Convolutional Neural Networks

COMP 322 | Image Processing

Local filters, learned features, image models, and 1D signals

A teaching compilation of selected public course slides

Original slide footers and embedded attributions are preserved.

SECTION

Image models preserve spatial structure

A fully connected layer treats the image as one long vector. Convolution applies the same small learned filter across locations, keeping local neighborhoods meaningful.

Transition written for this compilation. Core visual sequence: Stanford CS231n Lecture 5, slides 45-64.

SOURCE AND TRANSITION

Convolution basics

We will follow one filter from a local patch to a feature map, then combine many filters into a network. The original CS231n build slides are kept in sequence so the diagrams develop step by step.

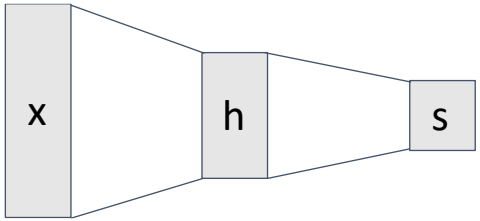
Source: Stanford CS231n, Lecture 5, slides 41-64. https://cs231n.stanford.edu/slides/2026/lecture_5.pdf

Convolutional Neural Networks

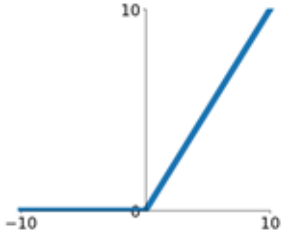
Today: Convolutional Networks

Fully-Connected Layer

We have already seen these



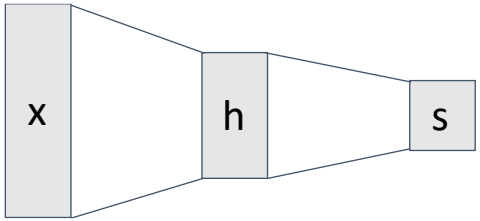
Activation Function



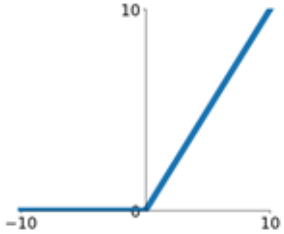
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

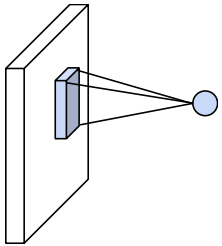


Activation Function

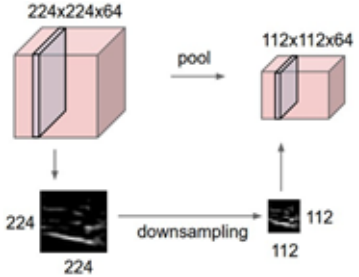


Today: Image-specific operators

Convolution Layer



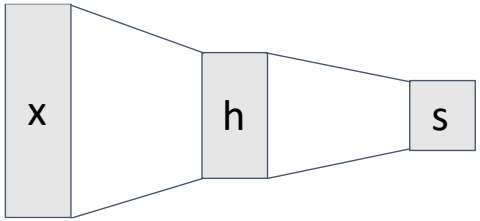
Pooling Layer



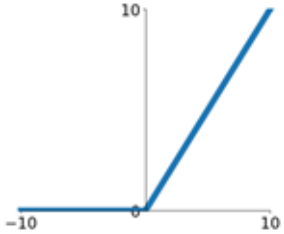
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

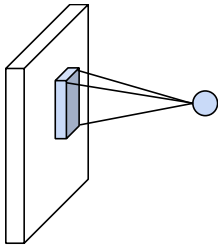


Activation Function

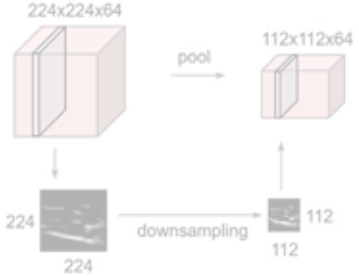


Today: Image-specific operators

Convolution Layer

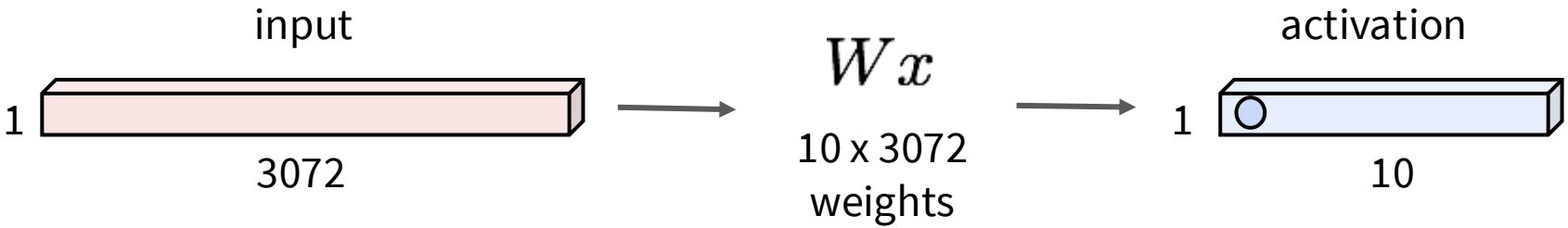


Pooling Layer



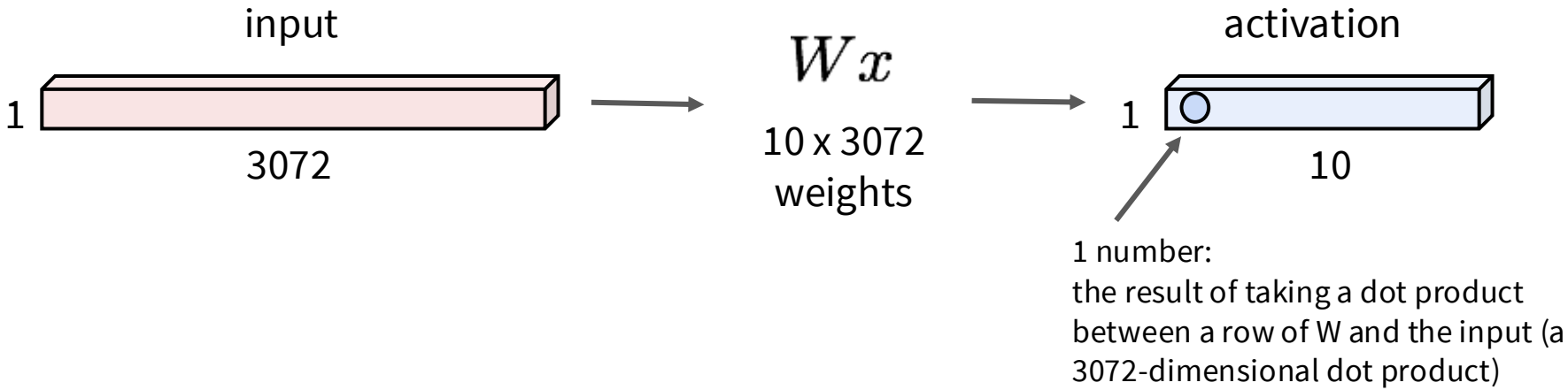
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



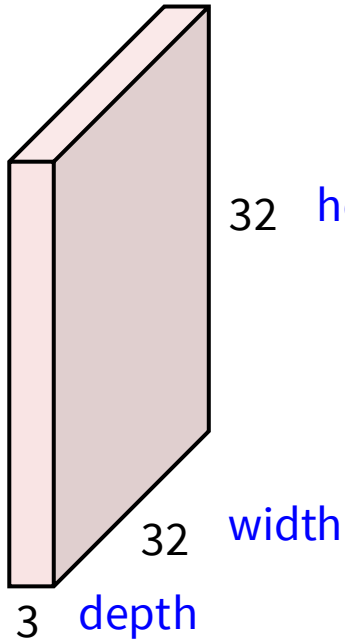
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



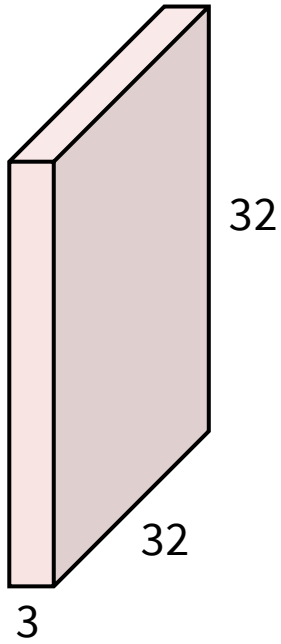
Convolution Layer

32x32x3 image -> preserve spatial structure

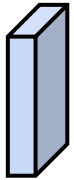


Convolution Layer

32x32x3 image



5x5x3 filter

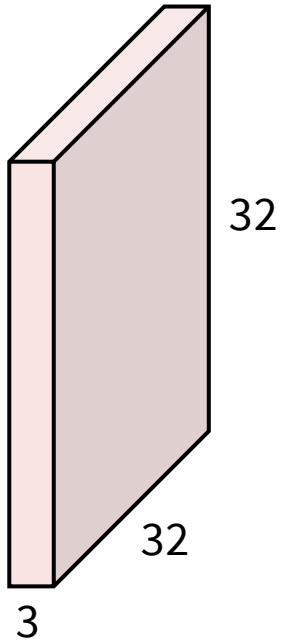


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

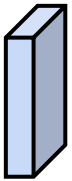
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

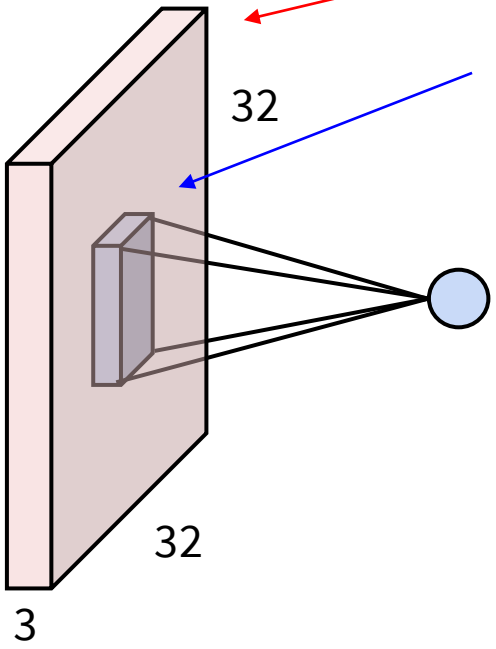


5x5x3 filter



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

Convolution Layer



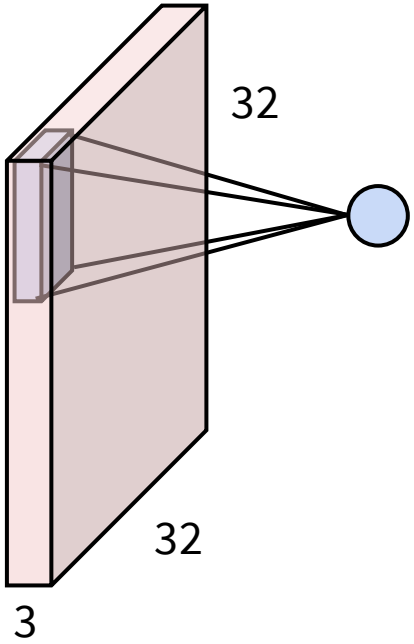
32x32x3 image

5x5x3 filter w

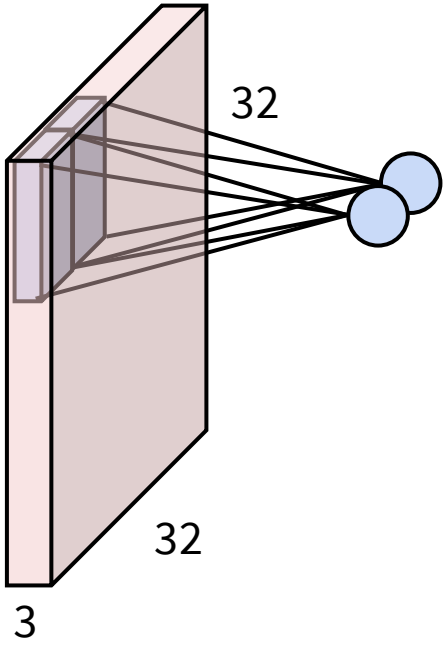
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

