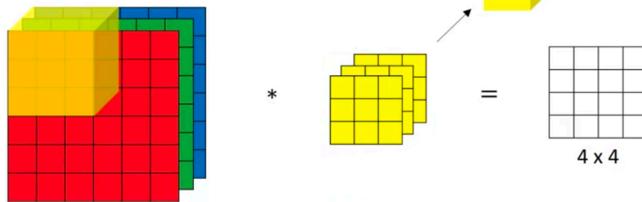


# Convolutional Layers

Convolutions on RGB image



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:

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

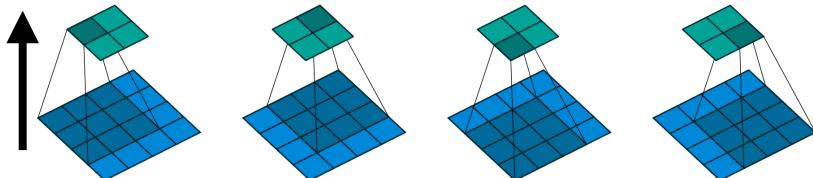


Figure from Dumoulin and Visin. "A guide to convolution arithmetic for deep learning" (2016)

# Convolutional Layers

Convolutions on RGB image

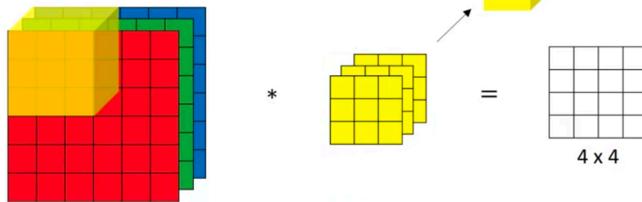


Figure from Andrew Ng

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:

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

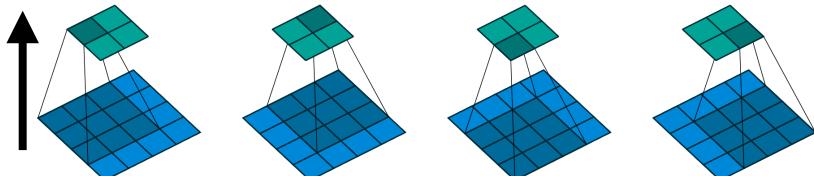
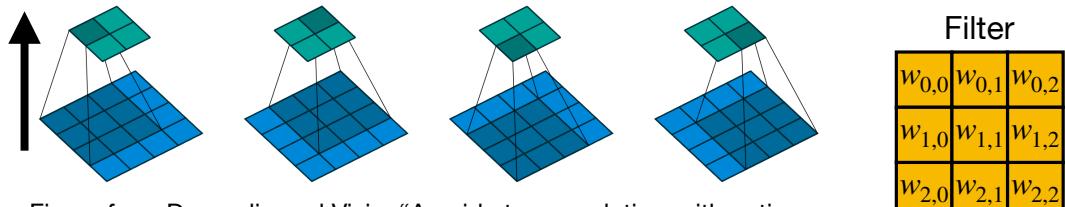


Figure from Dumoulin and Visin. "A guide to convolution arithmetic for deep learning" (2016)

↓  
Can we go backwards?

# Algorithm for Convolutions



- For  $i = 0, \dots, \left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor$ 
  - For  $j = 0, \dots, \left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor$ 
    - \*  $\text{start\_row} = i * s, \text{end\_row} = \text{start\_row} + f$
    - \*  $\text{start\_col} = j * s, \text{end\_col} = \text{start\_col} + f$
    - \*  $\text{output}[i, j] = \text{np.sum}(W * A[\text{start\_row:}\text{end\_row}, \text{start\_col:}\text{end\_col}])$

