Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A hybrid-model optimization algorithm based on the Gaussian process and particle swarm optimization for mixed-variable CNN hyperparameter automatic search*

Han YAN, Chongquan ZHONG, Yuhu WU, Liyong ZHANG, Wei LU†‡

School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China

†E-mail: luwei@dlut.edu.cn

Received Oct. 27, 2022; Revision accepted Apr. 20, 2023; Crosschecked July 17, 2023; Published online Sept. 7, 2023

Abstract: Convolutional neural networks (CNNs) have been developed quickly in many real-world fields. However, CNN's performance depends heavily on its hyperparameters, while finding suitable hyperparameters for CNNs working in application fields is challenging for three reasons: (1) the problem of mixed-variable encoding for different types of hyperparameters in CNNs, (2) expensive computational costs in evaluating candidate hyperparameter configuration, and (3) the problem of ensuring convergence rates and model performance during hyperparameter search. To overcome these problems and challenges, a hybrid-model optimization algorithm is proposed in this paper to search suitable hyperparameter configurations automatically based on the Gaussian process and particle swarm optimization (GPPSO) algorithm. First, a new encoding method is designed to efficiently deal with the CNN hyperparameter mixed-variable problem. Second, a hybrid-surrogate-assisted model is proposed to reduce the high cost of evaluating candidate hyperparameter configurations. Third, a novel activation function is suggested to improve the model performance and ensure the convergence rate. Intensive experiments are performed on image-classification benchmark datasets to demonstrate the superior performance of GPPSO over state-of-the-art methods. Moreover, a case study on metal fracture diagnosis is carried out to evaluate the GPPSO algorithm performance in practical applications. Experimental results demonstrate the effectiveness and efficiency of GPPSO, achieving accuracy of 95.26% and 76.36% only through 0.04 and 1.70 GPU days on the CIFAR-10 and CIFAR-100 datasets, respectively.

Key words: Convolutional neural network; Gaussian process; Hybrid model; Hyperparameter optimization; Mixed-variable; Particle swarm optimization

https://doi.org/10.1631/FITEE.2200515 **CLC number:** TP181

1 Introduction

In recent years, as one of the most useful deep learning models, convolutional neural networks (CNNs) have achieved state-of-the-art results in various artificial intelligence (AI) applications (Jiang and Luo, 2022; Tulbure et al., 2022). Through convolution operations, meaningful features are extracted from input data, which have greatly improved model performance (Sun et al., 2019). Such learning and expression abilities result in great success in various real-world applications, such as face detection (Li X et al., 2022) and autonomous driving (Grigorescu et al., 2020). Although CNNs have achieved great success, the design of CNN architecture is still extremely complicated, and obtaining efficient CNN models for solving specific tasks is still a challenge

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 62073056 and 61876029), the Applied Basic Research Project of Liaoning Province, China (No. 2023JH2/101300207), and the Key Field Innovation Team Project of Dalian, China (No. 2021RT14)

ORCID: Han YAN, https://orcid.org/0000-0003-1777-1859; Wei LU, https://orcid.org/0000-0002-5775-1222

(Fernandes and Yen, 2021; Guo et al., 2022). Furthermore, most efficient CNN models are designed and optimized by experienced AI algorithm engineers through tedious trial-and-error experiments, which does not help technicians in other industries who want to use AI technology.

Therefore, many researchers have started to consider designing CNN models in a more intelligent, automatic, and efficient way (Li JY et al., 2022). For example, to realize automation in the model design process, researchers regarded the process as an optimization problem, and found the optimal solution using intelligent algorithms (Zhan et al., 2022a). Intelligent optimization algorithms (OAs) are regarded as a class of methods of deriving optimal solutions that have been extensively researched, such as using Bayesian optimization (BO) (Snoek et al., 2012), particle swarm optimization (PSO) (Poli et al., 2007; Li JY et al., 2021), and other evolutionary computation (EC) approaches (Li JY et al., 2020; Zhan et al., 2022b; Wu SH et al., 2023). As an application of the Gaussian process (GP) in optimization problems, BO has been proposed to optimize search hyperparameters in machine learning algorithms. Snoek et al. (2012) selected expected improvement (EI) as the acquisition function to optimize three typical machine learning problems, which reduced training cost and improved training speed. Jin et al. (2019) used BO to guide the generation of network morphism for automatic construction of CNN models, and a new GP kernel function was designed according to the characteristics of network morphism exploration space. Li JY et al. (2023) proposed a surrogate-assisted hybrid model to optimize CNN's hyperparameters, where GP was used as a surrogate model to estimate the fitness function to save computational cost. Li JY et al. (2022) and Zhan et al. (2022a) used BO to search different combinations of hyperparameters to construct different architectures. The unique properties of BO can reduce the number of trained neural networks, resulting in a more efficient search process. As a branch of EC, PSO is efficient to find optimal solutions to high-complexity problems due to its wide exploration area and fast convergence (Li JY et al., 2022). For instance, Wang B et al. (2018) proposed a variable-length encoding method for CNNs and searched hyperparameters through PSO to optimize CNNs, whereas Fielding et al. (2019) used a PSO algorithm and ensemble learning to automatically design CNNs. Wu T et al. (2019) treated model pruning as a multi-objective problem and solved it by PSO to balance the accuracy and complexity, which can reduce weights by 80% without losing significant accuracy. Wang B et al. (2020) proposed an efficient PSO (EPSOCNN) method inspired by transfer learning to accelerate search process, which reduced the search space and demonstrated the transferability of the evolved block. Wang YQ et al. (2022) designed a novel and light-weight scale-adaptive fitness evaluation-based PSO method for reducing search time and providing search performance. In addition, some other EC algorithms can achieve better results in cooperation with model optimization (Alvarez-Rodriguez et al., 2021; Chen et al., 2022). Real et al. (2017) proposed a large-scale neuro-evolutionary method to discover the best CNN model, and achieved 94.6% and 77.0% accuracy on CIFAR-10 and CIFAR-100, respectively. Sun et al. (2020a) used a novel genetic algorithm (GA) called CNN-GA to search CNN architecture automatically. These algorithms have achieved promising results in CNN hyperparameter optimization tasks.

However, there are still some challenges for CNN hyperparameter optimization tasks due to following problems: mixed-value hyperparameter encoding, high computational cost, low convergence rate, and limited model performance. First, the CNN hyperparameter types are different (continuous or discrete) (Darwish et al., 2020), and such mixed-variable characteristics are proved difficult in efficient search space encoding. Second, for traditional OAs (BO and PSO), CNNs are evaluated by the fitness function through assessment criteria based on training, which increases the cost of fitness evaluation (FE) and damages the efficiency of OAs. Third, considering the large number of CNN hyperparameters, it is still necessary to research how to accelerate the convergence for FE and ensure model performance after search.

Therefore, in this paper we focus on these challenging tasks in the CNN hyperparameter optimization problem and propose a novel Gaussian process and particle swarm optimization (GPPSO) method based on GP and PSO, to solve these difficulties. The major challenges and contributions are summarized in Fig. 1.

1. A novel encoding strategy is proposed to

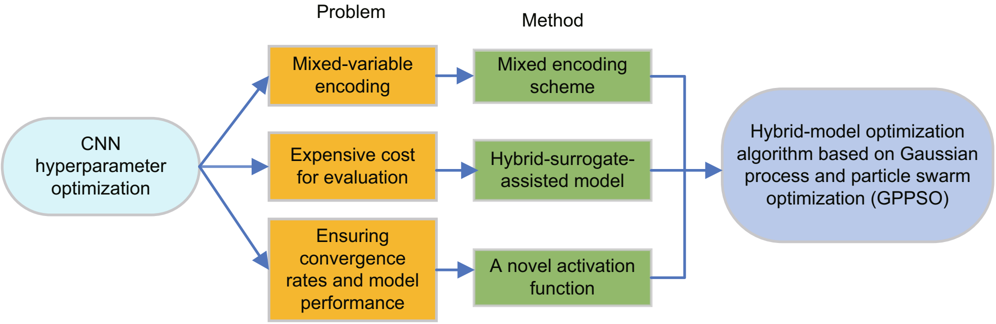

Problem
Method
CNN hyperparameter optimization
Mixed-variable encoding
Expensive cost for evaluation
Ensuring convergence rates and model performance
Mixed encoding scheme
Hybrid-surrogate-assisted model
A novel activation function
Hybrid-model optimization algorithm based on Gaussian process and particle swarm optimization (GPPSO)

Fig. 1 Major challenges and contributions of GPPSO

efficiently deal with the mixed-variable difficulty of CNN hyperparameters. A unified encoding strategy is designed to encode discrete and continuous variables in the same form, making the optimization process more efficient.

2. A hybrid-surrogate-assisted (HSA) model is proposed to deal with the expensive computational cost problem in the search process. During the search process, the GP model serves as a surrogate for the fitness function, while the PSO algorithm generates new individuals. To achieve a better balance between efficiency and performance, a multi-level evaluation mechanism is proposed to reduce computational cost.

3. A novel activation function (AF, Ta-ReLU) is proposed to accelerate the convergence in the process of population evolution and to improve the performance of the model after training. The improved AF has a tiny gradient in the region (<0), which not only enhances the model's performance, but also ensures efficient training.

