SP-NAS: Serial-to-Parallel Backbone Search for Object Detection

Chenhan Jiang*[1], Hang Xu*†[1],
Wei Zhang[1], Xiaodan Liang[2], Zhenguo Li[1]
[1]Huawei Noah's Ark Lab [2]Sun Yat-Sen University

Abstract

Advanced object detectors usually adopt a backbone network designed and pretrained by ImageNet classification. Recently neural architecture search (NAS) has emerged to automatically design a task-specific backbone to bridge the gap between the tasks of classification and detection. In this paper, we propose a two-phase serial-to-parallel architecture search framework named SP-NAS towards a flexible task-oriented detection backbone. Specifically, the serial-searching round aims at finding a sequence of serial blocks with optimal scale and output channels in the feature hierarchy by a Swap-Expand-Reignite search algorithm; the parallel-searching phase then assembles several sub-architectures along with the previous searched backbone into a more powerful parallel-structured backbone. We efficiently search a detection backbone by exploring a network morphism strategy on multiple detection benchmarks. The resulting architectures achieve SOTA results, i.e. top performance (LAMR: 0.055) on the automotive detection leaderboard of EuroCityPersons benchmark, improving 2.3% mAP with less FLOPS than NAS-FPN on COCO, and reaching 84.1% AP50 on VOC better than DetNAS and Auto-FPN in terms of both accuracy and speed.

1. Introduction

Object detection is one of the core computer vision tasks to localize and classify multiple objects in an image and has been widely used in real-world applications [29, 28, 7, 2]. Recent works, like FPN [19] and RetinaNet [20], directly use networks for ImageNet classification as backbone feature extractors, e.g., ResNet [15], which is neither task-specific nor data-specific. The backbone designs for object detection should be different from those for image classification, because the former need to localize and classify various objects at different scales in the feature hierarchy simultaneously while the latter only need to output the image-level labels by averaging the last feature map. Consequently, designing a capable backbone for object detection is more challenging and usually needs much human effort. Furthermore, some hand-crafted detection-specific backbone networks [38, 42, 19] are usually targeted at the COCO [21] dataset which cannot guarantee the adaptation to other detection tasks like in autonomous driving.

Model	Dataset	Task	Search space		
			Component	Size	Operator
ResNet [15]	ImageNet	Cls	Backbone	-	-
DetNet [18]	COCO	Det	Backbone	-	-
HRNet [42]	COCO	Det	Backbone	-	-
AmoebaNet [32]	ImageNet	Cls	Backbone	Fixed	Inter-block Connection
NAS-FPN [12]	COCO	Det	Neck	Fixed	Inter-block Connection
Auto-FPN [46]	COCO BDD VOC	Det	Neck Head	Fixed	Inter-block Connection
DetNAS [9]	COCO VOC	Det	Backbone	Fixed $\approx 10^{12}$	Kernel size
SP-NAS	COCO BDD ECP VOC	Det	Backbone	Growing $> 10^{20}$	#blocks Channels Downsampling

Table 1. Comparison of our SP-NAS and other backbone designs. 1) We directly search for a backbone on the detection dataset. 2) Our backbone keeps growing during search with a flexible and much bigger search space. 3) We search for the whole structure of backbone instead of the inter-block structure only.

Recently, Neural Architecture Search (NAS) algorithms demonstrate promising results on discovering top-accuracy architectures for image classification, which surpasses the performance of hand-crafted networks and saves human's efforts [49, 23, 45, 25, 31, 40, 30, 41]. However, NAS for object detection is usually more challenging due to the difficulty of fast evaluating the candidate models in the search space. Since the architecture of the backbone keeps changing during the search, one needs to pretrain the backbone on ImageNet [35] repeatedly, which is computationally infeasible. As a result, most NAS works do not directly search on the backbone for detection. For example, Zoph et al. [49] transfer the searched architecture from classification to detection backbone. NAS-FPN [12] and Auto-FPN [46] focus

*Both authors contributed equally to this work.
†Corresponding Author: xuhang33@huawei.com

on improving the feature fusion module (neck) by NAS.

To our knowledge, DetNAS [9] is the only work searching for the better backbone of detection (Table 1). It avoids repeated ImageNet training by creating a pre-trained weight-sharing super-net. By sampling different paths in the super-net, DetNAS can select the best child architecture without training. However, the weight-sharing diagram creates a vast gap between the performance of sampled child networks and the fully training result [36]. Thus the searching results may not reflect the true performance of the architectures (minor improvement compared to random search). Moreover, the pre-defined and fixed super-net greatly limits the backbone's search space (only changing of kernel size) while ignoring other key factors of backbones such as the number of blocks, downsampling rate and channel sizes. In contrast, our work aims to develop an efficient and flexible NAS scheme to find an optimal data-specific backbone for object detection.

In this work, we propose a novel architecture search framework named SP-NAS towards a flexible and data-oriented detection backbone. By observing the state-of-the-art design of the backbone detection, two components are important: a) proper allocation of computation on different backbone stages and proper receptive fields and spatial resolution for each feature level (e.g. extra stage and higher spatial resolution in DetNet [9]); b) better fusion of high-level and low-level features (e.g. repeated fusion across multi-resolution sub-networks in [42]). Thus, we propose a serial-to-parallel searching strategy: 1) the serial-searching phase aims at finding a sequence of serial blocks with optimal resolution, receptive fields and output channels in each stage of the feature hierarchy; 2) the parallel-searching phase then assembles several searched serial architectures into a more powerful parallel-structured backbone with a better fusion of high-level and low-level features.

