ECBM E4040
Neural Networks and Deep Learning

Convolutional Neural Networks (CNN) Examples

Zoran Kostić
Columbia University
Electrical Engineering Department &
Data Sciences Institute

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

References and Acknowledgments

- **Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, http://www.deeplearningbook.org/ , chapter 6.**
- **Lecture material by bionet group / Prof. Aurel Lazar (http://www.bionet.ee.columbia.edu/).**
- **NVIDIA GPU Teaching Kit (NVIDIA and New York University)**
- **Karpathy, Fei-Fei Li & Justin Johnson & Serena Yeung, Stanford course CS231n: Convolutional Neural Networks for Visual Recognition**

Classification: ImageNet Challenge top-5 error

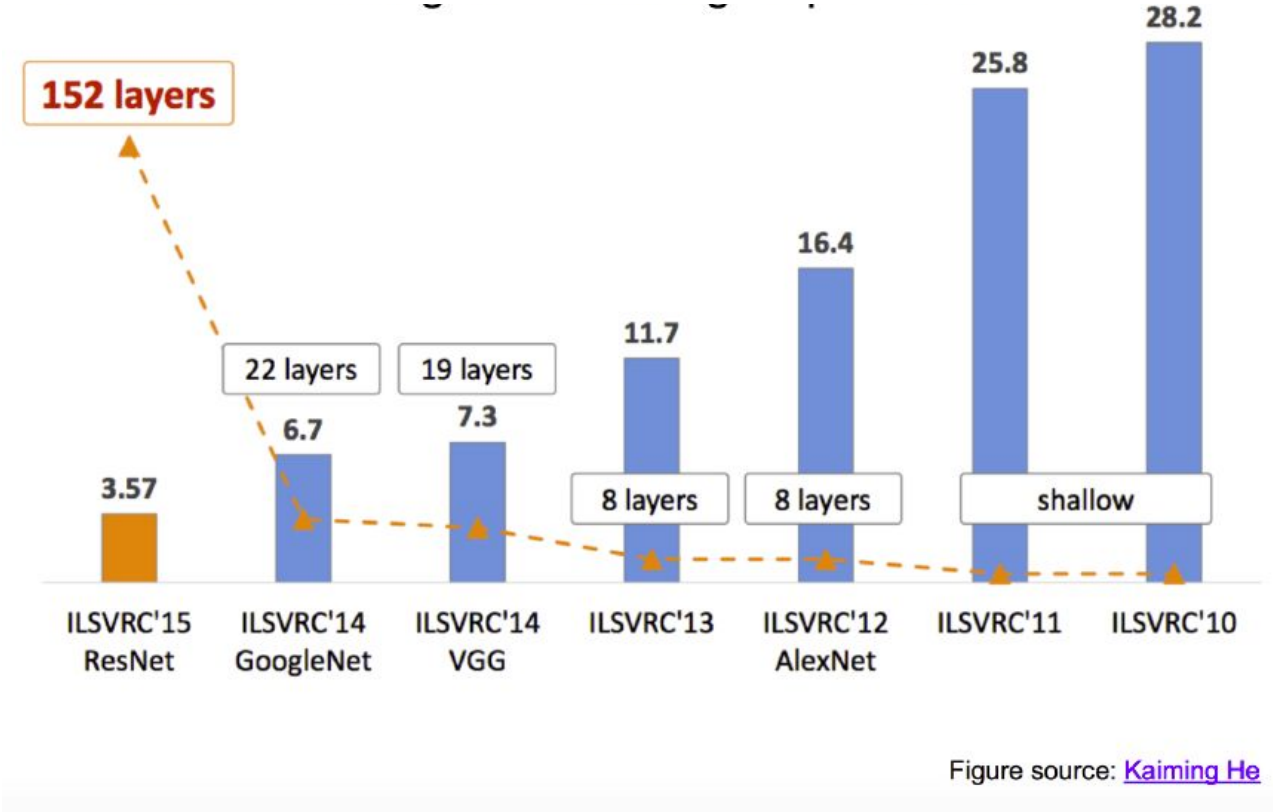


Figure source: Kaiming He

Convolutional Neural Networks

LeNet: read digits, zip codes. LeCun.

AlexNet: Computer vision; top 5 error of 16%; deeper, bigger, stacked convolutional layers. Hinton 2012.

ZF Net: Hyperparameter tweaking, smaller kernels and strides at input, large mid convolution layers. Zeiler & Fergus 2013

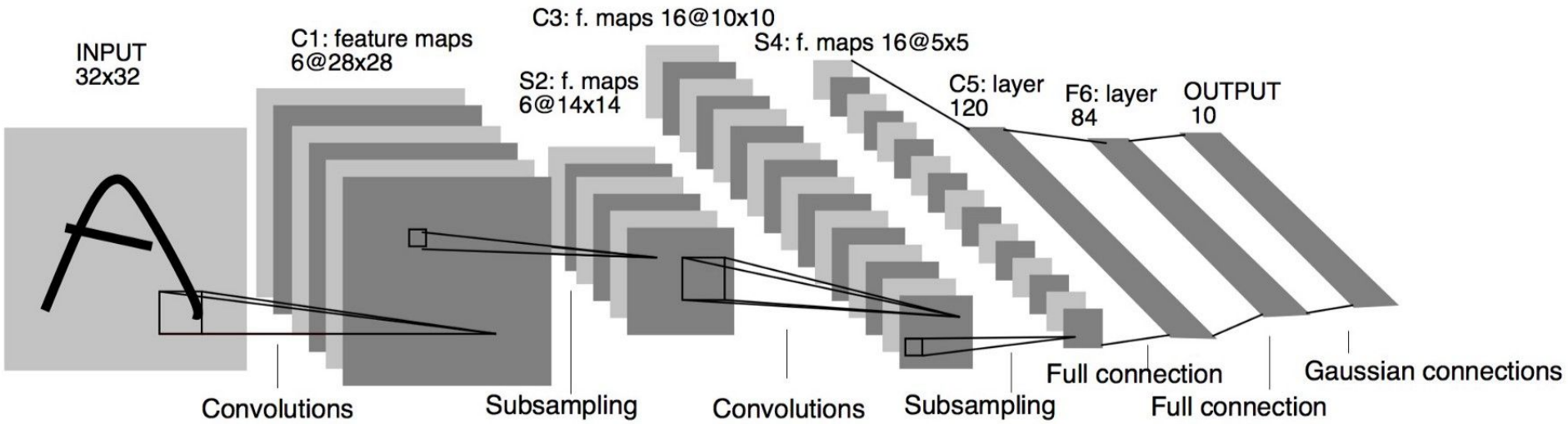
GoogLeNet, Inception, Xception: inception layers, average pooling towards output, Szegedy 2014.

VGGNet: Deep is good, simple architecture with repetitive 3x3 convolutions and 2x2 pooling; lots of memory, Simoyan 2014.

