Efficient Det

Tags	Medium Model Scaling
Resources	Good Overview: https://www.youtube.com/watch?v=OsA3zH5NKYc | Breakdown Video: https://www.youtube.com/watch?v=qZobxWXlJ0g |

▼ Article

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjCnJSwkoz6AhWxn2MGHQa-BwYQFnoECDgQAQ&url=https%3A%2F%2Ftowardsdatascience.com%2Fa-thorough-breakdown-of-efficientdet-for-object-detection-dc6a15788b73&usg=AOvVaw0N5pQM5eCRPF2D_8vJmxgR

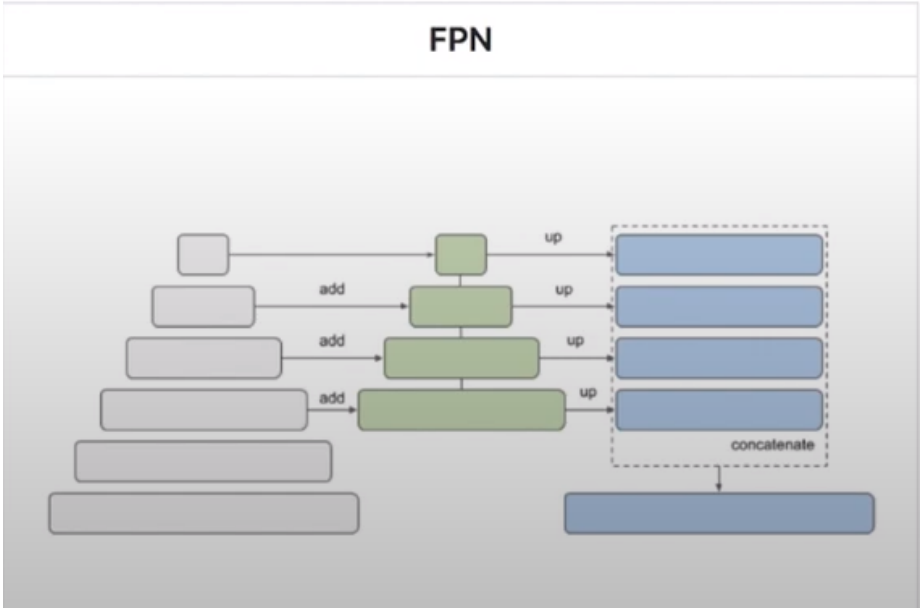
Objective of the Paper

- Performing better Feature Fusion using BiFPN
- How to perform joint scaling (Scaling the Backbone Network, Feature Fusion and Prediction network)
- Tackles the problem performing efficient object detection & Segementation

Things which are addressed in the Paper

Feature Fusion:

- After extracting low level features and eventually extracting the high level features, we don't reform the low level features after again the global context. For example, something which was extracted as circle in the start eventually will be turn out to be an ellipse in the later layers and when we simply use this circle in our feature fusion, it kind of doesn't make sense to adjust that circle to an ellipse
    - We simply use the low level features, without alteration, after gaining the overall view

