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 (AMC) receives significant interest in the context of current and future wireless communication systems. Deep learning emerged as a powerful AMC tool, as it allows for the joint learning of discriminative features, and signal classification. However, the optimization of Deep Neural Network (DNN) architectures for AMC is a manual and time-consuming process that requires profound domain knowledge and much effort. Moreover, most proposed solutions focus mainly on classification accuracy, while optimization of network complexity is neglected. In this paper, we propose a novel bi-objective memetic algorithm, BO-NSMA, to search optimal DNN architectures for AMC to maximize classification accuracy and minimize network complexity. We show that BO-NSMA, with a small initial population of six individuals and only ten generations, finds a DNN architecture that outperforms all human-crafted State-of-the-Art (SoA) models. BO-NSMA discovered the first low-complexity Convolutional Neural Network (CNN)-based model, which achieves slightly better performance than costly Recurrent Neural Network (RNN)-based approaches, allowing a 2.8-fold reduction in network complexity with 0.7% performance improvement. Compared to its counterparts from Network Architecture Search (NAS), BO-NSMA finds the best architecture, which achieves up to 18.24% accuracy gain and up to a 78.71-fold reduction in network complexity.**

***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.

Due to its ability to jointly learn discriminative features from raw In-phase/Quadrature (I/Q) data and perform signal classification based on them, Deep Learning (DL) has been widely adopted for AMC. We distinguish two streams of DL-based methodology: RNN [1] and CNN [2,3]. 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 not preferable for complex classification tasks [4] because the network performance degrades with depth. Recently, inspired by RNN, new CNN based models such as Residual Neural Network (ResNet) [5] and Aggregated Residual Transformations for Deep Neural Networks (ResNeXt) [6] have been proposed. These models outperform SoA CNN models, as shown for the ResNet-based AMC model in [2] and ResNeXt-based AMC model in [4]. ResNet and ResNeXt are modularized architectures where the pre-designed blocks are stacked. Several designs of ResNet blocks [5] and ResNeXt blocks [6] have been proposed. Despite the great successes in using 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 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. Given ResNet-18 with 18 layers (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) [7,8] and Genetic Algorithm (GA) [9]–[11] have been widely adopted in computer vision to automate the NAS. While the former uses RL to guide the search, the latter is a population-based metaheuristic reflecting natural selection [12]. RL-based approaches [7,8] suffer from prohibitively-high computational cost and are not readily applicable to Multi-Objective Optimization Problems (MOOPs) [13]. In contrast, GAs are highly-efficient for MOOPs [12]. NAS has seldom been considered for AMC [14,15].

Dataset-centric solutions: Most existing human-crafted AMC DNN architectures were optimized for a single set of modulations [1,16]. Adding new modulation formats and/or changing the input features are highly likely to deteriorate the DNN performance [4]. Thus, new target classes trigger re-optimization of the architecture. To make this re-optimization tractable, we need to design a flexible search space and encoding scheme for GAs to make them robust to input feature changes. Most of the GAs for NAS neglect this [11,14], and require a new search space and encoding scheme when the input feature space changes. [11] proposed pre-designed blocks, which fails on input feature changes as shown later. [14] proposed a shallow 2-layers network architecture that fails on complex feature spaces as shown in [4].

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 [14,15].

Besides RL and GA, there are a few alternative approaches to optimize DNN architecture. In [17], a human-crafted DNN is pre-selected, and then greedy criteria-based pruning is applied to reduce the number of trainable parameters, which is achieved by pruning unimportant features per layer basis. The performance of this method depends largely on the human-crafted initial architecture. Knowledge distillation was considered for NAS in [18,19], 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. 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 provide optimal global architecture.

