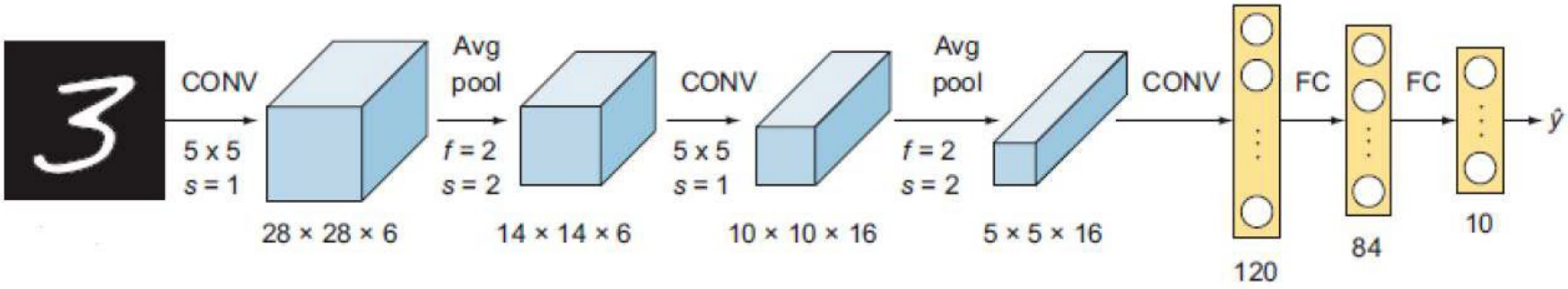
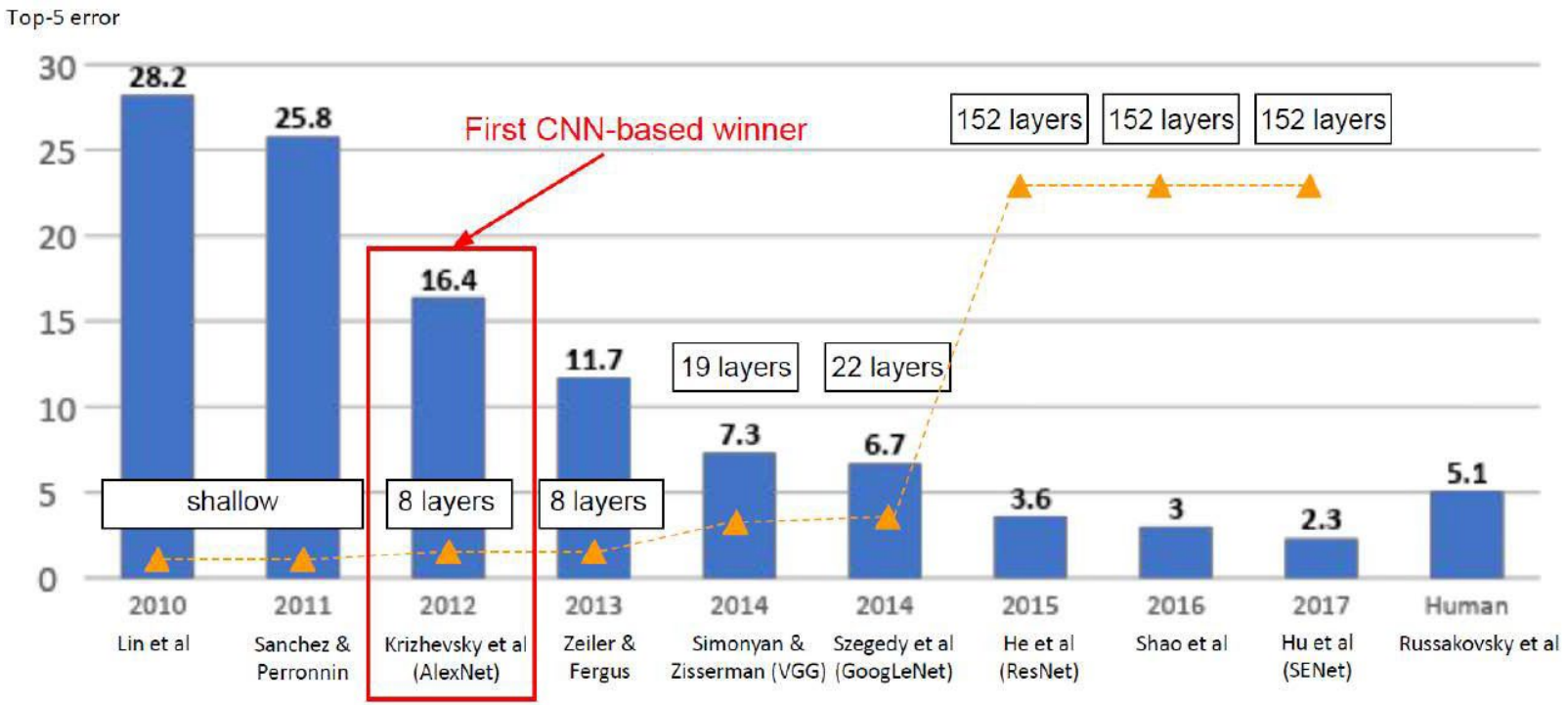
CNN

LeNet



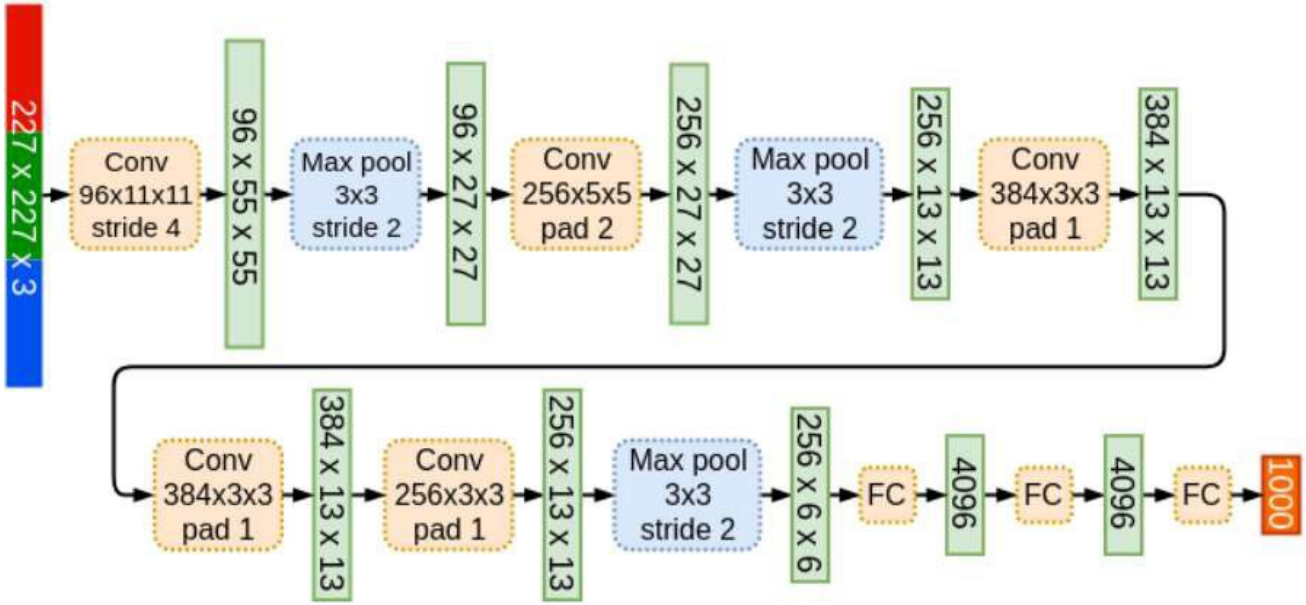
- Conv filters were 5x5 , applied at stride 1
- Subsampling (Pooling) layers were 2x2 applied at stride 2

AlexNet



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

AlexNet

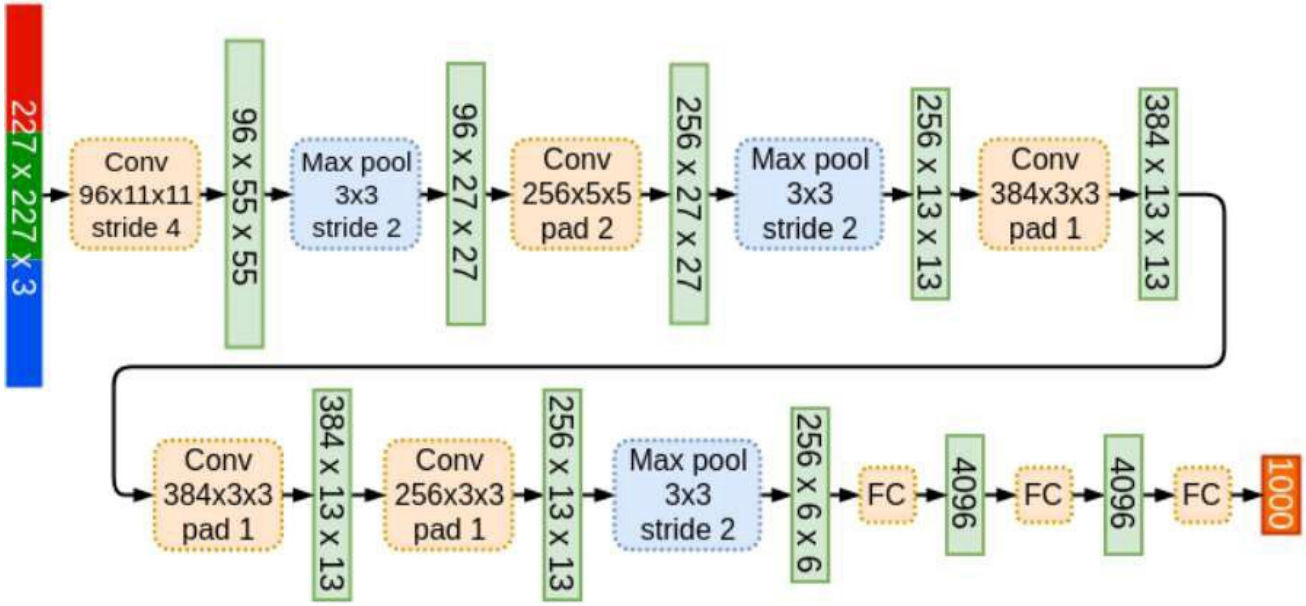


(CONV1): 96 11x11 filters applied at stride 4

Why is the output volume size 55?

