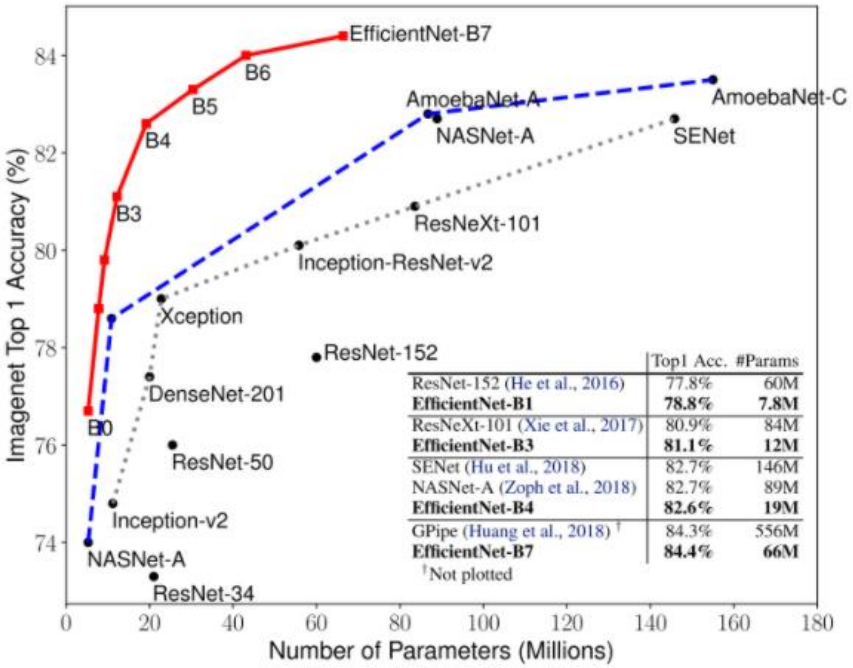
EfficientDet

- EfficientNet



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.4% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

[그림 1] 성능 비교

- GPipe보다 성능 우수, parameter 사용량은 약 **1/8배**

- Compound coefficient를 이용하여 depth/width/resolution을 균일하게 scaling

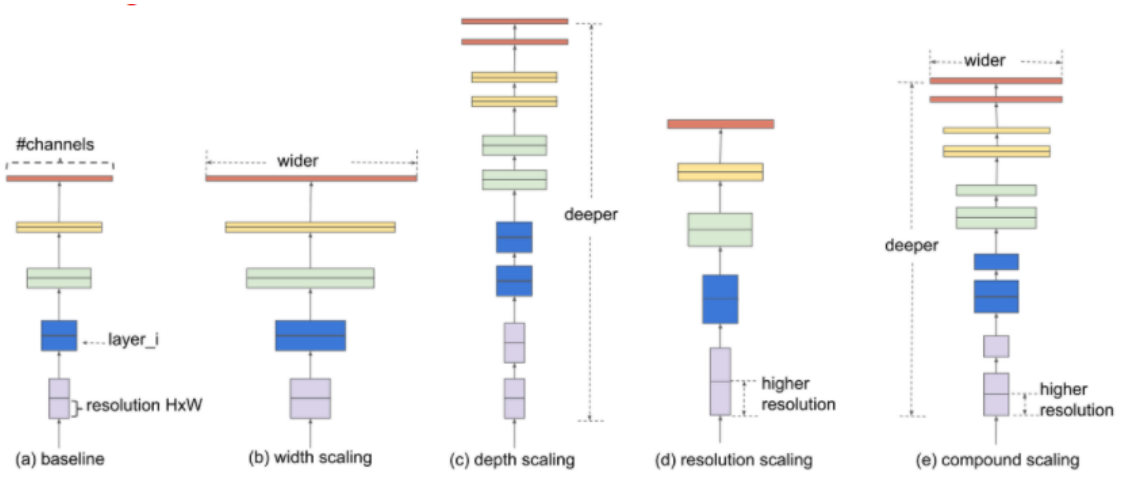


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

[그림 2] Scaling 비교

1. Width scaling

1) filter(channel)의 수를 늘이는 방식으로 scale-up

2) width를 넓게 할수록 미세한 정보(fine-grained feature)들을 더 많이 담을 수 있다고 한다.

