M2177.003100
Deep Learning

[7: Convolutional Neural Nets (Part 2)]

Electrical and Computer Engineering
Seoul National University

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

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
  - *Dive into Deep Learning (14.10 Transposed Convolution)* ▸ Link
  - *Convolution Arithmetic* ▸ Link 1 ▸ Link 2 ▸ arXiv
  - *Kunlun Bai's Blog on Convolution Types* ▸ Link
- note:
  - you should open this file in Adobe Acrobat to see animated images (other types of pdf readers will not work)

Outline

Recall: convolution

- with kernel flipping
  - commutative

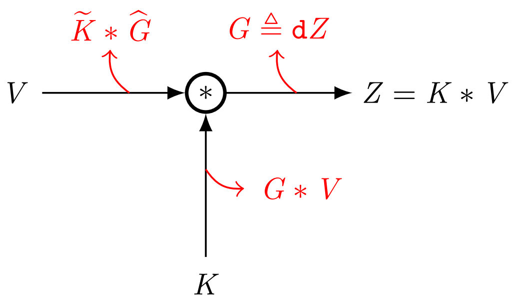
$$
\begin{aligned}
Z(i,j) = (K * V)(i,j) &= \sum_m \sum_n \underbrace{K(m,n)}_{\text{kernel}} \underbrace{V(i-m, j-n)}_{\text{input volume}} \\
&= \sum_m \sum_n K(i-m, j-n) V(m,n) \\
&= (V * K)(i,j)
\end{aligned}
$$

- without kernel flipping
  - not commutative

$$
Z(i,j) = (K * V)(i,j) = \sum_m \sum_n \underbrace{K(m,n)}_{\text{kernel}} \underbrace{V(i+m, j+n)}_{\text{input volume}}
$$

  - we stick to this definition of convolution

Preview: backprop over convolution



- forward

$$K * V = Z$$
$$C\mathbf{v} = \mathbf{z}$$

- backward[1]

$$\mathtt{d}V = \widetilde{K} * \widehat{G}$$
$$\mathbf{dv} = C^\top \mathbf{g}$$
$$\mathtt{d}K = G * V$$

$C = K*$

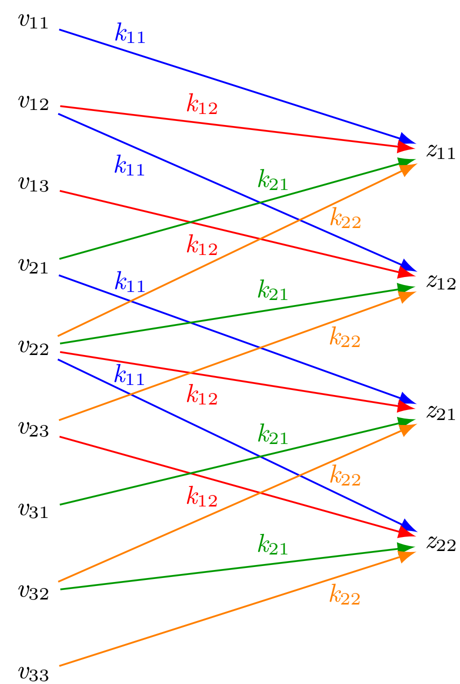
- $\mathbf{v}, \mathbf{z}, \mathbf{g}$: flattened versions of V, Z, G, respectively
- C : matrix representation of "convolution with K" $C = \quad [K * (\ )]$
  - $C^\top$: leads to transposed convolution $[\widetilde{K} * \widehat{(\ )}]$
- $\widetilde{K}$: flipped version of kernel K
- $\widehat{G}$: zero-padded version of G

[1] non-commutativity of "convolution without kernel flipping" $\Rightarrow$ difference in operations to compute $\mathtt{d}V$ and $\mathtt{d}K$

Running example

- $K * V = Z$
  - (k, m, s, p)[2] $= (2, 3, 1, 0)$

$$\underbrace{\begin{array}{|c|c|}\hline k_{11} & k_{12} \\ \hline k_{21} & k_{22} \\ \hline \end{array}}_{K} * \underbrace{\begin{array}{|c|c|c|}\hline v_{11} & v_{12} & v_{13} \\ \hline v_{21} & v_{22} & v_{23} \\ \hline v_{31} & v_{32} & v_{33} \\ \hline \end{array}}_{V} = \underbrace{\begin{array}{|c|c|}\hline z_{11} & z_{12} \\ \hline z_{21} & z_{22} \\ \hline \end{array}}_{Z}$$



  - result

$$z_{11} = k_{11} v_{11} + k_{12} v_{12} + k_{21} v_{21} + k_{22} v_{22}$$
$$z_{12} = k_{11} v_{12} + k_{12} v_{13} + k_{21} v_{22} + k_{22} v_{23}$$
$$z_{21} = k_{11} v_{21} + k_{12} v_{22} + k_{21} v_{31} + k_{22} v_{32}$$
$$z_{22} = k_{11} v_{22} + k_{12} v_{23} + k_{21} v_{32} + k_{22} v_{33}$$

[2] sizes of kernel, input, stride, and padding, respectively

Convolution as a matrix operation

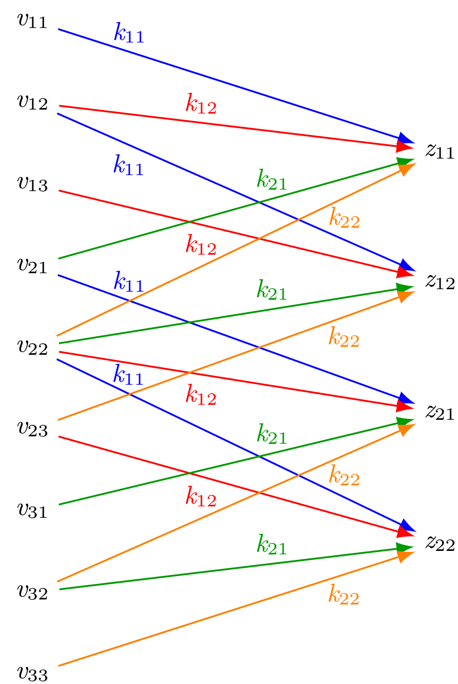
$$Cv = \mathbf{z}$$

$$k_{11}v_{11} + k_{12}v_{12} + k_{21}v_{21} + k_{22}v_{22} = z_{11}$$
$$k_{11}v_{12} + k_{12}v_{13} + k_{21}v_{22} + k_{22}v_{23} = z_{12}$$
$$k_{11}v_{21} + k_{12}v_{22} + k_{21}v_{31} + k_{22}v_{32} = z_{21}$$
$$k_{11}v_{22} + k_{12}v_{23} + k_{21}v_{32} + k_{22}v_{33} = z_{22}$$

where

$$\mathbf{v} = (v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, v_{23}, v_{31}, v_{32}, v_{33})$$
$$\mathbf{z} = (z_{11}, z_{12}, z_{21}, z_{22})$$
C = a (sparse) matrix

