EAutoDet: Efficient Architecture Search for Object Detection

Xiaoxing Wang[1], Jiale Lin[1], Juanping Zhao[2],
Xiaokang Yang[1], and Junchi Yan[1(✉)]

[1] Department of Computer Science and Engineering & MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University
[2] Guangdong OPPO Mobile Telecommunications Co., Ltd.
{figure1_wxx linjiale, xkyang, yanjunchi}@sjtu.edu.cn
zhaojuanping1325@oppo.com

Abstract. Training CNN for detection is time-consuming due to the large dataset and complex network modules, making it hard to search architectures on detection datasets directly, which usually requires vast search costs (usually tens and even hundreds of GPU-days). In contrast, this paper introduces an efficient framework, named EAutoDet, that can discover practical backbone and FPN architectures for object detection in 1.4 GPU-days. Specifically, we construct a supernet for both backbone and FPN modules and adopt the differentiable method. To reduce the GPU memory requirement and computational cost, we propose a kernel reusing technique by sharing the weights of candidate operations on one edge and consolidating them into one convolution. A dynamic channel refinement strategy is also introduced to search channel numbers. Extensive experiments show significant efficacy and efficiency of our method. In particular, the discovered architectures surpass state-of-the-art object detection NAS methods and achieve 40.1 mAP with 120 FPS and 49.2 mAP with 41.3 FPS on COCO test-dev set. We also transfer the discovered architectures to rotation detection task, which achieve 77.05 mAP_{50} on DOTA-v1.0 test set with 21.1M parameters. The code is publicly available at https://github.com/vicFigure/EAutoDet.

1 Introduction and Related Work

Handcrafted neural architectures that designed by experts with large amounts of trials and errors have achieved promising performance across computer vision tasks [13, 19, 44]. Automated architecture search methods have been recently explored, including reinforcement learning [45], evolutionary algorithm [25], Bayesian optimization [35], maximum flow in graph theory [38], as well as the more cost-effective one-shot NAS [1] that builds a supernet as the surrogate model to predict the performance of candidate architectures. DARTS [21] further introduces a differentiable method that reduces the search cost to a few GPU-days. However, DARTS requires vast GPU memory since it has to train an over-parameterized supernet, making it impossible to search on large datasets or complex tasks. Some works [34, 37, 5] are dedicated to reducing the memory requirement.

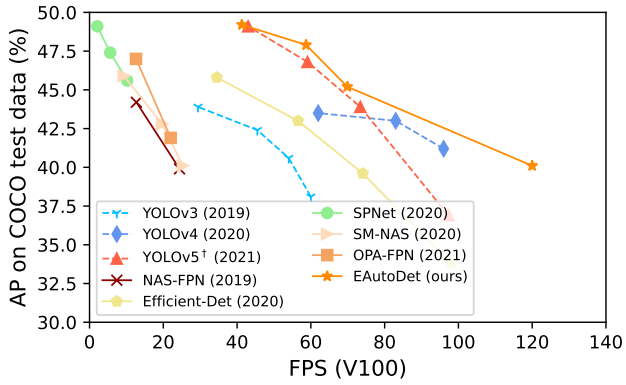


Fig. 1. Results of various detection models. Solid or dashed lines indicate NAS or handcrafted architectures. †: obtained by our experiments, otherwise from the references.

Though NAS has achieved great success on classification tasks, it is still an open question on how to directly search detection architectures for detection tasks with two major difficulties: **1) It is time-consuming to train detection models from scratch** due to its complex architecture, which consists of multiple modules, including backbone and feature pyramid network (FPN). So that many works [8, 2, 24] pre-train the backbone on ImageNet; **2) Training a detection model requires vast GPU memory cost**, especially for those NAS works [3, 10] that need to build an over-parameterized supernet. They even have to pre-train it on ImageNet, further increasing the search cost. To simplify the supernet and lower the search difficulty, they usually restrict the search space by either searching backbone [3, 6, 14] or FPN [7, 36, 33], which, however, actually ignores the relationship between the two modules. In contrast, this paper introduces kernel reusing technique and dynamic channel refinement to speed up the convergence to train a supernet and reduce GPU memory requirement. We thus propose an efficient search method, named EAutoDet, which can jointly search architectures of backbone and FPN on MS-COCO [20] detection dataset in a few GPU-days, with no need to pre-train a supernet on ImageNet.

Additionally, the prior NAS detection methods are based on RetinaNet (one-stage) or Faster-RCNN (two-stage) framework that adopts ResNet-like architectures. Few have explored to search for YOLO (one-stage) framework that could leverage its known fast speed and outstanding performance. The handcrafted YOLO models even outperform many NAS methods in similar inference speed (shown in Table 1), which implies the potential of a combination of NAS and YOLO. Nevertheless, a vanilla combination is undesirable. On the one hand, the handcrafted architecture of YOLO is subtle and elaborate. Such a nearly impeccable baseline puts forward higher requests for the ability of search method discovering the optimal architecture. On the other hand, it is better to absorb the knowledge of those well-designed architectures to design a sophisticated and large search space, which further asks for a flexible and efficient search method.

Table 1. Our method enables authentic fine-grained search w.r.t. operations, number of channels, and connections between layers, and is much faster than SM-NAS and Hit-Detector thanks to our kernel reusing and dynamic channel refinement techniques.

