EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] **Quoc V. Le** [1]

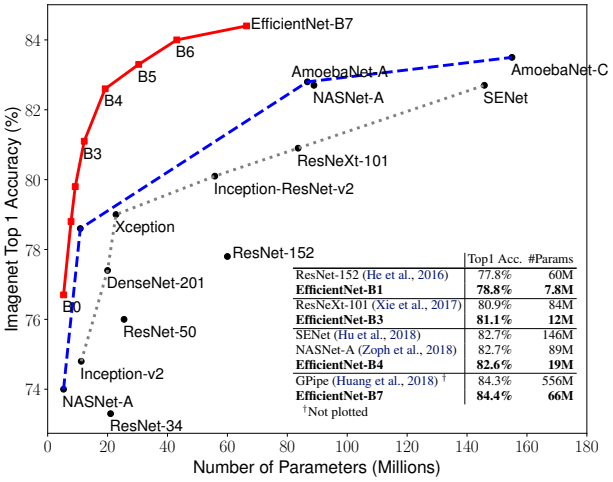
Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy on ImageNet, while being **8.4x smaller** and **6.1x faster** on inference than the best existing ConvNet. Our EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters.

1. Introduction

Scaling up ConvNets is widely used to achieve better accuracy. For example, ResNet (He et al., 2016) can be scaled up from ResNet-18 to ResNet-200 by using more layers; Recently, GPipe (Huang et al., 2018) achieved 84.3% ImageNet top-1 accuracy by scaling up a baseline model four time larger. However, the process of scaling up ConvNets has never been well understood and there are currently many



	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.

ways to do it. The most common way is to scale up ConvNets by their depth (He et al., 2016) or width (Zagoruyko & Komodakis, 2016). Another less common, but increasingly popular, method is to scale up models by image resolution (Huang et al., 2018). In previous work, it is common to scale only one of the three dimensions – depth, width, and image size. Though it is possible to scale two or three dimensions arbitrarily, arbitrary scaling requires tedious manual tuning and still often yields sub-optimal accuracy and efficiency.

In this paper, we want to study and rethink the process of scaling up ConvNets. In particular, we investigate the central question: is there a principled method to scale up ConvNets that can achieve better accuracy and efficiency? Our empirical study shows that it is critical to balance all dimensions of network width/depth/resolution, and surprisingly such balance can be achieved by simply scaling each of them with constant ratio. Based on this observation, we propose a simple yet effective *compound scaling method*. Unlike conventional practice that arbitrary scales these factors, our method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. For example, if we want to use 2^N times more computational

[1]Google Research, Brain Team, Mountain View, CA. Correspondence to: Mingxing Tan <tanmingxing@google.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

Basically,
- uniformly scaling depth, width, resolution by a factor.
- Scaling all three instead of just one

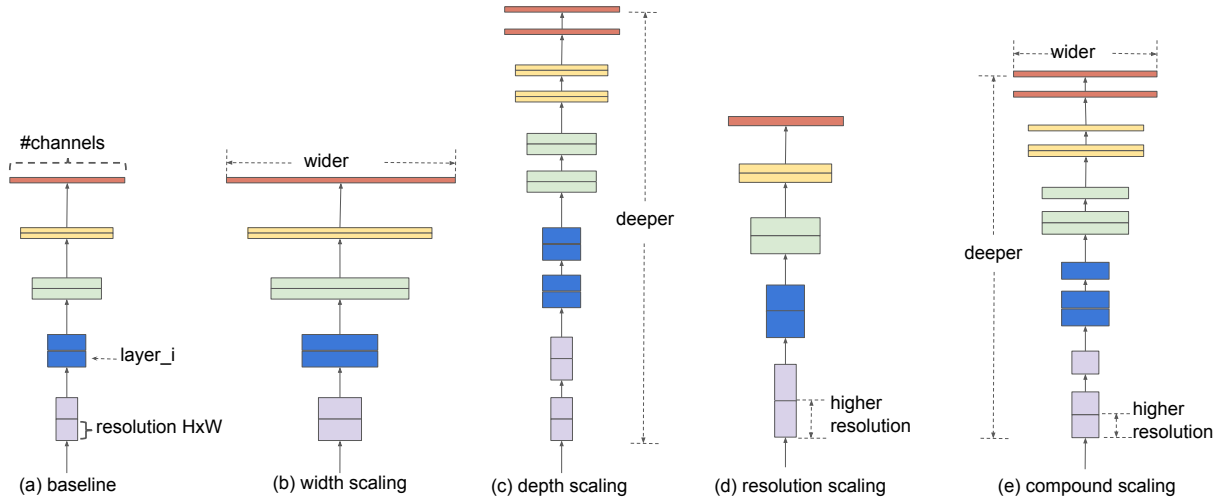


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

resources, then we can simply increase the network depth by α^N, width by β^N, and image size by γ^N, where α, β, γ are constant coefficients determined by a small grid search on the original small model. Figure 2 illustrates the difference between our scaling method and conventional methods.

Intuitively, the compound scaling method makes sense because if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image. In fact, previous theoretical (Raghu et al., 2017; Lu et al., 2018) and empirical results (Zagoruyko & Komodakis, 2016) both show that there exists certain relationship between network width and depth, but to our best knowledge, we are the first to empirically quantify the relationship among all three dimensions of network width, depth, and resolution.