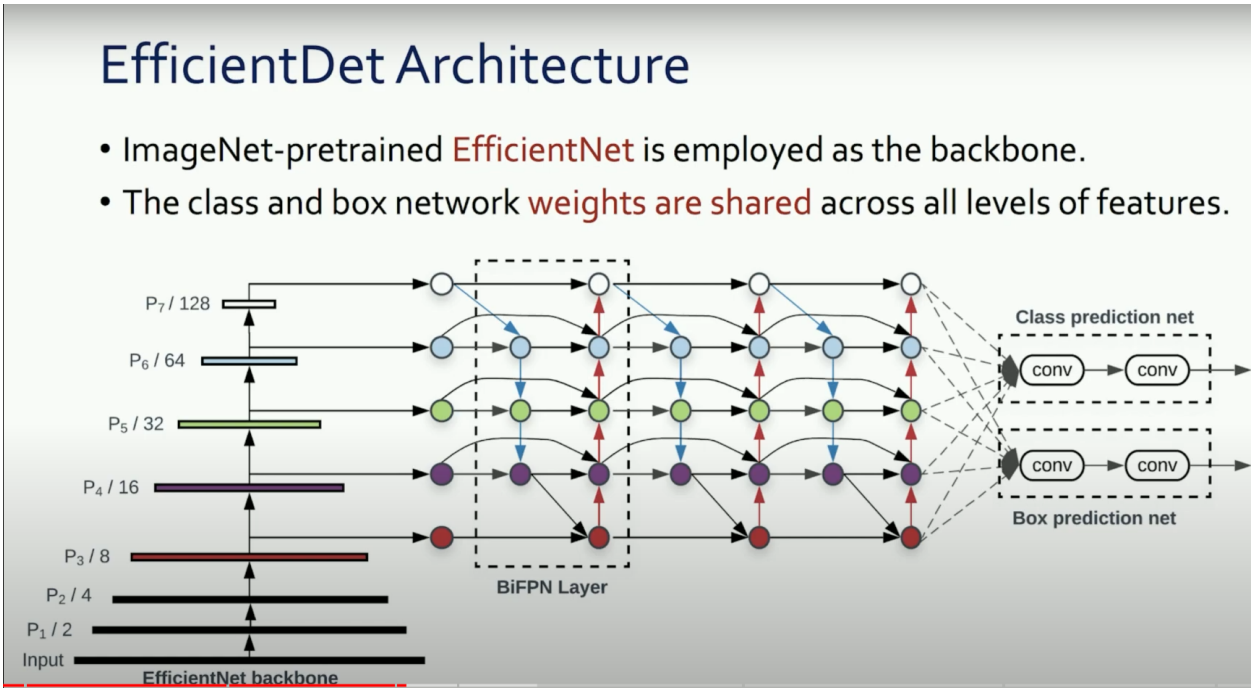


- We need a way to where global and local features interact and change their weights after their interaction

Model Scaling:

- EfficientNet provided ways to scale a model which are depth, width, resolution and compund (all of them).
- Generally, we create a family of models by using these scaling techniques

