NAS-FCOS: Fast Neural Architecture Search for Object Detection[*]

Ning Wang[1], Yang Gao[1], Hao Chen[2], Peng Wang[1], Zhi Tian[2], Chunhua Shen[2]

[1]School of Computer Science, Northwestern Polytechnical University, China
[2]School of Computer Science, The University of Adelaide, Australia

Abstract

The success of deep neural networks relies on significant architecture engineering. Recently neural architecture search (NAS) has emerged as a promise to greatly reduce manual effort in network design by automatically searching for optimal architectures, although typically such algorithms need an excessive amount of computational resources, e.g., a few thousand GPU-days. To date, on challenging vision tasks such as object detection, NAS, especially fast versions of NAS, is less studied. Here we propose to search for the decoder structure of object detectors with search efficiency being taken into consideration. To be more specific, we aim to efficiently search for the feature pyramid network (FPN) as well as the prediction head of a simple anchor-free object detector, namely FCOS [24], using a tailored reinforcement learning paradigm. With carefully designed search space, search algorithms and strategies for evaluating network quality, we are able to efficiently search a top-performing detection architecture within 4 days using 8 V100 GPUs. The discovered architecture surpasses state-of-the-art object detection models (such as Faster R-CNN, RetinaNet and FCOS) by 1.5 to 3.5 points in AP on the COCO dataset, with comparable computation complexity and memory footprint, demonstrating the efficacy of the proposed NAS for object detection.

1 Introduction

Object detection is one of the fundamental tasks in computer vision, and has been researched extensively. In the past few years, state-of-the-art methods for this task are based on deep convolutional neural networks (such as Faster R-CNN [20], RetinaNet [11]), due to their impressive performance. Typically, the designs of object detection networks are much more complex than those for image classification, because the former need to localize and classify multiple objects in an image simultaneously while the latter only need to output image-level labels. Due to its complex structure and numerous hyper-parameters, designing effective object detection networks is more challenging and usually needs much manual effort.

On the other hand, Neural Architecture Search (NAS) approaches [4,17,32] have been showing impressive results on automatically discovering top-performing neural network architectures in large-scale search spaces. Compared to manual designs, NAS methods are data-driven instead of

experience-driven, and hence need much less human intervention. As defined in [3], the workflow of NAS can be divided into the following three processes: 1) sampling architecture from a search space following some search strategies; 2) evaluating the performance of the sampled architecture; and 3) updating the parameters based on the performance.

One of the main problems prohibiting NAS from being used in more realistic applications is its search efficiency. The evaluation process is the most time consuming part because it involves a full training procedure of a neural network. To reduce the evaluation time, in practice a proxy task is often used as a lower cost substitution. In the proxy task, the input, network parameters and training iterations are often scaled down to speedup the evaluation. However, there is often a performance gap for samples between the proxy tasks and target tasks, which makes the evaluation process biased. How to design proxy tasks that are both accurate and efficient for specific problems is a challenging problem. Another solution to improve search efficiency is constructing a supernet that covers the complete search space and training candidate architectures with shared parameters [15; 18]. However, this solution leads to significantly increased memory consumption and restricts itself to small-to-moderate sized search spaces.

To our knowledge, studies on efficient and accurate NAS approaches to object detection networks are rarely touched, despite its significant importance. To this end, we present a fast and memory saving NAS method for object detection networks, which is capable of discovering top-performing architectures within significantly reduced search time. Our overall detection architecture is based on FCOS [24], a simple anchor-free one-stage object detection framework, in which the feature pyramid network and prediction head are searched using our proposed NAS method.

Our main contributions are summarized as follows.

- In this work, we propose a fast and memory-efficient NAS method for object detection networks, with carefully designed proxy tasks, search space and evaluation strategies, which is able to find top-performing architectures over 3,000 architectures using 28 GPU-days only.

  Specifically, this high efficiency is enabled with the following designs.

  − Employing an efficient anchor-free one-stage detection framework with simple post processing;

  − Developing a fast proxy task training scheme by caching powerful pre-trained backbone features;

− Using a more discriminative criterion for evaluation of searched architectures.

- We empirically demonstrate that the performance of the recently introduced anchor-free one-stage detector FCOS can be considerably improved with the automatically discovered feature pyramid architecture and the prediction head part. We term the discovered architecture NAS-FCOS.

- We show that the overall structure of NAS-FCOS is general and flexible in that it can be equipped with various backbones including MobileNetV2, ResNet-50, ResNet-101 and ResNeXt-101, and surpasses state-of-the-art object detection algorithms using comparable computation complexity and memory footprint. More specifically, our model can improve the AP by $1.5 \sim 3.5$ points on all above models comparing to their FCOS counterparts.

2 Related Work

2.1 Object Detection

The frameworks of deep neural networks for object detection can be roughly categorized into two types: one-stage detectors [12] and two-stage detectors [6, 20].

Two-stage detection frameworks first generate class-independent region proposals using a region proposal network (RPN), and then classify and refine them using extra detection heads. In spite of achieving top performance, the two-stage methods have noticeable drawbacks: they are computationally expensive and have many hyper-parameters that need to be tuned to fit a specific dataset.

In comparison, the structures of one-stage detectors are much simpler. They directly predict object categories and bounding boxes at each location of feature maps generated by a single CNN backbone.

Note that most state-of-the-art object detectors (including both one-stage detectors [12, 16, 19] and two-stage detectors [20]) make predictions based on anchor boxes of different scales and aspect ratios at each convolutional feature map location. However, the usage of anchor boxes may lead to high imbalance between object and non-object examples and introduce extra hyper-parameters. More recently, anchor-free one-stage detectors [9, 10, 24, 29, 30] have attracted increasing research interests, due to their simple fully convolutional architectures and reduced consumption of computational resources.

2.2 Neural Architecture Search

NAS is usually time consuming. We have seen great improvements from $24,000$ GPU-days [32] to 0.2 GPU-day [28]. The trick is to first construct a supernet containing the complete search space and train the candidates all at once with bi-level optimization and efficient weight sharing [13, 15].But the large memory allocation and difficulties in approximated optimization prohibit the search for more complex structures.

Recently researchers [1, 5, 23] propose to apply single-path training to reduce the bias introduced by approximation and model simplification of the supernet. DetNAS [2] follows this idea to search for an efficient object detection architecture. One limitation of the single-path approach is that the search space is restricted to a sequential structure. Single-path sampling and straight through estimate of the weight gradients introduce large variance to the optimization process and prohibit the search for more complex structures under this framework. Within this very simple search space, NAS algorithms can only make trivial decisions like kernel sizes for manually designed modules.

Object detection models are different from single-path image classification networks in their way of merging multi-level features and distributing the task to parallel prediction heads. Feature pyramid networks (FPNs) [4, 8, 11, 14, 27], designed to handle this job, plays an important role in modern object detection models. NAS-FPN [4] targets on searching for an FPN alternative based on one-stage framework RetinaNet [12]. Feature pyramid architectures are sampled with a recurrent neural network (RNN) controller. The RNN controller is trained with reinforcement learning (RL). However, the search is very time-consuming even though a proxy task with ResNet-10 backbone is trained to evaluate each architecture.

To speed up reward evaluation of RL-based NAS, the work of [17] proposes to use progressive tasks and other training acceleration methods. By caching the encoder features, they are able to train semantic segmentation decoders with very large batch sizes very efficiently. In the sequel of this paper, we refer to this technique as fast decoder adaptation. However, directly applying this technique to object detection tasks does not enjoy similar speed boost, because they are either not in using a fully-convolutional model [11] or require complicated post processing that are not scalable with the batch size [12].

To reduce the post processing overhead, we resort to a recently introduced anchor-free one-stage framework, namely, FCOS [24]. In our exploratory experiments, *we found that the processing time of anchor-box matching in RetinaNet can take as much as 50% of the entire search time.* Compared to its anchor-based counterpart, FCOS significantly reduces the training memory footprint while being able to improve the performance. Different from NAS-FPN, we also search the prediction head. Note that our proposed approach is orders of magnitude faster than NAS-FPN [4] in terms of search efficiency.

3 Our Approach

In our work, we search for anchor-free fully convolutional detection models with fast decoder adaptation. Thus, NAS methods can be easily applied.

3.1 Problem Formulation

We base our search algorithm upon a one-stage framework FCOS due to its simplicity. Our training tuples $\{(\mathbf{x}, Y)\}$ consist of input image tensors $\mathbf{x}$ of size $(3 \times H \times W)$ and FCOS output targets Y in a pyramid representation, which

is a list of tensors $\mathbf{y}_l$ each of size $((K + 4 + 1) \times H_l \times W_l)$ where $H_l \times W_l$ is feature map size on level p of the pyramid. $(K + 4 + 1)$ is the output channels of FCOS, the three terms are length-K one-hot classification labels, 4 bounding box regression targets and 1 centerness factor respectively.

The network $g : \mathbf{x} \rightarrow \hat{Y}$ in original FCOS consists of three parts, a backbone b, FPN f and multi-level subnets we call prediction heads h in this paper. First backbone $b : \mathbf{x} \rightarrow C$ maps the input tensor to a set of intermediate-leveled features $C = \{\mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5\}$, with resolution $(H_i \times W_i) = (H/2^i \times W/2^i)$. Then FPN $f : C \rightarrow P$ maps the features to a feature pyramid $P = \{\mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7\}$. Then the prediction head $h : \mathbf{p} \rightarrow \mathbf{y}$ is applied to each level of P and the result is collected to create the final prediction. To avoid overfitting, same h is often applied to all instances in P.

Since objects of different scales require different effective receptive fields, the mechanism to select and merge intermediate-leveled features C is particularly important in object detection network design. Thus, most researches [16, 20] are carried out on designing f and h while using widely-adopted backbone structures such as ResNet [7]. Following this principle, our search goal is to decide when to choose which features from C and how to merge them.

Similar to [17], we reuse the parameters in b pretrained on target dataset and search for the optimal structures after that. Following the nomenclature in that paper, we call the network components to search for, namely f and h, together the decoder structure for the objection detection network.

f and h take care of different parts of the detection job. f extracts features targeting different object scales in the pyramid representations P, while h is a unified mapping applied to each feature in P to avoid overfitting. In practice, people seldom discuss the possibility of using a more diversified f to extract features at different levels or how many layers in h need to be shared across the levels. In this work, we use NAS as an automatic method to test these possibilities.

3.2 Search Space

The building block for our decoder is similar to [17]. We replace the cell structure with atomic operations to provide even more flexibility. To construct one basic block, we first choose two layers $\mathbf{x}_1, \mathbf{x}_2$ from the sampling pool X at `id1`, `id2`, then two operations `op1`, `op2` are applied to each of them and an aggregation operation `agg` merges the two output into one feature. The structure of the basic block is shown in Fig. 1. To build a deep decoder structure, we apply multiple basic blocks with their outputs added to the sampling pool. Our basic block $bb_t : X_{t-1} \rightarrow X_t$ at time step t transforms the sampling pool X_{t-1} to $X_t = X_{t-1} \cup \{\mathbf{x}_t\}$, where $\mathbf{x}_t$ is the output of bb_t.

We rely on a similar set of operations as for semantic segmentation [17] (Table 1). We include only separable/depthwise convolutions so that the decoder can be efficient. In order to enable the decoder to apply convolutional filters on irregular grids, here we have also included deformable 3×3 convolutions [31]. For the aggregation operations, we include element-wise sum and concatenation followed by a 1×1 convolution.
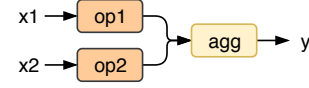
Figure 1 – Basic building block of NAS-FCOS. Two operations `op1`, `op2` are applied to two specified features at `id1`, `id2` and then the outputs are aggregated by `agg`. If the two features have different sizes, the smaller one is scaled up via bilinear upsampling to match the larger one.

ID	Description
0	separable conv 3×3
1	separable conv 3×3 with dilation rate 3
2	separable conv 5×5 with dilation rate 6
3	skip-connection
4	deformable 3×3 convolution

Table 1 – Unary operations used in the search process.

The decoder configuration can be represented by a sequence with three components, FPN configuration, head configuration and weight sharing stages. We provide detailed descriptions to each of them in the following sections. The complete diagram of our decoder structure is shown in Fig. 2.

3.2.1 FPN Search Space

As mentioned above, the FPN f maps the convolutional features C to P. First, we initialize the sampling pool as $X_0 = C$. Our FPN is defined by applying the basic block 7 times to the sampling pool, $f := bb_1^f \circ bb_2^f \circ \cdots \circ bb_7^f$. To yield pyramid features P, we collect the last three basic block outputs $\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}$ as $\{\mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}$.

To allow shared information across all layers, we use a simple rule to create global features. If there is some dangling layer $\mathbf{x}_t$ which is not sampled by later blocks $\{bb_i^f | i > t\}$ nor belongs to the last three layers $t < 5$, we use element-wise add to merge it to all output features

$$\mathbf{p}_i^* = \mathbf{p}_i + \mathbf{x}_t, \ i \in \{3, 4, 5\}. \tag{1}$$

Same as the aggregation operations, if the features have different resolution, the smaller one is upsampled with bilinear interpolation.

To be consistent with FCOS, $\mathbf{p}_6$ and $\mathbf{p}_7$ are obtained via a 3×3 stride-2 convolution on $\mathbf{p}_5$ and $\mathbf{p}_6$ respectively.

3.2.2 Prediction Head Search Space

Prediction head h maps each feature in the pyramid P to the output of corresponding $\mathbf{y}$, which in FCOS and RetinaNet, consists of four 3×3 convolutions. To explore the potential of the head, we therefore extend a sequential search space for its generation. Specifically, our head is defined as a sequence of six basic operations. Compared with candidate operations in the FPN structures, the head search space has two slight differences. First, we add standard convolution modules (including conv1x1 and conv3x3) to the head sampling pool for better comparison. Second, we replace all the Batch Normalization (BN) layers to Group Normalization (GN) [25] in the operations sampling pool of head,
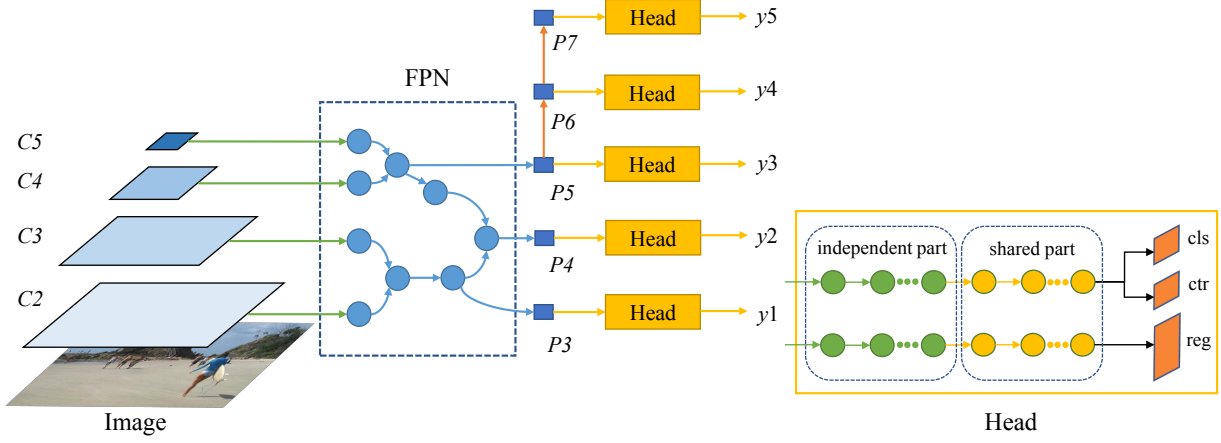
Figure 2 – A conceptual example of our NAS-FCOS decoder. It consists of two sub networks, an FPN f and a set of prediction heads h which have shared structures. One notable difference with other FPN-based one-stage detectors is that our heads have partially shared weights. Only the last several layers of the predictions heads (marked as yellow) are tied by their weights. The number of layers to share is decided automatically by the search algorithm. Note that both FPN and head are in our actual search space; and have more layers than shown in this figure. Here the figure is for illustration only.

considering that head needs to share weights between different levels, which causes BN invalid. The final output of head is the output of the last (sixth) layer.

3.2.3 Searching for Head Weight Sharing

To add even more flexibility and understand the effect of weight sharing in prediction heads, we further add an index i as the location where the prediction head starts to share weights. For every layer before stage i, the head h will create independent set of weights for each FPN output level, otherwise, it will use the global weights for sharing purpose.

Considering the independent part of the heads being extended FPN branch and the shared part as head with adaptive-length, we can further balance the workload for each individual FPN branch to extract level-specific features and the prediction head shared across all levels.

3.3 Search Strategy

Similar strategy as [17] is applied to the search process. We rely on an LSTM-based controller to predict the full configuration. We consider using a progressive search strategy rather than the joint search for both FPN structure and prediction head part, since the former requires less computing resources and time cost than the latter. The training dataset is randomly split into a meta-train D_t and meta-val D_v subset. To speed up the training, we fix the backbone network and cache the pre-computed backbone output C. This makes our single architecture training cost independent from the depth of backbone network. Taking this advantage, we can apply much more complex backbone structures and utilize high quality multilevel features as our decoder's input. Speedup techniques such as Polyak weight averaging are also applied during the training.

The most widely used detection metric is average precision (AP). However, due to the difficulty of object detection task, at the early stages, AP is too low to tell the good architectures from the bad ones, which makes the controller take much more time to converge. To make the architec-

ture evaluation process easier even at the early stages of the training, we therefore use negative loss sum as the reward instead of average precision:

$$R(a) = - \sum_{(x,Y) \in D_v} (L_{cls}(x, Y|a) + L_{reg}(x, Y|a) + L_{ctr}(x, Y|a)) \tag{2}$$

where L_{cls}, L_{reg}, L_{ctr} are the three loss terms in FCOS. Gradient of the controller is estimated via proximal policy optimization (PPO) [22].

4 Experiments

4.1 Implementation Details
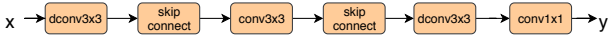
4.1.1 Searching Phase

We design a fast proxy task for evaluating the decoder architectures sampled in the searching phase. PASCAL VOC is selected as the proxy dataset, which contains 5715 training images with object bounding box annotations of 20 classes. The VOC training set is randomly split into a meta-train set with $4,000$ images and a meta-val set with 1715 images. For each sampled architecture, we train it on meta-train and compute the reward (2) on meta-val. Input images are resized to short size 384 and then randomly cropped to 384×384. Target object sizes of interest are scaled correspondingly. We use Adam optimizer with learning rate $8e{-}4$ and batch size 200. Polyak averaging is applied with the decay rates of 0.9. The decoder is evaluated after 300 iterations. As we use fast decoder adaptation, the backbone features are fixed and cached during the search phase. To enhance the cached backbone features, we first initialize them with pre-trained weights provided by open-source implementation of FCOS[1] and then finetune on VOC using the training strategies of FCOS. Note that the above finetuning process is only performed once at the begining of the search phase.

---

[1] https://tinyurl.com/FCOSv1
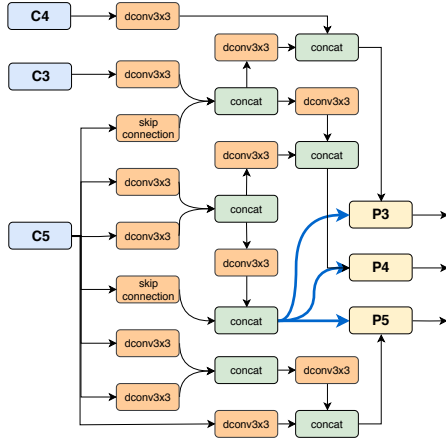
Figure 3 – Our discovered FPN structure. C_2 is omitted from this figure since it is not chosen by this particular structure during the search process.

Figure 4 – Our discovered Head structure.

Figure 5 – Trend graph of head weight sharing during search. The co-ordinates in the horizontal axis represent the number of the statistical period. A period consists of 50 head structures. The vertical axis represents the proportion of heads that fully share weights in 50 structures.

A progressive strategy is used for the search of f and h. We first search for the FPN part and retain the original head. All operations in the FPN structure have 64 output channels. The decoder inputs C are resized to fit output channel width of FPN via 1×1 convolutions. After this step, a searched FPN structure is fixed and the second stage searching for the head will be started based on it. Most parameters for searching head are identical to those for searching FPN structure, with the exception that the output channel width is adjusted from 64 to 128 to deliver more information.
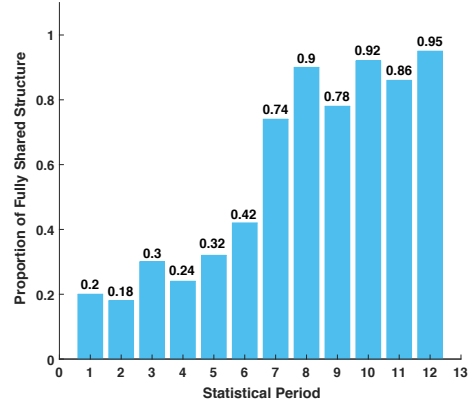
For the FPN search part, the controller model nearly converged after searching over 2.8K architectures on the proxy task. Then, the top-20 best performing architectures on the proxy task are selected for the next full training phase. For the head search part, we choose the best searched FPN among the top-20 architectures and pre-fetch its features. It takes about 600 rounds for the controller to nearly converge, which is much faster than that for searching FPN architectures. After that, we select for full training the top-10 heads that achieve best performance on the proxy task. In total, the whole search phase can be finished within 4 days using 8 V100 GPUs.

4.1.2 Full Training Phase

In this phase, we fully train the searched models on the MS COCO training dataset, and select the best one by evaluating them on MS COCO validation images. Our training setting is similar to that for the original FCOS. Input images are resized to short size 800 and the maximum long side is set to be 1333. The models are trained using 4 V100 GPUs with batch size 16 for 90K iterations. The initial learning rate is 0.01 and reduces to one tenth at the 60K-th and 80K-th iterations.

4.2 Search Results

The best FPN structure is illustrated in Fig. 3. The controller identifies that deformable convolution and concate-nation are the best performing operations for unary and aggregation respectively. From Fig. 4, we can see that the controller chooses to use 4 operations (with two skip connections), rather than the maximum allowed 6 operations. Note that the discovered "dconv + 1x1 conv" structure achieves a good trade-off between accuracy and FLOPs. Compared with the original head, our searched head has fewer FLOPs/Params (FLOPs 79.24G vs. 89.16G, Params 3.41M vs. 4.92M) and significantly better performance (AP 38.7 vs. 37.4).

We use the searched decoder together with either light-weight backbones such as MobileNet-V2 [21] or more powerful backbones such as ResNet-101 [7] and ResNeXt-101 [26]. To balance the performance and efficiency, we implement three decoders with different computation budgets: one with feature dimension of 128 (@128), one with 256 (@256) and another with FPN channel width 128 and prediction head 256 (@128-256). The results on the COCO test-dev with short side being 800 is shown in Table 2. The searched decoder with feature dimension of 256 (@256) surpasses its FCOS counterpart by 1.5 to 3.5 points in AP under different backbones. The one with 128 channels (@128) has significantly reduced parameters and calculation, making it more suitable for resource-constrained environments. In particular, our searched model with 128 channels and MobileNetV2 backbone suparsses the original FCOS with the same backbone by 0.8 AP points with only 1/3 FLOPS. The third type of decoder (@128-256) achieves a good balance between accuracy and parameters. Note that our searched model outperforms the strongest FCOS variant by 1.4 AP points (46.1 vs. 44.7) with slightly smaller FLOPs and Params.

In order to understand the importance of weight sharing in head, we add the number of layers shared by weights as an object of the search. Fig. 5 shows a trend graph of head weight sharing during search. We set 50 structures as a statistical cycle. As the search deepens, the proportion of fully shared structures increases, indicating that on the multi-scale detection model, head weight sharing is a necessity.

We also demonstrate the comparison with other NAS methods for object detection in Table 3. Our NAS method is

Decoder	Backbone	FLOPs (G)	Params (M)	AP
FPN-RetinaNet @256	MobileNetV2	133.4	11.3	30.8
FPN-FCOS @256	MobileNetV2	105.4	9.8	31.2
NAS-FCOS (ours) @128	MobileNetV2	**39.3**	**5.9**	32.0
NAS-FCOS (ours) @128-256	MobileNetV2	95.6	9.9	33.8
NAS-FCOS (ours) @256	MobileNetV2	121.8	16.1	**34.7**
FPN-RetinaNet @256	R-50	198.0	33.6	36.1
FPN-FCOS @256	R-50	169.9	32.0	37.4
NAS-FCOS (ours) @128	R-50	**104.0**	**27.8**	37.9
NAS-FCOS (ours) @128-256	R-50	160.4	31.8	39.1
NAS-FCOS (ours) @256	R-50	189.6	38.4	**39.8**
FPN-RetinaNet @256	R-101	262.4	52.5	37.8
FPN-FCOS @256	R-101	**234.3**	**50.9**	41.5
NAS-FCOS (ours) @256	R-101	254.0	57.3	**43.0**
FPN-FCOS @256	X-64x4d-101	371.2	89.6	43.2
NAS-FCOS (ours) @128-256	X-64x4d-101	**361.6**	**89.4**	**44.5**
FPN-FCOS @256 w/improvements	X-64x4d-101	371.2	89.6	44.7
NAS-FCOS (ours) @128-256 w/improvements	X-64x4d-101	**361.6**	**89.4**	**46.1**

Table 2 – Results on test-dev set of MS COCO after full training. R-50 and R-101 represents ResNet backbones and X-64x4d-101 represents ResNeXt-101 (64×4d). All networks share the same input image resolution. FLOPs and parameters are being measured on 1088×800, which is the median of the input size on COCO. For RetinaNet and FCOS, we use official models provided by the authors. For our NAS-FCOS, @128 and @256 means that the decoder channel width is 128 and 256 respectively. @128-256 is the decoder with 128 FPN width and 256 head width. The same improving tricks used on the newest FCOS version are used in our model for fair comparison.
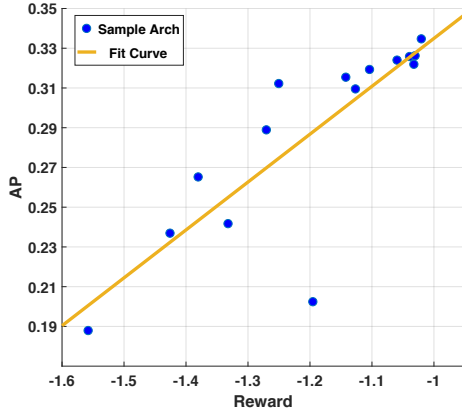
Figure 6 – Correlation between the search reward obtained on the VOC meta-val dataset and the AP evaluated on COCO-val.

able to search for twice more architectures than DetNAS [2] per GPU-day. Note that the AP of NAS-FPN [4] is achieved by stacking the searched FPN 7 times, while we do not stack our searched FPN. Our search model with ResNeXt-101 (64x4d) as backbone outperforms NAS-FPN by 1.3 AP points while using only $1/3$ FLOPs.

We further measure the correlation between rewards obtained during the search process with the proxy dataset and APs attained by same architectures trained on COCO. Specifically, we randomly sample 15 architectures from all the searched structures which are trained on COCO with batch size 16. Since full training on COCO is time-consuming, we reduce the iterations to 60K. The model is then evaluated on the COCO 2017 validation set. As visible in Fig. 6, there is a strong correlation between search rewards and APs obtained from COCO. Poor- and well-performing architectures can be distinguished by the rewards on the proxy task very well.

4.3 Ablation Study

4.3.1 Design of Reinforcement Learning Reward

As we discussed above, it is common to use widely accepted indicators as rewards for specific tasks in the search, such as mIOU for segmentation and AP for object detection. However, we found that using AP as reward did not show a clear upward trend in short-term search rounds (blue curve in Fig. 7). We further analyze the possible reason to be that the controller tries to learn a mapping from the decoder to the reward while the calculation of AP itself is complicated, which makes it difficult to learn this mapping within a limited number of iterations. In comparison, we clearly see the increase of AP with the validation loss as RL rewards (red curve in Fig. 7).

4.3.2 Effectiveness of Search Space

To further discuss the impact of the search spaces f and h, we design three experiments for verification. One is to search f with the original head being fixed, one is to search h with the original FPN being fixed and another is to search the entire decoder (f+h). As shown in Table 4, it turns out that searching f brings slightly more benefits than searching h only. And our progressive search which combines both f and h achieves a better result.

4.3.3 Impact of Deformable Convolution

As aforementioned, deformable convolutions are included in the set of candidate operations for both f and h, which are able to adapt to the geometric variations of objects. For fair comparison, we also replace the whole standard 3×3 convolutions with deformable 3×3 convolutions in FPN structure of the original FCOS and repeat them twice, mak-

Arch	FLOPs (G)	Search Cost (GPU-day)	Searched Archs	AP
NAS-FPN @256 R-50	>325.0	333×#TPUs	17000	<38.0
NAS-FPN 7@256 R-50	1125.5	333×#TPUs	17000	44.8
DetNAS-FPN-Faster	-	44	2200	40.0
DetNAS-RetinaNet	-	44	2200	33.3
NAS-FCOS (ours) @256 R-50	**189.6**	**28**	3000	39.8
NAS-FCOS (ours) @128-256 X-64x4d-101	361.6	**28**	3000	**46.1**

Table 3 – Comparison with other NAS methods. For NAS-FPN, the input size is 1280×1280 and the search cost should be timed by their number of TPUs used to train each architecture. Note that the FLOPs and AP of NAS-FPN @256 here are from Figure 11 in NAS-FPN [4], and NAS-FPN 7@256 stacks the searched FPN structure 7 times. The input images are resized such that their shorter size is 800 pixels in DetNASNet [2] and our models.

Decoder	Search Space	AP
FPN-FCOS @256	-	37.4
NAS-FCOS @256	h only	38.7
NAS-FCOS @256	f only	38.9
NAS-FCOS @256	$f + h$	**39.8**

Table 4 – Comparisons among different search space with ResNet-50 backbone.
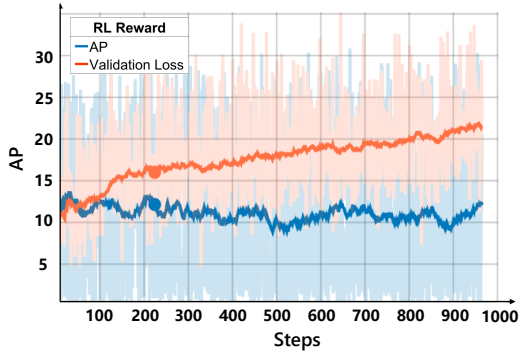
Figure 7 – Comparison of two different RL reward designs. The vertical axis represents AP obtained from the proxy task on the validation dataset.

ing the FLOPs and parameters nearly equal to our searched model. The new model is therefore called DeformFPN-FCOS. It turns out that our NAS-FCOS model still achieves better performance (AP = 38.9 with FPN search only, and AP = 39.8 with both FPN and Head searched) than the DeformFPN-FCOS model (AP = 38.4) under this circumstance.

5 Conclusion

In this paper, we have proposed to use Neural Architecture Search to further optimize the process of designing object detection networks. It is shown in this work that top-performing detectors can be efficiently searched using carefully designed proxy tasks, search strategies and model evaluation metrics. The experiments on COCO demonstrates the efficiency of our discovered model NAS-FCOS and its flexibility to be used with various backbone architectures.

References

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

[2] 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.

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[4] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

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

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[8] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollr. Panoptic feature pyramid networks. *arXiv: Computer Vision and Pattern Recognition*, 2019.

[9] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. Foveabox: Beyond anchor-based object detector. *arXiv preprint arXiv:1904.03797*, 2019.

[10] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2117–2125, 2017.

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object de-

tection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2980–2988, 2017.

[13] 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. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[14] Huanyu Liu, Chao Peng, Changqian Yu, Jingbo Wang, Xu Liu, Gang Yu, and Wei Jiang. An end-to-end network for panoptic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

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

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. SSD: Single shot multibox detector. In *Proc. Eur. Conf. Comp. Vis.*, pages 21–37. Springer, 2016.

[17] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

[19] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proc. Advances in Neural Inf. Process. Syst.*, pages 91–99, 2015.

[21] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4510–4520, 2018.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv: Comp. Res. Repository*, 2017.

[23] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

[24] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. *arXiv preprint arXiv:1904.01355*, 2019.

[25] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[26] Saining Xie, Ross Girshick, Piotr Dollr, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[27] Ting Zhao and Xiangqian Wu. Pyramid feature attention network for saliency detection. *arXiv: Computer Vision and Pattern Recognition*, 2019.

[28] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. BayesNAS: A bayesian approach for neural architecture search. *arXiv preprint arXiv:1905.04919*, 2019.

[29] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

[30] Chenchen Zhu, Yihui He, and Marios Savvides. Feature selective anchor-free module for single-shot object detection. *arXiv preprint arXiv:1903.00621*, 2019.

[31] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. *arXiv preprint arXiv:1811.11168*, 2018.

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

6 Supplementary

6.1 Comparison of APs, FLOPs, Parameters

By adjusting the number of channels in the searched decoder, our model shows its flexibility in balancing precision and computational budgets. For ease of observation, we visualize the relationship between AP and parameters/FLOPs respectively. As can be seen from Fig. 9 and Fig. 8, our model is competitive in both accuracy and efficiency with different backbones.
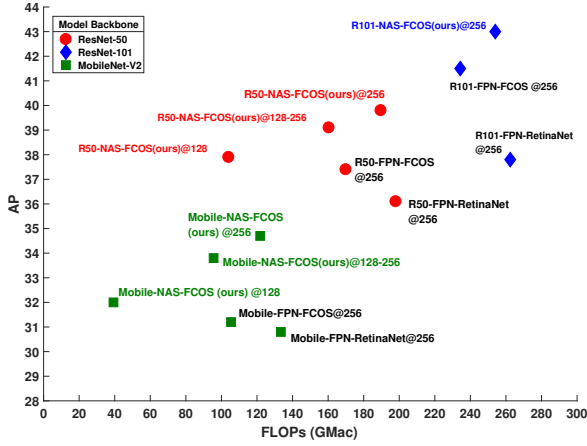
Figure 8 – Diagram of the relationship between FLOPs and AP with different backbones. Points of different shapes represent different backbones. NAS-FCOS@128 has a slight increase in precision which also gains the advantage of computation quantity. One with 256 channels obtains the highest precision with more computation complexity. Using FPN channel width 128 and prediction head 256 (@128-256) offers a trade-off.
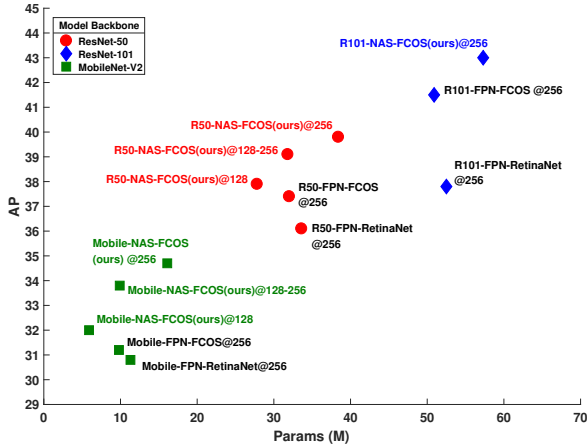

Figure 9 – Diagram of the relationship between parameters and AP with different backbones. Adjusting the number of channels in the FPN structure and head helps to achieve a balance between accuracy and parameters.

6.2 Visualization of RL-based Search Trend

According to the index which we call reward, the controller is updated by PPO algorithm. In this paper, we choose validation loss as reward. It is obtained by fast training and validating each sampled structure on the proxy dataset. As shown in Fig. 10, the reward curve continues to rise until almost 2,800 iterations, proving that the controller can be trained effectively during the search process.
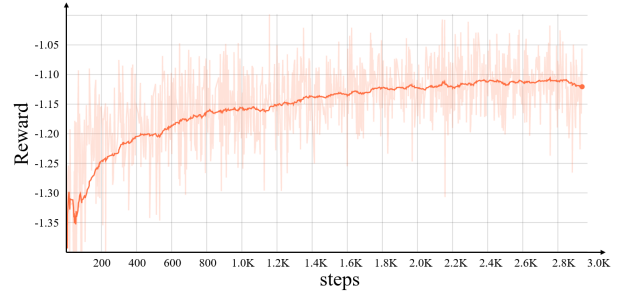
Figure 10 – Performance of reward during the proxy task, which has been growing throughout the process, indicating that the model of reinforcement learning works.

9