Object Detection Task

EfficientDet
DETR

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering & Artificial Intelligence

References

Main

- https://theaisummer.com/cnn-architectures/

blog Sub

- https://deepbaksuvision.github.io/Modu_ObjectDetection/

PyTorch tutorial

- website : https://pytorch.org/
- Korea website : https://pytorch.kr/

github

- https://github.com/pytorch
- https://github.com/9bow/PyTorch-tutorials-kr
- torchvision : https://github.com/pytorch/vision
- https://github.com/weiaicunzai/pytorch-cifar100

SOTA of CNN Architecture

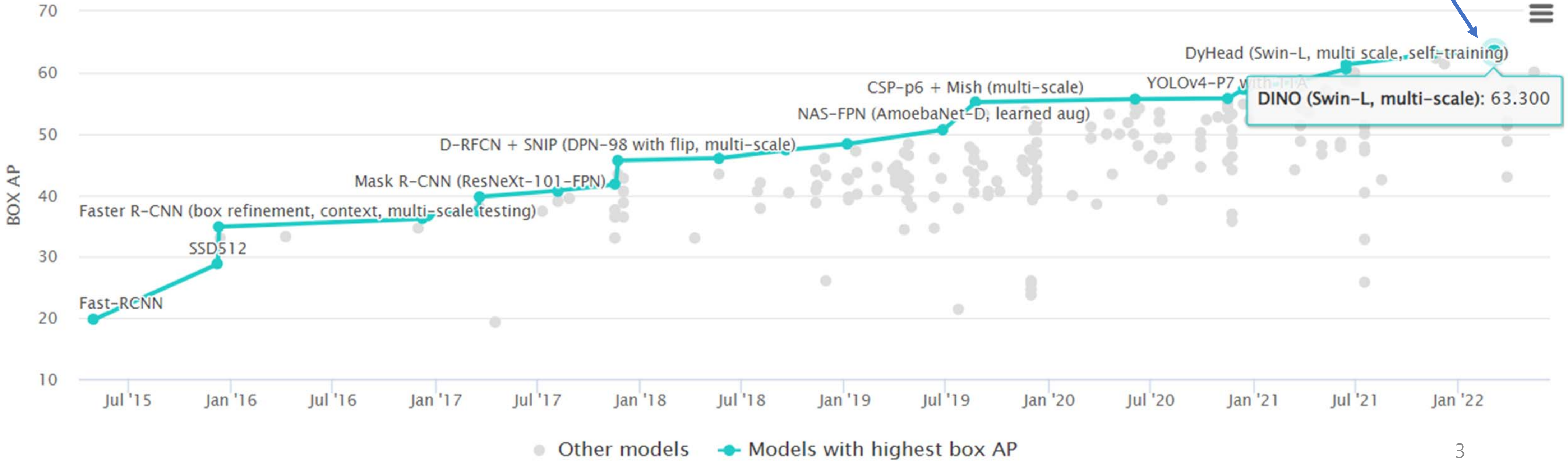
Part1

- Yolov4 / Volov5 / PP-YOLO / YOLOX / PP-YOLOE
- EfficientDet
- RegDet
- PANet++ / NAS-FPN / CSP
- SSD
- Fast R-CNN / Mask R-CNN / RetinNet

Part2

- UniverseNet
- Cascade Eff-B7 NAS-FPN
- DyHead / DINO
- DETR / Deformable DETR

DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection



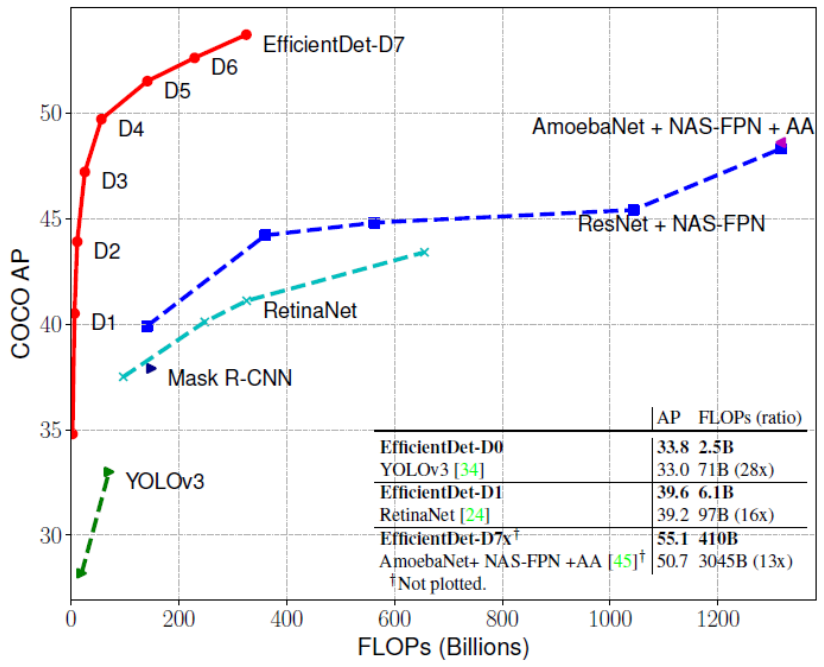
1 EfficientDet

Mingxing Tan, Ruoming Pang, Quoc V. Le, "EfficientDet: Scalable and Efficient Object Detection," CVPR2020

https://arxiv.org/abs/1911.09070

https://github.com/google/automl/tree/master/efficientdet

https://hoya012.github.io/blog/EfficientDet-Review/



	AP	FLOPs (ratio)
EfficientDet-D0	**33.8**	**2.5B**
YOLOv3 [34]	33.0	71B (28x)
EfficientDet-D1	**39.6**	**6.1B**
RetinaNet [24]	39.2	97B (16x)
EfficientDet-D7x†	**55.1**	**410B**
AmoebaNet+ NAS-FPN +AA [45]†	50.7	3045B (13x)

†Not plotted.

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.

Abstract

- **Model efficiency** has become increasingly important in computer vision.
- 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 multiscale 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 FLOPs1, being 4x – 9x smaller and using 13x – 42x fewer FLOPs than previous detectors.

EfficientDet

Introduction

- State of-the-art **object detectors 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
- Given these real-world resource constraints such as robotics and self-driving car, **model efficiency becomes increasingly important** for object detection.
  - ✓ Such as one-stage [27,33,34,24] and anchor-free detectors [21,44,40], or compress existing models [28,29].
- Although previous 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**,

Natural question

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)?

Introduction

Two Challenges

Challenge 1: Efficient multi-scale feature fusion

- **FPN** has been widely used for **multi-scale feature fusion**.
- **PANet** [26], **NAS-FPN** [10], and other studies [20,18,42] have developed more network structures for **cross-scale feature fusion**.
- **Most previous works simply sum them up without distinction;**
- **However, 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

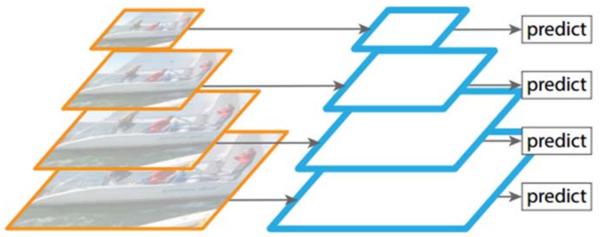
- Inspired by EfficientNet, we propose a compound scaling method for object detectors, which **jointly scales up the resolution/depth/width** for all backbone, feature network, box/class prediction network.

- Combining **EfficientNet backbones** with **BiFPN** and **compound scaling** → **EfficientDet**

BiFPN : Problem Formulation

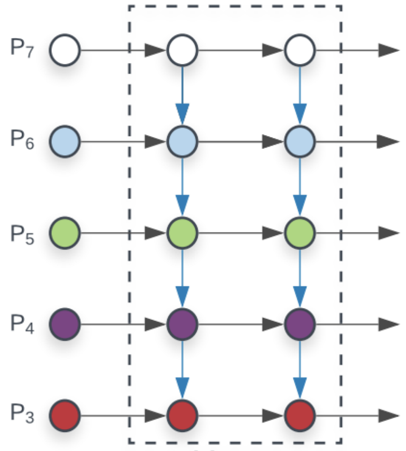
- Multi-scale feature fusion aims to aggregate features at different resolutions.
- Formally, given a list of multi-scale features $\vec{P}^{in} = (P^{in}_{l_1}, P^{in}_{l_2}, \cdots)$, where $P^{in}_{l_i}$ represents the feature at level l_i, our goal is to find a transformation f that can effectively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

- **FPN** takes level 3-7 input features $\vec{P}^{in} = (P^{in}_3, \cdots, P^{in}_7)$, where P^{in}_i represents a feature level with resolution of $1/2^i$ of the input images.
- For instance, if input resolution is 640x640, then P^{in}_3 represents feature level 3 ($640/2^3 = 80$) with resolution 80x80, while P^{in}_7 represents feature level 7 ($640/2^7 = 5$) with resolution 5x5.
- The conventional **FPN aggregates multi-scale features in a top-down manner**:



(a) Featurized image pyramid

$$P^{out}_7 = Conv(P^{in}_7)$$
$$P^{out}_6 = Conv(P^{in}_6 + Resize(P^{out}_7))$$
$$\cdots$$
$$P^{out}_3 = Conv(P^{in}_3 + Resize(P^{out}_4))$$



(a) FPN

BiFPN : Cross-Scale Connections

- **PANet** [26] adds an **extra bottom-up path aggregation network**.
- **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.

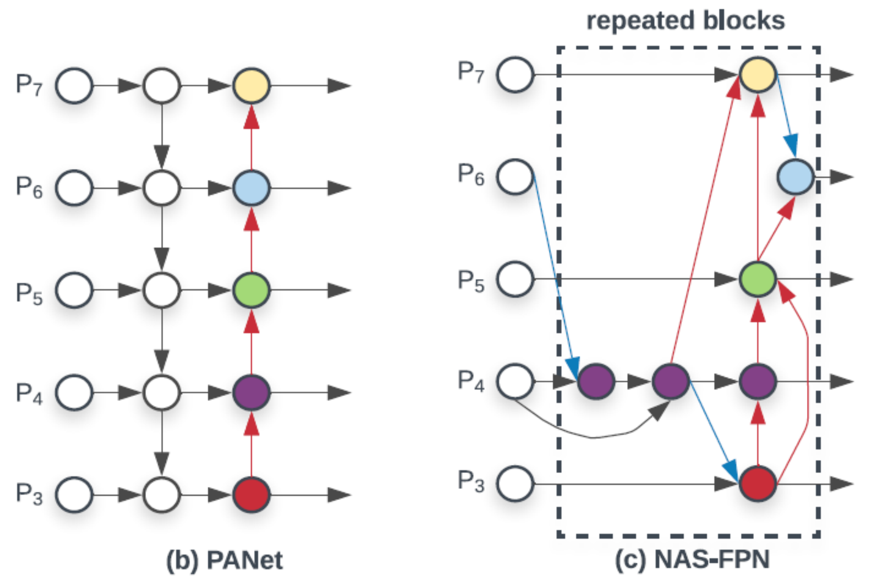
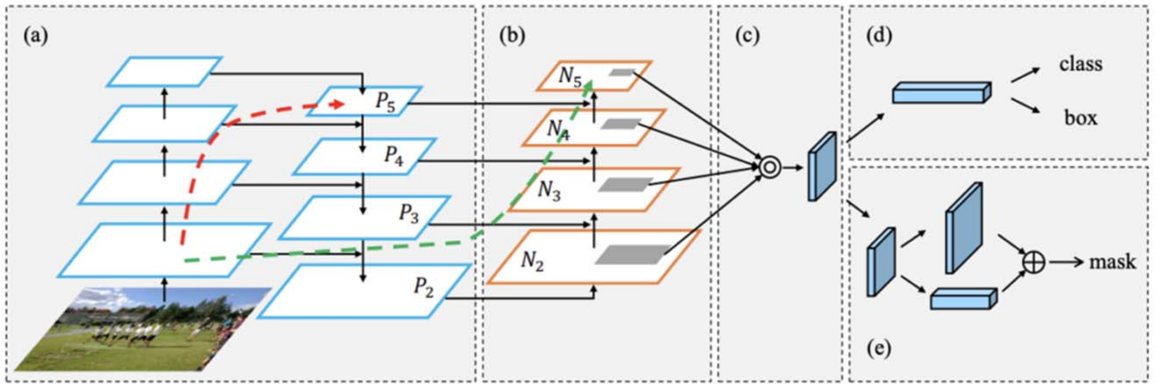


Figure 2. (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) **NAS-FPN** [10] use neural architecture search to find **an irregular feature network topology** and then repeatedly apply the same block

❖ **PANet**

- **Augmenting a top-down path** propagates **semantically strong features and enhances all features with reasonable classification capability** in FPN
- **Augmenting a bottom-up path** of **low-level patterns based on the fact that high response to edges or instance parts is a strong indicator to accurately localize instances**.



Path Aggregation Network for Instance Segmentation, 2018

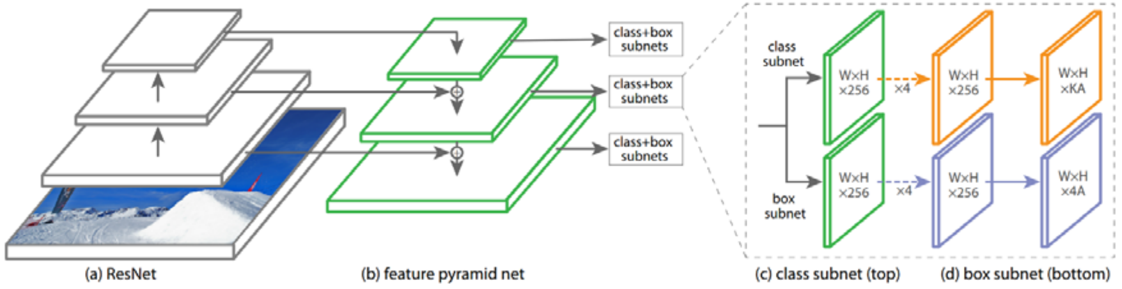
1 EfficientDet

[발췌 자료] [PR-217] EfficientDet : Scalable and Efficient Object Detection, Jin Won Lee

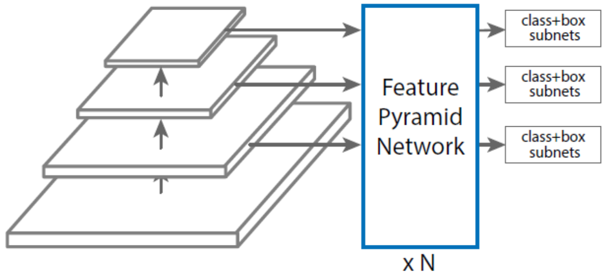
BiFPN : Cross-Scale Connections

❖ NAS-FPN

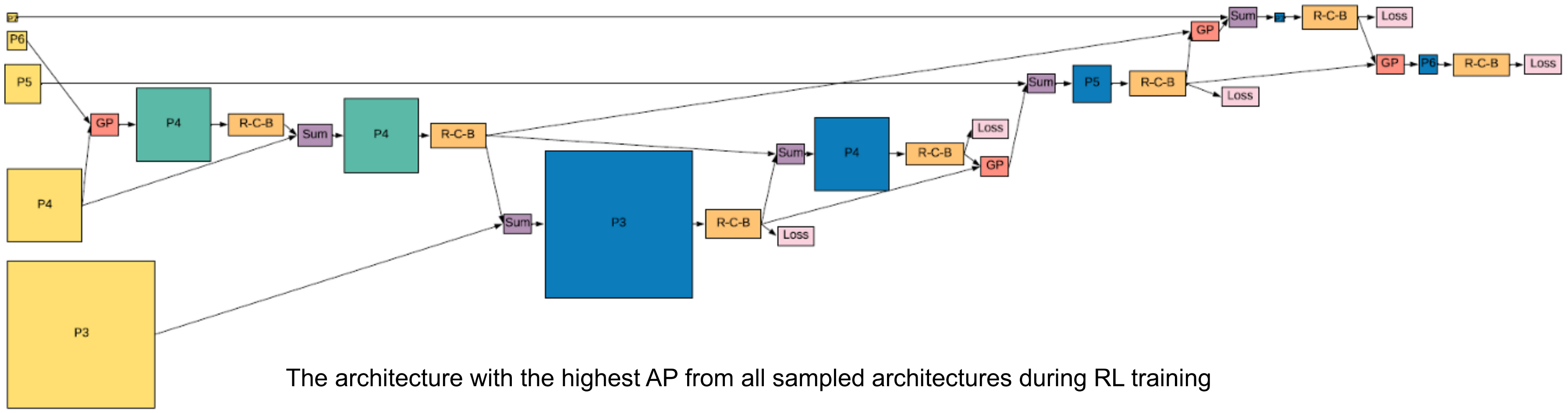
- Adopt Neural Architecture Search and **discover a new feature pyramid architecture in a novel scalable search space covering all cross-scale connections**
- The discovered architecture, named NAS-FPN, **consists of a combinations of top-down and bottom-up connections to fuse features across scales**.



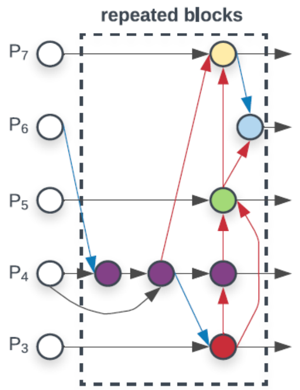
The Original RetinaNet



RetinaNet with NAS-FPN

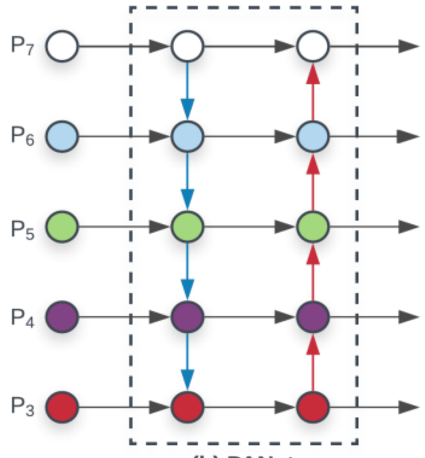


The architecture with the highest AP from all sampled architectures during RL training

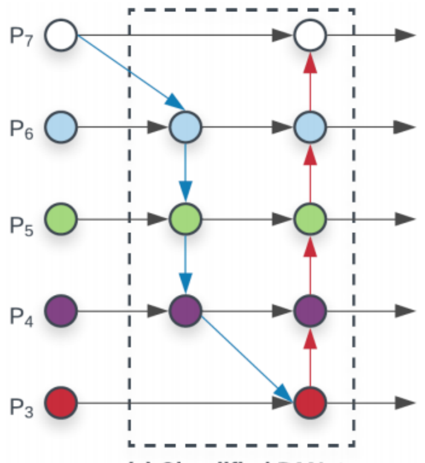


BiFPN : Cross-Scale Connections

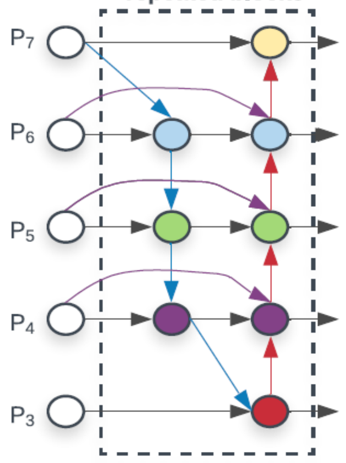
- **PANet achieves better accuracy** than FPN and NAS-FPN, but with the cost of more parameters and computations.
- First, we **remove those nodes that only have one input edge with no feature fusion**, then it will have **less contribution** to feature network that aims at fusing different features.
- 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**.



(b) PANet



(e) Simplified PANet



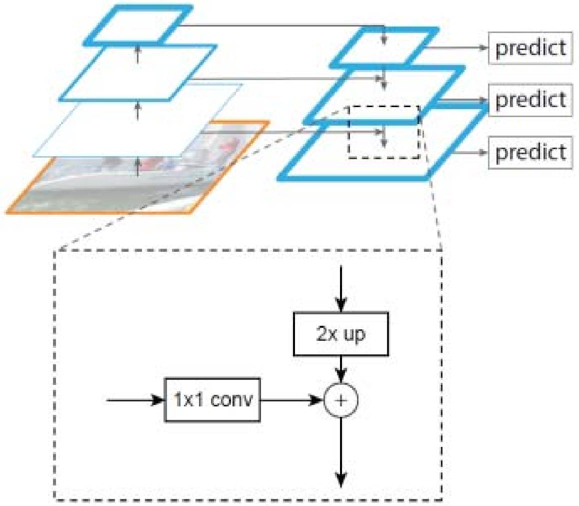
(d) BiFPN

1

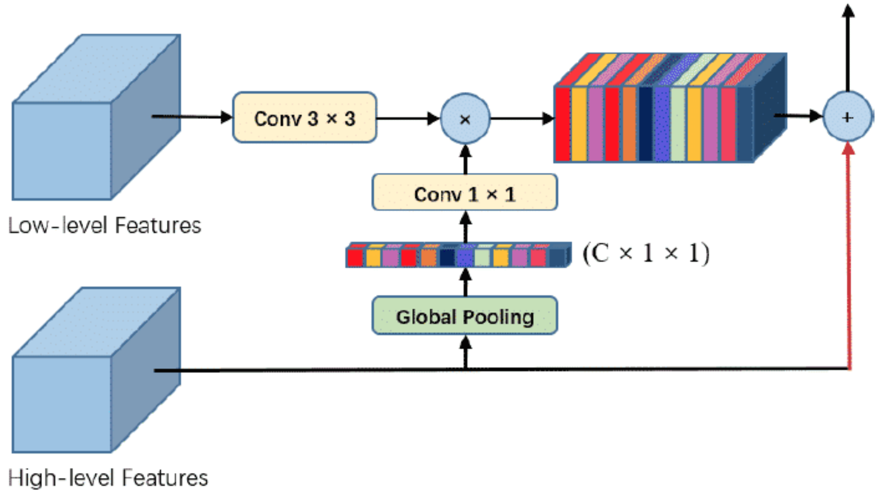
EfficientDet

BiFPN : 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.
- Since different input features are at different resolutions, they usually **contribute to the output feature unequally**.



Global Attention Upsample Module Structure

- 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, Softmax-based fusion, Fast normalized fusion

그림
https://ys-cs17.tistory.com/31

EfficientDet

BiFPN : Weighted Feature Fusion

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, 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 ϵ = 0:0001 is a small value to avoid numerical instability.
- 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

1

EfficientDet

BiFPN

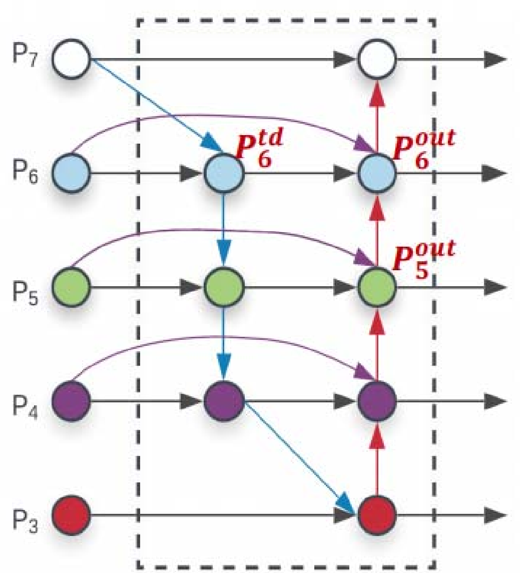
- Our final BiFPN integrates both the bidirectional cross-scale connections and the fast normalized fusion.

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

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

- 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.

- To further improve the efficiency, we use depthwise separable convolution [7, 37] for feature fusion.



(f) BiFPN

EfficientDet Architecture

- 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
- The class and box network weights are shared across all levels of features

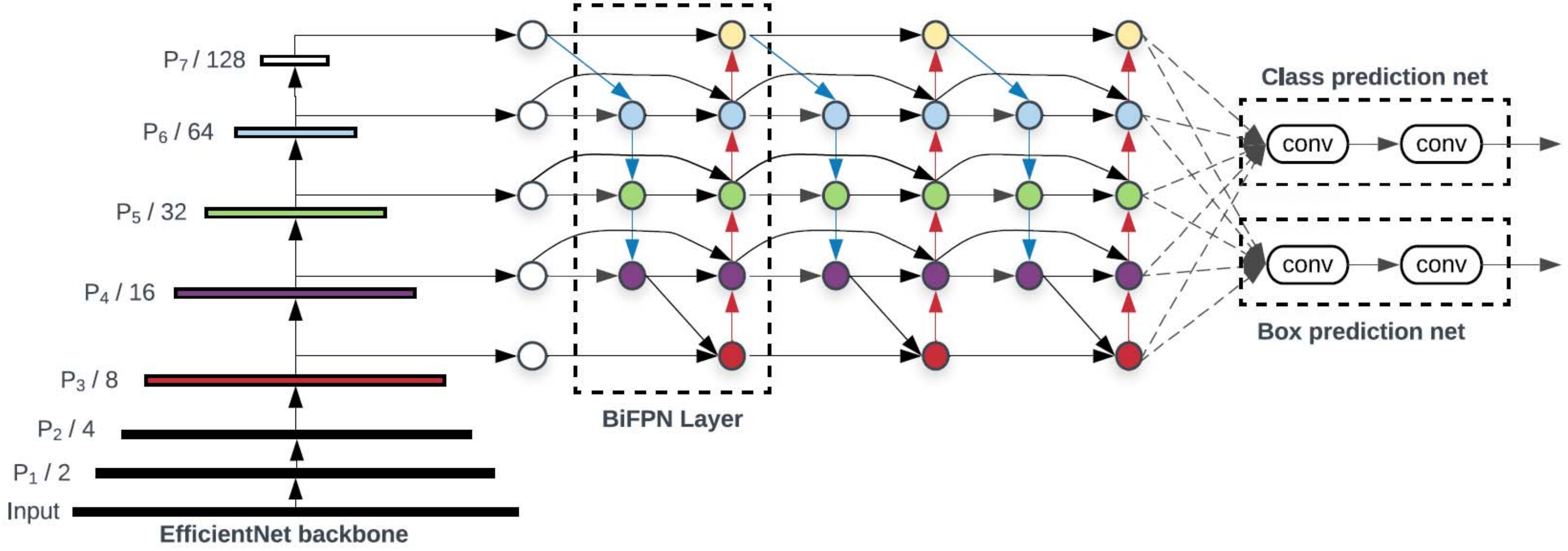


Figure 3: **EfficientDet architecture** – It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

EfficientDet – Compound Scaling

- How to scale up a baseline EfficientDet model.
- Inspired by EfficientNet, 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**
  - ✓ Same width/depth scaling coefficient of EfficientNet B0 to B6.
- **BiFPN network**
  - ✓ Exponentially grow BiFPN width W_{bifpm} (#channels), but linearly increase depth D_{bifpm} (#layers) since depth needs to be rounded to small integers.

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

- **Box/class prediction network**
  - ✓ Fix their width to always the same as BiFPN (i.e., $W_{pred} = W_{bifpm}$)
  - ✓ But, linearly increase the depth (#layers) using equation:

$$D_{box} = D_{class} = 3 + \lfloor \phi / 3 \rfloor$$

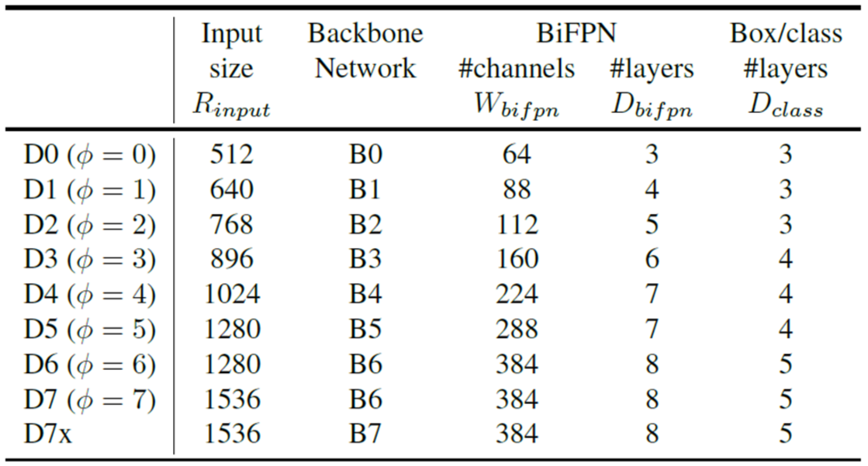
- **Input image resolution**
  - ✓ Since feature level 3-7 are used in BiFPN, the input resolution must be dividable by 2^7=128

$$R_{input} = 512 + \phi \cdot 128$$

EfficientDet – Compound Scaling

- Following Equations 1,2,3 with different ϕ, we have developed EfficientDet-D0 (ϕ=0) to D7 (ϕ=0) 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.

Table 1: **Scaling configs for EfficientDet D0-D6** – ϕ 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.



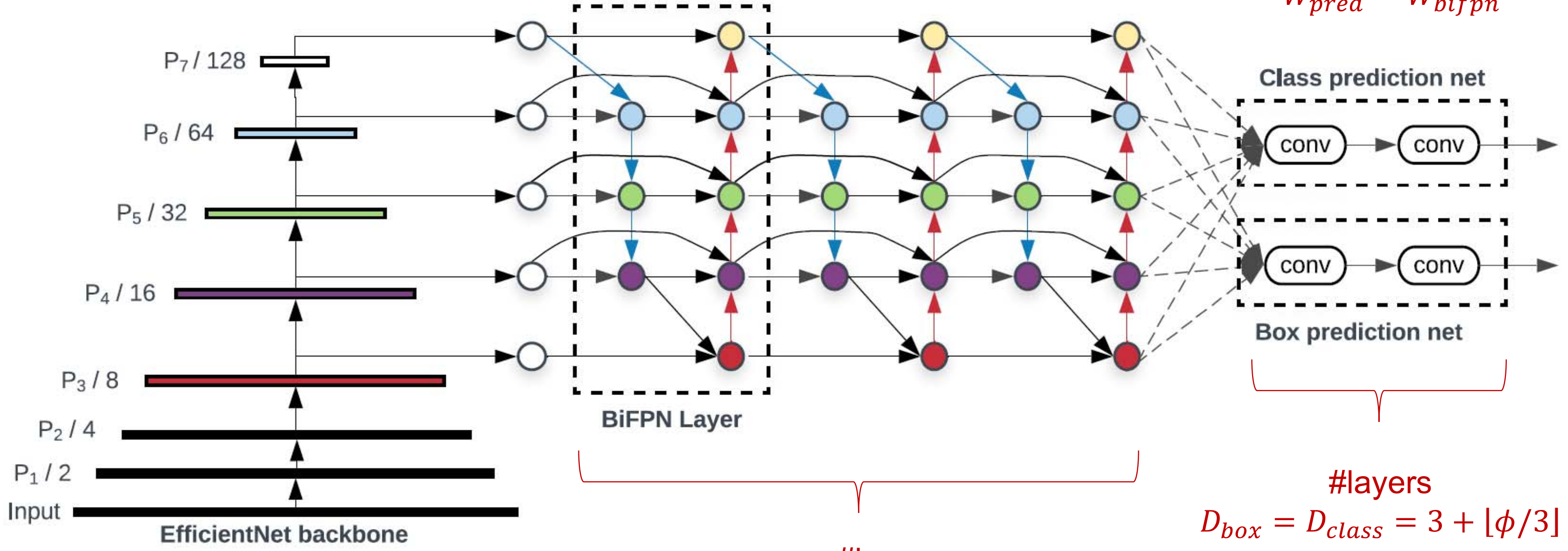
	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4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

EfficientDet – Compound Scaling

Backbone Network
(EfficientNet B0 ~ B6)

#channels
$W_{bifpm} = 64 \cdot (1.35^{\phi})$

#channels
$W_{pred} = W_{bifpn}$

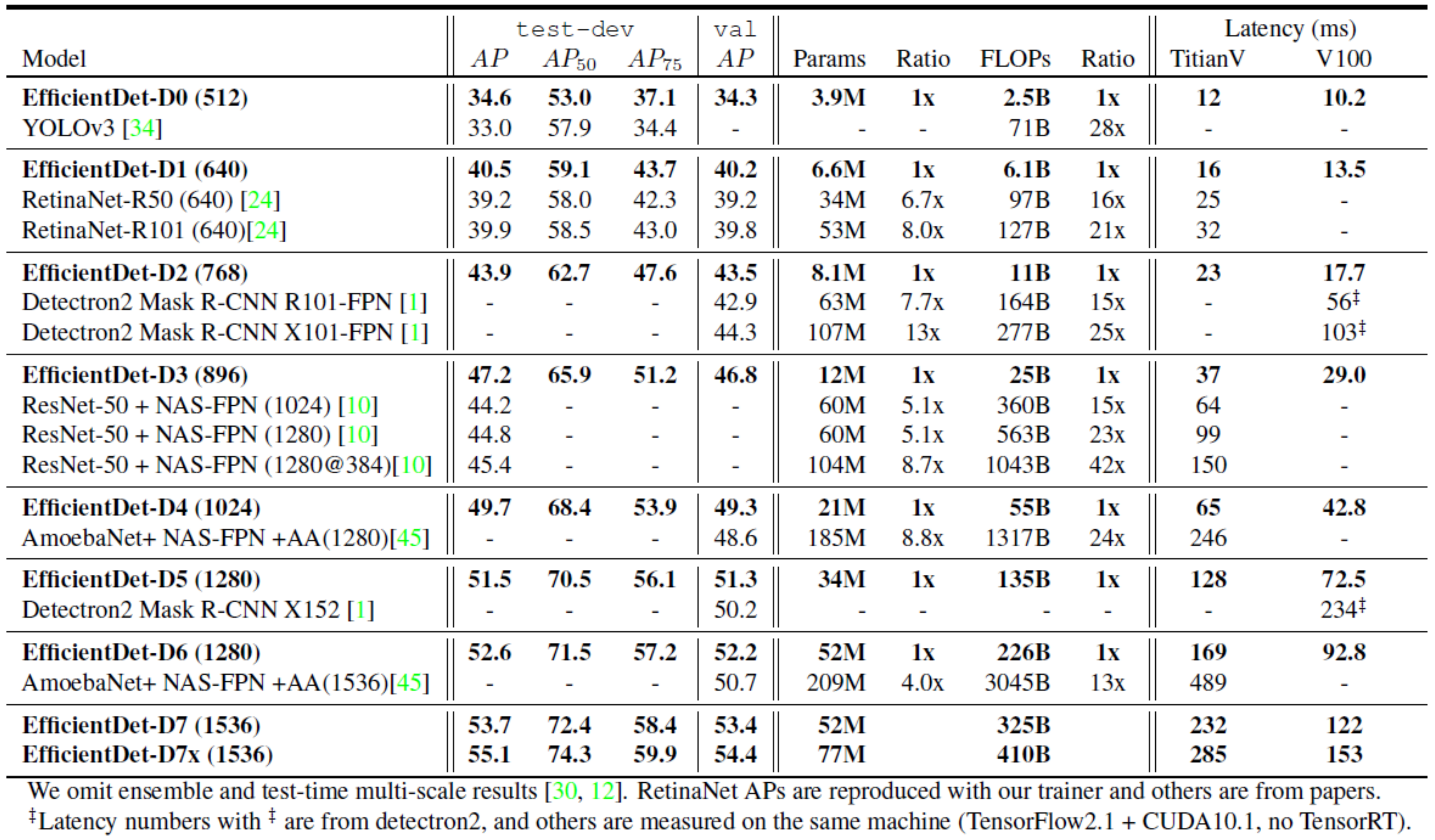


#layers
$D_{box} = D_{class} = 3 + \lfloor \phi / 3 \rfloor$

#layers
$D_{bifpm} = 2 + \phi$

Input resolution
$R_{input} = 512 + \phi \cdot 128$

Experiment



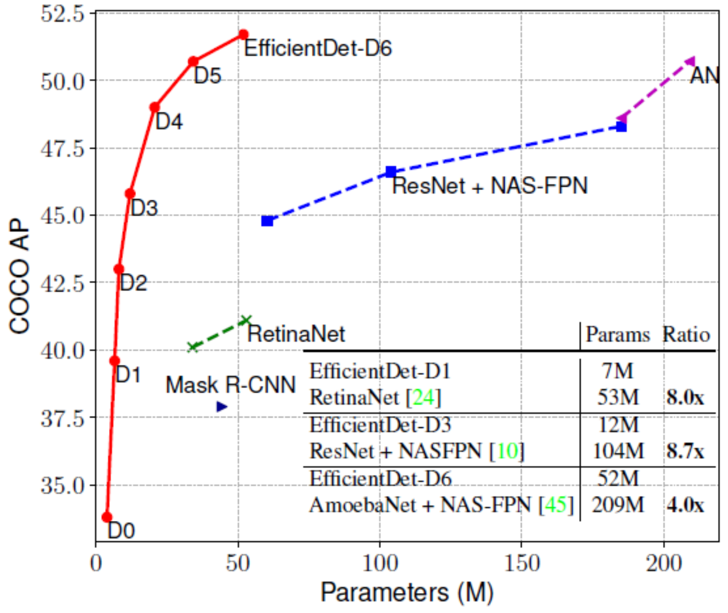
	test-dev			val					Latency (ms)	
Model	AP	AP_{50}	AP_{75}	AP	Params	Ratio	FLOPs	Ratio	TitianV	V100
EfficientDet-D0 (512)	**34.6**	**53.0**	**37.1**	**34.3**	**3.9M**	**1x**	**2.5B**	**1x**	**12**	**10.2**
YOLOv3 [34]	33.0	57.9	34.4	-	-	-	71B	28x	-	-
EfficientDet-D1 (640)	**40.5**	**59.1**	**43.7**	**40.2**	**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.