Inspired by recent advances in network morphism [43, 10], we adopt a "swap-expand-reignite" search strategy in the serial-level searching phase which allows continuous growth of the backbone until reaching the optimal design. At each iteration, we apply a set of modifications including "swap" and "expand" to the current network and train the resulting networks in parallel on multiple computing nodes with short training time. Then we choose the best modification among the nodes with the biggest improvement of performance. Both operations follow the network morphism. The backbone can keep training and growing while avoiding the repeated pretraining on ImageNet. We also find that when the morphed architecture deviates far from the original ImageNet pretrained backbone, the algorithm may get stuck with no further improvement of the performance. Thus, we use a "reignite" strategy by re-training on ImageNet for the stuck architecture to reignite the searching process. We found this "swap-expand-reignite" search algorithm works well and keeps increasing the detection performance efficiently and moves to an optimal serial backbone usually with only 1˜2 rounds of ImageNet pretraining.

For the parallel-level searching phase, we aim at finding an optimal parallel-level structure with the different number of subnets along with each backbone stage for a better fusion and extraction of high-level and low-level features. Since the search space is relatively small, we use random search with a resource-constraint sampling.

Extensive experiments are conducted on the widely used detection benchmarks, including Pascal VOC [11], COCO [21], BDD [47] and ECP [3]. The proposed method outperforms current state-of-the-art detection methods, i.e., achieving state-of-the-art performance on the automotive detection EuroCityPersons leaderboard. We also observe consistent performance gains over the hand-crafted backbone structure with the same setting of block structures. In particular, our method outperforms NAS-FPN by 2.5% mAP with less FLOPS and achieves higher performance with similar inference time than Cascade RCNN on COCO, and reaches 84.1% AP50 on VOC better than DetNAS and Auto-FPN in accuracy and speed.

2. Related Work

Object Detection. Modern anchor-based detection approaches such as Faster RCNN [34] and FPN [19] usually consist of several modules: backbone, feature fusion neck, region proposal network, and RCNN head. Most of the works on object detection directly use the backbone network designed and pretrained for the ImageNet classification task, e.g., ResNet [15]. Since the performance of object detectors highly relies on features extracted by backbones, recent advances focus on developing a specific backbone for detection. For example, DetNet [18] tries to develop a backbone for detection by using a higher spatial resolution in the later stage and a dilated bottleneck. However, those manually designed backbones are not data-specific (e.g. backbone designed for COCO may be sub-optimal for other datasets) and require great human expert's labors.

Neural Architecture Search. The goal of NAS is to automatically find an optimal network architecture and release humans from this tedious network architecture engineering. Most previous works [25, 5, 23, 39, 45] search basic CNN architectures for a classification model while a few of them focus on more complicated high-level vision tasks such as semantic segmentation and object detection [8, 22, 9, 12]. Baker and Zoph et al. [1, 49, 4, 48] apply reinforcement learning to train an RNN policy controller to generate a sequence of actions to specify a CNN architecture, which requires massive samples to converge. Real and Liu et al. [33, 24] try to "evolve" CNN architectures by mutating the current best architectures. Liu, Xie and Cai et al. [25, 45, 5] try to introduce architecture parameters

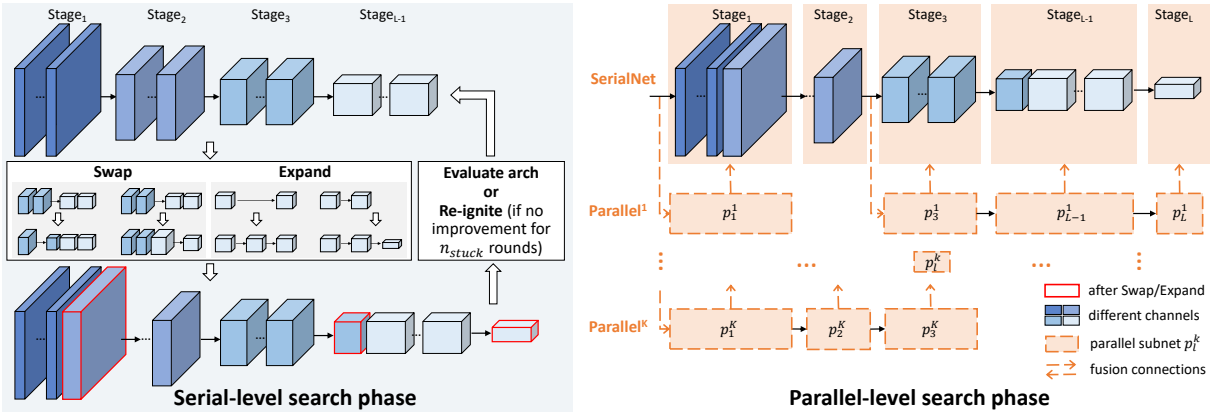

Figure 1. An overview of our SP-NAS towards a flexible and data-oriented backbone. Our algorithm can be decoupled into two phases. a) For serial-level search, we adopt a "swap-expand-reignite" strategy which allows continuous growth of the backbone until reaching an optimal design. The output model of this phase is called SerialNet. b) Parallel-level search aims at finding an optimal parallel-level structure with the different number of subnets p_l^k along with each stage in SerialNet for a better fusion and extraction of multi-scale features.

for continuous relaxation of the discrete search space, thus allowing weight-sharing and differentiable optimization of the CNN architectures. However, those classification based search methods do not generalize well to object detection due to the requirement of computationally intensive ImageNet pretraining.

For the detection task, Zoph et al. [49] transfer the searched architecture from classification to the detector backbone which cannot guarantee optimal adaptation to any detection dataset. NAS-FPN [12] and Auto-FPN [46] focus on improving the feature fusion neck by NAS to find a better feature fusion from different feature levels while ignoring the backbone. DetNAS [9] searches for better backbones for detection on a pre-trained weight-sharing super-net with an evolutionary search algorithm. On the contrary, our work aims to develop an efficient and flexible NAS scheme to find an optimal data-specific backbone for object detection.