2 Background and related works

The concepts of CNN, Gaussian process regression (GPR), and PSO as the basic algorithms of GPPSO are introduced in Sections 2.1, 2.2, and 2.3, respectively, which are helpful to know the details of the proposed GPPSO.

2.1 Convolutional neural network (CNN)

CNN is a type of deep feedforward neural network that has the advantages of local links and weight sharing (Alzubaidi et al., 2021). With the development of CNNs, the network structures have gradually become deeper, and VGGNet and ResNet have emerged as two state-of-the-art CNN structures in recent years.

VGGNet (Simonyan and Zisserman, 2014) is a frequently employed CNN for extracting features. Fig. 2 illustrates a 16-layer version of VGGNet, consisting of 13 Conv layers and three fully connected (FC) layers. Because smaller 3×3 Conv kernels are used to simulate 5×5 Conv kernels, VGGNet can obtain larger receptive fields with fewer parameters, resulting in effective and efficient feature extraction.

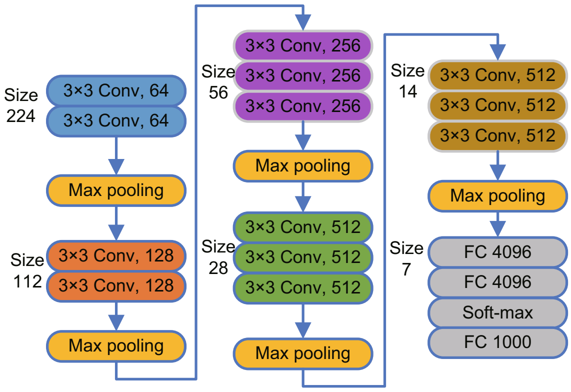


Fig. 2 VGG16 model structure

ResNet (He et al., 2016) is another commonly used residual structure CNN for feature extraction. ResNet used cross-layer connections to fit residual items, which extends the depth of CNNs. The residual block is shown in Fig. 3a, which outputs $y = F(x) + x$ after the input passes through the module. The overall structure of ResNet is shown in Fig. 3b.

Based on their effective characteristics, VGGNet and ResNet are chosen as basic models for the proposed algorithm.

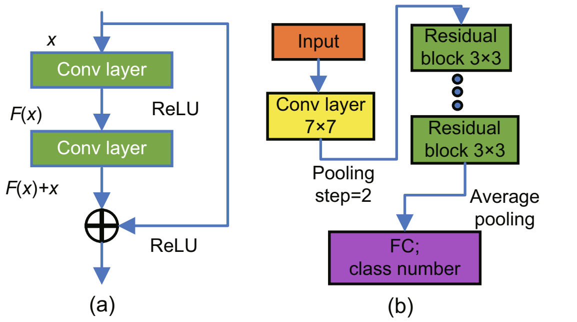


Fig. 3 ResNet model structure: (a) residual block; (b) ResNet

2.2 Gaussian process regression (GPR)

Gaussian process regression (GPR) is an efficient modeling algorithm based on statistical learning theory. In contrast to parameterized models in machine learning (e.g., Bayesian linear regression), GP is a nonparameterized model that can fit a black box function and give confidence in the fitting results. In GPR, it is assumed that an unknown function $f(\boldsymbol{x})$ is smooth and follows a GP. When N points $\boldsymbol{X}=[\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N]$ are sampled from $f(\boldsymbol{x})$, the resulting dataset follows a multivariate normal distribution as shown in Eq. (1):

$$[f(\boldsymbol{x}_1),f(\boldsymbol{x}_2),\cdots,f(\boldsymbol{x}_N)]^{\mathrm{T}}\sim\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{X}),\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})),\tag{1}$$

where $\boldsymbol{\mu}(\boldsymbol{X})$ is the mean vector, $\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})=[k(\boldsymbol{x}_i,\boldsymbol{x}_j)]_{N\times N}$ is the covariance matrix, and $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is a kernel function for measuring the similarity between two samples. Eq. (2) is the commonly used squared exponential kernel function in GP:

$$k(\boldsymbol{x}_i,\boldsymbol{x}_j)=\exp\left(\frac{-\|\boldsymbol{x}_i-\boldsymbol{x}_j\|}{2l^2}\right),\tag{2}$$

where l is the hyperparameter used to control the sensitivity of the kernel function.

In this study, the performance value of the deep learning model with different hyperparameters is assumed as a group of noisy observations of function $f(\boldsymbol{x})$: $\{(\boldsymbol{x}_n,y_n)\}_{n=1}^{N}$, where $y_n\sim\mathcal{N}\left(f(\boldsymbol{x}_n),\sigma^2\right)$ is the normally distributed observation of $f(\boldsymbol{x}_n)$, and σ is the noise variance. For the sample point $\boldsymbol{x}^*$ to be predicted, the corresponding observed value is y^*. Vector $\boldsymbol{y}=[y_1,y_2,\cdots,y_N]^{\mathrm{T}}$ is the existing observed value. According to the properties of GP, $[\boldsymbol{y},y^*]$ satisfies Eq. (3):

$$\begin{bmatrix}\boldsymbol{y}\\ y^*\end{bmatrix}\sim\mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}(\boldsymbol{X})\\ \boldsymbol{\mu}(\boldsymbol{x}^*)\end{bmatrix},\begin{bmatrix}\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I},(\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X}))^{\mathrm{T}}\\ \boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X}),\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{x}^*)\end{bmatrix}\right),\tag{3}$$

where $\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X})=[k(\boldsymbol{x}^*,\boldsymbol{x}_1),k(\boldsymbol{x}^*,\boldsymbol{x}_2),\cdots,k(\boldsymbol{x}^*,\boldsymbol{x}_n)]$. According to the above joint distribution, the posterior distribution of y^*, mean $\hat{\boldsymbol{\mu}}$, and variance value $\hat{\sigma}^2$ are calculated as follows:

$$p(y^*\mid\boldsymbol{X},\boldsymbol{y})=\mathcal{N}\left(\hat{\boldsymbol{\mu}},\hat{\sigma}^2\right),\tag{4}$$

$$\begin{aligned}\hat{\boldsymbol{\mu}}=&\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X})(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I})^{-1}\\ &\cdot(\boldsymbol{y}-\boldsymbol{\mu}(\boldsymbol{X}))+\boldsymbol{\mu}(\boldsymbol{x}^*),\end{aligned}\tag{5}$$

$$\begin{aligned}\hat{\sigma}^2=&k(\boldsymbol{x}^*,\boldsymbol{x}^*)-\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X})\\ &\cdot(\boldsymbol{K}(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I})^{-1}(\boldsymbol{K}(\boldsymbol{x}^*,\boldsymbol{X}))^{\mathrm{T}}.\end{aligned}\tag{6}$$

Based on the above properties, GPR is employed as a surrogate-assisted model to predict the performance of the model in this study.

2.3 Particle swarm optimization (PSO)

PSO is a type of EC approach based on artificial life and evolutionary computing theory. The main procedures of PSO are as follows: first, PSO initializes a population of individuals with position $\boldsymbol{X}_i^0$ and speed $\boldsymbol{V}_i^0$, where each individual corresponds to a random candidate solution to the object function. Then, based on the fitness value, individual extremum pbest and global extremum gbest are updated using Eqs. (7) and (8):

$$\begin{aligned}v_{id}^{k+1}=&wv_{id}^{k}+c_1r_1(\mathrm{pbest}_{id}^{k}-x_{id}^{k})\\ &+c_2r_2(\mathrm{gbest}_{d}^{k}-x_{id}^{k}),\end{aligned}\tag{7}$$

$$x_{id}^{k+1}=x_{id}^{k}+v_{id}^{k+1},\tag{8}$$

where v_{id} and x_{id} refer to the d^{th} velocity component and position component of the i^{th} particle of the k^{th} generation respectively, and c_1 and c_2 are learning rates, which control the amplitude of evolution to the individual best particle and the global best particle respectively. The above procedures will be repeated until the expected error value is reached or the maximum number of iterations is reached. Finally, the PSO outputs the best position of the particle, which corresponds to the optimal solution to the problem. Due to its ease of implementation and minimal tunable parameters, PSO can be a good choice for solving complex optimization problems in CNNs, and is also a basic algorithm of GPPSO.

3 Proposed algorithm

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

First, we present the mixed-variable encoding strategy, which is used to encode different types of CNN hyperparameters in the same form. Subsequently, we explain the HSA model used to search CNN hyperparameters, which deals with the expensive computational cost problem. We also detail the novel AF, which ensures convergence rate and model performance. Finally, we present the complete algorithm for better understanding of the proposed GPPSO.

3.1 Mixed-variable encoding strategy

In GPPSO, each sampled individual represents a group of CNN hyperparameters, where each dimension of the individual corresponds to a CNN hyperparameter. Because the CNN hyperparameters have distinct meanings, each individual has its own specific types and constraints. For example, some hyperparameters should be set as integers with a large range, such as the number of Conv kernels, whereas some hyperparameters should be discrete, such as the size of Conv kernels. Most traditional optimization methods are aimed at a single type of variable, which is difficult when optimizing mixed-variable types. Therefore, it is important to design a mixed-variable encoding scheme for handling the mixed-variable problem.

