Lecture 5:
Image Classification with CNNs

Administrative: Assignment 1

Assignment 1 Due **Wednesday 4/23** at 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features

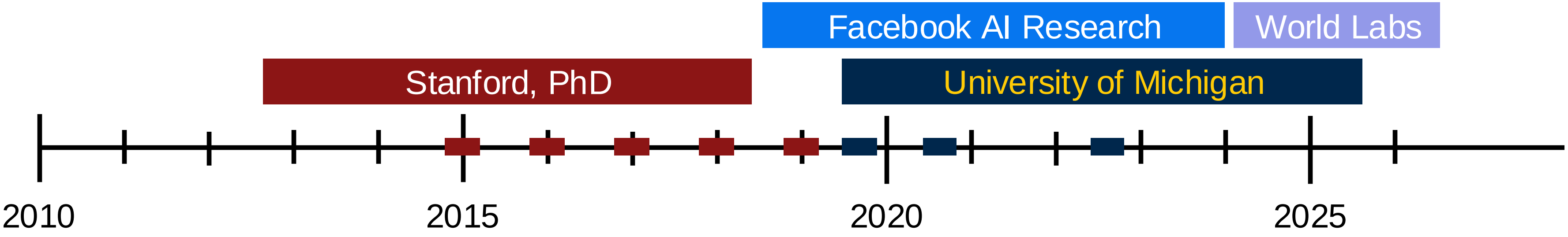
Administrative: Project Proposal

Due Fri 4/25

TA expertise is posted on the webpage.

http://cs231n.stanford.edu/office_hours.html

Hi I'm Justin



I taught CS231N at Stanford from 2015-2018

I taught the same course at Michigan 2019-2022

CS231n: Deep Learning for Computer Vision

- Deep Learning Basics (Lecture 2 – 4)
- Perceiving and Understanding the Visual World (Lecture 5 – 12)
- Generative and Interactive Visual Intelligence (Lecture 13 – 16)
- Human-Centered Applications and Implications (Lecture 17 – 18)

Recap: Image Classification with Linear Classifier

Image

Array of **32x32x3** numbers (3072 numbers total)

$\rightarrow$ f(**x**,**W**) $\rightarrow$ **10** numbers giving class scores

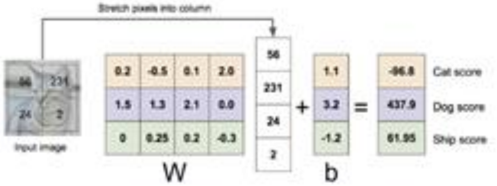
$\uparrow$

W
parameters or weights

$$f(x,W) = Wx + b$$

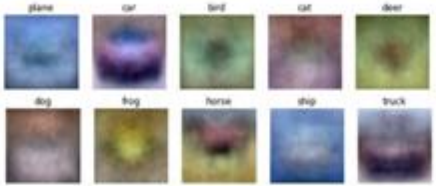
Algebraic Viewpoint

$$f(x,W) = Wx$$



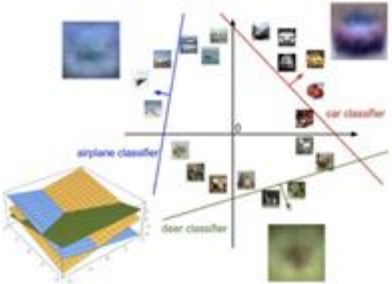
Visual Viewpoint

One template per class



Geometric Viewpoint

Hyperplanes cutting up space

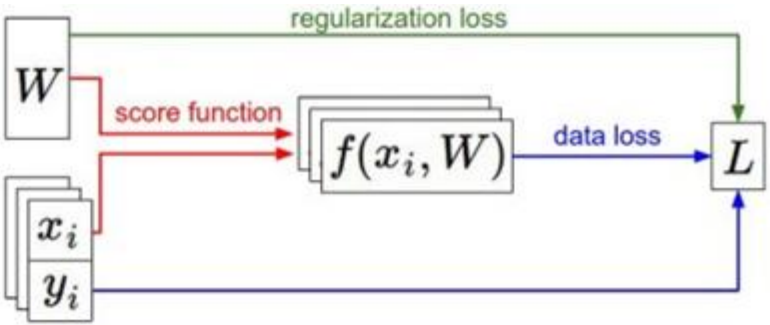


Recap: Loss Function

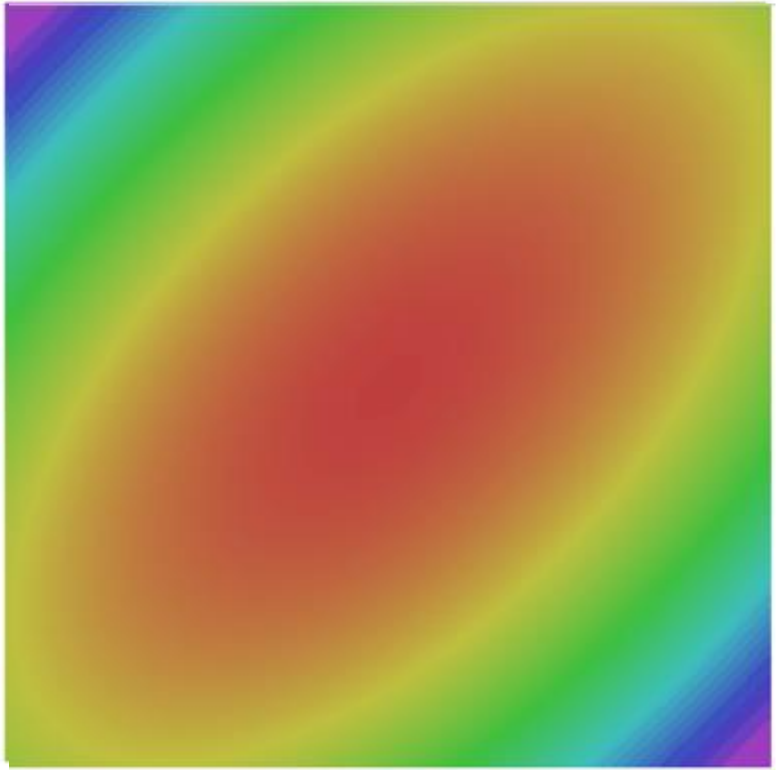
- We have some dataset of (x,y)
- We have a score function: $s = f(x; W) = Wx$
- We have a loss function:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ Softmax

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



Recap: Optimization



SGD

SGD+Momentum

RMSProp

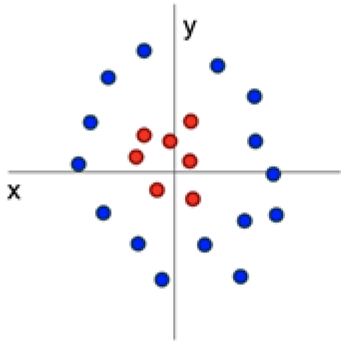
Adam

Problem: Linear Classifiers are not very powerful

Visual Viewpoint

Linear classifiers learn one template per class

Geometric Viewpoint

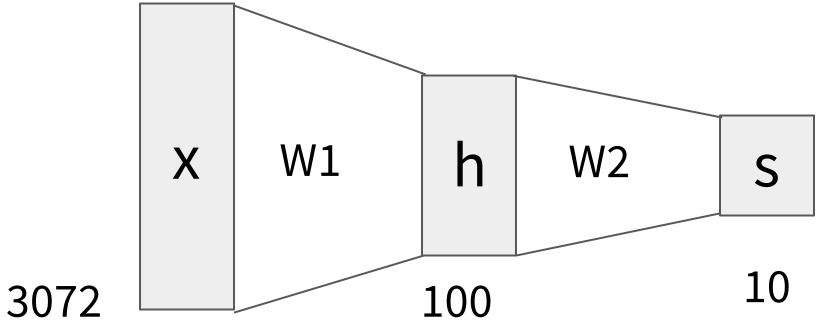


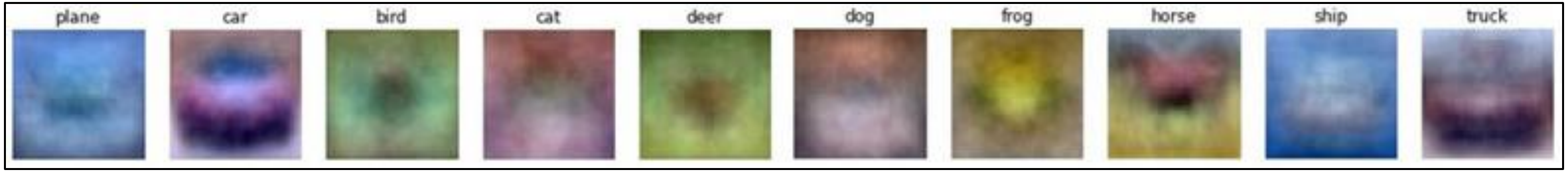
Linear classifiers can only draw linear decision boundaries

Last time: Neural Networks

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$

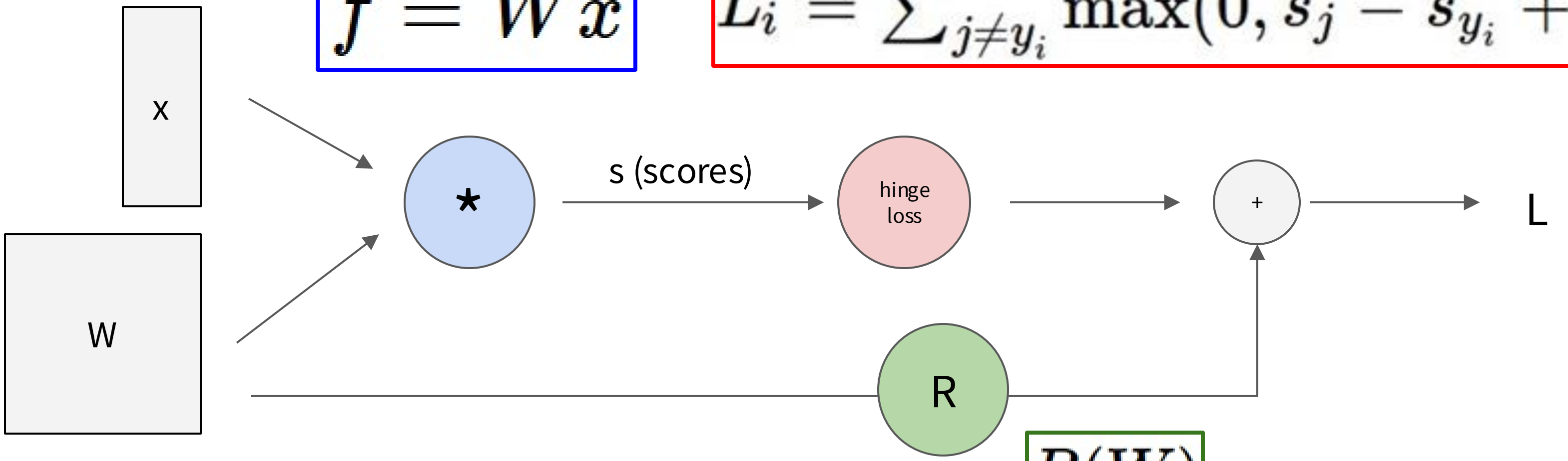




Last time: Computation Graph

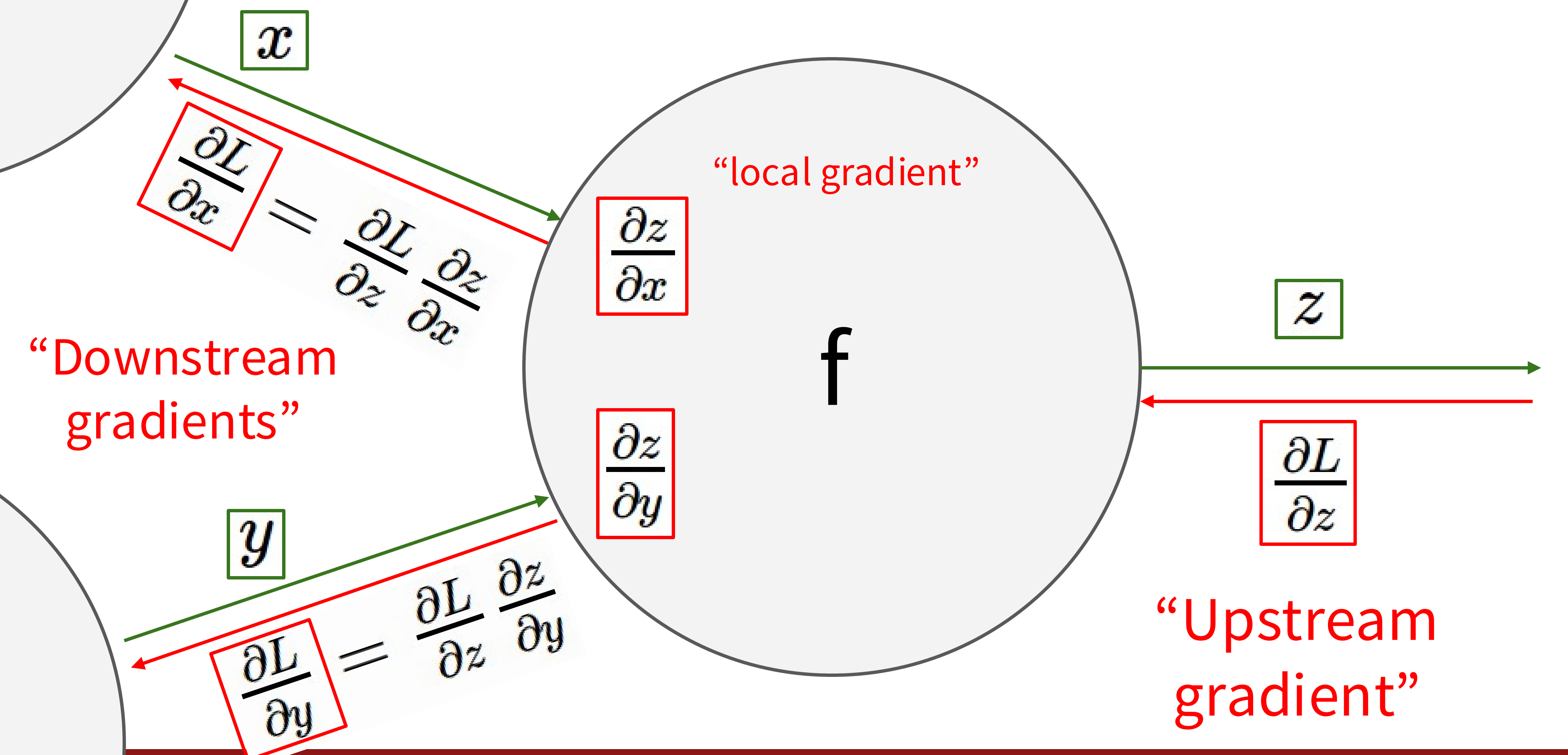
$$f = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

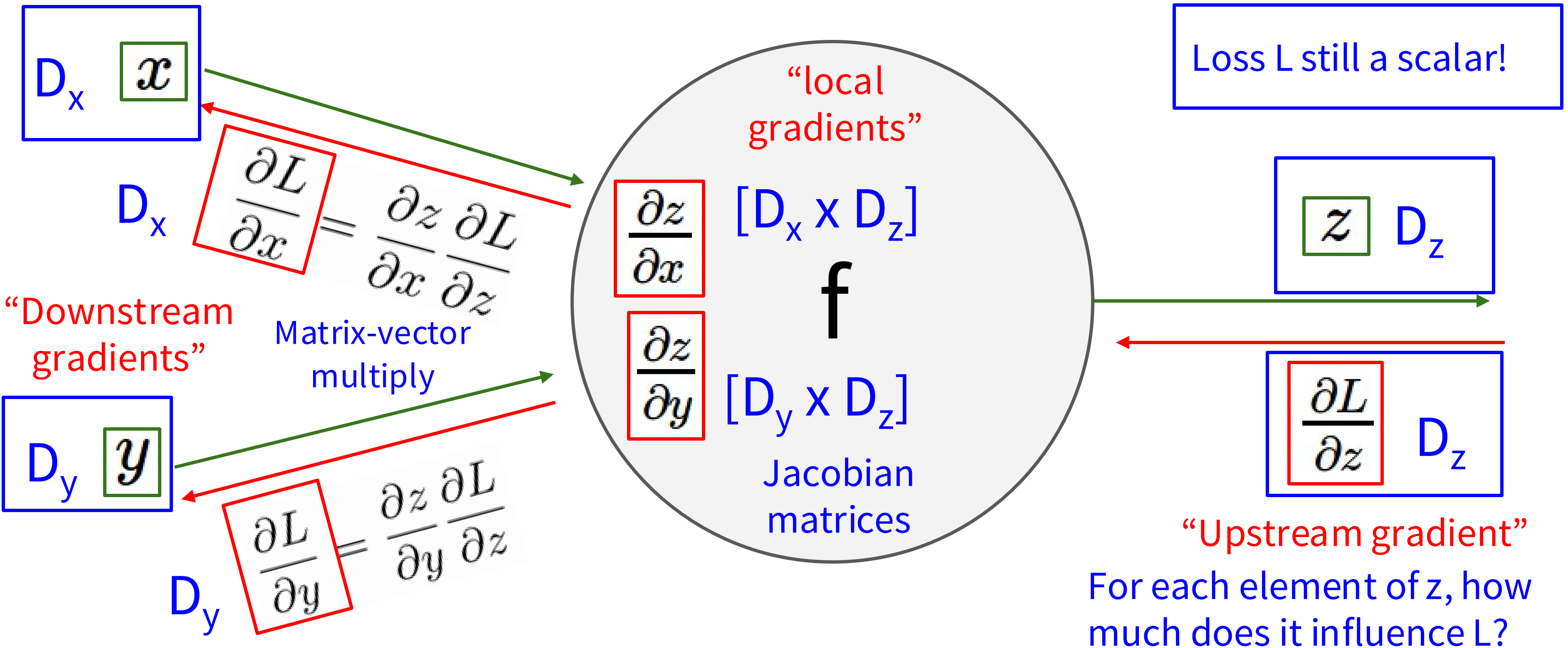


$$R(W)$$

Last time: Backpropagation

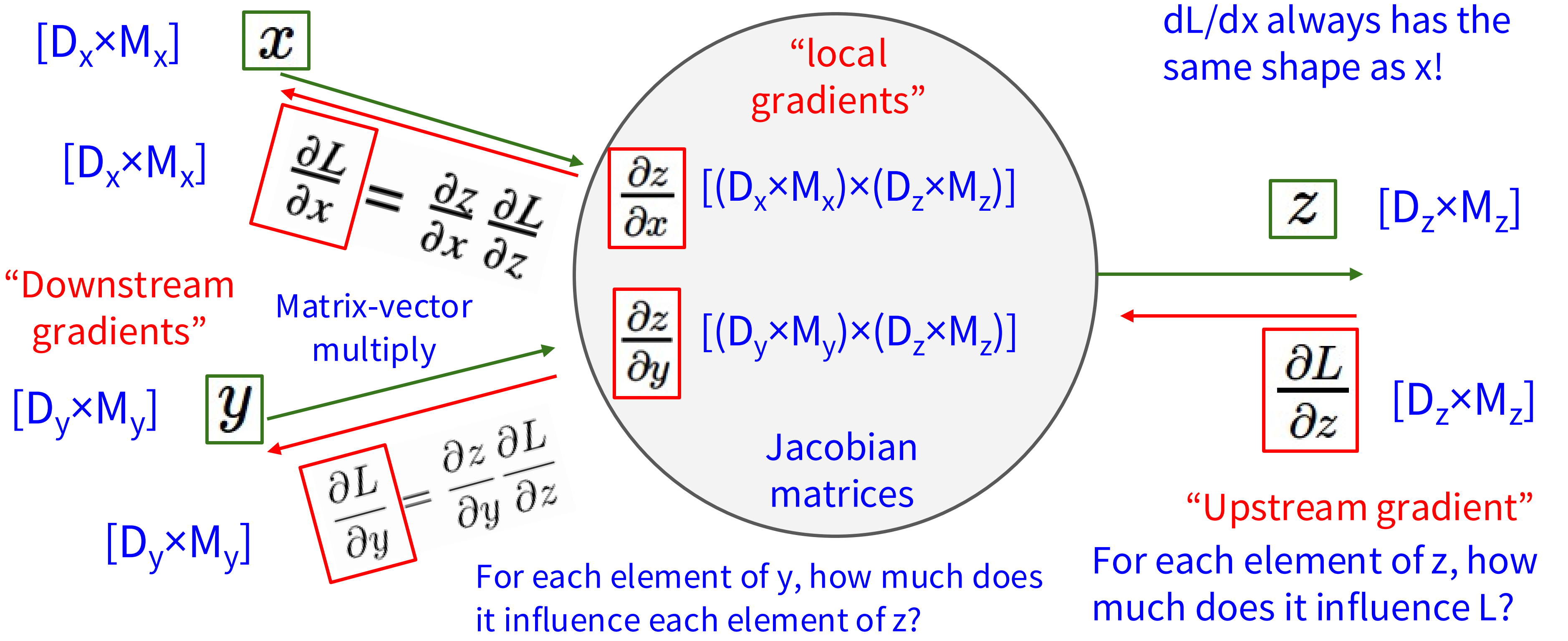


Backprop with Vectors



Backprop with Matrices (or Tensors)

Loss L still a scalar!



CS231n: Deep Learning for Computer Vision

- Deep Learning Basics (Lecture 2 – 4)
- Perceiving and Understanding the Visual World (Lecture 5 – 12)
- Generative and Interactive Visual Intelligence (Lecture 13 – 16)
- Human-Centered Applications and Implications (Lecture 17 – 18)

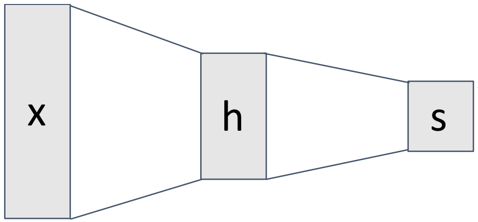
CS231n: Deep Learning for Computer Vision

- Deep Learning Basics (Lecture 2 – 4)
- Perceiving and Understanding the Visual World (Lecture 5 – 12)
- Generative and Interactive Visual Intelligence (Lecture 13 – 16)
- Human-Centered Applications and Implications (Lecture 17 – 18)

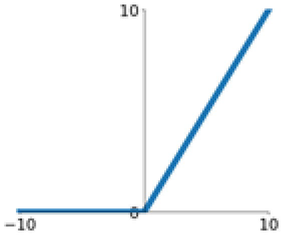
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer



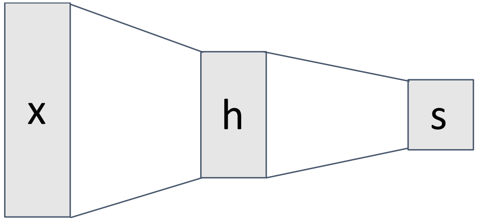
Activation Function



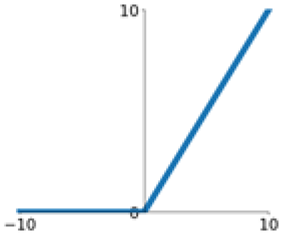
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

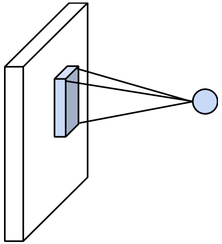


Activation Function



Today: Image-specific operators

Convolution Layer



Pooling Layer

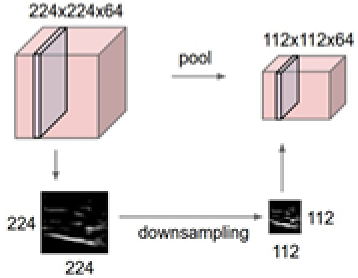


Image Classification: A core task in Computer Vision

(assume given a set of labels)
{dog, cat, truck, plane, ...}

→

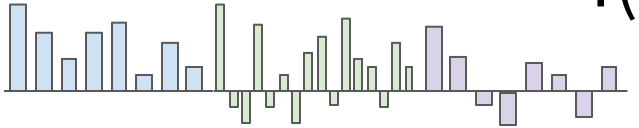
cat
dog
bird
deer
truck

Pixel space

Class scores

$$f(x) = Wx$$

Image features

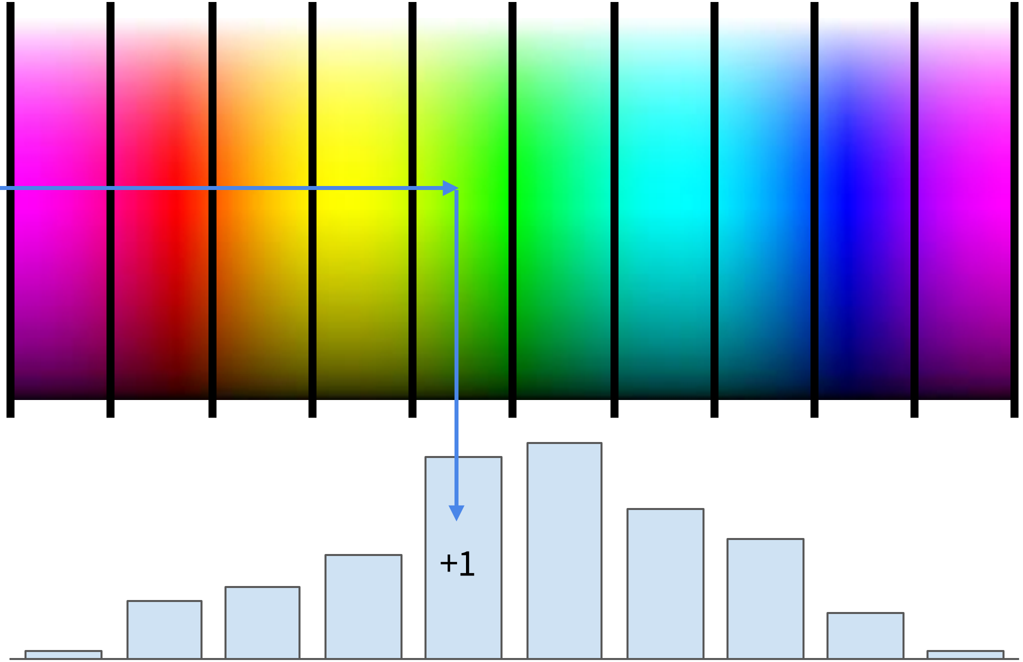


Feature Representation

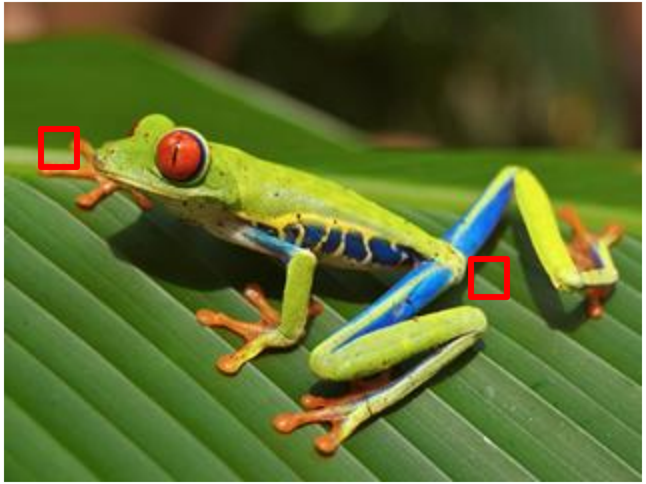
$f(x) = Wx$

Class scores

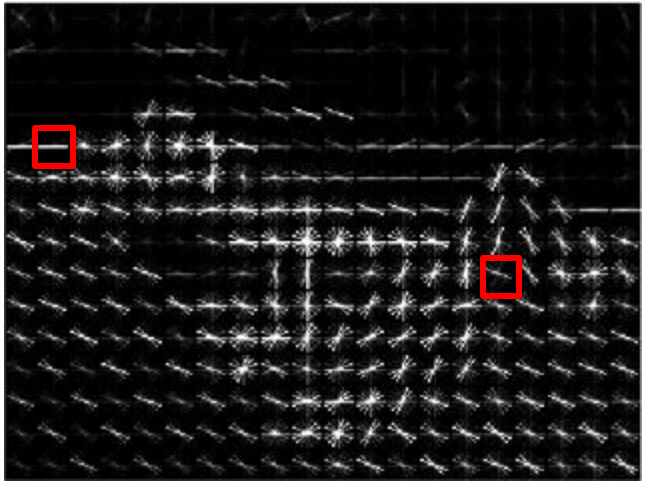
Example: Color Histogram



Example: Histogram of Oriented Gradients (HoG)



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins



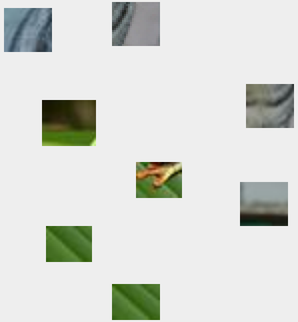
Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Example: Bag of Words

