Auto-FPN: Automatic Network Architecture Adaptation for Object Detection Beyond Classification

Hang Xu[1*] Lewei Yao[1*] Wei Zhang[1] Xiaodan Liang[2†] Zhenguo Li[1]
[1]Huawei Noah's Ark Lab [2]Sun Yat-sen University

Abstract

Neural architecture search (NAS) has shown great potential in automating the manual process of designing a good CNN architecture for image classification. In this paper, we study NAS for object detection, a core computer vision task that classifies and localizes object instances in an image. Existing works focus on transferring the searched architecture from classification task (ImageNet) to the detector backbone, while the rest of the architecture of the detector remains unchanged. However, this pipeline is not task-specific or data-oriented network search which cannot guarantee optimal adaptation to any dataset. Therefore, we propose an architecture search framework named Auto-FPN specifically designed for detection beyond simply searching a classification backbone. Specifically, we propose two auto search modules for detection: Auto-fusion to search a better fusion of the multi-level features; Auto-head to search a better structure for classification and bounding-box (bbox) regression. Instead of searching for one repeatable cell structure, we relax the constraint and allow different cells. The search space of both modules covers many popular designs of detectors and allows efficient gradient-based architecture search with resource constraint (2 days for COCO on 8 GPUs). Extensive experiments on Pascal VOC, COCO, BDD, VisualGenome and ADE demonstrate the effectiveness of the proposed method, e.g. achieving around 5% improvement than FPN in terms of mAP while requiring around 50% fewer parameters on the searched modules.

1. Introduction

Object detection is a core and challenging problem in computer vision. It aims to localize objects and predict the associated class labels in an image. This task widely benefits autonomous vehicles [9], surveillance video [43], lesion analysis in medical images [59], facial recognition in mobile phone [3], to name a few. Deep convolutional neural

Model	Dataset	Search Method	GPU Days	Resource Constraint	Task
ResNet [20]	ImageNet	Handcraft	-	-	Cls
FPN [32]	COCO	Handcraft	-	-	Det
NASNet [64]	CIFAR-10	RL-based	2000	×	Cls
	CIFAR-10	Transfer	2000	×	Det
AmoebaNet [45]	CIFAR-10	EA-based	2000	✓	Cls
DARTS [38]	CIFAR-10	Grad-based	4	×	Cls
PNASNet [36]	CIFAR-10	SMBO	150	×	Cls
DPC [11]	Cityscapes	EA-based	2600	×	Seg
Auto-deeplab [35]	Cityscapes	Grad-based	3	×	Seg
Auto-FPN	PASCAL VOC	Grad-based	4	✓	Det
	COCO	Grad-based	16	✓	Det
	BDD	Grad-based	8	✓	Det

Table 1. Comparing our Auto-FPN against other CNN architectures designs. 1) We make one of the first attempts to propose an architecture search framework for detection. 2) Our method is a gradient-based architecture search with resource constraints. 3) Our efficient search requires only 16 V100 GPU days for COCO.

networks have been proved successful in object detection and the performance grows rapidly. A modern object detection system [40, 47, 50, 32, 18, 33] usually consists of four components: pretrained backbone (e.g. Imagenet), feature fusion neck, region proposal network (in two-stage detection), and RCNN head. While a better feature extractor [54, 20, 57, 48, 49] certainly plays an important role, a lot of progress in this area comes from the better design of network architectures for feature fusion neck [32, 39, 29, 19] and RCNN head [13, 30, 8, 28].

There has been a growing interest in automatically designing a neural network architecture instead of relying heavily on human expert's labor and experience. Neural architecture search (NAS) has demonstrated much success in exploring architectures that exceed hand-crafted architectures on image classification problems [64, 36, 38, 45]. Image classification is straight-forward and relatively simple since it requires less computation and the training process is fast (CIFAR-10). In contrast, an object detection problem is usually more challenging due to high-resolution input image and longer training time (2 days for COCO on 8 GPUs). For the problem of detection, existing NAS work [64] only

*Equal contribution
†Corresponding author: xdliang328@gmail.com

transfers the searched architecture from classification task (ImageNet) to the detector backbone, while the rest of the architecture of detector remains unchanged. However, this pipeline is not task-specific or data-oriented search which cannot guarantee optimal adaptation to any domain. On the contrary, our work aims to develop an efficient NAS scheme specifically designed for object detection and can be adapted to any detection dataset.

In this paper, we propose an architecture search framework named Auto-FPN for detection beyond simply searching a classification backbone. We draw inspiration from the recent progress on two important components of detection system: feature fusion neck and RCNN head. Specifically, we propose two auto search modules for detection: Auto-fusion neck module aims at utilizing information from all feature levels in the in-network feature hierarchy and conducting feature fusion for better prediction; Auto-head to search a better and efficient structure for classification and bbox regression.

By observing the state-of-the-art design of the feature fusion neck, two components are important: 1) good path for aggregating different feature levels (e.g. top-down path in [32] and augmented bottom-up path in [39]), 2) proper receptive fields for each feature levels [29]. Thus in our Auto-fusion module, we design a fully connected search space with various dilated convolution operations between feature levels to allow a flexible feature fusion with different receptive fields. For RCNN head, typical two-stage detectors suffer from a heavy head (34% parameter size of the whole FPN) [30]. To build an efficient yet accurate detector, we consider a more advanced split-transform-merge paradigm for the search space of Auto-head. Instead of searching one repeatable cell structure [38, 37, 45, 5, 55, 41], we relax the constraint and endow different structures in each module. The search space of both modules is flexible enough to cover many popular designs of detectors. For feasible employment of the network, we further consider adding a joint computational constrain and taking the parameter size, FLOPS and MAC into regularizing the search.