Table 1 provides the CNN hyperparameter settings to be optimized in this study. In particular, the variables with many available choices or a large search range are encoded as continuous variables, whereas the variables with several fixed selections are encoded as discrete variables. Although the variables of Conv kernels and FC layer neurons are required to be integers, regarding them as continuous variables can be more flexible and efficient in the optimization process. The reason for processing integers as continuous variables is that the search space for the numbers of kernels and neurons is large and does not have a prior known upper bound (e.g., $[1, +\infty)$), which is inefficient and complex to encode them as finite integers. As for the discrete variables with only a few choices, we encode these discrete variables in a continuous way. The advantages of encoding discrete variables in this way are as follows: first, the size of Conv kernels, the types of AFs, and the types of pooling layers have only three or four available results. These results correspond to a small search space (e.g., $\{0, 1, 2, 3\}$), which is feasible for continuous encoding. Second, encoding continuous variables and discrete variables in the same way is convenient and efficient, which is conductive to the GPPSO algorithm.

Based on the above analysis, the mixed-variable encoding strategy is given as follows: in detail, $\boldsymbol{X}$ of every particle is a vector with dimension D; each dimension represents a CNN hyperparameter. As a result, each vector $\boldsymbol{X}$ represents an architecture of the candidate CNN structure. Because each dimension has a different meaning and the range of values in each dimension is different, the following strategy is designed for encoding: first, for the integer variable in continuous variables, the encoding strategy is shown in Eq. (9):

$$N_{\mathrm{k}} = \lfloor X_{\mathrm{nk}} \rfloor, \quad X_{\mathrm{nk}} \in [1, +\infty), \tag{9}$$

where N_{k} denotes the number of kernels in the Conv layer, X_{nk} denotes the search result of the number of kernels by the optimization method, and $\lfloor\cdot\rfloor$ denotes taking the largest integer that is no larger than the search result. Using this method, we translate the continuous variable in $[1,\ +\infty)$ to an integer variable. Because the initial learning rate and dropout rate are consecutive floating-point values in the search space, the GPPSO's result can be used as the final result during search. For discrete variables, each of their available values corresponds to an integer

Table 1 Settings of mixed-variable encoding for CNN hyperparameters

Type	Name	Available choice	Search space	Initialization space
Continuous variable	Number of kernels in Conv layers	$\{1, 2, \cdots, +\infty\}$	$[1, +\infty)$	$[8, 128]$
	Number of neurons in FC layers	$\{1, 2, \cdots, +\infty\}$	$[1, +\infty)$	$[64, 512]$
	Value of the initial learning rate		$(0, 1)$	$(0, 1)$
	Value of the dropout rate		$[0, 1)$	$[0, 1)$
Discrete variable	Kernel size of Conv layers	$\{3\times3, 5\times5, 7\times7\}$	$\{0, 1, 2\}$	$\{0, 1, 2\}$
	Type of activation function	{ReLU, Sigmoid, Tanh, Ta-ReLU}	$\{0, 1, 2, 3\}$	$\{0, 1, 2, 3\}$
	Type of pooling layer	{Max pooling, average pooling}	$\{0, 1\}$	$\{0, 1\}$

CNN: convolutional neural network; Conv: convolutional; FC: fully connected value, and the search range is set according to all the integer values that can be taken. Taking the example of discrete variable encoding for AF, the specific encoding strategy is shown in Eq. (10):

$$\mathrm{af}=\begin{cases}\text{ReLU}, & \lceil X_{\mathrm{af}}\rceil\in[0,\ 1),\\ \text{Sigmoid}, & \lceil X_{\mathrm{af}}\rceil\in[1,\ 2),\\ \text{Tanh}, & \lceil X_{\mathrm{af}}\rceil\in[2,\ 3),\\ \text{Ta-ReLU}, & \lceil X_{\mathrm{af}}\rceil\in[3,\ 4),\end{cases}\tag{10}$$

where af denotes the final result of AF, X_{af} denotes the GPPSO search result, and $\lceil\cdot\rceil$ denotes taking the smallest integer that is no less than the search result.

3.2 HSA model

After encoding the CNN hyperparameters using a mixed-variable encoding scheme, the GPPSO algorithm starts to search for the optimal hyperparameter combinations by the HSA model. As shown in Fig. 4, the search process is divided into two parts, master and slave. In the first part, the candidate model is evaluated by constructing a GP model at multiple levels. In the second part, the target model of the next exploration is generated according to the previous search results through the PSO algorithm.

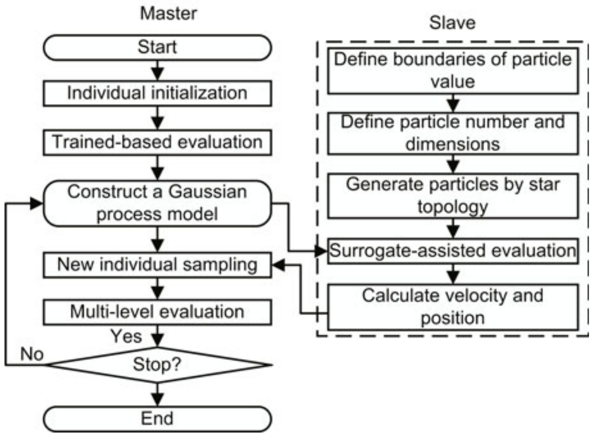

Master
Start
Individual initialization
Trained-based evaluation
Construct a Gaussian process model
New individual sampling
Multi-level evaluation
Yes
No
Stop?
End
Slave
Define boundaries of particle value
Define particle number and dimensions
Generate particles by star topology
Surrogate-assisted evaluation
Calculate velocity and position

Fig. 4 General flowchart of the hybrid surrogate-assisted model

3.2.1 Multi-level evaluation strategy based on GP

Because of the large size of CNN models and the large amount of training data, evaluating their fitness (i.e., classification loss or accuracy) is computationally expensive. Furthermore, even if there are enough computational resources and time to train the model until the accuracy converges, the optimal model obtained cannot guarantee, still having the best performance on the test set due to the difference between the validation data and test data. Therefore, unlike traditional OA, which obtains convergence accuracy through a lot of training as a fitness value, GPPSO is more efficient in evaluating and comparing different CNN candidate structures using a multi-level evaluation strategy.

GPPSO improves the computational expensive problem from two perspectives. First, GPPSO trains candidate models with few epochs during the search process to distinguish performance of different individuals. Second, a multi-level evaluation strategy based on GP is designed for the computational expensive problem in the search process. Each candidate is preliminarily computed by the GP model, and then some individuals with better evaluation results are trained to construct a new GP model. In addition, to improve the robustness of the algorithm, it is necessary to ensure that at least one individual of the group is evaluated by training. The pseudocode of the multi-level evaluation strategy based on the GP is given in Algorithm 1.

3.2.2 Individual generation strategy based on PSO

During the search process of GPPSO, it is necessary to evaluate individuals many times by the multi-level evaluation strategy and cause individuals to evolve to obtain the best outcome. Every time the multi-level evaluation strategy in Algorithm 1 is executed, an individual generation strategy is required to produce new candidates for the next subsequent generation. Traditional OA based on GP, such as the BO algorithm, defines an acquisition function to evaluate whether a sample can provide benefits for the GP model, and then determines whether it is a new individual to be explored. However, traditional acquisition functions generate new individuals based on existing individuals, which results in an incomplete search process, insufficient adaptability, and a limited exploration area.

Because the exploration performance of the above acquisition function may be not enough to generate new individuals accurately, an individual generation strategy based on PSO is proposed to increase performance in generating candidate individuals. The method is inspired by the PSO algorithm, candidate structures are treated as a group of particles, their initial value is treated as the position, and their velocity is calculated according to the GP

