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)

Table of Contents

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

Estimated Time Needed: **25 min**

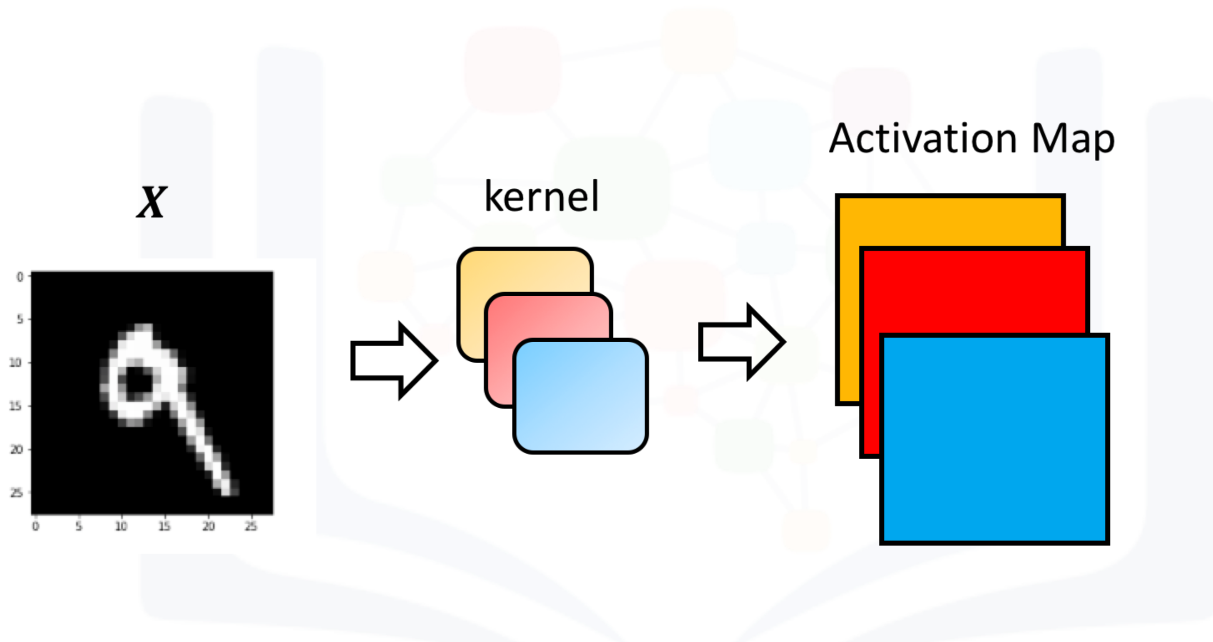
Import the following libraries:

In [1]:

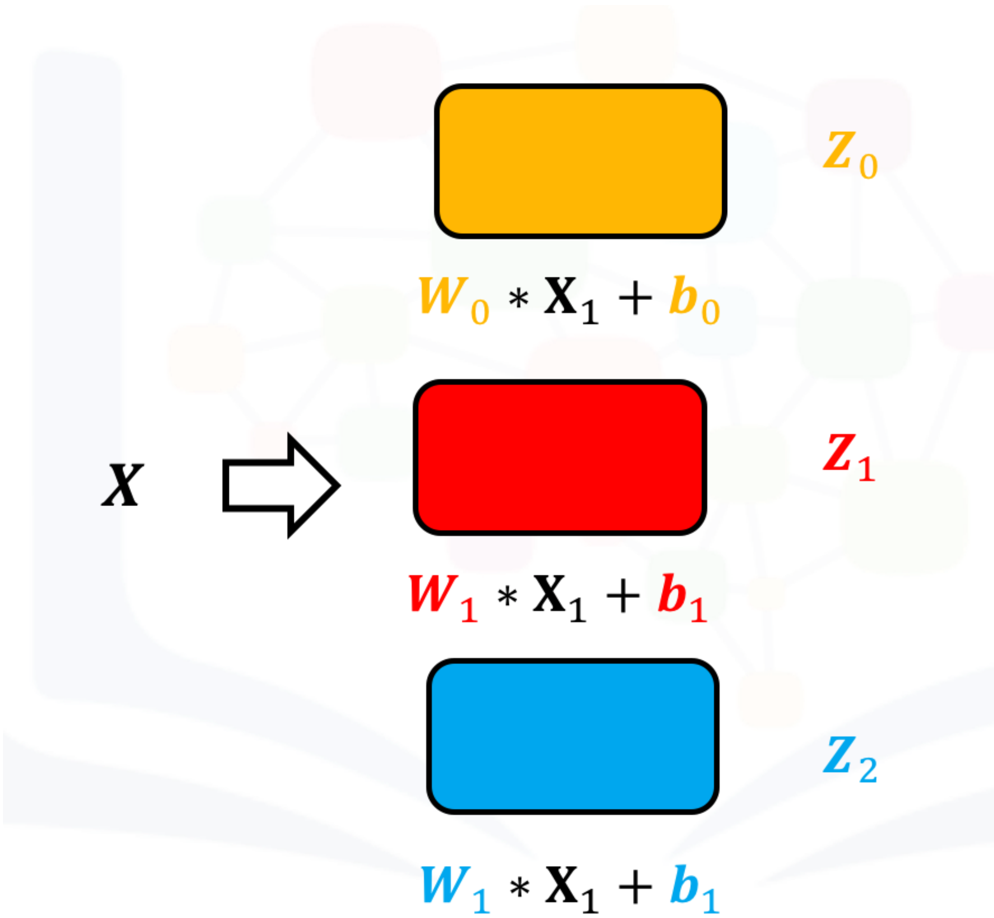
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage, misc
```

Multiple Output Channels

In Pytroch, you can create a `Conv2d` object with multiple outputs. For each channel, a kernel is created, and each kernel performs a convolution independently. As a result, the number of outputs is equal to the number of channels. This is demonstrated in the following figure. The number 9 is convolved with three kernels: each of a different color. There are three different activation maps represented by the different colors.



Symbolically, this can be represented as follows:



Create a `Conv2d` with three channels:

In [2]:

```
conv1 = nn.Conv2d(in_channels=1, out_channels=3,kernel_size=3)
```

Pytorch randomly assigns values to each kernel. However, use kernels that have been developed to detect edges:

In [3]:

```
Gx=torch.tensor([[1.0,0,-1.0],[2.0,0,-2.0],[1.0,0.0,-1.0]])
Gy=torch.tensor([[1.0,2.0,1.0],[0.0,0.0,0.0],[-1.0,-2.0,-1.0]])

conv1.state_dict()['weight'][0][0]=Gx
conv1.state_dict()['weight'][1][0]=Gy
conv1.state_dict()['weight'][2][0]=torch.ones(3,3)
```

Each kernel has its own bias, so set them all to zero:

In [4]:

```
conv1.state_dict()['bias'][:]=torch.tensor([0.0,0.0,0.0])
conv1.state_dict()['bias']
```

Out[4]:

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

Print out each kernel:

In [5]:

```
for x in conv1.state_dict()['weight']:
    print(x)
```

```
tensor([[[ 1.,  0., -1.],
         [ 2.,  0., -2.],
         [ 1.,  0., -1.]]])
tensor([[[ 1.,  2.,  1.],
         [ 0.,  0.,  0.],
         [-1., -2., -1.]]])
tensor([[[1., 1., 1.],
         [1., 1., 1.],
         [1., 1., 1.]]])
```

Create an input `image` to represent the input X:

In [6]:

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

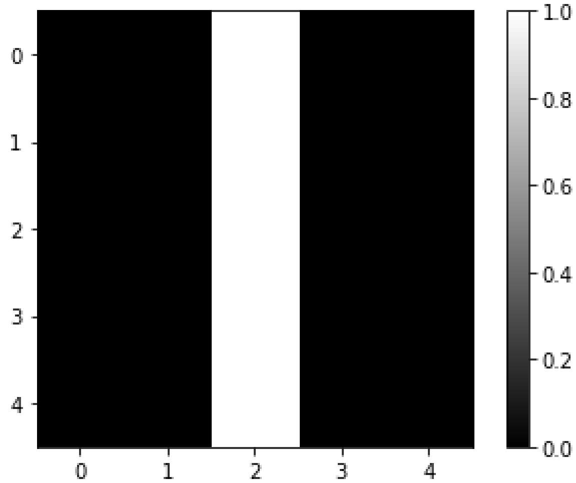
Out[6]:

```
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.]]]])
```

Plot it as an image:

In [7]:

```
plt.imshow(image[0,0,:,:].numpy(), interpolation='nearest', cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```



Perform convolution using each channel:

In [8]:

```
out=conv1(image)
```

The result is a 1x3x3x3 tensor. This represents one sample with three channels, and each channel contains a 3x3 image. The same rules that govern the shape of each image were discussed in the last section.

In [9]:

```
out.shape
```

Out[9]:

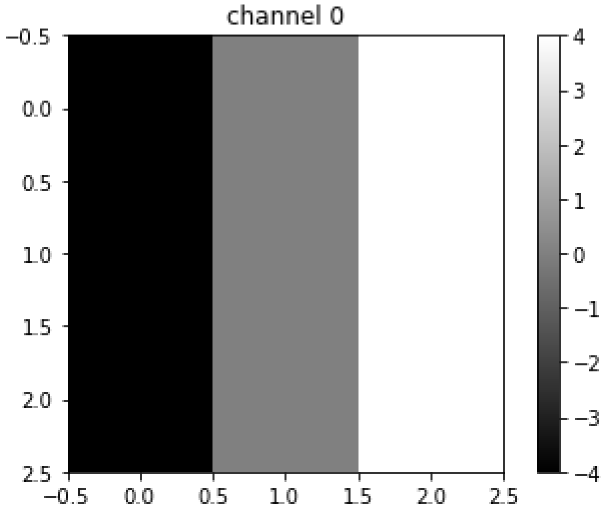
```
torch.Size([1, 3, 3, 3])
```

Print out each channel as a tensor or an image:

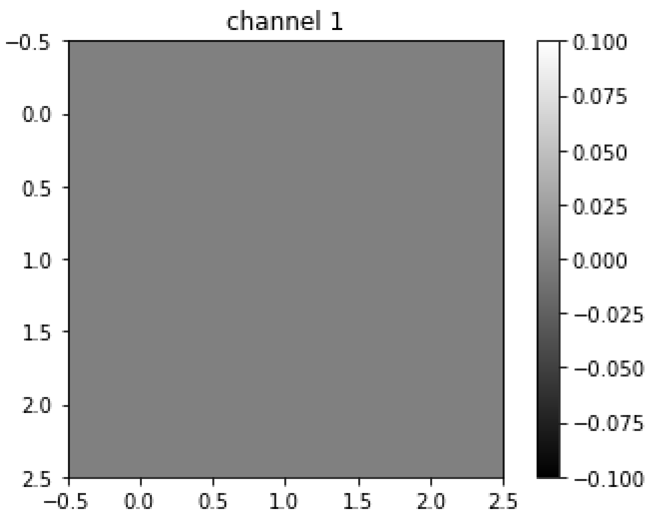
In [10]:

```
for channel,image in enumerate(out[0]):
    plt.imshow(image.detach().numpy(), interpolation='nearest', cmap=plt.cm.gray)
    print(image)
    plt.title("channel {}".format(channel))
    plt.colorbar()
    plt.show()
```

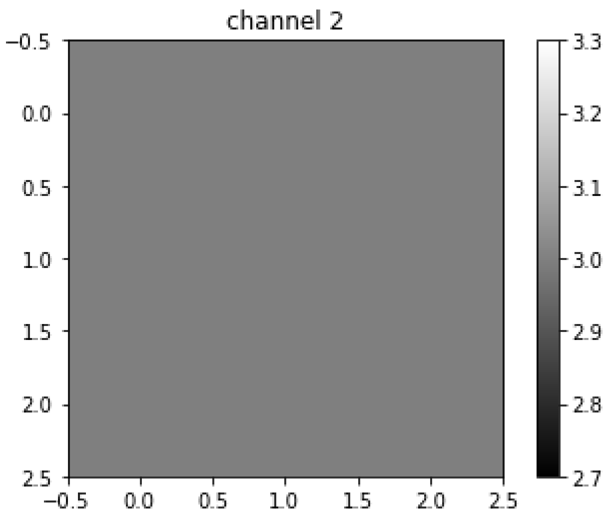
```
tensor([[-4.,  0.,  4.],
        [-4.,  0.,  4.],
        [-4.,  0.,  4.]], grad_fn=<SelectBackward>)
```



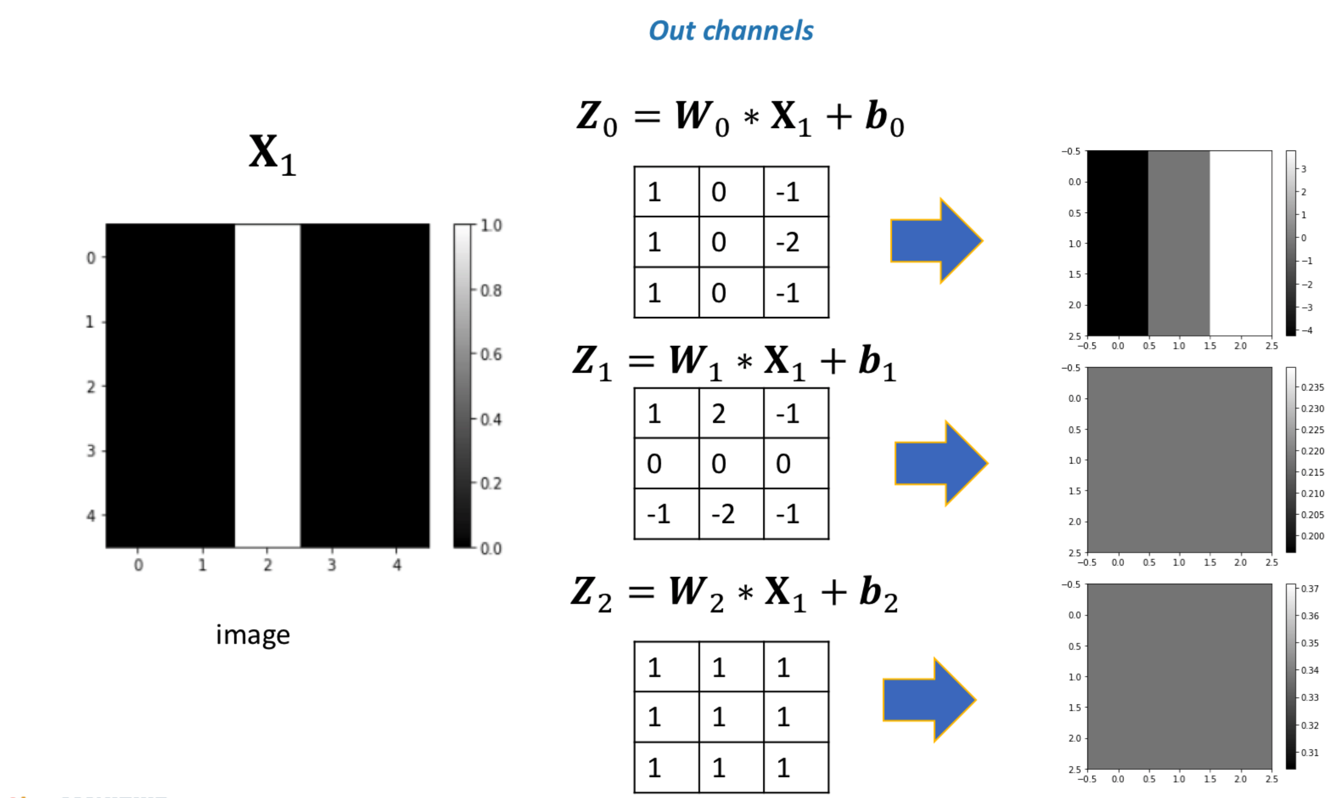

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], grad_fn=<SelectBackward>)
```




