

Convolutional Neural Networks

<http://cs231n.github.io/convolutional-networks/>

Adapted by R Martinez, F. Burkholder
credit T. Heilman and M. Addonisio
Sub-set of Karpathy Lecture 7

CNN use cases - Classification



CNN use cases - Retrieval



Polar Bears: Chat with Mike Fitz & PBI ...
[medium.com](#)



Polar Bear Origins Traced to a Si...
[wired.com](#)



Irish brown bears related to p...
[irishcentral.com](#)



the Pizzly Bears ...
[rainforesttrust.org](#)



Polar Bears Did Not Descend From Brow...
[nytimes.com](#)



Grizzly-polar bear hybrid - Wikipe...
[en.wikipedia.org](#)



'Escaped' bear shot dead at German zoo ...
[dw.com](#)



Brown Bears versus Polar Bears | Nature ...
[earthtimes.org](#)



to Survive a Bear Attack: Brown, Black ...
[themanual.com](#)



Polar Bear Evolution | Where Do Polar ...
[polarbearfacts.net](#)

CNN use cases

Classification



Retrieval

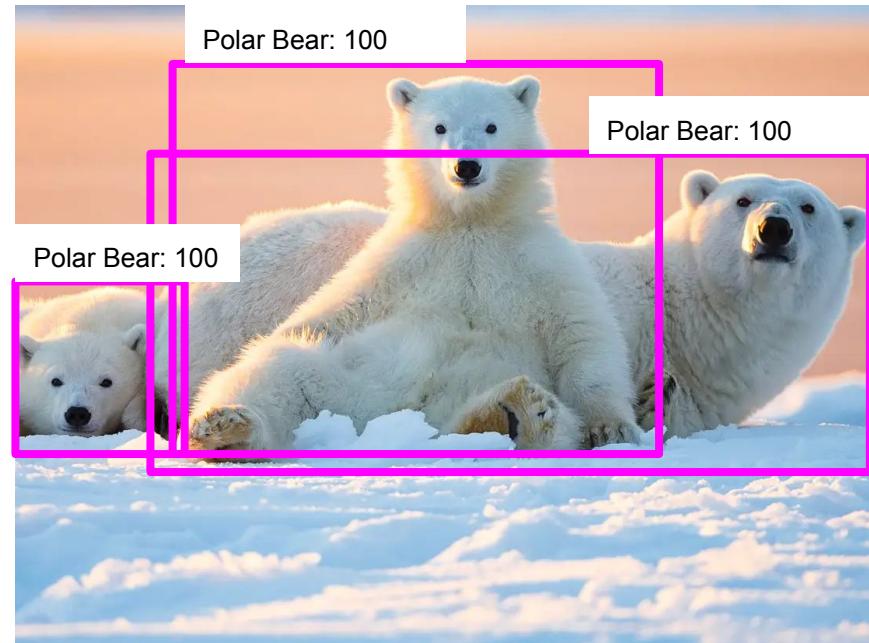
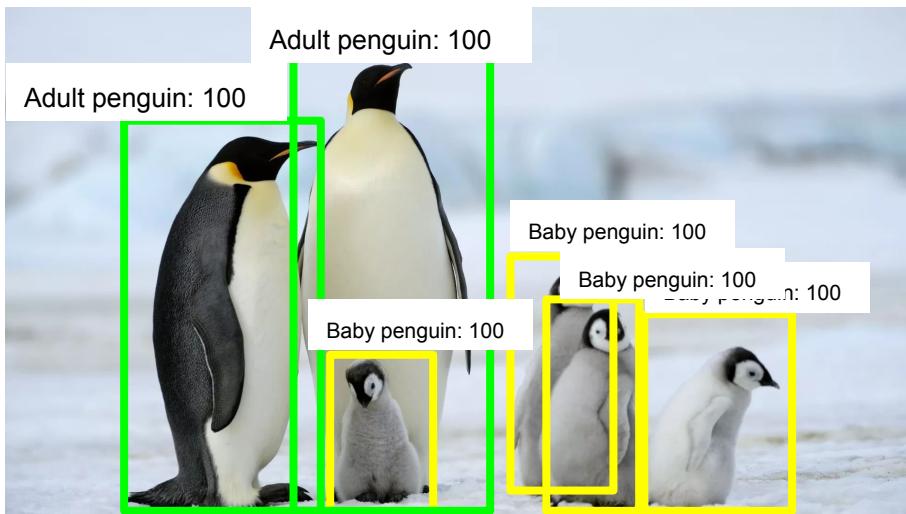


Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

CNN use cases - Object Detection



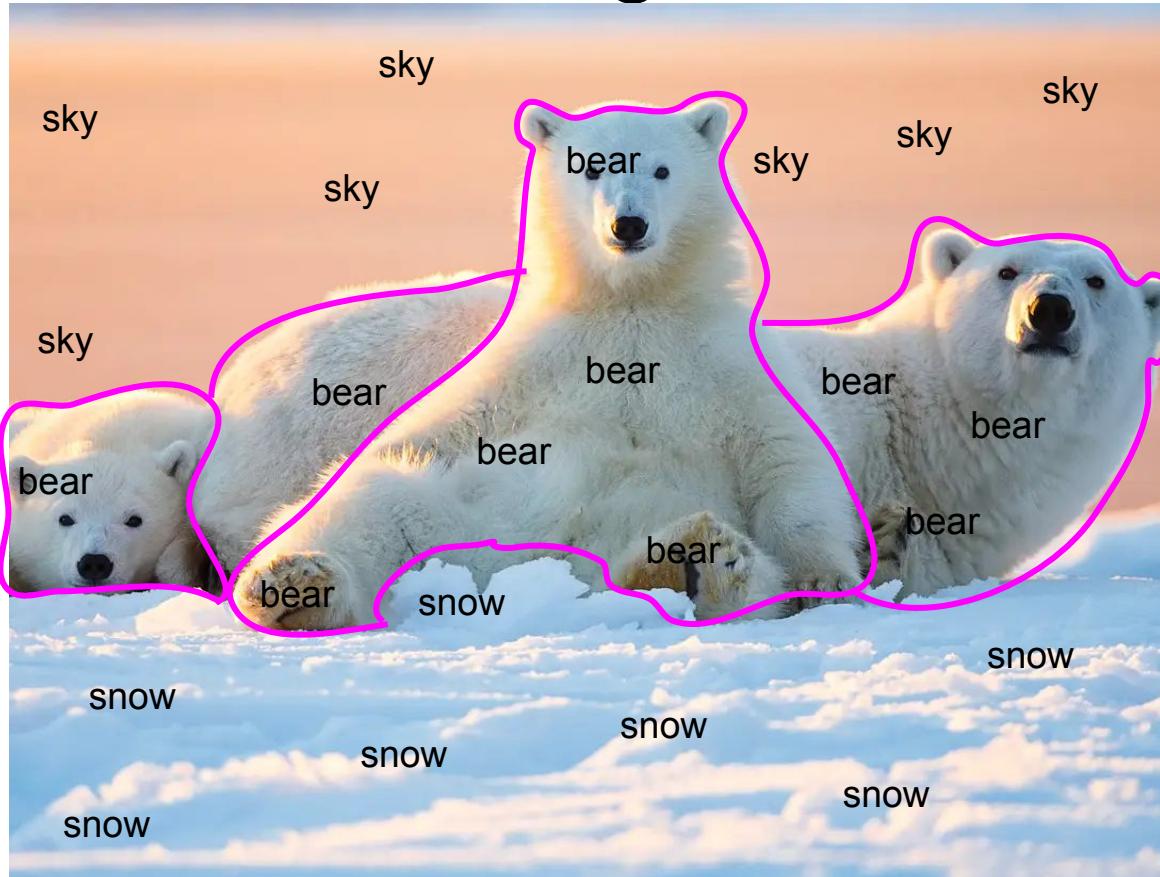
CNN use cases - Object Detection



CNN use cases - Segmentation



CNN use cases - Segmentation



CNN use cases

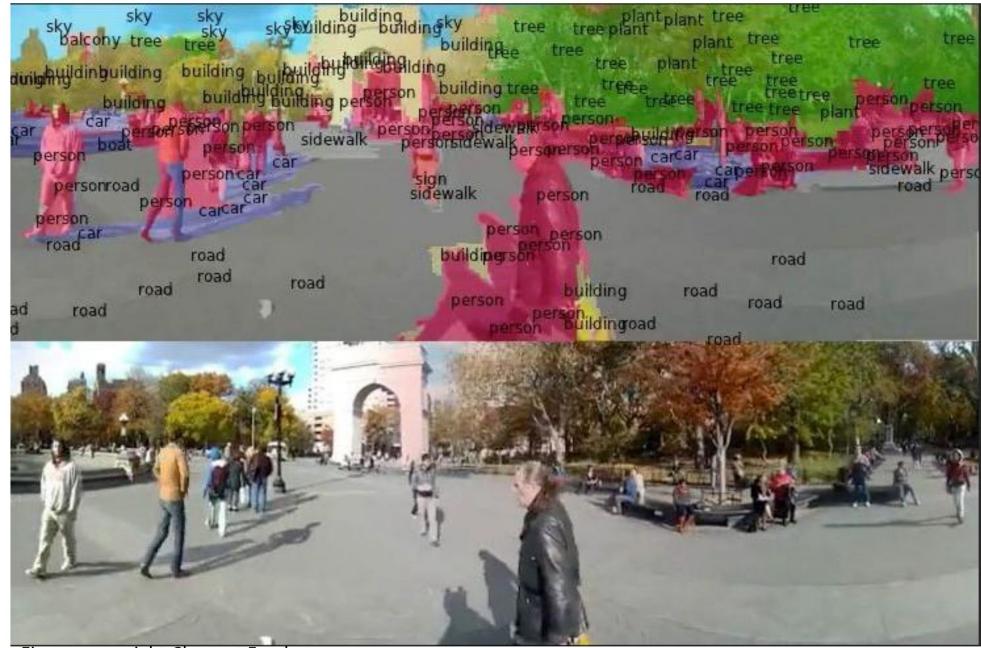
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

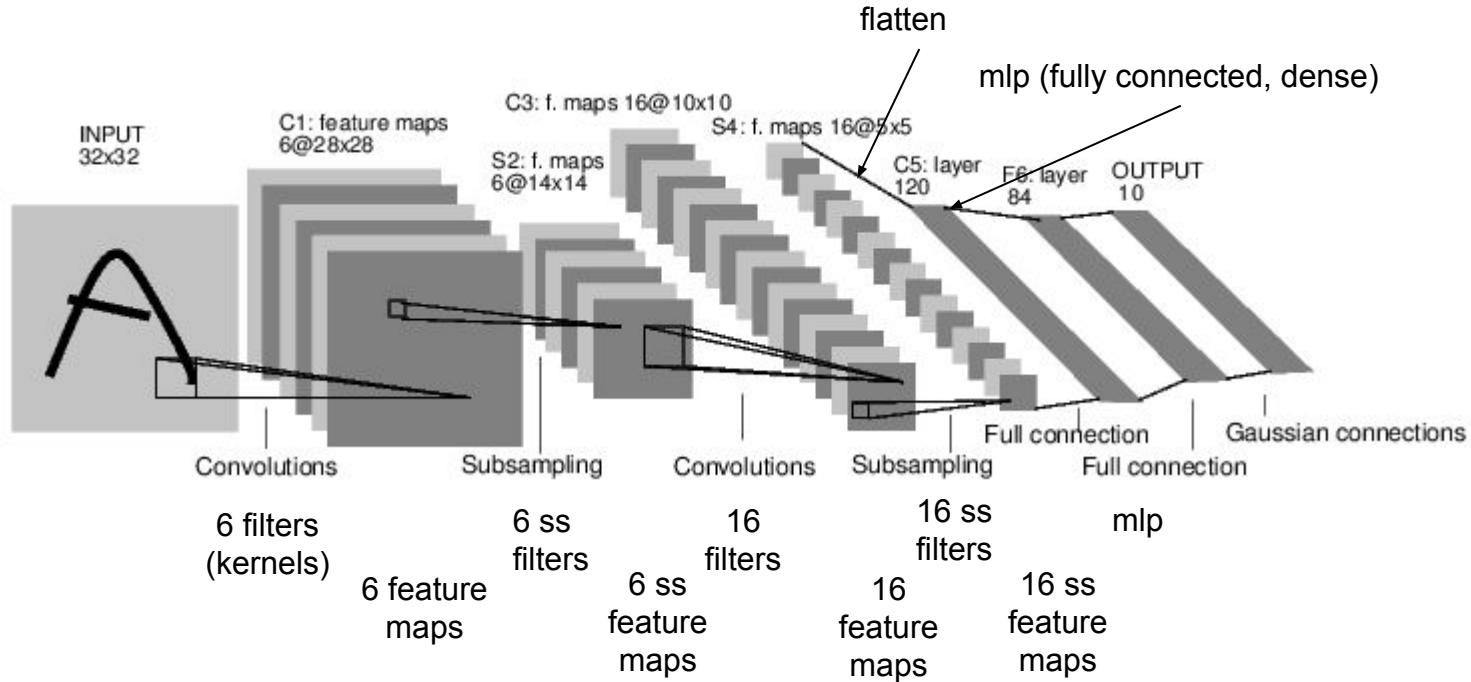
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



[Farabet et al., 2012]

CNN architecture overview



[LeNet-5, LeCun 1980]

What is an image kernel?

“An image kernel (filter, convolution) is a small matrix used to apply effects like you might find in Photoshop or Gimp, such as blurring, sharpening, etc. They're used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as *convolution*”

- Wikipedia

[Image kernels explained visually](#)

Convolutional Neural Networks

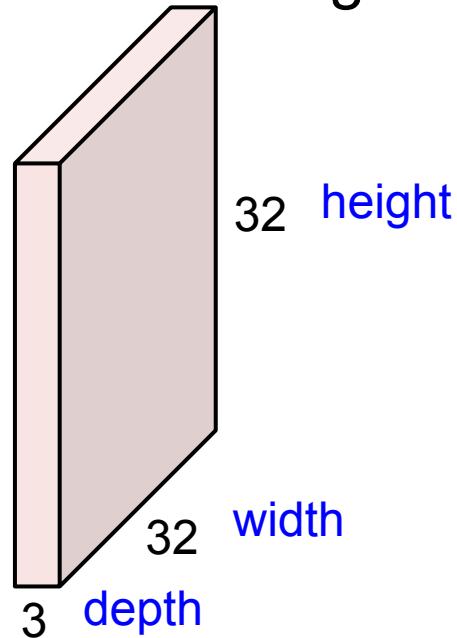
How they work?