We demonstrate that our scaling method work well on existing MobileNets (Howard et al., 2017; Sandler et al., 2018) and ResNet (He et al., 2016). Notably, the effectiveness of model scaling heavily depends on the baseline network; to go even further, we use neural architecture search (Zoph & Le, 2017; Tan et al., 2019) to develop a new baseline network, and scale it up to obtain a family of models, called *EfficientNets*. Figure 1 summarizes the ImageNet performance, where our EfficientNets significantly outperform other ConvNets. In particular, our EfficientNet-B7 surpasses the best existing GPipe accuracy (Huang et al., 2018), but using 8.4x fewer parameters and running 6.1x faster on inference. Compared to the widely used ResNet (He et al., 2016), our EfficientNet-B4 improves the top-1 accuracy from 76.3% of ResNet-50 to 82.6% with similar FLOPS. Besides ImageNet, EfficientNets also transfer well and achieve state-of-the-art accuracy on 5 out of 8 widely used datasets, while reducing parameters by up to 21x than existing ConvNets.

2. Related Work

ConvNet Accuracy: Since AlexNet (Krizhevsky et al., 2012) won the 2012 ImageNet competition, ConvNets have become increasingly more accurate by going bigger: while the 2014 ImageNet winner GoogleNet (Szegedy et al., 2015) achieves 74.8% top-1 accuracy with about 6.8M parameters, the 2017 ImageNet winner SENet (Hu et al., 2018) achieves 82.7% top-1 accuracy with 145M parameters. Recently, GPipe (Huang et al., 2018) further pushes the state-of-the-art ImageNet top-1 validation accuracy to 84.3% using 557M parameters: it is so big that it can only be trained with a specialized pipeline parallelism library by partitioning the network and spreading each part to a different accelerator. While these models are mainly designed for ImageNet, recent studies have shown better ImageNet models also perform better across a variety of transfer learning datasets (Kornblith et al., 2019), and other computer vision tasks such as object detection (He et al., 2016; Tan et al., 2019). Although higher accuracy is critical for many applications, we have already hit the hardware memory limit, and thus further accuracy gain needs better efficiency.

ConvNet Efficiency: Deep ConvNets are often over-parameterized. Model compression (Han et al., 2016; He et al., 2018; Yang et al., 2018) is a common way to reduce model size by trading accuracy for efficiency. As mobile phones become ubiquitous, it is also common to handcraft efficient mobile-size ConvNets, such as SqueezeNets (Iandola et al., 2016; Gholami et al., 2018), MobileNets (Howard et al., 2017; Sandler et al., 2018), and ShuffleNets (Zhang et al., 2018; Ma et al., 2018). Recently, neural architecture search becomes increasingly popular in designing efficient mobile-size ConvNets (Tan et al., 2019; Cai et al.,

2019), and achieves even better efficiency than hand-crafted mobile ConvNets by extensively tuning the network width, depth, convolution kernel types and sizes. However, it is unclear how to apply these techniques for larger models that have much larger design space and much more expensive tuning cost. In this paper, we aim to study model efficiency for super large ConvNets that surpass state-of-the-art accuracy. To achieve this goal, we resort to model scaling.

Model Scaling: There are many ways to scale a ConvNet for different resource constraints: ResNet (He et al., 2016) can be scaled down (e.g., ResNet-18) or up (e.g., ResNet-200) by adjusting network depth (#layers), while WideResNet (Zagoruyko & Komodakis, 2016) and MobileNets (Howard et al., 2017) can be scaled by network width (#channels). It is also well-recognized that bigger input image size will help accuracy with the overhead of more FLOPS. Although prior studies (Raghu et al., 2017; Lin & Jegelka, 2018; Sharir & Shashua, 2018; Lu et al., 2018) have shown that network deep and width are both important for ConvNets' expressive power, it still remains an open question of how to effectively scale a ConvNet to achieve better efficiency and accuracy. Our work systematically and empirically studies ConvNet scaling for all three dimensions of network width, depth, and resolutions.

3. Compound Model Scaling

In this section, we will formulate the scaling problem, study different approaches, and propose our new scaling method.

3.1. Problem Formulation

A ConvNet Layer i can be defined as a function: $Y_i = \mathcal{F}_i(X_i)$, where $\mathcal{F}_i$ is the operator, Y_i is output tensor, X_i is input tensor, with tensor shape $\langle H_i, W_i, C_i \rangle$[1], where H_i and W_i are spatial dimension and C_i is the channel dimension. A ConvNet $\mathcal{N}$ can be represented by a list of composed layers: $\mathcal{N} = \mathcal{F}_k \odot ... \odot \mathcal{F}_1 \odot \mathcal{F}_1(X_1) = \bigodot_{j=1...k} \mathcal{F}_j(X_1)$. In practice, ConvNet layers are often partitioned into multiple stages and all layers in each stage share the same architecture: for example, ResNet (He et al., 2016) has five stages, and all layers in each stage has the same convolutional type except the first layer performs down-sampling. Therefore, we can define a ConvNet as:

$$\mathcal{N} = \bigodot_{i=1...s} \mathcal{F}_i^{L_i}\big(X_{\langle H_i, W_i, C_i \rangle}\big) \tag{1}$$

where $\mathcal{F}_i^{L_i}$ denotes layer F_i is repeated L_i times in stage i, $\langle H_i, W_i, C_i \rangle$ denotes the shape of input tensor X of layer i. Figure 2(a) illustrate a representative ConvNet, where the spatial dimension is gradually shrunk but the channel dimension is expanded over layers, for example, from initial input shape $\langle 224, 224, 3 \rangle$ to final output shape $\langle 7, 7, 512 \rangle$.

