Introduction to Deep Learning and CNNs

Fiseha B. Tesema, PhD

Recap

- Bag of Visual Words
- Vola-Jones Object Detection

Outline

- Introduction to deep learning
- Our Neural Network
- Fully-Connected Network
- Convolutional Neural Networks (CNNs)
  - Activation functions
  - Fully-Connected layer to at a final stage: making a decision
  - Convolution layers
  - Pooling layers
  - Normalization

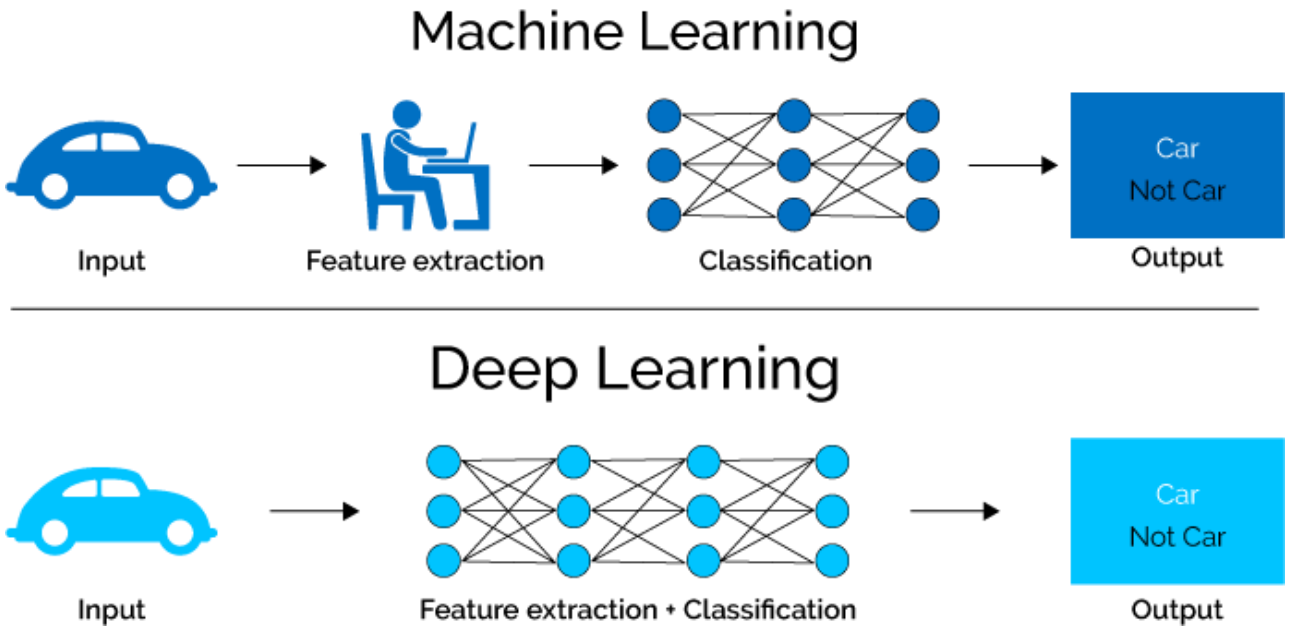
Slide Credit

- Majority of the slide is complied from Deep Learning for computer vision: University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Introduction to deep learning

ML vs. Deep Learning

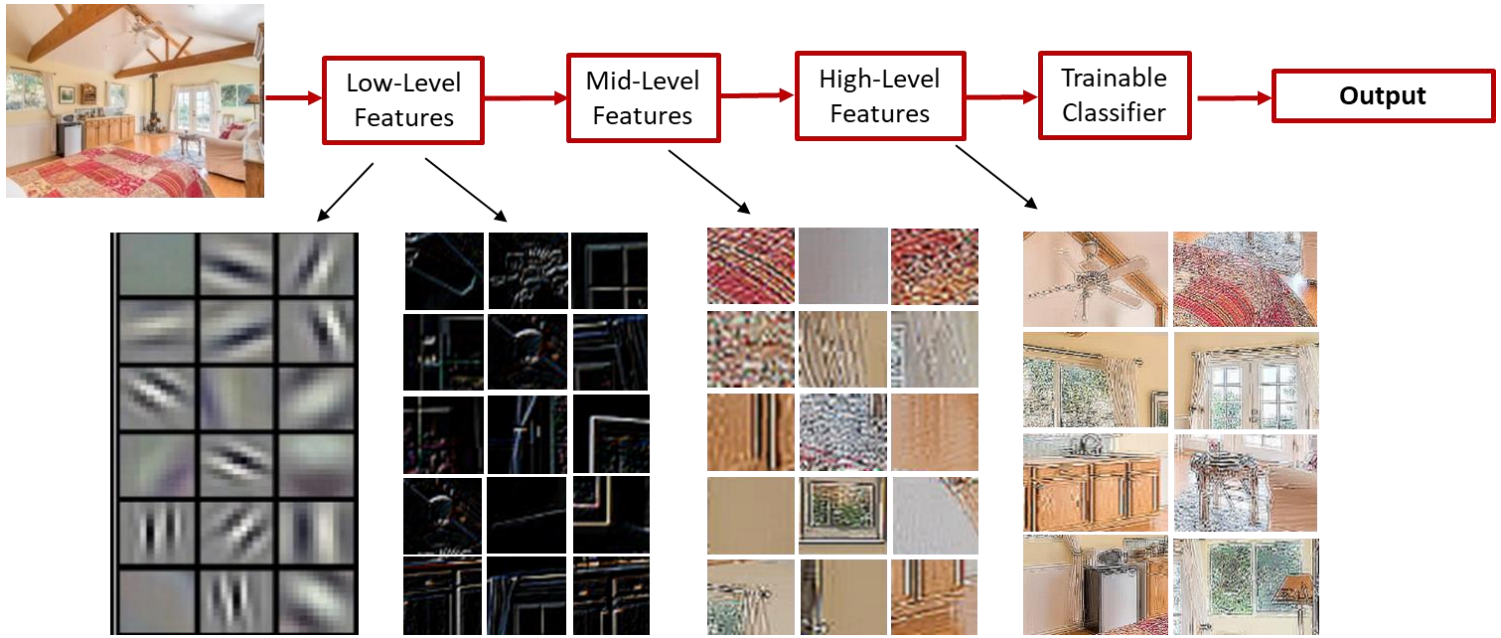
- Deep learning (DL) is a machine learning subfield that uses multiple layers for learning data representations
  - DL is exceptionally effective at learning patterns



Picture from: https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

ML vs. Deep Learning

- Deep learning is a popular AI technique
- DL applies a multi-layer process for learning rich hierarchical features (i.e., data representations)
  - Input image pixels → Edges → Textures → Parts → Objects



Slide credit: Param Vir Singh – Deep Learning

Why is DL Useful?

- DL provides a flexible, learnable framework for representing visual, text, linguistic information
  - Can learn in supervised and unsupervised manner
- DL represents an effective end-to-end learning system

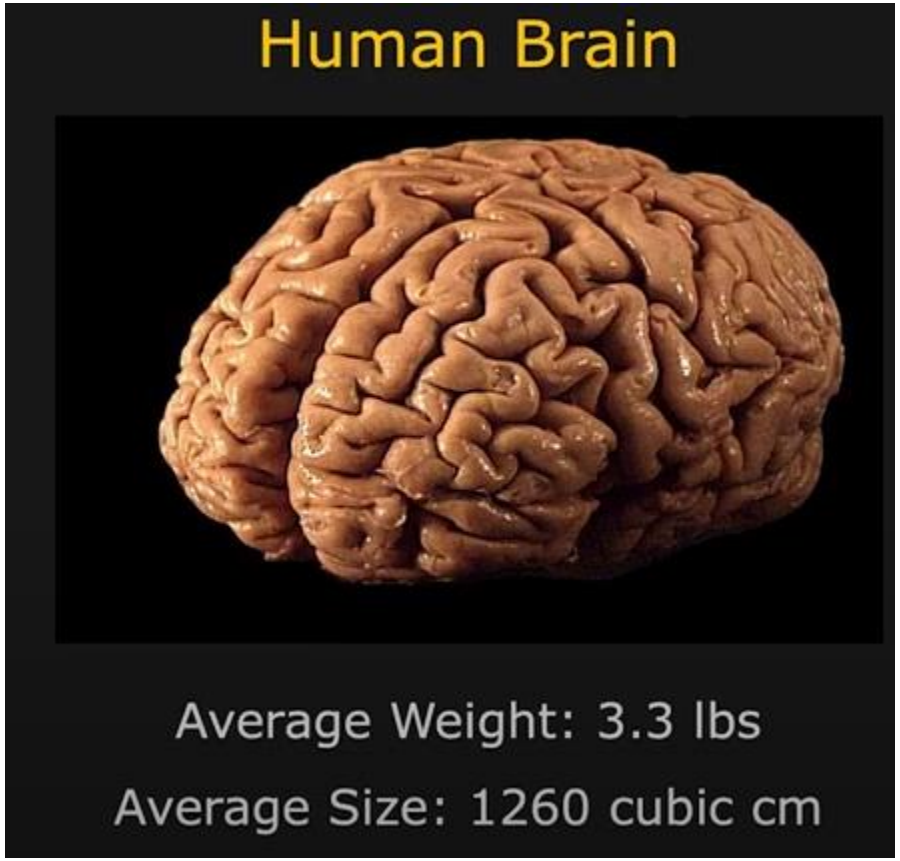
- Requires large amounts of training data
- Since about 2010, DL has outperformed other ML techniques
  - First in vision and speech, then NLP, and other applications

What led to the development of DL

- Several concurrent factors including:
  - GPU development
  - Algorithm improvements (e.g. ReLU function….)
  - Availability of large training image sets

Neural networks

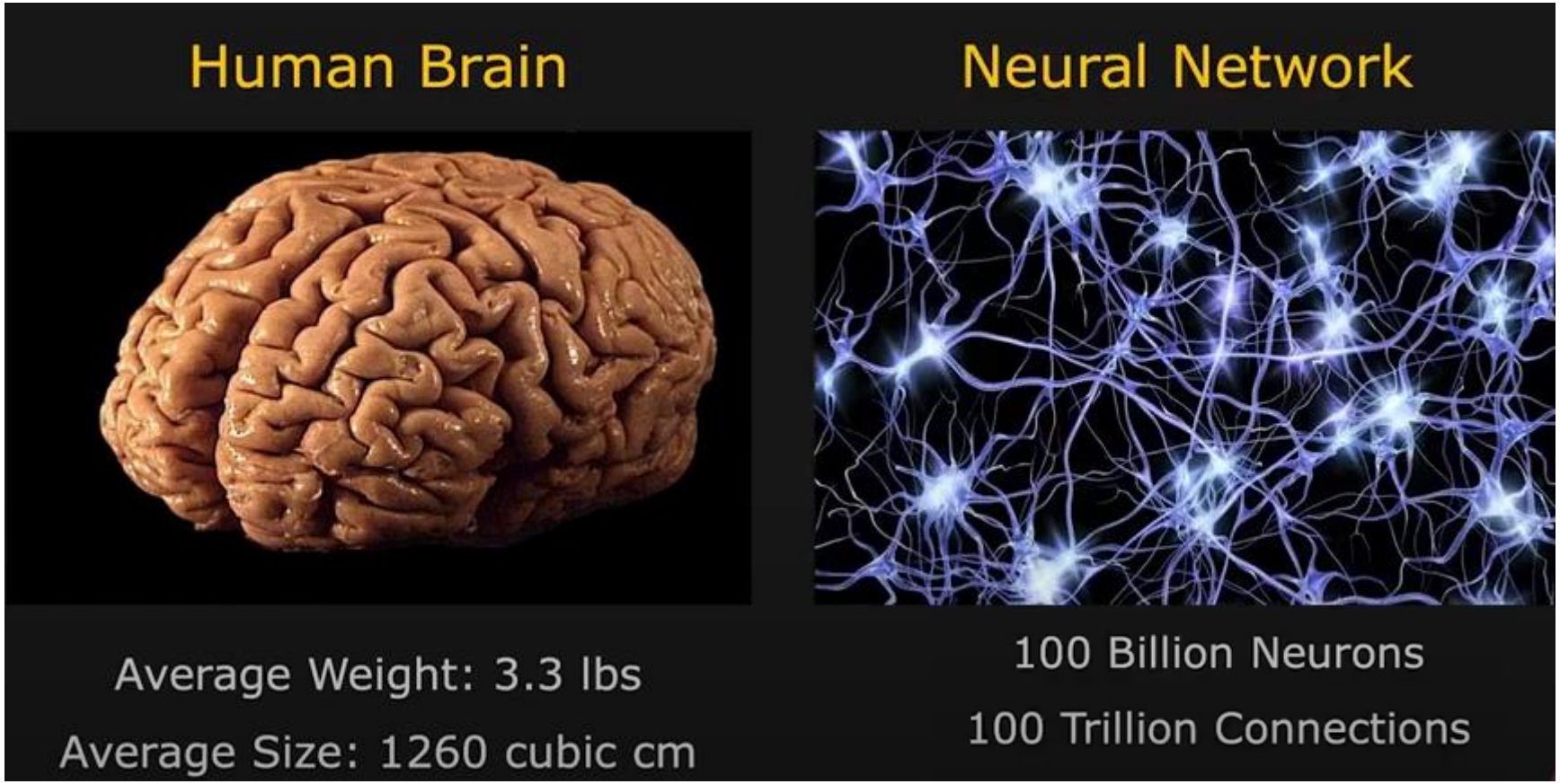
Our Neural Network



- It 2% weight of a person
- But, it consumes 20% of the energy
  - Why?
    - Lot of visual processing &
    - Other type of signal processing happening in the brain all the time.

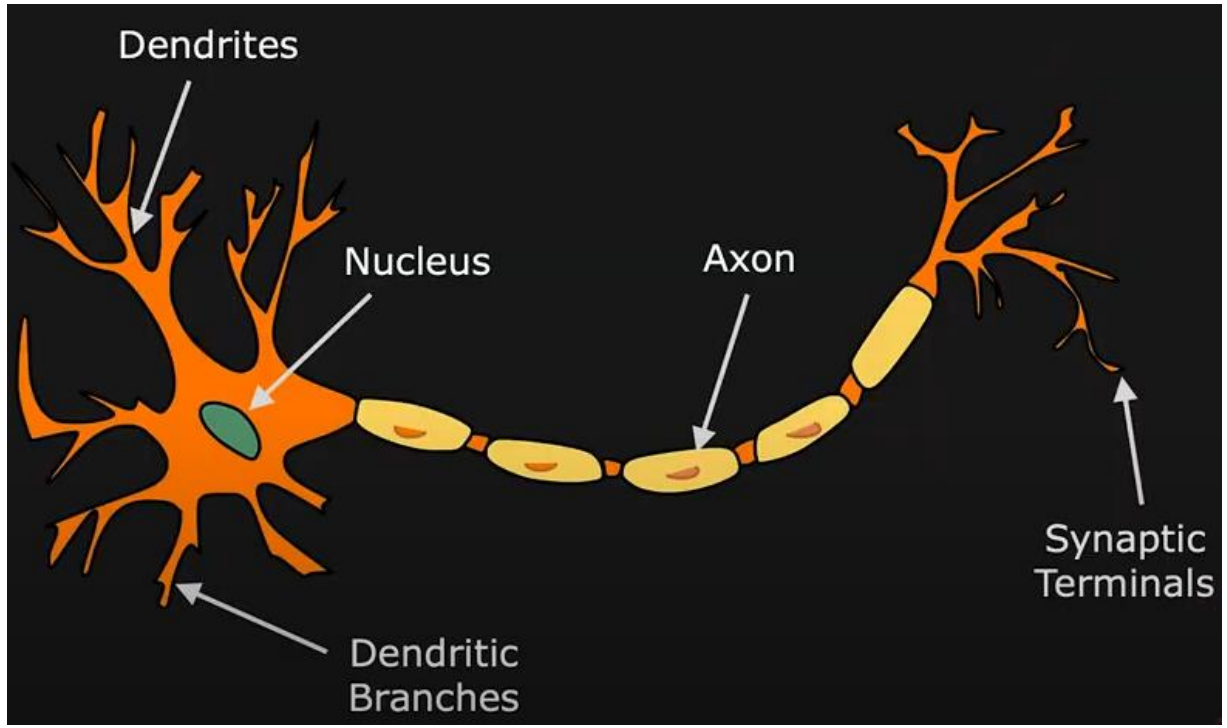
Our Neural Network

- Brain, a network of neurons



A Neuron

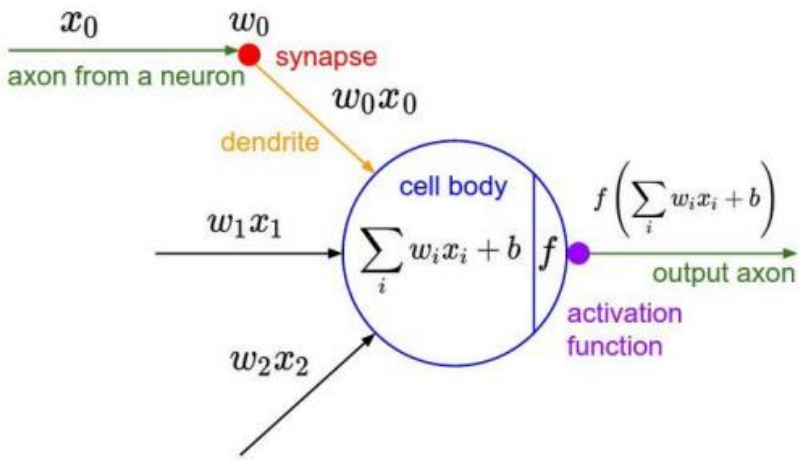
- A basic building bock of human brain.



- Dendrites: receive signal from other neurons.
- Nucleus: process input sign in a very simple way.
- The output travels though the Axon.
- Synaptic terminals: a point where one neuron make a connection with other neuron.

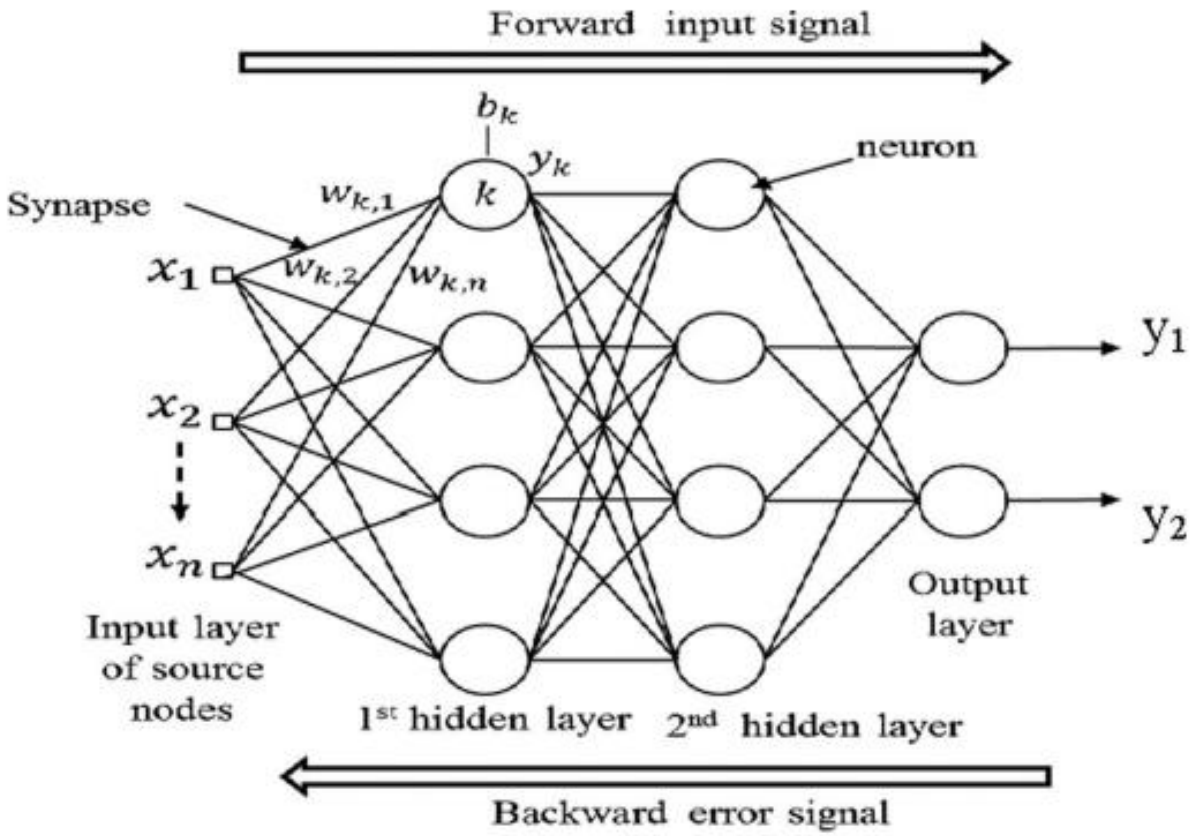
Artificial Neural Network (ANN)

- An **Artificial Neural Network (ANN)** is a computational model inspired by the human brain's neural networks. It consists of interconnected nodes (neurons) organized in layers, designed to recognize patterns, make decisions, and perform predictions.



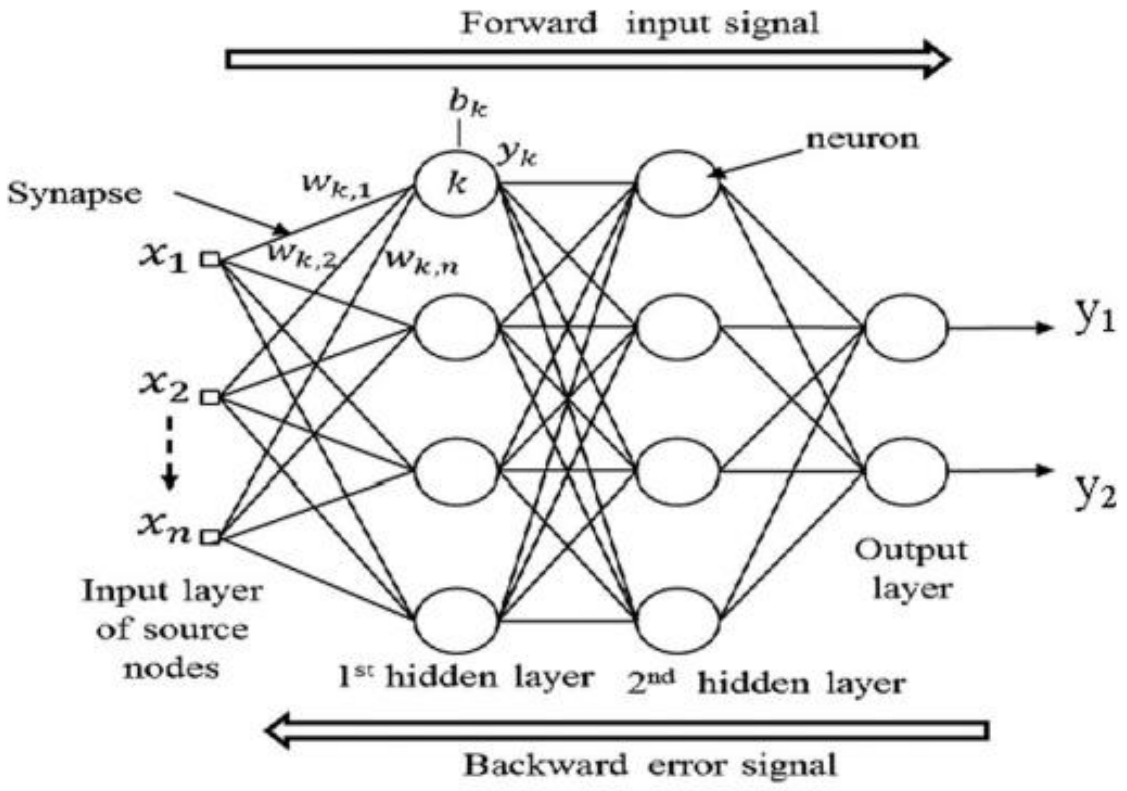
- Modeling artificial networks tend to use 100K-19b connections.

Basic Structure of Neural Nets



- An ANN has three main types of layers:
  - **Input Layer** – Receives raw data (e.g., pixels in an image, features in a dataset).
  - **Hidden Layers** – Process data through weighted connections and activation functions.
  - **Output Layer** – Produces the final prediction (e.g., class label, regression value

Training an ANN



- Steps in Training:
  - **Forward Propagation** – Input passes through the network to compute output.
  - **Loss Calculation** – Measures error (e.g., Cross-Entropy, MSE).
  - **Backpropagation** – Adjusts weights using gradient descent.
  - **Optimization** – Updates weights via optimizers (SGD, Adam, RMSprop).

Types of ANN

- **Feedforward Neural Network (FNN):** Simplest ANN, data flows in one direction (input → output).
  - Application: Basic classification, regression.
  - Types of FNN
    - Feedforward Networks
    - **Convolutional Neural Network (CNN):** Uses convolutional layers for spatial data (images).
      - Application: Image recognition, object detection.
- Recurrent Neural Network (RNN): Processes sequential data with memory (loops).
  - Application: Time series, NLP (text, speech).

Cont. …

- Long Short-Term Memory (LSTM): Improved RNN for long-term dependencies.
  - Application: Speech recognition, machine translation.
- **Generative Adversarial Network (GAN):** Two networks (generator & discriminator) compete.
  - Application: Image generation, deepfakes.

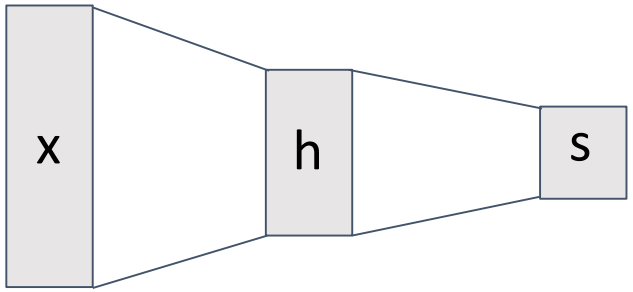
Fully Connected Networks

Fully Connected Networks

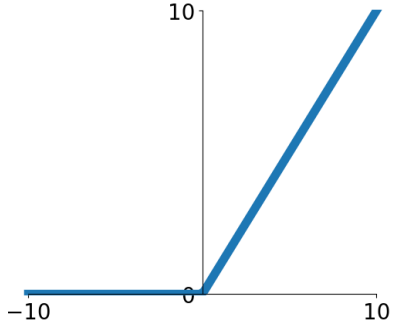
- Fully Connected Networks (FCNs):Networks with only dense (fully connected) layers (every neuron connects to all neurons in the next layer).
- Subtypes:
  - Single-Layer Perceptron (SLP): No hidden layers: Input → Output (e.g., linear/logistic regression).
    - Not an MLP (since MLPs require hidden layers).
  - Multilayer Perceptron (MLP):
    - ≥ 1 hidden layer: Input → FC → Activation → ... → Output.
    - The most common type of FCN.