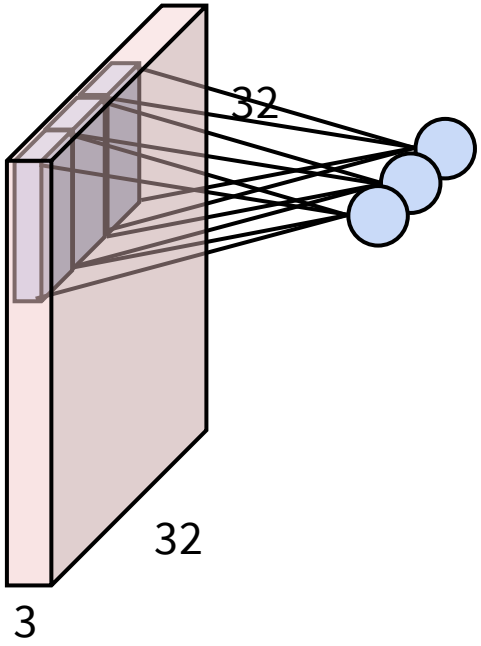
Convolution Layer



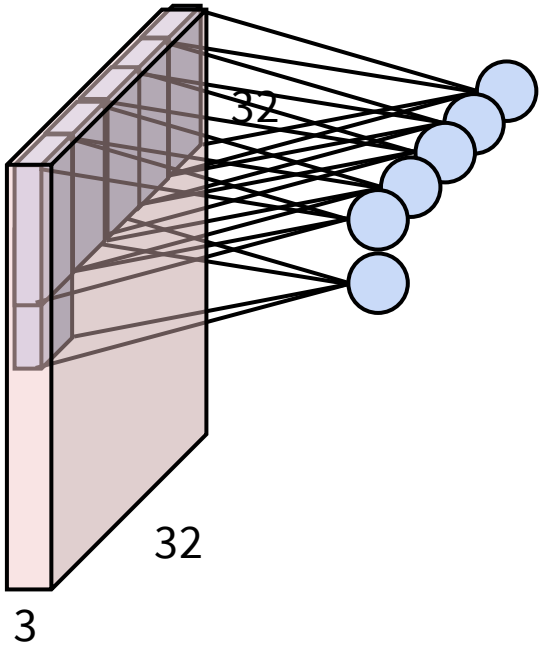
Convolution Layer



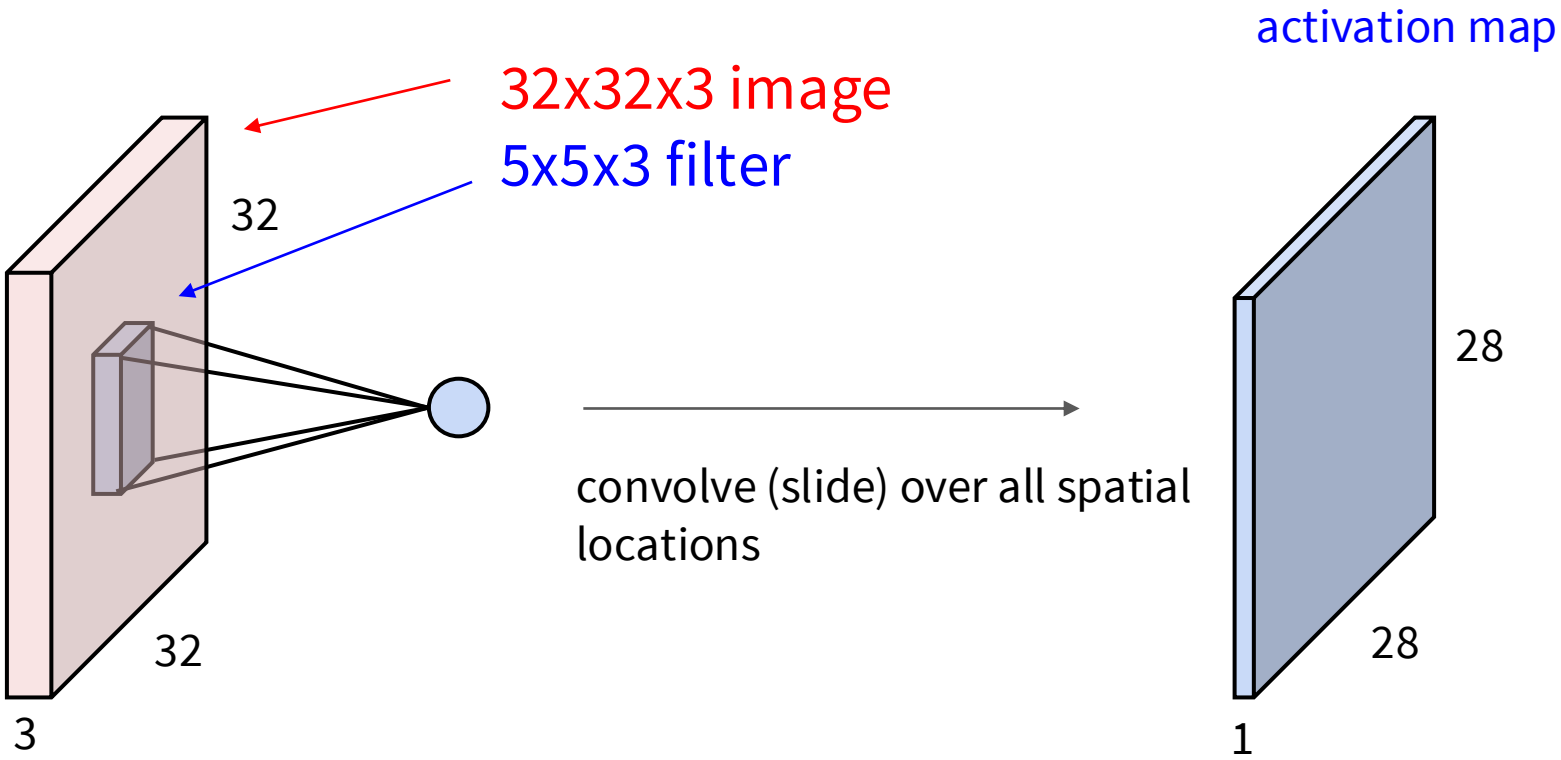
Convolution Layer



Convolution Layer

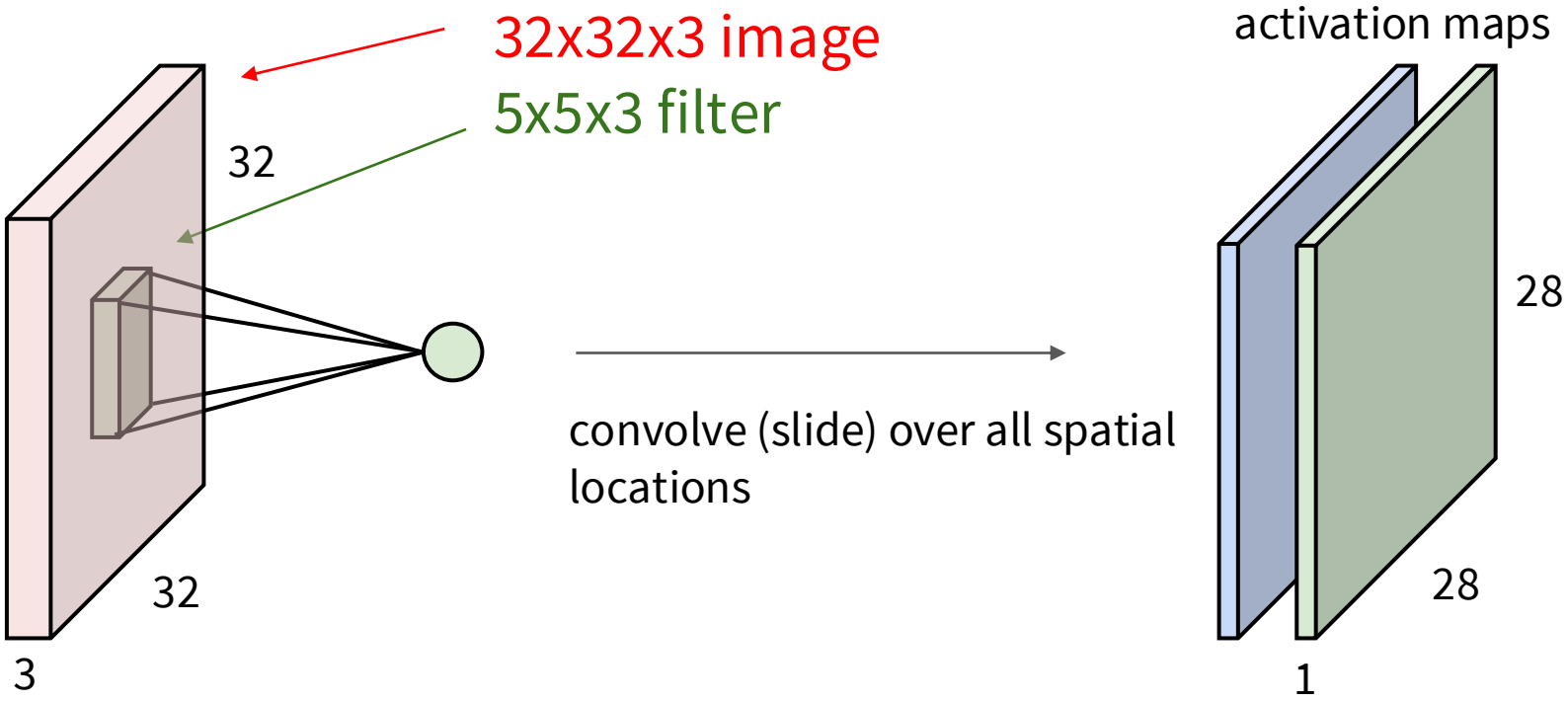


Convolution Layer

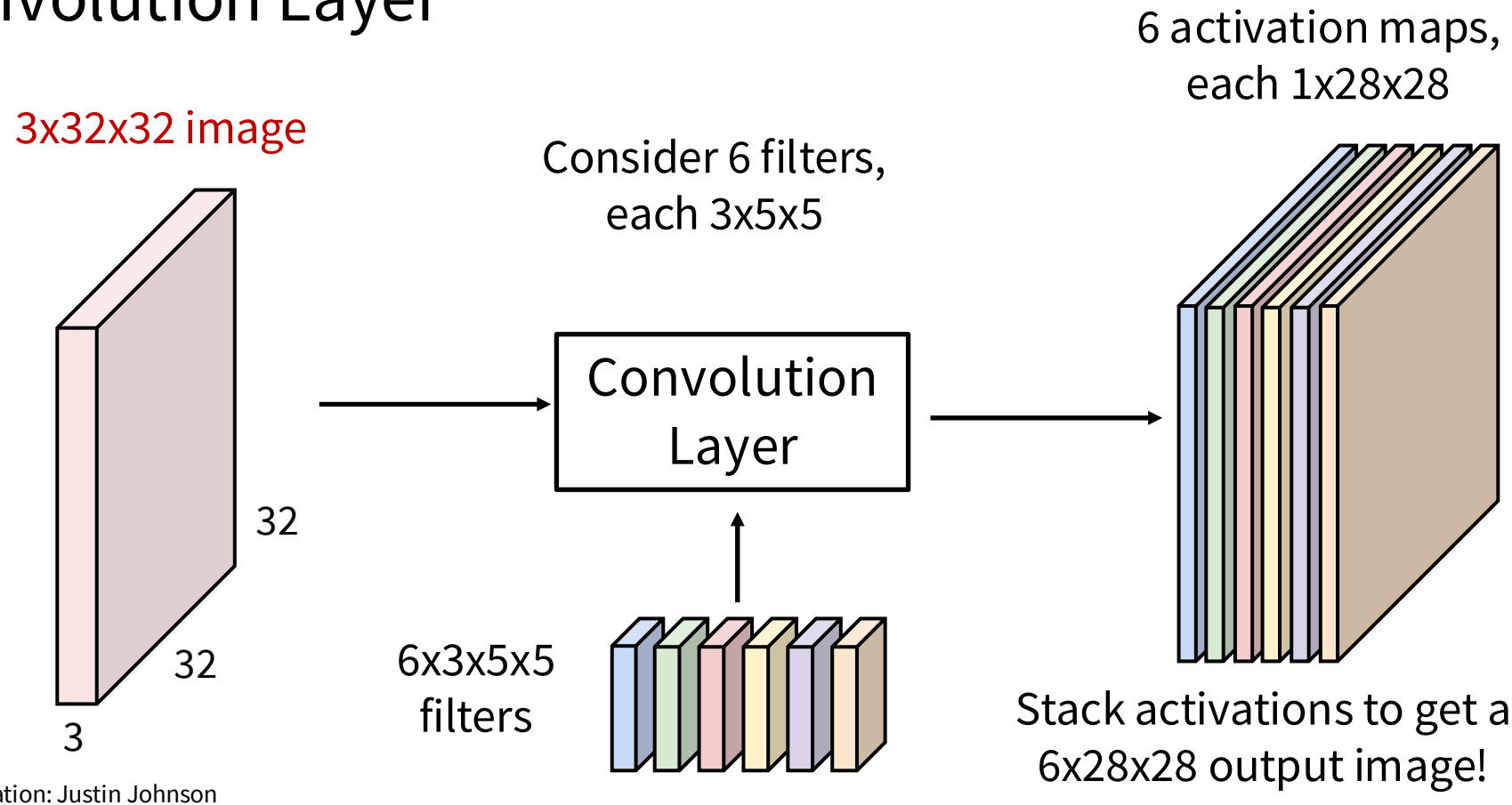


Convolution Layer

consider a second, green filter

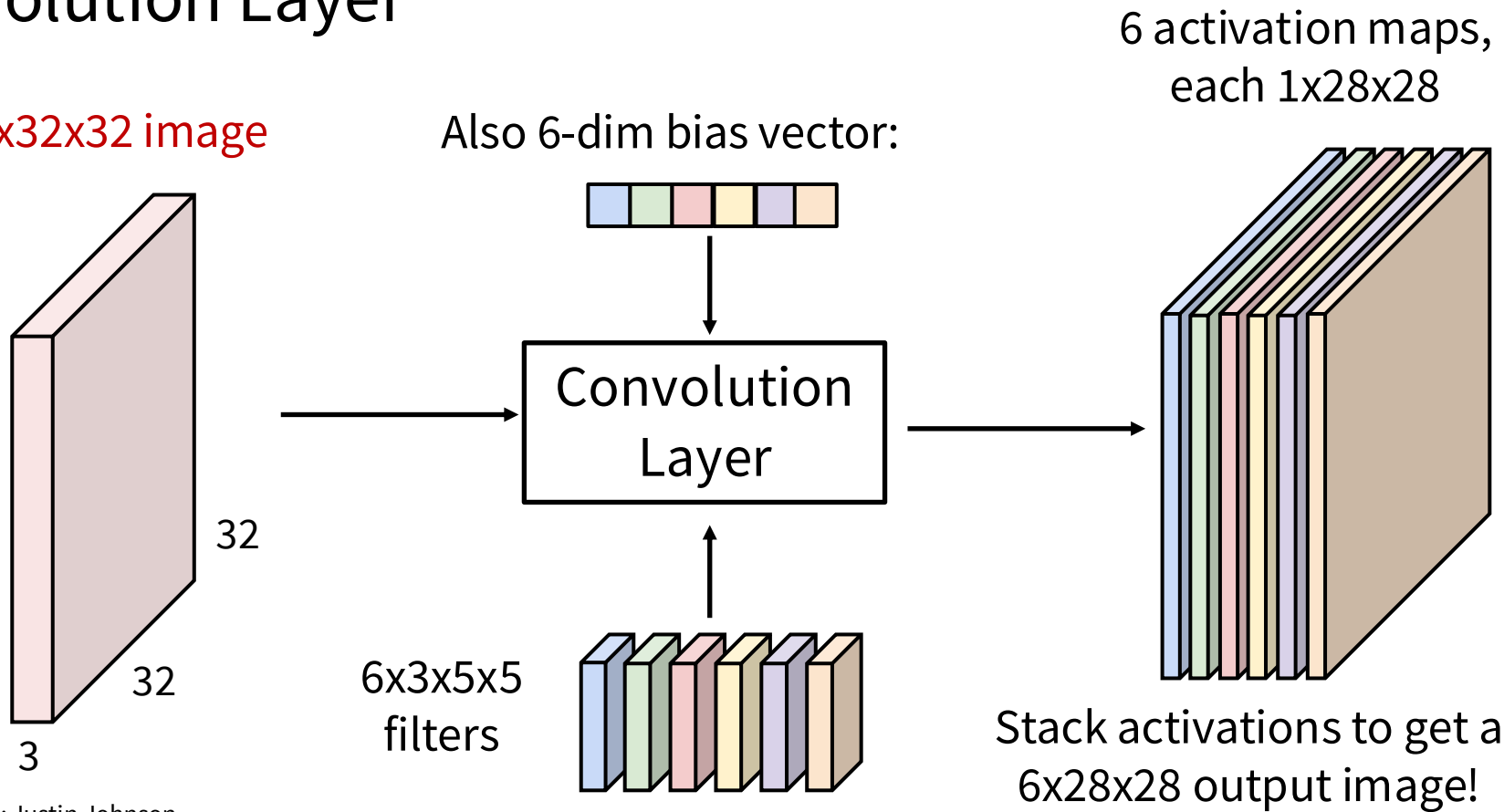


Convolution Layer



Slide inspiration: Justin Johnson

Convolution Layer



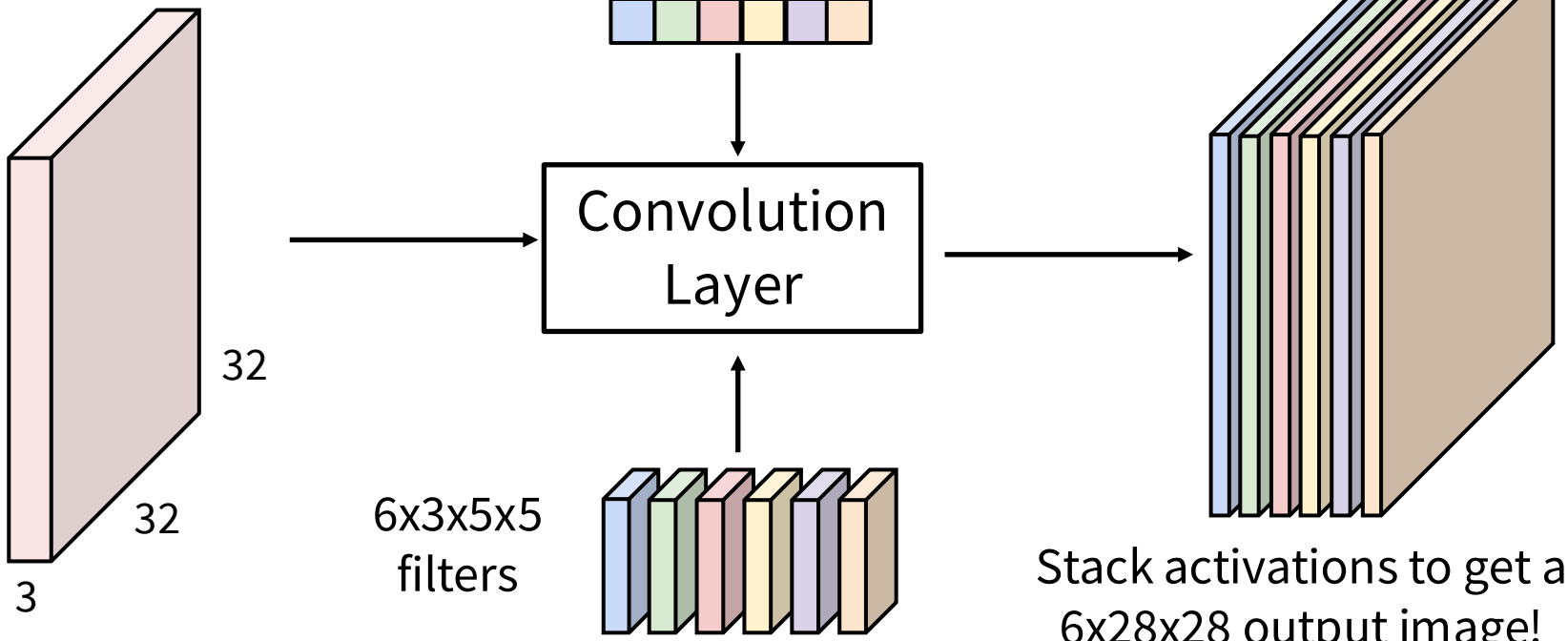
Slide inspiration: Justin Johnson

Convolution Layer

28x28 grid, at each point a 6-dim vector

3x32x32 image

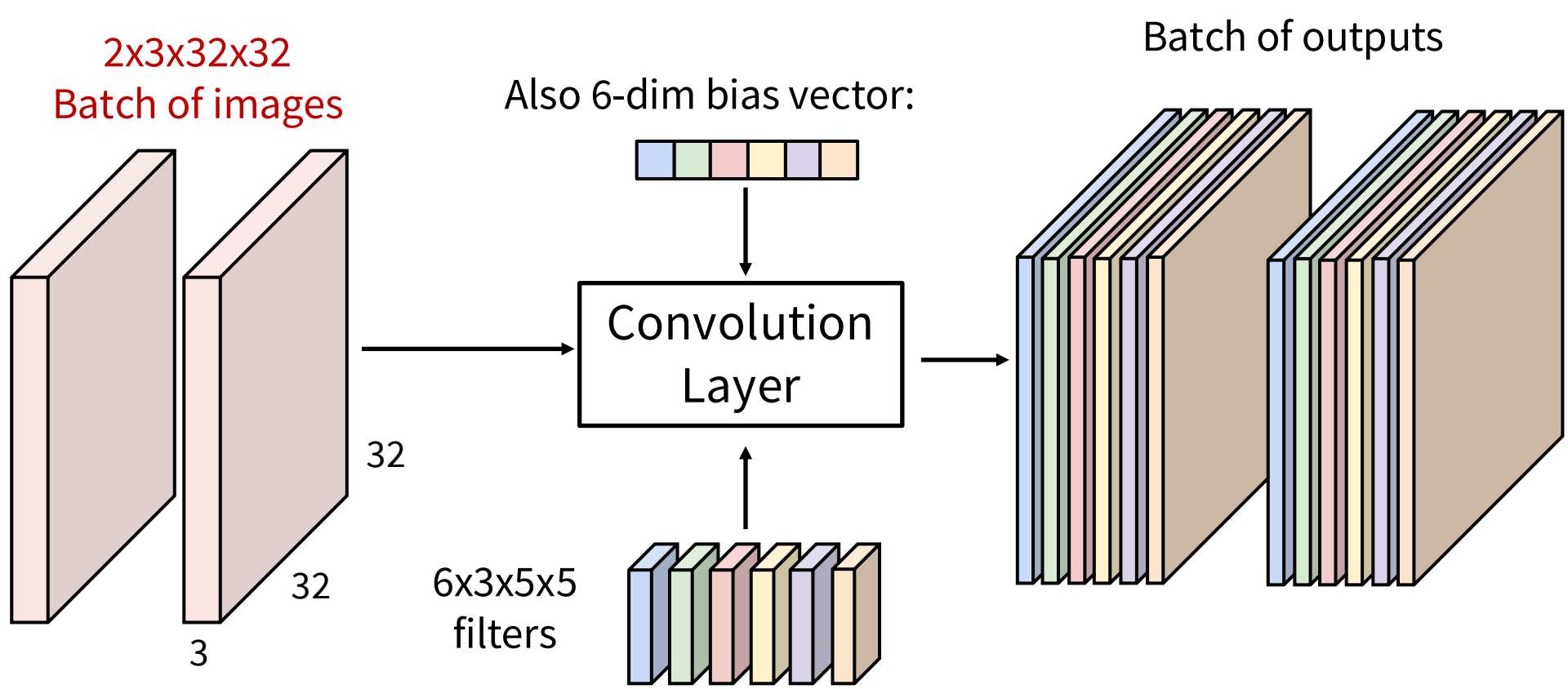
Also 6-dim bias vector:



Stack activations to get a 6x28x28 output image!

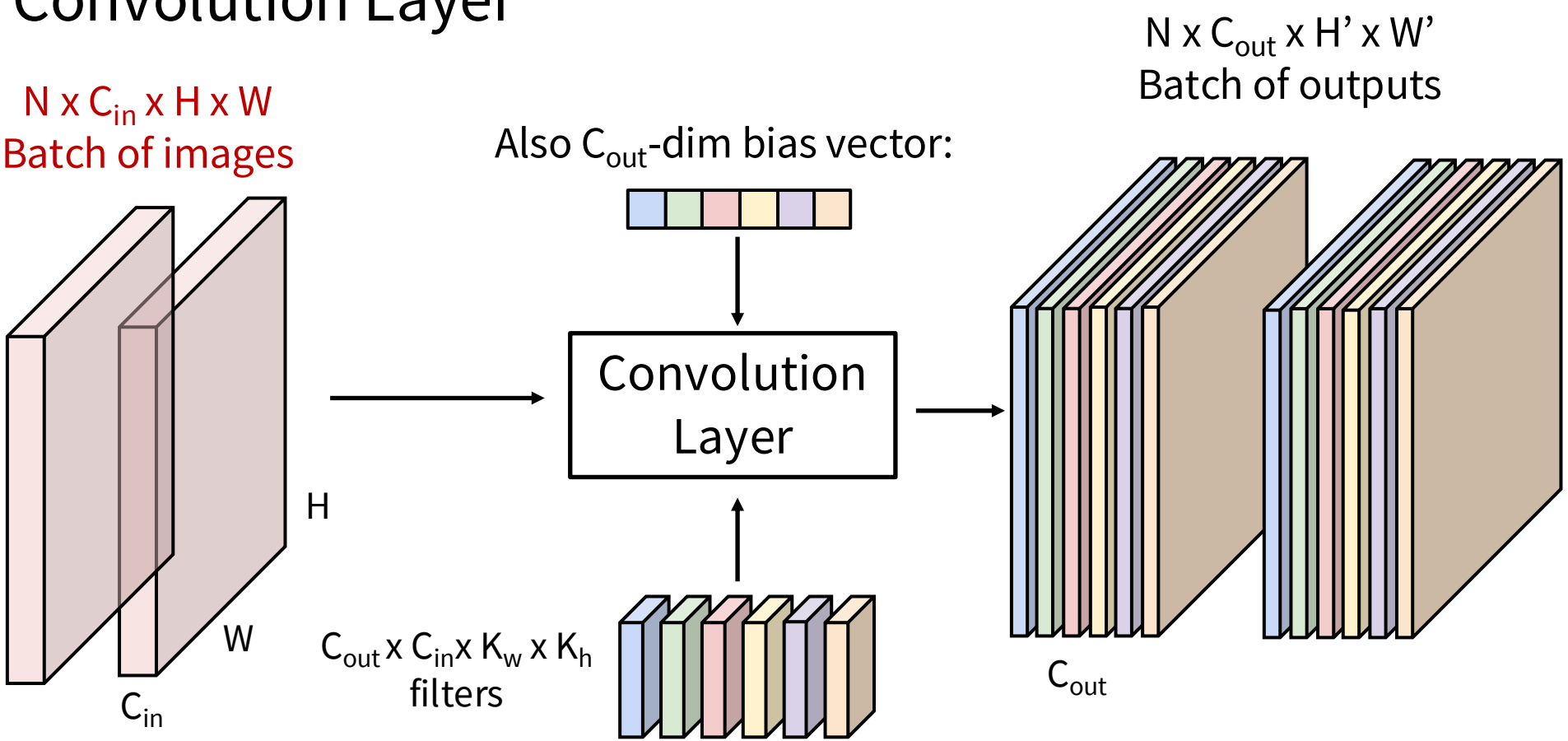
Slide inspiration: Justin Johnson

Convolution Layer



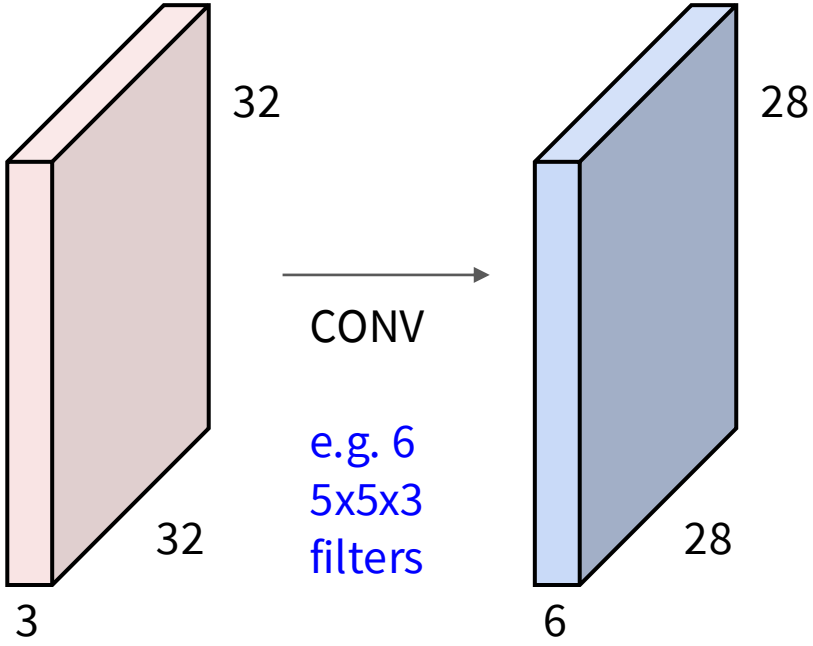
Slide inspiration: Justin Johnson

Convolution Layer

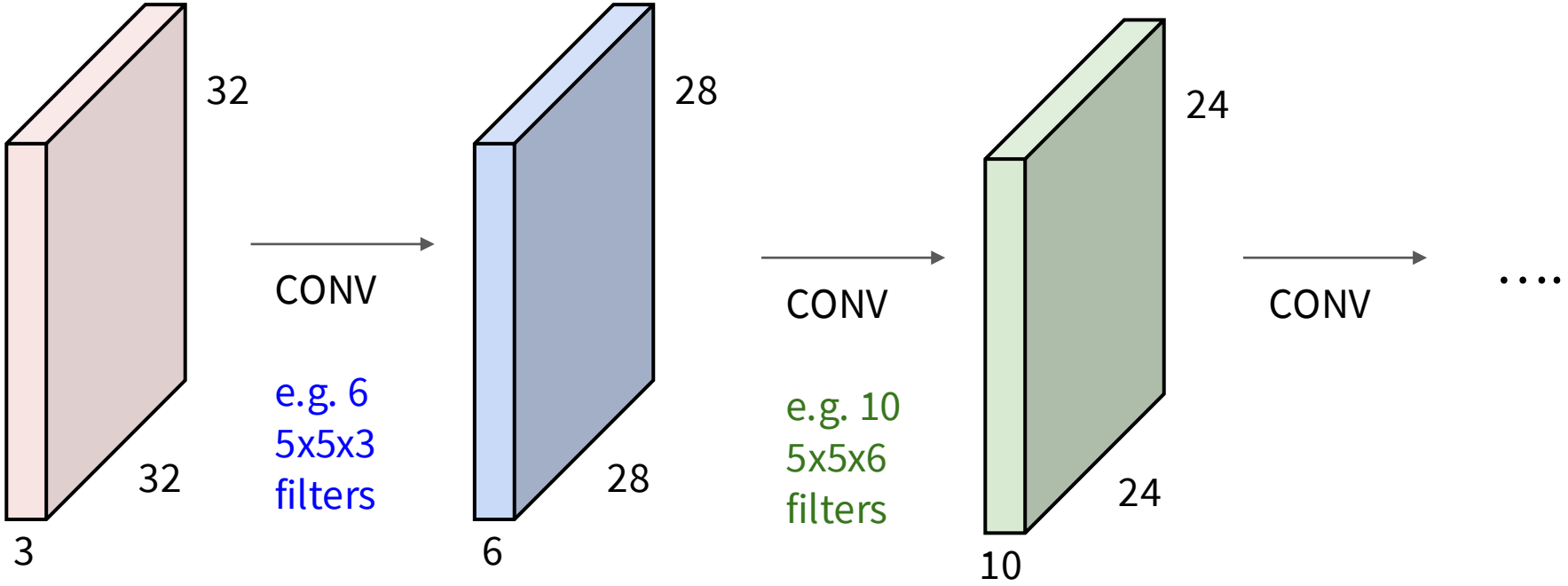


Slide inspiration: Justin Johnson

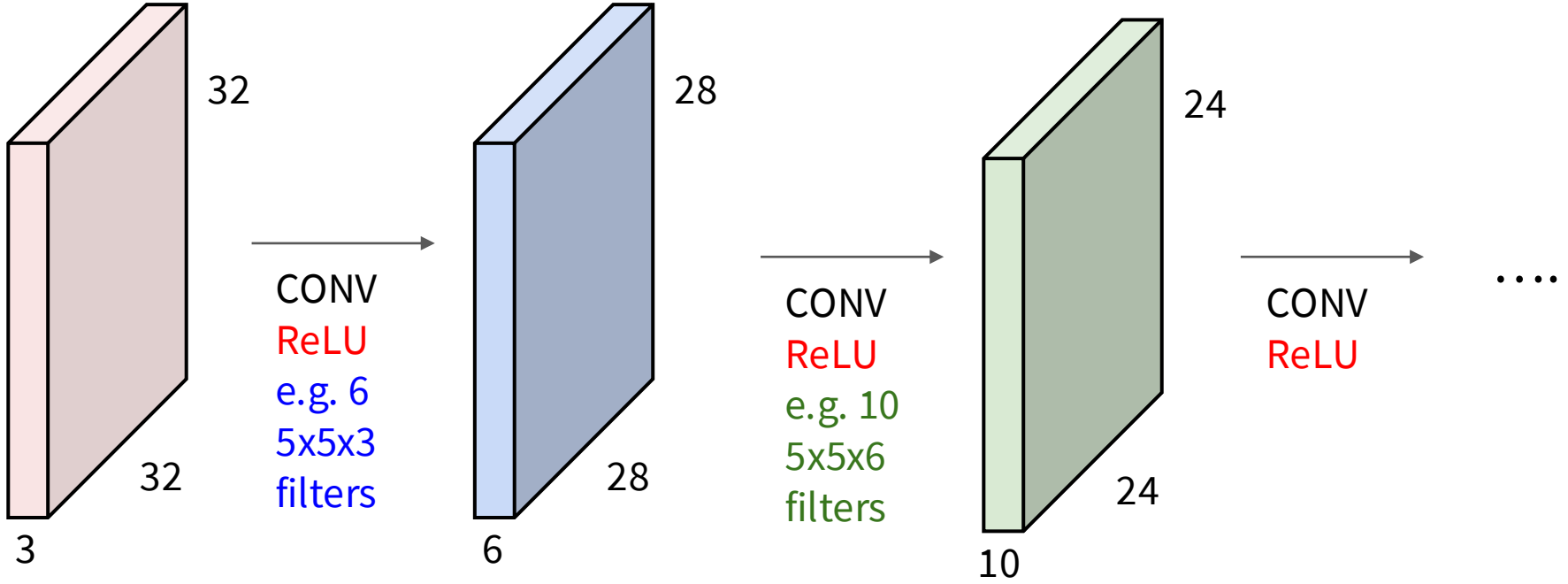
A ConvNet is a neural network with Conv layers



A ConvNet is a neural network with Conv layers



A ConvNet is a neural network with Conv layers with activation functions!



