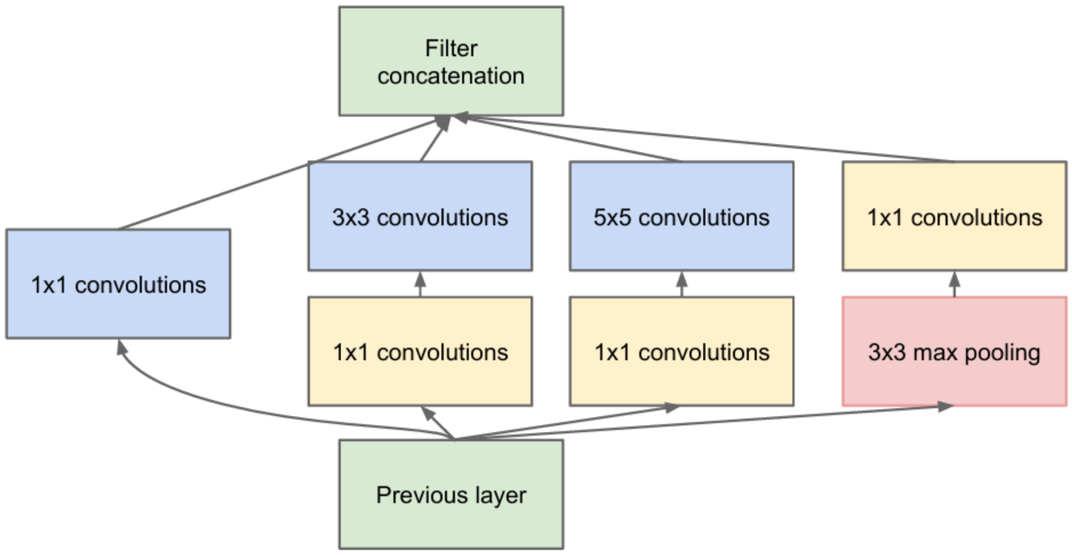
CNN

InceptionNet

Inception Net focuses on solving the question: What is the most suitable filter size instead of building a multi-kernel CNN with high cost.

1. Inception Module

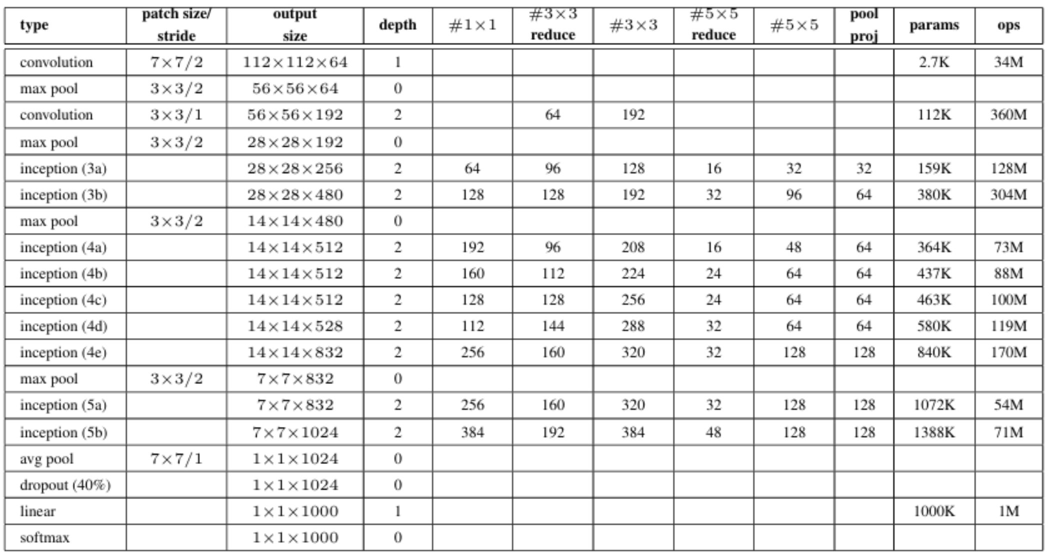


(b) Inception module with dimension reductions

- Inception Module includes 4 parallel branches with filter sizes of 1×1, 3×3, 5×5 respectively, helping to extract diverse features on perception areas of different sizes.
- 1×1 convolutions on top of 3×3, 5×5 convolutions to reduce channel depth and reduce the number of parameters.
- In the fourth branch, 3×3 max pooling is used to reduce the dimensionality of the data and then 1×1 convolutions are used to adjust the number of channels accordingly.
- One adjusts the padding and stride so that the branches have the same W and H and then concatenates them together so that the input size equals the output size.

2. Inception Network

In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion.

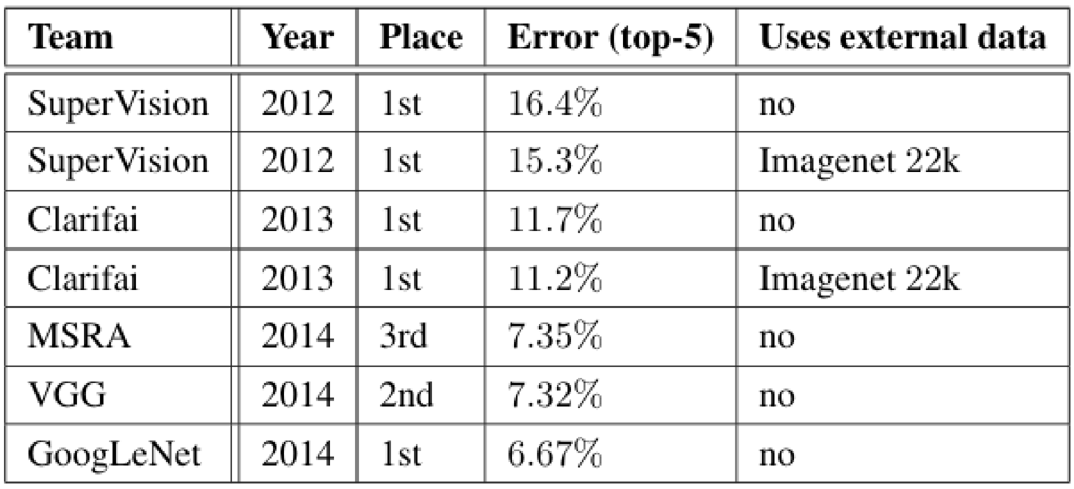


type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

? Auxiliary classifier

3. Result



Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance

The final submission of GoogLeNet (Inception - v1) in the ILSVRC2014 challenge obtains a top-5 error of 6.67% on both the validation and testing data, ranking the first among other participants. Beat the previous best model VGG.

