EfficientDet: Scalable and Efficient Object Detection

Mingxing Tan Ruoming Pang Quoc V. Le
Google Research, Brain Team
{tanmingxing, rpang, qvl}@google.com

Abstract

*Model efficiency has become increasingly important in computer vision. In this paper, we systematically study neural network architecture design choices for object detection and propose several key optimizations to improve efficiency. First, we propose a weighted bi-directional feature pyramid network (BiFPN), which allows easy and fast multi-scale feature fusion; Second, we propose a compound scaling method that uniformly scales the resolution, depth, and width for all backbone, feature network, and box/class prediction networks at the same time. Based on these optimizations and better backbones, we have developed a new family of object detectors, called EfficientDet, which consistently achieve much better efficiency than prior art across a wide spectrum of resource constraints. In particular, with single-model and single-scale, our EfficientDet-D7 achieves state-of-the-art **55.1 AP** on COCO `test-dev` with 77M parameters and 410B FLOPs[1], being **4x – 9x** smaller and using **13x – 42x** fewer FLOPs than previous detectors. Code is available at https://github.com/google/automl/tree/master/efficientdet.*

1. Introduction

Tremendous progresses have been made in recent years towards more accurate object detection; meanwhile, state-of-the-art object detectors also become increasingly more expensive. For example, the latest AmoebaNet-based NAS-FPN detector [45] requires 167M parameters and 3045B FLOPs (30x more than RetinaNet [24]) to achieve state-of-the-art accuracy. The large model sizes and expensive computation costs deter their deployment in many real-world applications such as robotics and self-driving cars where model size and latency are highly constrained. Given these real-world resource constraints, model efficiency becomes increasingly important for object detection.

There have been many previous works aiming to develop more efficient detector architectures, such as one-stage [27, 33, 34, 24] and anchor-free detectors [21, 44, 40], or compress existing models [28, 29]. Although these methods tend to achieve better efficiency, they usually sacrifice accuracy. Moreover, most previous works only focus on a specific or a small range of resource requirements, but the variety of real-world applications, from mobile devices to datacenters, often demand different resource constraints.

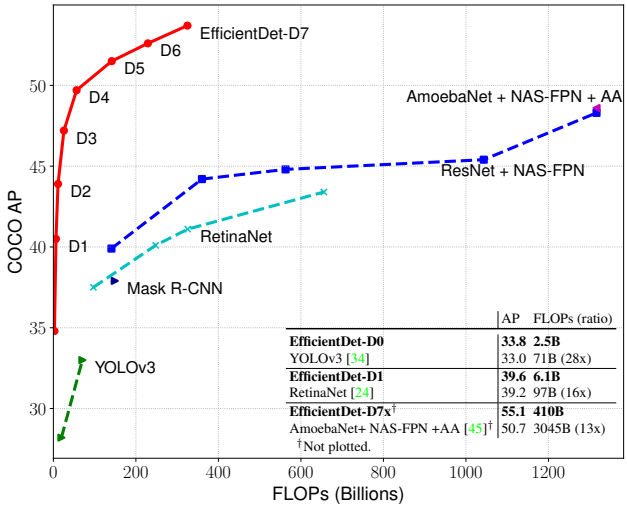


Figure 1: **Model FLOPs vs. COCO accuracy –** All numbers are for single-model single-scale. Our EfficientDet achieves new state-of-the-art 55.1% COCO AP with much fewer parameters and FLOPs than previous detectors. More studies on different backbones and FPN/NAS-FPN/BiFPN are in Table 4 and 5. Complete results are in Table 2.

A natural question is: Is it possible to build a *scalable detection architecture* with both *higher accuracy* and *better efficiency* across a wide spectrum of resource constraints (*e.g.*, from 3B to 300B FLOPs)? This paper aims to tackle this problem by systematically studying various design choices of detector architectures. Based on the one-stage detector paradigm, we examine the design choices for backbone, feature fusion, and class/box network, and identify two main challenges:

Challenge 1: efficient multi-scale feature fusion – Since introduced in [23], FPN has been widely used for multi-

[1]Similar to [14, 39], FLOPs denotes number of multiply-adds.

scale feature fusion. Recently, PANet [26], NAS-FPN [10], and other studies [20, 18, 42] have developed more network structures for cross-scale feature fusion. While fusing different input features, most previous works simply sum them up without distinction; however, since these different input features are at different resolutions, we observe they usually contribute to the fused output feature unequally. To address this issue, we propose a simple yet highly effective weighted bi-directional feature pyramid network (BiFPN), which introduces learnable weights to learn the importance of different input features, while repeatedly applying top-down and bottom-up multi-scale feature fusion.

Challenge 2: model scaling – While previous works mainly rely on bigger backbone networks [24, 35, 34, 10] or larger input image sizes [13, 45] for higher accuracy, we observe that scaling up feature network and box/class prediction network is also critical when taking into account both accuracy and efficiency. Inspired by recent works [39], we propose a compound scaling method for object detectors, which jointly scales up the resolution/depth/width for all backbone, feature network, box/class prediction network.

Finally, we also observe that the recently introduced EfficientNets [39] achieve better efficiency than previous commonly used backbones. Combining EfficientNet backbones with our propose BiFPN and compound scaling, we have developed a new family of object detectors, named EfficientDet, which consistently achieve better accuracy with much fewer parameters and FLOPs than previous object detectors. Figure 1 and Figure 4 show the performance comparison on COCO dataset [25]. Under similar accuracy constraint, our EfficientDet uses 28x fewer FLOPs than YOLOv3 [34], 30x fewer FLOPs than RetinaNet [24], and 19x fewer FLOPs than the recent ResNet based NAS-FPN [10]. In particular, with single-model and single test-time scale, our EfficientDet-D7 achieves state-of-the-art 55.1 AP with 77M parameters and 410B FLOPs, outperforming previous best detector [45] by 4 AP while being 2.7x smaller and using 7.4x fewer FLOPs. Our EfficientDet is also up to 4x to 11x faster on GPU/CPU than previous detectors.

