A Genetic Algorithm Approach to Automate Architecture Design for Acoustic Scene Classification

Noha W. Hasan, Ali S. Saudi, Mahmoud I. Khalil, and Hazem M. Abbas, *Senior Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) have been widely used with remarkable success in the acoustic scene classification (ASC) task. However, the performance of these CNNs highly relies on their architectures, requiring a lot of effort and expertise to design CNNs suitable for the investigated problem. In this work, we propose an efficient genetic algorithm (GA) that aims to find optimized CNN architectures for the ASC task. The proposed algorithm uses frequency-dimension splitting of the input spectrograms in order to explore the architecture search space in sub-CNN models in addition to classical single-path CNNs. Specifically, this algorithm aims to find the best number of sub-CNNs in addition to their architectures to better capture the distinct features of the input spectrograms. The proposed GA is specifically designed for sound classification to suit the ASC task than many other GAs that optimize conventional single-path CNN architectures. Experimental results on three benchmark datasets demonstrate the effectiveness of the proposed method. Specifically, the proposed algorithm has achieved around 17.8%, 16%, and 17.2%, relative improvement in accuracy with respect to the baseline systems on the development datasets of DCASE2018-Task1A, DCASE2019-Task1A, and DCASE2020-Task1A, respectively.**

***Index Terms*—Acoustic scene classification (ASC), convolutional neural networks (CNNs), evolutionary deep learning, genetic algorithms (GAs), machine listening, neural-network architecture optimization.**

I. Introduction

ACOUSTIC scene classification (ASC) refers to associating a semantic label to an audio sample that identifies the environment in which it has been produced [1]. The term $acoustic\ scene$ refers to the aggregation of sounds from various sources in a specific environment, typically from a real scenario, which combine to form a mixture. For example, the sound scene of a street can contain the sounds of cars passing by, people talking, footsteps, etc. The sound scene in a home might contain music from a TV, a washing machine humming, and children playing. An ASC system may classify a sound sample as one of a set of predefined categories such as home, airport, or a public street. The accuracy that the system can achieve is affected by many factors, such as the diversity of each class, the similarity between classes, the quality and amount of training data, the actual computational methods used, in addition to external factors such as interfering noises. The technical report proposed by Sawhney and Maes from the Massachusetts Institute of Technology (MIT) Media Lab in 1997 [2] is the first known paper in the literature that addresses the ASC problem. In this report, the authors recorded a dataset from a set of environmental classes and tried to map the relation between the acoustic features and the associated sound categories using a recurrent neural network (RNN) model. There has been a growing interest in the ASC task ever since, and the research approaches to the task have further developed over the past few years. The early approaches to the ASC task focused on feature engineering [3] to design the proper inputs to the classifier. In this context, an extensive variety of input representations such as mel-frequency cepstral coefficients (MFCCs) [4]; constant-Q transform (CQT) [5]; local binary patterns (LBPs) [6]; and histograms of oriented gradients (HOG) [7] were used. With the years and the emergence of convolutional networks in the deep learning field, CNNs have become a preferred option for the design of ASC systems, usually fed with log-mel spectrograms [8]. These networks have shown promising results [9], especially when they are trained on large datasets. This is why data augmentation techniques, such as mixup [10] and temporal cropping [11], are usually applied.

Recently, splitting of input feature maps along the frequency dimension into high and low frequencies for the ASC task has been proposed by McDonnell et al. in [11]. The authors of this work designed a CNN architecture with two identical parallel paths for processing high and low frequency bands separately. This means that each input feature map was first split into two sub-feature maps along the frequency dimension. Then, each split would be independently processed by a residual network with 17 convolutional layers. The idea behind such a split was that the features to be learned for high frequencies are likely to be different from those for low frequencies. The same approach was used by the second-ranking team [12] of the DCASE2020-Task1A challenge [13] who also used two sub-feature maps along the frequency-dimension in some of their proposed systems. Additionally, the top-ranking team [14] of the same challenge used three frequency-dimension splits in their proposed architecture. A similar approach was presented by Phaye et al. [15] who proposed the architecture of SubSpectralNet, which essentially creates horizontal slices of the log-mel spectrogram using predefined values for sub-spectrogram size and mel-bin hop-size. Then,

N. W. Hasan, M. I. Khalil, and H. M. Abbas are with the Department of Computer and Systems, Faculty of Engineering, Ain Shams University, Cairo, Egypt e-mail: (noha.wahdan@eng.asu.edu.eg; mahmoud.khalil@eng.asu.edu.eg; hazem.abbas@eng.asu.edu.eg).

A. S. Saudi is with the Faculty of Computers and Artificial Intelligence, Benha University, Qalubiya Governorate, Egypt e-mail: (ali.saudi@fci.bu.edu.eg).

Manuscript received 2022; revised 2022.

separate CNNs are trained on these sub-spectrograms. The classification output is finally obtained using a set of sub-classifiers connected to a global one. The best accuracy they achieved on the development dataset of DCASE2018-Task1A [16] was 74.08%, by using 30 sub-spectrogram size and 10 mel-bin hop-size.

On the other hand, the performance of CNNs highly relies upon their architectures. Thus, finding a suitable network architecture and corresponding hyper-parameters for a given problem remains a challenge. Furthermore, even though many works have studied the automated approach for designing neural network architectures for image classification [17], [18], [19], the approach is still new for ASC. Usually, network architectures of ASC systems and their corresponding hyper-parameters are manually optimized, with only limited research on the automated architecture design approach for ASC. Furthermore, the search space of current methods is only limited to classical single-path networks that are usually designed for image classification.

In this work, we propose an automatic CNN architecture design method using the genetic approach to effectively address the ASC problem. The novelty of the proposed algorithm lies in further utilizing the idea of frequency-dimension splitting by adding the number of splits as a parameter to be improved during the evolutionary process. The main contribution of this work is that this is the first genetic algorithm (GA) that explores the search space in sub-CNN models in addition to classical single-path CNNs. So, this algorithm could be used to investigate the best number of frequency-dimension splits to better capture the distinct features from different frequency bands for a given ASC dataset. This work extends our research in [20] where we summarized the proposed algorithm. This work presents the proposed algorithm in detail in addition to evaluating it on three benchmark ASC datasets. The conducted experiments show the effectiveness of the proposed method in discovering enhanced CNN architectures for the ASC problem within an acceptable search time.

The remainder of this article is organized as follows. First, the background and related work are presented in Section II. Then, the details of the proposed algorithm are documented in Section III. Next, the experimental setup, experimental results, as well as the analysis, are presented in Section IV. Finally, the conclusion and future works are summarized in Section V.

II. Related Work

Evolutionary algorithms (EAs) [21] refer to a set of population-based stochastic-optimization algorithms inspired by the biological evolution. EAs include genetic programming [22], evolutionary strategy [23], evolutionary programming [24], GAs [25], among others. Typically, EAs are initialized with a random population of potential solutions to a particular problem, commonly called individuals. Each individual represents a location in the search space for the optimal solution. The population is then evolved by a number of genetic operators, over a set of generations, to produce better solutions to the investigated problem. The evolutionary process is conducted by using genetic operators. Mainly, crossover and mutation are the key genetic operators used in most EA paradigms. The evolutionary process is repeated until the maximum number of generations has been reached. The population in the last generation then contains the best evolved solutions to the problem and may also incorporate the global optimum solution [26]. GAs, introduced by Holland et al. [27], are the most popular class of EAs due to their theoretical evidence [28] and their capability of achieving promising results in solving different optimization problems. Multiple forms of gene encoding strategies have been proposed in the literature, among which the bit-string fixed-length representation was often employed. However, in this work, we have utilized the variable-length encoding strategy proposed in [29] which has shown to be beneficial for evolving CNN architectures of unrestricted depth.

Limited publications have studied the automated architecture design approach for the ASC field. Specifically, Roletscheck et al. in [30] proposed an evolutionary approach, namely DeepSAGA, that followed the guidelines in [31] to automatically design CNN architectures for the development dataset of DCASE2018-Task1A [16]. With the best CNN achieved an average accuracy of 72.8%, the search procedure took 120 hours using 15 clients, each equipped with an NVIDIA GTX 1060 graphic card. Li et al. in [32] utilized the NSGA-II evolutionary algorithm [33] to search for promising architectures for ASC. Their searched network achieved a 90.3% F1-score on the DCASE2018-Task5 dataset [34]. Wu et al. in [35] proposed a neural architecture search (NAS) approach for the ASC task. The method was evaluated on the development dataset of DCASE2020-Task1B [36], with the best-performing model achieved an accuracy of 95.8%. In [37], we have developed a NAS approach for the ASC problem based on differentiable architecture search (DARTS) [38]. The work investigated optimizing DARTS by using residual blocks and squeeze-excitation blocks as possible candidate operations. This approach has achieved an average classification accuracy of 64.57% on the development dataset of DCASE2020-Task1A [39] with 7.5 GPU days needed for the search process. However, all of the aforementioned approaches are limited to optimizing classical single-path CNNs similar to those designed for image classification.

III. Proposed Algorithm

In this section, the framework of the proposed algorithm and its main components are discussed in detail.

A. Algorithm Overview

The overview for the proposed algorithm is presented in algorithm 1. In brief, the algorithm is first supplied with a set of predefined building blocks for CNNs, the population size, the maximum number of generations, and the ASC dataset. Then, the population is initialized based on the gene encoding strategy in section III-C. Subsequently, the evolution begins to take effect until the maximum number of generations has been reached.

