M2177.003100
Deep Learning

[6: Convolutional Neural Nets (Part 1)]

Electrical and Computer Engineering
Seoul National University

(last compiled at 16:05:00 on 2020/10/04)

Outline

References

- *Deep Learning* by Goodfellow, Bengio and Courville Link
  - Chapter 9
- online resources:
  - *Deep Learning Specialization (coursera)* Link
  - *Stanford CS231n: CNN for Visual Recognition* Link
- note:
  - you should open this file in Adobe Acrobat to see animated images (other types of pdf readers will not work)

Outline

Convolutional (neural) networks (CNNs)

- simply neural nets that use **convolution** in their layers
  - in place of general matrix multiplication
- employ a mathematical operation called *convolution*
  - convolution[1]: a special kind of linear operation
- tremendously successful in practical applications
  - especially for data with a known, grid-like topology

  e.g. time series (1D grid), image (2D grid of pixels), video (3D grid)
- explicit assumption: inputs have grid topology
  - allow us to encode certain properties into architecture
  - make forward function more efficient to implement
  - significantly reduce the amount of parameters

[1] usually does not correspond precisely to the definition of convolution as used in other fields

- an example of neuroscientific principles influencing deep learning
  - *e.g.* human vision system:
    - simple cells → complex cells → hyper-complex cells

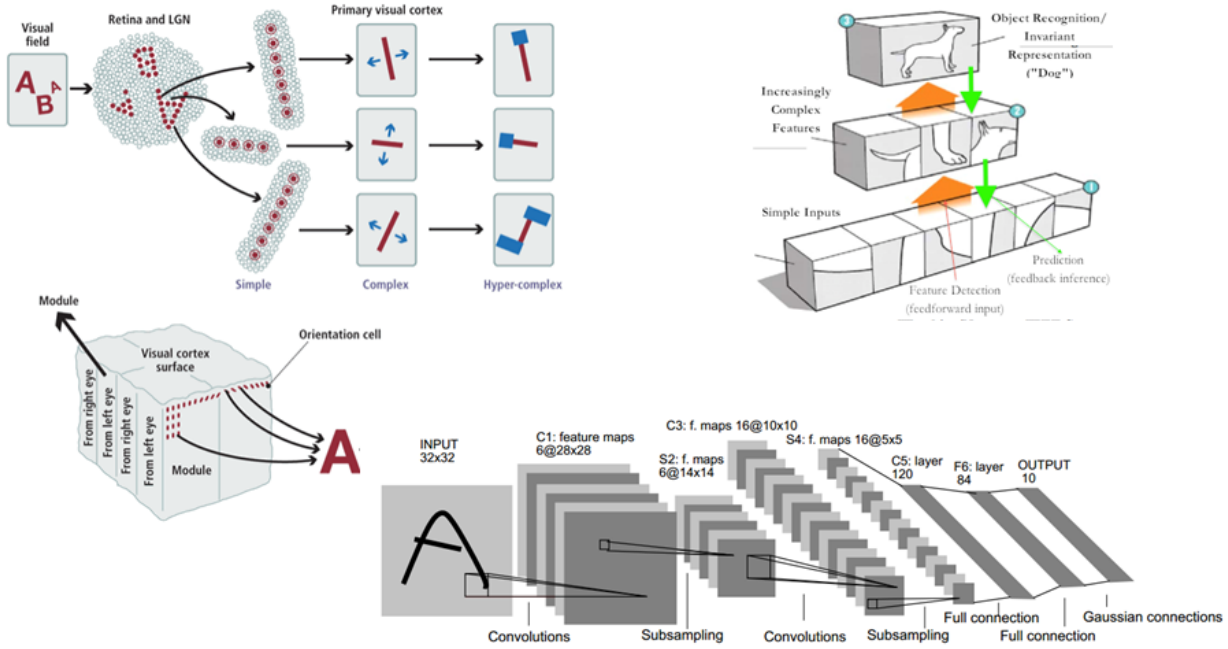


image source: https://www.vision-systems.com/boards-software/article/16749506/system-stimulus, https://i.pinimg.com/474x/5c/50/97/5c50979ab722d68a0b8fec2a98ba1f5c--deep-learning.jpg, Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998

Hubel and Wiesel

- neurophysiologists David Hubel and Torsten Wiesel
  - determined many facts about the mammalian vision system
- their findings with greatest influence on deep learning models:
  - based on recording the activity of individual neurons in cats

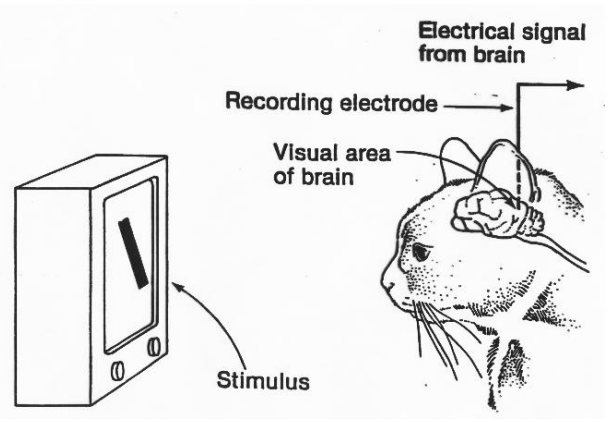
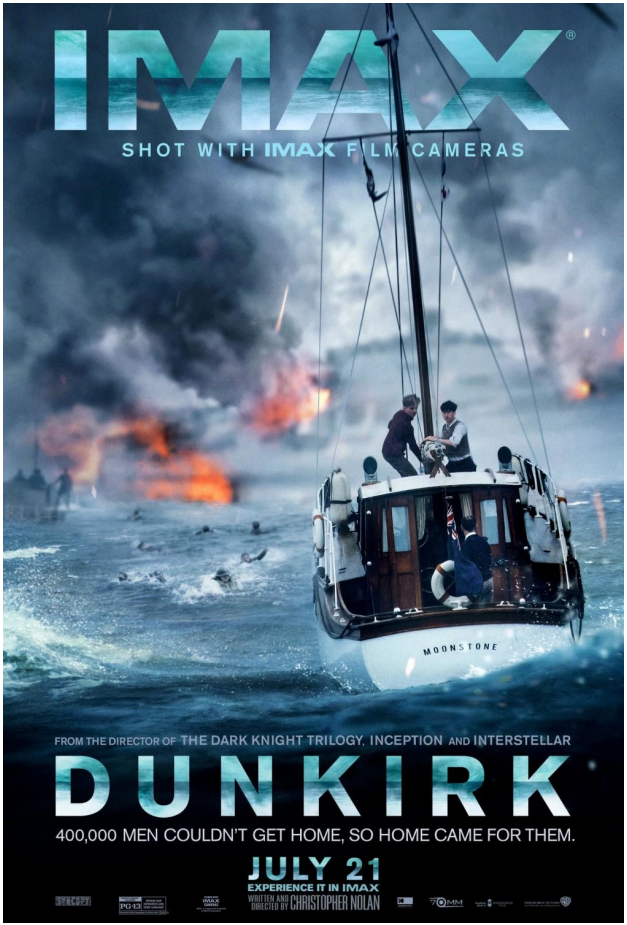
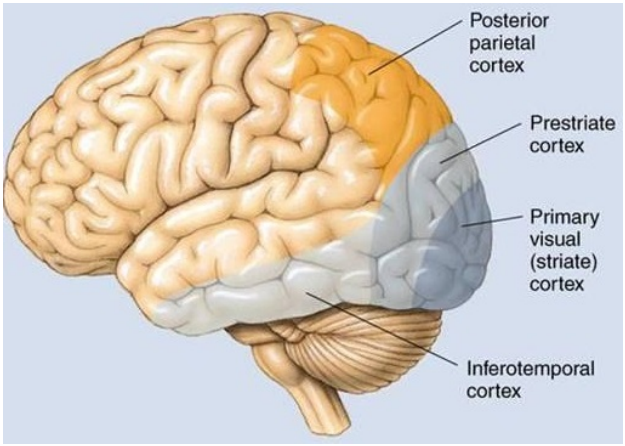


image sources: https://braintour.harvard.edu/archives/portfolio-items/hubel-and-wiesel; David Hubel and Torsten Wiesel (1959)

Human vision





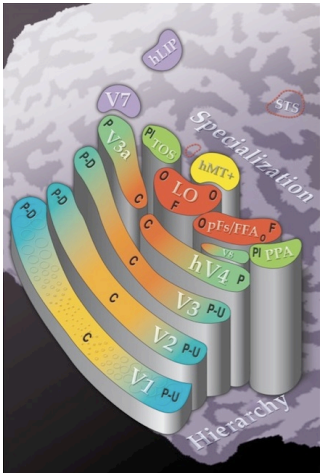


image sources: Warner Bros. Pictures (2017); https://images.app.goo.gl/1z86sq2CA4tyCAVP8; Morawetz, C., 2008. *The allocation of attentional resources across the visual field: Impact of eccentricity and perceptual load.*

Computer vision

- methods to acquire/process/analyze/understand
  - images and high-dimensional data from the real world
  - in order to produce numerical or symbolic information

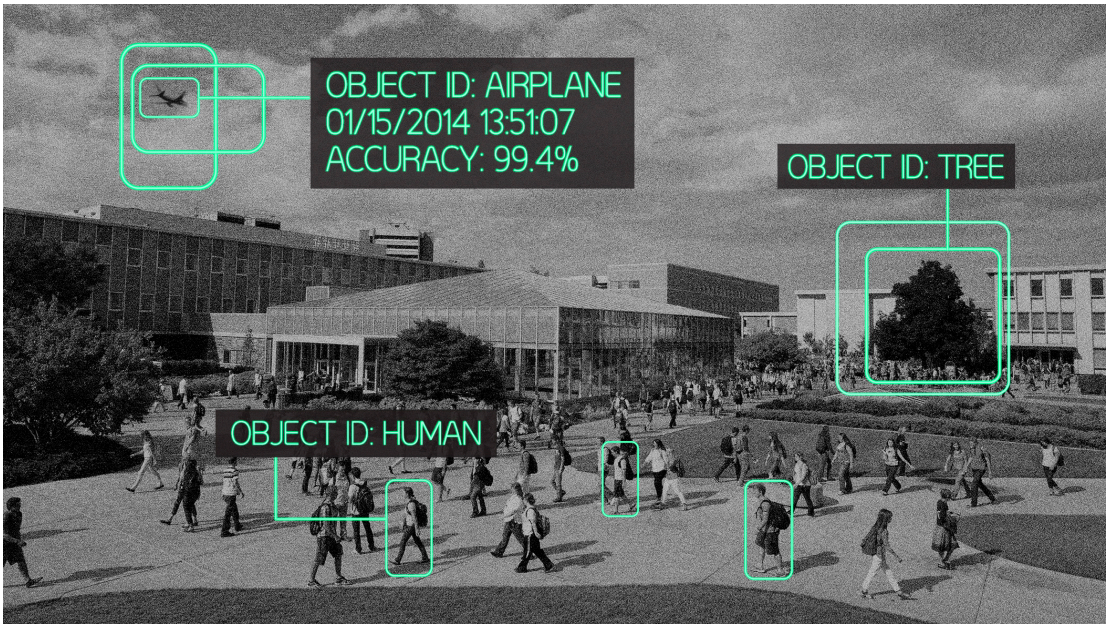
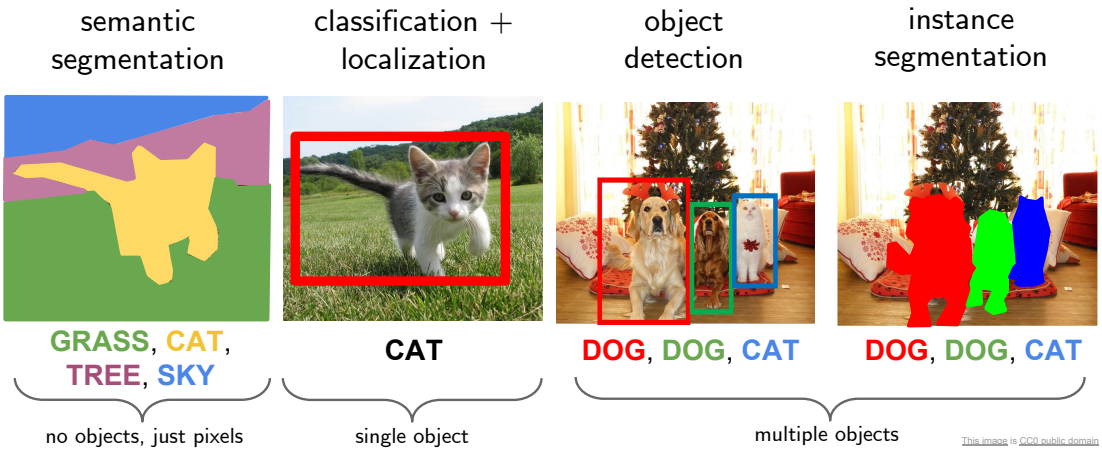


image source: https://news.byu.edu/archive14-jan-objectrecognition.aspx (no longer available)



