

# **Deep Learning Systems: Algorithms and Implementation**

## **Convolutional Networks**

Fall 2022

J. Zico Kolter (this time) and Tianqi Chen  
Carnegie Mellon University

# Outline

Convolutional operators in deep networks

Elements of practical convolutions

Differentiating convolutions

# Outline

Convolutional operators in deep networks

Elements of practical convolutions

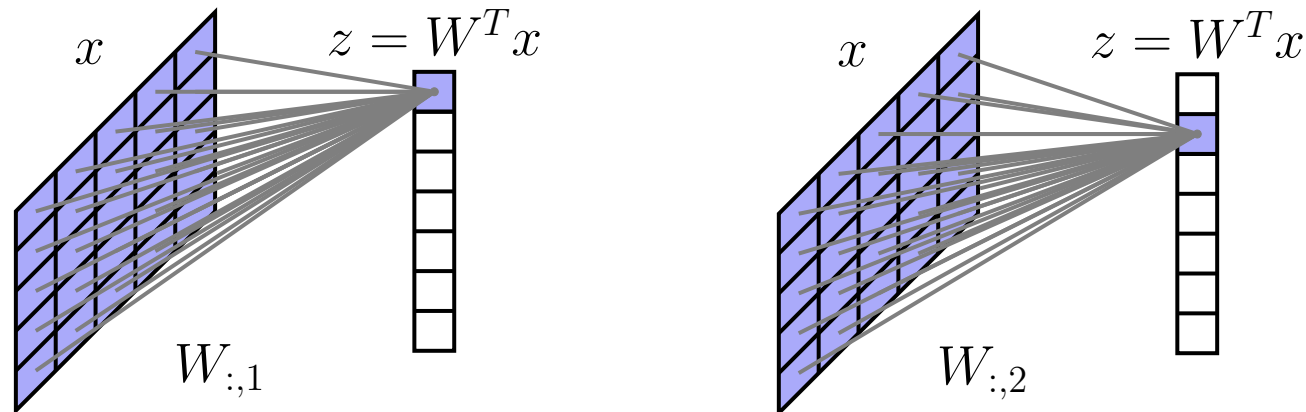
Differentiating convolutions

# The problem with fully connected networks

So far we have considered networks that treat input images as vectors

This creates a substantial problem as we attempt to handle larger images: a 256x256 RGB image  $\implies$   $\sim$ 200K dimensional input  $\implies$  mapping to 1000 dimensional hidden vector requires 200M parameters (for a *single layer*)

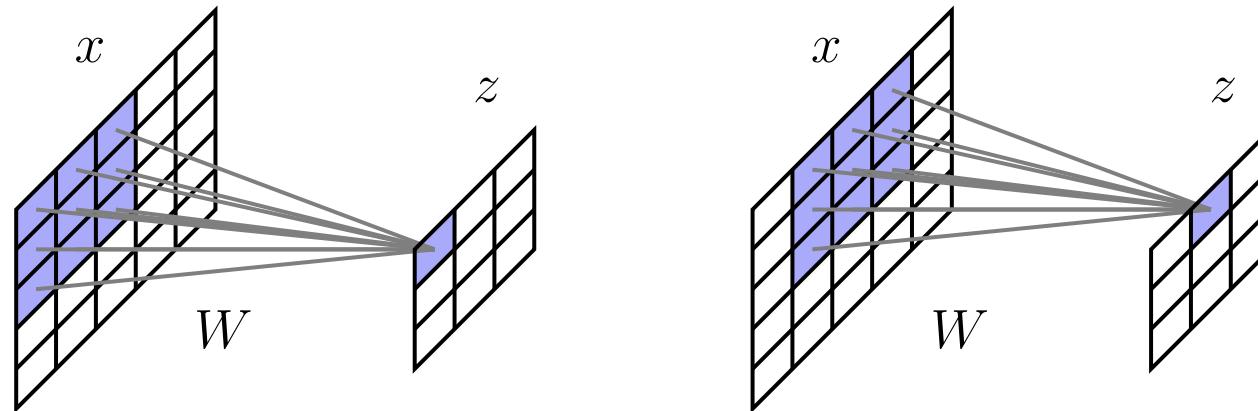
Does not capture any of the “intuitive” invariances that we expect to have in images (e.g., shifting image one pixel leads to very different next layer)