During evolution, the fitness of each individual is evaluated on the given dataset. Then, parent individuals are selected

based on their fitness scores to generate an offspring population using the crossover and mutation operations. Then, a small set of random individuals is also generated based on a predefined probability r. This operation could be considered as another form of the mutation operation that generates individuals whose architectures are independent on the initial population. The next generation is then selected from the current population in addition to the offspring and the randomly generated individuals by environmental selection. Finally, the best individual is selected from the last generation and decoded into the corresponding CNN for the final training and evaluation.

Algorithm 1: Algorithm Overview

Input : The population size N, the maximum number of generations G, the percentage of random population r, and the ASC dataset.
Output: The best discovered CNN architecture.

$P_0 \leftarrow$ Initialize a population of N individuals using the proposed gene encoding strategy;
$i \leftarrow 0$;
while $i < G$ **do**
  Evaluate the fitness of each individual in P_i using the proposed fitness evaluation function;
  $Q_i \leftarrow$ Generate offspring from selected parent individuals in P_i using the proposed crossover and mutation operators;
  $R_i \leftarrow$ Generate a set of $(r \times N)$ random individuals using the proposed gene encoding strategy;
  Evaluate the fitness of individuals in $(Q_i \cup R_i)$;
  $P_{i+1} \leftarrow$ Environmental selection of N individuals from $(P_i \cup Q_i \cup R_i)$;
  $i \leftarrow i + 1$;
end

Typically, the proposed GA constructs CNN architectures for a given ASC dataset using convolutional blocks in addition to max and average pooling layers. Fig. 1 shows the architecture of the pre-activation convolutional block utilized in this work, which consists of three layers; batch-normalization, ReLU, and convolution, respectively.

In general, the parameters of a convolutional layer are the number of filters, the stride size, the kernel size, and the convolutional operation type. However, we use fixed values for the kernel sizes, stride sizes, and padding type. Particularly, all convolutional layers in this work have a kernel size of 3×3, a stride of 1×1, $same$ padding, and initialized with normal distribution [40]. To this end, the parameter encoded for a convolutional block is only the number of the filters for the corresponding convolutional layer. In addition, pooling layers in this work are always used with $same$ padding, a kernel size of 3×3, as well as a stride of two along the temporal dimension and a stride of one along the frequency dimension as suggested by [11]. To this end, the only parameter encoded for a pooling layer is the pooling type, which is set as either average or max pooling.

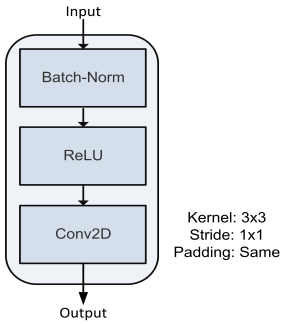

Fig. 1. Pre-activation convolutional block

Furthermore, the architecture designs of hand-crafted state-of-the-art CNNs, such as AlexNet[41], VGG16[42], Inception [43], and ResNet[44], have demonstrated that increasing the number of filters of convolutional layers with the depth of the network better captures input features and improves the network classification accuracy. We benefit from those architectures in our proposed GA so that it always yields CNN architectures with the number of filters of each convolutional block, either the same as or higher than the preceding one, with equal probabilities for both options. The aforementioned condition is always satisfied at the initialization stage. Furthermore, the crossover and mutation operations are designed accordingly to make sure that this condition is not violated.

B. System architecture

In this section, the general structure of the individual CNN architectures constructed with the proposed GA is discussed in detail. All CNN architectures constructed with the proposed algorithm have the general structure shown in Fig. 2. The input to the system is a spectrogram of shape $(T \times F \times C)$, where T is the number of features along the temporal dimension, F is the number of frequency bins, and C is the number of input channels. The proposed algorithm produces architectures that are fully convolutional in nature, which means that they can process spectrograms of arbitrary durations. So, the proposed GA can be used with any sound classification dataset to find good-performing CNN architectures for the classification of the given input sound spectrograms.

Specifically, an input spectrogram of shape $(T \times F \times C)$ is first split into N number of splits along the frequency dimension, and thus creating N spectrogram-splits of shape $(T \times (F/N) \times C)$. N is the arbitrary number of frequency-dimension splits specified for each individual during the evolution. Each split is then processed by a sub-CNN architecture consisting of convolutional blocks and pooling layers. All sub-CNNs are identical for a given individual, and their architecture is optimized for each individual during the evolutionary process. All splits are then concatenated and used as an input to the final part of the CNN. This part consists of two convolutional blocks, a batch normalization layer [45], a global average pooling layer, and a dropout layer [46], respectively. The number of filters of the second convolutional block is always equal to the number of dataset classes. The dropout

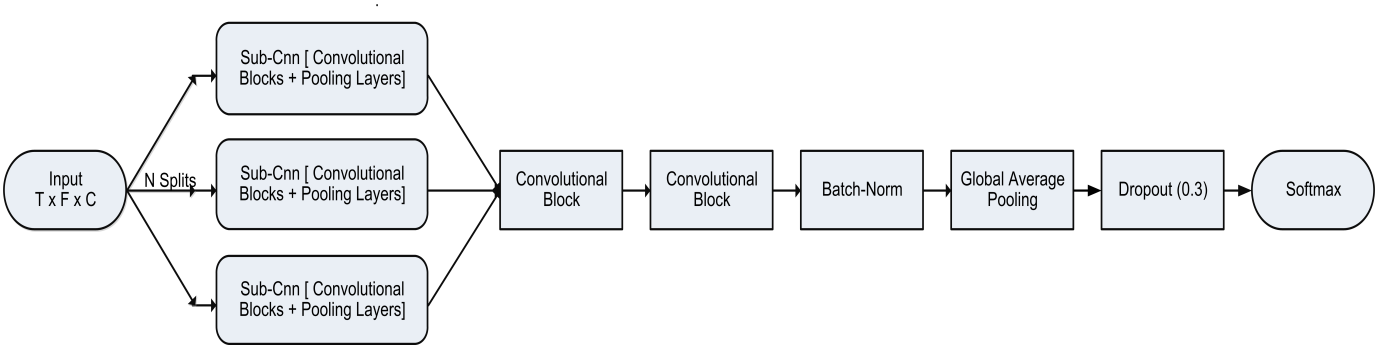

Fig. 2. System Architecture

layer is used as a regularization technique to reduce overfitting. Prediction results are then obtained by a softmax layer.

Particularly, the aim of the search process is to obtain the best number of frequency-dimension splits along with the best sub-CNN architecture. As a reference, Table I provides the list of parameters considered by the proposed algorithm in addition to their settings used in our experiments. The table shows that using the proper settings for the minimum and maximum number of frequency-dimension splits, the algorithm can effectively explore the search space of conventional single-path CNNs, splitted sub-CNNs, or both of them at the same time as we proposed in our experiments. However, it is an advisable practice to set the lowest number of sub-CNNs to 1, in order to involve classical single-path networks in the explored space as they could also be suitable for the task. To summarize, the following parameters are set and optimized for each individual during the evolutionary process.

1) The number of frequency-dimension splits N, which is also the number of the sub-CNNs.
2) The number of convolutional blocks in the associated sub-CNN architecture.
3) The number of pooling layers in the associated sub-CNN architecture.
4) The number of filters for each convolutional block in the sub-CNN architecture.
5) The type of each pooling layer in the sub-CNN architecture, which is either average or max pooling.
6) The exact sequence of convolutional blocks and pooling layers that defines the architecture of the sub-CNNs for each individual.

C. Encoding strategy

In this work, we only encode the variable information that is not the same for all individuals, which involves the number of frequency-dimension splits and the architecture of the corresponding sub-CNNs. Given that all sub-CNNs of a given individual would have the same architecture, only one architecture needs to be encoded for each individual. In particular, each individual CNN is encoded using the number of frequency-dimension splits followed by the architecture of the corresponding sub-CNNs as follows.

1) For convolutional blocks, only the number of filters for the convolutional layer is encoded.
2) For pooling layers, only the type of the pooling layer is encoded with a number between $[0-1]$, where a number ≤ 0.5 represents a max-pooling layer, and a number $>$ 0.5 represents an average-pooling layer.

An example for encoding an individual with five sub-CNNs, where each sub-CNN has eight convolutional blocks, a max-pooling layer, and an average pooling layer is $5-32-64-0.2-64-256-0.8-512-256-256-512$. The encoding string of each individual is then hashed using a 224-bit SHA-2 hash function [47] to be used as an identifier for its structure. Each individual is then stored with its encoding string followed by its hash value. The hash values are used as keys for storing and fetching individuals' fitness values from a local cache as we discuss in Section III-E

D. Population Initialization

We summarize the initialization procedure in algorithm 2. Briefly, N individuals are initialized and stored into the initial population P_0. The initialization procedure of each individual involves the generation of three random values: the number of frequency-dimension splits s, the number of convolutional blocks b, and the number of pooling layers p. Each of them is randomly generated from a predefined range of values. The number of filters for convolutional blocks is initialized to the minimum value from the predefined set. Each individual is represented by both the number of frequency-dimension splits s, and a linked list L, containing $b+p$ nodes that represent the architecture of the associated sub-CNNs. It should be noted that that the nodes of list L are randomly shuffled after the initialization (see line 13).

For each convolutional block in L, a number r is randomly generated from $[0-1]$. If $r > 0.5$, the number of filters is increased to the next higher value from the given set. However, if the maximum value is already hit, its value does not change. Then, the current value for the number of filters is added as the node's number of filters (see lines 17-21). For each pooling layer in L, a number r is randomly generated from $[0-1]$. If $r \leq 0.5$, the pooling type is set to max, otherwise it is set to average (see lines 23-28).

