# EfficientNet

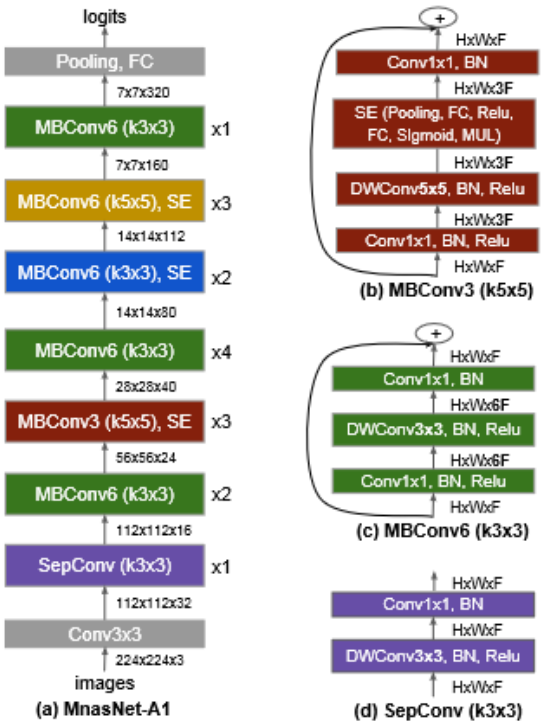
EfficientNet

## 구조

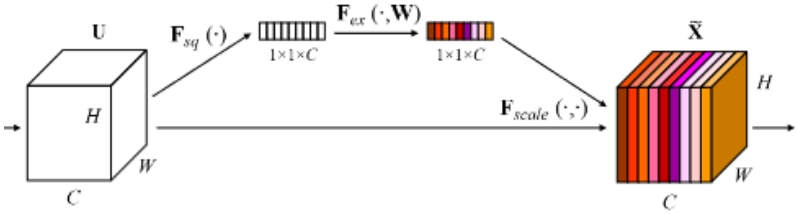
- 모델의 크기만을 바꾸기 때문에 바탕이 되는 모델을 적절히 선택하는 게 중요했다.
- 계산량과 정확도를 $ACC(m) \times [FLOPS(m)/T]^w$ 식으로 정량화하여 여러 모델을 탐색.
- 그 결과 구조는 MNasNet과 비슷한 EfficientNetB0을 얻었다.(상대적으로 조금 큼).

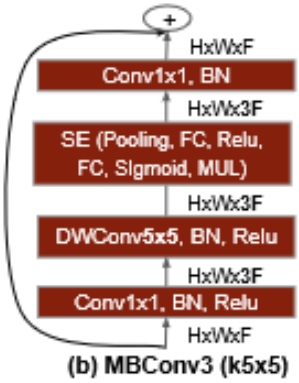
*Table 1.* **EfficientNet-B0 baseline network** – Each row describes a stage $i$ with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

| 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 | $224 \times 224$ | 32 | 1 |
| 2 | MBConv1, k3x3 | $112 \times 112$ | 16 | 1 |
| 3 | MBConv6, k3x3 | $112 \times 112$ | 24 | 2 |
| 4 | MBConv6, k5x5 | $56 \times 56$ | 40 | 2 |
| 5 | MBConv6, k3x3 | $28 \times 28$ | 80 | 3 |
| 6 | MBConv6, k5x5 | $14 \times 14$ | 112 | 3 |
| 7 | MBConv6, k5x5 | $14 \times 14$ | 192 | 4 |
| 8 | MBConv6, k3x3 | $7 \times 7$ | 320 | 1 |
| 9 | Conv1x1 & Pooling & FC | $7 \times 7$ | 1280 | 1 |




- mobile inverted bottleneck MBConv에 squeeze and excitation을 더한 layer가 주 구성요소이다.