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \\ v_{21} \\ v_{22} \\ v_{23} \\ v_{31} \\ v_{32} \\ v_{33} \end{bmatrix}}_{\mathbf{v}} = \underbrace{\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix}}_{\mathbf{z}}$$



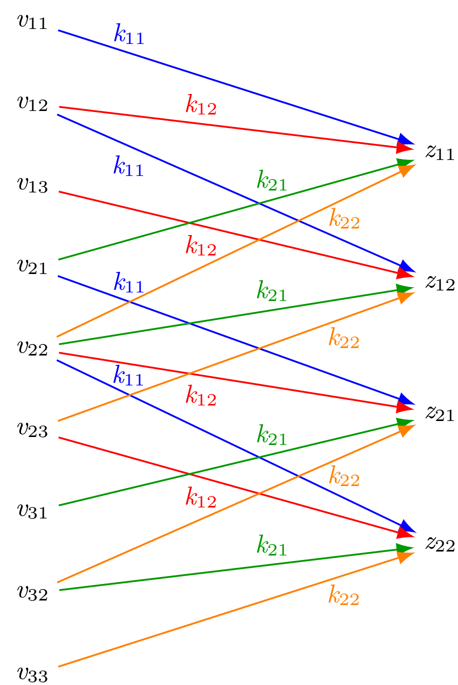
- simplified representation

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \\ v_{21} \\ v_{22} \\ v_{23} \\ v_{31} \\ v_{32} \\ v_{33} \end{bmatrix}}_{\mathbf{v}} = \underbrace{\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix}}_{\mathbf{z}}$$

$$k_{11} v_{11} + k_{12} v_{12} + k_{21} v_{21} + k_{22} v_{22} = z_{11}$$
$$k_{11} v_{12} + k_{12} v_{13} + k_{21} v_{22} + k_{22} v_{23} = z_{12}$$
$$k_{11} v_{21} + k_{12} v_{22} + k_{21} v_{31} + k_{22} v_{32} = z_{21}$$
$$k_{11} v_{22} + k_{12} v_{23} + k_{21} v_{32} + k_{22} v_{33} = z_{22}$$

$$\Downarrow$$

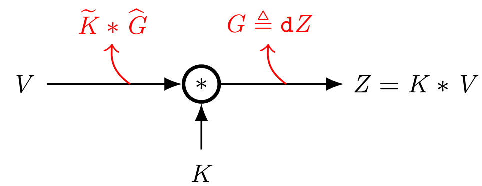
$$\begin{array}{c} \overbrace{\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{21} & v_{22} & v_{23} & v_{31} & v_{32} & v_{33} \end{bmatrix}}^{\mathbf{v}} \\ \underbrace{\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix}}_{C} \end{array} = \underbrace{\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix}}_{\mathbf{z}}$$



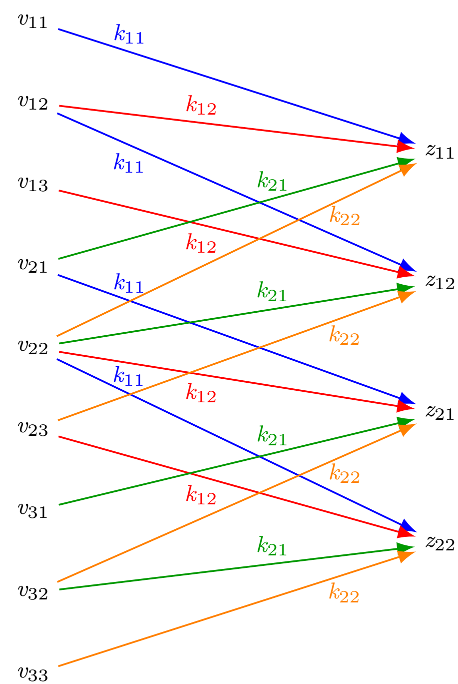
$\mathrm{d}V = \widetilde{K} * \widehat{G}$

fanout → sum

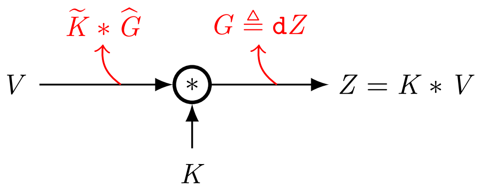
$$\mathrm{d}v_{11} = k_{11} g_{11}$$
$$\mathrm{d}v_{12} = k_{12} g_{11} + k_{11} g_{12}$$
$$\mathrm{d}v_{13} = k_{12} g_{12}$$
$$\mathrm{d}v_{21} = k_{21} g_{11} + k_{11} g_{21}$$
$$\mathrm{d}v_{22} = k_{22} g_{11} + k_{21} g_{12} + k_{12} g_{21} + k_{11} g_{22}$$
$$\mathrm{d}v_{23} = k_{22} g_{12} + k_{12} g_{22}$$
$$\mathrm{d}v_{31} = k_{21} g_{21}$$
$$\mathrm{d}v_{32} = k_{22} g_{21} + k_{21} g_{22}$$
$$\mathrm{d}v_{33} = k_{22} g_{22}$$



$$\underbrace{\begin{array}{|c|c|c|}\hline \mathrm{d}v_{11} & \mathrm{d}v_{12} & \mathrm{d}v_{13} \\ \hline \mathrm{d}v_{21} & \mathrm{d}v_{22} & \mathrm{d}v_{23} \\ \hline \mathrm{d}v_{31} & \mathrm{d}v_{32} & \mathrm{d}v_{33} \\ \hline \end{array}}_{\mathrm{d}V} = \underbrace{\begin{array}{|c|c|}\hline k_{22} & k_{21} \\ \hline k_{12} & k_{11} \\ \hline \end{array}}_{\widetilde{K}\text{: flipped } K} * \underbrace{\begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 0 & g_{11} & g_{12} & 0 \\ \hline 0 & g_{21} & g_{22} & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}}_{\widehat{G}\text{: padded } G = \mathrm{d}Z}$$