E. Fitness Evaluation

For each individual in a given population P_i, the fitness is then obtained either from the cache, if it was calculated

Algorithm 2: Population initialization

Input : The population size N, the number of splits ($S_{min} : S_{max}$), the number of convolutional blocks ($B_{min} : B_{max}$), the number of pooling layers ($P_{min} : P_{max}$), and the list of available values for the number of filters F.
Output: The initialized population P_0.

$P_0 \leftarrow \phi$;
for $i \leftarrow 1$ **to** N **do**
  $s \leftarrow$ Randomly generate the number of frequency-dimension splits ($S_{min} \leq s \leq S_{max}$);
  $b \leftarrow$ Randomly generate the number of convolutional blocks ($B_{min} \leq b \leq B_{max}$);
  $p \leftarrow$ Randomly generate the number of pooling layers ($P_{min} \leq p \leq P_{max}$);
  $L \leftarrow$ Create an empty list with $b + p$ nodes;
  for $j \leftarrow 0$ **to** b **do**
    $L[node\_j].type \leftarrow 1$
  end
  for $k \leftarrow b$ **to** L **do**
    $L[node\_k].type \leftarrow 2$
  end
  Randomly shuffle list L;
  $Num\_Filters \leftarrow$ Set to F_{min};
  foreach *node in list* L **do**
    if $node.type \leftarrow 1$ **then**
      $r \leftarrow$ Uniformly generate a random number from [0-1];
      if $r > 0.5$ *and* $Num\_Filters < F_{max}$ **then**
        $Num\_Filters \leftarrow$ Set to the next higher value in F;
      end
      $node.num\_filters \leftarrow Num\_Filters$;
    else
      $r \leftarrow$ Uniformly generate a random number from [0-1];
      if $r \leq 0.5$ **then**
        $node.pool\_type \leftarrow max$;
      else
        $node.pool\_type \leftarrow average$;
      end
    end
  end
  $I \leftarrow S \cup L$;
  $P_0 \leftarrow P_0 \cup I$;
end
Return P_0.

in an earlier generation, or it is calculated by decoding the architecture of this particular individual and placing it asynchronously on an available GPU for fitness evaluation. The fitness evaluation process involves training the given individual for a predefined number of epochs and then evaluating it on the given ASC dataset. The evaluation accuracy is used as the fitness score for this particular individual. The hash key of each individual is then stored in the cache along with its fitness score. The stored values can be used for the individuals surviving into the following generations. Additionally, if the same architecture is re-generated with the crossover and mutation operations in another generation, the stored accuracy could be used directly. Algorithm 3 summarizes the fitness evaluation process.

In this work, we aim to make full use of the available computational resources by combining the asynchronous computation scheme with an adaptable batch size scheme. The batch size used in the training process is usually a fixed parameter in most GA paradigms [48]. However, when a variable length encoding strategy is used, it may not be computationally efficient to use the same value for the batch size for all individuals. For example, a given fixed value for the batch size may cause an out-of-memory (OOM) exception with a large-sized CNN architecture. On the other hand, the same batch size value may not make full use of the available GPU memory when a relatively small-sized CNN architecture is being trained. The key idea of the adaptable batch size is to initially provide the algorithm with an acceptable range for the batch size value. The training process of each individual starts with the maximum value for the batch size. This value then gets halved every time an OOM exception occurs, until the training process can be initiated without a problem, or the minimum value is reached. If the training process fails, the accuracy of such individual is then set to 0.

Algorithm 3: Fitness evaluation

Input : The individuals of population P_i, the ASC dataset, the number of training epochs, the batch size ($B_{min} : B_{max}$).
Output: The fitness values for all individuals in P_i.

1 **foreach** *individual* Ind *in* P_i **do**
2 **if** $Ind.key$ *is in* $Cache$ **then**
3 $Ind.accuracy \leftarrow Cache.key.accuracy$
4 **else**
5 **if** *There is an available GPU* **then**
6 Decode the architecture of Ind and place it asynchronously on the available GPU;
7 $Batch\_Size \leftarrow B_{max}$;
8 Train Ind on the given ASC dataset to obtain $Ind.accuracy$;
9 **if** *OOM exception occurs* **then**
10 $Batch\_Size \leftarrow Batch\_Size \div 2$;
11 **if** $Batch\_Size \geq B_{min}$ **then**
12 Go to line 8
13 **else**
14 $Ind.accuracy \leftarrow 0$;
15 Continue;
16 **end**
17 **end**
18 **else**
19 Wait for a GPU to become available
20 **end**
21 **end**
22 **end**

F. Offspring Generation

Offspring generation consists of the crossover operation followed by the mutation operation. The details of the crossover operation in the proposed algorithm are shown in algorithm 4. Initially, two parent individuals are selected with binary tournament selection [49]. Then, taking into consideration that each of the two selected individuals has an associated sub-CNN architecture, the crossover operation takes place between the two architectures to generate two new sub-CNN architectures for the offspring. It is emphasized that the proposed crossover operation is performed between the associated architectures regardless of the corresponding number of frequency-dimension splits of each individual. The generated offspring always have the same number of frequency-dimension splits as their parents. The condition in line 5 of algorithm 4 is set to make sure that the number of filters of consecutive convolutional blocks in the sub-CNNs of the generated offspring is never decreasing with network depth or alternating randomly. Fig. 3 shows an example of the crossover operation that involves the sub-CNNs of two selected parents, P_1 and P_2. Supposing that the third position is randomly selected from the two sub-CNN architectures for the crossover operation, the sub-CNNs of the generated offspring individuals would be as shown in the same figure, denoted by O_1 and O_2.

The mutation operation typically maintains the diversity of the population by introducing another level of randomness to further explore the search space for more promising performance. For each individual O in the offspring population, the mutation operation is performed after the crossover operation, as illustrated in algorithm 5. Specifically, a random number r is initially generated from $[0-1]$. If $r <$ the predefined probability for the mutation operation γ, then one of the available mutation types is selected based on the corresponding probabilities. The available mutation types in the proposed algorithm are as follows.

1) Adding a unit at a randomly selected position in the associated sub-CNN architecture. A random number is generated to define its type as either a pooling layer or a convolutional block. Then, if the convolutional block type is selected, another random number is generated from [0-1] to define the number of filters for its convolutional layer. If $r < 0.5$, the number of filters would be the same as the preceding convolutional block. Otherwise, the number of filters would be the same as the following convolutional block. For pooling type, a random number is used to select its type as either average or max pooling.
2) Removing the unit at a randomly chosen position in the associated sub-CNN architecture.
3) Changing the unit type at a randomly chosen position in the associated sub-CNN architecture. In other words, a pooling layer would be changed into a convolutional block and vice versa. The number of filters for the convolutional block would be defined as in 1.
4) Modifying the pooling type of a randomly chosen pooling layer in the associated sub-CNN architecture. This means that an average pooling layer would be changed into max pooling, and vice versa.
5) Modifying the number of splits for the selected individual to a randomly chosen value from the predefined range.

Algorithm 4: Crossover operation

Input : Two parent individuals, P_1 and P_2, and the crossover probability β.
Output: Two offspring individuals O_1 and O_2

1 $r \leftarrow$ Uniformly generate a random number from [0-1];
2 **if** $r < \beta$ **then**
3 Randomly choose a position from the sub-CNN architecture of P_1;
4 Randomly choose a position from the sub-CNN architecture of P_2;
5 **if** *(the number of filters in the last convolutional layer in the first part of the sub-CNN architecture of $P_1 \leq$ the number of filters in the first convolutional layer in the second part of the sub-CNN architecture of P_2) AND (the number of filters in the last convolutional layer in the first part of the sub-CNN architecture of $P_2 \leq$ the number of filters in the first convolutional layer in the second part of the sub-CNN architecture of P_1)* **then**
6 continue;
7 **else**
8 Go to line 3;
9 **end**
10 $O_1 \leftarrow$ Combine the first part of the sub-CNN architecture of P_1 and the second part of the sub-CNN architecture of P_2 to construct a sub-CNN architecture for the new offspring individual;
11 $O_2 \leftarrow$ Combine the first part of the sub-CNN architecture of P_2 and the second part of the sub-CNN architecture of P_1 to construct a sub-CNN architecture for the new offspring individual;
12 $O_1.Num\_Splits \leftarrow P_1.Num\_Splits$;
13 $O_2.Num\_Splits \leftarrow P_2.Num\_Splits$;
14 **else**
15 $O_1 \leftarrow P_1$;
16 $O_2 \leftarrow P_2$;
17 **end**
18 Return O_1 and O_2.

For better understanding the mutation operation, an example of the $change\ unit\ type$ mutation is shown in Fig. 4. In this example, the fifth position, which contains an average pooling unit, is randomly selected from the sub-CNN architecture of individual O. So, the selected unit is replaced by a convolutional block. The generated sub-CNN of the mutated individual is denoted by Q. The fitness of all newly generated individuals is then calculated as illustrated in section III-E.