3. SP-NAS Pipeline

We propose a serial-to-parallel searching pipeline: 1) serial-level searching aims at finding a sequence of serial blocks with an optimal depth, resolution, receptive fields and output channels in each stage of the feature hierarchy. 2) parallel-level searching then assembles several searched stage architectures (parallel subnets) to a more powerful parallel backbone with better refinement and fusion of different semantic level features. An overview of our SP-NAS is shown in Figure 1. Our SP-NAS algorithm can be used to improve any detectors. Moreover, we adopt a "swap-expand-reignite" search strategy to explore novel backbone architectures without repeatedly pretraining on ImageNet.

3.1. Serial-level Search

The commonly used backbones of the detection network include VGG [37], ResNet [15], and ResNeXt [44]. Those networks are mainly designed for ImageNet classification

Basic Block (BB)	Bottleneck Block (BNB)	ResNeXt Block (XB)
conv3x3(In=n,out=n)	conv1x1(In=4n,out=n)	conv1x1(In=4n,out=2n)
conv3x3(In=n,out=n)	conv3x3(In=n,out=n)	conv3x3(In=2n,out=2n,group=32)
	conv1x1(In=n,out=4n)	conv1x1(In=2n,out=4n)
+ residual connection	+ residual connection	+ residual connection

Table 2. Detailed structures of our chosen blocks.

and thus it is sub-optimal to localize multiple objects at various scales in the feature hierarchy. Generally speaking, high-level feature layers strongly respond to entire objects thus are more discriminative while the low-level feature is more likely to be activated by local textures and patterns, and contains more spatial information. Thus, most hand-crafted detection-specific backbone networks [38, 42, 18] follows a serial structure and try to leverage and preserve the spatial information by adding dilation and a higher spatial resolution in the later stage (DetNet [18]), or preserving and refining features of different resolutions (FishNet [38]). In this work, we propose a serial-level backbone search space that contains a sequence of serial blocks with a flexible setting of blocks, resolution and output channels in each stage of the feature hierarchy.

Search Space. As shown in Figure 1, the serial-level backbone search space consists of a sequence of blocks. Each part of the backbone could be divided into several stages according to the resolution of the output features where stage refers to a number of blocks fed by the features with the same resolution. In this paper, we consider three kinds of blocks: basic block, bottleneck block in ResNet [15] and ResNeXt block [44] as shown in Table 2. The number of blocks in the backbone varies from 8 to 60. The number of stages can be chosen from 5 to 7 and each stage is gradually down-sampled with factor 2. We allow a different number of blocks in each stage thus the allocation of computation is flexible. Firstly, the input image is feed into a stem architecture following [16, 42], which consists of two 3×3 convolutions (stride=2) decreasing the resolution to $\frac{1}{4}$,

Algorithm 1: Serial_to_parallel Search

Input: Stop condition n_{stuckS}, n_{stuckP}, number of reignition n_r
Baseline model f_0 & ImageNet pretrained weight θ_0

$acc_{best}, \theta_{best} \leftarrow train\_eval(f_0, \theta_0)$ #initialize network
Serial_level: **While** $i < n_r$ **do**
  $count_f \leftarrow 0$
  While $count_f < n_{stuckS}$ **do** #network morphism
    $f, \theta_{init} \leftarrow swap(f_{best}, \theta_{best})$
    $f, \theta_{init} \leftarrow expand(f, \theta_{init})$
    $acc, \theta \leftarrow train\_eval(f, \theta_{init})$
    if $acc > acc_{best}$ **then** #update best network
      $acc_{best}, f_{best}, \theta_{best} \leftarrow acc, f, \theta$
      $count_f \leftarrow 0$
    end
  end
  $\theta_{best} \leftarrow reignite(f_{best})$
end
Parallel_level: $count_p \leftarrow 0$
$p_{best} \leftarrow init(f_{best})$
$acc_{best}, \omega_{best} \leftarrow train\_eval(p_{best}, \theta_{best})$
While $count_p < n_{stuckP}$ **do**
  $p \leftarrow$random search
  $acc, \omega \leftarrow train\_eval(p, \omega_{best})$
  if $acc > acc_{best}$ **then** #update best network
    $acc_{best}, p_{best}, \omega_{best} \leftarrow acc, p, \omega$
    $count_p \leftarrow 0$
  end
end
Output: p_{best}, θ_{best}

and the number of base channels in main body is equal to the number of output channels of the stem. In the original ResNet, the channel size is doubled at each downsampling block. However, since the channel size will greatly affect the feature representation capability and computation cost, we also search for the position where the channel size is doubled in the whole sequence.

A complete architecture is encoded like "BNB(11,1d1,111,1d11,11d1)". The first placeholder encodes the block setting (BB: basic block, BNB: bottleneck block, XB: ResNeXt block). "," separates each stage with different resolution. "1" means regular block with no change of channels while "d" indicates the number of base channels is doubled in this block. For example, ResNet50 can be encoded as "BNB(111,d111,d11111,d11)".

Swap-Expand-Reignite Search Strategy. Since we change the architecture of the backbone during the search, we cannot use the ImagNet [35] pretrained model. Most of the detection models require the initialization from the ImageNet pretrained models during training. Although He et al. [13] have shown that ImageNet pretraining is not indispensable for detection, a much longer training ($11_\times$) is required as a compensation, which makes it computationally difficult to search on the backbone directly.