For architecture search methods, evolutionary algorithms [46, 37, 45] and reinforcement learning [1, 63, 64, 4, 61] based searching methods usually require large amounts of repeated training and evaluation on candidate architectures thus are computationally intensive even on the low resolution image classification dataset (e.g. CIFAR-10). Therefore, both methods may not be suitable for object detection which requires high-resolution input image and longer training time. Inspired by the recent differentiable formulation of NAS [38, 58], we apply a continuous relaxation of the discrete architectures of both modules. Architecture search is then conducted via stochastic gradient descent. Since high-resolution is crucial for detecting small objects, we directly search on high-resolution images with size of 800 pixels. The search is very efficient for each module and takes only about 11 hours on 8 V100 GPUs for PASCAL VOC [14], 2 days for COCO [34] and 1 day for BDD [60]. In practice, training a good gradient-based NAS is not easy. We further explore and share the tricks to compare different training strategies in Section 4.3.

Extensive experiments are conducted on the widely used detection benchmarks, including Pascal VOC [14], COCO [34], BDD [60], Visual Genome (VG) [27], and ADE [62]. The proposed method outperforms current state-of-the-art detection methods, i.e., the baseline FPN [32] and PANet [39]. We observe consistent performance gains on the base detection network FPN [32] but with fewer parameters. In particular, our method achieves around 2.6% of relative mAP improvement on Pascal VOC, 5.2% on MS-COCO, 2.7% on BDD, 28.2% on VG and 15.2% on ADE with more than 50% fewer parameters in neck and head.

2. Related Work

Object Detection. Modern object detection methods can generally be categorized in two groups: one-stage detection methods such as SSD [40] and YOLO [47], and two-stage detection methods such as Faster R-CNN [50], FPN [32], and R-FCN [13]. Within the state-of-the-art systems, there are two essential components besides the classification backbone: multi-level features module and RCNN head. For example, FPN [32] and PANet [39] modified multi-level features module to obtain a better feature fusion. On the other hand, R-FCN [13] and Light-head RCNN [30] designed different structures of bbox head.

Neural Architecture Search. Neural architecture search aims at automatically searching for an efficient neural network architecture for a certain task and dataset, hence releasing human experts from labor of designing network. Apart from some RNN searching works [24, 63, 38] for NLP tasks, most works are based on searching CNN architectures for image classification. Only a few of them focus on more complicated vision tasks such as semantic segmentation [11, 35]. There are mainly three approaches in NAS area: 1) Evolutionary Algorithms (EA) based methods [46, 37, 45], which try to "evolves" architectures or Network Morphism by mutating the current best architectures; 2) Reinforcement learning based methods [1, 63, 64, 4, 61] train a RNN policy controller to generate a sequence of actions to specify CNN architecture; 3) Gradient based methods [38, 58, 35, 6] define an architecture parameter for continuous relaxation of the discrete search space, thus allowing differentiable optimization of the architecture. Among those approaches, EA and RL methods usually require massive computation during search, usually thousands of GPU days. Meanwhile, gradient based methods often take only several GPU days. Since training for object detection is already very time consuming, gradient based approach seems

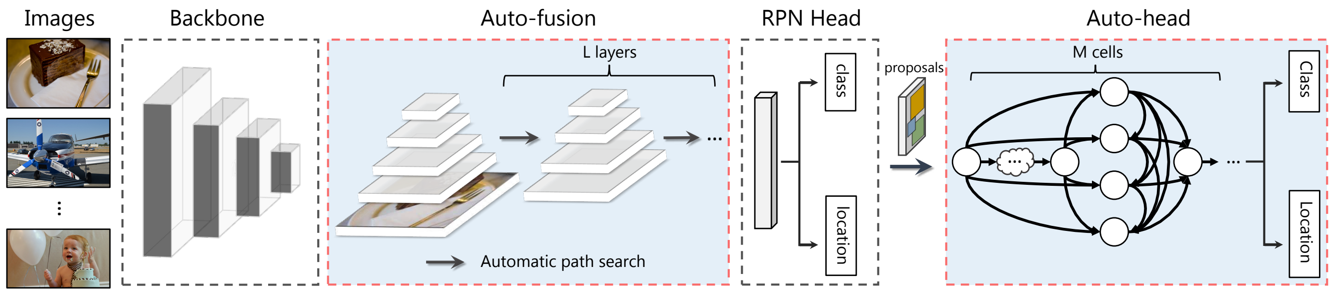


Figure 1. An overview of our architecture search framework for detection. Our method can be stacked on any backbone and focuses on searching better architectures of the neck and the RCNN head of a detection system. The proposed Auto-fusion module aims at finding a better architecture for utilizing information from all feature hierarchy and conducting feature fusion for better prediction. The Auto-head module is constructed by different auto-searched cells in order to perform a better classification and bbox prediction.

to be more desirable for current setting of the problem. The concurrent work NAS-FPN [16] is a nice related work released after the submission of this paper. We use a different search space and different algorithm compared to it. Although their results show much higher mAP, their model is considerably large. Their searched Neck accounts for more than 300B FLOPs and 25M parameters (larger than a ResNet50 backbone), while ours is even smaller than the vanilla FPN. This is because NAS-FPN only considers mAP as the objective, while our work also considers resource constraint to seek a best trade off between efficiency and performance. For a comparison, our method requires much less params/ FLOPs (0.04x/ 0.1x) and runs much faster(2.8x) than NAS-FPN.

