Efficient Multi-objective Evolutionary 3D Neural Architecture Search for COVID-19 Detection with Chest CT Scans

Xin He*, Shihao Wang†, Guohao Ying‡, Jiyong Zhang§ and Xiaowen Chu∗§

*Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
Email: {csxinhe,shwang,chxw}@comp.hkbu.edu.hk
†Viterbi School of Engineering, University of Southern California, CA, USA
Email: gying@usc.edu
‡School of Automation, Hangzhou Dianzi University, Hang Zhou, China
jzhang@hdu.edu.cn

Abstract—COVID-19 pandemic has spread globally for months. Due to its long incubation period and high testing cost, there is no clue showing its spread speed is slowing down, and hence a faster testing method is in dire need. This paper proposes an efficient Evolutionary Multi-objective neural Architecture Search (EMARS) framework, which can automatically search for 3D neural architectures based on a well-designed search space for COVID-19 chest CT scan classification. Within the framework, we use weight sharing strategy to significantly improve the search efficiency and finish the search process in 8 hours. We also propose a new objective, namely potential, which is of benefit to improve the search process’s robustness. With the objectives of accuracy, potential, and model size, we find a lightweight model (3.39 MB), which outperforms three baseline human-designed models, i.e., ResNet3D101 (325.21 MB), DenseNet3D121 (43.06 MB), and MC3_18 (43.84 MB). Besides, our well-designed search space enables the class activation mapping algorithm to be easily embedded into all searched models, which can provide the interpretability for medical diagnosis by visualizing the judgment based on the models to locate the lesion areas.

Index Terms—COVID-19, Evolutionary Algorithm, Neural Architecture Search, Multi-objective optimization

I. INTRODUCTION

Ten months after the global pandemic starts, COVID-19 is still a significant problem for most countries in the world. Besides the 14-days incubation period making it hard to trace and guarantee the patients, accurate diagnosis is also hard to perform. One of the most widely used testing methods is to use reverse transcription-polymerase chain reaction (RT-PCR) [1] for viral testing; however, it is relatively slow, expensive, and requires professionals, reagents, and exceptional devices to perform. To facilitate rapid COVID-19 diagnosis, many researchers attempt to accelerate COVID-19 diagnosis by deep learning (DL) techniques. However, most of the proposed models are designed manually, which requires the designer’s abundant experience and high expertise. This situation hinders the generalization of DL for COVID-19 diagnosis. At the same time, an excellent neural network needs to perform well on multiple metrics (e.g., precision and sensitivity) before applying it to actual disease diagnosis. However, in practice, no human expert can guarantee to find the optimal neural architecture.

This paper proposes an efficient evolutionary multi-objective neural architecture search (EMARS) method, which can automatically search 3D neural networks for COVID-19 detection. Our method can achieve 89.74% sensitivity on Clean-CC-CCII dataset [2], which is significantly higher than the average sensitivity of antigen tests (56.2%) [3], similar to radiologist’s average diagnosis sensitivity by Chest CT (92%) [4], and only slightly worse than average sensitivity of RT-PCR (95.2%) [3].

The neural architecture search (NAS) technique is a feasible and promising solution to automate and accelerate the process of model designing, as many studies have experimentally demonstrated that NAS-designed models outperform handcrafted models. There are mainly four classes of NAS methods: reinforcement learning (RL)-based methods [5]–[8], gradient descent (GD)-based methods [9]–[11], surrogate model-based optimization (SMBO) methods [12], and evolutionary algorithm (EA)-based methods [13]–[19]. Many early studies focus on searching for neural architectures that achieve higher performance (e.g., classification accuracy), regardless of the resource consumption and model size. For example, Zoph et al. [5] were the first to propose RL-based NAS methods and successfully found models outperforming state-of-the-art (SOTA) manually designed models, while they took 800 GPUs and 22,400 GPU days for searching, which is unacceptable for individuals and small companies. The following GD-based methods, such as DARTS [9], significantly improve the search efficiency. However, as stated in [10], DARTS tends to select the simpler operations (e.g., skip-connect) in the later stage of the search, resulting in a lack of diversity in the searched models. Although the EA-based methods can escape local optima and find promising models, it introduces randomness during the search stage. The EA-based methods have a huge requirement of computational resources and time, as the typical EA-based NAS methods generally need to train each individual for several epochs before obtaining their

§Corresponding author.
validation results.

