Luo and Wang
EURASIP Journal on Advances in Signal Processing (2022) 2022:43
https://doi.org/10.1186/s13634-022-00868-1

EURASIP Journal on Advances in Signal Processing

RESEARCH Open Access

Automatically search an optimal face detector for a specific deployment environment

Jiapeng Luo and Zhongfeng Wang*

*Correspondence: zfwang@nju.edu.cn

School of Electronic Science and Engineering, Nanjing University, Nanjing, China

Abstract

Face detection plays an important role in many artificial intelligence applications, such as identity recognition, facial expression recognition, and gender/age recognition. Recently, the development of deep learning techniques has greatly improved face detection's performance. However, it is still ineffective and time-consuming to manually design hyperparameters of face detectors for different deployment environments with diverse distributions. Besides, due to the limited computation capability, many previous networks are hard to meet the latency requirements in deployment environments, and the improved resolution of current cameras further increases the computation burden. Motivated by the above problems, we propose a searching framework aiming to automatically search a real-time face detector architecture with a fixed complexity constraint, to adapt a specific deployment environment. We model the whole searching space into two parts, including the hyperparameters of the network and the detector. Instead of only searching the network structure, the proposed method considers the whole model's hyperparameters space which contains the preprocessing and postprocessing parameters. The evolutionary algorithm is employed to find the optimal solution, and new evolutionary operations are proposed to explore architecture space. During the whole searching procedure, we guarantee the computation cost is under the restrictions. The advantages of the proposed framework are that it considers a hard computation cost constraint and the preprocessing and postprocessing hyperparameters, leading to a fully automatic design style and global optimization. Finally, we evaluate the proposed model on the most popular Widerface and FDDB datasets. The proposed detector significantly surpasses the existing lightweight face detectors in the comprehensive performances, and the average latency is twice as shorter as the best competitor.

Keywords: Face detection, Lightweight network, Real-time detection, Network architecture searching, Evolutionary algorithm

1 Introduction

The task of face detection is to decide the locations and sizes of the faces in an image. It is the stepping stone to many other facial analysis techniques and intelligence systems, such as face alignment, face modeling, face recognition, facial expression recognition, and gender/age recognition [1]. From the classical Viola–Jones detector [2] to the recent ASFD [3], the accuracy has been improved dramatically with the development of

deep learning techniques. The early methods use handcrafted features and a classifier to determine whether a face appears. Designing handcrafted features and classifiers robust enough is challenging, because its accuracy can be easily hampered by occlusion, illuminations, complex background, low resolution, blur, skin color, and other diversity in an unconstrained context. With the help of the massive labeled data, CNN can automatically explore better features with no need for human experiences. In recent years, CNN-based detectors make significant progress and have dominated the face detection fields, in both academic and industrial fields.

Nowadays, state-of-the-art detectors can make very accurate predictions, whose accuracies are several times higher than classical detectors [3, 4]. However, there are still several unsettled obstacles to deploy the detection model to real deployment environments. Firstly, the various input image sources lead to quite different data distribution, such as illumination, input image sizes, and face sizes. For example, the public surveillance's face targets are usually small and blur, while the indoor home camera's face targets are usually larger and clear. A model suitable for one environment may fail in another. Therefore, it is necessary to design different detector hyperparameters to reach the best performance in different applications. It is also particularly important to utilize the limited computation power for edge devices whose computation resources are very restricted. However, manually designing all the hyperparameters is suboptimal and time-consuming, because the interactions between each of the detector's hyperparameters are considerably complicated.

Furthermore, the high computing cost is still CNN's Achilles' heel. The state-of-the-art detectors mostly take a powerful and large network as the backbone, such as VGG-16 [5] and ResNet [6, 7]. The computation of these detectors can reach a Tera of FLOPs that only computers with accelerators such as GPUs can afford. Besides, high-resolution cameras have become more common in recent years. The advanced cameras can have a resolution higher than 4096×2160 (4K). The computation cost of a network is quadratic proportional to the input image's size, as the computation of 4K resolution is approximately 30 times larger than that of 640×480 (VGA resolution). But the existing face detector usually does not consider the impact of the input image's resolution. Even deployed in the powerful server, it still can exhaust the computation resources when the number of cameras rises. There are already some researches providing efficient and lightweight face detectors. However, the speed is still unsatisfactory when the image size is large.

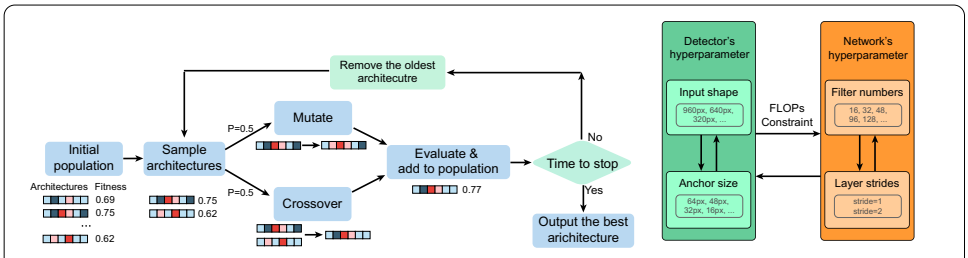
Motivated by the problems above, we propose a searching framework to find an optimal face detector with a given computation constraint. Given the input image resolution and the FLOPs constraint, the searching algorithm automatically searches the hyperparameters and makes an optimal trade-off between them. We employ an evolutionary algorithm to explore the detector architecture space instead of designing it by human experiences.

Neural architecture search (NAS) [8, 9] is a technique for automating the design of artificial neural networks. However, the existing neural architecture search methods

focus on the network space, which is obviously not optimal for the whole model. A typical CNN-based detection model is usually composed of preprocessing, network inference, and postprocessing. The parameters in preprocess and postprocess are not taken into account by existing methods. To solve this problem, we model the searching space as the network's hyperparameters and detector's hyperparameters, as shown in Fig. 1b. The proposed searching space includes not only the network's channels and strides but also the image scaling factor and anchor sizes, which have effects on the preprocessing and postprocessing. All these parameters interact with each other and are crucial to detection performance. For example, when the input images are resized to a smaller size, the network's filter can be larger to compensate for the reduction of computation. Nevertheless, the optimal point of this trade-off is difficult to be determined. In our work, the aging evolutionary algorithm is employed to find the optimal solution automatically. The evolutionary algorithm tries to find the best solution in the search space by updating a population of solutions. Figure 1a shows the overall framework of the evolutionary-based NAS. Evolutionary-based NAS has several advantages over other optimization algorithms such as reinforcement learning-based (RL-based) and differentiable NAS. First, the searching space has more freedom compared with differentiable NAS, because the evolutionary algorithm can process the discrete values. Compared with the RL-based NAS, the evolutionary-based NAS is simpler and can produce a competent result with RL-based methods and requires less time.

To make the evolutionary algorithm able to manage the detector's structure, we propose new mutation and crossover operations. As the core functions in the searching algorithm, the mutation and crossover operations generate the new individuals from parents' architectures. The proposed mutation and crossover operations process discrete parameters, such as the index of layers with convolution stride. The key problem brought by these operations is that the computation cost of the detector changed and can exceed the restriction. We solve it by uniformly rescaling each layer's channel number to fit the computation cost to the limitation.

Compared with prior works, the proposed detector is easier to be deployed to the production environment because of the simple architecture and it removes the



(a) Aging evolutionary algorithm (b) Searching Space

Fig. 1 The framework of the aging evolutionary algorithm and the proposed searching space. Each colored vector is a group of hyperparameters, which makes up a detector

nuisance of manually designing procedures. To demonstrate the performance of the proposed method, we evaluate the automatically designed detector on two popular benchmarks, Widerface and FDDB. The experiment result shows that the proposed method has a better comprehensive performance as the accuracy is on par with the best lightweight detectors and the speed is twice faster than the best competitor.

For clarity, the contributions of this paper are summarized as follows: (1) We propose a framework that employs evolutionary-based NAS to automatically design a real-time face detector adapting to specific deployment environments. (2) We model the whole searching space into two parts, which include not only the network's structure but also the whole detector's parameters, leading to a fully automatic design style. (3) The new mutation and crossover function of the network architecture are proposed to manage a detector's hyperparameters. (4) Compared with the state-of-the-art lightweight face detectors, the detector produced by our method achieves more than twice the inference speed and comparable accuracy.

The rest of this paper is organized as follows. Section 2 contains a brief review of lightweight face detectors and NAS. In Sect. 3, we detail the proposed framework. In Sect. 4, we compare our method with prior works on multiple aspects. Section 6 draws an overall conclusion of our works.

