Paper review

# EfficientDet: Scalable and Efficient Object Detection

Google Brain (CVPR 2020)

01

# Quick View

# CONTRIBUTES

- **Better Efficiency**
- **Higher Accuracy**

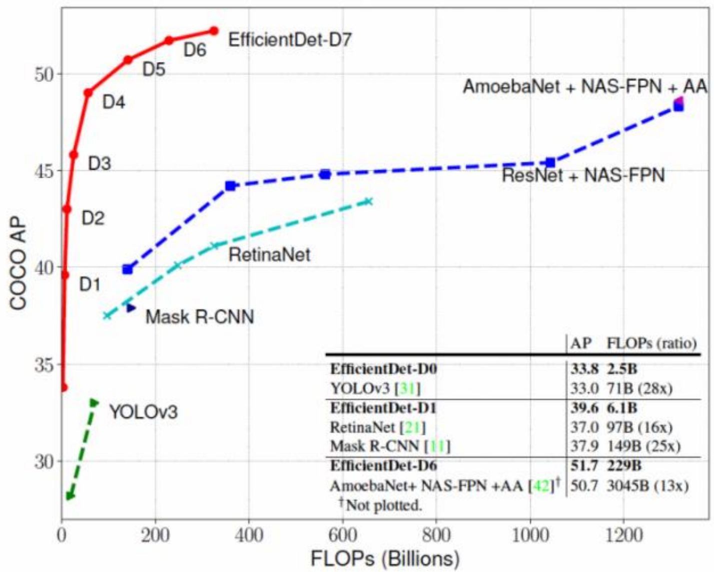



Figure 1: **Model FLOPs vs. COCO accuracy –** All numbers are for single-model single-scale. Our EfficientDet achieves new state-of-the-art 52.2% COCO AP with much fewer parameters and FLOPs than previous detectors. More studies on different backbones and FPN/NAS-FPN/BiFPN are in Table 4 and 5. Complete results are in Table 2.

Image Classification

## EfficientNet

1) Model Scaling(Depth, Width, Resolution)
2) Compound scaling method

Challenge

## 2 Challenges

1) BiFPN (weighted bi-directional feature pyramid network):
   easy and fast multi-scale feature fusion 가능
1) Compund Scaling method
   backbone, feature network & box/class prediction Networks에 대해 동시에 해상도, 깊이 및 폭을 균일하게 스케일링하는 방법

https://arxiv.org/pdf/1905.11946.pdf

02

# Prior Knowledge

# EfficientNet

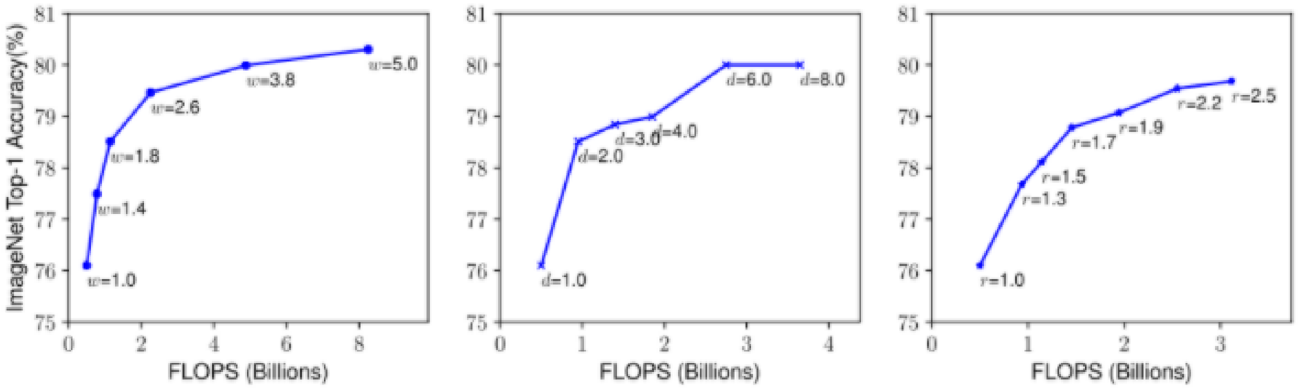



*Figure 3.* **Scaling Up a Baseline Model with Different Network Width ($w$), Depth ($d$), and Resolution ($r$) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