ResNet: skip connections, batch normalization, no FC at end, Kaiming 2015.

https://github.com/tensorflow/models

Convolutional Network- LeNet5



http://yann.lecun.com/exdb/lenet/ lenet 5 - see lenet in action: demonstrations in robustness

Convolutional Networks

ANNs: From second failure in 1990s

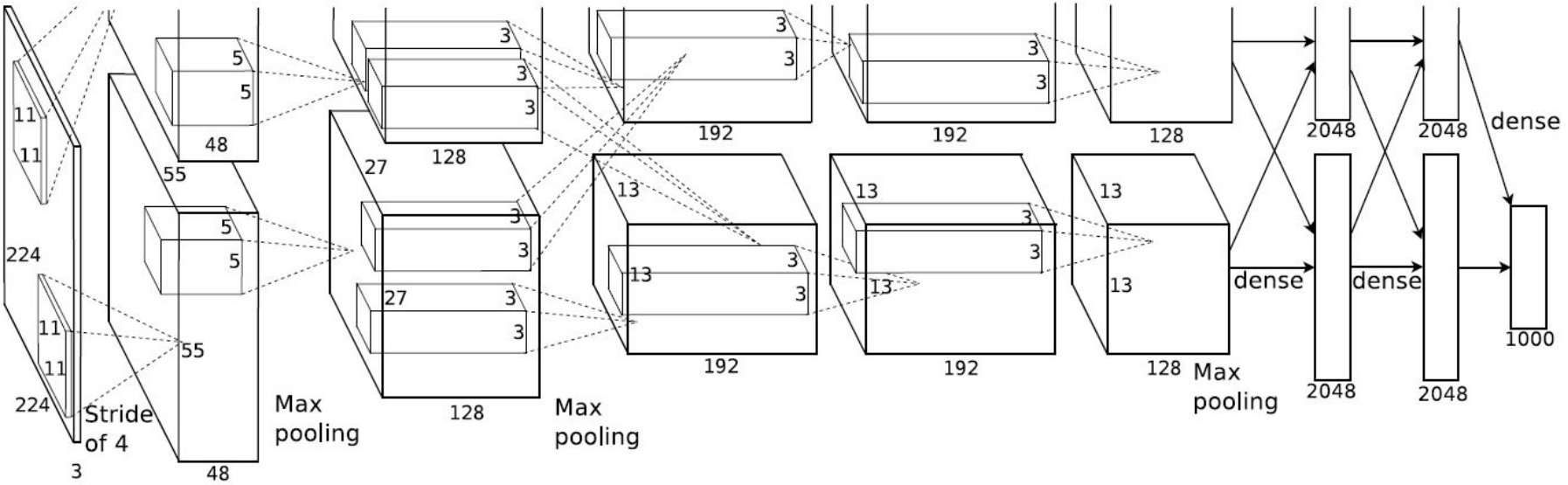
to

Return of the JEDI

…. in form of AlexNet

…. in 2012

Convolutional Network - AlexNet

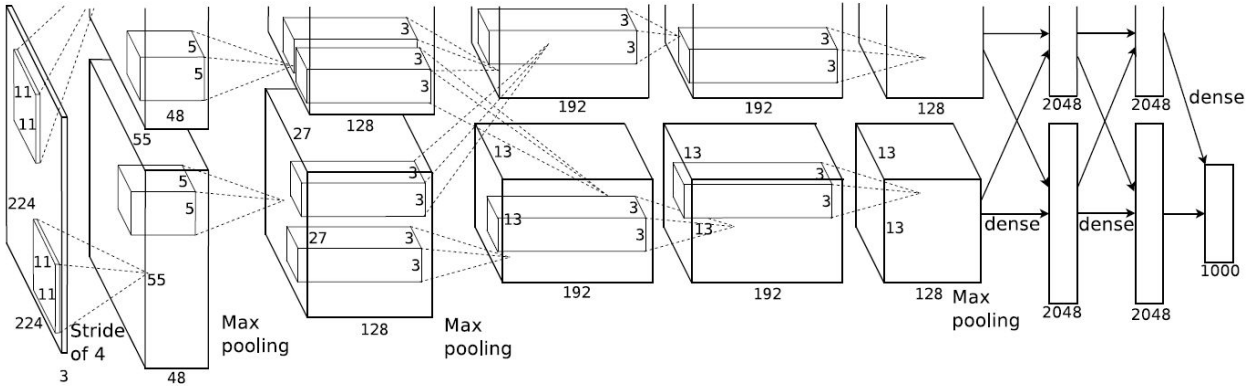


alexnet https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Convolutional Network - AlexNet

Architecture:

- CONV1
- MAX POOL1
- NORM1
- CONV2
- MAX POOL2
- NORM2
- CONV3
- CONV4
- CONV5
- Max POOL3
- FC6
- FC7
- FC8



\+ **ReLUs everywhere**

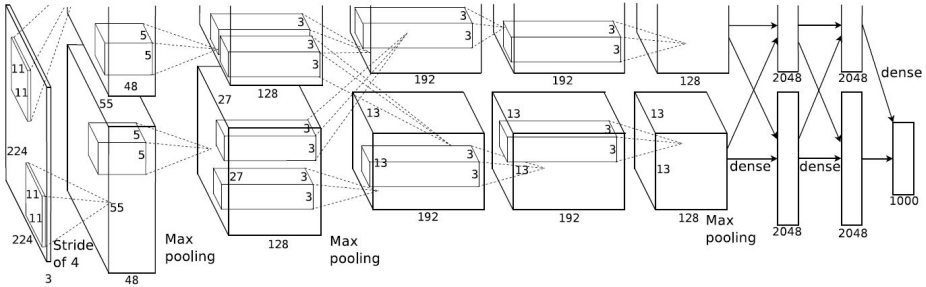
Some peculiar non-features of the diagram:

Input image size not correct

Two horizontal threads:

- **For task distribution across GPUs**

Convolutional Network - AlexNet



Architecture:

- CONV1
- MAX POOL1
- NORM1
- CONV2
- MAX POOL2
- NORM2
- CONV3
- CONV4
- CONV5
- Max POOL3
- FC6
- FC7
- FC8

Input Image Size: 227x227x3

First layer (CONV1): 96 11x11 filters with stride 4

Output volume size calculation: (227-11)/4+1 = 55

- **Output volume size: 55x55x96**

Total number of parameters in CONV1:

- **Parameters: (11*11*3)*96 = 35K**

Second layer (POOL1): 3x3 filters with stride 2

Output volume size calculation: (55-3)/2+1 = 27

- **Output volume size: 27x27x96**

Total number of parameters in POOL1:

- **Parameters: 0**

Convolutional Network - AlexNet

Architecture:

- CONV1
- MAX POOL1
- NORM1
- CONV2
- MAX POOL2
- NORM2
- CONV3
- CONV4
- CONV5
- Max POOL3
- FC6
- FC7
- FC8

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

Convolutional Network - AlexNet

Architecture:

- CONV1
- MAX POOL1
- NORM1
- CONV2
- MAX POOL2
- NORM2
- CONV3
- CONV4
- CONV5
- Max POOL3
- FC6
- FC7
- FC8

Key Architectural Properties of AlexNet:

- Introduction of ReLU
- Application of normalization
- Dropout

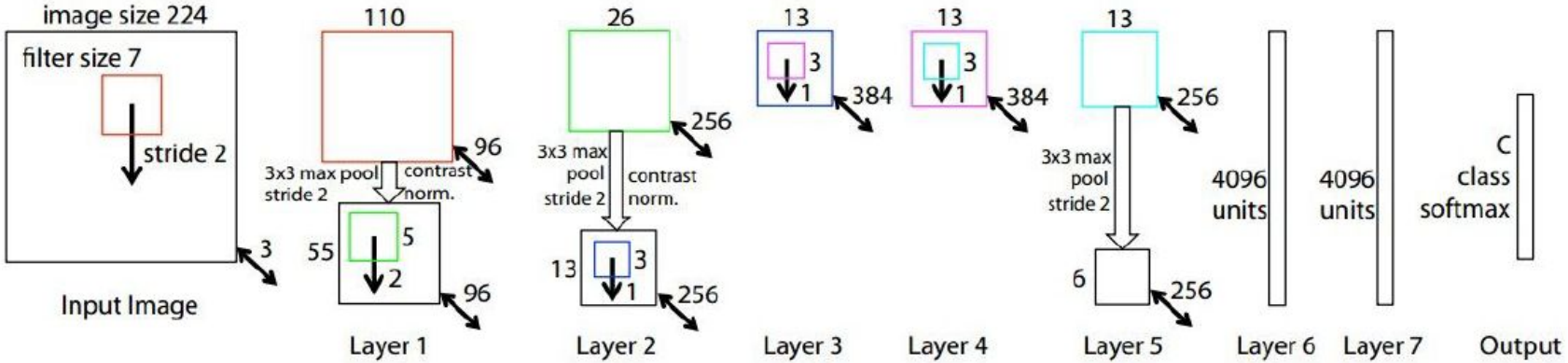
Data Conditioning:

- data augmentation

Parameters:

- dropout p = 0.5
- batch size = 128
- SGD Momentum rate = 0.9
- Learning rate = 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay = 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Convolutional Network - ZFNet



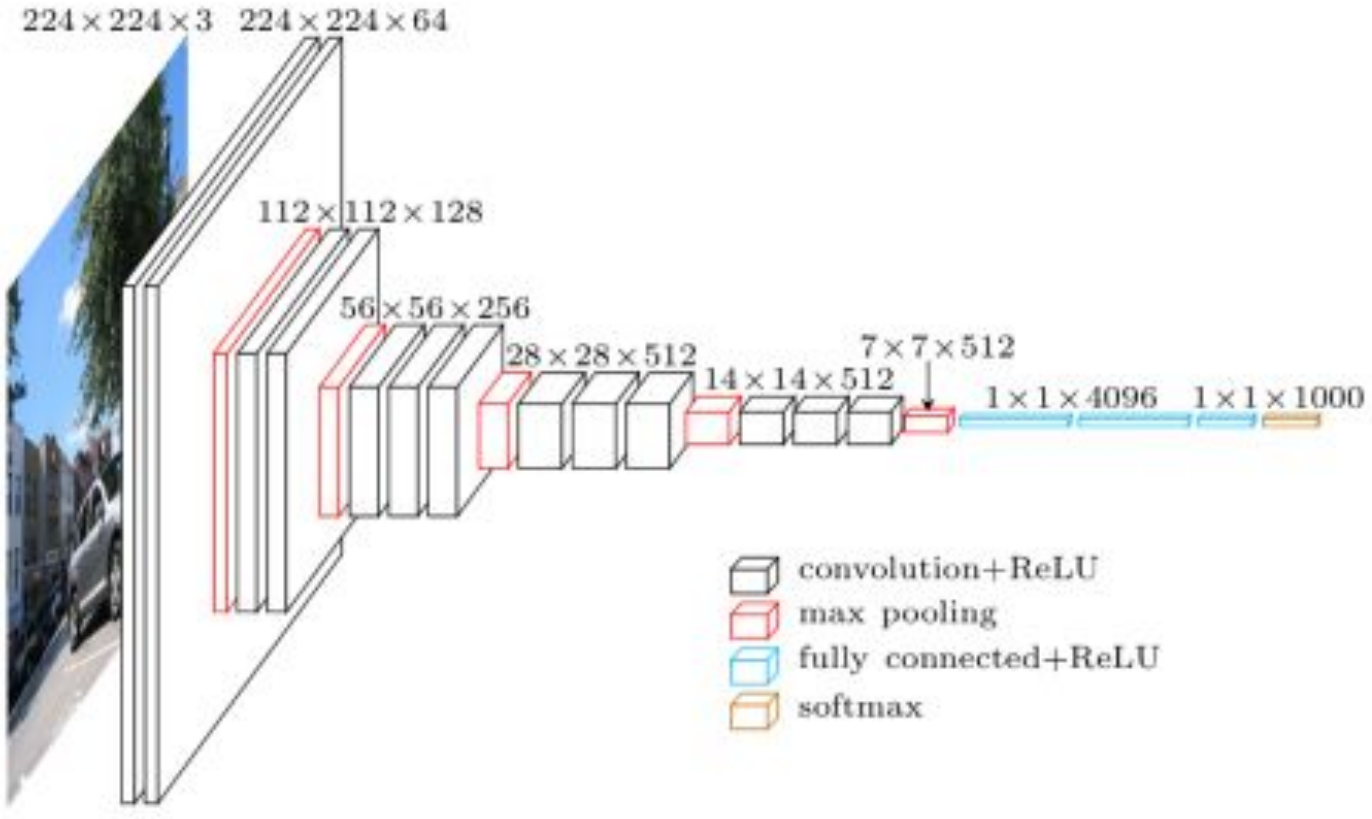
Modified AlexNet:

- **CONV1: change from (11x11 stride 4) to (7x7 stride 2)**
- **CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512**

ImageNet Top 5 error: 16.4%->11.7%

[Zeiler and Fergus, 2013]

Convolutional Network: VGG16 (and VGG19)



Convolutional Network: VGG16 (and VGG19)

The VGG network architecture was introduced by Simonyan and Zisserman in 2014: Very Deep Convolutional Networks for Large Scale Image Recognition.

The network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth.

Reducing volume size is handled by max pooling.

Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier.

The "16" and "19" stand for the number of weight layers in the network

Tensorflow model-> https://www.cs.toronto.edu/~frossard/post/vgg16/

VGG16 (and VGG19)

In 2014: 16 and 19 layer networks

were considered very deep.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 1: Very Deep Convolutional Networks for Large Scale Image Recognition, Simonyan and Zisserman (2014).

VGG16 and VGG19

Training VGG16 and VGG19 is challenging - specifically regarding convergence on the deeper networks.

To make training easier, the authors trained smaller versions of VGG with less weight layers (columns A and C) first.

The smaller networks converged and were then used as initializations for the larger, deeper networks — this process is called pre-training.

While making sense, pre-training is a very time consuming, tedious task, requiring an entire network to be trained before it can serve as an initialization for a deeper network.

VGG16 and VGG19

There are two major drawbacks with VGGNet:

- **It is painfully slow to train.**
- **The network architecture weights themselves are quite large (in terms of disk/bandwidth).**

Due to its depth and number of fully-connected nodes, VGG is over 533MB for VGG16 and 574MB for VGG19. This makes deploying VGG a tiresome task.

VGG is still used in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogLeNet, etc.).

VGG16 and VGG19

More and more often, pre-training is not used.

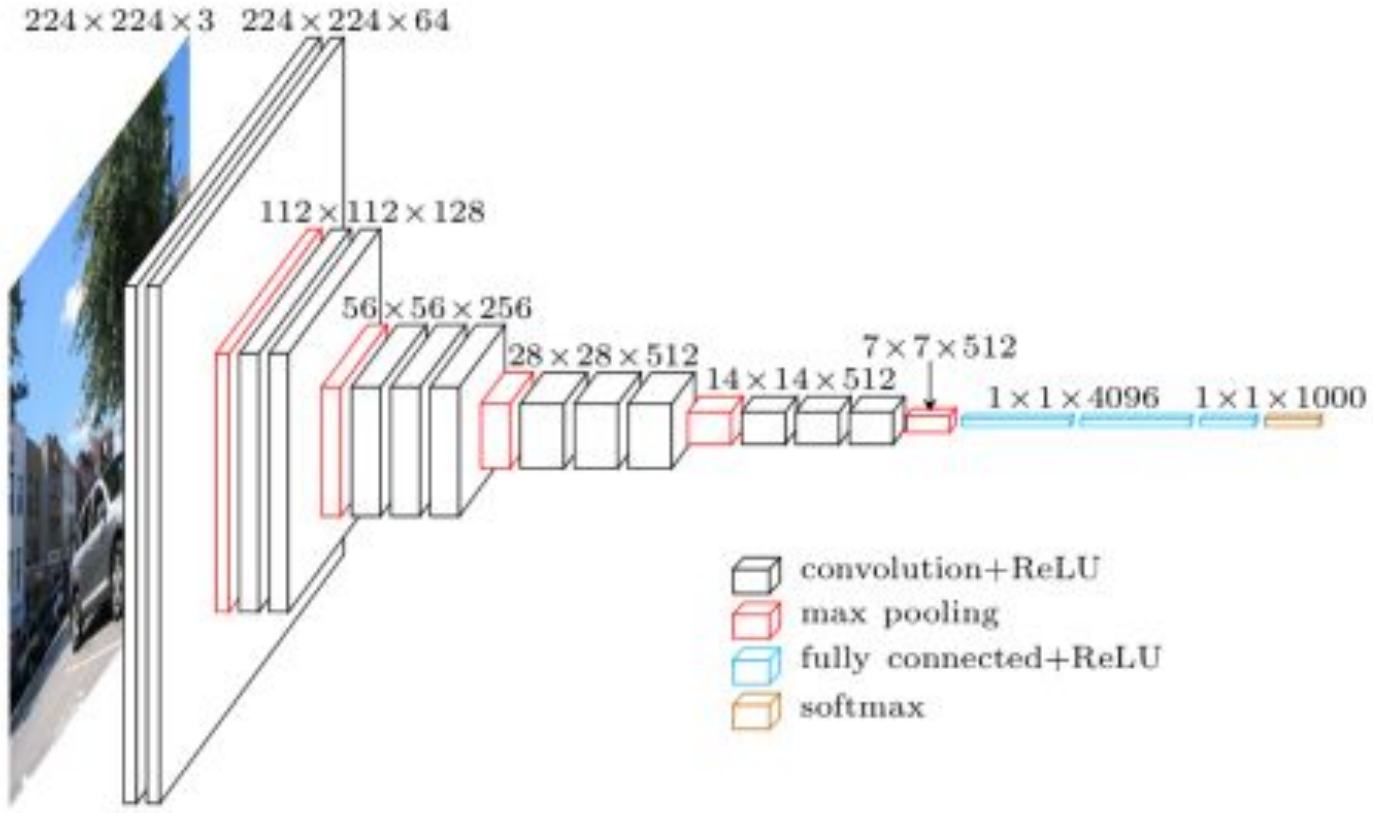
Instead, one can use Xaiver/Glorot initialization or MSRA initialization (sometimes called He et al. initialization from the paper, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification).

