



M2177.003100

Deep Learning

[4: Convolutional Neural Nets (Part 1)]

Electrical and Computer Engineering
Seoul National University

© 2018 Sungroh Yoon. this material is for educational uses only. some contents are based on the material provided by other paper/book authors and may be copyrighted by them.

(last compiled at 22:31:00 on 2018/09/12)

Outline

Introduction

Additional Topics

Convolution Operation

Summary

Architecture

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](#)
 - ▶ *Machine Learning Yearning* [▶ Link](#)
- note:
 - ▶ you should **open this file in Adobe Acrobat** to see animated images
(other types of pdf readers will not work)

Outline

Introduction

Additional Topics

Convolution Operation

Summary

Architecture

Convolutional (neural) networks (CNNs)

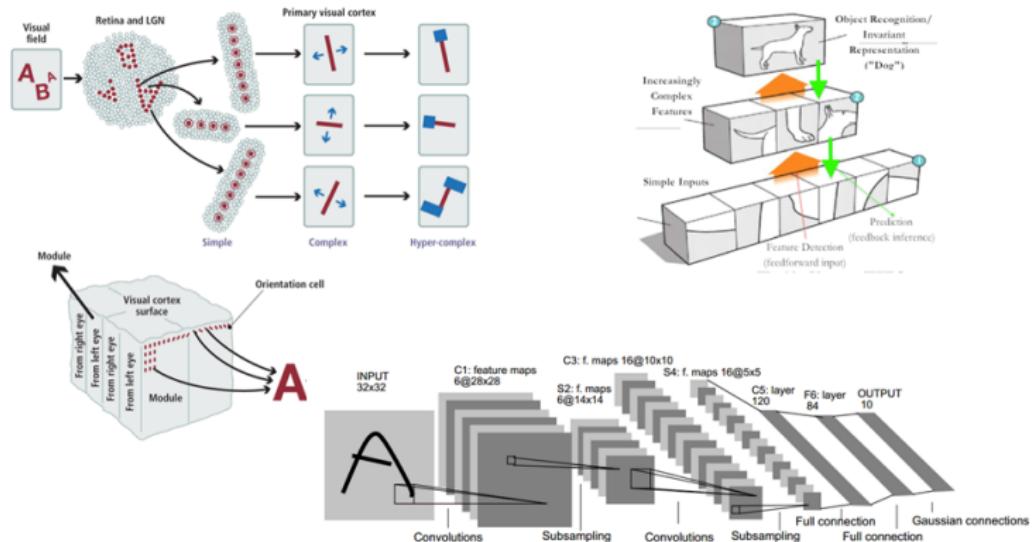
- simply neural nets that use **convolution** in their layers
 - ▶ in place of general matrix multiplication
- employ a mathematical operation called *convolution*
 - ▶ convolution¹: a special kind of linear operation
- tremendously successful in practical applications
 - ▶ especially for data with a 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

¹usually 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:

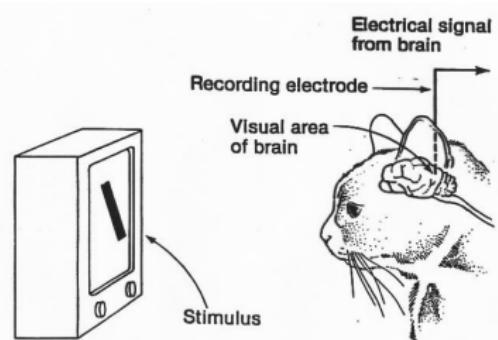
- ▶ simple cells → complex cells → hyper-complex cells



(source: LeCun)

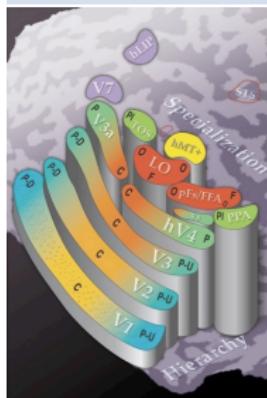
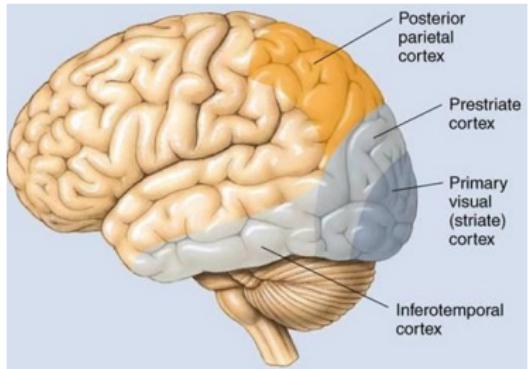
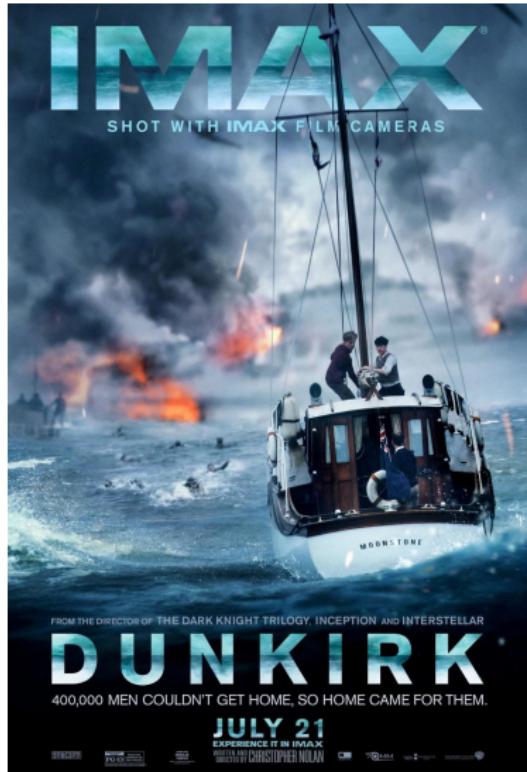
Hubel and Wiesel

- 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



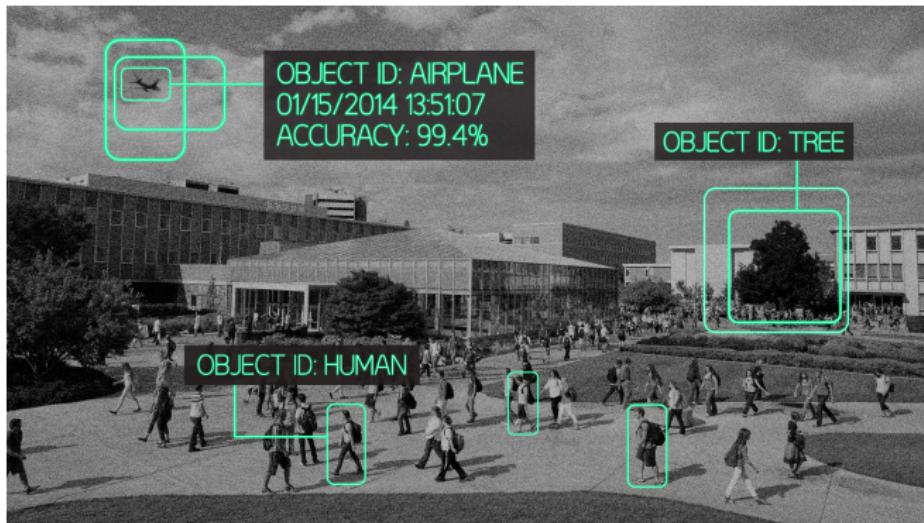
(source: Harvard U)

Human vision



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



(source: BYU Photo)