In this paper, we propose an Evolutionary Multi-objective neural ARChitecture Search (EMARS) framework to resolve the above inherent issues. Many previous NAS studies search only one cell and repeat a fixed number of searched cell to construct the final model. Inspired by [8], [20], our search space is factorized into multiple searchable cells and blocks (shown in Fig. 1); therefore, the model diversity in our work is guaranteed. Besides, we use the mobile inverted bottleneck convolution (MBConv) as the candidate operations. MBConv requires less computation than standard convolution modules and has been demonstrated effective in improving model performance. In EMARS, individuals indicate child architectures derived in the same SuperNet initialized at the beginning of the algorithm. In other words, all individuals share the weights of the SuperNet among each other, which can significantly improve the efficiency of our evolutionary algorithm. The search process of 100 epochs can be finished in about 8 hours using 4 Nvidia Tesla V100 GPUs.

Many multi-objective NAS methods [8], [11], [18], [19] only considered accuracy and model size as objectives. In this work, we introduce a new objective, namely potential, into the NSGA-III algorithm [21]. We experimentally demonstrated that potential is of great benefit to improve the robustness of the search process by applying EMARS to three publicly available COVID-19 CT scan datasets (Clean-CC-CCII [22], MosMedData [23], and Covid-CTset [24]). According to the experimental results, EMARS can effectively find a series of neural architectures with higher accuracy than baseline models (ResNet3D101 [25], DenseNet3D121 [26], and MC3_18 [25]), and these searched architectures cover a wide range over the model size. Furthermore, medical diagnoses generally require interpretability of the decision, so we apply the class activation mapping (CAM) [27] algorithm into our EMARS series models to visualize the judgment of the model, which can help doctors understand the chest CT scan while verifying the validity.

The contributions of our work are summarized as follows:

1) We design a 3D search space, which is factorized into multiple searchable cells and blocks and hence increase the model diversity. We also use the weight sharing strategy, which significantly improves search efficiency. The search process of 100 epochs can be finished in about 8 hours.

2) We propose an EA-based NAS framework, namely EMARS, which is capable of scalability, i.e., we can easily apply multiple objectives into EMARS for optimization. Specifically, we introduce a new objective, called potential, and experimentally prove it effective to improve the robustness of the search process.

3) With the proposed search space and EMARS, we find a series of neural architectures, all of which outperform three baseline models with a much smaller mode size.

4) In our search space, a global average pooling layer is inserted before the fully connected layer; therefore, the class activation mapping (CAM) [27] algorithm can be easily embedded into our searched models, which can help doctors locate the discriminative lesion areas on the CT scan images.

The rest of the paper is organized as follows. Section II describes the related work. Section III introduces the search space for building 3D neural architectures. Section IV illustrates our search algorithm, including the warm-up, selection, crossover, and mutation processes. We introduce the experimental implementations in Section V and present and analyze the results in Section VI. Section VII concludes the paper and proposes the future research directions.

II. RELATED WORK

A. DL-Based COVID-19 Detection

With the rapid growth of computational power, DL has become a popular way to assist the diagnosis of X-ray or CT images [28]. The growing amount of publicly available datasets in different aspects also facilitates researches to implement deep neural networks on different tasks. For COVID-19, there are two kinds of data, which are CT images and X-ray images. The difference between CT and X-ray is CT is a 3D format, which contains the textural information of a part of the human body composed of many slices of body cross sections images. X-ray is a 2D format contains overlapping textual information of the human body. There are several researches using deep learning on X-ray datasets [29]–[31]. Ghoshal et al. [29] achieved 88.39% accuracy on their X-ray dataset using the DL model. While Narin et al. [31] achieved 98% accuracy on a smaller X-ray dataset. Experiments using CT datasets are more popular compared with X-rays. There are two kinds of CT datasets, 2D and 3D, used for deep learning classification for COVID-19. According to [32], most of existing studies focus on a single representative slice from a CT scan volume for COVID-19 detection [33]–[37]. However, since 3D volumes contains more information, which is CT's advantage by design, more experiments choose to use 3D CT volume to do classification and segmentation tasks [22]–[24], [38], [39], in which Zheng et al. [38], Li et al. [39], Morozov et al. [23], Zhang et al. [22] designed 3D convolutional networks to analyze 3D CT volumes.

B. Neural Architecture Search

Recently, there is a growing interest in the NAS technique, as it has been applied to many areas and outperformed human-designed models [40], [41]. Arguably, the studies of [5], [7] mark the beginning of NAS, as they demonstrated that RL-based NAS methods could effectively discover good architectures. ENAS [6] accelerates the search process by adopting a parameter-sharing strategy, in which all child architectures are regarded as sub-graph of a super-net; this enables these architectures to share parameters, obviating the need to train each child model from scratch. Besides RL-based methods, several improved methods are also proposed to further improve NAS efficiency.