Multi-level Features. Multi-level features are commonly used in high-level recognition tasks. In segmentation, lower-level features can help to recognize the detailed edges of the objects. In detection, fusion of multi-level features can help to recognize information in small areas. SSD [40], DSSD [15], and MS-CNN [7] create a feature pyramid and assign proposals to appropriate feature levels for inference. HyperNet [26], ION [2] and Hypercolumn [17] concatenate features from different levels. FCN [42] and U-Net [51] fused features from lower layers through skip-connections. FPN [32], RetinaNet [33], TDM [53] and PANet [39] augmented a top-down path or a bottom-up path for object detection. TridentNet [29] further constructs a parallel multi-branch feature architecture with different receptive fields.

3. The Proposed Approach

3.1. Motivation and Overview

Modern object detection systems can be decoupled into four components: ImageNet pretrained backbone, feature fusion neck, region proposal network (RPN), and RCNN head for classification and bbox regression.

ImageNet pretrained backbone. The common backbone can be VGG [54], ResNet [20], and ResNeXt [57]. Most of the detection models require initialization of backbone from the ImageNet [52] pretrained models during training. Although He et al. [18] have shown that ImageNet pretraining is not indispensable for detection, a much longer training (x11) is required as a compensation for no pretraining, which makes it computationally infeasible for us to directly search on the backbone.

Feature fusion neck. Features from different layers (usually a feature pyramid) are commonly used in object detection to predict objects of various sizes. The feature fusion neck aims at utilizing information from all feature levels in the in-network feature hierarchy and conducting feature fusion for better prediction. NAS can be naturally implemented in this work for better feature fusion.

Region proposal network (RPN). RPN only exists in two-stage detectors for generating multiple anchor proposals within a particular image. The design of RPN is quite simple but effective. Therefore, in this paper we continue to use this design and do not search for it.

RCNN head. This part aims at refining the object proposal's location and predicting final classification result. However, handcrafting architecture is heavy and inefficient. We use NAS here for better classification and localization results.

We believe searching better architectures for the neck and head are currently the most important issues in object detection with no exploration in the community. Therefore, we make the first attempt and propose an architecture search framework consisting of two modules: Auto-fusion for neck and Auto-head, which are specifically designed for detection as shown in Figure 1.

3.2. Auto-Fusion

Searching a better feature fusion. Generally speaking, neurons in high layers strongly respond to entire objects while other neurons are more likely to be activated by local textures and patterns. Taking FPN [32] as an example, FPN's neck augments a top-down path to propagate semantically strong features to local textures and patterns and enhances all features with reasonable classification capability. PANet [39] further enhances the entire feature hierarchy

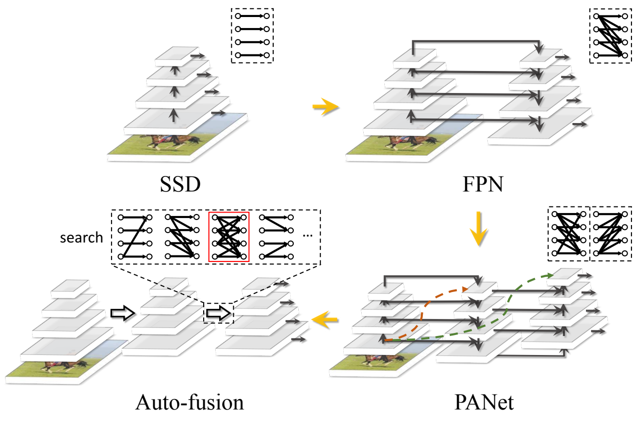


Figure 2. Illustration of different designs of detection neck. SSD [40] directly considers a feature pyramid structure to utilize the feature hierarchy for detection. FPN [32] proposes a top-down pathway to conduct feature fusion. PANet [39] further enhances the entire feature hierarchy by bottom-up path augmentation. Instead of those handcrafted designs, our Auto-fusion tries to find an optimal neck structure which can adapt to any detection dataset and tasks. Auto-fusion's search space can cover all connection patterns described above, and it tries to seek a better fusion process through automatic neural architecture search. Links in the dashed box at the top right represent the equivalent connection patterns under our search space. It can be found that our search space is general enough to cover SSD, FPN, and PANet.

with accurate localization signals in lower layers by bottom-up path augmentation, which shortens the information path between lower layers and the topmost features. A comparison of different kinds of necks can be found in Figure 2. To cover all the connection patterns, we propose an Auto-fusion module with a fully connected search space. On the other hand, TridentNet [29] adapts different dilated convolutions on each feature level for assigning proper receptive fields. Thus various dilated convolutions operations are included in our Auto-fusion.

Search space and architecture. Conventionally, we define that layers producing feature maps with the same spatial sizes are in the same network stage while each feature level corresponds to the output of each stage. Here, we take ResNet [20] as an example and use $\{P_1^0, P_2^0, P_3^0, P_4^0\}$ to denote feature levels generated by the backbone. From P_1 to P_4, the spatial size is gradually down-sampled with factor 2. We use $\{P_1^l, P_2^l, P_3^l, P_4^l\}$ to denote newly generated feature maps after l layers.

Among various neck architectures, we notice two principles that are consistent: the output scale of feature P_i^l remains unchanged as the input feature map P_i^0; P_i^l is upsampled or downsampled to the next layer with a certain operation. Following these common practices, we propose the following neck search space as shown in Figure 3a. Our Auto-fusion consists of L layers with $\{P_1^{l-1}, \ldots, P_i^{l-1}\}$ as input features of the lth layer. For each pair of input jth level feature P_j^{l-1} and output ith level feature P_i^l in the lth layer, links with different operations are considered: $P_i^l = O_{i\to j}(P_j^{l-1})$, where $O(.) \in \mathcal{O_N}$ is an operation, $l = 1, 2, ..., L$. The set of possible operation types $\mathcal{O_N}$ consists of the following operators:

- *no connection (none)*
- *skip connection (identity)*
- *3×3 dilated conv with rate 2*
- *3×3 dilated conv with rate 3*
- *5×5 dilated conv with rate 2*
- *5×5 dilated conv with rate 3*
- *3×3 depthwise-separable conv*
- *5×5 depthwise-separable conv*

Four kinds of dilated convolutions are considered since the spatial-awareness and receptive field are crucial for feature fusion. Finally, P_i^l is upsampled/downsampled to the corresponding target resolution of the lth level.