With simple modifications, we also demonstrate that our single-model single-scale EfficientDet achieves 81.74% mIOU accuracy with 18B FLOPs on Pascal VOC 2012 semantic segmentation, outperforming DeepLabV3+ [6] by 1.7% better accuracy with 9.8x fewer FLOPs.

2. Related Work

One-Stage Detectors: Existing object detectors are mostly categorized by whether they have a region-of-interest proposal step (two-stage [11, 35, 5, 13]) or not (one-stage [36, 27, 33, 24]). While two-stage detectors tend to be more flexible and more accurate, one-stage detectors are often considered to be simpler and more efficient by leveraging predefined anchors [17]. Recently, one-stage detectors have attracted substantial attention due to their efficiency and simplicity [21, 42, 44]. In this paper, we mainly follow the one-stage detector design, and we show it is possible to achieve both better efficiency and higher accuracy with optimized network architectures.

Multi-Scale Feature Representations: One of the main difficulties in object detection is to effectively represent and process multi-scale features. Earlier detectors often directly perform predictions based on the pyramidal feature hierarchy extracted from backbone networks [4, 27, 36]. As one of the pioneering works, feature pyramid network (FPN) [23] proposes a top-down pathway to combine multi-scale features. Following this idea, PANet [26] adds an extra bottom-up path aggregation network on top of FPN; STDL [43] proposes a scale-transfer module to exploit cross-scale features; M2det [42] proposes a U-shape module to fuse multi-scale features, and G-FRNet [2] introduces gate units for controlling information flow across features. More recently, NAS-FPN [10] leverages neural architecture search to automatically design feature network topology. Although it achieves better performance, NAS-FPN requires thousands of GPU hours during search, and the resulting feature network is irregular and thus difficult to interpret. In this paper, we aim to optimize multi-scale feature fusion with a more intuitive and principled way.

Model Scaling: In order to obtain better accuracy, it is common to scale up a baseline detector by employing bigger backbone networks (*e.g.*, from mobile-size models [38, 16] and ResNet [14], to ResNeXt [41] and AmoebaNet [32]), or increasing input image size (*e.g.*, from 512x512 [24] to 1536x1536 [45]). Some recent works [10, 45] show that increasing the channel size and repeating feature networks can also lead to higher accuracy. These scaling methods mostly focus on single or limited scaling dimensions. Recently, [39] demonstrates remarkable model efficiency for image classification by jointly scaling up network width, depth, and resolution. Our proposed compound scaling method for object detection is mostly inspired by [39].

3. BiFPN

In this section, we first formulate the multi-scale feature fusion problem, and then introduce the main ideas for our proposed BiFPN: efficient bidirectional cross-scale connections and weighted feature fusion.

3.1. Problem Formulation

Multi-scale feature fusion aims to aggregate features at different resolutions. Formally, given a list of multi-scale features $\vec{P}^{in} = (P^{in}_{l_1}, P^{in}_{l_2}, ...)$, where $P^{in}_{l_i}$ represents the feature at level l_i, our goal is to find a transformation f that can effectively aggregate different features and output a list of new features: $\vec{P}^{out} = f(\vec{P}^{in})$. As a concrete example,

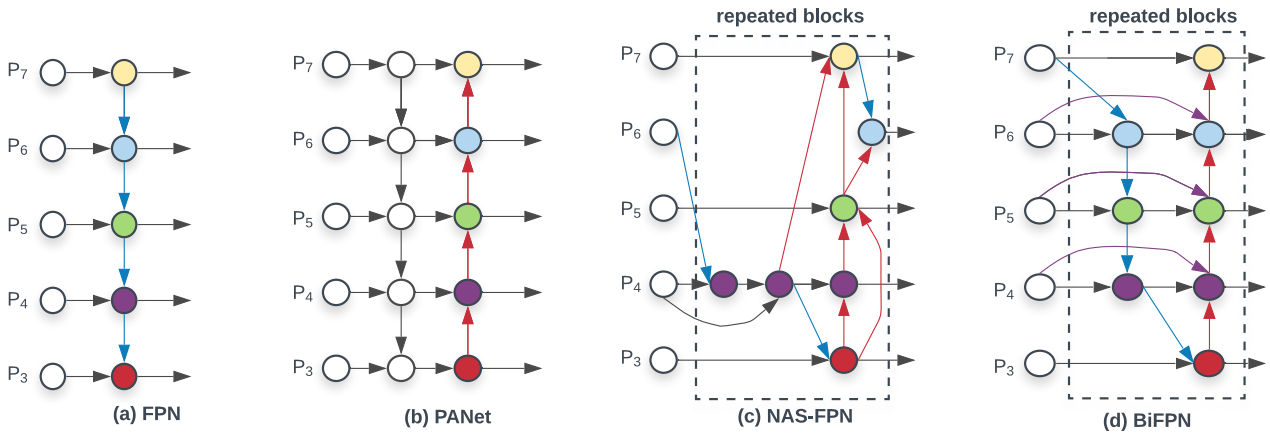


Figure 2: **Feature network design –** (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

Figure 2(a) shows the conventional top-down FPN [23]. It takes level 3-7 input features $\vec{P}^{in} = (P_3^{in}, ...P_7^{in})$, where P_i^{in} represents a feature level with resolution of $1/2^i$ of the input images. For instance, if input resolution is 640x640, then P_3^{in} represents feature level 3 $(640/2^3 = 80)$ with resolution 80x80, while P_7^{in} represents feature level 7 with resolution 5x5. The conventional FPN aggregates multi-scale features in a top-down manner:

$$
\begin{aligned}
P_7^{out} &= Conv(P_7^{in}) \\
P_6^{out} &= Conv(P_6^{in} + Resize(P_7^{out})) \\
&\dots \\
P_3^{out} &= Conv(P_3^{in} + Resize(P_4^{out}))
\end{aligned}
$$

where $Resize$ is usually a upsampling or downsampling op for resolution matching, and $Conv$ is usually a convolutional op for feature processing.