2 Related work

There are already many prior works about the NAS technique and face detection. However, most of these methods focus on either NAS or detection. A few methods that combine detection and NAS do not consider the hyperparameters outside the network. In this section, we give a brief introduction of the prior NAS and lightweight face detection works which are closely related to the proposed method.

2.1 Lightweight CNN-based face detector

As a special case of general object detection, face detection has many similarities and they can even be used interchangeably with proper modification. However, face detection has its own characteristics. For example, human faces are closer to a vertical ellipse and have more significant visual features than general objects. So, there are a lot of researchers focusing on face detection. CNN-based lightweight face detectors have been studied for many years. The prior works on it can be categorized into two branches, the cascaded detectors and the one-stage detectors. The cascaded architecture [10–12] utilizes the first stage to find many candidate boxes, which contain many false positive predictions and are refined by the following network. Because the training and inference of the cascaded architectures are complicated, one-stage detectors [13–20] are attracting more attention. One-stage detectors only use a single network to detect faces, so the number of faces has little effect on inference speeds. The prior works mainly focus on the new network structure, anchor matching strategy, loss function, etc.

General object detection methods can also be employed to detect faces. The SSDLite [21] and Pelee [22] are single-stage object detectors. The most popular detectors, such as YOLO [23–25] and SSD [26], also have lightweight versions. Nowadays, anchor-free methods also attract much attention. Instead of using anchors, FCOS [27] and CenterNet [28] use corner points and center points to locate the objects.

However, the existing methods mostly only consider the VGA resolution as the default input size, which is not optimal for high-resolution images such as 2K and 4K. In different deployment environments, the data distribution is different and the default hyperparameters of the detector are also not optimal. Besides, they share the common problem that it is difficult to have optimal hyperparameters subject to a computation cost limitation.

2.2 Neural architecture search

NAS algorithms aim to automatically learn a network topology that can achieve the best performance on a certain task. It provides a systematic and automatic way of learning high-performance model architectures. NAS samples a population of child networks and receives its performance metrics as rewards for learning to generate the desired architecture.

The first work that coined the NAS acronym is [29]. The author uses an RNN controller trained by the reinforcement learning (RL) algorithm to generate architectures sequentially. Then, the following works [30–32] improve it by changing the searching space, employing other controllers, taking a different controller trainer, and so forth. Proxy training datasets are needed because RL-based NAS is computationally expensive and difficult to be trained directly on large datasets. MNasNet [32] measures real-world inference latency and uses it as a regularization to the environment reward. It forces the final model to have a relatively fast speed.

To reduce time consumption, recent studies [33–36] use a weight sharing supernet to cover the entire search space. The supernet's weights and its differentiable architecture parameters are jointly optimized by gradient descent. Compared with RL-based NAS, differentiable NAS, which is also known as gradient-based NAS, needs only a small fraction of the searching cost. Among them, FBNet [37] uses the estimated network latency as a regularization to produce a desired fast model. But it is also a soft limitation as it does not have an actual upper bound.

Evolutionary algorithms generate a population of individuals to evolve offsprings for better performance. Offsprings of individuals are generated by mutation or crossover. Compared with RL-based and gradient-based NAS, the evolutionary algorithm is simpler and can manage more complicated hyperparameters. It does not need the hyperparameters in searching space to be differentiable, which is required by gradient-based NAS. Real et al. [38] shows that the evolutionary algorithm can produce a competent result with RL-based methods and requires less time. Kyriakides and Margaritis [39] combine the regularized evolution and

genome representation to conduct a global search. Miikkulainen et al. [40] utilize genetic algorithms and co-evolution of species to improve the cells in the network. However, like [38], the existing evolutionary algorithm focusing on image classification task is not suitable for searching lightweight face detectors for two reasons. Firstly, the detector's hyperparameters are not taken into account during the searching process. Secondly, the model's complexity is not directly constrained by these algorithms.

Some prior works also combine the NAS technique with the object detection task, such as the EfficientDet [41] and NAS-FCOS [42]. They achieve well detection performances but only focus on the structure of the network while ignoring the hyperparameters outside the network. Auto-FPN [43] and NAS-FPN [44] focus on the feature pyramid structure which also only uses the default input image resolution.

3 Methods

In this section, we give the problem definition and introduce the proposed evolutionary-based NAS and face detector structure.

3.1 Problem definition

We try to find the best combination of the hyperparameters of a face detector under an upper bound of the FLOPs. It can be formulated as a global optimization problem:

$$\begin{aligned} &\arg\max_{\boldsymbol{\beta}} \quad performance(\boldsymbol{\beta}) \\ &subject\ to\ flops(\boldsymbol{\beta}) <= \theta \\ &\qquad\quad \boldsymbol{\beta} \in \boldsymbol{D} \end{aligned} \quad , \tag{1}$$

where $\boldsymbol{\beta}$ is a vector of the selected hyperparameters, $\boldsymbol{D}$ is the definition set of the $\boldsymbol{\beta}$. $flops(\boldsymbol{\beta})$ is the detector's FLOPs according to $\boldsymbol{\beta}$. $performance(\boldsymbol{\beta})$ is the detection metric in a specific dataset as the optimization target. We use the mean average precision (mAP) on the validation set as the optimization target in experiments.

3.2 Evolutionary algorithm

The evolutionary algorithm (EA) is an optimization technique that utilizes a metaphor borrowed from natural and genetic selection mechanics. It is a heuristic-based approach suitable for solving problems that cannot be easily solved in polynomial time, such as classic NP-Hard problems. As we described in Sect. 1, evolutionary algorithms have several advantages over other NAS techniques. We select the aging evolutionary algorithm in our method because it does not need hyperparameters to be differentiable, which gives us more choice to design our searching space. The aging evolutionary algorithm discards the oldest individual instead of the weakest one to avoid the suboptimal points.

An evolutionary algorithm can be concluded as 5 steps as follows:

- Initialization: Randomly generate the initial population of individuals.
- Repeat:

- Evaluate the fitness of individuals in the population.
- Select individuals in the population for reproduction.
- Generate new individuals through crossover and mutation operation using selected ones.
- Add new individuals to the population and delete old ones.

Figure 1a presents the overall framework of the evolutionary algorithm. Each colored vector is an individual during evolution and represents a group of hyperparameters, which makes up a detector. The old individual is eliminated and offsprings are generated during each iteration of the evolution. The employed evolutionary algorithm' details are presented in Algorithm 1. The employed elimination strategy is similar to [38], namely the aging evolutionary algorithm. In our optimization problem, each individual is a detector and is represented by a chromosome that consists of several genes to be searched. The genes in a chromosome are the variables to be searched here. During the evolution process, the better detectors have a bigger chance to be the parents to produce the offspring and the weaker detectors have a bigger probability to be eliminated. It is analogous to natural selection and the detectors that survive in the population can be evolution step by step.

In Algorithm 1, we construct a population containing the P individuals firstly. Each individual is a potential optimal solution, which represents a group of hyperparameters in the context of NAS. Words "individual" and "architecture" are used interchangeably in the rest of this paper. During the initialization of the population, we add several seeded individuals initialized manually by human experiences to reduce the convergence time. The others are initialized with random architectures.

After the population is initialized, the aging algorithm begins to evolve the population's architectures at each cycle. At each cycle, S individuals are uniformly sampled from the population as the candidate parents. One or two best candidates are selected to be the *parents* (noted as $p1$, $p2$ in Algorithm 1) of the new individuals. Then, the new individual is produced randomly from the parents by mutation or crossover operation. The mutation operation creates child architecture from one parent by randomly modifying the attributes. The crossover operation produces a child by randomly selecting attributes from two parents. Algorithm 2 and Sect. 3.5 express mutation and crossover in more detail. After constructing a child architecture, to keep the population size unchanged, we need to discard an individual from the population and push the new individual into it. Instead of discarding the worst individual in the population, the proposed method discards the oldest individual to avoid the suboptimal points. Then, the algorithm pushes the constructed child into the end of the population. As time passes, even the best individual will die, which conforms with natural principles. The elites still have the advantage that they are more likely to be selected as parents.

Figure 2a shows the time-course of the model accuracy during the evolution process where the orange line is the moving average of accuracy. The accuracy approximately converges within the first 100 cycles. In the following cycles, the population's accuracy remains growing at a slower speed.

