

EfficientNet

参考文献:

1. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks <https://arxiv.org/abs/1905.11946v3>
Source code is at <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>
2. 【论文解读】一文看懂EfficientnetB0~B7模型所有细节 <https://zhuanlan.zhihu.com/p/366738106>
3. Complete Architectural Details of all EfficientNet Models
<https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142>

EfficientNet 是由谷歌人工智能提出，他们试图提出一种如其名字所暗示的更有效的方法，同时改进现有的技术成果。一般来说，模型做得太宽，太深，或者分辨率很高。增加这些特征最初有助于模型的建立，但很快就会饱和，所建立的模型只是有更多的参数，因此效率不高。在EfficientNet中，这些参数都以一种更加有效的方式逐渐增加。

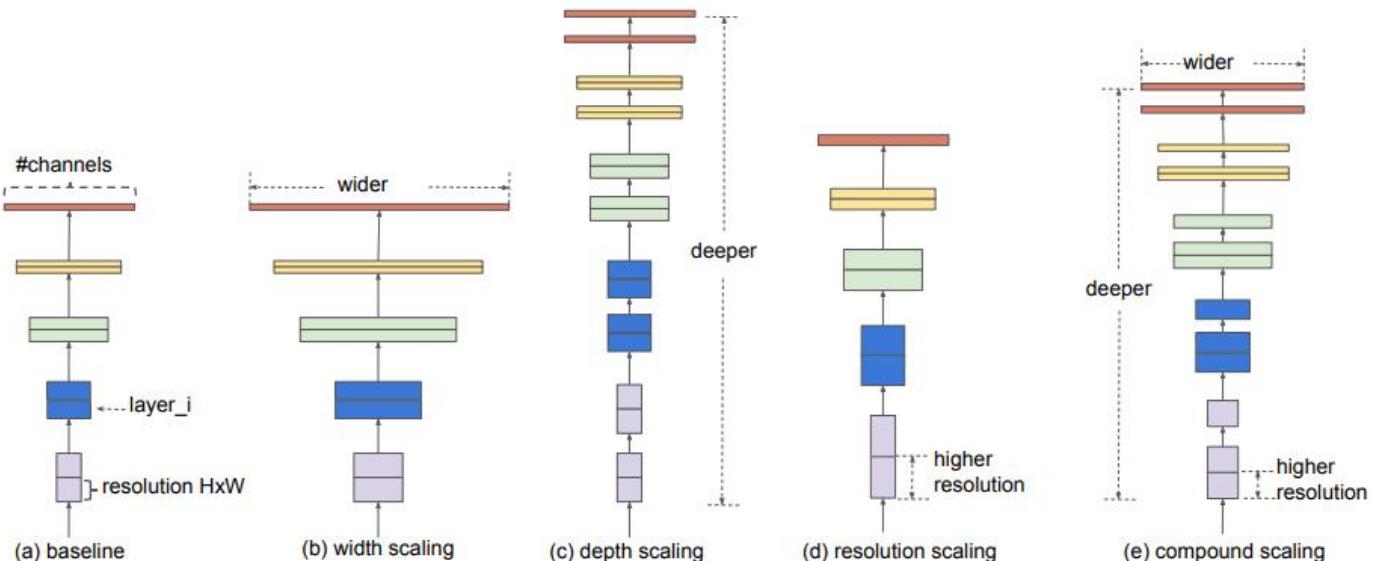
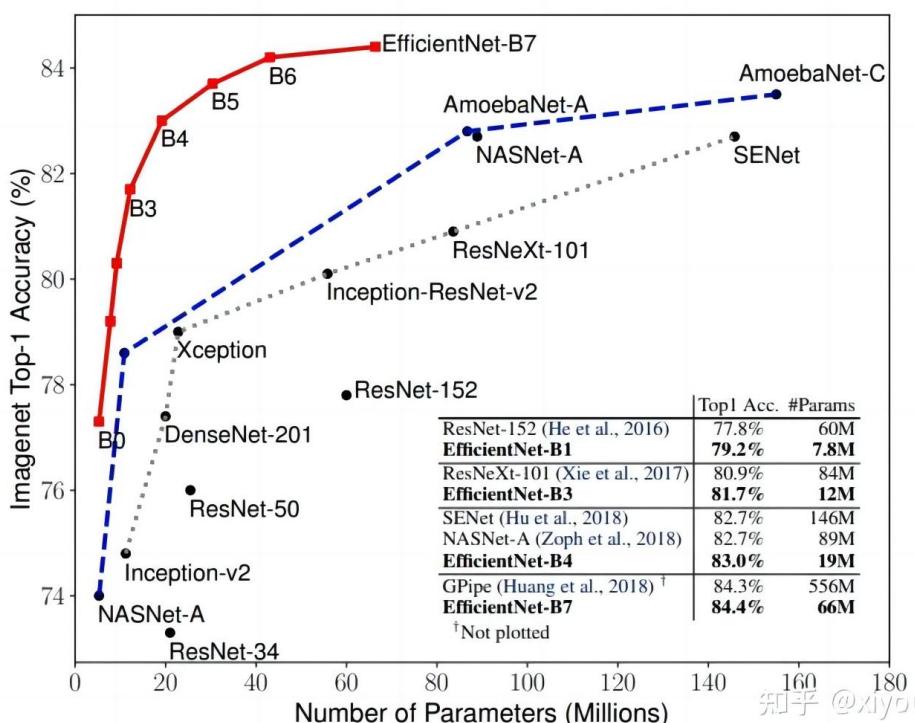


Figure 2. Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

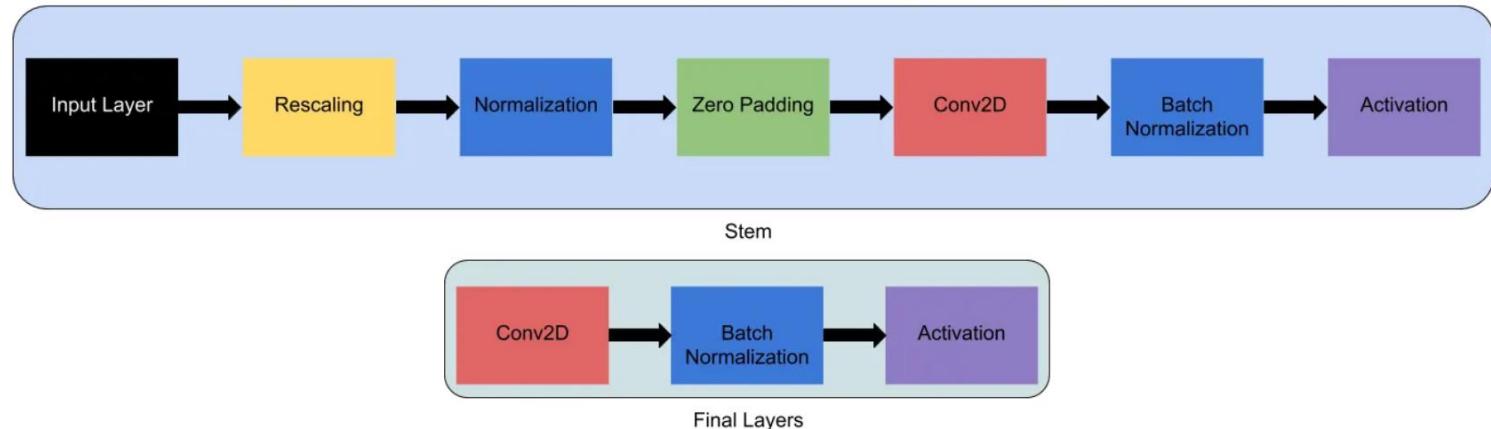
(a)是一个基线网络；(b)-(d)是传统的缩放方法，只增加网络宽度、深度或分辨率的一个维度。(e)是我们提出的复合缩放方法，以一个固定的比例统一缩放所有三个维度。