Initialization summary: All you need is a good init, Mishkin and Matas (2015).

VGG16 used for Pre-Training: Domain Transfer + Fine-Tuning

Start with ->

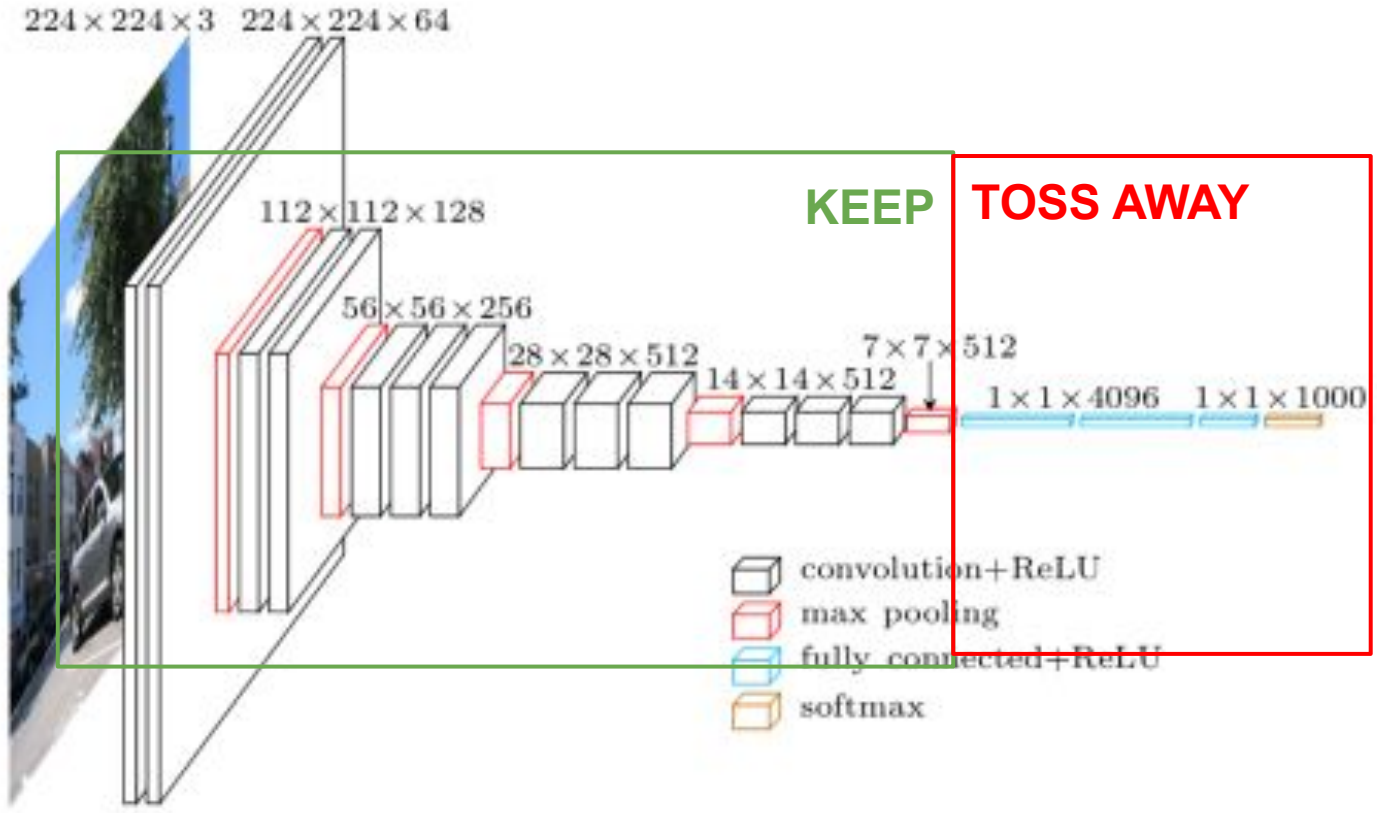
TRAIN