Inspired by recent advances on network morphism [43, 10], we adopt a "swap-expand-reignite" search strategy in the serial-searching phrase to avoid the repeated training

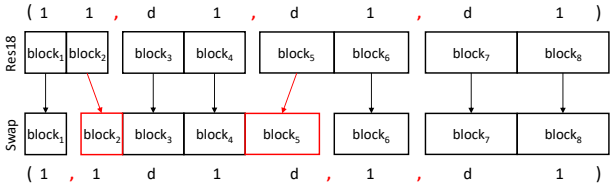

Figure 2. Illustrating the "swap" operation on ResNet18. The number of blocks in the second stage increases from 2 to 4 after the "swap". The previous model's weights are reused in the new architecture.

while keeping the "effect of ImageNet pretraining". At each search step, we apply a set of modifications including "swap" and "expand" to the current network $\mathbf{f}$.

As shown in Figure 1, for the "expand" operation, we add a new block between two existing blocks by IdMorph [43] while the number of input/output channels remains the same. The weight of the convolution in the new block is initialized as an identity matrix. For the "swap" operation, we just interchange the stride of the neighboring block while keeping the weight unchanged, as shown in Figure 2. Both modifications on the current network will maintain the output unchanged as much as possible at initialization, which keeps the "effect of ImageNet pretraining". Starting from a commonly used network with ImageNet pretraining, the backbone network can grow bigger and explore a wide range of architectures in the search space by iteratively "expand" and "swap" on the current network.

Detailed "swap-expand-reignite" search strategy can be found in Algorithm . Starting from a small base network (such as ImageNet pretrained ResNet or ResNeXt), we apply several sets of modifications including "swap" and "expand" to the current network at each iteration, and train the resulting networks on multiple nodes with a short training time (e.g. 3 epochs). Then we evaluate those architectures and choose the best-updated model by the most significant improvement of performance as the current model. In practice, we find that when the current architecture deviates from the original ImageNet pretrained backbone too much, the iterative search algorithm may get stuck at finding no modification that can further improve the performance. As a result, we use a "reignite" strategy by pre-training the ImageNet for the current stuck architecture again to re-ignite the searching process. We define the "stuck" status as repeated trying for n_{stuck} times with no further improvement. Empirically, we found this "swap-expand-reignite" works well and keeps increasing the performance of the detector efficiently and moves to a local optimal serial backbone with only 1~2 rounds of ImageNet pretraining (i.e., in Algorithm 1, n_r can be set to 1 or 2).

This algorithm greatly reduces the times of ImageNet pretraining while it remains a flexible search space and can run in parallel on multiple computing nodes efficiently. The output architecture of serial-level search is called SerialNet.

%	Method	Backbone	Input Size	train epoch	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Inf Time (fps)	FLOPS (G)
COCO	DetNet [18]	DetNet-59	800 × 1333	24	37.9	60.1	41.2	22.7	41.2	48.3	-	292.9
	FishNet [38]	FishNet-150	800 × 1333	24	40.6	-	-	23.3	43.9	53.7	-	294.9
	HRNet [42]	HRNet-W40	800 × 1333	24	41.6	62.5	45.6	23.8	44.9	53.8	12.0 (V100)	380.7
	Cascade RCNN [6]	ResNet101	800 × 1333	24	42.8	62.1	46.3	23.7	45.5	55.2	10.2 (V100)	323.3
	TridentNet [17]	ResNet101	800 × 1333	24	42.7	63.6	46.5	23.9	46.4	55.6	1.7 (V100)	-
	CBNet w Cascade [27]	ResNet101-TB	800 × 1333	24	44.9	63.9	48.9	-	-	-	5.5 (V100)	860.2
	DetNAS [9]	DetNAS	800 × 1333	24	40.0	61.5	43.6	23.3	42.5	53.8	-	289.4
	AmoebaNet w FPN [31]	AmoebaNet	1280 × 1280	50	43.4	-	-	-	-	-	4.7 (P100)	655.5
	NAS-FPN (7@384) [12]	AmoebaNet	1280 × 1280	150	48.0	-	-	-	-	-	3.6 (P100)	1317
	$SPNet_{COCO}$(BNB)	SPNet-BNB	800 × 1333	24	45.6	64.3	49.6	28.4	48.4	60.1	10.2 (V100)	391.1
	$SPNet_{COCO}$(XB)	SPNet-XB	800 × 1333	24	47.4	65.7	51.9	29.6	51.0	60.4	5.6 (V100)	654.5
	$SPNet_{COCO}$(XB)	**SPNet-XB**	1280 × 1280	50	**49.1**	**67.1**	**53.5**	**31.0**	**52.6**	**63.7**	2.1 (V100)	949.0

Table 3. Comparison of mAP and inference time of the state-of-the-art hand-crafted networks and NAS for detectors on COCO. Our searched $SPNet_{COCO}$ backbone stacked on Cascade RCNN outperforms hand-crafted and automatically designed backbones with similar or faster inference speed. The bold result is based on longer training and additional mask supervision.

	Method	LAMR				mAP
		Reasonable	Small	Occluded	All	
ECP	ResNet50 w FPN [19]	0.088	0.193	0.337	0.228	88.4
	ResNet101 w FPN [19]	0.089	0.194	0.340	0.226	87.8
	HRNet-W40 [42]	0.067	0.132	0.272	0.181	88.1
	HRNet-W18 [42]	0.067	0.132	0.284	0.187	88.7
	$SPNet_{ECP}$	**0.054**	**0.110**	**0.252**	**0.165**	**89.6**

Table 4. Comparison of LAMR and mAP of the state-of-the-art single-model on ECP. LAMR is the official metric (smaller is better). Our searched model currently reaches TOP 1 performance on the ECP leaderboard (see the dataset website).

