Multiple Input and Multiple Output Channels

Objetivos

Aprender sobre Multiple Input and Multiple Output Channels

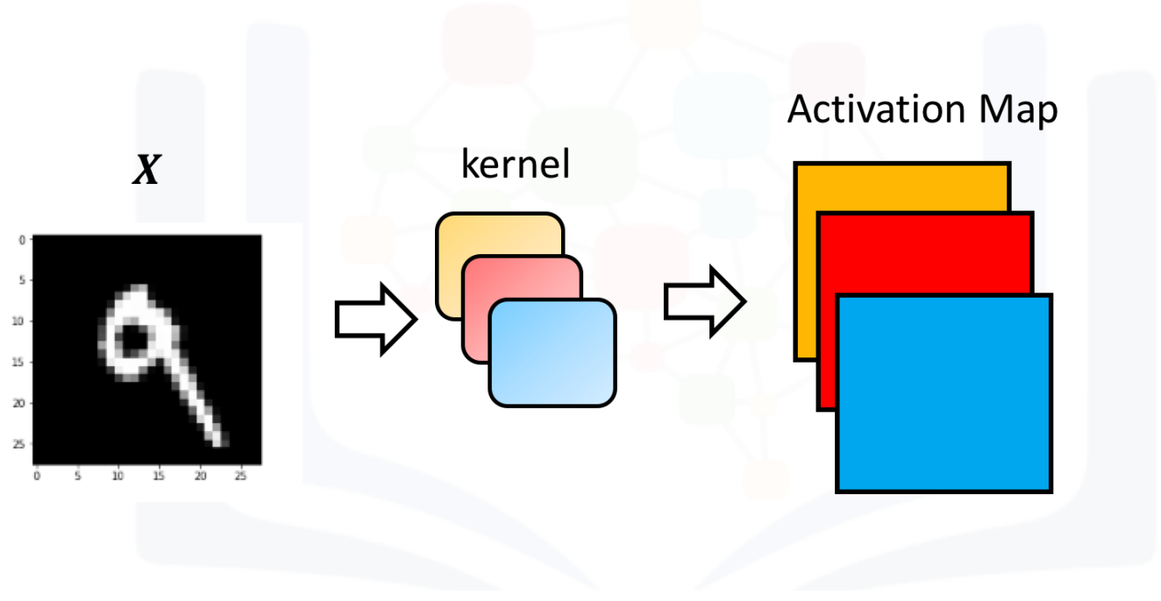
Tabla de contenido

In [1]:
```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

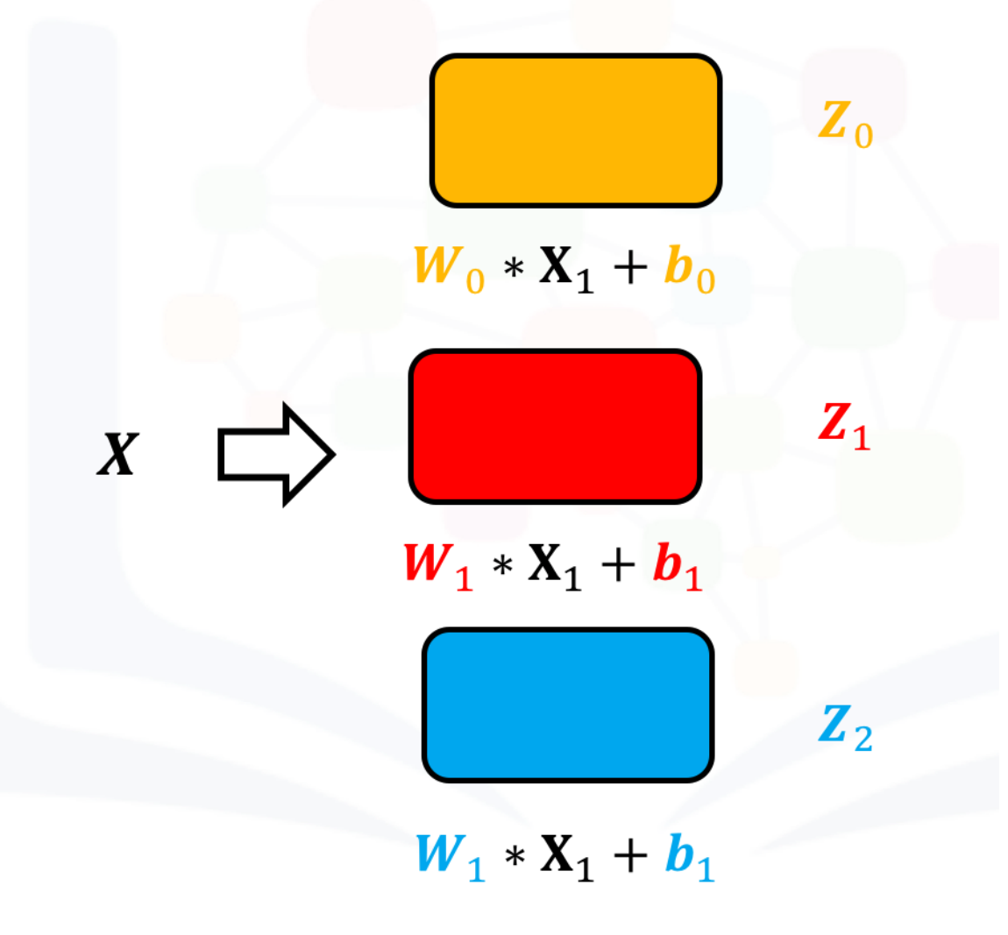
In [2]:
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np
from scipy import ndimage, misc
```

Múltiples canales de salida

En PyTorch puede crear un objeto `Conv2d` con múltiples salidas. Para cada canal, un kernel es creado, y cada kernel realiza una convolución de forma independiente. Como resultado, el número de salidas es igual al número de canales. Esto se muestra en la figura siguiente. El número 9 es convolucionado con 3 kernels: cada uno de diferente color. Hay 3 mapas de activación representados por diferentes colores.



