Hit-Detector: Hierarchical Trinity Architecture Search for Object Detection

Jianyuan Guo[1,2], Kai Han[2], Yunhe Wang[2], Chao Zhang[1]*, Zhaohui Yang[1]
Han Wu[1], Xinghao Chen[2], Chang Xu[3]

[1] Key Lab of Machine Perception (MOE), Dept. of Machine Intelligence, Peking University.

[2] Noah's Ark Lab, Huawei Technologies. [3] School of Computer Science, Faculty of Engineering, University of Sydney.

{jyguo, zhaohuiyang, wuhancs}@pku.edu.cn, chzhang@cis.pku.edu.cn
c.xu@sydney.edu.au, {kai.han, yunhe.wang, xinghao.chen}@huawei.com

Abstract

*Neural Architecture Search (NAS) has achieved great success in image classification task. Some recent works have managed to explore the automatic design of efficient backbone or feature fusion layer for object detection. However, these methods focus on searching only one certain component of object detector while leaving others manually designed. We identify the inconsistency between searched component and manually designed ones would withhold the detector of stronger performance. To this end, we propose a hierarchical trinity search framework to simultaneously discover efficient architectures for all components (*i.e. *backbone, neck, and head) of object detector in an end-to-end manner. In addition, we empirically reveal that different parts of the detector prefer different operators. Motivated by this, we employ a novel scheme to automatically screen different sub search spaces for different components so as to perform the end-to-end search for each component on the corresponding sub search space efficiently. Without bells and whistles, our searched architecture, namely Hit-Detector, achieves 41.4% mAP on COCO minival set with 27M parameters. Our implementation is available at https://github.com/ggjy/HitDet.pytorch.*

1. Introduction

Object detection is a fundamental task in computer vision and has been widely applied in the real world, such as autonomous vehicles and surveillance video. The advancement of deep learning results in a number of convolutional neural network based solutions of object detection task. Typically, deep learning based detectors can be divided into two categories: (i) one-stage methods including YOLO [40] and SSD [33] which directly utilize CNNs to predict the bounding boxes of interest; and (ii) two-stage approaches such as Faster R-CNN [42] that generates the bounding boxes after extracting region proposals upon a region proposal network (RPN). The advantage of single-stage methods lies in the high detection speed whereas two-stage methods dominate in detection accuracy.

Table 1. Comparing our model against some typical two-stage detectors on COCO benchmark. "4c" indicates four convolution layers. † means that NAS-FPN is originally searched for one-stage RetinaNet, we replace the neck in FPN with NAS-FPN to construct the two-stage detector.

Model	Backbone (#params/M)	Neck (#params/M)	Head (#params/M)	mAP (%)
FPN baseline [26]	Res50 (23.5)	FPN (3.3)	2fc (14.3)	36.2
NAS-FPN [13]†	Res50 (23.5)	NAS-FPN (30.4)	2fc (14.3)	38.9
Backbone baseline	Res50 (23.5)	FPN (3.3)	4c1fc (15.6)	36.8
DetNAS [8]	DetNet (9.4)	FPN (2.8)	4c1fc (15.6)	40.2
NAS-FPN + DetNAS	DetNet (9.4)	NAS-FPN (29.7)	4c1fc (15.6)	39.4
Hit-Detecotr (ours)	Ours (13.9)	Ours (2.7)	Ours (9.9)	41.4

A series of either one-stage [41, 27, 22] or two-stage approaches [9, 18, 5] have been developed to continuously boost the detection speed and accuracy. However, the manually designed architectures heavily rely on the expert knowledge [25] while still might be suboptimal. Thus neural architecture search (NAS) that automates the design of network architectures and minimizes human labor has drawn much attention and made impressive progress, especially in image classification tasks [57, 29, 45, 31, 43, 46, 48, 55, 17]. Compared with classification tasks that simply determine *what* the image is, detection tasks need to further figure out *where* the objects are. NAS for object detection therefore requires more careful design and is much more challenging.

Modern object detection systems usually consist of four components: (a) backbone for extracting semantic features, *e.g.* ResNet-50 [19] and ResNeXt-101 [49]; (b) neck for fusing multi-level features, *e.g.* feature pyramid networks (FPN) [26]; (c) RPN for generating proposals (usually in two-stage detector); and (d) head for object classification

*Corresponding author.

and bounding box regression. Recently, there are works that explore NAS in object detection tasks to search for a good architecture of the backbone [37, 8] or FPN [13, 51]. With the searched backbone or FPN architectures, these works have achieved higher accuracy than the manually designed baselines with similar numbers of parameters and FLOPs.

Nevertheless, exploiting only one part of the detector at a time cannot fulfill the the potential of each component, and the separately searched backbone and neck may not be optimal or compatible with each other. As shown in Table 1, NAS-FPN [13] for neck searching achieves a 38.9% mAP which is higher than that of vanilla FPN, and DetNAS [8] for backbone searching outperforms vanilla ResNet-50 backbone with 40.2% mAP. However, a straightforward combination of NAS-FPN and DetNAS leads to a worse mAP, *i.e*., 39.4%, let alone outperforms two models. This insightful observation motivates us to take the detector as a whole in NAS.

