工業技術研究院
Industrial Technology
Research Institute

CNN General Introduction: From Lenet to Resnet

Alex Lin

Safety Sensing & Control Department
Intelligent Mobility Technology Division
Mechanical and Systems Research Laboratories
Industrial Technology Research Institute

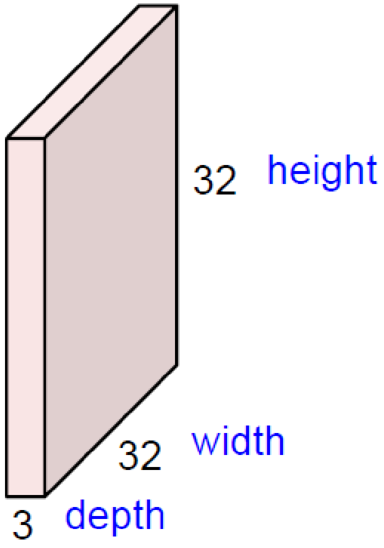
Outline

- **Basic operations of CNN**
- **CNNs**
  - **Lenet**
  - **Alexnet**
  - **VGG**
  - **GoogLeNet**
  - **Resnet**
- **CNN Speed-up**
- **Transfer Learning**
- **Other Learning Approaches**

Basic operation with color image: Convolution

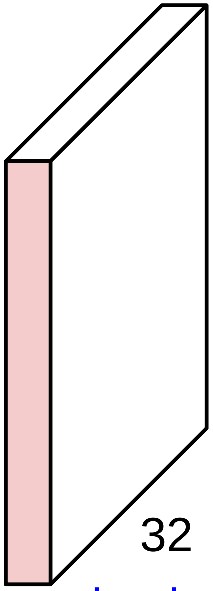
Convolution Layer

32x32x3 image



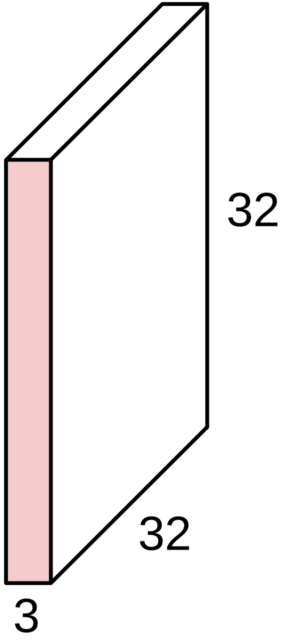
Basic operation with color image: Convolution

32x32x3 image

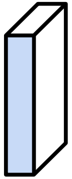


Basic operation with color image: Convolution

32x32x3 image

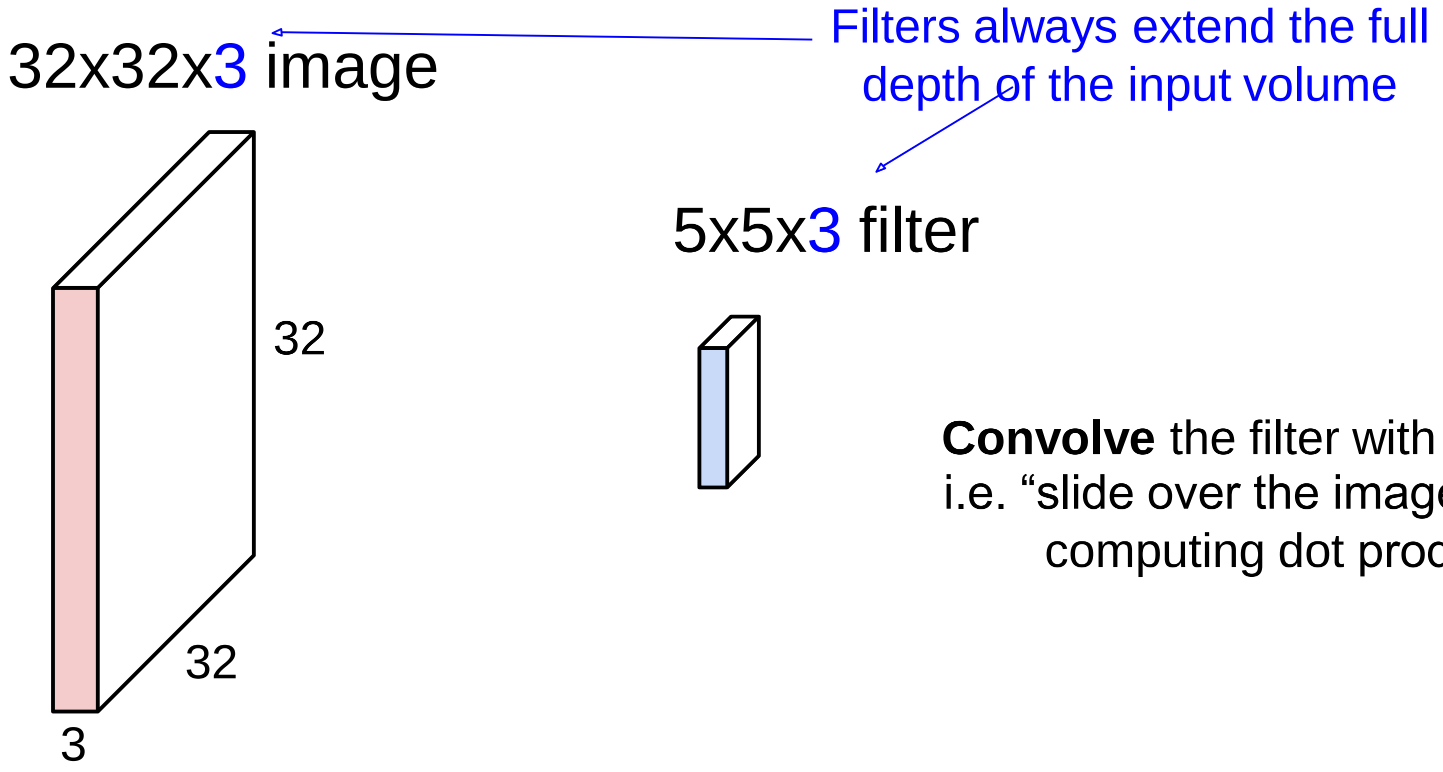


5x5x3 filter



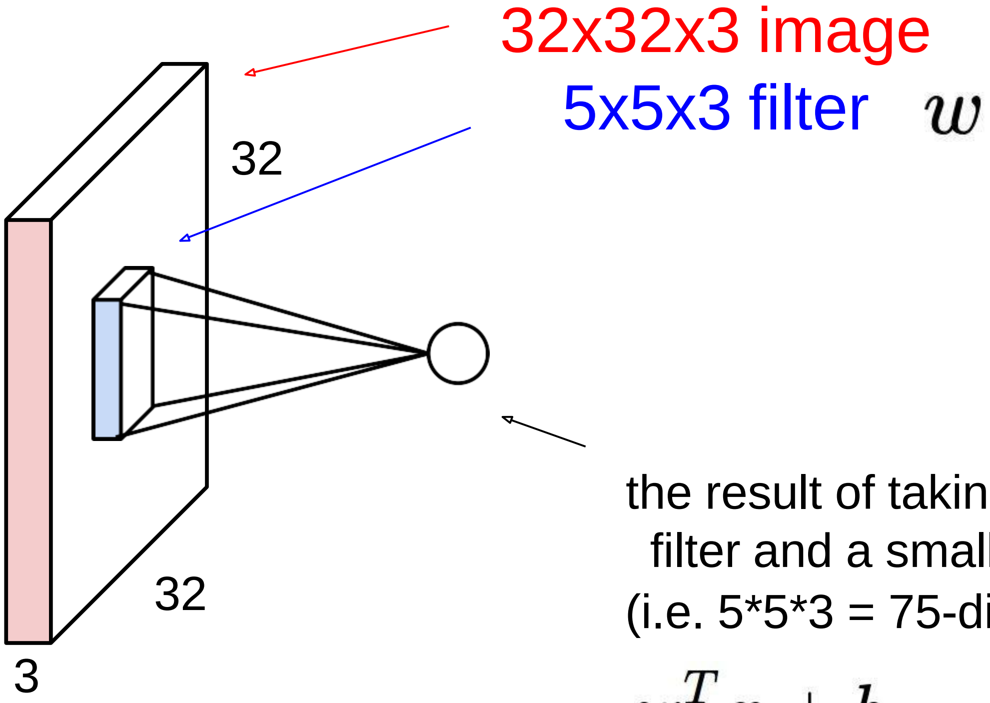
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Basic operation with color image: Convolution



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

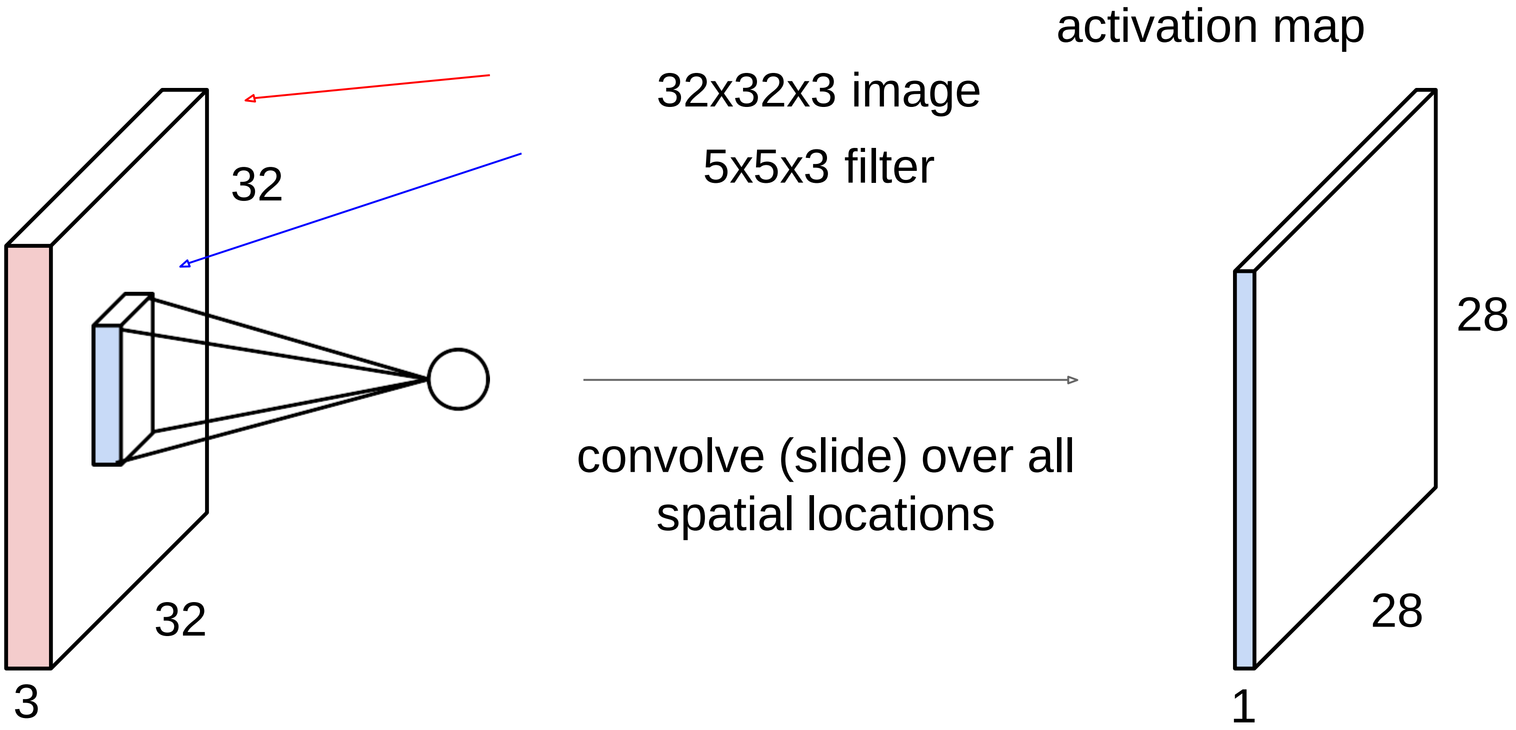
Basic operation with color image: Convolution



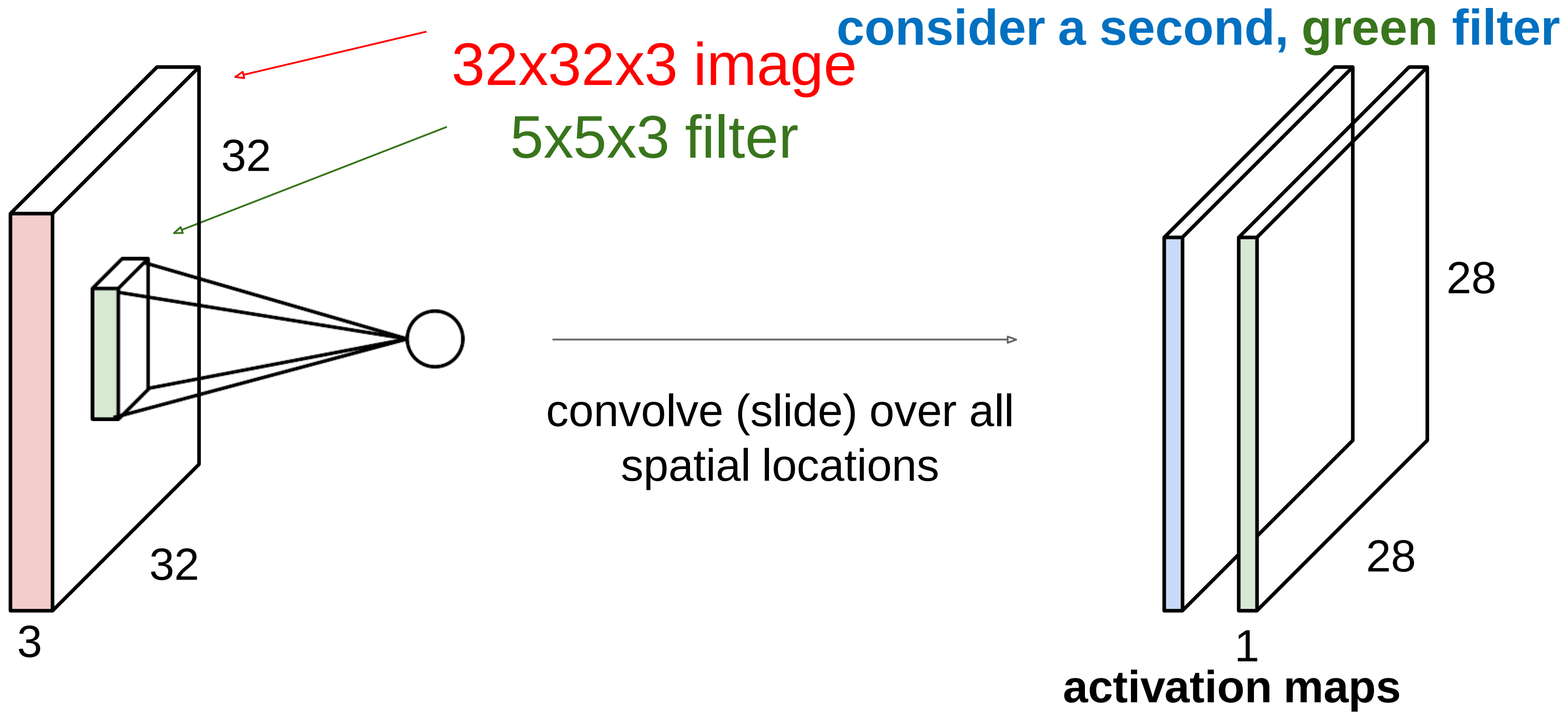
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

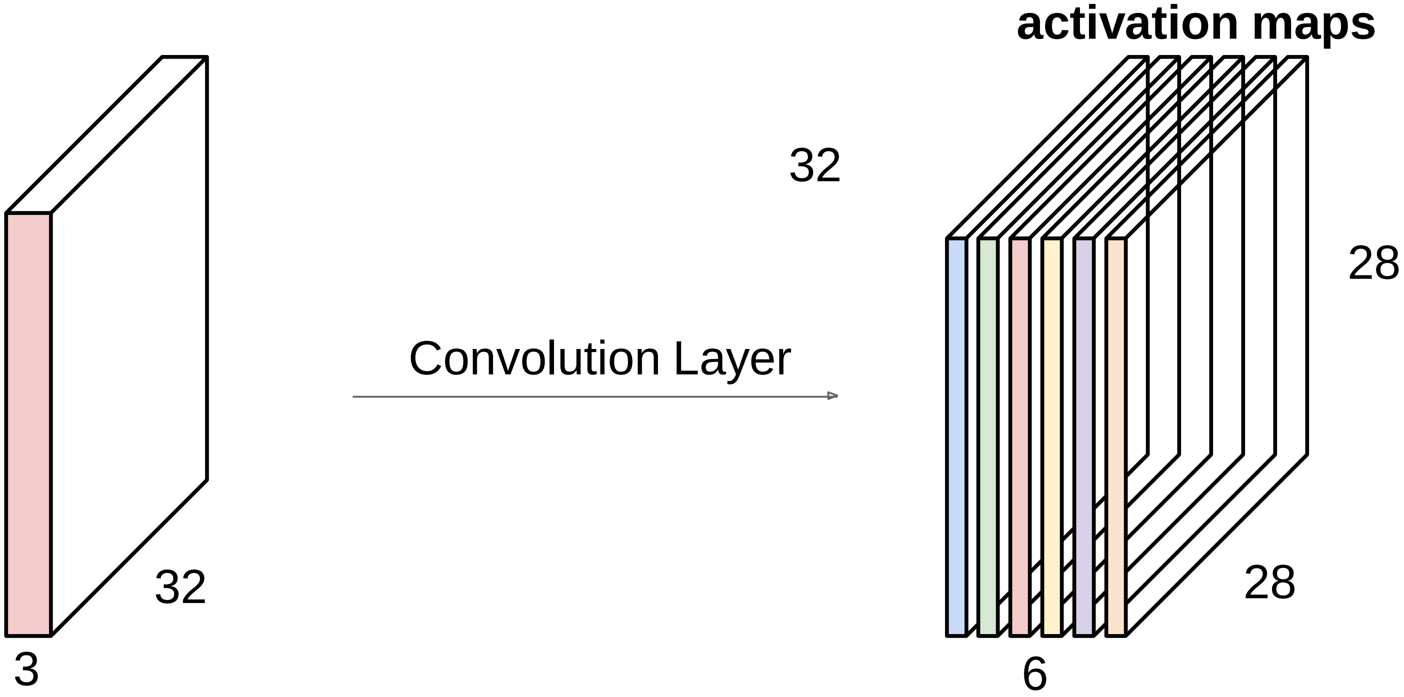
Basic operation with color image: Convolution



Basic operation with color image: Convolution



Basic operation with color image: Convolution

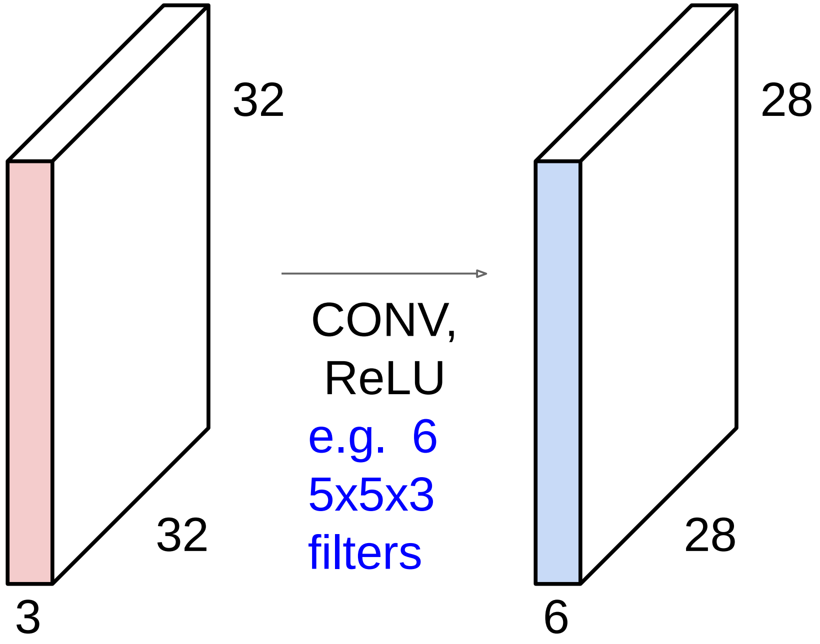


For example, if we had 6 5x5 filters, we' ll get 6 separate activation maps:

We stack these up to get a "new image" of size 28x28x6!

