Medical Image Analysis 54 (2019) 306–315

Contents lists available at ScienceDirect

Medical Image Analysis

journal homepage: www.elsevier.com/locate/media

Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising

Peng Liu[a], Mohammad D El Basha[a], Yangjunyi Li[a], Yao Xiao[a], Pina C. Sanelli[b,c,d], Ruogu Fang[a,*]

[a] *J. Crayton Pruitt Family Dept. of Biomedical Engineering, University of Florida, 1275 Center Drive, Gainesville, FL 32611 USA*
[b] *Imaging Clinical Effectiveness and Outcomes Research, Department of Radiology, Northwell Health, Northwell Health, 300 Community Drive, Manhasset, NY 11030 USA*
[c] *Donald and Barbara Zucker School of Medicine at Hofstra/Northwell, 500 Hofstra Blvd, Hempstead, NY 11549 USA*
[d] *Center for Health Innovations and Outcomes Research, Feinstein Institute for Medical Research, 350 Community Dr, Manhasset, NY 11030 USA*

ARTICLE INFO

Article history:
Received 1 October 2018
Revised 30 January 2019
Accepted 20 March 2019
Available online 21 March 2019

MSC:
41A05
41A10
65D05
65D17

Keywords:
Medical image denoising
Genetic algorithm
Convolutional neural networks

ABSTRACT

Deep convolutional neural networks offer state-of-the-art performance for medical image analysis. However, their architectures are manually designed for particular problems. On the one hand, a manual designing process requires many trials to tune a large number of hyperparameters and is thus quite a time-consuming task. On the other hand, the fittest hyperparameters that can adapt to source data properties (e.g., sparsity, noisy features) are not able to be quickly identified for target data properties. For instance, the realistic noise in medical images is usually mixed and complicated, and sometimes unknown, leading to challenges in applying existing methods directly and creating effective denoising neural networks easily. In this paper, we present a Genetic Algorithm (GA)-based network evolution approach to search for the fittest genes to optimize network structures automatically. We expedite the evolutionary process through an experience-based greedy exploration strategy and transfer learning. Our evolutionary algorithm procedure has flexibility, which allows taking advantage of current state-of-the-art modules (e.g., residual blocks) to search for promising neural networks. We evaluate our framework on a classic medical image analysis task: denoising. The experimental results on computed tomography perfusion (CTP) image denoising demonstrate the capability of the method to select the fittest genes for building high-performance networks, named EvoNets. Our results outperform state-of-the-art methods consistently at various noise levels.

1. Introduction

Deep convolutional neural networks (CNNs) have recently achieved notable success in many medical image analysis tasks, such as image denoising (Li et al., 2012; Wang and Zhou, 2006; Whitaker and Xue, 2001), lesion detection (Parikh et al., 2008), segmentation (Pham et al., 2000; Van Leemput et al., 2001), and classification (Van den Elsen et al., 1993; Li et al., 2014). These tasks are important for disease diagnosis and treatment planning. The success of deep CNNs is largely due to manual design of an effective feature extraction component (block) (e.g., inception (Szegedy et al., 2015)) and the tuning of a large number of hyperparameters, such as the number of layers, the number of filters, the type of nonlinear activation functions, and the choice of optimizer functions. Manually optimizing a CNN may take days or even weeks, which depends on the network scale and the training data properties (e.g., sparsity, noisy features).

Medical imaging techniques, such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), and X-rays are popular diagnostic tools. Nevertheless, these techniques are susceptible to noise. For example, CT perfusion images are often associated with complicated mixed noise due to the photon starvation artifacts. Removing these artifacts from training data can improve the learning capability of the CNN-based models and thus boost the lesion detection and classification accuracy. In the past decades, different methods have been widely investigated to solve the problem. Based on the image properties, we can classify the existing methods to be prior based (i.e., PCLR (Chen et al., 2015)), sparse coding-based (i.e., KSVD (Elad and Aharon, 2006)), low rank-based (i.e., WNNM (Gu et al., 2014)), and filter-based approaches (e.g., BM3D (Dabov et al., 2009)). However, complicated mixed noise in medical images still leads to the unsatisfactory

* Corresponding author.
E-mail address: ruogu.fang@bme.ufl.edu (R. Fang).

https://doi.org/10.1016/j.media.2019.03.004
1361-8415/© 2019 Elsevier B.V. All rights reserved.

performance of these methods and remains a valuable research direction.

CNNs recently have shown superior performance over traditional models on denoising tasks. A typical CNN is composed of several stacked layers, including layer connections and hyperparameters (e.g., the number of layers, the number of neurons in each layer, the choice of activation functions). RED-Net (Mao et al., 2016) consists of a chain of 30 convolutional layers and symmetric deconvolutional layers. Another state-of-the-art method, DnCNN (Zhang et al., 2017), adopts concise stacked-layer connections but achieves impressive performance via appropriate hyperparameters (e.g., ReLU (Nair and Hinton, 2010), Adam (Kingma and Ba, 2014)) selection. Consequently, hyperparameters play a dominant role in optimizing image denoising tasks.

These modern networks present promising image restoration performance; however, they are all manually designed based on expert empirical knowledge. It is expensive and slow to manually search for the optimal network structures with exponential combinations of hyperparameters and layers connections. To address this issue, we need to build a framework that can automatically construct promising CNN-based denoisers with concise layer connections and optimal hyperparameter combinations. Moreover, an efficient algorithm is essential to explore the optimal CNN structures within reasonable computational time.

In this work, we propose a neural network evolutionary framework and implement it by constructing a CNN-based medical image denoiser, named EvoNet, automatically. To more effectively navigate large search spaces, we formulate an optimized genetic algorithm (GA) (Holland, 1992). GA initializes candidate solutions (e.g., networks) as an initial generation, and then applies genetic operations to evolve the solutions in each generation. As shown in Fig. 1, for the population evolution process, we define three standard genetic operations: selection, crossover, and mutation. A fitness function is formulated to select the best individuals (e.g., CNN) in each generation. Fitness scores evaluate each of these solutions through a criterion of image quality (e.g., Peak-Signal-to-Noise Ratio (PSNR), structural similarity index measure (SSIM) (Hore and Ziou, 2010)).

“Genes” are at the core of a bio-evolutionary strategy as well as a CNN-evolutionary strategy. Theoretically, we can take any ingredient as a “gene”, such as the number of layers, the choice of activation functions, the type of learning strategy (e.g., residual), and the choice of loss functions, as long as it has the potential to help a CNN achieve satisfactory performance. Therefore, what genes to choose and how to organize them are important for quickly constructing a CNN. On the one hand, we promote the fittest genes that have been evaluated in practice (e.g., last generation) to a fine-gene set, which has the higher priority to be chosen in the following generation. We name this strategy to be experienced-based greedy initialization. On the other hand, we expedite evolutionary speed by dynamically decreasing the number of individuals (i.e., CNNs) in our population and the number of generations. Ultimately, most of the genes are eliminated, and only a few of them will survive.

Efficiency is still a challenge of using GA in large scale training data. GA typically has to face a massive search space. Although the strategies above can relieve the issue, the evolutionary process, built on extensive training data, is still quite slow. Transfer learning can overcome this challenge. A CNN is typically designed for a specific problem on a particular dataset. A subset of that data has similar properties to the original large dataset. Therefore, the fittest genes learned from the small sub-dataset should be transferable to the original large dataset. Eventually, the efficiency issue can be solved through transfer learning, particularly, by switching evolutionary processes from a small sub-dataset to the corresponding original large dataset.