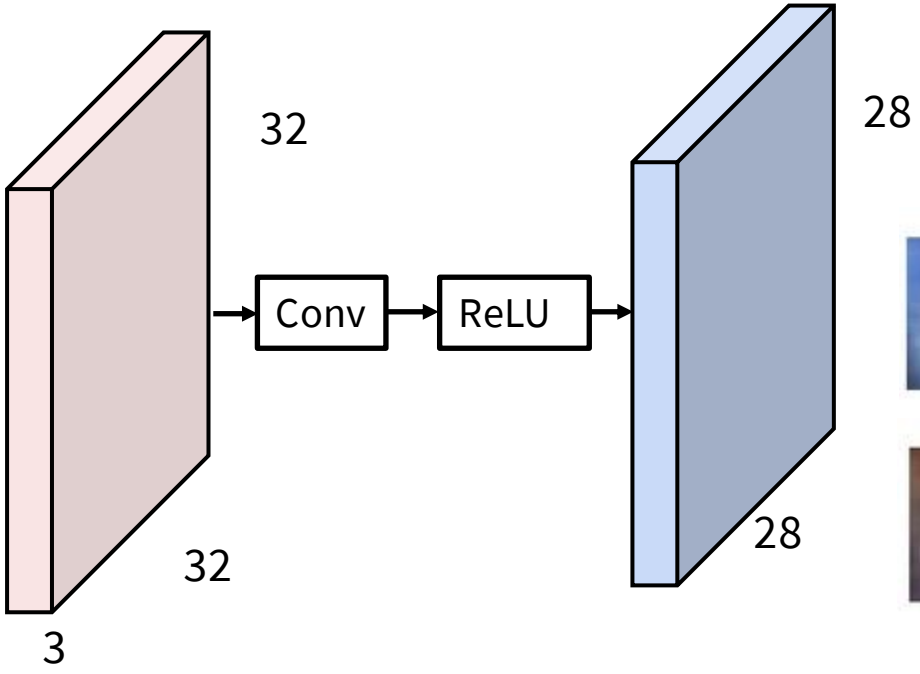
SOURCE AND TRANSITION

What do convolutional filters learn?

A filter detects a local pattern wherever it appears. Deeper layers combine these responses into larger, more complex features.

Source: Stanford CS231n, Lecture 5, slides 65-68. https://cs231n.stanford.edu/slides/2026/lecture_5.pdf

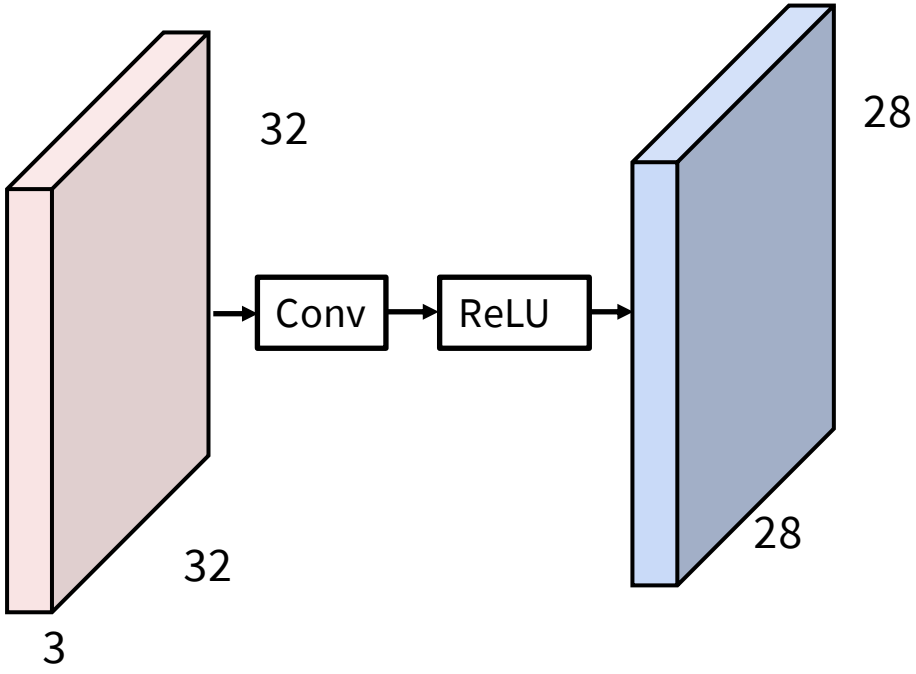
What do Conv filters learn?



Linear classifier: One template per class



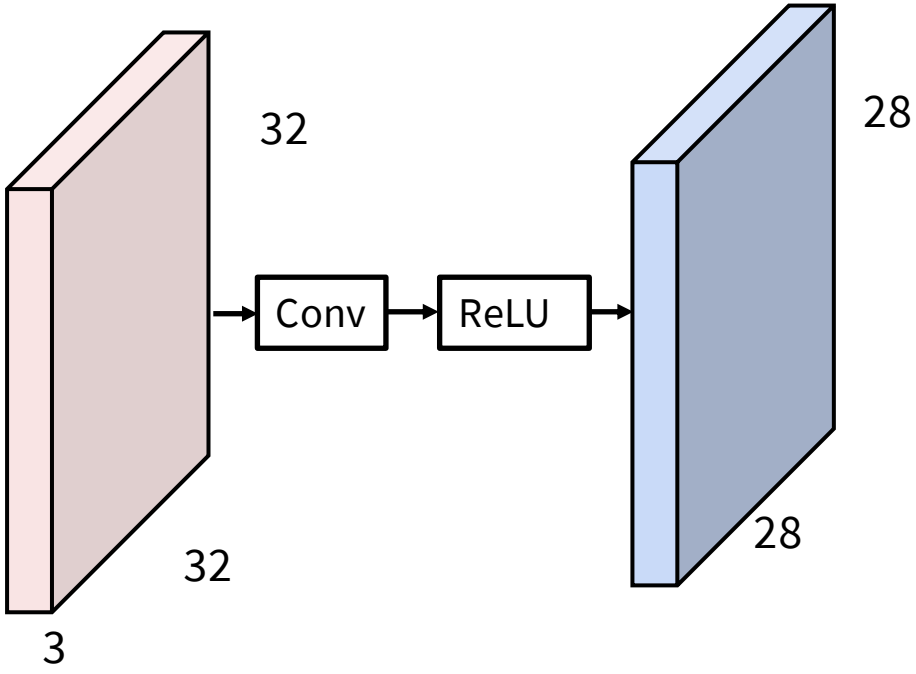
What do Conv filters learn?



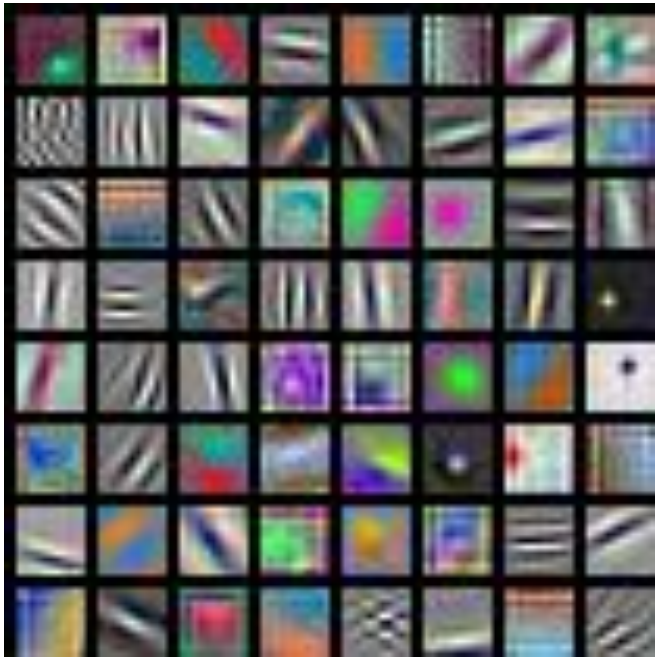
MLP: Bank of whole-image templates



What do Conv filters learn?

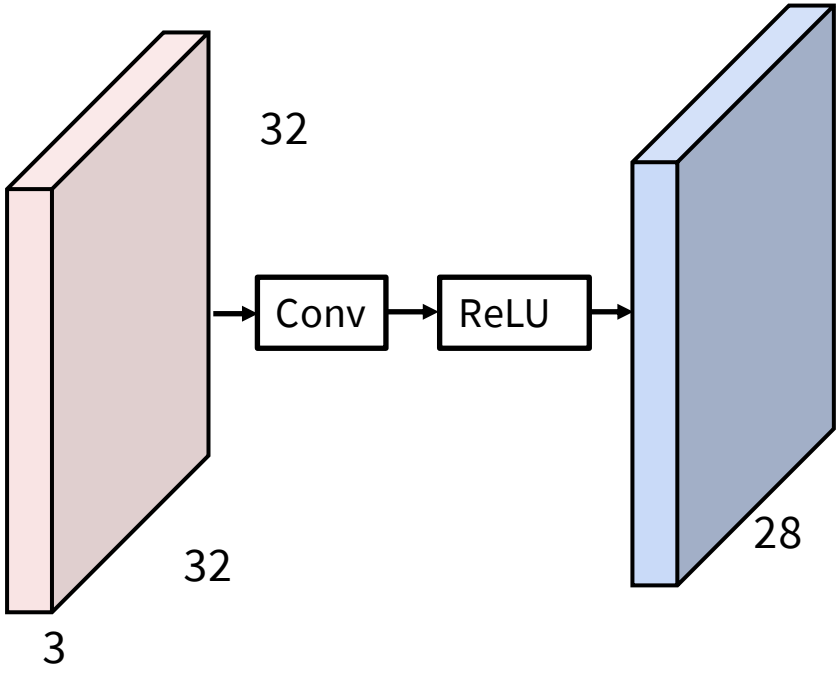


First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

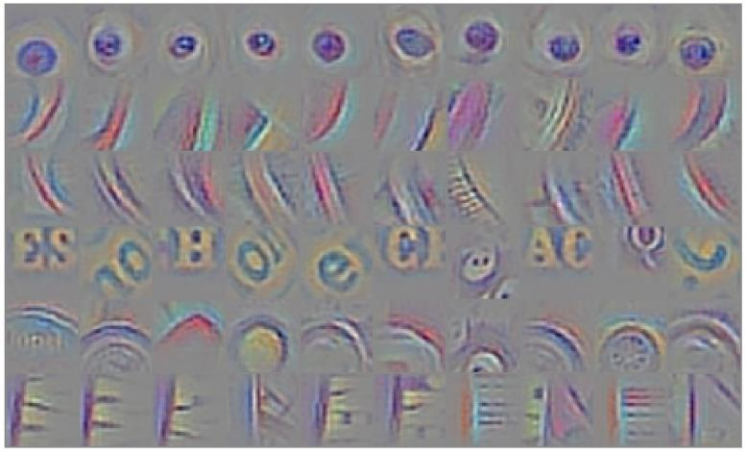


AlexNet: 64 filters, each 3x11x11

What do Conv filters learn?



Deeper conv layers: Harder to visualize
Tend to learn larger structures e.g. eyes, letters



6th layer conv layer from an ImageNet model

Visualization from [Springenberg et al, ICLR 2015]

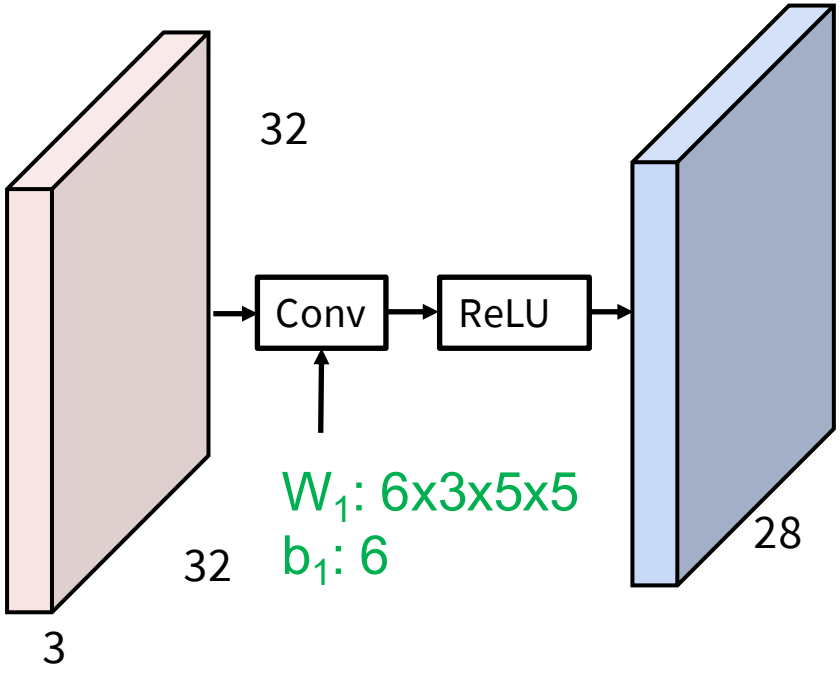
SOURCE AND TRANSITION

Output size and receptive field

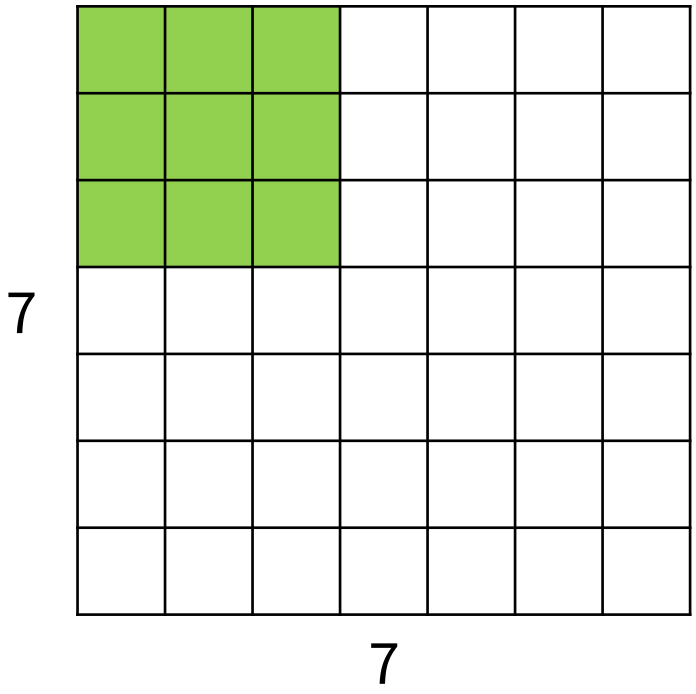
Filter size, padding, and stride determine spatial dimensions. Stacking layers expands the region of the input that can affect each activation.

Source: Stanford CS231n, Lecture 5, slides 69-87. https://cs231n.stanford.edu/slides/2026/lecture_5.pdf

Convolution: Spatial Dimensions

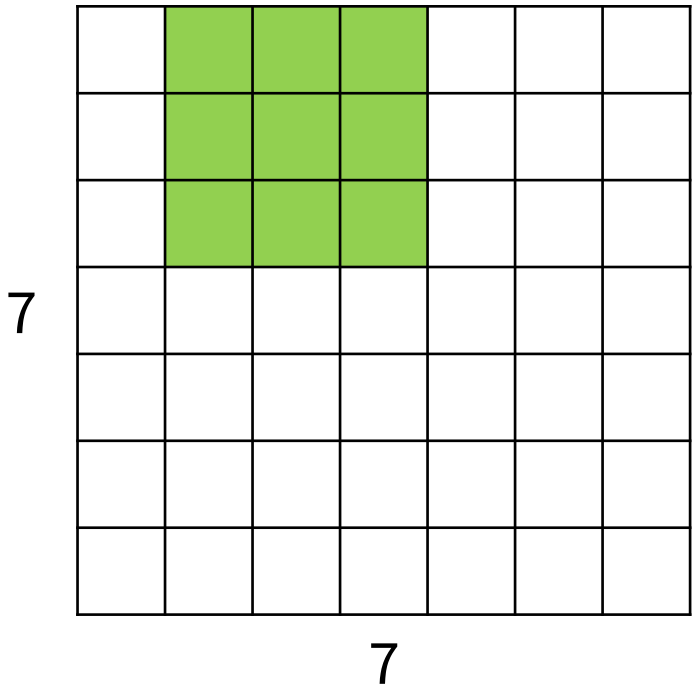


Convolution: Spatial Dimensions



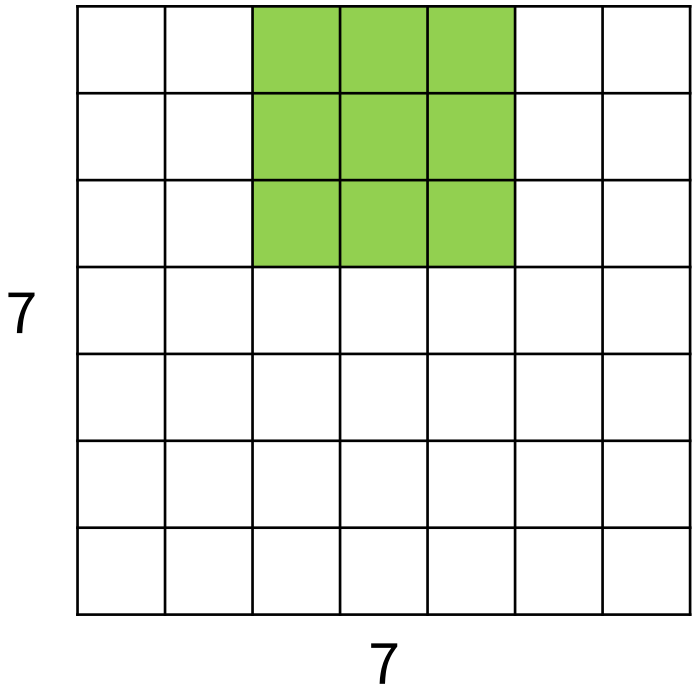
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



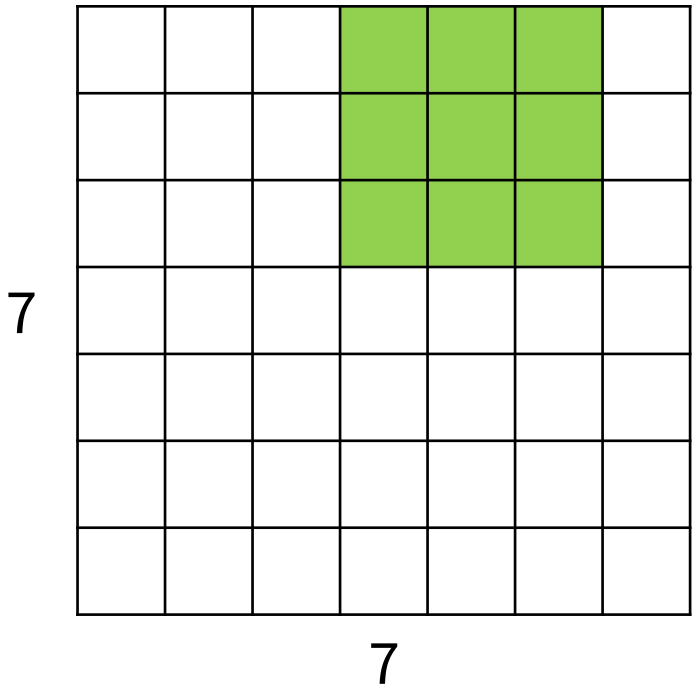
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



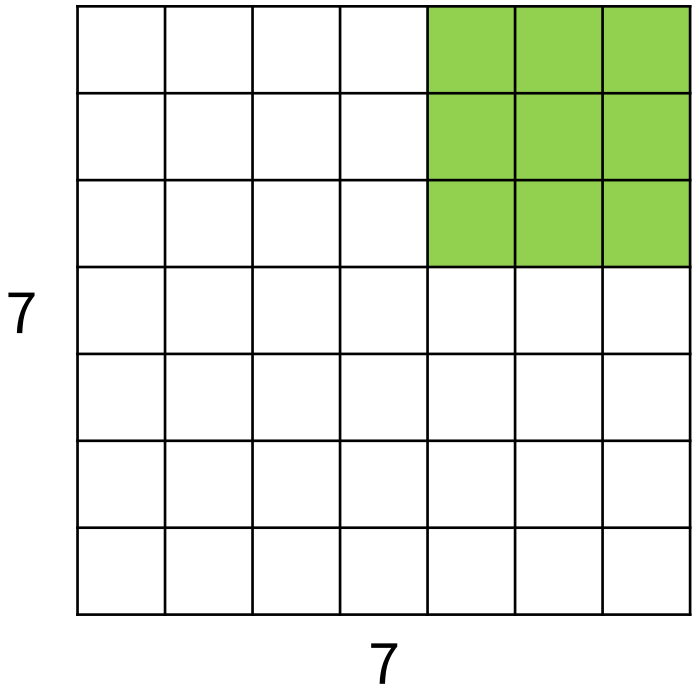
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



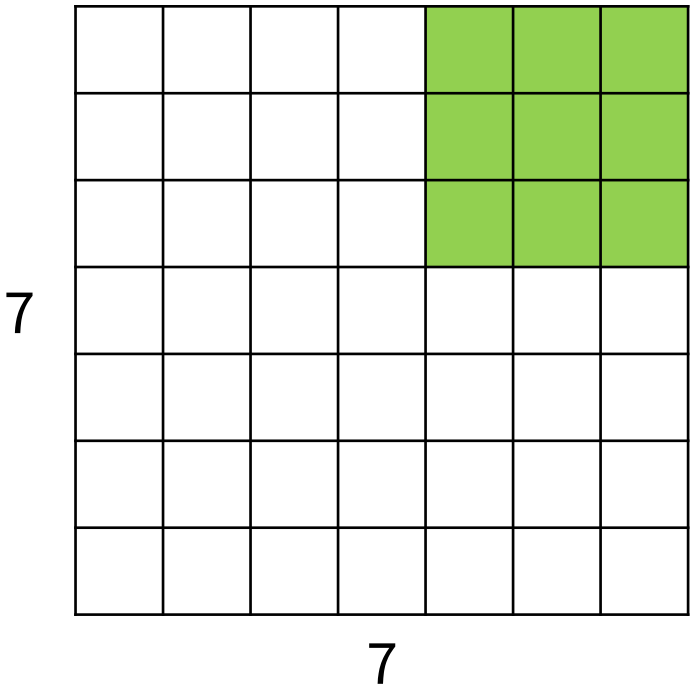
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



