

CPEN 455: Deep Learning

Lecture 5.2: Convolutional Neural Networks II

Renjie Liao

University of British Columbia

2026 Winter Term 1

October 7, 2026

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - **Transposed Convolution**
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

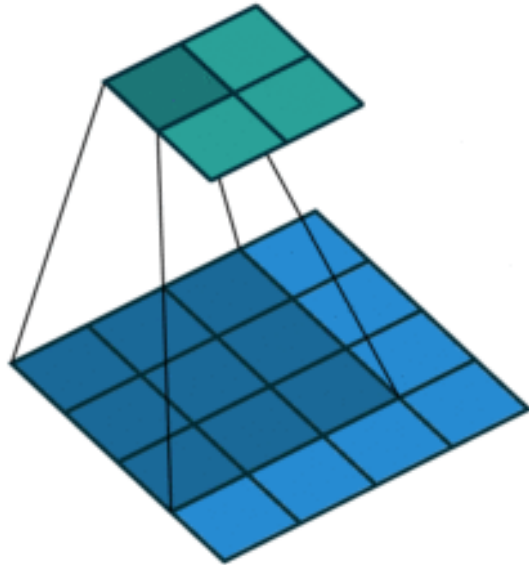
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

2D example: a 3x3 kernel

Shapes:
4x4 $\rightarrow$ 2x2



2D Convolution
(stride=1, padding=0)

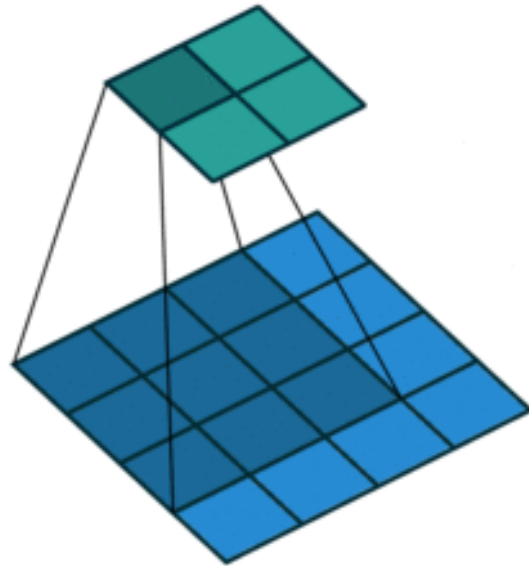
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

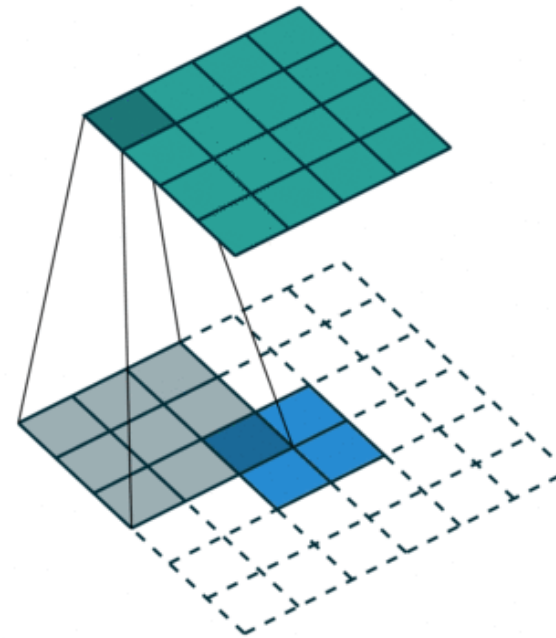
2D example: a 3×3 kernel

Shapes:
 $4 \times 4 \rightarrow 2 \times 2$



2D Convolution
(stride=1, padding=0)

Shapes:
 $2 \times 2 \rightarrow 4 \times 4$



Expanded-grid convolution
(stride=1, border=2)

2D Transposed Convolution

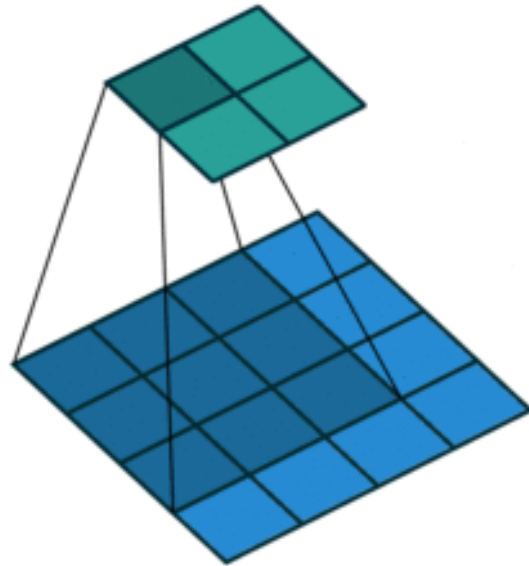
Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

2D example: a 3x3 kernel

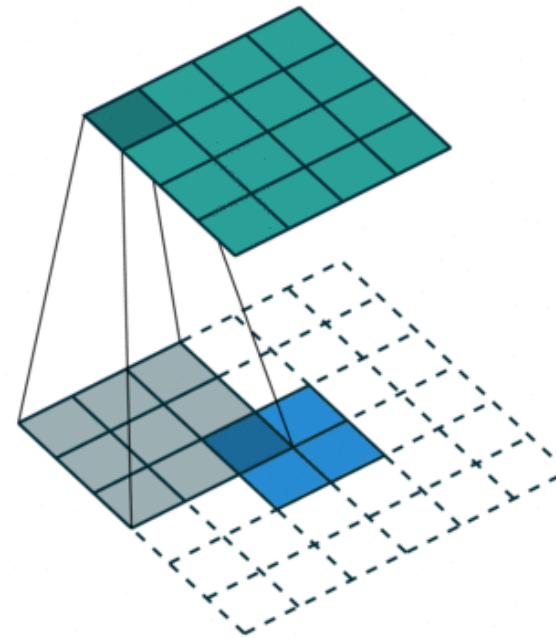
Adjoint: for real matrices, the transpose maps output-space vectors back to input space using the same weights. It generally does not recover the original input.

Shapes:
 $4 \times 4 \rightarrow 2 \times 2$



2D Convolution
(stride=1, padding=0)

Shapes:
 $2 \times 2 \rightarrow 4 \times 4$



Expanded-grid convolution
(stride=1, border=2)

2D Transposed Convolution

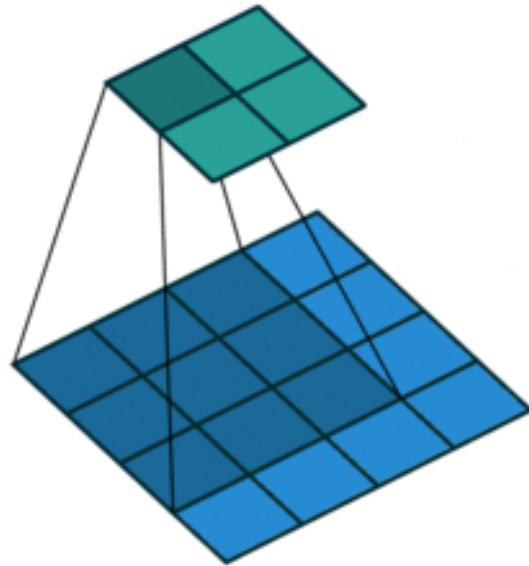
Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

2D example: a 3x3 kernel

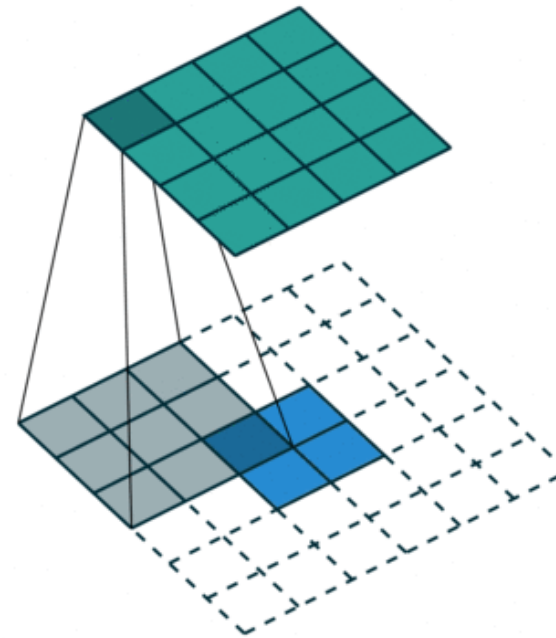
- The transpose is the adjoint of the convolution matrix, not its inverse.
- Deconvolution solves an inverse problem, which may be ill-posed.

Shapes:
4x4 $\rightarrow$ 2x2



2D Convolution
(stride=1, padding=0)

Shapes:
2x2 $\rightarrow$ 4x4



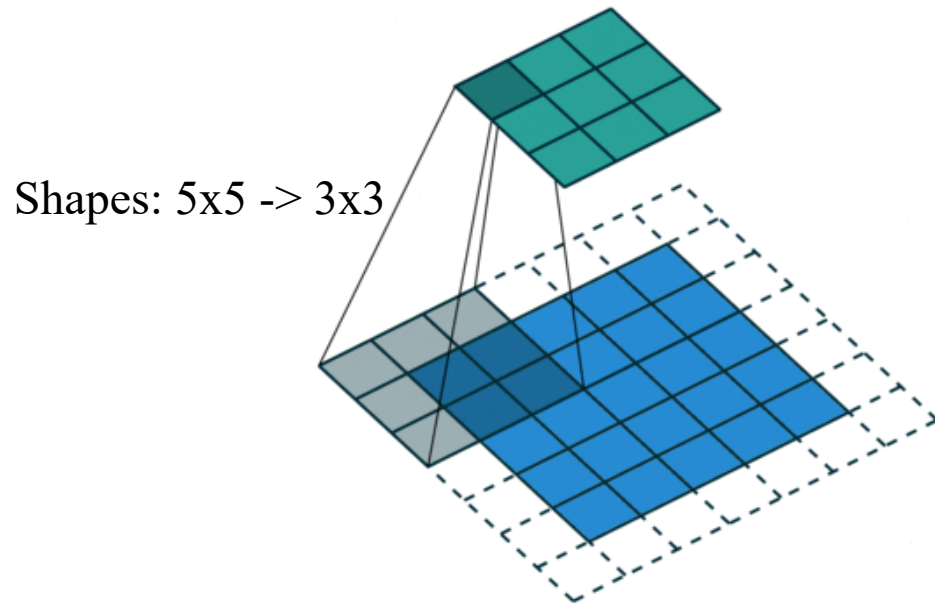
Expanded-grid convolution
(stride=1, border=2)

2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

Yes, transposed convolution!

Another 2D example: a 3x3 kernel



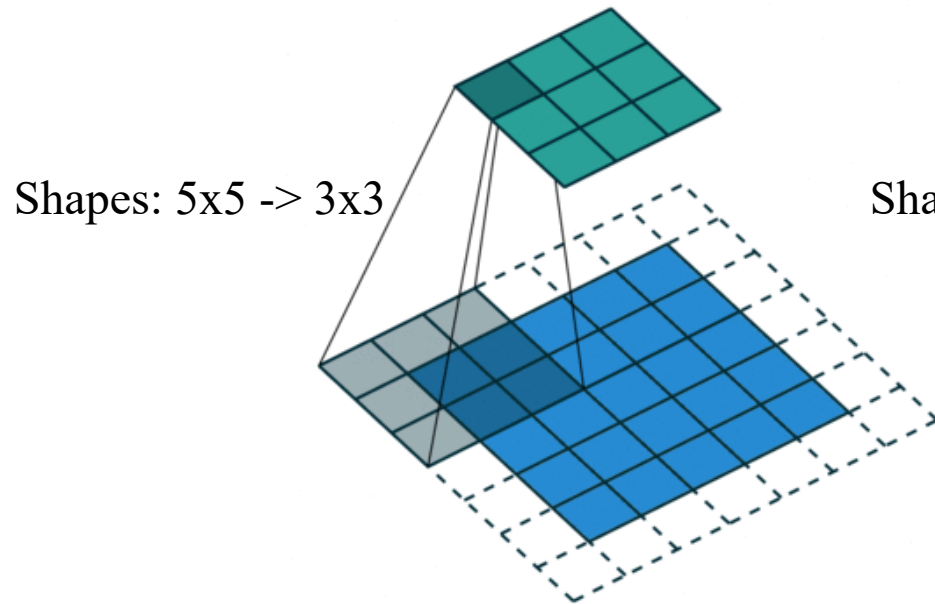
2D Convolution
(stride=2, padding=1)

2D Transposed Convolution

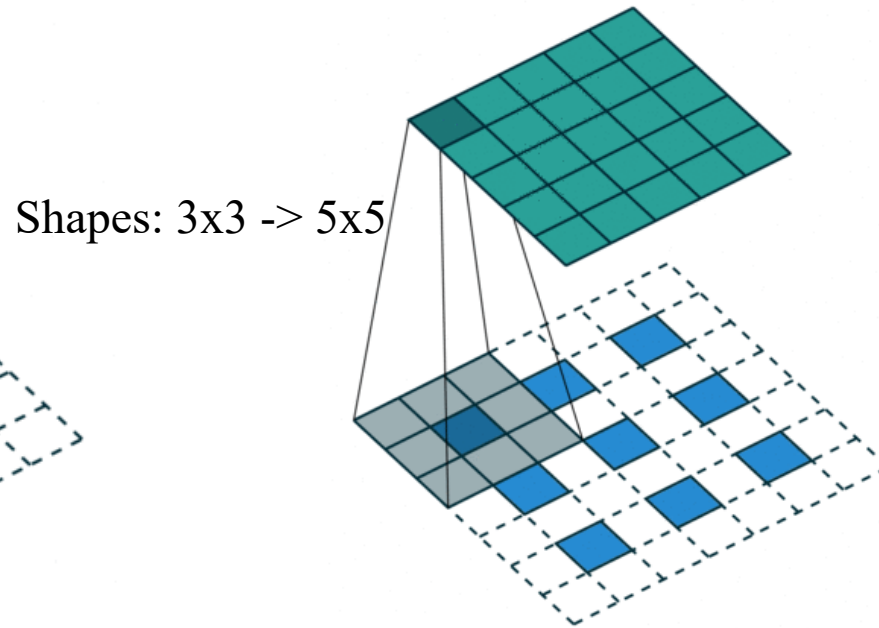
Convolution can reduce input size, e.g., with $\text{stride} > 1$. Can it also enlarge the input?

Yes, transposed convolution!