Unlike regular ConvNet designs that mostly focus on finding the best layer architecture $\mathcal{F}_i$, model scaling tries to expand the network length (L_i), width (C_i), and/or resolution (H_i, W_i) without changing $\mathcal{F}_i$ predefined in the baseline network. By fixing $\mathcal{F}_i$, model scaling simplifies the design problem for new resource constraints, but it still remains a large design space to explore different L_i, C_i, H_i, W_i for each layer. In order to further reduce the design space, we restrict that all layers must be scaled uniformly with constant ratio. Our target is to maximize the model accuracy for any given resource constraints, which can be formulated as an optimization problem:

$$\begin{aligned} \max_{d,w,r} \quad & Accuracy\big(\mathcal{N}(d,w,r)\big) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1...s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\big(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\big) \\ & \text{Memory}(\mathcal{N}) \le \text{target\_memory} \\ & \text{FLOPS}(\mathcal{N}) \le \text{target\_flops} \end{aligned} \tag{2}$$

where w, d, r are coefficients for scaling network width, depth, and resolution; $\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in baseline network (see Table 1 as an example).

3.2. Scaling Dimensions

The main difficulty of problem 2 is that the optimal d, w, r depend on each other and the values change under different resource constraints. Due to this difficulty, conventional methods mostly scale ConvNets in one of these dimensions:

Depth (d): Scaling network depth is the most common way used by many ConvNets (He et al., 2016; Huang et al., 2017; Szegedy et al., 2015; 2016). The intuition is that deeper ConvNet can capture richer and more complex features, and generalize well on new tasks. However, deeper networks are also more difficult to train due to the vanishing gradient problem (Zagoruyko & Komodakis, 2016). Although several techniques, such as skip connections (He et al., 2016) and batch normalization (Ioffe & Szegedy, 2015), alleviate the training problem, the accuracy gain of very deep network diminishes: for example, ResNet-1000 has similar accuracy as ResNet-101 even though it has much more layers. Figure 3 (middle) shows our empirical study on scaling a baseline model with different depth coefficient d, further suggesting the diminishing accuracy return for very deep ConvNets.

Width (w): Scaling network width is commonly used for small size models (Howard et al., 2017; Sandler et al., 2018;

[1]For the sake of simplicity, we omit batch dimension.

Preferred: more width, depth, resolution → hard to train, choose factor
→ grid search to find best

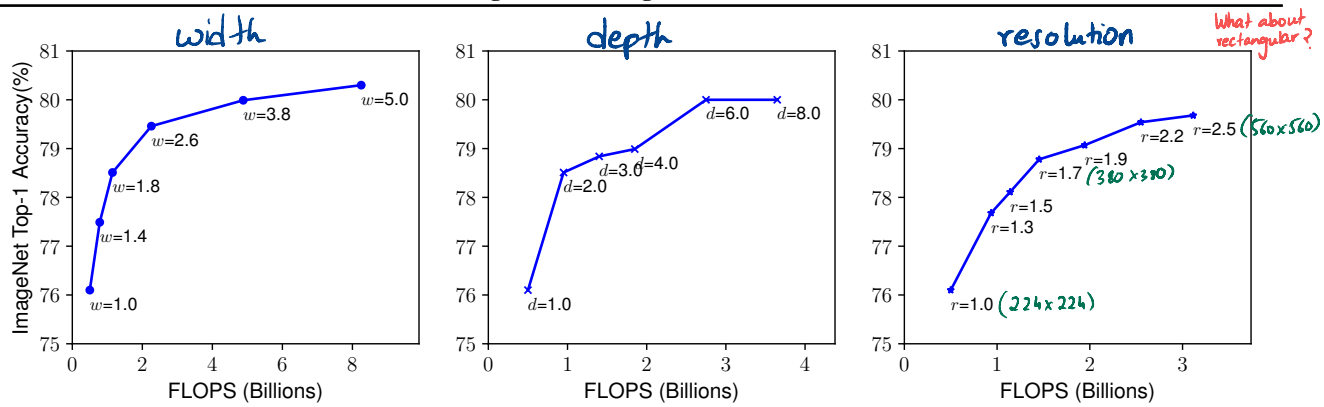


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single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

Tan et al., 2019)[2]. As discussed in (Zagoruyko & Komodakis, 2016), wider networks tend to be able to capture more fine-grained features and are easier to train. However, extremely wide but shallow networks tend to have difficulties in capturing higher level features. Our empirical results in Figure 3 (left) show that the accuracy quickly saturates when networks become much wider with larger w.

Resolution (r): With higher resolution input images, ConvNets can potentially capture more fine-grained patterns. Starting from 224x224 in early ConvNets, modern ConvNets tend to use 299x299 (Szegedy et al., 2016) or 331x331 (Zoph et al., 2018) for better accuracy. Recently, GPipe (Huang et al., 2018) achieves state-of-the-art ImageNet accuracy with 480x480 resolution. Higher resolutions, such as 600x600, are also widely used in object detection ConvNets (He et al., 2017; Lin et al., 2017). Figure 3 (right) shows the results of scaling network resolutions, where indeed higher resolutions improve accuracy, but the accuracy gain diminishes for very high resolutions ($r = 1.0$ denotes resolution 224x224 and $r = 2.5$ denotes resolution 560x560).

The above analyses lead us to the first observation:

Observation 1 – Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models.

