CPEN 455: Deep Learning

Lecture 5: Convolutional Neural Networks II

Renjie Liao

University of British Columbia

Winter, Term 2, 2024

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

2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

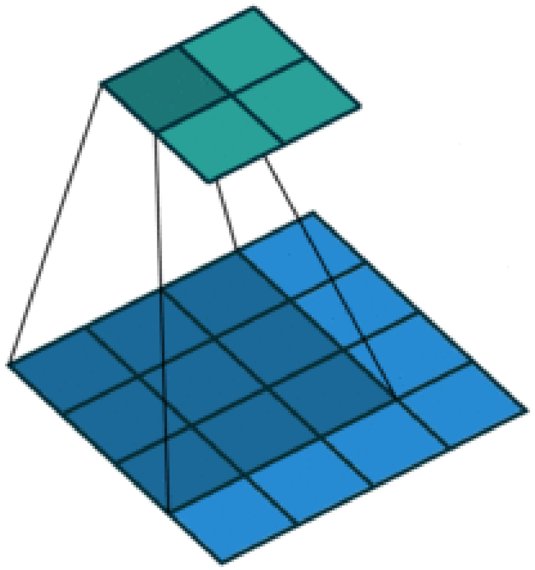
2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Suppose we have a 2D convolution (3x3 kernel):

Shapes: 4x4 -> 2x2



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

Image Credit: [1]

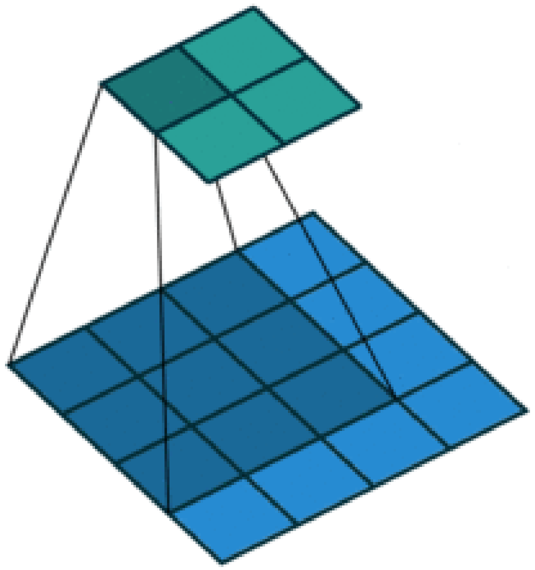
2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

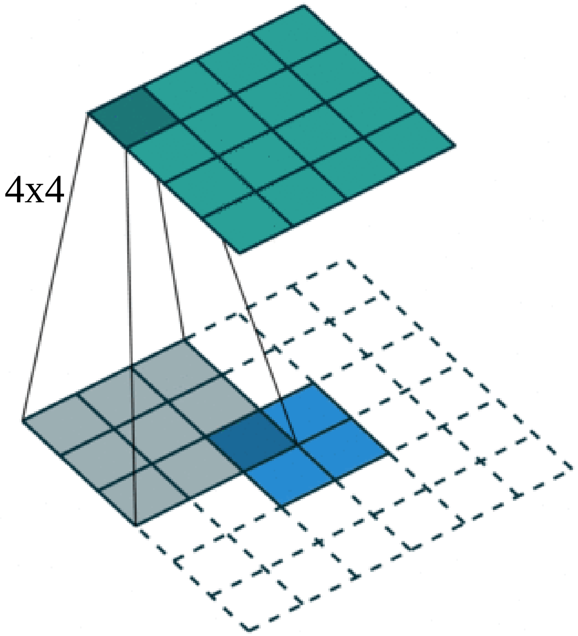
Suppose we have a 2D convolution (3x3 kernel):

Shapes: 4x4 -> 2x2



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

Shapes: 2x2 -> 4x4



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

Image Credit: [1]

2D Transposed Convolution

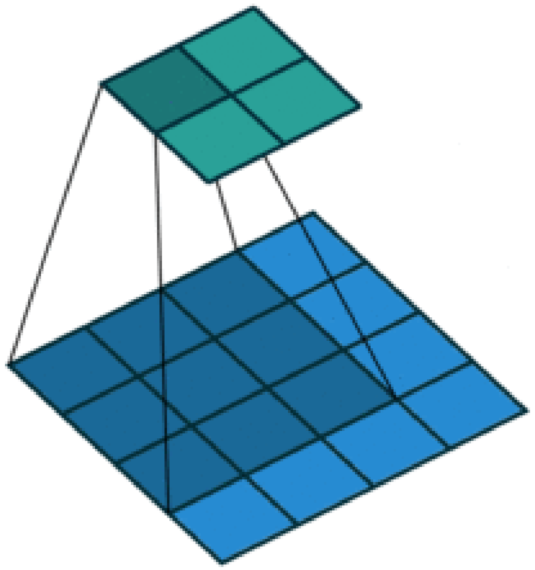
We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Suppose we have a 2D convolution (3x3 kernel):

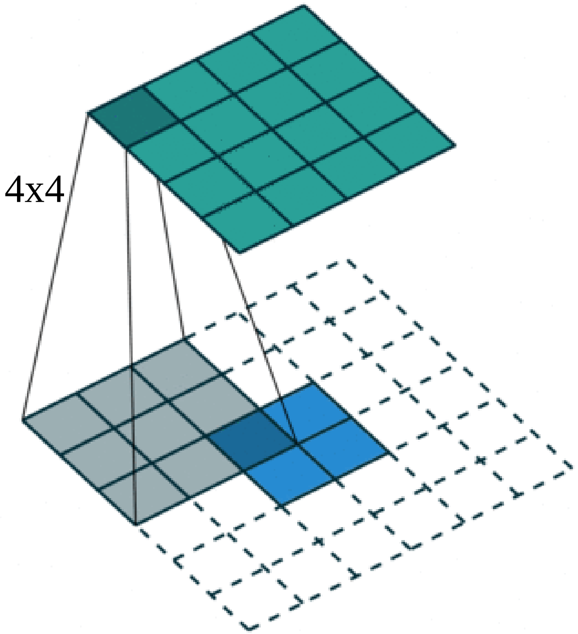
- Convolution and its transposed version are mutually inverse only w.r.t. shapes of input and output, but not w.r.t. values of input and output!

Shapes: 4x4 -> 2x2



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

Shapes: 2x2 -> 4x4



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

Image Credit: [1]

2D Transposed Convolution

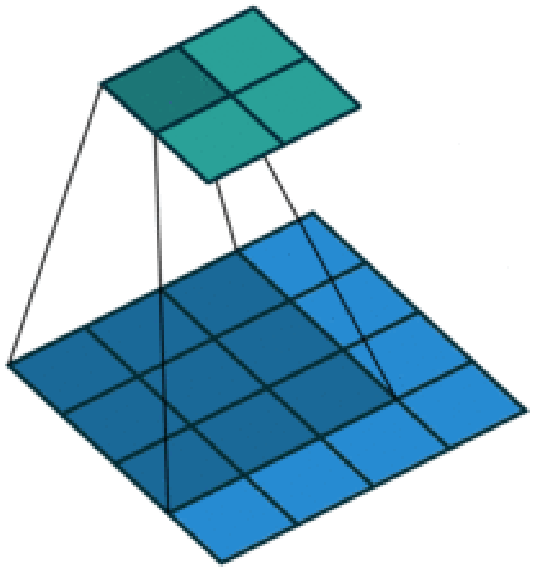
We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Suppose we have a 2D convolution (3x3 kernel):

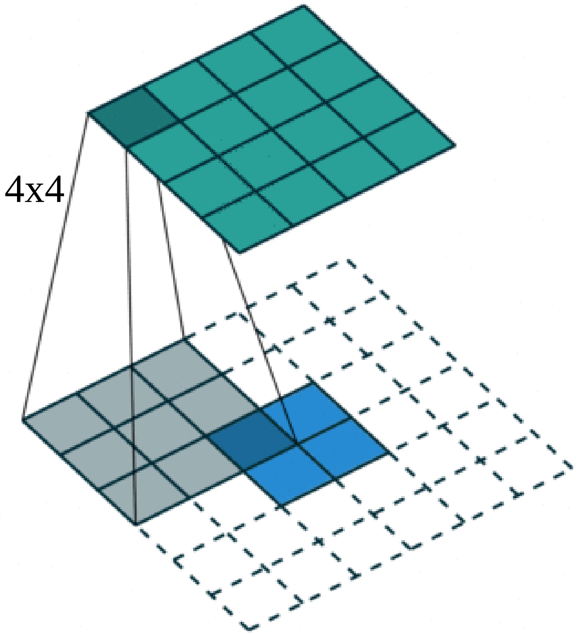
- Convolution and its transposed version are mutually inverse only w.r.t. shapes of input and output, but not w.r.t. values of input and output!
- Convolution and deconvolution are mutually inverse w.r.t. values of input and output!

Shapes: 4x4 -> 2x2



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

Shapes: 2x2 -> 4x4



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

Image Credit: [1]

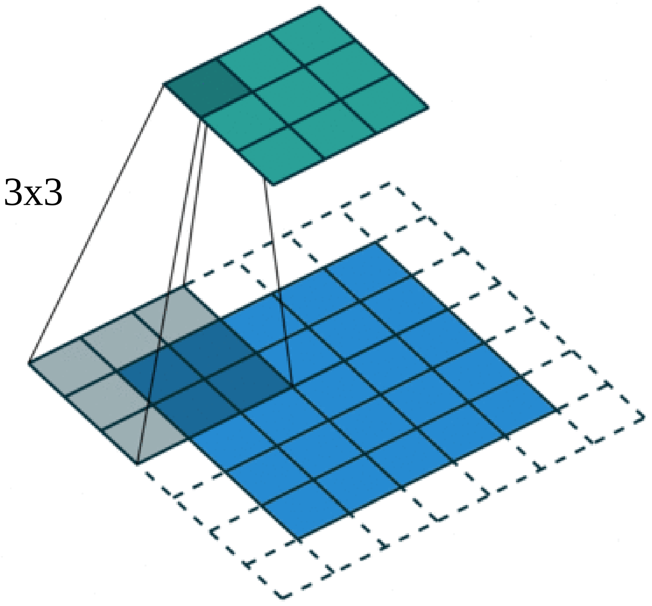
2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

Shapes: 5x5 -> 3x3



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

Image Credit: [1]

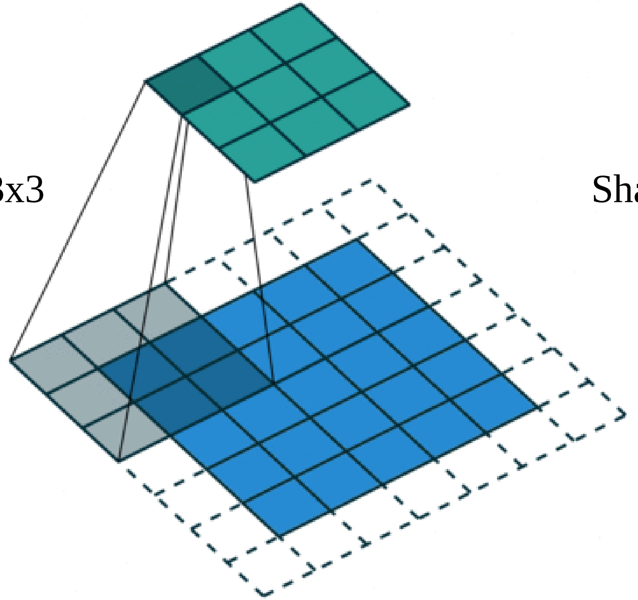
2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

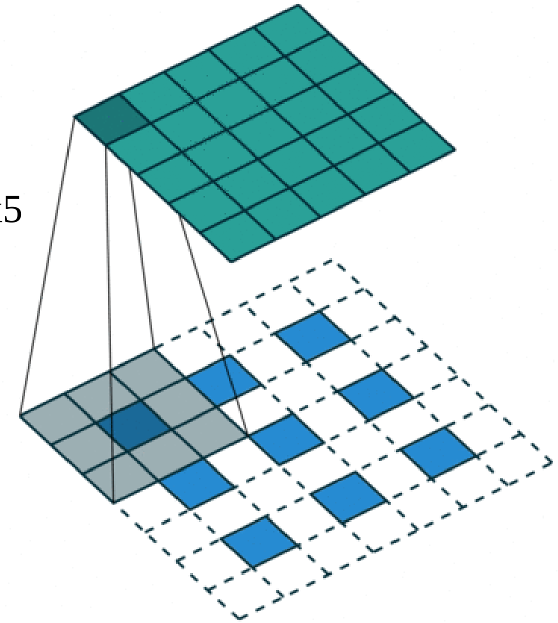
Try another example of 2D convolution (3x3 kernel):