Architecture



- The arrows to the circle from the backbone is done before max pooling

BiFPN Architecture

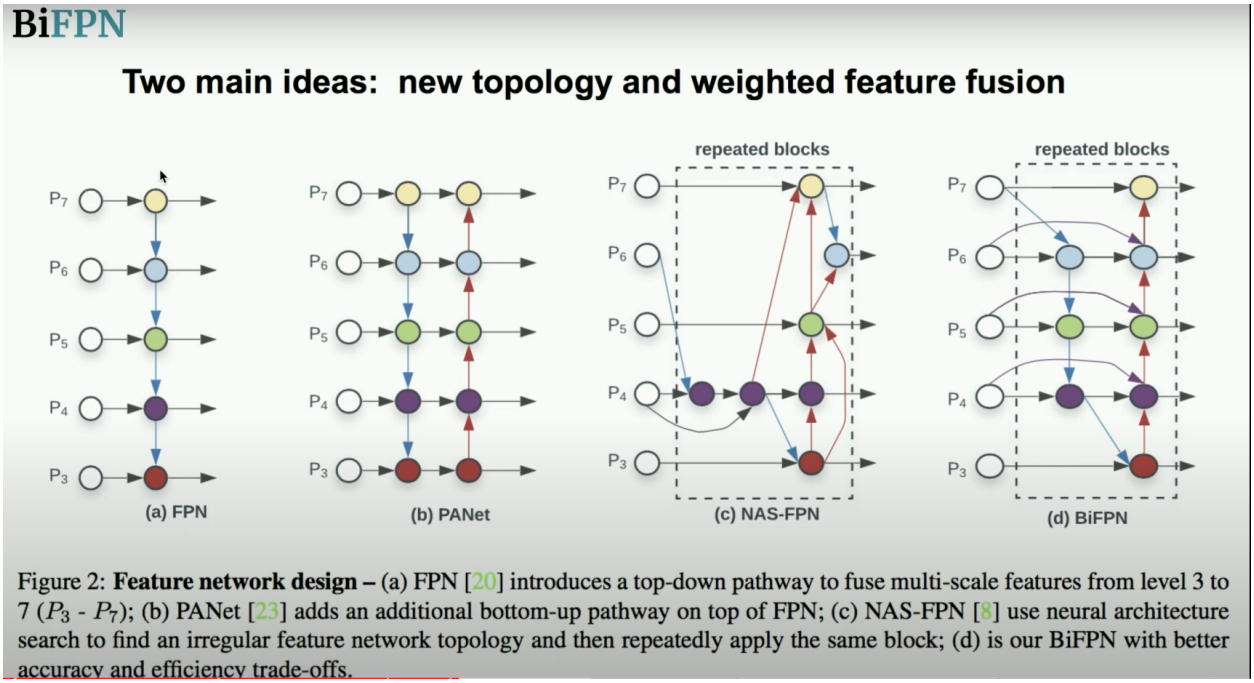
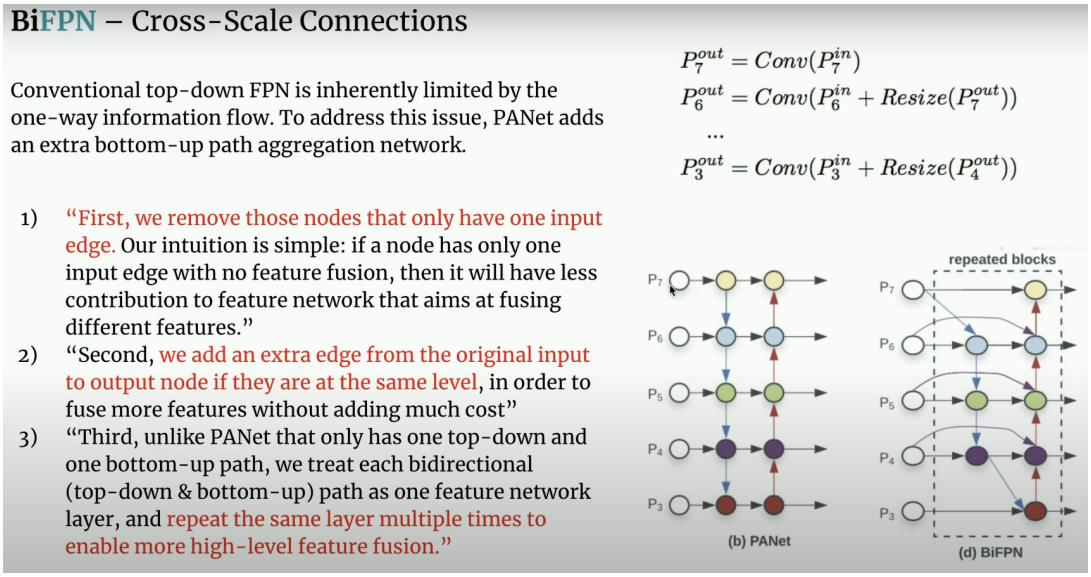


Figure 2: **Feature network design –** (a) FPN [20] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [23] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [8] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

Main difference between PANet and BiFPN

- BiFPN layers are stacked several times
  - So that low level and high level features continue to interact with each other which was not present in the PANet
- Nodes with only one input in PANet is removed in the BiFPN architecture
- BiFPN has Skip connections from the input
  - The intution behind having skip connections is that if a good feature is present at say P6, they can propagate easily through the network and influence the prediction well and these middle block (circles here) could learn new features which would otherwise have to retain information from the P6 if there weren't any skip layers.
  - They also reduce parameters which makes them more efficient

▼ Details



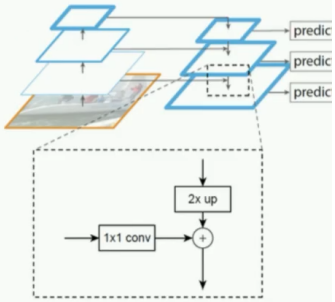
Weight Fusion

- In the above diagram, take P6 for instance, it is contributing to a lot of places like creating P7 and in several places in the BiFPN network. So, this P6 might not as essential as it's use, so the author uses wegiths which can learnt so that the date decides which feature is essential and how much.