ResNet

As network depth increases, training becomes more difficult due to challenges like vanishing/exploding gradients and degradation of accuracy. The
Residual Network (ResNet), introduced by He et al. in 2015, was a revolutionary approach to solving these challenges.

1. Key concept (Residual learning)

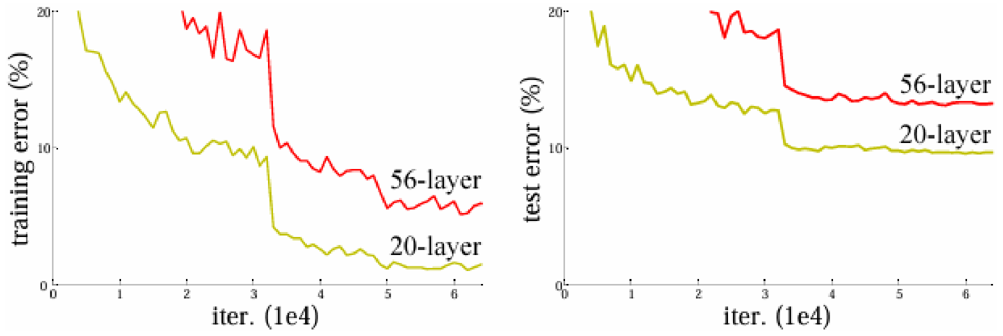


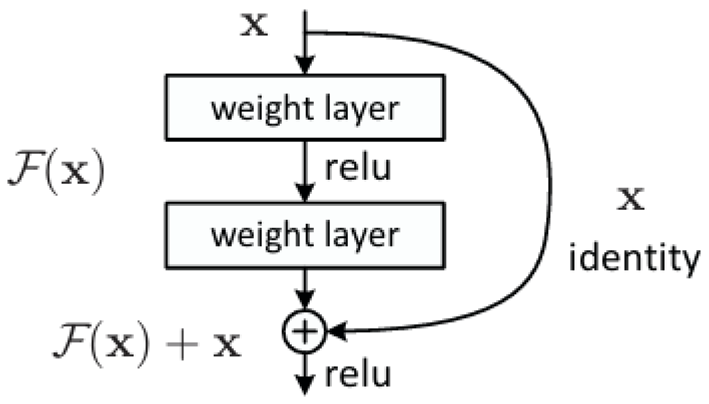
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Instead of fitting a desired mapping H(x), ResNet reformulates the problem to fit a residual mapping **F(x) = H(x)−x**. The output becomes:

H(x) = F(x)+x

This reformulation is driven by the degradation problem due to deep networks leading to vanishing gradients and information loss.

2. Residual Block



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

Residual block is a convolution block has two or more convolutional layers (with batch normalization and ReLU activations).

- **Input**: The input is passed through two or more convolutional layers (with batch normalization and ReLU activations).
- **Shortcut Connection**: The input is directly added to the output of the convolutional layers, help keep information from being lost.

- **Output**: The combined result **y = F(x,W)+x** is passed through the next layers of the network.

Types of residual block:

- **Identity Block**: The dimensions of the input and output are the same, so no additional operations are required for the shortcut connection.
- **Projection Block**: When the dimensions of the input and output differ (e.g., due to down-sampling), a 1×1 convolution is applied in the shortcut to match the dimensions.

