

Deep Learning

Lecture 4 – Convolutional Neural Networks 1



UPPSALA
UNIVERSITET

Joakim Lindblad

Department of Information Technology
Uppsala University

joakim.lindblad@it.uu.se

cb.uu.se/~joakim

Outline

Recap

Deep convolutional neural networks

Deep neural networks

Convolutional neural networks

Filtering and convolutions

All about convnets

Concluding

Further reads/links

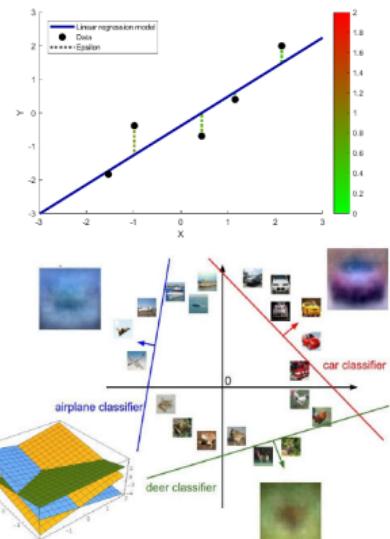
- Stanford CS231n deep learning course by Fei Fei's group, <http://cs231n.stanford.edu/syllabus.html>, 2016 version (skip to 2nd lecture, w. A. Karpathy)
<https://www.youtube.com/watch?v=g-PvXUjD6qg&list=PLlJy-eBtNFt6EuMxFYRiNRS07MCWN5UIA>
2017 version <https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv>
- 3blue1brown https://www.youtube.com/playlist?list=PLZHQObOWTQDNu6R1_67000Dx_ZCJB-3pi
- Machine learning by Andrew Ng (Coursera) <https://www.youtube.com/playlist?list=PLZ9qNFMHZ-A4rycgrg0Yma6zxF4BZGG>
<http://deeplearning.stanford.edu/tutorial/>
- Deep learning and reinforcement learning summer school http://videolectures.net/DLRLsummerschool2018_toronto/
- Ian Goodfellow's book on deep learning <http://www.deeplearningbook.org/>
- Stat212b: Topics Course on Deep Learning <http://joanbruna.github.io/stat212b/>
- Yann LeCun's "Gradient-based learning applied to document recognition"
<http://ieeexplore.ieee.org/document/726791/?arnumber=726791>
- An overview of gradient descent optimization algorithms <http://ruder.io/optimizing-gradient-descent/>
- Get going in MATLAB (if you're more confident there than in Python)
<https://mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html>
- WILDML <http://www.wildml.com/>
- Deep Learning Glossary <http://www.wildml.com/deep-learning-glossary/>
- Colah's blog <http://colah.github.io/>
- <https://icml.cc/virtual/2020/events/Tutorial>,
https://icml.cc/Conferences/2019/ScheduleMultitrack?session=&event_type=Tutorial&,
<https://icml.cc/Conferences/2018/Schedule?type=Tutorial>
- <https://arxiv.org> & <http://www.arxiv-sanity.com/> & <https://paperswithcode.com/>
- The annual report of <https://aiindex.org>
- And many many more ... (Google is your friend :-)



Recap

Recap

- Linear regression $y = Wx + b$
 - Maximum likelihood → Least squares optimization problem: $\hat{w}, \hat{b} = \operatorname{argmin}_{w,b} \sum_{i=1}^n (wx_i + b - y_i)^2$
- A linear classifier $y = f(Wx + b)$ – *Logistic regression*
 - Multi-class case encoding as a "one hot" vector
- Send the output through a nonlinearity (activation function) $y = f(Wx + b)$
- Send the output to another classifier, and another...
 $y = f(W^{(3)}f(W^{(2)}f(W^{(1)}x + b^{(1)}) + b^{(2)}) + b^{(3)}) = A$
Neural network
- Measure performance with a *loss function*
- Use Stochastic Gradient Descent (SGD) to minimize the loss
 - Take small steps in the direction of the negative gradient $\theta_{k+1} = \theta_k - \lambda \cdot \nabla_{\theta} L$, where stepsize (a.k.a. learning rate) λ typically in the range $[0.001, 0.01]$



Recap – Linear models

Linear regression

Output

$$y_i \in \mathbb{R}$$

Model

$$\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + b$$

Loss

$$L_i = (y_i - \hat{y}_i)^2$$

Logistic regression

Output

$$y_i \in \{0, 1\}$$

Model

$$p_i = \Pr(y_i = 1 | \mathbf{x}_i) = \frac{e^{\mathbf{w}^\top \mathbf{x}_i + b}}{1 + e^{\mathbf{w}^\top \mathbf{x}_i + b}}$$

Loss

$$L_i = -y_i \ln(p_i) - (1 - y_i) \ln(1 - p_i)$$

We find \mathbf{w} and b by minimizing sum of the losses

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w}, b}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L_i$$

- For linear regression the problem can be solved analytically.
- For logistic regression we need to use numerical optimization.

Interpreting a Linear Classifier: Visual Viewpoint

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



plane



car



bird



cat



deer



dog



frog



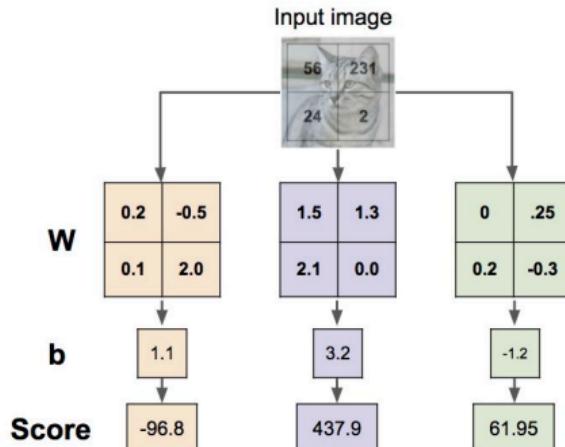
horse



ship



truck



Score

1.1

-96.8

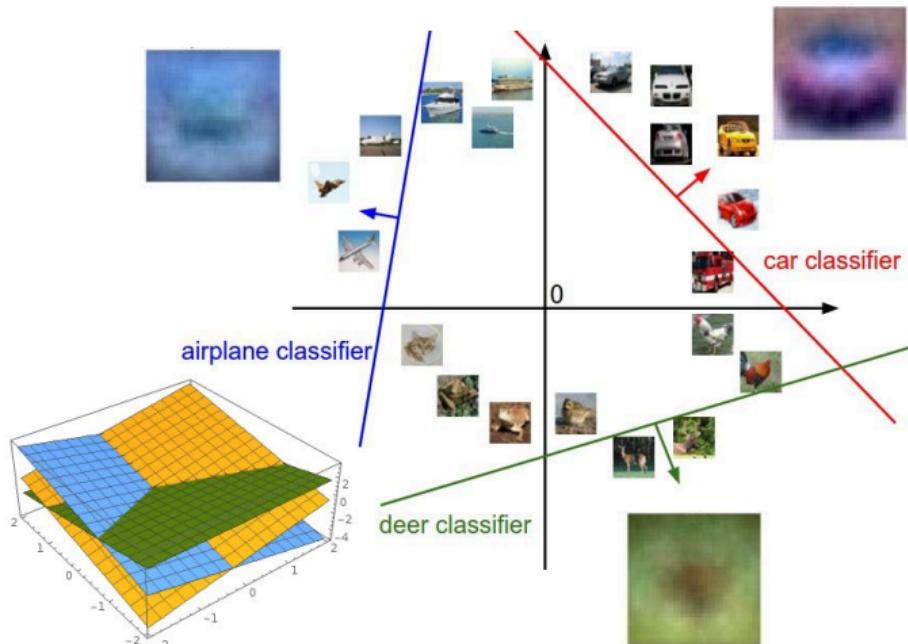
3.2

437.9

-1.2

61.95

Interpreting a Linear Classifier: Geometric Viewpoint



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



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

Plot created using [Wolfram Cloud](#)

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

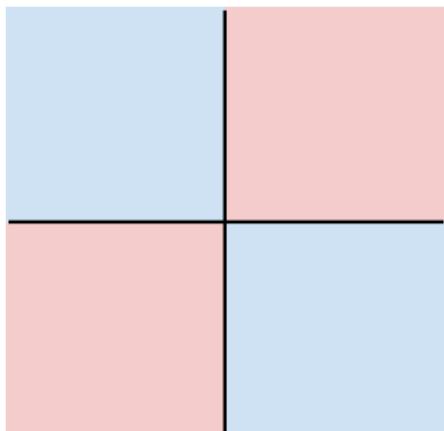
Hard cases for a linear classifier

Class 1:

First and third quadrants

Class 2:

Second and fourth quadrants

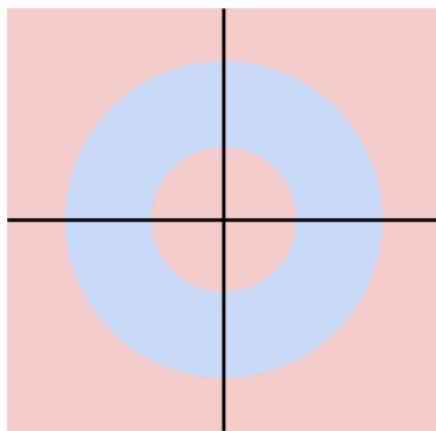


Class 1:

$1 \leq L_2 \text{ norm} \leq 2$

Class 2:

Everything else

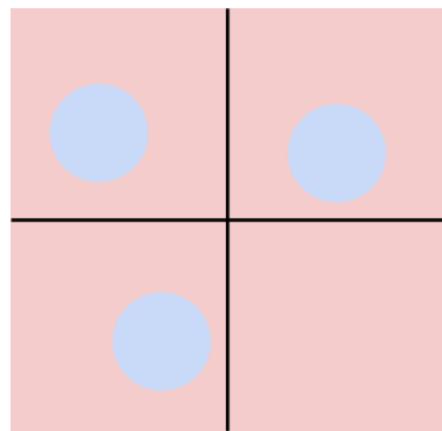


Class 1:

Three modes

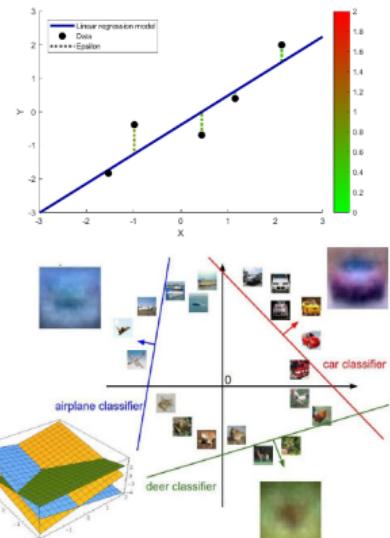
Class 2:

Everything else

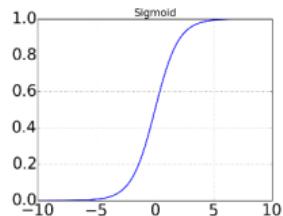


Recap

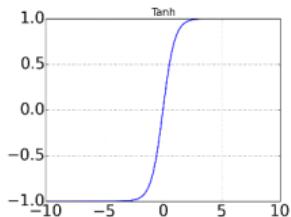
- Linear regression $y = Wx + b$
 - Maximum likelihood → Least squares optimization problem: $\hat{w}, \hat{b} = \operatorname{argmin}_{w,b} \sum_{i=1}^n (wx_i + b - y_i)^2$
- A linear classifier $y = f(Wx + b)$ – *Logistic regression*
 - Multi-class case encoding as a "one hot" vector
- Send the output through a nonlinearity, *activation function* $y = f(Wx + b)$
- Send the output to another classifier, and another...
 $y = f(W^{(3)}f(W^{(2)}f(W^{(1)}x + b^{(1)}) + b^{(2)}) + b^{(3)}) = A$
Neural network
- Measure performance with a *loss function*
- Use Stochastic Gradient Descent (SGD) to minimize the loss
 - Take small steps in the direction of the negative gradient $\theta_{k+1} = \theta_k - \lambda \cdot \nabla_{\theta} L$, where stepsize (a.k.a. learning rate) λ typically in the range $[0.001, 0.01]$



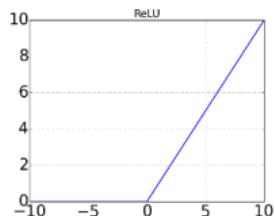
Recap – Activation functions $\sigma(x)$



$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\text{sigmoid}(2x) - 1$$



$$\text{ReLU}(x) = \max(0, x)$$

Recap

- A neural network is a sequential construction of several generalized linear regression models.

$$\mathbf{h}^{(0)} = \mathbf{x}$$

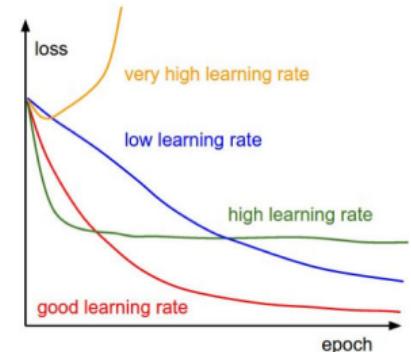
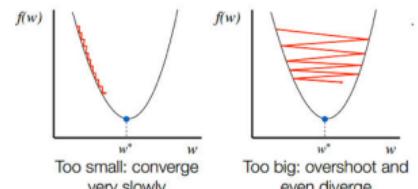
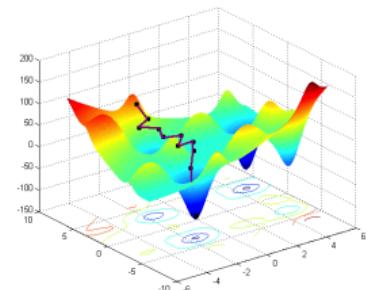
$$\mathbf{h}^{(l)} = \sigma \left(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \right), \quad l = 1, \dots, L-1$$

$$\hat{y} = \mathbf{W}^{(L)} \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}$$

- All weight matrices and offset vectors in all layers combined are the parameters of the network

$\boldsymbol{\theta} = [\text{vec}(\mathbf{W}^{(1)})^\top, \mathbf{b}^{(1)\top}, \dots, \text{vec}(\mathbf{W}^{(L)})^\top, \mathbf{b}^{(L)\top}]^\top$, which constitutes the parametric model $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$.