3.2. Cross-Scale Connections

Conventional top-down FPN is inherently limited by the one-way information flow. To address this issue, PANet [26] adds an extra bottom-up path aggregation network, as shown in Figure 2(b). Cross-scale connections are further studied in [20, 18, 42]. Recently, NAS-FPN [10] employs neural architecture search to search for better cross-scale feature network topology, but it requires thousands of GPU hours during search and the found network is irregular and difficult to interpret or modify, as shown in Figure 2(c).

By studying the performance and efficiency of these three networks (Table 5), we observe that PANet achieves better accuracy than FPN and NAS-FPN, but with the cost of more parameters and computations. To improve model efficiency, this paper proposes several optimizations for cross-scale connections: First, we remove those nodes that only have one input edge. Our intuition is simple: if a node has only one input edge with no feature fusion, then it will have less contribution to feature network that aims at fusing different features. This leads to a simplified bidirectional network; Second, we add an extra edge from the original input to output node if they are at the same level, in order to fuse more features without adding much cost; Third, unlike PANet [26] that only has one top-down and one bottom-up path, we treat each bidirectional (top-down & bottom-up) path as one feature network layer, and repeat the same layer multiple times to enable more high-level feature fusion. Section 4.2 will discuss how to determine the number of layers for different resource constraints using a compound scaling method. With these optimizations, we name the new feature network as bidirectional feature pyramid network (BiFPN), as shown in Figure 2 and 3.

3.3. Weighted Feature Fusion

When fusing features with different resolutions, a common way is to first resize them to the same resolution and then sum them up. Pyramid attention network [22] introduces global self-attention upsampling to recover pixel localization, which is further studied in [10]. All previous methods treat all input features equally without distinction. However, we observe that since different input features are at different resolutions, they usually contribute to the output feature *unequally*. To address this issue, we propose to add an additional weight for each input, and let the network to learn the importance of each input feature. Based on this idea, we consider three weighted fusion approaches:

Unbounded fusion: $O = \sum_i w_i \cdot I_i$, where w_i is a

learnable weight that can be a scalar (per-feature), a vector (per-channel), or a multi-dimensional tensor (per-pixel). We find a scale can achieve comparable accuracy to other approaches with minimal computational costs. However, since the scalar weight is unbounded, it could potentially cause training instability. Therefore, we resort to weight normalization to bound the value range of each weight.

Softmax-based fusion: $O = \sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot I_i$. An intuitive idea is to apply softmax to each weight, such that all weights are normalized to be a probability with value range from 0 to 1, representing the importance of each input. However, as shown in our ablation study in section 6.3, the extra softmax leads to significant slowdown on GPU hardware. To minimize the extra latency cost, we further propose a fast fusion approach.

Fast normalized fusion: $O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$, where $w_i \geq 0$ is ensured by applying a Relu after each w_i, and $\epsilon = 0.0001$ is a small value to avoid numerical instability. Similarly, the value of each normalized weight also falls between 0 and 1, but since there is no softmax operation here, it is much more efficient. Our ablation study shows this fast fusion approach has very similar learning behavior and accuracy as the softmax-based fusion, but runs up to 30% faster on GPUs (Table 6).

Our final BiFPN integrates both the bidirectional cross-scale connections and the fast normalized fusion. As a concrete example, here we describe the two fused features at level 6 for BiFPN shown in Figure 2(d):

$$P_6^{td} = Conv\left(\frac{w_1 \cdot P_6^{in} + w_2 \cdot Resize(P_7^{in})}{w_1 + w_2 + \epsilon}\right)$$
$$P_6^{out} = Conv\left(\frac{w_1' \cdot P_6^{in} + w_2' \cdot P_6^{td} + w_3' \cdot Resize(P_5^{out})}{w_1' + w_2' + w_3' + \epsilon}\right)$$

where P_6^{td} is the intermediate feature at level 6 on the top-down pathway, and P_6^{out} is the output feature at level 6 on the bottom-up pathway. All other features are constructed in a similar manner. Notably, to further improve the efficiency, we use depthwise separable convolution [7, 37] for feature fusion, and add batch normalization and activation after each convolution.

4. EfficientDet

Based on our BiFPN, we have developed a new family of detection models named EfficientDet. In this section, we will discuss the network architecture and a new compound scaling method for EfficientDet.

4.1. EfficientDet Architecture

Figure 3 shows the overall architecture of EfficientDet, which largely follows the one-stage detectors paradigm [27, 33, 23, 24]. We employ ImageNet-pretrained EfficientNets as the backbone network. Our proposed BiFPN serves as the feature network, which takes level 3-7 features $\{P_3, P_4, P_5, P_6, P_7\}$ from the backbone network and repeatedly applies top-down and bottom-up bidirectional feature fusion. These fused features are fed to a class and box network to produce object class and bounding box predictions respectively. Similar to [24], the class and box network weights are shared across all levels of features.

4.2. Compound Scaling

Aiming at optimizing both accuracy and efficiency, we would like to develop a family of models that can meet a wide spectrum of resource constraints. A key challenge here is how to scale up a baseline EfficientDet model.

Previous works mostly scale up a baseline detector by employing bigger backbone networks (*e.g.*, ResNeXt [41] or AmoebaNet [32]), using larger input images, or stacking more FPN layers [10]. These methods are usually ineffective since they only focus on a single or limited scaling dimensions. Recent work [39] shows remarkable performance on image classification by jointly scaling up all dimensions of network width, depth, and input resolution. Inspired by these works [10, 39], we propose a new compound scaling method for object detection, which uses a simple compound coefficient ϕ to jointly scale up all dimensions of backbone , BiFPN, class/box network, and resolution. Unlike [39], object detectors have much more scaling dimensions than image classification models, so grid search for all dimensions is prohibitive expensive. Therefore, we use a heuristic-based scaling approach, but still follow the main idea of jointly scaling up all dimensions.

