Assignment 3: Exercise 1 & 2

Nima Taheri, Mahdieh Sajedi Pour

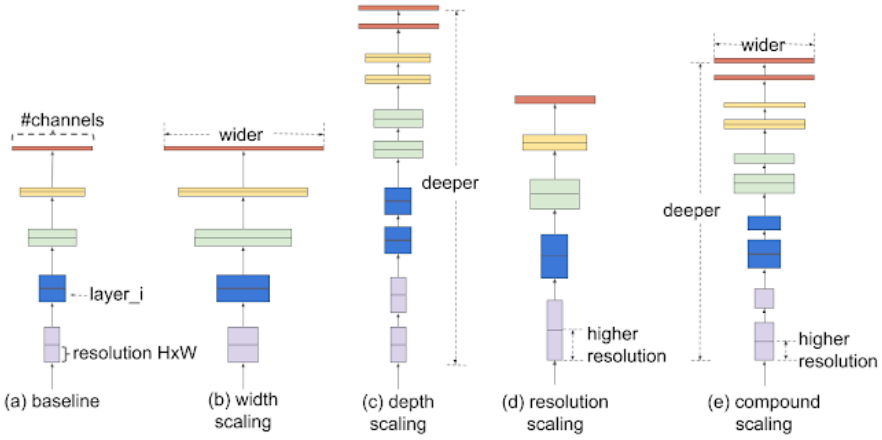
November 30, 2022

Exercise 1: EfficientNet

The reason for using this architecture is to use the available computing power optimally. We want to know if the amount of computing power decreases or increases, or if we want to train a network faster, how can we scale the network.

In convolutional network architectures, three methods are used to increase accuracy. These three methods include: increasing the network depth, the network width, and also increasing the input resolution. Increasing any of these features can improve the performance of the network. Here we examine the relationship between these three features. It is obvious that these features are directly related to each other, so with increasing resolution, there are more features to learn, so the network can have more depth.

So, effecientNet can search for the most efficient neural network according to the amount of computing power.



According to the picture above, there are three ways to scale up a convolutional network as follows:

The first method, increasing width: this method is used less. Width also means the number of channels in a network.

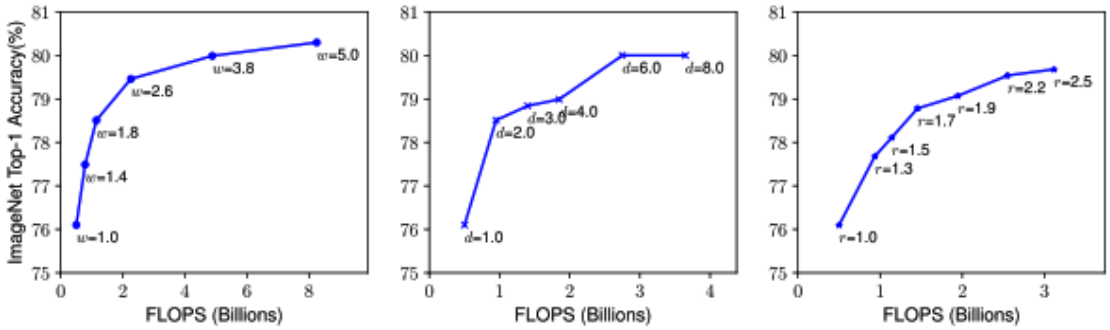
The second method, increasing depth: this method has been used the most in existing architectures so far. Increasing the depth means increasing the number of layers of a network.

The third method, increasing the resolution of the input image: this method has also been used sometimes in articles.

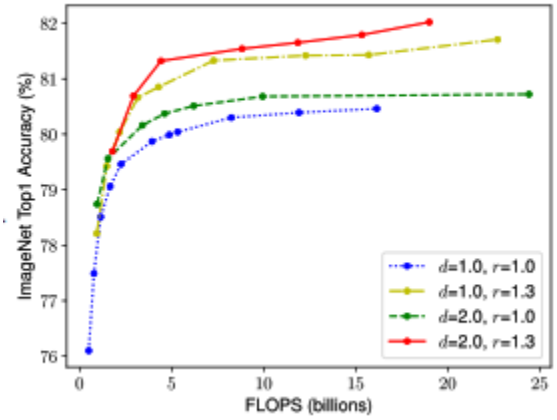
There is also a method based on increasing the depth, width and resolution to scale up a network, which can be seen in (e). It is called compound scaling and it needs to be discussed.