What is the total number of parameters in the first layer?

AlexNet



(POOL1): 3×3 filters applied at stride 2

Why is the output volume size 27?

What is the number of parameters?

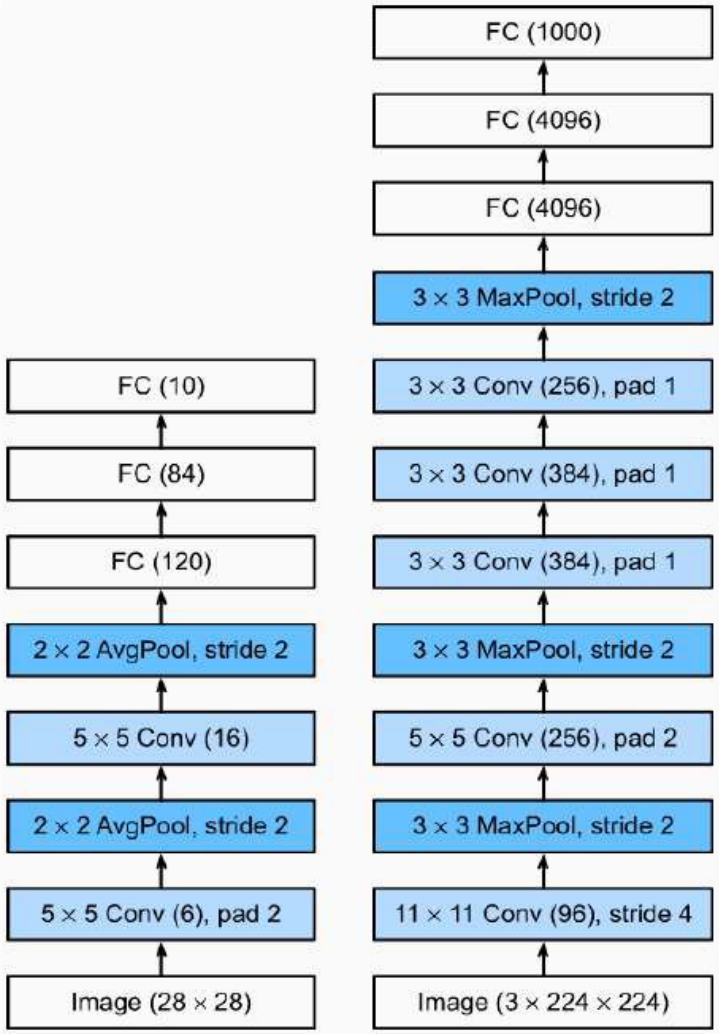
How is AlexNet Different from LeNet?

62M parameters vs 61K parameters

ReLU vs Sigmoid (why?)

Use of Regularization Techniques (Dropout, Weight Decay) (why?)

Use of Normalization (why?)



LeNet (left) to AlexNet (right)

VGG Net

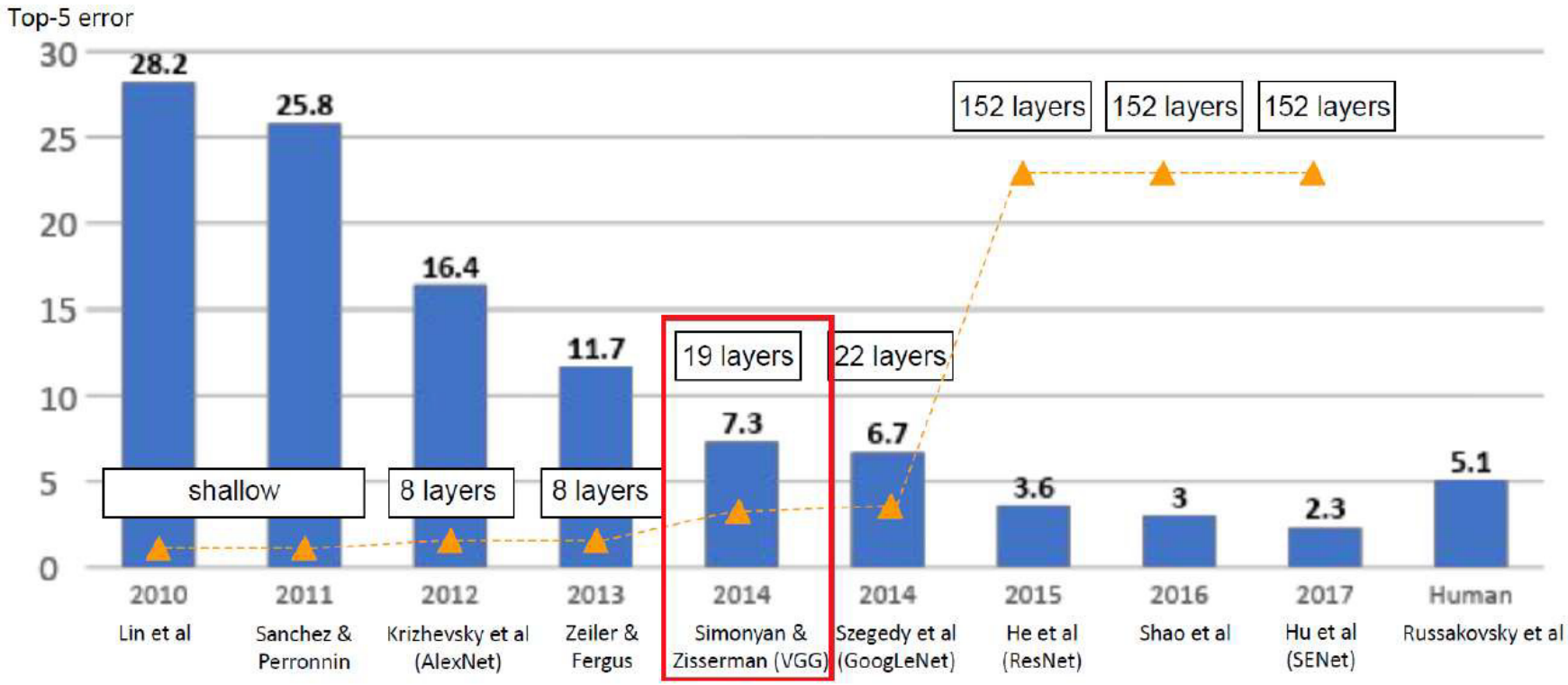
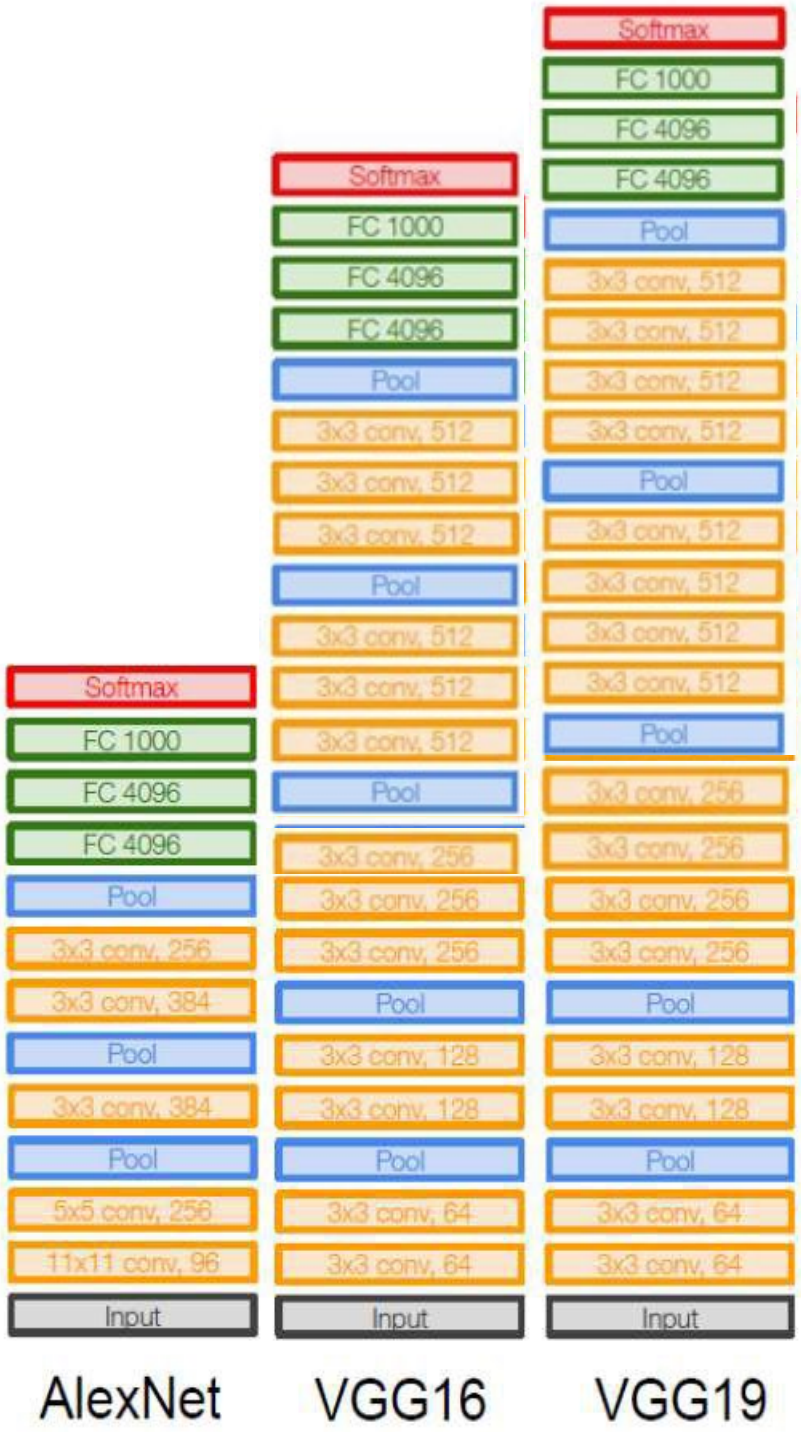


Figure: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

VGGNet

Do you see any difference?