首先，让我们看看效果怎么样。由于参数数量大大减少，该系列的模型效率很高，也能提供更好的结果。



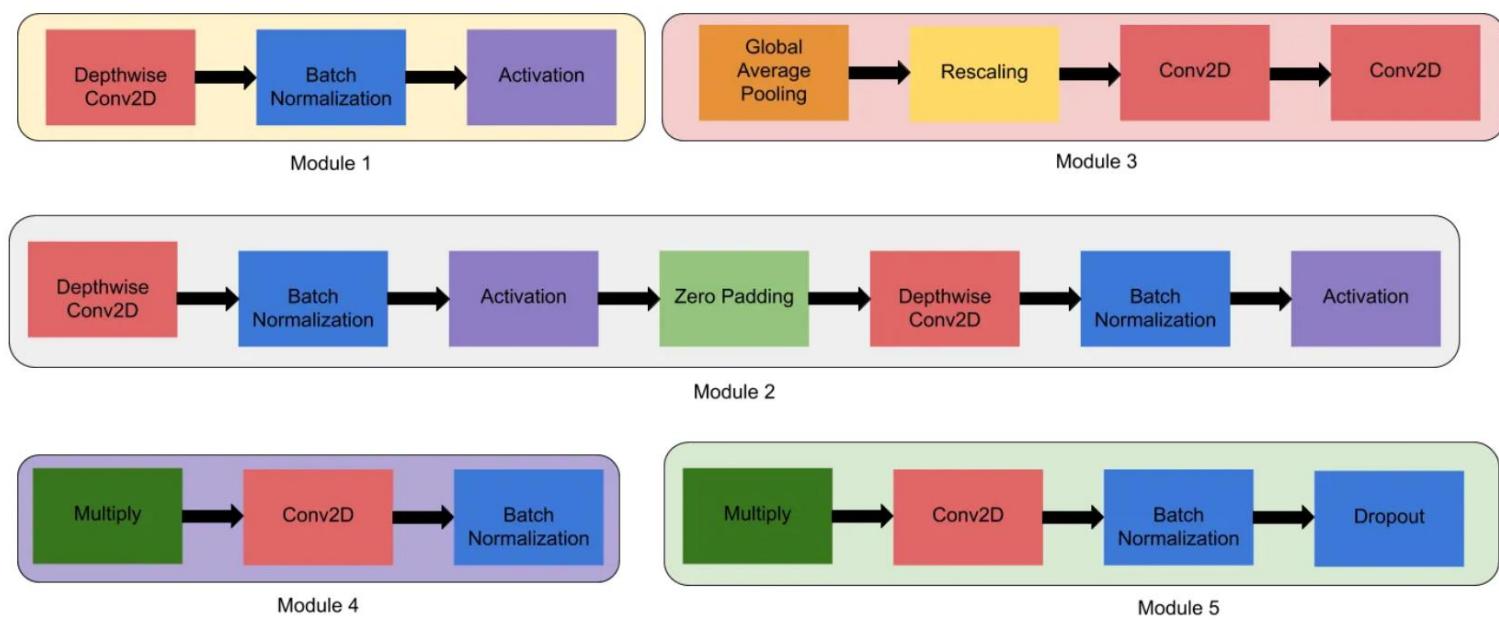
共有的结构

任何网络最关键的都是它的 stem，确定了之后才会进行后面的实验，这个结构（Stem）和最后一层（Final Layers）在所有八个模型（B0~B7）都是共同的。



EfficientNetB0~B7 的 Stem 和 Final Layers 之间都包含7个块(block)。这些块还有不同数量的子块(sub-block)，当我们从 EfficientNetB0 到 EfficientNetB7 时，子块的数量会增加。

如果计算 EfficientNet-B0 的总层数，总数为 237，而 EfficientNet-B7 的总层数为 813！不过不用担心，所有这些层都可以由下面的 5 个模块(module)和上面的 Stem 结构组成。



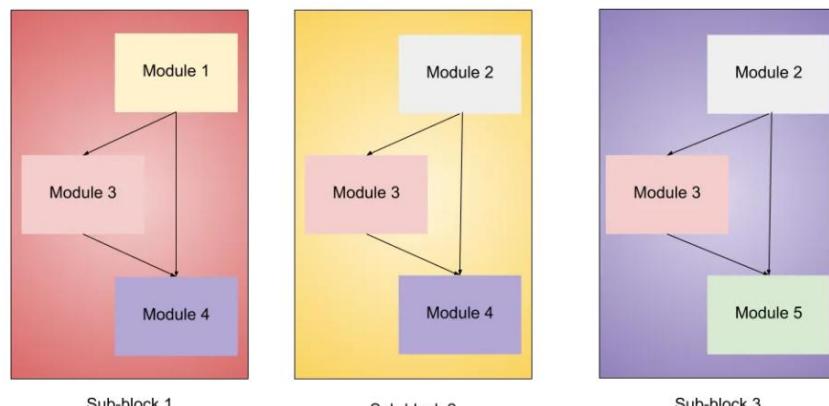
5 modules we will use to make the architecture.