Algorithm 1 Multi-level evaluation strategy

```
Input: Dataset of training CNNs, $\mathcal{D}_{\text{train}}$;
    dataset of evaluating CNNs, $\mathcal{D}_{\text{eval}}$;
    architecture constructed by individuals, $\mathcal{P}_{\text{arch}}$;
    fitness of individuals in the architecture, $\text{fit}_{\text{arch}}$;
    set of individuals to be evaluated, $\mathcal{P}$;
    number of initial individuals, $N_{\text{I}}$;
    number of all individuals, $N_{\text{A}}$;
    number of training epochs, $T$
Output: Set of individuals evaluated, $\mathcal{P}_{\text{eval}}$;
    the fitness set of individuals evaluated, $\text{fit}_{\text{eval}}$
begin
  /* construct the initial GP model */
  Initialize $N_{\text{I}}$ individuals;
  Obtain $\mathcal{P}_{\text{arch}}$ and $\text{fit}_{\text{arch}}$ of the initial individuals by training;
  $\text{GP}_{\text{I}} \leftarrow$ build an initial GP model with $\mathcal{P}_{\text{arch}}$ and $\text{fit}_{\text{arch}}$;
  /* first level evaluation by the GP model */
  gfitness $\leftarrow$ fitness of $\mathcal{P}$ predicted by $\text{GP}_{\text{I}}$;
  afitness $\leftarrow$ the average fitness of $\text{fit}_{\text{arch}}$;
  $\mathcal{P}_{\text{eval}} \leftarrow$ empty set;
  $\text{fit}_{\text{eval}} \leftarrow$ empty set;
  ri $\leftarrow$ a random integer in $\{1, 2, \cdots, N_{\text{A}}\}$;
  /* second level evaluation by training */
  for each individual $\mathcal{P}_i$ in $\mathcal{P}$ do
    if gfitness $>$ afitness or ri $==$ $i$ then
      Net $\leftarrow$ build a candidate architecture according to the hyperparameters in $\mathcal{P}_i$;
      Net $\leftarrow$ train Net with $T$ epochs on $\mathcal{D}_{\text{train}}$;
      accuracy $\leftarrow$ validate Net on $\mathcal{D}_{\text{eval}}$;
      $\text{fit}_{\text{eval}} \leftarrow \text{fit}_{\text{eval}} \cup \{\text{accuracy}\}$;
      $P_{\text{eval}} \leftarrow P_{\text{eval}} \cup P_i$;
      $\text{GP}_{\text{I}} \leftarrow$ evolve $\text{GP}_{\text{I}}$ through $\text{fit}_{\text{eval}}$ and $\mathcal{P}_{\text{eval}}$;
    end if
  end for
end
```

model. After iterating through the PSO, the optimal offspring obtained is the new individual to be evaluated in Algorithm 1. The specific pseudocode is shown in Algorithm 2.

Algorithm 2 Individual generation strategy

```
Input: The initial Gaussian process model, $\text{GP}_{\text{I}}$;
    number of particles, $N$;
    fitness of individual $i$, $\text{fit}_i$;
    dimension of particles, $d$;
    number of generations, $n$;
    number of iterations, $T$
Output: The generated individual to be evaluated, $\mathcal{P}_{\text{new}}$
begin:
  /* initialize a group of particles with $d$ dimensions corresponding to hyperparameters in CNNs */
  Initialize a set of $N$ particles as $\mathcal{P}$;
  for each particle $\mathcal{P}_i$ in $\mathcal{P}$ do
    for each dimension $d$ of $\mathcal{P}_i$ do
      Randomly initialize the particle position $x_{id}$ within the given range;
      Randomly initialize the particle velocity $v_{id}$ within the given range;
    end for
  end for
  /* search individuals for multi-level evaluation by PSO */
  for $k = 1$ to $T$ do
    for each individual $\mathcal{P}_i$ in $\mathcal{P}$ do
      $\text{fit}_i \leftarrow$calculated fitness by $\text{GP}_{\text{I}}$ in Algorithm 1;
      if the fitness value is larger than $\text{pbest}_{id}$ in history then
        $\text{pbest}_{id} \leftarrow \text{fit}_i$;
      end if
      $\text{gbest}_d \leftarrow$ the particle with the best fitness value;
      Calculate velocity $v_{id}$ through Eq. (7);
      Calculate position $x_{id}$ through Eq. (8);
    end for
  end for
  $\mathcal{P}_{\text{new}} \leftarrow x_{id}$ of the last $n$ generations;
end
```

3.3 An improved AF

AFs play an essential role in the CNN learning process by fitting complex functions. AFs can introduce nonlinearity into CNNs, and provide the entire model with the ability to solve complex problems. During the early development stage of neural networks, traditional AFs are mainly S-type saturation functions, such as Sigmoid and Tanh (Fig. 5), which tend to cause vanishing gradients, resulting in training difficulties. Moreover, the derivative of these functions is complicated, and will cause computational expensive problems during the process of gradient back propagation in training. With the development of deep learning, Krizhevsky et al. (2017) proposed rectified linear units (ReLUs) (Fig. 5) to solve the problem of vanishing gradients and speed up network training. However, the ReLU function has some limitations. Because x value of the negative half-axis and its gradient value are both zero, neurons can lose the ability to transmit information during the training process.

In the automatic search for CNN

hyperparameters, fast convergence, low computational cost, and high accuracy should be achieved in the network evaluation process. Traditional AFs are often unable to meet these requirements simultaneously. Therefore, a new AF has been designed as follows:

$$\text{Ta-ReLU} = \begin{cases} x, & x < 0, \\ \alpha \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}, & x \geq 0. \end{cases} \tag{11}$$

Ta-ReLU is a nonlinear and differentiable function with a sensitivity factor α that controls the function's negative semi-axis activity to the inputs. When $\alpha = 0$, Ta-ReLU reduces to ReLU. The function is discontinuous at $x = 0$ and is sensitive to inputs on the positive semi-axes and to the inputs close to 0 on the negative semi-axes. When the input changes, the output also changes significantly. However, Ta-ReLU is insensitive to other inputs; that is, the output does not change in correspondence with the input or produces only a small change. Given these properties, Ta-ReLU meets the required conditions of an AF. A comparison between Ta-ReLU and traditional AFs is shown in Fig. 5.

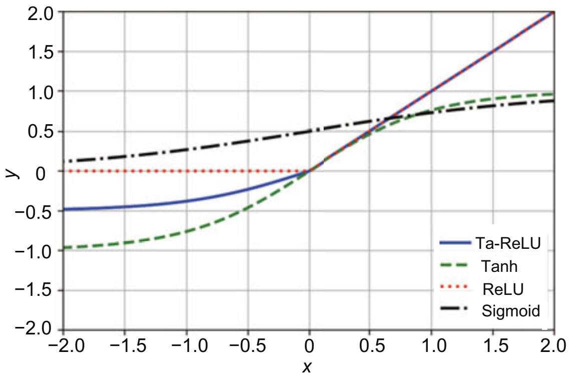


Fig. 5 Comparison of different activation functions

Fig. 5 shows that Ta-ReLU has several advantages compared to traditional AFs. First, it requires less computation and has a simpler derivative, making it more efficient and easier to implement. Additionally, Ta-ReLU has a larger gradient than ReLU at the x-negative half-axis near zero, ensuring that negative output values are not ignored in the network. This leads to higher search speeds during training and results in better performance in the trained model. These advantages will be further verified in Section 4.

3.4 Complete algorithm

The complete pseudocode of GPPSO is shown in Algorithm 3 and is detailed as follows:

Step 1: GPPSO initializes a set of individuals through a mixed-variable encoding scheme (Section 3.1) and evaluates these initial points (IPs) using training to determine their accuracy. Then, according to the accuracy, the best individuals and their fitness values will be evaluated and stored.

Step 2: for the first loop, setting search_iteration to zero. According to the evaluated points in step 1, the initial GP model is constructed. It should be noted that the GP model is dynamically updated during the cycle, and

Algorithm 3 Complete algorithm

```
Input: Input dataset, $\mathcal{D}_\mathrm{I}$;
    number of initial individuals, $N_\mathrm{I}$;
    number of sampling individuals, $N_\mathrm{S}$;
    the initial Gaussian process model, GP$_\mathrm{I}$;
    the initialized continuous variables, cx;
    the initialized discrete variables, dx;
    number of iterations, $T$
Output: The architecture searched by GPPSO, Net
begin:
$\mathcal{D}_\mathrm{train}$, $\mathcal{D}_\mathrm{valid}$ ← split $\mathcal{D}_\mathrm{I}$ by 5 : 1;
cx, dx ← perform mixed-variable encoding strategy;
$P$ ← combine cx and dx;
fitness ← ∅;
for each particle $\mathcal{P}_i$ in $\mathcal{P}$ do
    Net ← build a CNN according to $\mathcal{P}_i$;
    Net ← train Net with $S$ epochs on $\mathcal{D}_\mathrm{train}$;
    accuracy ← test Net on $\mathcal{D}_\mathrm{valid}$;
    fitness ← fitness ∪ {accuracy};
end for
$P_\mathrm{best}$, fit$_\mathrm{best}$ ← the best individual and fitness, respectively;
$P_\mathrm{arch}$ ← $P$, fit$_\mathrm{arch}$ ← fitness;
search_iteration ← 0;
while search_iteration < $T$ do
    GP$_\mathrm{I}$ ← construct the initial GP according to $N_\mathrm{I}$;
    $P$ ← sample $N_\mathrm{S}$ new individuals as candidate points through Algorithm 2;
    $P_\mathrm{eval}$, fit$_\mathrm{eval}$ ← perform Algorithm 1 on $P$;
    $P_\mathrm{arch}$ ← $P_\mathrm{arch}$ ∪ $P_\mathrm{eval}$, fit$_\mathrm{arch}$ ← fit$_\mathrm{arch}$ ∪ fit$_\mathrm{eval}$;
    $P_\mathrm{best}$, fit$_\mathrm{best}$ ← the best individual and its fitness, respectively;
    search_iteration ← search_iteration + 1;
end while
Net ← train the best CNN on $\mathcal{D}_\mathrm{I}$;
end
```

the GP model will evolve each time based on the evaluation results of candidate points unless the iterations are reached. In the process of candidate point evaluations, a multi-level evaluation strategy (Algorithm 1) is adopted to solve the computational expensive problem and improve the search efficiency.