3. Network

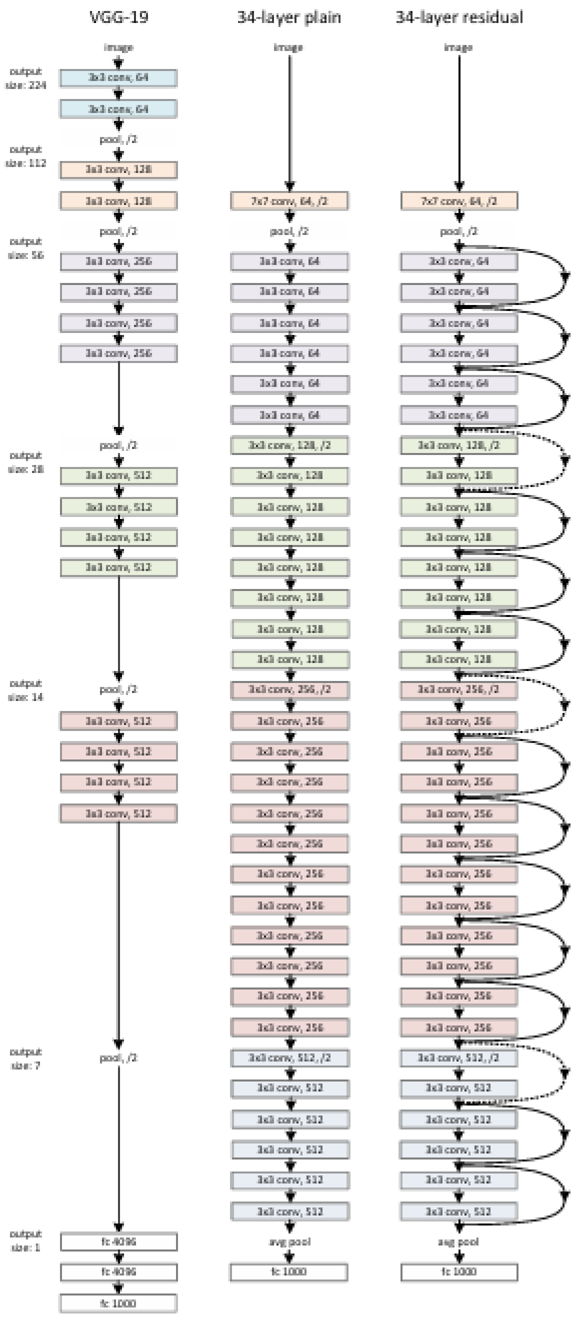
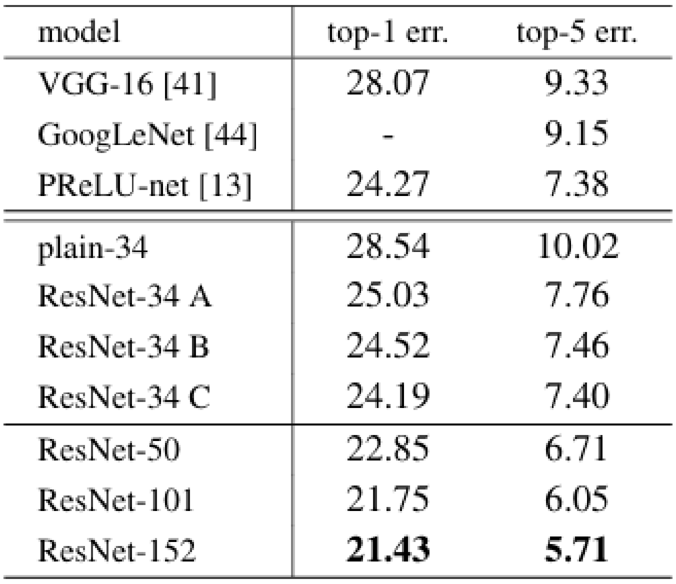


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Although ResNet inherits the block architecture from GoogleNet, it is much easier to abstract and implement because its basic architecture consists of only convolutional and deterministic blocks.

4. Result



model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

DenseNet

While deeper networks have shown remarkable success in solving complex tasks, they often suffer from challenges such as **vanishing gradients**, **redundant feature learning**, and increased parameter overhead. To overcome these, **DenseNet** (introduced by Huang et al. in 2017) proposes a novel architecture that encourages feature reuse and efficient parameter usage through **dense connectivity**.

1. Key concept:

The idea of DenseNet is based on **Taylor expansion**. DenseNet decompose the function into a definite function and a nonlinear function:

$$\mathbf{f(x) = x+g(x)}$$

The difference between DenseNet and ResNet is that instead of adding inputs to f(x), it concatenates outputs of increasing complexity together.

$$\mathbf{x} \rightarrow f_1(\mathbf{x})\ \mathbf{x} \rightarrow f_2(\mathbf{x}, f_1(\mathbf{x})) \ldots \mathbf{x} \rightarrow f_4(\mathbf{x}, f_3(\mathbf{x}, f_2(\mathbf{x}, f_1(\mathbf{x}))))$$

This structure encourages:

- **Feature Reuse**: Layers have direct access to rich features from earlier layers, reducing redundancy.
- **Reduced Parameters**: DenseNet is parameter-efficient because it avoids learning redundant filters.

2. Dense Block

The **Dense Block** is the fundamental building block of DenseNet. It comprises multiple convolutional layers, with each layer concatenating its output with the outputs of all preceding layers.

