Convolutional Layers

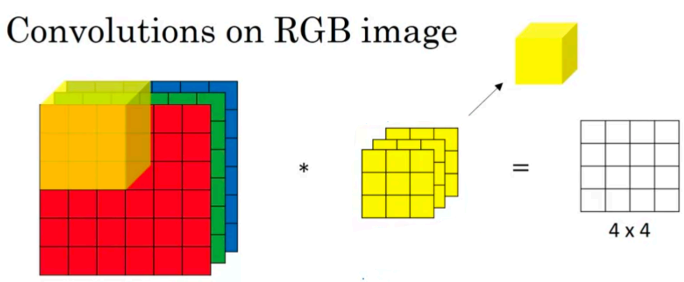


Figure from Andrew Ng

Convolutional layer:

- Apply filter at each location in input
    - Multiply all corresponding numbers
    - Sum results
- Input shape: $n_h \times n_w \times n_c$
- Output shape:

$$\left\lfloor \frac{n_h + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p - f}{s} + 1 \right\rfloor \times n_{filters}$$

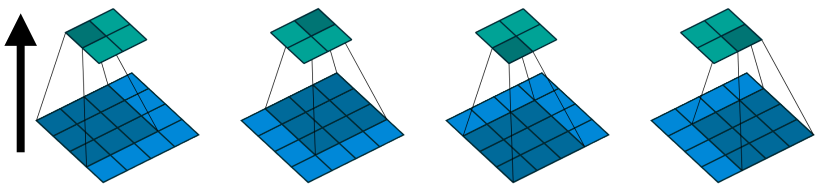


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)

Convolutional Layers

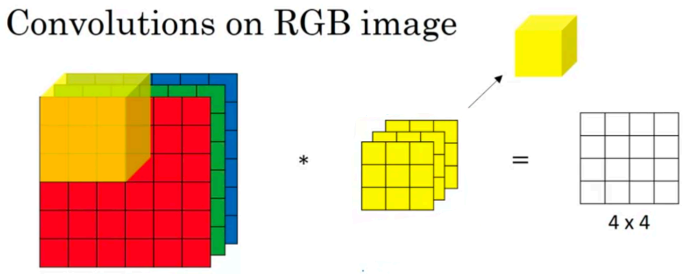


Figure from Andrew Ng

Convolutional layer:

- Apply filter at each location in input
  - Multiply all corresponding numbers
  - Sum results
- Input shape: $n_h \times n_w \times n_c$
- Output shape:

$$\left\lfloor \frac{n_h + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p - f}{s} + 1 \right\rfloor \times n_{filters}$$

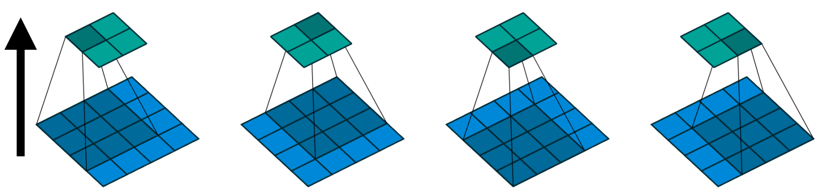


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)

Can we go backwards?

Algorithm for Convolutions

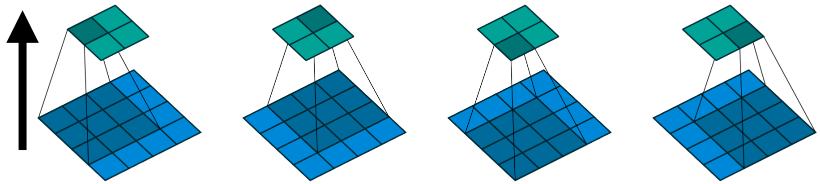
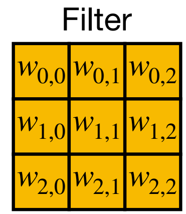


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)



- For i = 0, . . . , $\left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor$
  - For j = 0, . . . , $\left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor$
    - start_row = i * s, end_row = start_row + f
    - start_col = j * s, end_col = start_col + f
    - output[i, j] = np.sum(W * A[start_row:end_row, start_col:end_col])