3.3. Compound Scaling

We empirically observe that different scaling dimensions are not independent. Intuitively, for higher resolution images, we should increase network depth, such that the larger receptive fields can help capture similar features that include more pixels in bigger images. Correspondingly, we should also increase network depth when resolution is higher, in order to capture more fine-grained patterns with more pixels in high resolution images. These intuitions suggest that we need to coordinate and balance different scaling dimensions rather than conventional single-dimension scaling.

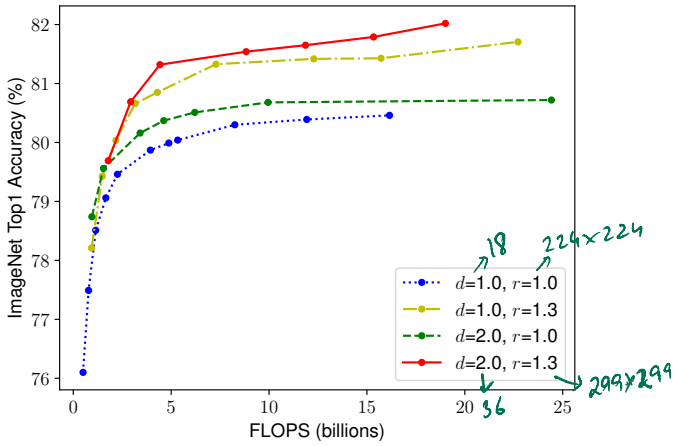


Figure 4. **Scaling Network Width for Different Baseline Networks.** Each dot in a line denotes a model with different width coefficient (w). All baseline networks are from Table 1. The first baseline network (d=1.0, r=1.0) has 18 convolutional layers with resolution 224x224, while the last baseline (d=2.0, r=1.3) has 36 layers with resolution 299x299.

To validate our intuitions, we compare width scaling under different network depths and resolutions, as shown in Figure 4. If we only scale network width w without changing depth (d=1.0) and resolution (r=1.0), the accuracy saturates quickly. With deeper (d=2.0) and higher resolution (r=2.0), width scaling achieves much better accuracy under the same FLOPS cost. These results lead us to the second observation:

Observation 2 – In order to pursue better accuracy and efficiency, it is critical to balance all dimensions of network width, depth, and resolution during ConvNet scaling.

[2]In some literature, scaling number of channels is called "depth multiplier", which means the same as our width coefficient w.

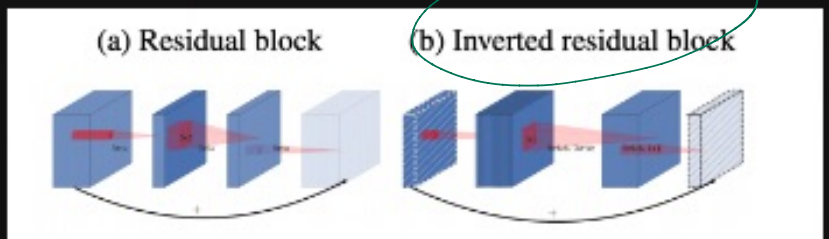
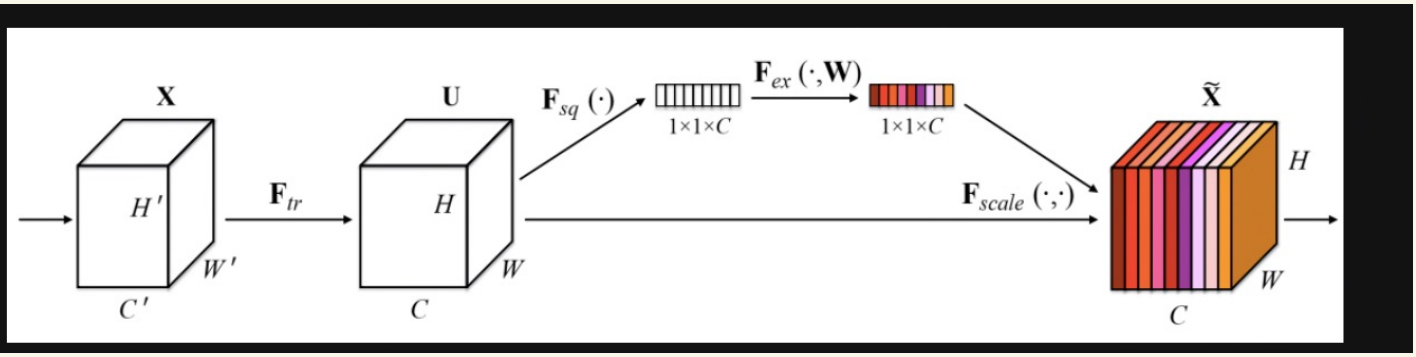


Figure 3: The difference between residual block [8, 30] and inverted residual. Diagonally hatched layers do not use non-linearities. We use thickness of each block to indicate its relative number of channels. Note how classical residuals connects the layers with high number of channels, whereas the inverted residuals connect the bottlenecks. Best viewed in color.

MBconv

Squeeze & excitation



linear (ch) → sig
linear (ch // ratio) → relu
global Avg Pool

In fact, a few prior work (Zoph et al., 2018; Real et al., 2019) have already tried to arbitrarily balance network width and depth, but they all require tedious manual tuning.

In this paper, we propose a new **compound scaling method**, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ & \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

where α, β, γ are constants that can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. Notably, the FLOPS of a regular convolution op is proportional to d, w^2, r^2, i.e., doubling network depth will double FLOPS, but doubling network width or resolution will increase FLOPS by four times. Since convolution ops usually dominate the computation cost in ConvNets, scaling a ConvNet with equation 3 will approximately increase total FLOPS by $\left(\alpha \cdot \beta^2 \cdot \gamma^2\right)^\phi$. In this paper, we constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ such that for any new ϕ, the total FLOPS will approximately[3] increase by 2^ϕ.