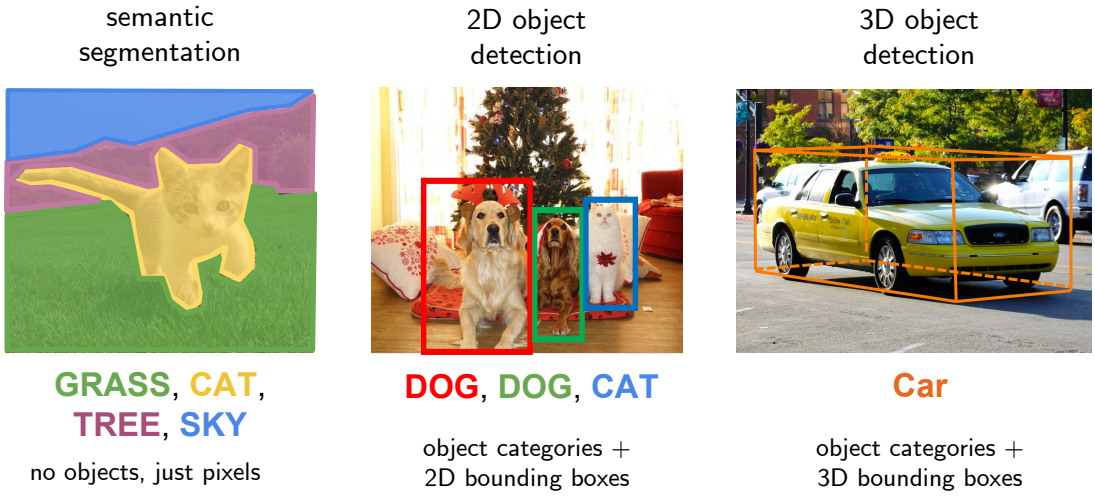


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Neural networks specifically adapted for CV

- ideal properties
  - must deal with very high-dimensional inputs
    (*e.g.* $150 \times 150 = 22,500$-dimensional inputs per channel)
  - can exploit 2D/3D topology of pixels
  - invariance to certain variations (translation, illumination)

- to this end, CNNs exploit
  1. local connectivity
  2. parameter sharing
  3. pooling/subsampling hidden units

ImageNet challenge winners

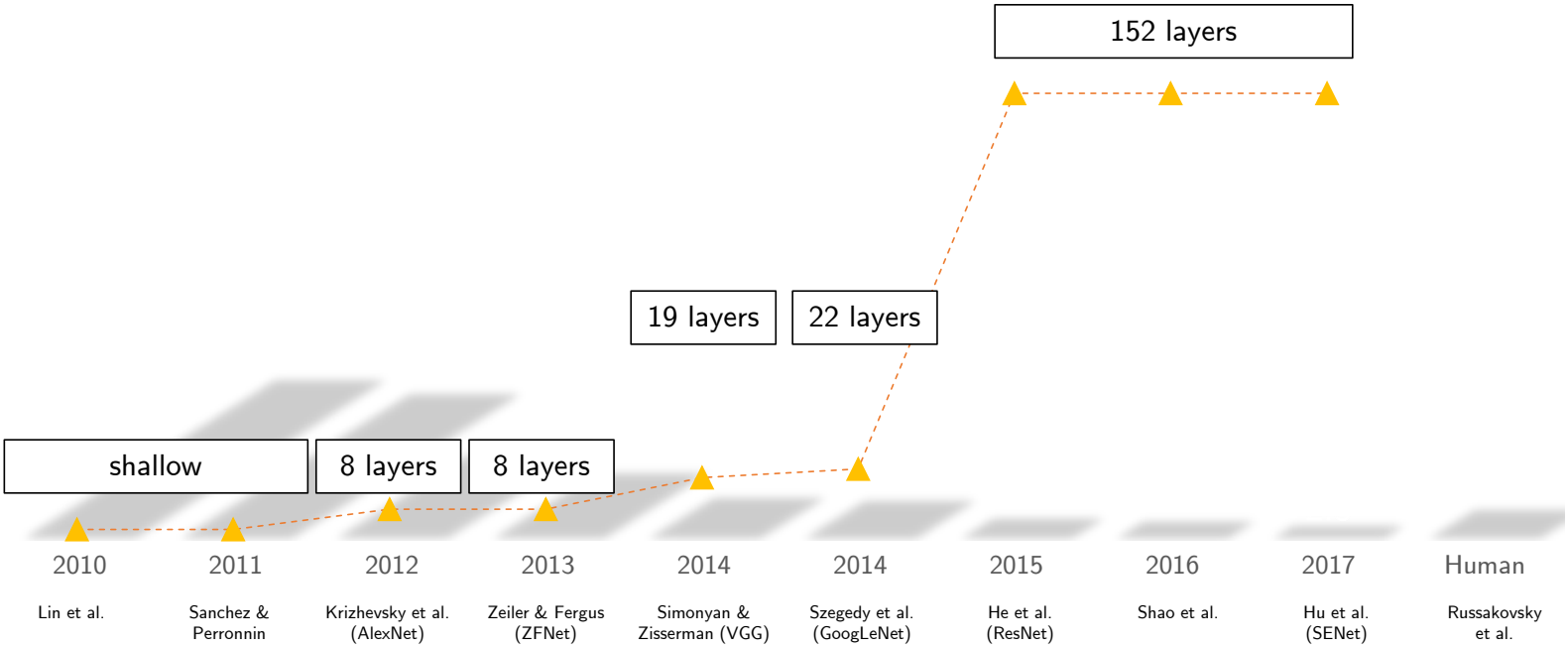


image modified from: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Outline

Motivating example

- suppose we are tracking the location of a plane with a laser sensor
  - the sensor provides a single output $x(t)$: position of the plane at time t
  - both x and t : real-valued
- now suppose: the sensor is noisy
- to get less noisy estimate of x
  - we average together several measurements
  - more recent measurements are more relevant
- we thus introduce a weighting function $w(a)$
  - to give more weight to recent measurements
  - a : the age of a measurement

Convolution

- if we apply such a weighted average operation at every moment
  - we get a new function s providing a smoothed estimate of x

$$s(t) = \sum_{-\infty}^{\infty} x(a)w(t-a)$$
$$\triangleq (x * w)(t)$$

- in CNN terminology
  - first argument (function x): *input*
  - second argument (function w): *kernel* or *filter*
  - output (function s): *feature map* or *activation map*

image source: https://en.wikipedia.org/wiki/Convolution

Multi-dimensional convolution

- convolutions over more than one axis at a time

e.g. if we use 2D image I as input

  - use 2D kernel $K \Rightarrow$ 2D convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

- note
  - convolution: commutative

$$(I * K)(i,j) = (K * I)(i,j)$$

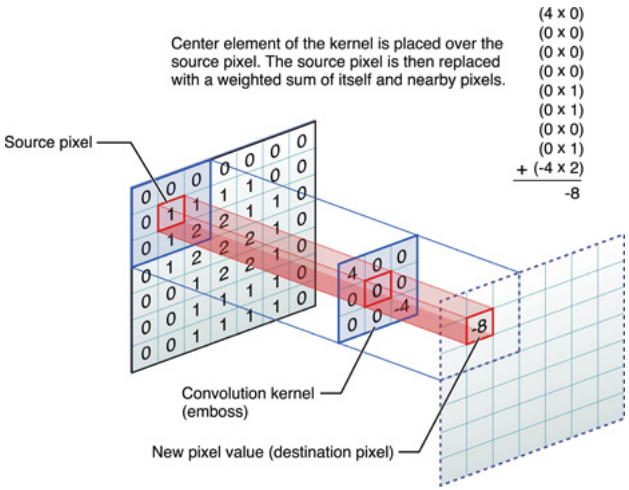


image source: Apple. *Performing Convolution Operations*. 2016. https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

Cross-correlation

- the same as convolution but without flipping the kernel:

$$\begin{aligned} S(i,j) &= (K * I)(i,j) \\ &= \sum_m \sum_n K(m,n) I(i+m, j+n) \end{aligned}$$

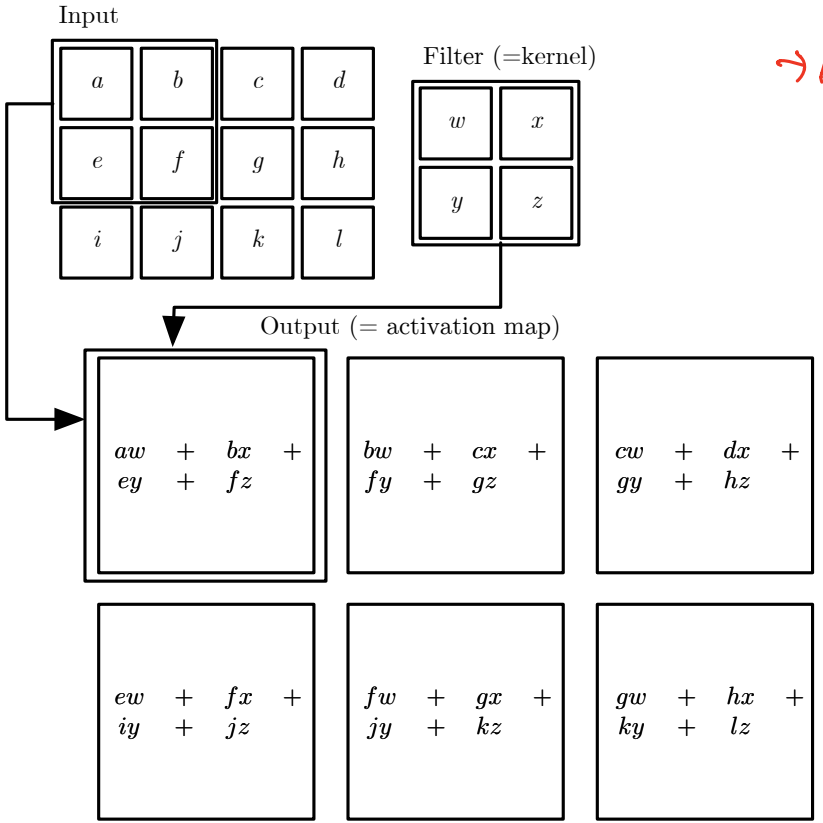
- neural net libraries implement cross-correlation but call it convolution
  - learned kernel will be in essence the same
- we call both operations convolution
  - specify whether flip (180° rotate) the kernel or not if needed

1	2	3
4	5	6
7	8	9

$\Longleftrightarrow$ flip

9	8	7
6	5	4
3	2	1

- example: convolution applied to a 2D tensor (without kernel flipping)



→ Acrobat Reader 로 보면 animation 있음.

