Convolutional Neural Networks

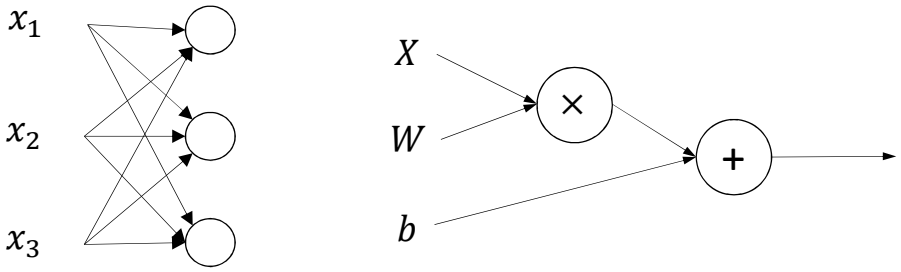
Kai-Lung Hua (花凱龍)

(Deep) Neural Networks

We can chain / stack more of these blocks (layers) to make "taller" (deeper) models.

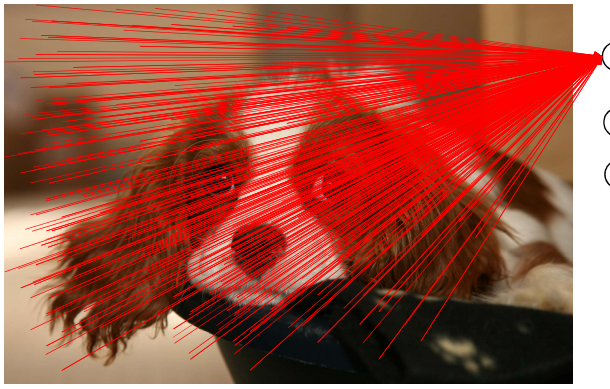
So far we only know of one type of layer

Fully Connected / Dense / Linear / Affine Layer



3 of 127

Limitations of Fully Connected layers



Suppose we have a $200 \times 200 \times 3$ RGB image

How many connections / parameters will a single fully connected layer with 1000 hidden units have?

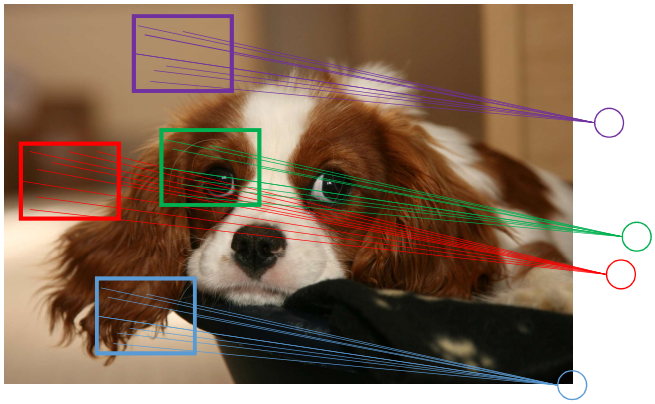
~120,000,000 parameters

We will not have enough data nor computational power to estimate that many parameters.

Curse of dimensionality!

4 of 127

Locally Connected layers



We can make one hidden unit fully connected to a small patch

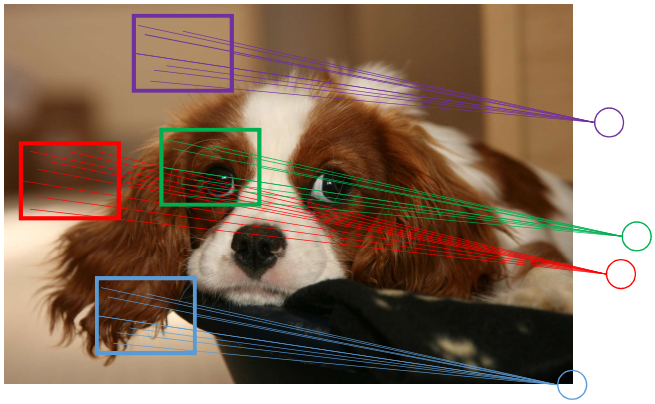
So now if we have 1000 hidden units and each unit corresponds to a patch size of 10×10 how many parameters will we have to learn?

100,000 parameters!

This is much smaller and much more manageable!

5 of 127

Locally Connected layers



But this model doesn't quite work well in practice.

Can you think of what are the limitations / assumptions that this model have?

This model only works well if the input image is registered (e.g. face recognition)

Is there some structure that we can still exploit?

6 of 127

(Digression) Cross-Correlation

Let f be an image and h be the kernel / filter. The cross-correlation of f with h is g, denoted as $g = f \star h$. (Also denoted as $f \otimes h$ in some literature.)

$$g[i,j] = (f \star h)[x,y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[x+i, y+j]$$

Elementwise multiplication and sum of a filter and the signal (image)

7 of 127

Slide Credit: Ranzato

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[.,.]$

8 of 127

Slide Credit: S. Seitz

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[.,.]$

	0	10							

9 of 127

Slide Credit: S. Seitz

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[.,.]$

	0	10	20						

10 of 127

Slide Credit: S. Seitz

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[.,.]$

	0	10	20	30					

11 of 127

Slide Credit: S. Seitz

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$g[.,.]$

	0	10	20	30	30				

12 of 127

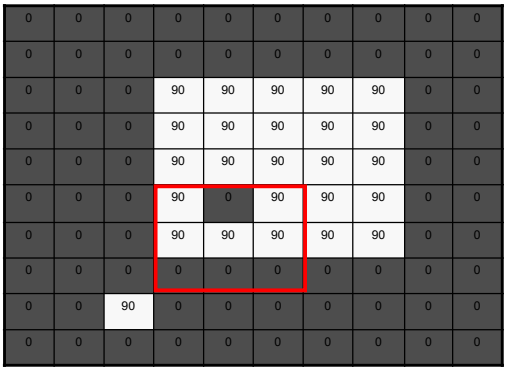
Slide Credit: S. Seitz

(Digression) Image filtering

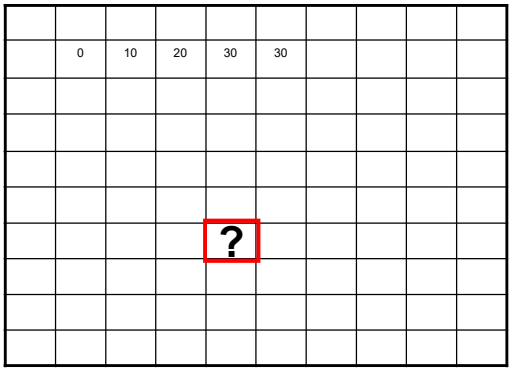
$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$g[.,.]$



13 of 127

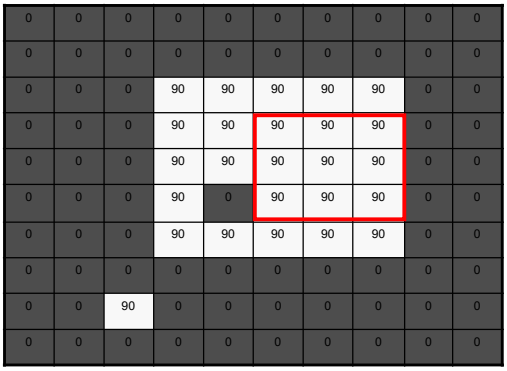
Slide Credit: S. Seitz

(Digression) Image filtering

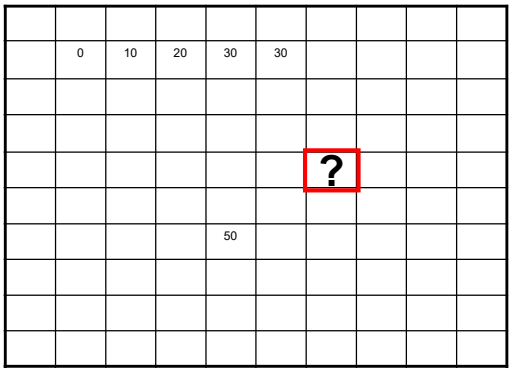
$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$g[.,.]$



14 of 127

Slide Credit: S. Seitz

(Digression) Image filtering

$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

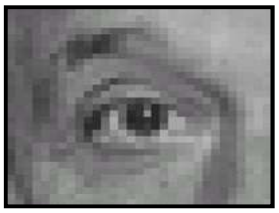
$g[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

15 of 127

Slide Credit: S. Seitz

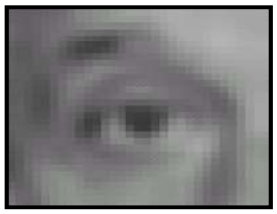
Image Filtering



$\star$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$=$



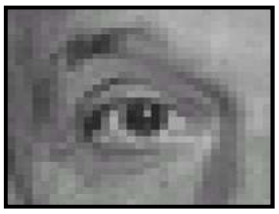
Original

Blur
(with a mean filter)

16 of 127

Slide Credit: D. Lowe

Image Filtering



Original

$\star$

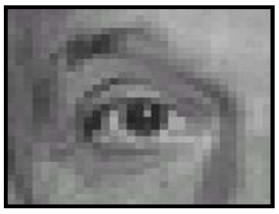
0	0	0
0	1	0
0	0	0

?

17 of 127

Slide Credit: D. Lowe

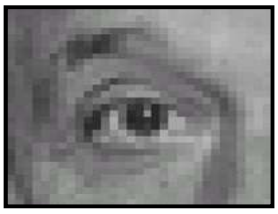
Image Filtering



Original

$\star$

0	0	0
0	1	0
0	0	0

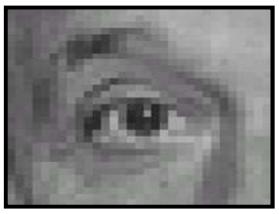


Filtered
(no change)

18 of 127

Slide Credit: D. Lowe

Image Filtering



Original

$\star$

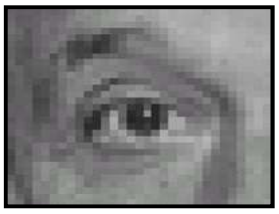
0	0	0
0	0	1
0	0	0

?

19 of 127

Slide Credit: D. Lowe

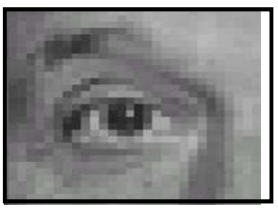
Image Filtering



Original

$\star$

0	0	0
0	0	1
0	0	0

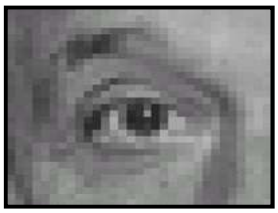


Shifted left
By 1 pixel

20 of 127

Slide Credit: D. Lowe

Image Filtering

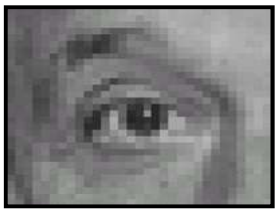


$$\star \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right] \quad ?$$

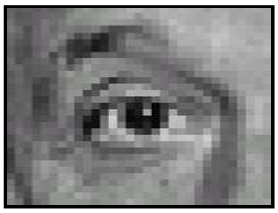
Original

Slide Credit: D. Lowe

Image Filtering



$$\star \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right]$$



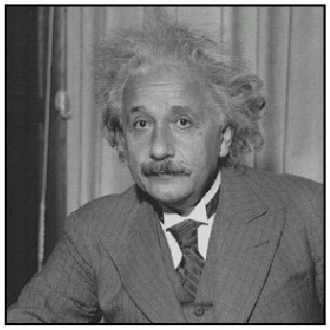
Original

Sharpening filter

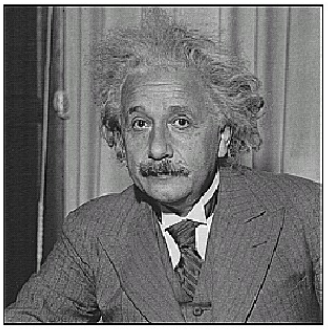
- Accentuates differences with local average
- Also known as Laplacian

Slide Credit: D. Lowe

Image Filtering



before

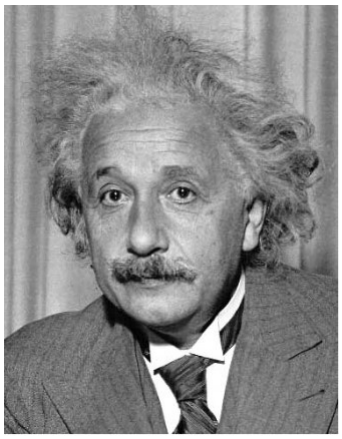


after

23 of 127

Slide Credit: D. Lowe

Image Filtering



★

1	0	-1
2	0	-2
1	0	-1

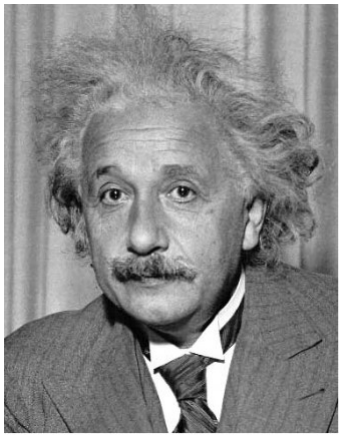
Sobel Filter

Vertical Edges

24 of 127

Slide Credit: D. Lowe

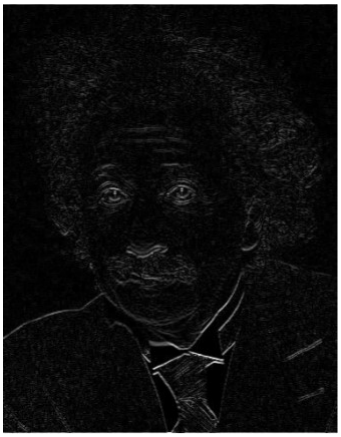
Image Filtering



$\star$

1	2	1
0	0	0
-1	-2	-1

Sobel Filter



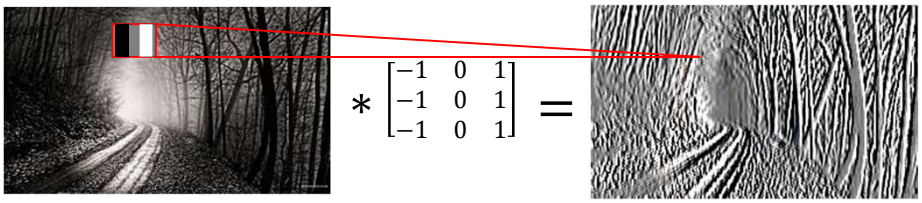
Horizontal Edges

25 of 127

Slide Credit: D. Lowe

Convolution

Let f be an image and h be the kernel / filter. The convolution of f with h is g, denoted as $g = f * h$



$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

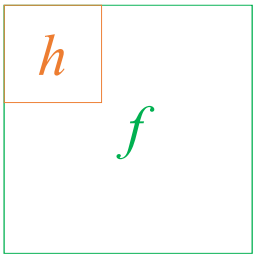
$$g[i,j] = f[x,y] * h[x,y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[x-i, y-j]$$

26 of 127

Slide Credit: Ranzato

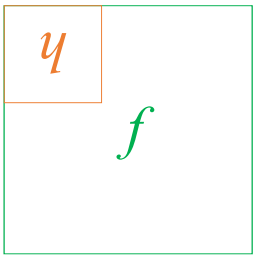
Cross-correlation

$$g[i,j] = f[x,y] \star h[x,y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[x+i, y+j]$$



Convolution

$$g[i,j] = f[x,y] * h[x,y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[x-i, y-j]$$



The only difference of convolution with cross-correlation is that the kernel is "flipped".