- 模块1 -- 这被用作子块的起点。
- 模块2 -- 这被用作所有7个主块(block)的第一个子块的起点，除了第1个主块(block)。
- 模块3 -- 这是与所有子块的 skip connection。（原文：This is connected as a skip connection to all the sub-blocks.）
- 模块4 -- 用于组合第一个子块中的 skip connection。（原文：This is used for combining the skip connection in the first sub-blocks.）
- 模块5 -- 每个子区块都以 skip connection 的方式与其前一个子区块相连，并使用该模块将它们合并。

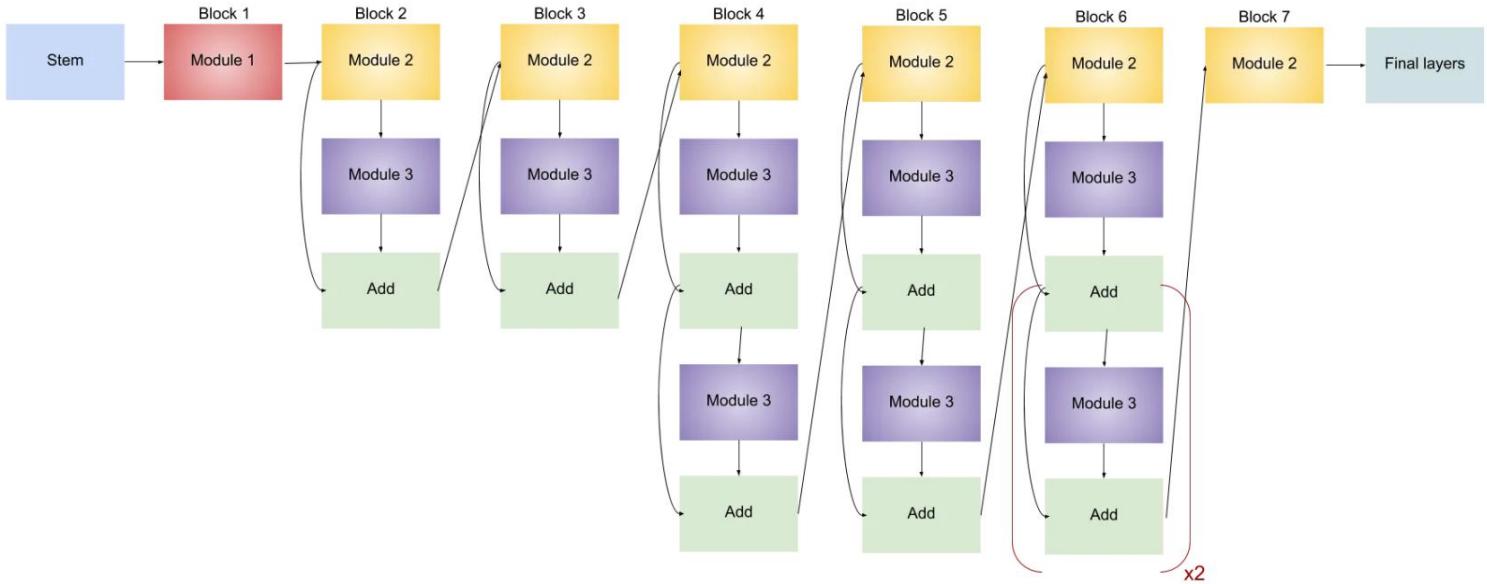
这些模块(module)被进一步组合成子块(sub-block)，这些子块将在块(block)中以某种方式使用。

- 第1子块 - 这只用作第一块中的第一个子块。
- 第2子块 - 这被用作所有其他块的第一个子块。
- 第3子块 - 用于所有块中除第一个子块外的任何子块。

到目前为止，我们已经指定了所有将被组合起来创建 EfficientNet 模型的内容，所以我们开始吧。

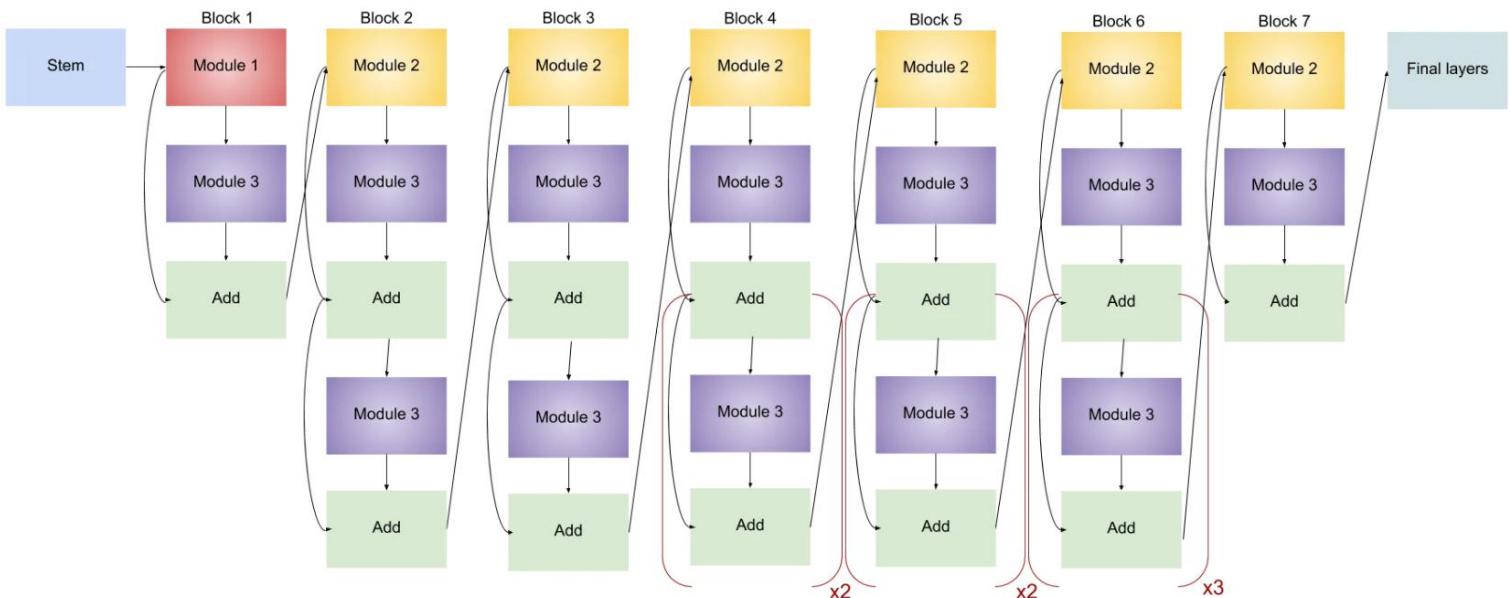


EfficientNet-B0



Architecture for EfficientNet-B0. (x2 means that modules inside the bracket are repeated twice)