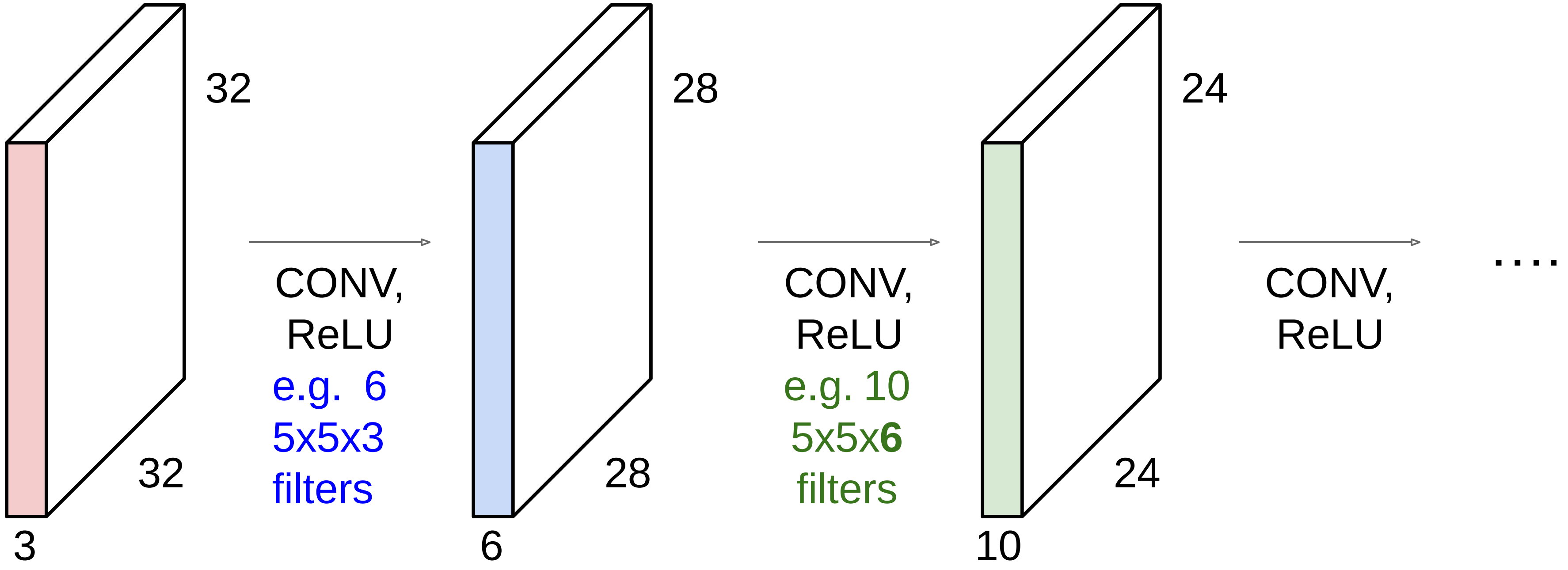
Basic operation with color image: Convolution

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

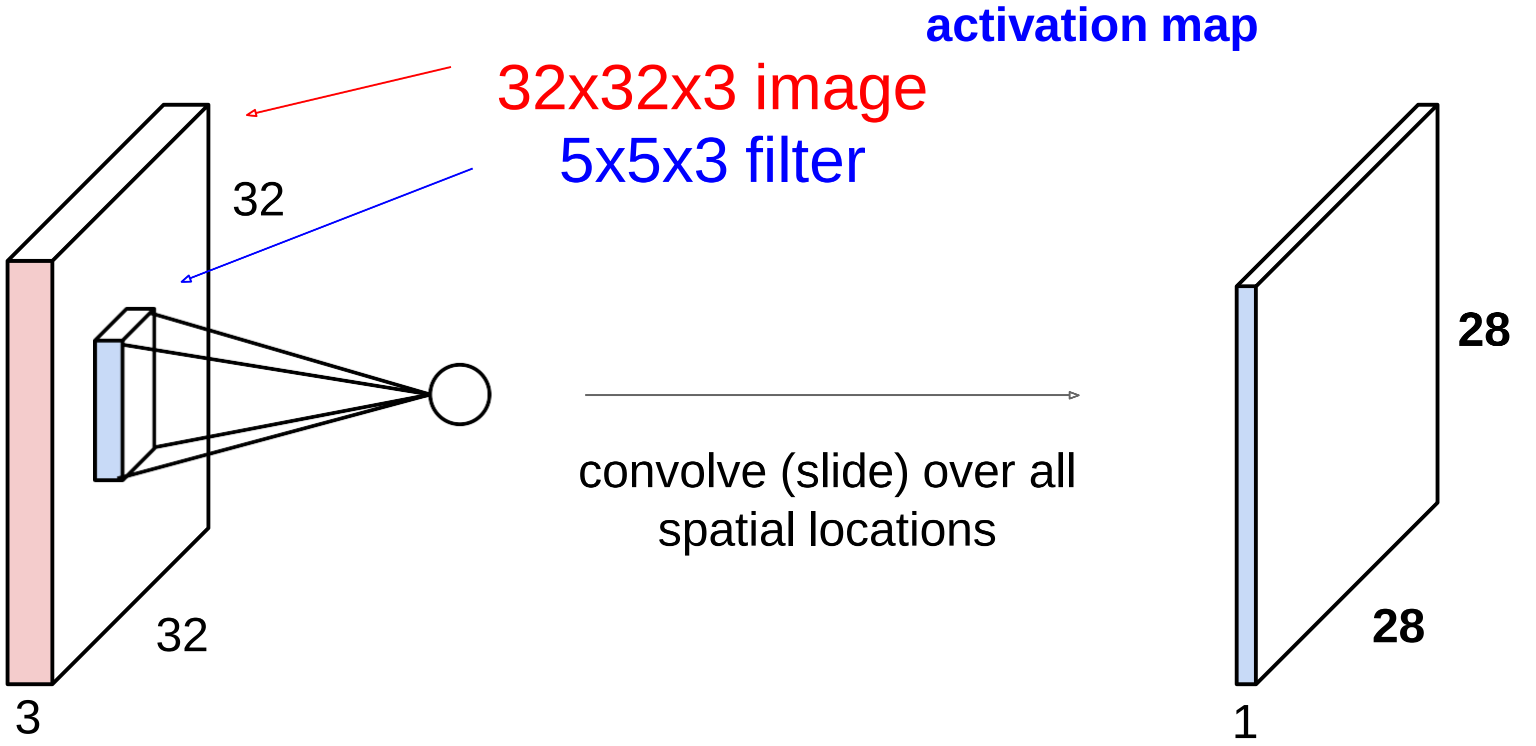


Basic operation with color image: Convolution

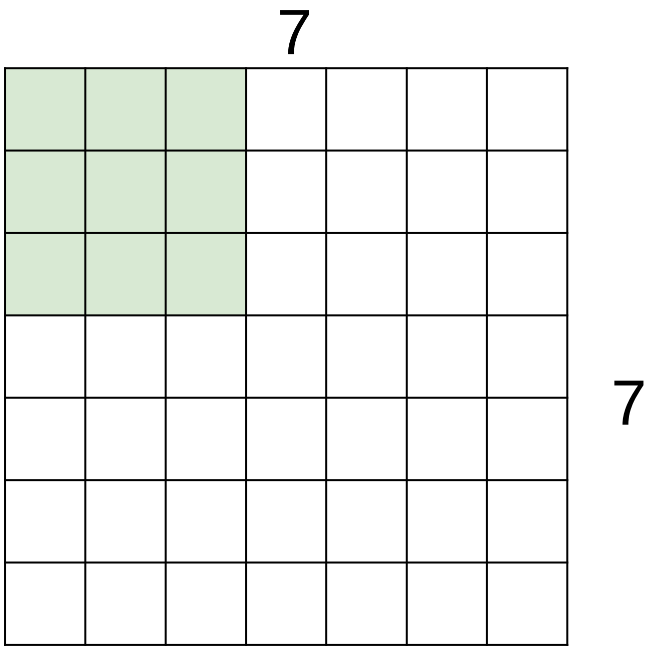
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolution: A closer look at spatial dimensions

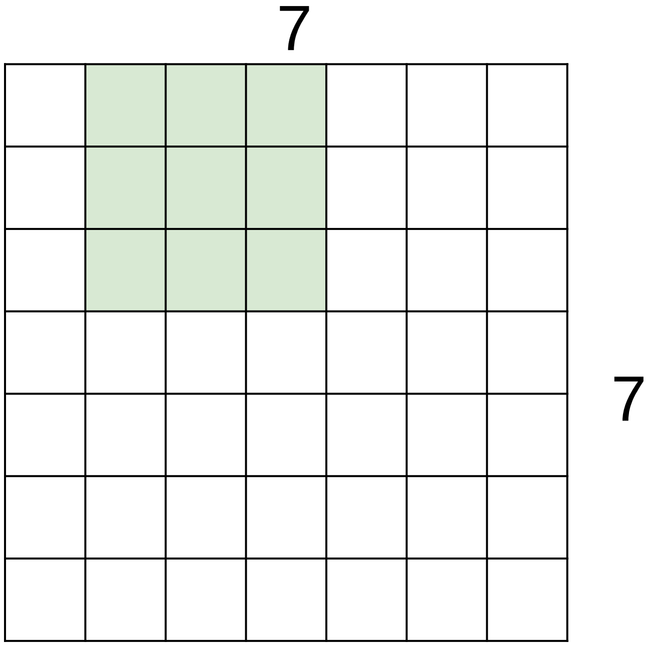


Convolution: A closer look at spatial dimensions



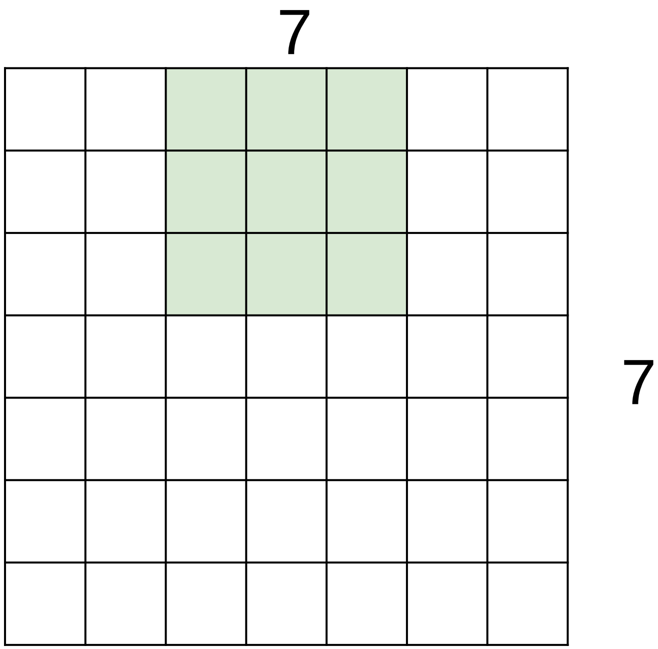
7x7 input (spatially)
assume 3x3 filter

Convolution: A closer look at spatial dimensions



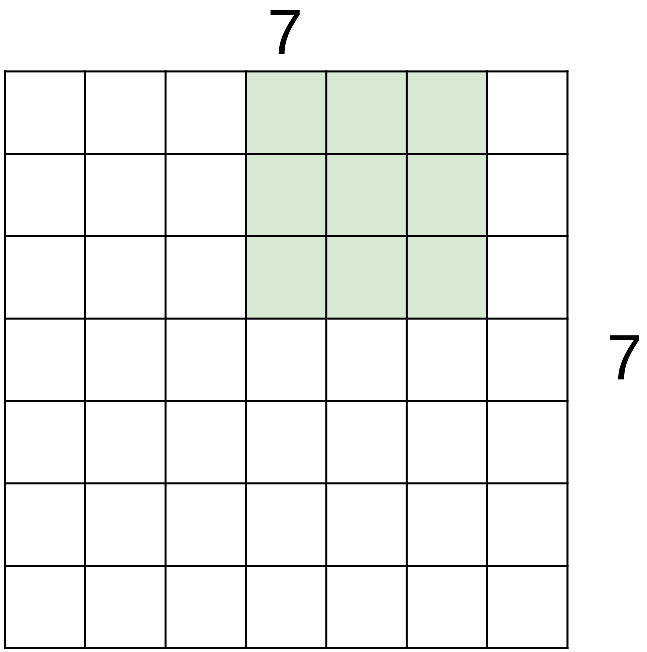
7x7 input (spatially)
assume 3x3 filter

Convolution: A closer look at spatial dimensions



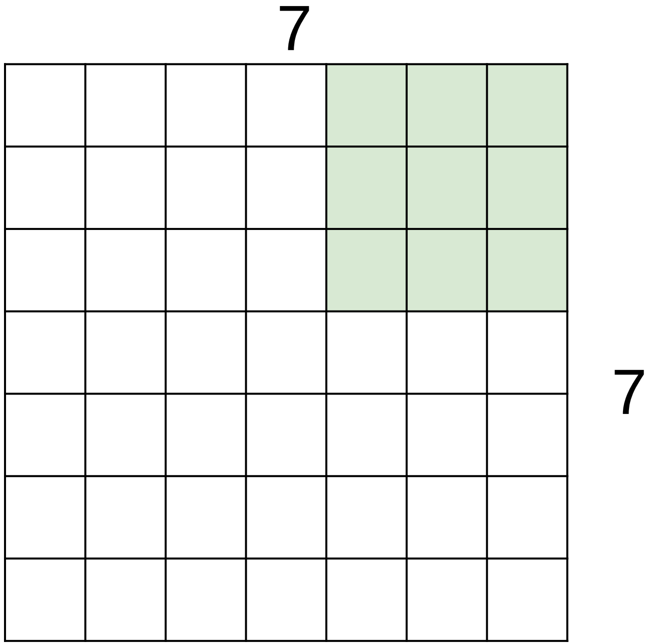
7x7 input (spatially)
assume 3x3 filter

Convolution: A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

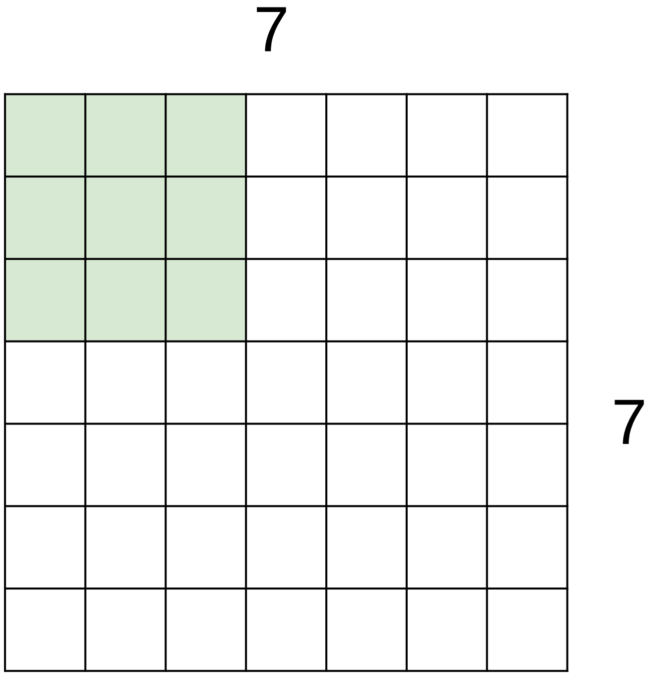
Convolution: A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

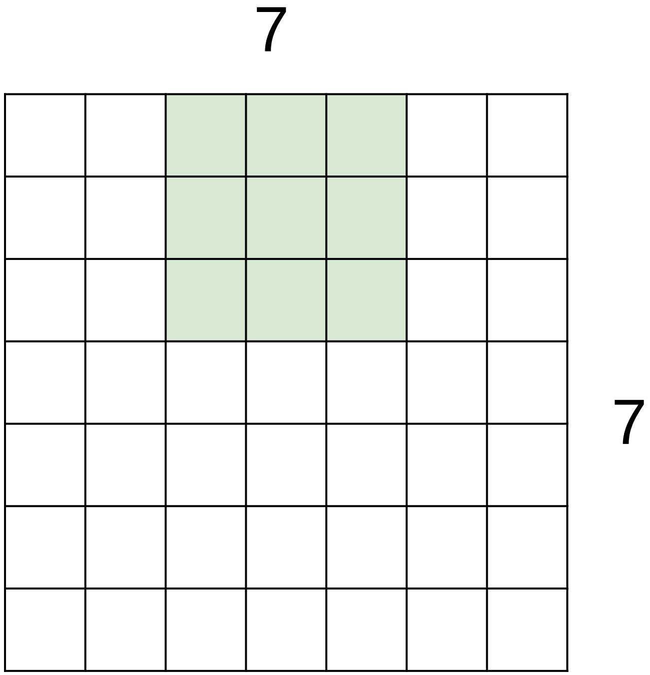
=> 5x5 output

Convolution: A closer look at spatial dimensions



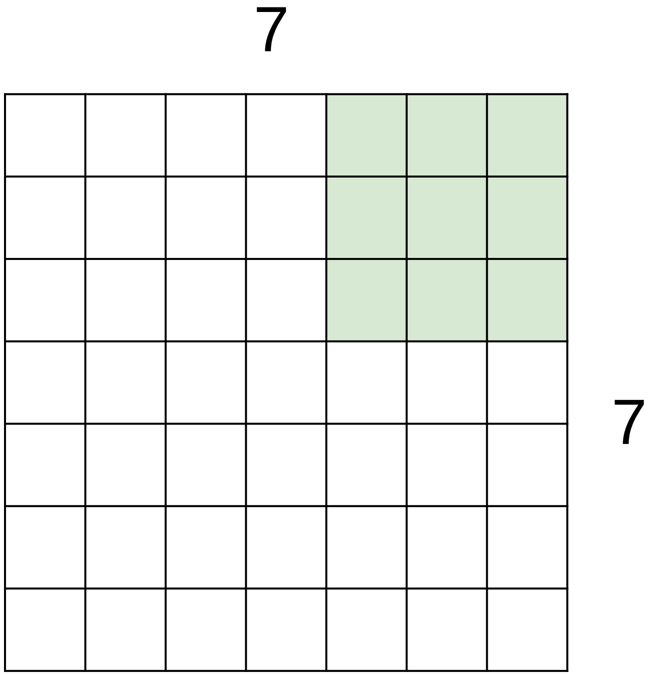
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution: A closer look at spatial dimensions