2. Depth scaling

1) layer의 수를 늘이는 방식으로 scale-up

2) depth가 깊을수록 더 좋은 성능을 가지지만 한계가 있다.(ResNet-1000과 ResNet-101은 거의 비슷한성능)

3. Resolution scaling

1) input image의 해상도를 높이는 방식으로 scale-up

2) 높이면 높일수록 좋은 성능

4. Compound scaling

1) EfficientNet에서 제안하는 방법

2) width, depth, resolution을 적절하게 조절하여 정확도를 높이고자 한다.

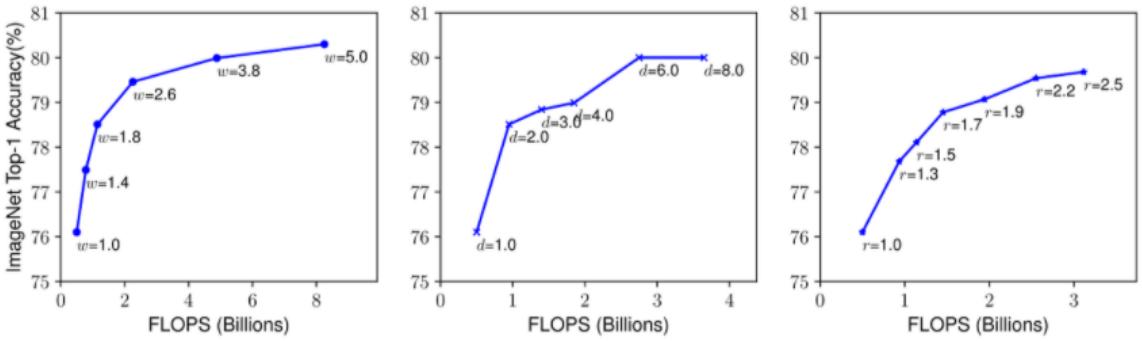
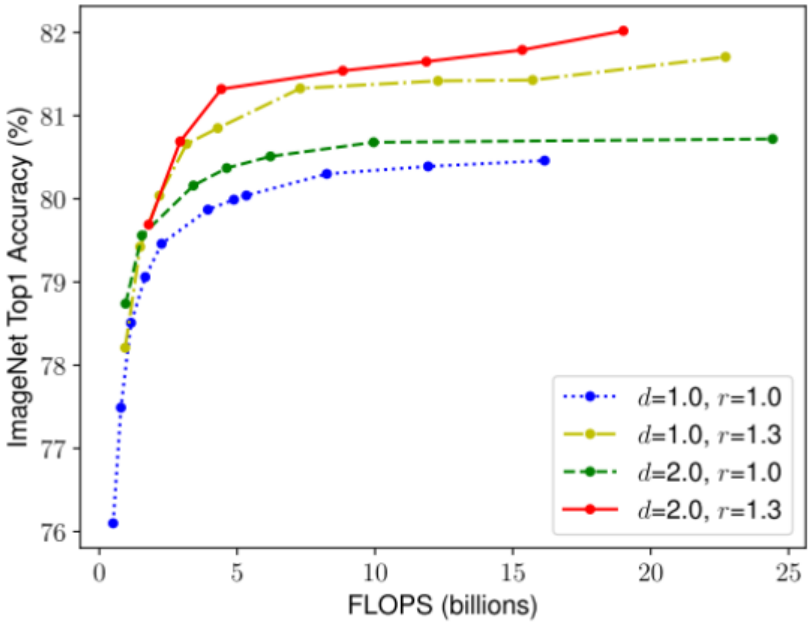


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single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

[그림 3] width/depth/resolution 에 따른 성능

width나 depth scaling 같은 경우 비교적 이른 시점에 값이 수렴하는 것을 볼 수 있다. 반면, resolution 은 값을 키우면 키울수록 성능이 향상되는 것을 확인할 수 있다.