- 1) Convert input image to 3D array of numbers
- 2) Create Filter (smaller 3D array of numbers)
- 3) Place Filter at upper left and calculate accordance
- 4) Move filter one step (“stride”) and register accordance
- 5) Repeat for all positions of filter on image

Convolution Layer

1) Convert input image to 3D array of numbers

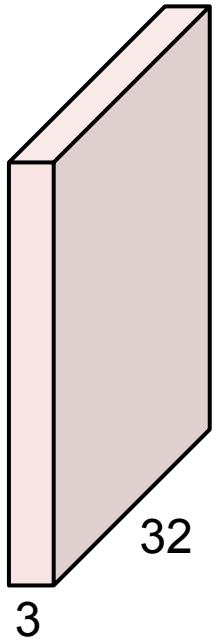
32x32x3 image



Convolution Layer

2) Create Filter (smaller 3D array of numbers)

32x32x3 image



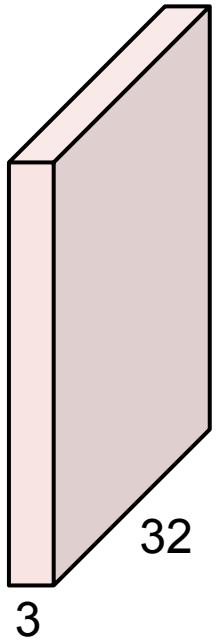
5x5x3 filter



Convolve the filter with the image
i.e. slide the filter over the image, in
the process computing the *sum* of
the *element-wise products*.

Convolution Layer

32x32x $\mathbf{3}$ image



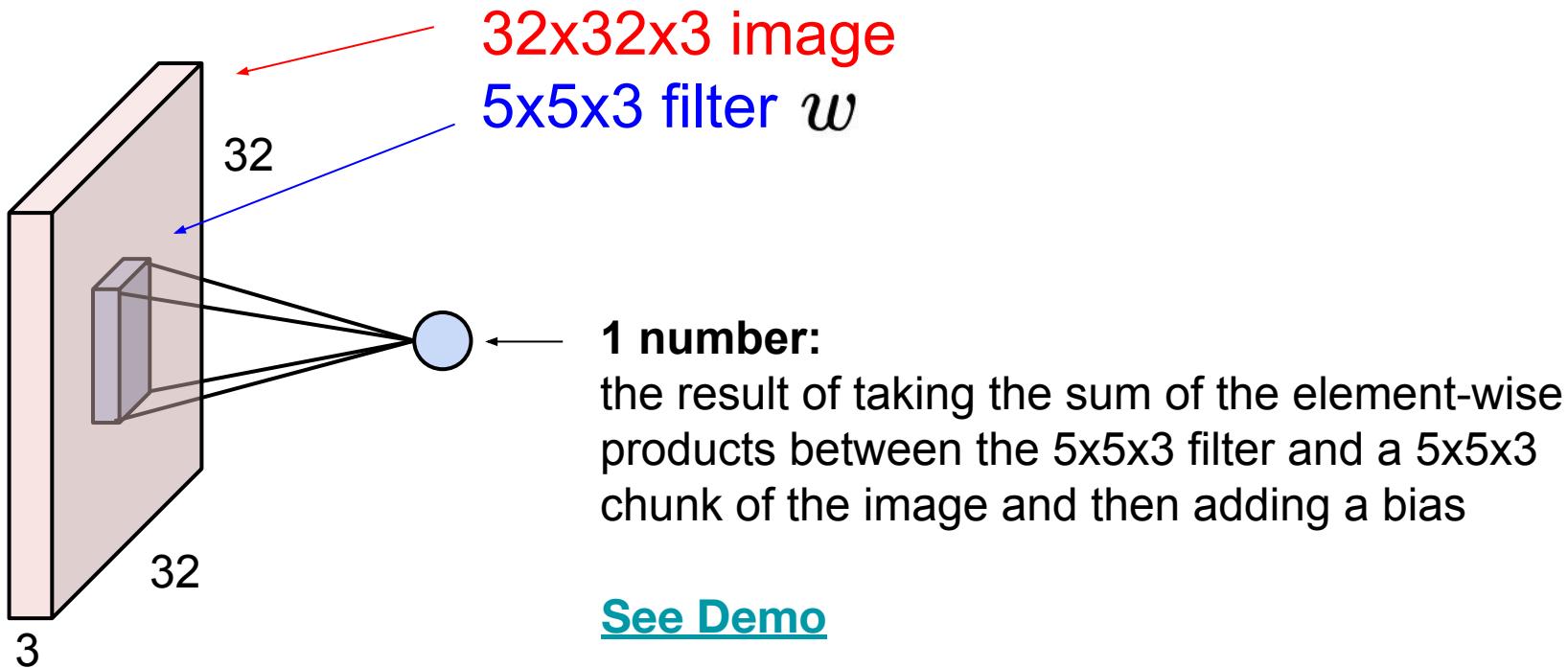
5x5x $\mathbf{3}$ filter



Filters always extend the full depth of the input volume

Convolve the filter with the image
i.e. slide the filter over the image, in
the process computing the *sum* of
the *element-wise products*.

Convolution Layer

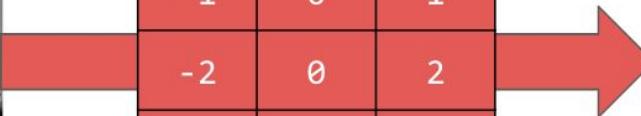




[Source](#)



$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$



[Source](#)



$$\begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{array}$$



[Source](#)

Convolution Breakout

1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

Input Image
(5x5x1)

*

1	1	1
1	4	1
1	1	1

Filter
(3x3x1)

= ?

Convolution Breakout

1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

Input Image
(5x5x1)

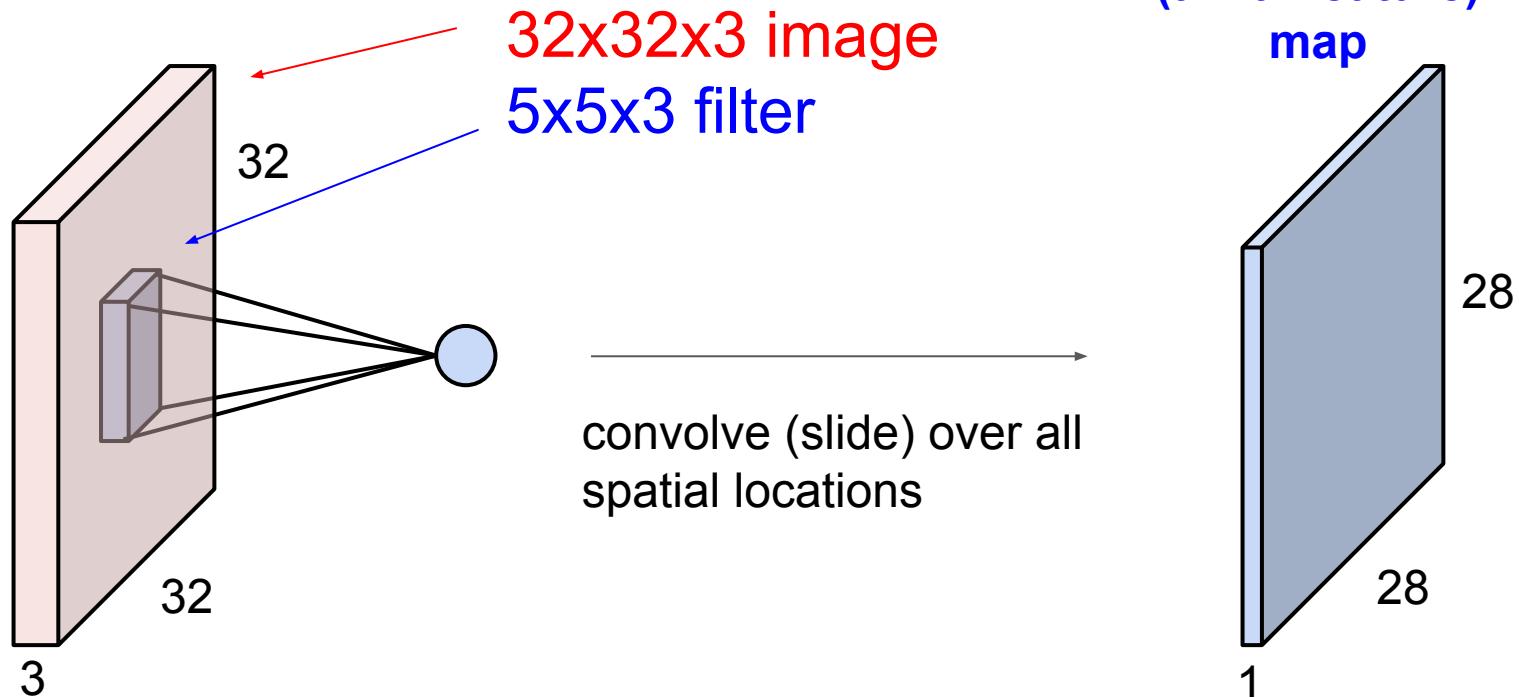
*

1	1	1
1	4	1
1	1	1

Filter
(3x3x1)

=

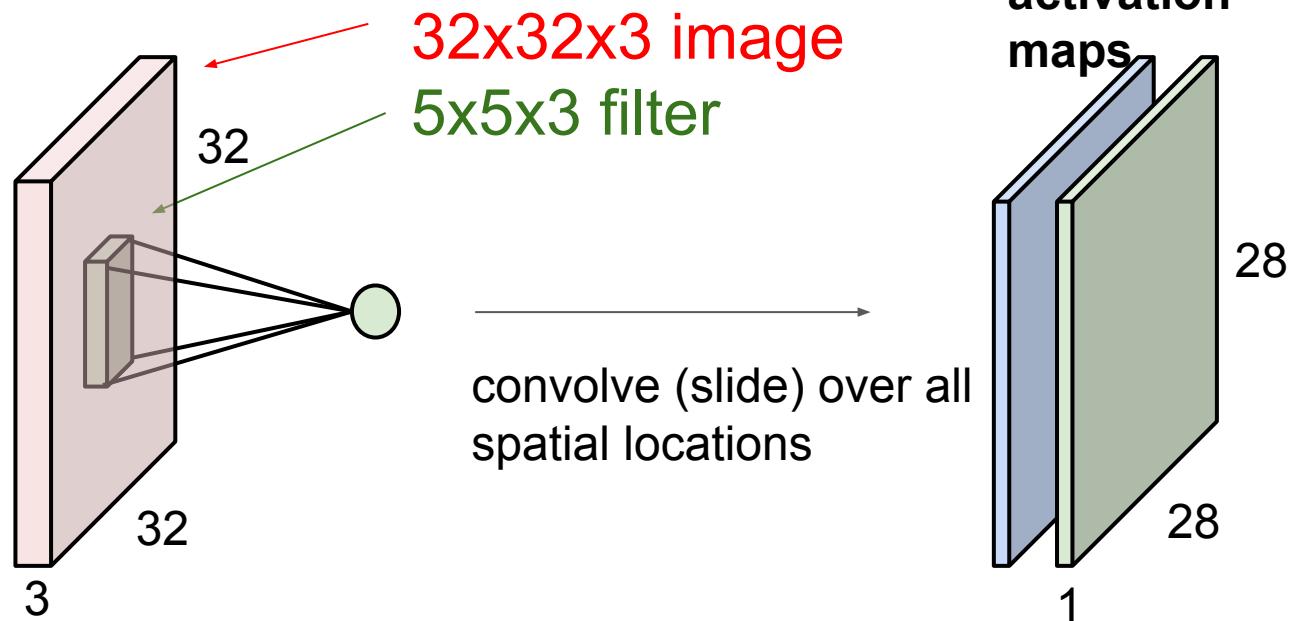
Convolution Layer



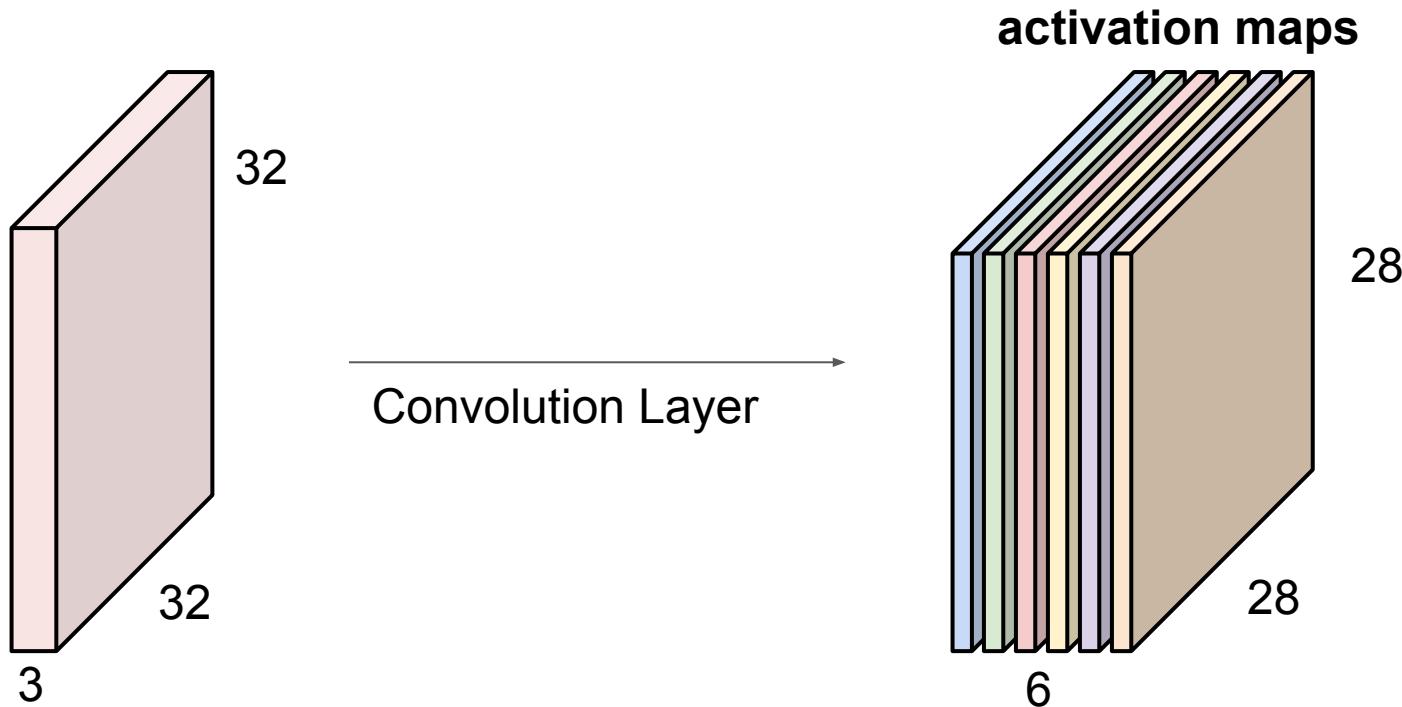
Convolution Layer

consider a second, green filter

- We can have as many filters as we want at each layer
- More filters → slower training, more accurate?
- Generally, at least 1 filter for each identifiable feature of an image