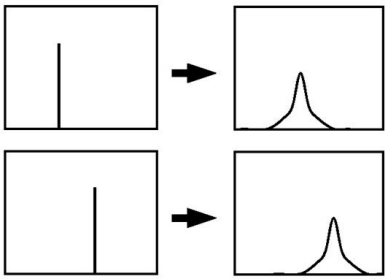
27 of 127

Properties of Convolution

Shift-invariance

same behavior regardless of pixel location

$$\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$$



Linearity

$$\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$$

Theoretical result: any linear shift-invariant operator can be represented as a convolution

28 of 127

Slide Credit: S. Lazebnik

Properties of Convolution

Commutative: $a * b = b * a$
- Conceptually no difference between filter and signal

Associative: $a * (b * c) = (a * b) * c$
- Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

Distributes over addition: $a * (b + c) = (a * b) + (a * c)$

Scalars factor out: $ka * b = a * kb = k(a * b)$

Identity: unit impulse $e = [\dots, 0, 0, 1, 0, 0, \dots]$, $a * e = a$

29 of 127

Slide Credit: S. Lazebnik

Some Applications of Image Filtering

Image enhancement
- Denoise
- Smooth
- Contrast
- Brightness

Extract information / features from images
- Texture
- Edges
- Distinctive points

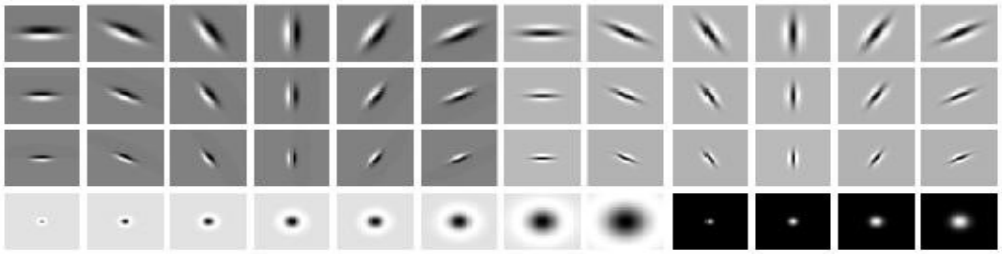
Detect Patterns
- Template Matching

30 of 127

Slide Credit: D. Hoiem

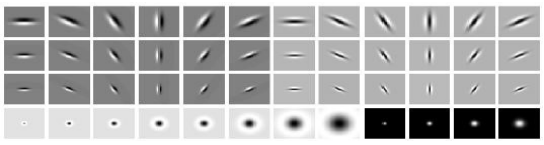
In practice we use many filters to capture more features