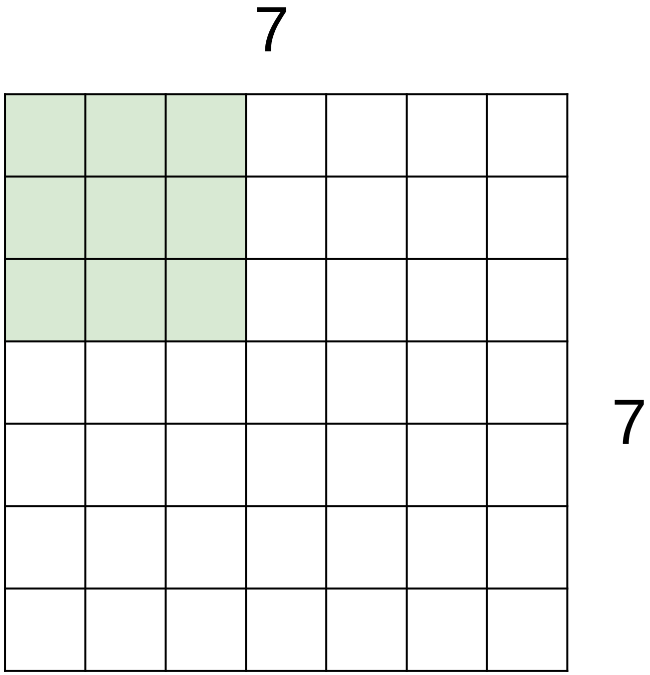
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution: A closer look at spatial dimensions



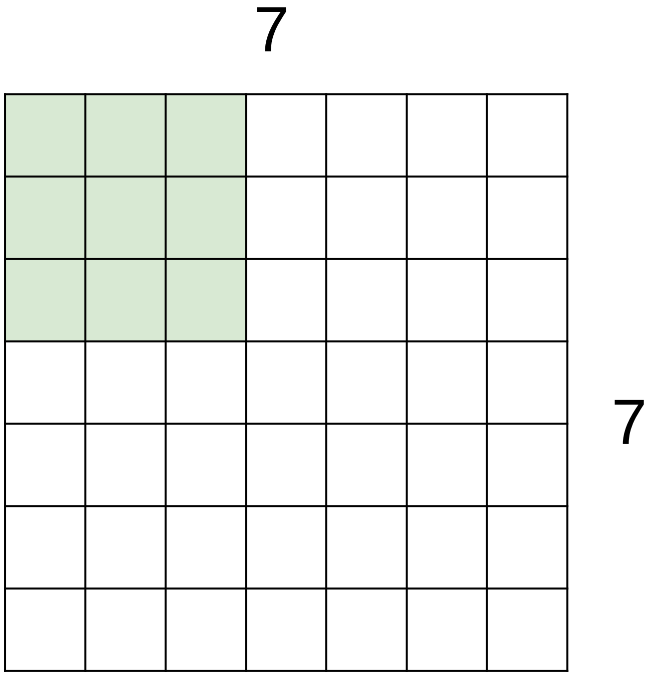
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolution: A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

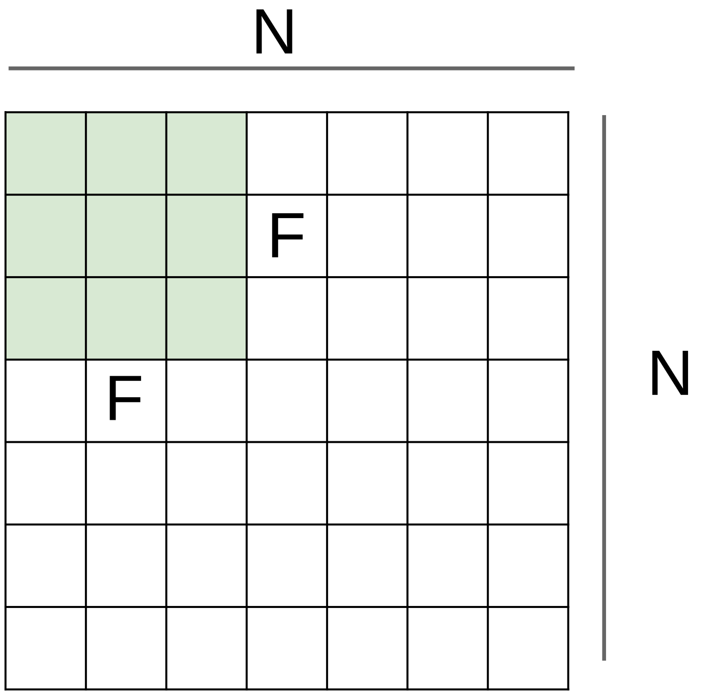
Convolution: A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolution: A closer look at spatial dimensions



Output size:

(N - F) / stride + 1

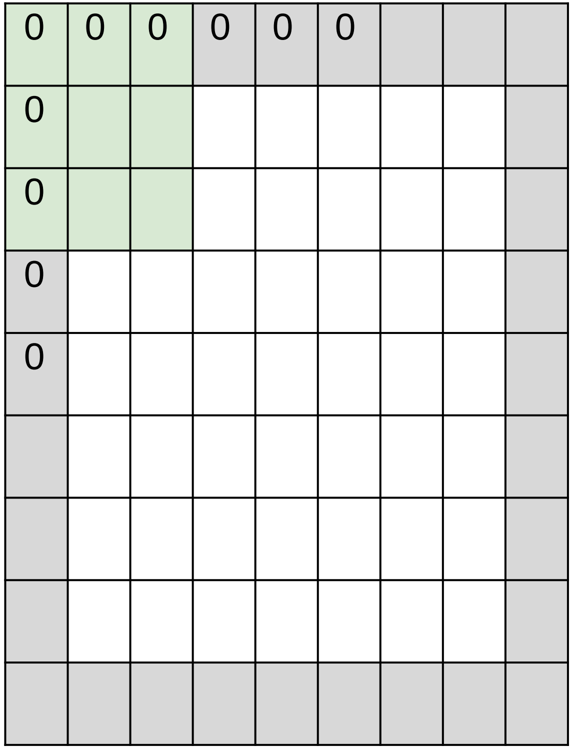
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Convolution: padding

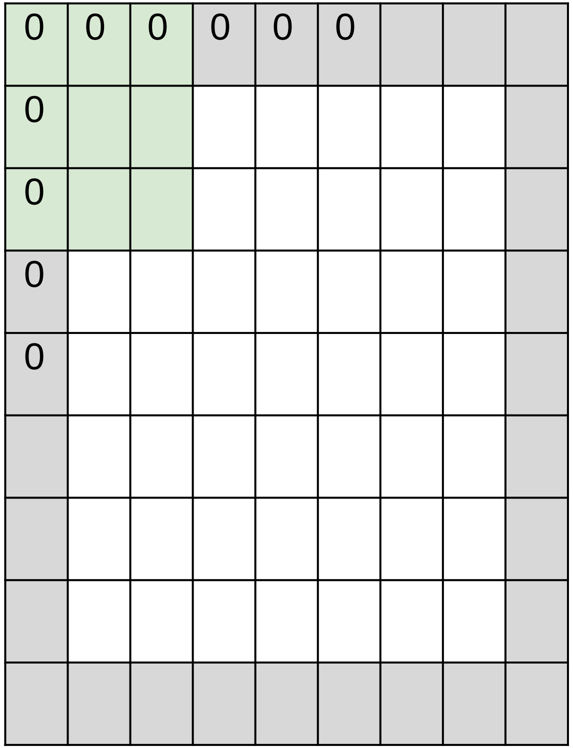


In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

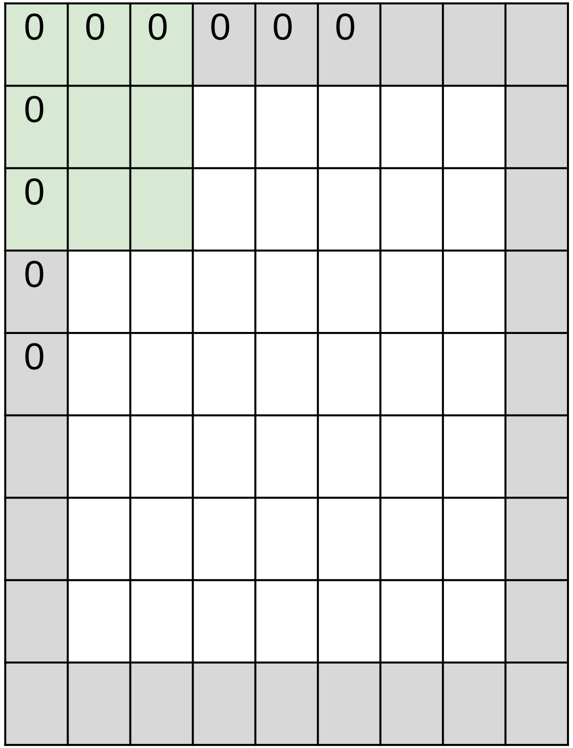
Convolution: padding



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Convolution: padding



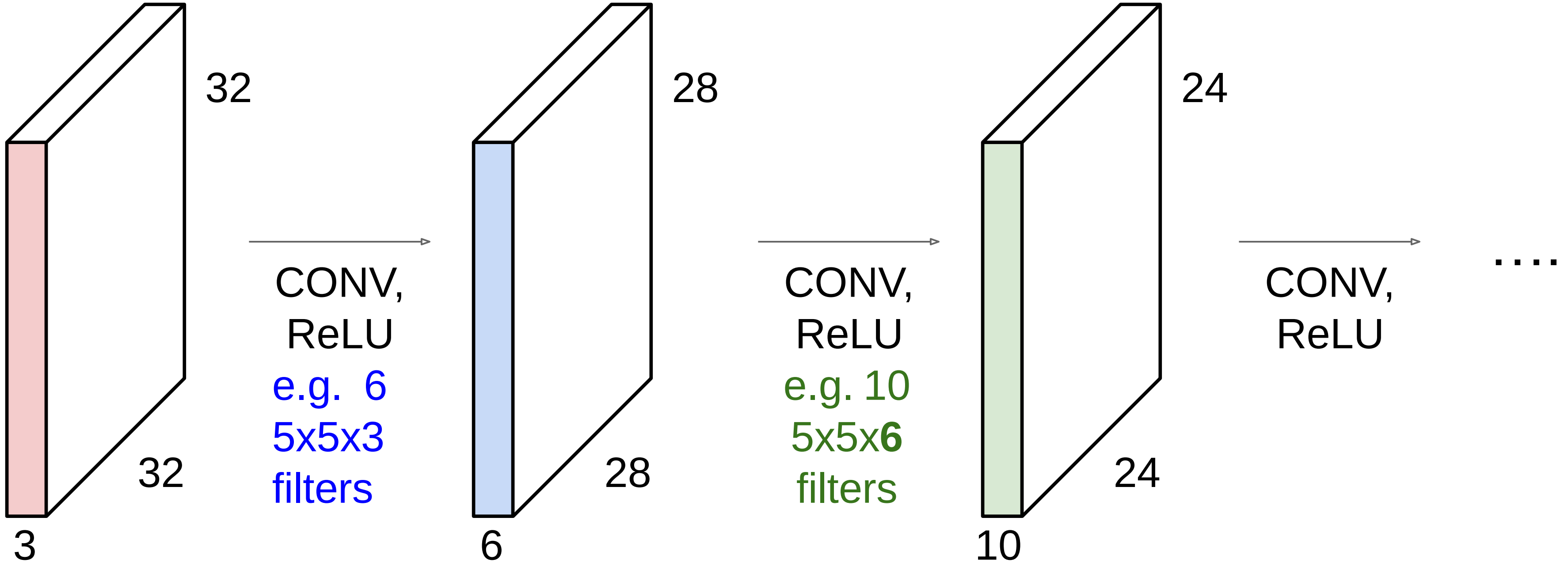
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolution: convolution with ReLU

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

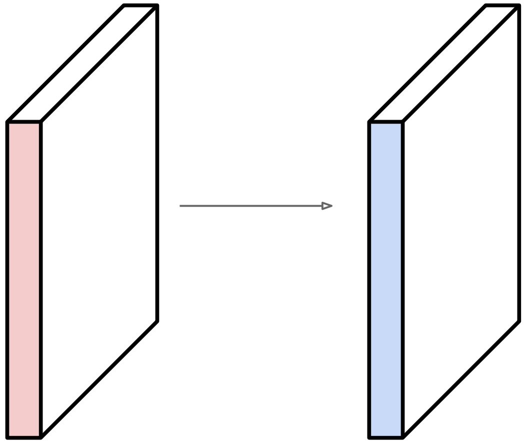


Convolution: feature map size calculation

Examples time:

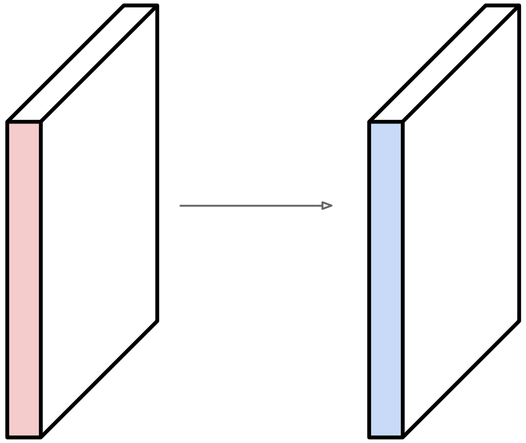
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Convolution: feature map size calculation

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

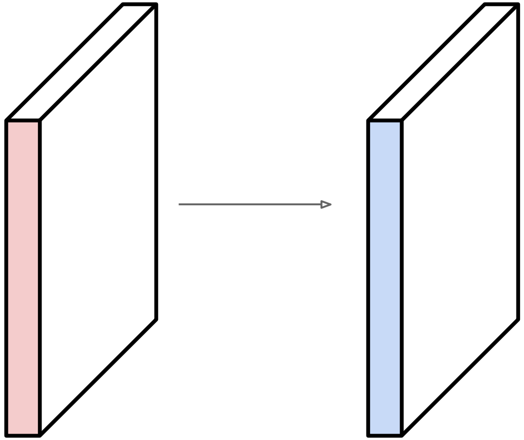
Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

Convolution: feature map size calculation

Examples time:

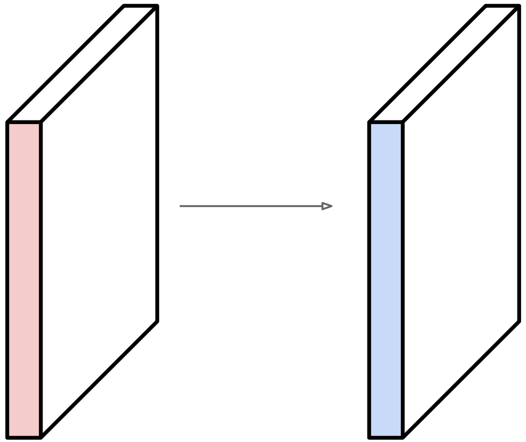


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Convolution: feature map size calculation

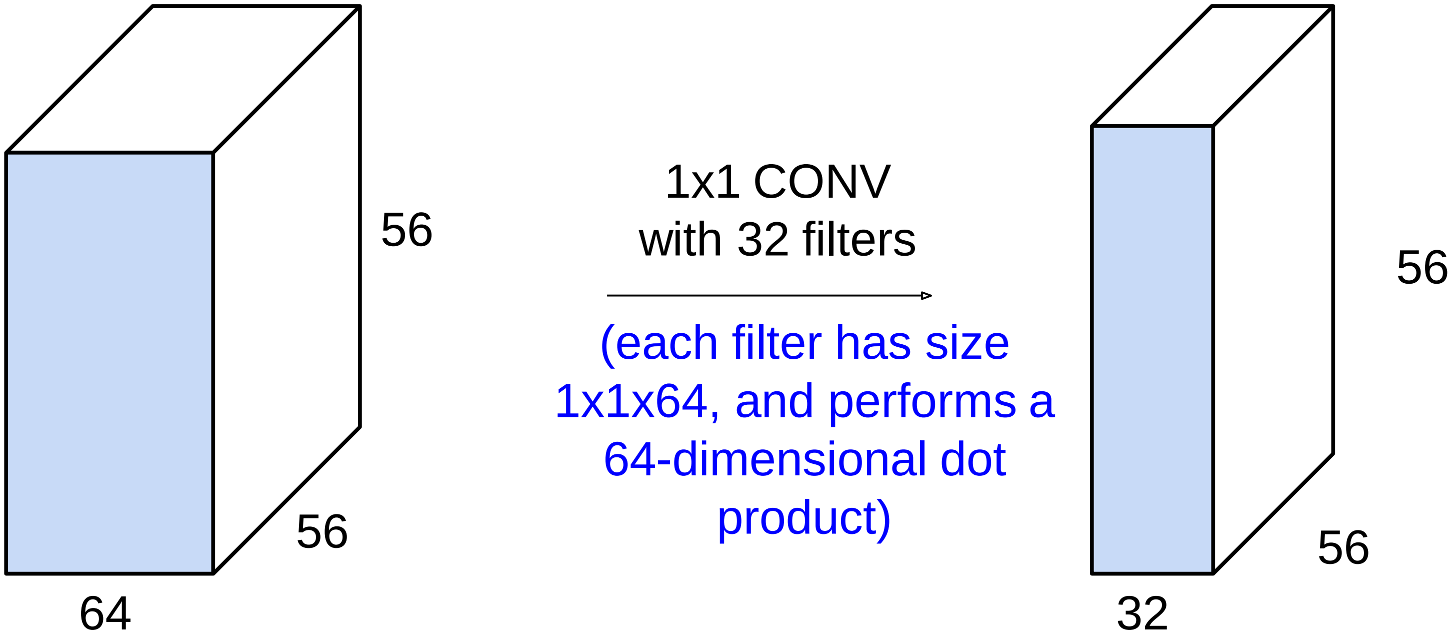
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