For a New Problem: Keep Most, Toss Away Some

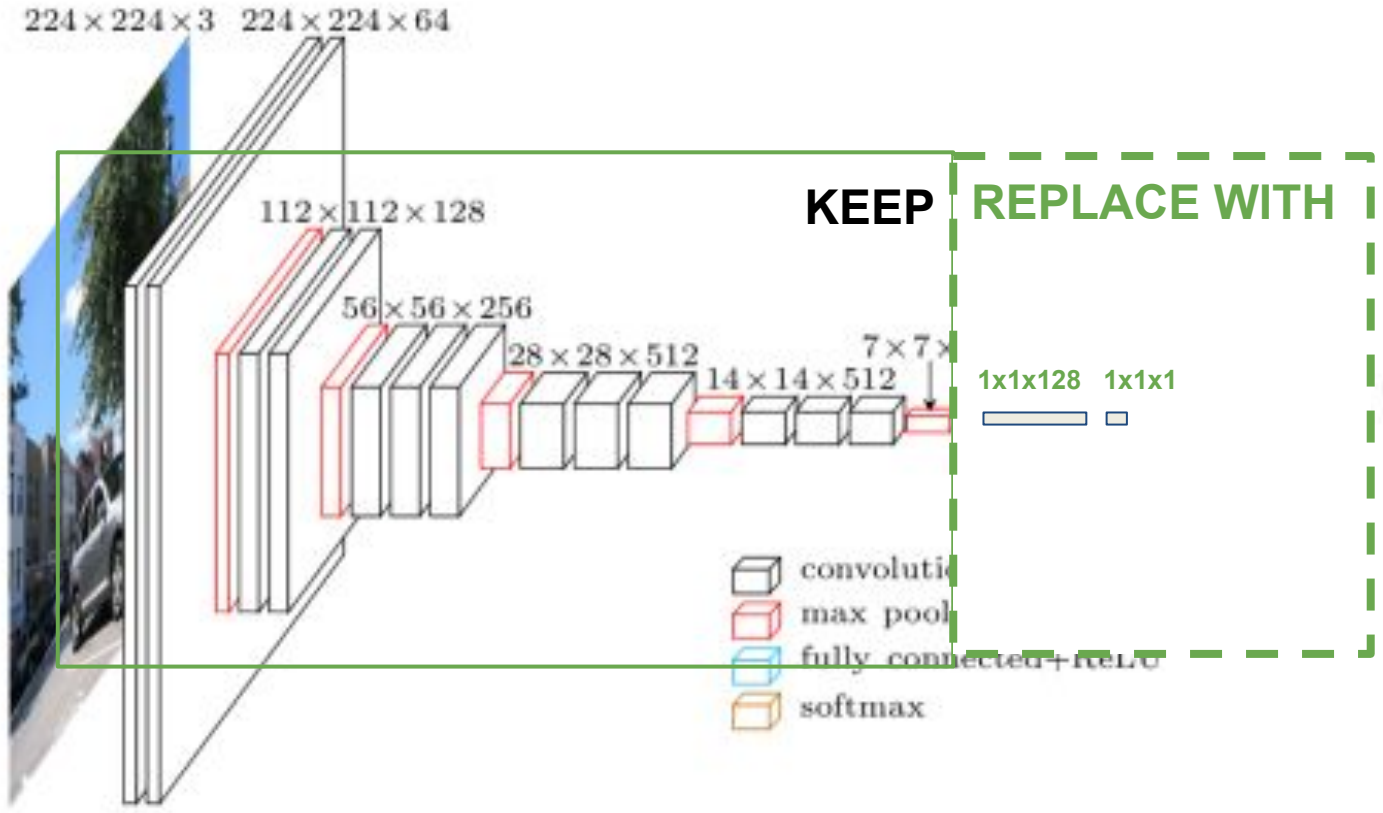
Domain Transfer + Fine-Tuning

TOSS AWAY



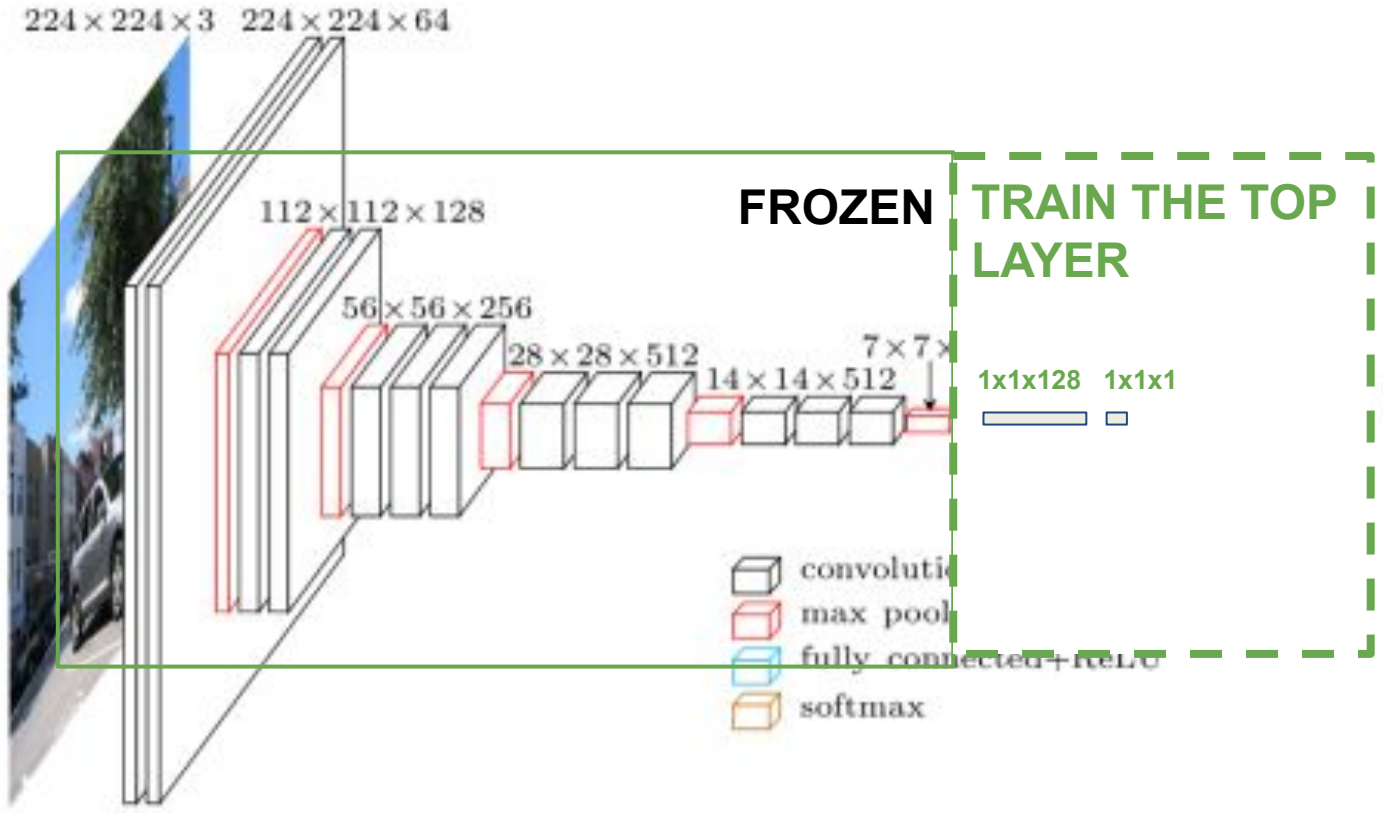
For a New Problem: Keep Most, Replace With Domain Transfer + Fine-Tuning