Backbone network – we reuse the same width/depth scaling coefficients of EfficientNet-B0 to B6 [39] such that we can easily reuse their ImageNet-pretrained checkpoints.

BiFPN network – we linearly increase BiFPN depth D_{bifpn} (#layers) since depth needs to be rounded to small integers. For BiFPN width W_{bifpn} (#channels), exponentially grow BiFPN width W_{bifpn} (#channels) as similar to [39]. Specifically, we perform a grid search on a list of values $\{1.2, 1.25, 1.3, 1.35, 1.4, 1.45\}$, and pick the best value 1.35 as the BiFPN width scaling factor. Formally, BiFPN width and depth are scaled with the following equation:

$$W_{bifpn} = 64 \cdot \left(1.35^{\phi}\right), \qquad D_{bifpn} = 3 + \phi \qquad (1)$$

Box/class prediction network – we fix their width to be always the same as BiFPN (i.e., $W_{pred} = W_{bifpn}$), but lin-

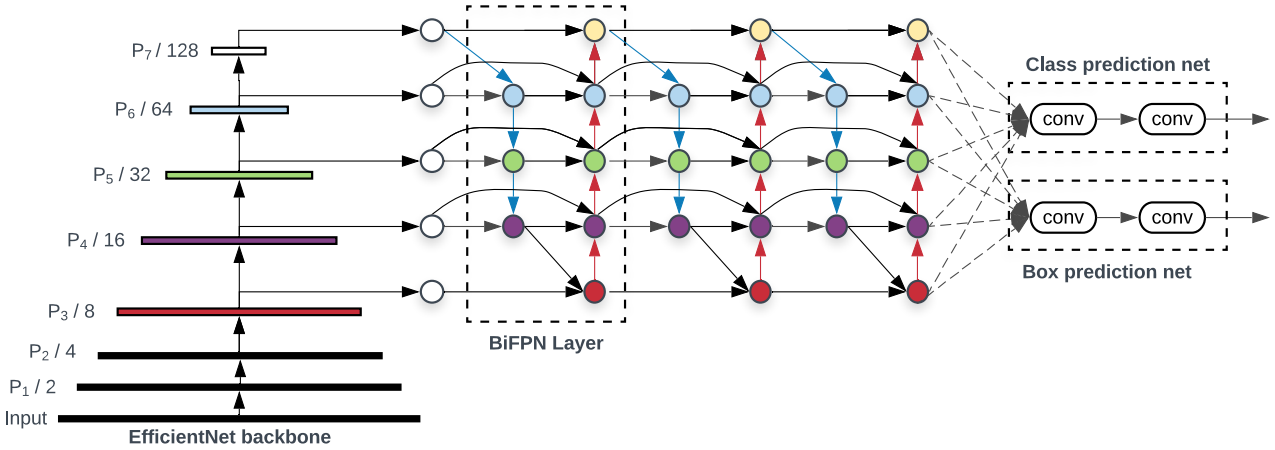


Figure 3: **EfficientDet architecture –** It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

early increase the depth (#layers) using equation:

$$D_{box} = D_{class} = 3 + \lfloor \phi/3 \rfloor \quad (2)$$

Input image resolution – Since feature level 3-7 are used in BiFPN, the input resolution must be dividable by $2^7 = 128$, so we linearly increase resolutions using equation:

$$R_{input} = 512 + \phi \cdot 128 \quad (3)$$

Following Equations 1,2,3 with different ϕ, we have developed EfficientDet-D0 ($\phi = 0$) to D7 ($\phi = 7$) as shown in Table 1, where D7 and D7x have the same BiFPN and head, but D7 uses higher resolution and D7x uses larger backbone network and one more feature level (from P_3 to P_8). Notably, our compound scaling is heuristic-based and might not be optimal, but we will show that this simple scaling method can significantly improve efficiency than other single-dimension scaling methods in Figure 6.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Table 1: **Scaling configs for EfficientDet D0-D6 –** ϕ is the compound coefficient that controls all other scaling dimensions; *BiFPN, box/class net, and input size* are scaled up using equation 1, 2, 3 respectively.

5. Experiments

5.1. EfficientDet for Object Detection

We evaluate EfficientDet on COCO 2017 detection datasets [25] with 118K training images. Each model is trained using SGD optimizer with momentum 0.9 and weight decay 4e-5. Learning rate is linearly increased from 0 to 0.16 in the first training epoch and then annealed down using cosine decay rule. Synchronized batch norm is added after every convolution with batch norm decay 0.99 and epsilon 1e-3. Same as the [39], we use SiLU (Swish-1) activation [8, 15, 31] and exponential moving average with decay 0.9998. We also employ commonly-used focal loss [24] with $\alpha = 0.25$ and $\gamma = 1.5$, and aspect ratio $\{1/2, 1, 2\}$. During training, we apply horizontal flipping and scale jittering [0.1, 2.0], which randomly rsizes images between 0.1x and 2.0x of the original size before cropping. We apply soft-NMS [3] for eval. For D0-D6, each model is trained for 300 epochs with total batch size 128 on 32 TPUv3 cores, but to push the envelope, we train D7/D7x for 600 epochs on 128 TPUv3 cores.