Simbólicamente, se representa como sigue:



Creamos un objeto Conv2d con3 canales:

In [3]:
```
conv1 = nn.Conv2d(in_channels=1, out_channels=3,kernel_size=3)
```

PyTorch asigna valores aleatorios a cada kernel. Sin embargo, usaremos kernels que han sido desarrollados para detectar bordes:

In [4]:
```
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)
```

Cada kernel tiene su propio sesgo, los estableceremos en 0:

In [5]:
```
conv1.state_dict()['bias'][:]=torch.tensor([0.0,0.0,0.0])
conv1.state_dict()['bias']
```

Out[5]:
```
tensor([0., 0., 0.])
```

Imprimimos cada kernel:

In [6]:
```
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.]]])
```

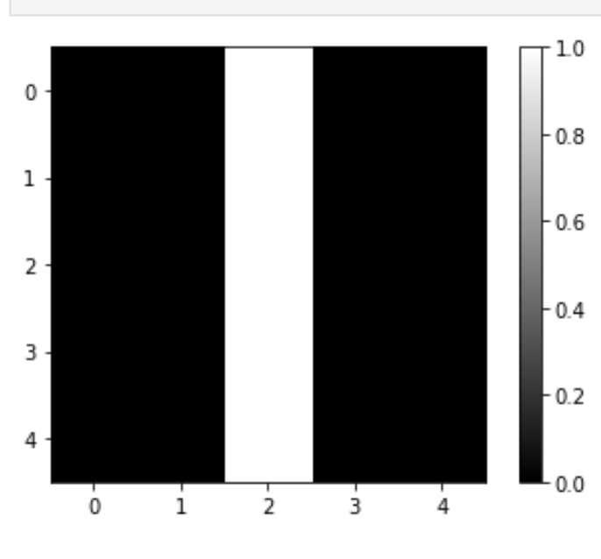
Creamos una entrada image para representar la entrada X:

In [7]:
```
image=torch.zeros(1,1,5,5)
image[0,0,:,2]=1
image
```

Out[7]:
```
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.]]]])
```

Lo graficamos como imagen:

In [8]:
```
plt.imshow(image[0,0,:,:].numpy(), interpolation='nearest', cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```



Realizamos convolución usando cada canal:

In [9]:
```
out=conv1(image)
```

El resultado es un tensor 1x3x3x3. Esto representa una muestra con 3 canales, y cada canal contiene una imagen 3x3.

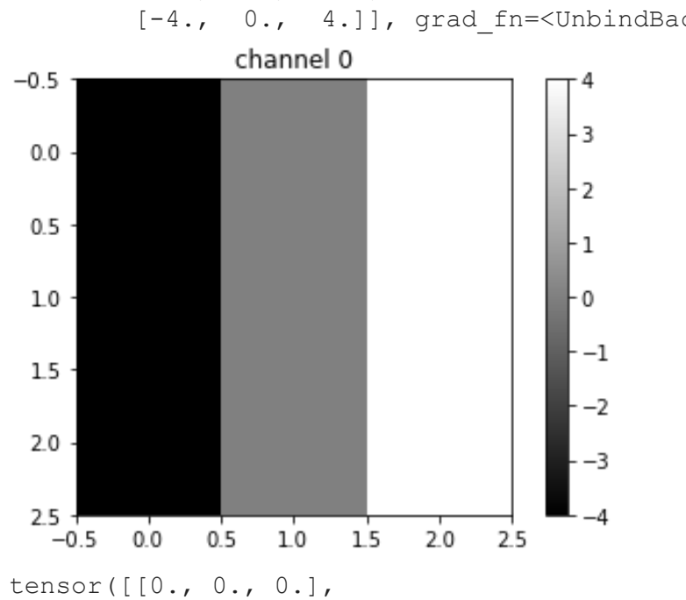
In [10]:
```
out.shape
```

Out[10]:
```
torch.Size([1, 3, 3, 3])
```

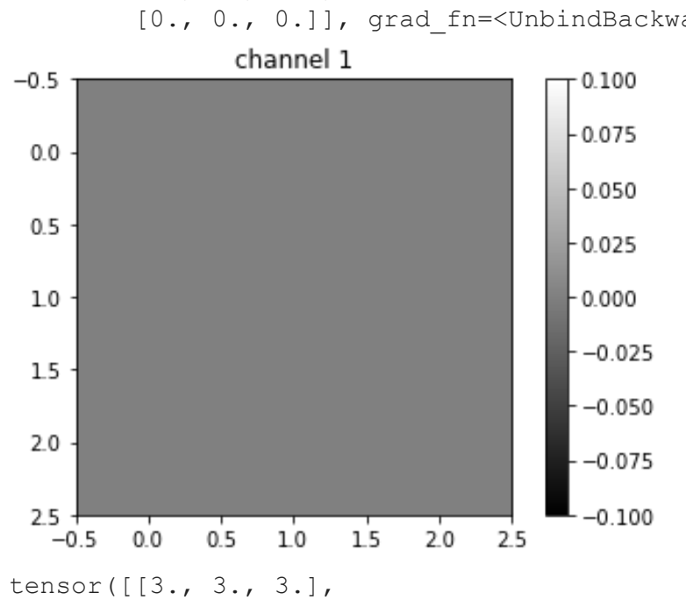
Imprimimos cada canal como tensor o imagen:

In [11]:
```
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()
```

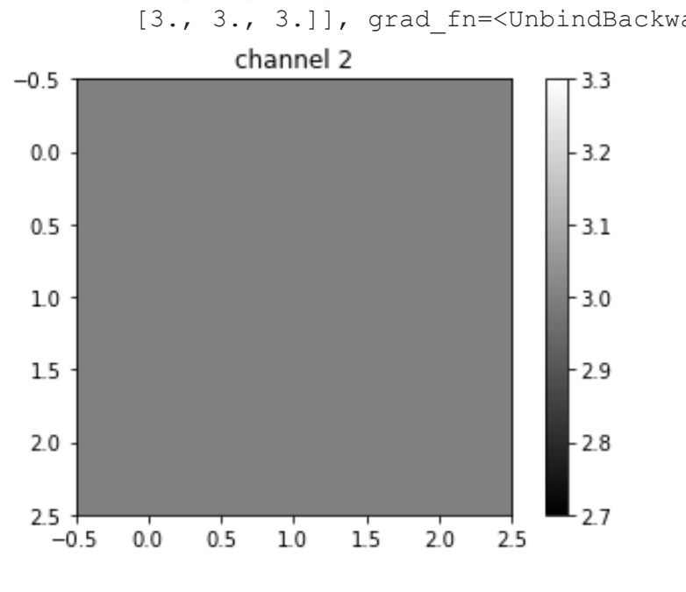
```
tensor([[-4.,  0.,  4.],
        [-4.,  0.,  4.],
        [-4.,  0.,  4.]], grad_fn=<UnbindBackward>)
```



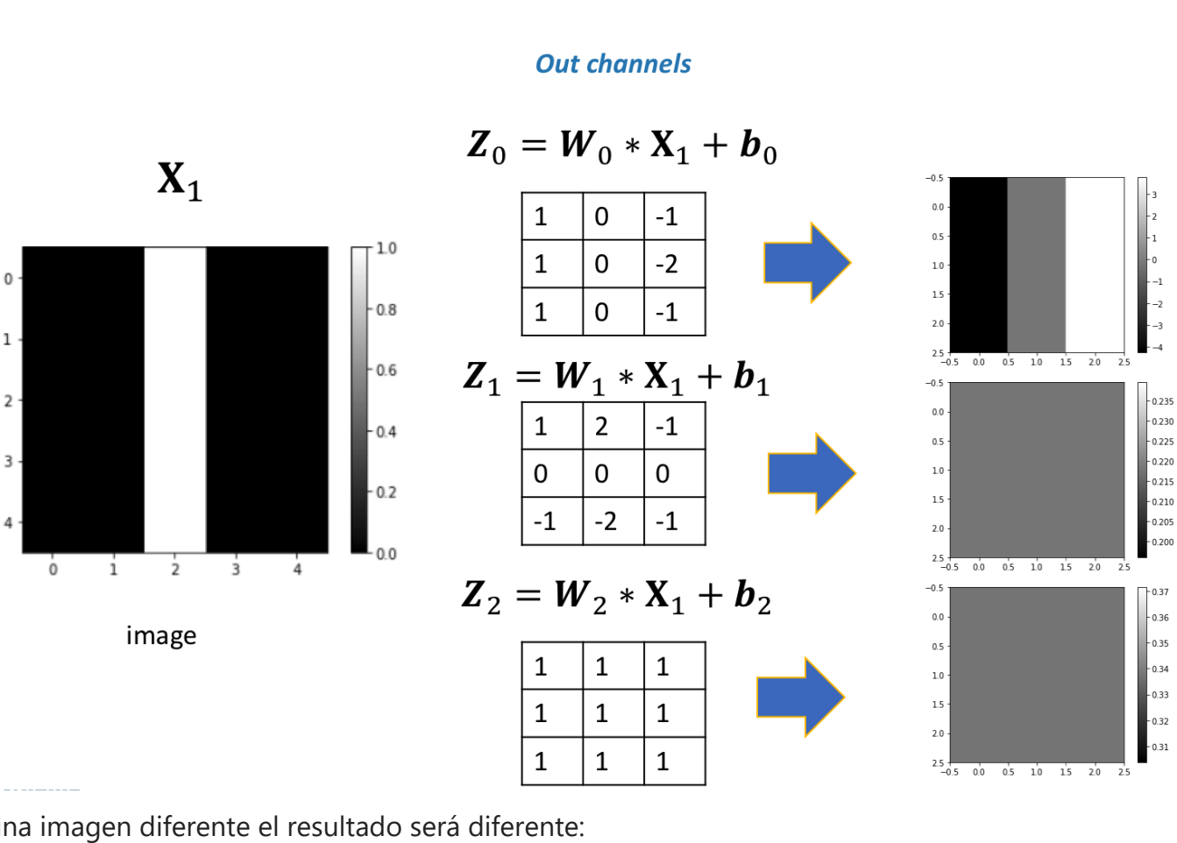
```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], grad_fn=<UnbindBackward>)
```