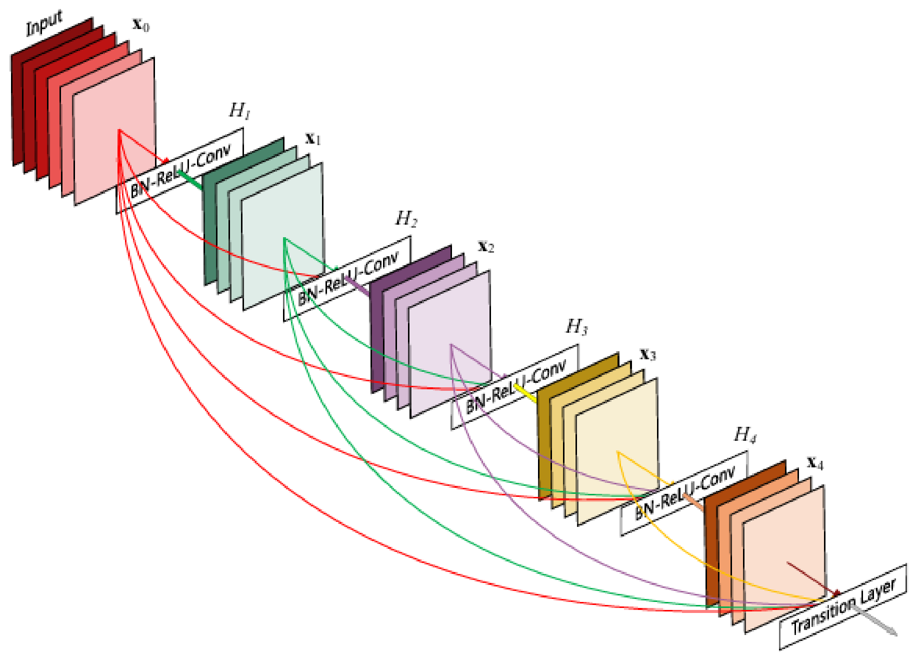


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

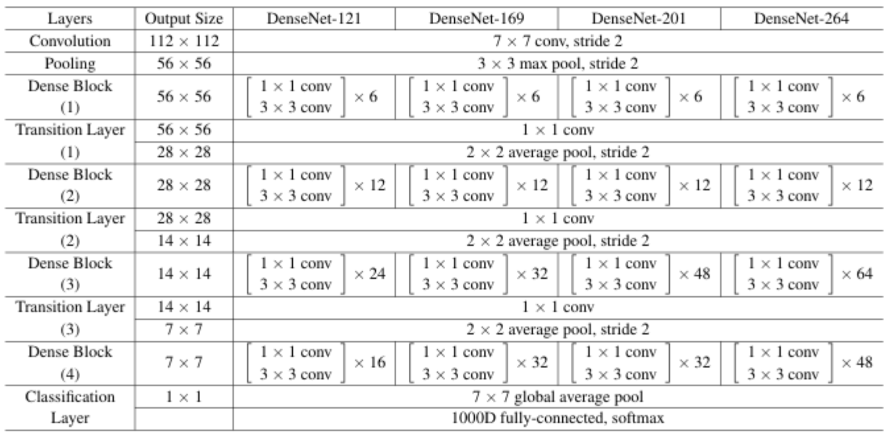
Structure of Dense Block:

- **Input**: The input to a dense block is passed through a series of convolutional layers.
- **Layer Connections**: Each layer concatenates the input feature maps with the outputs of all preceding layers.
- **Output**: The concatenated feature maps are passed to the next dense block or transition layer.

Some key features:

- **Growth Rate (k)**: Controls the number of output feature maps (channels) produced by each layer.
- **Bottleneck Layers**: To reduce computational cost, a 1×1 convolution is applied before the 3×3 convolution in each layer (similar to ResNet).
- **Transition layers**: Each dense block will increase the number of channels. But adding too many channels will create an overly complex model. Therefore, a feedforward layer will be used to control the complexity of the model. This layer uses a 1×1 convolution layer to reduce the number of channels

3. Network

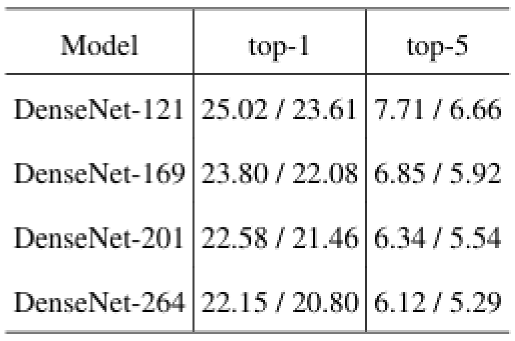


Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

DenseNet has the same architecture as ResNet. First, DenseNet uses a convolutional layer and a max-pooling layer like in ResNet. Then, similar to how ResNet uses residual blocks, DenseNet also uses dense blocks.

4. Result



Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

Table 3: The top-1 and top-5 error rates on the ImageNet validation set, with single-crop / 10-crop testing.

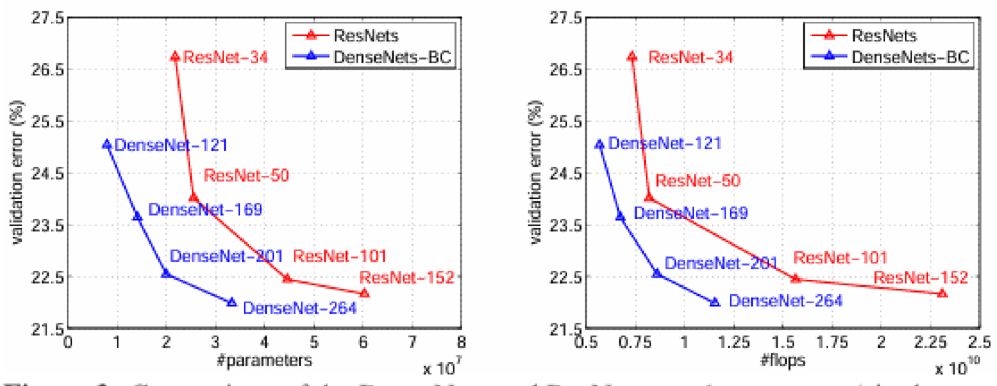


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

MobileNet

MobileNet, introduced by Andrew G. Howard et al. in 2017, is a **lightweight** convolutional neural network architecture designed specifically for mobile and embedded vision applications. The main goal of MobileNet is to achieve high accuracy while significantly reducing computational cost and model size, making it ideal for devices with limited computational power, such as smartphones and IoT devices.