Search Space	**Method**
Backbone alone	DetNAS [3], SpineNet [6], SPNet [14]
FPN alone	NAS-FPN [7], NAS-FCOS [33], OPA-FPN [17], Auto-FPN [36]
Joint search	SM-NAS [42], Hit-Detector [10], EAutoDet (ours)

However, our method offers a practical solution thanks to its low memory requirement and rapid convergence rate to train a supernet. Besides, our technique ameliorates the computation of convolutions in the supernet rather than restricting sub-architectures of supernet, making our method flexible to suit various search spaces. Experiments show the outstanding performance of our method on MS-COCO and DOTA-v1.0. Our contributions are summarized as follows:

1) Efficient Architecture Search Method for Object Detection. We propose kernel reusing and dynamic channel refinement techniques for the fine-grained search of backbone and FPN modules in 1.4 GPU-days on a single V100 GPU, significantly outperforming prior NAS methods, e.g., 28 GPU-days [33] and 44 GPU-days [3]. Unlike other NAS methods [3, 42, 10] that have to pre-train an over-parameterized supernet on the ImageNet, our supernet is trained from scratch on MS-COCO thanks to the property of our method: low memory requirement and rapid convergence rate to train a supernet.

2) Sophisticated and Large Search Space for Object Detection. By absorbing the knowledge of well-designed YOLO models, we design a complex search space for detection, including convolution types, channel numbers, and connection of layers for backbone and FPN. It puts forward higher requests for the flexibility and efficiency of search methods. This paper offers a solution, and to the best of our knowledge, it is the first NAS method that outperforms YOLO models with competitive high inference speed. Notice that our method can be easily applied to other detectors, e.g., Faster-RCNN [28] and RetinaNet [19].

3) Strong Performance and Fast Speed. Our design allows for direct search and evaluation without pre-training. The discovered architectures achieve outstanding performance on classic horizontal, as well as rotation detection tasks: 40.1 mAP with 120 FPS on MS-COCO where the bounding box is always assumed horizontal, and 77.05 mAP_{50} on DOTA which is a dominant rotation detection benchmark but has not been used in NAS literature.

Below we briefly discuss the related works.

Object Detection. Existing detection frameworks usually consist of four modules: backbone, feature fusion neck, region proposal network (in two-stage detectors), and detection head. For real-time detection, [24, 27, 19, 2, 32] design efficient architectures for the four modules. There are also emerging manually-designed detectors for rotation detection whereby different loss functions are carefully devised ranging from regression [11, 41] to classification [39, 40] models.

Unlike the above works requiring massive trials and expert experience to design CNNs, we aim to search architectures for detection automatically.

Neural Architecture Search. Researchers have been dedicated to efficient search algorithms for neural architectures in recent years. NASNet [45] utilizes reinforcement learning (RL) and proposes to generate candidate architectures by an RNN controller. [25] adopt evolutionary algorithms (EA) to derive new architectures by crossover and mutation. Besides the above time-consuming methods, one-shot NAS [1] is introduced and can reduce the search cost to a few GPU-days. DARTS [21] regards NAS as a bi-level optimization problem and proposes to solve it by a differentiable method. This paper adopts the differentiable method in DARTS due to its efficacy and high efficiency.

NAS for Object Detection. Recent NAS methods for object detection can be briefly categorized into three streams: 1) Search backbone architecture and fix FPN, e.g. DetNAS [3] and SP-NAS [14]. 2) Search FPN architecture and fix backbone, e.g. NAS-FPN [7], Auto-FPN [36], and NAS-FCOS [33]. 3) Jointly search backbone and FPN, e.g., SM-NAS [42], Hit-Detector [10] and our method. Unlike SM-NAS and Hit-Detector that require vast GPUs to search, this work introduces kernel reusing and dynamic channel refinement techniques that can significantly reduce the GPU memory requirement during the search process. Specifically, our EAutoDet can search under a more extensive search space on MS-COCO dataset directly on a single V100 GPU in 1.4 days. Moreover, unlike many NAS detection methods [3, 30, 42, 10] that need to pre-train supernets on the ImageNet, our EAutoDet trains the supernet from scratch on the MS-COCO during the search process, demonstrating its outstanding convergence ability.

2 The Proposed EAutoDet

Referring to DARTS [21] that regards NAS as a bi-level optimization task, we also build a supernet and define architecture parameters $\boldsymbol{\alpha}$ to denote the importance of candidate operations. However, it is intractable to search on detection dataset directly by DARTS since the supernet is an over-parameterized model, making it much more challenging to train it from scratch on detection datasets, e.g., MS-COCO. We illustrate one super-edge in the supernet of DARTS in Fig. 2 (left), which contains all independent candidate operations and thus requires massive GPU memory during the search stage. To address the above memory explosion issue, we introduce two techniques to reduce memory requirement and computational cost: *Kernel Reusing Technique* to search operation types, and *Dynamic Channel Refinement* to search channel numbers.

2.1 Kernel Reusing for Operation Type Search

Each edge in the supernet contains multiple convolutions with various kernel sizes. Suppose $\boldsymbol{X}$ and $\boldsymbol{Z}$ are the input and output of convolutions on one edge, then $\boldsymbol{Z} = \sum_{o \in \mathcal{O}} \boldsymbol{\alpha}_o \boldsymbol{X} \otimes \boldsymbol{\theta}_o$, where '$\otimes$' denotes convolution operation, $\mathcal{O}$ is the candidate set of convolutions, and $\boldsymbol{\theta}$ is the kernel weights.

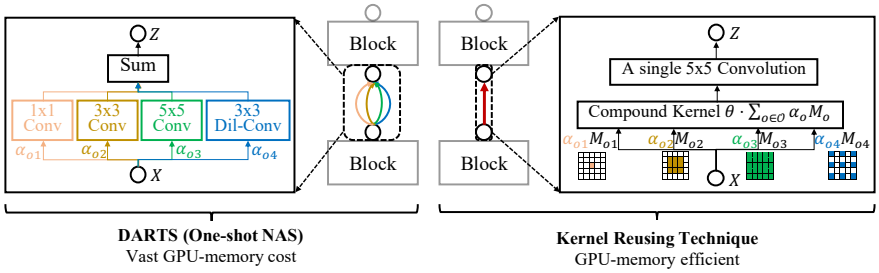


Fig. 2. Compared to DARTS, the kernel reusing technique compounds multiple convolutions into a single 5x5 convolution, which can reduce the memory cost and enables efficient search for backbone and FPN.