- matrix representation (reveals "transposed convolution")



$\boxed{\text{backward}}$ ($C^\top$: dim expander)

$$\mathrm{d}V = \widetilde{K} * \widehat{G}$$

$$\mathbf{dv} = C^\top \mathbf{g}$$

$\boxed{\text{forward}}$ (C: dim reducer)

$$K * V = Z$$

$$C\mathbf{v} = \mathbf{z}$$

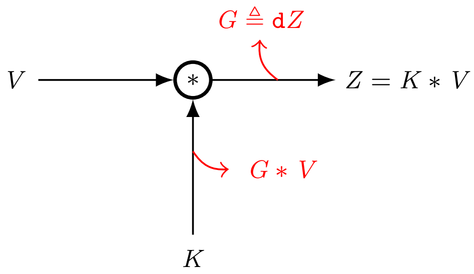
$$\overbrace{\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{21} & v_{22} & v_{23} & v_{31} & v_{32} & v_{33} \end{bmatrix}}^{\mathbf{v}}$$

$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix}}_{\mathbf{z}}$$

$$\overbrace{\begin{bmatrix} g_{11} & g_{12} & g_{21} & g_{22} \end{bmatrix}}^{\mathbf{g}}$$

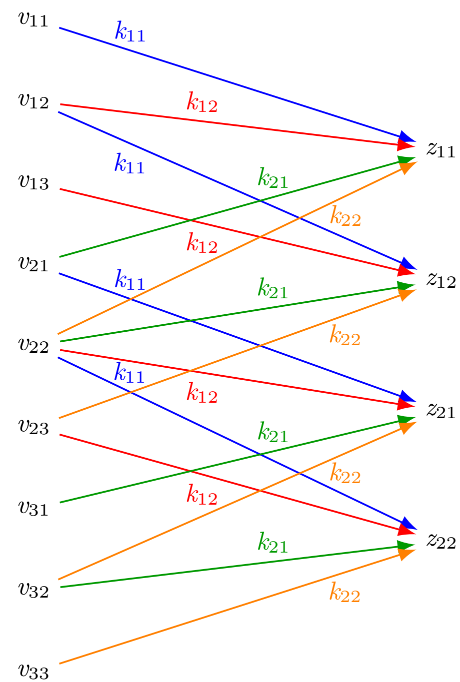
$$\underbrace{\begin{bmatrix} \mathrm{d}v_{11} \\ \mathrm{d}v_{12} \\ \mathrm{d}v_{13} \\ \mathrm{d}v_{21} \\ \mathrm{d}v_{22} \\ \mathrm{d}v_{23} \\ \mathrm{d}v_{31} \\ \mathrm{d}v_{32} \\ \mathrm{d}v_{33} \end{bmatrix}}_{\mathbf{dv}} = \underbrace{\begin{bmatrix} k_{11} & 0 & 0 & 0 \\ k_{12} & k_{11} & 0 & 0 \\ 0 & k_{12} & 0 & 0 \\ k_{21} & 0 & k_{11} & 0 \\ k_{22} & k_{21} & k_{12} & k_{11} \\ 0 & k_{22} & 0 & k_{12} \\ 0 & 0 & k_{21} & 0 \\ 0 & 0 & k_{22} & k_{21} \\ 0 & 0 & 0 & k_{22} \end{bmatrix}}_{C^\top}$$

$\mathtt{d}K = G * V$

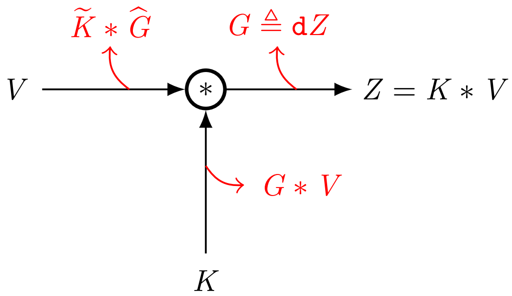


$$\mathtt{d}k_{11} = g_{11}v_{11} + g_{12}v_{12} + g_{21}v_{21} + g_{22}v_{22}$$
$$\mathtt{d}k_{12} = g_{11}v_{12} + g_{12}v_{13} + g_{21}v_{22} + g_{22}v_{23}$$
$$\mathtt{d}k_{21} = g_{11}v_{21} + g_{12}v_{22} + g_{21}v_{31} + g_{22}v_{32}$$
$$\mathtt{d}k_{22} = g_{11}v_{22} + g_{12}v_{23} + g_{21}v_{32} + g_{22}v_{33}$$

$$\underbrace{\begin{array}{|c|c|}\hline \mathtt{d}k_{11} & \mathtt{d}k_{12} \\ \hline \mathtt{d}k_{21} & \mathtt{d}k_{22} \\ \hline \end{array}}_{\mathtt{d}K} = \underbrace{\begin{array}{|c|c|}\hline g_{11} & g_{12} \\ \hline g_{21} & g_{22} \\ \hline \end{array}}_{G} * \underbrace{\begin{array}{|c|c|c|}\hline v_{11} & v_{12} & v_{13} \\ \hline v_{21} & v_{22} & v_{23} \\ \hline v_{31} & v_{32} & v_{33} \\ \hline \end{array}}_{V}$$



Review: backprop over convolution



- forward

$$K * V = Z$$
$$C\mathbf{v} = \mathbf{z}$$

- backward[3]