Compound Model Scaling

The goal here is not to present a new architecture, but we want to improve a model. First, we consider a baseline model and then we try to improve the baseline model by increasing the three dimensions of length, width and resolution. The problem is that in order to find the appropriate length, width and resolution, the space we have to search becomes very large, so we set a fixed coefficient for each layer to grow uniformly.



As we can see in the picture above, the increase of each of these dimensions increases the accuracy, but it stops when the accuracy reaches 80%.



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%**

But here we see that we can achieve more accuracy by combining these dimensions with each other. In other words, it can be stated that these dimensions are independent from each other. For example, if we increase the network width, we have more pixels that give us more information, so we can have a higher resolution as well. From another angle, we can say that

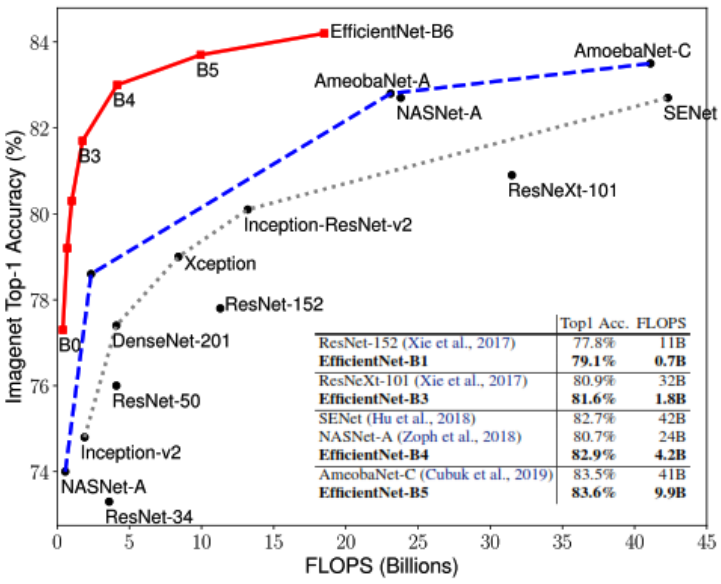
the greater the depth, the bigger the receptive field, which means that we have a higher-level feature for processing.

α, β, γ are constant coefficients determined by a small grid search. If we can increase the computational cost by a constant factor, we change the value of phi.

$$
\begin{aligned}
&Depth: && \alpha^{\varphi} \\
&Width: && \beta^{\varphi} \\
&Resolution: && \gamma^{\varphi} \\
&S.t: && \alpha * \beta^2 * \gamma^2 = 2 \\
&and && \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}
$$

Phi can also be set by the user depending on the computing power they have. The limitation mentioned in the last line means that the number of FLOPS can be increased by 2^{φ}.
The architecture that is considered as the baseline is also very important, because EfficientNet does not change the baseline architecture and only scales it. In addition to that, for example, if Alexnet is used as a baseline, and after changing the scale, the width, depth, and resolution are compared on the same Alexnet and not compared to another network, for example, REsnet.
Starting from the baseline EfficientNet-B0, we obtain other versions (B1-B7) by applying compound scaling method to scaleup in two steps:

1) First, we consider phi to be constant and equal to 1 ($\varphi = 1$), assume that we double the computing power and find the appropriate alpha, beta and gamma using a grid search. Considering the first mentioned condition.
2) In the second step, we consider alpha, beta and gamma constant and scale up B0 by considering different phis.