전체적으로 봤을 때는 점점 커질수록 얻는 이득은 적어진다.



[그림 4] depth와 resolution에 따른 성능 차이(width 고정)

[그림 4]를 통해 depth를 키우는 것보다는 resolution을 키우는 것이 더 효과적인 것을 확인할 수 있다.

위의 그림과 함께 세 가지 요소를 동시에 키우는 것이 가장 성능이 좋다는 것을 확인할 수 있다.

[표 1] EfficientNet-B0

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	28×28	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

논문에서 알파 값은 1.2 베타 값은 1.1 감마 값은 1.15를 사용하고 있으며, 이 scaling factor를 고정한 뒤 파이를 키워주며 모델의 사이즈를 키워준다.

[표 2] 성능 평가

Table 2. **EfficientNet Performance Results on ImageNet** (Russakovsky et al., 2015). All EfficientNet models are scaled from our baseline EfficientNet-B0 using different compound coefficient ϕ in Equation 3. ConvNets with similar top-1/top-5 accuracy are grouped together for efficiency comparison. Our scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude (up to 8.4x parameter reduction and up to 16x FLOPS reduction) than existing ConvNets.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**76.3%**	**93.2%**	**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	**78.8%**	**94.4%**	**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	**79.8%**	**94.9%**	**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	**81.1%**	**95.5%**	**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	**82.6%**	**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	**83.3%**	**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	**84.0%**	**96.9%**	**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).

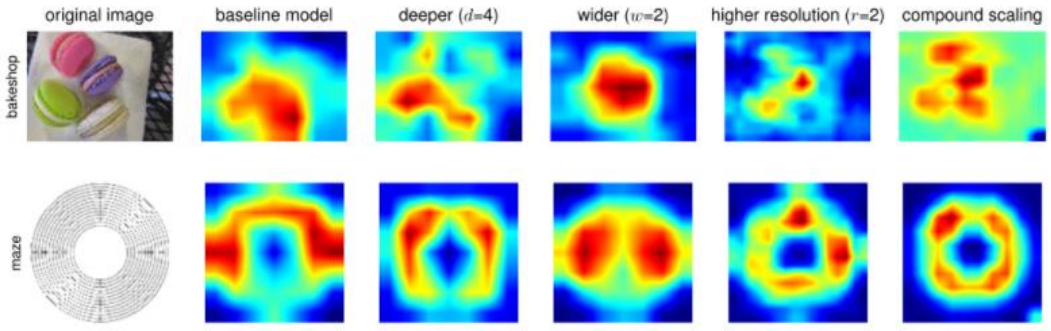


Figure 7. **Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods** – Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details.

[그림 5] Compound scaling 성능

위의 [그림 5]를 통해 width, depth, resolution 각각 고려했을 때보다 compound scaling을 통해 동시에 고려하여 scaling을 적용하는 것이 더 좋은 성능을 보이는 것을 확인할 수 있다.

- EfficientDet

EfficientDet은 EfficientNet 논문의 저자들이 속한 Google Brain 팀에서 쓴 논문이다.

여기서 BiFPN(Bi-directional Feature Pyramid Network)을 제안하였다. 이는 기존 FPN에서 레이어마다 가중치를 주어 좀 더 각각의 층에 대한 해상도 정보가 잘 들어갈 수 있도록 하는 트릭이다.

[주요 특징]

Efficient multi-scale Feature Fusion

- **기존 FPN**을 사용하는 선행 연구들이 모두 서로 다른 input feature 들을 합칠 때 구분없이 **단순히 더하는 방식**을 사용하고 있음을 지적하였다.