image sources: I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016;
http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

Comparison

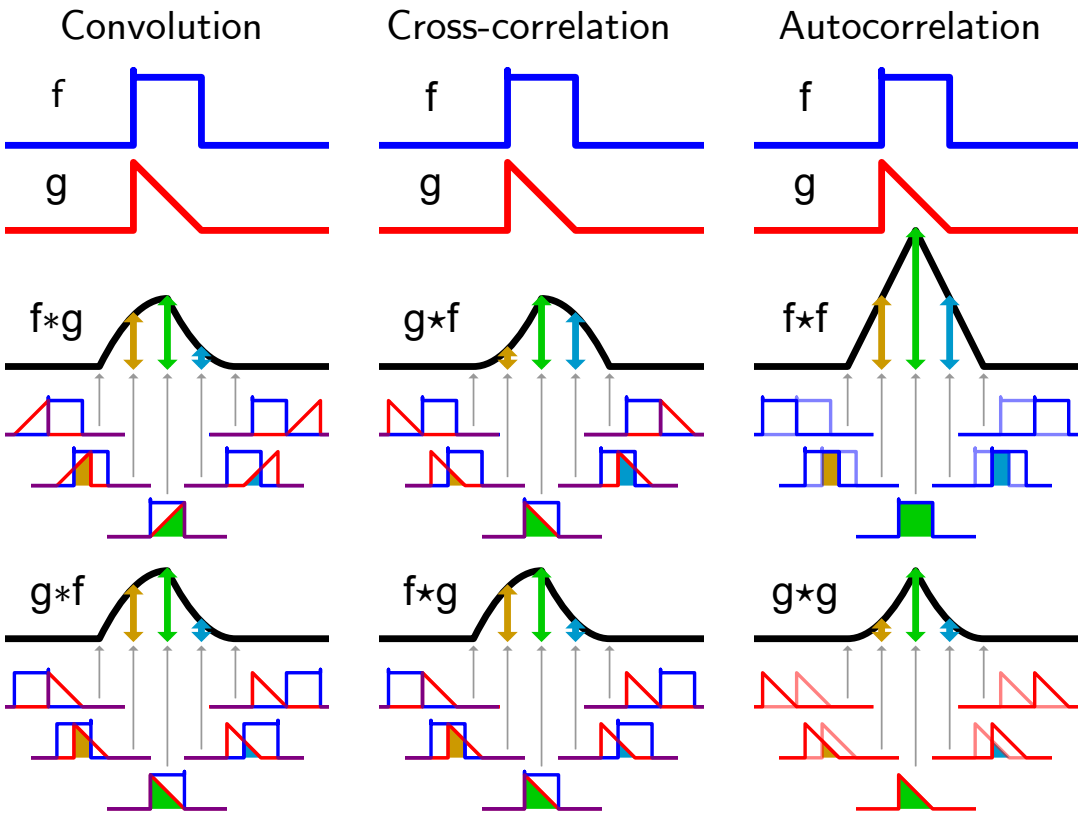
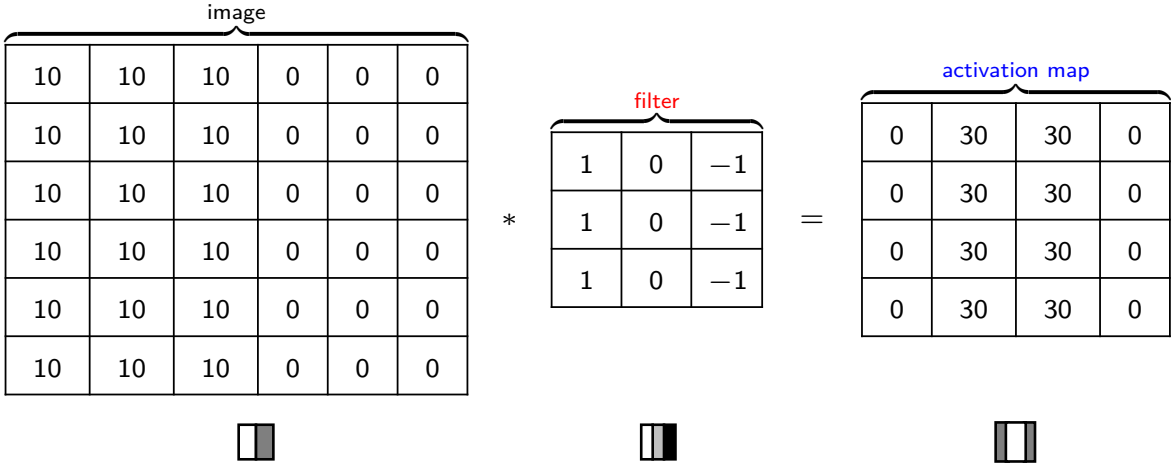


image source: https://en.wikipedia.org/wiki/Convolution

Filters

- feature detectors

  e.g. vertical edge detection



- hand-designed (conventional ML) vs learned (CNN)

image source: Ng, *Deep Learning* (Coursera), https://www.coursera.org/specializations/deep-learning

- filter examples (hand-designed):

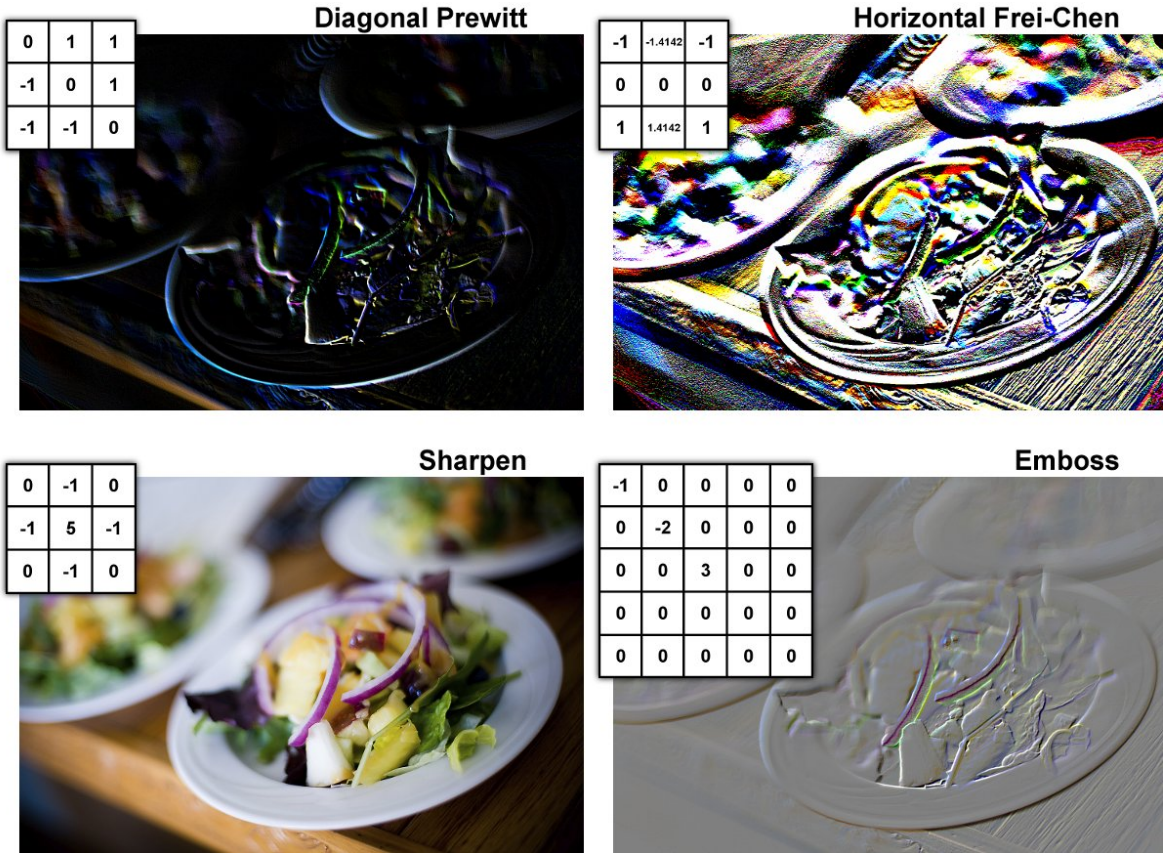


image source: http://www.gimpbible.com/files/convolution-matrix

- filter examples (learned):
  - 96 filters (size: 11×11) learned by AlexNet

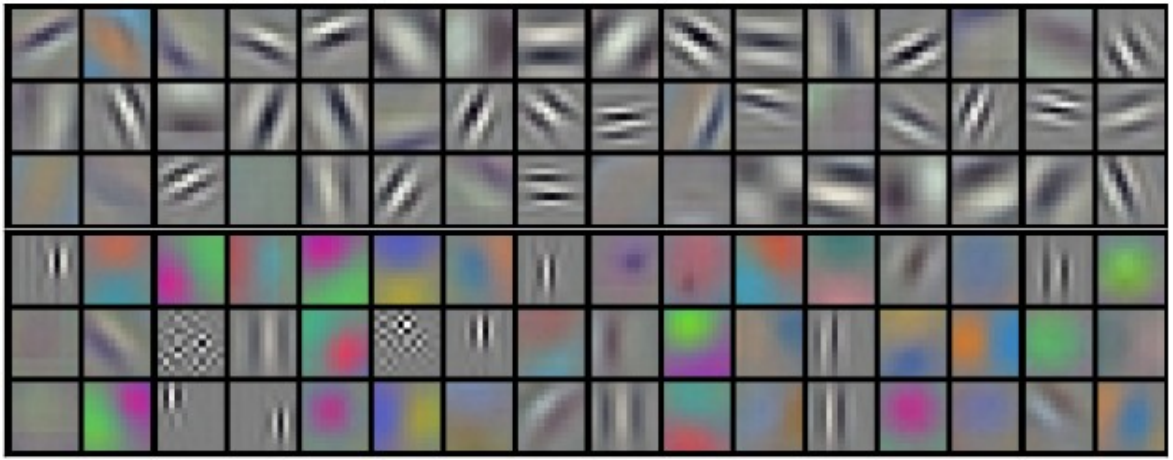


image source: A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012

Outline

Convolutions over RGB image

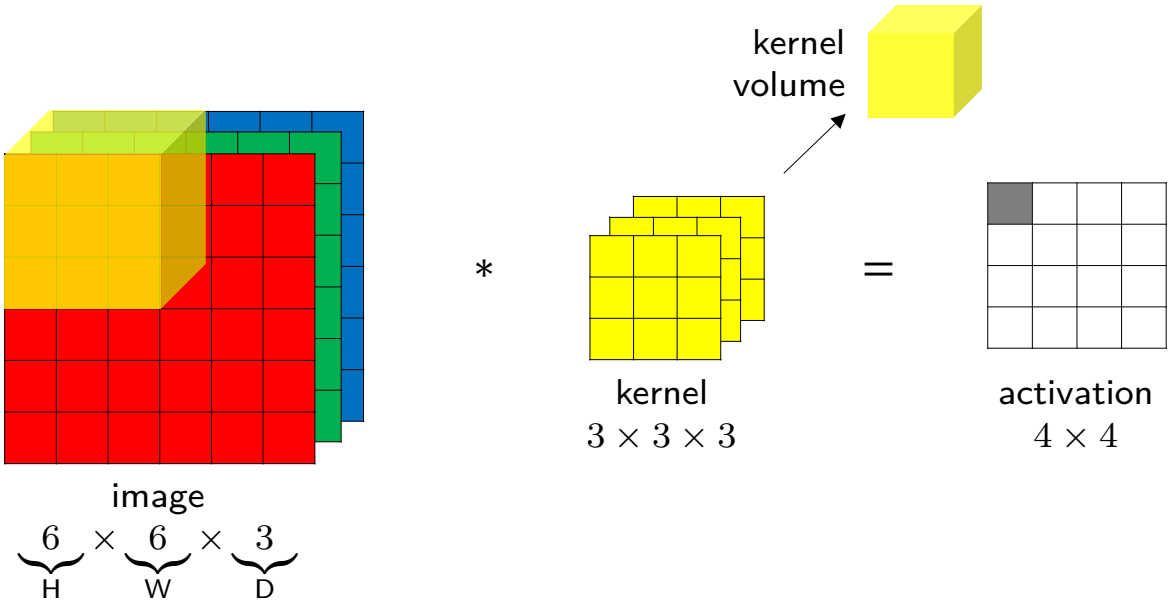


image source: Ng, *Deep Learning* (Coursera), https://www.coursera.org/specializations/deep-learning