▼ Details:

BiFPN – Weighted Feature Fusion

- When fusing multiple input features with different resolutions, a common way is to first resize them to the same resolution and then sum them up.
- Pyramid attention network 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. So, adding an additional weight for each input during featur fusion, making the network to learn the importance of each input feature.



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

▼ How these weights are normalised - TL;DR - Fast Normalised Fusion

BiFPN – Weighted Feature Fusion

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

Unbounded Fusion

Scalar weight being unbounded could potentially cause training instability

$$O = \sum_i w_i \cdot I_i$$

Softmax-Based Fusion

Softmax leads to significant slowdown on GPU hardware.

$$O = \sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot I_i$$

Fast Normalized Fusion

$w\_i \geq 0$, 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 .

$$O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$$

- How are they used finally

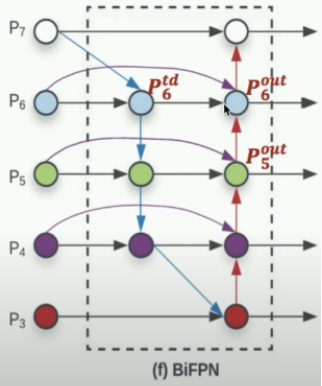
- BiFPN integrates both the bidirectional cross-scale connections and the fast normalized fusion.

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

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

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.

- Depthwise separable convolution is used for feature fusion.



(f) BiFPN

Compond Scaling

- Grid Search for Object Detection is computationally heavier than Grid Search for Classification
- So, the author's come with an heuristic value for scaling the entire model

