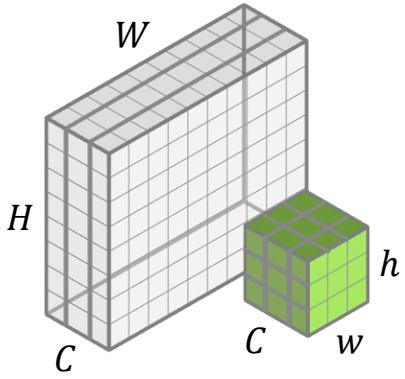
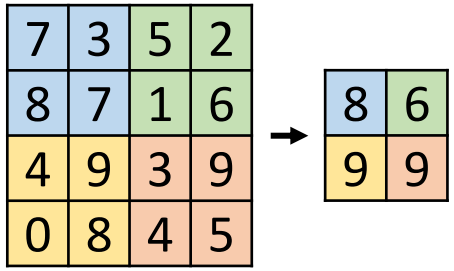
CNN Architectures

January 3rd, 2024

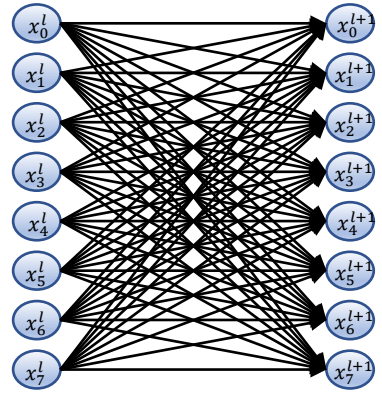
The ConvNet Building Blocks



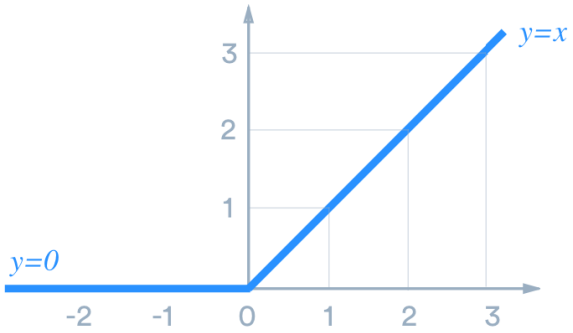
Conv Layer



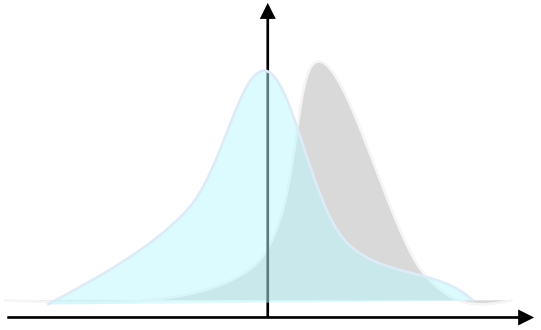
Max Pooling



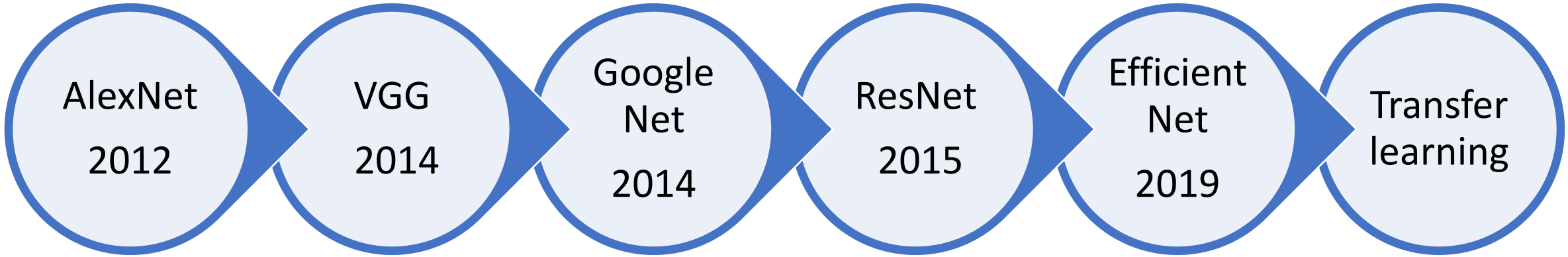
FC Layer



Activation Func.



Batch Norm


AlexNet
2012
VGG
2014
Google
Net
2014
ResNet
2015
Efficient
Net
2019
Transfer
learning

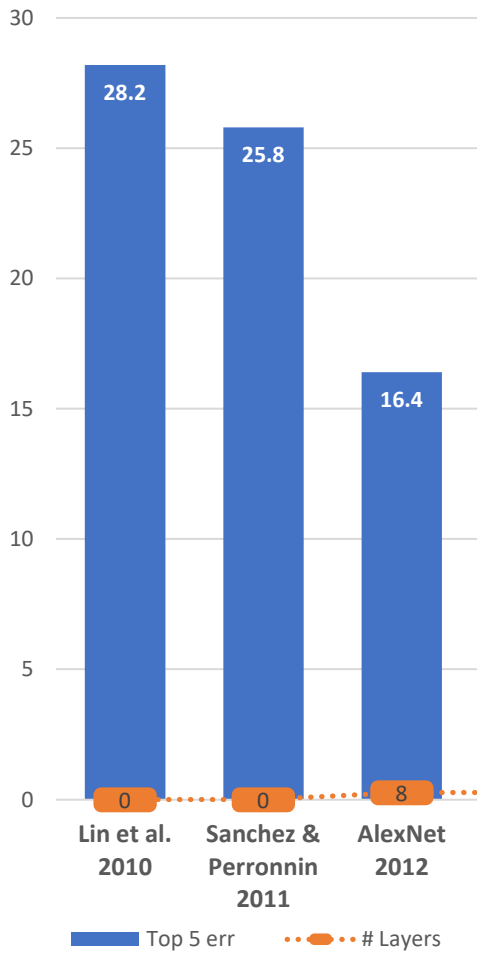
IMAGENET

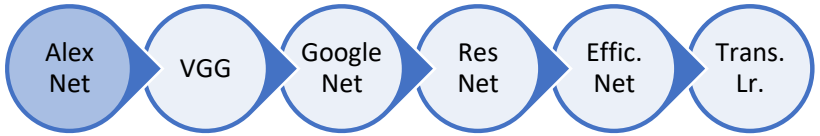
Visual Recognition Challenge (ILSVRC)

- 1.2M training images
- 100K testing images
- 1K categories

Olga Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

AlexNet

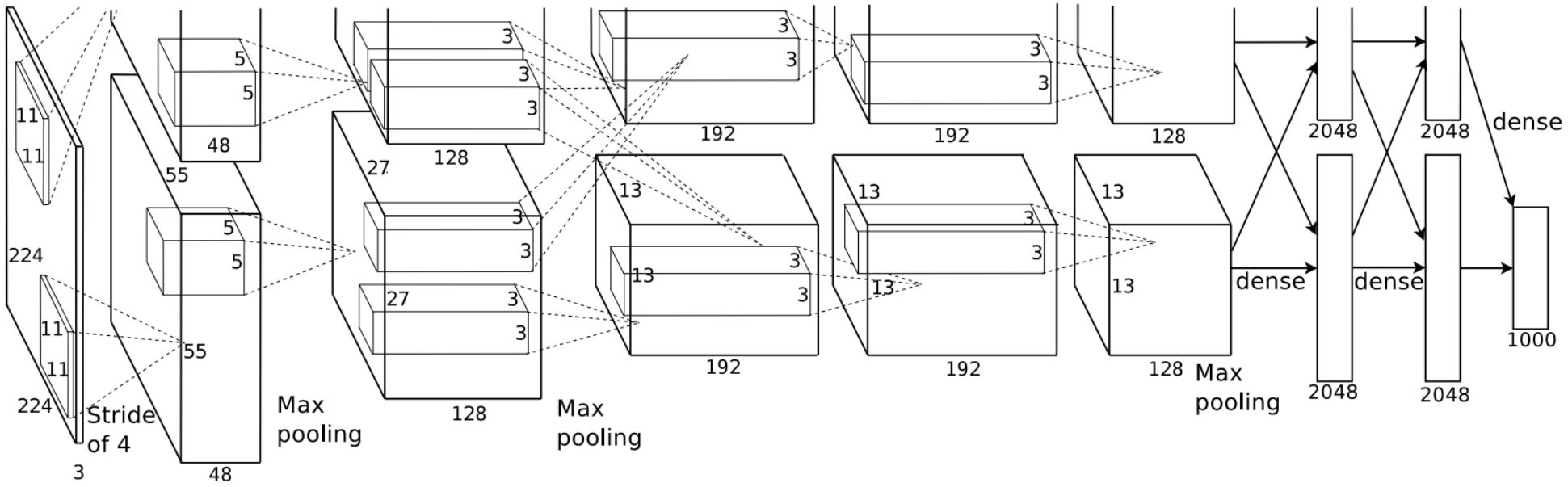




- Fukushima 1980
- LeCunn et al. 1989
- LeCunn et al. 1998

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks.NIPS, 2012

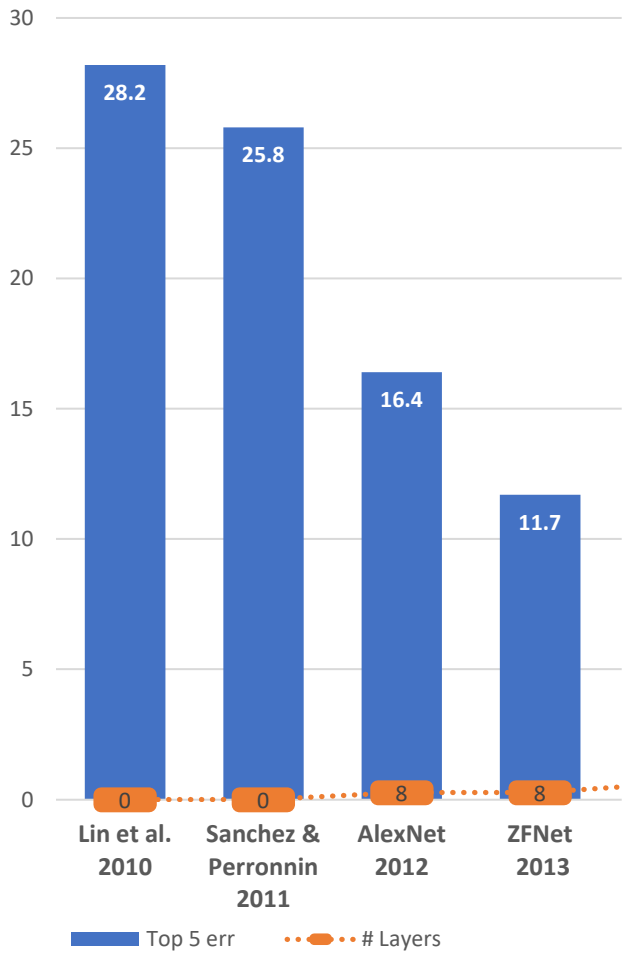
AlexNet

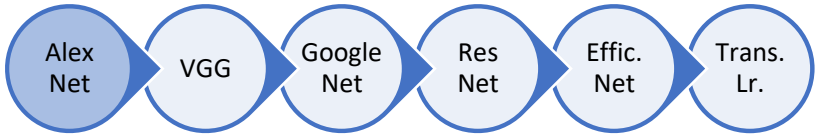


Architecture and training details:

- 5 conv + 3 fc layers
- ReLU activation
- Max Pooling
- Dropout
- SGD + Momentum

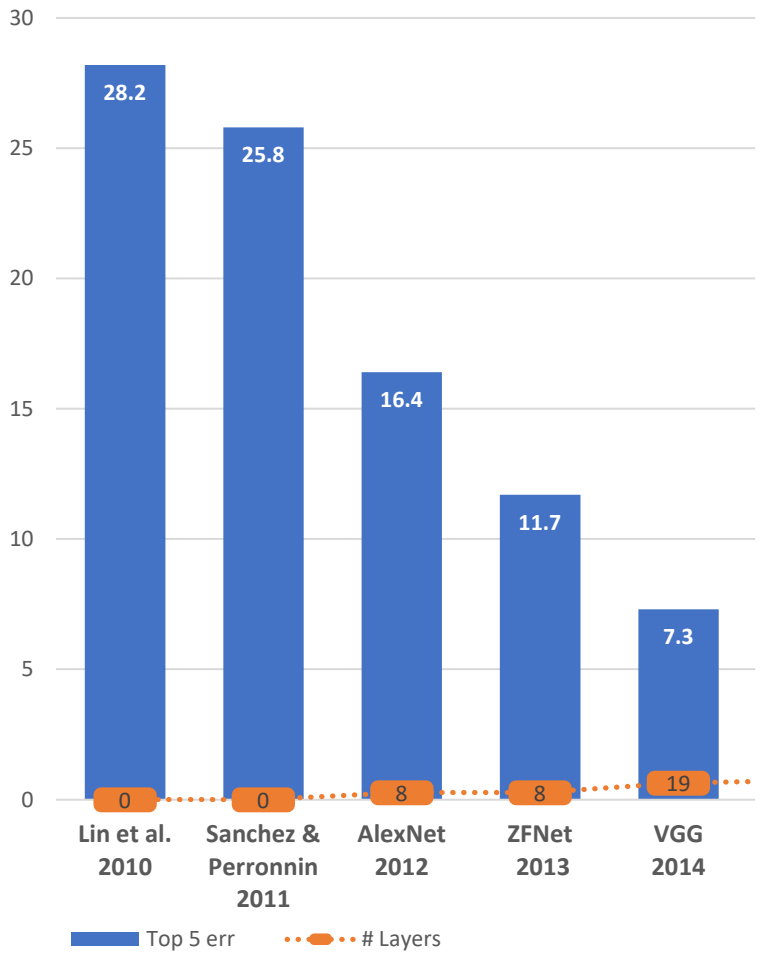
ZFNet

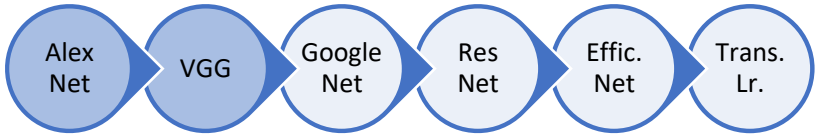


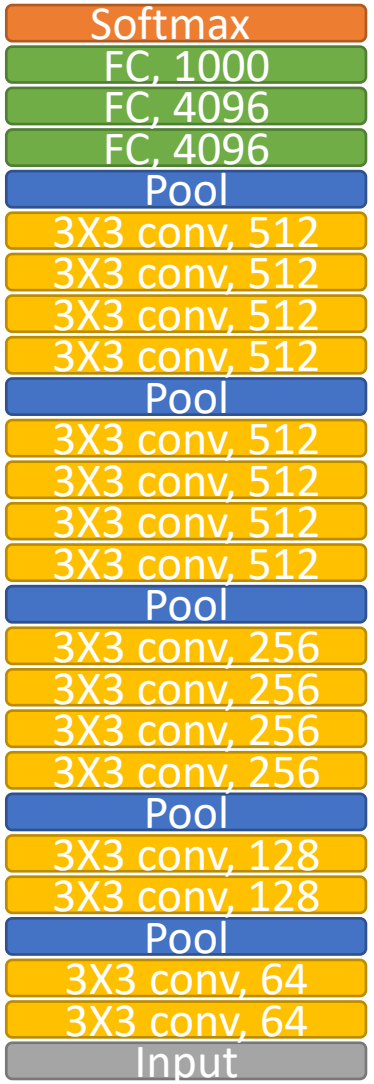


Matthew D Zeiler, Rob Fergus. Visualizing and Understanding Convolutional Networks. ECCV, 2014.