Multiple filters

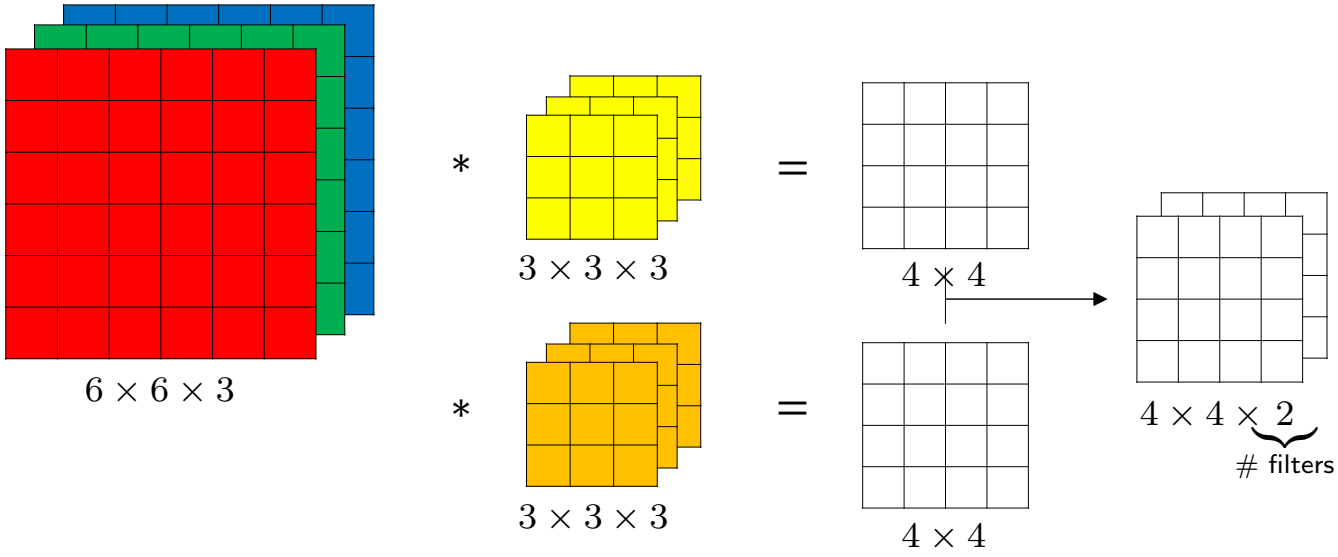


image source: Ng, *Deep Learning* (Coursera), https://www.coursera.org/specializations/deep-learning

Tensor convolution (textbook notation)

- representations
  - kernel: 4D tensor **K**

$$\mathsf{K}_{\underbrace{i}_{\substack{\text{output}\\\text{channel}}},\underbrace{j}_{\substack{\text{input}\\\text{channel}}},\underbrace{k}_{\substack{\text{row}\\\text{index}}},\underbrace{l}_{\substack{\text{column}\\\text{index}}}}$$

  - input (observed data): 3D tensor **V**

$$\mathsf{V}_{\underbrace{i}_{\text{channel}},\underbrace{j}_{\text{row}},\underbrace{k}_{\text{column}}}$$

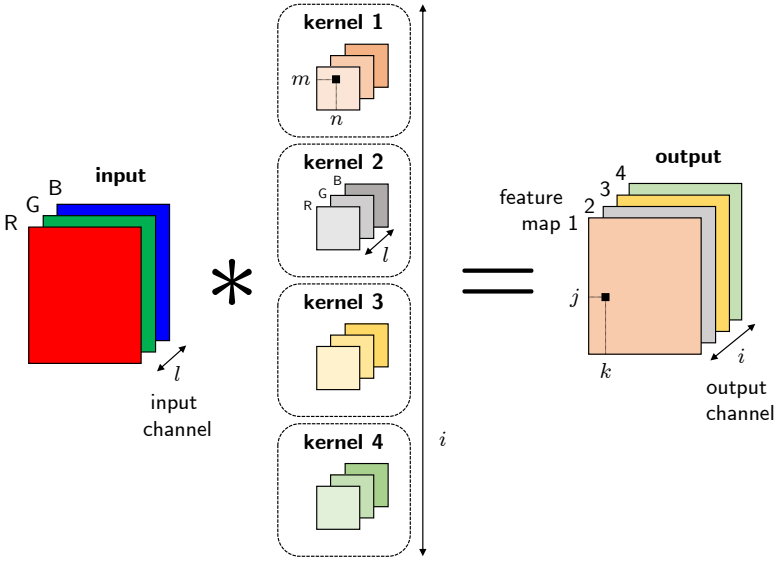
  - output: 3D tensor **Z**

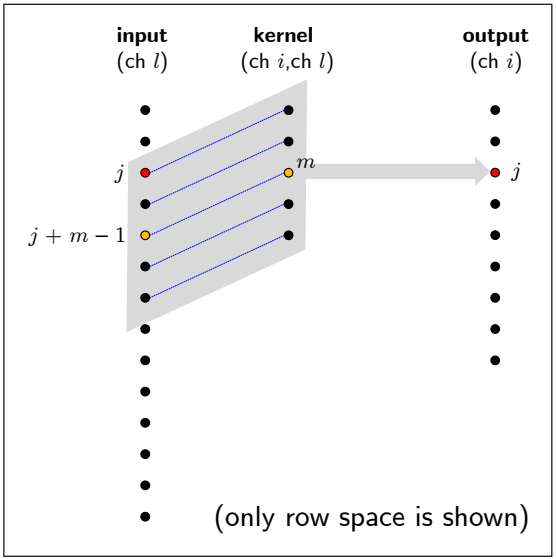
$$\mathsf{Z}_{\underbrace{i}_{\text{channel}},\underbrace{j}_{\text{row}},\underbrace{k}_{\text{column}}}$$

- if **Z** is produced by convolving **K** across **V** without flipping **K** :

$$\mathsf{Z}_{\underbrace{i}_{\text{out ch}},\underbrace{j}_{\text{row}},\underbrace{k}_{\text{col}}} = \sum_{l,m,n} \mathsf{V}_{\underbrace{l}_{\text{in ch}},\underbrace{j+m-1}_{\text{row}},\underbrace{k+n-1}_{\text{col}}} \mathsf{K}_{\underbrace{i}_{\text{out ch}},\underbrace{l}_{\text{in ch}},\underbrace{m}_{\text{row}},\underbrace{n}_{\text{col}}}$$

  - the summation[2] over l, m and n : over valid indices

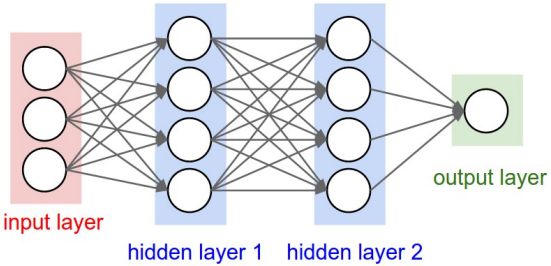




[2] in linear algebra notation, we index into arrays using a 1 for the first entry, which necessitates the -1 above; programming languages (such as C and Python) index starting from 0, rendering the above expression even simpler

Outline

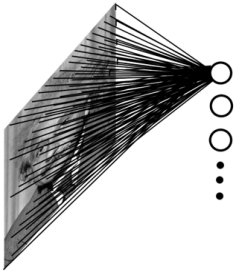
Review



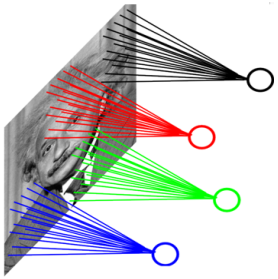
- feedforward neural nets
  - receive an input (a single vector), and
  - transform it through a series of hidden layers
- each hidden layer: made up of a set of neurons
  - each neuron: fully connected to all neurons in the previous layer
  - neurons in a single layer
    - function completely independently
    - do not share any connections
- the last fully connected layer: called the output layer
  - in classification: represents the class scores

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Motivation



- regular neural nets do not scale well to images
  - reason: full connectivity



- consider a fully connected neuron in the first hidden layer
  - to handle a color image of size 1000×1000
  - $\Rightarrow$ the neuron needs $1000 \times 1000 \times 3 = 3 \times 10^6$ weights

- this full connectivity: wasteful
  - the huge number of parameters $\Rightarrow$ quickly lead to overfitting

- CNNs
  - take advantage of the fact that input consists of images
  - constrain the architecture in a more sensible way

image source: M. Ranzato, *CVPR 2014 Tutorial on Large-Scale Visual Recognition* (2014), https://sites.google.com/site/lsvrtutorialcvpr14/home/deeplearning

Local connectivity

- connectivity of each neuron in CNN
  - spatial (height and width) axes: only a local region of the input volume
    - ▷ filter size = size of this local region (called receptive field)

      ↑

      hyperparameter
  - depth axis: the same as input depth (# ch)

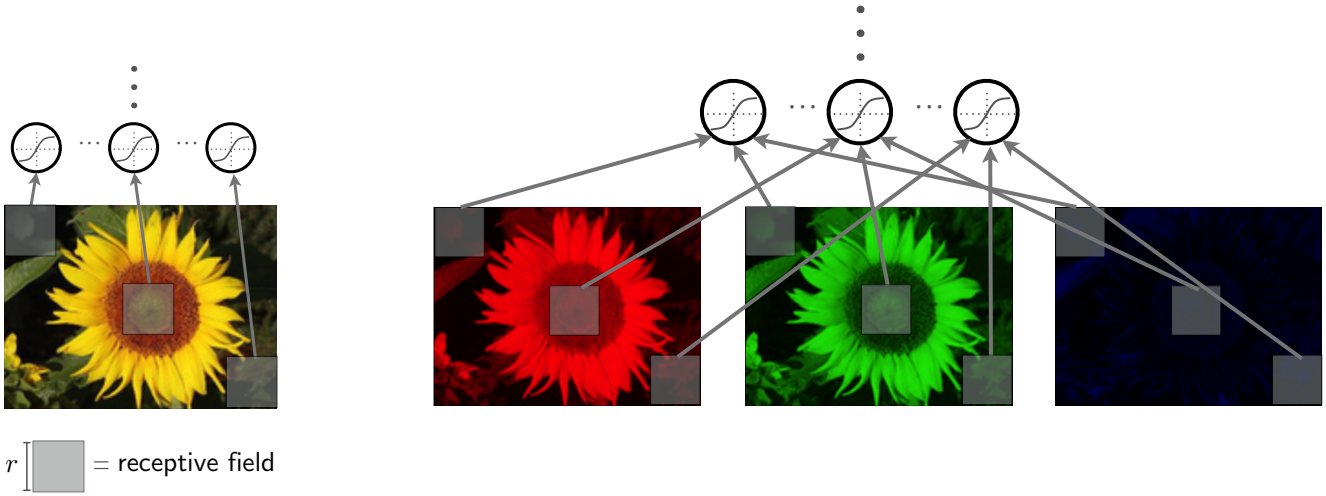


image source: H. Larochelle, *Online Course on Neural Networks*, www.dmi.usherb.ca/~larocheh/neural_networks

example:

- input volume: $32 \times 32 \times 3$ (*e.g.* an RGB CIFAR-10 image)
  - if receptive field (or filter size) is 5×5, then
  - each neuron will have weights to a $5 \times 5 \times 3$ region in input volume
  - $\#$ weights $= 5 \times 5 \times 3 = 75$ (and $+1$ bias parameter)

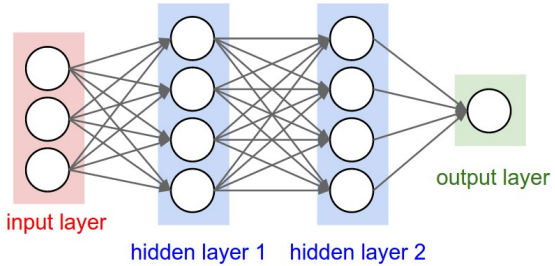
- the extent of the connectivity along the depth axis
  - must be 3 (this is the depth of the input volume)

3D volumes of neurons

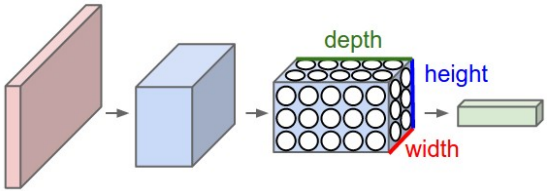
- neurons in a CNN
  - arranged in 3D: height, width, depth[3]

  e.g. a CIFAR-10 image $\Rightarrow$ an input volume of $32 \times 32 \times 3$

- every layer of a CNN
  - transforms 3D input volume to 3D output volume of neuron activations



(a) regular 3-layer neural net



(b) CNN

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

[3] the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full neural net (= total number of layers in a network)

- the neurons in a layer
  - will only be connected to a small region of the layer before it

    (instead of all of the neurons in a fully connected manner)

- the final output layer: a single vector of class scores (arranged along the depth)

  e.g. for CIFAR-10: $1 \times 1 \times 10$
  - CNN reduces the full image into a vector of class scores

- bottom line
  - a CNN is made up of layers
  - each layer = a volume transformer
    - ▷ $\boxed{\text{3D volume} \xrightarrow{\text{transform}} \text{3D volume}}$
    - ▷ uses a differentiable function that may or may not have parameters

Building a CNN

- four main types of layers $\Rightarrow$ Stacking them gives a full CNN architecture
  1. convolutional layer
  2. RELU layer
  3. pooling layer
  4. fully connected layer

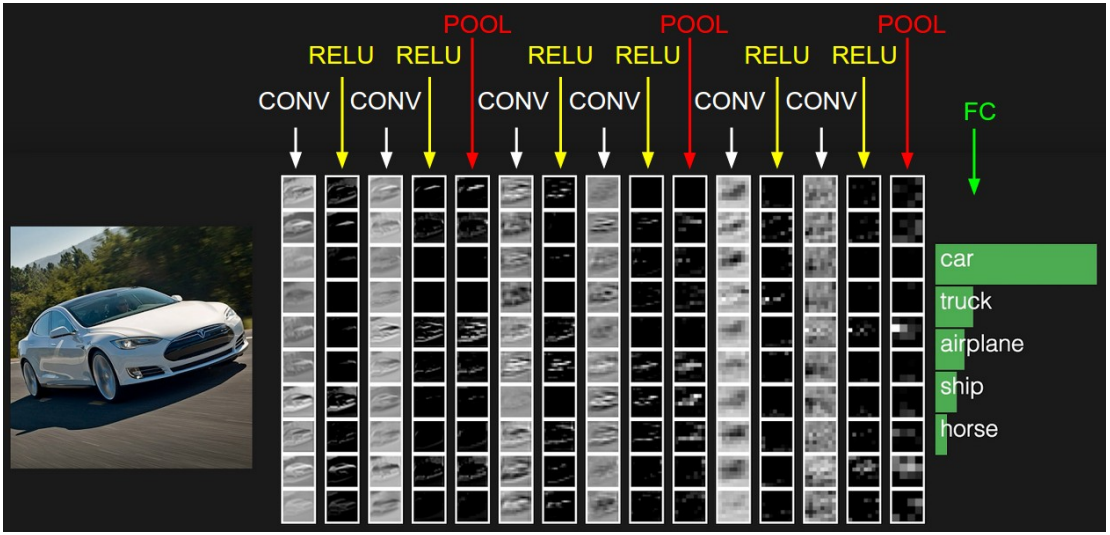


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

- each layer may or may not have (hyper)parameters

layer	parameters	hyperparameters
CONV	○	○
RELU	×	×
POOL	×	○
FC	○	○

- parameters
  - trained with gradient descent[4] (in a supervised fashion)

[4] the backward pass for a convolution operation (for both the data and the weights): also a convolution (but with spatially flipped filters)

Hierarchical feature learning

- CNN learns filters that activate when seeing some type of visual feature
  - implements hierarchical representation learning

  e.g. an edge of some orientation or a blotch of some color (first layer)

  more complicated patterns (higher layers)

image source: Ng, *Deep Learning* (Coursera), https://www.coursera.org/specializations/deep-learning

Outline

Convolutional layer

- the core building block of a CNN
  - does most of the computational heavy lifting

    input volume (3D) $*$ a filter $=$ output map (2D)

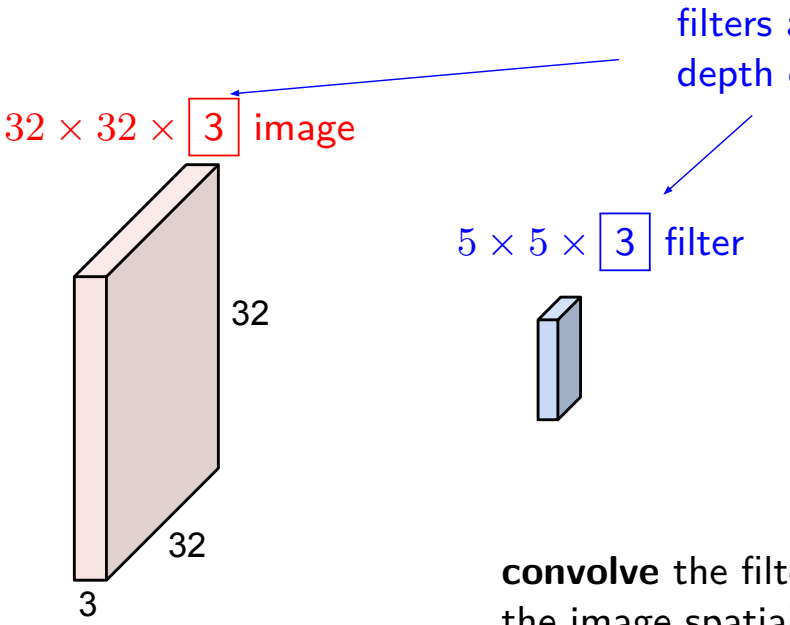
    input volume (3D) $*$ filters $=$ output volume (3D)

- each filter (3D):
  - is small spatially along height and width
  - but extends through the full depth of the input volume

- *e.g.* a typical filter on a first layer of a CNN
  - might have size $5 \times 5 \times 3$ (height $\times$ width $\times$ depth)

    ↑

    RGB color channels



convolve the filter with the image (*i.e.* slide over the image spatially, computing dot products)

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

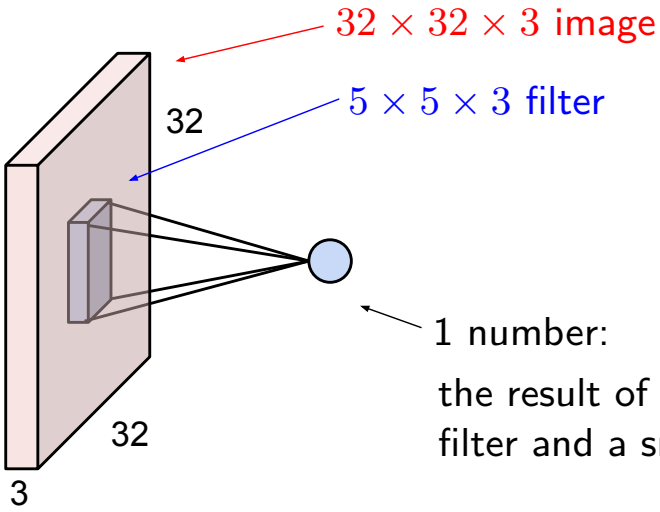


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

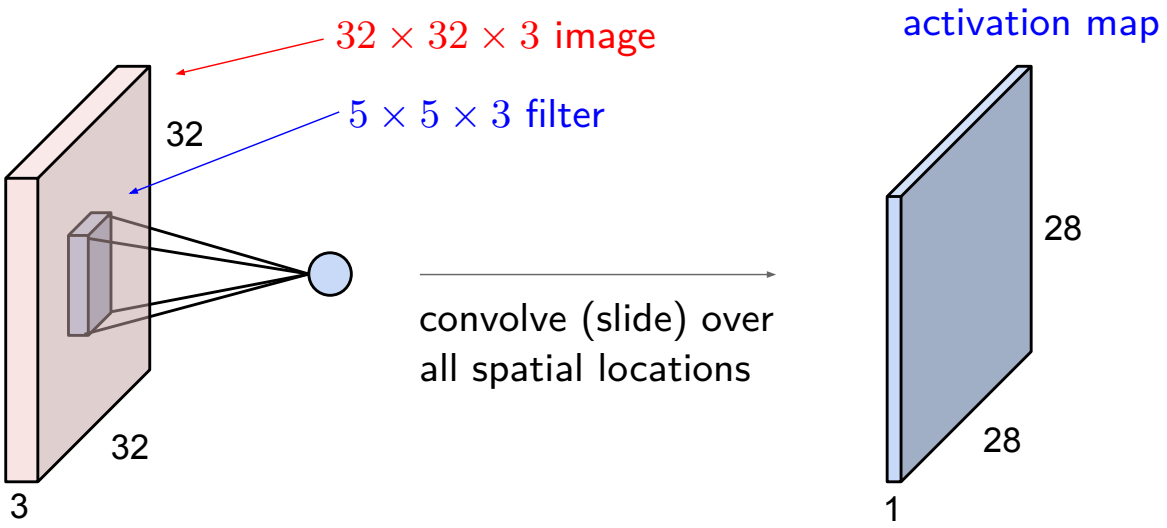


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Why convolution?

