

Convolutional Neural Networks

Based on [Lecture 4: Neural Networks and Backpropagation](#) and
[Lecture 5: Image Classification with CNNs](#)

CS231n: Deep Learning for Computer Vision
Stanford - Spring 2023,

Beginnings

Thresholded Logic Unit

1943

Perceptron

1957

Adaline

1960

1st Neural Winter

XOR Problem

1969

Multilayer Backprop

1982

CNNs

1986

LSTMs

1989

1997

2nd Neural Winter

SVMs

1995

GPU Era

Deep Nets

2006

Alex Net

2012

1940



S. McCulloch - W. Pitts

1950



R. Rosenblatt

1960



B. Widrow - M. Hoff

1970



M. Minsky - S. Papert

1980



P. Werbos
D. Rumelhart - G. Hinton - R. Williams

1990



Y. Lecun
J. Schmidhuber

2000

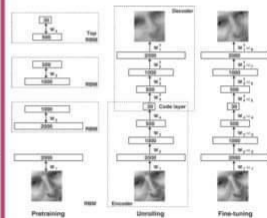
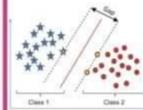
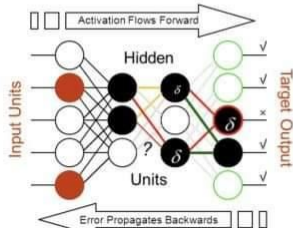
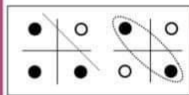
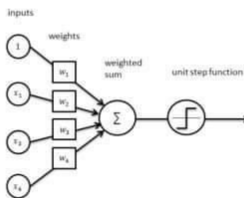
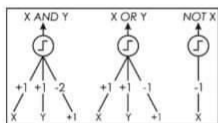


C. Cortes - V. Vapnik

2010



R. Salakhutdinov - J. Hinton - A. Krizhevsky - I. Sutskever



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI)—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

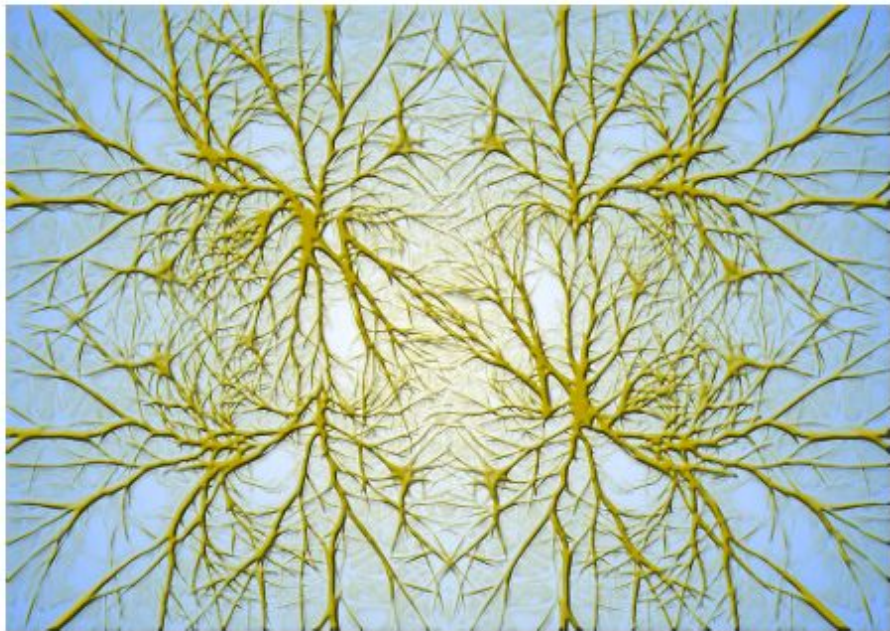
Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

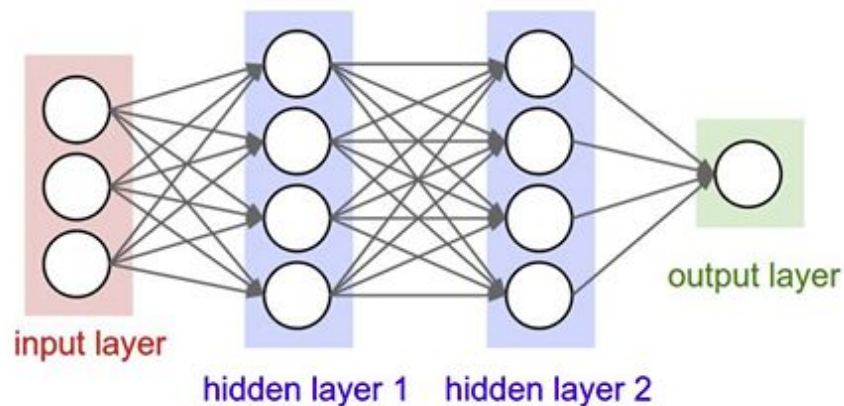
The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Biological Neurons: Complex connectivity patterns

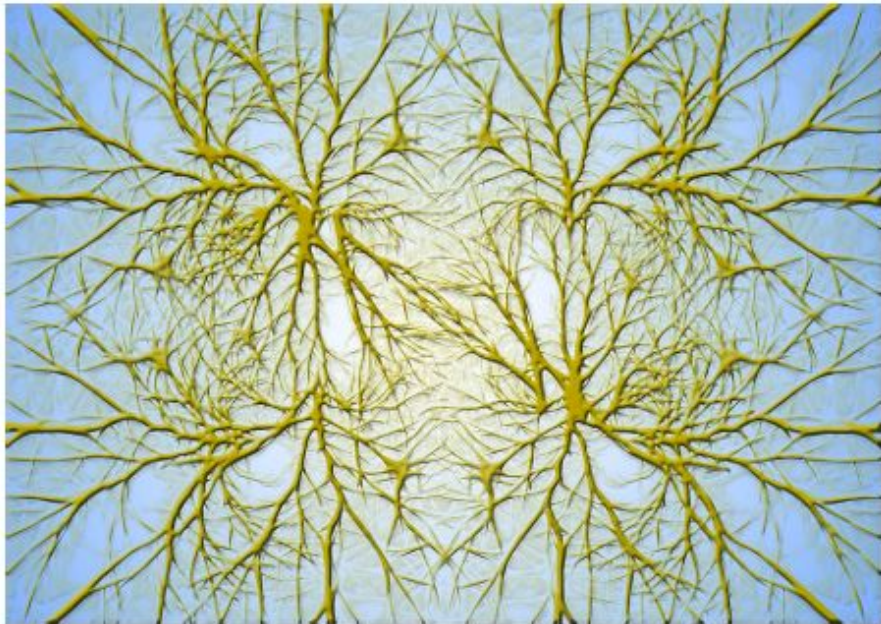


[This image is CC0 Public Domain](#)

Neurons in a neural network: Organized into regular layers for computational efficiency

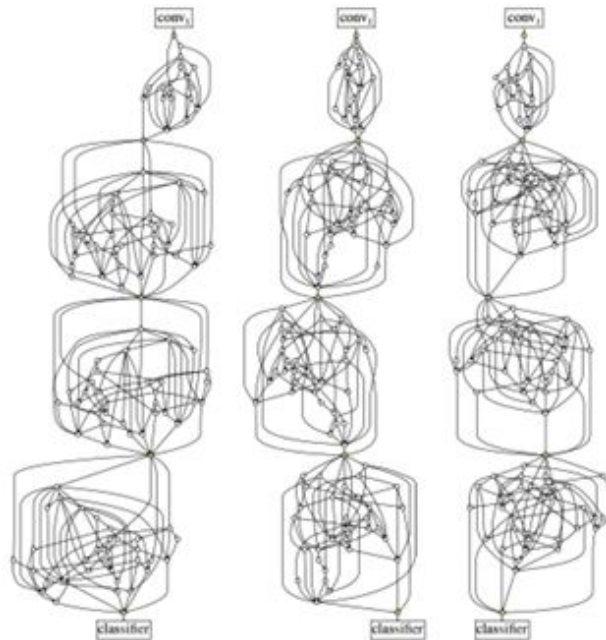


Biological Neurons: Complex connectivity patterns



[This image is CC0 Public Domain](#)

But neural networks with random connections can work too!



Xie et al, "Exploring Randomly Wired Neural Networks for Image Recognition", arXiv 2019

Be very careful with your brain analogies!

Biological Neurons:

- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system

[Dendritic Computation. London and Hausser]

Image Classification: A core task in Computer Vision



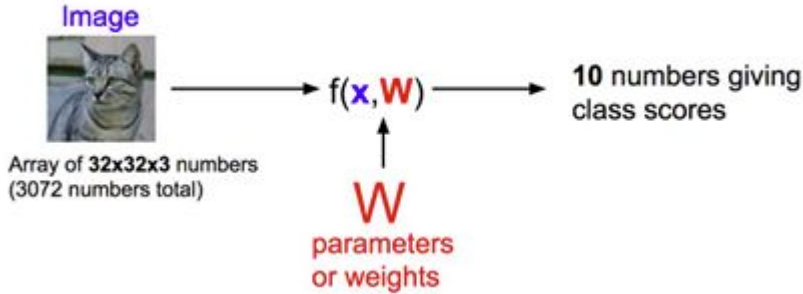
[This image](#) by [Nikita](#) is
licensed under [CC-BY 2.0](#)

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



cat
dog
bird
deer
truck

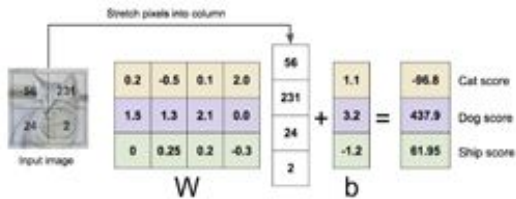
Image Classification with Linear Classifier



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

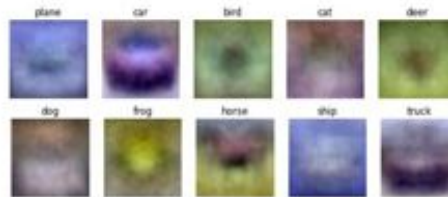
Algebraic Viewpoint

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



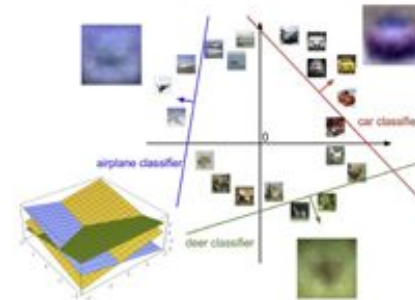
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



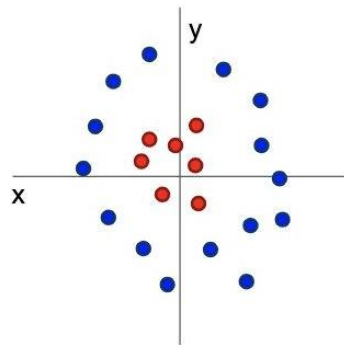
Problem: Linear Classifiers are not very powerful

Visual Viewpoint



Linear classifiers learn
one template per class

Geometric Viewpoint



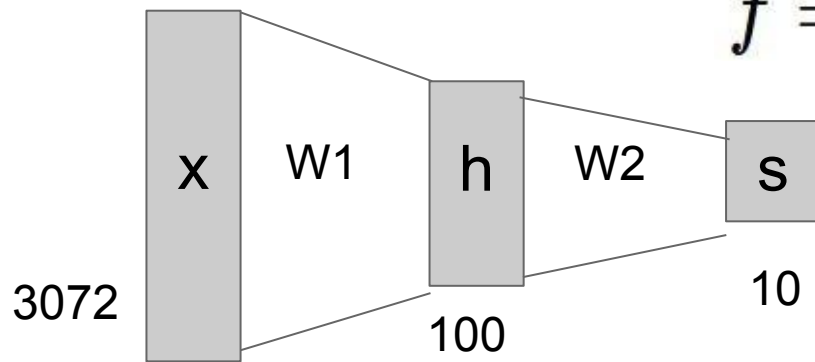
Linear classifiers
can only draw linear
decision boundaries

Neural Networks

Linear score function: 2-layer Neural Network

$$f = Wx$$

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



Pixel space

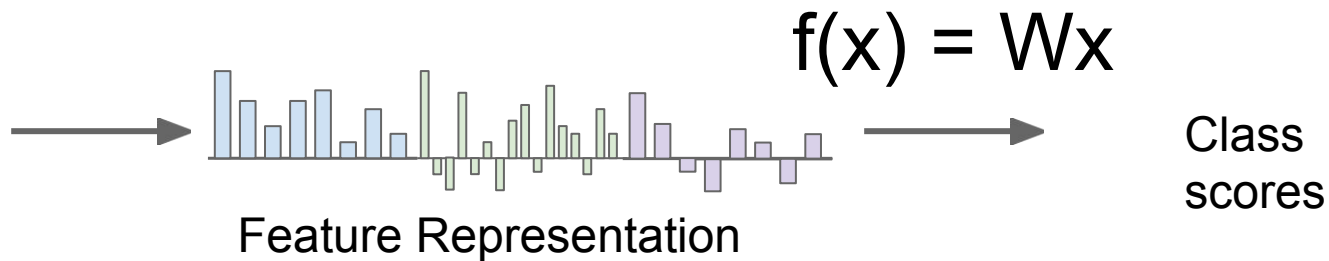


Class
scores

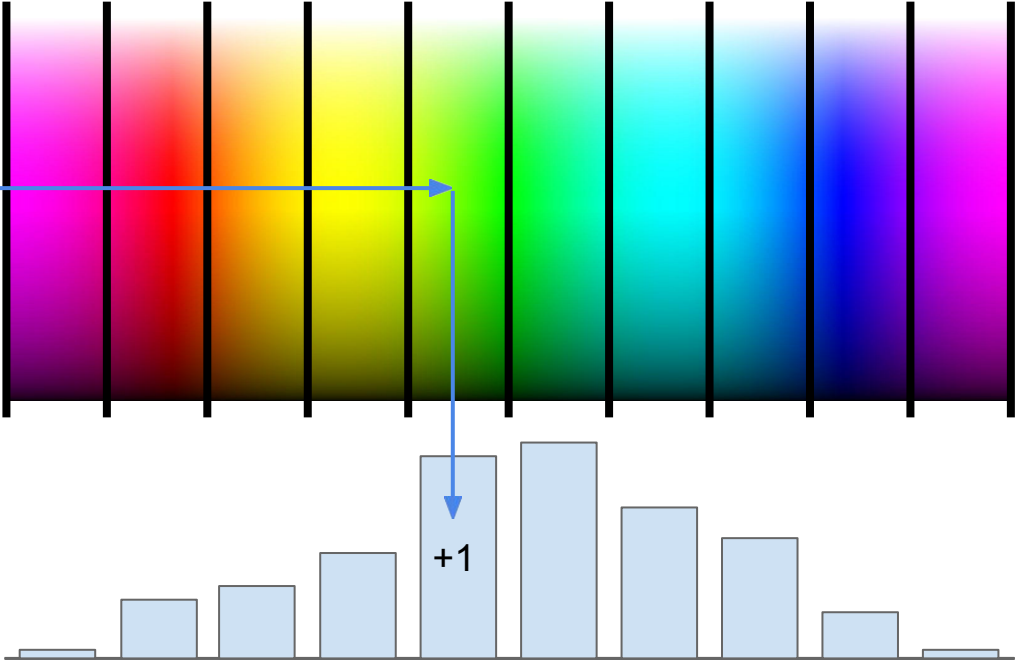
$$f(x) = Wx$$



Image features



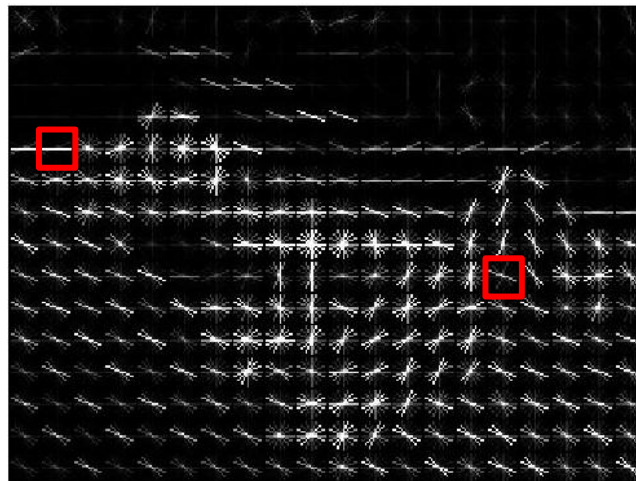
Example: Color Histogram



Example: Histogram of Oriented Gradients (HoG)



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



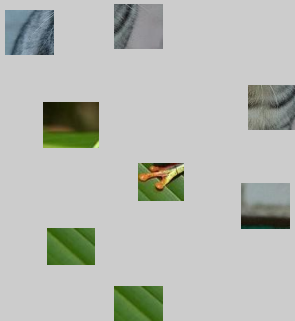
Example: 320x240 image gets divided
into 40x30 bins; in each bin there are
9 numbers so feature vector has
 $30 \times 40 \times 9 = 10,800$ numbers

Example: Bag of Words

Step 1: Build codebook



Extract random patches



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



Step 2: Encode images

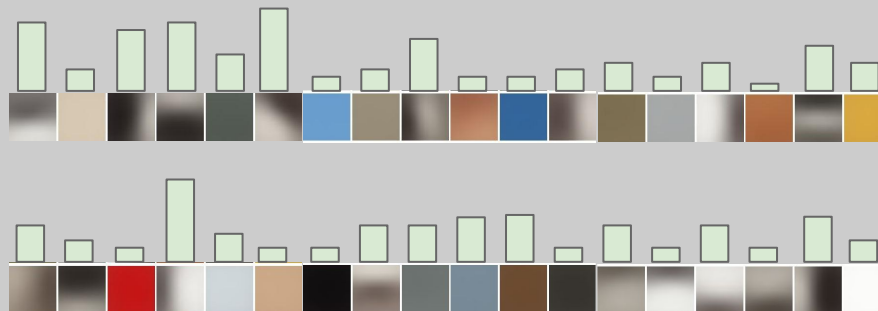


Image Features

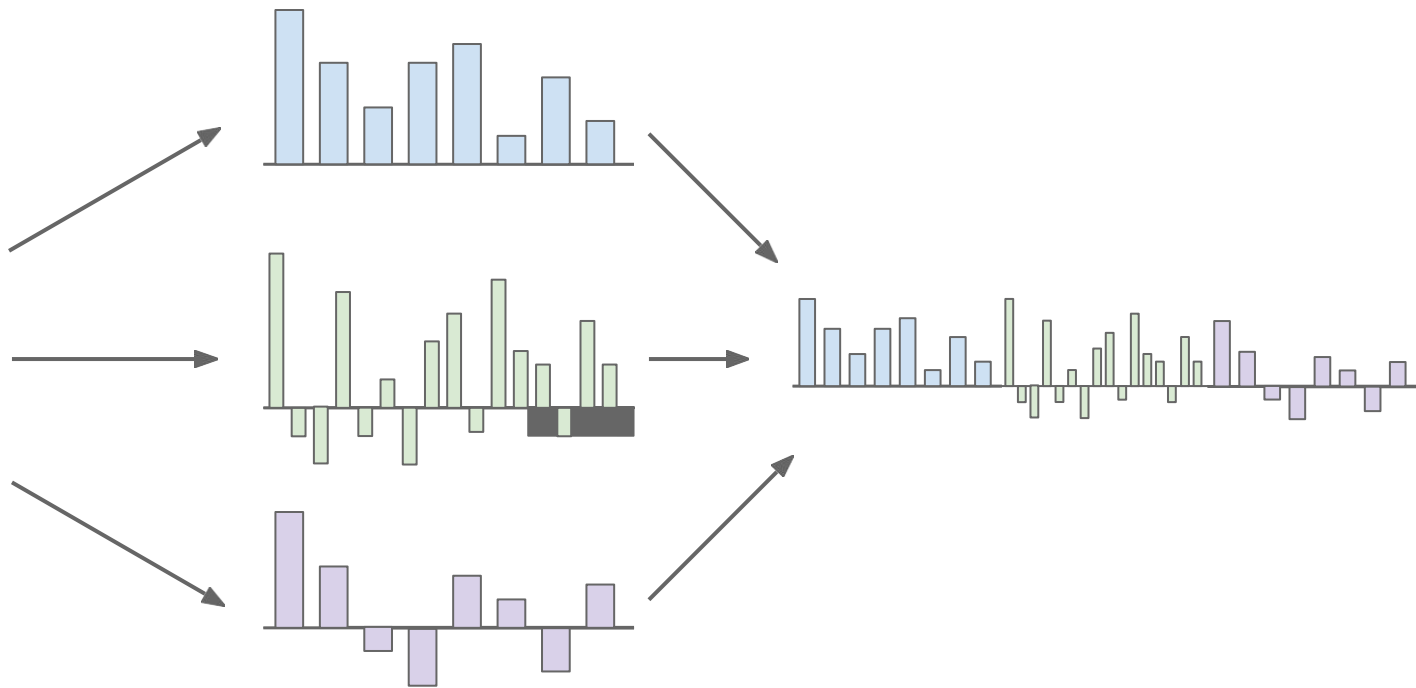
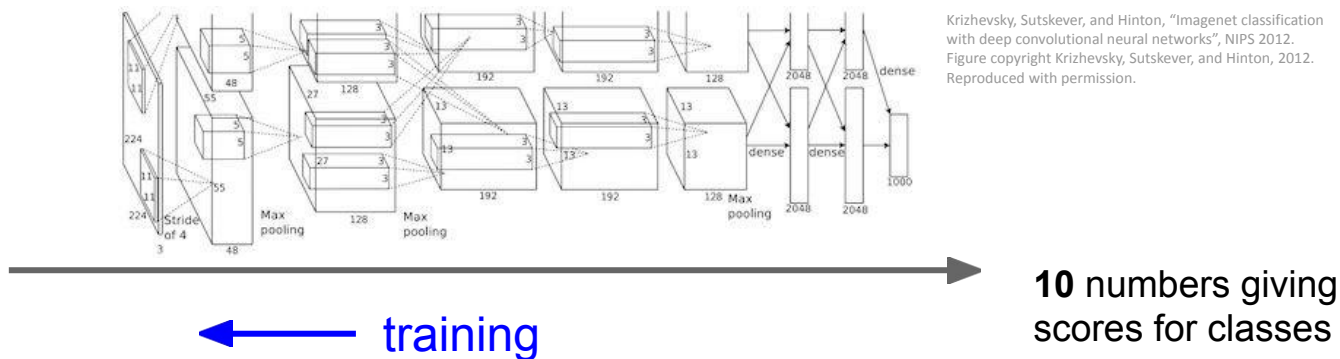
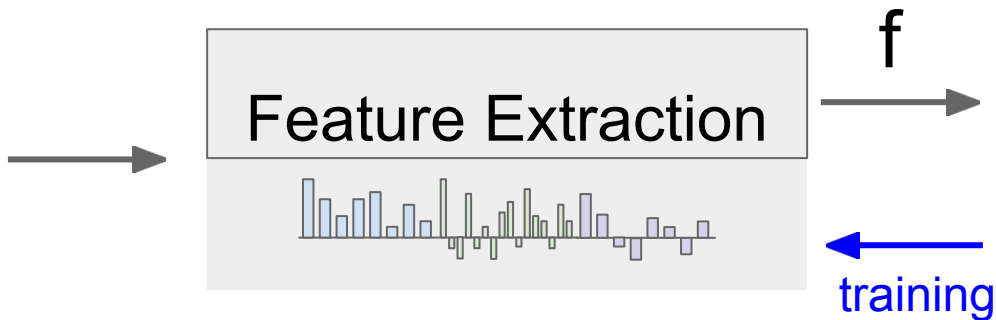


Image features vs. ConvNets



A bit of history...

