SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection

Lewei Yao,[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

The state-of-the-art object detection method is complicated with various modules such as backbone, RPN, feature fusion neck and RCNN head, where each module may have different designs and structures. How to leverage the computational cost and accuracy trade-off for the structural combination as well as the modular selection of multiple modules? Neural architecture search (NAS) has shown great potential in finding an optimal solution. Existing NAS works for object detection only focus on searching better design of a single module such as backbone or feature fusion neck, while neglecting the balance of the whole system. In this paper, we present a two-stage coarse-to-fine searching strategy named Structural-to-Modular NAS (SM-NAS) for searching a GPU-friendly design of both an efficient combination of modules and better modular-level architecture for object detection. Specifically, Structural-level searching stage first aims to find an efficient combination of different modules; Modular-level searching stage then evolves each specific module and pushes the Pareto front forward to a faster task-specific network. We consider a multi-objective search where the search space covers many popular designs of detection methods. We directly search a detection backbone without pre-trained models or any proxy task by exploring a fast training from scratch strategy. The resulting architectures dominate state-of-the-art object detection systems in both inference time and accuracy and demonstrate the effectiveness on multiple detection datasets, e.g. halving the inference time with additional 1% mAP improvement compared to FPN and reaching 46% mAP with the similar inference time of MaskRCNN.

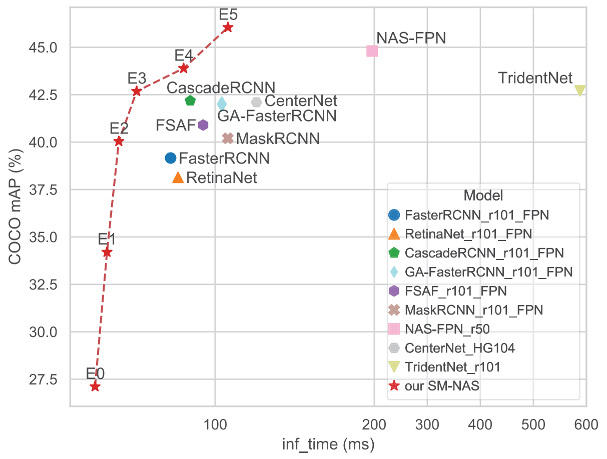

Figure 1: Inference time (ms) and detection accuracy (mAP) comparison on COCO dataset. SM-NAS yields state-of-the-art speed/accuracy trade-off.

Introduction

Real-time object detection is a core and challenging task to localize and recognize objects in an image on a certain device. This task widely benefits autonomous driving, surveillance video, facial recognition in mobile phone, to name a few. A state-of-the-art detection system (Liu et al.; Ren et al. 2016; 2015) usually consists of four modules: backbone, feature fusion neck, region proposal network (in two-stage detection), and RCNN head. Recent progress in this area shows various designs of each modules: backbone (Li et al. 2018), region proposal network (Wang et al. 2019a), feature fusion neck (Liu et al. 2018) and RCNN head (Li et al. 2017).

However, how to select the best combination of modules under hardware resource constrains remains unknown. This problem draws much attention from the industry because in practice adjusting each module manually based on a standard detection model is inefficient and sub-optimal. It is hard to leverage and evaluate the inference time and accuracy trade-off as well as the representation capacity of each module in different datasets. For instance, empirically we found that combination of Cascade-RCNN with ResNet18 (not a standard detection model) is even faster and more accurate than FPN with ResNet50 in COCO (Lin et al. 2014) and BDD (Yu et al. 2018) (autonomous driving dataset). However, this is not true in the case of VOC.

There has been a growing trend in automatically designing a neural network architecture instead of relying heavily on human efforts and experience. For the image classifi-

*Both authors contributed equally to this work.
†Corresponding Author: xdliang328@gmail.com

cation, (Zoph et al.; Liu, Simonyan, and Yang 2018; 2018) searched networks surpass the performance of hand-crafted networks. For the detection task, existing NAS works focus on optimizing a single component of the detection system instead of considering the whole system. For example, DetNAS (Chen et al. 2019a) searches for better backbones on a pre-trained super-net. NAS-FPN (Ghiasi, Lin, and Le 2019), Auto-FPN (Xu et al. 2019), and NAS-FCOS (Wang et al. 2019b) use NAS to find a better feature fusion neck and a more powerful RCNN head. However, those pipelines only partially solve the problem by changing one component while neglecting the balance and efficiency of the whole system. On the contrary, our work aims to develop a multi-objective NAS scheme specifically designed to find an optimal and efficient whole architecture.

In this work, we make the first effort on searching the whole structure for object detectors. By investigating the state-of-the-art design, we found three factors are crucial for the performance of a detection system: 1) size of the input images; 2) combination of modules of the detector; 3) architecture within each module. To find an optimal trade-off between inference time and accuracy with these three factors, we propose a coarse-to-fine searching strategy: 1) Structural-level search stage (Stage-one) first aims to find an efficient combination of different modules as well as the model-matching input sizes; 2) Modular-level search stage (Stage-two) then evolves each specific module and pushes forward to an efficient task-specific network.

