International Journal of Computer Vision (2021) 129:1087–1105
https://doi.org/10.1007/s11263-020-01415-x

AutoDet: Pyramid Network Architecture Search for Object Detection

Zhihang Li[1,2] · Teng Xi[3,4] · Gang Zhang[3] · Jingtuo Liu[3] · Ran He[1,2]

Received: 3 January 2020 / Accepted: 4 December 2020 / Published online: 6 January 2021

Abstract
Feature pyramids have delivered significant improvement in object detection. However, building effective feature pyramids heavily relies on expert knowledge, and also requires strenuous efforts to balance effectiveness and efficiency. Automatic search methods, such as NAS-FPN, automates the design of feature pyramids, but the low search efficiency makes it difficult to apply in a large search space. In this paper, we propose a novel search framework for a feature pyramid network, called AutoDet, which enables to automatic discovery of informative connections between multi-scale features and configure detection architectures with both high efficiency and state-of-the-art performance. In AutoDet, a new search space is specifically designed for feature pyramids in object detectors, which is more general than NAS-FPN. Furthermore, the architecture search process is formulated as a combinatorial optimization problem and solved by a Simulated Annealing-based Network Architecture Search method (SA-NAS). Compared with existing NAS methods, AutoDet ensures a dramatic reduction in search times. For example, our SA-NAS can be up to 30x faster than reinforcement learning-based approaches. Furthermore, AutoDet is compatible with both one-stage and two-stage structures with all kinds of backbone networks. We demonstrate the effectiveness of AutoDet with outperforming single-model results on the COCO dataset. Without pre-training on OpenImages, AutoDet with the ResNet-101 backbone achieves an AP of 39.7 and 47.3 for one-stage and two-stage architectures, respectively, which surpass current state-of-the-art methods.

Keywords Object detection · Neural architecture search · Feature pyramids

1 Introduction

Object detection, aiming at localizing any instances of objects from given categories in an image, is a fundamental and challenging problem in computer vision (Liu et al. 2019b). In recent years, deep convolutional neural network (CNN) (Krizhevsky et al. 2012; Simonyan and Zisserman 2014; He et al. 2016) has been demonstrated as powerful methods for learning representations automatically. Particularly, object detectors based on CNN have made great progress and achieved state-of-the-art performance on various benchmarks (Everingham et al. 2015; Lin et al. 2014; Deng et al. 2009).

As has been revealed in Liu et al. (2019b), the performance of object detection methods is still challenged by imaging condition variations. In unconstrained environments, the appearance, shape and scale of objects vary largely due to illumination, cameras, occlusion, viewing distances and backgrounds. All of these factors cause vast intra-class variations. Although recent object detection methods have relied on invariant CNN representation to improve robustness, object detection in the wild remains challenging. In this study, we focus on studying the effect of scale variance.

To remedy the problem of large-scale variation, a commonly-used method depends on multi-scale image pyramids (Adelson et al. 1984; Dalal and Triggs 2005a; Lowe 2004). The input image is resized into multiple scales and then repeatedly fed to detectors repeatedly. Therefore, objects with different

Communicated by Mei Chen.

Zhihang Li and Teng Xi have contributed equally to this work. This work was done when Z. Li was an intern at Baidu Inc.

✉ Ran He
rhe@nlpr.ia.ac.cn

1 NLPR, CRIPAC, CEBSIT, CAS, Beijing, China

2 School of Artifical Intelligence, University of Chinese Academy of Sciences, Beijing, China

3 Department of Computer Vision Technology (VIS), Baidu Inc., Beijing, China

4 Department of Computer Science and Technology, Tsinghua University, Beijing, China

sizes can be detected under a proper resolution. The final result is generated by merging outputs of multiple forwards. Thus, the object detector concentrates on detecting objects under a specific scale instead of full ranges. Recent works (Singh and Davis 2018; Singh et al. 2018) based on image pyramids have verified the effectiveness of multi-scale testing. Nevertheless, the time-consuming inference of image pyramids hinders practical applications.

Another way to solve the problem is to leverage an inherent feature hierarchy, which already exists in CNN and has various spatial resolutions. The idea is inspired by divide and conquer, where objects are grouped according to scales and each feature layer is utilized to detect objects with specific scales. Actually, feature pyramids are originally used in traditional shallow learning methods (Dalal and Triggs 2005b), but they generate feature maps of different scales by image pyramids. Thus, both memory and inference time are significantly increased. In the following work (Dollár et al. 2014) feature pyramid generation was sped up via extrapolation from nearby scales. As the pioneering work based on CNN, SSD (Liu et al. 2016) and MS-CNN (Cai et al. 2016) make full use of inherently multi-scale features in CNN and detect objects with various scales on distinct layers. Recently, observing the lack of semantic information in low-level feature maps, Feature Pyramid Network (FPN) (Lin et al. 2017a) proposed a novel and efficient way for combining high-level and low-level features by a top-down pathway with lateral connections. Experiments demonstrate that feature fusing in FPN extensively boosts the performance of object detection.

FPN essentially enriches semantic information for all features. Hence, the connections and operations among these feature maps become crucial to the performance of detectors. Although FPN has been validated as simple and effective, many studies (Fu et al. 2017; Li and Zhou 2017; Kong et al. 2016; Bell et al. 2016) have conducted on seeking the better cross-layer connections and fusing operations to build feature pyramids. A variety of feature pyramid networks have are designed for various tasks. However, there have been no design principles for feature pyramids until now. As stated in Ghiasi et al. (2019), the design space grows at an exponential rate with an increasing the number of layers. Therefore, it is impossible to find the best feature pyramid network by enumerating all structures in such an enormous space.

Recently, the neural architecture search (NAS), which automatically derives the optimal neural network architecture from a search space, has emerged and led to state-of-the-art accuracy on classification (Xie and Yuille 2017; Zoph and Le 2016; Zoph et al. 2018; Liu et al. 2018a, 2017; Jenatton et al. 2017; Li et al. 2020; Hu et al. 2020), segmentation (Liu et al. 2019a) and image restoration (Suganuma et al. 2018). Due to the vast search space and expensive training cost, a series of works (Liu et al. 2018b; Zhang et al. 2019) has addressed NAS search architectures on proxy tasks. As shown in Table 1, NASNet (Zoph et al. 2018) and AmoebaNet (Real et al. 2018) require 2, 000 GPU days to search the classification network on the small-scale Cifar-10 dataset. DARTS (Liu et al. 2018b) only searches a block and then stacks them to build a network. The image size is set to 312×312 on searching for DPC (Chen et al. 2018) and Auto-DeepLab (Liu et al. 2019a). Accordingly, directly searching on a large dataset such as ImageNet or COCO is still challenging due to the high computational cost. The most relevant work here is NAS-FPN (Ghiasi et al. 2019), which searches cross-scale connections to generate multiscale feature representations. The merging cell in its search space takes two input layers and outputs a feature layer, where fusing operations only contain two methods (sum and global pooling). During the architecture search, NAS-FPN requires 333 TPU days to sample 12, 000 child networks.

As mentioned above, feature pyramids are an effective method for addressing the scale variance in object detection. The design space of the feature pyramid network is so large that the optimal structure for varied tasks is hard to find by hand. NAS is a data-driven method that automatically enables an optimal architecture search. However, its high computation costs pose a challenge for practical application.

To address this problem, we design an efficient feature pyramid architecture search framework. As illustrated in Fig. 1, we first build a comprehensive search space that contains abundant typologies and varied fusing cells. Then, we design an efficient search algorithm to discover a better feature pyramids network. The performance of child network on the validation set is used as reward.

As a matter of fact, it is non-trivial to directly apply the NAS of classification to object detection. First, the search space should be task-specialized. As stated in Liu et al. (2019a), the architecture of CNN involves a two-level hierarchy where the outer topology controls the spatial resolution changes and the inner cell steers layer-wise computation. A plethora of current works (Liu et al. 2018b, 2017, 2018a; Real et al. 2018; Pham et al. 2018) on NAS only search the inner cell while pre-defining the outer network topology. This limited search space is unsatisfactory for object detection. In contrast, the inputs of feature pyramids are multi-scale features with different resolutions. And the topological connection of layers and the feature fusing strategies appear more significant for feature pyramids.

In addition, object detection inherently has more computational complexity than classification. To speed up, some NAS works (Liu et al. 2018b, 2017) search on CIFAR-10 with low-resolution inputs (32×32) and then transfer the discovered architectures to a high-resolution dataset directly. This strategy is not suitable for object detection, which is sensitive to the input resolution. Therefore, searching architectures efficiently on high-resolution inputs remains challenging.