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

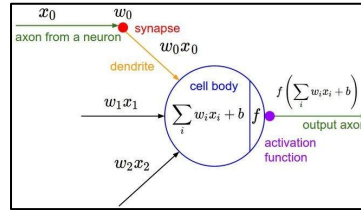
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

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

update rule:

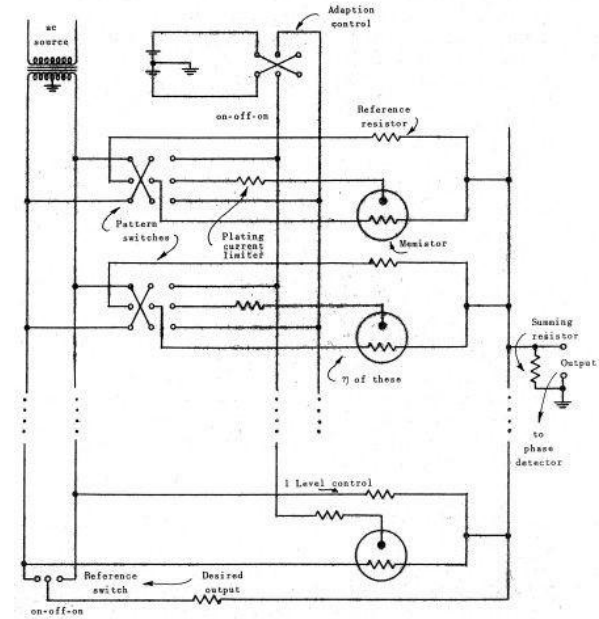
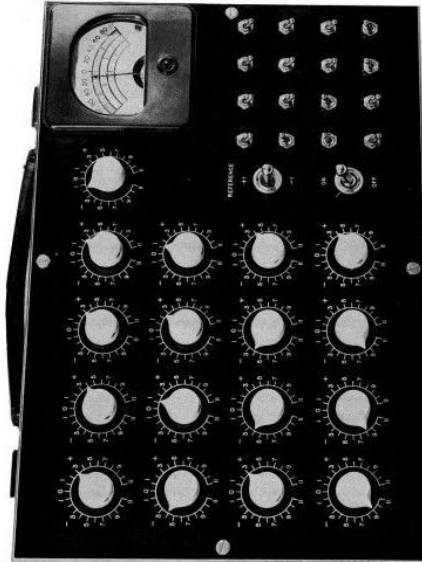
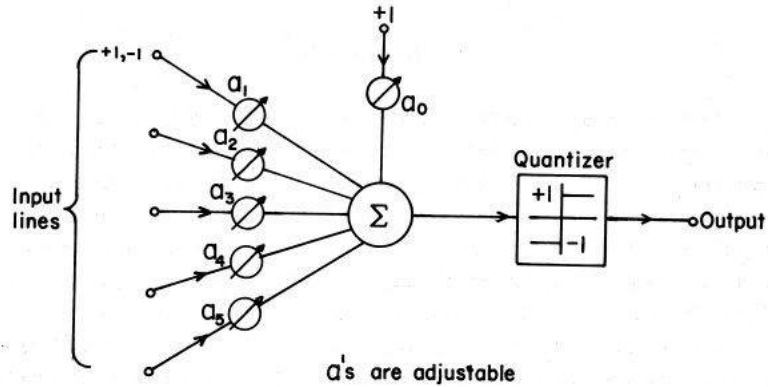
$$w_i(t + 1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$



[This image](#) by Rocky Acosta is licensed under [CC-BY 3.0](#)

Frank Rosenblatt, ~1957: Perceptron

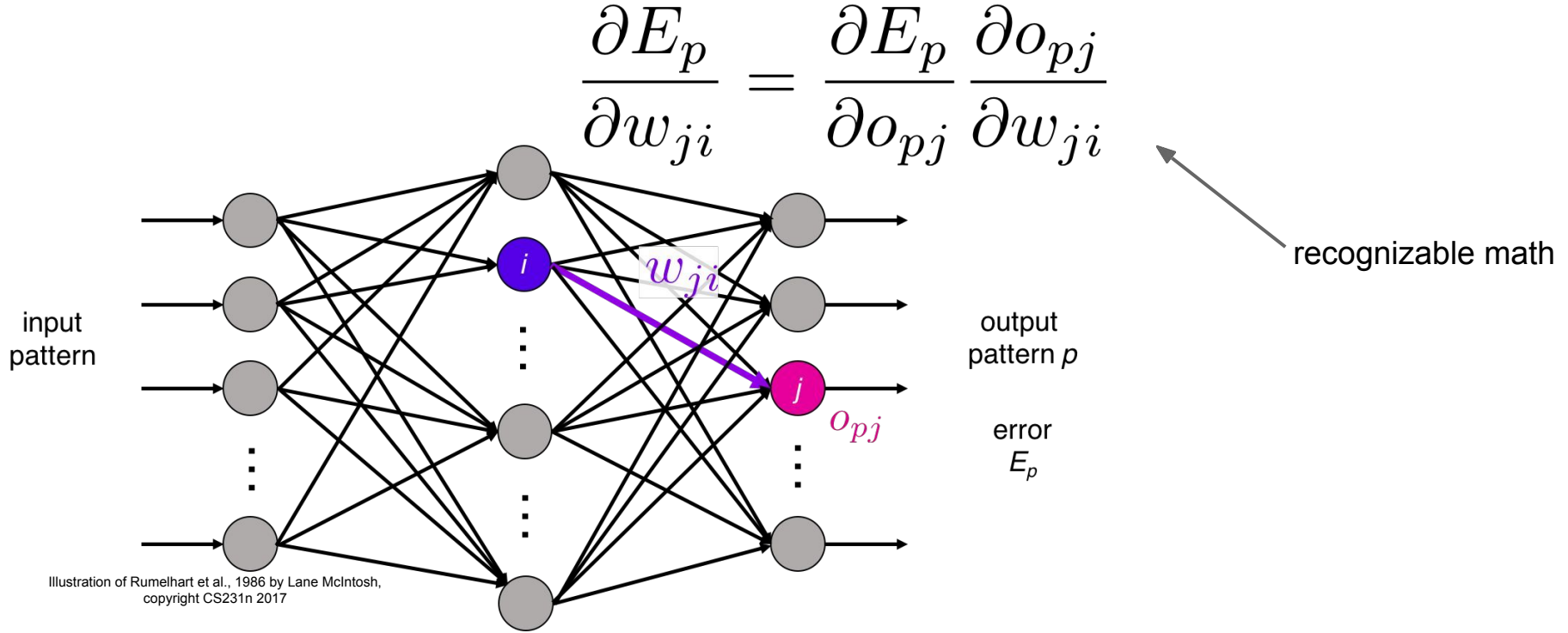
A bit of history...



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...

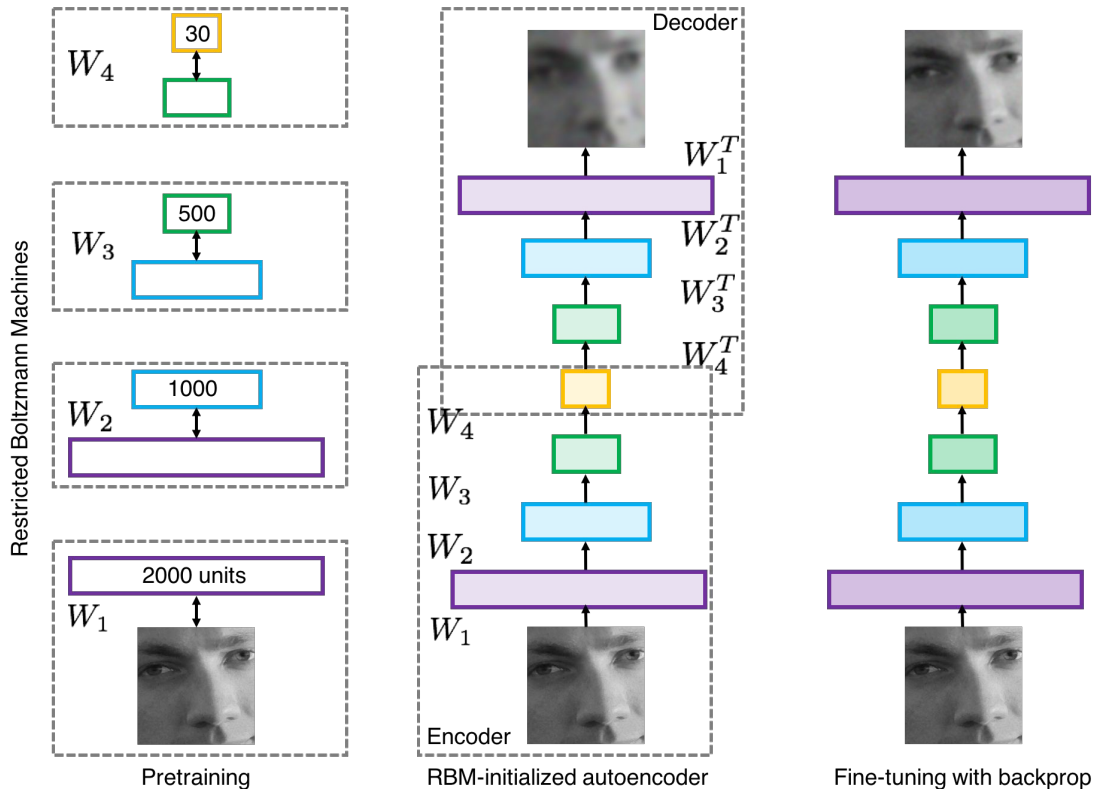


Rumelhart et al., 1986: First time back-propagation became popular

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning



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

Imagenet classification with deep convolutional neural networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

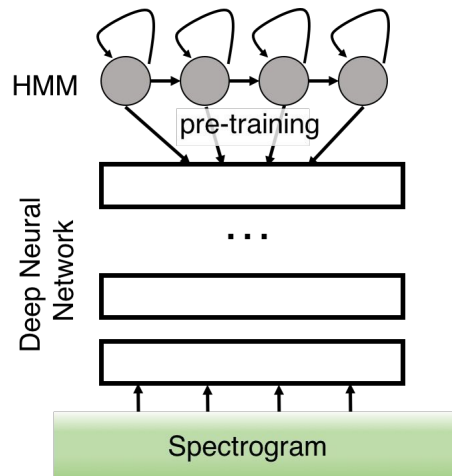
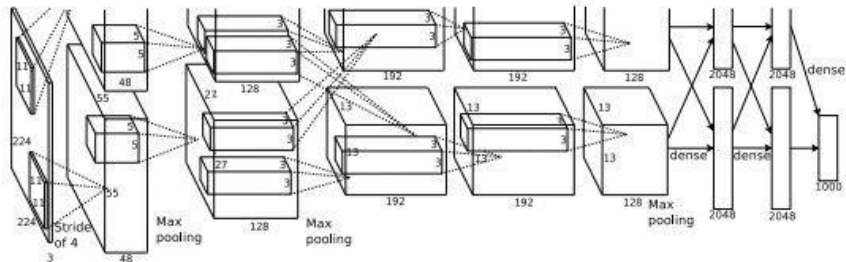


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017



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

A bit of history:

Hubel & Wiesel,

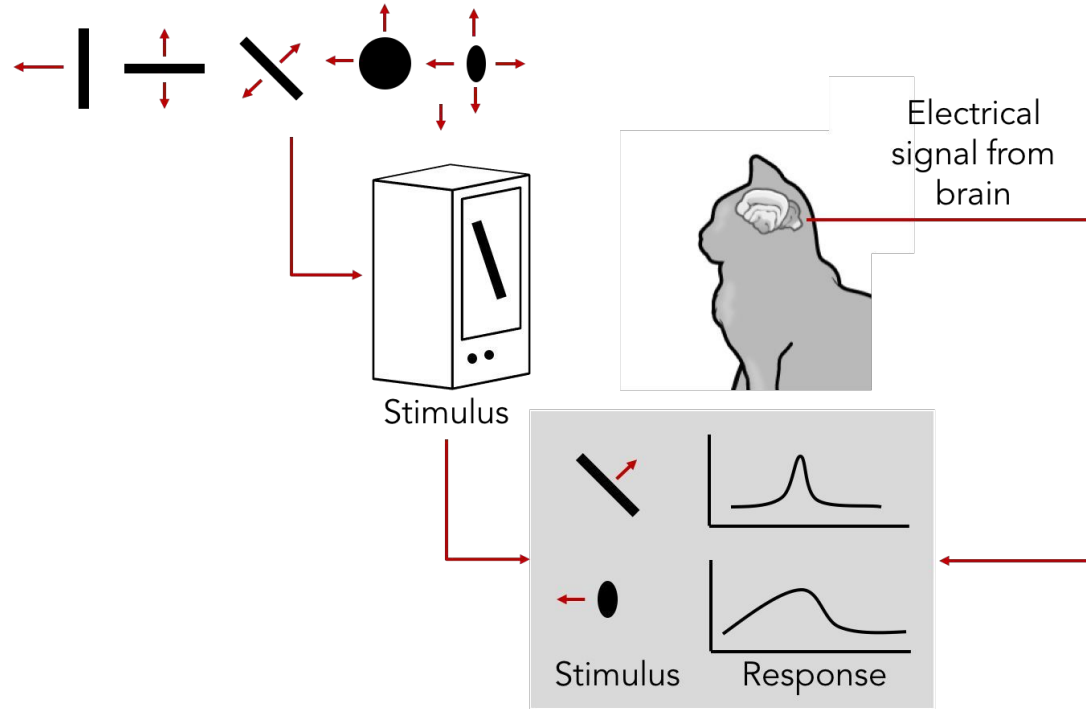
1959

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

1962

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

1968...

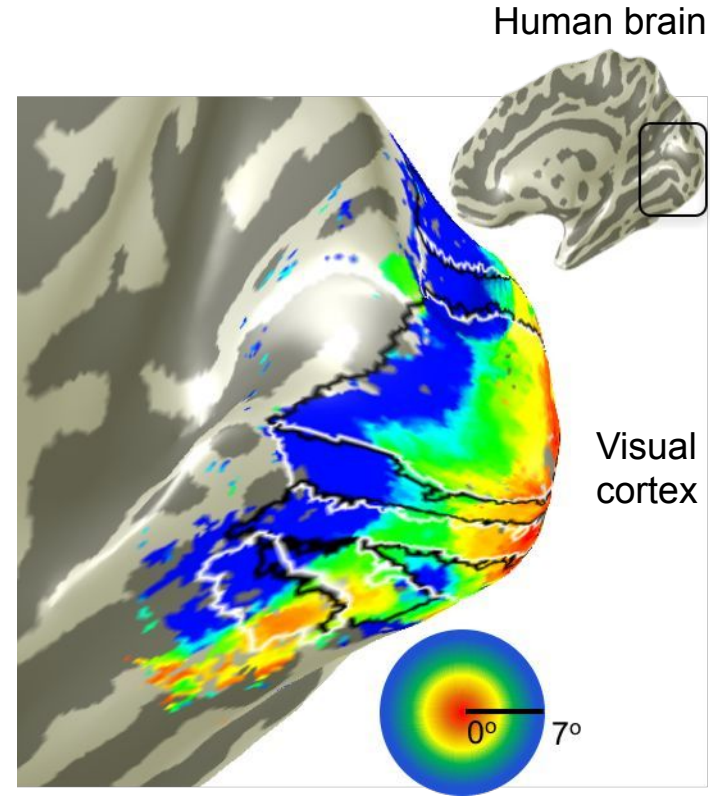
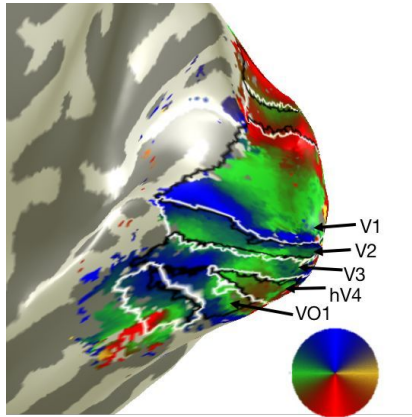


[Cat image](#) by CNX OpenStax is licensed under CC BY 4.0; changes made

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

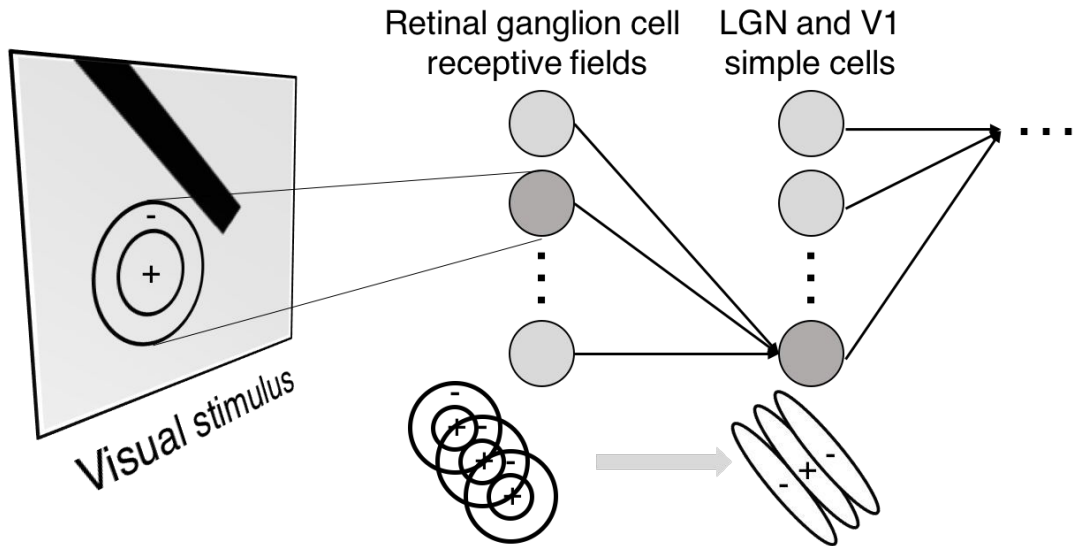
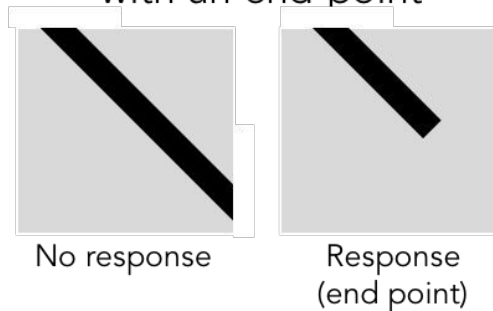


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

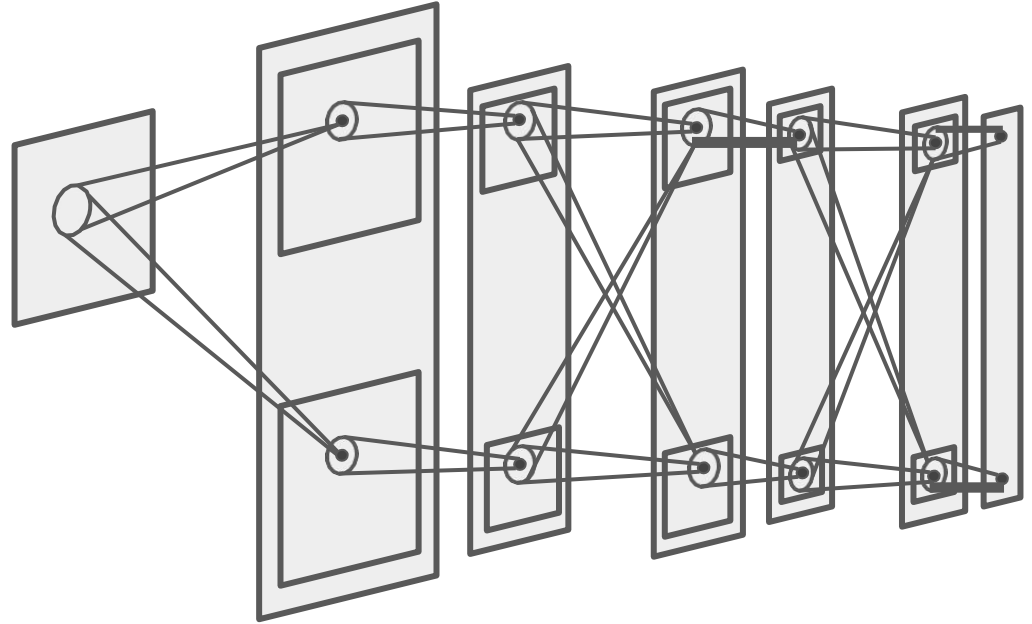


A bit of history:

Neocognitron

[Fukushima 1980]

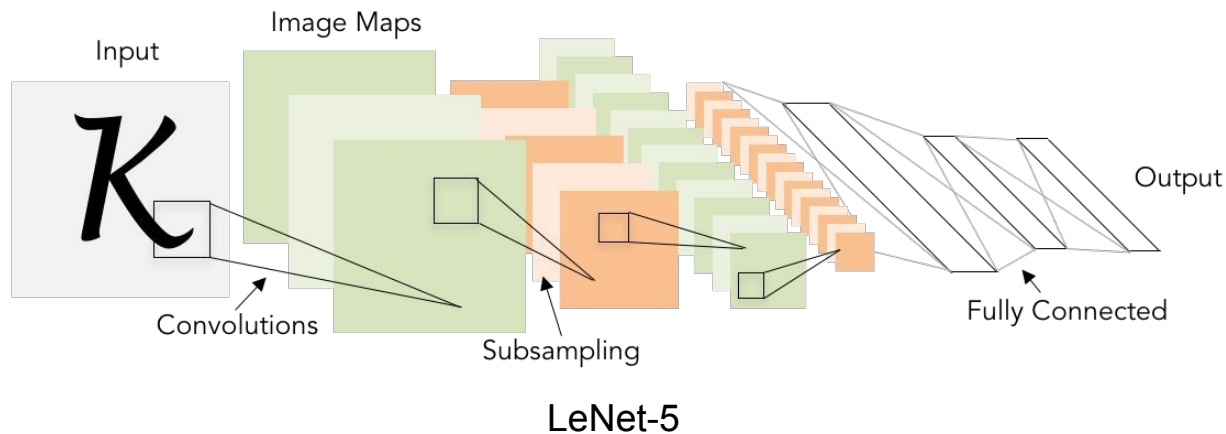
“sandwich” architecture (SCSCSC...)
simple cells: modifiable parameters
complex cells: perform pooling



A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



CIFAR: Canadian Institute for Advanced Research.

CIFAR encourages basic research without direct application



- motivated Hinton to move to Canada in 1987 and funded his work
- the funding was ended in the mid 90s just as sentiment towards neural nets was becoming negative again
- rather than relenting and switching his focus, Hinton fought to continue work on neural nets, and managed to secure more funding from CIFAR



But in 2004, Hinton asked to lead a new program on neural computation.

The mainstream machine learning community could not have been less interested in neural nets.

“It was the worst possible time,” says Bengio, a professor at the Université de Montréal and co-director of the CIFAR program since it was renewed last year.

“Everyone else was doing something different. Somehow, Geoff convinced them. We should give (CIFAR) a lot of credit for making that gamble.”



CIFAR “had a huge impact in forming a community around deep learning,” adds LeCun, the CIFAR program’s other co-director.



2019 A.M. Turing Award: Bengio, Hinton, LeCun (& [CIFAR](#))

CIFAR

CIFAR convenes extraordinary minds to address science and humanity's most important questions.



ARTIFICIAL INTELLIGENCE

Turing Award honours CIFAR's
'pioneers of AI'

CIFAR Fellows Yoshua Bengio, Geoffrey Hinton and Yann LeCun were jointly awarded the prestigious A.M. Turing Award



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.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



[The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset.](#)

[They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.](#)



“The paradigm shift of the ImageNet thinking is that while a lot of people are paying attention to models, let’s pay attention to data. Data will redefine how we think about models.” – Fei-Fei Li

- In July 2008, ImageNet had zero images.
- By December, it had categorized three million images across 6,000+ synsets.
- In April 2010, there were more than 11 million images in 15,000+ synsets.
- *They were made possible through crowdsourcing on Amazon’s Mechanical Turk platform.*
- In 2010, the first ever **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** was organized.
- Software programs competed to correctly classify and detect objects and scenes.
- ImageNet has given researchers a common set of images to benchmark their models and algorithms.
- In turn, this has driven research in machine learning and deep neural networks, making it easier to classify images and complete other tasks associated with computer vision.
- The data is available for free to researchers for non-commercial use.