We consider a multi-objective search targeting directly on GPU devices, which outputs a Pareto front showing the optimal designs of the detector under different resource constraints. During Stage-one, the search space includes different choices of modules to cover many popular one-stage/two-stage designs of detectors. We also consider putting the input image size into the search space since it greatly impacts the latency and accuracy (Tan and Le 2019). During Stage-two, we further consider to optimize and evolve the modules (e.g. backbone) following the optimal combination found in the previous stage. The previous works (Li et al. 2018) find that backbones originally designed for classification task might be sub-optimal for object detection. The resulting modular-level search thus leans the width and depth of the overall architecture towards detection task. With the improved training strategy, our search can be conducted directly on the detection datasets without ImageNet pre-training. For an efficient search, we combine evolutionary algorithms (Real et al.; Real et al. 2017; 2018) with Partial Order Pruning technique (Li et al. 2019a) for a fast search and parallelize the whole searching algorithm in a distributed training system to further speed up the whole process.

Extensive experiments are conducted on the widely used detection benchmarks, including Pascal VOC (Everingham et al. 2010), COCO (Lin et al. 2014), and BDD (Yu et al. 2018). As shown in Figure 1, SM-NAS yields state-of-the-art speed/accuracy trade-off and outperforms existing detection methods, including FPN , Cascade-RCNN and the most recent work NAS-FPN (Ghiasi, Lin, and Le 2019). Our E2 reaches half of the inference time with additional 1% mAP improvement compared to FPN. E5 reaches 46% mAP with similar inference time of MaskRCNN (mAP:39.4%).

To sum up, we make the following contributions to NAS for detection:

- We are among the first to investigate the trade-off for speed and accuracy of an object detection system with a different combination of different modules.
- We develop a coarse-to-fine searching strategy by decoupling the search into structural-level and modular-level to efficiently lift the Pareto front. The searched models reach the state-of-the-art speed/accuracy, dominating existing methods with a large margin.
- We make the first attempt to directly search a detection backbone without pre-trained models or any proxy task by exploring fast training from scratch strategy.

Related Work

Object Detection. Object detection is a core problem in computer vision. State-of-the-art anchor-based detection approaches usually consists of four modules: backbone, feature fusion neck, region proposal network (in two-stage detectors), and RCNN head. Most of the previous progress focus on developing better architectures for each module. For example, (Li et al. 2018) tries to develop a backbone for detection; FPN (Lin et al. 2017) and PANet (Liu et al. 2018) modify multi-level features fusion module; (Wang et al. 2019a) tries to make RPN more powerful. R-FCN (Dai et al. 2016) and Light-head RCNN (Li et al. 2017) design different structures of bbox head. However, community lacks of literatures comparing the efficiency and performance of different combination of different modules.

Neural Architecture Search. NAS aims at automatically finding an efficient neural network architecture for a certain task and dataset without labor of designing network. Most works are based on searching CNN architectures for image classification while only a few of them (Chen et al.; Liu et al.; Chen et al. 2018b; 2019; 2019a) focus on more complicated vision tasks such as semantic segmentation and detection. There are mainly three categories of searching strategies in NAS area: 1) Reinforcement learning based methods (Zoph et al.; Cai et al. 2018; 2018) train a RNN policy controller to generate a sequence of actions to specify CNN architecture; 2) Evolutionary Algorithms based methods and Network Morphism (Liu et al.; Real et al. 2017; 2018) try to "evolves" architectures by mutating the current best architectures; 3) Gradient based methods (Liu, Simonyan, and Yang 2018) define an architecture parameter for continuous relaxation of the discrete search space, thus allowing differentiable optimization of the architecture. Among those approaches, gradient based methods is fast but not so reliable since weight-sharing makes a big gap between the searching and final training. RL methods usually require massive samples to converge which is not practical for detection. Thus we use EA based method in this paper.

