



Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

([http://cocl.us/pytorch\\_link\\_top](http://cocl.us/pytorch_link_top))



## What's Convolution </h1 >

# Table of Contents

In this lab, you will study convolution and review how the different operations change the relationship between input and output.

- [What is Convolution](#)
- [Determining the Size of Output](#)
- [Stride](#)
- [Zero Padding](#)
- [Practice Questions](#)

Estimated Time Needed: **25 min**

Import the following libraries:

In [1]:

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage, misc
```

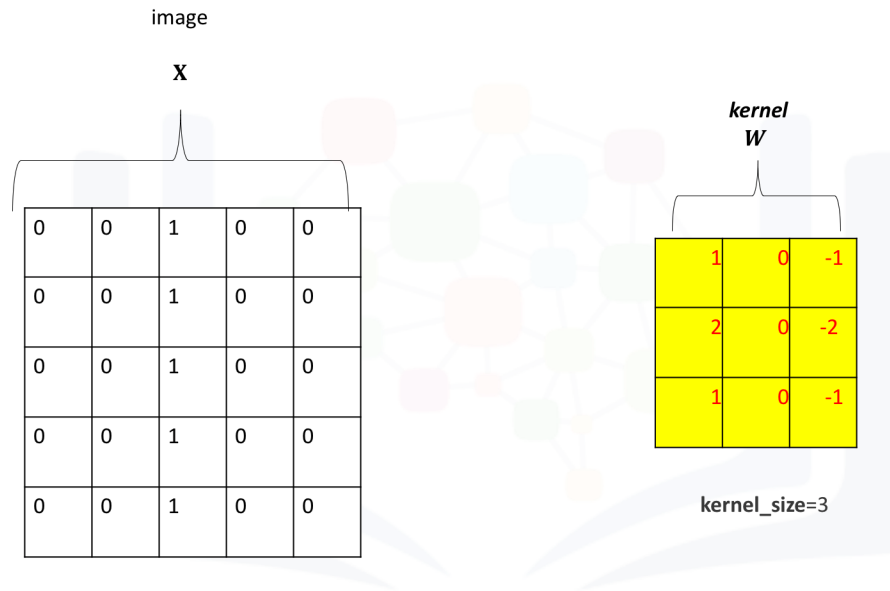
## What is Convolution?

Convolution is a linear operation similar to a linear equation, dot product, or matrix multiplication. Convolution has several advantages for analyzing images. As discussed in the video, convolution preserves the relationship between elements, and it requires fewer parameters than other methods.

You can see the relationship between the different methods that you learned:

\$\$\$linear \ equation :y=wx+b\$\$\$linear \ equation \ with \ multiple \ variables \ where \  $\mathbf{x}$  \ is \ a \ vector \  $\mathbf{y}=\mathbf{wx}+b$ \$\$\$ \ matrix \ multiplication \ where \  $\mathbf{X}$  \ in \ a \ matrix \  $\mathbf{y}=\mathbf{wX}+\mathbf{b}$ \$\$\$ \ convolution \ where \  $\mathbf{X}$  \ and \  $\mathbf{Y}$  \ is \ a \ tensor \  $\mathbf{Y}=\mathbf{w}*\mathbf{X}+\mathbf{b}$ \$\$\$

In convolution, the parameter **w** is called a kernel. You can perform convolution on images where you let the variable image denote the variable X and w denote the parameter.



Create a two-dimensional convolution object by using the constructor `Conv2d`, the parameter `in_channels` and `out_channels` will be used for this section, and the parameter `kernel_size` will be three.

In [2]:

```
conv = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3)
conv
```

Out[2]:

```
Conv2d(1, 1, kernel_size=(3, 3), stride=(1, 1))
```

Because the parameters in `nn.Conv2d` are randomly initialized and learned through training, give them some values.

In [3]:

```
conv.state_dict()['weight'][0][0]=torch.tensor([[1.0,0,-1.0],[2.0,0,-2.0],[1.0,0.0,-1.0]])
conv.state_dict()['bias'][0]=0.0
conv.state_dict()
```

Out[3]:

```
OrderedDict([('weight',
               tensor([[[[ 1.,  0., -1.],
                           [ 2.,  0., -2.],
                           [ 1.,  0., -1.]]]])),
              ('bias', tensor([0.]))])
```

Create a dummy tensor to represent an image. The shape of the image is (1,1,5,5) where:

(number of inputs, number of channels, number of rows, number of columns)

Set the third column to 1:

In [4]:

```
image=torch.zeros(1,1,5,5)
image[0,0,:,2]=1
image
```

Out[4]:

