	λ	6
Offspring size	μ	6
Max. no. of blocks	N_B	8
Max. no. of Dense layers	N_D	4
Max. block width	w_{max}	32
Max. block depth	d_{max}	4
Probability of adding a Dense layer	p_{dense}	0.4
Probability of adding a Pooling layer	p_{pool}	0.5
Max. allowed no. of trainable parameters	$\#\theta_{max}$	100,000
Absolute no. of iter.	$\mathcal{J}$	10
No. of initialization trials	$\mathcal{I}$	100
No. of iter. for convergence check	$\mathcal{H}$	3
Convergence threshold	ϵ	10^{-4}
No. of epochs	N_{epochs}	10
Probability to flip units in crossover	p_{c_flip}	0.6
Optimal classification accuracy	$\hat{p_c}$	0.9
Optimal no. of trainable parameters	$\hat{\#\theta}$	10,000
Tournament Selection Parameter	k	2
Max. length of individual	L_{max}	20

one global pooling layer, and several or no dense layers. An individual's length, L, is equal to the sum of the number of blocks, the number of dense layers, and one global pooling layer. Fig. 3 presents several examples of individual's genotypes. A phenotype of an individual is a DNN architecture built upon its genotype.

B. *Population initialization*

The search space, given in Table I, might result in a very complex network architecture. Many human-crafted DNN architectures have been proposed for AMC [7,8,11], and can help in the more intelligent design of the population initialization. Therefore, a control parameter referred to as the maximum allowed number of trainable parameters, $\#\theta_{max}$, with a value determined from the human-crafted SoA, is introduced. Even with limited network complexity, there are still many architectures to explore. The process of population initialization is explained in Algorithm 1. The initialization of an individual consists of adding block units (lines $11-17$), adding a global pooling unit (lines $18-19$), and adding dense layers (lines $20-22$). If the individual has a higher number of trainable parameters than $\#\theta_{max}$, it will be discarded and initialized again until the generated candidate satisfies the target number of trainable parameters or the number of trials, $\mathcal{I}$, does not reach maximum value. Although this might prolong the initialization time, it results in a much lower overall time cost induced by alternative complex network architectures.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519

Algorithm 1: Population initialization

```
Input: Input parameters given in Table II
Output: Initialized population $P_0$
1  $P_o \leftarrow \emptyset$
2  for $i \leftarrow 1$ to $\lambda$ do
3      $j \leftarrow 0$
4      while True do
5          $j \leftarrow j + 1$
6          $Individual \leftarrow Null$
7          $n_b \leftarrow$
             Uniformly generate an integer between $[1, N_B]$
8          $n_d = 0$
9          $r \leftarrow$ Uniformly generate a number between $[0, 1]$
10         if $r \leq p_{dense}$ then
11             $n_d \leftarrow$
                 Uniformly generate an integer between $[0, N_D]$
12         $\_list \leftarrow []$
13         $block_0 \leftarrow$ Randomly initialize a block unit with
             $d = 1$, $w = 1$, no the Identity branch, and $p_{pool}$
14         $\_list \leftarrow \_list \cup block_0$
15         for $j \leftarrow 1$ to $n_b$ do
16             $w \leftarrow$
                 Uniformly generate an integer between $[1, w_{max}]$
17             $d \leftarrow$
                 Uniformly generate an integer between $[1, d_{max}]$
18             $block \leftarrow$ Randomly initialize a block unit
                 with $d$, $w$, and $p_{pool}$
19             $\_list \leftarrow \_list \cup block$
20         $gp \leftarrow$ Randomly initialize a global pooling unit
21         $\_list \leftarrow \_list \cup gp$
22         for $j \leftarrow 1$ to $n_d$ do
23             $dl \leftarrow$ Randomly initialize a dense layer unit
24             $\_list \leftarrow \_list \cup dl$
25         $Individual.units \leftarrow \_list$
26         $Individual.accuracy \leftarrow 0.0$
27         $Individual.complexity \leftarrow count\_\#\theta()$
28         if $Individual.complexity < \#\theta_{max}$ or $j == \mathcal{I}$
             then
29             $P_0 = P_0 \cup Individual$
30             break
31 return $P_0$
```

C. Fitness evaluation

The fitness evaluation is performed in three steps: (1) counting of trainable parameters, (2) training of the decoded individual through a predefined number of epochs, N_{epochs}, and (3) evaluating the trained models on the validation dataset. The number of trainable parameters and validation classification accuracy are the objectives that are utilized during offspring generation and elimination. Adam optimizer [39] with a learning rate of 0.001 is adopted in this work. This learning rate is a reasonable trade-off between slow convergence at lower rates and inaccurate results at higher rates. Note that the Adam optimizer provides gradient normalization and momentum which make the learning rate important only for the initial learning before the learning rate is updated. Our selected learning rate is based on recommendations from prior work [39], whereby a learning rate of 0.001 works well for most problems.

D. Offspring generation

GA reflects the process of natural selection, where the fittest individuals are selected for mating to produce offspring for the next generation. Natural selection in GA is performed by selection, crossover, and mutation operators [18].

1) Selection: The deterministic k-tournament [18] is employed to select the individuals for mating without replacement, whereby k individuals are evaluated randomly, and the best one is chosen. The deterministic k-tournament selection does not require any global knowledge of the population, nor a quantifiable measure of quality like the other selection methods like the Roulette wheel method. Instead, it uses an ordering relation to compare and rank any two individuals. Further, the selection

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

pressure is easy to control by varying the tournament size k [40]. Since there are two objectives in considered problem, the well-known concept of Pareto Dominance (PD) is utilized to determine which individual is better. One individual is said to dominate the other if one of the following conditions is satisfied: (1) it has a higher classification accuracy, and a lower or equal number of trainable parameters, or (2) it has a higher or equal classification accuracy, and a lower number of trainable parameters. For each individual, the individuals by which it is dominated are counted. The individual with a lower number of individuals by which it is dominated is treated as better.

2) Crossover: The crossover operator is analogous to natural reproduction and is usually performed with a rate of 1. However, such a rate may result in the mating of parents with poor genes, which leads to poor offspring performance. The survival selection will highly likely eliminate poor offspring, resulting in slowing down or completely halting the GA progress. Thus, it is important to generate good offspring in order to speed up the solution search and increase the offspring's survival rate. To this end, self-adaptive crossover rates inspired by Q-learning [41] are developed. Specifically, a track of whether an individual is good for mating or not is kept by encoding information about its crossover rate, p_{cr}. Each individual in the initialized population P_0 has a crossover rate of 1. The crossover operator takes two parents as inputs, $parent_1$, $parent_2$, and returns two offspring, $offspring_1$, $offspring_2$. Before applying the crossover operation, offspring are pure copies of the selected parents, i.e., $offspring_1$ inherits all properties of $parent_1$, while $offspring_2$ inherits all properties of $parent_2$. After applying crossover operator, the offspring crossover rates are updated as below

$$p_{cr}^{offspring_i} = \gamma * p_{cr}^{parent_i} + (1-\gamma) * p_c^{offspring_i}; i = 1, 2, \tag{5}$$

where γ is the learning rate ranging from 0 to 1, $p_{cr}^{parent_i}$ denotes the crossover rate of $parent_i$, while $p_c^{offspring_i}$ denotes the validation classification accuracy of the $offspring_i$. The value of γ is mostly set to 0.3 in practice, as for higher values, Q-learning becomes unstable [41]. Intuitively, the higher the reward in maximizing the objective score, the higher the crossover rate will be. To avoid duplicates in the population, for offspring generated without crossover operator (copies of their parents), the mutation operator with a rate of 1 is applied. As a crossover operator, the uniform crossover with a flip probability of $p_{c\_flip} = 0.6$ is adopted [18] (see lines $6 - 13$ in Algorithm 3). Uniform crossover is applied to type-aligned units, as shown in Fig. 4.