https://arxiv.org/pdf/1905.11946.pdf

# EfficientNet

Model Scaling

## Compound Scaling

고해상도의 이미지의 경우,

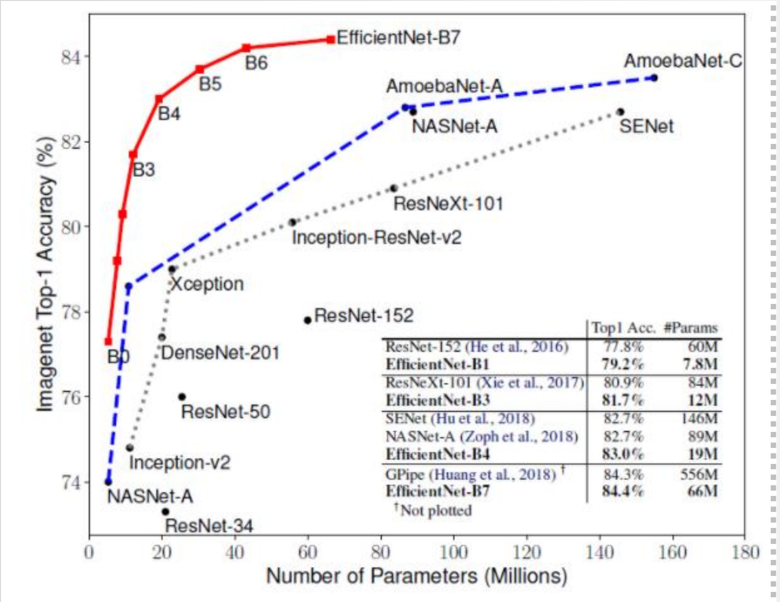
Network의 **depth**을 높여 큰 receptive field를 확보하는 것이 중요

Network의 **width**을 높여 더욱 정제된 feature들을 얻는 것이 중요

이와 같은 이유때문에 저자는 depth, width, resolution coefficient 중 하나의 coefficient만 조정하는 것이 아닌, **scaling의 균형을 맞추는 조정**이 필요함

https://arxiv.org/pdf/1905.11946.pdf

# EfficientNet




| Model | Top-1 Acc. | Top-5 Acc. | #Params | Ratio-to-EfficientNet | #FLOPS | Ratio-to-EfficientNet |
|---|---|---|---|---|---|---|
| **EfficientNet-B0** | **77.3%** | **93.5%** | **5.3M** | **1x** | **0.39B** | **1x** |
| ResNet-50 (He et al., 2016) | 76.0% | 93.0% | 26M | 4.9x | 4.1B | 11x |
| DenseNet-169 (Huang et al., 2017) | 76.2% | 93.2% | 14M | 2.6x | 3.5B | 8.9x |
| **EfficientNet-B1** | **79.2%** | **94.5%** | **7.8M** | **1x** | **0.70B** | **1x** |
| ResNet-152 (He et al., 2016) | 77.8% | 93.8% | 60M | 7.6x | 11B | 16x |
| DenseNet-264 (Huang et al., 2017) | 77.9% | 93.9% | 34M | 4.3x | 6.0B | 8.6x |
| Inception-v3 (Szegedy et al., 2016) | 78.8% | 94.4% | 24M | 3.0x | 5.7B | 8.1x |
| Xception (Chollet, 2017) | 79.0% | 94.5% | 23M | 3.0x | 8.4B | 12x |
| **EfficientNet-B2** | **80.3%** | **95.0%** | **9.2M** | **1x** | **1.0B** | **1x** |
| Inception-v4 (Szegedy et al., 2017) | 80.0% | 95.0% | 48M | 5.2x | 13B | 13x |
| Inception-resnet-v2 (Szegedy et al., 2017) | 80.1% | 95.1% | 56M | 6.1x | 13B | 13x |
| **EfficientNet-B3** | **81.7%** | **95.6%** | **12M** | **1x** | **1.8B** | **1x** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 95.6% | 84M | 7.0x | 32B | 18x |
| PolyNet (Zhang et al., 2017) | 81.3% | 95.8% | 92M | 7.7x | 35B | 19x |
| **EfficientNet-B4** | **83.0%** | **96.3%** | **19M** | **1x** | **4.2B** | **1x** |
| SENet (Hu et al., 2018) | 82.7% | 96.2% | 146M | 7.7x | 42B | 10x |
| NASNet-A (Zoph et al., 2018) | 82.7% | 96.2% | 89M | 4.7x | 24B | 5.7x |
| AmoebaNet-A (Real et al., 2019) | 82.8% | 96.1% | 87M | 4.6x | 23B | 5.5x |
| PNASNet (Liu et al., 2018) | 82.9% | 96.2% | 86M | 4.5x | 23B | 6.0x |
| **EfficientNet-B5** | **83.7%** | **96.7%** | **30M** | **1x** | **9.9B** | **1x** |
| AmoebaNet-C (Cubuk et al., 2019) | 83.5% | 96.5% | 155M | 5.2x | 41B | 4.1x |
| **EfficientNet-B6** | **84.2%** | **96.8%** | **43M** | **1x** | **19B** | **1x** |
| **EfficientNet-B7** | **84.4%** | **97.1%** | **66M** | **1x** | **37B** | **1x** |
| GPipe (Huang et al., 2018) | 84.3% | 97.0% | 557M | 8.4x | - | - |