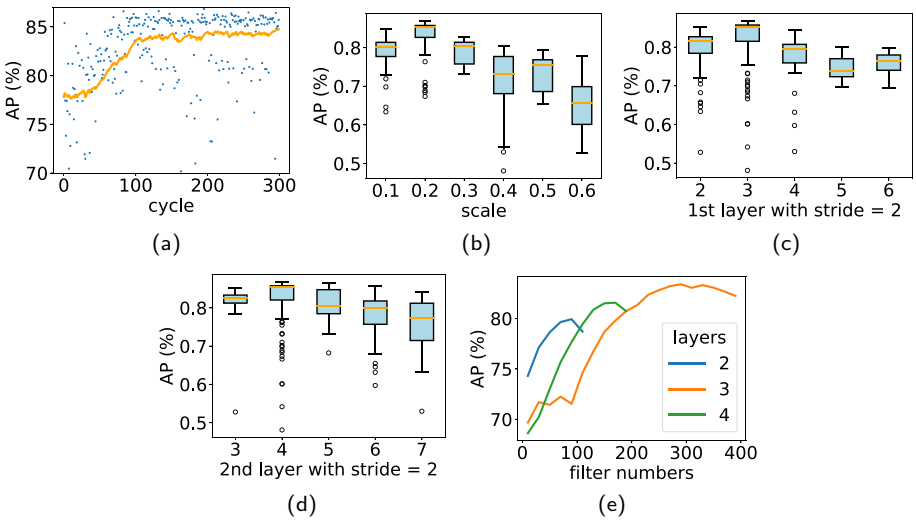


Fig. 2 Evolution process and effects of parameters on model accuracy

Algorithm 1: Aging Evolution

Input:
$population \leftarrow$ {seeded architectures};
$P \leftarrow$ population size;
$C \leftarrow$ total cycle number;
$S \leftarrow$ candidates number;
$F \leftarrow$ upper bound of FLOPs;

Procedure:
while $|\ population\ | < P$ **do**
  $model.arch \leftarrow$ **RandomInit**$(model.arch)$;
  $model.accuracy \leftarrow$ **Evaluate**$(model.arch)$;
  push $model$ into $population$;
end
while $|\ history\ | < C$ **do**
  sample S candidates architectures from $population$;
  if ***Uniform***$(0,\ 1) > 0.5$ **then**
    $parent \leftarrow$ highest accuracy $model$ in candidates;
    $child.arch \leftarrow$ **Mutate**$(parent.arch,\ F)$;
  else
    $p1, p2 \leftarrow$ two highest accuracy $models$ in candidates;
    $child.arch \leftarrow$ **Crossover**$(p1.arch, p2.arch, F)$;
  end
  $child.accuracy \leftarrow$ **Evaluate**$(child.arch)$;
  Push$(child,\ population)$;
  Push$(child,\ history)$;
  pop the oldest architecture from $population$;
end
Result: $model$ with highest accuracy in history

3.3 Detector structure

In this section, we describe the deep face detectors included in the search space. We custom design a simple and efficient detection framework that consists of preprocess, inference, and postprocess stages. The detection framework includes the variables which will be searched during the evolution process. During the searching process, all candidate

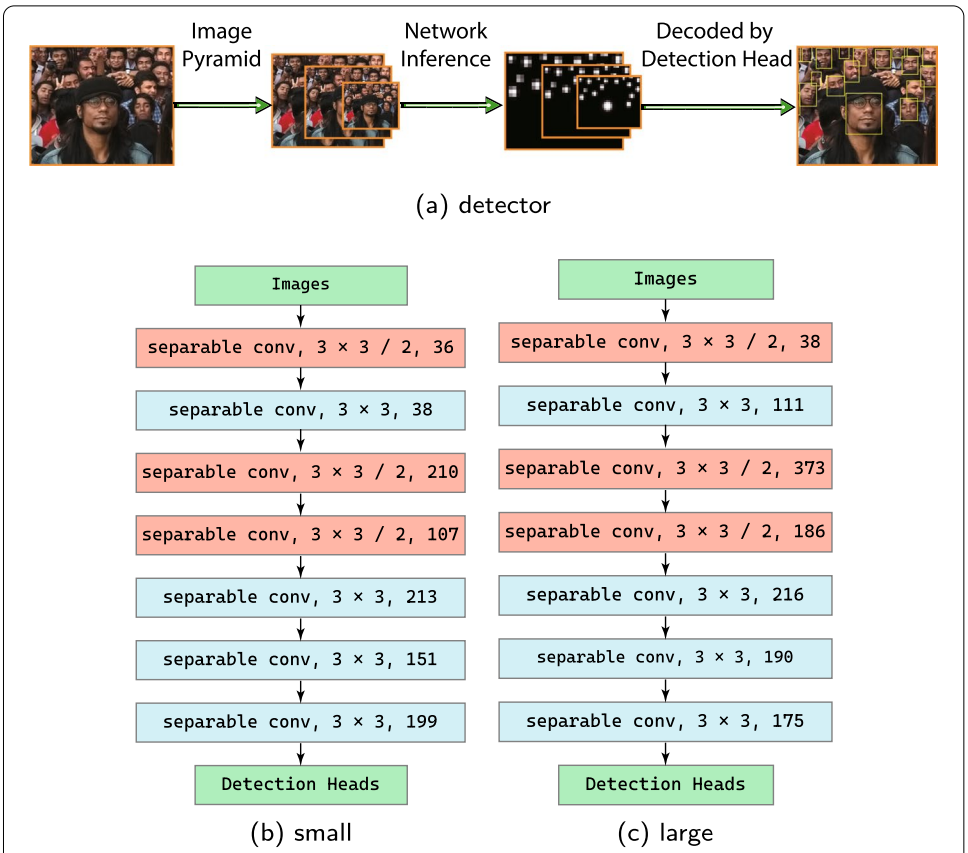


Fig. 3 Face detector Inference structure. **a** The detector is composed of three parts, preprocess, network, and postprocess. **b** and **c** are the searched network structure. "/2" means the convolution's stride is 2

detectors share the same framework. All these variables compose the final searching space described in the next section.

A straightforward CNN composed of separable depthwise convolution is employed as the backbone, as shown in Fig. 3a. Each block of the network consists of a separable depthwise convolution layer, a batch norm layer, and a ReLU layer. A separable convolution layer consists of a depthwise convolution and a pointwise convolution. It has been proven to be very efficient for lightweight networks [21, 45]. Therefore, the proposed network is very straightforward and is easy for training and deployment. To detect objects of different sizes, we employ the image pyramid strategy and use two square anchors. Each anchor has an independent detection head network for outputting the classification and regression features.

As shown in Fig. 3a, during the inference stage, the input image is resized to a smaller size with a scaling factor determined by EA. Through the image pyramid, the image is further resized to several images with different scales. The network runs on each resized image. Then, the postprocess can convert the network's output to bounding boxes of faces using the anchor-based strategy. The conversion is similar to Faster-RCNN [46]. A square grid is defined and it has the same height and width as the network output. The grid's stride is also the same as the network's stride, so the points of the grid can match the network output's receptive fields. For example, the dimension of the network

input is 640 × 640 × 3, as height, width, and RGB channels. Then, the dimension of the network's output is 80 × 80 × 6. The first two output channels are the classification logits, while the rest channels correspond to the transformed center offsets and size of the predicted bounding boxes. We present a square anchor box in each grid point and use the network's output to classify it and to adjust its boundary. Equation 2 is the predicted boxes where x, y, h and w are the coordinates and shape of the predicted boxes. x_a, y_a, h_a and w_a denote the preset anchor box's coordinates and shapes. x_t, y_t, h_t and w_t are the outputs of the network. s is the image scaling factor to resize the boxes to the original image resolution.

$$x = \frac{w_a x_t + x_a}{s},\ y = \frac{h_a y_t + y_a}{s},\ w = \frac{w_a e^{w_t}}{s},\ h = \frac{h_a e^{h_t}}{s} \tag{2}$$

Following the standard object detection, the training of the detector is actually the classification and regression of each preset anchor box, so the training loss consists of classification loss and regression loss. An anchor box is positive if it has an IoU higher than 0.35 with ground-truth face and vice versa. We use the cross-entropy loss as the classification loss. The regression loss is the Euclidean loss indicating the size and position distance between anchors and ground-truth faces. The target values of the regression are actually the x_t, y_t, h_t and w_t in Eq. 2 when we set x, y, h and w as the ground-truth face. The final regression loss is the sum of the positive anchor boxes' regression loss.

It can be noted that the employed detection framework can be propagated to the general object detection task. Although the proposed framework targets face detection and many parameters are optimized for it, there is no intrinsic contradiction to general objection. With proper modifications, the proposed framework can also support other detection tasks.

3.4 Searching space

The searching space of the evolutionary algorithm is the set of all the possible network architectures. The proposed searching space includes not only the network parameter space but also the complete detector hyperparameter space, covering preprocessing and postprocessing. The complete searching space is quite large and is difficult to converge, and therefore, we make several simplifications, as shown in Fig. 1b. The whole space consists of two parts, detectors' hyperparameters and network's hyperparameters, which include *scale*, *anchors*, *channels*, and *strides*. It is very easy to add more hyperparameters or new type hyperparameters if needed. The total layer number and total stride of the network are specified in advance.