VGG

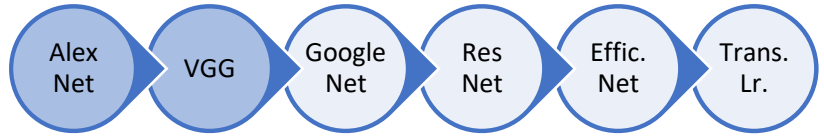




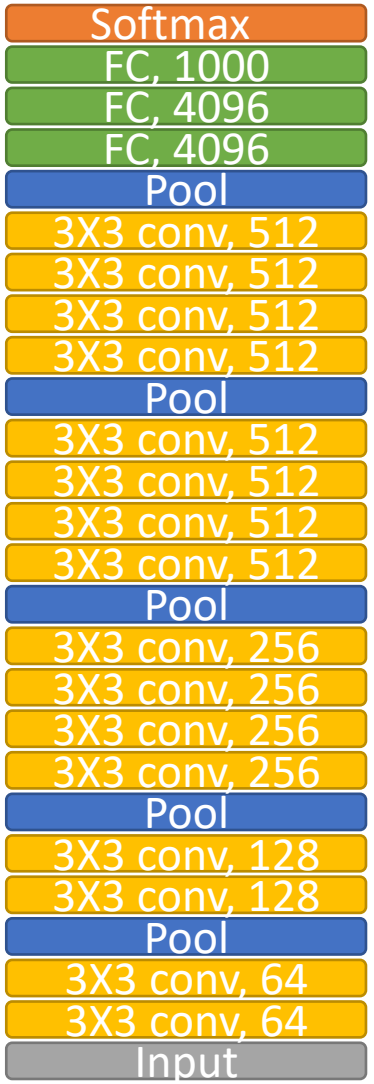


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

VGG

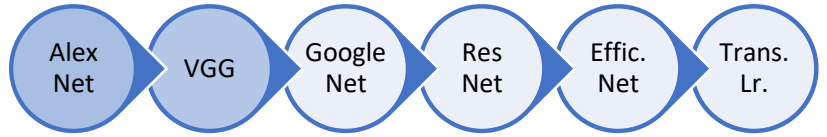


- All conv layers have 3x3 kernels, stride 1, pad 1
- All pooling layers are 2x2, stride 2
- After pooling, double the number of channels



Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

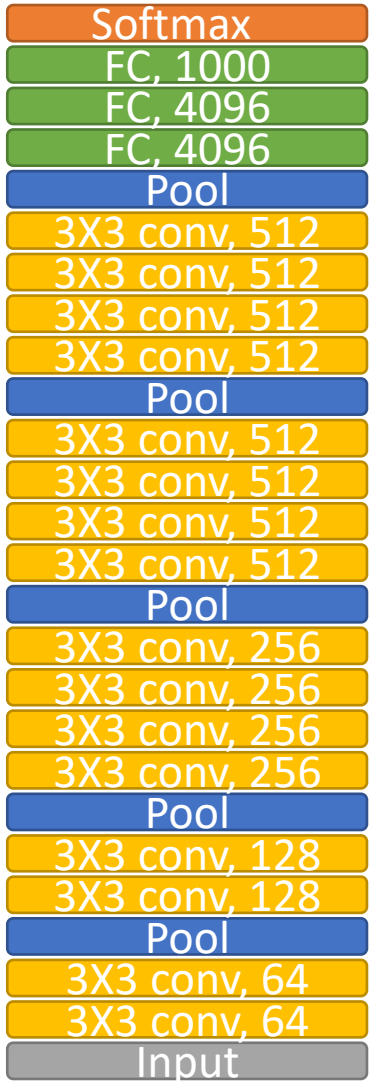
Why using 3x3 conv layers?



5X5 conv, C

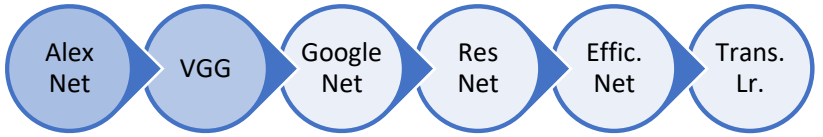
3X3 conv, C
3X3 conv, C

Receptive Field



Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?



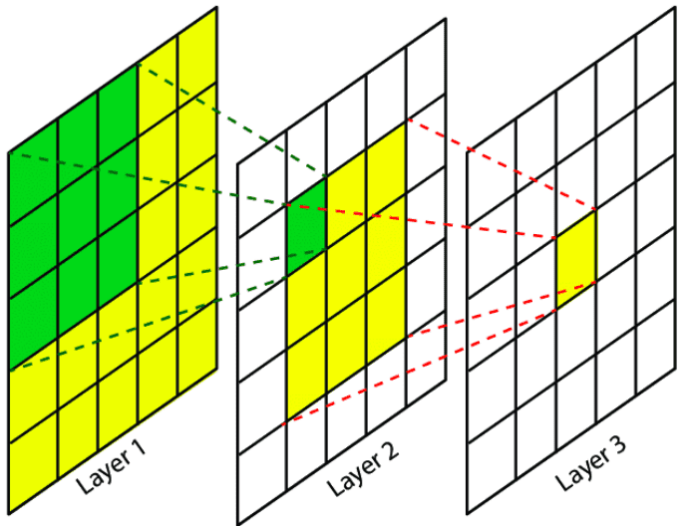
5X5 conv, C

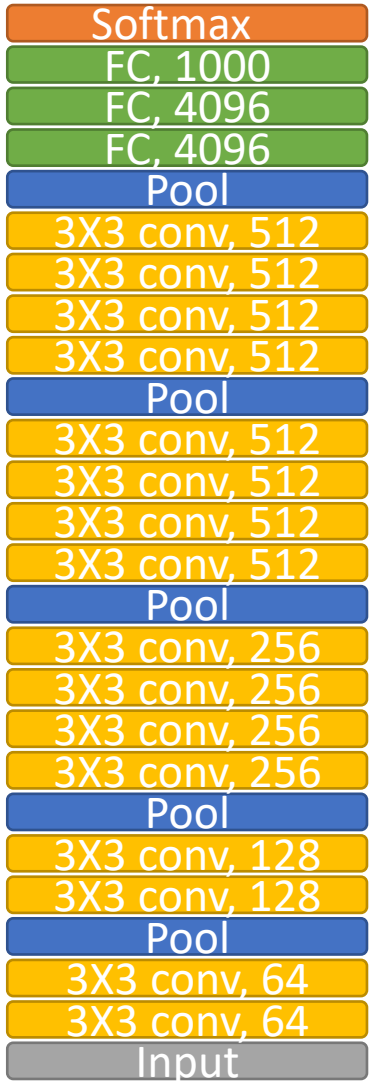
3X3 conv, C

3X3 conv, C

Receptive Field 5

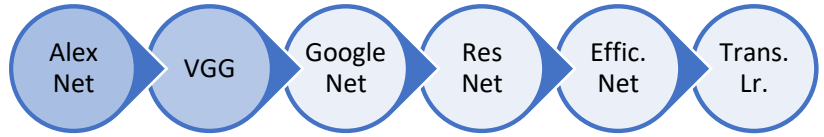
Parameters



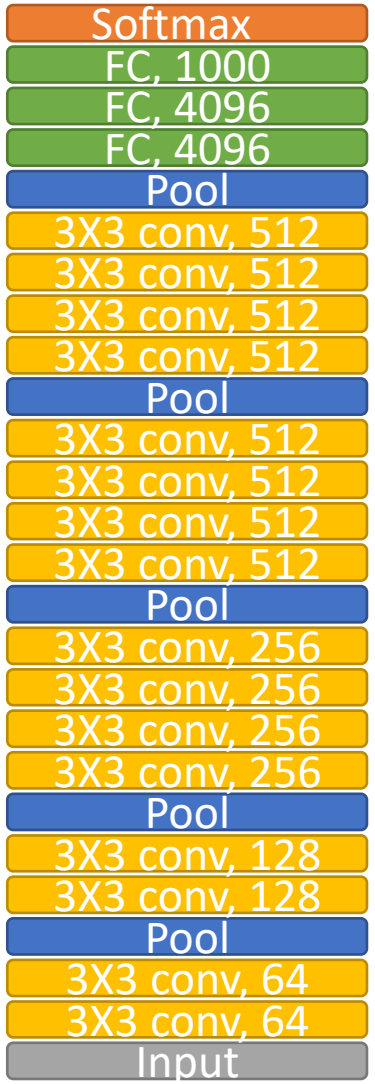


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?

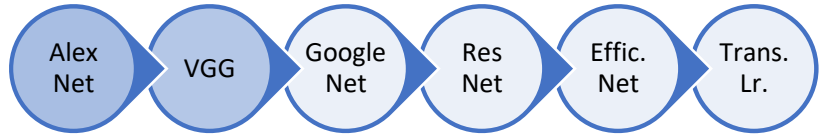


	5X5 conv, C	3X3 conv, C 3X3 conv, C
Receptive Field	5	5
# Parameters		

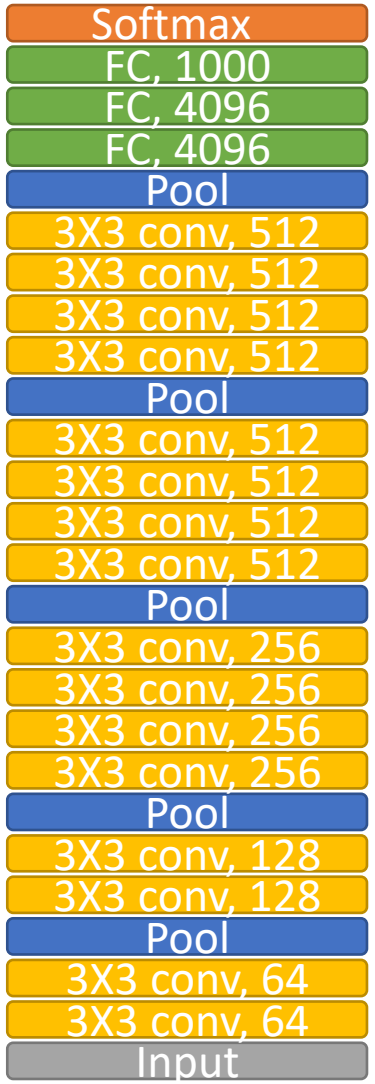


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?

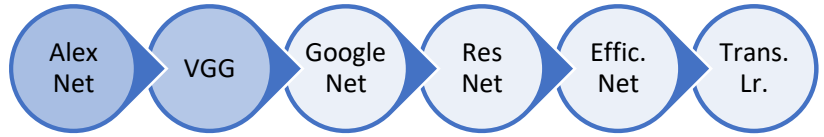


	5X5 conv, C	3X3 conv, C 3X3 conv, C
Receptive Field	5	5
# Parameters	$25C^2$	$9C^2+9C^2 = 18C^2$
# FLOPs		

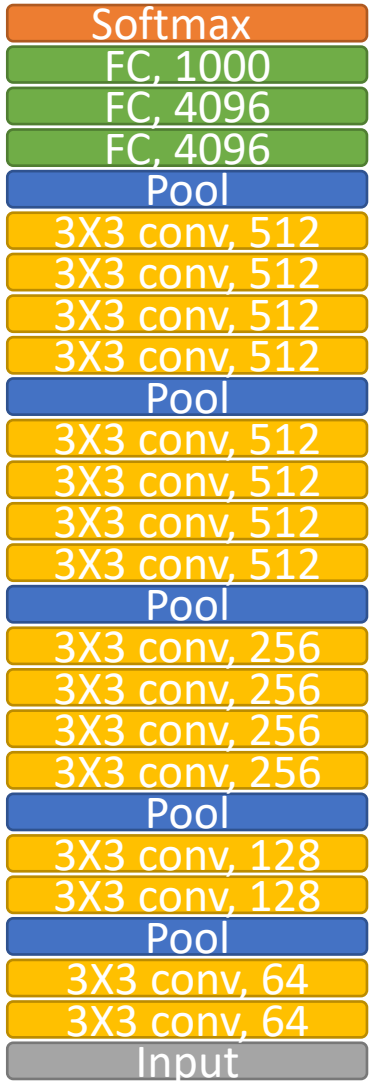


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?

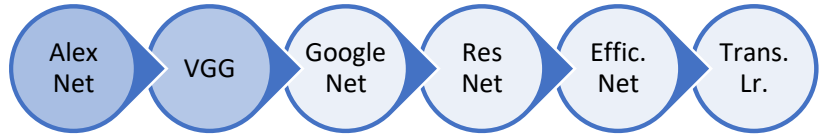


	5X5 conv, C	3X3 conv, C 3X3 conv, C
Receptive Field	5	5
# Parameters	$25C^2$	$9C^2+9C^2 = 18C^2$
# FLOPs	$\sim 25C^2HW$	$\sim 18C^2HW$

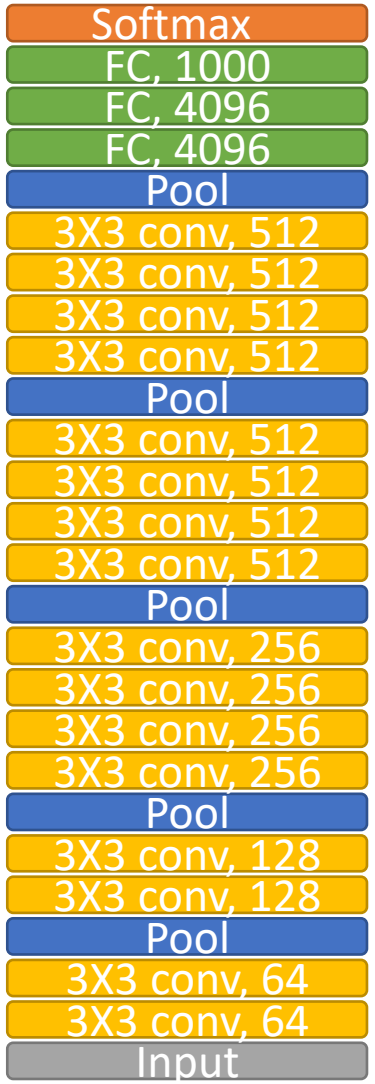


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?