We reuse the continuous relaxation by adding an architecture parameter α as described in [38]. We approximate each $O_{i\to j}^l$ with its continuous relaxation $\hat{O}_{i\to j}^l$, which is defined as:

$$\hat{O}_{i\to j}(P_j^{l-1}) = \sum_{O^k \in \mathcal{O_N}} \alpha_{i\to j}^{kl} O_{i\to j}^{kl}(P_j^{l-1})$$

where $\sum_{k=1}^{|\mathcal{O_N}|} \alpha_{i\to j}^{kl} = 1$, and $\alpha_{i\to j}^{kl} \geq 0$.

In other words, $\alpha_{i\to j}^{k}$ are normalized scalars associated with each operator $O(.) \in \mathcal{O_N}$, which can be easily implemented as a softmax function. Our goal is then to find a good path for each layer to fuse the features in the neck. In Figure 2, it can be found that our search space is general enough to cover many popular designs for the necks such as SSD [40], FPN [32] and PANet [39].

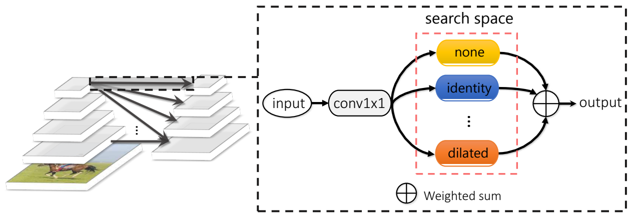
Decoding architectures from α. For each set of $\alpha_{i\to j}^{kl}$ we can choose the most likely operator by taking the $\arg\max(\alpha_{i\to j}^{kl})$. Then the neck is reconstructed by the selected operation. Finally, the whole model is fully trained to report the final performance.

3.2.1 Model Extension on One-stage Detector

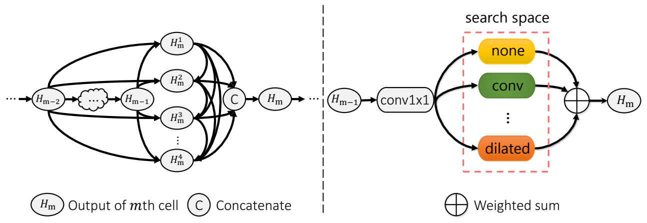
One stage detectors such as SSD [40] and RetinaNet [33] also consider a feature pyramid structure to predict bounding box and classification on different resolutions of feature level. For example, based on the backbone of VGG16, SSD512 considers a feature pyramid structure with 7 feature levels. RetinaNet uses a FPN neck based on ResNet. Thus, our Auto-fusion can be naturally implemented on the multi-level feature structure even on the one-stage detector and helps to find a better feature fusion for the detection task on a particular dataset. Experiments of model extension on SSD can be found in Section 4.2.

3.3. Auto-Head

Searching a better structure for classification and bbox regression. Given the extracted feature for each region proposal, RCNN head aims to predict final classifica-



(a) Auto-fusion layer



(b) Auto-head cell

Figure 3. Illustration of our search space. (a) Each layer of Auto-fusion consists of a series of fully connected links between all feature levels. Each link is composed of a conv1x1 followed by a group of parallel operators in $\mathcal{O}_\mathcal{N}$. (b) Auto-head cell consists of 7 nodes (2 input nodes, 4 intermediate nodes, and 1 output node). Analogous links are designed between nodes.

tion result and refine the proposal's location. Typical two-stage detectors suffer from a heavy head [30]. For example, for Faster-RCNN [50], a complete block of conv5_x of the ResNet is implemented here. For FPN [32], it uses two shared fully connected layers (34% parameters of the whole FPN). It is time-consuming here in terms of per-region prediction and even gets worse when a large number of proposals are utilized. In order to design an efficient and accurate head, we propose our Auto-head module for fast adaptation on any dataset with better classification and localization.

Search space. We consider a split-transform-merge paradigm for designing the search space in Auto-head as shown in Figure 3b. Following [38, 45, 64], we consider M convolutional cells of 7 nodes stacked to form the head. For the mth cell, the input nodes i.e. the first and second nodes are set equal to the outputs of the $(m-2)$th cell and the $(m-1)$th cell, and the output node is the depth-wise concatenation of all the intermediate nodes $\{H_m^1, H_m^2, H_m^3, H_m^4\}$. To allow a more flexible search space, our setup further relaxes the search space to allow different structure of each cell (e.g. different architecture parameter β_m for each cell). Unlike [38], we do not have reduction cell since the input spatial resolution is already very small (e.g. 7x7 for FPN). For the set of possible operations $\mathcal{O}_\mathcal{H}$, we further add 3x3 conv and 5x5 conv to allow fully utilization of extracted features of ROIs and delete dilated conv with rate 3 because dilated conv is not useful here in small spatial resolution.