Convolutions via Matrices

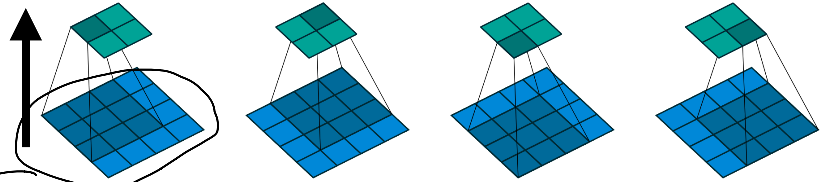
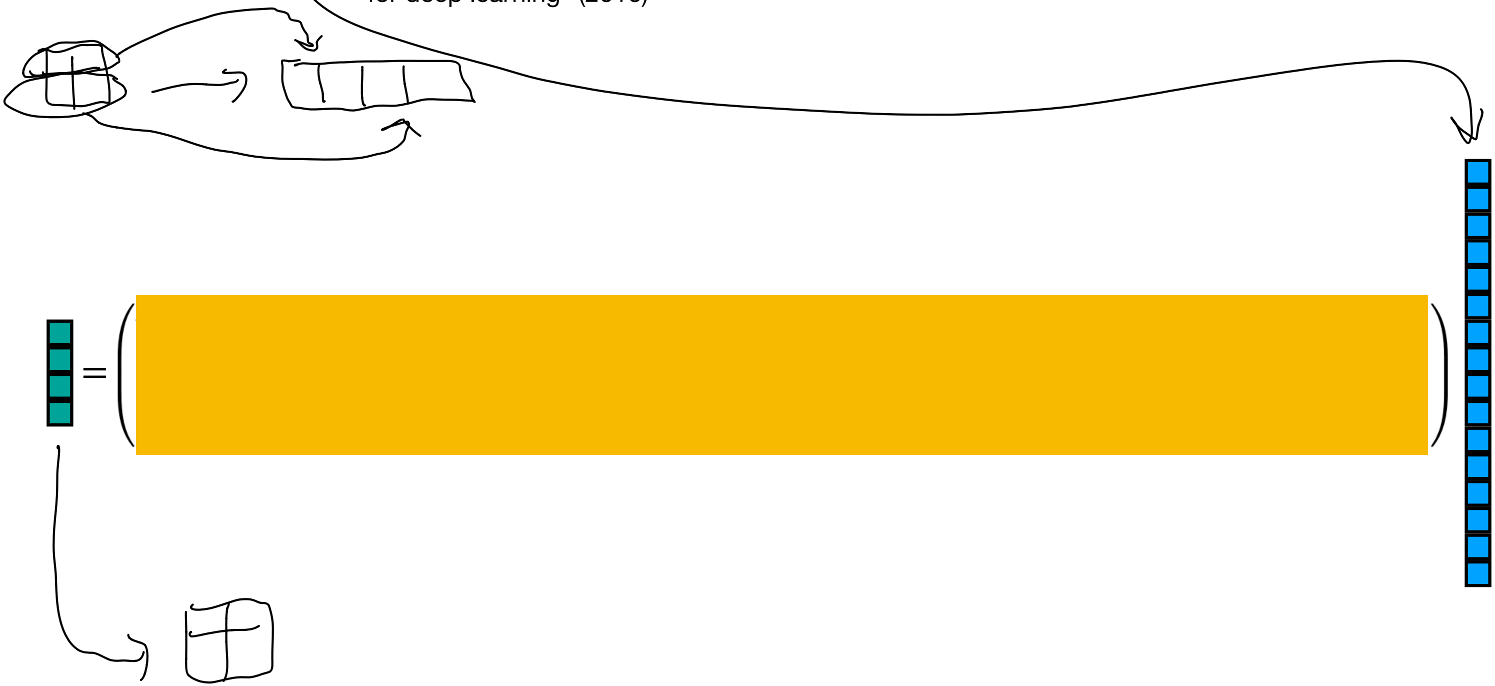


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)

Filter

$w_{0,0}$	$w_{0,1}$	$w_{0,2}$
$w_{1,0}$	$w_{1,1}$	$w_{1,2}$
$w_{2,0}$	$w_{2,1}$	$w_{2,2}$



Convolutions via Matrices

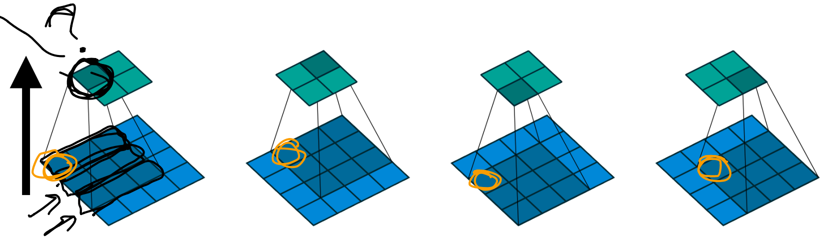
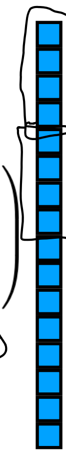


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)

Filter

$w_{0,0}$	$w_{0,1}$	$w_{0,2}$
$w_{1,0}$	$w_{1,1}$	$w_{1,2}$
$w_{2,0}$	$w_{2,1}$	$w_{2,2}$

$$= \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$



16 columns

because 4x4 input to convolution

4 rows because 2x2 output from convolution

Transposed Convolutions (aka “Deconvolutions”)