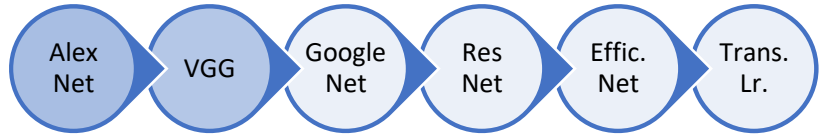


	5X5 conv, C	3X3 conv, C 3X3 conv, C
Receptive Field	5	5
# Parameters	$25C^2$	$9C^2+9C^2 = 18C^2$
# FLOPs	$\sim 25C^2HW$	$\sim 18C^2HW$
Memory Size (Output)		

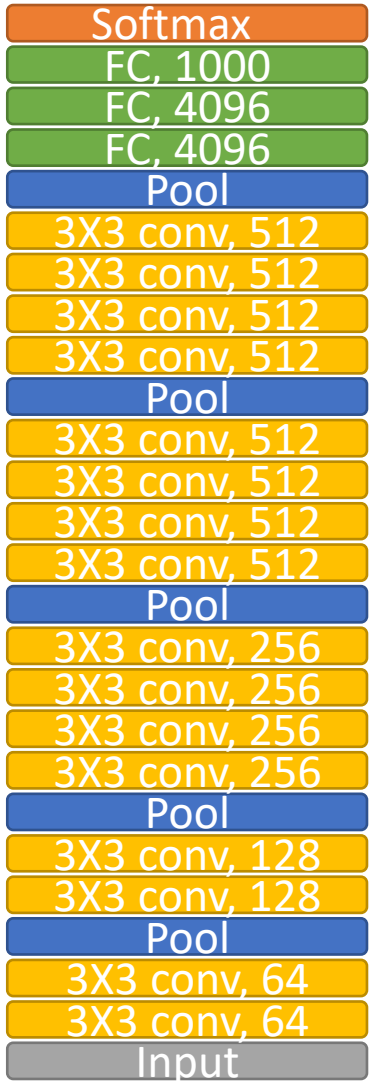


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

Why using 3x3 conv layers?

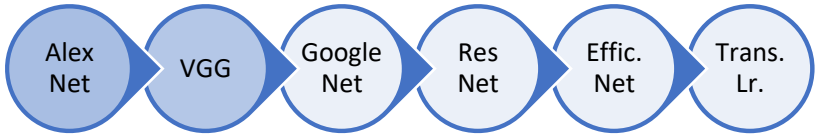


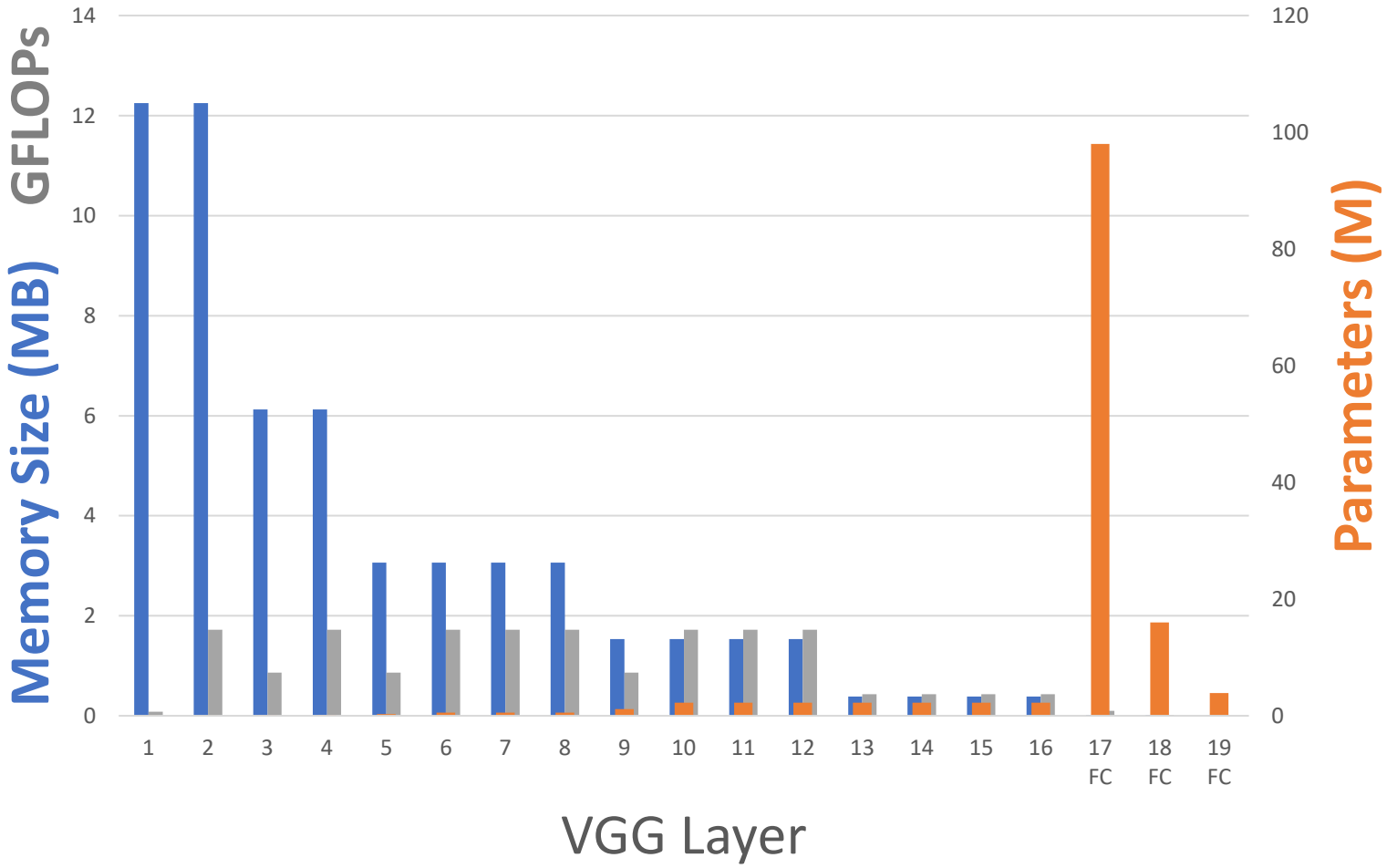
	5X5 conv, C	3X3 conv, C 3X3 conv, C
Receptive Field	5	5
# Parameters	$25C^2$	$9C^2+9C^2 = 18C^2$
# FLOPs	$\sim 25C^2HW$	$\sim 18C^2HW$
Memory Size (Output)	~HWC	~2HWC

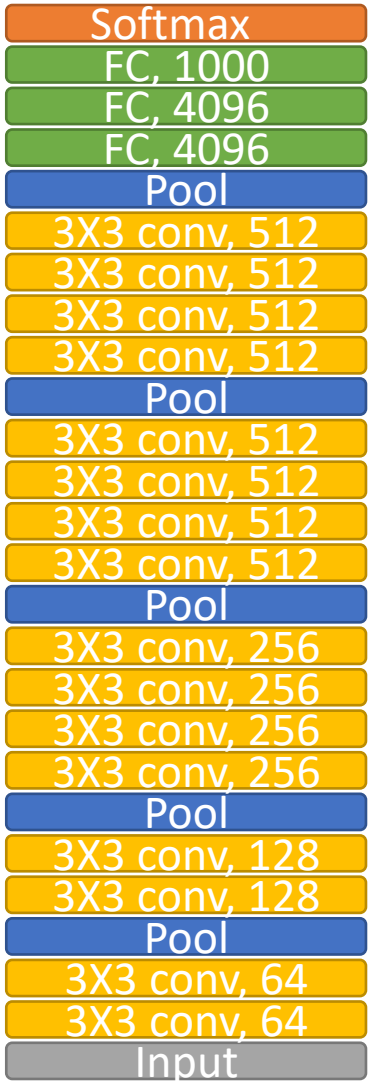


Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

VGG resources

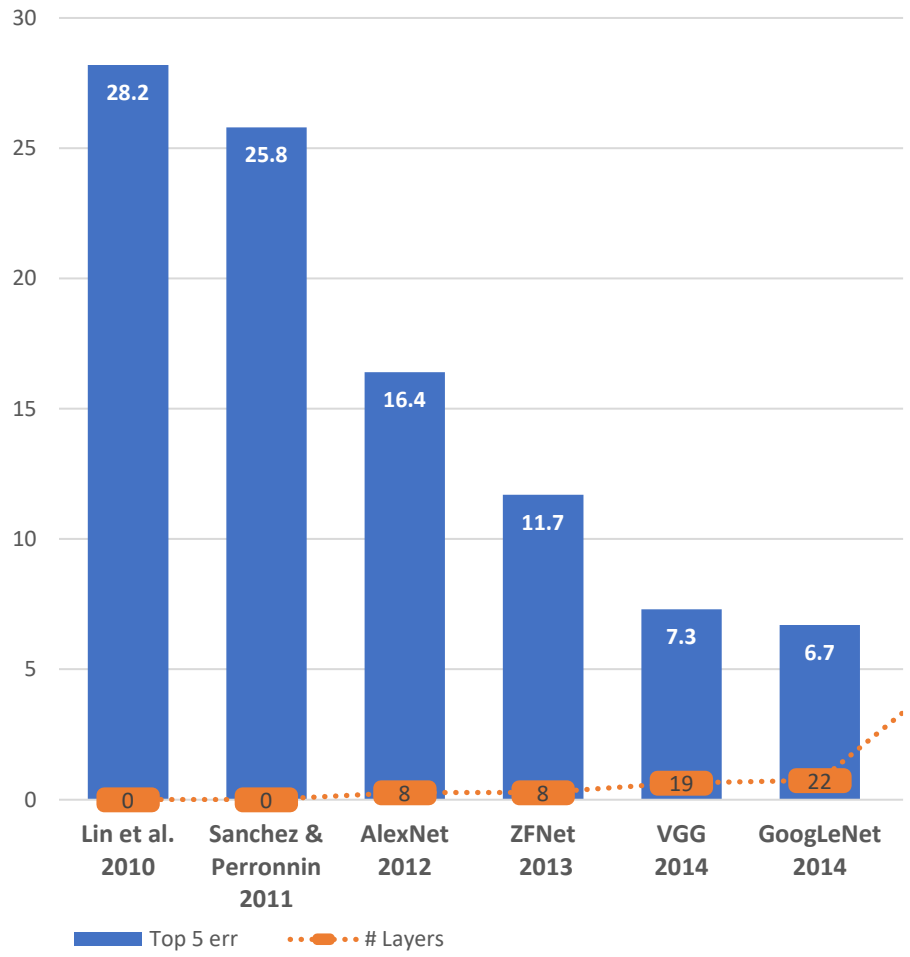


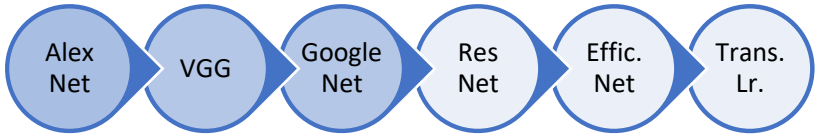




Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015.

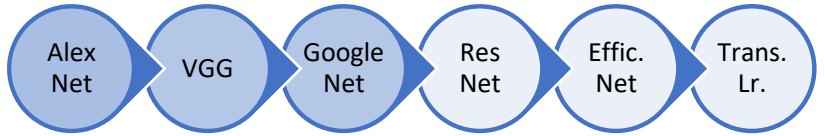
GoogLeNet

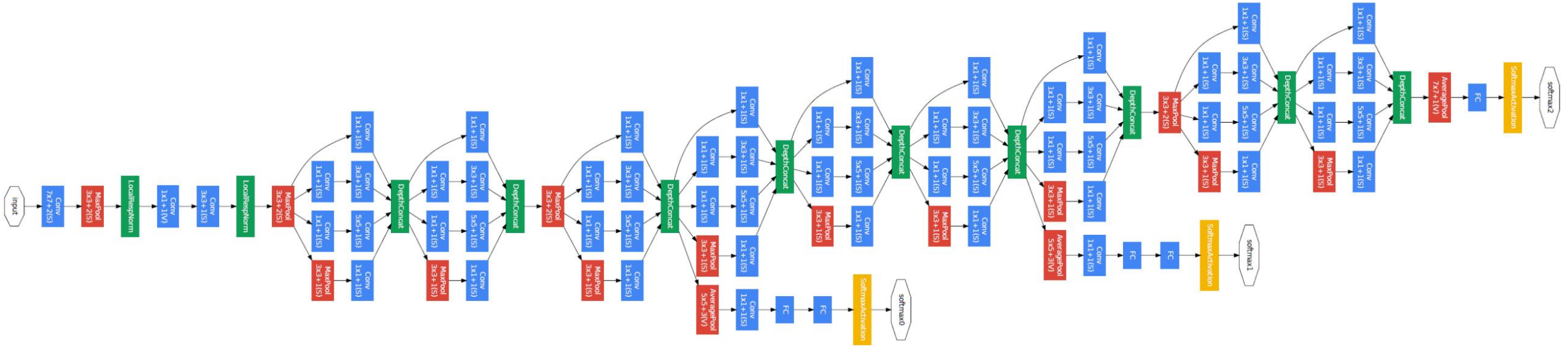




Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.