3) Mutation: As uniform crossover of aligned units does not impact the length of the offspring, each offspring inherits the parent's length. To al-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

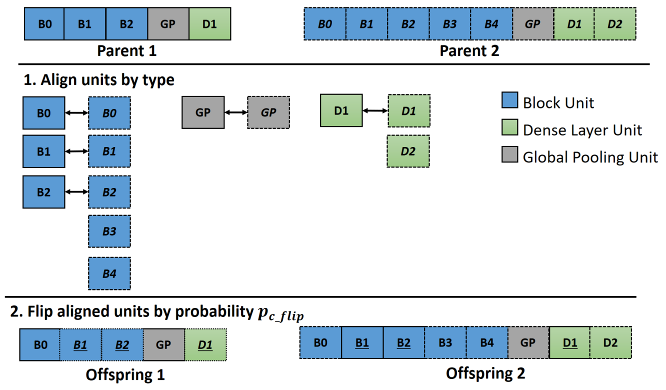

Figure 4: Crossover example

low both offspring length variability and hyperparameters' values changes, the following mutation operators: *Add new, Duplicate, Delete, and Reset* are introduced. With probability p_{m_unit}, which is inversely proportional to the individual's length L, one uniformly chosen mutation operator will be applied to each unit of the individual. The *Add new* operator adds a new unit with randomly selected hyperparameters' values. The *Duplicate* operator adds a unit with the same structure and hyperparameters' settings as the current unit. The *Delete* operator removes the unit. The *Reset* operator keeps the unit with the same structure but randomly changes its hyperparameters' settings. The mutation process is explained in Algorithm 2. Similarly, each individual has encoded information about the mutation rate p_{mr} as the crossover rate, which is adapted over generations. The initialized population has randomly selected mutation rates within a certain range. High mutation rates introduce more exploration in an individual's length. The mutation rates are updated using log-normal transformation [42], as given below:

$$p_{mr}^{new} = \Big(1 + \frac{1 - p_{mr}^{old}}{p_{mr}^{old}} \exp^{-\tau N(0,1)}\Big)^{-1}, \qquad (6)$$

where τ is the adaptation speed control parameter (set to 0.22), and $N(0,1)$ is a normal variable with zero mean value and unit variance. The log-normal transformation of the mutation rates keeps them between 0 and 1, and has been shown as an efficient technique for mutation rate self-adaptation [42].

Algorithm 2: Mutation in BO-NSMA

```
Input: Individual $x$, Input parameters given in Table II
Output: Mutated individual $\hat{x}$
1  $r \leftarrow$ Uniformly generate a number between $[0,1]$
2  if $r > p_m^x$ then
3      return $x$
4  $\hat{x} \leftarrow \emptyset$
5  $p \leftarrow \frac{1}{length(x)}$
6  for $i \leftarrow 1$ to $length(x)$ do
7      $r \leftarrow$ Uniformly generate a number between $[0,1]$
8      $u \leftarrow x[i]$
9      if $r < p$ and $length(\hat{x}) < L_{max}$ then
10         $m \leftarrow$
             Uniformly generate a number between $[0,4)$
11         if $m == 0$ then
12             $\hat{x} \leftarrow \hat{x} \cup u$
13             if $length(\hat{x}) < L_{max}$ then
14                 $b \leftarrow$ Randomly initialize a new unit, $b$
                     with the same type as $u$
15                 $\hat{x} \leftarrow \hat{x} \cup b$
16         if $m == 1$ then
17             $b \leftarrow$ Randomly initialize a new unit, $b$ with
                 the same type as $u$
18             $\hat{x} \leftarrow \hat{x} \cup b$
19         if $m == 2$ then
20             $\hat{x} \leftarrow \hat{x} \cup \emptyset$
21         if $m == 3$ then
22             $\hat{x} \leftarrow \hat{x} \cup u$
23             if $length(\hat{x}) < L_{max}$ then
24                 $\hat{x} \leftarrow \hat{x} \cup u$
25     else
26         $\hat{x} \leftarrow \hat{x} \cup u$
27 if $length(\hat{x}) == 0$ then
28     return $x$
29 else
30     return $\hat{x}$
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

E. Local search

Before applying the elimination strategy, Local Search (LS) is applied to the best and the worst individuals (offspring + parents). The best individuals are non-dominated by any other individual and belong to the Pareto optimal front. In contrast, the worst individuals have the maximum number of individuals by which they are dominated. LS explores the worst individuals' neighbourhood to find their fitter neighbours that might have a chance to survive. LS consists of applying mutation to the selected individual for two runs. The selected un-mutated individual is replaced with its mutated version only if one of the following conditions is satisfied: (1) the mutated individual has a higher classification accuracy, and its number of trainable parameters is not increased more than 5%; (2) the mutated individual has a lower number of trainable parameters, and its classification accuracy is not decreased more than 0.5%.

F. Elimination strategy

Crossover and mutation operators generate μ offspring. The $\lambda + \mu$ strategy is opted for the elimination strategy [18], where parents and offspring are merged, and the λ best individuals are selected for the next generation. The best individuals selection is made according to the modified classification accuracy by using Fitness Sharing (FS) for diversity promotion. FS is the most successful and widely used method for diversity promotion. While [16] uses the crowding method for diversity promotion, this work opted for FS for the following reasons. Although FS has a higher computational cost than crowding, FS tends to encourage searches in unexplored regions of the space and favors the formation of stable subpopulations [43]. In contrast, crowding has difficulties to preserve the stable subpopulations in some cases as a result of replacement errors. The complete overview of methods for diversity promotion can be found in [44]. The modified classification accuracy is given as below:

$$p_c' = p_c * \left[\sum_{r \in N_\sigma^i(x)} 1 - \left(\frac{d(x,r)}{\sigma}\right)^\alpha \right], \tag{7}$$

where σ denotes the threshold of dissimilarity, $d(x,r)$ is the distance between the individual x and the individual r, α is a constant parameter that regulates the shape of the sharing function, and $N_\sigma^i(x)$ denotes the σ neighbourhood of individual x in the current population P_i given as $N_\sigma^i(x) = \{r \in P_i | d(x,r) \leq \sigma\}$.

The distance $d(x,r)$ is calculated as the Euclidean distance between individuals' normalized complexities and classification accuracies as below:

$$d(x,r) = \sqrt{\left(\frac{\#\theta_x - \#\theta_r}{\#\theta_{max}}\right)^2 + (p_{c,x} - p_{c,r})^2}, \tag{8}$$

where $\#\theta_{max}$ is the maximum allowed number of trainable parameters. As the maximum distance can reach $\sqrt{2}$, σ is set to 0.2. The shape parameter α is set to 2, ensuring high diversity pressure in the σ

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

neighborhood.

G. Termination strategy

Unlearned termination criteria present one of the main GA's drawbacks. An infinite number of iterations might be required to reach an optimal global solution in a large search space. As each iteration of GA is very expensive, it is necessary to have the proper termination criteria, which indicates the point in time when further computations become unnecessary as they do not gain substantial performance improvement. The following termination conditions are adopted (lines $27-34$ in Algorithm 3): (1) the number of iterations is greater than or equal to a fixed number, $\mathcal{J}$, decided a priori; (2) an acceptable solution is reached - GA will terminate if the algorithm generates an individual with classification accuracy higher than a predefined value $\hat{p_c}$ and trainable parameters lower than a predefined number, $\hat{\#\theta}$; (3) when there has not been any improvement in the population for the last $\mathcal{H}$ iterations, i.e., differences between the generations in the last $\mathcal{H}$ iterations are less than a certain convergence threshold, ϵ. Each generation is represented as a set of Pareto points with size equal to the population size, λ. A Pareto point denotes an individual, x, represented by an ordered pair $(1-p_{c,x}, \#\theta_x)$ of its properties. To measure the similarity between two Pareto sets, the well-known averaged Hausdorff distance is utilized, which is a widely used tool to measure the distance between different objects in several research fields [45]. The averaged Hausdorff distance between two Pareto sets in BO-NSMA, $U=\{u_1,u_2,...,u_\lambda\}\subset\mathbf{R}^2$ and $V=\{v_1,v_2,...,v_\lambda\}\subset\mathbf{R}^2$ is defined as below:

$$\Delta_p(U,V)=\max\big((\frac{1}{\lambda}\sum_{i=1}^{\lambda}dist(u_i,V)^p)^{1/p}, (\frac{1}{\lambda}\sum_{i=1}^{\lambda}dist(v_i,U)^p)^{1/p}\big), \quad (9)$$

where $dist(u_i,V)$ is the minimal Euclidean distance from u_i to set V, $dist(v_i,U)$ is the minimal Euclidean distance from v_i to set U, and p is the control factor for outliers' penalty. The higher the value of p, the more penalized are the outliers. Since Δ_p is used as the termination condition, p is set to 1.

V. Performance evaluation

This section explains the experimental setup used for simulations including the selected baselines, datasets and implementation details. The obtained results are presented and analysed.

A. Experimental setup

1) Baselines: This work employs four baselines from the literature that human experts manually design: LSTM [7], ResNeXt [11], ResNet [8], and 1D-CNN [8]. Furthermore, this work employs two of the newest baselines from GA NAS in image processing: NSGA-Net [16] and EvoCNN [15]. The former is a bi-objective optimization with adopted PD for block-level NAS, while the latter is a single-objective optimization for layer stacking NAS. The

Algorithm 3: BO-NSMA