%	Method	Input Size	AP	AP_{50}	AP_{75}	Inf Time (fps)
BDD	ResNet50 w FPN [19]	1920 × 1024	36.3	61.2	36.8	12.4
	ResNet101 w FPN [19]	1920 × 1024	37.1	61.3	37.9	9.2
	Auto-FPN [46]	1920 × 1024	33.9	-	-	3.1
	$SPNet_{BDD}$	1920 × 1024	38.7	63.2	39.2	3.7
VOC	ResNet18 w FPN [19]	600 × 1333	-	77.6	-	30.4
	Auto-FPN [46]	600 × 1333	-	81.8	-	10.6
	DetNAS [9]	600 × 1333	-	81.5	-	-
	$SPNet_{VOC}$	600 × 1333	-	84.1	-	19.9

Table 5. Comparison of Average Precision and frames per second (fps) during inference on BDD and PASCAL VOC. $SPNet_{BDD}$ starts from ResNet50 with bottleneck block and $SPNet_{VOC}$ is based on ResNet18 with basic block.

3.2. Parallel-level Search

Recently, more works focus on a parallel structure stacked on the original backbone and aiming at better utilizing and fusing the information from all feature levels with different resolutions in the backbone feature hierarchy. This is due to the limit of the information barrier between different stages in serial-kind structure. For example, PANet [26] enhances the entire feature hierarchy with accurate localization signals in lower layers by a parallel bottom-up path augmentation. HRNet [42] combines multiple parallel high-to-low resolution subnets with repeated information exchange across multi-resolution subnetworks. To pursue a high-performance backbone structure, we consider a parallel-level search for the subnet structures based on the searched backbone SerialNet in the serial-level search as shown in Figure 1.

Search Space. The parallel-level backbone is stacked on the result (i.e., SerialNet) in serial-level search. The search space includes a series of subnets along with each stage of the searched backbone architecture. Initialization of each subnet is the copy of the corresponding stage of the trained SerialNet. The parallel-level structure enables better feature extraction and fusion.

As shown in Figure 1, the SerialNet backbone comprises L stages: s_1 to s_L the l-th stage s_l takes the output (denoted as x_{l-1}) of the previous s_{l-1} stage as input, which can be expressed as: $x_l = s_l(x_{l-1})$. For each stage s_l, we consider adding a series of parallel subnets p_l^1 to p_l^K to enhance the features representation. To be more specific, the output features of the previous stage is feed into p_{i1} to p_{ik} iteratively in a recurrent way. Taken x_{l-1} as the output feature of the previous stage, the recurrent operation of the subnets p_l^1 to p_l^K can be written as $x_l^k = p_l^k(x_{l-1}^k + upsample(x_l^{k-1}))$ where $k = 1, ..., K$, $x_l^0 = x_l$, $p_l^0 = s_l$, and $upsample(\cdot)$ consists of a 1x1 conv and an upsampling operation to constrain the channels and the resolutions to be consistent with x_{l-1}. As a result, the output features of the subnet p_l^k becomes the input of p_l^{k-1} at the same layer by the $upsample(\cdot)$. For the network structure inside the subnets, we directly copy the corresponding stage found by our SerialNet to the subnets, e.g. $p_l^k = s_l$. Note that we also copy the weight parameters from s_l. Thus we can make fully use of both the training of detection dataset and the ImageNet pretraining in the previous phase.

The original input feature maps x_{l-1} at the l-th stage will go through multiple subnets for better feature extraction. Moreover, by going through multiple $upsample(\cdot)$

	BDD (mAP)	ECP (LAMR)	VOC (AP50)
baseline	36.5	0.088	77.1
serial-level round1	$37.2^{+0.7}$	$0.064^{-0.024}$	$81.2^{+4.1}$
serial-level round2	$38.0^{+0.8}$	$0.061^{-0.003}$	$83.5^{+2.3}$
parallel-level	$\mathbf{38.7}^{+0.7}$	$\mathbf{0.054}^{-0.007}$	$\mathbf{84.1}^{+0.6}$

Table 6. Ablation studies during the serial-to-parallel search. The serial-level search is under two rounds of reignition. The BDD and ECP are searched from ResNet50 and use mAP and LAMR for evaluation respectively.

and downsampling in the subnet, the parallel-level search space enables a complete fusion of different level features. We encode parallel-level search space as a list consisting of the number of subnets for each stage. For example, if the SerialNet has 4 stages, "(0,2,1,3)" indicates the number of parallel subnets for each stage, e.g. "0" denotes no additional subnet in this stage. "3" means we copy the last stage 3 times to become the subnets of the final backbone.

Search Strategy. The parallel search space is relatively small with comparison to the one in the serial search: total $(K+1)^L$ of unique combinations where K is the maximum number of subnets ($K = 3$ in this paper) and L is the number of stages found by the previous round of search (usually 4~6). Since we reuse the weights of the backbone in serial-level search as the initialization of the parallel subnets, the training of the parallel-level backbone is efficient. Thus, in the parallel-level search, we just use a random search with a resource-constraint sampling to find the optimal combination of the parallel subnets. The sampling of the encoding is not uniform. The probability of adding one additional subnet is proportional to the inverse of FLOPS for each subnet which avoids adding too heavy subnets in the backbone.

At the end of the parallel-level search, the best performance architecture is named SPNet.

