DS-UA 301
Advanced Topics in Data Science
Advanced Techniques in ML and Deep Learning

LECTURE 6

Parijat Dube

Popular CNNs in Computer Vision

LeNet (1998)

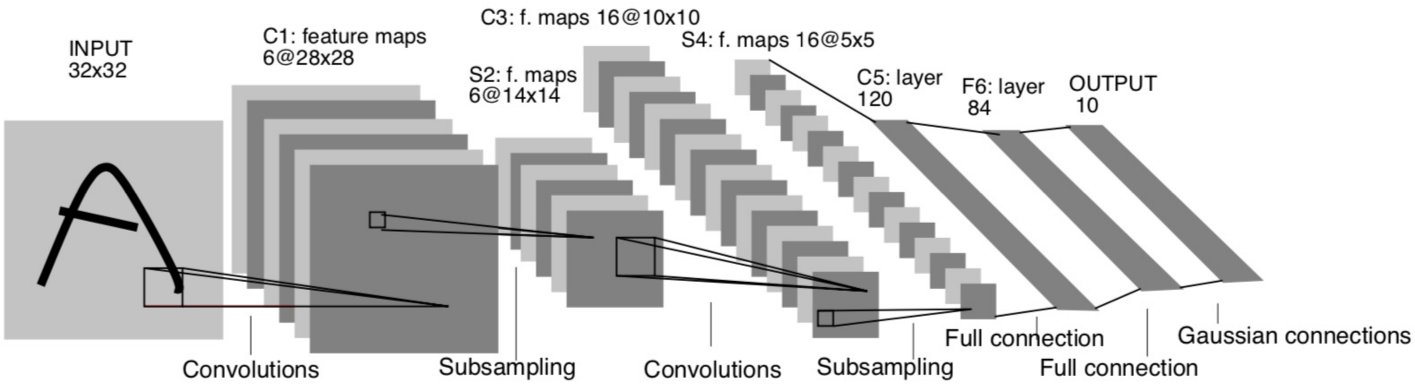


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- First convolutional layer (C1): 6 filters, 5x5, stride 1; # parameters = 156
- First pooling layer: 6 filters, 2x2, stride 2; # parameters = 12
- Second convolutional layer (C3): 16 filters, 5x5, stride 1; # parameters = 1516 (not a regular convolution layer, each unit in a feature map connected to a subset of input feature maps at identical locations)
- Second pooling layer: 16 filters, 2x2 stride, 2; # parameters = 32
- Third convolutional layer: 120 filters, 5x5, stride 1; # parameters = 25x16x120+120 = 48, 120. It is a fully connected layer
- Fully connected layer: #parameters = 84x120 +84 = 10,164

LeNet Questions

- How did we get number of parameters = 156 on first convolutional layer?
- How many connections in first convolutional layer ?
- If the first layer was fully connected instead what would be the number of parameters and connections? What about the number of parameters to learn?
- Why number of trainable parameters is 12 for first pooling layer ?
- Why third convolutional layer is technically a fully connected layer ?
- What type of layers are contributing the maximum number of trainable parameters ?

Alexnet (2012)