semantic
segmentation



GRASS, CAT,
TREE, SKY

no objects, just pixels

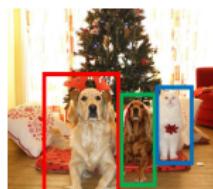
classification +
localization



CAT

single object

object
detection



DOG, DOG, CAT

multiple objects



DOG, DOG, CAT

This image is CC0 public domain

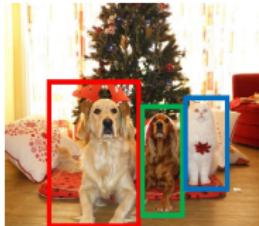
semantic
segmentation



GRASS, CAT,
TREE, SKY

no objects, just pixels

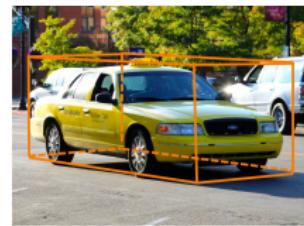
2D object
detection



DOG, DOG, CAT

object categories +
2D bounding boxes

3D object
detection



Car

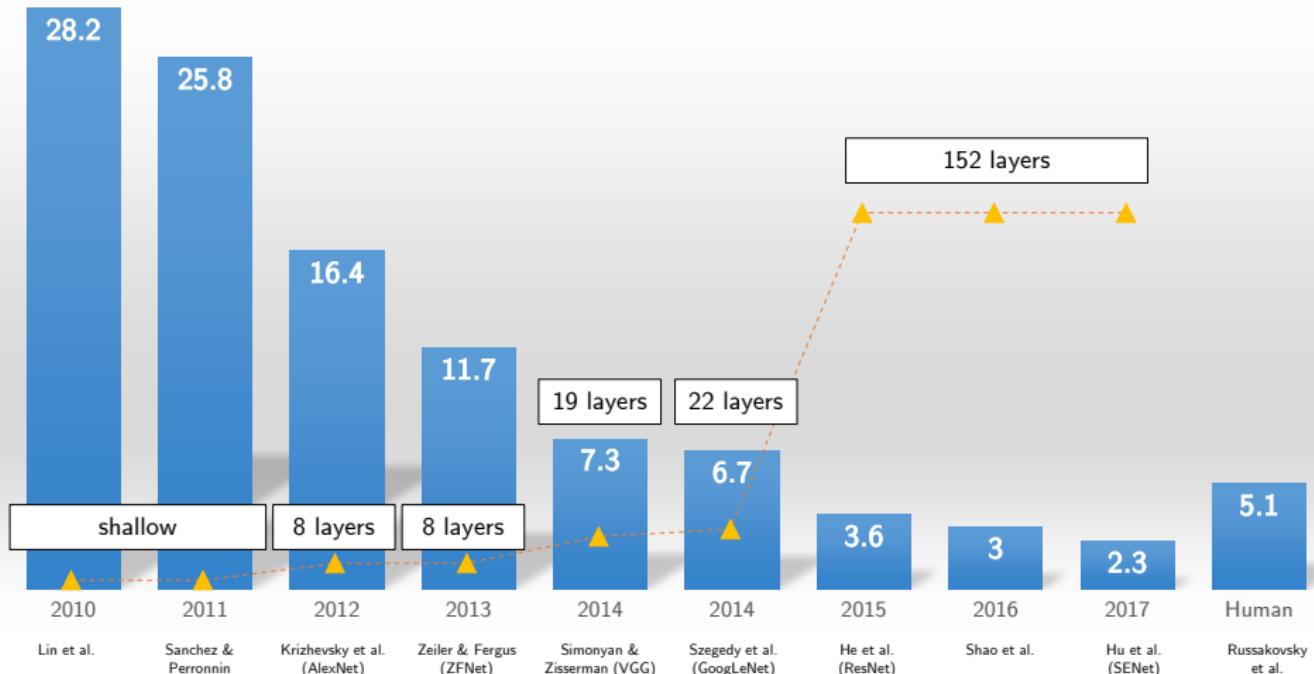
object categories +
3D bounding boxes

(source: cs231n)

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 => filter
 3. pooling/subsampling hidden units

ImageNet challenge winners



Outline

Introduction

Architecture

Convolution Operation

Convolution over Volumes

Additional Topics

Summary

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 1시간 전 위치보다 5분 전 위치가 더 믿을만하다라는 가정에서 출발
 - ▶ 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*

(source: Wikipedia)

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:

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

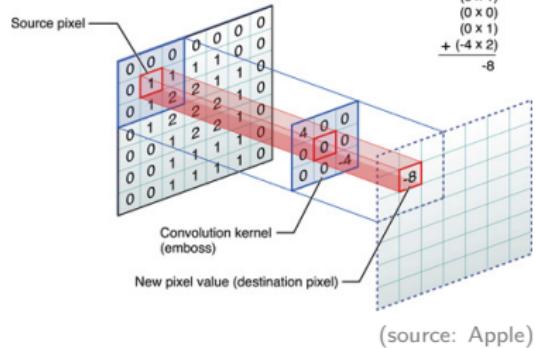
- note

- ▶ convolution: commutative

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

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

$$\begin{array}{r} (4 \times 0) \\ (0 \times 0) \\ (0 \times 0) \\ (0 \times 0) \\ (0 \times 1) \\ (0 \times 0) \\ (0 \times 1) \\ + (-4 \times 2) \\ \hline -8 \end{array}$$



Cross-correlation

- the same as convolution but without flipping the kernel:

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

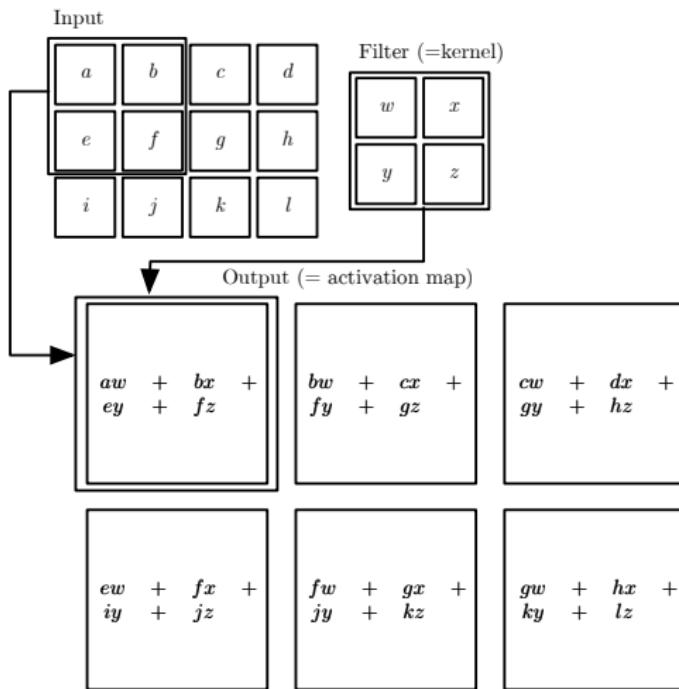
- neural net libraries implement **cross-correlation** but call it **convolution**
 - learned kernel will be in essence the same
- we call both operations convolution
 - specify whether flip the kernel or not if needed

flip: convolution
no flip: cross-correlation

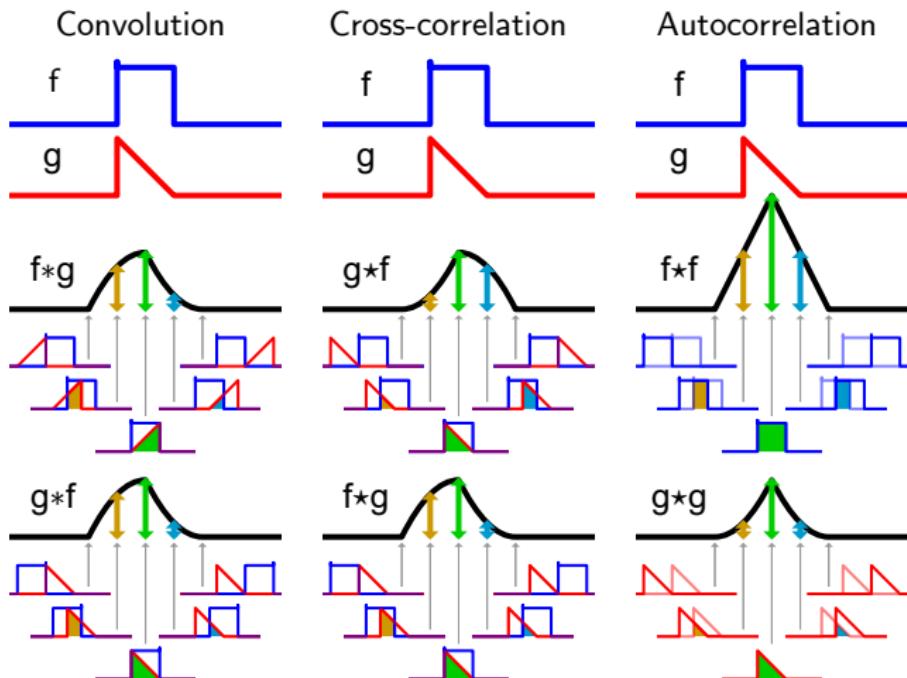
CNN에서는 flip 여부가 크게 중요하지 않아서 보통은 표시를 안함, 꼭 필요한 경우 flip 여부를 표시



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



Comparison



convolution: $f*g = g*f$

cross-correlation: $f*g \neq g*f$
but ML 입장에서 중요한 것은 아님

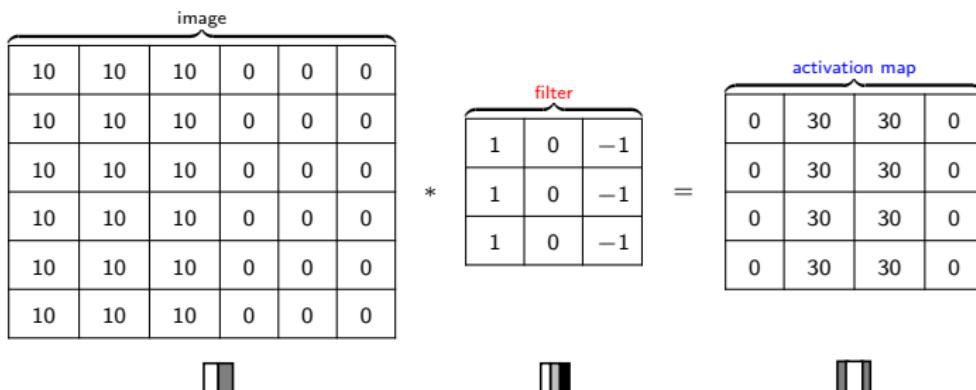
(source: Wikipedia)