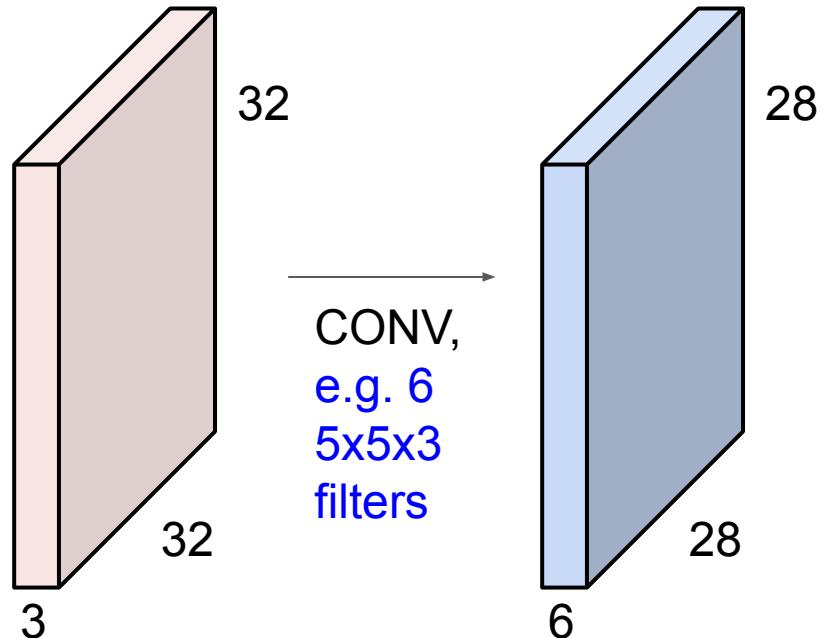


For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:

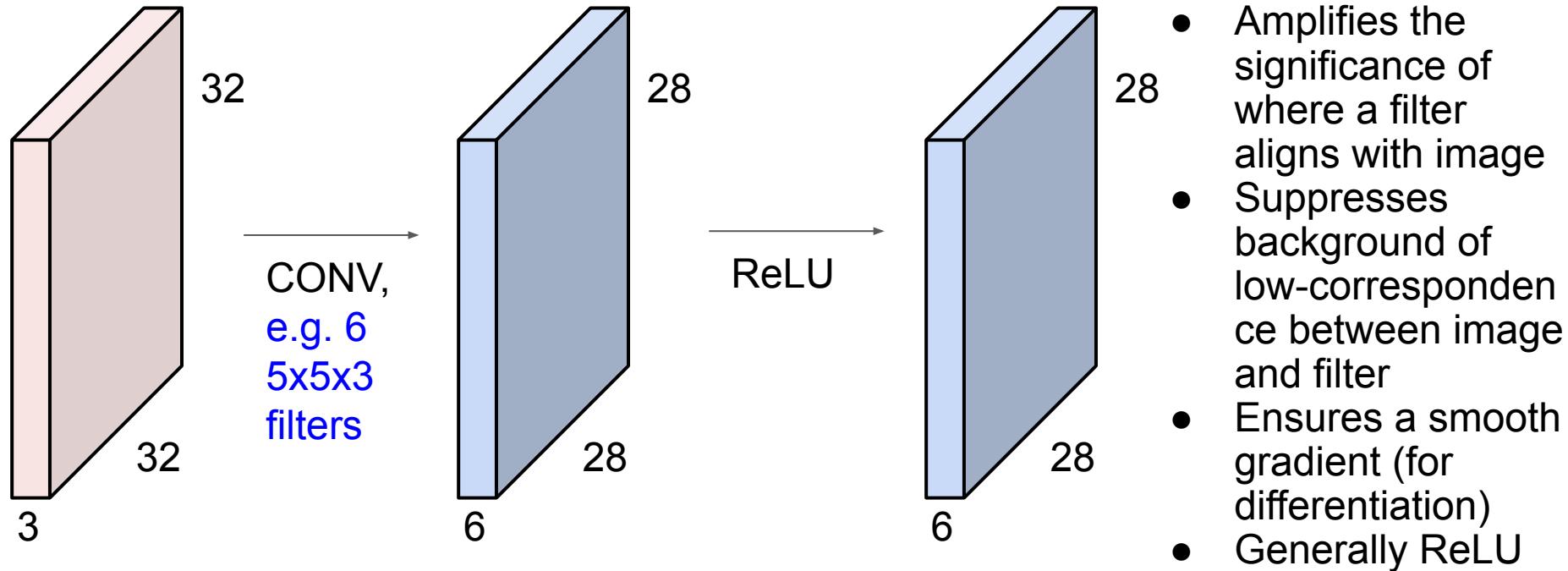


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

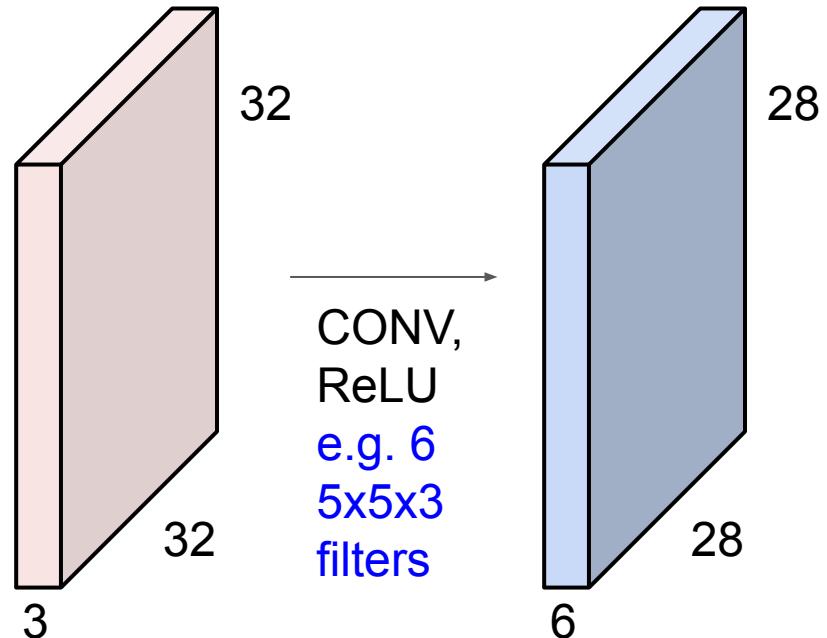
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



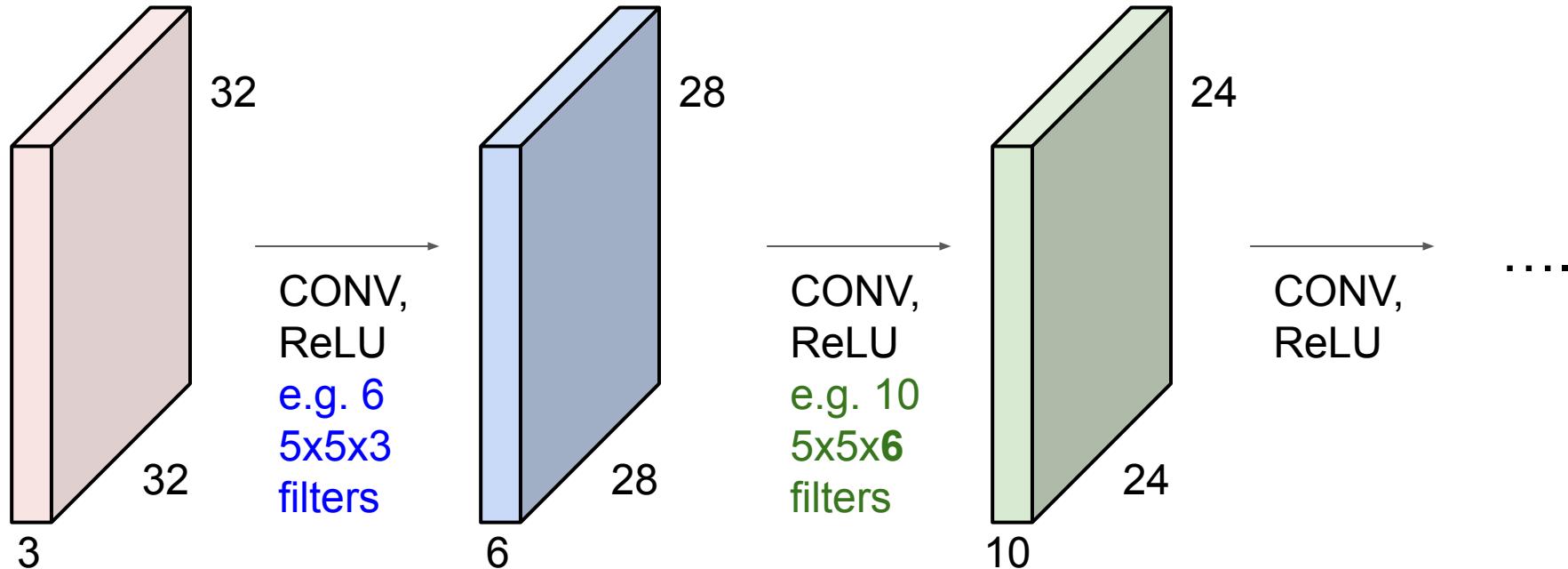
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



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



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



preview:

250 x 250 x 3
image

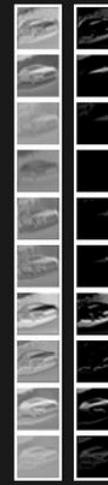


preview:

250 x 250 x 3
image



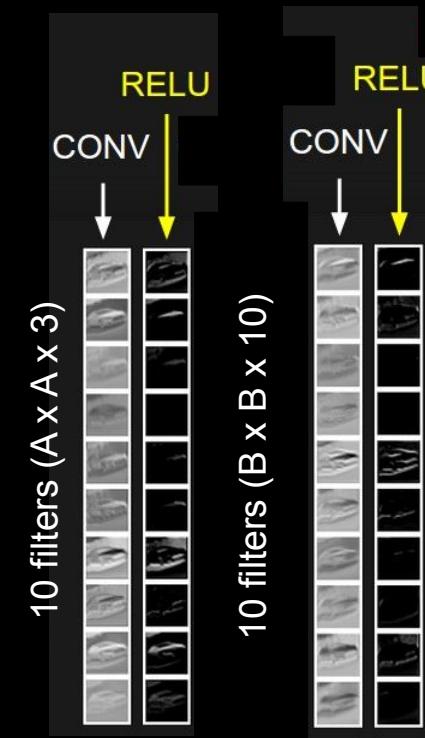
10 filters ($A \times A \times 3$)



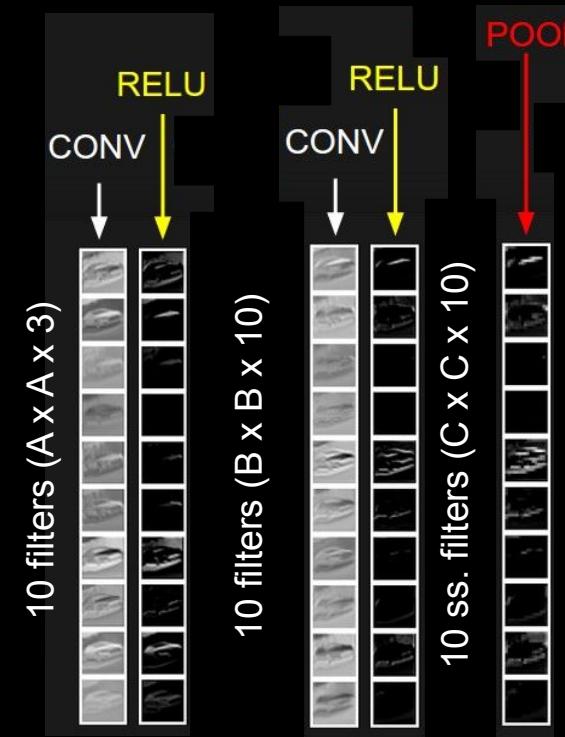
RELU

CONV

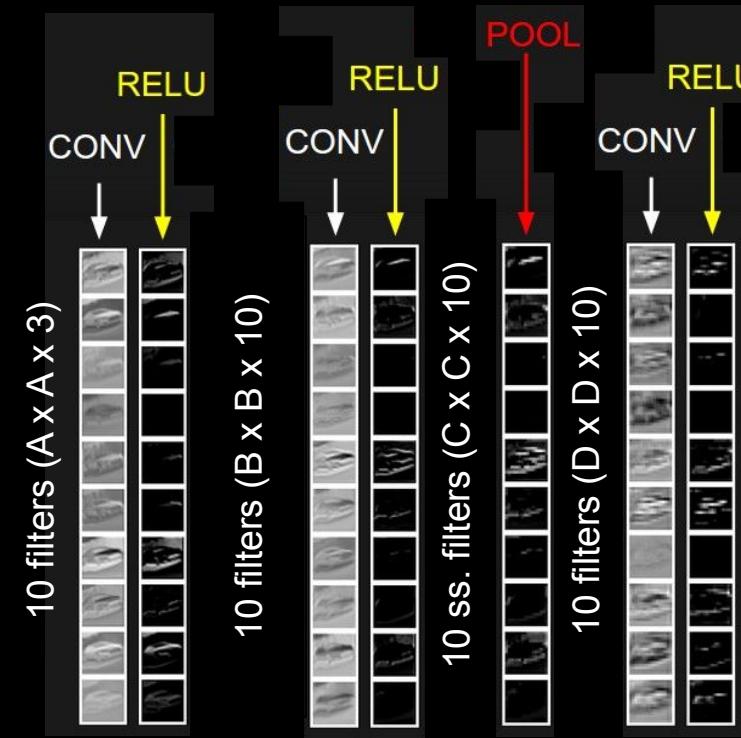
preview:



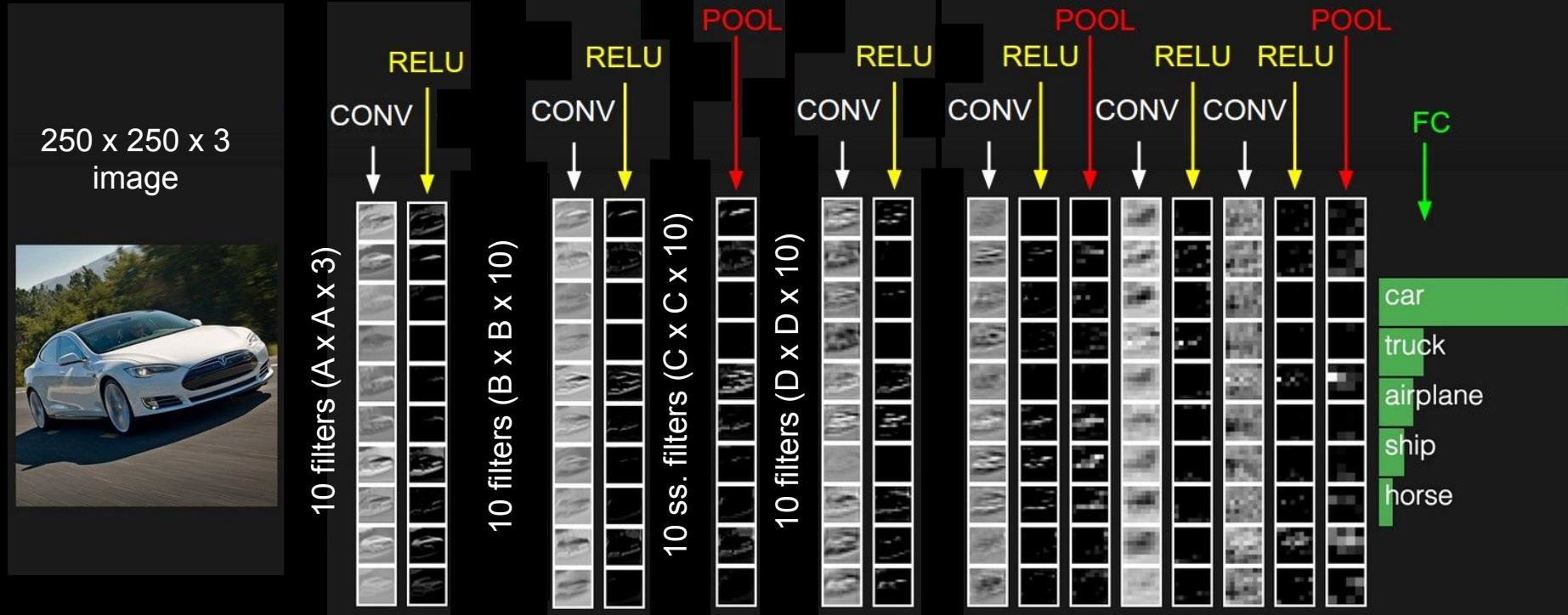
preview:



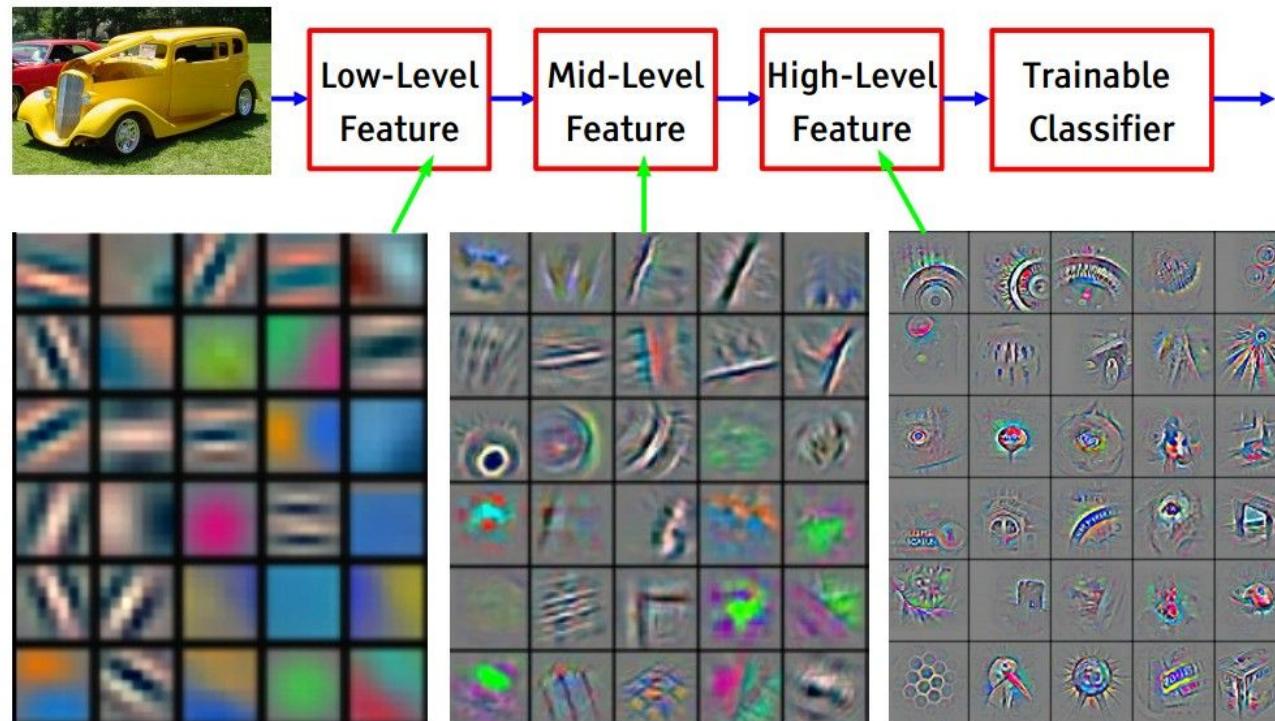
preview:



preview:



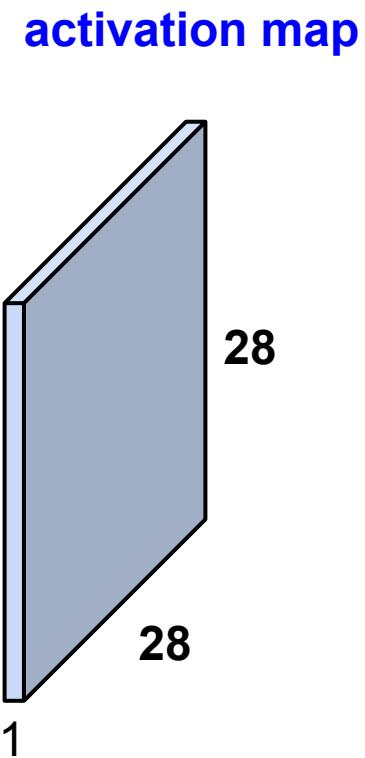
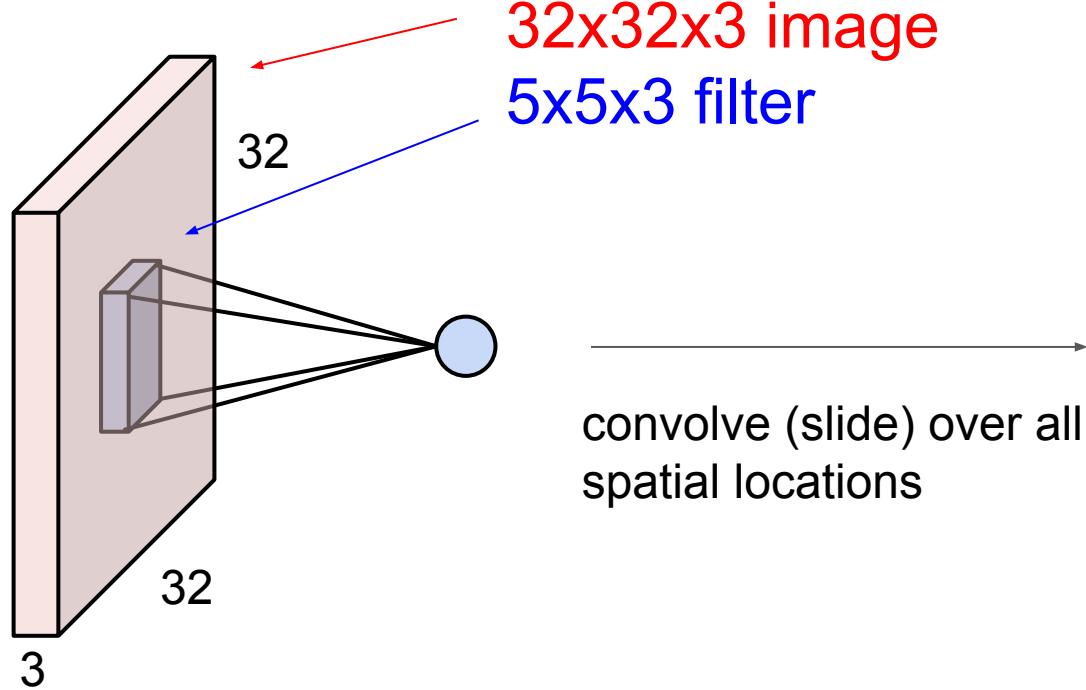
Hierarchical organization



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Activation (feature) maps

[From recent Yann LeCun slides]

A closer look at spatial dimensions:



A closer look at spatial dimensions:

3) Place Filter at upper left and calculate accordance

7

7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

- 4) Move filter one step (“stride”) and register accordance

7

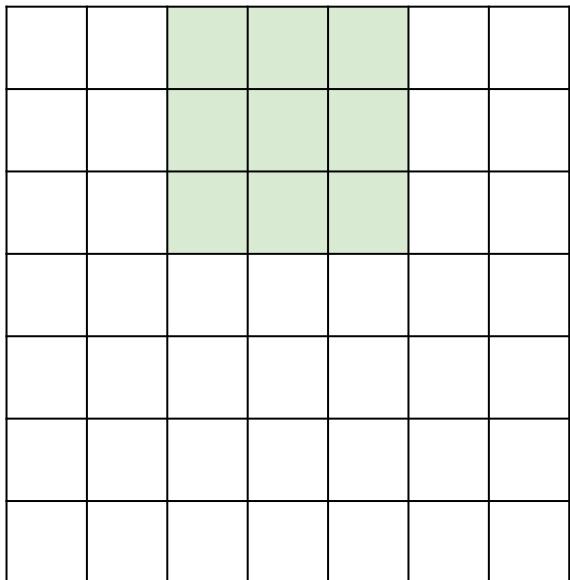
7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

5) Repeat for all positions of filter on image

7

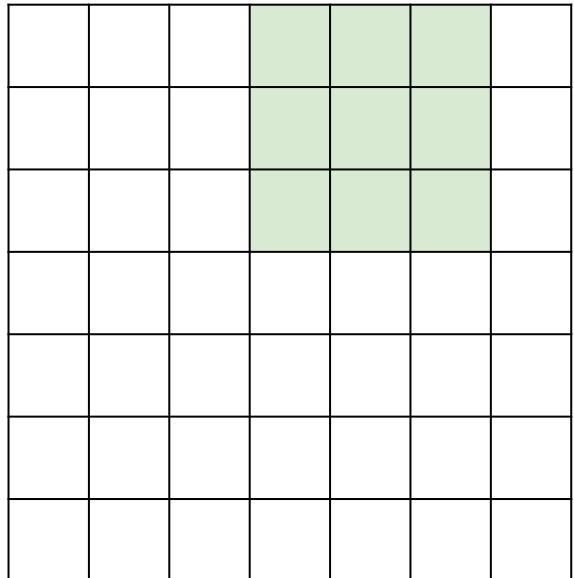


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

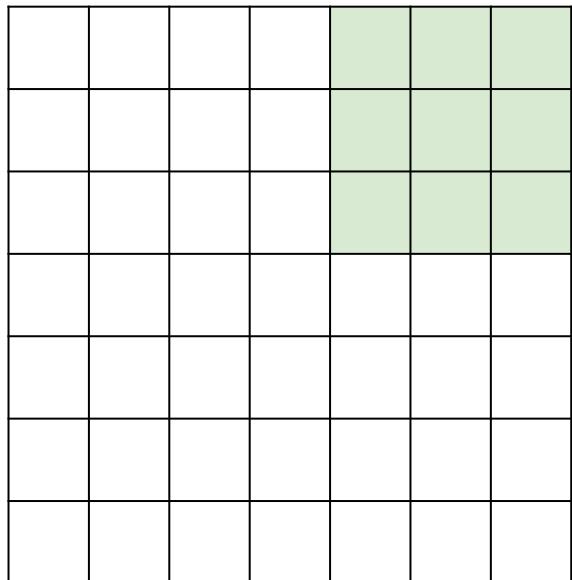


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

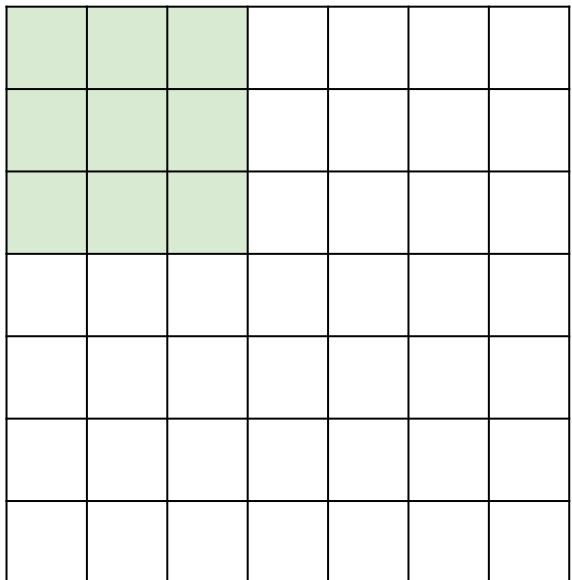


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:

7

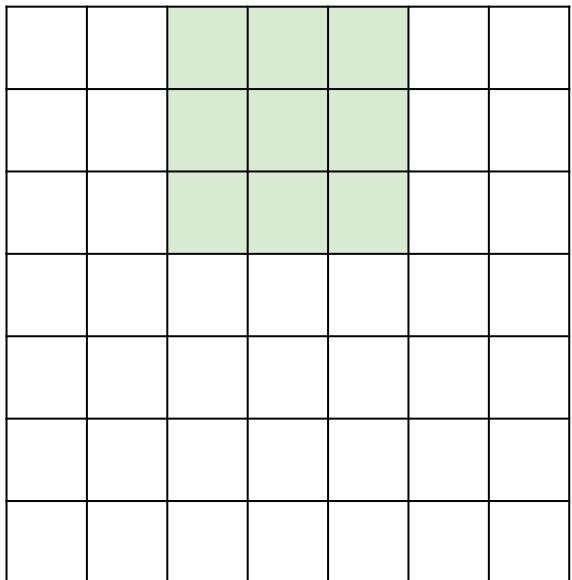


7

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

A closer look at spatial dimensions:

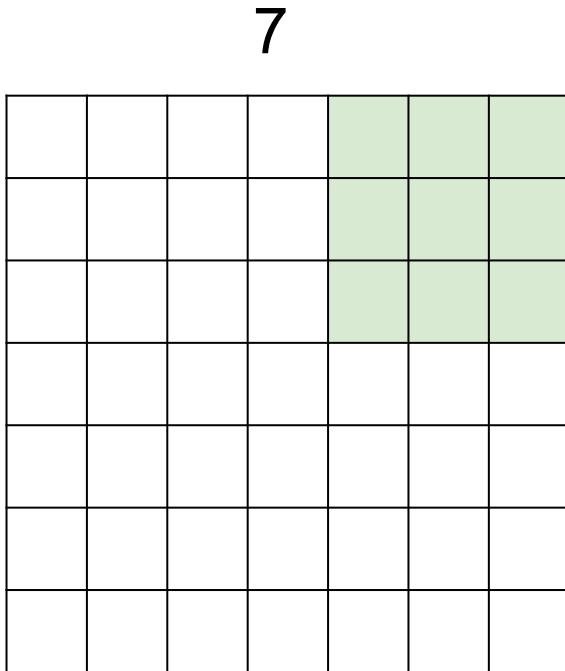
7



7

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

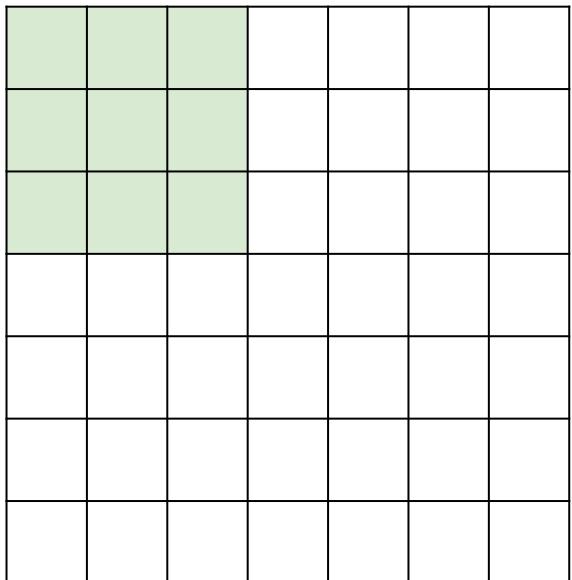
A closer look at spatial dimensions:



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

A closer look at spatial dimensions:

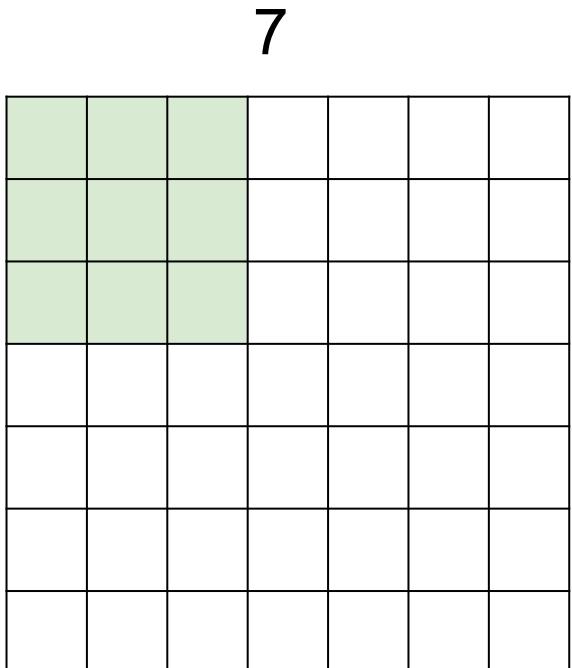
7



7

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

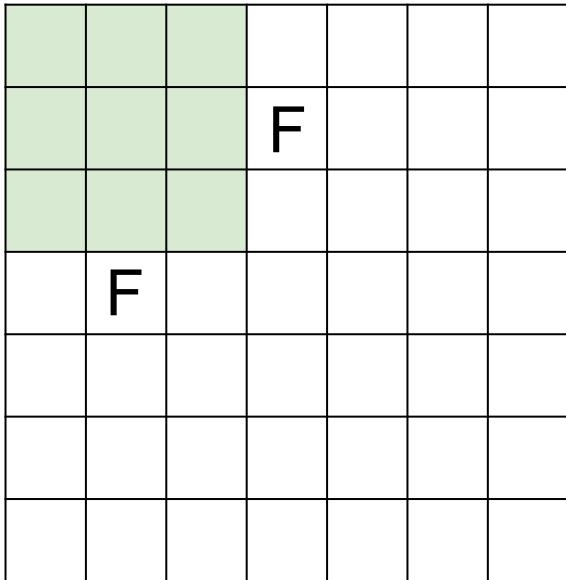
A closer look at spatial dimensions:



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

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

N



N = height/width of image (px)
F = height/width of filter (px)

N Output size (px):
(N - F) / stride + 1

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

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

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

stride 3 => $(7 - 3)/3 + 1 = 2.33$:\ No

In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

$$(N - F) / \text{stride} + 1$$

$$N = 7 + 2 * 1 = 9$$

$$F = 3$$

$$S = 1$$

$$(9 - 3) / 1 + 1 = 7$$

7x7 output!

In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

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. $F = 3 \Rightarrow$ zero pad with 1 zero

$F = 5 \Rightarrow$ zero pad with 2 zeros

$F = 7 \Rightarrow$ zero pad with 3 zeros

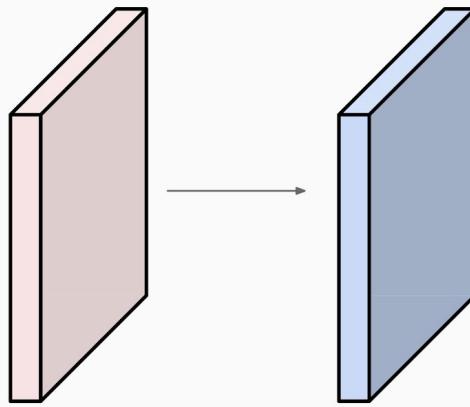
Padding and Striding

- Zero Padding:
 - Add a border of zeros around the original image to prevent the filter from overhanging the ends
- Stride:
 - The number of rows/columns to move the filter
 - Stride = 1:
 - measure at every position
 - high data density, low loss
 - Stride > 1:
 - More averaging
 - Stride < 1:
 - “Fractional Striding”: Expand, rather than condense the image

Padding Breakout

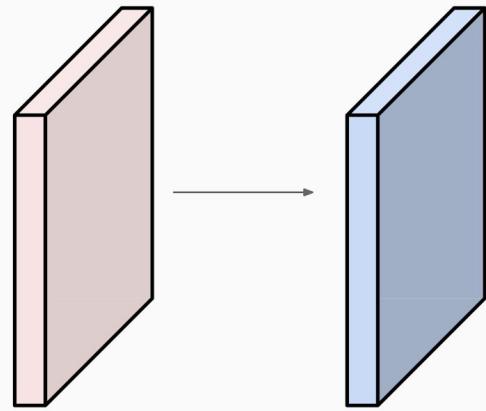
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



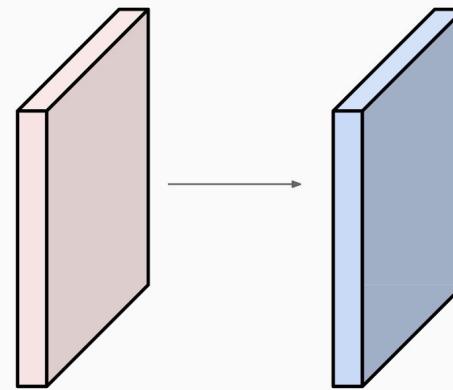
Padding Breakout

Input volume: **32x32x3**



Number of parameters (weights) breakout

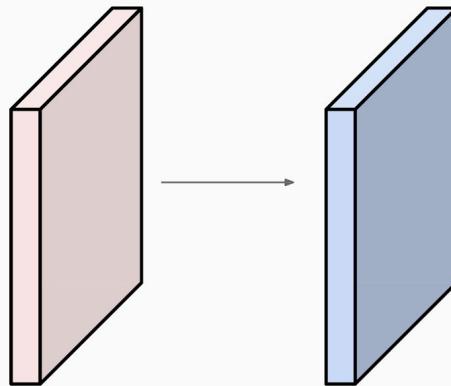
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Number of parameters (weights) breakout

The weights are the filter values.



Convolutional layer hyperparameters:

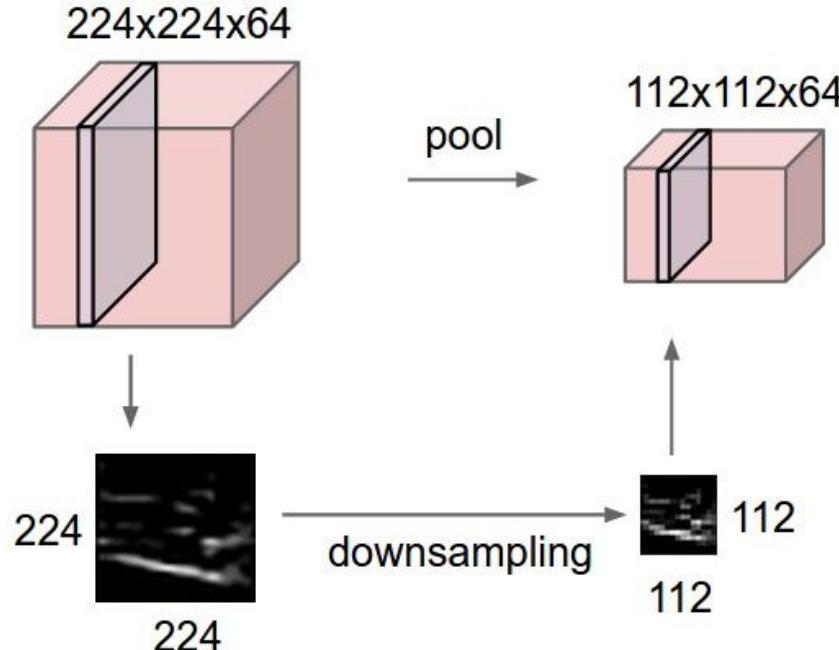
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

Common settings:

- $K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$
- $F = 3, S = 1, P = 1$
 - $F = 5, S = 1, P = 2$
 - $F = 5, S = 2, P = ? \text{ (whatever fits)}$
 - $F = 1, S = 1, P = 0$

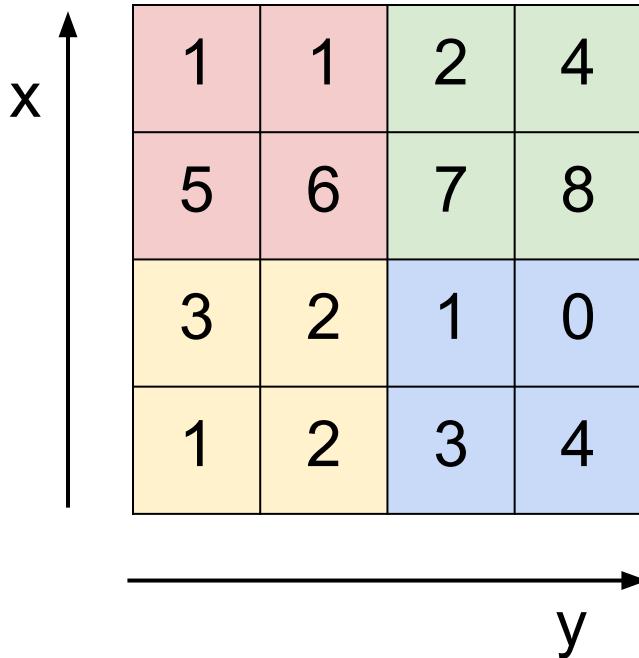
Pooling (subsampling) layer

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



MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2

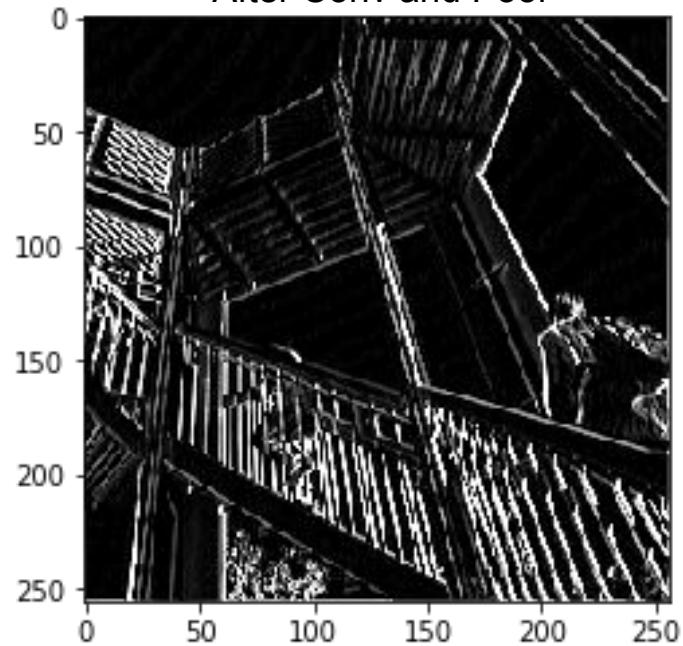
A 2x2 grid representing the output of the max pooling operation. It contains the maximum values from each 2x2 receptive field in the input tensor.

6	8
3	4

Input Image



After Conv and Pool



[Source](#)

Pooling layer hyperparameters:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
 - The type of pooling
- 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

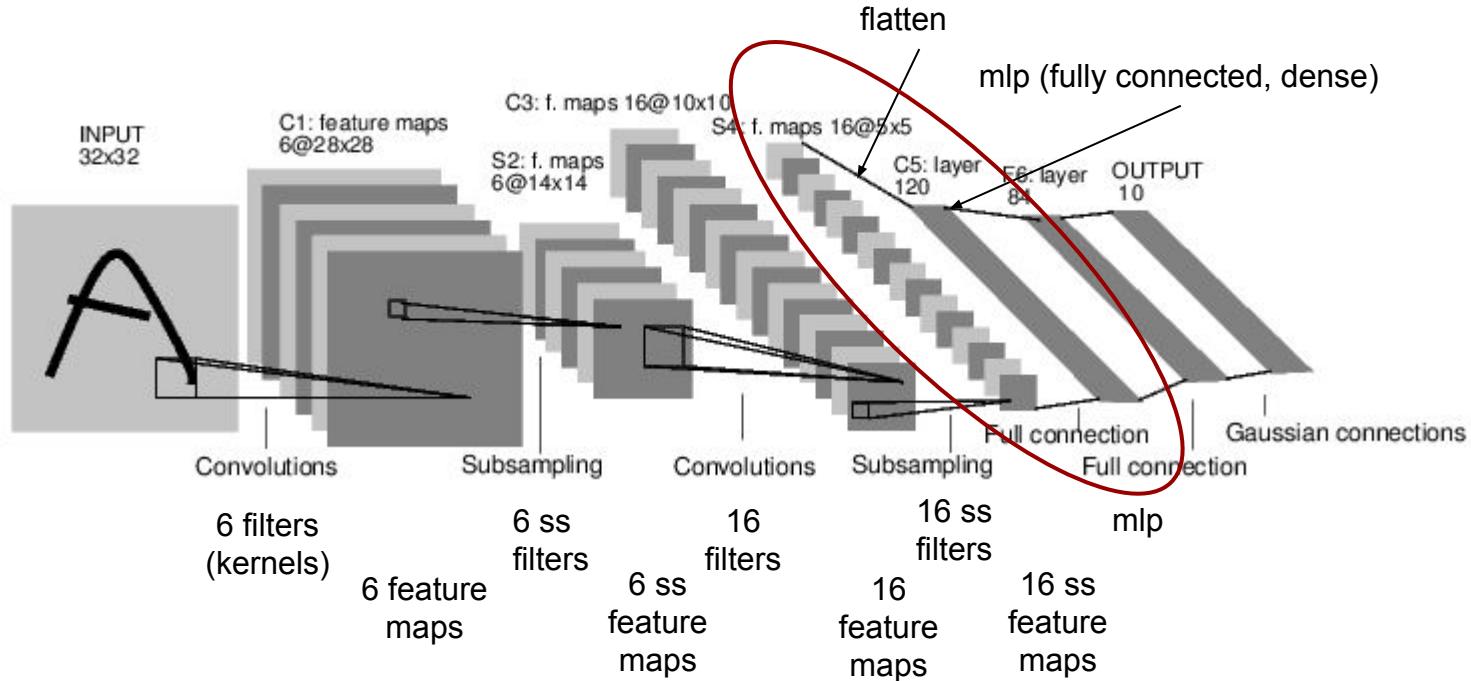
Common settings:

$F = 2, S = 2, \text{Max}$

$F = 3, S = 2, \text{Max}$

Fully Connected Layer

- flatten into conventional 1D array



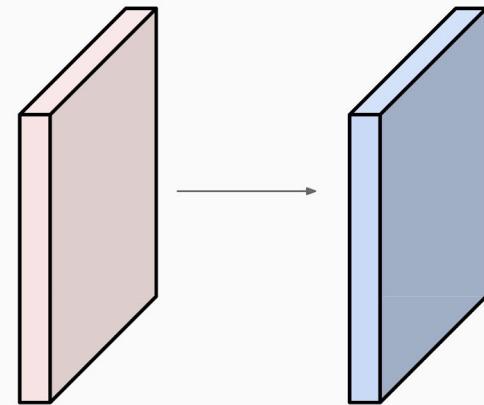
[LeNet-5, LeCun 1980]

Fully Connected Transition Breakout

Input volume: **8x8x64**

Flatten for input to dense layer

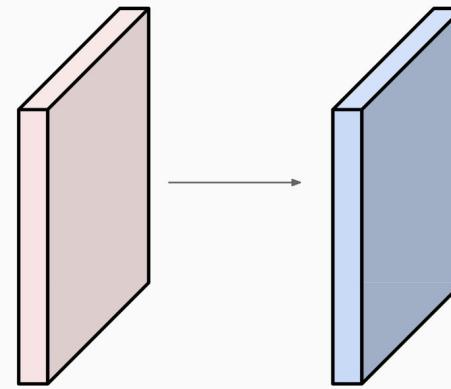
Output size: ?