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

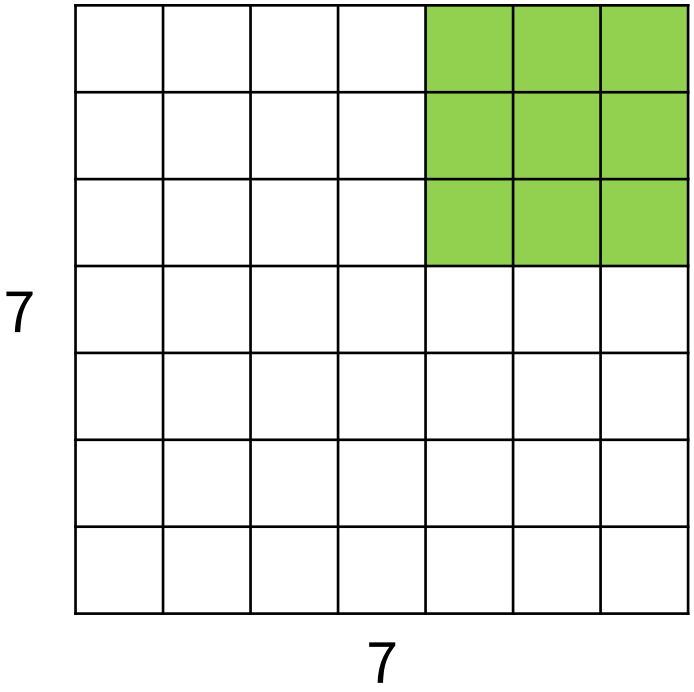
Convolution: Spatial Dimensions



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

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

Convolution: Spatial Dimensions

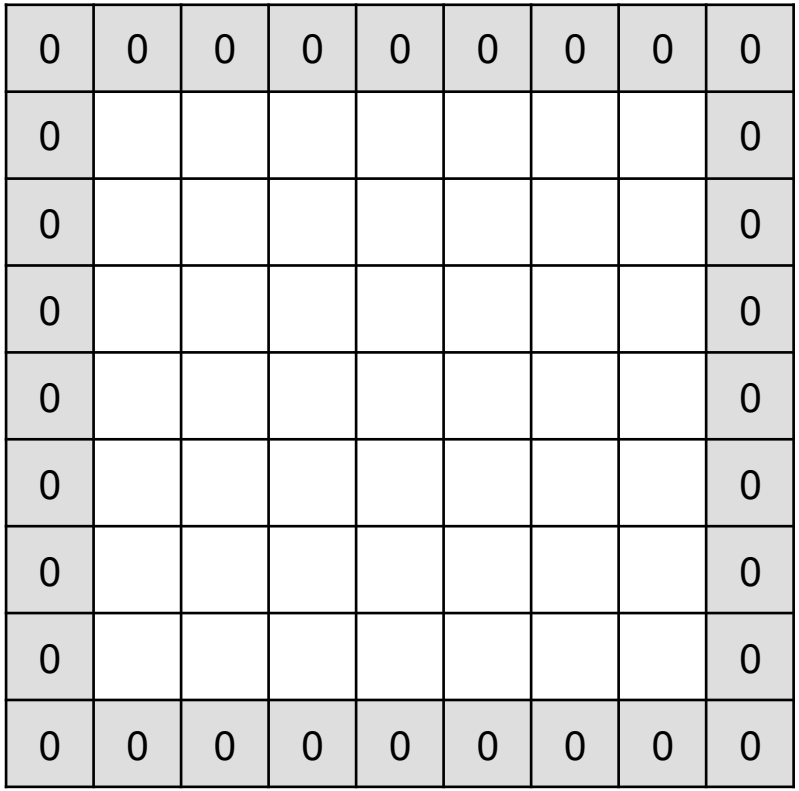


Input: 7x7
Filter: 3x3
Output: 5x5

Problem: Feature maps shrink with each layer!

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

Convolution: Spatial Dimensions



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

Problem: Feature maps shrink with each layer!

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

Solution: Add **padding** around the input before sliding the filter

Convolution: Spatial Dimensions

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

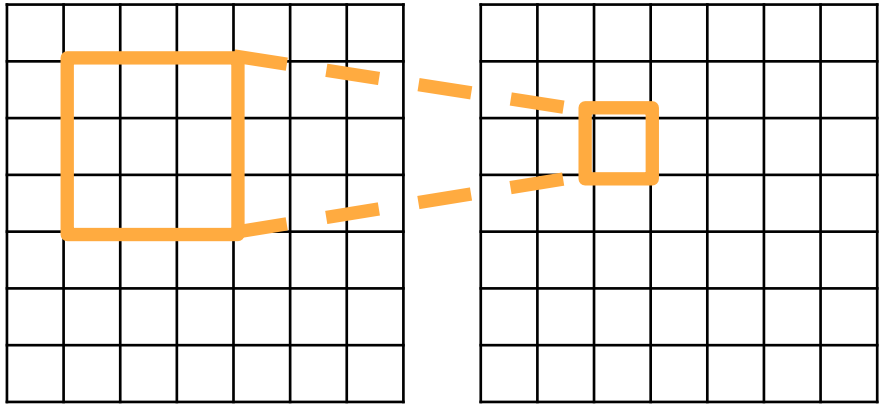
Input: 7x7
Filter: 3x3
Output: 5x5

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

Common setting:
P = (K – 1) / 2
Means output has same size as input

Receptive Fields

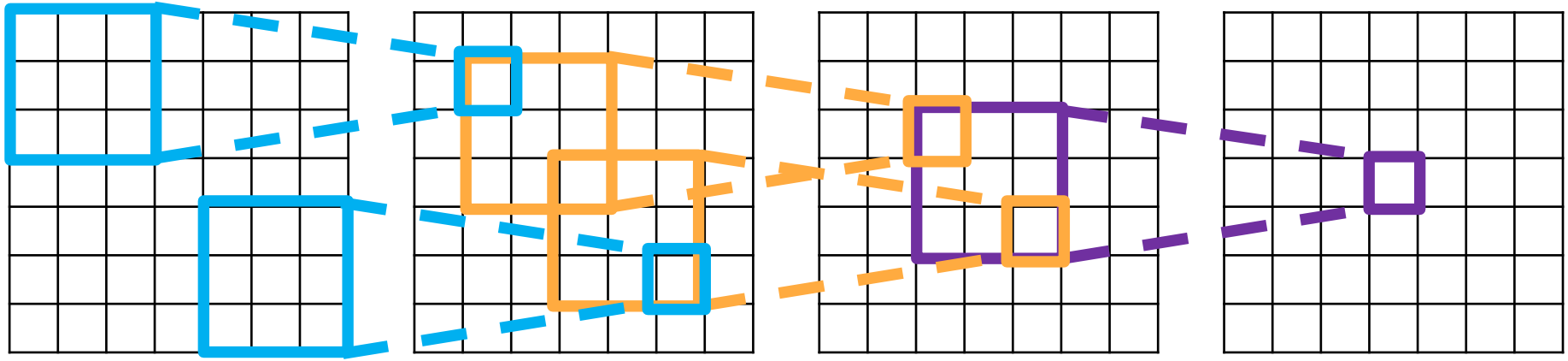
For convolution with **kernel size K**, each element in the output depends on a K x K receptive field in the input



Slide inspiration: Justin 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

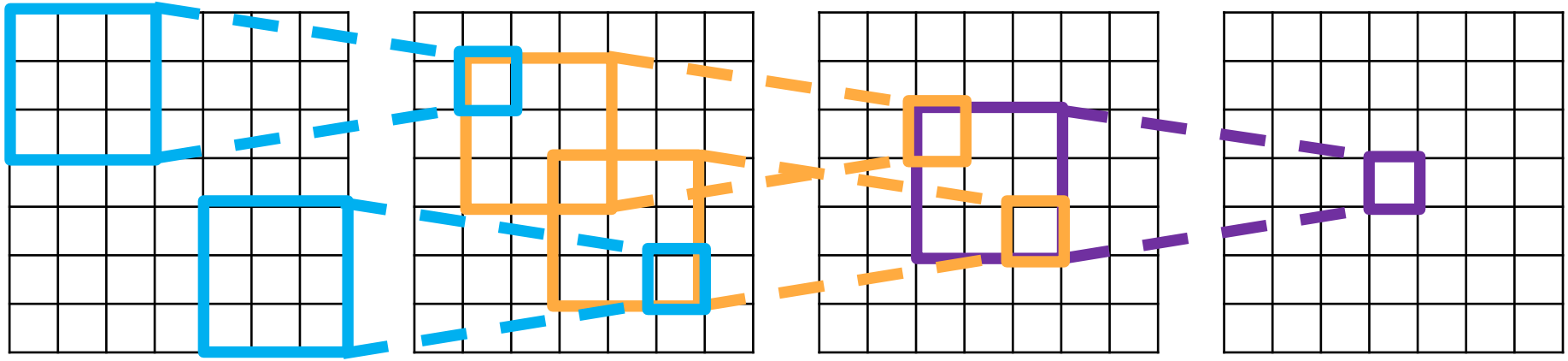
Output

Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

Slide inspiration: Justin 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

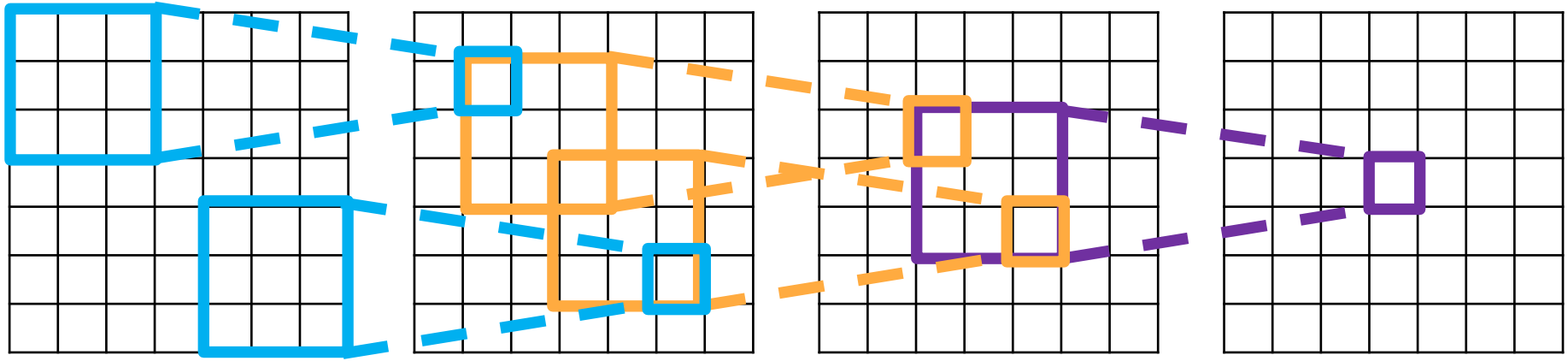
Problem: For large images we need many layers for each output to “see” the whole image image

Output

Slide inspiration: Justin 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

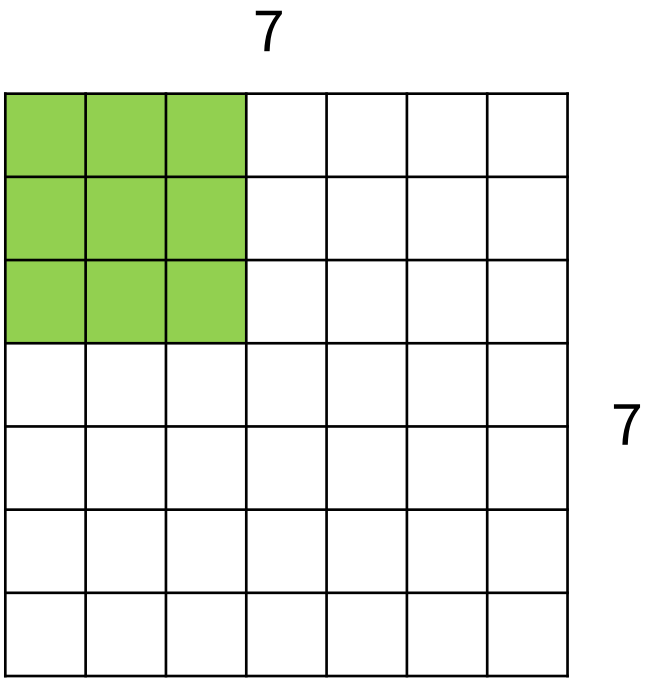
Output

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

Solution: Downsample inside the network

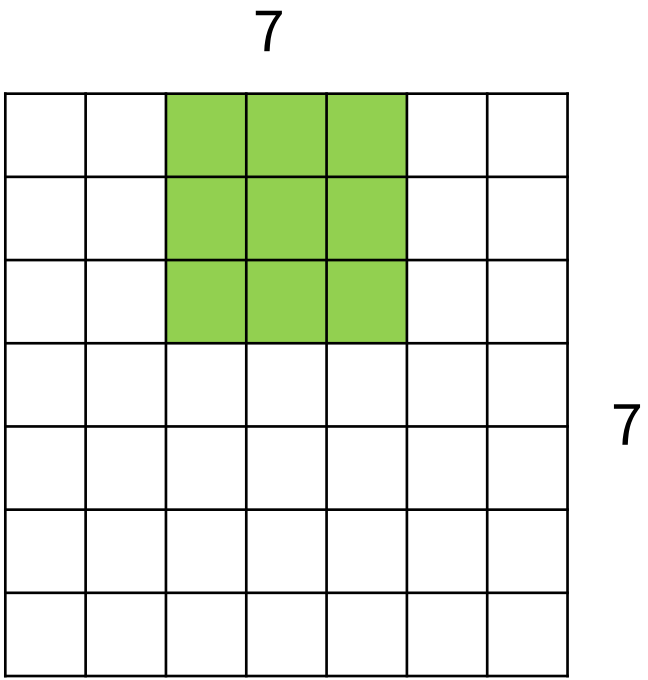
Slide inspiration: Justin Johnson

Strided Convolution



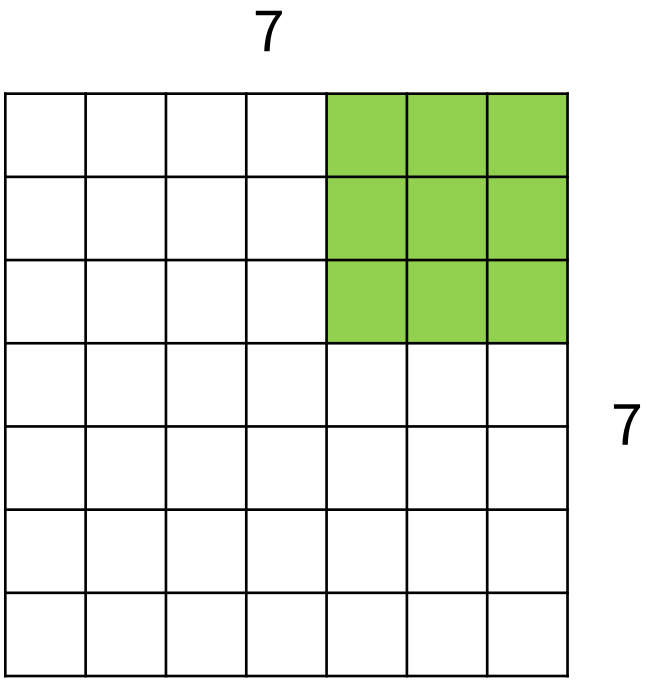
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



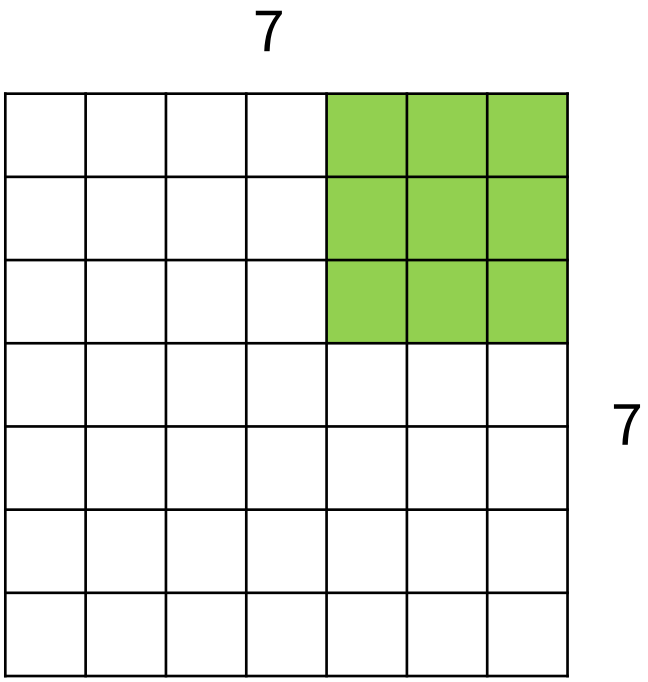
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3

Strided Convolution



Input: 7x7
Filter: 3x3
Stride: 2
Output: 3x3

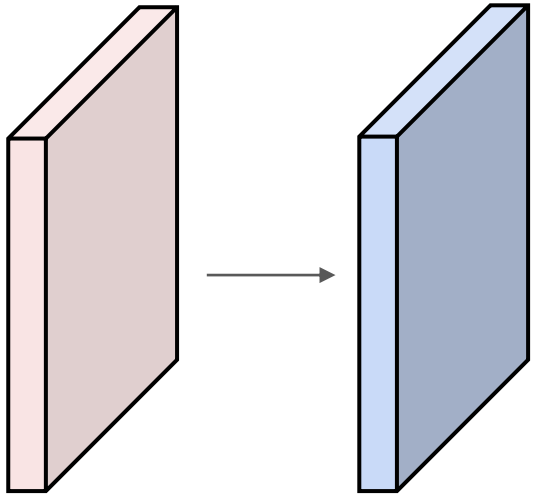
In general:
Input: W
Filter: K
Padding: P
Stride: S

→ Output:
(W – K + 2P) / S + 1

Convolution Example

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

Output volume size: ?



SOURCE AND TRANSITION

Pooling and translation equivariance

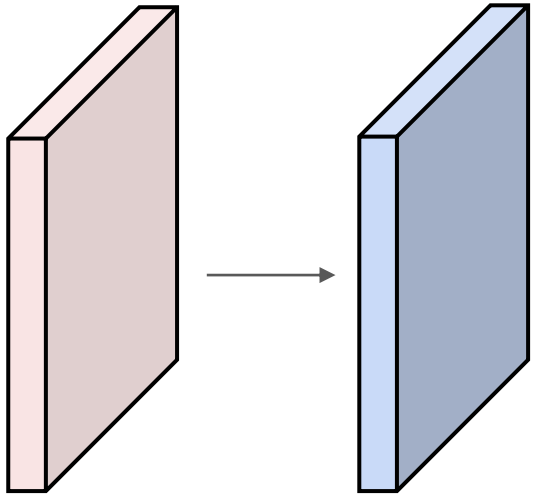
Pooling reduces spatial resolution. Convolution is translation equivariant: shifting the input shifts the feature map. This property helps explain what a CNN represents and where its limits are.

Source: Stanford CS231n, Lecture 5, slides 88-107. https://cs231n.stanford.edu/slides/2026/lecture_5.pdf

Convolution Example

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

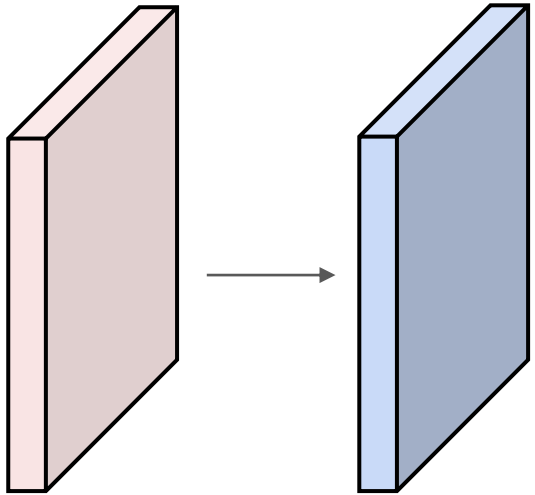
Output volume size: 10 x 32 x 32
32 = (32+2*2-5)/1+1



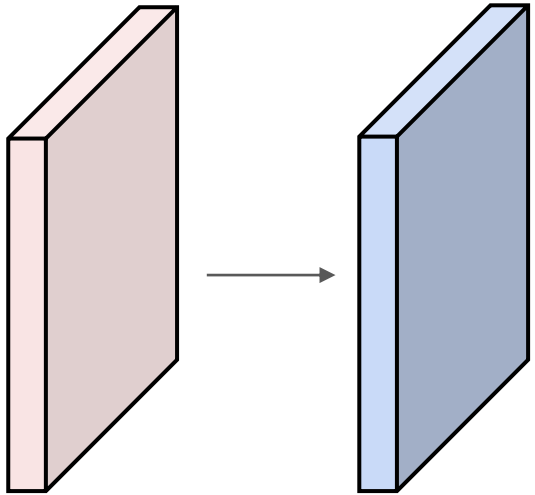
Convolution Example

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: ?