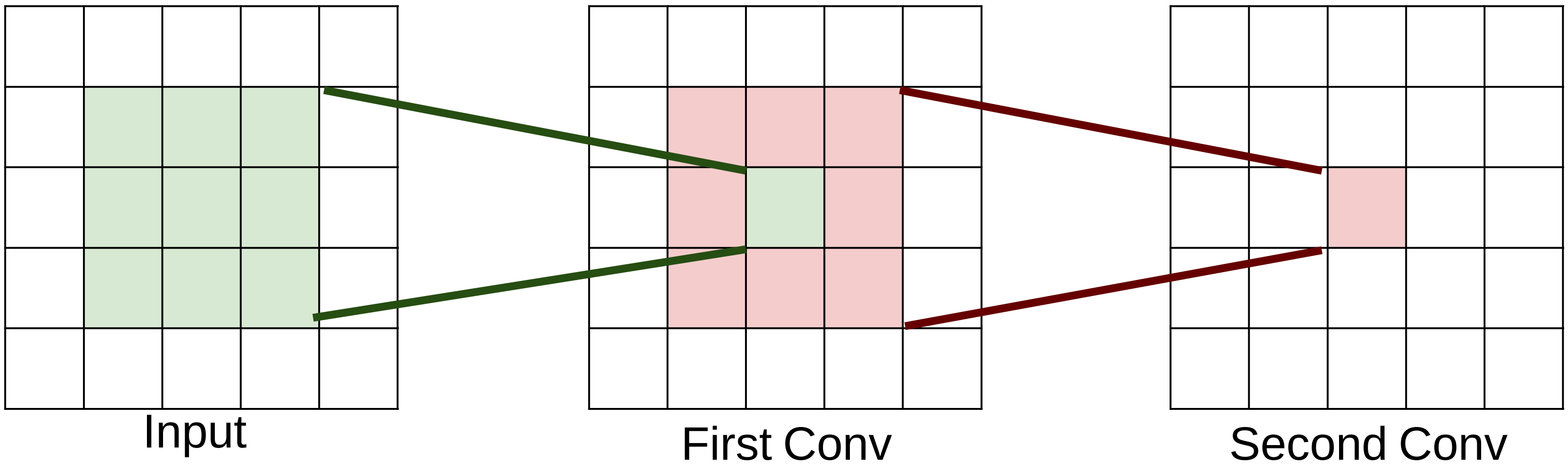
Number of parameters in this layer? each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Convolution: feature map size calculation
(btw, 1x1 convolution layers make perfect sense)



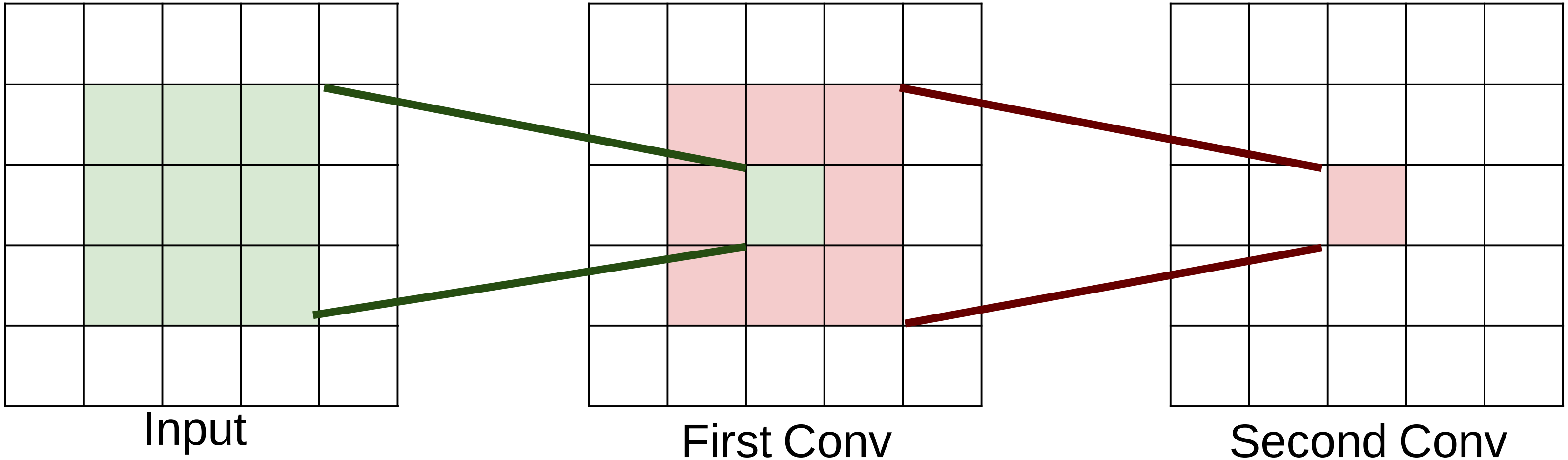
The power of small filters

Suppose we stack two 3x3 conv layers (stride 1)
Each neuron sees 3x3 region of previous activation map



The power of small filters

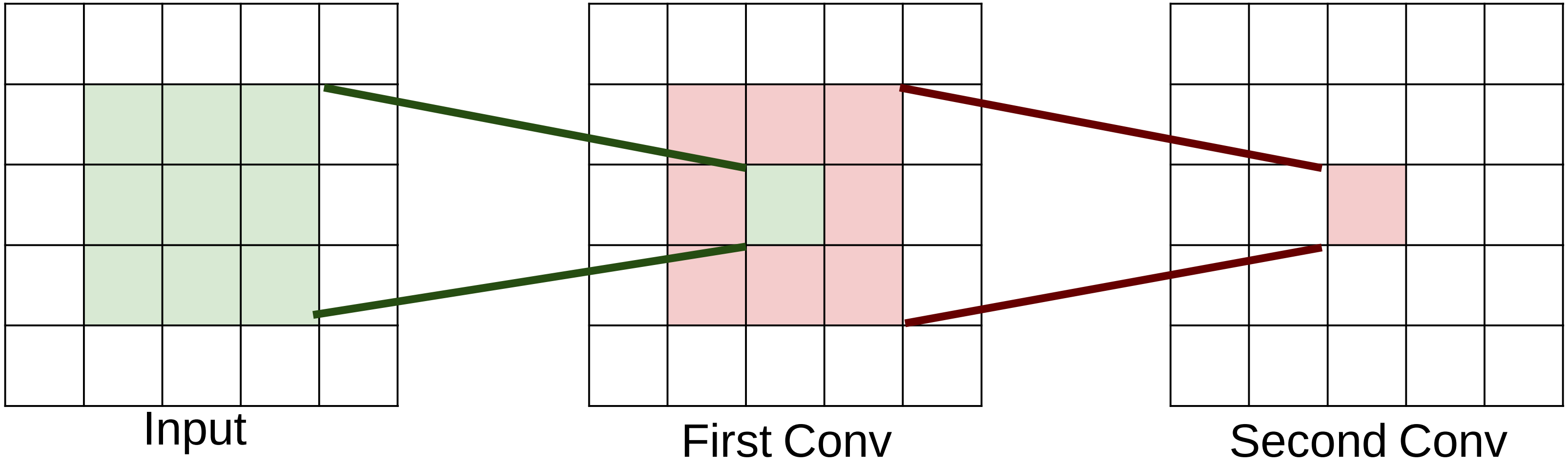
Question: How big of a region in the input does a neuron on the second conv layer see?



The power of small filters

Question: How big of a region in the input does a neuron on the second conv layer see?

Answer: 5 x 5



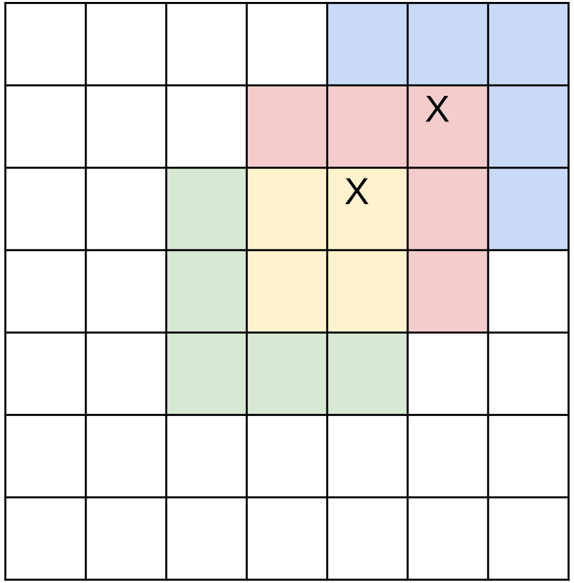
The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

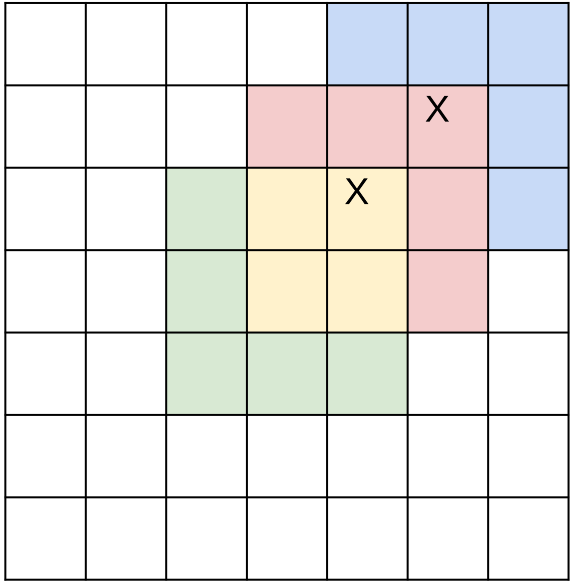
Answer: 7 x 7



The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

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)

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:

three CONV with 3 x 3 filters

Number of weights:

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:

$= C \times (7 \times 7 \times C) = \mathbf{49\ C^2}$

three CONV with 3 x 3 filters

Number of weights:

$= 3 \times C \times (3 \times 3 \times C) = \mathbf{27\ C^2}$

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:

$= C \times (7 \times 7 \times C) = \mathbf{49\ C^2}$

three CONV with 3 x 3 filters

Number of weights:

$= 3 \times C \times (3 \times 3 \times C) = \mathbf{27\ C^2}$

Fewer parameters, more nonlinearity = GOOD

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:

$= C \times (7 \times 7 \times C) = 49\ C^2$

Number of multiply-adds:

three CONV with 3 x 3 filters

Number of weights:

$= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$

Number of multiply-adds:

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:

$= C \times (7 \times 7 \times C) = 49\ C^2$

Number of multiply-adds:

$= (H \times W \times C) \times (7 \times 7 \times C)$

$= \mathbf{49\ HWC^2}$

three CONV with 3 x 3 filters

Number of weights:

$= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$

Number of multiply-adds:

$= 3 \times (H \times W \times C) \times (3 \times 3 \times C)$

$= \mathbf{27\ HWC^2}$

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:

$= C \times (7 \times 7 \times C) = 49\ C^2$

Number of multiply-adds:

$= \mathbf{49\ HWC^2}$

three CONV with 3 x 3 filters

Number of weights:

$= 3 \times C \times (3 \times 3 \times C) = 27\ C^2$

Number of multiply-adds:

$= \mathbf{27\ HWC^2}$

Less compute, more nonlinearity = GOOD

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

1. “bottleneck” 1 x 1 conv
   to reduce dimension

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

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)

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

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

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

1. “bottleneck” 1 x 1 conv to reduce dimension

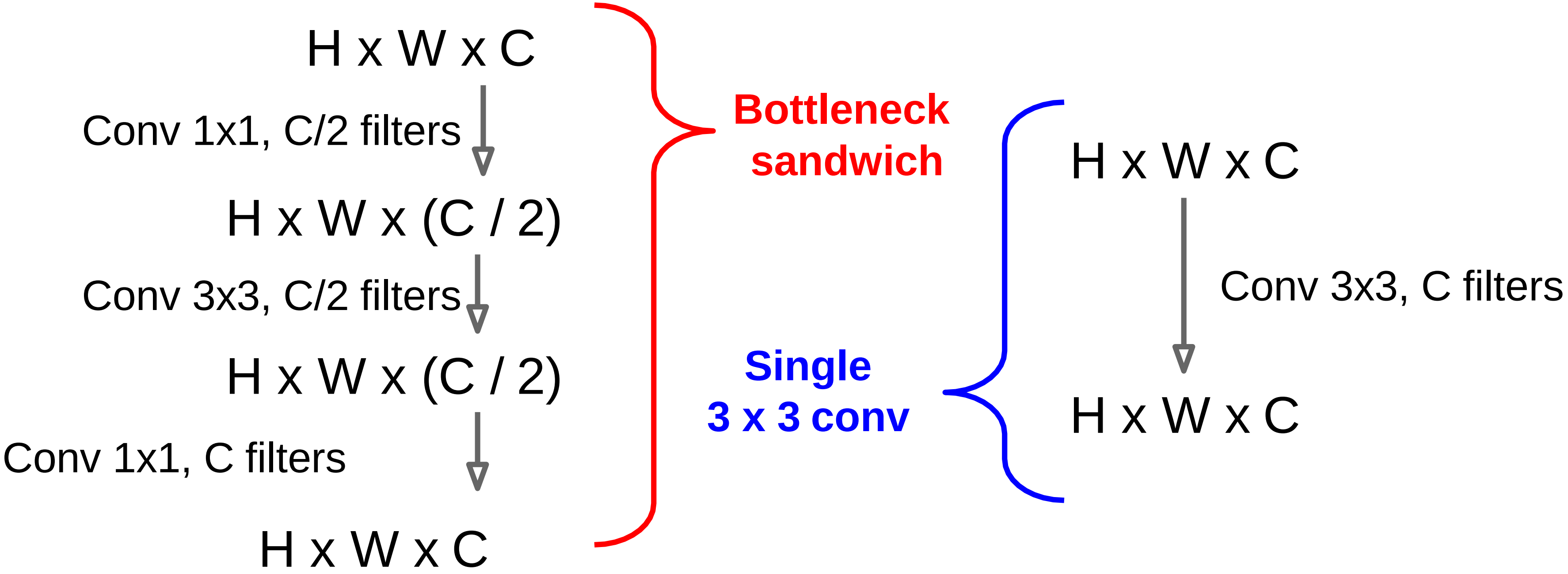
2. 3 x 3 conv at reduced dimension

3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, “Network in Network”, GoogLeNet, ResNet]

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

More nonlinearity,
fewer params,
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

H x W x C

3.25 C^2 parameters

H x W x C

Conv 3x3, C filters

H x W x C

9 C^2 parameters

The power of small filters

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

The power of small filters

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

$$H \times W \times C \xrightarrow{\text{Conv 1x3, C filters}} H \times W \times C \xrightarrow{\text{Conv 3x1, C filters}} H \times W \times C$$

The power of small filters

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

More nonlinearity,
fewer params,
less compute!

H x W x C

Conv 1x3, C filters

H x W x C

Conv 3x1, C filters

H x W x C

6 C^2 parameters

9 C^2 parameters

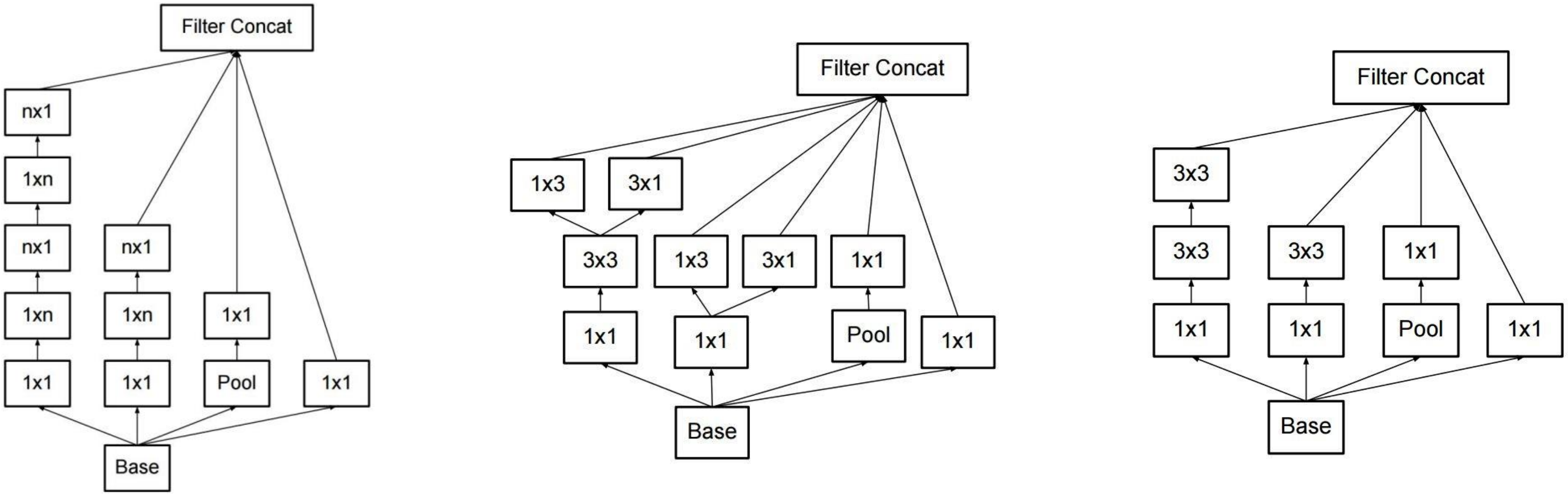
H x W x C

Conv 3x3, C filters

H x W x C

The power of small filters

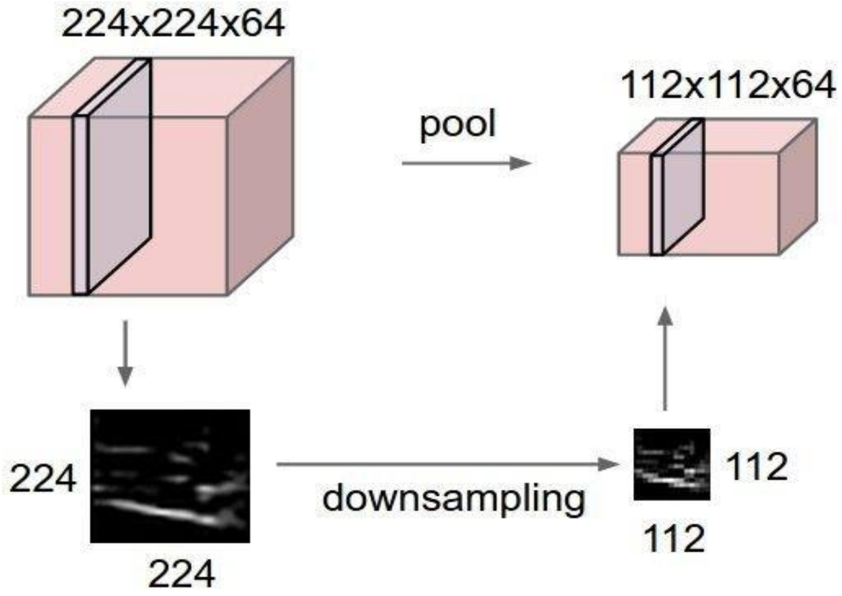
Latest version of GoogLeNet incorporates all these ideas



Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”