# Convolutions via Matrices

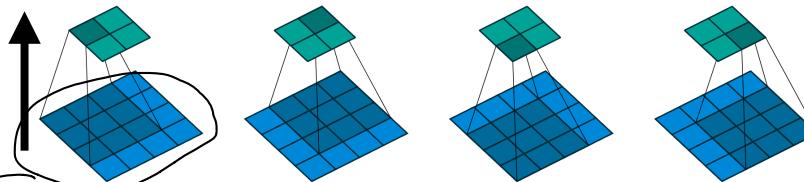
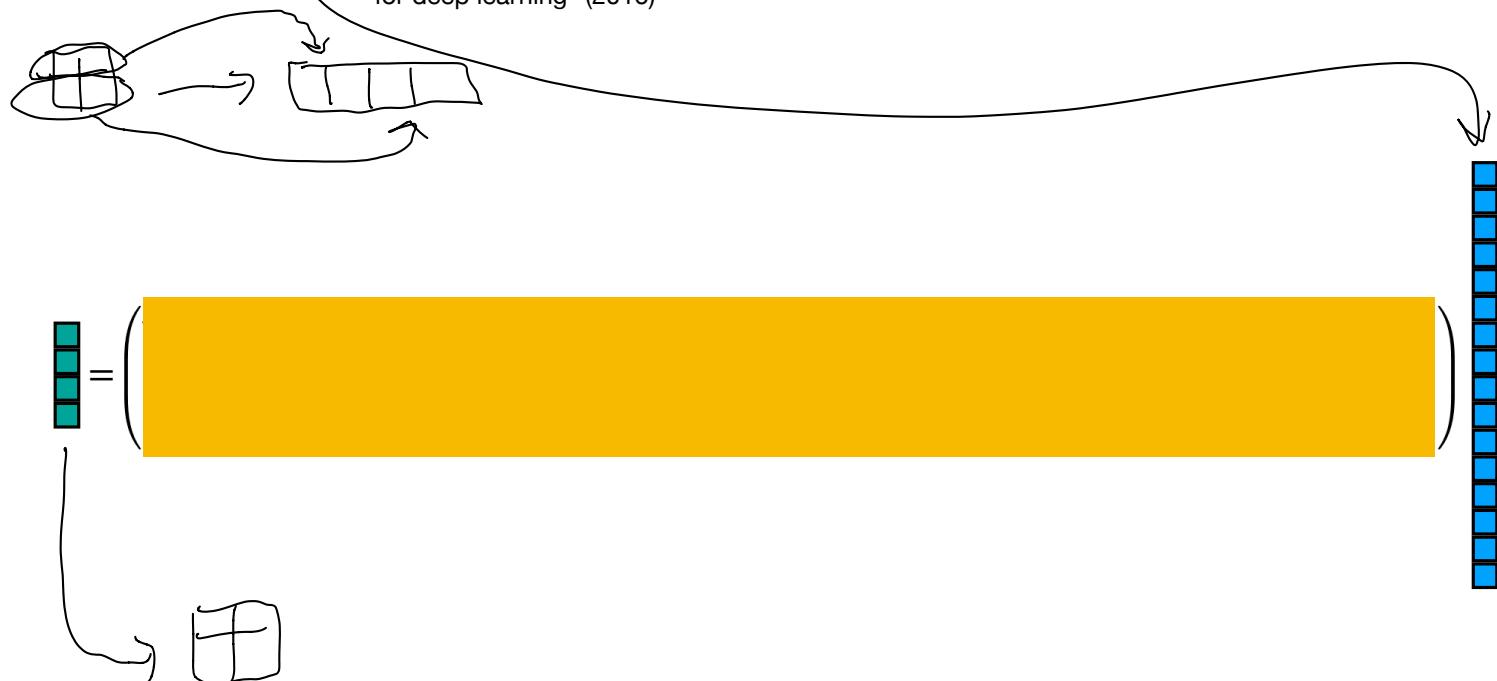
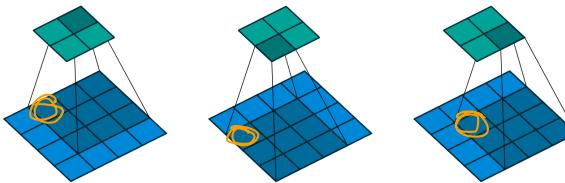
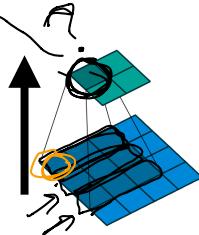


Figure from Dumoulin and Visin. "A guide to convolution arithmetic for deep learning" (2016)



