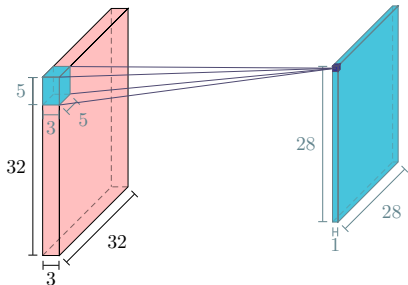


Lecture 7 - More on Convolutional Networks

DD2424

April 21, 2017

More details on the Convolution layers: **Striding & Zero Padding**

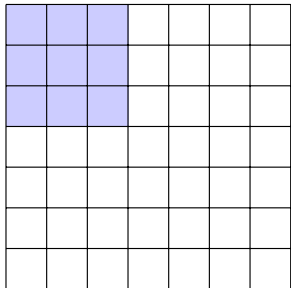


Convolve the image, X , with the filter F .

- Slide filter over all spatial locations in image.
- At each location output 1 number:

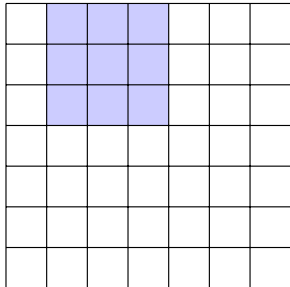
*compute **dot product** between F and a $5 \times 5 \times 3$ chunk of X*

A closer look at the spatial dimensions



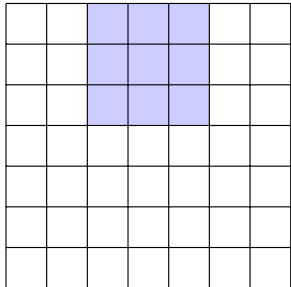
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions



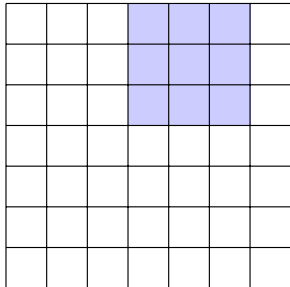
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions



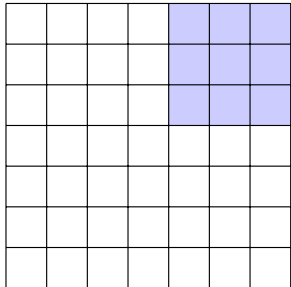
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions



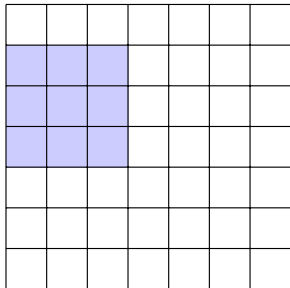
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions



- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions

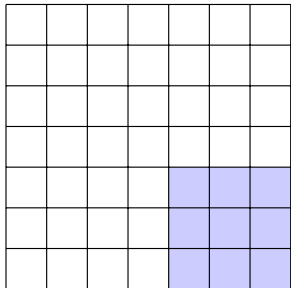


- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.

A closer look at the spatial dimensions

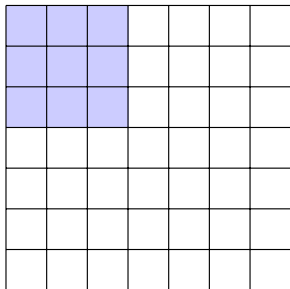
• • •

A closer look at the spatial dimensions



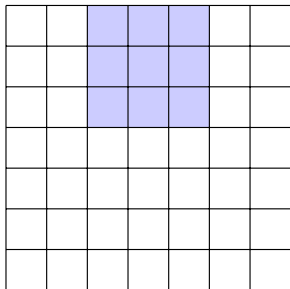
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- $\implies 5 \times 5$ output.

A closer look at the spatial dimensions



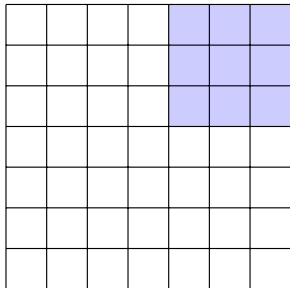
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride 2**.

A closer look at the spatial dimensions



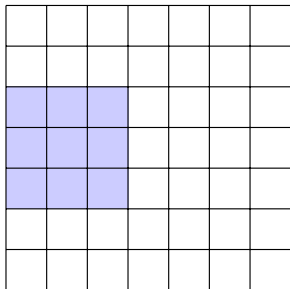
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** 2.

A closer look at the spatial dimensions



- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride 2**.

A closer look at the spatial dimensions

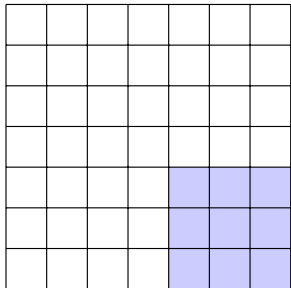


- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** 2.

A closer look at the spatial dimensions

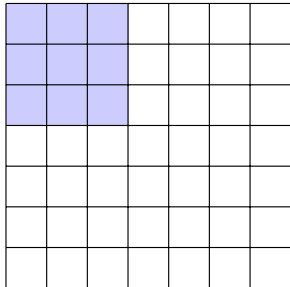
• • •

A closer look at the spatial dimensions



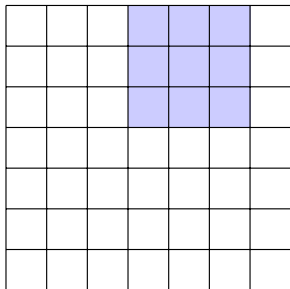
- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** 2.
- $\implies 3 \times 3$ output.

A closer look at the spatial dimensions

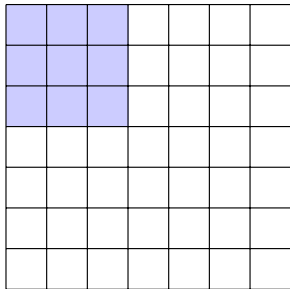


- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** 3.

A closer look at the spatial dimensions



- $7 \times 7 \times D$ input (just display one plane).
- Have $3 \times 3 \times D$ filter.
- Apply with **stride 3**.
- **Doesn't fit nicely!** Don't include last column and row.