The *scale* is the input image scaling factor. Directly executing the network on the original resolution image is unnecessary. Because we can train the detector to detect faces smaller than the ground-truth faces, properly resizing the image to a smaller size would not damage the accuracy of the detector. The FLOPs of a network decrease quadratically with the input image size, and each layer's FLOPs can be calculated by Eq. 3. The computation saved on a smaller resolution can be invested in the network. For example, when an image is resized to its half size, the network's FLOPs can be approximately four times larger, which keeps the total FLOPs unchanged. The *scale* has a strong relation to the *anchor*.

The *anchors* are the preset square box sizes, which directly affect the scales of detectable faces. The sensitivity of the network in different size faces is very different. Besides, the anchor sizes need to adapt to the scaling factor which is crucial to accuracy. When the image is zoomed to a smaller size, it is crucial to choose a suitable anchor size to match the faces.

The *channels* are the output channels of each convolutional layer in CNN. It is equal to the convolutional kernel number and also equal to the next layer's convolutional kernel depth, which is proportional to both the computation amount and the information amount of a feature map. Equation 3 is the FLOPs of a single conventional convolutional layer regardless of padding, where H and W are the input feature map's shape, K is the kernel size and S is the stride. As we focus on the lightweight network, the performance is sensitive to each layer's channel numbers.

$$FLOPs = \frac{H \times W \times K^2}{S^2} \tag{3}$$

The *strides* are the indexes of the layers whose stride is 2. The stride of a convolutional layer is the step of the convolution operation. When stride is 2, a convolution operation reduces the feature map size like a downsampling operation. Hence, a layer with a stride of 2 can squeeze the feature map information and reduce the latter layers' computation. The existing methods usually set each layer's stride by experience. If the shallow layer's stride is 2, the deep layer can have more convolutional filters when the total FLOPs is fixed. The stride of each layer can be 1 or 2, except that the strides of three of the layers need to be 2.

During experiments, we produce two detector models by using two different FLOPs constraints. We use "Our-small" and "Our-large" to indicate two detectors. The searched parameters of these two models are presented in Table 1. The detailed performances of the two detectors are expressed in Sect. 5.1.

3.5 Mutation and crossover

The mutation and crossover operations generate new individuals during the evolution process, which is the core operation during evolution. They provide a way to explore the searching space.

The term "mutation" and "crossover" originated from biology. The mutation and crossover happen during genetic recombination which causes gene diversity.

Table 1 The settings and searched parameters of the proposed detectors

Model name	Our-small	Our-large
FLOPs(VGA)	100M	200M
FLOPs(4K)	3.99 G	8.20 G
Filter numbers	36, 38, 210, 107, 213, 151, 199	38, 111, 373, 186, 216, 190, 175
Layers with stride 2	1, 3, 4	1, 3, 4
Scaling factor	0.237	0.216
Anchor sizes	16, 28	16, 28

Mutation operator involves a probability that an arbitrary bit in a genetic sequence will be flipped. The crossover operator produces new genes by exchanging the bits from the original ones. For our searching problem, an individual is a vector of variables, a mutation operation randomly changes some variables and the crossover operation combines two individuals to be one individual.

The original mutation and crossover can only process the data encoded which is usually a vector of 0s and 1s. We reform them to make it able to manage the detector's hyperparameters. Algorithm 2 is the mutation operation employed in the proposed method. The values of each individual are the genes, and the mutation's probability for each gene is 0.3. The new *scale* follows uniform distribution ranging from 0.1 to 0.7. The new *strides* are sampled randomly without replacement from 2 to 7, which indicates the layer index. Layer 1 is excluded because the first layer is always fixed as stride 2. Each channel number is randomly selected independently from an uniform distribution ranging from 0.5 to 1.5 times the original number. The new *anchors* are sampled from 8 to 32. After the mutation, the FLOPs of the detector may exceed the upper bound. To guarantee the FLOPs restriction, we greedily scale the channels after mutation and crossover, as shown in line 13 of Algorithm 2.

Different from the mutation, the crossover operation combines two parents' individuals to generate a new offspring. We use the uniform crossover for all parameters, as *Channels*, *strides*, *scale*, and *anchors* are uniformly selected from two parents. The channel number is also scaled like that in mutation to guarantee the restriction of FLOPs. Because the offspring can inherit the good genes from its parents, which are probable elites in the searching space. Compared with mutation, the offspring is probably a better individual by limiting probability space in its parents.

Algorithm 2: Mutation Operation

Input:
$arch = \{ scale, strides, channels\}$;
$F =$ FLOPs upper bound ;
1 **Procedure:**
2 with probability 0.3:
3 $anchor.scale =$ **Uniform**(0.25, 0.7);
4 with probability 0.3:
5 $arch.strides =$ {select two numbers from 2 to 7};
6 **for** $i = 1$ *to* $length(arch.channels)$ **do**
7 with probability 0.3:
8 $cmin = channels[i] \times 0.5$;
9 $cmax = channels[i] \times 1.5$;
10 $channels$[i] = **Uniform**(cmin, cmax);
11 **end**
12 **while** $Flops(arch) > F$ **do**
13 $arch.channels = arch.channels \times 0.9$;
14 **end**
Result: $arch$

4 Experiments

In this section, we describe our experiment settings. The experiment results are presented in the next section.

4.1 Datasets

The face detection dataset we used includes Widerface [47] and FDDB [48]. Widerface is one of the most widely used face detection datasets. It contains 32,203 images and 393,703 faces where 40%/10%/50% images are randomly selected as training, validation, and testing sets, respectively. During the evaluation and test stage, Widerface dataset has three difficulty level sets, including easy, medium, and hard sets. The widerface easy set only contains faces larger than 50 pixels. Therefore, it is more practical than the hard set because smaller faces are difficult for following analysis algorithms and are filtered by the quality control process in most applications. So we set the AP on Widerface easy set as the optimization goal of the searching algorithm, and pay more attention to the easy set, medium set during evaluation. FDDB [48] is another widely used face detection dataset. FDDB contains 5171 annotated faces in 2845 images and 5171 faces that vary largely in appearance, pose and scale. All the architectures are trained on Widerface train set and are evaluated on Widerface test set and FDDB dataset. Figure 4 is the distribution of the face height distribution of Widerface and FDDB. The distributions of these datasets are very different, as the most frequent face is around 10 pixels in Widerface hard set and 60 pixels in FDDB.

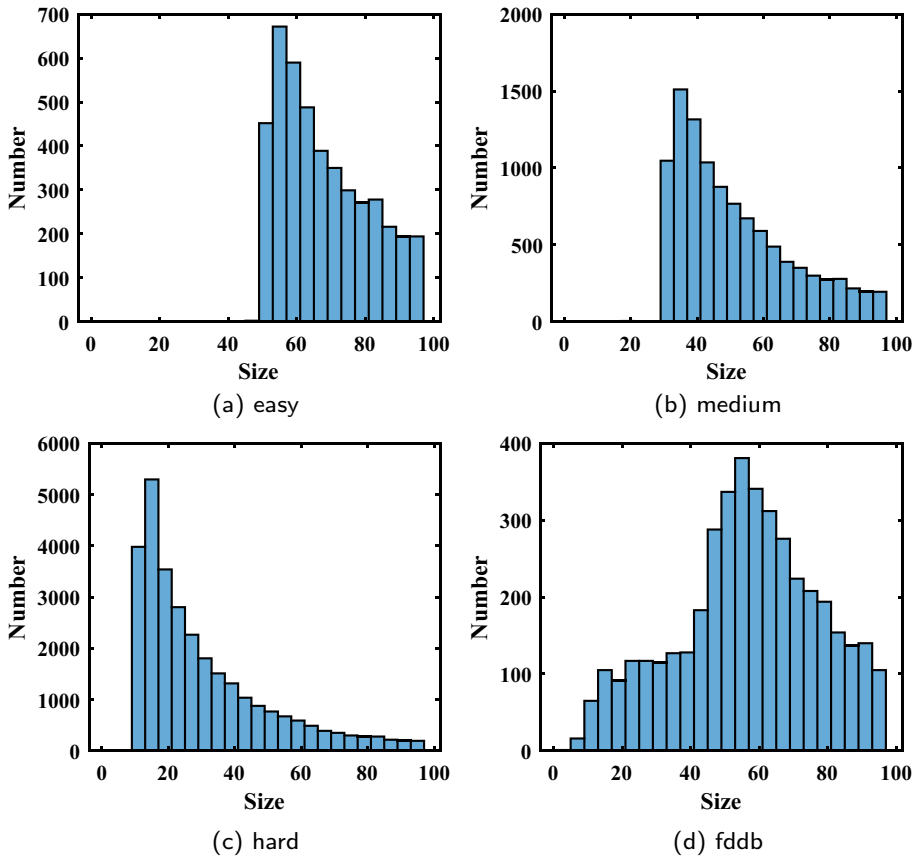


Fig. 4 The distribution of the face sizes in Widerface val set and FDDB

4.2 Experiment settings

To show the difference of the searched detector with different complexity constraints, we use the proposed searching algorithm to produce two models by fixing the network FLOPs to 100M and 200M with VGA resolution input. We use "Our-small" and "Our-large" to denote these two models. The FLOPs of the two models are fixed, and the model size is determined automatically by a searching algorithm under its constraint.

