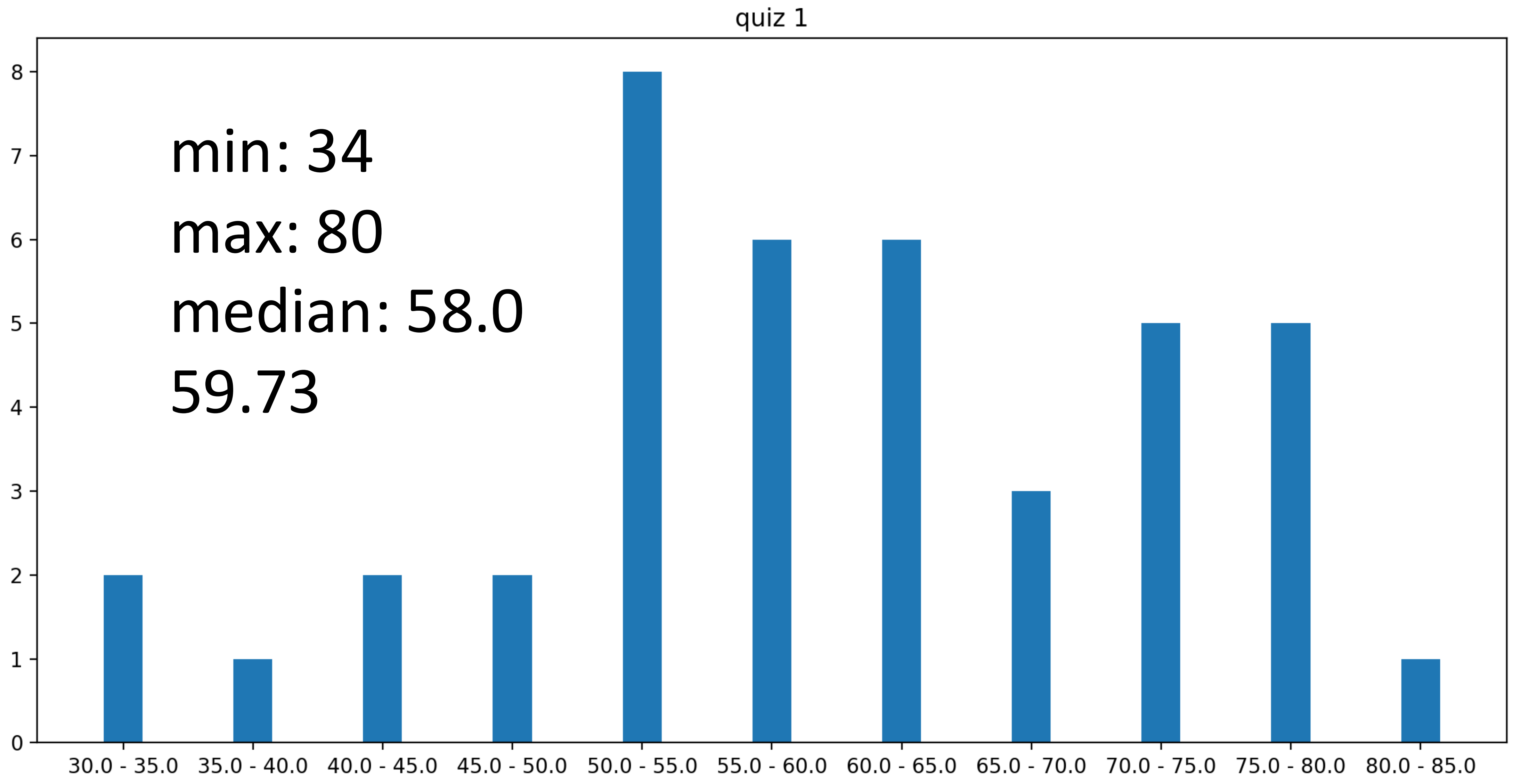
Convolutional Neural Networks II

CS7150, Spring 2025

Prof. Huaizu Jiang

Northeastern University

Distribution of the Grades of Quiz 1



How to cope with the unsatisfactory grade

- It’s one of several in-class quizzes. So it’s weight won’t be too high.
- The difficulty is similar to the midterm. After all, this is a 7000-level course.
- Think, instead of memorizing.
- Ask questions if anything is not clear.
- There are opportunities to earn extra credits that will be applied to your final grade directly.

Topic of the Course Project

- How can I pick a topic?
- How much time should the project take?
- Do I need to have start-of-the-art performance?
- Common Pitfalls and things to avoid.

How can I pick a topic?

- Choose a specific PROBLEM:
  - Computer vision
    - Object classification: which object is shown?
    - Object detection: where are all the objects?
    - Segmentation: Assign a region type to every pixel in the image.
  - Natural Language Processing
    - Translation (difficult)
    - Language modeling
    - Sentiment analysis
    - Co-reference resolution
  - Other domain
    - music genre identification
    - simple speech recognition (difficult)
    - Visual navigation in a 2D maze with a RL agent

How to choose a topic

- Choose an area of interest to you.
- However, beware of pitfalls described below.
- Look at Kaggle competitions.
  - You don’t have to win the competition!
- Look at common data sets and how they are typically used.

How much time should I spend on it?

- About the equivalent of ~3 problem assignments. A bit more or less is OK.
- As early as possible, get SOMETHING going.
  - Example: If I'm doing classification, make sure my basic idea works for just 2 classes before running it on 10 classes.
  - Example: If I'm doing detection, make sure my network is better than a simple R-CNN without pre-training.
  - Get something going soon! Then improve it. So-called baseline.
- Failure case: GANs.
  - Many students never get them to work AT ALL! Results are indistinguishable from a random image.
  - Try diffusion models.

Do I need to have state-of-the-art performance?

- In a word, “NO”.
- Here’s what you do need to do:
  - Show that your model is better than something very simple (at least consider the random guess chance first)
  - If you do variations of your model (say, different architectures), discuss the differences in performance for the variations.
  - Try to learn something about what improves or hurts performance.

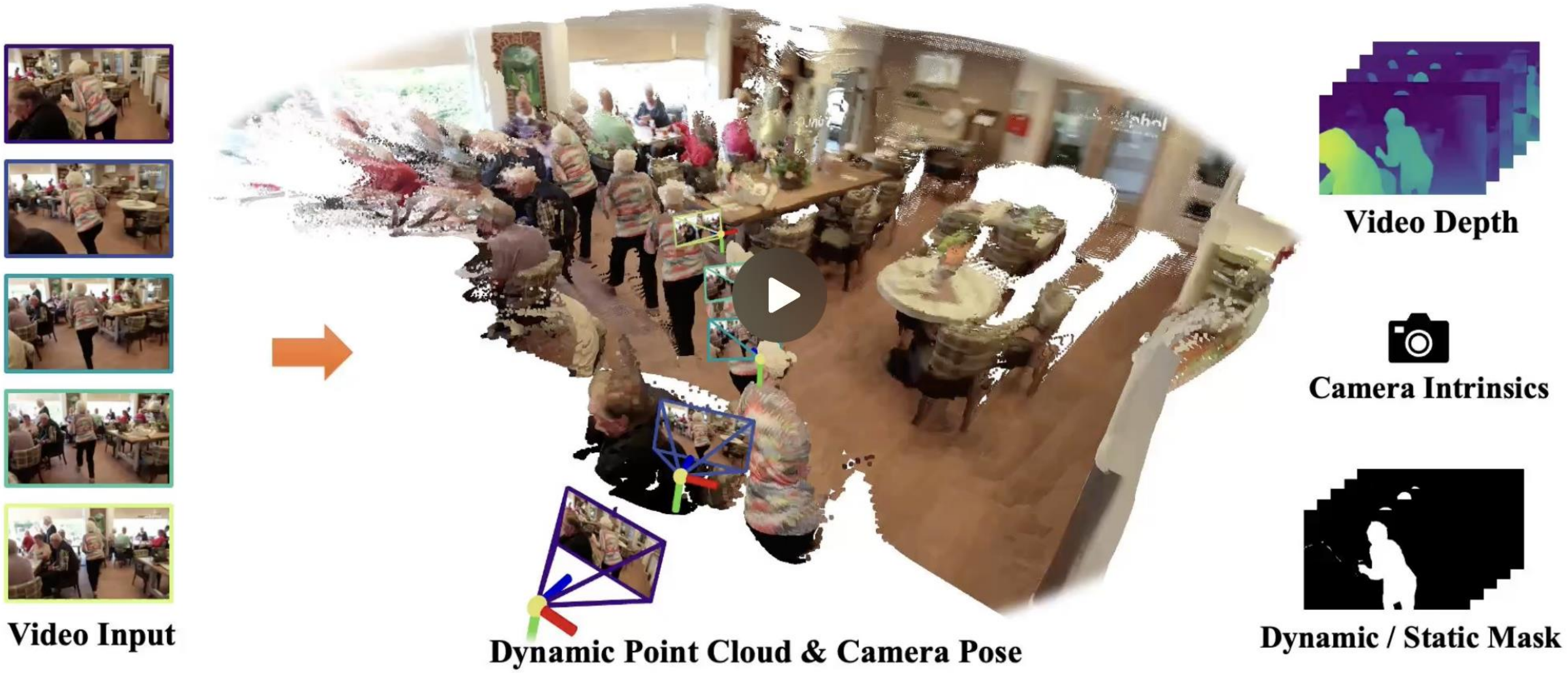
Common pitfalls

- No data set.
  - No training set.
  - No test set.
  - Don’t try to build your own data set!
  - You should have a data set in mind before you submit your proposal.
- Problem is too hard:
  - GANS, speech recognition, voice synthesis, etc. We will give you feedback about this if we think it’s too hard.
- Get started too late.

Get Feedback

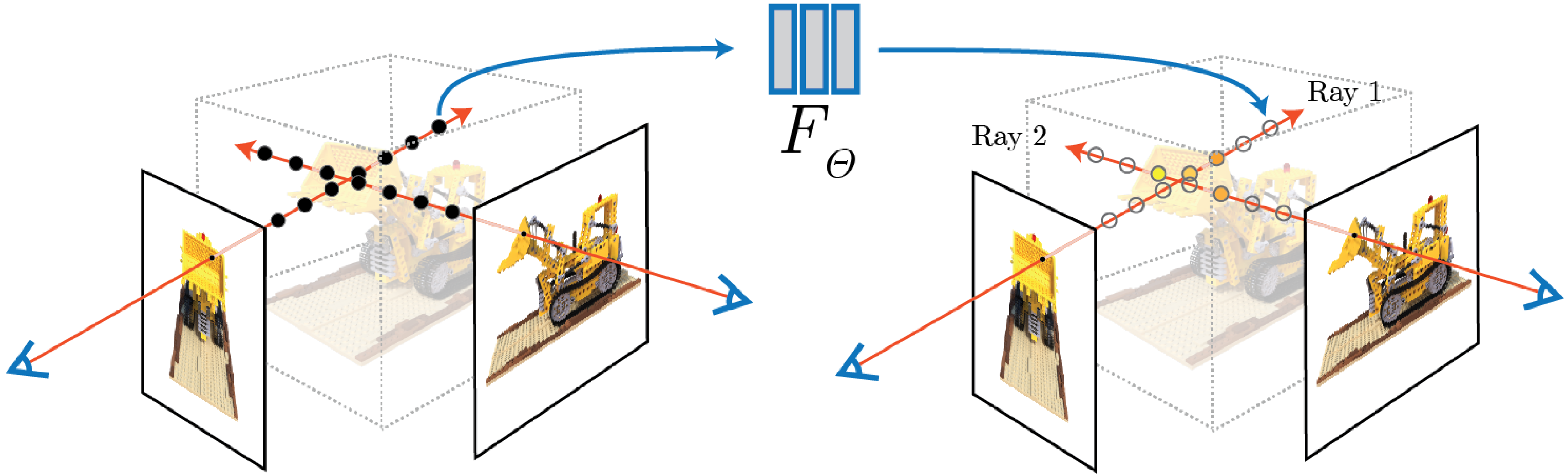
- Follow these guidelines
- Talk to the Professor if you are not sure about the topic of the topic
- We will provide feedback of your project proposal. We will tell you if we think it is too difficult and you need to scale it back or change subjects.

Project Idea 1: Scene reconstruction from egocentric videos



https://monst3r-project.github.io

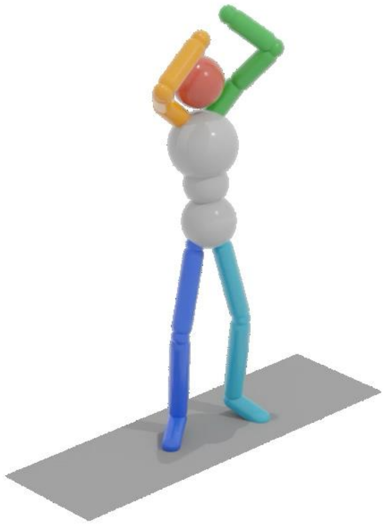
Project Idea 2: Volume rendering (e.g., 3D Gaussian Splatting) meets diffusion



What if we construct a feature field instead of radian field and use a pre-trained diffusion model to refine the rendered feature map.

Project Idea 3: Autoregressive diffusion for human motion generation

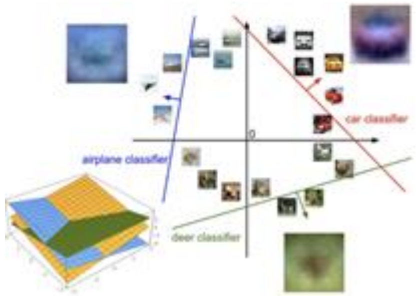
A person waves with both arms above head.



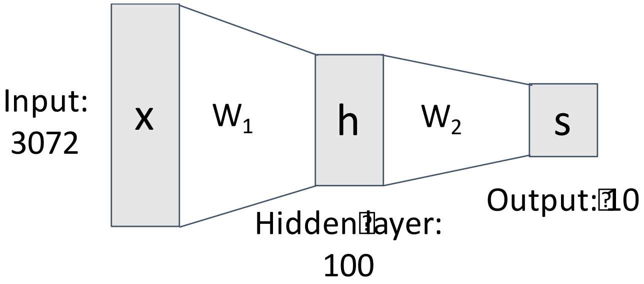
https://neu-vi.github.io/MARDM/

Recap

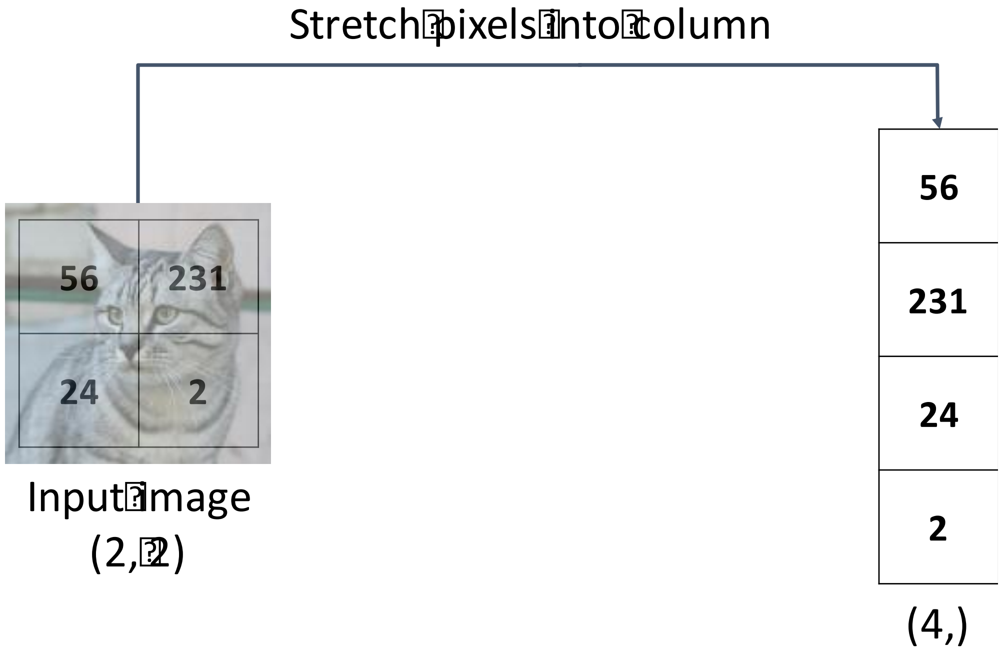
f(x,W) = Wx



$$f = W_2 \max(0, W_1 x)$$



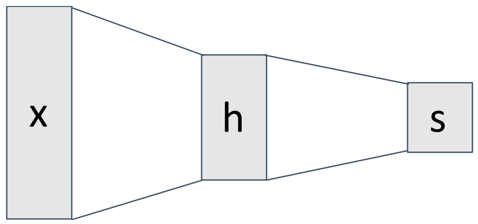
Problem: So far our classifiers don’t respect the spatial structure of images!



Slide credit: D Fouhey & J Johnson

Components of a Convolutional Network

Fully-Connected Layers

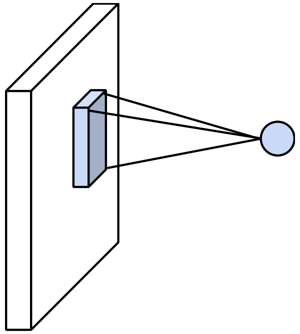


$$y = Wx + b$$

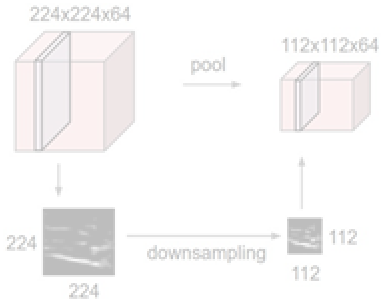
Activation Function

$$y = \max(0, x)$$

Convolution Layers



Pooling Layers



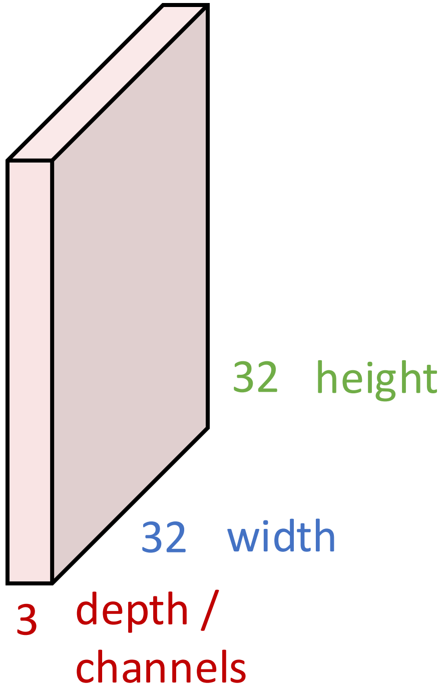
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