Pooling

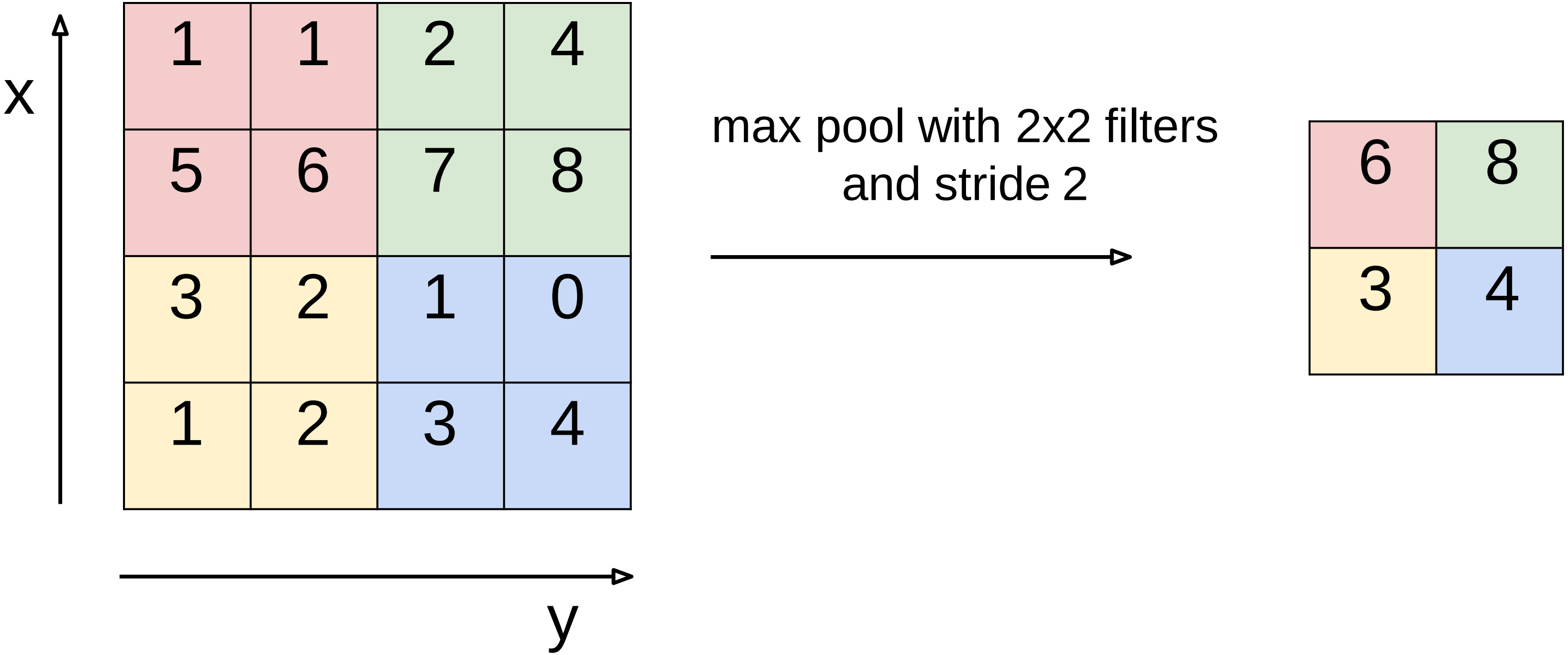
- makes the representations smaller and more manageable
    - operates over each activation map independently:



Pooling

MAX POOLING

Single depth slice



Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

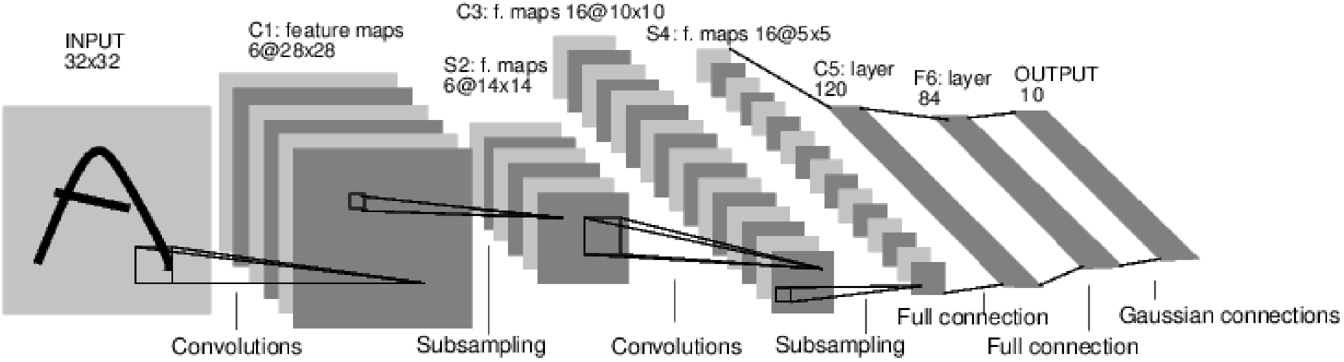
Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

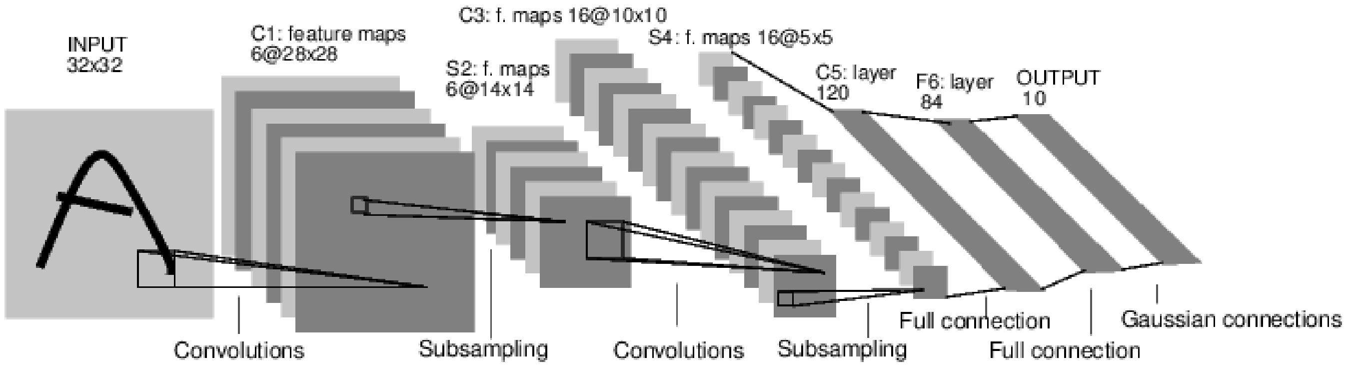
F = 2, S = 2
F = 3, S = 2

Lenet-5: C1



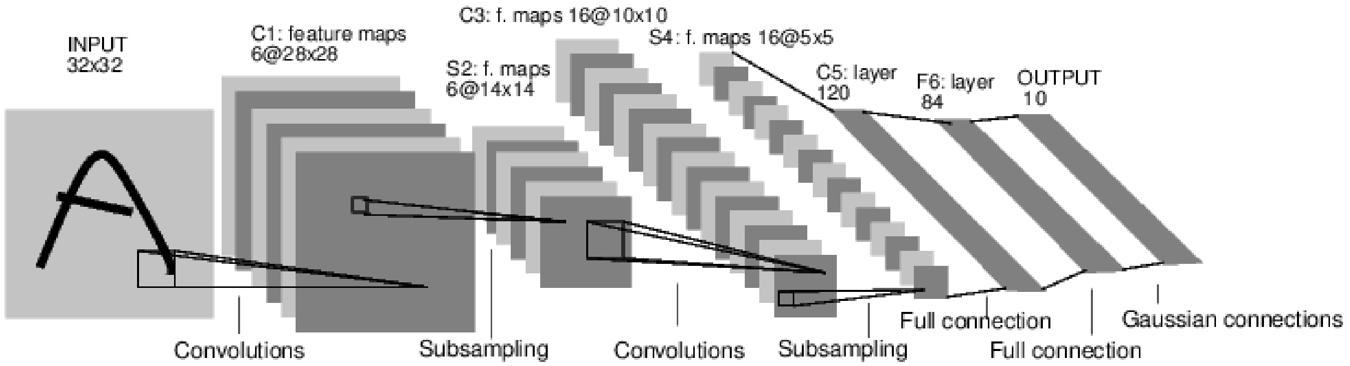
C1 layer has 6 feature maps (28x28), a 5x5 receptive filed resulting in (5*5+1)*6 = 156 learnable parameters which are from 28*28*(5*5*+1)*6 = 122,304 connections.

Lenet-5: S2



- S1 layer has 6 feature maps (14x14), a 2x2 receptive filed resulting in (1+1)*6 = 12 learnable parameters which are from 14*14*(2*2+1)*6 = 5,880 connections.

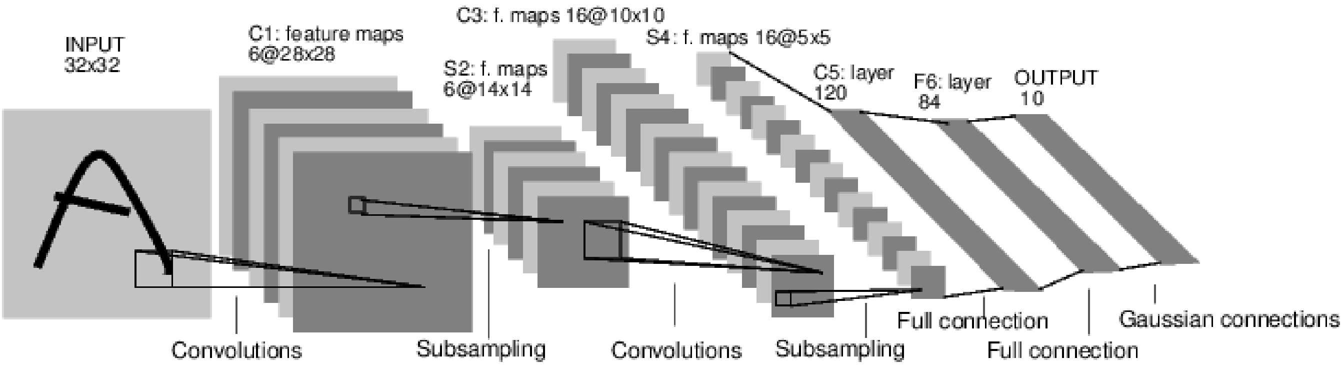
Lenet-5: C3



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

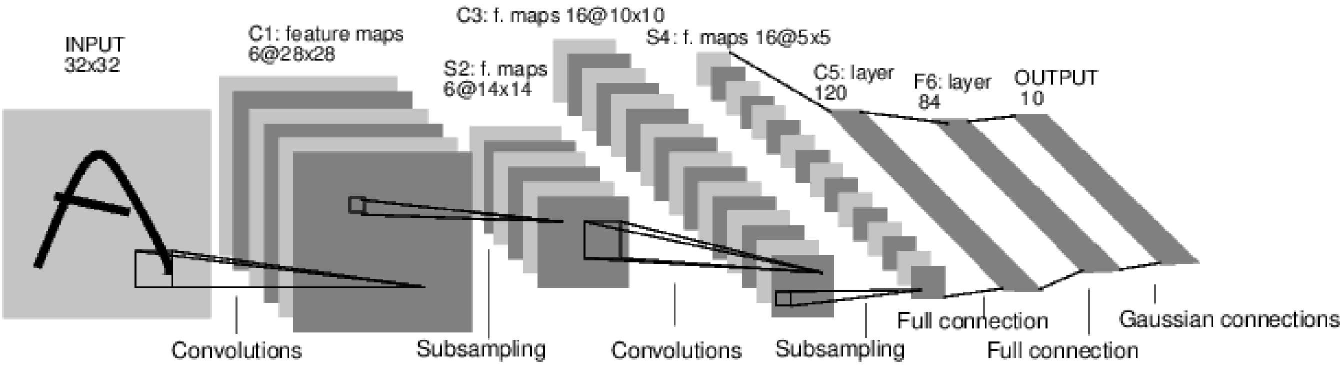
- Parameters=(5*5)*3*6+ (5*5)*4*9+ (5*5)*6*1+16=1516
- Connections: 1516*100=151600

Lenet-5: S4



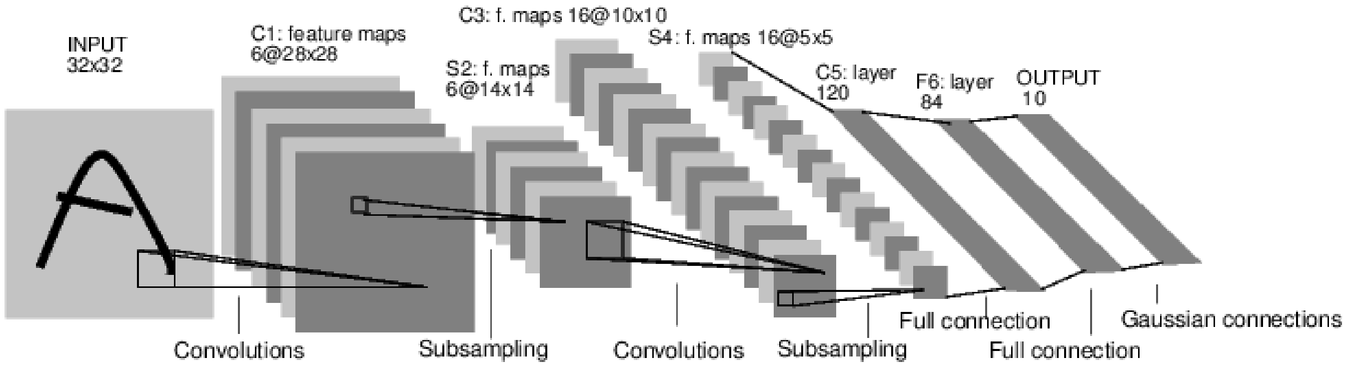
- S4: Subsampling layer with 16 feature maps of size 5x5
- Each unit in S4 is connected to the corresponding 2x2 receptive field at C3
- Layer S4: 16*2=32 trainable parameters.
- Connections: 5*5*(2*2+1)*16=2000

Lenet-5: C5



- C5: Convolutional layer with 120 feature maps of size 1x1
- Each unit in C5 is connected to all 16 5x5 receptive fields in S4
- Layer C5: 120*(16*25+1) = 48120 trainable parameters and connections (Fully connected)

Lenet-5: F6

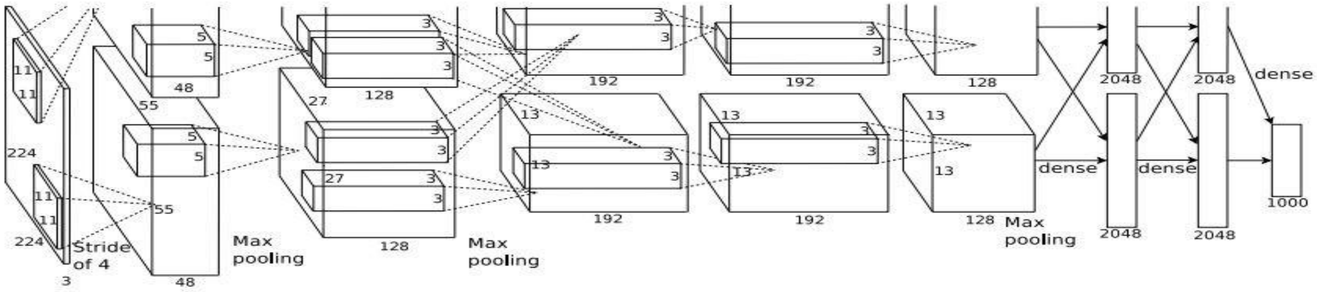


- Layer F6: 84 fully connected units. 84*(120+1)=10164 trainable parameters and connections.
- Output layer: 10RBF (One for each digit) 84=7x12, stylized image
- Weight update: Backpropagation

Lenet: website demo

Ref: http://scs.ryerson.ca/~aharley/vis/conv/

Modern CNN: AlexNet (Supervision)

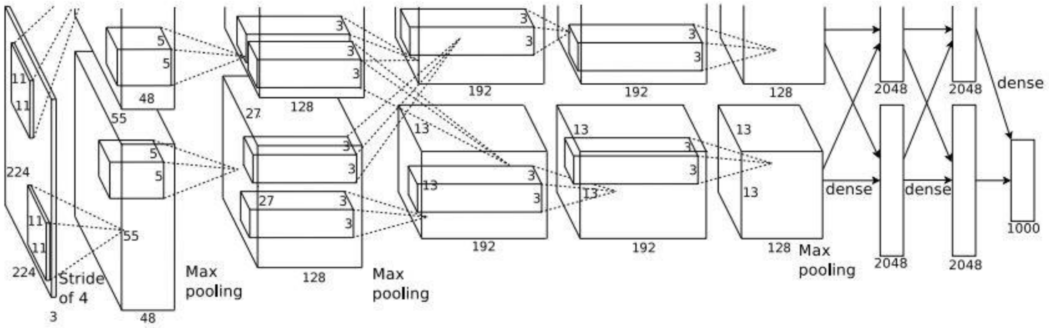


- **Similar framework to LeCun'98 but,**
- **Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)**
- **More data (10^6 vs. 10^3 images)**
- **GPU implementation (50x speedup over CPU)**
- **Trained on two GPUs for a week**
- **Better regularization for training (DropOut)**

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

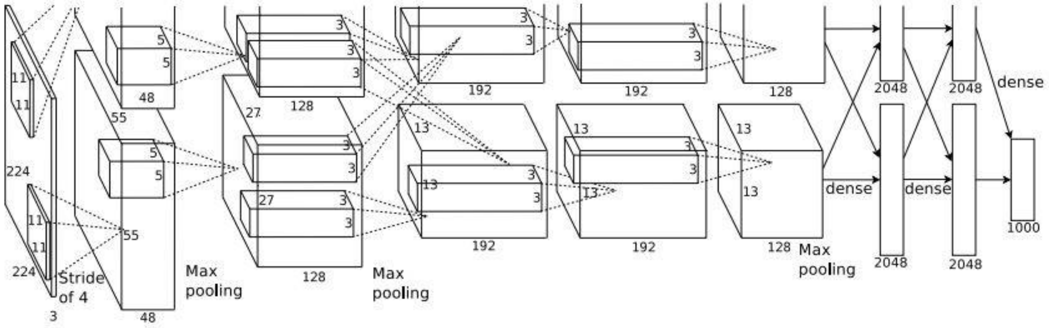
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

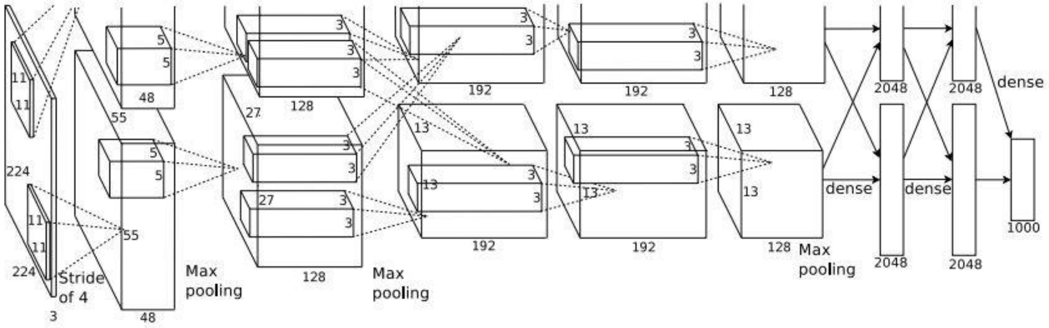
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