```
   Input: Input parameters given in Table II
   Output: Population
1  $P_o \leftarrow$ Initialize population using Algorithm 1
2  $i \leftarrow 1$
3  while True do
4      $P_i \leftarrow P_{i-1}$
5      for $j \leftarrow 1$ to $\mu/2$ do
6          $parent1, parent2 \leftarrow$ run k-tournament selection
             without replacement
7          $r \leftarrow$ Uniformly generate a number between $[0, 1]$
8          $crossover\_done \leftarrow False$
9          if $r < \big(p_{cr}^{parent1} + p_{cr}^{parent2}\big)/2$ then
10             offspring1, offspring2 $\leftarrow$ crossover(parent1,
                 parent2)
11             $crossover\_done \leftarrow True$
12         else
13             offspring1 $\leftarrow$ parent1
14             offspring2 $\leftarrow$ parent2
15         $r \leftarrow$ Uniformly generate a number between $[0, 1]$
16         if $r < p_{mr}^{offspring1}$ or
             $crossover\_done == False$ then
17             offspring1 $\leftarrow$ mutation(offspring1)
                 using Algorithm 2
18         $r \leftarrow$ Uniformly generate a number between $[0, 1]$
19         if $r < p_{mr}^{offspring2}$ or
             $crossover\_done == False$ then
20             offspring2 $\leftarrow$ mutation(offspring2)
                 using Algorithm 2
21         update $p_{cr}^{offspring1}$ and $p_{cr}^{offspring2}$ by the Eq.
             (5)
22         update $p_{mr}^{offspring1}$ and $p_{mr}^{offspring2}$ by the Eq.
             (6)
23         Fitness evaluation of $offspring1$, $offspring2$
24         $P_i \leftarrow P_i \cup \{offspring1, offspring2\}$
25     $LocalSearch(P_i)$
26     $P_{i+1} \leftarrow Elimination(P_i)$
27     if $i == \mathcal{J}$ then
28         stop BO-NSMA!
29     else if $\exists$ Individual, $x \in P_i,\ p_c^x \geq \hat{p_c}$ and $\#\theta^x \leq \hat{\#\theta}$
         then
30         stop BO-NSMA!
31     else if $\forall j \in [0, \mathcal{H}), \Delta(P_{i-j}, P_{i-j-1}) \leq \epsilon$ then
32         stop BO-NSMA!
33     else
34         $i \leftarrow i + 1$
35 return $P_i$
```

NSGA-Net searches for the network architecture based on a few pre-designed blocks with fixed hyperparameters. In contrast, the EvoCNN seeks to optimize each layer's hyperparameters in the DNN architecture, including the weights initialization values. As those baselines are applied for the 2D image processing problem, each 2D layer in the architectures is replaced with a corresponding 1D layer while keeping all other hyperparameters the same.

2) Datasets: Two modulation sets are used: (1) *a Baseline set*, containing $N_{mod} = 11$ low-order modulation formats: BPSK, QPSK, 8-PSK, 16/64-QAM, PAM4, GFSK, CPFSK, BFM, DSB-AM and SSB-AM; and (2) *an Extended set*, containing the simple ones and 9 additional modulations: OQPSK, 32/128/256-QAM, 16/32/64/128/256-APSK ($N_{mod} = 20$). The I/Q samples are generated at increasing Signal-Noise Ratio (SNR) (-6 dB to 18 dB). The performance of DNN models for different channel models (AWGN, Rayleigh, Rician) follows the same performance trends as long as there is enough labeled data at disposal, as shown in [11]. In this work, the emphasis is on the performance evaluation of BO-NSMA, where channel modeling does not play an important role. Thus, the channel is modeled as simple AWGN. For each combination of SNR and modulation type, 1000 instances are generated with a size of $N = 128$ and $N = 1024$ for the baseline set and the extended set, respectively. A seed is used to generate random mutually exclusive instance indices, which are then used to split the data into three subsets: training, validation, and testing at a ratio of 80:10:10, respectively.

3) Implementation details: BO-NSMA is implemented in Python [46]. Each evaluated AMC method is implemented using TensorFlow [47]. The fitness evaluation training is performed over $N_{epochs} = 10$ epochs and a batch size of 256. The models are trained and evaluated on a GPU server with eight Nvidia RTX 2080Ti cards. The default values of BO-NSMA input parameters (see Table II) are kept constant over all experiments. The population size is set to a low value, $\lambda = 6$, as each individual's evaluation is computationally heavy. The k parameter for deterministic tournament selection is set to 2, which gives a high chance that each individual is selected for crossover operator. All presented classification accuracies are averaged over the whole SNR range of $[-6, 18]$ dB.

B. Results

1) Performance evaluation of BO-NSMA: Besides GA's performance over generations common to any GA, there are two qualitative metrics to be considered to assess how good a certain multi-objective GA is for a given problem: (1) population accuracy, that is to determine how similar the population is to the Pareto optimal front, and (2) population diversity, that is to evaluate how well distributed individuals are in the population. Note that the Pareto optimal front denotes the set of non-dominated individuals. Keeping that in mind, below is justified why BO-NSMA is designed as described in Section IV by using the baseline set of modulations. As an evaluation with the extended dataset follows the same performance trends as with the baseline dataset, the conclusions given below remain valid for the extended dataset. BO-NSMA denotes the proposed GA with applied FS, LS, high mutation rate, $p_{mr} \in [0.5, 1]$, and self-adaptive crossover rates. To evaluate their impacts on the mentioned qualitative metrics over generations, seven experiments are run: (1) BO-NSMA, (2) BO-NSMA with low mutation rate $p_{mr} \in [0.05, 0.2]$, (3) BO-NSMA without FS in elimination and keep the λ individuals with the highest classification accuracy, (4) BO-NSMA with low mutation rate $p_{mr} \in [0.05, 0.2]$ and without FS in elimination and keep the λ individuals with the highest classification accuracy, (5) BO-NSMA without LS, (6) BO-NSMA with low mutation rate $p_{mr} \in [0.05, 0.2]$ and without LS, (7) BO-NSMA with constant crossover rates equal to 1.