To reduce the parameters, we reuse the weights of different convolutions as shown in Fig. 2 (right), that is, kernels of all convolutions can be extracted from unified weights by a binary mask $\boldsymbol{M}$. Moreover, since convolutions are linear operations, the weighted sum of multiple convolutions on the same edge can be compounded into one convolution. Therefore, the output of each edge can be simplified as Eq. 1, where $\boldsymbol{\theta}$ is the unified weights, and kernels of candidate convolution o can be obtained by $\boldsymbol{M}_o \cdot \boldsymbol{\theta}$.

$$\boldsymbol{Z} = \underbrace{\sum_{o \in \mathcal{O}} \boldsymbol{\alpha}_o \boldsymbol{X} \otimes [\boldsymbol{M}_o \cdot \boldsymbol{\theta}]}_{|\mathcal{O}| \text{ convolutions}} = \underbrace{\boldsymbol{X} \otimes \left[\boldsymbol{\theta} \cdot \sum_{o \in \mathcal{O}} \boldsymbol{\alpha}_o \boldsymbol{M}_o\right]}_{\text{One convolution}} \tag{1}$$

Advantage of Our Kernel Reusing Technique. Apart from our kernel reusing technique, sampling-based methods [37, 5] are also popular to reduce the memory and computational cost for classification tasks. However, they involve a dynamic network structure by sampling a sub-network of the supernet at each iteration, which will affect the supernet training. Though such an issue can be tolerated on classification tasks, it worsens on detection tasks. In contrast, our kernel reusing technique holds a stable network structure and reduces GPU memory and computational cost by compounding multiple kernels into one without interfering with the supernet training.

We illustrate the mAP of supernets on MS-COCO validation set during the search stage in Fig. 3. EAutoDet-s/m/l/x denotes four supernets under various search spaces (details are introduced in Sec. 2.3). They are built based on our kernel reusing technique. EAutoDet-s-sample denotes that an s-level supernet is built without kernel reusing and trained by sampling operation based on Gumbel reparameterization technique at each iteration [5]. We observe that: **1)** Our four supernets can converge to 30% mAP; 2) Our kernel reusing technique converges better and faster than sampling-based method, which confirms the above analysis on better convergence property of our method. Notice that Fig. 3 shows mAP of supernets during 50 epochs' training in the search stage. The ultimate performance of the discovered models is evaluated by training them from scratch for

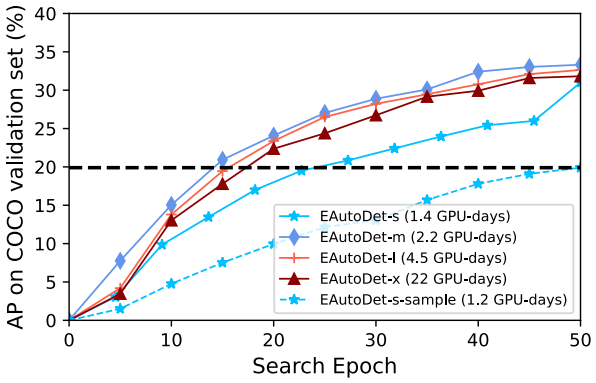


Fig. 3. MAP of supernets trained for 50 epochs. Search cost on a V100 GPU is given in the legend. EAutoDet-s-sample denotes to search by sampling-based method [5]. The horizontal dash line is the final AP of EAutoDet-s-sample.

300 epochs and is reported in Table 1. The search cost is also illustrated in the legend. Specifically, an effective s-level model can be discovered in 1.4 GPU-days on a single V100 GPU, showing the remarkable efficiency of our method.

Difference from the Prior Works. Unlike the prior NAS works [29, 34] that focus on classification tasks, our method aims to search detection models that are more complex and requires more computation resources. Our kernel reusing technique can significantly reduce the memory and computational cost, making it possible to discover an effective architecture in a few GPU-days. Apart from NAS works, RepVGG [4] is also related to our approach, which introduces a re-param strategy to merge skip-connection, 3×3 and 1×1 convolutions for a plain inference-time model. However, the motivation of RepVGG is to stabilize the training of VGG, and the merge process is applied after the training stage. In contrast, our kernel reusing technique is applied during the search process and aims to reduce the memory and computational cost.

2.2 Dynamic Channel Refinement for Channel Number Search

Layer channels are essential hyperparameters for neural network architectures affecting the model size and FLOPs. Unfortunately, few differentiable NAS works have explored to search channel numbers, especially for detection tasks. In this work, we introduce the dynamic channel refinement technique based on Gumbel reparameterization technique [9] to search for optimal expansion rates for layers. Specifically, we sample an expansion rate for each layer at every iteration and refine the operation weights $\boldsymbol{\theta}$ dynamically to fit the changeable channel numbers.

Sampling for Expansion Rate. Expansion rate for one layer can be sampled by Gumbel-argmax technique:

$$\boldsymbol{E} = \text{one\_hot}\left[\arg\max_i (\log \tilde{\boldsymbol{\alpha}}_e^i + \boldsymbol{g}^i)\right], \tag{2}$$

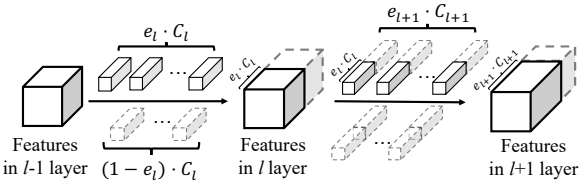


Fig. 4. Refinement for convolution weights $\boldsymbol{\theta}$ on adjacent layers in our dynamic channel refinement technique. e indicates the sampled expansion rate. Dotted blocks in light gray indicate the inactivated channels.