Table 1 The search time for different architecture search approaches with different image sizes

Models	Datasets	Image size	Days	Task
NASNet (Zoph et al. 2018)	CIFAR-10	32 × 32	2000	Cls
AmoebaNet (Real et al. 2018)	CIFAR-10	32 × 32	2000	Cls
PNASNet (Liu et al. 2018a)	CIFAR-10	32 × 32	150	Cls
DARTS (Liu et al. 2018b)	CIFAR-10	32 × 32	4	Cls
DPC (Chen et al. 2018)	Cityscapes	312 × 312	2600	Seg
Auto-DeepLab (Liu et al. 2019a)	Cityscapes	312 × 312	3	Seg
AutoDet	COCO	**512 × 512**	**2.2**	Det

Bold values indicate the larger image size and the least gpu-days
The main differences from others include: (1) we search the network architecture for the object detection task, (2) we search directly on the challenging COCO dataset, and (3) our method is quite efficient and requires only 2.2 P40 GPU days

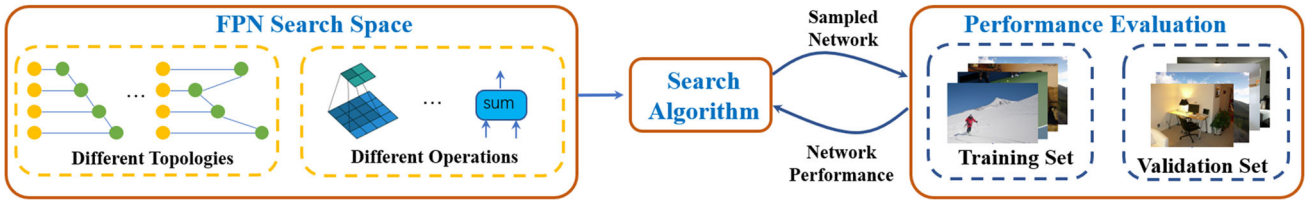


Fig. 1 Illustration of feature pyramid network search for object detection. The search spaces contain different connection typologies and feature fusing operations. The search algorithm first samples a candidate network from the search space. The network is trained on the training set, and then the performance on the validation set is returned to the search algorithm. Eventually, the search algorithm can discover the optimal architecture

The search strategies play an important role in an efficient architecture search. The popular methods, including reinforcement learning and evolutionary algorithms, are limited by intensive computations even on the small-scale CIFAR-10 database. Recently, differentiable architecture search eliminates the meta-controller and trains an over-parameterized supernet containing all candidate paths, which largely improve the search efficiency. Nevertheless, its over-parameterized supernet not only consumes a large amount of memory, but also limits the flexibility for searching typologies.

In this paper, we propose a novel framework, AutoDet, to automatically search the feature pyramid structure for object detection. In AutoDet, in order to search in-cell structure and outer topology at the same time, we design a new combinatorial search space for the feature pyramid network. Instead of only binary fusing operations (sum and global pooling) in NAS-FPN (Ghiasi et al. 2019), we design a flexible fusing strategy where the number of channels on each layer is searchable in our search space. Thus, our AutoDet is more general and includes NAS-FPN (Ghiasi et al. 2019) as a special case. Moreover, aiming at searching architectures directly on the COCO dataset at affordable cost, we introduce a fast Simulated Annealing (SA) algorithm, instead of reinforcement learning (RL) or evolutionary algorithm (EA), to search the state-of-the-art architectures in the search space.

To verify the effectiveness and flexibility of our AutoDet, we search feature pyramids directly on the COCO dataset for one-stage and two-stage detectors. With the ResNet-50 backbone model in one-stage SSD, AutoDet achieves AP 32.3 and 36.2 with the input sizes of 320 × 320 and 512 × 512, which outperform the vanilla FPN by 1.9 and 2.6 AP. When the backbone is replaced with ResNet-101, the improvements are also significant. For two-stage Faster R-CNN, we evaluate AutoDet with various backbone models, including ResNet-50 and ResNet-101. For example, AutoDet based the ResNet-50 as backbone achieves 40.2 AP, which is superior to FPN by 5.9 AP. Finally, the searched network can achieve AP 39.7 and 47.3 for one-stage and two-stage detectors, both of which outperform the results of state-of-the-art methods.
The contributions of this paper are four-fold:

- We propose a novel pyramid network architecture search method that can be applied to both one-stage and two-stage object detection, which is one of the pioneers for automatically searching the network architecture for the detection task.
- A novel pyramid network search space, including in-cell block architecture as well as outer topology, is designed for object detection.
- A novel heuristic Simulated Annealing-based Network Architecture Search (SA-NAS) is introduced to acceler-

ate the search process. Our method can be 30 times faster than RL-based approaches.
– The performance of the proposed solution is evaluated thoroughly via experiments on the COCO dataset. The results of one-stage and two-stage structures surpass state-of-the-art methods, which is a strong verification of the superiority of our automatic solution.

2 Related Work

2.1 Object Detection

Benefiting from the development of CNN (Krizhevsky et al. 2012; Simonyan and Zisserman 2014; He et al. 2016) in recent years, object detection based on CNN has achieved dramatic improvement in performance. Current object detection algorithms can be roughly divided into one-stage and two-stage detectors.

Two-stage methods, such as R-CNN (Girshick et al. 2014), fast R-CNN (Girshick 2015) and faster R-CNN (Ren et al. 2015), first generate the region proposals by selective search or a region proposal network (RPN). CNN features on these regions are extracted and classifiers are used to determine the label of proposals. Dai et al. (2016) proposed RFCN to further improve the detection speed by sharing the region-wise sub-network. Mask RCNN (He et al. 2017) extends Faster RCNN to tackle instance segmentation where a parallel branch in the second stage predicts a binary mask for each RoI. Merely using deep features with high-level semantic features but low-resolution makes it difficult for these methods to detect small objects.

The seminal works of one-stage detectors are SSD (Liu et al. 2016) and YOLO (Redmon and Farhadi 2018), which explore an end-to-end solution without proposals. YOLO (Redmon and Farhadi 2018) casts object detection as a bounding box regression problem from the input image. YOLOv2 (Redmon and Farhadi 2017) adopts DarkNet19 as the backbone and generates good anchor boxes by k-means. YOLO9000 can detect over 9000 object catergories in real time. SSD (Liu et al. 2016) makes full use of inherently multi-scale features in CNN and detects objects with different scales on different levels of layers, where shallow layers with high-resolution are used to detect small objects and deep layers with large reception fields are used to detect large objects. However, object detection requires not only locating objects but also classifying them, which means that both high-level semantics and low-level spatial details are significant. Thus, leveraging high-level or low-level features individually for detection is sub-optimal. Recently, inspired by pose estimation, Law and Deng (2019) formulated object detection to the keypoint regression problem and proposed CornerNet to detect paired top-left and bottom-right keypoints. The backbone is Hourglass network and corner pooling is used to localize corners. CornerNet outperforms all one-stage detectors, but the inference time is significantly slower than SSD and YOLO.

2.2 Feature Fusion

Feature fusion has been demonstrated to significantly improve the performance of object detection. To utilize the information propagation between different levels of feature maps, FCN (Long et al. 2015) and U-Net (Ronneberger et al. 2015) fuse the lower level feature map information via skip connections. In addition, DSSD (Fu et al. 2017) uses the deconvolution as an up-sampling method, and then combines the information of different levels. Similarly, top-down modulation (Shrivastava et al. 2016), RetinaNet (Lin et al. 2018) and FPN (Lin et al. 2017a) use a top-down pathway and lateral connections to enrich information from different layers. In FSSD (Li and Zhou 2017), features from different layers with different scales are concatenated together, followed by some down-sampling blocks to generate a new feature pyramid. The stacked hourglass network (Newell et al. 2016) uses skip connections between low-level and high-level feature maps to make full use of the information extracted from these feature maps. Inside-Outside Net (ION) (Bell et al. 2016) extracts fixed-size feature maps from several layers using RoI Pooling for object detection. HyperNet (Kong et al. 2016) combines feature maps by leveraging local response normalization and concatenation to generate the hyper feature map used for object detection. PANet (Liu et al. 2018d) fuses the FPN's output again from bottom to up, which shortens the information path between lower layers to higher features. Different from previous works simply summing them up without distinction, EfficientDet (Tan et al. 2020) proposes a weighted bi-directional feature pyramid network to learn the importance of different input features. Feature-based attention (Chen et al. 2016; Wang et al. 2017) (also known as channel-wise attention) is another form of feature fusion mechanism and involves learning a task-oriented modulation. M2Det (Zhao et al. 2019) aggregates multi-level multi-scale features with the same scale to construct a feature pyramid by a scale-wise feature aggregation module. All of the above discoveries on network architectures for feature fusion require substantial effort from human experts, while our work implements an automatic solution that can outperform the manual solutions.

2.3 Neural Architecture Search

Recently, NAS has achieved highly competitive performance in image classification. It can be roughly divided into three categories: RL-based approaches (Baker et al. 2017; Tan et al.

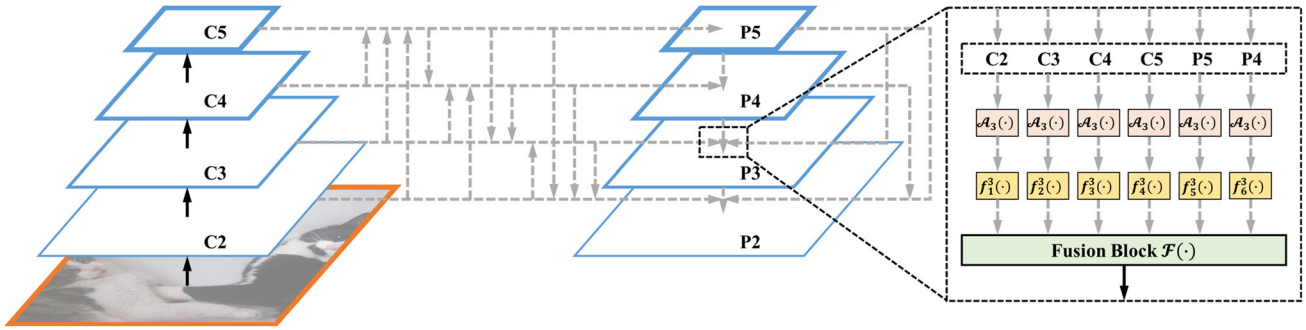


Fig. 2 The illustration of the AutoDet search space. The black dashed box is the feature generation cell. We take the P_3 generation process as an example. The gray dashed lines present all possible links in the AutoDet search space

2018; Zoph and Le 2016; Zoph et al. 2018; Liu et al. 2018a; Tan and Le 2019; Zhong et al. 2018; Cai et al. 2018), EA-based approaches (Liu et al. 2017; Xie and Yuille 2017; Real et al. 2018; Elsken et al. 2018), and differentiable approaches (Liu et al. 2018b; Xie et al. 2018; Luo et al. 2018; Brock et al. 2018; Bender et al. 2018; Wu et al. 2019; Dong and Yang 2019).