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

Image Credit: [1]

2D Transposed Convolution

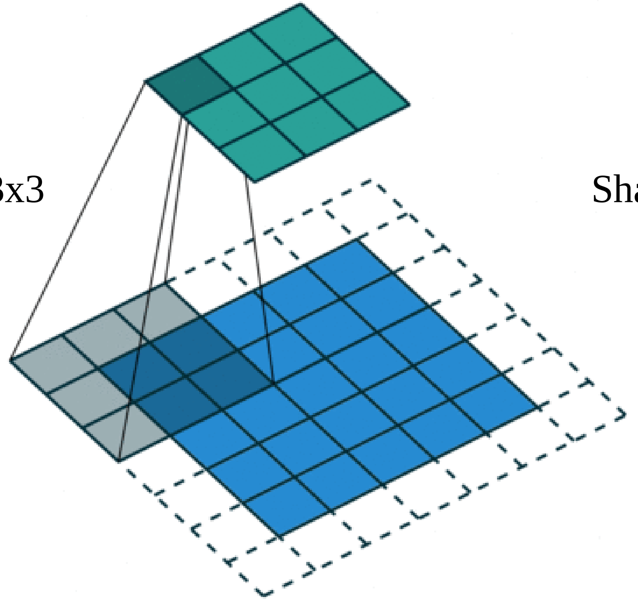
We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

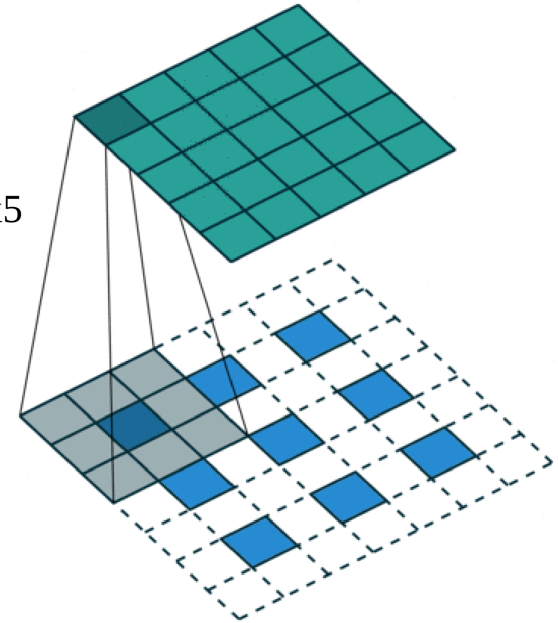
Transposed convolution is also known as fractionally strided convolution!

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

Image Credit: [1]

2D Transposed Convolution

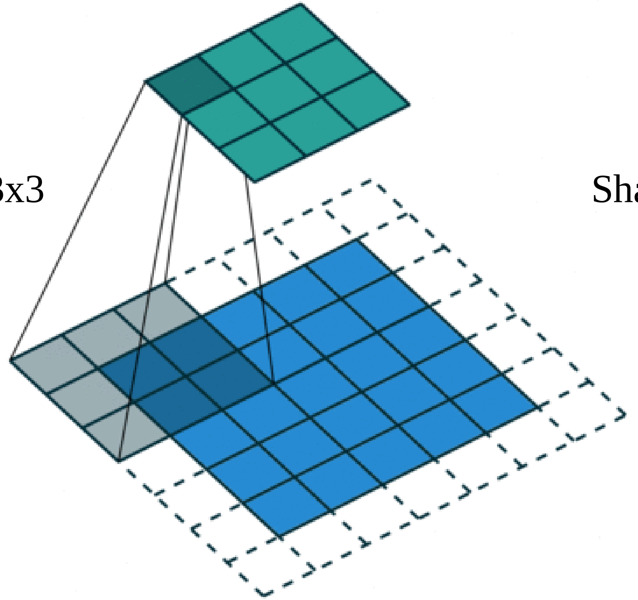
We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

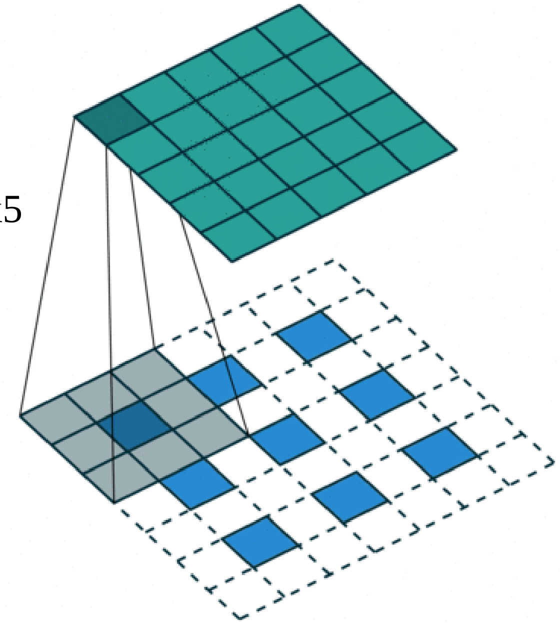
In practice, we do not pad zeros in between and then perform convolution due to its high computational cost.

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

Image Credit: [1]

2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

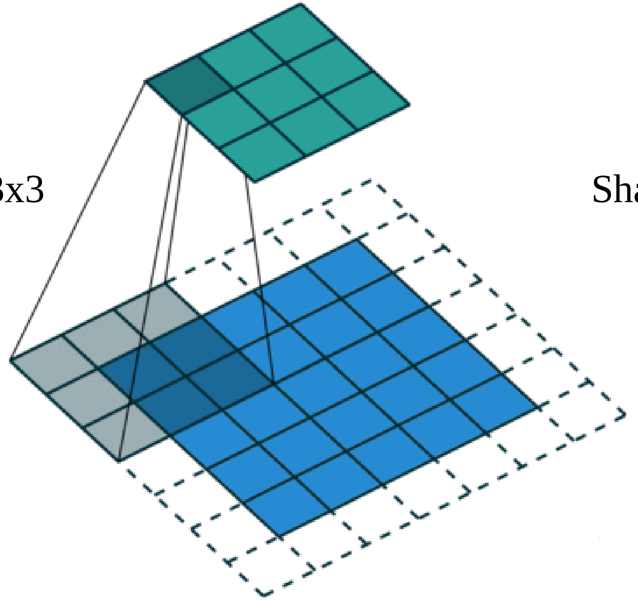
In practice, we do not pad zeros in between and then perform convolution due to its high computational cost. Instead, we leverage the gradient of convolution:

$$\mathbf{y} = W\mathbf{x} \qquad \text{Filter -> Matrix}$$

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

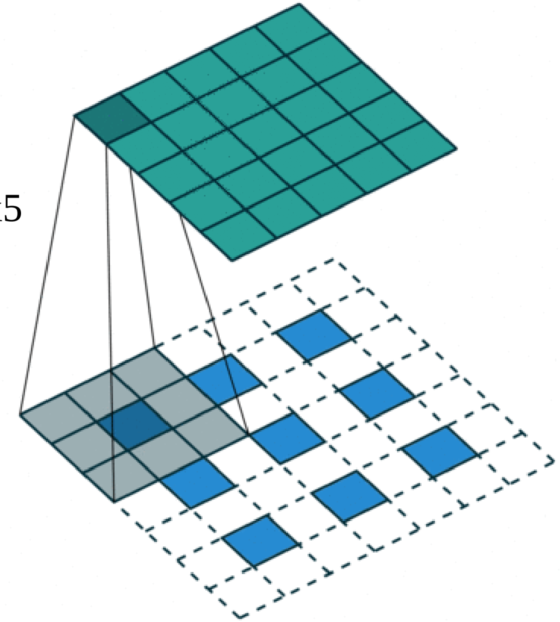
$$\hat{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^{\top} y$$

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

Image Credit: [1]

2D Transposed Convolution

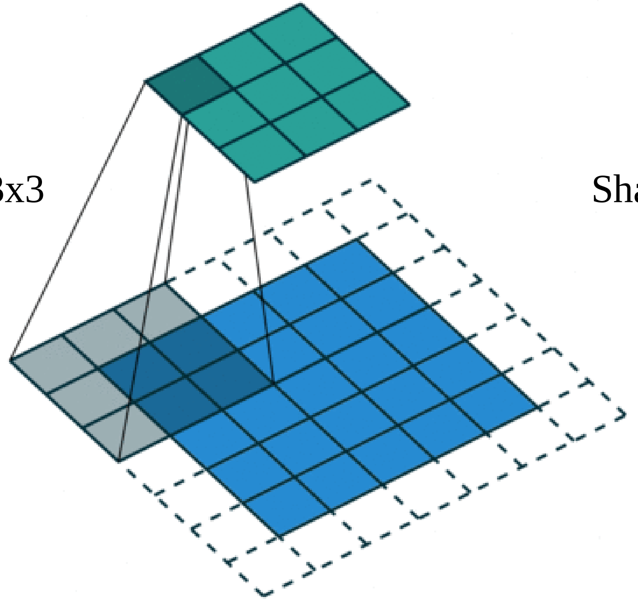
We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

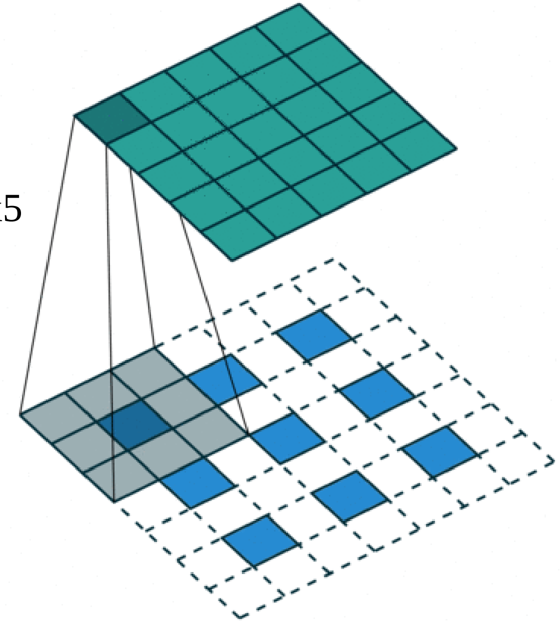
In practice, we do not pad zeros in between and then perform convolution due to its high computational cost. Instead, we leverage the gradient of convolution:

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

$$\mathbf{y} = W\mathbf{x}$$ Filter -> Matrix

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

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

This is why we need to specify the stride and padding of the corresponding convolution, e.g., in PyTorch.

For transposed convolution, stride is always 1 and we sometimes need (output) padding!

Image Credit: [1]

2D Transposed Convolution

We know convolution can reduce the input size, e.g., with stride > 1. Can any convolution operator enlarge the input size?

Yes, transposed convolution!

Try another example of 2D convolution (3x3 kernel):

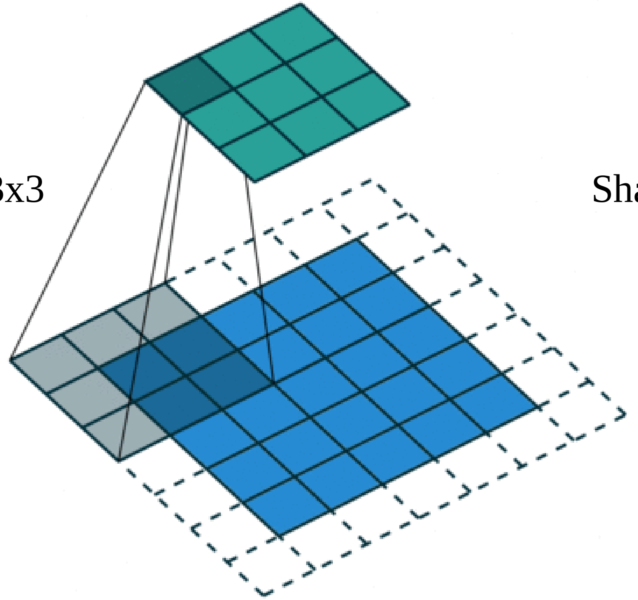
In practice, we do not pad zeros in between and then perform convolution due to its high computational cost. Instead, we leverage the gradient of convolution:

$$\mathbf{y} = W\mathbf{x}$$ Filter -> Matrix

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

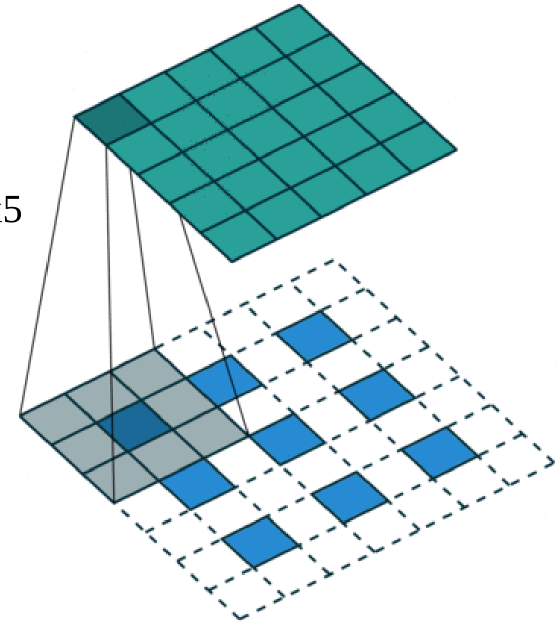
$$\hat{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}}^{\top} y$$

Shapes: 5x5 -> 3x3



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

Shapes: 3x3 -> 5x5



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

This is why we need to specify the stride and padding of the corresponding convolution, e.g., in PyTorch.