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

https://arxiv.org/pdf/1905.11946.pdf

03

# The EfficientDet model

Q Is it possible to build a scalable detection architecture with both **higher accuracy** and **better efficiency** across a wide spectrum of resource constraints?

# BiFPN

## 1. Problem Fomulation

$$\overrightarrow{P^{in}} = (P_{l1}^{in}, P_{l2}^{in}, \ldots, )$$

$$\overrightarrow{P^{out}} = f\left(\overrightarrow{P^{in}}\right)$$

# 2. Cross Scale Connections

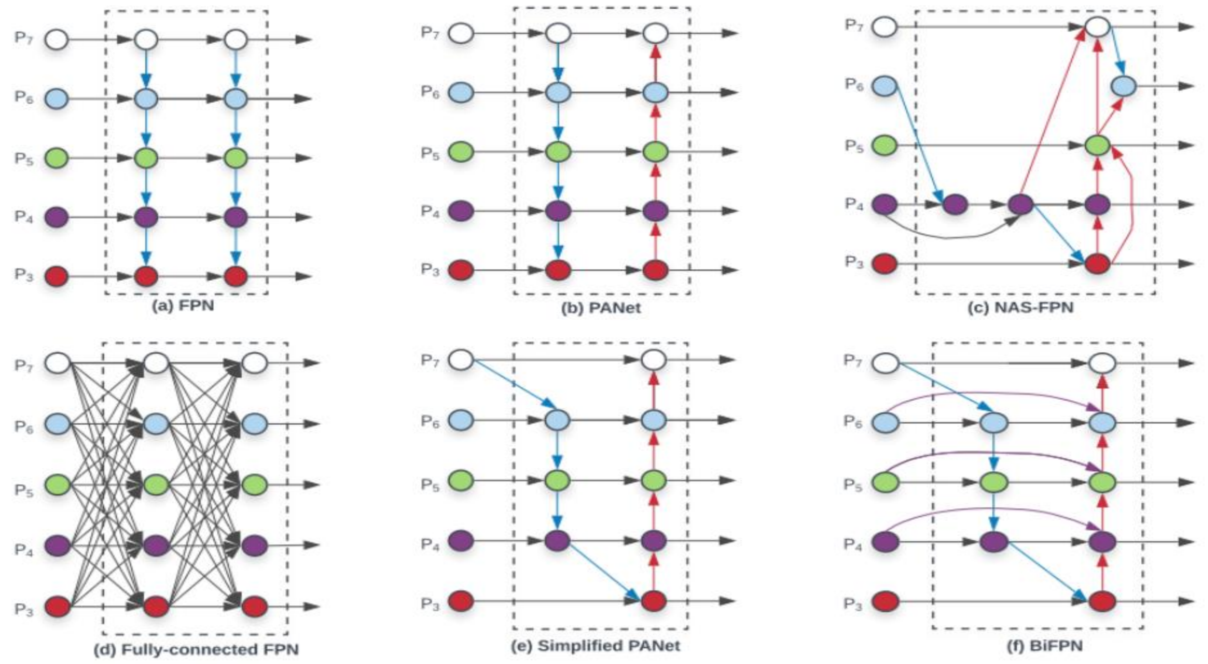



Figure 2: **Feature network design –** (a) FPN [16] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 ($P_3$ - $P_7$); (b) PANet [19] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [5] use neural architecture search to find an irregular feature network topology; (d)-(f) are three alternatives studied in this paper. (d) adds expensive connections from all input feature to output features; (e) simplifies PANet by removing nodes if they only have one input edge; (f) is our BiFPN with better accuracy and efficiency trade-offs.