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



Diferentes kernels pueden utilizarse para detectar varias características en una imagen. Puede verse que el primer canal fluctúa y el segundo y tercero producen un valor constante. La siguiente figura ilustra el proceso:



$$Z_0 = W_0 * X_1 + b_0 \quad \begin{bmatrix}1&0&-1\\1&0&-2\\1&0&-1\end{bmatrix}$$

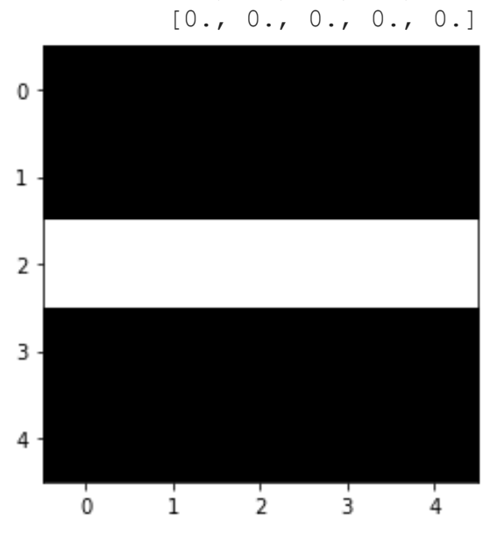
$$Z_1 = W_1 * X_1 + b_1 \quad \begin{bmatrix}1&2&-1\\0&0&0\\-1&-2&-1\end{bmatrix}$$

$$Z_2 = W_2 * X_1 + b_2 \quad \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

Si utiliza una imagen diferente el resultado será diferente:

In [12]:
```
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.gray
plt.show()
```

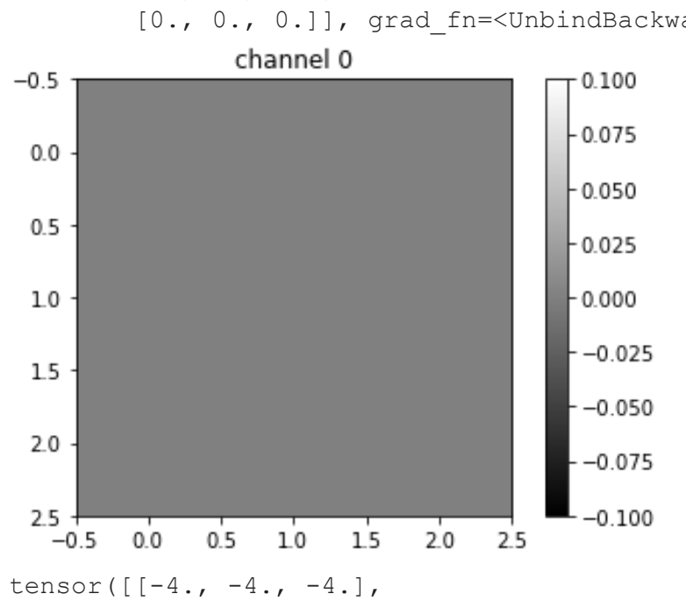
```
tensor([[[[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [1., 1., 1., 1., 1.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]]]])