Algorithm 5: Mutation operation

Input : The offspring individual O, the mutation probability γ, and the probabilities of selecting different mutation operations P_m.
Output: The mutated offspring Q

$r \leftarrow$ Uniformly generate a random number from [0-1];
if $r < \gamma$ **then**
  $mutation\_type \leftarrow$ Randomly select one from (Add - Remove - Change_Type - Modify_Pooling - Modify_Splits) based on the probabilities in P_m;
  if $mutation\_type\ is\ Add$ **then**
    $p \leftarrow$ Randomly choose a position from the sub-CNN architecture of O;
    $m \leftarrow$ Uniformly generate a random number from [0-1];
    if $m < 0.5$ **then**
      $Q \leftarrow$ Add a convolutional block at position $p+1$;
    else
      $Q \leftarrow$ Randomly select a max or average pooling layer to be added at position $p+1$;
    end
  else if $mutation\_type\ is\ Remove$ **then**
    $p \leftarrow$ Randomly choose a position from the sub-CNN architecture of O;
    $Q \leftarrow$ Remove the unit at p ;
  else if $mutation\_type\ is\ Change\_Type$ **then**
    $p \leftarrow$ Randomly choose a position from the sub-CNN architecture of O;
    $Q \leftarrow$ Exchange the unit type at p, i.e., a pooling layer would be changed into a convolutional block and vice versa;
  else if $mutation\_type\ is\ Modify\_Pooling$ **then**
    $p \leftarrow$ Randomly choose a pooling layer from the sub-CNN architecture of O;
    $Q \leftarrow$ Modify the pooling type of the chosen unit at p, i.e., an average pooling layer would be changed into max pooling, and vice versa;
  else
    $Q \leftarrow$ Modify the number of splits for this individual to a randomly chosen value from the predefined range;
  end
  Return Q
else
  $Q \leftarrow O$;
end

G. Environmental Selection

At this point, we further avoid the local optima in the search space by generating a number of $(r \times N)$ random individuals, where N is the number of individuals in each generation, and r is the selected percentage for generating random population at the end of each generation. The fitness scores of the generated individuals are then calculated to be used in the environmental selection process. Binary tournament selection [49] is then performed repeatedly to choose N individuals from the current population $(P_i \cup Q_i \cup R_i)$, where P_i is the population of the current generation, Q_i is the offspring population generated with crossover and mutation, and R_i is the generated $(r \times N)$ random population. Then, these selected individuals are placed into the next population (denoted by P_{i+1}) to serve as the parent individuals for the next generation. After that, the best individual is selected to check whether it has been placed into P_{i+1}. If not, it will be added by replacing the worst individual in P_{i+1}.

The idea behind generating a small percentage of random population is to provide further diversity in each generation by reducing dependency on the solutions of the initial population. In general, a GA begins with an initial random population and evolves to better solutions by maintaining the best solutions in each generation. Typically, the crossover operation is performed to explore and exploit the area "between" each two parent solutions and the classical mutation operation explores the area "around" the given population, usually by changing some properties of selected individuals. However, even though the GA benefits from the different types of the mutation operation, the resulting CNNs may still depend on the architectures of the initial population. As a result, the best performance may not be achieved due to trapping into local optima, and convergence to the global optimum solution within a limited number of generations cannot be guaranteed. Using this approach, the GA will keep exploring new areas in the search space with each generation, and then the crossover and mutation operations further investigate these areas in following generations. Consequently, the probability of finding better solutions increases with further generations. The randomly generated individuals must be evaluated prior to performing the environmental selection operation. Thus, only efficient candidate solutions are added to the next generation. However, generating a large number of random individuals would convert the GA to a primitive random search. So, similar to the mutation probability, the value of r should be relatively small.

IV. Experimental Setup and Results

In order to evaluate the effectiveness of the proposed algorithm, a set of experiments have been conducted on three ASC datasets. In this section, the used datasets are first introduced, followed by the audio representation selected to feed the network, and the parameter settings. Then, the results of the conducted experiments are discussed in comparison with state-of-the-art peer competitors.

A. Benchmark Datasets

In this work, we have employed three datasets to evaluate the effectiveness of the proposed algorithm. Each dataset consists of ten scene classes, including airport, tram, bus, metro, shopping mall, metro station, pedestrian street, public square, street traffic, and park. Moreover, each dataset is provided with an official baseline system that represents a

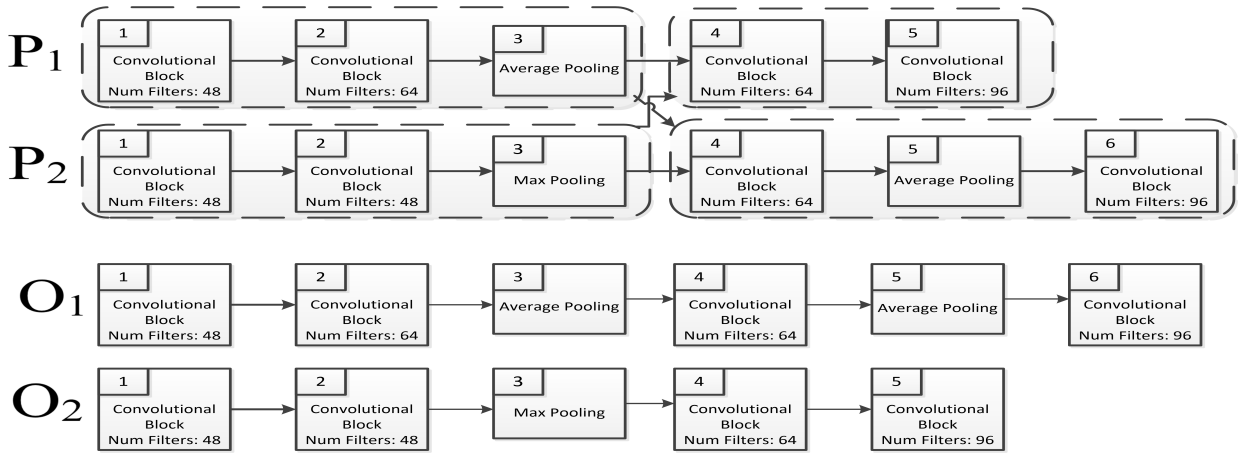

Fig. 3. P_1 and P_2 are the sub-CNNs of two selected parent individuals for the crossover operation. The corresponding generated sub-CNNs of offspring individuals are denoted by O_1 and O_2, respectively.

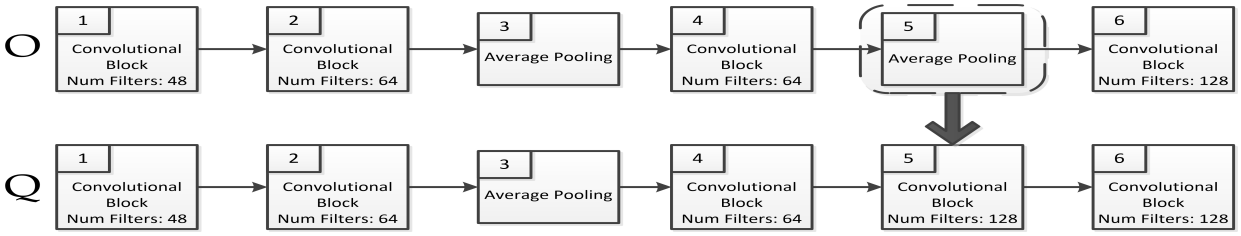

Fig. 4. Example of the *change unit type* mutation operation. The sub-CNN of the selected individual is denoted by O. Assuming that the fifth position is randomly selected for the mutation operation, the generated sub-CNN for the mutated individual is denoted by Q.

state-of-the-art approach to the classification task. Details of these datasets are listed as follows.

- *Development dataset of TAU Urban Acoustic Scenes 2020 Mobile [39]*. This is the official dataset of the DCASE2020-Task1A challenge. The set contains data from ten cities in ten acoustic scenes using nine devices. The dataset is provided with a training/test split. Specifically, the training split comprises 13965 segments and the test split comprises 2970 segments, with some devices appearing only in the test subset. Each segment is a 10-second audio clip. Audio samples are provided in a single-channel 44.1 kHz 24-bit format.
- *Development dataset of TAU Urban Acoustic Scenes 2019 [50]*. This is the official dataset of the DCASE2019-Task1A challenge. The dataset consists of 40 hours of recordings from ten cities in the same ten acoustic scenes as the previous dataset. The dataset is also partitioned into an official cross-validation setup for evaluation, with 9185 segments in the training set, and 4185 in the test set. All audio samples have a fixed length of ten seconds. The dataset was recorded with a single device using 48 kHz sampling rate and 24-bit resolution. In order to unify spectrogram shapes from all datasets, the audio samples of this dataset were re-sampled using a sampling rate of 44.1 kHz and averaged into a single channel.
- *Development dataset of TUT Urban Acoustic Scenes 2018 [16]*. This is the official dataset of the DCASE2018-Task1A challenge. The dataset consists of recordings from six cities in the same ten acoustic scenes as the previous datasets. The set is composed of 8640 10-second audio segments, with 6122 segments comprising the training subset and 2518 segments in the test subset. The dataset was recorded with a single device using 48 kHz sampling rate and 24-bit resolution. The audio samples of this dataset were also re-sampled using a sampling rate of 44.1 kHz and averaged into a single channel.

In all of our experiments, only the training set of each dataset is utilized in the search process. Each training set was further partitioned with the official cross-validation setup into a training subset and a validation subset, which were employed in the fitness evaluation procedure. Specifically, the fitness value of each individual represents the maximum achieved accuracy on the validation subset. The test set is only used to compare the classification accuracies with peer competitors. It should be noted that although we have unified input spectrogram shapes in our experiments, it is unnecessary to do so. In fact, the proposed algorithm produces fully-convolutional network architectures, which means that they can process spectrograms of arbitrary durations. However, the reason we unified all input shapes is that we aim to compare the classification accuracy of each produced architecture on all three datasets.

B. Feature Extraction

The initial data files from all datasets after pre-processing are single-channel audio clips. Each file is a 10-second audio

clip with a sampling rate of 44.1 kHz. The feature extraction process involved calculating the mel coefficients from the audio files using APIs from LibROSA [51], a python package for audio processing. In our experiments, we have used log-mel filter bank (LMFB) audio features which have been commonly used with CNNs for ASC due to their favorable performance over hand-crafted acoustic features [52], [53]. Specifically, we have transformed the input audio clips into power spectrograms using short-time Fourier transform (STFT) with 2048 points by skipping every 1024 samples with a 2048-length Hann window. The hidden Markov toolkit (HTK) formula [54] was used to define the mel scale. Our resulting spectrograms were of size $[431 \times 128 \times 1]$, with 431 time-samples, 128 frequency-bins, and a single input channel.