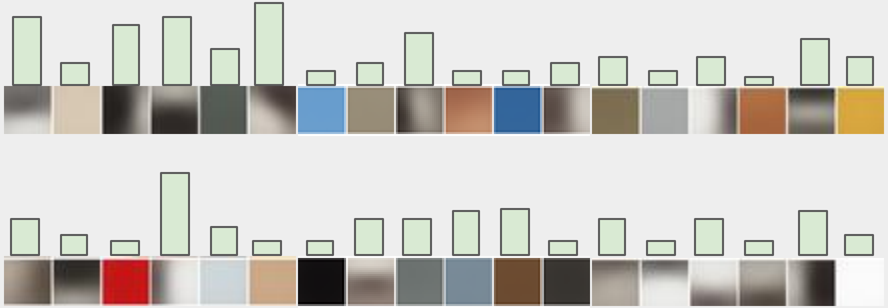
Step 1: Build codebook

Extract random patches



Cluster patches to form "codebook" of "visual words"

Step 2: Encode images



Fei-Fei and Perona, "A bayesian hierarchical model for learning natural scene categories", CVPR 2005

Image Features

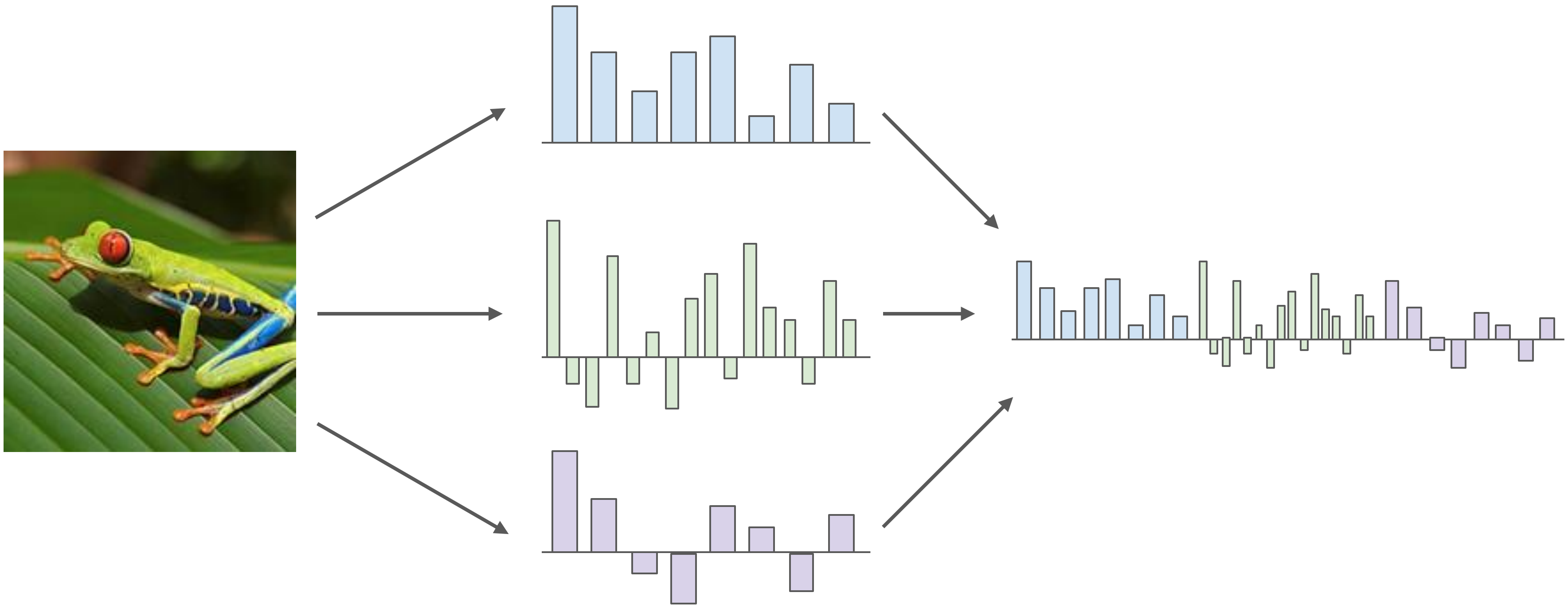
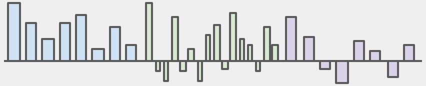


Image features vs. ConvNets

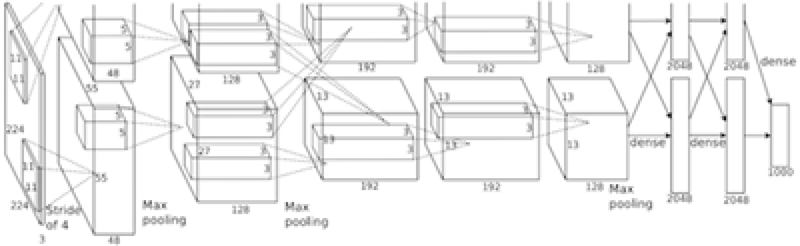
Feature Extraction



f

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.

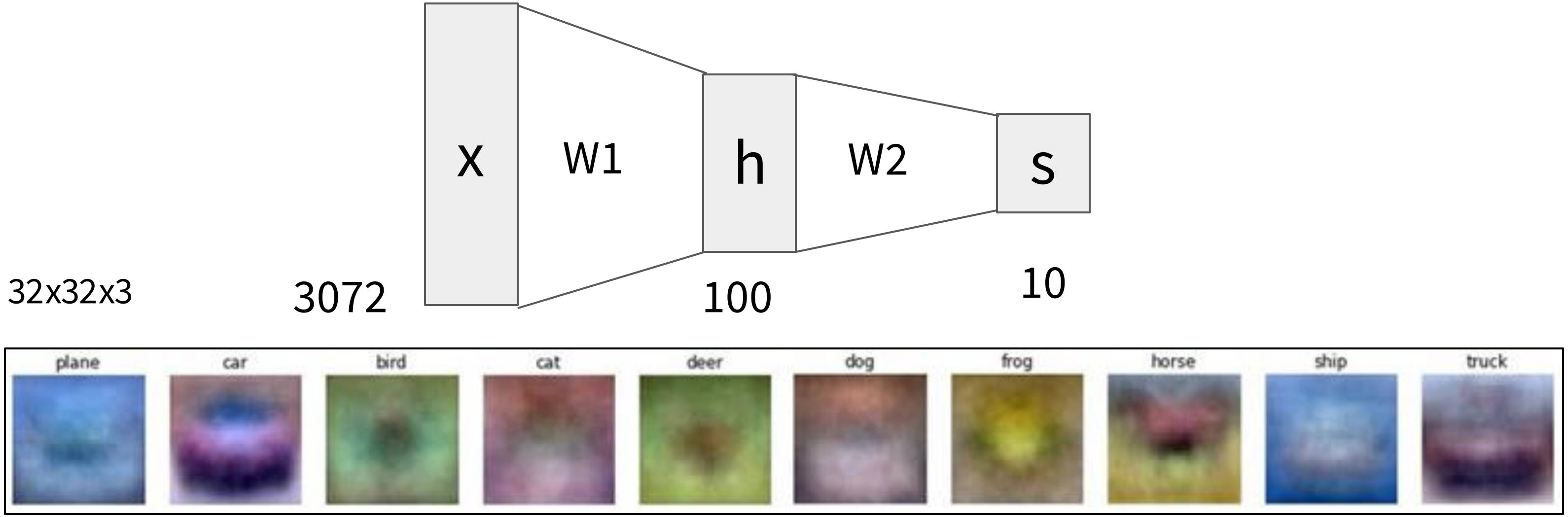
10 numbers giving scores for classes

training

Last Time: Neural Networks

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$



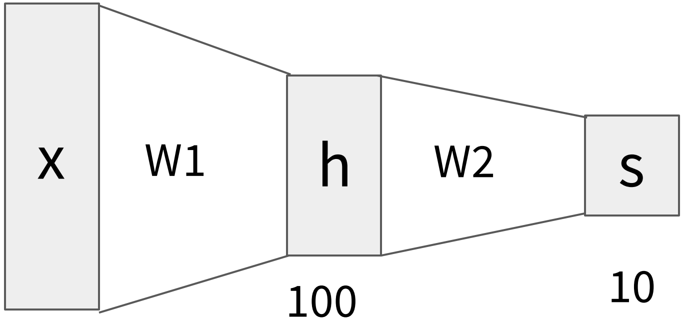
Last Time: Neural Networks

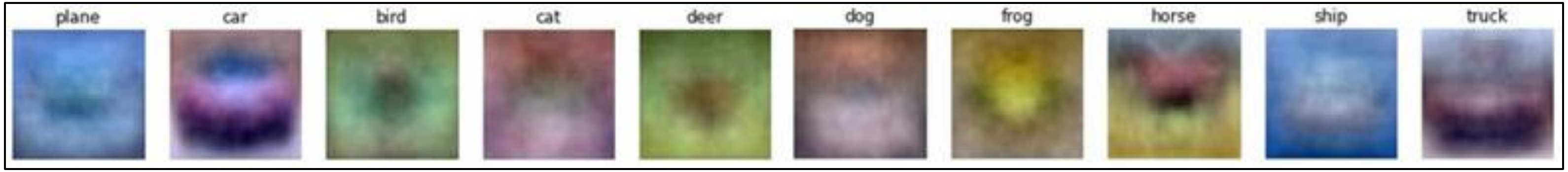
Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Problem: The spatial structure of images is destroyed!

32x32x3 $\boxed{3072}$





Next: Convolutional Neural Networks

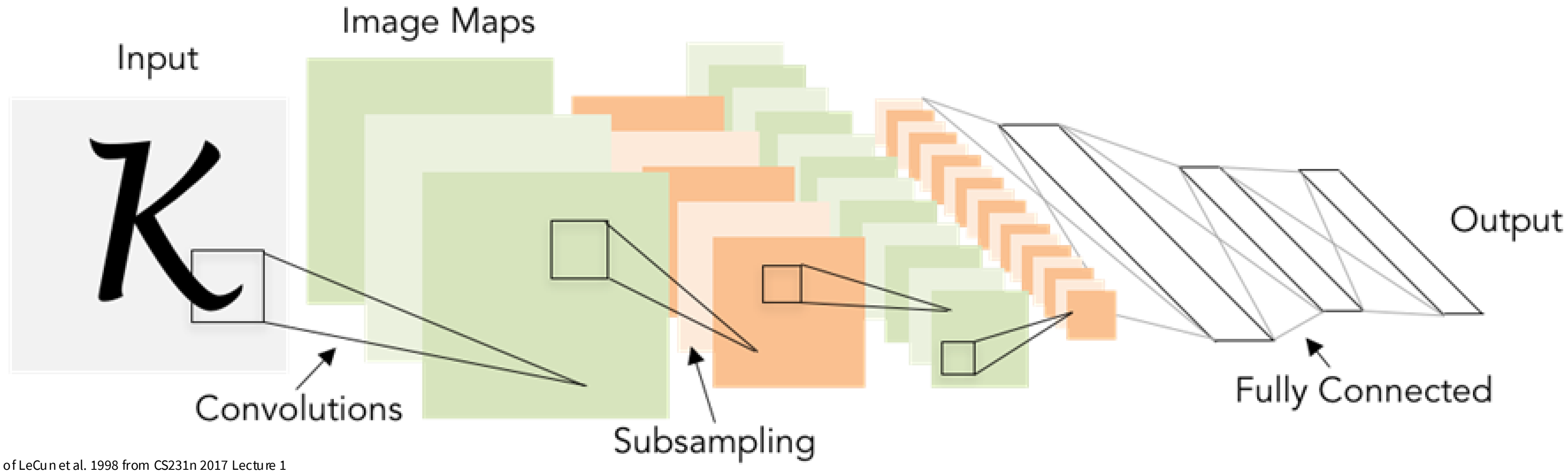


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Next: Convolutional Neural Networks

Convolution and **pooling** operators extract features while respecting 2D image structure

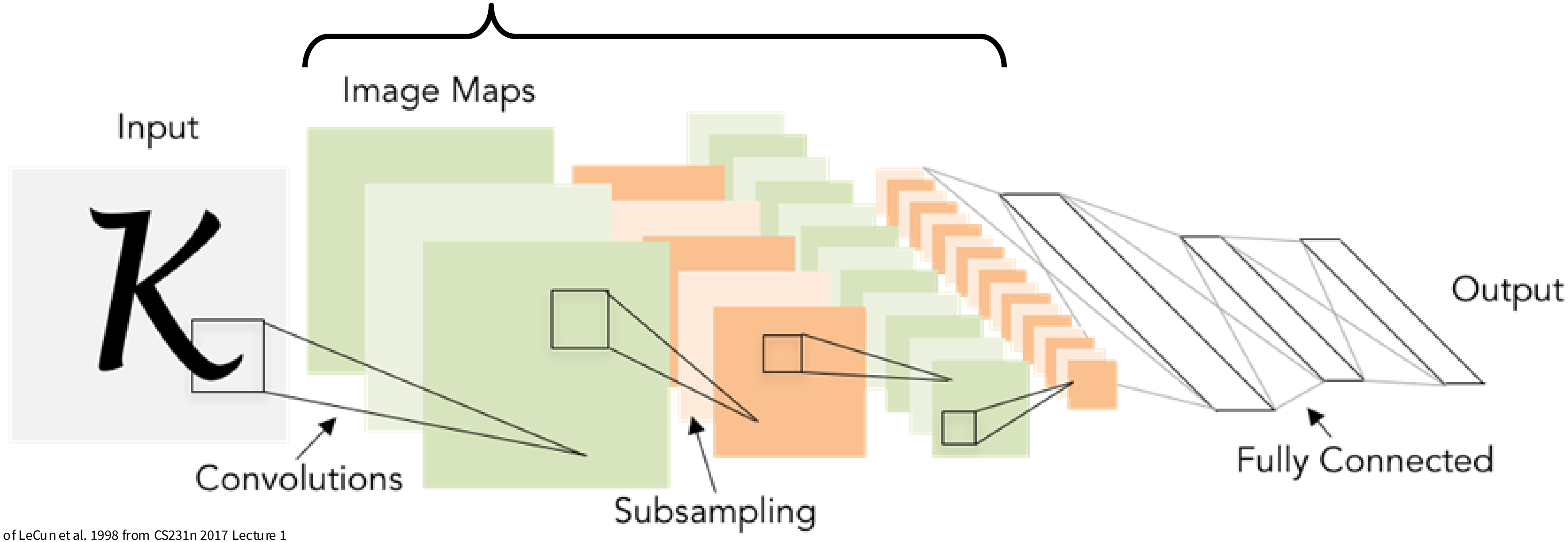


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Next: Convolutional Neural Networks

Convolution and **pooling** operators extract features while respecting 2D image structure

Fully-Connected layers form an MLP at the end to predict scores

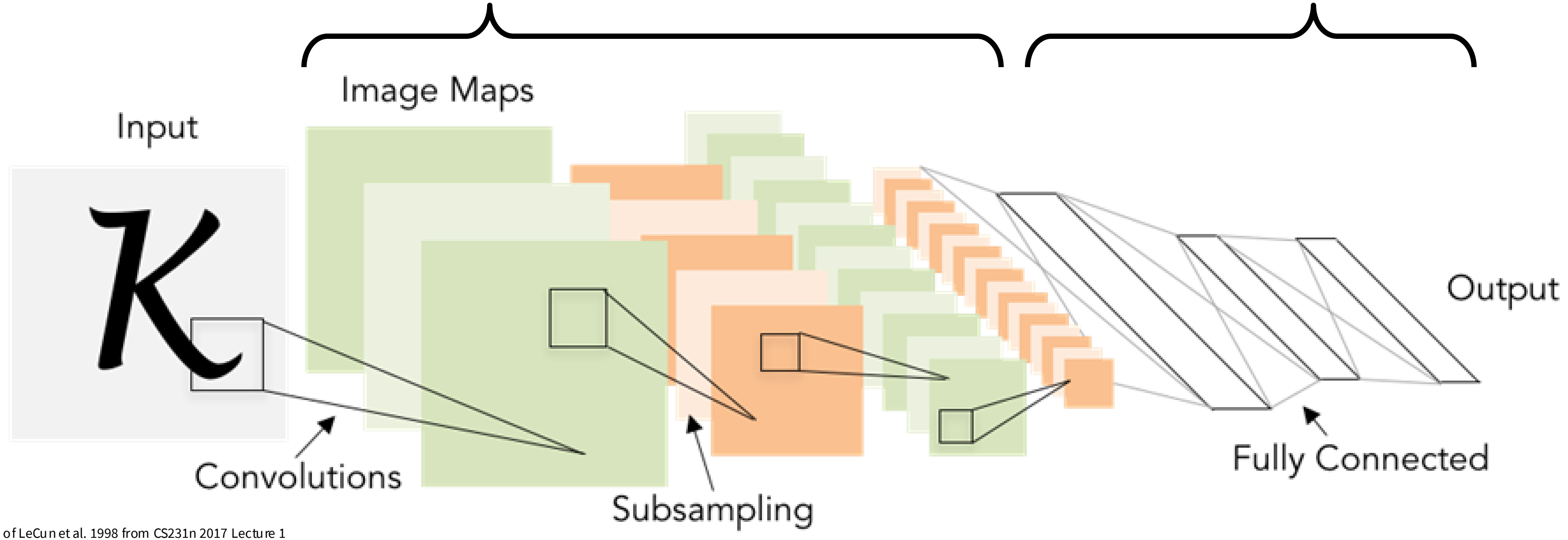


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Next: Convolutional Neural Networks

Trained end-to-end with backprop + gradient descent

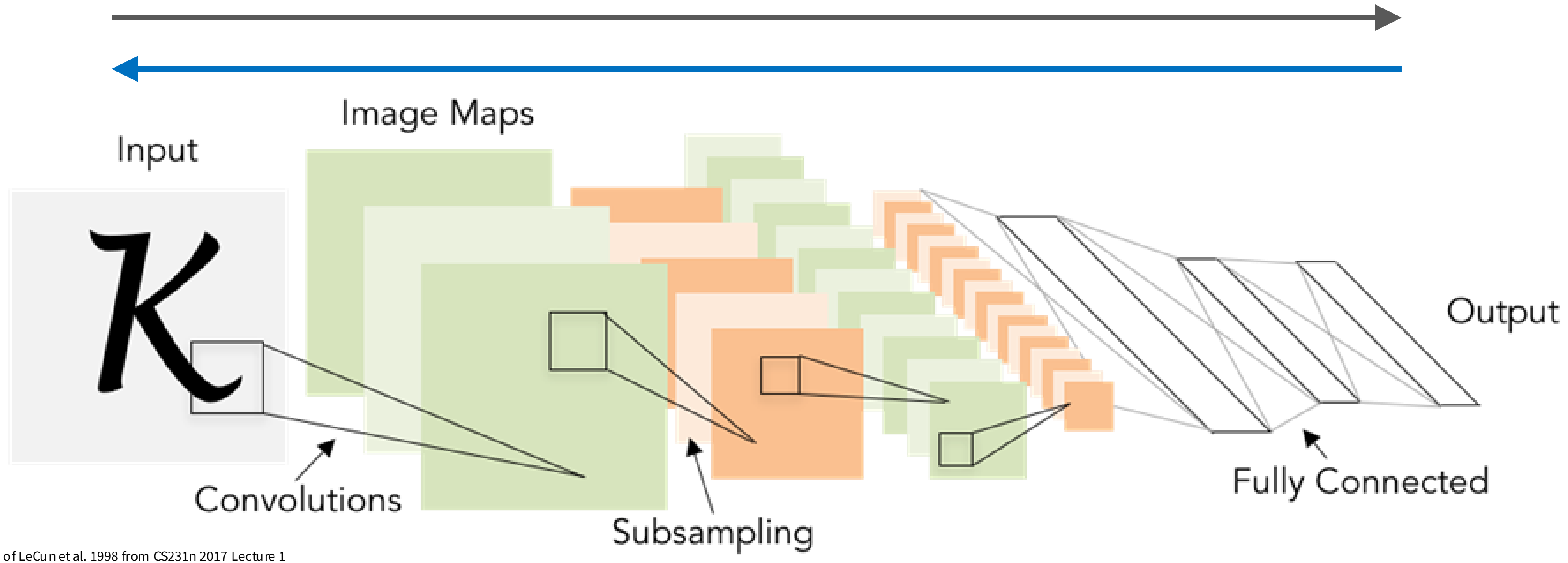
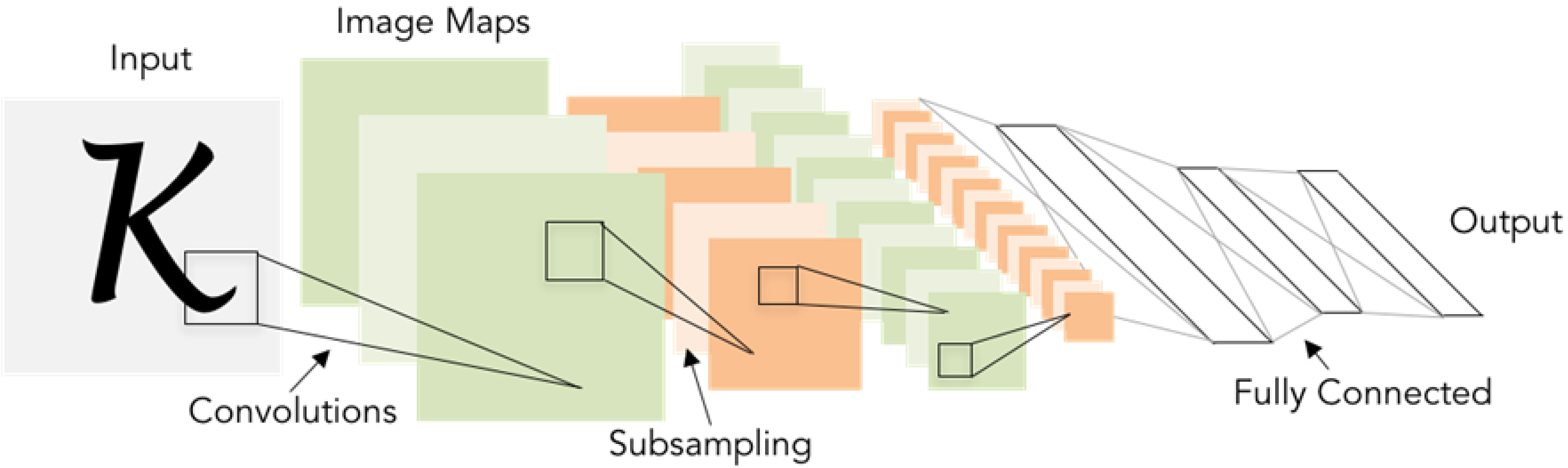


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history:

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]



A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky, Sutskever, Hinton, 2012]

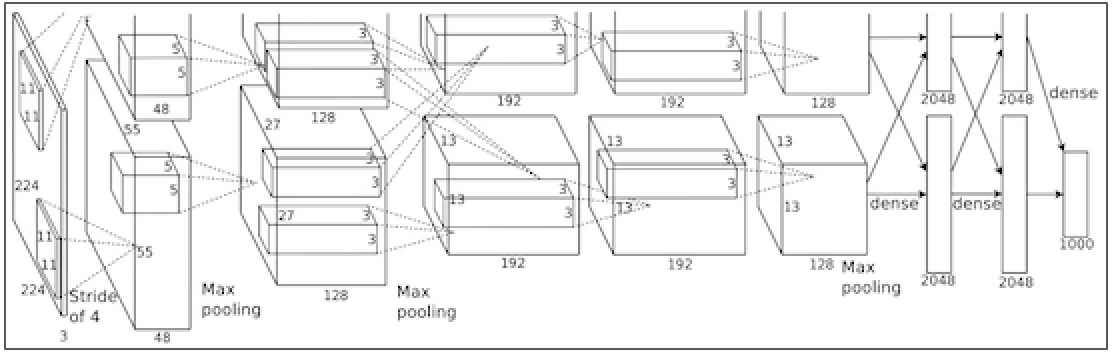
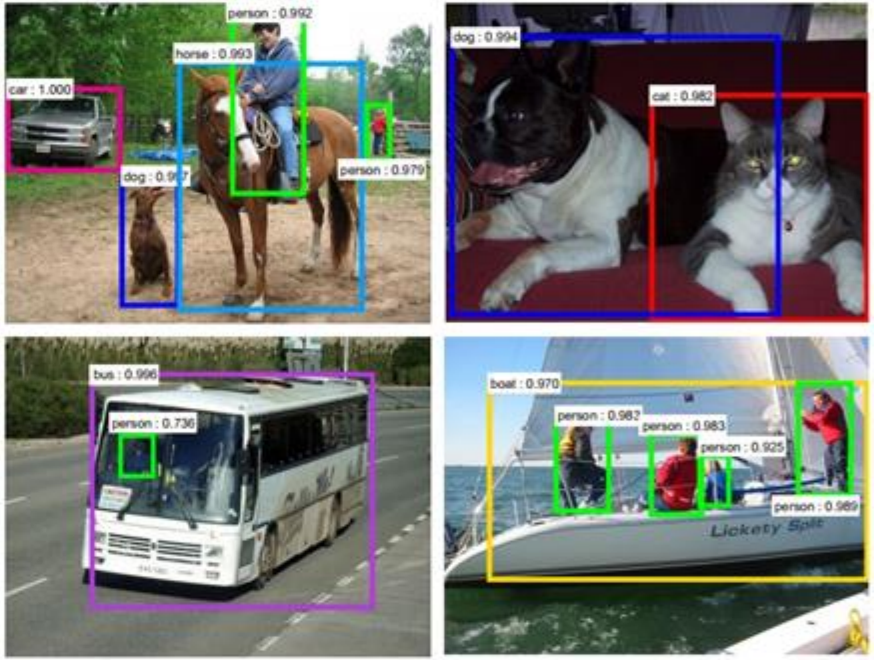


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

~2012 – 2020: ConvNets dominate all vision tasks

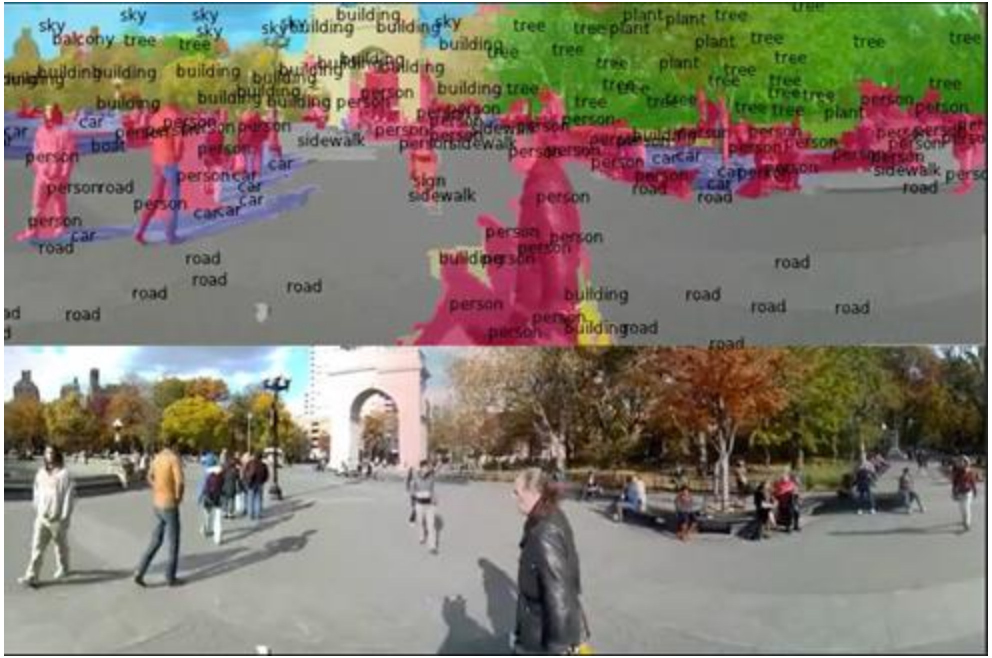
Detection



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

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

Detection



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

~2012 – 2020: ConvNets dominate all vision tasks

Image Captioning

A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

~2012 – 2020: ConvNets dominate all vision tasks

Text-to-Image Generation

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

A zombie in the style of Picasso

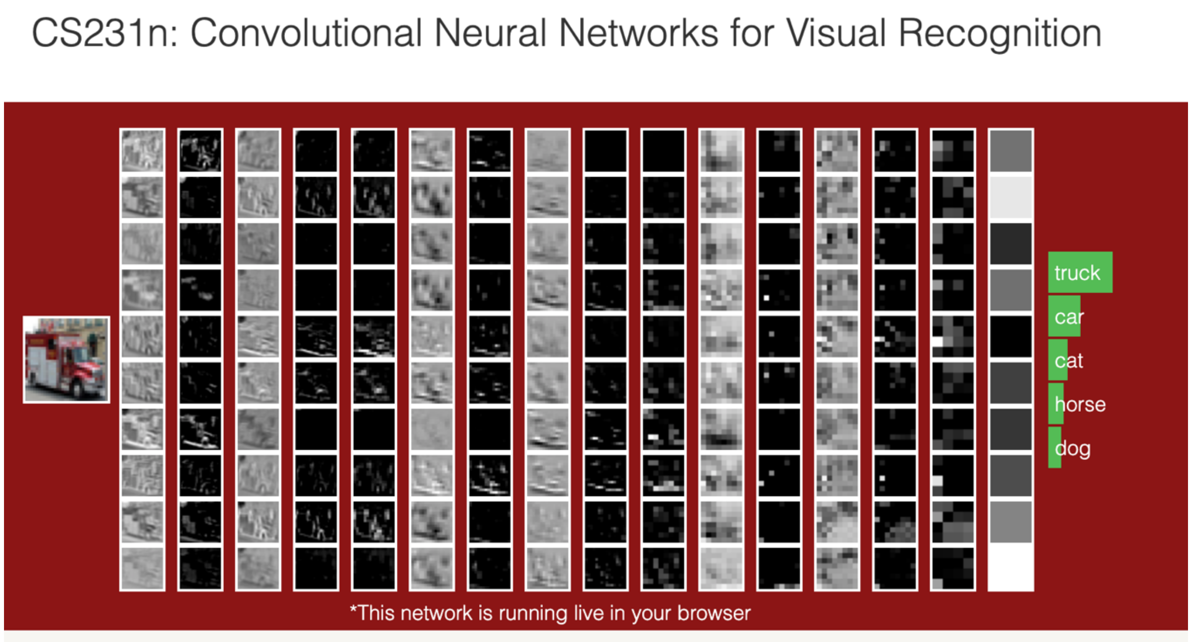
An image of a half mouse half octopus

A painting of a squirrel eating a burger

A watercolor painting of a chair that looks like an octopus

A shirt with the inscription: "I love generative models!"