Table 2 compares EfficientDet with other object detectors, under the single-model single-scale settings with no test-time augmentation. We report accuracy for both `test-dev` (20K test images with no public ground-truth) and `val` with 5K validation images. Notably, model performance depends on both network architecture and trainning settings (see appendix), but for simplicity, we only reproduce RetinaNet using our trainers and refer other models from their papers. In general, our EfficientDet achieves bet-

Model	test-dev AP	test-dev AP_{50}	test-dev AP_{75}	val AP	Params	Ratio	FLOPs	Ratio	Latency (ms) TitianV	Latency (ms) V100
EfficientDet-D0 (512)	**34.6**	**53.0**	**37.1**	**34.3**	**3.9M**	**1x**	**2.5B**	**1x**	**12**	**10.2**
YOLOv3 [34]	33.0	57.9	34.4	-	-	-	71B	28x	-	-
EfficientDet-D1 (640)	**40.5**	**59.1**	**43.7**	**40.2**	**6.6M**	**1x**	**6.1B**	**1x**	**16**	**13.5**
RetinaNet-R50 (640) [24]	39.2	58.0	42.3	39.2	34M	6.7x	97B	16x	25	-
RetinaNet-R101 (640)[24]	39.9	58.5	43.0	39.8	53M	8.0x	127B	21x	32	-
EfficientDet-D2 (768)	**43.9**	**62.7**	**47.6**	**43.5**	**8.1M**	**1x**	**11B**	**1x**	**23**	**17.7**
Detectron2 Mask R-CNN R101-FPN [1]	-	-	-	42.9	63M	7.7x	164B	15x	-	56‡
Detectron2 Mask R-CNN X101-FPN [1]	-	-	-	44.3	107M	13x	277B	25x	-	103‡
EfficientDet-D3 (896)	**47.2**	**65.9**	**51.2**	**46.8**	**12M**	**1x**	**25B**	**1x**	**37**	**29.0**
ResNet-50 + NAS-FPN (1024) [10]	44.2	-	-	-	60M	5.1x	360B	15x	64	-
ResNet-50 + NAS-FPN (1280) [10]	44.8	-	-	-	60M	5.1x	563B	23x	99	-
ResNet-50 + NAS-FPN (1280@384)[10]	45.4	-	-	-	104M	8.7x	1043B	42x	150	-
EfficientDet-D4 (1024)	**49.7**	**68.4**	**53.9**	**49.3**	**21M**	**1x**	**55B**	**1x**	**65**	**42.8**
AmoebaNet+ NAS-FPN +AA(1280)[45]	-	-	-	48.6	185M	8.8x	1317B	24x	246	-
EfficientDet-D5 (1280)	**51.5**	**70.5**	**56.1**	**51.3**	**34M**	**1x**	**135B**	**1x**	**128**	**72.5**
Detectron2 Mask R-CNN X152 [1]	-	-	-	50.2	-	-	-	-	-	234‡
EfficientDet-D6 (1280)	**52.6**	**71.5**	**57.2**	**52.2**	**52M**	**1x**	**226B**	**1x**	**169**	**92.8**
AmoebaNet+ NAS-FPN +AA(1536)[45]	-	-	-	50.7	209M	4.0x	3045B	13x	489	-
EfficientDet-D7 (1536)	**53.7**	**72.4**	**58.4**	**53.4**	**52M**		**325B**		**232**	**122**
EfficientDet-D7x (1536)	**55.1**	**74.3**	**59.9**	**54.4**	**77M**		**410B**		**285**	**153**

We omit ensemble and test-time multi-scale results [30, 12]. RetinaNet APs are reproduced with our trainer and others are from papers.
‡Latency numbers with ‡ are from detectron2, and others are measured on the same machine (TensorFlow2.1 + CUDA10.1, no TensorRT).

Table 2: **EfficientDet performance on COCO** [25] – Results are for single-model single-scale. `test-dev` is the COCO test set and `val` is the validation set. `Params` and `FLOPs` denote the number of parameters and multiply-adds. `Latency` is for inference with batch size 1. `AA` denotes auto-augmentation [45]. We group models together if they have similar accuracy, and compare their model size, FLOPs, and latency in each group.

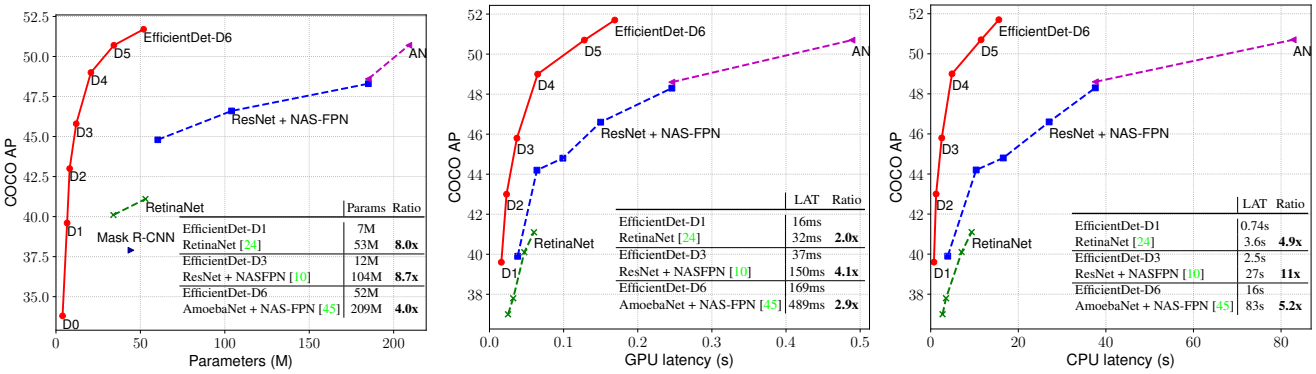
ter efficiency than previous detectors, being **4x – 9x** smaller and using **13x - 42x** less FLOPs across a wide range of accuracy or resource constraints. On relatively low-accuracy regime, our EfficientDet-D0 achieves similar accuracy as YOLOv3 with 28x fewer FLOPs. Compared to RetinaNet [24] and Mask-RCNN [13], our EfficientDet achieves similar accuracy with up to 8x fewer parameters and 21x fewer FLOPs. On high-accuracy regime, our EfficientDet also consistently outperforms recent object detectors [10, 45] with much fewer parameters and FLOPs. In particular, our single-model single-scale EfficientDet-D7x achieves a new state-of-the-art **55.1** AP on `test-dev`, outperforming prior art by a large margin in both accuracy (+4 AP) and efficiency (7x fewer FLOPs).