4. EfficientNet Architecture

Since model scaling does not change layer operators $\hat{\mathcal{F}}_i$ in baseline network, having a good baseline network is also critical. We will evaluate our scaling method using existing ConvNets, but in order to better demonstrate the effectiveness of our scaling method, we have also developed a new mobile-size baseline, called EfficientNet.

Inspired by (Tan et al., 2019), we develop our baseline network by leveraging a multi-objective neural architecture search that optimizes both accuracy and FLOPS. Specifically, we use the same search space as (Tan et al., 2019), and use $ACC(m) \times [FLOPS(m)/T]^w$ as the optimization goal, where $ACC(m)$ and $FLOPS(m)$ denote the accuracy and FLOPS of model m, T is the target FLOPS and w=-0.07 is a hyperparameter for controlling the trade-off between accuracy and FLOPS. Unlike (Tan et al., 2019; Cai et al., 2019), here we optimize FLOPS rather than latency since we are not targeting any specific hardware device. Our search produces an efficient network, which we name EfficientNet-B0. Since we use the same search space as (Tan et al., 2019), the architecture is similar to MnasNet, except our EfficientNet-B0 is slightly bigger due to the larger FLOPS target (our FLOPS target is 400M). Table 1 shows the architecture of EfficientNet-B0. Its main building block is mobile inverted bottleneck MBConv (Sandler et al., 2018; Tan et al., 2019), to which we also add squeeze-and-excitation optimization (Hu et al., 2018).

[3] FLOPS may differ from theocratic value due to rounding.

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

Starting from the baseline EfficientNet-B0, we apply our compound scaling method to scale it up with two steps:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

Notably, it is possible to achieve even better performance by searching for α, β, γ directly around a large model, but the search cost becomes prohibitively more expensive on larger models. Our method solves this issue by only doing search once on the small baseline network (step 1), and then use the same scaling coefficients for all other models (step 2).

5. Experiments

In this section, we will first evaluate our scaling method on existing ConvNets and the new proposed EfficientNets.

5.1. Scaling Up MobileNets and ResNets

As a proof of concept, we first apply our scaling method to the widely-used MobileNets (Howard et al., 2017; Sandler et al., 2018) and ResNet (He et al., 2016). Table 3 shows the ImageNet results of scaling them in different ways. Compared to other single-dimension scaling methods, our compound scaling method improves the accuracy on all these models, suggesting the effectiveness of our proposed scaling method for general existing ConvNets.

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).

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%**

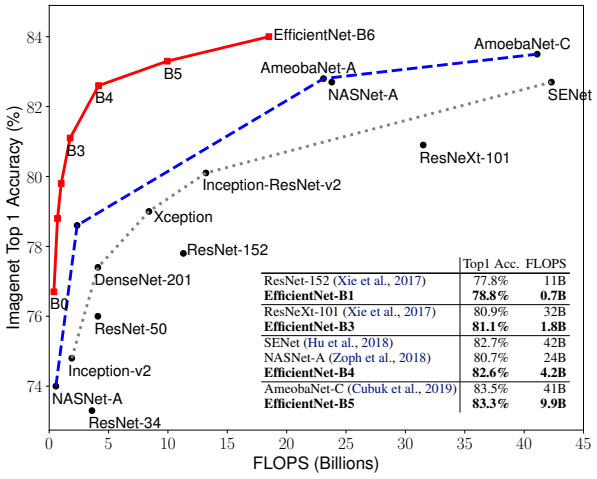


Figure 5. **FLOPS vs. ImageNet Accuracy.**

Table 4. **Inference Latency Comparison** – Latency is measured with batch size 1 on a single core of Intel Xeon CPU E5-2690.

	Acc. @ Latency		Acc. @ Latency
ResNet-152	77.8% @ 0.554s	GPipe	84.3% @ 19.0s
EfficientNet-B1	78.8% @ 0.098s	EfficientNet-B7	84.4% @ 3.1s
Speedup	**5.7x**	**Speedup**	**6.1x**

5.2. ImageNet Results for EfficientNet

We train our EfficientNet models on ImageNet using similar settings as (Tan et al., 2019): RMSProp optimizer with decay 0.9 and momentum 0.9; batch norm momentum 0.99; weight decay 1e-5; initial learning rate 0.256 that decays by 0.97 every 2.4 epochs. We also use swish activation

Table 5. **EfficientNet Performance Results on Transfer Learning Datasets**. Our scaled EfficientNet models achieve new state-of-the-art accuracy for 5 out of 8 datasets, with 9.6x fewer parameters on average.