Fig. 5 presents the performance of BO-NSMA over generations showing average classification accuracy for the entire population (top left), an average number of trainable parameters for the entire population (top right), and the maximum classification accuracy of the best individual in the population (bottom). Note that the classification accuracy values are averaged over the whole SNR range of $[-6, 18]$ dB. Fig. 6 (right) presents Pareto front approximation for the 10th generation. The Pareto optimal front is illustrated with the solid line in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

Fig. 6 (right), and the best-performing method should follow closely it. Fig. 6 (left) shows the Pareto set difference between two population generations which is important for GA convergence rate monitoring.

a) BO-NSMA performance over generations: As BO-NSMA without FS focuses only on maximizing the classification accuracy, it achieves the best average and maximum accuracy at the tenth generation (see Fig. 5 (top left and bottom)). High mutation rates introduce a high level of search exploration, with an unpredictable impact on the GA performance. LS in combination with high mutation rates always improves accuracy, for BO-NSMA with and without FS. However, BO-NSMA without LS operates better for low mutation rates. Consequently, more search exploration increases the average number of trainable parameters. Each BO-NSMA with low mutation rates converges to almost the same number of trainable parameters, around $60k$. In contrast, BO-NSMA cases with high mutation rates result in around $80k$ trainable parameters (see Fig. 5 (top right)). Further, Fig. 5 proves that LS expedites the convergence rate of GA. BO-NSMA with constant crossover rate, $p_{cr} = 1$ likely mates the poor parents, leading to worse performance. It has a 5% lower average accuracy and 2% lower maximum accuracy at the tenth generation than BO-NSMA with self-adaptive crossover rate.

b) Population accuracy and diversity: Fig. 6 shows that BO-NSMA with FS provides a diverse population for both mutation rates. On the other hand, BO-NSMA without FS and with high mutation rates converges to one optimal Pareto point after the sixth generation, while BO-NSMA without FS and with low mutation rates slowly converges, and at the tenth generation two optimal Pareto points can still be noticed. BO-NSMA with high mutation rates finds six diverse individuals and two optimal Pareto points, while BO-NSMA with low mutation rates finds five diverse individuals and three optimal Pareto points (Fig. 6 (right)). The individuals found by BO-NSMA with high mutation rates are closer to the Pareto optimal front compared to BO-NSMA with low mutation rates. Thus, it can be stated that BO-NSMA with high mutation rates found the best population in terms of both population accuracy and diversity.

To sum up, BO-NSMA without FS achieves the best average and maximum accuracy but at the cost of diversity loss. In contrast, BO-NSMA with FS slowly converges to the optimal Pareto points, but it provides a diverse population. High mutation rates enable a more accurate population, closer to the Pareto optimal front.

2) Impact of the search space and encoding on performance: The solutions found by any GA heavily depend on the given search space and individuals' representation. In order to assess how

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

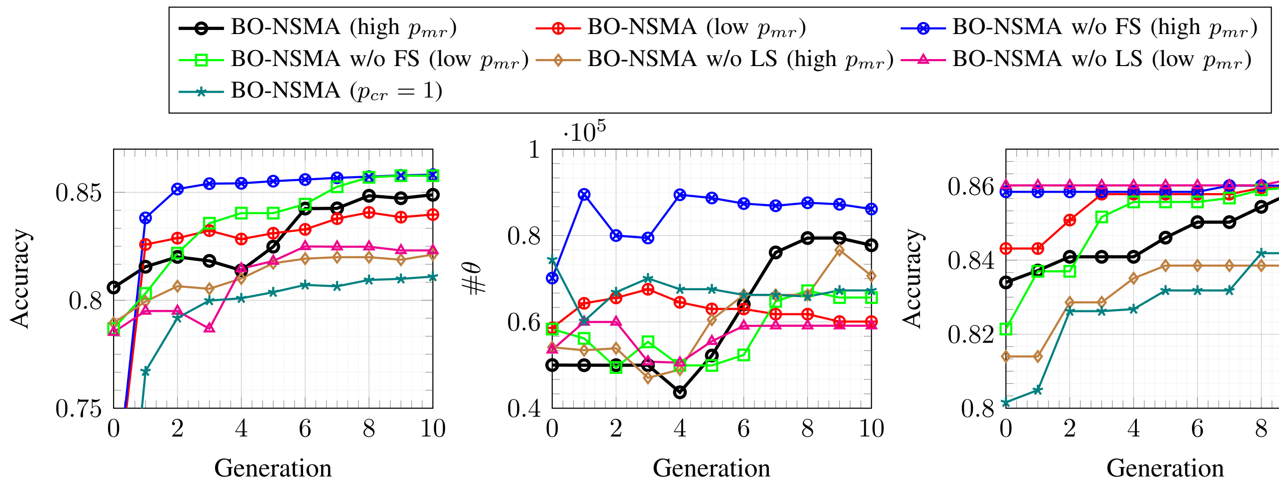

Figure 5: BO-NSMA components and their impact on: average accuracy (top left); average number of trainable parameters (top right); maximum accuracy (bottom) over generations.

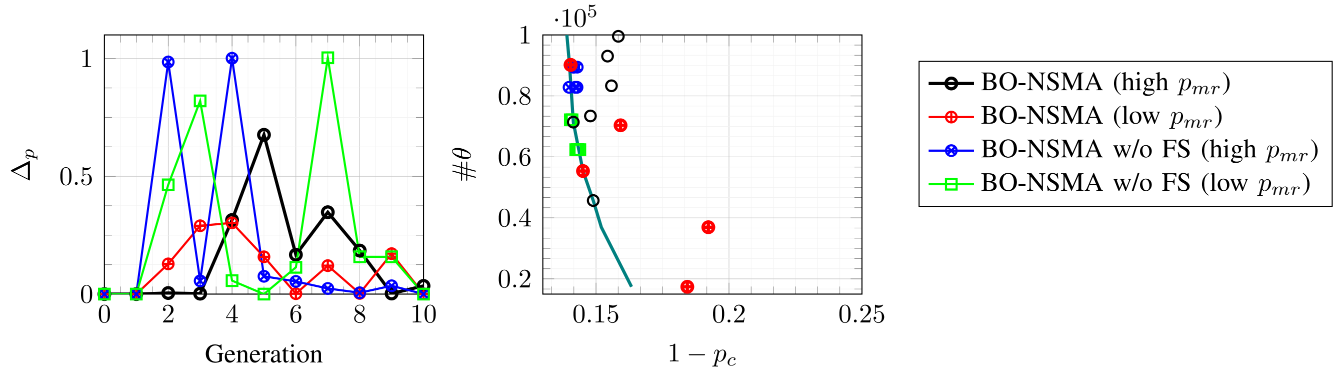

Figure 6: The averaged Hausdorff distance of two subsequent generations (left). Pareto front approximation for Generation 10 (right).

good is the proposed search space, its benefits are explored versus well-known ResNet blocks [12] and simple networks with layer stacking, while keeping all components of BO-NSMA (FS, LS, offspring generation) the same. ResNet blocks by design have $d = 3$, $w = 1$, and *Add* as merge function [12]. Thus, to demonstrate the impact of encoding, five experiments are run: (1) BO-NSMA with the proposed encoding, (2) BO-NSMA with layer stacking, (3) BO-NSMA with ResNet block stacking, (4) NSGA-Net [16] with their pre-designed blocks, and (5) EvoCNN [15]. Each experiment uses the baseline set of modulations. EvoCNN stacks the Conv, pooling and dense layers (the maximum number of layers is set to 15).

Fig. 7 shows that BO-NSMA with the proposed block design has 4% higher average classification accuracy and 4% higher maximum achieved classification accuracy after the sixth generation than BO-NSMA with layer stacking and the ResNet block stacking. Furthermore, BO-NSMA with ResNet block stacking finds the population with the lowest average number of trainable parameters. NSGA-Net achieves very poor AMC performance where each found architecture is overfitting. Although NSGA-Net employs PD, its proposed architecture search space with pre-designed blocks and the fixed hyperparameters values prevents GA from optimizing such architectures according to the considered problem. In contrast, EvoCNN gives a much higher number of degrees of freedom for hyperparameters values, resulting in complex network architectures compared to NSGA-Net and BO-NSMA. Moreover, EvoCNN for population size $\lambda = 6$ prematurely converges after the second generation to one non-optimal Pareto point. The high number of degrees of freedom for hyperparameters values and uncontrolled population initialization require a higher population size to avoid premature convergence. Thus, EvoCNN is run for $\lambda = 30$. Fig. 7 shows that EvoCNN with a higher population size will take a longer time to converge, whereas a higher population size will increase its chance to find at least one Pareto optimal point.