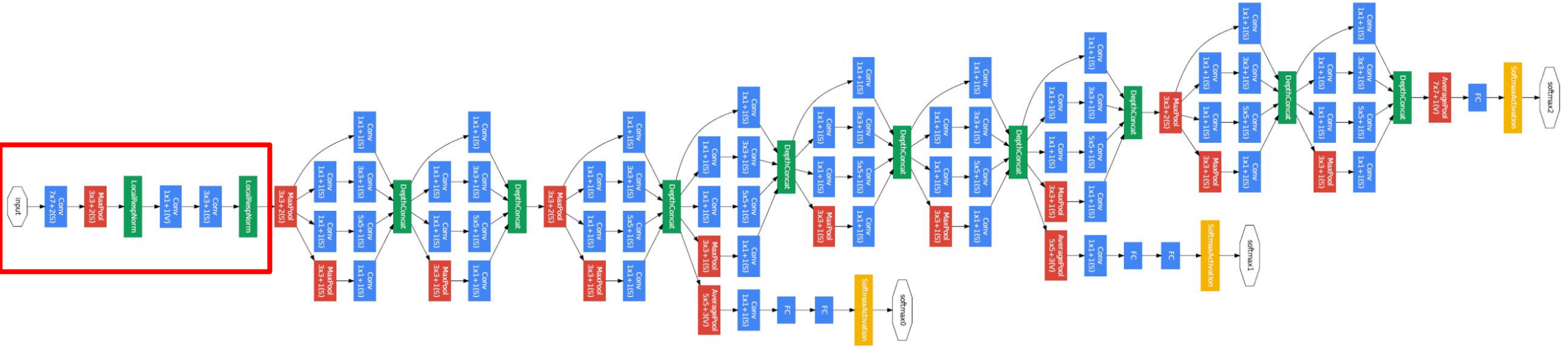
GoogLeNet





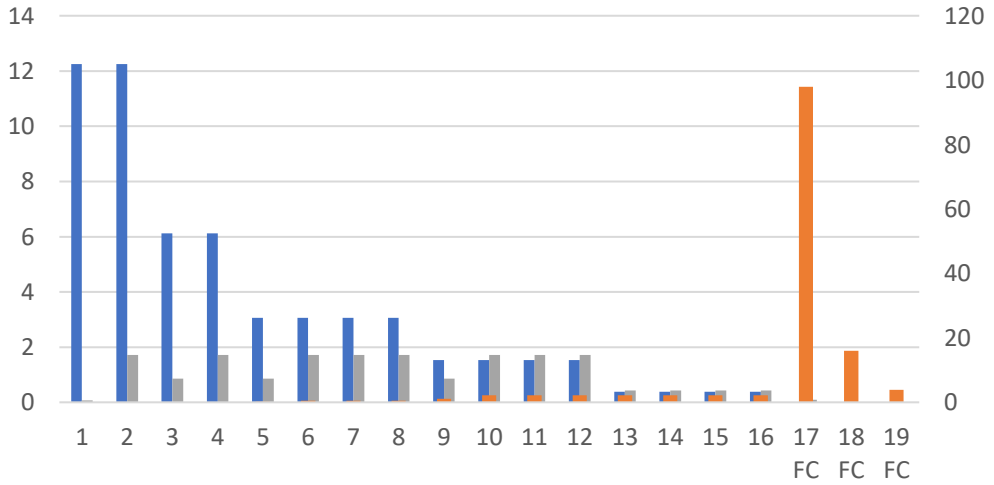
Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.

GoogLeNet

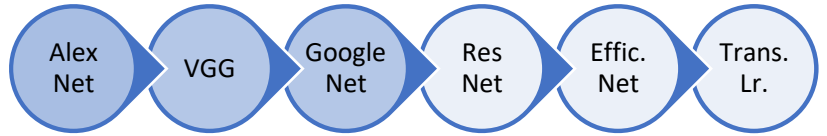


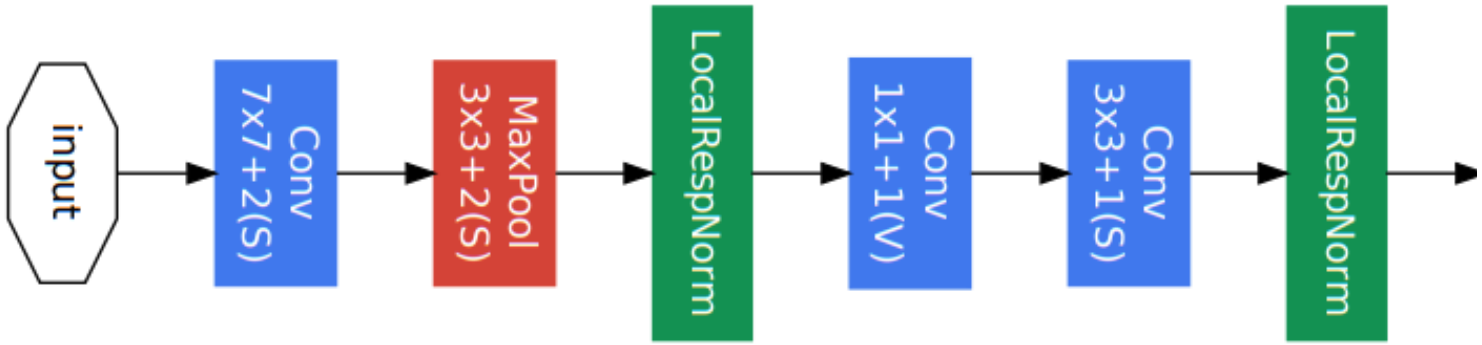
Want to avoid:

- High memory usage in the beginning
- High parameter count in the end

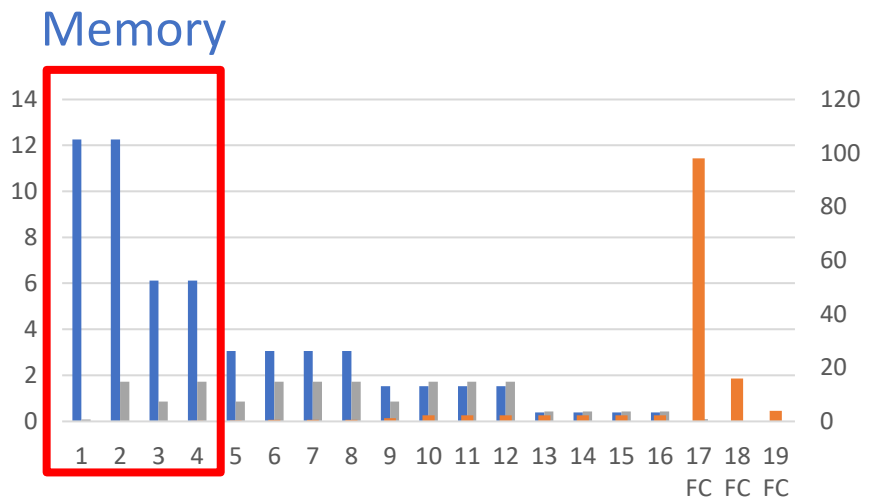


Reducing memory usage

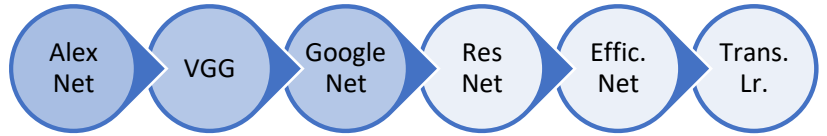


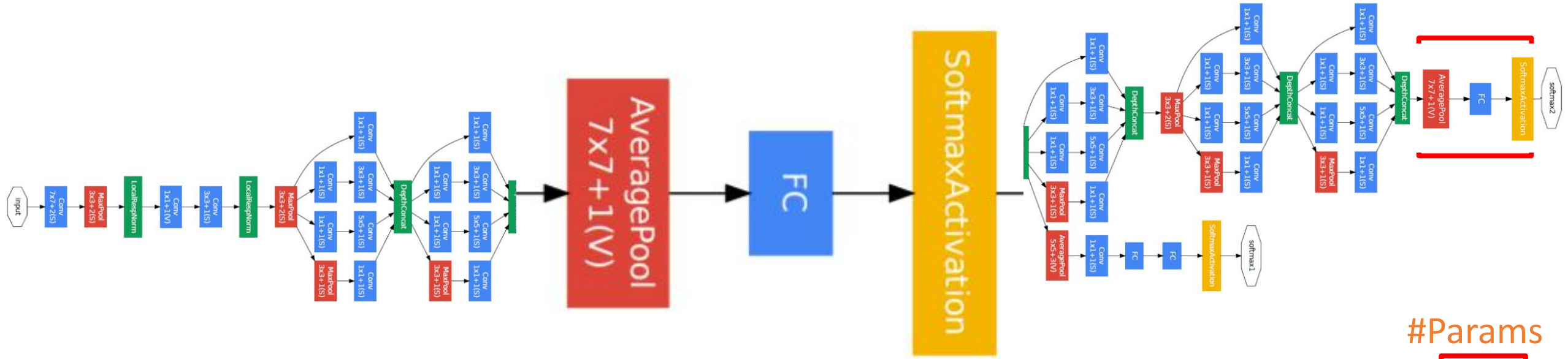


Downsample the input with 7*7 convolutions and MaxPool

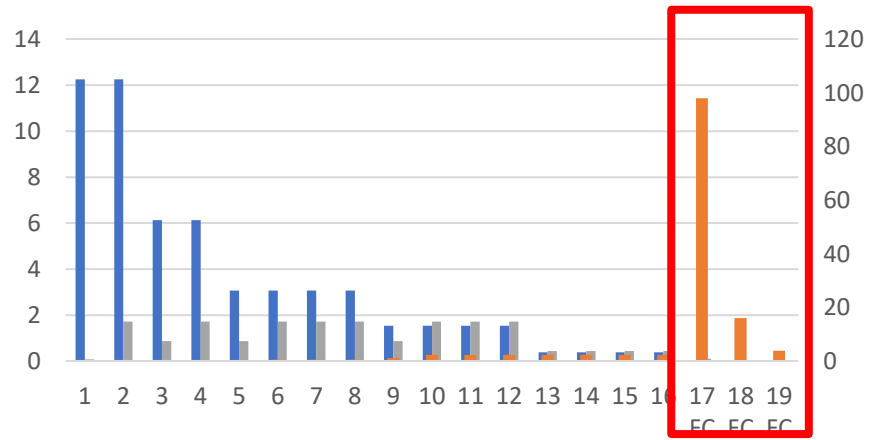


Reducing parameter count

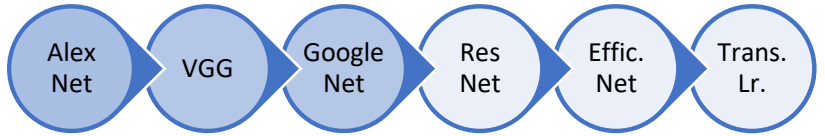


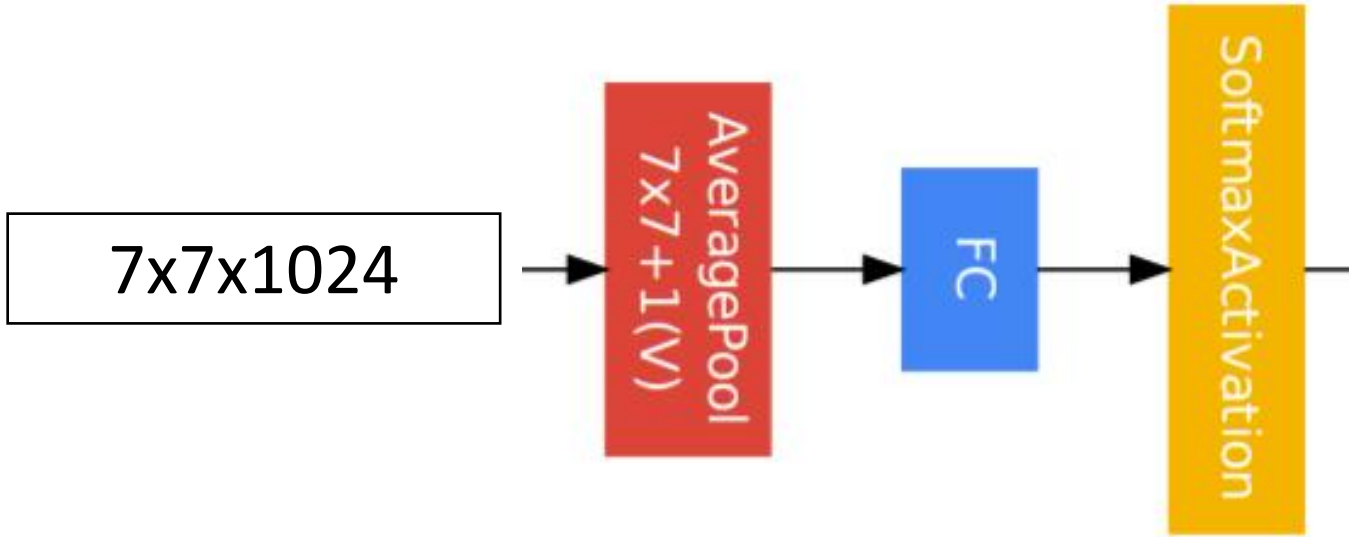


#Params

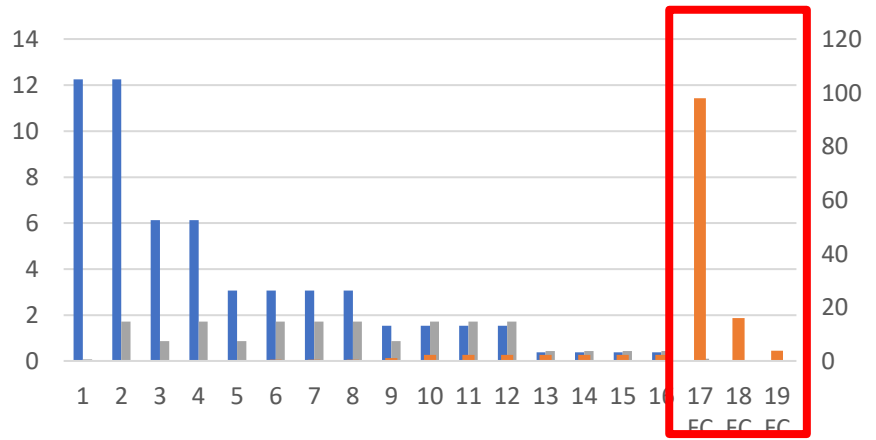


Global average pooling

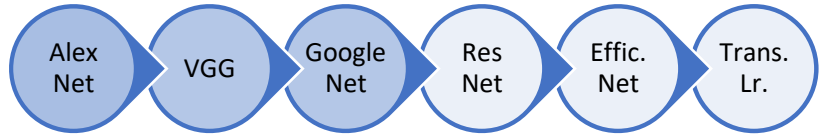


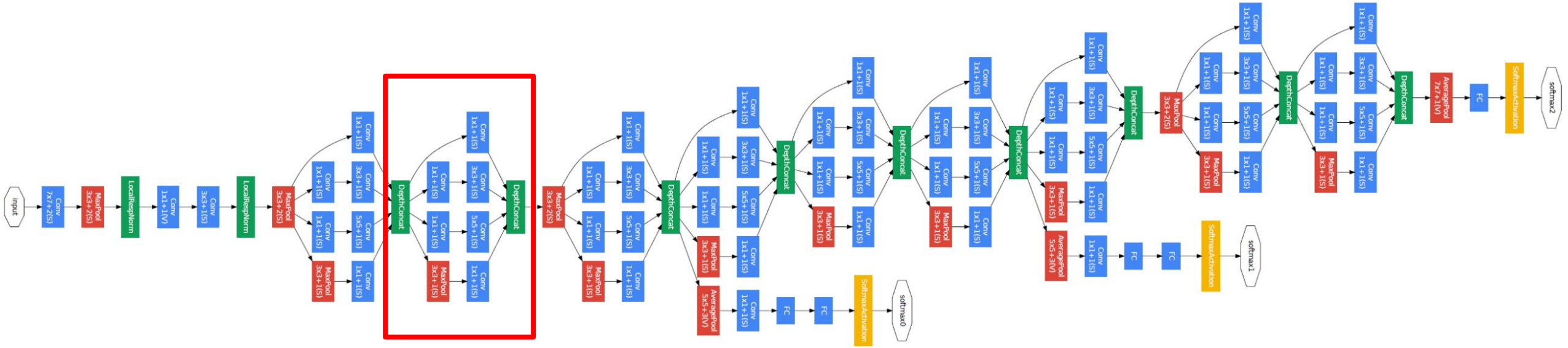


#Params



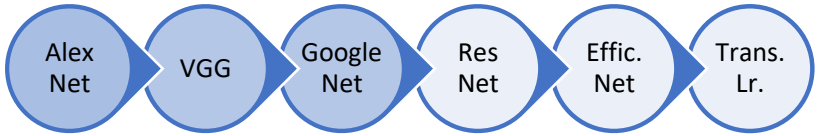
GoogLeNet



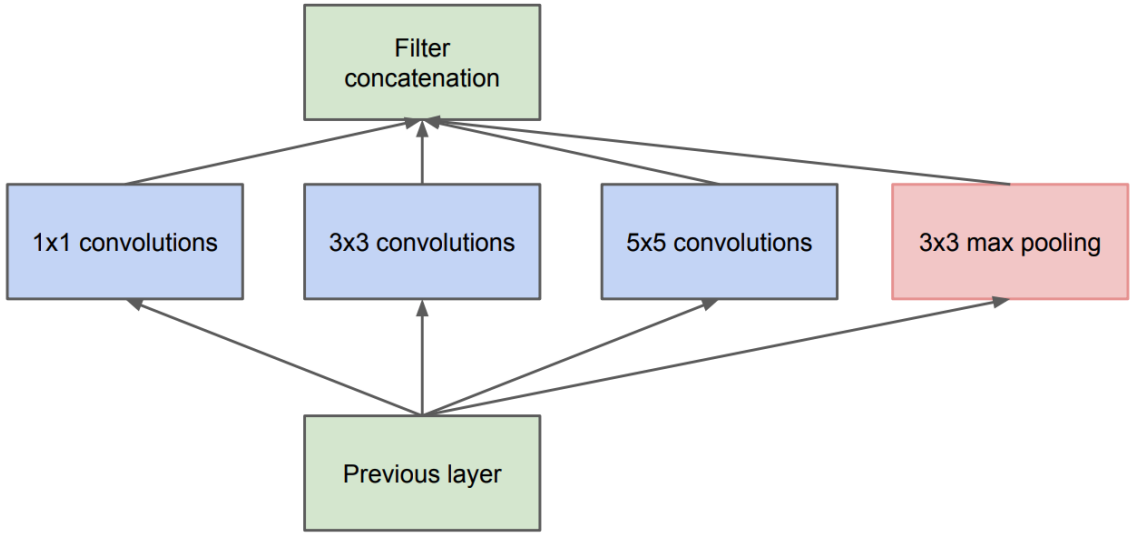


Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.

GoogLeNet: Inception blocks

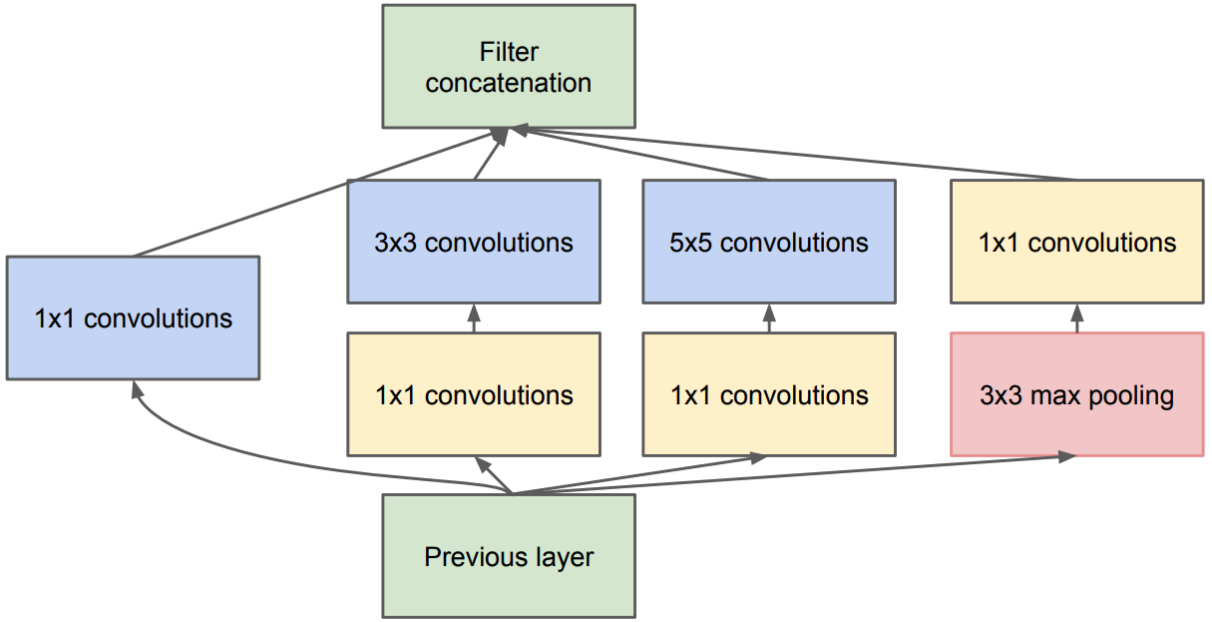


Inception block



Problem: too computationally expensive

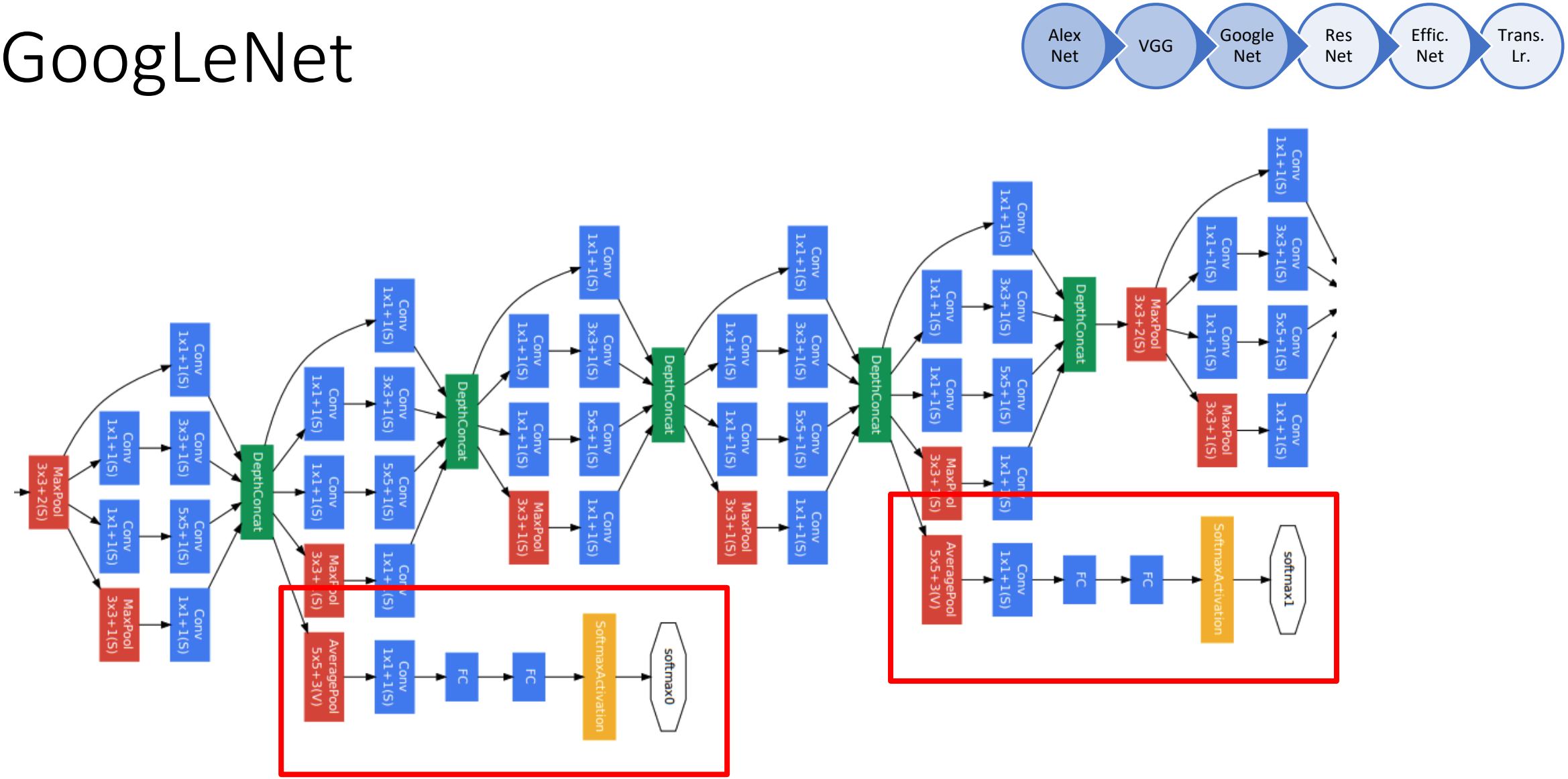
Inception block with dimension reductions



Solution: reduce # channels by 1*1 convolutions

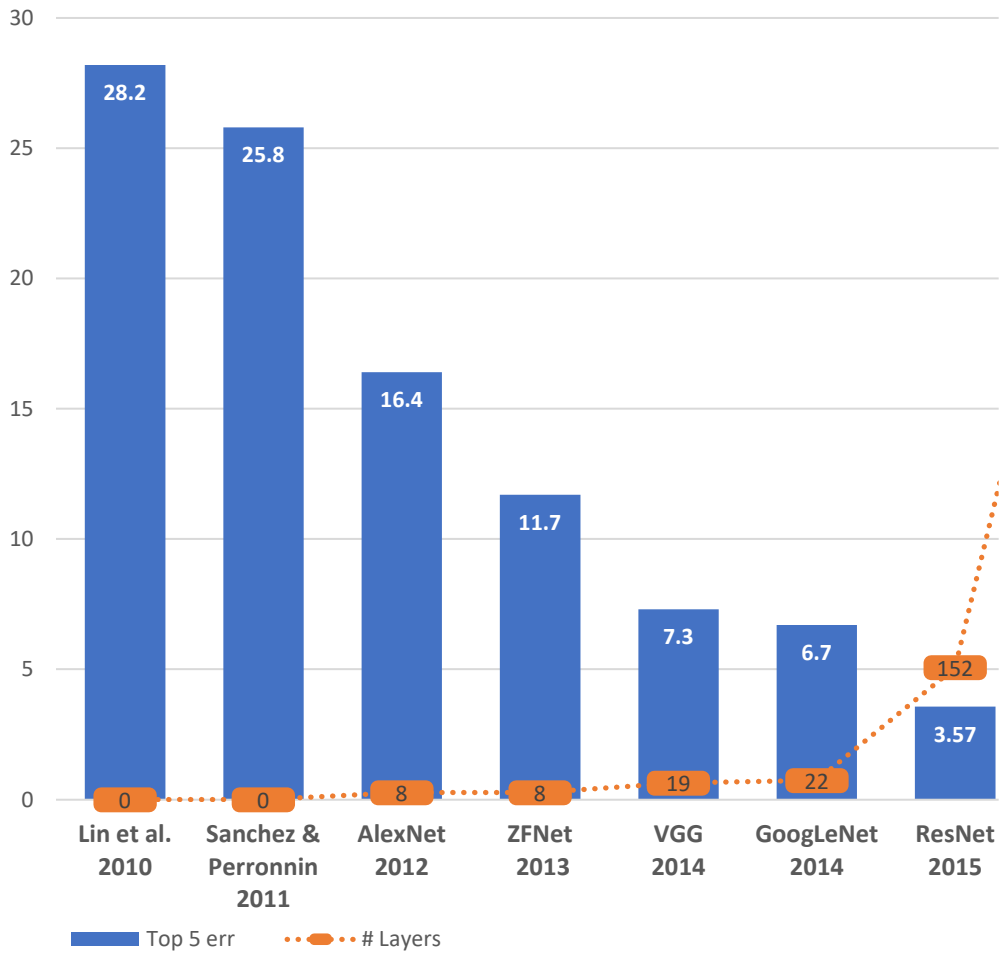
Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.

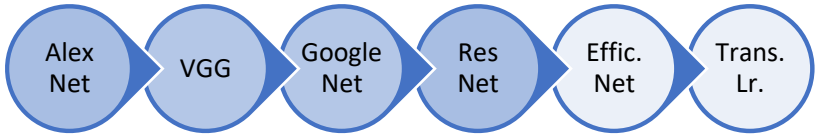
GoogLeNet



Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.