```



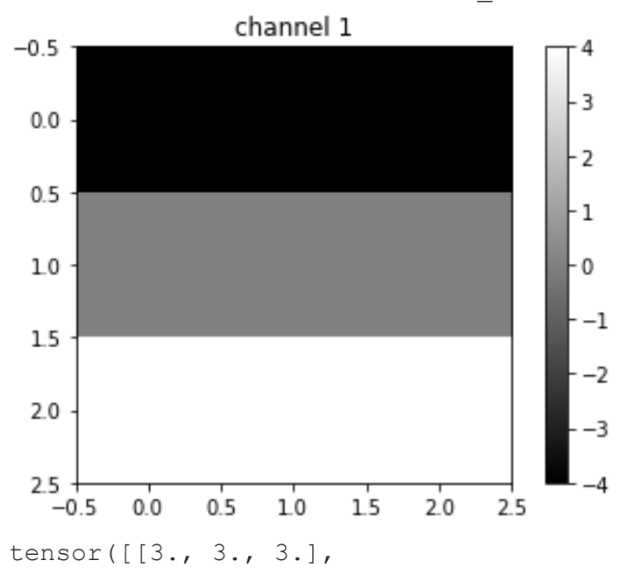
En este caso el segundo canal fluctúa y el primero y tercero producen un valor constante.

In [13]:
```
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()
```

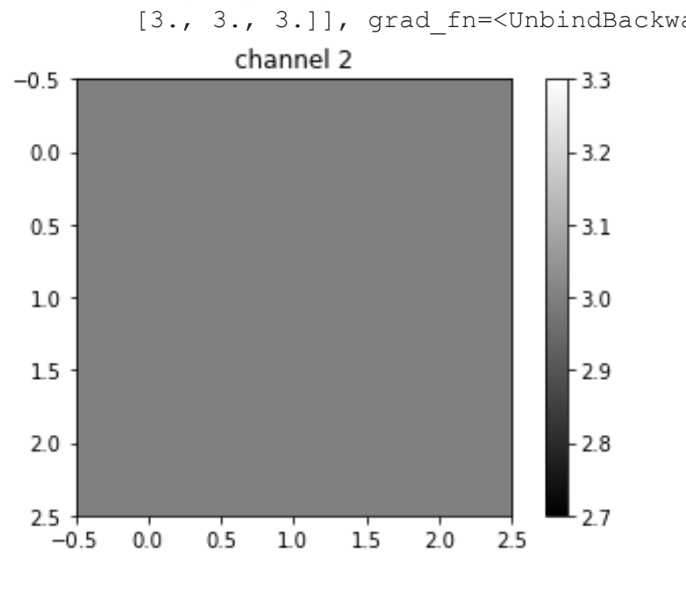
```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], grad_fn=<UnbindBackward>)
```



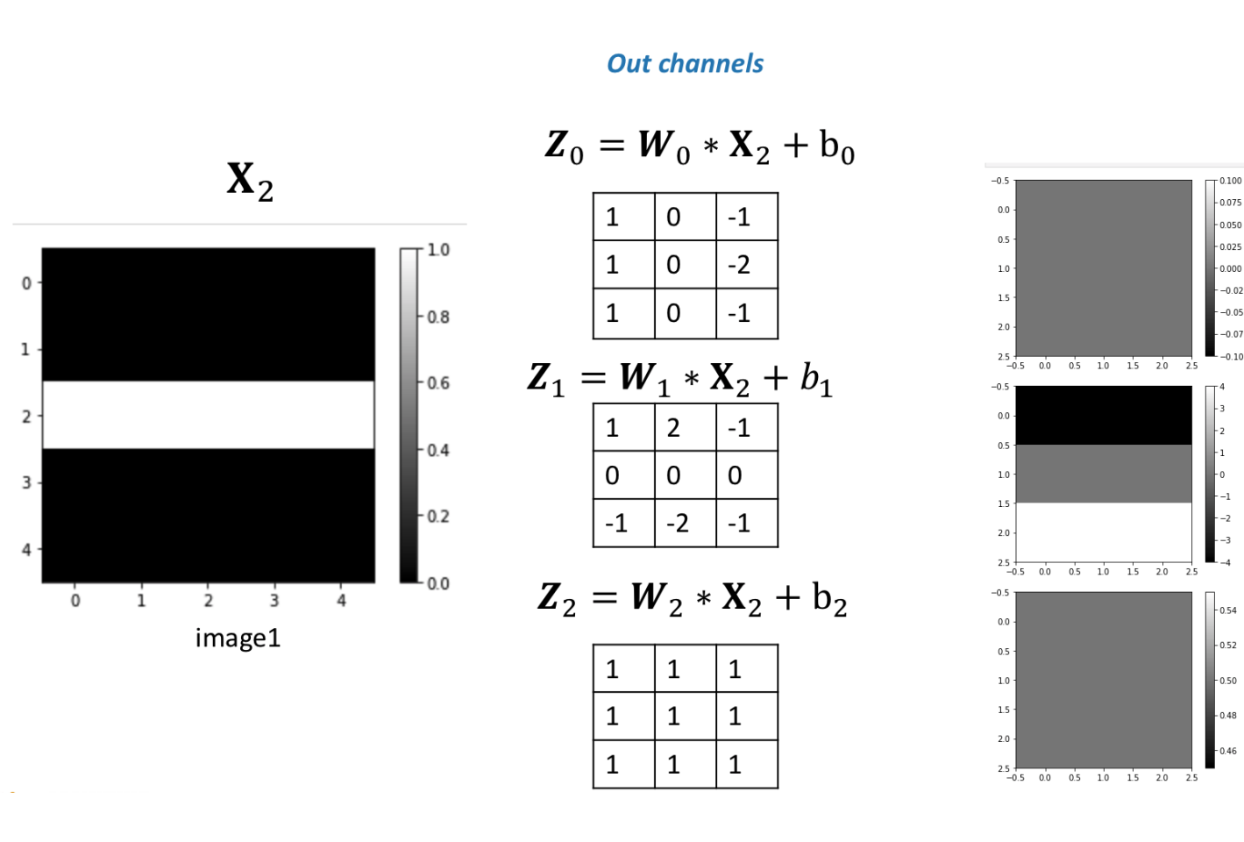
```
tensor([[-4., -4., -4.],
        [ 0.,  0.,  0.],
        [ 4.,  4.,  4.]], grad_fn=<UnbindBackward>)
```



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



La siguiente figura ilustra el proceso:



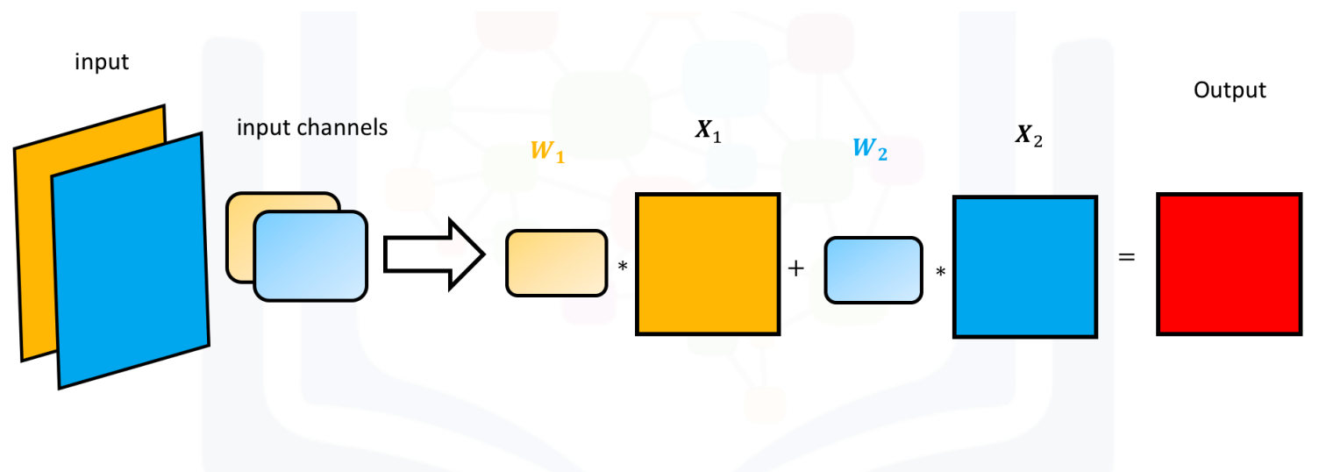
$$Z_0 = W_0 * X_2 + b_0 \quad \begin{bmatrix}1&0&-1\\1&0&-2\\1&0&-1\end{bmatrix}$$

$$Z_1 = W_1 * X_2 + b_1 \quad \begin{bmatrix}1&2&-1\\0&0&0\\-1&-2&-1\end{bmatrix}$$