	Comparison to best public-available results						Comparison to best reported results					
	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)
CIFAR-10	NASNet-A	98.0%	85M	EfficientNet-B0	98.1%	4M (21x)	†Gpipe	**99.0%**	556M	EfficientNet-B7	98.9%	64M (8.7x)
CIFAR-100	NASNet-A	87.5%	85M	EfficientNet-B0	88.1%	4M (21x)	Gpipe	91.3%	556M	EfficientNet-B7	**91.7%**	64M (8.7x)
Birdsnap	Inception-v4	81.8%	41M	EfficientNet-B5	82.0%	28M (1.5x)	GPipe	83.6%	556M	EfficientNet-B7	**84.3%**	64M (8.7x)
Stanford Cars	Inception-v4	93.4%	41M	EfficientNet-B3	93.6%	10M (4.1x)	‡DAT	**94.8%**	-	EfficientNet-B7	94.7%	-
Flowers	Inception-v4	98.5%	41M	EfficientNet-B5	98.5%	28M (1.5x)	DAT	97.7%	-	EfficientNet-B7	**98.8%**	-
FGVC Aircraft	Inception-v4	90.9%	41M	EfficientNet-B3	90.7%	10M (4.1x)	DAT	92.9%	-	EfficientNet-B7	**92.9%**	-
Oxford-IIIT Pets	ResNet-152	94.5%	58M	EfficientNet-B4	94.8%	17M (5.6x)	GPipe	**95.9%**	556M	EfficientNet-B6	95.4%	41M (14x)
Food-101	Inception-v4	90.8%	41M	EfficientNet-B4	91.5%	17M (2.4x)	GPipe	93.0%	556M	EfficientNet-B7	**93.0%**	64M (8.7x)
Geo-Mean						**(4.7x)**						**(9.6x)**

†GPipe (Huang et al., 2018) trains giant models with specialized pipeline parallelism library.
‡DAT denotes domain adaptive transfer learning (Ngiam et al., 2018). Here we only compare ImageNet-based transfer learning results.
Transfer accuracy and #params for NASNet (Zoph et al., 2018), Inception-v4 (Szegedy et al., 2017), ResNet-152 (He et al., 2016) are from (Kornblith et al., 2019).

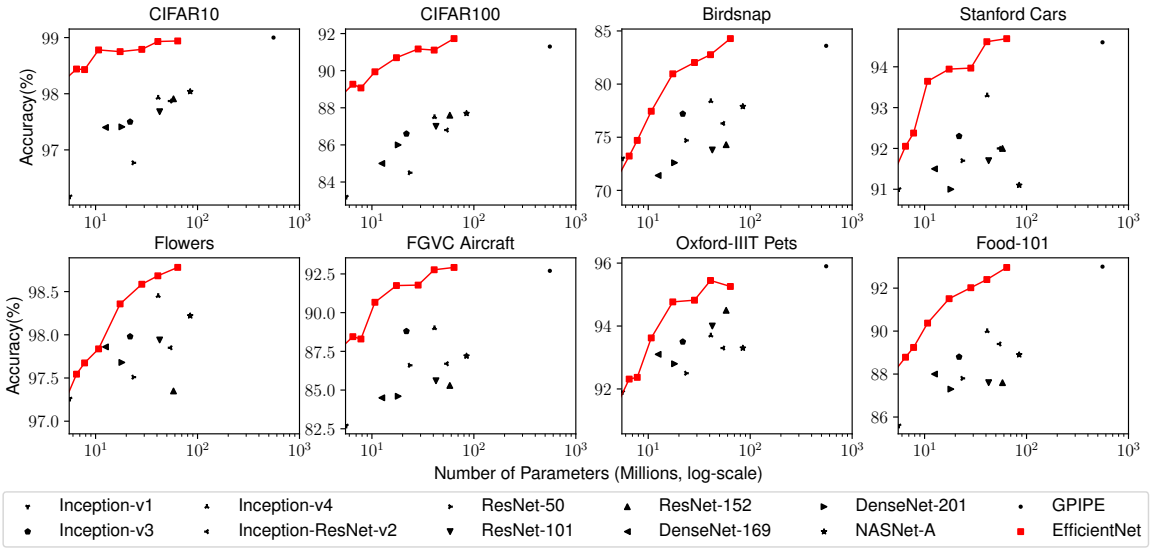


Figure 6. **Model Parameters vs. Transfer Learning Accuracy –** All models are pretrained on ImageNet and finetuned on new datasets.

(Ramachandran et al., 2018; Elfwing et al., 2018), fixed AutoAugment policy (Cubuk et al., 2019), and stochastic depth (Huang et al., 2016) with drop connect ratio 0.3. As commonly known that bigger models need more regularization, we linearly increase dropout (Srivastava et al., 2014) ratio from 0.2 for EfficientNet-B0 to 0.5 for EfficientNet-B7.

Model size ∝ Amt of Dropout

Table 2 shows the performance of all EfficientNet models that are scaled from the same baseline EfficientNet-B0. Our EfficientNet models generally use an order of magnitude fewer parameters and FLOPS than other ConvNets with similar accuracy. In particular, our EfficientNet-B7 achieves 84.4% top1 / 97.1% top-5 accuracy with 66M parameters and 37B FLOPS, being more accurate but **8.4x smaller** than the previous best GPipe (Huang et al., 2018).

Figure 1 and Figure 5 illustrates the parameters-accuracy and FLOPS-accuracy curve for representative ConvNets, where our scaled EfficientNet models achieve better accuracy with much fewer parameters and FLOPS than other ConvNets. Notably, our EfficientNet models are not only small, but also computational cheaper. For example, our EfficientNet-B3 achieves higher accuracy than ResNeXt-101 (Xie et al., 2017) using **18x fewer FLOPS**.

To validate the computational cost, we have also measured the inference latency on a real CPU as shown in Table 4, where we report average latency over 20 runs. Our EfficientNet-B1 runs **5.7x faster** than the widely used ResNet-152 (He et al., 2016), while EfficientNet-B7 runs about **6.1x faster** than GPipe (Huang et al., 2018), suggesting our EfficientNets are indeed fast on real hardware.