In this paper, we propose 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) [12]. Optimization of network complexity considers both the connections and hyperparameters of variable-length network architecture. The key contributions of this paper are summarized below:

- We are the first to apply multi-objective GA-based NAS for AMC. We identify the key components of our proposed BO-NSMA, such as Fitness Sharing (FS), LS, and self-adaptivity of mutation and crossover rates that enable it to find a diverse population close to the Pareto optimal front.
- We explore the impact of the search space and encoding on GA convergence rate and AMC's performance.
- We demonstrate that BO-NSMA can find a diverse population very close to the Pareto optimal front for a small set of 6 individuals. We show 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.
- We create the termination criteria that efficiently balances the trade-off between search duration and

gained performances.

II. Related Work

GAs have been successfully used for NAS in image processing [9,11,20]. 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 hyperparameters of the DNN architecture. We can categorize the literature in GA for NAS in two main streams.

*1) **Collaborative combination**:* GAs optimize both the connections and hyperparameters, 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, such as Feed-Forward Neural Network (FFNN) [21]–[23]. 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 [24]. Furthermore, intelligent weight initialization can slightly boost back-propagation performance, as shown in [9], where weights initialization values encoded into individuals as the additional hyperparameters are optimized over generations.

*2) **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 [24]. FFNN optimization is proposed in [25], CNN optimization in [10,20,26,27] and RNN optimization in [26,28]. Mostly considered hyperparameters are the number of hidden layers, learning rate, type of optimizer, number of filters, layers' positions, and activation functions. We can distinguish a few different research approaches within this stream. First, considering the network depth, we can separate them into two categories: fixed [10,27] and variable [20,26,28] 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, we can separate them into two categories: single-objective [32] and bi-objective [11,25] 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 [25] or Pareto Dominance (PD) approach in [11]. The scalarization method is very sensitive to the weighting of the objectives and may require a large number of iterations in order to converge to a small part of the Pareto optimal front. Third, considering the way of DNN architectures building, we can separate them into two categories: layer stacking [9,20] and block stacking [11] approaches. The layer stacking approach is not preferred for complex classification problems, as it requires a deeper network vulnerable to the vanishing gradient problem where gradients become vanishingly small, preventing network training [5,6]. 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 [11], 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 [29]–[32] or to optimize the DNN architecture [14,15]. However, none of these prior work considers the joint optimization of both the connections and the hyperparameters of the AMC's DNN architecture. Moreover, only the simple CNN or FFNN network architectures have been considered [14,15]. In this paper, we aim to jointly optimize both the connections and hyperparameters of the AMC's DNN architecture by a novel memetic algorithm that addresses the aforementioned drawbacks of GA methods in image processing.

III. Methodology

In this section, we introduce our proposed BO-NSMA (Bi-objective Network Search Memetic Al-

gorithm). BO-NSMA considers the block-level design and utilizes PD with a novel FS strategy to solve the bi-objective optimization problem. Simultaneously, in contrast to [11], it allows blocks with different connection settings, using expanded hyperparameters search space. As we consider a complex network architecture, we choose back-propagation for the weights optimization with the default Xavier weights initializer [24]. All works 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, we employ the averaged Hausdorff distance to avoid it. Further, we enhance exploration and exploitation by self-adaptive mutation and crossover rates. In what follows, we state the optimization problem and explain the components of the proposed BO-NSMA.

A. Problem definition

The 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 we search for an optimal CNN architecture, network complexity can be roughly approximated as the total number of trainable parameters of the model. We now give a mathematical representation of our problem.

Let $\mathcal{X} \subset \Re^{2\times N}$ be the feature space and $\mathcal{Y} = \{1, ..., c\}$ be the label space, where N is the instance size and c is the number of considered modulation classes. An instance is a vector of I/Q samples. Thus, our training dataset can be defined as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where $(x_i, y_i) \in (\mathcal{X} \times \mathcal{Y})$. A classifier is defined as a function that maps the input feature space to label space, $f : \mathcal{X} \rightarrow \Re^c$. 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