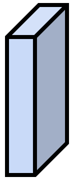
Slide credit: D Fouhey & J Johnson

Convolution Layer

3x32x32 image

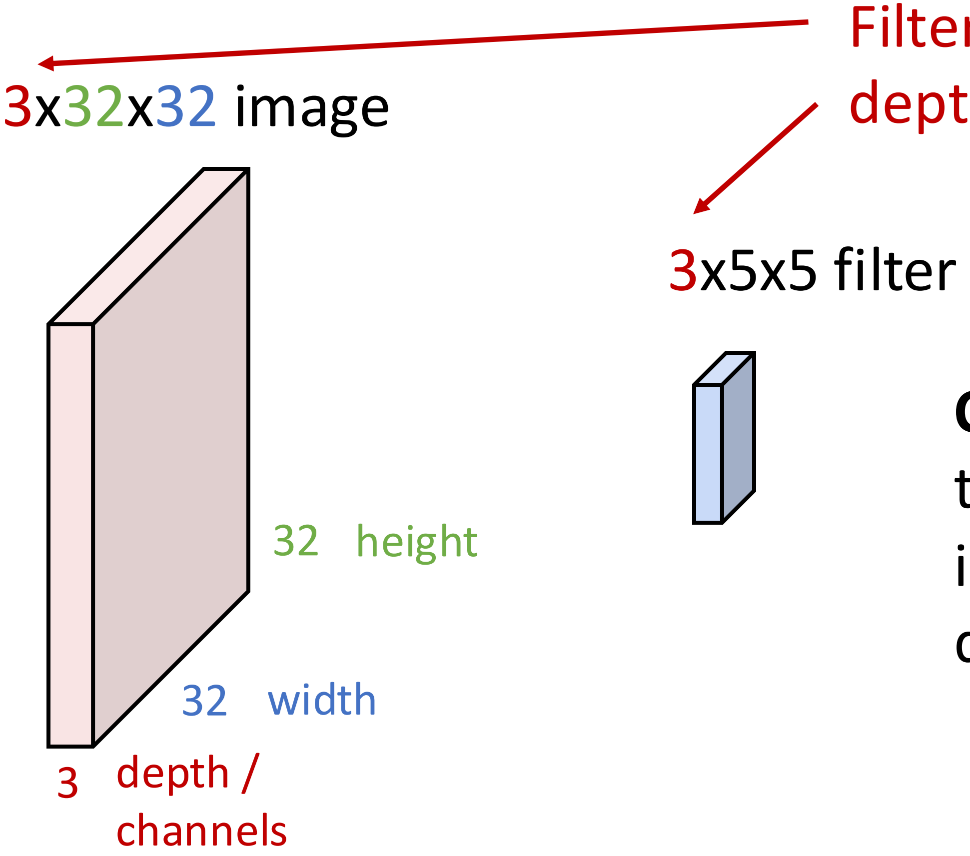


3x5x5 filter



Convolve the filter with the image: "slide over the image spatially, computing dot products"

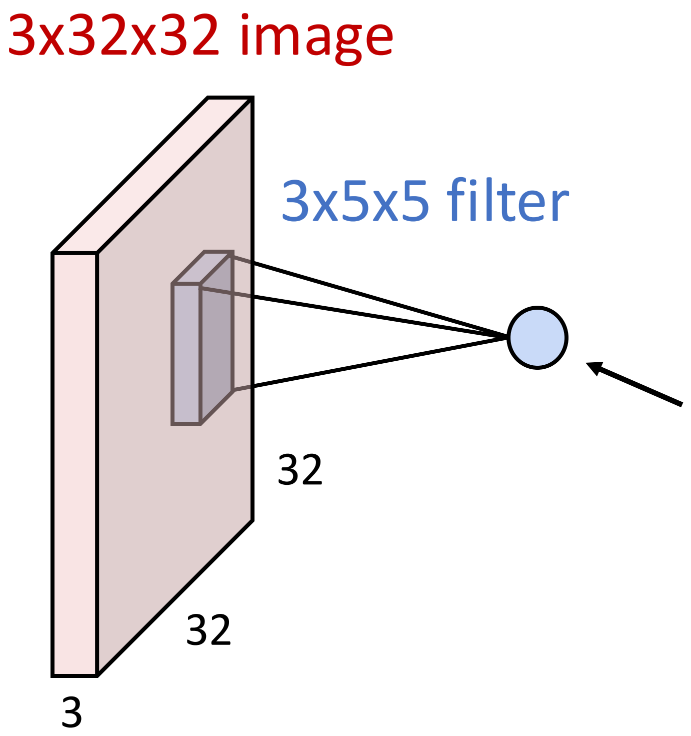
Convolution Layer



Filters always extend the full depth of the input volume

Convolve the filter with the image: “slide over the image spatially, computing dot products”

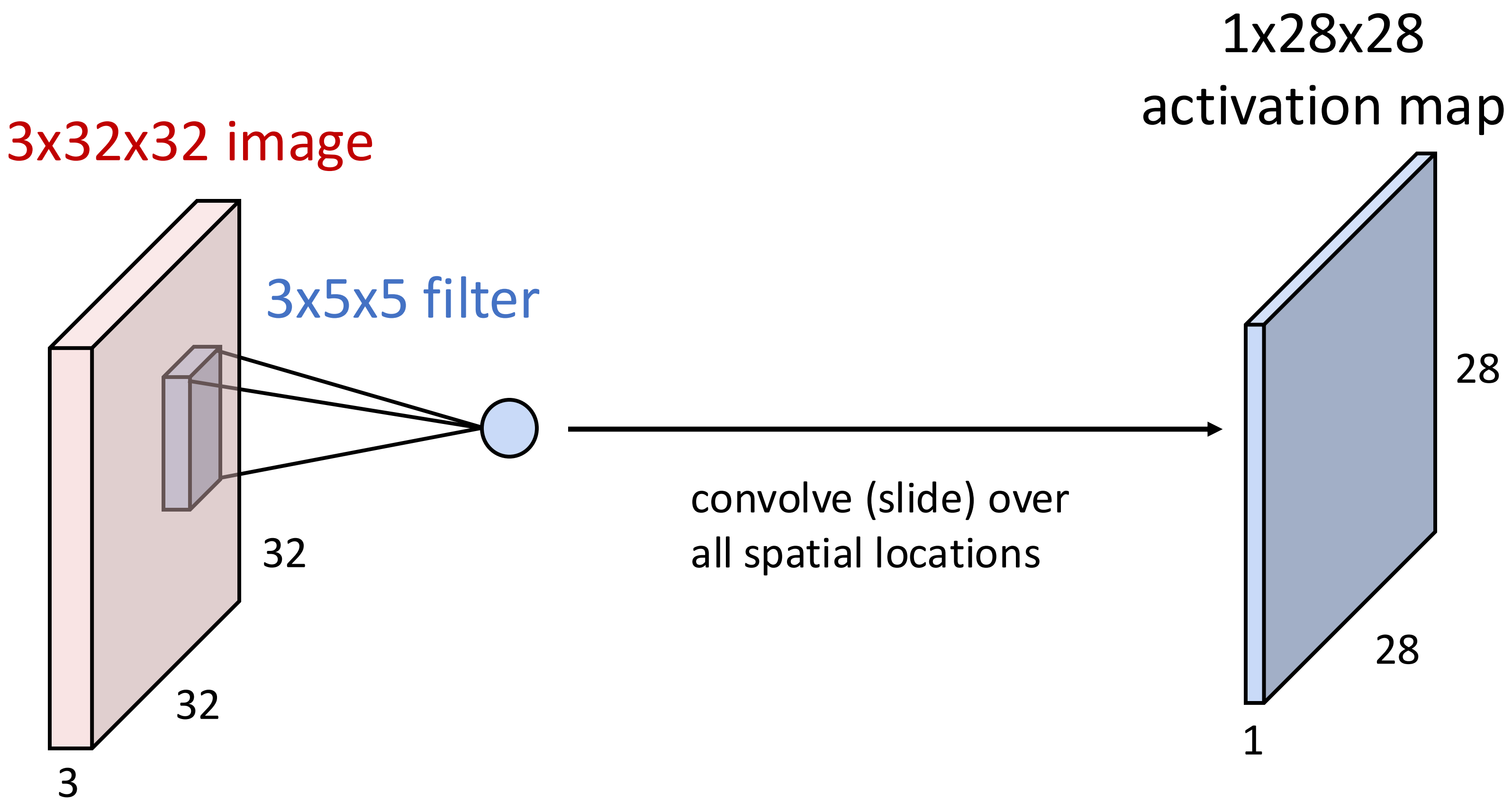
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

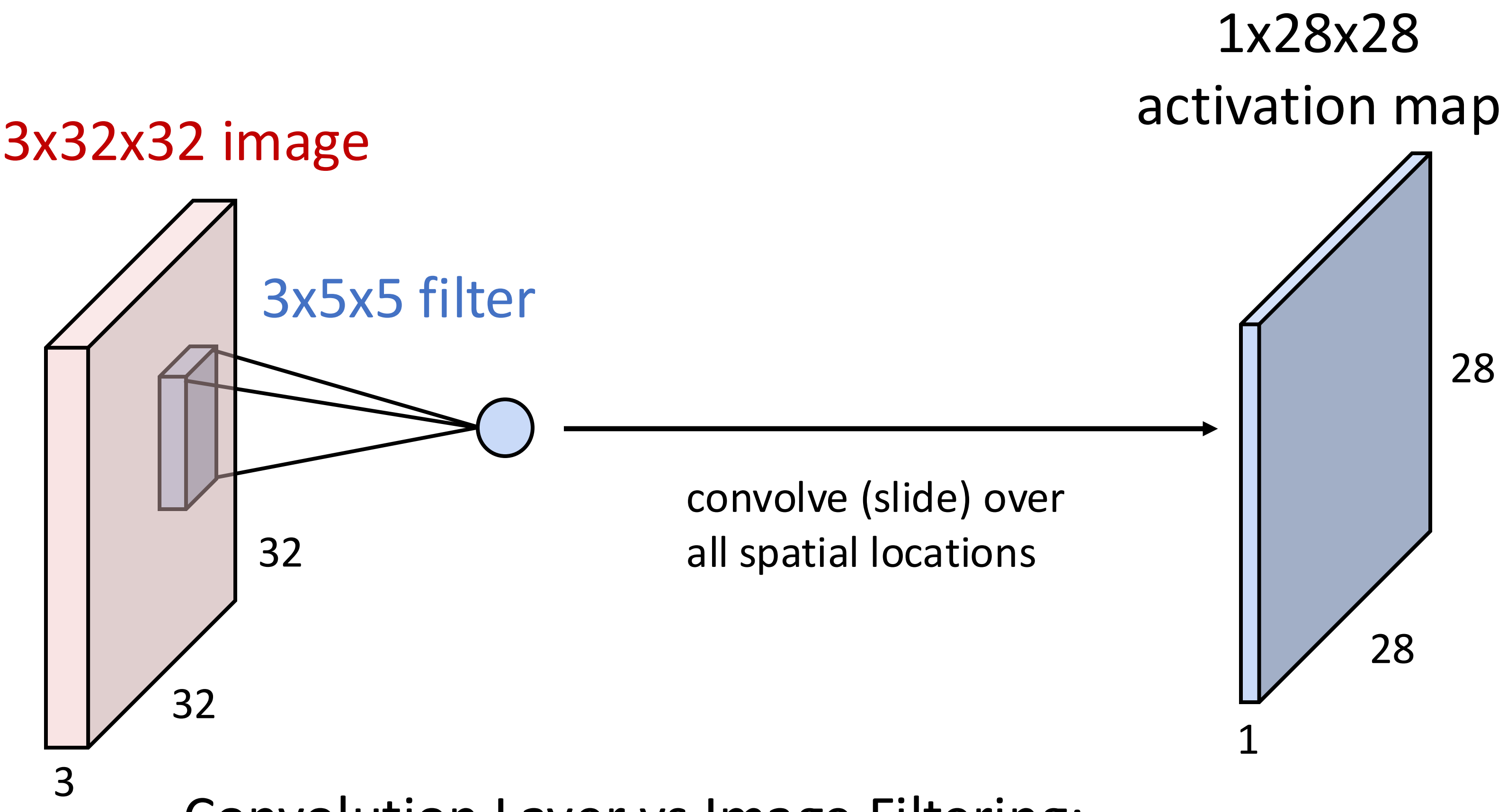
$$w^T x + b$$

Convolution Layer



Slide credit: D Fouhey & J Johnson

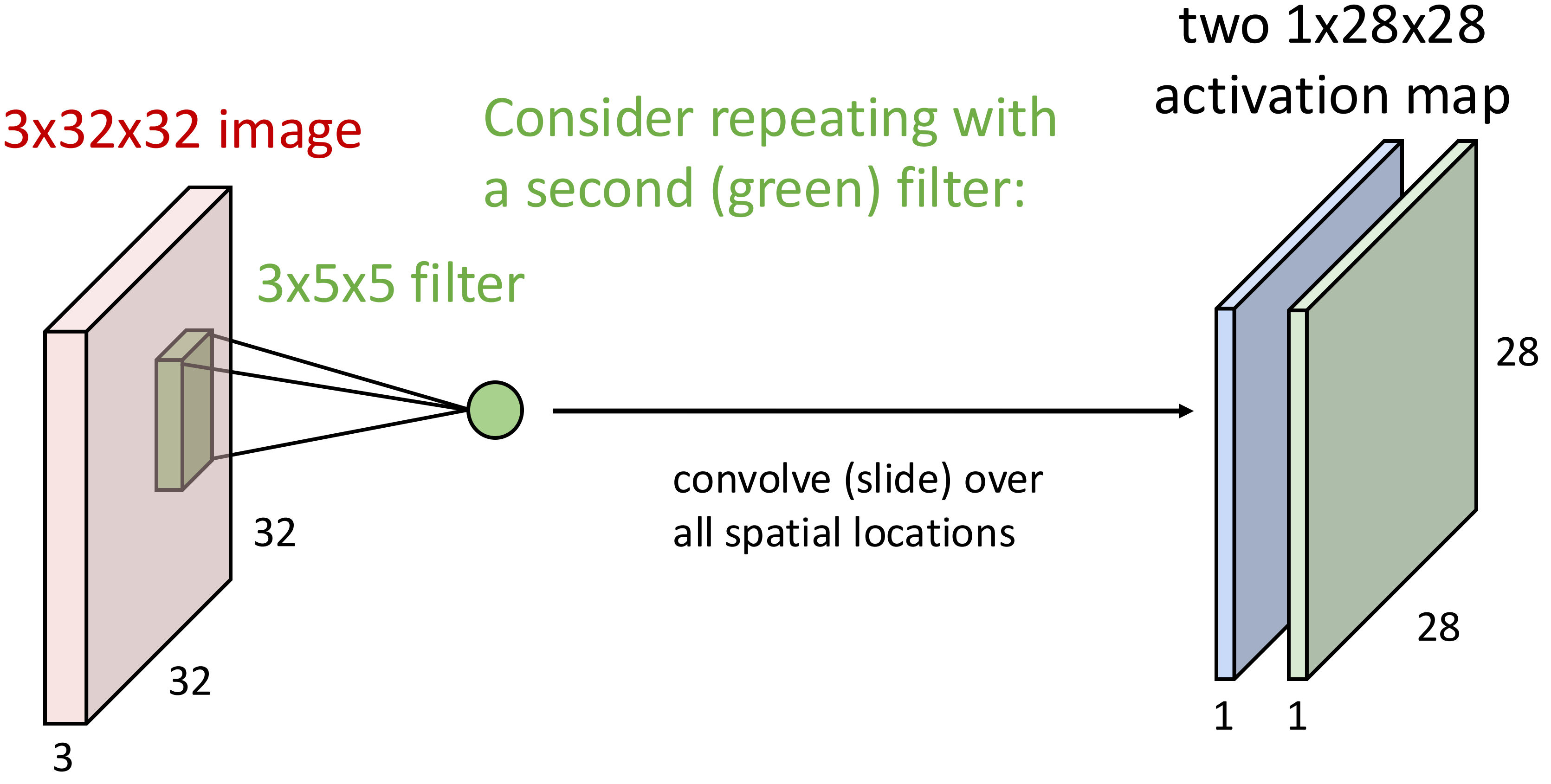
Convolution Layer



Convolution Layer vs Image Filtering:
>1 input and output channels
Forget about convolution vs cross-correlation

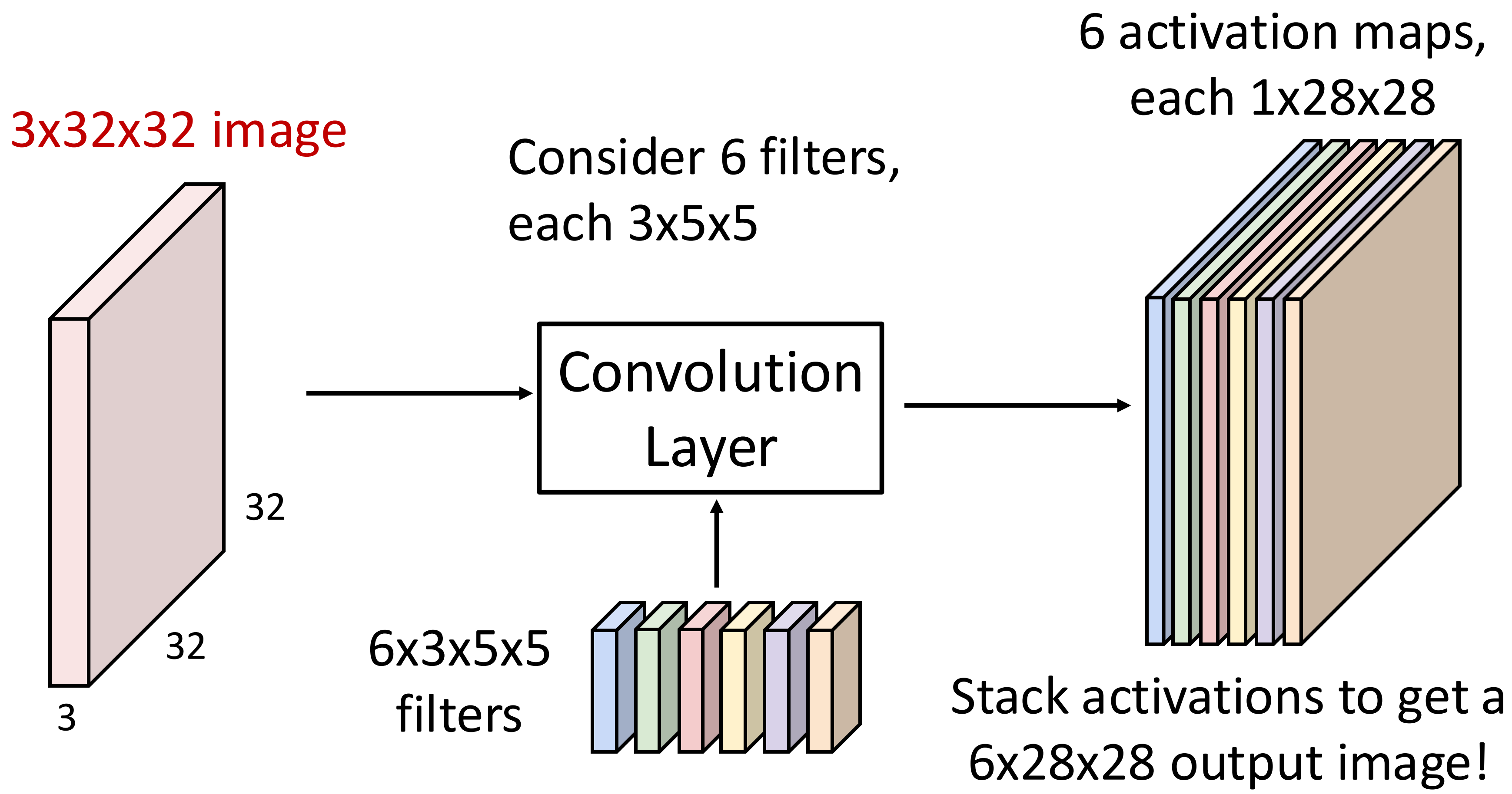
Convolution Layer

Consider repeating with a second (green) filter:

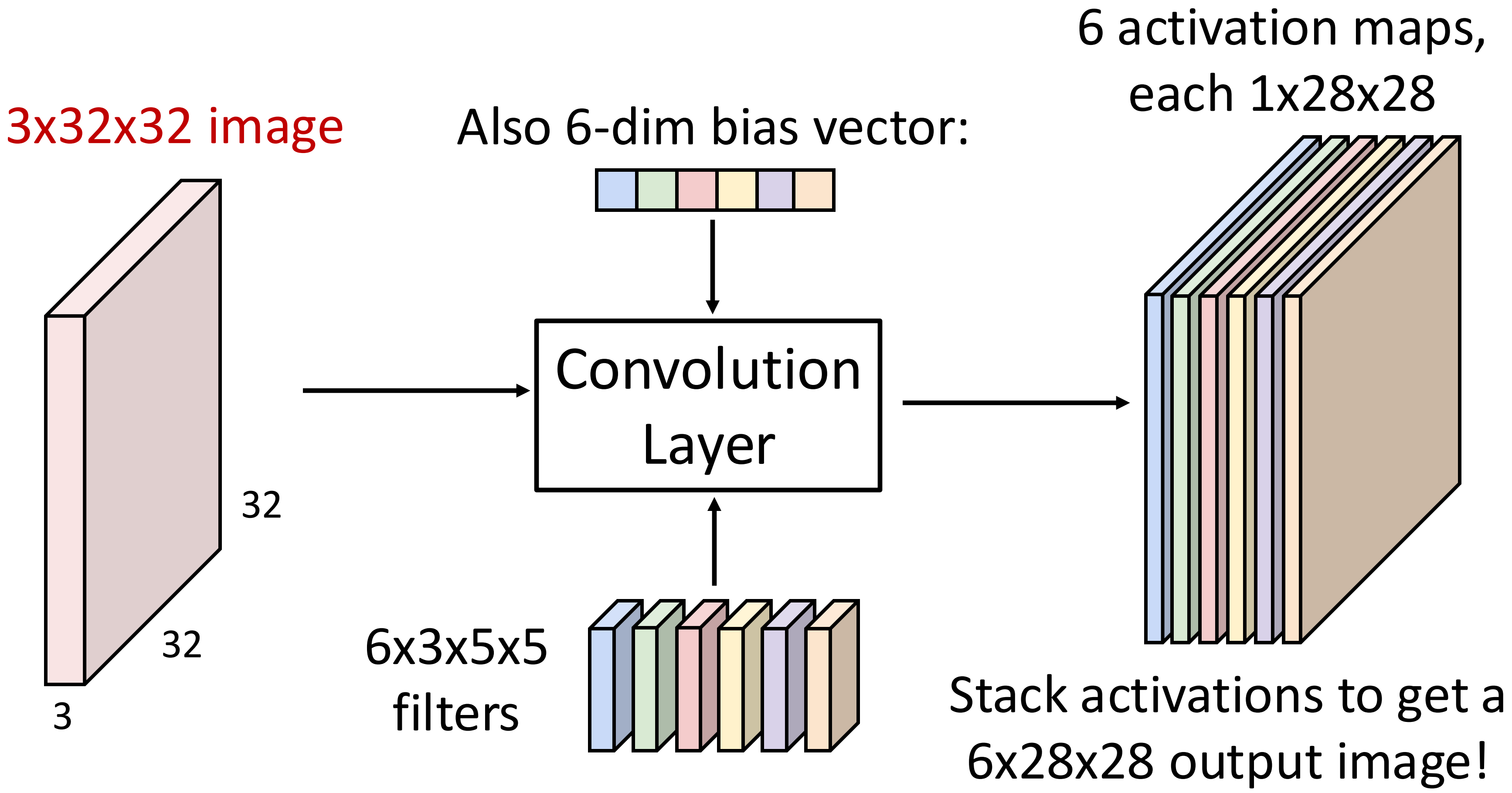


Slide credit: D Fouhey & J Johnson

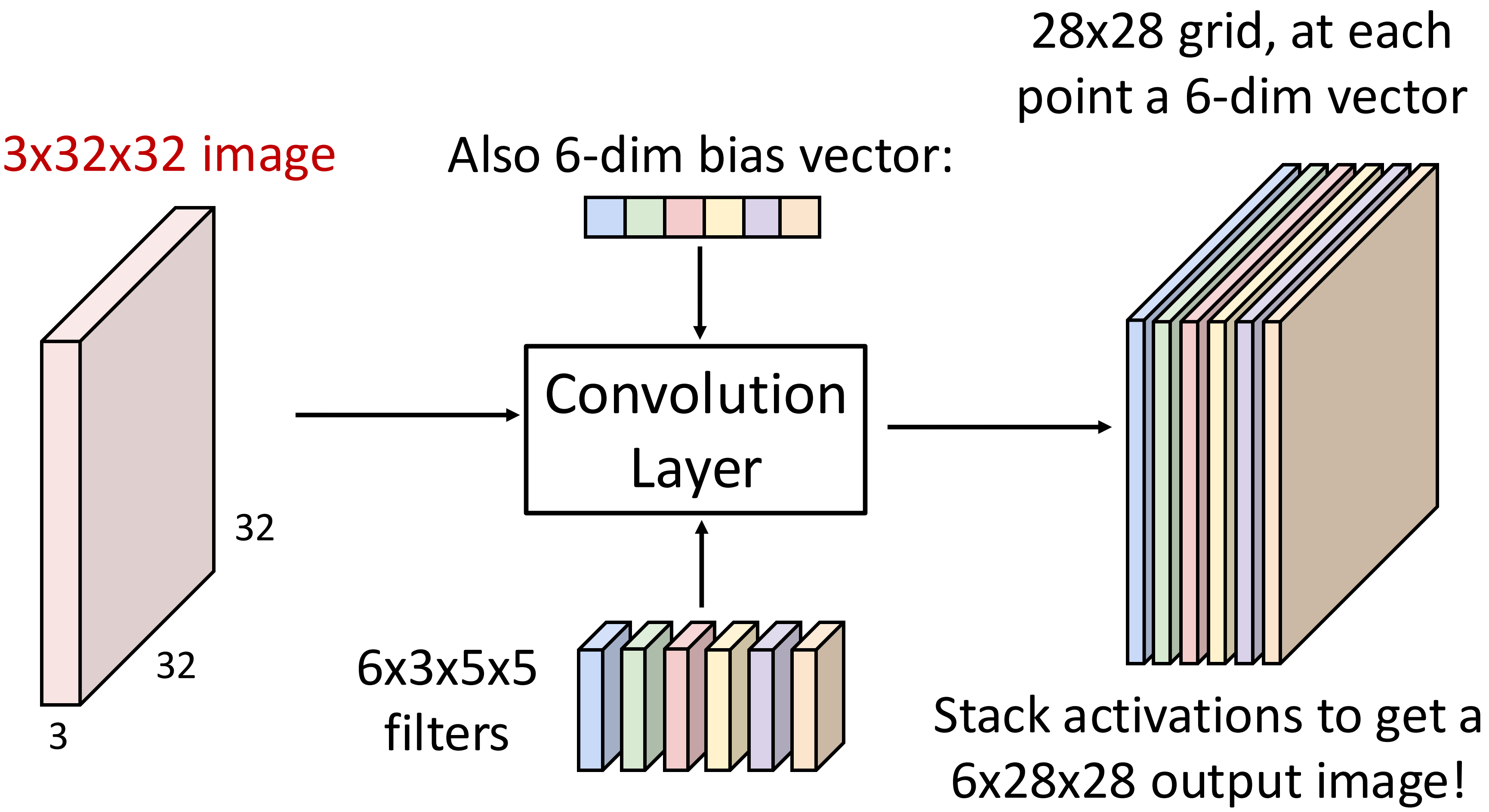
Convolution Layer



Convolution Layer

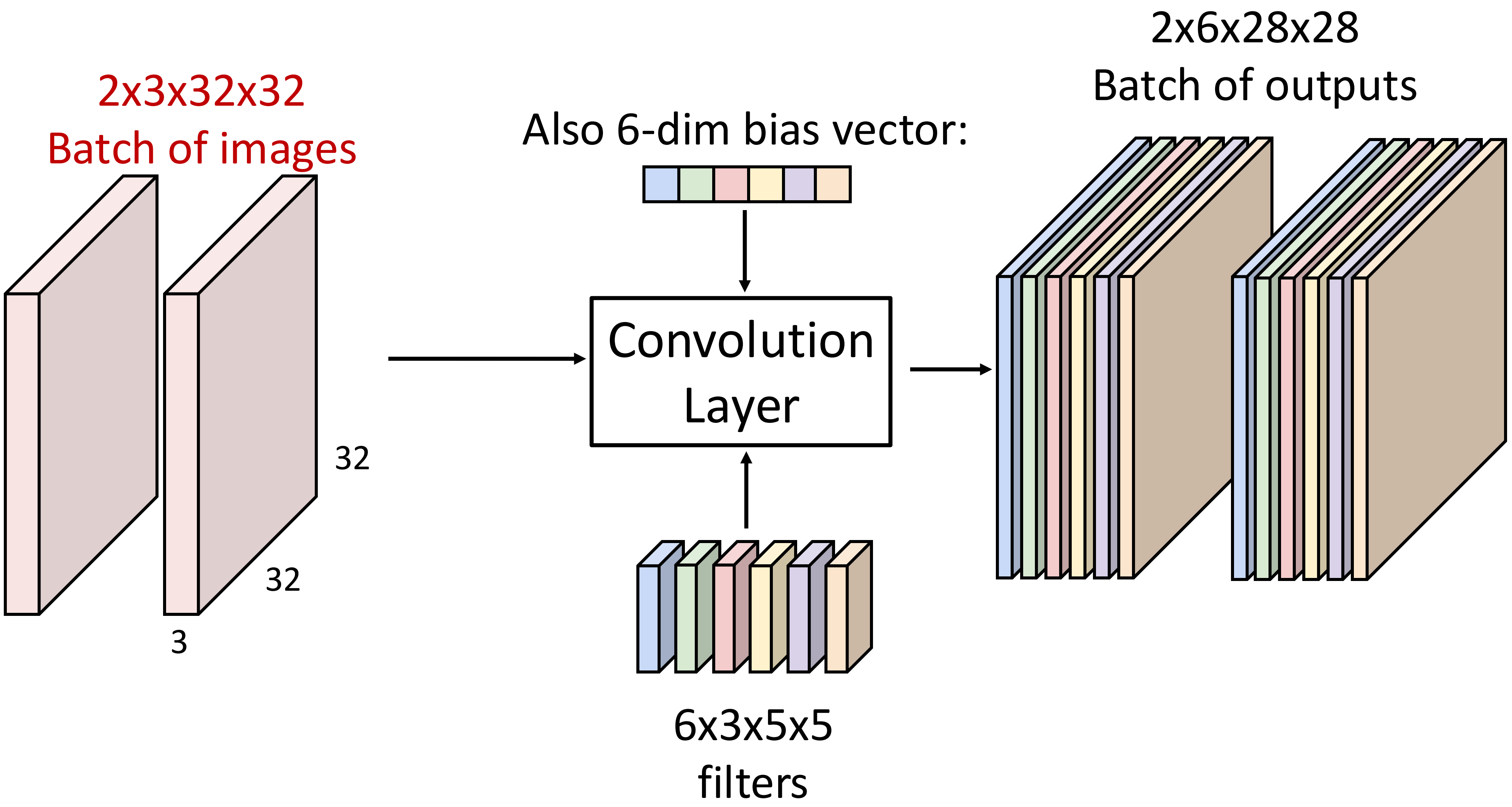


Convolution Layer



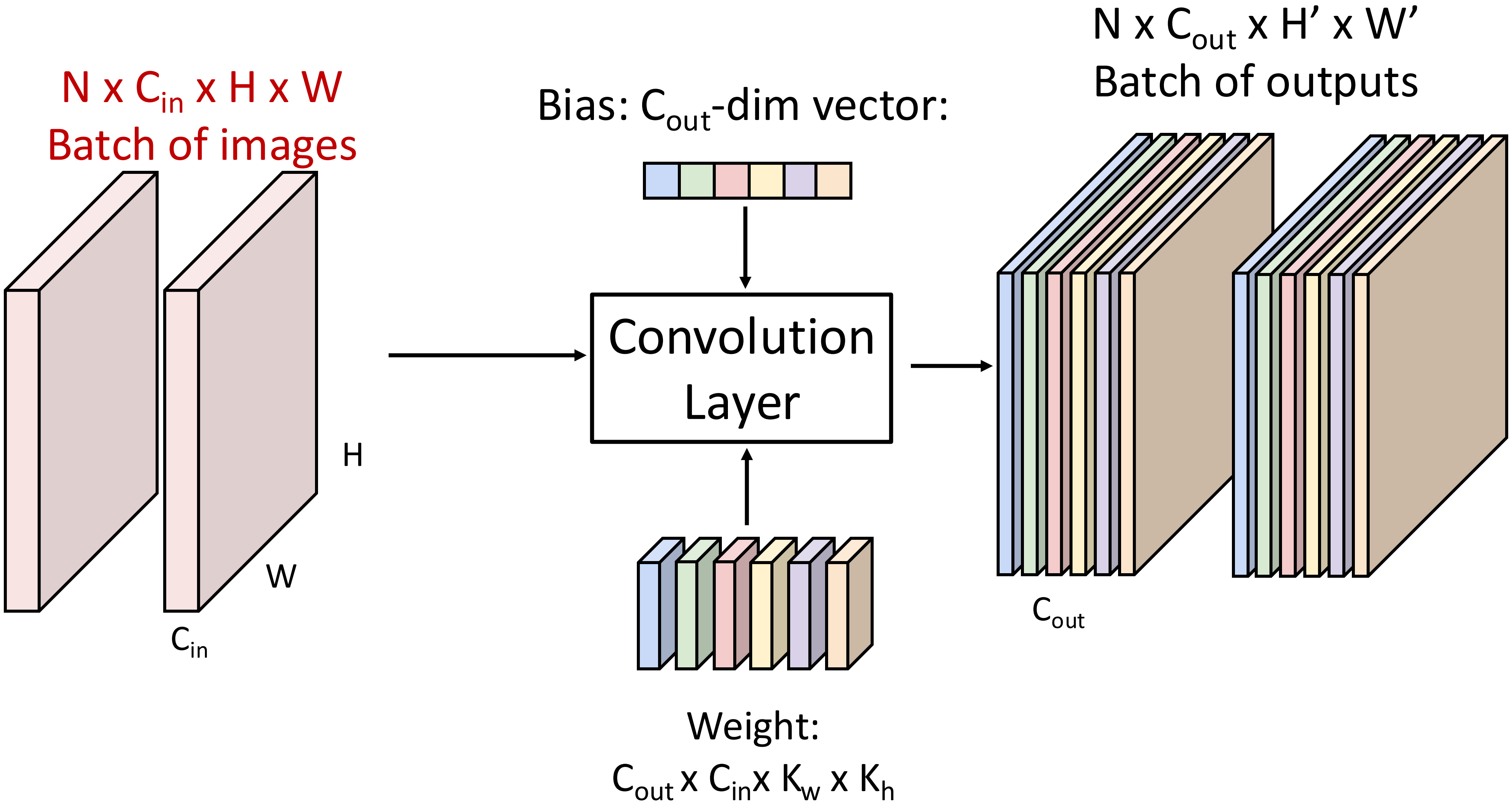
Slide credit: D Fouhey & J Johnson

Convolution Layer



Slide credit: D Fouhey & J Johnson

Convolution Layer



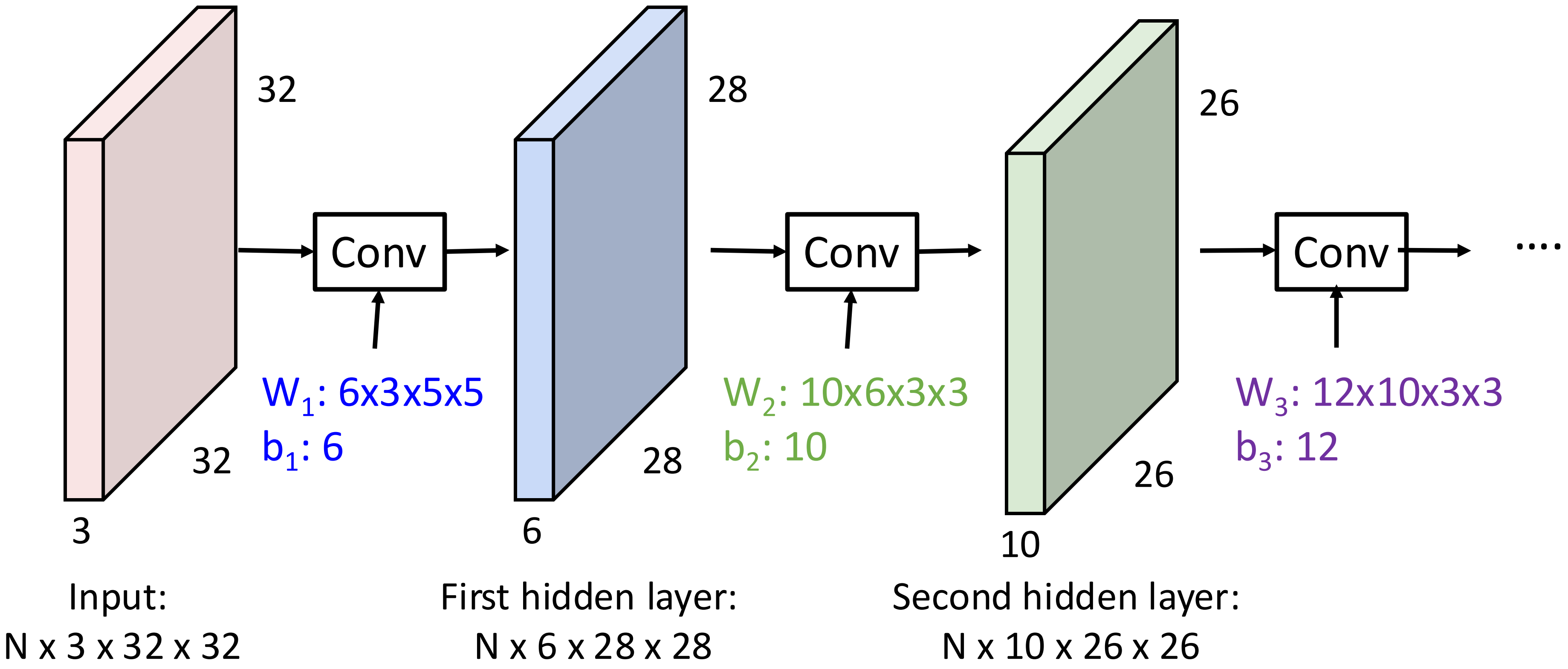
Slide credit: D Fouhey & J Johnson

Stacking Convolutions

(Recall $y=W_2W_1x$ is a linear classifier)

Q: What happens if we stack two convolution layers?

A: It's equivalent to just one convolution layer!