- 서로 다른 input feature 들은 해상도가 다르기 때문에 **output feature에 기여하는 정도를 다르게 가져가야 함을 주장**하였다. 단순히 더하기만 한다면 같은 weight로 기여하기 때문이다.

- 간단하지만 효과적인 **BiFPN** 구조를 제안한다.

Model Scaling

- EfficientNet에서 제안한 **Compound Scaling** 기법은 모델의 크기와 연산량을 결정하는 요소들인 width, depth, resolution을 동시에 고려하여 모델의 사이즈를 증가시키는 방법인데, 이런 아이디어를 **EfficientDet의 backbone, feature network, box/class prediction network** 등 **모든 곳에 적용**하였다.

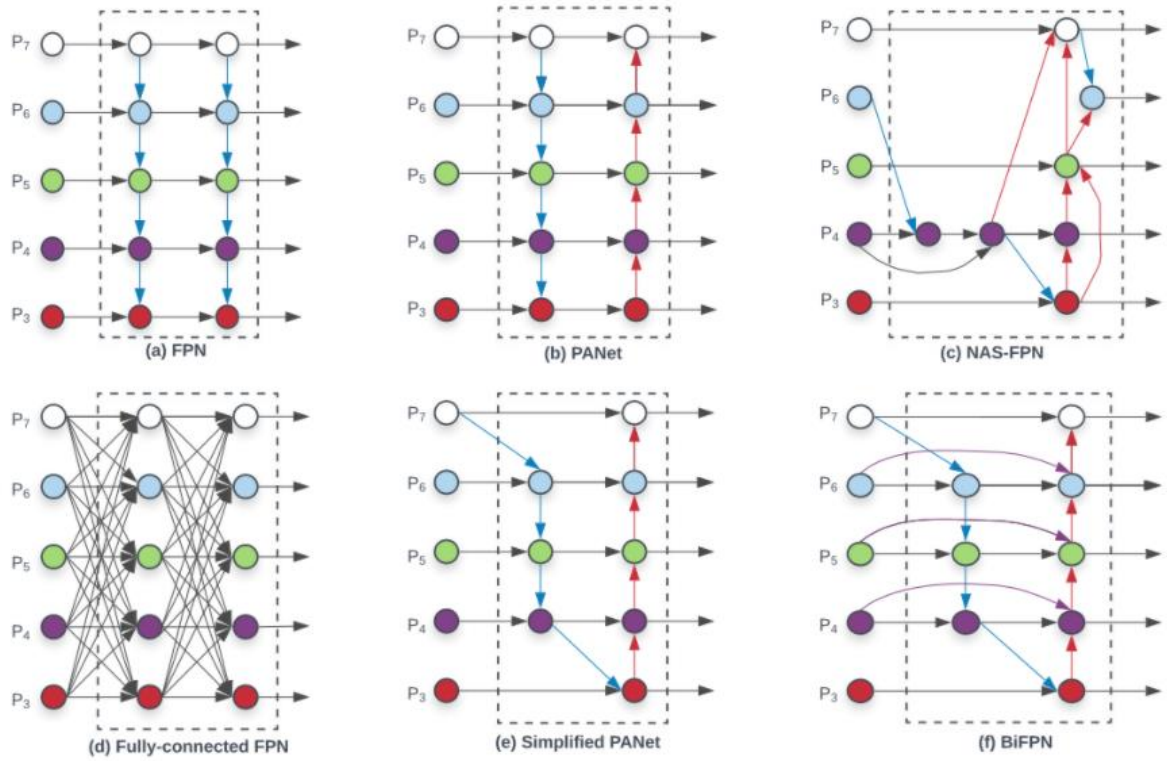


Figure 2: **Feature network design –** (a) FPN [16] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [19] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [5] use neural architecture search to find an irregular feature network topology; (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.