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

We can simply see that the accuracy of EfficientNet is more than other models.
To prove that the obtained results are not just because of the architecture, and EfficientNet has really improved the models, this method has been implemented on several models and similar results have been obtained, which can be seen in the table below. As applying the efficient net method on different models improved the accuracy, it can be generalized to all models.

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).

Latency is measured with batch size 1 on a single core of Intel Xeon CPU E5-2690.

	Acc. @ Latency
GPipe	84.3% @ 19.0s
EfficientNet-B7	84.4% @ 3.1s
Speedup	**6.1x**

It can be seen from the table that EfficientNet in all its versions has more accuracy, less parameter and less FLOPS than other models. EfficientNet-B7 has reached 84.3% accuracy and it is 8.4x smaller and 6.1x faster.

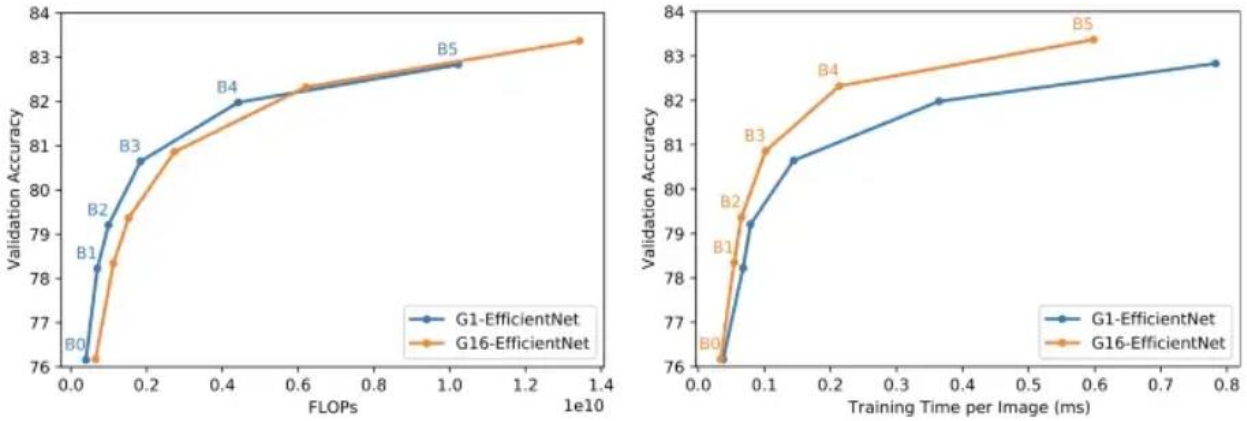
Innovations

There are three main ways to modify the EfficientNet and make it more efficient.

1) Group Convolutions
   Depth wise convolutions are well known for being FLOP and parameter efficient and have been utilized in many CNNs. But there is a challenge and that is hardware not being always fully utilized, resulting in “wasted” cycles. With a simple but significant

change in the MBConv block, it is possible to make better use of the IPU hardware and obtain a more efficient version of EfficientNet (G16-EfficientNet). For this, we increase the size of convolution groups from 1 to 16. Then, to compensate for the increase in FLOPs and parameters, and to address memory issues, we reduce the expansion ratio to 4. Although these changes were aimed at improving throughput, we find that this modification leads to a significant improvement in practical efficiency.

Picture below shows what we said.

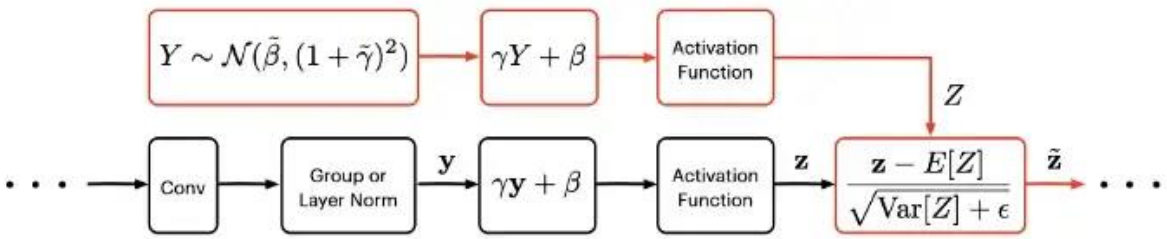


2) Proxy Normalized Activations

We use a batch-independent normalization method, Proxy Norm. This method builds on Group Normalization. There is an issue with Group Norm and Layer Norm and that's the activations. They can become channel-wise denormalized. This gets worse with depth because denormalization gets accentuated at every layer. But we can simply avoid this by reducing the size of groups in Group Norm. This will alter the expressivity and penalize performance.