A bit of history:

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

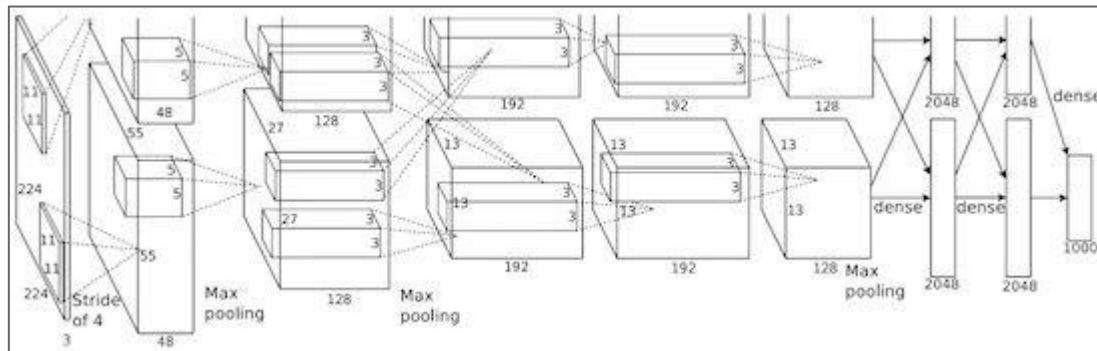
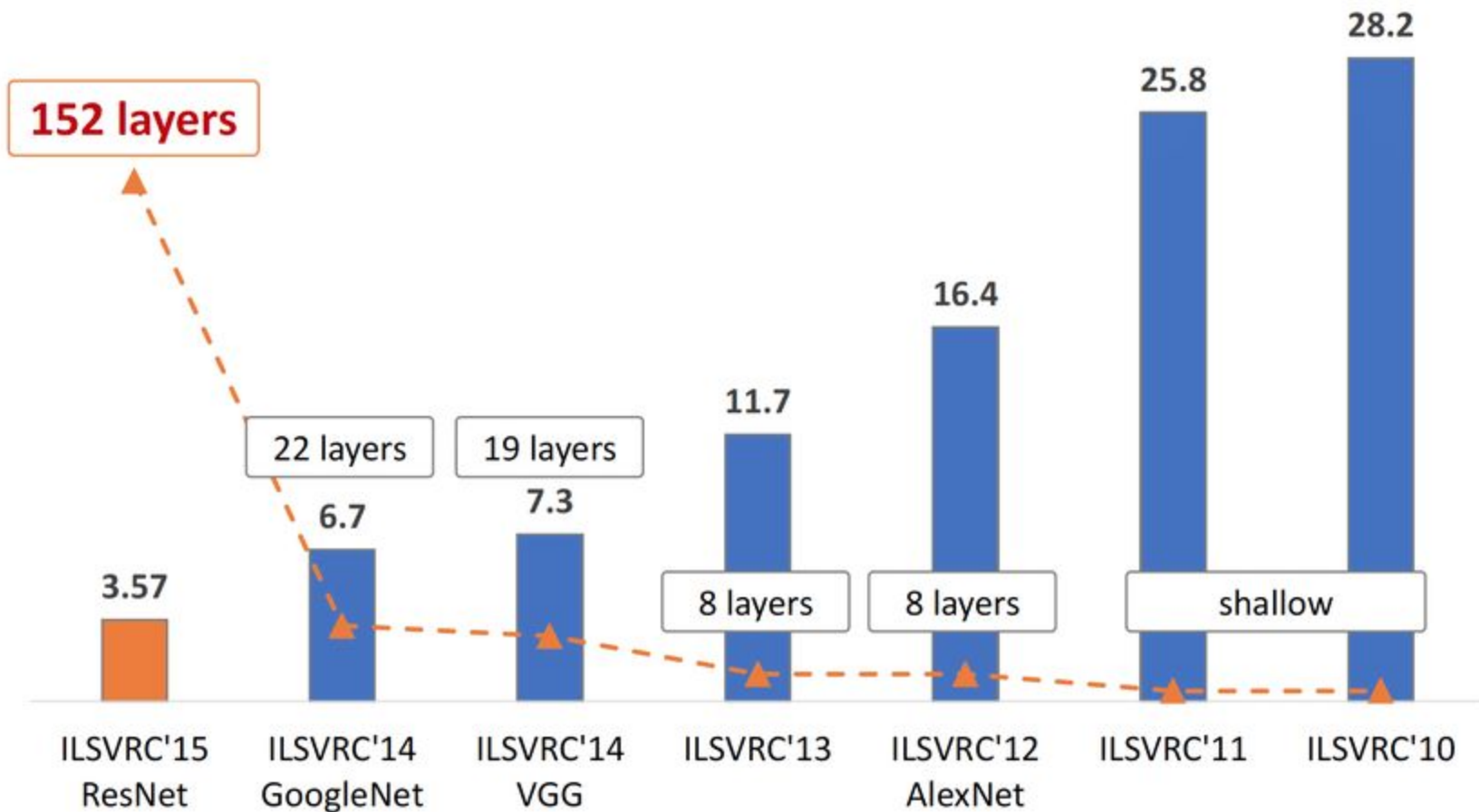


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

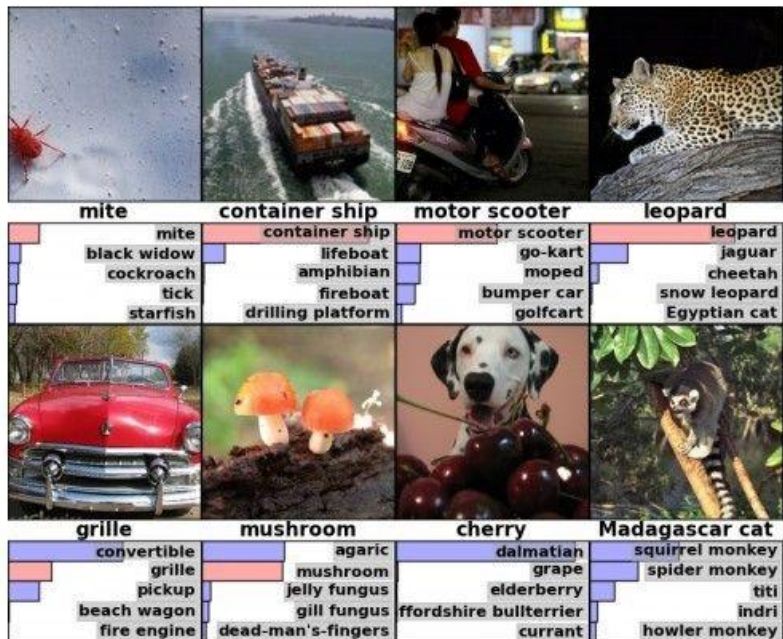




[The evolution of the winning entries on the ImageNet Large Scale Visual..](#)

Fast-forward to today: ConvNets are everywhere

Classification



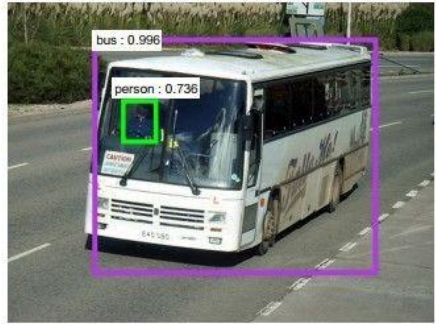
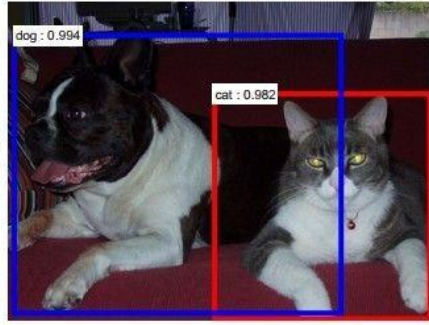
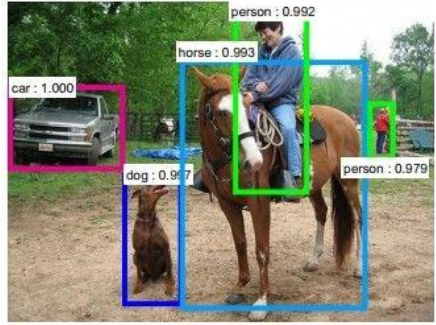
Retrieval



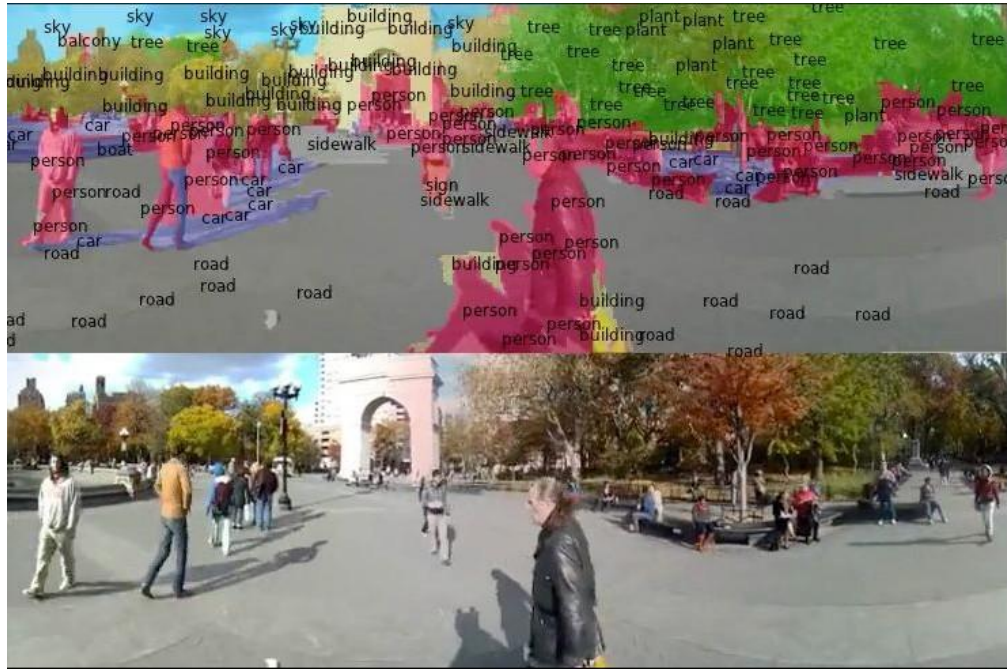
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Detection



Segmentation



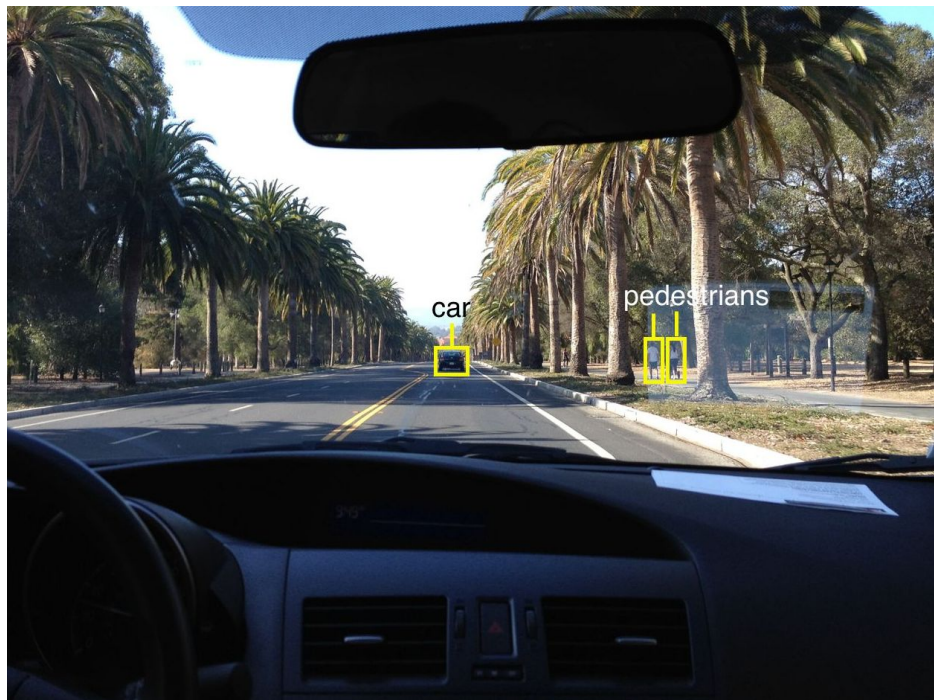
Figures copyright Clement Farabet, 2012. Reproduced with permission.

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

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

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



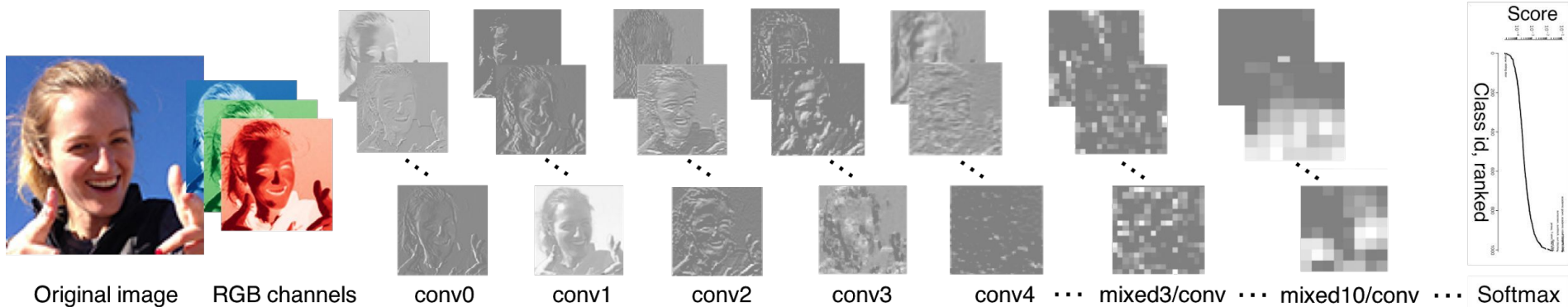
[This image](#) by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

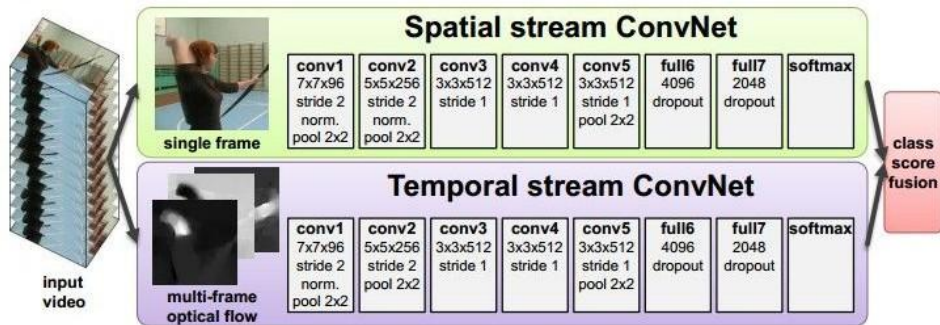
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of [inception-v3 architecture](#) [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

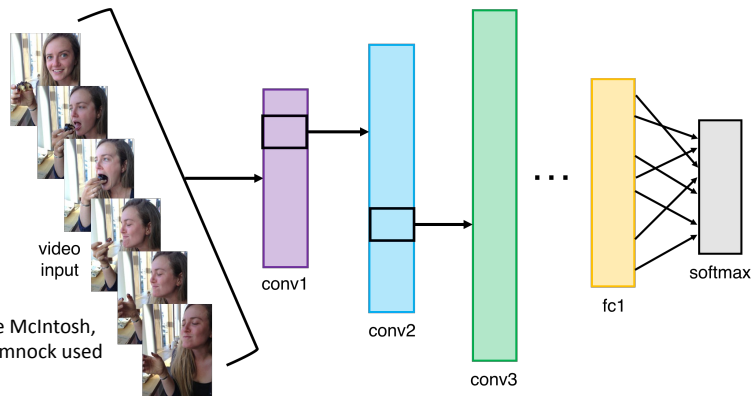


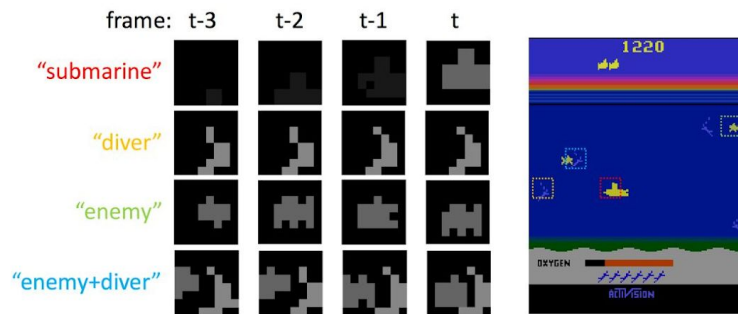
Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere



Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

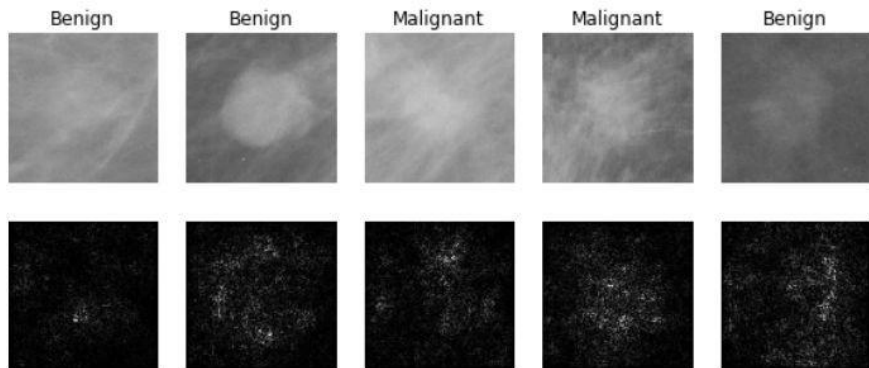
[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.



[Dieleman et al. 2014]

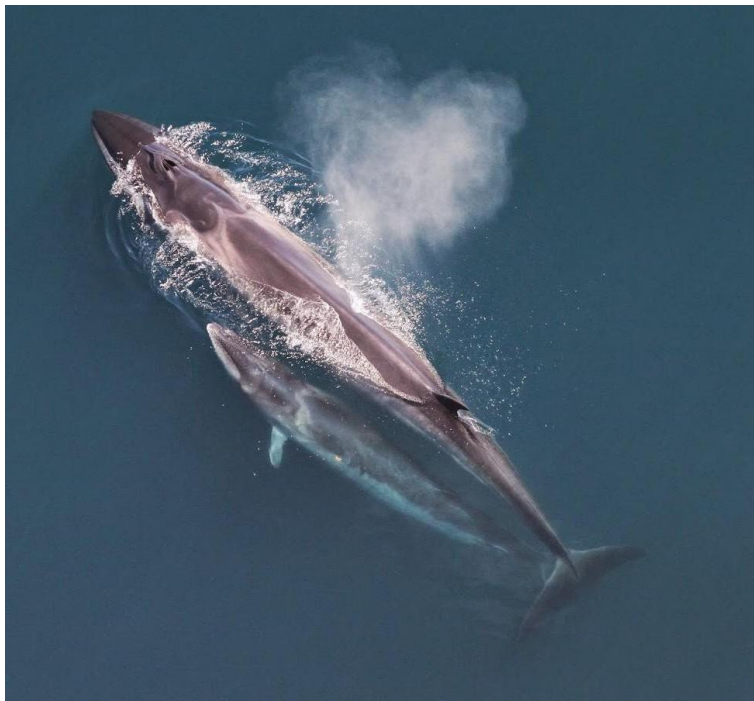
From left to right: [public domain by NASA](#), usage [permitted](#) by ESA/Hubble, [public domain by NASA](#), and [public domain](#).



[Sermanet et al. 2011] [Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.



Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.



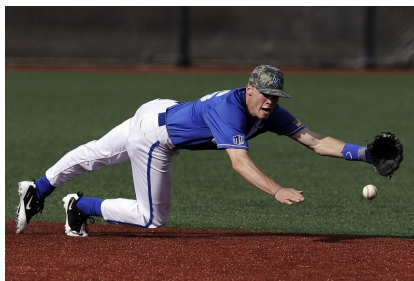
Mnih and Hinton, 2010

No errors



A white teddy bear sitting in the grass

Minor errors



A man in a baseball uniform throwing a ball

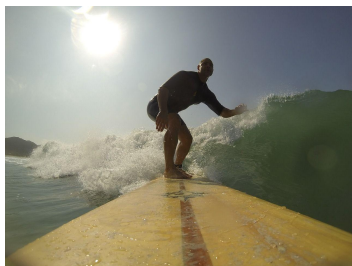
Somewhat related



A woman is holding a cat in her hand

Image Captioning

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



A man riding a wave on top of a surfboard



A cat sitting on a suitcase on the floor

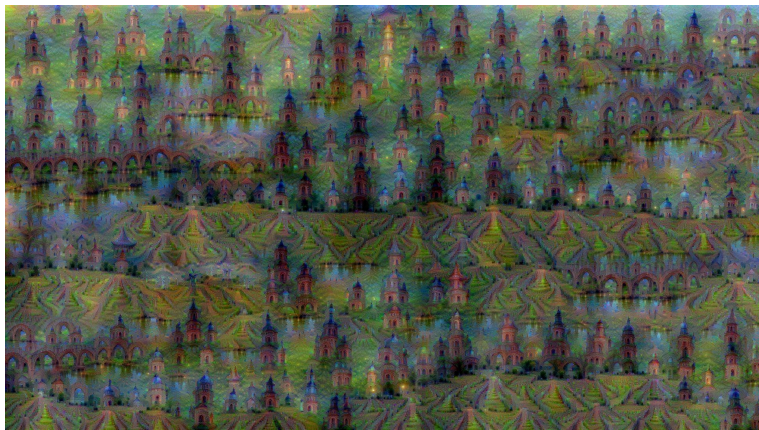
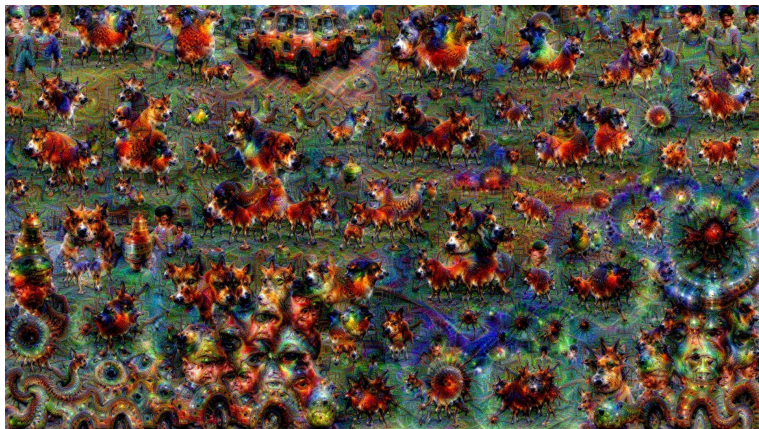


A woman standing on a beach holding a surfboard

All images are CC0 Public domain:

<https://pixabay.com/en/luggage-antique-cat-1643010/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>
<https://pixabay.com/en/handstand-lake-meditation-496008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [NeuralTalk2](#)



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.

[Original image](#) is CC0 public domain
[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain
[Bokeh image](#) is in the public domain
 Stylized images copyright Justin Johnson, 2017;
 reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
 Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Dall-E 2



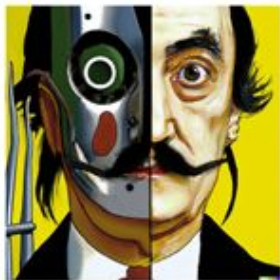
“Teddy bears working on new AI research on the moon in the 1980s.”



“Rabbits attending a college seminar on human anatomy.”