In addition, we have also compared the inference latency on Titan-V FP32 , V100 GPU FP16, and single-thread CPU. Notably, our V100 latency is end-to-end including preprocessing and NMS postprocessing. Figure 4 illustrates the comparison on model size and GPU/CPU latency. For fair comparison, these figures only include results that are measured on the same machine with the same settings. Compared to previous detectors, EfficientDet models are up to 4.1x faster on GPU and 10.8x faster on CPU, suggesting they are also efficient on real-world hardware.

5.2. EfficientDet for Semantic Segmentation

While our EfficientDet models are mainly designed for object detection, we are also interested in their performance on other tasks such as semantic segmentation. Following [19], we modify our EfficientDet model to keep feature level $\{P2, P3, ..., P7\}$ in BiFPN, but only use $P2$ for the final per-pixel classification. For simplicity, here we only evaluate a EfficientDet-D4 based model, which uses a ImageNet pretrained EfficientNet-B4 backbone (similar size to ResNet-50). We set the channel size to 128 for BiFPN and 256 for classification head. Both BiFPN and classification head are repeated by 3 times.

Table 3 shows the comparison between our models and previous DeepLabV3+ [6] on Pascal VOC 2012 [9]. Notably, we exclude those results with ensemble, test-time augmentation, or COCO pretraining. Under the same single-model single-scale settings, our model achieves 1.7% better accuracy with 9.8x fewer FLOPs than the prior art of DeepLabV3+ [6]. These results suggest that EfficientDet is also quite promising for semantic segmentation.



(a) Model Size (b) GPU Latency (c) CPU Latency

Figure 4: **Model size and inference latency comparison** – Latency is measured with batch size 1 on the same machine equipped with a Titan V GPU and Xeon CPU. AN denotes AmoebaNet + NAS-FPN trained with auto-augmentation [45]. Our EfficientDet models are 4x - 9x smaller, 2x - 4x faster on GPU, and 5x - 11x faster on CPU than other detectors.

Model	mIOU	Params	FLOPs
DeepLabV3+ (ResNet-101) [6]	79.35%	-	298B
DeepLabV3+ (Xception) [6]	80.02%	-	177B
Our EfficientDet†	**81.74%**	**17M**	**18B**

†A modified version of EfficientDet-D4.

Table 3: **Performance comparison on Pascal VOC semantic segmentation**.

	AP	Parameters	FLOPs
ResNet50 + FPN	37.0	34M	97B
EfficientNet-B3 + FPN	40.3	21M	75B
EfficientNet-B3 + **BiFPN**	44.4	12M	24B

Table 4: **Disentangling backbone and BiFPN –** Starting from the standard RetinaNet (ResNet50+FPN), we first replace the backbone with EfficientNet-B3, and then replace the baseline FPN with our proposed BiFPN.

6. Ablation Study

In this section, we ablate various design choices for our proposed EfficientDet. For simplicity, all accuracy results here are for COCO validation set.

6.1. Disentangling Backbone and BiFPN

Since EfficientDet uses both a powerful backbone and a new BiFPN, we want to understand how much each of them contributes to the accuracy and efficiency improvements. Table 4 compares the impact of backbone and BiFPN using RetinaNet training settings. Starting from a RetinaNet detector [24] with ResNet-50 [14] backbone and top-down FPN [23], we first replace the backbone with EfficientNet-B3, which improves accuracy by about 3 AP with slightly less parameters and FLOPs. By further replacing FPN with our proposed BiFPN, we achieve additional 4 AP gain with much fewer parameters and FLOPs. These results suggest that EfficientNet backbones and BiFPN are both crucial for our final models.

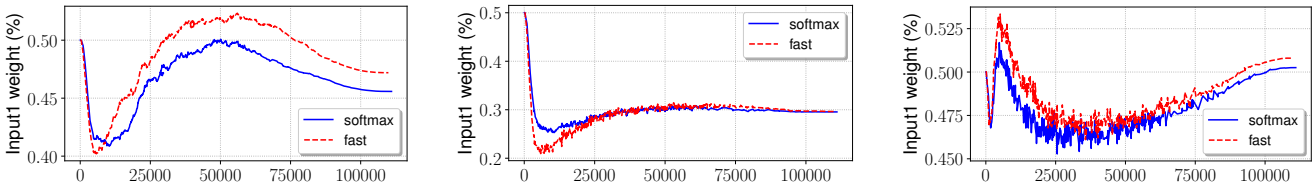
6.2. BiFPN Cross-Scale Connections

Table 5 shows the accuracy and model complexity for feature networks with different cross-scale connections listed in Figure 2. Notably, the original FPN [23] and PANet [26] only have one top-down or bottom-up flow, but for fair comparison, here we repeat each of them multiple times and replace all convs with depthwise separable convs, which is the same as BiFPN. We use the same backbone and class/box prediction network, and the same training settings for all experiments. As we can see, the conventional top-down FPN is inherently limited by the one-way information flow and thus has the lowest accuracy. While repeated FPN+PANet achieves slightly better accuracy than NAS-FPN [10], it also requires more parameters and FLOPs. Our BiFPN achieves similar accuracy as repeated FPN+PANet, but uses much less parameters and FLOPs. With the additional weighted feature fusion, our BiFPN further achieves the best accuracy with fewer parameters and FLOPs.

6.3. Softmax vs Fast Normalized Fusion

As discussed in Section 3.3, we propose a fast normalized feature fusion approach to get ride of the expensive softmax while retaining the benefits of normalized weights. Table 6 compares the softmax and fast normalized fusion approaches in three detectors with different model sizes. As shown in the results, our fast normalized fusion approach achieves similar accuracy as the softmax-based fusion, but runs 1.26x - 1.31x faster on GPUs.