EfficientNet-B1

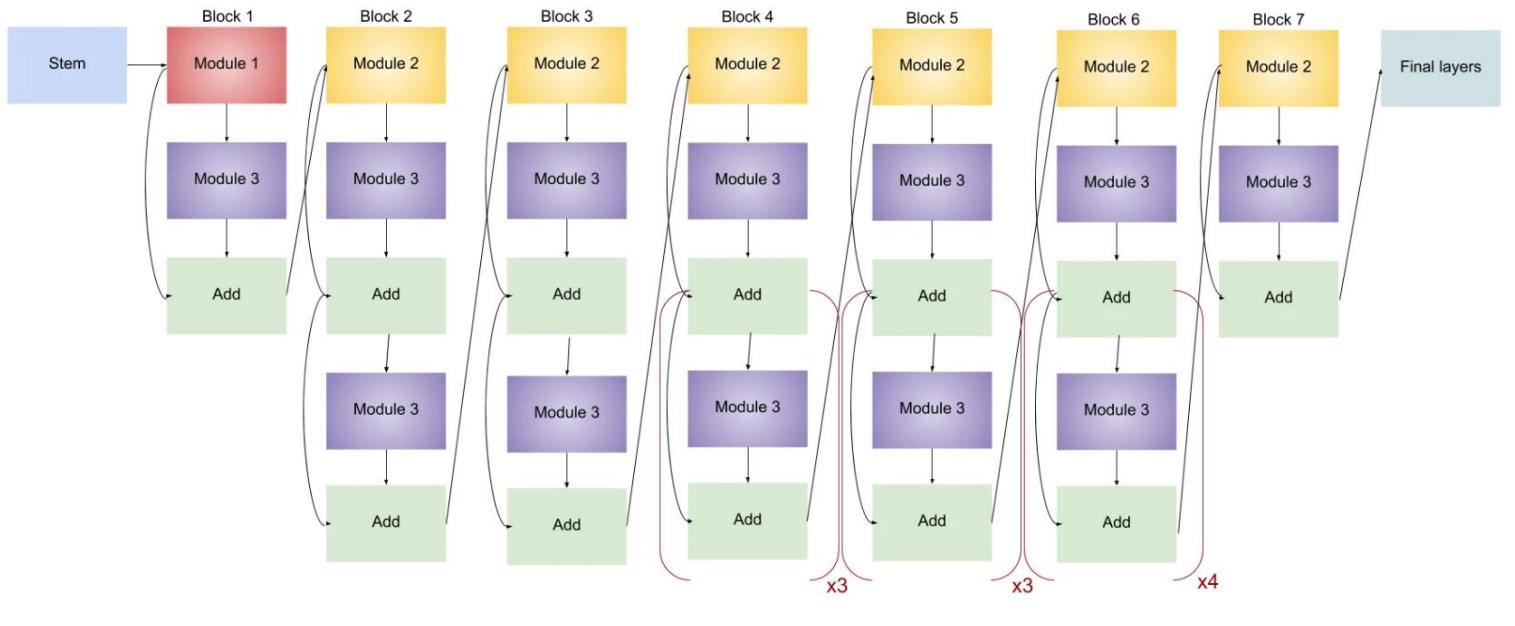


Architecture for EfficientNet-B1

EfficientNet-B2

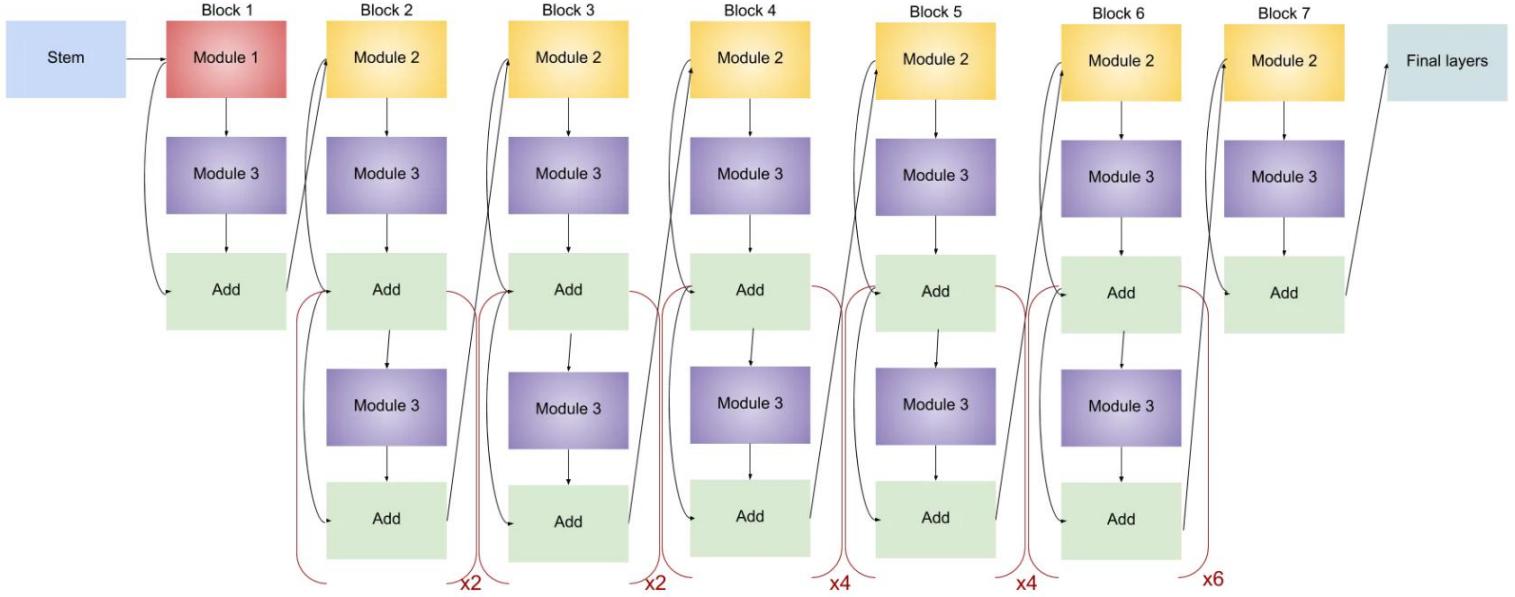
它的结构与 EfficientNet-B1 相同，它们之间唯一的区别是 feature maps（通道）的数量不同，从而增加了参数的数量。