3) Comparison with baselines: Finally, the performance of the best individual found by BO-NSMA is compared with selected baselines for both sets of modulations. All models are trained for 80 epochs and evaluated on the testing dataset. Table III presents classification accuracy averaged over all SNRs for the baseline and extended datasets mentioned in Section V-A2, respectively, whereas Fig. 8 presents accuracy across SNRs for the baseline dataset.

LSTM [7] is the best-performing SoA architecture for AMC evaluated on the baseline dataset, which achieves an average accuracy of 86% with over $200k$ trainable parameters. BO-NSMA finds the first genetically-optimized architecture, which achieves slightly higher average accuracy (an improvement of 1.43%) while reducing the number of trainable parameters to $69k$ (a 2.90-fold reduction in the number of trainable parameters). LSTM with default training parameters given in [7] fails to converge for the extended dataset. Next in terms of achieved performance are ResNet and ResNeXt architectures. For the baseline dataset, BO-NSMA achieves 2.02% and 1.88% accuracy gain over ResNet and ResNeXt, respectively, while keeping the number of trainable parameters over three times lower compared to ResNet. For the extended dataset, BO-NSMA achieves 3.49% and 2.80% accuracy gain over ResNet and ResNeXt, respectively, while keeping the number of trainable parameters over $4.1x$ lower compared to ResNet.

While exploring BO-NSMA's performance across SNR (Fig. 8) for the baseline dataset, note that it gets 3% higher classification accuracy at mid-SNR, which illustrates its robustness to noise compared to other baselines. BO-NSMA can also successfully optimize CNN with layer stacking, achieving the architecture with 1.7% and 2.6% higher classification accuracy at $1.7x$ and $1.9x$ lower number of trainable parameters compared to 1D-CNN [8] for the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

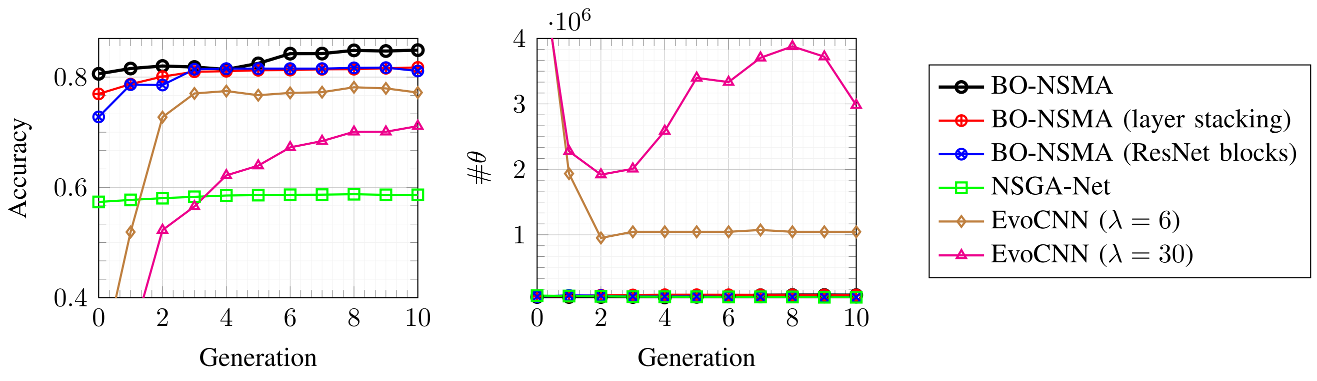

Figure 7: The search space and encoding impact on: average accuracy (top left) and average number of trainable parameters (top right) over generations.

Table III. BO-NSMA top-1 accuracy and corresponding $\#\theta$ vs baselines

Model	Baseline dataset		Extended dataset	
	Acc.(%)	$\#\theta$	Acc. (%)	$\#\theta$
LSTM [7]	86.17	200,075	46.49	201,236
ResNet [8]	85.58	255,115	79.71	313,620
ResNeXt [11]	85.72	85,051	80.40	86,212
1D-CNN [8]	82.01	100,811	79.54	142,932
BO-NSMA	**87.60**	**68,851**	**83.20**	**76,372**
EvoCNN [15]	80.30	450,157	64.47	6,261,901
BO-NSMA (layers)	83.73	83,115	82.12	72,708
BO-NSMA (ResNet blocks)	85.07	58,591	82.09	45,284

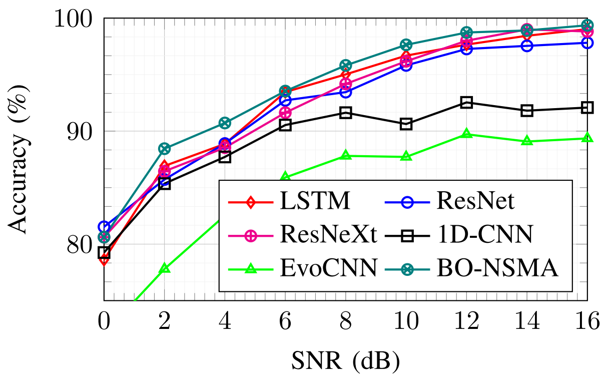

Figure 8: Classification accuracy across SNR (baseline dataset).

baseline dataset and extended dataset, respectively. Similarly, BO-NSMA finds a better architecture with ResNet blocks which has 0.5% lower and 2.4% higher classification accuracy at $4.0x$ and $6.9x$ lower number of trainable parameters than ResNet for the baseline dataset and extended dataset, respectively. BO-NSMA with layer stacking achieves to find a solution within 10 generations and population size of 6 that has 3.43% and 18.2% accuracy gain at $5.41x$ and $78.7x$ lower number of trainable parameters than its counterpart EvoCNN for the baseline dataset and extended dataset, respectively.

4) Statistical significance testing: Table III shows the BO-NSMA and EvoCNN performance of the best-found individual through multiple runs. However, GAs have a stochastic and non-deterministic nature; and thus, running one GA twice on the same optimization problem usually produces different results. A statistical test should then be applied to establish whether there is enough empirical evidence to claim a difference between the two algorithms. BO-NSMA is compared with its GA's counterparts: EvoCNN and NSGA-Net. As NSGA-Net has limited search space, it is unsuitable for AMC, and it is not compared with BO-NSMA. BO-NSMA is compared with EvoCNN for both

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

datsets. The Wilcoxon signed-rank test [48], with significance level $\alpha = 0.05$, was used to quantify the significance of the comparison. The Wilcoxon signed-rank test is a non-parametric statistic test used for comparing two paired sets of observations whose difference comes from a zero median distribution. In [49], it is shown that parametric tests are unsuitable for statistical analysis of evolutionary algorithms for continuous optimization problems, while non-parametric tests such as the Wilcoxon signed-rank test are good tools for such analysis. A null hypothesis H_0 is defined as *"there is no difference between BO-NSMA and EvoCNN"*. In this paper, the classification accuracy of the best-found individual was used for the comparison.

The statistical analysis of best-found individuals for BO-NSMA and EvoCNN is given in Table IV. For the baseline dataset, the classification accuracy of the best individuals found by BO-NSMA lies between 85.2% and 87.6%, while the best individuals found by EvoCNN have the classification accuracy within the wider range of $[67.6, 80.3]\%$. The mean value of the classification accuracy of the best individuals found by BO-NSMA is 9.8% lower for the baseline dataset and 24.5% for the extended dataset. The mean value of the number of trainable parameters of the best individuals found by BO-NSMA is $8.9x$ lower for the baseline dataset and $117.23x$ lower for the extended dataset. The statistical analysis given in Table IV clearly indicates that BO-NSMA outperforms EvoCNN in both classification accuracy and network complexity for both datasets.

The Wilcoxon signed-rank test is run to check if those results are significant. The Wilcoxon signed-rank test is also done for the human-crafted baselines. The human-crafted baselines are deterministic, so the results shown in Table III can be used as constant over multiple runs. The obtained p-value for each BO-NSMA counterpart is lower than the significance level ($\alpha = 0.05$) (see Table V). Thus, with a high level of confidence, it can be stated that BO-NSMA outperforms EvoCNN and the human-crafted baselines. Furthermore, the results are statistically significant.