For transposed convolution, stride is always 1 and we sometimes need (output) padding!

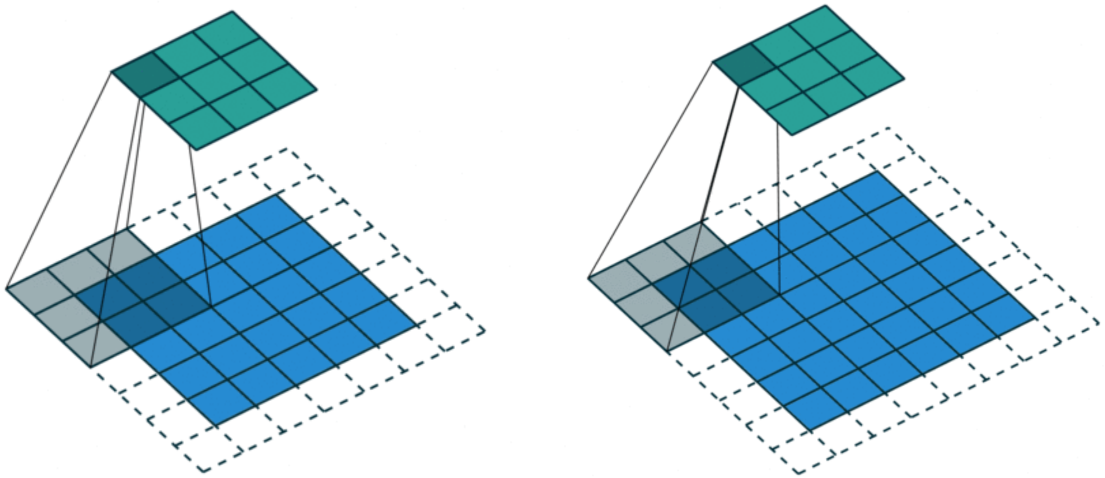
Image Credit: [1]

2D Transposed Convolution

The gradients of the following two convolutions have the same shape in im2patch (data-> toeplitz matrix) implementation.

Shapes: 5x5 -> 3x3

Shapes: 6x6 -> 3x3



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

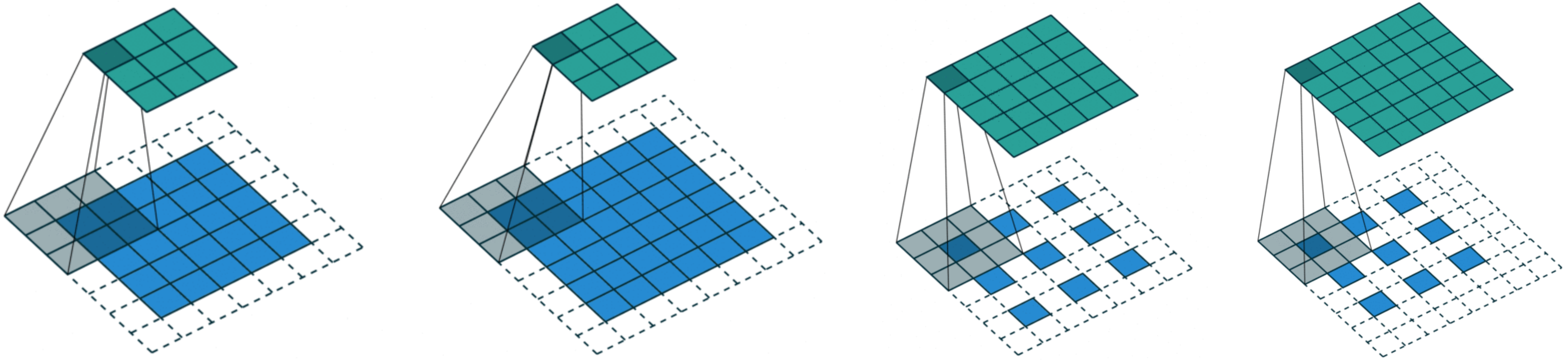
Image Credit: [1]

2D Transposed Convolution

The gradients of the following two convolutions have the same shape in im2patch (data-> toeplitz matrix) implementation.

To distinguish them and output correct shapes in their transposed convolutions, we add output padding on one side in the 2nd case.

Shapes: 5x5 -> 3x3 | Shapes: 6x6 -> 3x3 | Shapes: 3x3 -> 5x5 | Shapes: 3x3 -> 6x6



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

2D Transposed Convolution

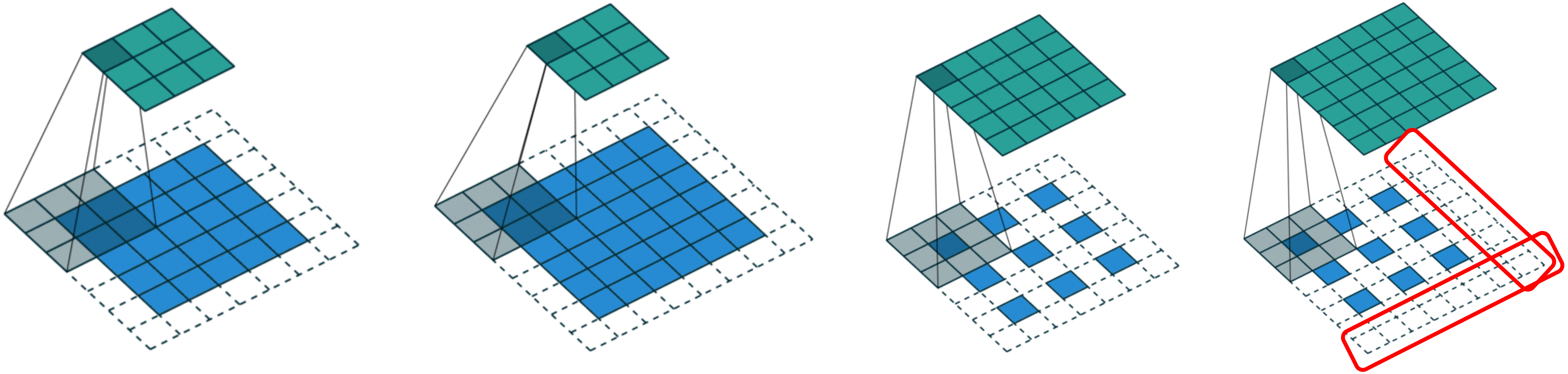
Image Credit: [1]

2D Transposed Convolution

The gradients of the following two convolutions have the same shape in im2patch (data-> toeplitz matrix) implementation.

To distinguish them and output correct shapes in their transposed convolutions, we add output padding on one side in the 2nd case.

Shapes: 5x5 -> 3x3 | Shapes: 6x6 -> 3x3 | Shapes: 3x3 -> 5x5 | Shapes: 3x3 -> 6x6



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

2D Transposed Convolution

output padding=0 | output padding=1

Image Credit: [1]

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 of convolution, not the stride of transposed convolution!

Image Credit: [1]

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.

padding of convolution, not the padding of transposed convolution!

Image Credit: [1]

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.

padding of transposed convolution!

Image Credit: [1]

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

2D Dilated Convolution

We know the kernel size decides what elements are used in convolution at one location. Can we enlarge the kernel size without increasing the number of parameters?

2D Dilated Convolution

We know the kernel size decides what elements are used in convolution at one location. Can we enlarge the kernel size without increasing the number of parameters?

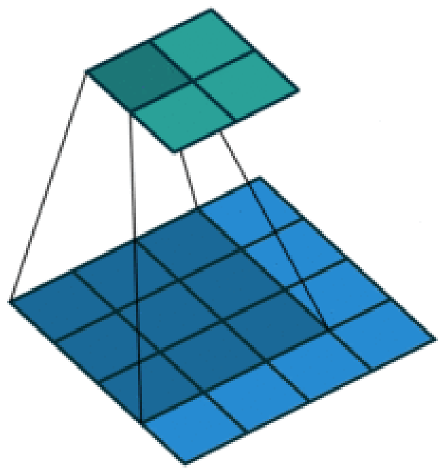
Yes, dilated (atrous) convolution!

2D Dilated Convolution

We know the kernel size decides what elements are used in convolution at one location. Can we enlarge the kernel size without increasing the number of parameters?

Yes, dilated (atrous) convolution!

Suppose we have a 2D convolution:



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

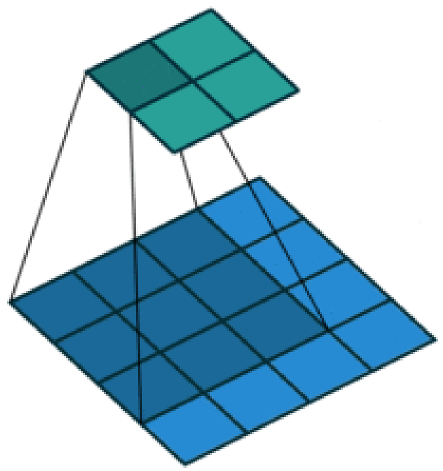
Image Credit: [1]

2D Dilated Convolution

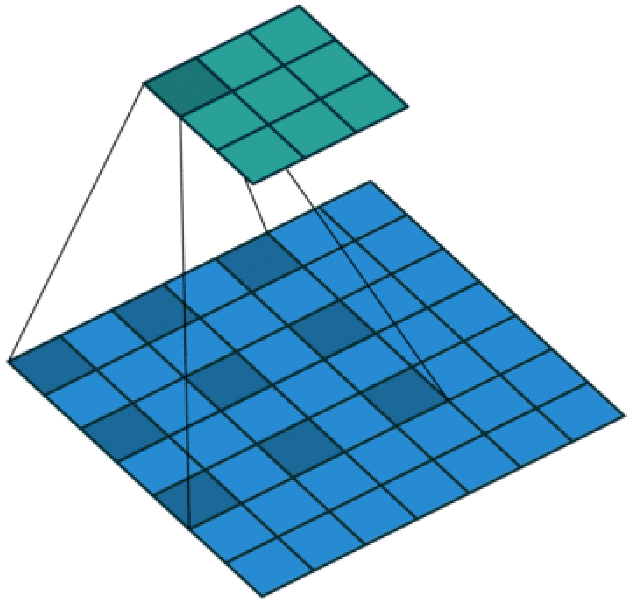
We know the kernel size decides what elements are used in convolution at one location. Can we enlarge the kernel size without increasing the number of parameters?

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)

Image Credit: [1]

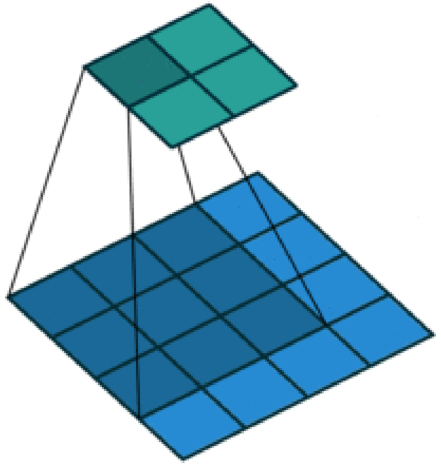
2D Dilated Convolution

We know the kernel size decides what elements are used in convolution at one location. Can we enlarge the kernel size without increasing the number of parameters?