1. parameter sharing[5]
   - a feature detector (*i.e.* filter) useful in one part of an image

   $\Rightarrow$ probably useful in another part of the image

2. sparse interactions (aka sparse connectivity/weights)
   - in each layer, each output value depends only on a small # inputs

3. flexibility
   - convolution allows us to work with inputs of variable size

- taken together
  - significant reduction in number of parameters
  - significant gain in performance (generalization & efficiency)

  e.g. 16,000,000,000 ops $\rightarrow$ 267,960 ops (fig 9.6 in textbook)

[5] causes the conv layer to have a property called equivariance to translation

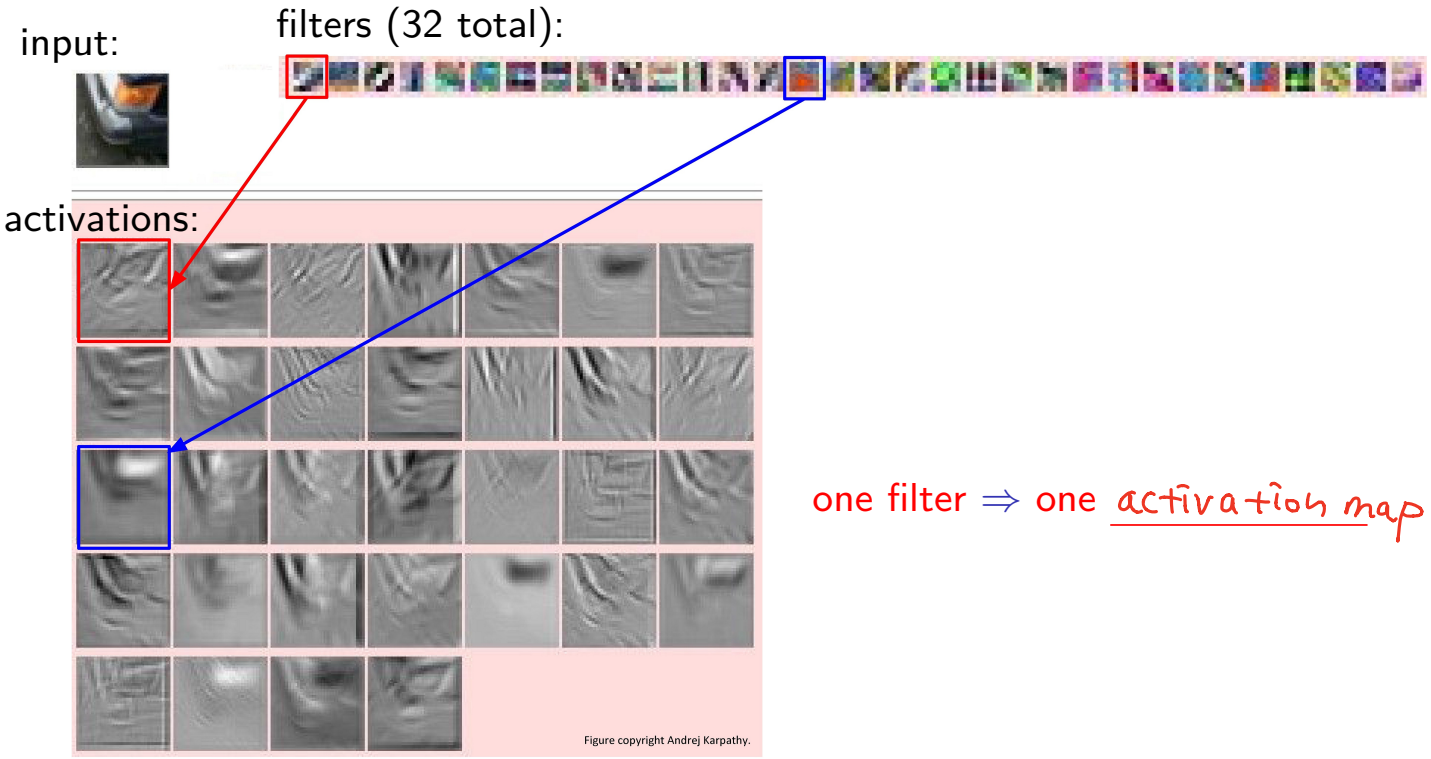
Producing output volume

- each CONV layer
  - has a set of $\underbrace{\text{filters}}_{\uparrow}$ (*e.g.* 12 filters)

    each of them will produce a separate 2D activation map
  - stacks these activation maps along the depth dimension

  $\Rightarrow$ produces output volume

- example: applying 32 filters $\Rightarrow$ output depth $= 32$



one filter $\Rightarrow$ one activation map

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

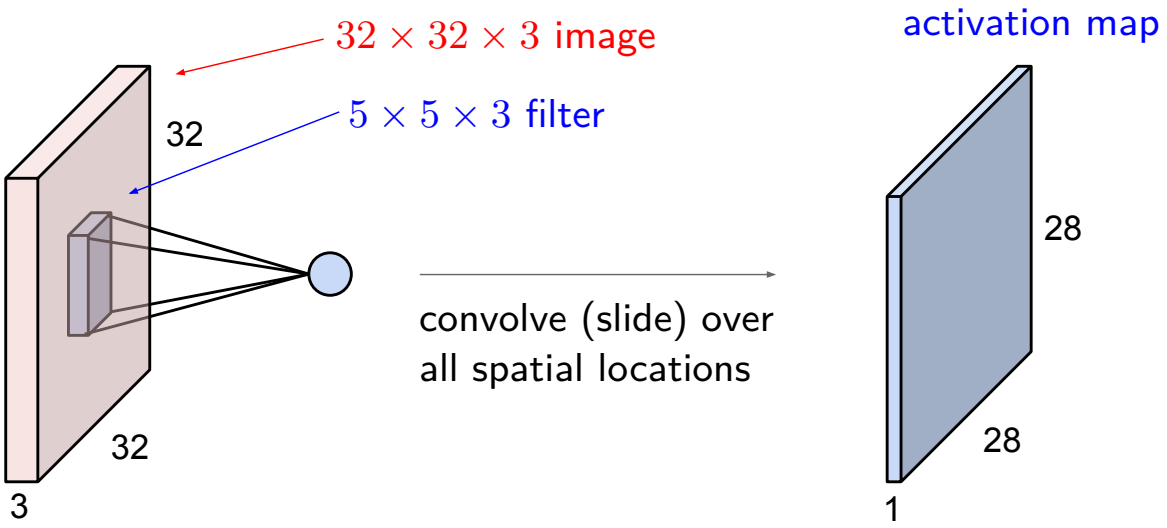


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

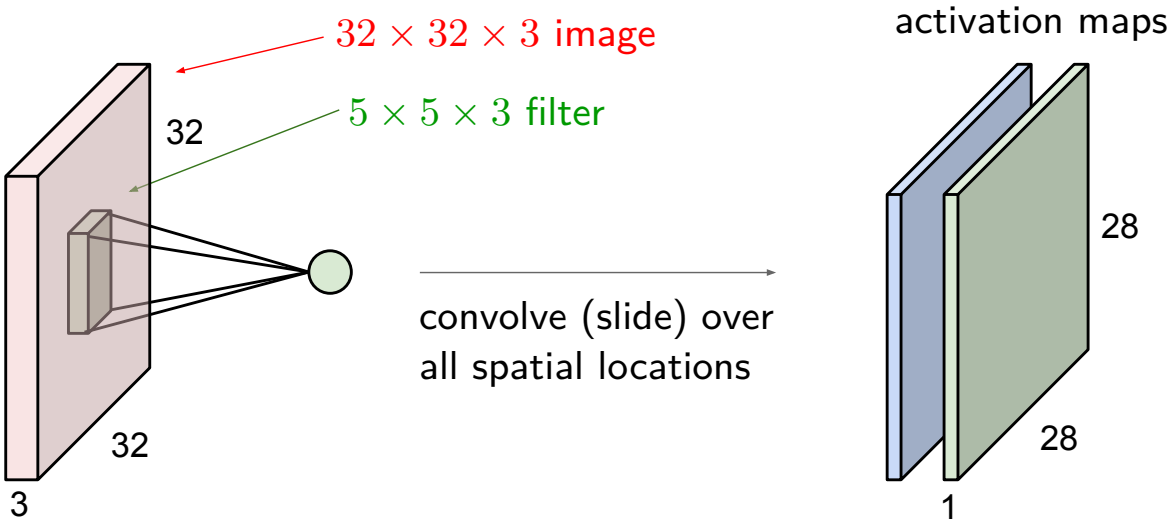
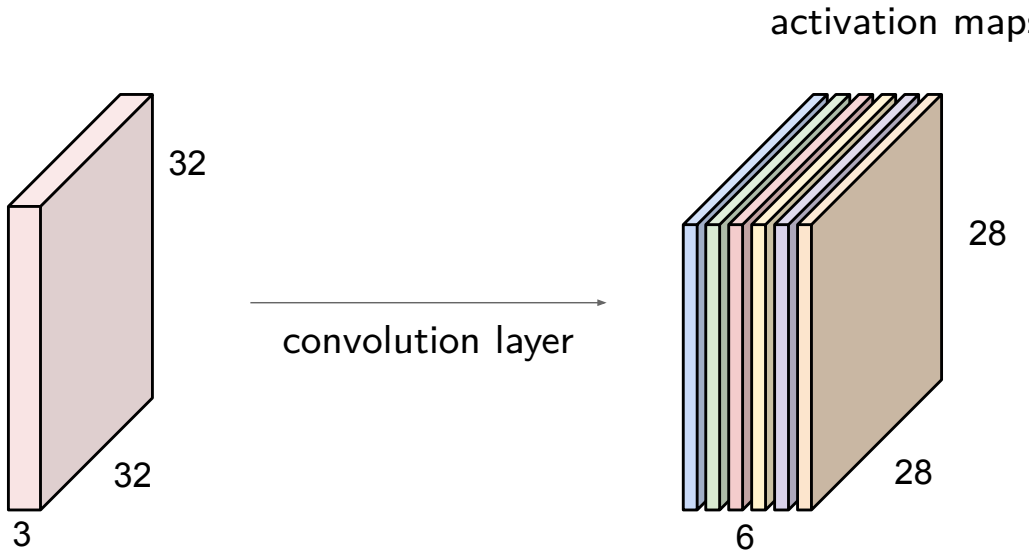


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

- *e.g.* if we had 6 5×5 filters $\Rightarrow$ get 6 separate activation maps:



- we stack these up to get a "new image" of size $28 \times 28 \times 6$

volume

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Spatial arrangement

- connectivity of each neuron in CONV layer
  - to the input volume: explained (*i.e.* spatially local but full depth)
  - to the output volume: controlled by three hyperparameters

1. depth
   - means depth of the output volume = # filters we would like to use
2. Stride
   - specifies how many pixels to jump when sliding each filter
3. Zero-padding
   - how to pad the input volume with zeros around the border
   - controls the spatial size (*i.e.* width and height) of the output volume

Example

- 7×7 input (spatially)
  - filter size: 3×3
  - stride: 1

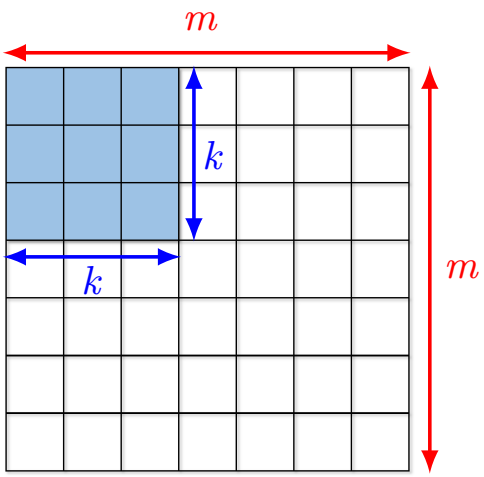
  $\Rightarrow$ 5×5 output

  - filter size: 3×3
  - stride: 2

  $\Rightarrow$ 3×3 output

Output size

- hyperparameters (spatially)
  - input size: $m \times m$
  - filter size: $k \times k$
  - stride: s



- output size:

$$\frac{m-k}{s}+1$$

- *e.g.* $m = 7,\ k = 3$

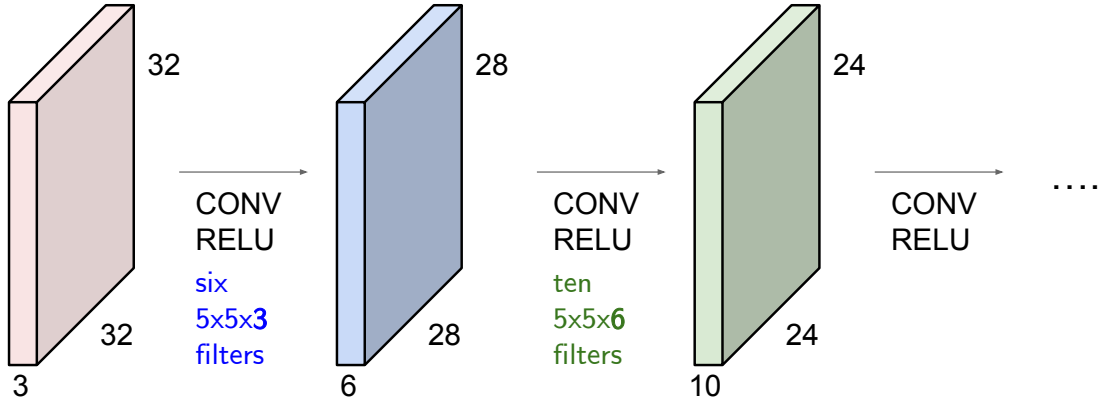
  $s = 1 \Rightarrow (7-3)/1+1 = 5$

  $s = 2 \Rightarrow (7-3)/2+1 = 3$

  $s = 3 \Rightarrow (7-3)/3+1 = 2.3$ (not fit)