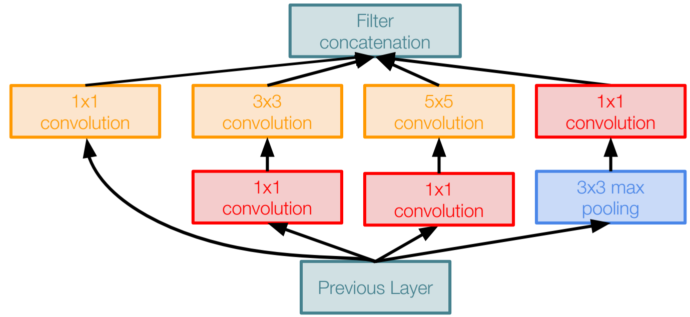
$$\mathtt{d}V = \widetilde{K} * \widehat{G}$$
$$\mathtt{d}\mathbf{v} = C^\top \mathbf{g}$$
$$\mathtt{d}K = G * V$$

- $\mathbf{v}, \mathbf{z}, \mathbf{g}$: flattened versions of V, Z, G, respectively
- C : matrix representation of "convolution with K" $[K * (\ )]$
  - $C^\top$: leads to transposed convolution $[\widetilde{K} * \widehat{(\ )}]$
- $\widetilde{K}$: flipped version of kernel K
- $\widehat{G}$: zero-padded version of G

[3] non-commutativity of "convolution without kernel flipping" $\Rightarrow$ difference in operations to compute $\mathtt{d}V$ and $\mathtt{d}K$

Outline

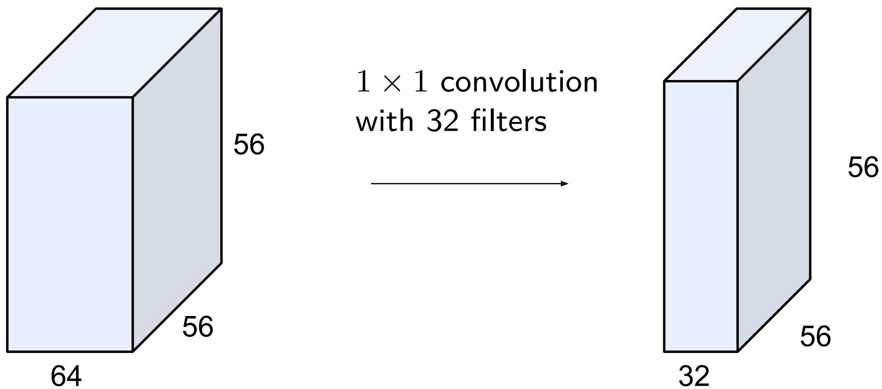
1×1 convolution



- *aka* pointwise convolution
- widely used for depth adjustment

  e.g. inception module in GoogeLeNet

- example:

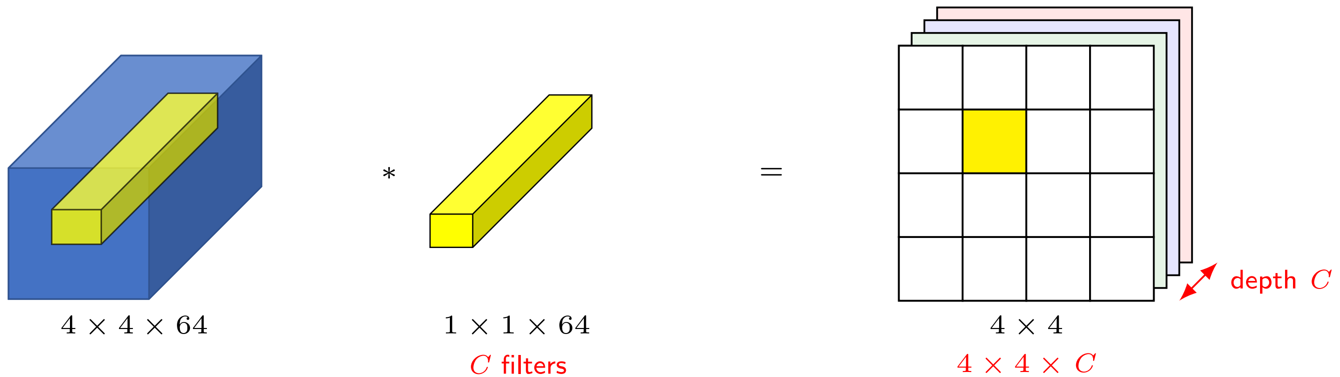


  - each filter: size $1 \times 1 \times 64$ (performs a 64-dim dot product)

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

1×1 convolution on volumes

$$\underbrace{\begin{bmatrix} 6 & 2 & 1 & 1 \\ 9 & 4 & 5 & 6 \\ 2 & 7 & 5 & 3 \\ 8 & 2 & 3 & 5 \end{bmatrix}}_{4 \times 4} * \begin{bmatrix} 2 \end{bmatrix} = \underbrace{\begin{bmatrix} 12 & 4 & 2 & 2 \\ 18 & 8 & 10 & 12 \\ 4 & 14 & 10 & 6 \\ 16 & 4 & 6 & 10 \end{bmatrix}}_{4 \times 4}$$



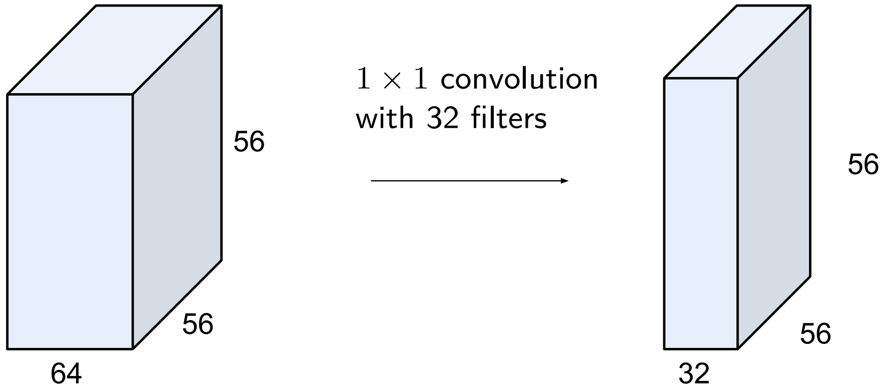
- nonlinearity (*e.g.* ReLU) can follow 1×1 convolution $\rightarrow$ "network in network"

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

Depth adjustment

- 1×1 convolution: widely used for depth adjustment

e.g.