# Convolutions via Matrices



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

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

$$\begin{matrix} \text{Input} \\ \downarrow \\ \text{Matrix} \end{matrix} = \left( \begin{array}{cccc|cccc|cccc|cccc} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \underline{w_{0,0}} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & \cancel{w_{0,0}} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \cancel{w_{0,0}} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \end{array} \right) \quad \begin{matrix} \text{Output} \\ \uparrow \\ \text{Matrix} \end{matrix}$$

16 columns

because 4x4 input to convolution

4 rows because 2x2 output from convolution

# Transposed Convolutions (aka “Deconvolutions”)

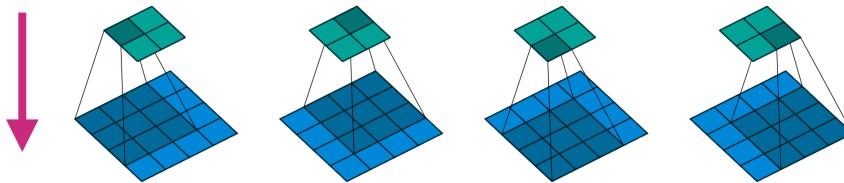


Figure from Dumoulin and Visin. “A guide to convolution arithmetic for deep learning” (2016)

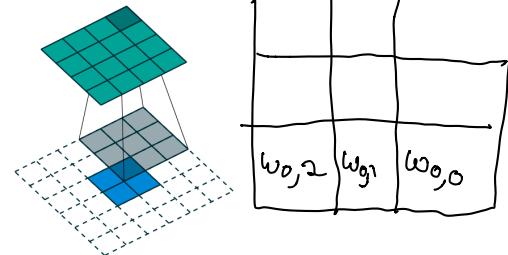
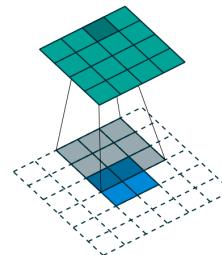
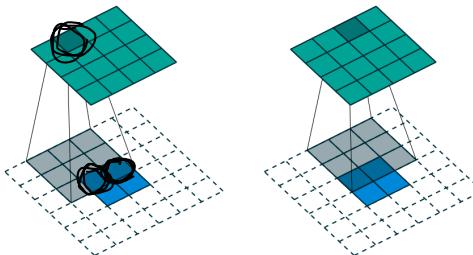
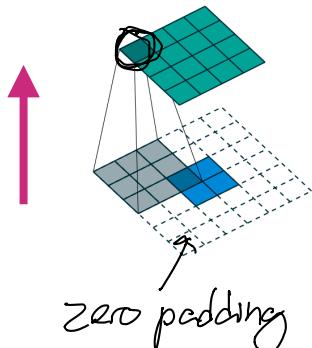
$$\begin{matrix} & \begin{matrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ w_{0,0} & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0}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

A diagram illustrating the computation of a transposed convolution. On the left, a vertical column of blue squares represents the input feature map. An equals sign follows this. To the right of the equals sign is a large matrix with 16 columns and 4 rows. The columns are labeled with weights  $w_{i,j}$  where  $i$  is the row index of the output unit and  $j$  is the column index of the input unit. The matrix is filled with zeros, except for the following non-zero elements:

- Row 0:  $w_{0,0}, w_{0,1}, w_{0,2}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$
- Row 1:  $0, w_{0,0}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$
- Row 2:  $0, 0, w_{0,1}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$
- Row 3:  $0, 0, 0, w_{0,0}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$

On the far right, a vertical column of green squares represents the output feature map.

# Transposed Convolutions are Still Convolutions!



$$\left[ \frac{n_h + 2p - f}{s} + 1 \right]$$

$$\begin{array}{c} \rightarrow \\ = \end{array} \begin{pmatrix} w_{0,0} & 0 & 0 & 0 \\ \underline{w_{0,1}} & \underline{w_{0,0}} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ w_{0,0} & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{pmatrix}$$

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

# Transpose is not the Inverse!

Denote this matrix by C:

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

- 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

$$\text{result} = C^T C \cdot \text{input}$$

result and input have same dimension but are not equal!