REPLACE THE TOP LAYER



For a New Problem: Keep Most, Train Top
Domain Transfer + Fine-Tuning

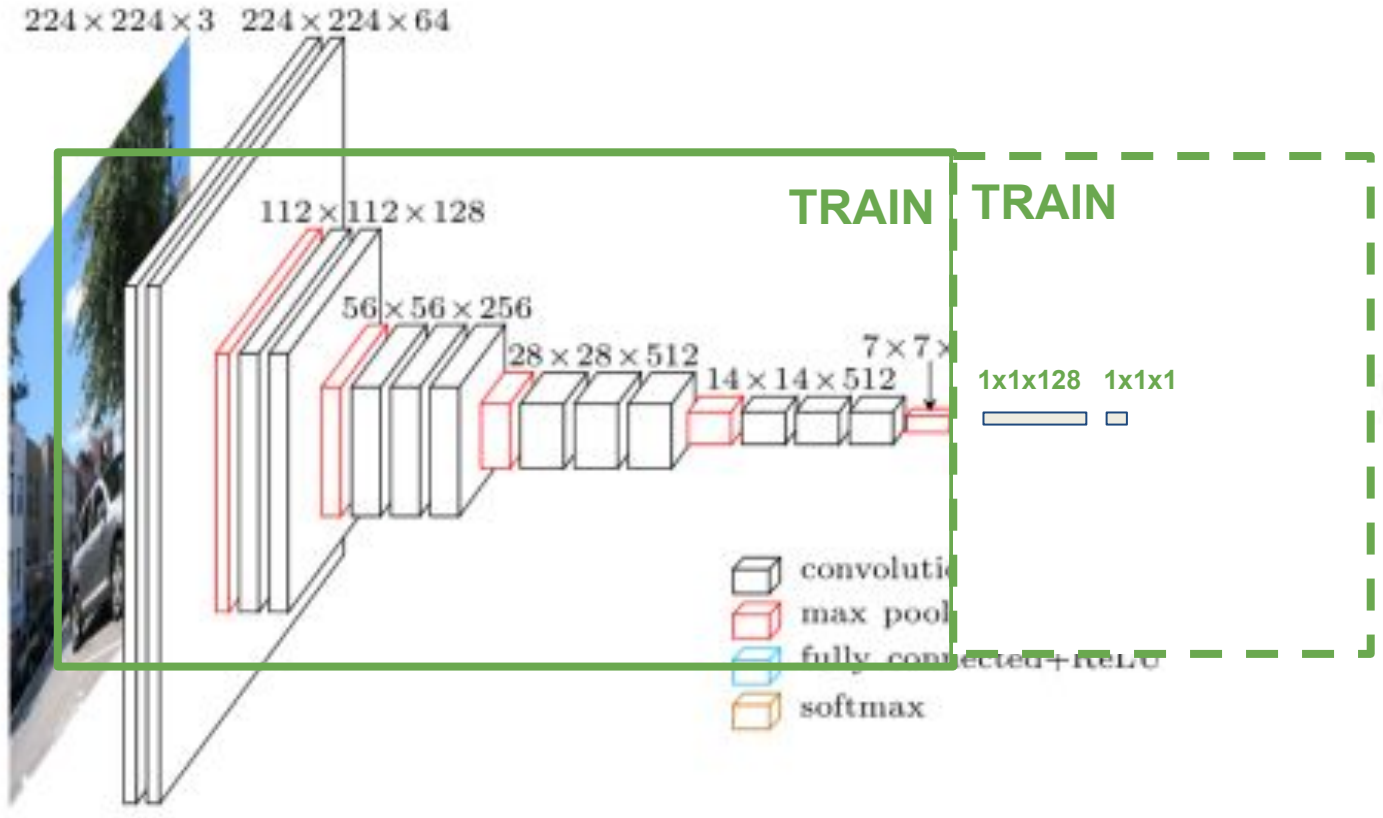
TRAIN THE TOP LAYER



For a New Problem: Keep Most, Train All

Domain Transfer + Fine-Tuning

TRAIN ALL



ResNet

ResNet is an "exotic architecture" which uses micro-architecture modules ("network-in-network architectures").

The term micro-architecture refers to the set of "building blocks" used to construct the network.

A collection of micro-architecture building blocks (along with standard CONV, POOL, etc. layers) leads to the macro-architecture (i.e,. the end network itself).

First introduced by He et al. in 2015 paper, Deep Residual Learning for Image Recognition, the ResNet architecture has become a seminal work, demonstrating that extremely deep networks can be trained using standard SGD (and a reasonable initialization function) through the use of residual modules.

https://arxiv.org/abs/1512.03385

ResNet

He et al. in 2015 paper, Deep Residual Learning for Image Recognition

"We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth."

"...We also present analysis on CIFAR-10 with 100 and 1000 layers"

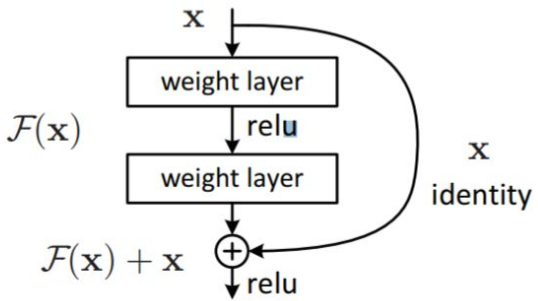


Figure 2. Residual learning: a building block.

ResNet

Formally, denoting the desired underlying mapping as H(x), we let the stacked nonlinear layers fit another mapping of F(x) := H(x)−x. The original mapping is recast into F(x)+x.

We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.

To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

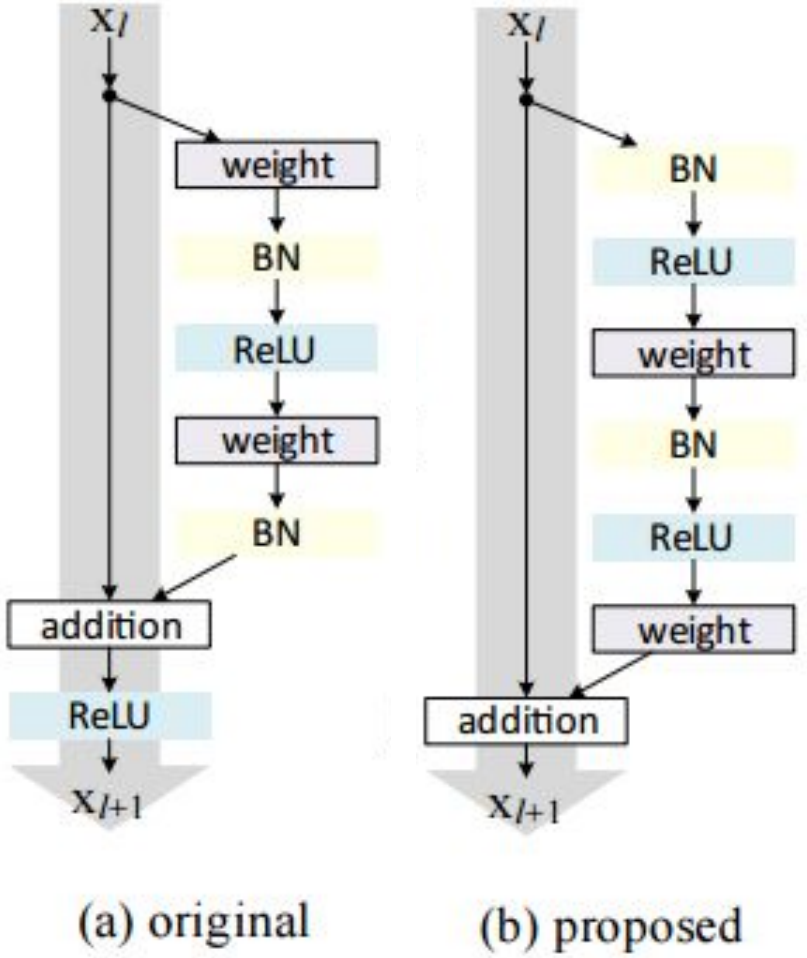
The formulation of *F(x) +x* can be realized by feedforward neural networks with "shortcut connections" (Fig. 2). Shortcut connections are those skipping one or more layers. In our case, the shortcut connections simply perform identity mapping

ResNet

The residual module in ResNet as originally proposed by He et al. in 2015.

The updated residual module using pre-activation.

Microsoft 2016: Identity Mappings in Deep Residual Networks
https://arxiv.org/abs/1603.05027



(a) original (b) proposed

ResNet Network Architecture

See Figure 3 of

https://arxiv.org/pdf/1512.03385.pdf

Left: VGG, Center:Plain, Right:Residual

Plain -> residual (by inserting shortcut connections)

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). T

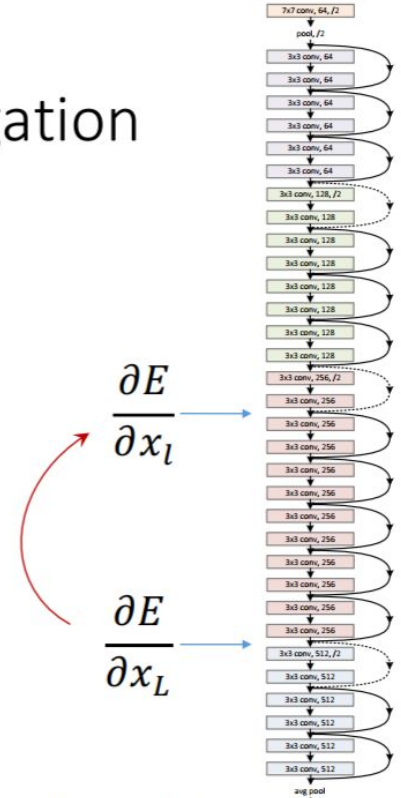
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase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

Backpropagation details

Very smooth backward propagation

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i)\right)$$

- Any $\frac{\partial E}{\partial x_L}$ is directly back-prop to any $\frac{\partial E}{\partial x_l}$, plus residual.
- Any $\frac{\partial E}{\partial x_l}$ is additive; unlikely to vanish
  - in contrast to multiplicative: $\frac{\partial E}{\partial x_l} = \prod_{i=l}^{L-1} W_i \frac{\partial E}{\partial x_L}$



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Identity Mappings in Deep Residual Networks". arXiv 2016.