$$Z_2 = W_2 * X_2 + b_2 \quad \begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

Múltiples canales de entrada

Para 2 entradas puede crear 2 kernels. Cada kernel realiza una convolución con su canal de entrada asociado. Luego se suma:



Creamos una imagen con 2 canales:

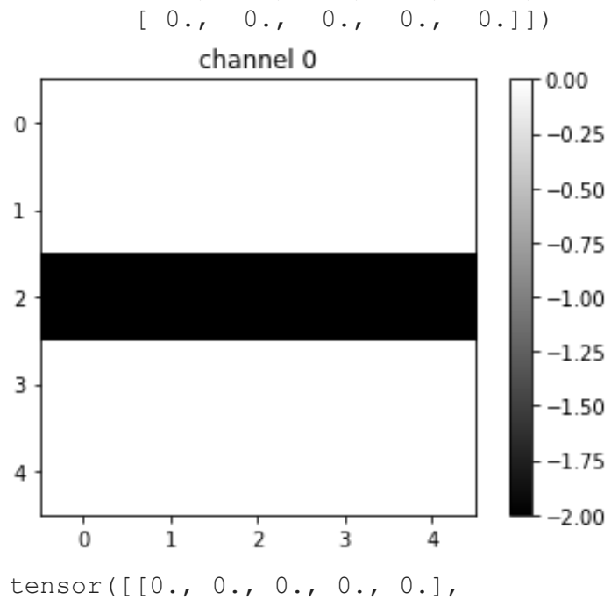
In [14]:
```
image2=torch.zeros(1,2,5,5)
image2[0,0,2,:]=-2
image2[0,1,2,:]=1
image2
```

Out[14]:
```
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.]]]])
```

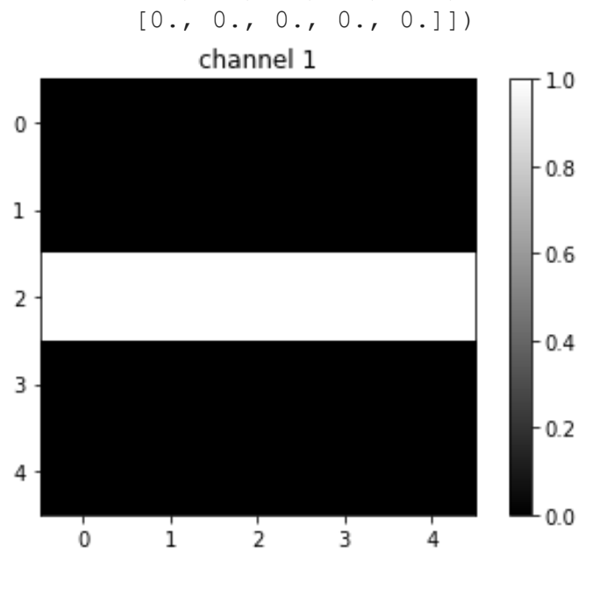
Graficamos cada imagen:

In [15]:
```
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.]])
```



Creamos un objeto Conv2d con 2 entradas:

In [16]:
```
conv3 = nn.Conv2d(in_channels=2, out_channels=1,kernel_size=3)
```

Asignamos valores de kernel para simplificar la matemática:

In [17]:
```
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 [18]:
```
conv3.state_dict()['weight']
```