Filters

filter => 입력 image의 feature를 detect함

- feature detectors

e.g. vertical edge detection

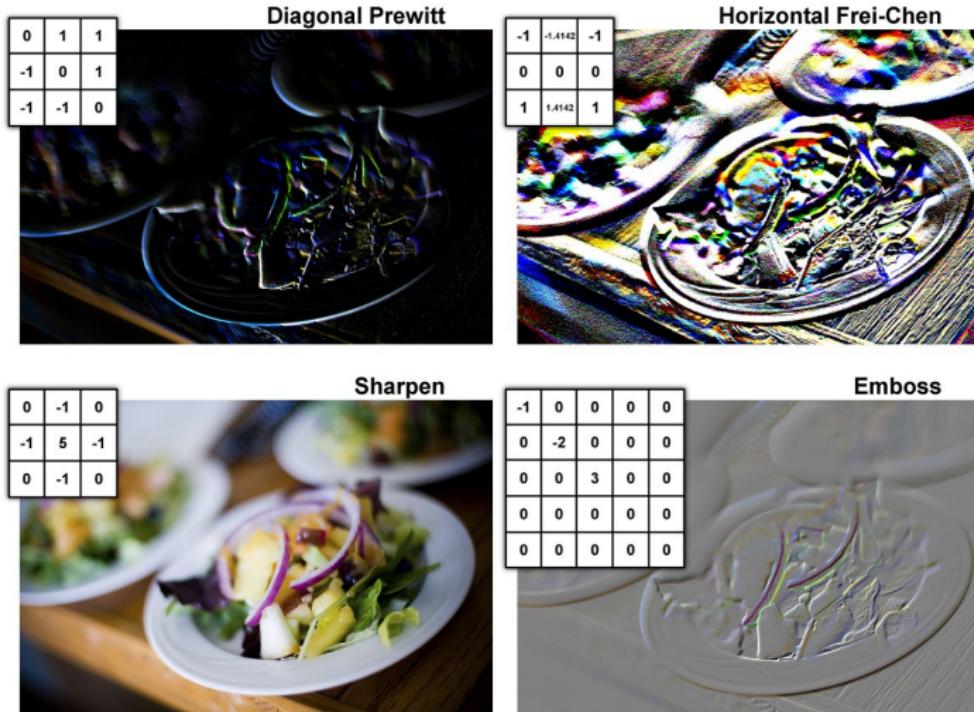


- hand-designed (conventional ML) vs learned (CNN) CNN에서는 filter의 값을 learn함

hand-designed: 전통적인 ML, 직접 filter를 찾음

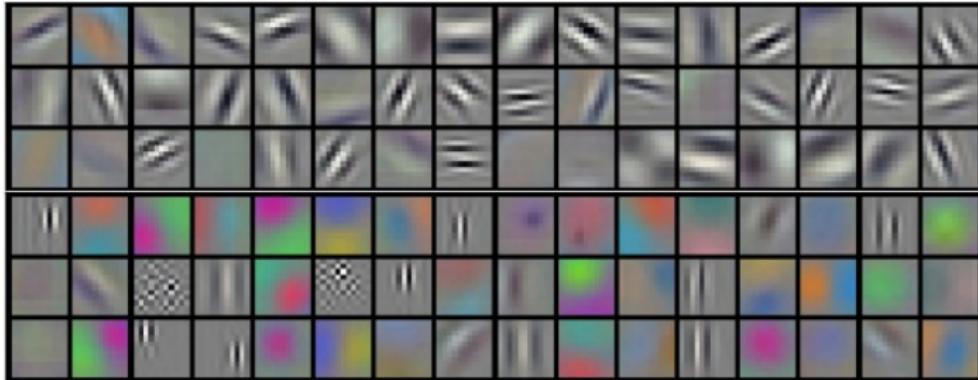
CNN filter: filter를 학습함

- filter examples (hand-designed):



(source: www.gimpible.com)

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



(source: Krizhevsky et al.)

Outline

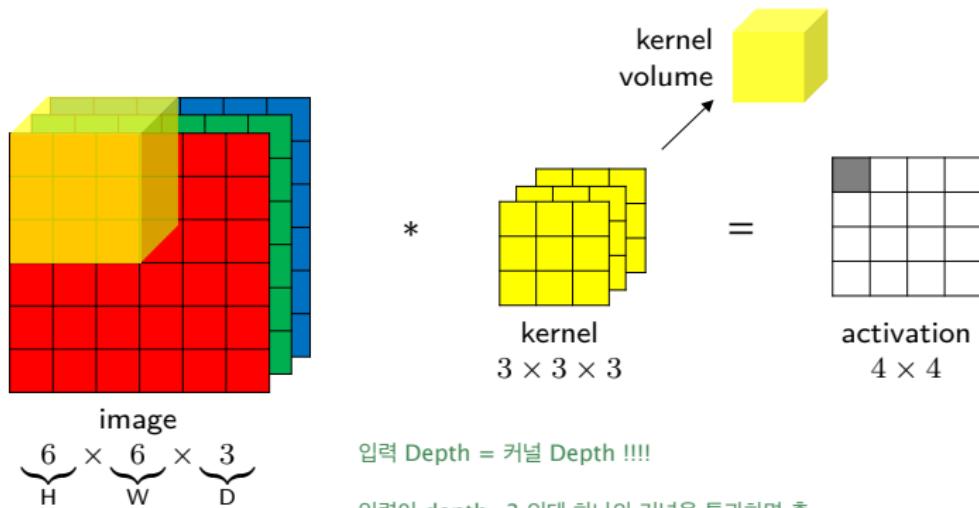
Introduction

Architecture

Convolution Operation
Convolution over Volumes

Additional Topics
Summary

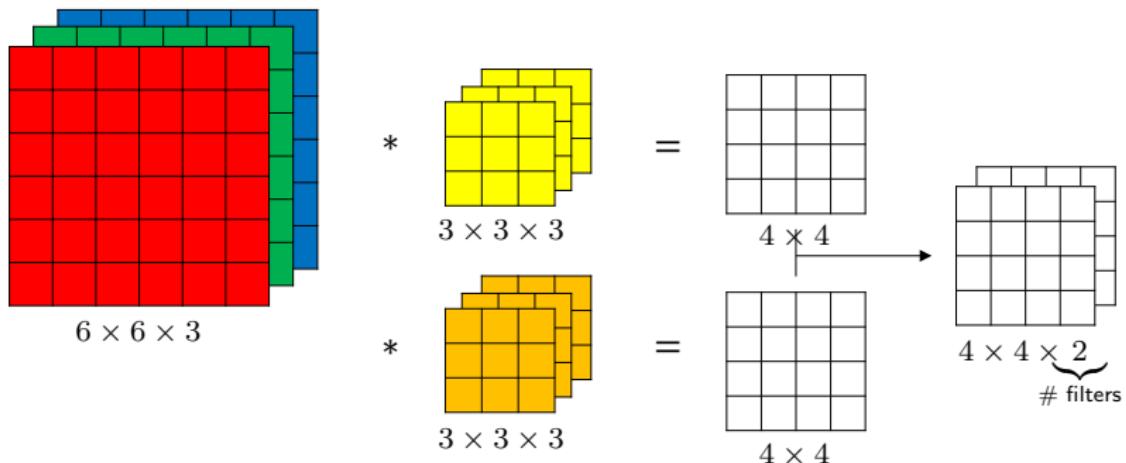
Convolutions over RGB image



filter depth = 입력 depth
filter 갯수 = 출력 depth

(source: Coursera)

Multiple filters



(source: Coursera)

Tensor convolution (textbook notation)

kernel: width, height, input depth, output depth(= # of kernels)

- representations

- ▶ kernel: 4D tensor \mathbf{K}

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

- ▶ input (observed data): 3D tensor \mathbf{V}

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

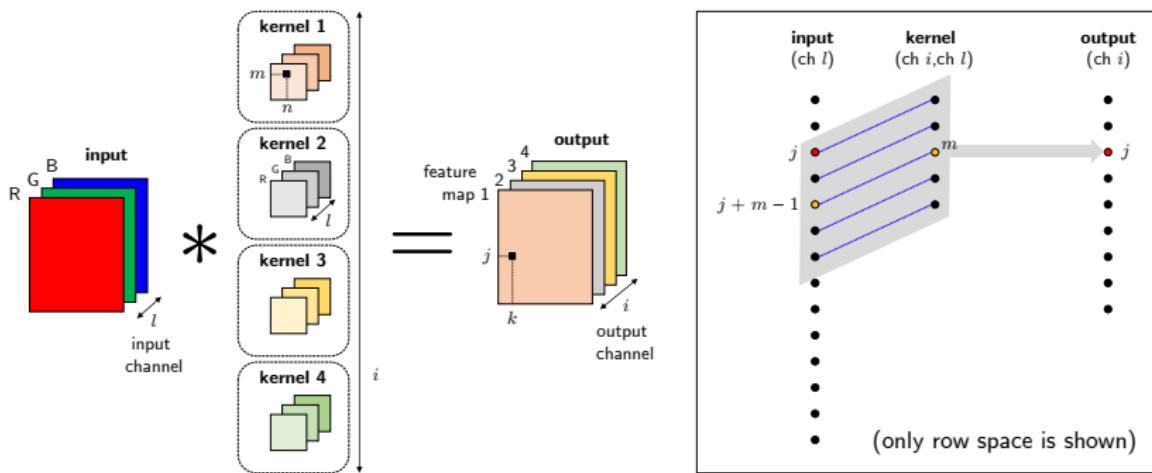
- ▶ output: 3D tensor \mathbf{Z}

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

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

$$Z_{\underbrace{i}_{\text{out ch}}, \underbrace{j}_{\text{row}}, \underbrace{k}_{\text{col}}} = \sum_{l,m,n} V_{\underbrace{l}_{\text{in ch}}, \underbrace{j+m-1}_{\text{row}}, \underbrace{k+n-1}_{\text{col}}} K_{\underbrace{i}_{\text{out ch}}, \underbrace{l}_{\text{in ch}}, \underbrace{m}_{\text{row}}, \underbrace{n}_{\text{col}}}$$