5.3. Transfer Learning Results for EfficientNet

We have also evaluated our EfficientNet on a list of commonly used transfer learning datasets, as shown in Table 6. We borrow the same training settings from (Kornblith et al., 2019) and (Huang et al., 2018), which take ImageNet pretrained checkpoints and finetune on new datasets.

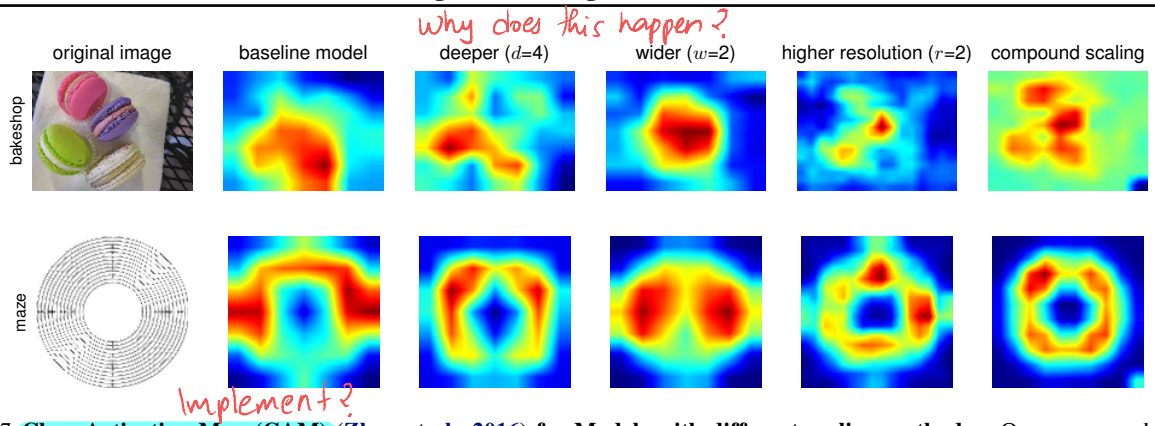


Figure 7. **Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods –** Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details.

Table 6. **Transfer Learning Datasets.**

Dataset	Train Size	Test Size	#Classes
CIFAR-10 (Krizhevsky & Hinton, 2009)	50,000	10,000	10
CIFAR-100 (Krizhevsky & Hinton, 2009)	50,000	10,000	100
Birdsnap (Berg et al., 2014)	47,386	2,443	500
Stanford Cars (Krause et al., 2013)	8,144	8,041	196
Flowers (Nilsback & Zisserman, 2008)	2,040	6,149	102
FGVC Aircraft (Maji et al., 2013)	6,667	3,333	100
Oxford-IIIT Pets (Parkhi et al., 2012)	3,680	3,369	37
Food-101 (Bossard et al., 2014)	75,750	25,250	101

Table 5 shows the transfer learning performance: (1) Compared to public available models, such as NASNet-A (Zoph et al., 2018) and Inception-v4 (Szegedy et al., 2017), our EfficientNet models achieve better accuracy with 4.7x average (up to 21x) parameter reduction. (2) Compared to state-of-the-art models, including DAT (Ngiam et al., 2018) that dynamically synthesizes training data and GPipe (Huang et al., 2018) that is trained with specialized pipeline parallelism, our EfficientNet models still surpass their accuracy in 5 out of 8 datasets, but using 9.6x fewer parameters

Figure 6 compares the accuracy-parameters curve for a variety of models. In general, our EfficientNets consistently achieve better accuracy with an order of magnitude fewer parameters than existing models, including ResNet (He et al., 2016), DenseNet (Huang et al., 2017), Inception (Szegedy et al., 2017), and NASNet (Zoph et al., 2018).

6. Discussion

To disentangle the contribution of our proposed scaling method from the EfficientNet architecture, Figure 8 compares the ImageNet performance of different scaling methods for the same EfficientNet-B0 baseline network. In general, all scaling methods improve accuracy with the cost of more FLOPS, but our compound scaling method can further improve accuracy, by up to 2.5%, than other single-dimension scaling methods, suggesting the importance of our proposed compound scaling.

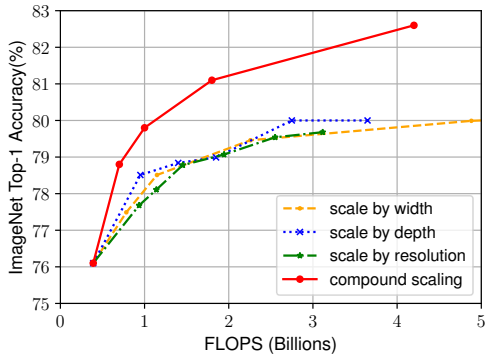


Figure 8. **Scaling Up EfficientNet-B0 with Different Methods.**

In order to further understand why our compound scaling method is better than others, Figure 7 compares the class activation map for a few representative models with different scaling methods. All these models are scaled from the same EfficientNet-B0 baseline with about 4x more FLOPS than the baseline. Images are randomly picked from ImageNet validation set. As shown in the figure, the model with compound scaling tends to focus on more relevant regions with more object details, while other models are either lack of object details or unable to capture all objects in the images.

7. Conclusion

In this paper, we systematically study ConvNet scaling and identify that carefully balancing network width, depth, and resolution is an important but missing piece, preventing us from better accuracy and efficiency. To address this issue, we propose a simple and highly effective compound scaling method, which enables us to easily scale up a baseline ConvNet to any target resource constraints in a more principled way, while maintaining model efficiency. Powered by this compound scaling method, we demonstrate that a mobile-size EfficientNet model can be scaled up very effectively, surpassing state-of-the-art accuracy with an order of magnitude fewer parameters and FLOPS, on both ImageNet and five commonly used transfer learning datasets.