In this paper, we propose to simultaneously search all components of the detector in an end-to-end manner. Due to the differences among optimum space for each component and the difficulty in optimizing within large search space, we introduce a hierarchical way to mine the proper sub search space from the large volume of operation candidates. In particular, our proposed *Hit-Detector* framework consists of two key procedures as shown in Fig. 1. First, given a large search space containing all the operation candidates, we screen out the customized sub search space suitable for each part of detector with the help of group sparsity regularization. Secondly, we search the architectures for each part within the corresponding sub search space by adopting the differentiable manner. Extensive experiments demonstrate that our Hit-Detector achieves state-of-the-art results on the benchmark dataset, which validates the effectiveness of the proposed method.

Our main contributions can be summarized as follows:

- This is the first time that architectures of backbone, neck and head are searched altogether in an end-to-end manner for object detection.
- We show that different parts prefer different operations, and propose a hierarchical way to specify appropriate sub search space for different components in detection system to improve the sampling efficiency.
- Our Hit-Detector outperforms either hand-crafted or automatically searched networks by a large margin with much less computational complexity.

2. Related Work

Object detection aims at determining *what* and *where* the object is when given an image. Riding the wave of convolutional neural networks, noticeable improvements in accuracy have been made in both one-stage [40, 33, 27, 22, 11, 56] and two-stage [42, 18, 9, 26, 20, 21, 5, 32] detectors. Generally, object detector consists of four parts: a backbone that extracts features from input images, a neck attached to backbone that fuses multi-level features, a region proposal network which generates prediction candidates on extracted features[1], and a head for classification and localization.

In the past few years, various methods in the literature have been proposed to tackle with detection task and attain significant progress. With NAS prospering automating design of model architecture, it has also boosted the probe into automatically searching for the best architecture for object detection, other than manually design. Here we briefly review some of the recent detectors in two dimensions:

2.1. Manual design

The manual design of detectors in mainstream evolution of object detection solutions is promoted by several works. R-CNN [15] is the first to show that a CNN could lead to dramatic performance improvement in object detection. Selective Search [47] is used to generate proposals and SVM is applied to classify each region. Following R-CNN, Fast R-CNN [14] is proposed to improve the speed by sharing computation of convolutional layers between proposals. Faster R-CNN [42] replaces Selective Search with a novel RPN (region proposal network), further promoting the accuracy and makes it possible to train model in an end-to-end manner. In addition, Mask R-CNN [18] extends Faster R-CNN mainly in instance segmentation task. Meanwhile, a series of proposal free detectors, *i.e*. one-stage detectors, has been proposed to speed up the detection. YOLO [40] and YOLOv2 [41] extract features from input images straightly for predicting bound boxes and associated class probabilities through a unified architecture. SSD [33] further improves the mAP by predicting a set of bounding boxes from several feature maps with different scales.

In the mean time, some arts concentrate on improving specific parts such as backbone, neck, and head to improve the efficiency in object detectors. DetNet [25] specifically designs a novel backbone network for object detection. FPN [26] develops a top-down architecture to effectively encode features from different scales. PANet [32] further modifies neck module to obtain a better fusion. Focal loss [27] is proposed to solve the problem of class imbalance. MetaAnchor [53] proposes a flexible mechanism which generates anchor from arbitrary prior boxes. Light-Head R-CNN [24] designs a light head for two-stage detector to decrease the computation cost accordingly.

2.2. Neural architecture search

NAS for image classification NAS (neutral architecture search) has attracted great attention recently. Several rein-

[1]There are only three parts (backbone, neck, and head) for one-stage detector. We take two-stage detector as an example here.

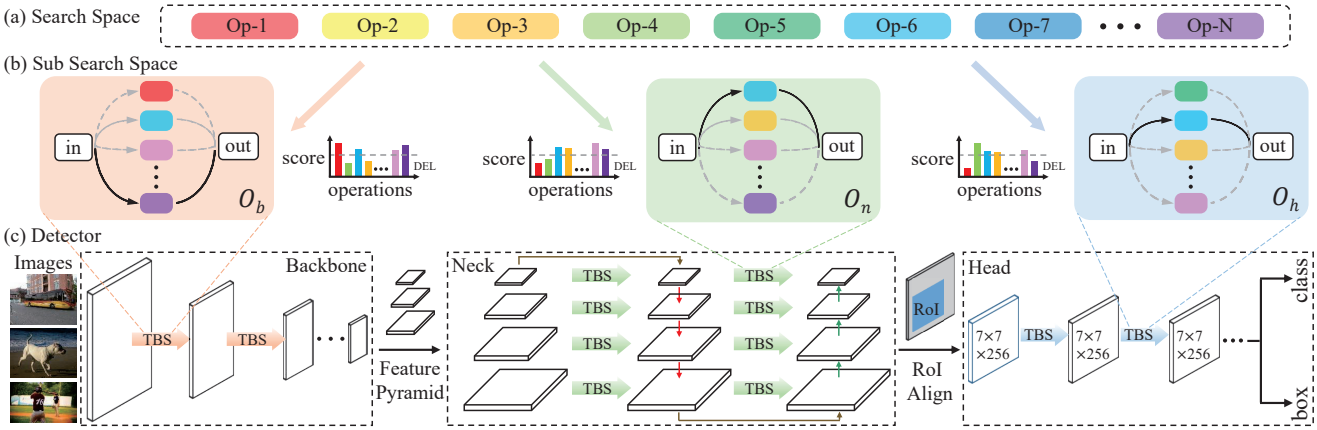


Figure 1. Overview of our Hit-Detector architecture search framework. Our method focuses on searching better architectures of the trinity, *i.e*. backbone, neck, and head for object detector. (a) is the whole search space; (b) indicates three sub search spaces for different components; and (c) shows the end-to-end searching for object detector. "TBS" denotes the layer to be searched.

forcement learning based methods [1, 3, 57, 58, 29] train a RNN controller to generate cell structure and form the network accordingly. Evolutionary algorithm based methods [30, 35, 38, 39, 16, 45, 54] are also proposed to update architectures by mutating current ones. To speed up searching process, gradient based methods [31, 7, 48, 50, 4, 52] are proposed for continuous relaxation of search space, which allow the differentiable optimization in architecture search.