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

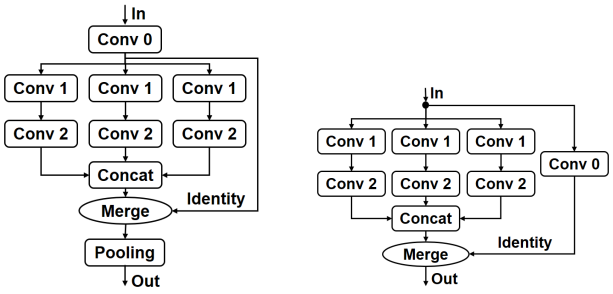
where θ is set of parameters of the classifier, $\mathcal{L}$ is cross-entropy loss, y_{ij} is the label of instance x_i (represented as j'th element of one-hot encoded label), and f_j denotes the j'th element of the classifier function f. The lower the classification risk, the higher classification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

$$\begin{aligned} \text{minimize} \quad & F(x) = \big(R_{\mathcal{L}}(f(x)), \#\theta\big) \\ \text{subject to} \quad & x \in \mathcal{X}, f(x) \in \mathcal{A}, \#\theta > 0, \end{aligned} \quad (2)$$

where $\mathcal{A}$ is the architecture search space and $\#\theta$ is the number of trainable parameters out of all classifier parameters θ. Given $\mathcal{A}$, we seek to find an optimal architecture $f(x)$ 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, $R_{\mathcal{L}}$.

B. Search space and encoding

Inspired by the ResNet [5] and ResNeXt [6] architectures, we design the CNN-based network architecture 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. 1. With the probability of p_{pool}, each block is followed by a pooling layer. We explore different variants of the merge function, including *Multiply, Add*, and *Concat*. 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 [4]. We propose 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, we employ to add one Conv layer either before the Conv branches and *Identity* branch (Fig. 1(a)) or in the *Identity* branch (Fig. 1(b)). 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



(a) (b)

Figure 1: 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.

Table I. Hyperparameters encoded into individuals

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

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 is represented as a list of several blocks, one global pooling layer, and several or no dense layers. A sum of the number of blocks, the number of dense layers, and one global pooling layer denotes an individual's length, L. Fig. 2 presents several examples of individuals.

C. 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 [1,2,4], and can help in the more intelligent

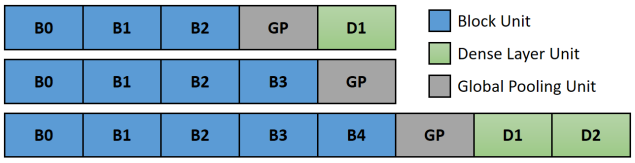


Figure 2: Examples of individuals

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 param.	$\#\theta_{max}$	100,000
Absolute no. of iter.	$\mathcal{J}$	10
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

design of the population initialization. Therefore, we introduce a control parameter referred to as the maximum allowed number of trainable parameters, $\#\theta_{max}$, with a value determined from the human-crafted SoA. 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. Although this might prolong the initialization time, it results in a much lower overall time cost induced by alternative complex network architectures.

D. Fitness evaluation

The fitness evaluation is performed in three steps: (1) counting of trainable parameters, (2) training of