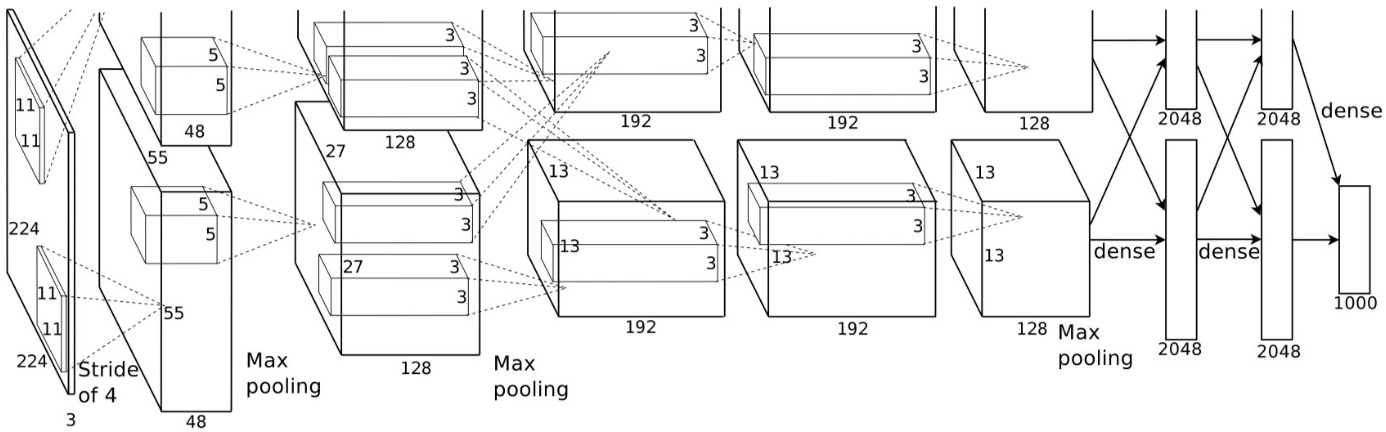


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

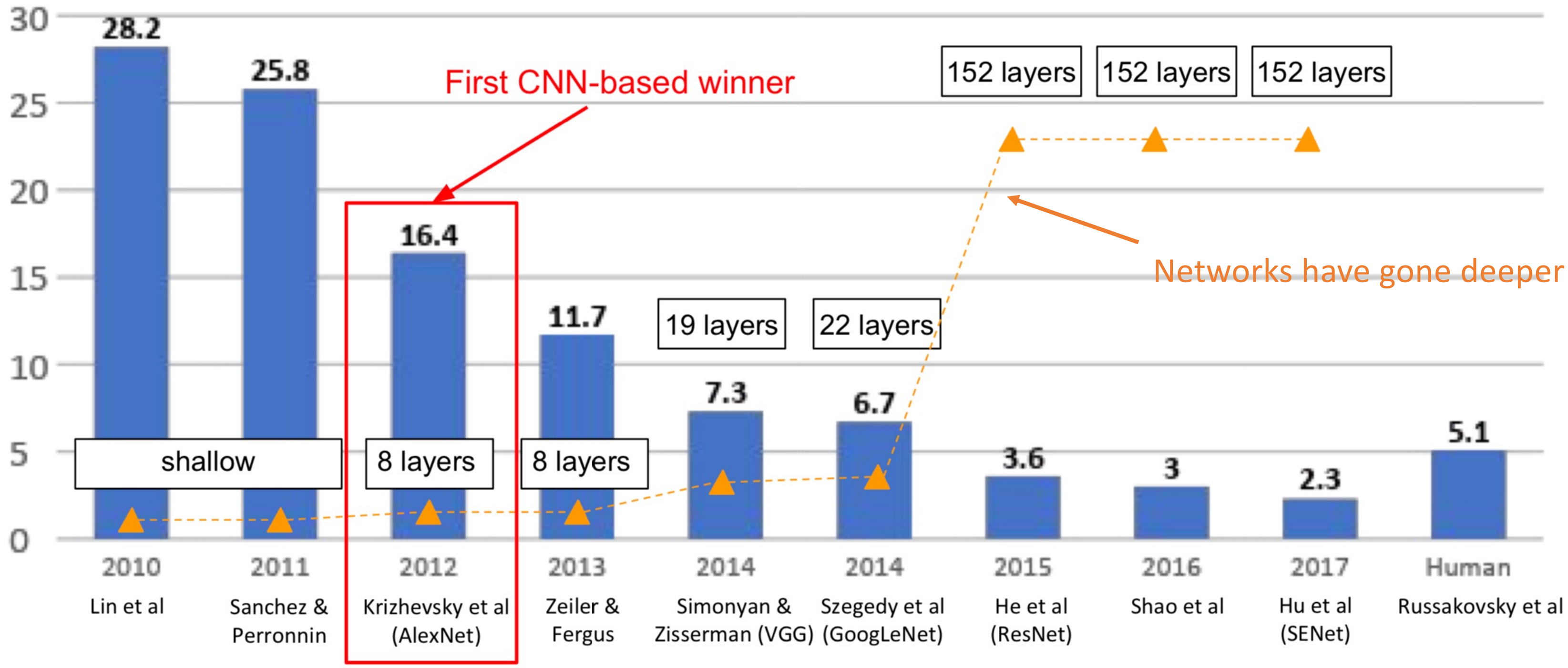
Alexnet architecture details

- ReLU activation functions (first to use in CNNs)
- Training on multiple GPUs
  - Single GTX 580 GPU has only 3GB
  - Network spread across two GPUs; Puts half of the kernels (or neurons) on each GPU
  - Inter-GPU communication only in certain layers
- Overlapping pooling – reduces overfitting
- 8 layers with weights
  - First five are convolutional, 3 max-pool (reduces output size by half)
  - Remaining three are fully- connected
  - Output of the last fully-connected layer is fed to a 1000-way softmax

Alexnet Training

- 60 Million parameters to learn (how did we get this number ? Hw3 question)
- Techniques employed to reduce overfitting
  - Data augmentation (done on CPU) --- pipelined with training, no GPU required, done online
    - Artificially enlarge the dataset using label-preserving transformations
      - Random cropping (224x224 from 256x256) and horizontal reflections
      - Altering intensities of RGB channels in training images
  - Dropout
    - In first two fully connected layers with dropout probability 0.5
- Batch size: 128
- SGD with 0.9 momentum
- Learning rate 1e-2, reduced by 10 manually when validation accuracy plateaus
- L2 weight decay 0.0005
- Error rate: 18.2% (1 CNN)-> 16.4% (5 CNNs ensemble)

Imagenet Large Scale Visual Recognition (ILSVRC) Challenge Winners



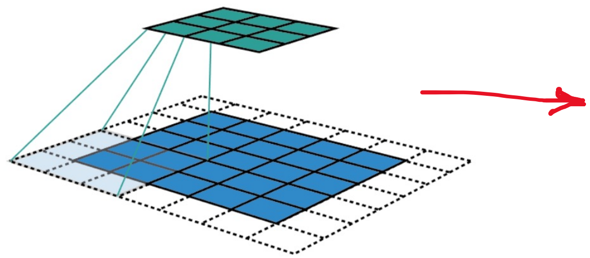
Receptive field

- A convolutional layer operates over a local region of the input to that layer
- The effective **receptive field** of a convolutional layer is the size of the input region to the network that contributes to a layers' activations
- For example:
  - if the first convolutional layer has a receptive field of 3x3 then it's effective receptive field is also 3x3
  - However if the second layer also has a 3x3 filter, then it's (local) receptive field is 3x3, but it's effective receptive field is 5x5

Receptive Field in a 2-d CNN

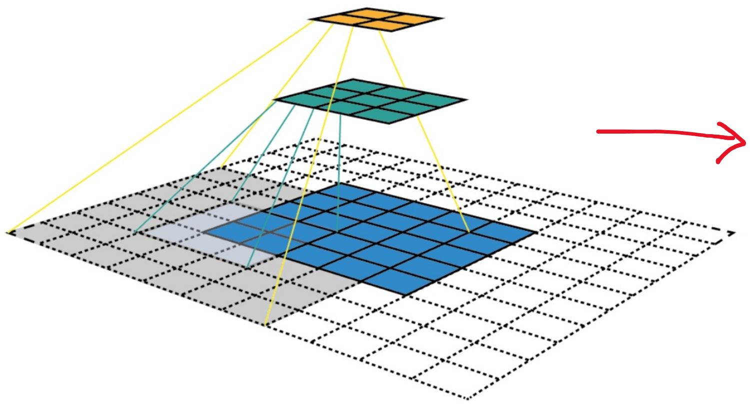
*The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by)*

Convolution with kernel size k = 3x3, padding size p = 1x1, stride s = 2x2



Applying the convolution on a 5x5 input map to produce the 3x3 green feature map

The common way to visualize a CNN feature map. Only looking at the feature map, we do not know where a feature is looking at (the center location of its receptive field) and how big is that region (its receptive field size). It will be impossible to keep track of the receptive field information in a deep CNN.