- We train our models by considering the optimization problem $\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$
- Optimization using Stochastic Gradient Descent (SGD)



Recap – Minibatch SG

1. Initialize θ_0 , set $k \leftarrow 1$, choose batch size n_b and number of epochs E .
 2. For $i = 1$ to E
 - (a) Randomly shuffle the training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$.
 - (b) For $j = 1$ to $\frac{n}{n_b}$
 - (i) Approximate the gradient of the loss function using the current minibatch $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=(j-1)n_b+1}^{jn_b}$
 - (ii) Take a step in the gradient direction $\hat{\mathbf{g}}_k = \frac{1}{n_b} \sum_{i=(j-1)n_b+1}^{jn_b} \nabla_{\theta} L_i(\theta) \Big|_{\theta=\theta_k} \approx \mathbf{g}_k$.
 - (iii) Update the iteration index $k \leftarrow k + 1$.
-

At each iteration we get a stochastic approximation $\hat{\mathbf{g}}_k$ of the true gradient

$$\mathbf{g}_k = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L_i(\theta) \Big|_{\theta=\theta_k}$$

Recap – Back-propagation

Back-propagation - "Backprop" computing partial derivatives (e.g. in a *computational graph*) $\nabla_{\theta}L(\theta; \mathbf{x}) = (\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots)$

- Forward pass, backward pass, parameter update, and iterate . . .
- Follow the chain rule
- Find the analytical gradient of each component (incl. activation and loss functions)
- Total gradient is the product of the gradient at that node multiplied with the gradient coming to the node from the output side (propagating backwards)
- If connected to multiple nodes, add the gradients
- For ReLU activations gradient pass to positive outputs only
- For deep networks, very many values are multiplied together → problematic with vanishing or exploding gradients

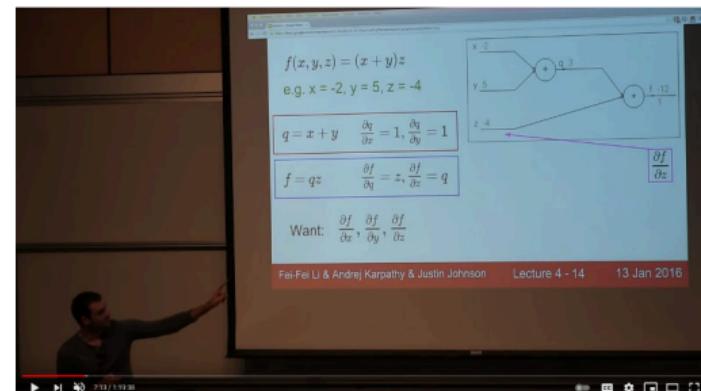
Become a backprop ninja!

Yes you should understand backprop!

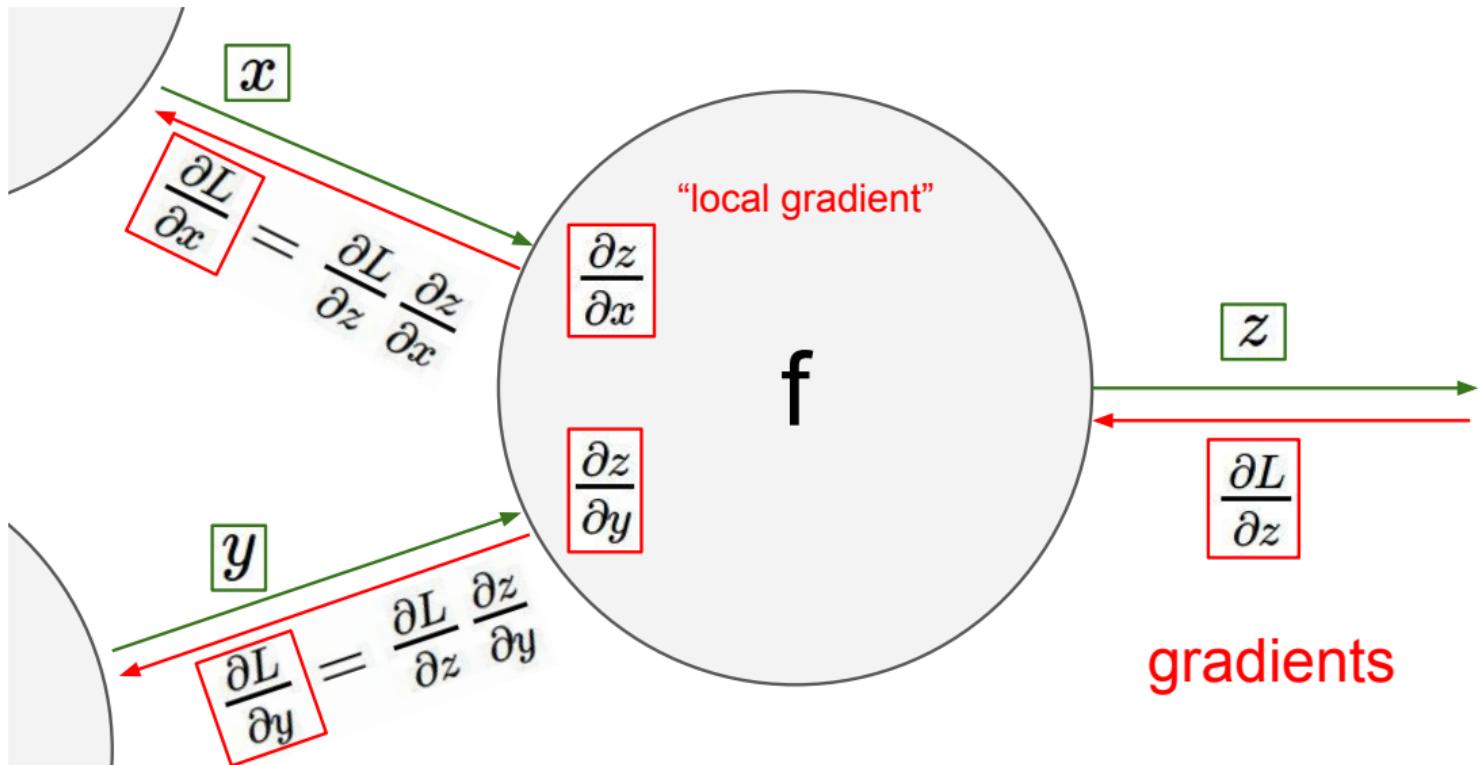
<https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>

Watch Andrej talk about Backprop:

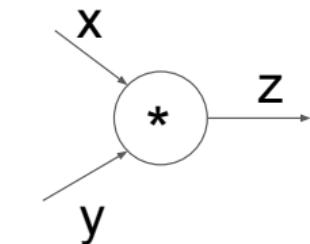
Stanford CS231n Winter 2016: Lecture 4:
Backpropagation, Neural Networks 1



<https://www.youtube.com/watch?v=i940vYb6noo&t=119s>



Modularized implementation: forward / backward API



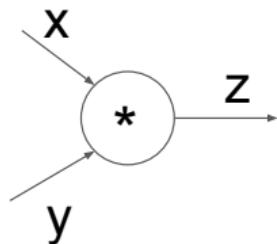
(x, y, z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$$\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial x}$$

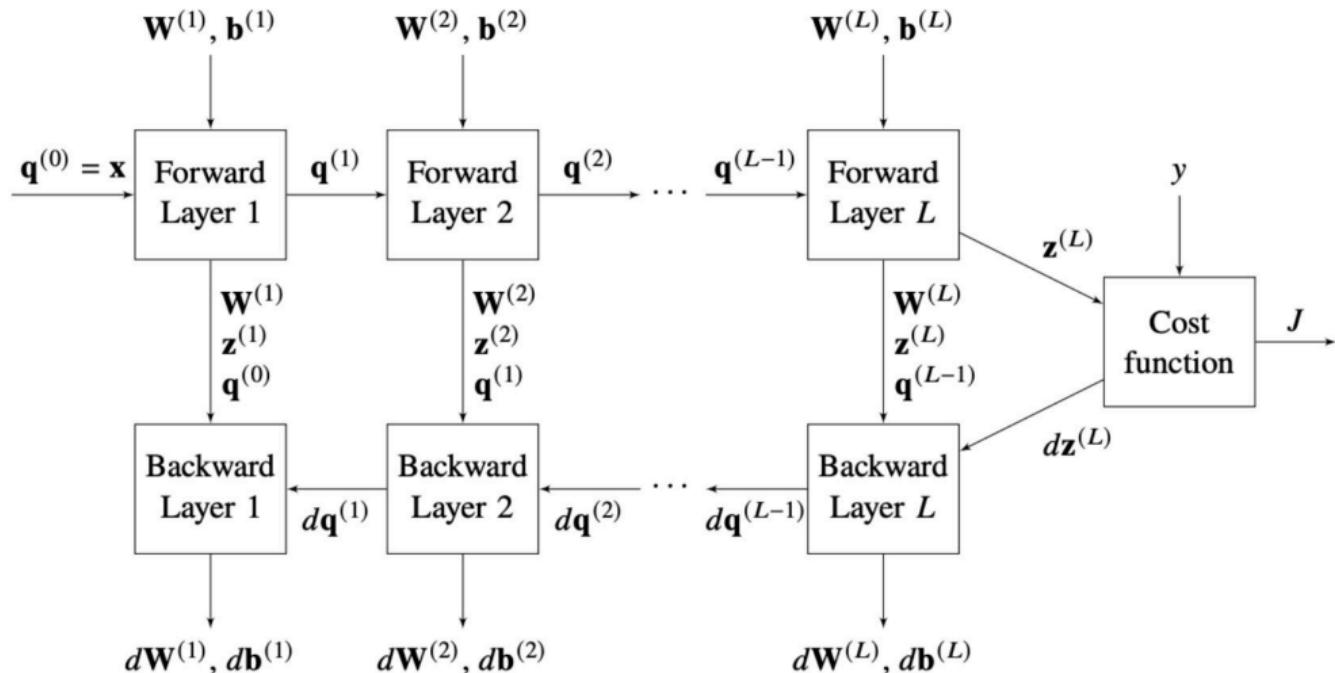
Modularized implementation: forward / backward API



(x, y, z are scalars)

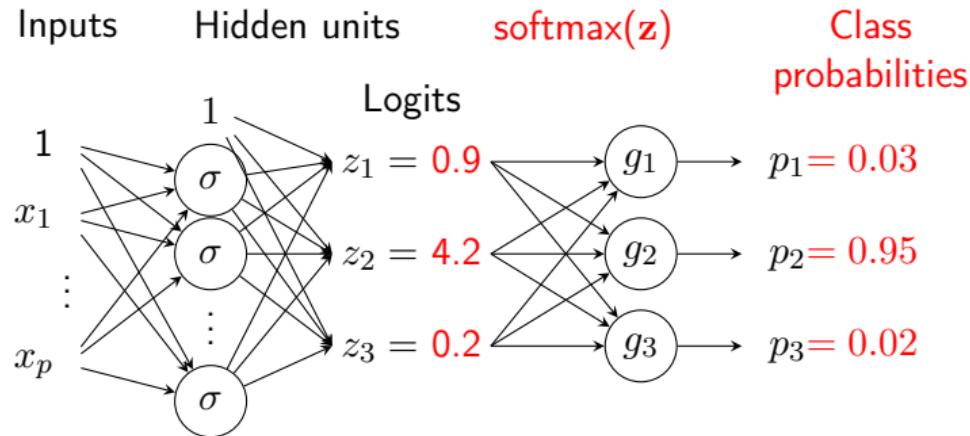
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Computational graph – backpropagation



NN classification with $K = 3$ classes

Consider an example with three classes $K = 3$.



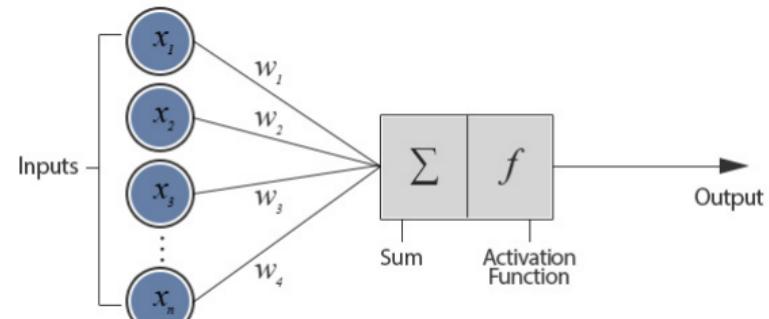
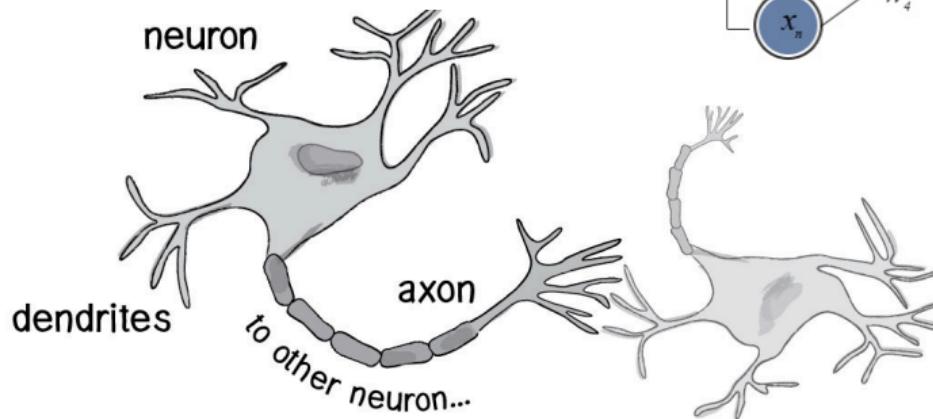
where

$$g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{l=1}^K e^{z_l}}, \quad \text{softmax}(\mathbf{z}) = [g_1(\mathbf{z}), \dots, g_K(\mathbf{z})]^\top.$$



Deep convolutional neural networks

Artificial neuronal networks



What used to be seen as a deep neural network...