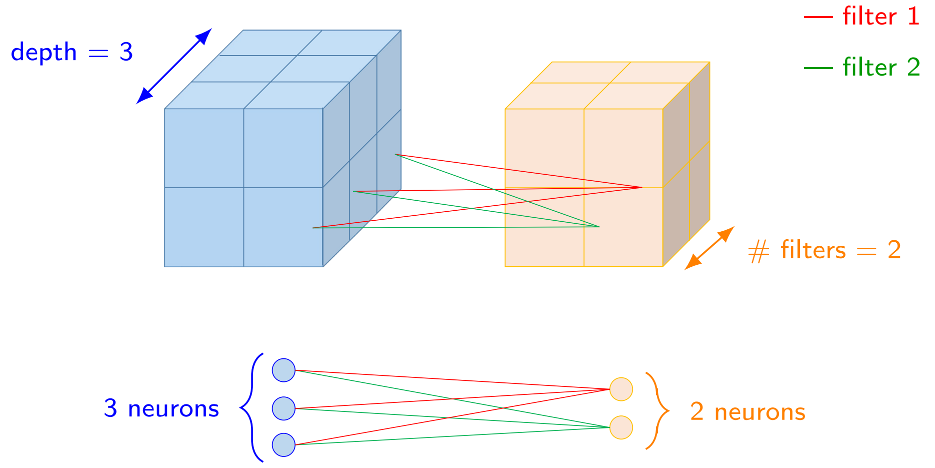
  - each filter: size $1 \times 1 \times 64$ (performs a 64-dim dot product)
  - preserves spatial dimensions and reduces depth
  - projects depth to lower dimension (combination of feature maps)

- in general
  - we can reduce/maintain/increase depth using 1×1 convolution

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

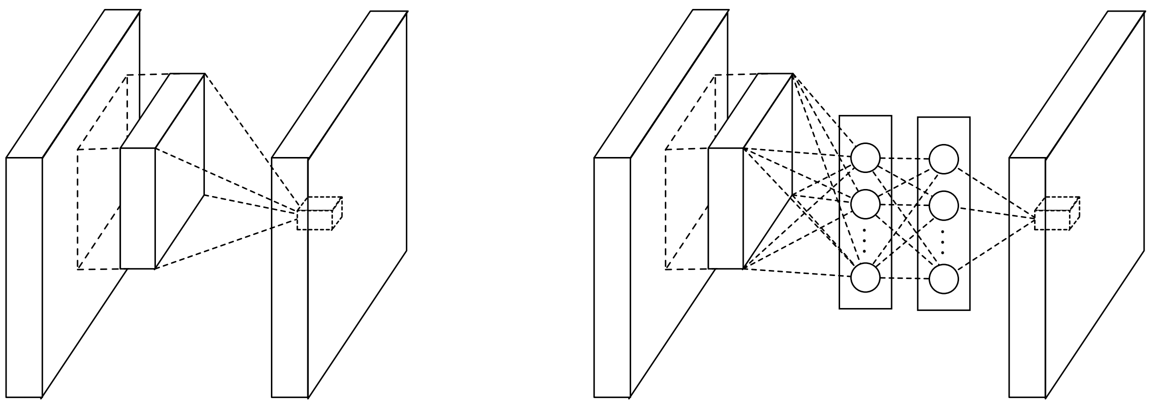
- a set of 1×1 conv filters: can be interpreted as forming an FC layer
  - input dimension of this FC layer
    $=$ depth of the input volume to 1×1 conv filters
  - output dimension of this FC layer
    $=$ number of 1×1 conv filters

- example: $2 \times 2 \times 3$ volume applied to two $1 \times 1 \times 3$ filters
  - $2 \times 2 = 4$ applications of the FC layer that maps 3 neurons to 2 neurons



Network in network

- Mlpconv layer with $\underbrace{\text{“micro network”}}_{\uparrow \atop \text{composed of FC layer (with } 1 \times 1 \text{ conv) + nonlinearity}}$ within each conv layer
  - can compute more abstract features for local patches
  - precursor to GoogLeNet and ResNet “bottleneck” layers



(a) linear convolution layer (b) Mlpconv layer

Lin, M., Chen, Q. and Yan, S., 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Outline

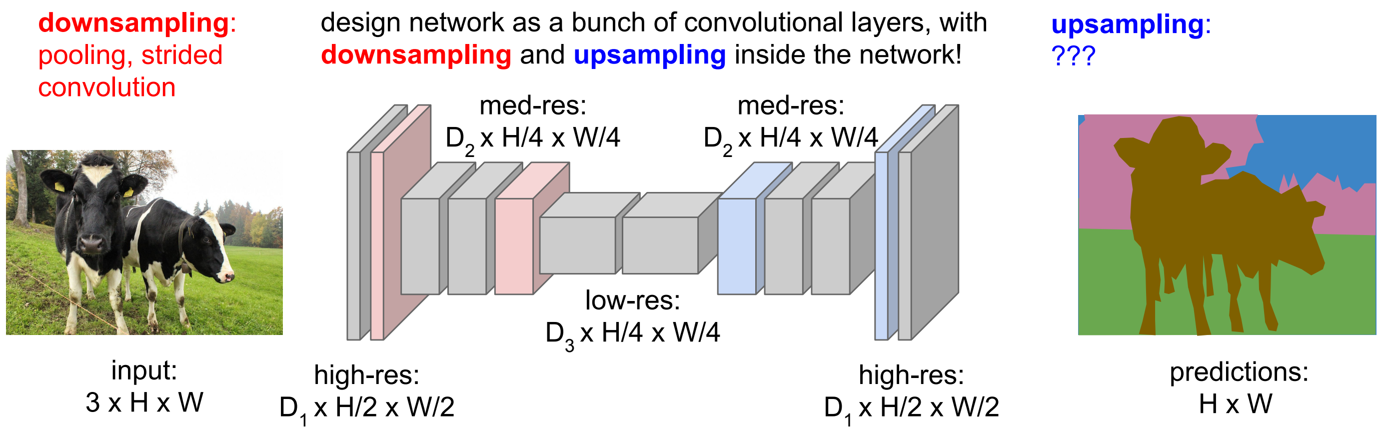
Motivation: upsampling

- desire to use a transform going in opposite direction of normal convolution

  e.g. decoding layer of a convolutional autoencoder

  project feature maps to a higher-dim space

- example in cv: semantic segmentation



  - downsampling: convolution + pooling
  - how to do upsampling?