SMBO methods evaluate the searched models with the surrogate function instead of metrics from trained architectures and thus shorten the search time. Furthermore, Liu et al. [12]
used learned a surrogate model to guide the search, and their method is five times more efficient than the RL-based method [7].

Liu et al. [9] were one of the first to propose the GD-based method, namely DARTS, which uses the softmax function to relax the discrete search space and significantly improve the search efficiency. But according to Liang et al. [10], the performance of DARTS is often observed to collapse, as DARTS tends to select the simpler operations (e.g., skip-connect) in the later stage of the search, which may result in a lack of diversity in the searched models.

Evolutionary algorithm (EA) is inspired by biological evolution. A new model (also known as individual) is evolved from a previous model with operations including selection, crossover and mutation. The early EA-based methods are compute-intensive, e.g., AmoebaNet [16] took 450 GPUs and 3,150 GPU days for searching. CARS [17] significantly improves the search efficiency by introducing the weight sharing strategy into the evolutionary algorithm. MoreNAS [18] combines RL and EA to obtain promising architecture for multi-objective optimal. LemonadeNAS [42] encodes network by function, and each network can be generated from network morphism operators.

C. Multi-objective Neural Architecture Search

In terms of the multi-objective tasks, the target will be complex, and hard to weigh different objectives. Some existing methods try to reduce multi-objective to single-objective. For example, MONAS [19] maps multi-objective to single by a linear combination, but it may lead to suboptimal. Since it’s tricky for one single network to surpass all the optimal target, architectures satisfied with the Pareto front are preferred. Yant et al. [17] introduced an improved method based on NSGA-III [21], namely pNSGA, to achieve optimal architectures with multi-objective. Besides, most multi-objective NAS methods [8], [11], [18], [19] only considered accuracy and model size as objectives. In this paper, we propose a new objective, namely potential, which is of great benefit to improve the robustness of the search process.

III. SEARCH SPACE

A well-designed search space is of great benefit to enhance the final model performance. The traditional cell-based search space [6], [9] has several problems: 1) the cell structure is inefficient for reference as it is a non-regularized combination of candidate operations; 2) the final model is constructed by repeating the searched cell, thus lacking diversity. To this end, we adopt the idea that is factorizing each network into cells and blocks [8], as shown in Fig. 1. The details of our search space are introduced as follows.

A. Block

Each block is a searchable module, which can be selected from a predefined number of candidate operations. To find a lightweight and high-quality 3D model for COVID-19 detection, we add a series of mobile inverted bottleneck convolution (MBConv) [20] into the candidate operation set. As shown in Fig. 2, MBConv3_3 comprises three sub-modules: 1) a 3D...
Therefore, this design paradigm enables model diversity.

D. Weight-sharing

Although EA can effectively solve the multi-objective problem that other optimization algorithms struggle to solve, it usually suffers from huge computational resources consumption. Therefore, inspired by [6], [9], [17], we adopt a weight-sharing strategy to improve efficiency.

An individual architecture denoted by \( \mathcal{N}(\alpha) \) is sampled from the SuperNet \( \mathcal{N} \), where \( \alpha \) is a set of one-hot sequences that encodes the individual architecture. Each one-hot sequence is decoded as a candidate operation. For example, as shown in Fig. 1, \([0, 0, 0, 0, 0, 1]\) is decoded as the identity (skip-connect) operation. All individuals share the weights \( \mathcal{W}_i \) of the SuperNet, and the weights of the \( i \)-th individual are denoted by \( \mathcal{W}(\alpha) \).

With loss of the individual \( L(\alpha) = \mathcal{H}(\mathcal{N}(\alpha), X, Y) \) where \( \mathcal{H} \) is the loss function, \( X \) is the input data and \( Y \) is the label data, the individual gradient \( \mathcal{W}(\alpha) \) can be calculated as

\[
d\mathcal{W}(\alpha) = \frac{\partial L(\alpha)}{\partial \mathcal{W}}. \tag{1}\n\]

Since the weights \( \mathcal{W}_i \) of the SuperNet is shared among all individual architectures, the gradient of \( \mathcal{W} \) can be calculated as the accumulation of gradients of all individuals.