where $\tilde{\boldsymbol{\alpha}}_e = \text{softmax}(\boldsymbol{\alpha}_e)$ is the normalized weights for candidate expansion rates, and $\boldsymbol{g}^i$ are random variables sampled from Gumbel$(0, 1)$ distribution. To make $\boldsymbol{E}$ differentiable w.r.t. $\boldsymbol{\alpha}_e$, we adopt Gumbel-softmax to relax the sampled vector as follows, where τ is a gradually decayed temperature.

$$\tilde{\boldsymbol{E}} = \frac{\exp\left[(\log \tilde{\boldsymbol{\alpha}}_e^i + \boldsymbol{g}^i)/\tau\right]}{\sum_j \exp\left[(\log \tilde{\boldsymbol{\alpha}}_e^j + \boldsymbol{g}^j)/\tau\right]}, \tag{3}$$

We adopt the one-hot vector $\boldsymbol{E}$ (Eq. 2) to activate one candidate expansion rate during the forward pass and utilize the relaxed vector $\tilde{\boldsymbol{E}}$ (Eq. 3) to obtain gradients for $\boldsymbol{\alpha}_e$ during the back-propagation, which is a popular reparameterization technique that has been widely-use in many recent works [5, 31].

Dynamic Channel Refinement for Operation Weights. After sampling an expansion rate e_l for layer l with base channel number C_l, the output channel becomes $e_l C_l$. The operation weights on layer l can be refined by preserving the first $e_l C_l$ filters. Besides, it affects the input channel of convolution on layer $l+1$. Generally, sampling channels on one layer will affect the operation weights on the current and next layer. We, therefore, dynamically refine channels for weights of adjacent layers, as shown in Fig. 4. FBNet-V2 [31] is related to our method, which also utilizes the Gumbel technique to search for classification model architectures. However, FBNet-V2 has to pad zero on channels to obtain a unified dimension due to short-cut connections, resulting in useless computation. In contrast, we discard the short-cut connection and dynamically refine the channel numbers for each layer at every iteration. There is no useless computation resulting from padding zeros on channels.

Transformation for Concatenation Layers. The channel number for each layer alters dynamically during the search process, which brings difficulty to refine weights for convolutions after a concatenation layer. Specifically, suppose two features with expansion rates and base channels (e_1, C_1) and (e_2, C_2) are concatenated. A convolution is applied after the concatenation layer, then the activated input channels of weight are separated ($0 \sim e_1 C_1$ and $C_1 \sim C_1 + e_2 C_2$), making it hard to extract. To this end, we first give the following proposition, which is proved in the supplementary material.

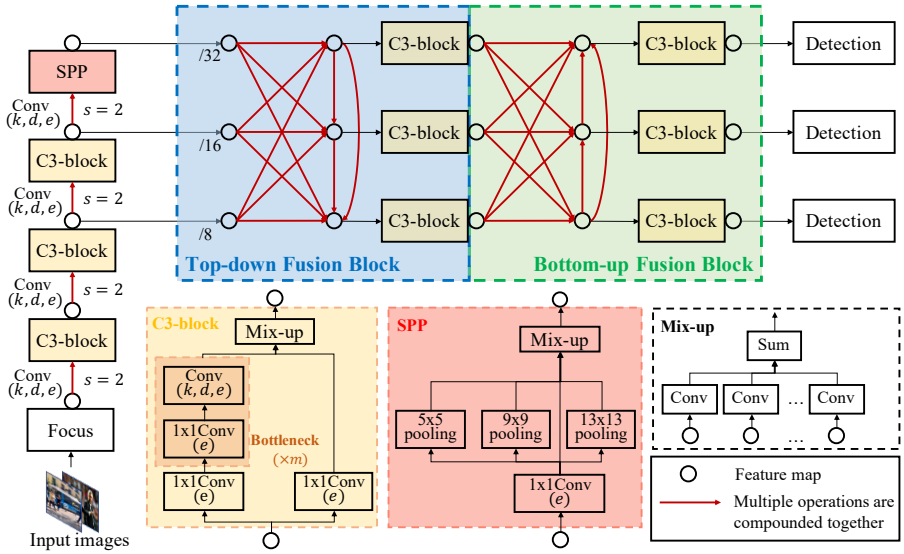


Fig. 5. The architecture of supernet, containing all candidate operations and connections in the search space. A red edge indicates candidate operations compounded by the kernel reusing technique, which is illustrated in Fig. 2. In backbone, C3-block, and SPP module, parentheses under 'Conv' indicate hyper-parameters to search: kernel size k, dilation ratio d, expansion rate of output channels e.

Proposition 1 *The output of a concatenation layer followed by a convolution layer is equivalent to the sum of separate convolutions on the inputs.*

2.3 Detection-oriented Search Space Design

Unlike ResNet-like and Mobile-like backbones in RetinaNet and Faster-RCNN that are transferred from classification tasks, YOLO models are specifically designed for detection tasks by experts considering both speed and performance. We would like to absorb the knowledge of the elaborate architectures and design a sophisticated and large detection-oriented search space. In particular, our method ameliorates the computation of convolutions in a supernet rather than restricting architectures of sub-models, making it flexible to suit such complex and large search spaces. Specifically, we resort to YOLOv5 [15] and PANet [22] and construct four types of supernets with various widths and depths, denoted as s (small), m (medium), l (large), and x (extra large), whose details are in the supplementary. In the following, we separately introduce the search spaces for backbone and FPN for their fundamentally different roles in the detection pipeline. The size of search space and details are given in the supplementary.

Search Space for Backbone. We propose to search operation types and channel numbers for down-sampling operators and structure of blocks. The supernet is shown in Fig. 5. **1)** Down-sampling operators have four candidates:

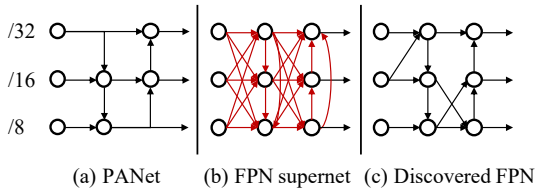


Fig. 6. Architectures of PANet, supernet of FPN and our discovered model. Nodes on three rows denote feature maps on three spatial sizes. Red edges indicate multiple operations are compounded on that edge.

