M2177.003100
Deep Learning

[7: Convolutional Neural Nets (Part 2)]

Electrical and Computer Engineering
Seoul National University

(last compiled at 18:15:00 on 2019/10/13)

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

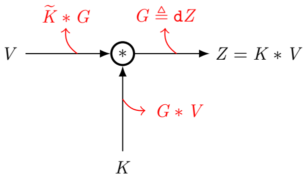
$$
\begin{aligned}
Z(i,j) = (K * V)(i,j) &= \sum_m \sum_n \underbrace{K(m,n)}_{\text{kernel}} \underbrace{V(i-m, j-n)}_{\text{input volume}} \\
&= \sum_m \sum_n K(i-m, j-n) V(m,n) \\
&= (V * K)(i,j)
\end{aligned}
$$

- 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

Backprop over convolution



- forward

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

- backward

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

- $\mathbf{v}, \mathbf{z}, \mathbf{g}$: flattened versions of V, Z, G, respectively
- C : matrix representation of convolution
  - $C^\top$: leads to __________ convolution
- $\widetilde{K}$: flipped version of kernel K

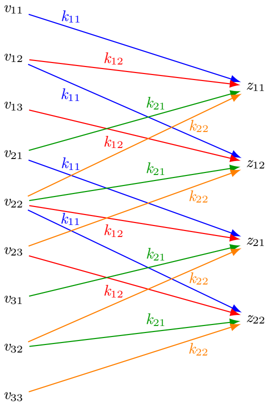
Running example

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

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

  - result

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



^{1}sizes of kernel, input, stride, padding, respectively

Convolution as a matrix operation

$$= \mathbf{z}$$

where

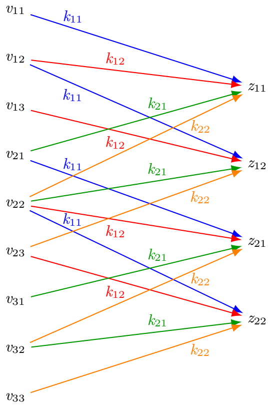
$$\mathbf{v} = (v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, v_{23}, v_{31}, v_{32}, v_{33})$$

$$\mathbf{z} = (z_{11}, z_{12}, z_{21}, z_{22})$$

$$C = \text{a sparse matrix}$$

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

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



- simplified representation

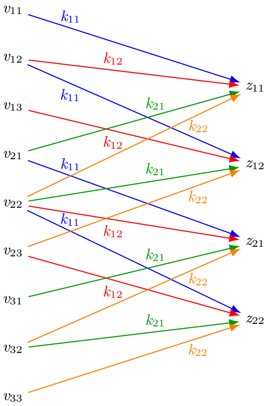
$$
\underbrace{\begin{bmatrix}
k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\
0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\
0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\
0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22}
\end{bmatrix}}_{C}
\underbrace{\begin{bmatrix}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

$$
\begin{aligned}
k_{11}v_{11} + k_{12}v_{12} + k_{21}v_{21} + k_{22}v_{22} &= z_{11} \\
k_{11}v_{12} + k_{12}v_{13} + k_{21}v_{22} + k_{22}v_{23} &= z_{12} \\
k_{11}v_{21} + k_{12}v_{22} + k_{21}v_{31} + k_{22}v_{32} &= z_{21} \\
k_{11}v_{22} + k_{12}v_{23} + k_{21}v_{32} + k_{22}v_{33} &= z_{22}
\end{aligned}
$$

$\Downarrow$

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

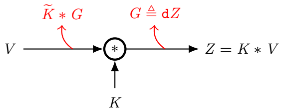
$$
\underbrace{\begin{bmatrix}
k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\
0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\
0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\
0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22}
\end{bmatrix}}_{C}
=
\underbrace{\begin{bmatrix}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