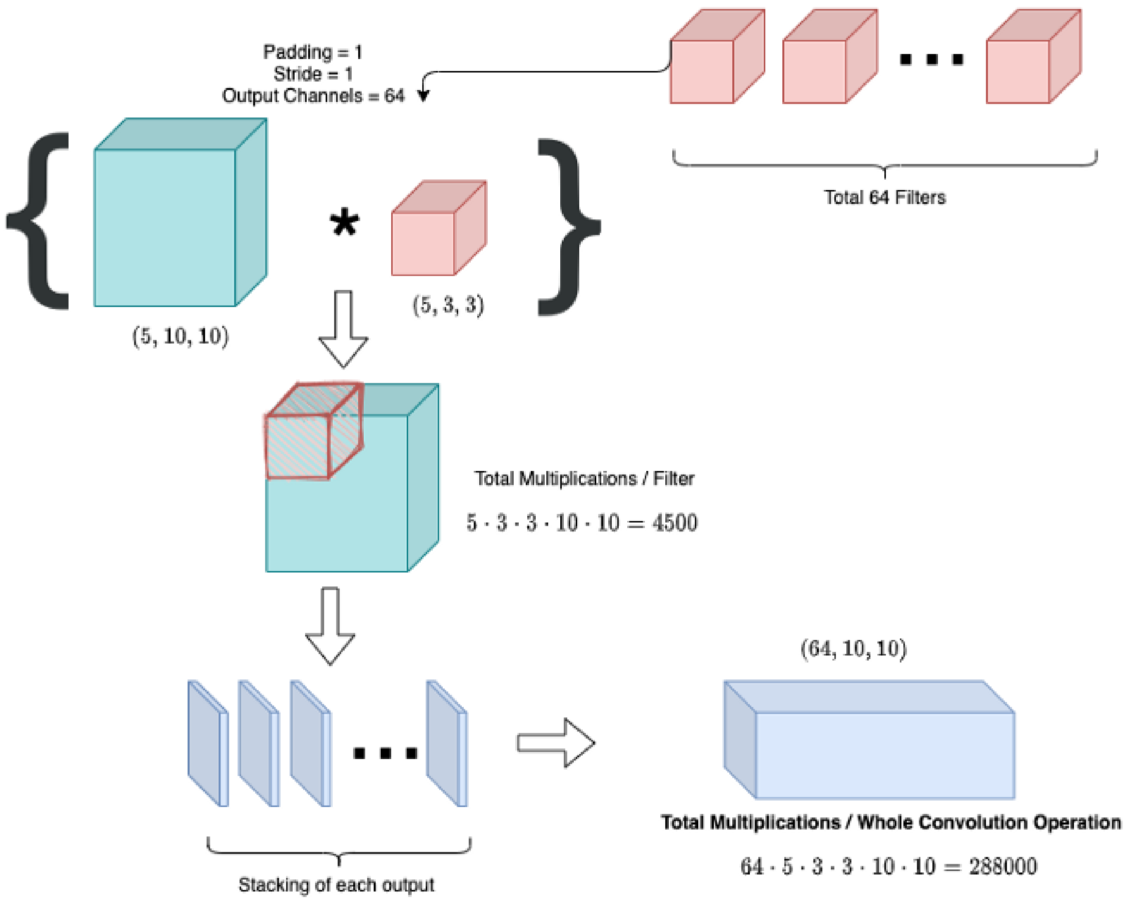
1. Key concept

Using Depthwise Separable Convolutions to help MobileNet reduce to several million parameters while maintaining stable accuracy.

2. Depthwise Seperable Convolutions

Depthwise separable convolution is a form of convolution that is decomposed into depthwise convolution called pointwise convolution. For MobileNets, depthwise convolution applies a single filter to each input channel. Pointwise convolution then applies depthwise convolution 1×1 to combine the depthwise convolution outputs.

Standard Convolution:

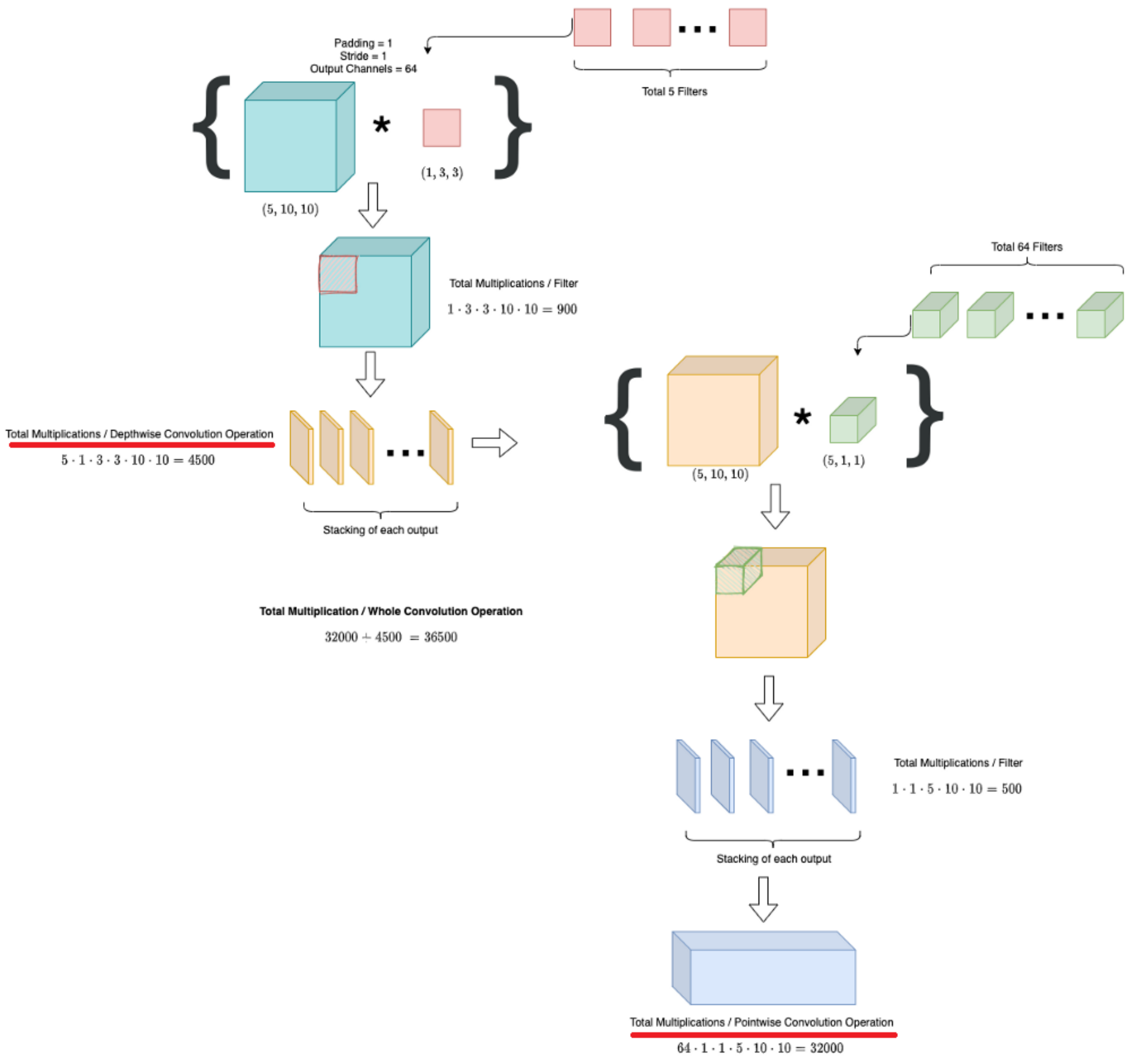

Padding = 1
Stride = 1
Output Channels = 64
Total 64 Filters
(5, 10, 10)
(5, 3, 3)
Total Multiplications / Filter
$5 \cdot 3 \cdot 3 \cdot 10 \cdot 10 = 4500$
(64, 10, 10)
Stacking of each output
Total Multiplications / Whole Convolution Operation
$64 \cdot 5 \cdot 3 \cdot 3 \cdot 10 \cdot 10 = 288000$

Note that: (5, 10, 10) = (channel, width, height)

When apply filter (5, 3, 3) to feature map (5, 10, 10), after performing convolution on the entire feature map (padding = 1, stride=1) we get the result 1×10×10. To archive 64 channels in the output, we have to apply 64 filter.

→ Total operation = 64 × 5 × 3 × 3 × 10 × 10 = 28800

Depthwise seperable convolution:



Padding = 1
Stride = 1
Output Channels = 64
Total 5 Filters
(5, 10, 10)
(1, 3, 3)
Total Multiplications / Filter
$1 \cdot 3 \cdot 3 \cdot 10 \cdot 10 = 900$
Total Multiplications / Depthwise Convolution Operation
$5 \cdot 1 \cdot 3 \cdot 3 \cdot 10 \cdot 10 = 4500$
Stacking of each output
Total Multiplication / Whole Convolution Operation
$32000 + 4500 = 36500$
Total 64 Filters
(5, 10, 10)
(5, 1, 1)
Total Multiplications / Filter
$1 \cdot 1 \cdot 5 \cdot 10 \cdot 10 = 500$
Stacking of each output
Total Multiplications / Pointwise Convolution Operation
$64 \cdot 1 \cdot 1 \cdot 5 \cdot 10 \cdot 10 = 32000$

a. **Depthwise convolution**: Instead of using 1 5×3×3 filter, we will use 5 1×3×3 filters and then stack the output together to get 5×10×10 output.

b. **Pointwise convolution**: We use the result from the deepwise convolution step. In this pointwise step, we only use a set of size 1×1. At the same time, the number of filters is equal to the number of channels we want to get. We want to increase to 64 channels, so let's use 64 filters.

→ Operations1 = 5 × 1 × 3 × 3 × 10 × 10 = 4500

→ Operations2 = 64 × 1 × 1 × 5 × 10 × 10 = 32000

Width Multiplier:

α is added to a layer, now the number of input channels and output channels is αM and αN, instead of M and N.

Thus, the computational cost of a convolution is:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

The smaller α is, the lower the computational cost (α provided include: 0.25, 0.5, 0.75, 1).

Resolution Multiplier:

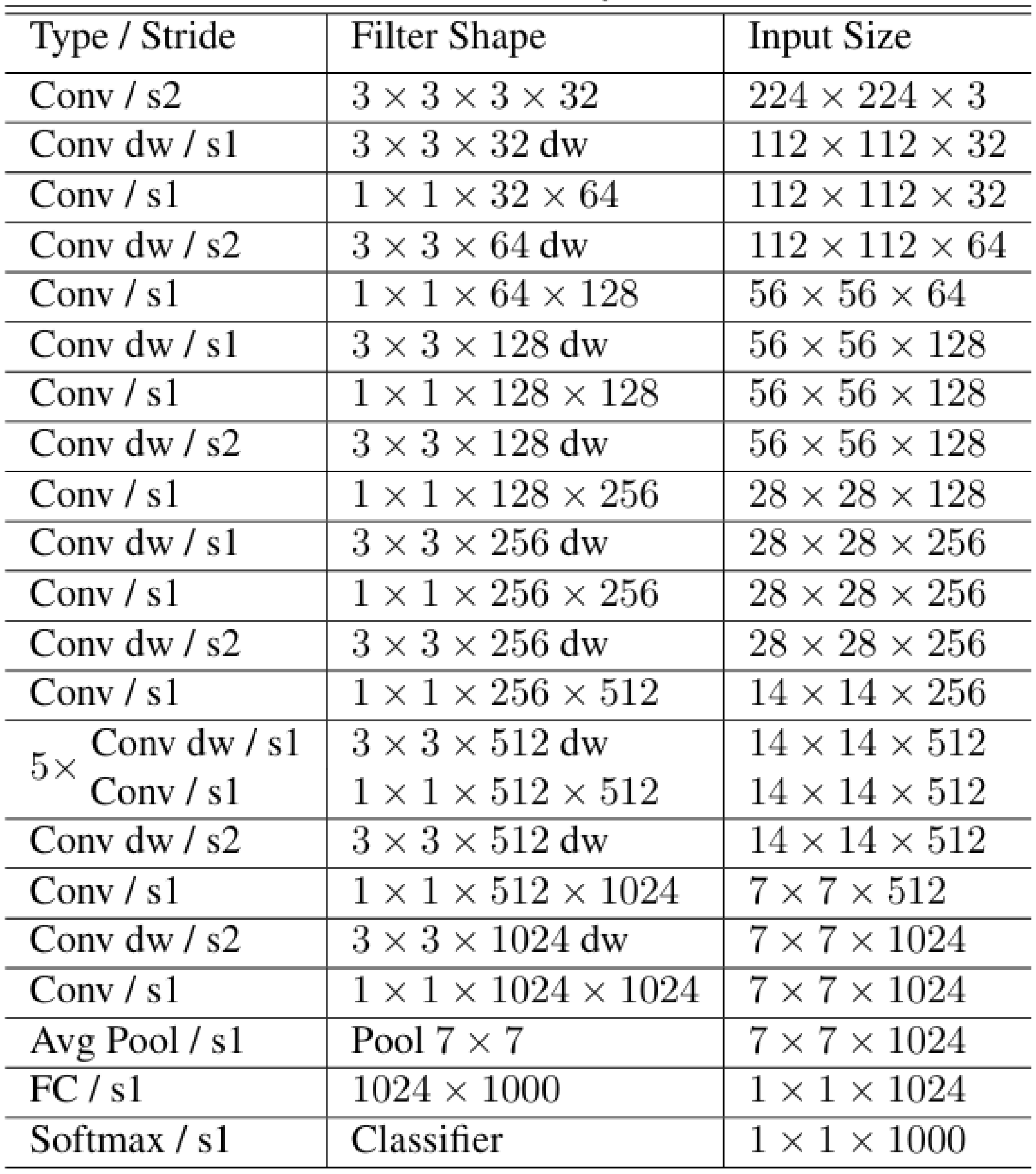
Similar to Width Multiplier α, Resolution Multiplier ρ is also added to layers to reduce the resolution of the input image and the internal representations between layers.