Algorithm 1: Population initialization

Input: Input parameters given in Table II
Output: Initialized population P_0

$P_o \leftarrow \emptyset$
for $i \leftarrow 1$ ***to*** λ **do**
  while *True* **do**
    $Individual \leftarrow Null$
    $n_b \leftarrow$ Uniformly generate an integer between $[1, N_B]$
    $n_d = 0$
    $r \leftarrow$ Uniformly generate a number between $[0, 1]$
    if $r \leq p_{dense}$ **then**
      $n_d \leftarrow$ Uniformly generate an integer between $[0, N_D]$
    $\_list \leftarrow []$
    $block_0 \leftarrow$ Randomly initialize a block unit with $d = 1$, $w = 1$, no the Identity branch, and p_{pool}
    $\_list \leftarrow \_list \cup block_0$
    for $j \leftarrow 1$ ***to*** n_b **do**
      $w \leftarrow$ Uniformly generate an integer between $[1, w_{max}]$
      $d \leftarrow$ Uniformly generate an integer between $[1, d_{max}]$
      $block \leftarrow$ Randomly initialize a block unit with d, w, and p_{pool}
      $\_list \leftarrow \_list \cup block$
    $gp \leftarrow$ Randomly initialize a global pooling unit
    $\_list \leftarrow \_list \cup gp$
    for $j \leftarrow 1$ ***to*** n_d **do**
      $dl \leftarrow$ Randomly initialize a dense layer unit
      $\_list \leftarrow \_list \cup dl$
    $Individual.units \leftarrow \_list$
    $Individual.accuracy \leftarrow 0.0$
    $Individual.complexity \leftarrow count\_\#\theta()$
    if $Individual.complexity < \#\theta_{max}$ **then**
      $P_0 = P_0 \cup Individual$
      break
return P_0

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. We use Adam optimizer [33] with a learning rate of 0.001. This learning rate is a reasonable trade-off between slow convergence at lower rates and inaccurate results at higher rates.

E. Offspring generation

GA reflects the process of natural selection, where the fittest individuals are selected for mating in order to produce offspring for the next generation. Natural selection in GA is performed by selection, crossover and mutation operators [12].

1) Selection: We employ the deterministic k-tournament [12] to select the individuals for mating without replacement, whereby k individuals are evaluated randomly, and the best one is chosen. Since there are two objectives in our problem, we utilize the well-known concept of Pareto Dominance (PD) 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, we count the individuals by which it is dominated. 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, we develop self-adaptive crossover rates inspired by Q-learning [34]. Specifically, we keep track of whether an individual is good for mating or not by encoding information about its crossover rate, p_{cr}. Each individual in the initialized population P_0 has a crossover rate of 1. Offspring crossover rates are updated as below:

$$p_{cr}^{offspring} = \gamma * p_{cr}^{parent} + (1 - \gamma) * p_c^{offspring}, \quad (3)$$

where γ is learning rate ranging from 0 to 1, p_{cr}^{parent} denotes the parent crossover rate, while $p_c^{offspring}$ denotes the validation classification accuracy of the offspring. The value of γ is mostly set to 0.3 in practice, as for higher values, Q-learning becomes unstable [34]. Intuitively, the higher the reward in maximizing the objective score, the higher the crossover rate. To avoid duplicates in the population, for offspring generated without crossover operator (copies of their parents), we apply the mutation operator with a rate of 1. As a crossover operator,

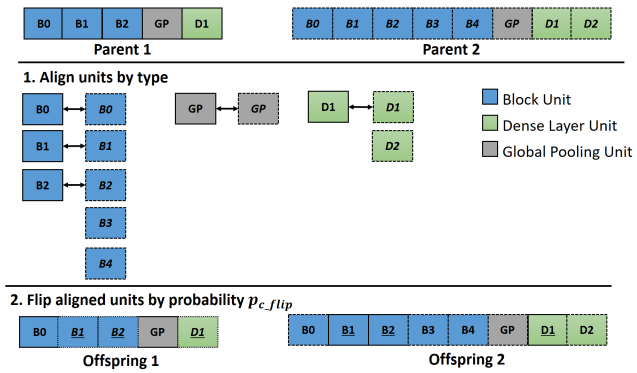


Figure 3: Crossover example

we adopt the uniform crossover with a flip probability of $p_{c\_flip} = 0.6$ [12] (see lines $6-13$ in Algorithm 2). Uniform crossover is applied to type-aligned units, as shown in Fig. 3.

3) Mutation: As uniform crossover of aligned units does not impact the length of the offspring, each offspring inherits the parent's length. To allow offspring length variability, we introduce the following mutation operators: *Add new, Duplicate, Delete, and Reset*. 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. 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 [35] as given below:

$$p_{mr}^{new} = \big(1 + \frac{1 - p_{mr}^{old}}{p_{mr}^{old}} \exp^{-\tau N(0,1)}\big)^{-1}, \tag{4}$$

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 [35].