The contributions of the paper are as follows:

- It is the first time to propose a GA-based method to construct CNN structures for medical image analysis automatically. This evolution approach provides the flexibility to optimize both CNN hyperparameters and network structures.
- We optimize the standard genetic algorithm to speed up the evolutionary process. Specifically, we use an experience based greedy strategy on the initialization stage to enrich high-performance genes and suppress the defective ones in the following generation. Besides, we select an appropriate mutation rate to make a trade-off between the diversity of the population and convergence of optimum generation.
- We dynamically update hyperparameter sets to make the architectures of the population transferable between datasets of different sizes. Particularly, we split all possible hyperparameters into fine-genes and complementary-genes for initialization and mutation, respectively. The learned fine-genes from a small dataset are transferable to construct a satisfied neural network quickly. Transfer learning further accelerates the evolution rate of the optimization.
- The proposed framework can flexibly treat learning strategies (e.g., residual learning (He et al., 2016)), loss functions (e.g., mean square error (MSE)), and the various network blocks (e.g., residual (He et al., 2016), interception (Szegedy et al., 2015)) as the genes in the evolutionary process to obtain state-of-the-art performance.
- The proposed framework also can be used for exploring new network blocks, optimizers, and loss functions automatically. In other words, GA can not only be used for exploring existing “genes” but also for creating new “genes”.

This work is an extension of our conference paper “Neural Network Evolution Using Expedited Genetic Algorithm for Medical Image Denoising” (Liu et al., 2018) published on Medical Image Computing and Computer Assisted Intervention (MICCAI) 2018. The main difference of this extension is we explored our framework’s flexibility. In particular, the genes can represent not only the hyperparameters but also the type of structural network blocks, such as the residual block. By embedding current state-of-the-art network blocks into the evolutionary processing, the explored neural networks outperform state-of-the-art methods.

2. Related work

2.1. Genetic algorithm

Genetic Algorithms (GAs) (Holland, 1992) belong to evolutionary algorithms and are inspired by the natural biological evolution. Typically, a GA is composed of a “population” *P* of *N* “individuals”, and has operations including initialization, individual selection, parents crossover, and children mutation (see Fig. 1). A sequence of operations is called an evolutionary “generation”. The competition among individuals is simulated by a *fitness* function that selects the fittest individuals over the weaker ones. During the population evolution process, all individuals enter an iterative competition, where a new population is evolved in each generation, consisting of the survivors, children generated from the crossover of survivors, and mutated individuals from the children.

GA is one of the heuristic algorithms for combinatorial optimization (Youssef et al., 2001). Simulated annealing (SA) and Tabu search (TS) are two other similar algorithms. On the one hand, all of them can be applied to many combinatorial optimization problems. On the other hand, they also have distinct properties. Firstly, GA requires the greatest computational cost to find the best solution. Secondly, the quality of the best solution obtained by GA is better than SA and comparable to TS (Youssef et al., 2001). More

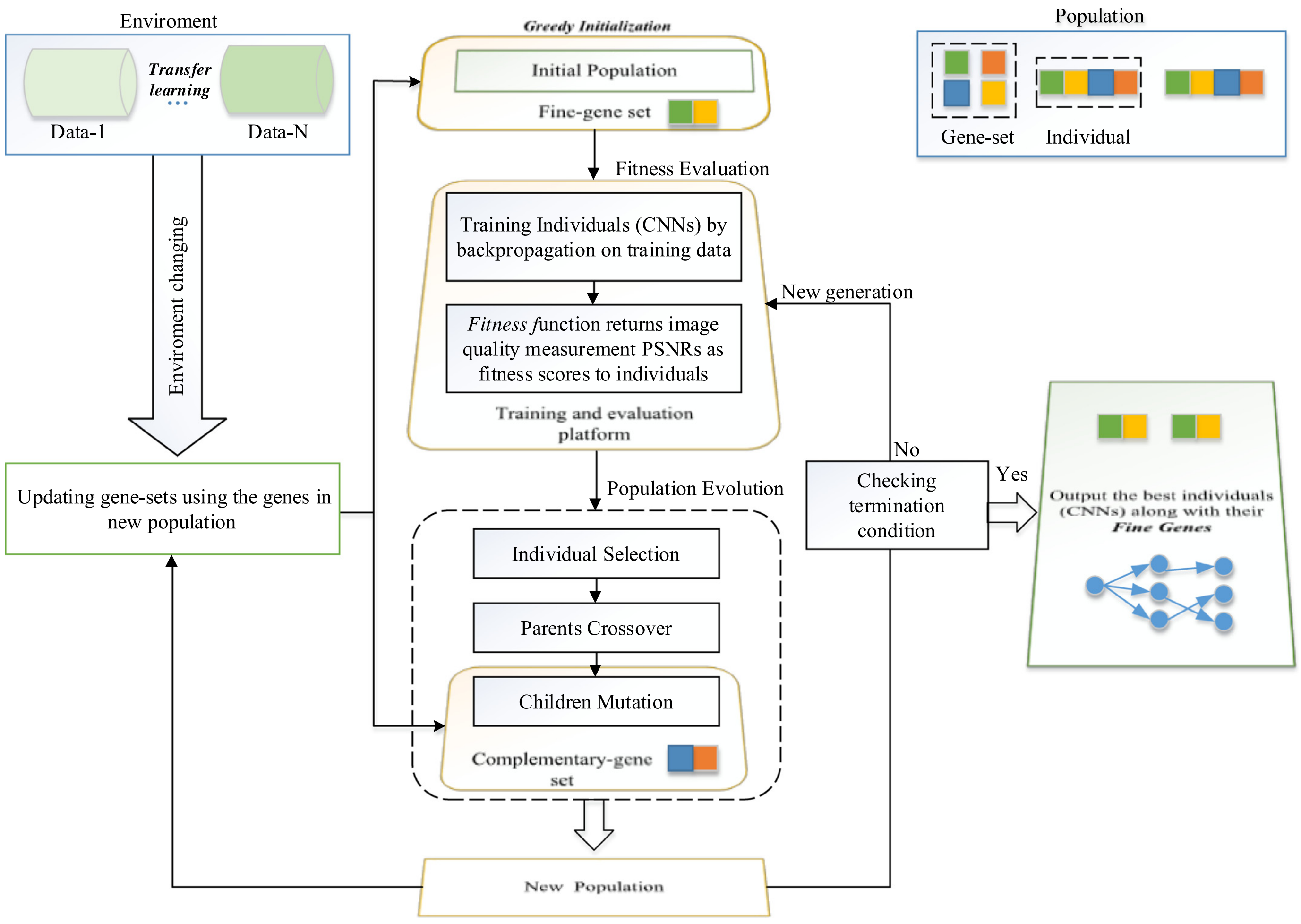
Enviroment
Transfer learning
Data-1
Data-N
Enviroment changing
Updating gene-sets using the genes in new population
Greedy Initialization
Initial Population
Fine-gene set
Fitness Evaluation
Training Individuals (CNNs) by backpropagation on training data
Fitness function returns image quality measurement PSNRs as fitness scores to individuals
Training and evaluation platform
Population Evolution
Individual Selection
Parents Crossover
Children Mutation
Complementary-gene set
New Population
New generation
No
Checking termination condition
Yes
Population
Gene-set
Individual
Output the best individuals (CNNs) along with their Fine Genes

Fig. 1. Overview of the proposed method composing of fitness evaluation and population evolution. The CNN architecture is trained on medical images using a fitness score. The individual networks labeled with the fitness scores are sent to individual selection. The survived individuals are presented as parents for crossover and mutation.

importantly, GA can incorporate domain-specific knowledge in all optimization or combinatorial phases to dictate the search strategy, but SA and TS lack such a feature. Neural network combination requires incorporating many constraints (domain knowledge) that are critical to a search strategy. Therefore, we choose GA over other heuristic algorithms for neural network evolution.