```
tensor([[[[0., 0., 1., 0., 0.],
          [0., 0., 1., 0., 0.],
          [0., 0., 1., 0., 0.],
          [0., 0., 1., 0., 0.],
          [0., 0., 1., 0., 0.]]]])
```

Call the object `conv` on the tensor `image` as an input to perform the convolution and assign the result to the tensor `z`.

In [5]:

```
z=conv(image)
z
```

Out[5]:

```
tensor([[[[-4., 0., 4.],
          [-4., 0., 4.],
          [-4., 0., 4.]]]], grad_fn=<Mkld
nnConvolutionBackward>)
```

The following animation illustrates the process, the kernel performs at the element-level multiplication on every element in the image in the corresponding region. The values are then added together. The kernel is then shifted and the process is repeated.

0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0

## Determining the Size of the Output

The size of the output is an important parameter. In this lab, you will assume square images. For rectangular images, the same formula can be used in for each dimension independently.

Let  $M$  be the size of the input and  $K$  be the size of the kernel. The size of the output is given by the following formula:

$$M_{\text{new}} = M - K + 1$$

Create a kernel of size 2:

In [6]:

```
K=2
conv1 = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=K)
conv1.state_dict()['weight'][0][0]=torch.tensor([[1.0,1.0],
[1.0,1.0]])
conv1.state_dict()['bias'][0]=0.0
conv1.state_dict()
conv1
```

Out[6]:

```
Conv2d(1, 1, kernel_size=(2, 2), stride=(1, 1))
```

Create an image of size 2:

In [7]:

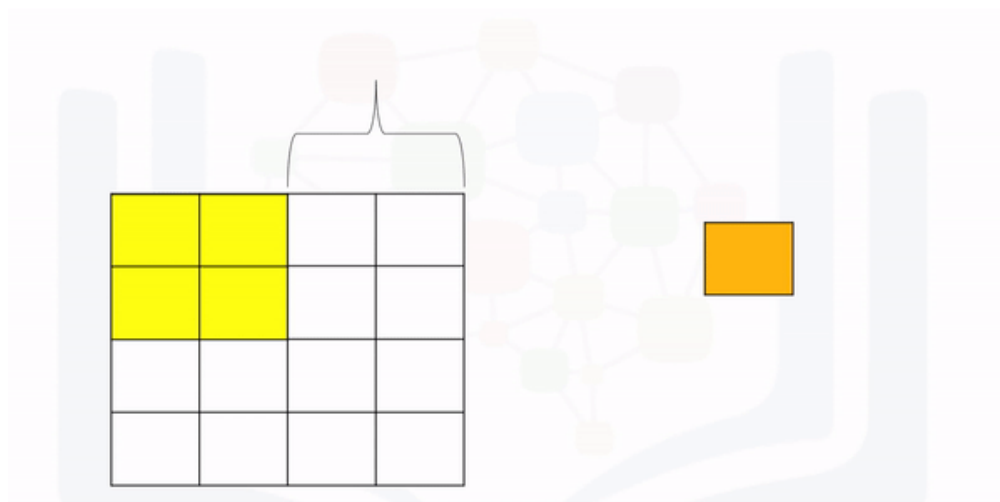
```
M=4
image1=torch.ones(1,1,M,M)
```



The following equation provides the output:

$$M_{\text{new}} = M - K + 1 \quad M_{\text{new}} = 4 - 2 + 1 \quad M_{\text{new}} = 3$$

The following animation illustrates the process: The first iteration of the kernel overlay of the images produces one output. As the kernel is of size K, there are M-K elements for the kernel to move in the horizontal direction. The same logic applies to the vertical direction.



Perform the convolution and verify the size is correct:

In [8]:

```
z1=conv1(image1)
print("z1:",z1)
print("shape:",z1.shape[2:4])
```

```
z1: tensor([[[[4., 4., 4.],
              [4., 4., 4.],
              [4., 4., 4.]]]], grad_fn=<MkldnnConvolutionBackward>)
shape: torch.Size([3, 3])
```



# Stride parameter

The parameter stride changes the number of shifts the kernel moves per iteration. As a result, the output size also changes and is given by the following formula:

$$M_{\text{new}} = \frac{M - K}{\text{stride}} + 1$$

Create a convolution object with a stride of 2:

In [9]:

```
conv3 = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=2, stride=2)

conv3.state_dict()['weight'][0][0] = torch.tensor([[1.0, 1.0], [1.0, 1.0]])
conv3.state_dict()['bias'][0] = 0.0
conv3.state_dict()
```

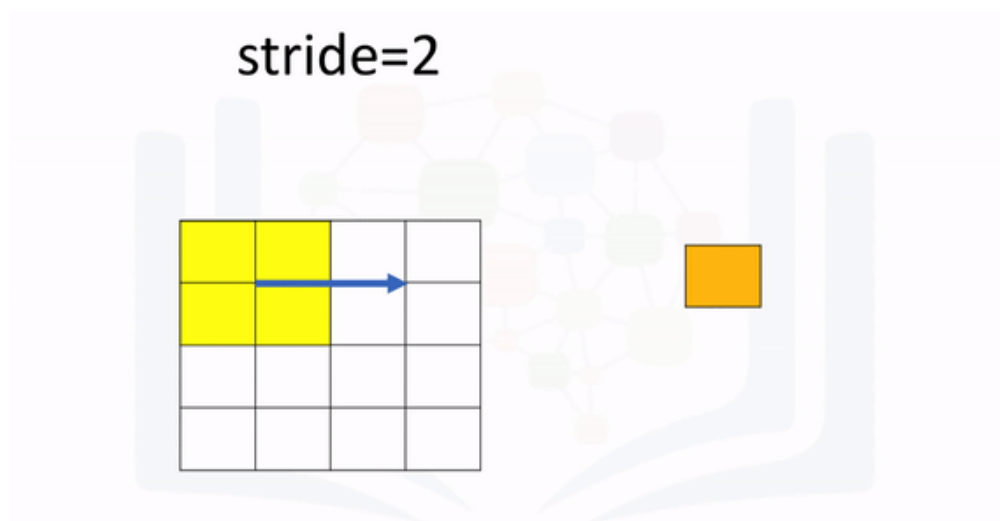
Out[9]:

```
OrderedDict([('weight',
              tensor([[[[1., 1.],
                        [1., 1.]]]])),
            ('bias', tensor([0.]])])]
```

For an image with a size of 4, calculate the output size:

$$M_{\text{new}} = \frac{M - K}{\text{stride}} + 1 \quad M_{\text{new}} = \frac{4 - 2}{2} + 1 \quad M_{\text{new}} = 2$$

The following animation illustrates the process: The first iteration of the kernel overlay of the images produces one output. Because the kernel is of size  $K$ , there are  $M-K+1$  elements. The stride is 2 because it will move 2 elements at a time. As a result, you divide  $M-K+1$  by the stride value 2:



Perform the convolution and verify the size is correct:

In [10]:

```
z3=conv3(image1)

print("z3:", z3)
print("shape:", z3.shape[2:4])

z3: tensor([[[[4., 4.],
              [4., 4.]]]], grad_fn=<MkldnnConvolutionBackward>)
shape: torch.Size([2, 2])
```

## Zero Padding

As you apply successive convolutions, the image will shrink. You can apply zero padding to keep the image at a reasonable size, which also holds information at the borders.

In addition, you might not get integer values for the size of the kernel. Consider the following image:

In [11]:

```
image1
```

Out[11]:

```
tensor([[[[1., 1., 1., 1.],
          [1., 1., 1., 1.],
          [1., 1., 1., 1.],
          [1., 1., 1., 1.]]]]])
```

Try performing convolutions with the `kernel_size=2` and a `stride=3`. Use these values:

$$M_{\text{new}} = \frac{M - K}{\text{stride}} + 1$$
$$M_{\text{new}} = \frac{4 - 2}{3} + 1$$
$$M_{\text{new}} = 1.666$$

In [12]:

```
conv4 = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=2, stride=3)
conv4.state_dict()['weight'][0][0]=torch.tensor([[1.0,1.0],[1.0,1.0]])
conv4.state_dict()['bias'][0]=0.0
conv4.state_dict()
z4=conv4(image1)
print("z4:", z4)
print("z4:", z4.shape[2:4])
```