Since RL is a fundamentally more difficult problem than optimization (Jiang et al. 2017), RL-based approaches tend to explore and exploit a very large number of architectures to find the nearly optimal solution. EA-based approaches (Real et al. 2018) improve evolutionary algorithms to optimize network structures, which suffer from enormous computational costs for the whole evolution. Due to the expensive search time shown in Table 1, most of RL and EA-based NAS methods first search on small proxy datasets, such as CIFAR-10 with an input size of 32×32. Then, they train the searched network on target datasets. Differentiable approaches such as DARTS (Liu et al. 2018b) reduce computational costs by applying a continuous relaxation of architecture representation to make gradient descent possible for optimization. However, a heavy super-net contains all sub-nets in the search space consume considerable GPU memory. FBNet (Wu et al. 2019) and GDAS (Dong and Yang 2019) simply sample and train a sub-net during the search stage. It not only reduces the memory but also decouples different operations within a layer. SNAS (Xie et al. 2018) tackles the problem by parameter optimization on a joint distribution of architecture space and improves the search efficiency via a generic differentiable loss. Due to the assumption of a fixed number of filters, these methods are weakened by limited search spaces. In addition, most of the rapid methods only use the same searched structure in all cells and neglect the topology between them, which is important in pyramid feature fusion. Recently, NAS-FPN (Ghiasi et al. 2019) searches the topology and binary fusing operations of feature pyramids with RL. Our AutoDet is the pioneer to apply the efficient NAS method to object detection tasks and simultaneously searches cell-level networks and pyramid-level structures.

3 Method

The current object detection framework based on CNN generally contains two main components: a backbone network and a feature pyramid network. Specifically, the backbone network is a pre-trained classification model, that is used to extract multi-scale features from an input image. Due to the multiple levels of processing by CNN (Lin et al. 2017a), feature maps of shallow layers contain more low-level information than semantics while feature maps of deeper layers are enriched with semantic information. To jointly utilize semantic features as well as local details, a feature pyramid network (Lin et al. 2017a) is proposed to fuse different levels of layers.

Since designing a feature pyramid network requires expert knowledge and its variants are too large to enumerate, AutoDet provides a Neural Architecture Search (NAS) solution, which automatically discovers the optimal structure of feature pyramids from a tailor-made search space. As shown in Fig. 1, NAS (Elsken et al. 2019) is known to consist of three components: search space, search strategy, and performance estimation strategy. The search space defines which architectures can be represented. The search strategy details how to explore the search space. And the performance estimation strategy is used to measure the performance of child networks. In general, a larger search space may contain more child networks and the better network architectures, but bring challenges for efficient search. Hence, the search space plays an important role in NAS. Moreover, some studies (Liu et al. 2017; Real et al. 2018; Liu et al. 2018b; Zheng et al. 2019) show that the search strategy is one of the important factors for accelerating search and discovering the optimal solution.

In this section, we begin by introducing a search space for the feature pyramid network. Then, we demonstrate how to prune the search space for a more efficient search. Finally, we devise a novel Simulated Annealing-based Network Architecture Search to obtain the pyramid network architecture.

3.1 Search Space Design

In this section, we present the search space for feature pyramid network. The generation of feature pyramid can be divided into two steps: for each output layer, we first determine a number of input layers, and then select the best fusing operations for them. Therefore, we design a two-level search space. For the outer network level, we search the connection topology. For the inner feature fusing cell level, we search the operation parameters.

The input of a feature pyramid network consists of a set of feature maps from the backbone with different scales $\mathcal{C} \triangleq \{c|C_{k_1}, C_{k_2}, \ldots, C_{k_n}\}$. In Fig. 2, $\mathcal{C} = \{C_2, C_3, C_4, C_5\}$ represents the outputs of $conv2$, $conv3$, $conv4$, and $conv5$ with the corresponding strides of {4, 8, 16, 32} pixels. Owing to the large memory consumption, we do not integrate the $conv1$ into the pyramid. Note that the space is compatible with $\mathcal{C}$ in a wide range of backbones. The output of a feature pyramid network has the identical scales as the input after feature fusion. Figure 2 shows the output features $\mathcal{P} = \{P_2, P_3, P_4, P_5\}$. Let $\mathcal{I}$ denote the set of indexes in $\mathcal{C}$ and $\mathcal{O}$ denote the set of indexes in $\mathcal{P}$. For Fig. 2, $\mathcal{I} = \mathcal{O} = \{2, 3, 4, 5\}$.

AutoDet generates the target features $\{P_2, P_3, P_4, P_5\}$ sequentially. When generating each target feature, a feature generation cell takes all possible previous nodes $\mathcal{C}$ as input and outputs a feature map in $\mathcal{P}$. Since the input features have different resolutions, they are first aligned to the same resolution by up-sampling, or down-sampling. Then, we append the auto transformation function $\mathcal{A}_j(.)$ to reduce the aliasing effect of sampling and preserve the discriminability of aligned feature, which is a fixed CNN function with ReLU activation and Batch Normalization (BN). Next, we utilize layer-specific operation function $f_i^j(.)$ to process each layer for feature fusing, which is subject to operation type and filter number, where i refers to the index of connected previous nodes and j refers to the index of output P_j, $\forall i \in \mathcal{I}$, $\forall j \in \mathcal{O}$. Finally, an fusion function $\mathcal{F}_j(.)$ is used to merge these features and generate target feature. Note that previously generated features are immediately added into the candidate set. In other words, when generating P_j, the candidate set not only contains the output features of backbone $\{C_2, C_3, C_4, C_5\}$, but also has previously generated pyramids $\{P_5, P_4, \ldots, P_{j-1}\}$.

For the outer connection topology, all gray dashed lines in Fig. 2 show possible connections. Each connection has two discrete states {0, 1}, which denote whether to connect the preceding node. For the inner feature fusing cell, the computation CNN operations include operation type and filter number. Here, we add "no connection" as an operation into the cell. In this way, the outer topology and inner cell operation can be unified into a search framework. Therefore, we formulate the pyramid network architecture search as a combinatorial optimization problem. All types of operations are defined as follows:

- 1×1 conv
- 5×5 conv
- 3×3 conv
- no connection

The set of filter numbers is discretized as {128, 256, 384, 512}. AutoDet can also generalize to other discrete spaces. Here, the purpose of the "no connection" operator is to determine whether the previous feature is chosen for output feature generation, which is also used DARTS (Liu et al. 2018b) for network topology search.

To generate the j-th output of feature pyramids P_j, k preceding nodes $\{c_1, c_2, \ldots, c_k\}$ are first aligned to the same resolution by down- and up-sampling operations. Then, auto transformation function $\mathcal{A}_j(.)$ is applied to preserve discrimination. To search the operation types and filter numbers, we utilize layer-specific operation function $f_i^j(.)$. Finally, a simple element-wise summation is used to fuse these features. Overall, the j-th output of feature representation P_j is formulated as follows:

$$P_j = \mathcal{F}(f_1^j(\mathcal{A}_j(c_1)), f_2^j(\mathcal{A}_j(c_2)), \ldots, f_k^j(\mathcal{A}_j(c_k))),$$
$$c_i \in \mathcal{N}_j, \forall j \in \mathcal{O}, \forall i \in \{1, 2, \ldots, k\}, k = |\mathcal{N}_j|. \quad (1)$$

In equation (1), $\mathcal{N}_j$ is the set of previous nodes of P_j, where,

$$\mathcal{P}_i \in \mathcal{N}_j, \forall i > j. \quad (2)$$

For Fig. 2, we have:

$$\mathcal{N}_5 = \{C_2, C_3, C_4, C_5\},$$
$$\mathcal{N}_4 = \{C_2, C_3, C_4, C_5, P_5\},$$
$$\mathcal{N}_3 = \{C_2, C_3, C_4, C_5, P_5, P_4\},$$
$$\mathcal{N}_2 = \{C_2, C_3, C_4, C_5, P_5, P_4, P_3\} \quad (3)$$

3.2 Simulated Annealing-Based Network Architecture Search

We propose a Simulated Annealing-based Network Architecture Search (SA-NAS) to obtain the pyramid network architecture. Simulated Annealing (SA) (Chopard and Tomassini 2018) is a probabilistic algorithm that makes a good approximation to the global optimal solution of the optimization problem in a large search space. The simulated annealing (SA) is a stochastic optimization approach that simulates the physical annealing process. SA initially sets the temperature high and then allows it to slowly 'cool' as the algorithm runs. While this temperature is high the algorithm will be allowed,

Algorithm 1: SA-NAS

Input:
$\mathcal{C}$, $\mathcal{O}$;
Output:
$\mathcal{P}$;
Initialize T°, ξ, and $Iteration\_num$;
Initialize the vector of decision variable var;
Initialize the initial reward r;
while $Iteration\_num > 0$ **do**
  Generate a neighbor vector of var, var_n;
  Generate $\mathcal{N}_j$ according to equation (3);
  Get $f_i^j(.)$ according to var_n, subject to the pruned search space.
  Get $\mathcal{P}_j$ according to equation (1);
  Train the model with AutoDet for a small number of epochs and get sub reward r_n, where $\mathcal{C}$ is frozen;
  $\Delta r = r_n$ - r;
  if $\Delta r > 0$ **then**
    $r = r_n$;
    $var = var\_n$;
  else
    if $rand(0, 1) < exp(-\Delta r/T^{\circ}))$ **then**
      $r = r_n$;
      $var = var\_n$;
      $T^{\circ} = T^{\circ} * \xi$;
    end
  end
  $Iteration\_num - -$;
end
return(var, r);

with more frequency, to accept solutions that are worse than our current solution. As the temperature is reduced, SA gradually focuses on an area of the search space and converge to an optimum solution. SA can efficiently obtain a near-optimal solution of a wide range of NP-hard problems, such as the traveling salesman problem (TSP), set cover problem (SCP), and max coverage problem (MCP). Since the pyramid network architecture search is similar to TSP and SCP, we can tackle it by SA to obtain the near-optimal solution more efficiently.