NAS for object detection In addition to NAS works on classification, some recent works attempt to develop NAS for object detector. NATS [37] claims that the effective receptive field of backbone is critical and uses NAS to search different dilation rates for each convolution layer in backbone. Similarly, DetNAS [8] aims to search a better backbone for detection task. NAS-FPN [13] targets at a better architecture of feature pyramid network for object detection, adopting NAS to discover a new feature pyramid architecture covering all cross-scale connections. Auto-FPN [51] sequentially searches a better fusion of the multi-level features for neck and a better structure for head. However, there are two shortcomings in above methods: (i) the search space is defined by human prior and might be too naive for searching (*e.g*. four choices based on ShuffleNetV2 [34] in DetNAS [8]); (ii) in each work, only one certain part is searched (*e.g*. backbone in [37, 8], neck in [13]) can lead to suboptimal result in detection task. To tackle these two challenges, we propose Hit-Detector that filters proper search space for each parts hierarchically and searches every parts for a better detector in an end-to-end manner.

3. Hit-Detector

In this section, we introduce the proposed Hierarchical Trinity architecture search algorithm for object detection and the resulted detector, *i.e*. Hit-Detector. We first identify and analyze the problems in current NAS algorithms for object detection to clarify our motivation that we need to search all components together. Then we detail how to hierarchically filter the sub search space for each parts, and last, the end-to-end search process for Hit-Detector is depicted. The following statement of search algorithm is based on two-stage detection methods and can be easily applied to one-stage methods.

3.1. Preliminaries and Motivation

Two-stage detection system can be decoupled as four components: (i) **Backbone.** Commonly used backbones in detection system such as ResNet [19] and ResNeXt [49] are mostly manually designed for classification tasks. Usually, the largest proportion of parameters in a detector comes from backbone. For example, ResNet-101, the backbone of FPN [26], takes up 71% parameters of all, leaving a large potential for searching; (ii) **Neck.** Employing in-network feature pyramids to approximate different receptive fields can help detector localize objects better. Previous architectures of feature fusion neck are manually designed while NAS can exploit better operations for merging features from different scales; (iii) **RPN.** The typical RPN is lightweight and efficient: a convolution layer followed by two fully connected layers for region proposal classification and bounding box regression. We follow this design for its efficiency and do not search for this part; and (iv) **Head.** Detectors usually have a heavy head attached to former network. For example, Faster R-CNN [42] employs the 5-th stage in ResNet [19] and FPN [26] uses two large fully connected layers (34% parameters of whole detector) to execute classification and regression, which are inefficient for detection. We call the backbone, the neck, and the head which are valuable to be searched as detector trinity in this paper.

Recently, several methods have been proposed to search either backbone [8] or feature pyramid architecture [13] for

object detection. Denoting the search space $\mathcal{A}$ as a directed acyclic graph (DAG) where the nodes indicate the features while directed edges are associated with multifarious operations such as convolution layer and pooling layer. We can view each path from start point to end point in the graph as an architecture $\alpha \in \mathcal{A}$. Previous NAS works for object detection can be formulated as the optimization problem:

$$\begin{aligned}\alpha^* &= \arg\min_{\alpha\in\mathcal{A}} f(\alpha) = \arg\min_{\alpha\in\mathcal{A}} \mathcal{L}_{val}^{det}(\alpha, w^*(\alpha)) \\ &= \arg\min_{\alpha\in\mathcal{A}} \mathcal{L}_{val}^{det}(\alpha, \arg\min_{w} \mathcal{L}_{train}^{det}(\alpha, w)).\end{aligned} \tag{1}$$

The purpose of NAS process is to find a specific architecture $\alpha \in \mathcal{A}$ that minimizes the validation loss $\mathcal{L}_{val}^{det}(\alpha, w_\alpha^*)$ with the trained weights w_α^*. The above formulation can represent searching on backbone (*e.g.* $\mathcal{A}$ denotes the backbone search space) or feature pyramid network (*e.g.* $\mathcal{A}$ denotes the FPN search space).

However, the backbone, the neck (feature fusion network), and the head in object detection system should be highly consistent to each other. Only redesigning the backbone or neck is not enough, which can lead to suboptimal results. As shown in Table 1, combining the separately searched backbone in DetNAS and neck in NAS-FPN leads to a worse result. We claim that separately search backbone α, neck β, and head γ in detector is inferior to search all these components end-to-end:

$$\begin{aligned}&\mathcal{L}_{val}^{det}(\alpha', \beta', \gamma') \geq \mathcal{L}_{val}^{det}(\alpha^*, \beta^*, \gamma^*), \\ &\text{s.t. } \alpha', \beta', \gamma' = \arg\min_{\alpha} f(\alpha), \arg\min_{\beta} f(\beta), \arg\min_{\gamma} f(\gamma), \\ &\alpha^*, \beta^*, \gamma^* = \arg\min_{\alpha,\beta,\gamma} \mathcal{L}_{val}^{det}(\alpha, \beta, \gamma, w^*(\alpha, \beta, \gamma)), \\ &\alpha \in \mathcal{A}_b,\ \beta \in \mathcal{A}_n,\ \gamma \in \mathcal{A}_h.\end{aligned} \tag{2}$$

where α', β', γ' are obtained by solving corresponding optimization problem as in Eq. 1 separately, while $\alpha^*, \beta^*, \gamma^*$ are optimized through the end-to-end searching algorithm, and $\mathcal{A}_b, \mathcal{A}_n, \mathcal{A}_h$ are search spaces for backbone, neck, and head, respectively.