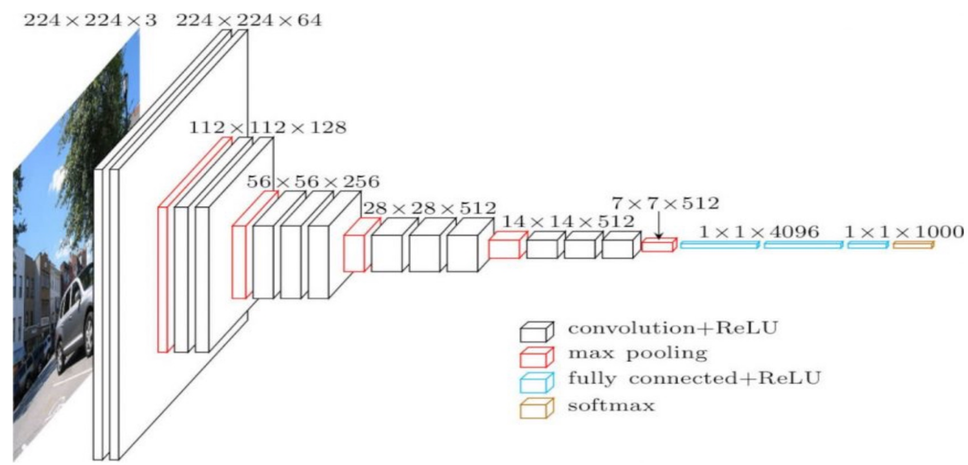


Applying the same convolution on top of the green feature map to produce the 2x2 orange feature map.

https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

VGG (2014)

ConvNet Configuration				VGG16	VGG19
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					



VGG16 Architecture

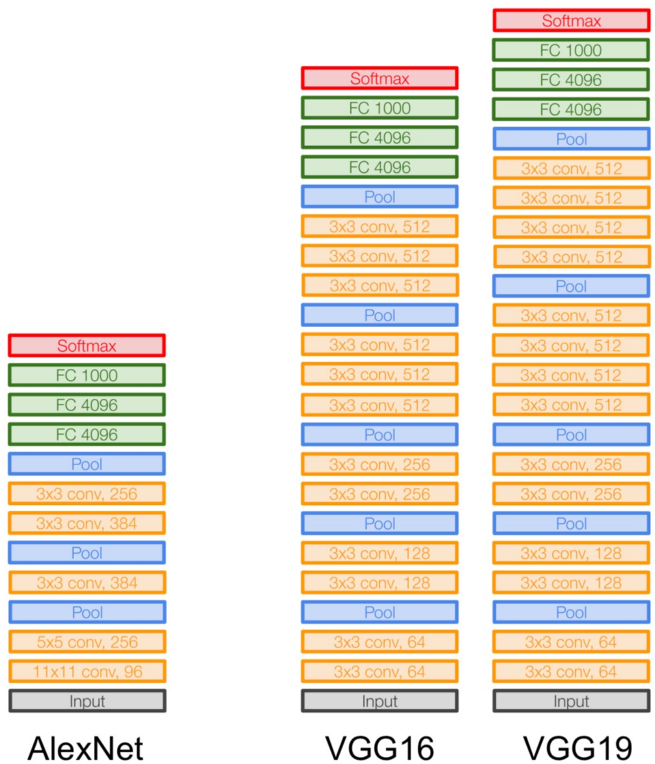
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG memory and compute calculation

Layer	Number of Activations (Memory)	Parameters (Compute)
Input	$224 * 224 * 3 = 150K$	0
CONV3-64	$224 * 224 * 64 = 3.2M$	$(3 * 3 * 3) * 64 = 1728$
CONV3-64	$224 * 224 * 64 = 3.2M$	$(3 * 3 * 64) * 64 = 36{,}864$
POOL2	$112 * 112 * 64 = 800K$	0
CONV3-128	$112 * 112 * 128 = 1.6M$	$(3 * 3 * 64) * 128 = 73{,}728$
CONV3-128	$112 * 112 * 128 = 1.6M$	$(3 * 3 * 128) * 128 = 147{,}456$

VGG Architecture



- Smaller filters (3x3, stride 1) compared to 11x11 and 5x5 in Alexnet
- Deeper nets: more layers compared to Alexnet (8 vs 16 or 19)
- Why smaller filters ?
  - Receptive field of 3 3x3 stacked conv. layers is same as a single 7x7 conv layer
  - More non-linearities with stack of smaller conv layers => makes decision function more discriminative
  - Lesser number of parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer (55% less)

Receptive field equivalence

- Output activation map length: L-F+1
- Single 7x7 filter
  - Output activation map length: L-7+1 = L-6
  - Number of parameters (for input and output depth C): (7x7xC)XC = 7^2C^2
- 3 stacked 3x3 filters
  - Output activation map length after first conv layer: L-3+1 = L-2
  - Output activation map length after second conv layer: (L-2)-3+1=L-4
  - Output activation map length after third conv layer: (L-4)-3+1 = L-6
  - Number of parameters (for input and output depth C):
    - One conv layer: (3x3xC)XC=3^2C^2
    - 3 conv layers: 3.(3^2C^2)
- Parameter saving: 55% (27 vs 49) less with 3 stacked 3x3 vs a single 7x7

Stack of Small Convolution Layers

- A stack of three 3×3 conv. layers instead of a single 7×7 layer
  - Same receptive field
  - 3 non-linear rectification layers instead of one
  - Less number of parameters
- Imposing a regularization on the 7×7 conv. filters, forcing them to have a decomposition through the 3×3 filters (with non-linearity injected in between)

The power of small filters

- Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:

= C x (7 x 7 x C) = **49 C^2**

three CONV with 3 x 3 filters

Number of weights:

= 3 x C x (3 x 3 x C) = **27 C^2**

Fewer parameters, more nonlinearity

Number of multiply-adds:

= (H x W x C) x (7 x 7 x C)

= **49 HWC^2**

Number of multiply-adds:

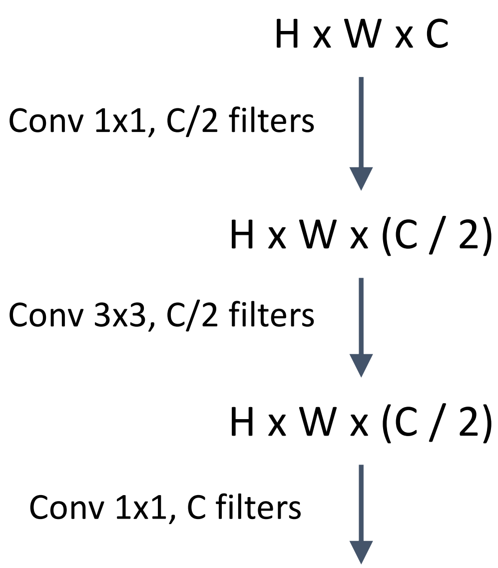
= 3 x (H x W x C) x (3 x 3 x C)

= **27 HWC^2**

Less compute, more nonlinearity

The power of small filters

- Why stop at 3 x 3 filters? → use 1 x 1!