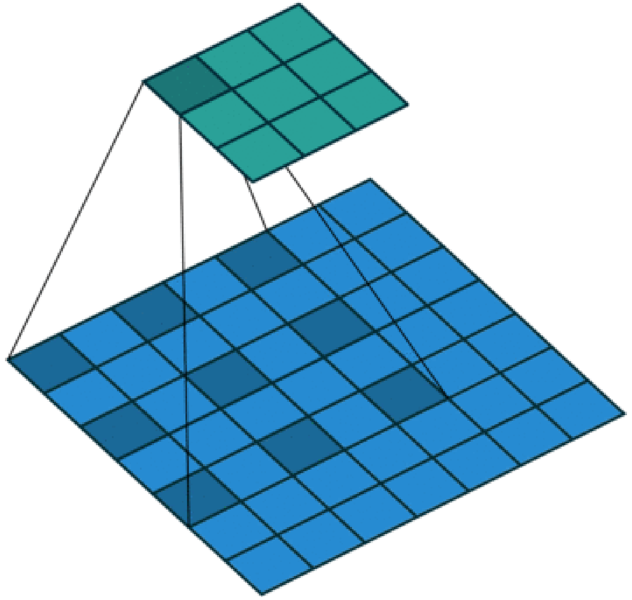
Yes, dilated (atrous) convolution!

Suppose we have a 2D convolution:

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



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



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

Image Credit: [1]

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

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 first proposed in AlexNet [2]:

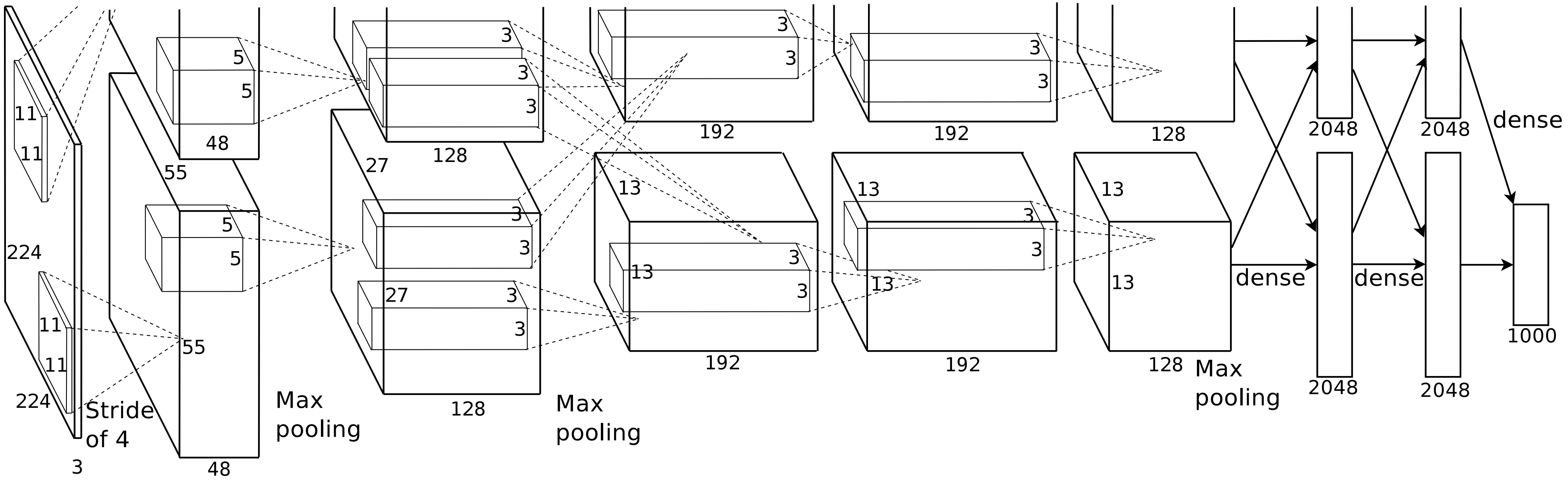


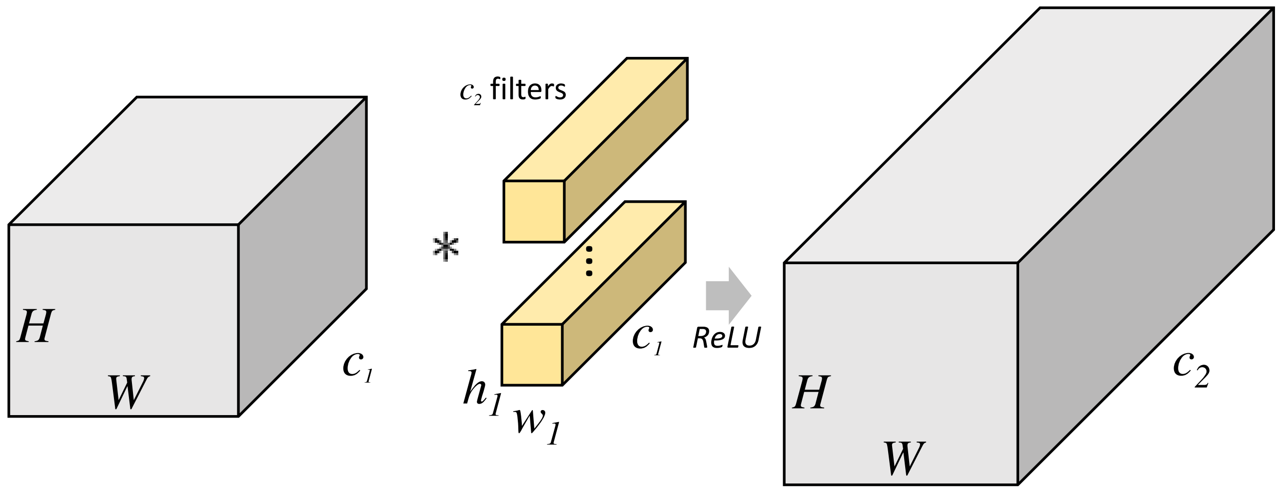
Image Credit: [2]

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 $\mathrm{H} \times W \times c_1$):



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

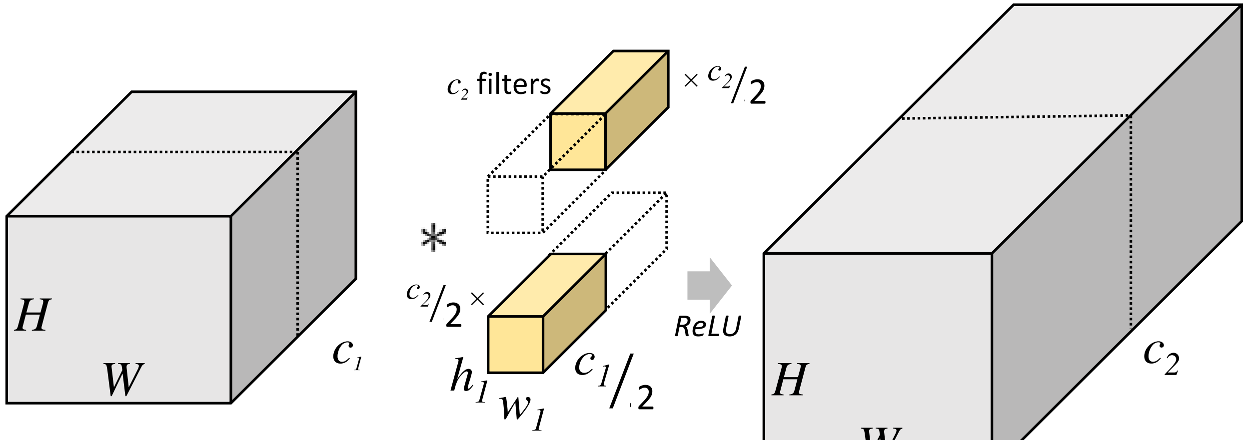
Image Credit: [3]

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 $\mathrm{H} \times W \times c_1$):



We have 2 groups of filters, and the total number of parameters is the same as a single filter before!

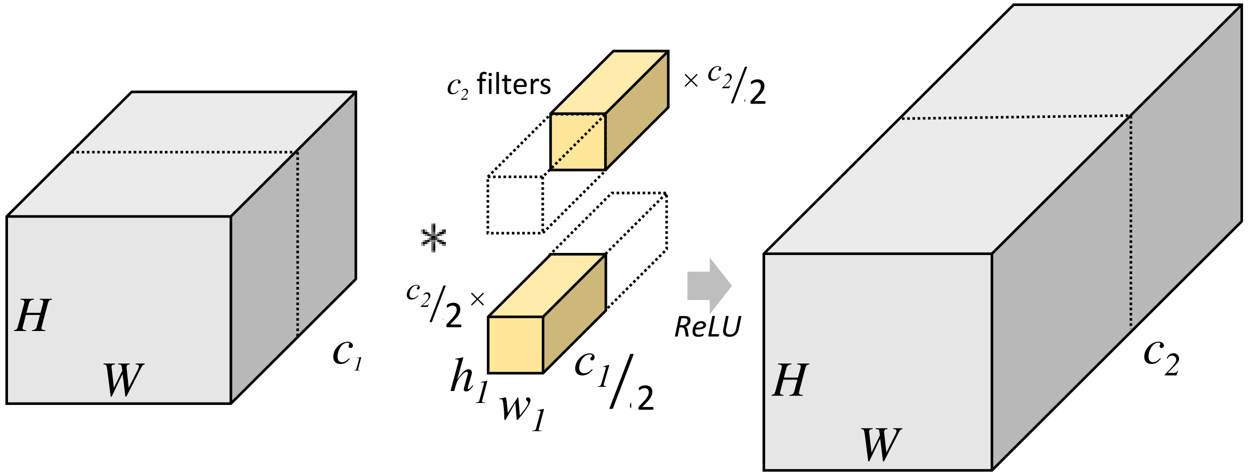
Image Credit: [3]

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 $\mathrm{H} \times W \times c_1$):



We have 2 groups of filters, and the total number of parameters is the same as a single filter before!

Generalize it to multi-groups by yourself!

Image Credit: [3]

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

Separable Convolution

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

Separable Convolution

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

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

Separable Convolution

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

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

Let us look at a 3x3 convolutional kernel:

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

Image Credit: [4]

Separable Convolution

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

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

Let us look at a 3x3 convolutional kernel:

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

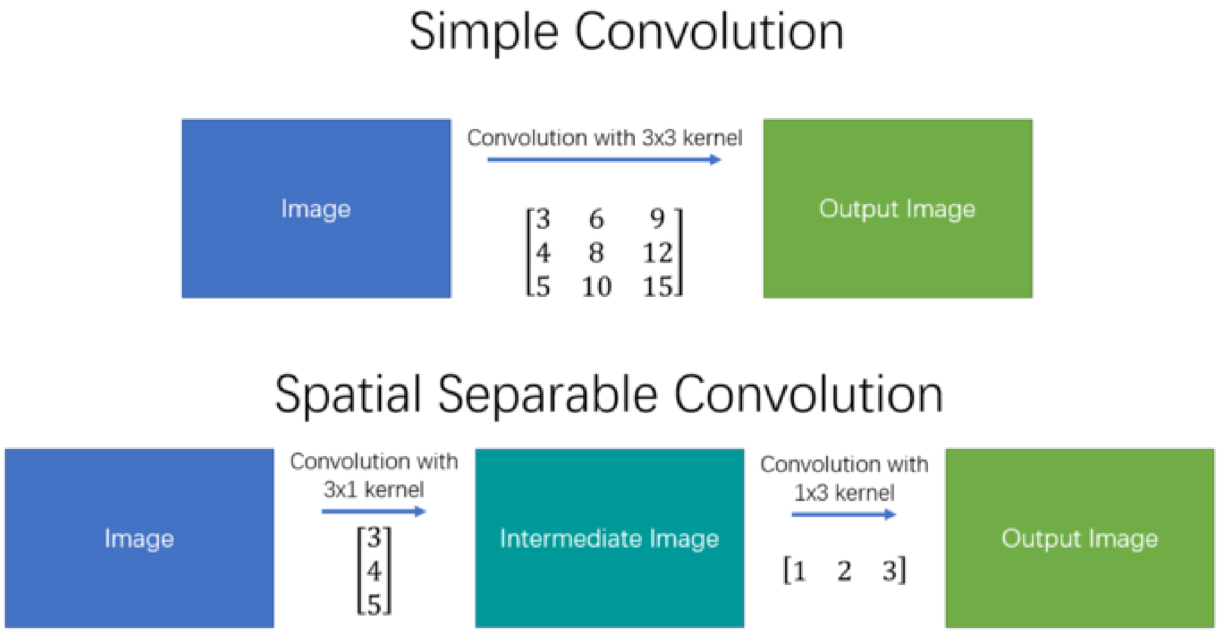


Image Credit: [4]

Separable Convolution

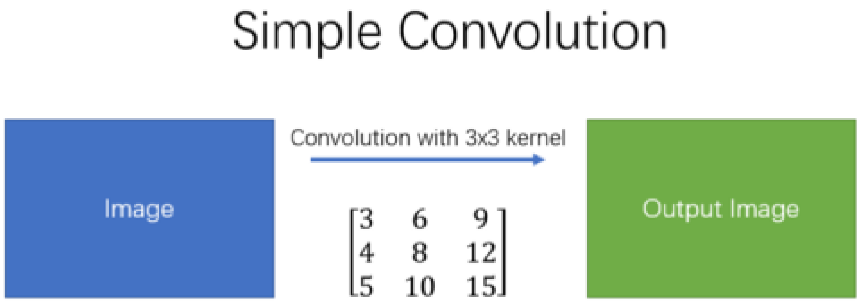
Can we maintain the same shaped input and output in convolution with fewer number of parameters?