Another 2D example: a 3×3 kernel



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

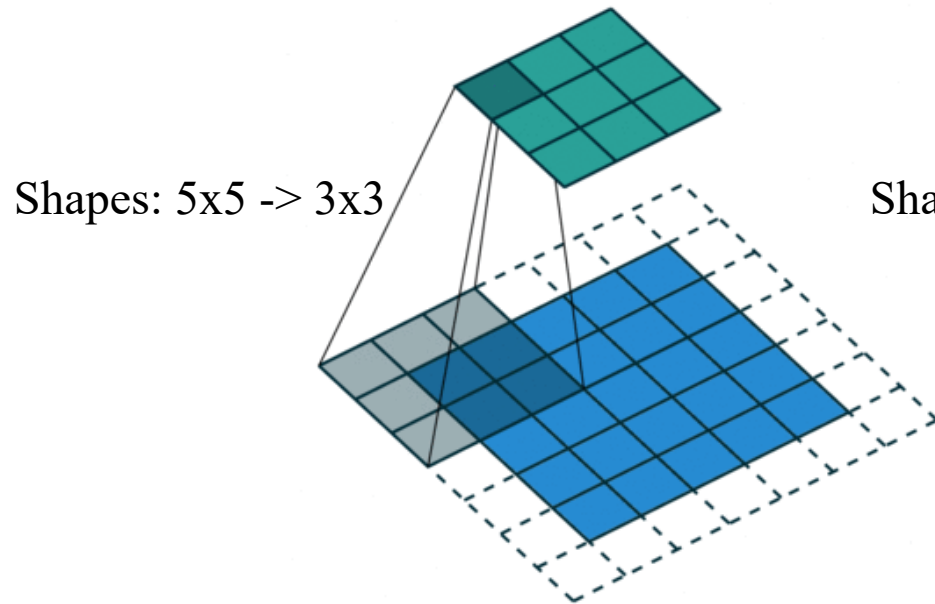
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

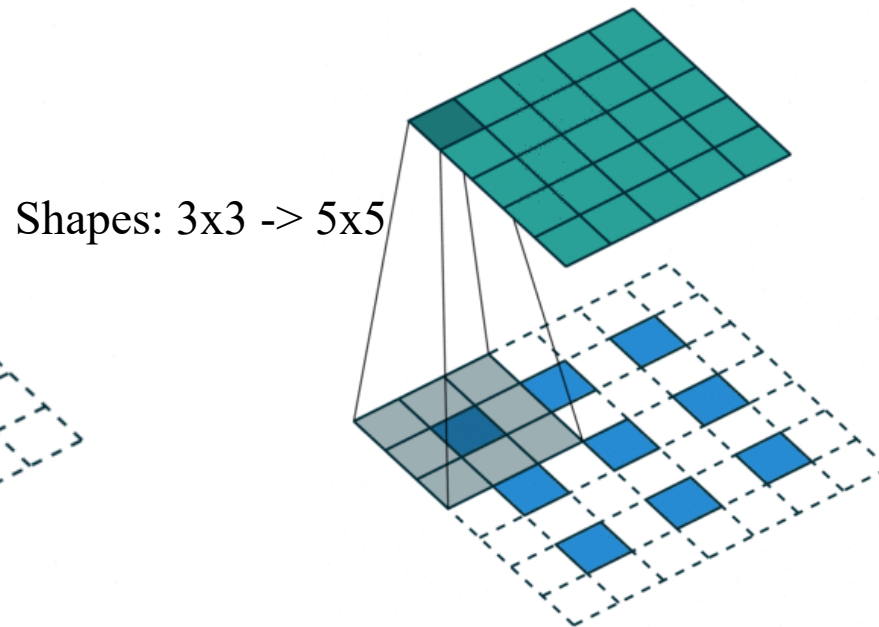
Yes, transposed convolution!

Another 2D example: a 3×3 kernel

Also called fractionally strided convolution!



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

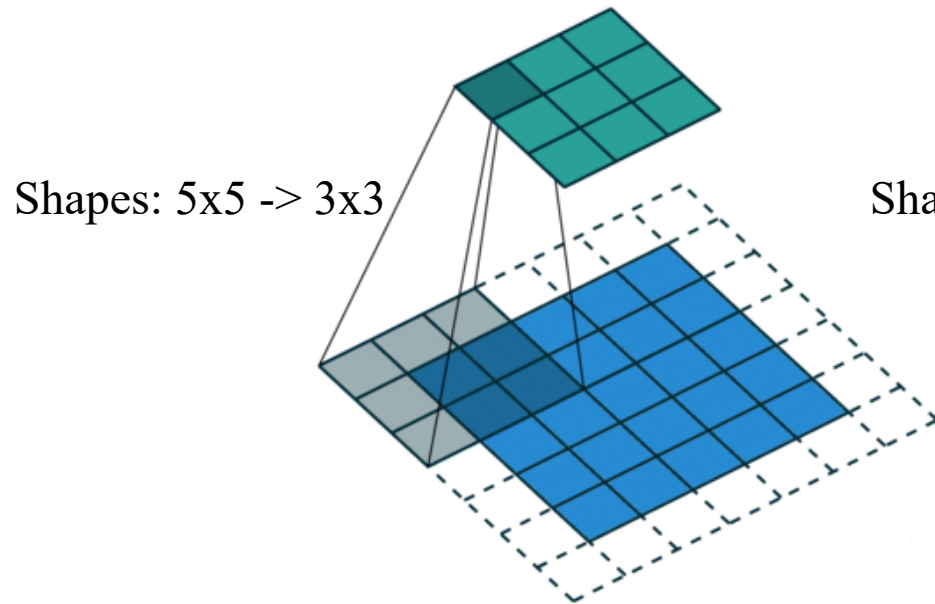
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

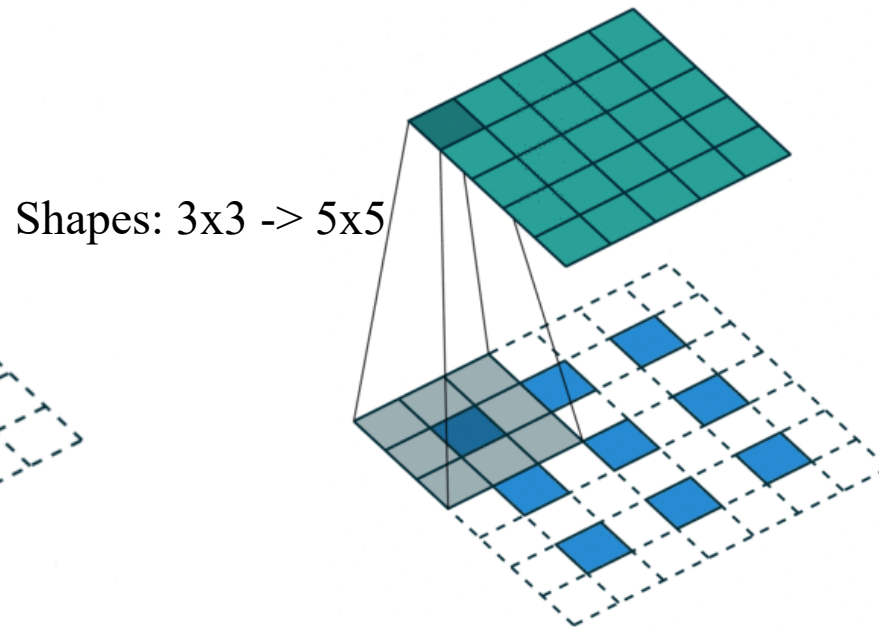
Yes, transposed convolution!

Another 2D example: a 3×3 kernel

Efficient implementations need not materialize the inserted zeros.



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

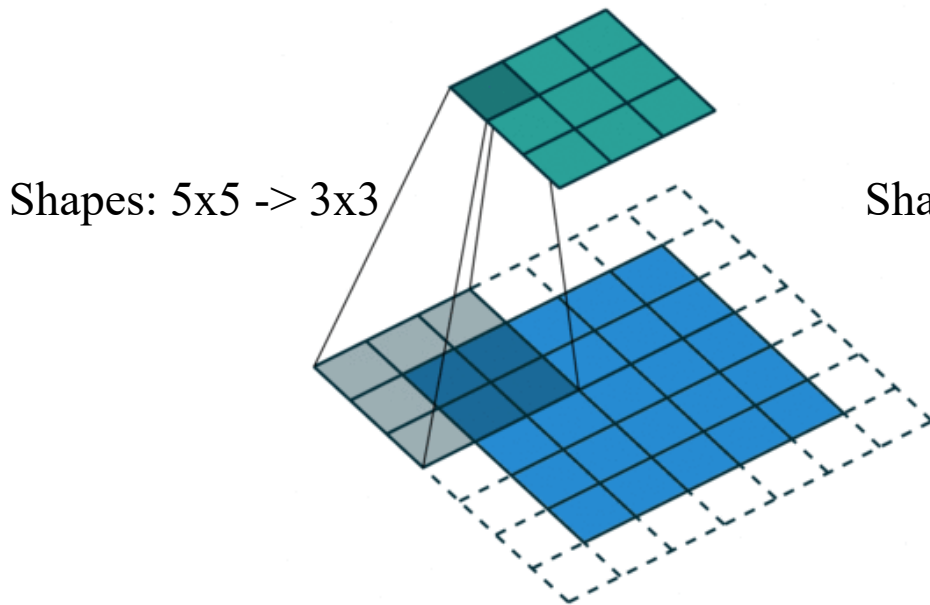
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

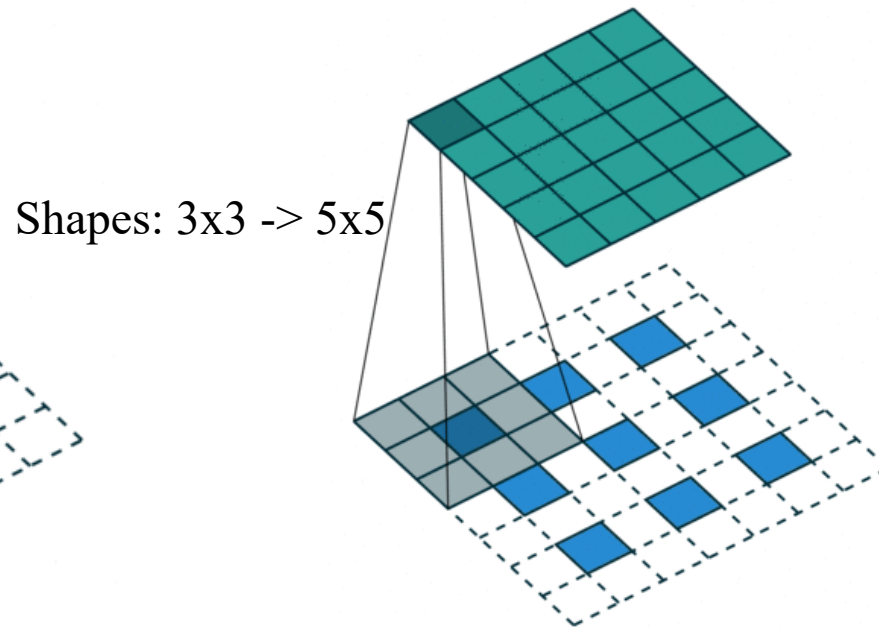
Yes, transposed convolution!

Another 2D example: a 3x3 kernel

Efficient implementations need not materialize the inserted zeros.
The Jacobian transpose maps vectors back to input space:



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

$$\mathbf{y} = \mathbf{W}\mathbf{x} \quad \text{Filter} \rightarrow \text{Matrix}$$
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$$
$$\hat{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^\top y$$

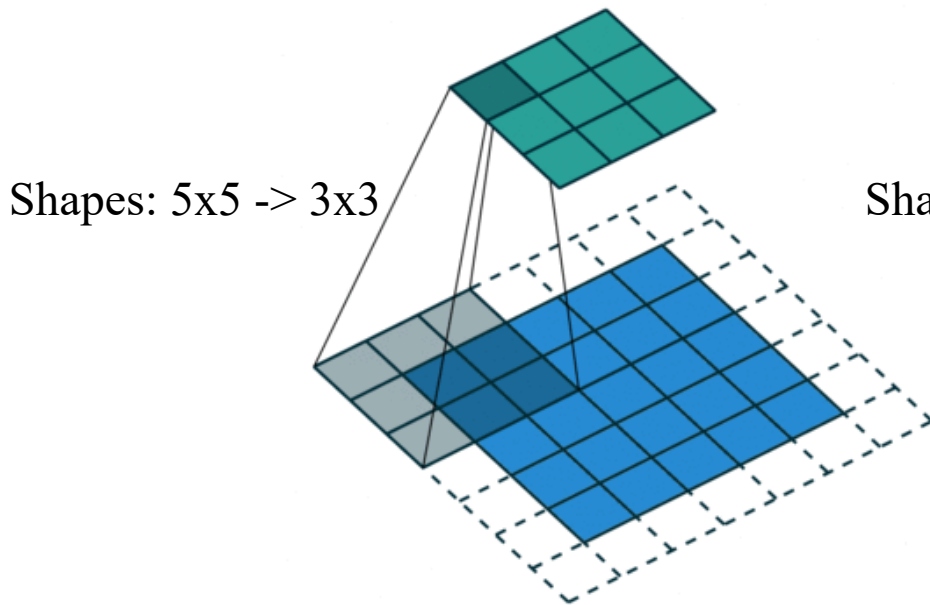
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

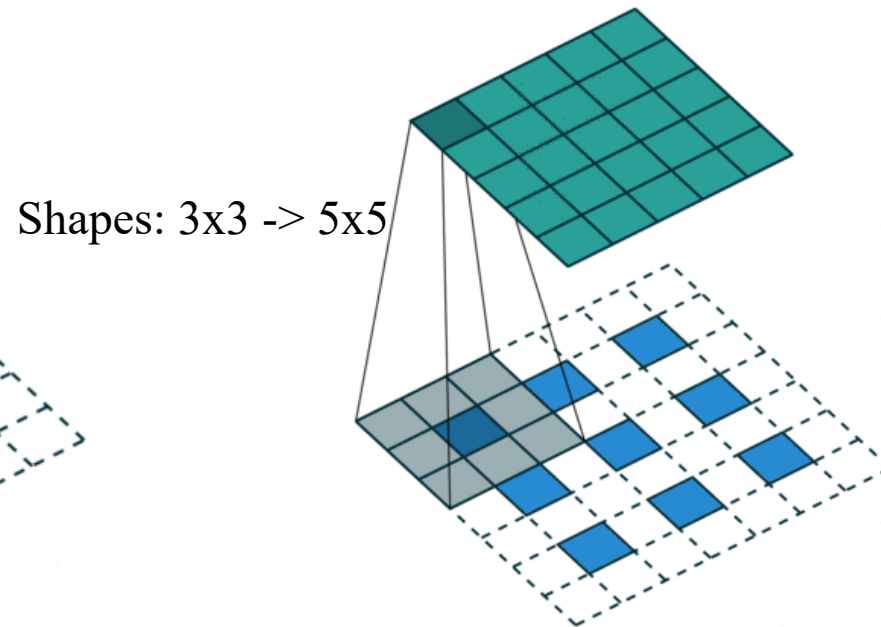
Yes, transposed convolution!

Another 2D example: a 3x3 kernel

Efficient implementations need not materialize the inserted zeros.
The Jacobian transpose maps vectors back to input space:



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

$$\mathbf{y} = \mathbf{W}\mathbf{x} \quad \text{Filter} \rightarrow \text{Matrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$$

$$\hat{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^\top \mathbf{y}$$

PyTorch ConvTranspose2d:
stride=2, padding=1 [14].
Stride 1 describes
convolution on the expanded
grid above.