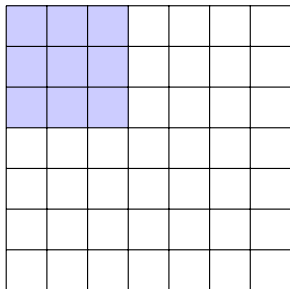


- $w \times h \times D$ input.
- Have $f \times f \times D$ filter.
- Apply with **stride** S .
- **Output dimension:** $w' \times h'$

$$w' = (w - f) / S + 1$$

$$h' = (h - f) / S + 1$$

Output volume dimension



- $w \times h \times D$ input.
- Have $f \times f \times D$ filter.
- Apply with **stride** S .
- **Output dimension:** $w' \times h'$

$$w' = (w - f) / S + 1$$

$$h' = (h - f) / S + 1$$

- **Our example:**
 $w = h = 7, f = 3$

$$S=1 \implies w' = (7 - 3) / 1 + 1 = 5$$

$$S=2 \implies w' = (7 - 3) / 2 + 1 = 3$$

$$S=3 \implies w' = (7 - 3) / 3 + 1 = 2.33$$

In practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- $7 \times 7 \times D$ input.
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** S .
- Pad input with 1 pixel border.
- **Output dimension:** ?
- Remember

$$w' = (w - f) / S + 1$$

$$h' = (h - f) / S + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- $7 \times 7 \times D$ input.
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** S .
- Pad input with 1 pixel border.
- **Output dimension:** ?
- Remember

$$w' = (w - f) / S + 1$$

$$h' = (h - f) / S + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

- $7 \times 7 \times D$ input.
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** S .
- Pad input with 1 pixel border.
- **Output dimension:** 7×7

In practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- $7 \times 7 \times D$ input.
- Have $3 \times 3 \times D$ filter.
- Apply with **stride** S .
- Pad input with 1 pixel border. ($P = 1$)
- **Output dimension:** 7×7
- In general

$$w' = (w + 2P - f) / S + 1$$

$$h' = (h + 2P - f) / S + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- $7 \times 7 \times D$ input.
 - Have $3 \times 3 \times D$ filter.
 - Apply with **stride** S .
 - Pad input with 1 pixel border. ($P = 1$)
 - **Output dimension:** 7×7
 - In general, common to have convolutional layers with
 - stride $S = 1$,
 - filters of size $f \times f \times D$,
 - zero-padding $P = (f - 1)/2$
- $\implies w' = w, h' = h$

- **Input volume dimension:** $32 \times 32 \times 3$
- Hyper-parameters of Conv layer:
 - 10 filters of size $5 \times 5 \times 3$
 - Stride: $S = 1$
 - Pad: $P = 2$
- **Output volume dimension:** ?

Example of Dimension Counting

- **Input volume dimension:** $32 \times 32 \times 3$
- Hyper-parameters of Conv layer:
 - 10 filters of size $5 \times 5 \times 3$
 - Stride: $S = 1$
 - Pad: $P = 2$
- **Output volume dimension:** $w' \times h' \times 10$
where

$$w' = (w - 2P - f)/S + 1 = (32 - 2 * 2 - 5)/1 + 1 = 32$$

$$h' = (h - 2P - f)/S + 1 = (32 - 2 * 2 - 5)/1 + 1 = 32$$

Example of Dimension Counting

- **Input volume dimension:** $32 \times 32 \times 3$
- **Hyper-parameters of Conv layer:**
 - 10 filters of size $5 \times 5 \times 3$
 - Stride $S = 1$
 - Pad $P = 2$
- **# of parameters in this layer: ?**

Example of Dimension Counting

- **Input volume dimension:** $32 \times 32 \times 3$
- **Hyper-parameters of Conv layer:**
 - 10 filters of size $5 \times 5 \times 3$
 - Stride: $S = 1$
 - Pad: $P = 2$
- **# of parameters in this layer:** 760
 - Each filter has $5 \times 5 \times 3 + 1 = 76$ parameters.
 - There are 10 filters $\implies 76 * 10$ total parameters.

Summary of dimensions in Convolutional Layer

- **Input:**

Volume, $X^{(i)} = \{X_1^{(i)}, \dots, X_D^{(i)}\}$, of size $w \times h \times D$

- Requires 4 hyper-parameters:

- Number of filters: n_F
- Spatial size of filters: f
- Stride: S
- Amount of zero padding: P

- **Output:**

Volume $S^{(i+1)} = \{S_1^{(i+1)}, \dots, S_{n_f}^{(i+1)}\}$ size $w' \times h' \times n_F$ where

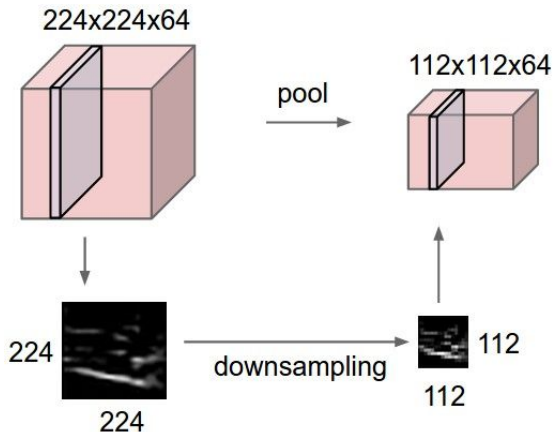
- $w' = (w - f + 2P)/S + 1$
- $h' = (h - f + 2P)/S + 1$

where

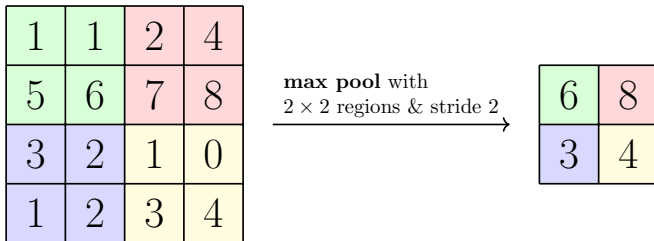
$$S_j^{(i+1)} = X^{(i)} * F_j + b_j$$

- $n_F = \text{powers of 2 (e.g. 32, 64, 128, 512)}$
 - $f = 3, S = 1, P = 1$
 - $f = 5, S = 1, P = 2$
 - $f = 5, S = 2, P = ?$ (whatever fits)
 - $f = 1, S = 1, P = 1$

- Make the representation smaller and more manageable
- Operates over each response/activation map independently



Simple Example: Max Pooling one response map



- Denote the max pooling operation with regions of size $r \times r$ and stride s with

$$\tilde{H} = \text{MaxPool}(H, r, s)$$

- If H has size $w \times h \times D$
 $\implies \tilde{H}$ has size $(w - r)/s + 1 \times (h - r)/s + 1 \times D$
- Mathematical expression for each entry in $\tilde{H}$:

$$\tilde{H}_{ijk} = \max_{\substack{i' \leq l \leq i' + r - 1 \\ j' \leq m \leq j' + r - 1}} H_{lmk} \quad \text{where } i' = (i - 1)s + 1, j' = (j - 1)s + 1$$

- Common settings $r = 2, s = 2$ or $r = 3, s = 2$.