- Smaller filters
- Deeper networks
- Only 3×3 CONV with stride 1, pad 1 and 2×2 MAX POOL with stride 2



VGGNet

Why use smaller filters? (3×3 conv)

- Stack of three 3×3 conv (stride 1) layers has same effective receptive field as one 7×7 conv layer
- Deeper network means more non-linearities which leads to more capacity

GoogLeNet

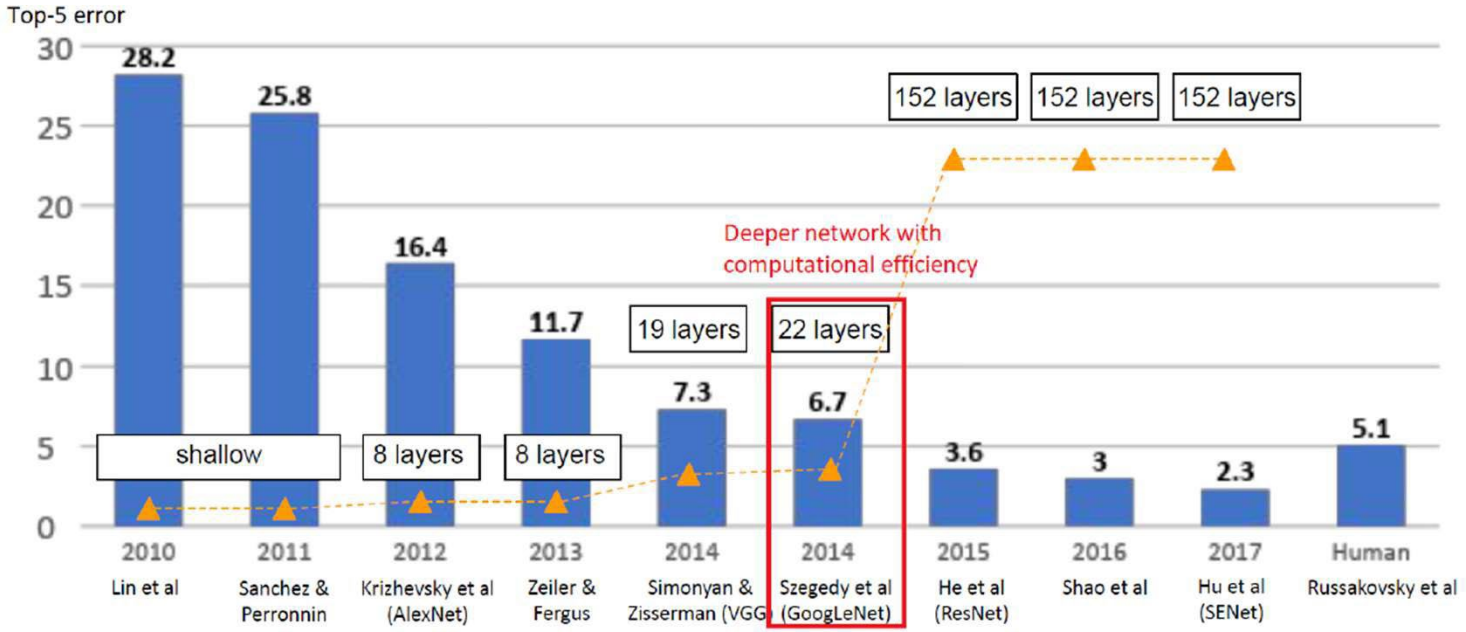
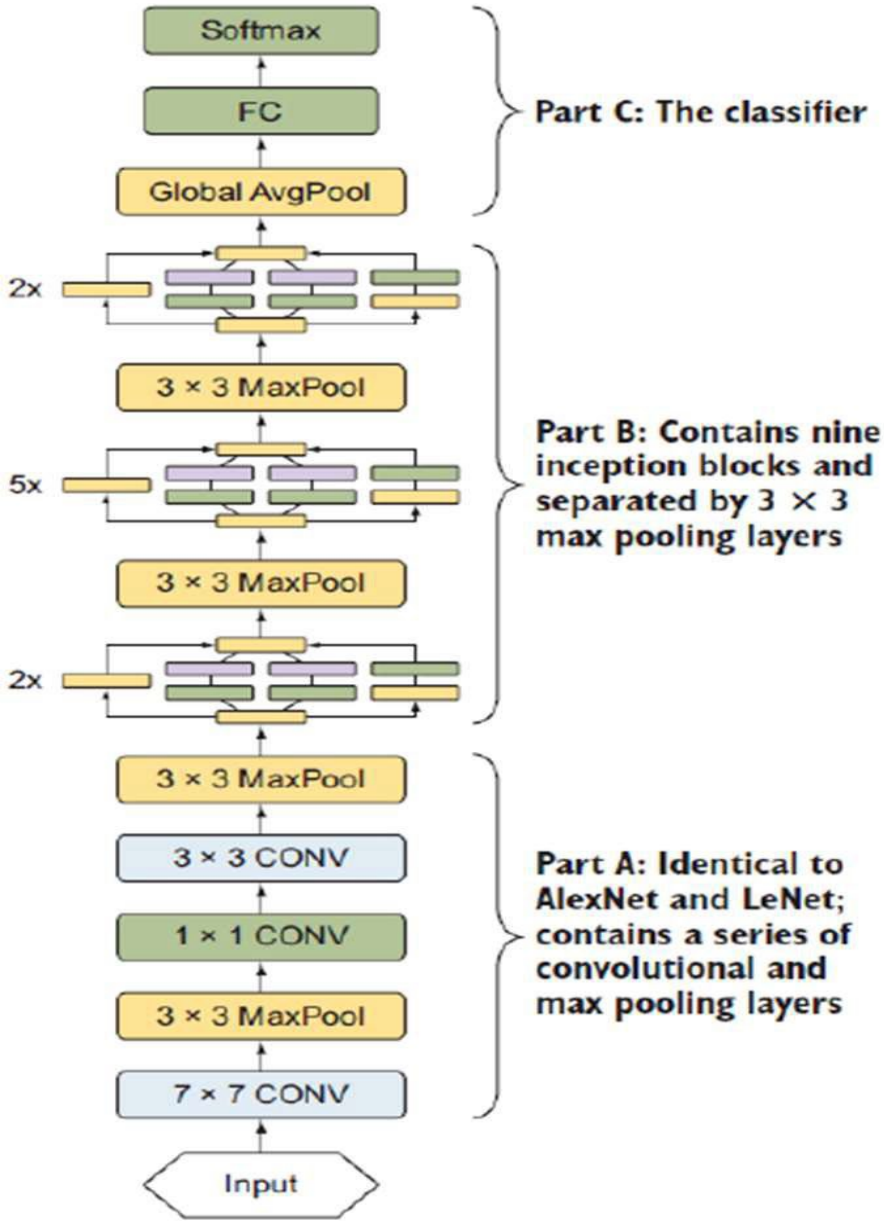