Motivations for zero-padding

- consider a 32×32 input convolved repeatedly with 5×5 filters
  - $\Rightarrow$ volume shrinks spatially ($32 \rightarrow 28 \rightarrow 24 \cdots$)
    - shrinking too fast: not good (does not work well)



- pixels along the edges: less used for convolutions $\Rightarrow$ potential information loss

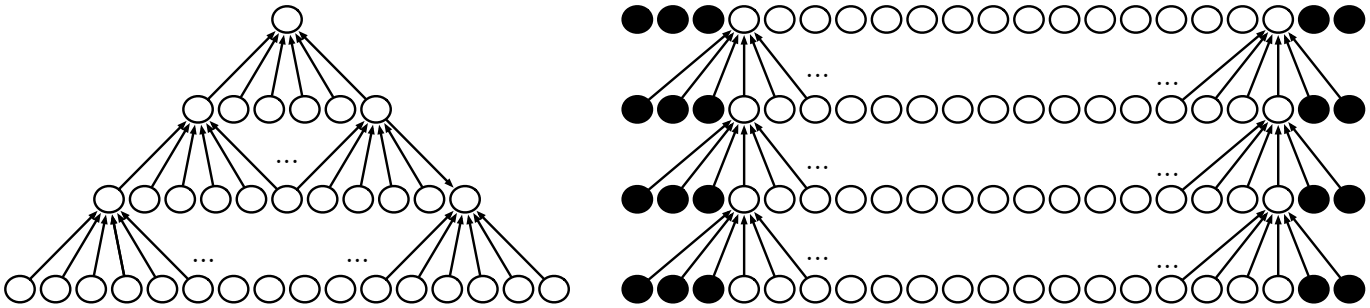
image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Zero-padding

- one essential feature of any CNN implementation
  - implicitly zero-pad input to make it wider
  - allows us to control kernel width and output size independently

- without this feature, we must
  - shrink the spatial extent of the net rapidly or use small filters

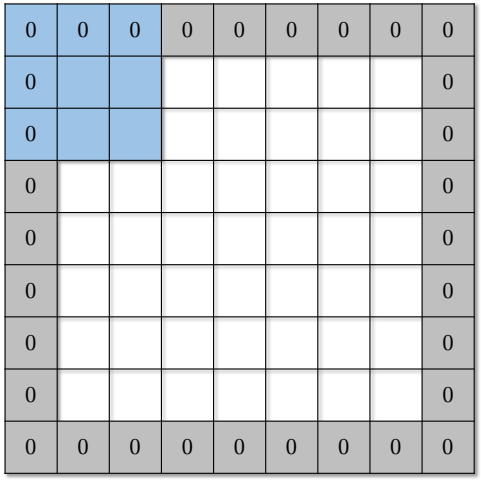
  $\Rightarrow$ both significantly limit expressive power of the net



- also helpful for preserving information along the edges

image source: I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016

Example



- 7×7 input, 3×3 filter, stride 1
  - zero-pad with 1 pixel border
  - otuput size: $(7 + 2 - 3)/1 + 1 = 7$

  $\Rightarrow$ input size preserved after convolution

- in general, to preserve spatial size
  - zero-pad with $(k-1)/2$

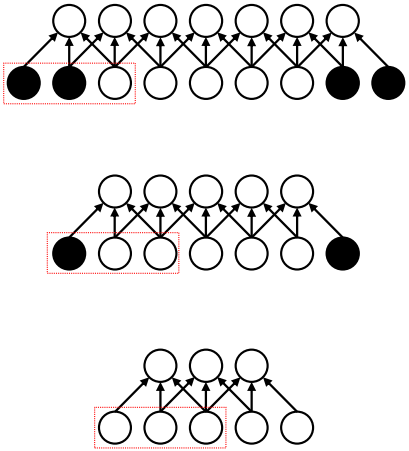
  e.g. $k = 3 \Rightarrow$ zero pad with 1

  $k = 5 \Rightarrow$ zero pad with 2

Three types of zero-padding schemes

type	output	# zeros padded		
		left	right	total
full	$m + (k-1)$	$k-1$	$k-1$	$2(k-1)$
same	m	k even: $\lfloor \frac{k-1}{2} \rfloor$	k even: $\lfloor \frac{k-1}{2} \rfloor + 1$	$k-1$
		k even: $\lfloor \frac{k-1}{2} \rfloor + 1$	k even: $\lfloor \frac{k-1}{2} \rfloor$	
		k odd: $\frac{k-1}{2}$	k odd: $\frac{k-1}{2}$	
valid	$m - (k-1)$	0	0	0

(m: input width; k: kernel width; stride $s = 1$)



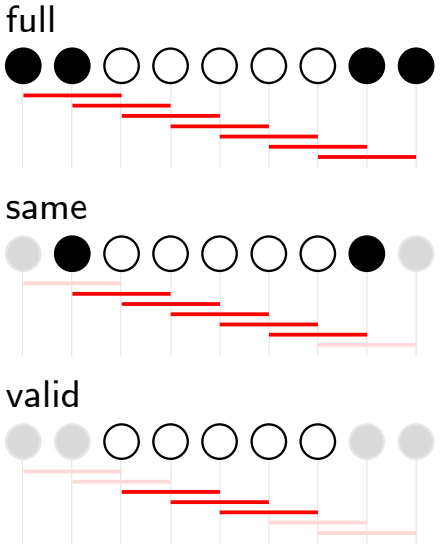
optimal zero padding (in terms of test accuracy)

- usually lies somewhere between "valid" and "same" convolution

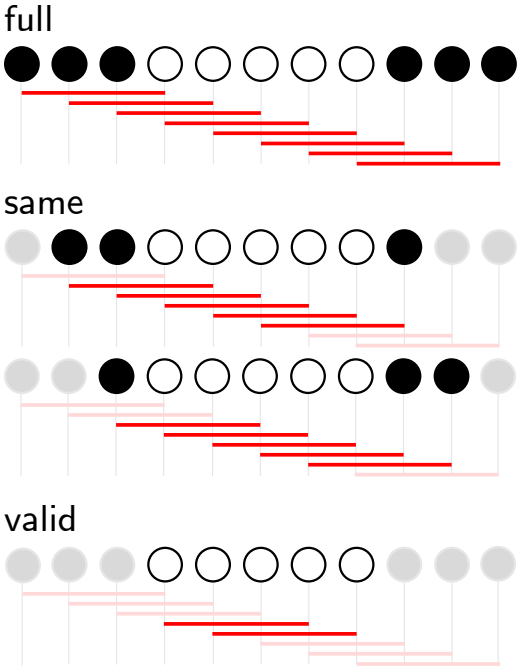
※ value of k: typically odd

examples:

- $m = 5, k = 3, s = 1$



- $m = 5, k = 4, s = 1$



Summary: activation map size

- input size: $m \times m$
- hyperparameters
  - filter size: $k \times k$ (k : normally odd)
  - padding: p per side
  - stride: s
- output size:

$$\underbrace{\left\lfloor \frac{m+2p-k}{s} + 1 \right\rfloor}_{\text{height}} \times \underbrace{\left\lfloor \frac{m+2p-k}{s} + 1 \right\rfloor}_{\text{width}}$$

Summary: convolution layer l

- notation
  - $f^{[l]}$: filter size
  - $p^{[l]}$: zero padding size
  - $s^{[l]}$: stride size
  - $n_c^{[l]}$: total # filters in l
- total # parameters in l
  - # weights $= f^{[l]} \cdot f^{[l]} \cdot n_c^{[l-1]} \cdot n_c^{[l]}$
  - # biases $= n_c^{[l]}$ (1 per filter)

dimensions

- input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
- each filter: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
- output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
  - height & width:

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

  - for size-m minibatch:

    $\mathbf{A}^{[l]} : m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

    (default order in TF)

Outline

Other layers

- three more layers to go:
  - RELU layer
  - pooling layer
  - fully connected layer

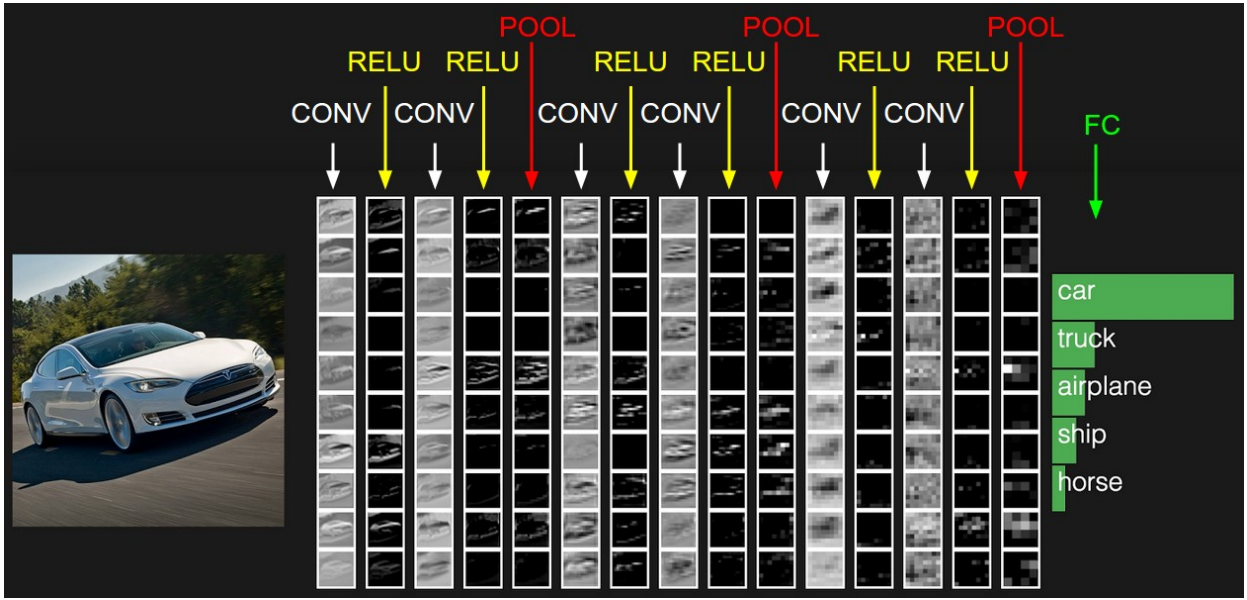


image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

RELU layer

- purpose
  - increase the nonlinear properties (of decision function and of overall net)
- facts
  - often called detector (or nonlinear) stage
  - introduces no (hyper)parameters

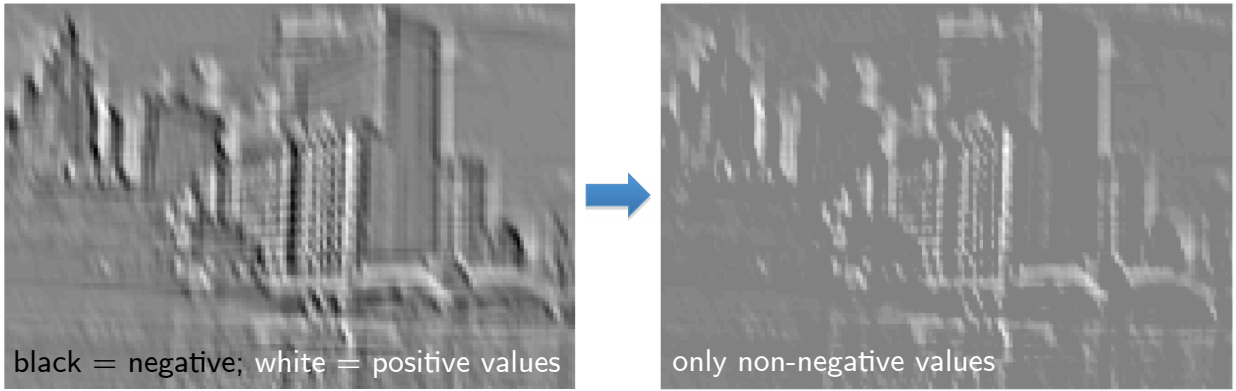


image source: Fergus *et al.*, *Deep Learning and Feature Learning Methods for Vision*, CVPR 2012 Tutorial, http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Pooling layer