- Denote the max pooling operation with regions of size $r \times r$ and stride s with

$$\tilde{H} = \text{MaxPool}(H, r, s)$$

- If H has size $w \times h \times D$
 $\implies \tilde{H}$ has size $(w - r)/s + 1 \times (h - r)/s + 1 \times D$
- Mathematical expression for each entry in $\tilde{H}$:

$$\tilde{H}_{ijk} = \max_{\substack{i' \leq l \leq i' + r - 1 \\ j' \leq m \leq j' + r - 1}} H_{lmk} \quad \text{where } i' = (i - 1)s + 1, j' = (j - 1)s + 1$$

- Common settings $r = 2, s = 2$ or $r = 3, s = 2$.

Jacobian Computations for a Max Pooling layer

- Let

$$\tilde{H} = \text{MaxPool}(H, r, r)$$

- For backprop need to calculate:

$$\frac{\partial \text{vec}(\tilde{H})}{\partial \text{vec}(H)} = ?$$

- Can write MaxPool operation as a matrix operation

$$\text{vec}(\tilde{H}) = A_{\text{MaxPool}} \text{vec}(H)$$

$\Rightarrow$

$$\frac{\partial \text{vec}(\tilde{H})}{\partial \text{vec}(H)} = A_{\text{MaxPool}}$$

- What are the entries of A_{MaxPool} ?

Jacobian Computations for a Max Pooling layer

- Let

$$\tilde{H} = \text{MaxPool}(H, r, r)$$

- For backprop need to calculate:

$$\frac{\partial \text{vec}(\tilde{H})}{\partial \text{vec}(H)} = ?$$

- Can write MaxPool operation as a matrix operation

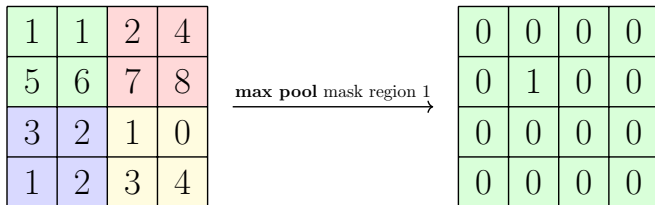
$$\text{vec}(\tilde{H}) = A_{\text{MaxPool}} \text{vec}(H)$$

$\implies$

$$\frac{\partial \text{vec}(\tilde{H})}{\partial \text{vec}(H)} = A_{\text{MaxPool}}$$

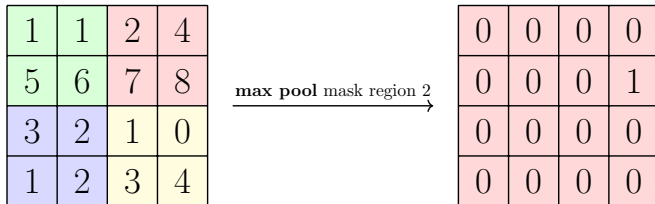
- What are the entries of A_{MaxPool} ?

Simple Example: Max Pooling as a matrix multiplication



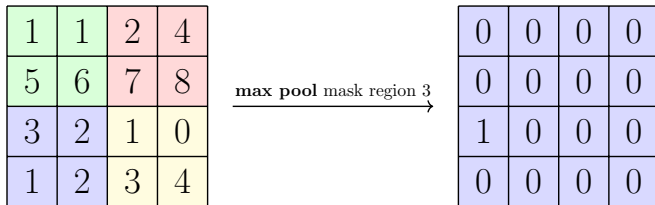
$$(\tilde{H}_{11}) = (0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) \text{vec}(H)$$

Simple Example: Max Pooling as a matrix multiplication



$$\begin{pmatrix} \tilde{H}_{11} \\ \tilde{H}_{12} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{vec}(H)$$

Simple Example: Max Pooling as a matrix multiplication



$$\begin{pmatrix} \tilde{H}_{11} \\ \tilde{H}_{12} \\ \tilde{H}_{21} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{vec}(H)$$

Simple Example: Max Pooling as a matrix multiplication



[illegible]

[illegible]

Simple Example: Max Pooling as a matrix multiplication

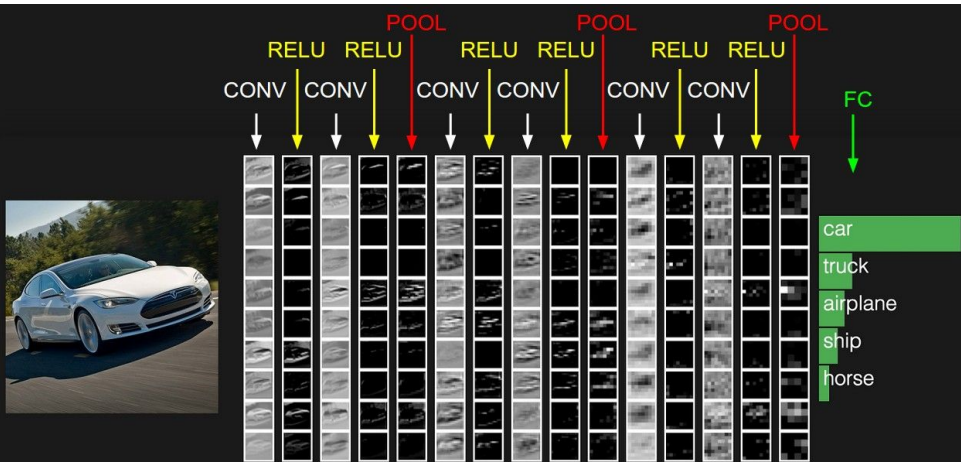


$$\begin{pmatrix} \tilde{H}_{11} \\ \tilde{H}_{12} \\ \tilde{H}_{21} \\ \tilde{H}_{22} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{vec}(H)$$

ool and has size 4×16

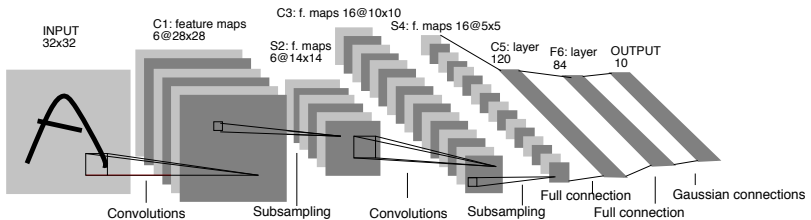
- Say in each region we pool by **averaging** the responses instead of choosing the max.
- What is A_{AvgPool} for our simple example?