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

Let us look at a 3x3 convolutional kernel:

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

Spatial separable kernels are rank one and can not represent full-rank kernels, thus being limited in terms of expressiveness!



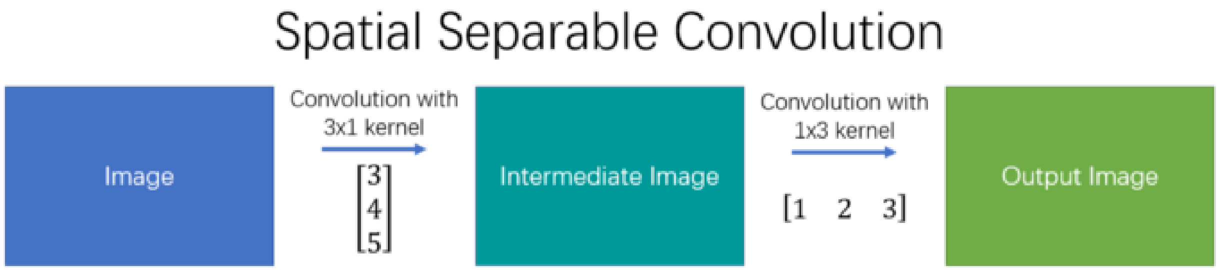


Image Credit: [4]

Separable Convolution

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

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

In practice, one often use *depthwise separable convolution*:

Image Credit: [4]

Separable Convolution

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

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

In practice, one often use *depthwise separable convolution*:

- Depthwise spatial convolution

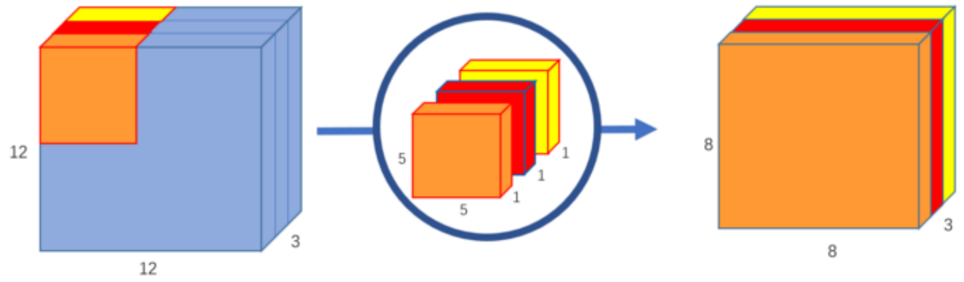


Image Credit: [4]

Separable Convolution

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

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

In practice, one often use *depthwise separable convolution*:

- Depthwise spatial convolution
- Pointwise 1x1 convolution

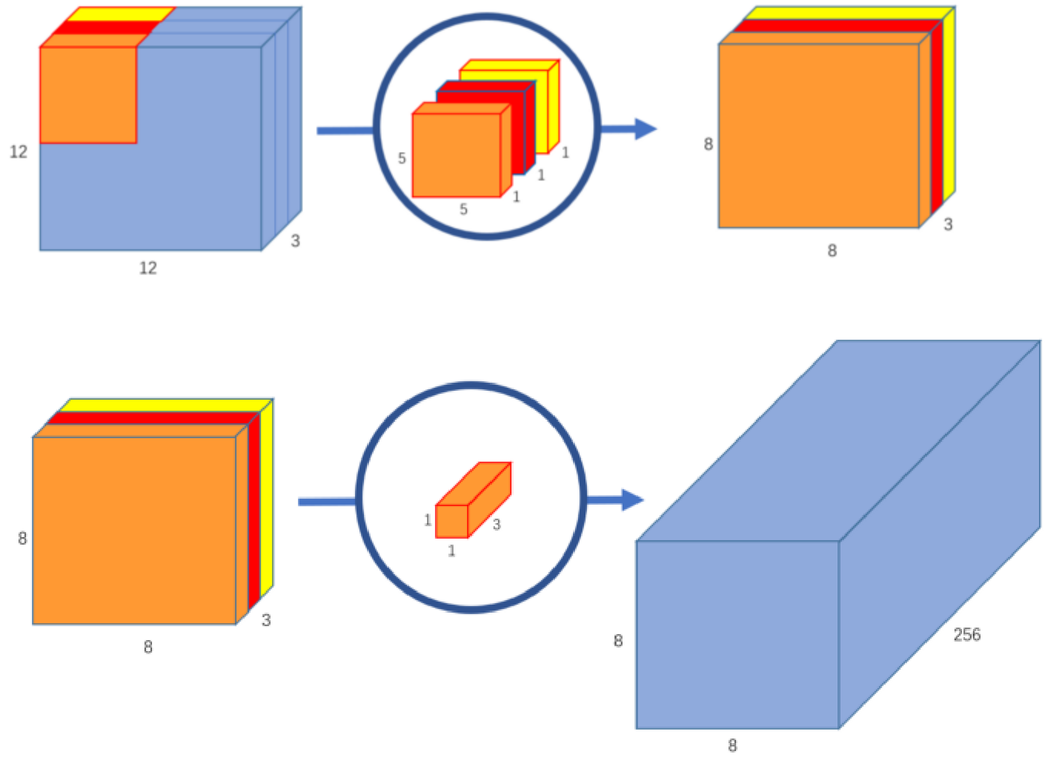


Image Credit: [4]

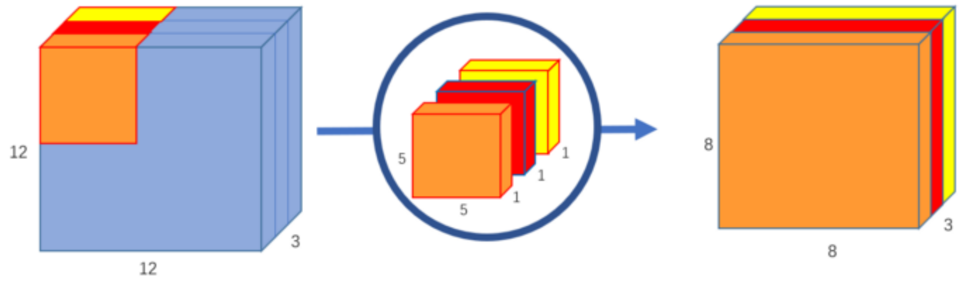
Separable Convolution

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

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

In practice, one often use *depthwise separable convolution*:

- Depthwise spatial convolution
- Pointwise 1x1 convolution



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

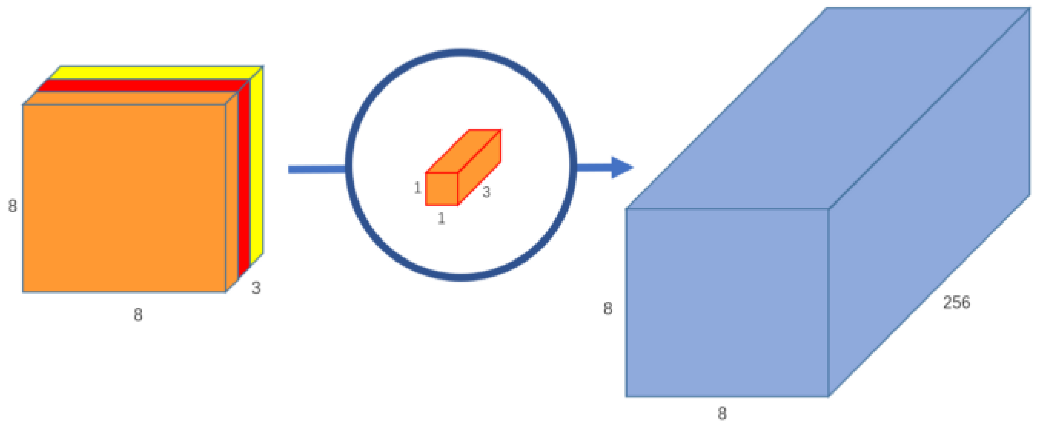


Image Credit: [4]

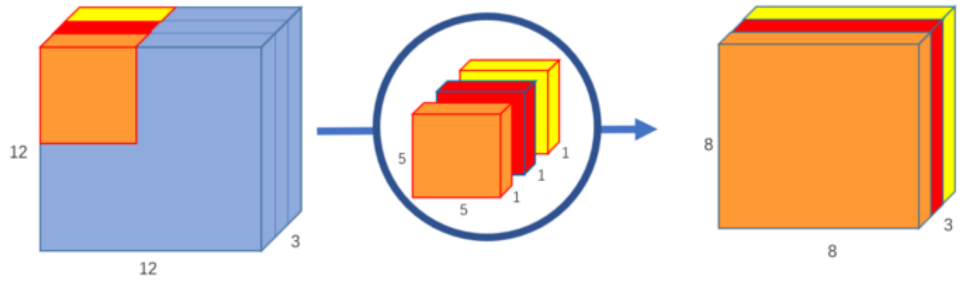
Separable Convolution

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

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

In practice, one often use *depthwise separable convolution*:

- Depthwise spatial convolution
- Pointwise 1x1 convolution



It is a separable convolution: spatial × depth (channel)!

Work out the numbers of parameters and operations, you will find it saves both!

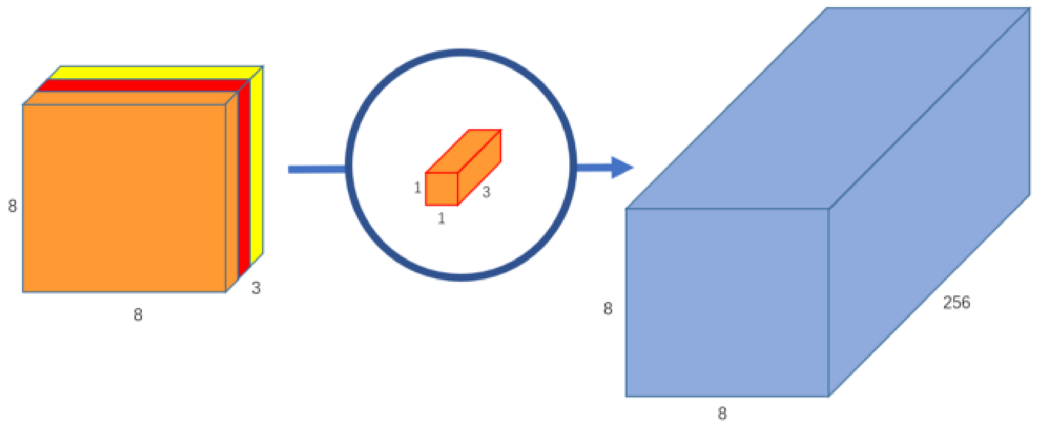


Image Credit: [4]