EfficientNet-B3



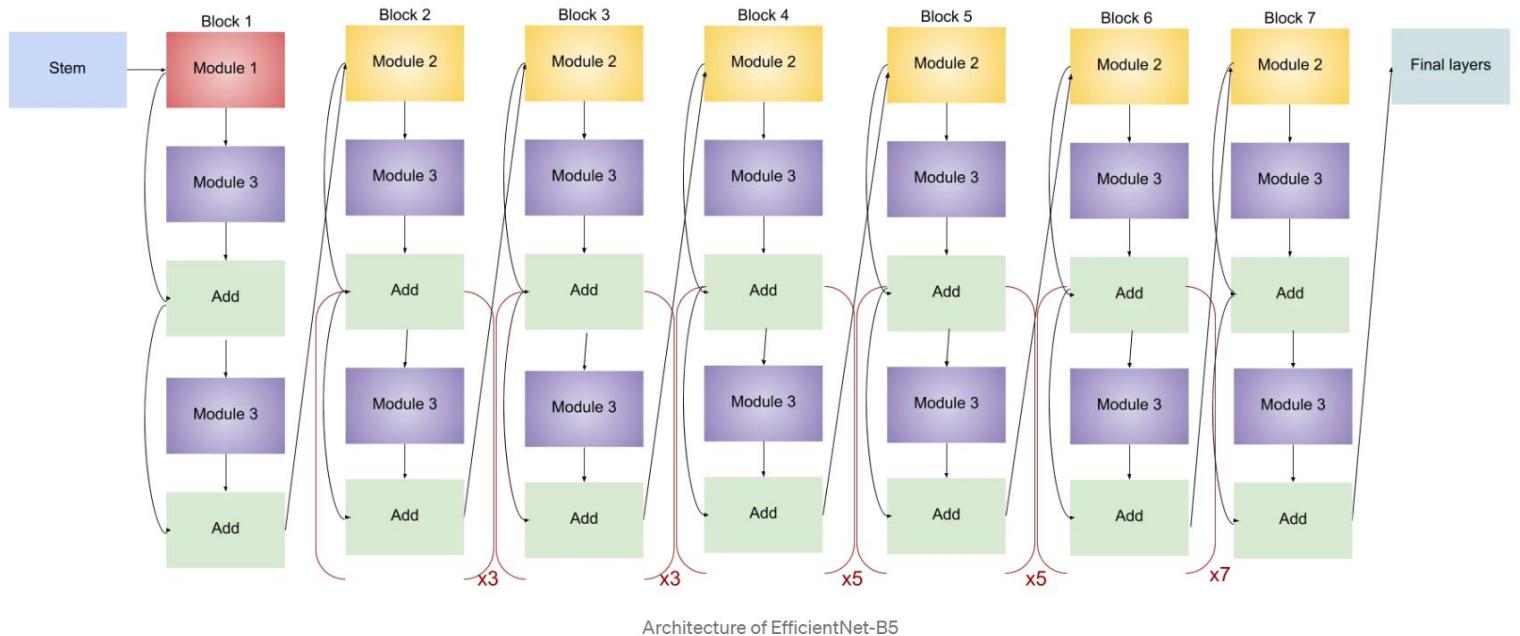
Architecture for EfficientNet-B3

EfficientNet-B4



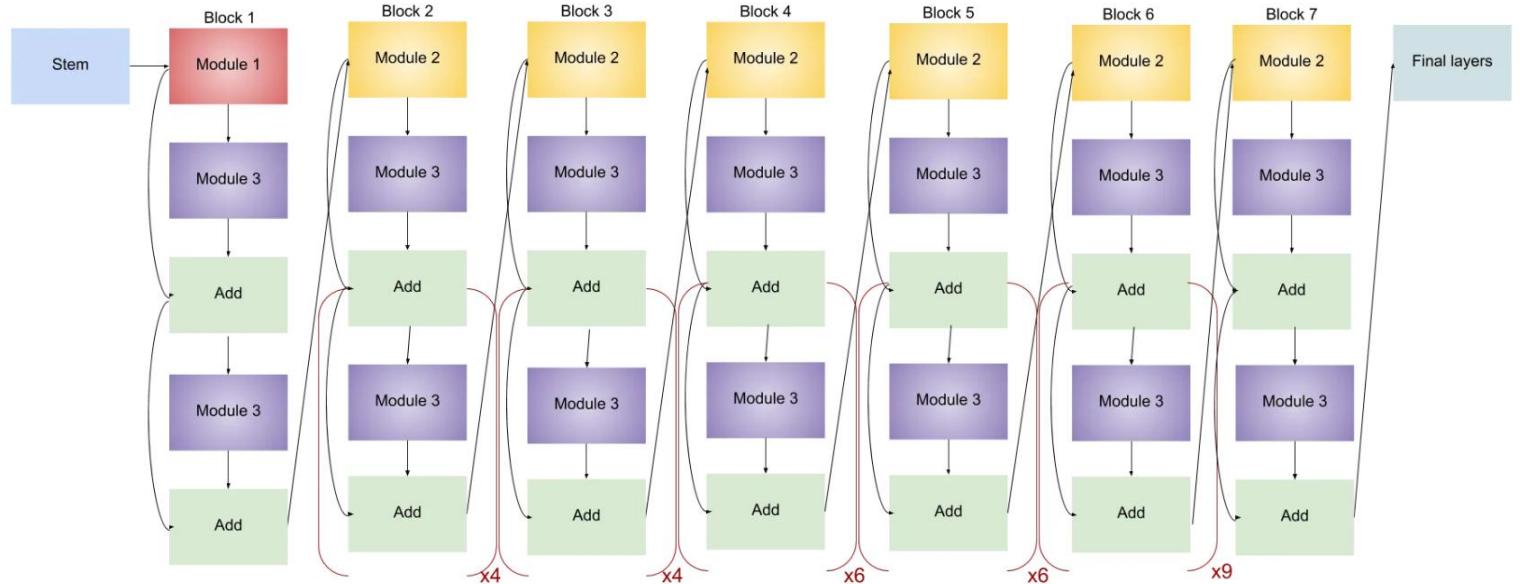
Architecture for EfficientNet-B4

EfficientNet-B5



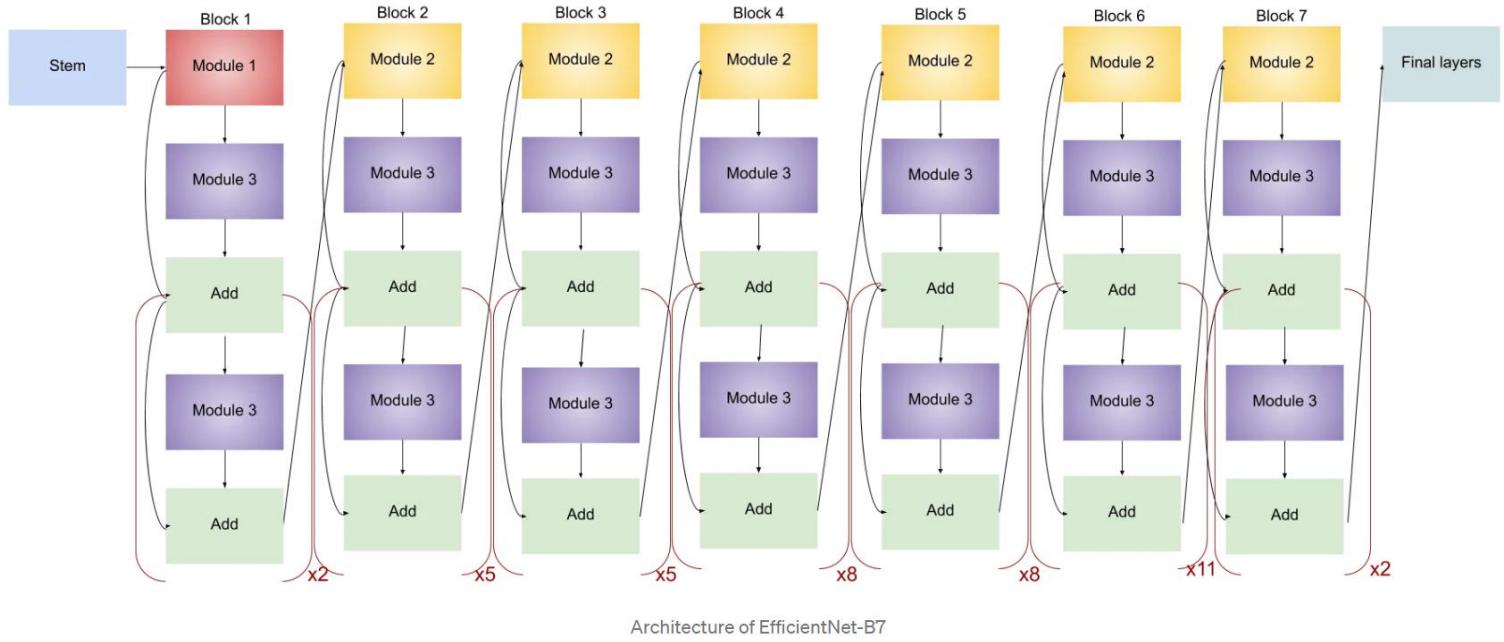
Architecture of EfficientNet-B5

EfficientNet-B6



Architecture of EfficientNet-B6

EfficientNet-B7



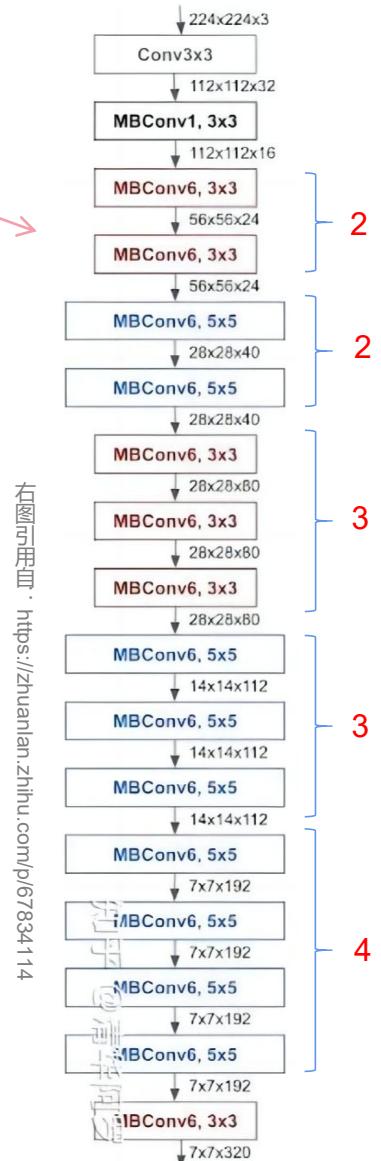
It's easy to see the difference among all the models and they gradually increased the number of sub-blocks. If you understood the architectures I will encourage you to take any model and print its summary and have a go through it to know it more thoroughly. The table shown below denotes the kernel size for convolution operations along with the resolution, channels, and layers in **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{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	MBCov1, k3x3	112 × 112	16	1