# Ablation Study

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

Table 3: **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.

| | mAP | #Params ratio | #FLOPS ratio |
|---|---|---|---|
| Top-Down FPN [16] | 42.29 | 1.0x | 1.0x |
| Repeated PANet [19] | 44.08 | 1.0x | 1.0x |
| NAS-FPN [5] | 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 4: **Comparison of different feature networks** – Our weighted BiFPN achieves the best accuracy with fewer parameters and FLOPS.

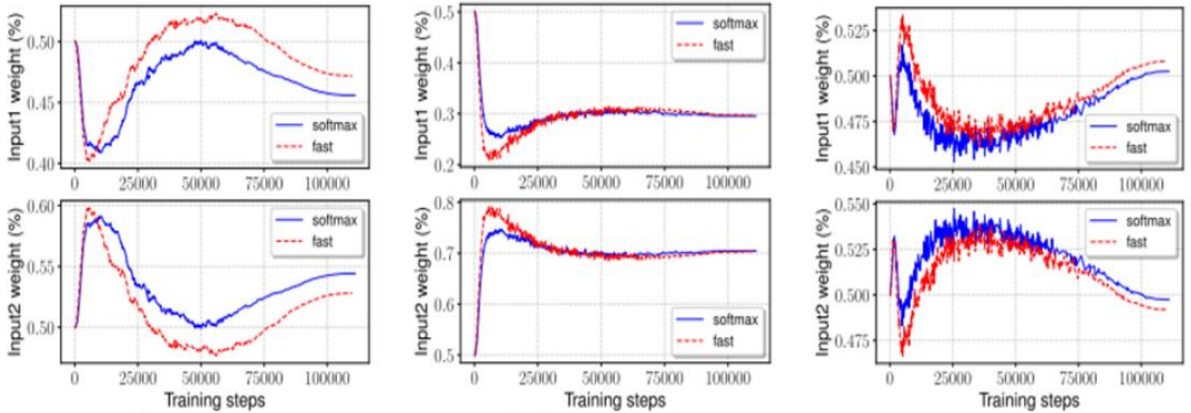
# BiFPN

## 3. Weighted Feature Fusion

1) **Unbounded Fusion**: unbounded 되어있기 때문에 학습에 불안정성 유발
2) **SoftMax-based Fusion**: GPU 하드웨어에서 slowdown을 유발
3) **Fast normalized Fusion**: weight들을 ReLU를 거치기 때문에 non-zero
   분모가 0이 되는 것을 막기 위해 입실론 추가
   weigh값이 0~1사이로 normalize가 되는 것은 Softmax와 유사하다

| | | | |
|---|---|---|---|
| $O = \sum_i I_i.$ | $O = \sum_i w_i \cdot I_i,$ | $O = \sum_i \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot I_i.$ | $O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$ |
| **Conventional Feature Fusion** | **Unbounded Feature Fusion** | **SoftMax-based Feature Fusion** | **Fast normalized Feature Fusion** |

# Ablation Study




(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.