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

Rule-based upsampling

- nearest neighbor

1	2
3	4

input: 2 x 2

$\rightarrow$

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

output: 4 x 4

- "bed of nails"

1	2
3	4

input: 2 x 2

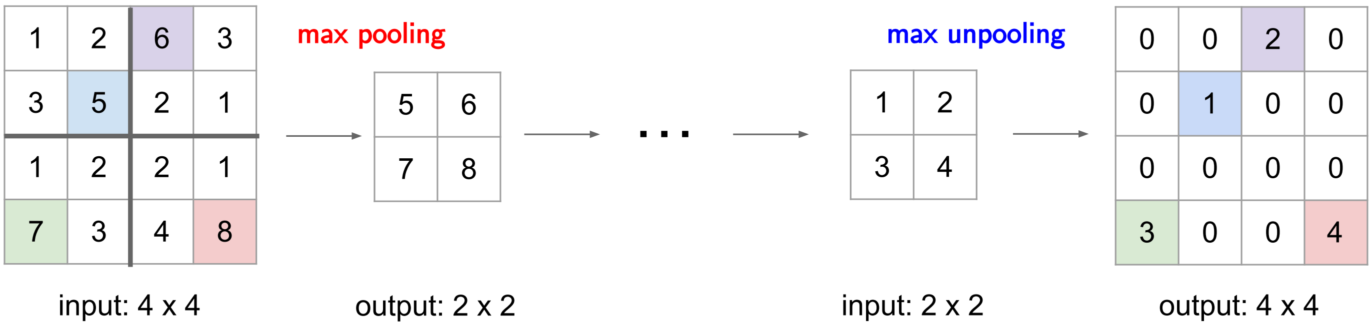
$\rightarrow$

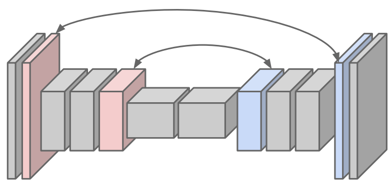
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

output: 4 x 4

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 unpooling: remember max positions from pooling layer





corresponding pairs of downsampling and upsampling layers

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

Transposed convolution

- learnable upsampling: smaller map → larger map

  c.f. regular convolution: downsampling (larger map → smaller map)
  - does not refer to the deconvolution (*i.e.* inverse of convolution)[4]
  - *aka* fractionally strided[5] conv, upconv, backward strided conv

- example

  image source: Dumoulin & Visin (2016), `https://arxiv.org/pdf/1603.07285.pdf`
  - 3×3 filter
  - 2×2 input → 5×5 output

- note:

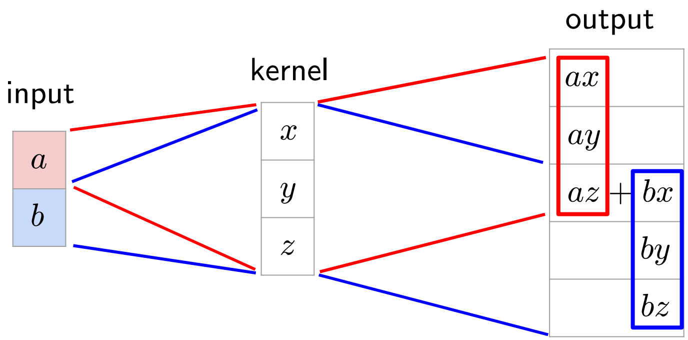
$$
\begin{tikzcd}
\text{input} \arrow[r, shift left, "\text{conv}"] \arrow[d, phantom, "\neq"] & \text{feature map} \arrow[l, shift left, "\text{inverse of conv}"] \arrow[dl, "\text{transposed conv}"] \\
\widehat{\text{input}} &
\end{tikzcd}
$$

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

[5] stride: gives ratio between movement in output and input (*i.e.* in-pixels/out-pixels)

more examples:

- 1D transposed convolution (3×1 kernel)
  - output contains copies of kernel weighted by input
  - overlaps are summed



- 2D transposed convolution (2×2 kernel)

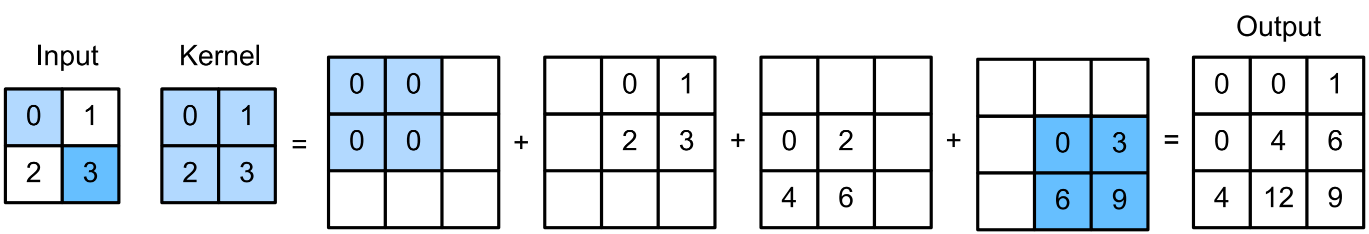
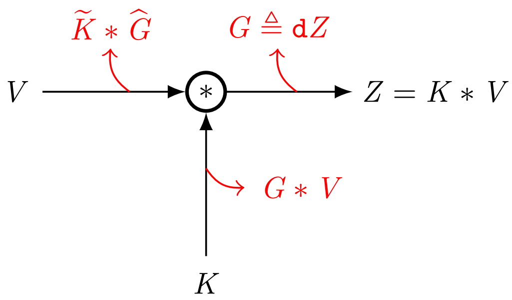


image sources: Fei-Fei Li, J. Johnson, S. Yeung, CS231n: Convolutional Neural Networks for Visual Recognition (2017), http://cs231n.stanford.edu/2017/index.html; Zhang *et al.*, *Dive into Deep Learning* (2019), https://d2l.ai/