In order to further understand the behavior of softmax-based and fast normalized fusion, Figure 5 illustrates the



(a) Example Node 1 (b) Example Node 2 (c) Example Node 3

Figure 5: **Softmax vs. fast normalized feature fusion –** (a) - (c) shows normalized weights (i.e., importance) during training for three representative nodes; each node has two inputs (input1 & input2) and their normalized weights always sum up to 1.

	AP	#Params ratio	#FLOPs ratio
Repeated top-down FPN	42.29	1.0x	1.0x
Repeated FPN+PANet	44.08	1.0x	1.0x
NAS-FPN	43.16	0.71x	0.72x
Fully-Connected FPN	43.06	1.24x	1.21x
BiFPN (w/o weighted)	**43.94**	**0.88x**	**0.67x**
BiFPN (w/ weighted)	**44.39**	**0.88x**	**0.68x**

Table 5: **Comparison of different feature networks –** Our weighted BiFPN achieves the best accuracy with fewer parameters and FLOPs.

Model	Softmax Fusion AP	Fast Fusion AP (delta)	Speedup
Model1	33.96	33.85 (-0.11)	1.28x
Model2	43.78	43.77 (-0.01)	1.26x
Model3	48.79	48.74 (-0.05)	1.31x

Table 6: **Comparison of different feature fusion –** Our fast fusion achieves similar accuracy as softmax-based fusion, but runs 28% - 31% faster.

learned weights for three feature fusion nodes randomly selected from the BiFPN layers in EfficientDet-D3. Notably, the normalized weights (*e.g.*, $e^{w_i}/\sum_j e^{w_j}$ for softmax-based fusion, and $w_i/(\epsilon + \sum_j w_j)$ for fast normalized fusion) always sum up to 1 for all inputs. Interestingly, the normalized weights change rapidly during training, suggesting different features contribute to the feature fusion unequally. Despite the rapid change, our fast normalized fusion approach always shows very similar learning behavior to the softmax-based fusion for all three nodes.

6.4. Compound Scaling

As discussed in section 4.2, we employ a compound scaling method to jointly scale up all dimensions of depth/width/resolution for backbone, BiFPN, and box/class prediction networks. Figure 6 compares our compound scaling with other alternative methods that scale up a single dimension of resolution/depth/width. Although starting from the same baseline detector, our compound scaling method achieves better efficiency than other methods, suggesting the benefits of jointly scaling by better balancing difference architecture dimensions.

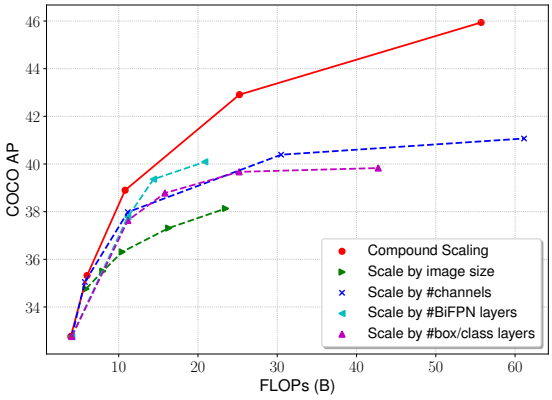


Figure 6: **Comparison of different scaling methods –** compound scaling achieves better accuracy and efficiency.

7. Conclusion

In this paper, we systematically study network architecture design choices for efficient object detection, and propose a weighted bidirectional feature network and a customized compound scaling method, in order to improve accuracy and efficiency. Based on these optimizations, we develop a new family of detectors, named *EfficientDet*, which consistently achieve better accuracy and efficiency than the prior art across a wide spectrum of resource constraints. In particular, our scaled EfficientDet achieves state-of-the-art accuracy with much fewer parameters and FLOPs than previous object detection and semantic segmentation models.

Acknowledgements

Special thanks to Golnaz Ghiasi, Adams Yu, Daiyi Peng for their help on infrastructure and discussion. We also thank Adam Kraft, Barret Zoph, Ekin D. Cubuk, Hongkun Yu, Jeff Dean, Pengchong Jin, Samy Bengio, Reed Wanderman-Milne, Tsung-Yi Lin, Xianzhi Du, Xiaodan Song, Yunxing Dai, and the Google Brain team. We

thank the open source community for the contributions.

References

[1] Detectron2. https://github.com/facebookresearch/detectron2. Accessed: 05/01/2020. 6, 10

[2] Md Amirul Islam, Mrigank Rochan, Neil DB Bruce, and Yang Wang. Gated feedback refinement network for dense image labeling. *CVPR*, pages 3751–3759, 2017. 2

[3] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. *ICCV*, pages 5561–5569, 2017. 5

[4] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. *ECCV*, pages 354–370, 2016. 2

[5] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. *CVPR*, pages 6154–6162, 2018. 2

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018. 2, 6, 7

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CVPR*, pages 1610–02357, 2017. 4

[8] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. 5

[9] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 2015. 6

[10] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. *CVPR*, 2019. 2, 3, 4, 6, 7

[11] Ross Girshick. Fast r-cnn. *ICCV*, 2015. 2

[12] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *ICCV*, 2019. 6, 10

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *ICCV*, pages 2980–2988, 2017. 2, 6

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016. 1, 2, 7

[15] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 5

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *ICCV*, 2019. 2

[17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *CVPR*, 2017. 2

[18] Seung-Wook Kim, Hyong-Keun Kook, Jee-Young Sun, Mun-Cheon Kang, and Sung-Jea Ko. Parallel feature pyramid network for object detection. *ECCV*, 2018. 2, 3

[19] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollr. Panoptic feature pyramid networks. *CVPR*, 2019. 6

[20] Tao Kong, Fuchun Sun, Chuanqi Tan, Huaping Liu, and Wenbing Huang. Deep feature pyramid reconfiguration for object detection. *ECCV*, 2018. 2, 3