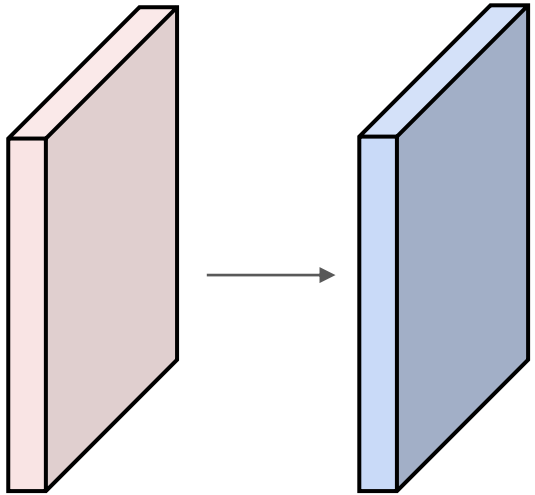
Convolution Example



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**

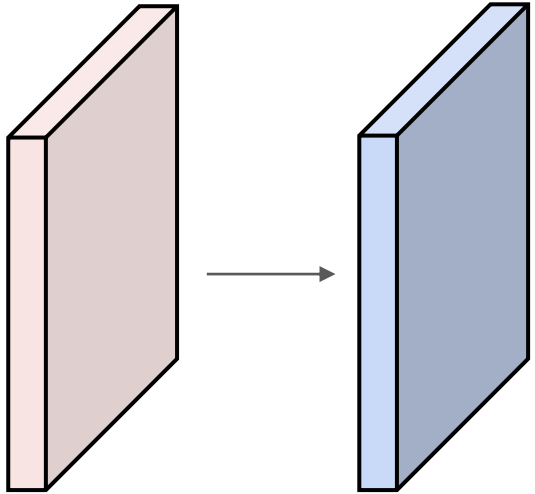
Convolution Example



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**

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

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)

PyTorch Convolution Layer

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

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

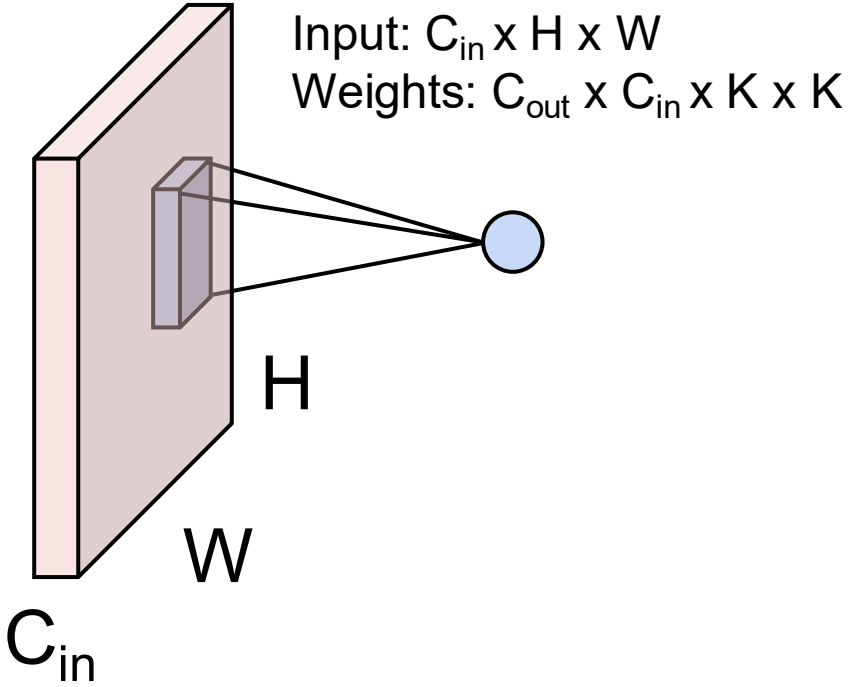
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

We didn't talk about groups or dilation…

Other Types of Convolution

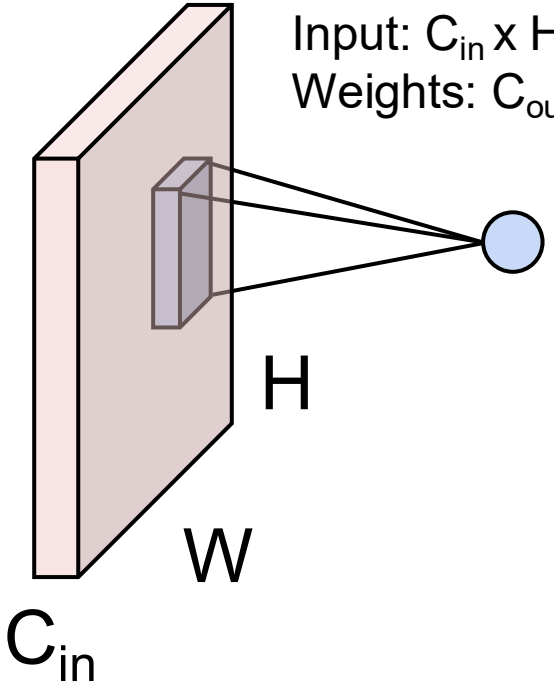
So far: 2D Convolution



Other Types of Convolution

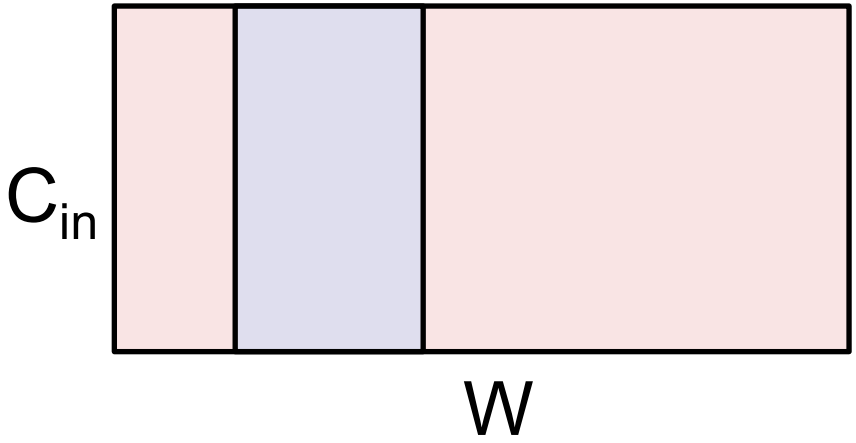
So far: 2D Convolution

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



1D Convolution

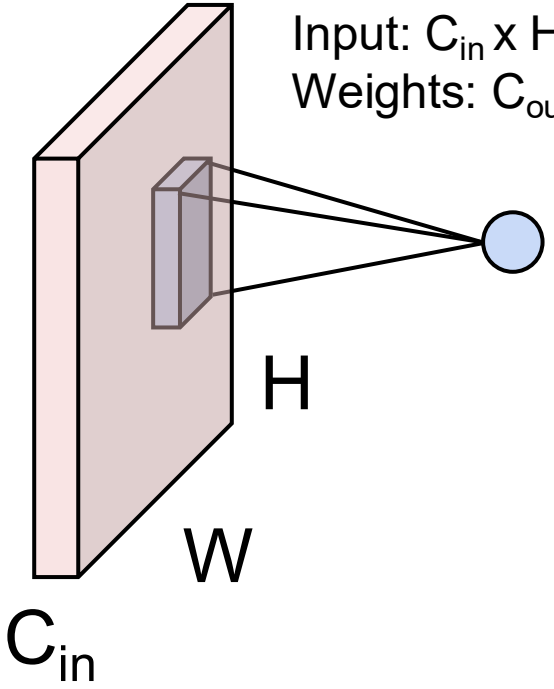
Input: C_{in} x W
Weights: C_{out} x C_{in} x K



Other Types of Convolution

So far: 2D Convolution

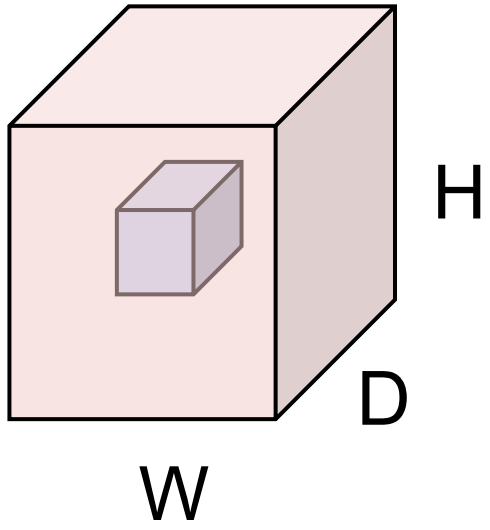
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



3D Convolution

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

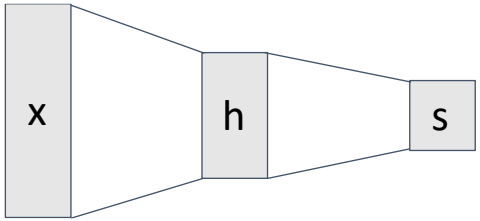
C_{in}-dim vector
at each point
in the volume



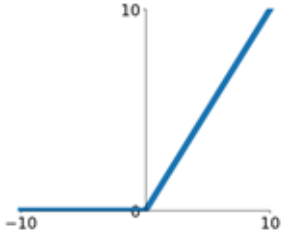
Convolutional Networks

We have already seen these

Fully-Connected Layer

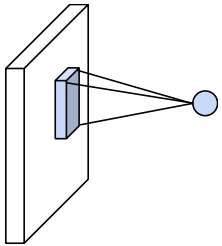


Activation Function

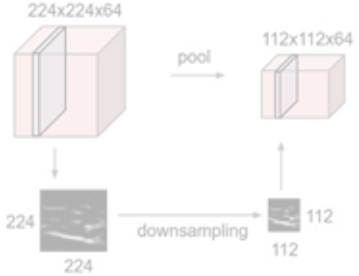


Today: Image-specific operators

Convolution Layer



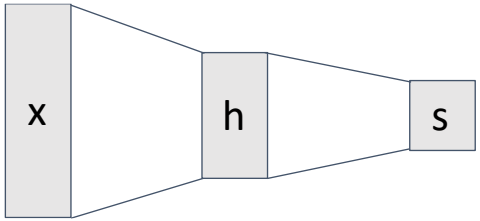
Pooling Layer



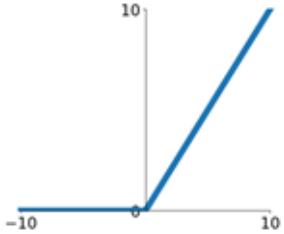
Convolutional Networks

We have already seen these

Fully-Connected Layer

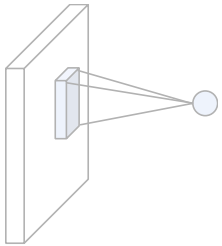


Activation Function

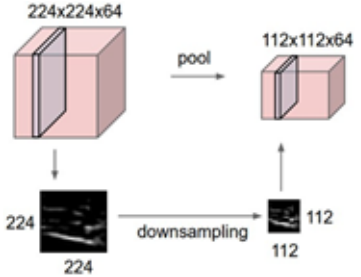


Today: Image-specific operators

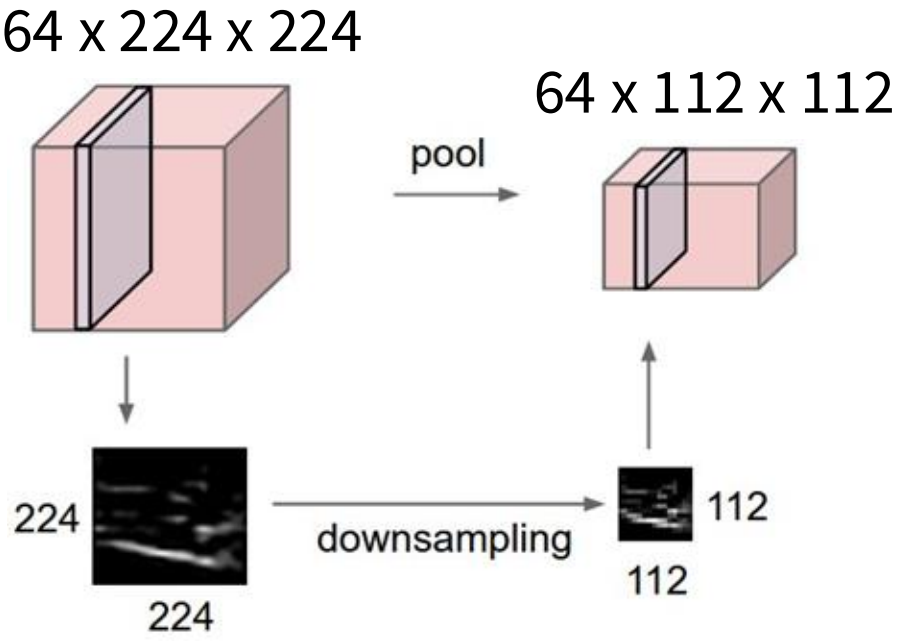
Convolution Layer



Pooling Layer



Pooling Layers: Another way to downsample



Given an input C x H x W,
downsample each 1 x H x W plane

Hyperparameters:
Kernel Size
Stride
Pooling function

Pooling Layers: Another way to downsample

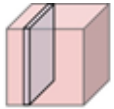
Single depth slice

x

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

y

64 x 224 x 224



Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Gives **invariance** to small spatial shifts. No learnable parameters.

Pooling Summary

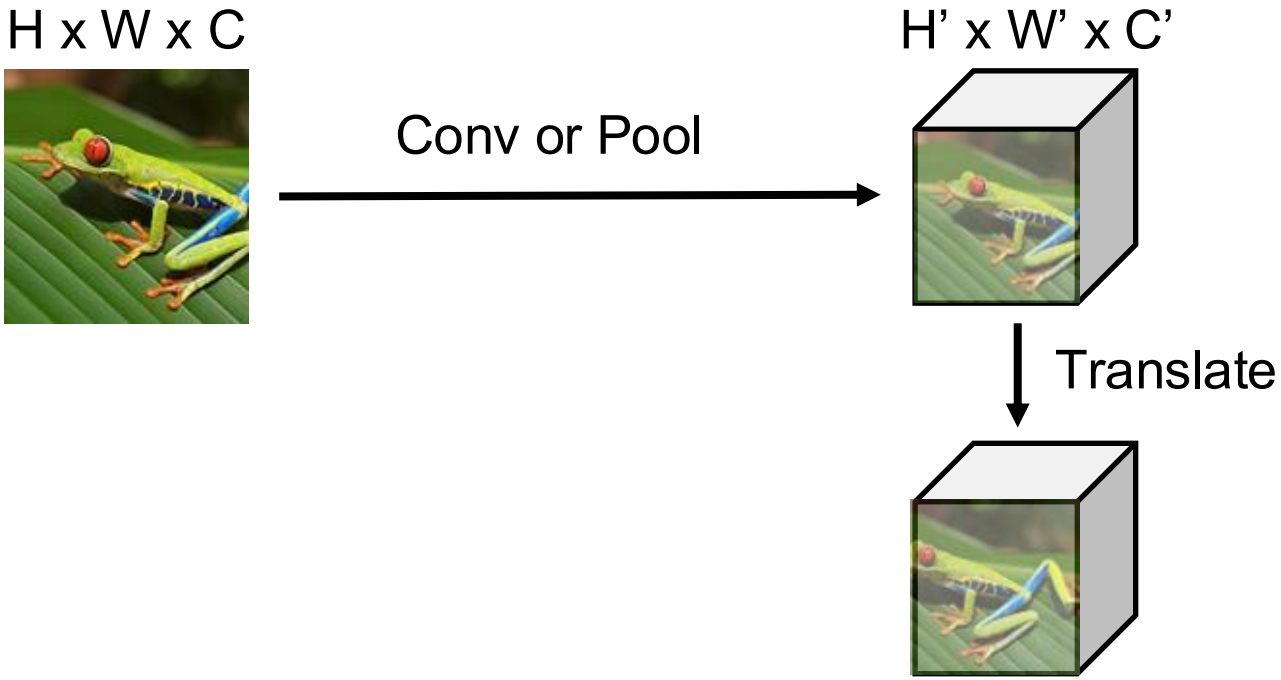
Input: C x H x W
Hyperparameters:
- **Kernel size**: K
- **Stride**: S
- **Pooling function**: max, avg

Output size: C x H' x W' where:
- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

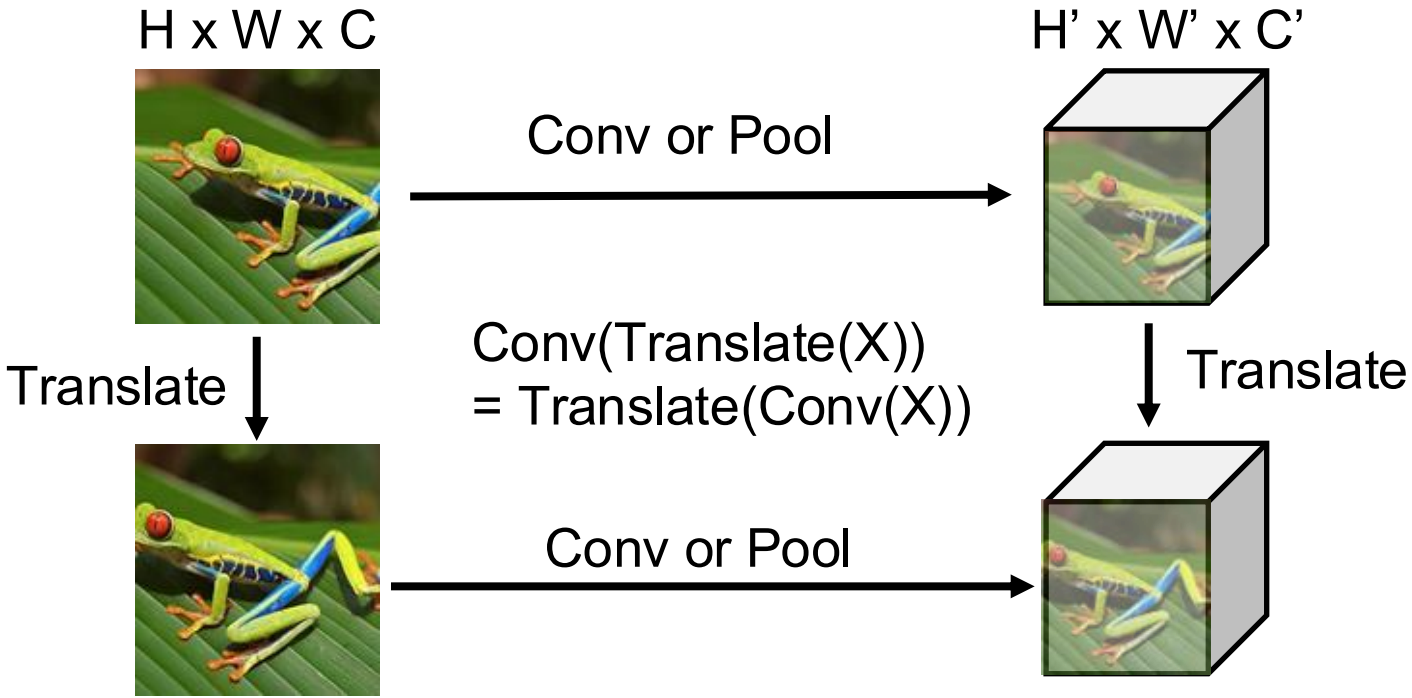
No learnable parameters

Common setting:
max, K=2, S=2 => Gives 2x downsampling

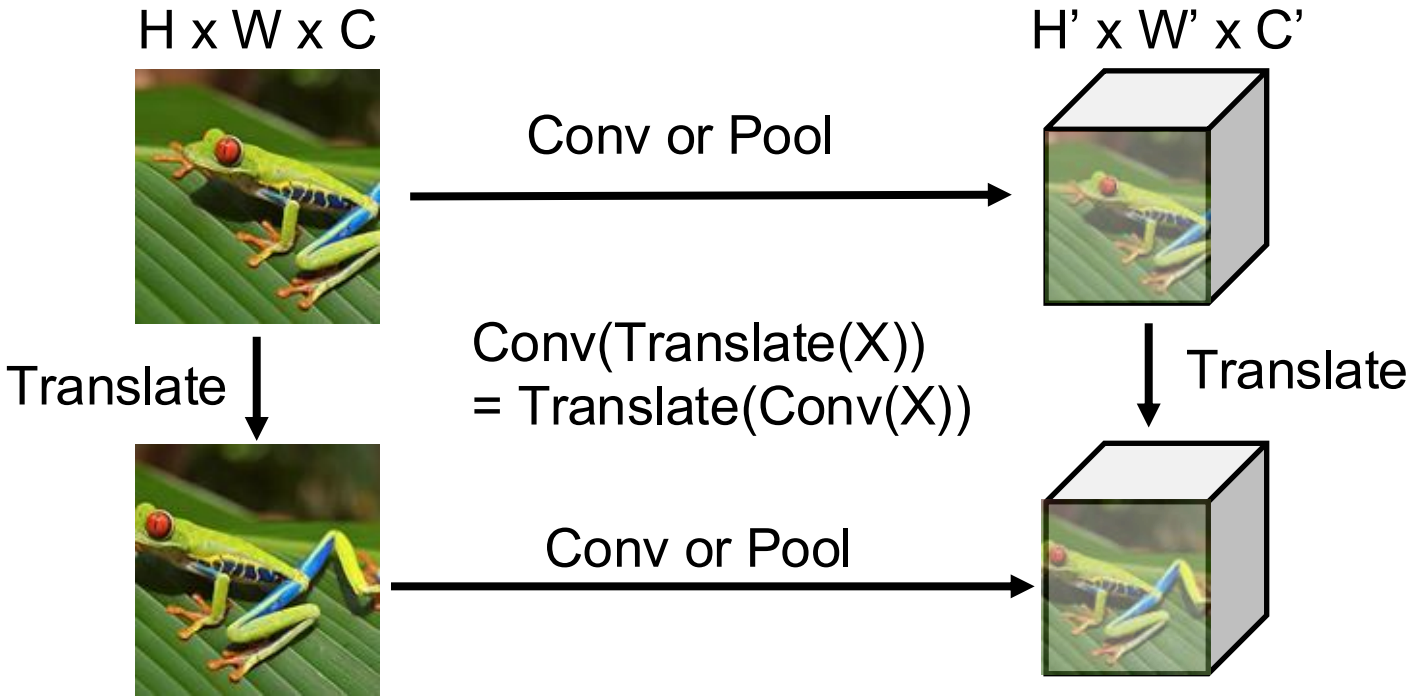
Convolution and Pooling: Translation Equivariance



Convolution and Pooling: Translation Equivariance



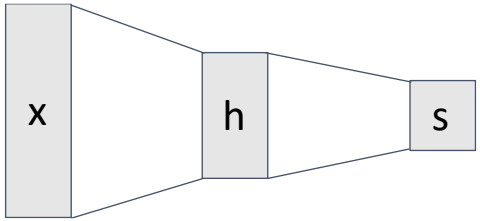
Convolution and Pooling: Translation Equivariance



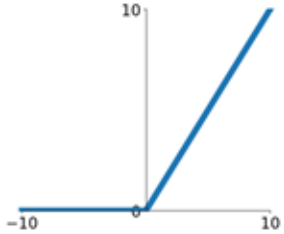
Intuition: Features of images don't depend on their location in the image

Summary: Convolutional Networks

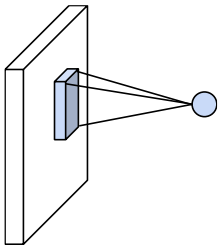
Fully-Connected Layer



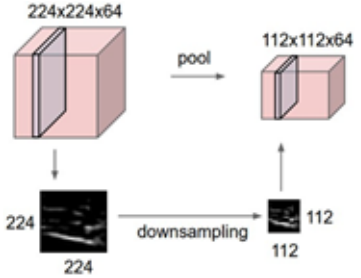
Activation Function



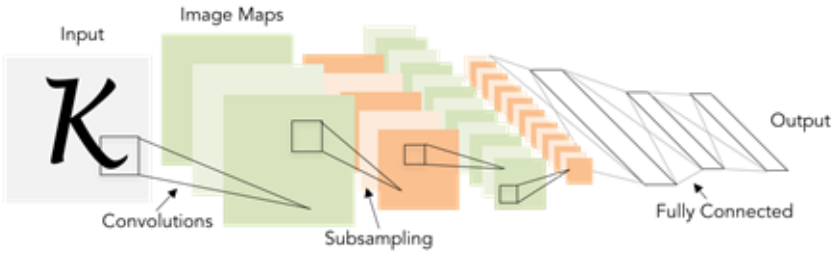
Convolution Layer

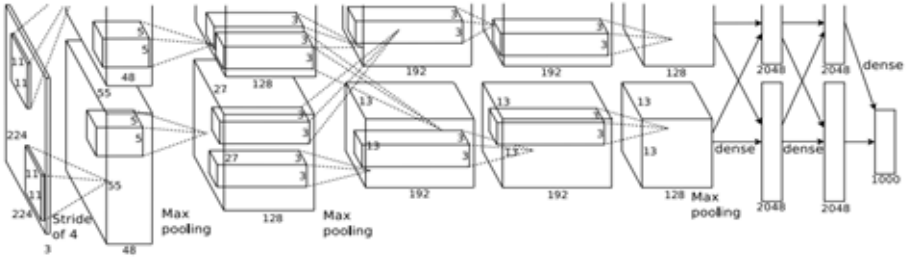


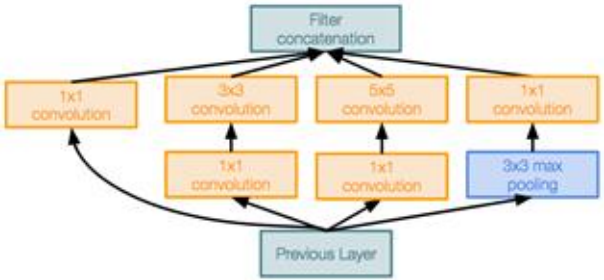
Pooling Layer

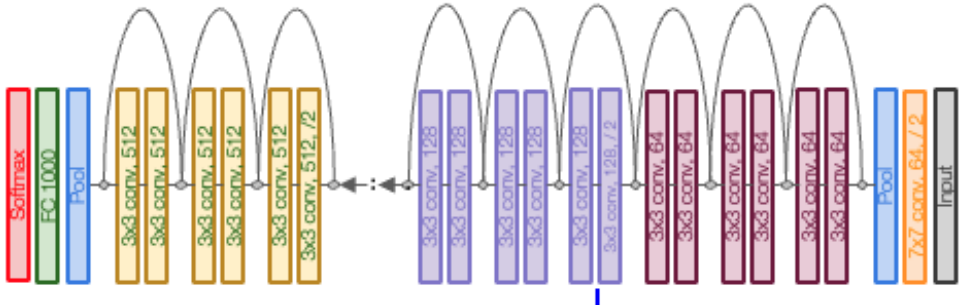


Next time: CNN Architectures









SECTION

From layers to architectures

A convolution layer is a building block. Networks differ in how they arrange convolutions, activations, normalization, pooling, and skip connections.