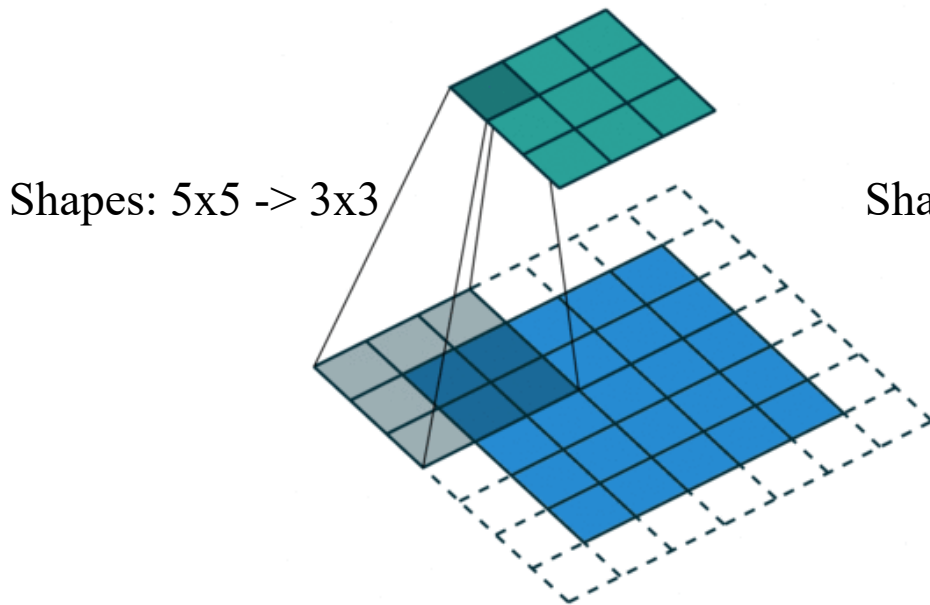
2D Transposed Convolution

Convolution can reduce input size, e.g., with stride > 1 . Can it also enlarge the input?

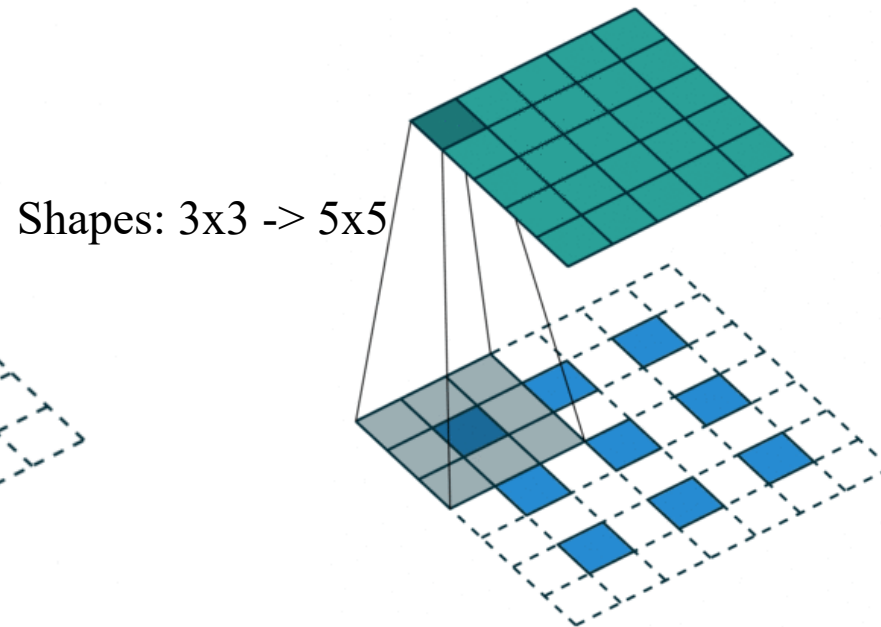
Yes, transposed convolution!

Another 2D example: a 3x3 kernel

Efficient implementations need not materialize the inserted zeros.
The Jacobian transpose maps vectors back to input space:



2D Convolution
(stride=2, padding=1)



Expanded-grid convolution
(stride=1, padding=1)

$$\mathbf{y} = \mathbf{W}\mathbf{x} \quad \text{Filter} \rightarrow \text{Matrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$$

$$\hat{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^\top \mathbf{y}$$

PyTorch ConvTranspose2d:
stride=2, padding=1 [14].
Stride 1 describes
convolution on the expanded
grid above.

Transposed Convolution and Backpropagation

For a fixed filter, vectorized convolution is a linear map (bias omitted).

$$\mathbf{y} = W\mathbf{x}, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = W \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad W \in \mathbb{R}^{m \times n}$$

Let the scalar loss L depend on the convolution output.

Transposed Convolution and Backpropagation

For a fixed filter, vectorized convolution is a linear map (bias omitted).

$$\mathbf{y} = W\mathbf{x}, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = W \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad W \in \mathbb{R}^{m \times n}$$

Let the scalar loss L depend on the convolution output.

Backpropagation applies the chain rule:

$$\frac{\partial L}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^\top \frac{\partial L}{\partial \mathbf{y}} = W^\top \frac{\partial L}{\partial \mathbf{y}}$$

The input gradient is the transpose matrix times the upstream output gradient.

Transposed Convolution and Backpropagation

For a fixed filter, vectorized convolution is a linear map (bias omitted).

$$\mathbf{y} = W\mathbf{x}, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = W \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad W \in \mathbb{R}^{m \times n}$$

Let the scalar loss L depend on the convolution output.

Backpropagation applies the chain rule:

$$\frac{\partial L}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^\top \frac{\partial L}{\partial \mathbf{y}} = W^\top \frac{\partial L}{\partial \mathbf{y}}$$

The input gradient is the transpose matrix times the upstream output gradient.

Treat the backward-input computation as an operator on any output-space vector:

$$T(\mathbf{u}) = W^\top \mathbf{u}$$

$\mathbf{u}$ can be any tensor with the forward output's shape.

$T(\mathbf{u})$ has the forward input's shape; backprop uses the upstream gradient [14].

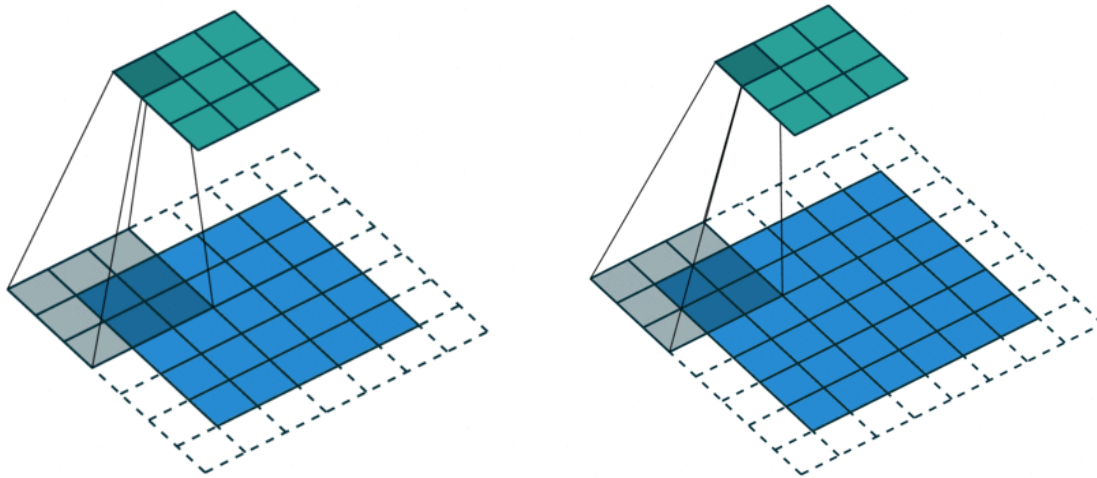
2D Transposed Convolution

Kernel size $k=3$, stride $s=2$, padding $p=1$, dilation $d=1$: forward inputs 5 or 6 give output 3.

Transpose: $n_{\text{in}} = 3, \quad n_{\text{out}} \in \{5, 6\}$

Shapes: $5 \times 5 \rightarrow 3 \times 3$

Shapes: $6 \times 6 \rightarrow 3 \times 3$



2D Convolution
(stride=2, padding=1)

2D Transposed Convolution

Kernel size $k=3$, stride $s=2$, padding $p=1$, dilation $d=1$: forward inputs 5 or 6 give output 3.

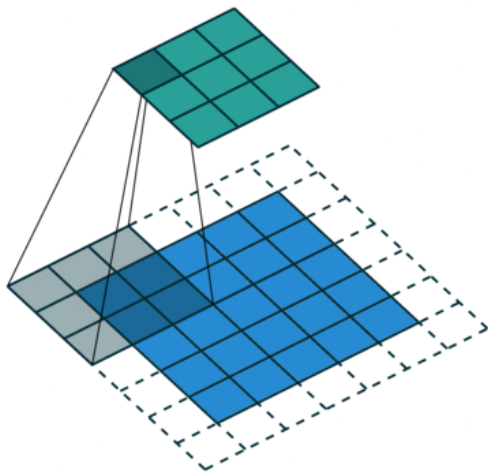
Transpose: $n_{\text{in}} = 3$, $n_{\text{out}} \in \{5, 6\}$ output_padding o selects size; it does not append zeros [14].

Shapes: $5 \times 5 \rightarrow 3 \times 3$

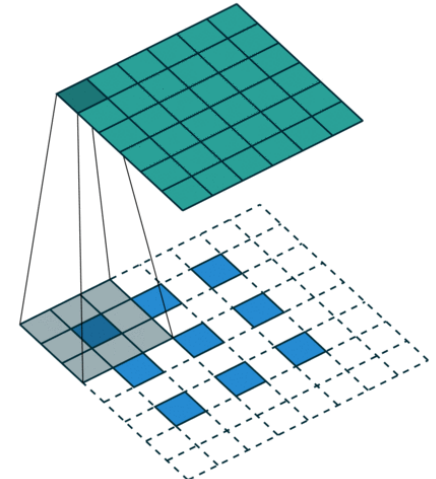
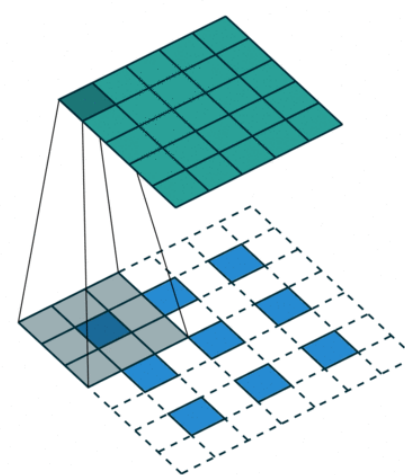
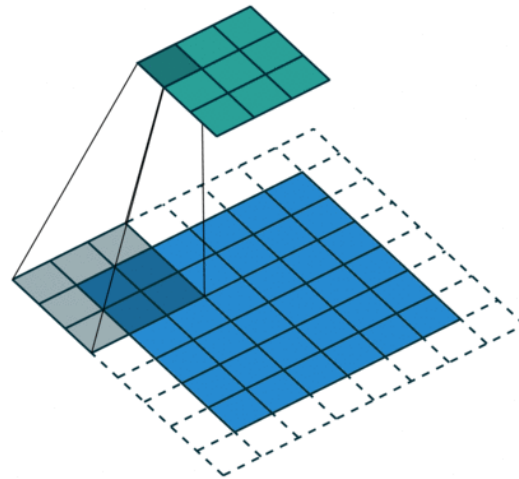
Shapes: $6 \times 6 \rightarrow 3 \times 3$

Shapes: $3 \times 3 \rightarrow 5 \times 5$

Shapes: $3 \times 3 \rightarrow 6 \times 6$



2D Convolution
(stride=2, padding=1)



2D Transposed Convolution

2D Transposed Convolution

Kernel size $k=3$, stride $s=2$, padding $p=1$, dilation $d=1$: forward inputs 5 or 6 give output 3.

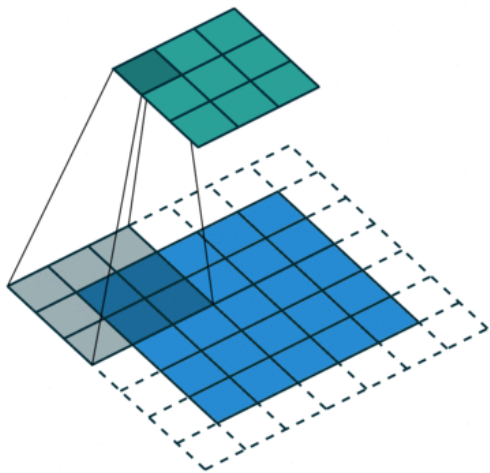
Transpose: $n_{in} = 3$, $n_{out} \in \{5, 6\}$ output_padding o selects size; it does not append zeros [14].

Shapes: $5 \times 5 \rightarrow 3 \times 3$

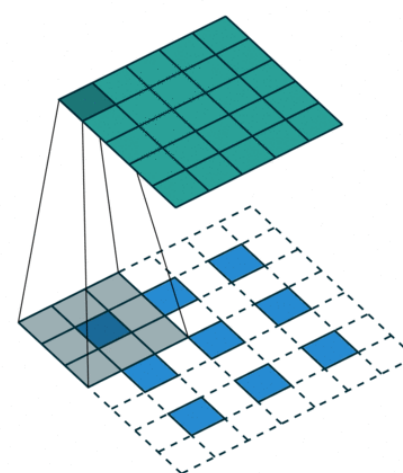
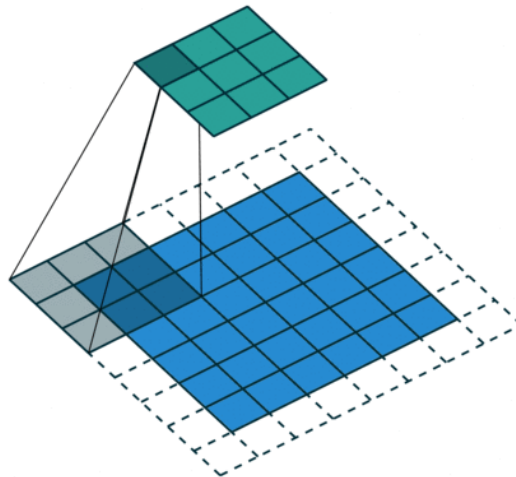
Shapes: $6 \times 6 \rightarrow 3 \times 3$

Shapes: $3 \times 3 \rightarrow 5 \times 5$

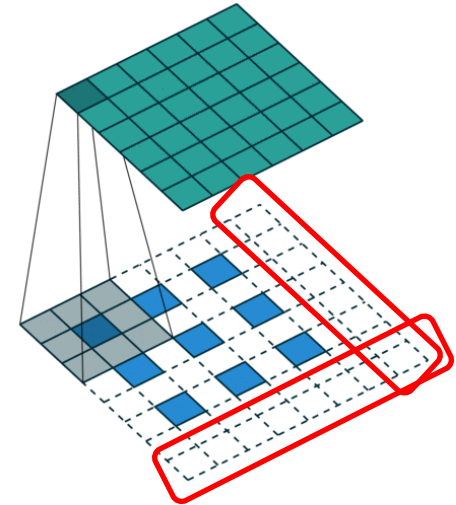
Shapes: $3 \times 3 \rightarrow 6 \times 6$



2D Convolution
(stride=2, padding=1)



2D Transposed Convolution
output padding=0



output padding=1

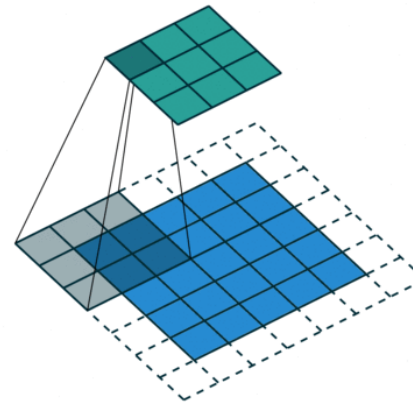
Output Padding

Kernel size $k=3$, stride $s=2$, forward padding $p=1$.