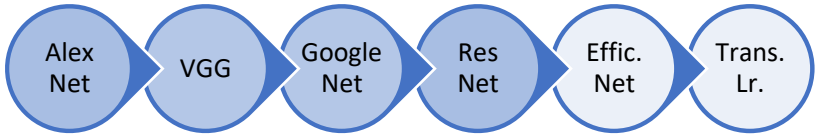
ResNet

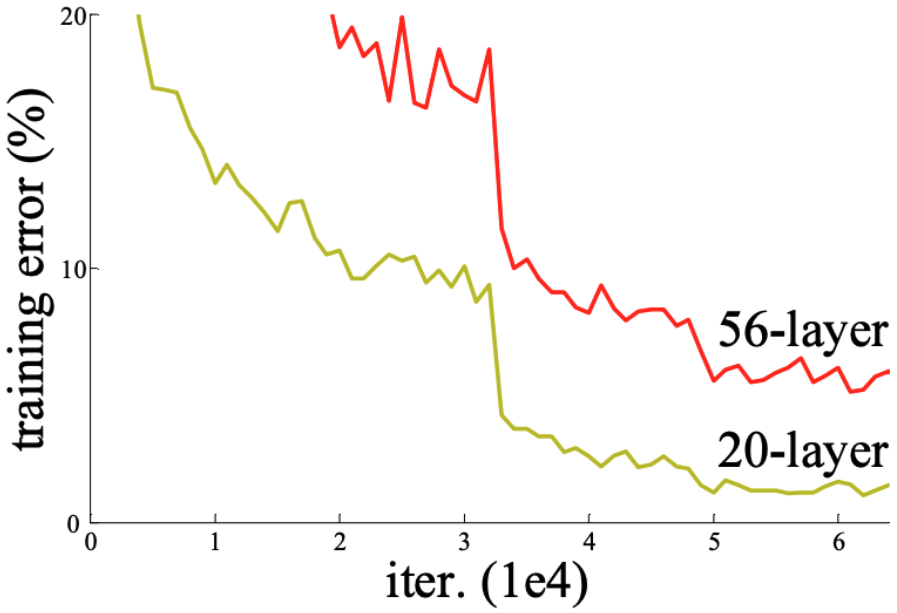


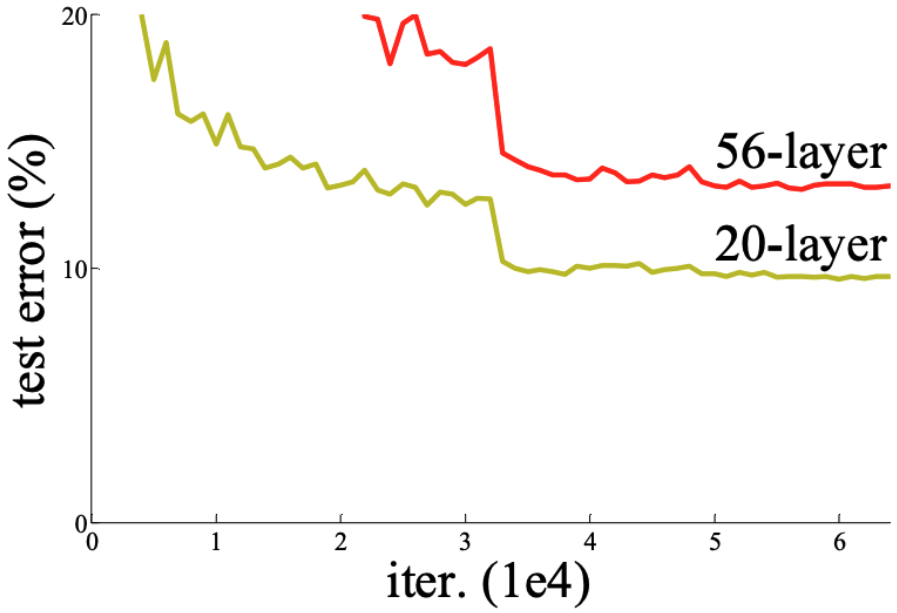


Kaiming He et al. Deep Residual Learning for Image Recognition. CVPR, 2016.

ResNet

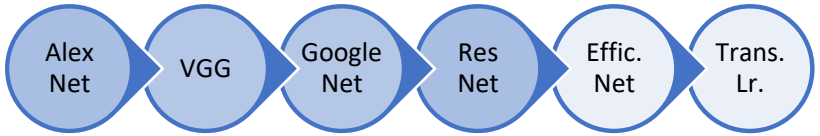


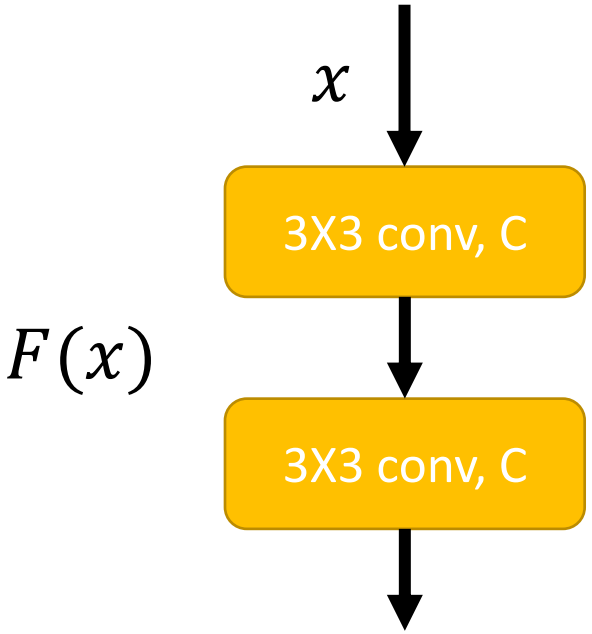




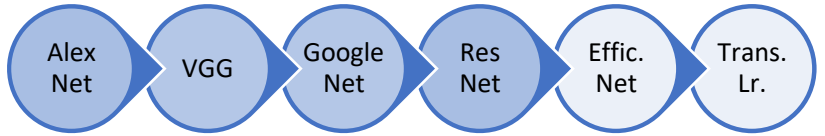
Deeper models are harder to optimize - vanishing gradients problem

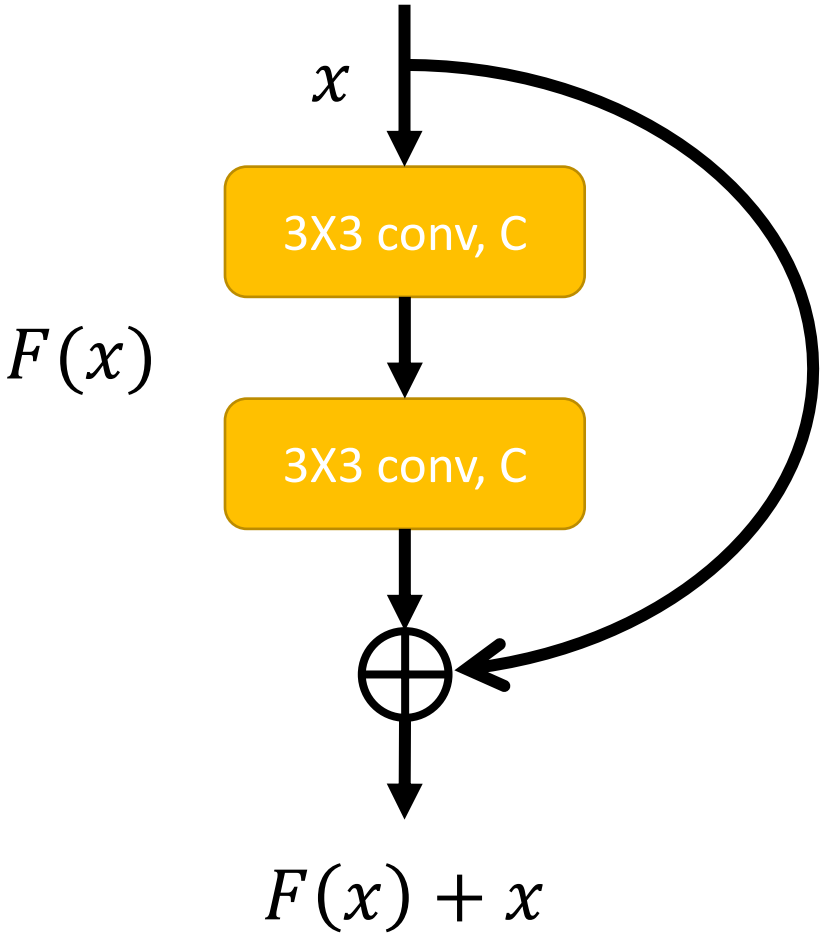
Residual Building Block



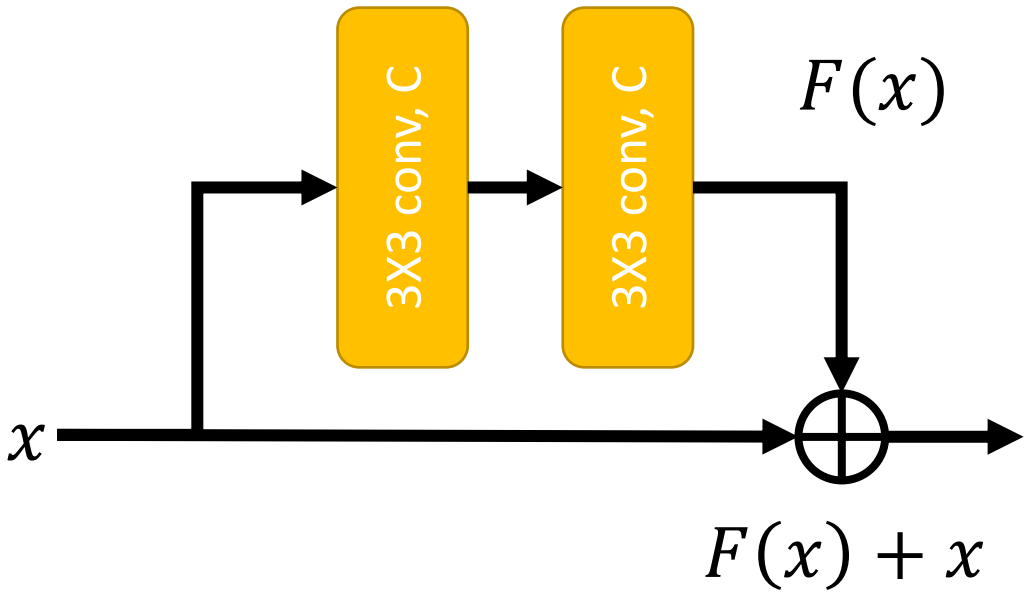


Residual Building Block



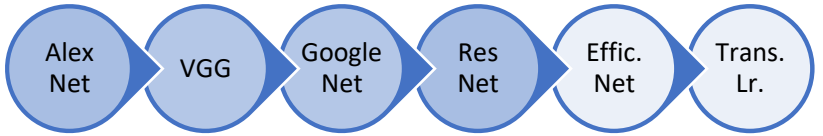


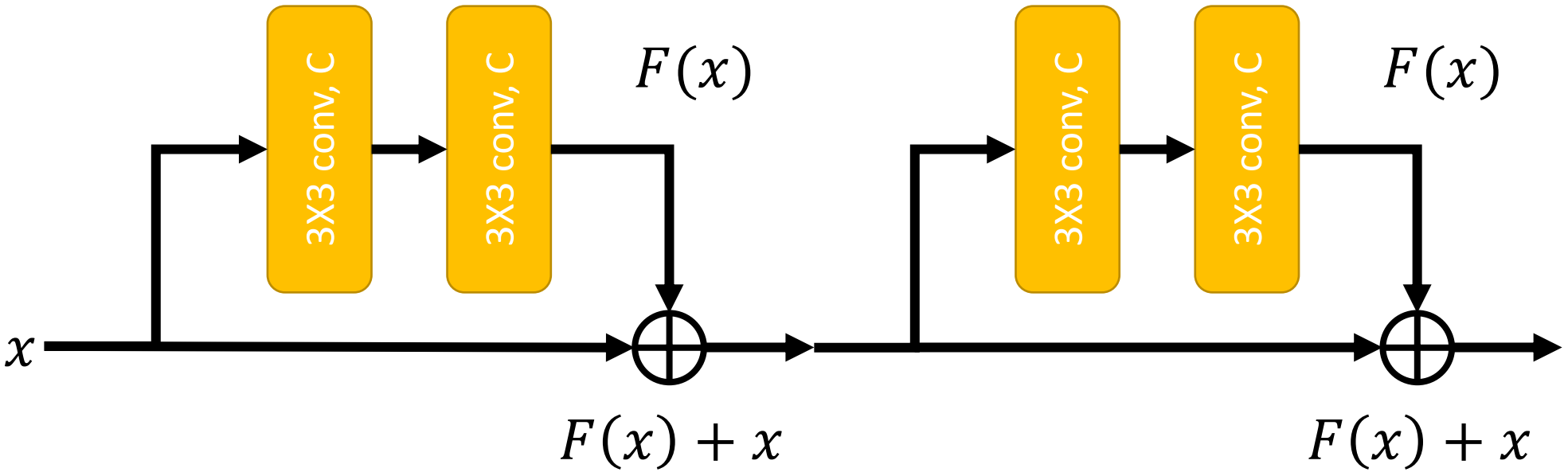
Same blocks



Use network layers to learn a residual mapping

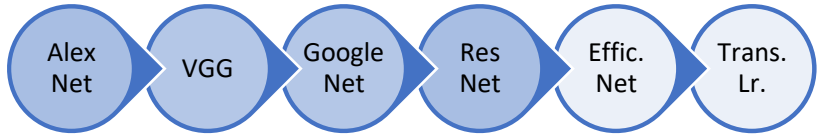
ResNet Gradient propagation

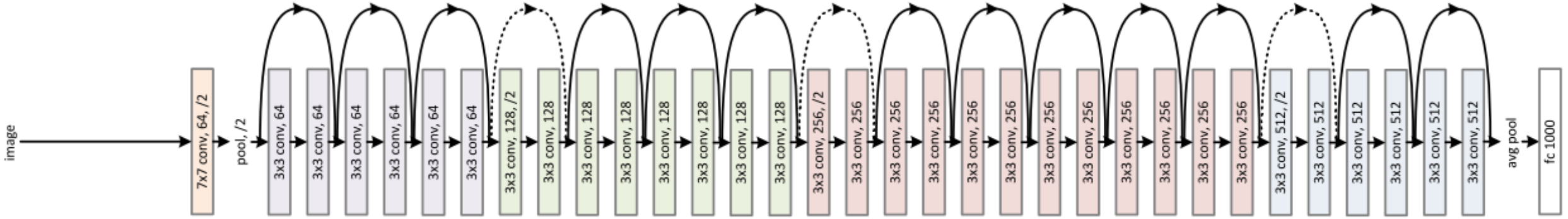




Gradient highway

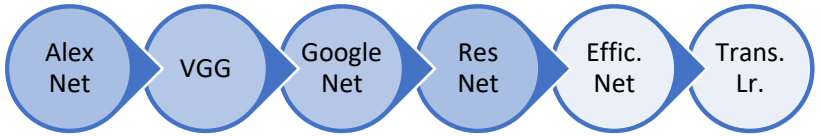
ResNet Architecture





- Residual connections help gradient propagation to initial layers
- No fully connected layers at the end
- Rapid downsampling of input