preview:



Architecture of Modern ConvNets

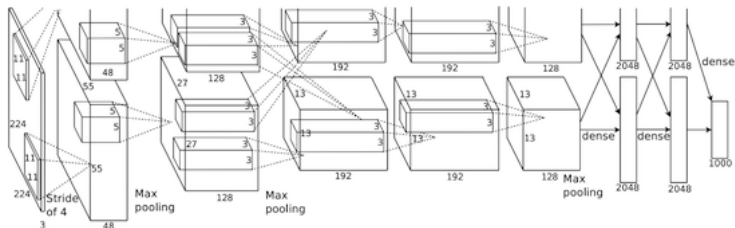
LeNet-5 [LeCun et al., 1998]



- Conv filters are 5×5 , applied at stride 1
- Pooling layers are 2×2 , applied at stride 2
- Architecture is

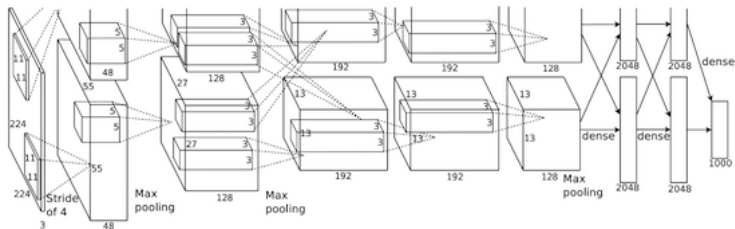
[CONV-POOL-TANH-CONV-POOL-TANH-FC-TANH-FC-TANH-FC]

AlexNet [Krizhevsky et al. 2012]



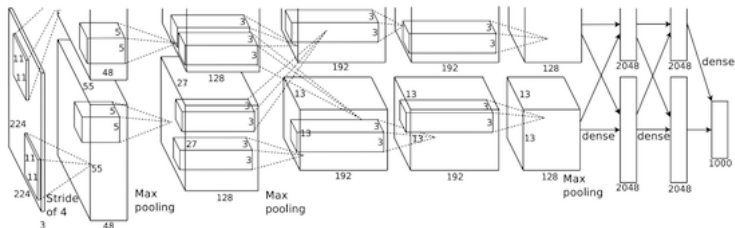
- **Input:** $227 \times 227 \times 3$ image
- **First layer:**
 - Convolutional layer
 - 96 filters of size $11 \times 11 \times 3$ applied at stride 4
- What is the output volume size?

AlexNet [Krizhevsky et al. 2012]



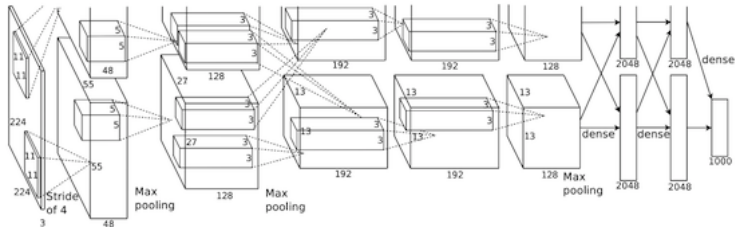
- **Input:** $227 \times 227 \times 3$ image
- **First layer:**
 - Convolutional layer
 - 96 filters of size $11 \times 11 \times 3$ applied at stride 4
- **What is the output volume size?** $55 \times 55 \times 96$
as $(227 - 11)/4 + 1 = 55$.

AlexNet [Krizhevsky et al. 2012]



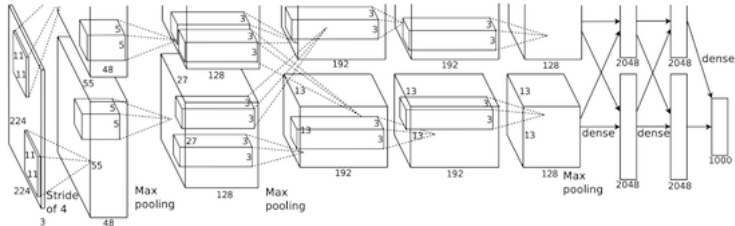
- **Input:** $227 \times 227 \times 3$ image
- **First layer:**
 - Convolutional layer
 - 96 filters of size $11 \times 11 \times 3$ applied at stride 4
- **What is the output volume size?** $55 \times 55 \times 96$
- **# of parameters:** $11 * 11 * 3 * 96 = 35k$

AlexNet [Krizhevsky et al. 2012]



- **Input size:** $227 \times 227 \times 3$ image
- **Size after 1st layer:** $55 \times 55 \times 96$ image
- **Second layer:**
 - Pooling layer
 - 3×3 regions applied at stride 2
- What is the output volume size?

AlexNet [Krizhevsky et al. 2012]



- **Input size:** $227 \times 227 \times 3$ image
- **Size after 1st layer:** $55 \times 55 \times 96$ image
- **Second layer:**
 - Pooling layer
 - 3×3 regions applied at stride 2
- **What is the output volume size?** $27 \times 27 \times 96$
as $(55 - 3)/2 + 1 = 27$.

Full (simplified) AlexNet architecture:

Output size	Layer type	Details
$227 \times 227 \times 3$	Input	
$55 \times 55 \times 96$	Conv	96 filters, size 11×11 at stride 4, pad 0
$27 \times 27 \times 96$	Max Pool	3×3 regions at stride 2
$27 \times 27 \times 96$	Norm	Normalization layer
$27 \times 27 \times 256$	Conv	256 filters, size 5×5 at stride 1, pad 2
$13 \times 13 \times 256$	Max Pool	3×3 regions at stride 2
$13 \times 13 \times 256$	Norm	Normalization layer
$13 \times 13 \times 384$	Conv	384 filters, size 3×3 at stride 1, pad 1
$13 \times 13 \times 384$	Conv	384 filters, size 3×3 at stride 1, pad 1
$13 \times 13 \times 256$	Conv	256 filters, size 3×3 at stride 1, pad 1
$6 \times 6 \times 256$	Max Pool	3×3 regions at stride 2
4096	Fully connected	4096 neurons
4096	Fully connected	4096 neurons
1000	Fully connected	1000 neurons (class scores)

Details/Retrospectives:

- First use of ReLU
- Used Normalization layers (not common anymore).
- Heavy data augmentation
- Dropout training with $p = 0.5$
- Batch size: 128
- SGD + Momentum with $\alpha = 0.9$
- Learning rate initialized: 1e-2; divided by 10 when validation accuracy plateaus.
- L2 weight decay 5e-4
- 18.2% (1 CNN) $\rightarrow$ 15.4% (7 CNN ensemble)

VGGNet [Simonyan and Zisserman, 2014]

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					

The 6 different architectures of VGG Net. Configuration D produced the best results

Best model marked by the yellow box.