VI. Discussion and research directions

In this section, the findings from the experimental results are utilized to discuss BO-NSMA time complexity and limitations, which could provide valuable insights on the applications of the proposed BO-NSMA method. In the end, the potential future research directions are outlined.

A. BO-NSMA time complexity

Given the steps in Algorithm 3, the time complexity of each component of BO-NSMA is as follows.

1) Population initialization: Let $T(x)$ and $E(x)$ be the time to compute the number of trainable parameters and the time to evaluate the classification accuracy of an individual x.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Table IV. BO-NSMA and EvoCNN statistical comparison

Metric	Baseline dataset		Extended dataset	
	BO-NSMA	EvoCNN	BO-NSMA	EvoCNN
Max. Acc. (%)	87.6	80.3	83.7	64.5
Min Acc. (%)	85.2	67.6	80.2	45.3
Mean. Acc. (%)	86.5	76.7	82.2	57.7
Max $\#\theta$	99,643	1,679,792	98,020	50,803,314
Min $\#\theta$	21,803	193,874	51,252	814,226
Mean $\#\theta$	68,405	611,113	81,179	9,517,301
Mean Search Time (h)	5.65	0.57	32.2	1.85

Table V. The p-values obtained by the Wilcoxon signed-rank test

Model	p-value	
	Baseline dataset	Extended dataset
LSTM [7]	0.004	0.005
ResNet [8]	$2.47 \cdot 10^{-6}$	0.005
ResNeXt [11]	$3.69 \cdot 10^{-6}$	0.0068
1D-CNN [8]	$1.64 \cdot 10^{-6}$	0.005
EvoCNN [15]	$1.73 \cdot 10^{-6}$	0.005

The minimum time complexity of population initialization is given as $\mathcal{O}(\lambda \times (E(x)+T(x)))$, while maximum time complexity is given as $\mathcal{O}(\lambda \times E(x) + \mathcal{I} \times \lambda \times T(x))$, where $\mathcal{I}$ is the maximum number of trials allowed in the initialization step. As $E(x) >> T(x)$, it is the dominant term; thus, the time complexity for the initialization step is $\mathcal{O}(\lambda \times E(x))$.

2) Selection: the selection step is done through binary tournament and PD. Two iterating loops through the entire population are required to count the number of dominant individuals for each individual in the population. Thus, the time complexity of the selection step is $\mathcal{O}(\lambda^2)$.

3) Offspring generation: Let $C(x_1, x_2)$ and $M(x)$ be the time to perform crossover of two individuals, x_1, x_2, and the time to perform mutation of an individual x. The time complexity of the offspring generation step is given as $\sum_{i=1}^{\lambda/2} \left[(p_{cr}^{x_{2i-1}} + p_{cr}^{x_{2i}})/2 \times C(x_{2i-1}, x_{2i})\right] + \sum_{i=1}^{\lambda} \left[p_{mr}^{x_i} \times M(x_i)\right]$. By focusing only on the dominant terms, the time complexity for the offspring generation step is $\mathcal{O}(\lambda)$.

4) Local search: Local search is applied to both parents and offspring. The time complexity of the LS is given as $\sum_{i=1}^{\lambda+\mu} \left[p_{ls}^{x_i} \times 2E(x_i)\right]$, where $p_{ls}^{x_i}$ denotes the probability that an individual x_i satisfies the conditions for LS. LS is applied to the best and worst individuals, and it will likely be done for the whole population. Thus, the time complexity of the LS step in the worst case is $\mathcal{O}(2 \times (\lambda + \mu)) = \mathcal{O}(4\lambda)$.

5) Elimination: The elimination step adopts FS. Let $D(x_1, x_2)$ be the time to calculate the distance between two individuals, x_1, x_2 by Eq. 8. The time required to calculate distances for the entire population $\lambda + \mu$ has the time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

complexity of $\mathcal{O}((\lambda+\mu)^2 \times D(x_1,x_2))$. The selection of λ individuals for the next generation by using Eq. 7 has a time complexity of $\mathcal{O}(\lambda(\lambda+\mu))$.

By summing up all five steps, focusing on the dominant terms and removing any constants, the overall BO-NSMA time complexity, over $\mathcal{J}$ generations, is given as $\mathcal{O}\big(\mathcal{J}(6\lambda+7\lambda^2)\big) \approx \mathcal{O}\big(\mathcal{J}\lambda^2\big)$. EvoCNN has a time complexity of $\mathcal{O}\big(\mathcal{J}\lambda\big)$, while NSGA-Net has a time complexity of $\mathcal{O}\big(\mathcal{J}\lambda\log\lambda\big)$ [50]. The average time complexity in hours for BO-NSMA and EvoCNN running for the baseline dataset is $5.65h$ and $0.57h$, respectively. Whereas, the average time complexity in hours for BO-NSMA and EvoCNN running for the extended dataset is $32.2h$ and $1.85h$, respectively (see Table IV). The extended dataset requires a longer time to obtain fitness of the population.

Next it is evaluated whether EvoCNN can find a better individual within the same time window as BO-NSMA. Thus, one more experiment is run where EvoCNN has 6h to search for the best individual for the baseline dataset. The λ is set to 36 and $\mathcal{J}$ is set to 20 for that experiment. In 6h, EvoCNN finds the best individual with classification accuracy of 81.89% and number of trainable parameters of $133,307$. EvoCNN slightly improves the classification accuracy and reduces the network complexity by $4.5x$ of its best-found individual for a longer search period. To sum up, for the same time complexity, BO-NSMA outperforms EvoCNN in terms of both classification accuracy and network complexity. BO-NSMA's best-found individual has a 4.6% higher classification accuracy and 50% lower network complexity.

B. BO-NSMA limitations and applications

Based on the analysis given above, the following limitations of BO-NSMA with insights into its potential applications are identified.