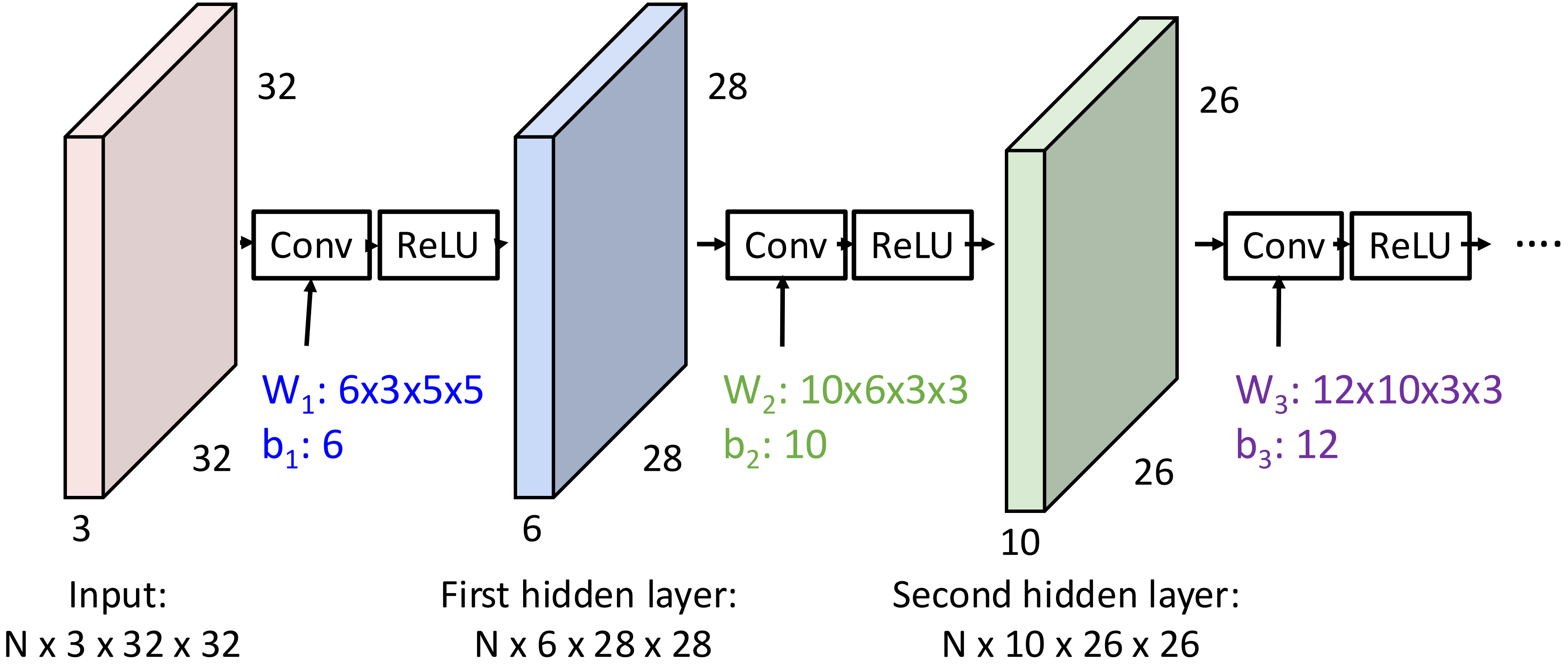


Stacking Convolutions

(Recall $y=W_2W_1x$ is a linear classifier)

Q: What happens if we stack two convolution layers?

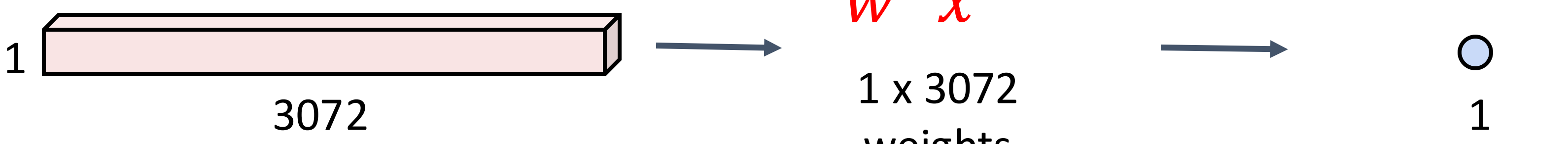
A: It's equivalent to just one convolution layer!



Solution: Add a nonlinearity between each conv layer

Slide credit: D Fouhey & J Johnson

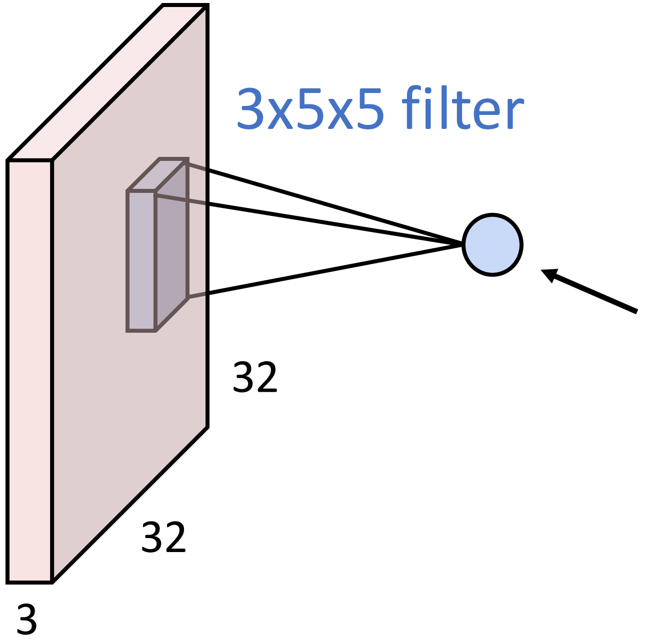
Fully-connected Layer vs Convolution Layer



$w^T x$

1 x 3072
weights

3x32x32 image



$w^T x$

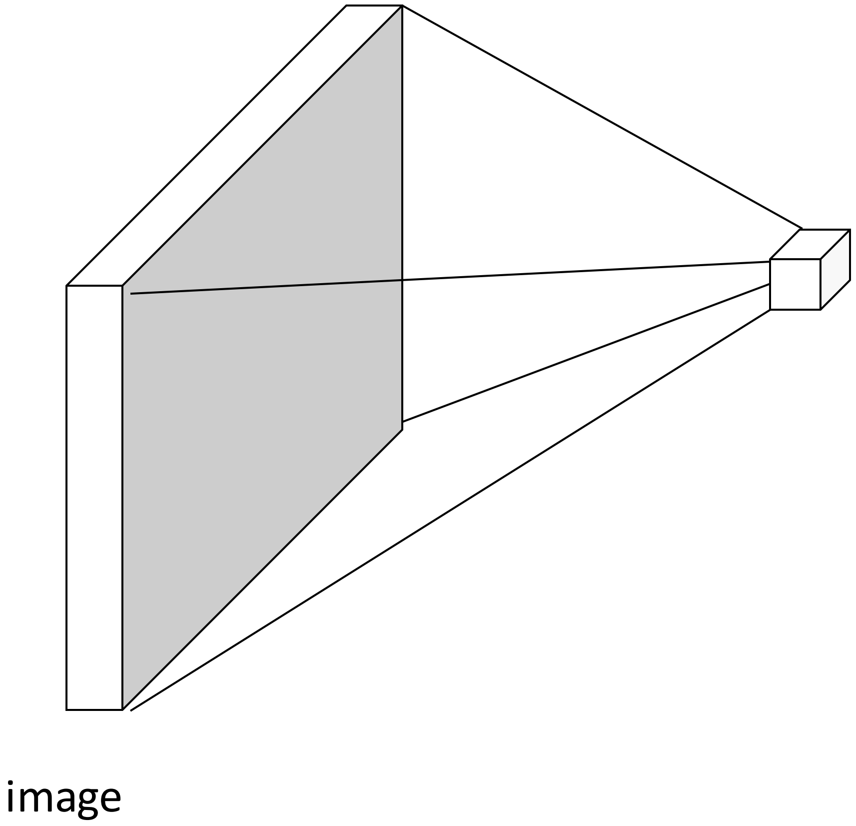
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image

What if the kernel size is 32*32? What is the number of parameters?

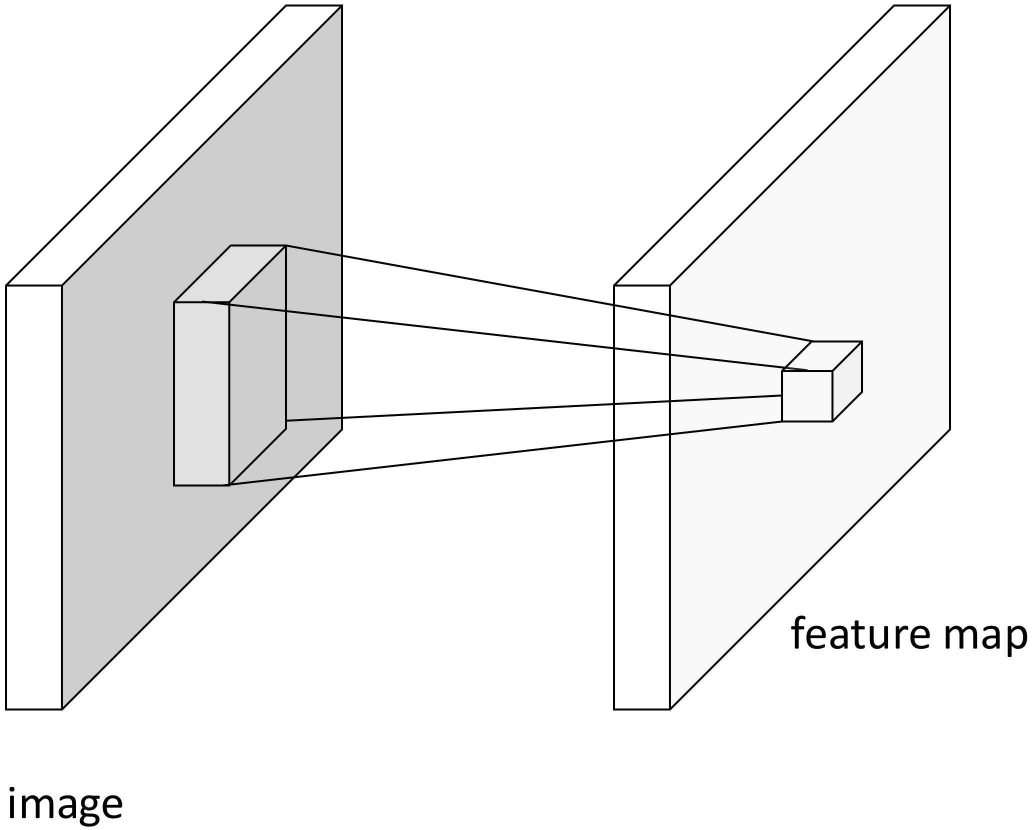
A fully-connected layer is equivalent to a convolution layer.

Fully-connected Layer vs Convolution Layer

Fully connected layer



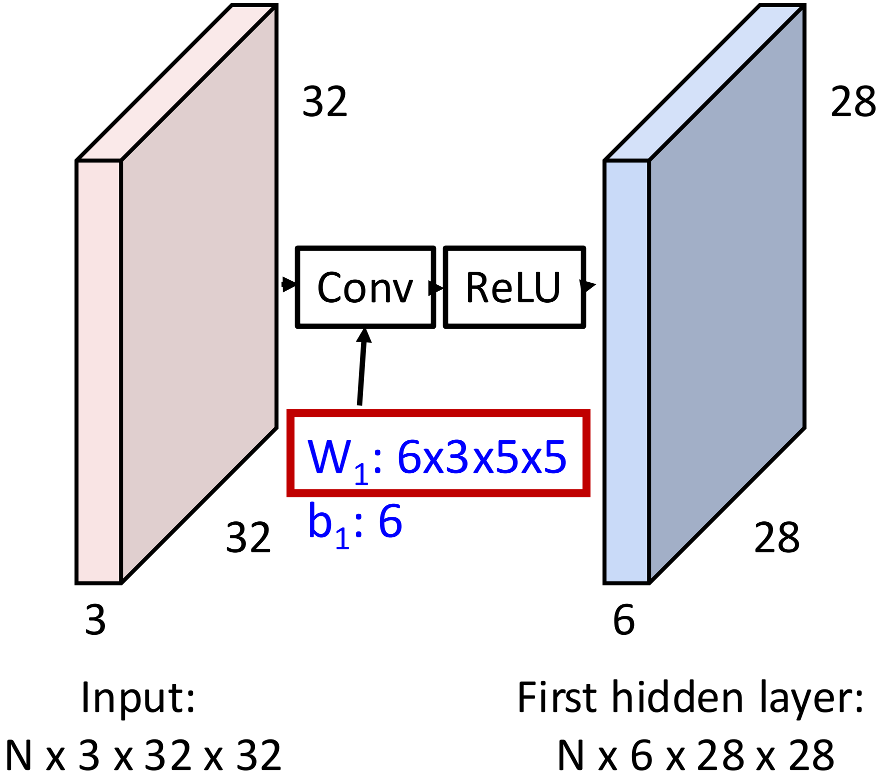
Convolution layer



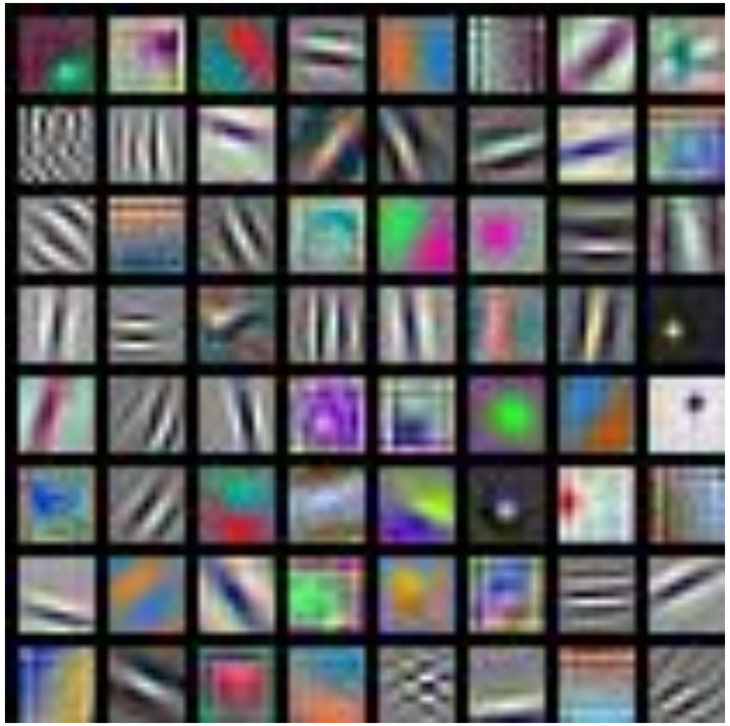
Today’s Class

- More about convolution layer

What do Conv Filters Learn?

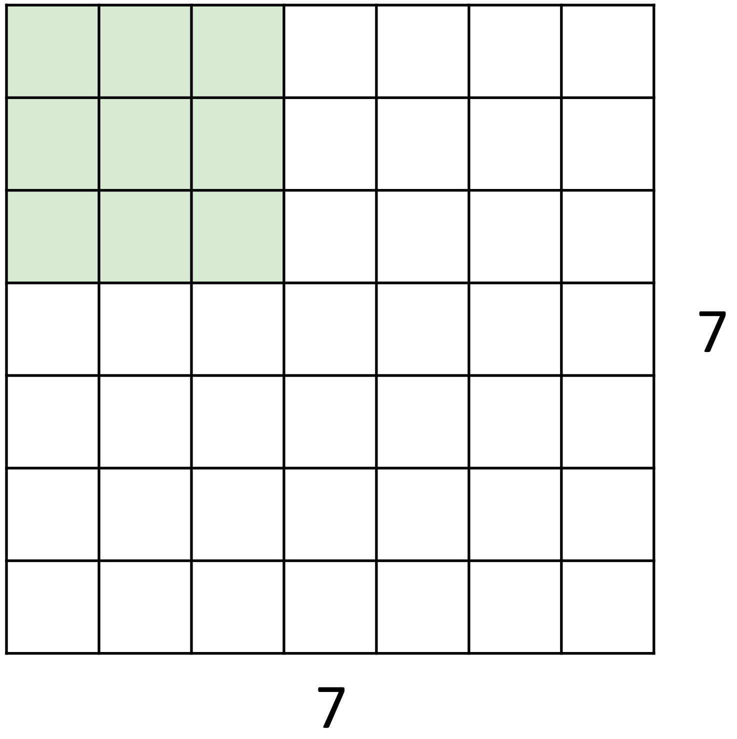


First-layer conv filters: local image templates (Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11

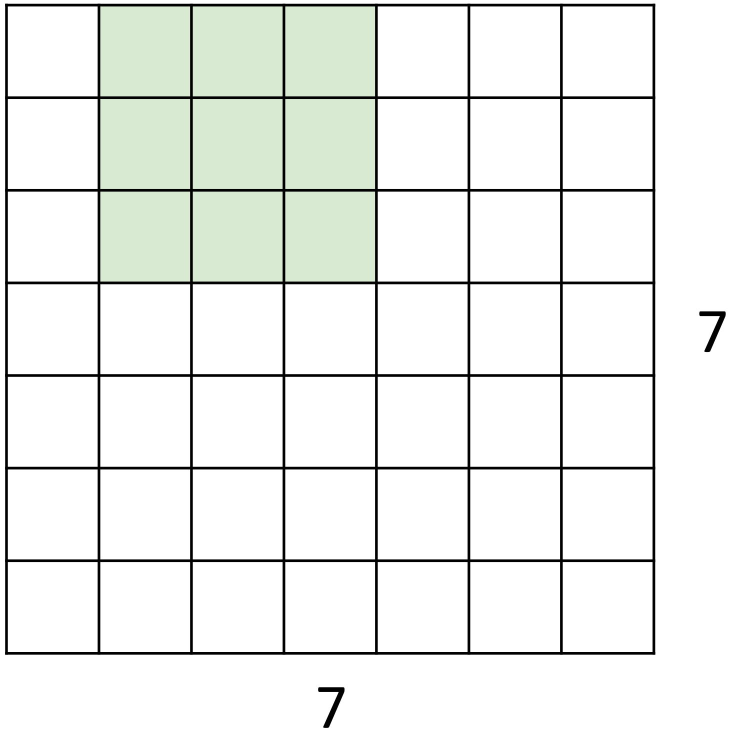
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

Convolution Spatial Dimensions

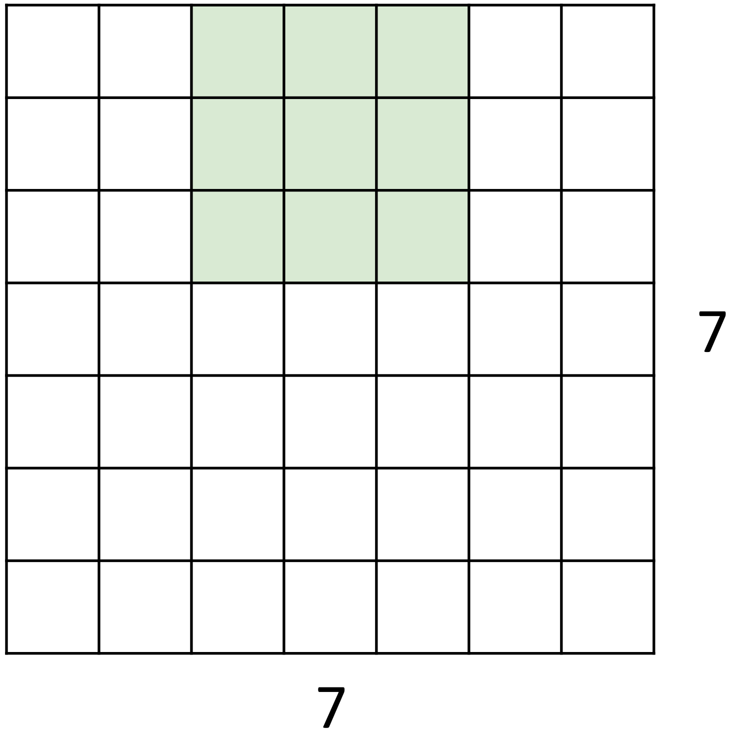


Input: 7x7
Filter: 3x3

Q: How big is output?

Slide credit: D Fouhey & J Johnson

Convolution Spatial Dimensions

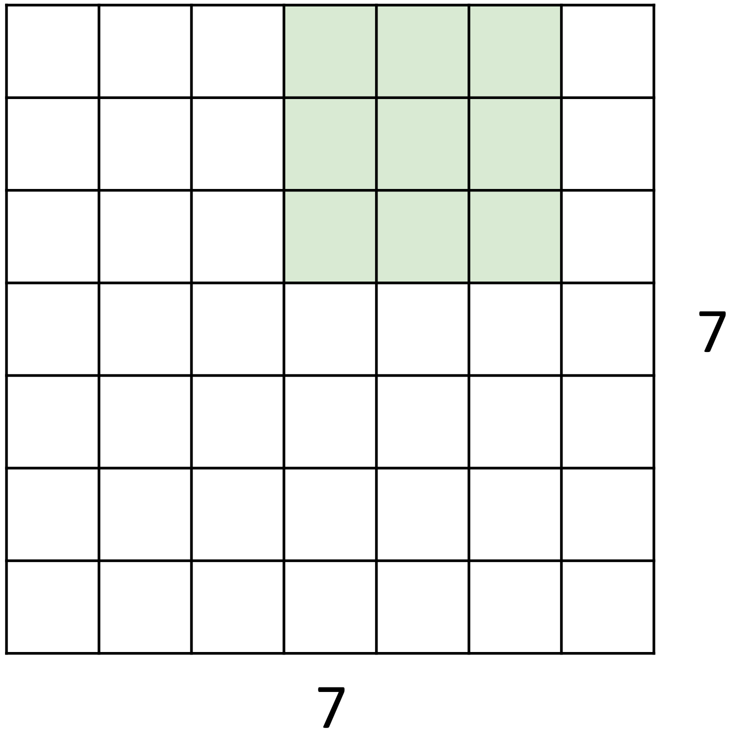


Input: 7x7
Filter: 3x3

Q: How big is output?