Step 3: during the cycle, new candidate points need to be generated for evaluation through an individual generation strategy based on PSO (Algorithm 2). First, the boundaries, numbers, and dimensions of particles are set to generate particles through a star topology, and then the fitness of the particles is evaluated using the GP model. After several evolutionary iterations, the optimal particles are obtained for multiple evaluations.

Step 4: if the stop iterations are not met, the algorithm goes back to step 2 and the procedure is repeated; otherwise, the CNN constructed by the corresponding optimal hyperparameters is trained on the entire dataset for the best performance and outputs the CNN model as the final output. It should be noted that the Ta-ReLU designed in Section 3.3 is used as the AF of the Conv layer in the GPPSO algorithm. This choice aims to accelerate the convergence and ensure the model performance.

4 Experimental results

To verify the effectiveness and efficiency of the proposed GPPSO, a series of experiments were designed and conducted. First, the relevant settings of the experiments are introduced in Sections 4.1–4.3, including the datasets and evaluation metrics, the compared state-of-the-art methods, and the parameter settings of the proposed algorithm. Next, the proposed GPPSO is compared with advanced algorithms on the CIFAR datasets to investigate its theoretical effectiveness. Finally, a metal fracture (MF) diagnosis case in a real-world industrial scenario is presented to prove the feasibility of GPPSO in practical industrial applications.

4.1 Datasets and metrics

To evaluate the performance of the proposed GPPSO, three datasets were used for experimentation in this subsection: CIFAR-10, CIFAR-100, and the MF dataset. Two different CIFAR datasets (Fig. 6) were used to verify the theoretical effectiveness of the proposed algorithm, because they provided varying difficulties for image classification tasks. Both CIFAR datasets contained 50 000 training images and 10 000 test images, where each image has 32×32 pixels and three channels. The MF dataset was used in the experiments to demonstrate the effectiveness of GPPSO in practical applications. The MF dataset consisted of 1500 scanning electron microscopy images of three MF categories, with the resolution of 512×512. The initial dataset was expanded from 1500 to 7500 by random angle rotation, proportional scaling, horizontal and vertical flipping, and a training-to-test set ratio of 4:1. Fig. 7 shows the examples of the MF dataset.

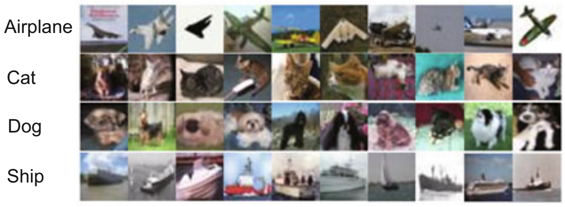


Fig. 6 Examples of the CIFAR datasets

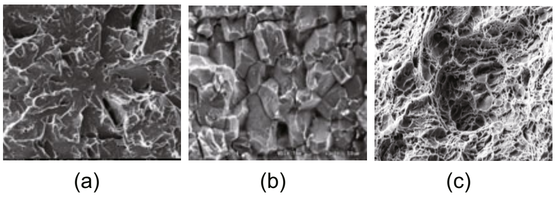


Fig. 7 Examples of the metal fracture dataset: (a) cleavage fracture; (b) intergranular fracture; (c) dimple fracture

During the experiments, three aspects were considered to compare the algorithm's strength: model performance, model size, and model training time. In terms of these aspects, three popular metrics were adopted: the classification accuracy on the test dataset, the number of parameters in the model, and the time consumed. It should be noted that the time required for training is different for each computer due to the different hardware configurations even with the same graphics processing unit (GPU). Therefore, GPU days were used only as a reference index of other methods; the training time in our experimental environment was used as a horizontal comparison index.

4.2 Compared methods

To demonstrate the effectiveness of the proposed GPPSO, a series of state-of-the-art algorithms were used for comparison based on the evaluation metrics in Section 4.1. The compared algorithms can be divided into three categories: manually designed CNNs, non-OA-based methods, and OA-based methods.

Specifically, the manually designed CNNs include the famous architectures maxout (Goodfellow et al., 2013), network in network (Lin et al., 2013), ALL-CNN (Springenberg et al., 2014), VGGNet (Simonyan and Zisserman, 2014), highway network (Srivastava et al., 2015), FractalNet (Larsson et al., 2016), ResNet (He et al., 2016), and DenseNet (Huang et al., 2017), which have shown state-of-the-art results in image classification tasks. For the non-OA-based methods, some representative algorithms are adopted, such as BO (Snoek et al., 2012), Auto-Keras (AK) (Jin et al., 2019), NAS (Zoph and Le, 2017), MetaQNN (Baker et al., 2017), EAS (Cai et al., 2018), and Block-QNN-S (Zhong et al., 2018). As for the OA-based algorithms, PSO, hierarchical evolution (Liu et al., 2017), large-scale evolution (Real et al., 2017), genetic CNN (Xie and Yuille, 2017), CGP-CNN (Suganuma et al., 2017), CNN-GA (Sun et al., 2020a), AE-CNN (Sun et al., 2020b), AE-CNN+E2EPP (Sun et al., 2020c), and SHEDA-CNN (Li JY et al., 2023) are selected to compare with GPPSO. Because of the better performance of OA-based methods in automatic search algorithms, these methods are ideal for comparison with GPPSO. Due to the expensive computational cost and different experiment environments, some final results of the literature were cited directly for comparison, which is also a convention in the deep learning study. In addition, to ensure the effectiveness of the comparisons, the classical algorithms in each category, such as ResNet50, BO algorithm, and PSO algorithm, were implemented in the experimental configuration of this study to make a direct comparison with various indicators of GPPSO.

4.3 Algorithm settings

A 20-depth version of ResNet was used as the basic model of GPPSO in the experiments on the CIFAR datasets. There were 19 Conv layers and one pooling layer in ResNet20 for feature extraction, and one FC layer and a final softmax layer for image classification. According to the ResNet20 structure and mixed-variable encoding strategy (Section 3.1), the model was encoded using 22-dimensional continuous variables and 32-dimensional discrete variables. The specific setting is shown in Table 1.

The GPPSO parameters were set as follows: first, considering the balance between the computational cost and model performance, the number of IPs selected to build the GP model was 20. Second, the minimum number of iterations was set to 30, after that the search will finish when a better CNN cannot be found in three generations. In addition, the candidate model obtained through the search process will be trained for two epochs as a preliminary evaluation of performance. As for the individual generation category, the parameters were configured according to the default values in PySwarms (Miranda, 2018): $c_1 = 0.5$, $c_2 = 0.3$, and $w = 0.9$. Finally, for the CNN training category, the optimizer was set as Adam, and the initial learning rate was set to 1×10^{-3}.

In addition, data augmentation was applied before training using the conventional method through Keras. The experiments were conducted and evaluated using the Python programming language with the TensorFlow (Abadi et al., 2016) deep learning library on a 2.30 GHz Intel Core i7-12700H CPU and 16 GB memory NVIDIA RTX 3080Ti graphics card.

4.4 Comparisons with state-of-the-art methods

The effectiveness of the GPPSO algorithm was evaluated using two comparisons, as shown in Tables 2 and 3. The first part represents the comparison between GPPSO and state-of-the-art algorithms. Then in the second part, a comparison between GPPSO and the basic algorithms (BO and PSO) is presented.

As shown in Table 2, the GPPSO required 0.04 GPU days to achieve 95.26% classification accuracy with 5.26×10^6 parameters on the CIFAR-10 dataset and to achieve 76.36% classification accuracy with 4.44×10^6 parameters on the CIFAR-100 dataset. Compared with state-of-the-art algorithms, GPPSO shows the great performance in both classification accuracy and search time. First, when compared with the manually designed models, GPPSO achieves at least 0.53% and at most

Table 2 Comparisons with the manually designed CNNs, non-OA-based methods, and OA-based methods on CIFAR-10 and CIFAR-100 datasets

Method	Peer	Accuracy (%)		Number of parameters	Search time
	competitor	CIFAR-10	CIFAR-100	($\times 10^6$)	(GPU days)
Manually designed CNN	VGGNet	93.34	71.95	20.04	
	ResNet	93.57	78.84	1.7	
	DenseNet	94.76	75.58	0.8	
	Maxout	90.70	61.40		
	Network in network	91.19	64.32		
	Highway network	92.40	67.66		
	ALL-CNN	92.75	66.29	1.3	
	FractalNet	94.76	77.51	22.9	
Non-OA-based method	BO	92.91	66.47	4.70	0.02
	AK	88.56		1.7	
	NAS	93.99		0.8	22 400
	MetaQNN	93.08	72.86		100
	EAS	95.77		23.4	10
	Block-QNN-S	95.62	79.35	6.1	90
OA-based method	PSO	84.17	54.25	4.65/4.70	0.05/0.19
	Large-scale evolution	94.60	77.00	5.4/40.4	2750
	Hierarchical evolution	96.37			300
	CGP-CNN	94.02		1.68	27
	CNN-GA	96.78	79.47	2.9/4.1	35/40
	AE-CNN	95.70	79.15		27/36
	AE-CNN+E2EPP	94.70	77.98		8.5
	SHEDA-CNN	96.36	78.84	10.88/18.64	0.58/0.97
	Genetic CNN	92.90	70.97		817
Our method	GPPSO	95.26	76.36	5.26/4.44	0.04