Code

```python
class CNNBlock(nn.Module):
    def __init__(
        self, in_channels, out_channels, kernel_size, stride, padding, groups=1
    ):
        super(CNNBlock, self).__init__()
        self.cnn = nn.Conv2d(
            in_channels,
            out_channels,
            kernel_size,
            stride,
            padding,
            groups=groups,
            bias=False,
        )
        self.bn = nn.BatchNorm2d(out_channels)
        self.silu = nn.SiLU() # SiLU <-> Swish

    def forward(self, x):
        return self.silu(self.bn(self.cnn(x)))
```

all inps convolved to all outps

Conv
↓
BN
↓
Swish / SELU

```python
base_model = [
    # expand_ratio, channels, repeats, stride, kernel_size
    [1, 16, 1, 1, 3],
    [6, 24, 2, 2, 3],
    [6, 40, 2, 2, 5],
    [6, 80, 3, 2, 3],
    [6, 112, 3, 1, 5],
    [6, 192, 4, 2, 5],
    [6, 320, 1, 1, 3],
]

phi_values = {
    # tuple of: (phi_value, resolution, drop_rate)
    "b0": (0, 224, 0.2),  # alpha, beta, gamma, depth = alpha ** phi
    "b1": (0.5, 240, 0.2),
    "b2": (1, 260, 0.3),
    "b3": (2, 300, 0.3),
    "b4": (3, 380, 0.4),
    "b5": (4, 456, 0.4),
    "b6": (5, 528, 0.5),
    "b7": (6, 600, 0.5),
}
```

depth = α^{ϕ}

α β γ

```python
class InvertedResidualBlock(nn.Module):
    def __init__(
            self,
            in_channels,
            out_channels,
            kernel_size,
            stride,
            padding,
            expand_ratio,
            reduction=4, # squeeze excitation
            survival_prob=0.8, # for stochastic depth
    ):
        super(InvertedResidualBlock, self).__init__()
        self.survival_prob = 0.8
        self.use_residual = in_channels == out_channels and stride == 1
        hidden_dim = in_channels * expand_ratio
        self.expand = in_channels != hidden_dim
        reduced_dim = int(in_channels / reduction)

        if self.expand:
            self.expand_conv = CNNBlock(
                in_channels, hidden_dim, kernel_size=3, stride=1, padding=1,
            )

        self.conv = nn.Sequential(
            CNNBlock(
                hidden_dim, hidden_dim, kernel_size, stride, padding, groups=hidden_dim,
            ),
            SqueezeExcitation(hidden_dim, reduced_dim),
            nn.Conv2d(hidden_dim, out_channels, 1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

    def stochastic_depth(self, x):
        if not self.training:
            return x

        binary_tensor = torch.rand(x.shape[0], 1, 1, 1, device=x.device) < self.survival_prob
        return torch.div(x, self.survival_prob) * binary_tensor

    def forward(self, inputs):
        x = self.expand_conv(inputs) if self.expand else inputs

        if self.use_residual:
            return self.stochastic_depth(self.conv(x)) + inputs
        else:
            return self.conv(x)
```

```python
class SqueezeExcitation(nn.Module):
    def __init__(self, in_channels, reduced_dim):
        super(SqueezeExcitation, self).__init__()
        self.se = nn.Sequential(
            nn.AdaptiveAvgPool2d(1), # C x H x W -> C x 1 x 1
            nn.Conv2d(in_channels, reduced_dim, 1),
            nn.SiLU(),
            nn.Conv2d(reduced_dim, in_channels, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return x * self.se(x)

```

Ginsum 2, Apparently not oops

```python
class EfficientNet(nn.Module):
    def __init__(self, version, num_classes):
        super(EfficientNet, self).__init__()
        width_factor, depth_factor, dropout_rate = self.calculate_factors(version)
        last_channels = ceil(1280 * width_factor)
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.features = self.create_features(width_factor, depth_factor, last_channels)
        self.classifier = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(last_channels, num_classes),
        )

    def calculate_factors(self, version, alpha=1.2, beta=1.1):
        phi, res, drop_rate = phi_values[version]
        depth_factor = alpha ** phi
        width_factor = beta ** phi
        return width_factor, depth_factor, drop_rate

    def create_features(self, width_factor, depth_factor, last_channels):
        channels = int(32 * width_factor)
        features = [CNNBlock(3, channels, 3, stride=2, padding=1)]
        in_channels = channels

        for expand_ratio, channels, repeats, stride, kernel_size in base_model:
            out_channels = 4*ceil(int(channels*width_factor) / 4)
            layers_repeats = ceil(repeats * depth_factor)

            for layer in range(layers_repeats):
                features.append(
                    InvertedResidualBlock(
                        in_channels,
                        out_channels,
                        expand_ratio=expand_ratio,
                        stride = stride if layer == 0 else 1,
                        kernel_size=kernel_size,
                        padding=kernel_size//2, # if k=1:pad=0, k=3:pad=1, k=5:pad=2
                    )
                )
                in_channels = out_channels

        features.append(
            CNNBlock(in_channels, last_channels, kernel_size=1, stride=1, padding=0)
        )

        return nn.Sequential(*features)

    def forward(self, x):
        x = self.pool(self.features(x))
        return self.classifier(x.view(x.shape[0], -1))
```

Make sure it is div by 4