LM Filter Bank

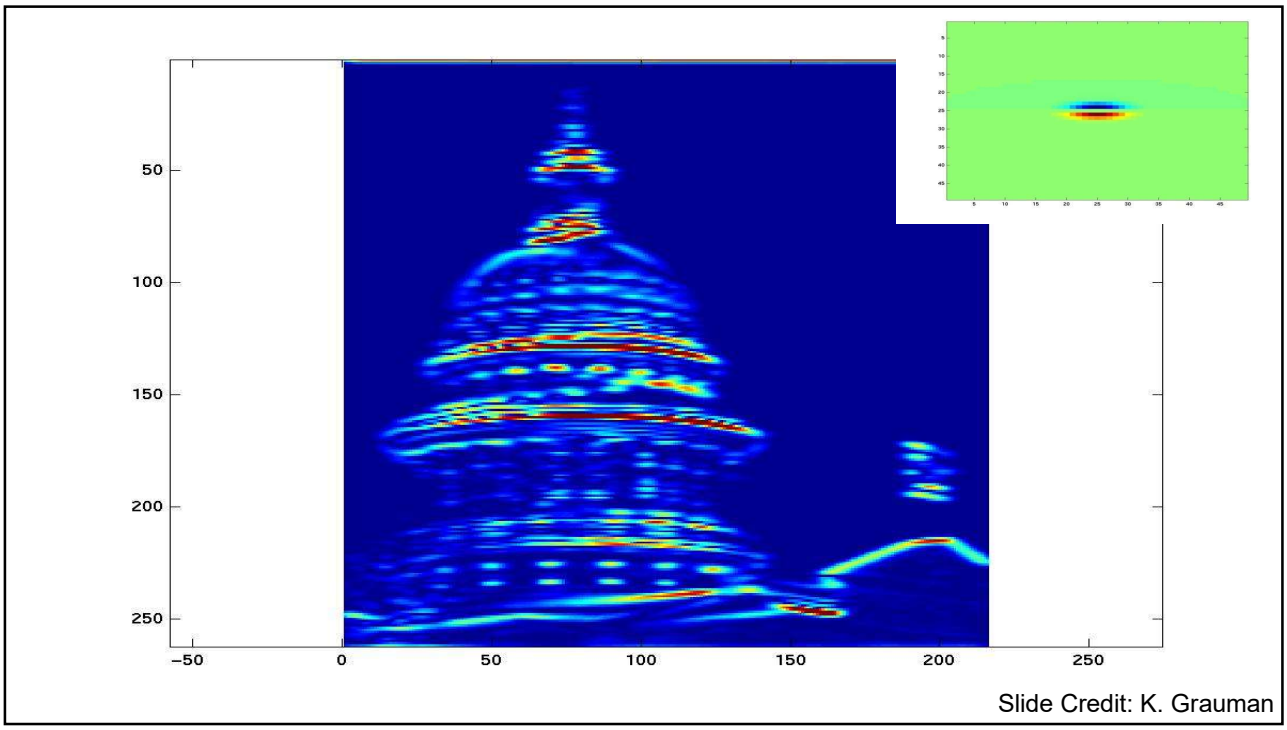


Code for filter banks: *www.robots.ox.ac.uk/~vgg/research/texclass/filters.html*

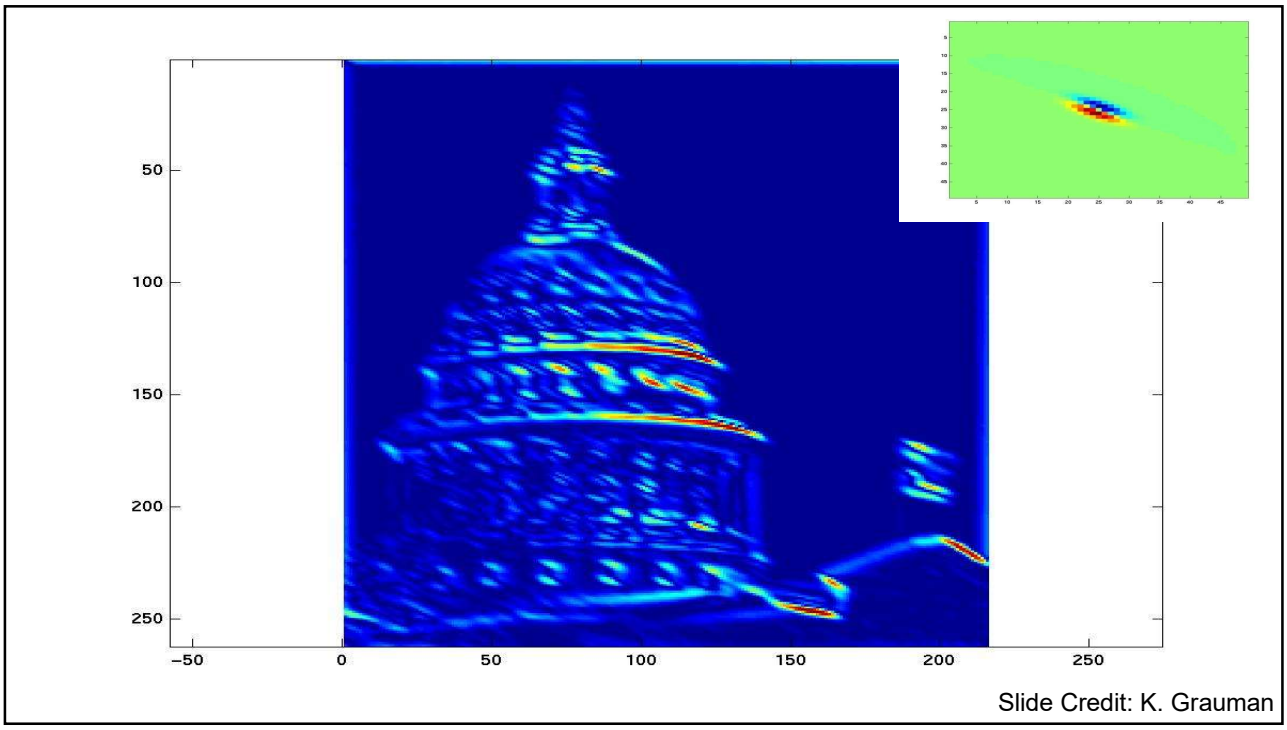
31 of 127



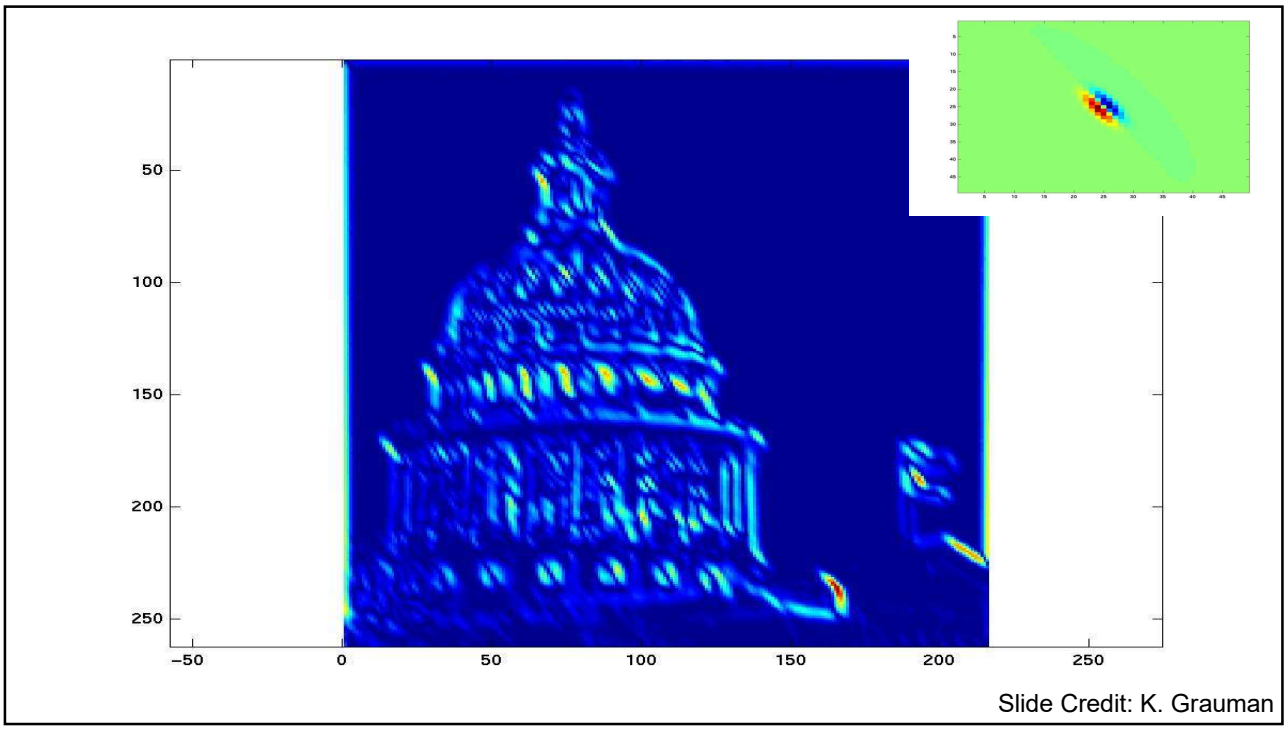
Slide Credit: K. Grauman



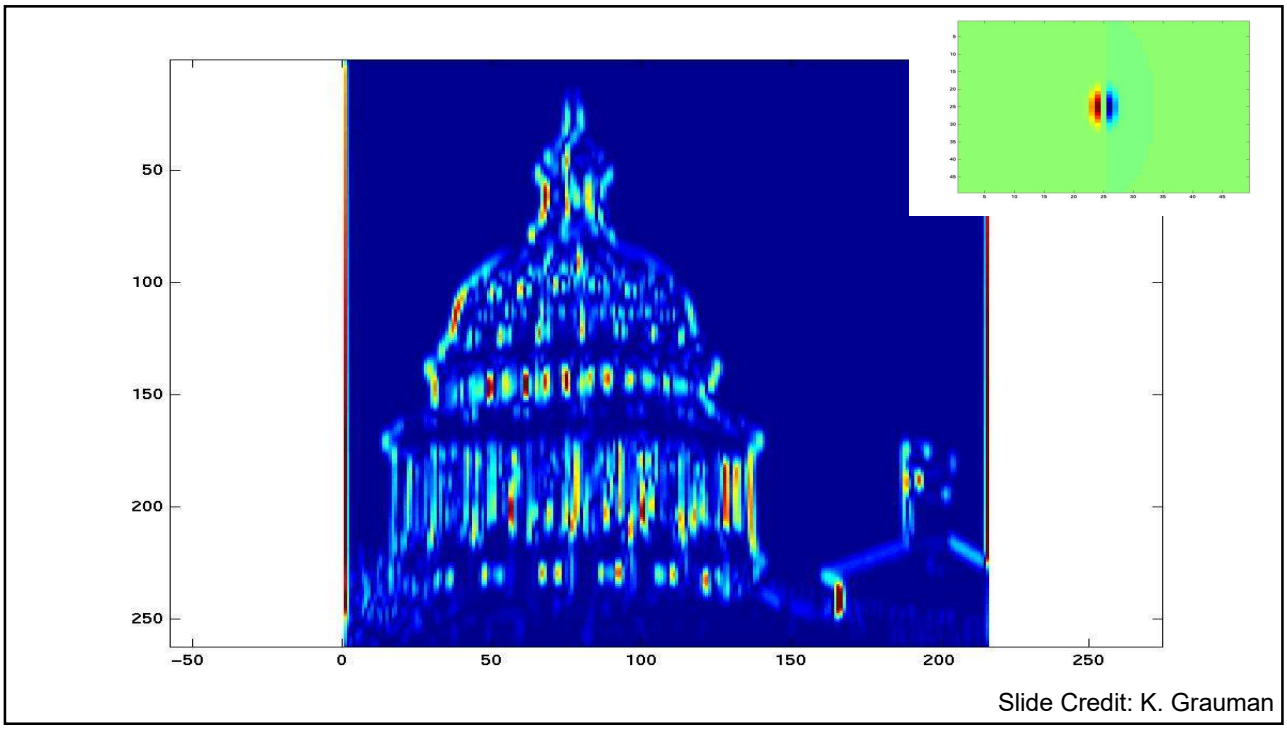
Slide Credit: K. Grauman



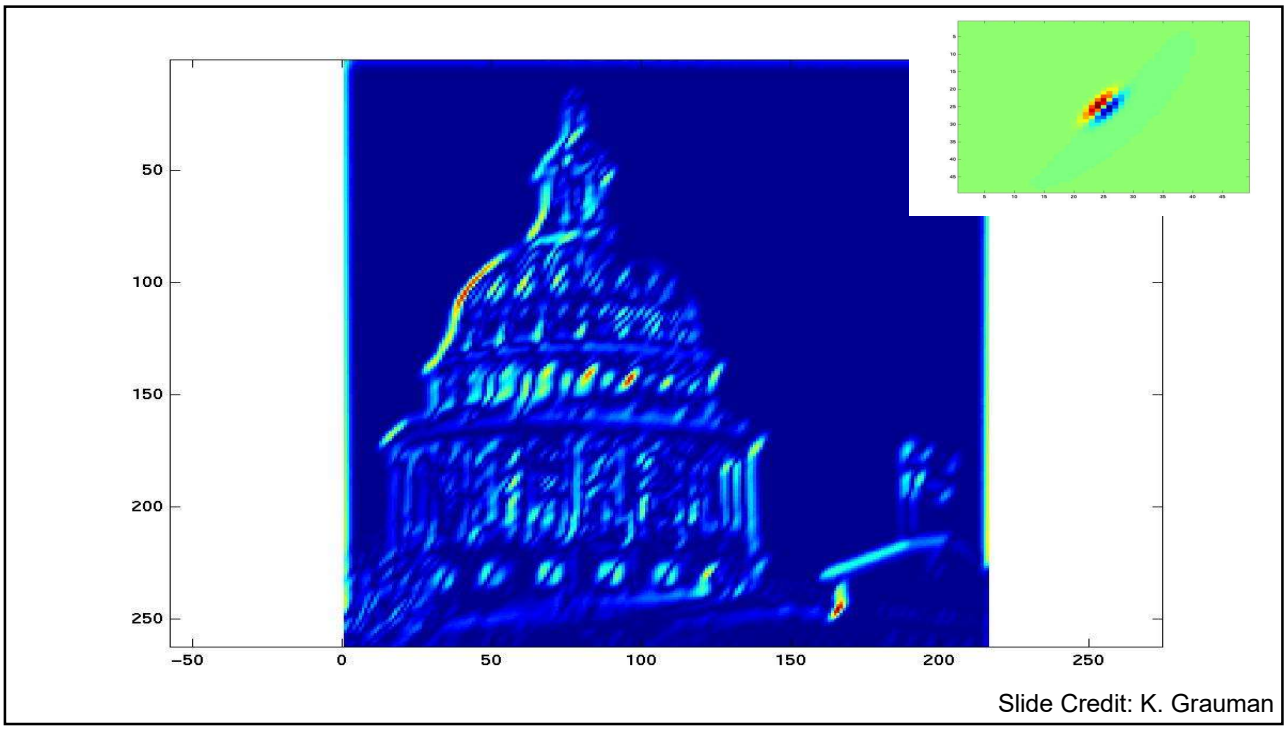
Slide Credit: K. Grauman



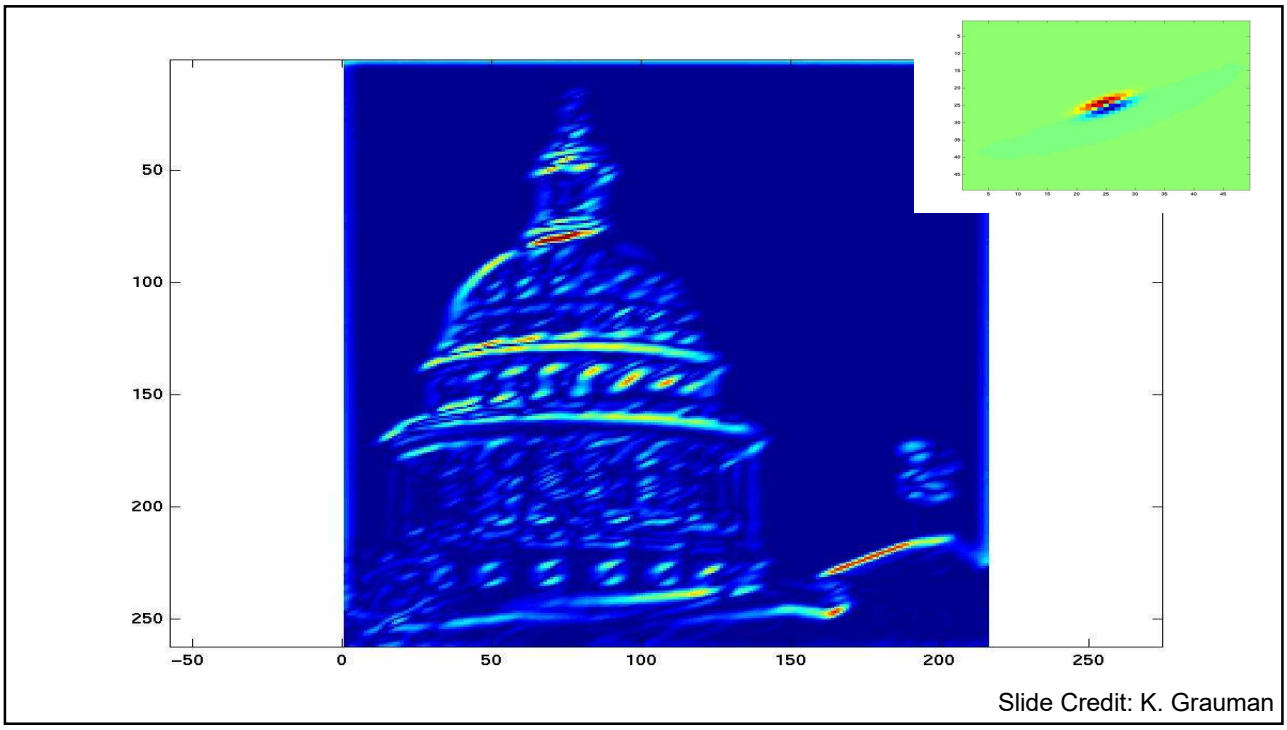
Slide Credit: K. Grauman



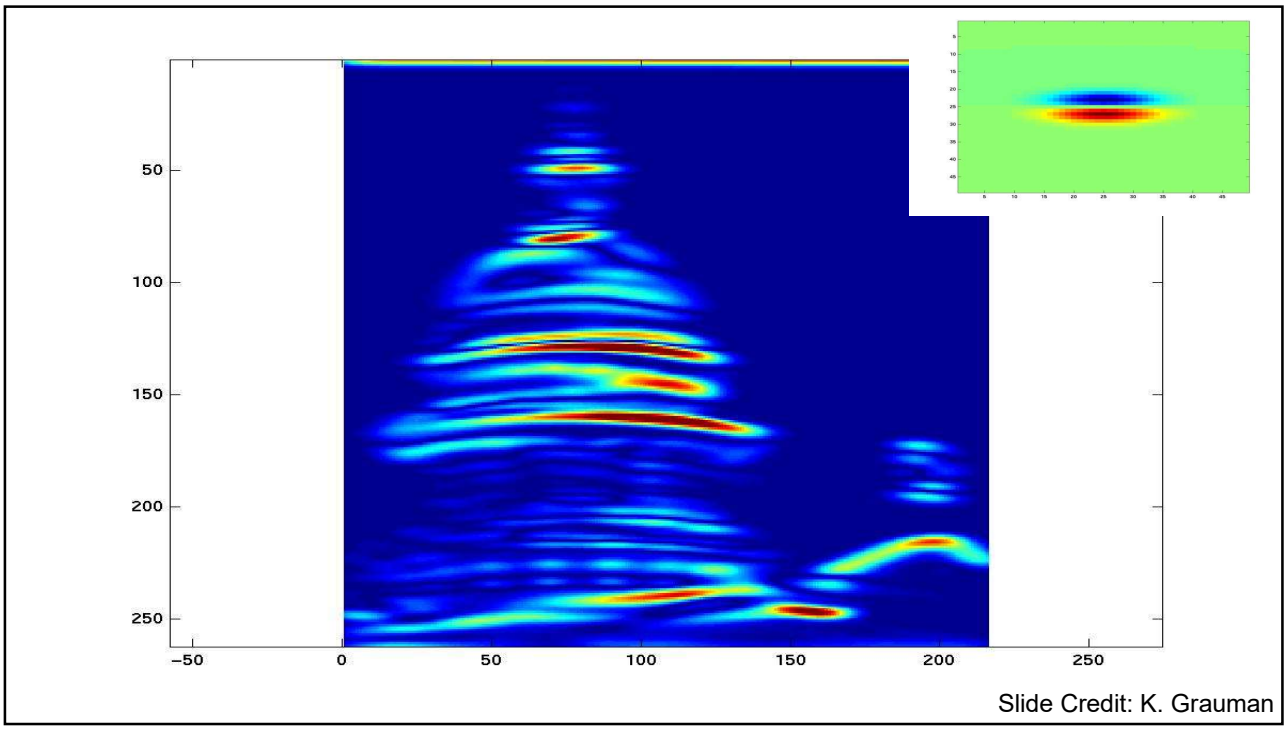
Slide Credit: K. Grauman



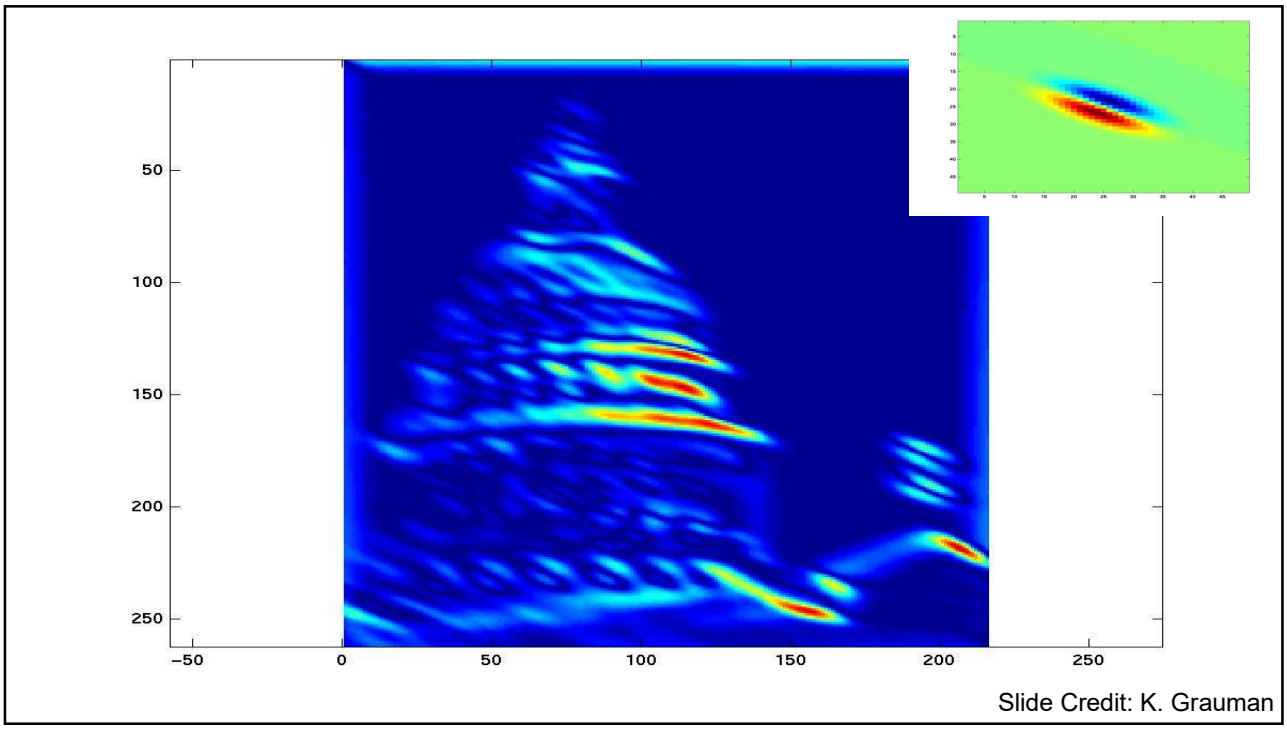
Slide Credit: K. Grauman



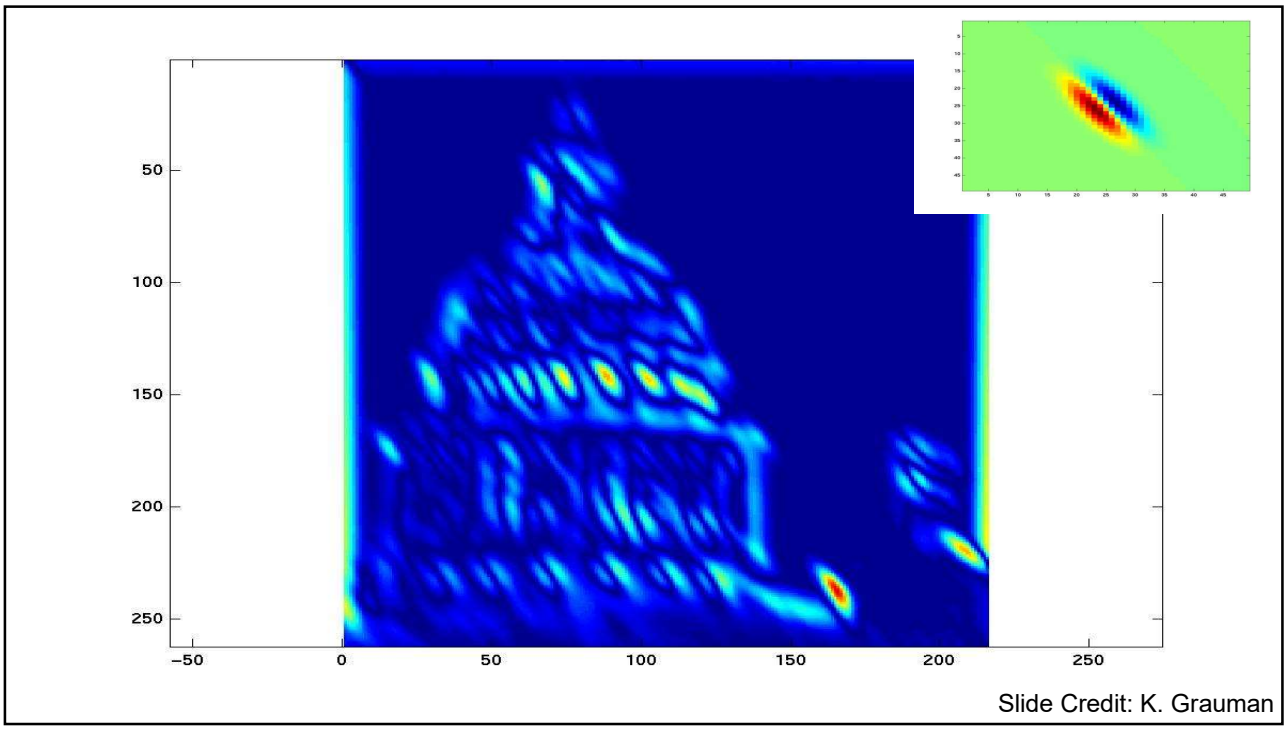
Slide Credit: K. Grauman



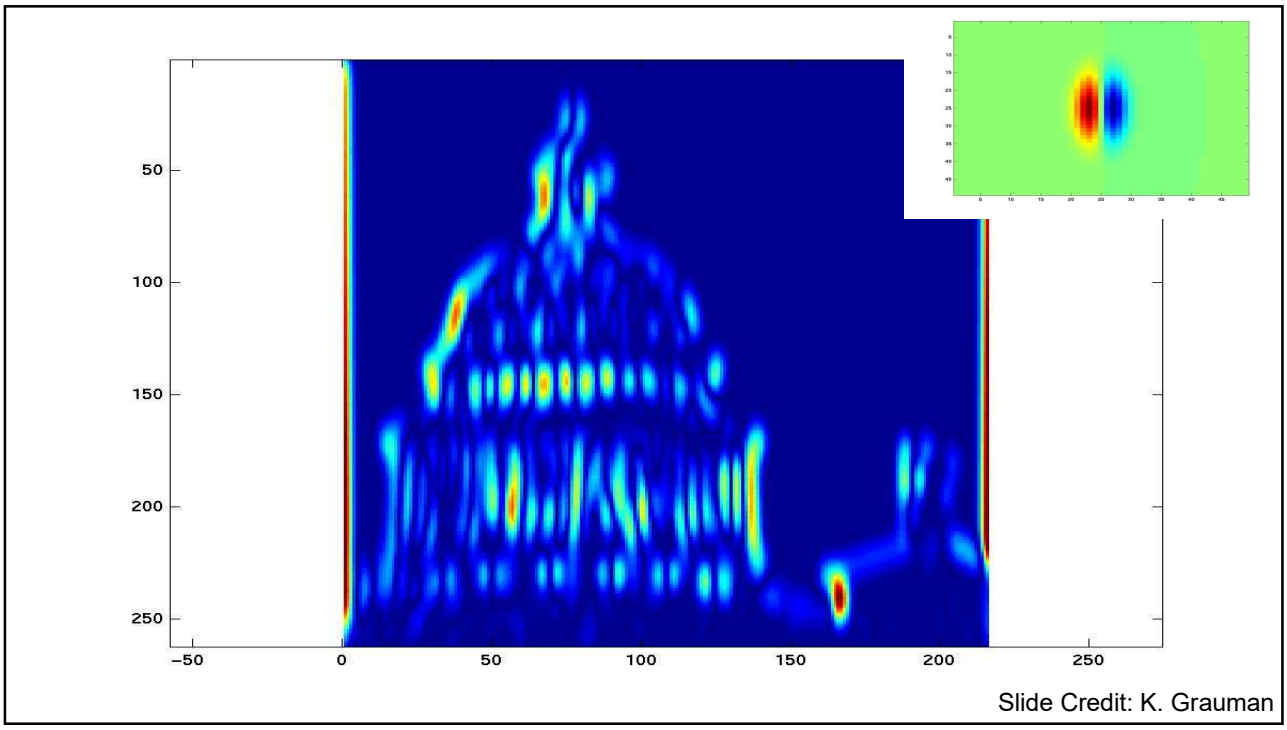
Slide Credit: K. Grauman



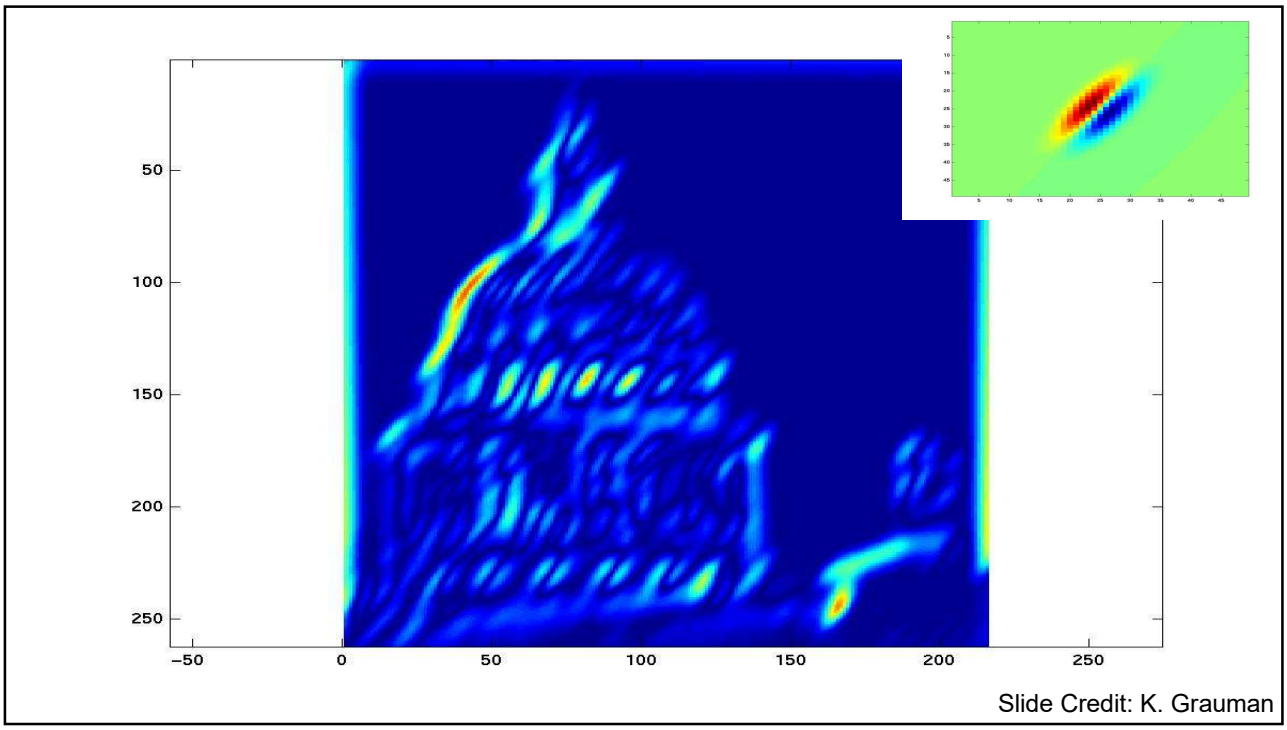
Slide Credit: K. Grauman



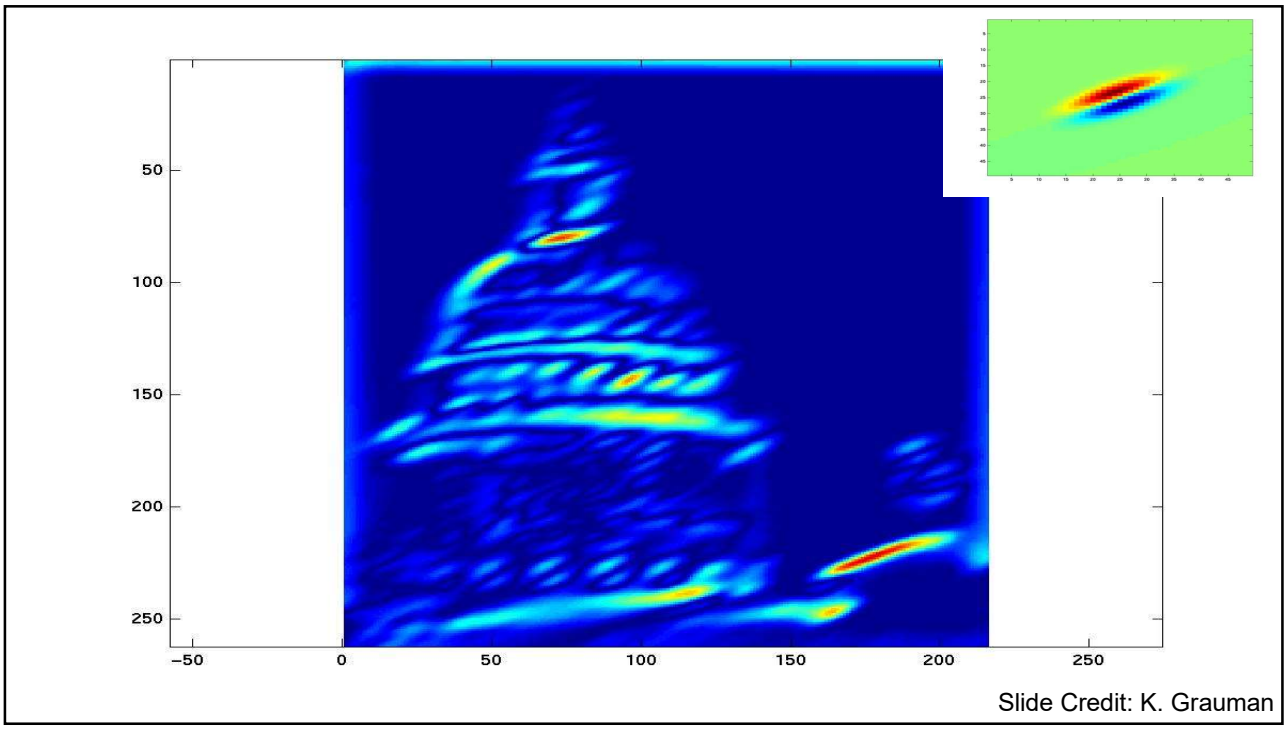
Slide Credit: K. Grauman



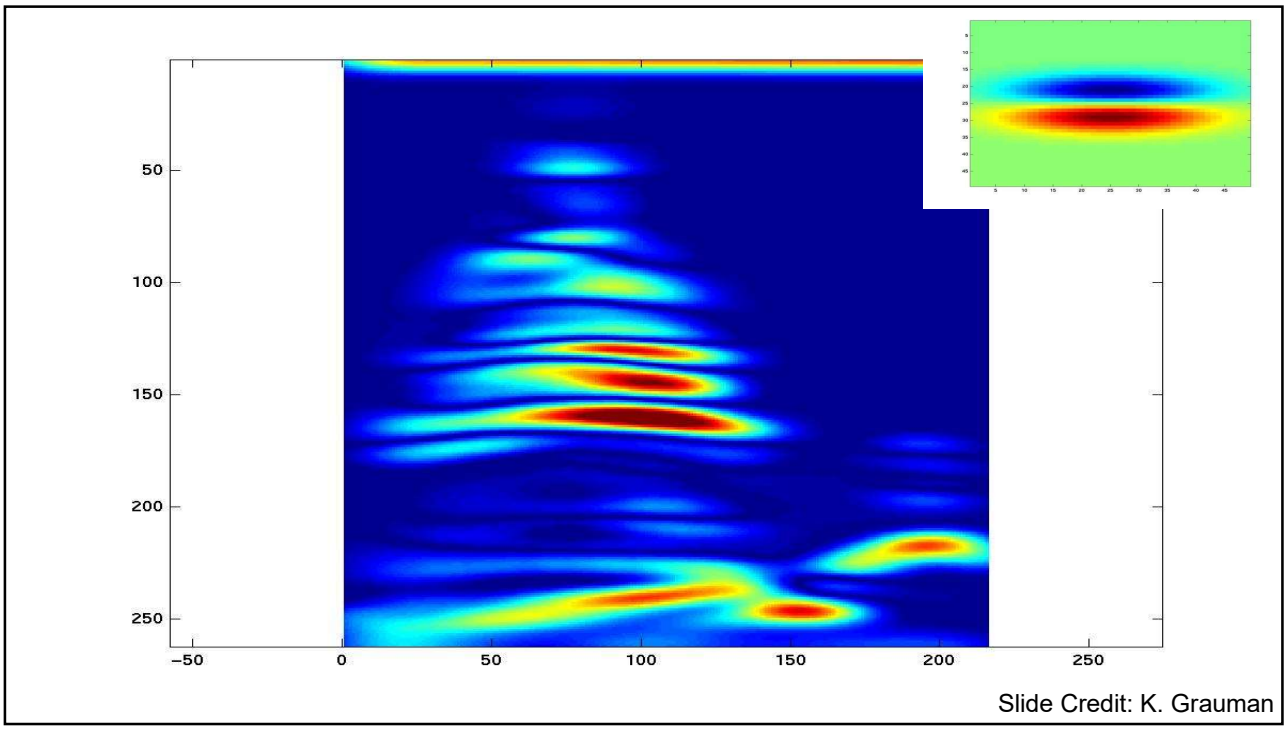
Slide Credit: K. Grauman



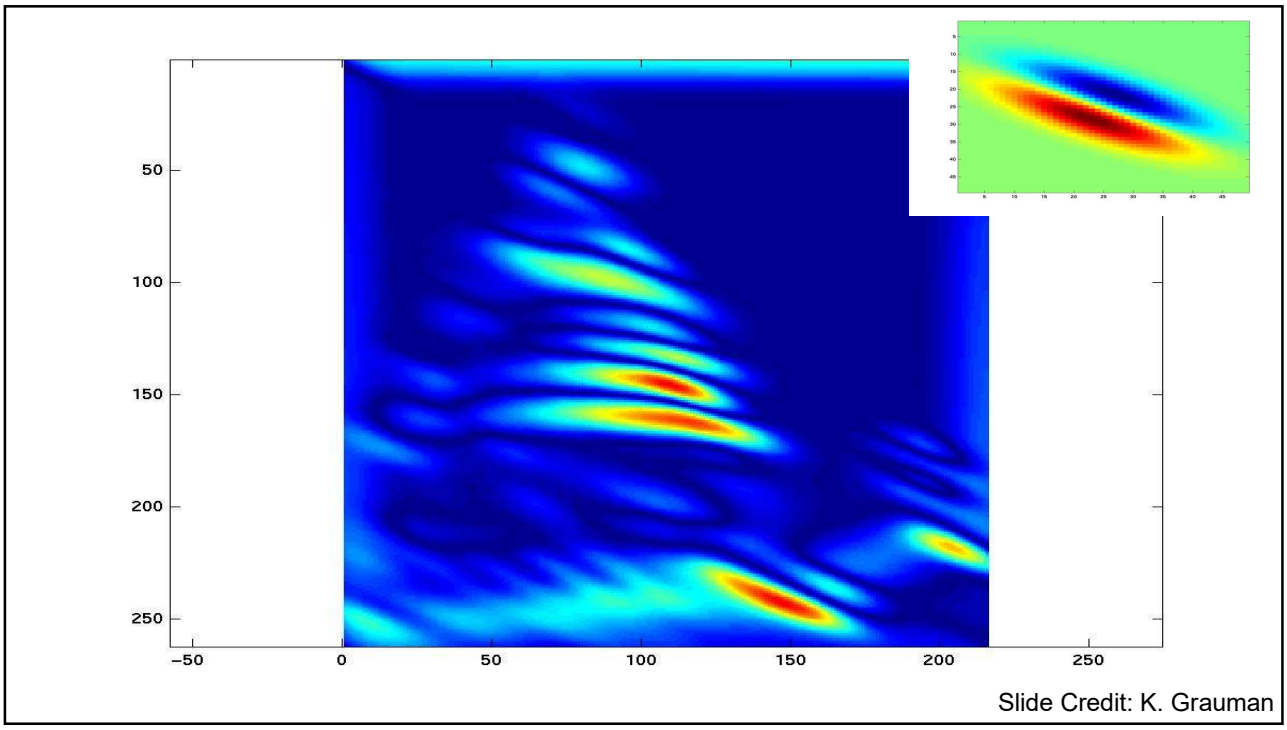
Slide Credit: K. Grauman



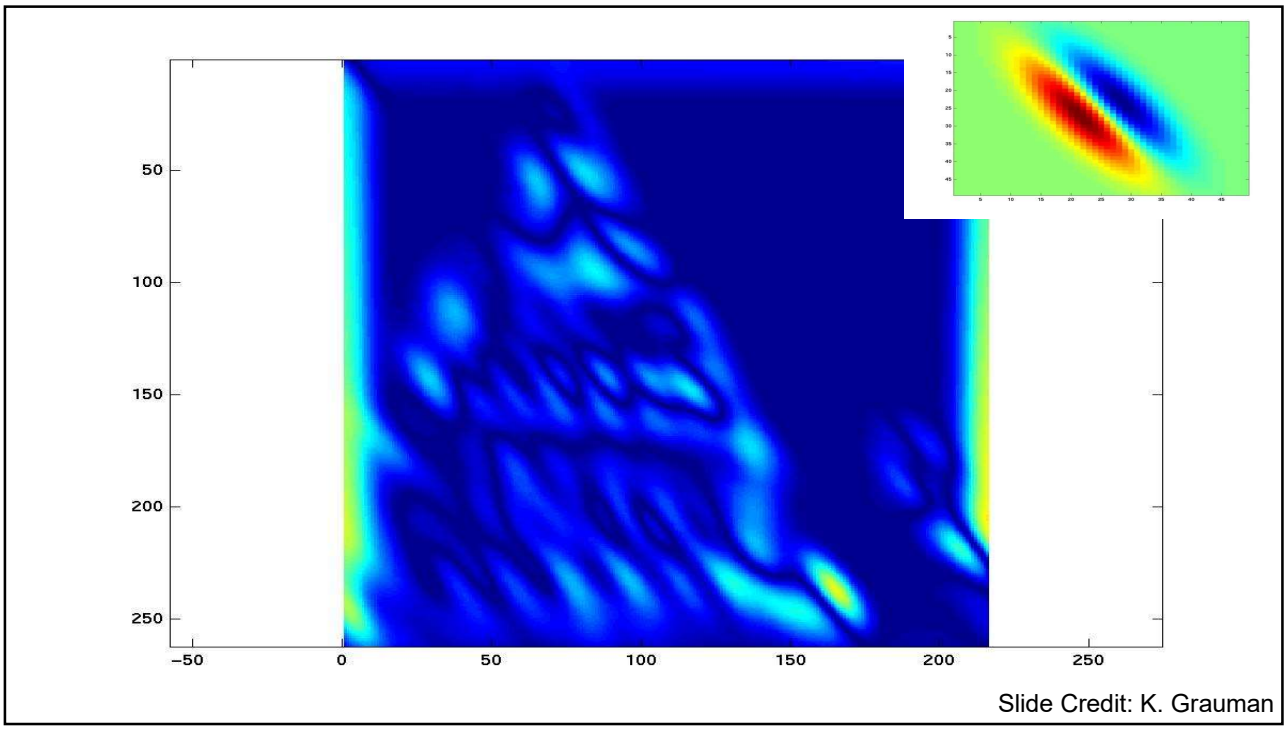
Slide Credit: K. Grauman



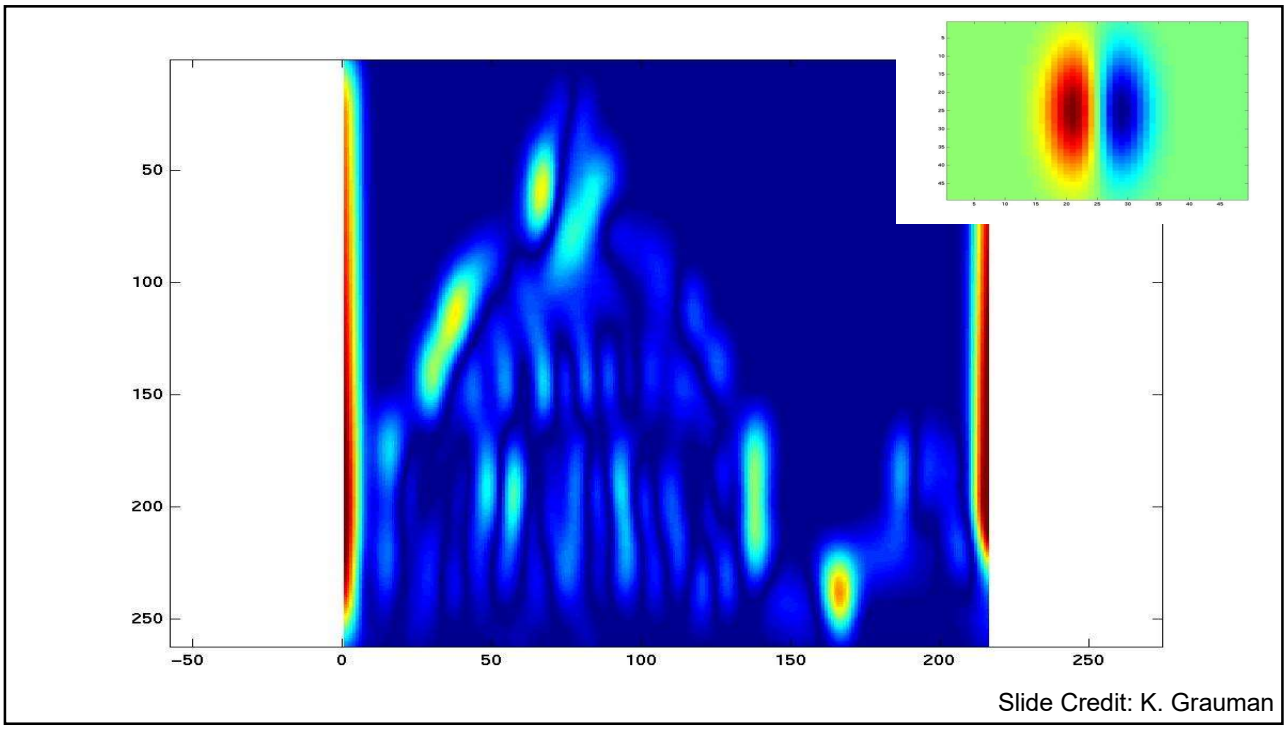
Slide Credit: K. Grauman



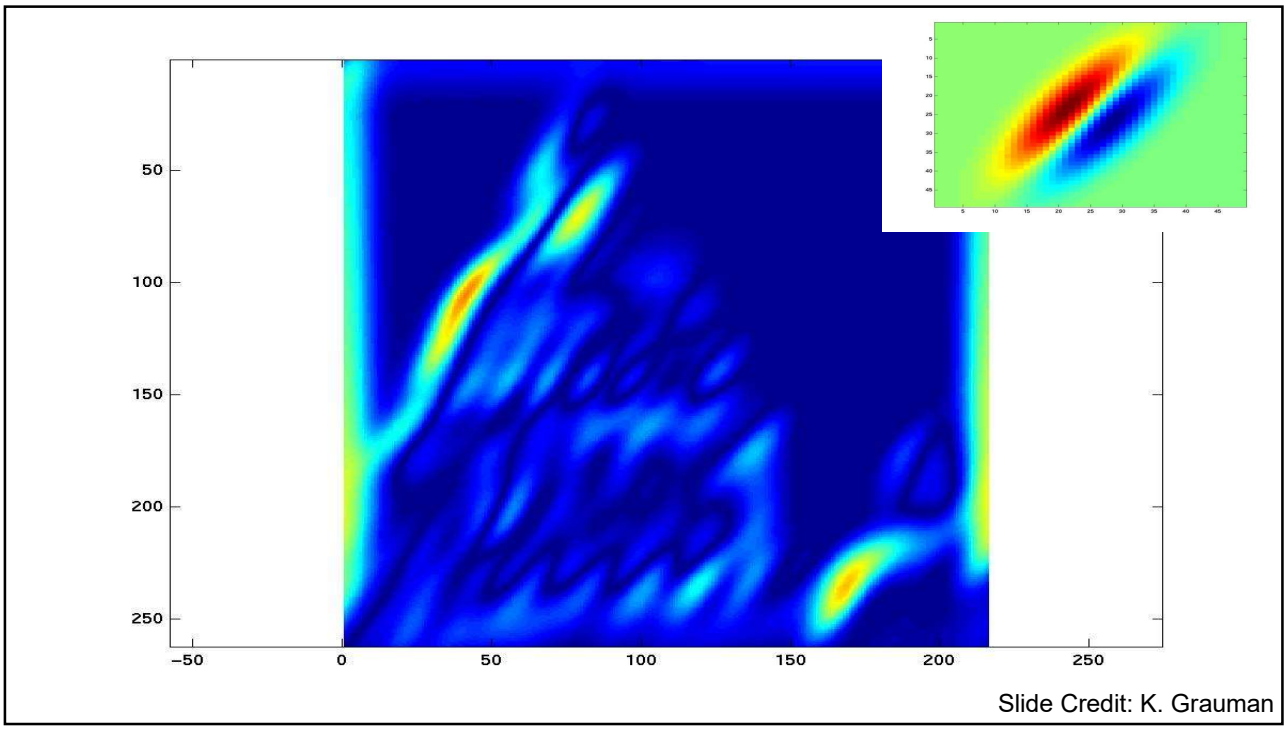
Slide Credit: K. Grauman



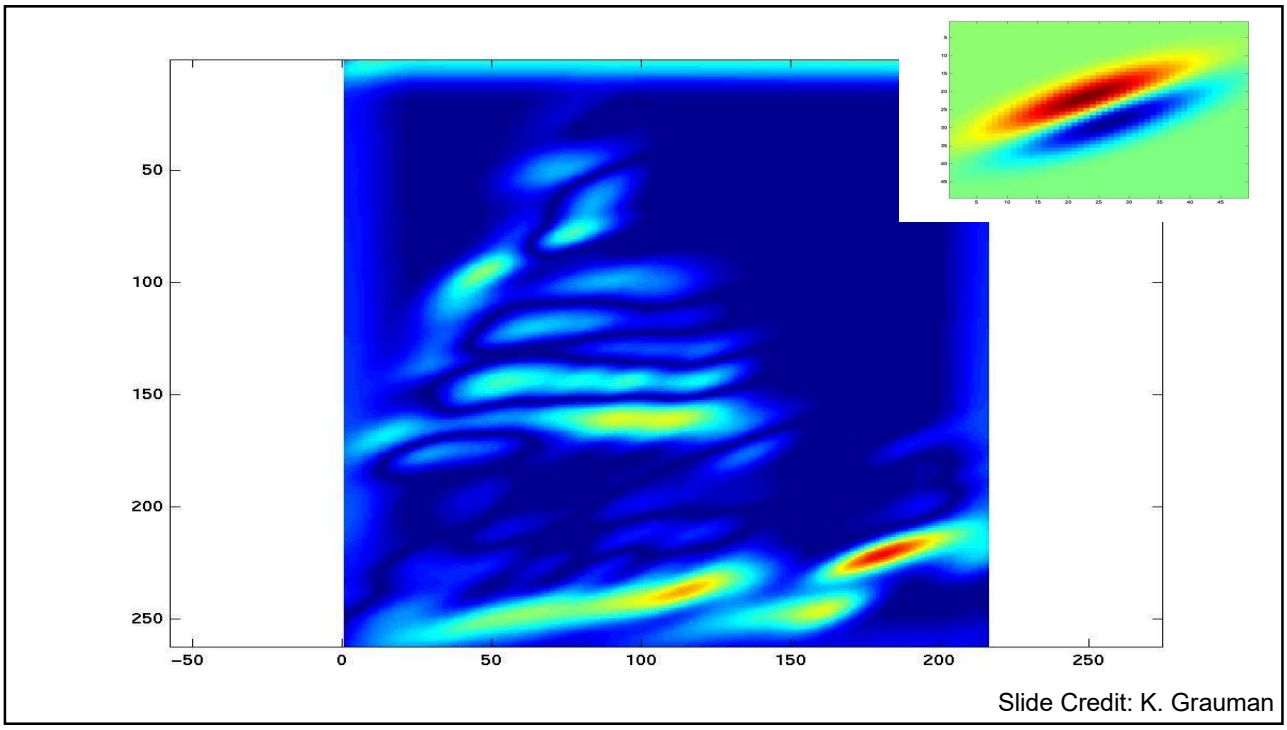