3.4. Optimization

Using the continuous relaxation allows the architecture parameters α and β to control the connection strength among all operations. Therefore the architecture can be optimized jointly with the network parameters efficiently using stochastic gradient descent. For fast training, we adopt the first-order approximation in [38] and randomly partition the training data into two disjoint sets with the same size: *trainA* and *trainB*. The optimization alternates between these two steps until converge: update network parameters w by $\nabla_w \mathcal{L}_{trainA}(w, \alpha, \beta)$; update architecture parameters α, β by $\nabla_w \mathcal{L}_{trainB}(w, \alpha, \beta)$. The loss function $\mathcal{L}$ consists of 5 parts: localization losses for RPN and RCNN-head, classification losses for RPN and RCNN-head, and the resource constraint $C(\alpha, \beta)$.

3.5. Resource Constraint

For feasible employment of the detection network, we consider adding a constrain on the computational cost to regularize the searching progress. Directly estimating the forwarding time of the whole network is not feasible since the forwarding time is not an explicit differentiable function of the architecture parameters. In this paper, we consider three indices for $C(\alpha, \beta)$: 1) the parameter size; 2) the number of float-point operations (FLOPs); 3) the memory access cost (MAC). We further consider MAC since MAC can distinguish identity and none.

Then the resource constraint can be added as a regularizer in the objective function:

$$\mathcal{L}(w, \alpha, \beta) = \mathcal{L}_{model}(w, \alpha, \beta) + \lambda C(\alpha, \beta)$$

where $C(\alpha, \beta)$ is the resource constraint associated with architecture parameters α, β. Since $C(\alpha, \beta)$ can be decomposed to each operation and it is linear in terms of all softmax architecture parameters α, β:

$$C(\alpha, \beta) = \sum_{i,j,k,l} \alpha_{i\to j}^{kl} C(O_{i\to j}^{kl}) + \sum_{i,j,k} \beta_{i\to i}^{k} C(O_{i\to j}^{k}),$$

where $C(O_{i\to j}^{k})$ is the computational cost of each operation with a sum of normalized parameter size, FLOPs and MAC. Note that the $C(\alpha, \beta)$ is differentiable with respected to α, β. Thus the resource constraint can be easily applied in the current optimization.

4. Experiments

4.1. Architecture Search Implementation Details

We consider a total of $L = 2$ layers and $M = 2$ cells for searching the neck and head in the network. The Auto-fusion search space has 7.9×10^{28} unique paths, and Auto-head has 1.9×10^{25}. Note that this search space is larger than DARTS [38] since we do not use fixed cell.

ResNet-50 [20] pre-trained on ImageNet [52] is used as our backbone network. For Auto-fusion, the number of output channels of each feature level is 256 during searching and training. Max pooling/Bilinear upsampling is used to reduce/increase spatial size between levels. For Auto-head,

method	Search On	Params (neck)/M	Params (head)/M	Params (total)/M	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
FPN	-	3.34	14.31	41.76	38.6	60.4	42.0	23.7	42.4	49.2
Auto-fusion$_S$	COCO	1.39	14.31	39.8	39.3	61.7	42.5	24.9	43.5	50
Auto-fusion$_M$	COCO	1.61	14.31	40.02	39.6	61.9	42.8	24.6	44.0	50.3
Auto-fusion$_L$	COCO	1.83	14.31	40.24	39.7	62.0	42.9	25.7	43.8	50.4
Auto-head$_S$	COCO	3.34	2.42	29.87	39.1	59.9	42.6	23.2	43.3	49.1
Auto-head$_M$	COCO	3.34	6.93	34.37	39.9	60.5	43.4	25.6	44.2	50.2
Auto-head$_L$	COCO	3.34	6.93	34.37	40.2	60.6	44.2	25.3	44.0	51.2
Auto-FPN	COCO	**1.61**$^{-52\%}$	**6.93**$^{-52\%}$	**32.64**$^{-22\%}$	**40.5**$^{+1.9}$	**61.5**$^{+1.1}$	**43.8**$^{+1.8}$	**25.6**$^{+1.9}$	**44.9**$^{+2.5}$	**51.0**$^{+1.8}$
Auto-FPN	BDD	1.61	5.7	31.41	39.2	60.7	42.3	24.1	43.4	49.9
Auto-FPN	VOC	1.83	5.39	31.32	38.9	60.4	41.9	23.3	43.0	49.6

Table 2. Comparison of mean Average Precision (mAP) and parameter size on COCO (*minval*). The backbones are ResNet-50.

the output of each cell has 512 channels. The output of the last cell goes through a global average pooling to become a vector and then fed into the classification and bbox regression. Adaptive feature pooling is applied as [39]. Batch normalization is not used in both modules during searching.

To evaluate the our methods on different domains, we conduct architecture search on the PASCAL VOC [14], COCO [34] and BDD [60] for detection. We conduct all experiments using Pytorch [44, 10], 8 V100 cards on a single server. During searching, we augment with flipped images and multi-scaling (pixel size=$\{400 \sim 800\}$). We randomly select half of the images in training set as trainA, and the other half as trainB. During searching and training, the batch size is 2. When learning model parameters, SGD optimizer with initial learning rate 0.02 is used, cosine learning rate 0.02 to 0.001, momentum 0.9 and 10^{-4} as weight decay. For learning the architecture parameters, we use Adam optimizer [25] with learning rate 0.003 and weight decay 0.001. The architecture searching is conducted for a total of 24 epochs for COCO and BDD and 36 for VOC. We also tried longer epochs, but did not observe benefit. We begin optimizing architecture parameters after the 1/3 of the total epochs. Each module is searched sequentially due to the GPU memory constraint. Our final model Auto-FPN is the optimal combination of the searched architecture of Auto-fusion and Auto-head. It takes about 11 hours for PASCAL VOC, 2 days for COCO and 1 day for BDD to complete searching of each module. We consider three settings of λ for the resource constraints for our model (small: $\lambda = 0.2$, middle: 0.02, big: 0.002, and half size for VOC).