4. Experiments

Datasets and Evaluation Metrics. To evaluate our methods on different domains, we conduct architecture search on PASCAL VOC [11], COCO [21], ECP [3], BDD [47] for common object detection and autonomous driving detection. **COCO** [21] is a common object detection dataset containing 80 classes with 118K images for training, 5K for evaluation. **PASCAL VOC (VOC)** [11] contains 20 object classes. Training data is the union of VOC 2007 trainval and VOC 2012 trainval (10K images) and evaluation is on VOC 2007 test (4.9K images). We only report mAP scores using IoU at 0.5 as common practice. **EuroCity Persons Dataset (ECP)** [3] is an autonomous driving dataset for pedestrian detection. ECP contains about 24K training images and 4.3K validation images. **Berkeley Deep Drive (BDD)** [47] is an autonomous driving dataset with 10 object classes. BDD contains about 70K images for training and 10K for evaluation.

For the evaluation metrics, we adopt the metrics from COCO detection evaluation criteria [21] which is mean Average Precision (mAP) across IoU thresholds from 0.5 to 0.95 with an interval of 0.05 and different scales (small, medium, big). Additionally, in order to compare the performance on the ECP leader-board, we use the Log Average Miss-Rate (LAMR) with ignoring regions under different cases (reasonable, small, occluded, all) [3] as: $LAMR = exp(\frac{1}{9}\sum_f log(mr(argmax_{fppi(c)\leq f} fppi(c))))$, where $fppi(c)$ is the number of false positives per image under confidence level c, $mr = 1 - recall(c)$. Thus, the lower LAMR the better performance of the detector.

4.1. Implementation Details & Intermediate Results

Implementation Details for Serial-level Search. For each dataset, we choose an ImageNet pretrained model for the starting architecture, VOC: ResNet18 [15], BDD and ECP: ResNet50, COCO: ResNet50 and ResNeXt101 [44]. To propose a new architecture, we adopt 3 random modifications ("swap" and "expand") on the current backbone architecture as well as the weight initialization and feed the new architecture into a "fast training scheme". This training scheme only trains the network for 3 epochs with SGD optimizer with cosine decay learning rate 0.02 to 0.0001, momentum 0.9, batch size$= 2 \times 8$ and 10^{-4} as weight decay. All the backbone parameters are updated during the training phase. We train and test the new architectures in parallel on 4 computing nodes, and each has 8 Nvidia V100 GPU cards. Empirically, we found that training with 3 epochs already can separate good models from bad models. It takes about 3 hours on COCO, 4 hours on BDD, 1 hours on ECP and 0.5 hours on VOC to finish evaluating one architecture. n_r is set to be 2 and $n_{stuck} = 20$. Since the whole architecture is continuously growing, the search space is very big, containing more than 10^{20} possible paths.

Implementation Details for Parallel-level Search. During parallel-level search, we first use the SerialNet obtained at serial-level search as the base backbone and add RPN and RCNN head on the backbone as a common two-stage detector. Then we randomly sample a parallel-level encoding and construct the subnets structure. Each subnet is initialized by the corresponding weights from the SerialNet. We also use the same "fast training scheme" to obtain an estimate of the accuracy on validation. Since the model is very big, we use a batch size 1×8 on 8 GPUs (one image per GPU). We allow the maximum number of subnets $K = 3$. The search space is relatively small, containing less than 4×10^3 unique candidates.

Analysis of Intermediate Results. Figure 3 shows the performance trajectories (blue line) on the ECP dataset during serial-level search and parallel-level search. It can be found that both searching rounds can consistently boost performance by discovering new architectures. Since the

Dataset	block type	Start backbone	Serial-level	Parallel-level
VOC	BB	ResNet18 (11, d1, d1, d1)	(11d, 1111, d111111111, d111, 1, 1111)	(0, 0, 0, 0, 0, 1)
ECP	BNB	ResNet50 (111, d111, d11111, d11)	(111d1111, d1111, 11d, 1111, 1111)	(0, 2, 1, 1, 0)
BDD			(1111111111d, 1111111111111111, d1111111111111111, d111111, 111, 1)	(0, 1, 2, 3, 0, 0)
COCO			(1111d1, 11111111d, 11, 1111111111111111d, 111111111111, 1)	(0, 0, 0, 1, 0, 0)
COCO	XB	ResNeXt101_32x4d (111, d111, d11111111111111111111111, d11)	(1111, d1111, d1111111111111111111111111, d11, 1)	(1, 0, 2, 1, 2)

Table 7. The detailed backbones searched by SP-NAS on ECP, BDD, COCO and VOC.

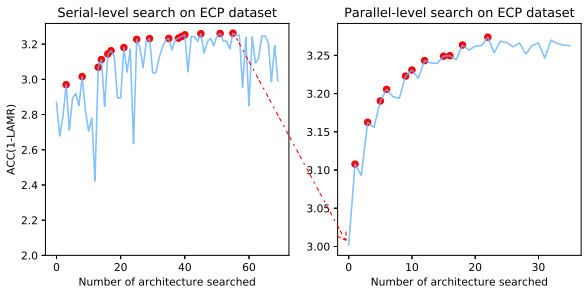

Figure 3. The performance trajectories during serial-level search and parallel-level search on the ECP dataset. The red points are the current best models with highest performance during search. Our search algorithm can consistently boost the performance in both phases.