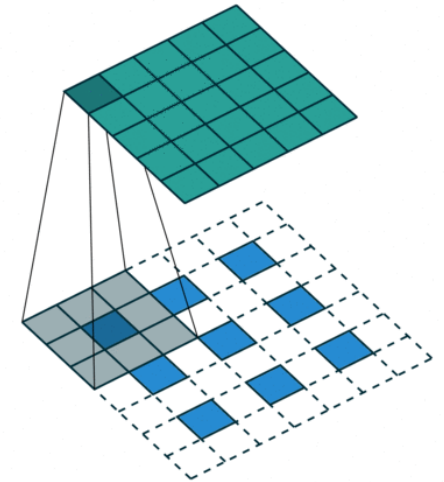
Transpose input/output side lengths: $n_{\text{in}} = 3$, $n_{\text{out}} = 5$ ($o = 0$)

Insert zeros between the input positions:

$$n_{\text{expanded}} = (n_{\text{in}} - 1)s + 1 = 5$$



Forward convolution
 $5 \times 5 \rightarrow 3 \times 3$



Transposed convolution
 $3 \times 3 \rightarrow 5 \times 5$ ($o=0$)

Output Padding

Kernel size $k=3$, stride $s=2$, forward padding $p=1$.

Transpose input/output side lengths: $n_{\text{in}} = 3$, $n_{\text{out}} = 5$ ($o = 0$)

Insert zeros between the input positions:

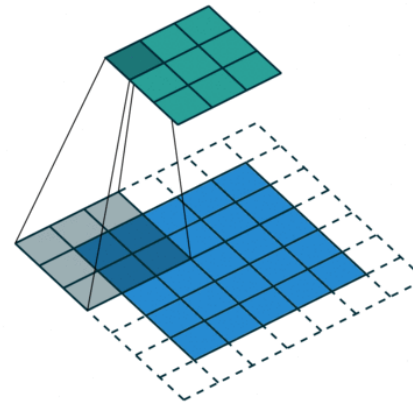
$$n_{\text{expanded}} = (n_{\text{in}} - 1)s + 1 = 5$$

q = number of border zeros per side of this grid.

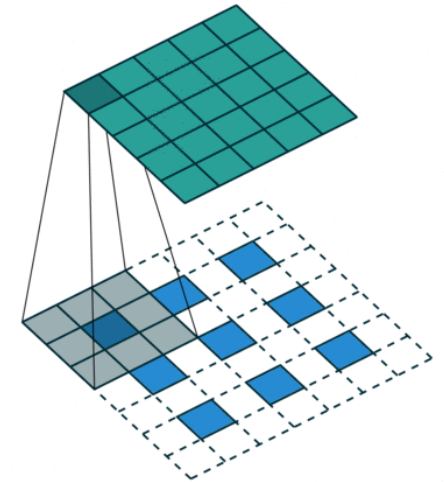
Slide the length-3 kernel with stride 1:

$$n_{\text{out}} = 5 + 2q - 3 + 1$$

The transpose output must match the original 5×5 input shape ($o=0$):



Forward convolution
 $5 \times 5 \rightarrow 3 \times 3$



Transposed convolution
 $3 \times 3 \rightarrow 5 \times 5$ ($o=0$)

Output Padding

Kernel size $k=3$, stride $s=2$, forward padding $p=1$.

Transpose input/output side lengths: $n_{\text{in}} = 3$, $n_{\text{out}} = 5$ ($o = 0$)

Insert zeros between the input positions:

$$n_{\text{expanded}} = (n_{\text{in}} - 1)s + 1 = 5$$

q = number of border zeros per side of this grid.

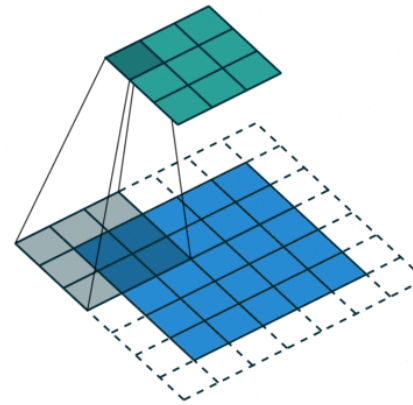
Slide the length-3 kernel with stride 1:

$$n_{\text{out}} = 5 + 2q - 3 + 1$$

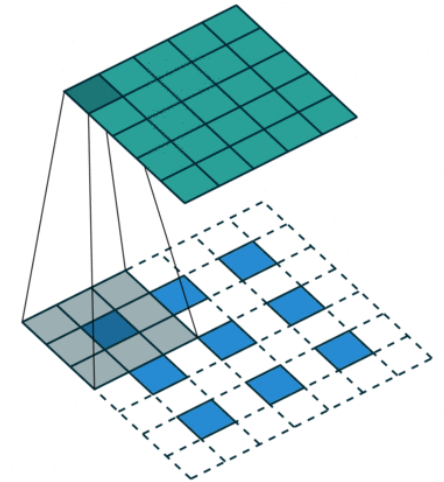
The transpose output must match the original 5×5 input shape ($o=0$):

$$5 + 2q - 3 + 1 = 5 \implies q = 1$$

p is the forward padding; q is the expanded-grid border.



Forward convolution
 $5 \times 5 \rightarrow 3 \times 3$



Transposed convolution
 $3 \times 3 \rightarrow 5 \times 5$ ($o=0$)

Output Padding

Kernel size $k=3$, stride $s=2$, forward padding $p=1$.

Transpose input/output side lengths: $n_{\text{in}} = 3$, $n_{\text{out}} = 5$ ($o = 0$)

Insert zeros between the input positions:

$$n_{\text{expanded}} = (n_{\text{in}} - 1)s + 1 = 5$$

Expanded-grid count ($o=0$):

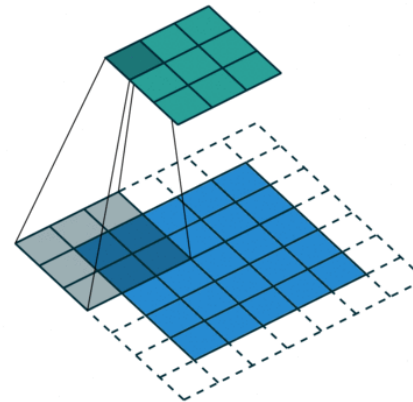
$$n_{\text{out}} = n_{\text{expanded}} + 2q - k + 1$$

Original input size from forward convolution:

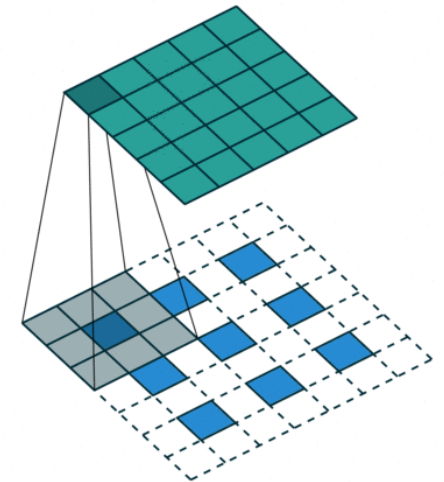
$$n_{\text{out}} = n_{\text{expanded}} + k - 1 - 2p$$

Equating the counts gives:

$$q = k - 1 - p = 3 - 1 - 1 = 1$$



Forward convolution
 $5 \times 5 \rightarrow 3 \times 3$



Transposed convolution
 $3 \times 3 \rightarrow 5 \times 5$ ($o=0$)

Output Padding

Kernel size $k=3$, stride $s=2$, forward padding $p=1$.

Transpose input/output side lengths: $n_{\text{in}} = 3$, $n_{\text{out}} \in \{5, 6\}$

Insert zeros between the input positions:

$$n_{\text{expanded}} = (n_{\text{in}} - 1)s + 1 = 5$$

Expanded-grid count ($o=0$):

$$n_{\text{out}} = n_{\text{expanded}} + 2q - k + 1$$

Original input size from forward convolution:

$$n_{\text{out}} = n_{\text{expanded}} + k - 1 - 2p$$

Equating the counts gives:

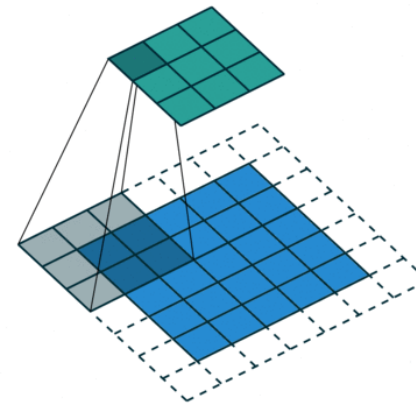
$$q = k - 1 - p = 3 - 1 - 1 = 1$$

Output size with output_padding o :

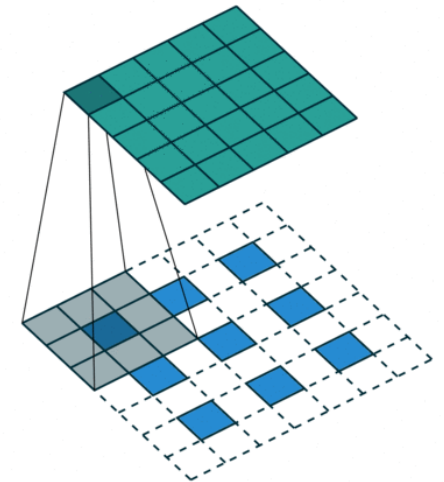
$$n_{\text{out}} = (n_{\text{in}} - 1)s - 2p + k + o = 5 + o$$

$o=0$ gives size 5; $o=1$ retains an extra bottom row/right column, giving size 6.

These coordinates contain computed values, rather than appended zeros [14].



Forward convolution
 $5 \times 5 \rightarrow 3 \times 3$



Transposed convolution
 $3 \times 3 \rightarrow 5 \times 5$ ($o=0$)

2D Transposed Convolution

Take the API in PyTorch as an example

```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None, dtype=None) \[SOURCE\]
```



Applies a 2D transposed convolution operator over an input image composed of several input planes.

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations [here](#) and the [Deconvolutional Networks](#) paper.

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

- `stride` controls the stride for the cross-correlation.
- `padding` controls the amount of implicit zero padding on both sides for `dilation * (kernel_size - 1) - padding` number of points. See note below for details.
- `output_padding` controls the additional size added to one side of the output shape. See note below for details.

[stride=2](#) inserts zeros between input elements in the expanded-grid view.

2D Transposed Convolution

Take the API in PyTorch as an example

```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None,  
dtype=None) \[SOURCE\]
```



Applies a 2D transposed convolution operator over an input image composed of several input planes.

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations [here](#) and the [Deconvolutional Networks](#) paper.

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

- `stride` controls the stride for the cross-correlation.
- `padding` controls the amount of implicit zero padding on both sides for `dilation * (kernel_size - 1) - padding` number of points. See note below for details.
- `output_padding` controls the additional size added to one side of the output shape. See note below for details.

For the 3×3 kernel, `padding=1` means one boundary zero on each side of the expanded grid.

2D Transposed Convolution

Take the API in PyTorch as an example

```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None,  
dtype=None) \[SOURCE\]
```



Applies a 2D transposed convolution operator over an input image composed of several input planes.

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations [here](#) and the [Deconvolutional Networks](#) paper.

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

- `stride` controls the stride for the cross-correlation.
- `padding` controls the amount of implicit zero padding on both sides for `dilation * (kernel_size - 1) - padding` number of points. See note below for details.
- `output_padding` controls the additional size added to one side of the output shape. See note below for details.

`output_padding` selects the output shape, not zero-valued output pixels.

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - **Dilated Convolution**
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

2D Dilated Convolution

Can we enlarge the receptive field without increasing the number of parameters?

2D Dilated Convolution

Can we enlarge the receptive field without increasing the number of parameters?

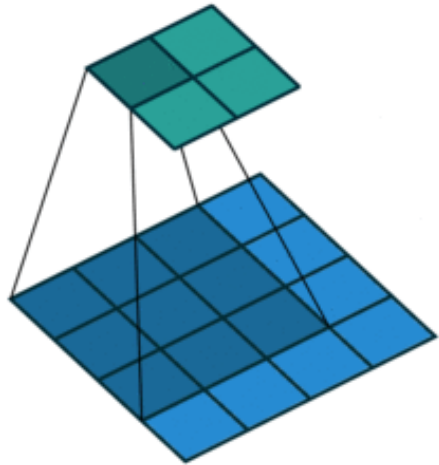
Yes, dilated (atrous) convolution!

2D Dilated Convolution

Can we enlarge the receptive field without increasing the number of parameters?

Yes, dilated (atrous) convolution!

Suppose we have a 2D convolution:



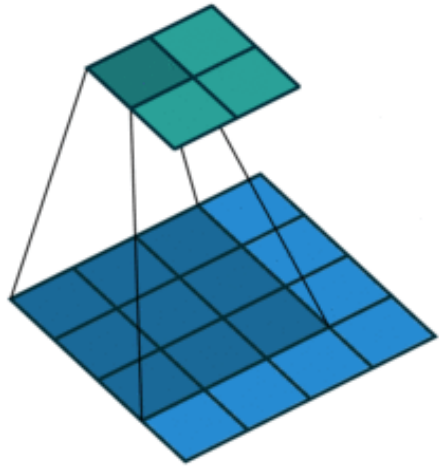
2D Convolution
(stride=1, padding=0)

2D Dilated Convolution

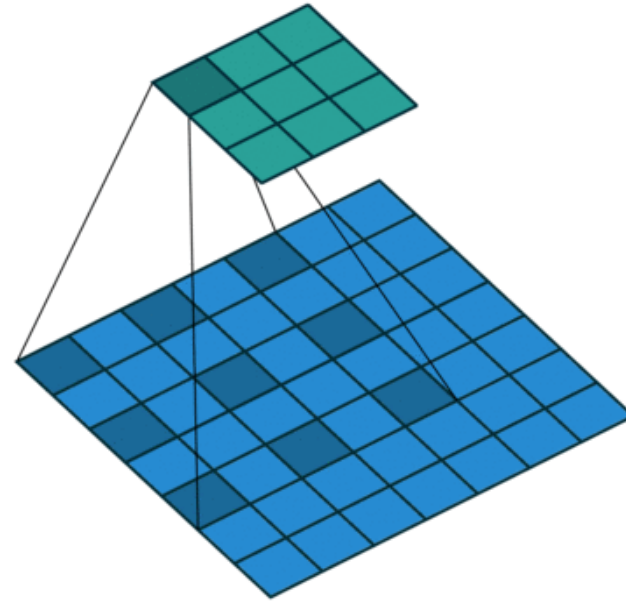
Can we enlarge the receptive field without increasing the number of parameters?

Yes, dilated (atrous) convolution!

Suppose we have a 2D convolution:



2D Convolution
(stride=1, padding=0)



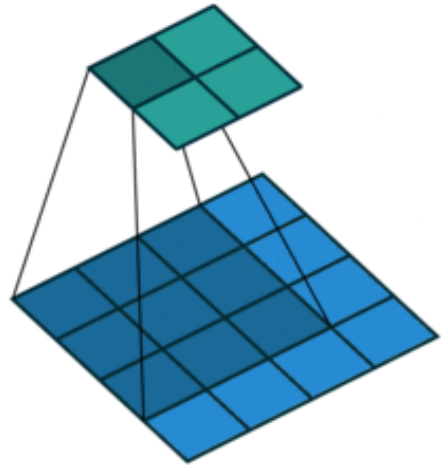
2D Dilated Convolution
(stride=1, padding=0, dilation=2)

2D Dilated Convolution

Can we enlarge the receptive field without increasing the number of parameters?