The results before and after "/" are based on the CIFAR-10 and CIFAR-100 datasets, respectively

5.03% classification accuracy improvement on the CIFAR-10 dataset. As for the CIFAR-100 dataset, GPPSO achieves at most 24.36% accuracy improvement over maxout, and just 3.15% lower than that of the 20-layer version of ResNet, proving that instead of trying deeper and more complex network models, optimizing the hyperparameters of existing well-performing CNN models using GPPSO can obtain outstanding results. As for the number of parameters, the model searched by GPPSO generally has fewer parameters than the directly connected networks such as VGGNet, but more than cross-layer connection models such as ResNet due to the concatenate and other operations. Second, compared with the non-OA-based methods, GPPSO still achieves better performance. On the CIFAR-10 dataset, GPPSO achieves better classification accuracy, and is better than BO, AK, MetaQNN, and NAS by 2.53%, 7.56%, 2.34%, and 1.35%, respectively. As for EAS and Block-QNN-S, GPPSO can achieve similar classification accuracy, but greatly reduces the search time. This means that GPPSO needs less time and computational cost to search for better CNN models. For example, for the non-OA-based methods, NAS needs 22 400 GPU days to find a good CNN model and EAS requires at least 10 GPU days, but GPPSO needs only 0.04 GPU days. On the CIFAR-100 dataset, the accuracy of GPPSO is a little worse than that of Block-QNN-S, but still achieves good performance of 76.36%. GPPSO has more model parameters than the NAS and AK methods, but fewer than the other algorithms. Finally, compared with the OA-based methods, GPPSO still obtains competitive results. As shown in Table 2, for classification accuracy, GPPSO ranks the fifth among the 10 algorithms based on the CIFAR-10 dataset, with up to 13.18% higher accuracy than that of PSO and 1.57% lower accuracy than that of the first algorithm (CNN-GA). As for CIFAR-100, GPPSO achieves good performance and ranks the sixth among the eight algorithms, 40.76% better than that of the last and only 3.91% lower than the first. However, compared with CNN-GA, GPPSO cost only about 0.11% and 0.10% GPU days on CIFAR-10 and CIFAR-100, respectively. When compared with the SHEDA-CNN method, GPPSO not only reduces the search time by 93.10% and 95.88%, but also reduces the model size by 51.65% and 76.18%, on CIFAR-10 and CIFAR-100 datasets respectively. This means that GPPSO not only achieves good recognition performance, but also greatly reduces the search time and model size. These advantages provide strong feasibility for the practical applications of deep learning. The above comparison results show the effectiveness and efficiency of GPPSO.

Table 3 presents a set of comparisons between GPPSO, ResNet20, BO algorithm, and the PSO algorithm to show the outstanding performance. It should be noted that all the results in Table 3 were generated in the experimental environment of this study, and algorithm_ac denotes that Ta-ReLU in Section 3.3 is in the search space. Because the experiments were carried out under the same hardware conditions, minute is used as the benchmark index for efficiency comparisons instead of GPU days. It can be seen in Table 3 that GPPSO_ac achieves the best test accuracy on CIFAR-10 and CIFAR-100, where it is 3.26% and 10.73% better than that of the basic model ResNet20 on CIFAR-10 and CIFAR-100, respectively. By using the automatic search method for manually designed CNN ResNet20, the accuracy will be improved by the BO algorithm and reduced by the PSO algorithm, whose test accuracy is lower than that of GPPSO. The number of parameters of the manually designed CNN is significantly smaller compared to those of the automatically searched models. Furthermore, the numbers of parameters in the automatically searched CNNs are in a similar order of magnitude (10^6). We think that this is due to the concatenate layer and other structures in the automatic search networks. As for the search time, the BO algorithm needs least 17 min to search CNNs, GPPSO takes longer time than BO, 42 min on average, and PSO takes the longest time at an average of 100 min on CIFAR-10 and 202 min on CIFAR-100. After searching for the CNNs, we train them for 200 epochs to obtain the final models. The accuracy-loss curves of training are shown in Fig. 8. It can be seen that the convergence speed of the model on the CIFAR-100 dataset is lower than that on CIFAR-10, and the error value is higher than that of CFIAR-10, which indicates that the 20-layer basic model has limited capability for large-scale output categories. For the model with Ta-ReLU function, the convergence rate is higher in the first 20 epochs (e.g., Figs. 8m and 8k), and GPPSO_ac has the best recognition accuracy on CIFAR-10 and CIFAR-100, which proves the effectiveness of the designed AF.

In conclusion, the comparisons with the basic algorithms of GPPSO and state-of-the-art algorithms prove the effectiveness and efficiency of GPPSO.

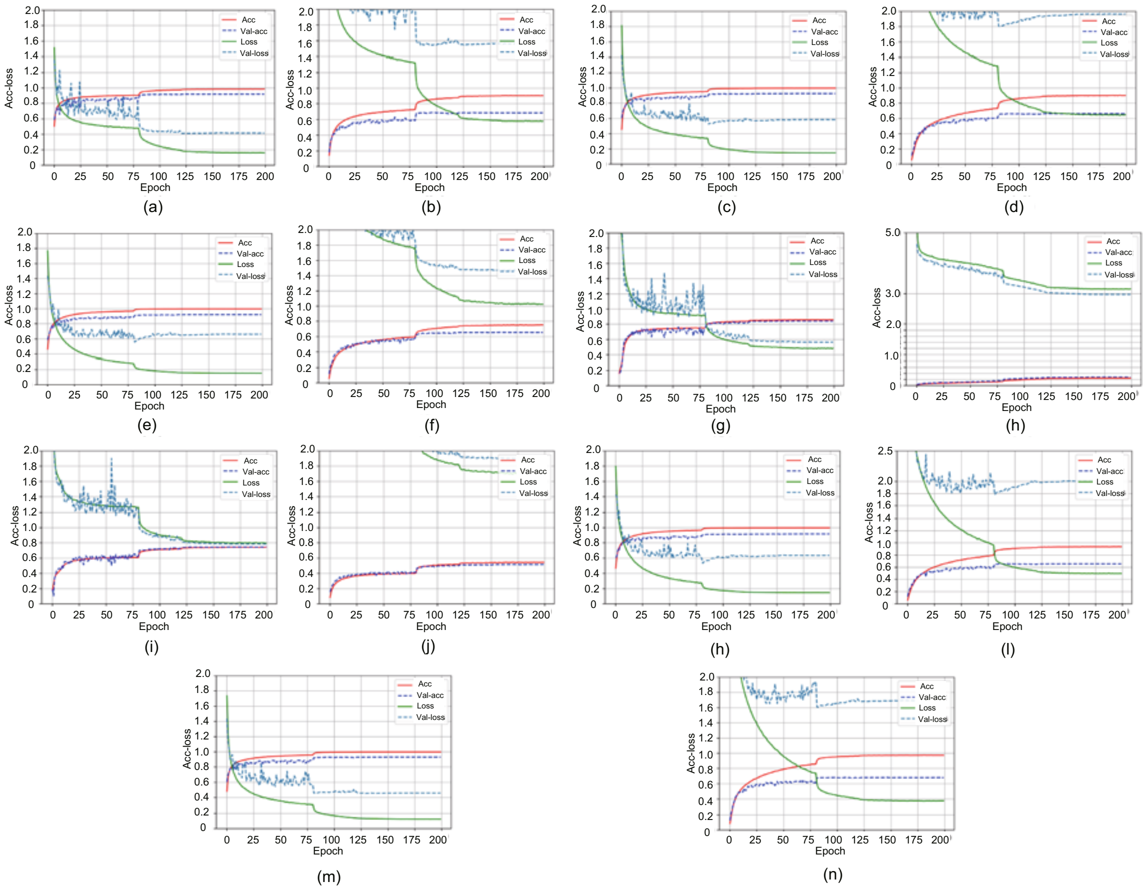
4.5 Ablation experiments

In the GPPSO algorithm, the initial GP model will be constructed from a set of individuals, so the number of individuals may affect the GPPSO performance. To verify the influence of the number of IPs, GPPSO is compared with variants using different numbers of IPs on CIFAR-10. During the experiments, the range of the number of IPs was set from 10 to 50, and the sampling interval was set as 10; the experimental results are shown in Table 4. It can be seen that with different numbers of IPs, the classification accuracy of all searched models is >92.50%, and the search time increased almost linearly, with an additional 5 min required for each sampling interval increase. Furthermore, the classification accuracy initially increased and then decreased with an increase in the number of IPs, with peak performance achieved when the number of IPs was 20. This indicates that the model searched by GPPSO does not achieve higher performance with an increased number of IPs. We think the reason is that the random IPs cannot precisely describe the trends of Gaussian regression model, and the points obtained by acquisition function in the search process are more meaningful. In conclusion, the number of IPs can influence the search effectiveness of GPPSO, but the GPPSO is not that sensitive with the increase of the number of IPs, and the performance is not always improved with an increased number of IPs.