Transition written for this compilation. Next: Stanford CS231n Lecture 6.

SOURCE AND TRANSITION

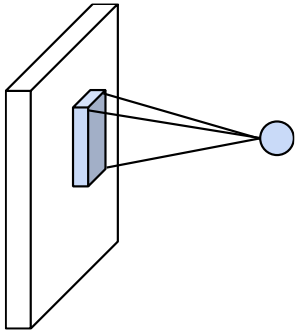
Activation functions and CNN architectures

Lecture 6 begins with normalization and activations, then studies VGG and ResNet. We keep the visual explanations most useful for understanding common CNN designs.

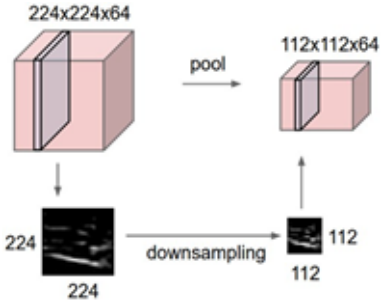
Source: Stanford CS231n, Lecture 6, activation and architecture slides. https://cs231n.stanford.edu/slides/2026/lecture_6.pdf

Components of CNNs

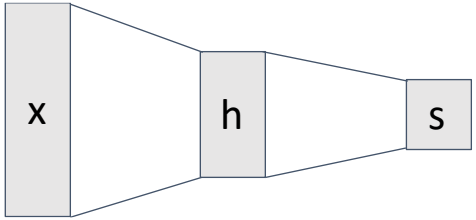
Convolution Layers



Pooling Layers



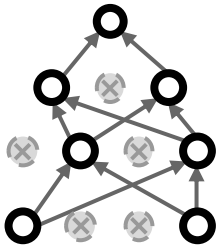
Fully-Connected Layers



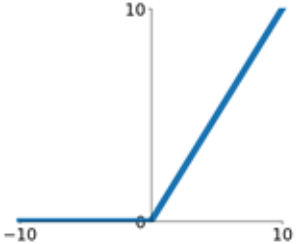
Normalization Layers

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

Dropout (sometimes)

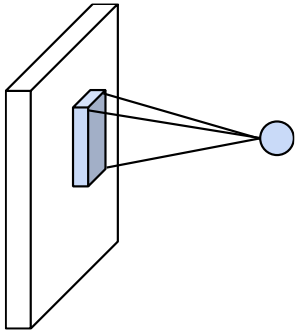


Activation Functions

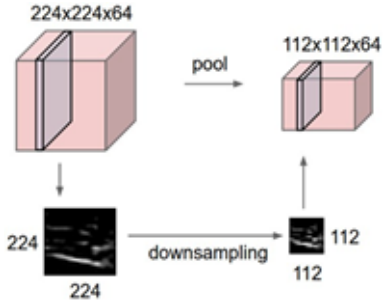


Components of CNNs

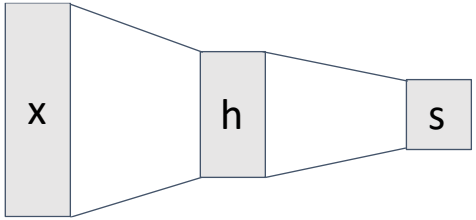
Convolution Layers



Pooling Layers



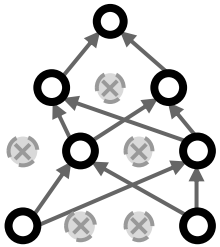
Fully-Connected Layers



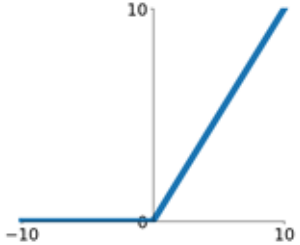
Normalization Layers

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

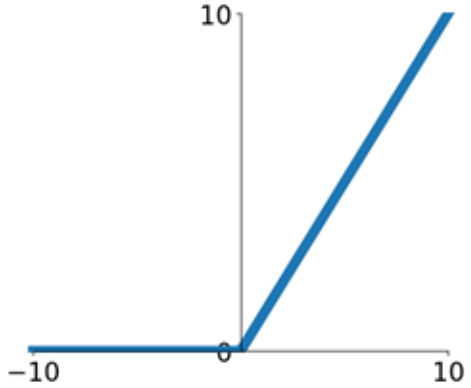
Dropout (sometimes)



Activation Functions



Activation Functions



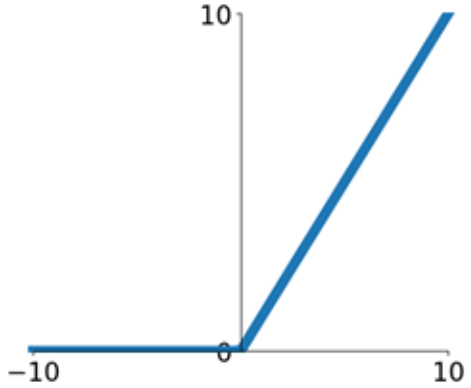
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice (e.g. 6x)

[Krizhevsky et al., 2012]

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

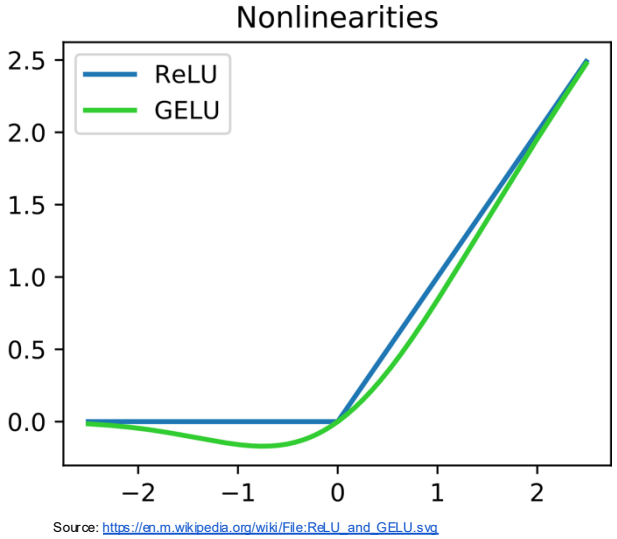
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid in practice (e.g. 6x)

- Not zero-centered output
- An annoyance:

Dead ReLUs when x < 0!

Activation Functions

[Hendrycks et al., 2016]



Source: https://en.m.wikipedia.org/wiki/File:ReLU_and_GELU.svg

GELU
(Gaussian Error Linear Unit)

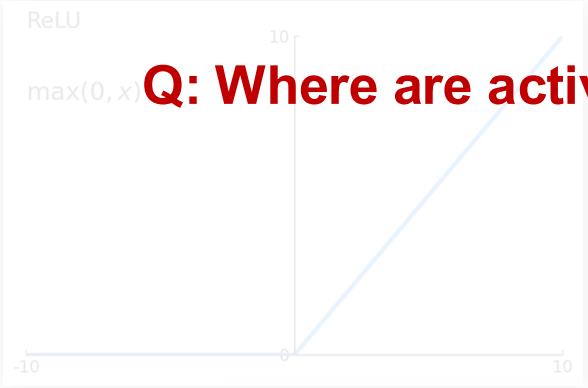
- Computes **f(x) = x*Φ(x)**

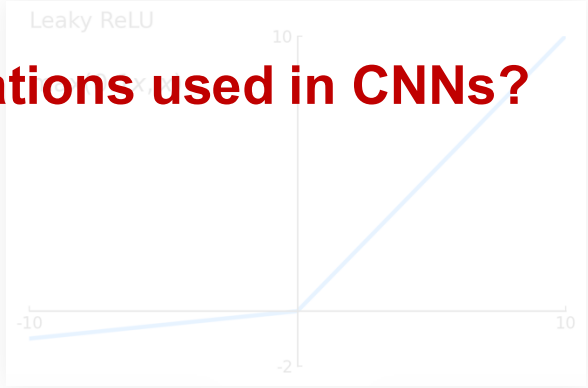
- Very nice behavior around 0
- Smoothness facilitates training in practice

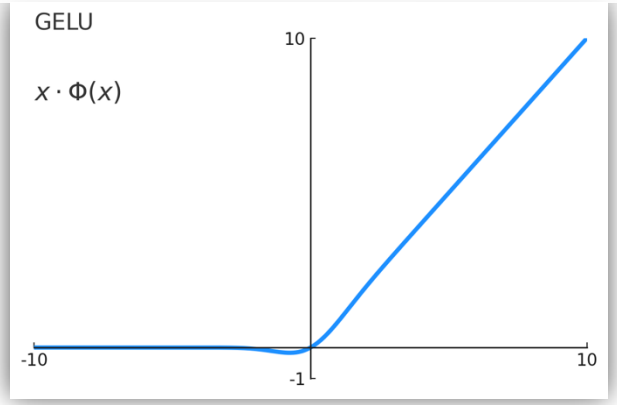
- Higher computational cost than ReLU
- Large negative values can still have gradient → 0

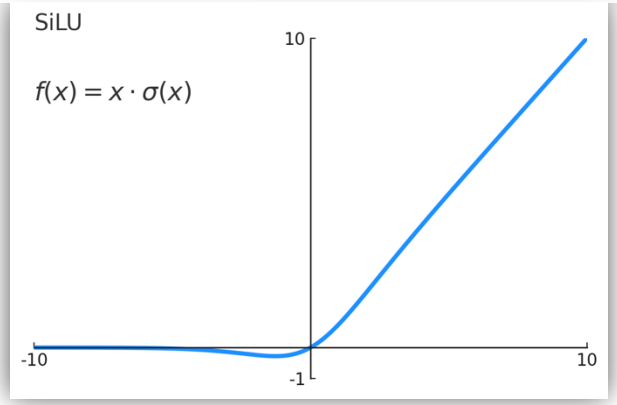
Activation Function Zoo

Q: Where are activations used in CNNs?





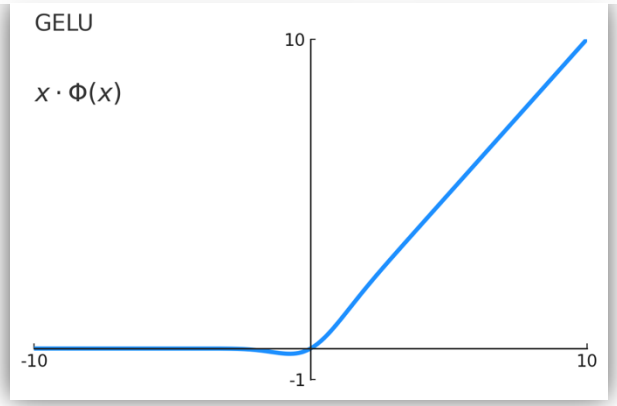


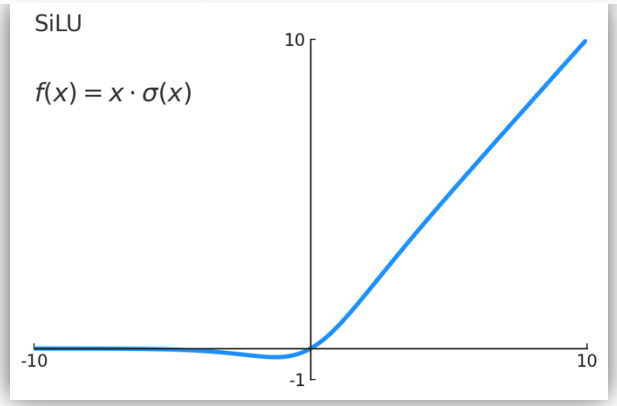


Activation Function Zoo

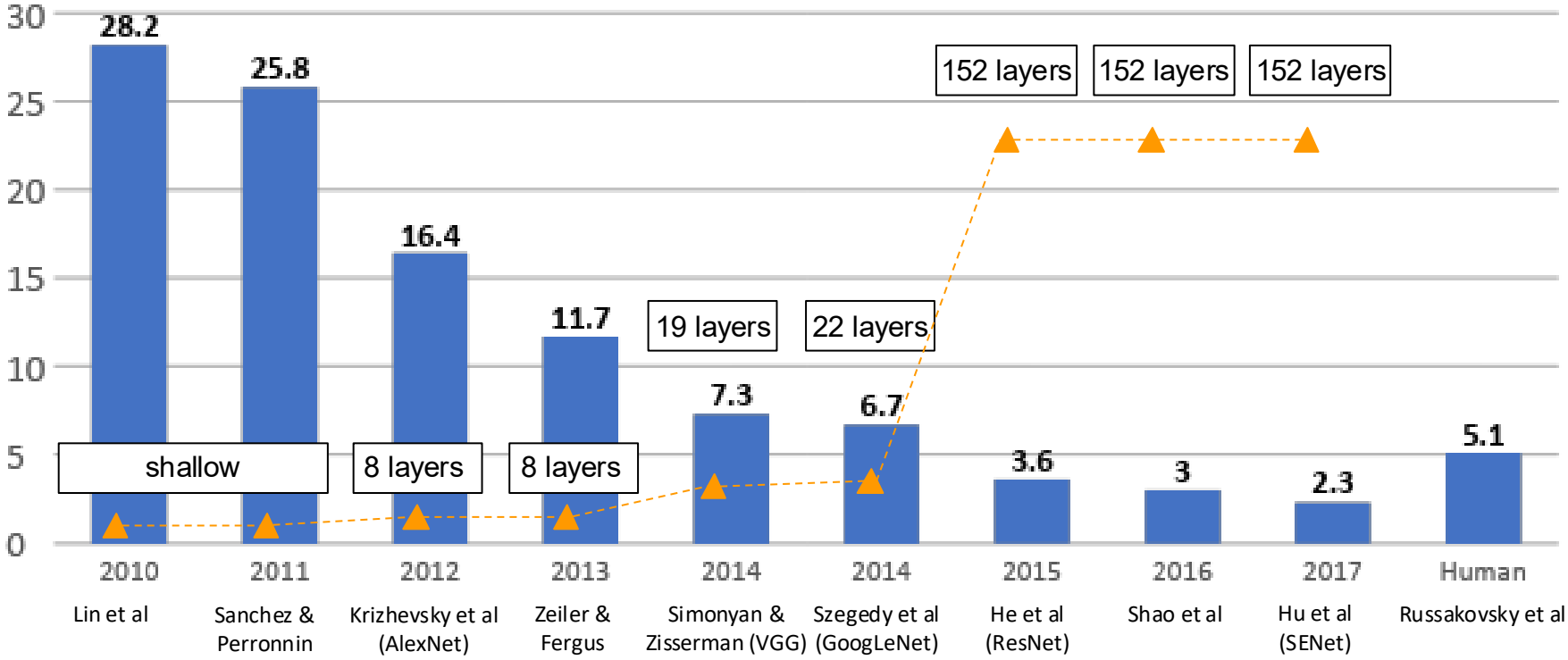
Q: Where are activations used in CNNs?

A: Generally placed after linear operators (feedforward/linear layer, convolutional layer, etc.)

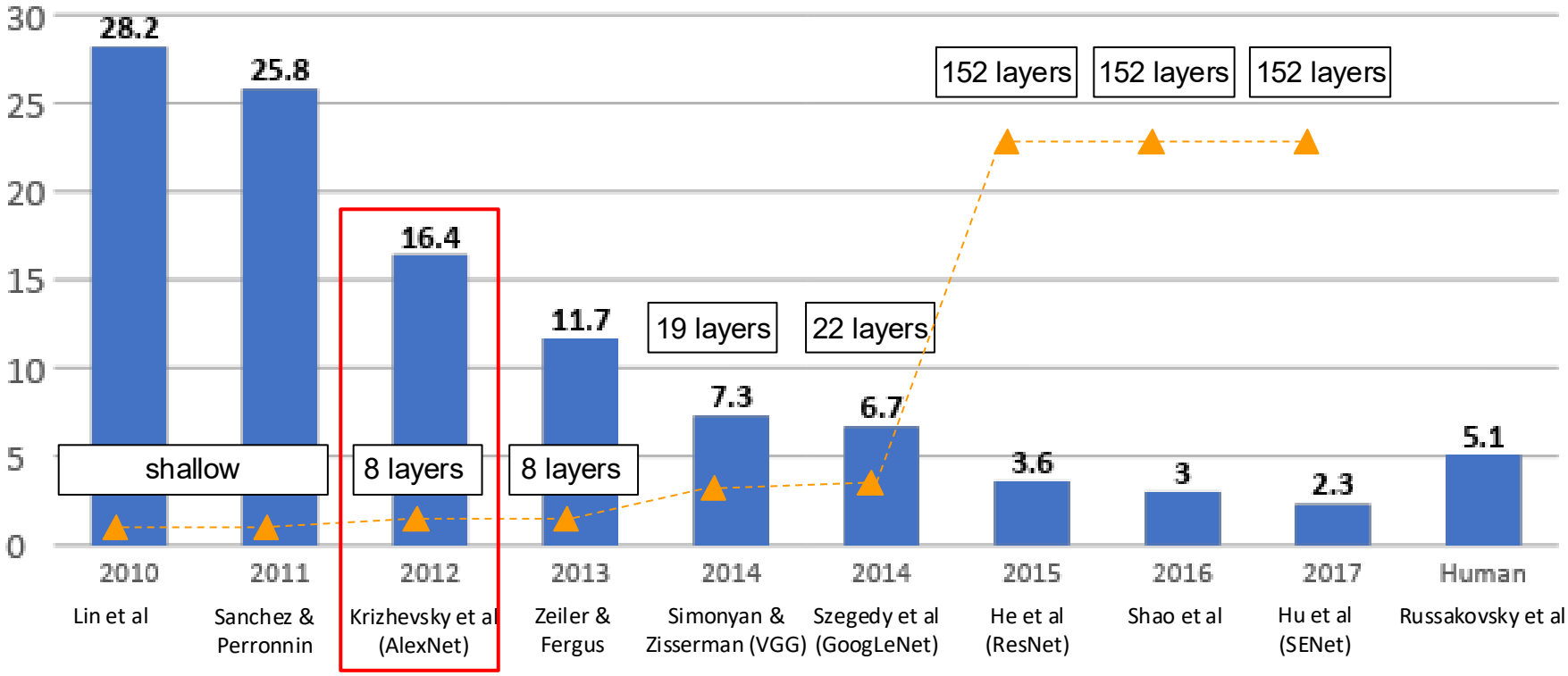




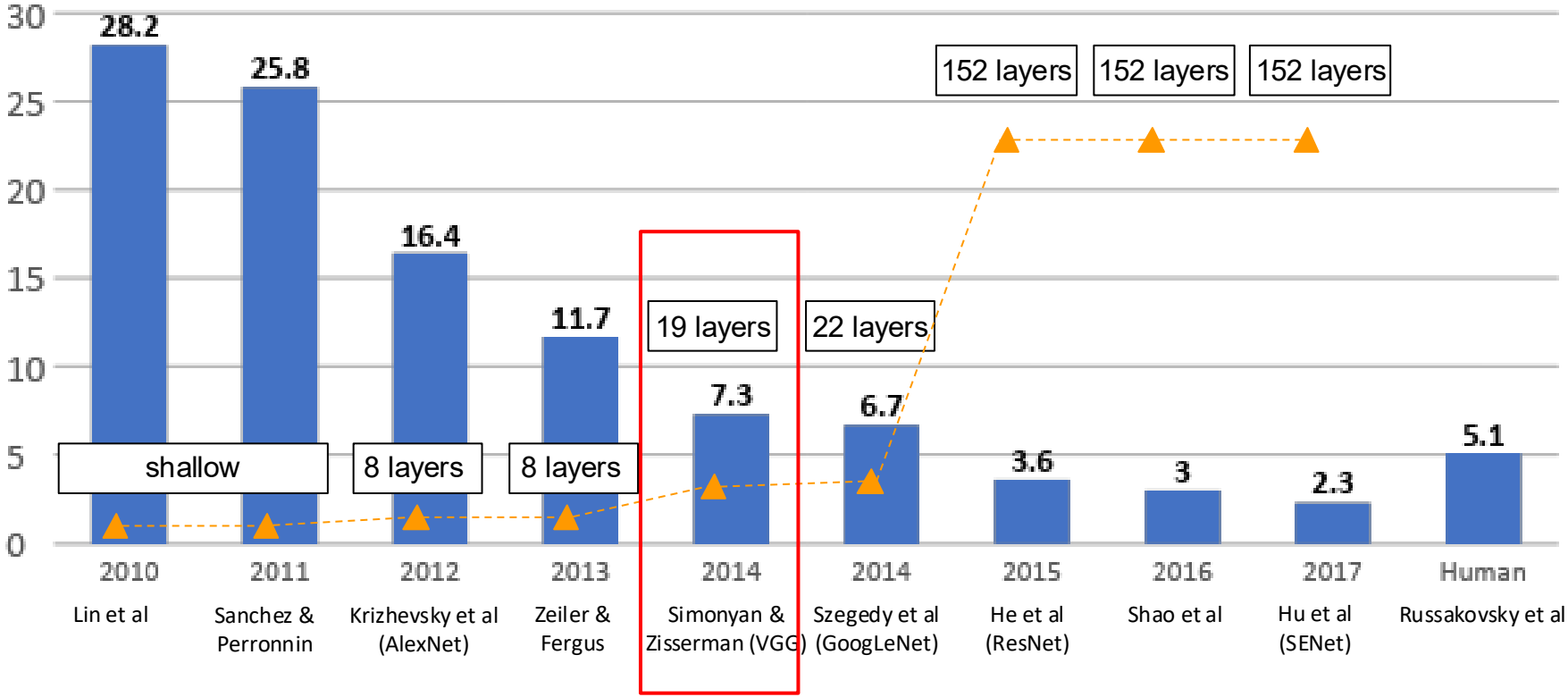
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