It should be mentioned that, the traditional SA cannot directly deal with the pruned search space. SA-NAS improves SA on the neighbor generation phase. Specifically, it divides the neighbor generation into three subphases. First, it chooses the target $\mathcal{P}_j$ by equal probability, $\forall j \in \mathcal{O}$. Then, based on the probability of combinatorial optimization, it decides the number of previous nodes that link to $\mathcal{P}_j$. Finally, it selects operations for each input layer.

Algorithm 1 shows the pseudocode of the SA-NAS algorithm, in which the goal is to find the best vector of decision variables, denoted as var. With the best var, the reward of AutoDet, r, is maximized. For the object detection task, the reward is the bounding box average precision. The SA-NAS complies with the following steps.

Step 1: Generate Initial State

Initially, the SA-NAS generates a feasible var and r as a starting point. For example, the SA-NAS can take the network architecture of the FPN as the initial state. To simplify the expression, we use the current state to denote the decision variable of the current iteration.

Step 2: Generate the Neighbor State of Current State.

In each iteration, a list of neighbor vectors (denoted as var_n) of var is generated. As mentioned above, the neighbor state has to be compatible with the pruned search space.

Step 3: Decode Neighbor State and get reward.

The previous nodes are first mapped by the auto transformation function $\mathcal{A}_j(.)$. SA-NAS then obtains the CNN operation function $f_i^j(.)$ and transforms the features according to $f_i^j(.)$. $\mathcal{P}_j$ is next calculated by fusing $f_i^j(\mathcal{A}_j(c))$, $i \in \{1, 2, 3, 4\}$. The whole process is formulated as equation (1).

Based on the decoded pyramid network architecture, SA-NAS trains for a small number of epochs and gets the reward r_n, where the backbone $\mathcal{C}$ is frozen.

Step 4: Update Current State.

If the new reward r_n is higher than current reward r, then we accept var_n as a feasible solution. Otherwise, we accept var_n based on a probability of acceptance to avoid falling into a local minimum. The annealing temperature T° is updated by a factor of ξ. The probability of acceptance is an exponentially decreasing function with parameter $exp(-\Delta r/T^{\circ}))$. After each iteration, the probability of acceptance decreases.

Step 5: Back to Step 2.

The while loop continues until reaching the iteration threshold. After the while loop, we can obtain an approximate optimal vector of decision variable var and its corresponding reward r.

4 Experiments

In this section, we first present the implementation details of the architecture search as well as the experimental setup. Then, we conduct an ablation study of our approach on the COCO dataset (Lin et al. 2014). We next compare our SA-NAS with other automatic search methods in terms of performance and search efficiency. To validate the effectiveness of the searched feature pyramid architecture through SA-NAS, we compare the results with state-of-the-art methods on the challenging COCO dataset (Lin et al. 2014). Then, we visualize the discovered feature pyramid architecture and summarize inspiring findings for architecture designs. To this end, we apply SA-NAS to search the backbone for on-device real-time image detection and classification.

4.1 Datasets and Metrics

The Microsoft COCO (Lin et al. 2014) dataset is one of the most challenging datasets for object detection in the wild. The COCO dataset has 80 classes with bounding box annotations. It consists of 80k images for training and 40k images for validation. Following (Bell et al. 2016; Lin et al. 2017a), we train models on the union of the training set and 35k subset of the validation set (trainval35k). The remaining 5k images are used for the validation set (minival). We evaluate our method on the test-dev set for fair comparisons, which contains 20k images. Since ground-truth labels of test-dev are not publicly available, we submit all results to the test-dev evaluation server. We follow the standard evaluation metrics, including average precision (AP) at different IoUs, $AP_{@0.5}$, $AP_{@0.75}$ and APs for different object sizes AP_S, AP_M and AP_L.

4.2 Implementation Details

We first search the best feature pyramid architecture directly on COCO through SA-NAS. Then we train the whole object detection to obtain the final model. Here, we describe our experimental setup of searching and training final object detection, respectively.

4.2.1 Experimental Setup of Searching

We follow the same training protocol in Liu et al. (2016) for the one-stage architecture and that in He et al. (2017) for the two-stage architecture. The proposals are generated from an independently trained RPN (He et al. 2017; Ren et al. 2015) to allow convenient ablation and fair comparison. We search architecture on the 512×512 image resized from the COCO dataset. To speed up the training of SA-NAS, we adopt a short training time as in Ghiasi et al. (2019), Zoph et al. (2018) that also correlates with the performance of the detector after converging. Specifically, we use AP on the validation set at 6, 400 iterations as the reward in SA-NAS. The set of indexes $\mathcal{I}$ and $\mathcal{O}$ are set as {2, 3, 4, 5}. The control number $|N_j|$ of the AutoDet search space is set to 4. The initial temperature in the SA-NAS is set to 2^{10} and the annealing rate is set to 0.85. The iteration times of the SA-NAS are set to 200. During the searching phase, we use a ResNet-50 backbone pre-trained on ImageNet with all parameters frozen.

4.2.2 Experimental Setup of Object Detection

We retrain the final object detector after discovering the best feature pyramid network. The hyperparameters in the final training are slightly different from those in the architecture search stage. For two-stage and one-stage objectors, we adopt the same end-to-end training process as in Faster R-CNN (Ren et al. 2015) and SSD (Liu et al. 2016). We still use the pre-trained model from ImageNet, but the parameters are not frozen as in the search stage. To further validate the generalization of the searched feature pyramid network, we evaluate on different backbones, such as ResNet-50 and ResNet-101.

For Faster R-CNN, we take 8 images in a batch for training and use 8 NVIDIA P40 GPUs (one image per GPU). For a fair comparison, we do not use pre-training data from OpenImages,[1] nor do we use Sync-BN (Peng et al. 2018) and deformable convolutions (Dai et al. 2017). The resolution of the input training images is $800 \times 1,333$, if not specifically noted. We train our model with a learning rate starting from 0.01, and it is decreased by a factor of 0.1 after 960k and 1, 280k iterations and finally terminates at 1, 440k iterations. This training schedule results in 12.17 epochs. The rest of the hyperparameters remain the same as the Faster R-CNN.

For SSD, the batch size is set to 64 with the input size 320×320, and the batch size is set to 32 with the input size 512×512 due to the GPU memory constraint. For gradient descent, we use the SGD optimizer with momentum 0.9 and weight decay 0.0001. The cosine learning rate is adopted, and the initial learning rate 0.04 is applied for first 50k iterations . The warm up learning rate is 0.013333 for the first 2k iterations. The batch normalization layers are applied after all convolution layers. The weight decay is 0.0001 and the momentum is 0.9.

4.3 Ablation Study

In this subsection, we comprehensively evaluate our method on the COCO dataset (Lin et al. 2014).

4.3.1 One-Stage Versus Two-Stage

AutoDet is a flexible and generalized architecture search framework, that is easy to be embeded into one-stage and two-stage detectors. The typical one-stage and two-stage detectors are SSD (Liu et al. 2016) and Faster RCNN (Ren et al. 2015). Therefore, we directly search feature pyramid networks with SSD (Liu et al. 2016) and Faster RCNN (Ren et al. 2015) respectively on the COCO dataset. The searched feature pyramid networks are compared with the human-invented FPN (Lin et al. 2017a). For a fair comparison, we report the performance of a single model at the single scale.

Tables 2 and 3 show the results of one-stage and two-stage detectors, respectively. Two-stage detectors always outperform one-stage detectors regardless of the backbone is used, because the two-stage scheme in the Faster RCNN enables a more accurate position regression and classification. For the one-stage detector in Table 3, AutoDet is consistently supe-

[1] Dataset available from https://storage.googleapis.com/openimages/web/index.html.