F. Local Search (LS)

Before applying the elimination strategy, we apply LS 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%.

G. Elimination strategy

Crossover and mutation operators generate μ offspring. We opted for $\lambda + \mu$ strategy for the elimination strategy [12], 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. The modified classification accuracy is given as below:

$$p_c' = p_c * \Big[\sum_{y \in N_\sigma^i(x)} 1 - \big(\frac{d(x,y)}{\sigma}\big)^\alpha\Big], \tag{5}$$

where σ denotes the threshold of dissimilarity, $d(x,y)$ is the distance between the individual x and the individual y, 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 $N_\sigma^i(x) = \big\{y \in P_i | d(x,y) \leq \sigma\big\}$. The distance $d(x,y)$ is calculated as the Euclidean distance between individuals' normalized complexities and classification accuracies as below:

$$d(x,y) = \sqrt{\left(\frac{\#\theta_x - \#\theta_y}{\#\theta_{max}}\right)^2 + (p_{c,x} - p_{c,y})^2}, \tag{6}$$

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

H. Termination strategy

Unlearned termination criteria present one of the main GA's drawbacks. In a large search space, an infinite number of iterations might be required to reach an optimal global solution. As each iteration of GA is very expensive, it is necessary to have the right termination criteria, which indicates the point in time when further computations become unnecessary as they do not gain substantial performance improvement. We adopt the following termination conditions (lines $27-34$ in Algorithm 2): (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. To measure the similarity between two Pareto sets, we utilize the averaged Hausdorff distance [36]. The averaged Hausdorff distance between two sets, $X = \{x_1, x_2, ..., x_n\} \subset \mathbf{R}^k$ and $Y = \{y_1, y_2, ..., y_m\} \subset \mathbf{R}^k$ is defined as below:

$$\Delta_p(X,Y) = \max\big((\frac{1}{N}\sum_{i=1}^{N} dist(x_i,Y)^p)^{1/p}, (\frac{1}{M}\sum_{i=1}^{M} dist(y_i,X)^p)^{1/p}\big), \quad (7)$$

where $dist(x_i, Y)$ is the minimal Euclidean distance from x_i to set Y, $dist(y_i, X)$ is the minimal Euclidean distance from y_i to set X, 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.

Algorithm 2: 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 update $p_{cr}^{offspring1}$ and $p_{cr}^{offspring2}$ by the Eq. (3)
12 $crossover\_done \leftarrow True$
13 **else**
14 offspring1 $\leftarrow$ parent1
15 offspring2 $\leftarrow$ parent2
16 $r \leftarrow$ Uniformly generate a number between $[0, 1]$
17 **if** $r < p_{mr}^{offspring1}$ *or* $crossover\_done == False$ **then**
18 offspring1 $\leftarrow$ mutation(offspring1)
19 $r \leftarrow$ Uniformly generate a number between $[0, 1]$
20 **if** $r < p_{mr}^{offspring2}$ *or* $crossover\_done == False$ **then**
21 offspring2 $\leftarrow$ mutation(offspring2)
22 update $p_{mr}^{offspring1}$ and $p_{mr}^{offspring2}$ by the Eq. (4)
23 Fitness evaluation of offspring1, offspring1
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

IV. Performance evaluation

A. Experimental setup

1) Baselines: We employ four baselines from the literature that are manually designed by human experts: LSTM [1], ResNeXt [4], ResNet [2], and 1D-CNN [2]. Furthermore, we employ two of the newest baselines from GA NAS in image processing: NSGA-Net [11] and EvoCNN [9]. 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 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, we replaced each 2D layer in the architectures with a corresponding 1D layer while keeping all other hyperparameters the same.