1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, "Network in Network", GoogLeNet, ResNet]

Bottleneck vs single conv

- More nonlinearity, fewer parameters, less compute!

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

Conv 1x1, C filters

Bottleneck sandwich $3.25\ C^2$ parameters

Single 3 x 3 conv $9\ C^2$ parameters

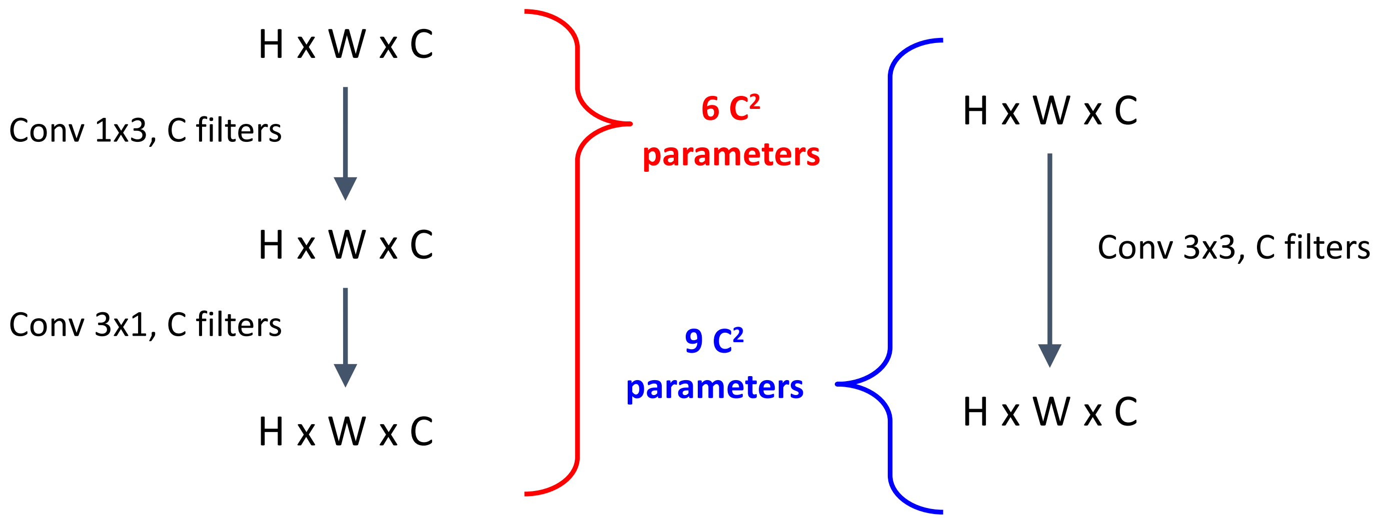
H x W x C

Conv 3x3, C filters

H x W x C

The power of small filters

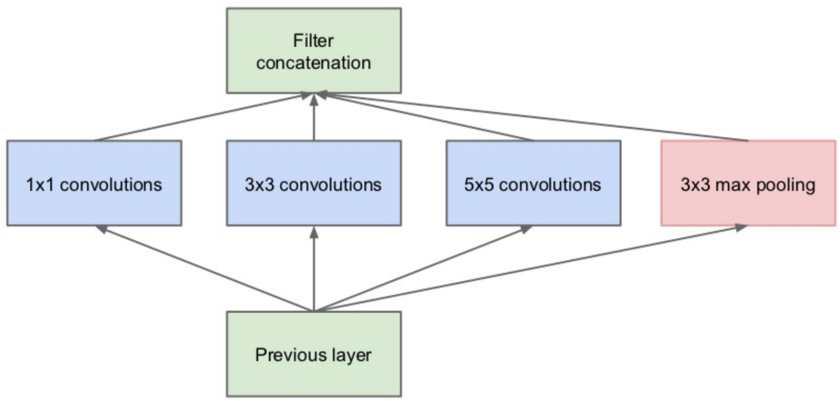
- Still using 3 x 3 filters … can we break it up?



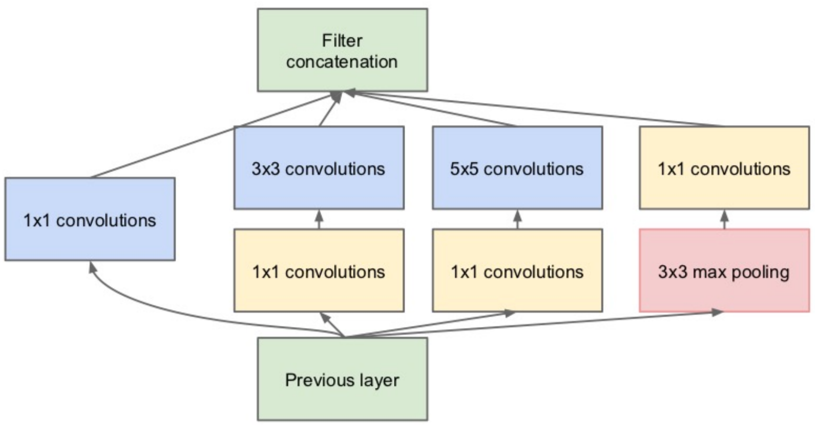
How to stack convolutions - Recap

- Replace large convolutions (5 x 5, 7 x 7) with stacks of 3 x 3 convolutions
- 1 x 1 “bottleneck” convolutions are very efficient
- Replace N x N convolutions into 1 x N and N x 1
- All of the above give fewer parameters, less compute, more nonlinearity
- Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”

GoogleNet (2014)