GA has been widely utilized as a heuristic search (Geem et al., 2001), an optimization technique (Horn et al., 1994) and also has been applied in machine learning approaches (Goldberg and Holland, 1988), function optimization (Goldberg et al., 1987), and feature selection (Yang and Honavar, 1998). Recently, Xie et al. Xie and Yuille (2017) and Suganuma et al. Suganuma et al. (2017) applied GA to explore CNN architectures automatically for image classification. These methods focus on exploring the structural modules and connections among layers. They first explore the representations of the networks with a binary coding scheme based on Cartesian genetic programming (CGP) (Miller and Turner, 2015) and then train the networks with back-propagation. However, the study of an efficient GA for automatically building a concise CNN-based model for image denoising is still lacking.

2.2. Neural network evolution

In the early 1990s, there were studies (Stanley and Miikkulainen, 2002; Yao, 1999) that attempted to combine evolutionary algorithms and artificial neural networks, and eventually formed the branch of Evolutionary Artificial Neural Networks (EANN), which is also part of Automated Machine Learning (AutoML) [1] Many AutoML studies (Klein et al., 2016; Feurer et al., 2015; Olson and Moore, 2016) have been proposed. However, most of them focus on the traditional machine learning algorithms (e.g., SVM (Cortes and Vapnik, 1995), Random Forest (Ho, 1995)) rather than deep CNN. Although Frank et al. Klein et al. (2016) included CNN in their Bayesian Optimization based AutoML framework, it focuses on finding several good hyperparameter settings on fixed CNN architectures, and not a flexible and general solution. Recently, there is more literature on EANN methods. Basically, we can classify them as evolutionary algorithm (e.g., GA) based (Xie and Yuille, 2017; Suganuma et al., 2017; Real et al., 2018) and reinforcement learning (RL) (Sutton and Barto, 1998) based (Zoph and Le, 2016; Real et al., 2017; Baker et al., 2016) methods.

Most of these methods focus on searching architecture combinations, such as the layer connections, the type of layers, and the way of connecting layers (e.g., summation, cascading). However, the performance of a CNN depends on not only a structural module (e.g., residual block, interception) but also a combination of hyperparameters (e.g., loss functions, optimizers, activation

[1] Automated Machine Learning provides methods and processes to make Machine Learning available for non-Machine Learning experts. AutoML includes (a) Preprocess and clean the data; (b) Select and construct appropriate features; (c) Select an appropriate model family; (d) Optimize model hyperparameters; (e) Postprocess machine learning models; (f) Critically analyze the results obtained (reference: http://www.ml4aad.org/automl/). In this study, we only focus on the (c)(d)(e).

functions). For example, Sajjadi et al. Sajjadi et al. (2016) proposes a single image super-resolution (SISR) method to achieve state-of-the-art performance by exploring a suitable combination of various loss functions (e.g., MSE, perceptual loss (Johnson et al., 2016), texture matching loss (Sajjadi et al., 2016), adversarial training loss (Goodfellow et al., 2014)). As more and more optimization methods are being proposed, a considerably large number of choices and combinations of various hyperparameters are increasing the difficulty of manually designing an effective CNN quickly. Thus, it is necessary to create a neural network evolution framework that can automatically explore not only a suitable structure module but also the fittest hyperparameter combination.

2.3. Computing resource constraint

Existing EANN frameworks take a few considerations of the accessible computing resource. Most of them focus on the method of creating competitive neural network architectures. Although they claim that their methods are scalable, only companies owning large-scale platforms can employ these methods (e.g., Google AutoML (Real et al., 2017)), and we cannot see them as a scalable solution in reality. Thus, we need a study to automate the process of designing deep learning models economically under an accessible computing resource, such as completing one automated process by using one or four GPUs within hours.

2.4. Transfer learning

Transfer learning is to develop methods that can transfer knowledge learned in one or more source tasks to improve learning in a related target task (Torrey and Shavlik, 2010). The transferable knowledge can be a trained model, a policy used, or hyperparameters learned in a source task. Few studies use transfer learning into neural network architecture searching. To the best of our knowledge, NASNet, recently proposed by Google Brain, is the earliest work adopting transfer learning to learn the best transferable architecture found on a small dataset by using a recurrent neural network (RNN) (Mikolov et al., 2010) and reinforcement learning.

There are three significant differences between our proposed framework and NASNet: (a) we adopt an evolutionary algorithm instead of RL; (b) our method is more flexible and able to treat not only network blocks but also hyperparameters, learning strategies, and loss functions as genes of evolution; (c) unlike NASNet, our work is scalable and can be implemented easily on accessible and economical computing resources.

3. Methodology

A concise but also promising CNN-based denoiser relies on a specific learning strategy (e.g., Residual learning) and one choice of hyperparameter combinations (e.g., DnCNN). Therefore, in this work, we aim at building a simple but effective CNN structure by focusing on exploring the effective combinations of CNN hyperparameters instead of the structural blocks and layer connections. Despite all this, our proposed framework is also flexible because “genes” can represent any component (e.g., block type, loss function, learning strategy) of a CNN. In our experiment Section 4, we provide two different groups of genes to evaluate our method’s flexibility in practice.

One significant challenge of using GA is how to accelerate the evolutionary process dynamically in a huge search space. Existing methods usually require large-scale computing resource (e.g., hundred GUPs). However, in practice, these computational platforms are not accessible to most users. To address this issue, in this section, we present an *Optimized Genetic Algorithm* (Algorithm 1) with an *Experience-based Greedy Exploration Strategy*, and also leverage *Transfer Learning* to further expedite the GA evolution on the large dataset.