VGGNet [Simonyan and Zisserman, 2014]

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
maxpool					
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					

The 6 different architectures of VGG Net. Configuration D produced the best results

- Only filters of size 3×3 in the convolutional layers, stride 1 and pad 1.
- Relatively sparse use of **Max Pooling** with region size 2×2 and stride 2.
- **Improved ILSVRC Performance** top 5 error: 2013 best 11.2% → 7.3% (VGGNet ensemble)

VGGNet architecture [Simonyan and Zisserman, 2014]

Output size	Layer type	Memory	# parameters
$224 \times 224 \times 3$	Input	$224*224*3 = 150K$	0
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2M$	$1,728 = (3*3*3)*64$
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2M$	$36,864 = (3*3*64)*64$
$112 \times 112 \times 64$	Max Pool	$112*112*64 = 800K$	0
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6M$	$73,728 = (3*3*64)*128$
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6M$	$147,456 = (3*3*128)*128$
$56 \times 56 \times 128$	Max Pool	$56*56*128 = 400K$	0
$56 \times 56 \times 256$	Conv	$56*56*256 = 800K$	$294,912 = (3*3*128)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800K$	$589,824 = (3*3*256)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800K$	$589,824 = (3*3*256)*256$
$28 \times 28 \times 256$	Max Pool	$28*28*256 = 200K$	0
$28 \times 28 \times 512$	Conv	$28*28*512 = 400K$	$1,179,648 = (3*3*256)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400K$	$2,359,296 = (3*3*512)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400K$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Max Pool	$14*14*512 = 100K$	0
$14 \times 14 \times 512$	Conv	$14*14*512 = 100K$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100K$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100K$	$2,359,296 = (3*3*512)*512$
$7 \times 7 \times 512$	Max Pool	$7*7*512 = 25K$	0
$1 \times 1 \times 4096$	Fully connected	4096	$102,760,448 = 7*7*512*4096$
$1 \times 1 \times 4096$	Fully connected	4096	$16,777,216 = 4096*4096$
$1 \times 1 \times 1000$	Fully connected	1000	$4,096,000 = 4096*1000$

VGGNet architecture [Simonyan and Zisserman, 2014]

Output size	Layer type	Memory	# parameters
$224 \times 224 \times 3$	Input	$224*224*3 = 150\text{K}$	0
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2\text{M}$	$1,728 = (3*3*3)*64$
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2\text{M}$	$36,864 = (3*3*64)*64$
$112 \times 112 \times 64$	Max Pool	$112*112*64 = 800\text{K}$	0
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6\text{M}$	$73,728 = (3*3*64)*128$
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6\text{M}$	$147,456 = (3*3*128)*128$
$56 \times 56 \times 128$	Max Pool	$56*56*128 = 400\text{K}$	0
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$294,912 = (3*3*128)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$589,824 = (3*3*256)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$589,824 = (3*3*256)*256$
$28 \times 28 \times 256$	Max Pool	$28*28*256 = 200\text{K}$	0
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$1,179,648 = (3*3*256)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$2,359,296 = (3*3*512)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Max Pool	$14*14*512 = 100\text{K}$	0
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$7 \times 7 \times 512$	Max Pool	$7*7*512 = 25\text{K}$	0
$1 \times 1 \times 4096$	Fully connected	4096	$102,760,448 = 7*7*512*4096$
$1 \times 1 \times 4096$	Fully connected	4096	$16,777,216 = 4096*4096$
$1 \times 1 \times 1000$	Fully connected	1000	$4,096,000 = 4096*1000$

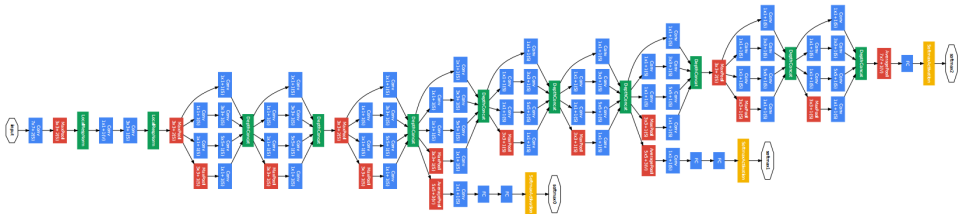
- **Total memory:** $24\text{M} * 4 \text{ bytes} \approx 93\text{MB}$ per image (only for fwd pass, $\approx *2$ for bwd pass)
- **Total parameters:** 128 Million parameters

VGGNet architecture [Simonyan and Zisserman, 2014]

Output size	Layer type	Memory	# parameters
$224 \times 224 \times 3$	Input	$224*224*3 = 150\text{K}$	0
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2\text{M}$	$1,728 = (3*3*3)*64$
$224 \times 224 \times 64$	Conv	$224*224*64 = 3.2\text{M}$	$36,864 = (3*3*64)*64$
$112 \times 112 \times 64$	Max Pool	$112*112*64 = 800\text{K}$	0
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6\text{M}$	$73,728 = (3*3*64)*128$
$112 \times 112 \times 128$	Conv	$112*112*128 = 1.6\text{M}$	$147,456 = (3*3*128)*128$
$56 \times 56 \times 128$	Max Pool	$56*56*128 = 400\text{K}$	0
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$294,912 = (3*3*128)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$589,824 = (3*3*256)*256$
$56 \times 56 \times 256$	Conv	$56*56*256 = 800\text{K}$	$589,824 = (3*3*256)*256$
$28 \times 28 \times 256$	Max Pool	$28*28*256 = 200\text{K}$	0
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$1,179,648 = (3*3*256)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$2,359,296 = (3*3*512)*512$
$28 \times 28 \times 512$	Conv	$28*28*512 = 400\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Max Pool	$14*14*512 = 100\text{K}$	0
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$14 \times 14 \times 512$	Conv	$14*14*512 = 100\text{K}$	$2,359,296 = (3*3*512)*512$
$7 \times 7 \times 512$	Max Pool	$7*7*512 = 25\text{K}$	0
$1 \times 1 \times 4096$	Fully connected	4096	$102,760,448 = 7*7*512*4096$
$1 \times 1 \times 4096$	Fully connected	4096	$16,777,216 = 4096*4096$
$1 \times 1 \times 1000$	Fully connected	1000	$4,096,000 = 4096*1000$

- Most memory is in early convolutional layers.
- Most parameters in the early fully connected layers.