(a) Inception module, naïve version

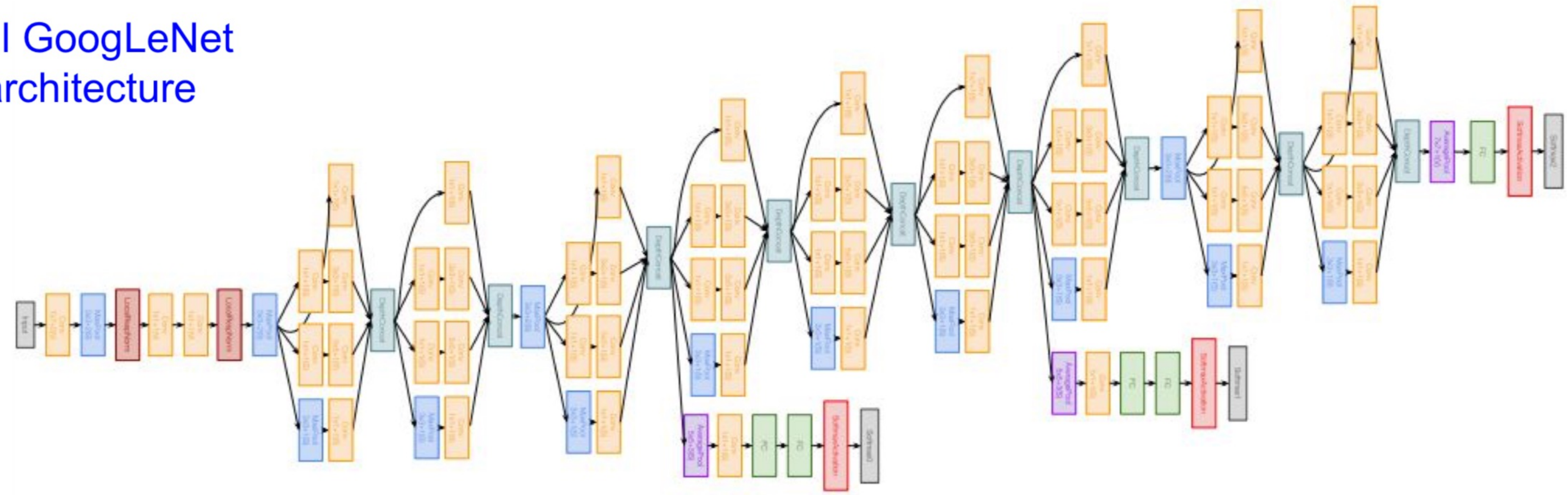


(b) Inception module with dimension reductions

- To preserve local and sparse correlations, parallel convolutional filters of different sizes
- Inception architecture: Inception modules stacked to create Googlenet
- Dimension reduction module has reduced computational complexity compared to naïve version

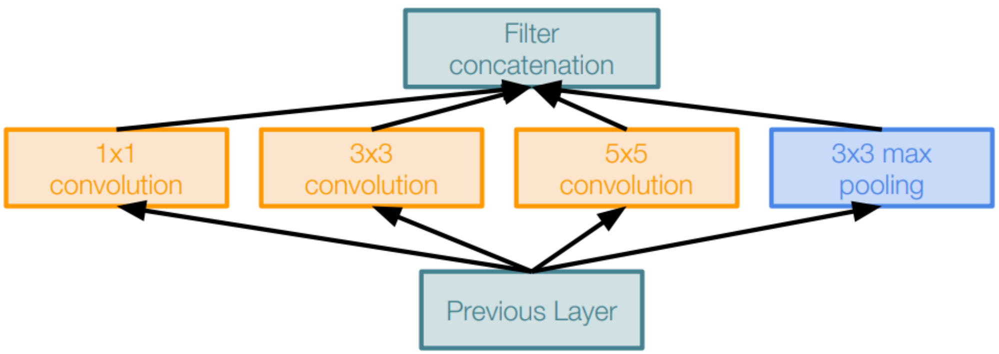
GoogleNet(2014)

Full GoogLeNet architecture



- Deeper network (22 layers)
- Computationally efficient “Inception” module
- Avoids expensive FC layers → uses average pooling
- 12x less params than AlexNet

Inception module

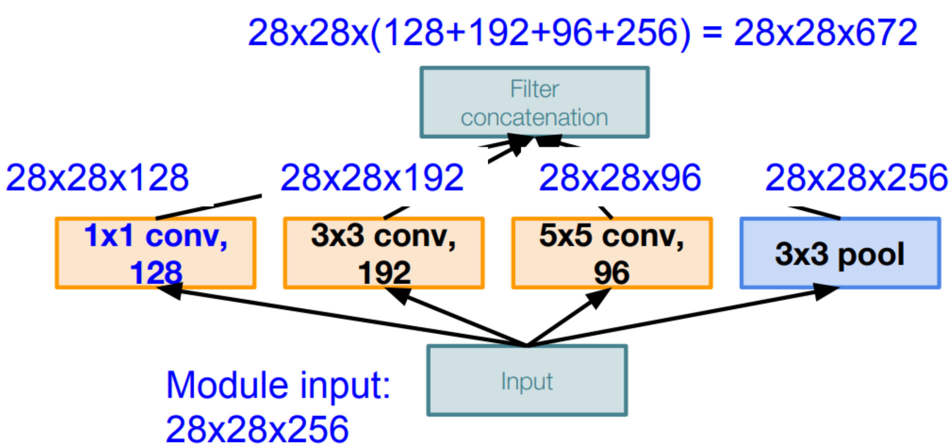


Naive Inception module

- Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise
- What is the problem with this?