Backpropagation details

Residual for every layer

forward: $x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$

Enabled by:

- shortcut mapping: h = identity
- after-add mapping: f = identity

backward: $\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i))$

ResNet

Even though ResNet is much deeper than VGG16 and VGG19,

- **the model size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers**
- **this reduces the model size down to 102MB for ResNet50.**

GoogLeNet -> Inception Vx (V3)

The "Inception" micro-architecture was introduced by Szegedy et al. in 2014 paper, Going Deeper with Convolutions: https://arxiv.org/abs/1409.4842

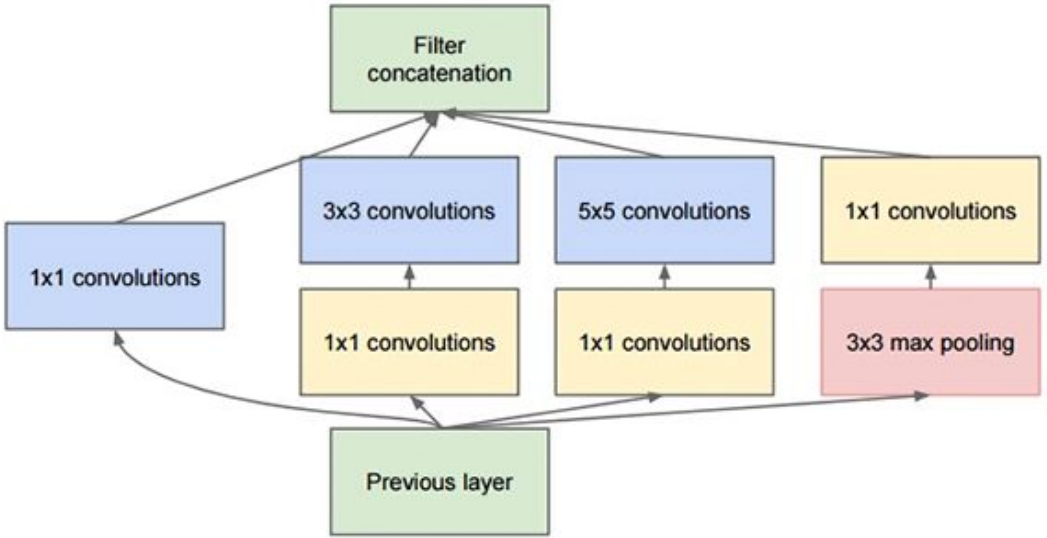
The goal of the inception module is to act as a "multi-level feature extractor" by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network

- the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network.

Szegedy et al., Rethinking the Inception Architecture for Computer Vision (2015) https://arxiv.org/abs/1512.00567

GoogLeNet -> Inception Vx (V3)

The "Inception" micro-architecture was introduced by Szegedy et al. in 2014 paper, Going Deeper with Convolutions: https://arxiv.org/abs/1409.4842



GoogLeNet -> Inception Vx (V3)

GoogleNet employed only 5 million parameters, which is a 12× reduction with respect to its predecessor AlexNet, which used 60 million parameters. Furthermore, VGGNet employed about 3x more parameters than AlexNet.

The computational cost of Inception is also much lower than VGGNet or its higher performing successors -> utilize Inception networks in big-data scenarios.

Much of the original gains of the GoogLeNet network arise from a very generous use of dimension reduction. This can be viewed as a special case of factorizing convolutions in a computationally efficient manner.

https://arxiv.org/pdf/1512.00567.pdf

https://github.com/tensorflow/models/blob/master/research/inception/inception/inception_model.py

GoogLeNet -> Inception Vx (V3)

A 5 × 5 convolution with n filters over a grid with m filters is 25/9 = 2.78 times more computationally expensive than a 3 × 3 convolution with the same number of filters.

Ask whether a 5 × 5 convolution could be replaced by a multi-layer network with less parameters with the same input size and output depth.

Sliding this small network over the input activation grid boils down to replacing the 5 × 5 convolution with two layers of 3 × 3 convolution.

https://arxiv.org/pdf/1512.00567.pdf

GoogLeNet -> Inception Vx (V3)

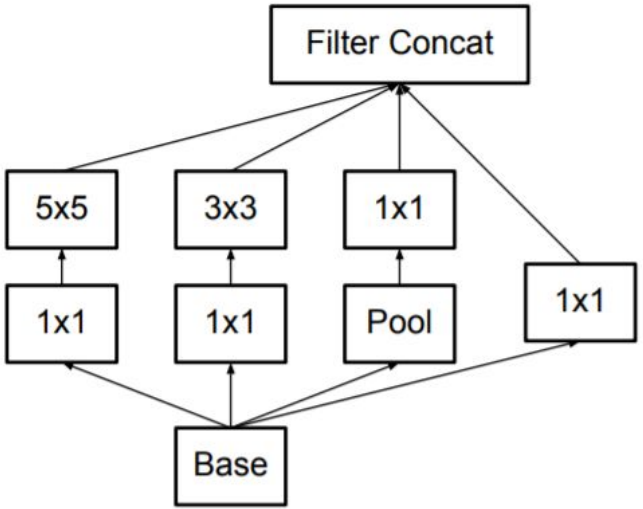


Figure 4. Original Inception module as described in [20].

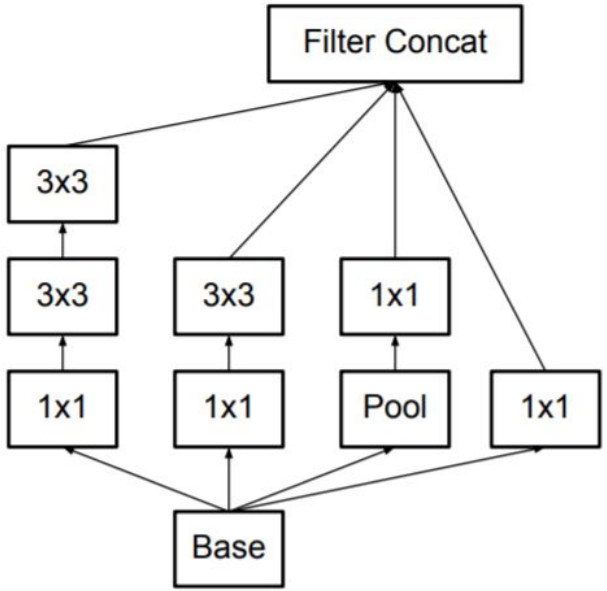
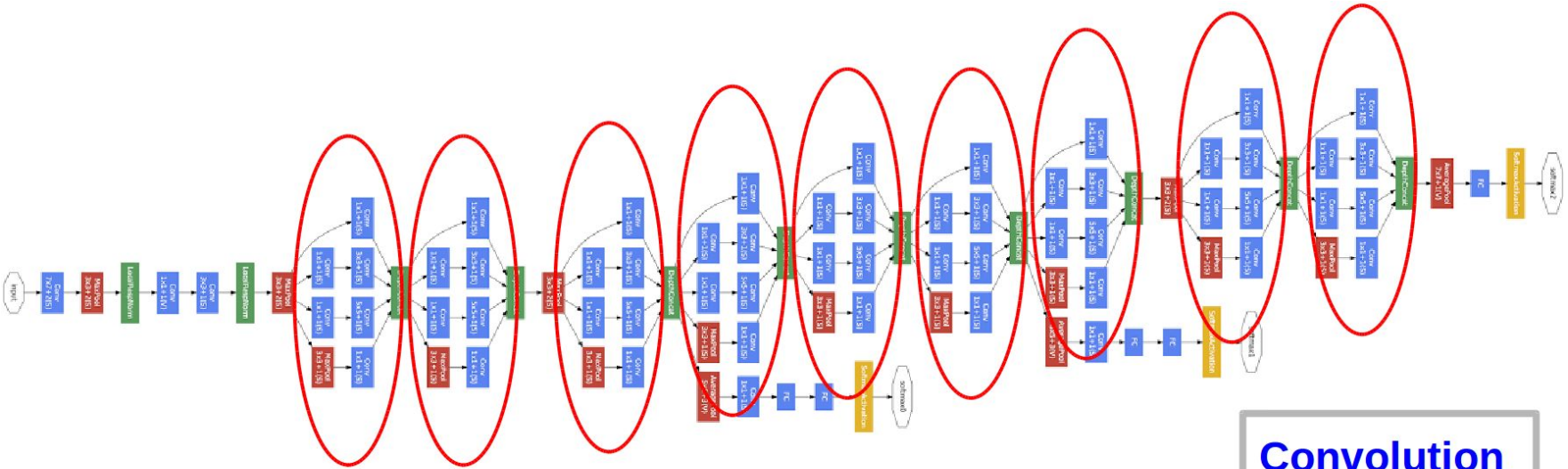