Figure 4 visualizes an example of the neck (left) and head architecture (right) found by our Auto-FPN on COCO with middle resource constraint. For the optimal neck architecture, the information of each feature level densely spreads to other feature levels with identity and dilated convolution with rates 2 and 3, suggesting the importance of large receptive fields. For the optimal head architecture, Auto-head found a structure consisting of conv5x5 and allowed interaction between intermediate nodes with fully

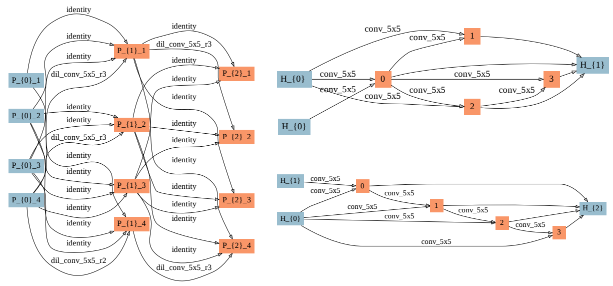


Figure 4. Example of the neck (left) architecture and head architecture (right) found by our Auto-FPN on COCO.

utilization of extracted features. More structures found on different datasets and λ are in the supplementary material.

4.2. Object Detection Results

We first search the architecture and then evaluate the performance on PASCAL VOC [14], COCO [34] and BDD [60]. Furthermore, we adapt the searched architecture on two new datasets: Visual Genome [27] and ADE [62] for testing the adaptation of new domains. Instead of picking the best performance with repeated running as DARTS [38] (4 times) and Auto-deeplab [35] (10 times), we report the performance of **running only once**.

After identifying the optimal architecture, stochastic gradient descent (SGD) is performed to train the full model on 8 GPUs with 2 images on each GPU. 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. For all the datasets, we train 24 epochs. During training, we augment with flipped images and multi-scaling (pixel size=$\{400 \sim 1300\}$). During testing, pixel size=800 is used (VOC: 600). For evaluation, we adopt the metrics from COCO detection evaluation criteria [34] which is mean Average Precision (mAP) across IoU thresholds from 0.5 to 0.95 and Average Recall (AR) with different scales.

PASCAL VOC (VOC) [14] contains 20 object classes. For PASCAL VOC, training is performed on the union of VOC 2007 trainval and VOC 2012 trainval (10K images)

Dataset	method	Search On	Params (neck)/M	Params (head)/M	Params (total)/M	mAP
PASCAL VOC	FPN	-	3.34	14.00	41.44	79.7
	Auto-fusion$_S$	VOC	1.54	14.00	39.65	80.7
	Auto-fusion$_M$	VOC	1.75	14.00	39.86	82.2
	Auto-fusion$_L$	VOC	1.83	14.00	39.93	82.7
	Auto-head$_S$	VOC	3.34	2.54	29.98	80.5
	Auto-head$_M$	VOC	3.34	5.24	32.69	81.3
	Auto-head$_L$	VOC	3.34	5.77	33.21	81.2
	Auto-FPN	VOC	**1.83**$^{-45\%}$	**5.24**$^{-63\%}$	**31.17**$^{-25\%}$	**81.8**$^{+2.1}$
	Auto-FPN	COCO	1.61	6.77	32.49	81.3
	Auto-FPN	BDD	1.61	5.54	31.26	81.4
BDD	FPN	-	3.34	13.95	41.39	33.0
	Auto-fusion$_S$	BDD	1.32	13.95	39.37	33.8
	Auto-fusion$_M$	BDD	1.61	13.95	39.66	33.8
	Auto-fusion$_L$	BDD	2.04	13.95	40.09	33.9
	Auto-head$_S$	BDD	3.34	1.44	28.88	33.8
	Auto-head$_M$	BDD	3.34	5.52	32.96	34.0
	Auto-head$_L$	BDD	3.34	6.34	33.78	33.9
	Auto-FPN	BDD	**1.61**$^{-52\%}$	**5.52**$^{-60\%}$	**31.23**$^{-25\%}$	**33.9**$^{+0.9}$
	Auto-FPN	COCO	1.61	6.75	32.46	33.7
	Auto-FPN	VOC	1.83	5.21	31.14	33.3

Table 3. Comparison of mean Average Precision (mAP) on PASCAL VOC and BDD. Auto-FPN is the combination of two modules. The backbones of all the models are ResNet-50.

and evaluation is on VOC 2007 test (4.9K images). We only report mAP scores using IoU at 0.5. **COCO** [34] contains 80 object classes. COCO 2017 contains 118k images for training, 5k for evaluation. **Berkeley Deep Drive (BDD)** [60] is an autonomous driving dataset with 10 object classes. BDD contains about 70K images for training and 10K for evaluation. For **Visual Genome (VG)** [27], we use the synsets [52] instead of the raw names of the categories due to inconsistent label annotations, following [22, 12, 23]. We consider 1000 most frequent classes. We split the remaining 92K images with objects on these class sets into 88K and 5K for training and testing, following [23]. For **ADE**[62], we use 20K images for training and 1K images for testing with 445 categories, following [12, 23].

Comparison with the state-of-the-art. Tables 2 and 3 show the results of the architecture searched from COCO, VOC and BDD by our methods. Auto-FPN achieves significant gains than the baseline FPN with less parameters on all the three detection benchmarks. Auto-FPN achieves an overall AP of 40.5% compared to 38.6% by FPN on COCO, 81.8% compared to 79.7% on VOC, and 33.9% compared to 33.0% on ADE, respectively. In terms of parameter size, the searched architecture is about 40%-50% smaller in neck and 50%-60% smaller in head. It can also be found that searching directly on the target dataset would be better than transferring architecture searched from other dataset.