Algorithm 1 The Proposed Genetic Algorithm for Exploring CNNs.

```
Require: one all-possible-gene set $\theta = \theta_c \cup \theta_f$, initial fine-gene set $\theta_f$, initial complementary-gene set $\theta_c$, initial population size $N$, initial number of generation $G$, percentage of selected individuals after each generation $\sigma$, number of children of crossed over out $O$, mutation rate $\epsilon$, termination condition $E$, small and large training datasets $D = \{D_s, D_l\}$
1:  for d = 1, 2, ..., length(D) do
2:    for g = 1, 2, ..., $G-1$ do
3:      for i = 1, 2, ..., $N$ do
4:        if $g = 1$ then
5:          Initialize a set of randomized individuals $\{P_i^g\}_{i=1}^N$ based on $\theta_f$
6:        end if
7:        Return trained individuals $\{P_i^{g,t}\}_{i=1}^N$ by Keras and Tensorflow
8:        Return fitness scores $F_i^g = F(P_i^{g,t})$ to individuals
9:      end for
10:     Sort $\left\{P_i^{g,trained}\right\}_{i=1}^N$ by $F_i^g$ with descending order
11:     $\Re = \left\{P_i^{g,trained,sorted}\right\}_{i=1}^{N*\sigma}$ Select the top $N*\sigma$ best individuals
12:     $P_i^{g,new} = P_i^{g,new} + = \varnothing$
13:     while $length(P_i^{g,new}) < N - length(\Re)$ do
14:       $\Upsilon_{mom}, \Upsilon_{dad} = uniformRandom(\Re)$ Select parents
15:       $\{\Psi_o\}_{o=1}^O = Genome(\Upsilon_{mom}, \Upsilon_{dad})$ Have children with crossover genes
16:       if $\epsilon > Random(0,1)$ then Mutation with a rate $\mu$
17:         $\Psi_{selected} = selectRandom(\{\Psi_o\}_{o=1}^O)$ Randomly select one child
18:         $\theta_{selected} = selectRandom(\theta_{\Psi_{selected}})$ Randomly select one gene
19:         $\theta_{c,selected} = selectRandom(\theta_c, Genotype(\theta_{selected}))$
20:         $\{\Psi_o^m\}_{o=1}^O = replace(\Psi_{selected}, \theta_{selected}, \theta_{c,selected})$
21:         $P_i^{g,new} = P_i^{g,new} + \{\Psi_o^m\}_{o=1}^O$
22:       else
23:         $P_i^{g,new} = P_i^{g,new} + \{\Psi_o\}_{o=1}^O$
24:       end if
25:       $P_i^{g,new} = removeDuplicate(P_i^{g,new})$
26:     end while
27:     $P_i^{g,new} = P_i^{g,new} + \Re$
28:     if $E = True$ then May say “the highest fitness score is not changing”
29:       Terminate generation and go to output
30:     end if
31:   end for
32:   $\theta_f^u = Update(\theta_f, \Re)$ Replace fine-gene set with the genes in $\Re$
33:   $\theta_c^u = \theta - \theta_f$ Update complementary-gene set
34: end for
Ensure: Select the best individuals (CNNs) from $P_i^{g,new}$
```

3.1. Gene splitting

A “gene” is the basic functional unit in a biological body. In an artificial neural network, genes represent the components of a CNN, such as hyperparameters, which includes the number of layers, the number of neurons, activation functions, optimizers, and loss functions. Genes can also represent structural blocks (e.g., residual block (He et al., 2016)) and learning strategies, such as residual learning and transductive learning (Joachims, 2003).

To speed up the evolution process, let θ be the set of all possible genes, and it is split into a fine-gene set θ_f and a complementary-gene set θ_c. Fine-genes are the hyperparameters selected from those state-of-the-art CNN structures in the literature (e.g., DnCNN) or previous GA generations. The remaining genes in θ are complementary genes. The first population is initialized based on θ_f. The mutation process is solely built upon θ_c. By combining this strategy with experience-based greedy exploration strategy (see 3.2), we can find the satisfied CNNs at the early evolutionary process stage.

Our method emphasizes the fittest genes more than the survived individuals (network structures). This strategy ensures the promising genes are passed down to offspring, and the fittest individuals are more likely to be explored effectively in early generations. Therefore, our approach can accelerate the evolution process via optimizing gene search space dynamically. The overview and algorithm details of the proposed method are shown in Fig. 1 and Algorithm 1 respectively. In our algorithm, an individual (CNN) is composed of different genes, and N individuals form a population-P.

3.2. Experience-based greedy exploration

We optimize GA with an experience-based greedy exploration strategy, which determines how and when to update gene sets. Experience represents CNN components (e.g., hyperparameters) learned from the last generation. We initialize the fine-gene sets θ_f with the genes from the top-performance CNNs evolved in the previous evolutionary environment. This strategy enables locating the best individuals at an early stage. Thus, it does not need to explore the entire search space. Our approach stores and transfers such experience to the next generation.

3.3. Transfer learning

Another novel contribution of our approach is using a *transfer learning* strategy (Yosinski et al., 2014) that allows the explored genes to be transferable among training data of different sizes or various modalities. For instance, we may use a small dataset to quickly optimize the gene-set space first, and then explore CNNs on a larger dataset by initializing a new population using the fine genes identified from the small dataset. Transfer learning further accelerates the evolution rate of the optimization.

3.4. Fitness evaluation

The *fitness* function $F(P_i)$ returns the restored image quality measure as a fitness score to each individual P_i. Fitness score performs the following functions: (1) evaluating individual fitness; (2) updating gene-sets; (3) serving as a stopping rule. Hence, the fitness function is critical for designing an effective GA-based method. Algorithm 1 presents the details of the proposed GA for exploring promising CNNs to handle medical image denoising.

3.5. Computational complexity

It is difficult to compute the computational complexity of an evolutionary metaheuristic. The complexity depends on the genetic operators (mutation, crossover, and selection), the implementation, which may have a very significant effect on overall complexity, the representation of the individuals and the population, and the fitness function. Generally, a genetic algorithms complexity is $O(GNE)$, where G the number of generations, N is the population size, and E is the size of the individuals which, in our case, is the number of epochs of training each neural network. The fitness function is not considered because it depends on a specific application. In our case, the computational complexity is the same as the original genetic algorithm because we did not change the internal mechanism of the genetic algorithm; rather, we applied transfer learning and greedy initialization strategy onto external of the genetic algorithm.

4. Experiments

4.1. Training and testing data

Our dataset is a collection of 10,775 cerebral perfusion CT images, all of which are 512×512 gray-scale images. Many of the collected images have very similar features due to the series of CT scans on the same patients. In order to avoid overfitting, we randomly select images to produce a diversified training dataset. Thus, training dataset D consists of randomly selected 250 images from the perfusion CT dataset, all of them are cropped uniformly to the size of 331×363. This pre-processing step removes the skull and background from raw CT images and improves feature learning efficiency during training. Testing data are randomly selected 250 images with no overlap with the training data. They remain as 512×512 grayscale images. Another 100 images with no overlap with the training/testing data are selected as the validation set. We use Peak Signal-to-Noise Ratio (PSNR) as the fitness function in approach.

4.2. Transfer learning

GA requires high computational resources due to a large search space, which leads to difficulties when evaluating performance on large datasets directly. Our strategy is to explore promising CNN hyperparameter combinations by training on a small subset D_s. In particular, 35 images from the training data are randomly selected and segmented with patch size 50×50 at a stride of 20. Therein, 8576 image patches are generated for the initial evolution. We then transfer hyperparameters observed from results on D_s to a large training set D_l. With the same patch size and stride length, 100 images of D_l are segmented into 17,280 patches for further evolution.

4.3. Low-dose noise simulation

Repeated scans at different radiation doses on the same patient are not ethical due to increased unnecessary radiation exposure. Therefore, in this paper, low-dose perfusion CT images are stimulated and added to the regular dose perfusion CT images. Specifically, spatially correlated normally distributed noise (Britten et al., 2004) is added to both training data and testing data. The added noise has a standard deviation of $\sigma = 17, 22, 32,$ which corresponds to the tube current-time product of 30, 20, 10 mAs. The regular dose level is 190 mAs.

4.4. Experimental setup

All possible genes θ are selected from CNN hyperparameters with promising performance reported in the literature (Zhang et al., 2017; Mao et al., 2016). In this paper, we consider a constrained case with θ consisting of four sub-genotypes: number of layers = (1, 2, 3, 4, 5, 6, 7), number of neurons in each layer = (16, 32, 64, 96, 128, 256), activation = (‘ReLU’,‘Tanh’,‘SELU’,‘ELU’,‘Sigmoid’), and optimizers = (‘Rmsprop’, ‘SGD’, ‘Adam’, ‘Adamax’, ‘Adadelta’, ‘Adagrad’). During initialization, we set the initial fine-gene set θ_f from set θ as number of layers = (5, 6), number of neurons in each layer = (32, 48), activation = (‘ReLU’, ‘ELU’, ‘Sigmoid’), and optimizers = (‘SGD’,‘Adam’).