(b) MBConv3 (k5x5)

```
class SEBlock(nn.Module):
    def __init__(self,channel,reduction_rate):
        super().__init__()
        self.squeeze=nn.AdaptiveAvgPool2d((1,1))#Global Average Pooling
        self.excitation=nn.Sequential(
            nn.Linear(channel,channel//reduction_rate),
            nn.ReLU(),
            nn.Linear(channel//reduction_rate,channel),
            nn.Sigmoid()
        )

    def forward(self,x):#x.shpae=(N,C,H,W)
        out=self.squeeze(x)#out.shpae=(N,C,1,1)
        out=out.view(x.size(0),-1)#out.shpae=(N,C)
        out=self.excitation(out)#out.shpae=(N,C)
        out=out.view(x.size(0),x.size(1),1,1)#out.shpae=(N,C,1,1)
        return out

class MBConv6(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1):
        super().__init__()
        self.convi=nn.Sequential(
            nn.Conv2d(in_channels=in_channels,out_channels=6*in_channels,kernel_size=1),
            nn.BatchNorm2d(6*in_channels,eps=1e-05, momentum=0.99),
            nn.ReLU()
        )
        self.DWConv=nn.Sequential(
            DepthwiseConv(in_channels=6*in_channels,kernel_size=kernel_size),
            nn.BatchNorm2d(6*in_channels,eps=1e-05, momentum=0.99),
            nn.ReLU()
        )
        self.SE=SEBlock(channel=6*in_channels,reduction_rate=8)
        self.convf=nn.Sequential(
            nn.Conv2d(in_channels=6*in_channels,out_channels=out_channels,kernel_size=1,stride=stride),
            nn.BatchNorm2d(out_channels,eps=1e-05, momentum=0.99)
        )
        self.skip=(stride==1) and (in_channels==out_channels)

    def forward(self, x):
        out=self.convi(x)
        out=self.DWConv(out)
        out=self.SE(out)*out
        out=self.convf(out)
        if self.skip:
            out=out+x
        return out
```

- 위 모델을 기반으로 $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$라는 최고의 계수를 찾아냈다. ~~BN 모델들은 N 제곱 크기 조절이 적용된 모델.

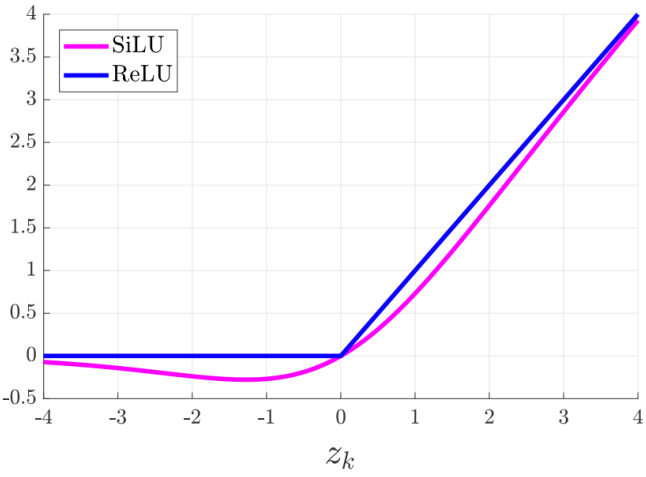
## 실험

### ▼ 실험조건

- AutoAugment
- Learning rate=0.256 decays by 0.97 per 2.4 epochs
- RMSProp optimizer with decay 0.9, momentum 0.9

$$G_t = \gamma G_{t-1} + (1-\gamma) g_t^2$$
$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

- Batch normalization momentum 0.99
- Weight decay 1e-5
- SiLU(Swish-1) for activation function $\mathrm{SiLU}(x) = x \cdot \sigma(x)$

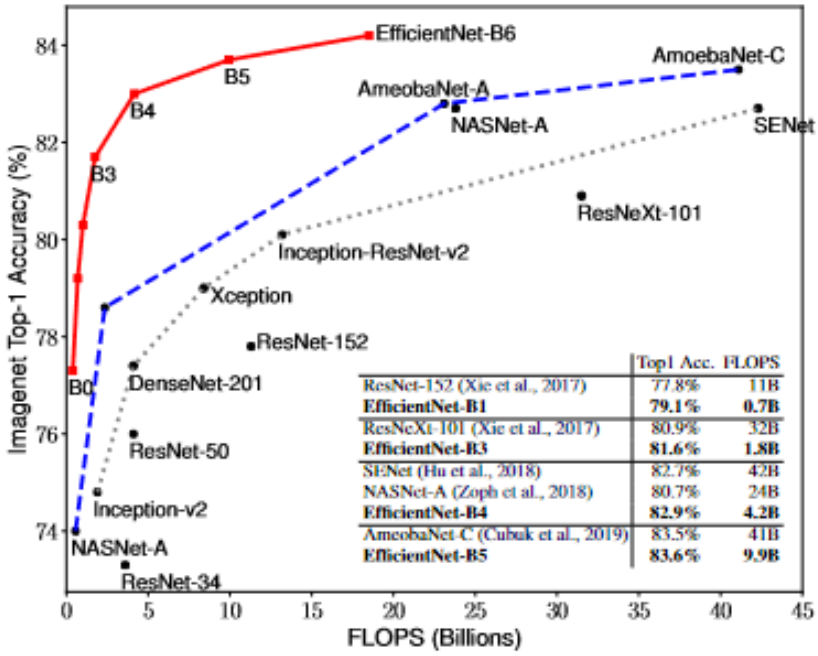



- Stochastic depth with survival probablity 0.8. (Residual block drop)
- Dropout ratio 0.2~0.5
- 25K ramdomly picked images for early stopping