→ Computational complexity

Inception module

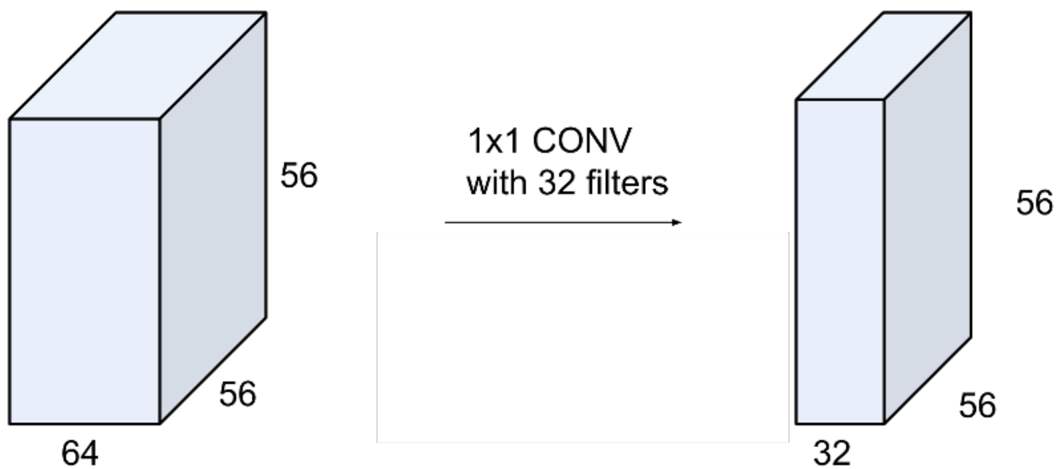


Naive Inception module

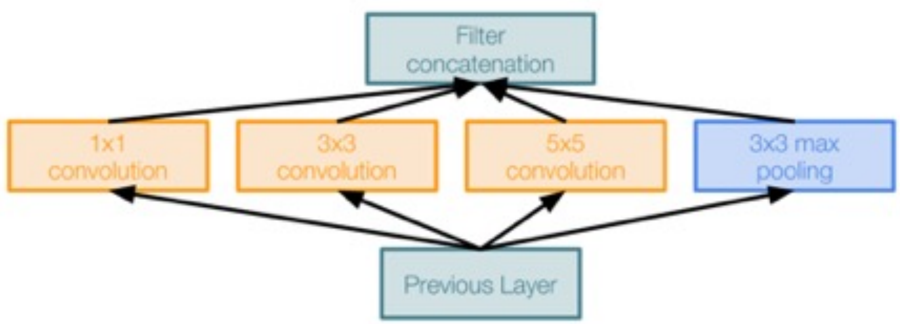
- Conv Ops:
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x256
  - [5x5 conv, 96] 28x28x96x5x5x256
- Total: 854M ops
- Very expensive computation
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!
- Solution: **bottleneck** layers that use 1x1 convolutions to reduce feature depth

1x1 Bottleneck convolution

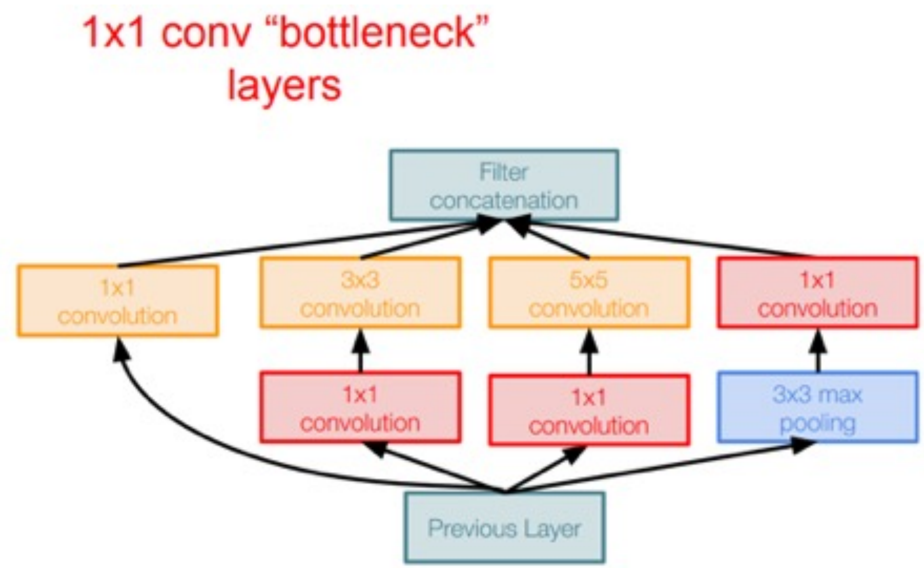
- Each filter has size 1x1xC_{in} and performs a C_{in}-dimensional dot product
- Preserves spatial dimensions, reduces depth!
- Projects depth to lower dimension (= combination of feature maps)



Inception module - Bottleneck layers

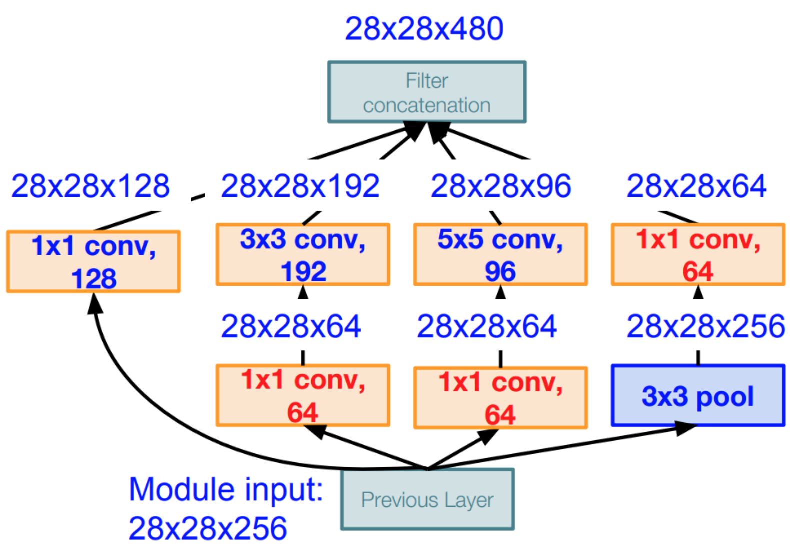


Naive Inception module



Inception module with dimension reduction

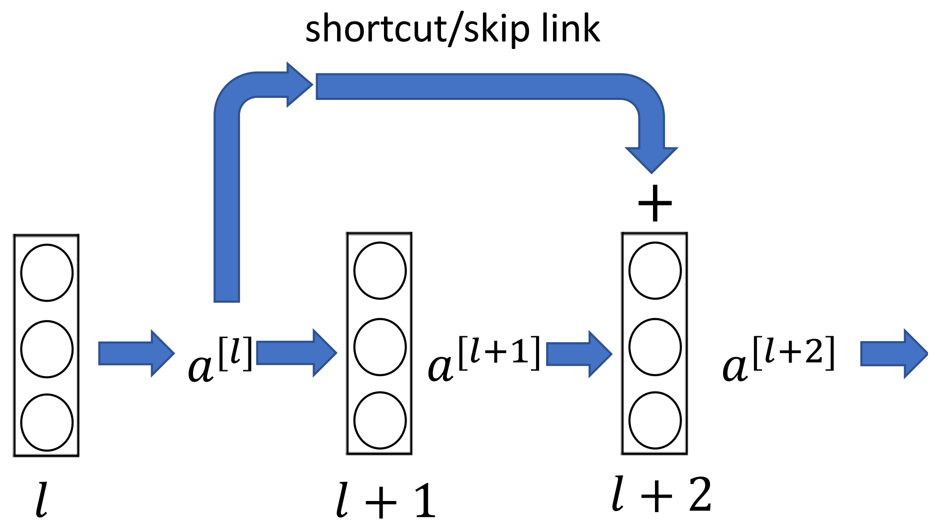
Inception module



- Using same parallel layers as naive example, and adding 1x1 conv with 64 filter bottlenecks:
- Conv Ops:
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x64
  - [5x5 conv, 96] 28x28x96x5x5x64
  - [1x1 conv, 64] 28x28x64x1x1x256
- Total: 358M ops
  - Compared to 854M ops for naive version
- Bottleneck can also reduce depth after pooling layer (480 compared to 672)

ResNet - Residual block

- **The (degradation) problem:**
  - With network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.
- **The core insight:**
  - Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution to the deeper model by construction: the layers are copied from the learned shallower model, and the added layers are identity mapping. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart.