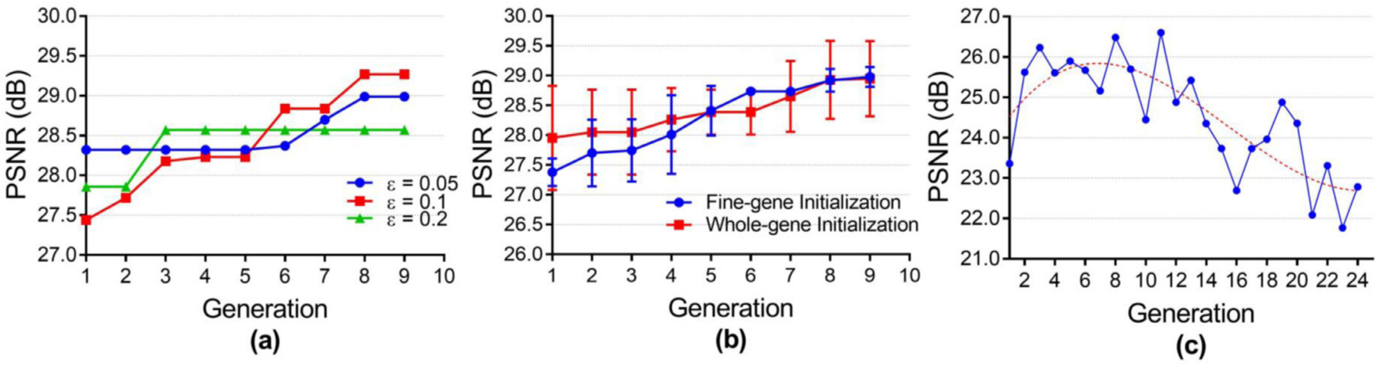

Fig. 2. (a) The performance of best individual with respect to mutation rate $\epsilon = 0.05, 0.1, 0.2$. (b) The average performance over top 5 individuals with respect to the initialization process with a fine-gene set θ_f and whole-gene set θ. (c) The average performance overall individuals with respect to the generation number. All training is processed on a large dataset D_l.

We create an initial population size $N = 20$ individuals and perform genetic operations for 10 rounds (generation). For each generation, we set mutation possibility rate $\epsilon = 0.1$. Crossover happens between any two random parents networks. After each crossover and mutation, we check the whole population and eliminate duplicate individuals (see Algorithm 1). Other hyperparameters (e.g., learning rate) follow Tensorflow default settings. Residual learning (Kiku et al., 2013) is adopted to accelerate the training process. All GA progress is processed on Tensorflow platform with GEFORCE GTX TITAN GPUs.

4.5. Parameters selection

We evaluate the performance of different mutation rate as shown in Fig. 2(a). When the mutation rate is too high, it increases the searching speed in the search space but may not find optimal individuals in each generation. On the other hand, when the mutation rate is too low, it can lead individuals to converge rapidly to local optimum instead of the global optimum. From Fig. 2(a), $\epsilon = 0.1$ gives the optimal performance. We also evaluate different initialization strategies as shown in Fig. 2(b). Fine-gene initialization with selected genes can reach the same performance as the whole gene initialization strategy after 8 generations while we set fine-genes as a greedy initialization set to help early generations find high-performance individuals. However, after certain generations, more mutation genes are introduced due to duplicate individual elimination, which increases population diversity but reduces the average performance. This strategy helps to stop early at an optimal generation and improves search efficiency. This is demonstrated in Fig. 2 (c). We use 10 generations as shown in Fig. 2 (c).

4.6. Gene evolution

We track the evolution of genes over generations and illustrate the optimizer function genes in Fig. 3. We statistically summarize the gene occurrences amount top 5 individuals in each generation trained on a small training set and after transferring to a large training set (Fig. 3(a)). The low-performance genes are eliminated over the generations, such as *SGD* and *Adagrad*. At the same time, the high-performance genes are introduced from mutation, such as *Adadelta*. After being transferred to a large training set (Fig. 3(b)), the initialization set is transferred from (a), where good "genes" such as *Adam*, *Adadelta*, and *Adamax* are preserved. Through the evolution, top performance genes such as *Adamax* and *Adadelta* dominate the optimizer functions genes. This tracking process demonstrates that our greedy initialization strategy helps to search for high-performance genes efficiently. More importantly, it shows that the learned CNN hyperparameters (genes) and structures are transferable from a small dataset to a large dataset.

4.7. Flexibility

The proposed framework is flexible as "gene" can represent various components of a CNN. In our case, flexibility means we have different genes that contain different parts of the neural network architectures. The one who's going to use our evolutionary algorithm could choose to activate or mute genes as needed. Here, "activate" means we introduce genes into the evolutionary process, while "mute" means we don't change the gene or just fix the structure in neural networks.

To show the flexibility, we set up an experiment using another group of genes, including type of block = ('RC' (Liu and Fang, 2018), 'inception' (Szegedy et al., 2015),'residual' (He et al., 2015), 'transductive'[2]), number of blocks = (4,9), activation = ('ReLU','Tanh','Sigmoid'), optimizers = ('SDG', 'Adam', 'Adamax', 'Adadelta', 'Adagrad'), loss function = ('binary cross-entropy','mean absolute percentage error','mean squared error','mean absolute error','logcosh'), and learning strategy = ('residual','transductive'). In contrast with the first experiment, we activate the following new genes: the type of blocks, the loss functions, and the learning strategy, and also mute some genes, such as the number of layers and the number of neurons.

The number of neurons in each layer is embedded into various network blocks, such as Residual, RC, and Interception, with default values 64 and 32 (only in Interception block). This strategy can take advantage of the state-of-the-art network blocks in feature extraction and also reduce search space significantly because the layer-number and the neuron-number have numerous options but relatively don't contribute much to performance compared to network-structure and loss functions. In a CNN, the loss function determines the direction of optimization. It is essential to include it into the evolutionary framework. Besides, learning strategy is also important. Notably, in image denoising cases, such as DnCNN, residual learning (predicting the difference between input and ground-truth images) is easier than transductive learning (directly mapping input to ground-truth).