\[
d\mathcal{W} = \frac{1}{P} \sum_{i=1}^{P} d\mathcal{W}(\alpha_i) = \frac{1}{P} \sum_{i=1}^{P} \frac{\partial L(\alpha_i)}{\partial \mathcal{W}} \tag{2}\n\]

where \( P \) is the size of population. In [17], the authors used a mini-batch architectures to obtain an unbiased approximation to Eq. 2 detailed as Eq. 5.

\[
d\mathcal{W} \approx \frac{1}{B} \sum_{i=1}^{B} d\mathcal{W}(\alpha_i) \tag{3}\n\]

where \( B \) is the number of individuals in a mini-batch and \( B < P \). In our experiments, we find that \( B = 1 \) works just fine, i.e., we can update \( \mathcal{W} \) using the gradient from any single individual sequentially sampled from the population.

IV. Search Algorithm

Our evolutionary search algorithm is based on NSGA-III [21], which is composed of selection, crossover, mutation, and update steps. Alg. 1 summarizes the detailed steps of our evolutionary algorithm for searching 3D neural architectures.
Algorithm 1: Efficient Multi-objective Evolutionary Algorithm for Neural Architecture Search

**Input:** SuperNet $\mathcal{N}$, SuperNet weight $\mathcal{W}$, population size $P$, population $\mathcal{A}$, selection size $K$, crossover probability $p_c$, mutation probability $p_m$, multi-objective $\mathcal{T} = \{T_1, ..., T_M\}$, loss function $\mathcal{H}$, training set $D_{\text{train}}$, validation set $D_{\text{val}}$

**Output:** a population of individual architectures $\{\mathcal{N}(\alpha_1), ..., \mathcal{N}(\alpha_P)\}$

1. $\mathcal{A}(0) = \{\alpha^{(0)}_1, ..., \alpha^{(0)}_P\} \leftarrow \text{Warm-up}(\mathcal{N}, \mathcal{P})$
2. for $e = 1:$ Evolve do
   3. for Mini-batch data $X$, label $Y$ in $D_{\text{train}}$ do
      4. Select $i$-th individual encoding $\alpha^{(e-1)}_i$
         (i ← $e \mod P$);
      5. Get $i$-th individual architecture $\mathcal{N}(\alpha^{(e-1)}_i)$;
      6. Calculate loss $L = \mathcal{H}(\mathcal{N}(\alpha^{(e-1)}_i), X, Y)$;
      7. Update individual weight $\mathcal{W}(\alpha_i)$;
   end
   // Start evolution
   9. $\mathcal{A}_{\text{keep}} \leftarrow \text{SelectTopK}(\mathcal{A}^{(e-1)}, K, \mathcal{T}, D_{\text{val}})$;
   10. $\mathcal{A}_{\text{new}} \leftarrow \{\}$
   while $|\mathcal{A}_{\text{new}}| < P - K$ do
      12. $p \leftarrow \text{random probability}$;
      13. if $p < 0.5$ then
         14.       $i, j (i \neq j) \leftarrow \text{GenerateRandomInteger}(K)$;
         15.       $\alpha \leftarrow \text{Crossover}(\mathcal{A}_{\text{keep}}[i], \mathcal{A}_{\text{keep}}[j], p_c)$;
         16.       $\alpha \leftarrow \text{Mutation}(\alpha, p_m)$;
      else
         17.       $\alpha \leftarrow \text{RandomSamplingIndividual}()$;
      if $\alpha$ not in $\mathcal{A}_{\text{new}}$ then
         19.       Append $\alpha$ to $\mathcal{A}_{\text{new}}$;
      end
   end
   22. Update population $\mathcal{A}(e) \leftarrow \mathcal{A}_{\text{keep}} \cup \mathcal{A}_{\text{new}}$;
end

A. Warm-up stage

In our experiment, the SuperNet weights $\mathcal{W}$ are randomly initialized and shared among all individuals; if we evolve from the beginning, then the first set of sampled architectures can get more training. In this case, these architectures may dominate in the later stage and compromise the search’s effectiveness. Similar to [17], [46], we adopt uniform sampling to treat all individual architectures equally during the warm-up stage. In our experiments, all searchable blocks are selected from eight different operations, and each operation is sampled with a probability of $\frac{1}{8}$. After the warm-up stage, many individuals are sampled and trained, and the top $P$ best-performing individual architectures are collected as the initial population for evolution.