```
tensor([[3., 3., 3.],
        [3., 3., 3.],
        [3., 3., 3.]], grad_fn=<SelectBackward>)
```



Different kernels can be used to detect various features in an image. You can see that the first channel fluctuates, and the second two channels produce a constant value. The following figure summarizes the process:

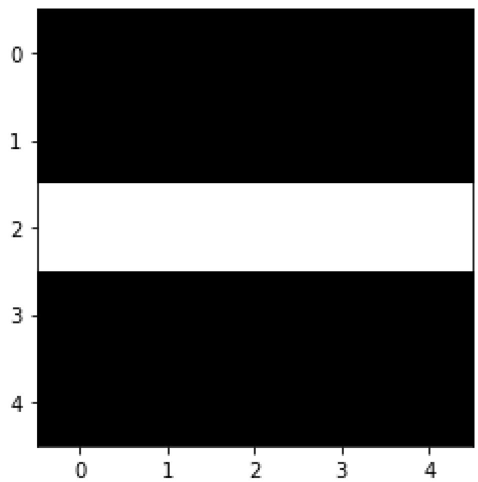


If you use a different image, the result will be different:

In [11]:

```
image1=torch.zeros(1,1,5,5)
image1[0,0,2,:]=1
print(image1)
plt.imshow(image1[0,0,:,:].detach().numpy(), interpolation='nearest', cmap=plt.cm.g
ray)
plt.show()
```

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

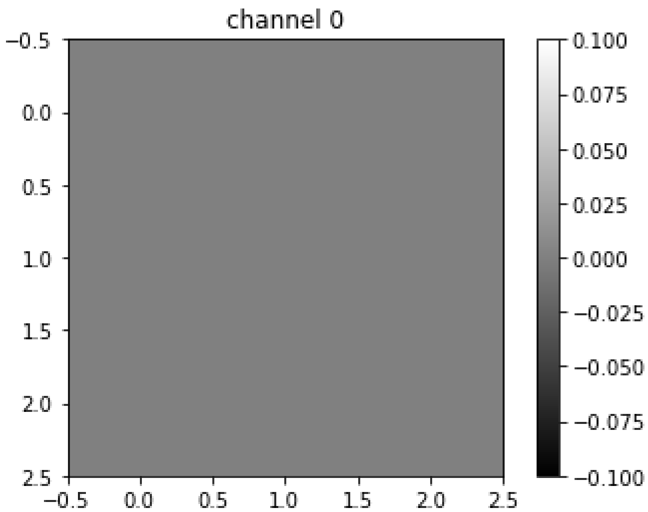


In this case, the second channel fluctuates, and the first and the third channels produce a constant value.

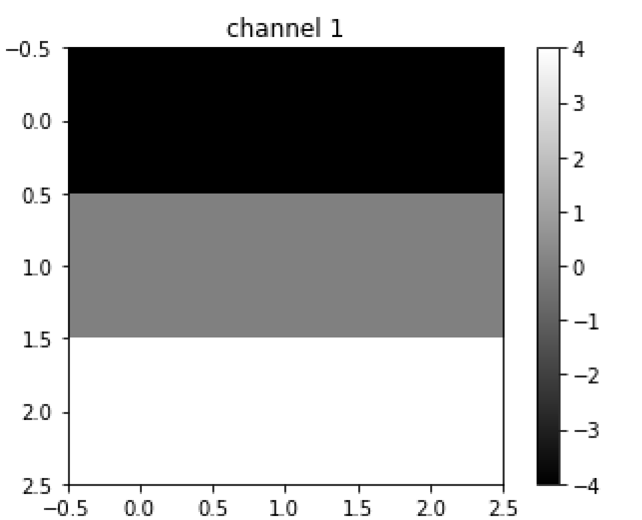
In [12]:

```
out1=conv1(image1)
for channel,image in enumerate(out1[0]):
    plt.imshow(image.detach().numpy(), interpolation='nearest', cmap=plt.cm.gray)
    print(image)
    plt.title("channel {}".format(channel))
    plt.colorbar()
    plt.show()
```

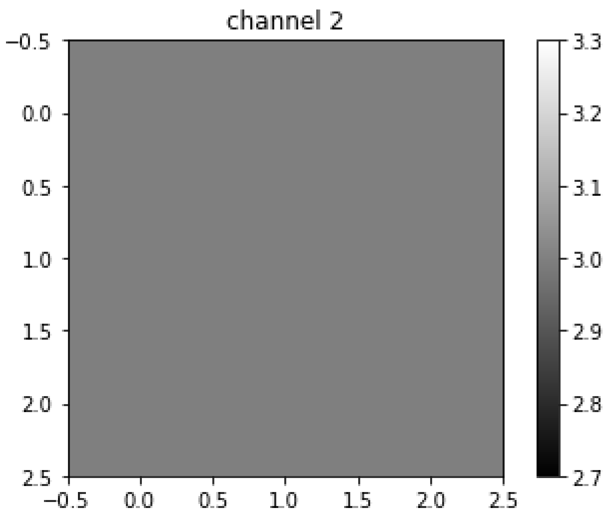
```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], grad_fn=<SelectBackward>)
```



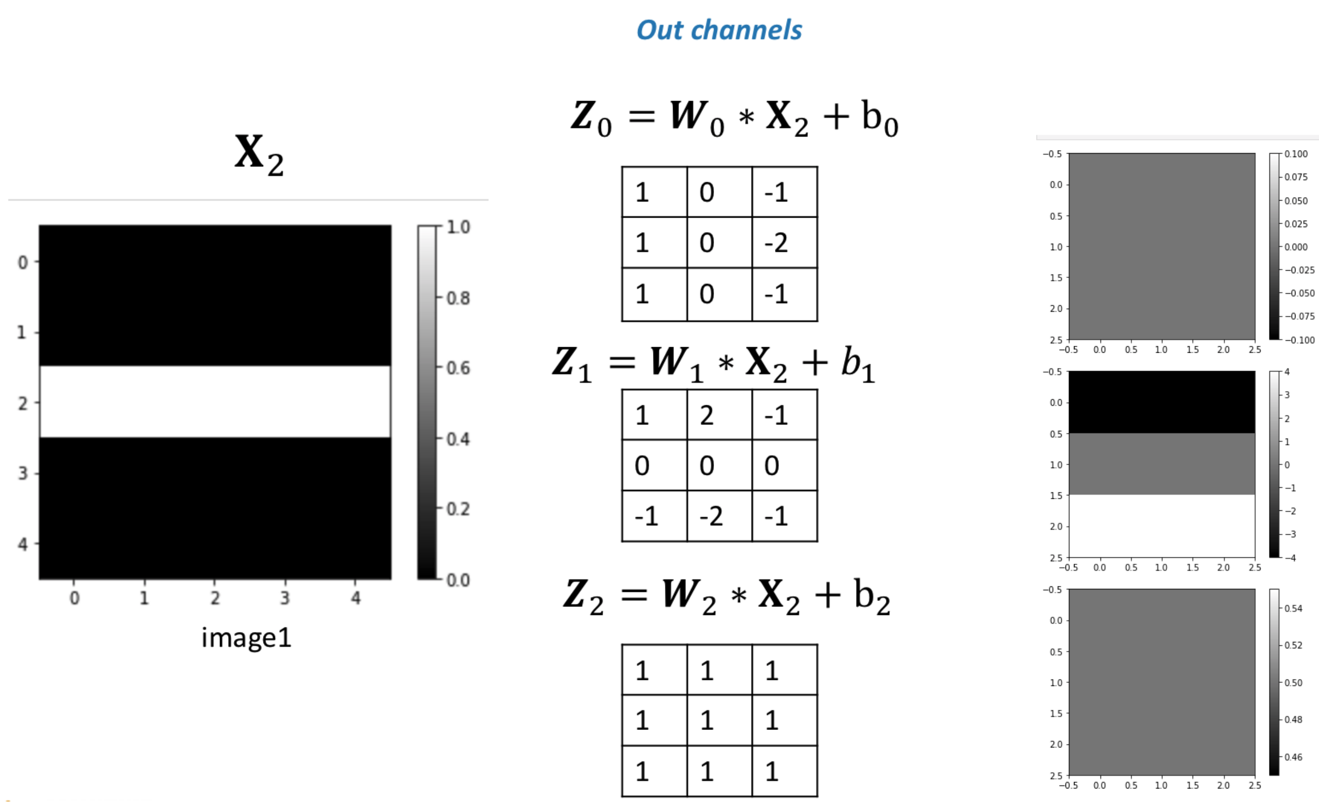

```
tensor([[-4., -4., -4.],
        [ 0.,  0.,  0.],
        [ 4.,  4.,  4.]], grad_fn=<SelectBackward>)
```




```
tensor([[3., 3., 3.],
        [3., 3., 3.],
        [3., 3., 3.]], grad_fn=<SelectBackward>)
```

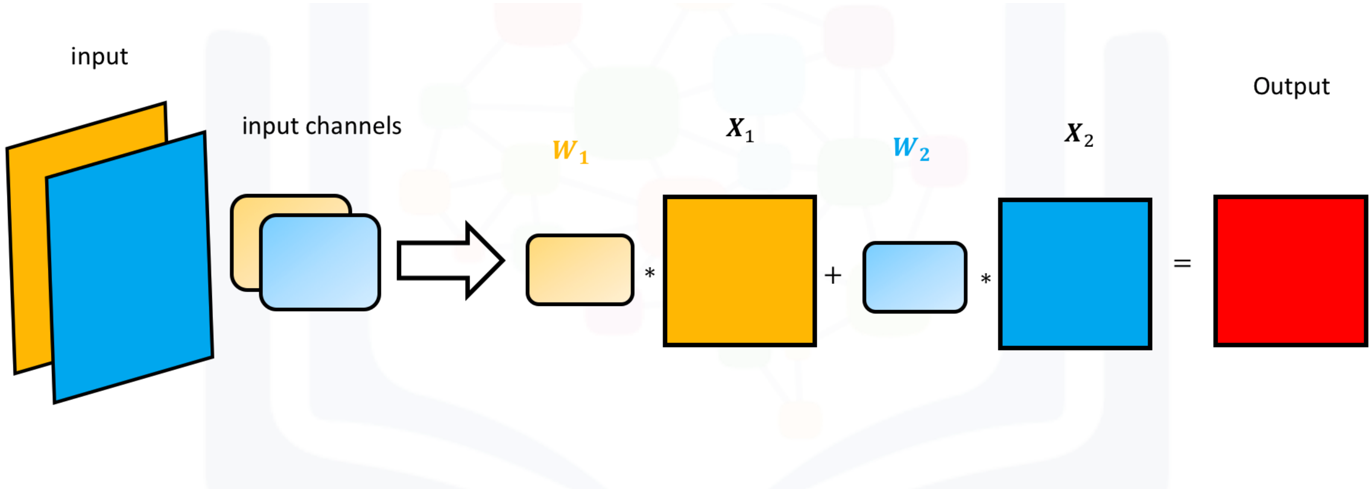


The following figure summarizes the process:



Multiple Input Channels

For two inputs, you can create two kernels. Each kernel performs a convolution on its associated input channel. The resulting output is added together as shown:



Create an input with two channels:

In [13]:

```
image2=torch.zeros(1,2,5,5)
image2[0,0,2,:]=-2
image2[0,1,2,:]=1
image2
```

Out[13]:

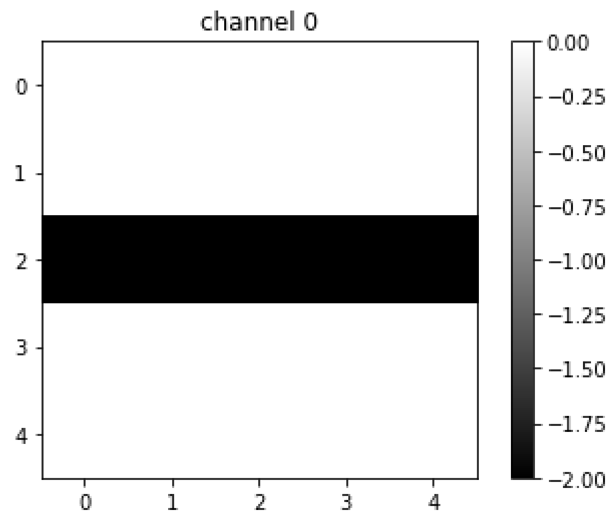
```
tensor([[[[ 0.,  0.,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  0.,  0.],
          [-2., -2., -2., -2., -2.],
          [ 0.,  0.,  0.,  0.,  0.],
          [ 0.,  0.,  0.,  0.,  0.]],

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

Plot out each image:

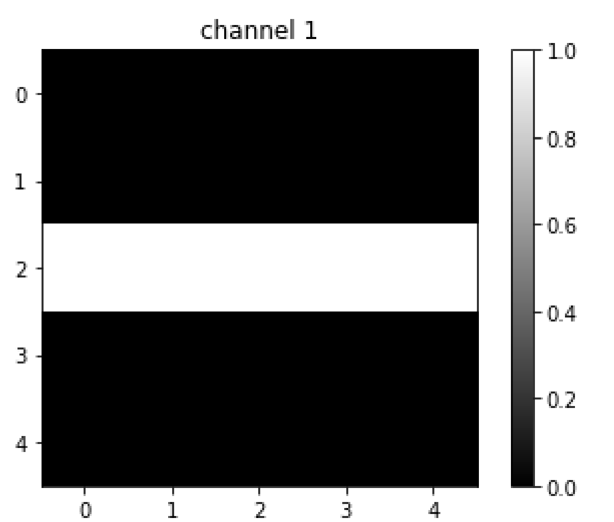
In [14]:

```
for channel,image in enumerate(image2[0]):
    plt.imshow(image.detach().numpy(), interpolation='nearest', cmap=plt.cm.gray)
    print(image)
    plt.title("channel {}".format(channel))
    plt.colorbar()
    plt.show()
```

```
tensor([[ 0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.],
        [-2., -2., -2., -2., -2.],
        [ 0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.]])
```




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



Create a `Conv2d` object with two inputs:

In [15]:

```
conv3 = nn.Conv2d(in_channels=2, out_channels=1,kernel_size=3)
```

Assign kernel values to make the math a little easier:

In [16]:

```
Gx1=torch.tensor([[0.0,0.0,0.0],[0,1.0,0],[0.0,0.0,0.0]])
conv3.state_dict()['weight'][0][0]=1*Gx1
conv3.state_dict()['weight'][0][1]=-2*Gx1
conv3.state_dict()['bias'][:]=torch.tensor([0.0])
```

In [17]:

```
conv3.state_dict()['weight']
```

Out[17]:

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

         [[-0., -0., -0.],
          [-0., -2., -0.],
          [-0., -0., -0.]]]])
```

Perform the convolution:

In [18]:

```
conv3(image2)
```

Out[18]:

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

The following images summarize the process. The object performs Convolution.

$$W_1 * X_1 \quad + \quad W_2 * X_2$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then, it adds the result:

$$\mathbf{Z} = W_1 * X_1 + W_2 * X_2$$

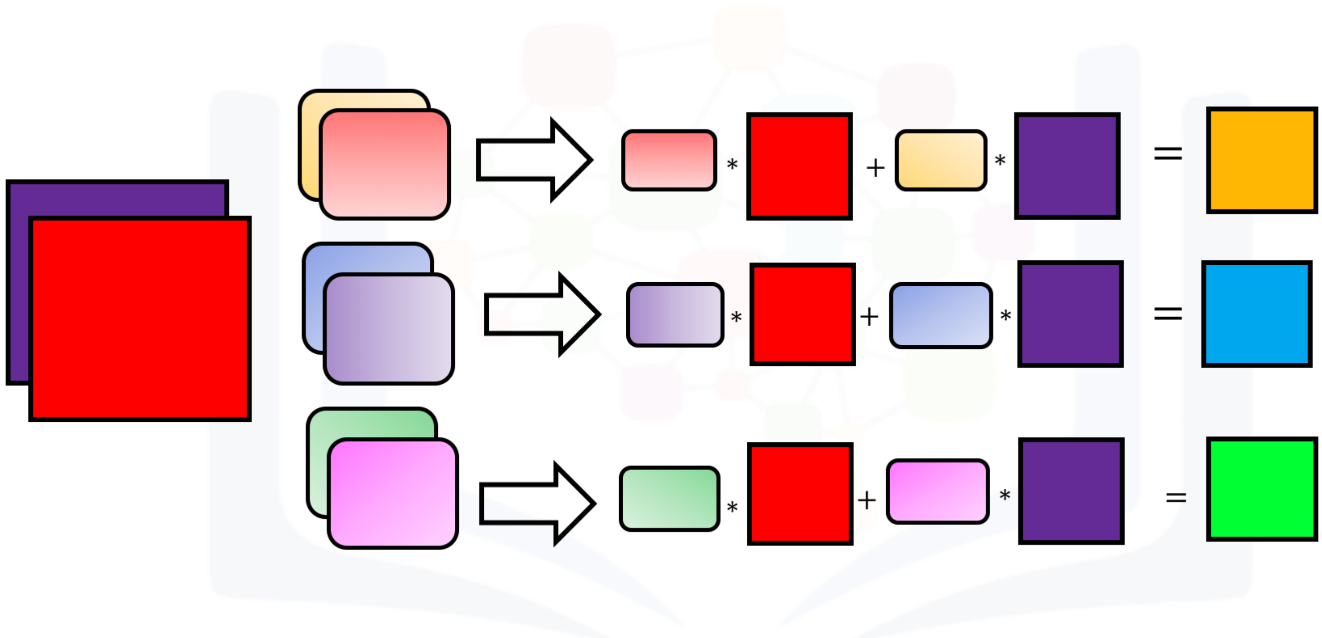
$$\begin{bmatrix} 0 & 0 & 0 \\ -2 & -2 & -2 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ -2 & -2 & -2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0+0 & 0+0 & 0+0 \\ -2-2 & -2-2 & -2-2 \\ 0+0 & 0+0 & 0+0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ -4 & -4 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

Multiple Input and Multiple Output Channels

When using multiple inputs and outputs, a kernel is created for each input, and the process is repeated for each output. The process is summarized in the following image.

There are two input channels and 3 output channels. For each channel, the input in red and purple is convolved with an individual kernel that is colored differently. As a result, there are three outputs.



Create an example with two inputs and three outputs and assign the kernel values to make the math a little easier:

In [19]:

```
conv4 = nn.Conv2d(in_channels=2, out_channels=3,kernel_size=3)
conv4.state_dict()['weight'][0][0]=torch.tensor([[0.0,0.0,0.0],[0,0.5,0],[0.0,0.0,0.0]])
conv4.state_dict()['weight'][0][1]=torch.tensor([[0.0,0.0,0.0],[0,0.5,0],[0.0,0.0,0.0]])


conv4.state_dict()['weight'][1][0]=torch.tensor([[0.0,0.0,0.0],[0,1,0],[0.0,0.0,0.0]])
conv4.state_dict()['weight'][1][1]=torch.tensor([[0.0,0.0,0.0],[0,-1,0],[0.0,0.0,0.0]])