2) Datasets: We use two modulation sets: (1) *Simple set*, containing 11 low-order modulation formats: BPSK, QPSK, 8-PSK, 16/64-QAM, PAM4, GFSK, CPFSK, BFM, DSB-AM and SSB-AM; and (2) *Complex set*, containing the simple ones and 9 more modulations: OQPSK, 32/128/256-QAM, 16/32/64/128/256-APSK. 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 follows the same trends as long as there is enough labelled data at disposal, as shown in [4]. In this work, the emphasis is on the performance evaluation of BO-NSMA, and thus we modelled the channel as AWGN. For each combination of SNR and modulation type, we generated 1000 instances with a size of 128 and 1024 for the simple set and the complex 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: Each evaluated AMC method is implemented using TensorFlow [37]. The fitness evaluation training is performed over 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. We opted for $k = 2$ for deterministic tournament selection, 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, we will justify why we design BO-NSMA as described in Section III by using the simple set of modulations. 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, we run seven experiments: (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. 4 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 accuracy of the best individual in the population (bottom). Fig. 5 (right) presents Pareto front approximation for the 10th generation. The Pareto optimal front is illustrated with the solid line in Fig. 5 (right), and the best-performing method should follow closely it. Fig. 5 (left) shows the Pareto set difference between two population generations which is important for GA convergence rate monitoring.

a) BO-NSMA performance over generations: Fig. 4 (top left and bottom) shows that BO-NSMA without FS, for both mutation rates, achieves the best average accuracy and maximum accuracy at the tenth generation. BO-NSMA with FS and high mutation rates has a better performance compared to BO-NSMA with low mutation rates, whereas it converges to the same maximum accuracy as the BO-NSMA without FS after the tenth generation. BO-NSMA without LS and BO-NSMA with constant crossover rate, $p_{cr} = 1$ have the worst average accuracy over generations. BO-NSMA with constant crossover rate, $p_{cr} = 1$ has 5% lower average accuracy and 2% lower maximum classification accuracy at the tenth generation than our proposed self-adaptive crossover rate (see Fig. 4). Regarding the average number of trainable parameters, each BO-

NSMA with low mutation rates converges to almost the same number, around $60k$, while BO-NSMA cases with high mutation rates end in the higher average number of trainable parameters, around $80k$. High mutation rates introduce more exploration, explaining the increase in trainable parameters after the fourth generation for BO-NSMA cases with high mutation rates.

b) Population accuracy and diversity: Fig. 5 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 we can still notice two optimal Pareto points. 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. 5 (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, we can state 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 good is our proposed search space, we explore the benefits of our proposed block design versus well-known ResNet blocks [5] 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 [5]. Thus, to demonstrate the impact of encoding, we run five experiments: (1) BO-NSMA with our designed encoding, (2) BO-NSMA with layer stacking, (3) BO-NSMA with ResNet block stacking, (4) NSGA-Net [11] with their pre-designed blocks, and (5) EvoCNN [9].

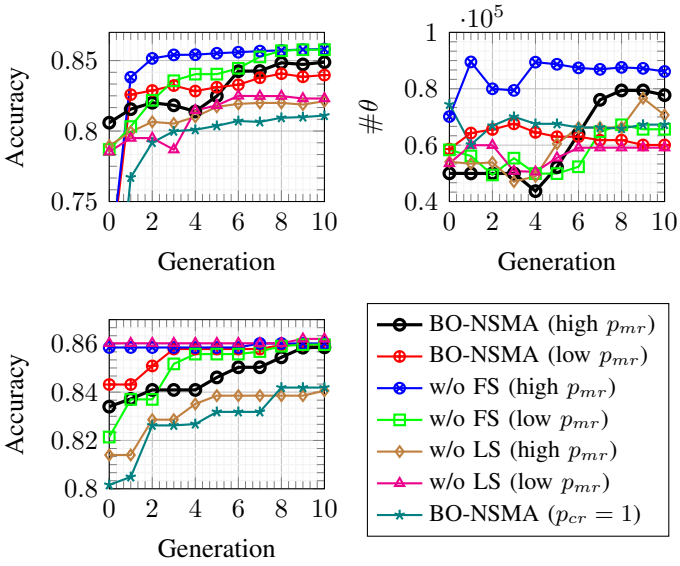


Figure 4: BO-NSMA components and their impact on: average accuracy (top left); average number of trainable parameters (top right); maximum accuracy (bottom) over generations.