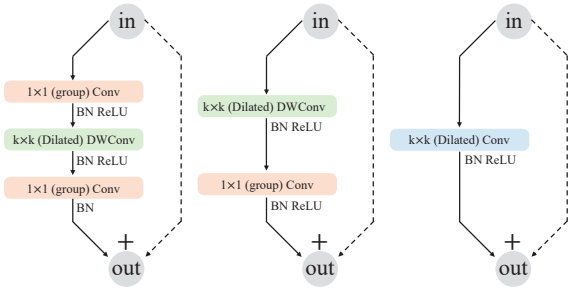
In this paper, we propose to search the trinity in detectors, *i.e.* backbone, neck, and head in an end-to-end manner:

$$\begin{aligned}&\alpha^*, \beta^*, \gamma^* = \arg\min_{\alpha,\beta,\gamma} \mathcal{L}_{val}^{det}(\alpha, \beta, \gamma, w^*(\alpha, \beta, \gamma)) = \\ &\arg\min_{\alpha,\beta,\gamma} \mathcal{L}_{val}^{det}(\alpha, \beta, \gamma, \arg\min_{w} \mathcal{L}_{train}^{det}(\alpha, \beta, \gamma, w)).\end{aligned} \tag{3}$$

As shown in Figure 1, we propose the hierarchical trinity architecture search framework to solve problems in Eq. 3, which includes two procedures: screening sub search space for each component and searching the trinity end-to-end.

3.2. Screening Sub Search Space

Search space is one of the key factors in neural architecture search. The search spaces in [8, 37, 51] are manually designed with a limited number of operation candidates. Furthermore, there is usually the case that manually designed search space is not suitable for the architecture that needs to be optimized. In order to prevent insufficient search space, we start from the FBNet [48] and traverse as many candidates as possible to form a large set of operation candidates, which are illustrated in Figure 2. Inverted residual block [44] contains a 1×1 convolution, a $k \times k$ depthwise convolution, another 1×1 convolution and an expansion factor e. Separable block contains a $k \times k$ depthwise convolution and a 1×1 convolution. If the output dimension is the same as input's, we use a skip connection to add them together.



(a) Inverted Residual block (b) Separable block (c) Convolution block

Figure 2. The block structure of our search space, *e.g.*, candidate operations in inverted residual block can choose a different expansion rate, kernel size and number of groups for group convolution.

As shown in Figure 1(a), the whole search space consists of N different operation candidates, *e.g.*, $N = 32$ in our experimental setting. If we directly apply this large search space for NAS, the memory and computational overhead is so heavy that ordinary hardware cannot support the search process efficiently. Moreover, in Sec 4.4, we empirically show that the same operation can have different impacts on final result at different parts in detection system. In order to find the most suitable search space for each component and reduce the computational burden, we propose a screening scheme to hierarchically filter operation candidates for each component. As shown in Figure 1(b), every candidate is associated with a score. The candidates with higher scores are retained, and others with lower scores are deleted.

Take the search space of backbone as an example, we conduct layer-wise search where each layer can choose any operation from the candidates. Assuming that the backbone has L layers, the score of the i-th operation in the l-th layer is denoted as $\alpha_{l,i}$. All scores form the architecture parameter matrix $\alpha \in \mathbb{R}^{L\times N}$, where the i-th column represents the scores of the i-th operation in corresponding layers. With a relatively large amount of candidates, the scores tend to be similar and are hard to distinguish. In order to screen the most suitable operation subset, the matrix α is imposed with the column-sparse regularization:

$$\min_{\alpha} f(\alpha) + \mu \min_{i}(\sqrt{\sum_{l=1}^{L} \alpha_{l,i}^2}), \tag{4}$$

μ is a trade-off hyper-parameter. In screening stage, the architecture parameter α is learned and the scores of candidates are ranked accordingly, the last several candidates will be removed from search space gradually until a search space with size N_b is obtained. The sub search spaces for backbone, neck, and head are hierarchically selected from the whole search space, namely, $\mathcal{O}_b$, $\mathcal{O}_n$, and $\mathcal{O}_h$.

3.3. End-to-end Search in Hit-Detector

After obtaining the suitable sub search space for each component, we start the end-to-end search for object detector. We adopt the differentiable manner proposed in [31] to solve Eq. 3, and represent the sub search space by a stochastic supernet. During searching, each intermediate node is computed as a weighted sum based on all candidates. For backbone, the l-th node is formulated as

$$x_l = \sum_{o \in \mathcal{O}_b} \frac{\exp(\alpha_l^o)}{\sum_{o' \in \mathcal{O}_b} \exp(\alpha_l^{o'})} o(x_{l-1}), \tag{5}$$

where x_l is the output of the l-th layer, α_l^o is the parameter for operation $o(\cdot)$, and $\mathcal{O}_b$ indicates the sub search space for backbone. The nodes in neck and head are similar to Eq. 5. This continuous relaxation makes the entire framework differentiable to both operation weights and architecture parameters, so that we can perform architecture search in an end-to-end manner. In testing phase, we can easily decode architectures from α, β, γ by choosing operation with the highest score in each layer and construct detector using the selected operation.