Proxy Norm allows us to maximize the group size and to preserve expressivity without the issue of channel-wise denormalization.

In the picture below, convolution block with additional Proxy-Normalized Activation operations is shown in red.

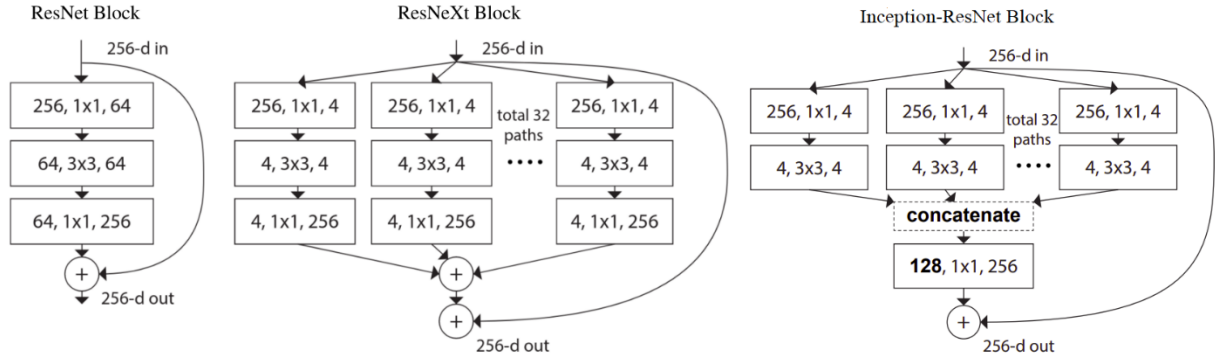


3) Reduced Resolution Training

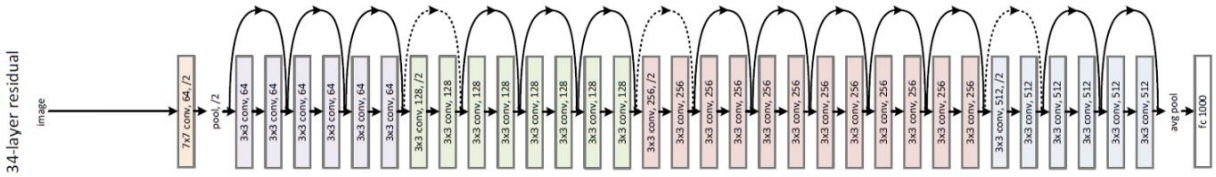
   We should choose the training resolution that maximizes efficiency. Comparing two different resolutions, “native” resolution and approximately half the pixel count, can help us to understand how larger images impact efficiency. Obtained result shows that training with half-size images yields considerable theoretical and practical efficiency improvements.

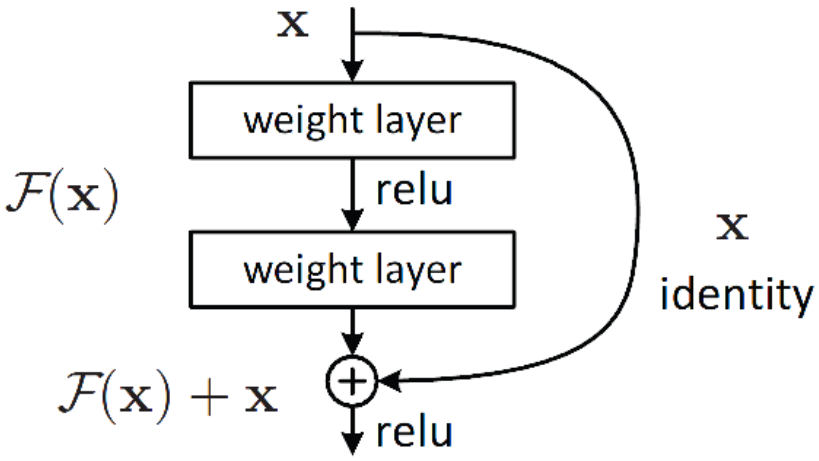
Using these three methods together, we can achieve 7x improvement in practical training efficiency and 3.6x improvement in practical inference efficiency on IPU.