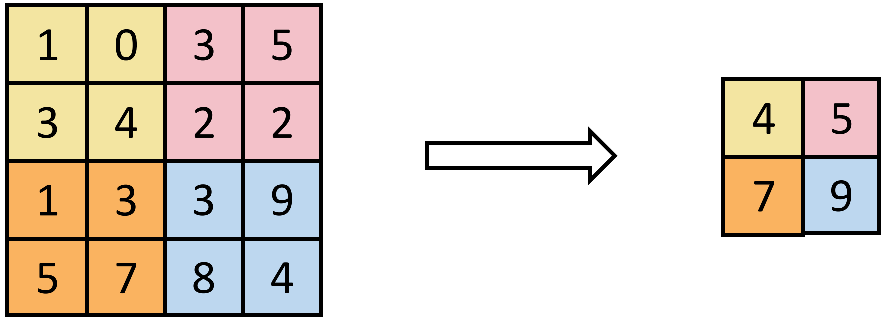
Outline

- 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

Pooling

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

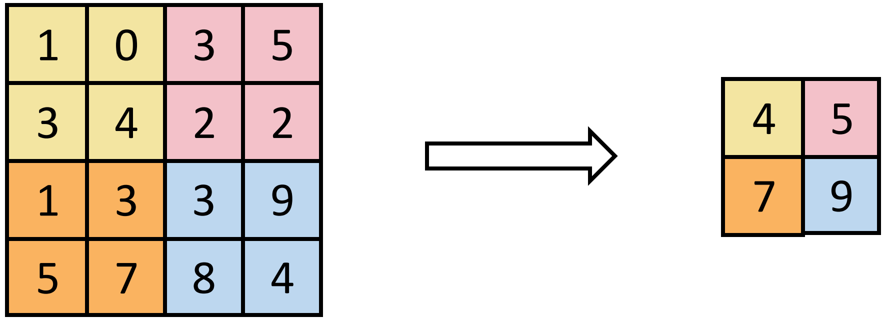
2 X 2 Max Pooling with Stride 2



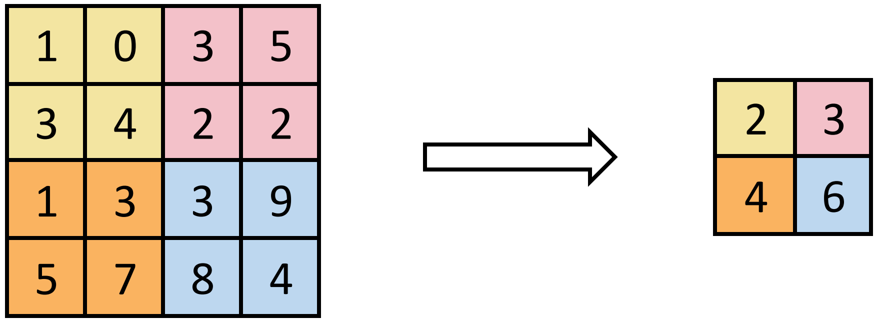
Pooling

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



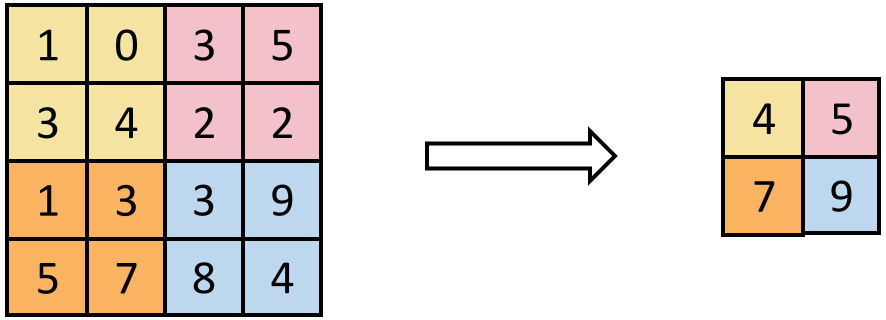
2 X 2 Mean Pooling with Stride 2



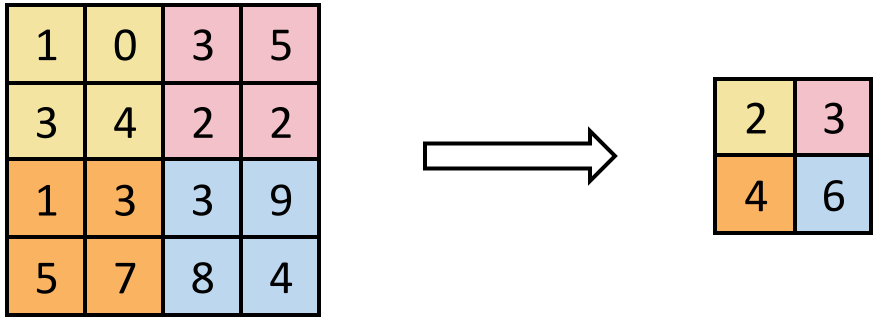
Pooling

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



Pooling gives you permutation-invariance!

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**

Convolutional Neural Networks (CNNs)

Let us look at an example CNN:

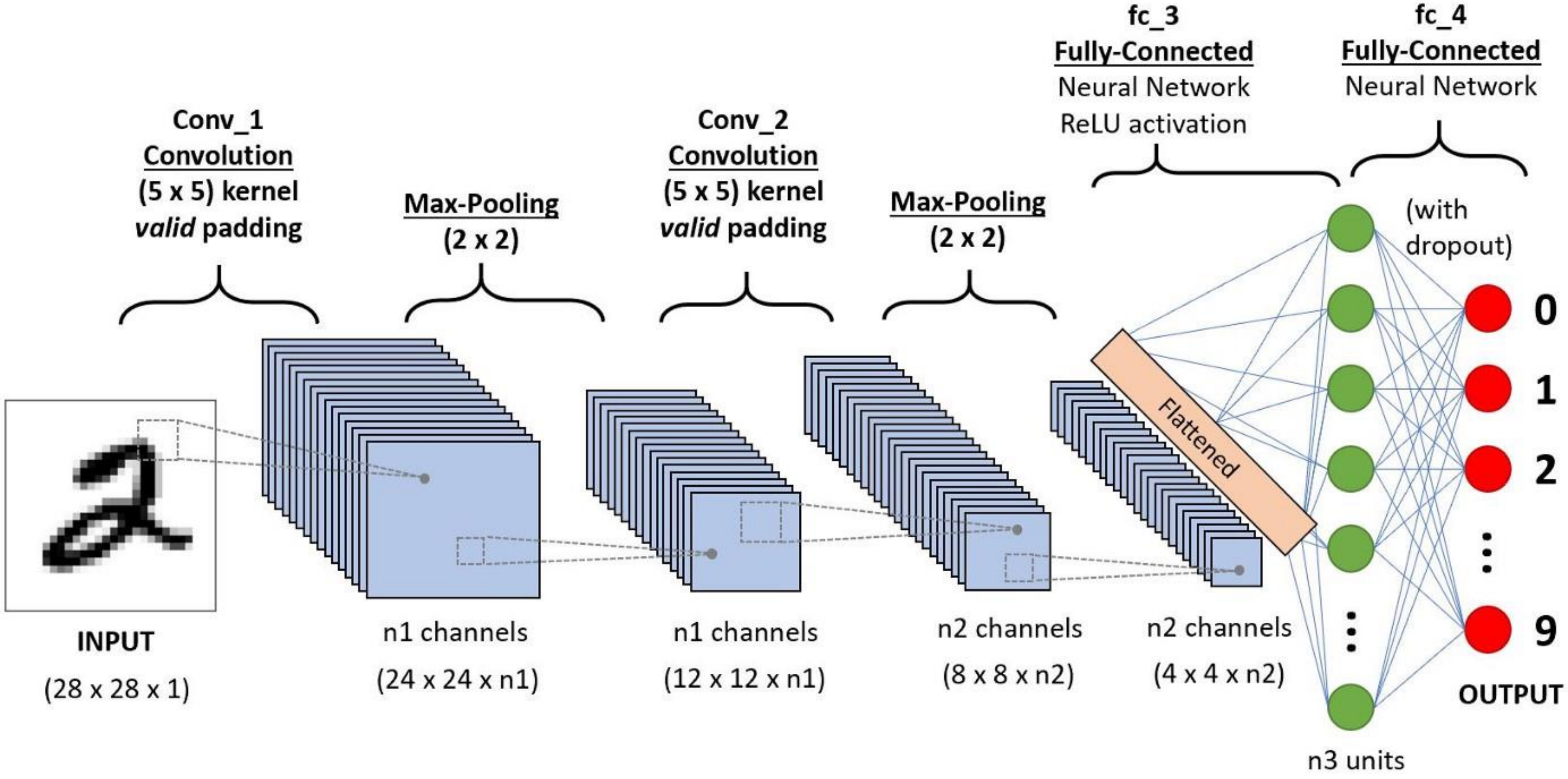


Image Credit: [5]

Translation/Shift Invariance

Suppose background does not change and one only shifts the foreground object, pooling gives you shift-invariance!

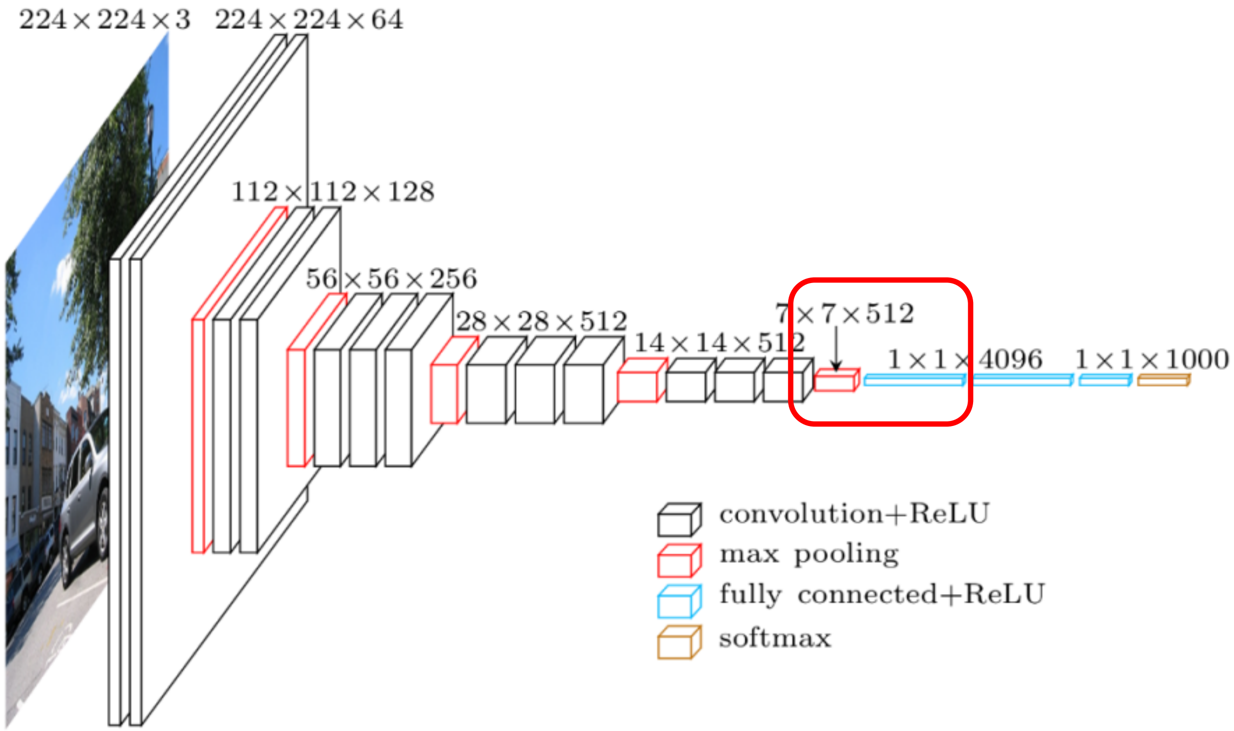
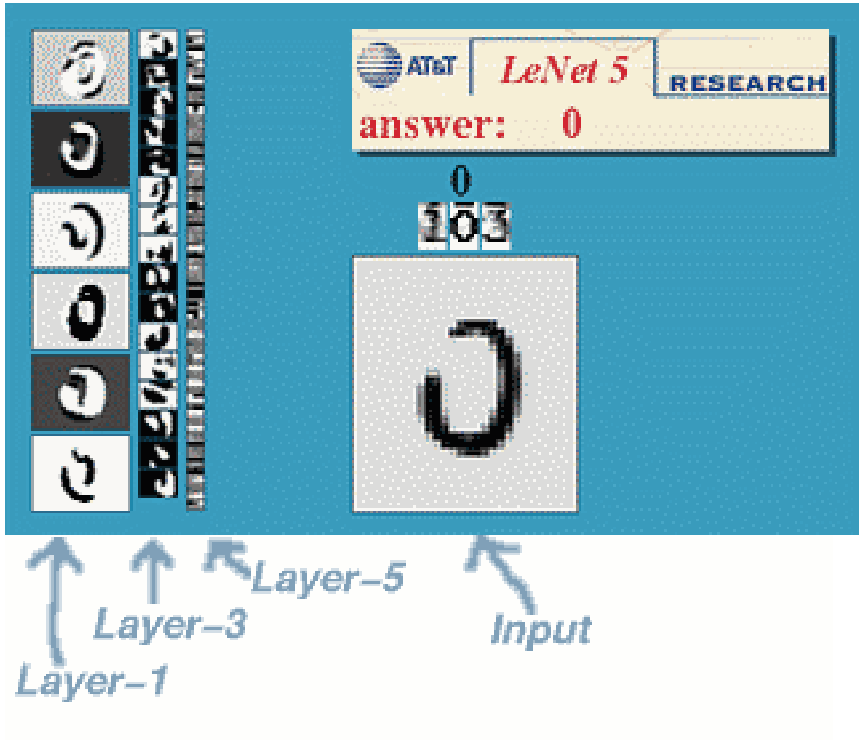