We use the same strategy (selecting genes from existing methods (e.g., DnCNN, REDNet)) to initialize the fine-gene set θ_f as type

[2] two stacked convolutional layers without other techniques

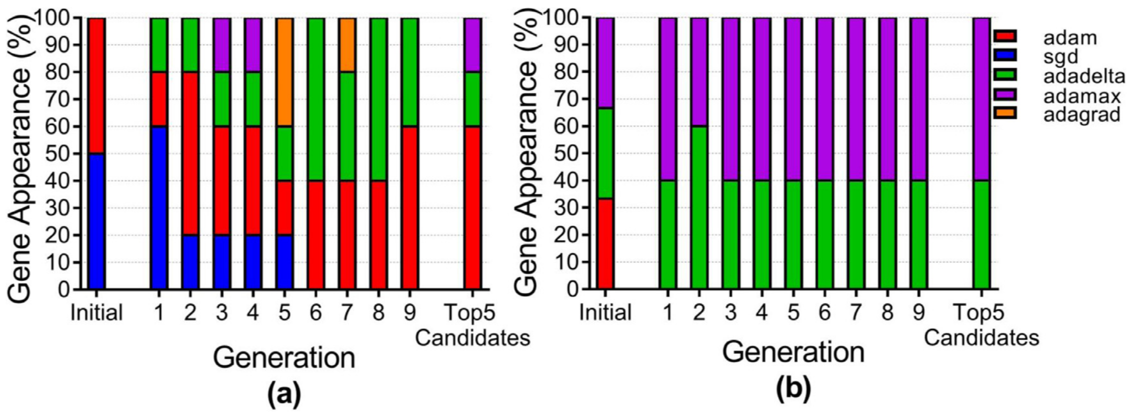

Fig. 3. The optimizer gene appearance changing during evolutionary progress on the small dataset D_s (a) and the large dataset D_l (b) with transferred initialization set. In each generation, 5 top performing individuals are selected to summarize changes. "Top 5 Candidates" bar refers to the final optimizer gene distribution after one evolutionary progress. The final color bar in (a) has a different meaning from the initial color bar in (b). The height of each color bar represents the number of the genes appearing in a generation. The final color bar in (a) represents the gene types appearing in the top 5 performing neural networks and the corresponding gene quantity. The initial color bar (b) represents the gene types in the initial fine-gene set used for the evolution on a large dataset and the corresponding gene quantity. The fine genes are smoothly transferred from the final generation of the neural network evolution that is performed on a small dataset. The transfer process is achieved by replacing the fine-gene set with these fine genes presented in the "Top 5 Candidates". Thus, each type of gene has the same amount (e.g., equal to 1) at the initial stage of the evolutionary process on a large dataset.. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Average PSNR and SSIM of algorithms: BM3D, DnCNN, EvoNet-5, and EvoNet-17 at different noise levels $\sigma = 17, 22, 32$. Best performance is highlighted in bold.

σ	**BM3D** Dabov et al. (2009)		**DnCNN** Zhang et al. (2017)		EvoNet-5		EvoNet-17		EvoNet-block	
	PSNR(dB)	SSIM	PSNR(dB)	SSIM	PSNR(dB)	SSIM	PSNR(dB)	SSIM	PSNR(dB)	SSIM
17	29.07	0.4515	36.64	0.9158	36.30	0.9062	36.65	0.9074	**36.91**	**0.9170**
22	26.98	0.3578	35.87	0.8863	35.66	0.8914	35.92	0.8988	**36.18**	**0.9037**
32	23.95	0.2385	35.03	0.8671	34.35	0.8578	35.04	0.8846	**35.35**	**0.8879**

of block = ('RC','residual','transductive'), number of blocks = (4), activation = ('ReLU', 'Sigmoid'), optimizers = ('SGD','Adam'), loss function = ('mean absolute error','mean squared error'), and learning strategy = ('residual'). The mutation possibility rate ϵ remains at 0.1.

The corresponding evolution processing results are shown in Table 1. We name the network obtained in this experiment as EvoNet-block. As one can see, EvoNet-block has better performance than EvoNet-17; therefore, getting the promising existing CNN blocks and learning strategies involved in evolution processing can raise the possibility of finding a better deep CNN model.

4.8. Comparison with state-of-the-art methods

Both quantitative and qualitative comparisons are provided. We compared with state-of-the-art methods including BM3D and DnCNN. DnCNN is one of few deep learning-based studies (Jifara et al., 2017) in medical image denoising, still very active and published recently. We obtained the EvoNet-5 (5 layers, 64 neurons in each layer, *Adadelta, ReLu*) from D_s, and EvoNet-17 (17 layers, 64 neurons in each layer, *Adadelta, ReLu*) from D_l. In addition, we obtained the EvoNet-block by involving the types of network blocks and trained it on D_l. EvoNet-block has 21 layers (9 residual blocks and additional 3 layers for input-end and output-end), 64 × 3 × 3 neurons in each layer, *mean absolute error* as the loss function, A*dam* as the optimizer, *residual* as the learning strategy, and *Tanh* as the activation function. We visualize the architectures of EvoNet-5, EvoNet-17, and EvoNet-block in Fig. 5.

In Table 1, we present the summary of quantitative results. The deeper EvoNet-block and EvoNet-17 outperform other state-of-the-art methods with PSNR on the testing dataset. The shallow EvoNet-5 achieves comparable performance to DnCNN; however, it is deep (20 layers) while the EvoNet-5 is a compact structure with stacked convolutional layers without regularization technique. During the first experiment, deeper (6, 7 layers) and larger (128, 256 neurons) networks are eliminated due to overfitting on small data. Therefore, we initialize the number of blocks in the second experiment with a small number to save the computing resource and speed up evolution processing. Fig. 4 shows the visual results. Our method perfectly restores physiological structures, circuit contour and texture of the cerebral cortex and gains high PSNR values. It is matching with quantitative results.

The flexibility of our evolutionary framework can find a better combination of components of a neural network that is suitable for the target task. In Table 1, as one can see, EvoNet-block has much better performance than EvoNet-17 that is obtained from stacking convolutional layers without involving state-of-the-art network blocks. Taking advantage of existing CNN components is a promising strategy to explore better network architectures.

4.9. Non-expedited and expedited evolution comparison

We summarize the neural network evolutionary time of using the expedited and the non-expedited genetic algorithm in our case. The evolutionary time is primarily determined by the number of candidates (neural networks) and the total number of training images. The iteration number of training each candidate also affects the total evolutionary time. However, compared to the two aspects mentioned above, the training iteration (or epoch) number is a relatively small influence factor. In addition, because we usually adopt early stopping strategy for training each candidate, the exact evolutionary time generated from iteration number is difficult to approximate. The similar case is also applicable to the learning rate. Many other aspects can also affect the evolutionary time, including hardware (e.g., GPU), operating system, and deep learning training platform (e.g., Tensorflow). However, apart from GPUs, these

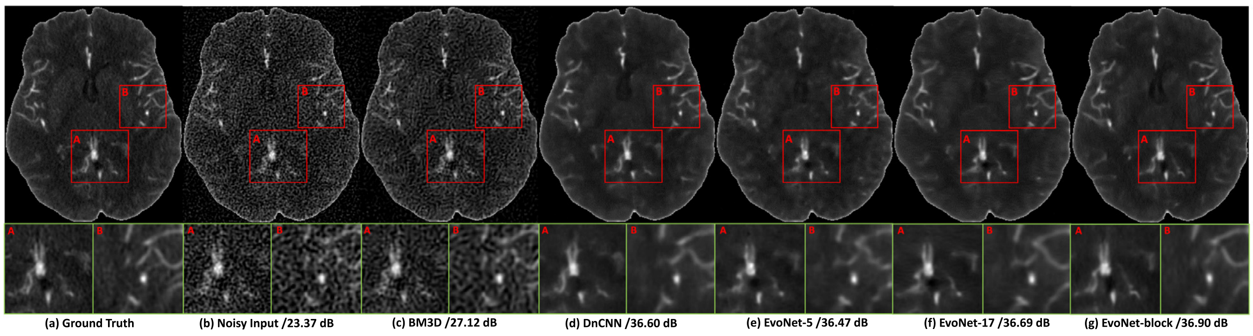

Fig. 4. Visual Results of perfusion CT dataset with noise $\sigma = 22$. A region of Interest (ROI) is selected (red region) and scaled up for better visual comparison. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