Table 2 Comparisons of ablation results on the COCO object detection test-dev benchmark on **single model**

Method	Backbone	AP	FPS
FPN(Lin et al. 2017a)	ResNet-50	34.3	26.3
FPN*	ResNet-50	37.5	26.3
AutoDet	ResNet-50	**40.2**	25.4
FPN(Lin et al. 2017a)	ResNet-101	36.2	19.6
FPN*	ResNet-101	40.1	19.6
AutoDet	ResNet-101	**43.3**	18.8

Bold values indicate the best performance which model achieves
We make ablation comparison on two-stage architecture Faster R-CNN based with both ResNet-50 and ResNet-101 backbones. $*$ indicates we re-implement it with the same training settings

Table 3 Comparisons of ablation results on the COCO object detection benchmark on **single model**

Method	Backbone	Input size	AP	FLOPs	Params
FPN	ResNet-50	320 × 320	30.4	48.58G	35.59M
AutoDet	ResNet-50	320 × 320	**32.3**	48.75G	36.38M
FPN	ResNet-50	512 × 512	33.6	124.36G	35.59M
AutoDet	ResNet-50	512 × 512	**36.2**	124.79G	36.38M
FPN	ResNet-101	320 × 320	31.2	56.18G	54.58M
AutoDet	ResNet-101	320 × 320	**34.5**	56.34G	55.37M
FPN	ResNet-101	512 × 512	34.4	143.81G	54.58M
AutoDet	ResNet-101	512 × 512	**37.7**	144.24G	55.37M

Bold values indicate the best performance which model achieves
We make ablation comparison on one-stage architecture SSD based with ResNet-50 and ResNet-101 backbones

rior to FPN under various experimental setups. For example, with input size 320 × 320 and ResNet-101 backbone, feature pyramids searched by AutoDet achieve 34.5% AP. The improvement is impressive (i.e., 3.3%) compared with the FPN, which shows the superiority of AutoDet. Furthermore, more remarkable promotion is achieved on the two-stage detector. Although Faster RCNN with a FPN is a stronger baseline, AutoDet greatly improves the AP from 36.2% to 43.3% (approximately 6.1% AP improvement) with ResNet-101 in Table 2. Moreover, we also adopts the same training settings to re-train the baseline method (Lin et al. 2017a) (FPN*). Although the FPN* indeed improves the AP value, it is still much inferior to our method. Overall, consistent experimental results on both one-stage and two-stage detectors verify the generalization of AutoDet.

4.3.2 Search on Different Backbone Architectures

We evaluate the AutoDet on different backbone architectures, including ResNet-50 and ResNet-101, to demonstrate the effectiveness of the discovered feature pyramids. As shown in Tables 2 and 3, we find that the weaker backbone ResNet-50 is usually inferior to the stronger backbone ResNet-101, since ResNet-101 provides more powerful representations as input to the feature pyramid network. Moreover, regardless of what backbone is utilized, AutoDet always produces competitive AP scores and is better than its counterpart FPN. For example, in the two-stage strategy, AutoDet with ResNet-50 improves the AP scores from 34.3% to 40.2%, and AutoDet with ResNet-101 improves the AP scores from 36.2% to 43.3%. These improvements can be mainly attributed to two aspects: First, manually designed FPN is not the optimal feature pyramid for object detection. Second, since different backbone architectures provide the input features with different receptive fields and semantic levels, a unified feature pyramid structure (like FPN) cannot be adapted to various backbones. Therefore, we prefer the customized and specifically designed feature pyramids for individual backbone architectures. Experiments show that AutoDet is an alternative method for dynamically searching feature pyramid network for different backbones. Some qualitative results comparison between FPN and AutoDet on the COCO dataset are presented in Fig. 3.

4.3.3 Search on Different Input Sizes

Except for the backbone, the input size of the image has been demonstrated to be an important factor in performance (Liu et al. 2016; Ren et al. 2015; He et al. 2017; Lin et al. 2017a). Intuitively, the input image with high-resolution conveys a more detailed appearance of the object, which is very beneficial to small object detection. Thus, relatively large objects in high-resolution images are usually detected on deep but large receptive field layers (such as P_4 and P_5). Conversely, because low-resolution input image have fewer cues for objects, shallow but high-resolution features (such as P_2 and P_3) in the backbone are more crucial. To analyze the above problems, experiments on the image sizes of 320×320 and 512×512 are conducted in a one-stage detector. Comparing the detection results between AutoDet and FPN in Table 3, we notice that due to a large number of small objects in COCO (Singh and Davis 2018), models with large input size are more advantageous than models with small input sizes. Although FPN obtains an AP gain of approximately 3.2% with ResNet-50 as the backbone, AutoDet presents impressive results, i.e., AP from 32.3% to 36.2%. In addition, AutoDet constantly performs better than FPN under different input sizes. Since hierarchical features in the backbone play different roles on different input sizes, an individualized feature fusing strategy in feature pyramid network is required. AutoDet drives a flexible and adaptive feature pyramids on different image sizes for better performance.

Inference speed. We evaluate the inference speed, number of parameters and FLOPs of AutoDet and the baseline approach in Tables 2 and 3. The speed is tested with batch

Fig. 3 Selected examples of comparison results between AutoDet(left) and FPN(right)

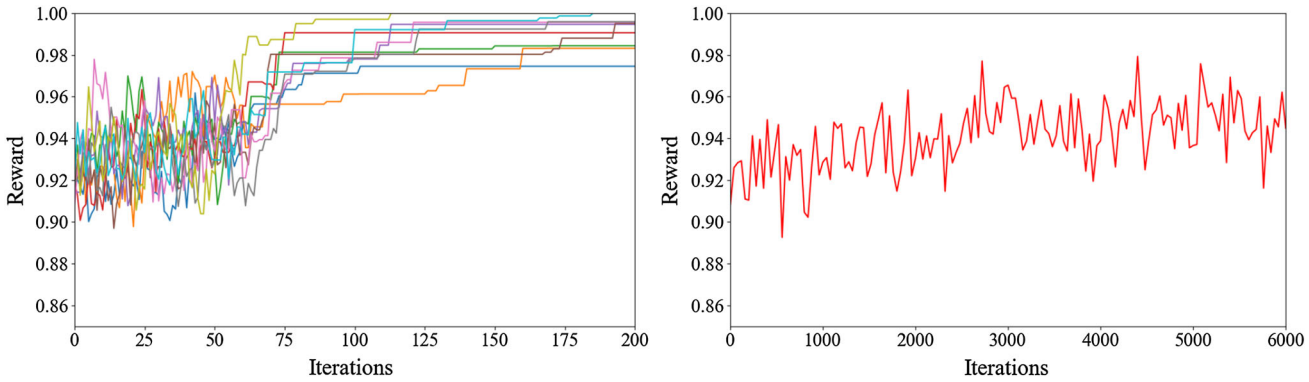


Fig. 4 The illustration of the reward increasing alongside iterations of SA-NAS(left) and RL(right). We can find a quite fast convergence of SA-NAS

size 1 on NVIDIA Titan V100, CUDA 10.0 and cuDNN v7. Compared with the FPN, our AutoDet achieves a comparable inference speed (25.4 FPS and 18.8 FPS with ResNet-50 and ResNet-101 backbones) in the two-stage architecture Faster RCNN. For the one-stage architecture SSD, we evaluate the FLOPs and parameters of different methods in Table 3. We find that AutoDet only increases slight FLOPs and parameters but achieves significant improvements under all experimental settings. Overall, AutoDet offers the better tradeoff of speed and accuracy, which demonstrates the effectiveness of the discovered feature pyramid.

4.3.4 Search Efficiency Analysis

To analyze the search efficiency of NAS, we compare our SA-NAS with RL-based NAS (Liu et al. 2018a) as a standard reference. For a fair comparison, we keep the same experimental setup (including search space, and training a child network, etc.) except for the search algorithm. As shown in Fig. 4, the x axis denotes the iteration number, and the y axis denotes the reward normalized by the same factor. The left and right images correspond to the reward increasing alongside iterations of SA-NAS and RL-based NAS, respectively. Note that we set different scopes of the x axis for a better view. To evaluate the stability of SA-NAS, we conduct multiple independent search processes that are visualized with different colors in Fig. 4. Obviously, SA-NAS tends to oscillate in the first 100 iterations, but it has a rapid and stable convergence within only 200 iterations. However, the RL-based NAS still has huge fluctuations even in 6, 000 iterations. The underlying reason may be low data efficiency in RL. For the one-stage architecture, the SA-NAS requires 16 minutes to get the reward for a child ResNet-50 network on a single NVIDIA P40 GPU. Thus, the SA-NAS only requires 2.2 GPU days to converge, which is at least 30× faster than RL-based methods. The time consumption will vary according to the complexity of the whole networks. Such an observation is in accordance with findings in other NAS works (Liu et al. 2018b; Zhang et al. 2019).

Table 4 Results of one-stage methods on the COCO `test-dev` benchmark of **single model** with the input size of 320 × 320

Method	Backbone	AP	$AP_{@0.5}$	$AP_{@0.75}$	AP_S	AP_M	AP_L	AP_1	AP_{10}	AP_{100}	AP_S	AP_M	AP_L
YOLOv2(Redmon and Farhadi 2017)	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
SSD300* (Liu et al. 2016)	VGG-16	25.1	43.1	25.8	6.6	25.9	41.4	23.7	35.1	37.2	11.2	40.4	58.4
RON384++ (Kong et al. 2017)	VGG-16	27.4	49.5	27.1	–	–	–	–	–	–	–	–	–
DSSD321 (Fu et al. 2017)	ResNet-101	28.0	46.1	29.2	7.4	28.1	47.6	25.5	37.1	39.4	12.7	42.0	62.6
DSOD300 (Shen et al. 2017)	DS	29.3	47.3	30.6	9.4	31.5	47.0	27.3	40.7	43.0	16.7	47.1	65.0
DES300 (Zhang et al. 2018a)	VGG-16	28.3	47.3	29.4	8.5	29.9	45.2	25.6	38.3	40.7	14.1	44.7	62.0
RetinaNet400 (Lin et al. 2017b)	ResNet-101	31.9	49.5	34.1	11.6	35.8	48.5	–	–	–	–	–	–
RefineDet320 (Zhang et al. 2018a)	VGG-16	29.4	49.2	31.3	10.0	32.0	44.4	–	–	–	–	–	–
RefineDet320 (Zhang et al. 2018a)	ResNet-101	32.0	51.4	34.2	10.5	34.7	50.4	–	–	–	–	–	–
RFBNet300 (Liu et al. 2018c)	VGG-16	30.3	49.3	31.8	11.8	31.9	45.9	–	–	–	–	–	–
EFIP300 (Pang et al. 2019)	VGG-16	30.0	48.8	31.7	10.9	32.8	46.3	–	–	–	–	–	–
M2Det320 (Zhao et al. 2019)	VGG-16	33.5	52.4	35.6	14.4	37.6	47.6	–	–	–	–	–	–
M2Det320 (Zhao et al. 2019)	ResNet-101	34.3	53.5	36.5	14.8	38.8	47.9	–	–	–	–	–	–
DAFS320 (Li et al. 2019)	VGG-16	31.2	50.8	33.4	10.8	34.0	47.1	–	–	–	–	–	–
DAFS320 (Li et al. 2019)	ResNet-101	33.2	52.7	35.7	10.9	35.1	52.0	–	–	–	–	–	–
SSD320 + **AutoDet**	ResNet-101	**34.5**	**53.2**	**37.0**	**15.6**	**35.6**	50.6	30.4	47.0	49.3	26.2	48.4	66.7