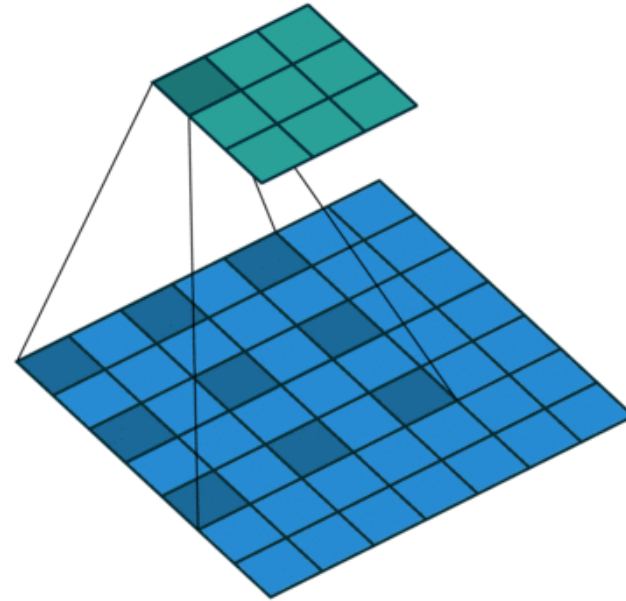
Yes, dilated (atrous) convolution!

Suppose we have a 2D convolution:



2D Convolution
(stride=1, padding=0)

By using dilated kernels, we effectively increase the receptive field (the region of input that affects the output)!



2D Dilated Convolution
(stride=1, padding=0, dilation=2)

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - **Grouped Convolution**
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

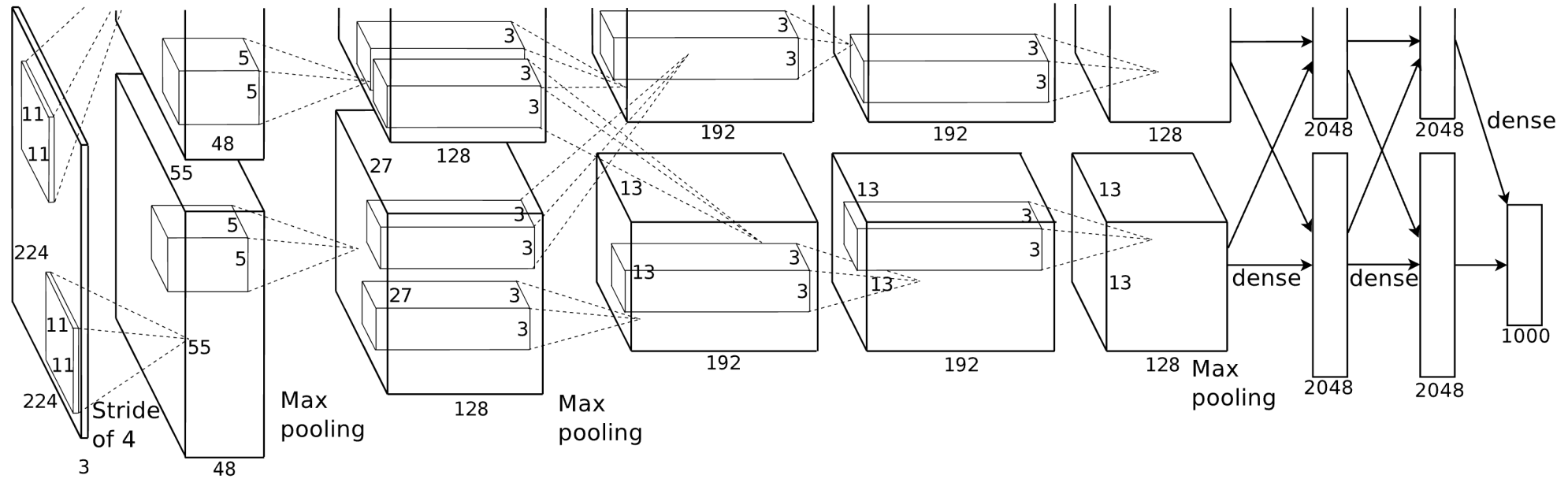
Yes, grouped convolution!

Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

Yes, grouped convolution!

It was used in AlexNet [2]:

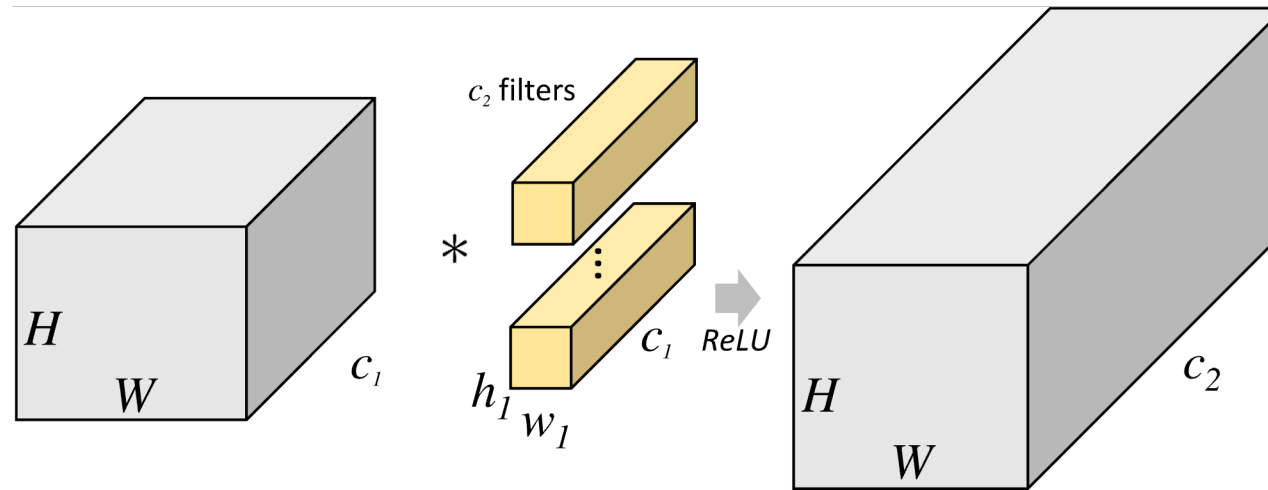


Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

Yes, grouped convolution!

Suppose we have a convolution layer applied to input (shape $H \times W \times c_1$):



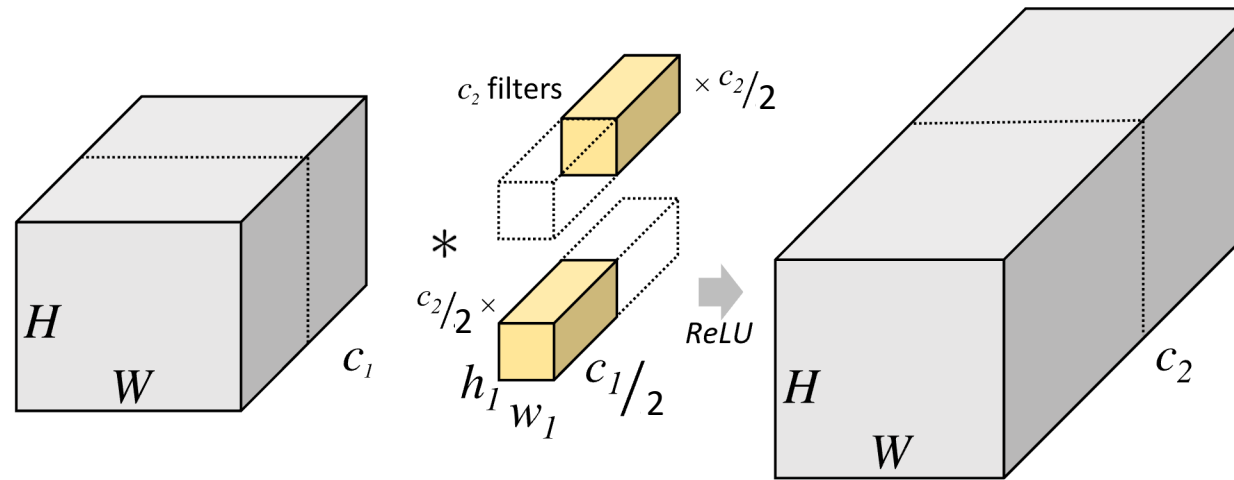
We have c_2 filters with kernel size $h_1 \times w_1 \times c_1$

Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

Yes, grouped convolution!

Now we switch to a grouped (# groups=2) convolution layer applied to the same input (shape $H \times W \times c_1$):



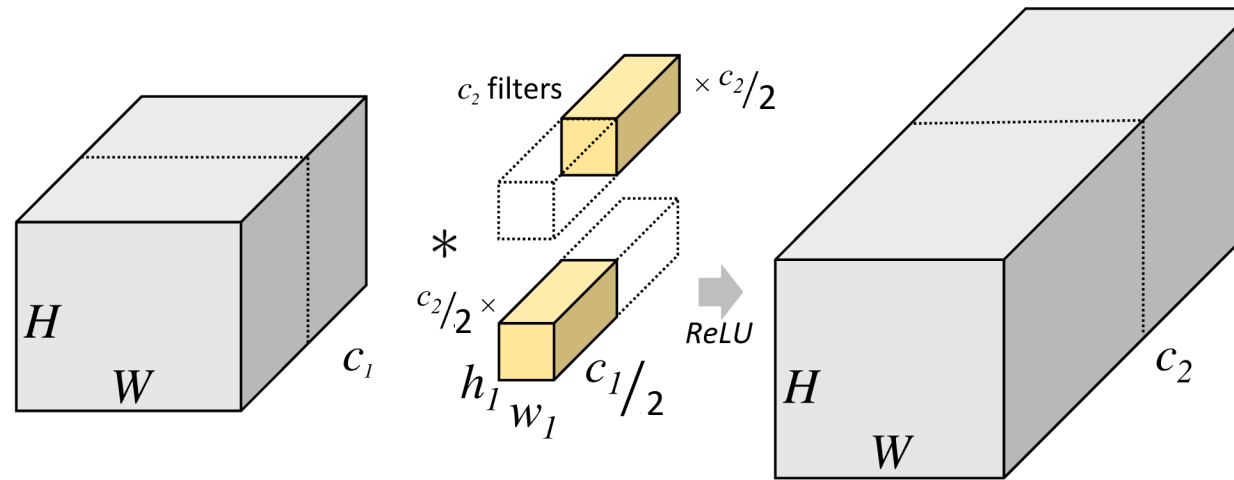
With 2 groups, each filter sees half the input channels.
The weight count is half that of the dense layer (no biases).

Grouped Convolution

Can we maintain the same shaped input and output in convolution with fewer number of parameters?

Yes, grouped convolution!

Now we switch to a grouped (# groups=2) convolution layer applied to the same input (shape $H \times W \times c_1$):



With 2 groups, each filter sees half the input channels.
The weight count is half that of the dense layer (no biases).

$$P_{\text{group}} = G \left(h_1 w_1 \frac{c_1}{G} \frac{c_2}{G} \right) = \frac{h_1 w_1 c_1 c_2}{G}$$

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - **Separable Convolution**
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

Let us look at a 3x3 convolutional kernel:

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times [1 \quad 2 \quad 3]$$

Separable Convolution

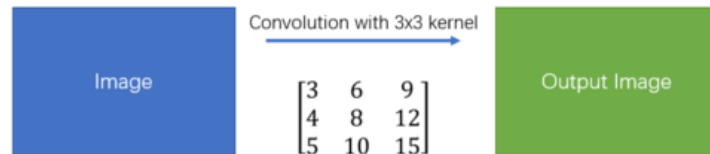
Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

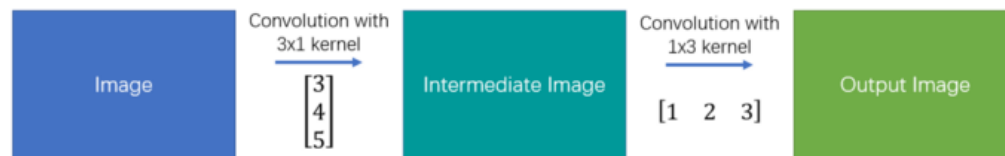
Let us look at a 3x3 convolutional kernel:

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Simple Convolution



Spatial Separable Convolution



Separable Convolution

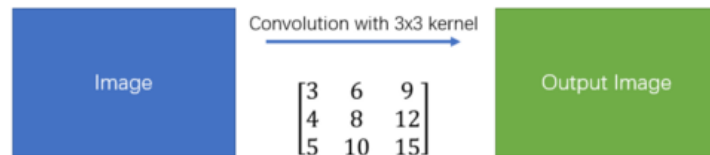
Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

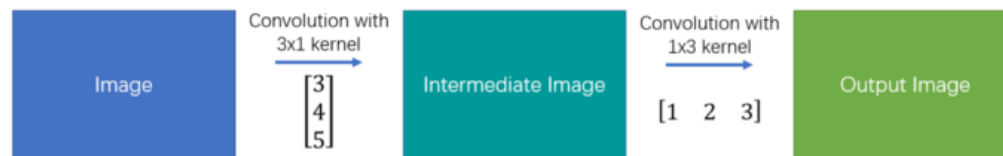
Let us look at a 3x3 convolutional kernel:

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Simple Convolution



Spatial Separable Convolution



Spatially separable kernels have rank one.
They cannot represent full-rank kernels.

Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

In practice, we often use depthwise separable convolution:

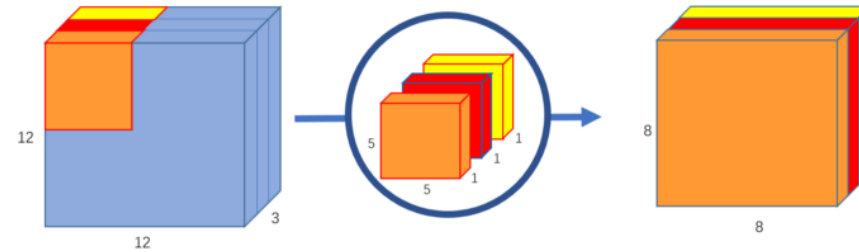
Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

In practice, we often use depthwise separable convolution:

- Depthwise spatial convolution



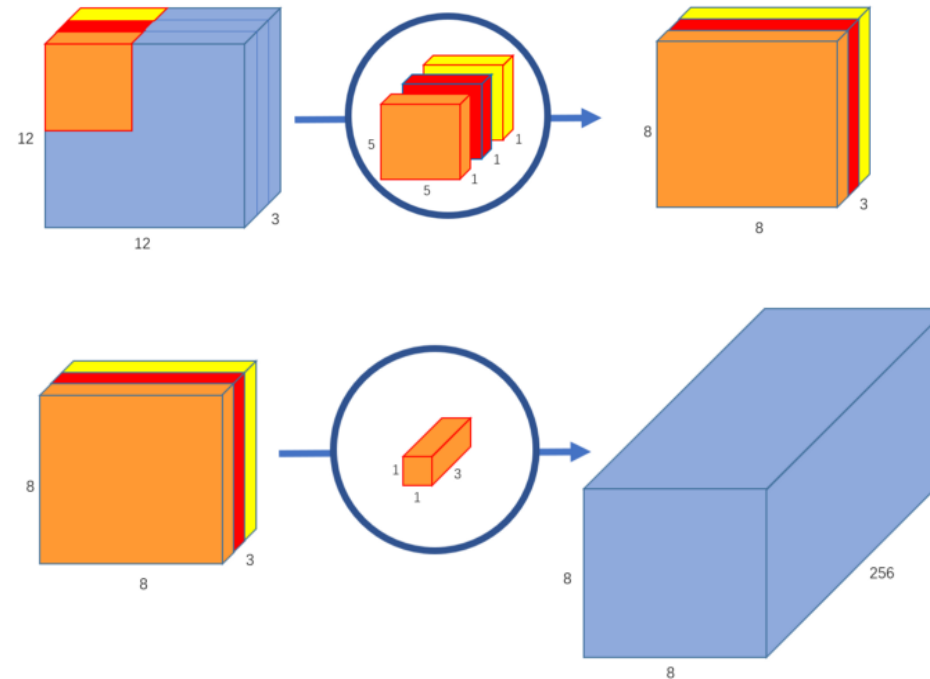
Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