{1x1 conv, 3x3 conv, 5x5 conv, 3x3 dilated conv} and three choices of expansion rate for output channel: {0.5, 0.75, 1.0}; **2)** Bottleneck cell consists of two convolutions with three choices of expansion rate: {0.5, 0.75, 1.0}, and the second convolution have three candidates: {3x3 conv, 5x5 conv, 3x3 dilated conv}; **3)** C3-block has two 1x1 convolutions with two choices of expansion rates: {0.75, 1.0}. Architectures of different layers are independently searched.

Search Space for Feature Pyramid Network. The supernet for top-down and bottom-up fusion blocks is shown in Fig. 5, which enables to search connections of features in three spatial scales, operation types and channel numbers of each connection, and the structure of C3-blocks. **1)** Nodes indicate feature maps and connect with all their predecessors in the supernet. After the search stage, only two predecessors will be selected for each node; **2)** Each red edge contains four candidate operations: {1x1 conv, 3x3 conv, 5x5 conv, 3x3 dilated conv} with three possible expansion rates for the output channel: {0.5,0.75,1.0}; **3)** C3-blocks, whose architectures are also searched, are concatenated at the end of each fusion block to independently extract multi-scale features, as shown in Fig. 5. For each fusion block, we introduce architecture parameters $\boldsymbol{\alpha}_e$ and $\boldsymbol{\alpha}_o$ to denote the importance of edges and operations. Suppose $\tilde{\boldsymbol{\alpha}} = \text{softmax}(\boldsymbol{\alpha})$ is the normalized weight. The fused feature $z_j = \sum_{i<j} \left[\tilde{\alpha}_e^{(i,j)} \cdot \sum_{o\in\mathcal{O}} \tilde{\alpha}_o^{(i,j)} \cdot o(x_i)\right]$, where $\mathcal{O}$ is the candidate operation set, x_i is the features of predecessors.

Deriving the Final Architecture. We utilize the magnitude of architecture parameters as the importance estimation for operations. The final backbone architecture is derived by preserving the best operation. While for the feature pyramid network, nodes in each fusion block preserve top-2 connections, and each connection will preserve the best operation. Fig. 6 compares the architecture of PANet, our supernet, and the discovered FPN module.

3 Experiments

We conduct experiments on two popular detection tasks: classic detection and rotation detection. The former is typical in many detection contests to locate common objects; The latter has been widely used in aerial images aiming to locate the ground object instances with an oriented bounding box (OBB). For

Table 2. Comparison with prior works on the COCO test-dev. FPS for YOLOv5 and our method are calculated on a single V100 GPU, and results for other methods are directly obtained from their papers. Different blocks indicate models with various inference speeds and prediction performance. '†': The results are obtained by our experiments. '-': The value is not provided by the original paper. '⋆': The unit of search cost is TPU-days, while the unit of other methods is GPU-days. '‡': SPNet[14] shows the search cost on VOC is 26 GPU-days, and is six times lower than that on COCO.

Method	FPS	#Params (M)	mAP (%)	AP_{50} (%)	AP_{75} (%)	AP_S (%)	AP_M (%)	AP_L (%)	Search Cost
YOLOv4 [2]	96	-	41.2	62.8	44.3	20.4	44.4	56.0	-
YOLOv5s† [15]	113	7.3	36.9	56.0	40.0	19.9	41.1	46.0	-
EfficientDet-D0 [30]	98	3.9	33.8	52.2	35.8	12.0	38.3	51.2	-
NAS-FPN [7]	24	60.3	39.9	-	-	-	-	-	333⋆
NAS-FCOS@128 [33]	-	27.8	37.9	-	-	-	-	-	28
SpineNet-49S [6]	-	11.9	39.5	59.3	43.1	20.9	42.2	54.3	-
SM-NAS:E2 [42]	25	-	40.0	58.2	43.4	21.1	42.4	51.7	187
EAutoDet-s (ours)	**120**	**9.1**	**40.1**	**58.7**	**43.5**	**21.7**	**43.8**	**50.5**	**1.4**
YOLOv3 + ASFF [23]	54	-	40.6	60.6	45.1	20.3	44.2	54.1	-
YOLOv4 [2]	83	-	43.0	64.9	46.5	24.3	46.1	55.2	-
YOLOv4-csp [32]	80†	43	46.2	64.8	50.2	24.6	50.4	61.9	-
YOLOv5m† [15]	88	21.4	43.9	62.5	47.6	25.1	48.1	54.9	-
EfficientDet-D1 [30]	74	6.6	39.6	58.6	42.3	17.9	44.3	56.0	-
DetNAS [3]	-	-	42.0	63.9	45.8	24.9	45.1	56.8	44
NAS-FPN [7]	13	60.3	44.2	-	-	-	-	-	333⋆
Auto-FPN [36]	-	32.6	40.5	61.5	43.8	25.6	44.9	51.0	16
NAS-FCOS@256 [33]	-	57.3	43.0	-	-	-	-	-	28
SpineNet-49 [6]	-	28.5	42.8	62.3	46.1	23.7	45.2	57.3	-
SM-NAS:E3 [42]	20	-	42.8	61.2	46.5	23.5	45.5	55.6	187
Hit-Detector [10]	-	27.1	41.4	62.4	45.9	25.2	45.0	54.1	-
OPA-FPN@64 [17]	22	29.5	41.9	-	-	-	-	-	4
EAutoDet-m (ours)	**70**	**28.1**	**45.2**	**63.5**	**49.1**	**25.7**	**49.1**	**57.3**	**2.2**
YOLOv3 + ASFF [23]	46	-	42.4	63.0	47.4	25.5	45.7	52.3	-
YOLOv4 [2]	62	-	43.5	65.7	47.3	26.7	46.7	53.3	-
YOLOv4-csp [32]	65†	53	47.5	66.2	51.7	28.2	51.2	59.8	-
EAutoDet-csp (ours)	**55**	**49.8**	**47.8**	**66.1**	**51.9**	**28.6**	**51.5**	**60.1**	**4.2**
YOLOv5l† [15]	59	47.1	46.8	65.4	50.9	27.7	51.0	58.5	-
EfficientDet-D2 [30]	57	8.1	43.0	62.3	46.2	22.5	47.0	58.4	-
SPNet(BNB) [14]	10	-	45.6	64.3	49.6	28.4	48.4	60.1	156‡
SM-NAS:E5 [42]	9	-	45.9	64.6	48.6	27.1	49.0	58.0	187
OPA-FPN@160 [17]	13	60.6	47.0	-	-	-	-	-	4
EAutoDet-l (ours)	**59**	**34.4**	**47.9**	**66.3**	**52.0**	**28.3**	**52.0**	**59.9**	**4.5**
YOLOv3 + ASFF [23]	29	-	43.9	64.1	49.2	27.0	46.6	53.4	-
YOLOv5x† [15]	43	87.8	49.1	67.5	53.6	30.2	53.4	61.4	-
EfficientDet-D3 [30]	35	12	45.8	65.0	49.3	26.6	49.4	59.8	-
SPNet(XB) [14]	6	-	47.4	65.7	51.9	29.6	51.0	60.4	156‡
EAutoDet-x (ours)	**41**	**86.0**	**49.2**	**67.5**	**53.6**	**30.4**	**53.4**	**61.5**	**22**