Slide credit: D Fouhey & J Johnson

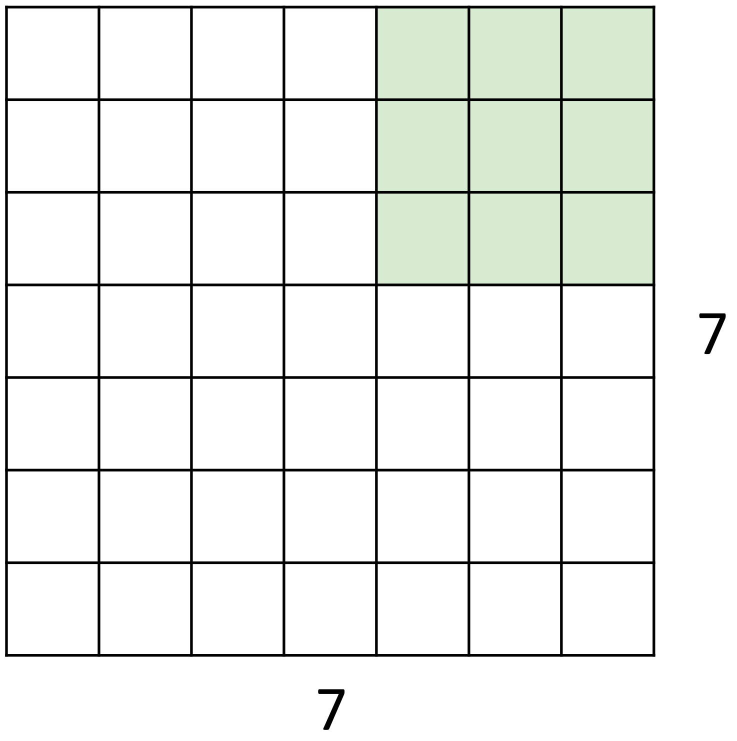
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3

Q: How big is output?

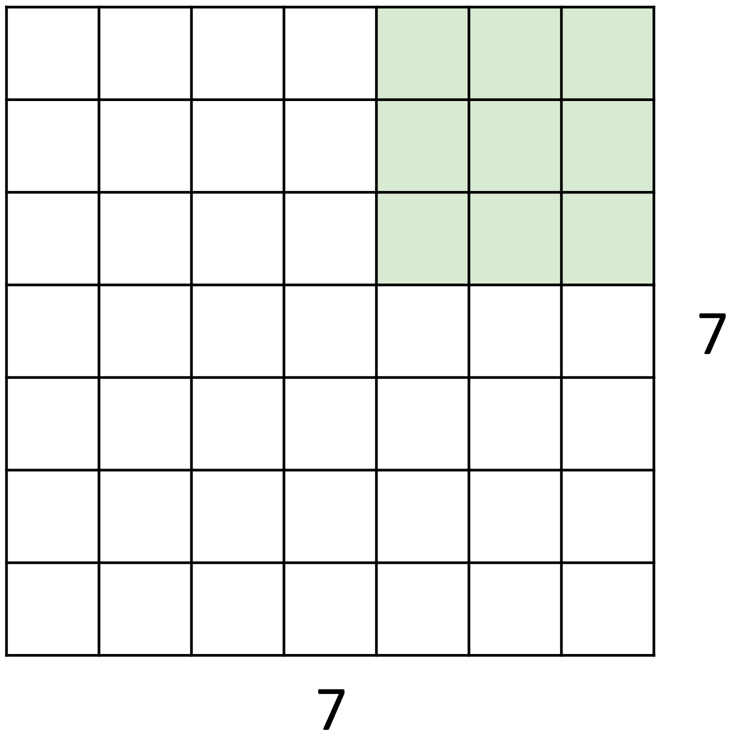
Convolution Spatial Dimensions



Input: 7x7
Filter: 3x3
Output: 5x5

Slide credit: D Fouhey & J Johnson

Convolution Spatial Dimensions

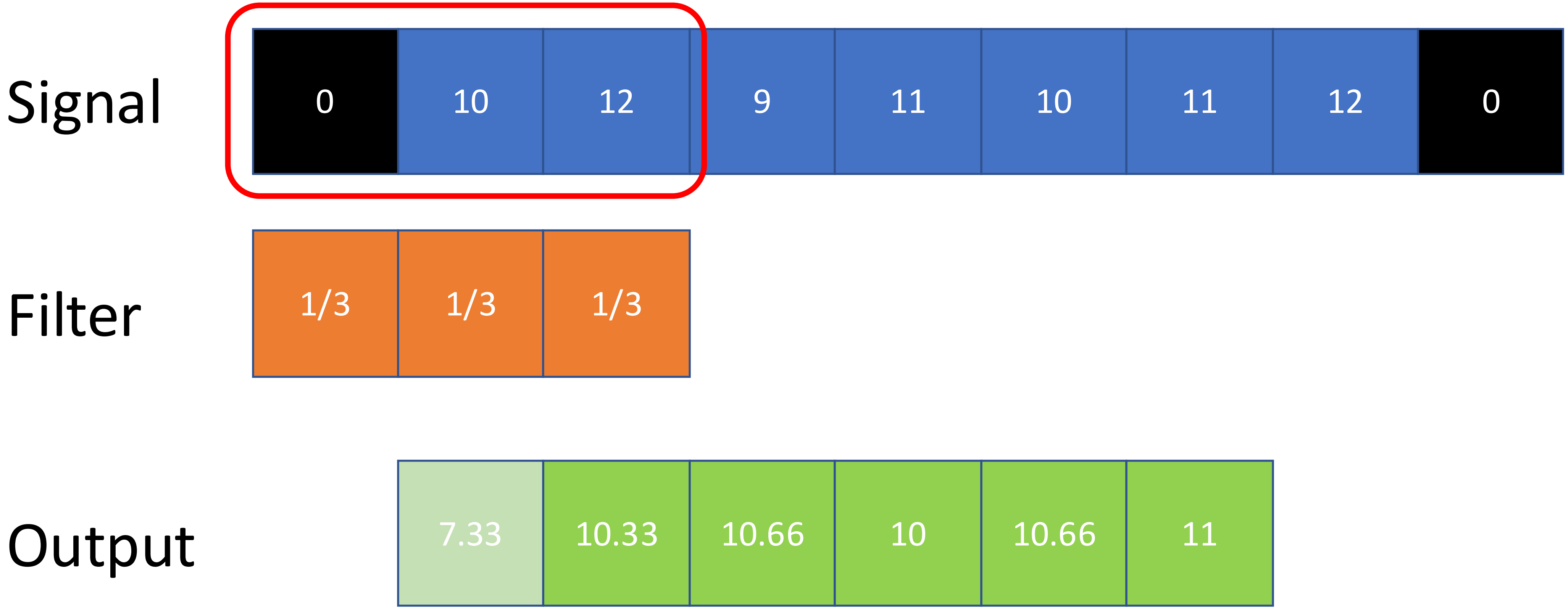


Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Output: W – K + 1

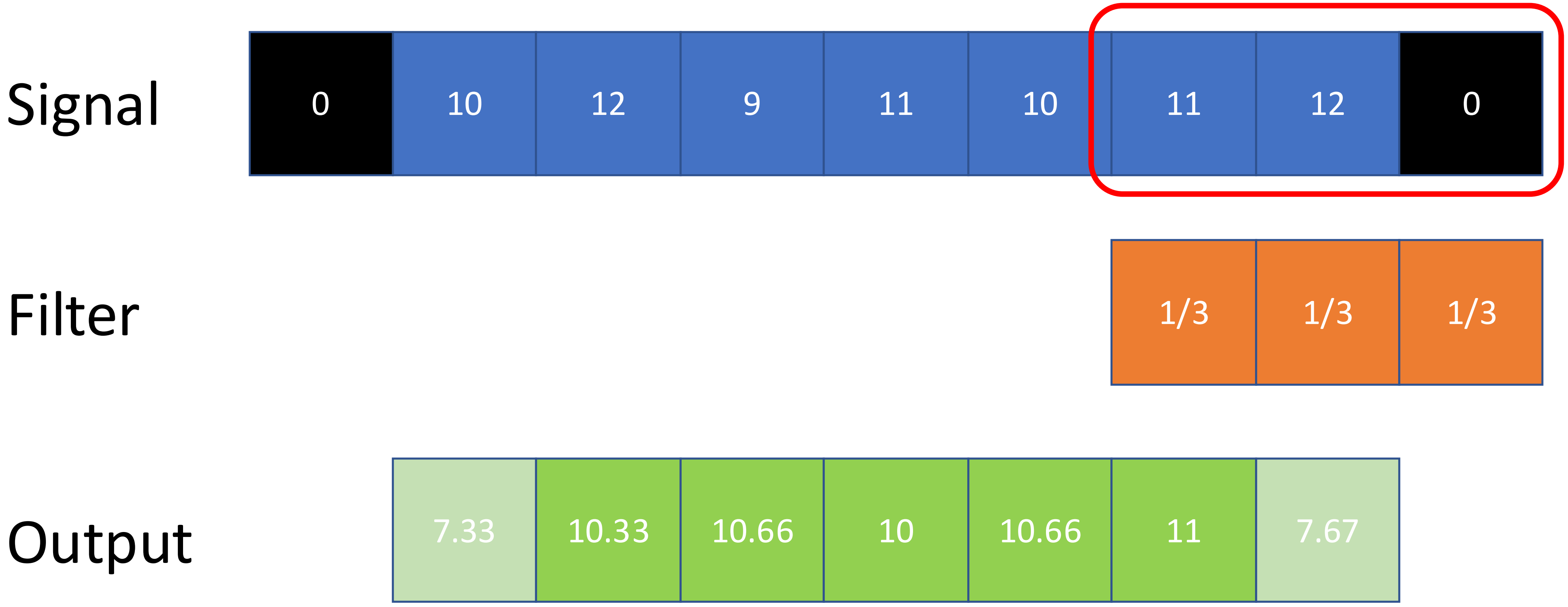
Problem:
Feature maps "shrink" with each layer!

Padding: 1D Case



Slide inspired by : J Johnson

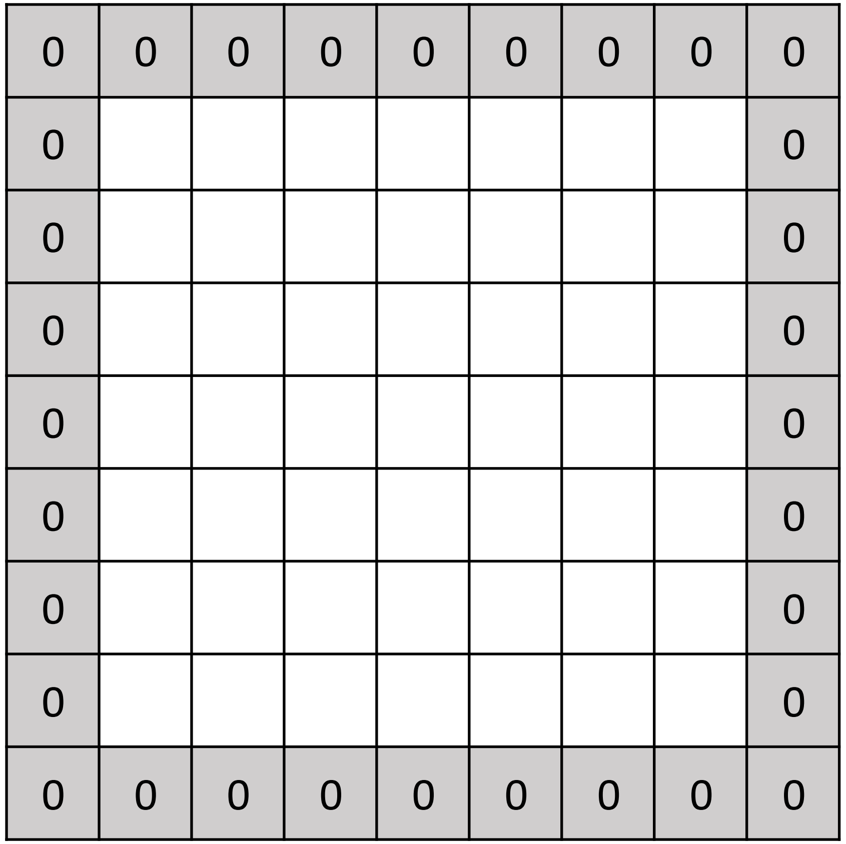
Padding: 1D Case



Input: 7 vs output: 7

Slide inspired by : J Johnson

Convolution Spatial Dimensions



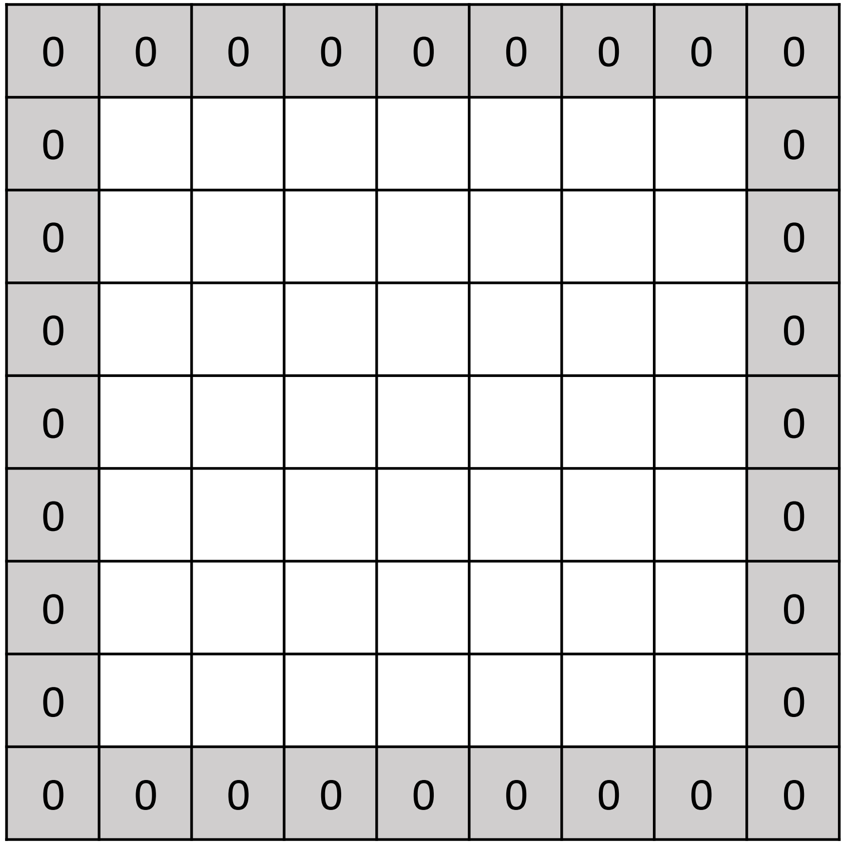
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P

Problem:
Feature maps "shrink" with each layer!

Solution: **padding**
Add zeros around the input

Convolution Spatial Dimensions



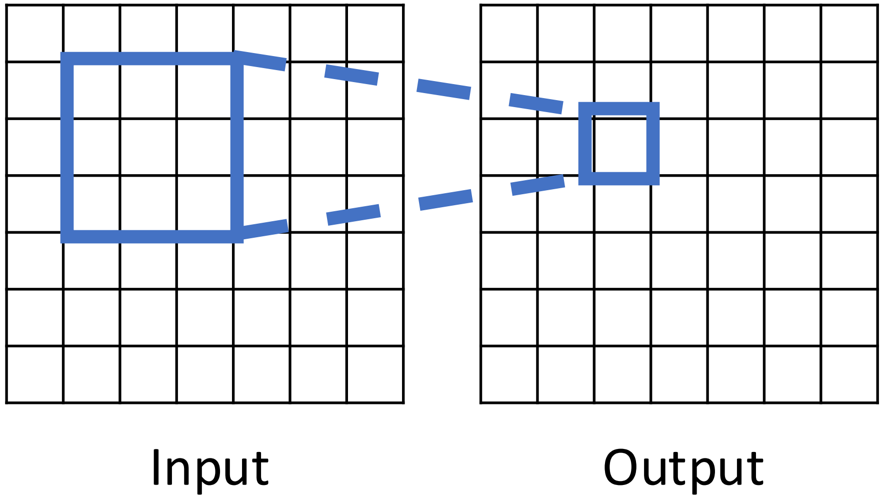
Input: 7x7
Filter: 3x3
Output: 5x5

In general:
Input: W
Filter: K
Padding: P
Output: W – K + 1 + 2P

Very common: “same padding”
Set P = (K – 1) / 2
Then output size = input size

Receptive Fields

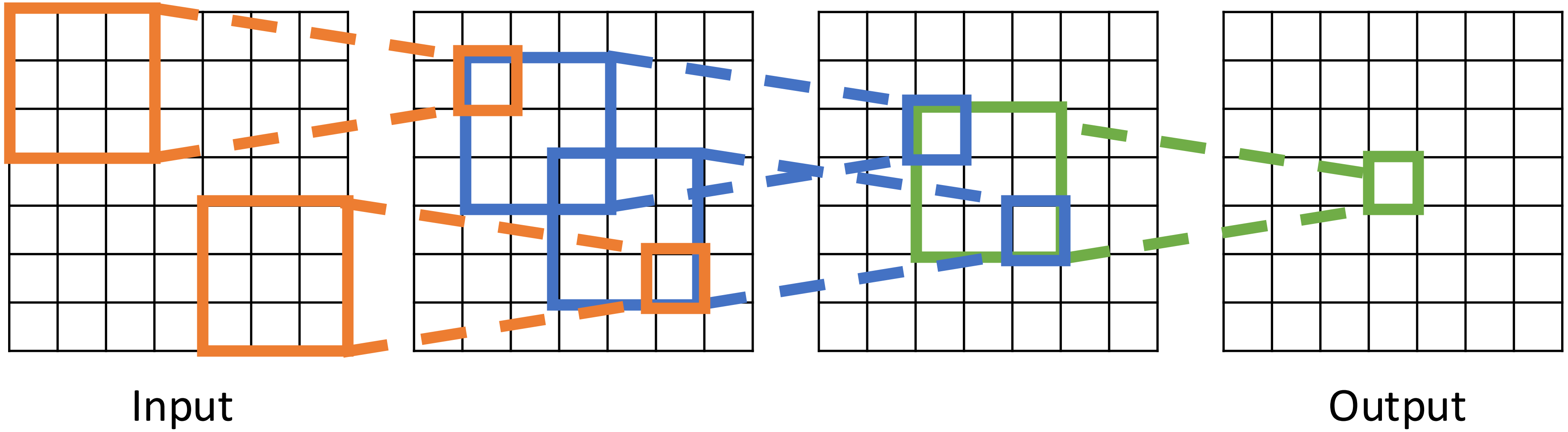
For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Slide credit: D Fouhey & J Johnson

Receptive Fields

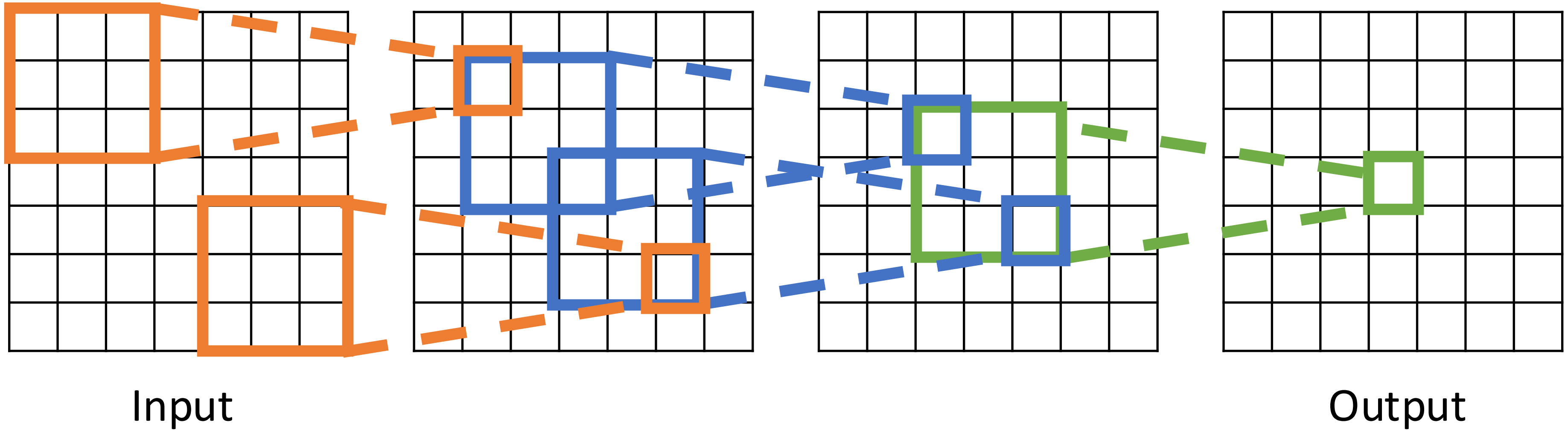
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Careful – ”receptive field in the input”
vs “receptive field in the previous layer”
Hopefully clear from context!