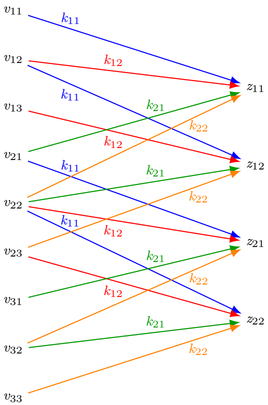


$\mathtt{d}V = \widetilde{K} * G$

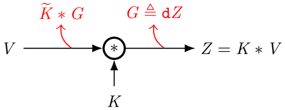
$$
\begin{aligned}
\mathtt{d}v_{11} &= k_{11} g_{11} \\
\mathtt{d}v_{12} &= k_{12} g_{11} + k_{11} g_{12} \\
\mathtt{d}v_{13} &= k_{12} g_{12} \\
\mathtt{d}v_{21} &= k_{21} g_{11} + k_{11} g_{21} \\
\mathtt{d}v_{22} &= k_{22} g_{11} + k_{21} g_{12} + k_{12} g_{21} + k_{11} g_{22} \\
\mathtt{d}v_{23}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

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





- matrix representation (reveals "________ convolution")



backward

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

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

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

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

forward

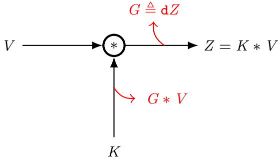
$$K * V = Z$$

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

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

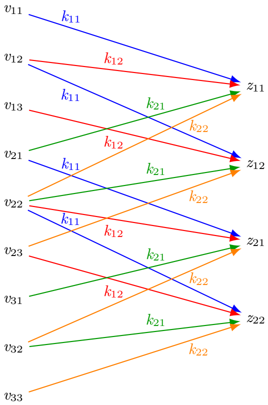
$$\underbrace{\begin{bmatrix} k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & k_{11} & k_{12} & 0 & k_{21} & k_{22} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix}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

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



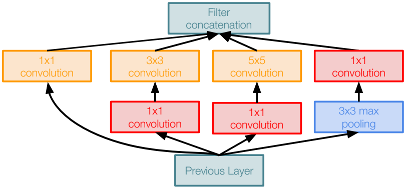
$$\mathtt{d}k_{11} = g_{11}v_{11} + g_{12}v_{12} + g_{21}v_{21} + g_{22}v_{22}$$
$$\mathtt{d}k_{12} = g_{11}v_{12} + g_{12}v_{13} + g_{21}v_{22} + g_{22}v_{23}$$
$$\mathtt{d}k_{21} = g_{11}v_{21} + g_{12}v_{22} + g_{21}v_{31} + g_{22}v_{32}$$
$$\mathtt{d}k_{22} = g_{11}v_{22} + g_{12}v_{23} + g_{21}v_{32}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

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



Outline

1×1 convolution

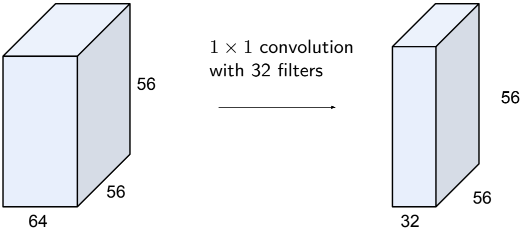


(source: Szegedy et al., 2014)

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

  e.g. inception module in GoogeLeNet

- example:

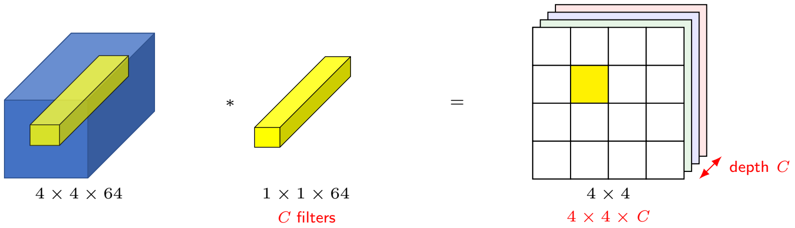


(source: cs231n)

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

1×1 convolution on volumes

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



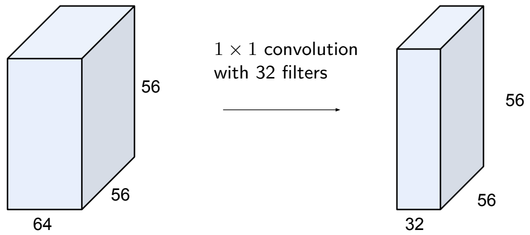
(source: coursera)

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

Depth adjustment

- 1×1 convolution: widely used for depth adjustment

  e.g.