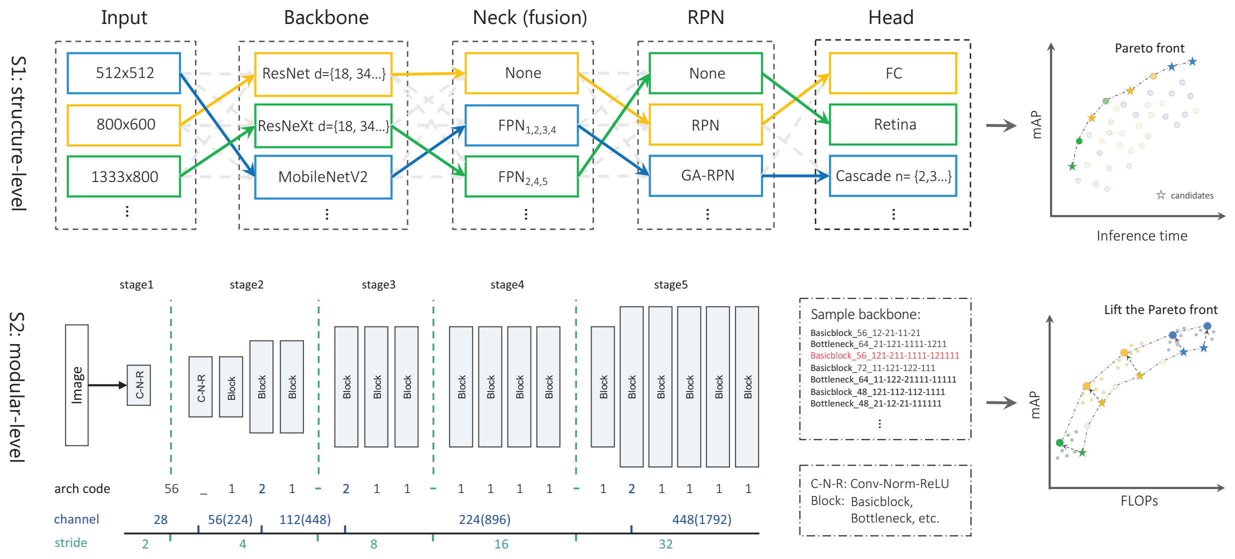

Figure 2: An overview of our SM-NAS for detection pipeline. We propose a two-stage coarse-to-fine searching strategy directly on detection dataset: S1: Structural-level searching stage first aims to finding an efficient combination of different modules; S2: Modular-level search stage then evolves each specific module and pushes forward to a faster task-specific network.

id	Dataset	Model	Backbone	Input Img	Time(ms)	mAP
1	COCO	FPN	ResNet50	800x600	43.6	36.3
2	COCO	RetinaNet	ResNet50	800x600	46.7	34.8
3	VOC	FPN	ResNet50	800x600	38.4	80.4
4	VOC	RetinaNet	ResNet50	800x600	34.8	79.7
5	COCO	FPN	ResNet101	1333x800	72.0	39.1
6	COCO	CascadeRCNN	ResNet50	800x600	54.9	39.3

Table 1: Preliminary empirical experiments. Inference time is tested on one V100 GPU. The performance of a detection model is highly related to the dataset (Exp1-4). Better combination of modules and input resolution can leads to an efficient detection system (Exp 5&6).

The Proposed Approach

Motivation and Preliminary Experiments

With preliminary empirical experiments, we have found some interesting facts:

1) One-stage detector is not always faster than two-stage detector. Although RetinaNet (He et al. 2016) is faster than FPN (Lin et al. 2017) on VOC (Exp 3&4), it is slower and worse than FPN on COCO (Exp 1&2).

2) Reasonable combination of modules and input resolution can lead to an efficient detection system. Generally, Cascade-RCNN is slower than FPN with the same backbone since it has 2 more cascade heads. However, with a better combination of modules and input resolution, Cascade-RCNN with ResNet50 can be faster and more accurate than FPN with ResNet101 (Exp 5&6)

It can be found that customizing different modules and input-size is crucial for real-time object detection system for task specific datasets. Thus we present the SM-NAS for searching an efficient combination of modules and better modular-level architecture for object detection.

NAS Pipeline

As in Figure 2, we propose a coarse-to-fine searching pipeline: 1) Structural-level searching stage first aims to find an efficient combination of different modules; 2) Modular-level search stage then evolves each specific module and pushes forward to a faster task-specific network. Moreover, we explore a strategy of fast training from scratch for the detection task, which can directly search a detection backbone without pre-trained models or any proxy task.

Stage-one: Structural-level Searching Modern object detection systems can be decoupled into four components: backbone, feature fusion neck, region proposal network (RPN), and RCNN head. We consider putting different popular and latest choices of modules into the search space to cover many popular designs.

Backbone. Commonly used backbones are included in the search space: ResNet (He et al. 2016) (ResNet18, ResNet34, ResNet50 and ResNet101), ResNeXt (Xie et al. 2017a) (ResNeXt50, ResNeXt101) and MobileNet V2 (Sandler et al. 2018). During Stage-one, we loaded the backbones pre-trained from ImageNet (Russakovsky et al. 2015) for fast convergence.

Feature Fusion Neck. Features from different layers are commonly used to predict objects across various sizes. The feature fusion neck aims at conducting feature fusion for better prediction. Here, we use $\{P_1, P_2, P_3, P_4\}$ to denote feature levels generated by the backbone e.g. ResNet. From P_1 to P_4, the spatial size is gradually down-sampled with factor 2. We further add two smaller P_5 and P_6 feature maps down-sampled from P_4 following RetinaNet (Lin et al. 2018). The search space contains: no FPN (the original Faster RCNN setting) and FPN with different choices of input and output feature levels (ranging from P_1 to P_6).

Region Proposal Network (RPN). RPN generates multiple foreground proposals within each feature map and only exists in two-stage detectors. Our search space is chosen to be: no RPN (one-stage detectors); with RPN; with Guided anchoring RPN (Wang et al. 2019a).

RCNN Head. RCNN head refines the objects location and predicts final classification results. Cai and Vasconcelos proposed cascade RCNN heads to iterative refine the detection results, which has been proved to be useful yet requiring more computational resources. Thus, we consider regular RCNN head (Ren et al.; Lin et al. 2015; 2017), RetinaNet head (Lin et al. 2018), and cascade RCNN heads with different number of heads (2 to 4) as our search space to exam the accuracy/speed trade-off. Note that our search space covers both one-stage and two-stage detection systems.