Bold values indicate the best performance which model achieves

Table 5 Results of one-stage methods on the COCO `test-dev` benchmark of **single model** with the input size of 512 × 512

Method	Backbone	AP	$AP_{@0.5}$	$AP_{@0.75}$	AP_S	AP_M	AP_L	AP_1	AP_{10}	AP_{100}	AP_S	AP_M	AP_L
SSD512* (Liu et al. 2016)	VGG-16	28.8	48.5	30.3	10.9	31.8	43.5	26.1	39.5	42.0	16.5	46.6	60.8
SSD513 (Fu et al. 2017)	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8	28.3	42.1	44.4	17.6	49.2	65.8
YOLOv2 (Redmon and Farhadi 2017)	DarkNet	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
YOLOv3 (Redmon and Farhadi 2018)	DarkNet-53	33.0	57.9	34.4	18.3	35.4	41.9	–	–	–	–	–	–
DSSD513 (Fu et al. 2017)	ResNet-101	33.2	53.3	35.2	13.0	35.4	51.1	28.9	43.5	44.6	21.8	49.1	66.4
DES512 (Zhang et al. 2018b)	VGG-16	32.8	53.2	34.6	13.9	36.0	47.6	28.4	43.5	46.2	21.6	50.7	64.6
RetinaNet500 (Lin et al. 2017b)	ResNet-101	34.4	53.1	36.8	14.7	38.5	49.1	–	–	–	–	–	–
RefineDet512 (Zhang et al. 2018a)	VGG-16	33.0	54.5	35.5	16.3	36.3	44.3	–	–	–	–	–	–
RefineDet512 (Zhang et al. 2018a)	ResNet-101	36.4	57.5	39.5	16.6	39.9	51.4	–	–	–	–	–	–
RFBNet512 (Liu et al. 2018c)	VGG-16	33.8	54.2	35.9	16.2	37.1	47.4	–	–	–	–	–	–
RFBNet-E512 (Liu et al. 2018c)	VGG-16	34.4	55.7	36.4	17.6	37.0	47.6	–	–	–	–	–	–
EFIP512 (Pang et al. 2019)	VGG-16	34.6	55.8	36.8	18.3	38.2	47.1	–	–	–	–	–	–
M2Det (Zhao et al. 2019)	VGG-16	37.6	56.6	40.5	18.4	43.4	51.2	–	–	–	–	–	–
M2Det (Zhao et al. 2019)	ResNet-101	38.8	59.4	41.7	20.5	43.9	53.4	–	–	–	–	–	–
SSD512 + **AutoDet**	ResNet-101	**39.7**	**59.8**	**43.0**	**21.5**	41.8	53.0	32.6	48.9	50.5	33.9	55.4	68.0

Bold values indicate the best performance which model achieves

Table 6 Comparison with other NAS methods for object detection on the COCO test-dev benchmark

Method	Backbone	Search cost (GPU-day)	AP
DetNAS-FPN-Faster	–	44	40.2
DetNAS-RetinaNet	–	44	33.3
NAS-FPN @256	ResNet-50	333×#TPUs	< 38.0
NAS-FPN 7@256	ResNet-50	333×#TPUs	44.8
AutoDet-SSD	ResNet-50	2.2	36.2
AutoDet-Faster	ResNet-50	2.2	40.2

Note that the AP of NAS-FPN @256 here are from Figure 11 in NAS-FPN (Ghiasi et al. 2019), and NAS-FPN 7@256 stacks the searched FPN structure 7 times

4.4 Comparisons with Other NAS Methods

AutoDet automates the design of the feature pyramid network, which involves not only the performance of the searched model but also its search efficiency. In order to evaluate these two sides, we compare AutoDet with other methods using automatic search on both search cost and AP. The compared methods include NAS-FPN (Ghiasi et al. 2019) and DetNAS (Chen et al. 2019). The NAS-FPN is the first to report the success of applying Neural Architecture Search for pyramidal architecture in object detection. Our AutoDet and NAS-FPN propose different ideas to achieve an identical target. The pioneering task of DetNAS aims at searching for backbones in object detectors. Table 6 shows the results of different NAS methods on the COCO dataset. DetNAS spends 44 GPU-days searching for a backbone and achieves 33.3 and 40.2 AP on one-stage and two-stage frameworks. Taking the ResNet-50 as the backbone, yet NAS-FPN requires 333×#TPUs. The heavy NAS-FPN 7@256 gains a high AP 44.8 because of stacking the searched FPN structure 7 times. It is obvious that our AutoDet achieves the best tradeoff between search cost and performance. For SSD and Faster RCNN, we only require 2.2 GPU-days to search the better FPN and perform a competitive AP.

4.5 Comparisons with State-of-the-Art Detectors

We further compare the results of AutoDet with other state-of-the-art models of both one-stage and two-stage approaches with different feature fusion methods on COCO test-dev. For the one-stage approach, our model is trained and tested on single-scale images. For the two-stage approach, we add an extra experiment on multi-scale training and testing for abundant comparisons.

One-stage detectors are usually evaluated under different input sizes, i.e., 320 × 320 and 512 × 512. AutoDet searches feature pyramids based on ResNet-101 for each input size. Table 4 shows the comparison of different methods with the input size of 320 × 320. Due to the independent feature representation as detection layers, the vanilla SSD only achieves the AP score of 25.1%, which is significantly worse than other detectors with the feature fusing strategy. RON384++ (Kong et al. 2017), DSSD321 (Fu et al. 2017) and RetinaNet400 (Lin et al. 2017b) discover that shallow layers lack high-level semantic information in SSD. Thus, they have designed several strategies of feature fusing and yielded great AP improvement at different object scales. Particularly, the AP of RetinaNet400 (Lin et al. 2017b) with FPN has surpassed 30%. Hence it is necessary to design a suitable feature pyramid network for object detection. Experimental results clearly show that our AutoDet yields superior overall performance compared with other object detectors. It is worth pointing out that AutoDet remarkably outperforms state-of-the-art methods under almost all setups, such as various scales and IoU thresholds. This is because AutoDet automatically searches for the most suitable feature pyramid network. Table 5 shows the performances of different object detectors with an input size of 512 × 512. We observe that all methods perform better than those with the low-resolution input. However, the performances of all methods are linear to performances with the input size of 320 × 320. As expected, AutoDet achieves the highest performance in terms of AP, $AP_{@}0.5$ and $AP_{@}0.75$. Note that the advantage of AutoDet is particularly apparent in detecting small objects. All of these results suggest that the searched feature pyramids by AutoDet are effective for object detection (Table 6).

We evaluate the proposed methods on two-stage detectors based on ResNet-101. To verify the performance of AutoDet, we compare our method with state-of-the-art object detection methods, including CoupleNet (Zhu et al. 2017), Faster R-CNN (Ren et al. 2015) with FPN, Cascade R-CNN (Cai and Vasconcelos 2018), Mask R-CNN (He et al. 2017), D-RFCN (Dai et al. 2017), SNIP (Singh and Davis 2018) and SNIPER (Singh et al. 2018). Some methods adopt extra tricks to further boost the performance, like soft-nms (Bodla et al. 2017), deformable convolution (Dai et al. 2017), multi-scale training and testing, etc. Hence, based on Cascade R-CNN (Cai and Vasconcelos 2018), we report two versions of AutoDet with different setups. The results for two-stage detectors on COCO are presented in Table 7. The performances of all methods increase dramatically compared with one-stage detectors. CoupleNet (Zhu et al. 2017) obtains the lowest

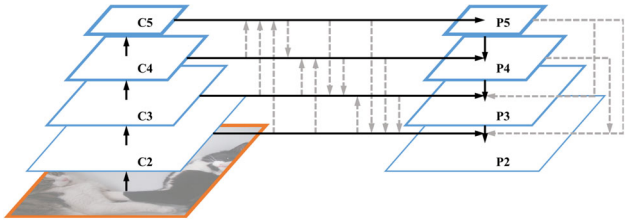
Table 7 Comparison with two-stage results on the COCO `test-dev` benchmark with a **single model**