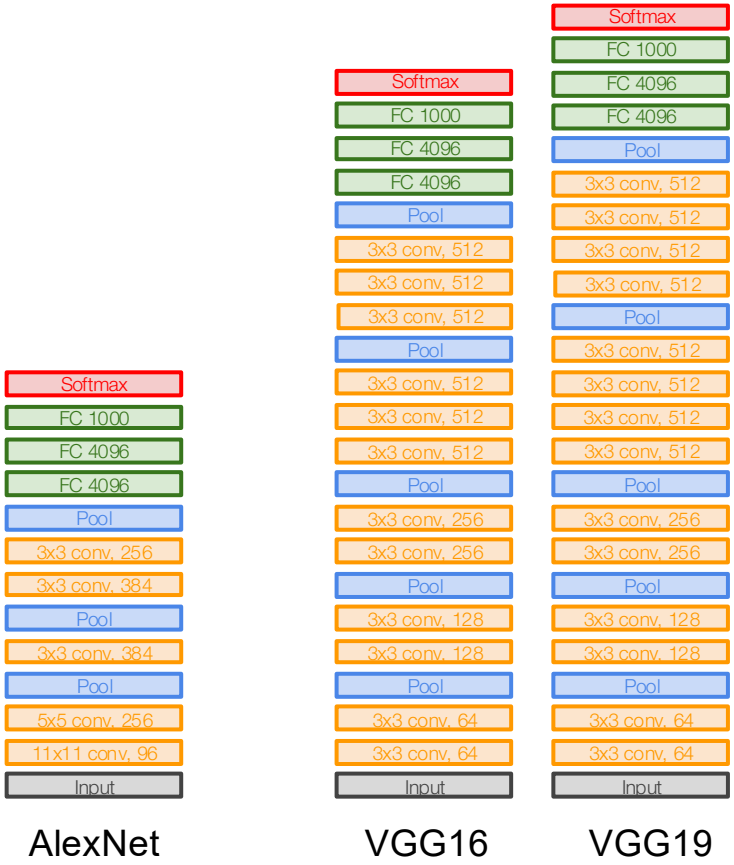
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

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

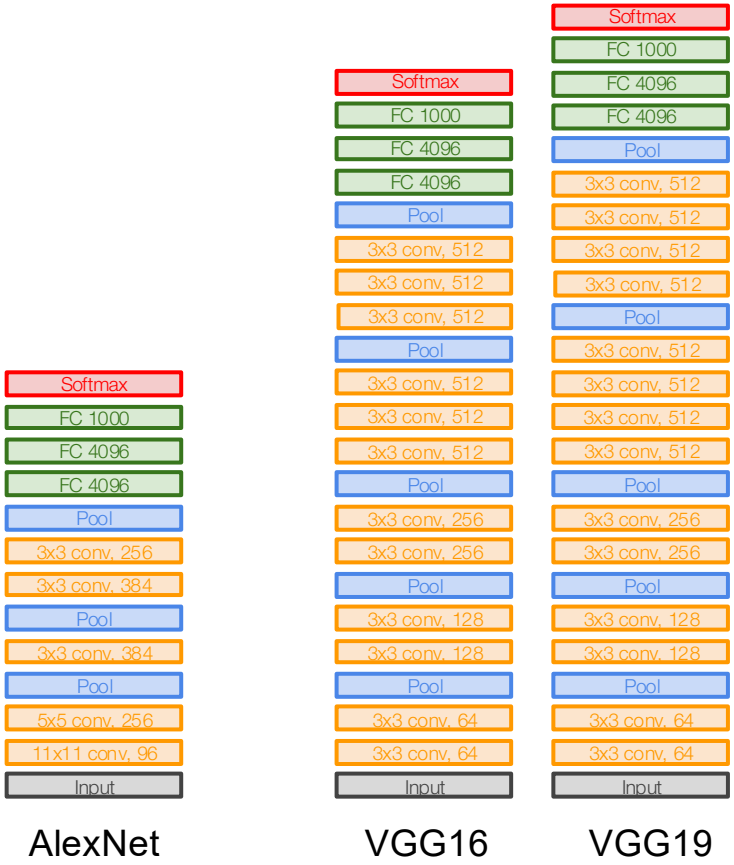
11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

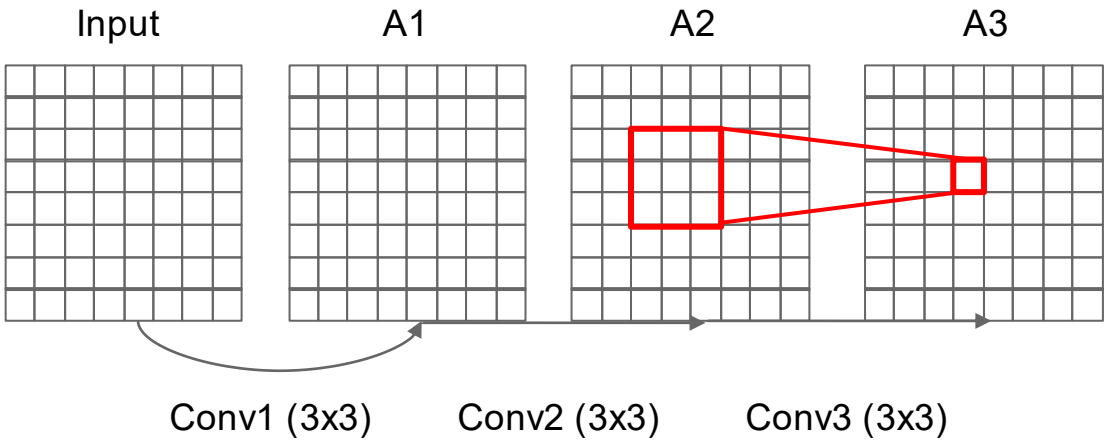
Q: Why use smaller filters? (3x3 conv)



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

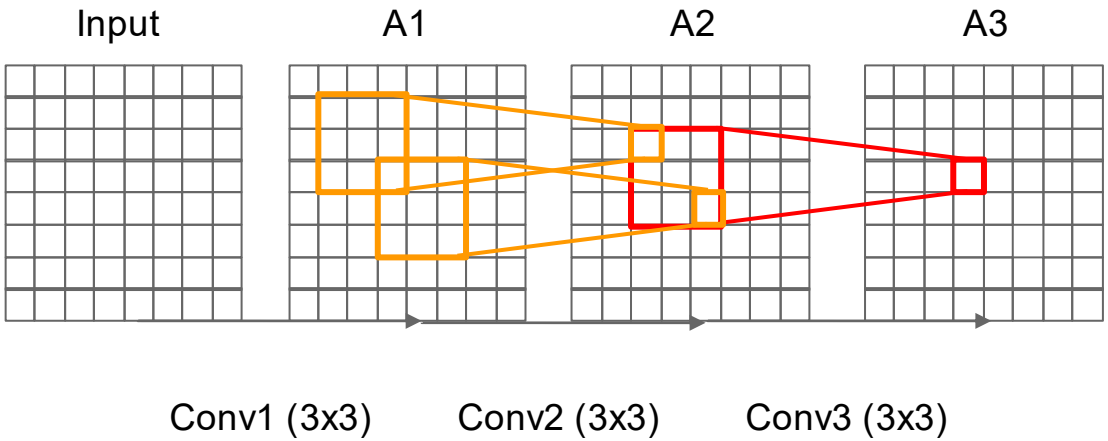




Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?





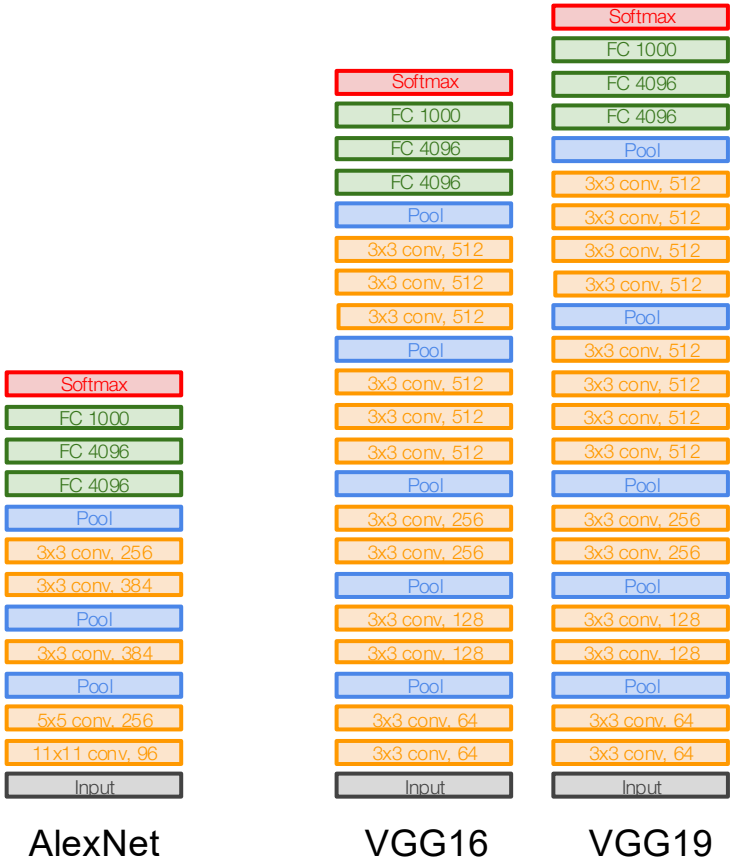
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

[7x7]



Case Study: VGGNet

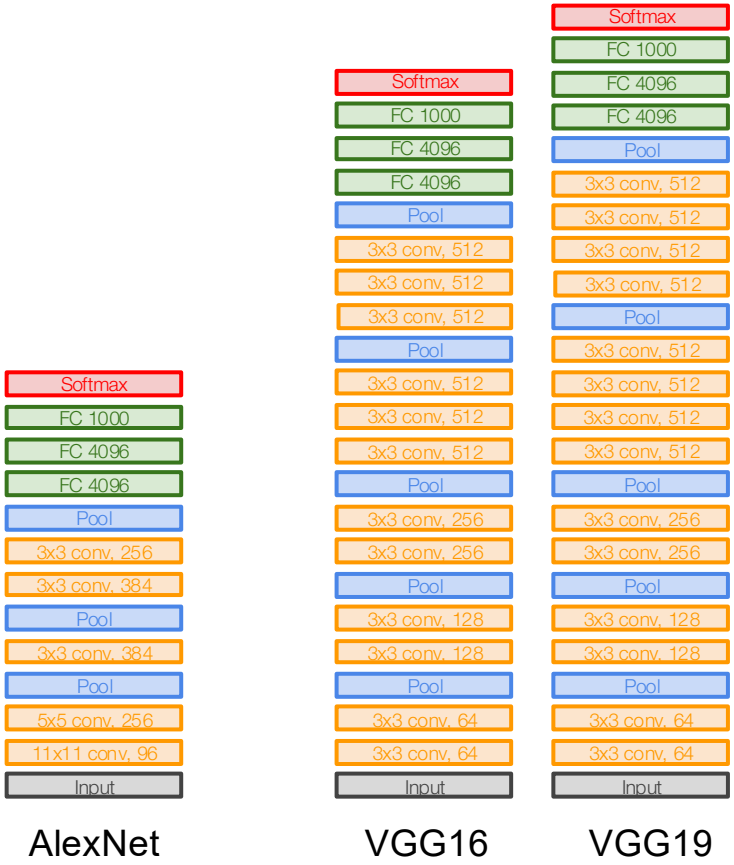
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



Case Study: ResNet

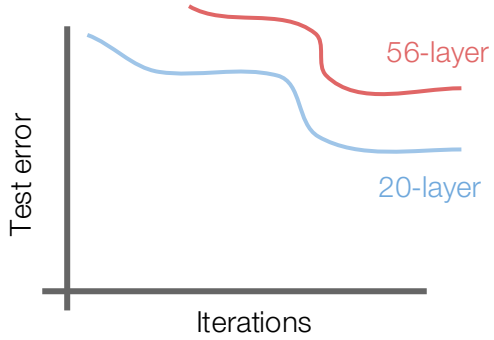
[He et al., 2015]

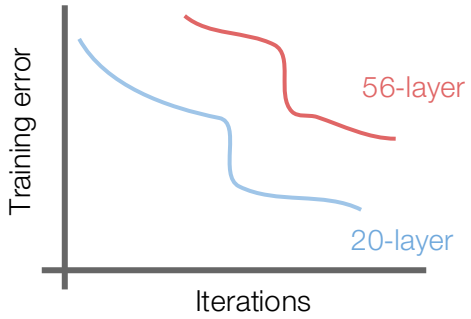
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

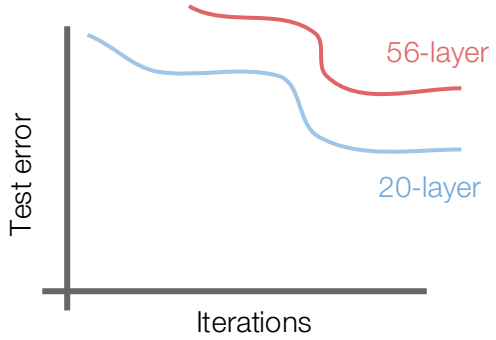


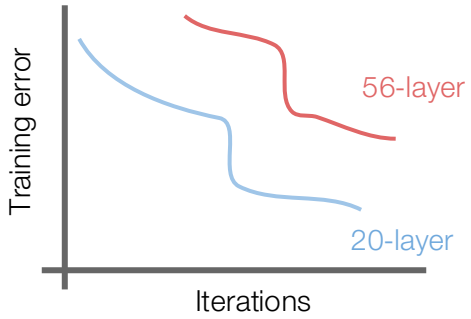


Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both test and training error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, **deeper models are harder to optimize**

Case Study: ResNet

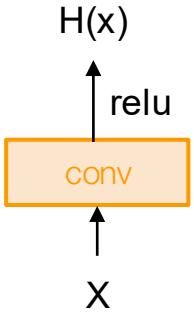
[He et al., 2015]

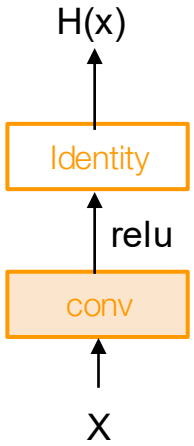
Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

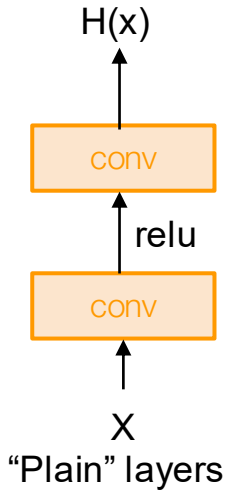




Case Study: ResNet

[He et al., 2015]

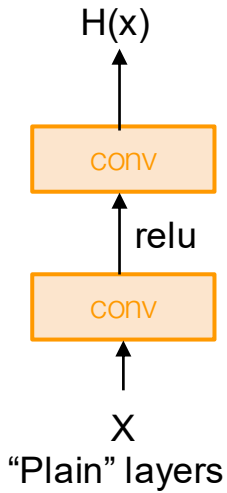
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

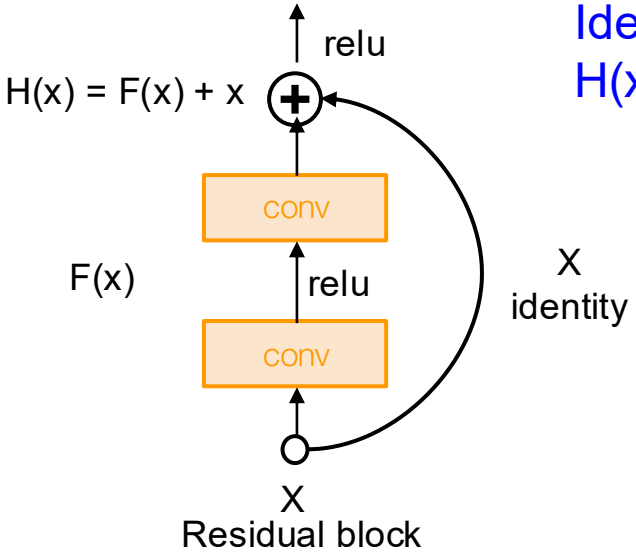


Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



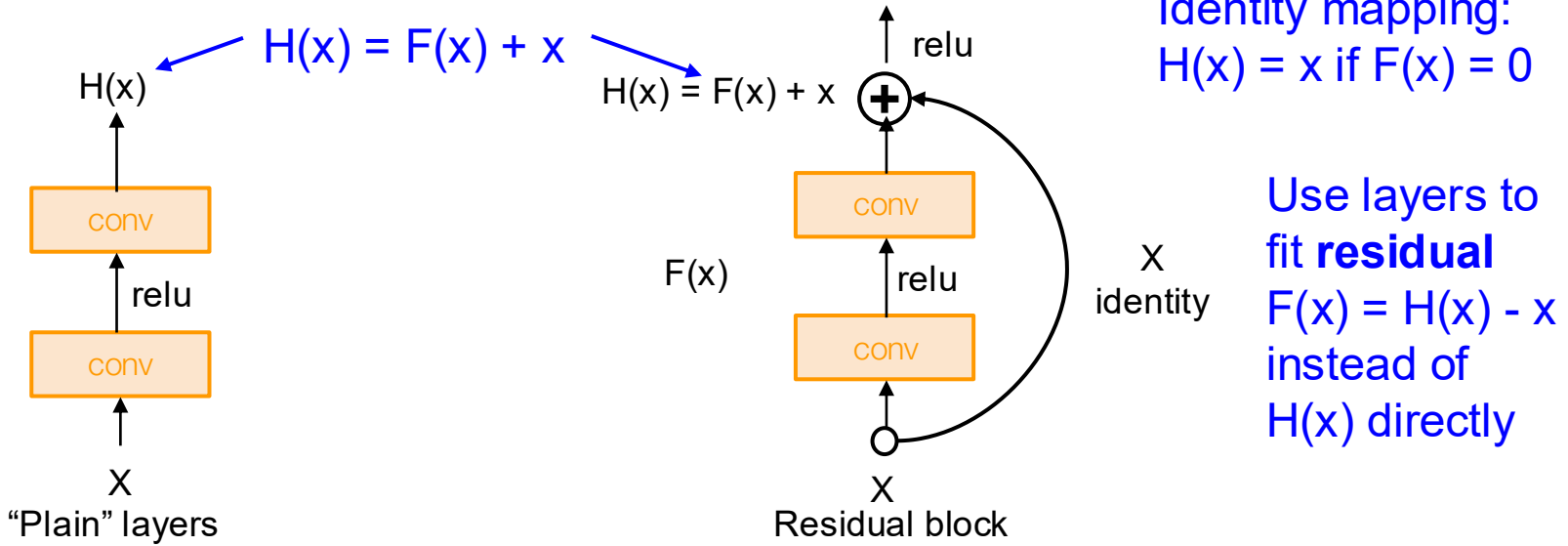


Identity mapping:
H(x) = x if F(x) = 0

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

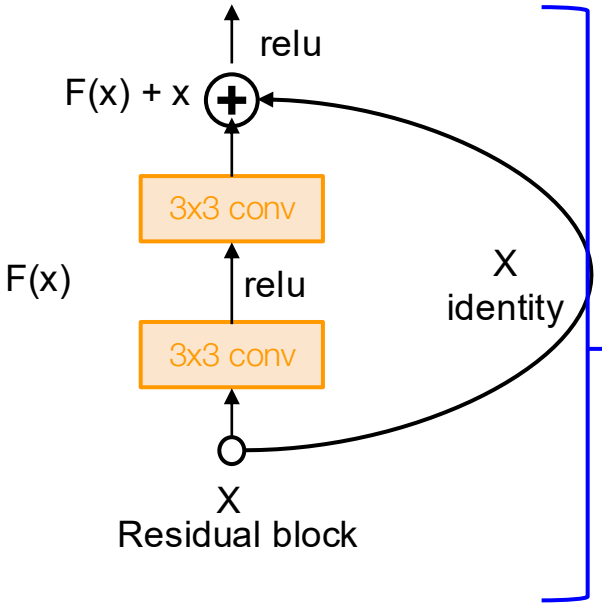
Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

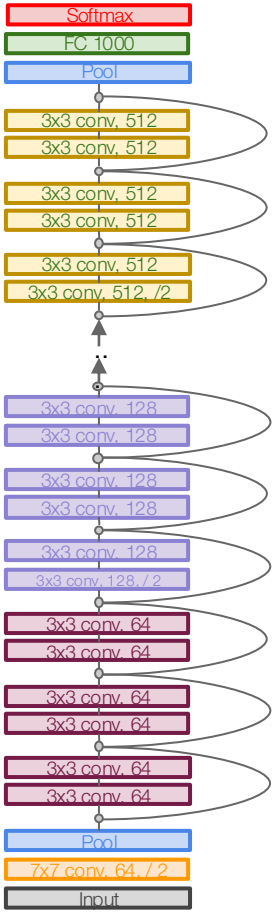
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers



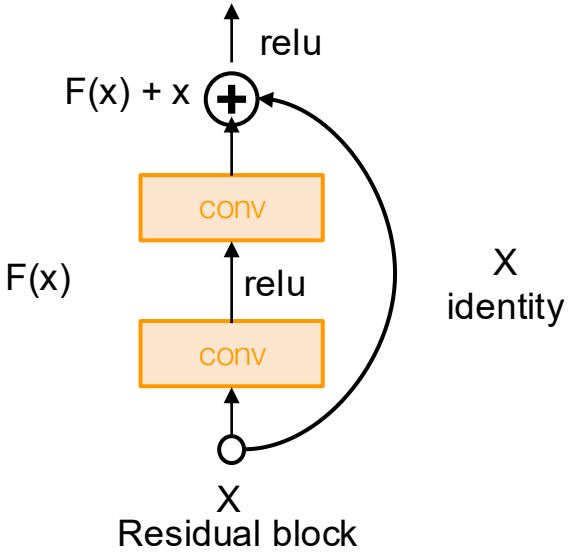


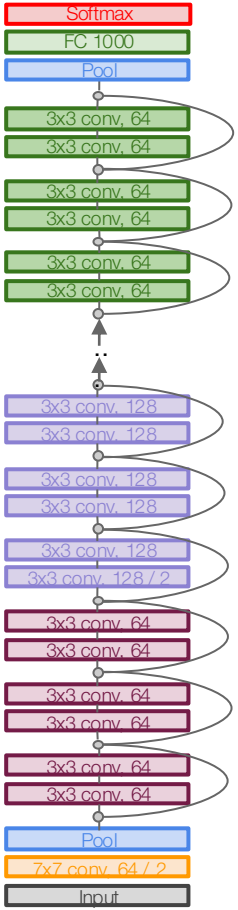
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





SOURCE AND TRANSITION

Training data and generalization

CNNs can memorize a training set. Data augmentation creates plausible variations so the model sees more examples of the same label. The transformations must preserve the task label.