Widerface training set is employed as the training dataset. As the hard set contains many faces as small as 10 pixels size, we focus more on easy, medium, and FDDB sets. The medium set's AP is our optimization target. Image augmentation during the training procedure includes random cropping, random scaling, and color distortion. The image size during training is 256×256. The image pyramid is not employed during training. All the parameters are initialized with the random "Xavier" method. The learning rate is 0.01 for Adam optimizer. Each architecture is trained within 3 epochs, which costs a quarter of an hour in all on a 4 GPUs workstation. As we set the cycles of the evolutionary algorithm to 300, it costs around 3 days with only 4 GPUs, which is much faster than most of the existing methods. After the evolution process ends, the best individual is trained again with a better training procedure. During final training, the total epoch is 15, and we employ a learning rate decay of 0.1 in every 4 epochs.

To verify the model's actual speed in the deployment environment, we test the inference efficiency using the deployment toolchain on two production platforms, including NVIDIA RTX2080Ti and NVIDIA Jetson Xavier. We export the model to ONNX format and use the NVIDIA TensorRT framework to accelerate the network inference. Latency and average latency are measured. Latency is the time of processing a single image when batch size is 1, while the average latency is the time for a single image with the best batch size. The results are shown in Sect. 5.3.

To verify the effectiveness of the proposed searching scheme, we compared the searching scheme with a random searching scheme and handcrafted designed detectors. The random searching scheme shares the same searching space with the proposed method but uses uniform distribution to randomly generate parameters. 3 handcrafted designed detectors are fully trained and the best one is chosen to be compared with the proposed searching method. The results are shown in Sect. 5.4.

5 Result and discussion

In this section, we analyze the experiment results. For clarity, we summarize them as follows: (1) We present the detection performance and the searching time of the proposed model in Sects. 5.1 and 5.2. (2) We compare the speed and complexity of the proposed model against other methods in Sect. 5.3. (3) We compare the proposed searching scheme with the random searching baseline scheme in Sect. 5.4. (4) We analyze the evolution process and present the details about the evolutionary trends of parameters in Sect. 5.5. (5) We demonstrate the detection examples in Sect. 5.6.

5.1 Evaluation of searched detectors

In this section, we evaluate the proposed detectors' accuracy and inference latency. The search result is shown in Table 1, and the detail network architecture is shown in Fig. 3.

Table 2 is the comparison between the proposed detector and the existing lightweight face detectors. The size column presents the models' parameters number. The FLOPs column is the FLOPs during inference when input resolution is 4096 × 2160. The proposed models have the smallest FLOPs among all the lightweight detectors. The easy, medium, hard, and FDDB column is the average precision (AP) in Widerface test sets and FDDB dataset. The precision-recall curves are shown in Fig. 5. The TinaFace has the best detection accuracy, but the model size is very large. The speed is also far from real-time speed. Retinaface-0.25's detection accuracy is slightly better than the proposed

Table 2 The comparison with other detectors in model size, computation amount, and performances (AP%)

	Size	FLOPs	Easy	Medium	Hard	FDDB
TinaFace [49]	37.48 M	5150 G	97.0	96.3	93.4	98.3
EXTD-32 [19]	0.06 M	129 G	88.3	87.3	79.9	96.6
ASFD-D0 [3]	0.62 M	21.9 G	90.1	87.5	74.4	–
Retina-0.25 [4]	0.43 M	22.0 G	90.0	87.5	73.1	97.4
Yolov5s [25]	7.26 M	183 G	87.3	87.9	83.1	93.5
LFFD [20]	1.52 M	201 G	86.0	84.8	74.9	96.6
Our-small	0.10 M	3.99 G	88.0	86.0	64.3	96.3
Our-large	0.49 M	8.20 G	89.8	87.3	63.0	96.9

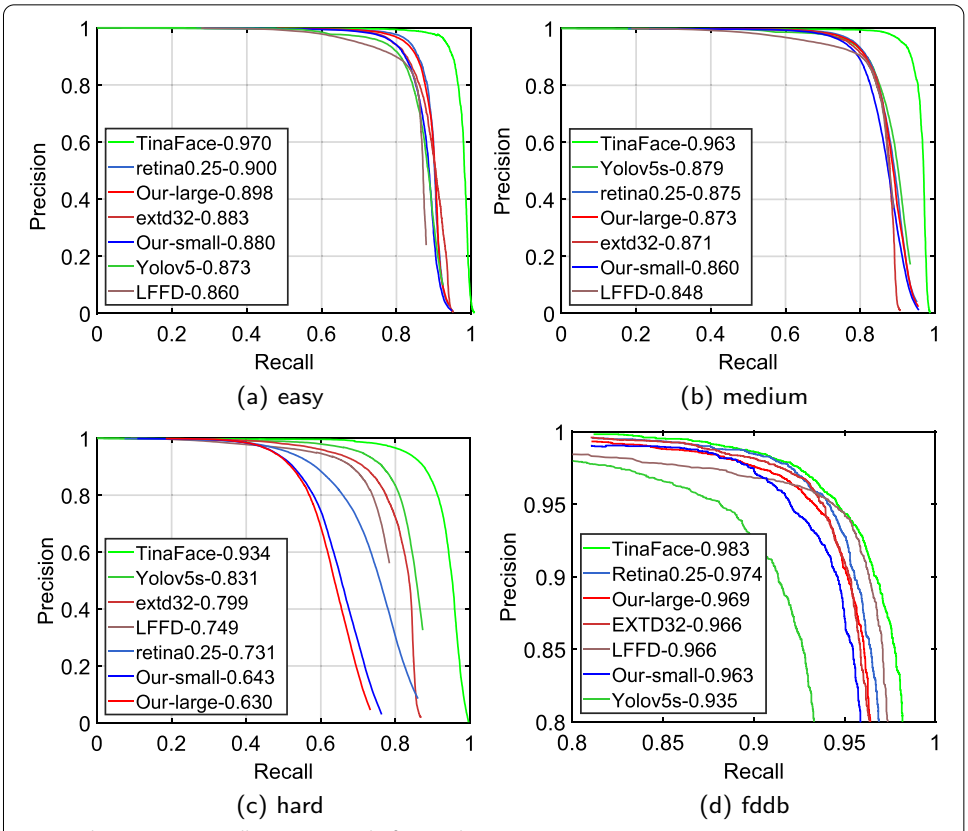


Fig. 5 The precision–recall curves on widerface and FDDB

method, but there is a large gap between FLOPs. Our large model only needs one-third of the FLOPs of the Retinaface-0.25, and the AP is just less than Retinaface-0.25 0.002 on Widerface easy and medium sets. Besides, the proposed model's speed is 2–3 times faster than Retinaface-0.25. Although EXTD has the smallest model size, it reuses the parameters too much and uses a very complicated network structure, making it very time-consuming.

5.2 Searching time

As a non-convex optimization problem, it is difficult to find the optimal point during searching. So, we need to trade off the detection performance and the searching time. The searching time is related to the single architecture's training time and the total evolution cycles. We analyze the relationship between the single detector's training epochs and its performance to choose a proper training epoch. The relationship between the evolution cycle and the searched result is also analyzed. Figure 6a is the detection performance of ten random sampled models from the searching process. Figure 6b is the average margin to the final best performance of these models. The X-axis is the training epoch. It is clear that in the third epoch, the model almost converges. And the third epoch's performance also reflects the final performance rank. So, we only train 3 epochs for each model during searching. Figure 2a shows the time-course of the model accuracy during the evolution process. After 100 epochs, the evolution process starts to converge and the best model appears around 200 epochs. Based upon these observations, we terminate the evolution process in the 300th epoch. The current best model is good enough and it needs much more time to find a better model. It takes one hour to train each model in a single GPU, so the total GPU time is around 300 GPU hours, which is acceptable. With 4 GPUs, it only takes us 3 days to finish the whole process. The time can be shortened with more GPUs. If we can afford more time and allow more training epochs and evolution cycles, the performance of the final searched model can be better.