1) High time complexity: It is shown that BO-NSMA finds a DNN architecture that outperforms each human-crafted DNN architecture within $5.65h$ and $32.2h$ for the baseline dataset and extended dataset, respectively. Manual searching for an optimal DNN architecture can take days or months. EvoCNN cannot find a better individual even within the same search time window as BO-NSMA. For offline searches where the computational resources are not critical, BO-NSMA is an efficient tool for NAS. No one GA will be an optimal choice for NAS running on the edge devices with low computational power.

2) Control parameters settings: Although BO-NSMA features self-adaptive crossover and mutation rates, a few control parameters are set manually. The control parameters for FS are set based on prior knowledge of the target optimal solutions (classification accuracy close to 100% and the number of trainable parameters lower than the maximum

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2021.3137519, IEEE Transactions on Cognitive Communications and Networking

allowed number of trainable parameters). The maximum number of trainable parameters is set based on expert knowledge of the task of interest. The best-known human-crafted DNN architectures for AMC are explored, and their trainable parameter count is utilized as a range for BO-NSMA. The control parameters for FS might not be optimal if BO-NSMA is applied to non-classification tasks. The maximum allowed number of trainable parameters would require some prior knowledge about a considered task. Thus, to be applied to non-classification tasks, those control parameter values have to be revisited.

C. Future research directions

In this paper, the performance evaluation of BO-NSMA is done only for AMC. It would be interesting to explore BO-NSMA performance for other classification tasks such as image classification in computer vision. BO-NSMA's sensitivity to the manual control parameters for other tasks should be evaluated. Future research could also consider learning those parameters automatically as it is already done for crossover and mutation rates. LS and FS are computationally costly. Having smart policies to turn them on/off over generations would lead to a significant decrease in the time complexity of BO-NSMA. Those policies should achieve a good trade-off of time complexity and convergence. Although the number of trainable parameters is a valuable metric of network complexity for CNN-based architectures, future research would benefit from a more refined metric for network complexity, such as inference time.

VII. Conclusions

Although DNNs have achieved remarkable results for AMC, the manual optimization of their architectures is challenging due to the immense search space. In addition, a given optimized architecture often does not properly transfer when input features change, triggering repetitive and tedious optimization. Automated Network Architecture Search (NAS) using Genetic Algorithm (GA)s has received considerable attention in computer vision. However, a smooth transfer of those methods to time-series problems such as modulation recognition results in suboptimal performances, as it is shown for NSGA-Net. Thus, this paper proposes BO-NSMA, a novel bi-objective memetic algorithm for joint architecture and network complexity optimization for DNN-based modulation recognition applications. The Local Search (LS) is added to GA to accelerate the convergence rate and to enhance performance. Following extensive experimentation, it is shown that BO-NSMA finds a diverse population very close to the Pareto optimal front defined by performance and complexity. The architecture found by BO-NSMA outperforms all human-crafted DNNs. Moreover, it is demonstrated that BO-NSMA does not have a premature convergence problem for low population

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

size, as is the case with its counterpart EvoCNN. The significance of obtained results is proved by using the Wilcoxon signed-rank test.

Acknowledgment

This work has been partially supported by the Swiss Federal Office for Defence Procurement (armasuisse) (project code Aramis CYD-C-2020003) and the NATO Science for Peace and Security Programme under grant G5461.

References

[1] F. Hameed, O. A. Dobre, and D. C. Popescu, "On the likelihood-based approach to modulation classification," *IEEE Transactions on Wireless Communications*, vol. 8, no. 12, pp. 5884–5892, 2009.

[2] J. L. Xu, W. Su, and M. Zhou, "Likelihood-Ratio Approaches to Automatic Modulation Classification," *IEEE Trans. Systems, Man, and Cybernetics Part C: Applications and Reviews*, vol. 41, pp. 455–469, July 2011.

[3] S. Majhi, R. Gupta, *et al.*, "Hierarchical Hypothesis and Feature-Based Blind Modulation Classification for Linearly Modulated Signal," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, 2017.

[4] F. Yang, B. Hao, *et al.*, "A Method of High-Precision Signal Recognition Based on Higher-Order Cumulants and SVM," in *2018 5th International Conference on Systems and Informatics (ICSAI)*, pp. 455–459, 2018.

[5] W. Xiong, P. Bogdanov, and M. Zheleva, "Robust and Efficient Modulation Recognition Based on Local Sequential IQ Features," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1612–1620, 2019.

[6] X. Zhang, J. Sun, and X. Zhang, "Automatic Modulation Classification Based on Novel Feature Extraction Algorithms," *IEEE Access*, vol. 8, pp. 16362–16371, 2020.

[7] S. Rajendran, W. Meert, *et al.*, "Deep Learning Models for Wireless Signal Classification with Distributed Low-Cost Spectrum Sensors," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433–445, 2018.

[8] T. O'Shea, T. Roy, and T. C. Clancy, "Over the Air Deep Learning Based Radio Signal Classification.," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, 2018.

[9] T. J. O'Shea and J. Corgan, "Convolutional radio modulation recognition network.," *International Conference on Engineering Applications of Neural Networks.*, pp. 213–226, 2016.

[10] Y. Wang, M. Liu, *et al.*, "Data-Driven Deep Learning for Automatic Modulation Recognition in Cognitive Radios," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 4074–4077, 2019.