Source: Stanford CS231n, Lecture 6, slides 70-76. https://cs231n.stanford.edu/slides/2026/lecture_6.pdf

Regularization: A common pattern

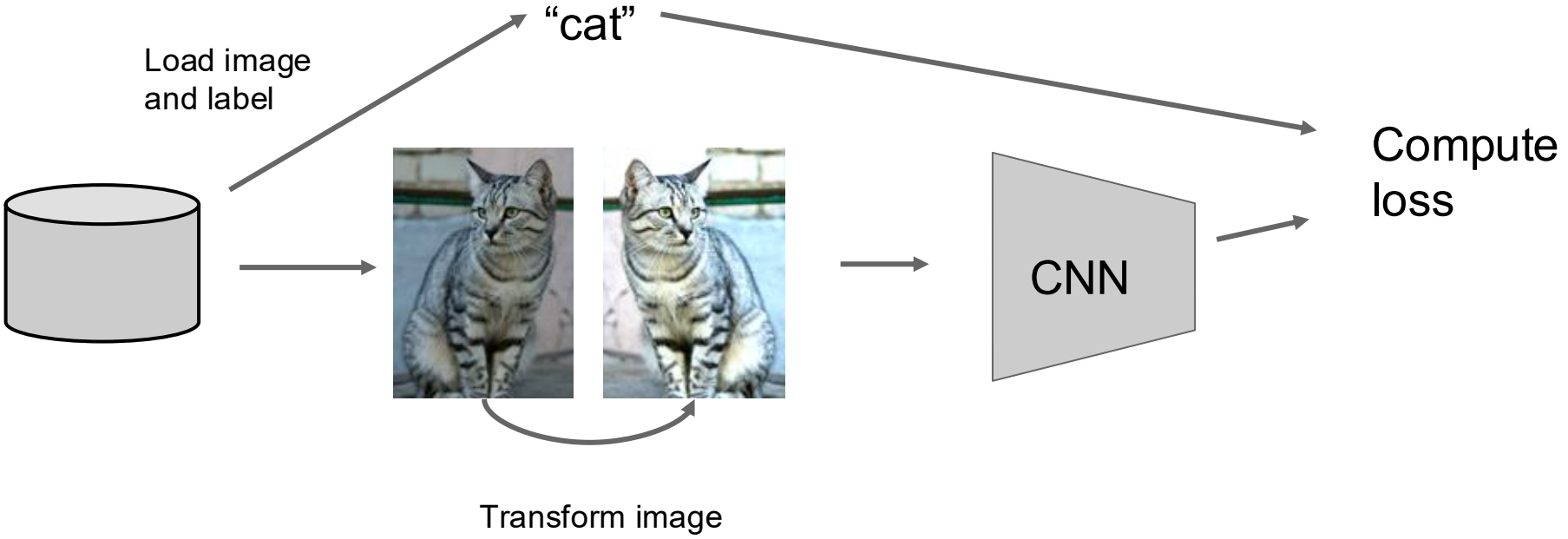
Training: Add some kind of randomness

$$y = f_W(x, z)$$

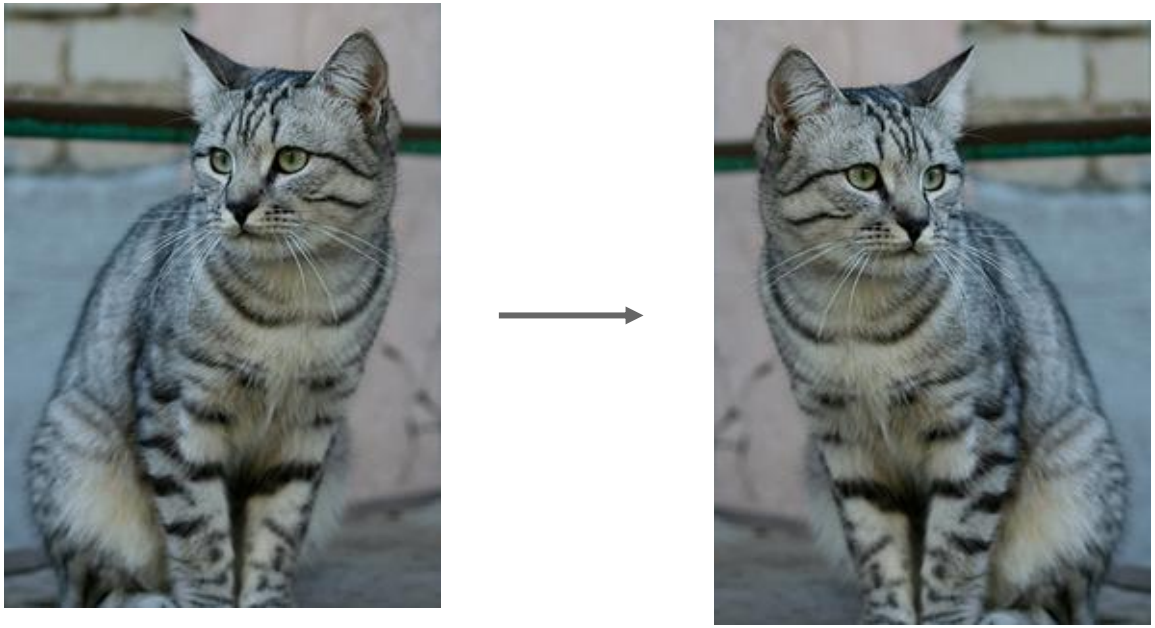
Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Regularization: Data Augmentation

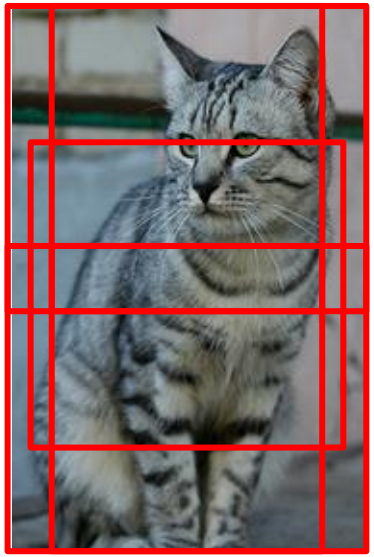


Data Augmentation
Horizontal Flips



Data Augmentation

Random crops and scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Test Time Augmentation: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation
Color Jitter

Simple: Randomize
contrast and brightness



Regularization: Cutout

Training: Set random image regions to zero

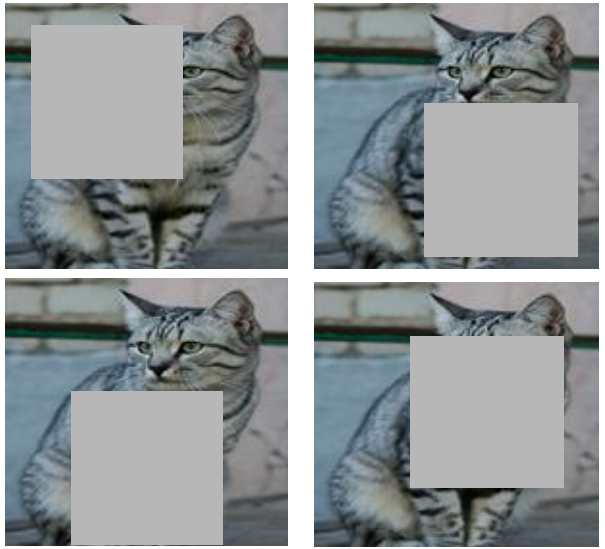
Testing: Use full image

Examples:

Dropout

Data Augmentation

Cutout / Random Crop



Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

DeVries and Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, arXiv 2017

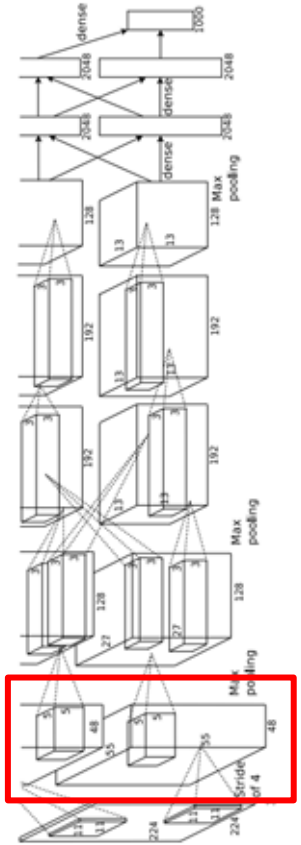
SOURCE AND TRANSITION

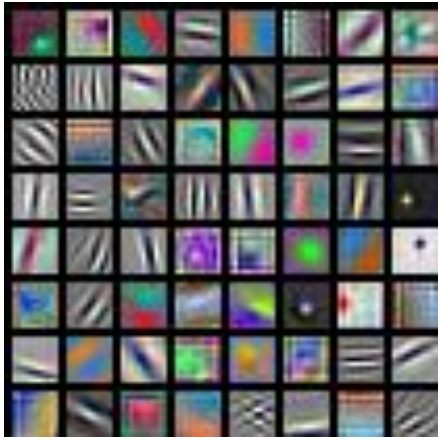
Transfer learning

A pretrained CNN can supply useful visual features when the target dataset is small. Replace or retrain the final classifier, then decide how much of the backbone to fine-tune.

What if you don’t have a lot of data? Can you still train CNNs?

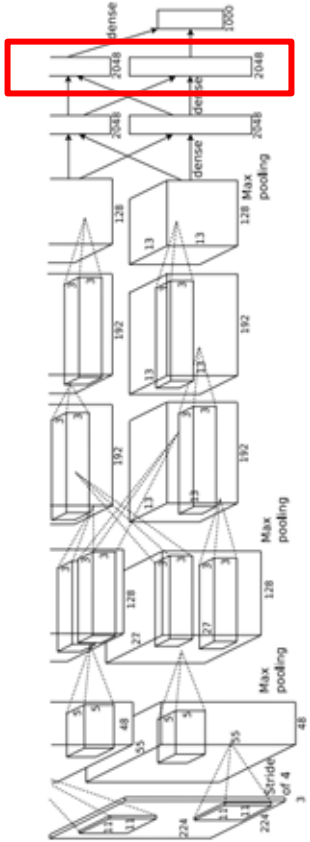
Transfer Learning with CNNs





AlexNet:
64 x 3 x 11 x 11

Transfer Learning with CNNs



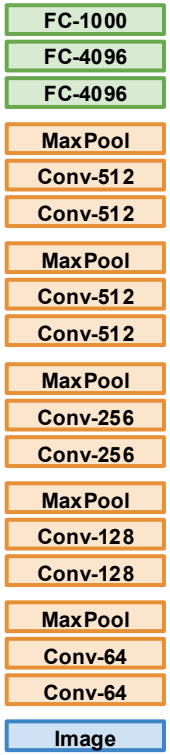
Test image L2 Nearest neighbors in feature space



Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

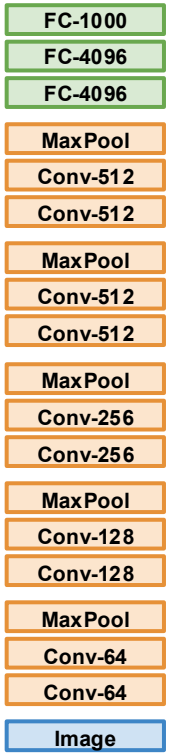
1. Train on Imagenet (or internet scale data)



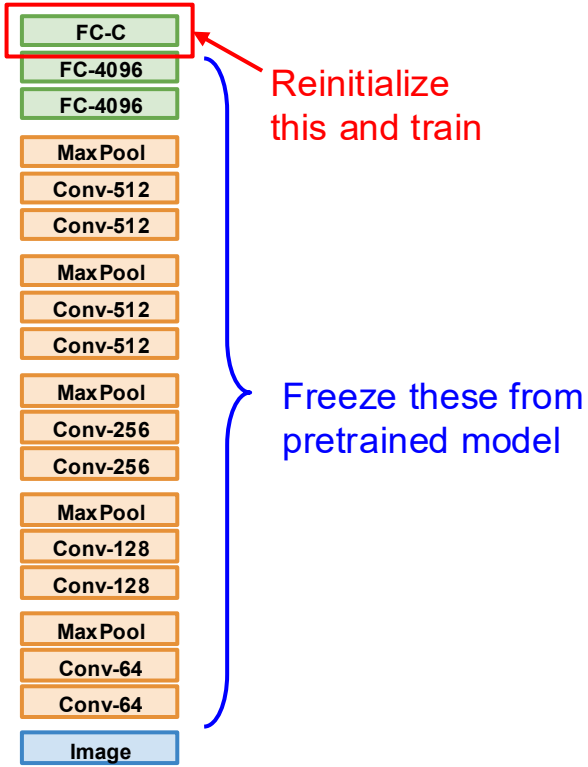
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



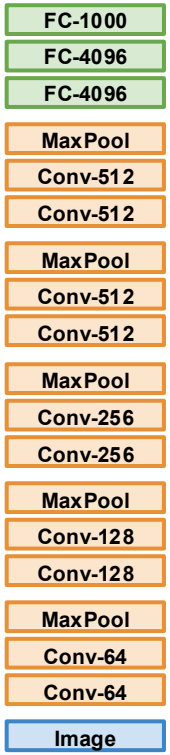
2. Small Dataset (C classes)



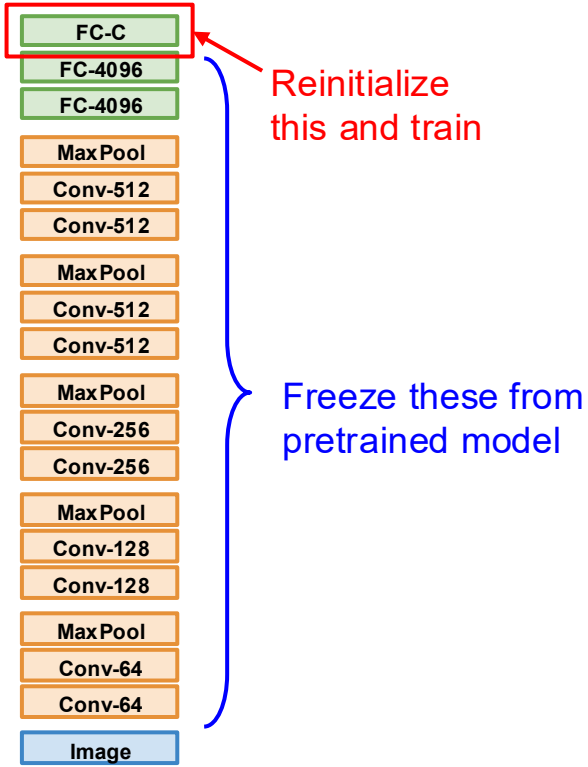
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

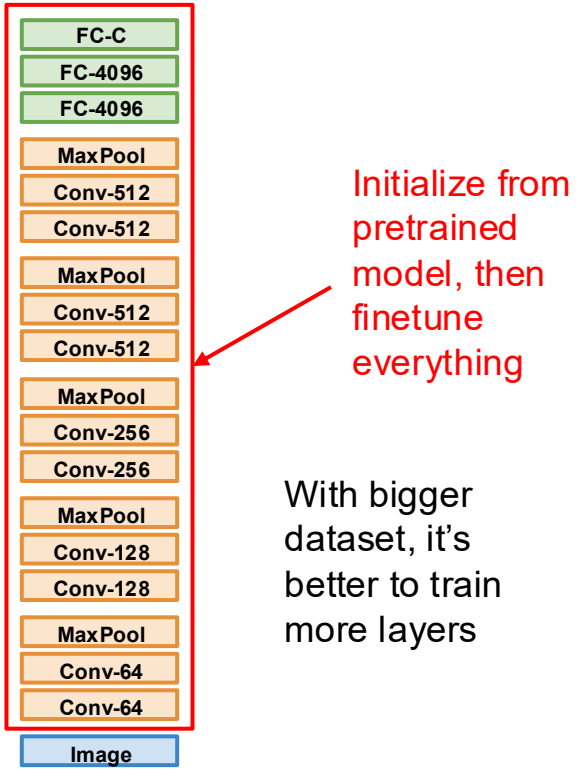
1. Train on Imagenet



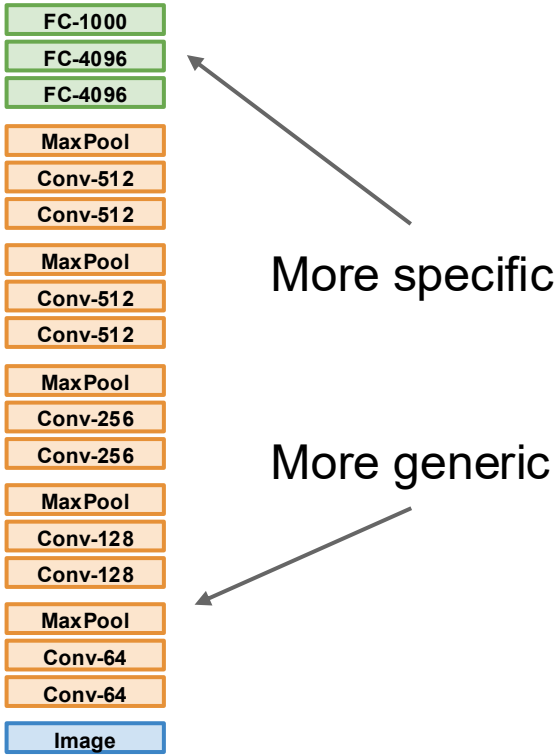
2. Small Dataset (C classes)



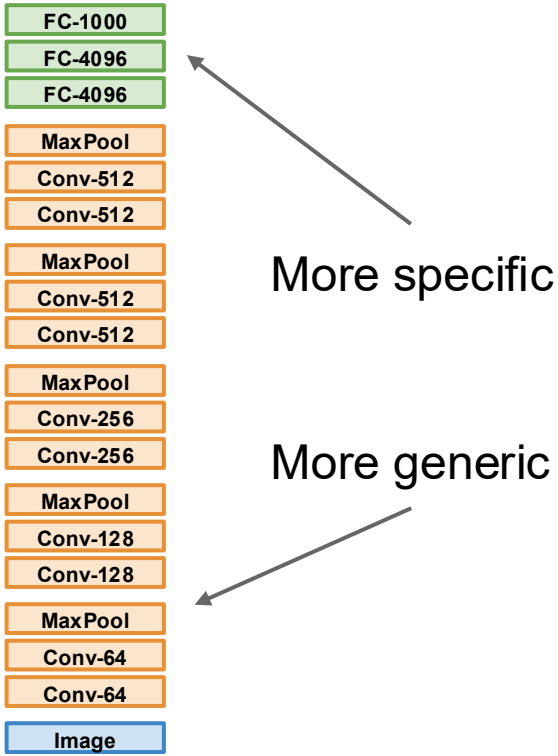
3. Bigger dataset



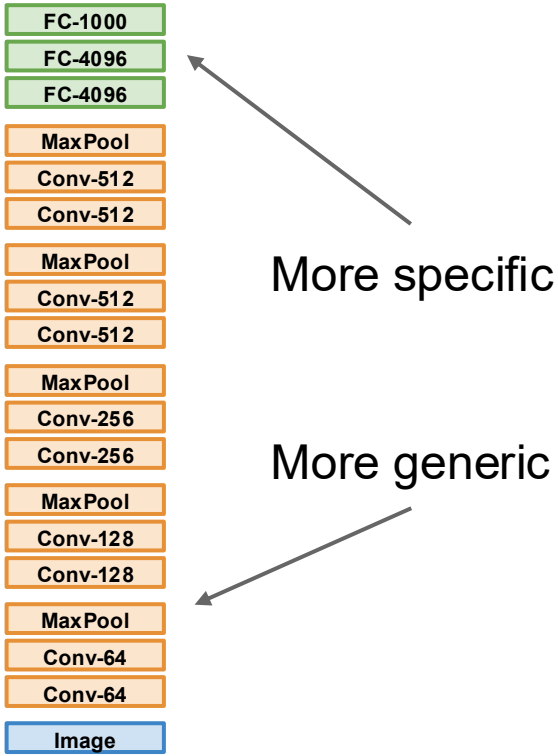
With bigger dataset, it's better to train more layers



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



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on final layer	?
quite a lot of data	Finetune all model layers	?



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on final layer	Try another pretrained model or collect more data ☹
quite a lot of data	Finetune all model layers	Either finetune all model layers or train from scratch!

SECTION

Sources and attribution

Original teaching slides reproduced from public course materials. Their original footers, author names, figure credits, and slide-level attributions remain on the reproduced pages. Newly written transition slides are labeled as such.

Stanford University, CS231n: Deep Learning for Computer Vision, Lectures 5 and 6. https://cs231n.stanford.edu/slides/2026/lecture_5.pdf and https://cs231n.stanford.edu/slides/2026/lecture_6.pdf. UC Berkeley Stat 153, Liberty Hamilton, Lecture 24: Convolutional Neural Networks Part

SECTION

CNNs beyond images: waveforms

Images are two-dimensional grids. Audio waveforms are one-dimensional sequences. In both cases, a local filter can detect patterns and share its weights across positions.

Transition written for this compilation. Berkeley Stat 153 Lecture 24, slides 28-34.

SOURCE AND TRANSITION

Sound classification with 1D convolution

UC Berkeley Stat 153 introduces sound classification directly from waveforms. The following slides show how 1D convolution, nonlinearities, pooling, and stacked filters parallel the 2D image case.

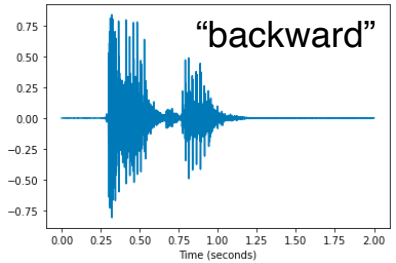
Source: Liberty Hamilton, UC Berkeley Stat 153, Lecture 24, slides 28-34 (Spring 2026). https://stat153.berkeley.edu/spring-2026/cnns-1/

Sound classification

- **Problem:** we want to classify *sounds*, represented as waveforms

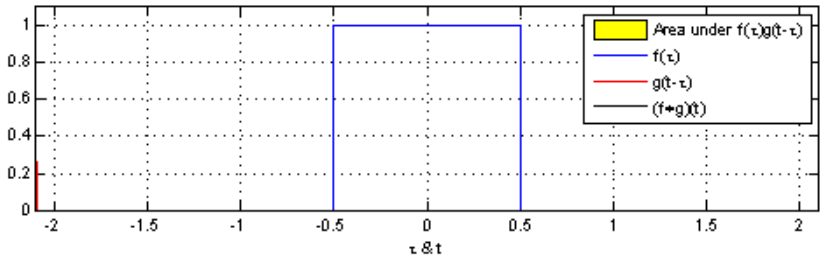
  * Each sound comprises (up to) 16000 samples

  * Can we apply the same tricks as for images?



Sound classification

- Solution: **1-D convolution**!



from https://en.wikipedia.org/wiki/Convolution

Sound classification

- Solution: **1-D convolution**!
  - Assuming: $f \in \mathcal{R}^T, \{h, g\} \in \mathcal{R}^n$ *(f is length T vector, h and g are length n vectors)*
  - Definition: $(f \star g)[t] = \sum_{\tau=-\infty}^{\infty} f[t]g[t-\tau]$ *(defining g[.]=0 for indices greater than n or less than 1)*
  - Or: $(f \star h)[t] = \sum_{\tau=1}^{n} f[t-\tau]h[\tau] = f[t-n:t] \cdot h$

Sound classification

- We can stack multiple convolutional filters (aka kernels) into a **1-D convolutional layer**:
  - Let $g_1 \cdots g_k$ be a set of length-n vectors, and x be a length-T vector (input)
  - Define

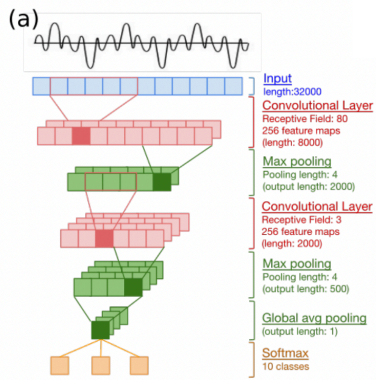
$$g(x) = \begin{bmatrix} g_1 \star x \\ g_2 \star x \\ \cdots \\ g_k \star x \end{bmatrix}$$

Sound classification

- Then we can:
  - apply a nonlinearity (e.g. ReLU) to the results
  - downsample the results using **max-pooling** or **average-pooling**
  - (maybe pass the results into a fully-connected layer?)

Sound classification

- Example: **M3** (Dai et al., 2016)



Sound classification

- 1-D convolutional networks!
  - (they’re pretty much the same as 2-D convolutional networks, so far)
  - You’ll see a demo in lab!