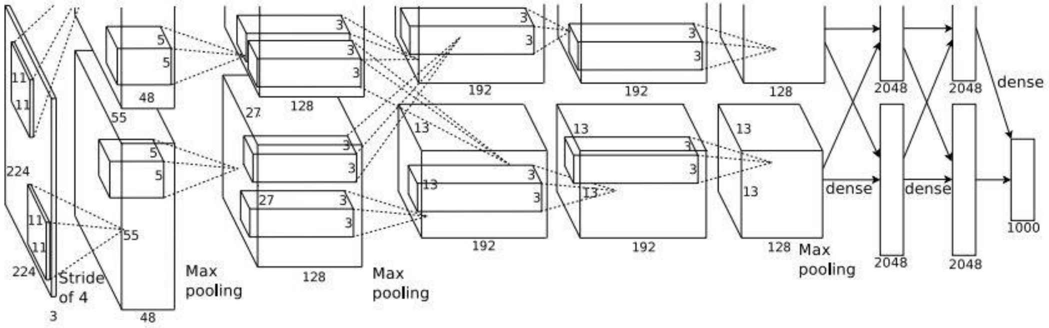
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

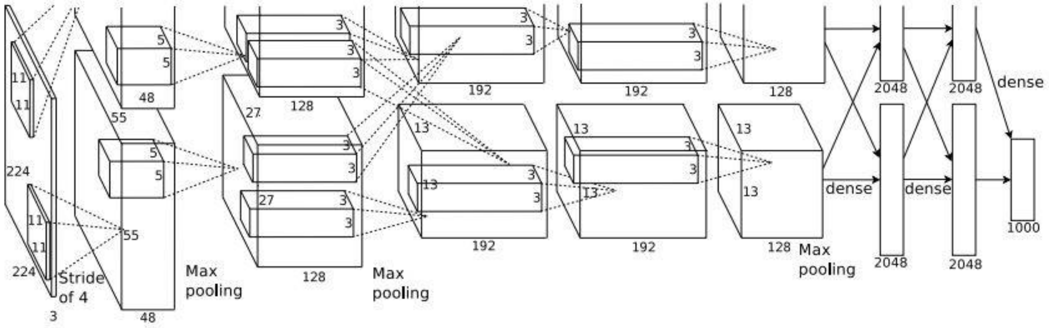
Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

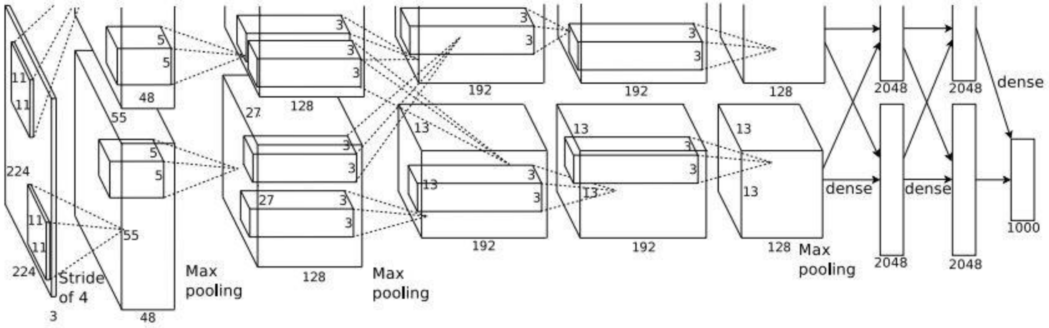
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



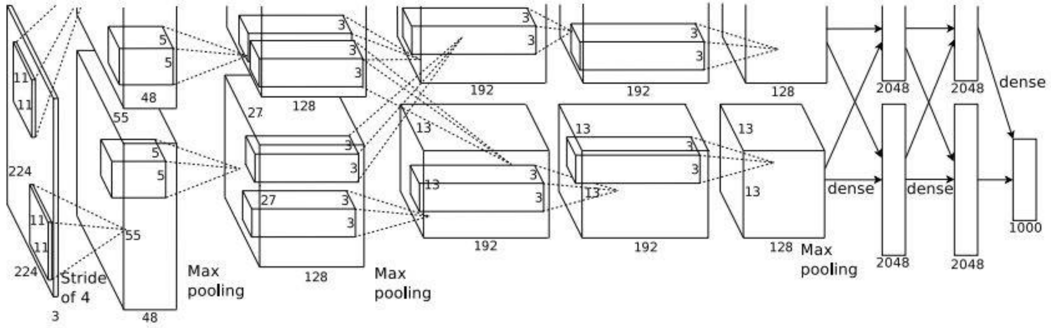
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

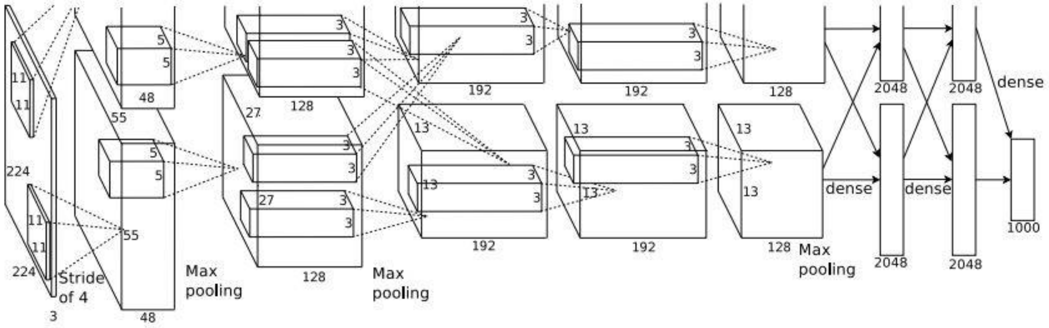


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

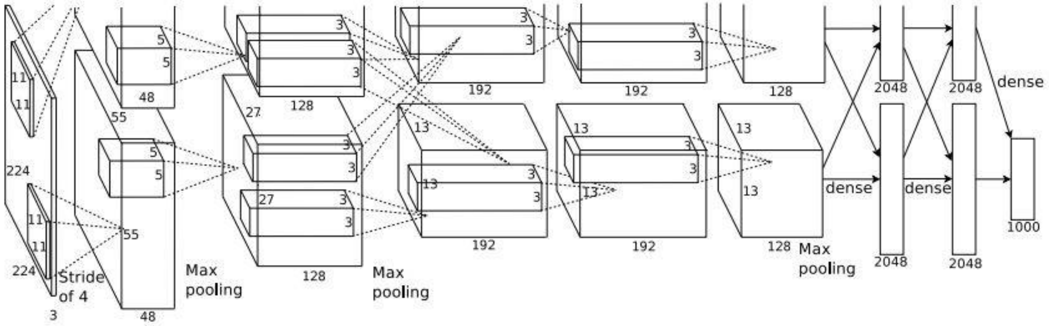


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2 [27x27x96]
NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2 [13x13x256]
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)

The size of each layer in AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

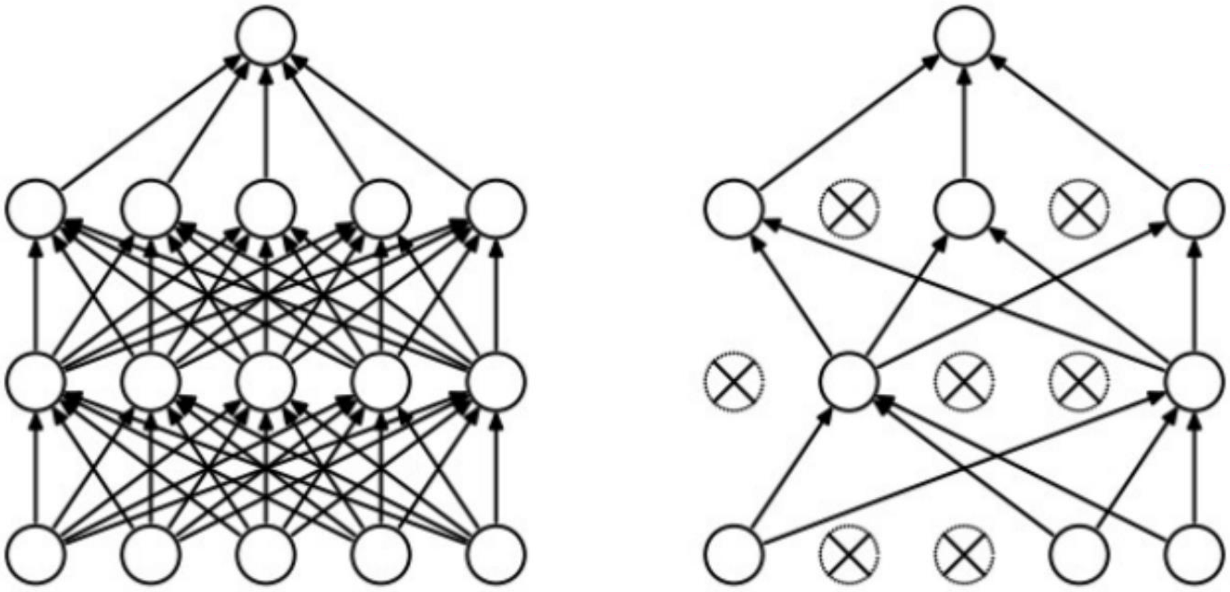


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2 [27x27x96]
NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2 [13x13x256]
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)

Details/Retrospectives:
-first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 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%

Do Better: Random in Forward-Pass

- **Dropout**

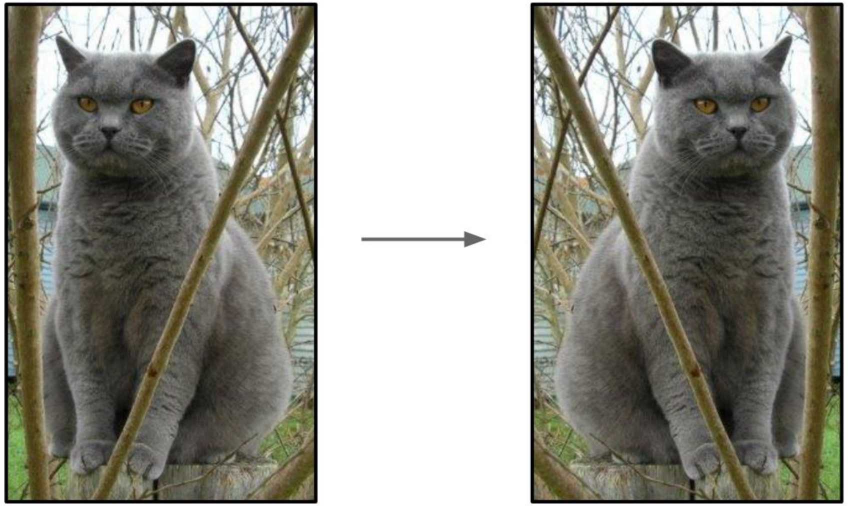


(a) Standard Neural Net (b) After applying dropout.

Krizhevsky et al. “ImageNet classification with deep convolutional neural networks,” NIPS, 2012.

Do Better: Data Augmentation

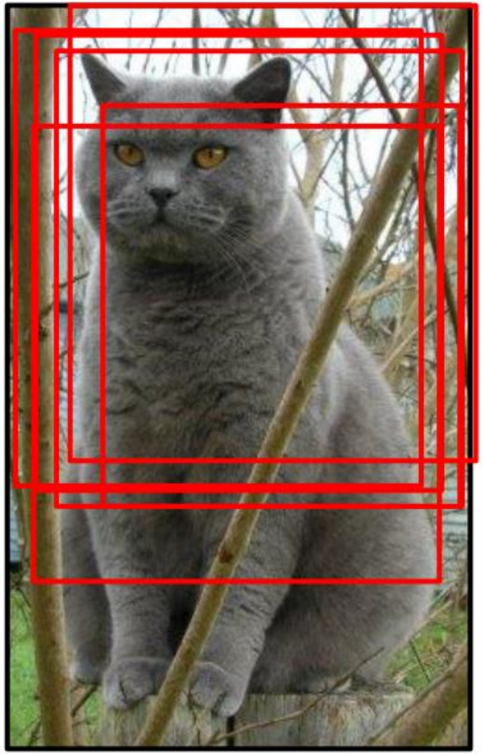
- **Simulating “fake” data**
- **Explicitly encoding image transformations that shouldn’t change object identity.**
  - **Flip horizontally**



Do Better: Data Augmentation

- **Random/Multiple crops/scales**

  Common up to 150 or more crops



Do Better: Data Augmentation

- **Random mix/combinations of :**
  - **Translation**
  - **Rotation**
  - **Stretching**
  - **Shearing,**
  - **Lens distortions**
  - **Color jittering**

- **DeepImage – Baidu 2015**
  - **Data augmentation + multi GPU (error 5.3%)**

Ren We et al., “Deep Image: Scaling up Image Recognition,” arXiv:1501.02876v1, Jan. 2015.

The final trick: Ensemble

- **Same model, different initialization**
- **Top models discovered during cross-validation**
- **Different checkpoints of a single model**

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

VGGNet

- Only 3x3 CONV stride 1, pad 1
- and 2x2 MAX POOL stride 2

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 architecures of VGG Net. Configuration D produced the best results

Best model

11.2% top-5error & 7.3% top-1 error in ILSVRC 2013

Result comparison:

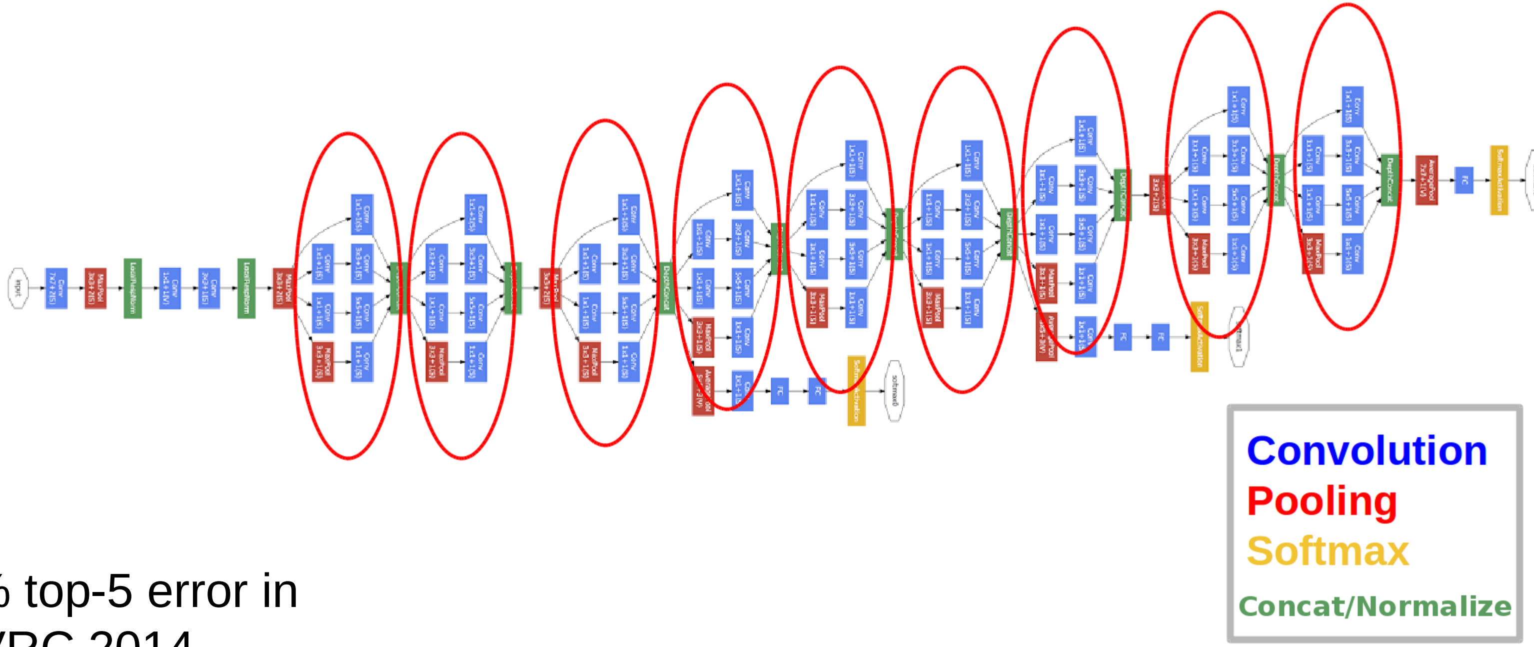
Table 7: **Comparison with the state of the art in ILSVRC classification**. Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

VGGNet: Summary

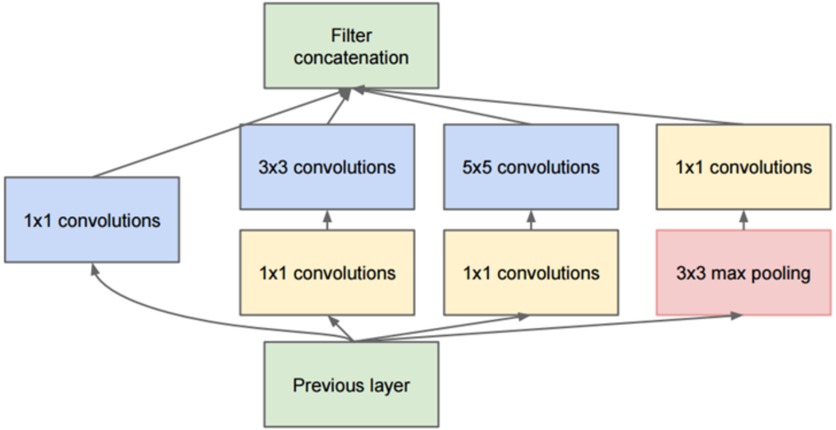
- Built model with the Caffe toolbox.
- Used scale jittering as one data augmentation technique during training.
- Interesting to notice that the number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.
- Used ReLU layers after each conv layer and trained with batch gradient descent.
- Trained on 4 Nvidia Titan Black GPUs for **two to three weeks**.
- CNNs have to have a deep network of layers in order for this hierarchical representation of visual data to work.

GoogLeNet

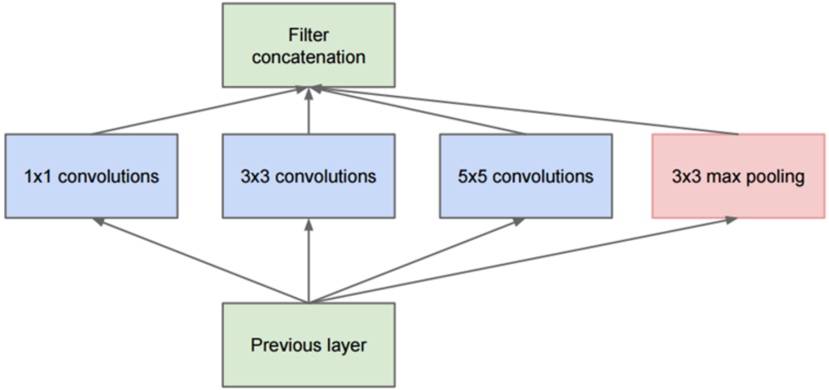


6.7% top-5 error in ILSVRC 2014

The idea of inception module



Full Inception module



Naïve idea of an Inception module

GoogLeNet classification performance break down

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Table 3: GoogLeNet classification performance break down.