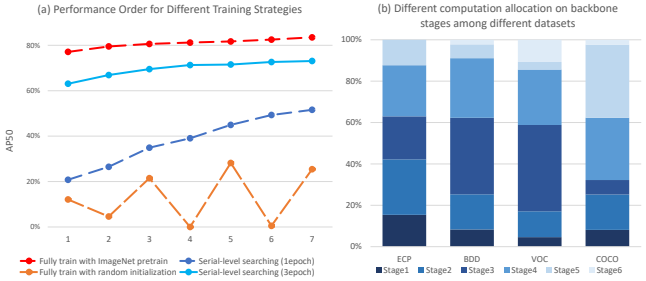

Figure 4. (a) Comparison of the performance between fully training with ImageNet pretraining and our serial-level searching on VOC. The solid blue line is actual serial-level searching trajectories with a fast training scheme. Each point on the line is the local optimal architecture during each searching round. We fully train those architectures with ImageNet pretrained/random initialization. (b) The computational allocation (FLOPS) on the backbone stages of the searched architectures. The COCO with more complex objects needs more computation on the larger receptive fields, while others focus on the preceding stages.

network morphism in serial-level search is also a weight-sharing method, we want to examine whether the performance ranking found by our search method is consistent with the fully training with ImageNet pretrained backbone models. Figure 4 (a) shows the performance of our serial-level searched models compared with fully training on VOC. It can be found that our searching method can maintain the same performance ranking with the standard training scheme with ImageNet initialization. Fully training with random initialization can only yield confusion results. Thus our "swap-expand" strategy is a good proxy to the detection task and avoids huge computation resources of repeated training.

Ablation Study. Intermediate results during the parallel-level phase and serial-level phase are shown in Table 6. For the BDD dataset (starting from ResNet50), mAP is used for evaluation and the serial-level search is under two rounds of reignition. AP50 is used for evaluating VOC (starting from ResNet18) and LAMR is used for ECP (starting from ResNet50). We can find that the first reignition has the greatest improvement for VOC and BDD. The results show continuous improvement with further reignition.

4.2. Object Detection Results

After identifying the optimal architecture on each dataset, we fully train the model since the "fast training scheme" only has trained 3 epochs on that architecture. Since the model is already very big, we can only train it with one image per GPU with SGD. For all the datasets, we train 24 epochs if not otherwise stated. The initial learning rate is 0.02, and reduces two times ($\times 0.1$) during fine-tuning; 10^{-4} as weight decay; 0.9 as momentum. During training, image flipping is used for data augmentation. Only for COCO, $0.8_\times \sim 1.2_\times$ multi-scale training is used. During testing, **multi-scale testing is not used**. Pixel size=800×1333 is used for COCO (VOC: 600×1333, ECP: 1920×1024, BDD: 1920×1080).

Detailed Final Searched Architectures. Table 7 shows the detailed architecture of the final searched models for VOC, COCO, ECP and BDD. Unlike hand-crafted ResNet/ResNeXt and automatic searched DetNAS series improve network capability for different tasks by simply stacking blocks in the same backbone stage or increasing width over the whole network, the proposed SP-NAS can grow a network for task-specific feature level and proper receptive fields. We found that for ECP, all raising channels happen at early stages which means that lower-level feature plays an important role for localization. For the parallel-level structure, it can be found that the output feature from stage1 is important for ECP and BDD thus more subnets are added in the following stages. For a thorough understanding, we illustrate the proportion of computation allocation (FLOPS) of the searched SPNets on different backbone stages in Figure 4 (b). More computation power is al-

Method	ImageNet Cls		Det w FPN on COCO
	FLOPs	Accuracy	mAP
ResNet50 [15]	3.8G	76.1	36.5
ResNet101 [15]	7.6G	77.4	39.4
DetNAS [9]	1.3G	77.2	40.0
DetNet-59 [18]	4.8G	76.5	40.2
FishNet-150 [38]	6.8G	78.1	40.6
HRNet-W40 [42]	11.8G	78.9	41.6
SPNet*$_{COCO}$(BNB)	**9.5G**	**79.1**	**41.7**

Table 8. Comparison of performance on ImageNet Classification task and COCO object detection task with existing backbone networks. The FPN detector is used for all backbone networks. * means only serial-level search.

located to stage5 and stage6 with a larger recptive field due to complex scene and category scales in COCO. However, for less categories and autonomous driving scenes, feature representation in early stages (stage1-3) is more important.

Comparison with the State-of-the-art. To measure the inference speed, we run 2000 testing images on one V100 GPU and take the last 1000 images to compute the average of the inference time for comparison. Tables 3, 4 and 5 show the results of the architecture searched from COCO, ECP, BDD and VOC by SP-NAS. SPNet achieves significant gains than the baseline FPN on all detection benchmarks. In Table 3, we make a detailed comparison with existing hand-crafted detectors which most focus on backbone design: DetNet [18], FishNet [38], HRNet [42], Cascade RCNN [6], TridentNet [17], CBNet [27] and NAS for detectors: DetNAS [9], NAS-FPN [12] and AmoebaNet [31]. The inference time is tested on single V100 GPU (some models marked with other GPU devices follow the original papers). We list those methods with reported results without multi-scale testing and the reported SPNet$_{COCO}$ is based on Cascade RCNN. Our searched models dominate most SOTA models on speed/FLOPS and accuracy trade-off. For example, SPNet$_{COCO}$(BNB) performs 2.8% better than Cascade RCNN [6] with the same inference time and outperforms NAS-FPN [12] with less FLOPS and training epochs. It is worth noting that simply increasing network depth from ResNet50 to ResNet101 has 0.6% mAP drop on ECP in Table 4, and extending network width from HRNet-W18 to HRNet-W40 is also useless. This reflects the backbone gap between common detection and automotive datasets. In contrast, our SPNet$_{ECP}$ achieves 0.067 LAMR compared to 0.088 on ECP [3] validation set and achieves SOTA on the final leaderboard. Table 5 shows the searched SPNet$_{BDD}$ is faster than Auto-FPN [46] with better performance. SPNet$_{VOC}$ achieves 84.1% AP50 compared to 77.6% on VOC.