Part2: Compound Scaling

Main idea: use a single ϕ to governor all width, depth, and resolution scaling

BiFPN width&depth

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

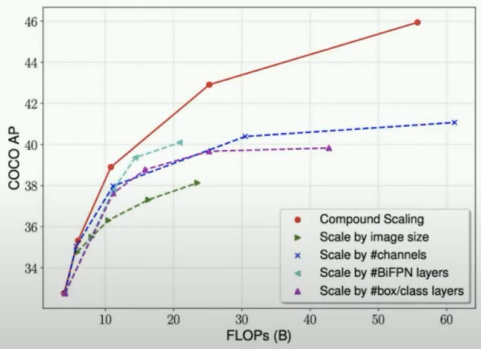
box/class prediciton network depth

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

Resolution

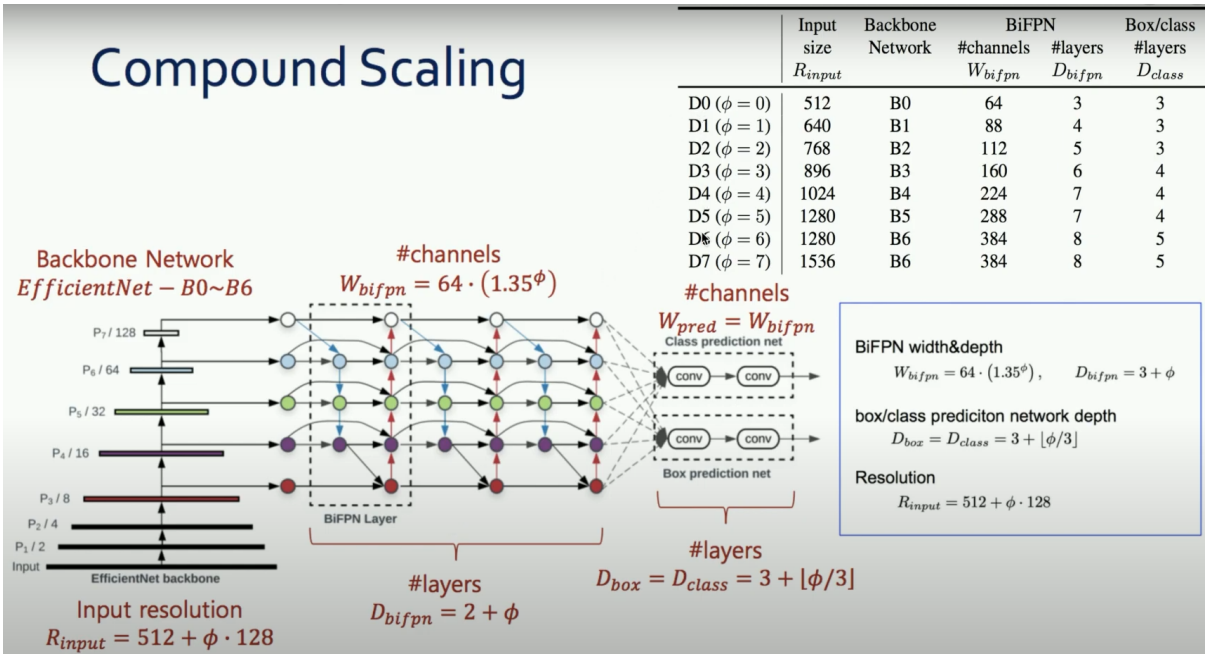
$$R_{input} = 512 + \phi \cdot 128$$

Heuristic-based scaling rules



Compound scaling vs single-scaling

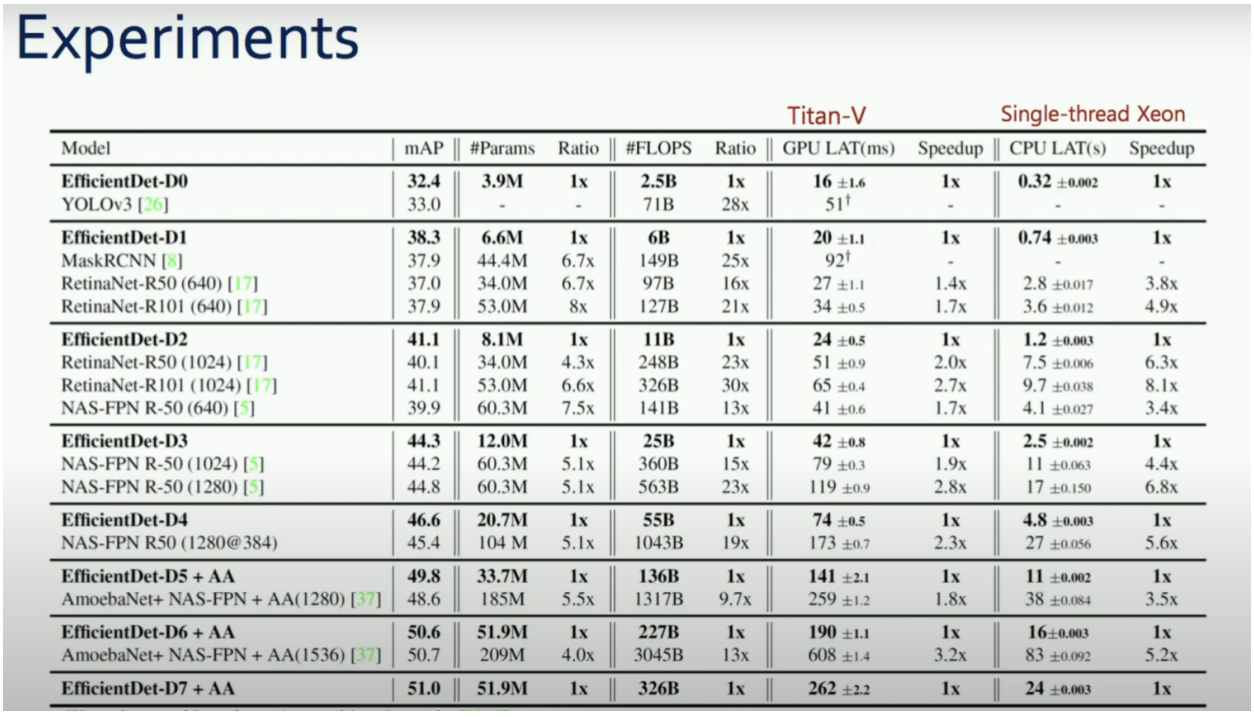
- Example



	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

Results

This below diagram shows how efficient the network in terms of params,speed



						Titan-V		Single-thread Xeon	
Model	mAP	#Params	Ratio	#FLOPS	Ratio	GPU LAT(ms)	Speedup	CPU LAT(s)	Speedup
EfficientDet-D0	**32.4**	**3.9M**	**1x**	**2.5B**	**1x**	**16** ±1.6	**1x**	**0.32** ±0.002	**1x**
YOLOv3 [26]	33.0	-	-	71B	28x	51†	-	-	-
EfficientDet-D1	**38.3**	**6.6M**	**1x**	**6B**	**1x**	**20** ±1.1	**1x**	**0.74** ±0.003	**1x**
MaskRCNN [8]	37.9	44.4M	6.7x	149B	25x	92†	-	-	-
RetinaNet-R50 (640) [17]	37.0	34.0M	6.7x	97B	16x	27 ±1.1	1.4x	2.8 ±0.017	3.8x
RetinaNet-R101 (640) [17]	37.9	53.0M	8x	127B	21x	34 ±0.5	1.7x	3.6 ±0.012	4.9x