| 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

## 1. EfficientDet Architecture

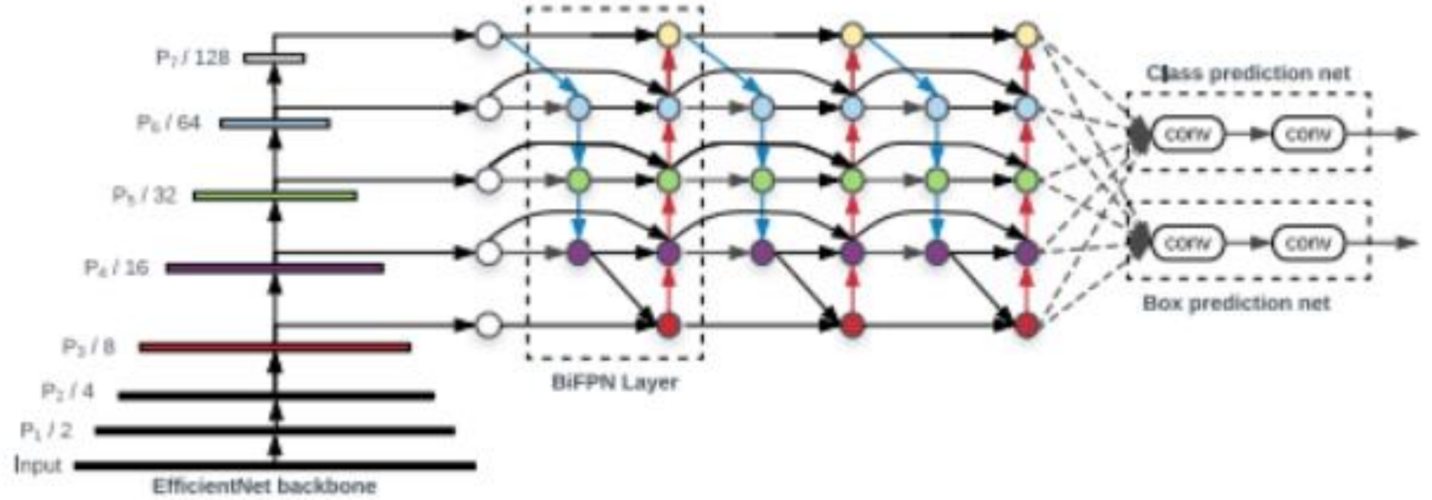



Figure 3: **EfficientDet architecture** – It employs EfficientNet [36] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

# EfficientDet

## 2. Compound Scaling Method

1) **Backbone Network**: EfficientNet-B0부터 B6까지 확대

2) **BiFPN Network**: BiFPN의 width와 depth를 아래와 같이 확장 $W_{bifpn} = 64 \cdot \left(1.35^{\phi}\right), \qquad D_{bifpn} = 3 + \phi$

3) **Class/box Network**: width는 항상 BiFPN과 동일한 값으로 지정 $D_{box} = D_{class} = 3 + \lfloor \phi/3 \rfloor$

4) **Resolution**: Resolution은 아래와 같이 확장 $R_{input} = 512 + \phi \cdot 128$

# EfficientDet

## 2. Compound Scaling Method

| | 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 |
| D6 ($\phi = 7$) | 1536 | B6 | 384 | 8 | 5 |

Table 1: **Scaling configs for EfficientDet D0-D6 –** $\phi$ 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.