B. Selection

As Alg. [1] illustrates, all individuals from the population $\mathcal{A}$ are equally trained for batches before the selection step. Then, top $K$ best-performing individuals are selected for the following evolution steps. The selection process allows us to preserve strong individuals while eliminating weak ones. The most commonly used selection method is to select individuals based on their fitness, such as validation accuracy [14], [47]. Practically, we are not only concerned with model accuracy, but also with other metrics such as model size. We use NSGA-III [21] to select promising individuals along the Pareto front of multi-objective.

In practice, we denote $\{\mathcal{N}(\alpha_1), ..., \mathcal{N}(\alpha_P)\}$ as a population of individual architectures and $\mathcal{T} = \{T_1, ..., T_M\}$ as multi-objective. We want to minimize the number of architectures by replacing some architecture with ones dominating them. In $\{\mathcal{N}(\alpha_1), ..., \mathcal{N}(\alpha_P)\}$, $\mathcal{N}(\alpha_i)$ dominates $\mathcal{N}(\alpha_j)$ when $\mathcal{N}(\alpha_i)$ are not worse than $\mathcal{N}(\alpha_j)$ in each metrics of multi-objective.

Formally:

$$T_k(\mathcal{N}(\alpha_i)) \geq T_k(\mathcal{N}(\alpha_j)) \quad \forall k \in \{1, ..., M\}$$

$$T_k(\mathcal{N}(\alpha_i)) > T_k(\mathcal{N}(\alpha_j)) \quad \forall k \in \{1, ..., M\}$$

From the above explanation, we can guarantee $\mathcal{N}(\alpha_i)$ must have at least one metric better than $\mathcal{N}(\alpha_j)$ with others metric at least the same. Thus, we can replace $\mathcal{N}(\alpha_j)$ with $\mathcal{N}(\alpha_i)$ and ensure measurement increasing at the process of evolution.

Most existing multi-objective NAS methods [8], [11], [18], [19] only considered the accuracy and model size, which may cause Matthew Effect, because models with relatively more training tend to achieve higher validation accuracy and thus are more likely to be trained, while other models may therefore lose the opportunity to compete. Therefore, we propose a new objective, namely potential ($\mathcal{P}$), which predicts the individual performance by incorporating the individuals’ history performance. The individual potential is represented by the slope of the line after applying the linear fitting to the individual’s history performance. The potential of one individual is derived as follows

$$\mathcal{P} = \begin{cases} Y \\ (X^TX)^{-1}X^TY \end{cases}$$

The potential is the individual potential, $X \in \mathbb{R}^{S \times 1}$ stores the epoch index when the individual is sampled, $Y \in \mathbb{R}^{S \times 1}$ indicates the corresponding validation accuracy, and $S$ represents the number of times the individual is sampled.

In our experiments, we consider three objectives: accuracy, model size, and potential. Therefore, we can generate three Pareto stages by applying the non-dominated sorting algorithm to each objective. Then we merge three Pareto stages to get the final Pareto stage.

C. Crossover

After selection, $K$ best-performing individuals are selected for the crossover, which exchanges architecture encodings between two different parent individuals to result in recombiant individuals. Since we use the one-hot encoding to represent the categorical candidate operations, crossovers are performed on the one-hot encodings other than binary encodings. Each pair of one-hot encodings of two parent individuals are crossed over with a probability of $p_c$. Fig. [5] (a) presents an example that only one crossover occurs between two parent individuals, both of which consist of three one-hot encodings.
We apply transforma-

to 512

to verify its transferability. In order to provide a more thorough

for exploitation, while mutations are usually for

o-encoding. In Fig. 3 (b), the second one-hot encoding of the

of the three datasets differently. For the MosMedData
data consists of 40 slices, and a slice

and MosMedData, the image format is PNG (portable network

each patient may have several CT scans, and each scan data is

We conducted multiple search experiments on the Clean-

to scans, including resize, center-crop, and normalization. For the training set, we randomly perform the horizontal and

(D) Mutation

Since crossovers are performed between two promising

pop randomly for exploitation, while mutations are usually for

Fig. 3. Examples of crossover and mutation. The basic unit for both crossover

The length of the one-hot sequence indicates the total number of
candidate operations (here 6).

V. EXPERIMENTAL IMPLEMENTATION

In this section, we first describe three datasets used in our

A. Datasets

In this paper, we use three publicly available datasets:

Clean-CC-CCII consists of three classes: NCP (novel coronavirus pneumonia),

Besides using the weight-sharing strategy, we reduce the

resolution to 64×64 to improve search

Efficiency. The training set is divided into the sub-training set

Valid. To avoid the Matthew

effect, we performed the warm-up stage before the evolution

and initialized the population with 20. The SuperNet weights

are optimized using the stochastic gradient descent (SGD) optimizer with a momentum of 3e-4. The initial learning rate

is cross-entropy.