3	MBCov6, k3x3	112 × 112	24	2
4	MBCov6, k5x5	56 × 56	40	2
5	MBCov6, k3x3	28 × 28	80	3
6	MBCov6, k5x5	14 × 14	112	3
7	MBCov6, k5x5	14 × 14	192	4
8	MBCov6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

This table was included in the original paper. The resolution remains the same as for the whole family. I don't know about whether the kernel size changes or remains the same so if anyone knows leave a reply. The number of layers is already shown above in the figures. The number of channels varies and it is calculated from the information seen from each model's summary and is presented below.

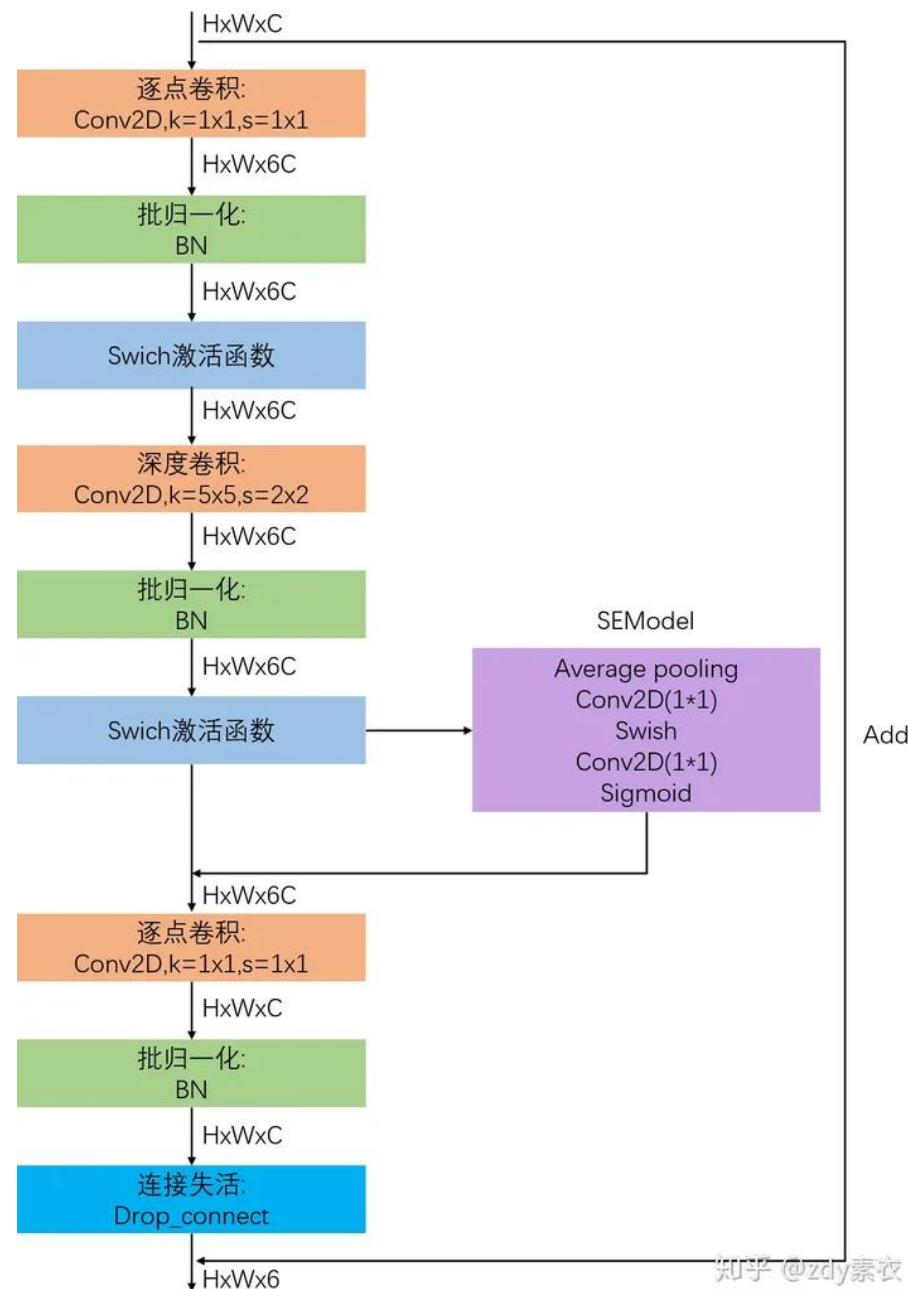
Stage	B1	B2	B3	B4	B5	B6	B7
1	32	32	40	48	48	56	64
2	16	16	24	24	24	32	32
3	24	24	32	32	40	40	48
4	40	48	48	56	64	72	80
5	80	88	96	112	128	144	160
6	112	120	136	160	176	200	224
7	192	208	232	272	304	344	384
8	320	352	384	448	512	576	640
9	1280	1408	1536	1792	2048	2304	2560