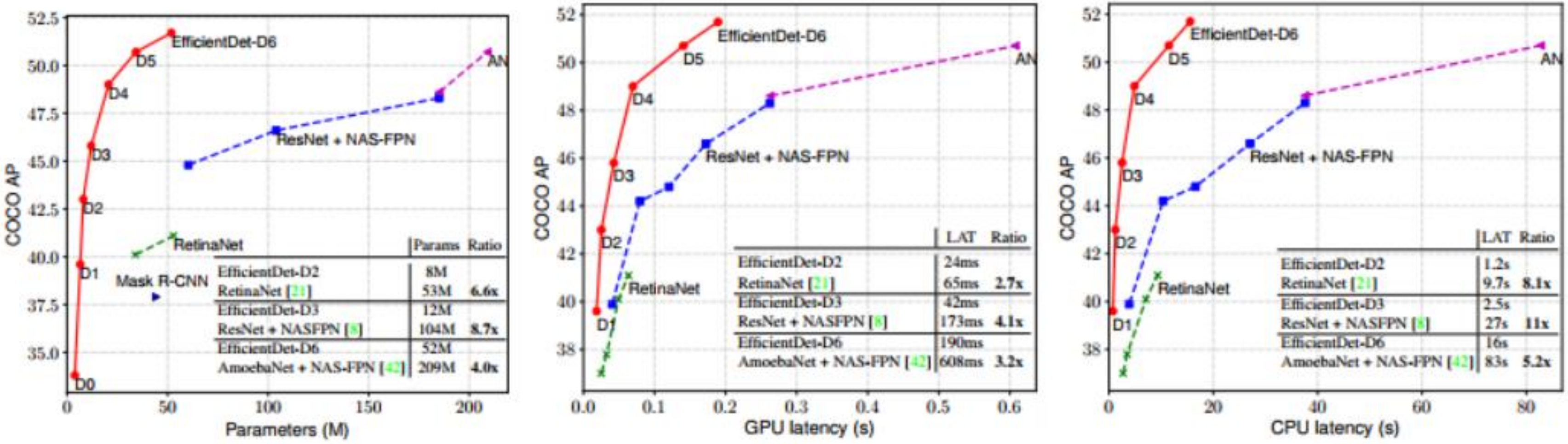
04

# Experiments

| Model | test-dev $AP$ | test-dev $AP_{50}$ | test-dev $AP_{75}$ | val $AP$ | Params | Ratio | FLOPs | Ratio | Latency GPU$_{ms}$ | Latency CPU$_{s}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| **EfficientDet-D0 (512)** | **33.8** | **52.2** | **35.8** | **33.5** | **3.9M** | **1x** | **2.5B** | **1x** | **16** | **0.32** |
| YOLOv3 [31] | 33.0 | 57.9 | 34.4 | - | - | - | 71B | 28x | 51† | - |
| **EfficientDet-D1 (640)** | **39.6** | **58.6** | **42.3** | **39.1** | **6.6M** | **1x** | **6.1B** | **1x** | **20** | **0.74** |
| RetinaNet-R50 (640) [21] | 37.0 | - | - | - | 34M | 6.7x | 97B | 16x | 27 | 2.8 |
| RetinaNet-R101 (640)[21] | 37.9 | - | - | - | 53M | 8.0x | 127B | 21x | 34 | 3.6 |
| Mask R-CNN [11] | 37.9 | - | - | - | 44M | 6.7x | 149B | 25x | 92† | - |
| **EfficientDet-D2 (768)** | **43.0** | **62.3** | **46.2** | **42.5** | **8.1M** | **1x** | **11B** | **1x** | **24** | **1.2** |
| RetinaNet-R50 (1024) [21] | 40.1 | - | - | - | 34M | 4.3x | 248B | 23x | 51 | 7.5 |
| RetinaNet-R101 (1024) [21] | 41.1 | - | - | - | 53M | 6.6x | 326B | 30x | 65 | 9.7 |
| ResNet-50 + NAS-FPN (640) [8] | 39.9 | - | - | - | 60M | 7.5x | 141B | 13x | 41 | 4.1 |
| **EfficientDet-D3 (896)** | **45.8** | **65.0** | **49.3** | **45.9** | **12M** | **1x** | **25B** | **1x** | **42** | **2.5** |
| ResNet-50 + NAS-FPN (1024) [8] | 44.2 | - | - | - | 60M | 5.1x | 360B | 15x | 79 | 11 |
| ResNet-50 + NAS-FPN (1280) [8] | 44.8 | - | - | - | 60M | 5.1x | 563B | 23x | 119 | 17 |
| ResNet-50 + NAS-FPN (1280@384)[8] | 45.4 | - | - | - | 104M | 8.7x | 1043B | 42x | 173 | 27 |
| **EfficientDet-D4 (1024)** | **49.4** | **69.0** | **53.4** | **49.0** | **21M** | **1x** | **55B** | **1x** | **74** | **4.8** |
| AmoebaNet+ NAS-FPN +AA(1280)[42] | - | - | - | 48.6 | 185M | 8.8x | 1317B | 24x | 259 | 38 |
| **EfficientDet-D5 (1280)** | **50.7** | **70.2** | **54.7** | **50.5** | **34M** | **1x** | **135B** | **1x** | **141** | **11** |
| **EfficientDet-D6 (1280)** | **51.7** | **71.2** | **56.0** | **51.3** | **52M** | **1x** | **226B** | **1x** | **190** | **16** |
| AmoebaNet+ NAS-FPN +AA(1536)[42] | - | - | - | 50.7 | 209M | 4.0x | 3045B | 13x | 608 | 83 |
| **EfficientDet-D7 (1536)** | **52.2** | **71.4** | **56.3** | **51.8** | **52M** | **1x** | **325B** | **1x** | **262** | **24** |

We omit ensemble and test-time multi-scale results [27, 10].
†Latency marked with † are from papers, and others are measured on the same machine with Titan V GPU.

Table 2: **EfficientDet performance on COCO** [22] – 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` denotes inference latency with batch size 1. `AA` denotes auto-augmentation [42]. We group models together if they have similar accuracy, and compare their model size, FLOPs, and latency in each group.




(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 [42]. Our EfficientDet models are 4x - 9x smaller, 2x - 4x faster on GPU, and 5x - 11x faster on CPU than other detectors.