Fully Connected Transition Breakout

Input volume: **8x8x64**

Flatten for input to dense layer



Flatten → Output

- Converts 2D Filters to 1D
- MLP Layers:
 - Densely Connected
 - Dropouts
- Output Depends on Problem Definition:
 - Regression:
 - Single Numeric Value (linear activation)
 - Multiple Numeric values (with linear activation)
 - Classification:
 - Binary (Dog vs. Cat):
 - Single Logit Value (0...1)
 - Multicategory: (Dog, Cat, Horse, Goat):
 - Softmax: Series of values summing to 1. (Hopefully one value very close to 1 and others close to zero)

Loss Functions

- Regression:
 - RMSE
 - MAE
- Classification:
 - Binary Cross-Entropy
 - Multi-Category Cross-Entropy

Keras Demo: CIFAR 10 dataset

[The CIFAR-10 dataset](#)

keras_example_cifar10.py

ConvNetJS demo: training on CIFAR-10

The Javascript demo:

<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

```

def define_model(num_classes, input_shape):
    Instantiating model → model = Sequential()
    Initial Input Layer → model.add(Conv2D(32, (3, 3), padding='same'
                                             input_shape=input_shape))
    Activation Function → model.add(Activation('relu'))
    Repeat another Conv layer → {model.add(Conv2D(32, (3, 3)))
                                  model.add(Activation('relu'))}
    Max Pool Layer → model.add(MaxPooling2D(pool_size=(2, 2)))
    Dropout → model.add(Dropout(0.25))

Drop out improves efficiency by throwing some of the neurons out in training, helps reduce overfitting
    → model.add(Conv2D(64, (3, 3), padding='same'))
    → model.add(Activation('relu'))
    → model.add(Conv2D(64, (3, 3)))
    → model.add(Activation('relu'))
    → model.add(MaxPooling2D(pool_size=(2, 2)))
    → model.add(Dropout(0.25))

    Unravels to a 1-D array → model.add(Flatten())
    A hidden layer in a typical MLP → model.add(Dense(512))
                                    → model.add(Activation('relu'))
                                    → model.add(Dropout(0.5))

    Output Layer → model.add(Dense(num_classes))
    Output Activation → model.add(Activation('softmax'))
    → return model

```

[CIFAR10.py](#)

Number of filters

Filter shape

Padding?

One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. [Source](#)

Original size of your image w/ number of channels

For categorical, how many classes do you have?

Typical for multi-class outcomes

```
def compile_model(model, optimizer):
    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    return model
```

Depends on your type of problem

```
def train_model(model, data_augmentation, batch_size, epochs,
                x_train, x_test, y_train, y_test):
```

```
    if not data_augmentation:
        print('Not using data augmentation.')
        model.fit(x_train, y_train,
                   batch_size=batch_size,
                   epochs=epochs,
                   validation_data=(x_test, y_test),
                   shuffle=True)
```

Many to choose from, a hyper parameter to tune: [source](#)

Several options depending on your problem: [source](#)

Recall what your batch size should be, this depends on how often you update your weights (filters)

Another number to tune, stop too soon and you might miss something

Resources

- [Stanford CNN course materials](#)
- [pyimagesearch](#) blog
- [Deep Learning with Python](#) book (not free)
- [MachineLearningMastery](#) [Image Augmentation using Keras](#) blog

Solutions Slides

Convolution Breakout

1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

Input Image
(5x5x1)

*

1	1	1
1	4	1
1	1	1

Filter
(3x3x1)

=

9	11	12
3	9	11
1	3	9

Result
(3x3x1)

Padding Breakout

Input volume: **32x32x3**

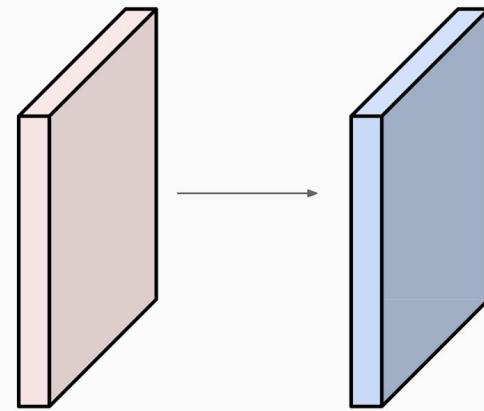
10 5x5 filters with stride **1**, pad **2**

Output volume size:

$$(32+2*2-5)/1+1 = 32 \text{ spatially, so}$$

32x32x10

$(N - F) / \text{stride} + 1$ dimension for each activation map

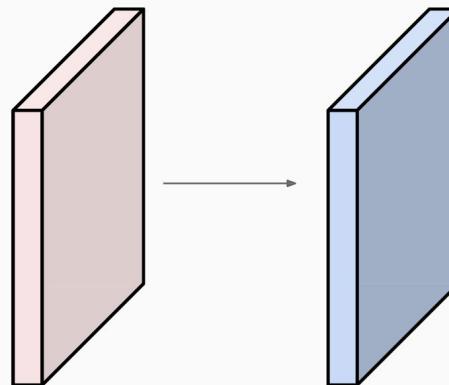


Number of parameters (weights) breakout

The weights are the filter values.

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)

$$\Rightarrow 76 * 10 = 760$$

Fully Connected Transition Breakout

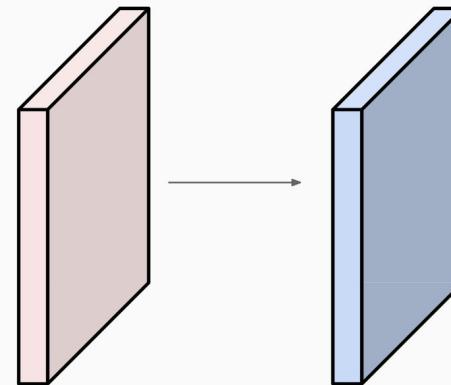
Input volume: **8x8x64**

Flatten for input to dense layer

Output volume size:

8*8*64 = 4096 (it's that simple!)

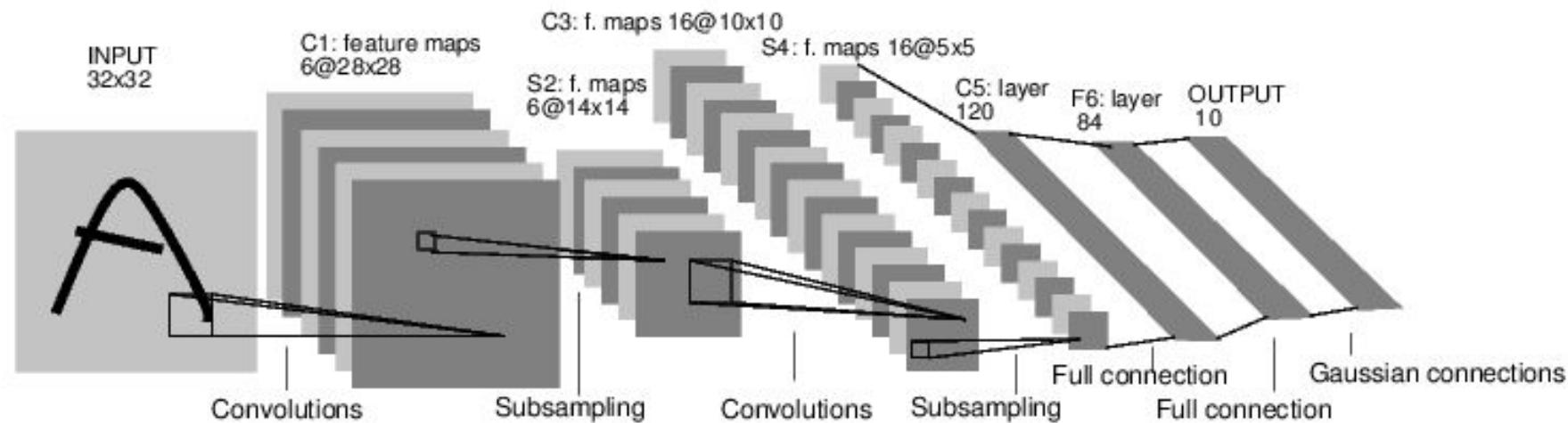
Note that you don't have to map to a flat layer 1-to-1.



Appendix (Extra Slides)

Case Study: LeNet-5 (End of DSI lecture - for ref.)

[LeCun et al., 1998]

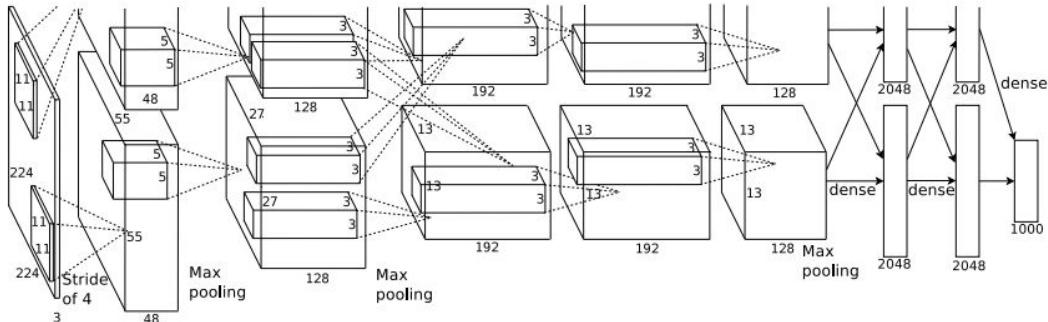


Conv filters were 5×5 , applied at stride 1

Subsampling (Pooling) layers were 2×2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

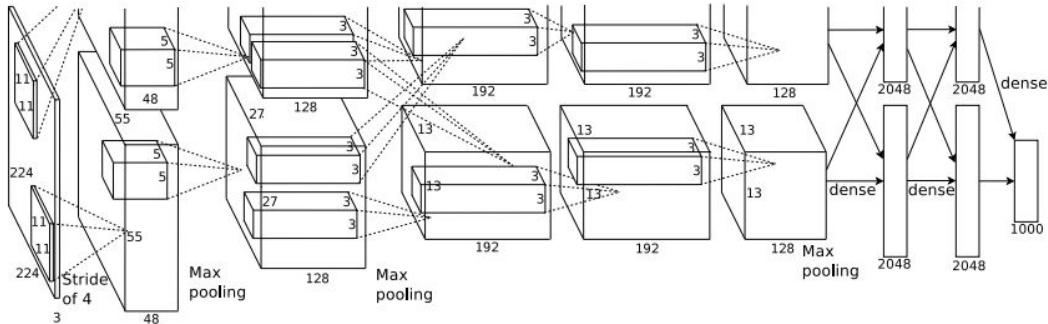
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

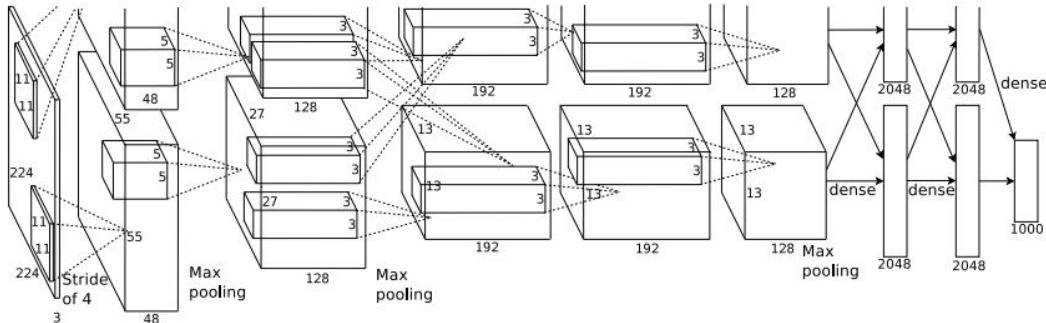
=>

Output volume **[55x55x96]**

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

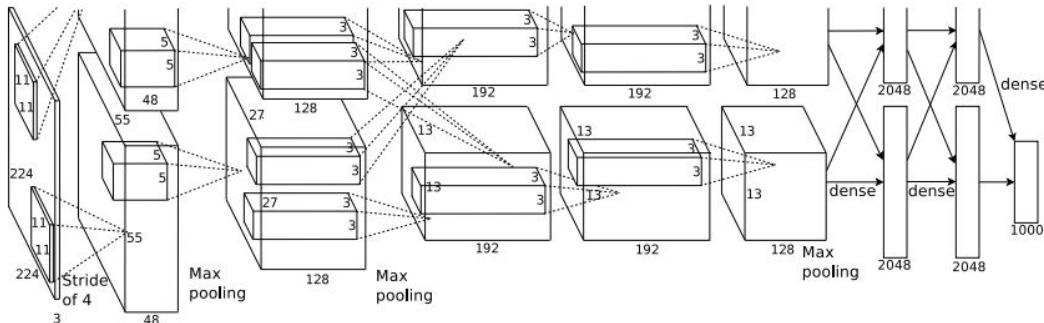
=>

Output volume **[55x55x96]**

Parameters: $(11 \times 11 \times 3) \times 96 = 35K$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

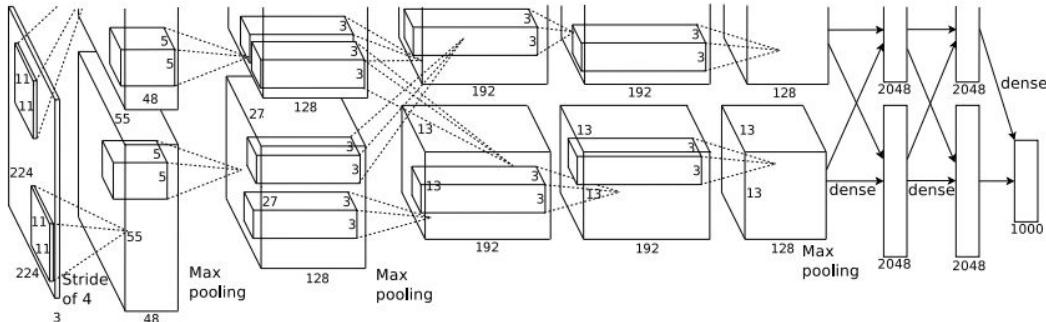
After CONV1: 55x55x96

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

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

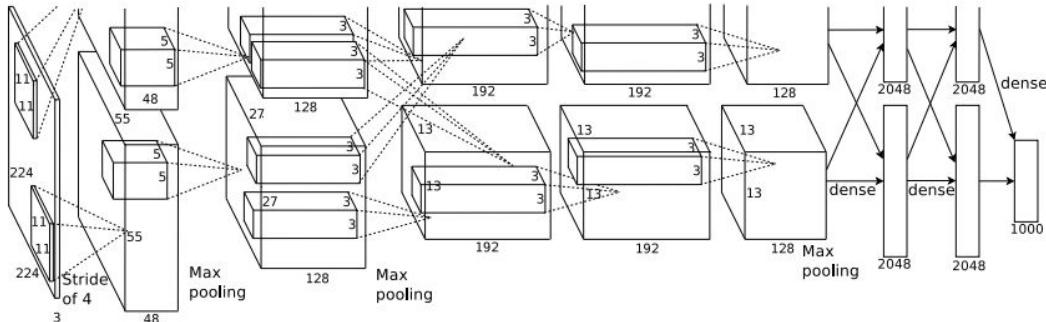
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

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

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!