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

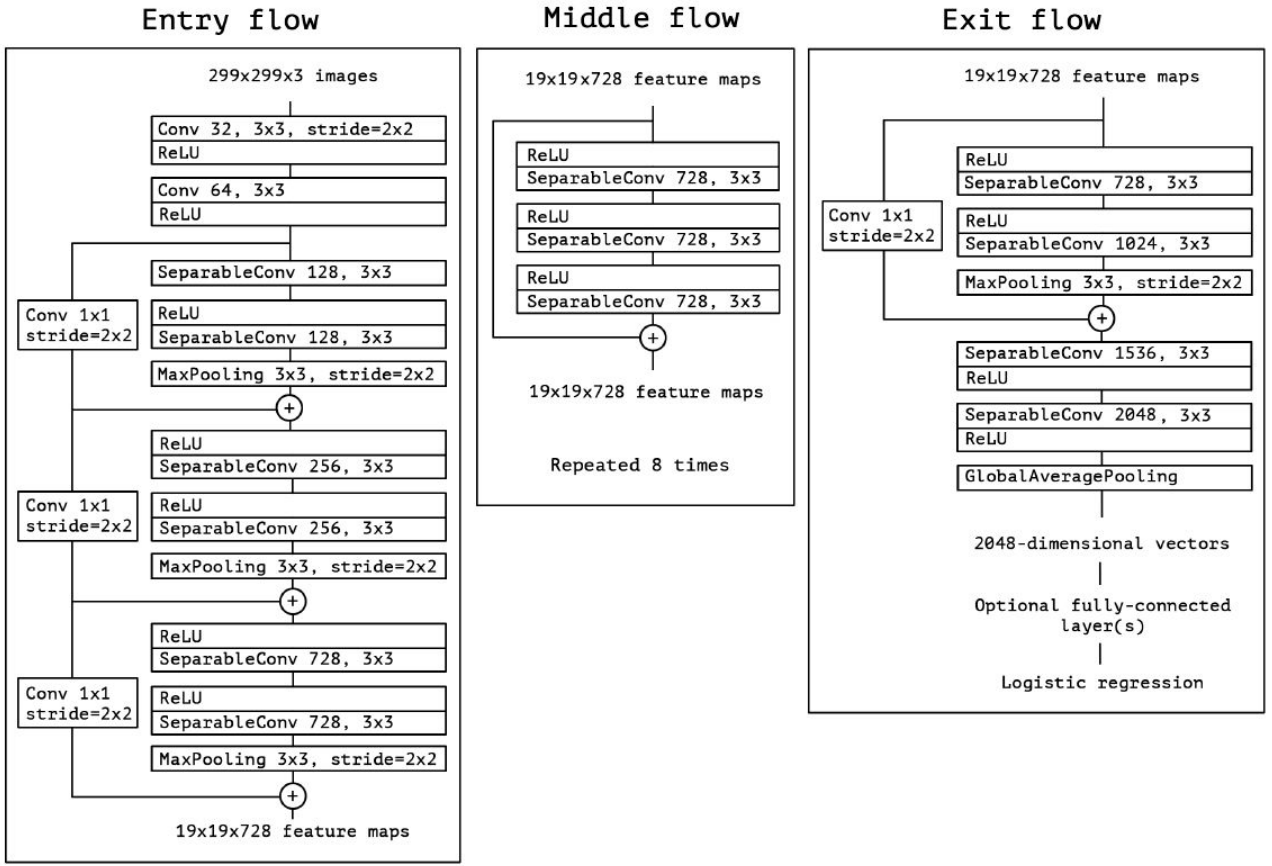
https://arxiv.org/pdf/1512.00567.pdf

GoogLeNet -> Inception Vx (V3)



Exception

Exception architecture replaces the standard Inception modules with depthwise separable convolutions



Xception: Deep Learning with Depthwise Separable Convolutions
https://arxiv.org/abs/1610.02357

Classification: ImageNet Challenge top-5 error

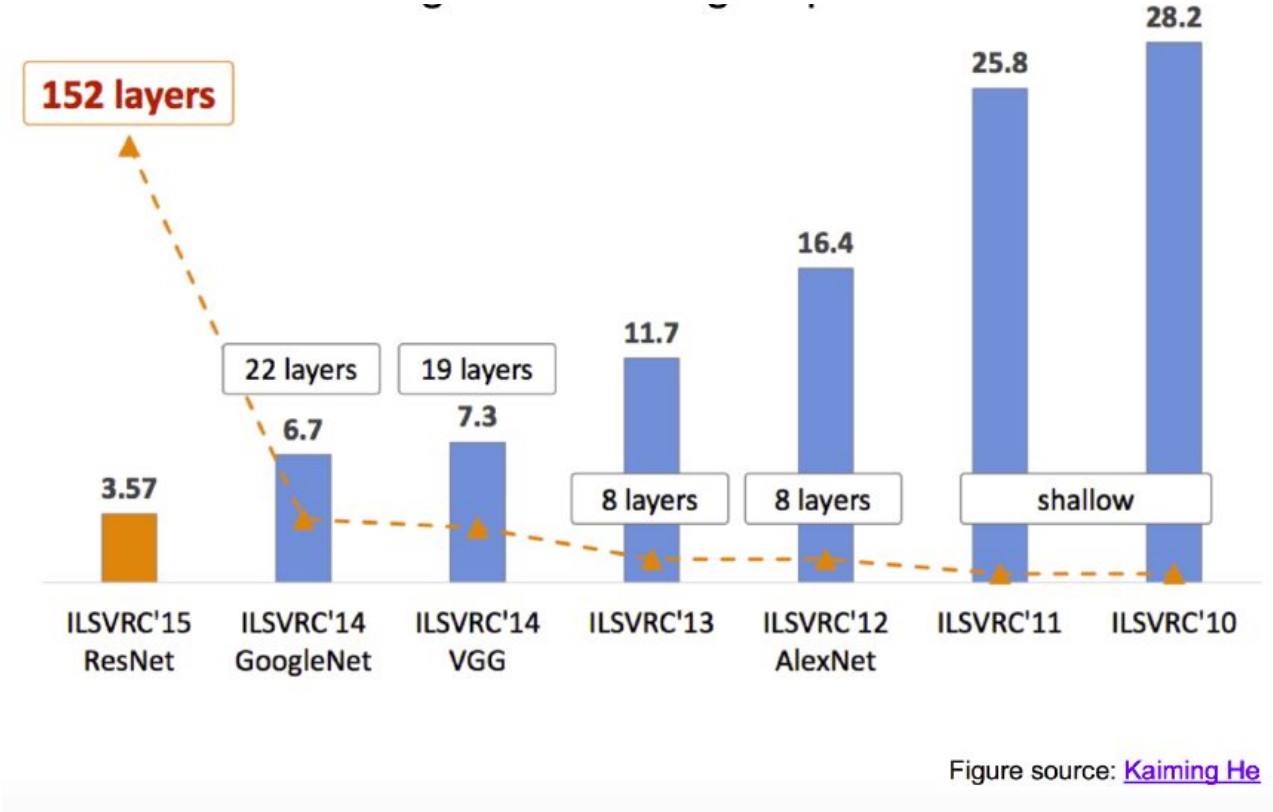


Figure source: Kaiming He

Backup Slides

Various