Method	Backbone	AP	$AP_{@0.5}$	$AP_{@0.75}$	AP_S	AP_M	AP_L	AP_1	AP_{10}	AP_{100}	AP_S	AP_M	AP_L
CoupleNet (Zhu et al. 2017)	ResNet-101	34.4	54.8	37.2	13.4	38.1	52.0	30.0	45.0	46.4	20.7	53.1	68.5
Faster R-CNN (Ren et al. 2015) + FPN (Lin et al. 2017a)	ResNet-101	36.2	59.1	39.0	18.2	39.0	48.2	–	–	–	–	–	–
Faster R-CNN (Ren et al. 2015) + G-RMI (Huang et al. 2017)	Inception-ResNet-v2 (Szegedy et al. 2017)	34.7	55.5	36.7	13.5	38.1	52.0	–	–	–	–	–	–
Faster R-CNN (Ren et al. 2015) + TDM (Shrivastava et al. 2016)	Inception-ResNet-v2	36.8	57.7	39.2	16.2	39.8	52.1	31.6	49.3	51.9	28.1	56.6	71.1
Cascade R-CNN (Cai and Vasconcelos 2018)	ResNet-101-FPN	42.8	62.1	46.3	23.7	45.5	55.2	–	–	–	–	–	–
Mask R-CNN (He et al. 2017)	ResNet-101	38.2	60.3	41.7	20.1	41.1	50.2	–	–	–	–	–	–
Mask R-CNN (He et al. 2017)	ResNeXt-101(seg)	39.8	62.3	43.4	22.1	43.2	51.2	–	–	–	–	–	–
D-RFCN (Dai et al. 2017) + Soft-nms (Bodla et al. 2017)	ResNet-101	38.4	60.1	41.6	18.5	41.6	52.5	–	50.5	53.8	–	–	–
D-RFCN (Dai et al. 2017) + Soft-nms (Bodla et al. 2017)	ResNet-101(6 scales)	40.9	62.8	45.0	23.3	43.6	53.3	–	54.7	60.4	–	–	–
D-RFCN (Dai et al. 2017) + SNIP (Singh and Davis 2018)	ResNet-101(3 scales)	43.4	65.5	48.4	27.2	46.5	54.9	–	–	–	–	–	–
Faster R-CNN (Ren et al. 2015) + SNIP (Singh and Davis 2018)	ResNet-101(3 scales+deformable)	44.4	66.2	49.9	27.3	47.4	56.9	–	–	–	–	–	–
Faster R-CNN (Ren et al. 2015) + SNIPER(Singh et al. 2018)	ResNet-101(3 scales+deformable)	46.1	67.0	51.6	29.6	48.9	58.1	–	–	–	–	–	–
Cascade R-CNN + **AutoDet**	ResNet-101	46.3	67.5	50.2	26.7	50.7	59.7	36.4	58.3	64.9	40.3	67.4	79.2
Cascade R-CNN + **AutoDet**	ResNet-101(3 scales)	**47.3**	**68.7**	51.1	27.4	**51.4**	**61.7**	36.7	60.6	65.4	41.2	69.6	80.4

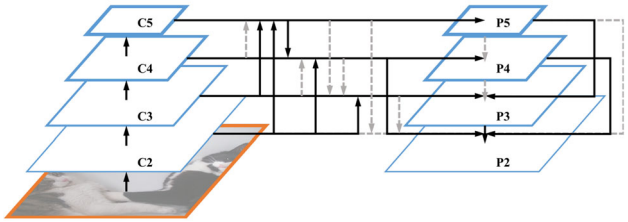
Bold values indicate the best performance which model achieves

We point out the specific method used for the feature map fusion and extra tricks behind each model. (seg) denotes that segmentation mask is also used

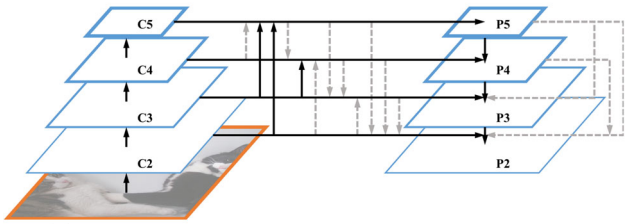
Fig. 5 Qualitative examples on MS COCO



(a) The topology of FPN.



(b) The searched topology on two-stage architecture.



(c) The searched topology on one-stage architecture.

Fig. 6 The topology of FPN, searched pyramid network architecture with Faster RCNN and SSD in the AutoDet search space. The solid and dash lines denote the connection and non-connection respectively

AP. Faster R-CNN (Ren et al. 2015) with FPN can be further improved by the cascade mechanism in Cascade R-CNN (Cai and Vasconcelos 2018) and multi-task learning in Mask R-CNN (He et al. 2017). It is interesting to observe that deformable convolution (Dai et al. 2017) as well as multi-scale training and testing (Singh and Davis 2018; Singh et al. 2018) promote large improvement. For example, SNIPER (Singh et al. 2018) with deformable convolution improves the AP scores from 36.2% to 46.1%. It's noteworthy that our naive AutoDet without multi-scale strategies and deformable convolution already outperforms all of other state-of-the-art methods. When conducting multi-scale training and testing, our AutoDet is further enhanced and achieves AP scores of 47.3%, which is the highest performance. Figure 5 shows qualitative examples of MS COCO. These results further validate the effectiveness of searched feature pyramids by AutoDet.

4.6 Visualization and Analysis of the Discovered Feature Pyramids

In this section, we analyze the discovered feature pyramid and summarize some inspiring findings, which may be helpful for architecture design. Figure 6 illustrates the topology of FPN and the searched feature pyramids architecture on both one-stage and two-stage detectors. We make the following inspiring findings:

- All feature maps have previous nodes up-sampled by the factors of 2 and 4, but none have previous nodes with 8 up-sampled operations. It means that the up-sampling factor of 8 will lose too much information or introduce some noise.
- For the two-stage architecture, C_2 is the previous node of all other scales $\{P_3, P_4, P_5\}$ except for its own scale P_2. This highlights the importance of the high-resolution P_2.
- We can gain benefit from a previous node down-sampled by a factor of 8. Thus, down-sample operation may preserve some original information.
- If the previous node has the same resolution, more channels of operations can bring a better reward.
- If the previous node is down-sampled, the reward first increases then decreases with increasing the number of filters.
- If the previous node is up-sampled, the reward constantly decreases with increasing the number of filters.

The first three findings are about topology and the last three findings are about the channel number of the CNN operation. The increasing or decreasing channel number is relative to 256, which is used in the FPN (Lin et al. 2017a).

4.7 Searching for On-Device Real-Time Image Recognition Models

The previous sections have illustrated that our SA-NAS can search a good feature pyramid for object detection based on heavy backbones, like ResNet-50 and ResNet-101, which pays more attention to the high performance rather than lightweight models. Recently, many embedded and mobile devices become widely available, such as smartphones, and wearable devices, etc. It is desirable to recognize objects of images using battery-powered systems where energy is limited and real-time inference is so important. It is urgent to search for an on-device real-time image recognition model. Fortunately, our SA-NAS is a general neural architecture search framework and decoupled with tasks.

In this subsection, we evaluate the performance of SA-NAS on real-time image detection and classification scenarios. [2]

For the real-time object detection, we follow MobileNetV3 (Howard et al. 2019) and search for the high-efficiency backbone of ssdlite, because most of the computations are spent on the backbone and the neck part (i.e., FPN) is usually removed for fast inference. Our search space is based on MobileNetV3-large, and search the kernel size $k \in \{3, 5, 7\}$,

[2] https://lpcv.ai/2020CVPR/ovic-track.

Table 8 Comparison on the real-time image classification and detection

Method	Task	Latency (ms)	Performance	Metric
MobileNetV3	Classification	12.0	75.2	− 0.26902
SA-NAS-A	Classification	11.5	**75.8**	− 0.24181
SA-NAS-B	Classification	**9.5**	74.3	**− 0.16158**
MobileNetV3+ssdlite	Detection	31.0	22.0	− 0.01594
SA-NAS-A+ssdlite	Detection	30.0	**22.2**	− 0.00840
SA-NAS-B+ssdlite	Detection	**25.8**	21.7	**0.01208**

Bold values indicate the best performance which model achieves

expansion rate $n \in \{3, 6\}$ of each inverted bottleneck block and whether the squeezing and excitation mechanism is enabled or not. In the search stage, backbone in the search space is encoded into a vector var, and then we use SA-NAS to iteratively optimize the better architecture. Following the Low-Power Image Recognition Challenge (LPIRC), [3] we evaluate the real-time object detector on COCO and operate image on a Pixel 4 smartphone (CPU). The performance is evaluated by AP. For the real-time image classification, we focus on ImageNet classification models and use the top-1 accuracy for evaluation. The latency is usually adopted to evaluate the inference speed. There does not exist a common metric for comparing different methods in terms of energy efficiency and accuracy in recognition. To test the overall performance in terms of accuracy and latency, we introduce the metric from LPIRC. The real-time scenarios utilize $Metric = A - k \log(latency) + a_0$ to punish the performance A where the parameters are $k = 49.84607103726407$ and $a_0 = -34.42191514521174$. $k = 16.894553358968146$ and $a_0 = -21.759878323711725$ for object detection. To verify the performance of SA-NAS, we compare our method with the state-of-the-art image recognition method MobileNetV3.