The details of our detector supernet are described as follows. The basic structure of **backbone** consists of a 3×3 convolution head with stride of 2 followed by four stages that contain $4+4+8+4=20$ blocks to be searched. Conventionally, we define the layers that producing feature map with the same spatial size belong to the same network stage. In each stage, the first block has a stride of 2 for downsampling, and the last feature map generated by the backbone has a downsample rate of 32 compared to the input image. The channel of each stage is set to $\{48, 96, 256, 352\}$ respectively. We use $\{C_1, C_2, C_3, C_4\}$ to represent feature maps of stride $\{4, 8, 16, 32\}$ generated by backbone. Then we send the feature pyramid into the **neck**. Generally, high level features have better semantic information while low level features have more accurate location information. To propagate semantic signals and accurate information about localization to features from lower and higher level, we use both top-down and bottom-up path augmentation to enhance features from different levels inspired by [32]. We use $\{P_1, P_2, P_3, P_4\}$ and $\{N_1, N_2, N_3, N_4\}$ to denote feature maps after top-down and bottom-up path, respectively. The whole neck contains $4+4=8$ lateral connections to be searched, thus each feature level can search for different operations to assign proper receptive fields. Given the aligned feature maps generated by region proposal network, **head** is applied to predict final classification and refine the bounding box of the object. We design 4 blocks to be searched followed by a fully connected layer to form the detection head. The number of the output channels for each level in neck and head is 256 and the output of the fully connected layer is a 512 dimensional vector. In our experiments, we set $N_b = N_n = N_h = 8$. Even if we extract the sub search spaces carefully, the final search space contains $8^{(20+8+4)} \approx 7.9 \times 10^{28}$ possible architectures.

3.4. Optimization

In order to control the computational cost of the searched detector at the same time, we add the FLOPs constraint as a regularization term in the loss function and rewrite the Eq. 3 as:

$$\min_{\alpha,\beta,\gamma} \mathcal{L}_{val}^{det}(\alpha, \beta, \gamma, w^*(\alpha, \beta, \gamma)) + \lambda(\mathrm{C}(\alpha) + \mathrm{C}(\beta) + \mathrm{C}(\gamma))) \tag{6}$$

where λ is the coefficient to balance accuracy and cost of the detector, $\mathrm{C}(\alpha)$ indicates FLOPs of the backbone part and can be decomposed as linear sum of each operations:

$$\mathrm{C}(\alpha) = \sum_{l} \sum_{o \in \mathcal{O}_b} \alpha_l^o \mathrm{FLOPs}(o, l), \tag{7}$$

$\mathrm{C}(\beta)$ and $\mathrm{C}(\gamma)$ can be calculated similarly.

It is clear that continuous relaxation from Eq. 5 is differentiable with respect to architecture parameters and operation weights, thus $\{\alpha, \beta, \gamma, w\}$ can be optimized jointly using stochastic gradient descent. We adopt first-order approximation following [31] and update architecture parameters and operation weights alternately. We first fix $\{\alpha, \beta, \gamma\}$ and compute $\partial\mathcal{L}/\partial w$ to train the network weights on 50% training data, then we fix the network weights and calculate $\partial\mathcal{L}/\partial\alpha$, $\partial\mathcal{L}/\partial\beta$, and $\partial\mathcal{L}/\partial\gamma$ to update the architecture parameters on the remaining 50% training data, where the loss function $\mathcal{L}$ is the localization and classification losses calculated on the detection mini-batch. The optimization alternates until the supernet converges.

4. Experiments

In this section, we investigate the effectiveness of proposed Hit-Detector by conducting elaborate experiments on COCO benchmark.

4.1. Dataset and Metrics

We conduct experiments on MS COCO 2014 dataset [28], which contains 80 object classes. Following [2, 26], our training set is the union of 80k training images and 35k subset of validation images (trainval35k), and validation set

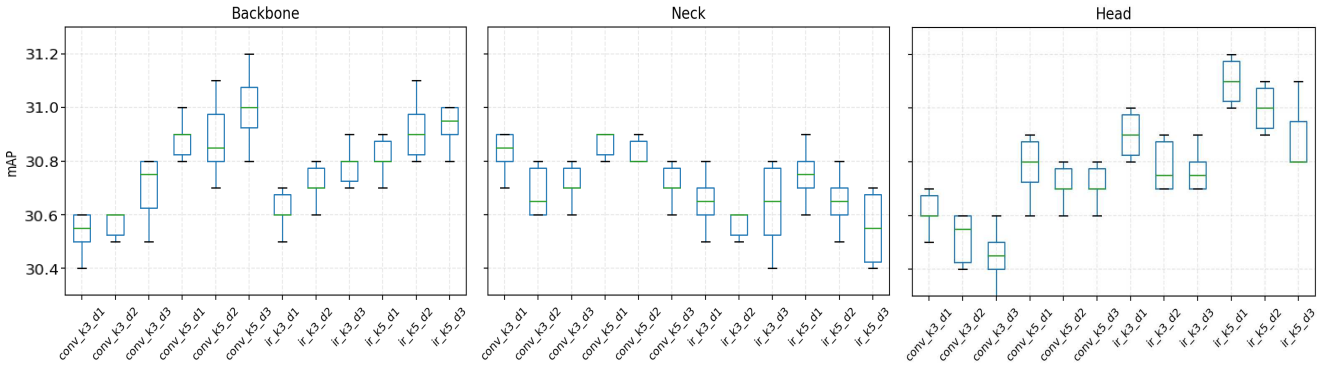


Figure 3. Influence of replacing a certain operation in different parts of the detector on COCO minival. FPN (4clfc head) is taken as a base detector and the input image is of size 320×320. Taking the left figure as an example, each time one layer from backbone is randomly chosen to be replaced by an operation candidate. For one operation candidate, such random process is repeated for 6 times.