More formally

- recall: backprop over convolution
  - assume: $\dim(\mathbf{v}) > \dim(\mathbf{z})$



  - forward (C: dim reducer)

$$K * V = Z$$
$$C\mathbf{v} = \mathbf{z}$$

  - backward ($C^\top$: dim expander)

$$\mathtt{d}V = \widetilde{K} * \widehat{G}$$
$$\mathbf{dv} = C^\top \mathbf{g}$$

- transposed convolution with a kernel
  - defined as backward pass of regular convolution with the same kernel

$$\mathrm{TransposedConv}(K, X) = \widetilde{K} * \widehat{X}$$

  - more details: Link 1 Link 2

Outline

More on Convolution

- Backprop over Convolution
- 1×1 Convolution
- Transposed Convolution
- Other Types

Summary

Dilated convolution (atrous convolution)

- introduces another convolution parameter: dilation rate
  - defines a spacing between values in a filter

e.g. 3×3 filter with dilation rate 2

$\Rightarrow$ 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

animation here

image source: Dumoulin & Visin (2016), `https://arxiv.org/pdf/1603.07285.pdf`

Separable convolution

- two types
  - spatially separable conv
  - depthwise separable conv: popular (*e.g.* MobileNet, Xception)

- spatially separable convolution
  - operates on 2D spatial dimensions (height and width)
  - decomposes a convolution into two separate operations

i.e. a 2D kernel $\rightarrow$ two 1D kernels

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

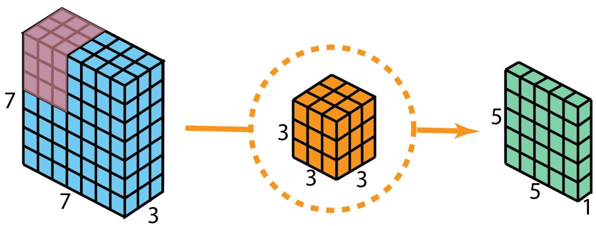
  - applies each 1D kernel in turn ▸ example

$\Rightarrow$ reduction in computation compared with regular convolution

  - rarely used in deep learning (not every 2D kernel is decomposable)

- **depthwise separable** convolution: two-step process
    1. depthwise convolution
    2. pointwise (1×1) convolution
- *e.g.* $7 \times 7 \times 3$ image $\rightarrow$ $5 \times 5 \times 1$ feature map $\qquad [(k, m, s, p) = (3, 7, 1, 0)]$

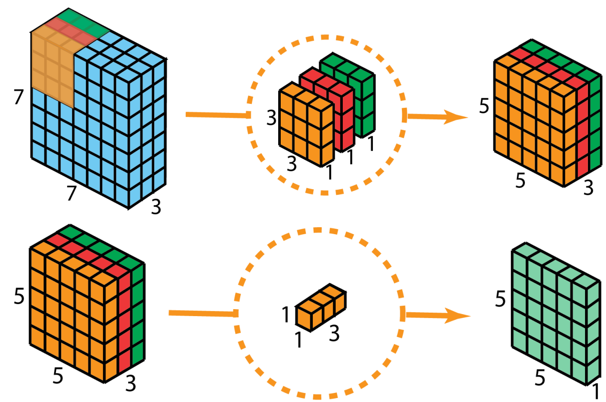
regular convolution



$$(3 \times 3 \times 3) \times (5 \times 5) = 675 \text{ mults}$$

no computational gain for 1 filter!

depthwise separable convolution

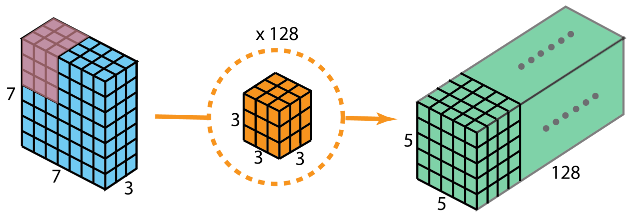


$$3 \times (3 \times 3 \times 1) \times (5 \times 5) = 675$$
$$+$$
$$(1 \times 1 \times 3) \times (5 \times 5) = 75$$

total: $675 + 75 = 750$ mults

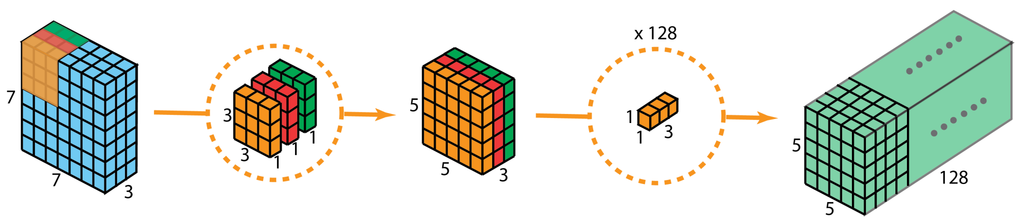
image source: `https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`

- how about multiple (*e.g.* 128) filters?
  - regular convolution



$$128 \times (3 \times 3 \times 3) \times (5 \times 5) = 128 \times 675 = 86,400 \text{ mults}$$

  - depthwise separable convolution



$$3 \times (3 \times 3 \times 1) \times (5 \times 5) + 128 \times (1 \times 1 \times 3) \times (5 \times 5) = 9,600 \text{ mults}$$

$\Rightarrow$ huge computational gain! (only 11% of regular conv)