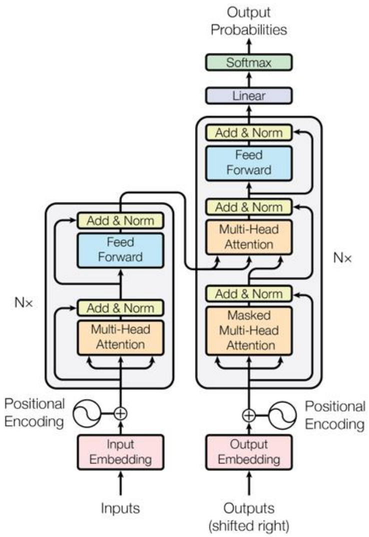
~2012 – 2020: ConvNets dominate all vision tasks



This class used to be focused on ConvNets!

2021 - Present: Transformers have taken over

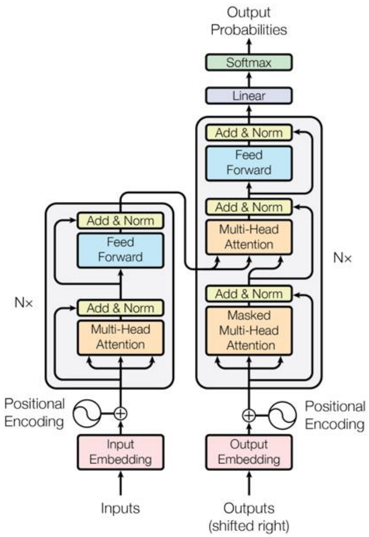
2017: Transformers for language tasks



Vaswani et al, “Attention is all you need”, NeurIPS 2017

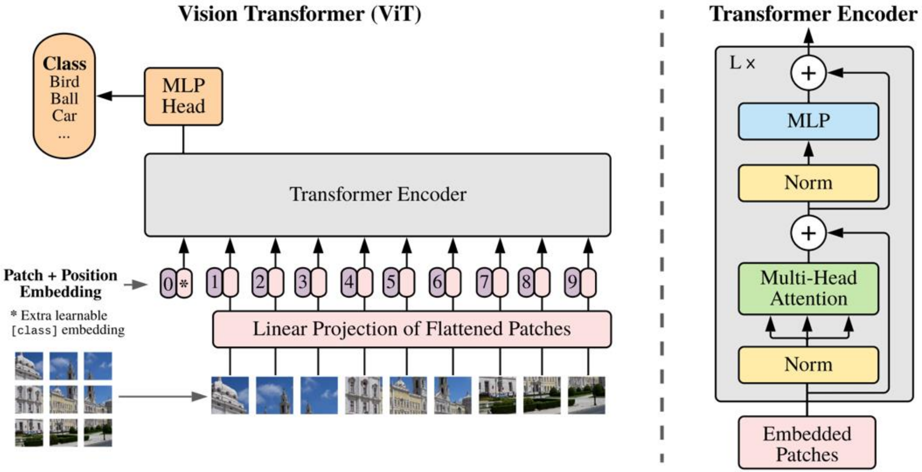
2021 - Present: Transformers have taken over

2017: Transformers for language tasks



Vaswani et al, "Attention is all you need", NeurIPS 2017

2021: Transformers for vision tasks



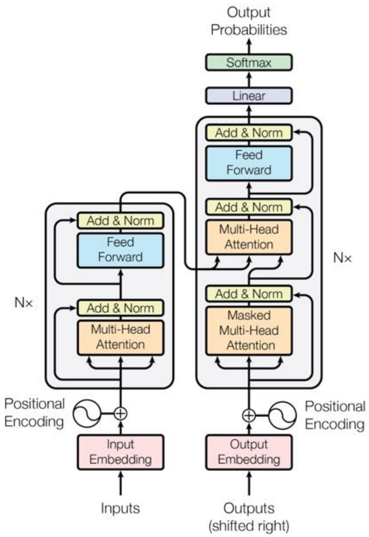
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

2021 - Present: Transformers have taken over

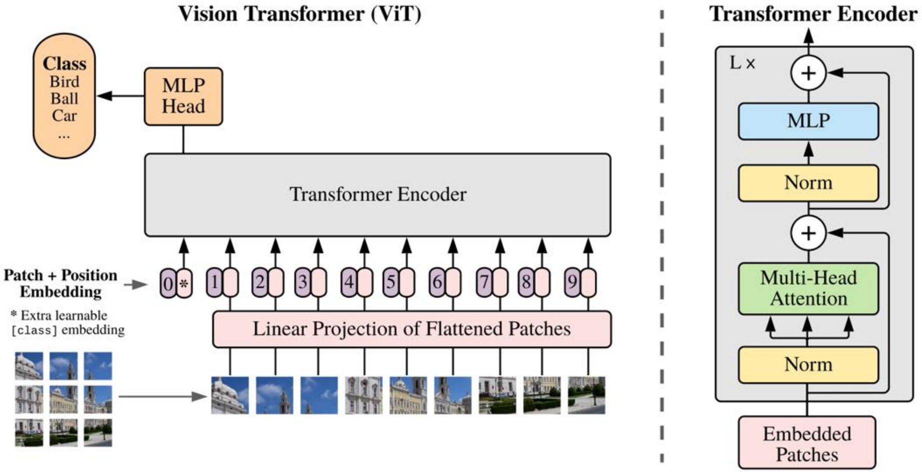
2017: Transformers for language tasks

2021: Transformers for vision tasks

Wait until Lecture 8!



Vaswani et al, "Attention is all you need", NeurIPS 2017



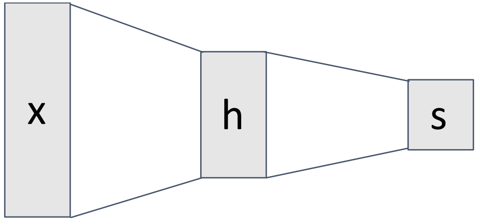
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Convolutional Neural Networks

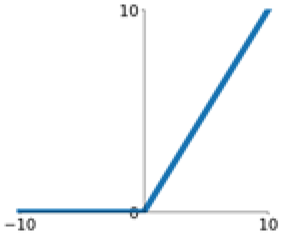
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer



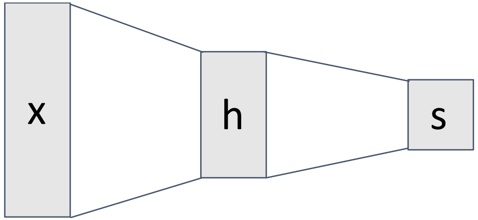
Activation Function



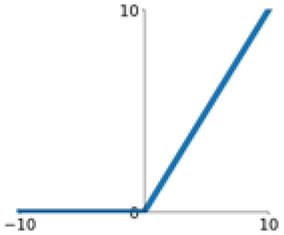
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

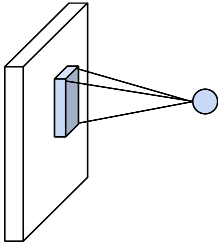


Activation Function

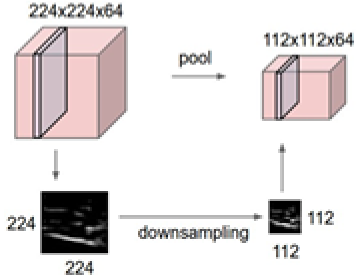


Today: Image-specific operators

Convolution Layer



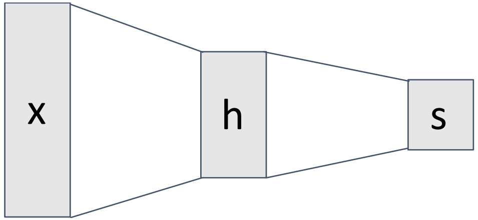
Pooling Layer



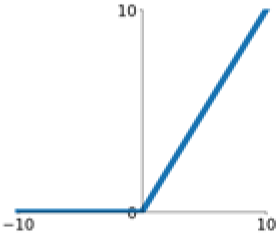
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

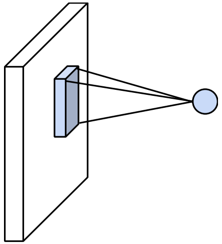


Activation Function

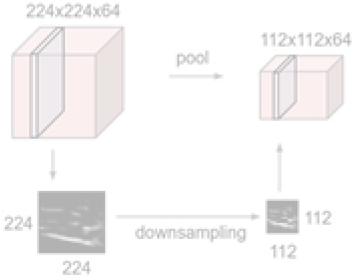


Today: Image-specific operators

Convolution Layer

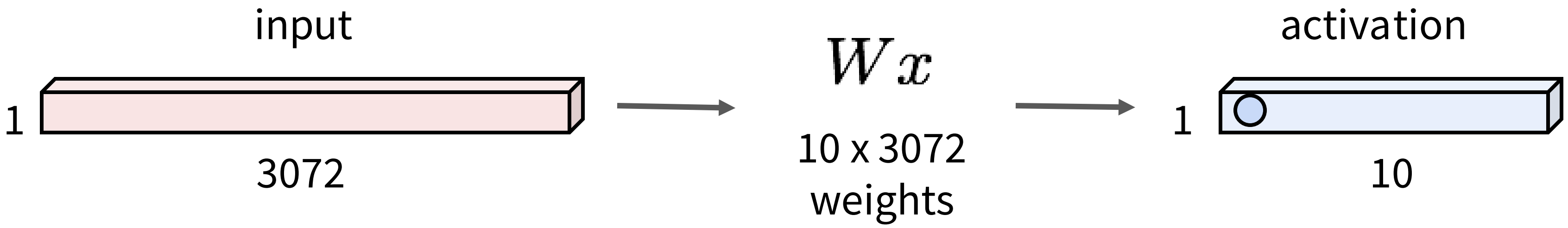


Pooling Layer



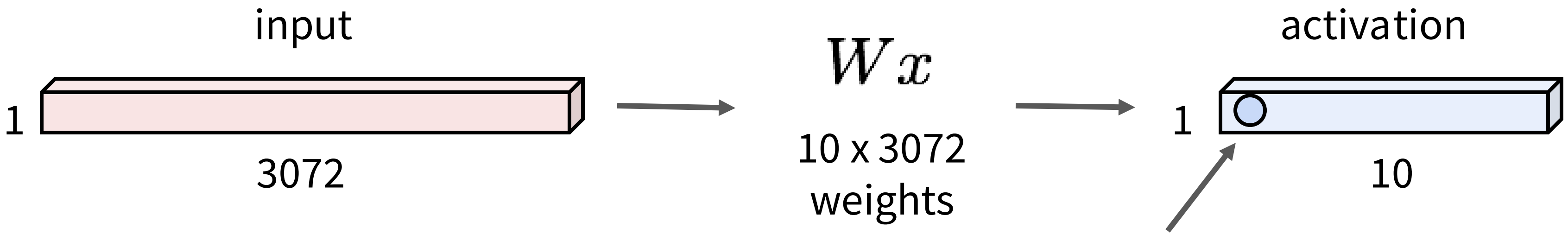
Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

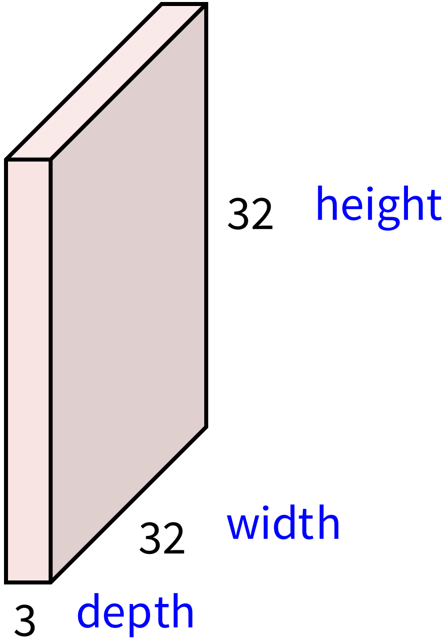
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

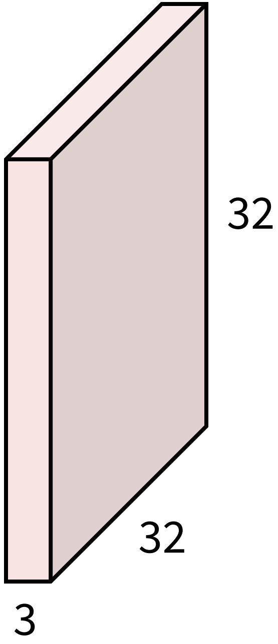
Convolution Layer

32x32x3 image -> preserve spatial structure

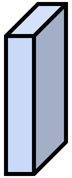


Convolution Layer

32x32x3 image



5x5x3 filter

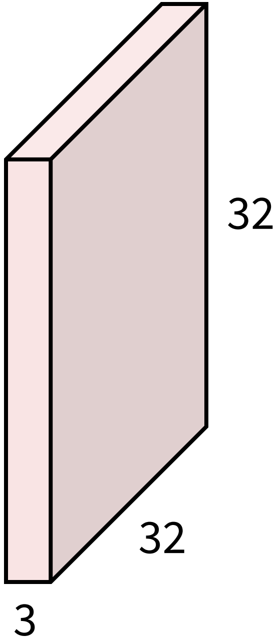


Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"

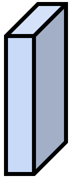
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

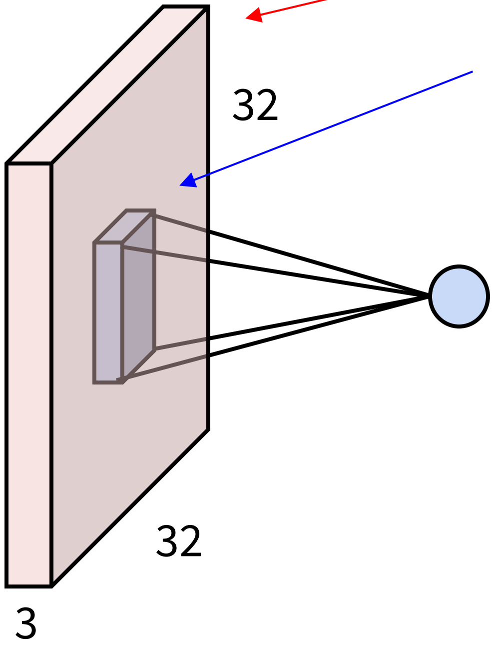


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

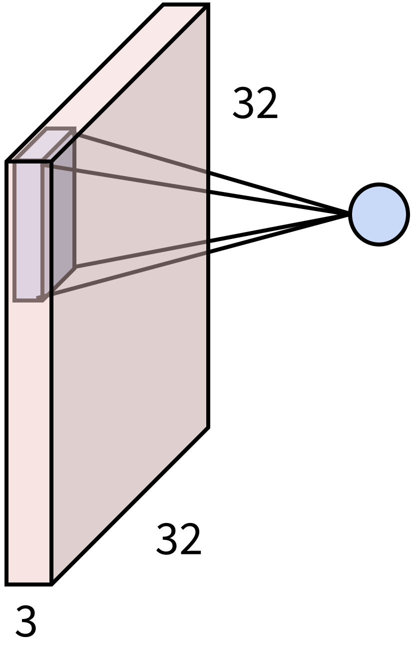
5x5x3 filter w



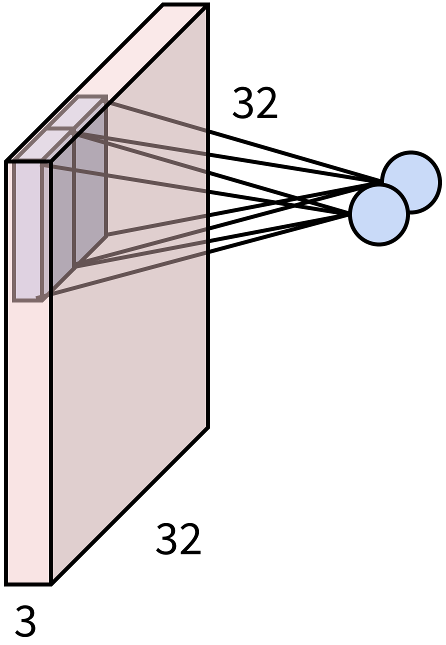
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