More architectures



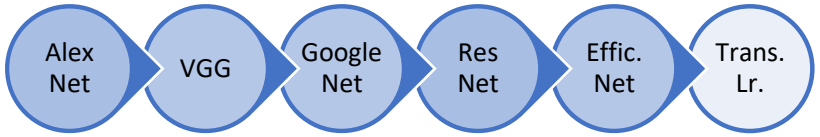
Accuracy

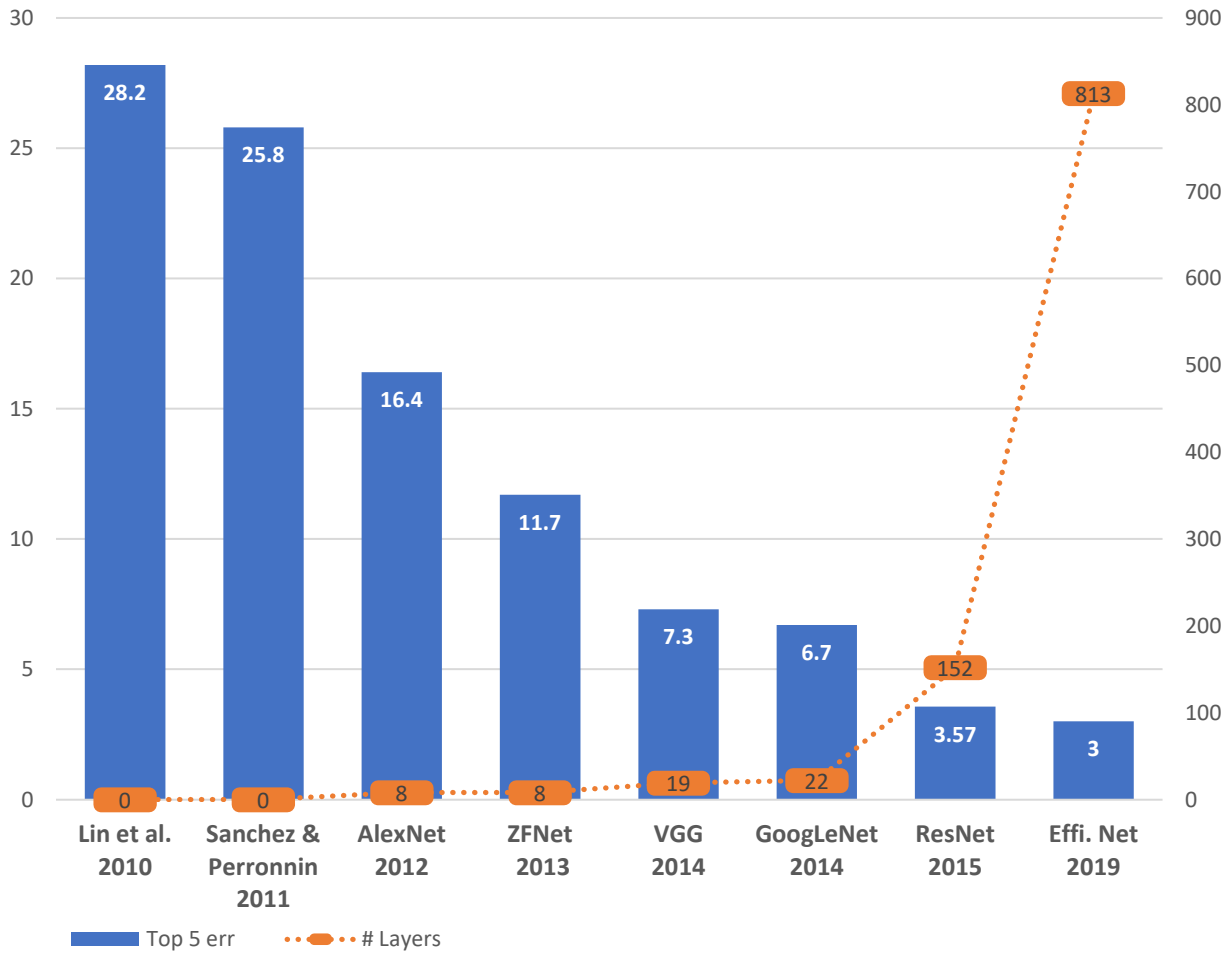
- DenseNet, Huang et al. 2017
- ResNext, Xie et al. 2017
- SENet, Hu et al. 2018

Efficiency

- MobileNet, Howard et al. 2017
- ShuffleNet, Zhang et al. 2018
- Neural Architecture Search, Zoph and Le. 2017

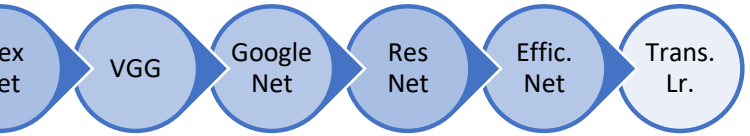
EfficientNet

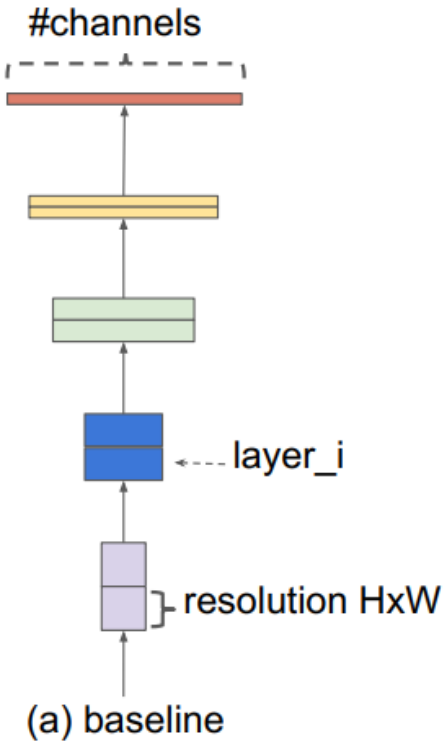




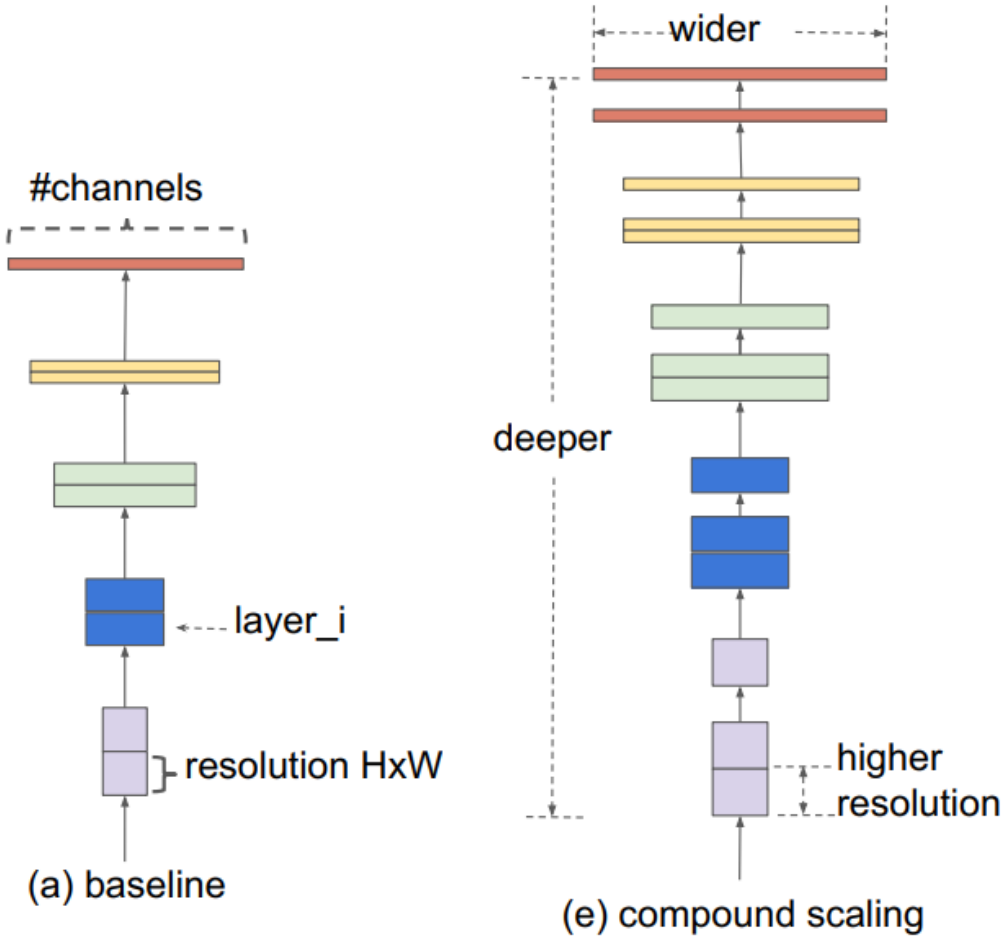
Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML, 2019.

EfficientNet





EfficientNet



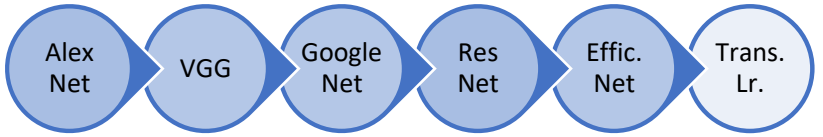
If we want to use 2^N more resourses (FLOPs):

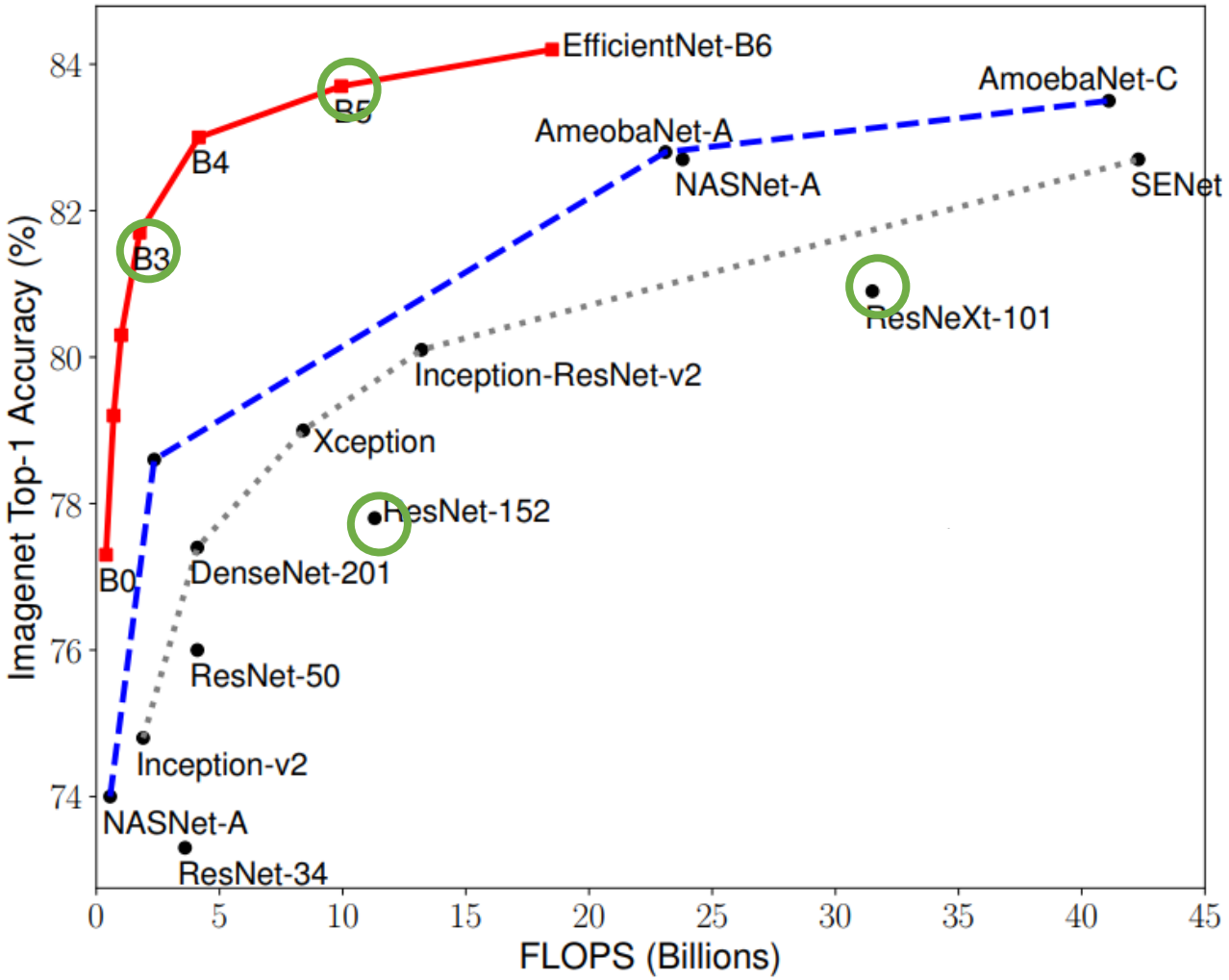
- Increase depth by α^N
- Increase width by β^N
- Increase input resolution by γ^N

α, β, γ – constant scaling coefficients determined by a small grid search on the original small model

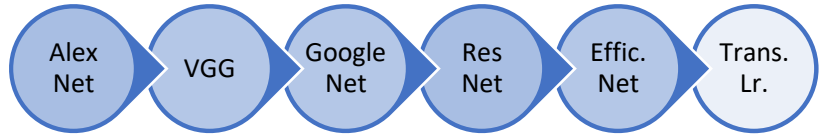
(they found $\alpha = 1.2,\ \beta = 1.1, \gamma = 1.15$)

Results


Alex Net
VGG
Google Net
Res Net
Effic. Net
Trans. Lr.

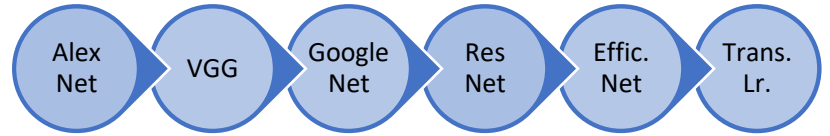

EfficientNet-B6
AmoebaNet-C
AmeobaNet-A
NASNet-A
SENet
B4
B3
ResNeXt-101
Inception-ResNet-v2
Xception
ResNet-152
B0
DenseNet-201
ResNet-50
Inception-v2
NASNet-A
ResNet-34
Imagenet Top-1 Accuracy (%)
FLOPS (Billions)
84
82
80
78
76
74
0
5
10
15
20
25
30
35
40
45