- common practice:
  - periodically insert a pooling layer in-between successive conv layers

- the function of pooling layers:
  - progressively reduce the Spatial size of the representation

  $\Rightarrow$ reduce the amount of parameters and computation + control overfitting

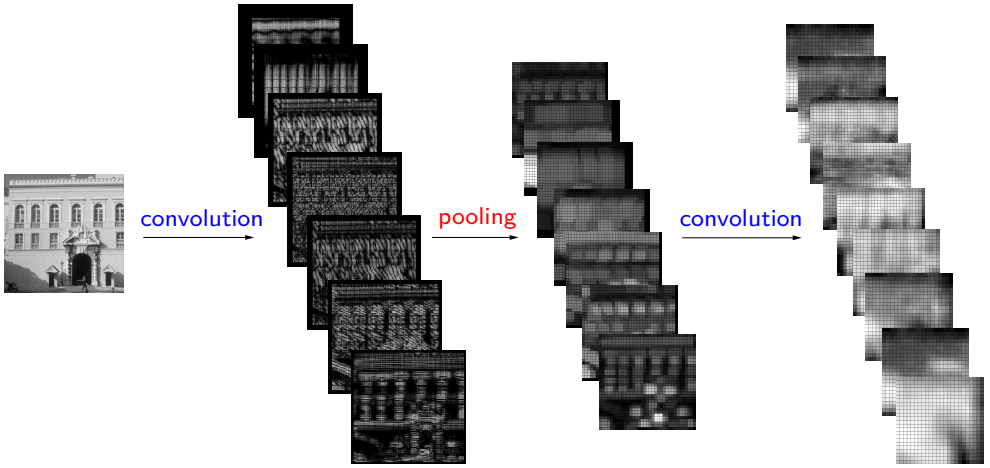
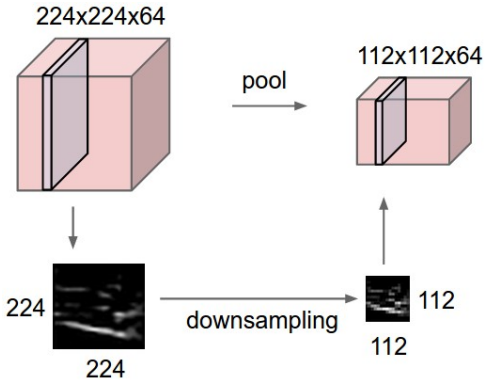


image source: C. Thériault, N. Thome, and M. Cord, "Extended coding and pooling in the HMAX model," *IEEE Transactions on Image Processing*, vol. 22, no. 2, pp. 764–777, 2012

- each pooling layer
  - operates independently on every depth slice of the input
  - resizes it spatially (using *e.g.* max operation)

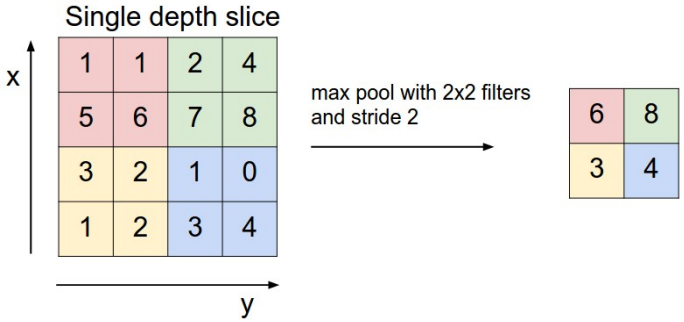


- facts
  - the depth dimension remains unchanged
  - pooling introduces no parameters (hyperparameter : 0)
  - pooling gives invariance to (local) translation

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Max pooling

- the most common form: 2×2 max pooling with stride 2
  - discards 75% of activations



- other types of pooling: average pooling, L^2-norm pooling, ...

  ↑ often used historically but phased out by max pooling

image source: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html

Eliminating pooling

- some researchers proposes to discard pooling layer
  - in favor of architecture that only consists of repeated CONV layers

  i.e. use larger stride in CONV layer once in a while

- also been found important in training good generative models

  e.g. generative adversarial nets (GANs), variational autoencoders (VAEs)

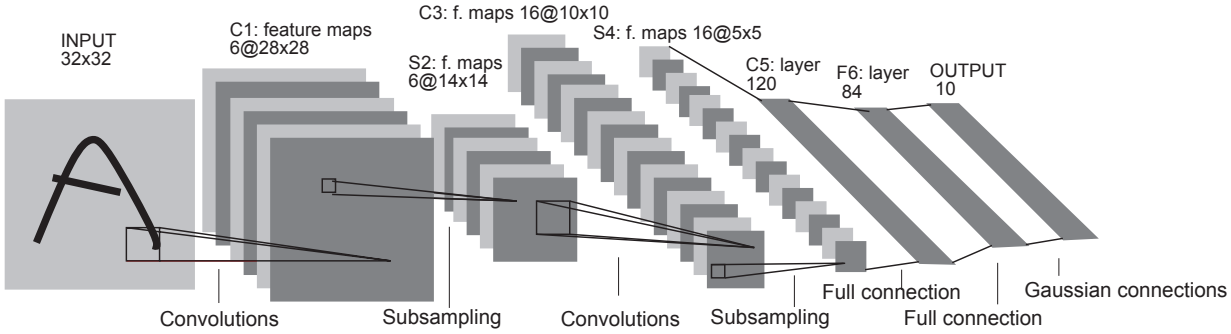
- future architectures
  - likely to feature very few to no pooling layers

Fully connected (FC) layer

- connections:
  - the same as in regular multilayer perceptrons
  - often dropout is used to reduce overfitting in FC layer
- role of FC layer
  - high-level reasoning such as classification and regression
  - often with the softmax function embedded
- remarks: FC layers
  - not spatially located any more $\Rightarrow$ no conv layers possible after a FC layer
  - may account for the majority of parameters in a CNN architecture
    - recent models try to minimize use of FC layers

Putting it all together

- Lenet-5 for digits recognition:
  - layer $1 \rightarrow$ layer l : input size $\downarrow$ but # filters $\uparrow$ (common trend in CNNs)



- demo
  - ConvNetJS: training on CIFAR-10 ▸ Link

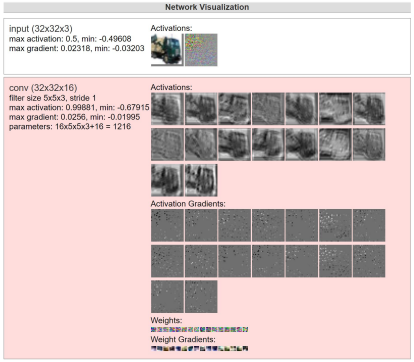


image sources: Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998;
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

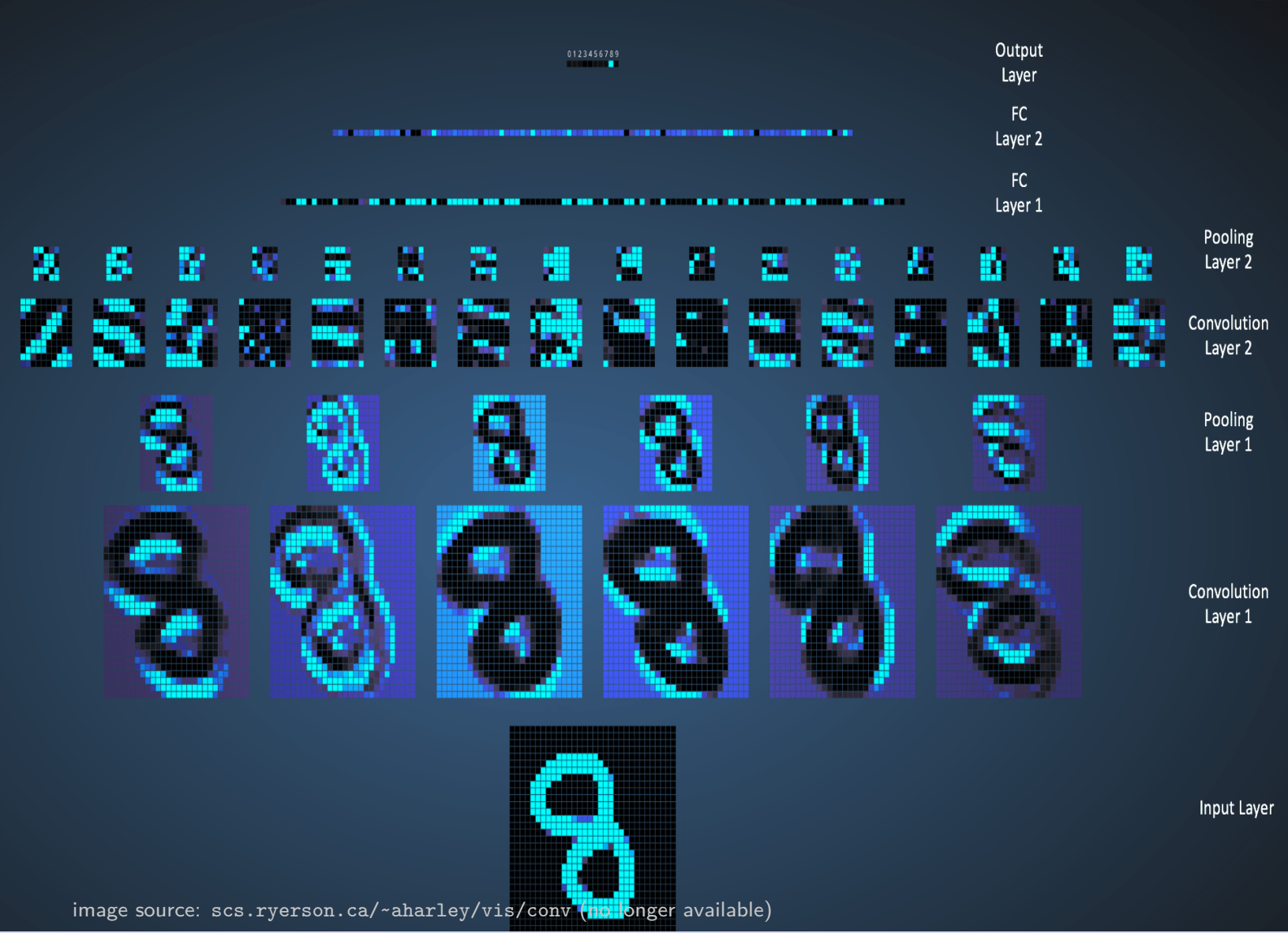


image source: scs.ryerson.ca/~aharley/vis/conv (no longer available)

Outline

Summary

- convolution (neural) networks (CNNs)
  - neural networks with convolution operations
  - specialized for grid topology (*e.g.* images and time series)
  - tremendous commercial success (intense interest by industry)
- fundamental principles and properties
  - sparse interactions, parameter sharing $\Rightarrow$ efficiency
  - equivariance (convolution), invariance (pooling)
- CNN layers: convolution, RELU, pooling, and fully connected
  - transform 3D input $\rightarrow$ 3D output by differentiable function
  - may or may not have (hyper)parameters
  - trained by backpropagation