参考文献:

1. EfficientDet网络原理讲解与实现 <https://zhuanlan.zhihu.com/p/128521423>

具体结构：MBConv卷积块



参考文献：

1. EfficientNet-B0解读 <https://zhuanlan.zhihu.com/p/111115509>

本案例中我们选用EfficientNets系列中的基础网络模型 EfficientNet-B0。该网络的核心结构为移动翻转瓶颈卷积（mobile inverted bottleneck convolution, **MBConv**）模块，该模块还引入了压缩与激发网络（Squeeze-and-Excitation Network, **SENet**）的注意力思想，SENet在提出时也在ImageNet数据集上达到了当时最高的准确率。

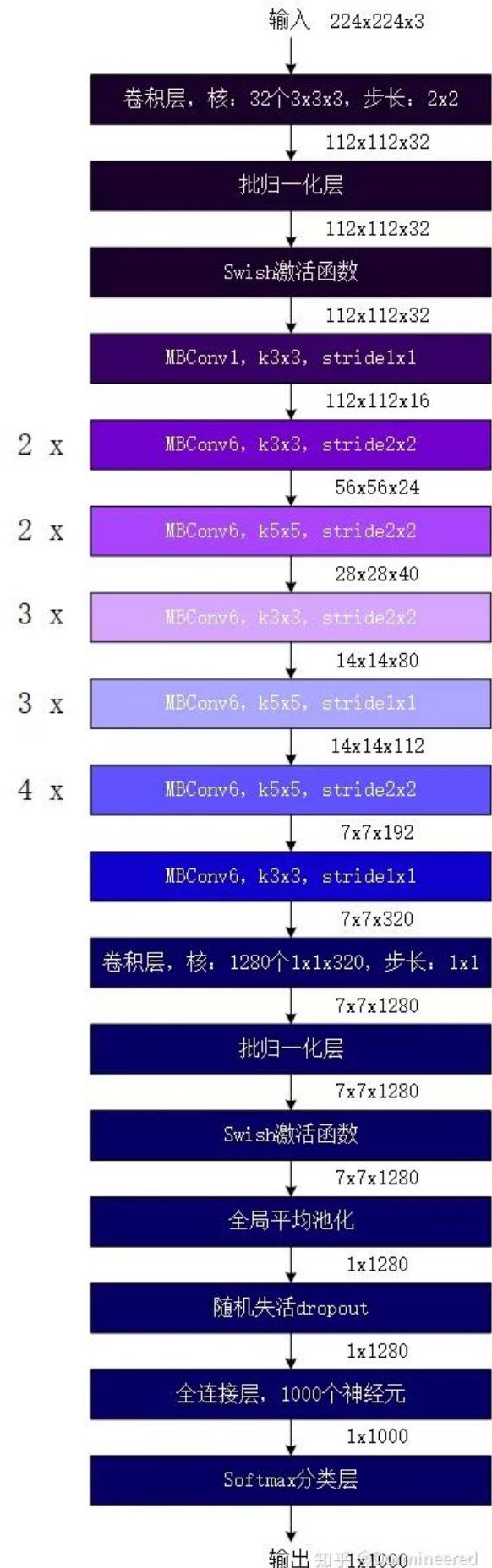
移动翻转瓶颈卷积也是通过神经网络架构搜索得到的，该模块结构与深度分离卷积（depthwise separable convolution）相似：

- 该**移动翻转瓶颈卷积**首先对输入进行 1×1 的逐点卷积并根据扩展比例(expand ratio) 改变输出通道维度（如扩展比例为3时，会将通道维度提升3倍。但如果扩展比例为1，则直接省略该 1×1 的逐点卷积和其之后批归一化和激活函数）。
- 接着进行 $k \times k$ 的**深度卷积**（depthwise convolution）。
- 如果要引入压缩与激发操作，该操作会在深度卷积后进行。再以 1×1 的逐点卷积结尾恢复原通道维度。
- 最后进行**连接失活**（drop connect）和输入的**跳跃连接**（skip connection），这一做法源于论文《Deep networks with stochastic depth》，它让模型具有了随机的深度，剪短了模型训练所需的时间，提升了模型性能（注意，在EfficientNets中，只有当相同的移动翻转瓶颈卷积重复出现时，才会进行连接失活和输入的跳跃连接，且还会将其中的深度卷积步长变为1），连接失活是一种类似于随机失活（dropout）的操作，并且在模块的开始和结束加入了恒等跳越。

注意该模块中的每一个卷积操作后都会进行批归一化，激活函数使用的是Swish激活函数。

移动翻转瓶颈卷积模块中的压缩与激发操作，简称**SE模块**，是一种基于注意力的特征图操作，SE模块首先对特征图进行压缩操作，在通道维度方向上进行全局平均池化操作（global average pooling），得到特征图通道维度方向的全局特征。然后对全局特征进行激发操作，使用激活比例（R，为浮点数）乘全局特征维数（C）个 1×1 的卷积对其进行卷积（原方法使用全连接层），学习各个通道间的关系，再通过sigmoid激活函数得到不同通道的权重，最后乘以原来的特征图得到最终特征。 本质上，SE模块是在通道维度上做（注意力）attention或者（门控制）gating操作，这种注意力机制让模型可以更加关注信息量最大的通道特征，而抑制那些不重要的通道特征。另外一点是SE模块是通用的，这意味着其可以嵌入到现有的其它网络架构中。注意在移动翻转瓶颈卷积模块中，与激活比例相乘的是移动翻转瓶颈卷积模块的输入通道维度，而不是模块中深度卷积后的输出通道维度。

EfficientNet-B0 的结构由16个**移动翻转瓶颈卷积模块**，2个卷积层，1个全局平均池化层和1个分类层构成。其结构如右图所示，图中不同的颜色代表了不同的阶段。（各阶段的较详细的介绍以及参数个数的计算见原文）



参考文献:

- 【论文复现】Swish-Activation (2017) https://blog.csdn.net/qq_38253797/article/details/118793355