“A wise cat meditating in the Himalayas searching for enlightenment.”



a vibrant portrait painting of Salvador Dalí with a robotic half face



a shiba inu wearing a beret and black turtleneck



a close up of a handpalm with leaves growing from it



an espresso machine that makes coffee from human souls, artstation



panda mad scientist mixing sparkling chemicals, artstation



a corgi's head depicted as an explosion of a nebula



a dolphin in an astronaut suit on saturn, artstation

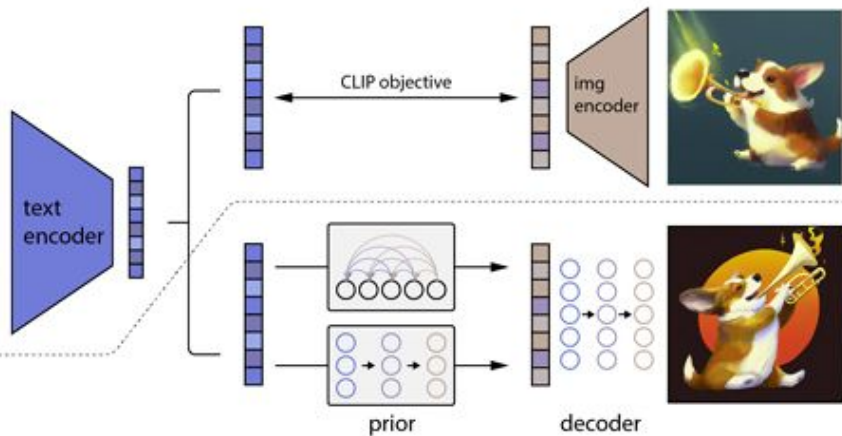


a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese



a teddybear on a skateboard in times square

"a corgi playing a flame throwing trumpet"



Ramesh et al., Hierarchical Text-Conditional Image Generation with CLIP Latents, 2022.

GPT-4

User What is unusual about this image?



Source: [Bamorama](#)

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

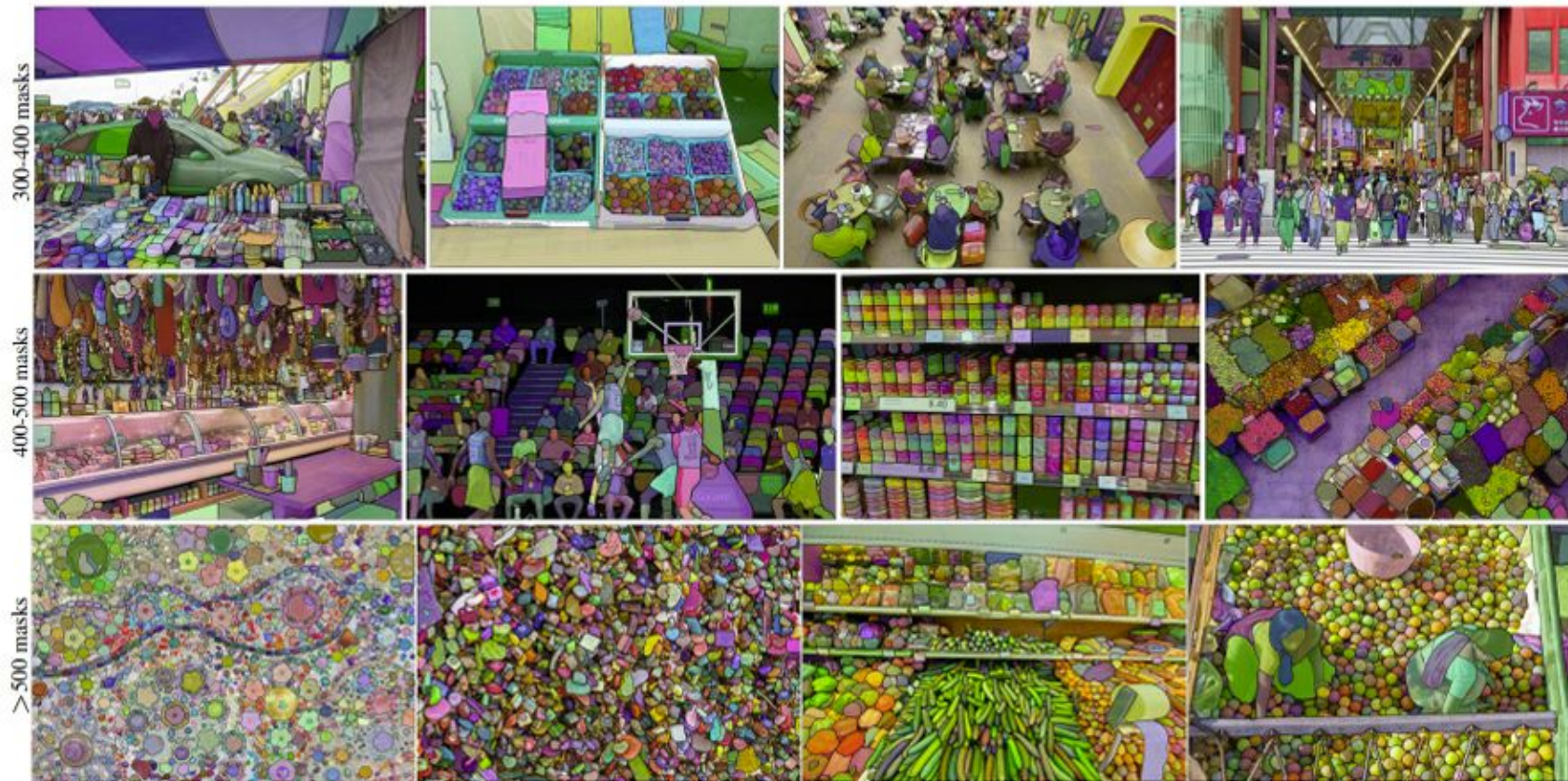
User Can you explain this meme?

Sometimes I just look at pictures of the earth from space and I marvel at how beautiful it all is.



GPT-4 This meme is a joke that combines two unrelated things: pictures of the earth from space and chicken nuggets. The text of the meme suggests that the image below is a beautiful picture of the earth from space. However, the image is actually of chicken nuggets arranged to vaguely resemble a map of the world. The humor in this meme comes from the unexpected juxtaposition of the text and the image. The text sets up an expectation of a majestic image of the earth, but the image is actually something mundane and silly.

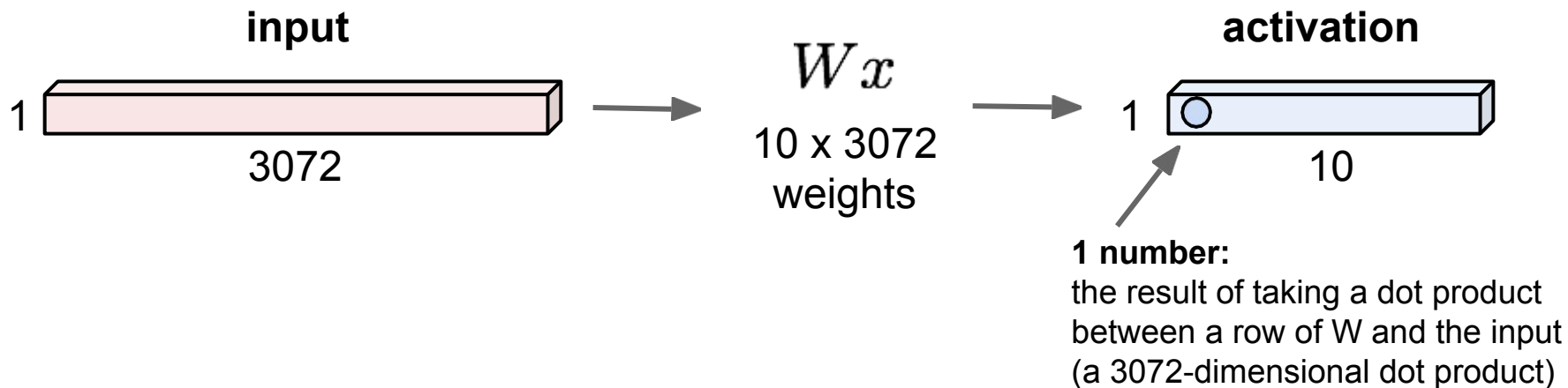
Segment Anything Model (SAM)



Convolutional Neural Networks

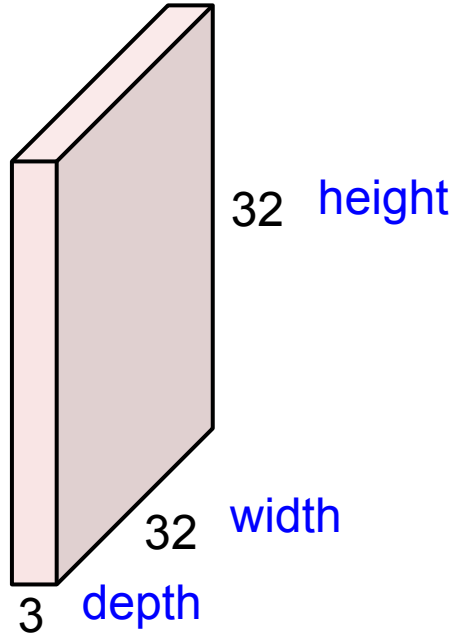
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



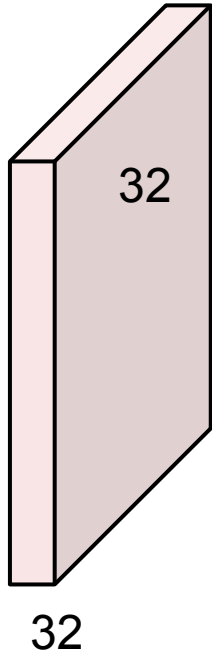
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



5x5x3 filter

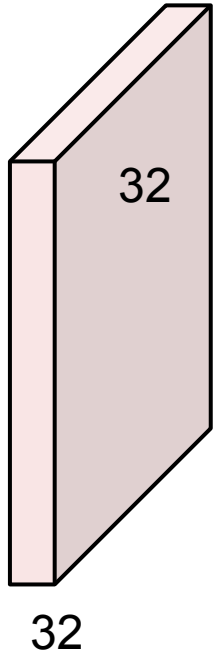


Convolve the filter with the image

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

Convolution Layer

32x32x3 image



Filters always extend the full depth of the input volume

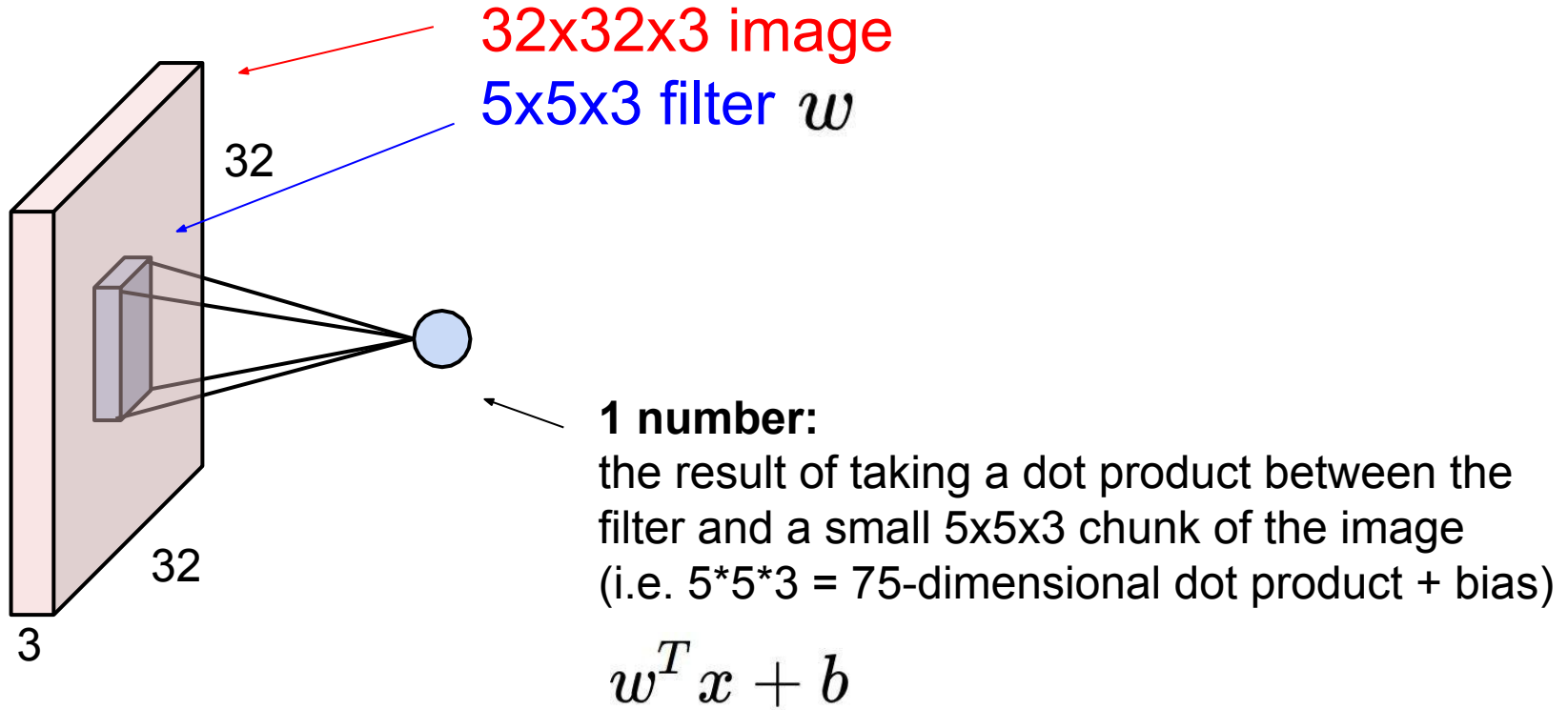
5x5x3 filter



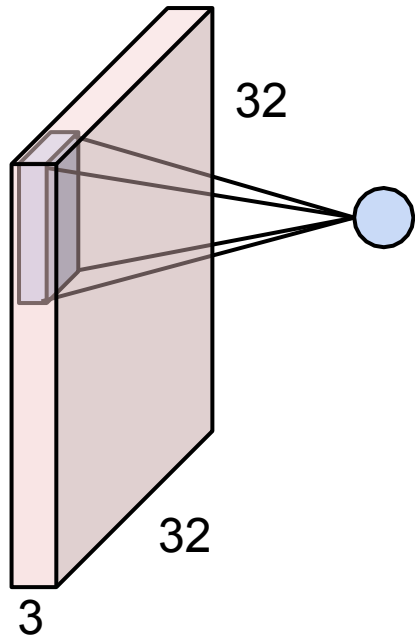
Convolve the filter with the image

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

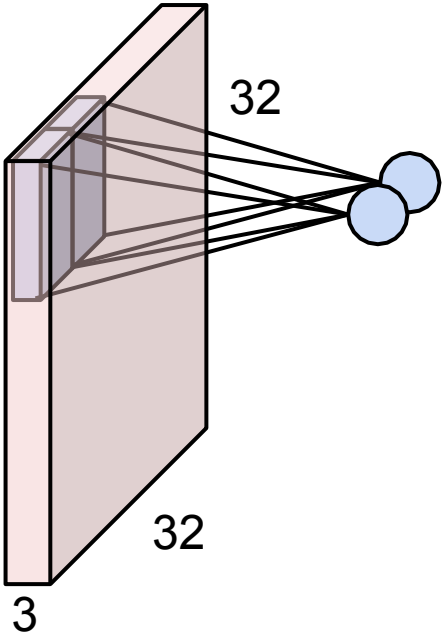
Convolution Layer



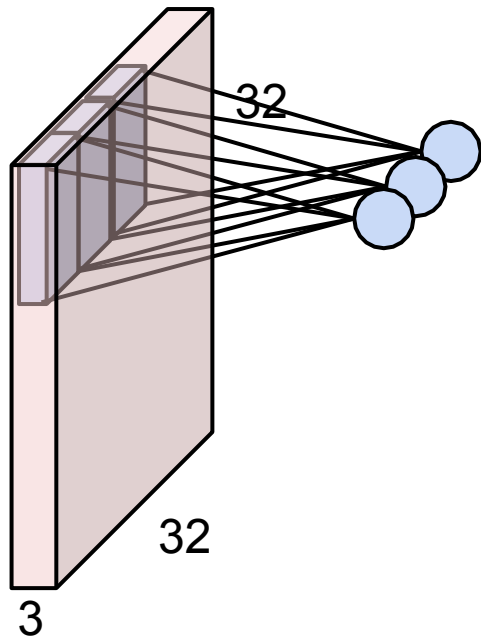
Convolution Layer



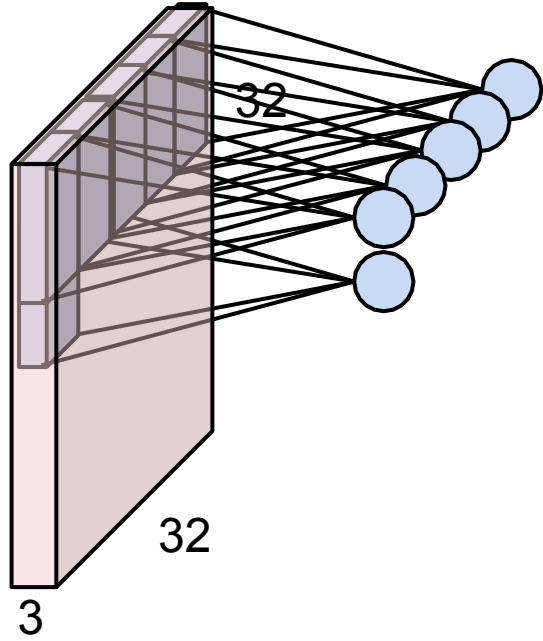
Convolution Layer



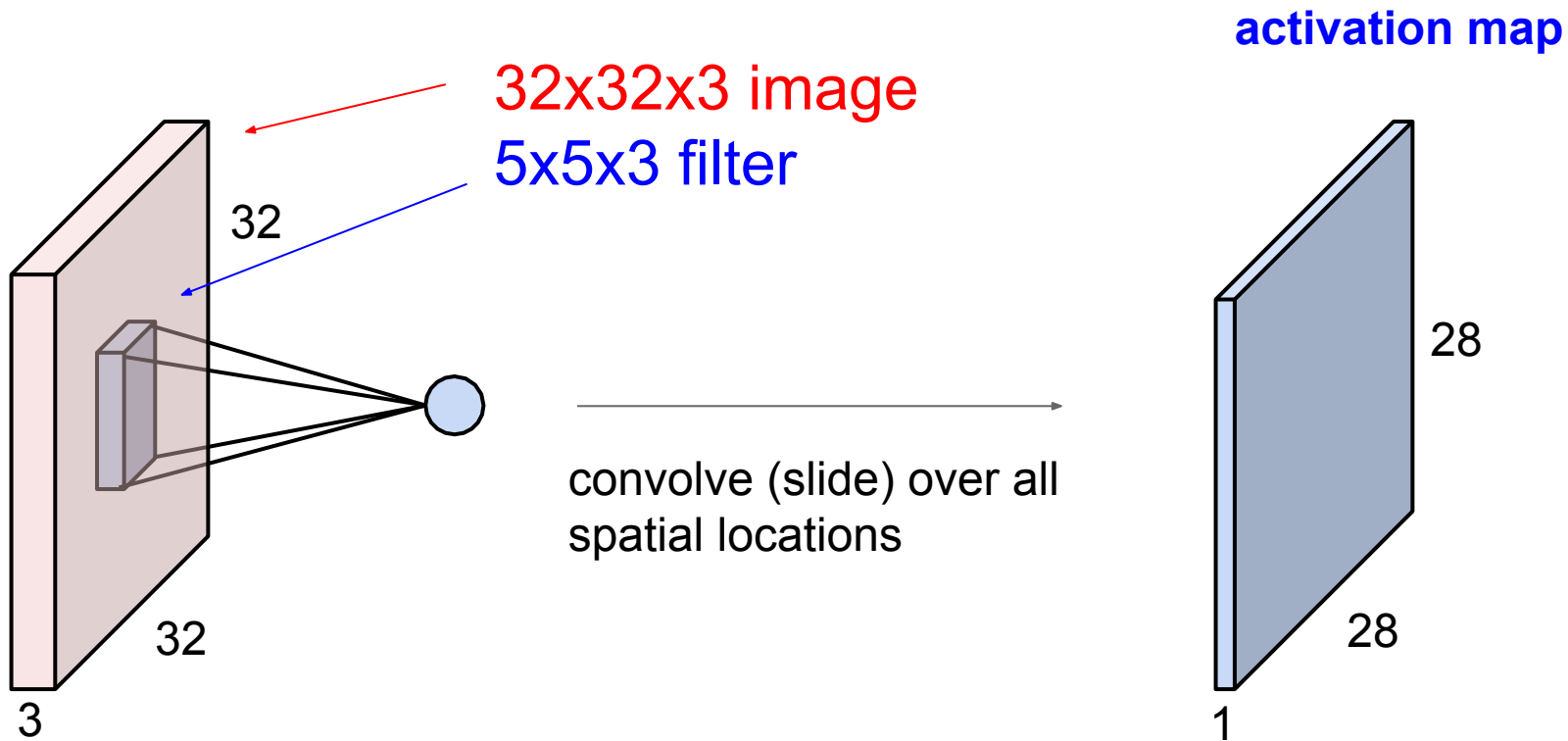
Convolution Layer



Convolution Layer

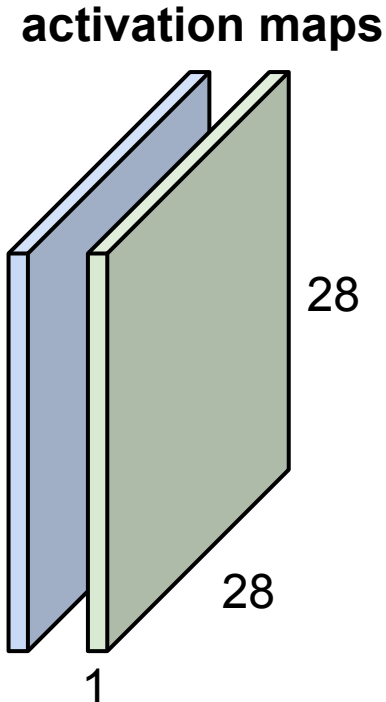
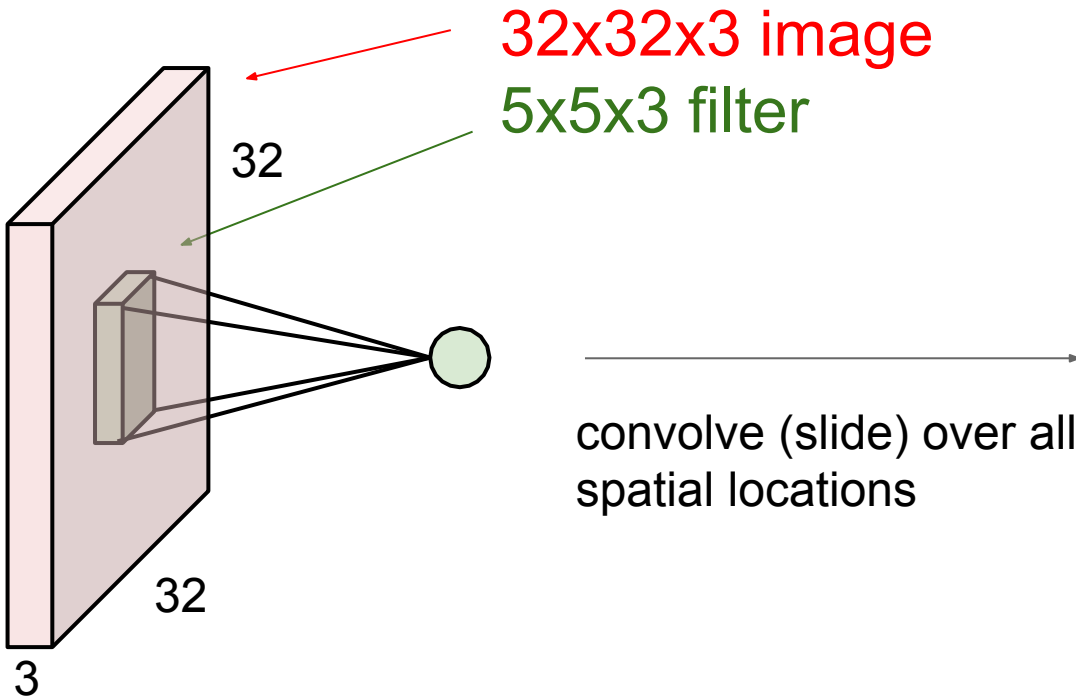


Convolution Layer



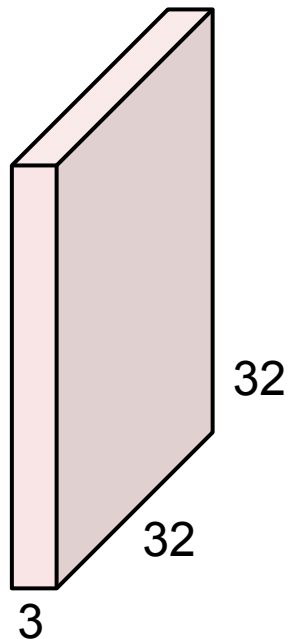
Convolution Layer

consider a second, green filter

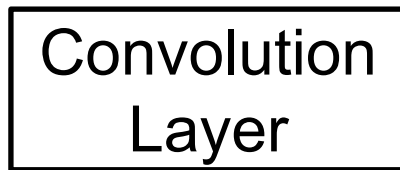


Convolution Layer

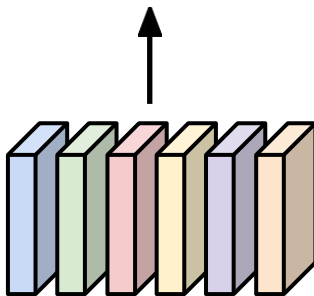
3x32x32 image



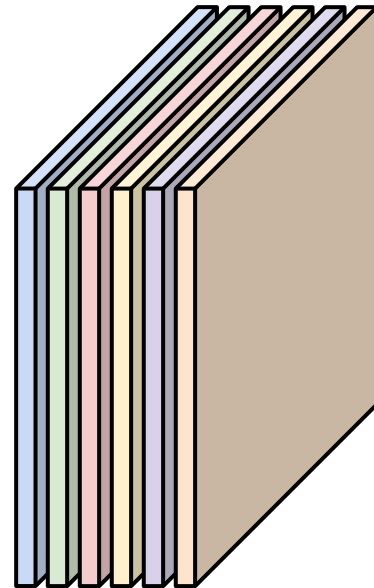
Consider 6 filters,
each 3x5x5



6x3x5x5
filters



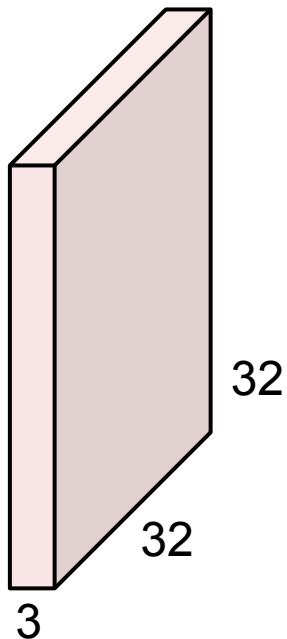
6 activation maps,
each 1x28x28



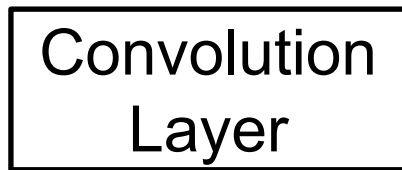
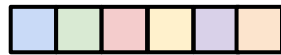
Stack activations to get a
6x28x28 output image!