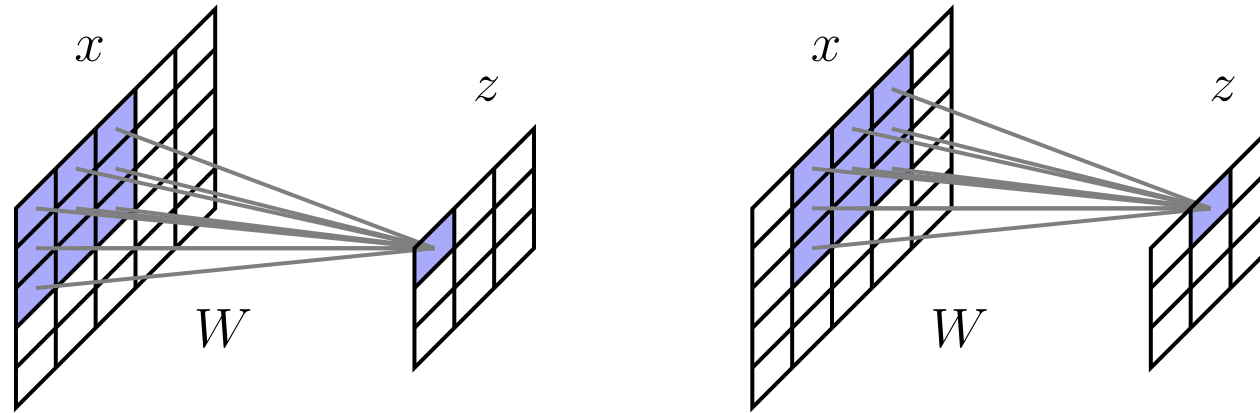
# How convolutions “simplify” deep networks

Convolutions combine two ideas that are well-suited to processing images

1. Require that activations between layers occur only in a “local” manner, and treat hidden layers themselves as spatial images
2. Share weights across all spatial locations



# Advantages of convolutions



*Drastically* reduces the parameter count

- 256x256 grayscale image  $\implies$  256x256 single-channel hidden layer: 4 *billion parameters* in fully connected network to 9 *parameters* in 3x3 convolution

Captures (some) “natural” invariances

- Shifting input image one pixel to the right shifts creates a hidden shifts the hidden unit “image”

# Convolutions in detail

Convolutions are a basic primitive in many computer vision and image processing algorithms

Idea is to “slide” the weights  $k \times k$  weight  $w$  (called a filter, with kernel size  $k$ ) over the image to produce a new image, written  $y = z * w$

$z_{11}$	$z_{12}$	$z_{13}$	$z_{14}$	$z_{15}$
$z_{21}$	$z_{22}$	$z_{23}$	$z_{24}$	$z_{25}$
$z_{31}$	$z_{32}$	$z_{33}$	$z_{34}$	$z_{35}$
$z_{41}$	$z_{42}$	$z_{43}$	$z_{44}$	$z_{45}$
$z_{51}$	$z_{52}$	$z_{53}$	$z_{54}$	$z_{55}$

 $*$ 

$w_{11}$	$w_{12}$	$w_{13}$
$w_{21}$	$w_{22}$	$w_{23}$
$w_{31}$	$w_{32}$	$w_{33}$

 $=$ 

$y_{11}$	$y_{12}$	$y_{13}$
$y_{21}$	$y_{22}$	$y_{23}$
$y_{31}$	$y_{32}$	$y_{33}$

  
$$y_{23} = z_{23}w_{11} + z_{23}w_{12} + z_{23}w_{13} + z_{33}w_{21} + \dots$$

# Convolutions in image processing

Convolutions (typically with *prespecified* filters) are a common operation in many computer vision applications: convolution networks just move to *learned* filters



Original image  $z$



Gaussian blur



Image gradient

$$z * \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 4 & 4 & 1 \end{bmatrix} / 273$$

$$\left( \left( z * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \right)^2 + \left( z * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \right)^2 \right)^{\frac{1}{2}}$$