Table 3. Joint search VS. independent search for backbone and FPN on MS-COCO validation set. Results show the superiority of jointly search against independent search.

Architecture		**s-level (small)**		**m-level (medium)**	
Backbone	**FPN**	**mAP**$_{(\%)}$	$\Delta_{(\%)}$	**mAP**$_{(\%)}$	$\Delta_{(\%)}$
default	default	36.9	+0.0	44.0	+0.0
searched	default	37.4	+0.5	44.6	+0.6
default	*searched*	38.9	+2.0	45.0	+1.0
searched	***searched***	**40.1**	**+3.2**	**45.5**	**+1.5**

the classic detection task, we adopt MS-COCO 2017 benchmark with 80 common object categories. For the rotation detection task, we adopt DOTA-v1.0 benchmark, one of the largest aerial detection benchmarks.

Search Settings. The training set of MS-COCO is divided into two parts to train architecture parameters and network weights. The final architecture is derived after alternately optimizing architecture parameters and network weights for 50 epochs by an SGD optimizer.

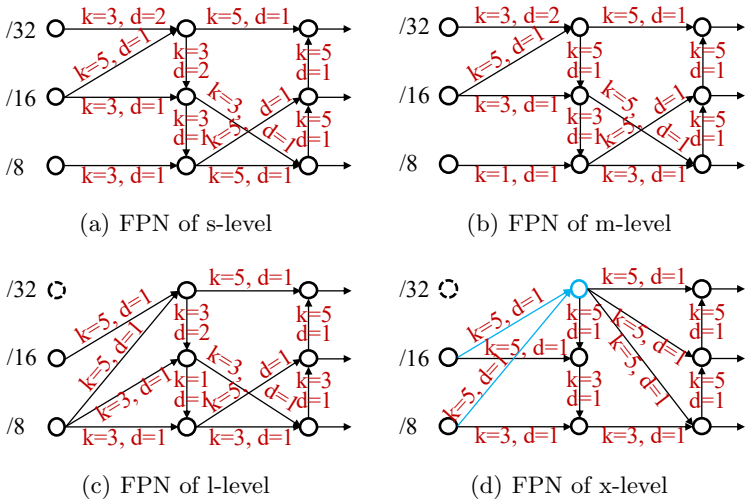
Evaluation Settings. Firstly, the discovered architectures are trained on MS-COCO training set from scratch for 300 epochs by an SGD optimizer and evaluated on its validation and test sets. We directly utilize the hyper-parameters in YOLOv5. To fairly compare the speed (FPS) with YOLO methods, we convert the trained models to the style of YOLOv4 [2] and evaluate the FPS on the Darknet platform [26], which is written in C and CUDA. Secondly, we train the architectures on rotation detection task on DOTA-v1.0 training set from scratch for 300 epochs and evaluate them on the validation and test sets. Notice that we train and test on a single input scale unlike previous works [11, 41] that adopt multi-scale training technique and random rotation augmentation.

3.1 Results on Classic Detection Benchmark: MS-COCO

Table 2 reports the performance of our methods and compares with other state-of-the-art works on the COCO test-dev dataset. Different blocks indicate models with various inference speeds and prediction performance. We observe that our discovered models (EAutoDet) achieve the best performance. Specifically, EAutoDet-s achieves 40.1 mAP with 120 FPS, outperforming EfficientDet-D0 by 6.3% mAP with similar inference speed. Compared to manually-designed detectors (YOLO series), our method has competitive and even better performance. EAutoDet-m achieves 45.2% mAP with 69.9 FPS, surpassing YOLOv4 by 2.2% mAP. Moreover, we compare to YOLOv4-csp [32] by inheriting its backbone CD-53 and searching FPN architecture ('EAutoDet-csp' in Table 2). The discovered architecture outperforms YOLOv4-csp by 0.3% mAP with fewer parameters.