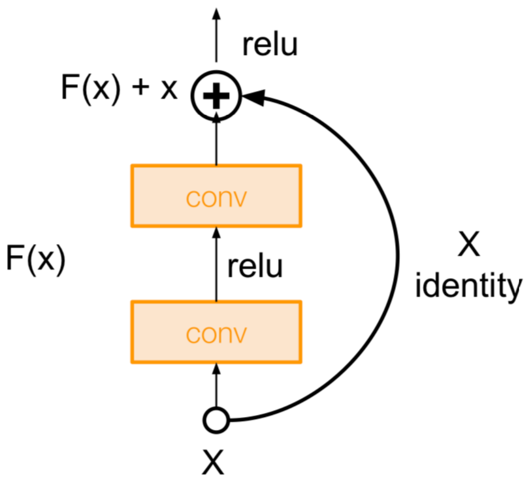
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

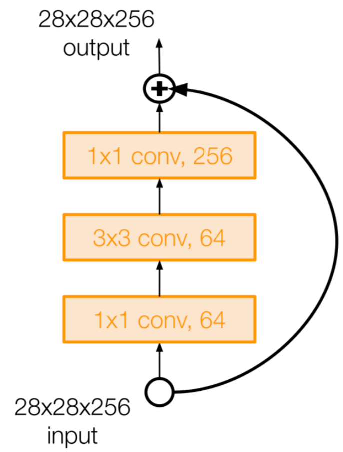
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

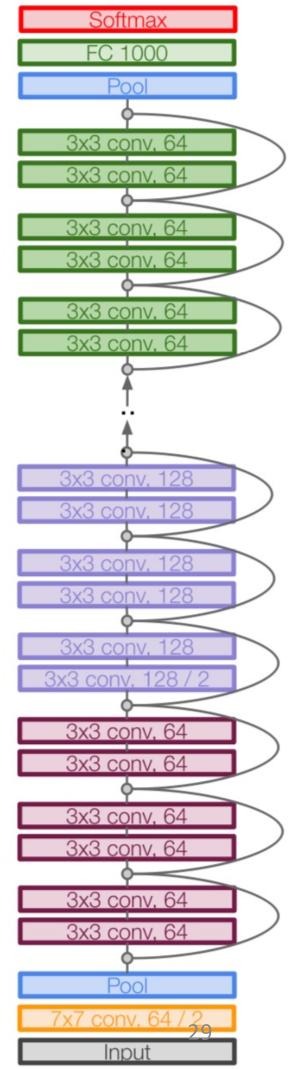
$$a^{[l+2]} = g(z^{[l+2]} \color{red}{+ \boldsymbol{a^{[l]}}})$$