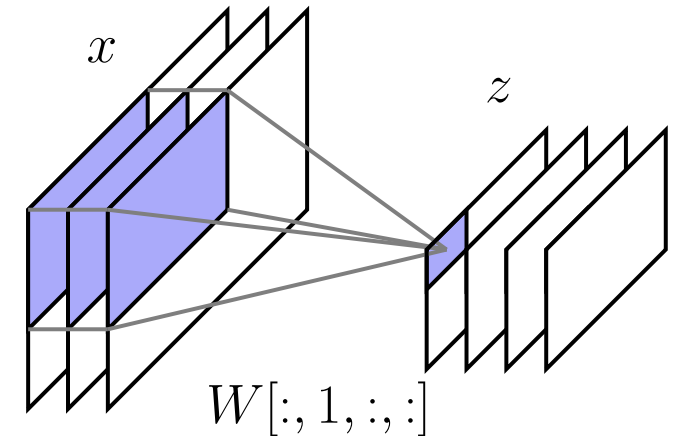
# Convolutions in deep networks

Convolutions in deep networks are virtually always *multi-channel* convolutions: map multi-channel (e.g., RGB) inputs to multi-channel hidden units

- $x \in \mathbb{R}^{h \times w \times c_{in}}$  denotes  $c_{in}$  channel, size  $h \times w$  image input
- $z \in \mathbb{R}^{h \times w \times c_{out}}$  denotes  $c_{out}$  channel, size  $h \times w$  image input
- $W \in \mathbb{R}^{c_{in} \times c_{out} \times k \times k}$  (order 4 tensor) denotes convolutional filter

Multi-channel convolutions contain a convolutional filter for *each input-output channel pair*, single output channel is sum of convolutions over all input channels

$$z[:, :, s] = \sum_{r=1}^{c_{in}} x[:, :, r] * W[r, s, :, :]$$



# Multi-channel convolutions in matrix-vector form

There is, in my view, a more intuitive way to think about multi-channel convolutions: they are a generalization of traditional convolutions with scalar multiplications replaced by matrix-vector products

These are each  $\mathbb{R}^{c_{out} \times c_{in}}$  matrices

$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$
$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$	$x_{25}$
$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$	$x_{35}$
$x_{41}$	$x_{42}$	$x_{43}$	$x_{44}$	$x_{45}$
$x_{51}$	$x_{52}$	$x_{53}$	$x_{54}$	$x_{55}$

$$\begin{matrix} & \begin{matrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{matrix} \\ * & \end{matrix} = \begin{matrix} \begin{matrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \\ z_{31} & z_{32} & z_{33} \end{matrix} \end{matrix}$$

$$z_{22} = W_{11}x_{22} + W_{12}x_{23} + W_{13}x_{24} + W_{21}x_{32} + \dots$$

These are matrix-vector products

These are each vectors in  $\mathbb{R}^{c_{in}}$

# Outline

Convolutional operators in deep networks

Elements of practical convolutions

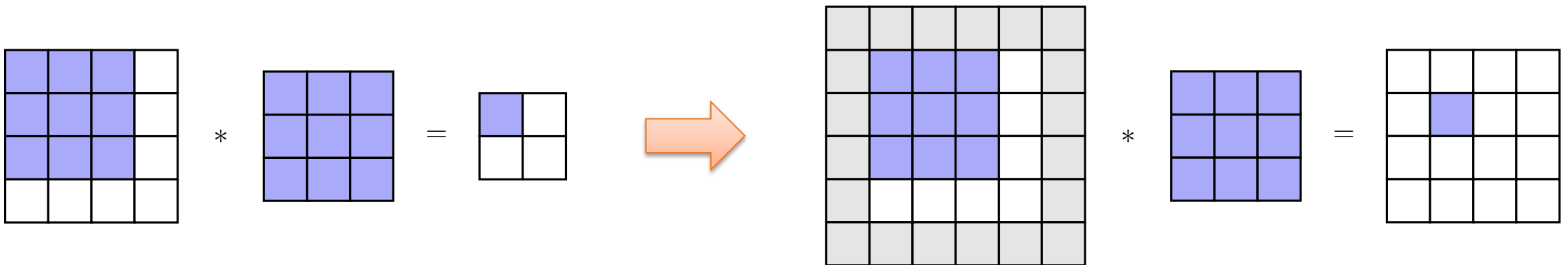
Differentiating convolutions

# Padding

**Challenge:** “Naïve” convolutions produce a smaller output than input image

**Solution:** for (odd) kernel size  $k$ , pad input with  $(k - 1)/2$  zeros on all sides, results in an output that is the same size as the input