conv4.state_dict()['weight'][2][0]=torch.tensor([[1.0,0,-1.0],[2.0,0,-2.0],[1.0,0.0,-1.0]])
conv4.state_dict()['weight'][2][1]=torch.tensor([[1.0,2.0,1.0],[0.0,0.0,0.0],[-1.0,-2.0,-1.0]])
```

For each output, there is a bias, so set them all to zero:

In [20]:

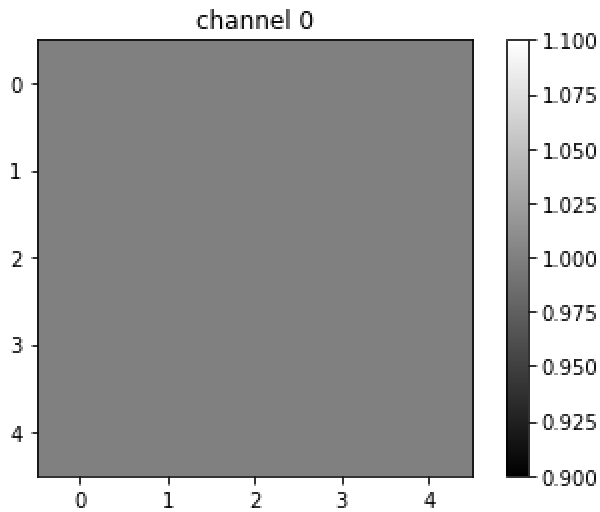
```
conv4.state_dict()['bias'][:]=torch.tensor([0.0,0.0,0.0])
```

Create a two-channel image and plot the results:

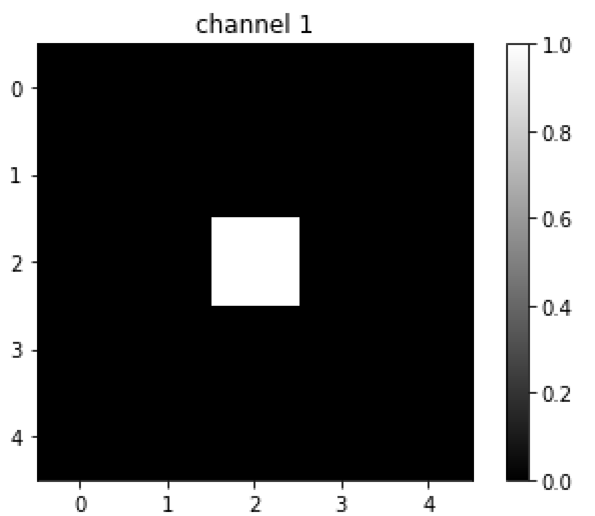
In [21]:

```
image4=torch.zeros(1,2,5,5)
image4[0][0]=torch.ones(5,5)
image4[0][1][2][2]=1
for channel,image in enumerate(image4[0]):
    plt.imshow(image.detach().numpy(), interpolation='nearest', cmap=plt.cm.gray)
    print(image)
    plt.title("channel {}".format(channel))
    plt.colorbar()
    plt.show()
```

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




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



Perform the convolution:

In [22]:

```
z=conv4(image4)
z
```

Out[22]:

```
tensor([[[[ 0.5000,  0.5000,  0.5000],
          [ 0.5000,  1.0000,  0.5000],
          [ 0.5000,  0.5000,  0.5000]],

         [[ 1.0000,  1.0000,  1.0000],
          [ 1.0000,  0.0000,  1.0000],
          [ 1.0000,  1.0000,  1.0000]],

         [[-1.0000, -2.0000, -1.0000],
          [ 0.0000,  0.0000,  0.0000],
          [ 1.0000,  2.0000,  1.0000]]]], grad_fn=<MkldnnConvolutionBac
kward>)
```

The output of the first channel is given by:

$$
\boldsymbol{Z}_0 = \underset{\boldsymbol{W}_{0,0}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \underset{\boldsymbol{X}_0}{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}} + \underset{\boldsymbol{W}_{0,1}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}} * \underset{\boldsymbol{X}_1}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}
$$

$$
= \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}
$$

The output of the second channel is given by:

$$
Z_1 = \underbrace{\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}}_{W_{1,0}} * \underbrace{\begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}}_{X_0} + \underbrace{\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}}_{W_{1,1}} * \underbrace{\begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}}_{X_1}
$$

$$
= \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} + \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 0 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}
$$

The output of the third channel is given by:

$$
Z_2 = \underbrace{\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}}_{W_{2,0}} * \underbrace{\begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}}_{X_0} + \underbrace{\begin{array}{|c|c|c|}\hline 1 & 2 & -1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}}_{W_{2,1}} * \underbrace{\begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}}_{X_1}
$$

$$
= \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}
$$

Practice Questions

Use the following two convolution objects to produce the same result as two input channel convolution on imageA and imageB as shown in the following image:

In [23]:

```
imageA=torch.zeros(1,1,5,5)
imageB=torch.zeros(1,1,5,5)
imageA[0,0,2,:]=-2
imageB[0,0,2,:]=1


conv5 = nn.Conv2d(in_channels=1, out_channels=1,kernel_size=3)
conv6 = nn.Conv2d(in_channels=1, out_channels=1,kernel_size=3)


Gx1=torch.tensor([[0.0,0.0,0.0],[0,1.0,0],[0.0,0.0,0.0]])
conv5.state_dict()['weight'][0][0]=1*Gx1
conv6.state_dict()['weight'][0][0]=-2*Gx1
conv5.state_dict()['bias'][:]=torch.tensor([0.0])
conv6.state_dict()['bias'][:]=torch.tensor([0.0])
```

$$\boldsymbol{W_1} * \boldsymbol{X_1} \quad + \quad \boldsymbol{W_2} * \boldsymbol{X_2}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{Z} = \boldsymbol{W_1} * \boldsymbol{X_1} + \boldsymbol{W_2} * \boldsymbol{X_2}$$

0	0	0
-2	-2	-2
0	0	0

+

0	0	0
-2	-2	-2
0	0	0

=

0+0	0+0	0+0
-2-2	-2-2	-2-2
0+0	0+0	0+0

=

0	0	0
-4	-4	-4
0	0	0

In [24]:

```
conv5(imageA)+conv6(imageB)
```

Out[24]:

```
tensor([[[[ 0.,  0.,  0.],
          [-4., -4., -4.],
          [ 0.,  0.,  0.]]]], grad_fn=<AddBackward0>)
```

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

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/)