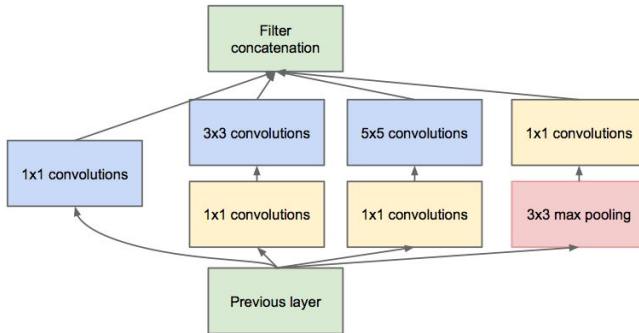
GoogLeNet [Szegedy et al., 2014]



- Convolution layer
- Pooling layer
- SoftMax layer

ILSVRC 2014 winner (6.7% top 5 error)

GoogLeNet: Inception Module

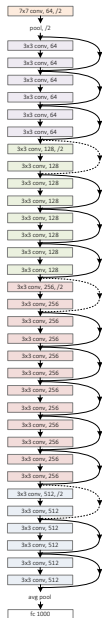


Use $1 \times 1 \times D$ filters to reduce the depth of the response volume before applying the larger more computationally expensive filters.

GoogLeNet: architecture

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								

- Only 5 million params! (Only one FC layer for convenience)
- Compared to AlexNet:
 - 12× less params
 - 2× more compute
 - Performance on ILSVRC: 6.67% (vs. 16.4%)



- ILSVRC 2015 winner (3.6% top 5 error)
- 2-3 weeks of training on 8 GPU machine
- At runtime: faster than a VGGNet! (even though it has $8\times$ more layers)

← “shallow” version of winning entry.

Next slides from:

Deep Residual Networks, Deep Learning Gets Way Deeper by Kaiming He,
ICML 2016 tutorial.

ResNets @ ILSVRC & COCO 2015 Competitions

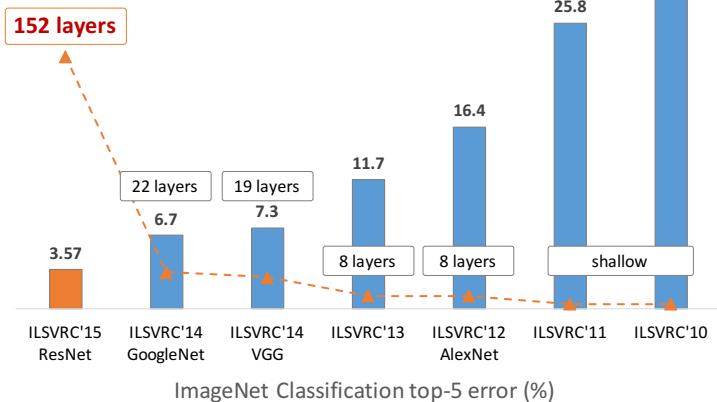
- **1st places in all five main tracks**

- ImageNet Classification: “*Ultra-deep*” 152-layer nets
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

Revolution of Depth



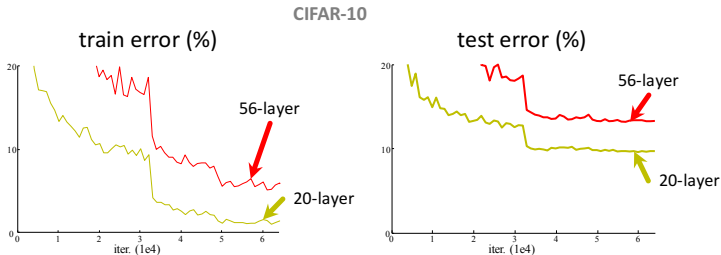
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Going Deeper requires

- Care with initialization ✓
- Batch Normalization ✓

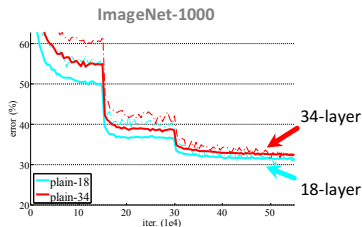
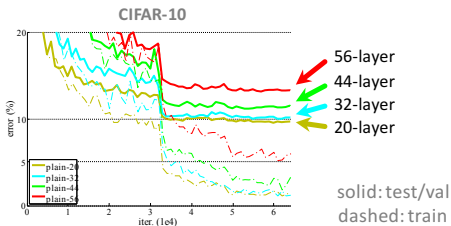
Is learning better networks as simple as stacking more layers?

Simply stacking layers?



- *Plain* nets: stacking 3x3 conv layers...
- 56-layer net has **higher training error** and test error than 20-layer net

Simply stacking layers?

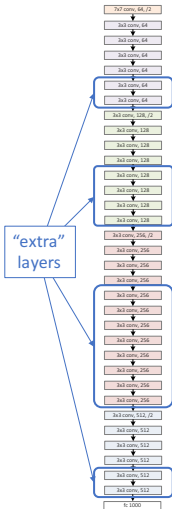


- “Overly deep” plain nets have **higher training error**
- A general phenomenon, observed in many datasets

a shallower model
(18 layers)



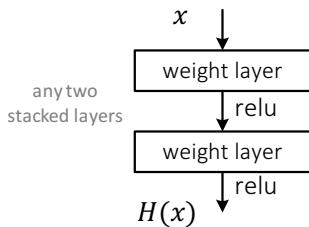
a deeper counterpart
(34 layers)



- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
 - original layers: copied from a learned shallower model
 - extra layers: set as **identity**
 - at least the same training error
- **Optimization difficulties**: solvers cannot find the solution when going deeper...