[21] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *ECCV*, 2018. 1, 2

[22] Hanchao Li, Pengfei Xiong, Jie An, and Lingxue Wang. Pyramid attention networks. *BMVC*, 2018. 3

[23] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *CVPR*, 2017. 1, 2, 3, 4, 7

[24] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Focal loss for dense object detection. *ICCV*, 2017. 1, 2, 4, 5, 6, 7, 10

[25] 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. *ECCV*, 2014. 2, 5, 6

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

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

[28] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *ICLR*, 2019. 1

[29] Jonathan Pedoeem and Rachel Huang. Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers. *arXiv preprint arXiv:1811.05588*, 2018. 1

[30] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector, 2018. 6

[31] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *ICLR workshop*, 2018. 5

[32] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019. 2, 4

[33] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *CVPR*, 2017. 1, 2, 4

[34] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 1, 2, 6

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

[36] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *ICLR*, 2014. 2

[37] Laurent Sifre. Rigid-motion scattering for image classification. *Ph.D. thesis section 6.2*, 2014. 4

[38] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*, 2019. 2

[39] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019. 1, 2, 4, 5

[40] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. *ICCV*, 2019. 1

[41] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CVPR*, pages 5987–5995, 2017. 2, 4

[42] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. *AAAI*, 2019. 2, 3

[43] Peng Zhou, Bingbing Ni, Cong Geng, Jianguo Hu, and Yi Xu. Scale-transferrable object detection. *CVPR*, pages 528–537, 2018. 2

[44] Xingyi Zhou, Dequan Wang, and Philipp Krhenbhl. Objects as points. *arXiv:1904.07850*, 2019. 1, 2

[45] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection. *arXiv preprint arXiv:1804.02767*, 2019. 1, 2, 6, 7

Appendix

1.1. Hyperparameters

Neural network architecture and training hyperparamters are both crucial for object detection. Here we ablate two important hyperparamters: training epochs and multi-scale jittering, using RetinaNet-R50 and our EfficientDet-D1. All other hyperparameters are kept the same as section 5.

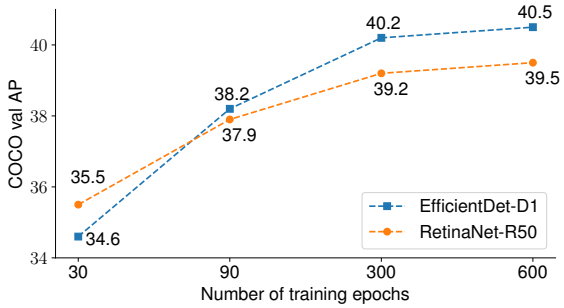


Figure 7: **Accuracy vs. Training Epochs.**

Training Epochs: Many previous work only use a small number of epochs: for example, Detectron2 [1] trains each model with 12 epochs (1x schedule) in default, and at most 110 epochs (9x scahedule). Recent work [12] shows training longer is not helpful if using pretrained backbone networks; however, we observe training longer can significiantly improve accuracy in our settings. Figure 7 shows the performance comparison for different training epochs. We obseve: (1) both models benefit from longer training until reaching 300 epochs; (2) longer training is particularly important for EfficientDet, perhaps due to its small model size; (3) compared to the default 37 AP [24], our reproduced RetinaNet achieves higher accuracy (+2AP) using our training settings. In this paper, we mainly use 300 epochs for the good trade-off between accuracy and training time.

Scale Jittering: A common training-time augmentation is to first resize images and then crop them into fixed size, known as scale jitterinig. Previous object detectors often use small jitters such as [0.8, 1.2], which randomly sample a scaling size between 0.8x to 1.2x of the original image size. However, we observe large jitters can improve accuracy if training longer. Figure 8 shows the results for different jitters: (1) when training with 30 epochs, a small jitter like [0.8, 1.2] performs quite good, and large jitters like [0.1, 2.0] actually hurts accuracy; (2) when training with 300 epochs, large jitters consistently improve accuracy, perhaps due to the stronger regularization. This paper uses a large jitter [0.1, 2.0] for all models.

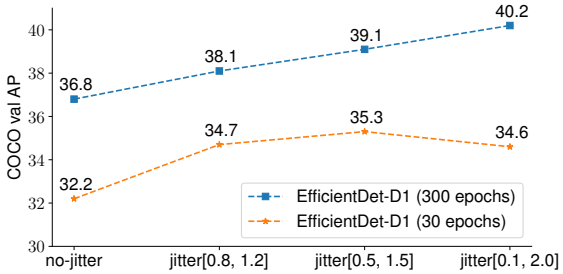


Figure 8: **Accuracy vs. Scale Jittering.**

1.2. Image Resolutions

In addition to our compound scaling that progressively increases image sizes, we are also interested in the accuracy-latency trade-offs with fixed image resolutions. Figure 9 compares EfficientDet-D1 to D6 with fixed and scaled resolutions. Surprisingly, their accuracy-latency trade-offs are very similar even though they have very different preferences: under similar accuracy constraints, models with fixed resolutions require much more parameters, but less activations and peak memory usage, than those with scaled resolutions. With fixed 640x640, our EfficientDet-D6 achieves real-time 47.9AP at 34ms latency.

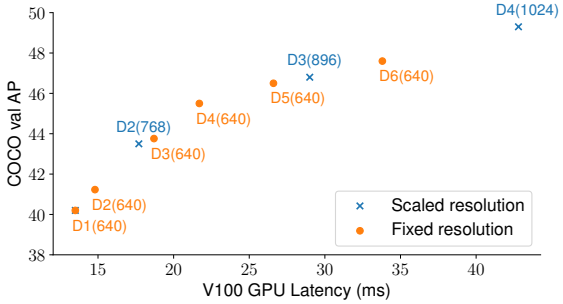


Figure 9: **Comparison for Fixed and Scaled Resolution** – *fixed* denotes 640x640 size and *scaled* denotes increased sizes.