Figure: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

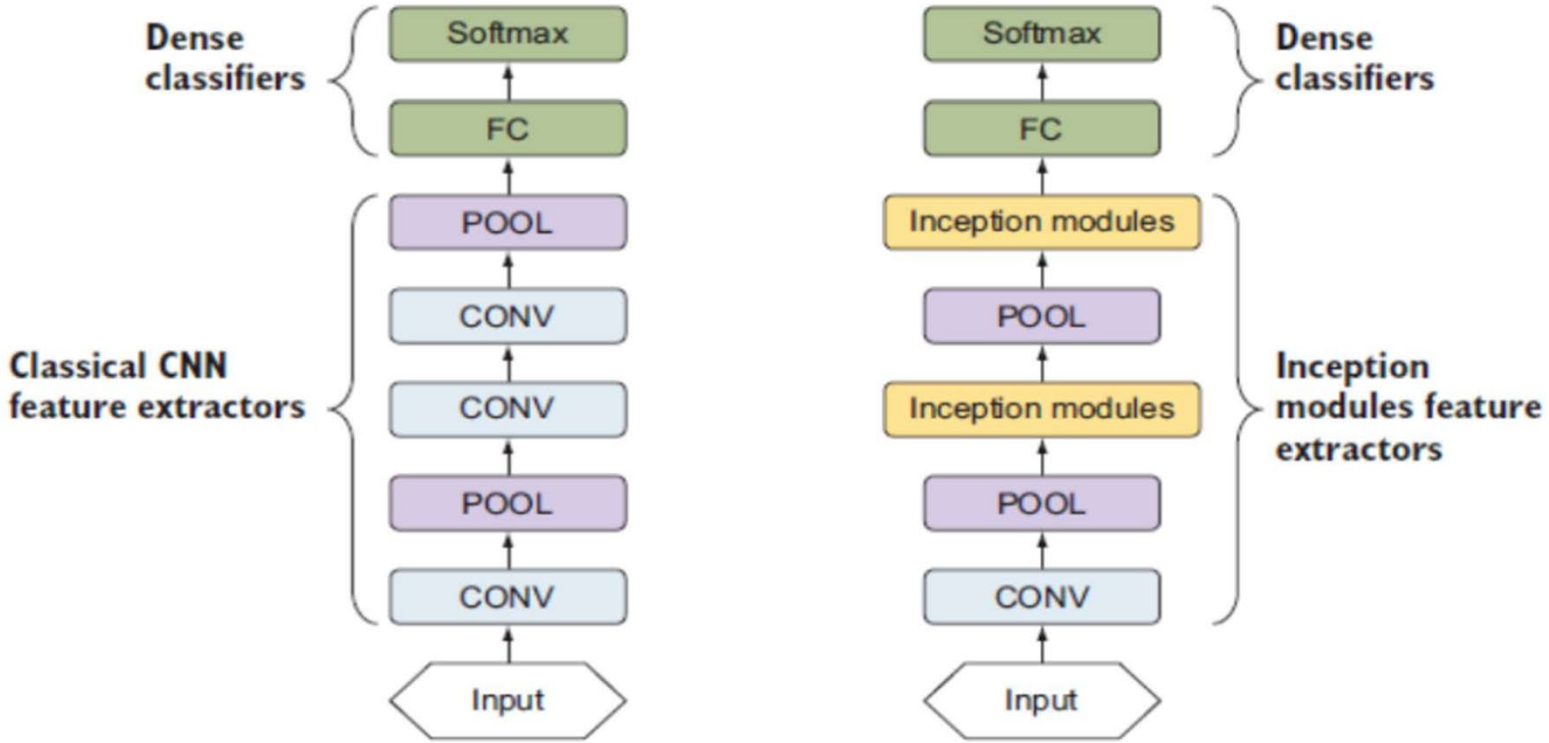
GoogleNet

- Only 5 million parameters! (12x less than AlexNet and 27x less than VGG-16)
- Efficient "Inception" module
- No longer multiple expensive FC layers



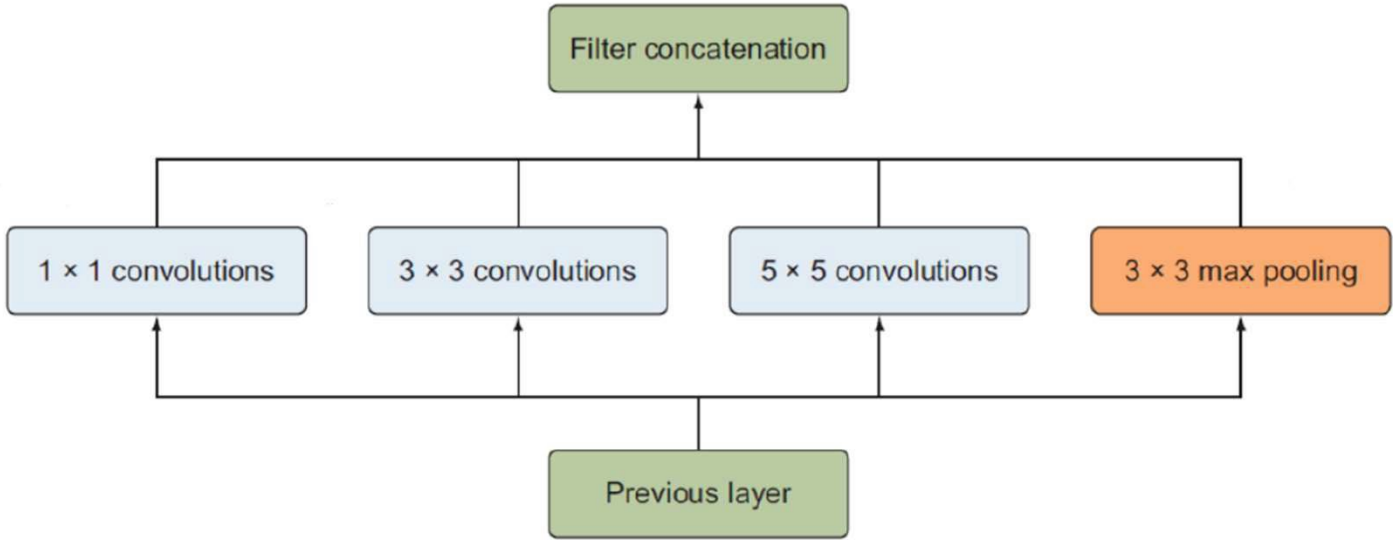
GoogLeNet

Inception modules instead of classical CNNs for feature extraction

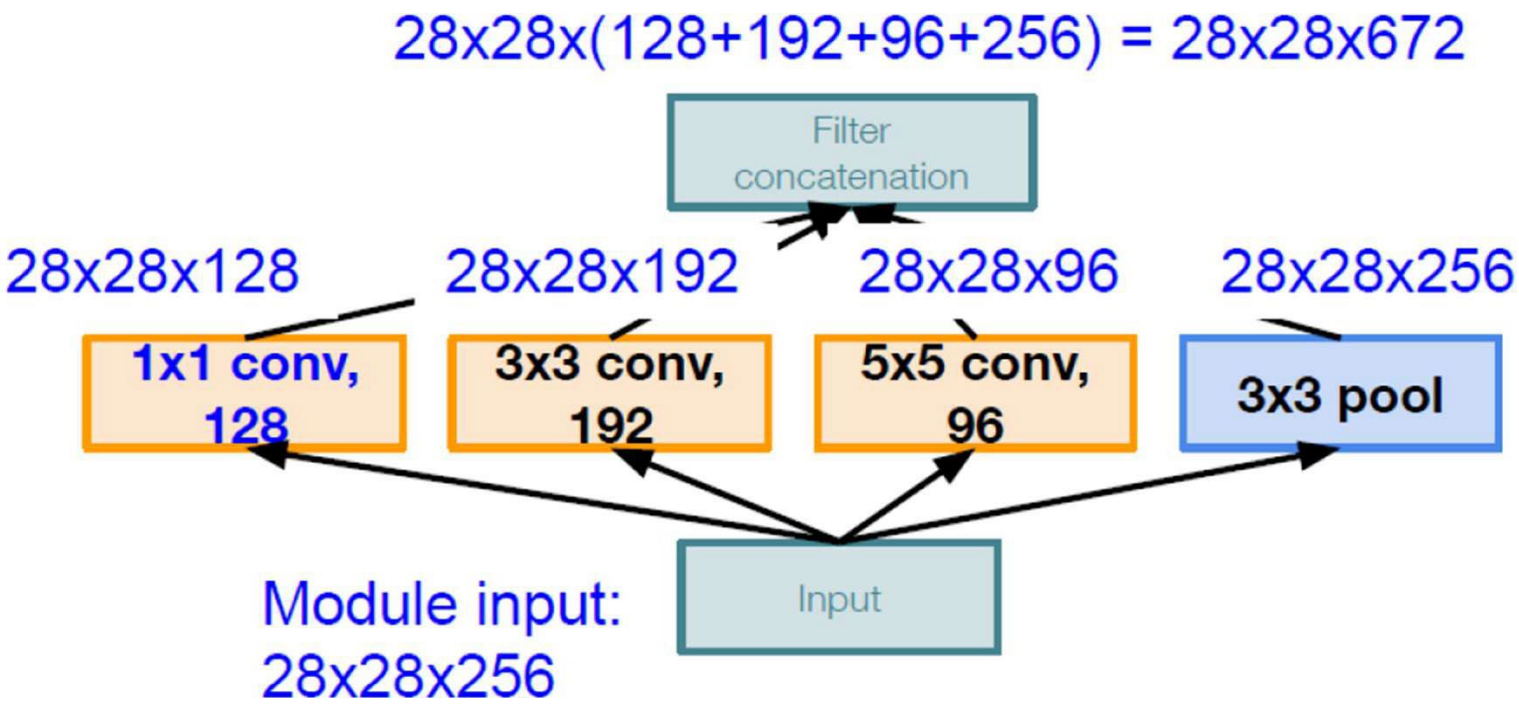


Naïve Inception Module

- Apply parallel filter operations on the input from previous layer
- Multiple receptive field sizes for convolution ($1 \times 1, 3 \times 3, 5 \times 5$)
- Pooling operation (3×3)
- Concatenate all filter outputs together channel-wise