- Variants like circular padding, padding with mean values, etc

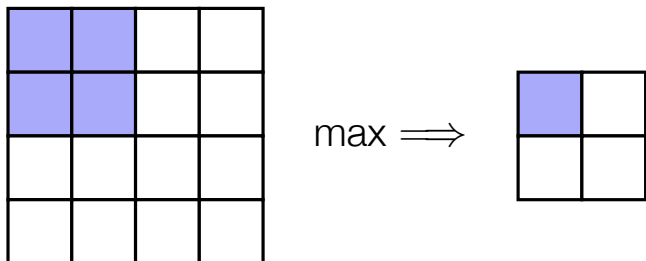


# Strided Convolutions / Pooling

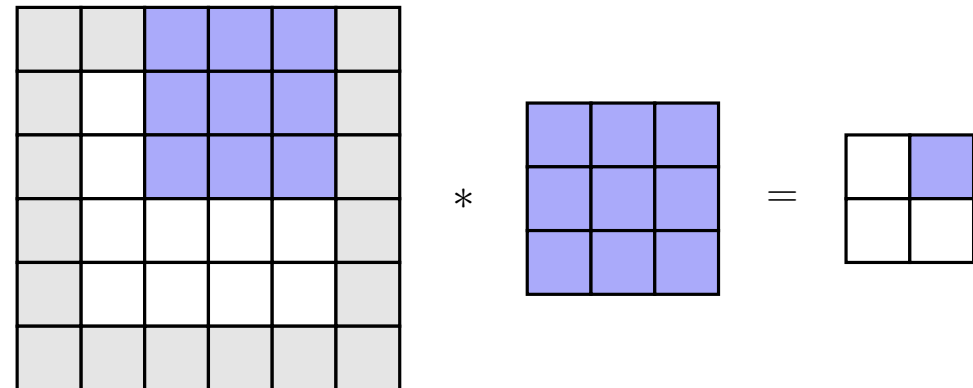
**Challenge:** Convolutions keep the same resolution of the input at each layer, don't naively allow for representations at different "resolutions"

**Solution #1:** incorporate max or average *pooling* layers to aggregate information

**Solution #2:** slide convolutional filter over image in increments  $>1$  (= stride)



Max pooling

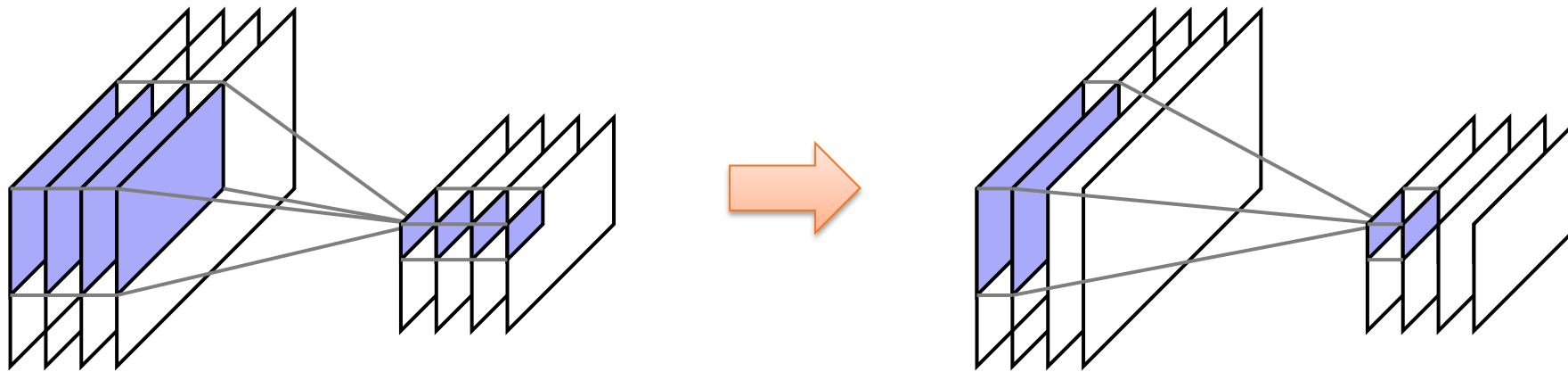


Strided convolution

# Grouped Convolutions

Challenge: for large numbers of input/output channels, filters can still have a large number of weights, can lead to overfitting + slow computation