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)

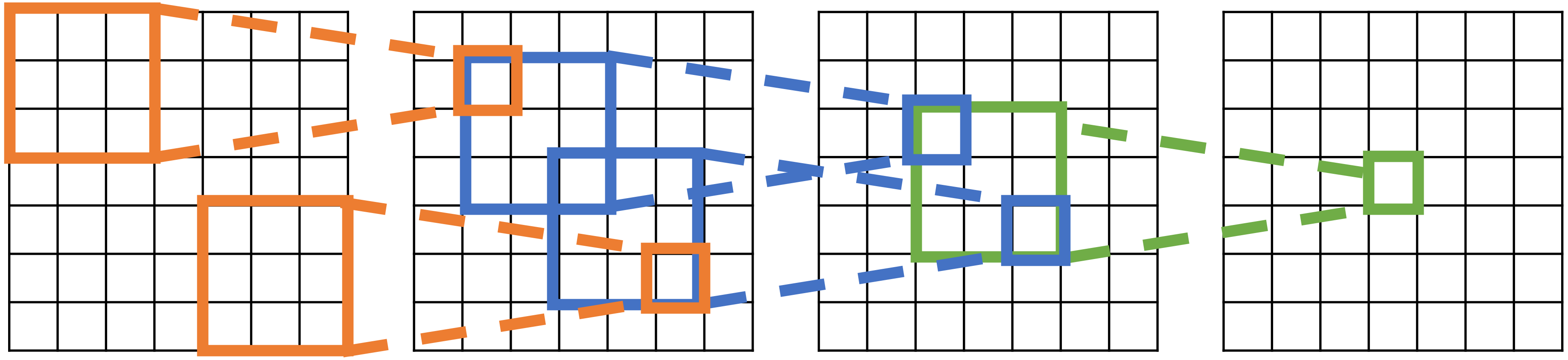


Problem: For large images we need many layers
for each output to "see" the whole image image

Slide credit: D Fouhey & J Johnson

Receptive Fields

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Input

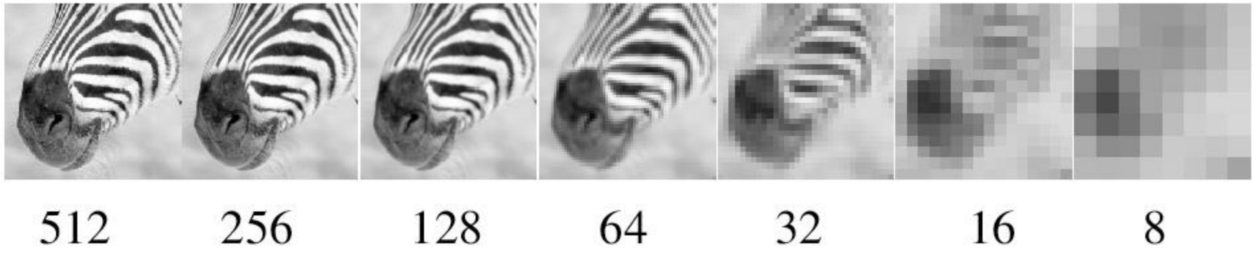
Output

Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

Slide credit: D Fouhey & J Johnson

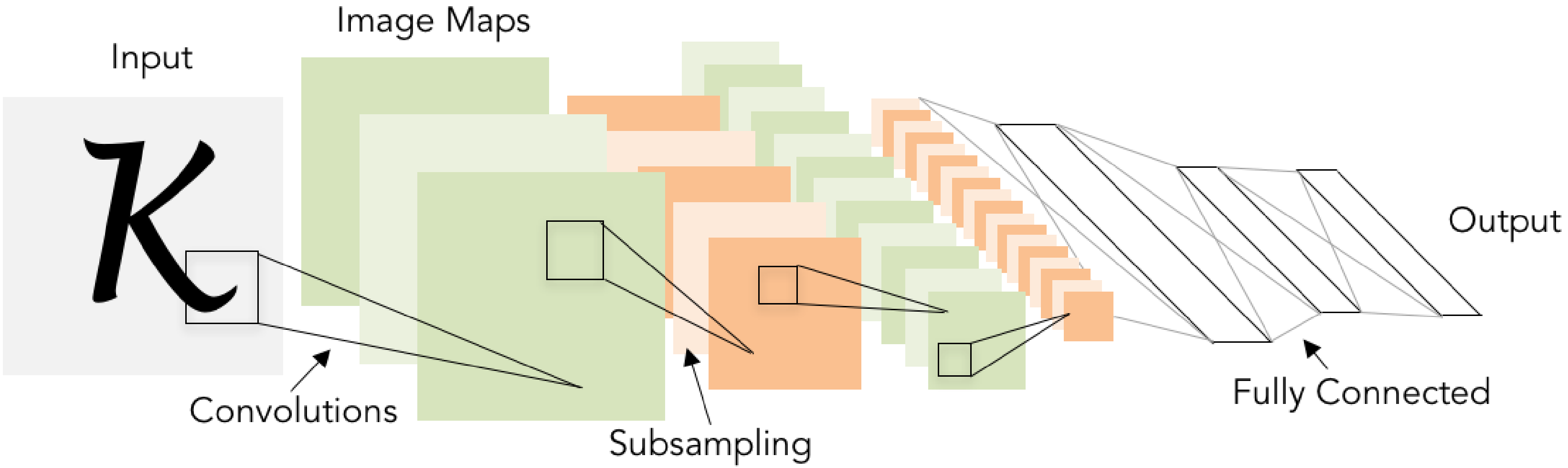
Idea: Image/Feature Pyramid



Convolutional Networks with downsampling

Classic architecture:
[Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

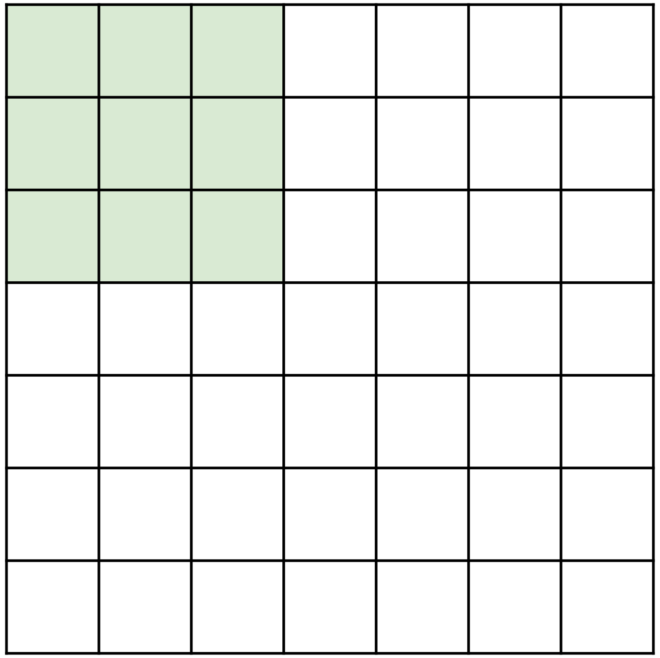
Example: LeNet-5



Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

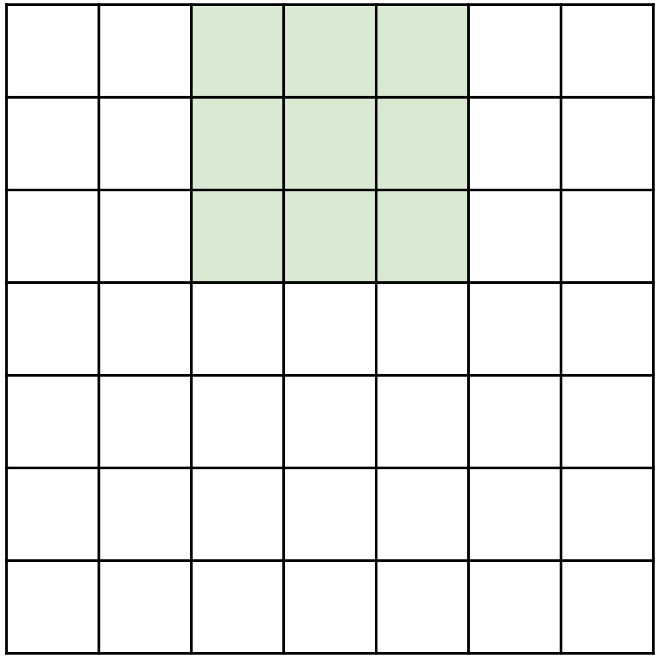
Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

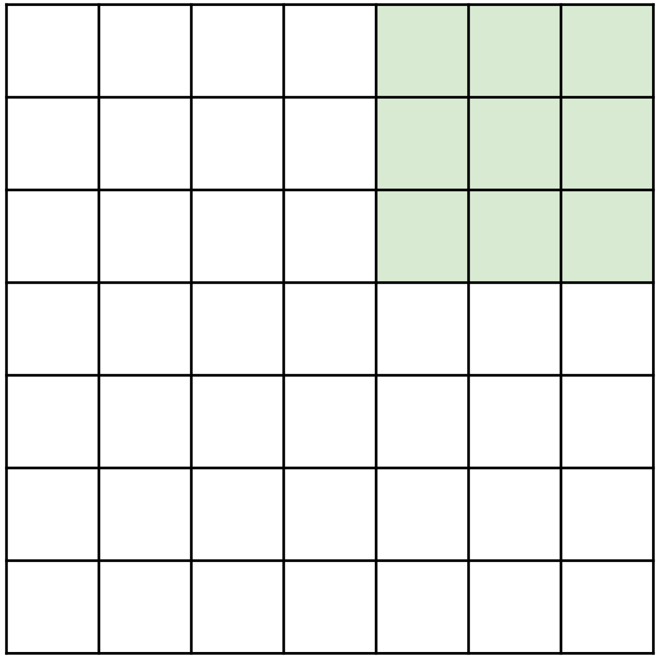
Slide credit: D Fouhey & J Johnson

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



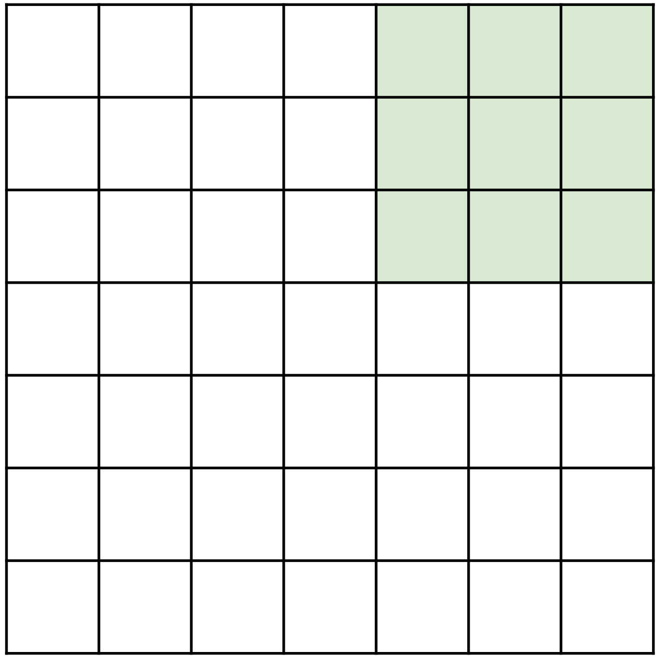
Input: 7x7

Filter: 3x3

Stride: 2

Output: 3x3

Strided Convolution



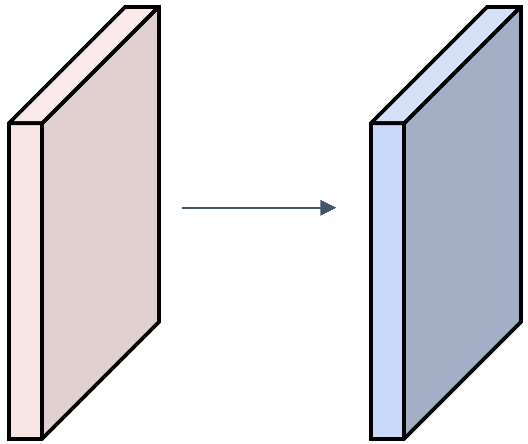
Input: 7x7
Filter: 3x3
Stride: 2

Output: 3x3

In general:
Input: W
Filter: K
Padding: P
Stride: S
Output: Ceil((W – K + 1 + 2P) / S)

Slide credit: D Fouhey & J Johnson

Convolution Example

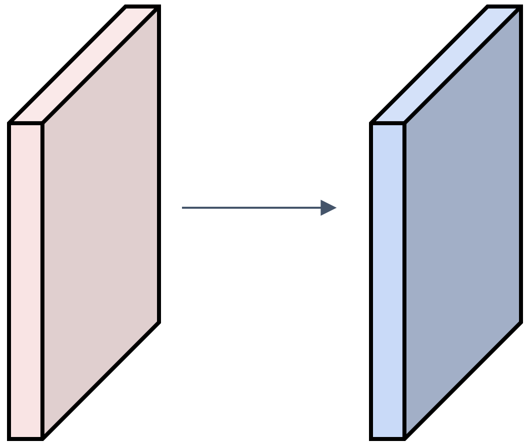


Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Slide credit: D Fouhey & J Johnson

Convolution Example



Input volume: 3 x **32** x **32**

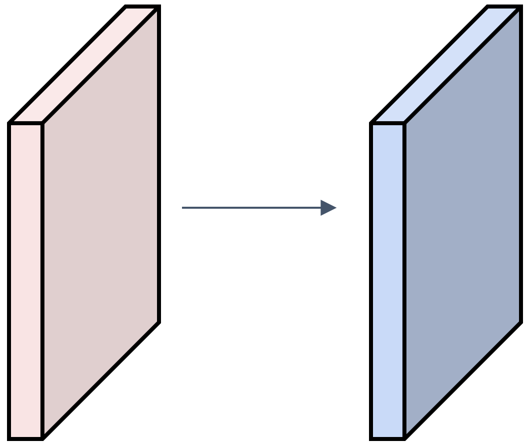
10 5x5 filters with stride **1**, pad **2**

Output volume size:

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

10 x 32 x 32

Convolution Example

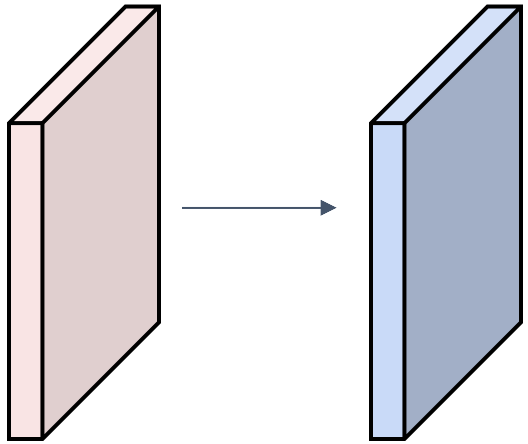


Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: ?

Slide credit: D Fouhey & J Johnson

Convolution Example

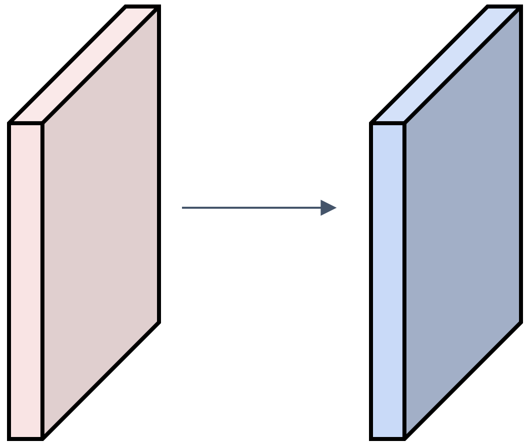


Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: **760**
Parameters per filter: 3*5*5 + 1 (for bias) = 76
10 filters, so total is 10 * 76 = **760**

Slide credit: D Fouhey & J Johnson

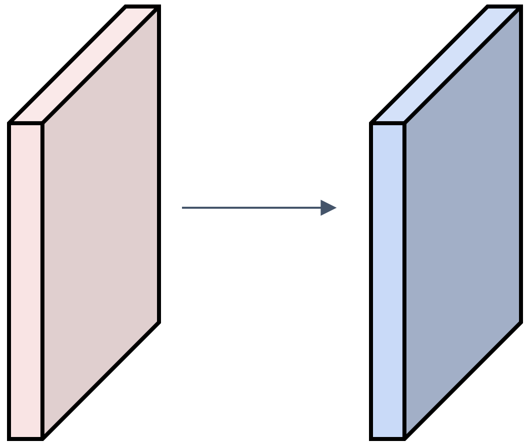
Convolution Example



Input volume: 3 x 32 x 32
10 5x5 filters with stride 1, pad 2

Output volume size: 10 x 32 x 32
Number of learnable parameters: 760
Number of multiply-add operations: ?