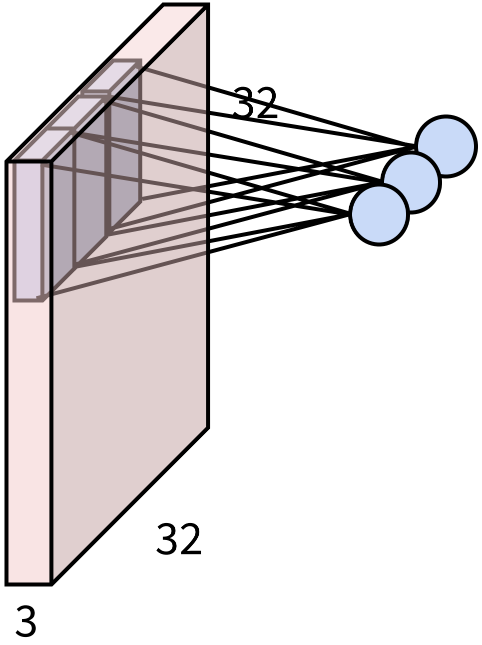
Convolution Layer



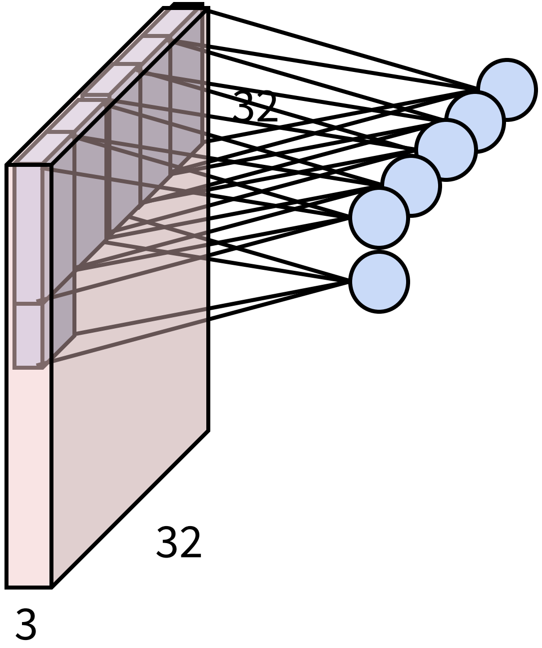
Convolution Layer



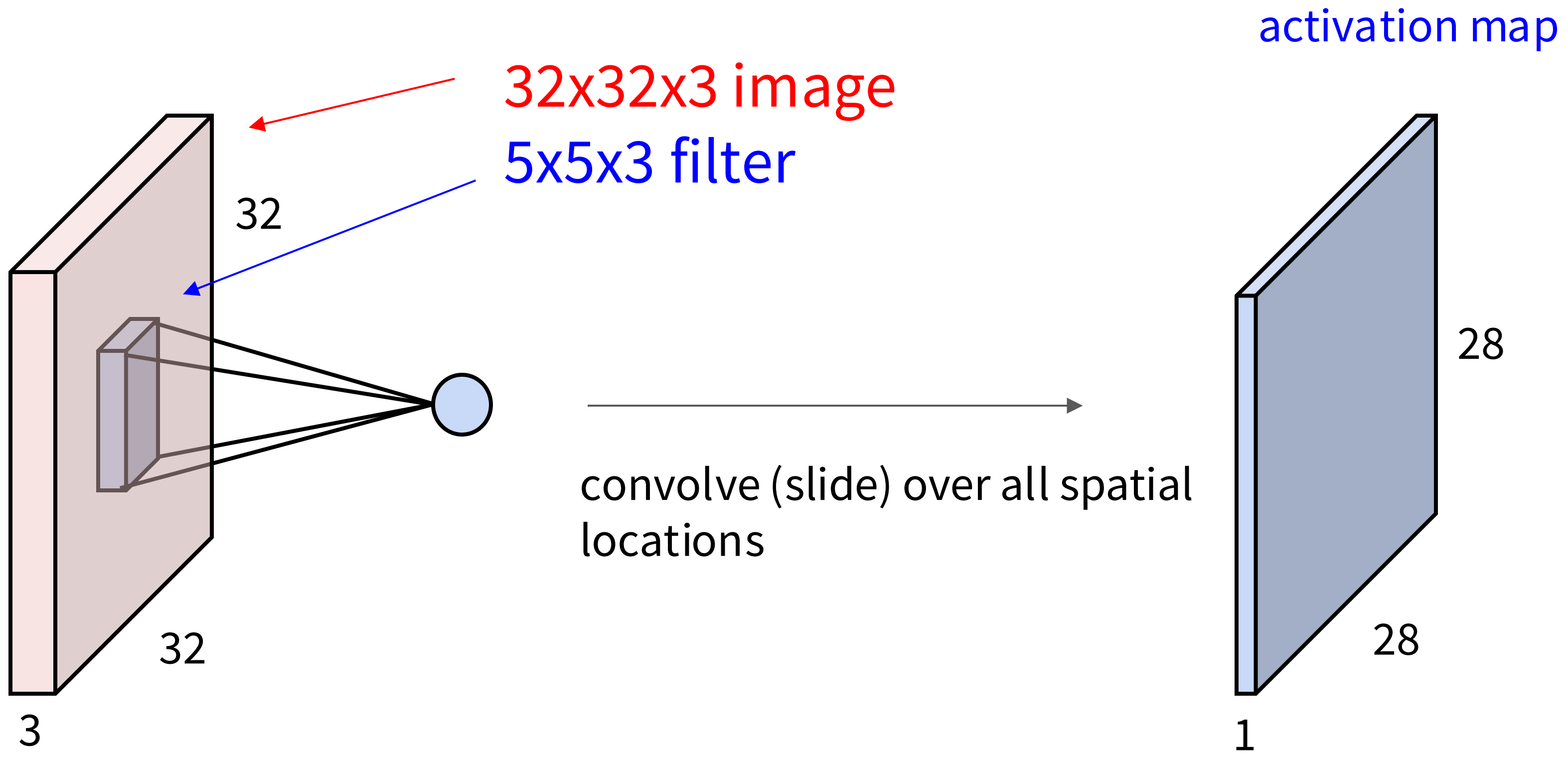
Convolution Layer



Convolution Layer

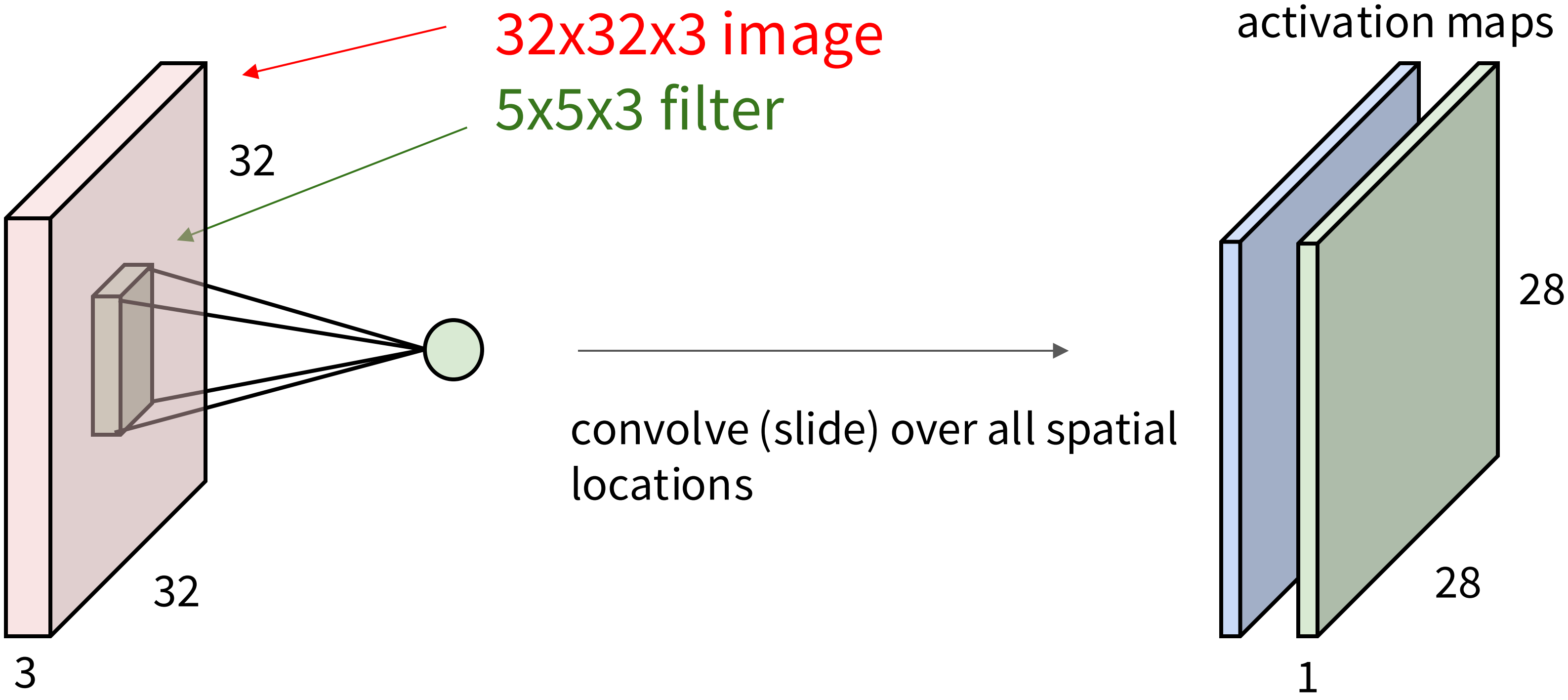


Convolution Layer

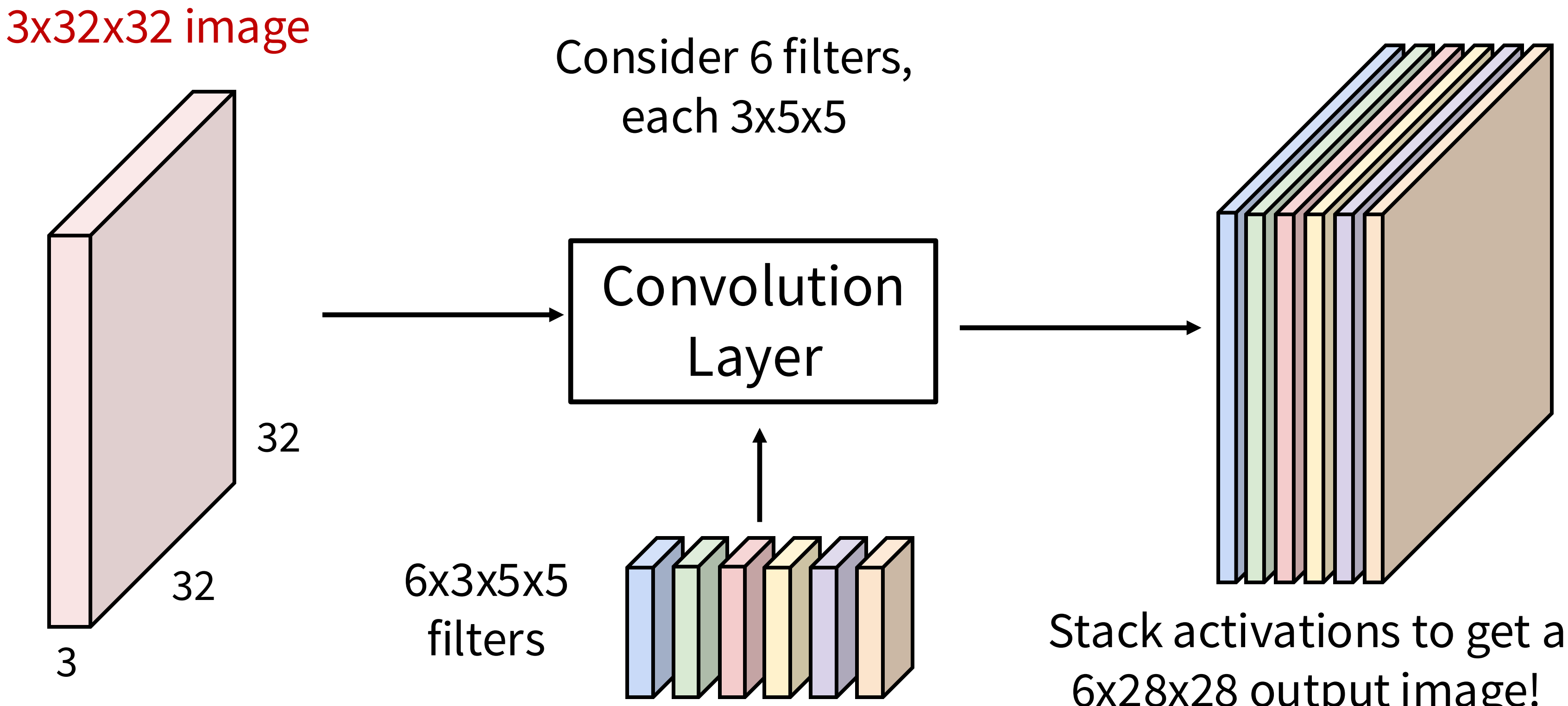


Convolution Layer

consider a second, green filter

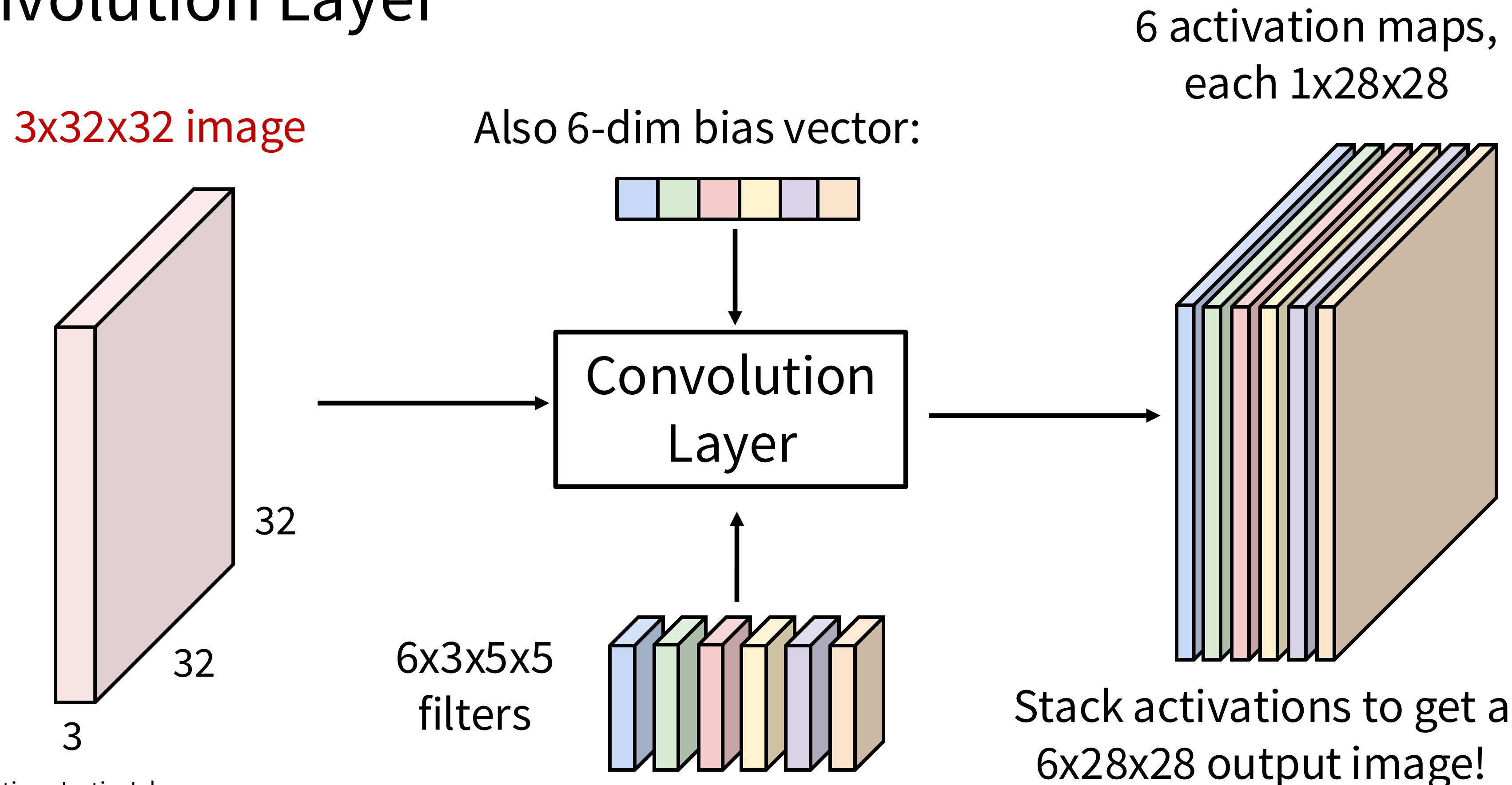


Convolution Layer



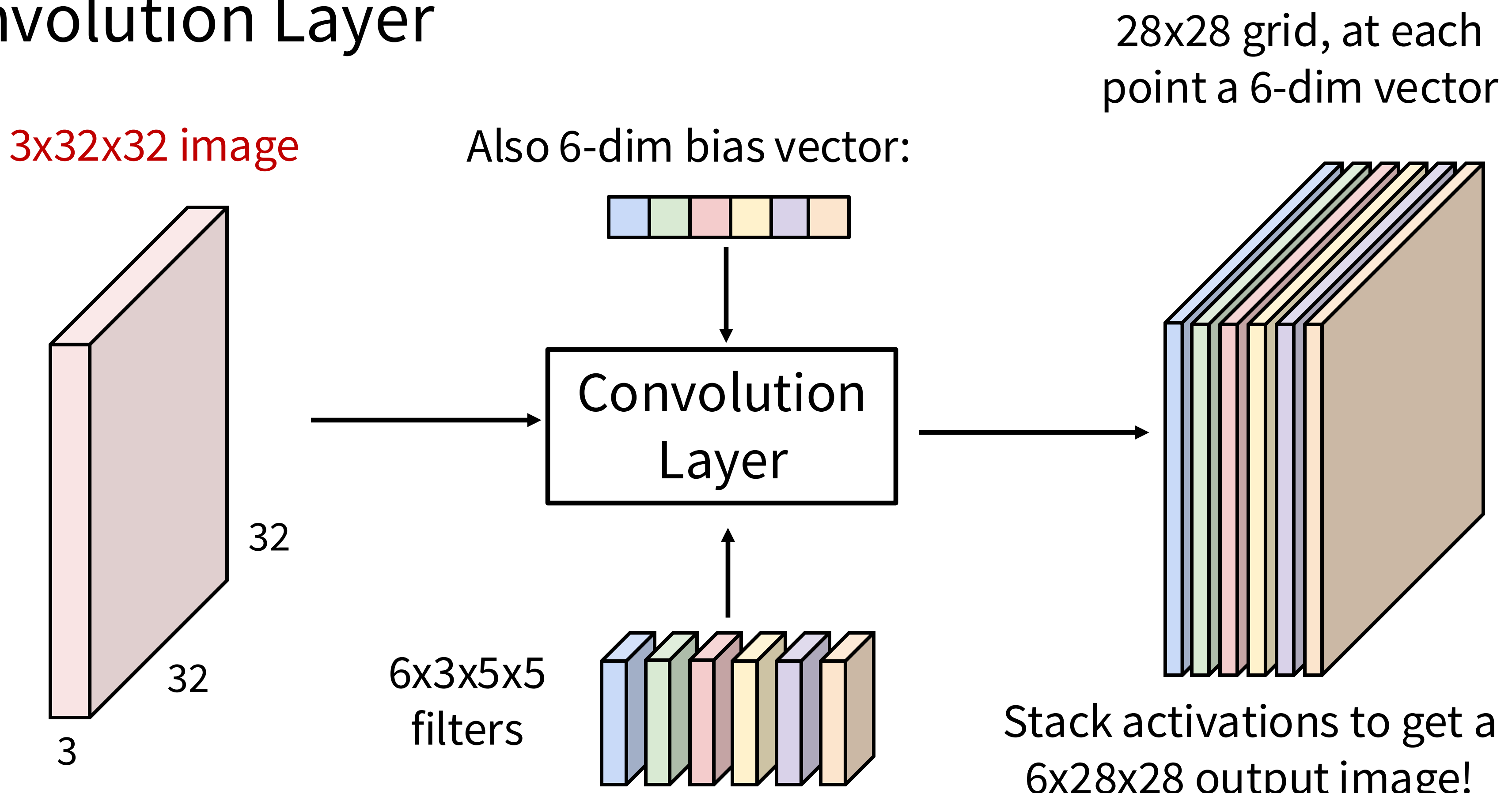
Slide inspiration: Justin Johnson

Convolution Layer



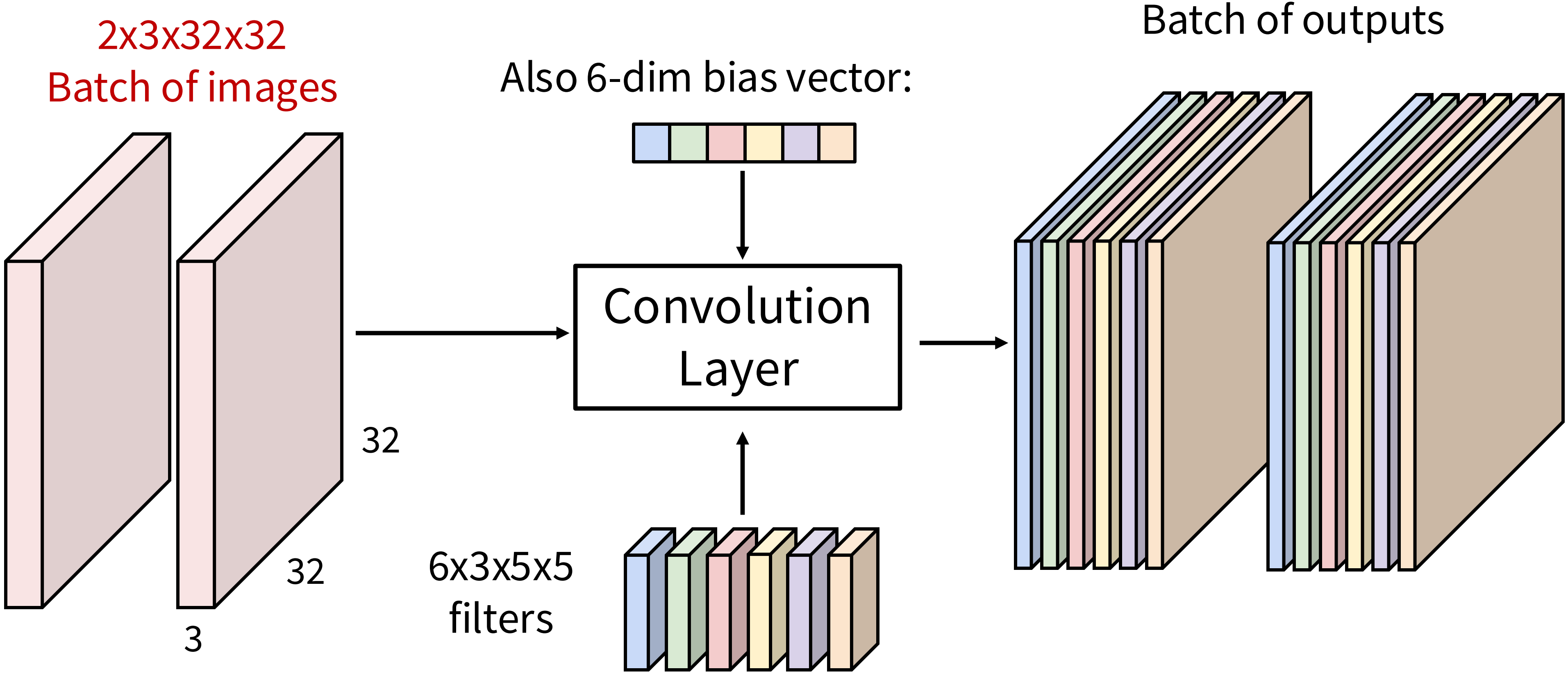
Slide inspiration: Justin Johnson

Convolution Layer



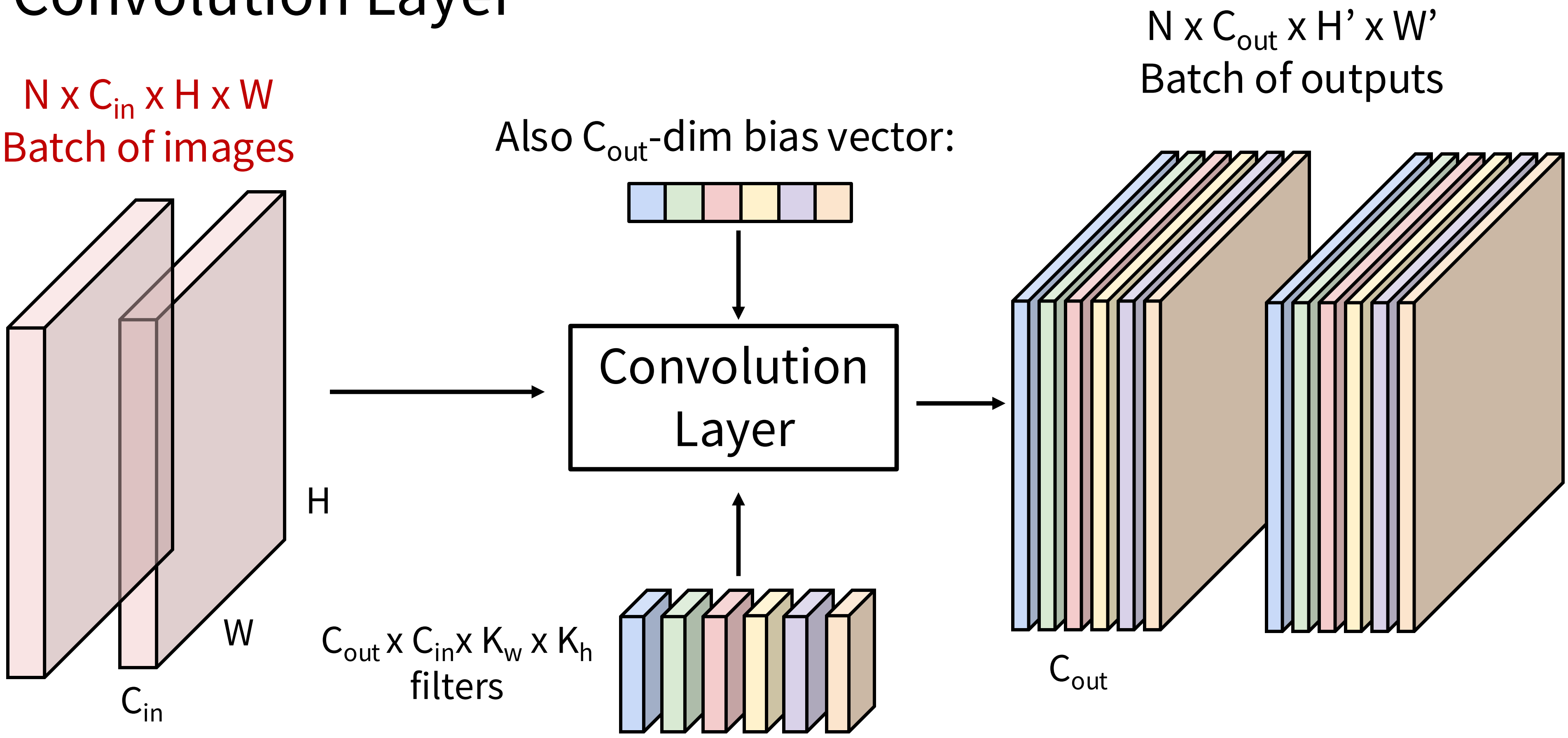
Slide inspiration: Justin Johnson

Convolution Layer



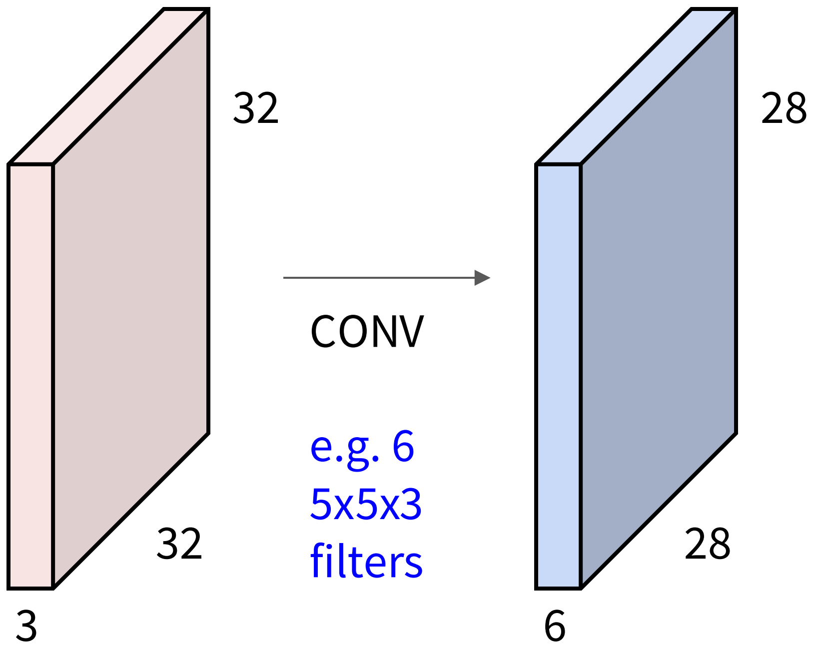
Slide inspiration: Justin Johnson

Convolution Layer

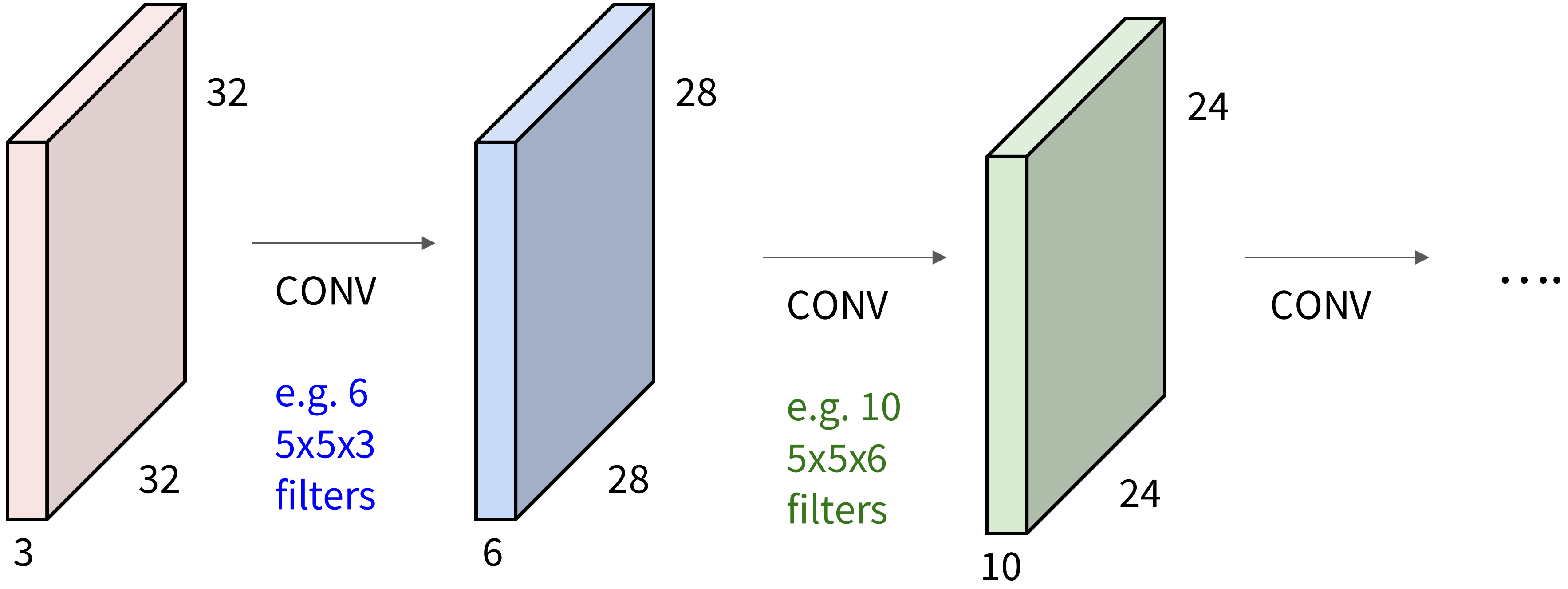


Slide inspiration: Justin Johnson

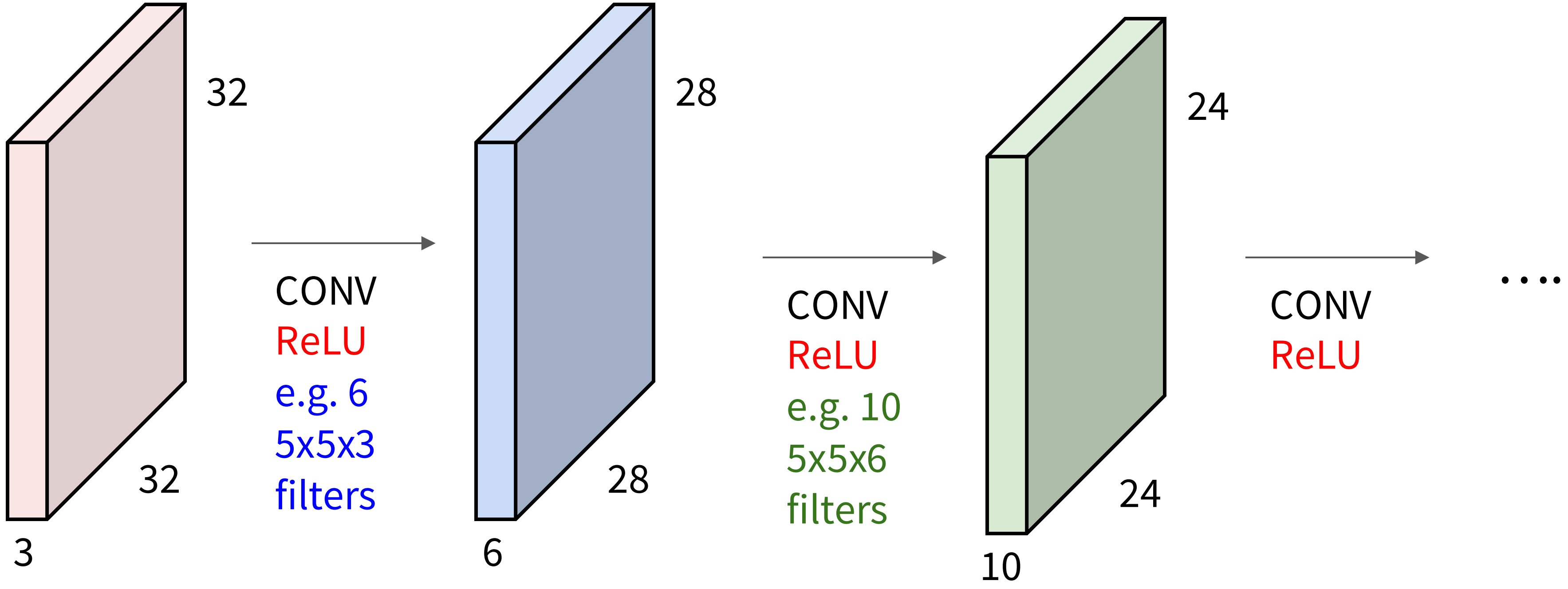
A ConvNet is a neural network with Conv layers



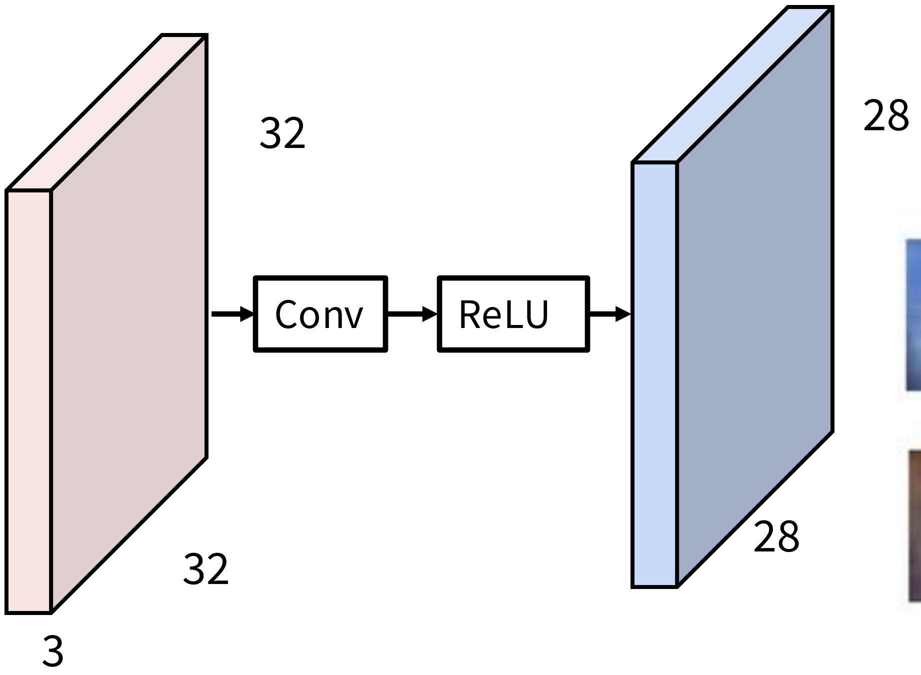
A ConvNet is a neural network with Conv layers



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

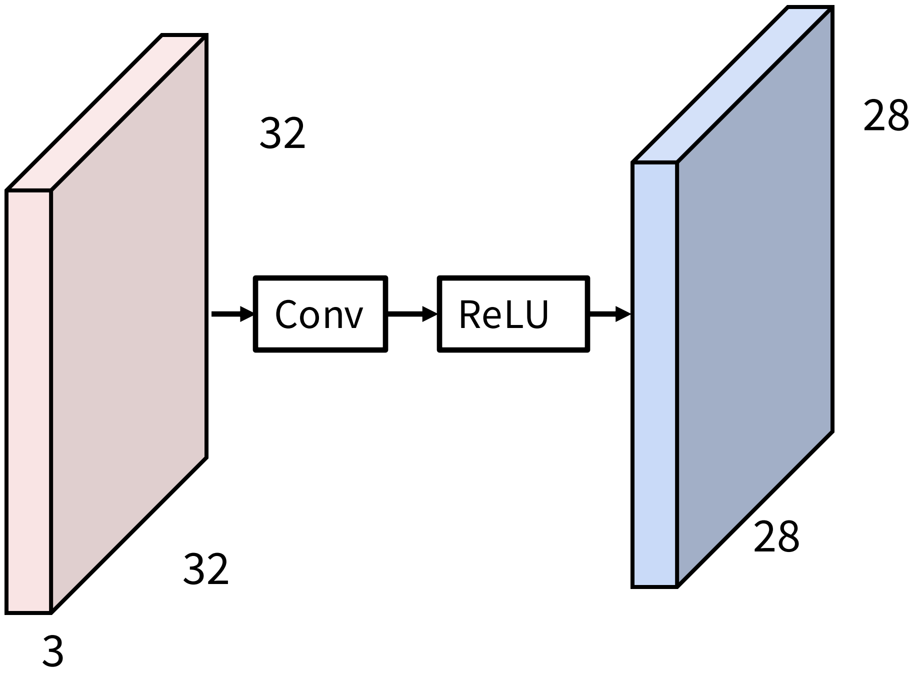


What do Conv filters learn?