ResNet



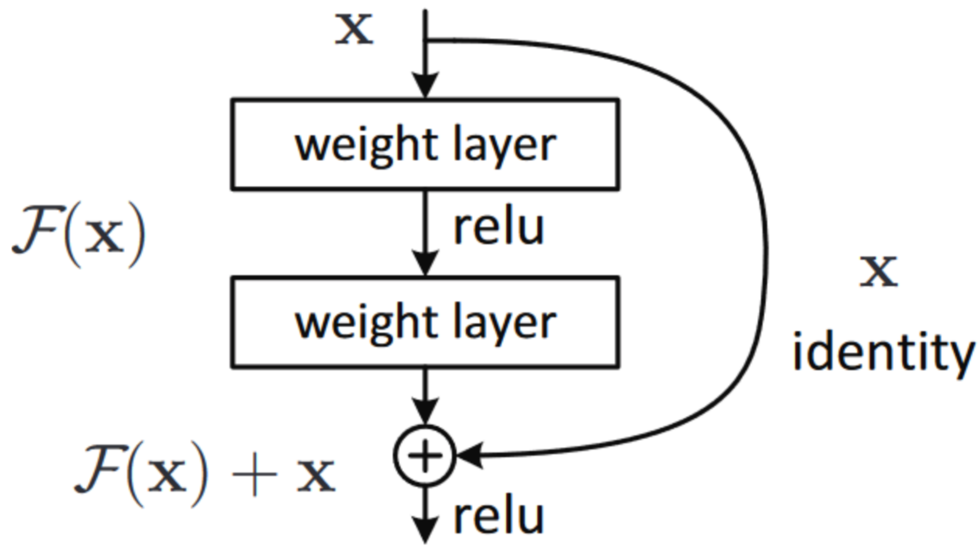
Residual block





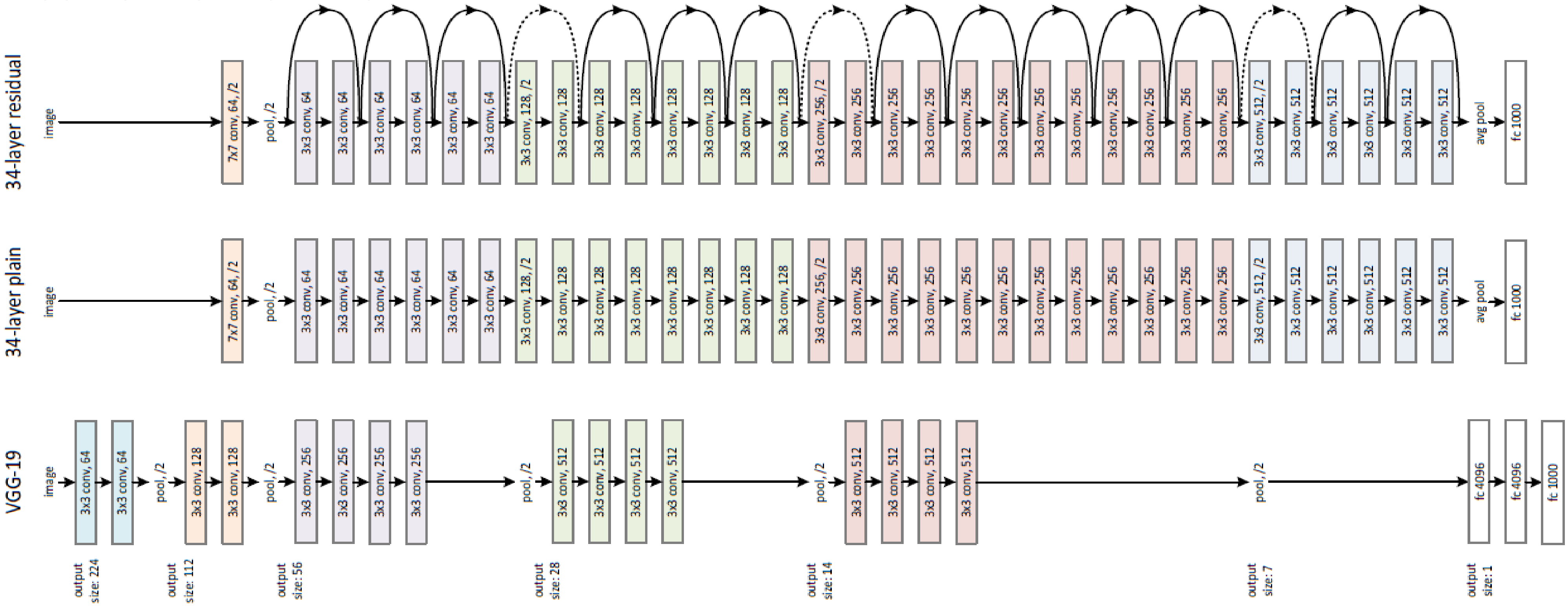
- Stack of residual blocks: each residual block has 2 3x3 conv layers; number of filters is doubled periodically
- Global average pooling layer after last conv layer
- Different depths: 34, 50, 101, 152
- Deeper network (ResNet50+) add 1x1 conv for computational efficiency

Residual connections prevents vanishing gradient



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}\left(\frac{\partial F}{\partial x} + 1\right) = \frac{\partial L}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial L}{\partial H}$$

ResNet vs Plain vs VGG



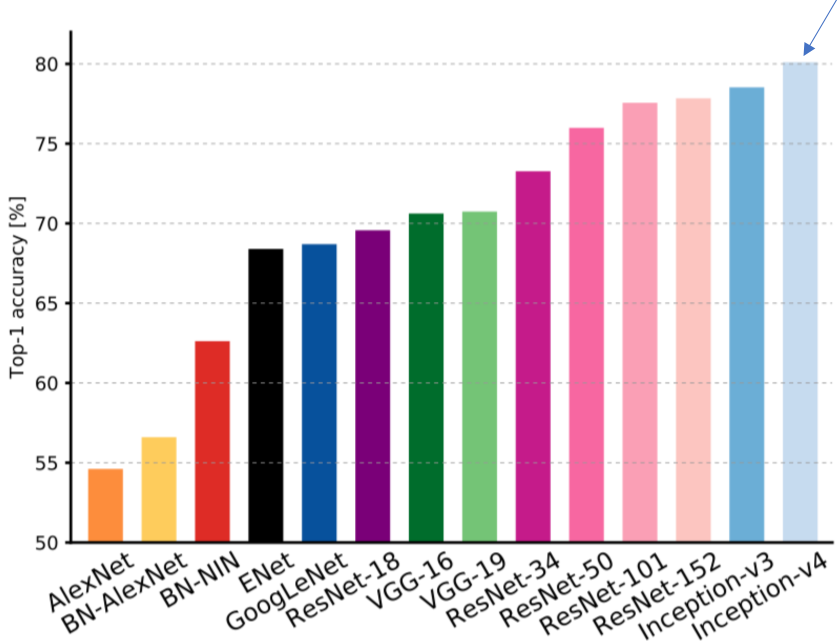
- 34 layer plain becomes 34 layers residual adding shortcut (skip) links
- Dotted shortcut: dimensions change and a transformation is needed to reduce dimensions
- /2 means stride 2 or pool 2x2 (reduce size by 4)

Combining Inception with Residual Connections (Inception V4)

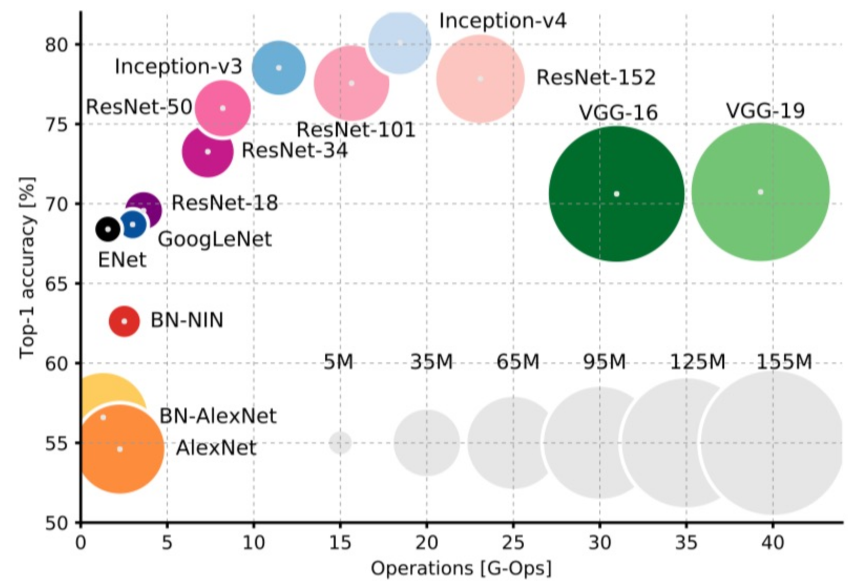
- Whether there are any benefit in combining the Inception architecture with residual connections ?
  - Training with residual connections accelerates the training of Inception networks significantly
  - Some evidence that residual Inception networks outperforms similarly expensive Inception networks without residual connections by a thin margin

Top1 Accuracy Comparison

Inception-v4: Resnet + Inception!



Top1 *vs.* network.



Top1 *vs.* operations, size $\propto$ parameters

Observations

- VGGNet uses the highest memory and the most number of operations, while GoogLeNet is the most efficient in terms of memory and number of operations.
- AlexNet has the lowest accuracy (with smaller compute but heavy memory requirements), while ResNet has the highest accuracy (with moderate efficiency depending on model).