Another important parameter in GPPSO is the particle number (PN) in PSO. To investigate the influence of PN, a set of comparisons are given in Table 5. It can be seen that GPPSO is compared with its variants using PNs from 10 to 100; with the PNs increase, the classification accuracy exhibits similarity, when PN=30, 70, and 100, the accuracy can achieve >93%, and when PN=50, the performance of GPPSO is at its best (>95%). That is because GPPSO is not sensitive to the PN. In addition, with an increase in the PN, the GPPSO's search time increases as well. Therefore, considering

Table 3 Comparisons with the basic algorithms of GPPSO on CIFAR-10 and CIFAR-100 datasets

Method	CIFAR-10					CIFAR-100				
	Test accuracy (%)	Training accuracy (%)	Number of parameters ($\times10^6$)	Search time (min)	Training time (min)	Test accuracy (%)	Training accuracy (%)	Number of parameters ($\times10^6$)	Search time (min)	Training time (min)
Manual (ResNet20)	92.25	98.71	0.38		116	68.96	90.78	0.40		111
BO	92.91	99.92	4.70	20	274	66.47	90.62	4.70	20	279
BO_ac	92.26	99.96	5.26	17	367	65.64	75.60	3.57	23	326
PSO	84.17	86.09	4.65	50	258	54.25	28.20	4.70	192	265
PSO_ac	74.57	74.75	6.17	150	361	51.77	54.77	4.74	212	296
GPPSO	91.85	99.95	4.77	45	270	65.91	94.00	4.74	33	330
GPPSO_ac	95.26	99.96	5.26	39	261	76.36	97.65	4.44	39	304



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (h) (l) (m) (n)

Fig. 8 Accuracy-loss curves of training and validation on the CIFAR-10 and CIFAR-100 datasets: (a) ResNet20-10; (b) ResNet20-100; (c) BO-10; (d) BO-100; (e) BO-ac-10; (f) BO-ac-100; (g) PSO-10; (h) PSO-100; (i) PSO-ac-10; (j) PSO-ac-100; (k) GPPSO-10; (l) GPPSO-100; (m) GPPSO-ac-10; (n) GPPSO-ac-100

the performance and the computation costs, PN=50 is suitable and is recommended for GPPSO.

In the GPPSO process, after evaluating by the surrogate-assisted model, qualified models will be evaluated based on training. Hence, to reduce the time consumption and the computational cost, only epoch T will be selected for training, which will be a key issue for the effectiveness of the GPPSO. Therefore, an experiment between different epoch T values was carried out to study the influence on performance. Considering the hardware performance of the experiments and the time complexity of the

Table 4 Comparion with different numbers of initial points on CIFAR-10

Number of initial points	Accuracy (%)	Search time (min)
10	93.68	33
20	95.26	39
30	93.76	44
40	93.04	49
50	92.75	55

Table 5 Comparison with different particle numbers (PNs) on CIFAR-10

PN	Accuracy (%)	Search time (min)
10	94.12	16
30	93.57	28
50	95.26	39
70	93.76	51
100	93.64	65

practical applications, GPPSO was used to search for optimal CNNs by training epoch T from 1 to 5 in the experiments. Table 6 shows the experimental results of different epoch T values on CIFAR-10. To intuitively compare the impact of epoch T on model performance, as shown in Eq. (12), a comparison method is designed to measure it:

$$\mathrm{CE}_T = \frac{\mathrm{acc}_{i+1} - \mathrm{acc}_i}{\mathrm{time}_{i+1} - \mathrm{time}_i}, \tag{12}$$

where CE_T denotes the ratio between the accuracy (acc) difference and the search time difference. It can be seen that when the epochs increase from 1 to 2, the classification accuracy is improved by 2.75%, and CE_T is 0.17 when T=2. Then, as the number of epochs changes from 2 to 5, the value of CE_T changes from 0.03 to 0.01, and approximately equals 0 when T=4. This means that continuing to increase the epochs will not improve the performance obviously and will consume a lot of computational resources. In conclusion, the epoch T=2 of training in the search process will result in the maximum GPPSO efficiency.

Table 6 Comparison with different epochs on CIFAR-10

Epoch	Accuracy (%)	Search time (min)	CE_T
1	92.71	24	
2	95.26	39	0.17
3	95.67	52	0.03
4	95.68	67	≈ 0
5	95.87	81	0.01

4.6 Application on MF diagnosis

To prove the effectiveness of GPPSO in real-world problems, an application concerning MF diagnosis in industrial scenarios is presented in this subsection. Metal materials are essential in modern industrial fields such as aerospace, transportation, and metallurgical manufacturing. In a complex environment, metal materials in service cause failure accidents such as fracture, corrosion, and fatigue, which cause heavy economic losses and casualties. Therefore, to achieve accurate MF recognition automatically and efficiently, AI methods such as CNNs will be used, which are suitable for testing the performance of GPPSO.

In the experiments, a deep learning metal fracture classification (DMFC) model is designed to recognize MFs. The model structure is shown in Fig. 9.

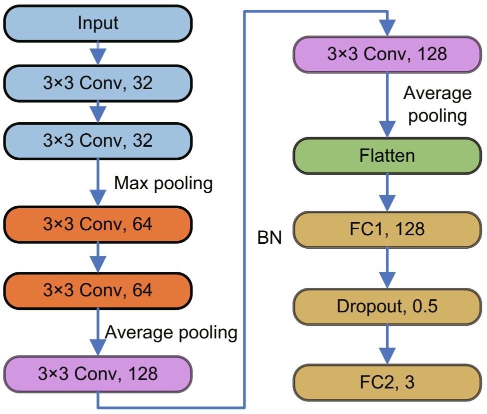


Fig. 9 DMFC model structure (FC: fully connected; BN: batch normalization)

The Conv kernels in the DMFC model are all 3 × 3, and three pooling layers are constructed using one max pooling layer and two average pooling layers. A flatten layer is added to reduce the dimension of the output feature maps produced by the last Conv layer. This serves as a transition between the Conv layer and the FC layer. The first FC layer has 128 neurons, followed by a batch normalization (BN) layer and a dropout layer. The output layer has three neurons, corresponding to three types of MFs. DMFC is the basic model of MF recognition task. The GPPSO algorithm searches the Conv kernel size, Conv kernel number, pooling layer type, AF type, and other hyperparameters in the DFMC to obtain the GPPSO-DMFC.

To test the effectiveness, comparisons among VGG, ResNet, DenseNet, DMFC, and GPPSO-DMFC are given in Table 7. The effectiveness and efficiency of the algorithms are measured by accuracy and training time, respectively. As shown in Table 7, the proposed DMFC model achieved an accuracy of 94.94% and a training time of 11 s/epoch. When using state-of-the-art methods, the recognition accuracy of MF is significantly improved. However, the training time is increased. For example, DenseNet achieved an accuracy of 98.03%, but required training time of 93 s/epoch. After using the GPPSO algorithm to search hyperparameters for the DMFC model, the result achieved the highest accuracy of 98.16% and the shortest training time of 9 s/epoch, indicating the effectiveness and efficiency of GPPSO. Therefore, this application shows that the GPPSO has potential for solving real-world tasks.

Table 7 Results of the metal fracture diagnosis

Method	Accuracy (%)	Time (s/epoch)
DMFC	94.94	11
VGG16	98.02	25
VGG19	97.72	29
ResNet50	96.97	24
DenseNet110	98.03	93
GPPSO-DMFC	98.16	9

5 Conclusions

In this paper, a novel method, GPPSO, was proposed for efficient optimization of CNN hyperparameters. First, the GPPSO encoded different types of hyperparameters in CNNs using a mixed-variable encoding strategy to deal with the mixed-variable problem. Then, the HSA model based on the GP and PSO was designed to save computational costs. Finally, a novel AF, Ta-ReLU, was suggested to improve the model performance and ensure convergence rate. Experiments on two benchmark datasets have proven the efficiency of GPPSO. Furthermore, a series of ablation experiments have been used to investigate the parameter sensitivity. We also presented a case study of industrial scenarios to demonstrate the effectiveness of GPPSO in real-world tasks. For further work, we plan to (1) search for the CNN hyperparameters and architectures jointly and (2) design a more efficient OA for obtaining CNNs to handle engineering problems with practical applications.

Contributors

Han YAN designed the research and performed the experiments. Han YAN and Chongquan ZHONG implemented the software and drafted the paper. Yuhu WU, Liyong ZHANG, and Wei LU revised and finalized the paper.

Compliance with ethics guidelines

Han YAN, Chongquan ZHONG, Yuhu WU, Liyong ZHANG, and Wei LU declare that they have no conflict of interest.

Data availability

Due to the nature of this research, all authors of this paper did not agree for their data to be shared publicly, so supporting data are not available.