B. Baselines

In our experiments, we use three hand-crafted 3D neural

architectures as the baseline models: DenseNet3D121, ResNet3D101, and MC3_18. We apply transformations to scans, including resize, center-crop, and normalization. For the training set, we randomly perform the horizontal and vertical flip operation. We use the Adam optimizer with the weight decay of 5e-4. The learning rate is initialized to 0.001. The cosine annealing scheduler is applied to adjust the learning rate. Three baseline models are trained for 200 epochs. The loss function $H$ is cross-entropy.

C. Evolutionary Multi-objective Neural Architecture Search

EMARS includes two stages: the search stage and the

retraining stage. The experimental configuration of each stage is as follows:

1) Search stage: The SuperNet comprises six cells, and

the number of searchable blocks in each cell is $[4,4,4,4,4,1]$. Each block is selected from eight candidate operations (see Section III-C). The blocks within the same cell keep the same number of input and output channels, and all blocks have a stride of 1. In other words, the spatial dimensions of the output features of each cell are determined by the calibration block. Here, we empirically set the number of output channels of each cell to $[24,40,80,96,192,320]$, and the stride of calibration block in each cell to $[2,2,2,1,2,1]$. The stem block is a Conv3D-BN3D-ReLU6 sequential module, with the number of output channels fixed to 32.

Besides using the weight-sharing strategy, we reduce the

resolution of input scan data to 64×64 to improve search
efficiency. The training set is divided into the sub-training set

$D_{\text{train}}$, and the validation set $D_{\text{val}}$. To avoid the Matthew

effect, we performed the warm-up stage before the evolution

and initialized the population with 20. The SuperNet weights

are optimized using the stochastic gradient descent (SGD) optimizer with a momentum of 3e-4. The initial learning rate

is 0.001.

Selection. For each epoch in the evolution process, all individual architectures from the population were equally trained with the training data batches before the selection. Using the NSGA-III algorithm, we evaluate the impact of different objectives on search results, including accuracy, potential, and

| Dataset | [Input size] | Classes | #Patients | #Scans |
|---------|--------------|---------|-----------|--------|
| Clean-CC-CCII | [128×128] | NCP | 726 | 190 | 1213 | 302 |
| | | CP | 778 | 186 | 1210 | 303 |
| | | Normal | 660 | 158 | 772 | 193 |
| | | Total | 2164 | 534 | 3195 | 798 |
| MosMedData | [256×256] | NCP | 601 | 255 | 601 | 255 |
| | | Normal | 178 | 76 | 178 | 76 |
| | | Total | 779 | 331 | 779 | 331 |
| COVID-CTset | [512×512] | NCP | 202 | 42 | 202 | 42 |
| | | Normal | 200 | 82 | 200 | 82 |
| | | Total | 402 | 124 | 402 | 124 |
model size. Each experiment is conducted on four Nvidia Tesla V100 GPUs (the 32GB PCIe version) and can be finished in about 8 hours. We set selection size \( K \) in Alg. 1 to 10, which indicates that we preserved 10 most promising individuals for exploitation and generated 10 new individuals for exploration.

**Crossover & Mutation.** As shown in Alg. 1 after the selection, we first generated a random probability \( p \in (0, 1) \). If \( p > 0.5 \), we randomly sampled a new individual; otherwise, we performed crossover and mutation on the selected individuals to generate a new individual. The basic unit for both crossover and mutation is the one-hot encoding. The probability of crossover and mutation for each one-hot encoding is \( p_c = 0.3 \) and \( p_m = 0.2 \), respectively.

2) Retraining stage: After the search stage, we export top-10 promising individual architecture along Pareto front of the search stage’s objectives. We train each exported individual for a few epochs and finally choose the best-performing one for further retraining. The experimental configuration of retraining is the same as the baseline experiment.

**VI. Results & Analysis**

In this section, we first introduce our evaluation metrics. Then, we present and analyze the experimental results.