We have also calculated the log-mel deltas and delta-deltas without padding as suggested by [11], which reduced the number of time samples to 423. So, the final shape of the used spectrograms became $[423 \times 128 \times 3]$ after stacking computed deltas into the channel axis. Additionally, many data augmentation methods are usually utilized in ASC systems due to the small size of the used datasets and to compensate for the device mismatch [13]. So, similar to state-of-the-art peer competitors, basic data augmentation techniques have been utilized in our experiments to help prevent overfitting and increase the diversity of data distribution. Specifically, mixup [10], and temporal cropping [11], have been used in this work.

C. Experimental Setup

In this work, the proposed algorithm was used to generate the best performing CNNs for each of the three datasets. The best CNN architecture discovered by the GA in each experiment was selected from the population of the last generation. Then, the best discovered CNN in each experiment was evaluated on each of the three datasets to test the generalization capability of the generated CNN architectures. The final evaluation process of each CNN involved training it from scratch for five independent runs for up to 510 epochs in a warm-restart learning-rate schedule [55]. The maximum and minimum learning rates were set to 0.1 and 1×10^{-5}, respectively. The best model was selected based on its validation accuracy to be tested on the test split. The average classification accuracy of the five runs for each CNN was used to compare the generated CNN architectures with state-of-the-art peer competitors.

In our experiments, we set most of the parameters of the proposed GA based on the conventions [21]. Table I shows the parameter settings used in our experiments. Specifically, we set the population size to 20, and the number of generations was set to 15. Although using larger values for the population size and the number of generations should enable the GA to find better CNN architectures, we show in section IV-E that the proposed algorithm was still able to find good-performing CNNs within a limited number of generations, and thus could be used with limited computational resources. The crossover and mutation probabilities were set to 0.9 and 0.1, respectively. We set equal probabilities for all mutation types of 0.2. For fitness evaluation, we have trained each individual by stochastic gradient descent (SGD) [56] for up to 350 epochs with early stopping if the validation accuracy does not improve by 70 epochs. The best classification accuracy of the validation split was then used as the fitness value. The possible values for the number of filters of convolutional blocks were set to $\{24, 48, 64, 96, 128, 160, 192, 256, 288, 384, 512, 572\}$. In addition, the number of convolutional blocks in each sub-CNN was set from 5 to 25, and the number of pooling layers was set from 2 to 10. Additionally, the number of frequency-dimension splits was set from 1 to 10. The choice of this range covers the design of regular single-path classical CNNs and up to ten splits. Since only a limited number of two or three splits has been usually utilized in the literature, the use of more than ten splits would provide unjustified search complexity. The minimum and maximum allowed batch sizes were set to 8 and 128, respectively. In our experiments, we use a small percentage for generating random population equal to 0.1, in order to offer more diversity without high randomness. This selection would generate two random individuals at the end of each generation to be included in the environmental selection process.

TABLE I
PARAMETER SETTINGS

parameter	Value
Population size	20
Number of generations	15
Crossover probability	0.9
Mutation Probability	0.1
Mutation: Add unit	0.2
Mutation: Remove unit	0.2
Mutation: Exchange unit	0.2
Mutation: Change pooling type	0.2
Mutation: Change number of splits	0.2
Percentage of random population	0.1
Number of epochs	350
Number of filters	24, 48, 64, 96, 128, 160, 192, 256, 288, 384, 512, 572
Number of convolutional blocks (Min – Max)	5 – 25
Number of pooling layers (Min – Max)	2 - 10
Number of frequency dimension splits (Min – Max)	1 - 10
Batch size (Min - Max)	8 - 128

All of our experiments were performed on a single machine equipped with an NVIDIA Tesla K80 dual GPU with 24 GB of GPU RAM. The dual GPU card was able to test the fitness of each two individuals in parallel using the proposed algorithm, with a maximum allowed memory of 12 GB for each individual. We report the consumed GPU Days in section IV-E as the total number of days taken by the aforementioned dual GPU to complete 15 generations of the proposed GA with the parameter configuration shown in Table I.

D. Compared Methods

The best CNN architectures discovered by the proposed algorithm on the development datasets of DCASE2020-Task1A, DCASE2019-Task1A, and DCASE2018-Task1A, are provided in Tables II, III, and IV, respectively. As can be seen from Table II, the best architecture found on the development dataset

of DCASE2020-Task1A has five sub-CNNs. Each sub-CNN is composed of nine convolutional blocks and four pooling layers. We refer to this architecture as GA-2020 which has 7,015,384 parameters. The best architecture found on the development dataset of DCASE2019-Task1A, GA-2019, is shown in Table III, which has three frequency-dimension splits. Each split is processed by a sub-CNN architecture with 18 layers comprising 9,767,328 parameters. The third architecture, denoted by GA-2018, is the best one found on the development dataset of DCASE2018-Task1A. This architecture has 2,400,776 parameters, and five sub-CNN architectures. Each sub-CNN comprises seven convolutional blocks and six pooling layers. It is interesting to know that even though the search space covered much more complex architectures with hundreds of millions of trainable parameters, the best architectures found on the three datasets have relatively simple structures with limited numbers of parameters.

In order to show the effectiveness and efficiency of the CNN architectures designed with the proposed algorithm, the state-of-the-art approaches to the ASC problem are selected as the peer competitors to this algorithm. Particularly, the peer competitors are chosen from three different categories. The first category includes manually designed state-of-the-art architectures for the ASC problem in addition to the official baseline systems of the chosen datasets. Furthermore, this category also includes a set of state-of-the-art image-classification CNNs, in order to gain some intuition about the efficiency of different architectures on the ASC task. For a fair comparison, in case that multiple models are evaluated in a given research, only the performance of the best model is presented. In the case that an ensemble of several models is evaluated, only the result of the best single model making up the ensemble is considered.

For further system evaluation, the second category includes a selected set of the top systems from each competition. Specifically, out of the top five teams according to the official rankings of each competition, we add the results of systems that have released the classification scores of the best single model for the development dataset. It is important to note that some systems have only released the classification results of the leaderboard dataset instead of the development dataset, while others have only released the results of the ensemble models. Such systems are excluded from the presented comparison tables. The third category demonstrates the automated-design approaches to each of the chosen benchmark datasets, which are still limited as we discussed in the introduction of this work. All systems are compared based on the classification accuracy and the number of parameters which represents an indicator for system complexity. The symbol "–" is used to imply that the information is not explicitly provided in the corresponding paper. Additionally, the number of consumed GPU days in the search process is provided for systems in the third category. Specifically, the number of GPU Days for a given algorithm is calculated by multiplying the number of days the algorithm needed until the best architecture was found by the number of the employed GPU cards. The provided models from each category in Tables V, VI, VII are sorted in ascending order based on their classification accuracy, with bold lines splitting the three categories in each table.

TABLE II
THE SUB-CNN ARCHITECTURE OF THE BEST DISCOVERED CNN ARCHITECTURE ON THE DEVELOPMENT DATASET OF DCASE2020-TASK1A [NUMBER OF SPLITS = 5]

Unit Id	Unit Type	Unit Configuration
1	Convolutional block	Number of filters = 24
2	Convolutional block	Number of filters = 48
3	Convolutional block	Number of filters = 48
4	Convolutional block	Number of filters = 96
5	Convolutional block	Number of filters = 128
6	Convolutional block	Number of filters = 192
7	Pooling Layer	Max Pooling
8	Convolutional block	Number of filters = 192
9	Convolutional block	Number of filters = 192
10	Pooling Layer	Max Pooling
11	Pooling Layer	Max Pooling
12	Pooling Layer	Average Pooling
13	Convolutional block	Number of filters = 192

TABLE III
THE SUB-CNN ARCHITECTURE OF THE BEST DISCOVERED CNN ARCHITECTURE ON THE DEVELOPMENT DATASET OF DCASE2019-TASK1A [NUMBER OF SPLITS = 3]

Unit Id	Unit Type	Unit Configuration
1	Convolutional block	Number of filters = 64
2	Convolutional block	Number of filters = 64
3	Convolutional block	Number of filters = 64
4	Convolutional block	Number of filters = 64
5	Convolutional block	Number of filters = 64
6	Convolutional block	Number of filters = 96
7	Convolutional block	Number of filters = 128
8	Pooling Layer	Max Pooling
9	Convolutional block	Number of filters = 160
10	Convolutional block	Number of filters = 192
11	Convolutional block	Number of filters = 192
12	Convolutional block	Number of filters = 192
13	Convolutional block	Number of filters = 192
14	Pooling Layer	Max Pooling
15	Convolutional block	Number of filters = 192
16	Convolutional block	Number of filters = 192
17	Convolutional block	Number of filters = 192
18	Convolutional block	Number of filters = 256

The presented values for peer-competitors in Tables V, VI, VII are provided following their research papers, with a few exceptions for state-of-the-art image-classification CNNs which were later evaluated on the ASC datasets in the specialized literature. Specifically, ResNet [44] was evaluated on the development dataset of DCASE2020-Task1A in [57], the development dataset of DCASE2019-Task1A in [58], and the development dataset of DCASE2018-Task1A in [59]. VGG-8 [42] was evaluated on the development dataset of DCASE2019-Task1A in [58], and VGG-19 was evaluated on the development dataset of DCASE2018-Task1A in [59]. Additionally, for the development dataset of DCASE2020-Task1A, SegNet [60] was evaluated in [61], and LCNN [62] was evaluated in [57]. MobileNet was evaluated on the development dataset of DCASE2019-Task1A in [58], and finally, Xception was evaluated on the development dataset of DCASE2018-Task1A in [53]. So, we report their accuracy values from these researches.