Slide Credit: K. Grauman



Slide Credit: K. Grauman

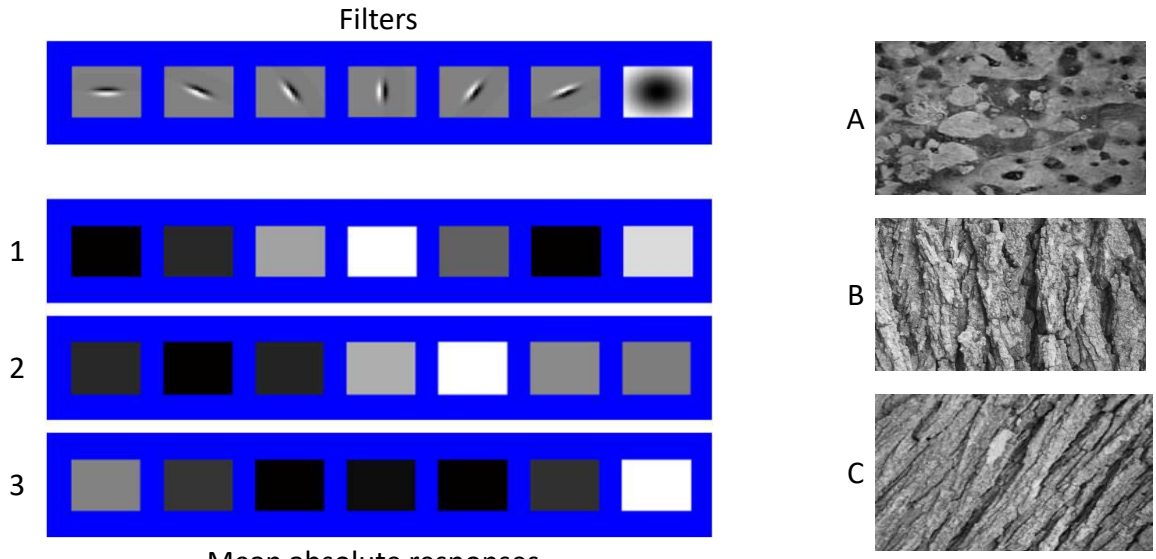


Slide Credit: K. Grauman



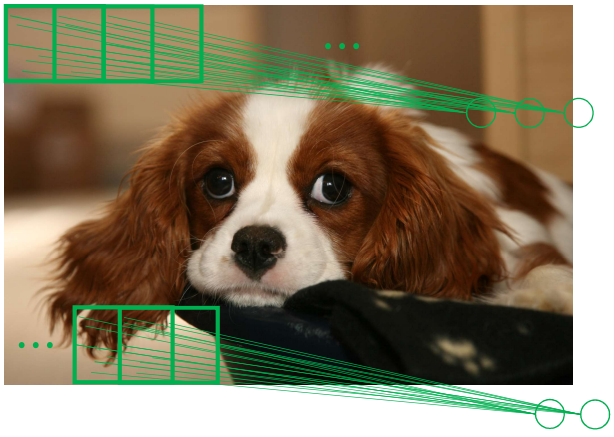
Slide Credit: K. Grauman

Can you match the texture to the response?



51 of 127

Convolutional Layers



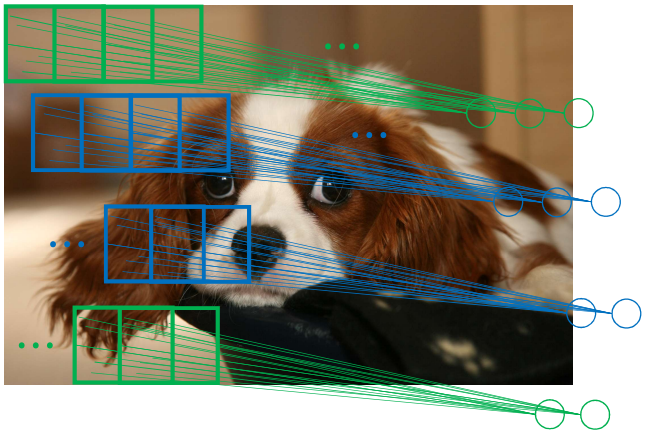
Use convolutions with learned kernels! (In practice its implemented as cross-correlation)

We can leverage on the stationarity* of images and share the same parameters across different locations.

*Stationarity: statistics are similar at different locations

52 of 127

Convolutional Layers



Learn multiple filters!

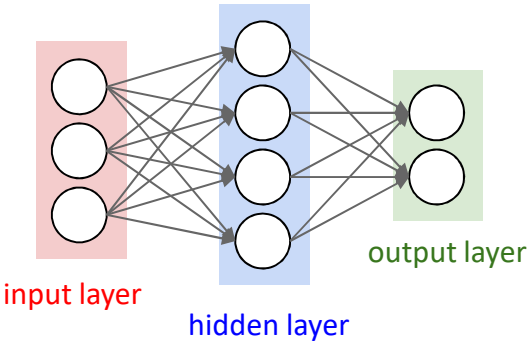
Given a $200 \times 200 \times 3$ RGB Image and we use 100 Filters of size 10×10

How many parameters do we need to learn?

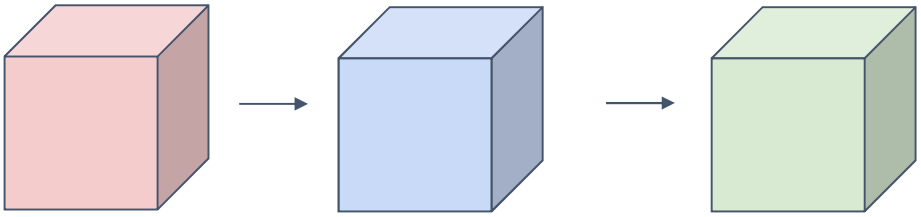
Only 10,000!

53 of 127

before:



now:

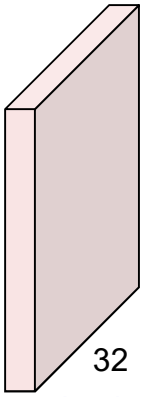


54 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Convolution Layer

32x32x3 image

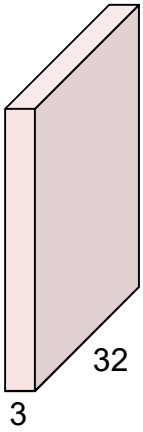


55 of 127

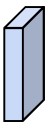
Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Convolution Layer

32x32x3 image



5x5x3 filter



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