- ▶ the summation² over l, m and n : over valid indices



²in 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

Introduction

Other Layers

Convolution Operation

Additional Topics

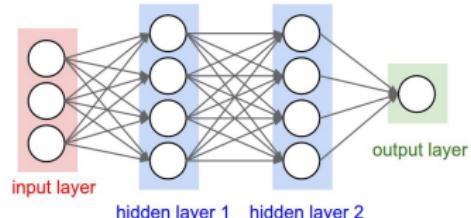
Architecture

Summary

Convolutional Layer

Review

DNN => 현재 hidden layer의 neuron이 이전 layer의 neuron들과 fully connected임

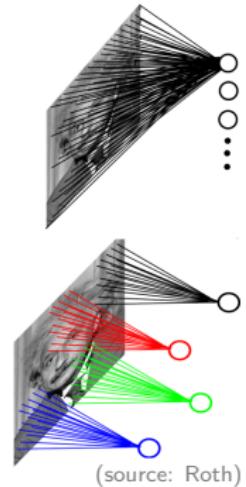


(source: cs231n)

- 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

Motivation

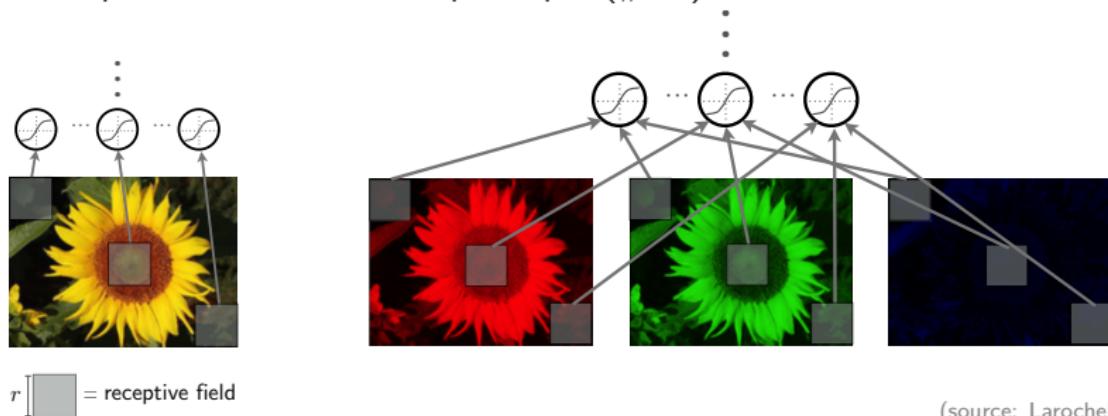
- regular neural nets do not scale well to images
 - ▶ reason: full connectivity
- consider a fully connected neuron in the first hidden layer
 - ▶ to handle a color image of size 1000×1000
 - ⇒ the neuron needs $1000 \times 1000 \times 3 = 3 \times 10^6$ weights
- this full connectivity: wasteful
 - ▶ the huge number of parameters ⇒ quickly lead to overfitting
- CNNs
 - ▶ take advantage of the fact that input consists of images
 - ▶ constrain the architecture in a more sensible way



(source: Roth)

Local connectivity

- connectivity of each neuron in CNN
 - ▶ spatial (height and width) axes: only a local region of the input volume
 - ▷ $\underbrace{\text{filter size}}$ = size of this local region (called receptive field)
↑
hyperparameter
 - ▶ depth axis: the same as input depth (# ch)



example:

filter size: 보통 출수x출수 정사각으로 정함

- 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

CNN 각각의 layer는 3D volume transformer임
=> input으로 3D volume을 받아서 output으로
새로운 3D volume을 내놓음

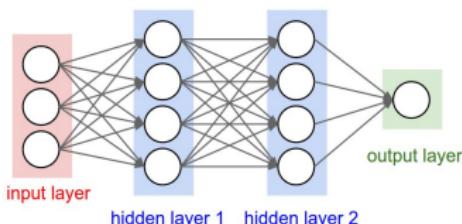
- neurons in a CNN

- ▶ arranged in 3D: height, width, depth³

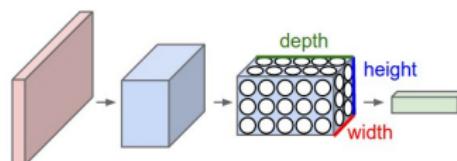
e.g. a CIFAR-10 image \Rightarrow an input volume of $32 \times 32 \times 3$

- every layer of a CNN

- ▶ transforms 3D input volume to 3D output volume of neuron activations



(a) regular 3-layer neural net



(b) CNN

(source: cs231n)

³the 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)

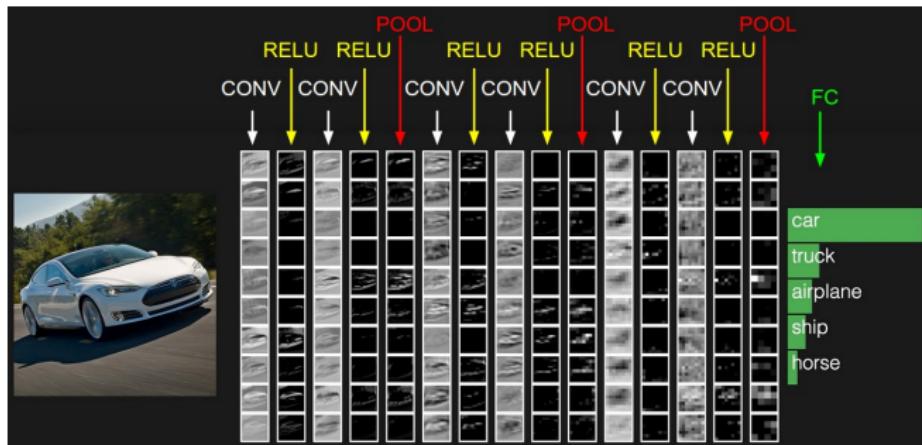
- the neurons in a layer
 - ▶ will only be connected to a small region of the layer before it
(instead of all of the neurons in a fully connected manner)
- the final output layer: a single vector of class scores (arranged along the depth)
 - e.g.* for CIFAR-10: $1 \times 1 \times 10$
 - ▶ CNN reduces the full image into a vector of class scores
- bottom line
 - ▶ a CNN is made up of layers
 - ▶ each layer = a volume transformer
 - ▷ $3D\ volume \xrightarrow{\text{transform}} 3D\ volume$
 - ▷ uses a differentiable function that may or may not have parameters

Building a CNN

convolution layer가 연산량이 많은 이유: FC
layer는 모든 입력 pixel에 대하여 1번의 곱을 수행
하는데 conv layer에서는 1번 이상(중복되는
pixel에 대하여) 실행하므로 계산량이 많음

- four main types of layers \Rightarrow stacking them gives a full CNN architecture

1. convolutional layer \Rightarrow 연산량 많음
2. RELU layer
3. pooling layer
4. fully connected layer \Rightarrow parameter 수 많음



(source: cs231n)

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

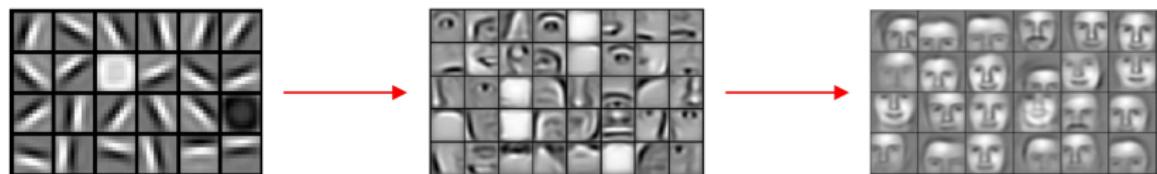
layer	parameters	hyperparameters
CONV	○	○
RELU	✗	✗
POOL	✗	○
FC	○	○

- parameters
 - ▶ trained with gradient descent⁴ (in a supervised fashion)

⁴the 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)



(source: Coursera)

Outline

Introduction

Other Layers

Convolution Operation

Additional Topics

Architecture

Summary

Convolutional Layer

Convolutional layer

연산량은 conv layer에서,
parameter 수는 FC layer에서
대부분을 차지

- the core building block of a CNN

- does most of the computational heavy lifting

$$\text{input volume (3D)} * \text{a filter} = \text{output map (2D)}$$

$$\text{input volume (3D)} * \text{filters} = \text{output volume (3D)}$$

- each filter (3D):