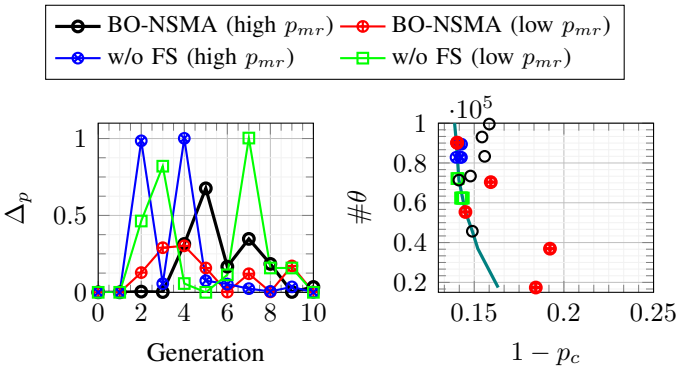


Figure 5: The averaged Hausdorff distance of two subsequent generations (left). Pareto front approximation for Generation 10 (right).

Each experiment uses the simple set of modulations. EvoCNN stacks the Conv, pooling and dense layers (the maximum number of layers is set to 15).

Fig. 6 shows that BO-NSMA with our 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-

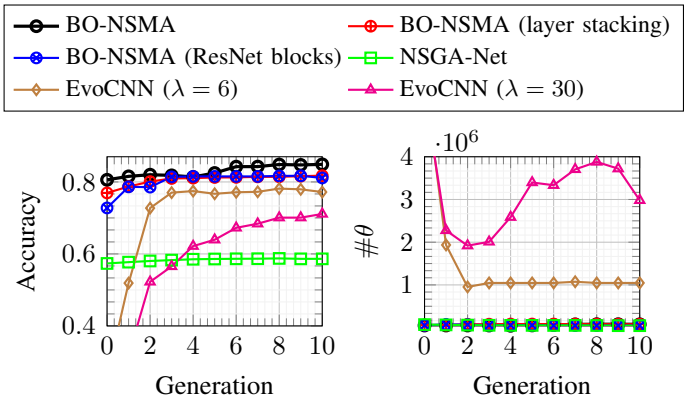


Figure 6: The search space and encoding impact on: average accuracy (top left) and average number of trainable parameters (top right) over generations.

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, we run EvoCNN for $\lambda = 30$. Fig. 6 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 SoA AMC: Finally, we compare the performance of the best individual found by BO-NSMA 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 simple and complex modulation datasets mentioned in Section IV-A2, respectively, whereas Fig. 7 presents accuracy across SNRs for the simple modulation dataset.

LSTM [1] is the best-performing SoA architecture for AMC evaluated on the simple modulation dataset, which achieves an average accuracy of 86%

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

Model	Simple Modulation Set		Complex Modulation Set	
	Acc.(%)	$\#\theta$	Acc. (%)	$\#\theta$
LSTM [1]	86.17	200,075	46.49	201,236
ResNet [2]	85.58	255,115	79.71	313,620
ResNeXt [4]	85.72	85,051	80.40	86,212
1D-CNN [2]	82.01	100,811	79.54	142,932
BO-NSMA	**86.87**	**71,399**	**82.71**	**79,560**
EvoCNN [9]	78.36	1,045,844	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