Input Resolution. Furthermore, the input resolution is closely related to the accuracy and speed. Qin et al. also suggested that input resolution should match the capability of the backbone, which is not measurable in practice. Intuitively, we thus add input resolution in our search space to find the best matching with different models: 512x512, 800x600, 1080x720 and 1333x800.

Inference time is then evaluated for each combination of modules. Together with the accuracy on validation dataset, a Pareto front is then generated showing the optimal structures of the detector under different resource constraints.

Stage-two: Modular-level Search On the Pareto front generated by Stage-one, we can pick up several efficient detection structures with different combination of modules. Then in Stage-two, we search the detailed architecture for each module and lift the boundary of speed/accuracy tradeoff of the selected structures.

Qin et al. suggested that in detection backbone, early-stage feature maps are larger with low-level features which describe spatial details, while late-stage feature maps are smaller with high-level features which are more discriminative. Localization subtask is sensitive to low-level features while high-level features are crucial for classification. Thus, a natural question is to ask how to leverage the computational cost over different stages to obtain an optimal design for detection. Therefore, inside the backbone, we design a flexible search space to find the optimal base channel size, as well as the position of down-sampling and channel-raising.

As shown in Figure 2, the Stage-two backbone search space consists of 5 stages, each of which refers to a bench of convolutional blocks fed by the features with the same resolution. The spatial size of stage 1 to 5 is gradually downsampled with factor 2. As suggested in Li et al., we fix stage 1 and the first layer of stage 2 to be a 3x3 conv (stride=2). We use the same block setting (basic / bottleneck residual block, ResNeXt block or MBblock (Sandler et al. 2018)) as the structures selected from the result of Stage-one. For example, if the candidate model selected from Stage-one's Pareto front is with ResNet101 as the backbone, we will use the corresponding bottleneck residual block as its search space.

Furthermore, the backbone architecture encoding string is like "basicblock_54_1211-211-1111-12111" where the first placeholder encodes the block setting; 54 is the base channel size; "-" separates each stage with different resolution; "1" means regular block with no change of channels and "2" indicated the number of base channels is doubled in this block. The base channel size is chosen from $48, 56, 64, 72$. Since there is no pre-trained model available for customized backbones, we use a fast train-from-scratch technique instead which will be elaborated in the next section.

Besides the flexible backbone, we also adjust the channel size of the FPN during the Stage-two search. The input channel size is chosen from $128, 256, 512$ and the channels of the head is adjusted correspondingly. Thus, the objective of Stage-two is to further refine the detailed modular structure of the selected efficient architectures.

Train from Scratch and Fast Evaluate the Architecture

Most of the detection models require initialization of backbone from the ImageNet (Russakovsky et al. 2015) pretrained models during training. Any modification on the structure of backbone requires training again on the ImageNet, which makes it harder to evaluate the performance of a customized backbone. This paradigm hinders the development of efficient NAS for detection problem. Shen et al. first explores the possibility of training a detector from scratch by the deeply supervised networks and dense connections. He, Girshick, and Dollár and ScratchDet (Zhu et al. 2018) find that normalization play an significant role in training from scratch and a longer training can then help to catch up pretrained counterparts. Inspired by those works, we conjecture the difficulty from two factors and try to fix them:

1) Inaccurate Batch Normalization because of smaller batch size: During the training, the batch-size is usually very small because of high GPU consumption, which leads to inaccurate estimation of the batch statistics and increasing the model error dramatically (Wu and He 2018). To alleviate this problem, we use Group Normalization (GN) instead of standard BN since GN is not sensitive to the batch size.

2) Complexity of the loss landscape: Shen et al. suggested that the multiple loss and ROI pooling layer in detection hinder the gradient of region-level backward to the backbone. Significant loss jitter or gradient explosion are often observed during training from scratch. BN has been proved to be an effective solution of the problem through significantly smoothing the optimization landscape. Instead of using BN, which is not suitable for small batch size training, we adopt Weight Standardization (WS) (Qiao et al. 2019) for the weights in the convolution layers to further smooth the loss landscape.

Experiments in the later section show that with GN and WS, a much larger learning rate can be adopted, thus enabling us to train a detection network from scratch even faster than the pre-trained counterparts.

Multi-objective Search Algorithm

For each stage, we aims at generating a Pareto front showing the optimal trade-off between accuracy and different computation constrains. To generate the Pareto front, we use non-

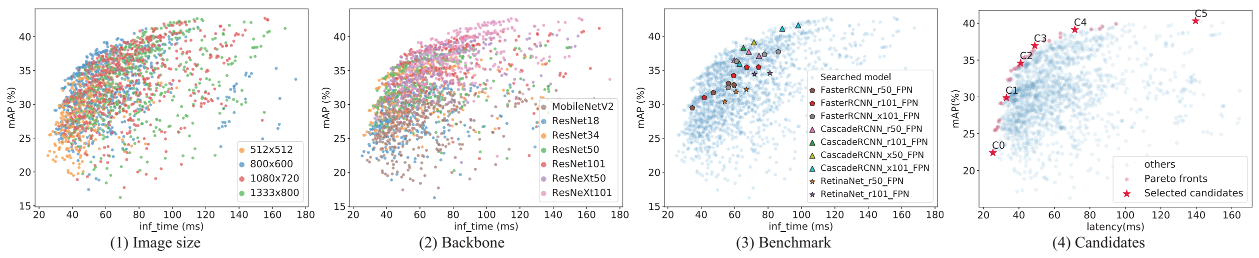

Figure 3: Intermediate results for Stage-one: Structural-level Searching. Comparison of mAP and inference time of all the architectures searched on COCO. Inference time is tested on one V100 GPU. It can be found our searching already found many structures dominate state-of-the-art objectors. On the Pareto front, we pick 6 models (C0 to C5) and further search for better modular-level architectures in Stage-two.