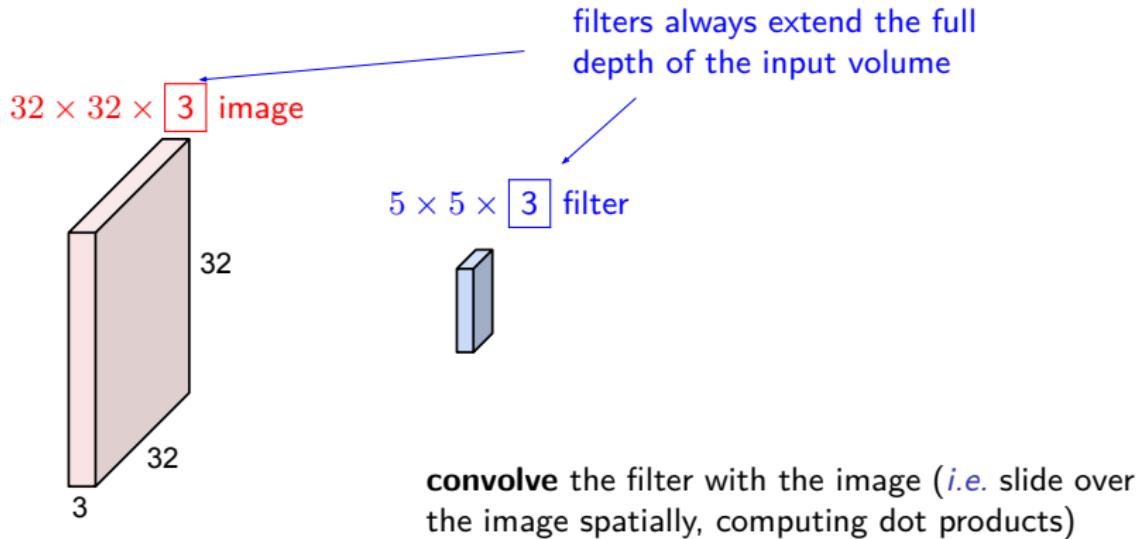
- is small spatially along height and width
 - but extends through the full depths of the input volume

- e.g. a typical filter on a first layer of a CNN

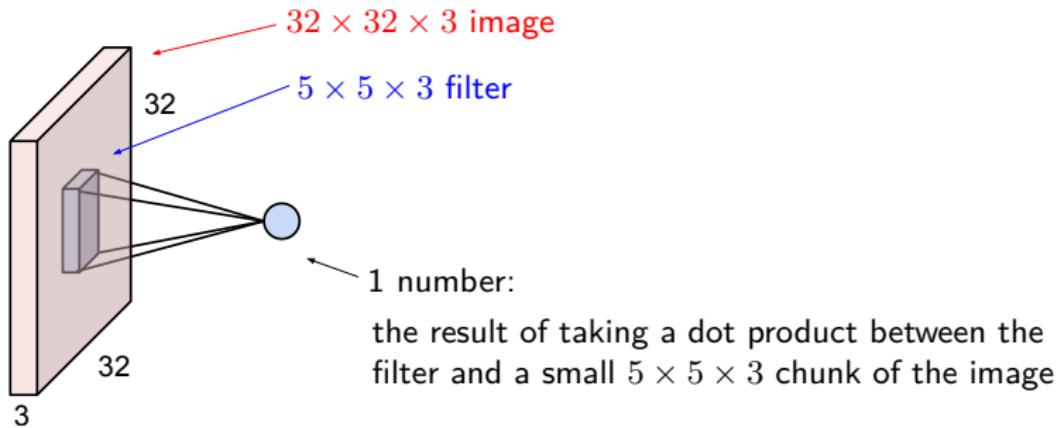
- might have size $5 \times 5 \times 3$ (height \times width \times $\underbrace{\text{depth}}$)

↑
RGB color **channels**

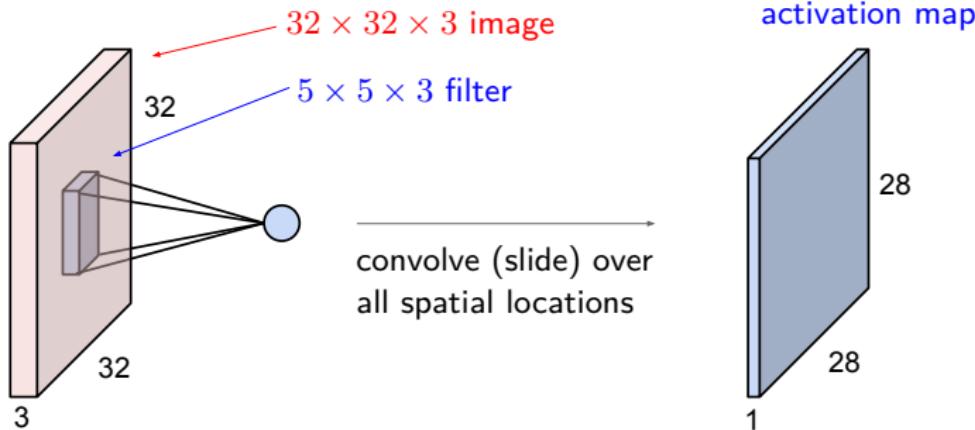
filter depth는 언제나 입력 depth와 같음!!!



(source: cs231n)



(source: cs231n)



(source: cs231n)

Why convolution?

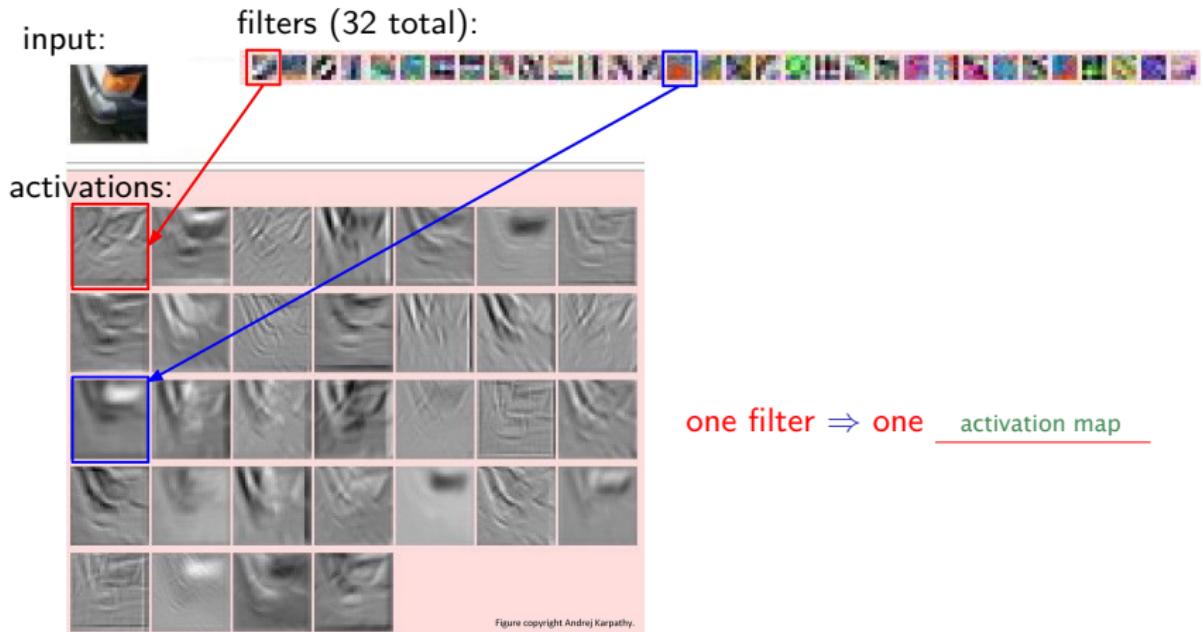
1. parameter sharing⁵
 - ▶ a feature detector (*i.e.* filter) useful in one part of an image
 - ⇒ probably useful in another part of the image
 2. sparse interactions (aka sparse connectivity/weights)
 - ▶ in each layer, each output value depends only on a small # inputs
 3. flexibility
 - ▶ convolution 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 → 267,960 ops (fig 9.6 in textbook)