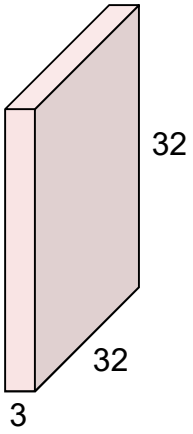
56 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

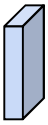
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



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

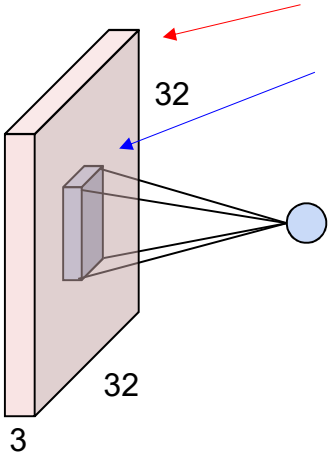
57 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Convolution Layer

32x32x3 image

5x5x3 filter w



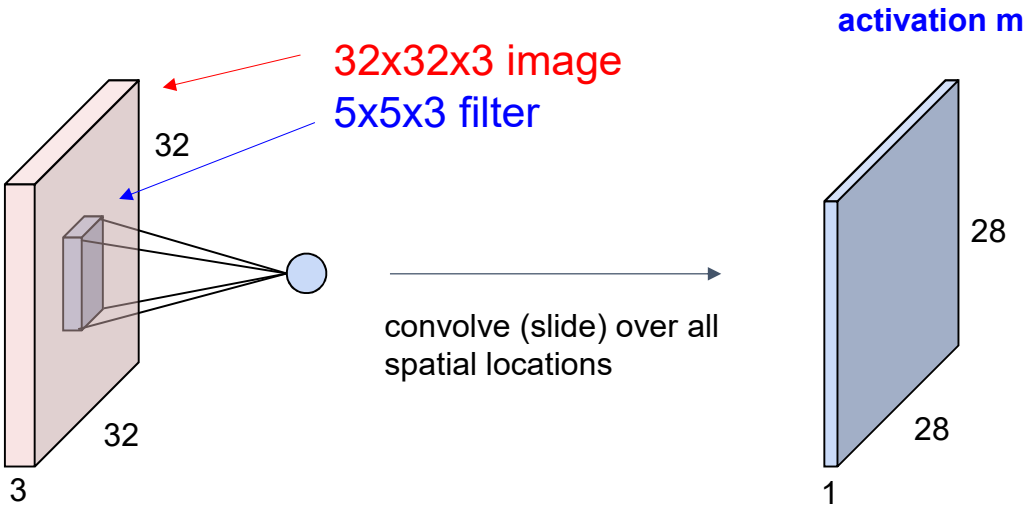
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

58 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Convolution Layer



59 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

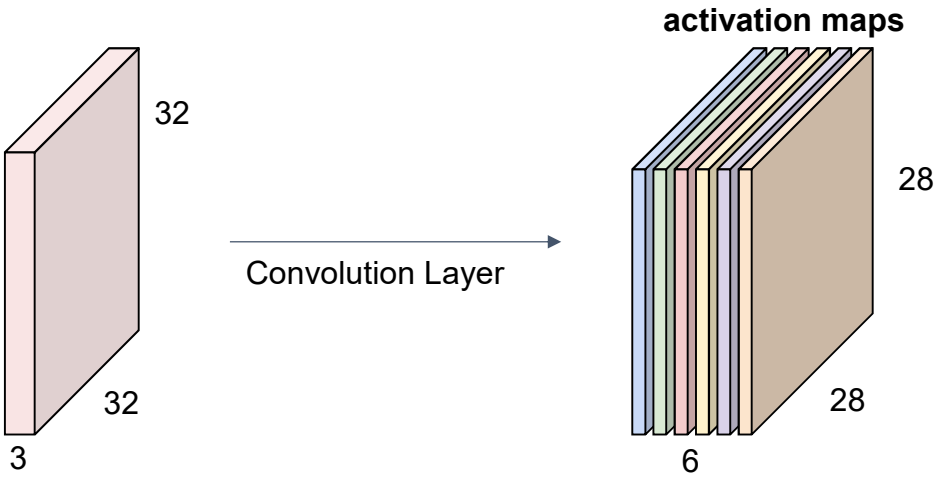
Convolution Layer

consider a second, green filter

60 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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

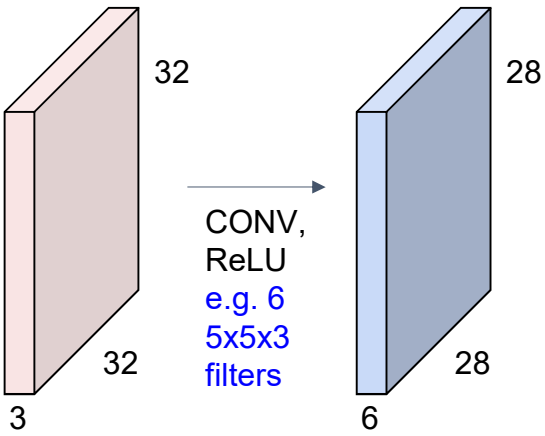


We stack these up to get a "new image" of size 28x28x6!

61 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

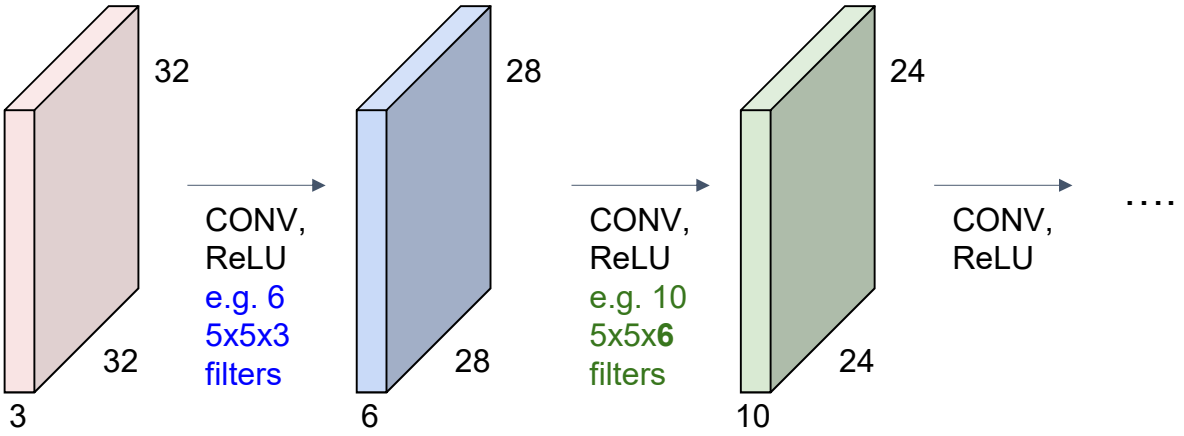
A ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



62 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

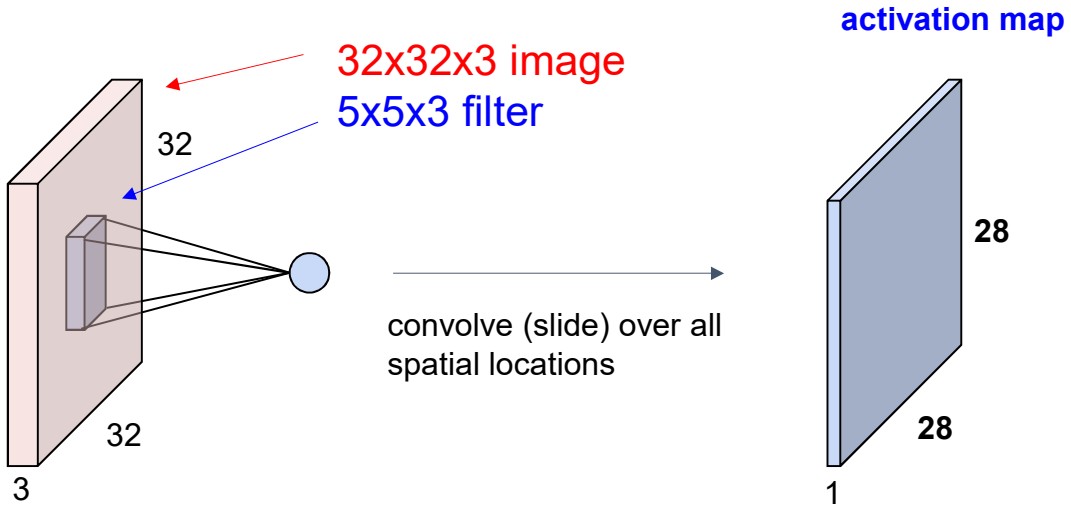
A ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