In practice, we often use depthwise separable convolution:

- Depthwise spatial convolution
- Pointwise 1x1 convolution



Separable Convolution

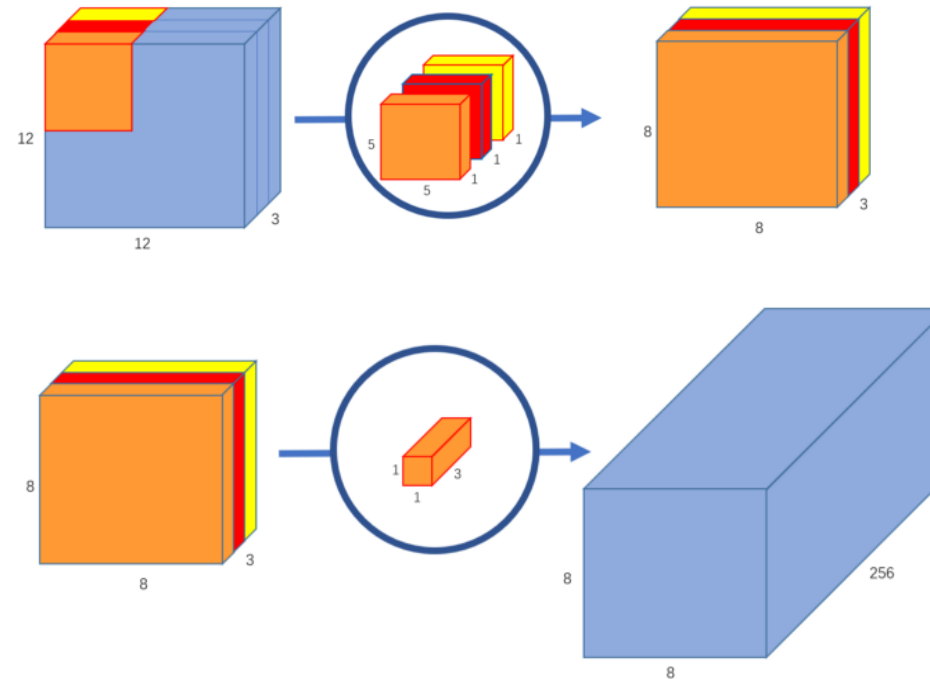
Can we keep the same input/output shapes with fewer convolution parameters?

Yes, grouped convolution & **separable convolution**!

In practice, we often use depthwise separable convolution:

- Depthwise spatial convolution
- Pointwise 1x1 convolution

It is a separable convolution: spatial $\times$ depth (channel)!



Separable Convolution

Can we keep the same input/output shapes with fewer convolution parameters?

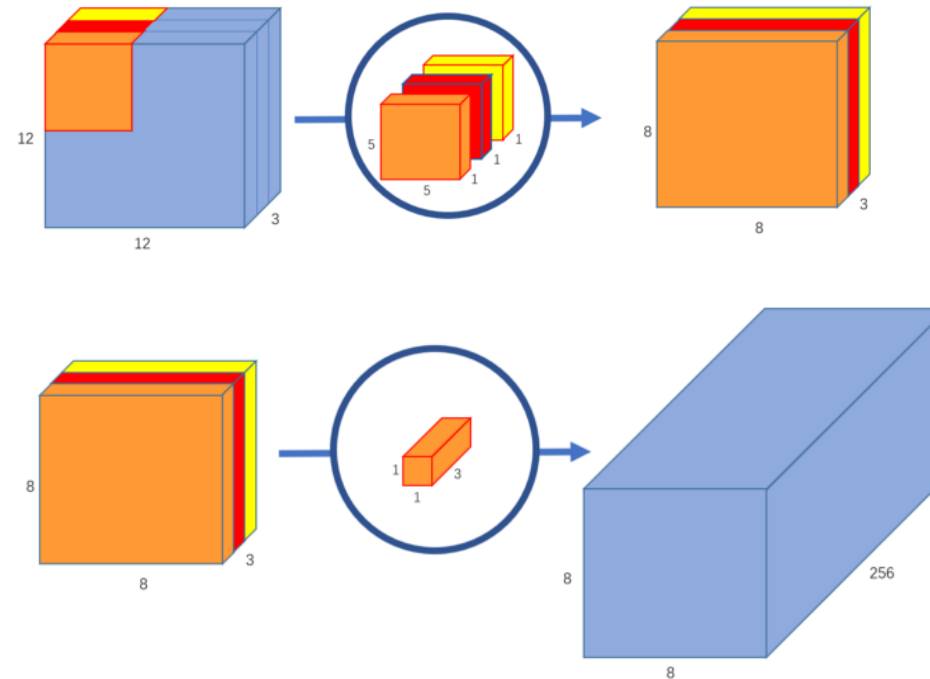
Yes, grouped convolution & **separable convolution**!

In practice, we often use depthwise separable convolution:

- Depthwise spatial convolution
- Pointwise 1x1 convolution

It is a separable convolution: spatial $\times$ depth (channel)!

Spatial filtering and channel mixing use different sets of weights.



Convolution Parameter Counts

Compare a 3x3 kernel with 32 input channels and 64 output channels (no biases).

$$P_{\text{dense}} = h_1 w_1 c_1 c_2, \quad P_{\text{sep}} = h_1 w_1 c_1 + c_1 c_2 \quad \frac{P_{\text{sep}}}{P_{\text{dense}}} = \frac{1}{c_2} + \frac{1}{h_1 w_1}$$

Operator	Output channels	Weights
Dense	64	18,432
Grouped (G = 4)	64	4,608
Depthwise only	32	288
Depthwise + pointwise	64	2,336

Depthwise filtering keeps channels separate. Pointwise convolution mixes them.

Outline

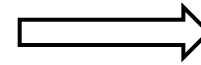
- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- **Pooling**
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

Pooling

A similar idea as convolution except that you replace *weighted sum* operator with some pooling operator (e.g., *max*, *mean*)

2 X 2 Max Pooling with Stride 2

1	0	3	5
3	4	2	2
1	3	3	9
5	7	8	4



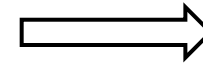
4	5
7	9

Pooling

A similar idea as convolution except that you replace *weighted sum* operator with some pooling operator (e.g., *max*, *mean*)

2 X 2 Max Pooling with Stride 2

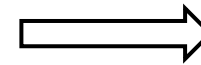
1	0	3	5
3	4	2	2
1	3	3	9
5	7	8	4



4	5
7	9

2 X 2 Mean Pooling with Stride 2

1	0	3	5
3	4	2	2
1	3	3	9
5	7	8	4

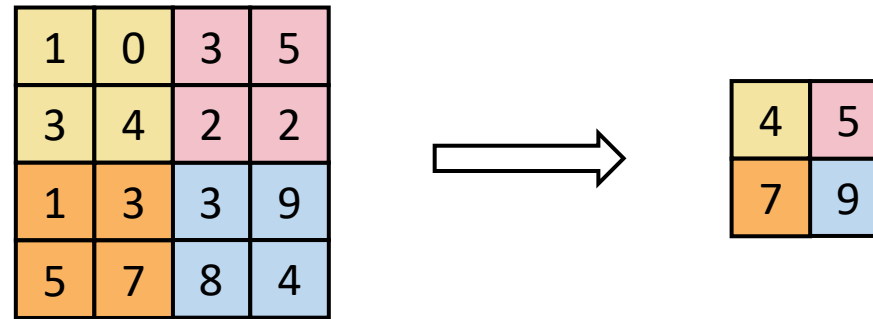


2	3
4	6

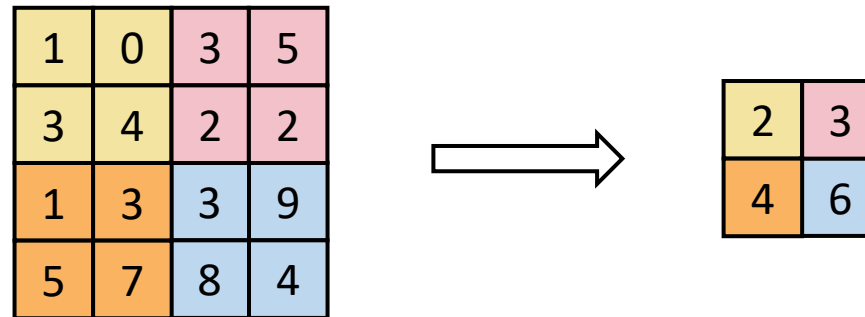
Pooling

A similar idea as convolution except that you replace *weighted sum* operator with some pooling operator (e.g., *max*, *mean*)

2 X 2 Max Pooling with Stride 2



2 X 2 Mean Pooling with Stride 2



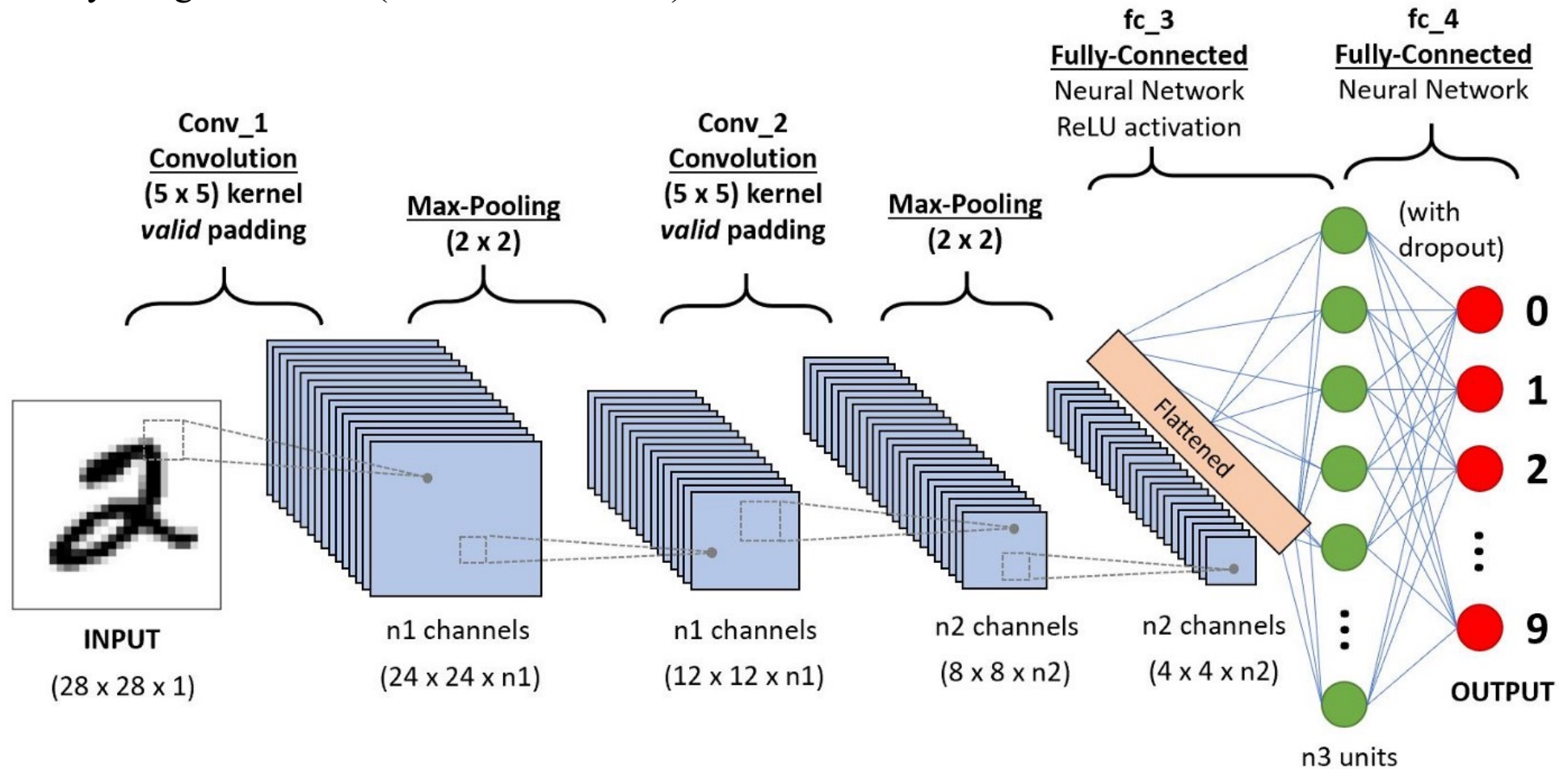
Max/mean pooling is permutation invariant within each fixed window.

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - **LeNet**, U-Net, ResNet/ResNeXt, MobileNet, ConvNeXt

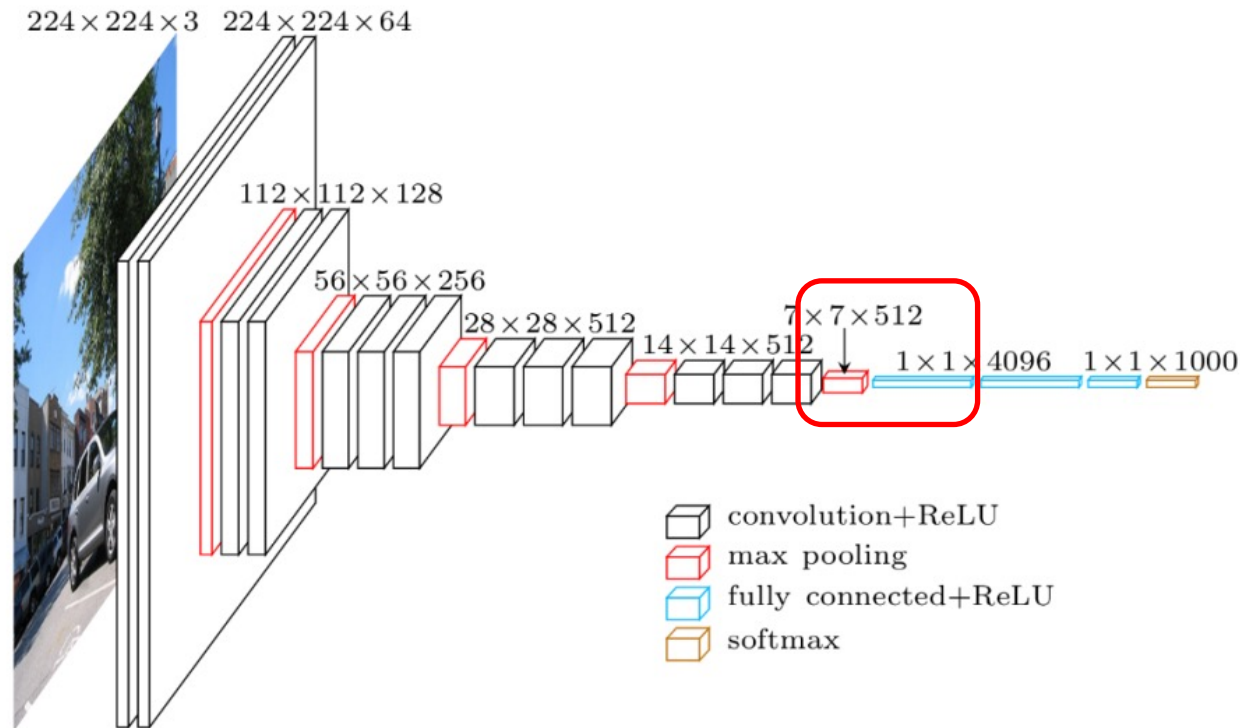
LeNet

A LeNet-style digit classifier (illustrative variant):



LeNet: Pooling and Shift Robustness

Local pooling can reduce sensitivity to small shifts, but does not guarantee shift invariance [15].