Image Credit: [6]

Translation/Shift Equivariance Invariance

Yann LeCun's LeNet Demo:



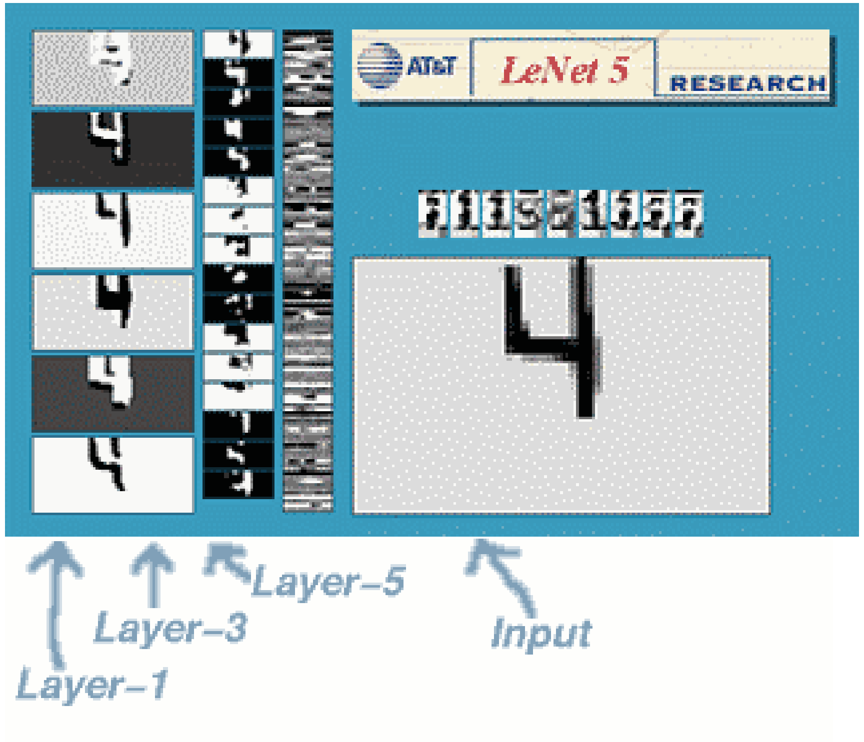


Image Credit: [7]

More on Invariance & Equivariance

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].

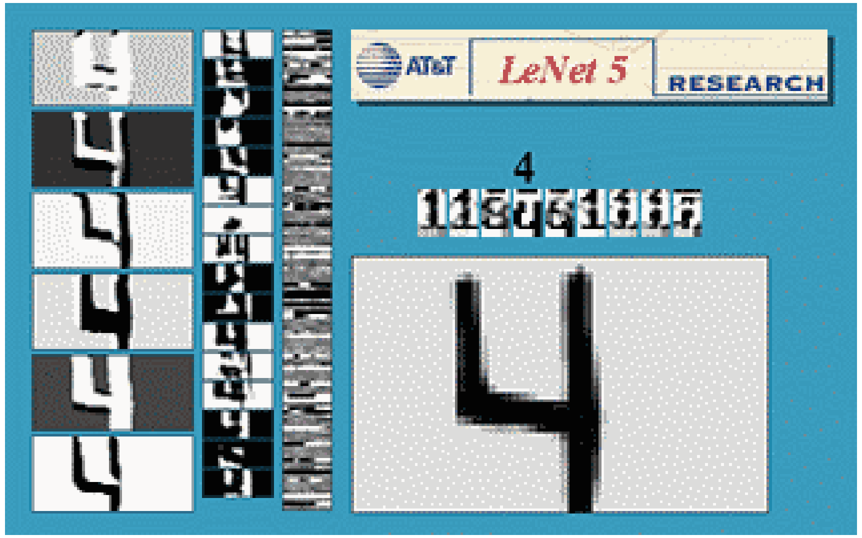


Image Credit: [8]

U-Net

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

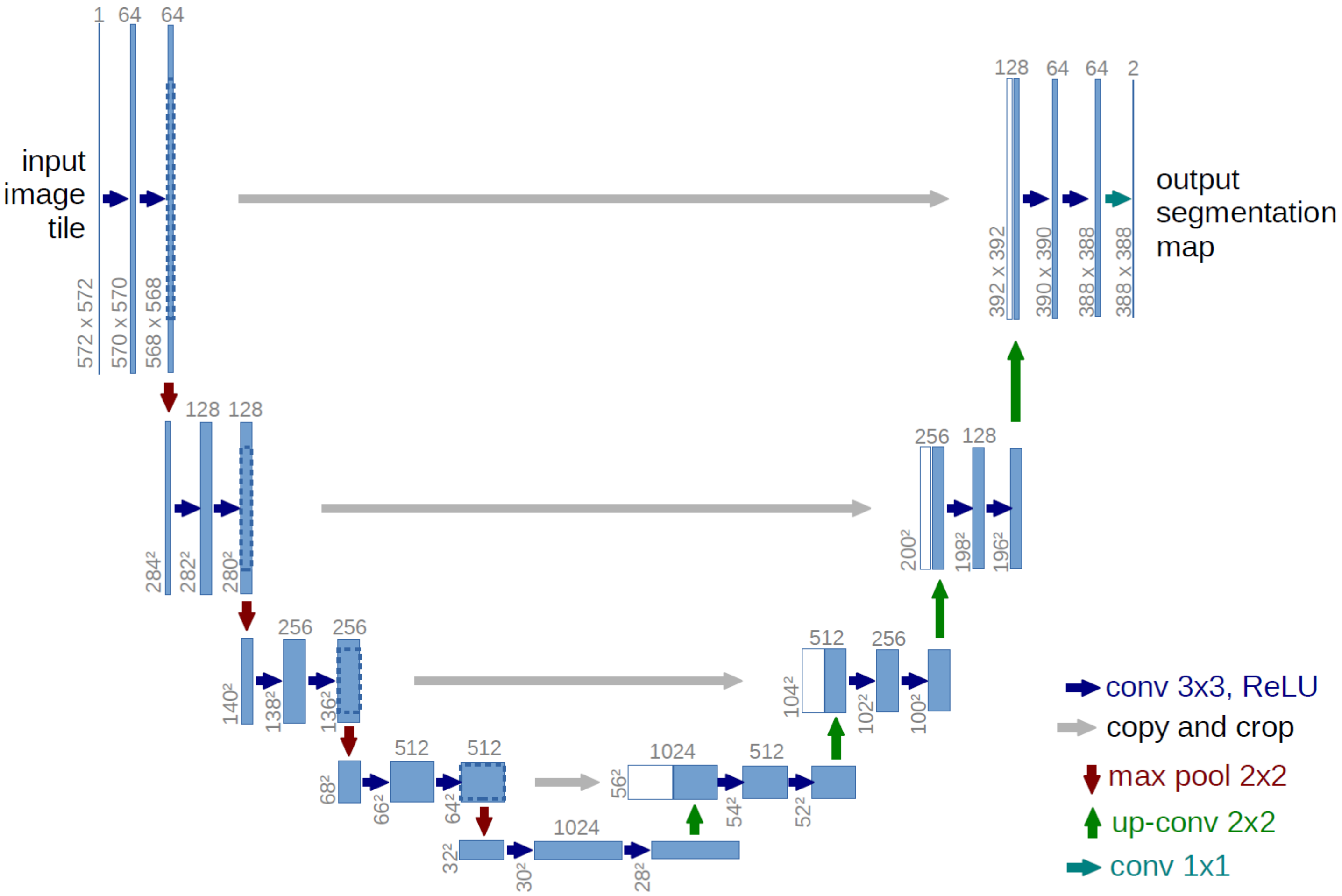
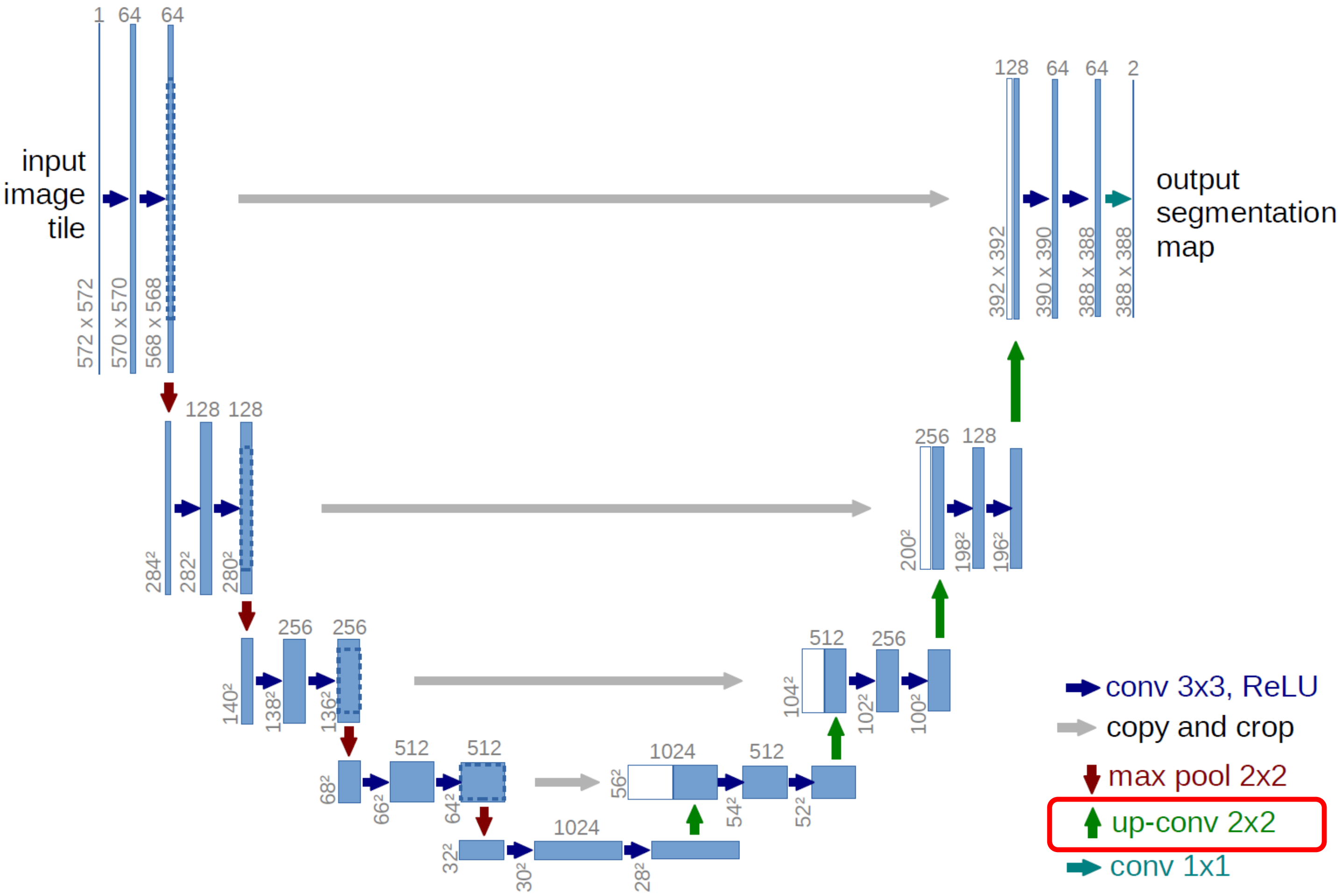


Image Credit: [10]

U-Net

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



Transposed Convolution

Image Credit: [10]

ResNet

ResNet [11] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.