TABLE IV
THE SUB-CNN ARCHITECTURE OF THE BEST DISCOVERED CNN ARCHITECTURE ON THE DEVELOPMENT DATASET OF DCASE2018-TASK1A [NUMBER OF SPLITS = 5]

Unit Id	Unit Type	Unit Configuration
1	Convolutional block	Number of filters = 24
2	Convolutional block	Number of filters = 48
3	Pooling Layer	Max Pooling
4	Convolutional block	Number of filters = 96
5	Convolutional block	Number of filters = 96
6	Pooling Layer	Max Pooling
7	Convolutional block	Number of filters = 96
8	Convolutional block	Number of filters = 128
9	Pooling Layer	Max Pooling
10	Convolutional block	Number of filters = 128
11	Pooling Layer	Max Pooling
12	Pooling Layer	Max Pooling
13	Pooling Layer	Average Pooling

E. Results and Analysis

Table V shows the comparison results between the proposed algorithm and the chosen peer competitors on the development dataset of DCASE2020-Task1A. The results show that the discovered network on this dataset, GA-2020, has outperformed the official dataset baseline, DCASE2020 baseline, by 17.2%. Specifically, it has achieved a classification accuracy of 71.3% outperforming the top approach from the first category, AMFM [63], by 0.5%. However, the top five teams of the competition presented in the second category outperform the proposed system by 0.4% to 5.6%. The best-discovered network on DCASE2018-Task1A, GA-2018, was also evaluated on this dataset. Using only 2.4M parameters, this network has achieved a competitive classification accuracy of 70.7% outperforming architectures of the first category, except for AMFM [63] which had a slightly improved classification accuracy of 70.8%. On the other hand, the best discovered network on DCASE2019-Task1A, GA-2019, had a significantly lower classification accuracy of 65.6%, and thus indicating that it does not generalize well on this dataset. However, all of the three architectures outperformed the only publicly available automated approach on this dataset, E-DARTS [37]. Specifically, GA-2020 has outperformed it by 6.7%, GA-2018 by 6.1%, and GA-2019 by 1.0%.

Table VI shows the comparison results between the proposed algorithm and the peer competitors on the development dataset of DCASE2019-Task1A. Noting that only manually designed approaches are available on this dataset, the results show that GA-2019 outperforms all of the manually designed peer competitors of the first category in addition to two systems from the second category. Specifically, GA-2019 achieves a classification accuracy of 78.2%, which is approximately 15.7% higher than the official dataset baseline and 1.0% higher than the top system of the first category, Factorized CNN [64]. Out of the three systems of the second category, only Zhang_IOA_task1a [65] has outperformed this result by 6.1%. Furthermore, GA-2018 was also evaluated on this dataset. Using a significantly lower number of parameters than most peer competitors, this architecture has achieved a competitive classification accuracy of 77.8% outperforming all manually designed architectures of the first category. However, the squeeze-excitation network [66] and Factorized CNN [64] have achieved competitive classification results using only 506k and 871K parameters, respectively. Interestingly, GA-2020 has slightly outperformed GA-2019 on this dataset by achieving a classification accuracy of 78.5%.

The comparison results between the proposed algorithm and the peer competitors on the development dataset of DCASE2018-Task1A are shown in Table VII. The results show that the proposed GA-2018 achieves a classification accuracy of 76.7% outperforming all peer competitors of the first category except the MoE-decoder framework proposed in [67], which outperforms GA-2018 by 0.8%. Additionally, it outperforms the official dataset baseline by 17.0% , and three systems of the second category by 0.1% to 6.4%. Interestingly, GA-2020 has also slightly outperformed GA-2018 on this dataset by achieving a classification accuracy of 77.5%. This result is equal to that achieved by the top system of the first category and outperforms four systems from the second one, while only Liping_CQU_task1a [68] has outperformed this result by 2.3%. Furthermore, GA-2019 was also evaluated on this dataset, and it achieved a classification accuracy of 74.4%. However, all of the three architectures outperformed the only publicly-available automated approach on this dataset, DeepSAGA [30]. Specifically, GA-2018 has outperformed it by 3.9%, GA-2019 by 1.6%, and GA-2020 by 4.7%.

It is important to think about the competitive performance of GA-2020 on all three datasets even though it was discovered by searching on the development dataset of DCASE2020-Task1A. A closer look at the nature of the sound samples from those datasets reveals the challenging nature of the multi-device dataset of DCASE2020-Task1A over the two single-device datasets of DCASE2019-Task1A and DCASE2018-Task1A. In fact, to make it more challenging, some recording devices appear only in the test subset of this dataset. This reveals that searching for an efficient architecture on more complex and challenging datasets can generate even more promising architectures than searching over the simpler ones. On the other hand, each of the discovered architectures on the two single-device datasets, GA-2018 and GA-2019, has relatively generalized well on both datasets. This can be explained by the closely related nature of the two datasets. Furthermore, only GA-2018 has generalized well on the multi-device dataset of DCASE2020-Task1A, when GA-2019 could not achieve such good generalization performance. This indicates that searching over single-device datasets does not guarantee a generalized good performance over multi-device datasets.

Using the proposed algorithm, the search process on the development dataset of DCASE2020-Task1A was the longest one with 30 GPU days, due to the large number of training samples compared to the other datasets. The search process on DCASE2019-Task1A follows it with 25 GPU days, and finally, the search process on DCASE2018-Task1A took 17 GPU days. We can see that the search time was much larger in DeepSAGA with 75 GPU days to search on the dataset of DCASE2018-Task1A. On the other hand, E-DARTS only needed 7.5 GPU days to search on the dataset of DCASE2020-Task1A. However, the compared method has outperformed the two automated approaches in terms of classification accuracy.

TABLE V
THE CLASSIFICATION RESULTS ON THE DEVELOPMENT DATASET OF DCASE2020-TASK1A

System	Accuracy	Complexity (# of Parameters)	Time (GPU days)	Type (Automatic – Hand-crafted)
DCASE2020 Baseline [13]	54.1%	5 M	-	Hand-crafted
SegNet [60]	56.6%	31.8 M	-	Hand-crafted
CNN w/ SE-module [69]	64.2%	0.5 M	-	Hand-crafted
ResNet-based [70]	64.2%	-	-	Hand-crafted
Mini-SegNet [61]	65.2%	2.1 M	-	Hand-crafted
ResNet [44]	67.3%	23.6 M	-	Hand-crafted
VGGNet-based w/ HRTF [71]	67.5%	-	-	Hand-crafted
ResNet-based w/ HRTF [71]	68.5%	-	-	Hand-crafted
Tag-Rep [72]	69.3%	0.6 M	-	Hand-crafted
LCNN [62]	69.4%	5.6 M	-	Hand-crafted
LCNN w/ CBAM [57]	70.4%	0.85 M	-	Hand-crafted
AMFM [63]	**70.8**%	1.5 M	-	Hand-crafted
Gao_UNISA_task1a [73]	71.7%	4.3 M	-	Hand-crafted
Koutini_CPJKU_task1a [74]	71.8%	36 M	-	Hand-crafted
Jie_Maxvision_task1a [75]	72.1%	3 M	-	Hand-crafted
Suh_ETRI_task1a [14]	73.7%	13 M	-	Hand-crafted
Hu_GT_task1a [12]	**76.9**%	17 M	-	Hand-crafted
E-DARTS [37]	64.6%	5.5 M	7.5 Days	Automatic
Proposed GA-2019	65.6%	9.8 M	25 Days	Automatic
Proposed GA-2018	70.7%	2.4 M	17 Days	Automatic
Proposed GA-2020	**71.3**%	7 M	30 Days	Automatic

TABLE VI
THE CLASSIFICATION RESULTS ON THE DEVELOPMENT DATASET OF DCASE2019-TASK1A

System	Accuracy	Complexity (# of Parameters)	Time (GPU days)	Type (Automatic – Hand-crafted)
DCASE2019 Baseline [16]	62.5%	116 k	-	Hand-crafted
BCNN [76]	66.0%	-	-	Hand-crafted
ResNet [44]	66.4%	23.6 M	-	Hand-crafted
LogMel-DNN [77]	66.8%	-	-	Hand-crafted
MobileNet[78]	67.9%	3.2 M	-	Hand-crafted
CMAM [79]	68.8%	-	-	Hand-crafted
CompactNet [58]	69.0%	9.8 M	-	Hand-crafted
VGG-8 [42]	70.0%	4.7 M	-	Hand-crafted
CAA-Net [80]	72.2%	-	-	Hand-crafted
SSL-based ResNet [81]	75.1%	11.7 M	-	Hand-crafted
Squeeze-Excitation Network [66]	76.7%	506 k	-	Hand-crafted
Factorized CNN [64]	**77.2**%	871 K	-	Hand-crafted
Seo_LGE_task1a [82]	76.0%	-	-	Hand-crafted
Huang_IL_task1a [83]	77.9%	24.5 M	-	Hand-crafted
Zhang_IOA_task1a [65]	**84.3**%	-	-	Hand-crafted
Proposed GA-2018	77.8%	2.4 M	17 Days	Automatic
Proposed GA-2019	**78.2**%	9.8 M	25 Days	Automatic
Proposed GA-2020	**78.5**%	7 M	30 Days	Automatic

V. CONCLUSION

The aim of this research is to develop a GA that could automatically generate optimized CNN architectures for the ASC task by exploring the architecture search space in sub-CNN models in addition to classical single-path CNNs. The proposed algorithm has successfully achieved this goal by showing promising results on three challenging ASC datasets compared to state-of-the-art hand-crafted architectures. Furthermore, this algorithm presents an effective fitness evaluation function that fully utilizes the available computational resources by combining the use of the cache component with an adaptable batch size technique in a parallel computation scheme. Additionally, the early stopping method has been utilized to stop training inefficient architectures and thus save more resources for evaluating the efficient ones.