Fully Connected Network Components

Fully-Connected Layers



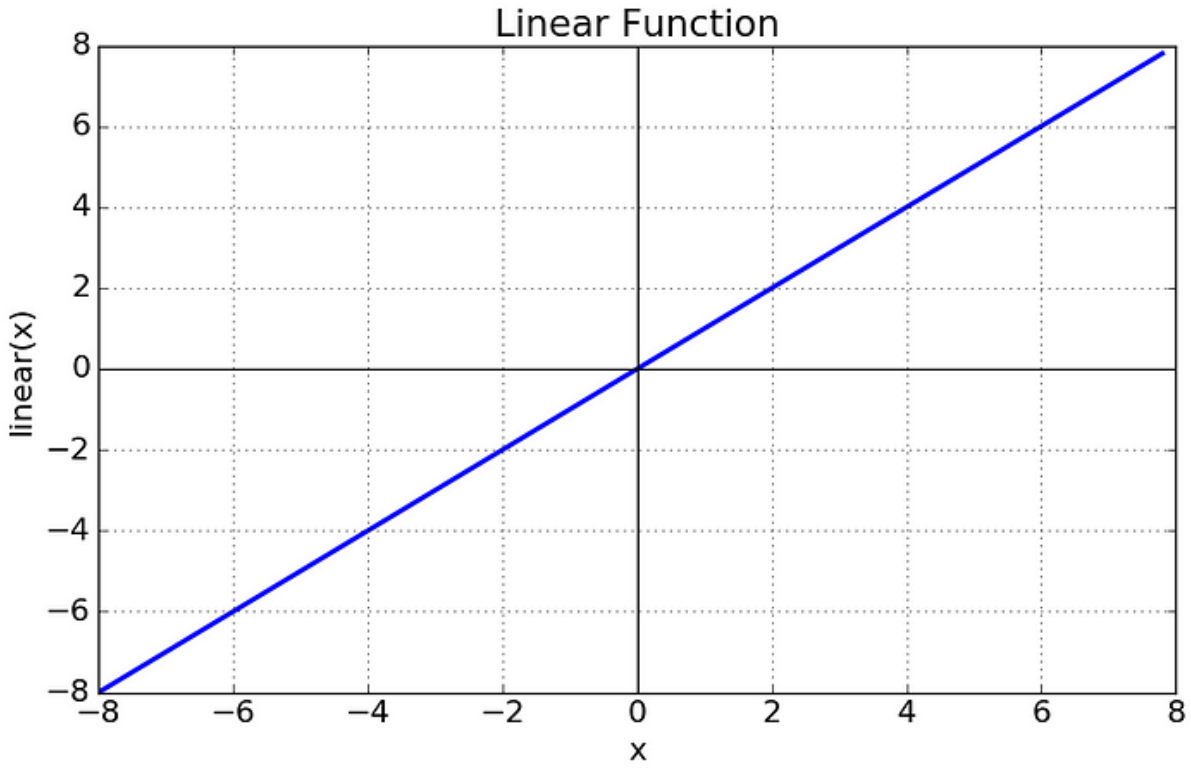
Activation Function



Activation function

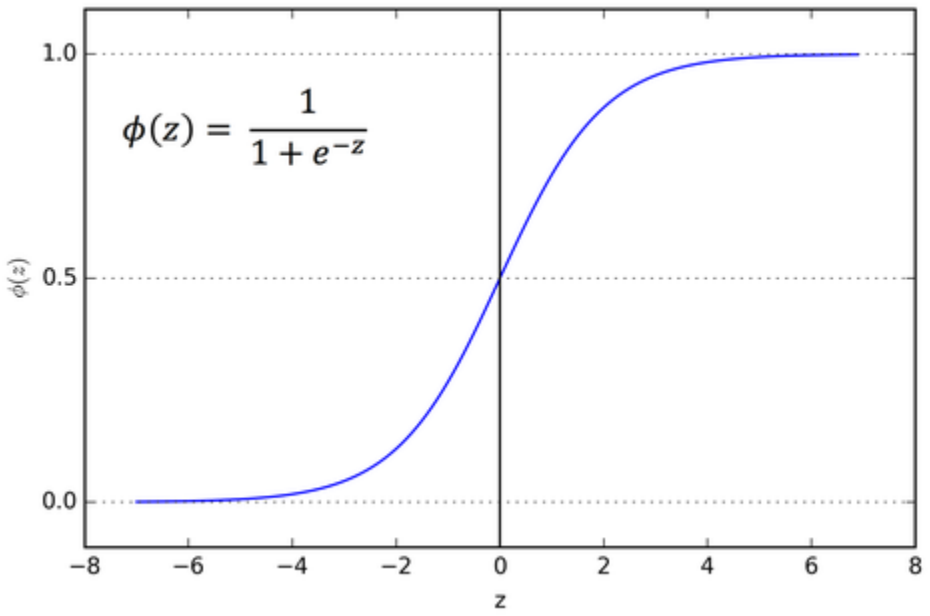
- An activation function determines the output of a neuron given its input.
- It introduces non-linearity into the network, allowing it to learn complex patterns (without it, a neural network would just be a linear model, no matter how many layers it has).
- It maps the resulting values in between **0 to 1 or -1 to 1 etc**. (depending upon the function).
- The Activation Functions can be basically divided into 2 types:
  - Linear Activation Function
  - Non-linear Activation Functions

Linear or Identity Activation Function



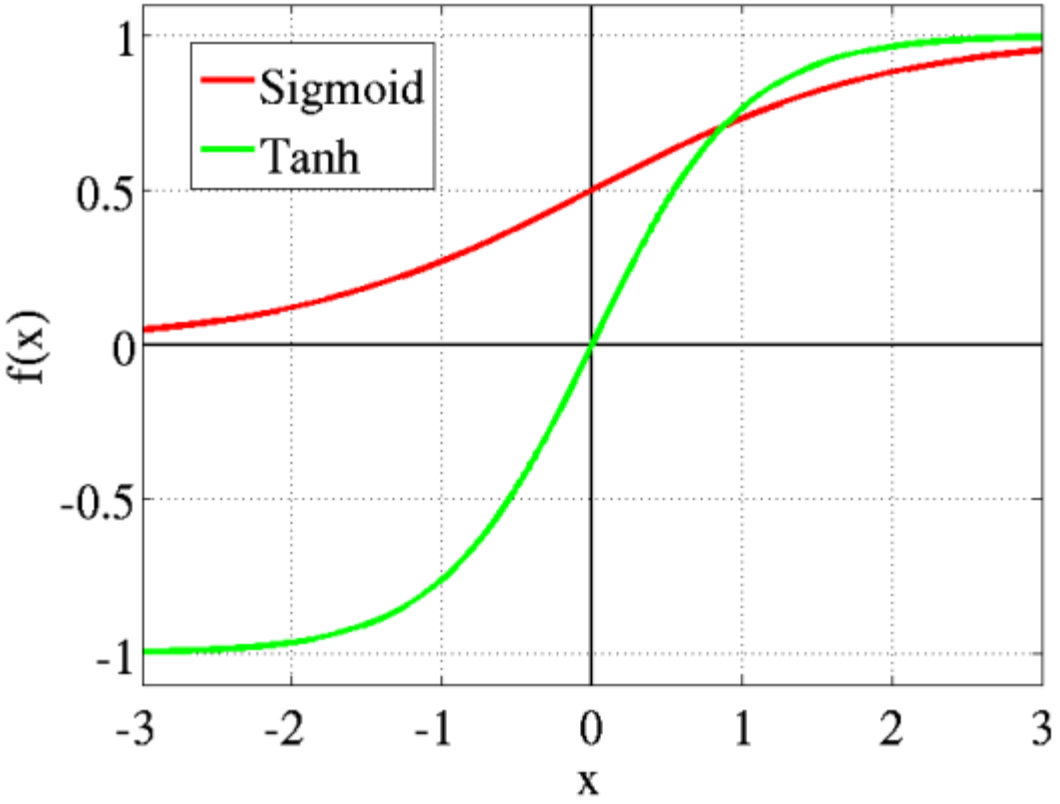
- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.
- Equation : f(x) = x
- Range : (-infinity to infinity)

Non-linear Activation Function



- **Sigmoid or Logistic Activation Function**
  - The main reason why we use sigmoid function is because it exists between (0 to 1).
  - It's especially used for models where we have to predict the probability as an output.
    - Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

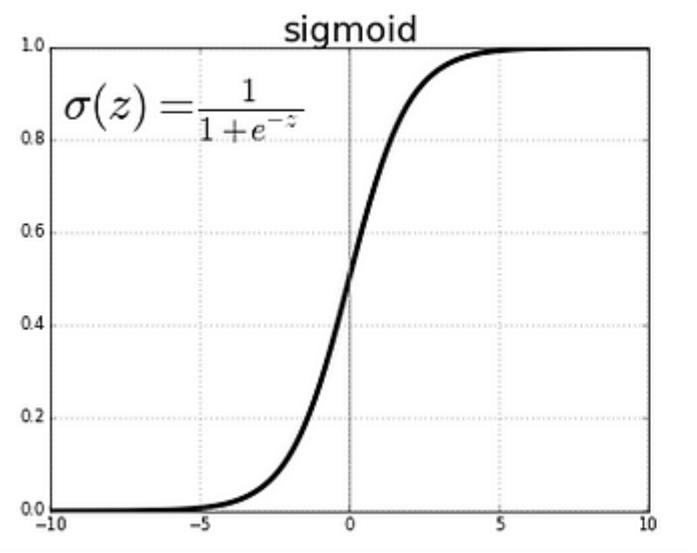
Non-linear Activation Function

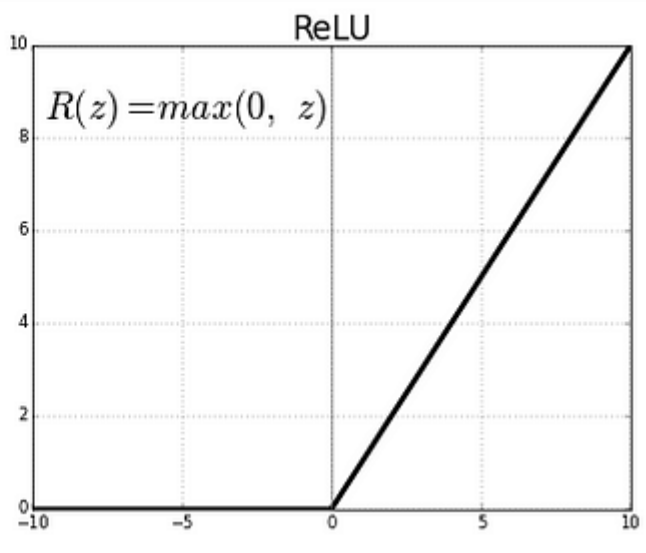


- Tanh or hyperbolic tangent Activation Function
  - tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).
- Both tanh and logistic sigmoid activation functions are used in **feed-forward nets**.

Non-linear Activation Function

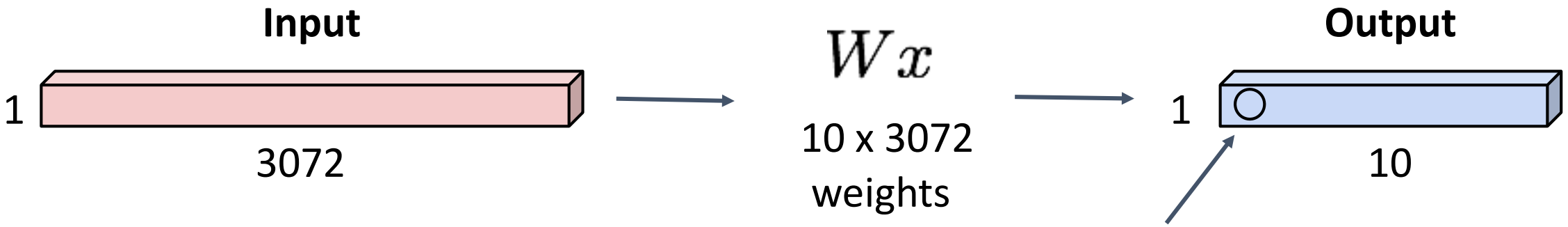
- The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.
- Range: [0 to infinity)



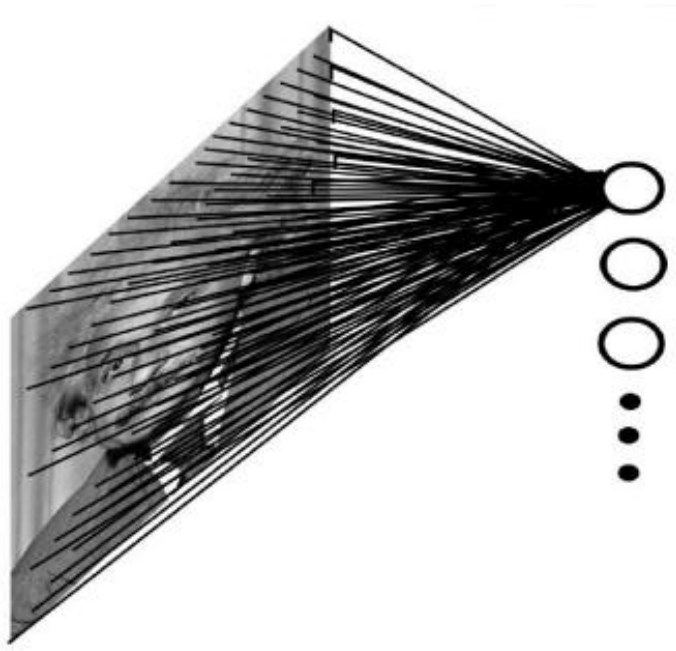


Fully-Connected Layer

- Don’t support image
- 32x32x3 image -> stretch to 3072 x 1



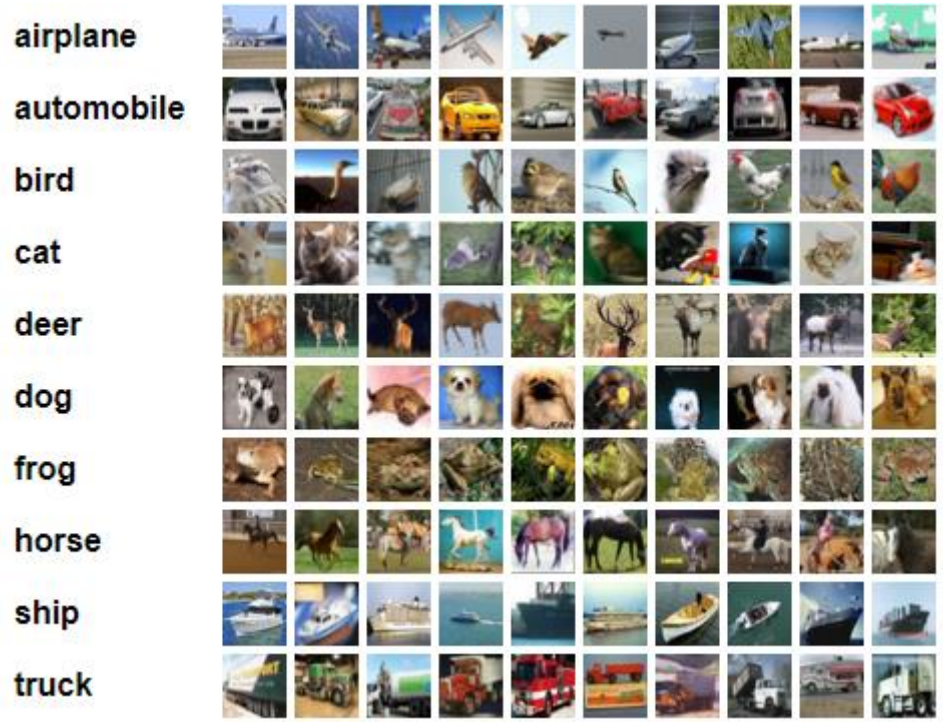
FCN are inefficient for CV tasks



- **Ignores Spatial Locality**
  - Images have local patterns (edges, textures, objects) that are best detected by examining small regions.
  - FCNs treat each pixel as independent, losing spatial structure.
- **Parameter Explosion**
  - For a 256x256 RGB image, an FCN input layer would need: 256×256×3=196,608 weights per neuron in the next layer!
  - A single hidden layer with 1K neurons → 196 million parameters (extremely inefficient).
- **Most Weights Are Useless**
  - In FCNs, >99% of weights connect distant pixels with no meaningful relationship (effectively zero).
- Would require an impractically large training set to learn this many weights

The CIFAR-10 dataset

- The CIFAR-10 dataset consists of 60000 **32x32** colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

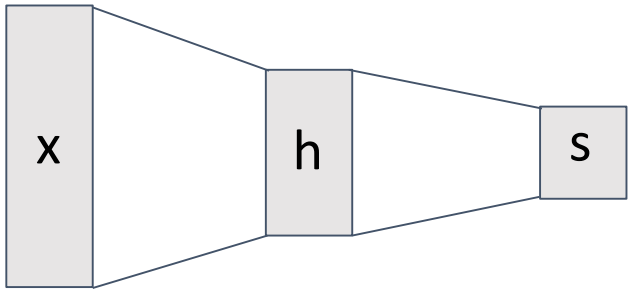


https://www.cs.toronto.edu/~kriz/cifar.html

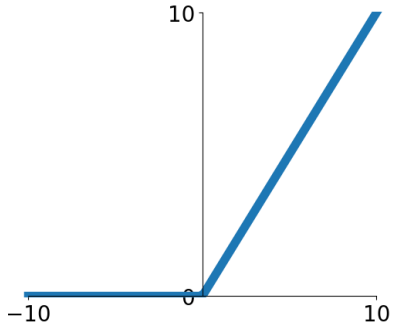
Convolutional Neural Network

Components of a Convolutional Network

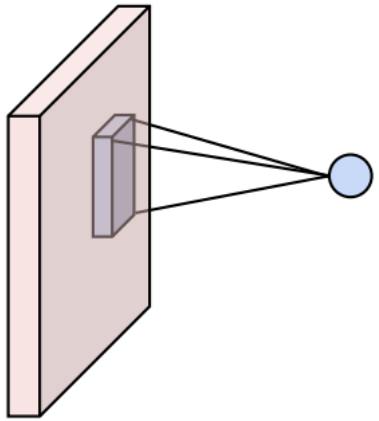
Fully-Connected Layers



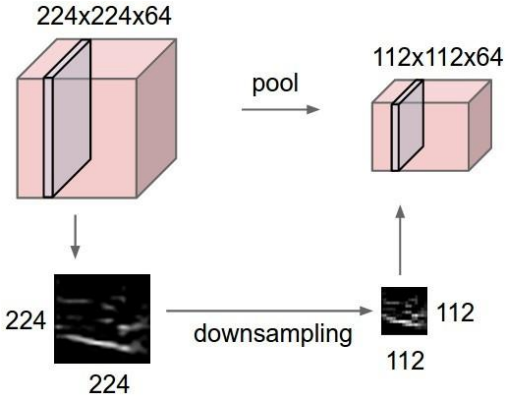
Activation Function



Convolution Layers



Pooling Layers



Normalization

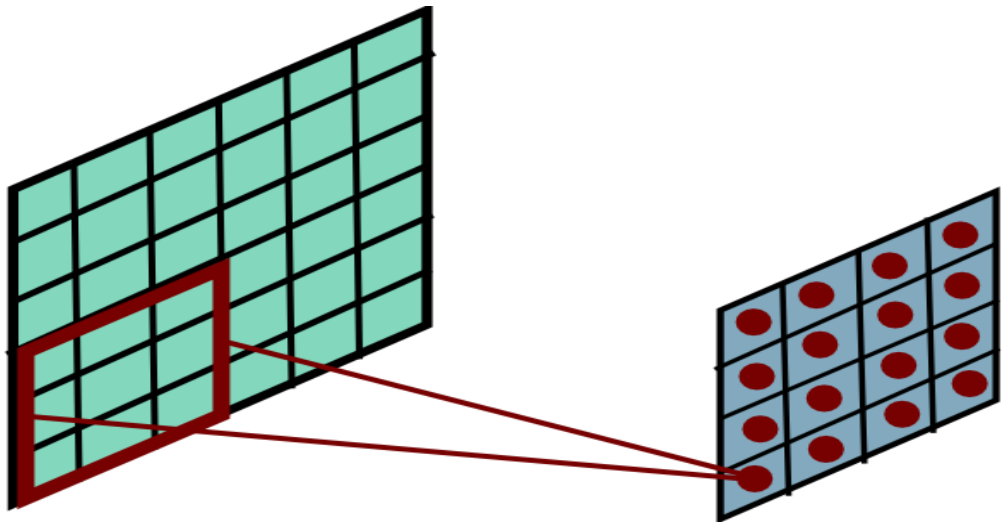
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Convolutional Neural Network

- When we are talking about deep learning for computer vision, we normally mean CNNs.

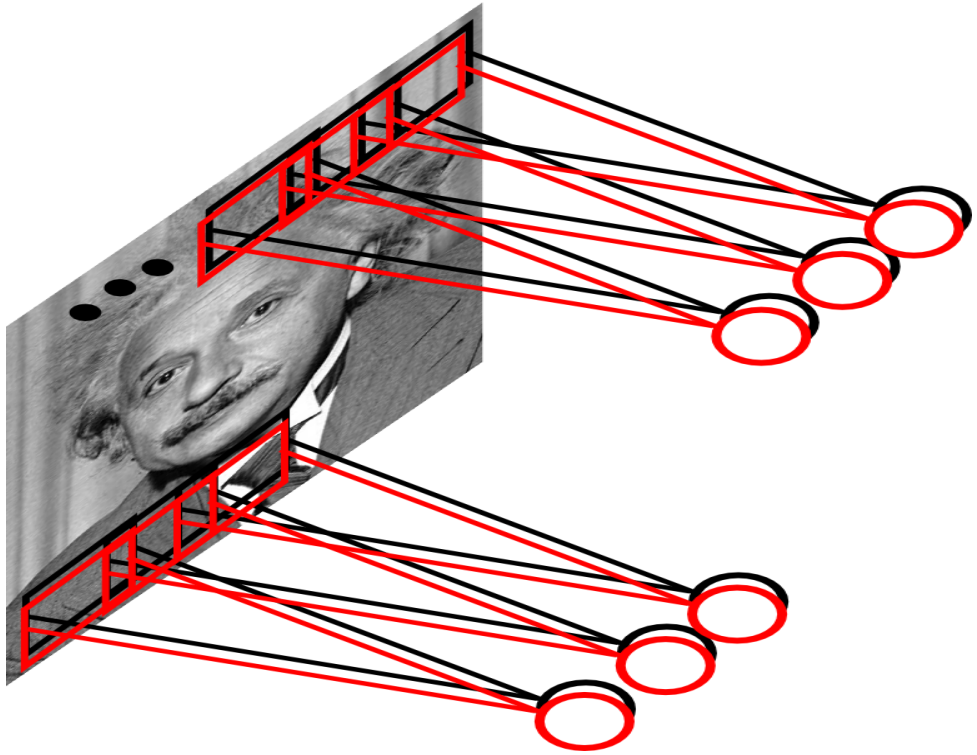
- ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture.

Why CNN for image processing



- It assumes that nearby pixels share similar structure
- The parameters for a single filter are shared across the entire input image in the sense that a single filter “slides” across an image and produces a new output with one image channel.
- This allows the parameters to be invariant to the location of objects in images.

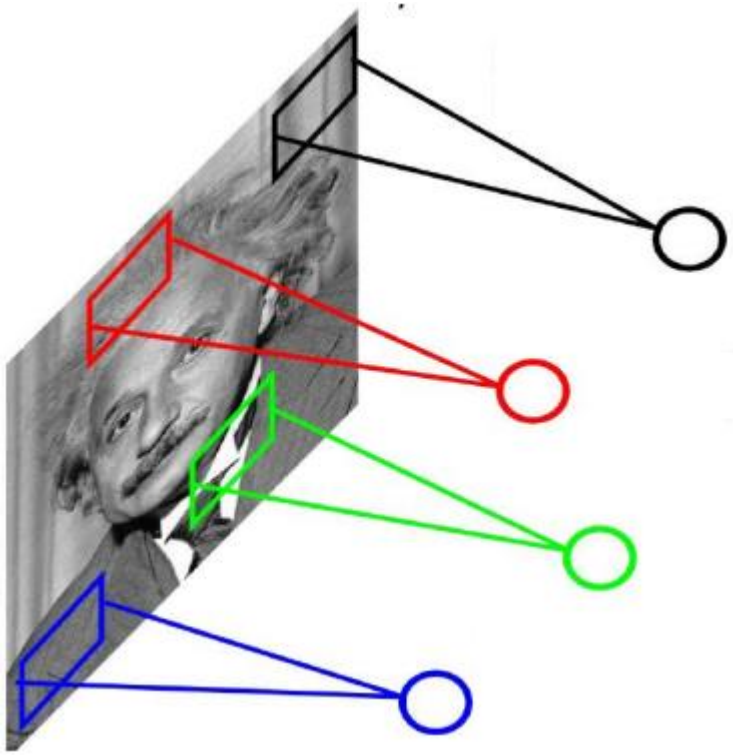
Convolutional Layer



Learn multiple filters.

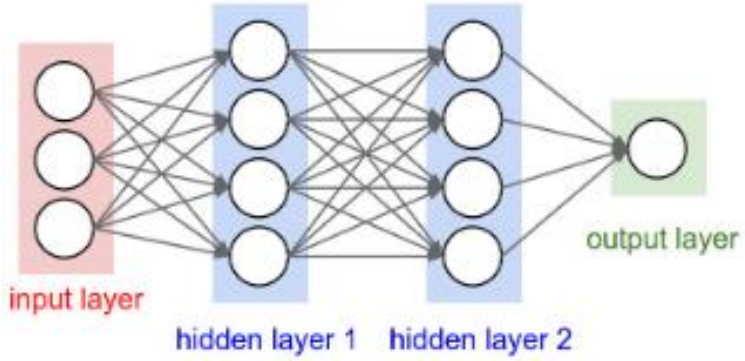
E.g.: 200x200 image, 100 Filters, Filter Dimension(size): 10x10, 10K parameters

Convolution Layers

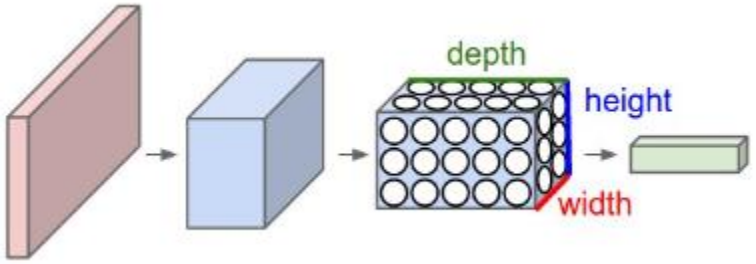


- In image processing/vision we usually want to apply the same convolution mask at each location.
- Each neuron has the same weights
  - e.g. 200x200 image, 10 x 10 mask means only 100 weights to learn

MLP Versus CNN



A regular 3-layer Neural Network.