Convolution Example



Input volume: **3** x 32 x 32
10 **5x5** filters with stride 1, pad 2

Output volume size: **10 x 32 x 32**
Number of learnable parameters: 760
Number of multiply-add operations: **768,000**
10*32*32 = 10,240 outputs; each output is the inner product of two **3**x**5x5** tensors (75 elems); total = 75*10240 = **768K**

Slide credit: D Fouhey & J Johnson

Convolution Summary

Input: C_{in} x H x W
Hyperparameters:
- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H’ x W’ where:
- H’ = Ceil((H – K + 2P + 1) / S)
- W’ = Ceil((W – K + 2P + 1) / S)

Slide credit: D Fouhey & J Johnson

Convolution Summary

Input: C_{in} x H x W

Hyperparameters:

- **Kernel size**: K_H x K_W
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S

Weight matrix: C_{out} x C_{in} x K_H x K_W
giving C_{out} filters of size C_{in} x K_H x K_W

Bias vector: C_{out}

Output size: C_{out} x H' x W' where:

- H' = Ceil((H – K + 2P + 1) / S)
- W' = Ceil((W – K + 2P + 1) / S)

Common settings:

$K_H = K_W$ (Small square filters)
P = (K – 1) / 2 ("Same" padding)
C_{in}, C_{out} = 32, 64, 128, 256 (powers of 2)
K = 3, P = 1, S = 1 (3x3 conv)
K = 5, P = 2, S = 1 (5x5 conv)
K = 1, P = 0, S = 1 (1x1 conv)
K = 3, P = 1, S = 2 (Downsample by 2)

Forward pass of a convolution layer

```
x_padded = pad_input(x)
H_out, W_out = compute_output_dimension()
N, C, H, W = x.shape
F, C, HH, WW = w.shape
# shape of (F, C’), where C’=C * HH * WW
w_reshape = reshape_to_F_Cp(w).T
for i in range(W_out):
  for j in range(H_out):
    startx, starty = get_top_left_position()
    # shape of (N, C, HH, WW)
    x_data = get_patch(x_padded, startx, starty, HH, WW)
    # shape of (N, C’), where C’=C* HH * WW
    x_data_reshape = reshape_to_N_Cp(x_data)
    # shape of each location is (N, F)
    out[:, :, j, i] = dot_product(x_data_reshape, w_reshape) + b
```

Backward pass of a convolution layer

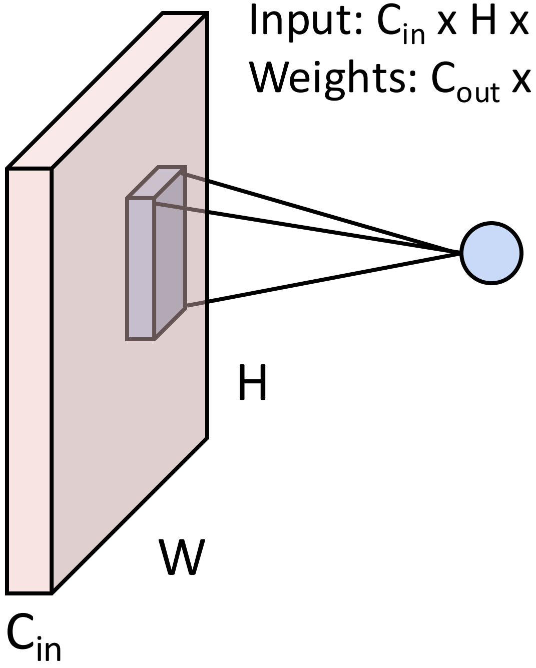
```
x_padded = pad_input(x)
H_out, W_out = compute_output_dimension()
N, C, H, W = x.shape
F, C, HH, WW = w.shape
# shape of (F, C'), where C'=C * HH * WW
w_reshape = reshape_to_F_Cp(w).T
dx_padded = np.zeros((N, C, H + pad * 2, W + pad * 2))
dw, db = np.zeros_like(w), np.zeros_like(b)
for i in range(W_out):
  for j in range(H_out):
    startx, starty = get_top_left_position()
    x_data = get_patch(x_padded, startx, starty, HH, WW)
    dx_padded_ij = get_patch(dx_padded, startx, starty, HH, WW)
    # for dx_padded, its local gradients is simply w_reshape, we need to add it (why?) to dx_padded
    # in the same patch as x_data. We need to combine dout and local gradients to get dx_padded.
    # for dw, its local gradient is simply to add x_data. We need to multiply it by dout.
    # for db, its local gradient is simply 1. We need to multiply it by dout.

dx = crop_border(dx_padded)
```

Other types of convolution

So far: 2D Convolution

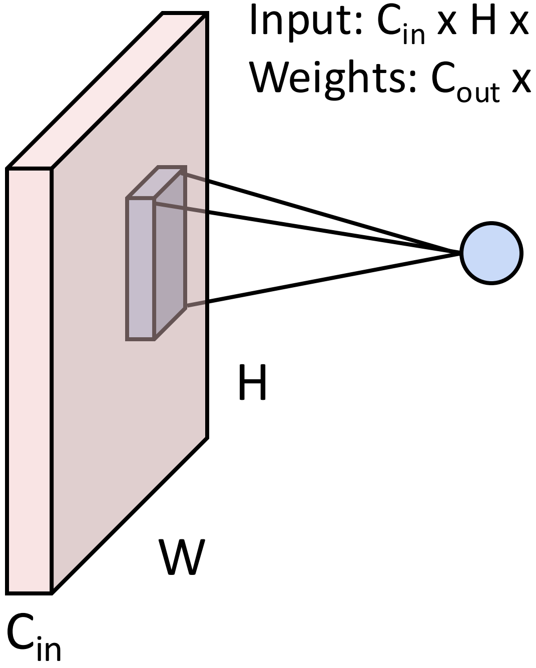
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



Other types of convolution

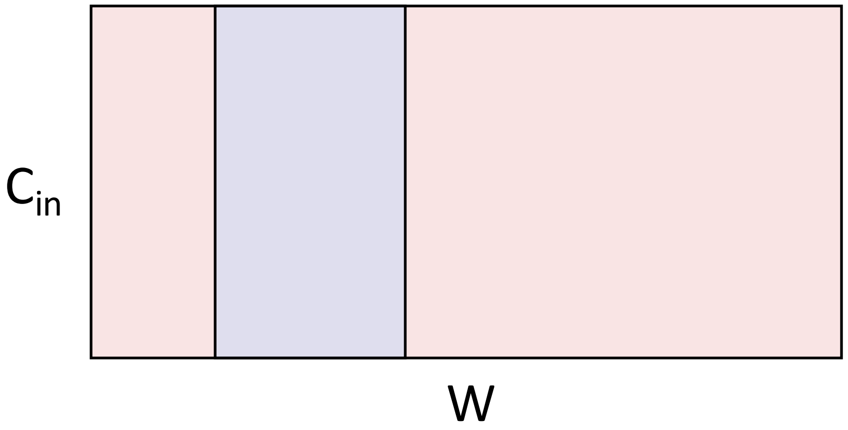
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



1D Convolution

Input: C_{in} x W
Weights: C_{out} x C_{in} x K

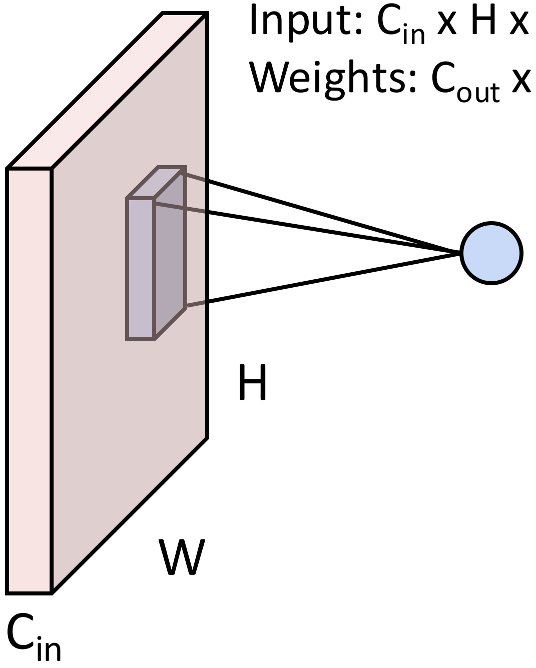


Slide credit: D Fouhey & J Johnson

Other types of convolution

So far: 2D Convolution

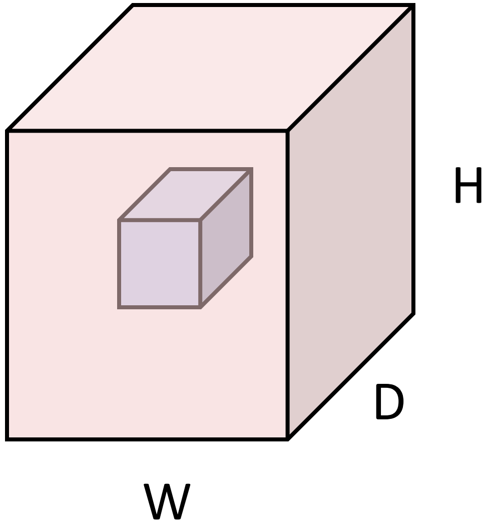
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



3D Convolution

Input: C_{in} x H x W x D
Weights: C_{out} x C_{in} x K x K x K

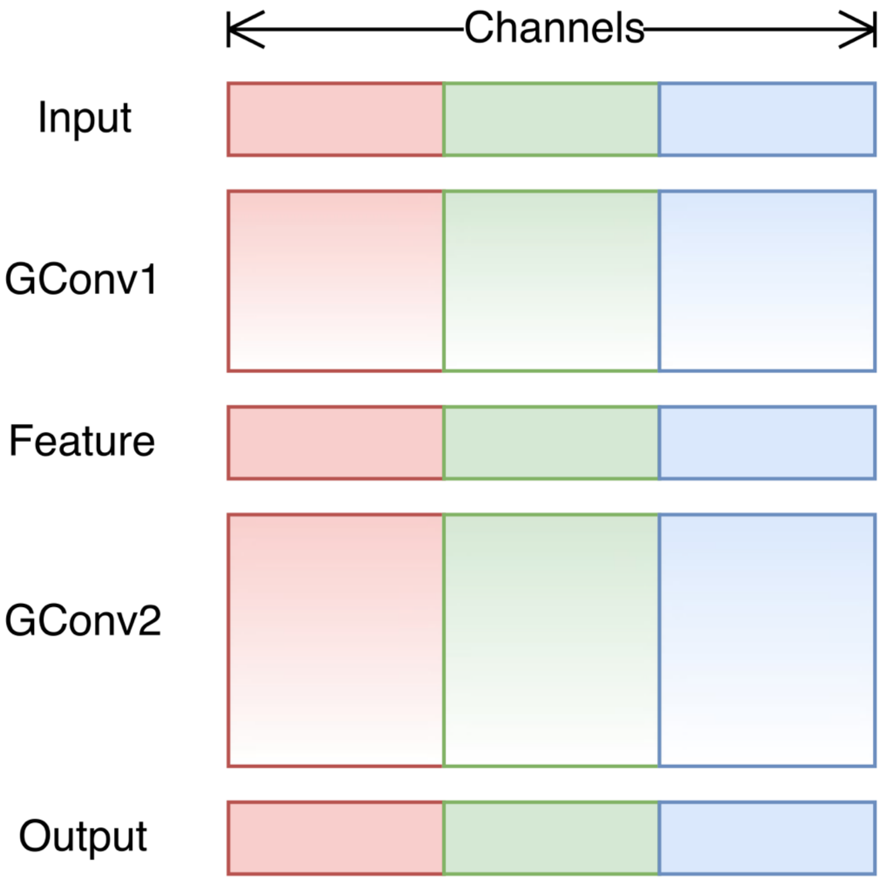
C_{in}-dim vector at each point in the volume



Slide credit: D Fouhey & J Johnson

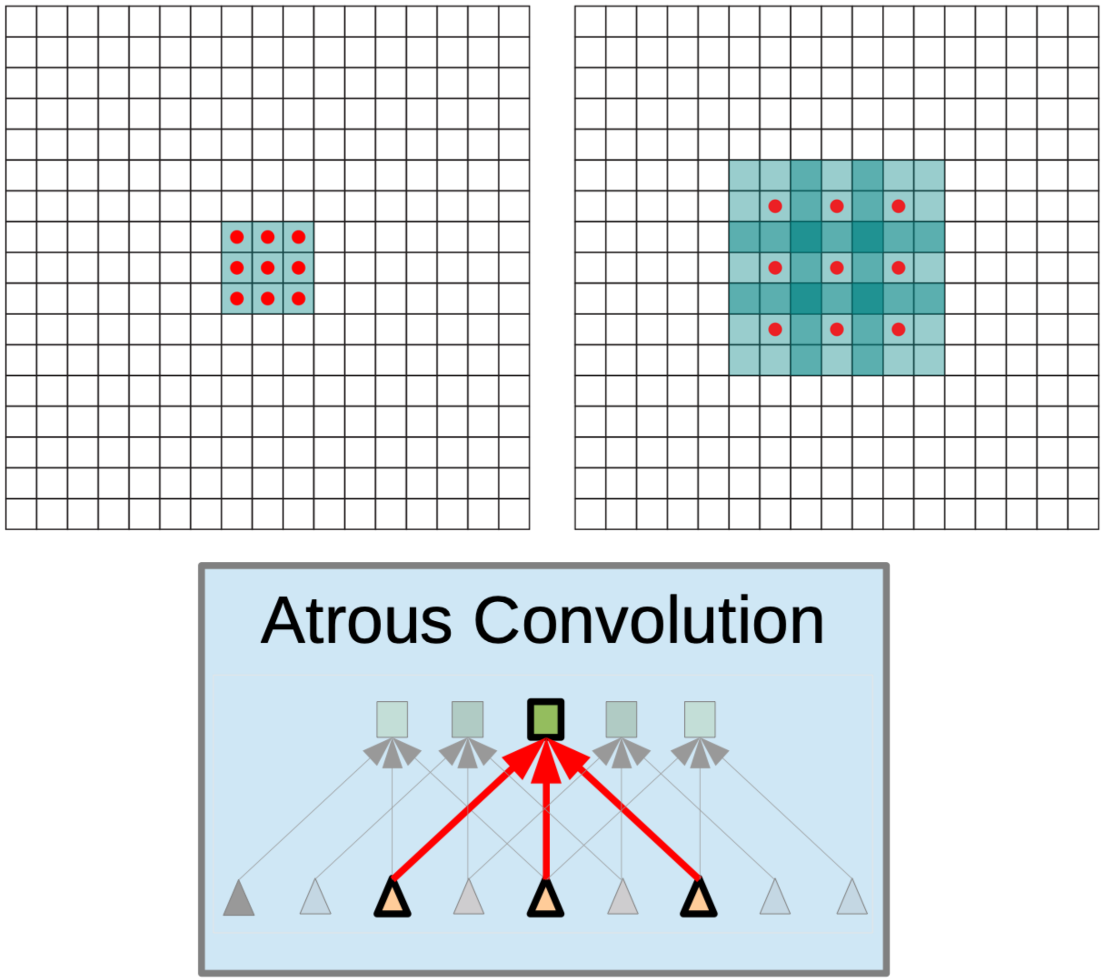
Other types of convolution

Group-based convolution



[Zhang et al., CVPR 2017]

Dilated convolution



[Yu and Koltun. ICLR 2016]
[Chen et al., ICLR 2015]

PyTorch Convolution Layer

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

Slide credit: D Fouhey & J Johnson

PyTorch Convolution Layers

Conv2d

CLASS `torch.nn.Conv2d(`*`in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'`*`)` [SOURCE]

Conv1d

CLASS `torch.nn.Conv1d(`*`in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'`*`)` [SOURCE]

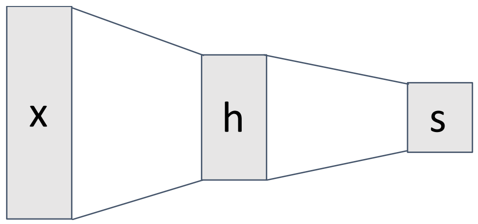
Conv3d

CLASS `torch.nn.Conv3d(`*`in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'`*`)` [SOURCE]

Slide credit: D Fouhey & J Johnson

Components of a Convolutional Network

Fully-Connected Layers

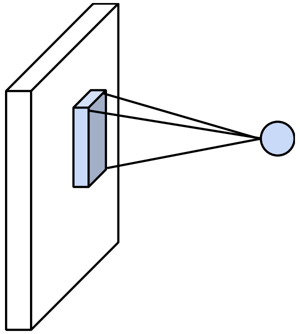


$$y = Wx + b$$

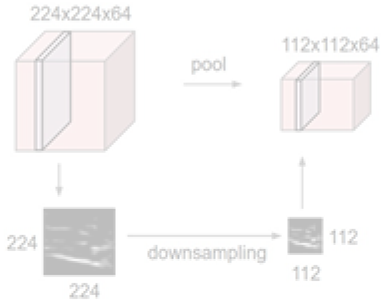
Activation Function

$$y = \max(0, x)$$

Convolution Layers



Pooling Layers



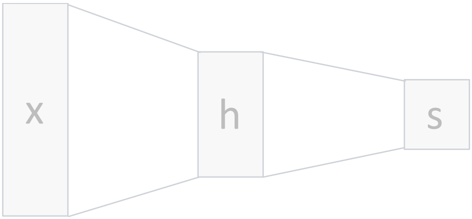
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide credit: D Fouhey & J Johnson

Components of a Convolutional Network

Fully-Connected Layers

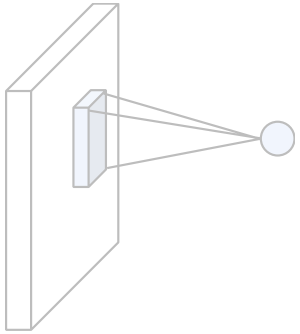


$$y = Wx + b$$

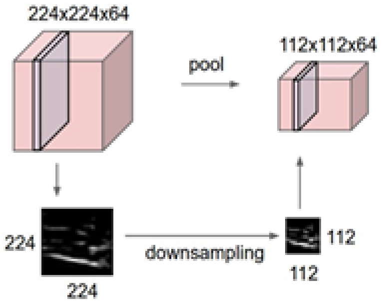
Activation Function

$$y = \max(0, x)$$

Convolution Layers



Pooling Layers

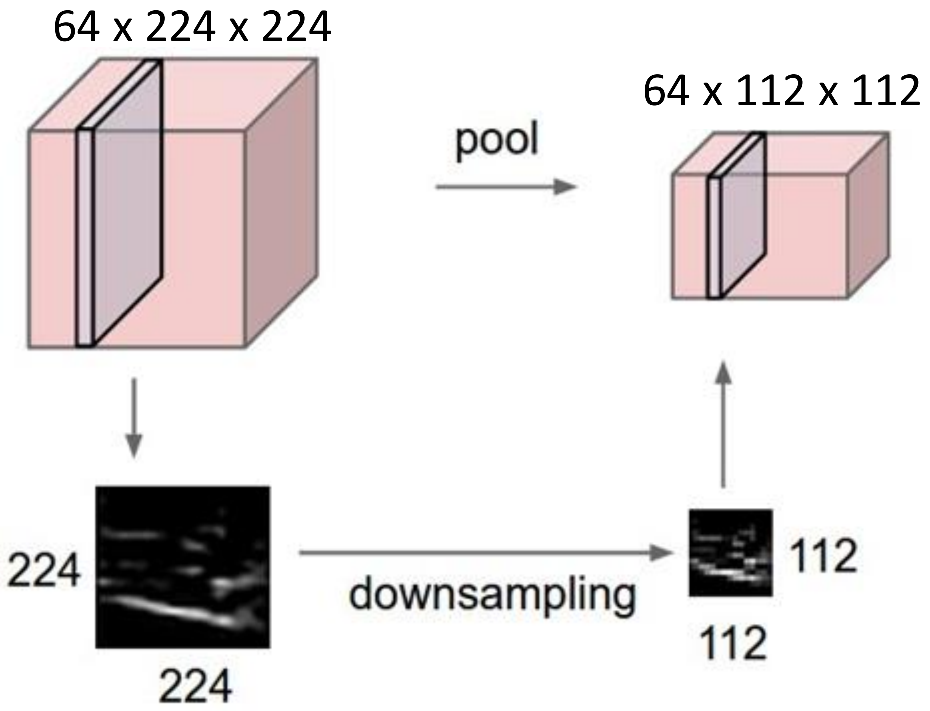


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide credit: D Fouhey & J Johnson