5.3 Speed and complexity comparison

We test the inference efficiency on two production platforms, including NVIDIA RTX2080Ti and NVIDIA Jetson Xavier. Table 3 is the inference speed comparison on RTX2080Ti and Jetson Xavier. The input image resolution is 4096×2160. During the latency test, all the models are exported in ONNX format and accelerated by the TensorRT framework. The latency column indicates the time consuming when batch size is 1, while the average latency is the average latency time on each image when selecting the best batch size. The image pyramid inference time is included during latency and average latency tests. The postprocessing time is approximate 1 ms for the proposed detector and the data transferring time is all the same in all methods, so we exclude the postprocessing and data transferring time for convenience, which does not affect the conclusion. The data transferring can also be pipelined for the inference module with a very small overhead. It is clear that the proposed detector has the fastest inference speed among all these detectors. The TinaFace employed the ResNet-50 as backbone can only reach 6 fps even on RTX 2080Ti. The proposed model can achieve 2–3 times faster speed than Retinaface-0.25, which is the fastest competitor.

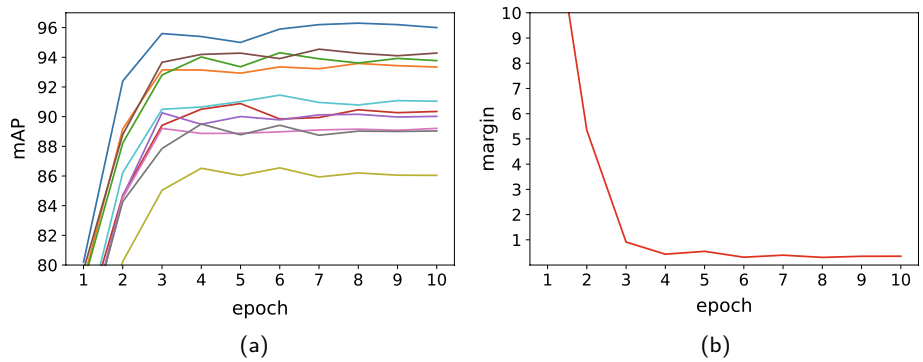


Fig. 6 The relationship between training epoch and detection performance. **a** Ten random sampled models from the searching process. **b** The average margin to the final best performance of these models in each epoch

Table 3 Inference latency comparison with 4K resolution image on GPU and Jetson Xavier

	Latency (ms)		Avg. latency (ms)	
	Xavier	**2080Ti**	**Xavier**	**2080Ti**
TinaFace [49]	1363.40	155.62	1362.15	153.45
EXTD-32 [19]	417.24	51.50	417.24	51.50
LFFD [20]	104.41	14.57	98.63	11.84
Yolov5s [25]	182.80	18.84	182.58	18.27
Retina0.25 [4]	52.54	6.85	52.21	5.34
Our-small	26.81	2.93	23.91	1.70
Our-large	34.36	3.61	31.12	2.26

Figure 7 demonstrates the comparison of the network complexity and the model runtime complexity. The Y-axis is set to be log scale. Because the difference between network complexity and runtime complexity is mainly caused by image pyramids, other detectors' runtime complexity is equal to the network complexity because of no image pyramid. The detectors produced by our searching algorithm have different runtime FLOPs and network FLOPs. In deployment scenarios, the complexity is associated with the input resolution. We select 3 most common resolutions which are 640×480 (VGA), 1920×1080 (2K) and 3840×2160 (4K). Although the runtime FLOPs are larger than

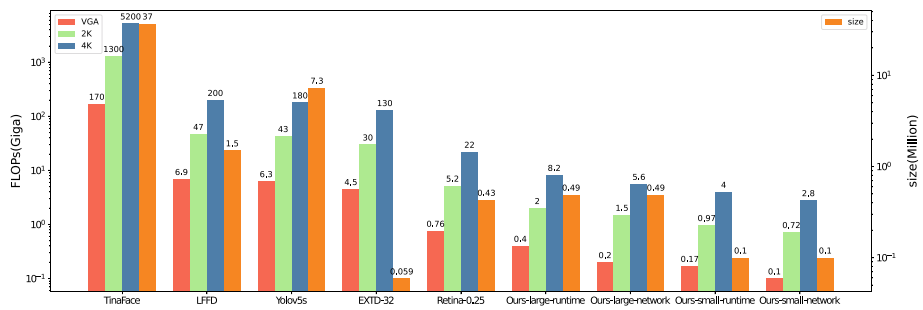


Fig. 7 The comparison of runtime complexity and the network complexity between detectors. Except for the proposed model, the other detectors have the same complexity for runtime and network because of no image pyramid

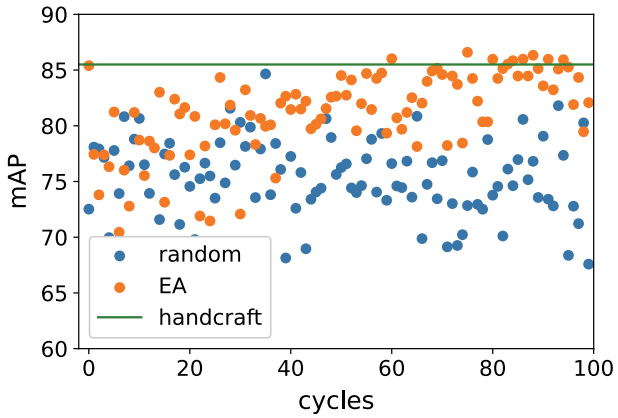


Fig. 8 The comparison between the proposed searching scheme, the random searching, and the handcrafted detector. The orange points are the detectors searched by the proposed algorithm in every evolution cycle. The green line is the best handcrafted model's performance. Blue points are random models

the network FLOPs, the proposed methods still have the smallest FLOPs among all the detectors (Fig. 7).

5.4 Comparison with random search and handcrafted detectors

To verify the effectiveness of the proposed searching scheme, we compare the proposed searching scheme with the random searching scheme and handcrafted detectors and they share the same detection framework. 100 randomly generated detectors and 5 manually designed detectors are trained. The results are shown in Fig. 8. The orange points are the detectors searched by the proposed algorithm in every evolution cycle. The green line is the best handcrafted model's performance. Blue points are random models. The randomly generated detectors in the early stage have comparable performance with the proposed method. In the latter stage, the proposed scheme is obviously better by gradual evolution. The handcrafted model can achieve good performance but still has a significant margin with the best model generated by the searching algorithm.

5.5 Evolution process analysis

We analyze the evolution process of the proposed "Our-small" detector. Figure 2a shows the time-course of the model accuracy during the evolution process. The orange line is the moving average of accuracy. The moving average accuracy increases very fast within the first 100 cycles. In the following cycles, it gradually converges over time.

Figure 2b shows the effects of the image scaling factor on the model accuracy. The orange bar in each box is the average accuracy. Obviously, the *scale* has a very significant effect on detection performance. The detection accuracy becomes poor when the face is scaled too small, although the model can have a larger convolutional kernel. On the other hand, the detector can receive a higher resolution image with a larger scaling factor, while the model has to be squeezed to a smaller size, and the accuracy decreases.

Figure 2c, d is the boxplots of detection accuracy and network strides. When a layer's stride is 2, the feature map's resolution is smaller and the spatial size is reduced by

half size, which is similar to the dimension reduction. The latter layers have a smaller input size and can have a larger convolutional filter number to compensate for the reduction of computation cost.

Figure 2e shows the relation between the convolutional filter numbers and the model AP. The y-axis is the moving average AP. To clarify, we select the second, third, and fourth convolutional layers for representation. Although the joint distribution of filter numbers is very complicated, it is still very significant that the AP of each layer has a significant optimal point. The shallow layer's best filter number is obviously smaller than the deep layers.