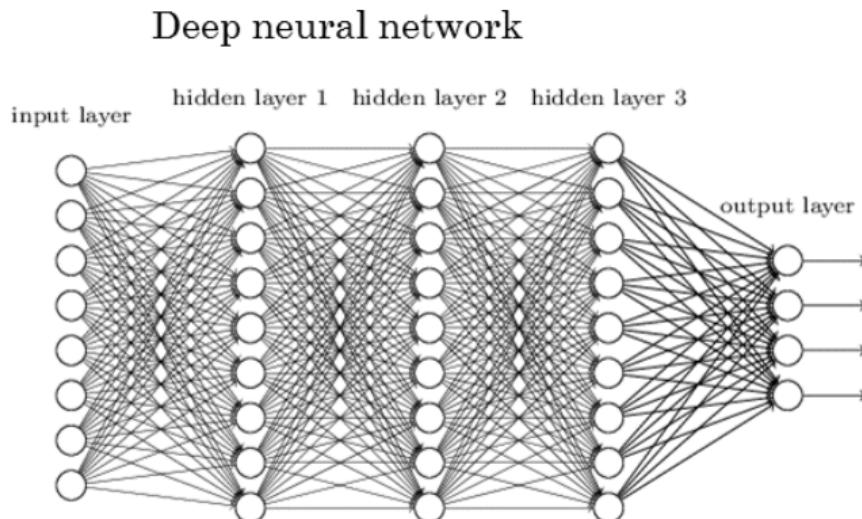


Figure: Fully connected Neural network

Src. <http://www.rsipvision.com/exploring-deep-learning/>

- The number of weights grows super fast!
 - Computationally heavy! Memory hungry! (Think of a 1000×1000 px image.)
 - Difficult to train! Slow convergence, needing very large amounts of training data!

Fully connected neural networks...

Fully connected neural networks

- connect every input with every other
- every input is treated equally, it is one dimension in a high dimensional space
- no concern for spatial structure
- every input is equally "near" any other input
- the upper left part of an image is completely independent from the upper right, and the lower left, and the lower right
- the network is forced to learn individually what a cat looks like in all parts of the image

Can we do better?

A bit of history:

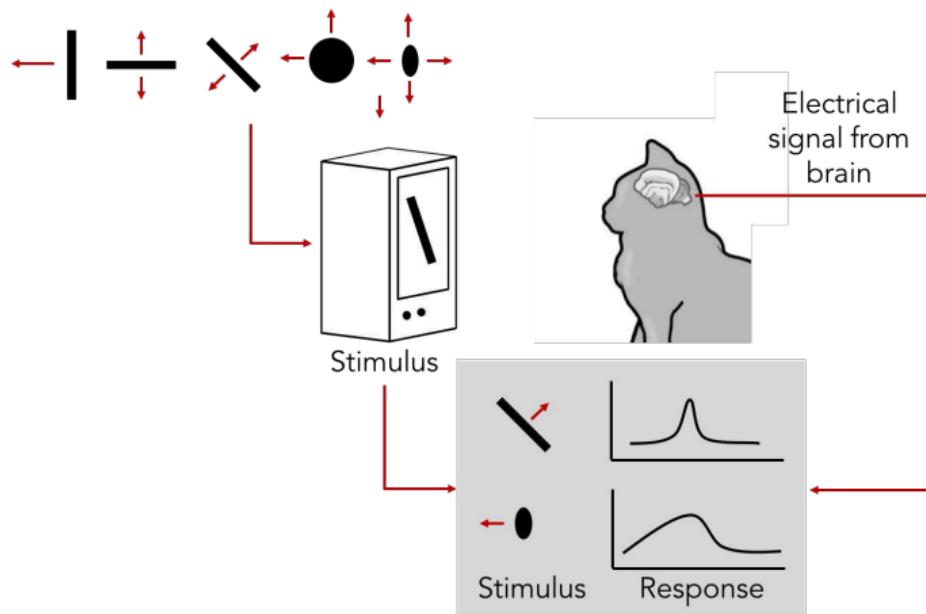
Hubel & Wiesel, 1959

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

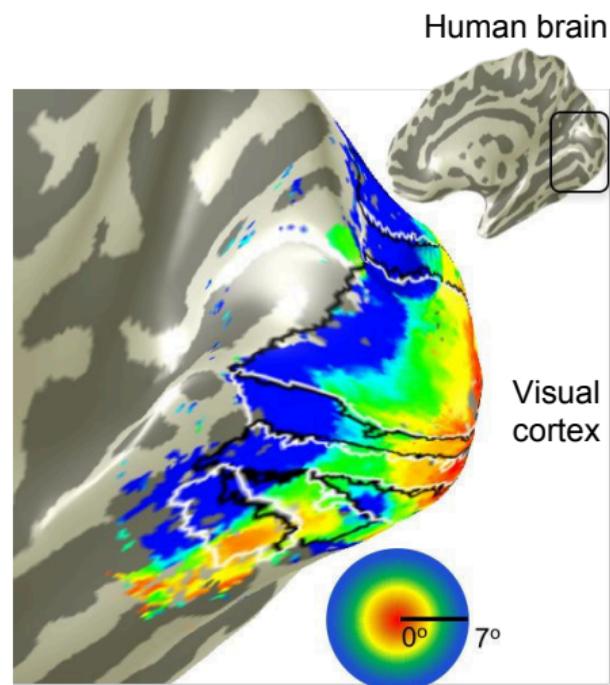
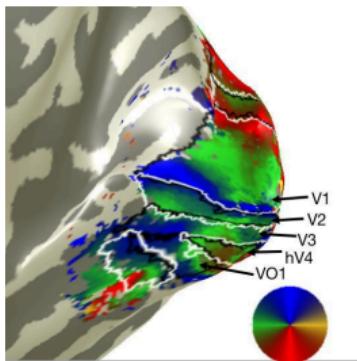
1968...



[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

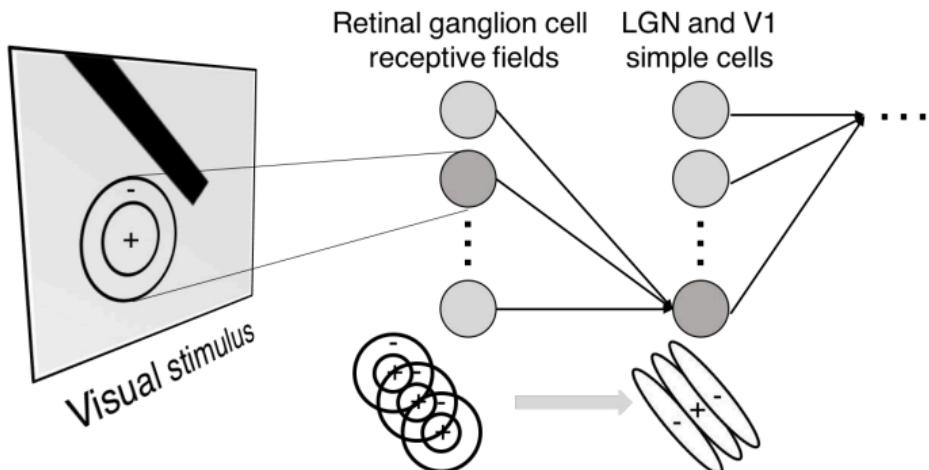
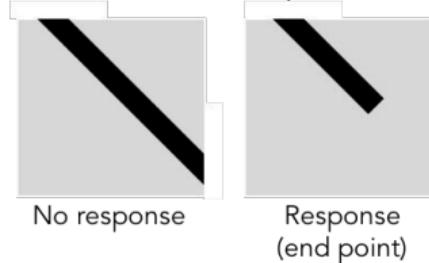


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

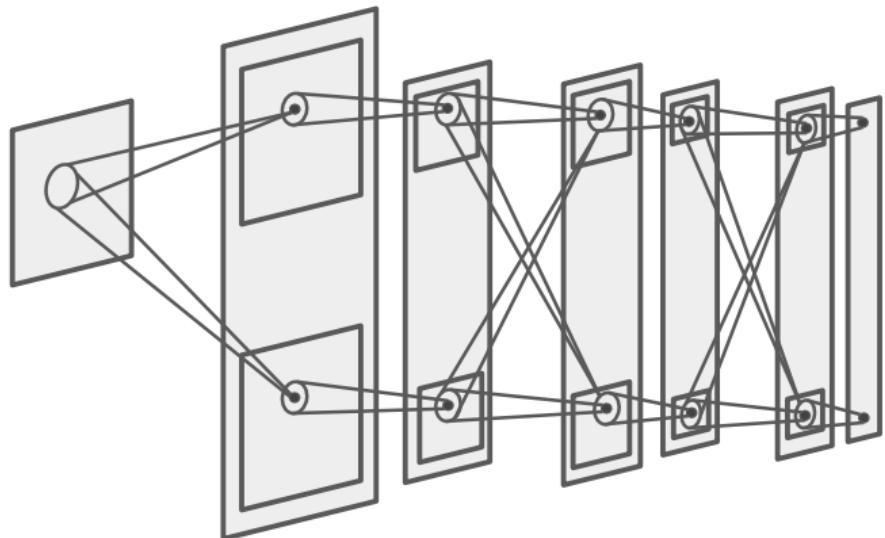
Hypercomplex cells:
response to movement with an end point



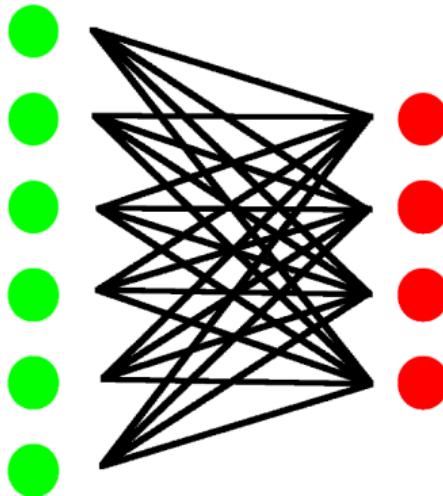
A bit of history:

Neocognitron [Fukushima 1980]

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

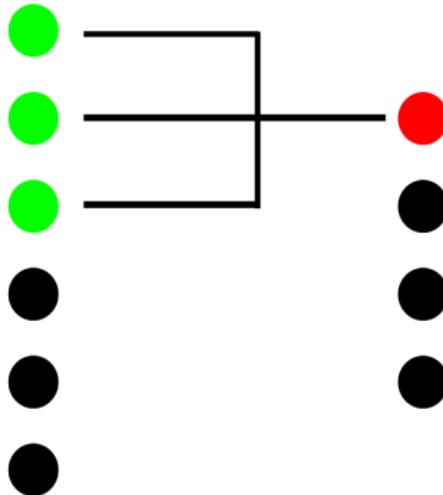


Fully connected neural network



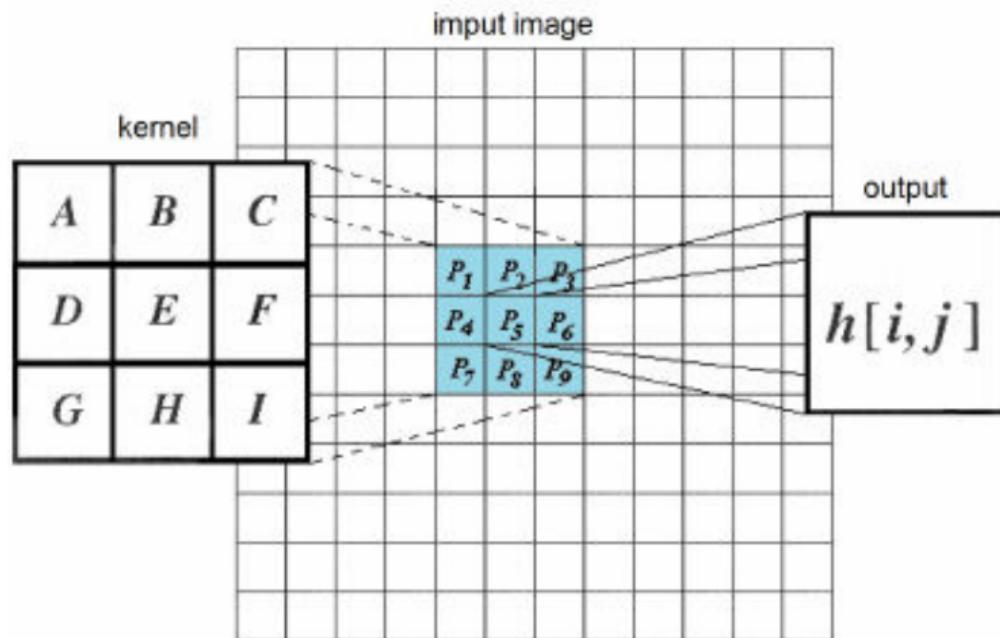
- $y_j = \sum_{i \in \text{input}} W_{ij}x_i + b_j$
- Everything connected
- Both local and non-local connections
- Does not care for spatial relationships
- Requires fixed input size
- Translation (even one pixel) matters a lot
- Very many parameters, difficult to train
- Rarely more than a few (last) layers in a network

Convolutional neural network



- Instead: Learn local filters (filter kernels)
- $y_j = \sum_{i=-k}^k W_k x_{j+k} + b_j$
- Only local connections
- Spatial relationships preserved
- Parameter sharing – massive reduction in number of parameters!
- Translation equivariant
shifted input → shifted output
- Can handle varying input sizes
- Widely used in image analysis
- A linear finite impulse response (FIR) filter

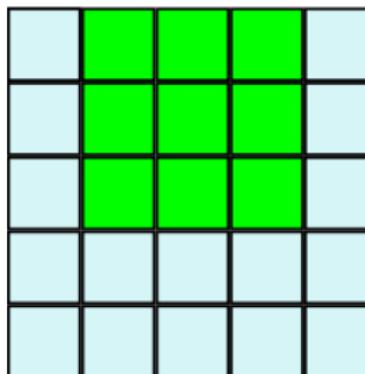
2d filtering = Cross correlation



$$h(i,j) = A*P_1 + B*P_2 + C*P_3 + D*P_4 + E*P_5 + F*P_6 + G*P_7 + H*P_8 + I*P_9$$

2d filtering

Layer 1



Layer 2

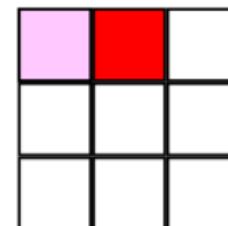


Figure: $3 \times 3 + 1 = 10$ weights for processing the whole image (of any size).
(Cf. $25 \times 9 + 1 = 226$ weights for a Fully connected layer for 5×5 images.)