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

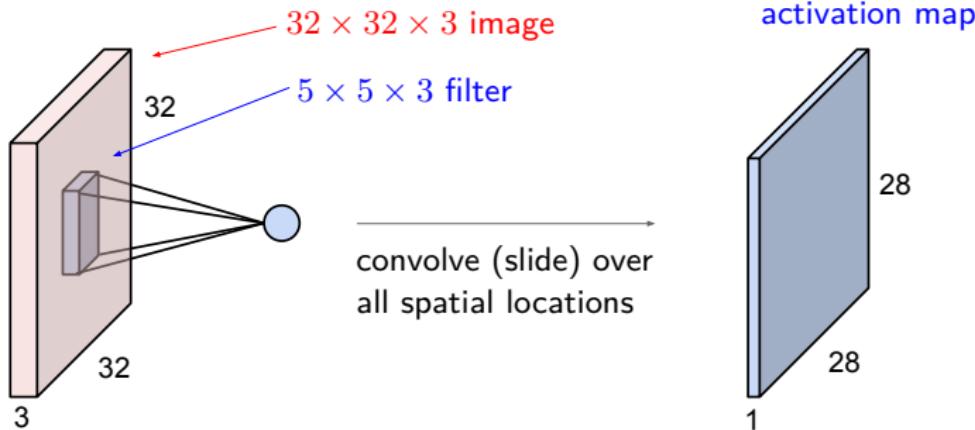
Producing output volume

- each CONV layer
 - ▶ has a set of filters (*e.g.* 12 filters)
 - ↑
each of them will produce a separate 2D activation map
 - ▶ stacks these activation maps along the depth dimension
 - ⇒ produces output volume

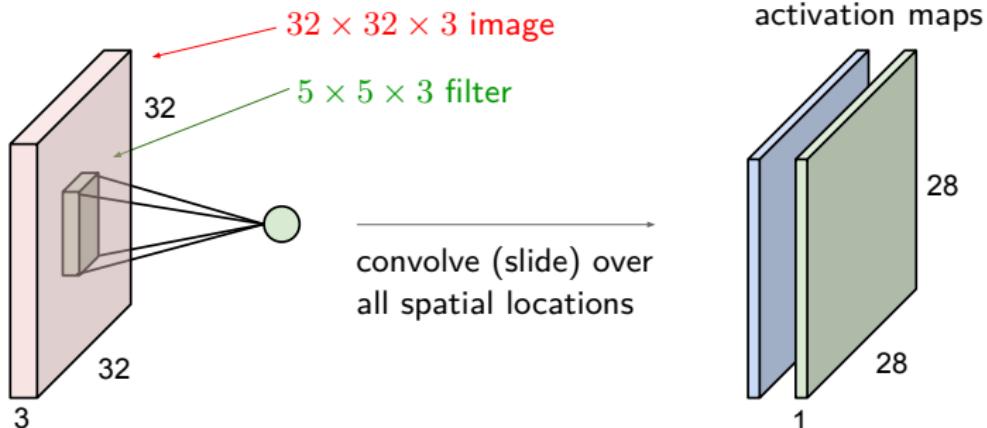
- example: applying 32 filters \Rightarrow output depth = 32



(source: cs231n)

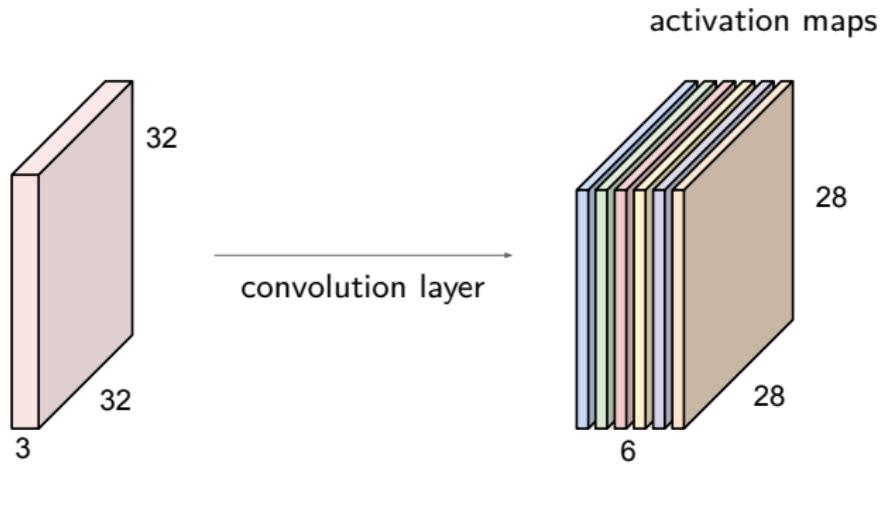


(source: cs231n)



(source: cs231n)

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



(source: cs231n)

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

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)

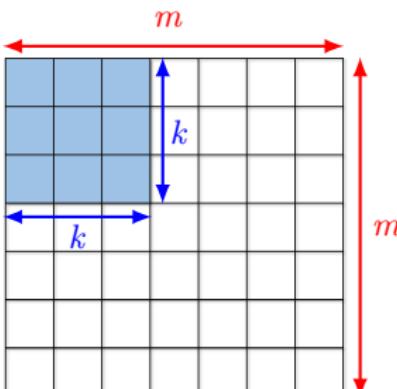
- ▶ filter size: 3×3
- ▶ stride: 1

⇒ 5×5 output

- ▶ filter size: 3×3
- ▶ stride: 2

⇒ 3×3 output

Output size

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


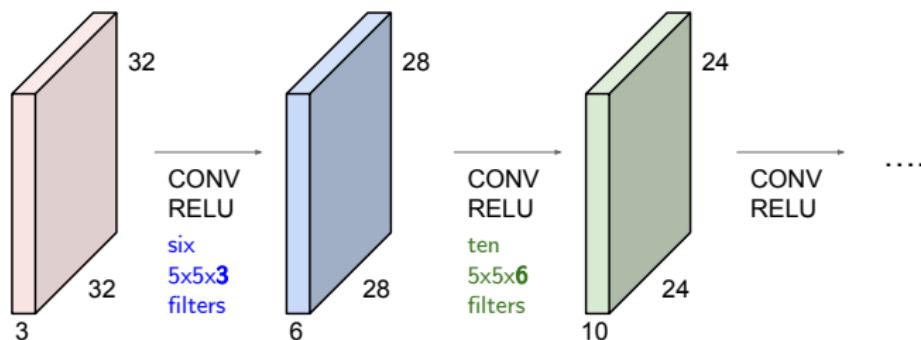
The diagram illustrates the convolution process. A 7x7 input grid (labeled m) is processed by a 3x3 filter (labeled k). The stride is 1. The output size is calculated as $(m-k)/s + 1$, which is 5. The output grid is shown as a 5x5 matrix.

$$(m-k)/s + 1$$
 - 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
 - ⇒ volume shrinks spatially ($32 \rightarrow 28 \rightarrow 24 \dots$)
 - ▶ shrinking too fast: not good (does not work well)

여기서 추가로 외각 pixel은 잘 반영이 안됨(외각 pixel은 계산되는 횟수가 적으므로)



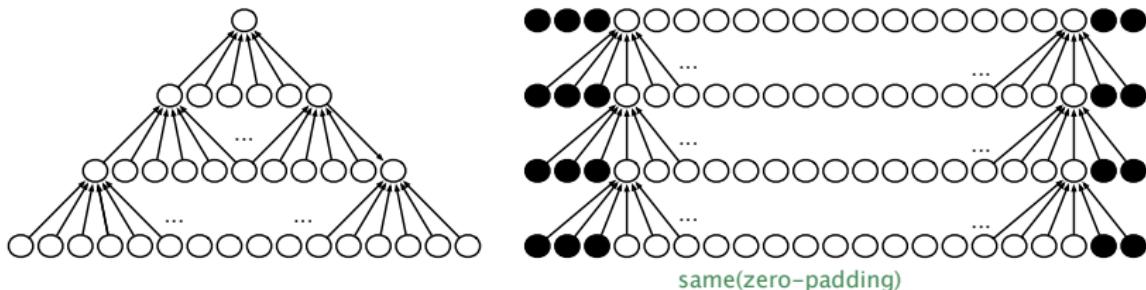
(source: cs231n)

- pixels along the edges: less used for convolutions ⇒ potential information loss
=> under representation 됨

Zero-padding

=> kernel size와 output size를 독립적으로 바꿀 수 있다. 왜냐하면 위의 2가지를 맘대로 정한 후에 stride와 padding을 그에 맞게 바꿔줄 수 있으므로

- 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
 - ⇒ both significantly limit expressive power of the net



- also helpful for preserving information along the edges

Example

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

- 7×7 input, 3×3 filter, stride 1
 - ▶ zero-pad with 1 pixel border
 - ▶ output size: $(7 + 2 - 3)/1 + 1 = 7$
 - ⇒ input size preserved after convolution
 - in general, to preserve spatial size
 - ▶ zero-pad with $\boxed{(k-1) / 2}$
- e.g. $k = 3 \Rightarrow$ zero pad with 1
 $k = 5 \Rightarrow$ zero pad with 2

Three types of zero-padding schemes

kernel size는 보통 출수로 사용!!!