To compare with state-of-the-art methods, we also implement different backbones with the searched architecture from COCO in Table 4. We report the accuracy numbers of the competing methods directly from the original paper except FPN. Our implementation of the baseline FPN with ResNet-101 has higher accuracy than that in original works (40.7% versus 36.2% [32]). As can be seen, Auto-FPN performs 4.2% better than the baseline FPN with less parameters and all the other hand-crafted competitors with same backbone. Note that most competitive models are bigger than FPN. The qualitative results can be found in Figure 5.

%	Method	backbone	mAP
COCO	Faster-RCNN [50]	ResNet-101	34.9
	FPN [32]	ResNet-101	40.7
	Relation Network [21]	ResNet-101	38.8
	RetinaNet [33]	ResNet-101	39.1
	DetNet [31]	DetNet-59	40.2
	GA-Faster RCNN [56]	ResNet-50	39.8
	PANet [39]	ResNet-50	39.8
	TridentNet [29]	ResNet-101	42.0
	Auto-FPN	ResNet-50	40.5
	Auto-FPN	ResNet-101	42.5$^{+1.8}$
	Auto-FPN	ResNeXt-101	**44.3**

Table 4. Comparison of mAP of the single-model on COCO.

Data	Method	Search From	Params (neck)/M	Params (head)/M	Params (total)/M	mAP
VG	FPN	-	3.34	19.03	46.47	7.0
	Auto-FPN	VOC	1.83	7.75	33.68	7.0
	Auto-FPN	COCO	**1.61**$^{-52\%}$	**9.29**$^{-51\%}$	**35.00**$^{-25\%}$	**7.2**$^{+0.2}$
	Auto-FPN	BDD	1.61	8.06	33.77	7.0
ADE	FPN	-	3.34	16.18	43.63	10.5
	Auto-FPN	VOC	1.83	6.33	32.26	12.0
	Auto-FPN	COCO	**1.61**$^{-52\%}$	**7.86**$^{-51\%}$	**33.58**$^{-28\%}$	**12.1**$^{+1.6}$
	Auto-FPN	BDD	1.61	6.63	32.35	11.8

Table 5. Transferability of Auto-FPN on Visual Genome (VG) and ADE. We transferred the searched architecture from VOC, COCO and BDD to the new dataset and evaluated the performance on VG and ADE. Backbones of all the models are ResNet-50.

Architecture Transfer: VG and ADE. To evaluate the domain transferability of our Auto-FPN, we transfer the searched architecture from the above three datasets to VG and ADE, as shown in Table 5. With the architecture found in COCO, our method boosts 0.2% mAP for VG and 1.6% for ADE with about 25% less parameters. The better transferability of the model found in COCO may be due to more training data and categories in COCO.

Model extension: Auto-fusion for SSD. To evaluate our model extension, we implement our Auto-fusion module on the top of SSD512 [40] and search on the VOC and COCO. The training settings follow SSD512 [40]. We obtain a smaller model with better performance (increase 0.1% mAP on VOC and 2.5% on COCO). Feature fusion seems to be important for COCO and our model with neck is even 12%

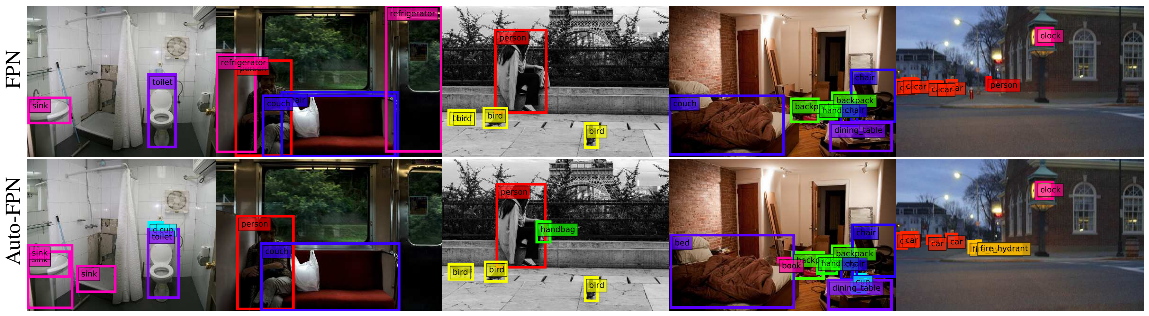


Figure 5. Qualitative result comparison on COCO between FPN and Auto-FPN. The backbones are ResNet-50.

Data	Method	Search On	Params (neck&head)/M	Params (total)/M	mAP
VOC	SSD [40]	-	3.69	27.19	79.8
	Auto-fusion$_S$	VOC	3.32	26.82	79.8
	Auto-fusion$_M$	VOC	**3.54**$^{-4\%}$	**27.04**$^{-0.5\%}$	**79.9**$^{+0.1}$
	Auto-fusion$_L$	VOC	6.92	34.43	79.7
COCO	SSD-VGG19 [40]	-	12.54	36.04	29.3
	SSD-ResNet101 [15]	-	-	-	31.2
	Auto-fusion$_S$	COCO	8.30	31.80	30.8
	Auto-fusion$_M$	COCO	**8.37**$^{-33\%}$	**31.88**$^{-12\%}$	**31.8**$^{+2.5}$
	Auto-fusion$_L$	COCO	9.86	33.36	31.6

Table 6. Auto-fusion for one-stage detector SSD on VOC and COCO. We further apply our Auto-fusion on the top of baseline method SSD512. Backbones of Auto-fusion are VGG19.

smaller than SSD since the output channel is 256 (smaller than the output in SSD512).

4.3. Comparison of different training strategies and more NAS baselines

We found that training a good gradient based NAS is not very easy. Here we share our exploration of tricks and different training strategies as shown in Table 7: 1) Starting optimizing architecture parameters in the middle of training can improve the results by 1%; 2) Freezing the backbone parameters during searching not only accelerates the training but also improves the performance; 3) Searching with BN will decrease the performance by 1.7%; 4) During searching for the head, loading the pretrained neck will boost the performance by 2.9%; 5) Without resource constraints, our method becomes larger with only a small improvement in neck but no improvement in head.

We further added more NAS baselines to validate the performance under current search space: a) Random Search: We randomly sample architectures from the current search space and conduct a fully train for each architecture under the same training setting in our experiments; b) Evolutionary Algorithm: EA search is implemented with an early-stop strategy (due to limited time, 6 epochs for VOC and 4 epochs for COCO) and the best architecture is selected to perform a final fully train; c) SNAS[58]: We adopt the new architecture parameter update according to [58]. The final trained results are showed in Table 8. Compared to other NAS baselines, our method can find better architectures with less time.

Training method	Auto-fusion		Auto-head	
	params	mAP	params	mAP
baseline	1.75M	82.2	5.24M	81.3
w/o step training	1.97M	81.7	5.47M	80.7
w/o frozen backbone	2.15M	65.2	1.10M	78.2
w BN	2.08M	80.5	2.82M	78.3
w/o pretrained neck	-	-	1.42M	78.4
w/o resource constraint	1.9M	82.6	6.51M	80.9

Table 7. Comparison of using different training strategies during searching on VOC. "Step training" is starting optimizing architecture parameters in the middle of searching.

	Search Method	No. arch searched	Search time (GPU days)	Average/Best mAP of searched arch
PASCAL VOC	Random	20	~ 18.3	76.4/80.3
	Evolutionary	60	~ 20.0	81.1
	SNAS[58]	1	~ 0.8	81.0
	Auto-FPN	1	**~ 0.8**	**81.8**
MS-COCO	Random	10	~ 130.0	36.4/38.2
	Evolutionary	30	~ 68.3	38.6
	SNAS[58]	1	~ 16.0	37.9
	Auto-FPN	1	**~ 16.0**	**40.5**

Table 8. More NAS baselines upon current search space on VOC and COCO.

5. Conclusion

In this work, we propose Auto-FPN to search for an efficient and better architecture for two important components of the detection system which is one of the first attempts to extend NAS to object detection. The search space is specially designed for detection and flexible enough to cover many popular designs of detectors. Auto-FPN outperforms the state-of-the- art detection methods but with fewer parameters.

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] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *CVPR*, 2016. 3

[3] 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

[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, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. 2018. 2

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

[7] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, 2016. 3

[8] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. 1

[9] 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

[10] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. mmdetection. https://github.com/open-mmlab/mmdetection, 2018. 6

[11] 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 *NIPS*, 2018. 1, 2

[12] Xinlei Chen, Li-Jia Li, Li Fei-Fei, and Abhinav Gupta. Iterative visual reasoning beyond convolutions. In *CVPR*, 2018. 7

[13] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 1, 2

[14] 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, 2010. 2, 6

[15] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. In *ICCV*, 2017. 3, 8

[16] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 3

[17] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 3

[18] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, 2018. 1, 3

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *TPAMI*, 37(9):1904–1916, 2015. 1

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 3, 4, 5

[21] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *CVPR*, 2018. 7

[22] Ronghang Hu, Piotr DollÃ¡r, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to segment every thing. In *CVPR*, 2018. 7

[23] ChenHan Jiang, Hang Xu, Xiaodan Liang, and Liang Lin. Hybrid knowledge routed modules for large-scale object detection. In *NIPS*. 2018. 7

[24] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015. 2

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[26] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*, 2016. 3

[27] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *IJCV*, 2016. 2, 6, 7

[28] Bo Li, Tianfu Wu, Lun Zhang, and Rufeng Chu. Auto-context r-cnn. *arXiv preprint arXiv:1807.02842*, 2018. 1

[29] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. *arXiv preprint arXiv:1901.01892*, 2019. 1, 2, 3, 4, 7

[30] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. In *CVPR*, 2017. 1, 2, 5

[31] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. In *ECCV*, 2018. 7

[32] 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, 4, 5, 7

[33] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. 1, 3, 4, 7

[34] 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. 2, 6, 7

[35] 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. 1, 2, 6

[36] 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

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

[38] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2, 4, 5, 6

[39] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 1, 2, 3, 4, 6, 7

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 1, 2, 3, 4, 7, 8

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *CVPR*, 2017. 2

[42] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 3

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

[44] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017. 6

[45] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 1, 2, 5

[46] 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

[47] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. 1, 2

[48] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *CVPR*, 2017. 1

[49] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2, 5, 7

[51] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 3

[52] 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*, 2015. 3, 5, 7

[53] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016. 3

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3

[55] 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

[56] Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring. *arXiv preprint arXiv:1901.03278*, 2019. 7

[57] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 1, 3

[58] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019. 2, 8

[59] Ke Yan, Mohammadhadi Bagheri, and Ronald M Summers. 3d context enhanced region-based convolutional neural network for end-to-end lesion detection. In *MICCAI*, 2018. 1

[60] 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. 2, 6, 7

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

[62] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 2, 6, 7

[63] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2

[64] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2, 5