```
z4: tensor([[[[4.]]]], grad_fn=<MkldnnConvolutionBackward>)
z4: torch.Size([1, 1])
```

You can add rows and columns of zeros around the image. This is called padding. In the constructor `Conv2d`, you specify the number of rows or columns of zeros that you want to add with the parameter `padding`.

For a square image, you merely pad an extra column of zeros to the first column and the last column. Repeat the process for the rows. As a result, for a square image, the width and height is the original size plus 2 x the number of padding elements specified. You can then determine the size of the output after subsequent operations accordingly as shown in the following equation where you determine the size of an image after padding and then applying a convolutions kernel of size  $K$ .

$$M' = M + 2 \times \text{padding} \quad M_{\text{new}} = M' - K + 1$$

Consider the following example:

In [13]:

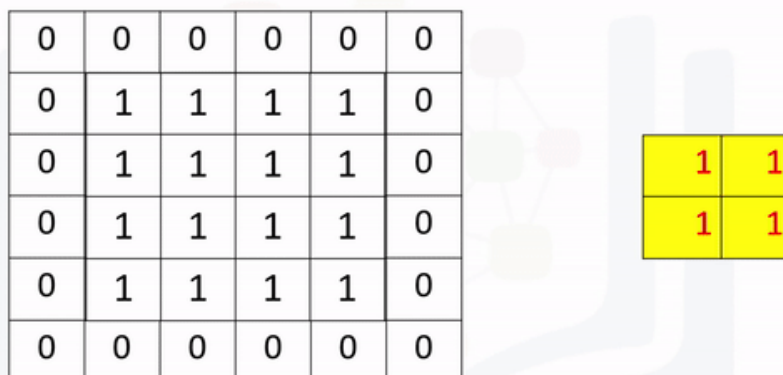
```
conv5 = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=2, stride=3, padding=1)

conv5.state_dict()['weight'][0][0]=torch.tensor([[1.0,1.0], [1.0,1.0]])
conv5.state_dict()['bias'][0]=0.0
conv5.state_dict()
z5=conv5(image1)
print("z5:", z5)
print("z5:", z4.shape[2:4])
```

```
z5: tensor([[[[1., 2.],
              [2., 4.]]]], grad_fn=<MkldnnConvolutionBackward>)
z5: torch.Size([1, 1])
```

In [ ]:

The process is summarized in the following animation:



0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

1	1
1	1

# Practice Question

A kernel of zeros with a kernel size=3 is applied to the following image:

In [14]:

```
Image=torch.randn( (1,1,4,4) )  
Image
```

Out[14]:

```
tensor([[[[-0.4685, -1.2821, -0.2023, -0.09  
50],  
          [-0.4453, -1.3130, -0.4968, -0.26  
10],  
          [-0.9570,  0.7594, -0.0128, -1.77  
85],  
          [ 1.0972,  0.6900, -0.5578, -1.12  
60]]]])
```

Question: Without using the function, determine what the outputs values are as each element:

ANS: As each element of the kernel is zero, and for every output, the image is multiplied by the kernel, the result is always zero

Double-click **here** for the solution.

Question: Use the following convolution object to perform convolution on the tensor `Image` :

In [15]:

```
conv = nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3)
conv.state_dict()['weight'][0][0]=torch.tensor([[0,0,0],[0,0,0],[0,0.0,0]])
conv.state_dict()['bias'][0]=0.0
```

In [16]:

```
conv(Image)
```

Out[16]:

```
tensor([[[[0., 0.],
          [0., 0.]]]], grad_fn=<MkldnnConvolutionBackward>)
```

Double-click **here** for the solution.

Question: You have an image of size 4. The parameters are as follows kernel\_size=2, stride=2. What is the size of the output?

ANS:  $(M-K)/\text{stride} + 1$   $(4-2)/2 + 1 = 2$

Double-click **here** for the solution.

## Get IBM Watson Studio free of charge!

Build and train AI & machine learning models, prepare and analyze data – all in a flexible, hybrid cloud environment. Get IBM Watson Studio Lite Plan free of charge.



### Learn

Get started or get better with built-in learning.



### Create

Use the best of open source tooling with IBM innovation.



### Collaborate

Work smarter using community, work faster with your team.

[Sign Up For a Free Trial](#)

[http://cocl.us/pytorch\\_link\\_bottom](http://cocl.us/pytorch_link_bottom)

## About the Authors:

Joseph Santarcangelo (<https://www.linkedin.com/in/joseph-s-50398b136/>) has a PhD in Electrical Engineering. His research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition.

Other contributors: Michelle Carey (<https://www.linkedin.com/in/michelleccarey/>), Mavis Zhou (<https://www.linkedin.com/in/jiahui-mavis-zhou-a4537814a/>)



Copyright © 2018 [cognitiveclass.ai](https://cognitiveclass.ai?utm_source=bducopyrightlink&utm_medium=dswb&utm_campaign=bigdatauniversity) ([cognitiveclass.ai?utm\\_source=bducopyrightlink&utm\\_medium=dswb&utm\\_campaign=bigdatauniversity](https://cognitiveclass.ai?utm_source=bducopyrightlink&utm_medium=dswb&utm_campaign=bigdatauniversity))  
This notebook and its source code are released under the terms of the [MIT License](https://bigdatauniversity.com/mit-license/) (<https://bigdatauniversity.com/mit-license/>).

In [ ]: