ZENDET: REVISITING EFFICIENT OBJECT DETECTION BACKBONES FROM ZERO-SHOT NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

In object detection models, the detection backbone consumes more than half of the overall inference cost. Recent researches attempt to reduce this cost by optimizing the backbone architecture with the help of Neural Architecture Search (NAS). However, existing NAS methods for object detection require hundreds to thousands of GPU hours of searching, making them impractical in fast-paced research and development. In this work, we propose a novel zero-shot NAS method to address this issue. The proposed method, named ZenDet, automatically designs efficient detection backbones without training network parameters, reducing the architecture design cost to nearly zero yet delivering the state-of-the-art (SOTA) performance. Under the hood, ZenDet maximizes the differential entropy of detection backbones, leading to a better feature extractor for object detection under the same computational budgets. After merely one GPU day of fully automatic design, ZenDet innovates SOTA detection backbones on multiple detection benchmark datasets with little human intervention. Comparing to ResNet-50 backbone, ZenDet is $+2.0\%$ better in mAP when using the same amount of FLOPs/parameters and is 1.54 times faster on NVIDIA V100 at the same mAP. Code and pre-trained models will be released after publication.

1 INTRODUCTION

Seeking better and faster deep models for object detection is never an outdated task in computer vision. The performance of a deep object detection network heavily depends on the feature extraction backbone (Li et al., 2018; Chen et al., 2019b). Currently, most state-of-the-art (SOTA) detection backbones (He et al., 2016; Xie et al., 2017; Zhu et al., 2019) are designed by human experts manually which could take years to develop. Since the detection backbone consumes more than half of the total inference cost in many detection frameworks, it is critical to optimize the backbone architecture for better speed-accuracy trade-off on different hardware platforms varying from server-side GPUs to mobile-side chipsets. To reduce the manual design, Neural Architecture Search (NAS) has emerged to facilitate the architecture design. Various NAS methods have demonstrated their efficacy in designing SOTA image classification models (Zoph et al., 2018; Liu et al., 2018; Cai et al., 2019; Tan & Le, 2019). These successful stories inspire recent researchers to use NAS to design detection backbones (Chen et al., 2019b; Du et al., 2020; Jiang et al., 2020) in an end-to-end way.

The existing NAS methods for detection backbone design are all training-based, meaning that they need to train network parameters to evaluate the performance of candidates on the target dataset, taking a long time and consuming huge hardware resources. Hardware consumption and long time searching make these training-based methods inefficient in modern fast-paced research and development. To reduce the training cost, training-free methods are proposed recently, also known as *zero-shot NAS* in previous literatures (Tanaka et al., 2020; Mellor et al., 2021; Chen et al., 2021; Lin et al., 2021). The zero-shot NAS predicts network performance without training network parameters therefore is way faster than training-based NAS. As a relatively new technique, existing zero-shot NAS methods are mostly validated on image classification datasets. Applying zero-shot NAS to object detection backbone design is still an intact challenge.

In this work, we present the first effort of introducing zero-shot NAS technique to design efficient object detection backbones. We show that directly transferring existing zero-shot NAS methods from image classification to detection backbone design will encounter fundamental difficulties. While image classification network only needs to predict the class probability, object detection network needs to additionally predict the bounding boxes of multiple objects, making the direct architecture transfer sub-optimal. To this end, a novel zero-shot NAS method, termed ZenDet, is proposed for searching object detection backbones. The key idea behind ZenDet is inspired by the *Principle of Maximum Entropy* (PME) (Reza, 1994; Kullback, 1997; Brillouin, 2013). Informally speaking, when a network is formulated as an information processing system, its capacity is maximized when its differential entropy (Shannon, 1948) achieves maximum under budget constraints, leading to a better feature extractor for object detection. Based on this observation, ZenDet maximizes the differential entropy of detection backbones by searching for the optimal configuration of network depth and width without training network parameters.

The above strategy raises two technical challenges. **The first challenge is how to estimate the differential entropy of a deep network**. The exact computation of differential entropy requires knowing the precise probability distribution of deep features in high dimensional space which is difficult to estimate in practice. To address this issue, ZenDet estimates the Gaussian upper bound of the differential entropy which only requires computing the variance of the feature maps. **The second challenge is how to efficiently capture objects of different sizes**. In object detection benchmark datasets such as MS COCO Lin et al. (2014), the distribution of object size is data-dependent and non-uniform. To bring in this prior knowledge in backbone design, we introduce the *Multi-Scale Entropy Prior* (MSEP) in backbone entropy estimation to capture different-scale objects. We find that the MSEP improves the detection performance significantly. The overall computation of ZenDet only requires one forward inference of the detection backbone at initialization therefore is nearly zero-cost comparing to previous backbone NAS methods.

The main contributions of this work are summarized as follows:

- Based on the entropy theory, the multi-scale entropy prior is present to rank the expressivity of the backbone instead of training on the target datasets, speeding up searching.
- While using less than one GPU day and 2GB GPU memory, ZenDet achieves competitive performance over other NAS methods on COCO with at least 50x times faster.
- ZenDet is the first zero-shot NAS method designed for object detection with SOTA performance in multiple benchmark datasets under multiple popular detection frameworks.

2 Related work

Backbone for Object Detection Recently, object detectors composed of backbone, neck and head have become increasingly popular due to their effectiveness and high performance (Lin et al., 2017a;b; Tian et al., 2019; Li et al., 2020; 2021). Prevailing detectors directly use the backbone designed for image classification to extract multi-scale features from an image, such as ResNet (He et al., 2016), ResNeXt (Xie et al., 2017) and Deformable Convolutional Network (DCN) (Zhu et al., 2019). Nevertheless, the backbone migrated from image classification may be suboptimal in object detection (Ghiasi et al., 2019). To tackle the gap, many architectures are end-to-end designed for object detection, including Stacked Hourglass Newell et al. (2016), FishNet Sun et al. (2018), DetNet Li et al. (2018), HRNet Wang et al. (2020a) and so on. Albeit with improved performance, these hand-crafted detection architectures heavily rely on human labor and tedious trial-and-error processes.

Neural Architecture Search Neural Architecture Search (NAS) is initially developed to automatically design network architectures for image classification models (Zoph et al., 2018; Liu et al., 2018; Real et al., 2019; Cai et al., 2019; Tan & Le, 2019; Lin et al., 2021). Using NAS to design object detection models has not been well studied. Currently, existing detection NAS methods are all training-based methods. Some methods focus on searching detection backbones, such as DetNAS (Chen et al., 2019b), SpineNet (Du et al., 2020) and SP-NAS (Jiang et al., 2020), while the others focus on searching FPN neck, such as NAS-FPN (Ghiasi et al., 2019), NAS-FCOS (Wang et al., 2020b) and OPANet (Liang et al., 2021). These methods require training and evaluation on

the target datasets which is intensive in computation. ZenDet distinguishes itself as being the first zero-shot NAS method for the backbone design of object detection.

3 PRELIMINARY

In this section, we first formulate a deep network as a system with continuous state space. Then we define the differential entropy of this system and show how to estimate this entropy via its Gaussian upper bound. Finally we introduce the basic concept of vanilla network search space for designing our detection backbones.

Continuous State Space of Deep Networks A deep network $F(\cdot) : \mathbb{R}^d \to \mathbb{R}$ maps an input image $\boldsymbol{x} \in \mathbb{R}^d$ to its label $y \in \mathbb{R}$. The topology of a network can be abstracted as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V}$ consists of neurons and the edge set $\mathcal{E}$ consists of spikes between neurons. For any $v \in \mathcal{V}$ and $e \in \mathcal{E}$, $h(v) \in \mathbb{R}$ and $h(e) \in \mathbb{R}$ present the values endowed with each vertex v and each edge e respectively. The set $\mathcal{S} = \{h(v), h(e) : \forall v \in \mathcal{V}, e \in \mathcal{E}\}$ defines the continuous state space of the network F.

According to the Principle of Maximum Entropy, we want to maximize the differential entropy of network F, under some given computational budgets. The entropy $H(\mathcal{S})$ of set $\mathcal{S}$ measures the total information contained in the system (network) F, including the information contained in the latent features $H(\mathcal{S}_v) = \{h(v) : v \in \mathcal{V}\}$ and in the network parameters $H(\mathcal{S}_e) = \{h(e) : e \in \mathcal{E}\}$. As for object detection backbone design, we only care about the entropy of latent features $H(\mathcal{S}_v)$ rather than the entropy of network parameters $H(\mathcal{S}_e)$. Informally speaking, $H(\mathcal{S}_v)$ measures the feature representation power of F while $H(\mathcal{S}_e)$ measures the model complexity of F. Therefore, in the remainder of this work, the differential entropy of F refers to the entropy $H(\mathcal{S}_v)$ by default.

Entropy of Gaussian Distribution The differential entropy of Gaussian distribution can be found in many textbooks such as (Norwich, 1993). Suppose x is sampled from Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. Then the differential entropy of x is given by

$$H^*(x) = \frac{1}{2}\log(2\pi) + \frac{1}{2} + H(x) \qquad H(x) := \log(\sigma)\,. \tag{1}$$

From Eq. 1, the entropy of Gaussian distribution only depends on the variance. In the following, we will use $H(x)$ instead of $H^*(x)$ as constants do not matter in our discussion.

Gaussian Entropy Upper Bound Since the probability distribution $\mathbb{P}(\mathcal{S}_v)$ is a high dimensional function, it is difficult to compute the precise value of its entropy directly. Instead, we propose to estimate the upper bound of the entropy, given by the following well-known theorem (Cover & Thomas, 2012):

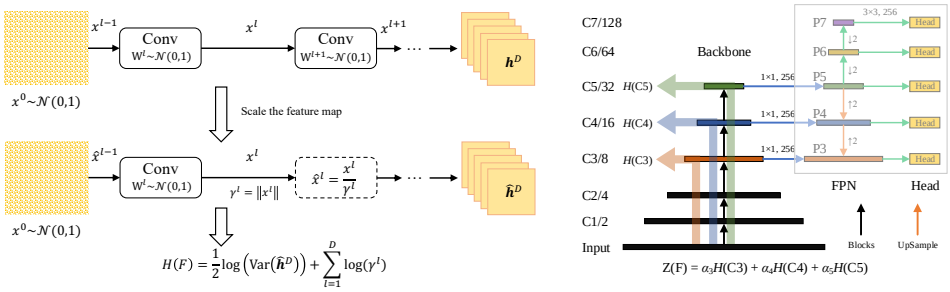
Theorem 1. *For any continuous distribution $\mathbb{P}(x)$ of mean μ and variance σ^2, its differential entropy is maximized when $\mathbb{P}(x)$ is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$.*

Theorem 1 says that the differential entropy of a distribution is upper bounded by a Gaussian distribution with the same mean and variance. Combining this with Eq. (1), we can easily estimate the network entropy $H(\mathcal{S}_v)$ by simply computing the feature map variance and then use Eq. (1) to get the Gaussian entropy upper bound for the network.

Vanilla Network Search Space Following previous works, we design our backbones in the vanilla convolutional network space (Li et al., 2018; Chen et al., 2019b; Du et al., 2020; Lin et al., 2021) because this space is widely adopted in detection backbones and is used as a prototype in theoretical literature (Poole et al., 2016; Serra et al., 2018; Hanin & Rolnick, 2019). A vanilla network is stacked by multiple convolutional layers followed by RELU activations. Auxiliary components such as residual link (He et al., 2016), Batch Normalization (BN) (Ioffe & Szegedy, 2015) and Squeeze-and-Excitation (SE) (Hu et al., 2018) are all removed **during the search and only during the search**. These removed auxiliary components are plugged in back after the search. Therefore, the final architecture for training still has these components.

Consider a vanilla convolutional network with D layers of weights $\mathbf{W}^1$, ..., $\mathbf{W}^D$ whose output feature maps are $\boldsymbol{x}^1$, ..., $\boldsymbol{x}^D$. The input image is $\boldsymbol{x}^0$. Let $\phi(\cdot)$ denote the RELU activation function. Then the forward inference is given by

$$\boldsymbol{x}^l = \phi(\boldsymbol{h}^l) \quad \boldsymbol{h}^l = \mathbf{W}^l * \boldsymbol{x}^{l-1} \quad for\ l = 1, ..., D\,. \tag{2}$$



(a) Single-scale entropy score with rescaling (b) Multi-scale entropy score for detection

Figure 1: Computational graph of entropy score for detection backbone with multi-scale features.

For sake of simplicity, we set the bias of the convolutional layer to zero.

4 ZenDet-NAS for Object Detection

In this section, we first describe how to compute the differential entropy of the single-scale feature for very deep vanilla networks with the re-scaling trick. Then we introduce the Multi-Scale Entropy Prior (MSEP) to better capture the prior distribution of object size in ZenDet. Finally, we present the complete ZenDet-NAS under the Evolutionary Algorithm (EA) framework.

4.1 Single-Scale Entropy for Very Deep Backbones

To compute the entropy of the detection backbone, we initialize all backbone parameters by the standard Gaussian distribution $\mathcal{N}(0,1)$. Then we randomly generate an image filled with the standard Gaussian noise and perform forward inference. Based on the discussion in Section 3, the (Gaussian upper bound) entropy $H(F)$ of the network F is then given by

$$H(F) = \frac{1}{2}\log(\mathrm{Var}(\boldsymbol{h}^D)) \ . \tag{3}$$

Please note that the variance is computed on the last pre-activation feature map $\boldsymbol{h}^D$.

For very deep networks, directly using Eq. (3) might cause a numerical overflow. The same problem is also observed in Zen-NAS (Lin et al., 2021). Inspired by the analysis in Zen-NAS where the authors show that the gradient norm is scale-invariant to BN layers after variance compensation, we propose a more straightforward solution. We directly re-scale each feature map $\boldsymbol{x}^l$ by some constants γ^l during inference, that is $\boldsymbol{x}^l = \phi(\boldsymbol{h}^l)/\gamma^l$, and then compensate the entropy of the network by

$$H(F) = \frac{1}{2}\log(\mathrm{Var}(\hat{\boldsymbol{h}}^D)) + \sum_{l=1}^{D}\log(\gamma^l) \ . \tag{4}$$

The values of γ^l can be arbitrarily given as long as the forward inference does not overflow or underflow. In practice, we find that simply setting γ^l to the Euclidean norm of the feature map works well. The overall process is illustrated in Figure 1 (a).

4.2 Multi-Scale Entropy Prior for object detection

In real-world images, the distribution of object size is not uniform. To bring in this prior knowledge, the detection backbone has 5 stages where each stage downsamples the feature resolution to half. The MSEP collects feature maps from the final output of each stage and weighted-sum the corresponding feature map entropies as a new measurement. We name this new measurement as multi-scale entropy. The overall process is illustrated in Figure 1 (b). In this figure, the backbone extracts multi-scale features $\boldsymbol{C} = (\mathrm{C1}, \mathrm{C2}, ..., \mathrm{C5})$ at different resolutions. Then the FPN neck fuses $\boldsymbol{C}$ as input features $\boldsymbol{P} = (\mathrm{P1}, \mathrm{P2}, ..., \mathrm{P7})$ for the detection head. The multi-scale entropy $Z(F)$ of

backbone F is then defined by

$$Z(F) := \alpha_1 H(\mathrm{C1}) + \alpha_2 H(\mathrm{C2}) + \cdots + \alpha_5 H(\mathrm{C5}) \tag{5}$$

where $H(Ci)$ is the entropy of Ci for $i = 1, 2, \cdots, 5$. The weights $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_5)$ store the multi-scale entropy prior. For most detection tasks, we find that $\boldsymbol{\alpha} = (0, 0, 1, 1, 6)$ works well in practice.

4.3 ZenDet-NAS in Evolutionary Algorithm Framework

Algorithm 1 ZenDet-NAS with Coarse-to-Fine Evolution

Require: Search space $\mathcal{S}$, inference budget B, maximal depth L, total number of iterations T, evolutionary population size N, initial structure F_0, fine-search flag *Flag*.
Ensure: NAS-designed ZenDet backbone F^*.

```
Initialize population P = {F_0}, Flag=False.
for t = 1, 2, ··· , T do
    if t equals to T/2 then
        Keep top 10 networks of highest multi-scale entropy in P and remove the others.
        Set Flag = True.
    end if
    Randomly select F_t ∈ P.
    Mutate F̂_t = MUTATE(F_t, S, Flag)
    if F̂_t exceeds inference budget or has more than L layers then
        Do nothing.
    else
        Get multi-scale entropy Z(F̂_t).
        Append F̂_t to P.
    end if
    Remove networks of the smallest multi-scale entropy if the size of P exceeds B.
end for
Return F*, the network of the highest multi-scale entropy in P.
```

Algorithm 2 MUTATE

Require: Structure F_t, search space $\mathcal{S}$, fine-search flag *Flag*.
Ensure: Randomly mutated structure $\hat{F}_t$.

```
Uniformly select a block h in F_t.
if Flag equals to True then
    Uniformly alternate the kernel size, width within some range.
else
    Uniformly alternate the block type, kernel size, width and depth within some range.
end if
Return the mutated structure F̂_t.
```

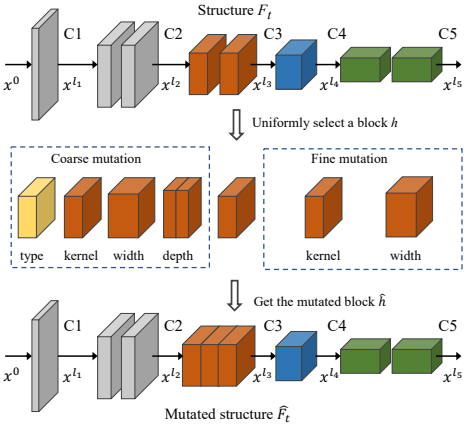
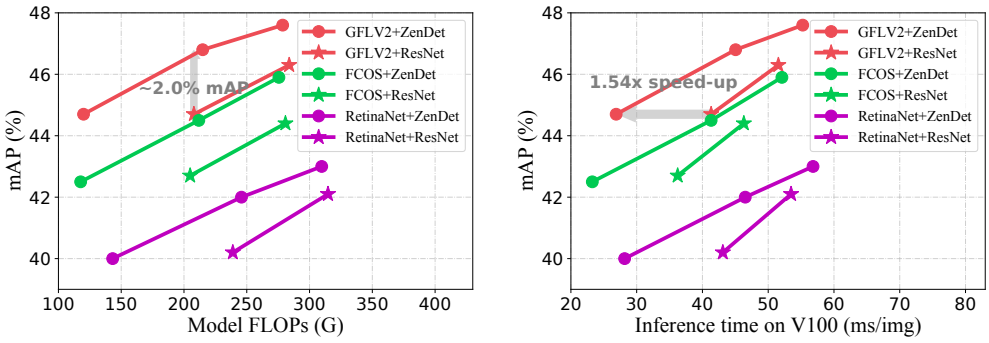


Figure 2: Visualization of Algorithm 2.

Combining all above, we present our ZenDet-NAS in Algorithm 1. The ZenDet-NAS maximizes the multi-scale differential entropy of detection backbones using Evolutionary Algorithm (EA). To improve the evolution efficiency, a coarse-to-fine strategy is proposed to reduce the search space gradually. First, we randomly generate N seed architectures to fill the population $\mathcal{P}$. As shown in Figure 2, a seed architecture F_t consists of a sequence of building blocks such as ResNet block (He et al., 2016) or MobileNet block (Sandler et al., 2018). Then we randomly select one block and replace it with its mutated version. We use coarse-mutation in the early stage of EA and switch to fine-mutation after $T/2$ EA iterations. In the coarse-mutation, the block type, kernel size, depth and width are mutated randomly; in the fine-mutation, only kernel size and width are mutated.



(a) mAP V.S. FLOPs (b) mAP V.S. Speed

Figure 3: mAP v.s. FLOPs and inference speed on COCO val 2017 in Table 1. Note that FLOPs in (a) is the value of the full detector, cotaining backbone, FPN and head.

After the mutation, if the inference cost of the new structure $\hat{F}_t$ does not exceed the budget (e.g., FLOPs, parameters and latency) and its depth is smaller than budget L, $\hat{F}_t$ is appended into the population $\mathcal{P}$. The maximal depth L prevents the algorithm from generating over-deep structures. During EA iterations, the population is maintained to a certain size by discarding the worst candidate of the smallest multi-scale entropy. At the end of ZenDet-NAS, the backbone with the highest multi-scale entropy is returned.

5 Experiments

In this section, we first describe detail settings for searching and training with ZenDet-NAS. Then in subsection 5.2, we apply ZenDet-NAS to design better ResNet-like backbones on COCO dataset (Lin et al., 2014). We align the inference budget with ResNet-50/101. The performance of ZenDet and ResNet are compared under multiple detection frameworks including RetinaNet (Lin et al., 2017b), FCOS (Tian et al., 2019), and GFLV2 (Li et al., 2021). For fairness, we use the same training setting in all experiments for all backbones. In subsection 5.3, we compare the search cost of ZenDet to SOTA NAS methods for object detection. Subsection 5.4 shows the comparison with other zero-shot NAS proxies designed for image classification but adapted to object detection. Subsection 5.5 reports the ablation studies of different components in ZenDet-NAS. Finally, subsection 5.6 verifies the transferability of ZenDets on several detection and segmentation datasets with SOTA performance.

5.1 Experiment Settings

Searching Details In ZenDet-NAS, the evolutionary population N is set to 256. The total EA iterations $T = 96000$. Following the previous designs (Chen et al., 2019b; Jiang et al., 2020; Du et al., 2020), ZenDet is optimized for FLOPs. The resolution for computing multi-scale entropy is 384×384.

Dataset and Training Details We evaluate detection performance on COCO (Lin et al., 2014) using the official training/testing splits. The mAP is evaluated on val 2017 by default and GFLV2 is additionally evaluated on test-dev 2007 following common practice. All models are trained from scratch (He et al., 2019) for 6X (73 epochs) on COCO. Following the Spinenet (Du et al., 2020), we use multi-scale training and Synchronized Batch Normalization (SyncBN). For VOC dataset, train-val 2007 and train-val 2012 are used for training, and test 2007 for evaluation. For image classification, all models are trained on ImageNet-1k (Deng et al., 2009) with a batch size of 256 for 120 epochs. Other setting details can be found in Appendix A.

5.2 Design Better ResNet-like Backbones

We search efficient ZenDet backbones for object detection and align with ResNet-50/101 in Table 1. ZenDet-S uses 60% less FLOPs than ResNet-50; ZenDet-M is aligned with ResNet-50 with

Table 1: ZenDet and ResNet on the COCO. All results using the same training setting. FPS on V100 is benchmarked on the full model with NVIDIA V100 GPU, pytorch, FP32, batch size 32.

Backbone	FLOPs Backbone	Params Backbone	Head	val2017 $\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$	test-dev $\mathbf{AP_{test}}$	FPS on V100
R50	83.6G	23.5M	RetianNet	40.2	24.3	43.3	52.2	-	23.2
			FCOS	42.7	28.8	46.2	53.8	-	27.6
			GFLV2	44.7	29.1	48.1	56.6	45.1	24.2
R101	159.5G	42.4M	RetianNet	42.1	25.8	45.7	54.1	-	18.7
			FCOS	44.4	28.3	47.9	56.9	-	21.6
			GFLV2	46.3	29.9	50.1	58.7	46.5	19.4
ZenDet-S	48.7G	21.2M	RetianNet	40.0	23.9	43.3	52.7	-	35.5
			FCOS	42.5	26.8	46.0	54.6	-	43.0
			GFLV2	44.7	27.6	48.4	58.2	44.8	37.2
ZenDet-M	89.9G	25.8M	RetianNet	42.0	26.7	45.2	55.1	-	21.5
			FCOS	44.5	28.6	48.1	56.1	-	24.2
			GFLV2	46.8	29.9	50.4	60.0	46.7	22.2
ZenDet-L	152.9G	43.9M	RetianNet	43.0	27.3	46.5	56.0	-	17.6
			FCOS	45.9	30.2	49.4	58.4	-	19.2
			GFLV2	47.6	30.2	51.8	60.8	48.0	18.1

similar FLOPs and number of parameters as ResNet-50; ZenDet-L is aligned with ResNet-101. The feature dimension in the FPN and heads is set to 256 for ZenDet-M and ZenDet-L but is set to 192 for ZenDet-S. The fine-tuned results of models pre-trained on ImageNet-1k are reported in Appendix B.

In Table 1, ZenDet outperforms ResNet by a large margin. The improvements are consistent across three detection frameworks. Particularly, when using the newest framework GFLV2, ZenDet improves COCO mAP by $+2\%$ at the similar FLOPs of ResNet-50, and speeds up the inference by 1.54x times faster at the same accuracy as ResNet-50. Figure 3 visualizes the comparison in Table 1.

Remark Please note that we did not copy the numbers of baseline methods reported in previous works in Table 1 because the mAP not only depends on the architecture but also depends on the training schedule, such as training epochs, learning rate, pre-training and so on. Therefore, for a fair comparison, all models in Table 1 are trained by the same training schedule. For comparison with numbers reported in previous works, see subsection 5.3.

5.3 Comparison with SOTA NAS Methods

Table 2: Comparisons with SOTA NAS methods for object detection. FLOPs are counted for full detector.

Method	Training-free	Search Cost GPU Days	Search Part	FLOPs All	Pretrain/ Scratch	Epochs	COCO ($\mathbf{AP_{test}}$)
DetNAS	×	68	backbone	289G	Pretrain	24	43.4
SP-NAS	×	26	backbone	655G	Pretrain	24	47.4
SpineNet	×	100x TPUv3†	backbone+FPN	524G	Scratch	350	48.1
ZenDet	✓	**0.6**	backbone	279G	Scratch	73	48.0

†: SpineNet paper did not report the total search cost, only mentioned that 100 TPUv3 was used for searching.

In Table 2, we compare ZenDet with SOTA NAS methods for the backbone design in object detection. We directly use the numbers reported in the original papers. Since each NAS method uses different design spaces and training settings, it is impossible to make an absolutely fair comparison for all methods that everyone agrees with. Nevertheless, we list the total search cost, mAP and FLOPs of the best models reported in each work. This gives us an overall impression of how each NAS method works in real-world practice. From Table 2, ZenDet is the only zero-shot (training-free) method with 48.0% mAP on COCO, using 0.6 GPU days of search. SpineNet (Du et al., 2020) achieves a slightly better mAP with 2x more FLOPs. It uses 100 TPUv3 for searching but the total

Table 3: Fair comparsion between ZenDet, DetNAS (Chen et al., 2019b) and SpineNet (Du et al., 2020). All backbones are trained under GFLV2 head with 6X training epochs. FLOPs and parameters are counted for full detector.

Backbone	Search Part	Search Space	FLOPs	Params	$\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$	FPS on V100
DetNAS-3.8G	backbone	ShuffleNetV2 +Xception	205G	35.5M	46.4	29.3	50.0	59.0	17.6
SpineNet-96	backbone+FPN	ResNet Block	216G	41.3M	46.6	29.8	50.2	58.9	19.9
ZenDet-M	backbone	ResNet Block	215G	34.9M	46.8	29.9	50.4	60.0	22.2

search cost is not reported in the original paper. ZenDet achieves better mAP than DetNAS (Chen et al., 2019b) and SP-NAS(Jiang et al., 2020) while being $50 \sim 100$ times faster in search.

To further fairly compare different backbones under the same training settings, we train backbones designed by ZenDet-NAS and previous backbone NAS methods in Table 3. Because the implementation of SP-NAS is not open-sourced, we re-train ZenDet, DetNAS and SpineNet on COCO from scratch. Table 3 shows that ZenDet requires fewer parameters and has a faster inference speed on V100 when achieving the competitive performance over DetNAS and SpineNet on COCO.

5.4 Comparison with Zero-Shot Proxies for Image Classification

Table 4: Different zero-shot proxies on COCO with FCOS. All methods use the same search space, FLOPs budget, searching strategy and training schedule.

Proxy	FLOPs Backbone	Params Backbone	$\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$
R50	84G	23.5M	38.0	23.2	40.8	47.6
SyncFlow	90G	67.4M	35.6	21.8	38.1	44.8
NASWOT	88G	28.1M	36.7	23.1	38.8	45.9
Zen-NAS	91G	67.9M	38.1	23.2	40.5	48.1
ZenDet	89G	25.8M	**39.4**	23.7	42.3	50.0

We compare ZenDet with architectures designed by zero-shot proxies for image classification in previous works, including SyncFlow (Tanaka et al., 2020), NASWOT (Mellor et al., 2021), Zen-NAS (Lin et al., 2021). For a fair comparison, All methods use the same search space, FLOPs budget $91\,G$, searching strategy and training schedule. All searched backbones are trained on COCO with the FCOS head and 3X training from scratch. The results are reported in Table. 4.

Among these methods, SyncFlow and NASWOT perform worse than ResNet-50 on COCO albeit they show competitive performance in image classification tasks. Zen-NAS achieves competitive performance over ResNet-50. The ZenDet outperforms Zen-NAS by $+1.3\%$ mAP with slightly fewer FLOPs and nearly one third of parameters.

5.5 Ablation Study and Analysis

Table 5: Comparison of different evolutionary searching strategies in ZenDet-NAS. C-to-F: Coarse-to-Fine.

		ImageNet-1K			COCO with YOLOF				COCO with FCOS			
Score	Mutation	FLOPs	Params	TOP-1 %	$\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$	$\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$
R50	None	4.1G	23.5M	78.0	37.8	19.1	42.1	53.3	38.0	23.2	40.8	47.6
Single-scale	Coarse	4.4G	**60.1M**	78.7	39.8	19.9	44.4	56.5	38.8	23.1	41.4	50.1
Multi-scale	Coarse	4.3G	29.4M	78.9	40.1	21.1	44.5	55.9	39.4	23.7	42.3	50.0
Multi-scale	C-to-F	4.4G	25.8M	**79.1**	**40.3**	**20.8**	**44.7**	**56.4**	**40.0**	**24.5**	**42.6**	**50.6**

Table 5 reports the ZenDet backbones searched by different evolutionary strategies and whether using multi-scale entropy prior. The COCO mAPs of models trained in two detection frameworks (YOLOF and FCOS) are reported in the right big two columns. YOLOF models are trained by 12 epochs with ImageNet pre-trained initialization, while FCOS models are trained with the 3X training epochs. We also compare their image classification ability on ImageNet-1k. All models are constrained by the FLOPs less than 4.4 G while the number of parameters is not constrained.

Single-scale entropy v.s. Multi-scale entropy In this comparison, we only use coarse-mutation in the EA process to exclude the impact from coarse-to-fine refinement. from Table 5, comparing with ResNet-50, the model searched by single-scale entropy score obtain $+0.7\%$ accuracy gain on ImageNet, $+2\%$ mAP gain on COCO with FPN-free YOLOF. When using multi-scale entropy, both single-scale model and multi-scale model get similar accuracy on ImageNet. The single-scale model uses 2X more parameters than the multi-scale model under the same FLOPs constraint. In terms of mAP, multi-scale model outperforms single-scale model by $+0.3\%$ on COCO with YOLOF and 0.6% on COCO with FCOS.

Coarse-to-Fine Improvement From the last row of Table 5, the coarse-to-fine mutation further enhances the performance of multi-scale entropy prior by $+0.2\%$ on ImageNet, $+0.2\%$ mAP on YOLOF and $+0.6\%$ mAP on FCOS. The overall improvement over ResNet-50 is $+1.1\%$ on ImageNet-1k, $+2.5\%$ on COCO with YOLOF and $+2.0\%$ on COCO with FCOS. More details about the searching process and architectures could be found in Appendix D, E.

5.6 Transfer Learning

Table 6: Transferability of ZenDet in multiple object detection and instance segmentation tasks. FLOPs and parameters reported are counted for full detector.

Task	Head	Dataset	Backbone	Resolution	Epochs	FLOPs	$\mathbf{AP_{val}}$	$\mathbf{AP^{mask}_{val}}$
Object Detection	FCOS	VOC	R50	1000×600	12	120G	76.8	-
			ZenDet-M	1000×600	12	123G	**80.9**	-
		Citescapes	R50	2048×1024	64	411G	37.0	-
			ZenDet-M	2048×1024	64	426G	**38.1**	-
Instance Segmentation	MASK R-CNN	COCO	R50	1333×800	73	375G	43.2	39.2
			ZenDet-M	1333×800	73	379G	**44.5**	**40.3**
			R50†	640×640	350	228G	42.7	37.8
			SpineNet-49†	640×640	350	216G	42.9	38.1

†: Numbers are cited from SpineNet paper (Du et al., 2020).

VOC and Cityscapes To evaluate the transferability of ZenDet in different datasets, we transfer the FCOS-based ZenDet-M to VOC and Cityscapes dataset, as shown in the upper half of Table 6. The models are fine-tuned after pre-trained on ImageNet. Comparing to ResNet-50, ZenDet-M achieves $+4.1\%$ better mAP in VOC, $+1.1\%$ better mAP in Cityscape.

Instance Segmentation The lower half of Table 6 reports results of Mask R-CNN models for the COCO instance segmentation task. Comparing to ResNet-50, ZenDet-M achieves $+1.3\%$ better AP and $+1.1\%$ better mask AP with similar model size and FLOPs. ZenDet-M also outperforms training-based SpineNet by $+1.6\%$ in AP and $+2.2\%$ in mask AP.

6 Conclusion

In this paper, we propose a zero-shot NAS method termed ZenDet-NAS for designing high performance backbones for object detection. While achieving better or competitive detection accuracy, the search speed of ZenDet-NAS is several orders of magnitude faster than previous training-based NAS methods. Within one GPU day, the ZenDet automatically designed by ZenDet-NAS is significantly more efficient than popular SOTA backbones for object detection in terms of FLOPs and inference latency. Extensive experiments and analyses on various datasets validate its excellent transferability. In the future, we will generalize ZenDet-NAS to search for detection backbone and FPN neck in a unified framework.

REFERENCES

Leon Brillouin. *Science and information theory*. Courier Corporation, 2013.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

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

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *Proceedings of the International Conference on Learning Representations*, 2021.

Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. *Advances in Neural Information Processing Systems*, 32: 6642–6652, 2019b.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11592–11601, 2020.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7036–7045, 2019.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pp. 2596–2604. PMLR, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4918–4927, 2019.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Chenhan Jiang, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sp-nas: Serial-to-parallel backbone search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11863–11872, 2020.

Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. In *NeurIPS*, 2020.

Xiang Li, Wenhai Wang, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss v2: Learning reliable localization quality estimation for dense object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11632–11641, 2021.

Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: Design backbone for object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 334–350, 2018.

Tingting Liang, Yongtao Wang, Zhi Tang, Guosheng Hu, and Haibin Ling. Opanas: One-shot path aggregation network architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10195–10203, 2021.

Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance deep image recognition. In *2021 IEEE/CVF International Conference on Computer Vision*, 2021.

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 *European conference on computer vision*, pp. 740–755. Springer, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017a.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017b.

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

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations*, 2017.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pp. 7588–7598. PMLR, 2021.

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pp. 483–499. Springer, 2016.

Kenneth H Norwich. *Information, sensation, and perception*. Academic Press San Diego, pp. 81-82, 1993.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Advances in neural information processing systems*, 29:3360–3368, 2016.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Fazlollah M Reza. *An introduction to information theory*. Courier Corporation, 1994.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pp. 4558–4566. PMLR, 2018.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. In *Advances in Neural Information Processing Systems*, pp. 760–770, 2018.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020.

Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9627–9636, 2019.

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2020a.

Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11943–11951, 2020b.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9308–9316, 2019.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A TRAINING DETAILS

Searching Details In ZenDet-NAS, the evolutionary population N is set as 256 while total iterations $T = 96000$. Residual blocks and bottleneck blocks are utilized as searching space when comparing with ResNet series backbone (He et al., 2016). Following the previous designs (Chen et al., 2019b; Jiang et al., 2020; Du et al., 2020), ZenDet is optimized for the budget of FLOPs according to the target networks, i.e., ResNet-50 and ResNet-101. To balance the computational complexity and large resolution demand, the resolution in search is set as 384×384 for ZenDet. When starting the search, the initial structure is composed of 5 downsampling stages with small and narrow blocks to meet the reasoning budget. In the mutation, whether the coarse-mutation or fine-mutation, the width of the selected block is mutated in a given scale $\{1/1.5, 1/1.25, 1, 1.25, 1.5, 2\}$, while the depth increases or decreases 1 or 2. The kernel size is searched in set $\{3, 5\}$. Note that blocks deeper than 10 will be divided into 2 blocks equally to enhance the diversity.

Dataset and Training Details For object detection, trainval35k with 115K images in the COCO dataset is mainly used for training. With the single-scale testing of resolution 1333×800, COCO mAP results are reported on the val 2017 for most experiments and the test-dev 2007 for GFLV2 results in Table 1. When training on the COCO dataset, the initial learning rate is set to 0.02, and decays two times with the ratio of 0.1 during training. SGD is adopted as optimizer with momentum 0.9; weight decay of 10^{-4}; batch size of 16 (on 8 Nvidia V100 GPUs); patch size of 1333×800.

Additionally, multi-scale training and Synchronized Batch Normalization (SyncBN) are adopted to enhance the stability of the scratch training without increasing the complexity of inference. Training from scratch is used to avoid the gap between ImageNet pre-trained model, to ensure a fair comparison with baselines. 3X learning schedule is applied for the ablation study with a multi-scale range between $[0.8, 1.0]$ (36 epochs, decays at 28 and 33 epochs), and 6X learning schedule for the SOTA comparisons with the range between $[0.6, 1.2]$ (73 epochs, decays at 65 and 71 epochs). All object detection training is produced under mmdetection (Chen et al., 2019a) for fair comparisons, and hyperparameters not mentioned in the paper are always set to default values in mmdection.

For image classification, all models are trained on ImageNet-1K with a batch size of 256 for 120 epochs. When training on ImageNet-1K, We use SGD optimizer with momentum 0.9; cosine learning rate decay (Loshchilov & Hutter, 2017); initial learning rate 0.1; weight decay 4×10^{-5}.

B OBJECT DETECTION WITH IMAGENET PRE-TRAIN MODELS

Table 7: Results between Scratch and Pretrain strategy on the COCO with single-scale testing. Training strategy on ImageNet is same as Table.5.

Backbone	FLOPs Backbone	Params Backbone	Head	Strategy	Epochs	$\mathbf{AP_{val}}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$
R50	83.6G	23.5M	GFLV2	Scratch	73	44.7	29.1	48.1	56.6
			GFLV2	Pretrain	24	44.0	27.1	47.8	56.1
			GFLV2	Pretrain	24	43.9†	-	-	-
ZenDet-M	89.9G	25.8M	RetianNet	Scratch	73	42.0	26.7	45.2	55.1
			RetianNet	Pretrain	24	42.3	25.3	46.5	56.0
			FCOS	Scratch	73	44.5	28.6	48.1	56.1
			FCOS	Pretrain	24	44.5	28.8	48.5	56.9
			GFLV2	Scratch	73	46.8	29.9	50.4	60.0
			GFLV2	Pretrain	24	46.0	29.0	50.0	59.9

†: results in this line are reported in the official github (Li et al., 2021).

In the main body of the paper, training from scratch is used to avoid the gap between ImageNet pre-trained model, to ensure a fair comparison with baselines He et al. (2019). Since6X training from scratch inevitably consumes 3 times more time than 2X pre-trained training, we use the ImageNet pre-trained model to initialize the ZenDet-M in various heads, including RetianNet, FCOS and GFLV2. As present in Table. 1, 7, whether using training from scratch or ImageNet pre-training, ZenDet could outperform ResNet-50 in the three popular FPN-based frameworks by large margins.

C WEIGHTS ARRANGEMENT

Table 8: Results between different arrangements of weights in MSEP on COCO. R50 represents ResNet-50.

Backbone	α_3:α_4:α_5	FLOPs	Params	$\mathbf{AP_{val}}$	$\mathbf{AP}_{50}$	$\mathbf{AP}_{75}$	$\mathbf{AP}_S$	$\mathbf{AP}_M$	$\mathbf{AP}_L$
R50	None	83.6G	23.5M	38.0	55.2	41.0	23.2	40.8	47.6
ZenDet	1:1:1	84.4G	11.5M	37.4	54.6	40.0	23.6	39.8	46.6
ZenDet	1:1:2	84.8G	13.4M	37.8	54.9	40.5	23.2	40.0	47.8
ZenDet	1:1:4	85.9G	17.2M	38.6	56.0	41.4	23.4	41.3	48.6
ZenDet	1:1:6	88.7G	29.4M	**39.4**	**57.3**	**42.1**	**23.7**	42.3	**50.0**
ZenDet	1:1:8	89.9G	31.7M	**39.4**	57.2	42.0	**23.7**	**42.5**	49.5
ZenDet	1:4:1	86.3G	10.9M	35.7	52.6	38.3	22.2	38.1	44.9
ZenDet	4:1:1	86.1G	11.1M	33.9	50.2	36.7	20.4	36.1	43.4

In Table 8, we tune the different arrangements in a large range and all searched models are trained on the COCO dataset with FCOS head and 3X learning schedule. Table 8 shows that if the same weights are arranged to C3-C5, the performance of ZenDet on COCO is worse than that of the baseline ResNet-50. We take the performance and complexity into account and confirm the ratio of 1:1:6 may be good enough for the current FPN structure.

D DETAIL STRUCTURE OF ZENDETS

We list detail structure in Table 1, 5.

The 'block' column is the block type. 'Conv' is the standard convolution layer followed by BN and RELU. 'ResBlock' is the residual bottleneck block used in ResNet-50 and is stacked by two Blocks in our design. 'kernel' is the kernel size of kxk convolution layer in each block. 'in', 'out' and 'bottleneck' are numbers of input channels, output channels and bottleneck channels respectively. 'stride' is the stride of current block. '# layers' is the number of duplication of current block type.

Table 9: Architecture of single-scale entropy score with coarse mutation in Table. 5

block	kernel	in	out	stride	bottleneck	# layers	level
Conv	3	3	96	2	-	1	C1
ResBlock	5	96	208	2	32	2	C2
ResBlock	5	208	560	2	56	1	C3
ResBlock	5	560	1264	2	112	2	C4
ResBlock	5	1264	1712	2	224	3	C5
ResBlock	5	1712	2048	1	224	3	C5
ResBlock	5	2048	2048	1	256	4	C5

Table 10: Architecture of multi-scale entropy score with coarse mutation in Table. 5

block	kernel	in	out	stride	bottleneck	# layers	level
Conv	3	3	32	2	-	1	C1
ResBlock	5	32	128	2	24	1	C2
ResBlock	5	128	512	2	72	5	C3
ResBlock	5	512	1632	2	112	5	C4
ResBlock	5	1632	2048	2	216	4	C5

E VISUALIZATION OF SEARCHING PROCESS

The visualizations of searching process on different entropy scores are shown in Figure 4, 5, 6. We compare the change of layers, FLOPs and parameters during different iterations. Visualizations prove our assumptions in the main body of the paper.

Table 11: Architecture of multi-scale entropy score with coarse-to-fine mutation in Table. 5 / ZenDet-M architecture in Table. 1

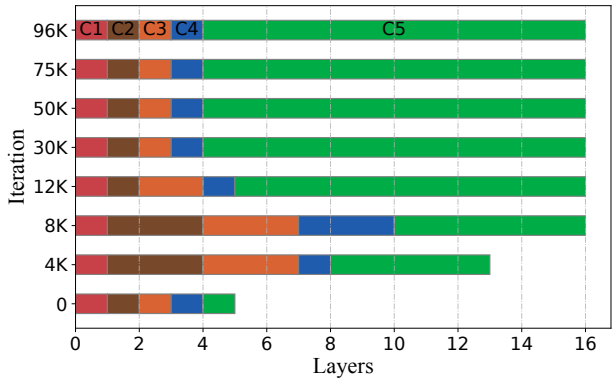
block	kernel	in	out	stride	bottleneck	# layers	level
Conv	3	3	64	2	-	1	C1
ResBlock	3	64	120	2	64	1	C2
ResBlock	5	120	512	2	72	5	C3
ResBlock	5	512	1632	2	112	5	C4
ResBlock	5	1632	2048	2	184	4	C5

Table 12: ZenDet-S in Table. 1

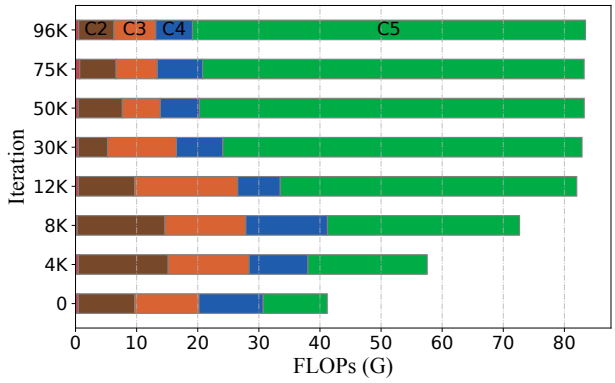
block	kernel	in	out	stride	bottleneck	# layers	level
Conv	3	3	32	2	-	1	C1
ResBlock	5	32	48	2	32	1	C2
ResBlock	3	48	272	2	120	2	C3
ResBlock	5	272	1024	2	80	5	C4
ResBlock	3	1024	2048	2	240	5	C5