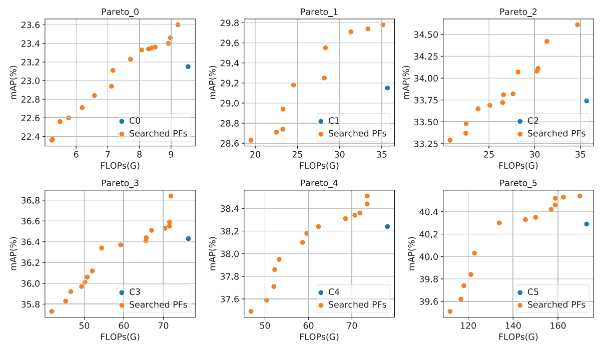

Figure 4: Intermediate results for Modular-level Search. The architectures with blue dot are the selected model C0-C5 based on the previous Stage-one. The orange dots are architectures forming the Pareto front found by our algorithm.

dominate sorting to determinate whether one model dominates another in terms of both efficiency and accuracy. In Stage-one, we use inference time on one V100 GPU as the efficiency metric to roughly compare the actual performance between different structures. In Stage-two, we use FLOPs instead of actual time since FLOPs is more accurate than inference time to compare different backbones with the same kind of block (the inference time has some variation because of the GPU condition). Moreover, FLOPs is able to keep the consistency of rank when changing the BN to GN+WS during searching in Stage-two.

The architecture search step is based on: 1) the evolutionary algorithm to mutate the best architecture on the Pareto front; 2) Partial Order Pruning method (Li et al. 2019a) to prune the architecture search space with the prior knowledge that deeper models and wider models are better. Our algorithm can be parallelized on multiple computation nodes (each has 8 V100 GPUs) and lift the Pareto front simultaneously.

Experiments

Architecture Search Details

We conduct architecture search on the well-known **COCO** (Lin et al. 2014) dataset, which contains 80 object classes with 118K images for training, 5K for evaluation. For Stage-one, we consider a totally 1.1×10^4 combination of modules. For Stage-two, the search space is much larger, containing about 5.0×10^{12} unique paths. We conduct all experiments using Pytorch (Paszke et al.; Chen et al. 2017; 2018a), multiple computational nodes with 8 V100 cards on each server. To measure the inference speed, we run all the testing images on one V100 GPU and take the average inference time for comparison. All experiments are performed under CUDA 9.0 and CUDNN 7.0.

Implementation Details for Stage-one. During searching, we first generate some initial models with a random combination of modules. Then evolutionary algorithm is used to mutate the best architecture on the Pareto front and provides candidate models. During architectures evaluation, we use SGD optimizer with cosine decay learning rate from 0.04 to 0.0001, momentum 0.9 and 10^{-4} as weight decay. Pre-trained models on ImageNet (Russakovsky et al. 2015) are used as our backbone for fast convergence. Empirically, we found that training with 5 epochs can separate good models from bad models. In this stage, we evaluate about 500 architectures and it takes about 2000 GPU hours for the whole searching process.

Intermediate Results for Stage-one. The first two figures in Figure 3 show the comparison of mAP and inference time of the architectures searched on COCO. From Figure 3-1, it can be found that different input resolution can variate the speed and accuracy. We also found that MobileNet V2 is dominated by other models although it has mush less FLOPs in Figure 3-2. This is because it has higher memory access cost thus is slower in practice (Li et al. 2019a). Therefore, using the direct metric, i.e. inference time, rather than approximate metric such as FLOPs is necessary for achieving the best speed/accuracy trade-off and our searching found some structures dominate classic detectors. On the generated Pareto front, we pick 6 models (C0 to C5) and further search for the better modular-level architectures in Stage-two.

Implementation Details for Stage-two. During Stage-two, we use the training strategy with GN and WS methods discussed in the previous section. We use cosine decay learning rate ranging from 0.24 to 0.0001 with batch size 8 on each GPU. The model is trained with 9 epochs to fully explore the different modular-level structures. It is

Model	Input size	Backbone	Neck	RPN	RCNN Head	Backbone FLOPs	Time (ms)	mAP
E0	512x512	basicblock_64_1-21-21-12	FPN(P_2-P_5, c=128)	RPN	2FC	7.2G (0.75)	24.5	27.1
E1	800x600	basicblock_48_12-21-11111-211111	FPN(P_2-P_5, c=256)	RPN	2FC	28.3G (0.79)	32.2	34.3
E2	800x600	basicblock_56_12-11111-211-1112	FPN(P_1-P_5, c=128)	RPN	Cascade(n=3)	23.8G (0.67)	39.5	40.1
E3	800x600	bottleneck_56_211-111111111-2111111-11112111	FPN(P_1-P_5, c=128)	RPN	Cascade(n=3)	59.2G (0.78)	50.7	42.7
E4	800x600	Xbottleneck_56_21-21-111111111111111-2111111	FPN(P_1-P_5, c=256)	GA-RPN	Cascade(n=3)	73.5G (0.96)	80.2	43.9
E5	1333x800	Xbottleneck_56_21-21-11111111111111-21111111	FPN(P_1-P_5, c=256)	GA-RPN	Cascade(n=3)	162.45G (0.94)	108.1	46.1

Table 2: Detailed architecture of the final SM-NAS models from E0 to E5. For the backbone, basicblock and bottleneck follow the same as in ResNet (He et al. 2016) and Xbottleneck refers to the block setting of ResNeXt (Xie et al. 2017b);. For Neck, P_2-P_5 and "c" denotes the choice and the channels of output feature levels in FPN. For RCNN head, "2FC" is the regular setting of two shared fully connected layer; "n" means the stages of the cascade head.

id	Norm Method	ImageNet Pretrain	Epoch	Batchsize	lr	mAP
0	BN	✓	12	2x8	0.02	36.5
1	BN	✓	24	2x8	0.02	37.4
2	BN	×	12	2x8	0.02	24.8
3	BN	×	12	8x8	0.20	28.3
4	GN	×	12	2x8	0.02	29.4
5	GN+WS	×	12	2x8	0.02	30.7
6	GN+WS	×	12	2x8	0.10	36.4
7	**GN+WS**	×	**16**	**4x8**	**0.16**	**37.5**

Table 3: FPN with ResNet-50 trained with different strategies, evaluated on COCO val. "GN" is group normalization by Wu and He. "WS" is the Weight Standardization method by Qiao et al.. We found that with group normalization, Weight Standardization, larger learning rate and batchsize, we can train a detection network from scratch using less epochs than standard training procedure.

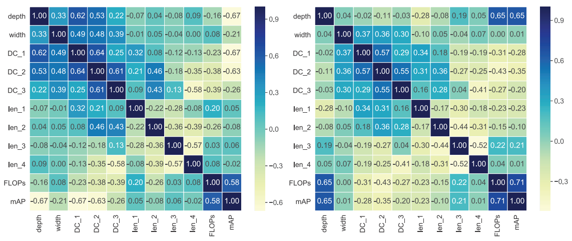

Figure 5: Correlation between factors of the searched models on COCO dataset. The left figure shows the results of Pareto front 4. The depth and width are the number of blocks and base channel size of backbone. DC_x denotes the positions where the channel size is doubled; and len_x denotes the proportion of the total blocks of the xth stage.

worth mention that we directly search on the COCO without pre-trained models. In Stage-two, we evaluate about 300 architectures for each group and use about 2500 GPU hours.

Intermediate Results for Stage-two. Figure 4 shows mAP/speed improvement of the searched models compared to the optimal model selected in Stage-one. It can be found that SM-NAS can further push the Pareto front to a better trade-off of speed/accuracy.

Object Detection Results

On the COCO dataset, the optimal architectures E0 to E5 are identified with our two-stages search. We change the backbone back to BN and no Weight Standardization mode since these practices will slow down the inference time. We first pre-train those searched backbones on ImageNet following common practice (He et al. 2016) for fair comparison with other methods. Then stochastic gradient descent (SGD) is performed to train the full model on 8 GPUs with 4 images on each GPU. Following the setting of 2x schedule (He, Girshick, and Dollár 2018), the initial learning rate is 0.04 (with a linear warm-up), and reduces two times ($\times 0.1$) during fine-tuning; 10^{-4} as weight decay; 0.9 as momentum. The training and testing is conducted with the searched optimal input resolutions. Image flip and scale jitter is adopted for augmentation during training, and evaluation procedure follows the COCO official setting (Lin et al. 2014).

Architectures of the Final Searched Models. Table 2 shows architecture details of the final searched E0 to E5. Comparing the searched backbones with classical ResNet/ResNeXt, we find that early stages in our models are very short which is more efficient since feature maps in an early stage is very large with a high computational cost. We also found that for high-performance detectors E3-E5, raising channels usually happens in very early stage which means that lower-level feature plays an important role for localization.The classification performance of the backbone of E0 to E5 on ImageNet can also be found in the supplementary materials. We can find the searched backbones are also efficient in the classification task.

In Table 4, we make a detailed comparison with existing detectors: YOLOv3 (Redmon and Farhadi 2018), RetinaNet (Lin et al. 2018), FSAF (Zhu, He, and Savvides 2019), CornerNet (Law and Deng 2018), CenterNet (Zhou, Wang, and Krähenbühl 2019), AlignDet (Chen et al. 2019b), GA-FasterRCNN (Wang et al. 2019a), Faster-RCNN (Ren et al. 2015), Mask-RCNN (He et al. 2017), Cascade-RCNN (Cai and Vasconcelos 2018), TridentNet (Li et al. 2019b), and NAS-FPN (Ghiasi, Lin, and Le 2019). Most reported results are tested with single V100 GPU (some models marked with other GPU devices following the original papers). For a fair comparison, multi-scale testing is not adopted for all methods. From E0 to E5, SM-NAS constructs a Pareto front that dominates most SOTA models as shown in Figure 1.

Ablative Study for Training from Scratch. Since in modular-level searching stage, we keep changing the backbone structure, we need to find an optimal setting of training strategies for efficiently training a detection network from scratch. Table 3 shows an ablative study of FPN

Method	Backbone	Input size	Inf time (ms)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLO v3	DarkNet-53	608x608	51.0 (TitanX)	33.0	57.9	34.4	18.3	35.4	41.9
RetinaNet	ResNet101-FPN	1333x800	91.7 (V100)	39.1	59.1	42.3	21.7	42.7	50.2
FSAF	ResNet101-FPN	1333x800	92.5 (V100)	40.9	61.5	44.0	24.0	44.2	51.3
CornerNet	Hourglass-104	512x512	244.0 (TitanX)	40.5	56.5	43.1	19.4	42.7	53.9
CenterNet	Hourglass-104	512x512	126.0 (V100)	42.1	61.1	45.9	24.1	45.5	52.8
AlignDet	ResNet101-FPN	1333x800	110.0 (P100)	42.0	62.4	46.5	24.6	44.8	53.3
GA-Faster RCNN	ResNet50-FPN	1333x800	104.2 (V100)	39.8	59.2	43.5	21.8	42.6	50.7
Faster-RCNN	ResNet101-FPN	1333x800	84.0 (V100)	39.4	-	-	-	-	-
Mask-RCNN	ResNet101-FPN	1333x800	105.0 (V100)	40.2	-	-	-	-	-
Cascade-RCNN	ResNet101-FPN	1333x800	97.9 (V100)	42.8	62.1	46.3	23.7	45.5	55.2
TridentNet	ResNet101	1333x800	588 (V100)	42.7	63.6	46.5	23.9	46.4	55.6
TridentNet	ResNet101-deformable-FPN	1333x800	2498.3 (V100)	48.4	69.7	53.5	31.8	51.3	60.3
DetNAS	Searched Backbone	1333x800	-	42.0	63.9	45.8	24.9	45.1	56.8
NAS-FPN	ResNet50-FPN(@384)	1280x1280	198.7 (V100)	45.4	-	-	-	-	-
SM-NAS: E2	Searched Backbone	800x600	**39.5**(V100)	**40.0**	58.2	43.4	21.1	42.4	51.7
SM-NAS: E3	Searched Backbone	800x600	**50.7**(V100)	**42.8**	61.2	46.5	23.5	45.5	55.6
SM-NAS: E5	Searched Backbone	1333x800	**108.1**(V100)	**45.9**	64.6	49.6	27.1	49.0	58.0

Table 4: Comparison of mAP and inference time of the state-of-the-art single-model on COCO test-dev. Our searched models dominate most SOTA models in terms of speed/accuracy by a large margin.

Dataset	Input size	model	inf_time (ms)	mAP
VOC	800x600	FPN w R50	38.4	80.4
		E0	**21.5**	81.4
		E1	35.9	83.7
		E3	47.0	**84.4**
BDD	1333x800	FPN w R101	84.6	36.9
		E0	**27.8**	30.2
		E1	45.2	37.9
		E3	67.2	**39.6**

Table 5: Transferability of our models on PASCAL VOC (VOC) and Berkeley Deep Drive dataset(BDD).

with ResNet-50 trained with different strategies, evaluated on COCO. Exp-0 and Exp-1 are the 1x and 2x standard FPN training procedure following He, Girshick, and Dollár. Comparing Exp-2&3, with Exp-4, it can be found smaller batch size leads to inaccurate batch normalization statistics. Using group normalization can alleviate this problem and improve the mAP from 24.8 to 29.4. From Exp-5, adding WS can further smooth the training and improve mAP by 1.3. Furthermore, enlarging the learning rate and batch size can increase the mAP to 37.5 in 16-epoch-training (see Exp 5&6&7). Thus, we can train a detection network from scratch using fewer epochs than the pre-trained counterparts.

Architecture Transfer: VOC and BDD. To evaluate the domain transferability of the searched models, we transfer the searched architecture E0-E3 from COCO to Pascal VOC and BDD. For **PASCAL VOC** dataset (Everingham et al. 2010) with 20 object classes, training is performed on 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 using IoU at 0.5. **Berkeley Deep Drive (BDD)** (Yu et al. 2018) is an autonomous driving dataset with 10 object classes, containing about 70K images for training and 10K for evaluation. We use the same training and testing configurations for a fare comparison. As shown in Table 5, on Pascal VOC, E0 reduces half of the inference time compared to FPN with a higher mAP. For BDD, E3 is 17.4ms faster than FPN. The searched architectures show good transferability.

Correlation Analysis of Architecture and mAP. It is interesting to analyze the correlation between the factors of architecture and mAP. Figure 5 shows the correlation between factors of all the searched models on COCO dataset. The left figure shows the results of Pareto front 4 in Stage-two. It can be found that under the constraints of FLOPs, better architecture should decrease the depth and put the computation budget in the low-level stage. The right figure shows correlation for all the searched models. Depth shows strong positive relation with mAP, raising channels in early stage is good for detection. It is better to have a longer high-level stage and shorter low-level stage.

Conclusion

We propose a detection NAS framework for searching both an efficient combination of modules and better modular-level architectures for object detection on a target device. The searched SM-NAS networks achieve state-of-the-art speed/accuracy trade-off. The SM-NAS pipeline can keep updating and adding new modules in the future.

References

Cai, Z., and Vasconcelos, N. 2018. Cascade r-cnn: Delving into high quality object detection. In *CVPR*.

Cai, H.; Chen, T.; Zhang, W.; Yu, Y.; and Wang, J. 2018. Efficient architecture search by network transformation. In *AAAI*.

Chen, K.; Pang, J.; Wang, J.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Shi, J.; Ouyang, W.; Loy, C. C.; and Lin, D. 2018a. mmdetection. https://github.com/open-mmlab/mmdetection.

Chen, L.-C.; Collins, M.; Zhu, Y.; Papandreou, G.; Zoph, B.; Schroff, F.; Adam, H.; and Shlens, J. 2018b. Searching for efficient multi-scale architectures for dense image prediction. In *NIPS*.

Chen, Y.; Yang, T.; Zhang, X.; Meng, G.; Pan, C.; and Sun, J. 2019a. Detnas: Neural architecture search on object detection. In *NIPS*.

Chen, Y.; Han, C.; Wang, N.; and Zhang, Z. 2019b. Revisiting feature alignment for one-stage object detection. *arXiv preprint arXiv:1908.01570*.

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

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

Ghiasi, G.; Lin, T.-Y.; and Le, Q. V. 2019. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*.

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

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *ICCV*.

He, K.; Girshick, R.; and Dollár, P. 2018. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*.

Law, H., and Deng, J. 2018. Cornernet: Detecting objects as paired keypoints. In *ECCV*.

Li, Z.; Peng, C.; Yu, G.; Zhang, X.; Deng, Y.; and Sun, J. 2017. Light-head r-cnn: In defense of two-stage object detector. In *CVPR*.

Li, Z.; Peng, C.; Yu, G.; Zhang, X.; Deng, Y.; and Sun, J. 2018. Detnet: A backbone network for object detection. In *ECCV*.

Li, X.; Zhou, Y.; Pan, Z.; and Feng, J. 2019a. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *CVPR*, 9145–9153.

Li, Y.; Chen, Y.; Wang, N.; and Zhang, Z. 2019b. Scale-aware trident networks for object detection. *arXiv preprint arXiv:1901.01892*.

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

Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2017. Feature pyramid networks for object detection. In *CVPR*.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2018. Focal loss for dense object detection. *TPAMI*.

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

Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

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

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

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. In *ICLR*.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS Workshop*.

Qiao, S.; Wang, H.; Liu, C.; Shen, W.; and Yuille, A. 2019. Weight standardization. *arXiv preprint arXiv:1903.10520*.

Qin, Z.; Li, Z.; Zhang, Z.; Bao, Y.; Yu, G.; Peng, Y.; and Sun, J. 2019. Thundernet: Towards real-time generic object detection. *arXiv preprint arXiv:1903.11752*.

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y. L.; Tan, J.; Le, Q. V.; and Kurakin, A. 2017. Large-scale evolution of image classifiers. In *ICML*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

Redmon, J., and Farhadi, A. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *IJCV* 115(3):211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 4510–4520.

Shen, Z.; Liu, Z.; Li, J.; Jiang, Y.-G.; Chen, Y.; and Xue, X. 2017. Dsod: Learning deeply supervised object detectors from scratch. In *ICCV*, 1919–1927.

Tan, M., and Le, Q. V. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

Wang, J.; Chen, K.; Yang, S.; Loy, C. C.; and Lin, D. 2019a. Region proposal by guided anchoring. In *CVPR*, 2965–2974.

Wang, N.; Gao, Y.; Chen, H.; Wang, P.; Tian, Z.; and Shen, C. 2019b. Nas-fcos: Fast neural architecture search for object detection. *arXiv preprint arXiv:1906.04423*.

Wu, Y., and He, K. 2018. Group normalization. In *ECCV*, 3–19.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2017a. Aggregated residual transformations for deep neural networks. In *CVPR*, 1492–1500.

Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2017b. Aggregated residual transformations for deep neural networks. In *CVPR*.

Xu, H.; Yao, L.; Zhang, W.; Liang, X.; and Li, Z. 2019. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *ICCV*.

Yu, F.; Xian, W.; Chen, Y.; Liu, F.; Liao, M.; Madhavan, V.; and Darrell, T. 2018. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*.

Zhou, X.; Wang, D.; and Krähenbühl, P. 2019. Objects as points. *arXiv preprint arXiv:1904.07850*.

Zhu, R.; Zhang, S.; Wang, X.; Wen, L.; Shi, H.; Bo, L.; and Mei, T. 2018. Scratchdet: Exploring to train single-shot object detectors from scratch. *arXiv preprint arXiv:1810.08425*.

Zhu, C.; He, Y.; and Savvides, M. 2019. Feature selective anchor-free module for single-shot object detection. *arXiv preprint arXiv:1903.00621*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.