[그림 6] Feature Network 구조

(a) 전통적인 FPN 구조(같은 scale에만 connection이 존재)

(b) PANet은 추가로 botton-up pathway를 FPN에 추가한 방법(같은 scale에만 connection이 존재)

(c) AutoML의 Neural Architecture Search를 FPN에 적용시킨 방법(scale이 다른 경우에도 connection이 존재하는 cross-scale connection 적용)

(d), (e) 본문에서 추가로 제안하고 실험한 방식

(f) 본 논문에서 제안하는 BiFPN 구조

- scale은 I를 말하는 듯.

Weight Feature Fusion

기존 FPN에서는 **서로 다른 resolution의 input feature를 합칠 때, 같은 해상도가 되도록 resize 시킨 뒤 합치는 방식을 이용**한다. 이 논문에서는 이런 점을 개선하기 위해 input feature에 가중치를 주고, 학습을 통해 가중치를 배울 수 있는 방식을 제안하였다.

weight는 아래와 같이 세 가지 방식으로 줄 수 있다.

- scalar (= per-feature)

- vector (= per-channel)

- multi-dimensional tensor (= per-pixel)

이 논문에서는 scalar를 사용하는 것이 정확도와 연산량 측면에서 효율적임을 실험을 통해서 밝혔다.

Input Feature에 가중치를 주는 Weight Feature Fusion 방법은 아래와 같이 세 가지로 나뉜다.

- Undounded Feature Fusion

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

이는 unbound 되어 있기 때문에 학습에 불안정성을 유발할 수 있어서 weight normalization을 사용하였다고 한다.

- Softmax-based Feature Fusion

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

Softmax를 사용한 방식인데 이는 GPU에서 slowdown을 유발한다고 한다.

- Fast normalized Feature Fusion

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

weight들은 ReLU를 거치지 때문에 non-zero임이 보장되고, 분모가 0이 되는 것을 막기 위해 0.0001크기의 입실론을 넣어준다.

weight 값이 0-1 사이로 normalize되는 softmax와 유사하며 ablation study(모델이나 알고리즘의 특징을 제거하면서 그게 성능에 어떤 영향을 줄 지를 연구하는 것, 요약하자면 좀 다른 시행착오)를 통해 softmax-based fusion 방식보다 좋은 성능을 보여준다.

위 BiFPN을 기반으로 EfficientDet이라는 One-stage Detector 구조를 제안하였다.

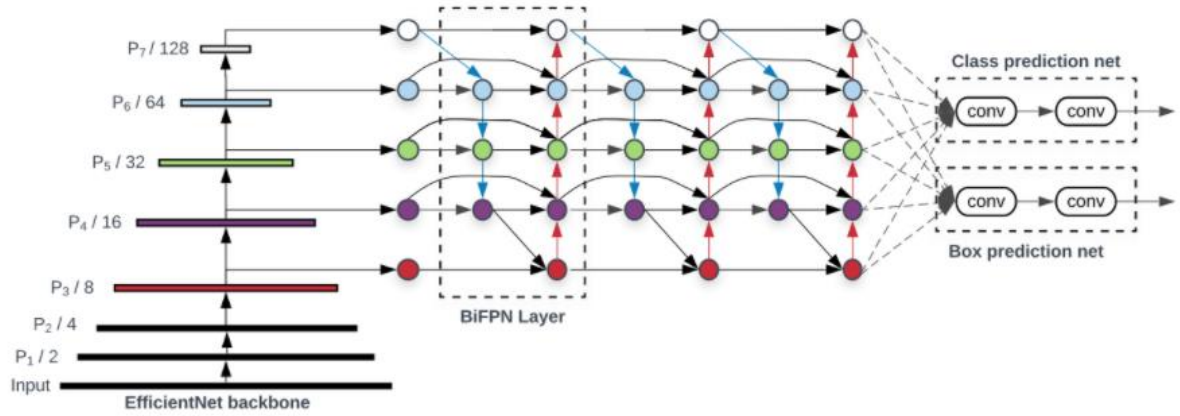


Figure 3: **EfficientDet architecture –** It employs EfficientNet [31] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

[그림 7] EfficientDet 구조

- Efficient의 backbone으로는 ImageNet-pretrained EfficientNet을 사용하였으며, BiFPN을 Feature로 사용하였고 level 3-7 feature에 적용하였다.