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$$

The ρ given to the input image have resolutions including: 224, 192, 160, 128.

3. Network

Table 1. MobileNet Body Architecture



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

4. Result

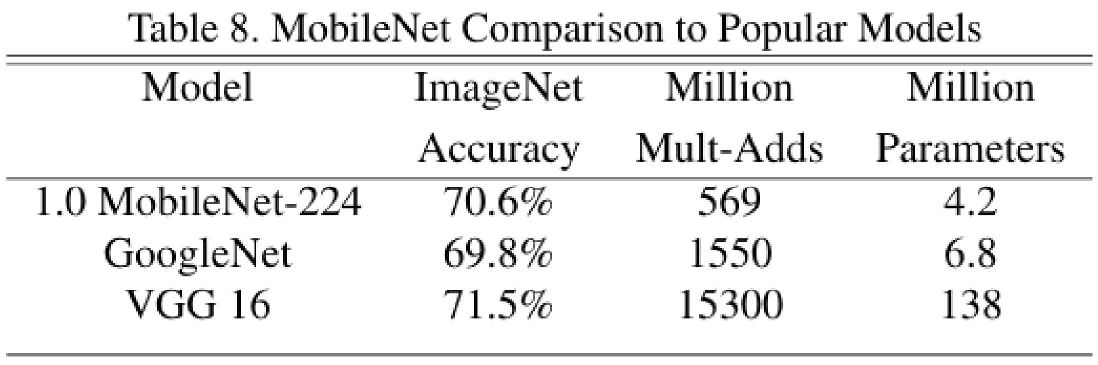


Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

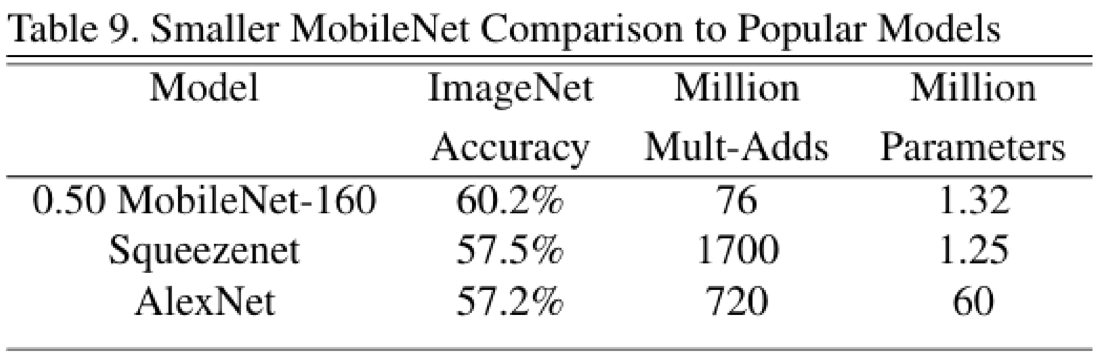


Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

SE-Net

SE Net was born from the idea "We have enhanced information spatially. Why don't we enhance information between channels themselves?"

1. Key concept

SE is a fairly simple network consisting of just a few layers that aims to enhance the information between channels and thereby improve the representation quality of the CNN model. SE does this by using all the information and then **selectively emphasizing each channel with important features** and paying less attention to the less important channels.

2. Squeeze-and-Excitation block

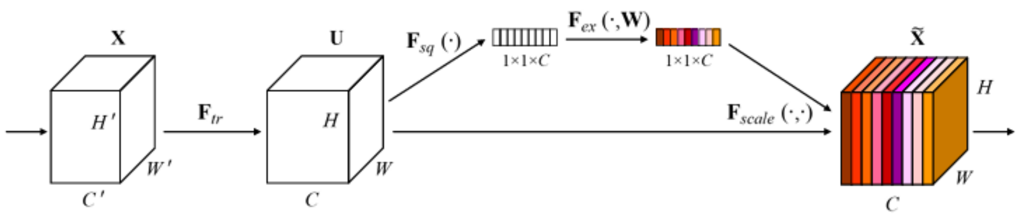


Fig. 1. A Squeeze-and-Excitation block.