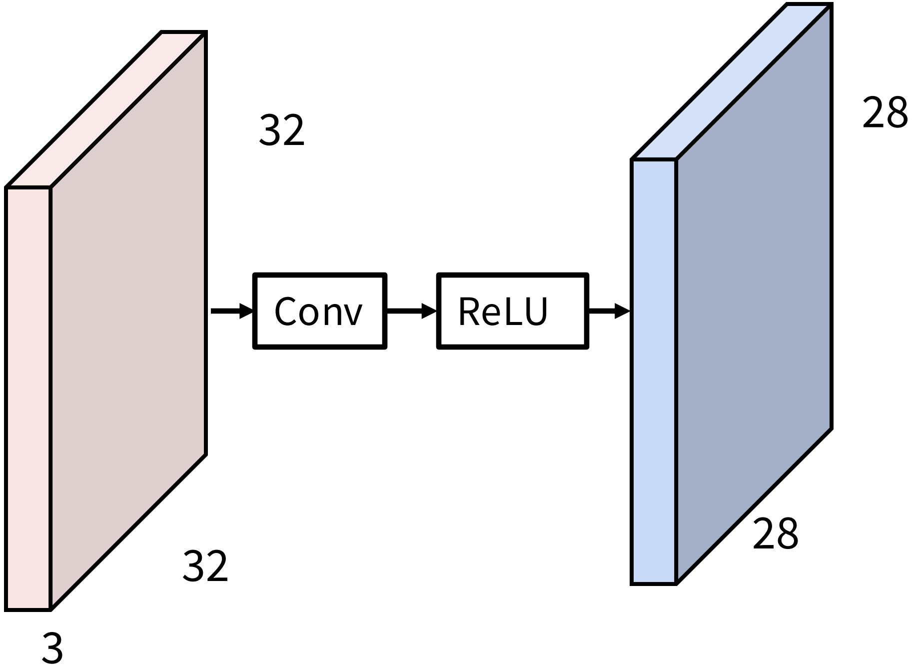
Linear classifier: One template per class

What do Conv filters learn?

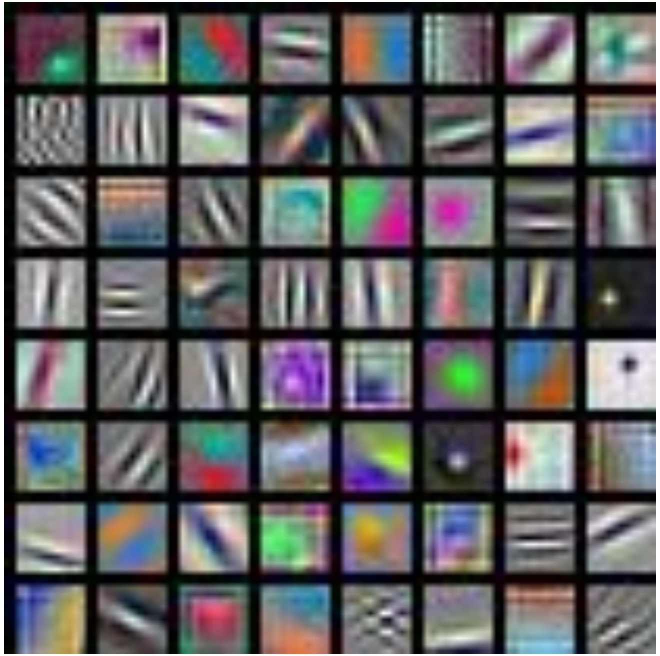


MLP: Bank of whole-image templates

What do Conv filters learn?

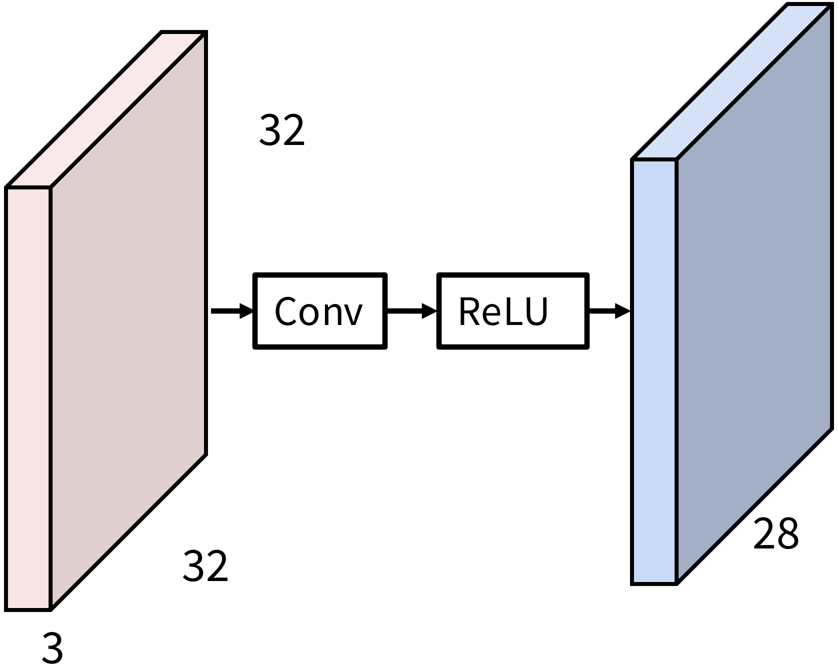


First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

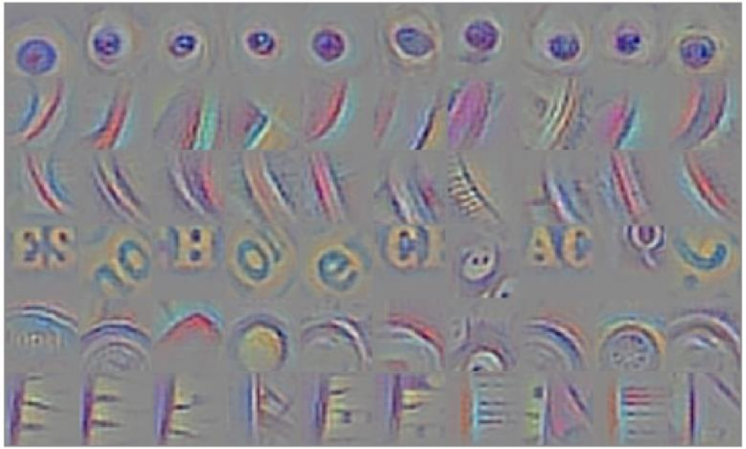


AlexNet: 64 filters, each 3x11x11

What do Conv filters learn?



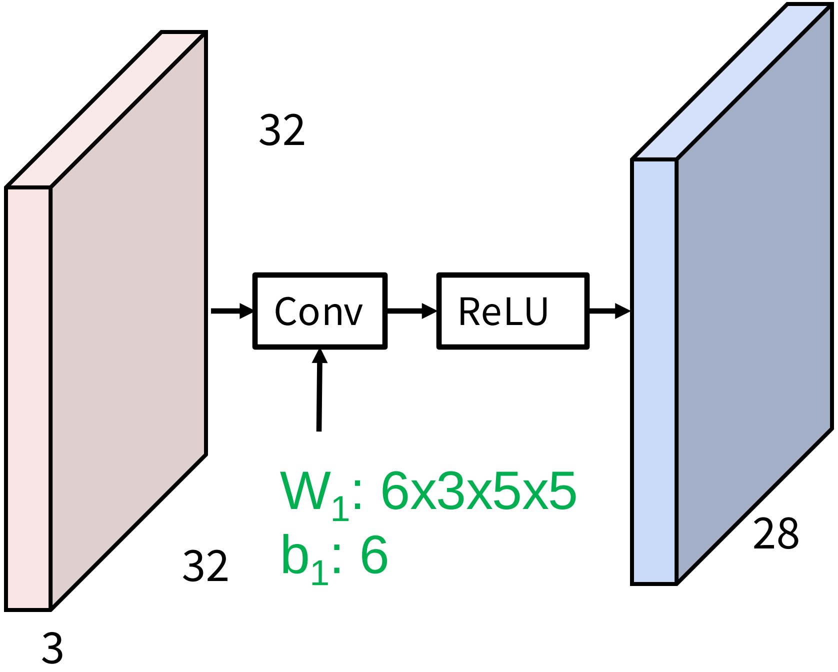
Deeper conv layers: Harder to visualize
Tend to learn larger structures e.g. eyes, letters



6th layer conv layer from an ImageNet model

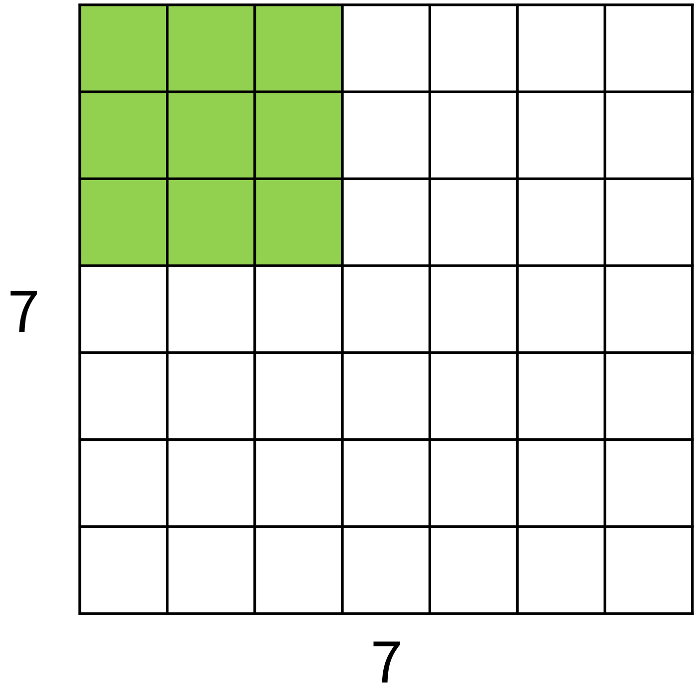
Visualization from [Springenberg et al, ICLR 2015]

Convolution: Spatial Dimensions



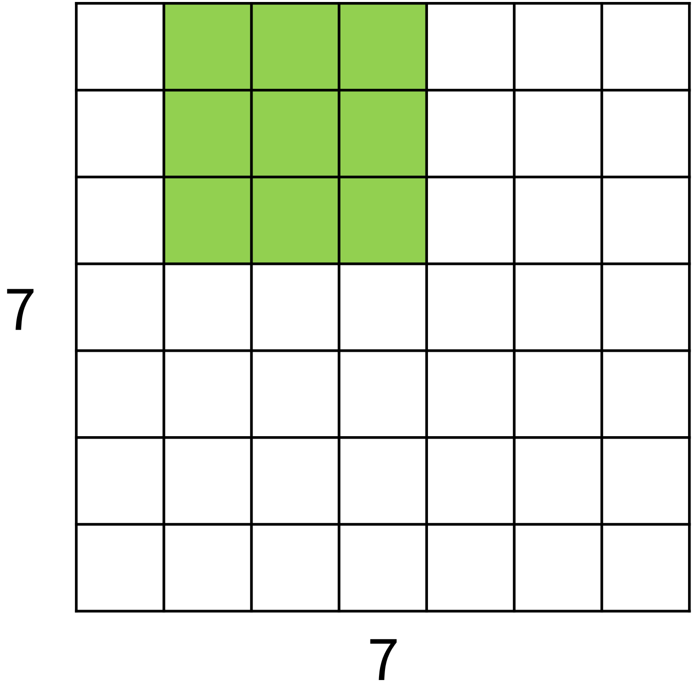
W_1: 6x3x5x5
b_1: 6

Convolution: Spatial Dimensions



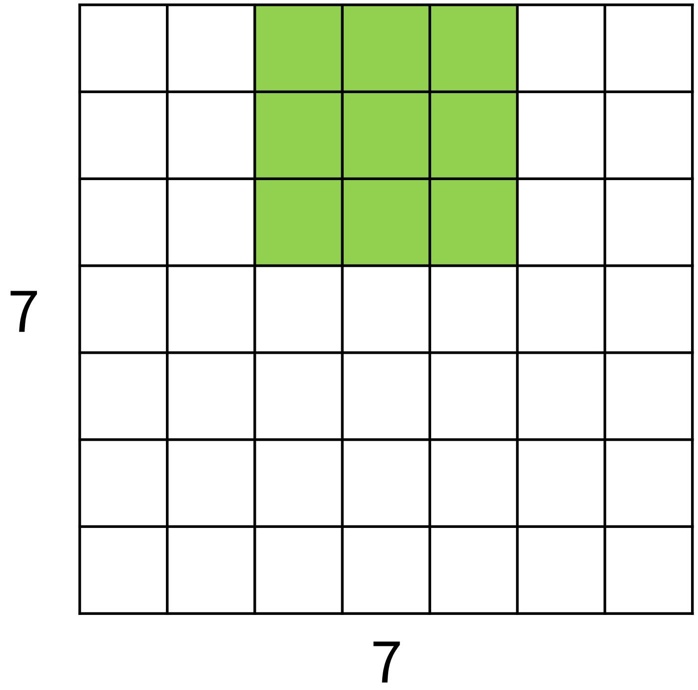
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



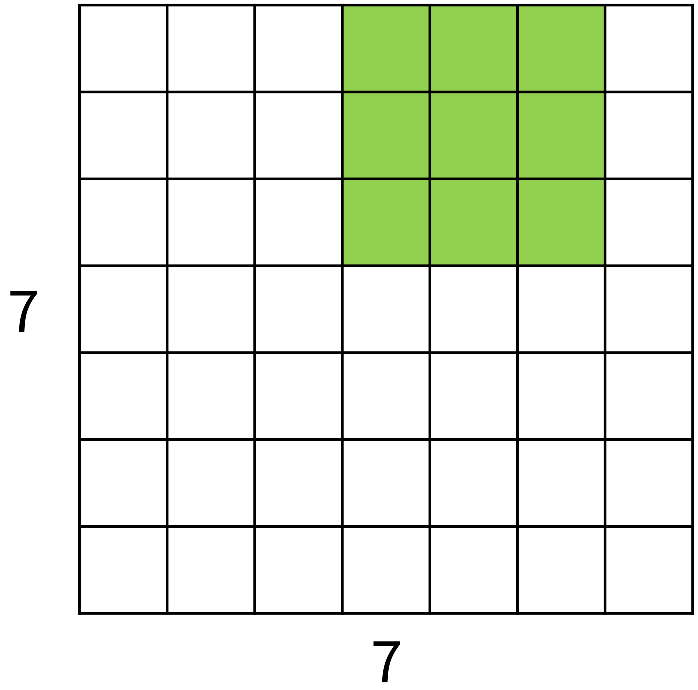
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



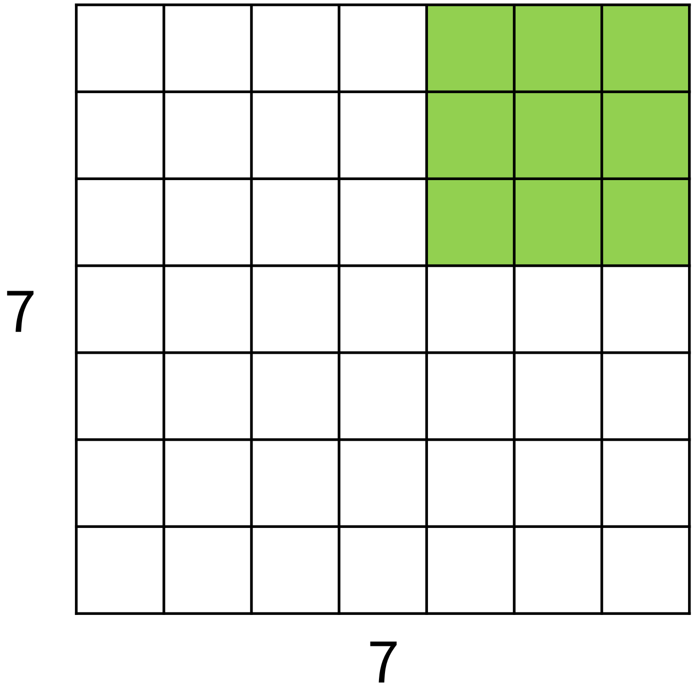
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



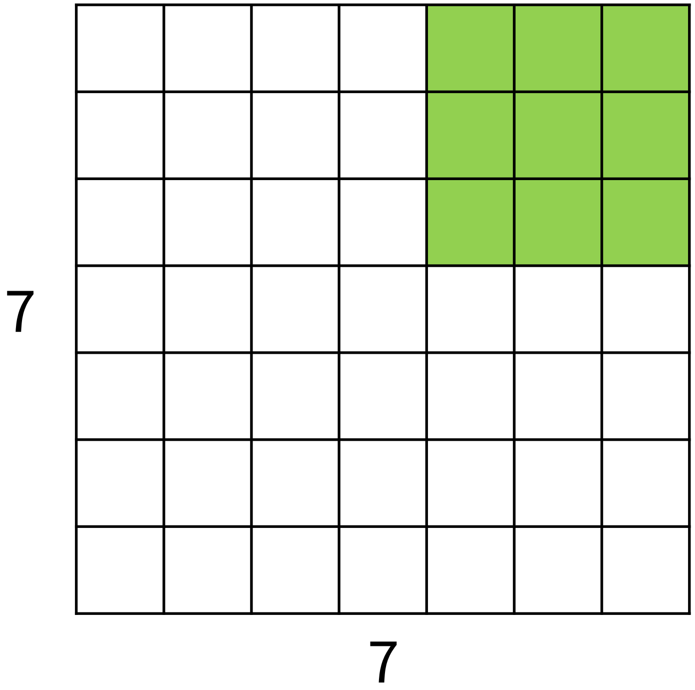
Input: 7x7
Filter: 3x3

Convolution: Spatial Dimensions



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

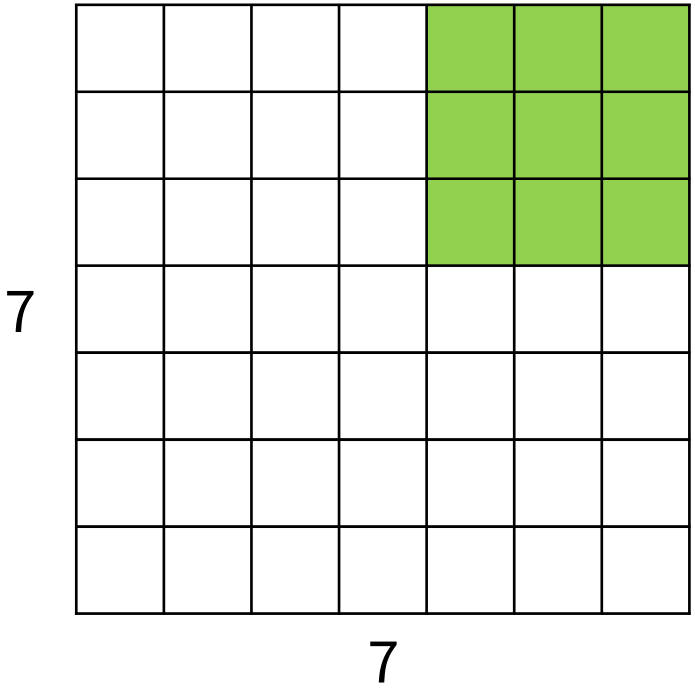
Convolution: Spatial Dimensions



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

<u>Problem</u>: Feature maps shrink with each layer!

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

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

Problem: Feature maps shrink with each layer!

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

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

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

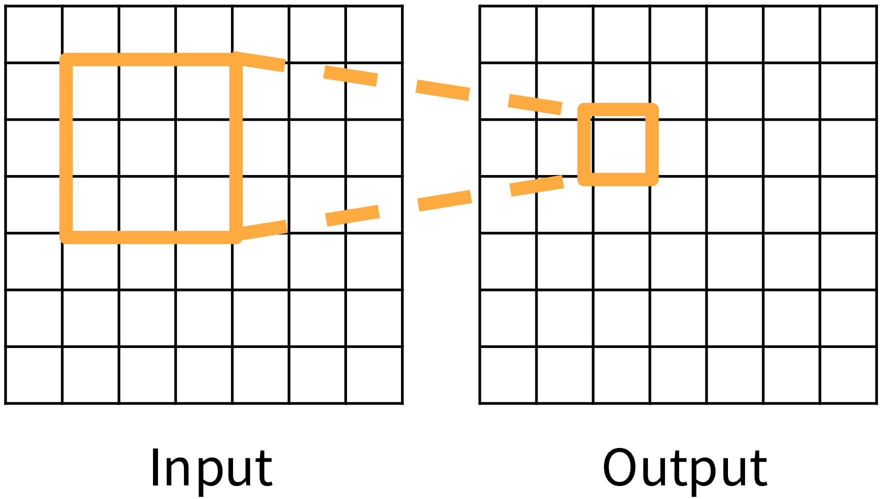
Input: 7x7
Filter: 3x3
Output: 5x5

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

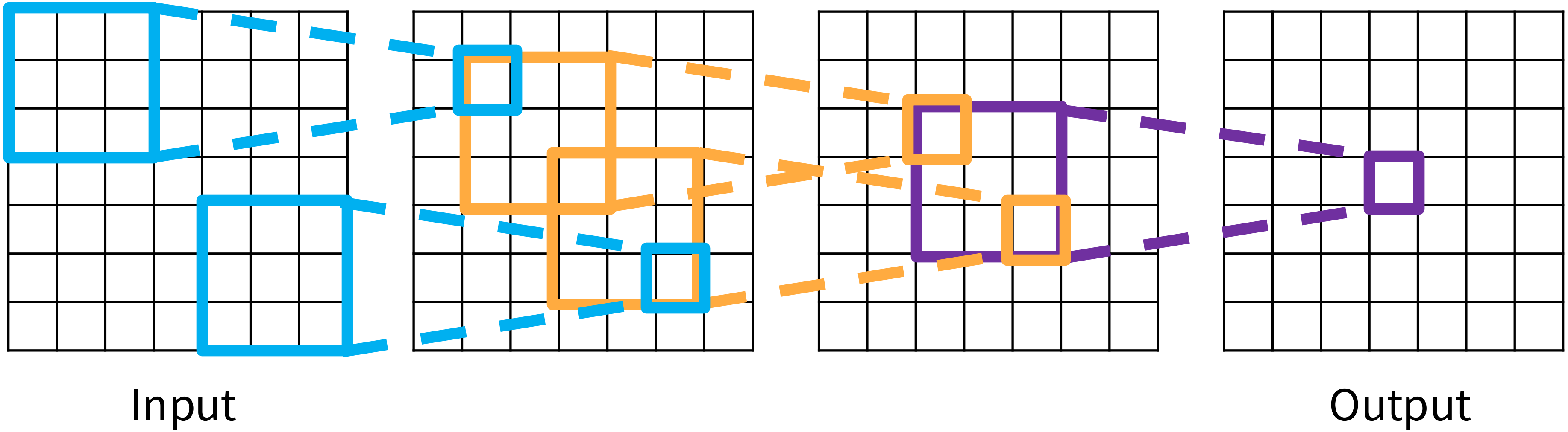
For convolution with **kernel size K**, each element in the output depends on a K x K receptive field in the input



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)

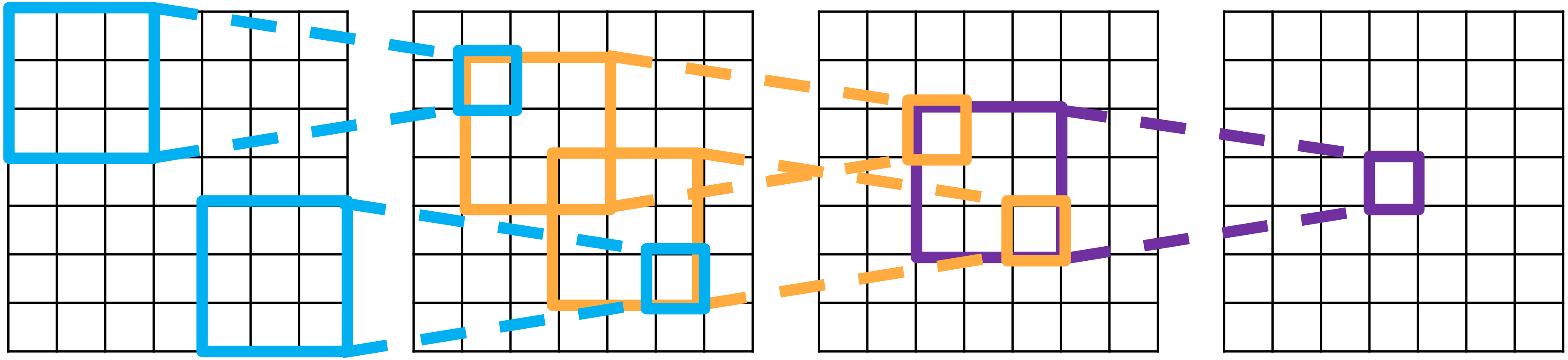


Be careful – "receptive field in the input" vs. "receptive field in the previous layer"

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

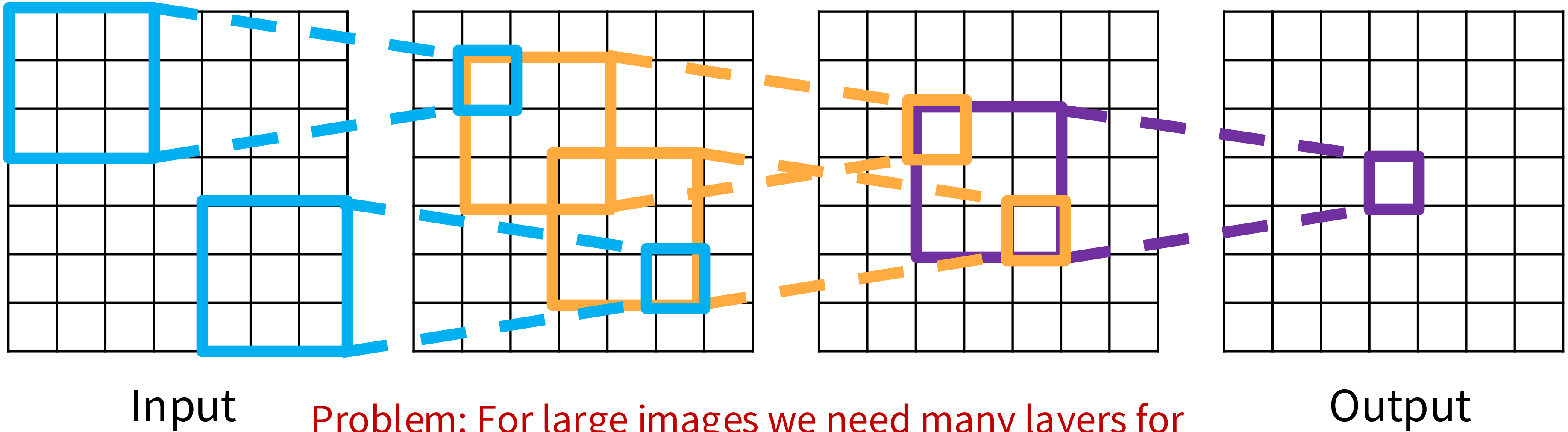
Output

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

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)

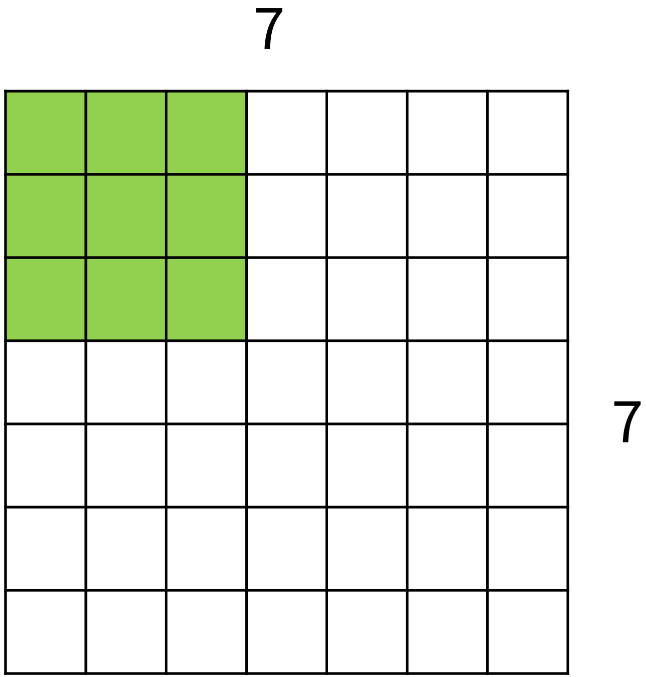


Problem: For large images we need many layers for each output to "see" the whole image image