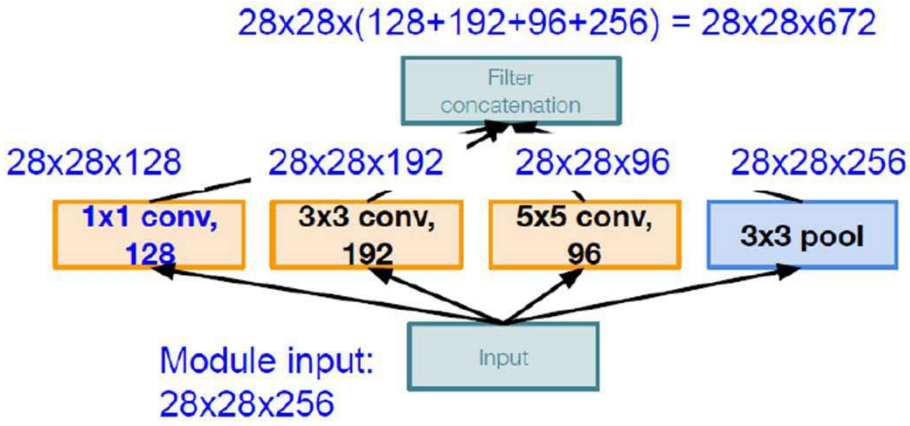


Naïve inception module



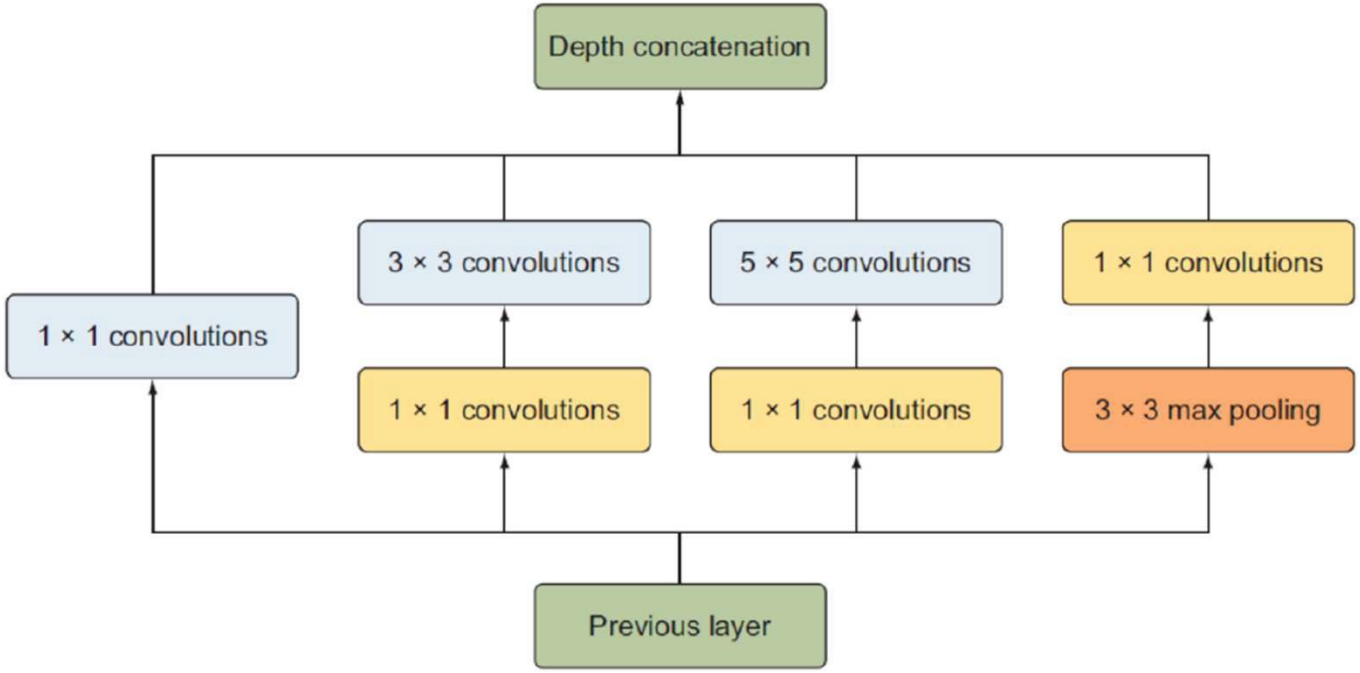
Naïve inception module

- What is the problem with this?
- Computational complexity!
  - Conv Ops:
  - $[1 \times 1 \text{ conv}, 128]\ 28 \times 28 \times 128 \times 1 \times 1 \times 256$
  - $[3 \times 3 \text{ conv}, 192]\ 28 \times 28 \times 192 \times 3 \times 3 \times 256$
  - $[5 \times 5 \text{ conv}, 96]\ \ 28 \times 28 \times 96 \times 5 \times 5 \times 256$
  - Total: 854M ops



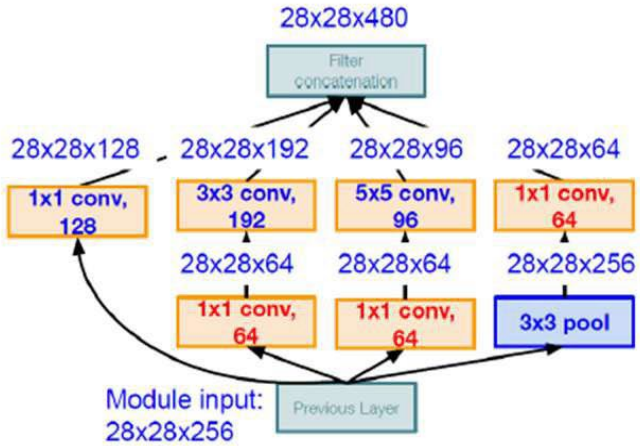
Inception module

- Any Solution?
- "bottleneck" layers that use 1×1 convolutions to reduce feature channel size



Dimension reduction in inception module

- Using same parallel layers as naive example, and adding "1×1 conv, 64 filter" bottlenecks
  - $[1 \times 1 \text{ conv}, 64]\ 28 \times 28 \times 64 \times 1 \times 1 \times 256$
  - $[1 \times 1 \text{ conv}, 64]\ 28 \times 28 \times 64 \times 1 \times 1 \times 256$
  - $[1 \times 1 \text{ conv}, 128]\ 28 \times 28 \times 128 \times 1 \times 1 \times 256$
  - $[3 \times 3 \text{ conv}, 192]\ 28 \times 28 \times 192 \times 3 \times 3 \times 64$
  - $[5 \times 5 \text{ conv}, 96]\ 28 \times 28 \times 96 \times 5 \times 5 \times 64$
  - $[1 \times 1 \text{ conv}, 64]\ 28 \times 28 \times 64 \times 1 \times 1 \times 256$
  - Total: 358M ops
- Compared to 854M ops for naive version Bottleneck can also reduce depth after pooling layer



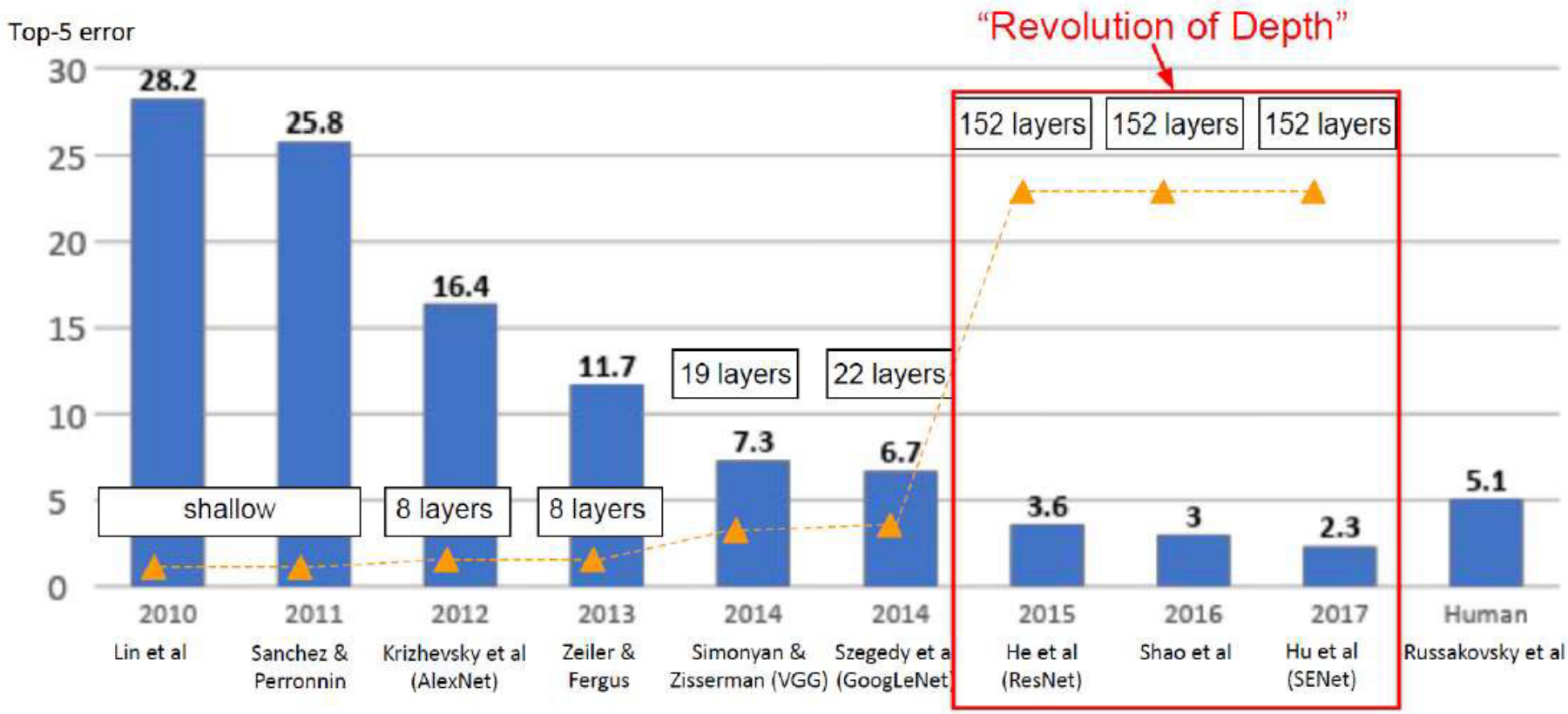
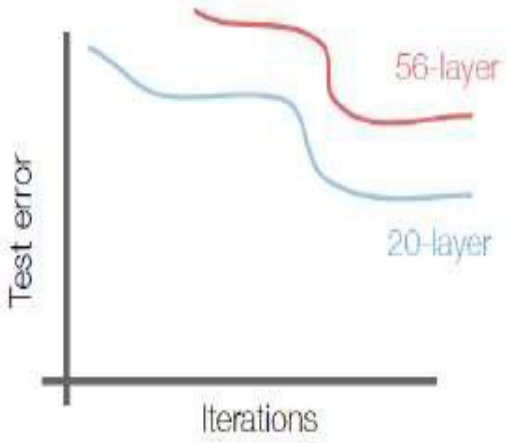
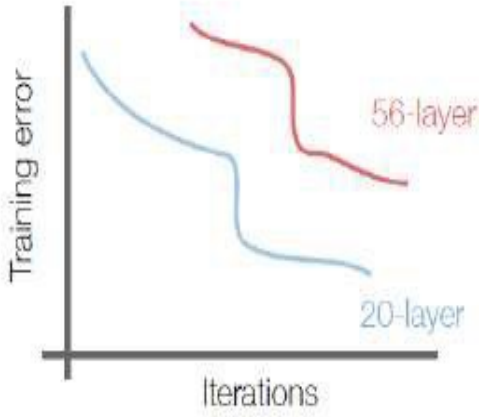