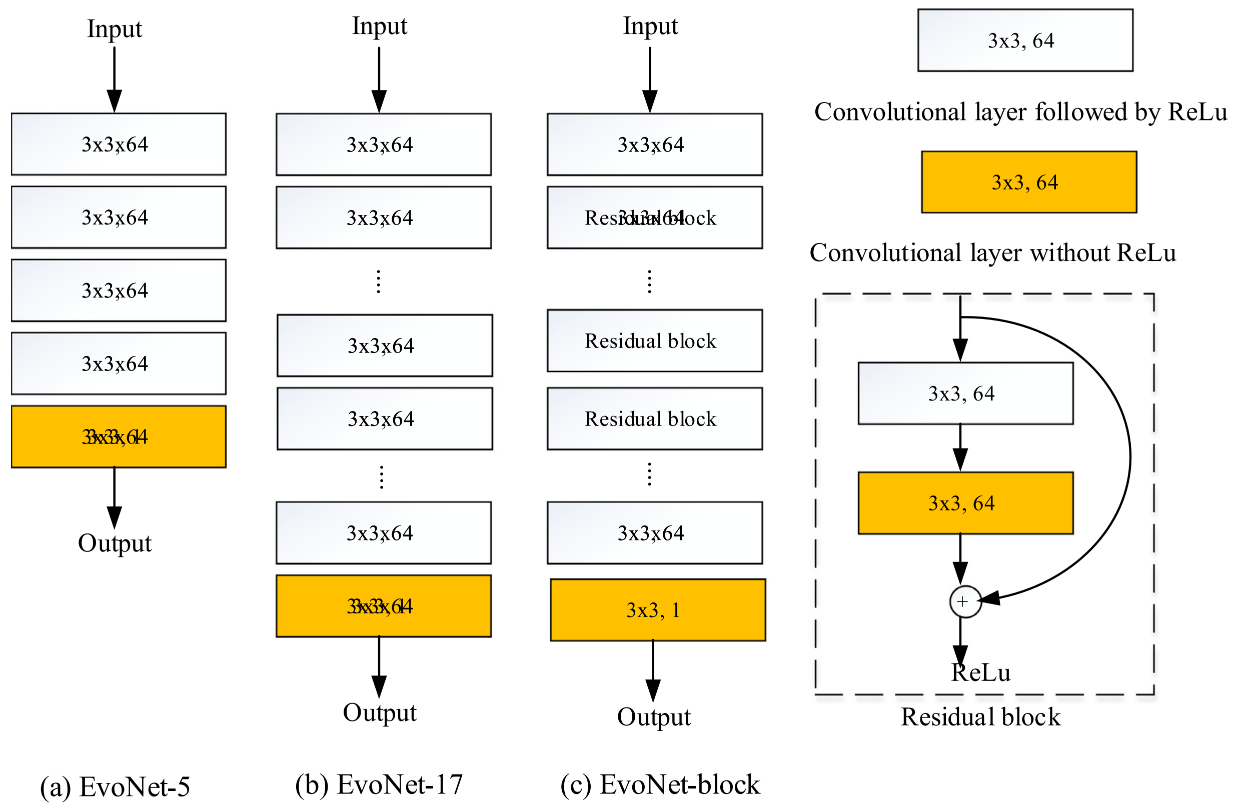

Fig. 5. The visualization of the architectures of EvoNet-5, EvoNet-17, and EvoNet-block. As one can see, although all these network architectures are standard and common, they can achieve competitive and even superior performance, thus the fittest hyperparameters obtained through the proposed evolutionary framework can train a better CNN..

aspects are also considered as minor effect factors in the evolutionary process when compared with training data size and candidate number. Therefore, we calculate the evolutionary time with only considering the number of candidates and the number of training images as effect-factors.

The expedited evolutionary process time is the summation of the evolutionary process time on a small dataset and on a large dataset. The non-expedited evolutionary process time is the time of exploring neural networks on a large dataset directly. In our experiments, we use 4 GEFORCE GTX TITAN GPUs for processing 200 candidates, and the approximate expedited evolutionary process time is about 135 hours, which includes 5 hours evolutionary process time on a small dataset (e.g., 30 images) for finding the best genes from the 200 candidates and then 130 hours evolutionary process time on a large dataset (e.g., 250 images) for composing of the best CNN architectures from a reduced gene search space. In contrast, the approximate non-expedited evolutionary process time is about 330 hours. In other words, compared to using the original genetic algorithm, our expedited genetic algorithm based evolutionary framework can reduce an approximate 59% evolutionary process time for exploring the best neural network. The efficiency improvement also depends on the stop condition considerably. The stop condition can vary from case to case. In our case, it is stopped once the top 5 individuals do not change in the next generation.

5. Discussion & conclusion

In this work, we propose an optimized GA-based strategy to explore CNN structure for medical image denoising. We introduce an experience-based greedy exploration strategy and transfer learning to accelerate GA evolution. We evaluate EvoNets on a perfusion CT dataset and demonstrate promising performance.

Compared to other neural network architecture searching methods, the proposed framework has two key advantages. First, our method focuses on not only network structure combination but also hyper-parameters optimization. Other methods mostly focus on finding the best structural components. However, the hyper-parameters are critical for optimizing performance. Second, our framework is more flexible and can work in real applications without the needs of large-scale computing resources.

In experiments, we first change normal genes such as the type of the optimizers, the choice of the activation function, the number of layers, and the number of neurons. Then we generate EvoNet-5 and EvoNet-17 from two different datasets, which perform competitively. To achieve higher performance, we introduced block gene, which contains current state-of-the-art block structures, such as RC and residual, leading to the increased performance of the EvoNet-block.

In the current work, we only consider a constrained case. In future work, the proposed method can be extended to explore more flexible CNN structures for challenging tasks, such as lesion detection, segmentation, and classification. Our proposed framework is general and flexible. For example, if we apply the proposed method for medical image segmentation, the changes to our proposed method can be summarized by (a) replacing the fitness function with Intersection Over Union (IOU) metric, which is to measure the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks; (b) adding segmentation loss function (e.g., dice loss (Milletari et al., 2016)) into the gene set; (c) replacing the training and evaluation dataset. All of these changes are easily implemented. Thus, the proposed framework can be easily generalized to other applications and data domains.

Conflict of interest

None.

Acknowledgments

This work is partially supported by National Science Foundation IIS-1564892 and National Institute of Health National Center for Advancing Translational Sciences UL1 TR001427.

References

Baker, B., Gupta, O., Naik, N., Raskar, R., 2016. Designing neural network architectures using reinforcement learning. arXiv preprint arXiv: 1611.02167

Britten, A., Crotty, M., Kiremidjian, H., Grundy, A., Adam, E., 2004. The addition of computer simulated noise to investigate radiation dose and image quality in images with spatial correlation of statistical noise: an example application to x-ray ct of the brain. Br. J Radiol 77 (916), 323–328.

Chen, F., Zhang, L., Yu, H., 2015. External patch prior guided internal clustering for image denoising. In: *Proceedings of the IEEE international conference on computer vision*, pp. 603–611.

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.

Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K., 2009. BM3D image denoising with shape-adaptive principal component analysis. *SPARS'09-Signal Processing with Adaptive Sparse Structured Representations*.

Elad, M., Aharon, M., 2006. Image denoising via sparse and redundant representations over learned dictionaries. IEEE Trans. Image Process. 15 (12), 3736–3745.