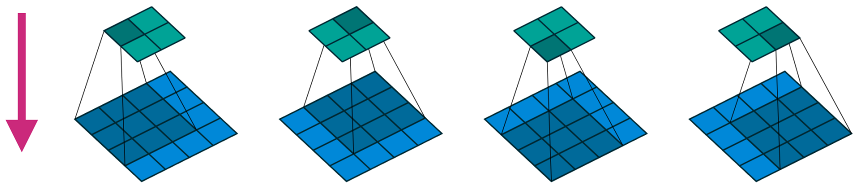
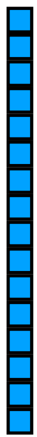
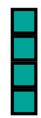


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)



$$
= \begin{pmatrix}
w_{0,0} & 0 & 0 & 0 \\
w_{0,1} & w_{0,0} & 0 & 0 \\
w_{0,2} & w_{0,1} & 0 & 0 \\
w_{0,0} & w_{0,2} & 0 & 0 \\
w_{1,0} & 0 & w_{0,0} & 0 \\
w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\
w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\
0 & w_{1,2} & 0 & w_{0,2} \\
w_{2,0} & 0 & w_{1,0} & 0 \\
w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\
w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\
0 & w_{2,2} & 0 & w_{1,2} \\
0 & 0 & w_{2,0} & 0 \\
0 & 0 & w_{2,1} & w_{2,0} \\
0 & 0 & w_{2,2} & w_{2,1} \\
0 & 0 & 0 & w_{2,2}
\end{pmatrix}
$$



Transposed Convolutions are Still Convolutions!

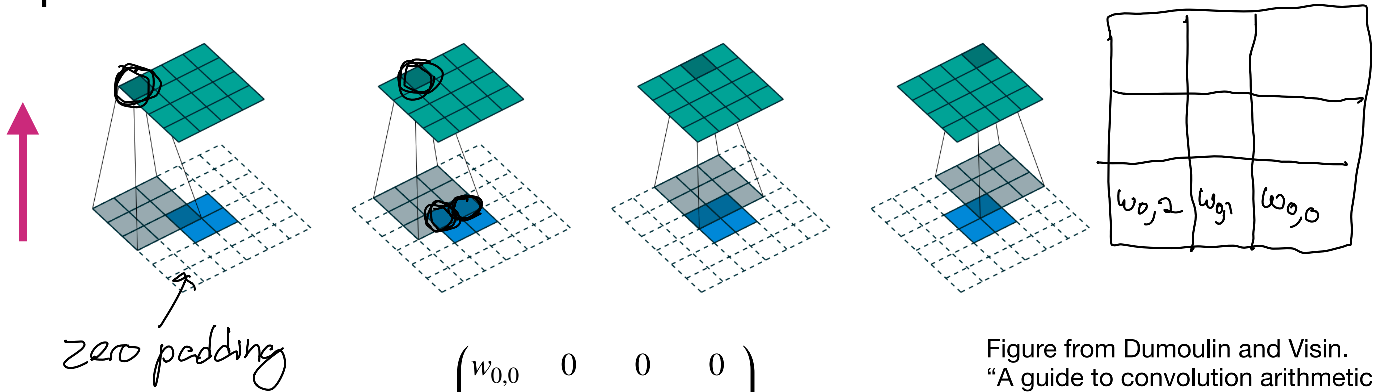
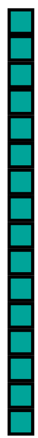
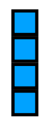


Figure from Dumoulin and Visin.
"A guide to convolution arithmetic for deep learning" (2016)

$$\left\lfloor \frac{n_h + 2p - f}{s} + 1 \right\rfloor$$



$$= \begin{pmatrix}
w_{0,0} & 0 & 0 & 0 \\
w_{0,1} & w_{0,0} & 0 & 0 \\
w_{0,2} & w_{0,1} & 0 & 0 \\
w_{0,0} & w_{0,2} & 0 & 0 \\
w_{1,0} & 0 & w_{0,0} & 0 \\
w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\
w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\
0 & w_{1,2} & 0 & w_{0,2} \\
w_{2,0} & 0 & w_{1,0} & 0 \\
w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\
w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\
0 & w_{2,2} & 0 & w_{1,2} \\
0 & 0 & w_{2,0} & 0 \\
0 & 0 & w_{2,1} & w_{2,0} \\
0 & 0 & w_{2,2} & w_{2,1} \\
0 & 0 & 0 & w_{2,2}
\end{pmatrix}$$



Transpose is not the Inverse!

Denote this matrix by C:

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

- C is not a square matrix, so it is not invertible! (Does not have full column rank!)
- The transposed convolution C^T gets us back to the original dimensions
- The transposed convolution does not technically "undo" the original convolution

result = $C^T C \cdot$ input

result and input have same dimension but are not equal!