Swish 激活函数

$$\text{Swish}(x) = x \cdot \sigma(\beta x) = x \cdot \frac{1}{1+e^{-\beta x}}$$

其中 β 是一个常数或者可训练的参数。

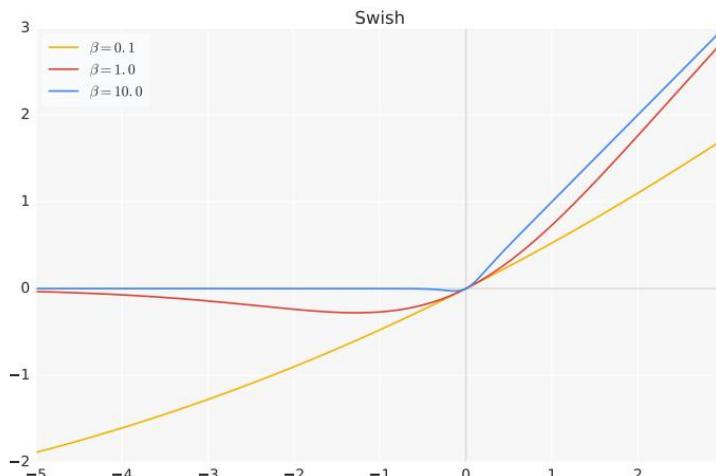


Figure 4: The Swish activation function.

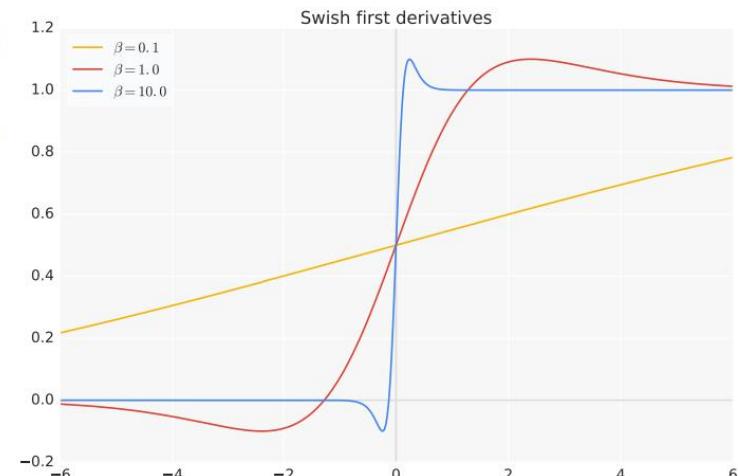


Figure 5: First derivatives of Swish. https://blog.csdn.net/qq_38253797

如上图为不同参数下的 $\text{Swish}(x)$ 函数，可以看到：

- 当 $\beta = 1$ 时，则 Swish 等效于 Elfwing 等人的 Sigmoid 加权线性单元 (SiL)，其被提议用于强化学习。
- 当 $\beta = 0$ 时，则 Swish 变为缩放线性函数 $x/2$ 。
- 当 $\beta \rightarrow \infty$ 时，sigmoid 组件变成接近 0-1 函数，因此 Swish 变得像 ReLU 一样的函数。

总结：由此可见 $\text{Swish}(x)$ 函数是介于线性函数和 ReLU 函数之间进行线性插值的函数，而参数 β 则控制插值的程度。

而且从图中也可以看出：

- 和 ReLU 函数一样，Swish 函数也是一个无上界（避免梯度饱和）有下界（产生更强的正则化效果）的函数。
- 和 ReLU 函数不一样的是，Swish 函数还是一个平滑（处处可导 更易训练）且非单调（很重要）的函数。

当 $x > 0$ 时，通过对函数求一阶导可知：

$$\begin{aligned} f'(x) &= \sigma(\beta x) + \beta x \cdot \sigma(\beta x)(1 - \sigma(\beta x)) \\ &= \sigma(\beta x) + \beta x \cdot \sigma(\beta x) - \beta x \cdot \sigma(\beta x)^2 \\ &= \beta x \cdot \sigma(\beta x) + \sigma(\beta x)(1 - \beta x \cdot \sigma(\beta x)) \\ &= \beta f(x) + \sigma(\beta x)(1 - \beta f(x)) \end{aligned}$$

https://blog.csdn.net/qq_38253797

当 $\beta = 1$ 时，Swish 函数具有同 ReLU 函数一样的性质：梯度保持性（平滑性），即当 $x > 0$ 时，处处可导（这点从上面的一阶导数函数图也可以看出来）。这可以证明直接取 $\beta = 1$ 这是有效的（实践中也通常是取 1，证明是有效果的），但是 β 的最佳效果并不一定是取 1。

当 $x < 0$ 时，由下图也可以看出函数的大部分都落在了区间 $(-5, 0)$ 之间，且呈峰形分布（不单调），这也可以表面 Swish 函数的非单调性是一个很重要的性质。

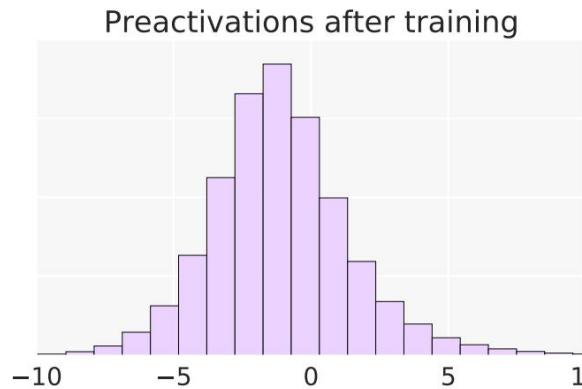


Figure 6: Preactivation distribution after training of Swish with $\beta = 1$ on ResNet-32.

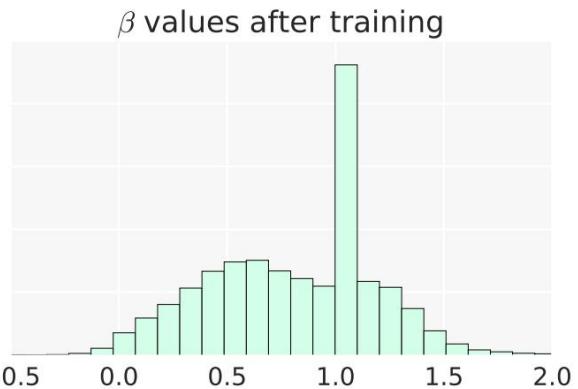


Figure 7: Distribution of trained β values of Swish on Mobile NASNet-A.

https://blog.csdn.net/qq_38253797

在实验过程（代码）中，我们一般都是直接令 $\beta = 1$ 的。

最后，整理下Swish的优点有：

- 无上界（避免过拟合）
- 有下界（产生更强的正则化效果）
- 平滑（处处可导，更容易训练）
- $x < 0$ 具有非单调性（对分布有重要意义，这点也是Swish和ReLU的最大区别）。

（Swish 函数的 pytorch 实现以及hard-swish函数）见原文。