GoogLeNet: summary

- Used 9 Inception modules in the whole architecture,
- No use of fully connected layers saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet.
- Trained on “a few high-end GPUs **within a week**


WE NEED TO GO DEEPER

Residual Net

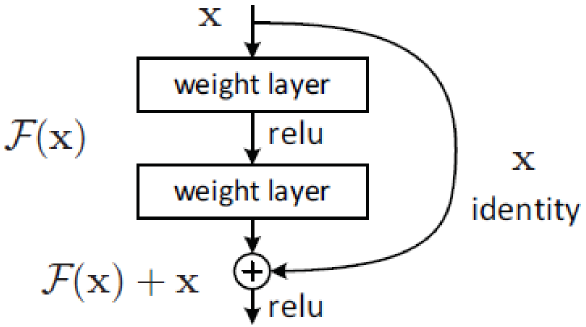
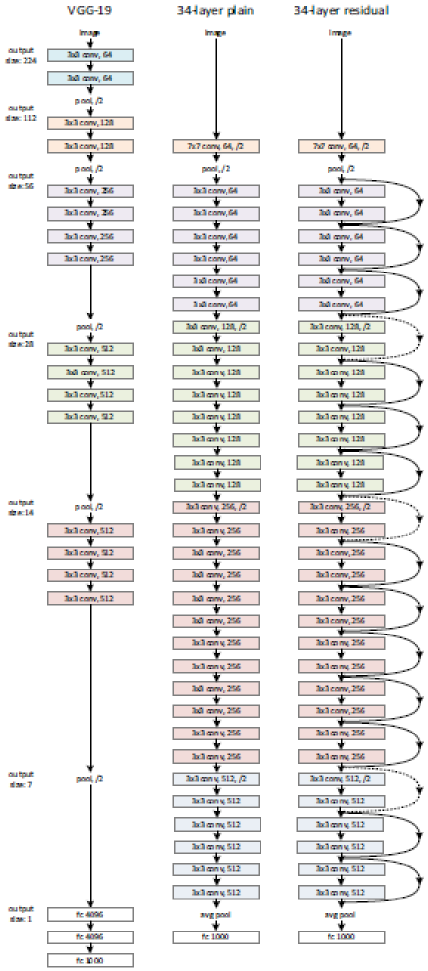
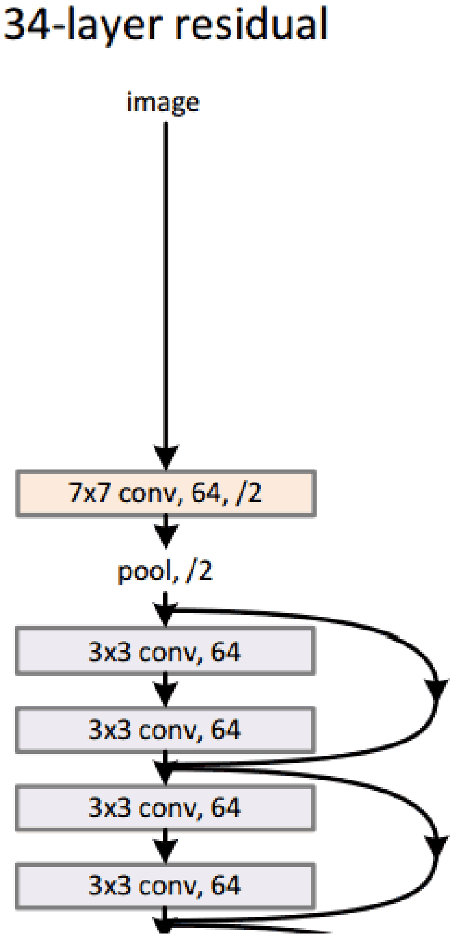


Figure 2. Residual learning: a building block.





Is going deeper always beneficial?

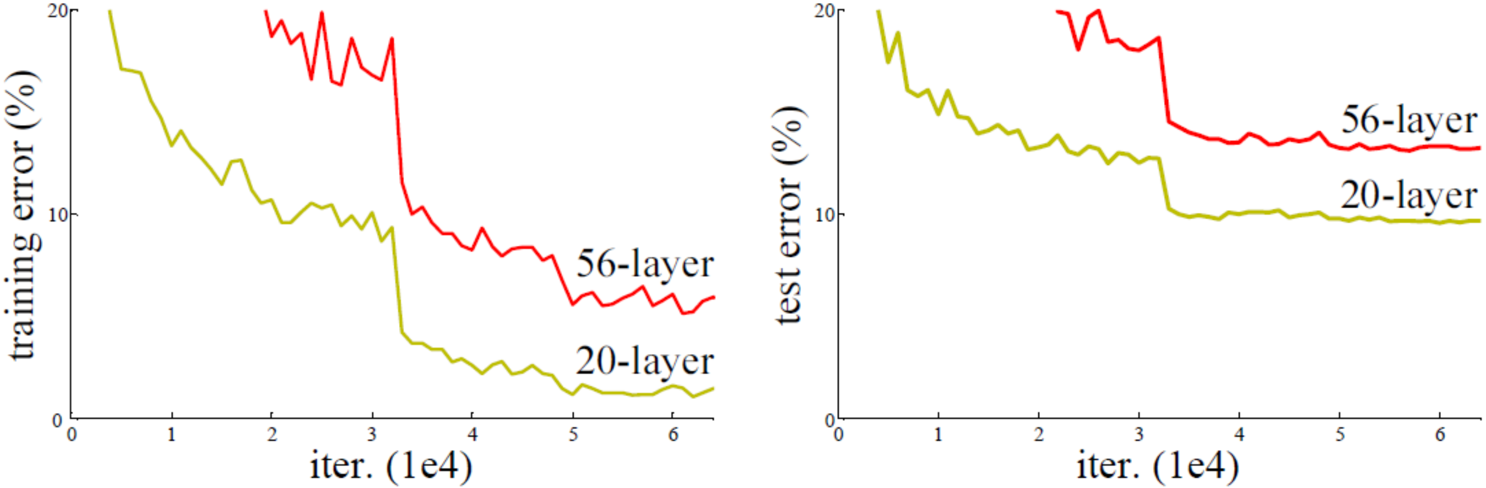


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Plain Nets vs ResNets

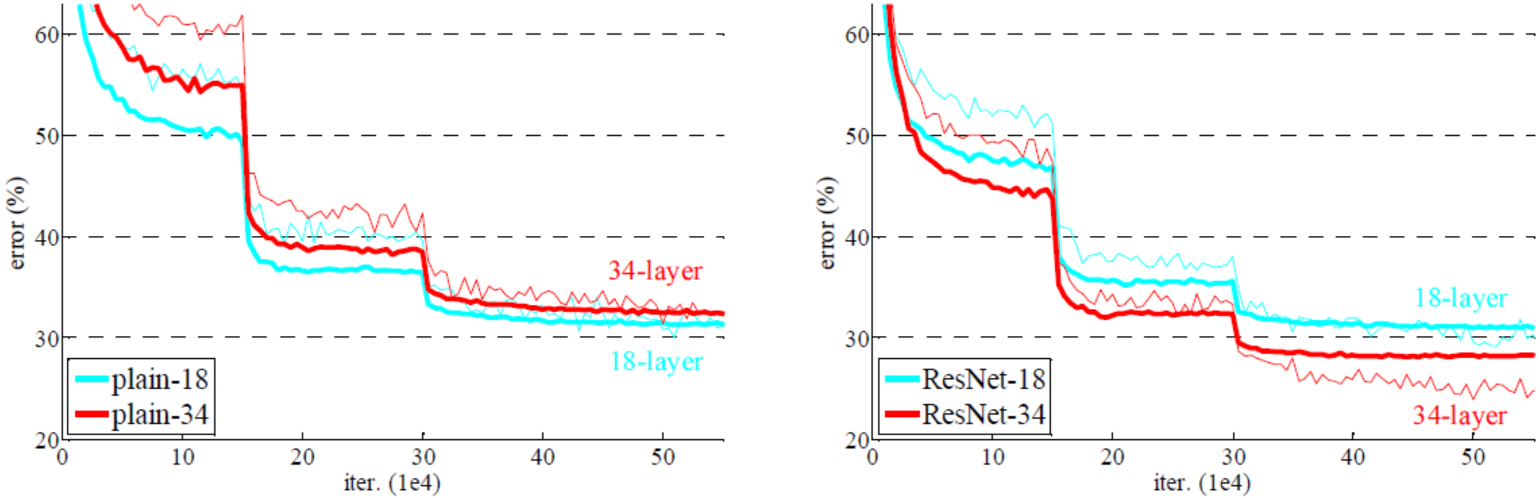


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Results: 10-crop testing on ImageNet validation

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

Results: Single model testing on ImageNet validation

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

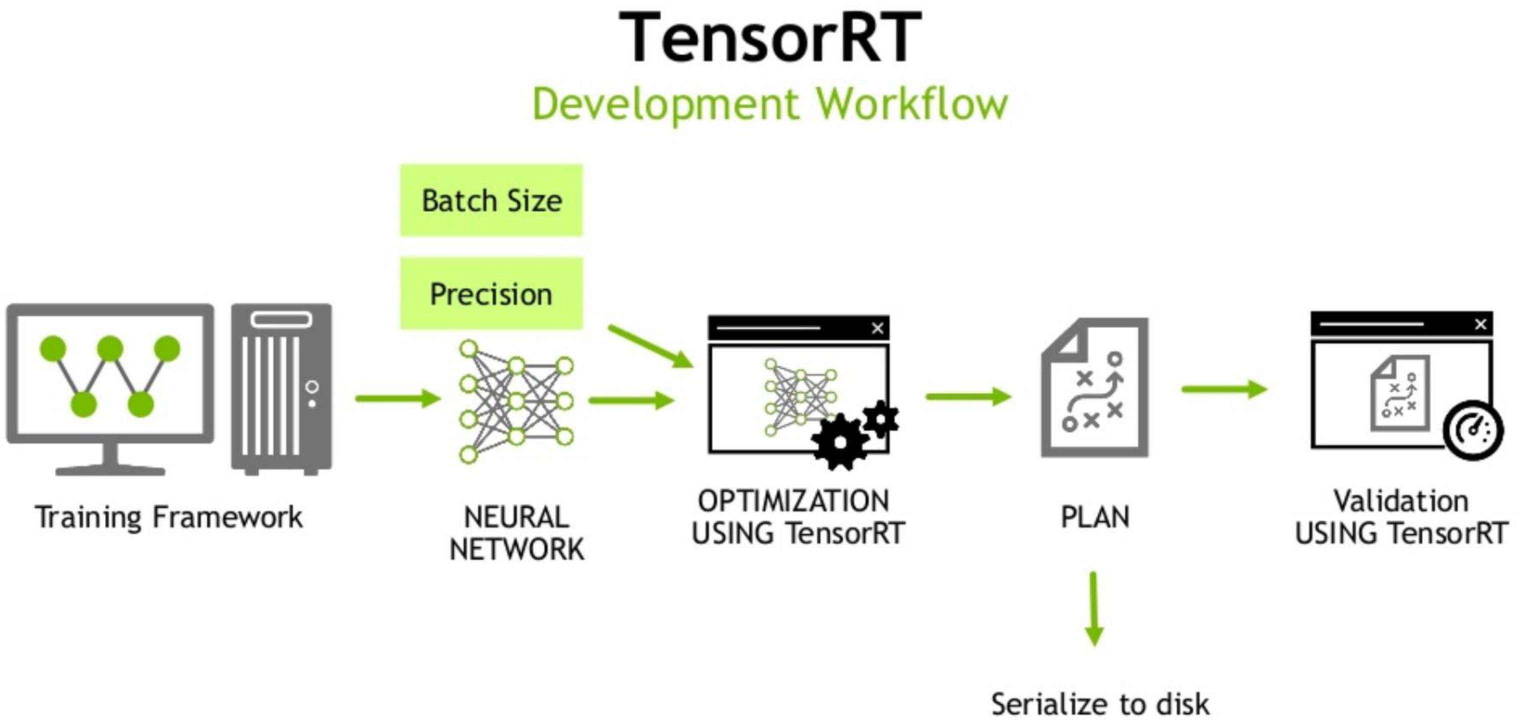
Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Best Results: Model ensembles on ImageNet test

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Can we speed up the inference?



FP32 is not inevitable

Clip slide

8-BIT INFERENCE

Top-1 Accuracy

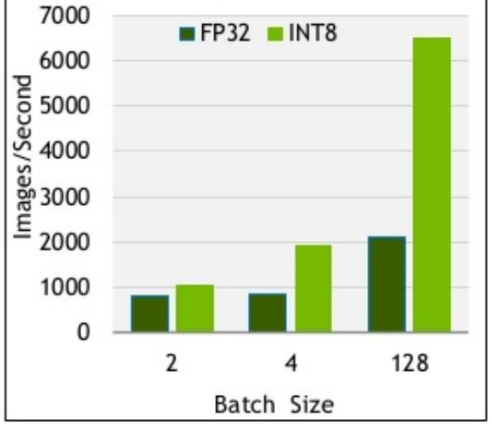
Network	FP32 Top1	INT8 Top1	Difference	Perf Gain
Alexnet	57.22%	56.96%	0.26%	3.70x
Googlenet	68.87%	68.49%	0.38%	3.01x
VGG	68.56%	68.45%	0.11%	3.23x
Resnet-152	75.18%	74.56%	0.61%	3.42x

Int8 precision

New in TensorRT

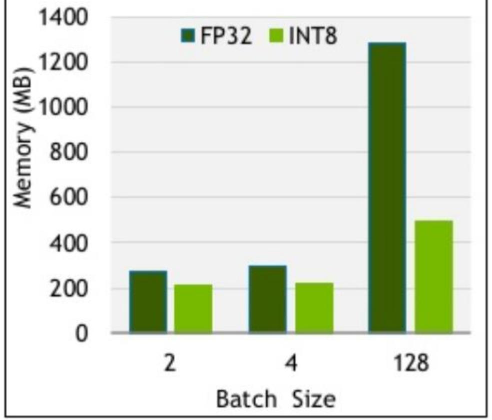
PERFORMANCE

Up To 3x More Images/sec with INT8 Precision



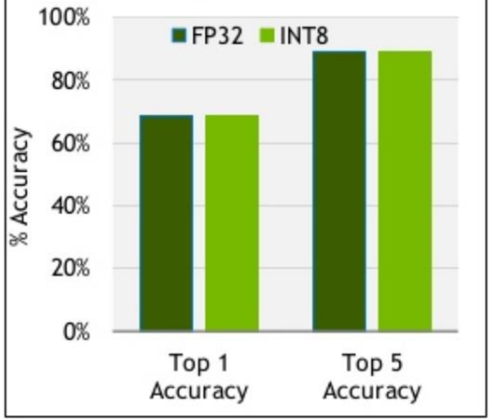
EFFICIENCY

Deploy 2x Larger Models with INT8 Precision



ACCURACY

Deliver full accuracy with INT8 precision



***GoogLenet**, FP32 vs INT8 precision + TensorRT on*
***Tesla P40 GPU**, 2 Socket Haswell E5-2698 v3@2.3GHz with HT off*

Good Artists Copy, Great Artists Steal.

-Steve Jobs

Transfer Learning

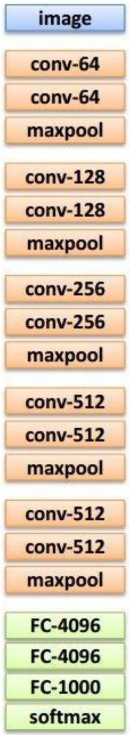
“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

"You need a lot of data if you want to train/use CNNs"