**A. Evaluation Metrics**

We use several commonly used evaluation metrics to compare the model performance, as follows:

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (6)
\]

\[
\text{Sensitivity (Recall)} = \frac{TP}{TP + FN} \quad (7)
\]

\[
\text{F1-score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \quad (8)
\]

\[
\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP} \quad (9)
\]

To be noticed, the positive and negative cases are assigned to the COVID-19 class and the non-COVID-19 class, respectively. Specifically, \( TP \) and \( TN \) indicate the number of correctly classified COVID-19 (i.e., NCP) and non-COVID-19 (i.e., CP and Normal) scans, respectively. \( FN \) and \( FP \) indicate the number of wrongly classified COVID-19 and non-COVID-19 scans, respectively. The accuracy is the micro-averaging value for all test data to evaluate the overall performance of the model. Besides, we also use model size as an evaluation metric to compare the model efficiency.

**B. Model Size-aware Search**

To verify the ability of multi-objective optimization of the EMARS algorithm, we set up three experiments with different model size objectives. The three experiments run for 100 epochs. We visualized the search results of three experiments in Fig. 4 in which each point indicates the result of one individual. We split 100 epochs into two parts: the purple points represent the evolution results of the first 50 epochs, and the yellow points represent the evolution results of the last 50 epochs. Fig. 4 shows the distribution of evolution with the objectives of (a) only the validation accuracy, (b) the validation accuracy and large model size, (c) the validation accuracy and small model size, respectively. We can see that the results of the three experiments are in line with expectations. The yellow points in Fig. 4(a) cover a wider range over the model size.
Fig. 5. Comparison among the evolution results of different experiments with or without the potential objective. Each point indicates the results of one individual and is divided into first or second half of epochs. Solid line indicates the average accuracy of individuals. Dashed line indicates the 30/70 percentile of individual distribution. The 30/70 percentile in Fig. 5(a) is relatively higher than that of Fig. 5(b), but the 30/70 percentile of Fig. 5(b) is closer than of Fig. 5(a), which indicates that potential can effectively improve search process’ robustness. Fig. 5(c) combines the advantages of accuracy, potential, and small model size objectives.

TABLE III

| Model   | Model size (MB) | Objectives                      | Accuracy (%) | Precision (%) | Sensitivity (%) | F1-score |
|---------|-----------------|--------------------------------|--------------|---------------|-----------------|----------|
| EMARS-A | 5.93            | Accuracy                       | 89.67        | 91.07         | 87.75           | 0.8958   |
| EMARS-D | 5.63            | Potential                      | 88.78        | 93.68         | 88.41           | 0.9097   |
| EMARS-E | 3.39            | Accuracy + potential + small model size | 89.61 | 98.16 | 88.41 | 0.9303 |

D. Performance Comparison between Baseline and EMARS on Clean-CC-CCII dataset

Table. IV summarizes the performance between three 3D baseline models and EMARS series models on the Clean-CC-CCII dataset. Our searched architectures cover an extensive range of model sizes, ranging from 3.39 MB to 20.61 MB. In other words, we can easily select a suitable architecture for different deploy devices. Besides, all EMARS series models outperform the baseline models with a much smaller model size. EMARS-A achieves the best accuracy of 89.67% among all models, and it surpasses ResNet3D101, DenseNet3D121, and MC3\_18 by 4.83\%, 3.05\%, and 4.07\%, respectively. EMARS-C achieves the best sensitivity compared with other models. EMARS-E achieves the best precision and f1-score and has the smallest size (3.39 MB), which is 98.96\%, 92.13\%, and 92.27\% smaller than ResNet3D101, DenseNet3D121, and MC3\_18.

F. Interpretability

CAM is an algorithm that can visualize the regions that the model focuses on, and hence provide the interpretability for our searched models. We apply it to a 3D CT scan volume...
from the Clean-CC-CCII dataset using EMARS-E model. Fig. 6 presents the generated heat maps of some slices. A red and brighter region means that it have a larger impact on the model’s decision to classify it as COVID-19.

From the perspective of the scan volume, we can see that some slices have more impacts on the model’s decision than the others. In terms of a single slice, the areas that EMARS-E focuses on has ground-glass opacity, which is proved a powerful and promising solution for assisting in COVID-19 detection. In the future, we will apply our EMARS framework to more complex tasks, such as 3D medical image segmentation.

VII. Conclusion

In this work, we propose a factorized 3D search space, in which all child architectures share weights among each other. We introduce an efficient evolutionary multi-objective neural architecture search (EMARS) framework to search for 3D models for COVID-19 CT scan classification. We also propose a new objective, namely potential, that can effectively improve the robustness of the search process. The results on three COVID-19 datasets show that a series of models searched by EMARS cover a wide range over the model size, and they all outperform the baseline models on the Clean-CC-CCII dataset. We also verify the EMARS series models’ transferability by training two representative models on the MosMedData and Covid-CTset datasets. Our work demonstrates that NAS is a powerful and promising solution for assisting in COVID-19 CT scan detection. In the future, we will apply our EMARS framework to more complex tasks, such as 3D medical image segmentation.