*Table 3.* **Scaling Up MobileNets and ResNet.**

| Model | FLOPS | Top-1 Acc. |
|---|---|---|
| Baseline MobileNetV1 (Howard et al., 2017) | 0.6B | 70.6% |
| Scale MobileNetV1 by width ($w$=2) | 2.2B | 74.2% |
| Scale MobileNetV1 by resolution ($r$=2) | 2.2B | 72.7% |
| **compound scale ($d$=1.4, $w$=1.2, $r$=1.3)** | **2.3B** | **75.6%** |
| Baseline MobileNetV2 (Sandler et al., 2018) | 0.3B | 72.0% |
| Scale MobileNetV2 by depth ($d$=4) | 1.2B | 76.8% |
| Scale MobileNetV2 by width ($w$=2) | 1.1B | 76.4% |
| Scale MobileNetV2 by resolution ($r$=2) | 1.2B | 74.8% |
| **MobileNetV2 compound scale** | **1.3B** | **77.4%** |
| Baseline ResNet-50 (He et al., 2016) | 4.1B | 76.0% |
| Scale ResNet-50 by depth ($d$=4) | 16.2B | 78.1% |
| Scale ResNet-50 by width ($w$=2) | 14.7B | 77.7% |
| Scale ResNet-50 by resolution ($r$=2) | 16.4B | 77.5% |
| **ResNet-50 compound scale** | **16.7B** | **78.8%** |

- MobileNet과 ResNet에 한 영역에 대해서만 크기 조절을 한 모델과 우리의 기법을 적용한 것을 ImageNet에서 돌려서 비교한 결과.




- 결과를 다른 Conv. Net과 비교해보니 효율면에서 아주 좋은 성적을 보여주었다.
  ResNext-101과 B3를 보면 성능이 더 좋음에도 18배의 계산량 차이가 났다.

*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 $\phi$ 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** | **77.1%** | **93.3%** | **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** | **79.1%** | **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** | **80.1%** | **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** | **81.6%** | **95.7%** | **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** | **82.9%** | **96.4%** | **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** | **83.6%** | **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%** | **96.8%** | **43M** | **1x** | **19B** | **1x** |
| **EfficientNet-B7** | **84.3%** | **97.0%** | **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).

- 아래는 B0 모델을 크기 조절 방식을 다르게 한 결과 비교이다.

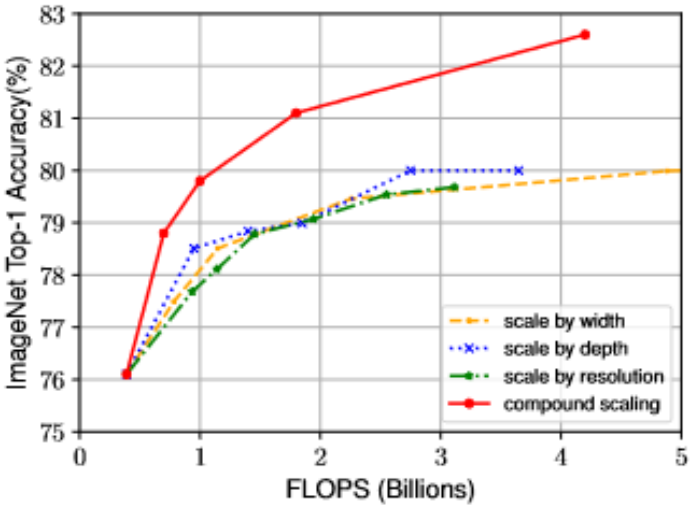



*Figure 8.* **Scaling Up EfficientNet-B0 with Different Methods.**

*Table 7.* **Scaled Models Used in Figure 7.**

| Model | FLOPS | Top-1 Acc. |
|---|---|---|
| Baseline model (EfficientNet-B0) | 0.4B | 77.3% |
| Scale model by depth ($d$=4) | 1.8B | 79.0% |
| Scale model by width ($w$=2) | 1.8B | 78.9% |
| Scale model by resolution ($r$=2) | 1.9B | 79.1% |
| **Compound Scale ($d$=1.4, $w$=1.2, $r$=1.3)** | **1.8B** | **81.1%** |

## 결론

Conv. Net의 크기를 조절하는 법에 대해 알아보았는데 세 영역에 대한 크기를 각각 조절하는 것은 힘들다. 그렇기에 지금까지 본 것처럼 효율적인 결과를 보여주는 세 영역을 일정 비율로 늘리는 방법을 제안했다.