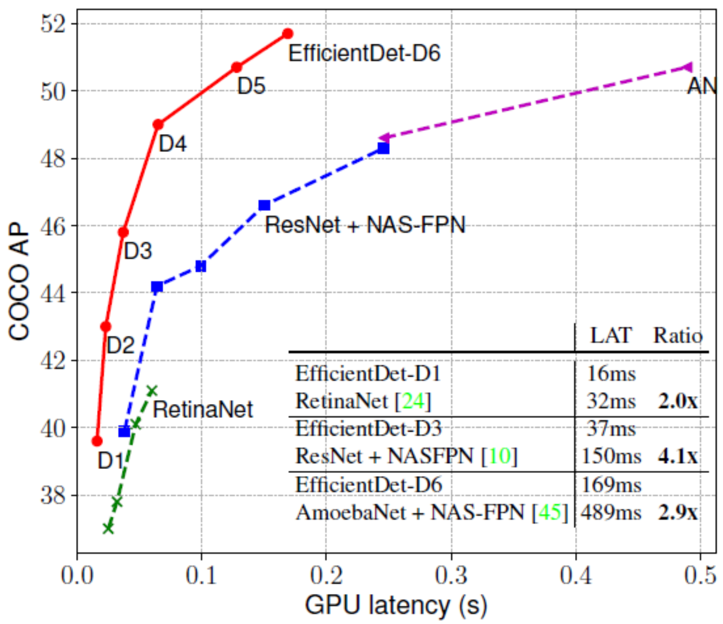
EfficientDet

Experiment



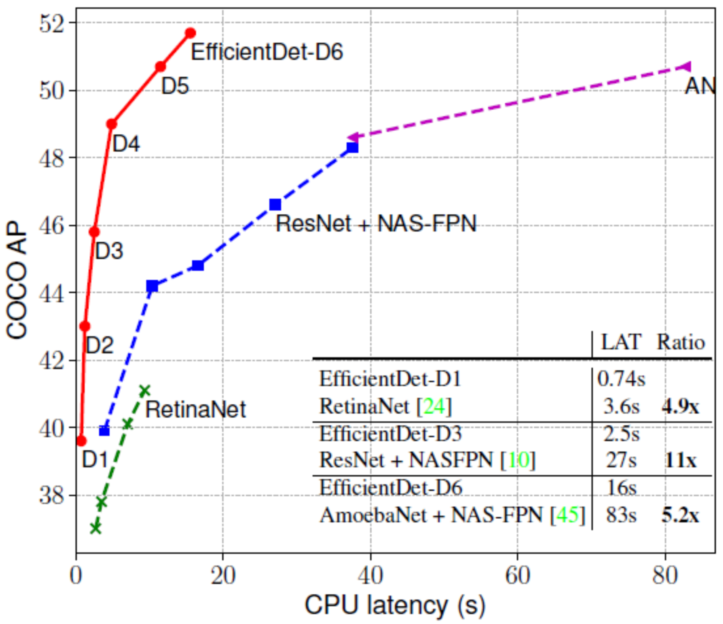
	Params	Ratio
EfficientDet-D1	7M	
RetinaNet [24]	53M	**8.0x**
EfficientDet-D3	12M	
ResNet + NASFPN [10]	104M	**8.7x**
EfficientDet-D6	52M	
AmoebaNet + NAS-FPN [45]	209M	**4.0x**

(a) Model Size



	LAT	Ratio
EfficientDet-D1	16ms	
RetinaNet [24]	32ms	**2.0x**
EfficientDet-D3	37ms	
ResNet + NASFPN [10]	150ms	**4.1x**
EfficientDet-D6	169ms	
AmoebaNet + NAS-FPN [45]	489ms	**2.9x**

(b) GPU Latency



	LAT	Ratio
EfficientDet-D1	0.74s	
RetinaNet [24]	3.6s	**4.9x**
EfficientDet-D3	2.5s	
ResNet + NASFPN [10]	27s	**11x**
EfficientDet-D6	16s	
AmoebaNet + NAS-FPN [45]	83s	**5.2x**

(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.

EfficientDet

Ablation Study

Disentangling Backbone and BiFPN

	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

BiFPN CrossScale Connections

	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

Softmax vs Fast Normalized Fusion

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

EfficientDet

Ablation Study

Compound Scaling

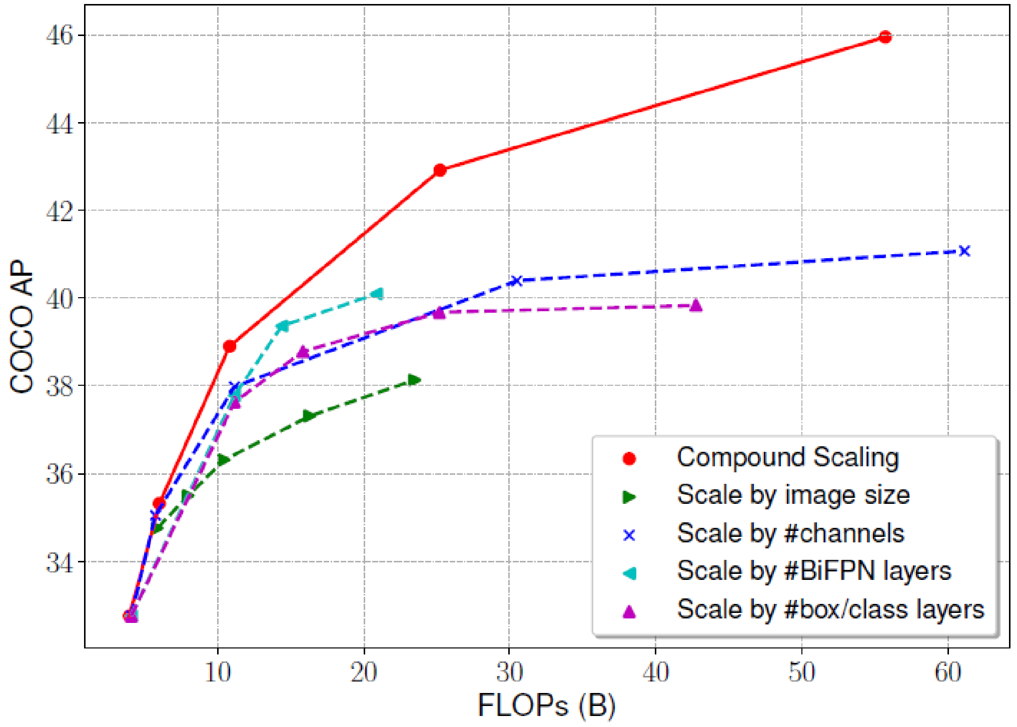


Figure 6: **Comparison of different scaling methods** – compound scaling achieves better accuracy and efficiency.

Conclusions

- Propose **a weighted bidirectional feature network and a customized compound scaling method**, in order to improve **accuracy and efficiency**.
- **EfficientDet-D7** achieves state-of-the-art 51.0 mAP on COCO dataset with 52M parameters and 326B FLOPs, being 4x smaller and using 9.3x fewer FLOPS yet still more accurate (+0.3% mAP) than the best previous detector.
- EfficientDet is also up to **3.2x faster on GPUs and 8.1x faster on GPUs.**