- top-down, bottom-up bidirectional feature fusion을 반복적으로 사용하였다.

기존 Compound Scaling처럼 input의 resolution과 backbone network의 크기를 키워주었으며, BiFPN과 Box/class network도 동시에 키웠다.

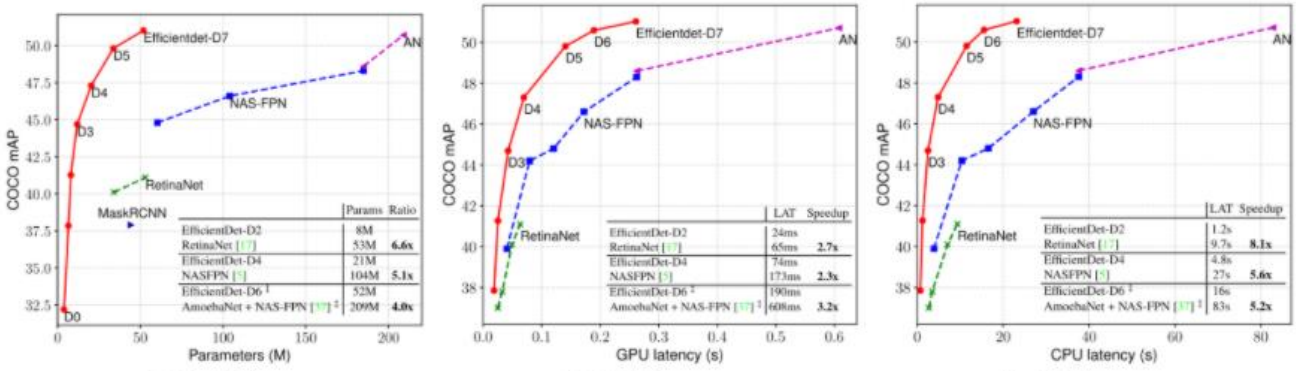
	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	2	3
D1 ($\phi = 1$)	640	B1	88	3	3
D2 ($\phi = 2$)	768	B2	112	4	3
D3 ($\phi = 3$)	896	B3	160	5	4
D4 ($\phi = 4$)	1024	B4	224	6	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1408	B6	384	8	5
D7	1536	B6	384	8	5

Table 1: **Scaling configs for EfficientDet D0-D7 –** 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. D7 has the same settings as D6 except using larger input size.

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**

We omit ensemble and test-time multi-scale results [23, 7].
†Latency numbers marked with † are from papers, and all others are measured on the same machine.

Table 2: **EfficientDet performance on COCO** [18] – Results are for single-model single-scale. `#Params` and `#FLOPS` denote the number of parameters and multiply-adds. `LAT` denotes inference latency with batch size 1. `AA` denotes auto-augmentation [37]. We group models together if they have similar accuracy, and compare the ratio or speedup between EfficientDet and other detectors 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 [37]. Our EfficientDet models are 4x - 6.6x smaller, 2.3x - 3.2x faster on GPU, and 5.2x - 8.1x faster on CPU than other detectors.

<출처, 참고자료>

[1] https://eehoeskrap.tistory.com/404

[2] https://hoya012.github.io/blog/EfficientNet-review/

[3] https://norman3.github.io/papers/docs/efficient_net

[4] https://hoya012.github.io/blog/EfficientDet-Review/

[5] https://arxiv.org/abs/1911.09070

[6] https://arxiv.org/abs/1905.11946