3.2 Ablation Study of Backbone and FPN Search

We compare the performance of joint and independent search for backbone and FPN in Table 3, where 'default' indicates that we directly adopt the architecture



(a) FPN of s-level (b) FPN of m-level

(c) FPN of l-level (d) FPN of x-level

Fig. 7. FPN architecture of our searched models. 'k' denotes kernel size and 'd' denotes dilation ratio. Note that s-level and m-level have the same topology, while l-level and x-level discard the 32× down-sampled features (dashed node), which are the output of spatial pyramid pooling layer (SPP). In (d), we use blue lines and nodes to highlight the discarded SPP layer. We further analyze the effect of SPP by replacing one of the blue lines with the edge between dashed and blue nodes.

Table 4. Transferred models VS. directly searched models on the MS-COCO validation set. 'Tf-x' denotes that we transfer the discovered x-level (extra large) model to s/m/l-level. 'Tf-m' denotes that we transfer the discovered m-level (medium) model to s-level. 'Search' denotes the directly searched models.

Model	**s-level (small)**				**m-level (medium)**			**l-level (large)**		
	YOLOv5	Tf-x	Tf-m	Search	YOLOv5	Tf-x	Search	YOLOv5	Tf-x	Search
#Params(M)	7.3	8.0	8.0	9.1	21.4	21.8	28.1	47.1	48.6	42.2
mAP(%)	36.9	37.4	39.5	40.1	44.0	44.6	45.5	47.0	47.3	47.9

of YOLOv5, and 'searched' indicates that we search for the architectures. We observe that 1) Joint search achieves the best performance, showing the effectiveness of our algorithm and the necessity of joint search for detection models. 2) The search performance is much more sensitive to the architecture of FPN compared to the backbone module.

3.3 Transferablity Evaluation

We transfer the discovered x-level model to s, m, and l levels to evaluate the transferability of the discovered architecture. Table 4 compares the performance of the discovered models and the transferred models on the validation set of

Table 5. Study of SPP on MS-COCO validation set. 's-16-8' is the original transferred model from x-level, while 's-32-8' and 's-32-16' are the modified models by adding connections from the SPP layer.

Model	w/ SPP	mAP	Δ
YOLOv5-s	✓	36.9	+0.0
s-16-8 (transferred from x)	×	37.4	+0.5
s-32-16 (transferred from x	✓	39.0	+2.1
s-32-8 (transferred from x)	✓	38.8	+1.9

MS-COCO. We observe that: 1) Though transferred s-level and m-level can outperform baselines (YOLOv5), significant gaps exist between them and the directly searched models; 2) The transferred l-level model and the searched one achieve competitive performance. We attribute it to the different architecture preferences for small and large neural networks. By comparing the discovered architectures, we find that they have similar backbone structures but rather different FPN structures, as shown in Fig. 7. Specifically, s-level and m-level have the same FPN topology but differ in operation types (kernel size and dilation ratio of convolutions). Besides, both l-level and x-level discard the 32× downsampled features, which is the output of SPP.

To verify our analysis, we transfer the searched m-level model to s-level since they have similar FPN structures. Table 4 shows that the s-level model transferred from m achieves 39.5% mAP, significantly surpassing the one transferred from x by more than 2%.

3.4 Discussion on Spatial Pyramid Pooling

Spatial pyramid pooling (SPP) [12] is designed to integrate various receptive fields and extract multi-scale features with the same spatial size. Most recent manually-designed detectors adopt it by default, including YOLOv4 and YOLOv5. However, our experiments show that the value of SPP degrades when the network gets deeper. As shown in Fig. 7, models on l-level and x-level discard the SPP layer and choose to enlarge the receptive field by 5×5 convolution. While models on s-level and m-level still prefer the SPP layer.

In our analysis, SPP is vital for shallow networks as it can increase the receptive field to extract global information. However, the receptive field is enough for deep networks, making the SPP layer dispensable with the increment of network depth. Results in Table 4 supports our analysis: When transferring x-level models to s-level, the performance degrades significantly. To verify the effectiveness of SPP for shallow networks, we manually add SPP for the transferred s-level model. In Fig. 7(d), the blue node connects with 16× and 8× downsampled features. We construct two models by removing one of the connections (blue lines) and connecting the blue node with the dashed node (output of SPP layer), whose performance on the COCO validation set is reported in Table 5.

Table 6. Comparison to YOLOv5 on the test set of oriented bounding box (OBB) task in DOTA-v1.0.

Model	s-level (small)		m-level (medium)	
	YOLOv5	EAutoDet	YOLOv5	EAutoDet
#Params$_{(M)}$	7.6	21.7	8.7	22.7
FLOPS$_{(G)}$	17.5	50.6	21.5	50.8
mAP$_{50(\%)}$	74.00	76.33	75.72	77.05

We observe that after recovering the SPP layer, the performance of transferred model can be improved significantly.

3.5 Results on Rotation Detection Benchmark: DOTA

We utilize Circular Smooth Label (CSL) technique [40] to obtain robust angular prediction through classification without suffering boundary conditions. Our models are trained with rotation classification loss used in CSL [40] from scratch for 300 epochs. We compare to YOLOv5 based on the open-sourced codes [16]. Results are shown in Table 6. We also compare with other rotation detection baselines that adopts RetinaNet [19] detection framework with ResNet152 [13] backbone and FPN [18] module are given in the supplementary.

The above results show the generalization of our discovered architectures, which also verify the effectiveness of our search method. Notice that the discovered architectures are not limited to the specific CSL as tested in our experiment, as the search paradigm is agnostic to the choice of rotation detection loss, e.g., GWD [41] and BBAVectors [43], which we leave for future work.