5.6 Visualization of the detection results

At last, we give the visualized detection results of several methods for comparison in Fig. 9. Limited by the length of the article, we select three samples that include typical face detection deployment scenarios. Because the detection result is affected by the

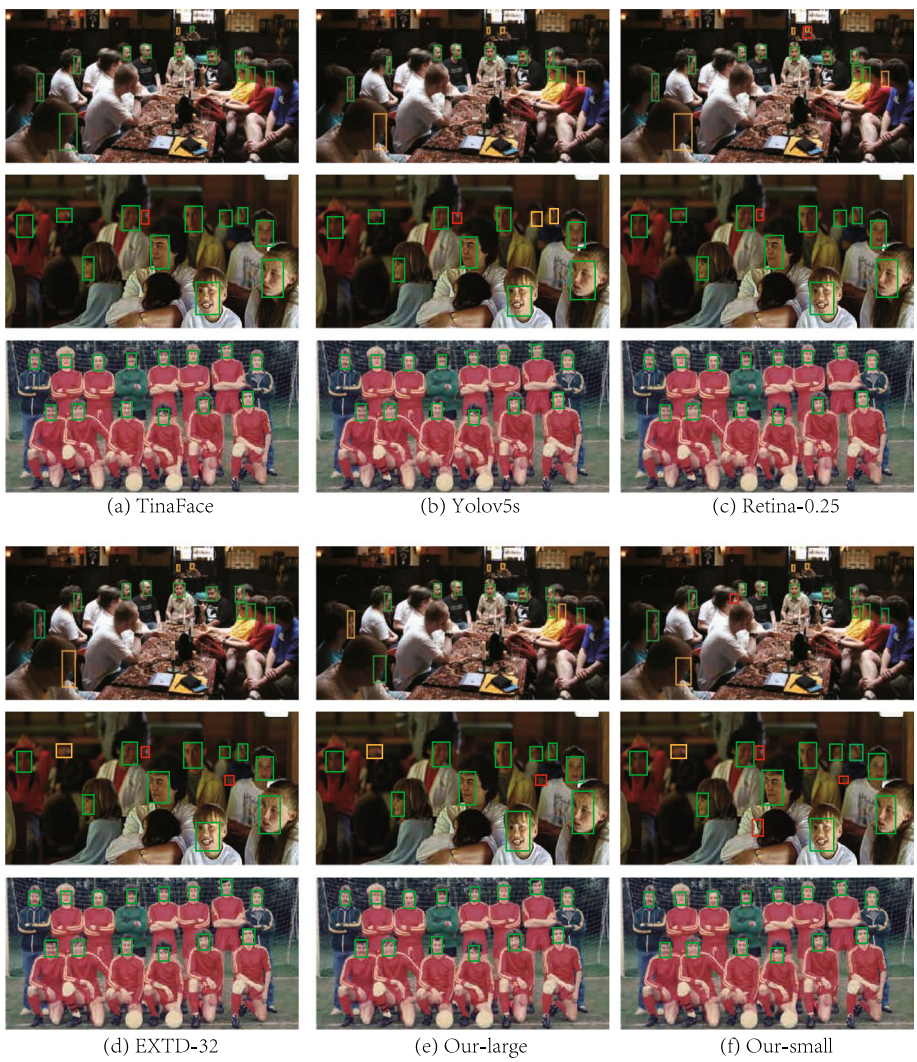


Fig. 9 Visualization of the detection results of six methods. The green, red, and orange boxes are the correct, incorrect, and missed detections, respectively

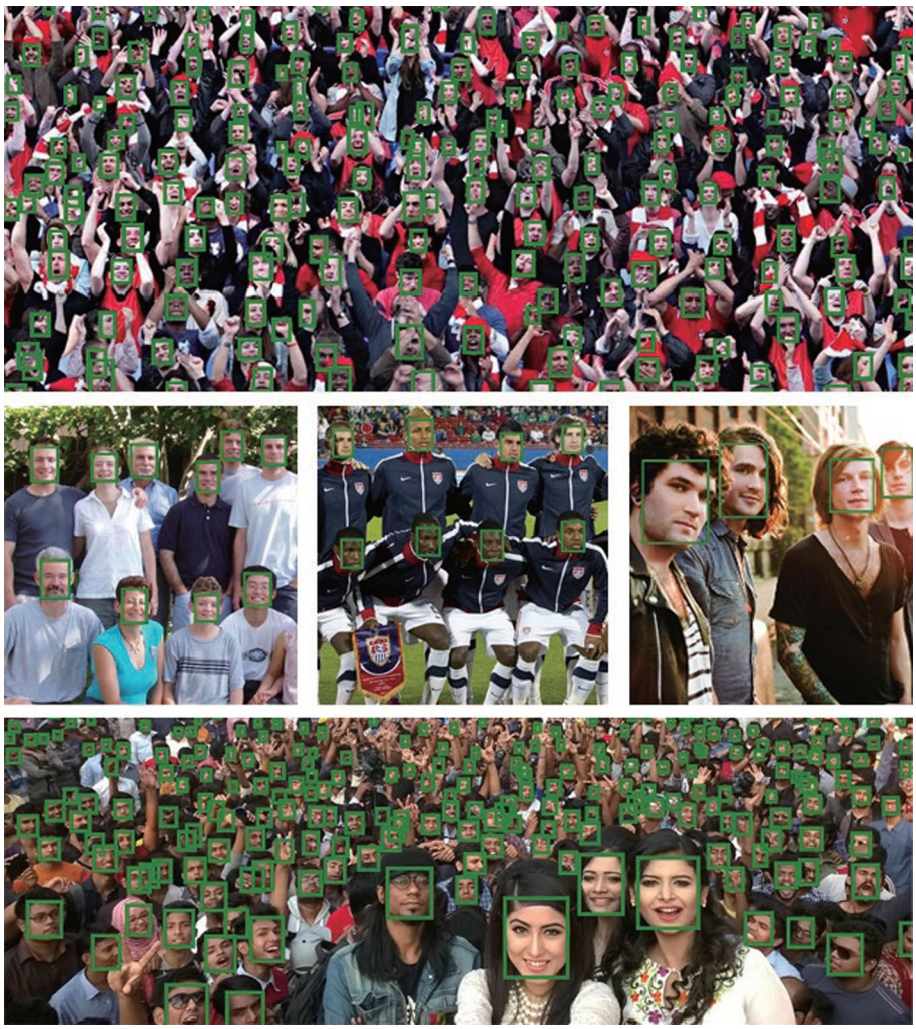


Fig. 10 Demonstrations of the prediction results of the proposed method

confidence threshold, we set the threshold to keep each model's precision to 80% for a fair comparison. The green, red, and orange boxes are the correct, incorrect, and missed detections, respectively. The TinaFace shows the most accurate detection performance by dozens of times complexity as other competitors. The other lightweight face detectors and the proposed method show comparable detection performance, but the proposed model's computation complexity is much smaller than the other methods. More detection examples of the proposed method are presented in Fig. 10.

6 Conclusion

We propose a framework to automatically search the optimal lightweight face detector by an optimized evolutionary algorithm. It aims to automatically find optimal hyperparameter settings for specific deployment environments. We model the whole searching space into two parts, not only including the network's hyperparameters but also the detector's hyperparameters. The produced face detector can achieve real-time face detection on 4K resolution images and surpass existing detectors' performance. Other

model compression methods, such as pruning, quantization, and decomposition, can still be applied to the proposed model, which can further improve the model's performance. Although we only apply the proposed framework to the face detection task, it can be easily extended to other applications.

Abbreviations
CNN: Convolutional neural network; RL: Reinforcement learning; mAP: Mean average precision; EA: Evolutionary algorithm.

Authors' Information
Jiapeng Luo received the M.S. degree in computer science and technology from Nanjing Normal University, Nanjing, China. He is currently a Ph.D. candidate in electronic and communications engineering of Nanjing University. His current research interests include artificial intelligence, deep learning and computer vision.
Zhongfeng Wang received both the B.E. and M.S. degrees in the Dept. of Automation at Tsinghua University, Beijing, China, in 1988 and 1990, respectively. He obtained the Ph.D. degree from the University of Minnesota, Minneapolis, in 2000. He has been working for Nanjing University, China, as a Distinguished Professor since 2016. Previously he worked for Broadcom Corporation, California, from 2007 to 2016 as a leading VLSI architect. Before that, he worked for Oregon State University and National Semiconductor Corporation. Dr. Wang is a world-recognized expert on Low-Power High-Speed VLSI Design for Signal Processing Systems. He has published over 200 technical papers with multiple best paper awards received from the IEEE technical societies, among which is the VLSI Transactions Best Paper Award of 2007. He has edited one book VLSI and held more than 20 U.S. and China patents. In the current record, he has had many papers ranking among top 25 most (annually) downloaded manuscripts in IEEE Trans. on VLSI Systems. In the past, he has served as Associate Editor for IEEE Trans. on TCAS-I, T-CAS-II, and T-VLSI for many terms. He has also served as TPC member and various chairs for tens of international conferences. Moreover, he has contributed significantly to the industrial standards. So far, his technical proposals have been adopted by more than fifteen international networking standards. In 2015, he was elevated to the Fellow of IEEE for contributions to VLSI design and implementation of FEC coding. His current research interests are in the area of Optimized VLSI Design for Digital Communications and Deep Learning.