Usually, neural architecture search and optimization algorithms have been used to generate the best network architectures for a given dataset to be used with its specific data. However, our experiments have revealed that performing a search process over a challenging multi-device dataset may generate CNN architectures that generalize well on simpler datasets, and possibly even better than those optimized specifically for them. Further studies using the proposed algorithm with a variety of ASC datasets can help better understand the outcomes of this study. Additionally, future work should investigate whether these results generalize over other architecture search algorithms and possibly with other classification tasks. Such studies would be useful to map the relation between specific classification tasks and the nature of the optimized architectures, which could be helpful to find general good architectures for a set of closely-related datasets. On top of that, being optimized for sound processing, the proposed

TABLE VII
THE CLASSIFICATION RESULTS ON THE DEVELOPMENT DATASET OF DCASE2018-TASK1A

System	Accuracy	Complexity (# of Parameters)	Time (GPU days)	Type (Automatic – Hand-crafted)
DCASE2018 Baseline [16]	59.7%	116 k	-	Hand-crafted
Hybrid-DNN [84]	66.1%	-	-	Hand-crafted
TTM-CNN [85]	66.2%	-	-	Hand-crafted
CNN-8 [86]	68.0%	4.7 M	-	Hand-crafted
LogMel-DNN [77]	68.2%	-	-	Hand-crafted
VGG-19 [42]	68.2%	45.3 M	-	Hand-crafted
Xception [87]	68.3 %	21.9 M	-	Hand-crafted
ResNet-101 [44]	68.7%	42.6 M	-	Hand-crafted
VGGNet w/ distillation [59]	69.6%	45.3 M	-	Hand-crafted
FRCNN [53]	71.6%	17 M	-	Hand-crafted
CNN w/ attention [88]	72.7%	-	-	Hand-crafted
CAA-Net [80]	72.7%	-	-	Hand-crafted
SubSpectralNet [15]	74.1%	-	-	Hand-crafted
SceneNet [89]	75.8%	1.3 M	-	Hand-crafted
MoE-decoder framework [67]	**77.5**%	-	-	Hand-crafted
Zeinali_BUT_task1a [90]	70.3%	1 M	-	Hand-crafted
Sakashita_TUT_task1a [91]	72.9%	-	-	Hand-crafted
Li_BIT_task1a [92]	76.6%	6 M	-	Hand-crafted
Dorfer_CPJKU_task1a [93]	77.1%	-	-	Hand-crafted
Liping_CQU_task1a [68]	**79.8**%	22 M	-	Hand-crafted
DeepSAGA [30]	72.8%	689 K	75 Days	Automatic
Proposed GA-2019	74.4%	9.8 M	25 Days	Automatic
Proposed GA-2018	**76.7**%	2.4 M	17 Days	Automatic
Proposed GA-2020	**77.5**%	7 M	30 Days	Automatic

GA could also be used with further optimizations for other sound classification tasks which could be tested in the future. Possible optimizations to the proposed algorithm may include optimizing the fixed parameters of convolutional and pooling layers, or even replacing them with more advanced building blocks such as residual and squeeze-excitation blocks. Another area to explore would be to introduce some level of overlap between splits, as to not lose potentially useful information from inspected spectrograms.

REFERENCES

[1] D. Barchiesi, D. Giannoulis, D. Stowell, and M. D. Plumbley, "Acoustic scene classification: Classifying environments from the sounds they produce," *IEEE Signal Processing Magazine*, vol. 32, no. 3, pp. 16–34, 2015.

[2] N. Sawhney and P. Maes, "Situational awareness from environmental sounds," *Project Rep. for Pattie Maes*, pp. 1–7, 1997.

[3] I. Martín-Morató, M. Cobos, and F. J. Ferri, "A case study on feature sensitivity for audio event classification using support vector machines," in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2016, pp. 1–6.

[4] S. Waldekar and G. Saha, "Analysis and classification of acoustic scenes with wavelet transform-based mel-scaled features," *Multimedia Tools and Applications*, vol. 79, no. 11, pp. 7911–7926, 2020.

[5] J. C. Brown, "Calculation of a constant q spectral transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

[6] S. Abidin, R. Togneri, and F. Sohel, "Spectrotemporal analysis using local binary pattern variants for acoustic scene classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 11, pp. 2112–2121, 2018.

[7] A. Rakotomamonjy and G. Gasso, "Histogram of gradients of time–frequency representations for audio scene classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 1, pp. 142–153, 2014.

[8] M. Valenti, S. Squartini, A. Diment, G. Parascandolo, and T. Virtanen, "A convolutional neural network approach for acoustic scene classification," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 1547–1554.

[9] J. Abeßer, "A review of deep learning based methods for acoustic scene classification," *Applied Sciences*, vol. 10, no. 6, 2020.

[10] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv:1710.09412*, 2017.

[11] M. D. McDonnell and W. Gao, "Acoustic scene classification using deep residual networks with late fusion of separated high and low frequency paths," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 141–145.

[12] H. Hu, C.-H. H. Yang, X. Xia, X. Bai, X. Tang, Y. Wang, S. Niu, L. Chai, J. Li, H. Zhu *et al.*, "Device-robust acoustic scene classification based on two-stage categorization and data augmentation," *arXiv:2007.08389*, 2020.

[13] T. Heittola, A. Mesaros, and T. Virtanen, "Acoustic scene classification in dcase 2020 challenge: generalization across devices and low complexity solutions," *arXiv:2005.14623*, 2020.

[14] S. Suh, S. Park, Y. Jeong, and T. Lee, "Designing acoustic scene classification models with cnn variants," DCASE2020 Challenge, Tech. Rep, Tech. Rep., 2020.

[15] S. S. R. Phaye, E. Benetos, and Y. Wang, "Subspectralnet–using sub-spectrogram based convolutional neural networks for acoustic scene classification," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 825–829.

[16] A. Mesaros, T. Heittola, and T. Virtanen, "A multi-device dataset for urban acoustic scene classification," *arXiv:1807.09840*, 2018.

[17] D. Agnelli, A. Bollini, and L. Lombardi, "Image classification: an evolutionary approach," *Pattern Recognition Letters*, vol. 23, no. 1-3, pp. 303–309, 2002.

[18] Z. Liu, A. Liu, C. Wang, and Z. Niu, "Evolving neural network using real coded genetic algorithm (ga) for multispectral image classification," *Future Generation Computer Systems*, vol. 20, no. 7, pp. 1119–1129, 2004.

[19] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 277–291, 2021.

[20] N. W. Hasan, A. S. Saudi, M. I. Khalil, and H. M. Abbas, "Automatically designing cnn architectures for acoustic scene classification," in *2021 16th International Conference on Computer Engineering and Systems (ICCES)*, 2021, pp. 1–6.

[21] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[22] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming: an introduction*. Morgan Kaufmann Publishers San Francisco, 1998, vol. 1.

[23] I. Rechenberg, "Evolution strategy: Nature's way of optimization," in *Optimization: Methods and applications, possibilities and limitations*. Springer, 1989, pp. 106–126.

[24] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolution*. Wiley, 1966.

[25] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, no. 2–3, p. 95–99, Oct. 1988.

[26] E. Galvan and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Transactions on Artificial Intelligence*, pp. 1–1, 2021.

[27] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[28] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, no. 1-2, pp. 1–61, 2001.

[29] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020.

[30] C. Roletscheck, T. Watzka, A. Seiderer, D. Schiller, and E. André, *Using an evolutionary approach to explore convolutional neural networks for acoustic scene classification*. Universität Augsburg, 2018.

[31] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[32] J. Li, C. Liang, B. Zhang, Z. Wang, F. Xiang, and X. Chu, "Neural architecture search on acoustic scene classification," *arXiv:1912.12825*, 2019.

[33] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[34] G. Dekkers, L. Vuegen, T. van Waterschoot, B. Vanrumste, and P. Karsmakers, "Dcase 2018 challenge-task 5: Monitoring of domestic activities based on multi-channel acoustics," *arXiv:1807.11246*, 2018.

[35] Y. Wu and T. Lee, "Searching for efficient network architectures for acoustic scene classification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020)*, Tokyo, Japan, November 2020, pp. 220–224.

[36] T. Heittola, A. Mesaros, and T. Virtanen, "Tau urban acoustic scenes 2020 3class, development dataset," 2020.

[37] N. W. Hasan, A. S. Saudi, M. I. Khalil, and H. M. Abbas, "E-darts: Enhanced differentiable architecture search for acoustic scene classification," in *2021 16th International Conference on Computer Engineering and Systems (ICCES)*, 2021, pp. 1–6.

[38] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv:1806.09055*, 2018.

[39] T. Heittola, A. Mesaros, and T. Virtanen, "Tau urban acoustic scenes 2020 mobile, development dataset," 2020.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[45] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," *arXiv:1702.03275*, 2017.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[47] R. Housley, "A 224-bit one-way hash function: Sha-224," RFC 3874, September, Tech. Rep., 2004.

[48] H. Iba and N. Noman, *Deep Neural Evolution: Deep Learning with Evolutionary Computation*. Springer, 2020.