Pooling illustration in a deeper CNN (not the LeNet architecture) [6].

LeNet: Pooling and Shift Robustness

Does local max pooling guarantee shift invariance [15]?

Max pooling: window size 2, stride 2 (circular right shifts)

Original



Pooled outputs



LeNet: Pooling and Shift Robustness

Does local max pooling guarantee shift invariance [15]?

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Pooled outputs

1	3
---	---

LeNet: Pooling and Shift Robustness

Does local max pooling guarantee shift invariance [15]?

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Pooled outputs

1	3
3	2

LeNet: Pooling and Shift Robustness

Does local max pooling guarantee shift invariance [15]?

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Pooled outputs

1	3
3	2

The outputs differ: local stride-2 pooling is not shift invariant.

LeNet: Pooling and Shift Robustness

Equivariance: does shifting commute with pooling [15]?

S_1 shifts an array one position to the right (circularly).

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Shift the original pooled output right by 1:

Pooled outputs

1	3
3	2
3	1

$$P(S_1 \mathbf{x}) \stackrel{?}{=} S_1 P(\mathbf{x})$$

LeNet: Pooling and Shift Robustness

Equivariance: does shifting commute with pooling [15]?

S_1 shifts an array one position to the right (circularly).

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Shift the original pooled output right by 1:

Pooled outputs

1	3
3	2
3	1

$$P(S_1 \mathbf{x}) = [3, 2] \neq [3, 1] = S_1 P(\mathbf{x})$$

Local max pooling with stride 2 is not equivariant to arbitrary shifts [15].

With circular boundaries, shifting the input by 2 shifts the pooled output by 1.

LeNet: Pooling and Shift Robustness

Does local max pooling guarantee shift invariance [15]?

Max pooling: window size 2, stride 2 (circular right shifts)

Original

0	1	2	3
3	0	1	2

Shifted right by 1
(circular)

Pooled outputs

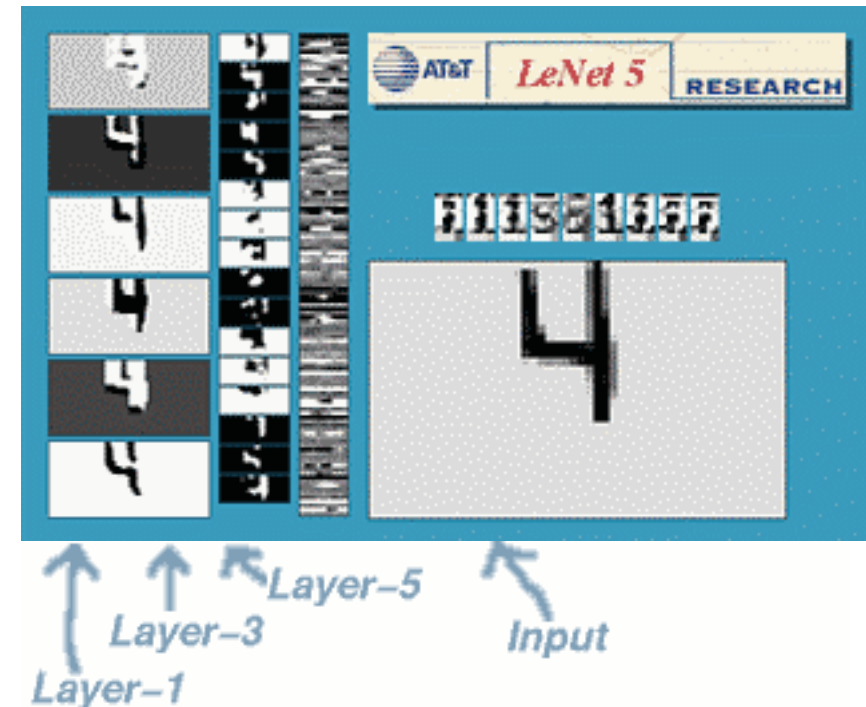
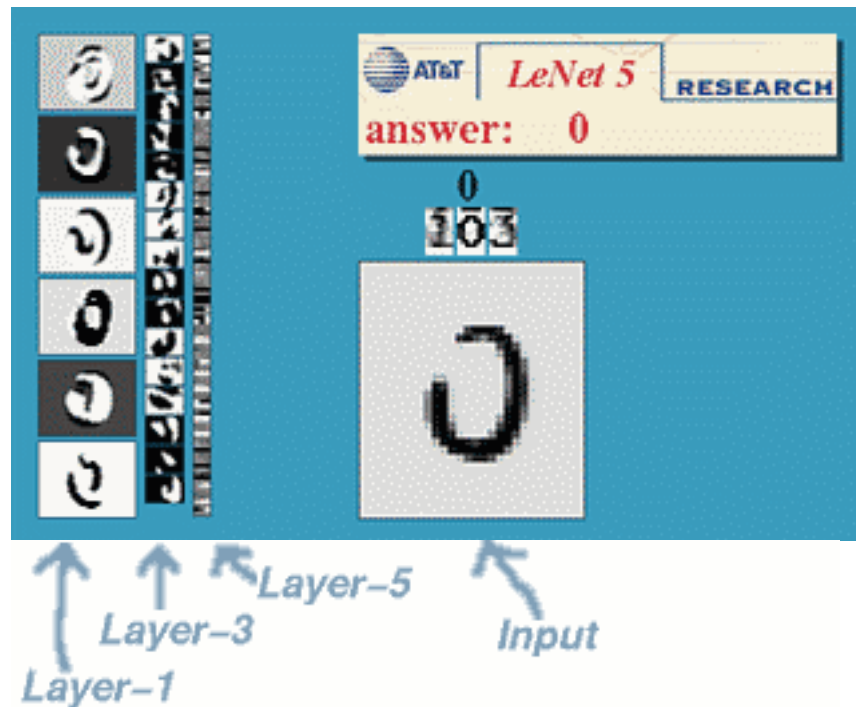
1	3
3	2

The outputs differ: local stride-2 pooling is not shift invariant.

Global max is 3 for both. Global pooling ignores permutations of the whole feature map, including shifts that only permute its spatial positions.

LeNet: Shift Robustness Demo

Yann LeCun's LeNet Demo:



LeNet: Other Transformations

What about other transformations, e.g., scaling, 2D/3D rotations?

Vanilla CNNs do not have such properties. One can add data augmentation to make the model approximately have them.

One can also design CNN architectures, e.g., spherical CNNs (rotation equivariant), that are guaranteed to have such properties [9].



Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, **U-Net**, ResNet/ResNeXt, MobileNet, ConvNeXt

Upsampling and Checkerboard Artifacts

Transposed convolution can create uneven overlap and checkerboard artifacts [16].

Transposed convolution



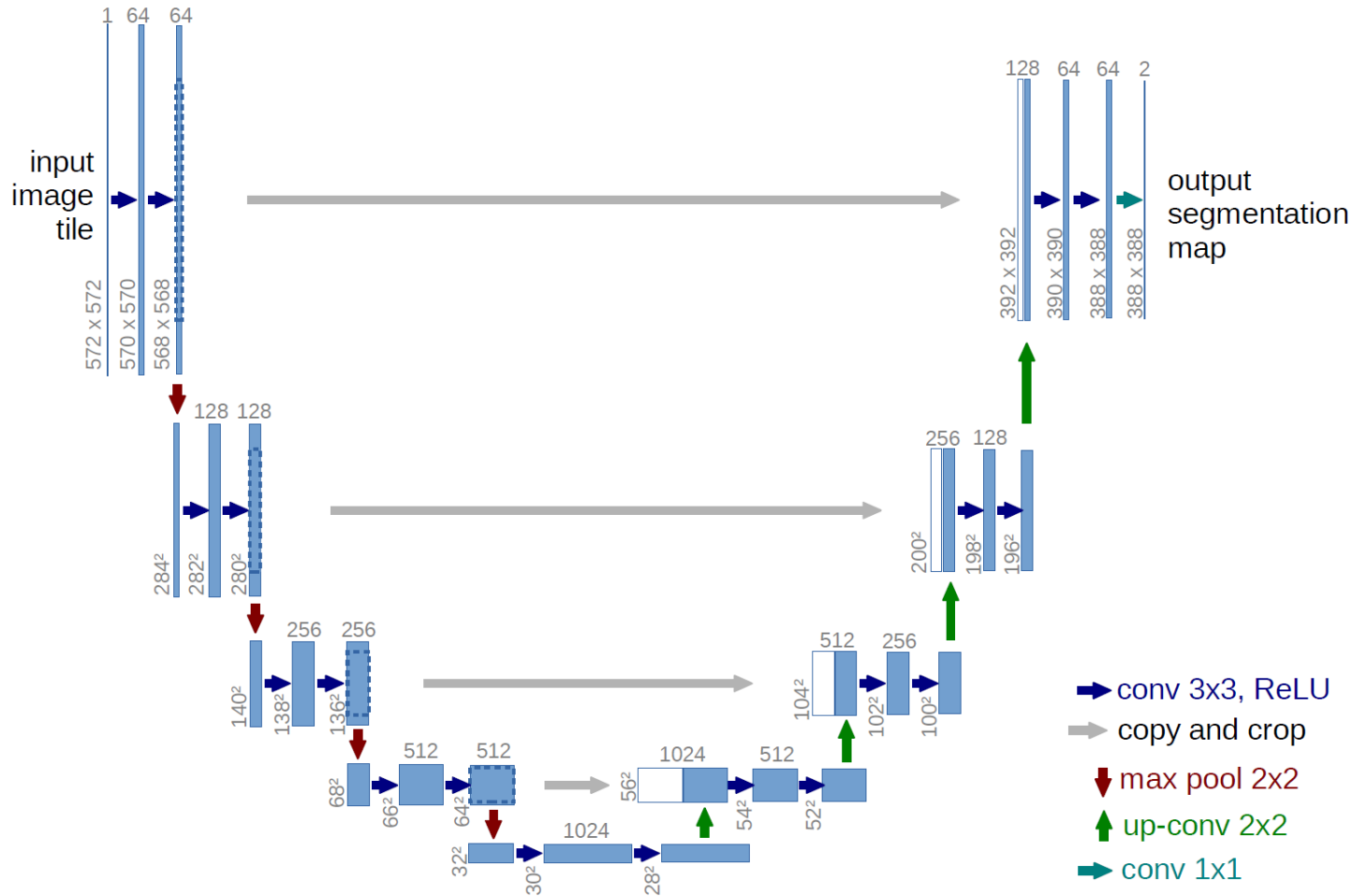
Resize, then convolution



Uneven overlap occurs when kernel size is not divisible by stride.
Even overlap alone does not guarantee an artifact-free learned filter.

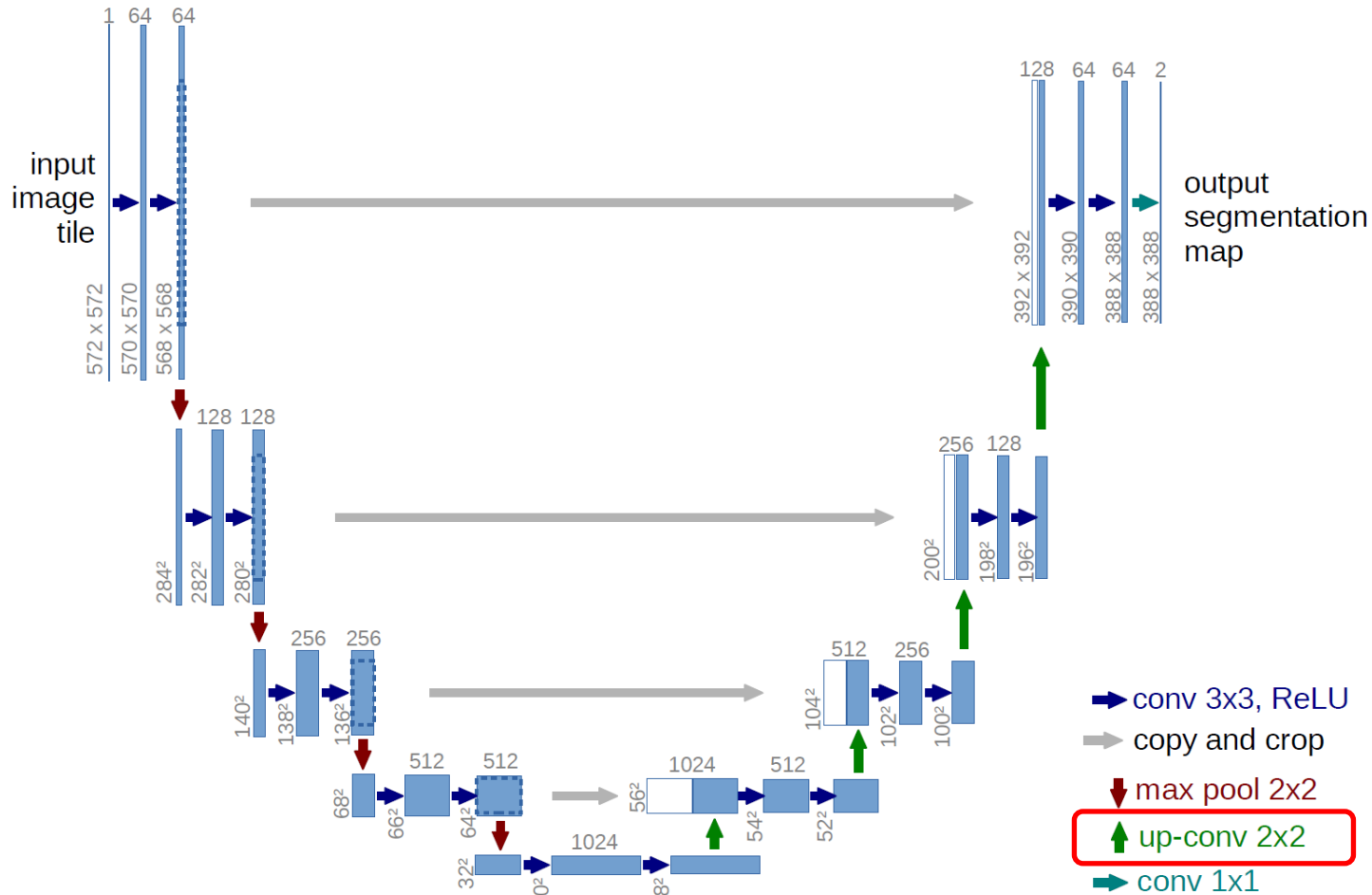
U-Net

U-Net [10] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.



U-Net

U-Net [10] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.