Van den Elsen, P.A., Pol, E.-J., Viergever, M.A., 1993. Medical image matching-a review with classification. IEEE Eng. Med. Biol. Mag. 12 (1), 26–39.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F., 2015. Efficient and robust automated machine learning. In: *Advances in Neural Information Processing Systems*, pp. 2962–2970.

Geem, Z.W., Kim, J.H., Loganathan, G.V., 2001. A new heuristic optimization algorithm: harmony search. Simulation 76 (2), 60–68.

Goldberg, D.E., Holland, J.H., 1988. Genetic algorithms and machine learning. Mach. Learn. 3 (2), 95–99.

Goldberg, D.E., Richardson, J., et al., 1987. Genetic algorithms with sharing for multimodal function optimization. In: Genetic algorithms and their applications: *Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum, pp. 41–49.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: *Advances in neural information processing systems*, pp. 2672–2680.

Gu, S., Zhang, L., Zuo, W., Feng, X., 2014. Weighted nuclear norm minimization with application to image denoising. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2862–2869.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. arXiv preprint arXiv: 1512.03385

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

Ho, T.K., 1995. Random decision forests. In: Document analysis and recognition, 1995., *Proceedings of the third international conference on*, . IEEE, pp. 278–282.

Holland, J.H., 1992. Genetic algorithms. Sci. Am. 267 (1), 66–73.

Hore, A., Ziou, D., 2010. Image quality metrics: PSNR vs. SSIM. In: *Pattern recognition (icpr), 2010 20th international conference on*. IEEE, pp. 2366–2369.

Horn, J., Nafpliotis, N., Goldberg, D.E., 1994. A niched pareto genetic algorithm for multiobjective optimization. In: Evolutionary Computation, 1994. *IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, pp. 82–87.

Jifara, W., Jiang, F., Rho, S., Cheng, M., Liu, S., 2017. Medical image denoising using convolutional neural network: a residual learning approach. J. Supercomput. 1–15.

Joachims, T., 2003. Transductive learning via spectral graph partitioning. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 290–297.

Johnson, J., Alahi, A., Fei-Fei, L., 2016. Perceptual losses for real-time style transfer and super-resolution. In: *European Conference on Computer Vision*. Springer, pp. 694–711.

Kiku, D., Monno, Y., Tanaka, M., Okutomi, M., 2013. Residual interpolation for color image demosaicking. In: 2013 *IEEE International Conference on Image Processing*. IEEE, pp. 2304–2308.

Kingma, D.P., Ba, J., 2014. Adam: a method for stochastic optimization. arXiv preprint arXiv: 1412.6980

Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F., 2016. Fast Bayesian optimization of machine learning hyperparameters on large datasets. arXiv preprint arXiv: 1605.07079

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D.D., Chen, M., 2014. Medical image classification with convolutional neural network. In: *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*. IEEE, pp. 844–848.

Li, S., Yin, H., Fang, L., 2012. Group-sparse representation with dictionary learning for medical image denoising and fusion. IEEE Trans. Biomed. Eng. 59 (12), 3450–3459.

Liu, P., Fang, R., 2018. SDCNet: Smoothed dense-convolution network for restoring low-dose cerebral ct perfusion. In: *Biomedical Imaging (ISBI 2018), 2018 IEEE 15th International Symposium on*. IEEE, pp. 349–352.

Liu, P., Li, Y., El Basha, M.D., Fang, R., 2018. Neural Network Evolution Using Expedited Genetic Algorithm for Medical Image Denoising. International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham., pp. 12–20.

Mao, X.-J., Shen, C., Yang, Y.-B., 2016. Image restoration using convolutional auto-encoders with symmetric skip connections. arXiv preprint arXiv: 1606.08921

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S., 2010. Recurrent neural network based language model. In: *Eleventh Annual Conference of the International Speech Communication Association*.

Miller, J., Turner, A., 2015. Cartesian genetic programming. In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, pp. 179–198.

Milletari, F., Navab, N., Ahmadi, S.-A., 2016. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In: 3D Vision (3DV), *2016 Fourth International Conference on*. IEEE, pp. 565–571.

Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Olson, R.S., Moore, J.H., 2016. Tpot: A tree-based pipeline optimization tool for automating machine learning. In: *Workshop on Automatic Machine Learning*, pp. 66–74.

Parikh, T., Drew, S.J., Lee, V.S., Wong, S., Hecht, E.M., Babb, J.S., Taouli, B., 2008. Focal liver lesion detection and characterization with diffusion-weighted MR imaging: comparison with standard breath-hold t2-weighted imaging. Radiology 246 (3), 812–822.

Pham, D.L., Xu, C., Prince, J.L., 2000. Current methods in medical image segmentation. Annu. Rev. Biomed. Eng. 2 (1), 315–337.

Real, E., Aggarwal, A., Huang, Y., Le, Q.V., 2018. Regularized evolution for image classifier architecture search. arXiv preprint arXiv: 1802.01548

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q., Kurakin, A., 2017. Large-scale evolution of image classifiers. arXiv preprint arXiv: 1703.01041

Sajjadi, M.S., Schölkopf, B., Hirsch, M., 2016. Enhancenet: single image super-resolution through automated texture synthesis. arXiv preprint arXiv: 1612.07919

Stanley, K.O., Miikkulainen, R., 2002. Evolving neural networks through augmenting topologies. Evol. Comput. 10 (2), 99–127.

Suganuma, M., Shirakawa, S., Nagao, T., 2017. A genetic programming approach to designing convolutional neural network architectures. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, pp. 497–504.

Sutton, R.S., Barto, A.G., 1998. Reinforcement learning: An introduction, 1. MIT Press Cambridge.
Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al., 2015. *Going deeper with convolutions*. CVPR.
Torrey, L., Shavlik, J., 2010. Transfer Learning. In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, pp. 242–264.
Van Leemput, K., Maes, F., Vandermeulen, D., Colchester, A., Suetens, P., 2001. Automated segmentation of multiple sclerosis lesions by model outlier detection. IEEE Trans. Med. Imaging 20 (8), 677–688.
Wang, Y., Zhou, H., 2006. Total variation wavelet-based medical image denoising. Int. J. Biomed. Imaging. 2006.
Whitaker, R.T., Xue, X., 2001. Variable-conductance, level-set curvature for image denoising. In: *Image Processing, 2001. Proceedings. 2001 International Conference on*, 3. IEEE, pp. 142–145.
Xie, L., Yuille, A., 2017. Genetic cnn. arXiv preprint arXiv: 1703.01513
Yang, J., Honavar, V., 1998. Feature subset selection using a genetic algorithm. In: *Feature Extraction, Construction and Selection*. Springer, pp. 117–136.
Yao, X., 1999. Evolving artificial neural networks. Proc. IEEE 87 (9), 1423–1447.
Yosinski, J., Clune, J., Bengio, Y., Lipson, H., 2014. How transferable are features in deep neural networks? In: *Advances in neural information processing systems*, pp. 3320–3328.
Youssef, H., Sait, S.M., Adiche, H., 2001. Evolutionary algorithms, simulated annealing and tabu search: a comparative study. Eng. Appl. Artif. Intell. 14 (2), 167–181.
Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L., 2017. Beyond a gaussian denoiser: residual learning of deep CNN for image denoising. IEEE Trans. Image Process. 26 (7), 3142–3155.
Zoph, B., Le, Q.V., 2016. Neural architecture search with reinforcement learning. arXiv preprint arXiv: 1611.01578