Exercise 2: ResNext vs. Inception-ResNet



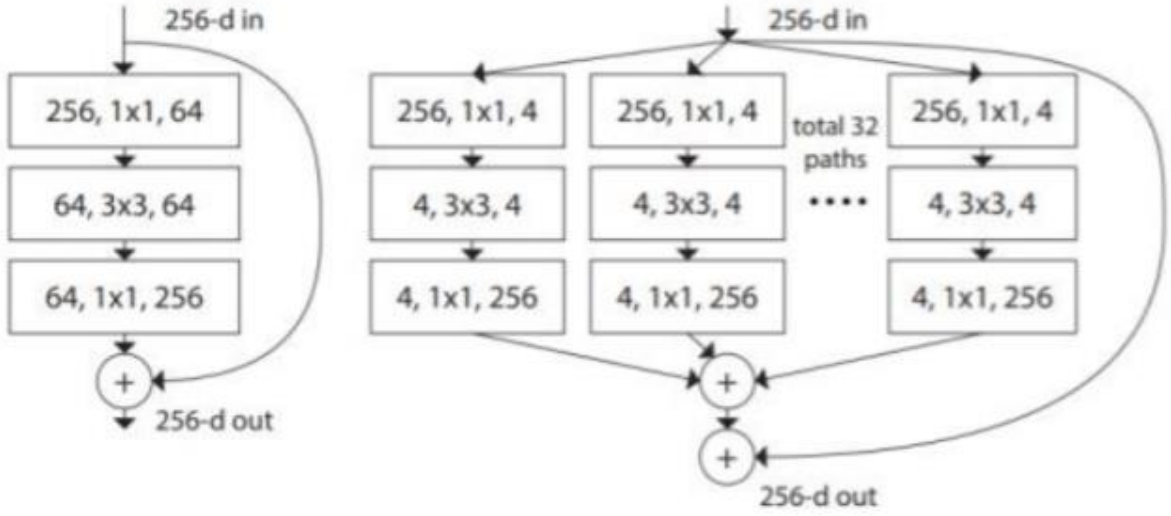
A breif explanation of ResNet





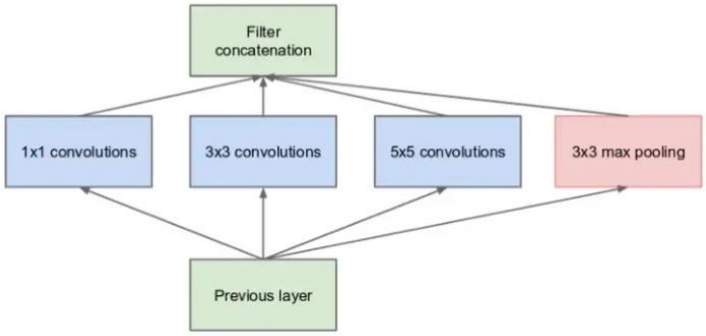
Residual neural network (ResNet) is an artificial neural network (ANN). This is the first very deep feedforward neural network with hundreds of layers, much deeper than previous neural networks. Skip connections or shortcuts are used to jump over some layers. There are two main reasons for adding skip connections: firstly, to avoid vanishing gradients, thus leading to easier optimization of networks, secondly, to mitigate the Degradation (accuracy saturation) problem. where adding more layers to a suitable deep model results in a higher training error.