Symbol:

- **X**: Input image has size H' x W' x C′
- **F_tr**: Set of transformations: a few convolution layers, or a stage of VGG, a block in ResNet, ...
- **U**: Feature map has size H x W x C

Architecture:

- **Step 1**: The input image X is passed through a set of transformations F_tr to extract a feature map U.
- **Step 2**: The feature map U (H x W x C) is passed through the **squeeze function** to generate a feature description matrix of each channel (1 × 1 x C) by aggregating the feature map U in the H and W dimensions. An example of the squeeze function here could be global average pooling.
- **Step 3**: Following the squeeze function is the **excitation function**. The excitation function acts as a mechanism to describe the dependencies between channels. The function takes as input the feature matrix of each channel calculated from step 2 through several transformation layers such as convolution, activation function, ... and finally through the gate function to generate attention weights for each channel. These weights are then multiplied by the feature map U to calculate the output of the SE block. The output of the SE block now contains only the information that is really important for the problem. The gate function here is usually the sigmoid function.

Squeeze: Global Information Embedding

A global pooling operation aggregates the spatial information into a compact vector of each channel descriptors. This operation captures the global statistics of each channel, providing a global context for feature recalibration.

Excitation: Adaptive Recalibration

As in **Step 3**.

3. Network

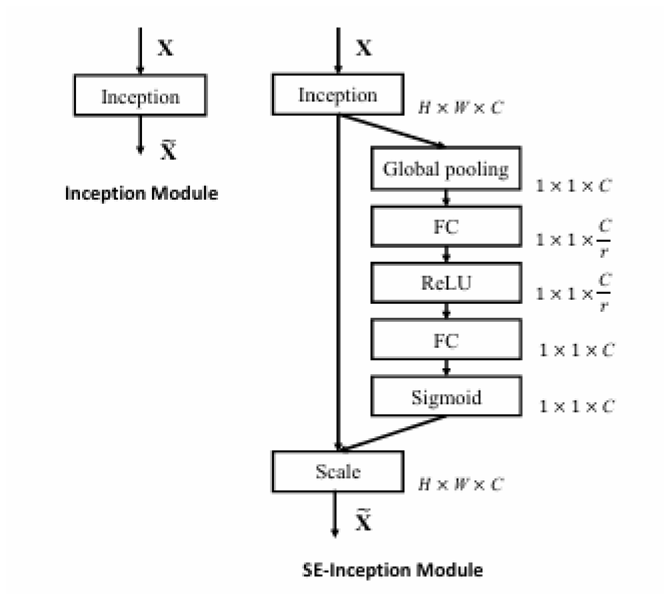


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

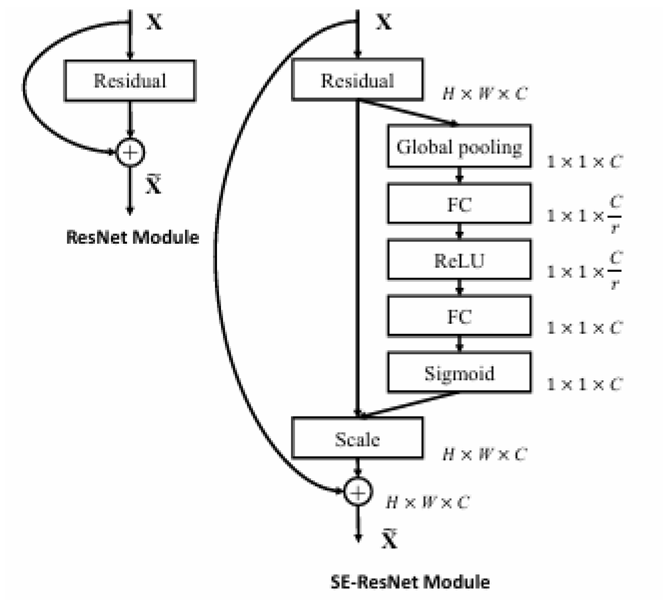


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

SE blocks are incorporated into popular CNN architectures to form **Squeeze-and-Excitation Networks (SENets)**. Some examples include:

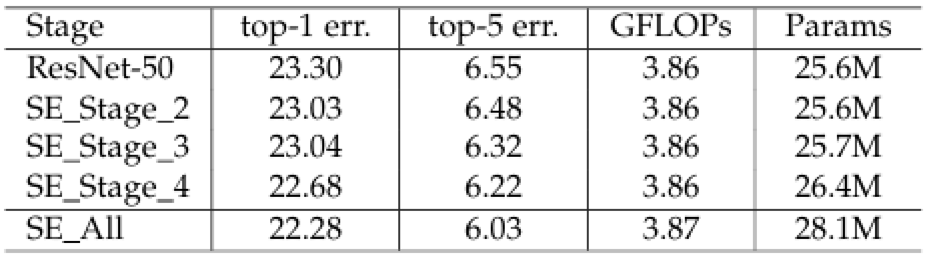
- **SE-ResNet**: The SE block is added to residual modules, typically after the ReLU activation function.
- **SE-Inception**: SE blocks are applied to Inception modules, recalibrating feature channels output by the multi-branch structure.

- **SE-ResNeXt**: SE blocks integrate with the grouped convolutional layers of ResNeXt to further enhance representation power.

The integration of SE blocks allows the network to recalibrate feature channels dynamically, adapting to the specific needs of the task. Despite introducing additional parameters, SE blocks impose only a slight computational overhead while consistently boosting performance across architectures and datasets.

4. Result

TABLE 13
Effect of integrating SE blocks with ResNet-50 at different stages on ImageNet (error rates %).



Stage	top-1 err.	top-5 err.	GFLOPs	Params
ResNet-50	23.30	6.55	3.86	25.6M
SE_Stage_2	23.03	6.48	3.86	25.6M
SE_Stage_3	23.04	6.32	3.86	25.7M
SE_Stage_4	22.68	6.22	3.86	26.4M
SE_All	22.28	6.03	3.87	28.1M