is the remaining 5k validation images (minival). We consider Average Precision with different IoU thresholds from 0.5 to 0.95 with an interval of 0.05 as evaluation metric, *i.e.*, mAP, AP_{50}, AP_{75}, AP_{S}, AP_{M} and AP_{L}. The last three measure performance w.r.t. objects with different scales.

4.2. Implementation Details

Our implementation is based on mmdetection [6] with Pytorch framework [36]. We firstly filter three sub search spaces for backbone, neck, and head, respectively. Then we search for our Hit-Detector following the algorithm in sec 3.4. Finally we train our searched model on the training set mentioned above. We use only horizontal flipping as data augmentation for training, and there is no data augmentation for testing. The experiments are conducted on 8 V100 GPUs.

Screening sub search space. We sequentially screen sub search spaces for different parts of the detector due to the GPU memory constraint. The number of operations in the whole search space $\mathcal{O}$ is 32 as shown in Fig. 2, and the number of operations to be screened for all three sub search spaces $\mathcal{O}_b$, $\mathcal{O}_n$, and $\mathcal{O}_h$ is set as 8. We halved the depth of backbone supernet to $2+2+4+2=10$ for simplifying this process. We first pretrain the backbone supernet on ImageNet for 10 epochs with fixed architecture parameters, then fine-tune the entire detector supernet on COCO. SGD optimizer with momentum 0.9 and cosine schedule with initial learning rate 0.04 is used for learning model weights, while Adam optimizer with learning rate 0.0004 is adopted for updating architecture parameters. The μ in Eq. 4 is empirically set to 0.1. We begin to optimize architecture parameters at the 6-th epochs and finish searching at 12 epochs, and we don't use resource constraint in this stage.

Trinity Architecture searching. After screening sub search spaces, we start to search the backbone, neck and head in an end-to-end manner. We pretrain the new backbone supernet on ImageNet based on the corresponding sub search space, and then search the detector on COCO dataset. The optimizers to learn the architecture parameters and weights are the same as we do when screening sub search spaces. The λ in Eq. 6 is empirically set to 0.01 to trade off the accuracy and the FLOPs constraint.

Training details. We first pretrain the searched backbone on ImageNet for 300 epochs, then we fine-tune the whole detector on COCO training set. The input image is resized such that its shorter side has 800 pixels. We use SGD optimizer with a batch size of 4 images per GPU, and our model is trained for 12 epochs, known as $1\times$ schedule. The initial learning rate is 0.04 and is divided by 10 at the 8-th and 11-th epoch. We set momentum as 0.9 and weight decay as 0.0001.

4.3. Main Results

Comparisons with hand-crafted methods. FPN [26] with ResNet-50 as backbone is the baseline model here. We replace the backbone in FPN with other excellent backbones, *i.e.* MobileNetV2 [44] and ResNeXt-101 [49], and form two competitor models accordingly. As shown in Table 2, Hit-Detector surpasses baseline by a large margin with much less parameters. In addition, Our method is 1.1% higher on mAP compared to the ResNeXt based detector with less than one half of the parameters. And we outperform MobileNetV2 by 11.3% on mAP with only a bit more parameters. This demonstrates that our method can find a better architecture than hand-crafted baselines.

Comparisons with NAS based methods. As shown in Table 2, we compare our method with both detector searched on COCO benchmark and detector that adopts NAS based model as backbone. FBNet [48] is searched on ImageNet dataset and we directly apply it as the backbone of a detector, however, its performance on detection

Table 2. Comparisons of the number of parameters, FLOPs and mAP on COCO minival. The FLOPs is based on the 800×1200 input and 1000 proposals in region proposal network. ‡ means the 2x schedule in training.

model	modified			# params	# FLOPs	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	B	N	H	(total)	(total)						
FPN [26]	-	-	-	41.76M	197.4B	36.2	58.0	39.1	21.3	40.0	46.1
MobileNetV2 [44]	✓			19.61M	116.94B	30.1	51.8	30.9	16.7	33.0	38.7
ResNeXt-101 [49]	✓			60.38M	273.3B	40.3	62.1	44.1	23.6	45.0	51.6
FBNet-C [48]	✓			21.40M	119.0B	35.1	57.4	37.2	19.3	38.3	46.7
DetNAS-1.3G [8]	✓		✓	28.45M	254.1B	40.2	61.5	43.6	23.3	42.5	53.8
NASFPN [13]		✓		68.86M	616.9B	38.9	59.3	42.3	22.3	42.8	49.8
DetNAS-1.3G + NASFPN	✓	✓	✓	55.29M	672.9B	39.4	59.6	42.1	23.7	43.2	50.4
NATS-C [37]	✓			41.76M	197.4B	38.4	61.0	41.2	22.5	41.8	50.4
Auto-FPN‡ [51]		✓	✓	32.64M	476.6B	40.5	61.5	43.8	**25.6**	44.9	51.0
Hit-Detector	✓	✓	✓	27.12M	272.3B	**41.4**	**62.4**	**45.9**	25.2	**45.0**	**54.1**

task is disappointing. DetNAS [8] aims to search a better backbone directly on detection benchmark while leaves the rest parts unchanged. Our method outperforms DetNAS by 1.2% with less parameters and pretty much FLOPs. It can be seen that Hit-Detector surpasses all previous NAS based methods, which indicates that it is important to search the trinity in an object detector.