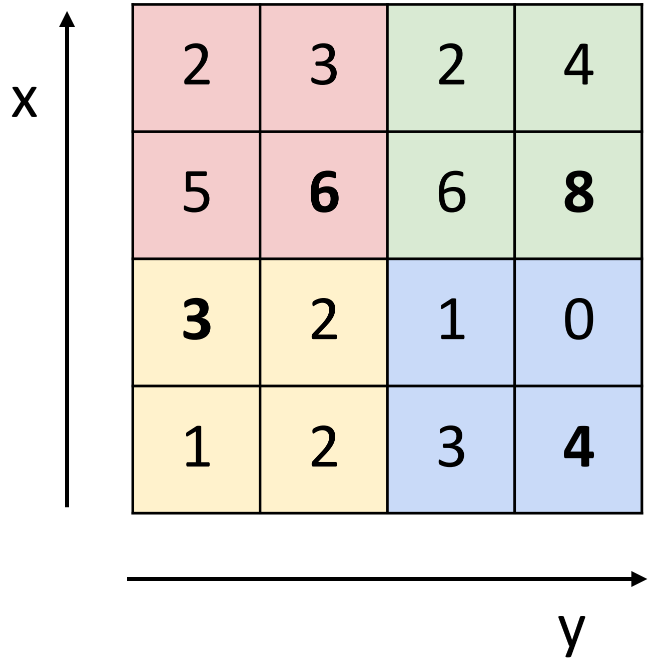
Pooling Layers: Downampling



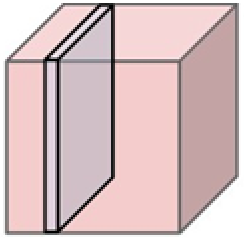
Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling

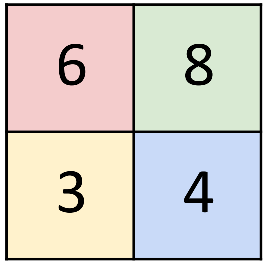
Single depth slice



64 x 224 x 224



Max pooling with 2x2
kernel size and stride 2

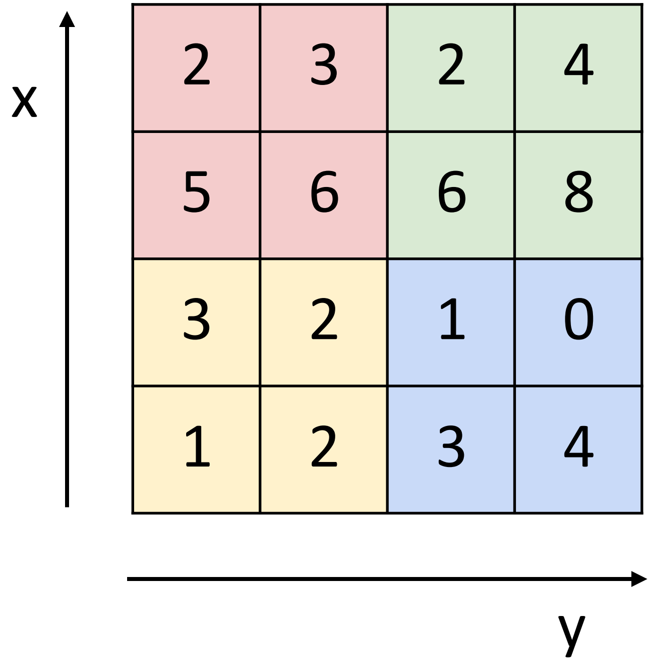


Introduces **invariance** to
small spatial shifts
No learnable parameters!

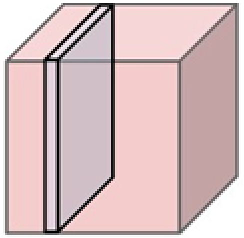
Slide credit: D Fouhey & J Johnson

Average Pooling

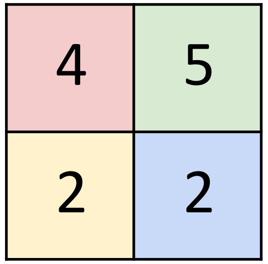
Single depth slice



64 x 224 x 224



Avg pooling with 2x2
kernel size and stride 2



Introduces **invariance** to
small spatial shifts
No learnable parameters!

Pooling Summary

Input: C x H x W

Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

Output: C x H’ x W’ where
- H’ = Ceil((H – K + 2P + 1) / S)
- W’ = Ceil((W – K + 2P + 1) / S)

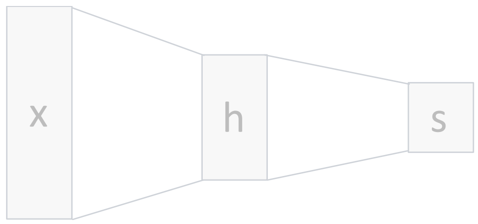
Learnable parameters: None!

Common settings:
max, K = 2, S = 2
max, K = 3, S = 2 (AlexNet)

Slide credit: D Fouhey & J Johnson

Components of a Convolutional Network

Fully-Connected Layers

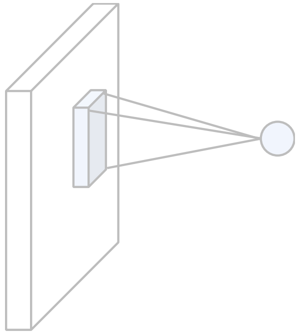


$$y = Wx + b$$

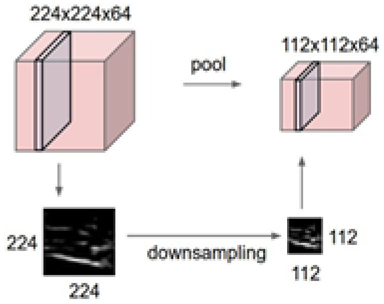
Activation Function

$$y = \max(0, x)$$

Convolution Layers



Pooling Layers



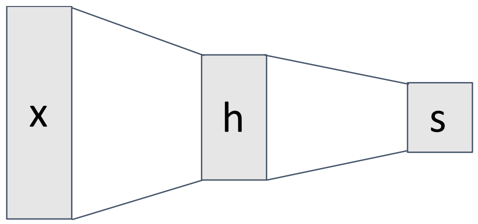
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide credit: D Fouhey & J Johnson

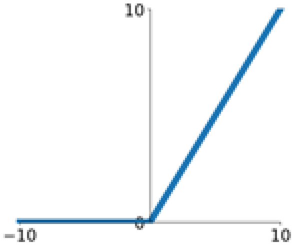
Components of a Convolutional Network

Fully-Connected Layers



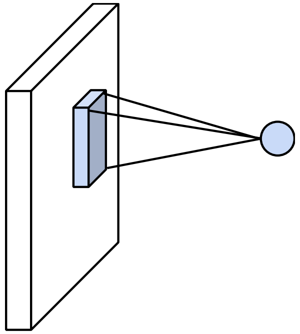
$$y = Wx + b$$

Activation Function

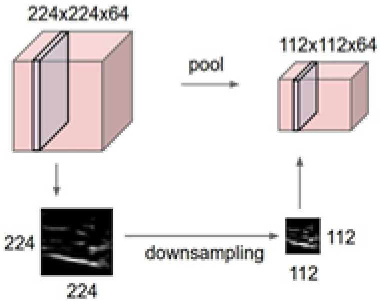


$$y = \max(0, x)$$

Convolution Layers



Pooling Layers



Normalization

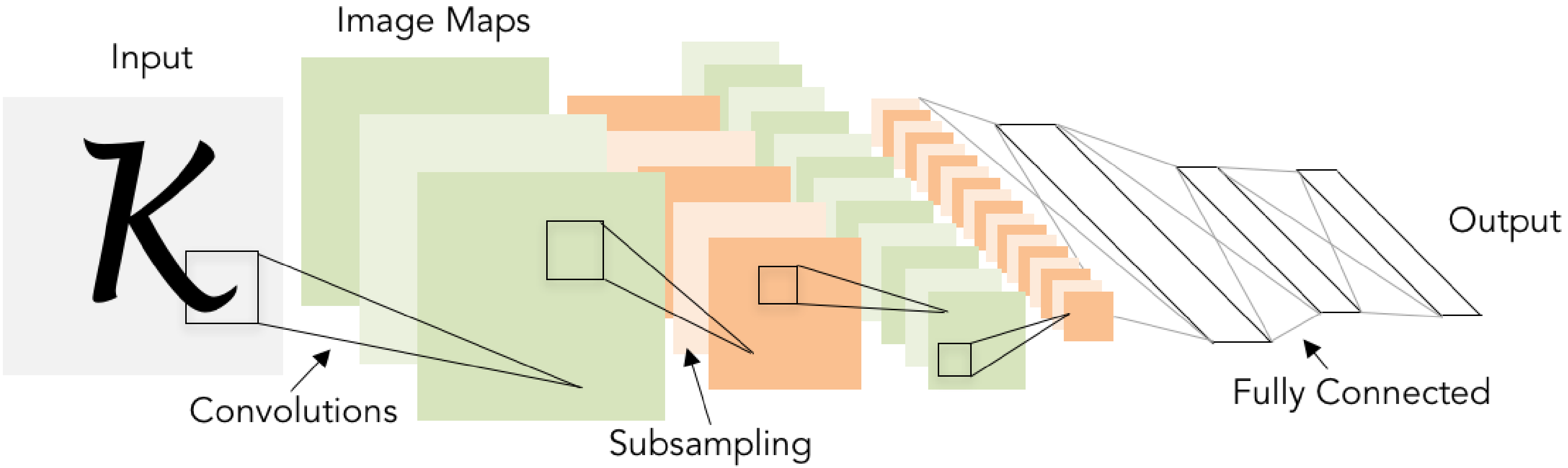
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide credit: D Fouhey & J Johnson

Convolutional Networks

Classic architecture:
[Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5

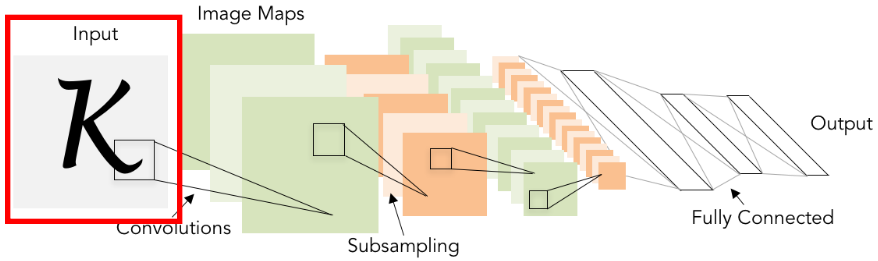


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: LeNet-5

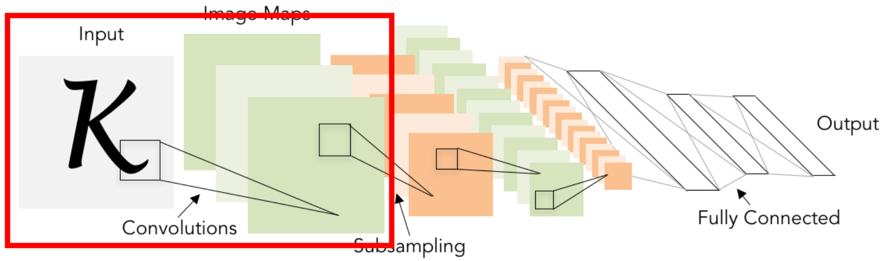
Layer	Output Size	Weight Size
Input	1 x 28 x 28	



Lecun et al, “Gradient-based learning applied to document recognition”, 1998

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	

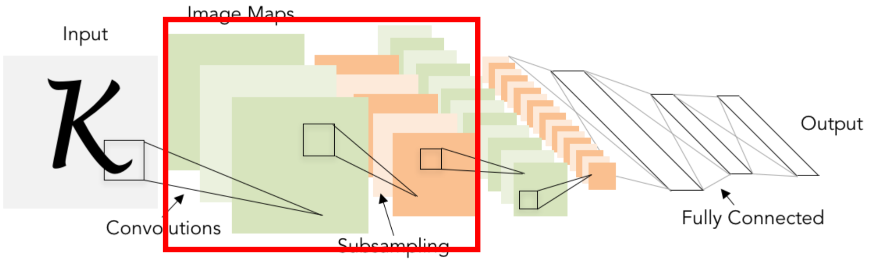


Lecun et al, “Gradient-based learning applied to document recognition”, 1998

Slide credit: D Fouhey & J Johnson

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	

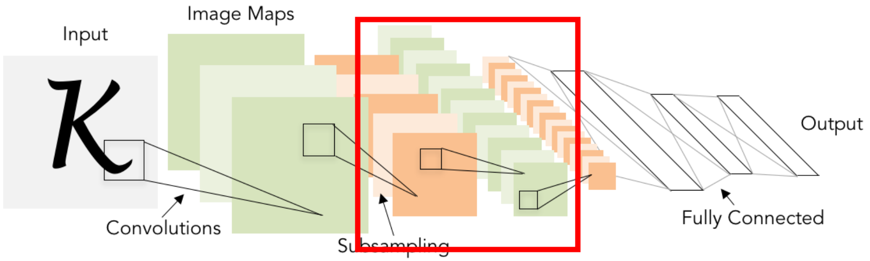


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	

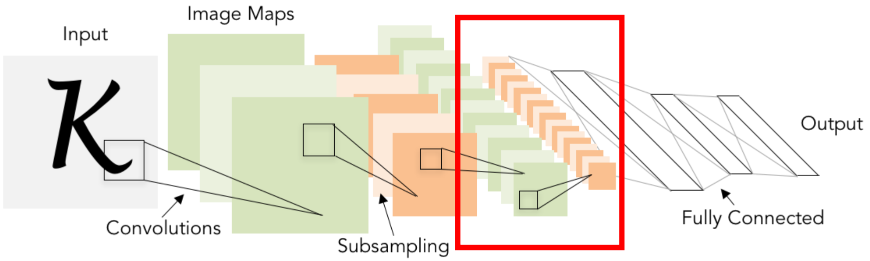


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: LeNet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	

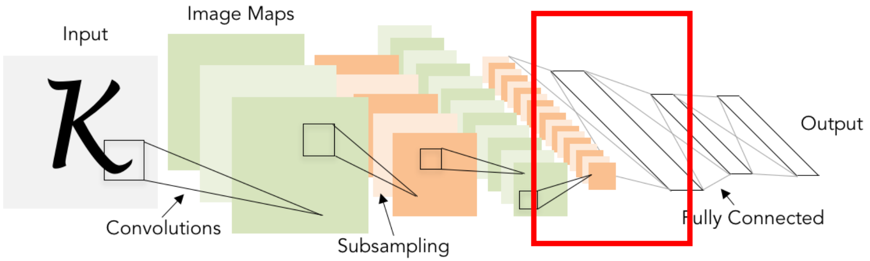


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: Lenet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	

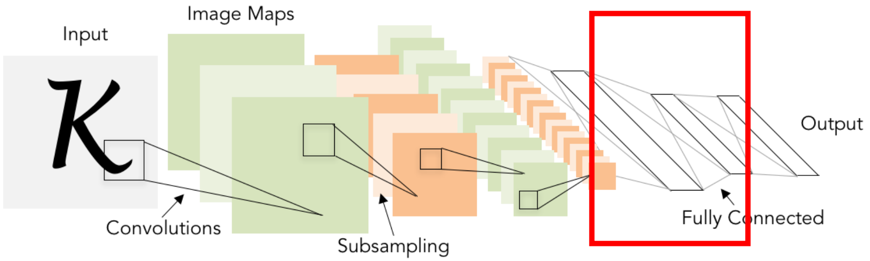


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: Lenet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	

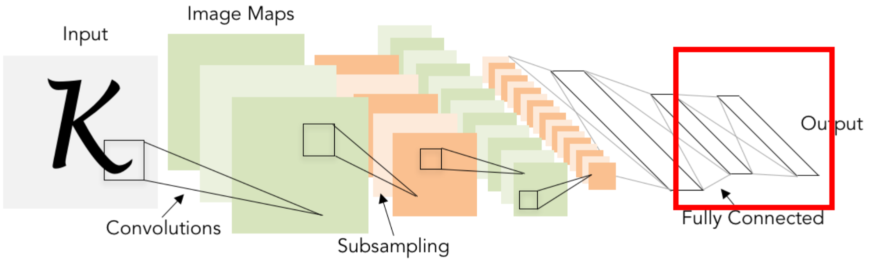


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: Lenet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10

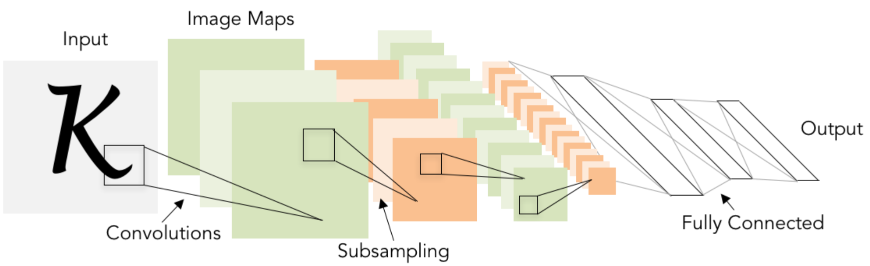


Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Example: Lenet-5

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



As we go through the network:

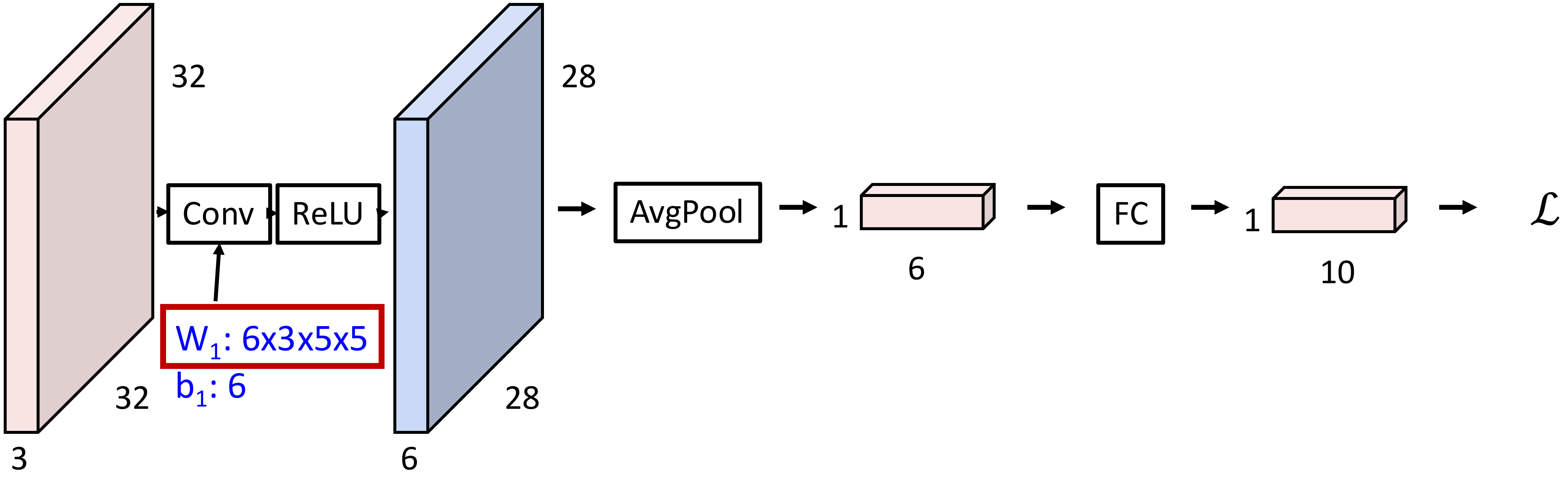
Spatial size **decreases**
(using pooling or strided conv)

Number of channels **increases**
(total "volume" is preserved!)

Lecun et al, "Gradient-based learning applied to document recognition", 1998

Slide credit: D Fouhey & J Johnson

Backpropagation for CNNs



hidden layer:
N x 6 x 28 x 28

What does its computational graph look like?

Next Class

More about

Convolutional Neural Networks