63 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

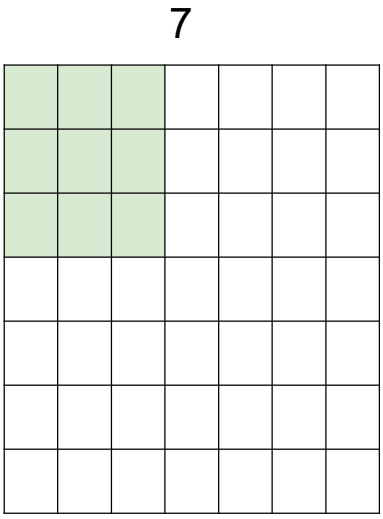
A closer look at spatial dimensions:



64 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

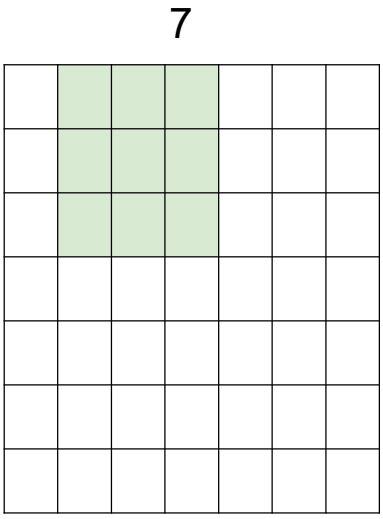


7x7 input (spatially)
assume 3x3 filter

65 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

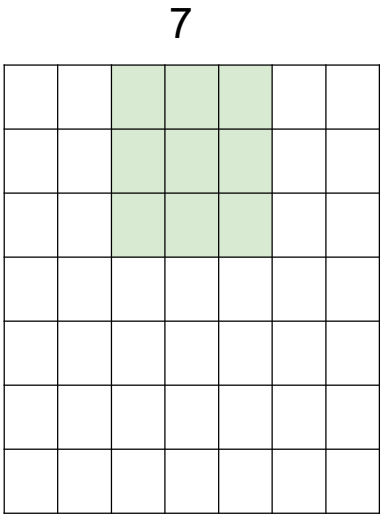


7x7 input (spatially)
assume 3x3 filter

66 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

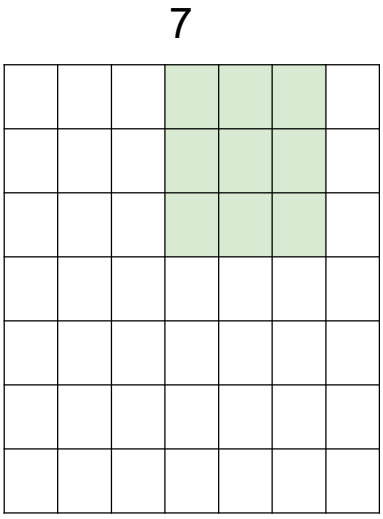


7x7 input (spatially)
assume 3x3 filter

67 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

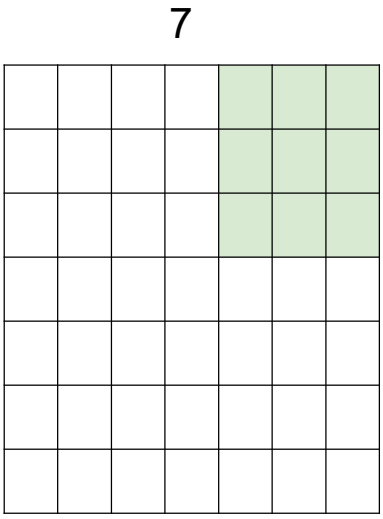


7x7 input (spatially)
assume 3x3 filter

68 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:



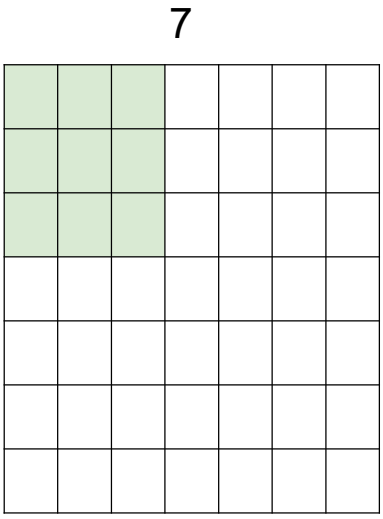
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

69 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

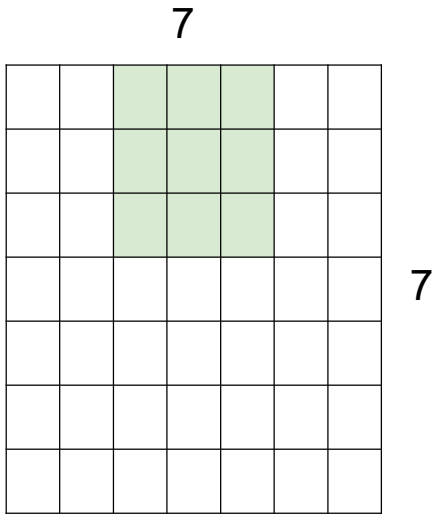


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

70 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

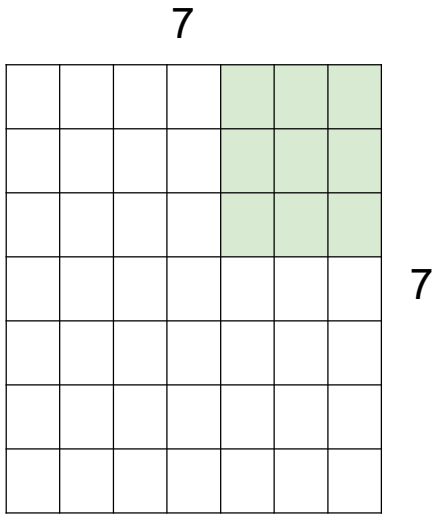


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

71 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

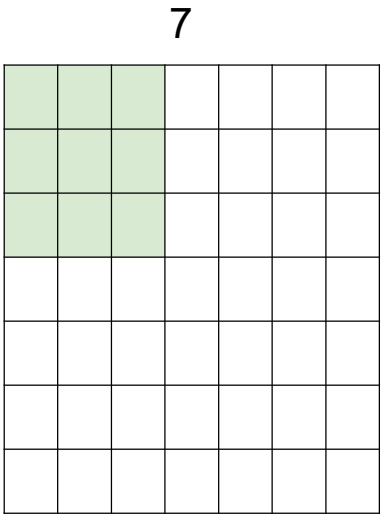


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

72 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

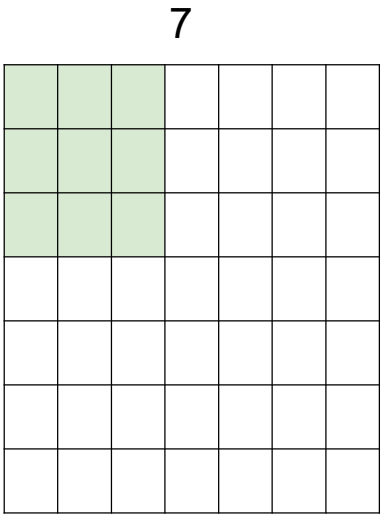


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

73 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

A closer look at spatial dimensions:

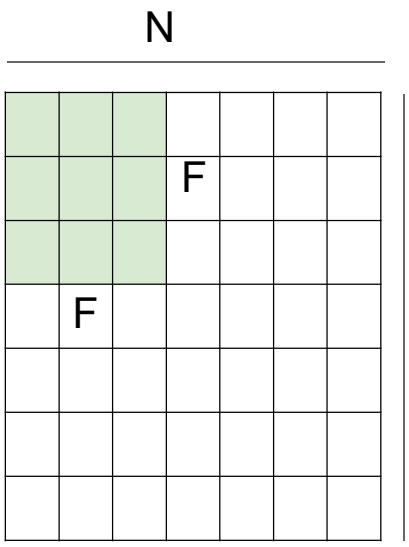


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

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

74 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson



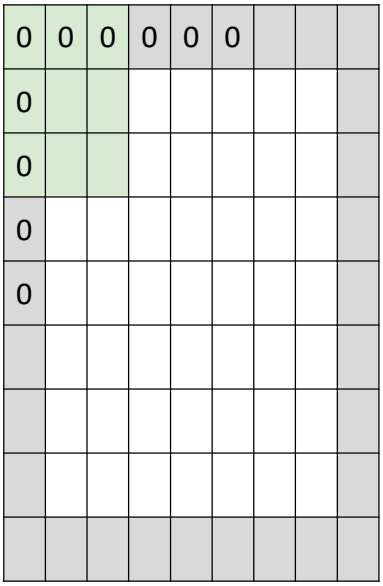
Output size:
(N - F) / stride + 1

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 :\

75 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

In practice: Common to zero pad the border

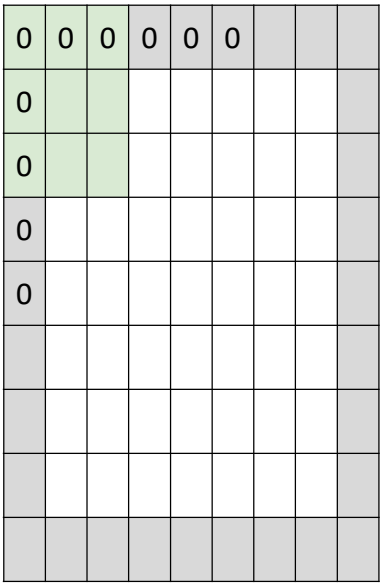


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

In practice: Common to zero pad the border

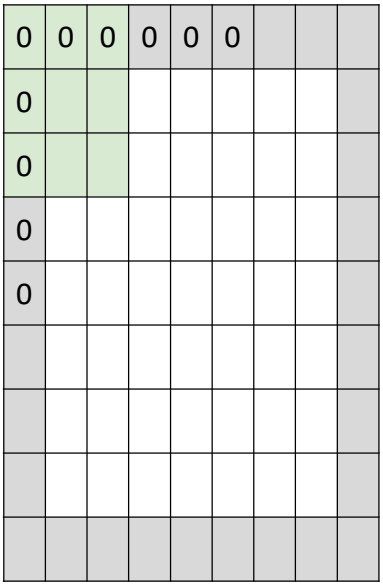


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

In practice: Common to zero pad the border

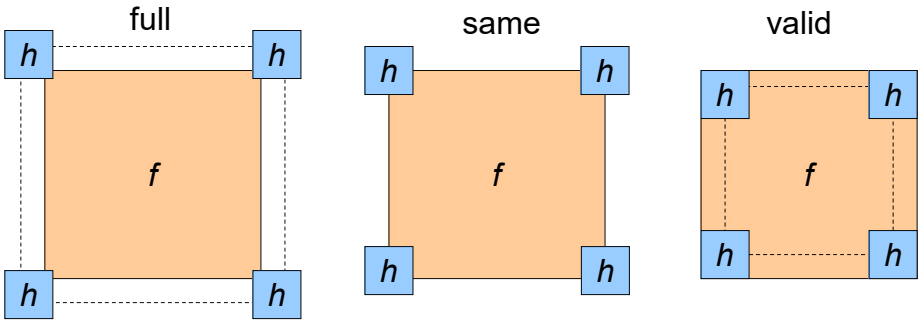


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

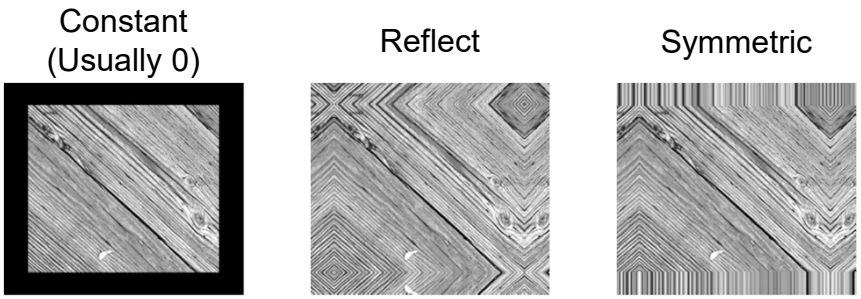
Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Dealing with the edges



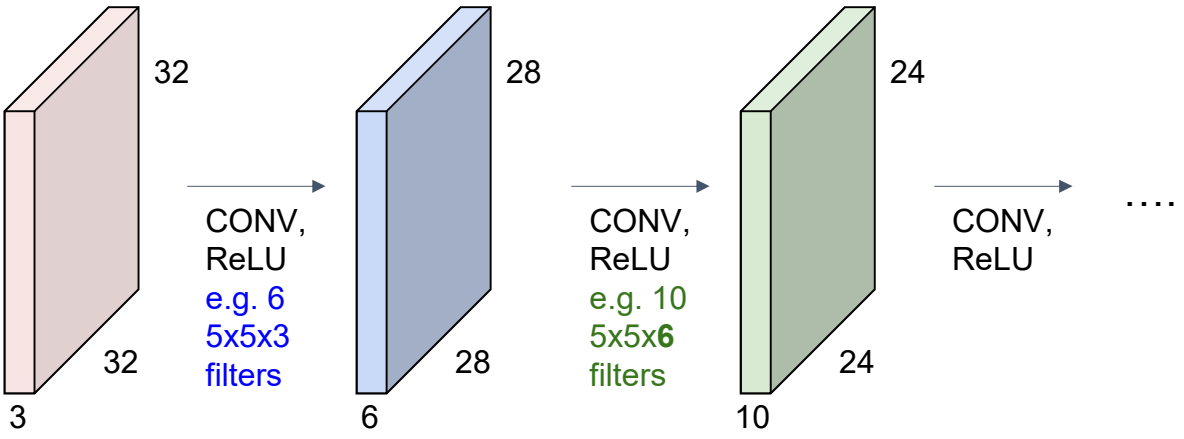
79 of 127

Some types of padding



80 of 127

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.

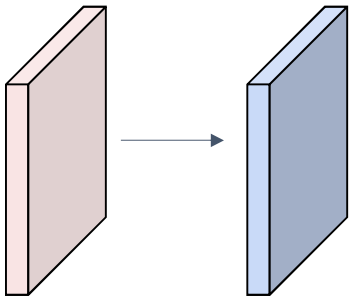


81 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Examples time:

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

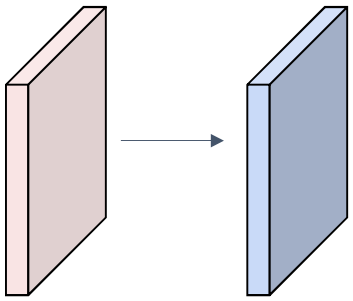


Output volume size: ?

82 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Examples time:

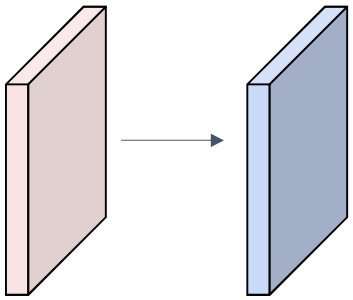


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Examples time:

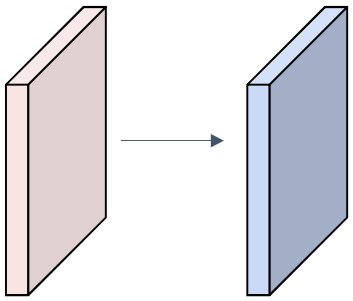


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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**

85 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Summary. To summarize, the Conv Layer:

- 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.

86 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Summary. To summarize, the Conv Layer:

- 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.

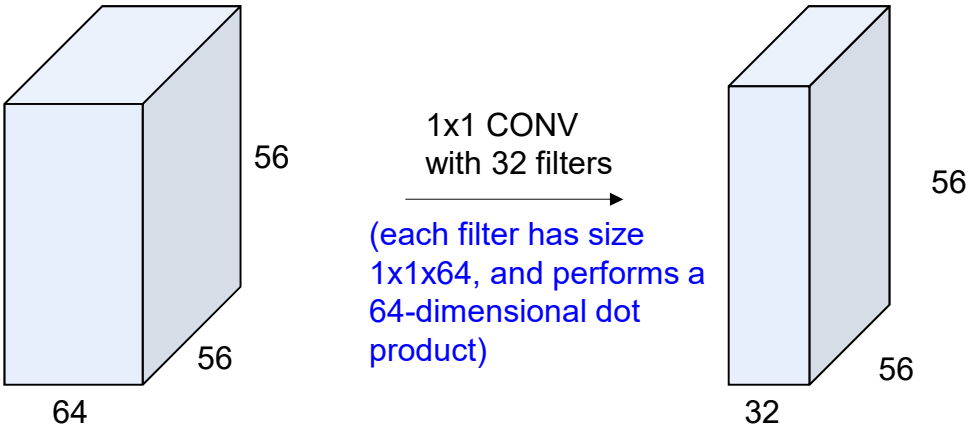
Common settings:

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

87 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

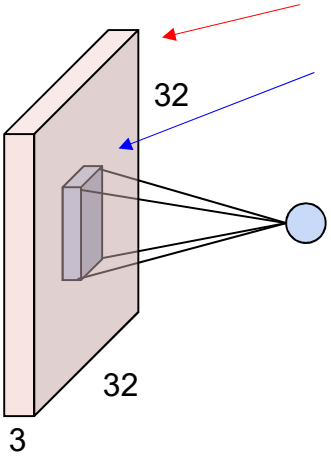
(btw, 1x1 convolution layers make perfect sense)



88 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

The brain/neuron view of CONV Layer



32x32x3 image
5x5x3 filter

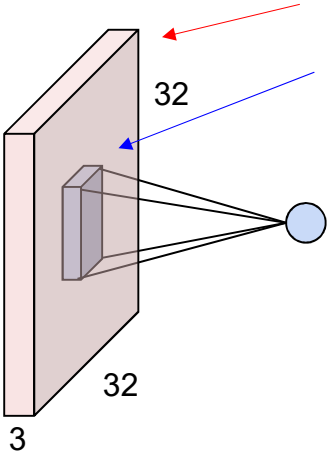
1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

89 of 127

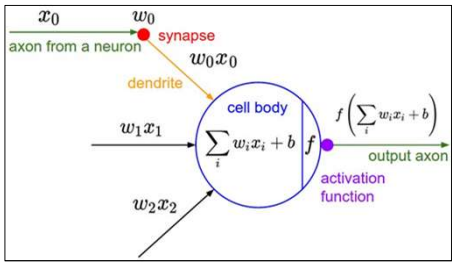
Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

89

The brain/neuron view of CONV Layer



32x32x3 image
5x5x3 filter



It's just a neuron with local connectivity...