4 Conclusion

This paper introduces kernel and dynamic channel refinement techniques and proposes a fast and memory-efficient search method for detection. We also design a sophisticated and large search space for detection by absorbing the knowledge of well-designed architectures of YOLO models. Our method can discover lightweighted models in 1.4 GPU-days, achieving 40.1 mAP on COCO test-dev with 120 FPS surpassing state-of-the-art NAS methods. Besides, our ablation studies suggest that the SPP plays a more vital role in shallow models than in deep models in the hope of facilitating future network design for detection. Moreover, the discovered architecture archives 77.05% mAP_{50} on DOTA-v1.0 benchmark, outperforming most of the manually-designed models e.g. CSL [40] (76.24%), further verifying the effectiveness of our search method.

Acknowledgements This work was supported in part by by National Key Research and Development Program of China (2020AAA0107600), National Science of Foundation China (61972250, 72061127003), and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

1. Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., Le, Q.V.: Understanding and simplifying one-shot architecture search. In: ICML (2018)
2. Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934 (2020)
3. Chen, Y., Yang, T., Zhang, X., Meng, G., Xiao, X., Sun, J.: Detnas: Backbone search for object detection. NeurIPS (2019)
4. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. In: CVPR (2021)
5. Dong, X., Yang, Y.: Searching for a robust neural architecture in four gpu hours. In: CVPR (2019)
6. Du, X., Lin, T.Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., Le, Q.V., Song, X.: Spinenet: Learning scale-permuted backbone for recognition and localization. In: CVPR (2020)
7. Ghiasi, G., Lin, T.Y., Le, Q.V.: Nas-fpn: Learning scalable feature pyramid architecture for object detection. In: CVPR (2019)
8. Girshick, R.: Fast r-cnn. In: ICCV (2015)
9. Gumbel, E.J.: Statistical theory of extreme values and some practical applications: a series of lectures, vol. 33. US Government Printing Office (1954)
10. Guo, J., Han, K., Wang, Y., Zhang, C., Yang, Z., Wu, H., Chen, X., Xu, C.: Hit-detector: Hierarchical trinity architecture search for object detection. In: CVPR (2020)
11. Han, J., Ding, J., Xue, N., Xia, G.: Redet: A rotation-equivariant detector for aerial object detection. In: CVPR (2021)
12. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. TPAMI (2015)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
14. Jiang, C., Xu, H., Zhang, W., Liang, X., Li, Z.: Sp-nas: Serial-to-parallel backbone search for object detection. In: CVPR (2020)
15. Jocher, G.: Yolov5 documentation. https://docs.ultralytics.com/ (May 2020)
16. Kaixuan, H.: Yolov5 for oriented object detection. https://github.com/hukaixuan19970627/yolov5_obb (2020)
17. Liang, T., Wang, Y., Tang, Z., Hu, G., Ling, H.: Opanas: One-shot path aggregation network architecture search for object detection. In: CVPR (2021)
18. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: CVPR (2017)
19. Lin, T., Goyal, P., Girshick, R.B., He, K., Dollár, P.: Focal loss for dense object detection. TPAMI **42**(2), 318–327 (2020)
20. Lin, T., Maire, M., Belongie, S.J., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. In: Fleet, D.J., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV (2014)
21. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)
22. Liu, S., Qi, L., Qin, H., Shi, J., Jia, J.: Path aggregation network for instance segmentation. In: CVPR (2018)
23. Liu, S., Huang, D., Wang, Y.: Learning spatial fusion for single-shot object detection. arXiv preprint arXiv:1911.09516 (2019)

24. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: ECCV (2016)
25. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
26. Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/ (2013–2016)
27. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR (2016)
28. Ren, S., He, K., Girshick, R.B., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) NeurIPS (2015)
29. Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-path NAS: designing hardware-efficient convnets in less than 4 hours. In: ECML (2019)
30. Tan, M., Pang, R., Le, Q.V.: Efficientdet: Scalable and efficient object detection. In: CVPR (2020)
31. Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., Vajda, P., Gonzalez, J.E.: Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In: CVPR (2020)
32. Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M.: Scaled-yolov4: Scaling cross stage partial network. In: CVPR (2021)
33. Wang, N., Gao, Y., Chen, H., Wang, P., Tian, Z., Shen, C., Zhang, Y.: Nas-fcos: Fast neural architecture search for object detection. In: CVPR (2020)
34. Wang, X., Xue, C., Yan, J., Yang, X., Hu, Y., Sun, K.: Mergenas: Merge operations into one for differentiable architecture search. In: IJCAI (2020)
35. White, C., Neiswanger, W., Savani, Y.: BANANAS: bayesian optimization with neural architectures for neural architecture search. In: AAAI (2021)
36. Xu, H., Yao, L., Li, Z., Liang, X., Zhang, W.: Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In: ICCV (2019)
37. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.J., Tian, Q., Xiong, H.: Pc-darts: Partial channel connections for memory-efficient architecture search. In: ICLR (2020)
38. Xue, C., Wang, X., Yan, J., Li, C.G.: A flow-based approach for neural architecture search. In: ECCV (2022)
39. Yang, X., Hou, L., Zhou, Y., Wang, W., Yan, J.: Dense label encoding for boundary discontinuity free rotation detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 15819–15829 (2021)
40. Yang, X., Yan, J.: Arbitrary-oriented object detection with circular smooth label. In: ECCV (2020)
41. Yang, X., Yan, J., Qi, M., Wang, W., Xiaopeng, Z., Qi, T.: Rethinking rotated object detection with gaussian wasserstein distance loss. In: International Conference on Machine Learning (2021)
42. Yao, L., Xu, H., Zhang, W., Liang, X., Li, Z.: Sm-nas: structural-to-modular neural architecture search for object detection. In: AAAI (2020)
43. Yi, J., Wu, P., Liu, B., Huang, Q., Qu, H., Metaxas, D.: Oriented object detection in aerial images with box boundary-aware vectors. In: Proceedings of the IEEE Winter Conference on Applications of Computer Vision. pp. 2150–2159 (2021)
44. Zhao, Z., Wu, Z., Zhuang, Y., Li, B., Jia, J.: Tracking objects as pixel-wise distributions (2022)
45. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)