The first panel of Table 8 shows the latency, top-1 accuracy and metric of different image classification methods. Compared with MobileNetV3, the searched SA-NAS-A achieves the higher accuracy and lower latency, which not only reduces latency from 12 ms to 11.5 ms, but also improves the top-1 accuracy from 75.2% to 75.8%. Therefore, SA-NAS-A is also superior to MobileNetV3 from the overall metric. We also search for a more efficient model, SA-NAS-B. Though its performance is slightly inferior to MobileNetV3, the latency is only 9.5 ms and much lower than its competitor. Experimental results in terms of object detection clearly show that our SA-NAS-A+ssdlite yields superior performance in both latency (30 ms) and average precision (22.2%). Moreover, our searched SA-NAS-B+ssdlite achieves the highest overall metric of 20.48 whose latency is only 25.8 ms. These results validate the effectiveness of SA-NAS in searching on-device real-time image detection and classification models.

[3] https://rebootingcomputing.ieee.org/lpirc.

5 Conclusions

In this paper, we move a step forward to automatically search a feature pyramid network for object detection directly on the challenging dataset COCO. To obtain a network with high performance at a low cost, we design a specific combinatorial search space and employ a Simulated Annealing-based Network Architecture Search (SA-NAS) to significantly reduce the convergence time. Experiments on COCO demonstrate that our AutoDet can outperform other state-of-the-art one-stage and two-stage approaches with the same settings and criteria. The efficiency of SA-NAS is more than 30x higher than that of RL NAS and consumes fewer GPU days than other methods on different tasks.

Acknowledgements This work is partially funded by Beijing Natural Science Foundation (Grant No. JQ18017), National Natural Science Foundation of China (Grant No. U20A20223), and Youth Innovation Promotion Association CAS (Grant No. Y201929).

References

Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J., & Ogden, J. M. (1984). Pyramid methods in image processing. *RCA Engineer*, *29*(6), 33–41.

Baker, B., Gupta, O., Naik, N., & Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *ICLR*.

Bell, S., Lawrence Zitnick, C., Bala, K., & Girshick, R. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*.

Bender, G., Kindermans, P. J., Zoph, B., Vasudevan, V., & Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *ICML*.

Bodla, N., Singh, B., Chellappa, R., & Davis, L. S. (2017). Soft-nms-improving object detection with one line of code. *ICCV*, 5561–5569.

Brock, A., Lim, T., Ritchie, J. M., & Weston, N. J. (2018). Smash: One-shot model architecture search through hypernetworks. In *ICLR*.

Cai, Z., & Vasconcelos, N. (2018). Cascade r-cnn: Delving into high quality object detection. *CVPR*, 6154–6162.

Cai, Z., Fan, Q., Feris, R. S., & Vasconcelos, N. (2016). A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*.

Cai, H., Yang, J., Zhang, W., Han, S., & Yu, Y. (2018). Path-level network transformation for efficient architecture search. *ICML*, 677–686.

Chen, L. C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., et al. (2018). Searching for efficient multi-scale architectures for dense image prediction. *NIPS*, 8713–8724.

Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., & Sun, J. (2019). Detnas: Backbone search for object detection. *NIPS*, 6638–6648.

Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W., & Chua, T. S. (2016). Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. arXiv preprint arXiv:1611.05594.

Chopard, B., & Tomassini, M. (2018). Simulated annealing. In *An introduction to metaheuristics for optimization* (pp. 59–79). Springer.

Dai, J., Li, Y., He, K., & Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., et al. (2017). Deformable convolutional networks. *CVPR*, 764–773.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR* (pp. 248–255).

Dollár, P., Appel, R., Belongie, S., & Perona, P. (2014). Fast feature pyramids for object detection. *TPAMI*, *36*(8), 1532–1545.

Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. *CVPR*, 1761–1770.

Elsken, T., Metzen, J. H., & Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. In *ICLR*.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *JMLR*, *20*(55), 1–21.

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *IJCV*, *111*(1), 98–136.

Fu, C. Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C. (2017). Dssd: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659.

Ghiasi, G., Lin, T. Y., & Le, Q. V. (2019). Nas-fpn: Learning scalable feature pyramid architecture for object detection. *CVPR*, 7036–7045.

Girshick, R. (2015). Fast r-cnn. *ICCv*, 1440–1448.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *ICCV*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for mobilenetv3. In *ICCV*.

Hu, Y., Wu, X., & He, R. (2020). Tf-nas: Rethinking three search freedoms of latency-constrained differentiable neural architecture search. In *ECCV*.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*.

Jenatton, R., Archambeau, C., González, J., & Seeger, M. (2017). Bayesian optimization with tree-structured dependencies. *ICML*, 1655–1664.

Jiang, N., Krishnamurthy, A., Agarwal, A., Langford, J., & Schapire, R. E. (2017). Contextual decision processes with low bellman rank are pac-learnable. In *ICML* (pp. 1704–1713). JMLR. org.

Kong, T., Sun, F., Yao, A., Liu, H., Lu, M., & Chen, Y. (2017). Ron: Reverse connection with objectness prior networks for object detection. *CVPR*, 5936–5944.

Kong, T., Yao, A., Chen, Y., & Sun, F. (2016). Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Law, H., & Deng, J. (2019). Cornernet: Detecting objects as paired keypoints. In *IJCV*.

Li, Z., & Zhou, F. (2017). Fssd: Feature fusion single shot multibox detector. arXiv preprint arXiv:1712.00960.

Li, Z., Xi, T., Deng, J., Zhang, G., Wen, S., & He, R. (2020). Gp-nas: Gaussian process based neural architecture search. In *CVPR*.

Li, S., Yang, L., Huang, J., Hua, X. S., & Zhang, L. (2019). Dynamic anchor feature selection for single-shot object detection. *ICCV*, 6609–6618.

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. *CVPR*, 2117–2125.

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *ICCV*, 2980–2988.

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2018). Focal loss for dense object detection. In *TPAMI*.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *ECCV*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *ECCV*.

Liu, C., Chen, L. C., Schroff, F., Adam, H., Hua, W., Yuille, A., & Fei-Fei, L. (2019). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. arXiv preprint arXiv:1901.02985.

Liu, S., Huang, D., et al. (2018). Receptive field block net for accurate and fast object detection. *ECCV*, 385–400.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M. (2019). Deep learning for generic object detection: A survey. In *IJCV*.

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path aggregation network for instance segmentation. In *CVPR*.

Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., et al. (2018). Progressive neural architecture search. *ECCV*, 19–34.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *CVPR*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, *60*(2), 91–110.

Luo, R., Tian, F., Qin, T., Chen, E., & Liu, T. Y. (2018). Neural architecture optimization. *NIPS*, 7816–7827.

Newell, A., Yang, K., & Deng, J. (2016). Stacked hourglass networks for human pose estimation. In *ECCV*.

Pang, Y., Wang, T., Anwer, R. M., Khan, F. S., & Shao, L. (2019). Efficient featurized image pyramid network for single shot detector. *CVPR*, 7336–7344.

Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., et al. (2018). Megdet: A large mini-batch object detector. *CVPR*, 6181–6189.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268.

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

Redmon, J., & Farhadi, A. (2017). Yolo9000: Better, faster, stronger. *CVPR*, 7263–7271.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.
Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.
Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.
Shen, Z., Liu, Z., Li, J., Jiang, Y.G., Chen, Y., & Xue, X. (2017). Dsod: Learning deeply supervised object detectors from scratch. In *ICCV*.
Shrivastava, A., Sukthankar, R., Malik, J., & Gupta, A. (2016). Beyond skip connections: Top-down modulation for object detection. arXiv preprint arXiv:1612.06851.
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
Singh, B., & Davis, L. S. (2018). An analysis of scale invariance in object detection snip. *CVPR*, 3578–3587.
Singh, B., Najibi, M., & Davis, L. S. (2018). Sniper: Efficient multi-scale training. *NIPS*, 9333–9343.
Suganuma, M., Ozay, M., & Okatani, T. (2018). Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. *ICML*, 4778–4787.
Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*.
Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 6105–6114.
Tan, M., Chen, B., Pang, R., Vasudevan, V., & Le, Q. V. (2018). Mnasnet: Platform-aware neural architecture search for mobile. arXiv preprint arXiv:1807.11626.
Tan, M., Pang, R., V. & Le, Q. (2020). Efficientdet: Scalable and efficient object detection. In *CVPR*.
Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., & Tang, X. (2017). Residual attention network for image classification. arXiv preprint arXiv:1704.06904.
Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., et al. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CVPR*, 10734–10742.
Xie, L., & Yuille, A. (2017). Genetic cnn. *ICCV*, 1379–1388.
Xie, S., Zheng, H., Liu, C., & Lin, L. (2018). Snas: Stochastic neural architecture search.
Zhang, Z., Qiao, S., Xie, C., Shen, W., Wang, B., & Yuille, A. L. (2018). Single-shot object detection with enriched semantics. *CVPR*, 5813–5821.
Zhang, C., Ren, M., & Urtasun, R. (2019). Graph hypernetworks for neural architecture search. In *ICLR*.
Zhang, S., Wen, L., Bian, X., Lei, Z., & Li, S. Z. (2018). Single-shot refinement neural network for object detection. *CVPR*, 4203–4212.
Zhao, Q., Sheng, T., Wang, Y., Tang, Z., Chen, Y., Cai, L., & Ling, H. (2019). M2det: A single-shot object detector based on multi-level feature pyramid network. In *AAAI*.
Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., & Tian, Q. (2019). Multinomial distribution learning for effective neural architecture search. In *ICCV*.
Zhong, Z., Yan, J., Wu, W., Shao, J., & Liu, C. L. (2018). Practical block-wise neural network architecture generation. *CVPR*, 2423–2432.
Zhu, Y., Zhao, C., Wang, J., Zhao, X., Wu, Y., & Lu, H. (2017). Couplenet: Coupling global structure with local parts for object detection. *ICCV*, 4126–4134.
Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.
Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *CVPR*, 8697–8710.