[11] E. Perenda, S. Rajendran, G. Bovet, S. Pollin, and M. Zheleva,

"Learning the unknown: Improving modulation classification performance in unseen scenarios," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021.

[12] K. He, X. Zhang, *et al.*, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[13] S. Xie, R. Girshick, *et al.*, "Aggregated Residual Transformations for Deep Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

[14] Z. Zhong, J. Yan, *et al.*, "Practical Block-Wise Neural Network Architecture Generation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2423–2432, 2018.

[15] Y. Sun, B. Xue, *et al.*, "Evolving Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2020.

[16] Z. Lu, I. Whalen, *et al.*, "Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2020.

[17] A. Shrestha and A. Mahmood, "Optimizing Deep Neural Network Architecture with Enhanced Genetic Algorithm," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 1365–1370, 2019.

[18] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd ed., 2015.

[19] B. Wu, X. Dai, *et al.*, "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, 2019.

[20] P. Vamplew, R. Dazeley, and C. Foale, "Softmax exploration strategies for multiobjective reinforcement learning," *Neurocomputing*, vol. 263, pp. 74–86, 2017.

[21] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, "On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts," in *AI 2008: Advances in Artificial Intelligence* (W. Wobcke and M. Zhang, eds.), (Berlin, Heidelberg), pp. 372–378, Springer Berlin Heidelberg, 2008.

[22] T. H. F. de Oliveira, L. P. de Souza Medeiros, A. Neto, and J. D. Melo, "Q-managed: A new algorithm for a multiobjective reinforcement learning," *Expert Syst. Appl.*, vol. 168, p. 114228, 2021.

[23] S. Wei, S. Zou, *et al.*, "Automatic Modulation Recognition Using Neural Architecture Search," in *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD IS)*, pp. 151–156, 2019.

[24] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning," *CoRR*, vol. abs/1611.06440, 2016.

[25] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-Wisely Supervised Neural Architecture Search With Knowledge Distillation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pp. 1986–1995, IEEE, 2020.

[26] V. Nekrasov, H. Chen, C. Shen, and I. Reid, "Fast Neural Architecture Search of Compact Semantic Segmentation Models via Auxiliary Cells," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9118–9127, 2019.

[27] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *J. Mach. Learn. Res.*, vol. 20, p. 1997–2017, Jan. 2019.

[28] A. Bakhshi, N. Noman, *et al.*, "Fast Automatic Optimisation of CNN Architectures for Image Classification Using Genetic Algorithm," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1283–1290, 2019.

[29] W. M. Jenkins, "Neural Network Weight Training by Mutation," *Comput. Struct.*, vol. 84, no. 31–32, p. 2107–2112, 2006.

[30] A. Nadi, S. S. Tayarani-Bathaie, and R. Safabakhsh, "Evolution of neural network architecture and weights using mutation based genetic algorithm," in *2009 14th International CSI Computer Conference*, pp. 536–540, 2009.

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, JMLR Workshop and Conference Proceedings, 13–15 May 2010.

[32] M. A. J. Idrissi, H. Ramchoun, *et al.*, "Genetic algorithm for neural network architecture optimization," in *2016 3rd International Conference on Logistics Operations Management (GOL)*, pp. 1–4, 2016.

[33] R. Akut and S. Kulkarni, "NeuroEvolution: Using Genetic Algorithm for optimal design of Deep Learning models.," in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–6, 2019.

[34] R. A. Viswambaran, G. Chen, *et al.*, "Evolving Deep Recurrent Neural Networks Using A New Variable-Length Genetic Algorithm," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2020.

[35] N. Ahmadi and R. Berangi, "Modulation classification of QAM and PSK from their constellation using Genetic Algorithm and hierarchical clustering," in *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pp. 1–5, 2008.

[36] M. W. Aslam, Z. Zhu, and A. K. Nandi, "Automatic Modulation Classification Using Combination of Genetic Programming and KNN," *IEEE Transactions on Wireless Communications*, vol. 11, no. 8, pp. 2742–2750, 2012.

[37] S. Huang, Y. Jiang, *et al.*, "Automatic Modulation Classification of Overlapped Sources Using Multi-Gene Genetic Programming With Structural Risk Minimization Principle," *IEEE Access*, vol. 6, pp. 48827–48839, 2018.

[38] R. Dai, Y. Gao, *et al.*, "Multi-objective Genetic Programming based Automatic Modulation Classification," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2019.

[39] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," *San Diego: The International Conference on Learning Representations (ICLR)*, vol. 1, no. 1, 2015.

[40] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM Journal on Computing*, vol. 2, no. 2, pp. 88–105, 1973.

[41] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[42] T. Bäck and M. Schütz, "Intelligent Mutation Rate Control in Canonical Genetic Algorithms.," in *Lecture Notes in Computer Science*, pp. 158–167, 06 1996.

[43] B. Sareni and L. Krahenbuhl, "Fitness sharing and niching methods revisited," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 3, pp. 97–106, 1998.

[44] S. W. Mahfoud, *Niching Methods for Genetic Algorithms*. PhD thesis, USA, 1996. UMI Order No. GAX95-43663.

[45] O. Schutze, X. Esquivel, *et al.*, "Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 504–522, 2012.

[46] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[47] M. Abadi, A. Agarwal, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[48] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[49] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization," *Journal of Heuristics*, vol. 15, pp. 617–644, 2009.

[50] M. Jensen, "Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.