Author contributions
JL completed the analysis, experiments and paper writing. ZW helped perform the analysis with constructive discussions and curate the manuscript. The authors read and approved the final manuscript.

Funding
This work was supported in part by the National Natural Science Foundation of China under Grant 62174084, 62104097 and in part by the High-Level Personnel Project of Jiangsu Province under Grant JSSCBS20210034, the Key Research Plan of Jiangsu Province of China under Grant BE2019003-4.

Availability of data and materials
Project name: NAS Face Detect. Project home page: https://sourceforge.net/projects/nas-face-detect/. Operating system(s): Ubuntu 18.04. Programming language: Python. License: FreeBSD

Declarations

Ethics approval and consent to participate
The article has been ethically approved and approved.

Competing interests
The authors declare that they have no competing interests.

Consent for publication
All presentations of the case report have been agreed for publication.

Received: 26 August 2021 Accepted: 2 April 2022
Published online: 16 May 2022

References
1. S. Zafeiriou, C. Zhang, Z. Zhang, A survey on face detection in the wild: past, present and future. Comput. Vis. Image Underst. **138**, 1–24 (2015). https://doi.org/10.1016/j.cviu.2015.03.015
2. P. Viola, M.J. Jones, Robust real-time face detection. Int. J. Comput. Vis. **57**, 137–154 (2004). https://doi.org/10.1109/iccv.2001.937709
3. B. Zhang, J. Li, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, Y. Xia, W. Pei, R. Ji, ASFD: automatic and scalable face detector (2020). arXiv:2003.11228
4. J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, S. Zafeiriou, Retinaface: single-stage dense face localisation in the wild (2019). arXiv:1905.00641

5. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014). arXiv:1409.1556
6. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition. In: Proceedings of the the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016), pp. 770–778. https://doi.org/10.1109/cvpr.2016.90
7. K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016), pp. 630–645
8. T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey. J. Mach. Learn. Res. **20**, 1997–2017 (2019)
9. Y. Liu, Y. Sun, B. Xue, M. Zhang, G.G. Yen, K.C. Tan, A survey on evolutionary neural architecture search. IEEE Trans. Neural Netw. Learn. Syst. (2021)
10. K. Zhang, Z. Zhang, Z. Li, Y. Qiao, Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Process. Lett. **23**, 1499–1503 (2016). https://doi.org/10.1109/lsp.2016.2603342
11. K. Zhang, Z. Zhang, H. Wang, Z. Li, Y. Qiao, W. Liu, Detecting faces using inside cascaded contextual CNN. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2017), pp. 3171–3179. https://doi.org/10.1109/iccv.2017.344
12. D. Triantafyllidou, P. Nousi, A. Tefas, Fast deep convolutional face detection in the wild exploiting hard sample mining. Big Data Res. **11**, 65–76 (2018). https://doi.org/10.1016/j.bdr.2017.06.002
13. R. Ranjan, V.M. Patel, R. Chellappa, Hyperface: a deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. IEEE Trans. Pattern Anal. Mach. Intell. **41**, 121–135 (2019). https://doi.org/10.1109/TPAMI.2017.2781233
14. P. Hu, D. Ramanan, Finding tiny faces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017), pp. 1522–1530. https://doi.org/10.1109/cvpr.2017.166
15. X. Shi, S. Shan, M. Kan, S. Wu, X. Chen, Real-time rotation-invariant face detection with progressive calibration networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018), pp. 2295–2303. https://doi.org/10.1109/cvpr.2018.00244
16. J. Han, D. Zhang, G. Cheng, N. Liu, D. Xu, Advanced deep-learning techniques for salient and category-specific object detection: a survey. IEEE Signal Process. Mag. **35**(1), 84–100 (2018). https://doi.org/10.1109/msp.2017.2749125
17. S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, S.Z. Li, Faceboxes: a CPU real-time face detector with high accuracy. In: IEEE International Joint Conference on Biometrics (IJCB) (2017). https://doi.org/10.1109/btas.2017.8272675
18. H. Zhang, X. Wang, J. Zhu, C.-C.J. Kuo, Fast face detection on mobile devices by leveraging global and local facial characteristics. Signal Process. Image Commun. (2019). https://doi.org/10.1016/j.image.2019.05.016
19. Y. Yoo, D. Han, S. Yun, EXTD: extremely tiny face detector via iterative filter reuse (2019). arXiv:1906.06579
20. Y. He, D. Xu, L. Wu, M. Jian, S. Xiang, C. Pan, Lffd: a light and fast face detector for edge devices (2019). arXiv:1904.10633
21. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018), pp. 4510–4520. https://doi.org/10.1109/cvpr.2018.00474
22. R.J. Wang, X. Li, C.X. Ling, Pelee: a real-time object detection system on mobile devices. Adv. Neural Inf. Process. Syst. **31**, 1963–1972 (2018)
23. J. Redmon, A. Farhadi, Yolov3: an incremental improvement. arXiv (2018)
24. A. Bochkovskiy, C.-Y. Wang, H.-Y.M. Liao, Yolov4: optimal speed and accuracy of object detection (2020). arXiv:2004.10934
25. Ultralytics: Yolov5. (2021). https://github.com/ultralytics/yolov5
26. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, A.C. Berg, SSD: single shot multibox detector. In: Proceedings of the European Conference on Computer Vision (ECCV) (2016), pp. 21–37
27. Z. Tian, C. Shen, H. Chen, T. He, FCOS: fully convolutional one-stage object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2019), pp. 9627–9636. https://doi.org/10.1109/iccv.2019.00972
28. X. Zhou, D. Wang, P. Krähenbühl, Objects as points (2019). arXiv:1904.07850
29. B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning (2016). arXiv:1611.01578
30. B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018), pp. 8697–8710. https://doi.org/10.1109/CVPR.2018.00907
31. H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
32. M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q.V. Le, Mnasnet: platform-aware neural architecture search for mobile. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019), pp. 2815–2823. https://doi.org/10.1109/CVPR.2019.00293
33. Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, K. Nishida, Adaptive stochastic natural gradient method for one-shot neural architecture search. In: Proceedings of the 36th International Conference on Machine Learning (ICML) (2019), pp. 171–180
34. H. Cai, L. Zhu, S. Han, ProxylessNAS: direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019)
35. X. Dong, Y. Yang, Searching for a robust neural architecture in four GPU hours. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019), pp. 1761–1770. https://doi.org/10.1109/CVPR.2019.00186
36. H. Liu, K. Simonyan, Y. Yang, Darts: differentiable architecture search. In: International Conference on Learning Representations (2019)
37. B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, FBNet: hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019), pp. 10734–10742. https://doi.org/10.1109/cvpr.2019.01099

38. E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33 (2019), pp. 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780
39. G. Kyriakides, K. Margaritis, Regularized evolution for macro neural architecture search. In: IFIP International Conference on Artificial Intelligence Applications and Innovations (2020), pp. 111–122
40. R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al., Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing (2019), pp. 293–312. https://doi.org/10.1016/B978-0-12-815480-9.00015-3
41. M. Tan, R. Pang, Q.V. Le, EfficientDet: scalable and efficient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2020), pp. 10781–10790. https://doi.org/10.1109/cvpr42600.2020.01079
42. N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, Y. Zhang, NAS-FCOS: fast neural architecture search for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2020), pp. 11943–11951. https://doi.org/10.1109/cvpr42600.2020.01196
43. H. Xu, L. Yao, W. Zhang, X. Liang, Z. Li, Auto-FPN: automatic network architecture adaptation for object detection beyond classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (2019), pp. 6649–6658. https://doi.org/10.1109/iccv.2019.00675
44. G. Ghiasi, T.-Y. Lin, Q.V. Le, NAS-FPN: learning scalable feature pyramid architecture for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019), pp. 7036–7045. https://doi.org/10.1109/cvpr.2019.00720
45. A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: efficient convolutional neural networks for mobile vision applications (2017). arXiv:1704.04861
46. S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (NIPS) (2015), pp. 91–99. https://doi.org/10.1109/tpami.2016.2577031
47. S. Yang, P. Luo, C.C. Loy, X. Tang, WIDER FACE: a face detection benchmark. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016), pp. 5525–5533. https://doi.org/10.1109/cvpr.2016.596
48. V. Jain, E. Learned-Miller, FDDB: a benchmark for face detection in unconstrained settings. Technical report, Technical Report UM-CS-2010-009, University of Massachusetts, Amherst (2010)
49. Y. Zhu, H. Cai, S. Zhang, C. Wang, Y. Xiong, TinaFace: strong but simple baseline for face detection (2020). arXiv:2011.13183

Publisher's Note