Figure: ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

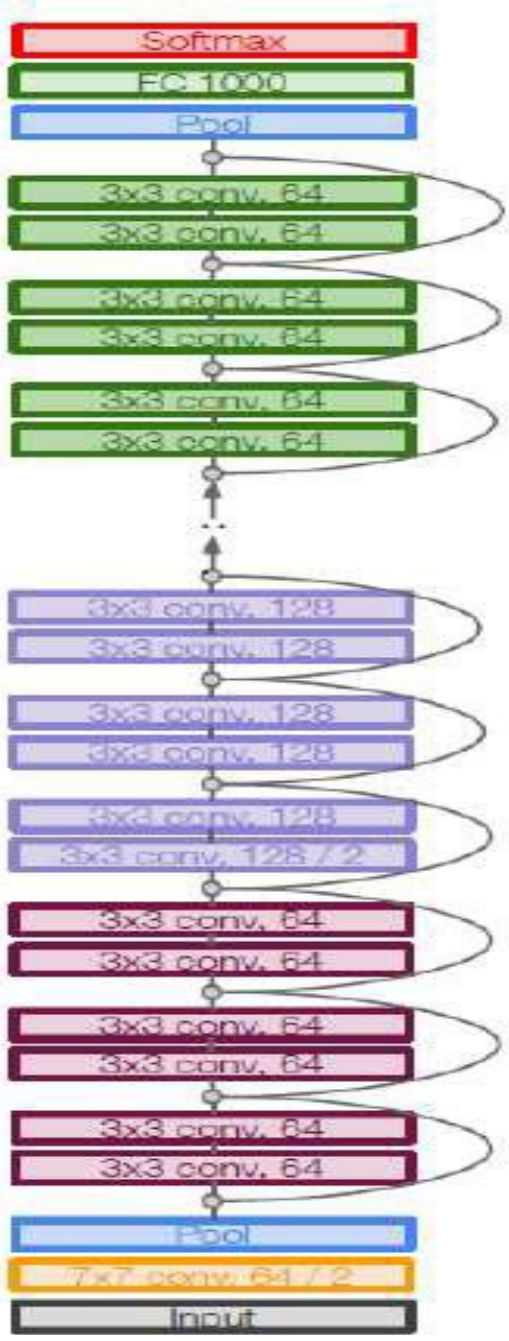
ResNet

- A very deep network using residual connections
- What happens when we continue stacking deeper layers on a "plain" convolutional



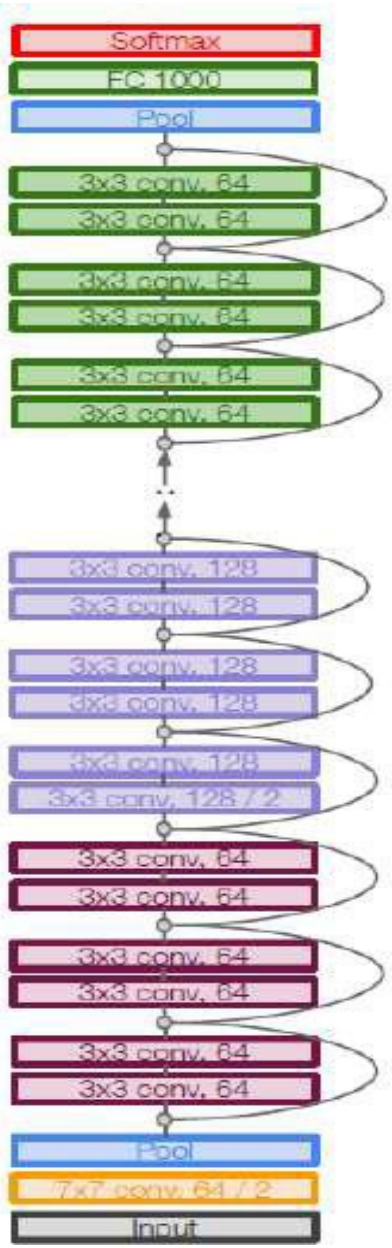


- The deeper model performs worse, but it's not caused by overfitting!



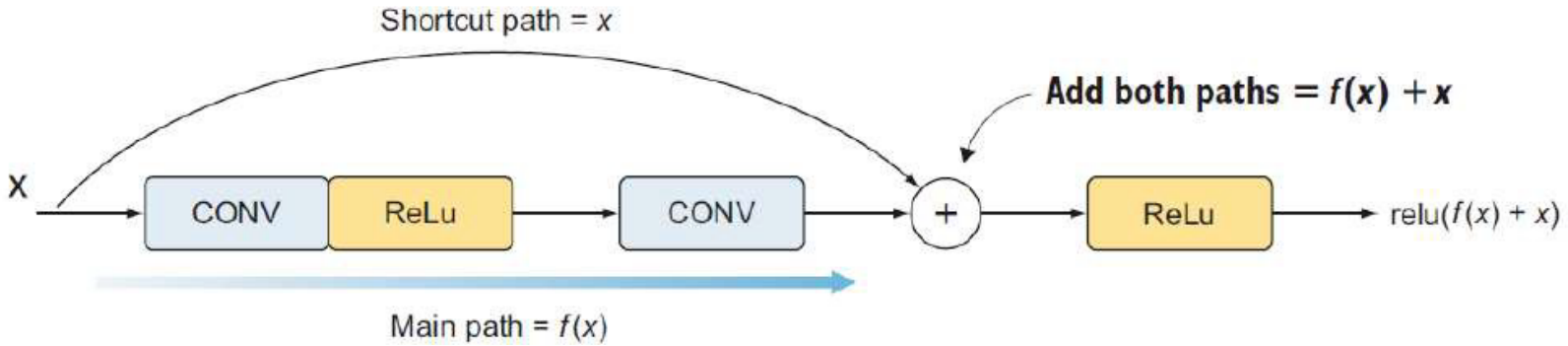
Res Net

- Fact: Deep models have more representation power (more parameters) than shallower models.
- Hypothesis: the problem is an optimization problem, deeper models are harder to optimize



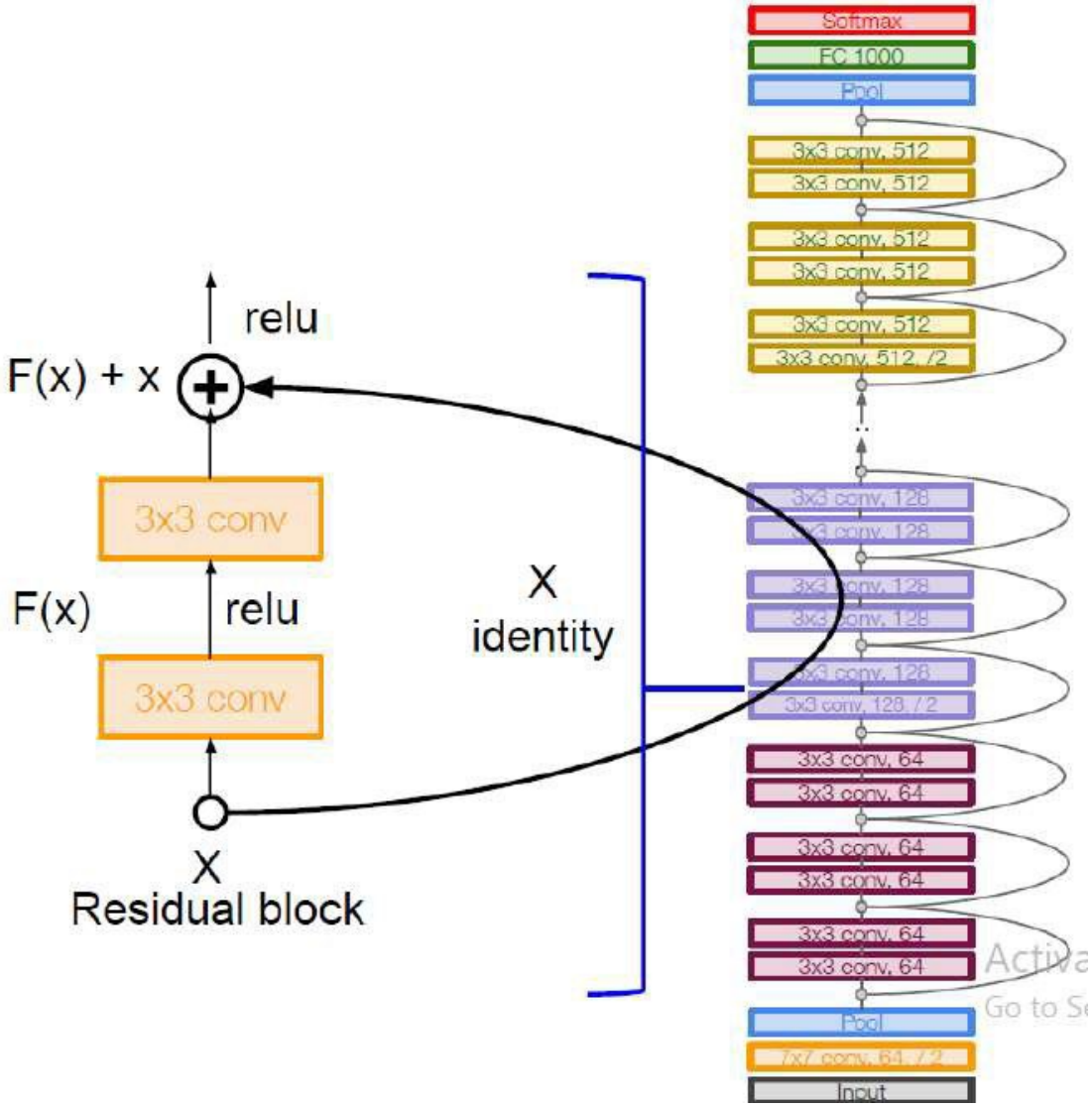
Skip Connection

Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet

- Stack residual blocks
- Every residual block has two 3×3 conv layers
- Periodically, double size of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers besides FC 1000 to output classes
- Global average pooling layer after last conv layer
- Batch Normalization after every CONV layer
- No dropout used