Table 13: ZenDet-L in Table. 1

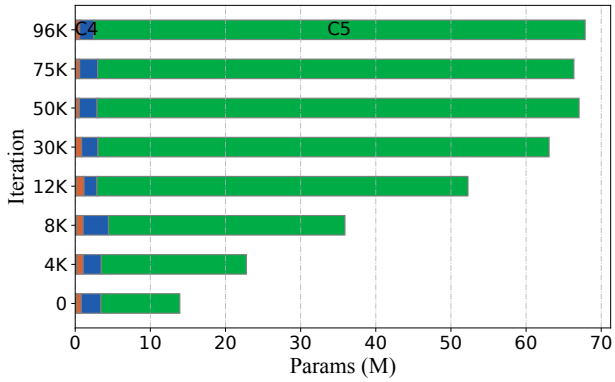
block	kernel	in	out	stride	bottleneck	# layers	level
Conv	3	3	80	2	-	1	C1
ResBlock	3	80	144	2	80	1	C2
ResBlock	5	144	608	2	88	6	C3
ResBlock	5	608	1912	2	136	6	C4
ResBlock	5	1912	2400	2	220	5	C5



(a) Iterations V.S. #Layers

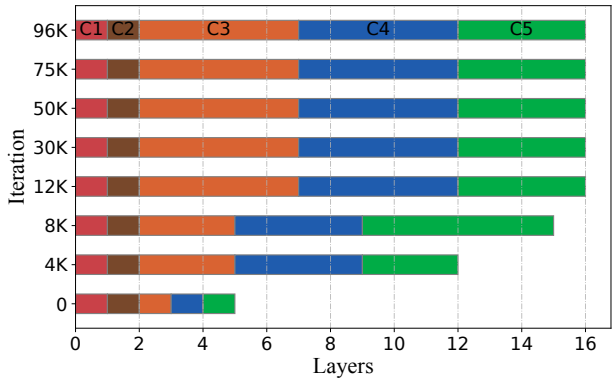


(b) Iterations V.S. FLOPs

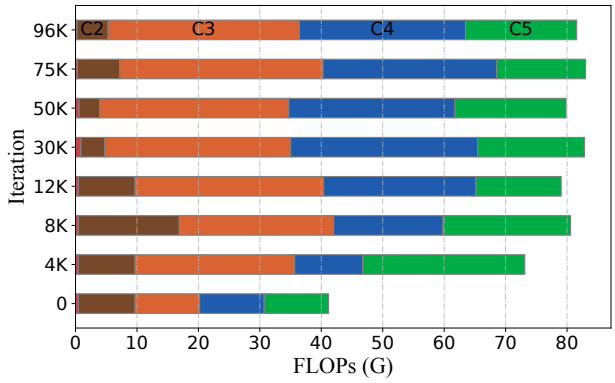


(c) IterationsP V.S. Parameters

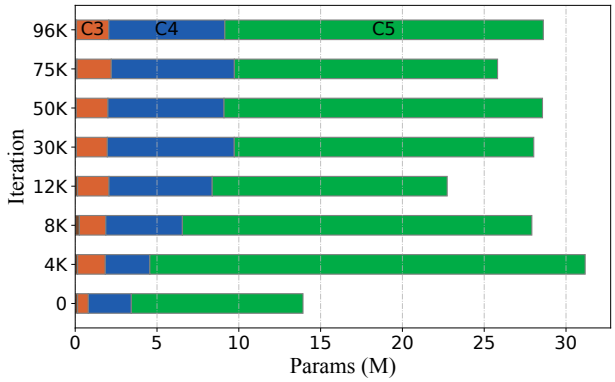
Figure 4: Visualization of single-scale entropy searching process. #layer is the number of each block of different levels.



(a) Iterations V.S. #Layers

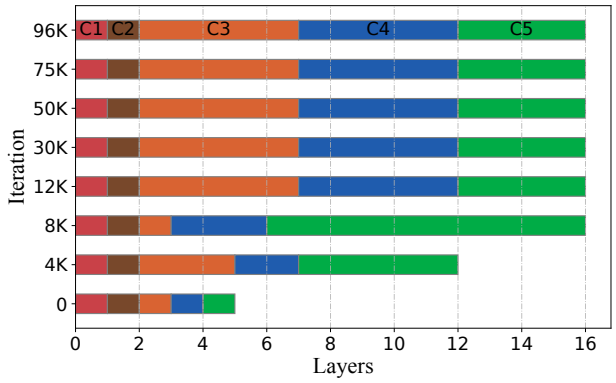


(b) Iterations V.S. FLOPs

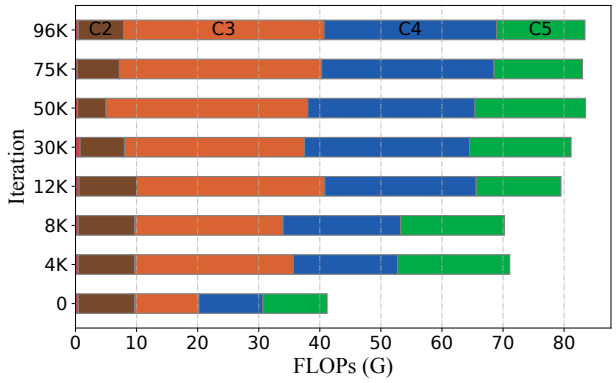


(c) IterationsP V.S. Parameters

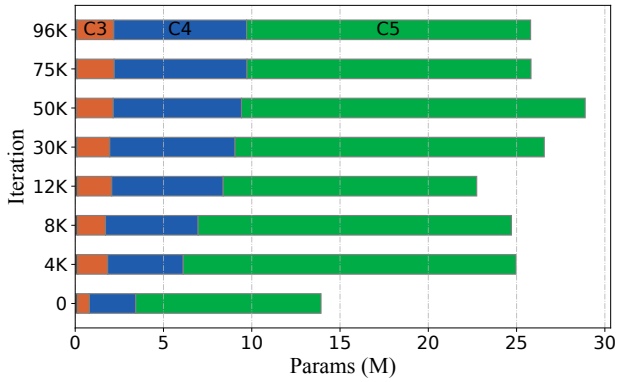
Figure 5: Visualization of multi-scale entropy searching process (Coarse). #layer is the number of each block of different levels.



(a) Iterations V.S. #Layers



(b) Iterations V.S. FLOPs



(c) IterationsP V.S. Parameters

Figure 6: Visualization of single-scale entropy searching process (Coarse-to-Fine). #layer is the number of each block of different levels.