Comparison with State-of-the-arts on test-dev set. We also compare the results of our Hit-Detector with other state-of-the-art methods on the COCO test-dev, and we summarize the comparisons in Table 3. Hit-Detector only applies horizontal flipping as data augmentation and 2x training scheme, achieves 44.5% mAP without bells and whistles. Our model has less parameters and performs even better compared to other detectors such as TridentNet and NAS-FPN. This demonstrates that our method can find a better architecture than hand-crafted or partly searched methods.

Table 3. Comparisons of single-model results on COCO test-dev.

model	test size	mAP
R-FCN [9]	600/1000	32.1
Faster R-CNN [42]	600/1000	30.3
Deformable [10]	600/1000	34.5
FPN [26]	800/1200	36.2
Mask R-CNN [18]	800/1200	38.2
RetinaNet [27]	800/1200	39.1
Light head R-CNN [24]	800/1200	41.5
PANet [32]	800/1000	42.5
TridentNet [23]	800/1200	42.7
NAS-FPN [13]	1024/1024	44.2
Hit-Detector	800/1200	44.5

4.4. Ablation Studies

Influence of different operations. We use a toy example to demonstrate that different parts of the detector are sensitive to different operations, so that different parts need different sub search spaces. Here we simply choose 12 operations from the whole search space as shown in Figure 2. For example, "conv_k3_d1" denotes the convolution block with kernel size 3 and dilation rate 1, and "ir_k3_d1" denotes the inverted residual block with kernel size 3 and dilation rate 1, respectively. Take the left figure as an example, for each operation candidate, we randomly select a layer from backbone and replace original operation with the operation candidate to explore the influence of the selected operation candidate. For each part (backbone, neck, and head), we repeat the random process 6 times to ensure that operation candidate can be inserted in layers of different depths. We can find that (i) different parts prefer different operations. For example, operation "conv_k5_d3" achieves the highest mAP in backbone while "conv_k5_d1" performs the best in head; (ii) one operation has different performances in different parts, "conv_k3_d1" attains 30.4%-30.6% mAP in backbone but gets 30.7%-30.9% mAP in neck; and (iii) the performance of one operation within the same part is stable enough so that it's reasonable for us to screen sub search spaces by different parts.

Column-sparse regularization. To further evaluate the influence of the column-sparse regularization, we set the μ in Eq. 4 to $\{0, 0.01, 0.1\}$ and randomly choose 4 operation candidates to draw the Figure 4. We can find that if there is no column-sparse regularization ($\mu = 0$), probabilities of different operations are rather similar, which makes the screening process unstable. As μ increases, differences between operations become more significant, which helps to screen proper sub search space easier. We set $\mu = 0.1$ in the rest of our experiments.

Importance of screening sub search space. We study the influence of screening different sub search spaces through the ablation study shown in Table 4. When three parts have the same sub search space, the mAP decreases from 41.4% to 40.1%, which indicates that different parts need to have their proper sub search space to achieve better performance.

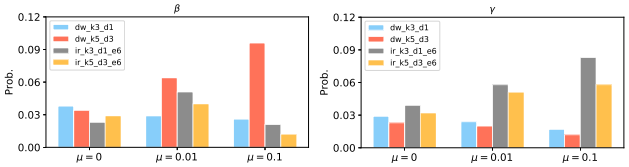


Figure 4. Ablation study of the trade-off parameter μ in column-sparse regularization.

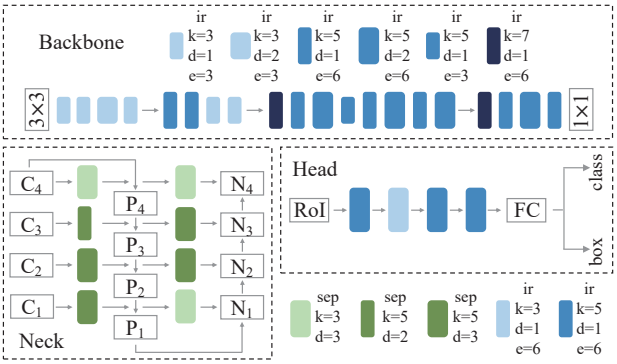


Figure 5. Visualization of the searched Hit-Detector. We use rectangle boxes to denote operations for each layer.

Table 4. Ablation study of screening sub search space.

model	mAP	AP_{50}	AP_{75}
1 sub search space	40.1	61.2	44.0
3 sub search space	41.4	62.4	45.9

Importance of trinity architecture searching. We explore the impact of each searched part in Hit-Detector here. Taking ResNet-50 based FPN [26] as baseline model, we replace one part of baseline model with our searched component each time and verify the new model on COCO minival. As shown in Table 5, our searched backbone architecture achieves 39.2% mAP, and the searched head also attains 38.5% mAP. With the searched trinity, the mAP of Hit-Detector increases to 41.4%, which is much higher than baseline model and the single part competitor. Another noteworthy finding is that the backbone and head can boost performance better than neck. We think the main reasons are that (i) object detection emphasizes more on perception to the location of each object in an image, thus the backbone designed for detection can perform better compared to the one designed for classification; and (ii) head aims to identify and refine the location of bounding boxes, thus searching more suitable convolution layers in head can bring more benefits to detection task.