with over $200k$ trainable parameters. BO-NSMA finds the first genetically-optimized architecture, which achieved slightly higher average accuracy (an improvement of 0.7%) while reducing the number of trainable parameters to $71k$ (a 2.80-fold reduction in the number of trainable parameters). LSTM with default training parameters given in [1] fails to converge for the complex modulation dataset. Next in terms of achieved performance are ResNet and ResNeXt architectures. For the simple modulation dataset, BO-NSMA achieves 1.29% and 1.15% accuracy gain over ResNet and ResNeXt, respectively, while keeping the number of trainable parameters over three times lower compared to ResNet. For the complex modulation dataset, BO-NSMA achieves 3.0% and 2.31% accuracy gain over ResNet and ResNeXt, respectively, while keeping the number of trainable parameters over $3.9x$ lower compared to ResNet. While exploring BO-NSMA's performance across SNR (Fig. 7) for the simple modulation dataset, we 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 [2] for the simple dataset and complex 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 simple dataset and complex dataset, respectively. BO-NSMA with layer stacking achieves to find a solution within

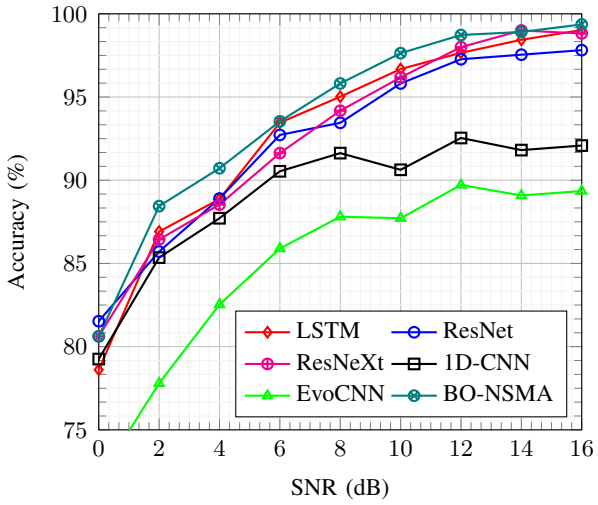


Figure 7: Classification accuracy across SNR (simple modulation dataset).

10 generations and population size of 6 that has 8.5% and 18.2% accuracy gain at $14.6x$ and $78.7x$ lower number of trainable parameters than its counterpart EvoCNN for the simple dataset and complex dataset, respectively.

V. 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 transfer properly 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 we showed for NSGA-Net. Thus, in this paper, we proposed BO-NSMA, a novel bi-objective memetic algorithm for joint architecture and network complexity optimization for DNN-based modulation recognition applications. Following extensive experimentation, we show 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, we demonstrated that BO-NSMA does not have a premature convergence problem for a low population size, as is the case with its counterpart EvoCNN. The reported results show that BO-NSMA is a promising tool for NAS, but its computational cost has not been optimized. Future research would benefit from focusing on the dynamic adaptivity of BO-NSMA components to reduce its computational cost by employing smart policies for LS use and a more refined metric for network complexity, such as inference time.

References

[1] 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.

[2] 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.

[3] 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.

[4] E. Perenda, S. Rajendran, *et al.*, "Learning the unknown: Improving modulation classification performance in unseen scenarios @ONLINE." IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, url="http://www.cs.albany.edu/ mariya/lab/papers/m67919-perenda.pdf", 2021.

[5] 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.

[6] 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.

[7] B. Baker, O. Gupta, *et al.*, "Designing Neural Network Architectures using Reinforcement Learning," *ArXiv*, vol. abs/1611.02167, 2017.

[8] 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.

[9] 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.

[10] L. Xie and A. Yuille, "Genetic CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397, 2017.

[11] 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.

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

[13] 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.

[14] W. Shi, D. Liu, *et al.*, "Particle Swarm Optimization-Based Deep Neural Network for Digital Modulation Recognition," *IEEE Access*, vol. 7, pp. 104591–104600, 2019.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] R. Hamdi, M. Njah, and M. Chtourou, "Multilayer perceptron training using an evolutionary algorithm," *Int. J. Model. Identif. Control.*, vol. 5, pp. 305–312, 2008.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

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

[36] 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.

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