[49] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[50] T. Heittola, A. Mesaros, and T. Virtanen, "Tau urban acoustic scenes 2019, development dataset," 2019.

[51] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8. Citeseer, 2015, pp. 18–25.

[52] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "Cnn architectures for large-scale audio classification," in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2017, pp. 131–135.

[53] T. Zhang, J. Liang, and B. Ding, "Acoustic scene classification using deep cnn with fine-resolution feature," *Expert Systems with Applications*, vol. 143, p. 113067, 2020.

[54] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The htk book," *Cambridge university engineering department*, vol. 3, no. 175, p. 12, 2002.

[55] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv:1608.03983*, 2016.

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[57] H.-j. Shim, J.-w. Jung, J.-h. Kim, and H.-j. Yu, "Capturing discriminative information using a deep architecture in acoustic scene classification," *Applied Sciences*, vol. 11, no. 18, p. 8361, 2021.

[58] J. Liang, T. Zhang, and G. Feng, "Channel compression: Rethinking information redundancy among channels in cnn architecture," *IEEE Access*, vol. 8, pp. 147 265–147 274, 2020.

[59] L. Gao, H. Mi, B. Zhu, D. Feng, Y. Li, and Y. Peng, "An adversarial feature distillation method for audio classification," *IEEE Access*, vol. 7, pp. 105 319–105 330, 2019.

[60] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[61] X. Ma, Y. Shao, Y. Ma, and W.-Q. Zhang, "Deep semantic encoder-decoder network for acoustic scene classification with multiple devices," in *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2020, pp. 365–370.

[62] X. Wu, R. He, Z. Sun, and T. Tan, "A light cnn for deep face representation with noisy labels," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2884–2896, 2018.

[63] H.-j. Shim, J.-h. Kim, J.-w. Jung, and H.-J. Yu, "Attentive max feature map for acoustic scene classification with joint learning considering the abstraction of classes," *arXiv:2104.07213*, 2021.

[64] J. Cho, S. Yun, H. Park, J. Eum, and K. Hwang, "Acoustic scene classification based on a large-margin factorized cnn," *arXiv:1910.06784*, 2019.

[65] H. Chen, Z. Liu, Z. Liu, P. Zhang, and Y. Yan, "Integrating the data augmentation scheme with various classifiers for acoustic scene modeling," DCASE2019 Challenge, Tech. Rep., June 2019.

[66] J. Naranjo-Alcazar, S. Perez-Castanos, P. Zuccarello, and M. Cobos, "Acoustic scene classification with squeeze-excitation residual networks," *IEEE Access*, vol. 8, pp. 112 287–112 296, 2020.

[67] L. Pham, H. Phan, T. Nguyen, R. Palaniappan, A. Mertins, and I. McLoughlin, "Robust acoustic scene classification using a multi-spectrogram encoder-decoder framework," *Digital Signal Processing*, vol. 110, p. 102943, 2021.

[68] Y. Liping, C. Xinxing, and T. Lianjie, "Acoustic scene classification using multi-scale features," DCASE2018 Challenge, Tech. Rep., September 2018.

[69] J. Naranjo-Alcazar, S. Perez-Castanos, M. Cobos, F. J. Ferri, and P. Zuccarello, "Task 1a dcase 2021: Acoustic scene classification with mismatch-devices using squeeze-excitation technique and low-complexity constraint," *arXiv:2107.14658*, 2021.

[70] Y. Lee, S. Lim, and I.-Y. Kwak, "Cnn-based acoustic scene classification system," *Electronics*, vol. 10, no. 4, p. 371, 2021.

[71] Y. Liu, H. Yang, C. Shi, and J. Liang, "Hrtf-based data augmentation method for acoustic scene classification," in *2021 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE, 2021, pp. 1–5.

[72] J.-h. Kim, J.-w. Jung, H.-j. Shim, and H.-j. Yu, "Audio tag representation guided dual attention network for acoustic scene classification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events Workshop (DCASE)*, 2020.

[73] W. Gao and M. McDonnell, "Acoustic scene classification using deep residual networks with focal loss and mild domain adaptation," DCASE2020 Challenge, Tech. Rep., June 2020.

[74] K. Koutini, F. Henkel, H. Eghbal-zadeh, and G. Widmer, "CP-JKU submissions to DCASE'20: Low-complexity cross-device acoustic scene classification with RF-regularized CNNs," DCASE2020 Challenge, Tech. Rep., June 2020.

[75] L. Jie, "Acoustic scene classification with residual networks and attention mechanism," DCASE2020 Challenge, Tech. Rep., June 2020.

[76] X. Y. Kek, C. S. Chin, and Y. Li, "Acoustic scene classification using bilinear pooling on time-liked and frequency-liked convolution neural network," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 3189–3194.

[77] C. Paseddula and S. V. Gangashetty, "Acoustic scene classification using single frequency filtering cepstral coefficients and dnn," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–6.

[78] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[79] R. Zhang, W. Zou, and X. Li, "Cross-task pre-training for on-device acoustic scene classification," *arXiv:1910.09935*, 2019.

[80] Z. Ren, Q. Kong, J. Han, M. D. Plumbley, and B. W. Schuller, "Caa-net: Conditional atrous cnns with attention for explainable device-robust acoustic scene classification," *IEEE Transactions on Multimedia*, 2020.

[81] A. M. Tripathi and A. Mishra, "Self-supervised learning for environmental sound classification," *Applied Acoustics*, vol. 182, p. 108183, 2021.

[82] S. Hyeji and P. Jihwan, "Acoustic scene classification using various pre-processed features and convolutional neural networks," DCASE2019 Challenge, Tech. Rep., June 2019.

[83] J. Huang, P. Lopez Meyer, H. Lu, H. Cordourier Maruri, and J. Del Hoyo, "Acoustic scene classification using deep learning-based ensemble averaging," DCASE2019 Challenge, Tech. Rep., June 2019.

[84] X. Bai, J. Du, Z.-R. Wang, and C.-H. Lee, "A hybrid approach to acoustic scene classification based on universal acoustic models." in *Interspeech*, 2019, pp. 3619–3623.

[85] T. Zhang and J. Wu, "Constrained learned feature extraction for acoustic scene classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 8, pp. 1216–1228, 2019.

[86] Q. Kong, T. Iqbal, Y. Xu, W. Wang, and M. D. Plumbley, "DCASE 2018 challenge surrey cross-task convolutional neural network baseline," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018)*, November 2018, pp. 217–221.

[87] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[88] Z. Ren, Q. Kong, J. Han, M. D. Plumbley, and B. W. Schuller, "Attention-based atrous convolutional neural networks: Visualisation and understanding perspectives of acoustic scenes," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 56–60.

[89] L. Zhang, Z. Shi, and J. Han, "Pyramidal temporal pooling with discriminative mapping for audio classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 770–784, 2020.

[90] H. Zeinali, L. Burget, and H. Cernocky, "Convolutional neural networks and x-vector embedding for dcase2018 acoustic scene classification challenge," DCASE2018 Challenge, Tech. Rep., September 2018.

[91] Y. Sakashita and M. Aono, "Acoustic scene classification by ensemble of spectrograms based on adaptive temporal divisions," DCASE2018 Challenge, Tech. Rep., September 2018.

[92] Z. Li, L. Zhang, S. Du, and W. Liu, "Acoustic scene classification based on binaural deep scattering spectra with CNN and LSTM," DCASE2018 Challenge, Tech. Rep., September 2018.

[93] M. Dorfer, B. Lehner, H. Eghbal-zadeh, H. Christop, P. Fabian, and W. Gerhard, "Acoustic scene classification with fully convolutional neural networks and I-vectors," DCASE2018 Challenge, Tech. Rep., September 2018.

Noha W. Hasan was born in Benha, Qalubiya Governorate, Egypt in 1993. She received her B.Sc. degree in computer and systems engineering from Ain Shams University, Cairo, Egypt, in 2016. Currently, she is a teaching and research assistant and working on her M.Sc. degree at the computer and systems engineering department, Ain Shams University, Cairo, Egypt. Her research interests include evolutionary algorithms, evolutionary deep learning, audio signal processing, and sound classification.

Ali S. Saudi received the B.Sc. degree in computer system engineering from Benha University, Qalubiya Governorate, Egypt, in 2002, the M.Sc. degree in computer engineering from Arab Academy for Science, Technology and Maritime Transport, Cairo, Egypt, in 2012, and the Ph.D. degree in computer and systems engineering from Ain Shams University, Cairo, Egypt, in 2019. He is currently a Lecturer at the Faculty of Computers and Artificial Intelligence, Benha University, Qalubiya Governorate, Egypt. His research interests are in the areas of speech signal processing, machine learning, deep learning, and artificial intelligence.

Mahmoud I. Khalil received his B.Sc. and M.Sc. degrees in electrical engineering from Ain Shams University, Cairo, Egypt, in 1992 and 1996, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada. He is currently a professor of computer and systems engineering. His current research interests include pattern recognition, computer vision, and neural networks.

Hazem M. Abbas received his B.Sc. and M.Sc. degrees in Computer Engineering from Ain Shams University, Cairo, and Ph.D. from Queen's University, Kingston, Canada. He is currently a Visiting Professor at the Queen's University School of Computing. He has held leading positions at the Dept. of Computer and Systems Engineering at Ain Shams University and at the German University in Cairo. He worked for the Royal Military College, IBM Toronto Lab, and Mentor Graphics. Prof. Abbas is a Senior Member of the IEEE and chaired the IEEE Signal Processing Chapter in Cairo. His research interests are in the areas of Machine Learning and Computational Intelligence.