References

Abadi M, Agarwal A, Barham P, et al., 2016. TensorFlow: large-scale machine learning on heterogeneous distributed systems. https://arxiv.org/abs/1603.04467

Alvarez-Rodriguez U, Battiston F, de Arruda GF, et al., 2021. Evolutionary dynamics of higher-order interactions in social networks. *Nat Hum Behav*, 5(5):586-595. https://doi.org/10.1038/s41562-020-01024-1

Alzubaidi L, Zhang JL, Humaidi AJ, et al., 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data*, 8(1):53. https://doi.org/10.1186/s40537-021-00444-8

Baker B, Gupta O, Naik N, et al., 2017. Designing neural network architectures using reinforcement learning. https://arxiv.org/abs/1611.02167

Cai H, Chen TY, Zhang WN, et al., 2018. Efficient architecture search by network transformation. Proc 32nd AAAI Conf on Artificial Intelligence, p.2787-2794. https://doi.org/10.1609/aaai.v32i1.11709

Chen ZG, Zhan ZH, Kwong S, et al., 2022. Evolutionary computation for intelligent transportation in smart cities: a survey. *IEEE Comput Intell Mag*, 17(2):83-102. https://doi.org/10.1109/MCI.2022.3155330

Darwish A, Hassanien AE, Das S, 2020. A survey of swarm and evolutionary computing approaches for deep learning. *Artif Intell Rev*, 53(3):1767-1812. https://doi.org/10.1007/s10462-019-09719-2

Fernandes FE, Yen GG, 2021. Automatic searching and pruning of deep neural networks for medical imaging diagnostic. *IEEE Trans Neur Netw Learn Syst*, 32(12):5664-5674. https://doi.org/10.1109/TNNLS.2020.3027308

Fielding B, Lawrence T, Zhang L, 2019. Evolving and ensembling deep CNN architectures for image classification. Int Joint Conf on Neural Networks, p.1-8. https://doi.org/10.1109/IJCNN.2019.8852369

Goodfellow IJ, Warde-Farley D, Mirza M, et al., 2013. Maxout networks. Proc 30th Int Conf on Machine Learning, p.1319-1327.

Grigorescu S, Trasnea B, Cocias T, et al., 2020. A survey of deep learning techniques for autonomous driving. *J Field Robot*, 37(3):362-386. https://doi.org/10.1002/rob.21918

Guo H, Zhang W, Nie XY, et al., 2022. High-speed planar imaging of OH radicals in turbulent flames assisted by deep learning. *Appl Phys B*, 128(3):52. https://doi.org/10.1007/s00340-021-07742-2

He KM, Zhang XY, Ren SQ, et al., 2016. Deep residual learning for image recognition. IEEE Conf on Computer Vision and Pattern Recognition, p.770-778. https://doi.org/10.1109/CVPR.2016.90

Huang G, Liu Z, van der Maaten L, et al., 2017. Densely connected convolutional networks. 30th IEEE Conf on Computer Vision and Pattern Recognition, p.2261-2269. https://doi.org/10.1109/CVPR.2017.243

Jiang WW, Luo JY, 2022. Graph neural network for traffic forecasting: a survey. *Expert Syst Appl*, 207:117921. https://doi.org/10.1016/j.eswa.2022.117921

Jin HF, Song QQ, Hu X, 2019. Auto-Keras: an efficient neural architecture search system. Proc 25th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining, p.1946-1956. https://doi.org/10.1145/3292500.3330648

Krizhevsky A, Sutskever I, Hinton GE, 2017. ImageNet classification with deep convolutional neural networks. *Commun ACM*, 60(6):84-90. https://doi.org/10.1145/3065386

Larsson G, Maire M, Shakhnarovich G, 2016. FractalNet: ultra-deep neural networks without residuals. https://arxiv.org/abs/1605.07648

Li JY, Zhan ZH, Wang C, et al., 2020. Boosting data-driven evolutionary algorithm with localized data generation. *IEEE Trans Evol Comput*, 24(5):923-937. https://doi.org/10.1109/TEVC.2020.2979740

Li JY, Zhan ZH, Liu RD, et al., 2021. Generation-level parallelism for evolutionary computation: a pipeline-based parallel particle swarm optimization. *IEEE Trans Cybern*, 51(10):4848-4859. https://doi.org/10.1109/TCYB.2020.3028070

Li JY, Zhan ZH, Zhang J, 2022. Evolutionary computation for expensive optimization: a survey. *Mach Intell Res*, 19(1):3-23. https://doi.org/10.1007/s11633-022-1317-4

Li JY, Zhan ZH, Xu J, et al., 2023. Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks. *IEEE Trans Neur Netw Learn Syst*, 34(5):2338-2352. https://doi.org/10.1109/TNNLS.2021.3106399

Li X, Lai SQ, Qian XM, 2022. DBCFace: towards pure convolutional neural network face detection. *IEEE Trans Circ Syst Video Technol*, 32(4):1792-1804. https://doi.org/10.1109/TCSVT.2021.3082635

Lin M, Chen Q, Yan SC, 2013. Network in network. https://arxiv.org/abs/1312.4400

Liu HX, Simonyan K, Vinyals O, et al., 2017. Hierarchical representations for efficient architecture search. https://arxiv.org/abs/1711.00436

Miranda LJV, 2018. PySwarms: a research toolkit for particle swarm optimization in Python. *J Open Source Softw*, 3(21):433. https://doi.org/10.21105/joss.00433

Poli R, Kennedy J, Blackwell T, 2007. Particle swarm optimization. *Swarm Intell*, 1(1):33-57. https://doi.org/10.1007/s11721-007-0002-0

Real E, Moore S, Selle A, et al., 2017. Large-scale evolution of image classifiers. https://arxiv.org/abs/1703.01041v2

Simonyan K, Zisserman A, 2014. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/abs/1409.1556

Snoek J, Larochelle H, Adams RP, 2012. Practical Bayesian optimization of machine learning algorithms. https://arxiv.org/abs/1206.2944

Springenberg JT, Dosovitskiy A, Brox T, et al., 2014. Striving for simplicity: the all convolutional net. https://arxiv.org/abs/1412.6806v3

Srivastava RK, Greff K, Schmidhuber J, 2015. Highway networks. https://arxiv.org/abs/1505.00387

Suganuma M, Shirakawa S, Nagao T, 2017. A genetic programming approach to designing convolutional neural network architectures. Proc Genetic and Evolutionary Computation Conf, p.497-504. https://doi.org/10.1145/3071178.3071229

Sun YN, Xue B, Zhang MJ, et al., 2019. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Trans Neur Netw Learn Syst*, 30(8):2295-2309. https://doi.org/10.1109/TNNLS.2018.2881143

Sun YN, Xue B, Zhang MJ, et al., 2020a. Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Trans Cybern*, 50(9):3840-3854. https://doi.org/10.1109/TCYB.2020.2983860

Sun YN, Xue B, Zhang M, et al., 2020b. Completely automated CNN architecture design based on blocks. *IEEE Trans Neur Netw Learn Syst*, 31(4):1242-1254. https://doi.org/10.1109/TNNLS.2019.2919608

Sun YN, Wang HD, Xue B, et al., 2020c. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Trans Evol Comput*, 24(2):350-364. https://doi.org/10.1109/TEVC.2019.2924461

Tulbure AA, Tulbure AA, Dulf EH, 2022. A review on modern defect detection models using DCNNs-deep convolutional neural networks. *J Adv Res*, 35:33-48. https://doi.org/10.1016/j.jare.2021.03.015

Wang B, Sun YN, Xue B, et al., 2018. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. IEEE Congress on Evolutionary Computation, p.1-8. https://doi.org/10.1109/CEC.2018.8477735

Wang B, Xue B, Zhang MJ, 2020. Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks. IEEE Congress on Evolutionary Computation, p.1-8. https://doi.org/10.1109/CEC48606.2020.9185541

Wang YQ, Li JY, Chen CH, et al., 2022. Scale adaptive fitness evaluation-based particle swarm optimisation for hyperparameter and architecture optimisation in neural networks and deep learning. *CAAI Trans Intell Technol*, early access. https://doi.org/10.1049/cit2.12106

Wu SH, Zhan ZH, Tan KC, et al., 2023. Orthogonal transfer for multitask optimization. *IEEE Trans Evol Comput*, 27(1):185-200.
https://doi.org/10.1109/TEVC.2022.3160196

Wu T, Shi J, Zhou DY, et al., 2019. A multi-objective particle swarm optimization for neural networks pruning. IEEE Congress on Evolutionary Computation, p.570-577.
https://doi.org/10.1109/CEC.2019.8790145

Xie LX, Yuille A, 2017. Genetic CNN. IEEE Int Conf on Computer Vision, p.1388-1397.
https://doi.org/10.1109/ICCV.2017.154

Zhan ZH, Li JY, Zhang J, 2022a. Evolutionary deep learning: a survey. *Neurocomputing*, 483:42-58.
https://doi.org/10.1016/j.neucom.2022.01.099

Zhan ZH, Zhang J, Lin Y, et al., 2022b. Matrix-based evolutionary computation. *IEEE Trans Emerg Top Comput Intell*, 6(2):315-328.
https://doi.org/10.1109/TETCI.2020.3047410

Zhong Z, Yan JJ, Wu W, et al., 2018. Practical block-wise neural network architecture generation. IEEE/CVF Conf on Computer Vision and Pattern Recognition, p.2423-2432.
https://doi.org/10.1109/CVPR.2018.00257

Zoph B, Le QV, 2017. Neural architecture search with reinforcement learning. https://arxiv.org/abs/1611.01578