Deep Residual Learning

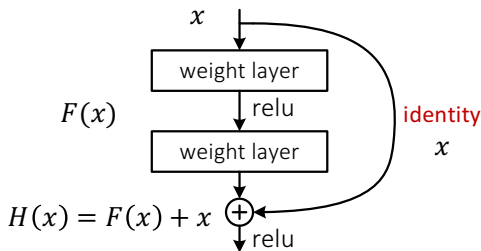
- Plain net



$H(x)$ is any desired mapping,
hope the 2 weight layers fit $H(x)$

Deep Residual Learning

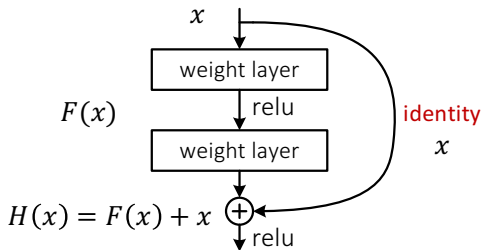
- **Residual** net



$H(x)$ is any desired mapping,
~~hope the 2 weight layers fit $H(x)$~~
hope the 2 weight layers fit $F(x)$
let $H(x) = F(x) + x$

Deep Residual Learning

- $F(x)$ is a **residual** mapping w.r.t. **identity**

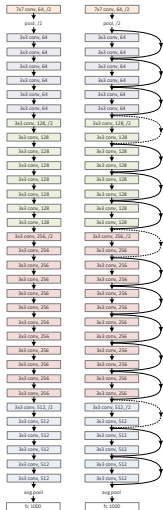


- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Network “Design”

- Keep it simple
- Our basic design (VGG-style)
 - all 3x3 conv (almost)
 - spatial size /2 => # filters x2 (~same complexity per layer)
 - **Simple design; just deep!**
- Other remarks:
 - no hidden fc
 - no dropout

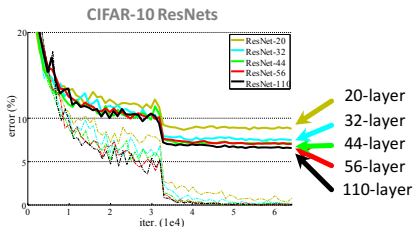
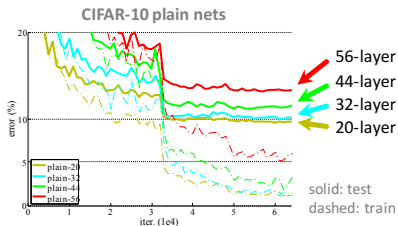
plain net



ResNet



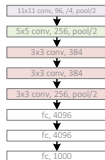
CIFAR-10 experiments



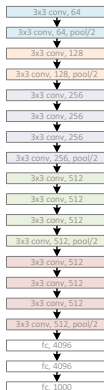
- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)

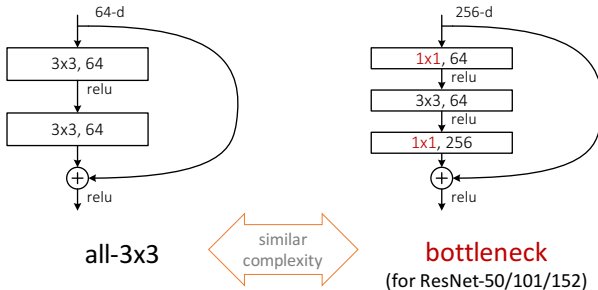


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

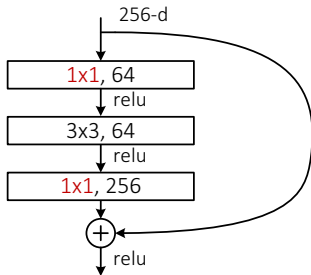
- Batch Normalization after every CONV layer.
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9).
- Learning rate: 0.1, divided by 10 when validation error plateaus.
- Mini-batch size 256.
- Weight decay of $1e-5$.
- No dropout used.

ImageNet experiments

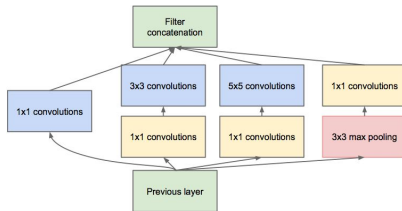
- A practical design of going deeper



Same trick as used in GoogLeNet

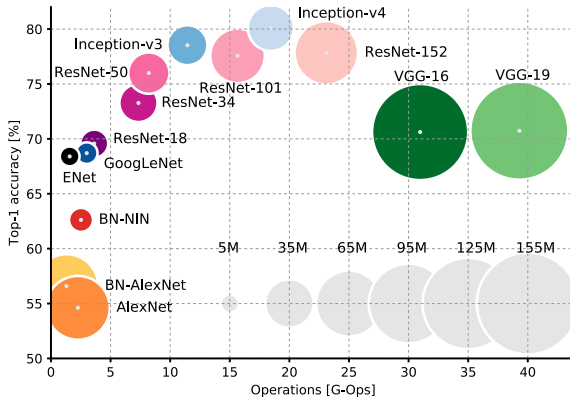


ResNet



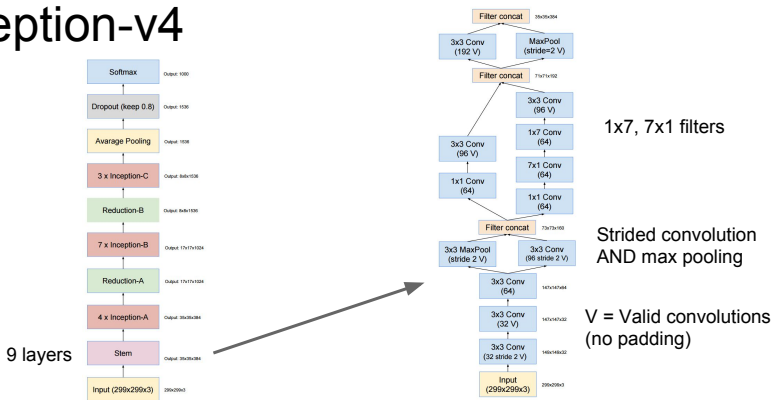
GoogLeNet

Accuracy on ImageNet Vs network size



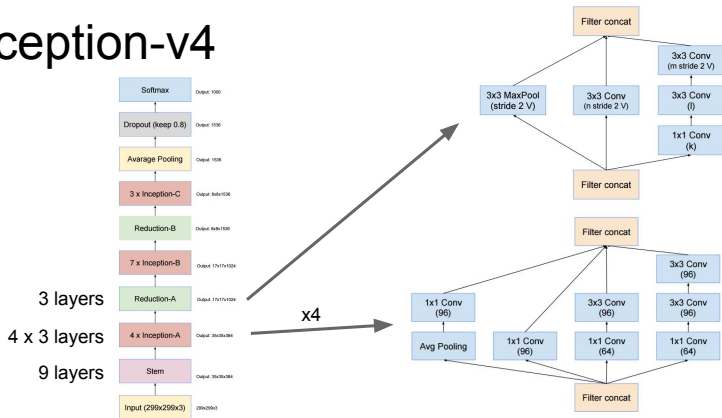
- Top-1 one-crop accuracy Vs amount of operations required for a single forward pass.
- The size of the blobs is proportional to the number of network parameters.