Training ResNet

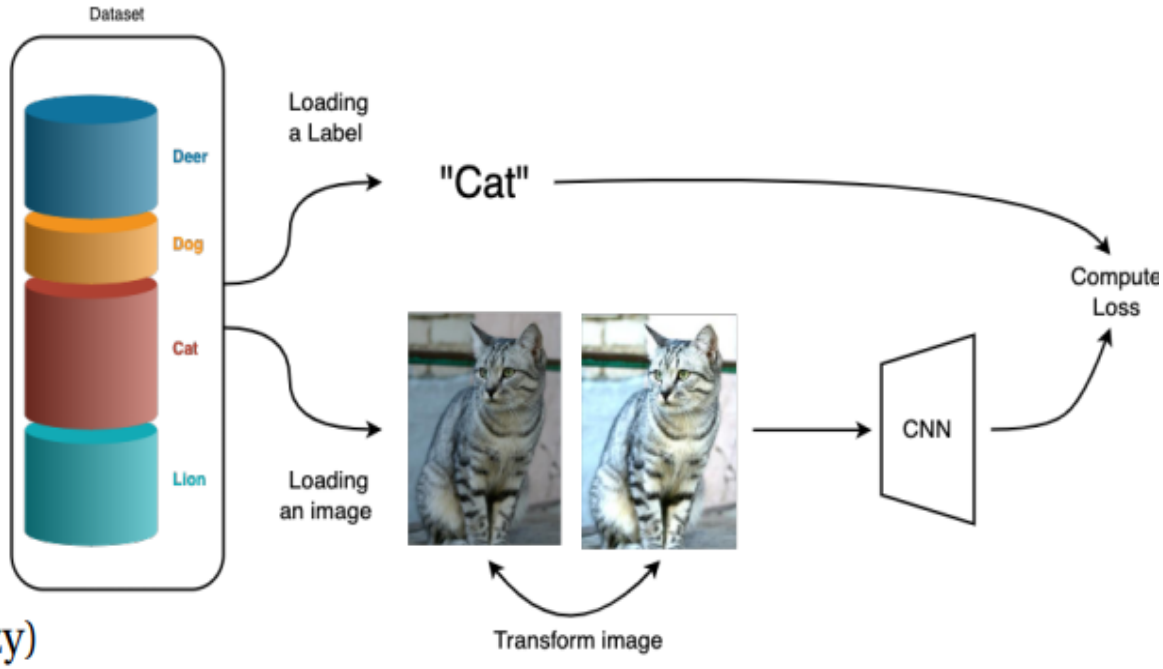
- Batch Normalization after every CONV layer
- Xavier initialization from He, et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Key points

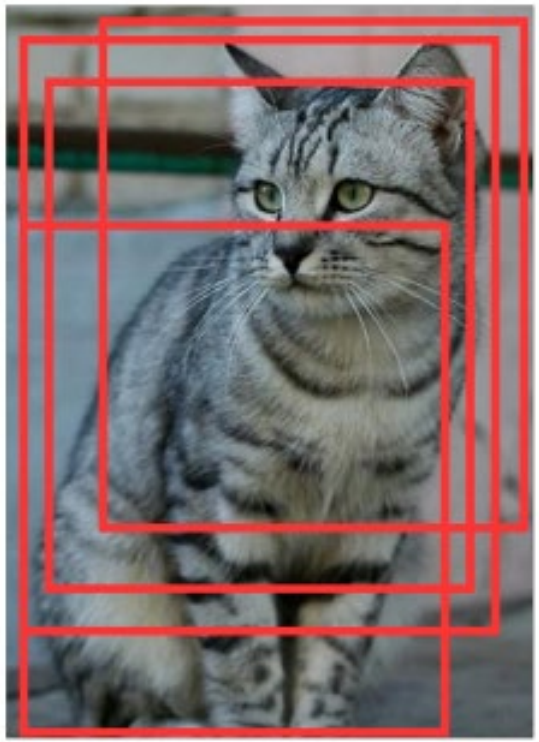
- **AlexNet** showed that you can use CNNs to train Computer Vision models.
- **ResNet** showed us how to train extremely deep networks.
  - → Limited only by GPU & memory!
  - → Showed diminishing returns as networks got bigger (increasing the depth or size of the network doesn't proportionally improve its performance)
- After ResNet: CNNs were better than the human metric and focus shifted to other topics:
  - ⋆ Efficient Networks: **MobileNet, ShuffleNet**
  - ⋆ **Neural Architecture Search** can now automate architecture design

Data Agumentation

- Getting more training data is often expensive
- But, we can often generate additional training examples from existing datasets.
- Transform each input data example in such a way that the label stays the same
- Benefits:
  - Reduces the risk of overfitting
  - Improves generalization
  - Acts as a regularization
- Examples of data augmentations
  - Translation
  - Flip (Horizontal/Vertical)
  - Rotation
  - Stretching
  - Shearing
  - Lens distortions, … (going crazy)