EfficientDet-D2	**41.1**	**8.1M**	**1x**	**11B**	**1x**	**24** ±0.5	**1x**	**1.2** ±0.003	**1x**
RetinaNet-R50 (1024) [17]	40.1	34.0M	4.3x	248B	23x	51 ±0.9	2.0x	7.5 ±0.006	6.3x
RetinaNet-R101 (1024) [17]	41.1	53.0M	6.6x	326B	30x	65 ±0.4	2.7x	9.7 ±0.038	8.1x
NAS-FPN R-50 (640) [5]	39.9	60.3M	7.5x	141B	13x	41 ±0.6	1.7x	4.1 ±0.027	3.4x
EfficientDet-D3	**44.3**	**12.0M**	**1x**	**25B**	**1x**	**42** ±0.8	**1x**	**2.5** ±0.002	**1x**
NAS-FPN R-50 (1024) [5]	44.2	60.3M	5.1x	360B	15x	79 ±0.3	1.9x	11 ±0.063	4.4x
NAS-FPN R-50 (1280) [5]	44.8	60.3M	5.1x	563B	23x	119 ±0.9	2.8x	17 ±0.150	6.8x
EfficientDet-D4	**46.6**	**20.7M**	**1x**	**55B**	**1x**	**74** ±0.5	**1x**	**4.8** ±0.003	**1x**
NAS-FPN R50 (1280@384)	45.4	104 M	5.1x	1043B	19x	173 ±0.7	2.3x	27 ±0.056	5.6x
EfficientDet-D5 + AA	**49.8**	**33.7M**	**1x**	**136B**	**1x**	**141** ±2.1	**1x**	**11** ±0.002	**1x**
AmoebaNet+ NAS-FPN + AA(1280) [37]	48.6	185M	5.5x	1317B	9.7x	259 ±1.2	1.8x	38 ±0.084	3.5x
EfficientDet-D6 + AA	**50.6**	**51.9M**	**1x**	**227B**	**1x**	**190** ±1.1	**1x**	**16**±0.003	**1x**
AmoebaNet+ NAS-FPN + AA(1536) [37]	50.7	209M	4.0x	3045B	13x	608 ±1.4	3.2x	83 ±0.092	5.2x
EfficientDet-D7 + AA	**51.0**	**51.9M**	**1x**	**326B**	**1x**	**262** ±2.2	**1x**	**24** ±0.003	**1x**