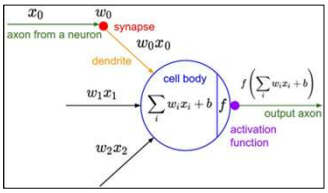
1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

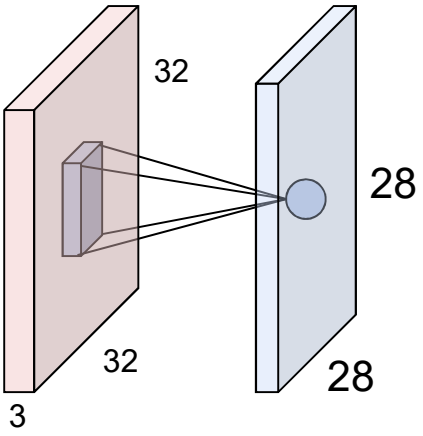
90 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

90

The brain/neuron view of CONV Layer





An activation map is a 28x28 sheet of neuron outputs:

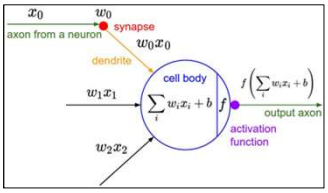
1. Each is connected to a small region in the input
2. All of them share parameters

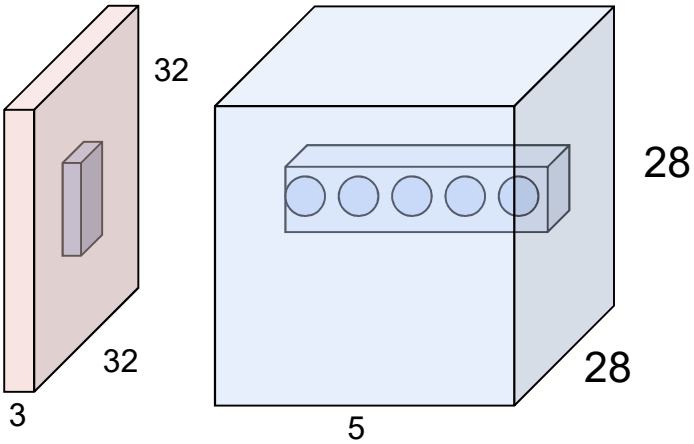
"5x5 filter" -> "5x5 receptive field for each neuron"

91 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

The brain/neuron view of CONV Layer





E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

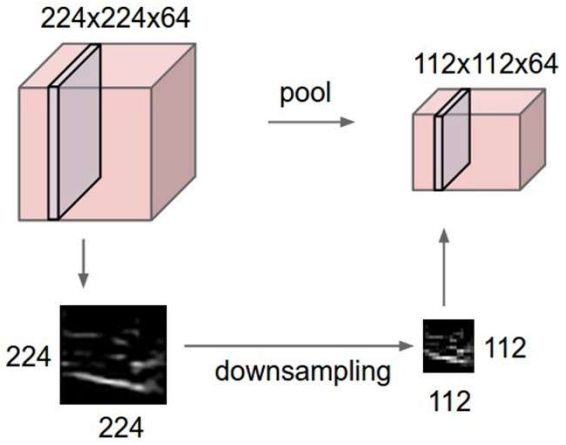
There will be 5 different neurons all looking at the same region in the input volume

92 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Pooling layer

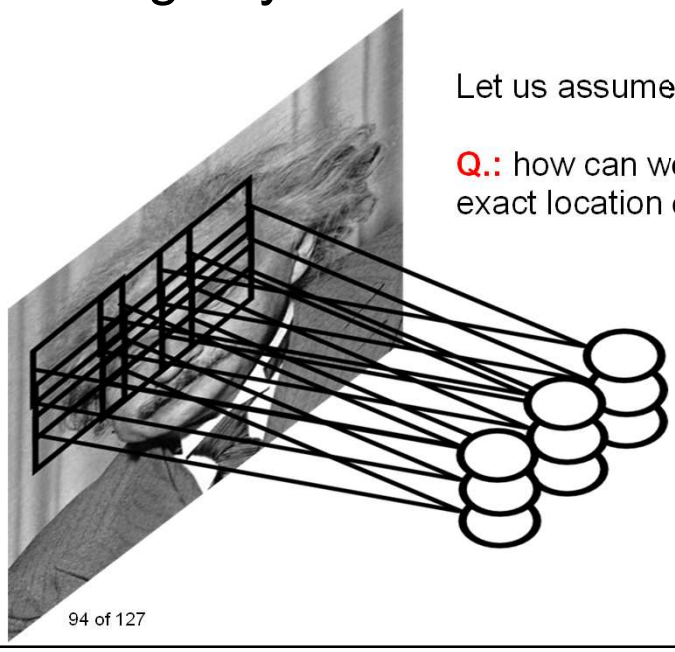
- makes the representations smaller and more manageable
- operates over each activation map independently:



93 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Pooling Layer



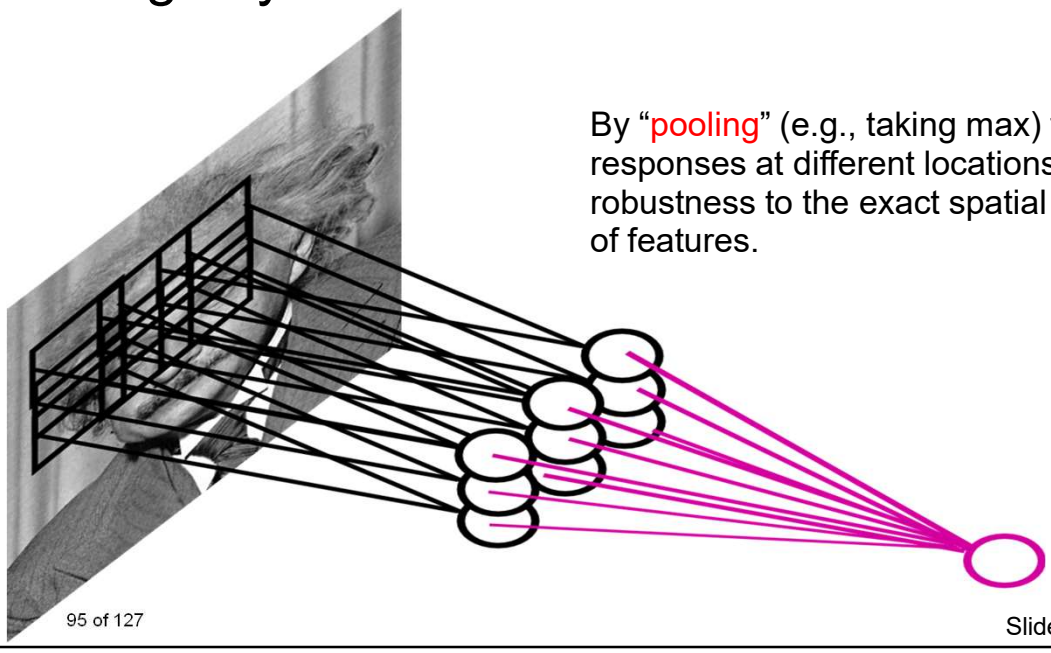
Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

94 of 127

Slide Credit: Ranzato

Pooling Layer



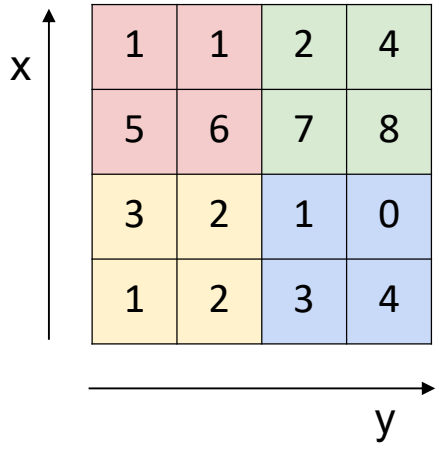
By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

95 of 127

Slide Credit: Ranzato

MAX POOLING

Single depth slice



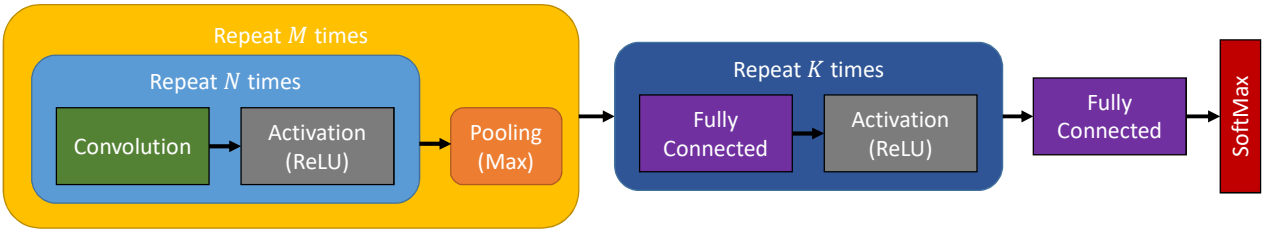
max pool with 2x2 filters and stride 2

6	8
3	4

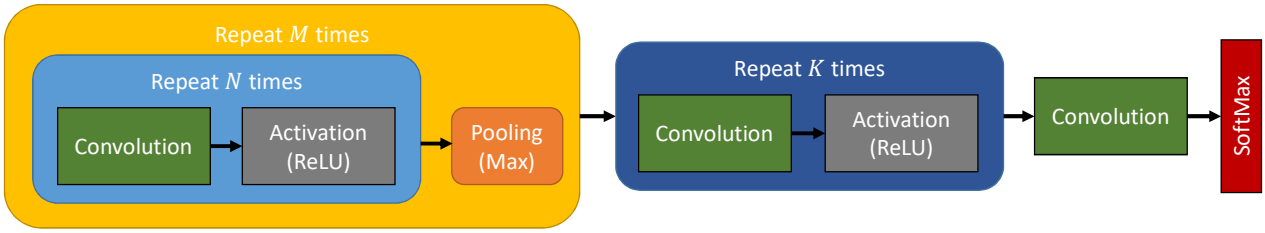
96 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

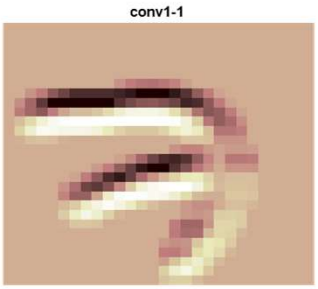
Common Network Architecture Pattern

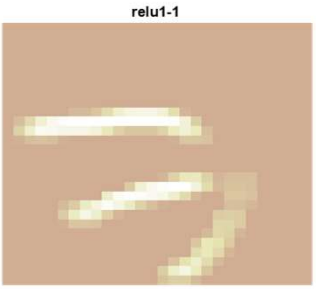


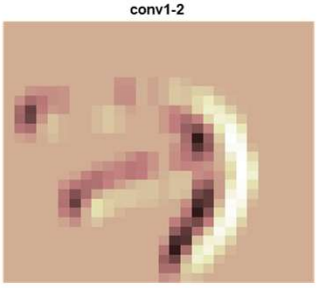
More Recent Architectures are Fully Convolutional

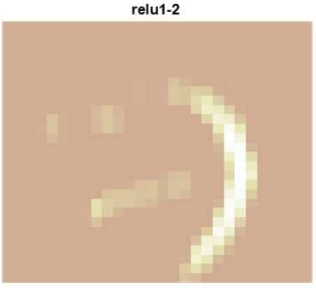


97 of 127

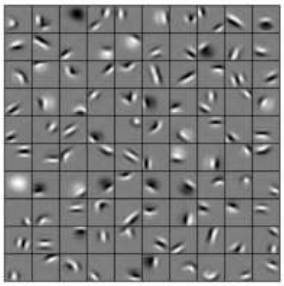




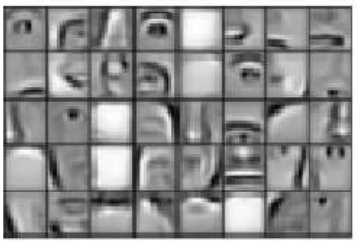




98 of 127



Layer 1



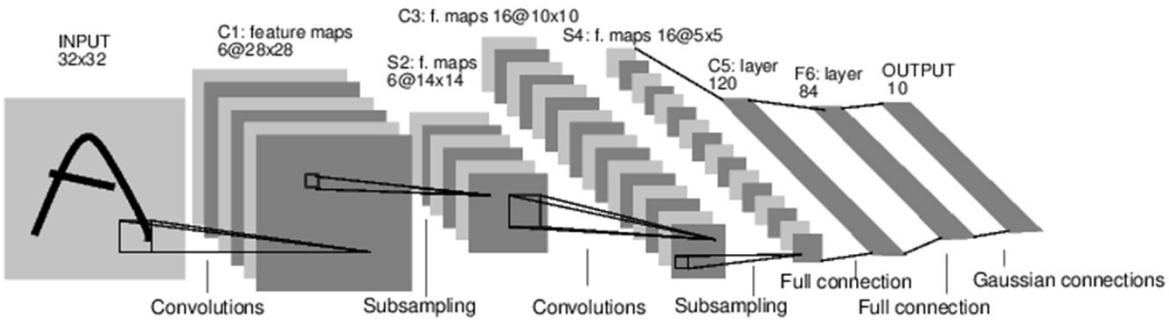
Layer 2

Layer 3

99 of 127

Case Study: LeNet-5

[LeCun et al., 1998]



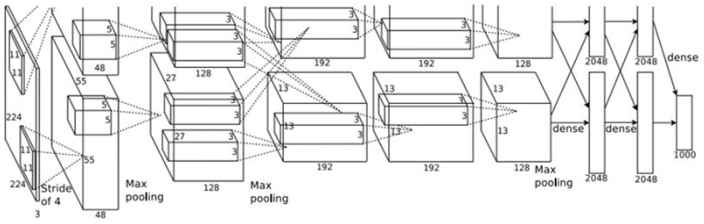
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

100 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



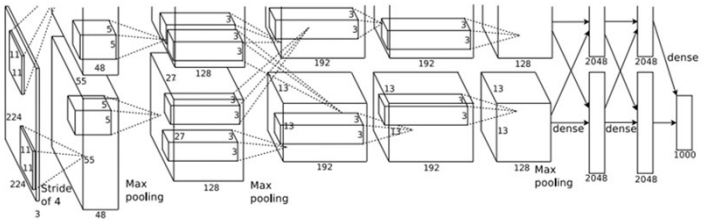
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

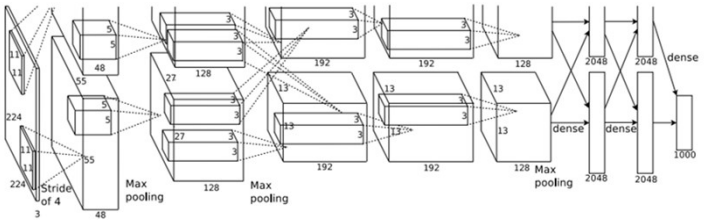
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



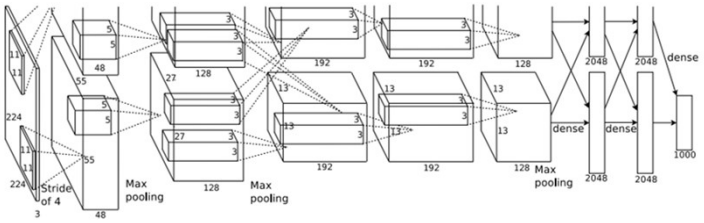
Input: 227x227x3 images

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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

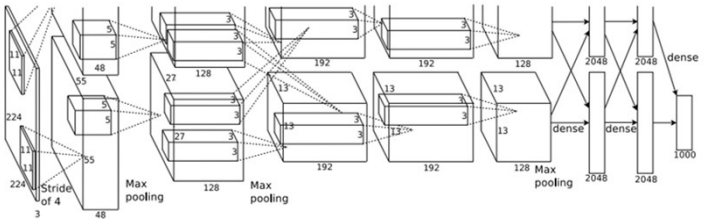
Second layer (POOL1): 3x3 filters applied at stride 2

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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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

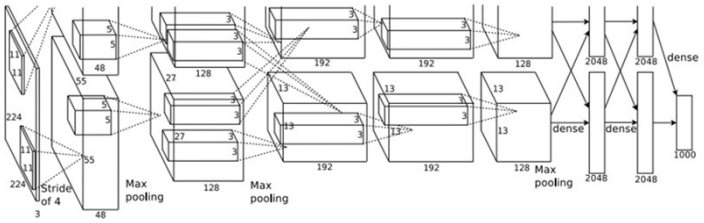
Q: what is the number of parameters in this layer?