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

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

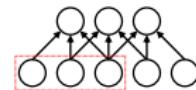
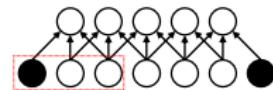
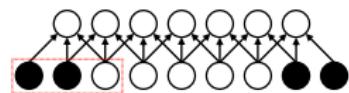
full: 모든 pixel이 완벽하게 동일한 기회를 얻음

same: input과 output의 spatial size가 같음

optimal zero padding (in terms of test accuracy)

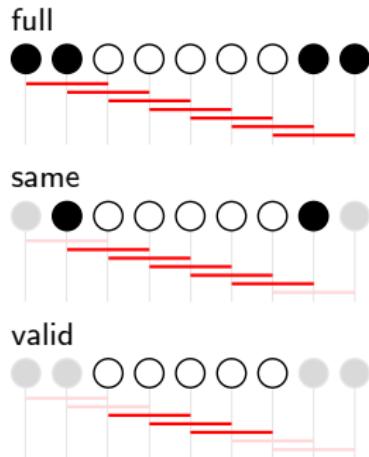
- ▶ usually lies somewhere between “valid” and “same” convolution

* value of k : typically odd

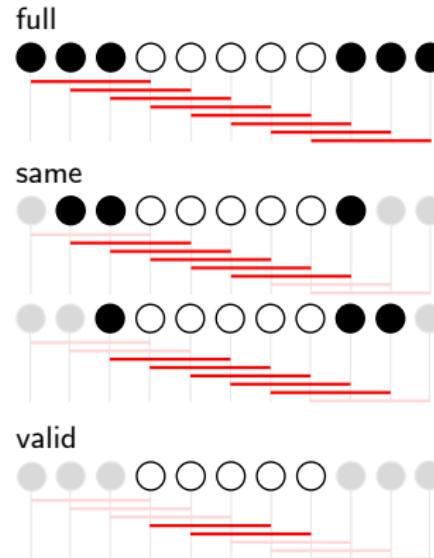


examples:

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



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



Summary: activation map size

conv layer의 출력 depth는 # of filter에 의해 결정됨!!!

- 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

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

- 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)

- ▶ 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

Introduction

Other Layers

Convolution Operation

Additional Topics

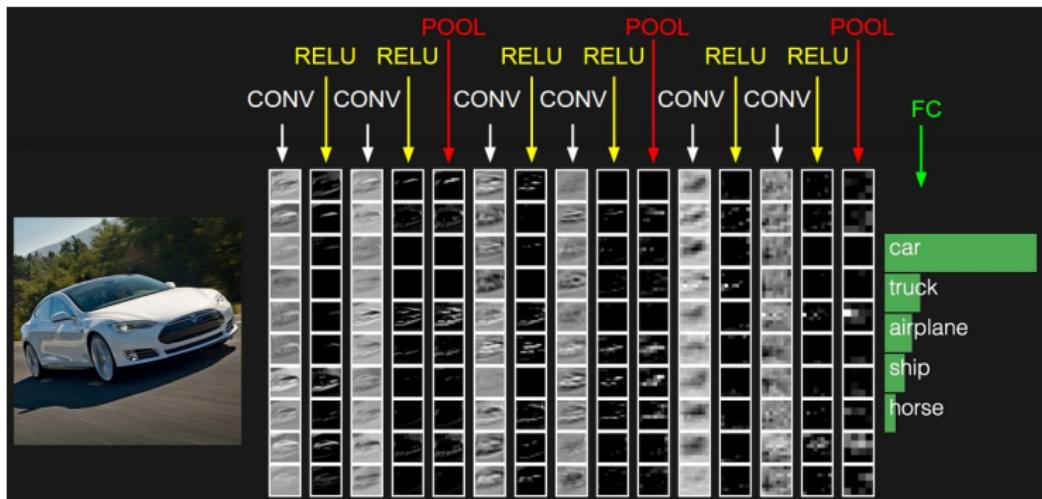
Architecture

Summary

Convolutional Layer

Other layers

- three more layers to go:
 - ▶ RELU layer
 - ▶ pooling layer => down sampling(size를 줄여줌)
 - ▶ fully connected layer

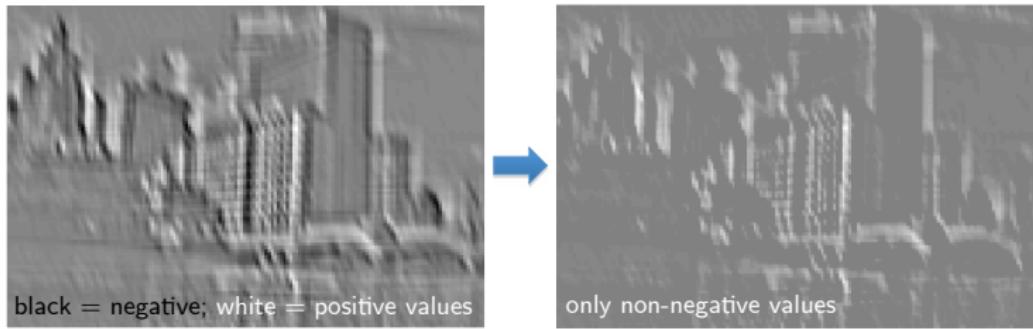


(source: cs231n)

ReLU layer

ReLU => parameter, hyperparameter 둘 다 없음!!!

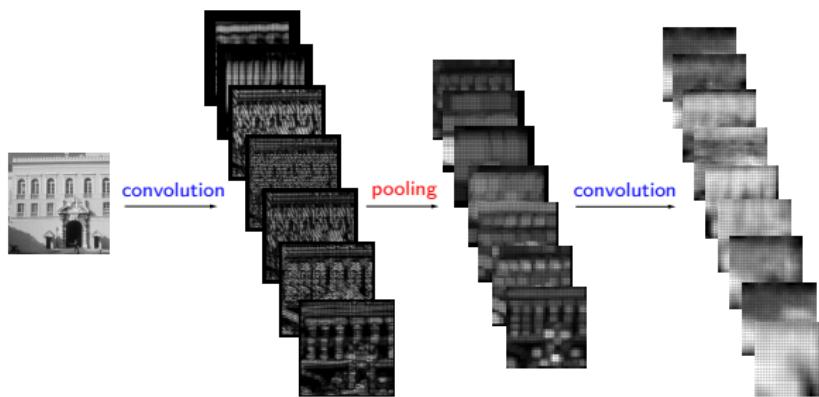
- purpose
 - ▶ increase the non-linear properties (of decision function and of overall net)
- facts
 - ▶ often called detector (or nonlinear) stage
 - ▶ introduces no (hyper)parameters



(source: Fergus)

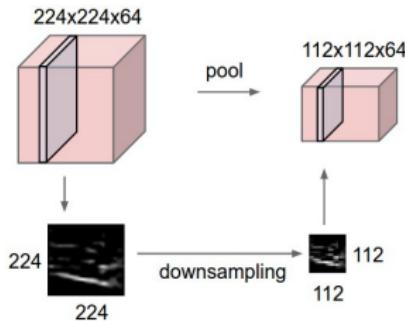
Pooling layer

- common practice: 주로 conv layer와 conv layer 사이에 사용
 - ▶ periodically insert a pooling layer in-between successive conv layers
- the function of pooling layers: width, height만 줄이고 depth는 유지!!!
 - ▶ progressively reduce the spatial size of the representation
 - ⇒ reduce the amount of parameters and computation + control overfitting



(source: Thériault, 2012)

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



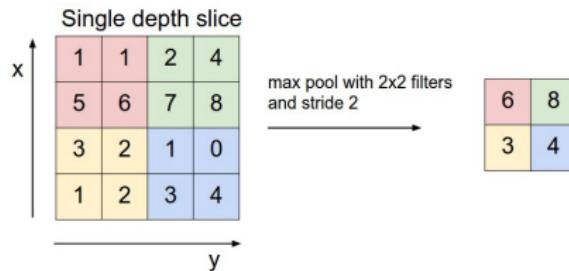
(source: cs231n)

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

Max pooling

max pooling을 가장 많이 씀

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



(source: cs231n)

- other types of pooling:
 $\underbrace{\text{average pooling}, L^2\text{-norm pooling}, \dots}$
often used historically but phased out by max pooling

Eliminating pooling

- some researchers propose 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

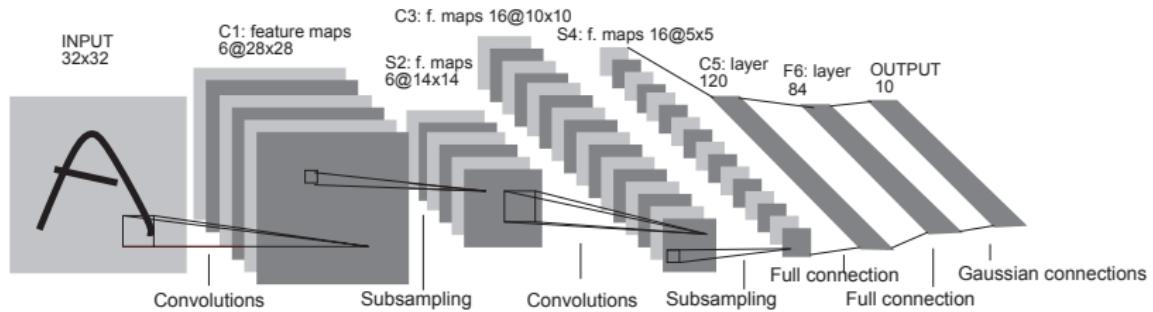
= MLP layer

- 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

FC layer 이후에 conv layer 불가능!!!