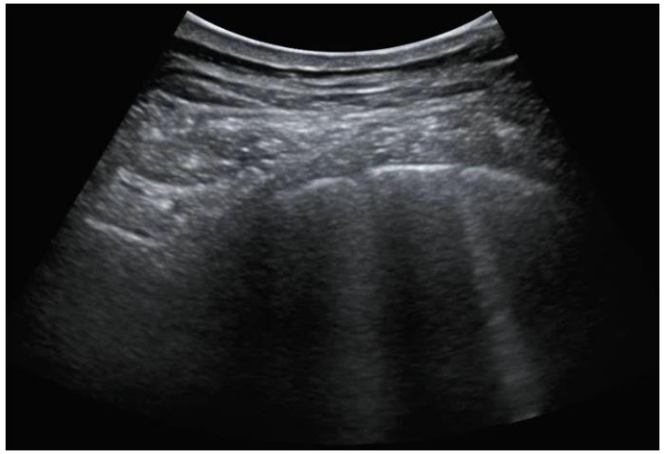
Architectures summary

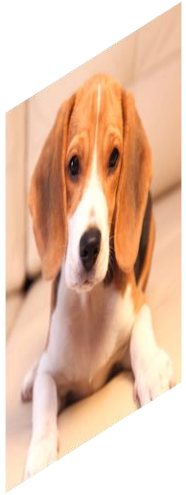


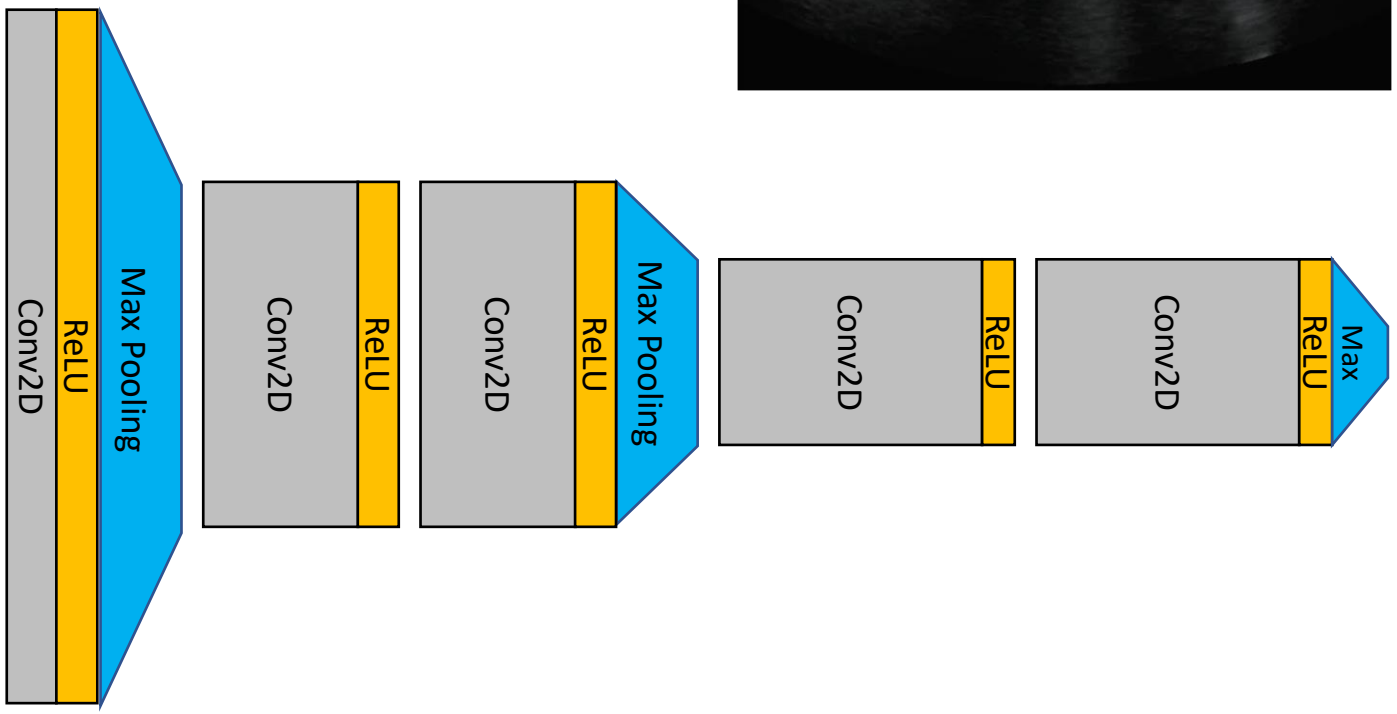
- Design your network according to your task and resources
- Take into consideration
  - # Parameters
  - # FLOPs
  - Memory Size
- Use skip connections :)
- Use **existing architectures** when possible
- Use **pre-trained models** when possible

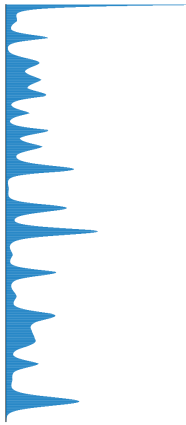
Transfer Learning





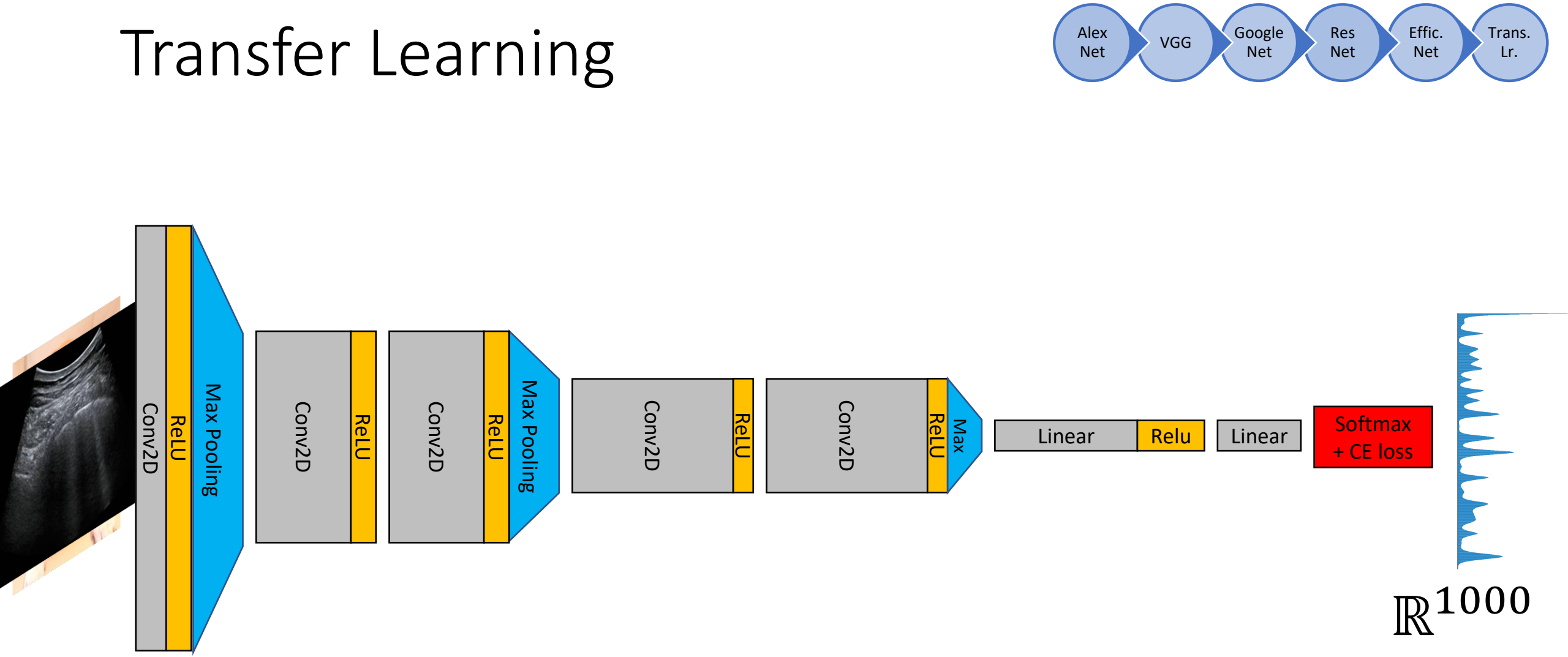




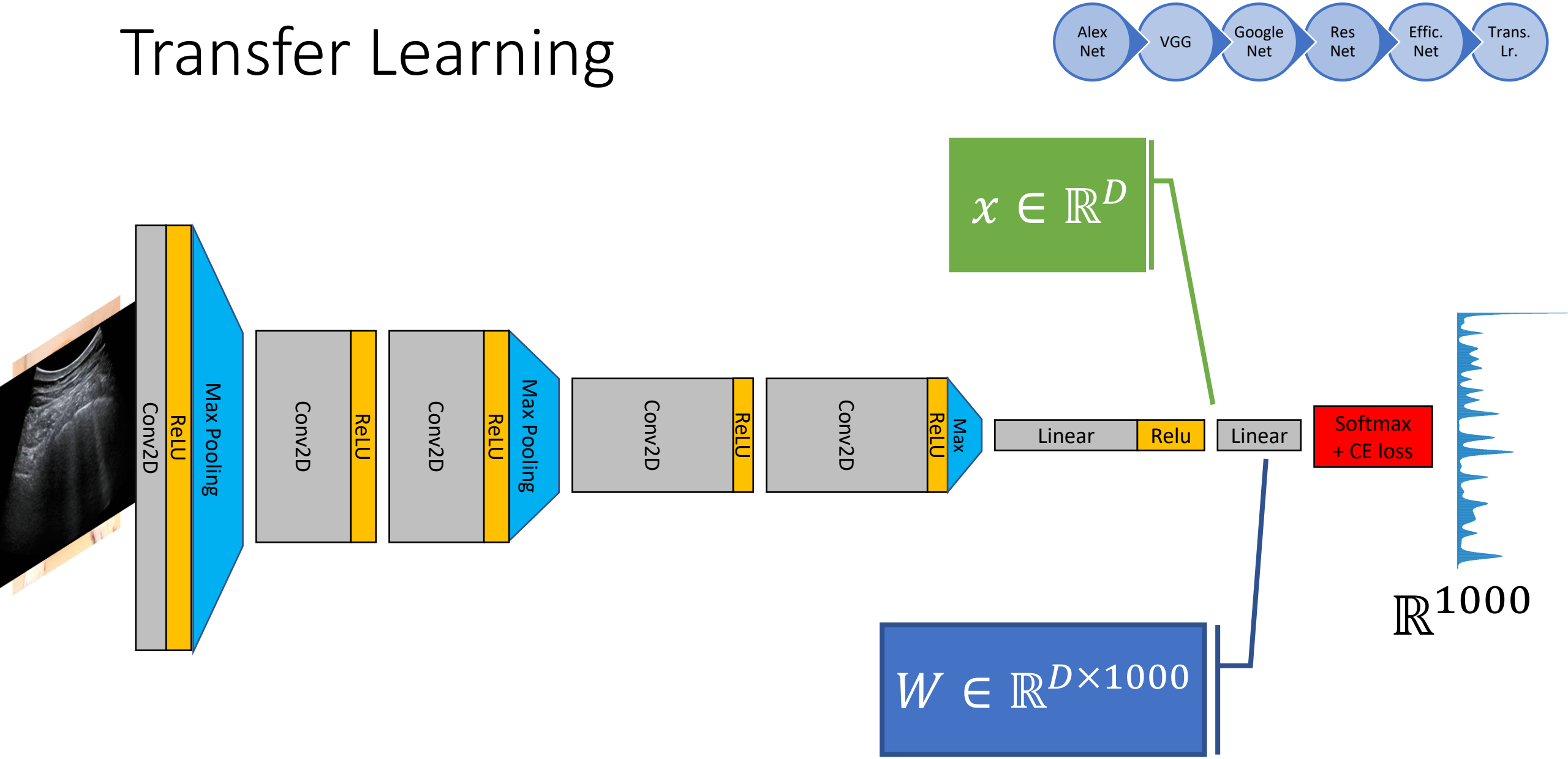


$\mathbb{R}^{1000}$

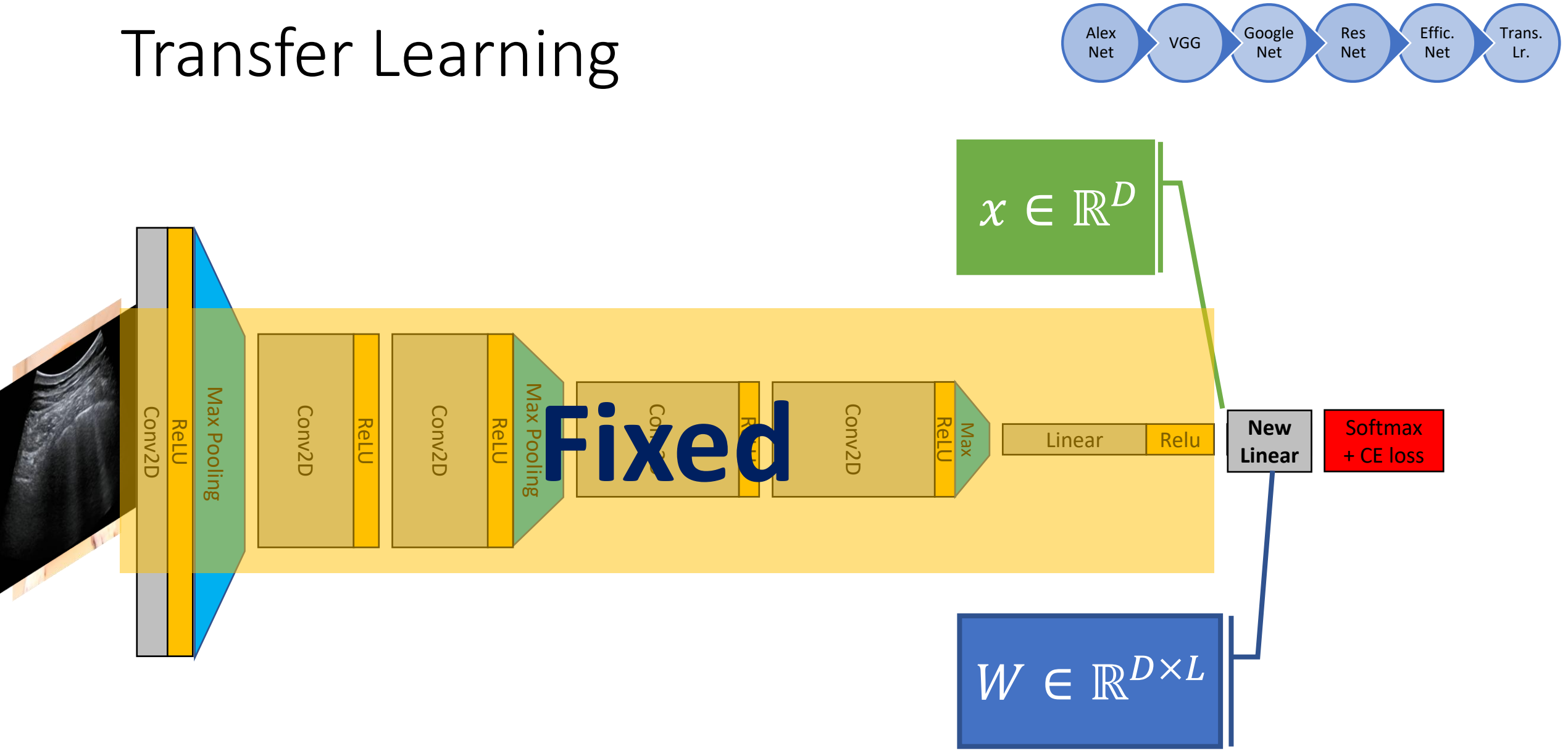
Transfer Learning



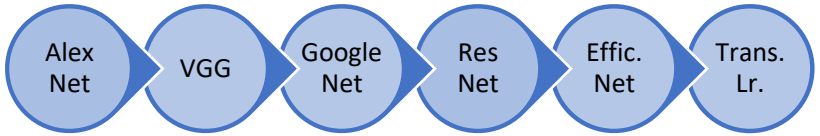
Transfer Learning



Transfer Learning



Transfer Learning



- In practice:

```python
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```

Questions?