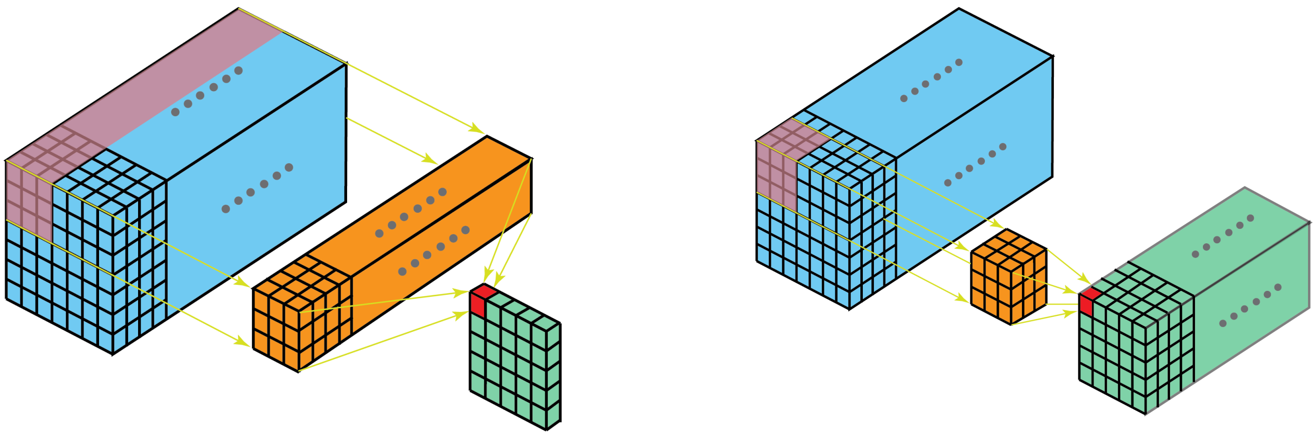
- significantly fewer kernel parameters ($128 \times 27 = 3,456$ vs $27 + 128 \times 3 = 411$)

  $\Rightarrow$ reduced model capacity (problematic if not properly trained)

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

3D convolution

- regular (2D) convolution
  - kernel depth = input depth
  - kernel moves only in 2D (width, height)
- 3D convolution
  - kernel depth < input depth
  - kernel moves in all 3D (width, height, depth)

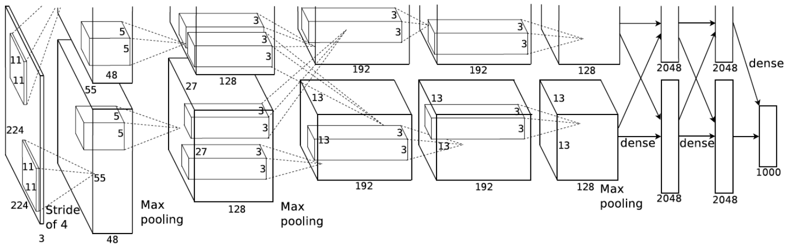


- d-dim conv: describes spatial relationships of object in d-dim space

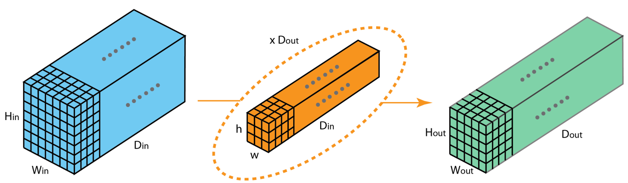
image source: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Grouped convolution

- first introduced in AlexNet (2012): mainly due to limited memory
  - other advantages also exist ▸ details



- regular convolution



- grouped convolution (2 groups)

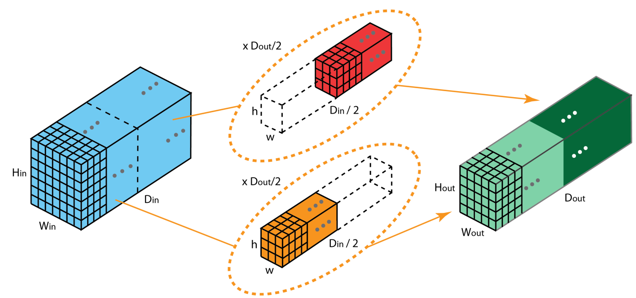


image sources: Krizhevsky *et al.*, 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*; https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

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

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

$\Rightarrow$ techniques that improve efficiency of only forward prop are useful

e.g. TensorRT, TensorFlowLite, Core ML, Caffe2Go

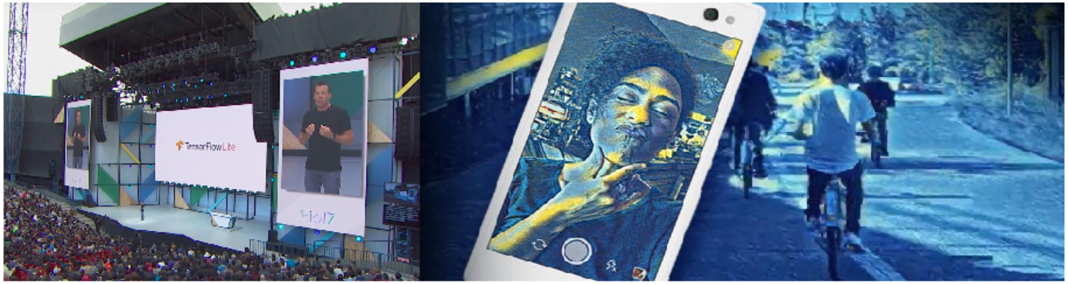


image sources: Apple Core ML, Google TensorFlow Lite, Facebook Caffe2Go

Outline

Summary

- backprop over convolution (assumption: $\dim(\mathbf{v}) > \dim(\mathbf{z})$)
  - forward (C: dim reducer)

$$K * V = Z$$

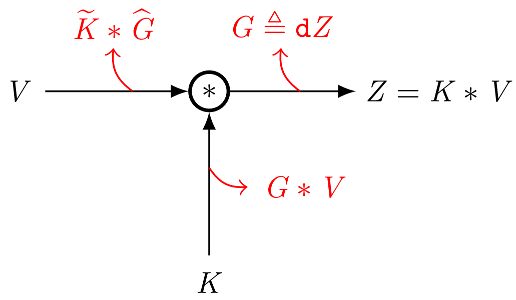
$$C\mathbf{v} = \mathbf{z}$$

  - backward ($C^\top$: dim expander)

$$\mathtt{d}V = \widetilde{K} * \widehat{G}$$

$$\mathtt{d}\mathbf{v} = C^\top \mathbf{g}$$

$$\mathtt{d}K = G * V$$



- various types of convolution operations exist (their fast inference: crucial)
  - pointwise (1×1) convolution: for depth adjustment
  - transposed (fractionally strided) convolution: for learnable upsampling
  - dilated convolution, separable convolution: for efficiency ($+$ alpha)
  - 3D convolution, grouped convolution, and many others