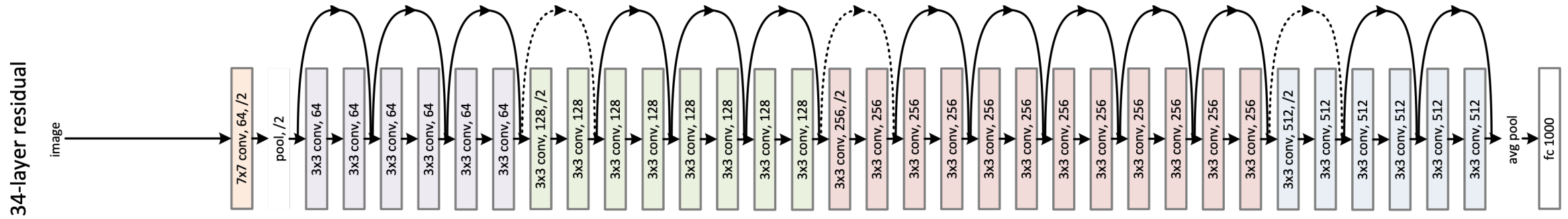
Transposed
Convolution

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, **ResNet/ResNeXt**, MobileNet, ConvNeXt

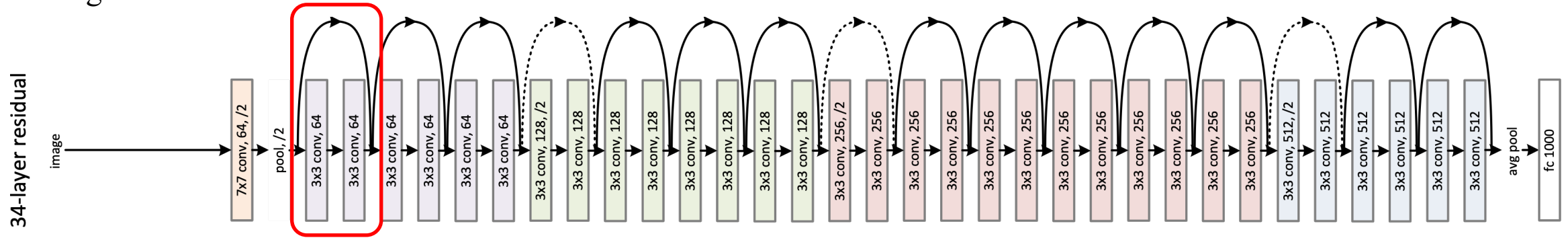
ResNet

ResNet [11] learns residual functions for image classification. Its backbone also supports detection and segmentation.



ResNet

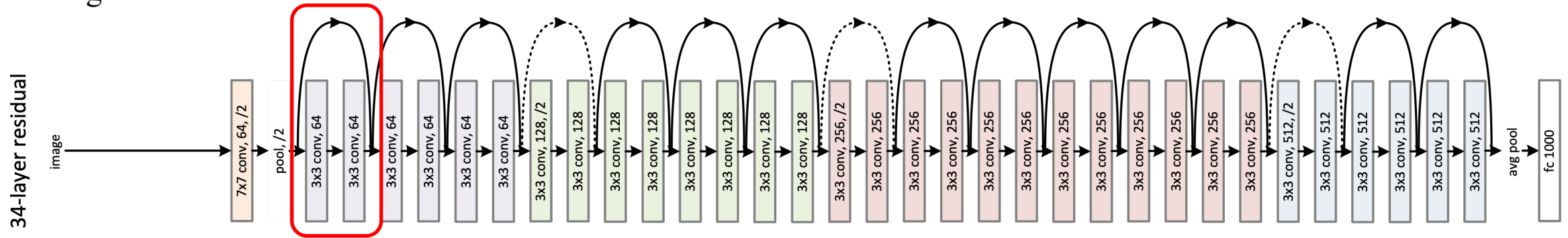
ResNet [11] learns residual functions for image classification. Its backbone also supports detection and segmentation.



Add the input to the learned residual.

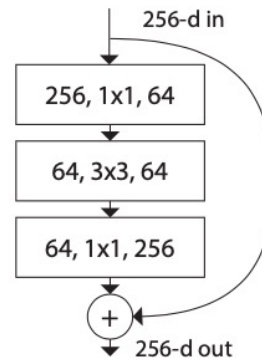
ResNet

ResNet [11] learns residual functions for image classification. Its backbone also supports detection and segmentation.



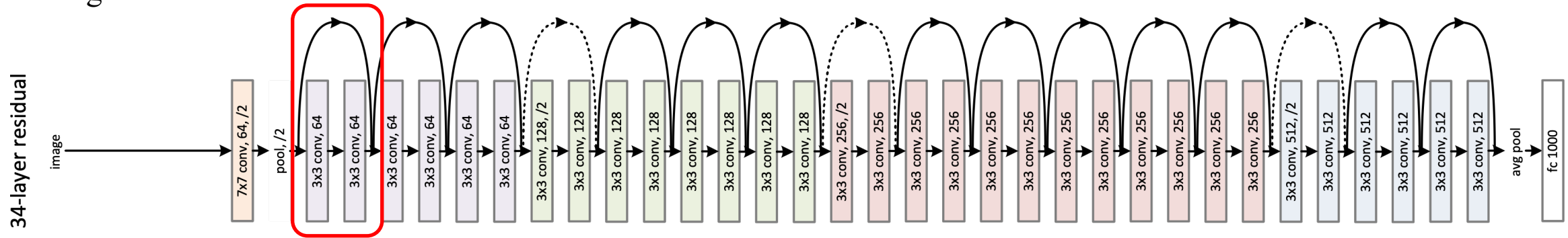
Add the input to the learned residual.

Build deeper ones (e.g., ResNet-50, ResNet-101) by replacing it with the bottleneck structure!



ResNet

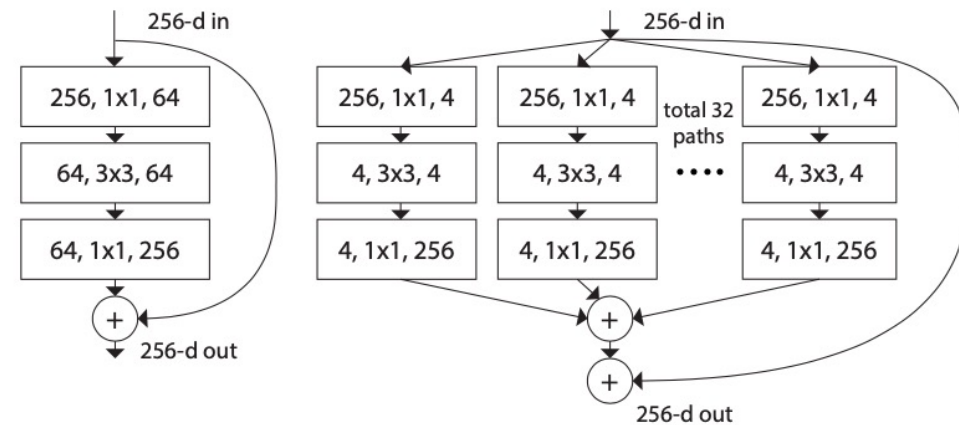
ResNet [11] learns residual functions for image classification. Its backbone also supports detection and segmentation.



Add the input to the learned residual.

Build deeper ones (e.g., ResNet-50, ResNet-101) by replacing it with the bottleneck structure!

ResNeXt [12] replaces it with aggregated transformations (similar to grouped convolution but with shared input)!

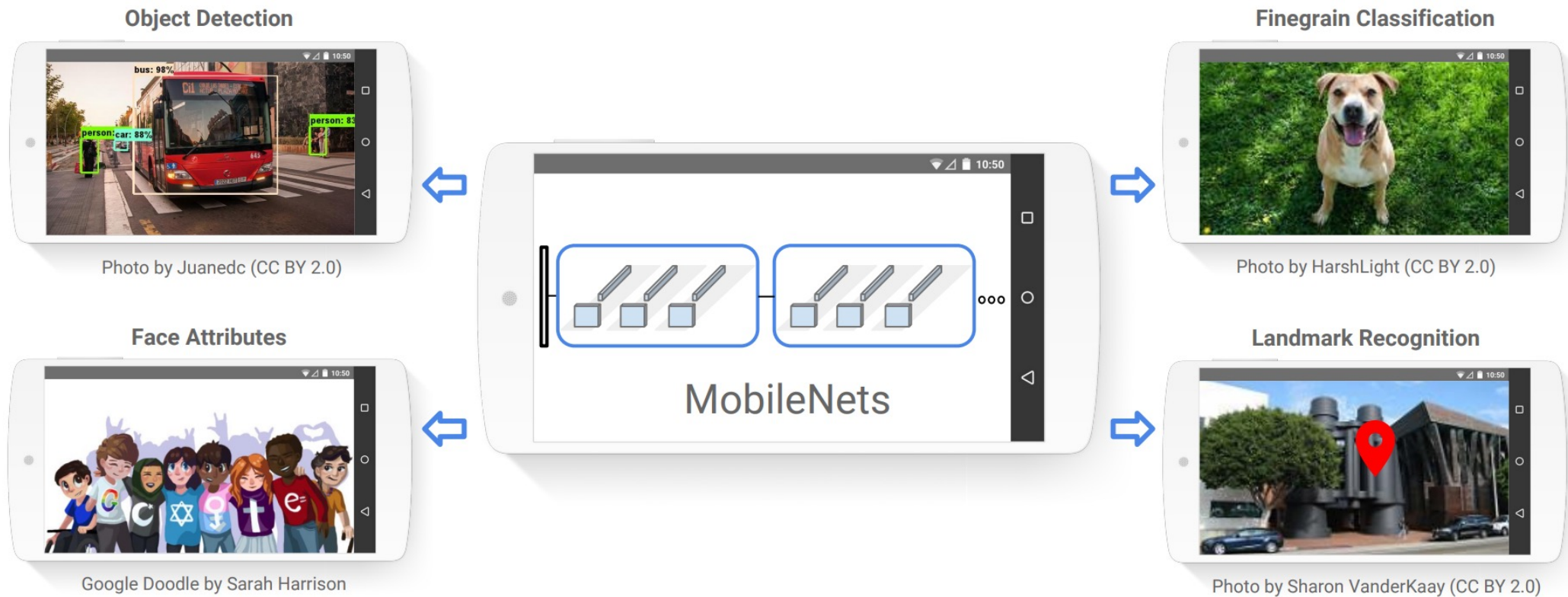


Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, **MobileNet**, ConvNeXt

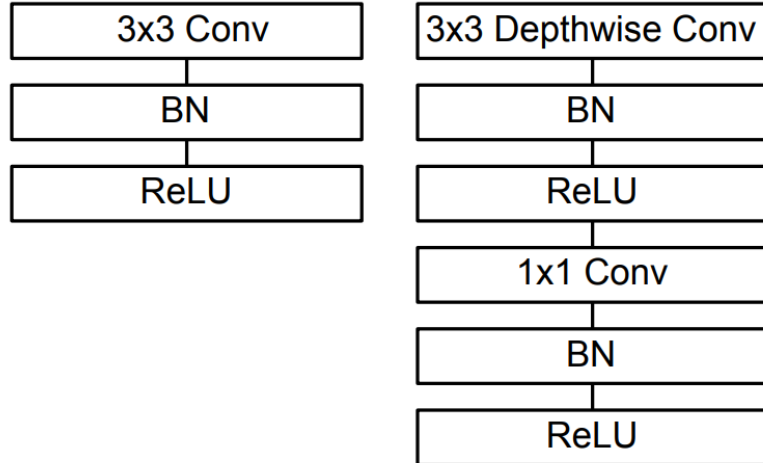
MobileNet

MobileNet [13] is designed to be used in mobile applications, achieving good performances with fewer computations.



MobileNet

MobileNet [13] is designed to be used in mobile applications, achieving good performances with fewer computations.



Replace the vanilla conv layer with
depthwise separable convolutional
layer

Outline

- Invariance & Equivariance
- Convolution
 - 1D Convolution
 - Matrix Multiplication Views
 - Translation Equivariance
 - 2D Convolution
- Convolution Variants
 - Transposed Convolution
 - Dilated Convolution
 - Grouped Convolution
 - Separable Convolution
- Pooling
- Example Architectures
 - LeNet, U-Net, ResNet/ResNeXt, MobileNet, **ConvNeXt**

ConvNeXt

ConvNeXt [17] combines familiar convolution operators with modern block design.

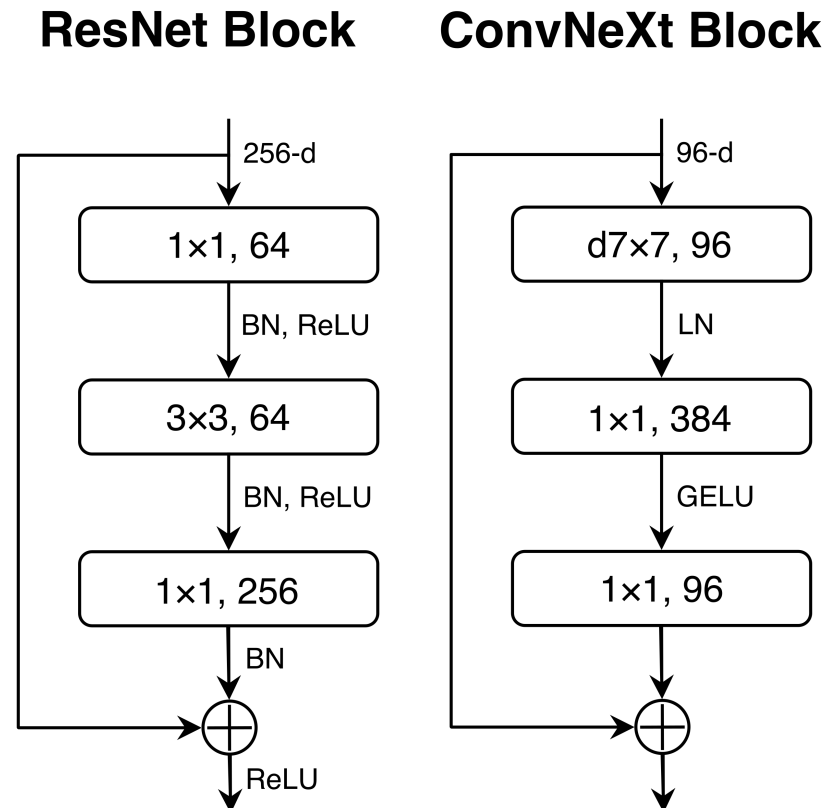
7x7 depthwise convolution

LayerNorm

Pointwise expansion and contraction

GELU and a residual connection

MobileNetV4 [18] also targets measured latency on different mobile hardware.



References

- [1] https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md
- [2] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems. 2012;25.
- [3] <https://blog.yani.ai/filter-group-tutorial/>
- [4] <https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>
- [5] <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [6] <https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529>
- [7] <http://yann.lecun.com/exdb/lenet/translation.html>
- [8] <http://yann.lecun.com/exdb/lenet/scale.html>
- [9] <https://pure.uva.nl/ws/files/60770359/Thesis.pdf>
- [10] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015.
- [11] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).
- [12] Xie S, Girshick R, Dollár P, Tu Z, He K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 1492-1500).
- [13] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. 2017 Apr 17.

References

- [14] PyTorch. <https://docs.pytorch.org/docs/2.14/generated/torch.nn.ConvTranspose2d.html>
- [15] Zhang R. Making Convolutional Networks Shift-Invariant Again. In International Conference on Machine Learning 2019 (pp. 7324-7334).
- [16] Odena A, Dumoulin V, Olah C. Deconvolution and Checkerboard Artifacts. Distill. 2016.
- [17] Liu Z, Mao H, Wu CY, Feichtenhofer C, Darrell T, Xie S. A ConvNet for the 2020s. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2022 (pp. 11976-11986).
- [18] Qin D, Leichner C, Delakis M, Fornoni M, Luo S, Yang F, Wang W, Banbury C, Ye C, Akin B, Aggarwal V, Zhu T, Moro D, Howard A. MobileNetV4: Universal Models for the Mobile Ecosystem. In European Conference on Computer Vision 2024.

Questions?