Convolution Layer

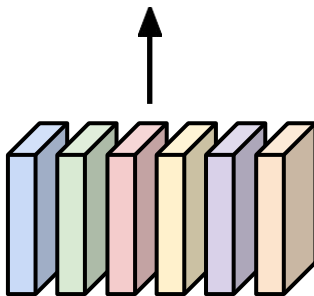
3x32x32 image



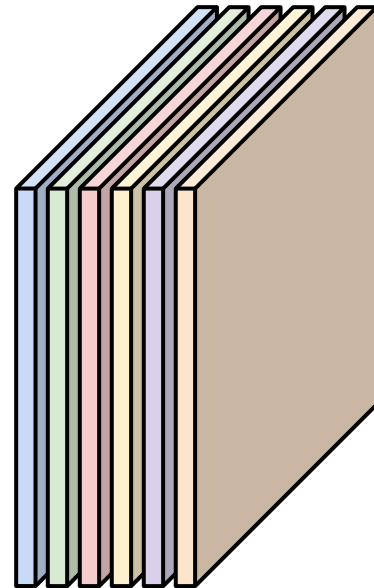
Also 6-dim bias vector:



6x3x5x5 filters



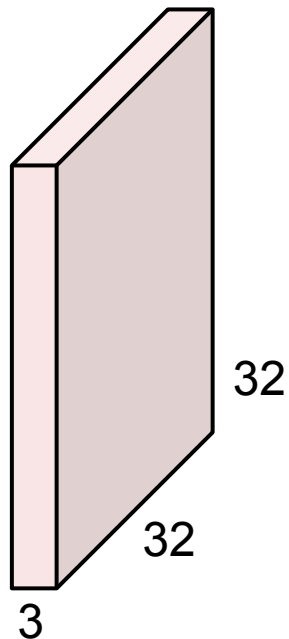
6 activation maps,
each 1x28x28



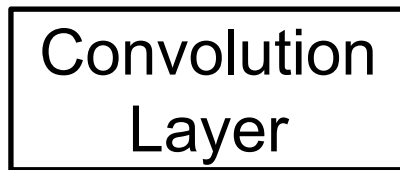
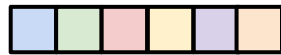
Stack activations to get a
6x28x28 output image!

Convolution Layer

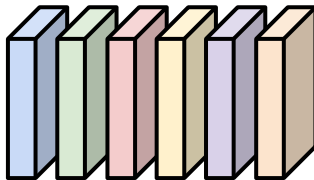
3x32x32 image



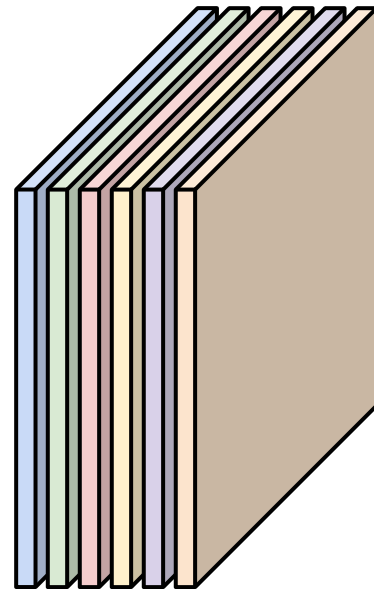
Also 6-dim bias vector:



6x3x5x5 filters



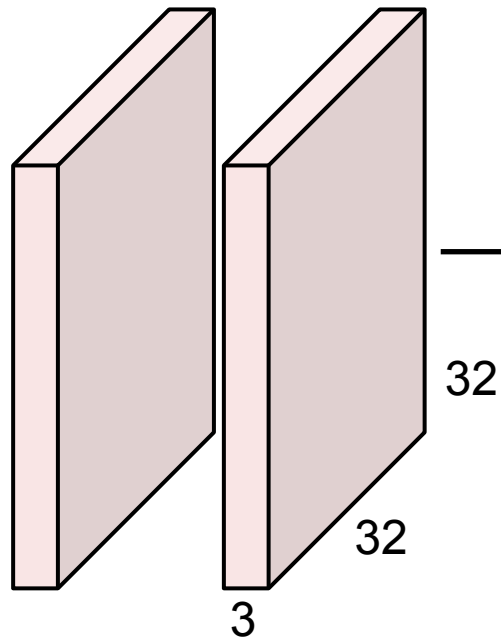
28x28 grid, at each point a 6-dim vector



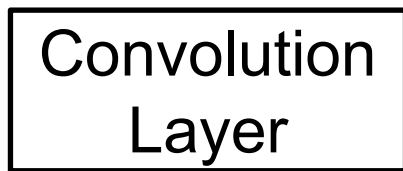
Stack activations to get a 6x28x28 output image!

Convolution Layer

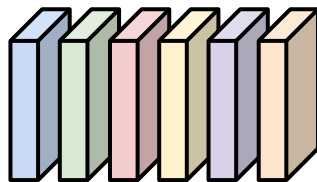
2x3x32x32
Batch of images



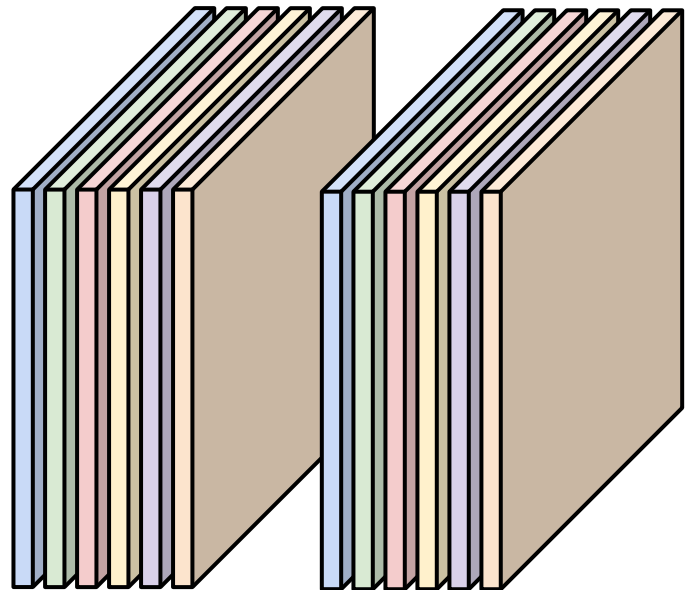
Also 6-dim bias vector:



6x3x5x5
filters



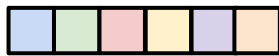
2x6x28x28
Batch of outputs



Convolution Layer

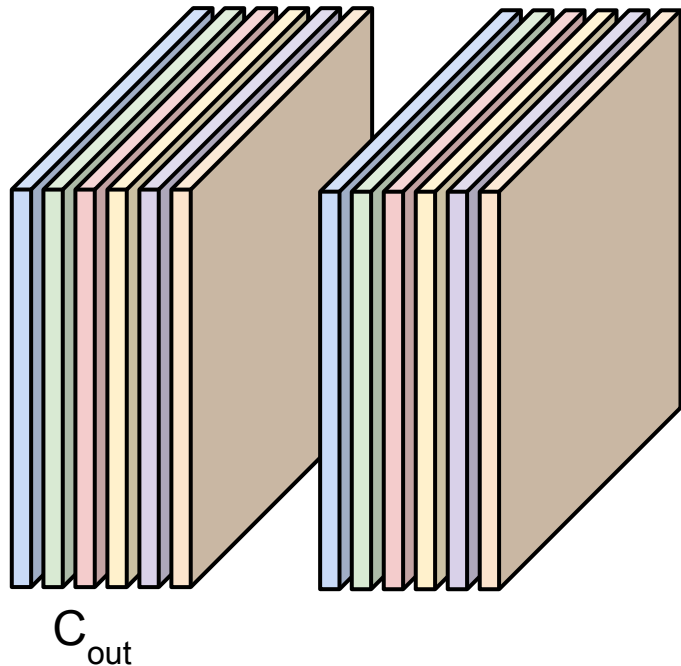
$N \times C_{in} \times H \times W$
Batch of images

Also C_{out} -dim bias vector:



Convolution
Layer

$N \times C_{out} \times H' \times W'$
Batch of outputs



H

W

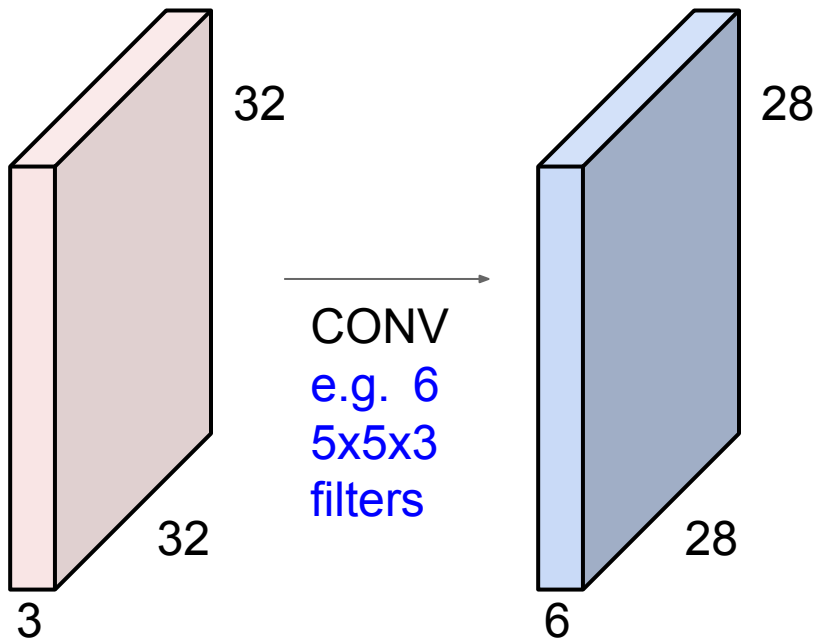
C_{in}

$C_{out} \times C_{in} \times K_w \times K_h$
filters

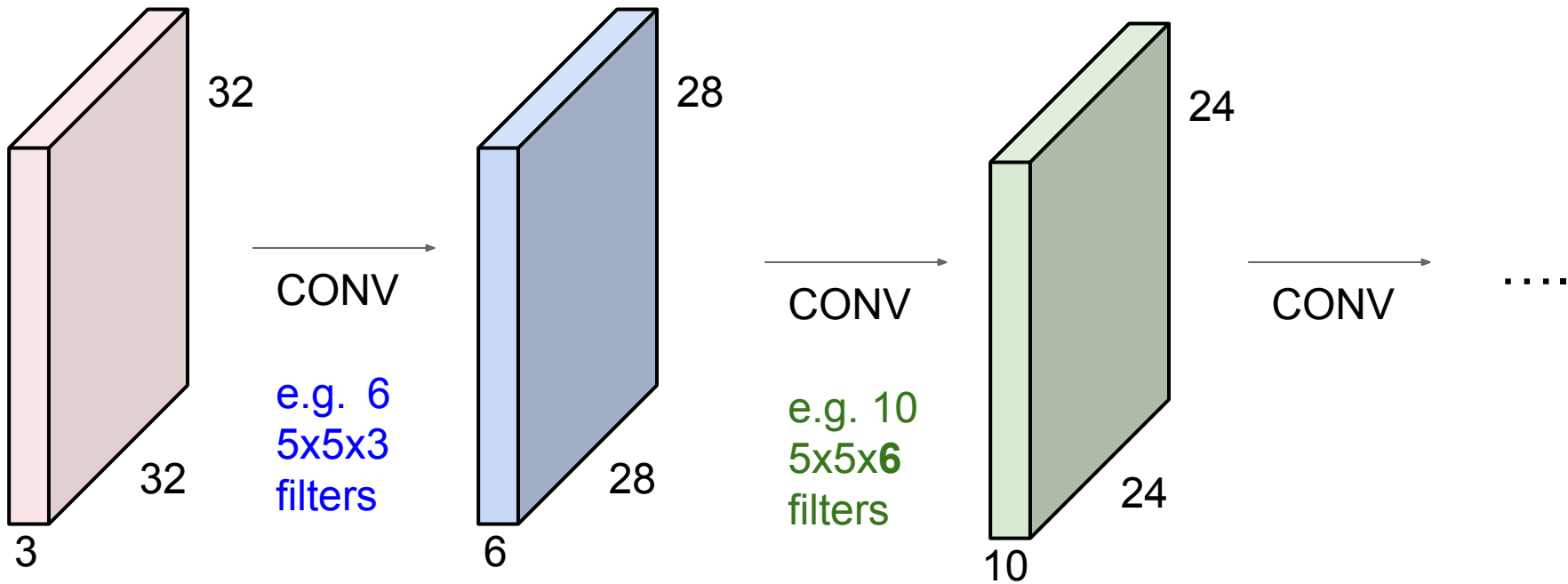


C_{out}

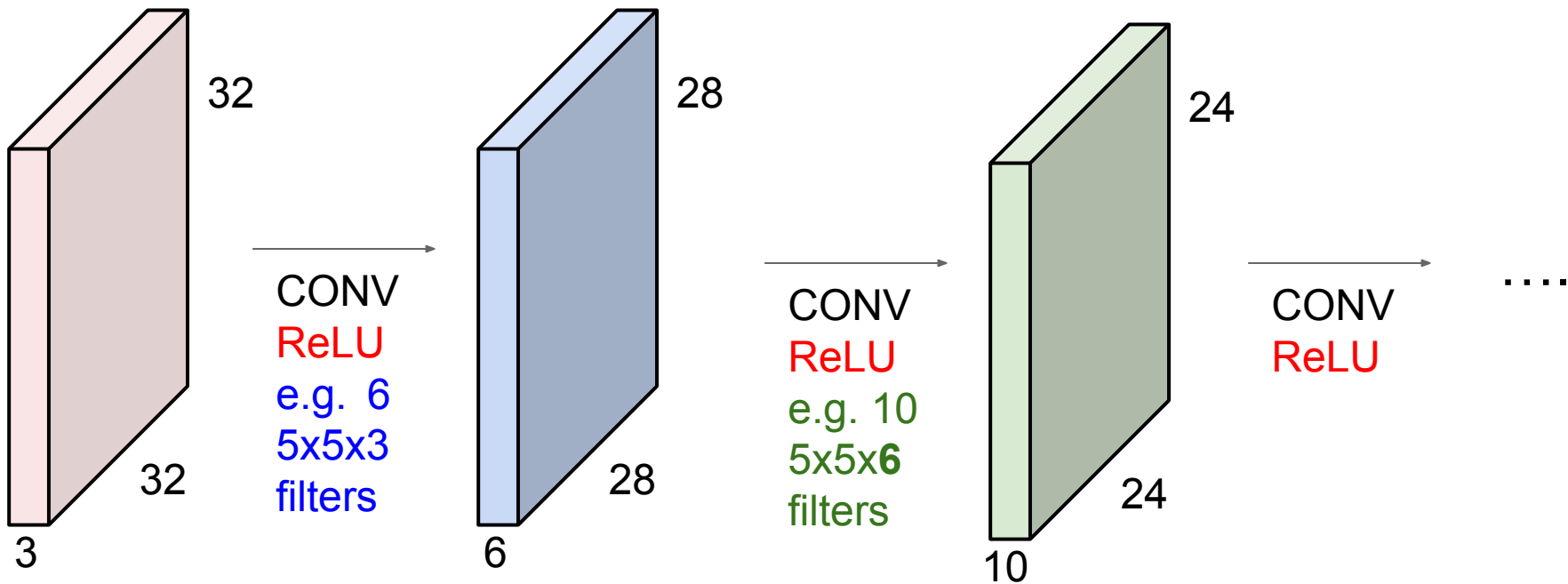
Preview: ConvNet is a sequence of Convolution Layers



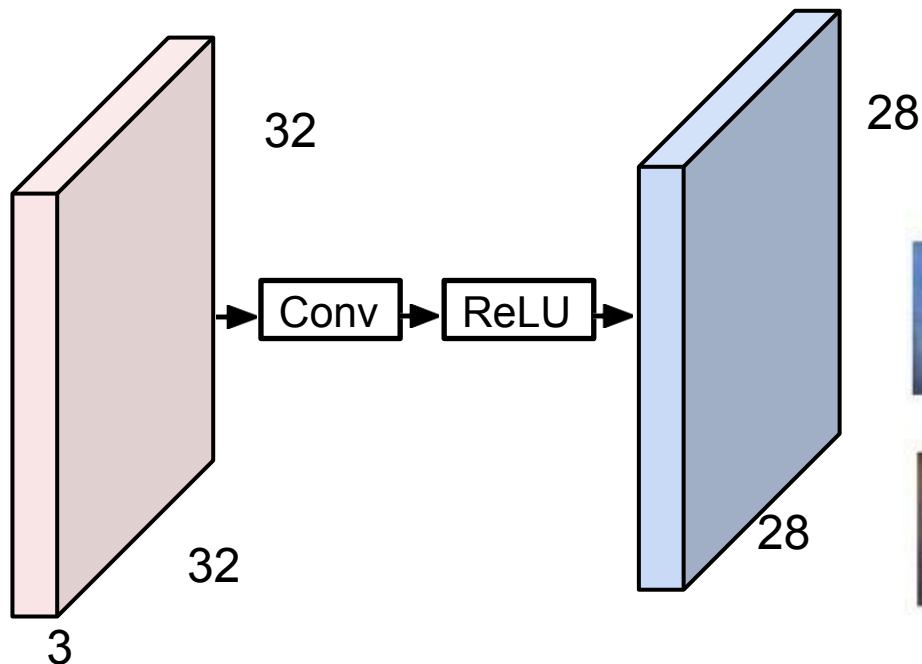
Preview: ConvNet is a sequence of Convolution Layers



Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



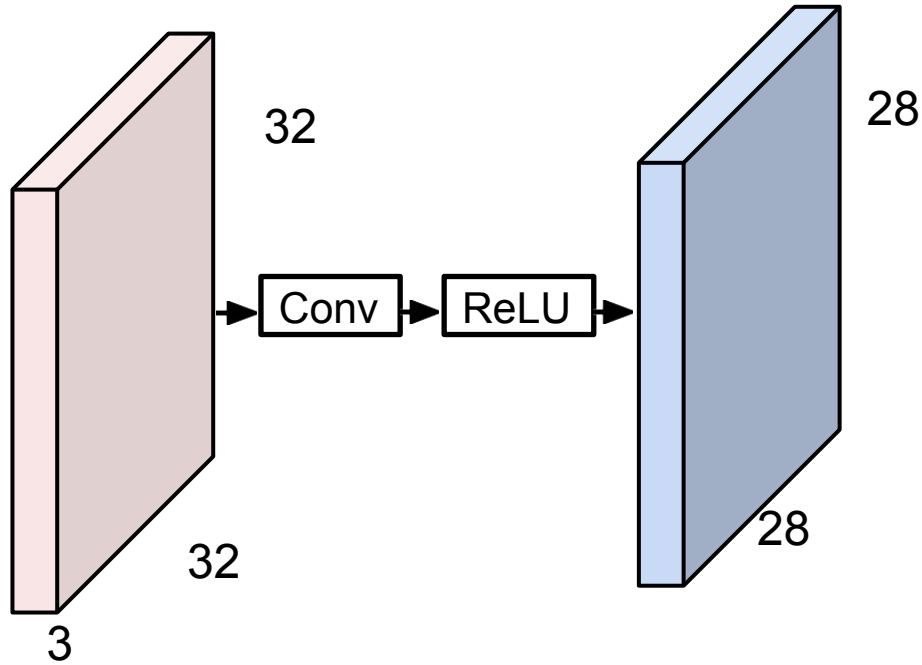
Preview: What do convolutional filters learn?



Linear classifier: One template per class



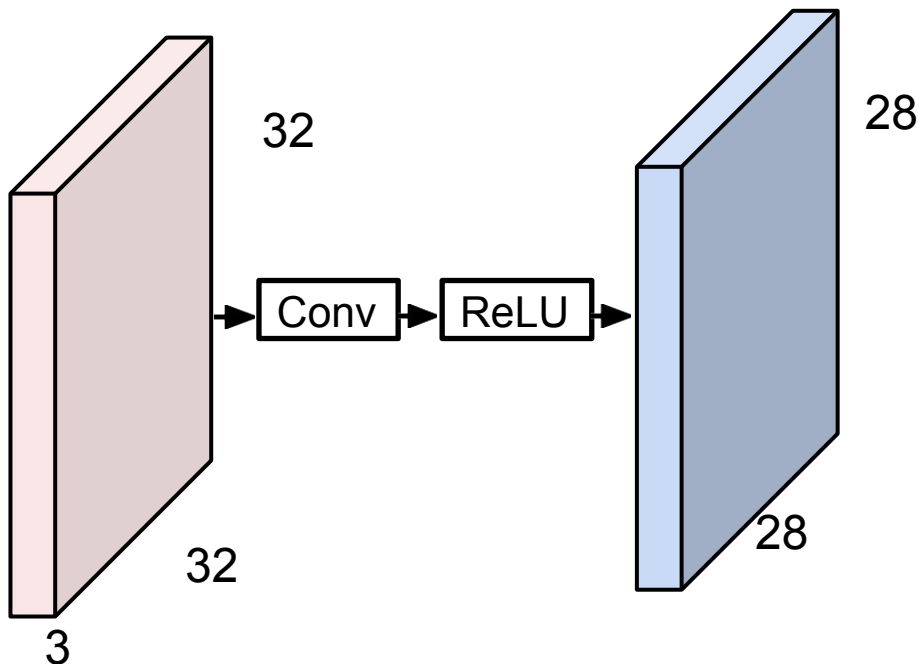
Preview: What do convolutional filters learn?