Case Study: AlexNet

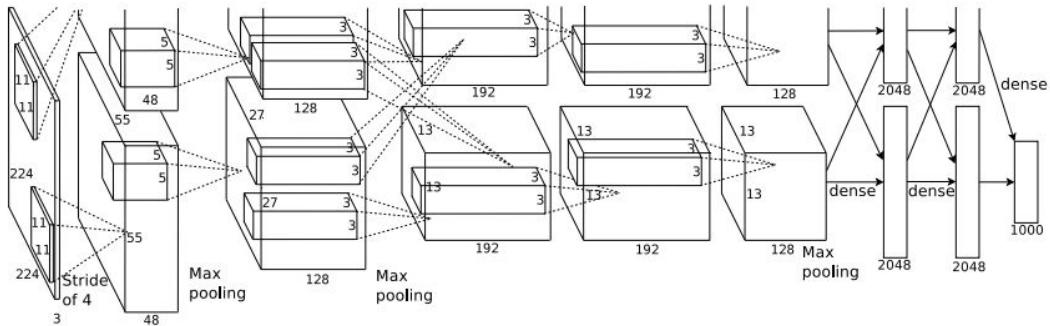
[Krizhevsky et al. 2012]

Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...



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

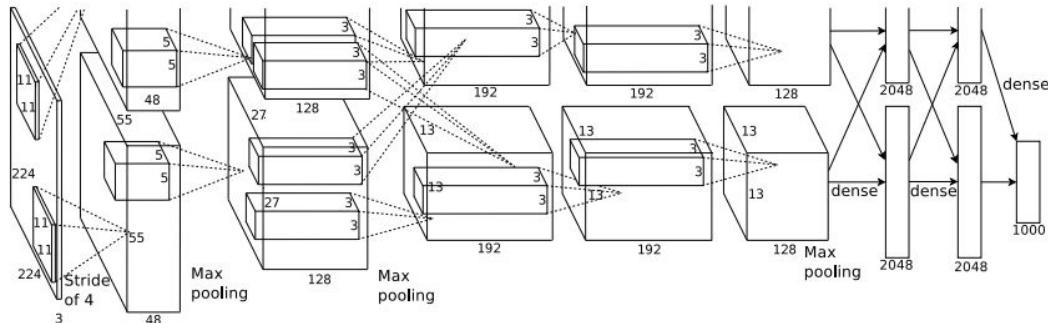
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



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

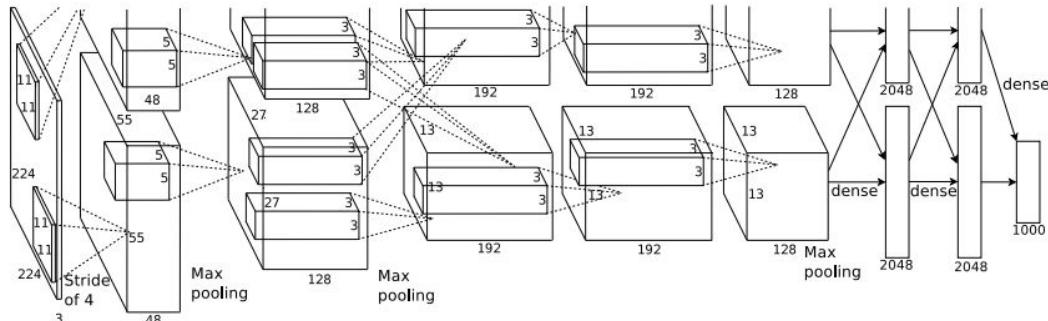
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

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

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

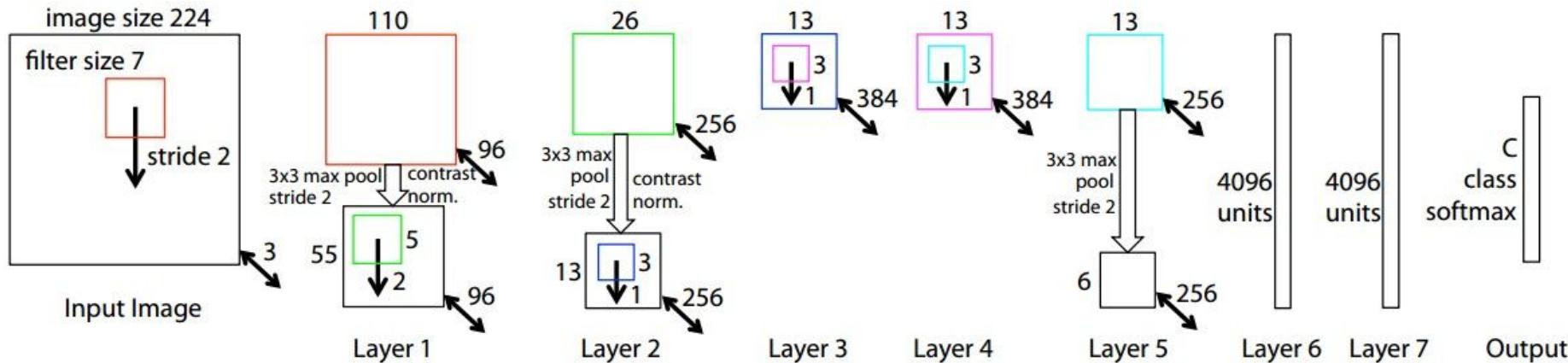


Details/Retrospectives:

- 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%

Case Study: ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error

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

Table 2: Number of parameters (in millions).

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

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

ConvNet Configuration			
B	C	D	E
13 weight layers	16 weight layers	16 weight layers	19 weight layers
put (224 × 224 RGB image)			
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
maxpool			
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
	conv1-256	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
maxpool			
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
	conv1-512	conv3-512	conv3-512
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0

FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

TOTAL memory: $24M * 4 \text{ bytes} \approx 93\text{MB} / \text{image}$ (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	19
13 weight layers	16 weight layers	16 weight layers	
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
conv3-64	conv3-64	conv3-64	cc
		maxpool	
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
		maxpool	
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	conv3-256	co
		maxpool	
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	conv3-512	co
		maxpool	
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	conv3-512	co
		maxpool	
FC-4096			
FC-4096			
FC-1000			
soft-max			

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: $(3*3*64)*64 = 36,864$

Note:

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: $(3*3*128)*128 = 147,456$

Most memory is in early CONV

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: 56*56*256=800K params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: 56*56*256=800K params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: 28*28*512=400K params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: 28*28*512=400K params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: 14*14*512=100K params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: 14*14*512=100K params: $(3*3*512)*512 = 2,359,296$

Most params are in late FC

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = \mathbf{102,760,448}$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

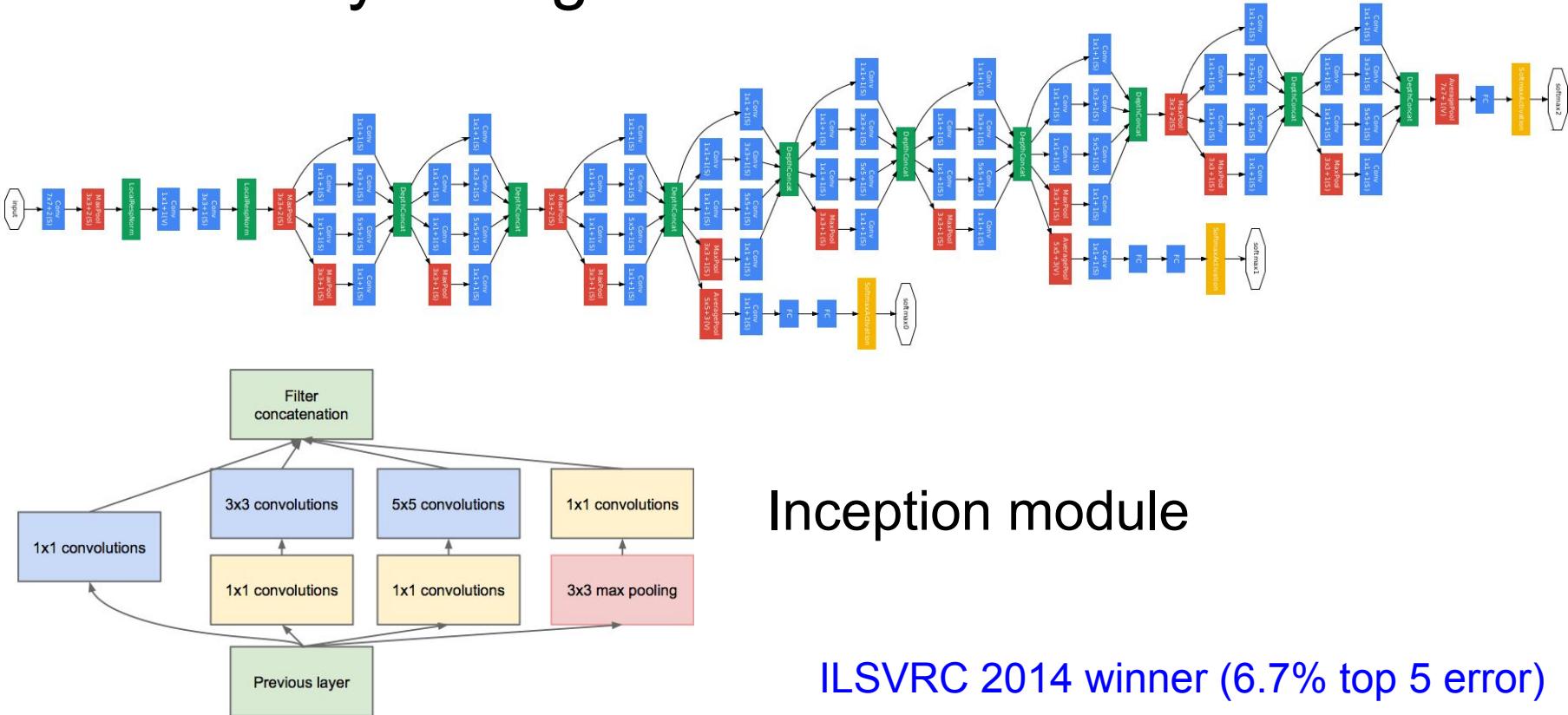
FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

TOTAL memory: 24M * 4 bytes \approx 93MB / image (only forward! ~ 2 for bwd)

TOTAL params: 138M parameters

Case Study: GoogLeNet

[Szegedy et al., 2014]



Case Study: GoogLeNet

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								

Fun features:

- Only 5 million params!
(Removes FC layers completely)

Compared to AlexNet:

- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



MSRA @ ILSVRC & COCO 2015 Competitions

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

- ImageNet Classification: “Ultra-deep” (quote Yann) **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

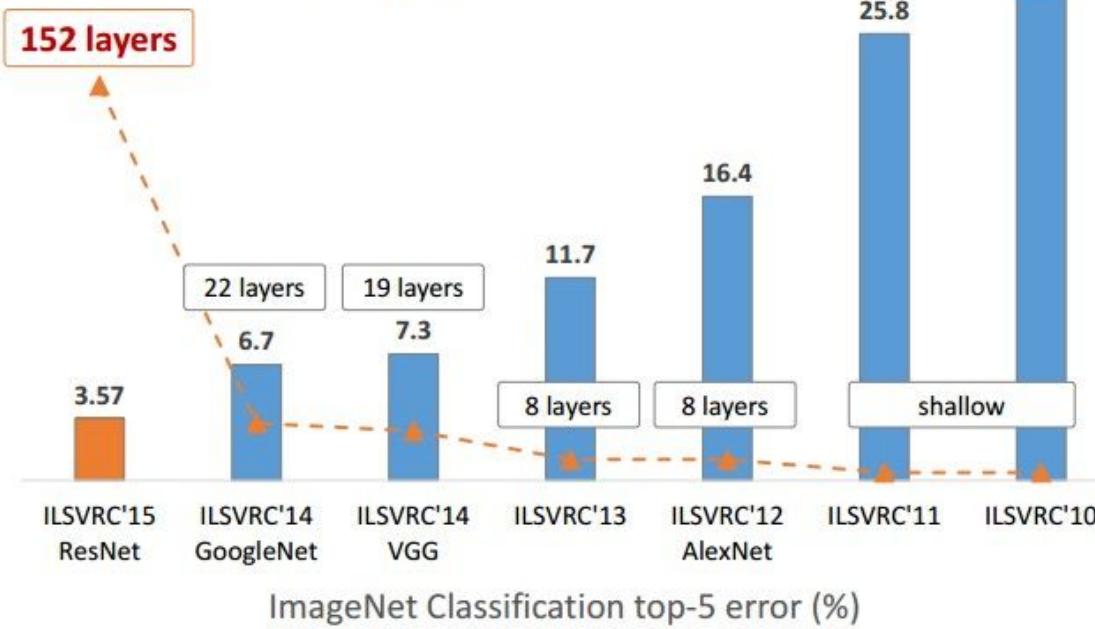
*improvements are relative numbers



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

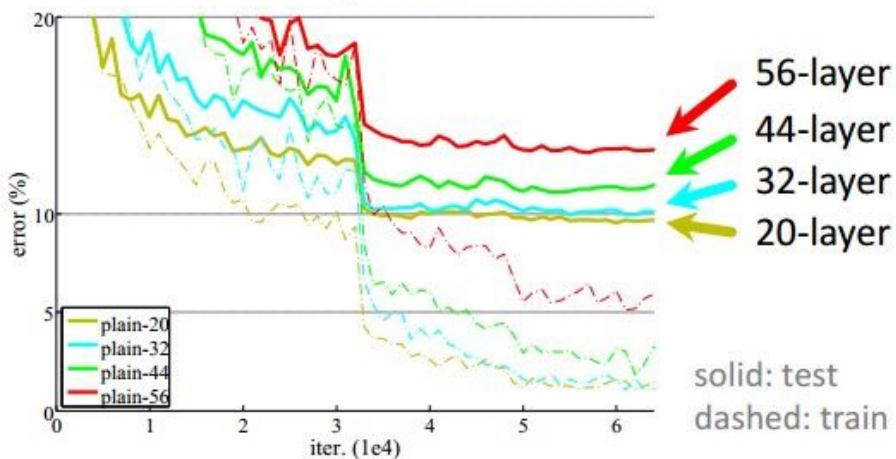
Slide from Kaiming He's recent presentation <https://www.youtube.com/watch?v=1PGLj-uKT1w>