Solution: Downsample inside the network

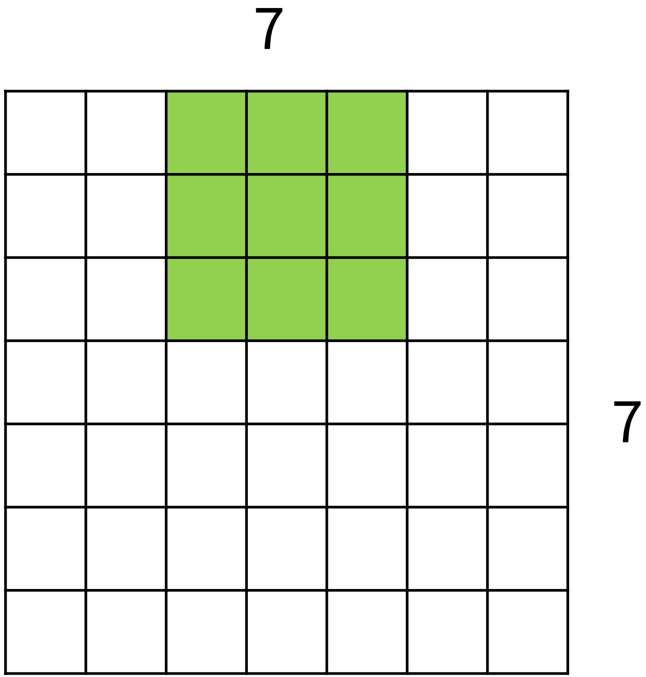
Slide inspiration: Justin Johnson

Strided Convolution



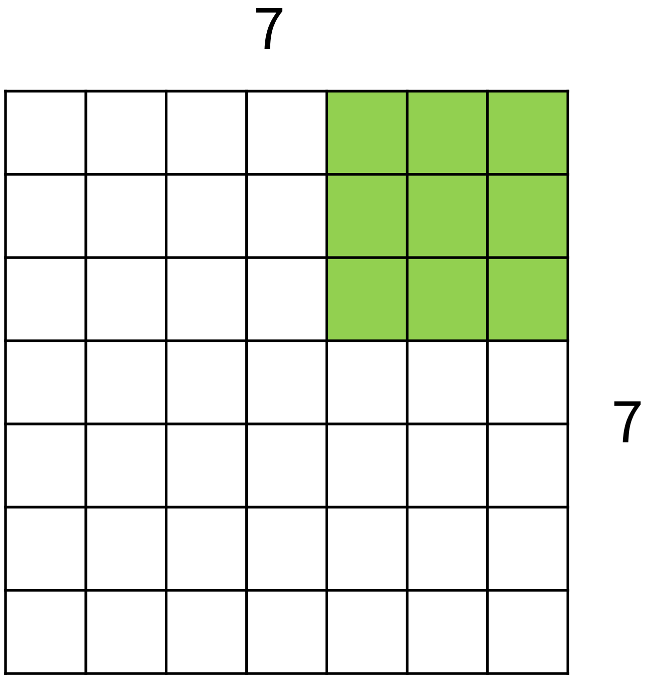
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



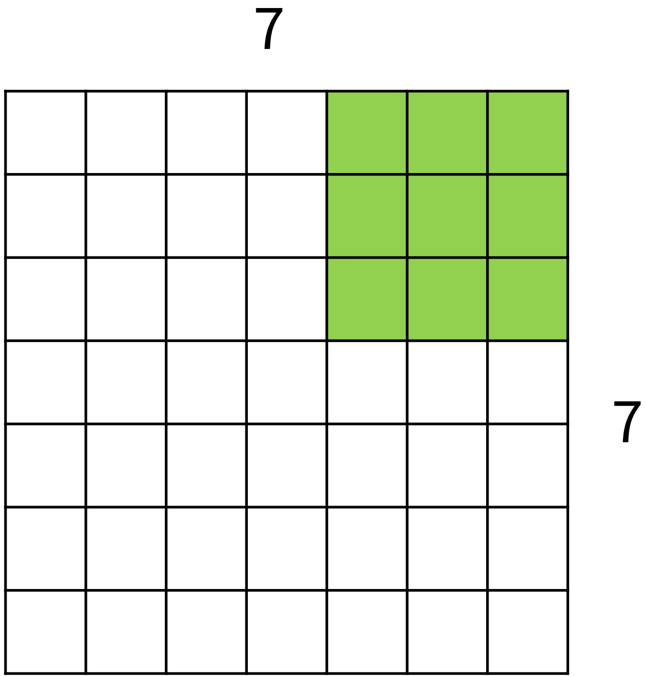
Input: 7x7
Filter: 3x3
Stride: 2

Strided Convolution



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

Strided Convolution



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

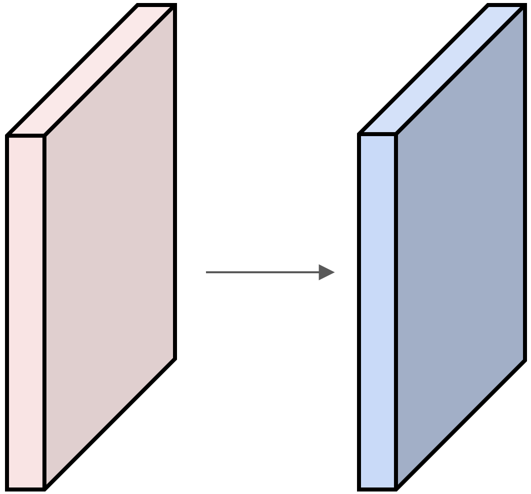
In general:
Input: W
Filter: K
Padding: P
Stride: S

→ Output:
$(W - K + 2P) / S + 1$

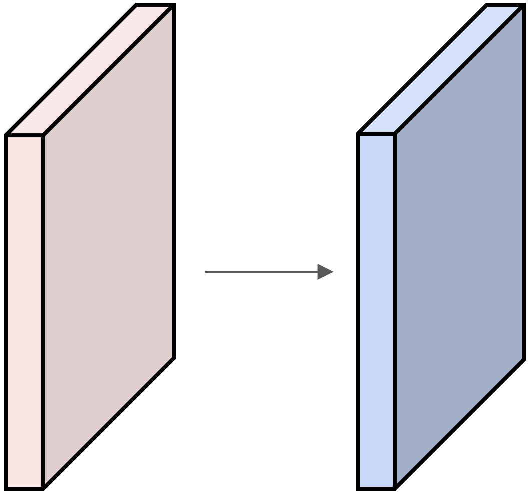
Convolution Example

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

Output volume size: ?



Convolution Example



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

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

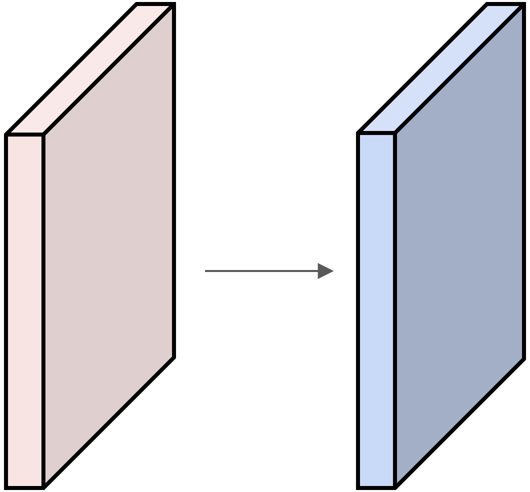
Output volume size: **10** x **32** x **32**

32 = (**32**+2*$\mathbf{2}$-**5**)/**1**+1

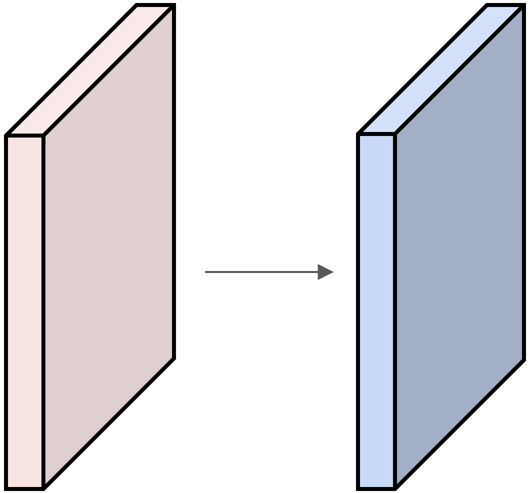
Convolution Example

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



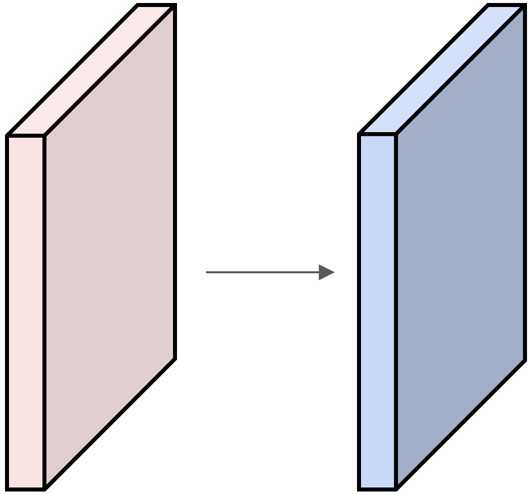
Input volume: **3** x 32 x 32
10 **5**x**5** filters with stride 1, pad 2

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**

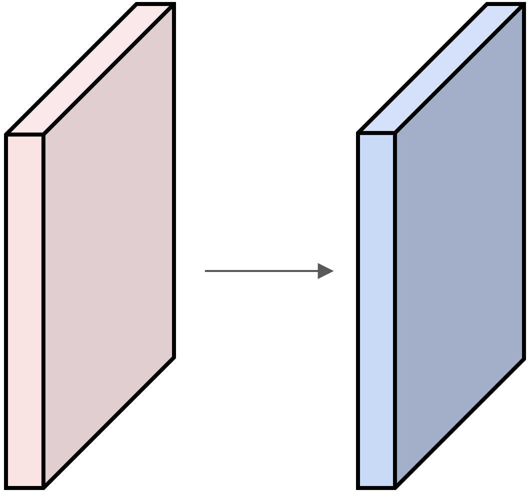
Convolution Example

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.

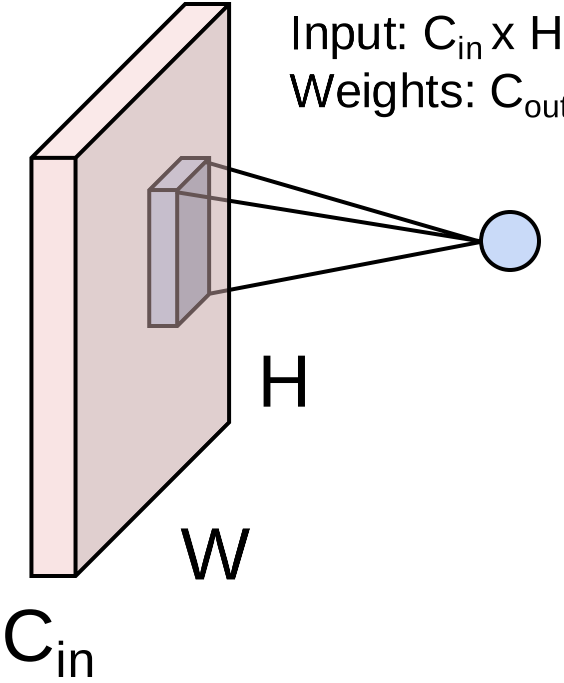
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

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

We didn't talk about groups or dilation…

Other Types of Convolution

So far: 2D Convolution

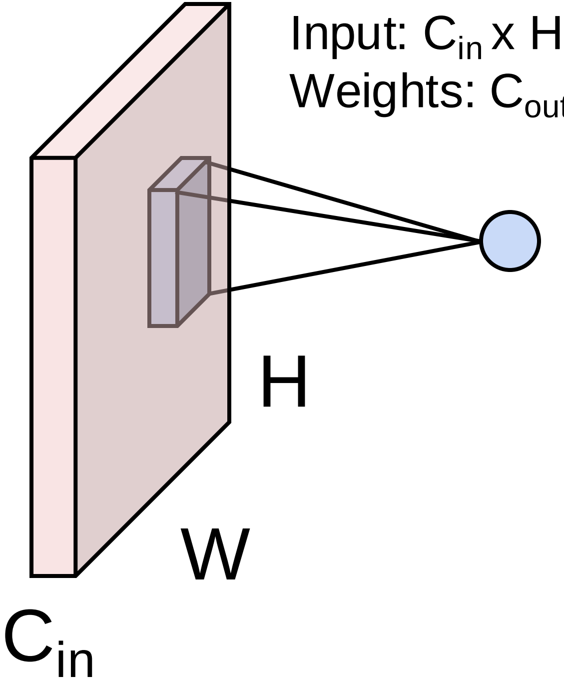


Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K

Other Types of Convolution

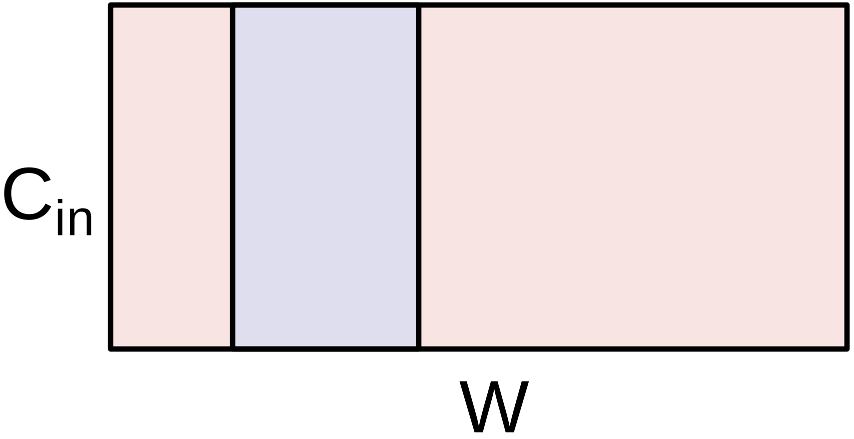
So far: 2D Convolution

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



1D Convolution

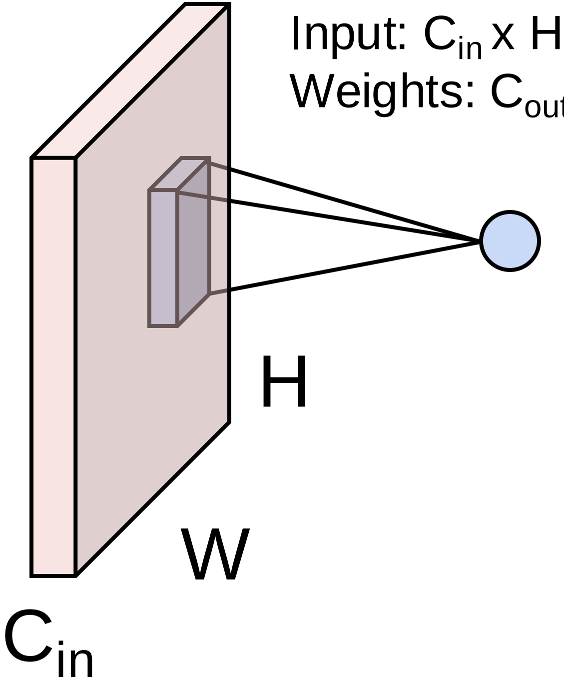
Input: C_{in} x W
Weights: C_{out} x C_{in} x K



Other Types of Convolution

So far: 2D Convolution

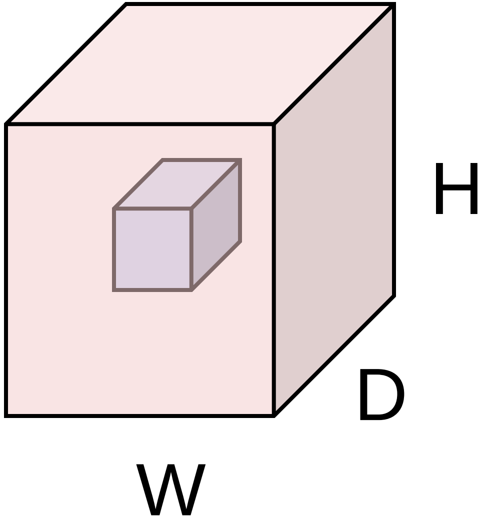
Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



3D Convolution

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

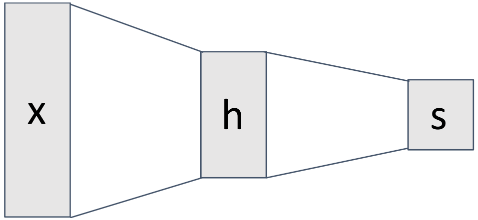
C_{in}-dim vector at each point in the volume



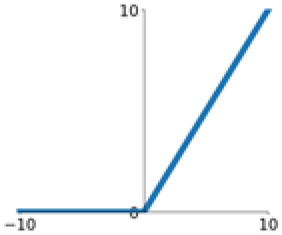
Convolutional Networks

We have already seen these

Fully-Connected Layer

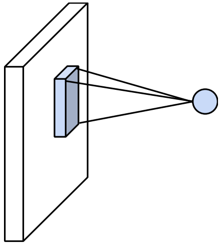


Activation Function

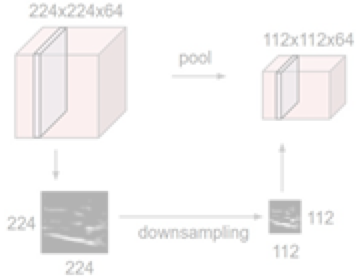


Today: Image-specific operators

Convolution Layer



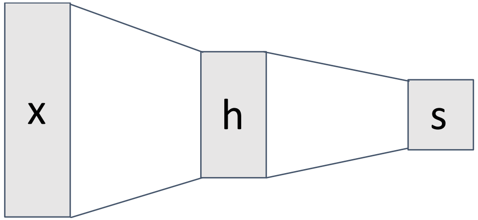
Pooling Layer



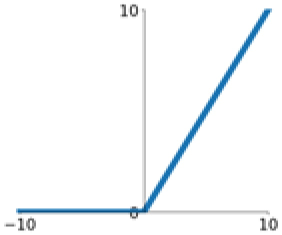
Convolutional Networks

We have already seen these

Fully-Connected Layer

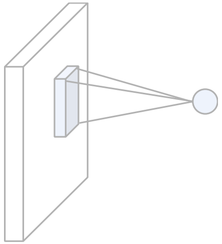


Activation Function

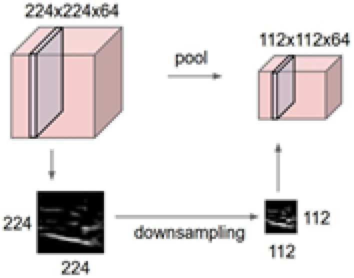


Today: Image-specific operators

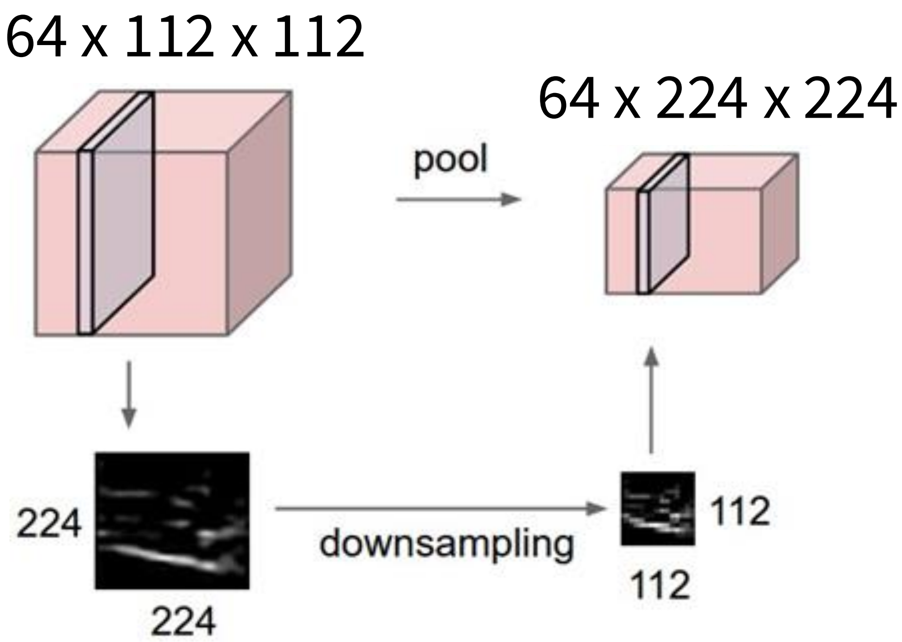
Convolution Layer



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

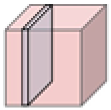
Single depth slice

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

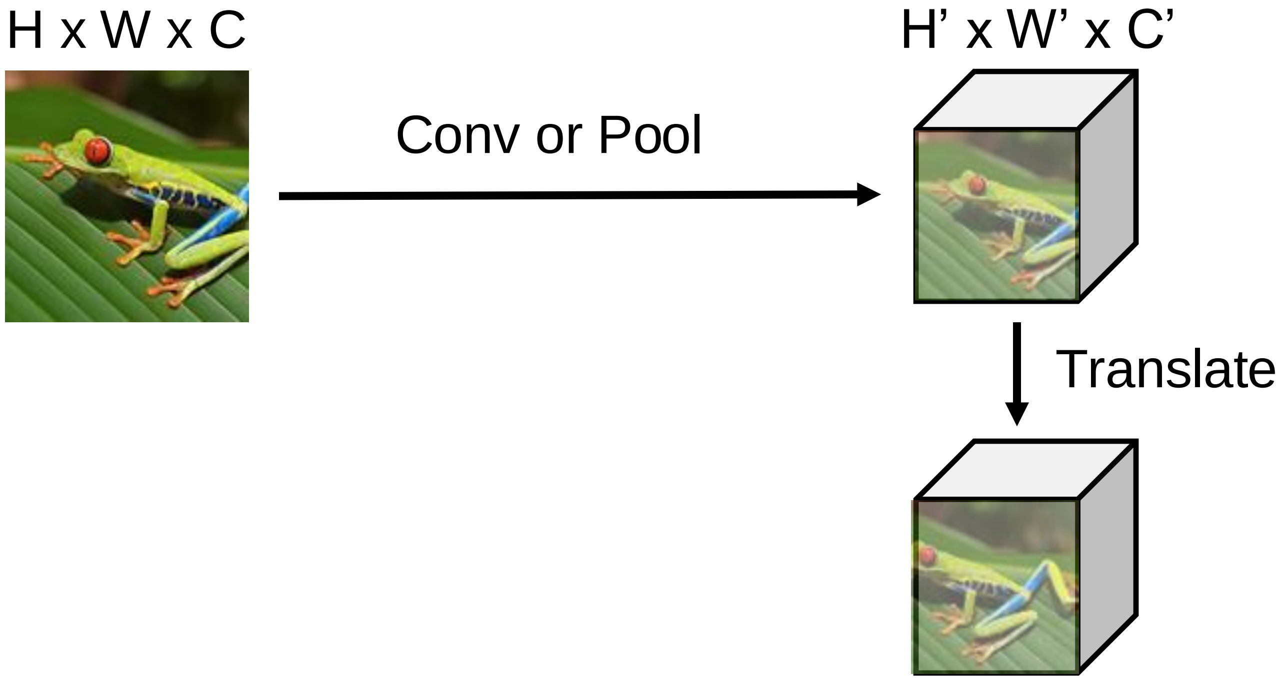
Input: C x H x W
Hyperparameters:
- **Kernel size**: K
- **Stride**: S
- **Pooling function**: max, avg

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

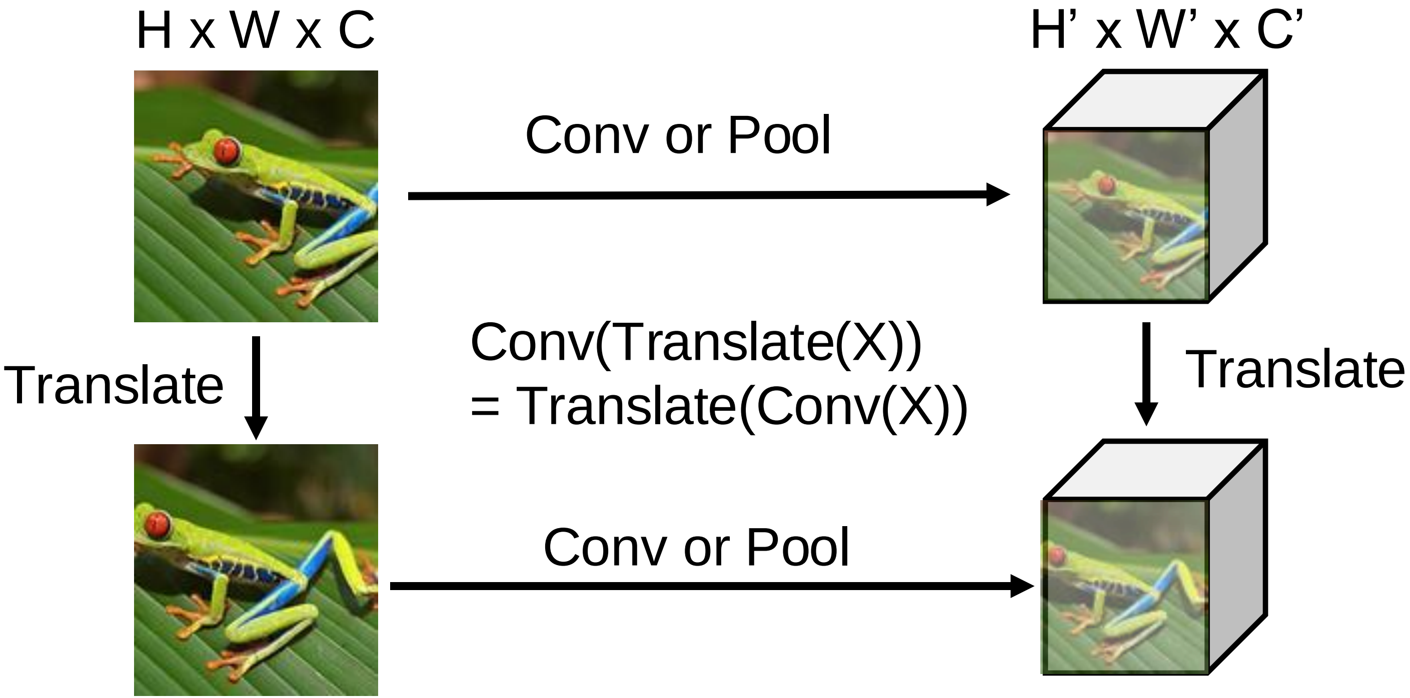
No learnable parameters

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

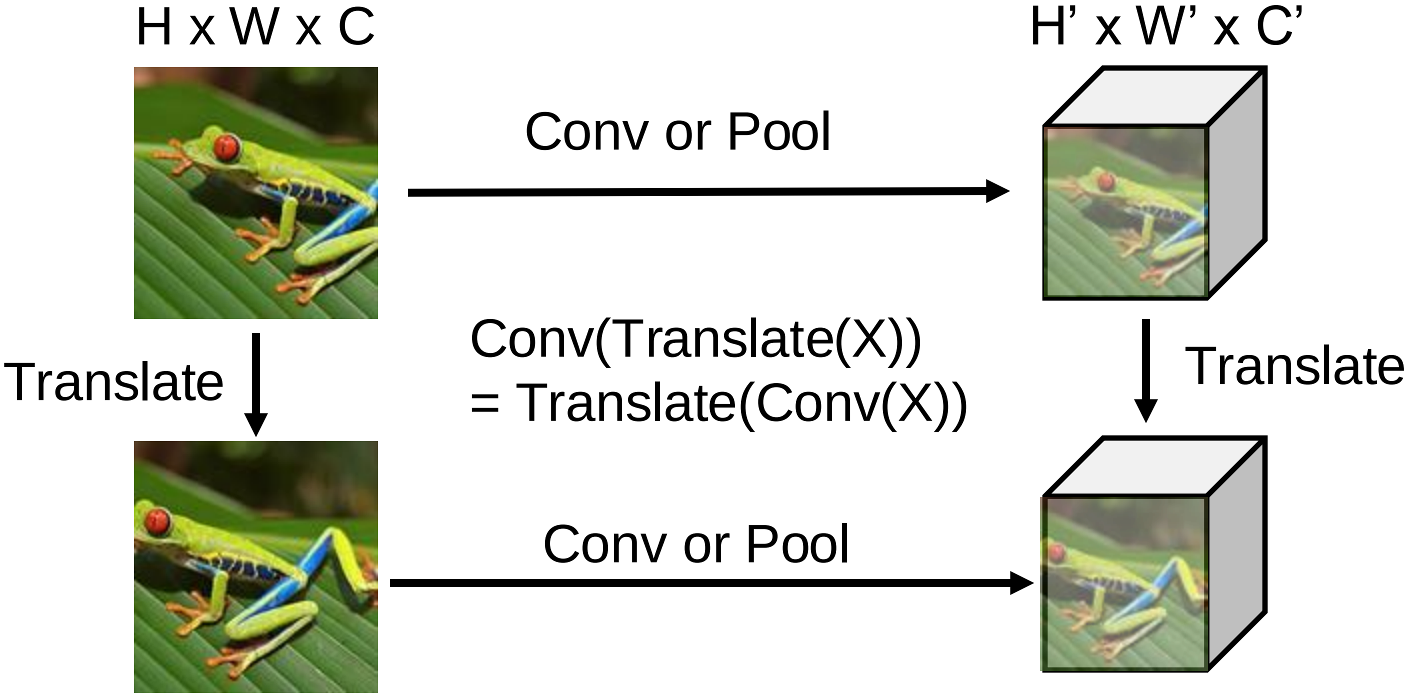
Convolution and Pooling: Translation Equivariance