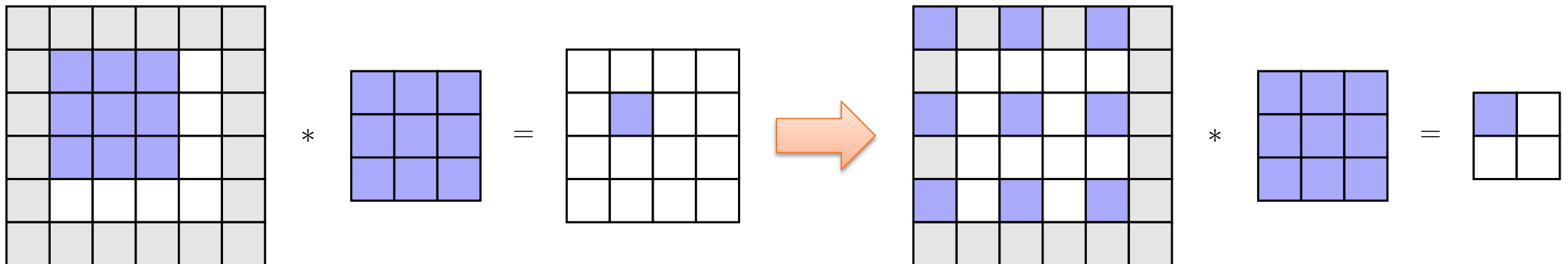
Solution: Group together channels, so that groups of channels in output only depend on corresponding groups of channels in input (equivalently, enforce filter weight matrices to be block-diagonal)



# Dilations

Challenge: Convolutions each have a relatively small receptive field size

Solution: *Dilate* (spread out) convolution filter, so that it covers more of the image (see also: later architectures we will discuss, like self-attention layers); note that getting an image of the same size again requires adding more padding



# Outline

Convolutional operators in deep networks

Elements of practical convolutions

Differentiating convolutions



# What is needed to differentiate convolution?

Recall that in order to integrate any operation into a deep network, we need to be able to multiply by its partial derivatives (adjoint operation)

So if we define our operation

$$z = \text{conv}(x, W)$$

how do we multiply by the adjoints

$$\bar{v} \frac{\partial \text{conv}(x, W)}{\partial W}, \quad \bar{v} \frac{\partial \text{conv}(x, W)}{\partial x}$$

# Refresher on differentiating matrix multiplication

Let's consider the simpler case of a matrix-vector product operation

$$z = Wx$$

Then  $\frac{\partial z}{\partial x} = W$ , so we need to compute the adjoint product

$$v^T W \iff W^T v$$

In other words, for a matrix vector multiply operation  $Wx$ , computing the backwards pass requires multiplying by the *transpose*  $W^T$

So what is the “transpose” of a convolution?

# Convolutions as matrix multiplication: Version 1

To answer this question, consider a 1D convolution to keep things a bit simpler:

$$\begin{bmatrix} 0 & x_1 & x_2 & x_3 & x_4 & x_5 & 0 \end{bmatrix} * \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 & z_5 \end{bmatrix}$$

We can write a 1D convolution  $x * w$  (e.g., with zero padding) as a matrix multiplication  $\widehat{W}x$  for some  $\widehat{W}$  properly defined in terms of the filter  $w$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \underbrace{\begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{bmatrix}}_{\widehat{W}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

# The adjoint of a convolution

So how can we multiply by the transpose  $\widehat{W}^T$ ?

Written out as in the previous slide, it's quite easy:

$$\widehat{W}^T = \begin{bmatrix} w_2 & w_1 & 0 & 0 & 0 \\ w_3 & w_2 & w_1 & 0 & 0 \\ 0 & w_3 & w_2 & w_1 & 0 \\ 0 & 0 & w_3 & w_2 & w_1 \\ 0 & 0 & 0 & w_3 & w_2 \end{bmatrix}$$

But notice that the operation  $\widehat{W}^T v$  is itself just a convolution with the “flipped” filter:  $[w_3 \ w_2 \ w_1]$ ;  $\implies$  adjoint operator  $\bar{v} \frac{\partial \text{conv}(x, W)}{\partial x}$  just requires convolving  $\bar{v}$  with a the flipped  $W$ !

# Convolutions as matrix multiplication: Version 2

What about the other adjoint,  $\bar{v} \frac{\partial \text{conv}(x, W)}{\partial W}$ ?

For this term, observe that we can *also* write the convolution as a matrix-vector product treating the *filter* as the vector

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \begin{bmatrix} 0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ x_4 & x_5 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

So adjoint requires multiplying by the transpose of this  $x$ -based matrix (actually a relatively practical approach, see future lecture on the “im2col” operation)