ResNext



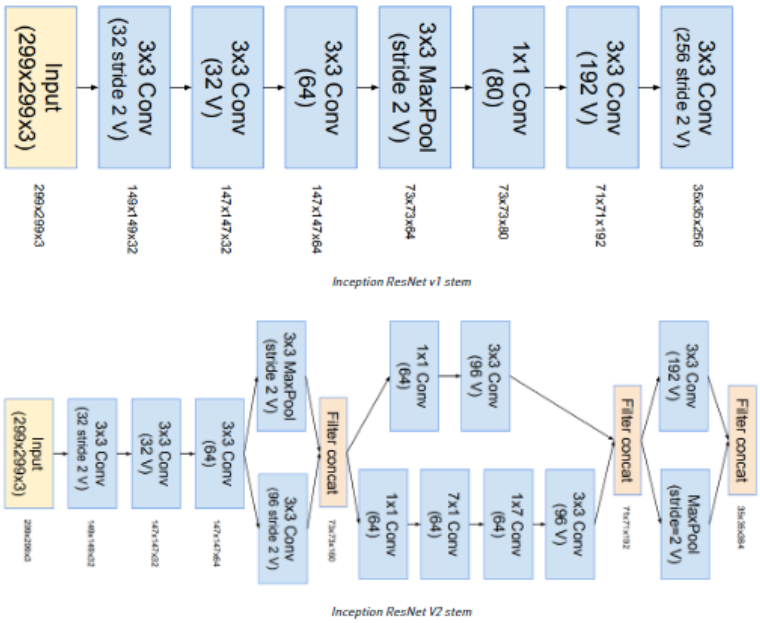
To improve ResNet we divide each block into parallel paths and change dimensions so that the number of parameters does not change. This will lead to simpler calculations.
For each path, Conv1×1–Conv3×3–Conv1×1 are done at each convolution path. This is the bottleneck design in ResNet block. The internal dimension for each path is denoted as d (d=4). The number of paths is the cardinality C (C=32). If we sum up the dimension of each Conv3×3 (i.e., d×C=4×32), it is also the dimensions of 128. The dimension is increased directly from 4 to 256, and then added together, and also added with the skip connection path.
ResNeXt has only 50% complexity compared to ResNet but achieved better accuracy.
If we compare ResNeXt-50 to ResNet-50, it has a validation error equal to 22.2%, but ResNet's validation error is 23.9% and is 1.7% higher.

A brief explanation of Inception networks



We know that the computation cost of deep neural networks is high. So, we limit the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions to have a cheaper computation. 1x1 layer is introduced after max pooling and is used for depth reduction. The results from the four parallel operations are then concatenated depth-wise to form the Filter Concatenation block.
Although we are adding an extra step, but it's cheaper than 5x5. Reducing the number of input channels also helps.

Inception-ResNet



Inception ResNet is a combination of Inception with ResNets. There are two versions of Inception-ResNet: ResNet V1 and ResNet V2.

Reduction Blocks have the same architecture in both versions. But they have different stems. There is also a difference in their hyperparameters in training. Inception-ResNet v2 is more accurate, but has a higher computational cost than Inception-ResNet v1.
We need to scale up the dimensionality of the filter bank so it will match the depth of input to next layer. So, after each Inception block there is a 1×1 convolution with no activation. We call that filter expansion.

Comparison

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (**64** × **4d**)	20.4	5.3	**19.1**	**4.4**

Table above shows results of single-crop testing on the ImageNet validation set. We can see that ResNext has a better resualt in practice.
Inception-ResNet needs to increase the dimension from 4 to 128 then to 256, but for designing each path, ResNeXt requires minimal extra effort.
We know that Inception focuses on computational cost and ResNet focuses on computational accuracy. So, it is understood that Inception-ResNet has improved both computational cost and computational accuracy.
As we mentioned before ResNeXt with only 50% complexity compared to ResNet has achieved better accuracy. So, both of them have better accuracy and less computational cost compared to their base models. In both of them the neurons at one path is not connected to the neurons at other paths.