105 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

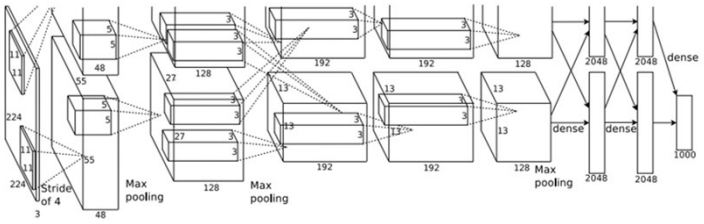
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

106 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]

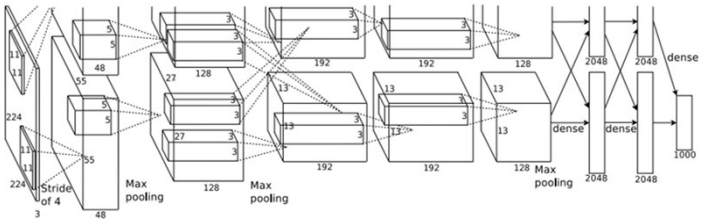


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]

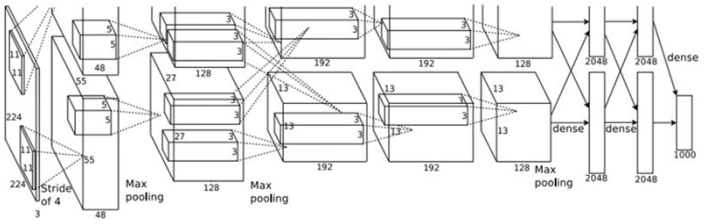


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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%

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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 **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					

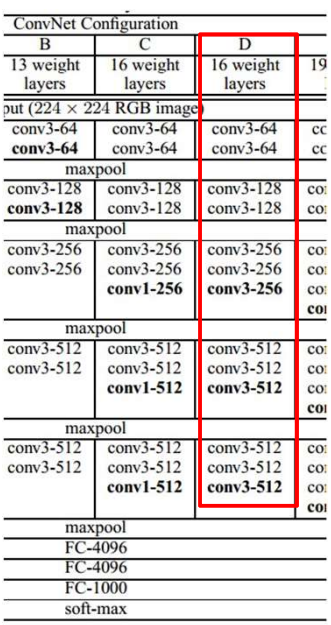
Table 2: **Number of parameters** (in millions).

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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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
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
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
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

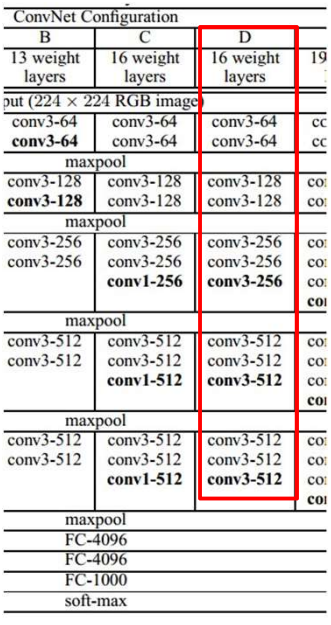


Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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
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
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
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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
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
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
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

Most params are in late FC

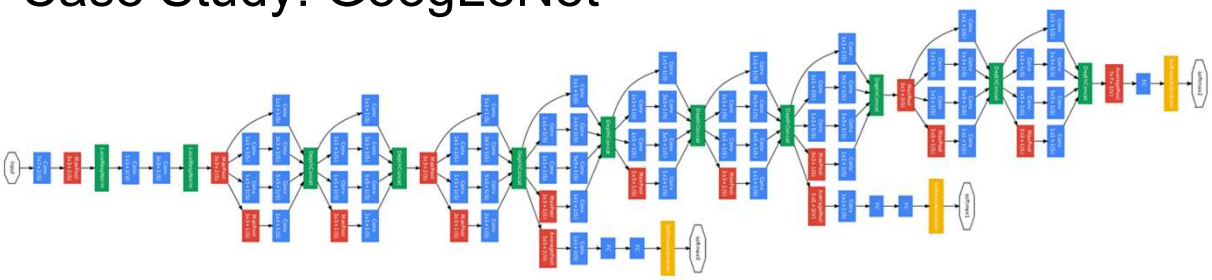
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

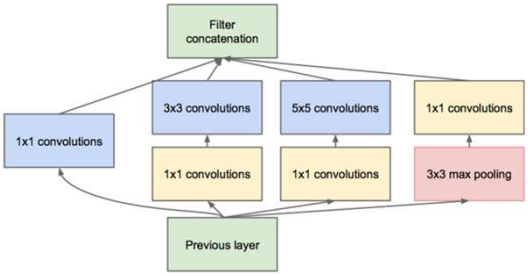
113 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

114 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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)

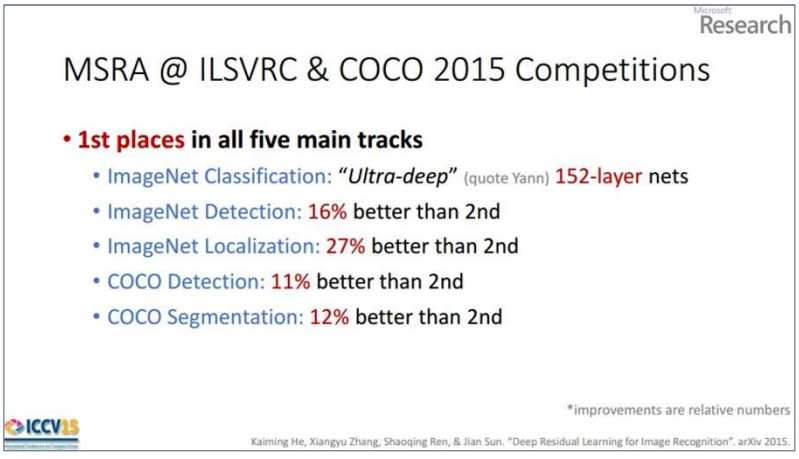
Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

115 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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

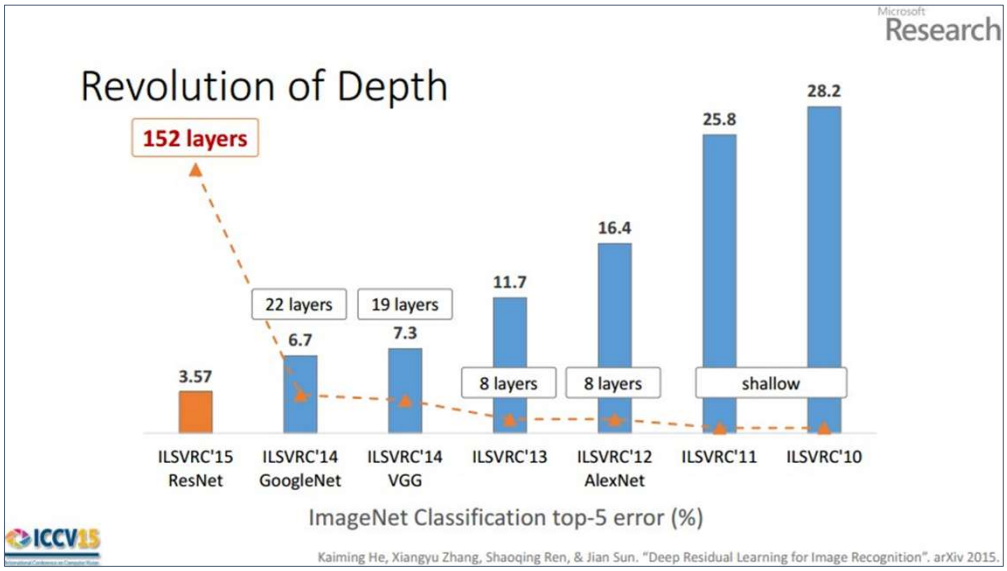
ILSVRC 2015 winner (3.6% top 5 error)



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

116 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

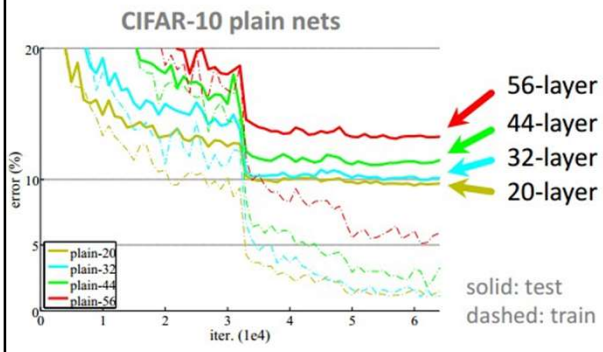


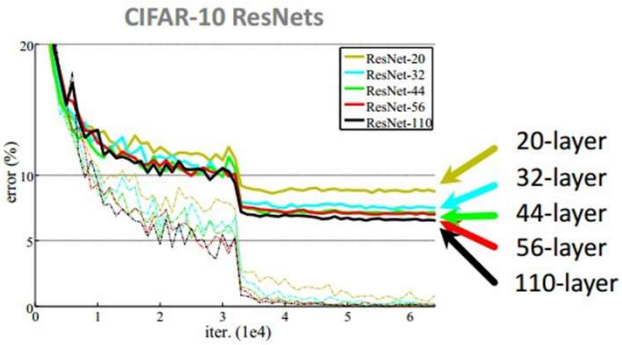
(slide from Kaiming He's recent presentation)

117 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

CIFAR-10 experiments



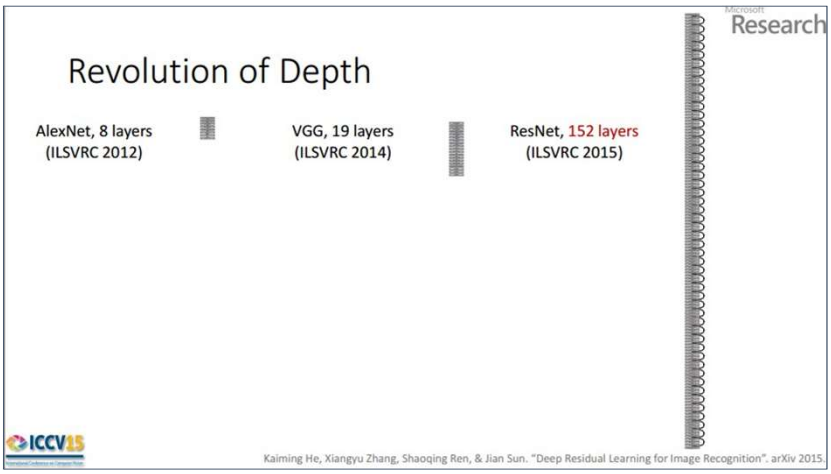


118 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

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

ILSVRC 2015 winner (3.6% top 5 error)



2-3 weeks of training on 8 GPU machine

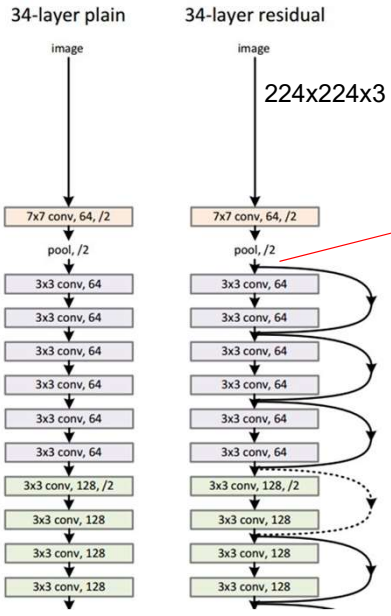
at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He's recent presentation)

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: ResNet

[He et al., 2015]

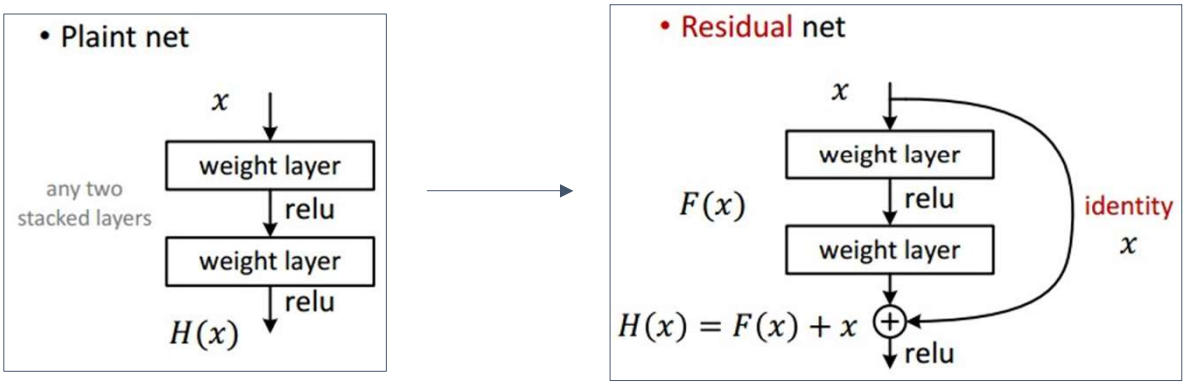


spatial dimension only 56x56!

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: ResNet

[He et al., 2015]



Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: ResNet

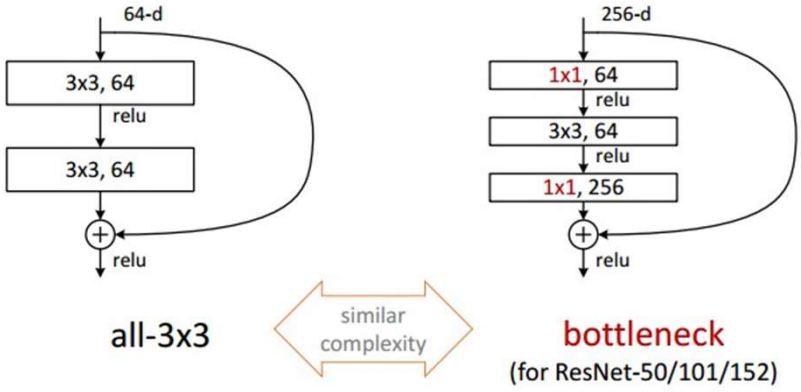
[He et al., 2015]

- 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

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: ResNet

[He et al., 2015]

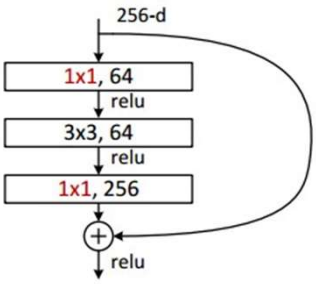


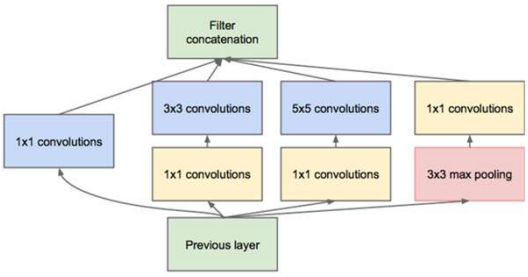
We are able to process bigger inputs in this formulation

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Case Study: ResNet

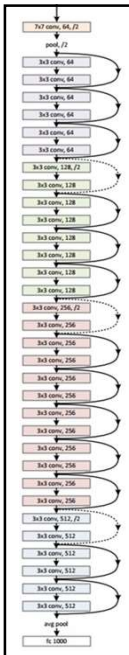
[He et al., 2015]





(this trick is also used in GoogLeNet)

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

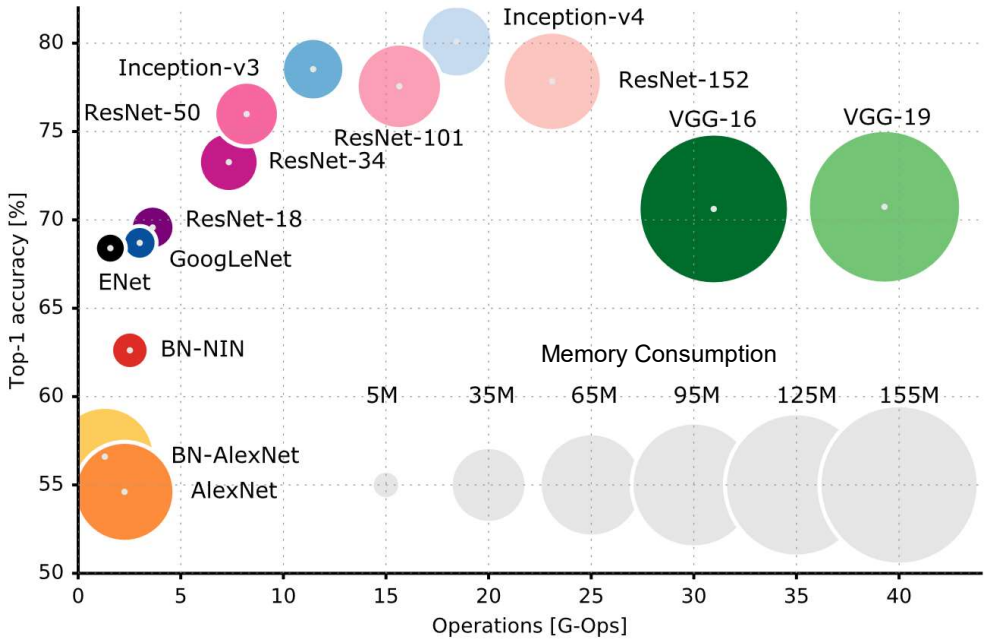


Case Study: ResNet *[He et al., 2015]*

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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

125 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson



126 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Summary

- ConvNets stack CONV,ReLU,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Early architectures look like
  - **[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX**
- but recent advances such as ResNet/GoogLeNet use only Conv-ReLU, 1x1 convolutions and Softmax

127 of 127

Slide Credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Homework -5

- Multinomial Regression will be posted tonight
- Deadline: next week Monday 11pm

128 of 127