Random Crops and Scales

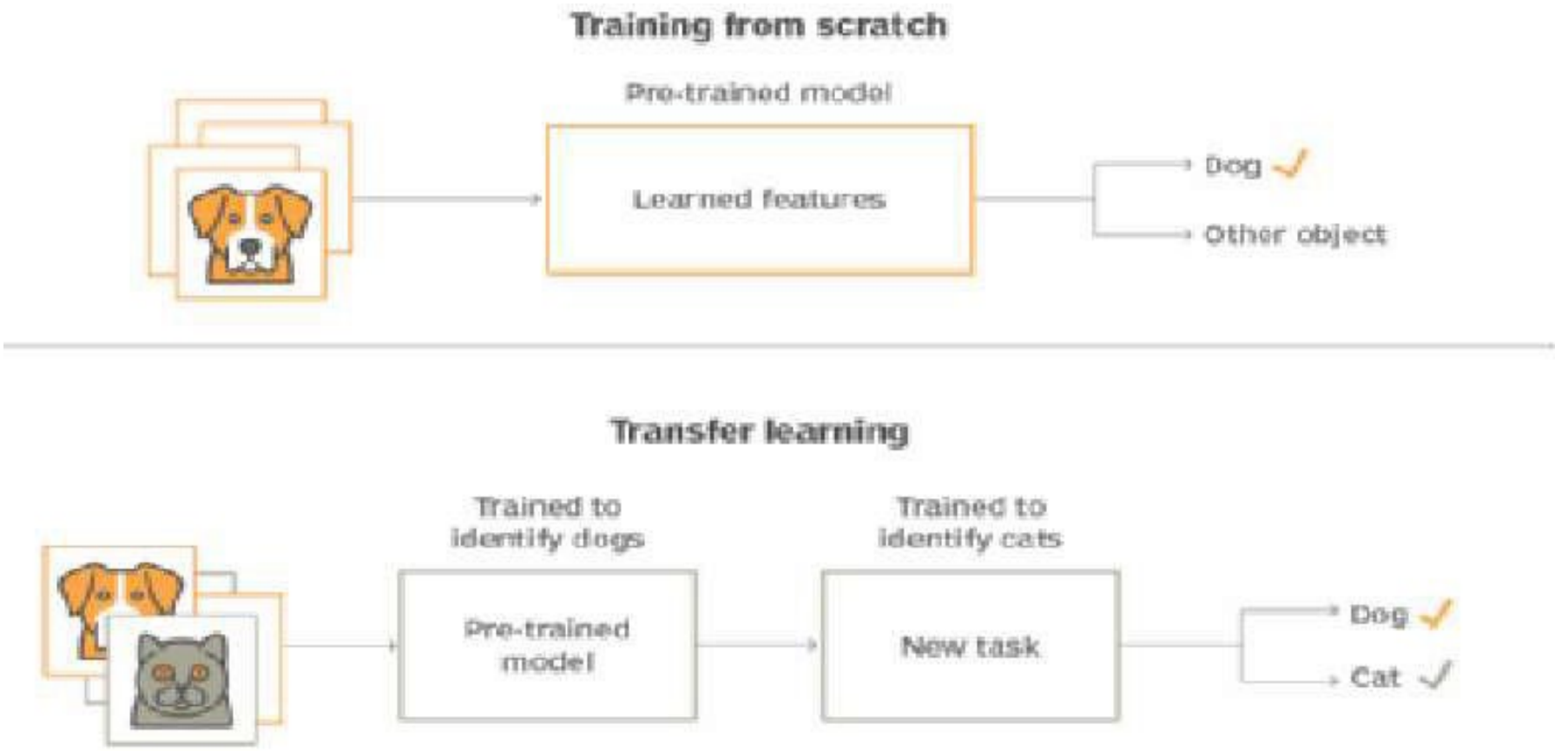


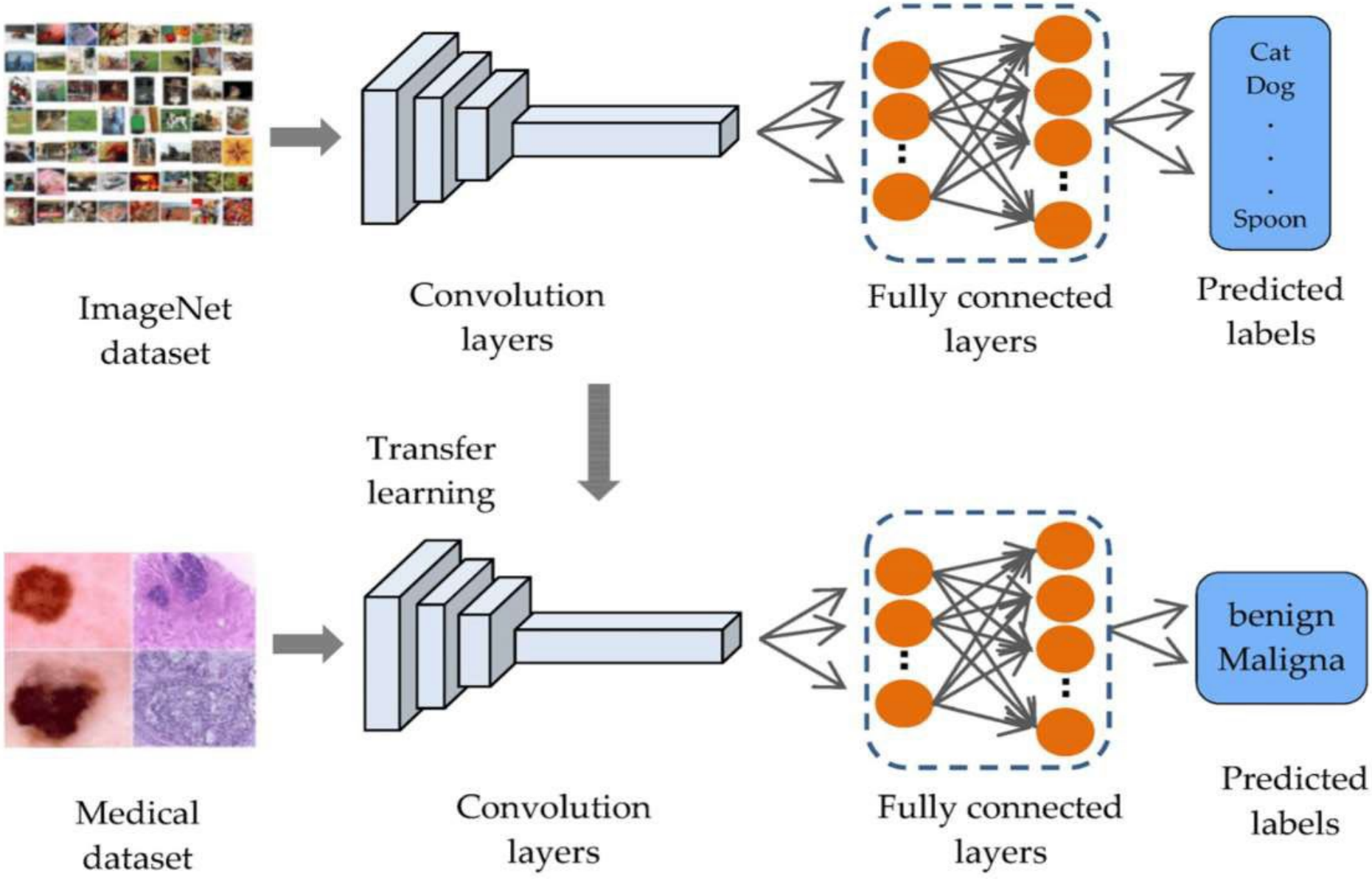
Cutout

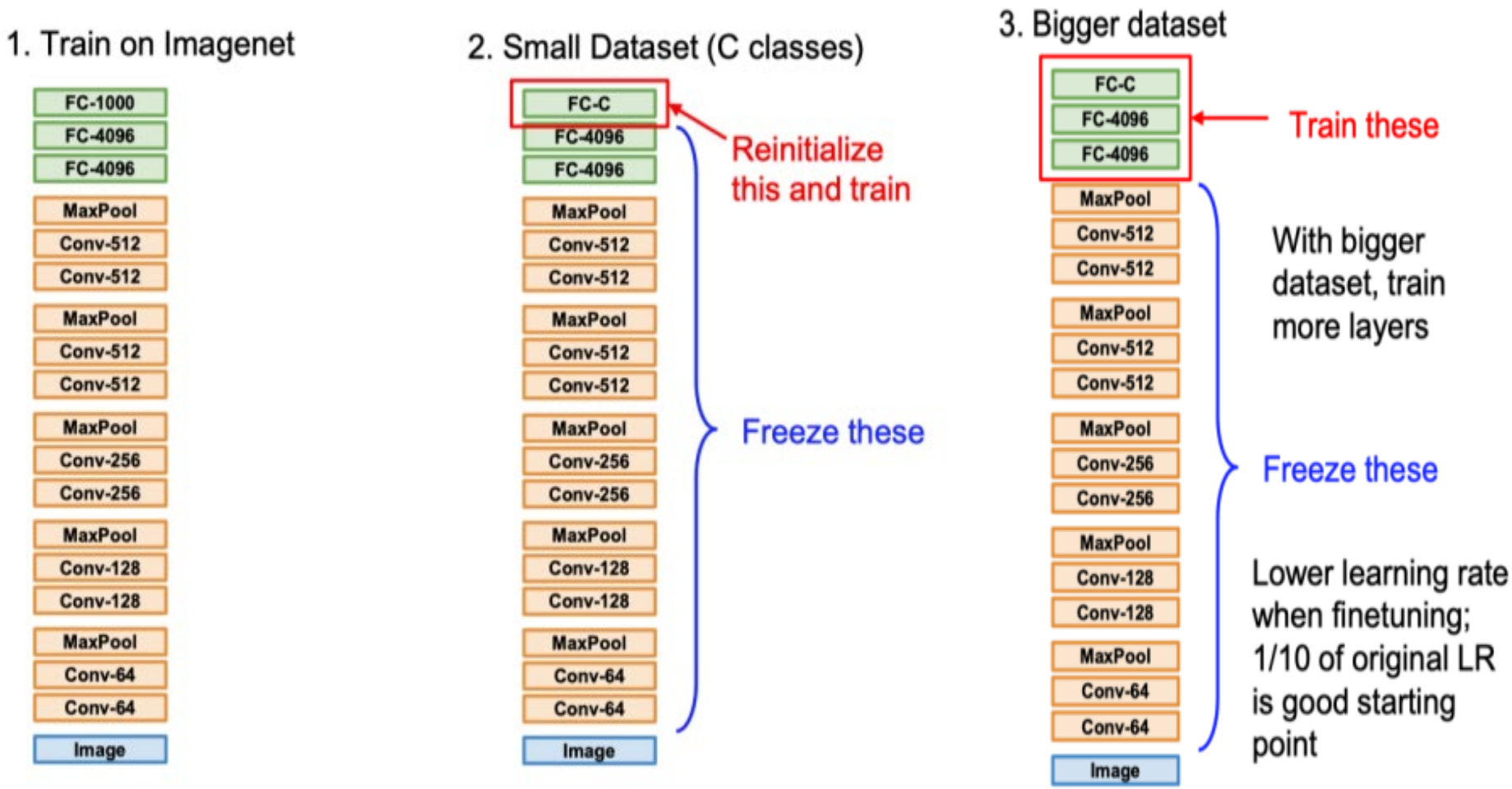
Transfer Learning

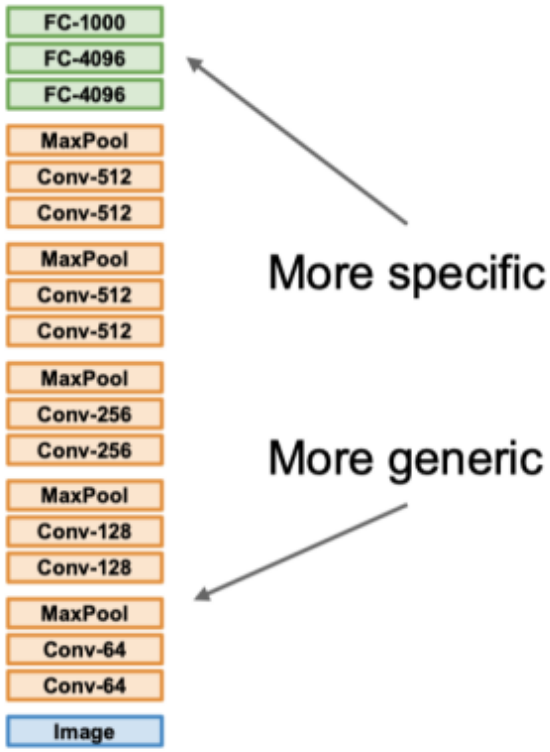
- What is Transfer Learning ?

How to transfer learning works




Cat
Dog
Spoon
ImageNet dataset
Convolution layers
Fully connected layers
Predicted labels
Transfer learning
benign
Maligna
Medical dataset
Convolution layers
Fully connected layers
Predicted labels


1. Train on Imagenet
FC-1000
FC-4096
FC-4096
MaxPool
Conv-512
Conv-512
MaxPool
Conv-512
Conv-512
MaxPool
Conv-256
Conv-256
MaxPool
Conv-128
Conv-128
MaxPool
Conv-64
Conv-64
Image
2. Small Dataset (C classes)
FC-C
FC-4096
FC-4096
MaxPool
Conv-512
Conv-512
MaxPool
Conv-512
Conv-512
MaxPool
Conv-256
Conv-256
MaxPool
Conv-128
Conv-128
MaxPool
Conv-64
Conv-64
Image
Reinitialize this and train
Freeze these
3. Bigger dataset
FC-C
FC-4096
FC-4096
Train these
MaxPool
Conv-512
Conv-512
MaxPool
Conv-512
Conv-512
MaxPool
Conv-256
Conv-256
MaxPool
Conv-128
Conv-128
MaxPool
Conv-64
Conv-64
Image
With bigger dataset, train more layers
Freeze these
Lower learning rate when finetuning; 1/10 of original LR is good starting point



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers or start from scratch!