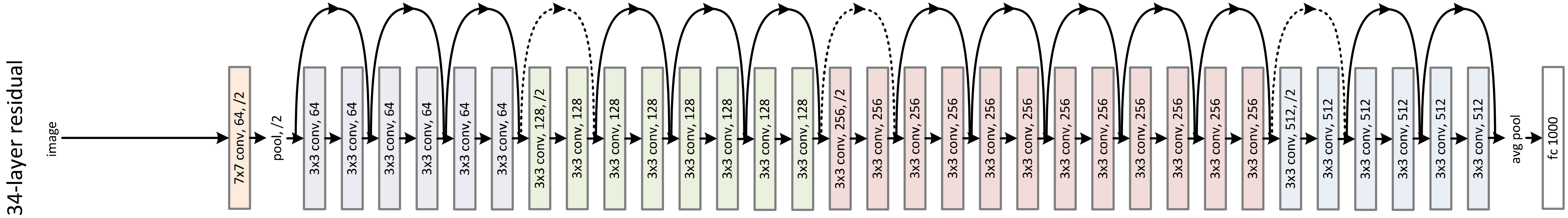


Image Credit: [11]

ResNet

ResNet [11] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.

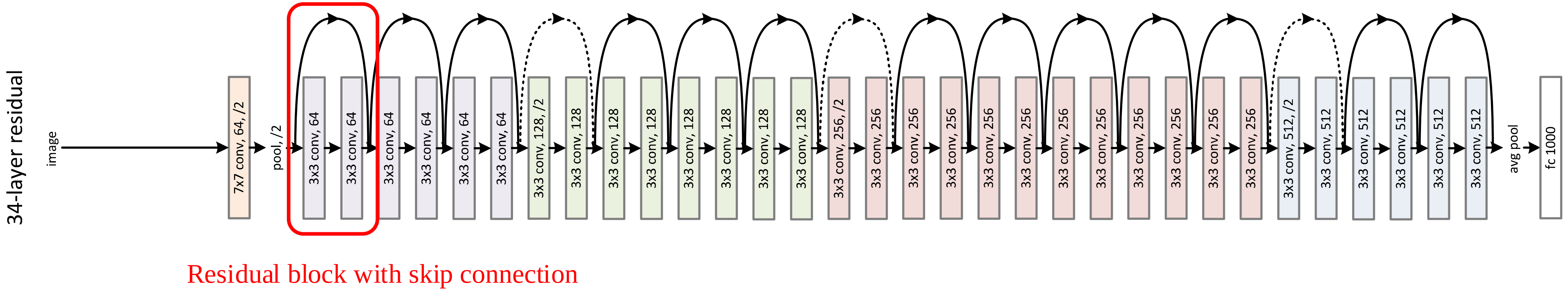
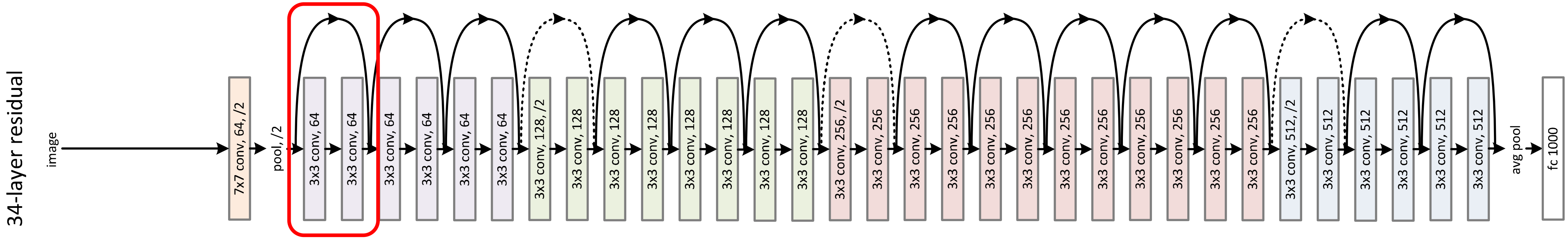


Image Credit: [11]

ResNet

ResNet [11] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.



Residual block with skip connection

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

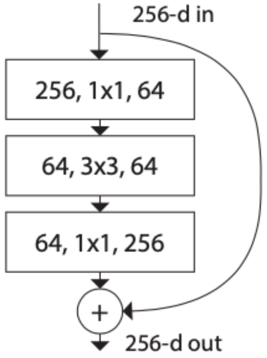
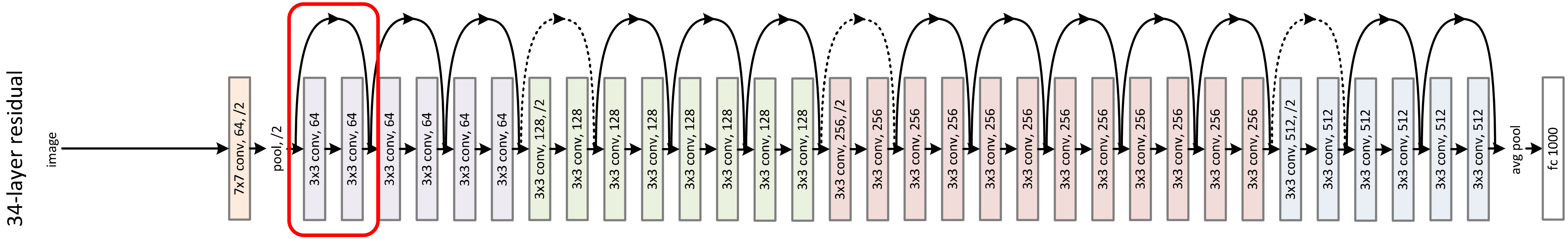


Image Credit: [11]

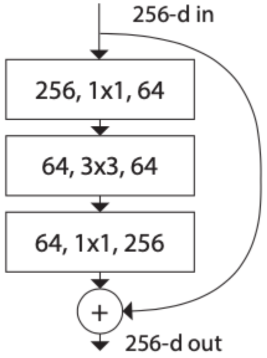
ResNet

ResNet [11] is a popular fully-convolutional CNN architecture for pixel-level tasks like image segmentation.



Residual block with skip connection

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)!

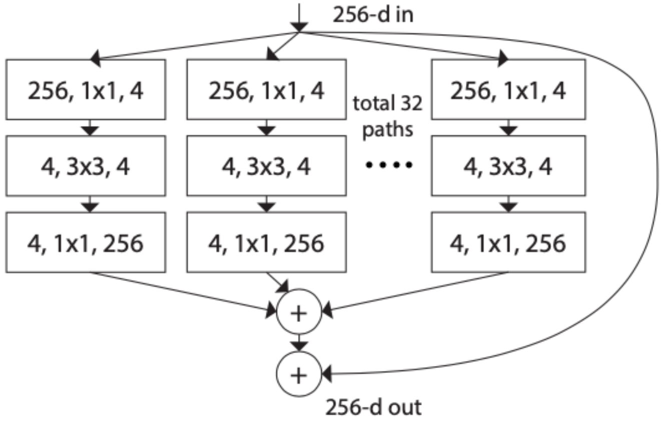


Image Credit: [11,12]

MobileNet

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

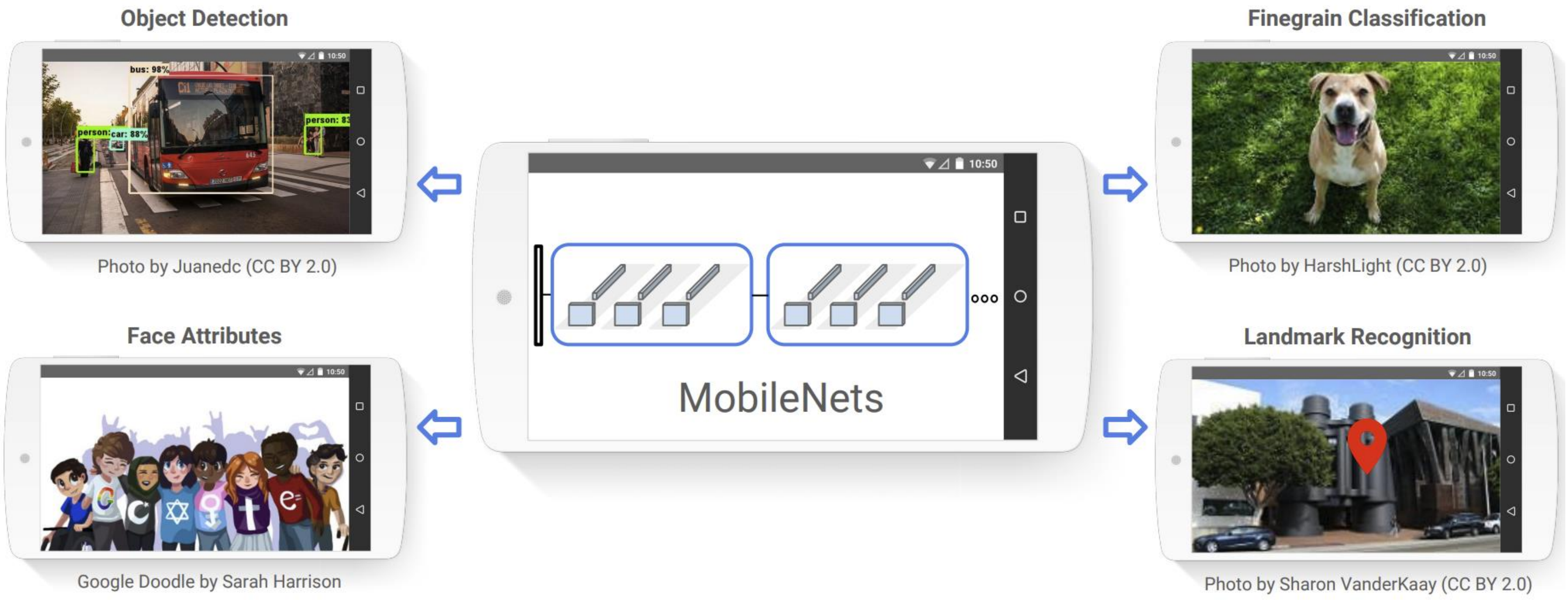
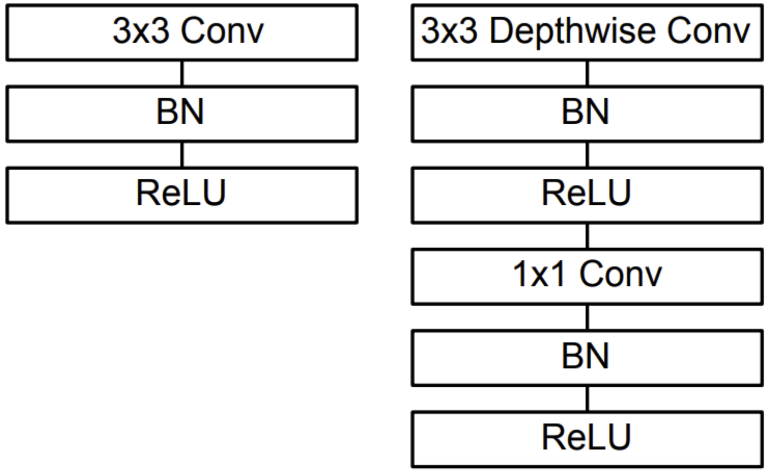


Image Credit: [13]

MobileNet

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

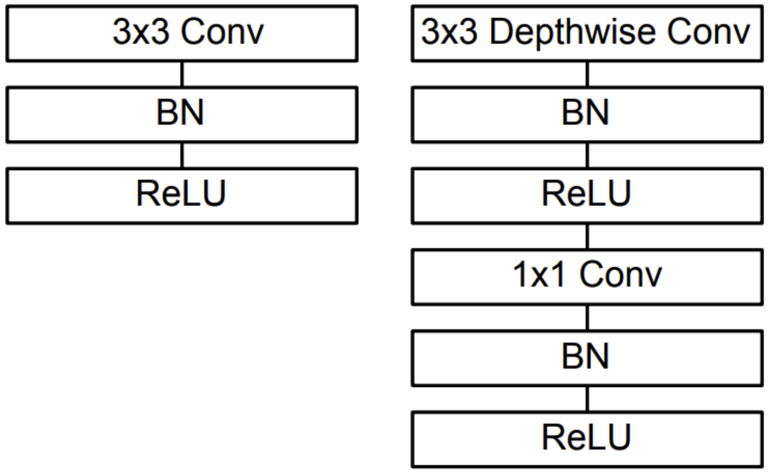


Replace the vanilla conv layer with depthwise separable convolutional layer

Image Credit: [13]

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

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Image Credit: [13]

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. InMedical Image Computing and Computer-Assisted Intervention (MICCAI) 2015.

[11] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings 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. InProceedings 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.

Questions?