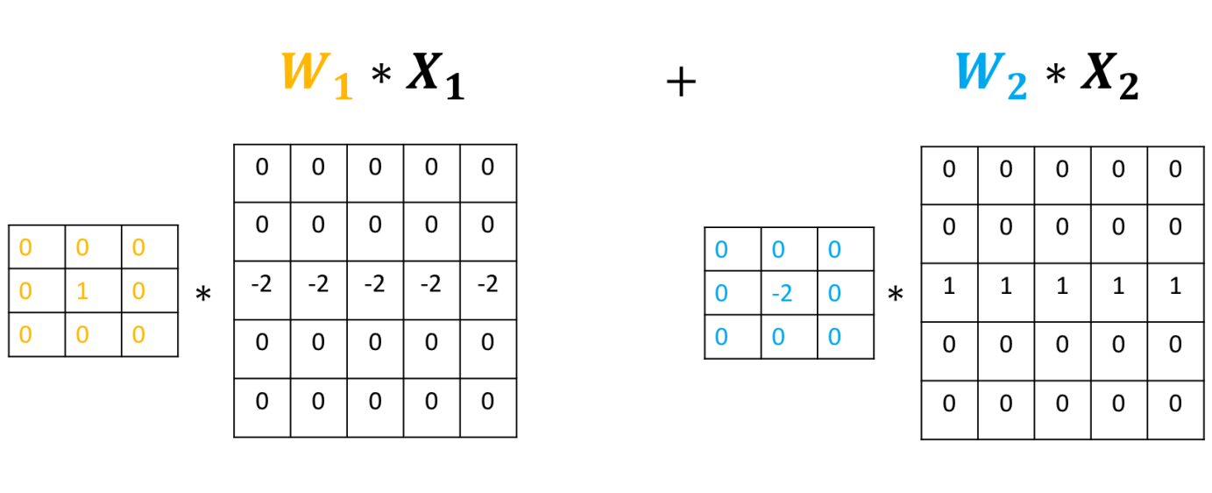
Out[18]:
```
tensor([[[[ 0.,  0.,  0.],
          [ 0.,  1.,  0.],
          [ 0.,  0.,  0.]],

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

Realizamos la convolución:

In [19]:
```
conv3(image2)
```

Out[19]:
```
tensor([[[[ 0.,  0.,  0.],
          [-4., -4., -4.],
          [ 0.,  0.,  0.]]]], grad_fn=<ThnnConv2DBackward>)
```

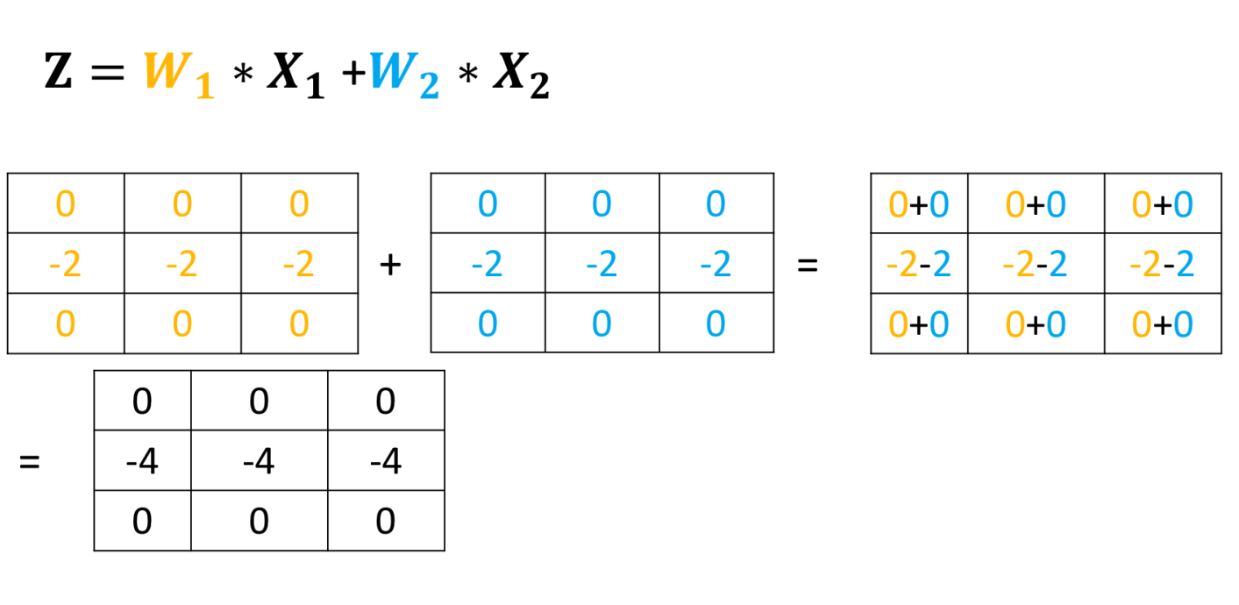
La siguiente imagen resume el proceso:



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

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

Luego, se suman los resultados:



$$Z = W_1 * X_1 + W_2 * X_2$$

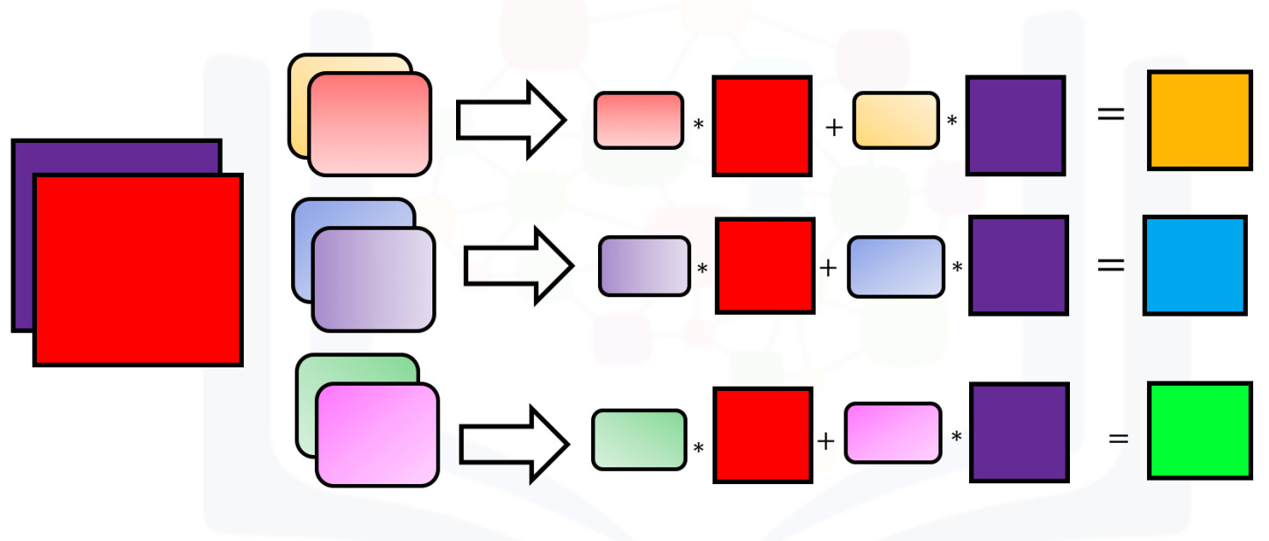
$$\begin{bmatrix}0&0&0\\-2&-2&-2\\0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0\\-2&-2&-2\\0&0&0\end{bmatrix} = \begin{bmatrix}0+0&0+0&0+0\\-2-2&-2-2&-2-2\\0+0&0+0&0+0\end{bmatrix}$$

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

Múltiples canales de entrada y múltiples canales de salida