Convolution and Pooling: Translation Equivariance



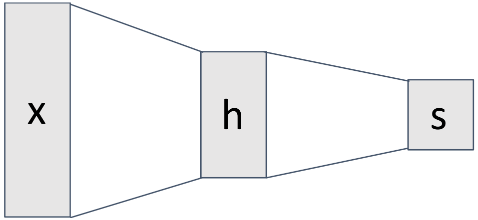
Convolution and Pooling: Translation Equivariance



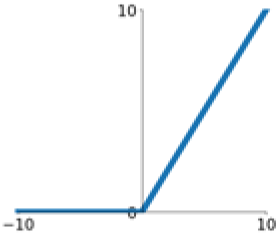
Intuition: Features of images don't depend on their location in the image

Summary: Convolutional Networks

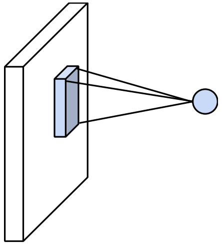
Fully-Connected Layer



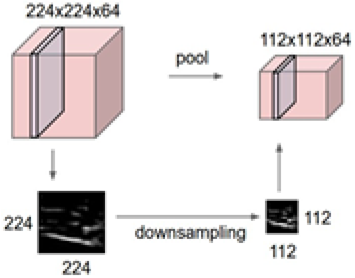
Activation Function



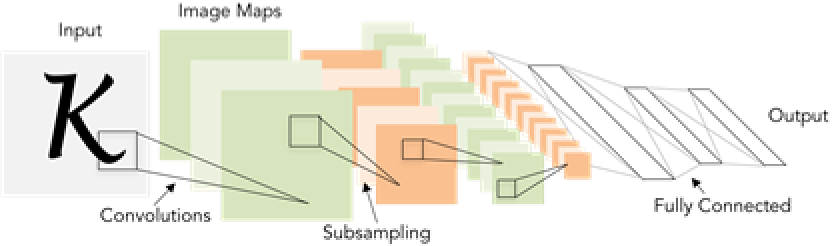
Convolution Layer

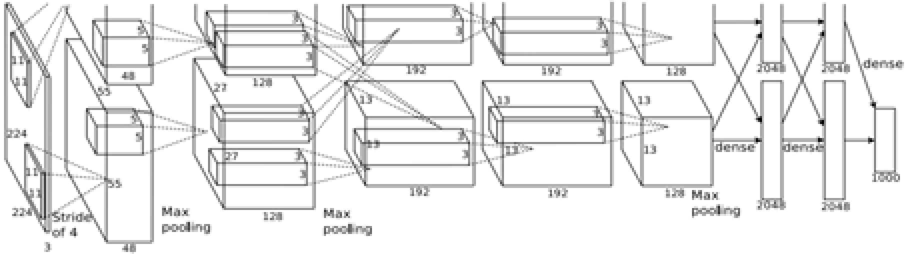


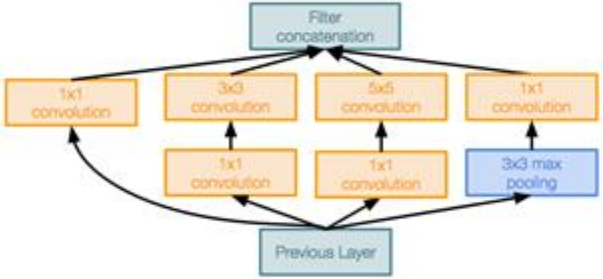
Pooling Layer

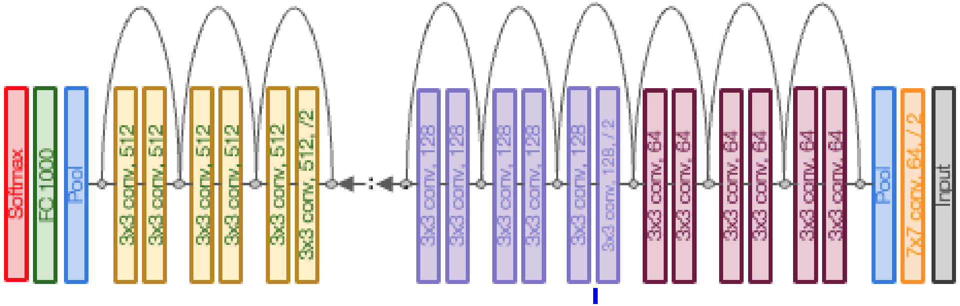


Next time: CNN Architectures







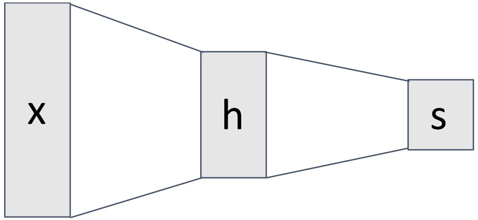


Appendix (Slides from Previous Years)

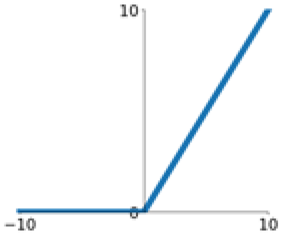
Today: Convolutional Networks

We have already seen these

Fully-Connected Layer

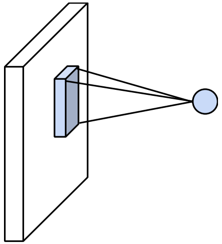


Activation Function

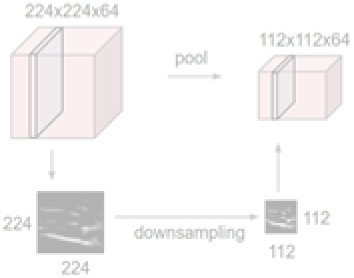


Today: Image-specific operators

Convolution Layer



Pooling Layer



Convolution layer: summary

Let's assume input is W_1 x H_1 x C

Conv layer needs 4 hyperparameters:

- Number of filters K
- The filter size F
- The stride S
- The zero padding P

This will produce an output of W_2 x H_2 x K

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:
- Number of filters K
- The filter size F
- The stride S
- The zero padding P

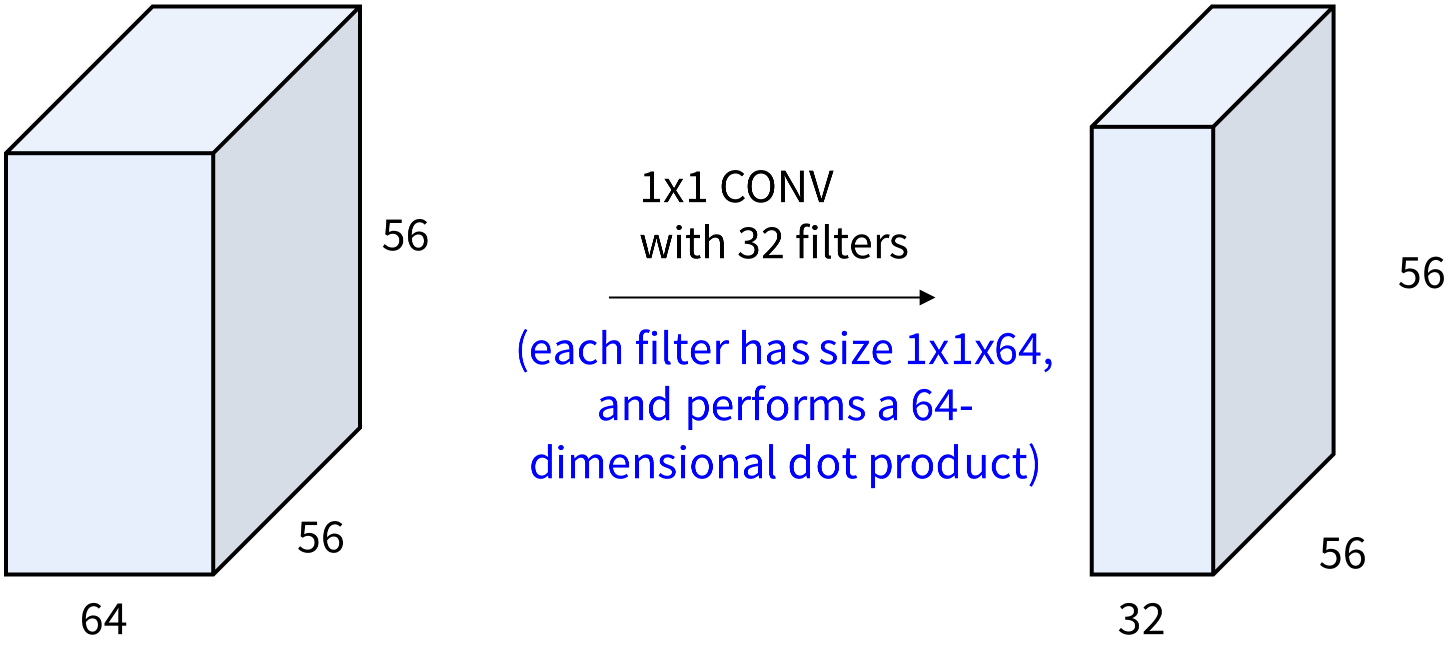
This will produce an output of W_2 x H_2 x K
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

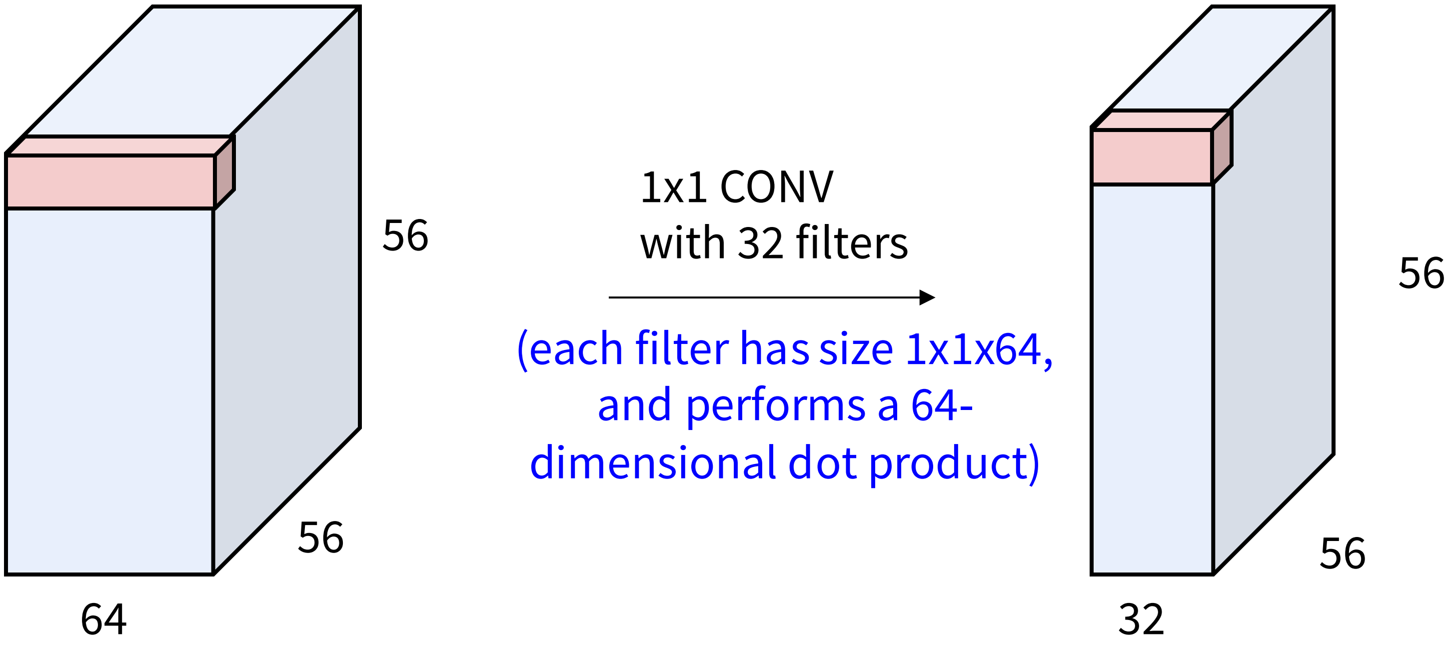
Common settings:

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

(btw, 1x1 convolution layers make perfect sense)



(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in PyTorch

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [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:

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

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
    - At groups=1, all inputs are convolved to all outputs.
    - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
    - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

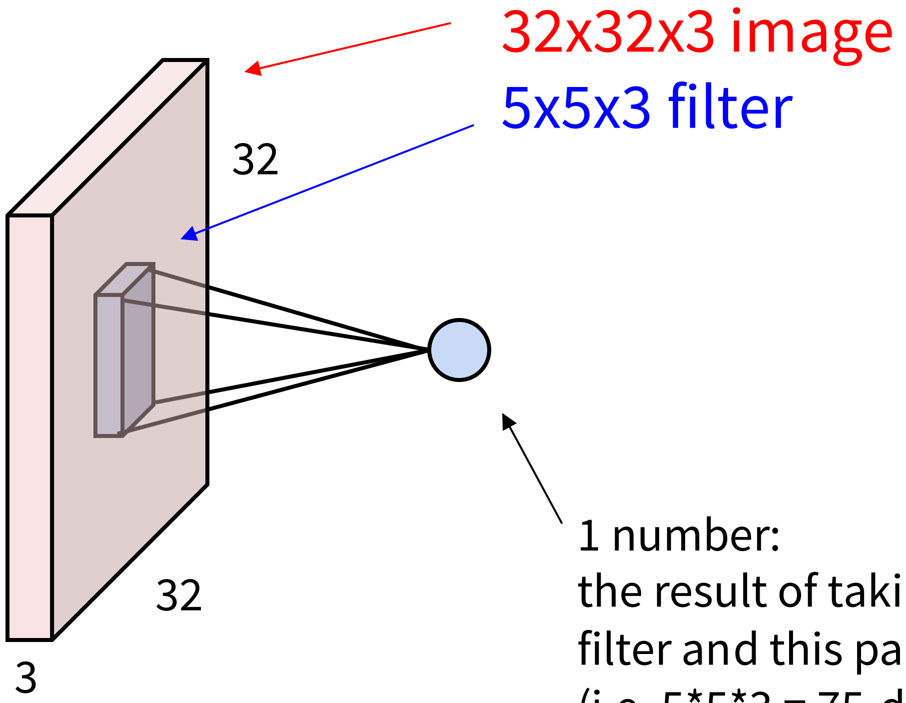
The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

Conv layer needs 4 hyperparameters:

- Number of filters K
- The filter size F
- The stride S
- The zero padding P

The brain/neuron view of CONV Layer

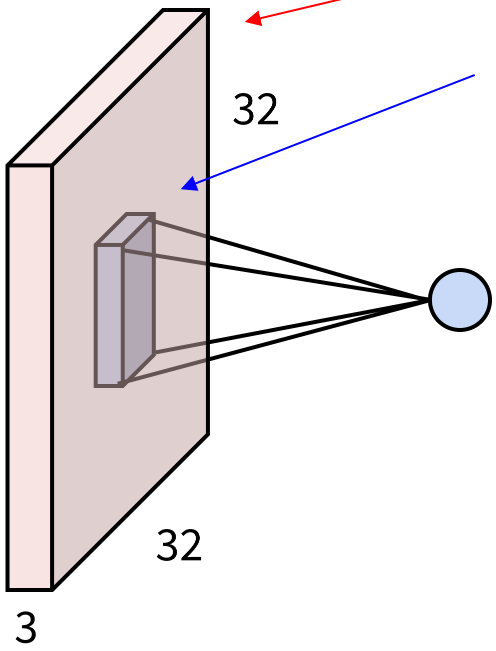


1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

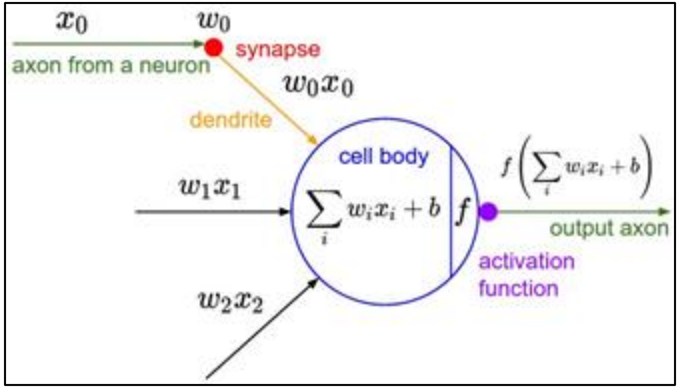
The brain/neuron view of CONV Layer

32x32x3 image

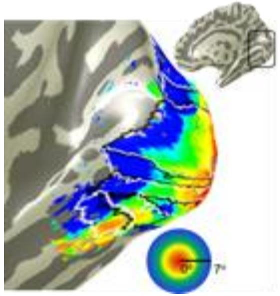
5x5x3 filter



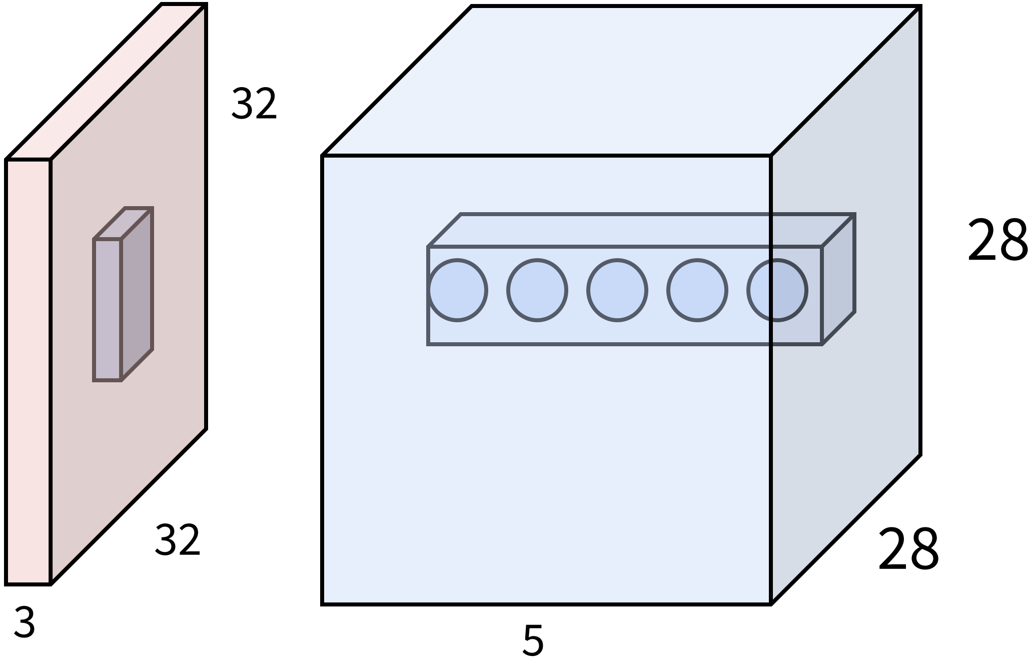
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

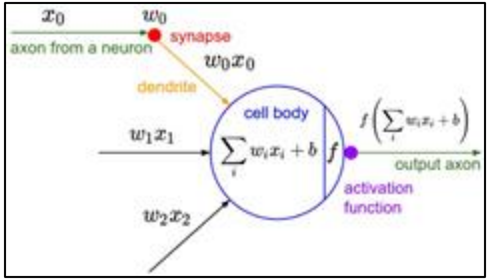


It's just a neuron with local connectivity...



The brain/neuron view of CONV Layer





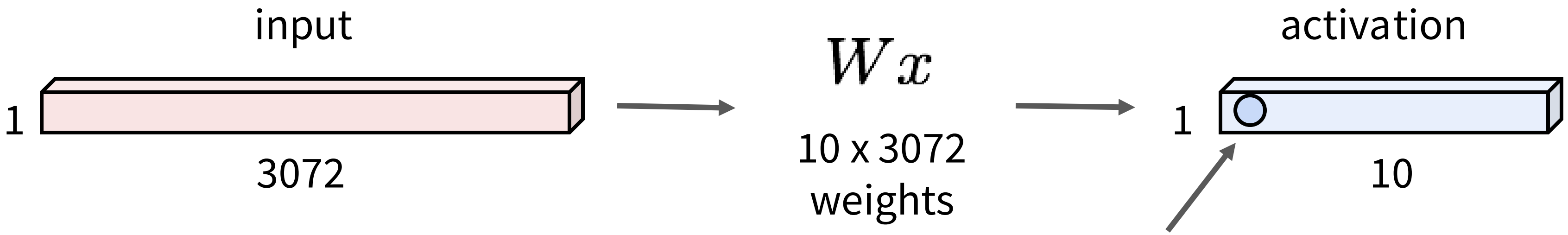
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid
(28x28x5)

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

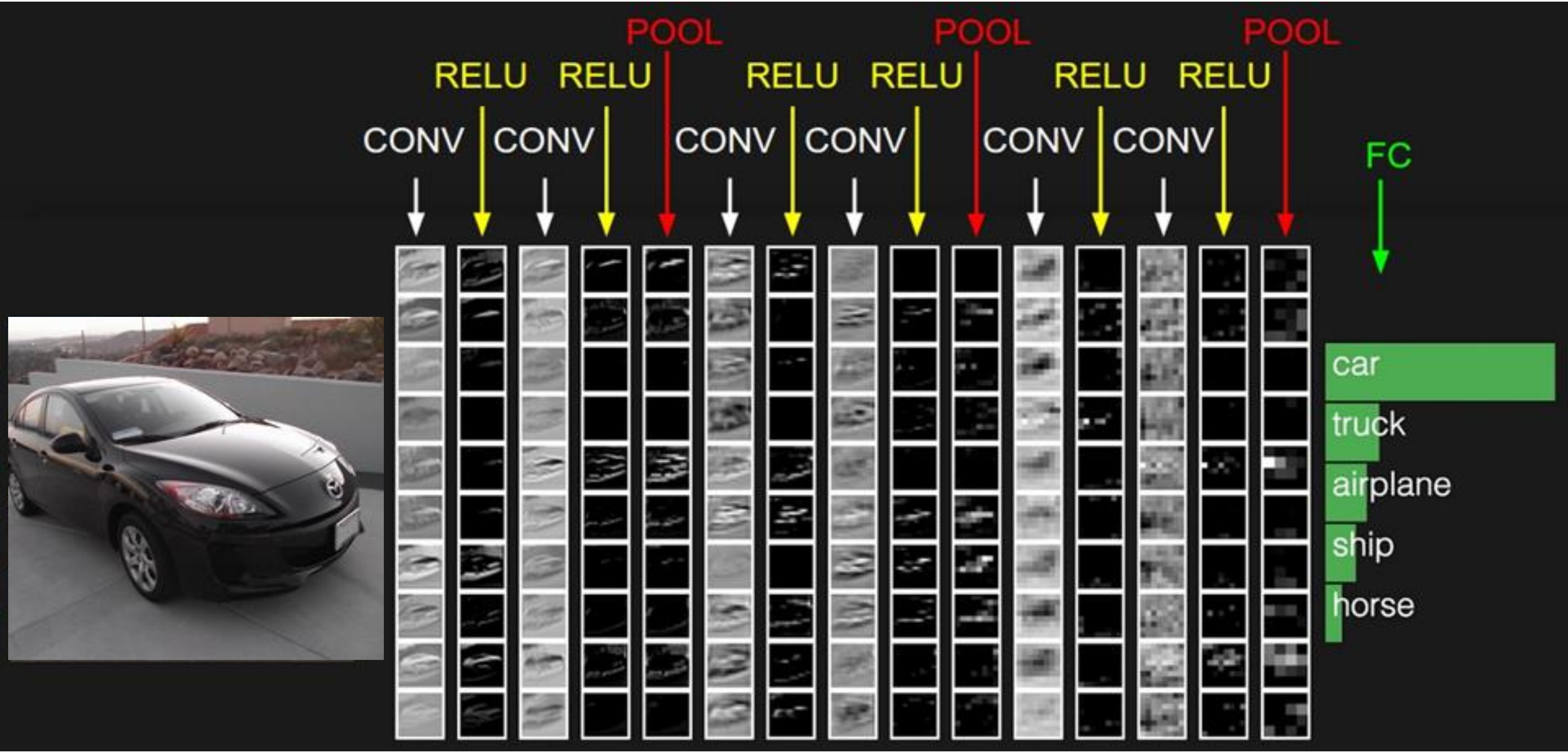
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

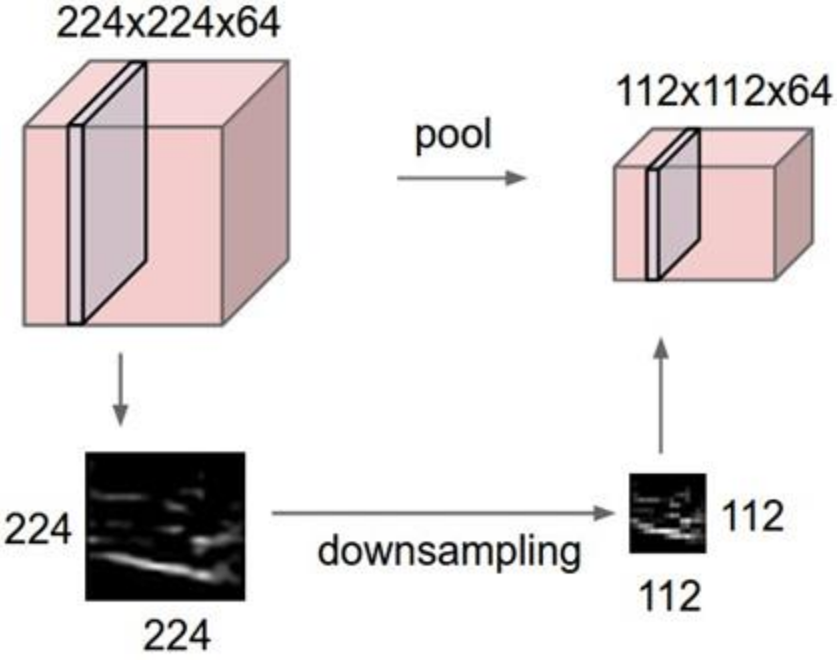


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

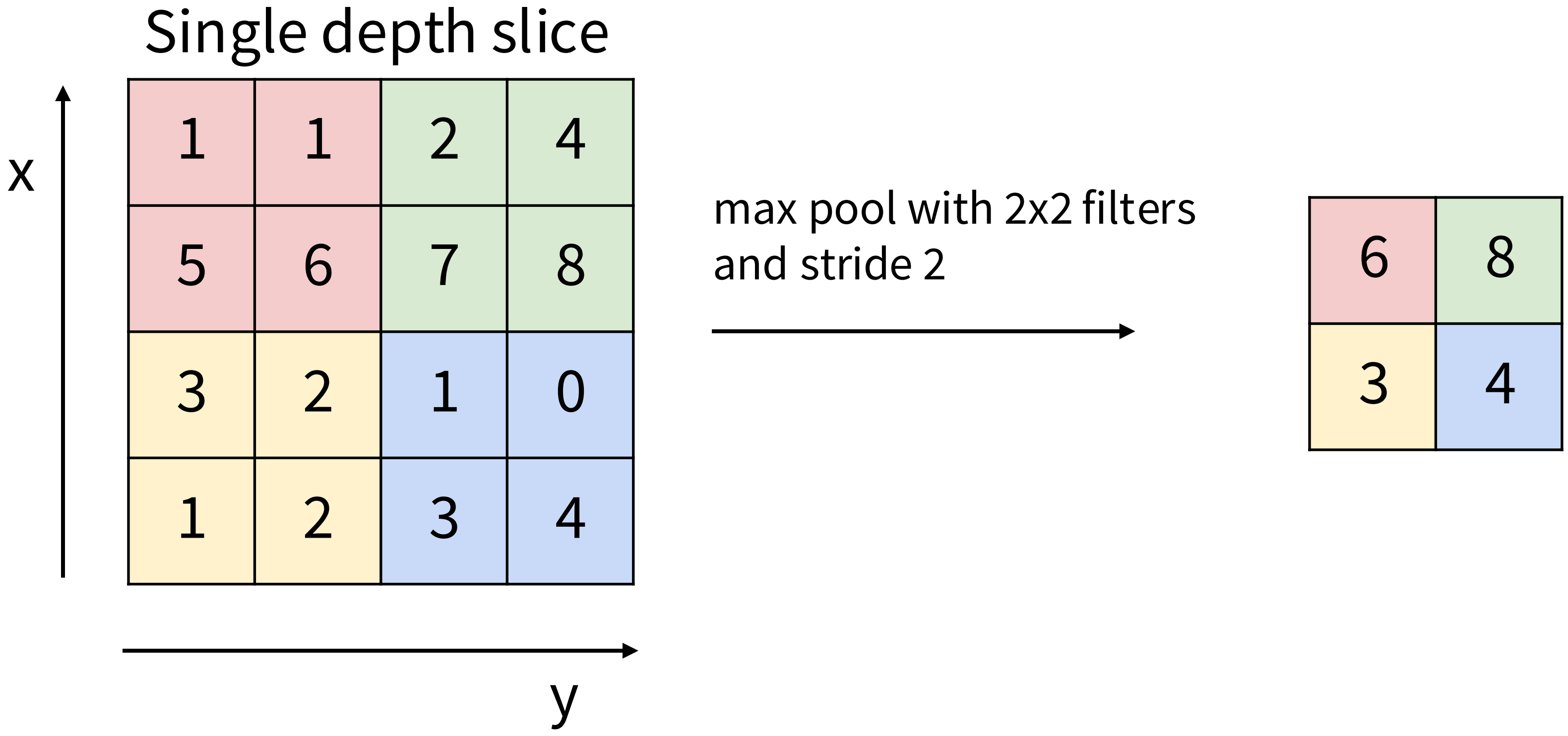

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

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



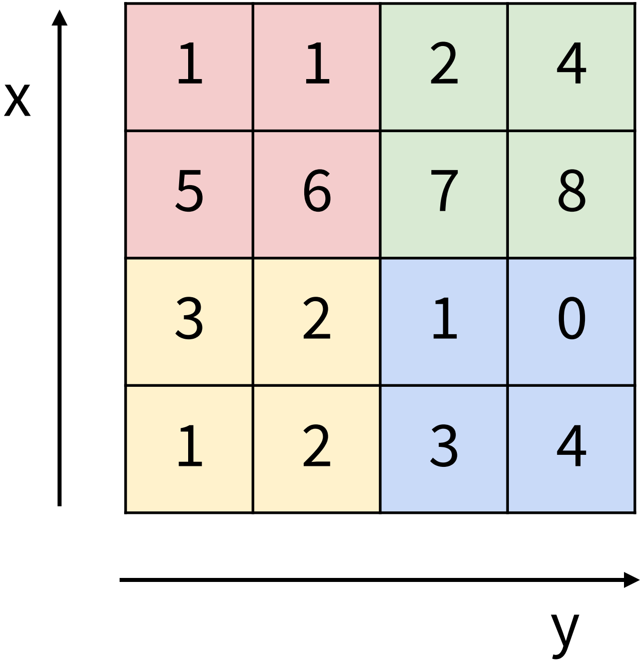
MAX POOLING



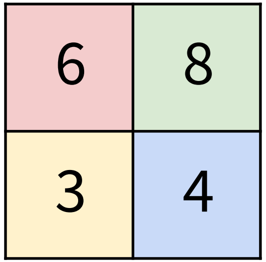
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2



- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 2 hyperparameters:
- The spatial extent F
- The stride S

This will produce an output of W_2 x H_2 x C where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

Summary

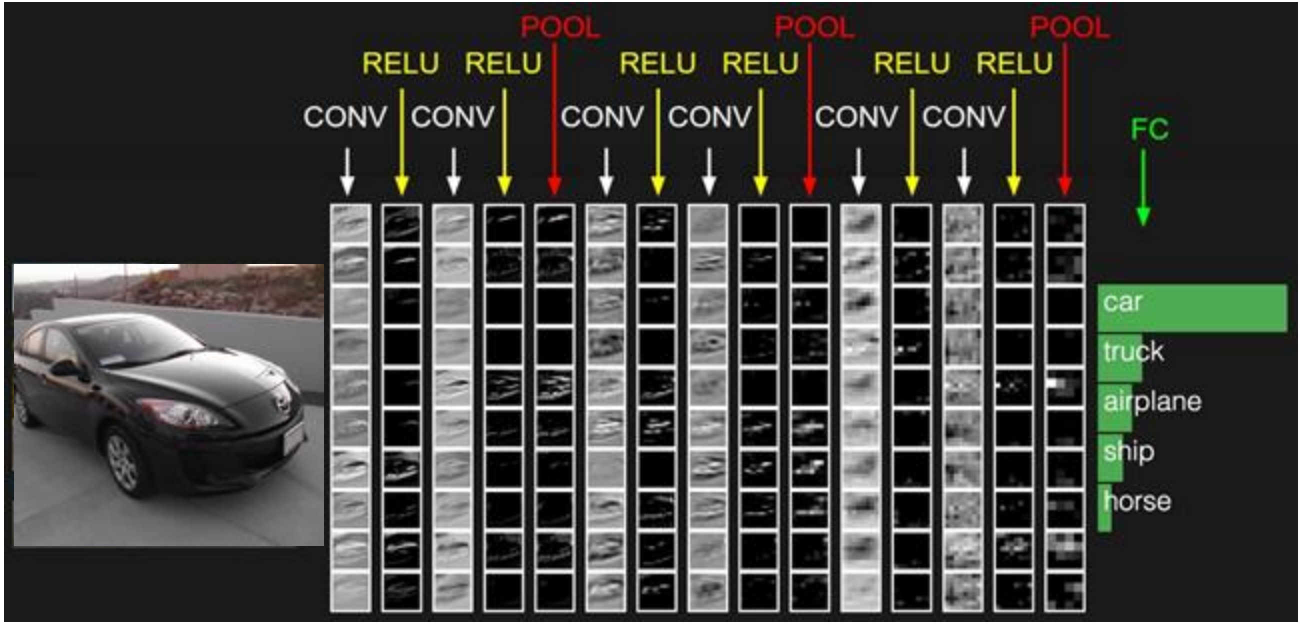
- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.
- But recent advances such as ResNet/GoogLeNet have challenged this paradigm

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

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

Deep Learning Overview

1. Encode your problem as y = f(x) where x and y are grids of numbers. Get a dataset of (x, y) pairs.

Image Classification

Image: x

[32 x 32 x 3]

Scores: y

Scores for each class

[10]

Deep Learning Overview

1. Encode your problem as y = f(x) where x and y are grids of numbers. Get a dataset of (x, y) pairs.
2. Define a **loss function** $L(y_{pred}, y_{gt})$ that measures the correctness of predictions with a single number

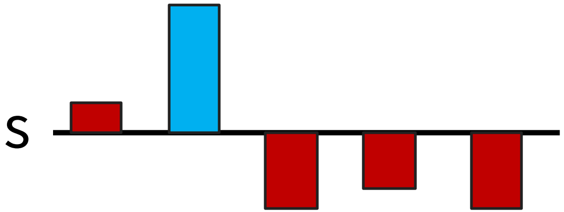
Softmax Loss

s: vector of n scores

y: int in [0, n)

$$L(s, y) = -\log\left(\frac{e^{-s_y}}{\sum_i e^{-s_i}}\right)$$

Scores for y should be +inf



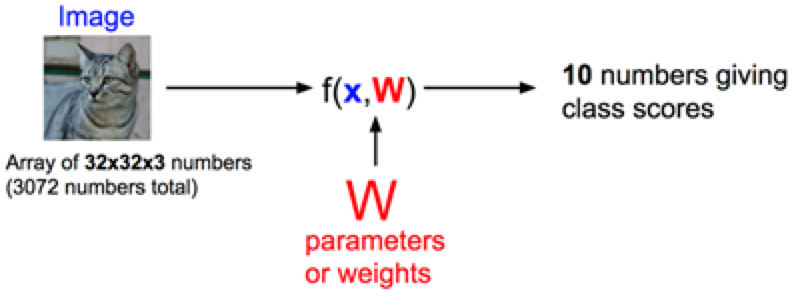
Others should be -inf

Deep Learning Overview

1. Encode your problem as y = f(x) where x and y are grids of numbers. Get a dataset of (x, y) pairs.
2. Define a **loss function** $L(y_{pred}, y_{gt})$ that measures the correctness of predictions with a single number
3. Define a **computational graph** that predicts y from x using learnable weights w

Linear Classifiers

$$f(x, w, b) = Wx + b$$

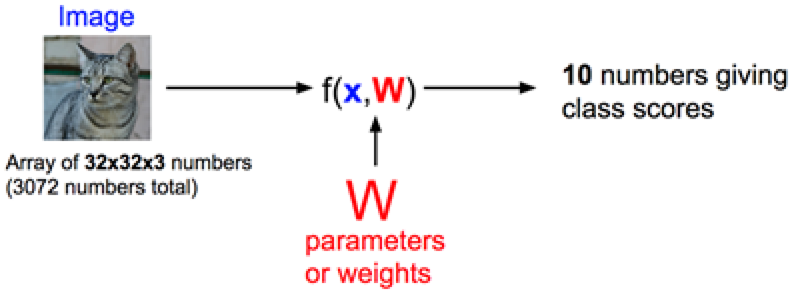


Deep Learning Overview

1. Encode your problem as y = f(x) where x and y are grids of numbers. Get a dataset of (x, y) pairs.
2. Define a **loss function** $L(y_{pred}, y_{gt})$ that measures the correctness of predictions with a single number
3. Define a **computational graph** that predicts y from x using learnable weights w

Linear Classifiers

$$f(x, w, b) = Wx + b$$



Deep Learning Overview

1. Encode your problem as y = f(x) where x and y are grids of numbers. Get a dataset of (x, y) pairs.
2. Define a **loss function** $L(y_{pred}, y_{gt})$ that measures the correctness of predictions with a single number
3. Define a **computational graph** that predicts y from x using learnable weights w
4. Compute gradients dL/dw using **backpropagation**
5. Find w that minimizes the loss using **optimization algorithms**

A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

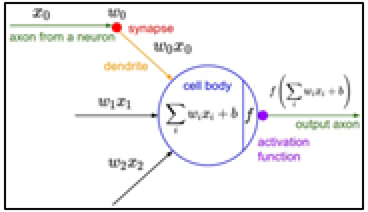
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

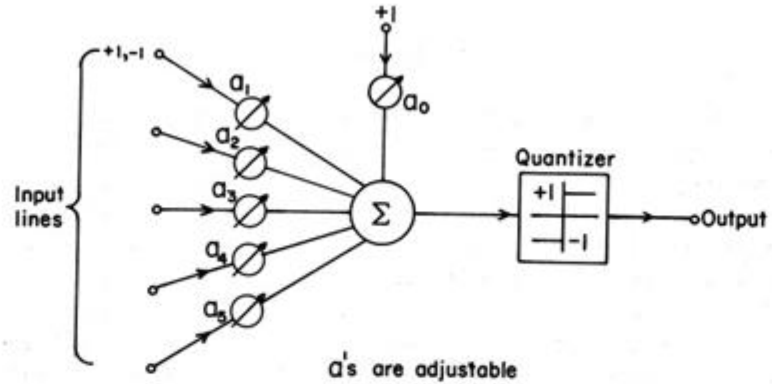
update rule:

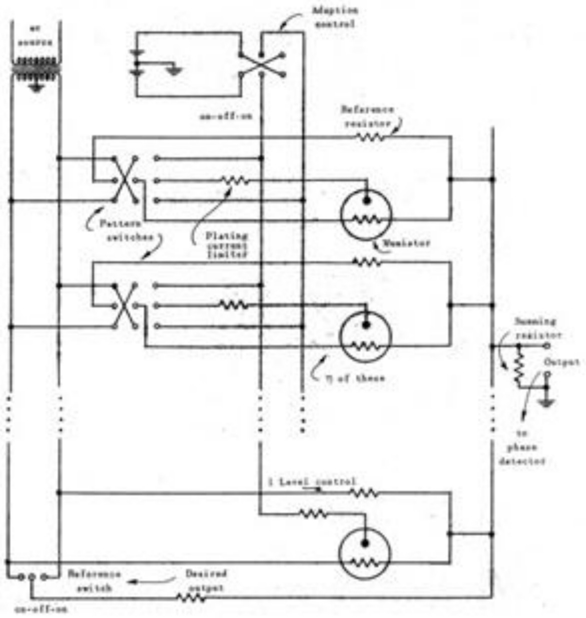
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$



Frank Rosenblatt, ~1957: Perceptron

A bit of history...





Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from [Widrow 1960, Stanford Electronics Laboratories Technical Report] with permission from [Stanford University Special Collections].

A bit of history...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

recognizable math

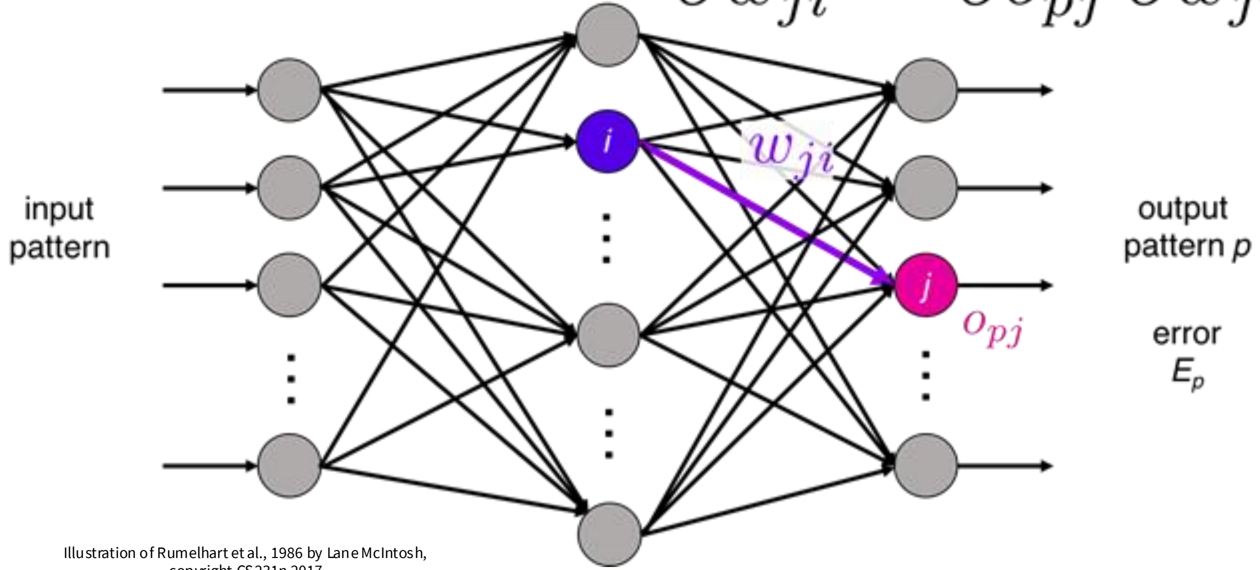


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

Rumelhart et al., 1986: First time back-propagation became popular

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

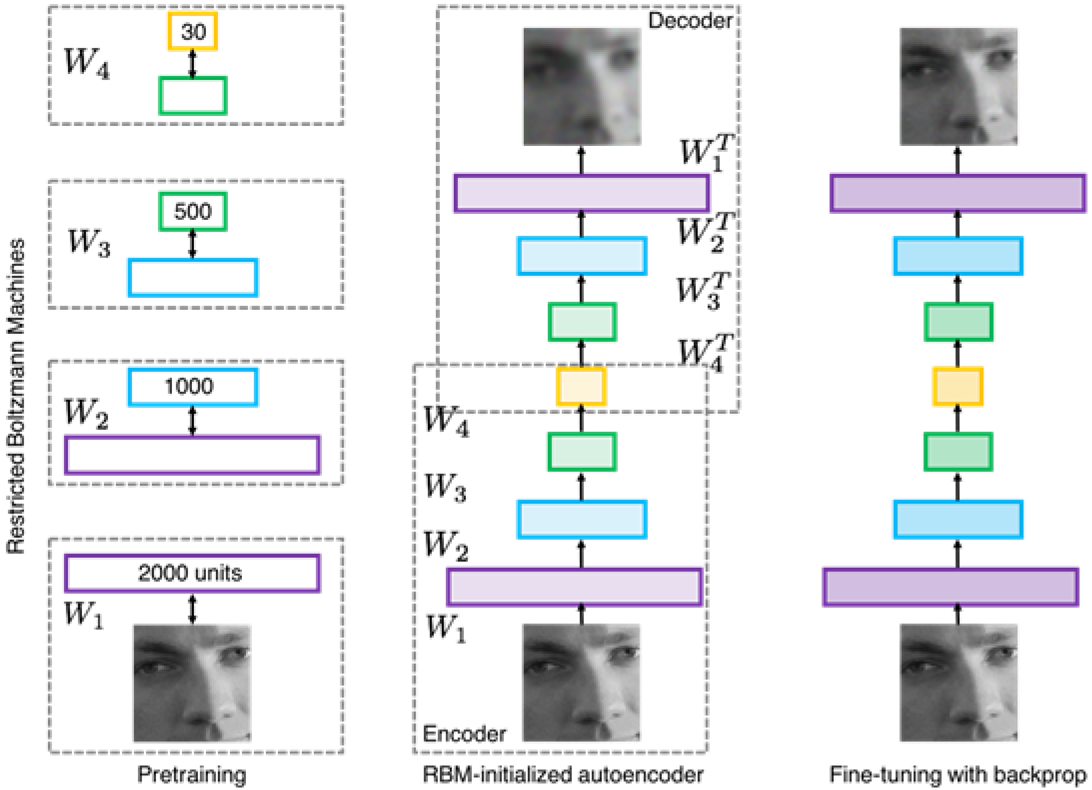


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

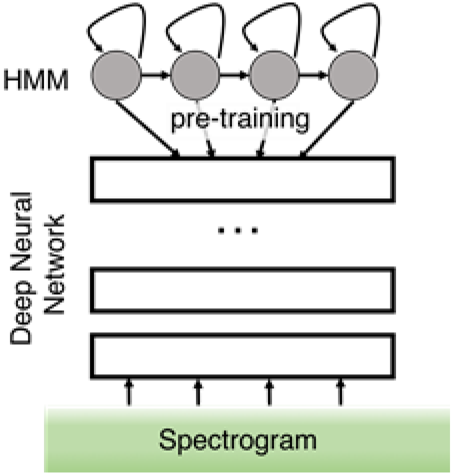
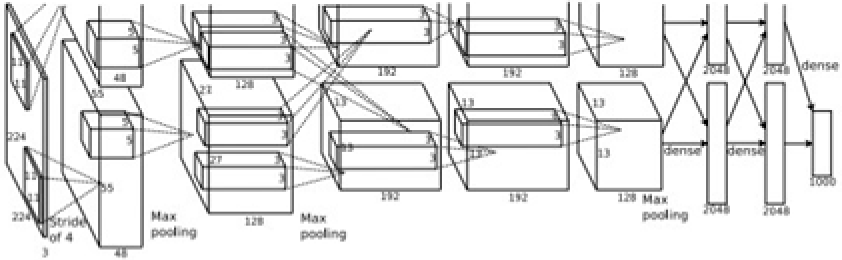
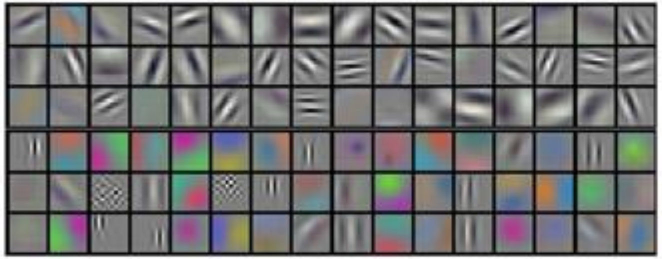


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





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

A bit of history:

Hubel & Wiesel,

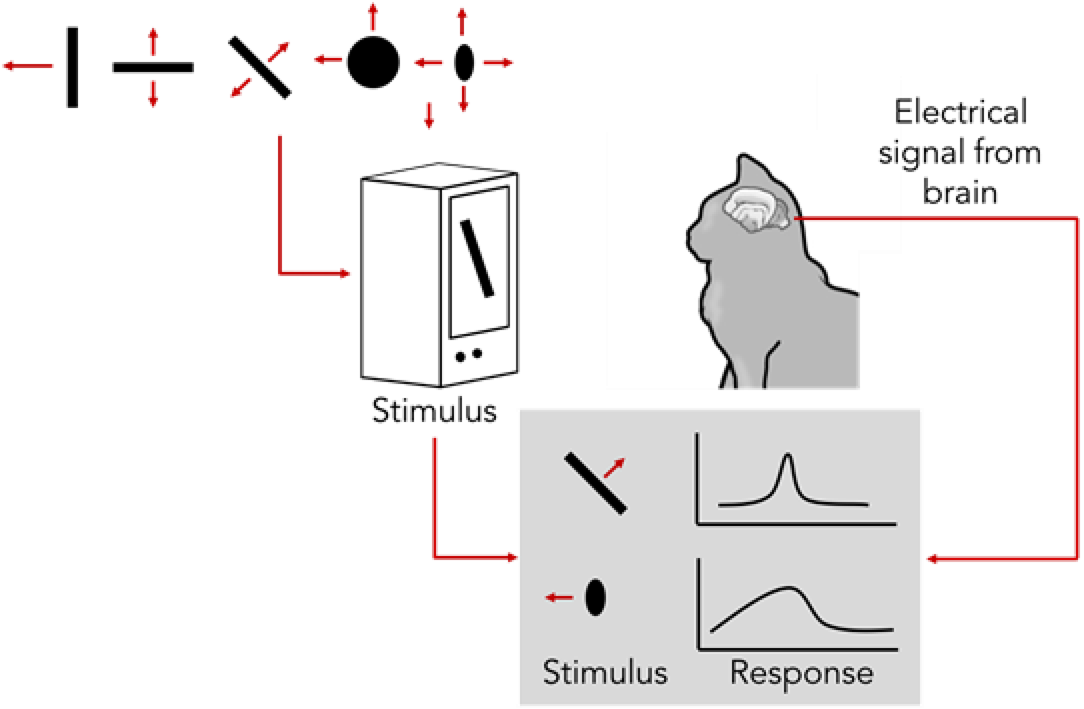
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962

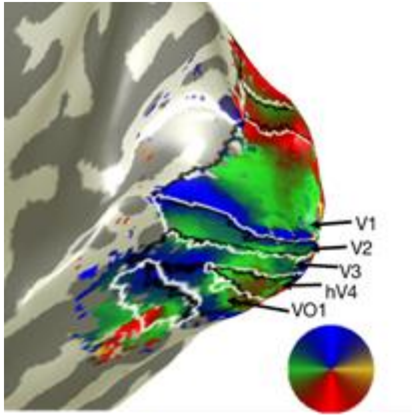
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

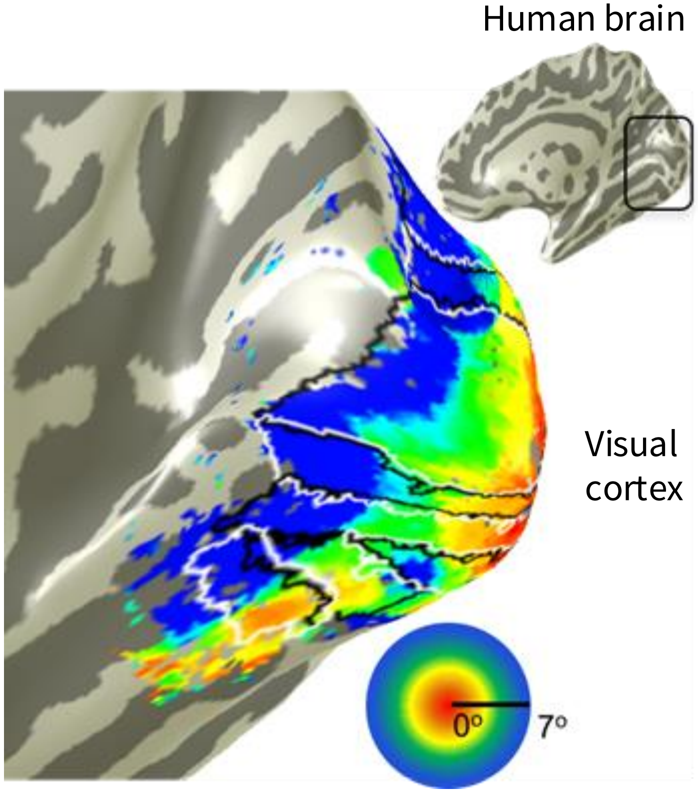
1968...



A bit of history

Topographical mapping in the cortex:
nearby cells in cortex represent
nearby regions in the visual field





Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Hierarchical organization

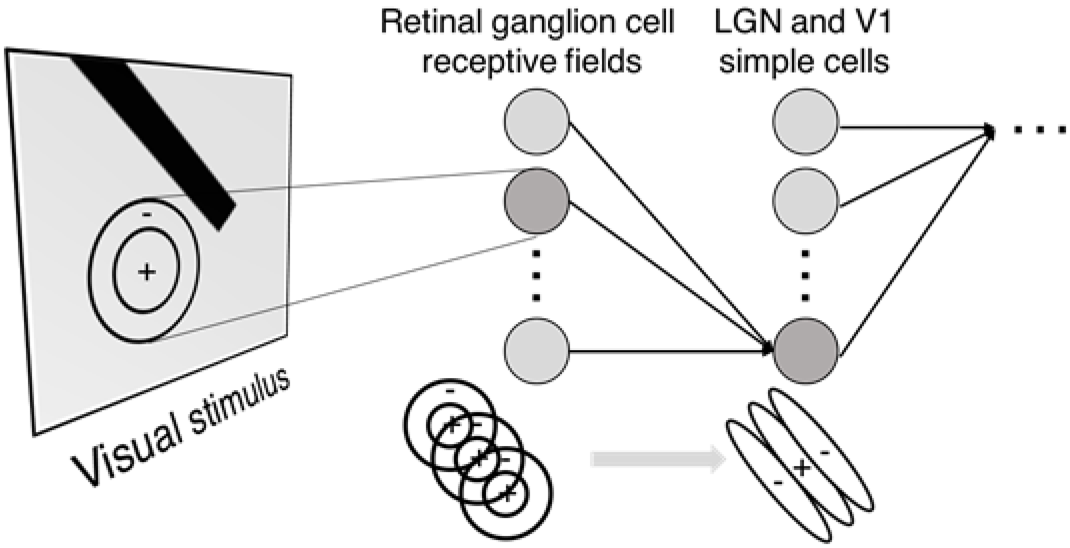
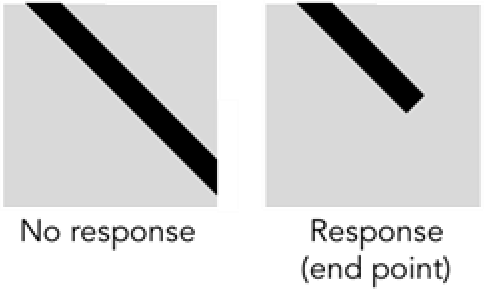


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

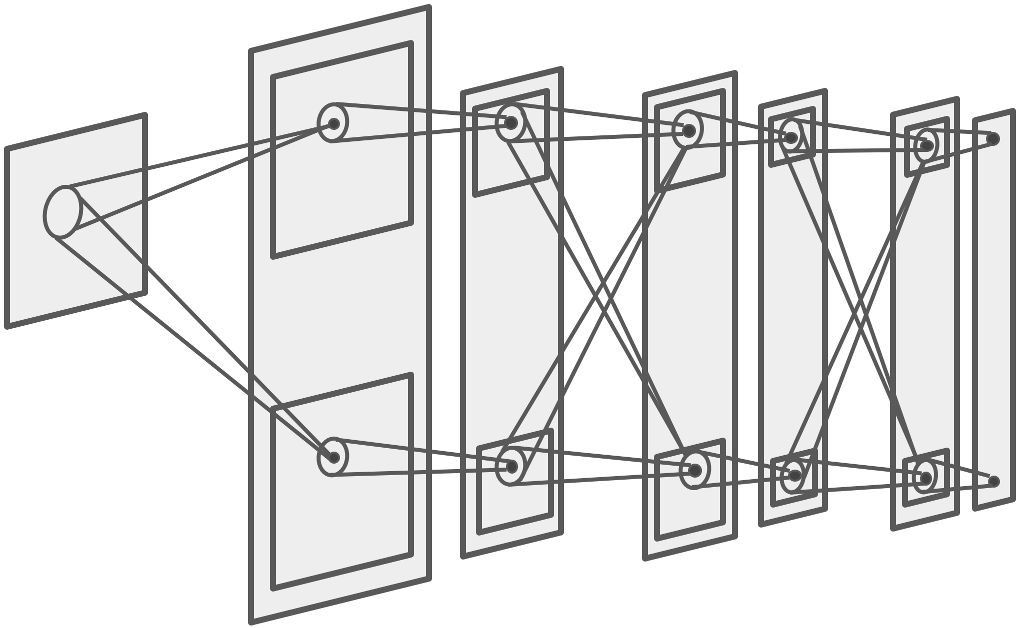
Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point



A bit of history:

Neocognitron
[Fukushima 1980]



"sandwich" architecture (SCSCSC...)
simple cells: modifiable parameters
complex cells: perform pooling