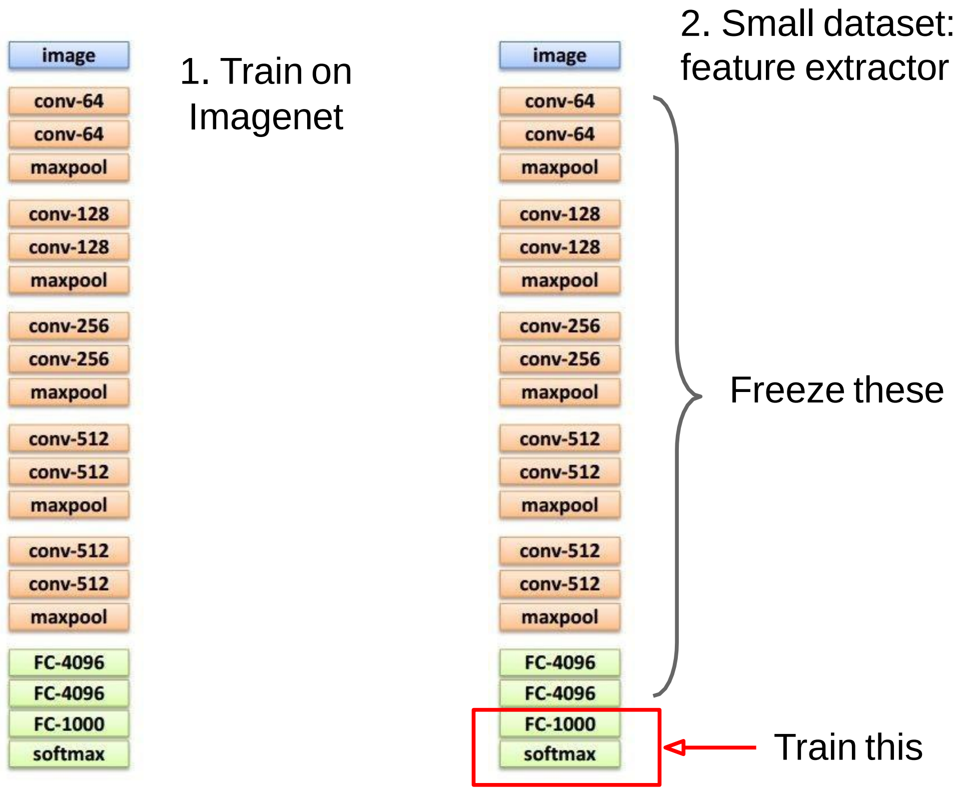
BUSTED

Transfer Learning with CNNs

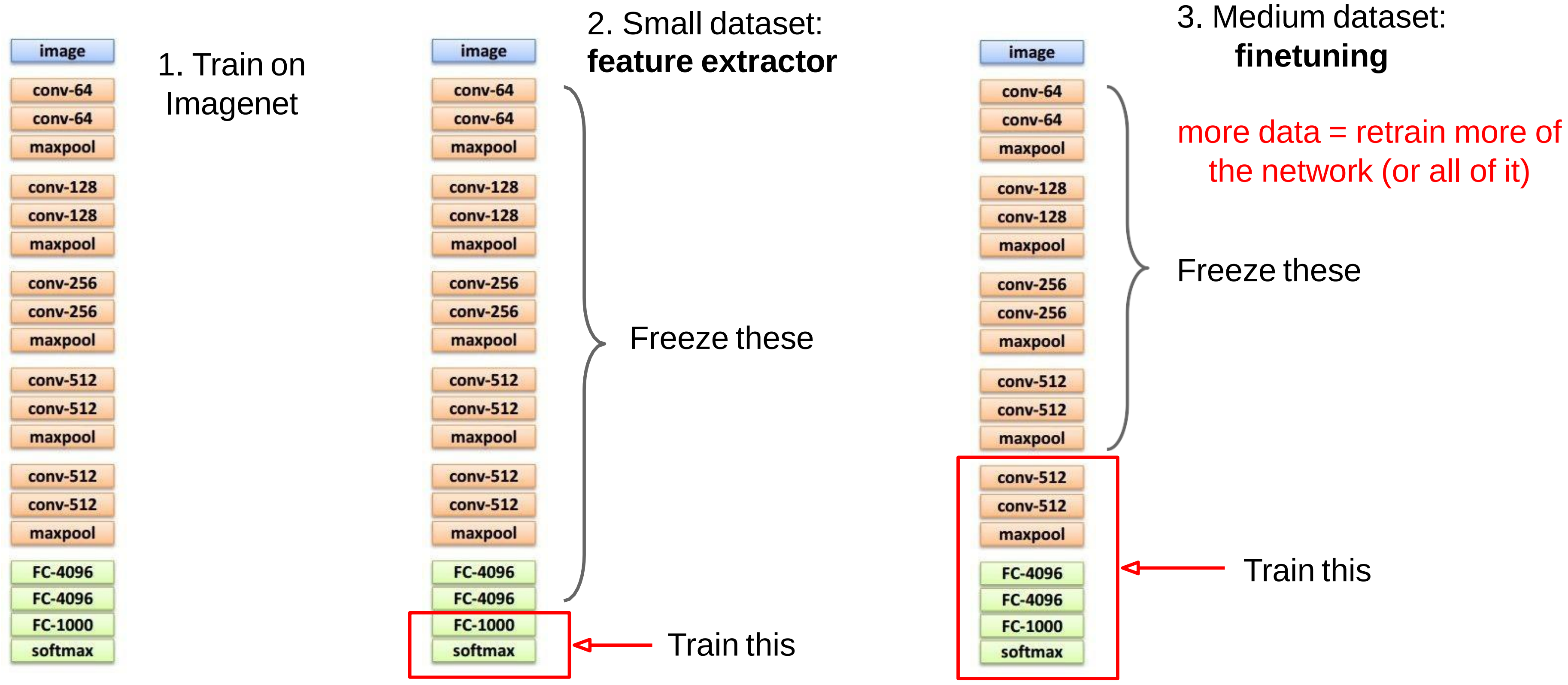


1. Train on Imagenet

Transfer Learning with CNNs



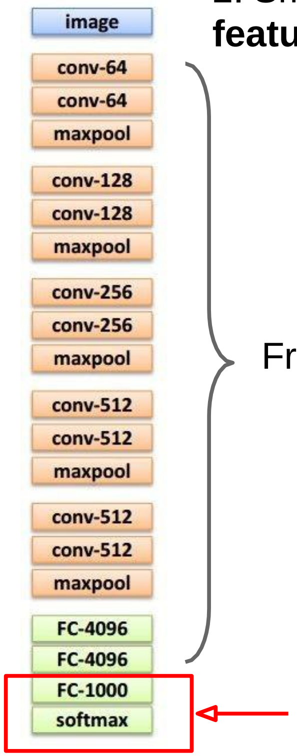
Transfer Learning with CNNs



Transfer Learning with CNNs

1. Train on Imagenet

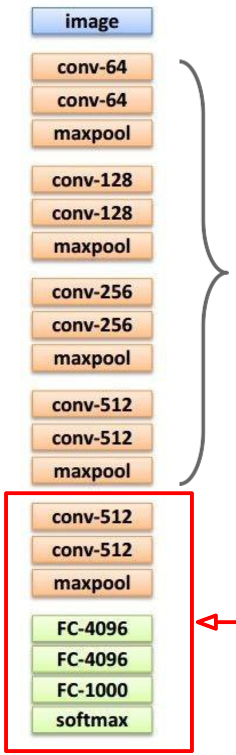
2. Small dataset: **feature extractor**



Freeze these

Train this

3. Medium dataset: **finetuning**



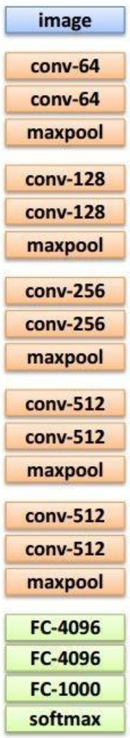
more data = retrain more of the network (or all of it)

Freeze these

tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

Train this

Transfer Learning with CNNs

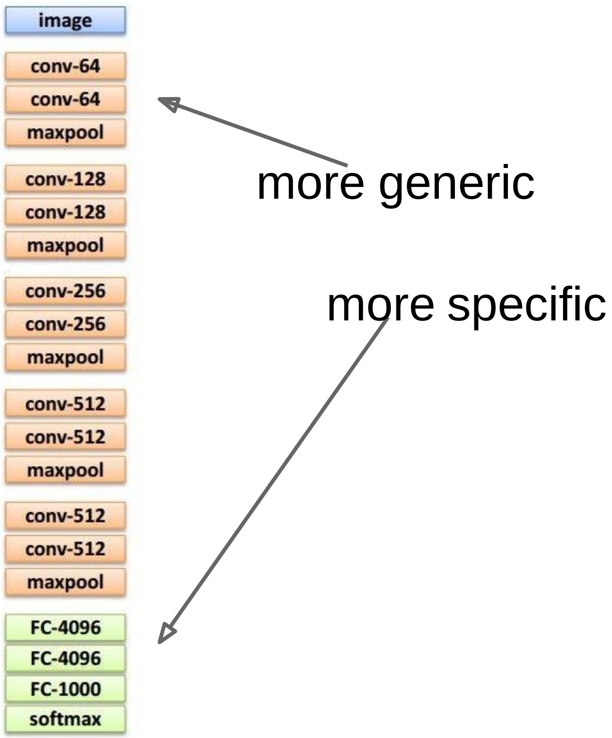


more generic

more specific

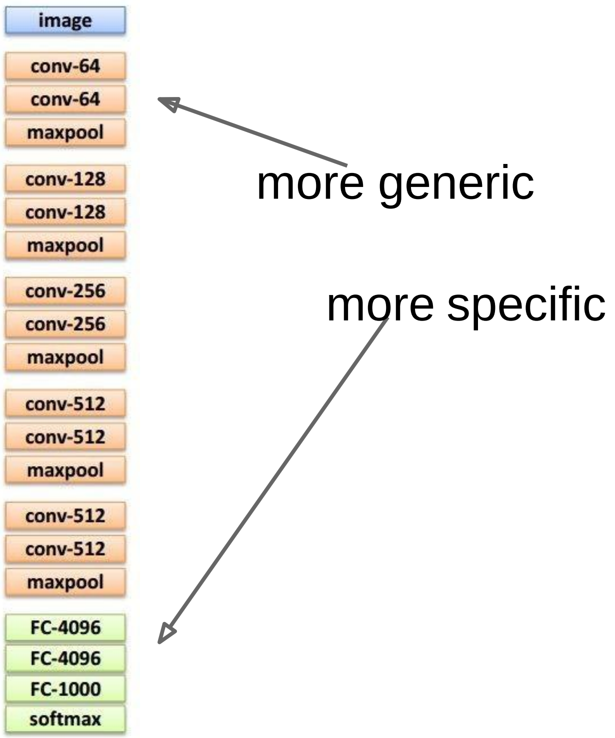
	very similar dataset	**very different dataset**
very little data	?	?
quite a lot of data	?	?

Transfer Learning with CNNs



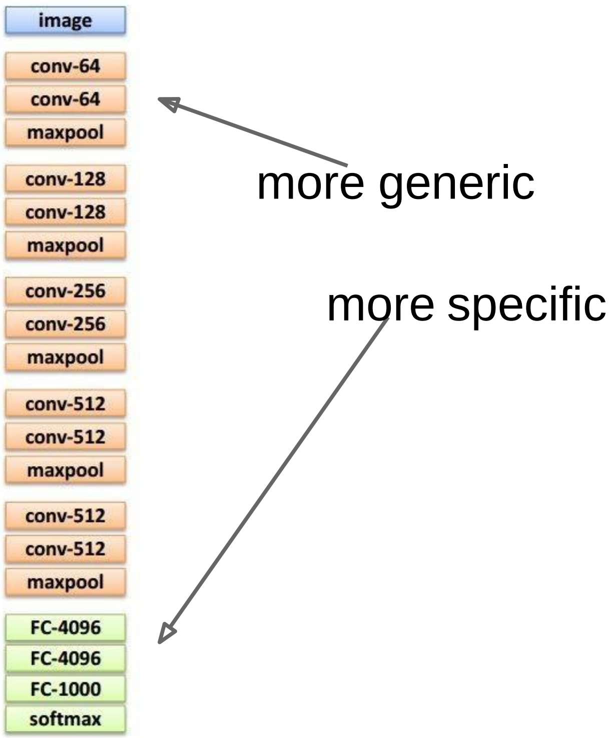
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	?
quite a lot of data	?	?

Transfer Learning with CNNs



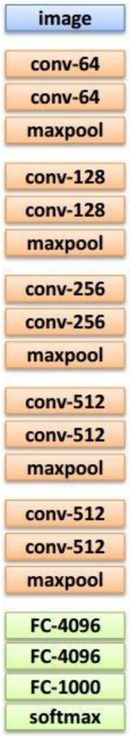
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?

Transfer Learning with CNNs



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer Learning with CNNs



more generic

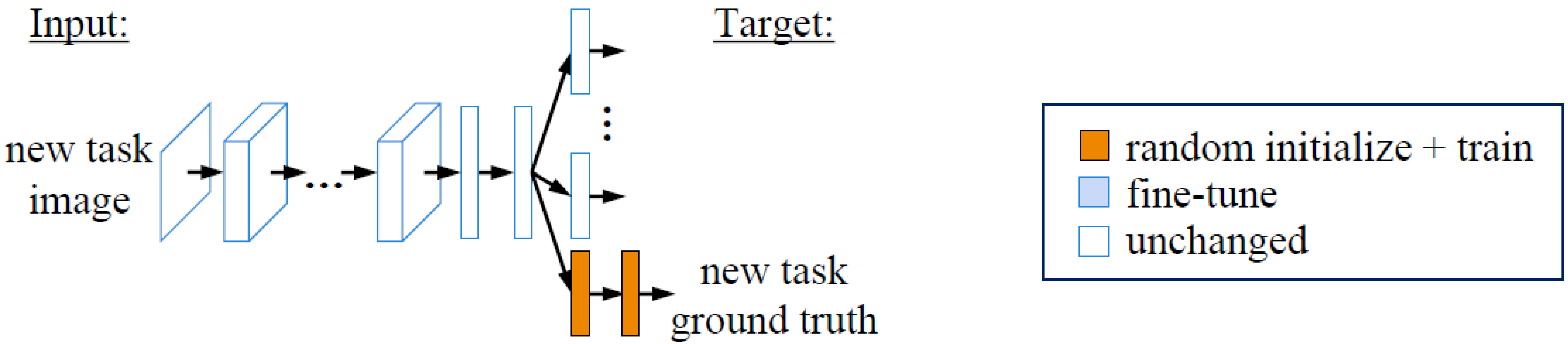
more specific

	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble!
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

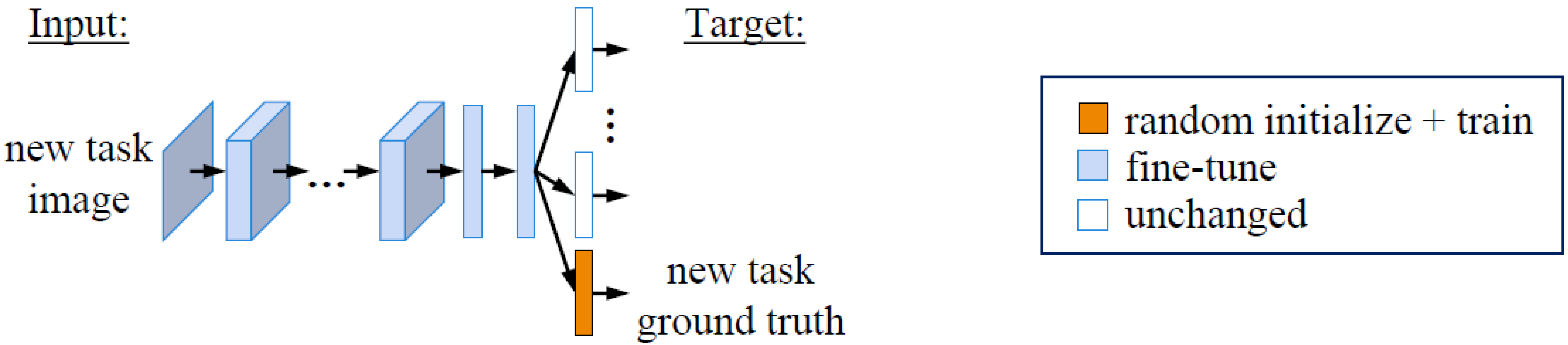
Different training approaches

- **Feature extraction**
- **Fine-tuning**
- **Joint Training**
- **Knowledge Distilling**
- **Learning Without Forgetting**

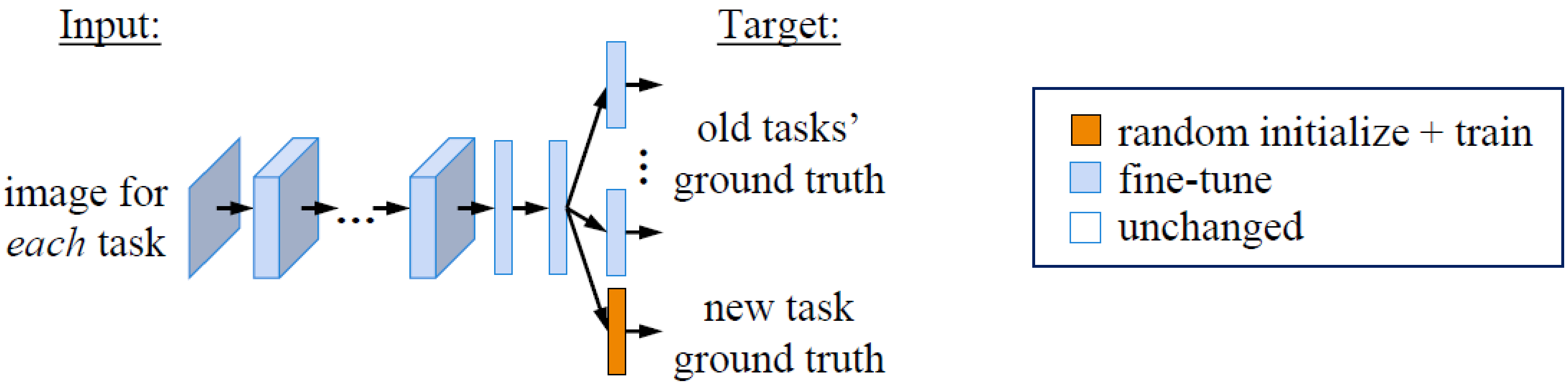
Learning approach 1: Feature Extraction



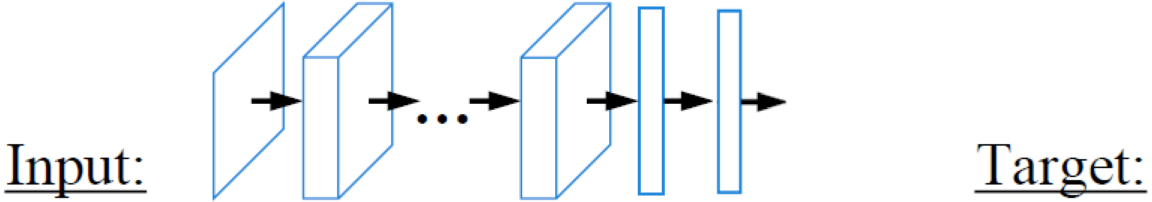
Learning approach 2: Fine-tuning



Learning approach 3: Joint Training

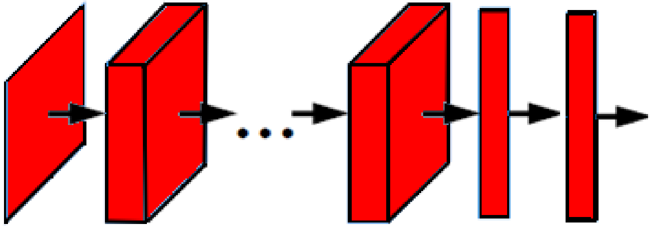


Learning approach 4: Distilling the Knowledge in a Neural Network*



Deeper

Images of a task



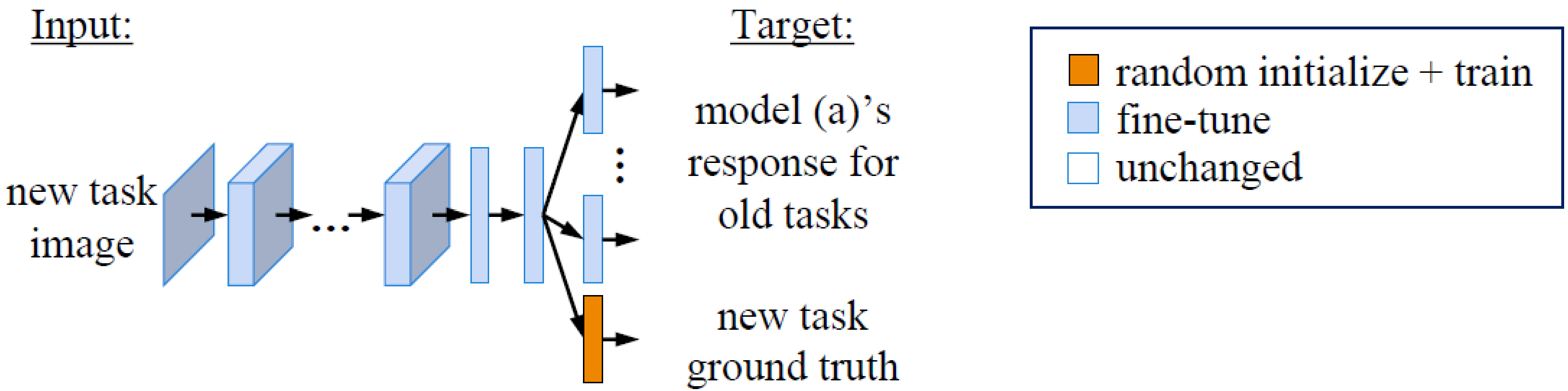
Shallow Network

(1)Output from an ensemble of Models or a deeper model

(2)Task ground truth

*Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Workshop (2014)

Learning approach 5: Learning without Forgetting



Comparison between different learning approaches

Fig. 1. We wish to add new prediction tasks to an existing CNN vision system without requiring access to the training data for existing tasks. This table shows relative advantages of our method compared to commonly used methods.

	Fine Tuning	Duplicating and Fine Tuning	Feature Extraction	Joint Training	Learning without Forgetting
new task performance	good	good	X medium	best	✓best
original task performance	X bad	good	good	good	✓good
training efficiency	fast	fast	fast	X slow	✓fast
testing efficiency	fast	X slow	fast	fast	✓fast
storage requirement	medium	X large	medium	X large	✓medium
requires previous task data	no	no	no	X yes	✓no

Thank you!