Our searched Hit-Detector is depicted in Figure 5. We observed that the backbone prefers operations that have large kernel size. However, convolution layer with dilation 3 which achieves the best mAP in Figure 3 is not chosen. One main reason is that the ResNet-50 backbone in toy example only includes 3×3 convolution, thus operation with dilation 3 can boost the performance prominently. Except that, backbone and head prefer operations with bigger expansion rate such as inverted residual block which has more intermediate channels to increase the expression of feature, while the neck prefers bigger dilation rate for larger receptive fields.

Table 5. Evaluations of different parts searched in Hit-Detector. ✓ means we replace the part in baseline with the searched one.

model	Searched backbone	Searched neck	Searched head	mAP
FPN baseline				36.2
4c1fc baseline			✓	36.8
Searched backbone	✓			39.2
Searched neck		✓		37.4
Searched head			✓	38.5
Hit-Detector	✓	✓	✓	41.4

Extension to one-stage detector. To evaluate the generalization of our method, we apply it to RetinaNet [27] for searching one-stage detector. The search algorithm is the same as mentioned in sec 3.4. As shown in Table 6, our model outperforms RetinaNet by 1.3% in terms of mAP and the model size is smaller than both VGG based SSD and ResNet-50 based RetinaNet.

Table 6. Extending Hit-Detector to one-stage detector on COCO minival (1x schedule). † denotes the result of our implementation.

model	# Params	mAP	AP_{50}	AP_{75}
SSD-VGG19 [33]	36.04M	29.3	-	-
SSD-ResNet101 [12]	-	31.2	50.4	33.3
DSSD [12]	-	33.2	53.3	35.2
MobileNetV2† [44]	16.68M	31.5	50.9	33.5
RetinaNet† [27]	37.97M	35.6	55.6	38.4
Hit-Detector	33.05M	36.9	55.2	39.5

5. Conclusion

In this work, we propose a hierarchical trinity architecture search scheme to address the problem that incomplete searching of detector would cause the inconsistency between different components and lead to the suboptimal performance. We reveal that different components prefer different operations and thus screen three sub search spaces to improve the searching efficiency. Then we search for all components of object detectors based on the sub search spaces in an end-to-end manner. The searched architecture, namely Hit-Detector, achieves state-of-the-art performance on COCO benchmark without bells and whistles.

Acknowledgement This work of J. Guo and C. Zhang is supported by the National Nature Science Foundation of China under Grant 61671027 and the National Key R&D Program of China under Grant 2017YFB1002400, and C. Xu is supported by the Australian Research Council under Project DE180101438.

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

[2] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016.

[3] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint: 1812.00332*, 2018.

[5] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018.

[6] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint:1906.07155*, 2019.

[7] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *arXiv preprint: 1904.12760*, 2019.

[8] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. In *arXiv preprint:1903.10979*, 2019.

[9] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[10] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

[11] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Object detection with keypoint triplets. *arXiv preprint: 1904.08189*, 2019.

[12] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint: 1701.06659*, 2017.

[13] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019.

[14] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[16] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint: 1904.00420*, 2019.

[17] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. *arXiv preprint:1911.11907*, 2019.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[20] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *CVPR*, 2018.

[21] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, 2017.

[22] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *ECCV*, 2018.

[23] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. *arXiv preprint: 1901.01892*, 2019.

[24] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint: 1711.07264*, 2017.

[25] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *arXiv preprint: 1804.06215*, 2018.

[26] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[29] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[30] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint: 1711.00436*, 2017.

[31] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *arXiv preprint:1806.09055*, 2018.

[32] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018.

[33] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[34] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[35] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving

deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2019.

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[37] Junran Peng, Ming Sun, Zhaoxiang Zhang, Tieniu Tan, and Junjie Yan. Efficient neural architecture transformation searchin channel-level for object detection. In *arXiv preprint:1909.02293*, 2019.

[38] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[39] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[41] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017.

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[43] Tomoya Sakai, Marthinus Christoffel du Plessis, Gang Niu, and Masashi Sugiyama. Semi-supervised classification based on classification from positive and unlabeled data. In *ICML*, 2017.

[44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[45] Syed Asif Raza Shah, Wenji Wu, Qiming Lu, Liang Zhang, Sajith Sasidharan, Phil DeMar, Chin Guok, John Macauley, Eric Pouyoul, Jin Kim, et al. Amoebanet: An sdn-enabled network service for big data science. *Journal of Network and Computer Applications*, 2018.

[46] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[47] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *IJCV*, 2013.

[48] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[49] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[50] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint: 1812.09926*, 2018.

[51] Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *ICCV*, 2019.

[52] Chao Xue, Junchi Yan, Rong Yan, Stephen M Chu, Yonggang Hu, and Yonghua Lin. Transferable automl by model sharing over grouped datasets. In *CVPR*, 2019.

[53] Tong Yang, Xiangyu Zhang, Zeming Li, Wenqiang Zhang, and Jian Sun. Metaanchor: Learning to detect objects with customized anchors. In *NIPS*, 2018.

[54] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. *arXiv preprint:1909.04977*, 2019.

[55] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *ICML*, 2019.

[56] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *CVPR*, 2018.

[57] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint: 1611.01578*, 2016.

[58] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.