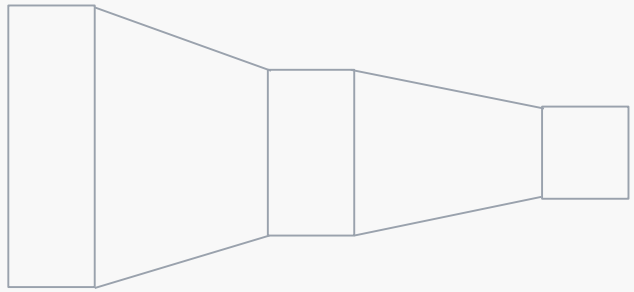


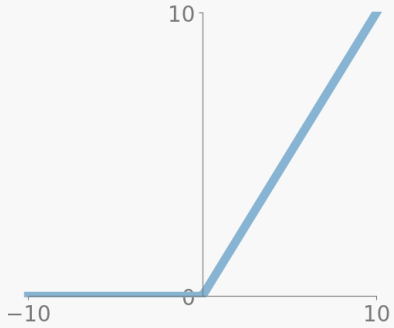
A ConvNet arranges its neurons in three dimensions

- Multilayer Perceptron Network
  - Wide application scenario – not just images
  - Neurons are fully connected – can't scale well to large size data (e.g. images)
- Convolutional Neural Network
  - The layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.
  - Each neuron is only connected to a small region of previous layer
  - Typical CNN structure: Input-conv-activation-pool-fully connected-output

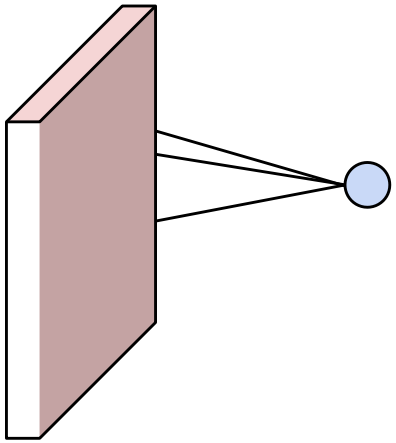
Convolution Layers

Components of a Convolutional Network

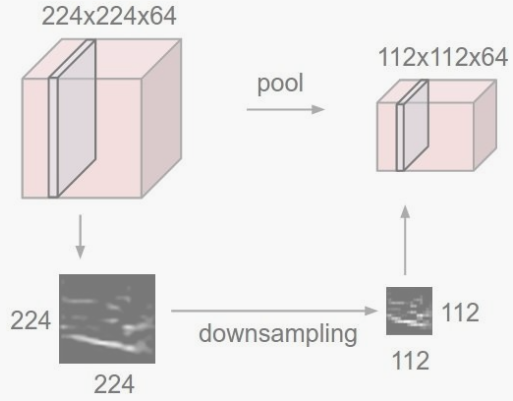




Convolution Layers



Pooling Layers

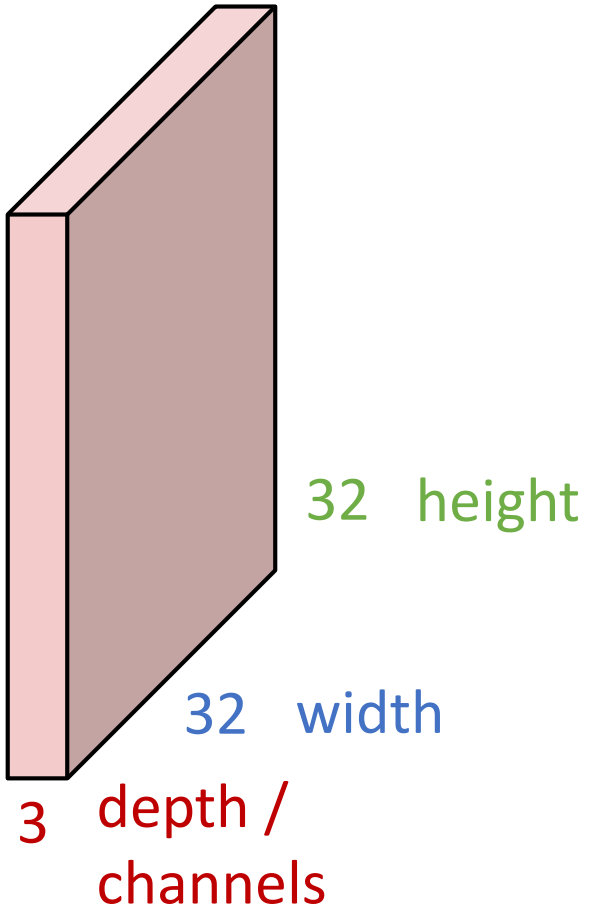


Normalization

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

Convolution Layer

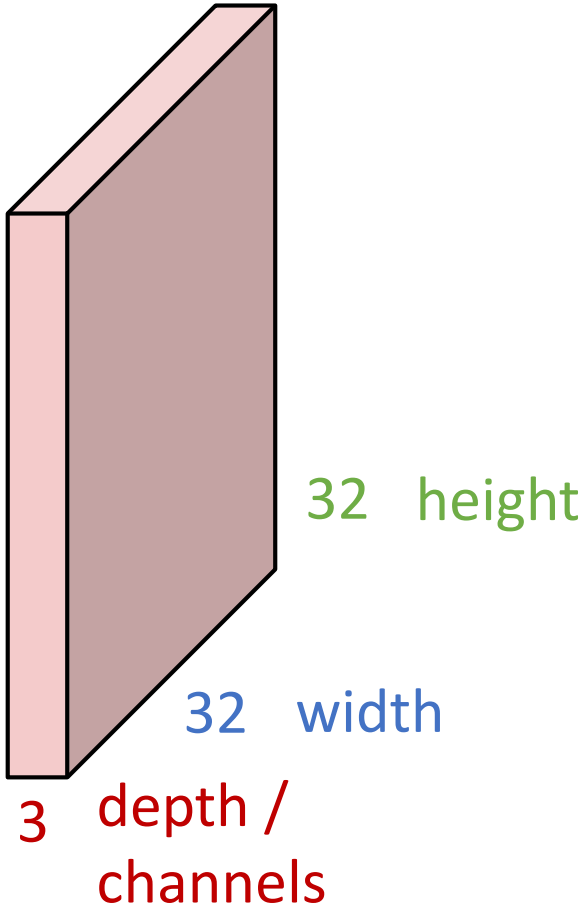
3x32x32 image: preserve spatial structure



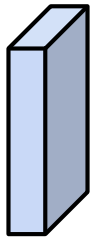
- Input 3D tensor (volume)
- No longer vector

Convolution Layer

3x32x32 image



3x5x5 filter

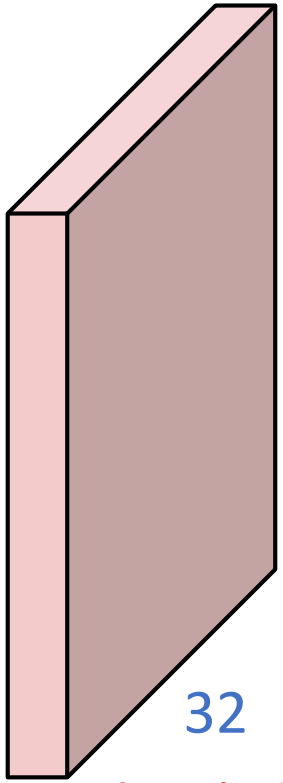


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

Filters always extend the full depth of the input volume

3x32x32 image

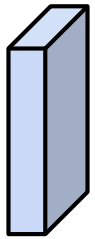


32 height

32 width

3 depth / channels

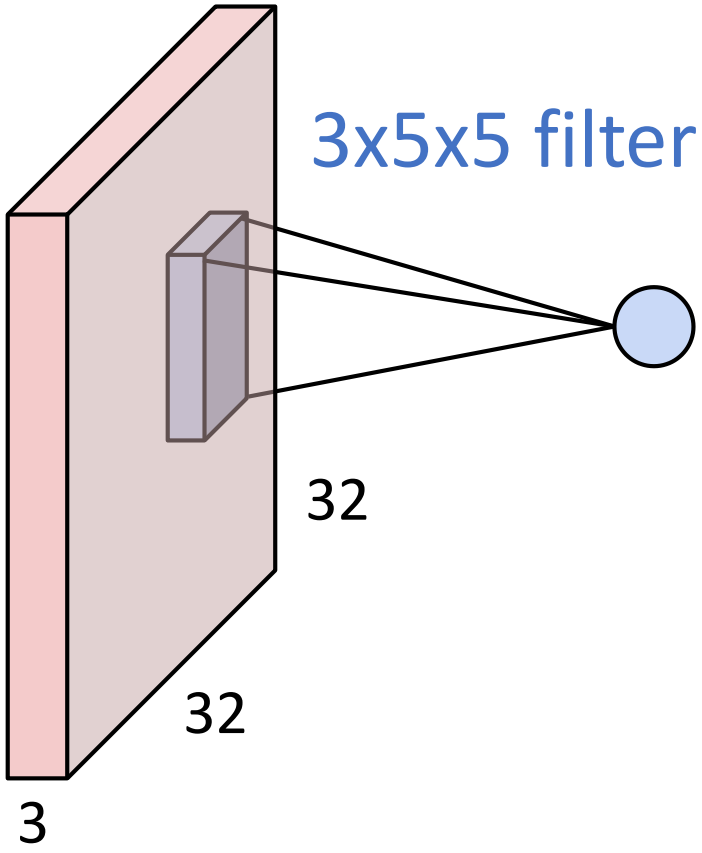
3x5x5 filter

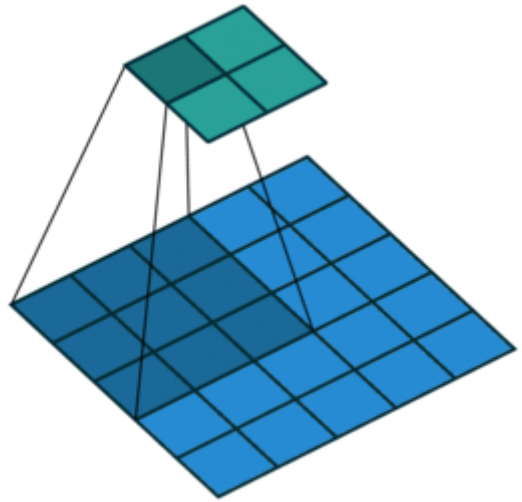


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

3x32x32 image

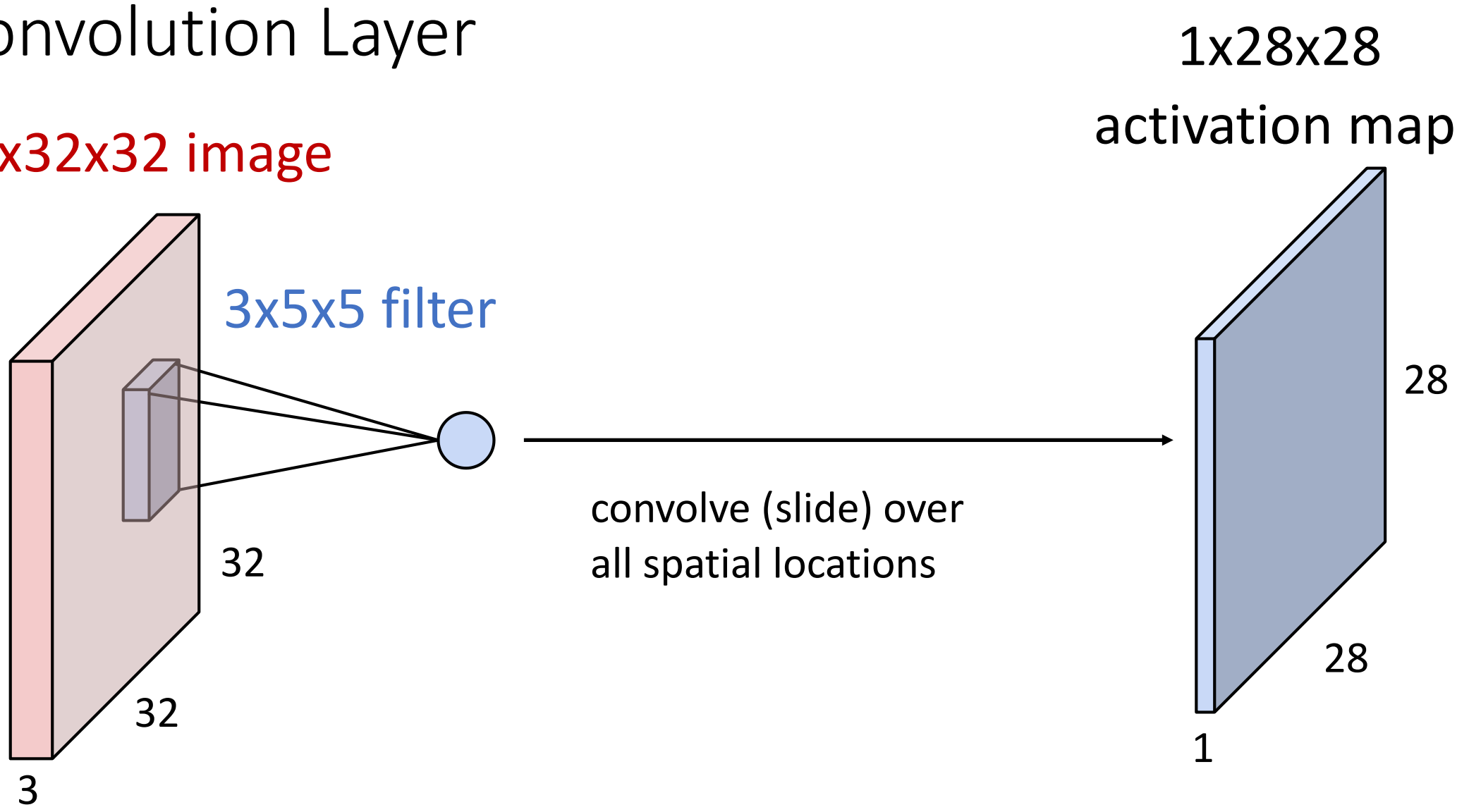




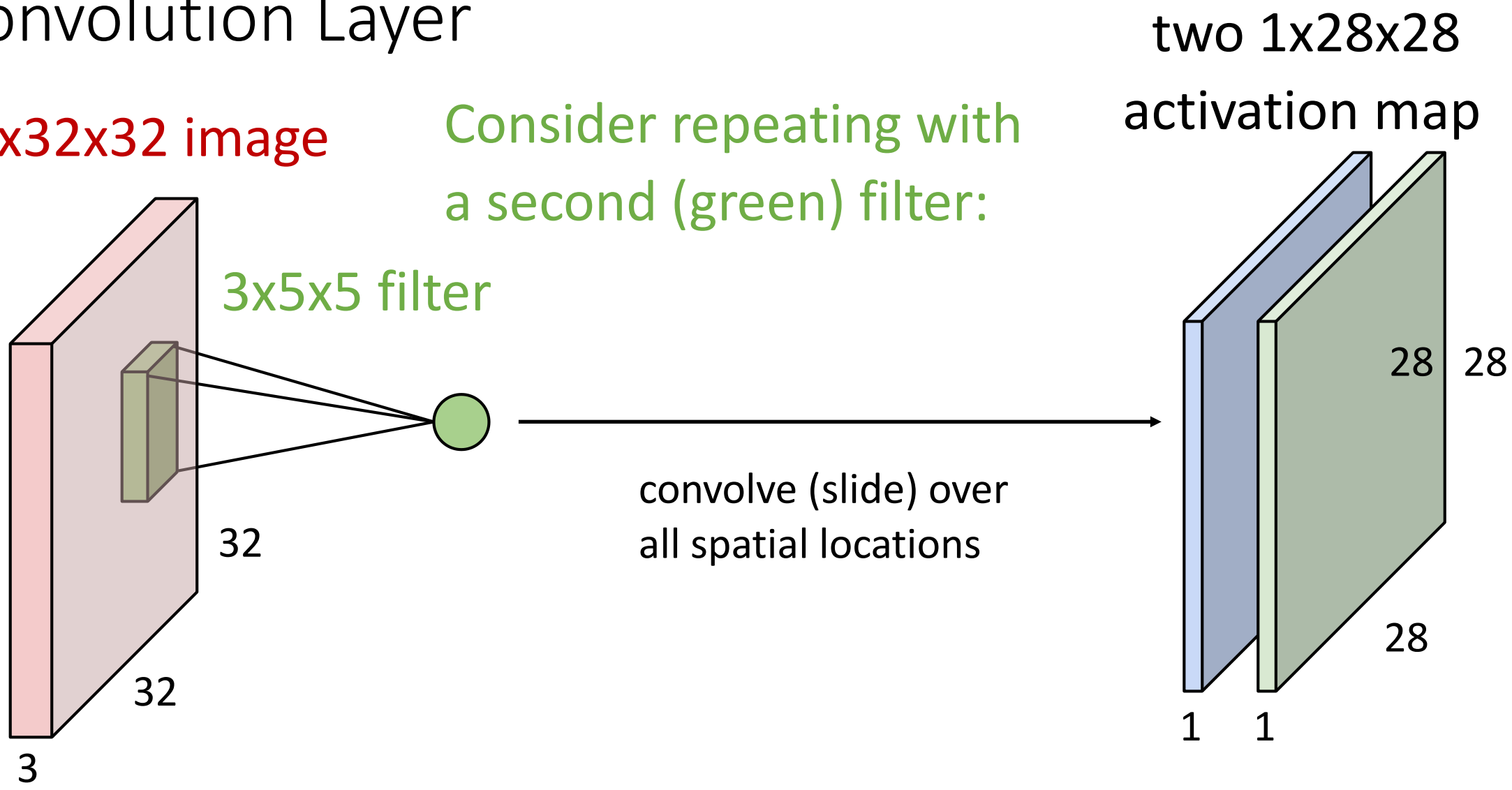
1 number:
the result of taking a dot product between the filter and a small 3x5x5 chunk of the image
(i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x + b$$