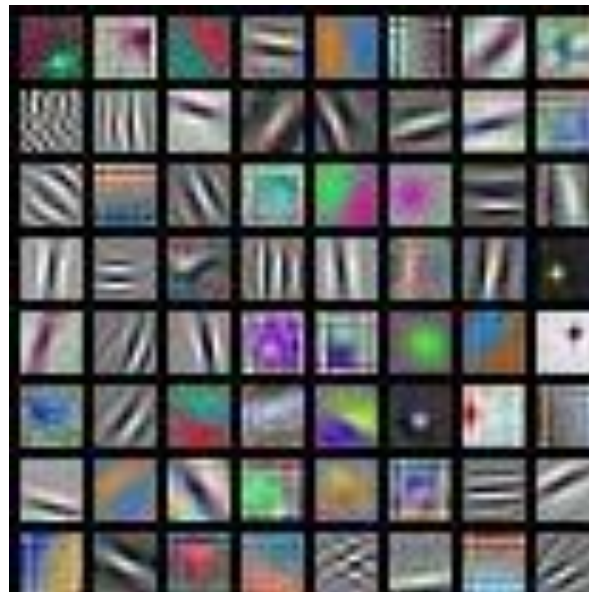
MLP: Bank of whole-image templates



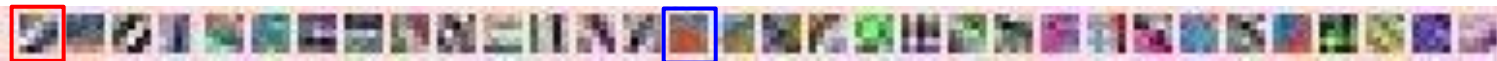
Preview: What do convolutional filters learn?



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



AlexNet: 64 filters, each
3x11x11



one filter => one
activation map

Activations:

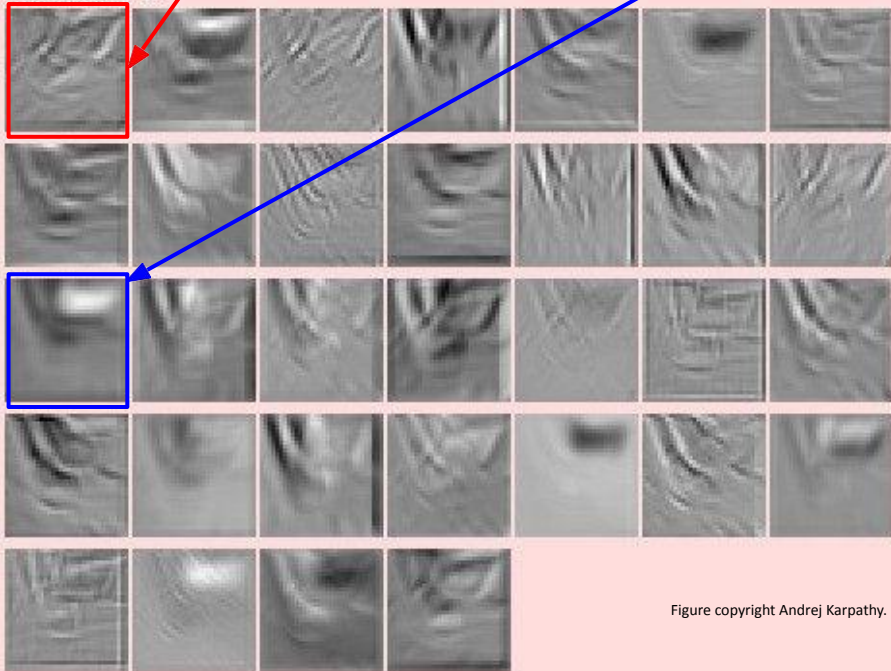



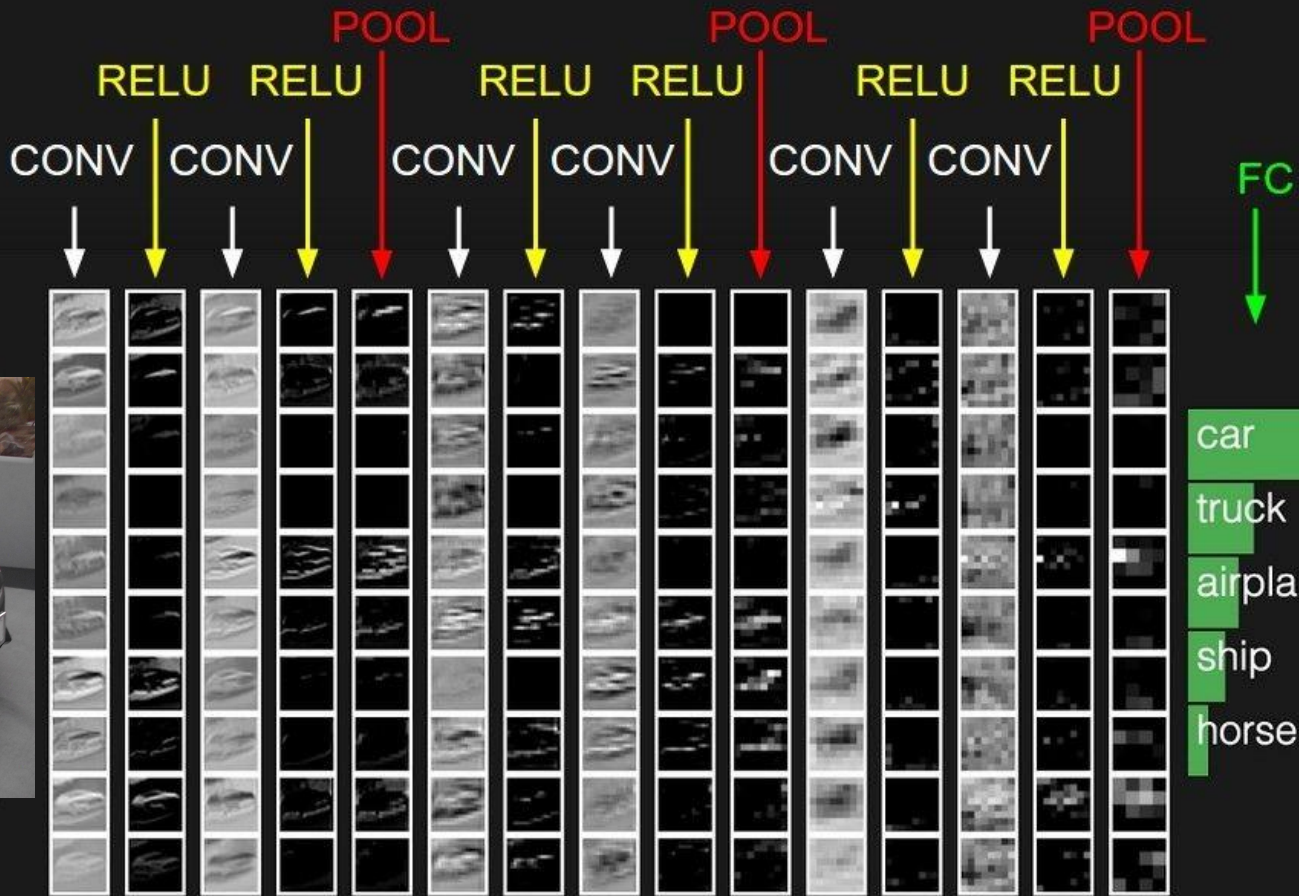
Figure copyright Andrej Karpathy.

example 5x5 filters
(32 total)

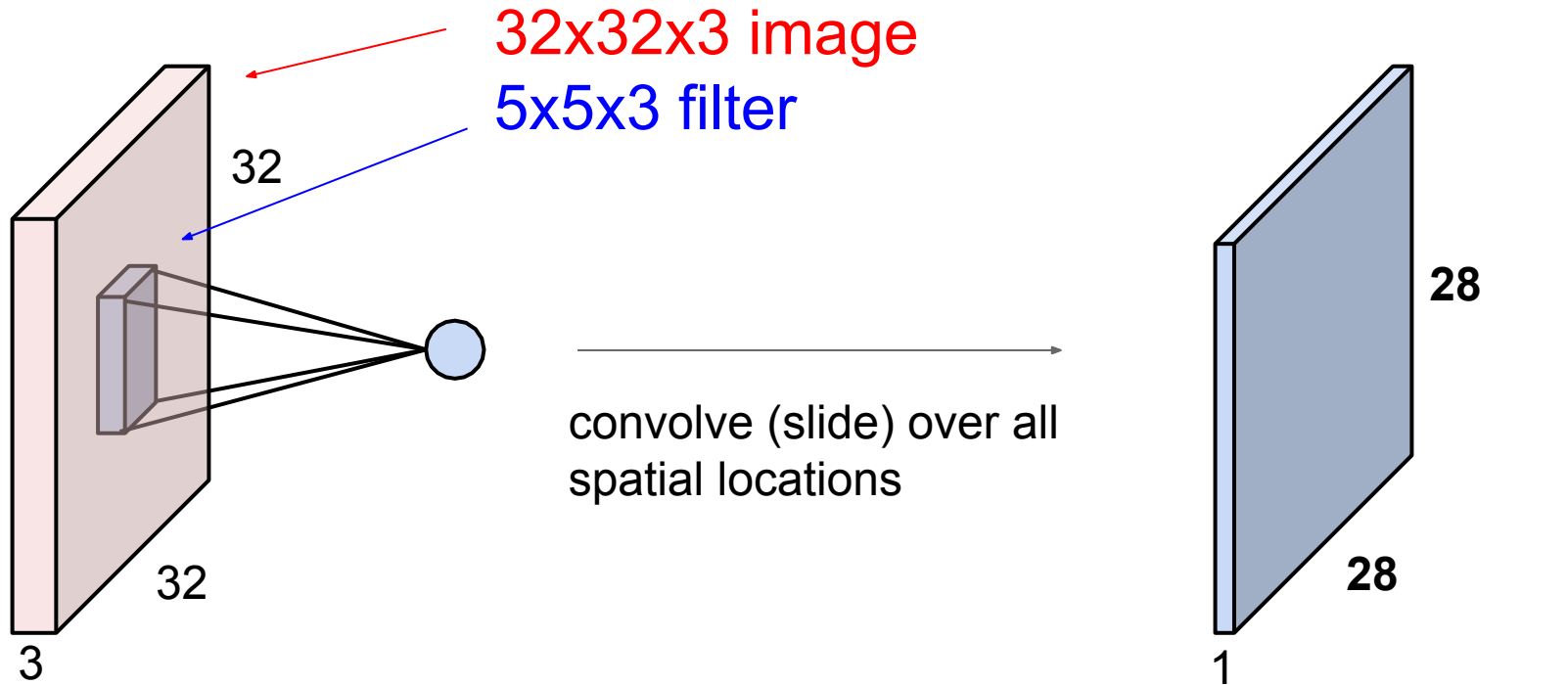
We call the layer convolutional
because it is related to convolution
of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2]$$


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

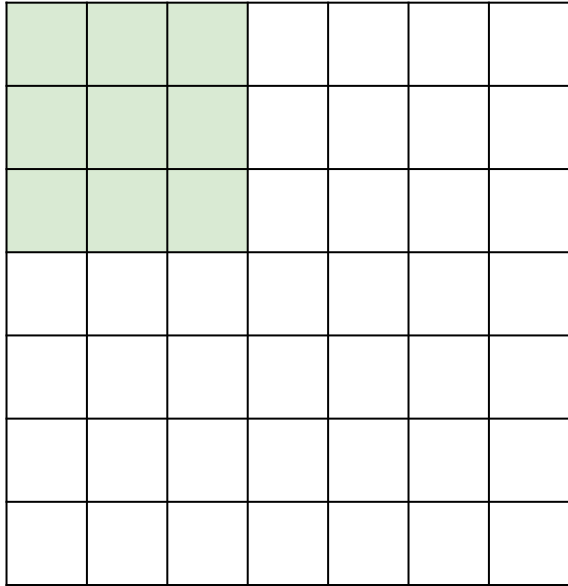


A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

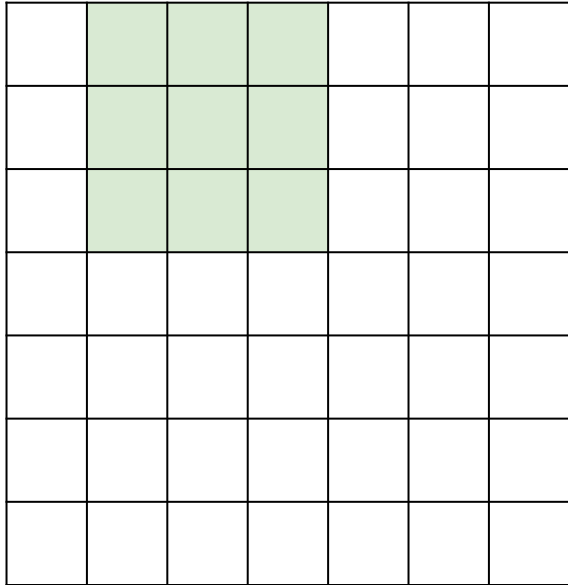


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

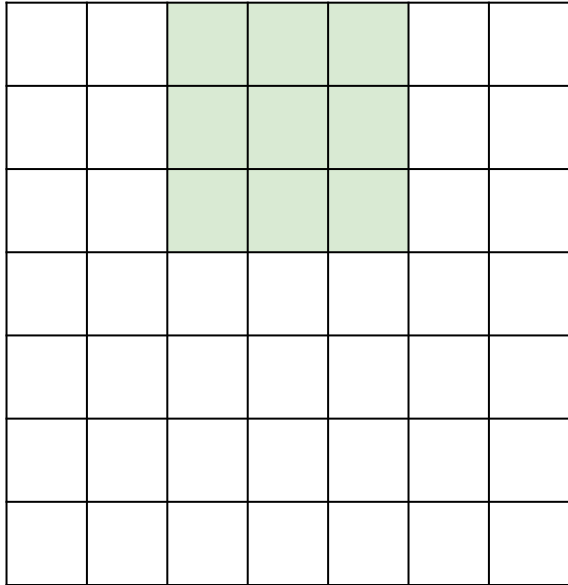


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

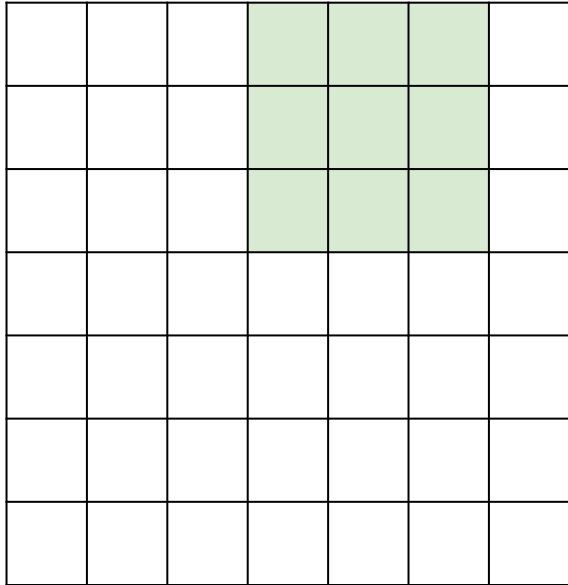


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

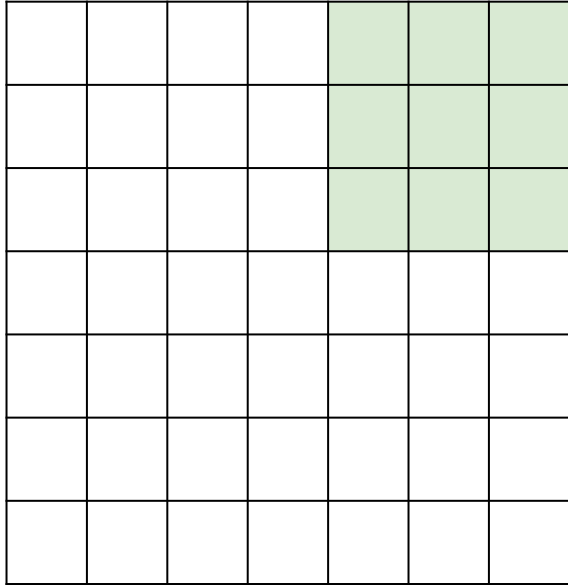


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7



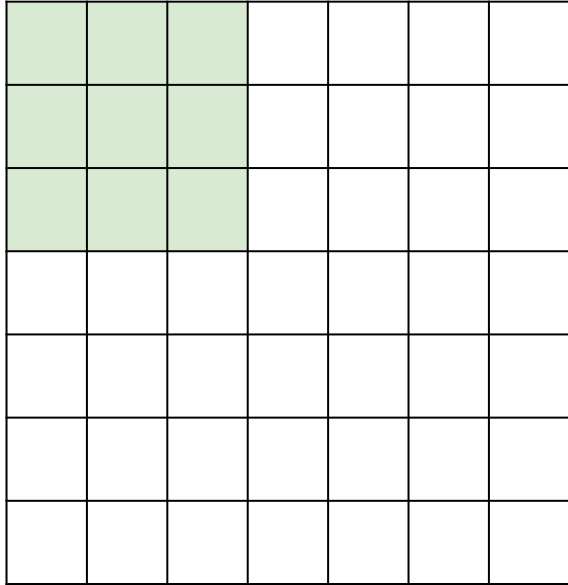
7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

7

A closer look at spatial dimensions:

7

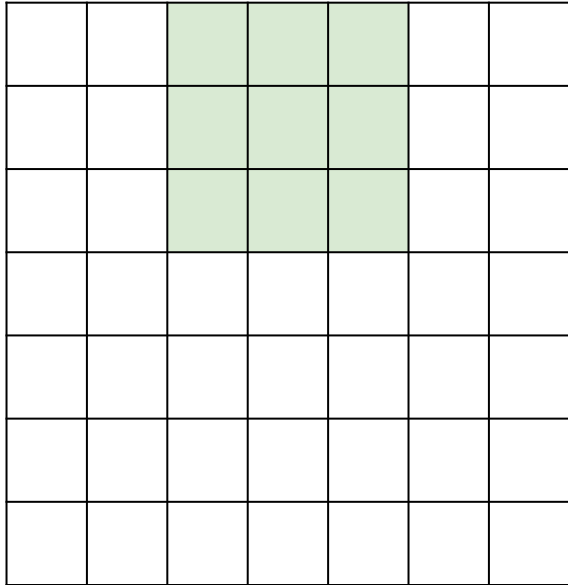


7

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

A closer look at spatial dimensions:

7

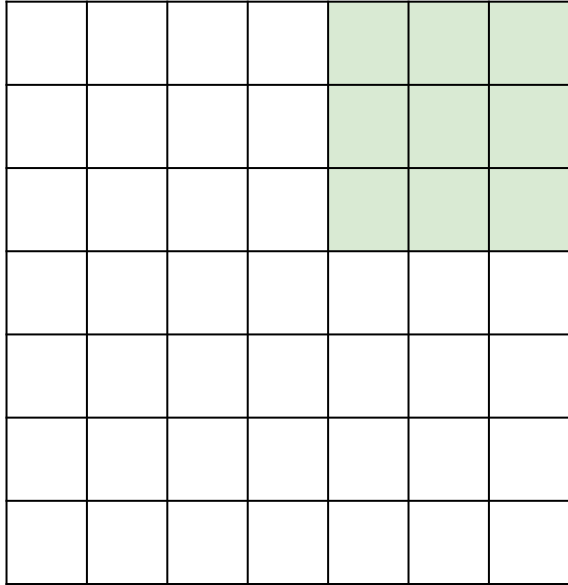


7

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

A closer look at spatial dimensions:

7

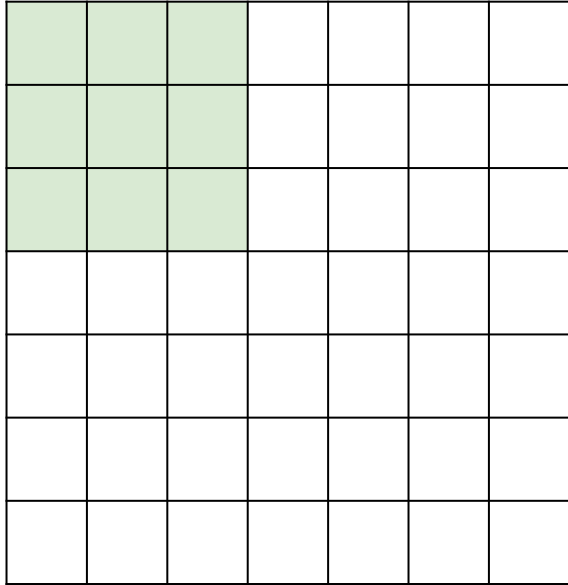


7

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

A closer look at spatial dimensions:

7

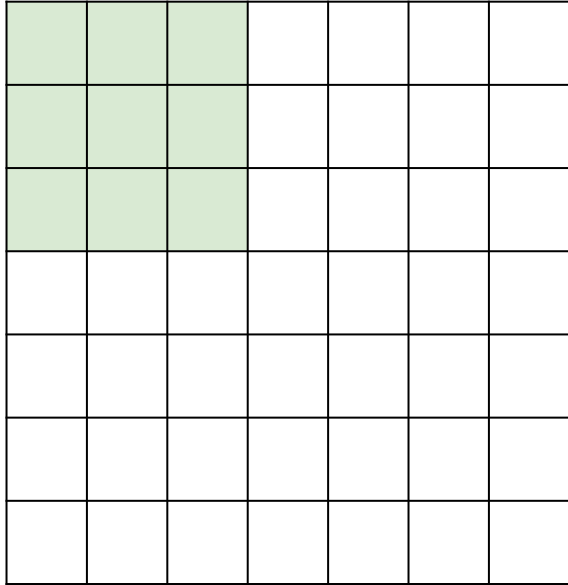


7

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

A closer look at spatial dimensions:

7

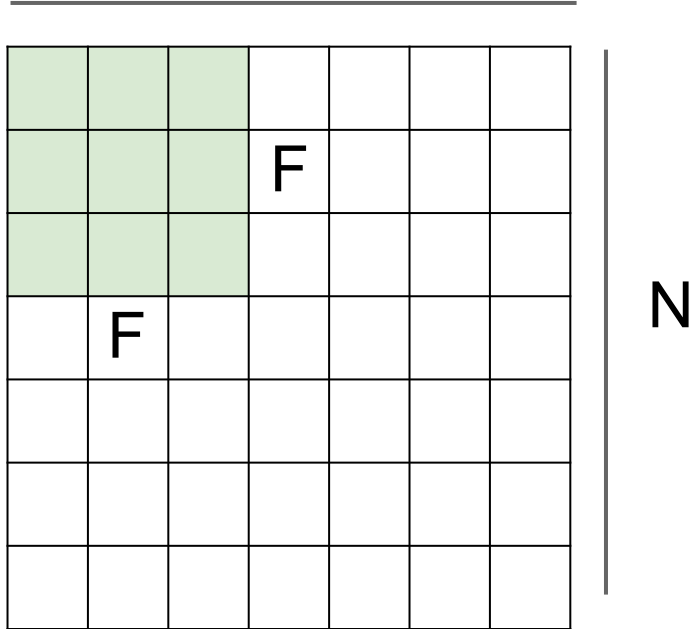


7

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

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

N



Output size:

$$(N - F) / \text{stride} + 1$$

e.g. $N = 7$, $F = 3$:

$$\text{stride } 1 \Rightarrow (7 - 3) / 1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3) / 2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3) / 3 + 1 = 2.33 \text{ :}\backslash$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