(source: cs231n)

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

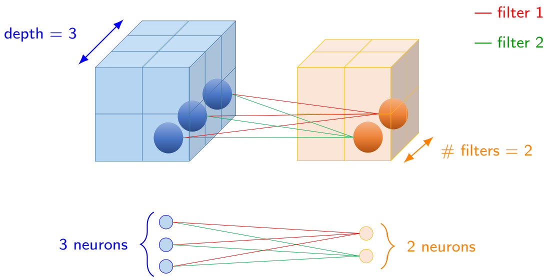
- in general
  - we can reduce/maintain/increase depth using 1×1 convolution

- a set of 1×1 conv filters: can be interpreted as forming an ________
  - input dimension of this FC layer

    $=$ _____ of the input volume to 1×1 conv filters
  - output dimension of this FC layer

    $=$ ______ of 1×1 conv filters

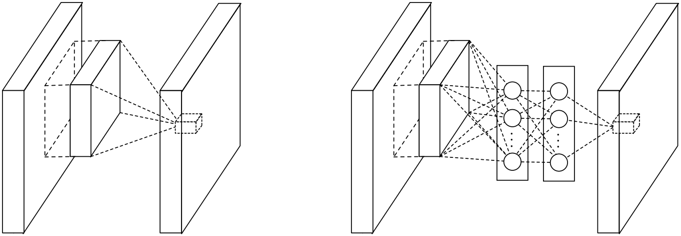
- example: $2 \times 2 \times 3$ volume applied to two $1 \times 1 \times 3$ filters
  - $2 \times 2 = 4$ applications of the FC layer that maps 3 neurons to 2 neurons



Network in network

- Mlpconv layer with “$\underbrace{\qquad\qquad\qquad}$” within each conv layer

  $\uparrow$
  composed of FC layer (with 1×1 conv) $+$ nonlinearity
  - can compute more abstract features for local patches
  - precursor to GoogLeNet and ResNet “bottleneck” layers



(a) linear convolution layer (b) Mlpconv layer

(source: Lin et al., 2014)

Outline

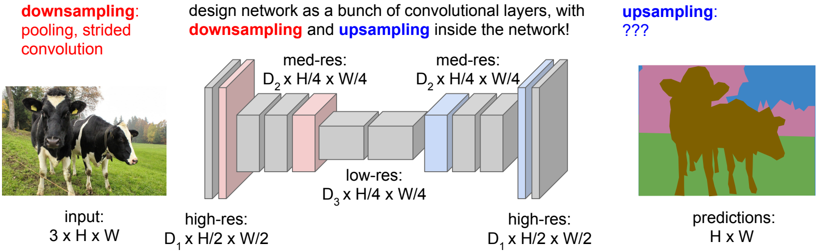
Motivation: upsampling

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

  e.g. ________ layer of a convolutional autoencoder

  project feature maps to a higher-dim space

- example in cv: semantic segmentation

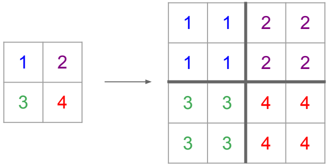


(source: cs231n)