TABLE IV

| Model       | Model size (MB) | Search Method | Search Cost (GPU days*) | Accuracy (%) | Precision (%) | Sensitivity (%) | F1-score |
|-------------|-----------------|---------------|-------------------------|--------------|---------------|----------------|----------|
| ResNet3D101 | 325.21          | manual        | 85.54                   | 90.62        | 87.15         | 0.8292         |
| DenseNet3D121 | 43.06          | manual        | -                       | 87.02        | 89.97         | 0.8728         | 0.8576   |
| MC3_18      | 43.84           | manual        | -                       | 86.16        | 87.11         | 0.8278         | 0.8489   |
| EMARS-A     | 5.93            | evolution     | 1.3                     | **89.67**    | 91.07         | 87.75          | 0.8938   |
| EMARS-B     | 20.61           | evolution     | 1.3                     | 87.92        | 95.80         | 83.11          | 0.9091   |
| EMARS-C     | 3.79            | evolution     | 1.3                     | 89.39        | 93.13         | **89.74**      | 0.9141   |
| EMARS-D     | 5.63            | evolution     | 1.3                     | 88.78        | 93.68         | 88.41          | 0.9007   |
| EMARS-E     | **3.39**        | evolution     | 1.3                     | **89.61**    | **98.16**     | 88.41          | **0.9303** |

TABLE V

| Dataset     | Model      | Model size (MB) | Accuracy (%) |
|-------------|------------|-----------------|--------------|
| MosMedData  | ResNet3D101 | 325.21          | 81.82        |
|             | DenseNet3D121 | 43.06          | 79.55        |
|             | MC3_18        | 43.84           | 80.4         |
|             | EMARS-B       | 20.61           | 81.51        |
|             | EMARS-C       | 3.79            | 92.09        |
|             | EMARS-D       | 5.63            | 92.86        |
|             | EMARS-E       | **3.39**        | **92.86**    |
| Covid-CTset | ResNet3D101  | 325.21          | 93.87        |
|             | DenseNet3D121 | 43.06          | 91.91        |
|             | MC3_18        | 43.84           | 92.57        |
|             | EMARS-B       | 20.61           | 92.09        |
|             | EMARS-E       | 3.39            | 92.86        |

REFERENCES

[1] I. Smyrlaki, M. Ekman, A. Lentini, N. R. de Sousa, N. Papanicolaou, M. Vondracek, J. Aarum, H. Safari, S. Muradrasoli, A. G. Rothfuchs, J. Albert, B. Högborg, and B. Reinius, “Massive and rapid COVID-19 testing is feasible by extraction-free SARS-CoV-2 RT-PCR,” Nature Communications, vol. 11, no. 1, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41467-020-18611-5

[2] X. He, S. Wang, S. Shi, X. Chu, J. Tang, X. Liu, C. Yan, J. Zhang, and G. Ding, “Benchmarking deep learning models and automated model design for covid-19 detection with chest ct scans,” medRxiv. 2020. [Online]. Available: https://www.medrxiv.org/content/early/2020/06/08/20125963

[3] J. Dinnes, J. J. Deeks, A. Adrian, S. Berhane, C. Davenport, S. Dittrich, D. Emperor, Y. Takwoingi, J. Cunningham, S. Beece, J. Dretzke, L. F. di Ruffano, I. M. Harris, M. J. Price, S. Taylor-Phillips, L. Hooft, M. M. Leeflang, R. Spijker, and A. V. den Bruel and, “Rapid, point-of-care antigen and molecular-based tests for diagnosis of SARS-CoV-2 infection,” Cochrane Database of Systematic Reviews, Aug. 2020. [Online]. Available: https://doi.org/10.1002/14651858.cd013705

[4] B. Xu, Y. Xing, J. Peng, Z. Zheng, W. Tang, Y. Sun, C. Xu, and F. Peng, “Chest CT for detecting COVID-19: a systematic review and meta-analysis of diagnostic accuracy,” European Radiology, vol. 30, no. 10, pp. 5720–5727, May 2020. [Online]. Available: https://doi.org/10.1007/s00330-020-06934-2

[5] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” arXiv preprint arXiv:1611.01578, 2016.

[6] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, “Efficient neural architecture search via parameter sharing,” arXiv preprint arXiv:1802.03268, 2018.

[7] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.