Putting it all together

- Lenet-5 for digits recognition:
 - layer 1 → layer l : input size ↓ but # filters ↑ (common trend in CNNs)



(source: LeCun, 1998)

- demo

- ConvNetJS: training on CIFAR-10 [Link](#)



Output
Layer

FC
Layer 2

FC
Layer 1

Pooling
Layer 2

Convolution
Layer 2

Pooling
Layer 1

Convolution
Layer 1

Input Layer

0123456789



(source: Harley, 2015)

Outline

Introduction

Convolution Operation

Architecture

Additional Topics

More on Convolution

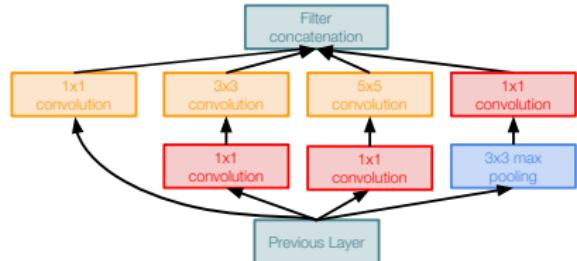
Other Issues

Summary

1×1 convolution

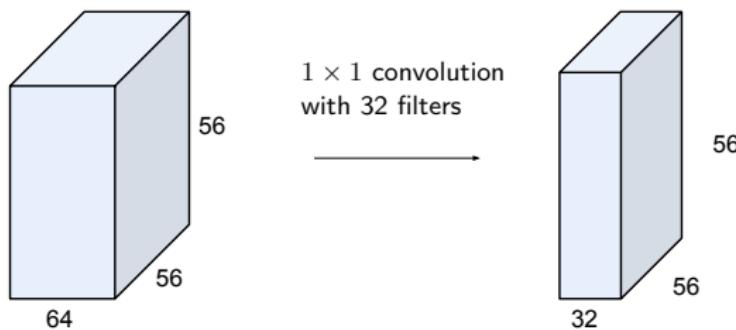
1×1 convolution => depth를 줄이는 역할

- widely used for depth adjustment
e.g. inception module in GoogLeNet



(source: Szegedy et al., 2014)

- example:



(source: cs231n)

- each filter: size $1 \times 1 \times 64$ (performs a 64-dim dot product)

Dilated convolution (atrous convolution)

=> 빵꾸뚱린 convolution

- introduces another convolution parameter: dilation rate
 - ▶ defines a spacing between values in a filter
- e.g. 3×3 filter with dilation rate 2
 - ⇒ the same field of view as 5×5 filter (but only uses 9 parameters)
- effect: a wider field of view at the same computational cost
 - ▶ real-time segmentation, speech synthesis
- example
 - ▶ 3×3 filter
 - ▶ dilation rate of 2
 - ▶ no zero-padding

(source: Theano tutorial)

Transposed convolution (fractionally strided convolution)

- performs a regular convolution but reverts spatial transformation
 - *i.e.* upsampling : smaller map \rightarrow larger map
 - ▶ does not perform the deconvolution (*i.e.* inverse of convolution)⁶
 - ▶ needs some fancy padding on the input
- example
 - ▶ 3×3 filter
 - ▶ 2×2 input $\rightarrow 5 \times 5$ output

(source: Theano tutorial)

⁶thus, transposed convolution is sometimes (inappropriately) called *deconvolution* but is different from the deconvolution in engineering and mathematics

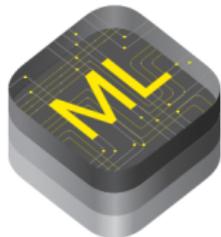
Convolution by frequency domain conversion

- convolution: equivalent to the following
 1. convert both input/kernel to frequency domain using Fourier transform
 2. perform point-wise multiplication of the two signals
 3. convert back to time domain using an inverse Fourier transform
- for some problem sizes
 - ▶ this can be faster than the naïve implementation of discrete convolution

convolution을 빠르게 하는 것이 관건 => on device를 위해

Remarks

- active areas of research
 - ▶ devising faster ways of performing convolution
 - ▶ approximate convolution without harming accuracy of the model
 - ▶ fast evaluation of forward propagation
 - in commercial setting
 - ▶ typically devote more resources to deployment of a net than its training
 - ⇒ techniques that improve efficiency of only forward prop are useful
- e.g. TensorRT, TensorFlowLite, Core ML, Caffe2Go



Outline

Introduction

Convolution Operation

Architecture

Additional Topics

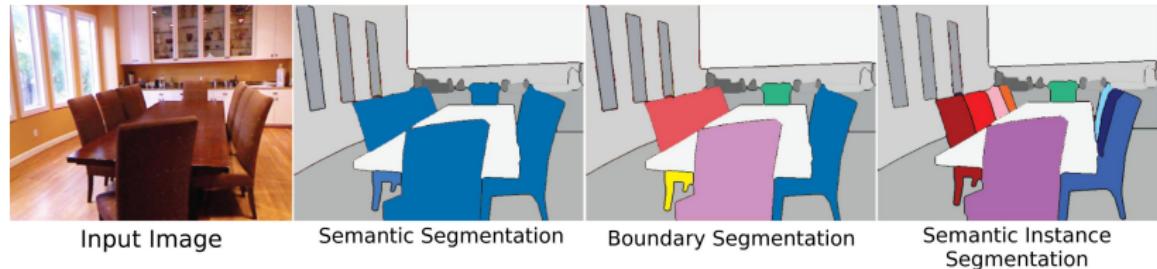
More on Convolution

Other Issues

Summary

Structured outputs

- CNNs can output a high-dimensional, structured object
 - ▶ rather than predicting class label (classification) or real value (regression)
 - ▶ typically this object: emitted by a standard convolutional layer
- e.g. tensor $S_{i,j,k}$: probability that input pixel (j, k) belongs to class i
 - ▶ this allows the model to
 - ▷ label every pixel in an image and draw precise object masks
 - ▶ this pixel labeling problem: our running example in this section



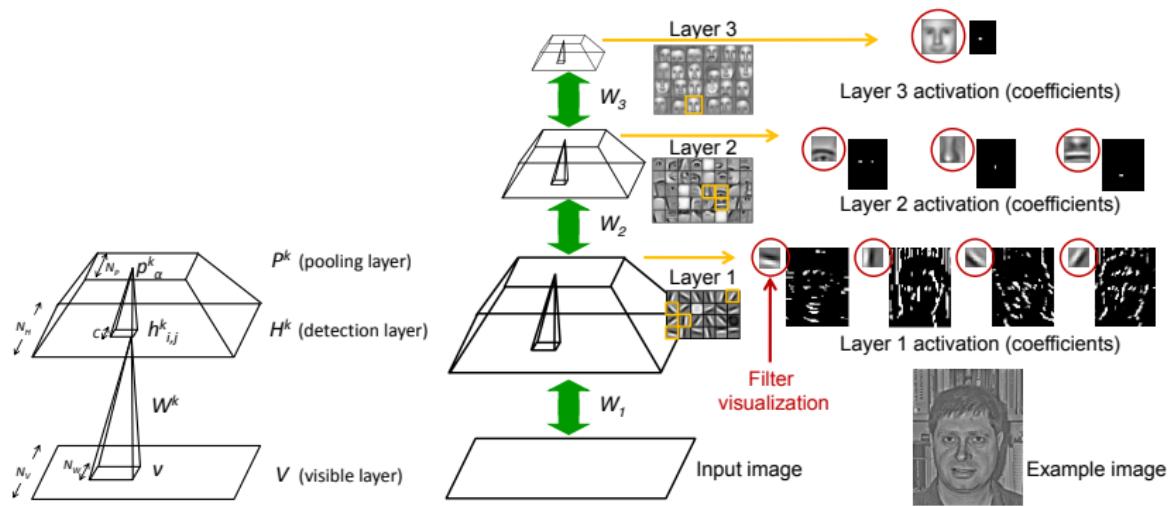
(source: Siberman et. al, 2012)

Unsupervised kernel learning

- three basic strategies
 1. simply initialize them randomly
 2. design them by hand
 3. learn kernels with an unsupervised criterion
- example for #2:
 - ▶ set each kernel to detect edges at a certain orientation or scale
- example for #3:
 - ▶ Coates *et al.* (2011)
 - ▷ apply k -means clustering to small image patches
 - ▷ then use each learned centroid as a convolution kernel
 - ▶ part III of textbook: more unsupervised learning approaches

An intermediate approach

- learn features but without full forward/backward prop at every gradient step
 - ▶ idea: greedy layer-wise pretraining (as in restricted Boltzmann machine)
- e.g. convolutional deep belief network (Lee et al., 2009)



(source: Lee et. al, 2009)

- for CNN, we can take pretraining strategy one step further than MLP
 - *i.e.* we can use train a CNN *without ever using convolution during training*
- how? instead of training an entire conv layer at a time,
 - train a model of a small patch (*e.g.* using k -means)
 - then use the parameters from this patch-based model to define kernels
- this approach⁷ can train very large models
 - incurs a high computational cost only at inference time
 - popular from ~2007–2013 (small labeled data/limited computing power)
- today: most CNNs are trained in a **purely supervised fashion**
 - using full forward/back-prop through entire net on each training iteration

⁷Ranzato *et al.*, 2007b; Jarrett *et al.*, 2009; Kavukcuoglu *et al.*, 2010; Coates *et al.*, 2013

Outline

Introduction

Additional Topics

Convolution Operation

Summary

Architecture

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