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

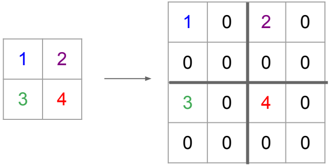
Rule-based upsampling

- nearest neighbor



input: 2 x 2 output: 4 x 4

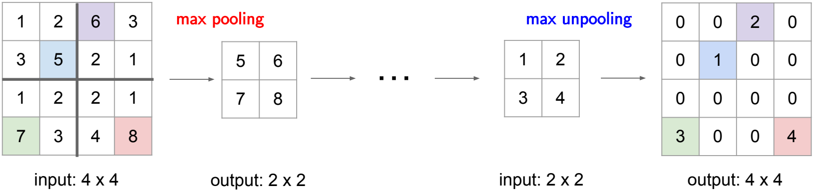
- "bed of nails"

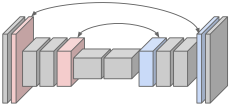


input: 2 x 2 output: 4 x 4

(source: cs231n)

- max unpooling: remember ____ positions from pooling layer





corresponding pairs of downsampling and upsampling layers

(source: cs231n)

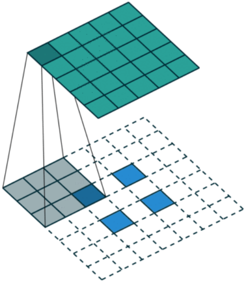
Transposed convolution

- _______ upsampling: smaller map → larger map

  c.f. regular convolution: downsampling (larger map → smaller map)
  - does not perform the deconvolution (*i.e.* inverse of convolution)[2]
  - *aka* fractionally strided[3] conv, upconv, backward strided conv

- example
  - 3×3 filter
  - 2×2 input $\to$ 5×5 output

(source: Theano tutorial)



- note:

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

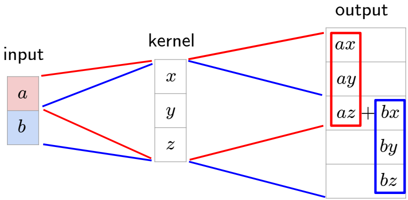
---

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

[3] stride: gives ratio between movement in output and input (in pixels/out pixels)

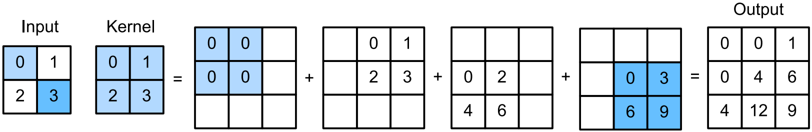
more examples:

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



(source: cs231n)

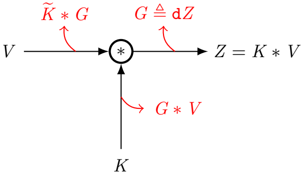
- 2D transposed convolution (2×2 kernel)



(source: dive into DL)

More formally

- recall: backprop over convolution



- forward

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

- backward

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

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

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

  - more details: Link 1 Link 2

- the kernel K defines a convolution
  - whose forward and backward passes:
    - computed by multiplying C and $C^\top$, respectively
- K *also* defines a transposed convolution
  - whose forward and backward passes:
    - computed by multiplying $C^\top$ and $(C^\top)^\top = C$, respectively
- implementation of transposed convolution
  - implement with a normal convolution + (complicated) zero-padding
    - easy to understand but much less efficient
  - implement as ________ of some convolution wrt its input
    - usually how it is implemented in practice

Outline

Dilated convolution (atrous convolution)

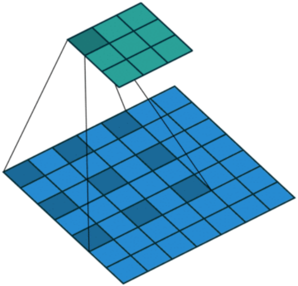
- introduces another convolution parameter: dilation rate
  - defines a spacing between values in a filter

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 _____ 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)

Separable convolution

- two types
  - spatially separable conv
  - ________ 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

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

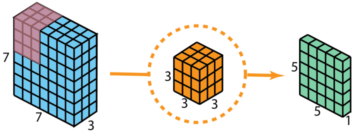
  - applies each 1D kernel in turn ▸ example