Comparison with Other Backbone Networks. Table 8 shows the comparison of FLOPS and accuracy on classification and detection tasks with other backbones. SPNet*$_{COCO}$(BNB) is the result from the serial-level phase, which already achieves superior performance both on ImageNet and COCO than common-used backbone ResNet and other automatic/handcrafted designed detection backbone networks. In terms of latency and mAP, our SPNet dominates others in Figure 5 (a). We further compare our SPNet and common-used backbones with different detectors: Faster RCNN with FPN [19], Cascade RCNN [6] and Cascade Mask RCNN [14] in Figure 5 (b). The searched SPNet can boost far more performance than hand-crafted backbone networks ResNet [15], ResNeXt [44] and HRNet-w32/48 [42] by replacing different detectors, and surpass them in terms of FLOPS/mAP.

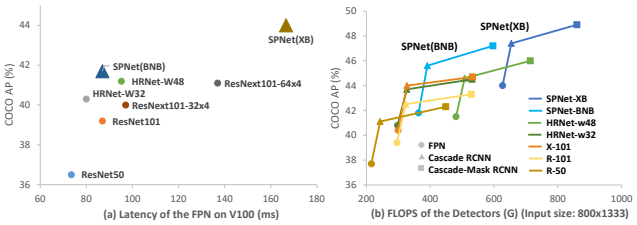

Figure 5. (a) Comparison of mAP vs. Latency on different common backbones with FPN on COCO. SPNet(BNB) is our searched network with the bottleneck block, XB: ResNeXt block. (b) Comparison of different backbones with several advanced detectors.

Comparison of search efficiency with other NAS methods. The searching time of our method and other NAS methods can be found in Table 9. We randomly sample 10 networks as the baseline, which has the same number of blocks and stages with SPNet$_{VOC}$ and is trained with ImageNet pretraining. The results show that our method is much more efficient than random sample by avoiding the repeated pretraining and can find better architectures with much less time than DetNAS [9] and NAS-FPN [12].

Search Method	#Searched Architecture	Total Search Time (GPU days)	Average/Best mAP of searched arch
Random	10	~76	76.4/80.3
Auto-FPN [46]	1	~0.8	81.8
NAS-FPN [12]	8000	$\gg$100	82.1
DetNAS [9]	1000	~68	81.5
SP-NAS	500	~26	**83.5**

Table 9. Comparison on NAS search efficiency on VOC.

5. Conclusion

In this paper, we propose a novel serial-to-parallel NAS search (SP-NAS) pipeline towards a flexible and task-oriented detection backbone. The search is efficient by reusing the weights during iterative training and keeps consistent performance ranking of searched networks with fully training models. The searched SPNet networks achieve state-of-the-art accuracy/speed trade-off on multiple detection benchmarks.

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 2

[2] Chandrasekhar Bhagavatula, Chenchen Zhu, Khoa Luu, and Marios Savvides. Faster than real-time facial alignment: A 3d spatial transformer network approach in unconstrained poses. In *ICCV*, 2017. 1

[3] Markus Braun, Sebastian Krebs, Fabian B. Flohr, and Dariu M. Gavrila. Eurocity persons: A novel benchmark for person detection in traffic scenes. *TPAMI*, 2019. 1, 4, 4, 4.2

[4] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018. 2

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 2

[6] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. 3.1, 4.2, 4.2

[7] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Celine Teuliere, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *CVPR*, 2017. 1

[8] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *NeurIPS*, 2018. 2

[9] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019. 1, 1, 2, 3.1, 3.2, 4.2, 4.2, 4.2

[10] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017. 1, 3.1

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, June 2010. 1, 4

[12] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 1, 1, 2, 3.1, 4.2, 4.2

[13] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, 2018. 3.1

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, pages 2961–2969, 2017. 4.2

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 1, 2, 3.1, 4.1, 4.2, 4.2

[16] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *CVPR*, pages 9145–9153, 2019. 3.1

[17] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. *arXiv preprint arXiv:1901.01892*, 2019. 3.1, 4.2

[18] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. In *ECCV*, 2018. 1, 2, 3.1, 3.1, 4.2, 4.2

[19] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 1, 2, 3.2, 3.2, 4.2

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017. 1

[21] 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. 1, 1, 4, 4

[22] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *arXiv preprint arXiv:1901.02985*, 2019. 2

[23] 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 *ECCV*, 2018. 1, 2

[24] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017. 2

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018. 1, 2

[26] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 3.2

[27] Yudong Liu, Yongtao Wang, Siwei Wang, TingTing Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. Cbnet: A novel composite backbone network architecture for object detection. *arXiv preprint arXiv:1909.03625*, 2019. 3.1, 4.2

[28] Ping Luo, Yonglong Tian, Xiaogang Wang, and Xiaoou Tang. Switchable deep network for pedestrian detection. In *CVPR*, 2014. 1

[29] Will Norcliffe-Brown, Efstathios Vafeais, and Sarah Parisot. Learning conditioned graph structures for interpretable visual question answering. In *NIPS*, 2018. 1

[30] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, pages 1882–1890, 2019. 1

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 1, 3.1, 4.2

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pages 4780–4789, 2019. 1

[33] 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. 2

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 2

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 1, 3.1

[36] Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *NeurIPS*, 2019. 1

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3.1

[38] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. In *Advances in Neural Information Processing Systems*, pages 754–764, 2018. 1, 3.1, 3.1, 4.2, 4.2

[39] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018. 2

[40] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1

[41] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. *arXiv preprint arXiv:1911.09070*, 2019. 1

[42] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *TPAMI*, 2019. 1, 1, 1, 3.1, 3.1, 3.2, 3.2, 4.2, 4.2, 4.2

[43] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *ICML*, pages 564–572, 2016. 1, 3.1

[44] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 3.1, 4.1, 4.2

[45] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019. 1, 2

[46] 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. 1, 1, 2, 3.2, 4.2, 4.2

[47] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018. 1, 4

[48] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018. 2

[49] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2