Preserving the spatial shape of the data (here a 5×5 scalar (grayscale) image).

Convolution – Linear Shift-Invariant (LSI) system

Convolution vs. correlation – Mirror the filter in all axes (in 2D this is a 180 degree rotation).

Not logged in | Talk | Contribut

Article Talk Read Edit View history Search Wikipedia

Convolution

From Wikipedia, the free encyclopedia

For the usage in formal language theory, see [Convolution \(computer science\)](#). For other uses, see [Convolute](#).

In mathematics (in particular functional analysis) convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it. Some features of convolution are similar to cross-correlation: for real-valued functions, of a continuous or discrete variable, it differs from cross-correlation only in that either $f(x)$ or $g(x)$ is reflected about the y -axis; thus it is a cross-correlation of $f(x)$ and $g(-x)$, or $f(-x)$ and $g(x)$.^[note 1] For continuous functions, the cross-correlation operator is the adjoint of the convolution operator.

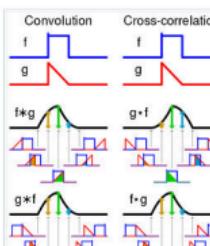
Convolution has applications that include probability, statistics, computer vision, natural language processing, image and signal

WIKIPEDIA The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools



Definition [edit]

The convolution of f and g is written $f * g$, using an asterisk or star. It is defined as the integral of the product of one reversed and shifted. As such, it is a particular kind of integral transform:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$

Properties [edit]

Algebraic properties [edit]

See also: [Convolution algebra](#)

The convolution defines a product on the linear space of integrable functions. This product satisfies the following algebraic properties given by convolution is a commutative associative algebra without identity (Strichartz 1994, §3.3). Other linear spaces are closed under the convolution, and so also form commutative associative algebras.

Commutativity

$$f * g = g * f$$

Proof: By definition

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$

Changing the variable of integration to $u = t - \tau$ and the result follows.

Associativity

$$f * (g * h) = (f * g) * h$$

Proof: This follows from using [Fubini's theorem](#) (i.e., double integrals can be evaluated as iterated integrals in either order).

Distributivity

$$f * (g + h) = (f * g) + (f * h)$$

Proof: This follows from linearity of the integral.

Associativity with scalar multiplication

$$a(f * g) = (af) * g$$

for any real (or complex) number a .

Convolutions

Convolutions are very common operations in Image processing.

Fast computation of large convolutions
can be done via the Fourier domain:

$$f * g = \mathcal{F}^{-1} \{ \mathcal{F}(f) \cdot \mathcal{F}(g) \}$$



Is a linear operation,
can be expressed as a matrix multiplication:

convmtx2

2-D convolution matrix

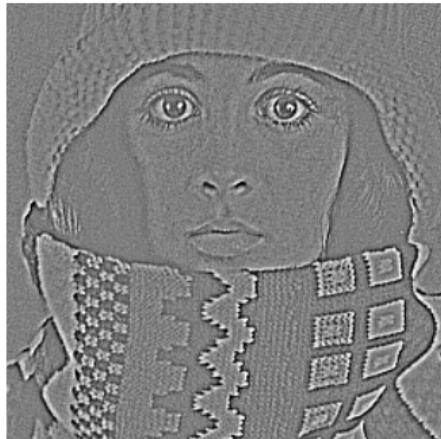
Syntax

```
T = convmtx2(H,m,n)
```

Description

`T = convmtx2(H,m,n)` returns the convolution matrix `T` for the matrix `H`. If `x` is an `m`-by-`n` matrix, then `reshape(T*x, size(H)+[m, n]-1)` is the same as `conv2(X, H)`.

Laplace filter



Laplace operator: $\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$

Sobel filter

Approximates the first derivatives: $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$



1	0	-1
2	0	-2
1	0	-1

S_x

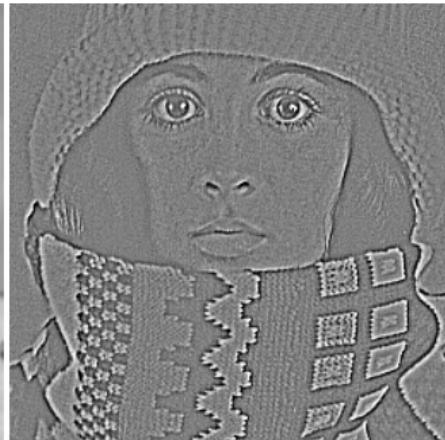
1	2	1
0	0	0
-1	-2	-1

S_y

Sharpening an image

$$\text{sharpened} = (1+\alpha) \cdot \text{original} - \alpha \cdot \text{smoothed}$$

$$\text{sharpened} = \text{original} + \alpha \cdot (\text{original} - \text{smoothed})$$



0	0	0
0	1	0
0	0	0

original

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

smoothed

-1	-1	-1
-1	8	-1
-1	-1	-1

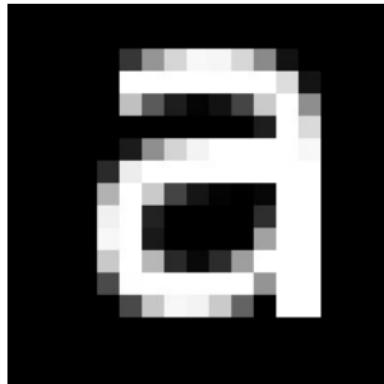
9 · diff.

Beyond smoothing and sharpening

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

An image of a piece of text.

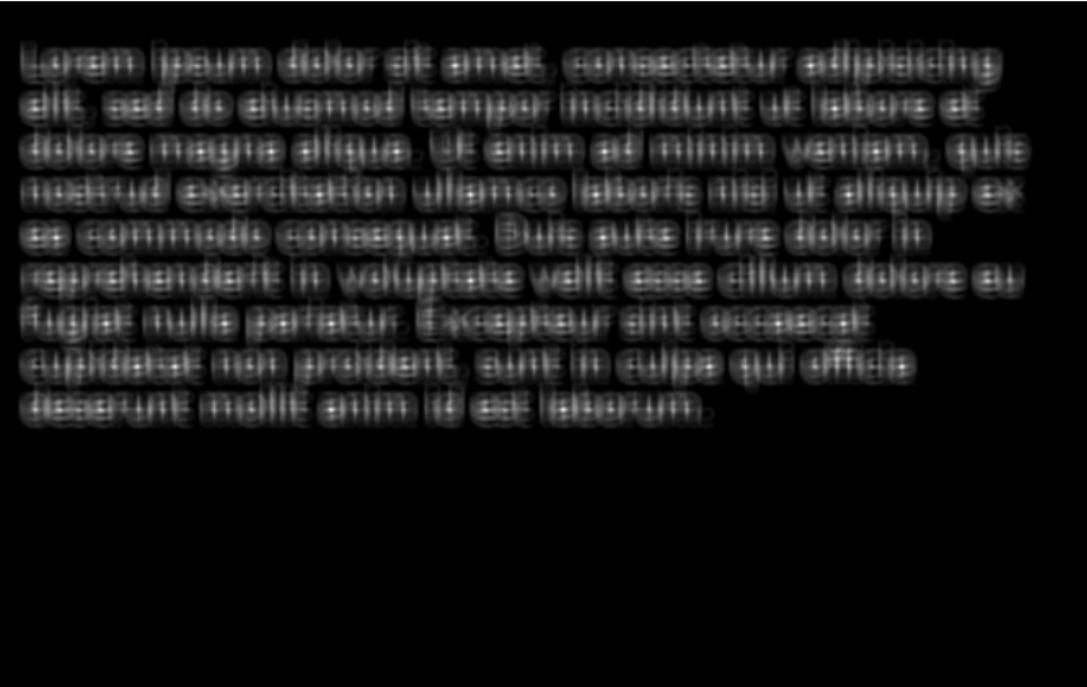
Beyond smoothing and sharpening



Filter kernel, image of letter "a".

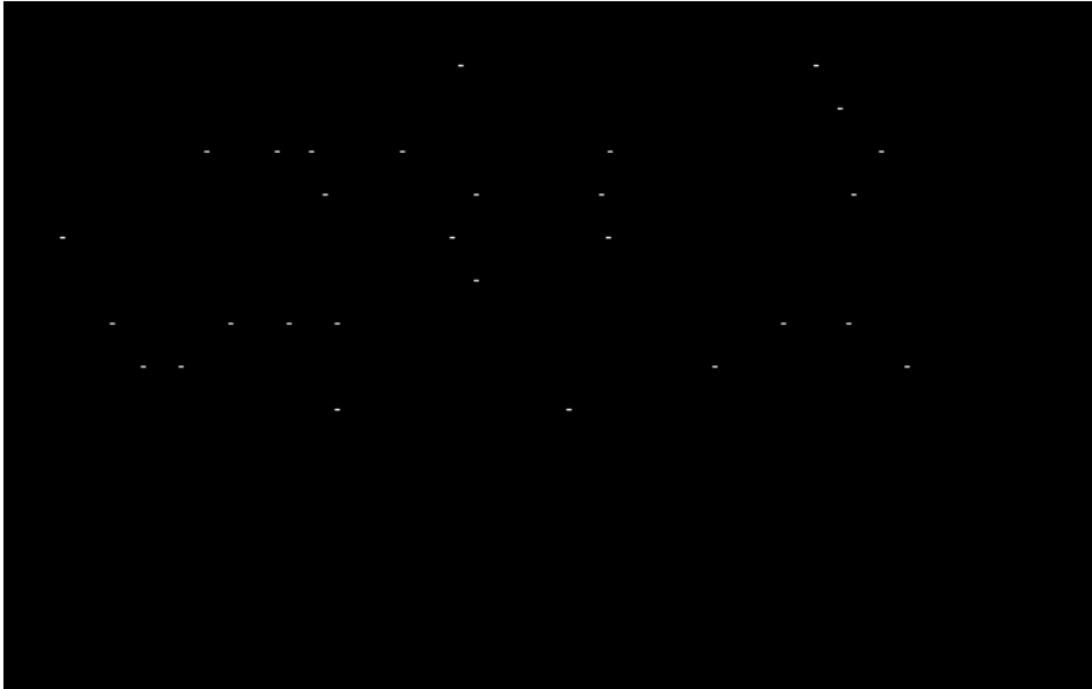
*What happens when we apply this kernel as a linear filter?
When is the output of this filter maximum/minimum?*

Beyond smoothing and sharpening



After linear filtering

Beyond smoothing and sharpening



Finding all pixels brighter than a manually selected threshold value

Beyond smoothing and sharpening

amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex
ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.

Detected instances of letter “a”.

Convolutional neural networks - born in 1989



Communicated by Dana Ballard

Backpropagation Applied to Handwritten Zip Code Recognition

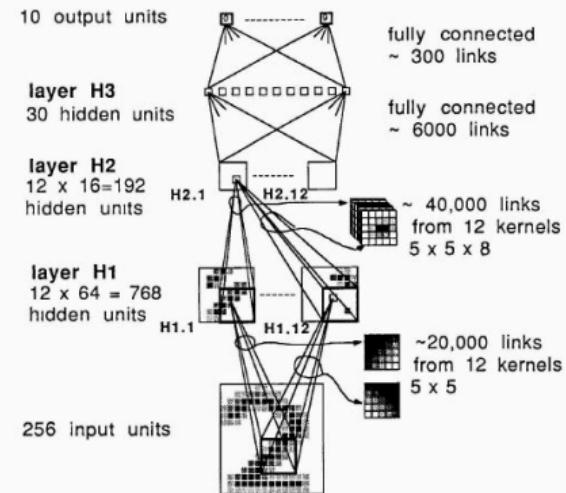
Y. LeCun
B. Boser
J. S. Denker
D. Henderson
R. E. Howard
W. Hubbard
L. D. Jackel
AT&T Bell Laboratories Holmdel, NJ 07733 USA

The ability of learning networks to generalize can be greatly enhanced by providing constraints from the task domain. This paper demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. This approach has been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service. A single network learns the entire recognition operation, going from the normalized image of the character to the final classification.