  $\Rightarrow$ 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 $\quad [(k, m, s, p) = (3, 7, 1, 0)]$

regular convolution

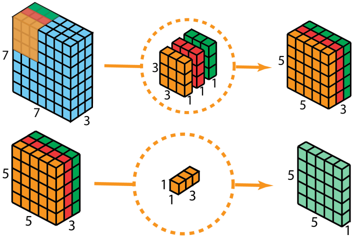


(source: Bai's blog)

$(3 \times 3 \times 3) \times (5 \times 5) = 675$ mults

__ computational gain for 1 filter!

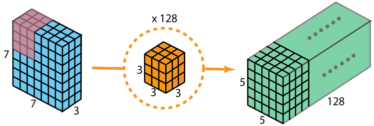
depthwise separable convolution



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

total: $675 + 75 = 750$ mults

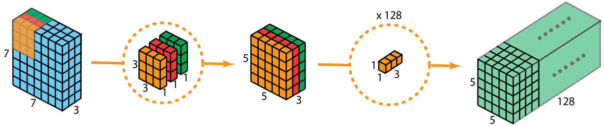
- how about multiple (*e.g.* 128) filters?
  - regular convolution



(source: Bai's blog)

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

  - depthwise separable convolution



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

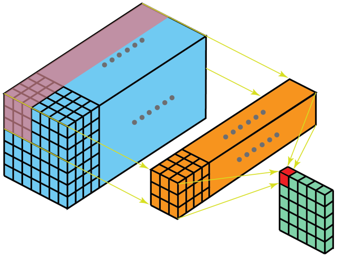
$\Rightarrow$ huge computational gain! (only 11% of regular conv)

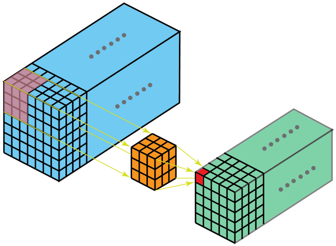
- significantly fewer kernel parameters ($128 \times 27 = 3,456$ vs $27 + 128 \times 3 = 411$)

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

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 ___ (width, height, depth)





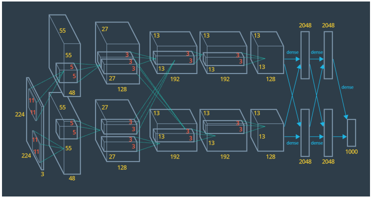
(source: Bai's blog)

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

Grouped convolution

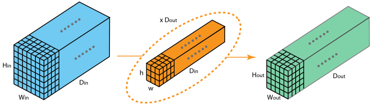
- first introduced in AlexNet (2012)
  - mainly due to limited ______
  - other advantages also exist

details

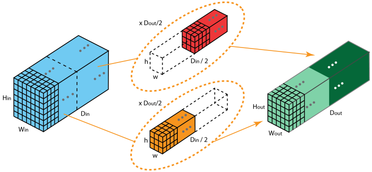


(source: yuchao.us)

- regular convolution



- grouped convolution (2 groups)



(source: Bai's blog)

Convolution by frequency domain conversion

- convolution: equivalent to the following
  1. convert both input/kernel to ________ 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

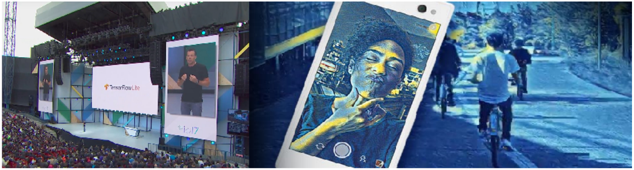
Remarks

- 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

  ⇒ techniques that improve efficiency of only ______ prop are useful

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



Outline

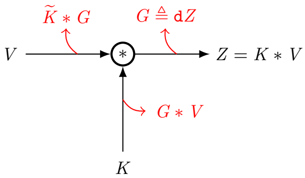
Summary

- backprop over convolution
  - forward

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

  - backward

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



- 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