Inception-v4



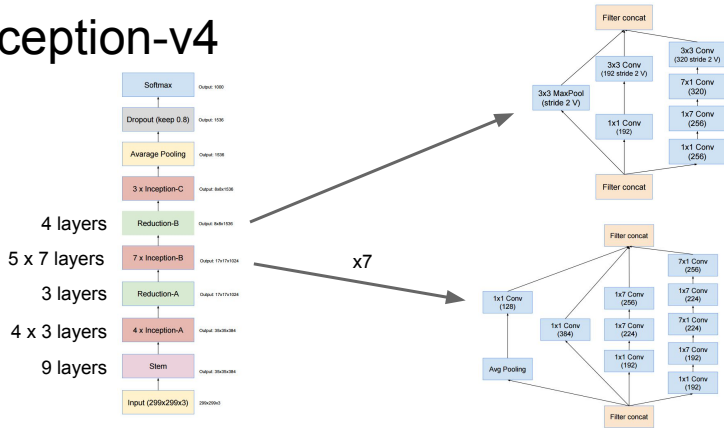
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4



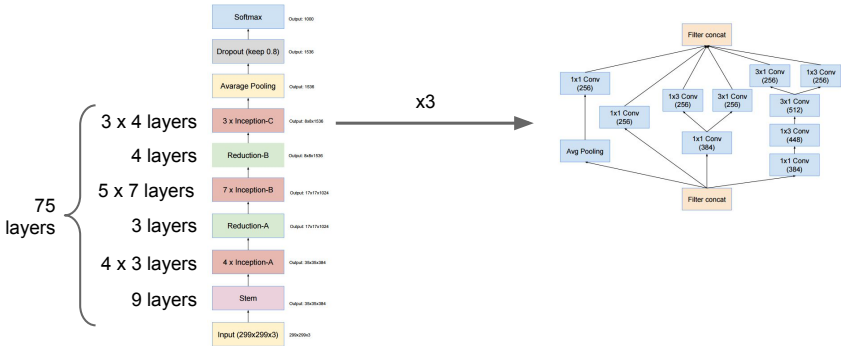
Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4



Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Inception-v4

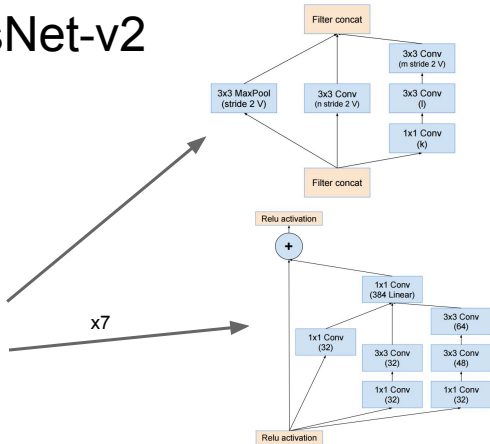
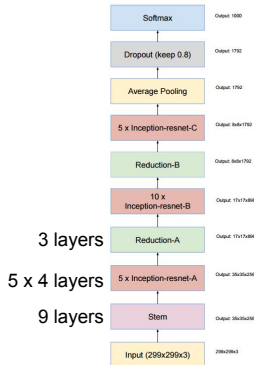


Szegedy et al, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

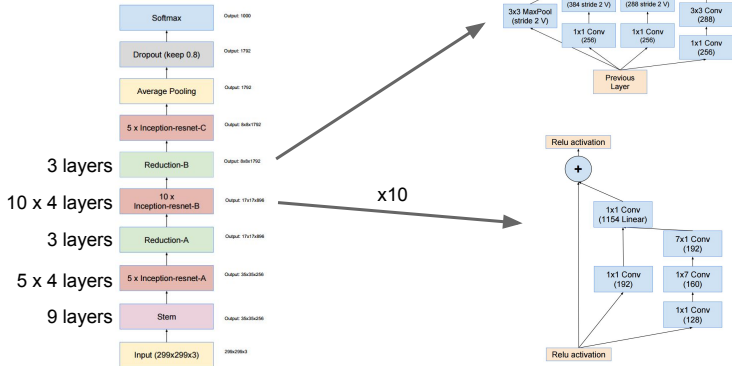
Inception-ResNet-v2



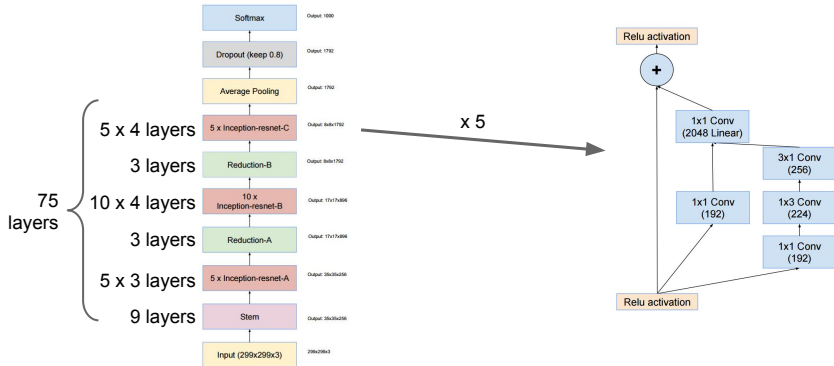
Inception-ResNet-v2



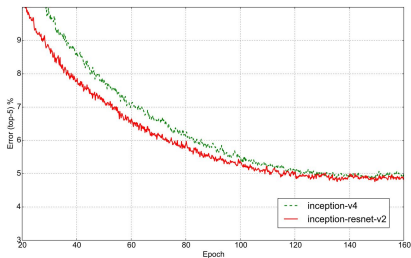
Inception-ResNet-v2



Inception-ResNet-v2



Inception-ResNet-v2



Residual and non-residual converge to similar value, but residual learns faster

- ConvNets stack CONV, ReLu, POOL layers and then FC layers.
- Trend towards smaller filters and deeper architectures.
- Trend towards fewer POOL & FC layers (just CONV)
- How can you implement a FC connected layer with a CONV layer?
- Typical architectures look like

$$[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K-SOFTMAX$$

where N is usually up to ~ 5 , M is large, $0 \leq K \leq 2$.

But recent advances (ResNet, GoogLeNet) challenge this architecture.