1 Introduction

Previous work performed on recognizing simple digit images (LeCun 1989) showed that good generalization on complex tasks can be obtained

548 LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel



Deep convolutional neural networks - born in 1998

PROC. OF THE IEEE, NOVEMBER 1998

1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. INTRODUCTION

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

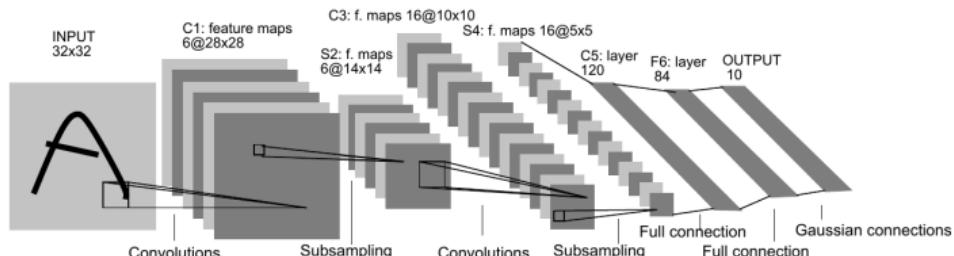


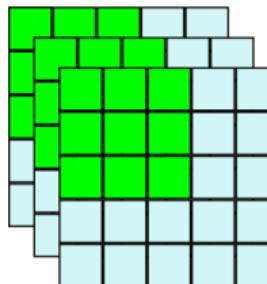
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Several channels of data

An RGB image has 3 (colour) channels

- Filter has equally many channels as **input**
- One (separate) filter for each **output** channel
- Filter coefficients are learned from data
- Local processing – efficient GPU implementations possible
- Hierarchical feature extraction

Layer 1



Layer 2

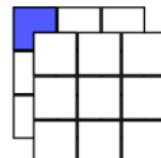
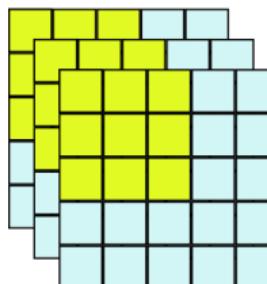
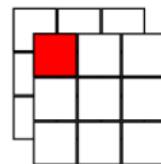
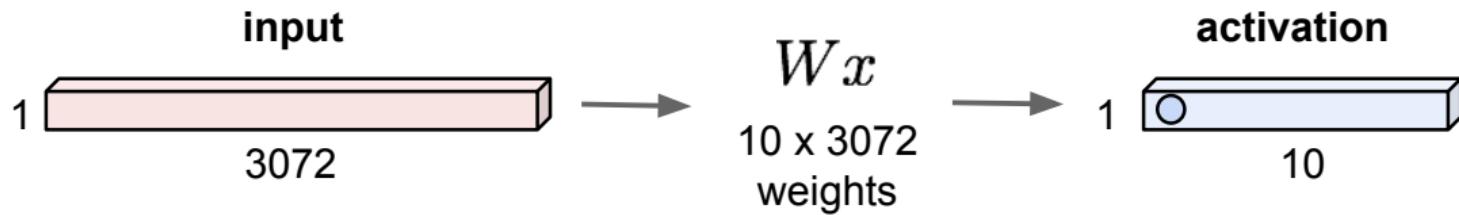


Figure: Example with 3 input channels
and 2 output channels

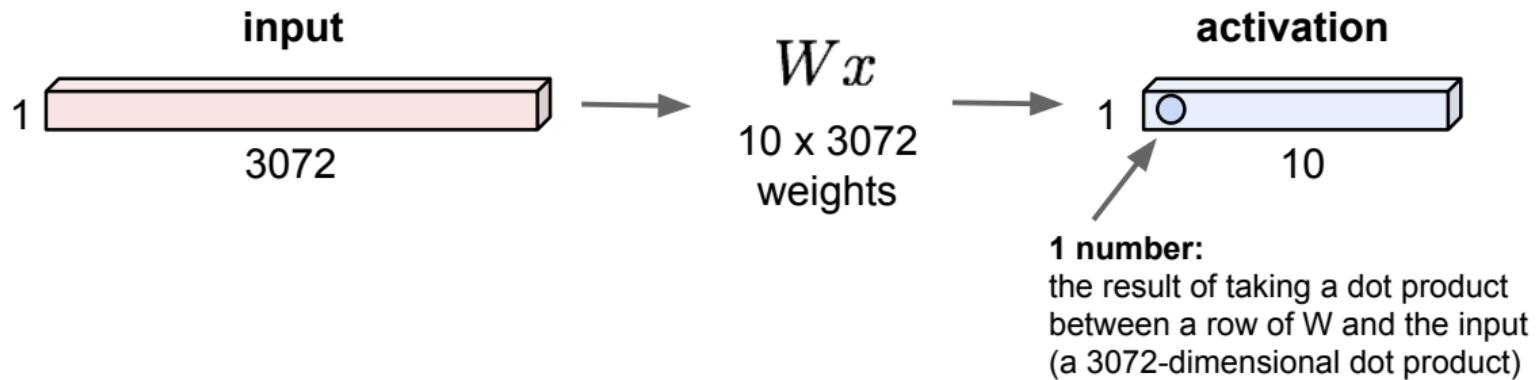
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



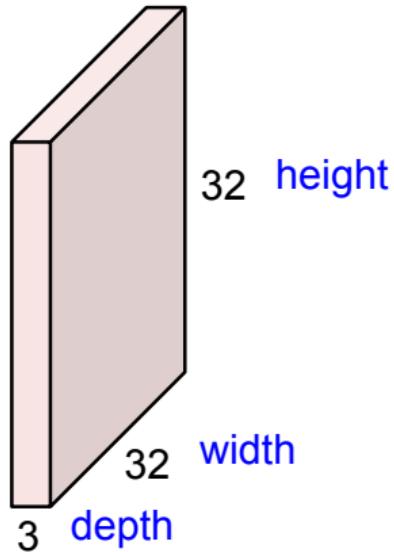
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



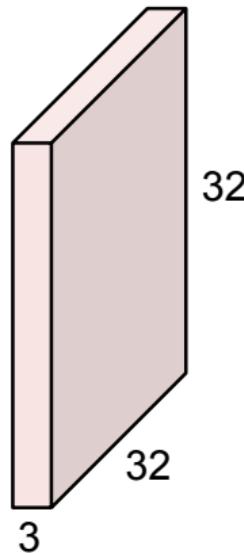
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



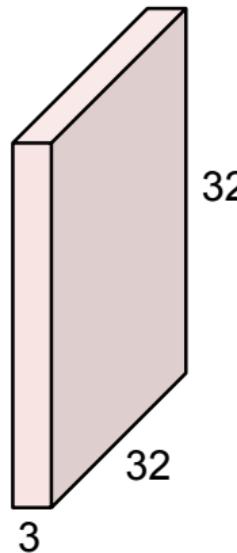
5x5x3 filter



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

Convolution Layer

32x32x3 image



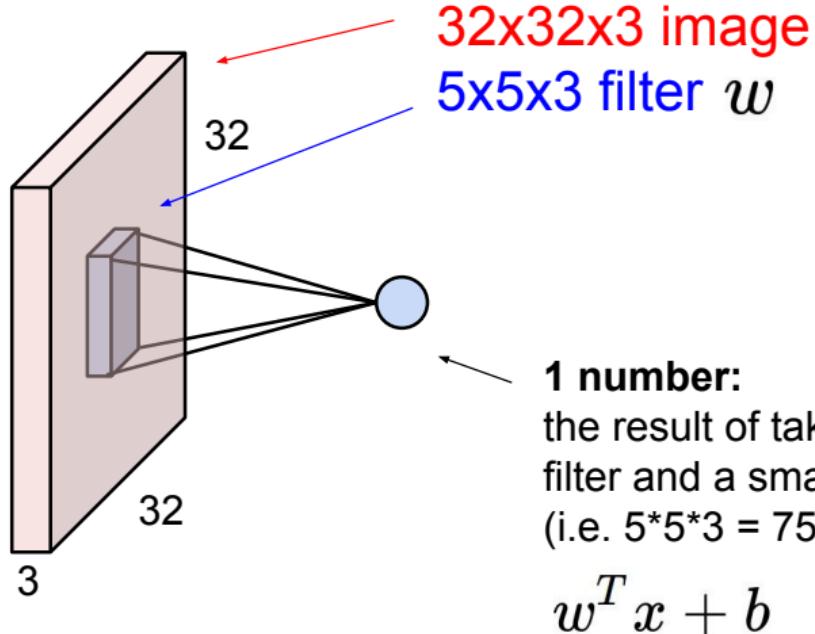
5x5x3 filter



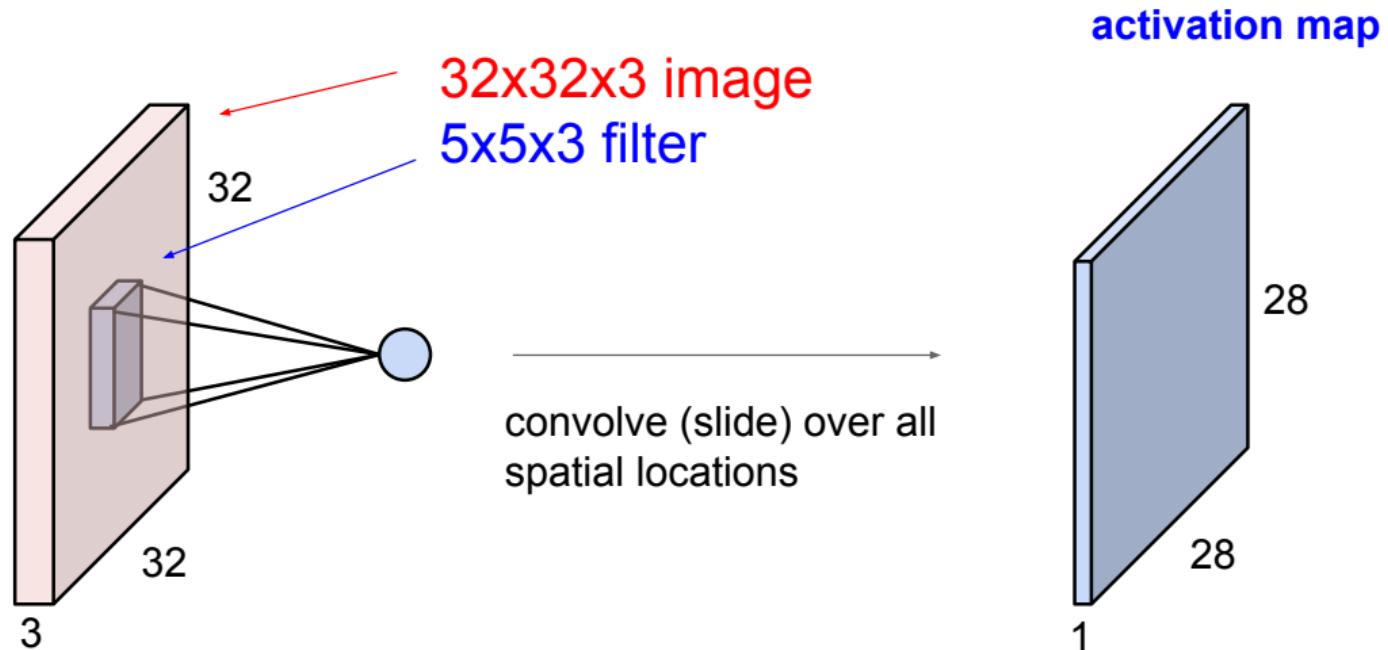
Filters always extend the full depth of the input volume

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

Convolution Layer

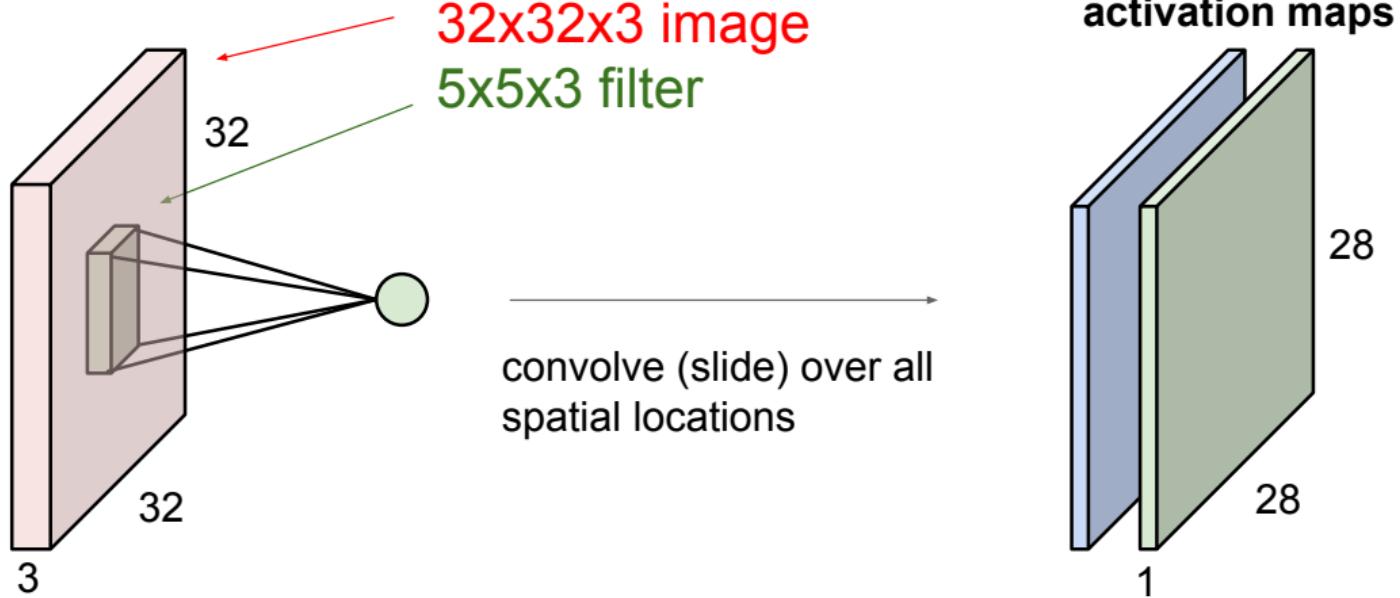


Convolution Layer

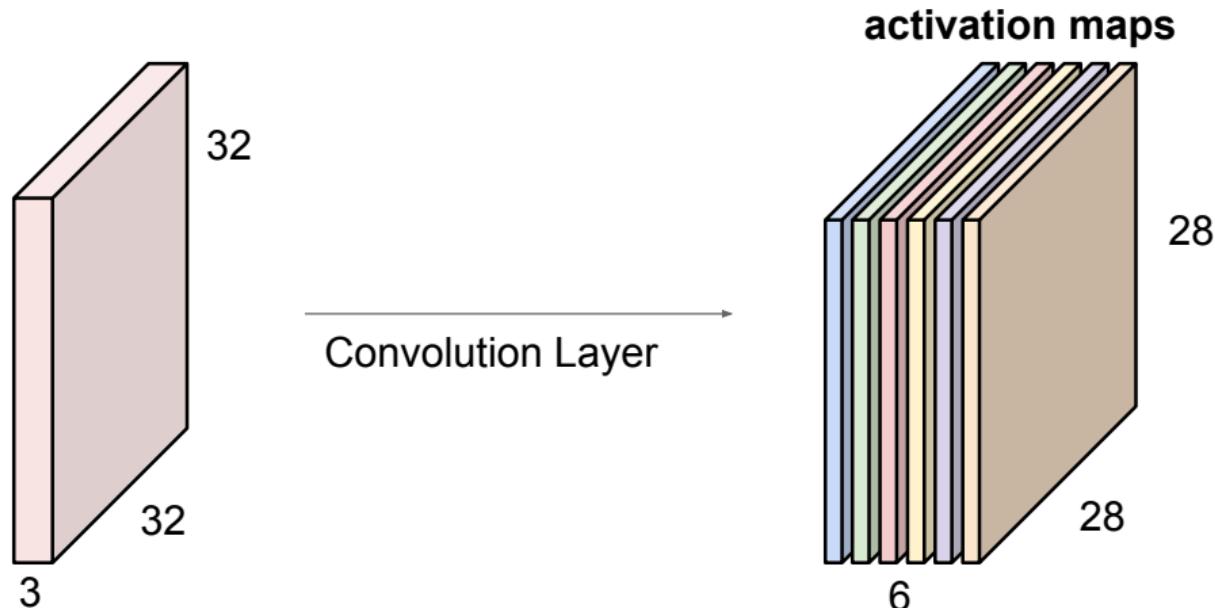


Convolution Layer

consider a second, green filter

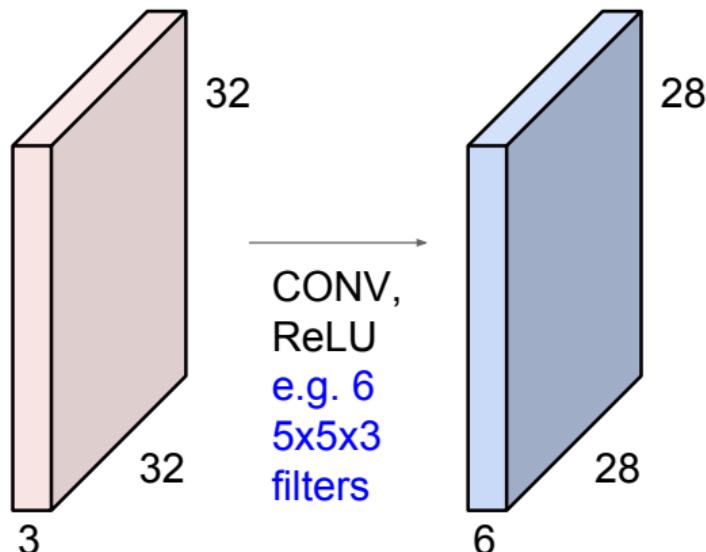


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

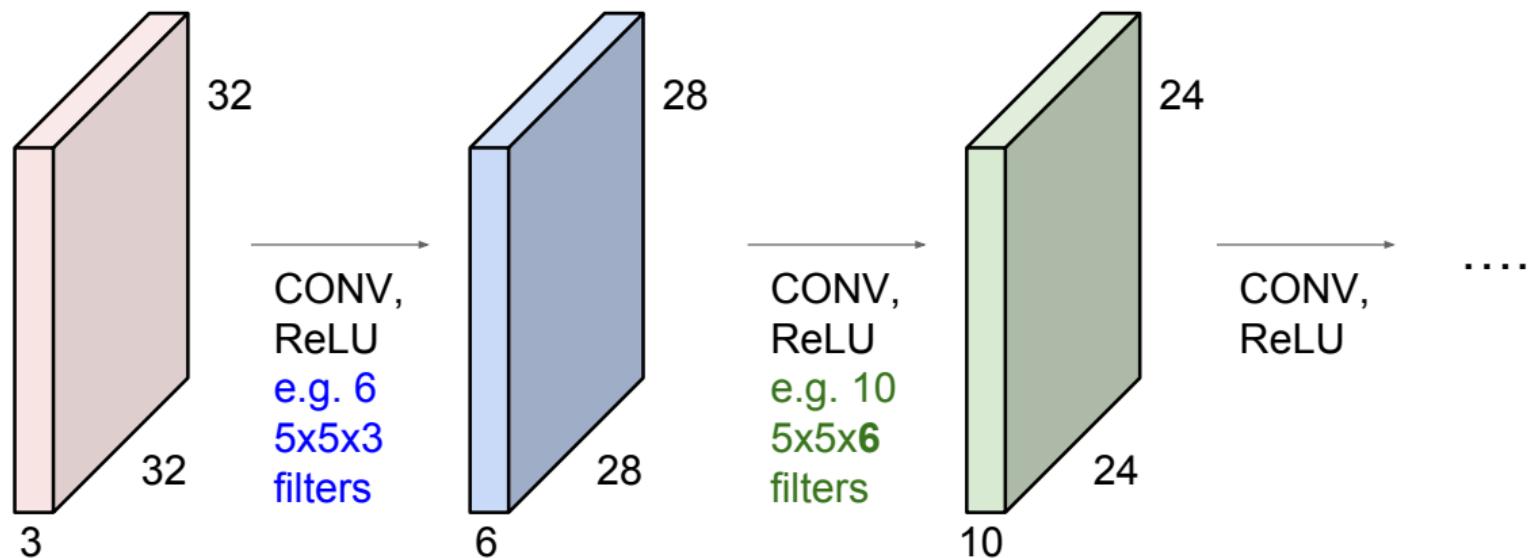


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

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



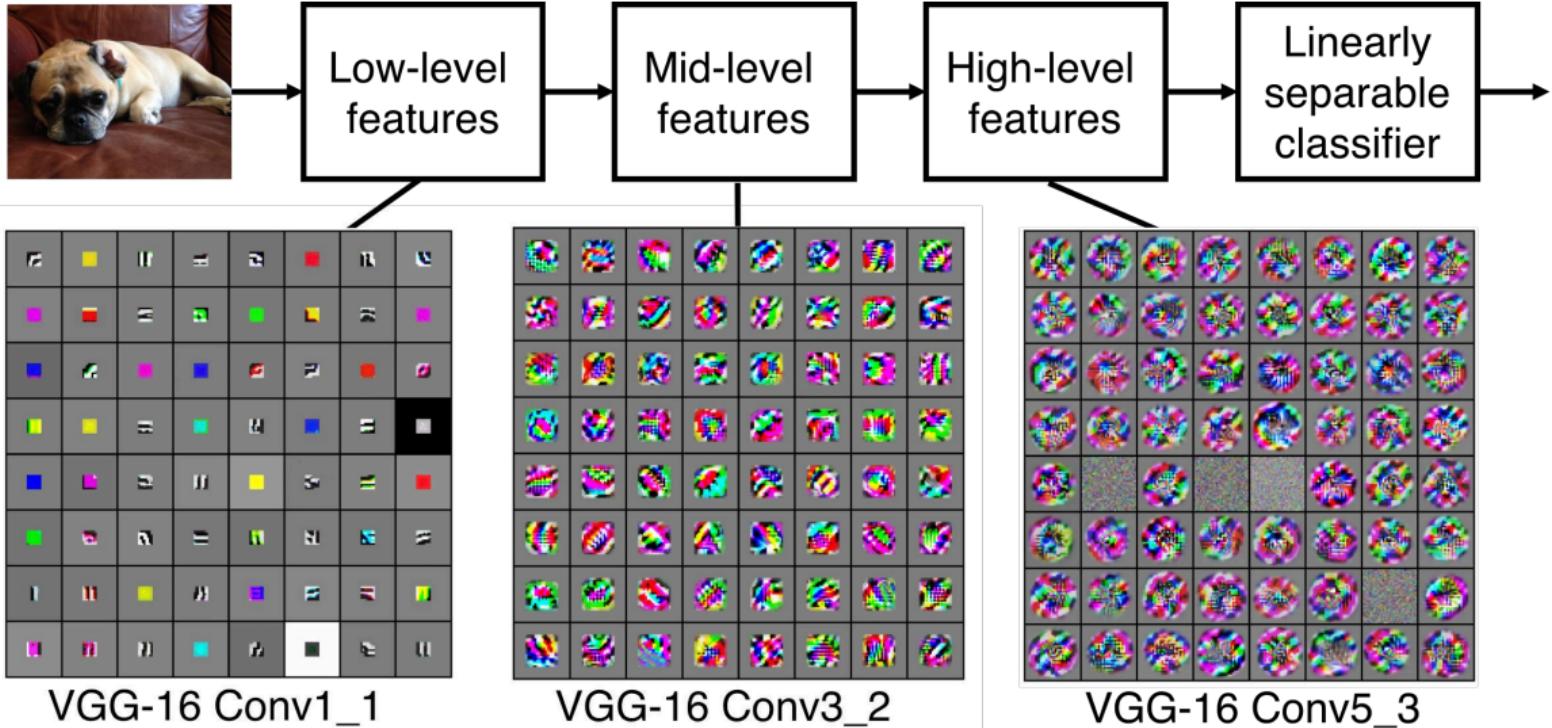
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



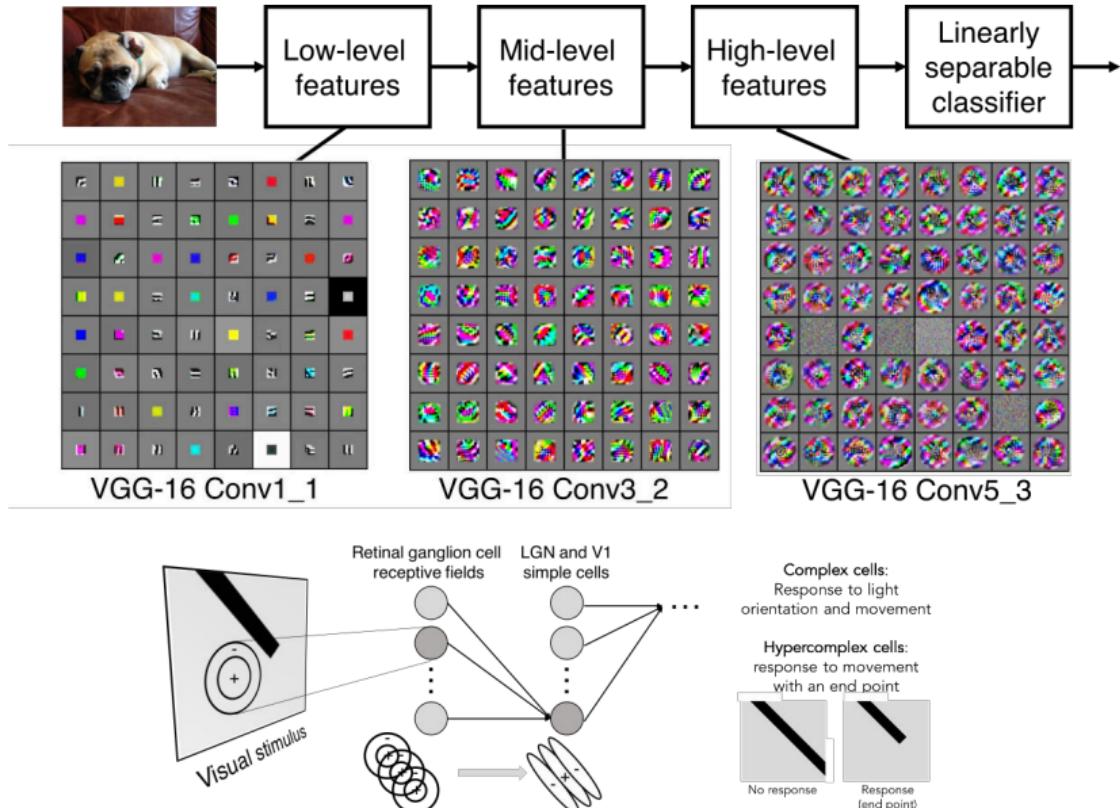
Preview

[Zeiler and Fergus 2013]

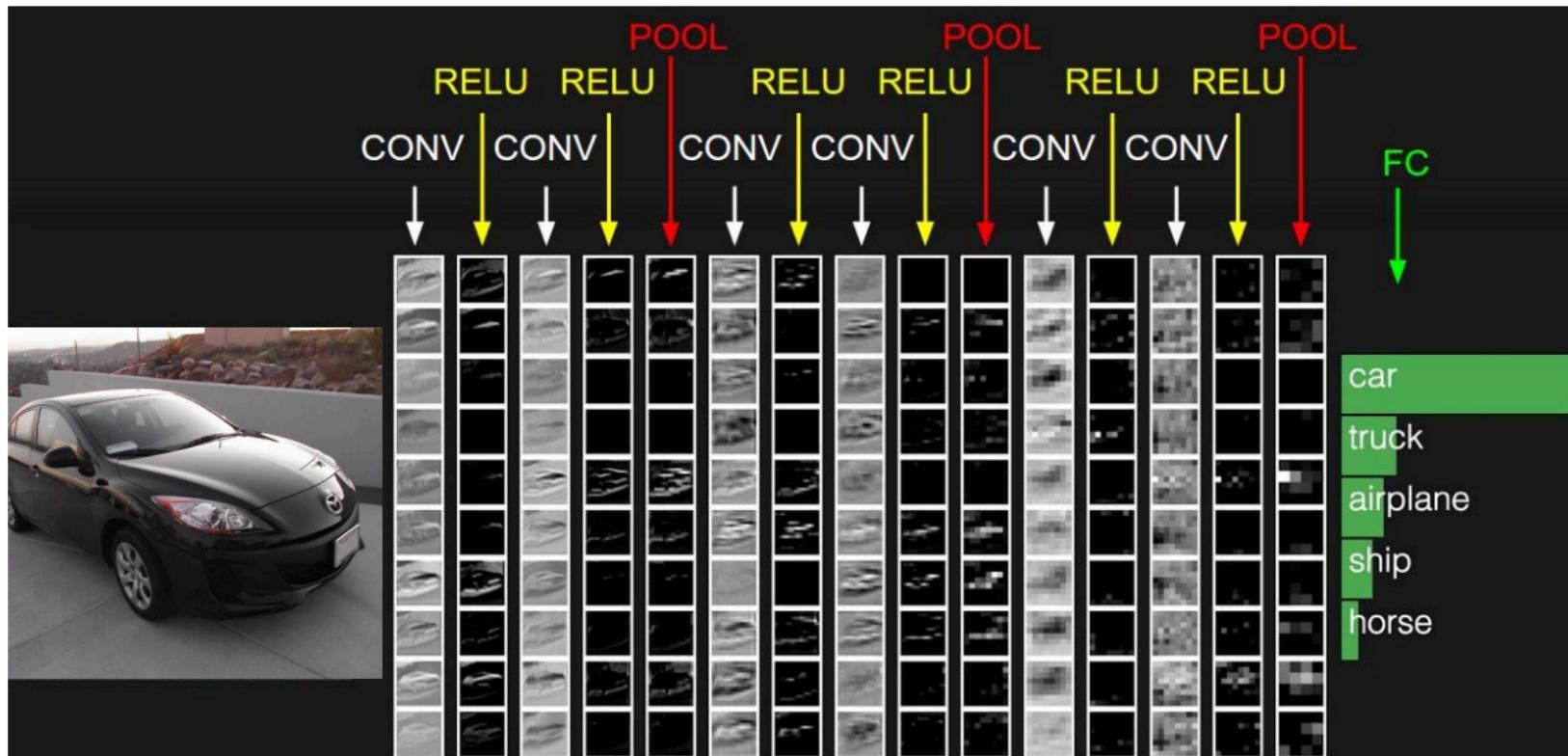
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



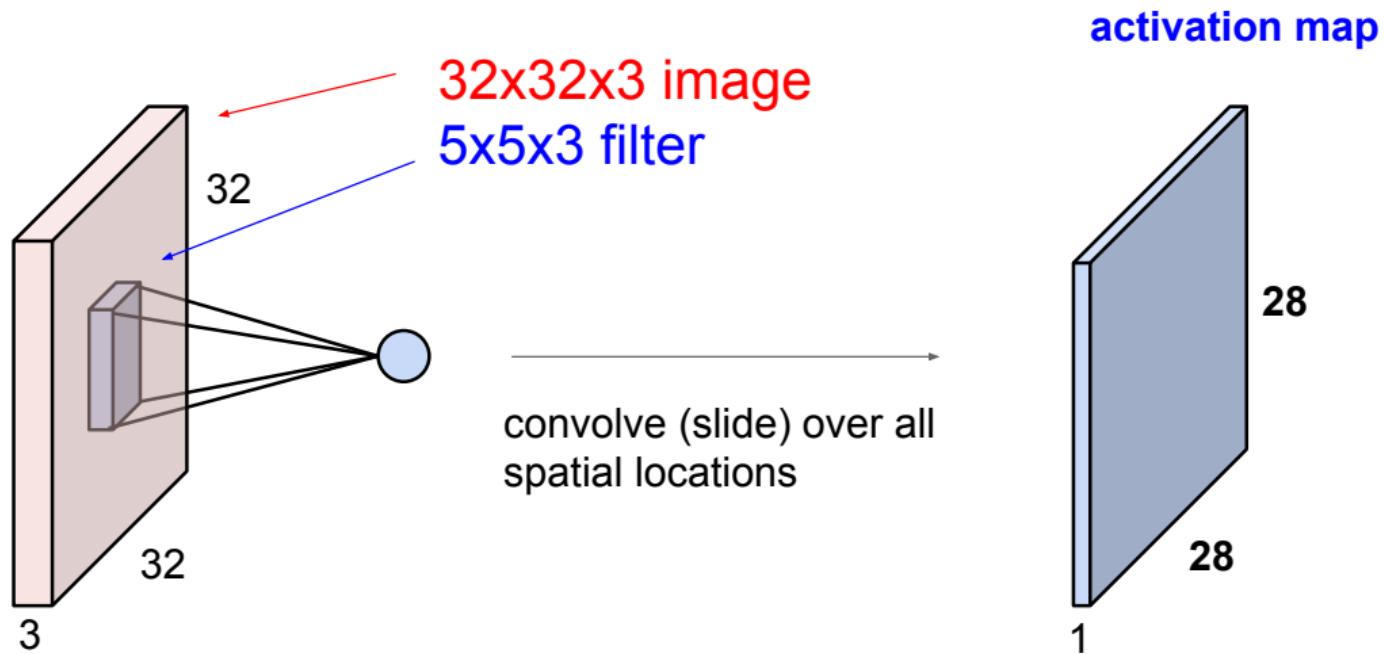
Preview



preview:

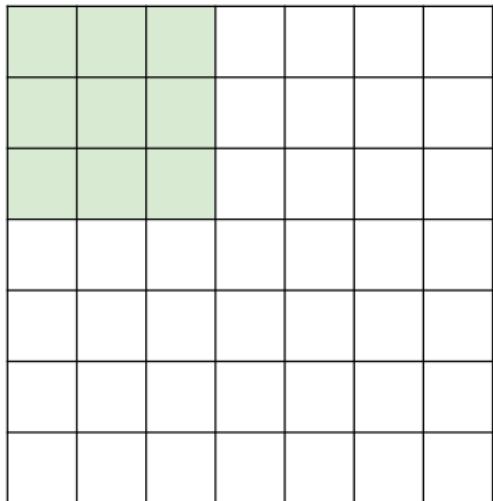


A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

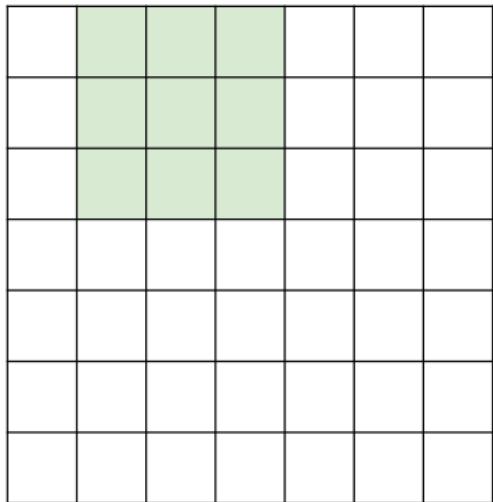


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

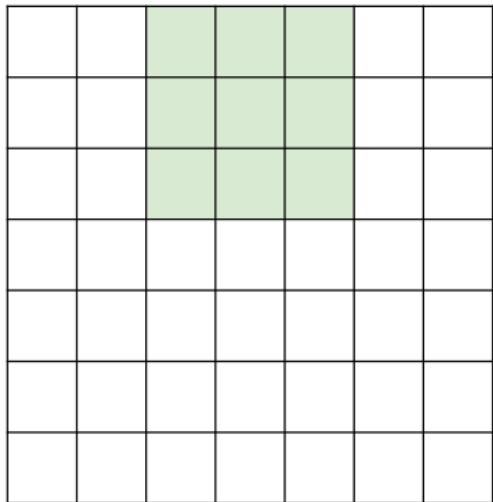


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

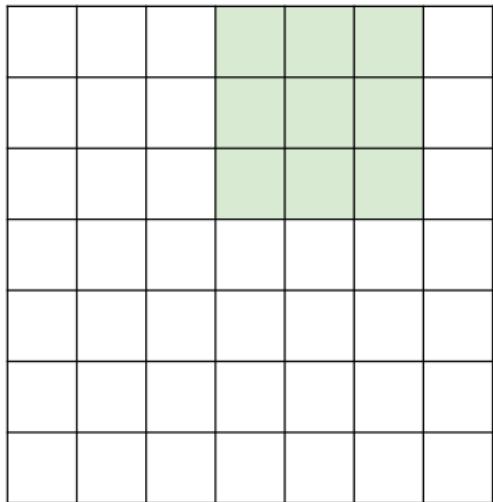


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

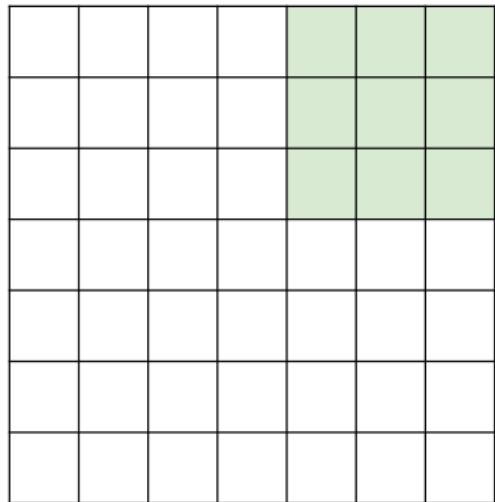


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7



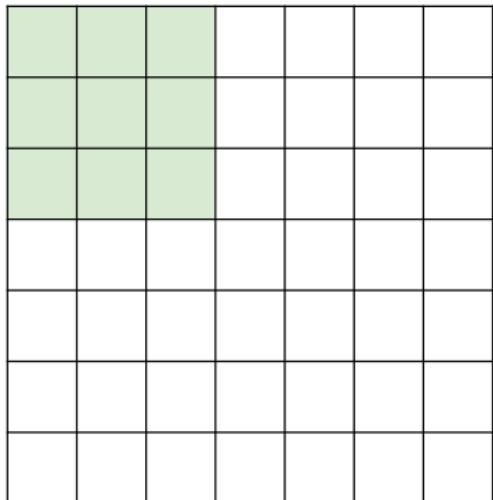
7x7 input (spatially)
assume 3x3 filter

=> **5x5 output**

7

A closer look at spatial dimensions:

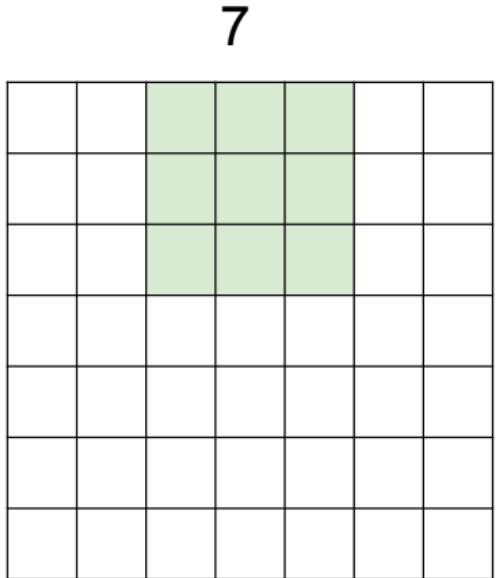
7



7

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

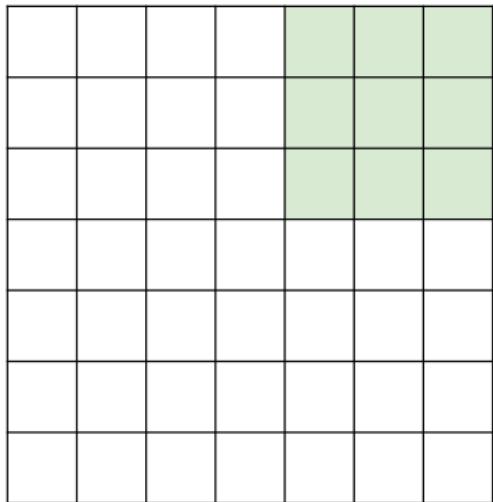
A closer look at spatial dimensions:



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!

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 FxF, and zero-padding with $(F-1)/2$. (will preserve size spatially)

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

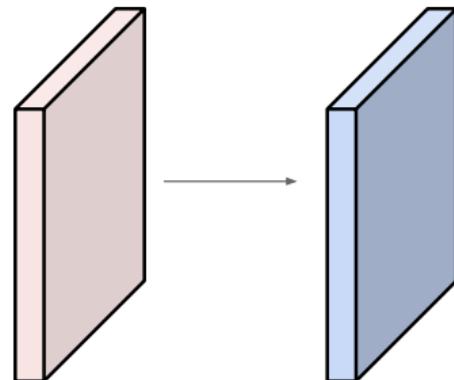
$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Examples time:

Input volume: **32x32x3**

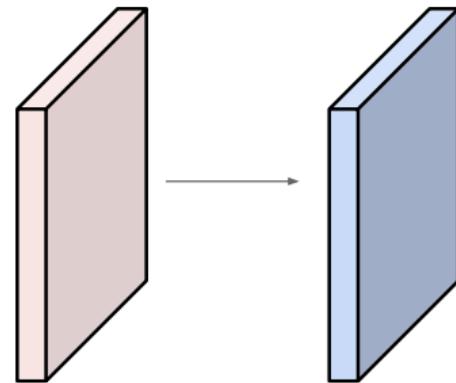
10 5x5 filters with stride 1, pad 2



Output volume size: ?

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride **1**, pad **2**

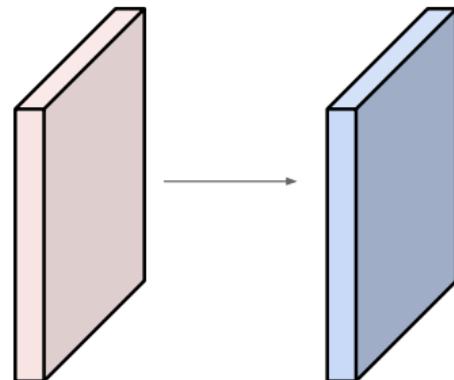


Output volume size:
 $(32+2*2-5)/1+1 = 32$ spatially, so
32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

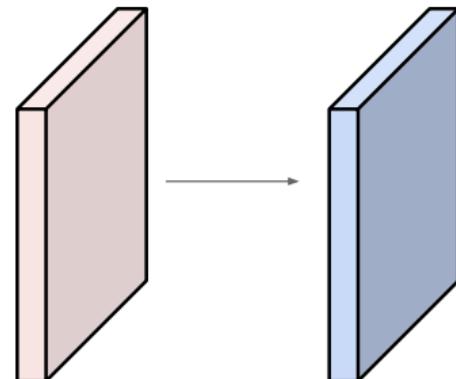


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

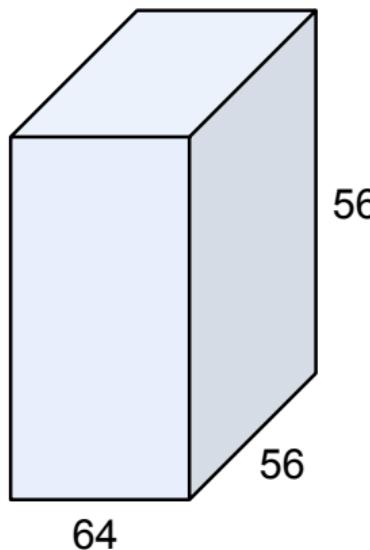
10 **5x5** filters with stride 1, pad 2



Number of parameters in this layer?

each filter has $5*5*3 + 1 = 76$ params (+1 for bias)
=> $76*10 = 760$

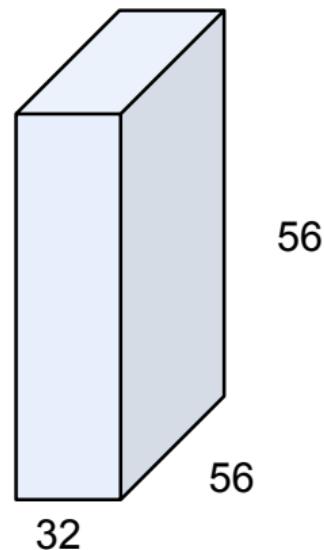
(btw, 1x1 convolution layers make perfect sense)



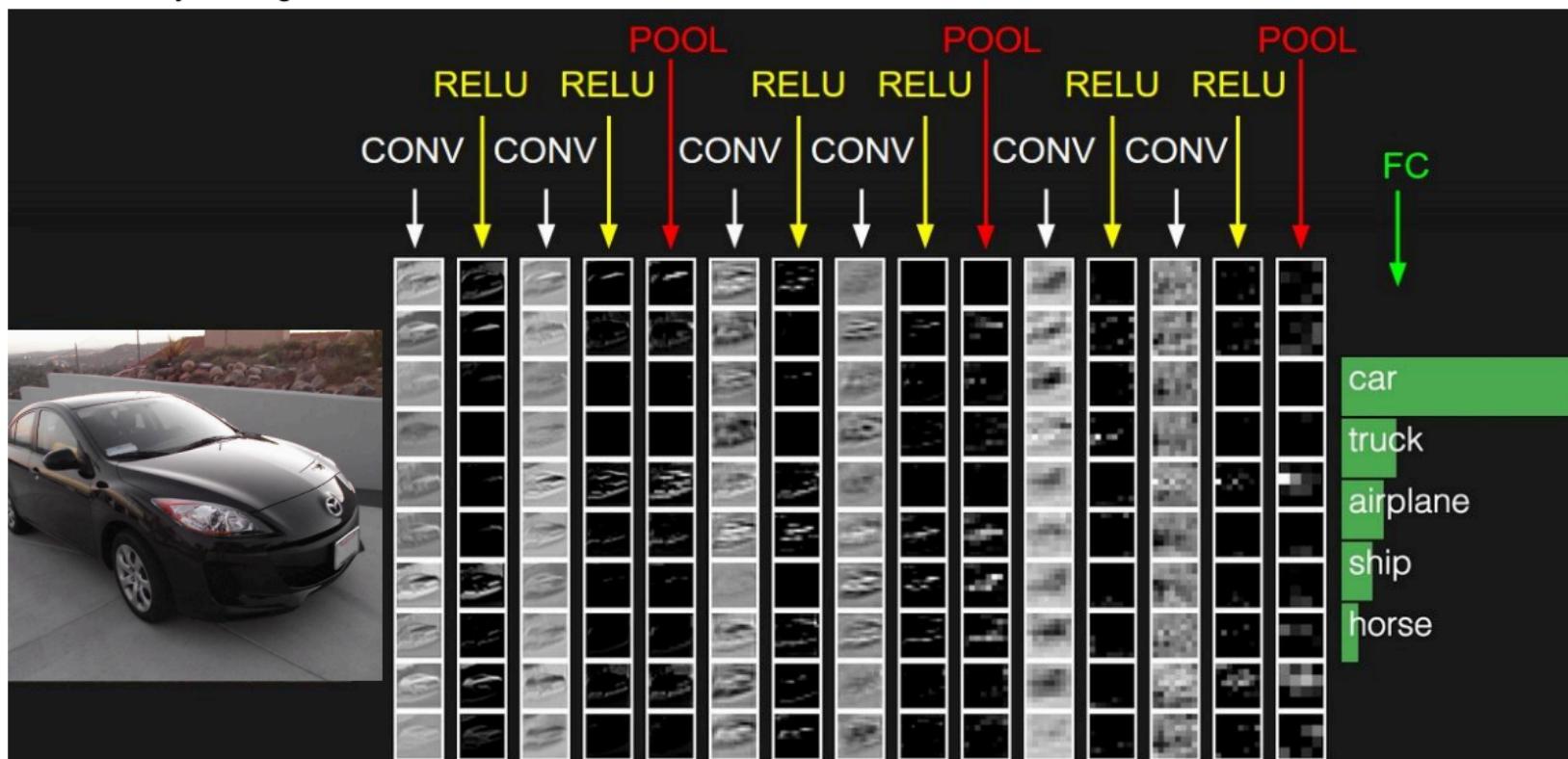
1x1 CONV
with 32 filters

→

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

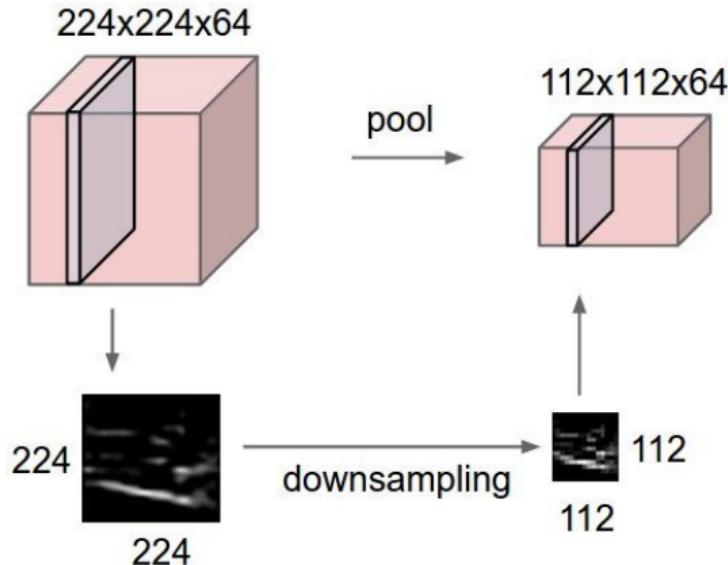


two more layers to go: POOL/FC

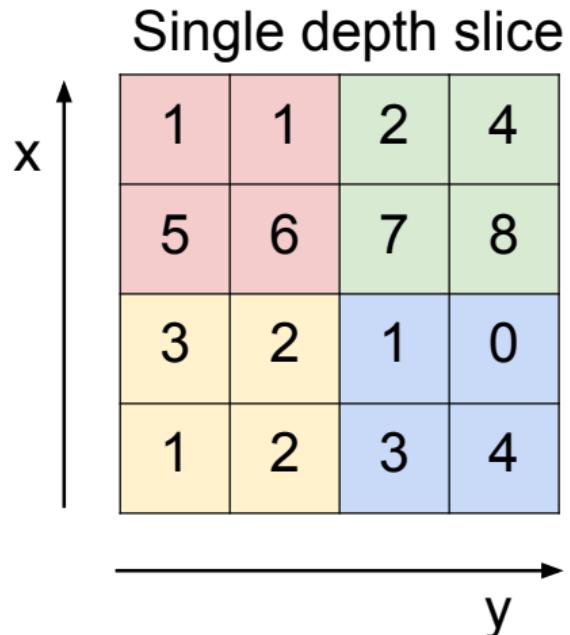


Pooling layer

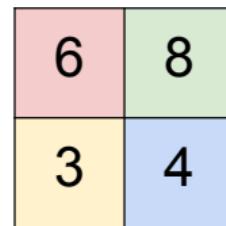
- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING



max pool with 2x2 filters
and stride 2



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.

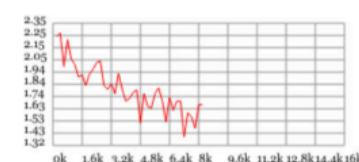
This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

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](#).

Training Stats

<input type="button" value="pause"/>	Loss:
Forward time per example: 8ms	
Backprop time per example: 14ms	
Classification loss: 1.78091	
L2 Weight decay loss: 0.00394	
Training accuracy: 0.37	
Validation accuracy: 0.34	
Examples seen: 8369	
Learning rate: 0.01	<input type="button" value="change"/>
Momentum: 0.9	<input type="button" value="change"/>
Batch size: 4	<input type="button" value="change"/>



Network Visualization

input (32x32x3)
max activation: 0.5, min: -0.43726
max gradient: 0.13763, min: -0.16955

Activations:

conv (32x32x16)
filter size 5x5x3, stride 1
max activation: 1.59913, min: -2.27393
max gradient: 0.09832, min: -0.08893
parameters: 16x5x5x3+16 = 1216

Activations:

Activation Gradients:

Weights:

Weight Gradients:

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

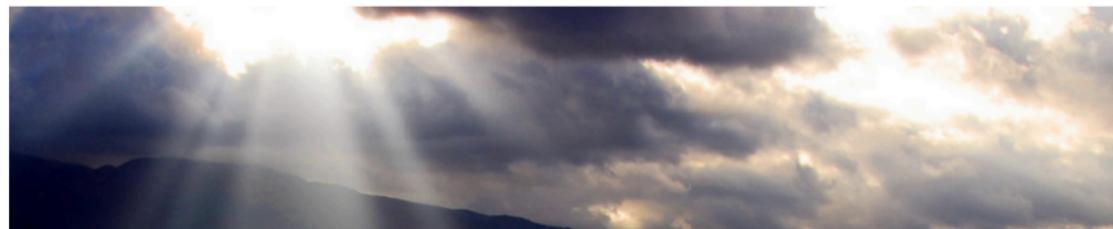
More about convolutions



The screenshot shows a header with a large 'M' icon, 'Sign in', and 'Get started' button. Below the header is a navigation bar with links: 'Towards Data Science', 'DATA SCIENCE', 'MACHINE LEARNING', 'PROGRAMMING', 'VISUALIZATION', 'AI', 'DATA JOURNALISM', and 'CONTRIBUTE'. The main content area features a large title 'A Comprehensive Introduction to Different Types of Convolutions in Deep Learning' and a subtitle 'Towards intuitive understanding of convolutions through visualizations'. Below the title is a profile picture of Kunlun Bai, a 'Follow' button, and the text 'Feb 11 · 31 min read'. A large, dramatic photograph of clouds with sunlight streaming through is at the bottom.



Kunlun Bai [Follow](#)
Feb 11 · 31 min read



<https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215>

A few concepts to summarize lecture 4

Fully connected layer: Each output is connected to every input. $(\text{Input size plus one}) \times \text{output size}$ number of parameters.

Convolutional layer: Only local connections and weight sharing; sliding filter over the image. $(\text{Filter size plus one}) \times \text{number of filters (depth)}$ number of parameters.

Pooling layer: Aggregation of local information plus downsampling (striding). Most often max-pooling, but sometimes average pooling. No parameters.

Striding: When only computing output for every n-th pixel/voxel, reducing the spatial size of the activation. For example strided convolution.

Deep learning: End-to-end learning, where the machine learning methods “learns” features.

Shallow learning: Machine learning which relies on engineered features instead of the raw data.

Further reads/links

- Stanford CS231n deep learning course by Fei Fei's group, <http://cs231n.stanford.edu/syllabus.html>, 2016 version (skip to 2nd lecture, w. A. Karpathy)
<https://www.youtube.com/watch?v=g-PvXUjD6qg&list=PLlJy-eBtNFt6EuMxFYRiNRS07MCWN5UIA>
2017 version <https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv>
- 3blue1brown https://www.youtube.com/playlist?list=PLZHQObOWTQDNu6R1_67000Dx_ZCJB-3pi
- Machine learning by Andrew Ng (Coursera) <https://www.youtube.com/playlist?list=PLZ9qNFMHZ-A4rycgrg0Yma6zxF4BZGG>
<http://deeplearning.stanford.edu/tutorial/>
- Deep learning and reinforcement learning summer school http://videolectures.net/DLRLsummerschool2018_toronto/
- Ian Goodfellow's book on deep learning <http://www.deeplearningbook.org/>
- Stat212b: Topics Course on Deep Learning <http://joanbruna.github.io/stat212b/>
- Yann LeCun's "Gradient-based learning applied to document recognition"
<http://ieeexplore.ieee.org/document/726791/?arnumber=726791>
- An overview of gradient descent optimization algorithms <http://ruder.io/optimizing-gradient-descent/>
- Get going in MATLAB (if you're more confident there than in Python)
<https://mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html>
- WILDML <http://www.wildml.com/>
- Deep Learning Glossary <http://www.wildml.com/deep-learning-glossary/>
- Colah's blog <http://colah.github.io/>
- <https://icml.cc/virtual/2020/events/Tutorial>,
https://icml.cc/Conferences/2019/ScheduleMultitrack?session=&event_type=Tutorial&,
<https://icml.cc/Conferences/2018/Schedule?type=Tutorial>
- <https://arxiv.org> & <http://www.arxiv-sanity.com/> & <https://paperswithcode.com/>
- The annual report of <https://aiindex.org>
- And many many more ... (Google is your friend :-)