Revolution of Depth

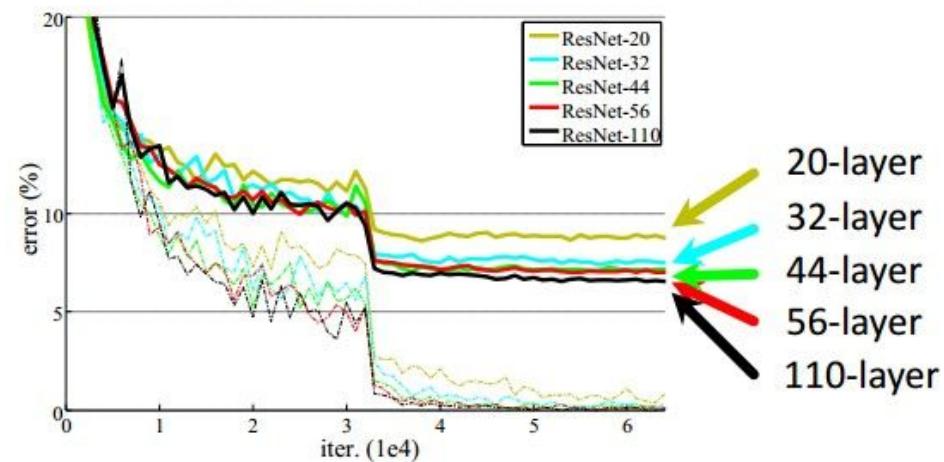


CIFAR-10 experiments

CIFAR-10 plain nets



CIFAR-10 ResNets



Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



ResNet, **152 layers**
(ILSVRC 2015)



2-3 weeks of training
on 8 GPU machine

at runtime: faster
than a VGGNet!
(even though it has
8x more layers)

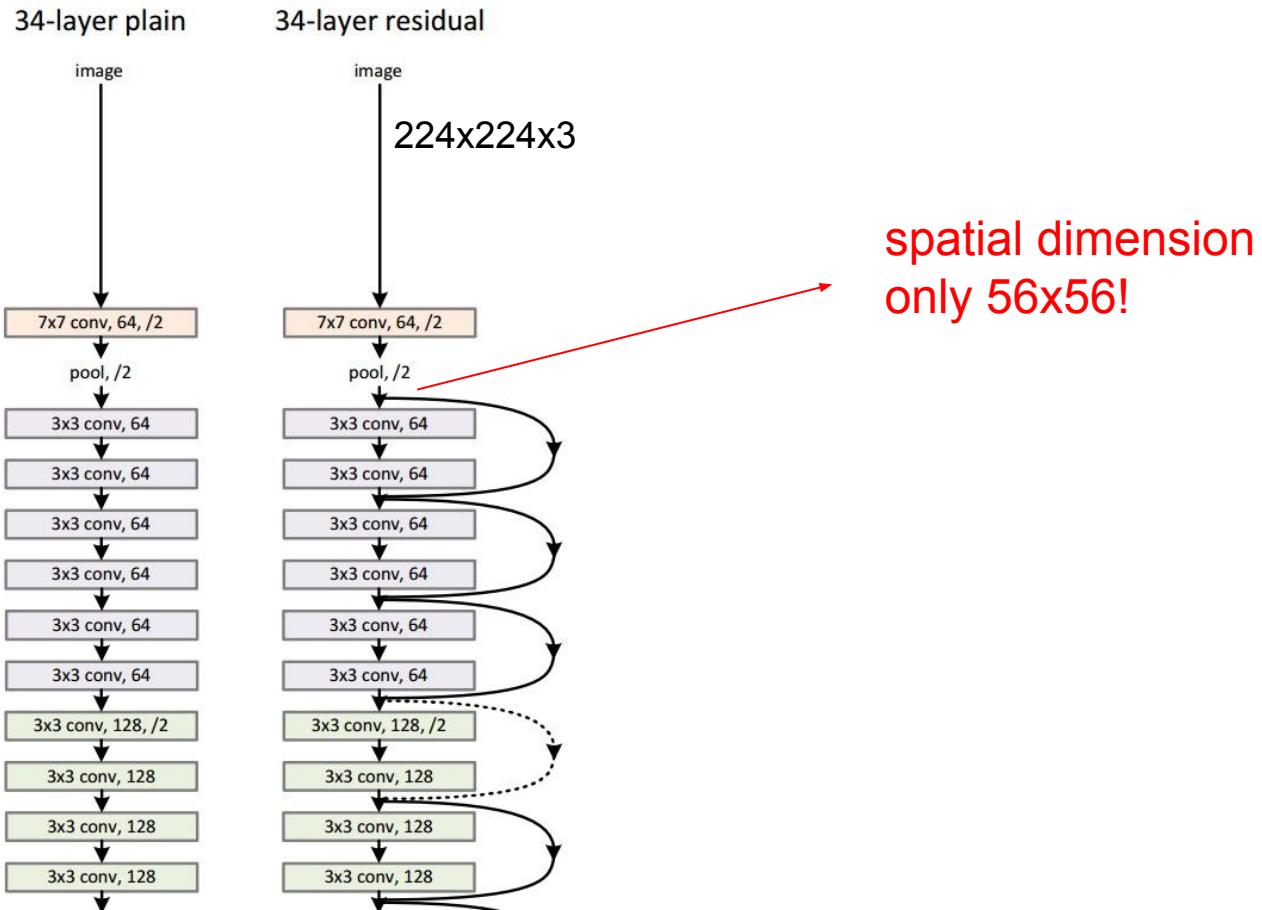


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

Case Study: ResNet

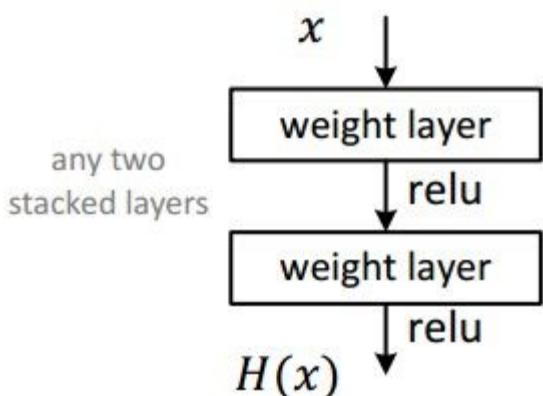
[He et al., 2015]



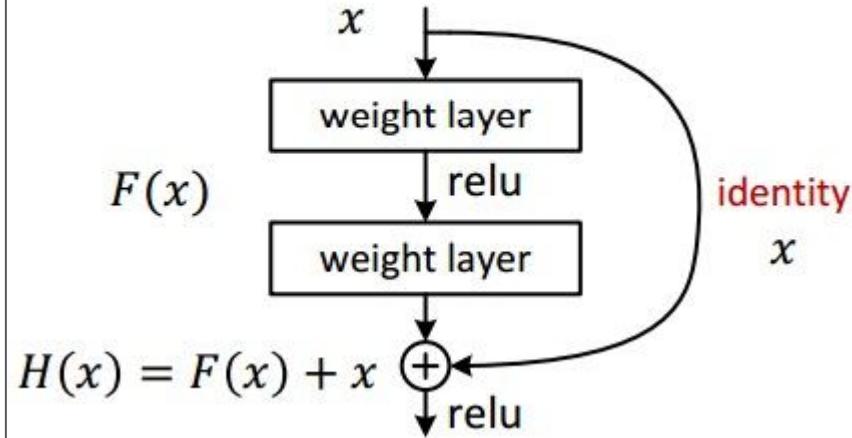
Case Study: ResNet

[He et al., 2015]

- Plain net



- Residual net



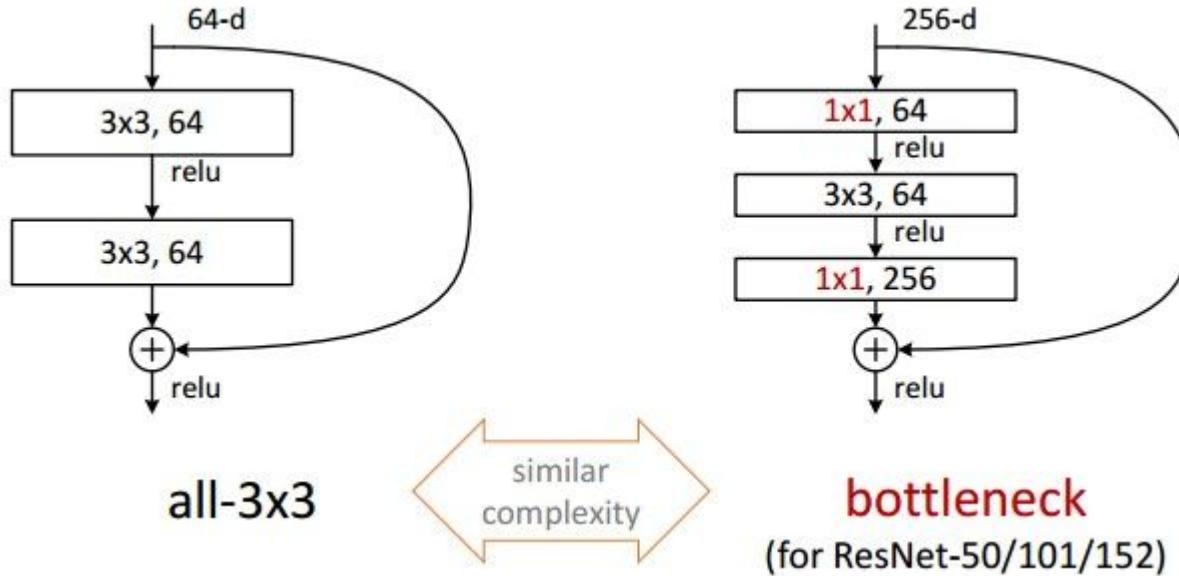
Case Study: ResNet

[He et al., 2015]

- 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

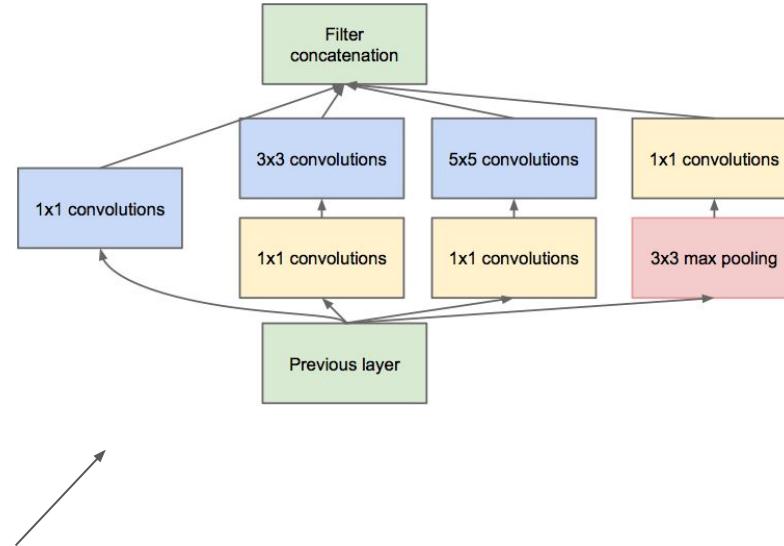
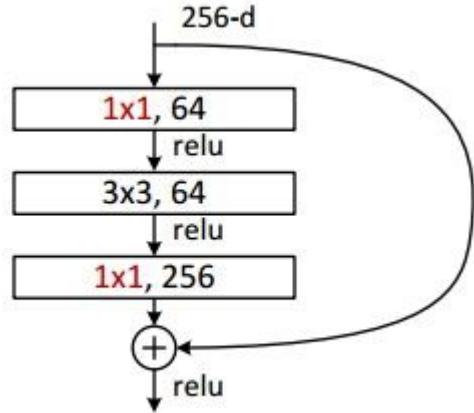
Case Study: ResNet

[He et al., 2015]



Case Study: ResNet

[He et al., 2015]

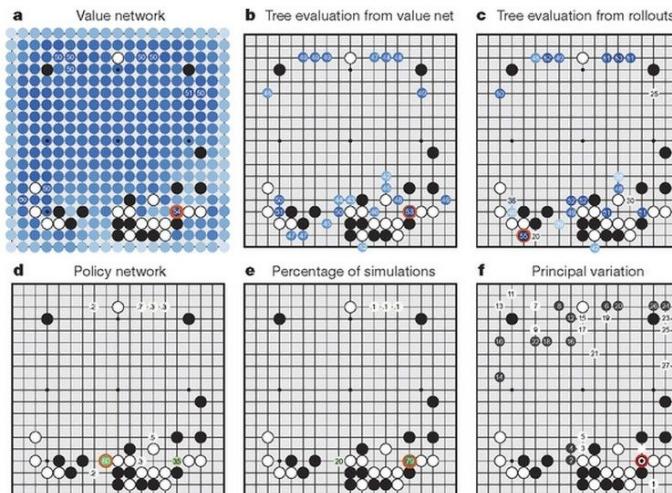


(this trick is also used in GoogLeNet)

Case Study: ResNet [He et al., 2015]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Case Study Bonus: DeepMind's AlphaGo



The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; [Fig. 2b](#) and [Extended Data Table 3](#) additionally show the results of training with $k = 128, 256$ and 384 filters.

policy network:

[$19 \times 19 \times 48$] Input

CONV1: 192 5×5 filters , stride 1, pad 2 => [$19 \times 19 \times 192$]

CONV2..12: 192 3×3 filters, stride 1, pad 1 => [$19 \times 19 \times 192$]

CONV: 1 1×1 filter, stride 1, pad 0 => [19×19] (*probability map of promising moves*)

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
 $[(\text{CONV-RELU})^* \text{N-POOL?}]^* \text{M-(FC-RELU)}^* \text{K,SOFTMAX}$
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet challenge this paradigm