(recall:)

$$(N + 2P - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

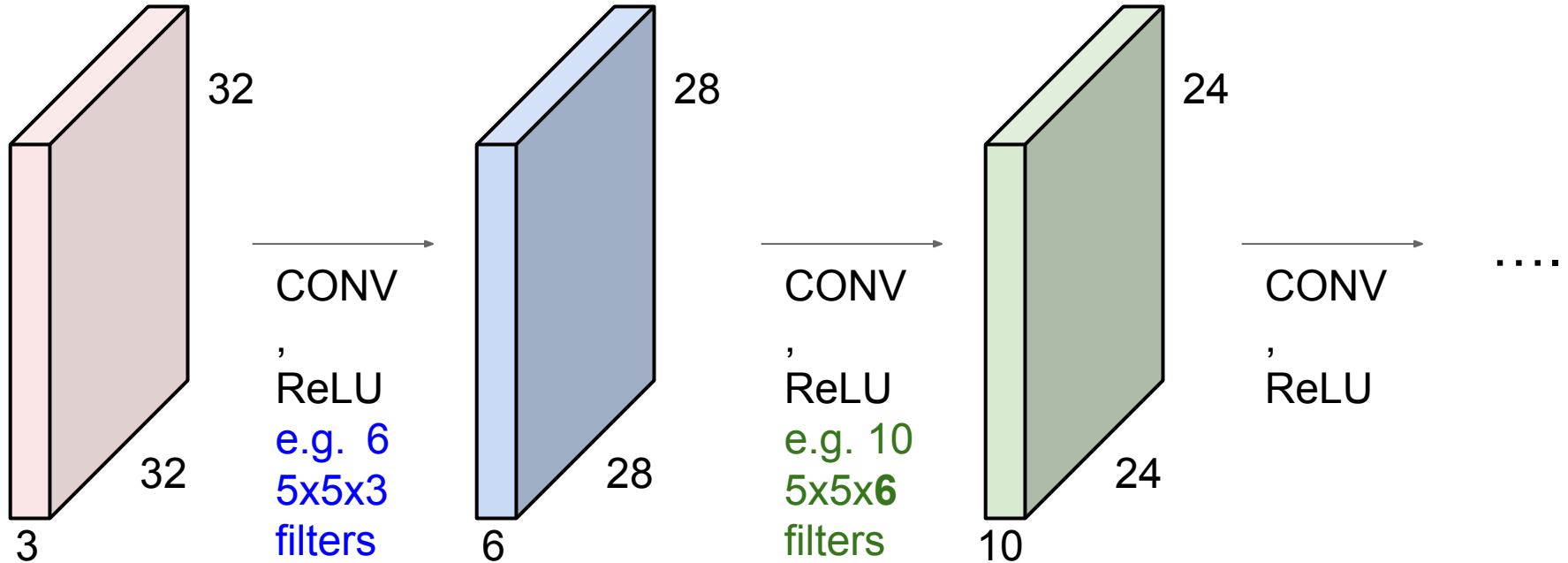
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

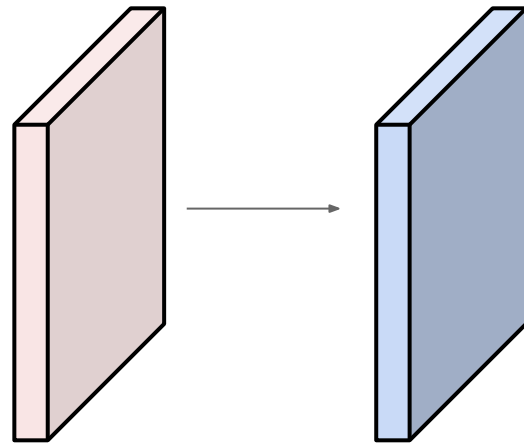


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:

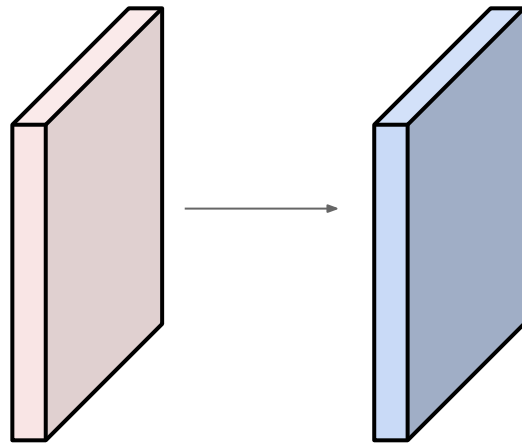
Input volume: **32x32x3**

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

Output volume size:

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

32x32x10

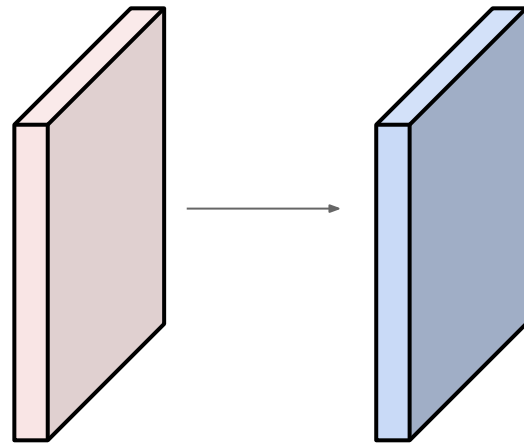


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

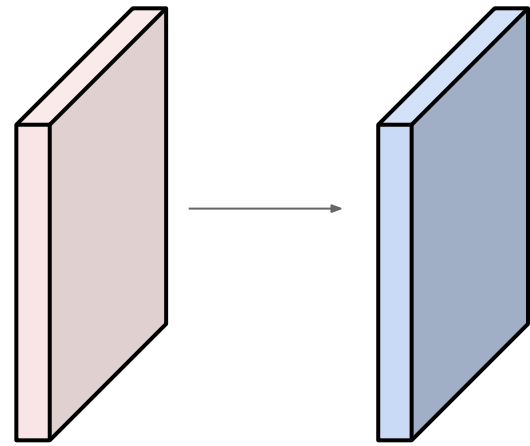
Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

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



Number of parameters in this layer?

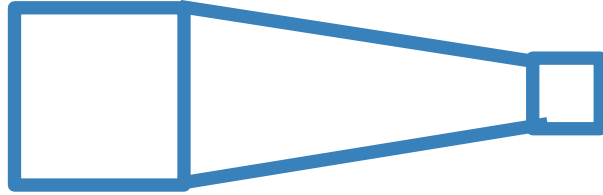
each filter has $5*5*3 + 1 = 76$ params

(+1 for bias)

$\Rightarrow 76*10 = 760$

Receptive Fields

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

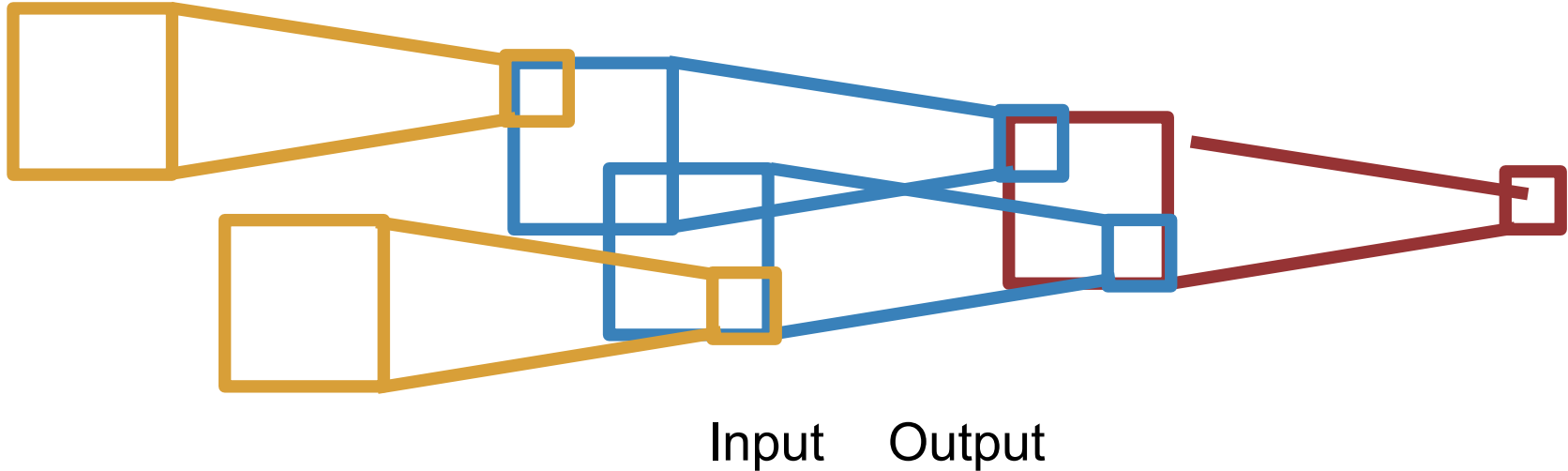


Input

Output

Receptive Fields

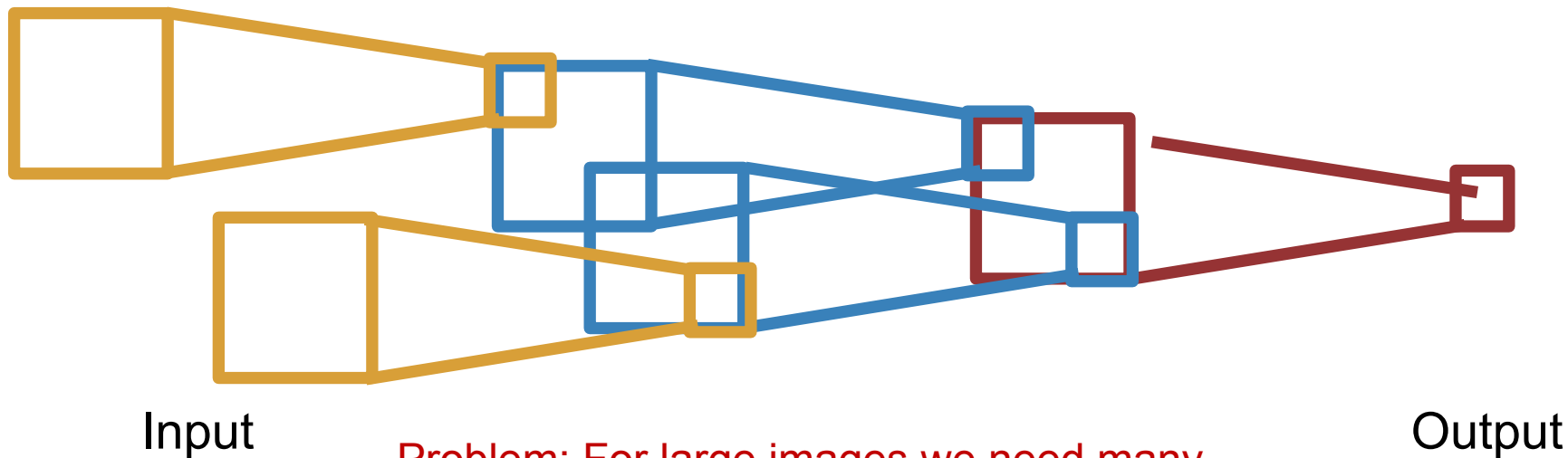
Each successive convolution adds $K - 1$ to the receptive field size with L layers the receptive field size is $1 + L * (K - 1)$



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

Receptive Fields

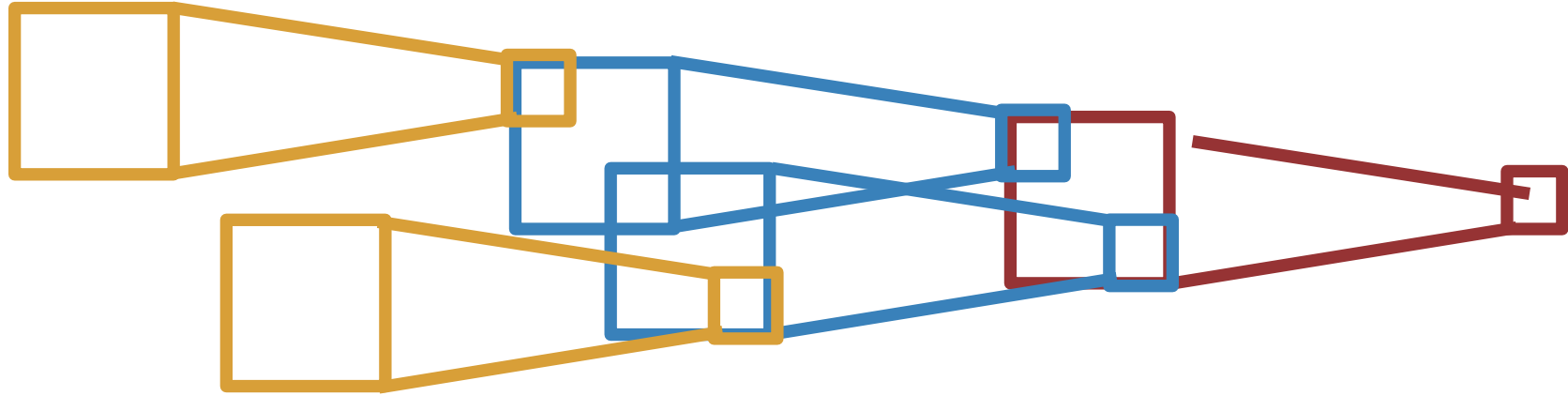
Each successive convolution adds $K - 1$ to the receptive field size. With L layers the receptive field size is $1 + L * (K - 1)$.



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

Receptive Fields

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



Input

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

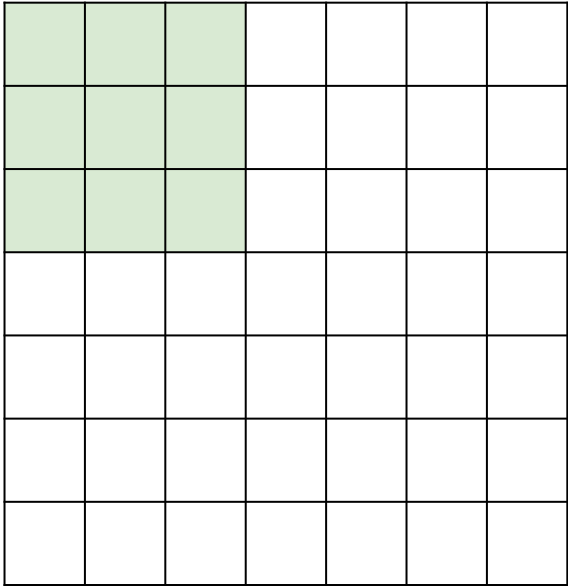
Output

Solution: Downsample inside the network

Slide inspiration: Justin Johnson

Solution: **Strided** Convolution

7

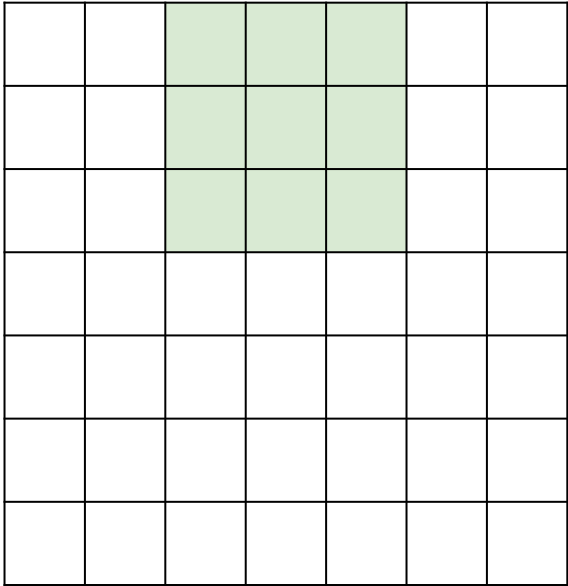


7

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

Solution: **Strided** Convolution

7



7

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

=> 3x3 output!

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$ 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 $W_2 \times H_2 \times K$ where:

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

Number of parameters: $F^2 C K$ and K biases

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

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 $W_2 \times H_2 \times K$

where:

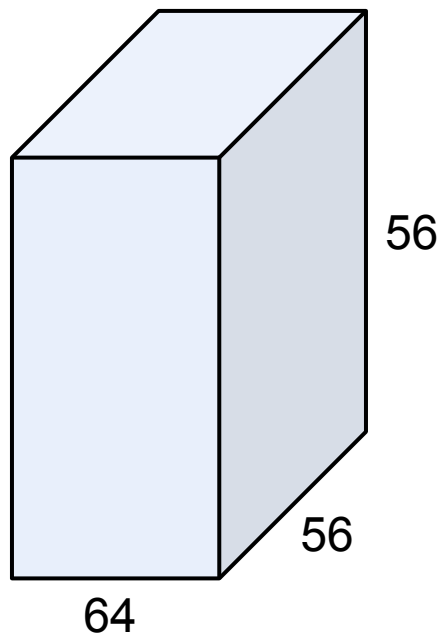
- $W_2 = (W_1 - F + 2P) / S + 1$
- $H_2 = (H_1 - F + 2P) / S + 1$

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)
- $F = 1, S = 1, P = 0$

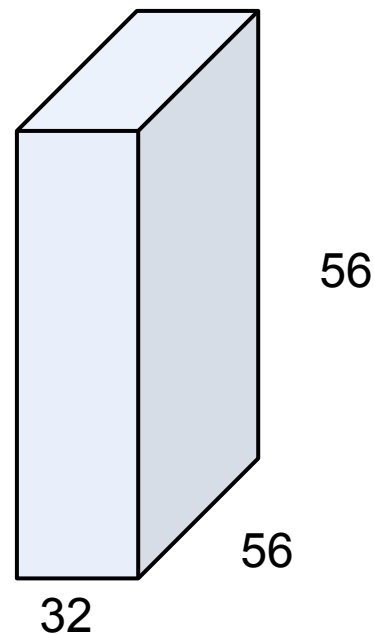
(btw, 1x1 convolution layers make perfect sense)



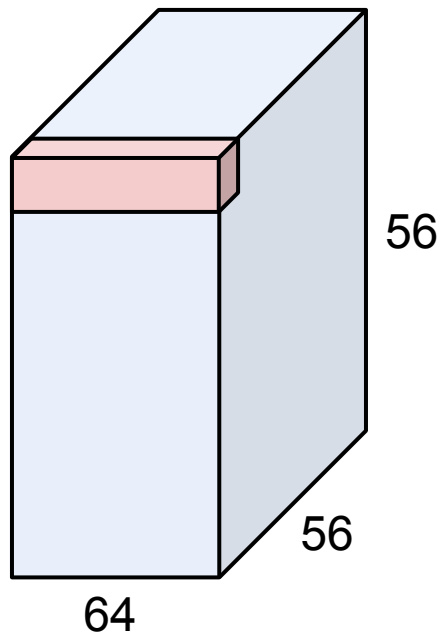
1x1 CONV
with 32 filters

→

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



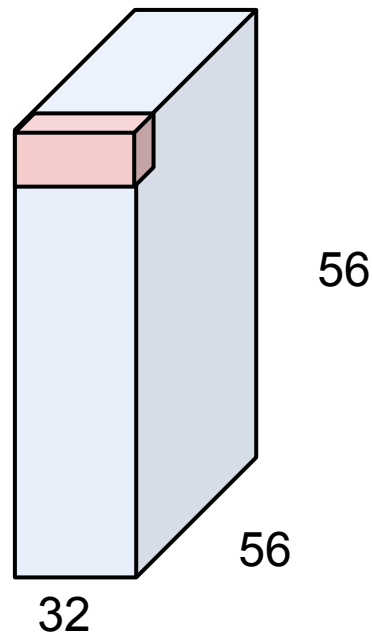
(btw, 1x1 convolution layers make perfect sense)



1x1 CONV
with 32 filters

→

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



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 $(N, C_{out}, H_{out}, W_{out})$ can be precisely described as:

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

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: $\begin{bmatrix} C_{out} \\ C_{in} \end{bmatrix}$.

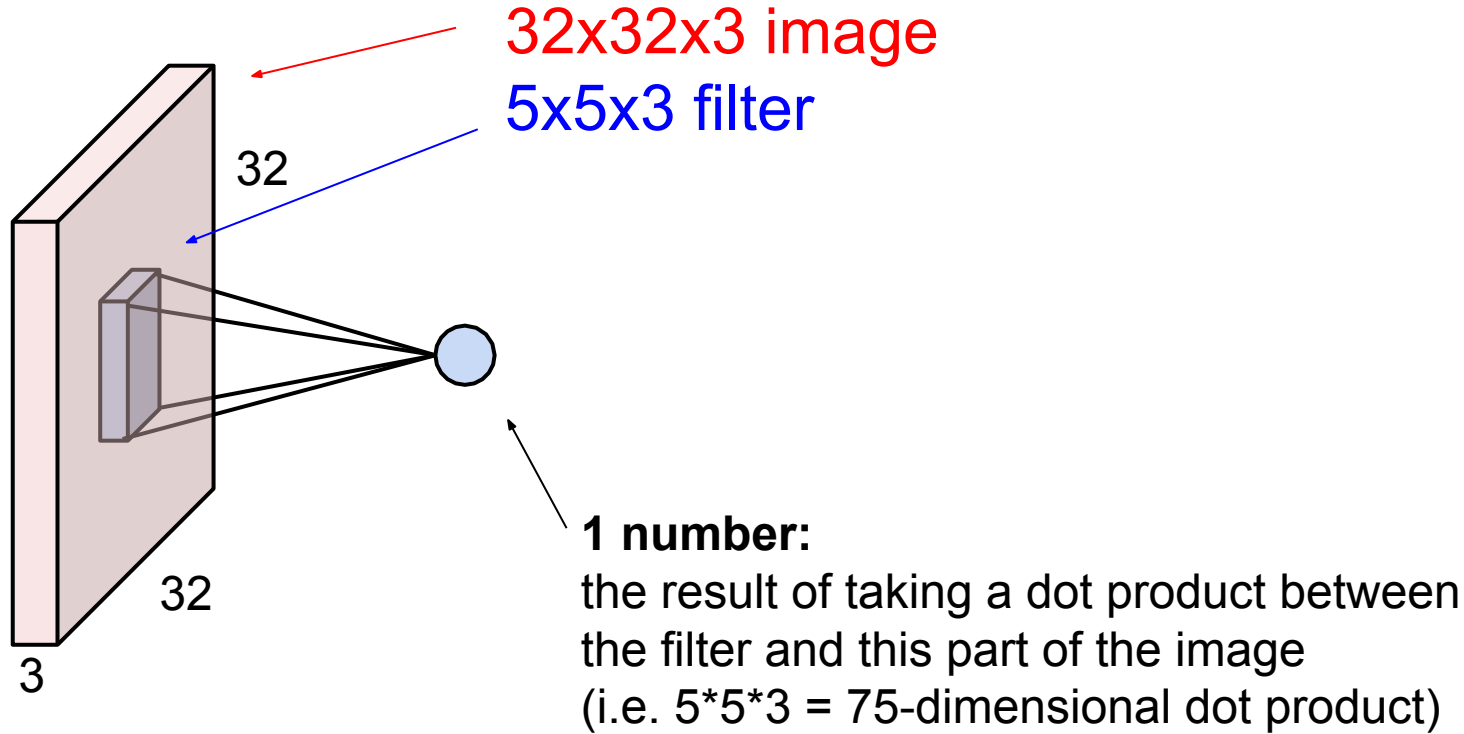
The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- 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

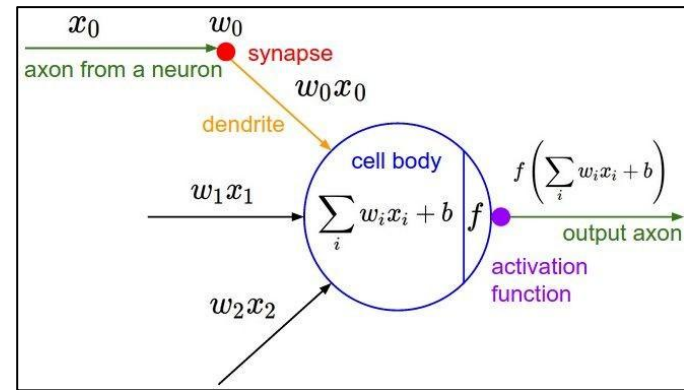
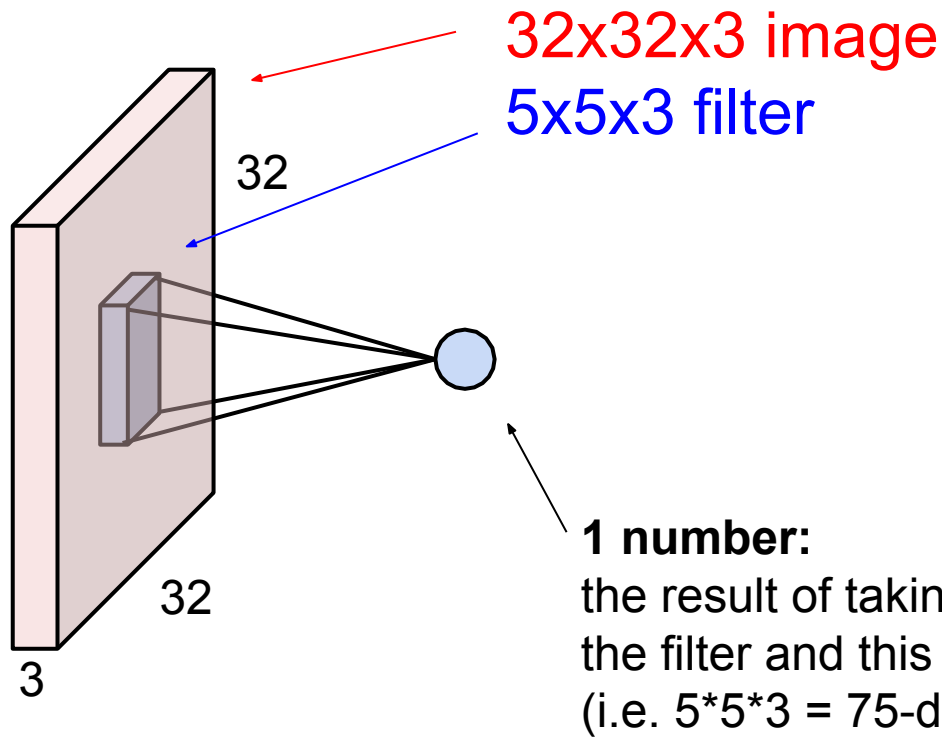
Conv layer needs 4 hyperparameters:

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

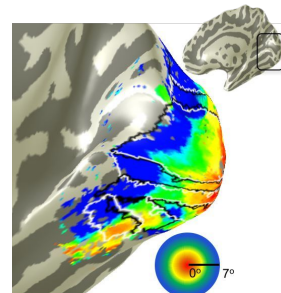
The brain/neuron view of CONV Layer



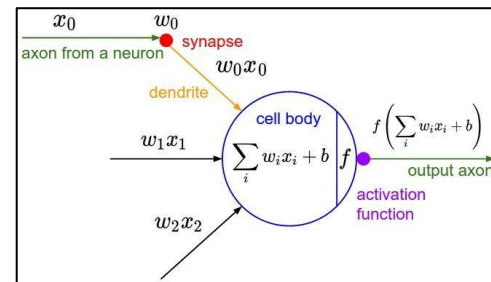
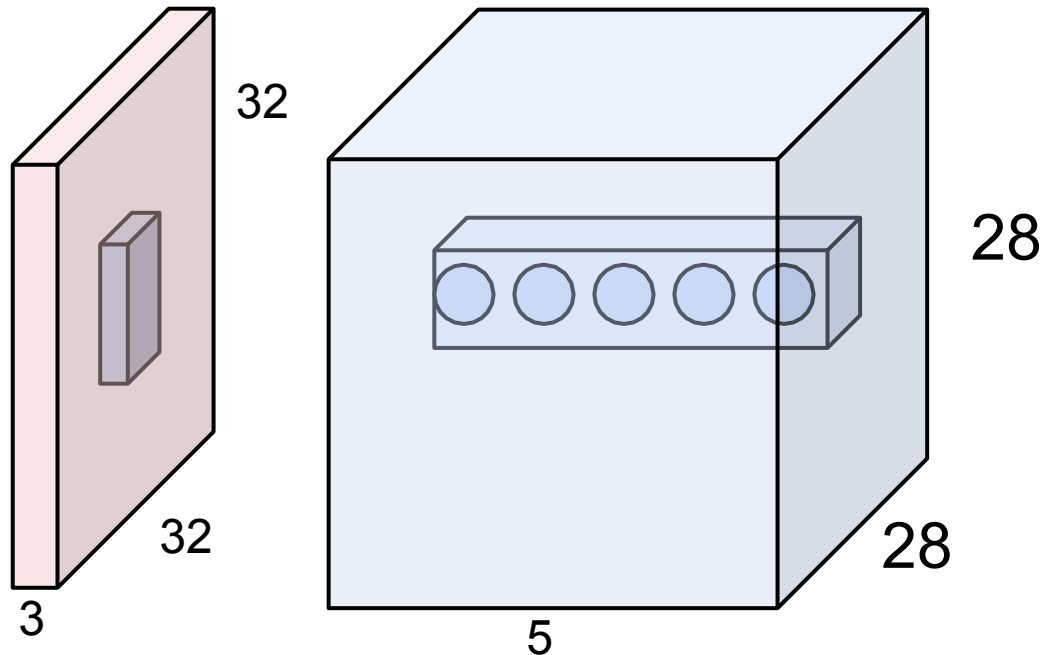
The brain/neuron view of CONV Layer



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)

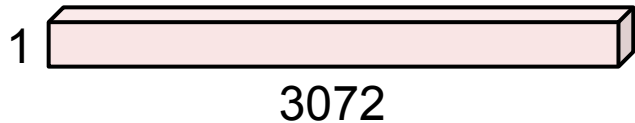
There will be 5 different
neurons all looking at the same
region in the input volume

Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron
looks at the full
input volume

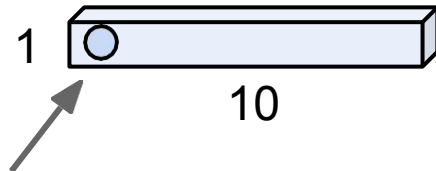
input



Wx

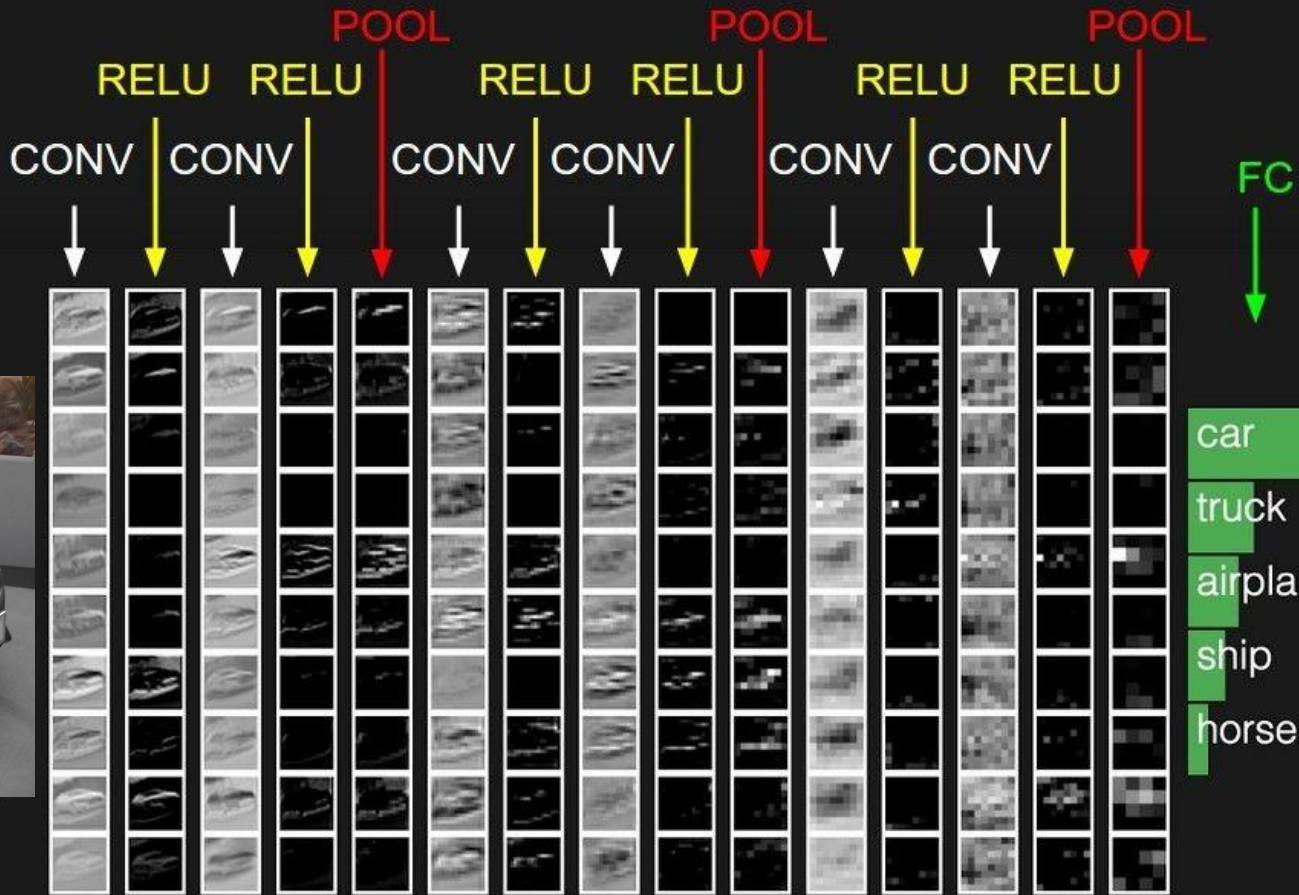
10 x 3072
weights

activation



1 number:

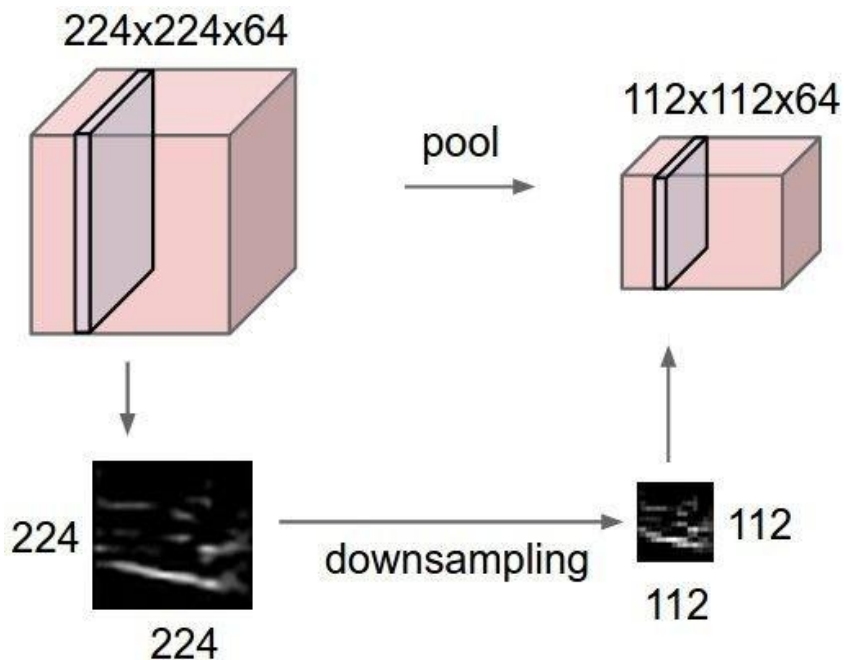
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)



Convolution demo

Pooling layer

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



MAX POOLING

Single depth slice

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

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

- No learnable parameters
- Introduces spatial invariance

Pooling layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

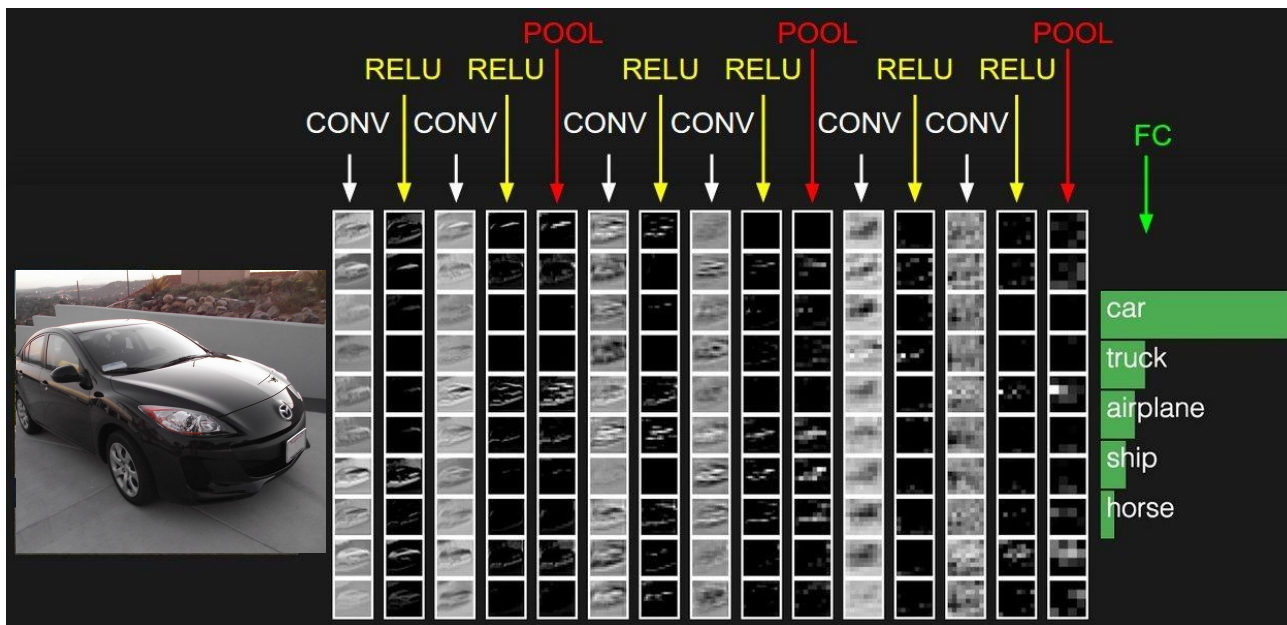
This will produce an output of $W_2 \times H_2 \times C$ where:

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

Number of parameters: 0

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

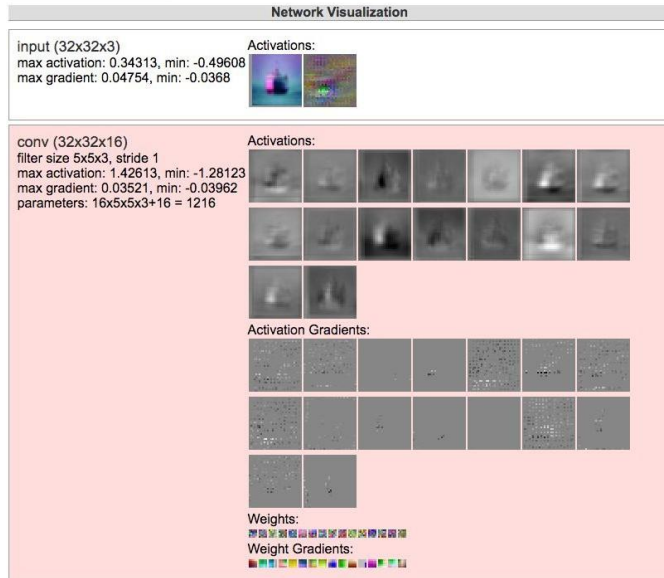
Description

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.

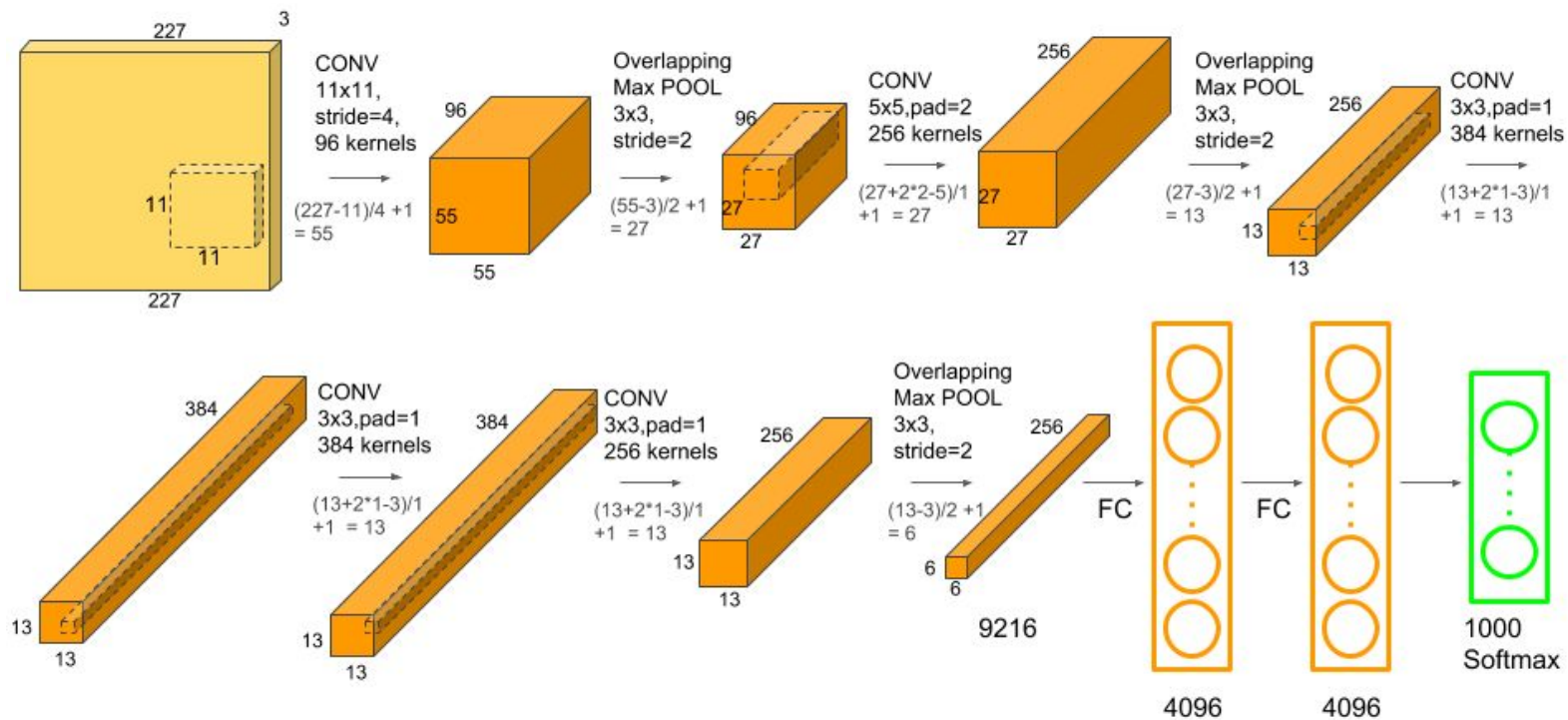
By default, in this demo we're using Adadelata which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



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

AlexNet



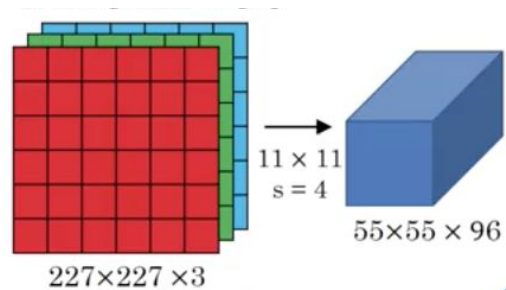
Conv1 Parameters

$$W_c = K^2 \times C \times N$$

$$B_c = N$$

$$P_c = W_c + B_c$$

AlexNet Conv-1

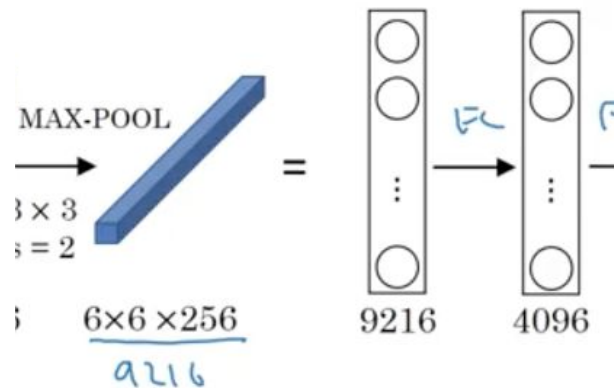


$$W_c = 11^2 \times 3 \times 96 = 34,848$$

$$B_c = 96$$

$$P_c = 34,848 + 96 = 34,944$$

FC layer parameters



$$W_{cf} = O^2 \times N \times F$$

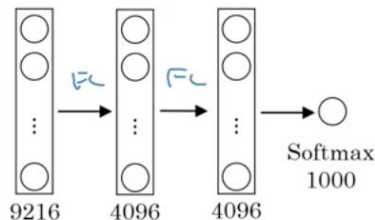
$$B_{cf} = F$$

$$P_{cf} = W_{cf} + B_{cf}$$

$$W_{cf} = 6^2 \times 256 \times 4096 = 37,748,736$$

$$B_{cf} = 4096$$

$$P_{cf} = W_{cf} + B_{cf} = 37,752,832$$



$$W_{ff} = F_{-1} \times F$$

$$B_{ff} = F$$

$$P_{ff} = W_{ff} + B_{ff}$$

$$W_{ff} = 4096 \times 1000 = 4,096,000$$

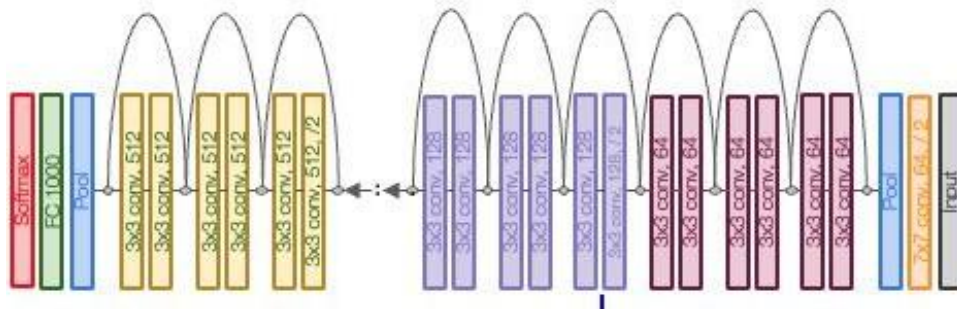
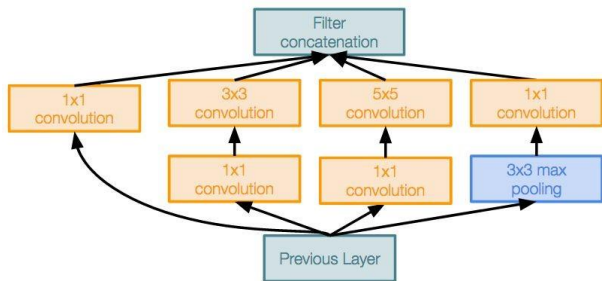
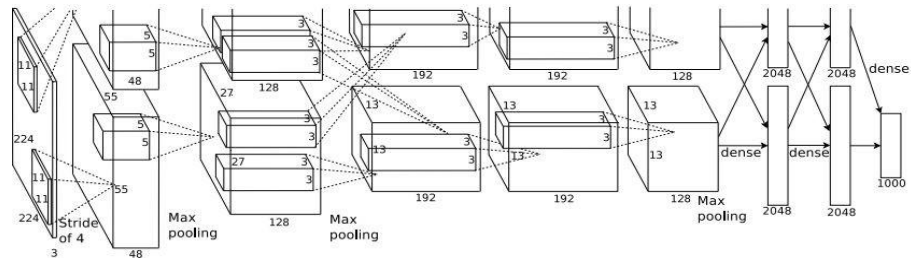
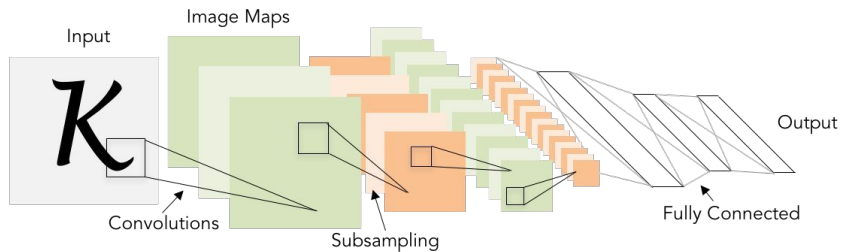
$$B_{ff} = 1,000$$

$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

Total parameters

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096x1	37,748,736	4,096	37,752,832
FC-2	4096x1	16,777,216	4,096	16,781,312
FC-3	1000x1	4,096,000	1,000	4,097,000
Output	1000x1	0	0	0
Total				62,378,344

CNN Architectures



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, $0 \leq K \leq 2$.

- But recent advances such as ResNet/GoogLeNet have challenged this paradigm