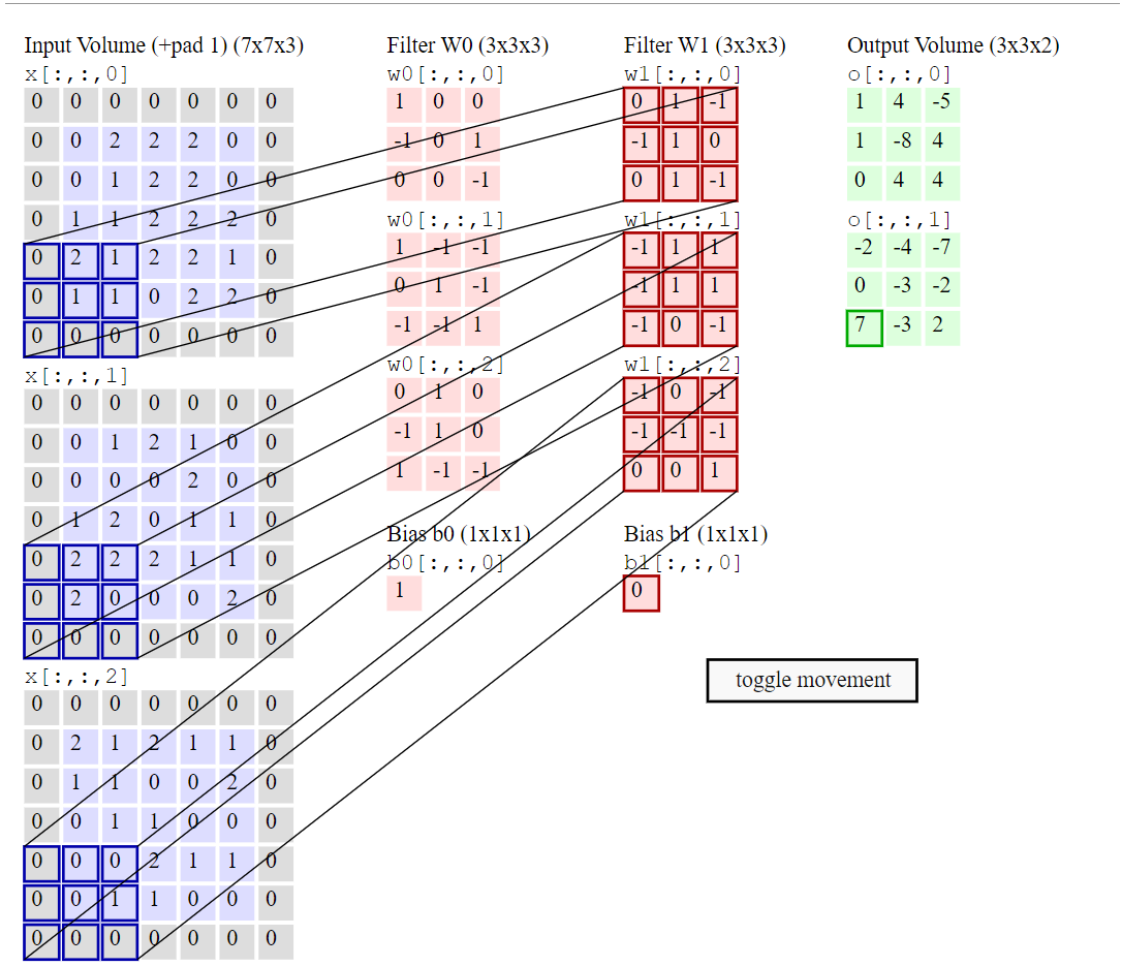
Convolution Layer



Convolution Layer

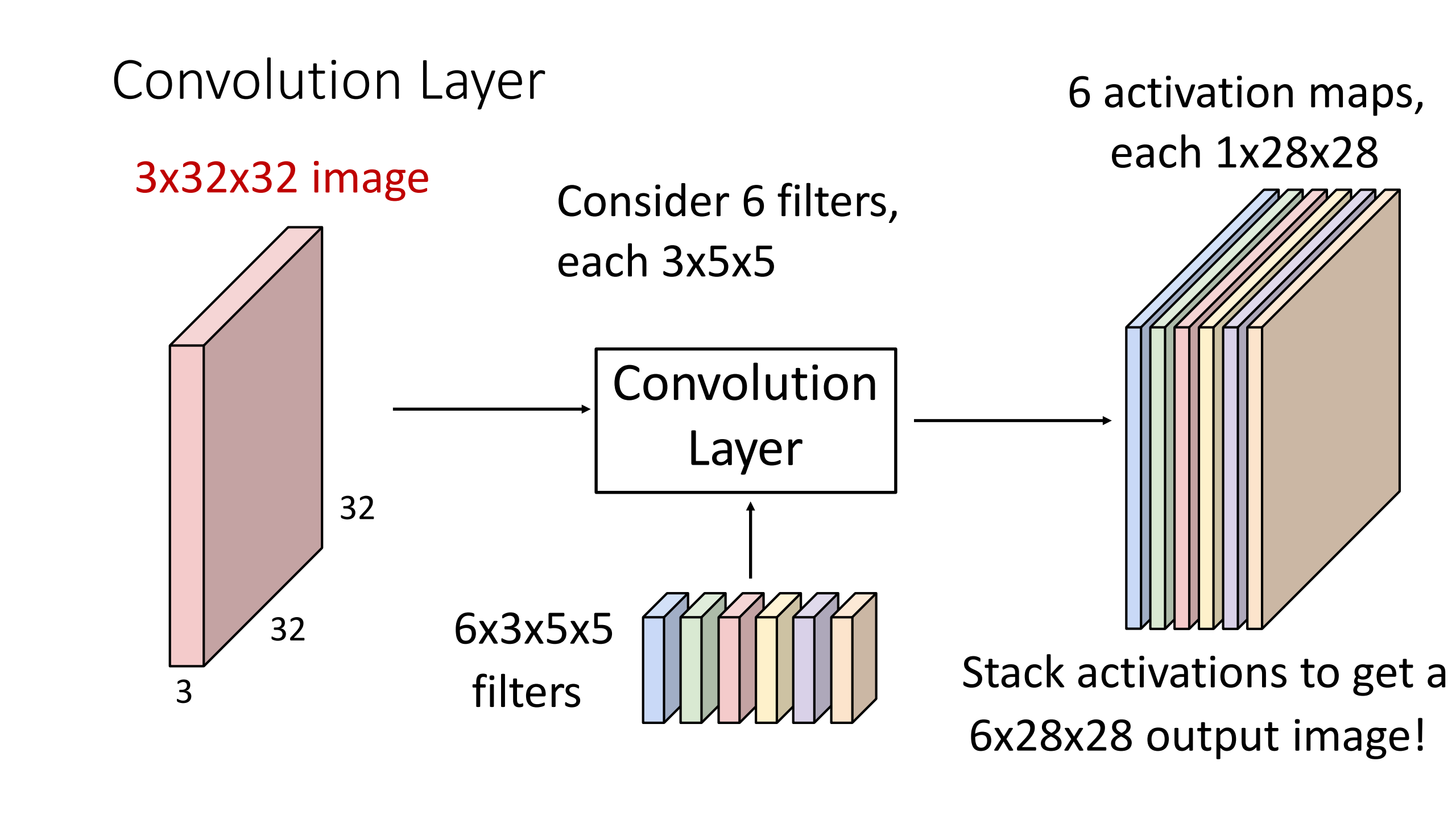


A closer look at spatial dimensions

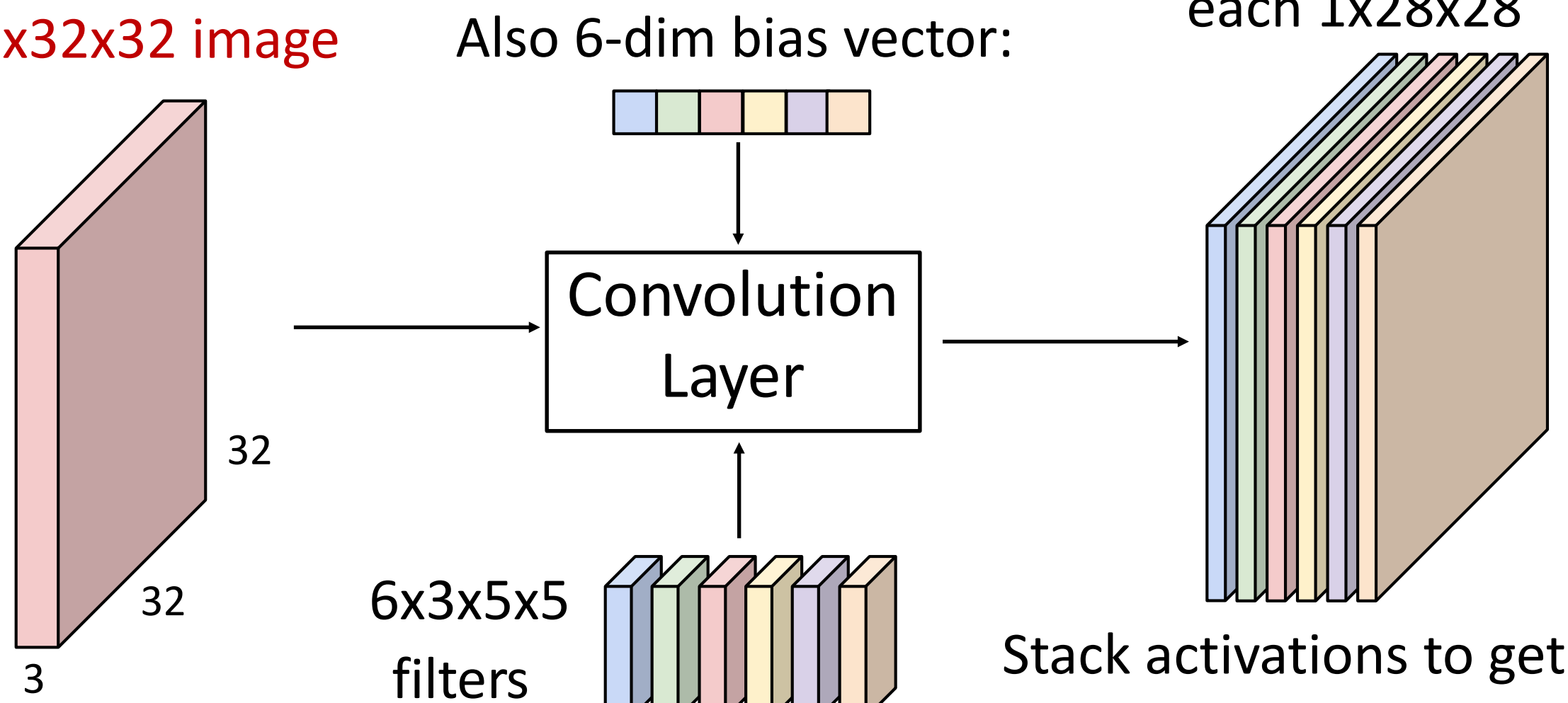


https://cs231n.github.io/convolutional-networks/

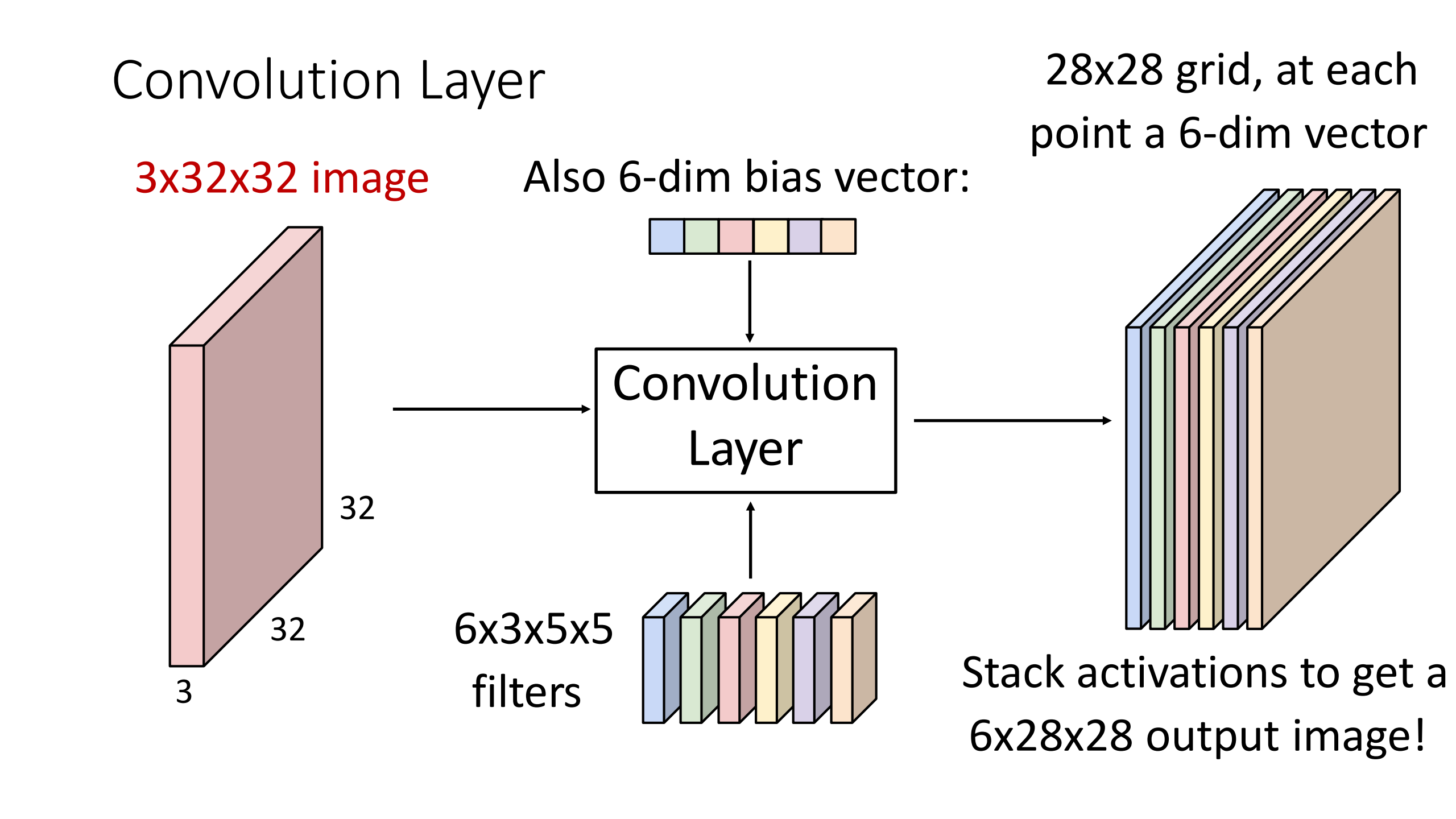
Convolution Layer



Convolution Layer

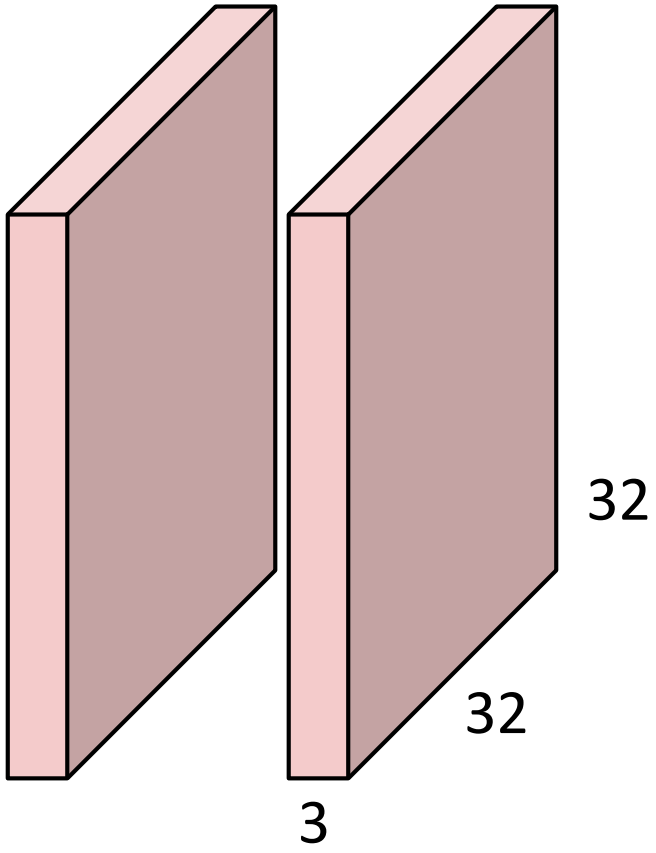


Convolution Layer

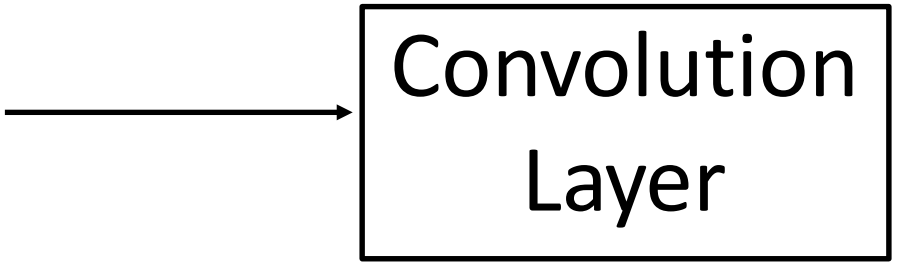


Convolution Layer

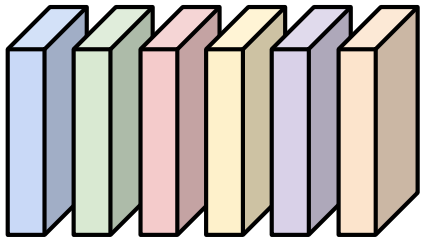
2x3x32x32
Batch of images



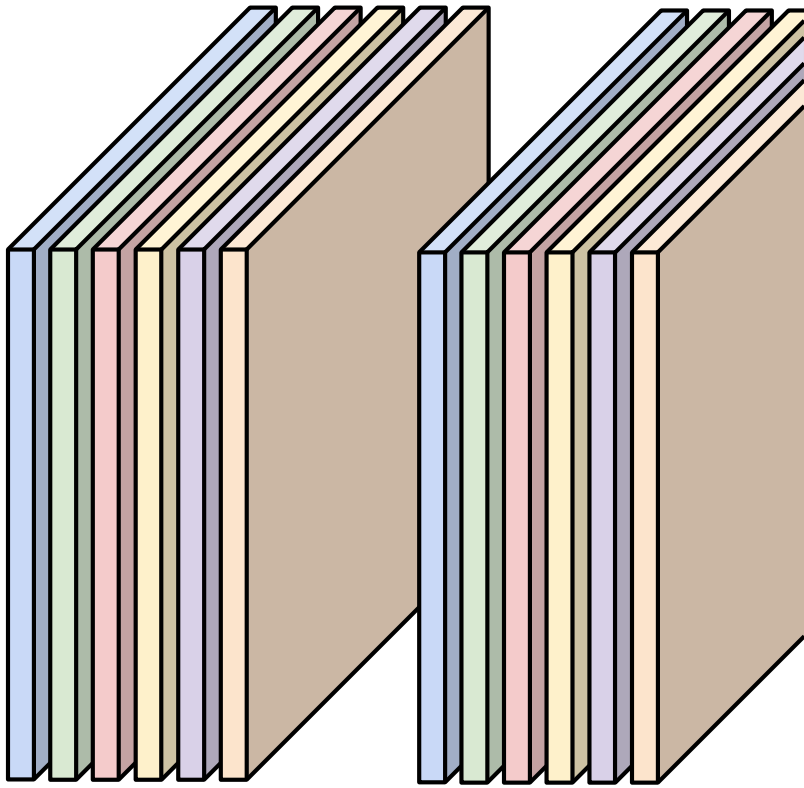
Also 6-dim bias vector:



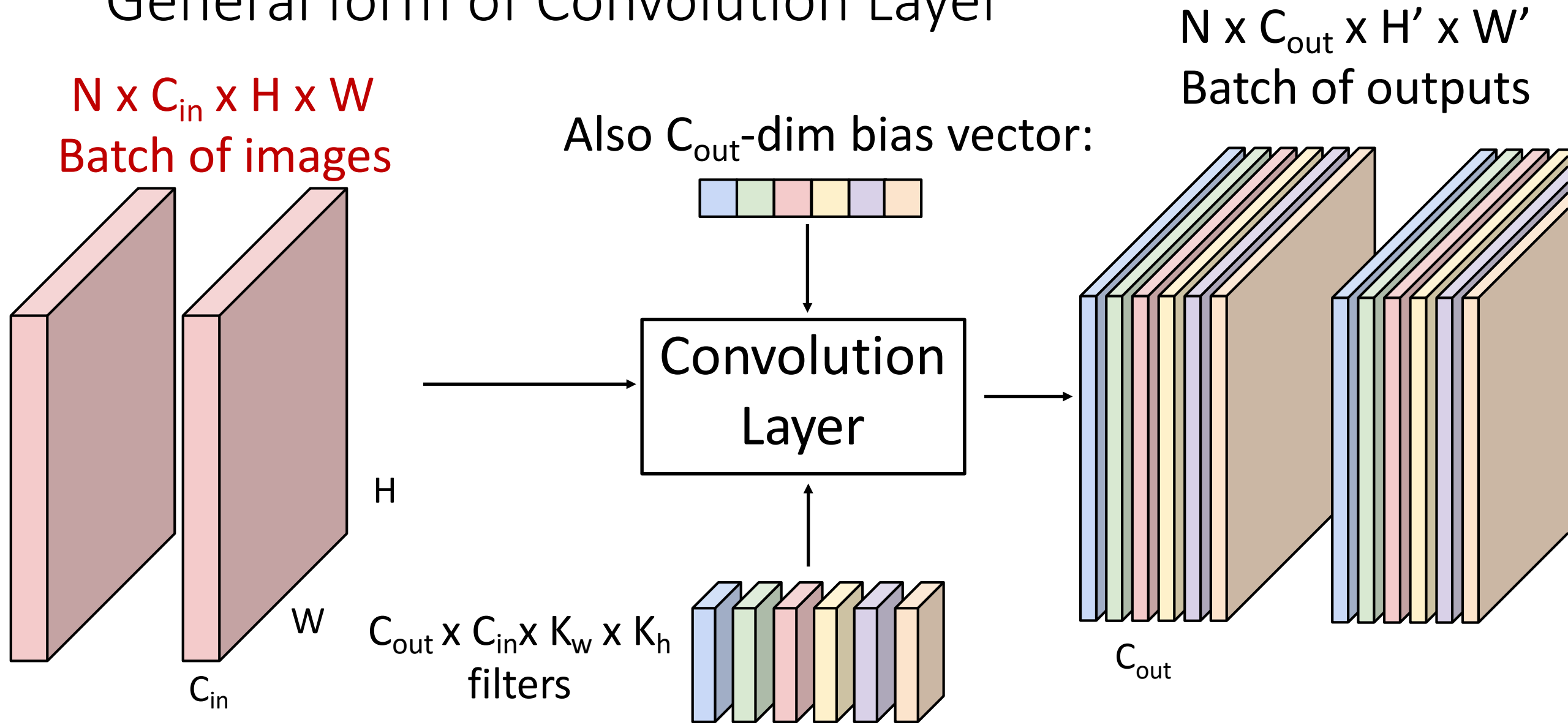
6x3x5x5
filters



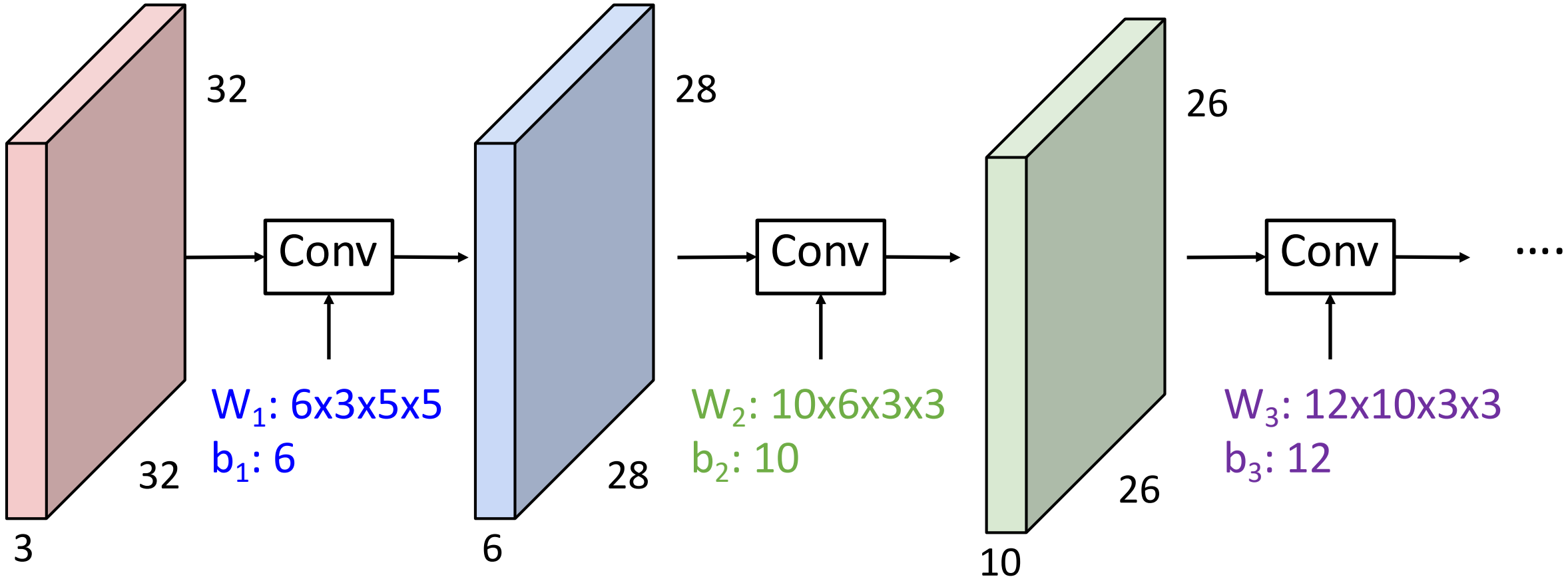
2x6x28x28
Batch of outputs



General form of Convolution Layer



Stacking Convolutions



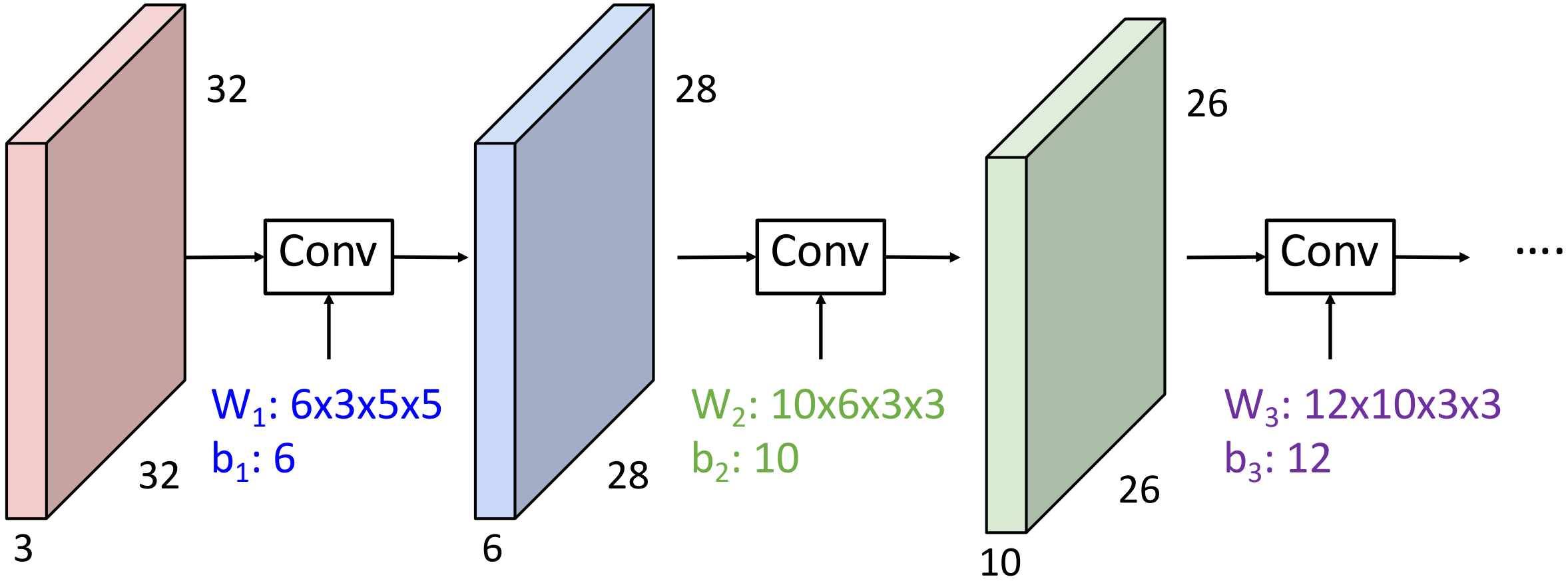
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

Stacking Convolutions

Q: What happens if we stack two convolution layers?

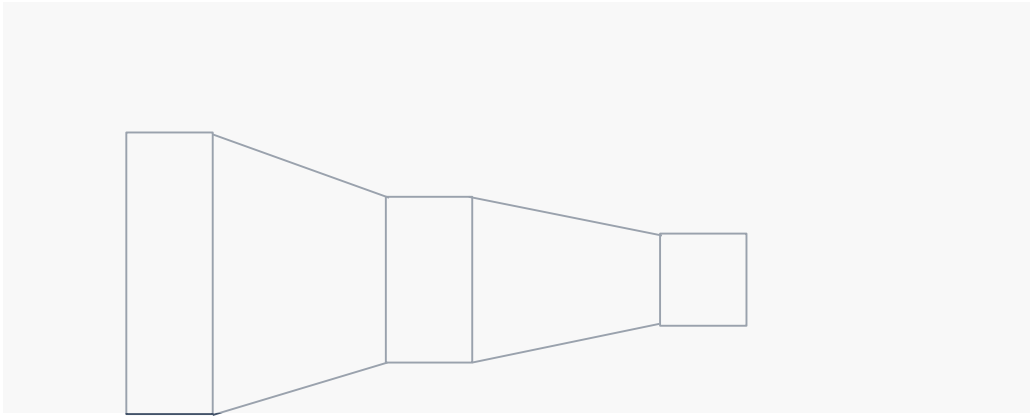


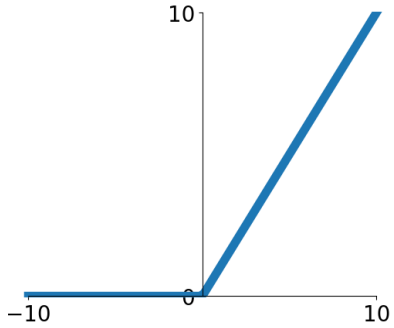
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

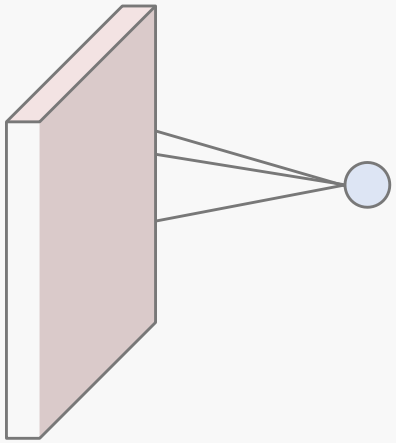
Second hidden layer:
N x 10 x 26 x 26

Components of a Convolutional Network

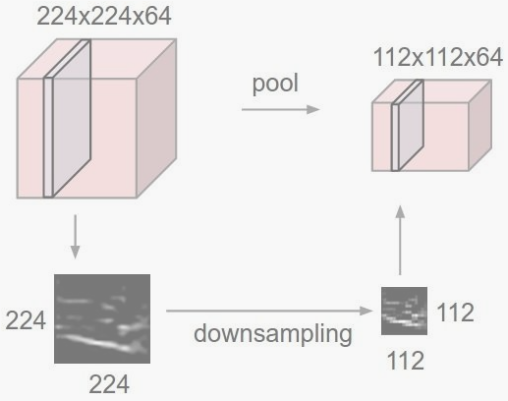




Convolution Layers



Pooling Layers



Normalization

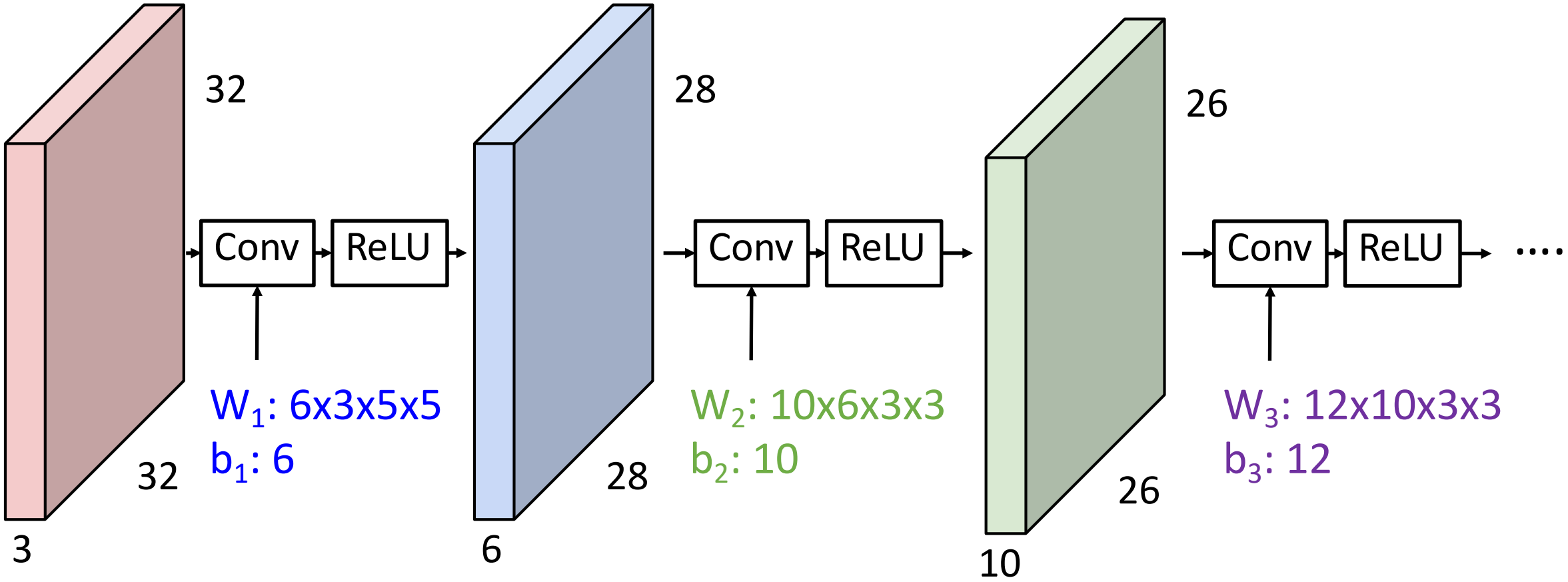
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Stacking Convolutions

Q: What happens if we stack two convolution layers?

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

A: We get another convolution!



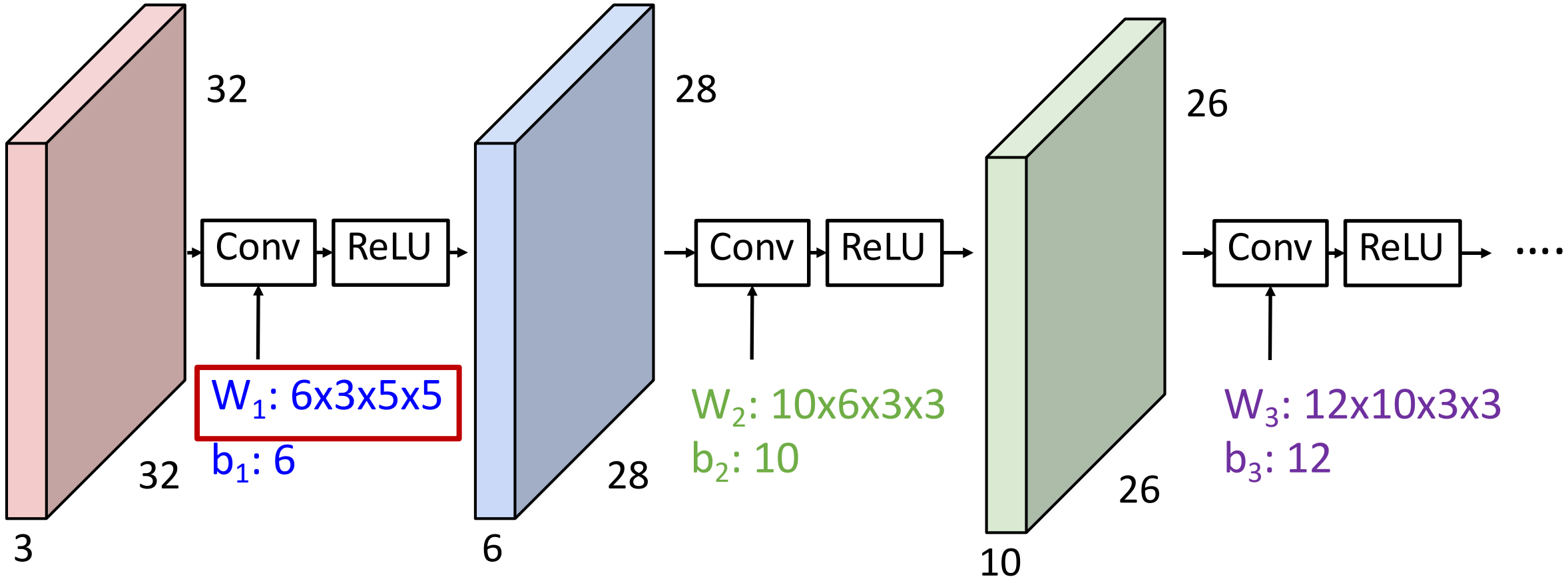
Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

What do convolutional filters learn?

What do convolutional filters learn?

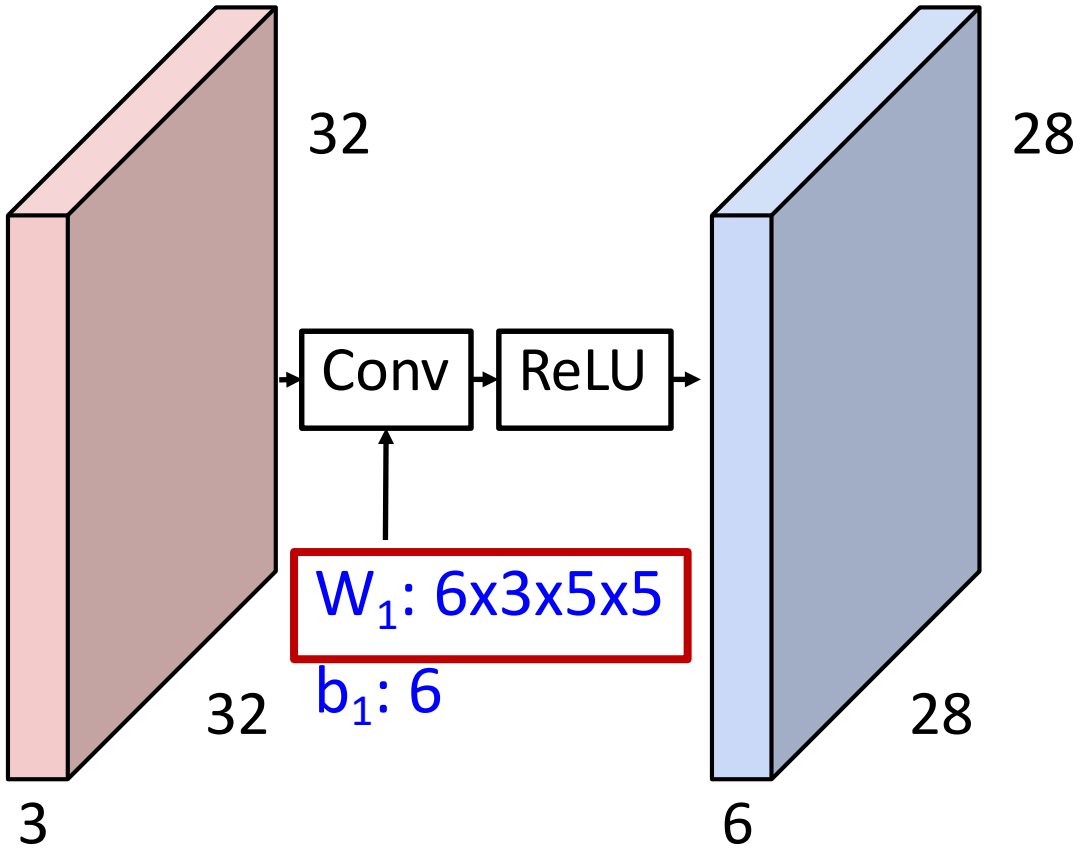


Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

Second hidden layer:
N x 10 x 26 x 26

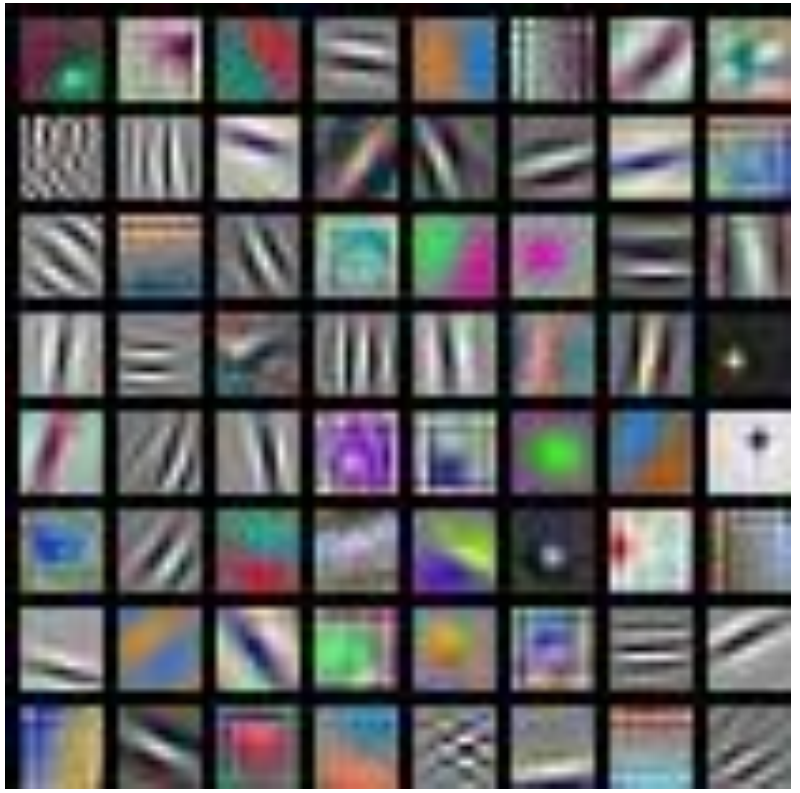
What do convolutional filters learn?



Input:
N x 3 x 32 x 32

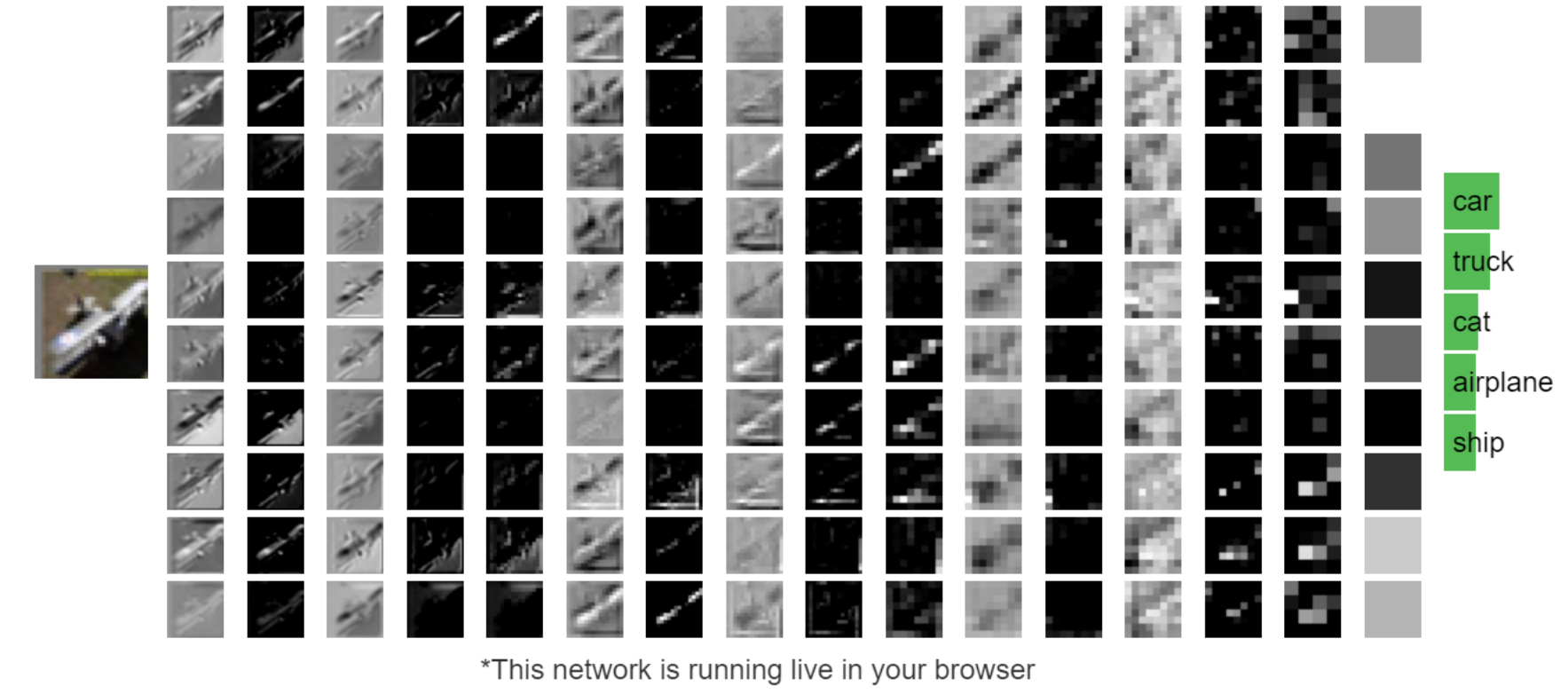
First hidden layer:
N x 6 x 28 x 28

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



AlexNet: 64 filters, each 3x11x11

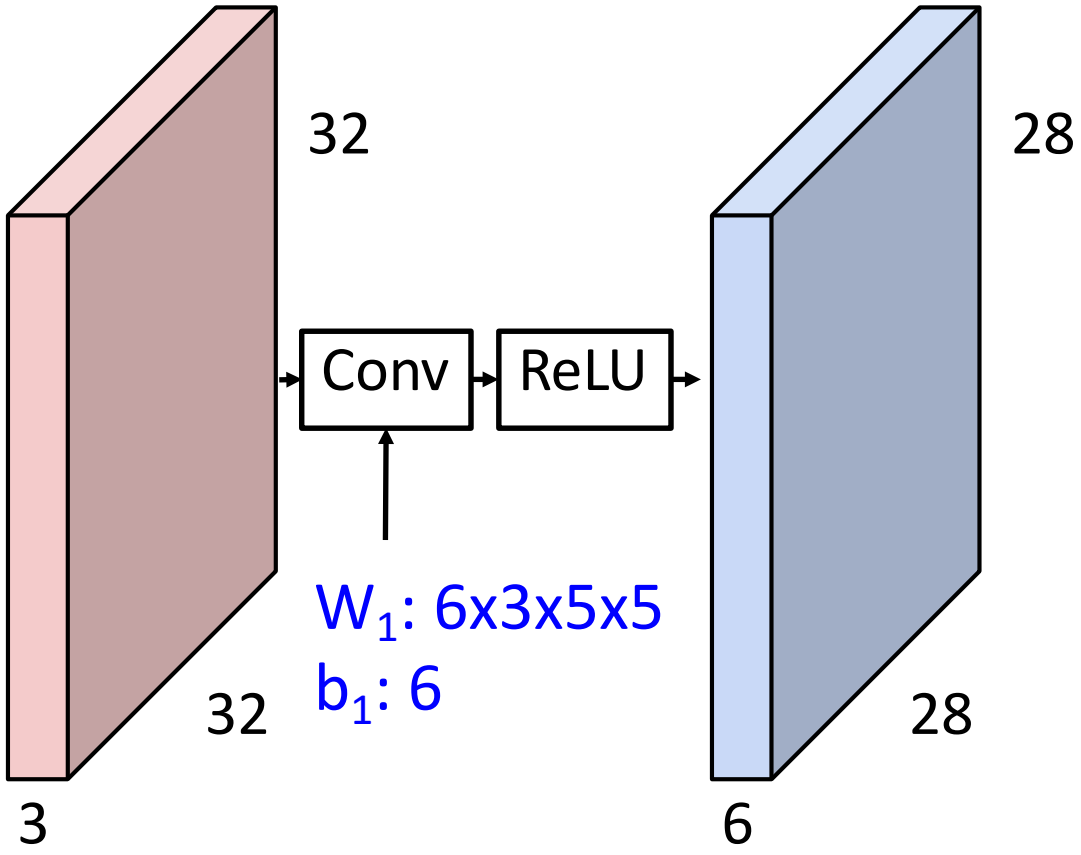
What do convolutional filters learn?



https://cs231n.stanford.edu/

A closer look at spatial dimensions

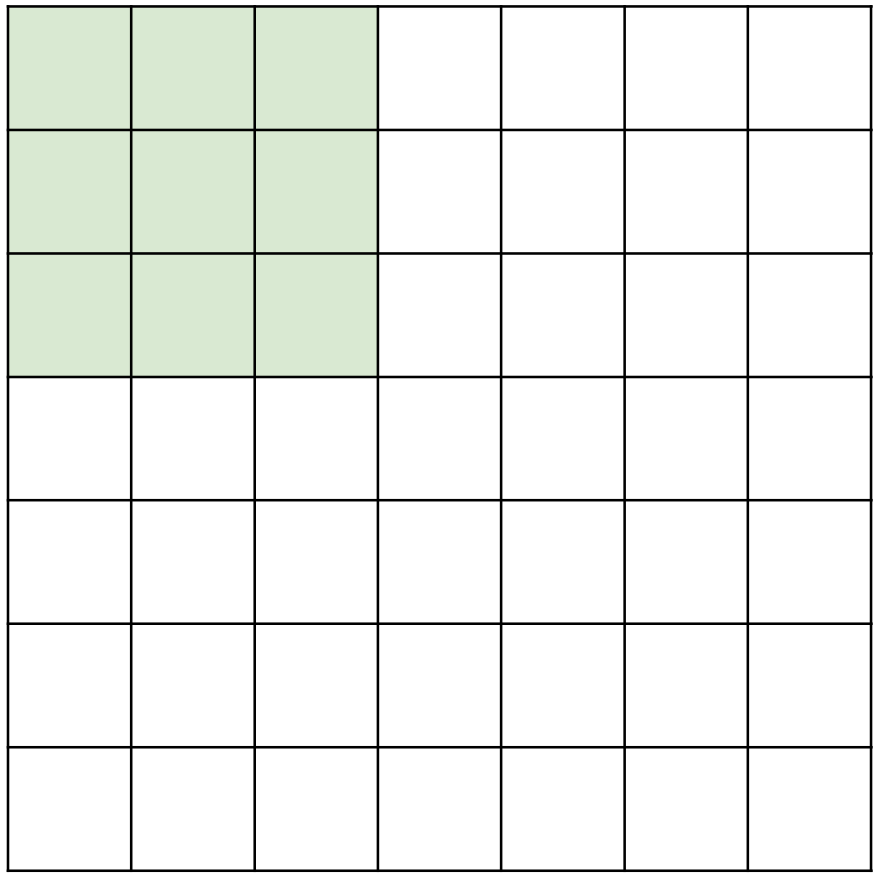
A closer look at spatial dimensions



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

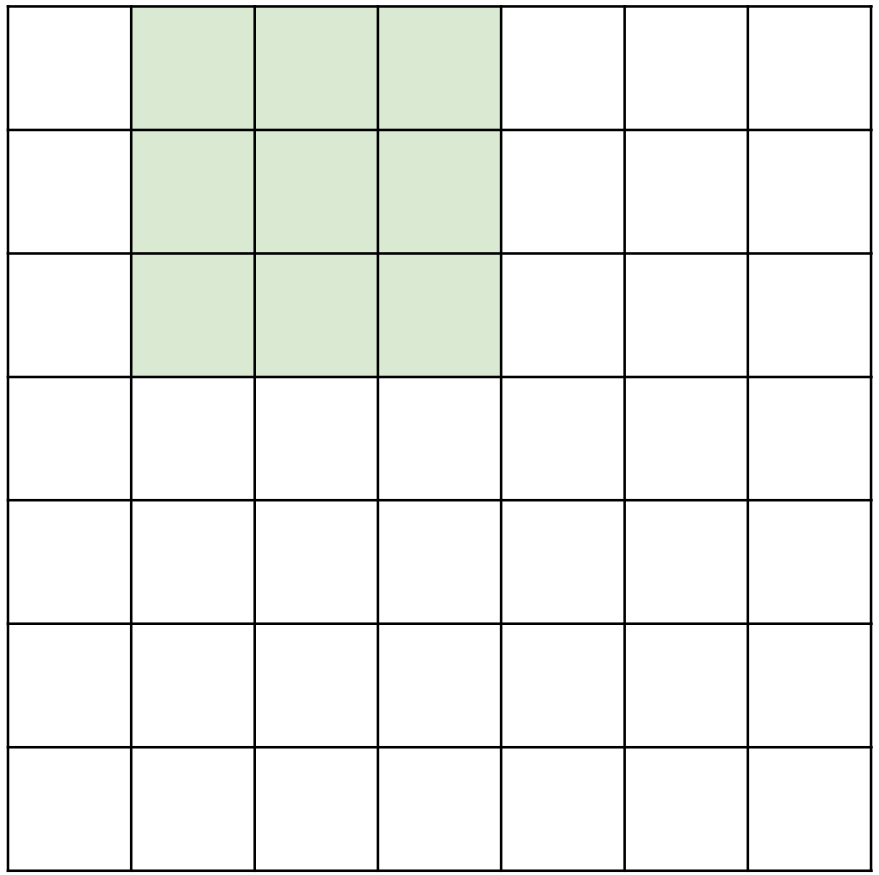
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

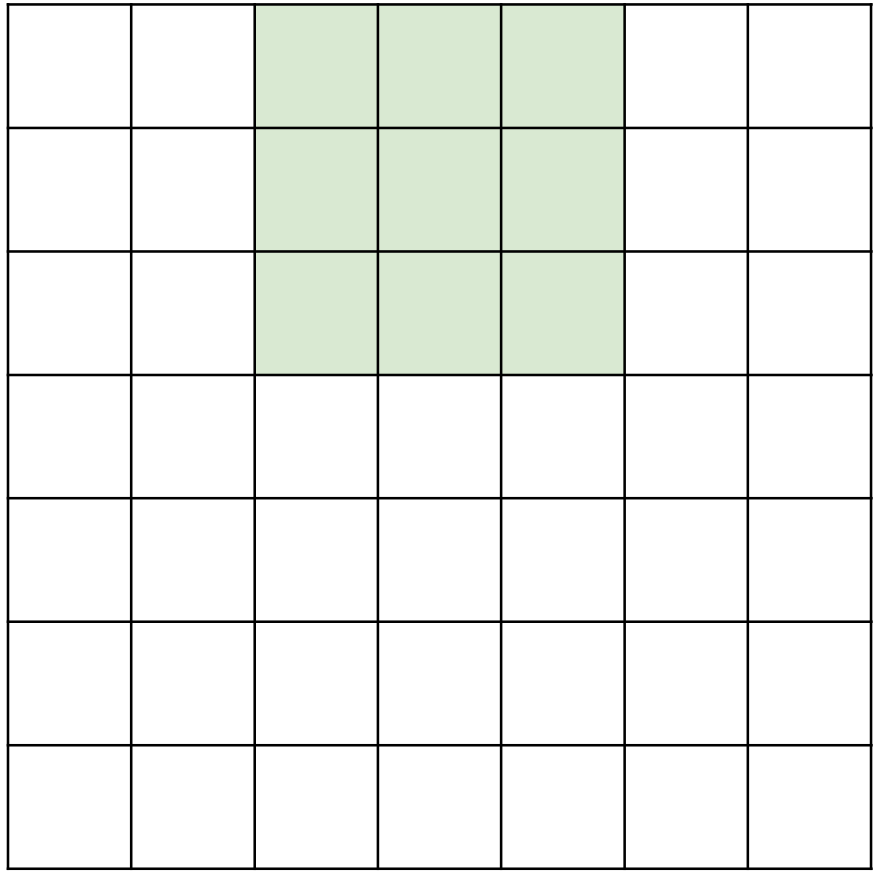
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

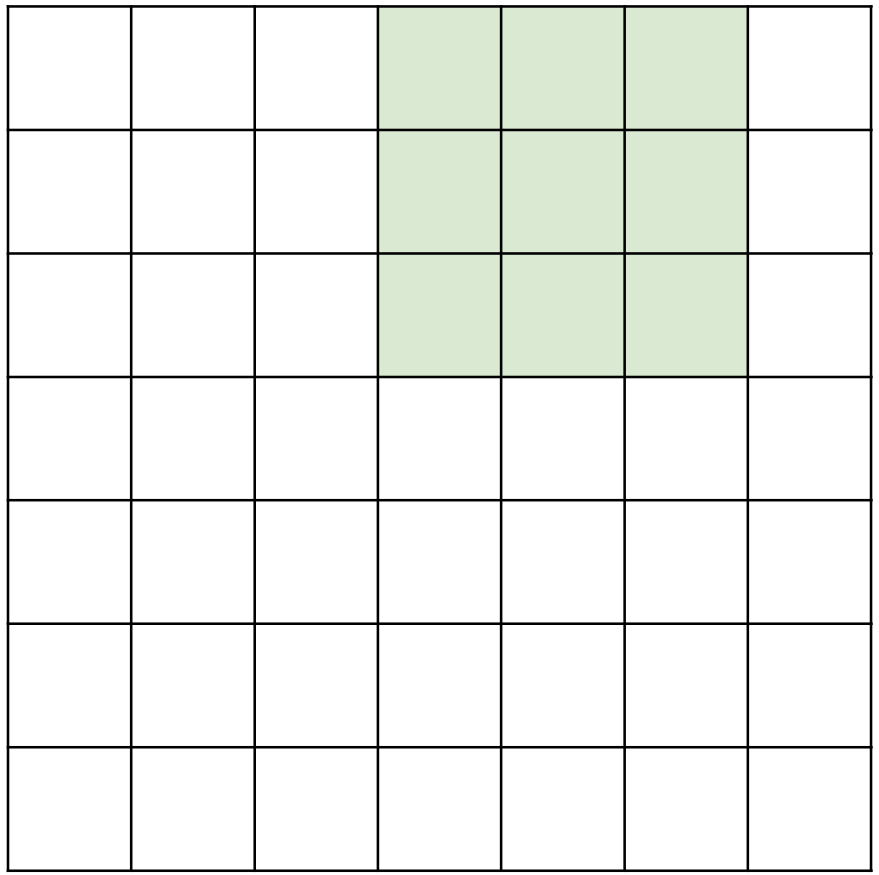
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

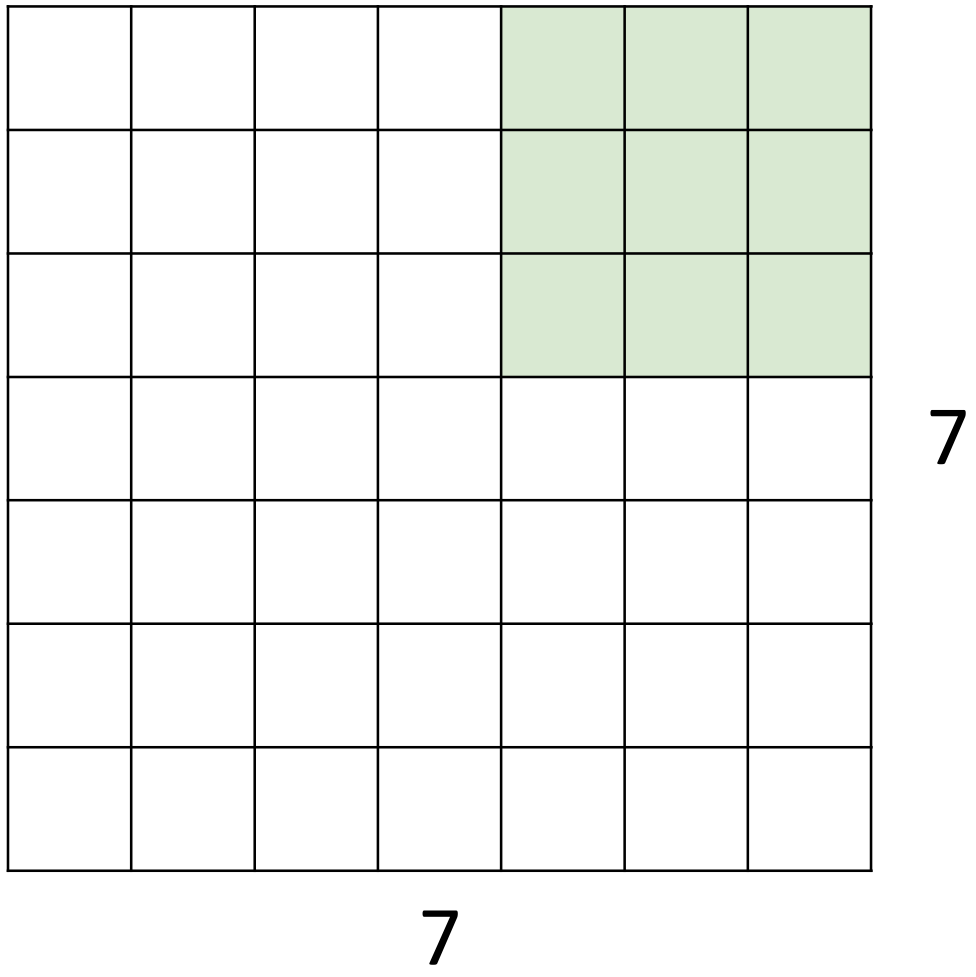
A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

A closer look at spatial dimensions

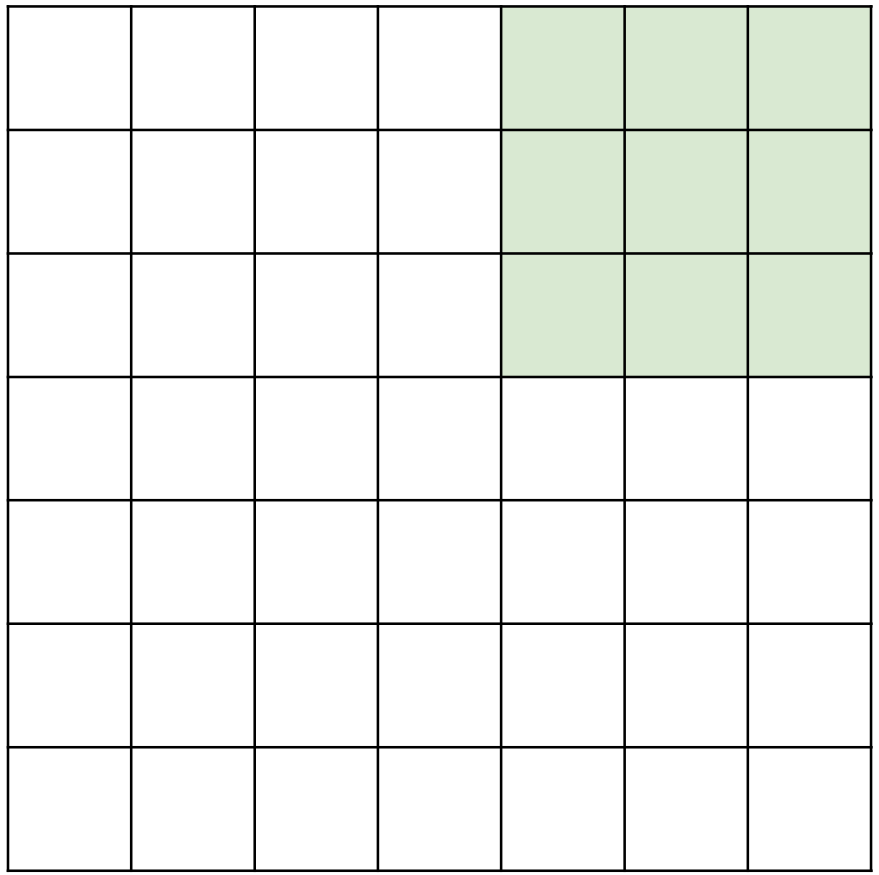


Input: 7x7

Filter: 3x3

Output: 5x5

A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

Output: 5x5

In general:

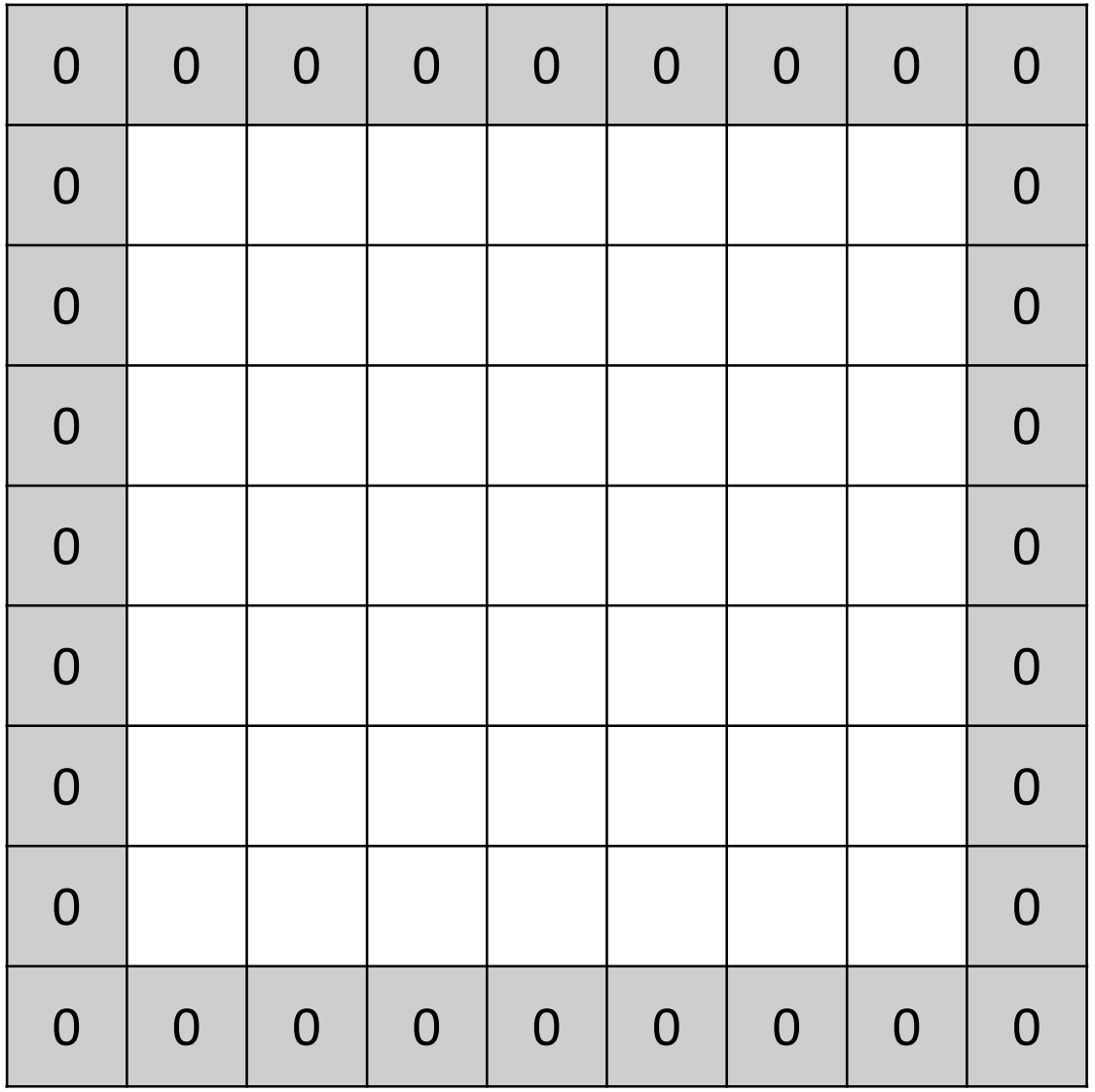
Input: W

Filter: K

Output: W – K + 1

Problem: Feature maps “shrink” with each layer!

A closer look at spatial dimensions



Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input: W

Filter: K

Output: W – K + 1 + 2P

Problem: Feature maps “shrink” with each layer!

Solution: **padding**

Add zeros around the input

A closer look at 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: 7x7

In general:

Input: W

Filter: K

Padding: P

Output: W – K + 1 + 2P

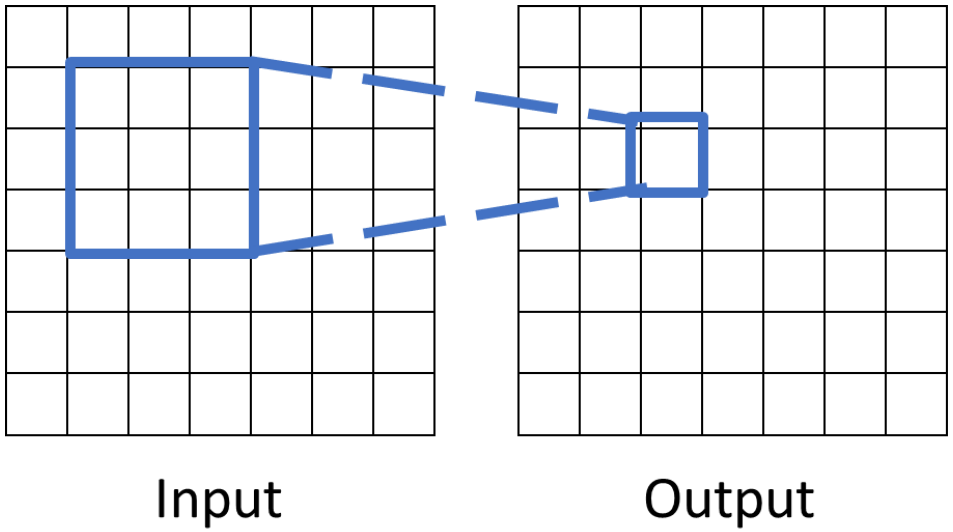
Very common:

Set P = (K – 1) / 2 to make output have same size as input!

P = 1

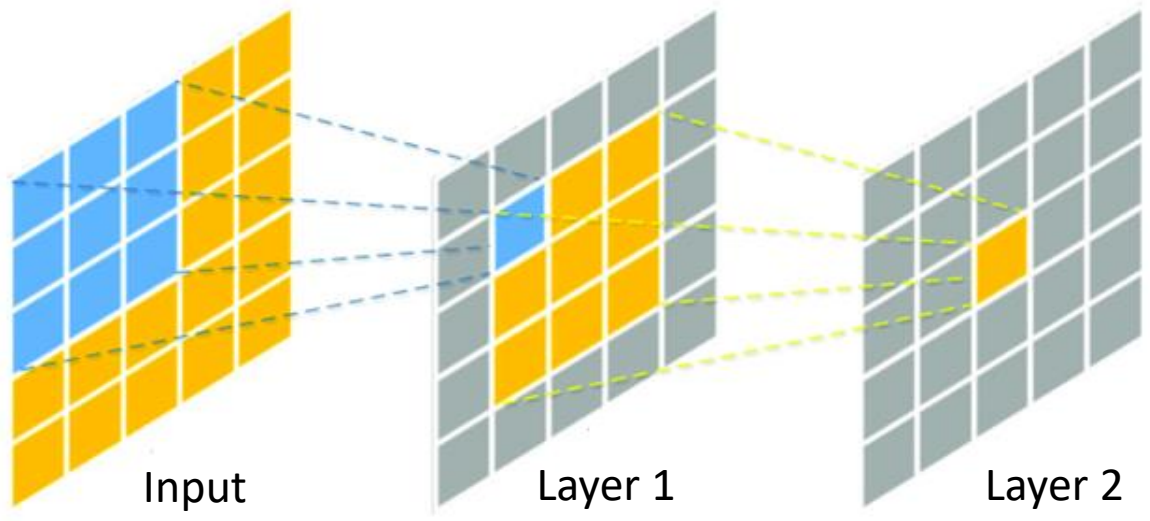
Receptive Fields

- For convolution with kernel size K, each in the input element in the output depends on a K x K **receptive field**.
- **Receptive field** refers to the region of the input image that a particular neuron in a convolutional layer is “looking at” or taking into account when making its predictions or feature extractions.



Receptive Fields

For the first layer the receptive field size is K and for each successive convolution adds K – 1 to the receptive field size

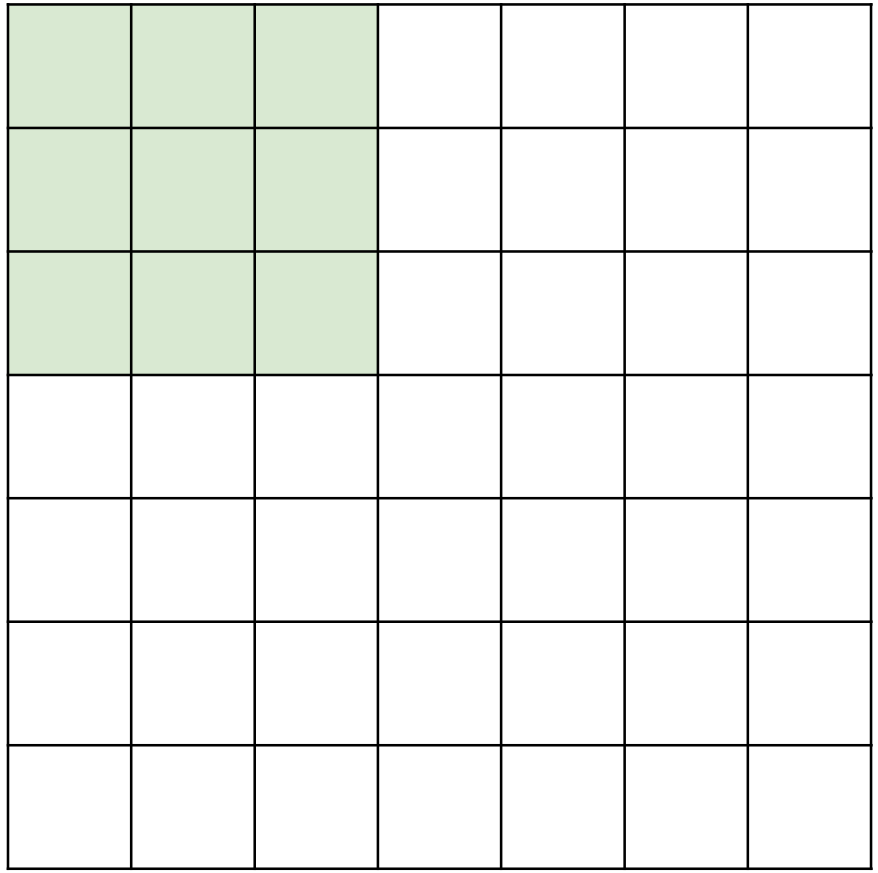


With L layers the receptive field size is 1 + L * (K –1)

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

Solution: Downsample inside the network

Strided Convolution

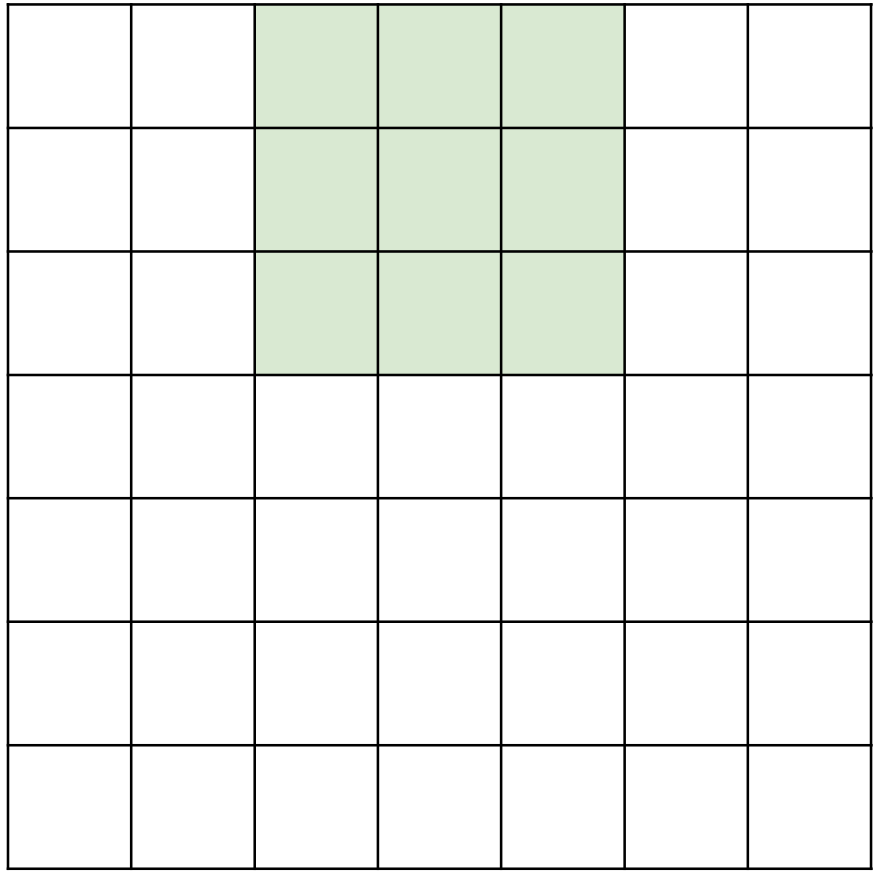


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution

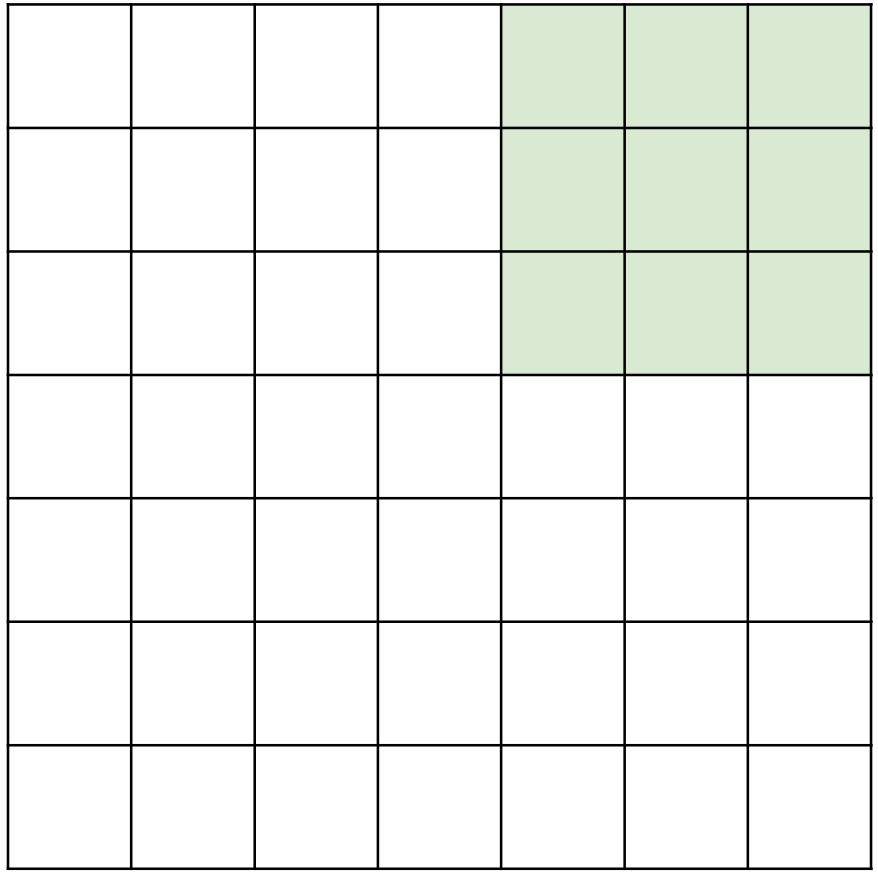


Input: 7x7

Filter: 3x3

Stride: 2

Strided Convolution



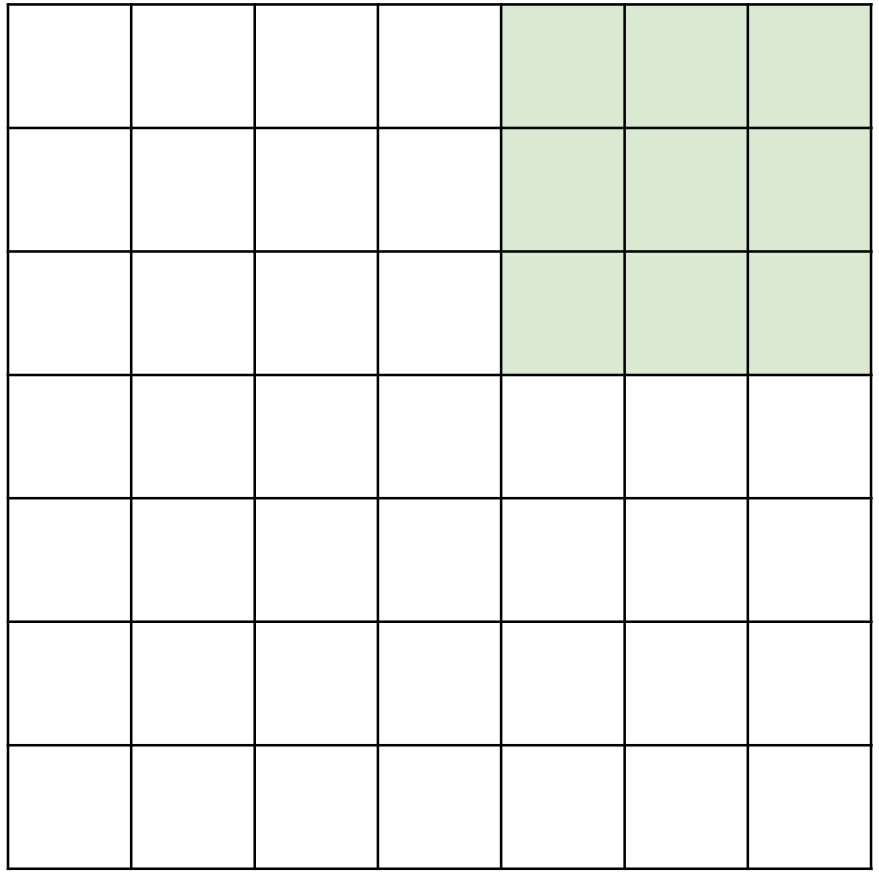
Input: 7x7

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

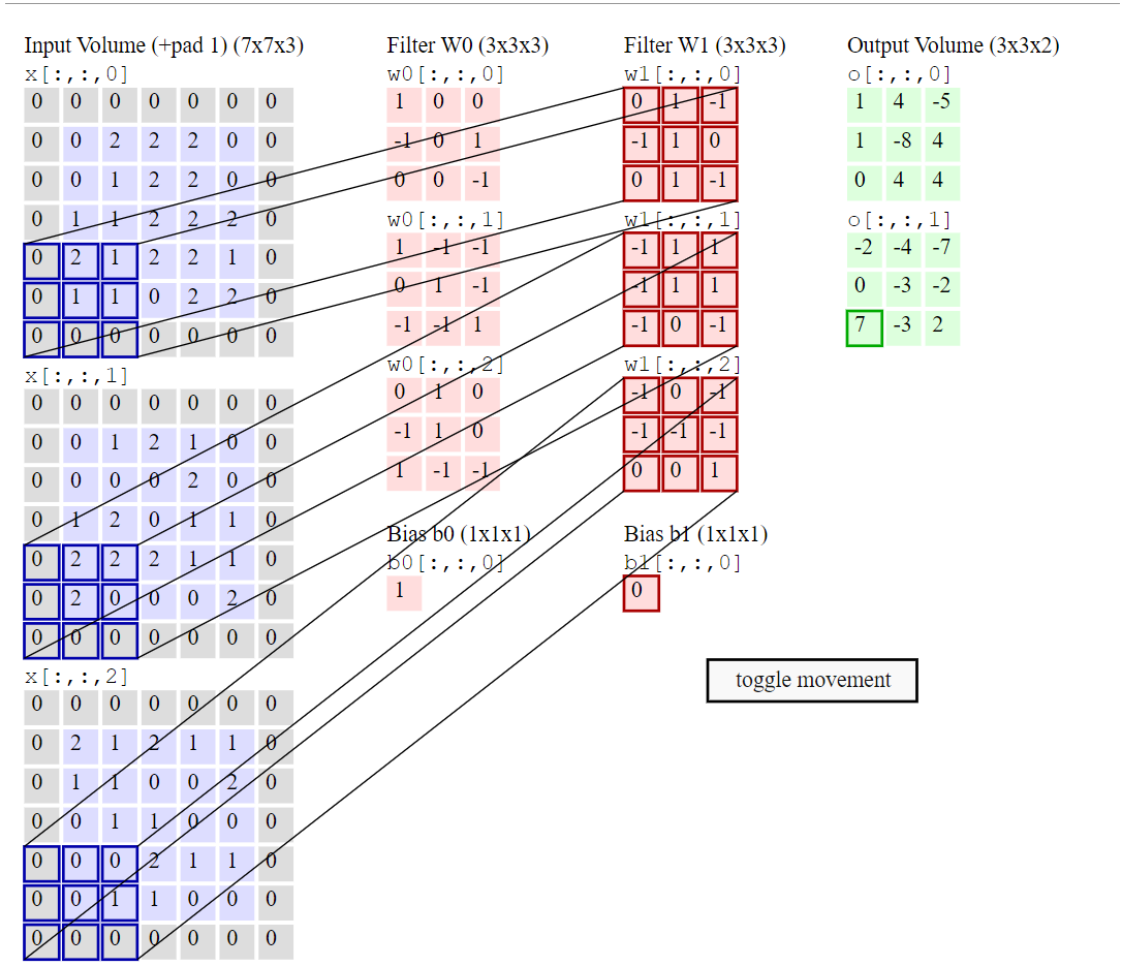
Output: $\frac{(W - K + 2p)}{S} + 1$

General formula to Calculate Receptive field

$$\mathrm{RF}_L = \mathrm{RF}_{L-1} + \left((K_L - 1) \times \prod_{i=1}^{L-1} S_i \right)$$

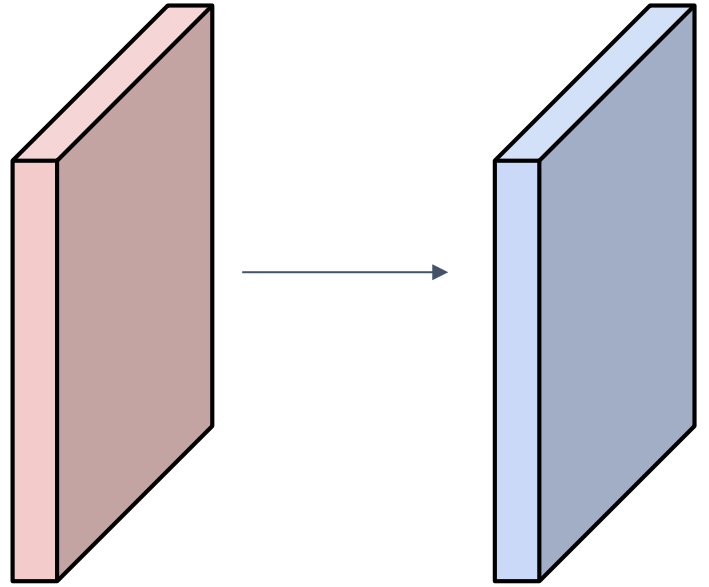
- **Where**
- $\boldsymbol{RF_L}$ is Receptive field of L
- $\boldsymbol{RF_{L-1}}$ Receptive field of the **previous layer** (L–1).
- $\boldsymbol{K_L}$
  Kernel size (e.g., 3 for a 3×3 filter) at layer L.
- $\boldsymbol{S_i}$ is stride at layer i

A closer look at spatial dimensions



https://cs231n.github.io/convolutional-networks/

Convolution Example

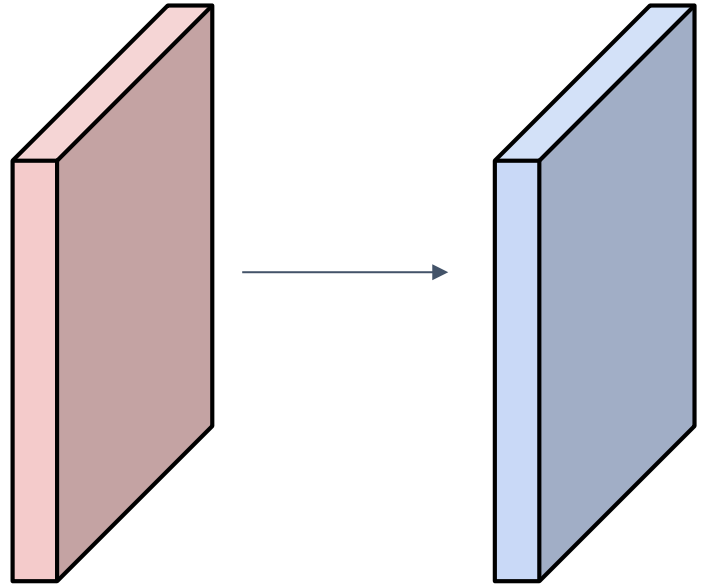


Input volume: 3 x 32 x 32

10 3x5x5 filters with stride 1, pad 2

Output volume size: ?

Convolution Example



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

10 **3x5x5** filters with stride **1**, pad **2**

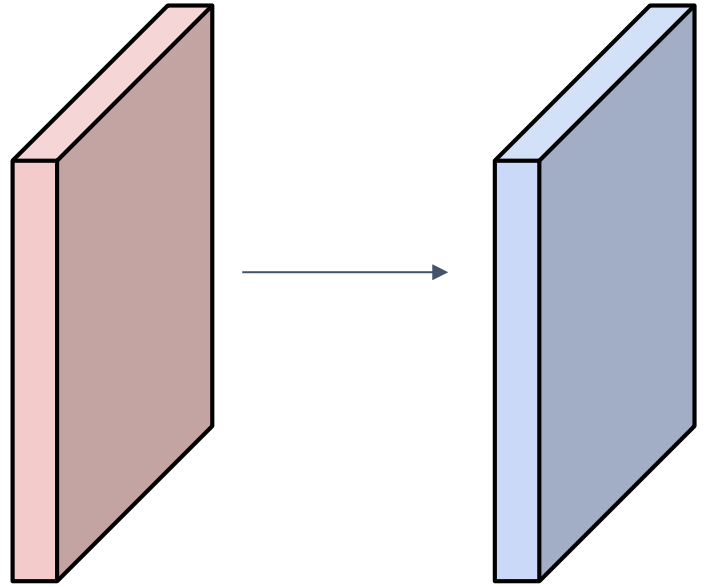
Output volume size:

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

10 x 32 x 32

$$\frac{(W - K + 2p)}{S} + 1$$

Convolution Example



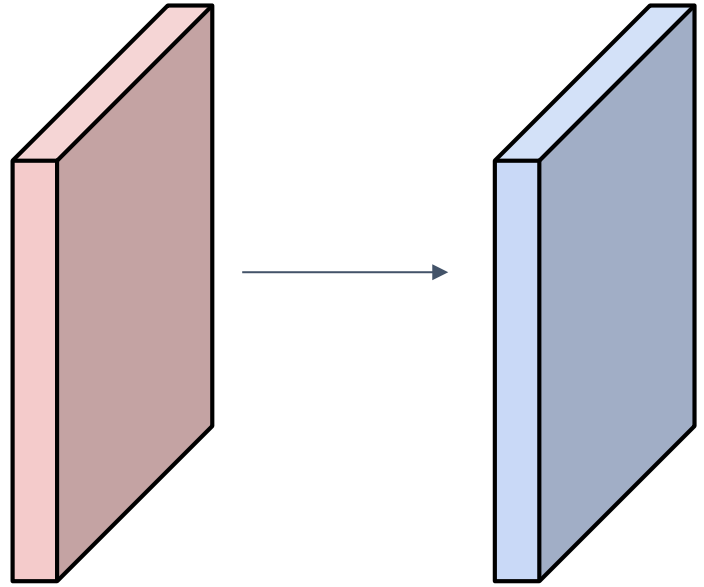
Input volume: 3 x 32 x 32

10 3x5x5 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

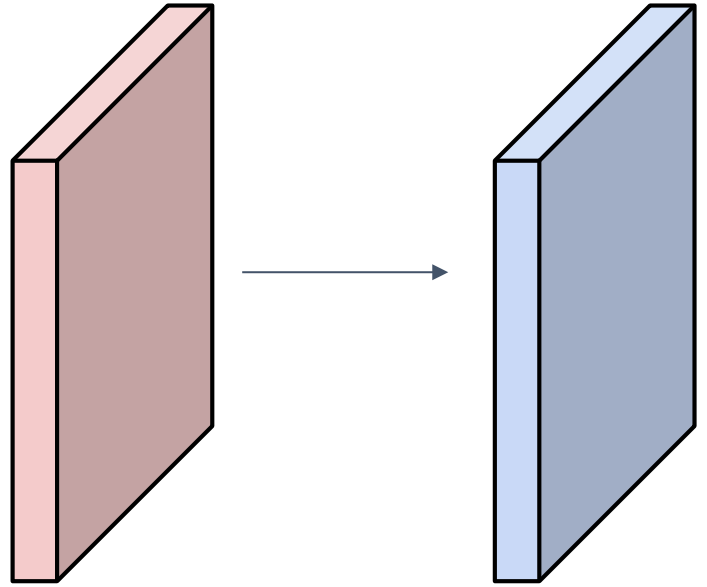
Output volume size: 10 x 32 x 32

Number of learnable parameters: **760**

Parameters per filter: 3*5*5 + 1 (for bias) = 76

10 filters, so total is 10 * 76 = **760**

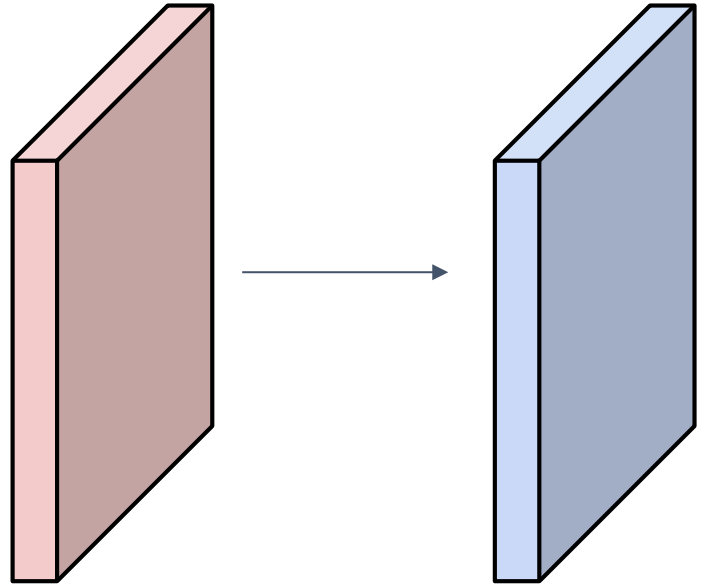
Convolution Example



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 **3x5x5** filters with stride 1, pad 2

Output volume size: **10 x 32 x 32**

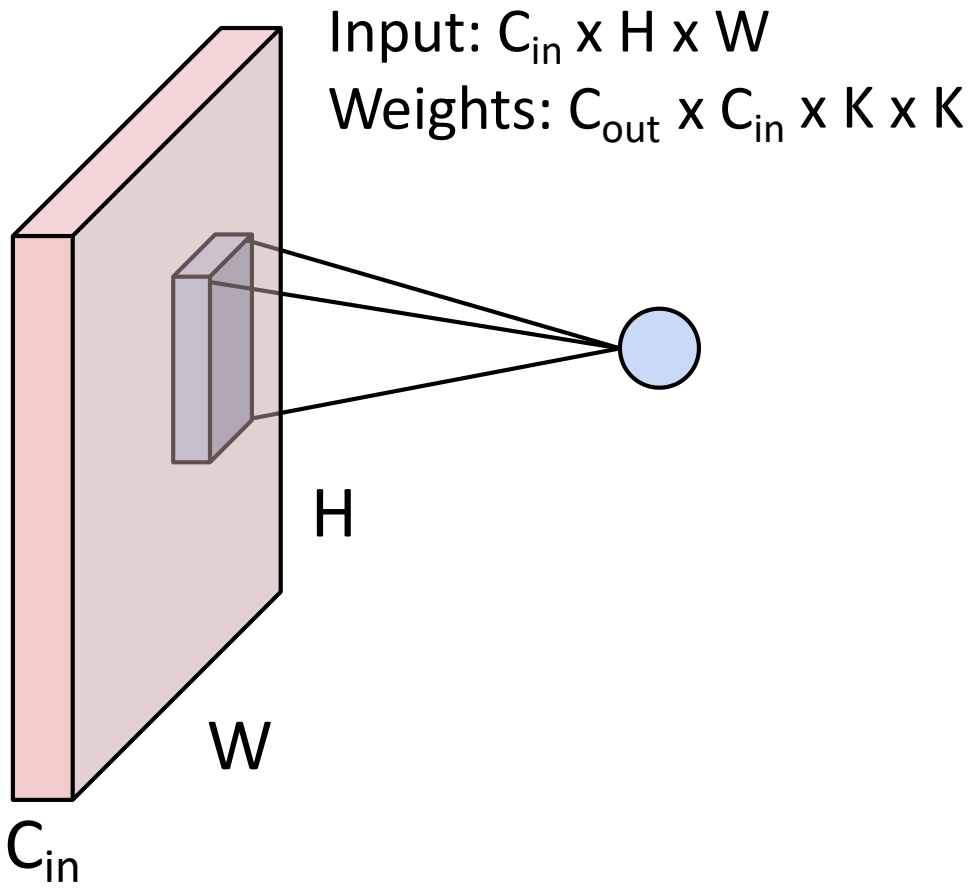
Number of learnable parameters: 760

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**

Other types of convolution

So far: 2D Convolution

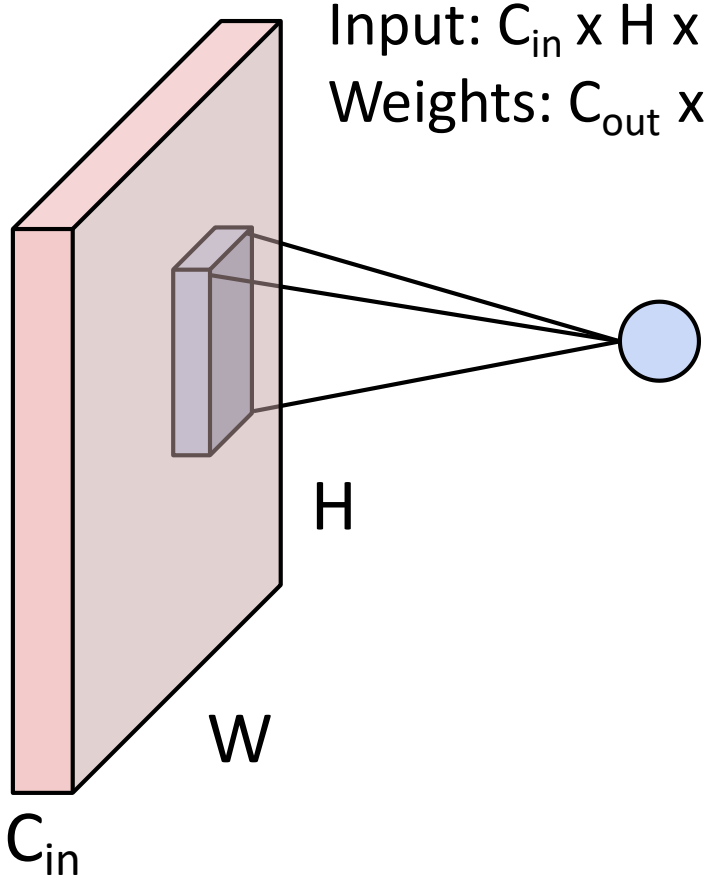


Other types of convolution

So far: 2D Convolution

Input: C_{in} x H x W

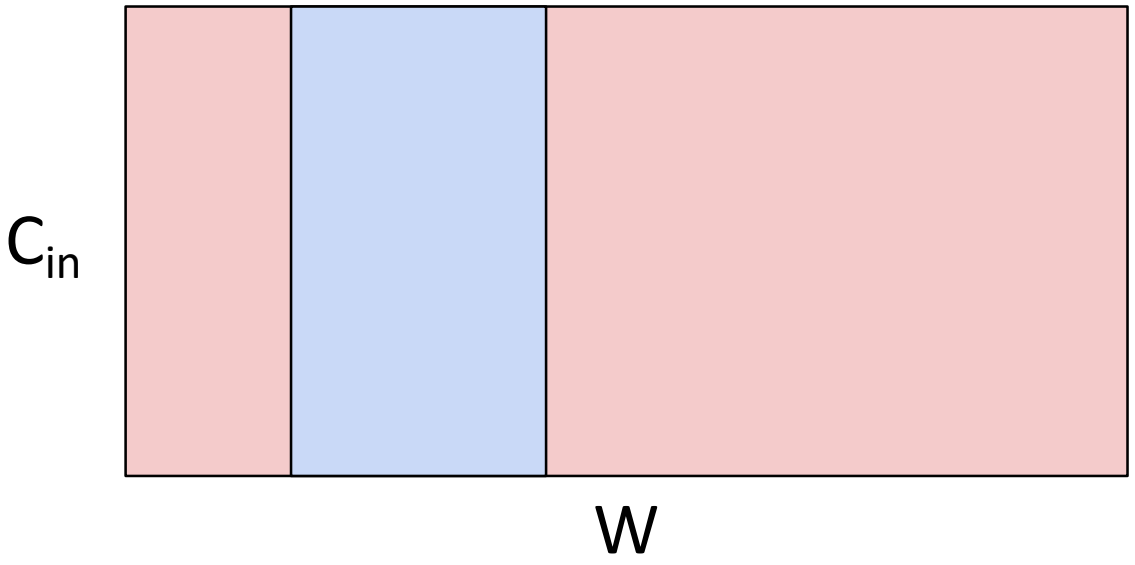
Weights: C_{out} x C_{in} x K x K



1D Convolution

Input: C_{in} x W

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

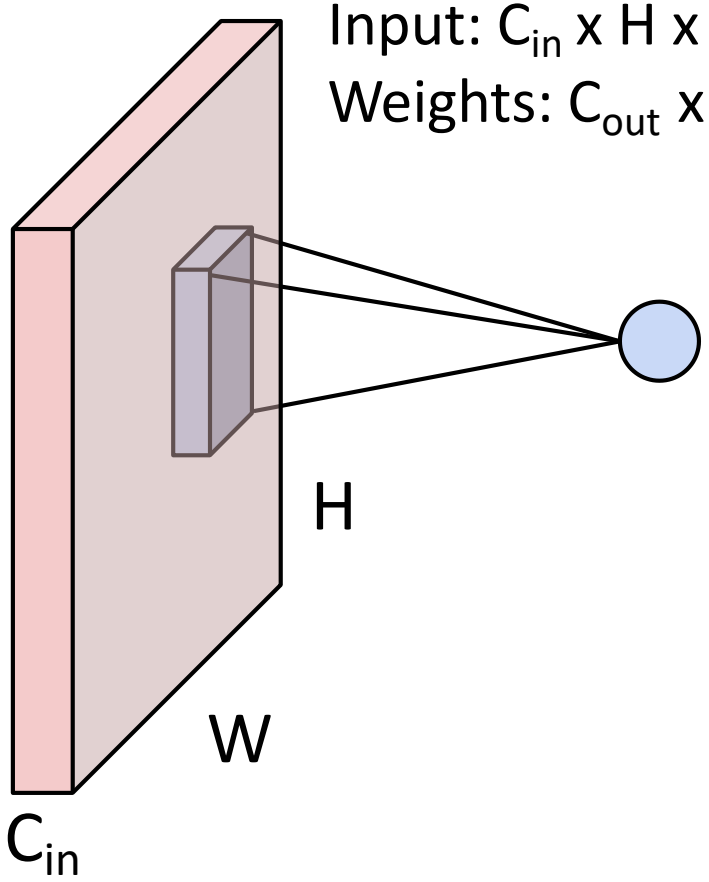


Other types of convolution

So far: 2D Convolution

Input: C_{in} x H x W

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

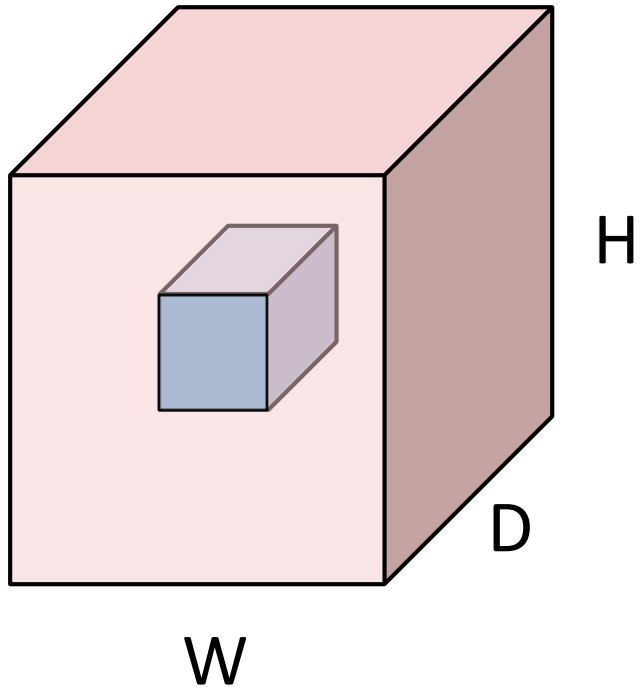


3D Convolution

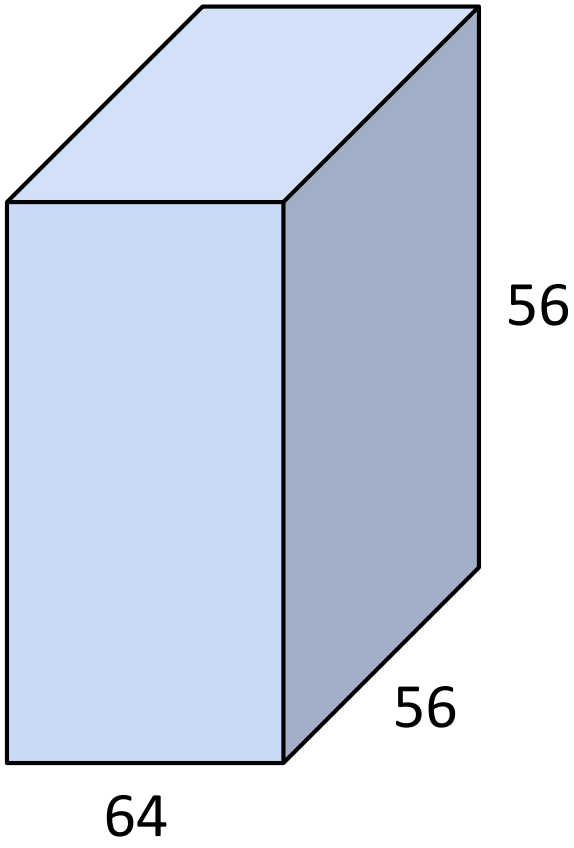
Input: C_{in} x H x W x D

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

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

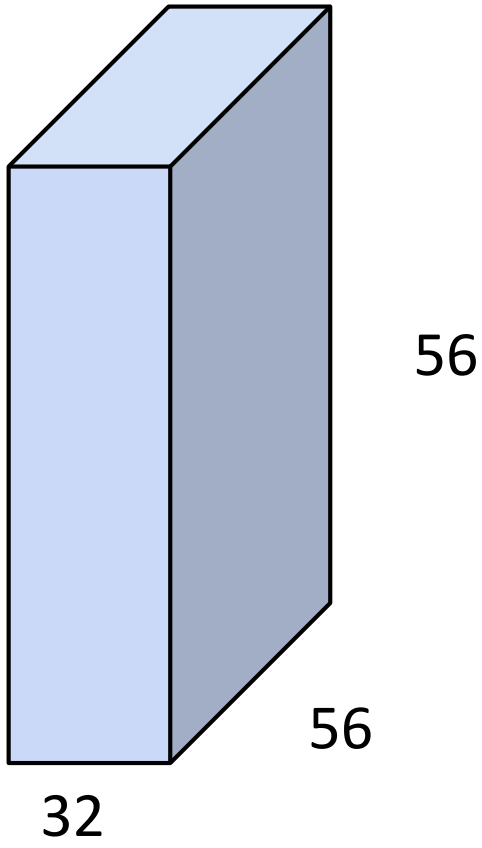


Example: 1x1 Convolution



1x1 CONV
with 32 filters

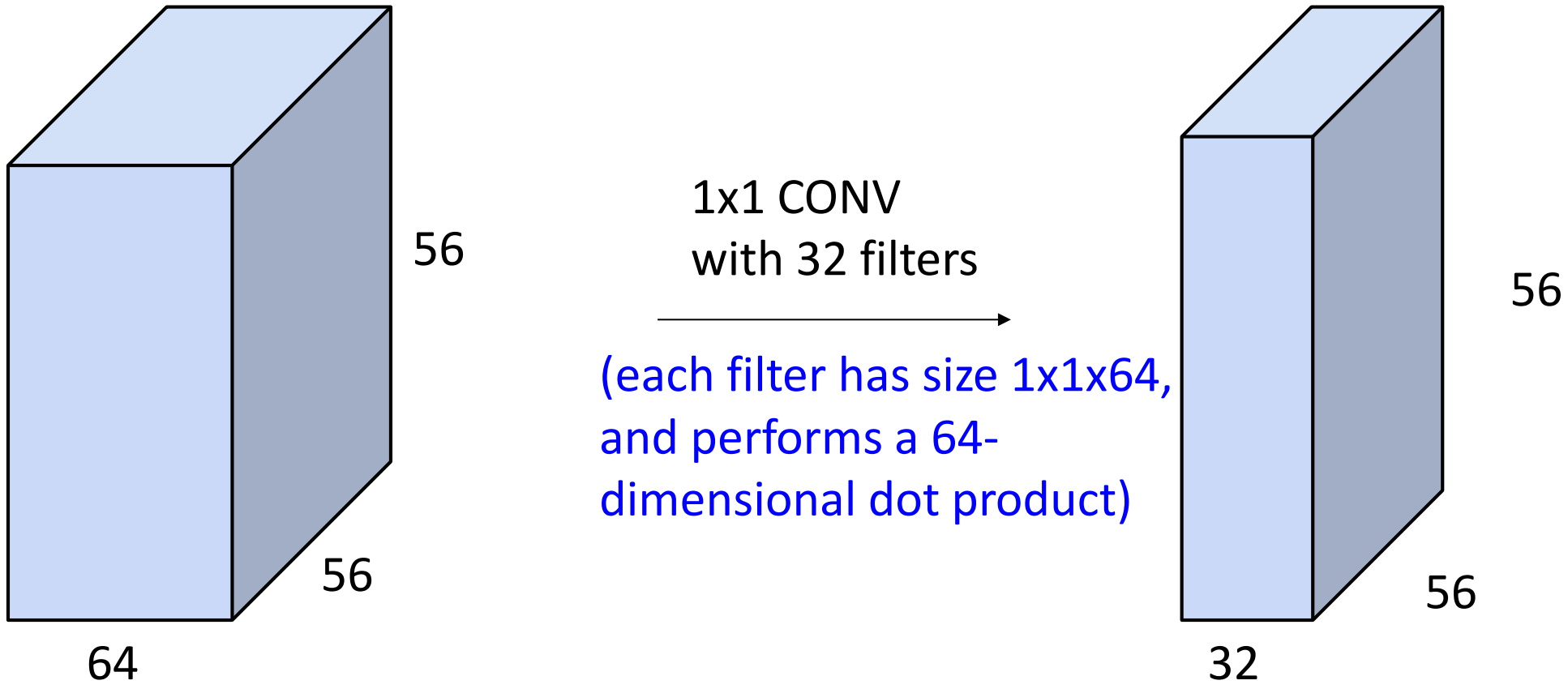
(each filter has size 1x1x64, and performs a 64-dimensional dot product)



Stacking 1x1 conv layers gives MLP operating on each input position

Lin et al, “Network in Network”, ICLR 2014

Example: 1x1 Convolution



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:

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

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

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:

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

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]

Conv1d

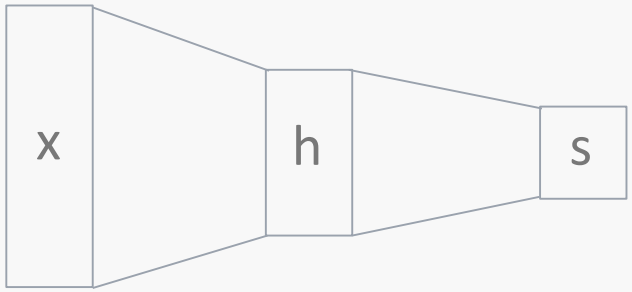
CLASS `torch.nn.Conv1d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

Conv3d

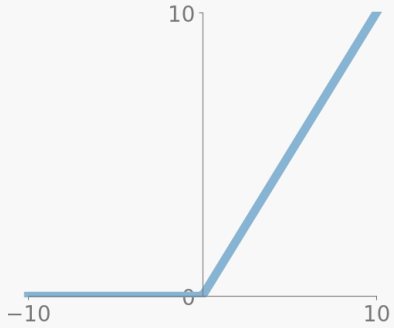
CLASS `torch.nn.Conv3d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros'*`)` [SOURCE]

Components of a Convolutional Network

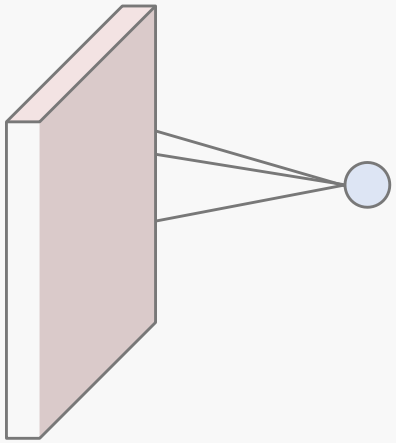
Fully-Connected Layers



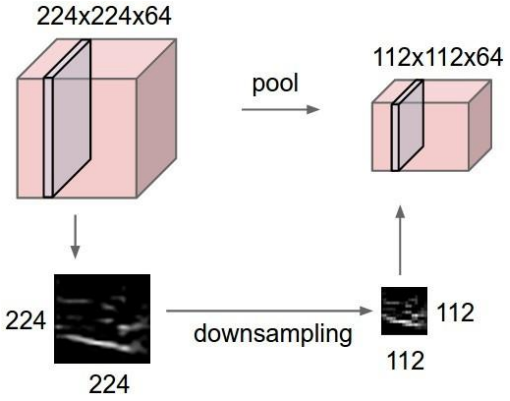
Activation Function



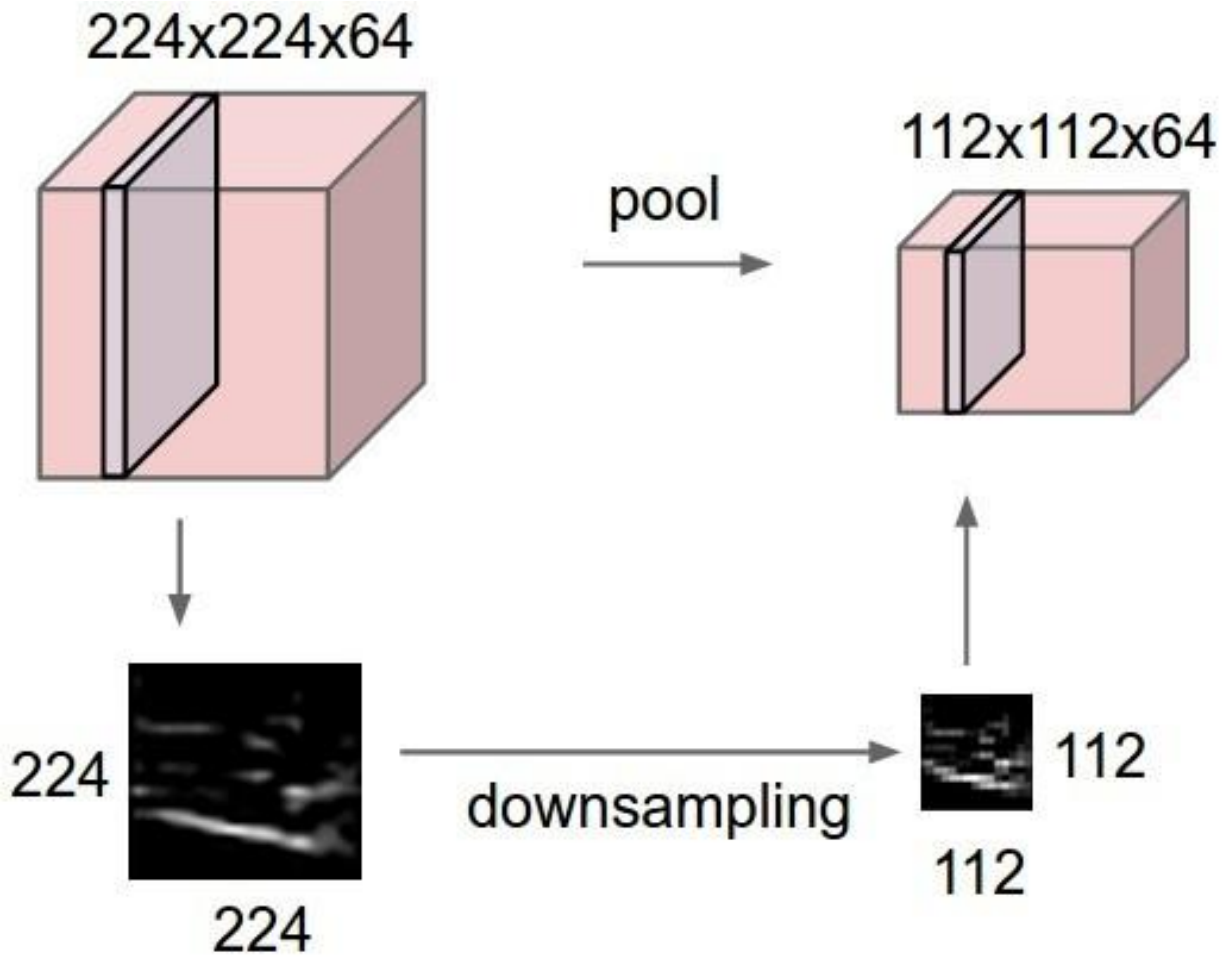
Convolution Layers



Pooling Layers

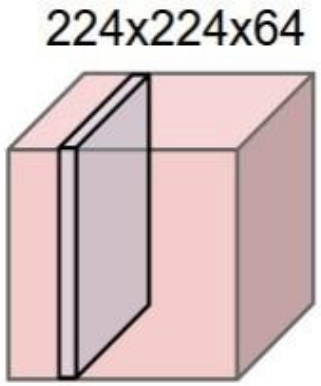


Pooling Layers: Another way to downsample



Hyperparameters:
Kernel Size
Stride
Pooling function

Max Pooling



Single depth slice

x

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

y

Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Advantage of using pooling instead of strided-convolution:

- Introduce **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

$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

Learnable parameters: None!

Common settings:

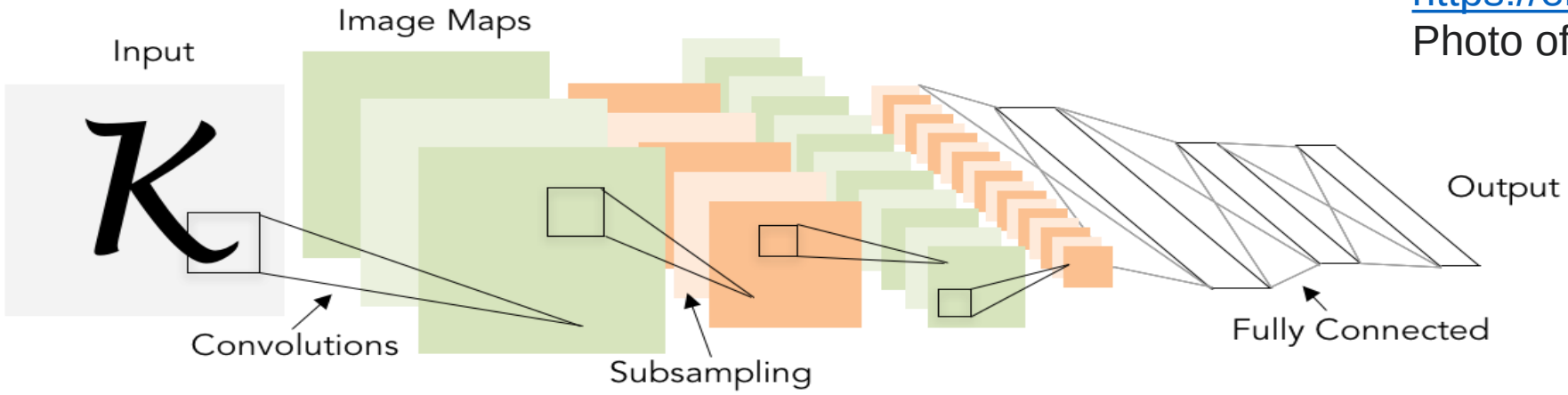
max, K = 2, S = 2

max, K = 3, S = 2 (AlexNet)

Convolutional Networks

https://en.wikipedia.org/wiki/LeNet
Photo of Yann LeCun in 2018

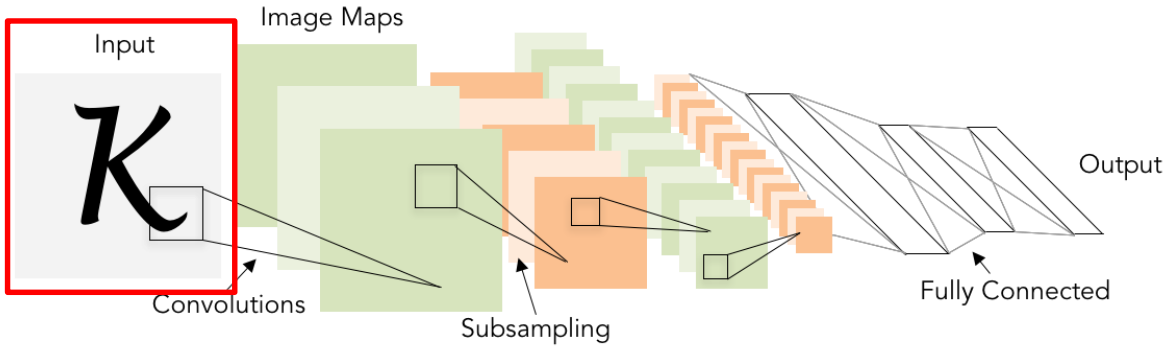
Example: LeNet-5



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	0

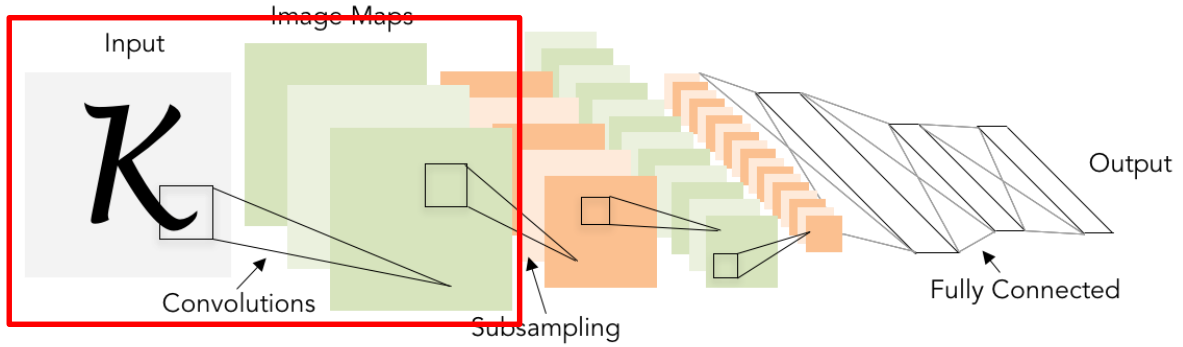


- Where the out put size is the Activation Volume Dimensions.
- Weight (kernel) (Filter) size is the number of trainable parameters.

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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0



Conv Output size

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

Pooling Output size

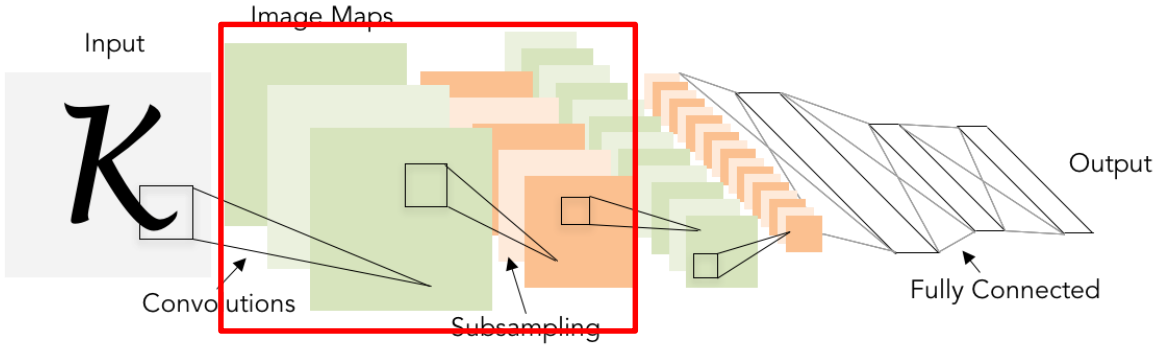
$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0



Conv Output size

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

Pooling Output size

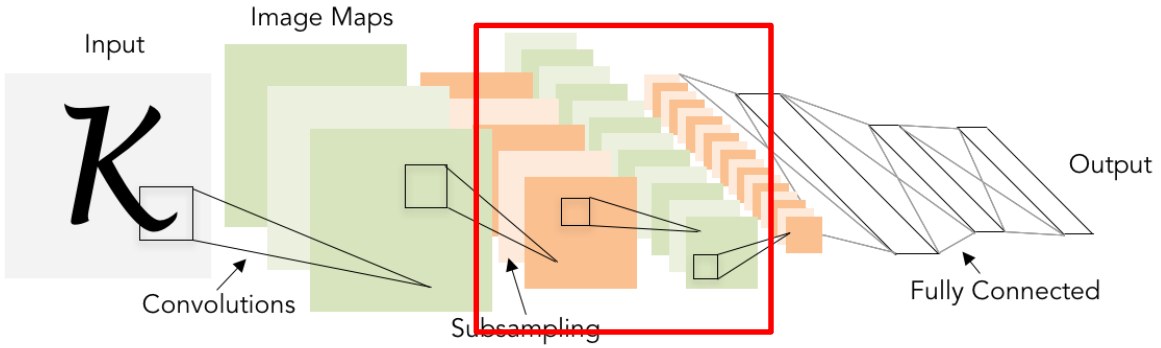
$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

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+1)
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+1)
ReLU	50 x 14 x 14	



Conv Output size

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

Pooling Output size

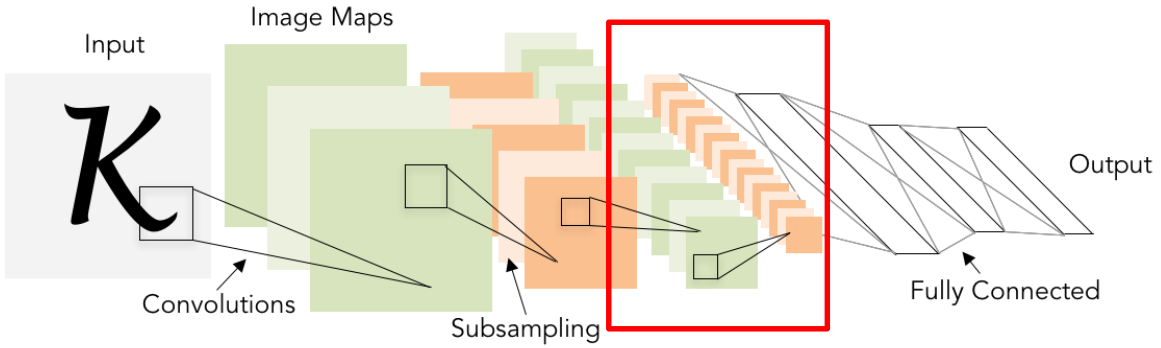
$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

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

Example: LeNet-5

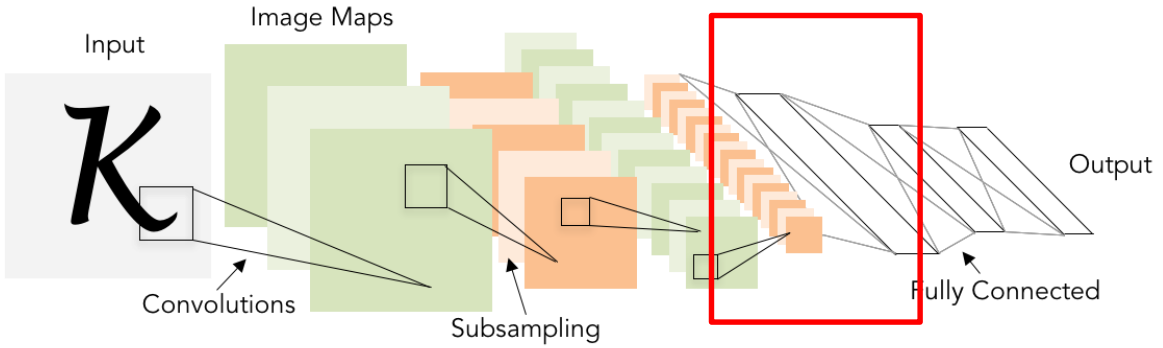
Layer	Output Size	Weight Size
Input	1 x 28 x 28	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x (20 x 5 x 5+1)
ReLU	50 x 14 x 14	0
MaxPool(K=2, S=2)	50 x 7 x 7	0



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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x (20 x 5 x 5+1)
ReLU	50 x 14 x 14	0
MaxPool(K=2, S=2)	50 x 7 x 7	0
Flatten	2450	0



Conv Output size

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

Pooling Output size

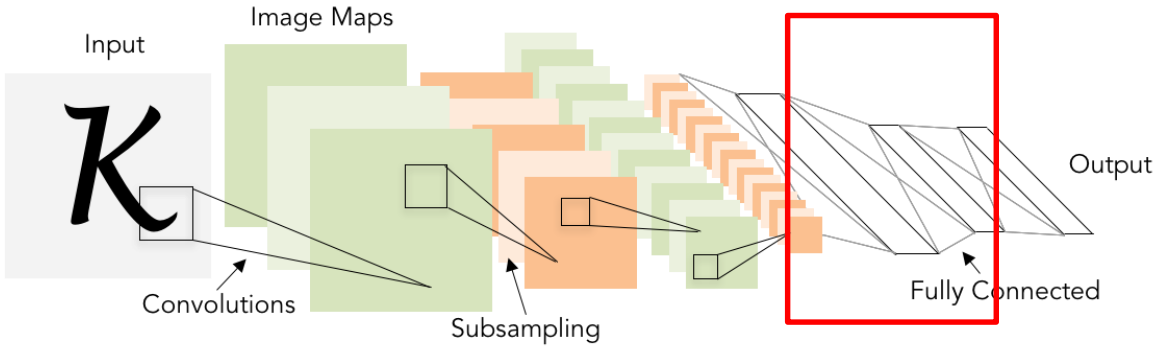
$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x (20 x 5 x 5 +1)
ReLU	50 x 14 x 14	0
MaxPool(K=2, S=2)	50 x 7 x 7	0
Flatten	2450	0
Linear (2450 -> 500)	500	(2450 x 500) + 500
ReLU	500	0



Conv Output size

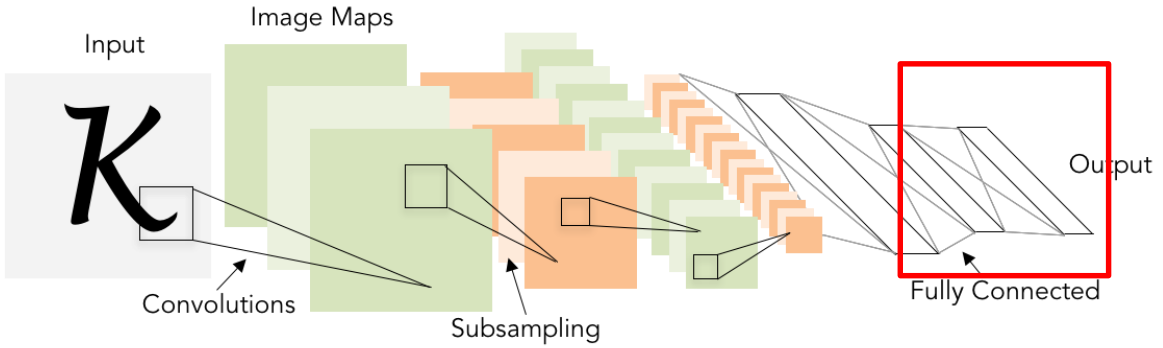
$$H' = \frac{(H - K + 2p)}{S} + 1 \qquad H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1 \qquad W' = \frac{(W - K)}{S} + 1$$

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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5+1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x (20 x 5 x 5 +1)
ReLU	50 x 14 x 14	0
MaxPool(K=2, S=2)	50 x 7 x 7	0
Flatten	2450	0
Linear (2450 -> 500)	500	(2450 x 500) + 500
ReLU	500	0
Linear (500 -> 10)	10	(500 x 10) + 10



Conv Output size

$$H' = \frac{(H - K + 2p)}{S} + 1$$

$$W' = \frac{(W - K + 2p)}{S} + 1$$

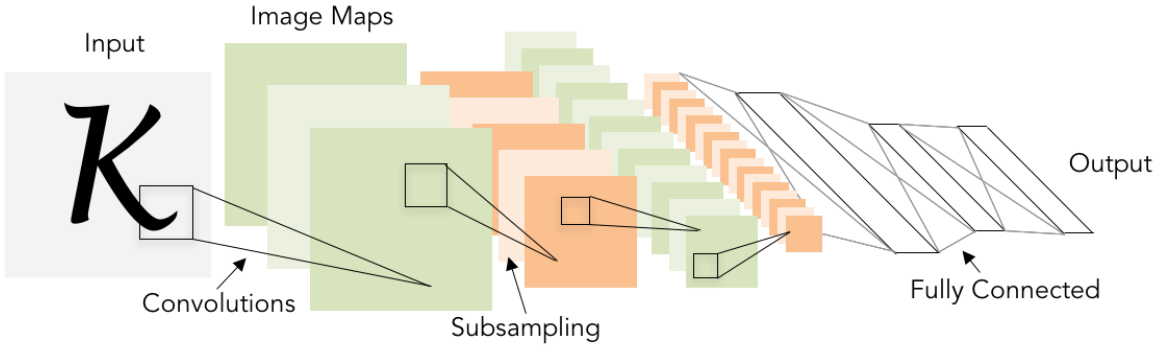
$$H' = \frac{(H - K)}{S} + 1$$

$$W' = \frac{(W - K)}{S} + 1$$

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	0
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x (1 x 5 x 5 + 1)
ReLU	20 x 28 x 28	0
MaxPool(K=2, S=2)	20 x 14 x 14	0
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x (20 x 5 x 5+1)
ReLU	50 x 14 x 14	0
MaxPool(K=2, S=2)	50 x 7 x 7	0
Flatten	2450	0
Linear (2450 -> 500)	500	(2450 x 500) + 500
ReLU	500	0
Linear (500 -> 10)	10	(500 x 10) + 10



As we go through the network:

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

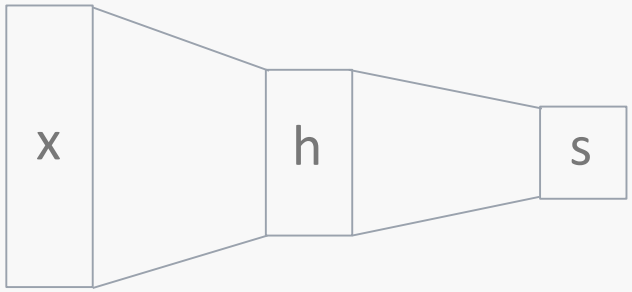
Number of channels **increases**
(total "volume" is preserved!)

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

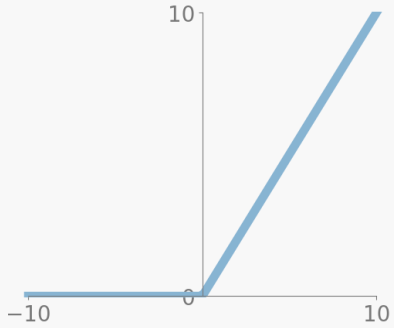
Problem: Deep Networks very hard to train!

Components of a Convolutional Network

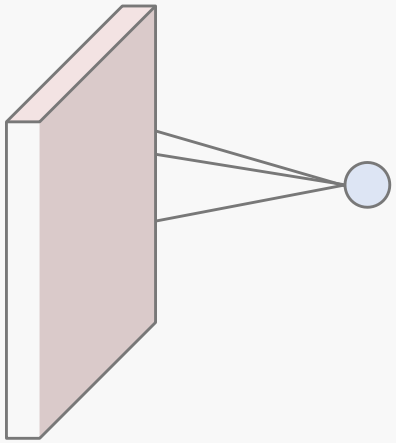
Fully-Connected Layers



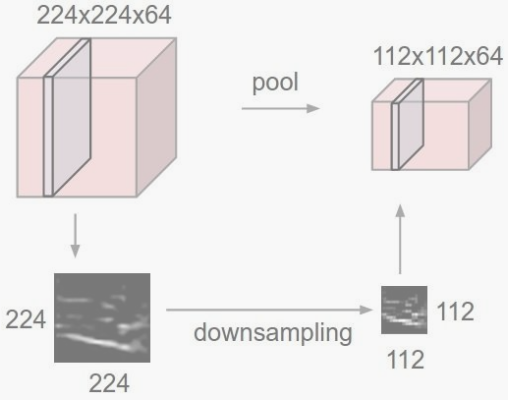
Activation Function



Convolution Layers



Pooling Layers



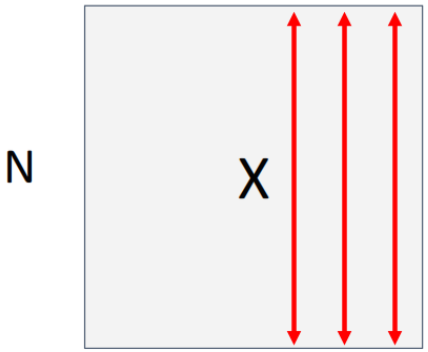
Normalization

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

Batch Normalization

- Idea: “Normalize” the outputs of a layer so they have zero mean and unit variance
- Why? Improves optimization

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

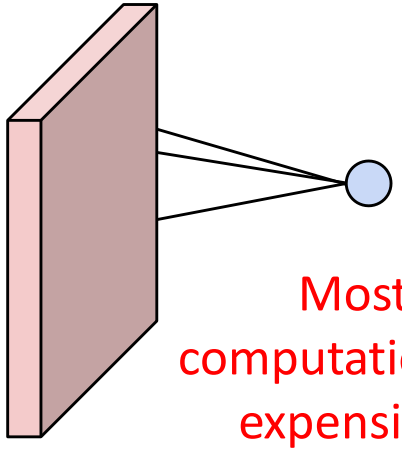
$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel std, shape is D

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

Ioffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, ICML 2015

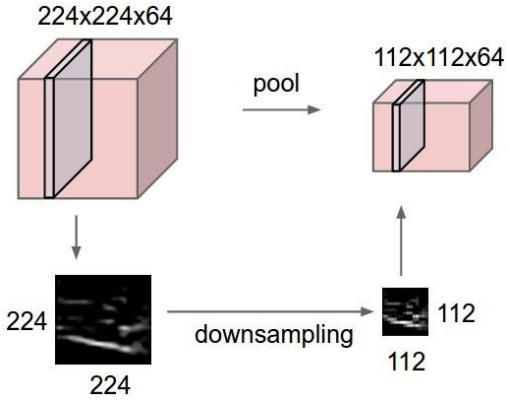
Components of a Convolutional Network

Convolution Layers

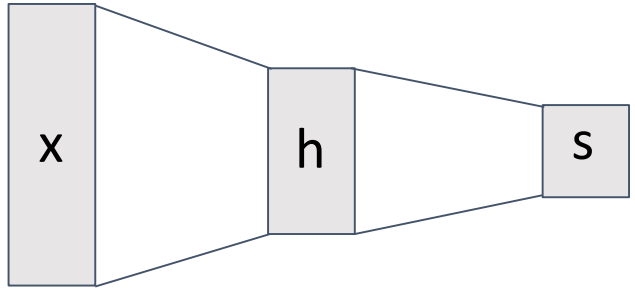


Most computationally expensive!

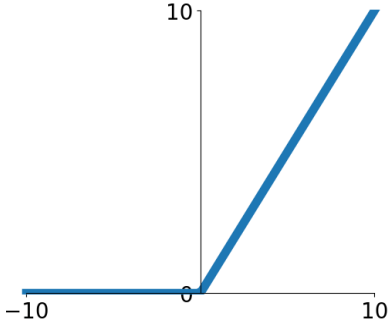
Pooling Layers



Fully-Connected Layers



Activation Function

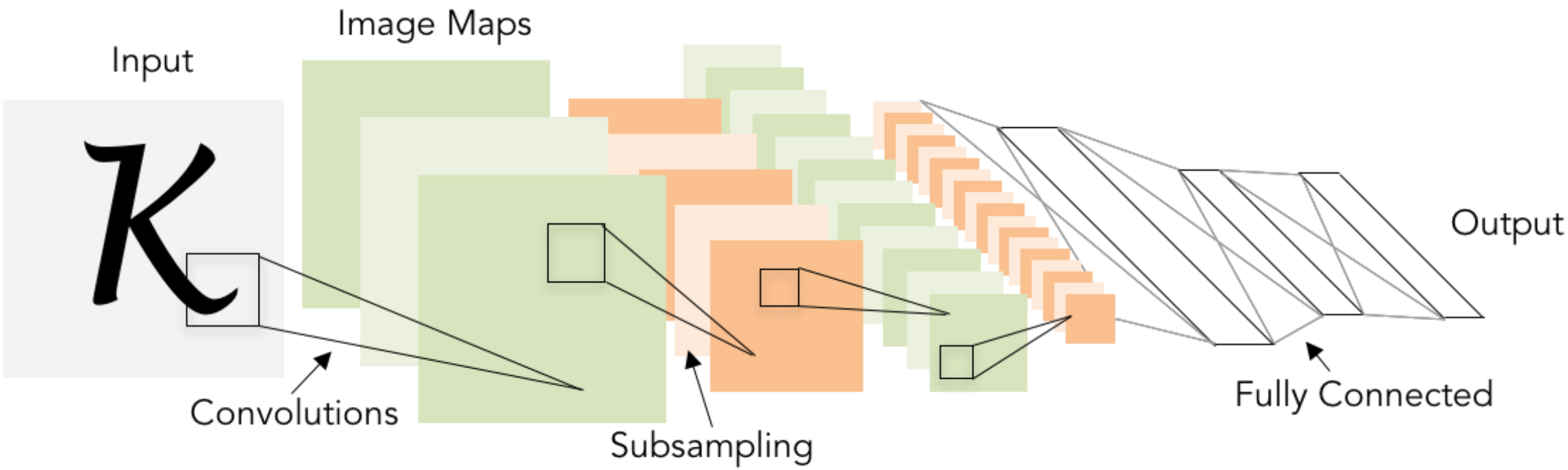


Normalization

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

Summary: Components of a Convolutional Network

Problem: What is the right way to combine all these components?



References

- https://sites.google.com/site/deeplearningsummerschool/
- http://www.deeplearningbook.org/
- http://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf
- https://cs231n.github.io/convolutional-networks/
- S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. Int. Conf. Machine Learning (ICML), 2015.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

Next: CNN Architectures