Cuando usamos múltiples entradas y salidas, un kernel es creado para cada entrada, y el proceso es repetido para cada salida. El proceso se resume en la siguiente imagen.

Hay 2 canales de entrada y 3 canales de salida. Para cada canal, la entrada en rojo y violeta es convolucionada con un kernel individual que es coloreado de forma diferente. Como resultado, hay 3 salidas.



Creamos un ejemplo con 2 entradas y 3 salidas y asignamos valores a los kernels:

In [20]:
```
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,-
conv4.state_dict()['weight'][2][1]=torch.tensor([[1.0,2.0,1.0],[0.0,0.0,0.0],[-1.0,-2
```

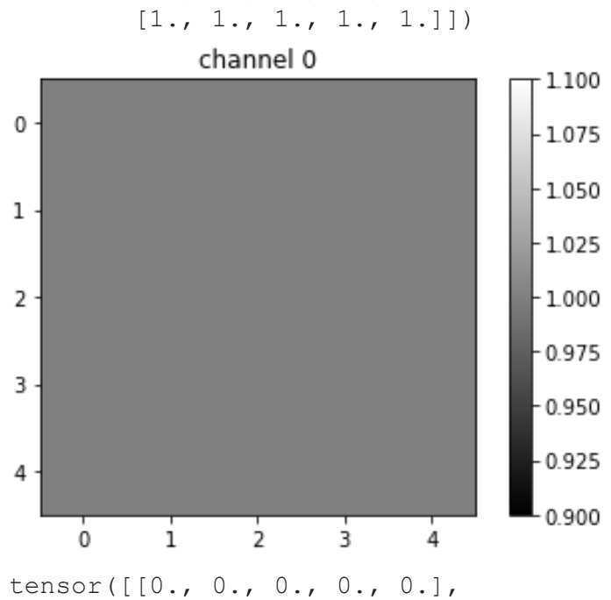
Para cada salida hay un sesgo, los establecemos en 0:

In [21]:
```
conv4.state_dict()['bias'][:]=torch.tensor([0.0,0.0,0.0])
```

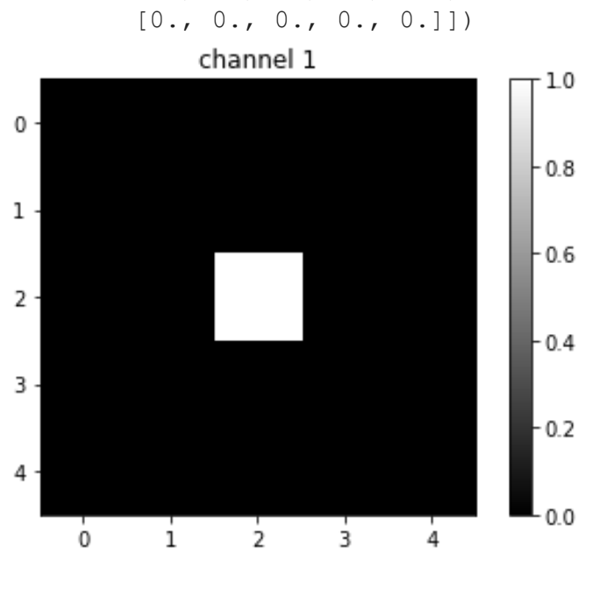
Creamos una imagen de 2 canales y graficamos:

In [22]:
```
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.]])
```

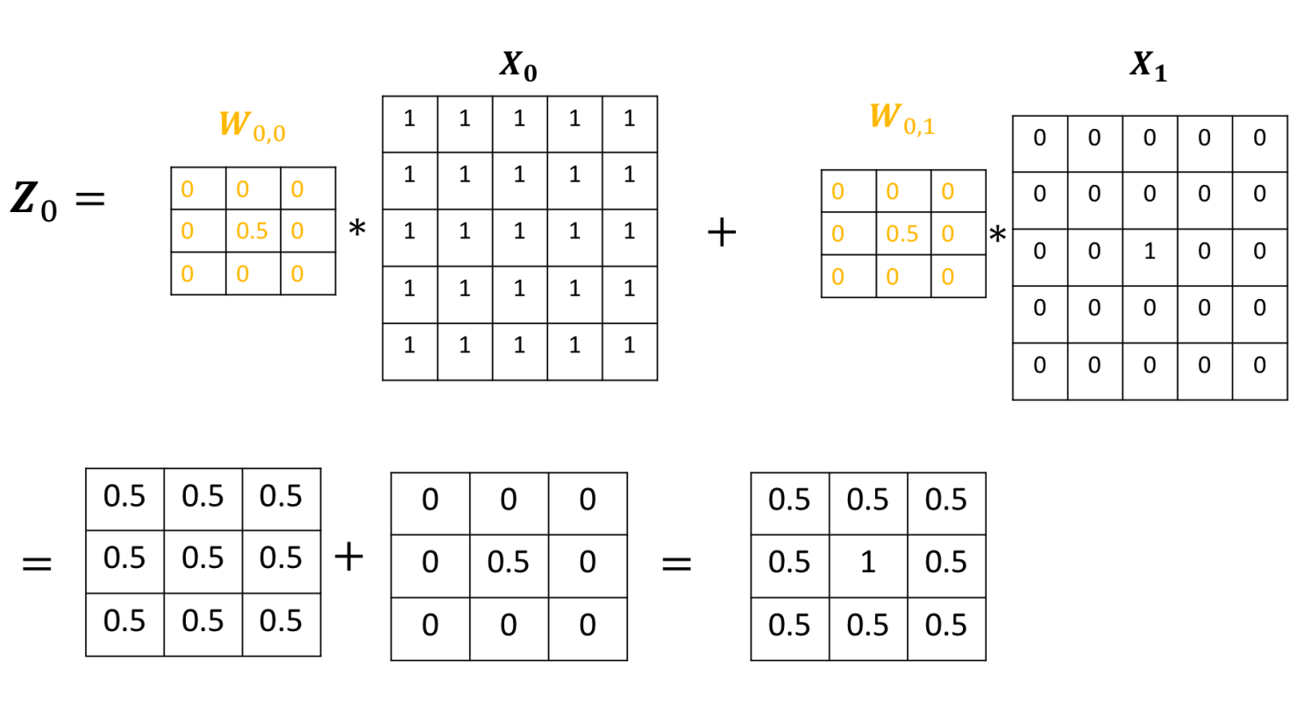


Realizamos la convolución:

In [23]:
```
z=conv4(image4)
z
```

Out[23]:
```
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=<ThnnConv2DBackward>)
```

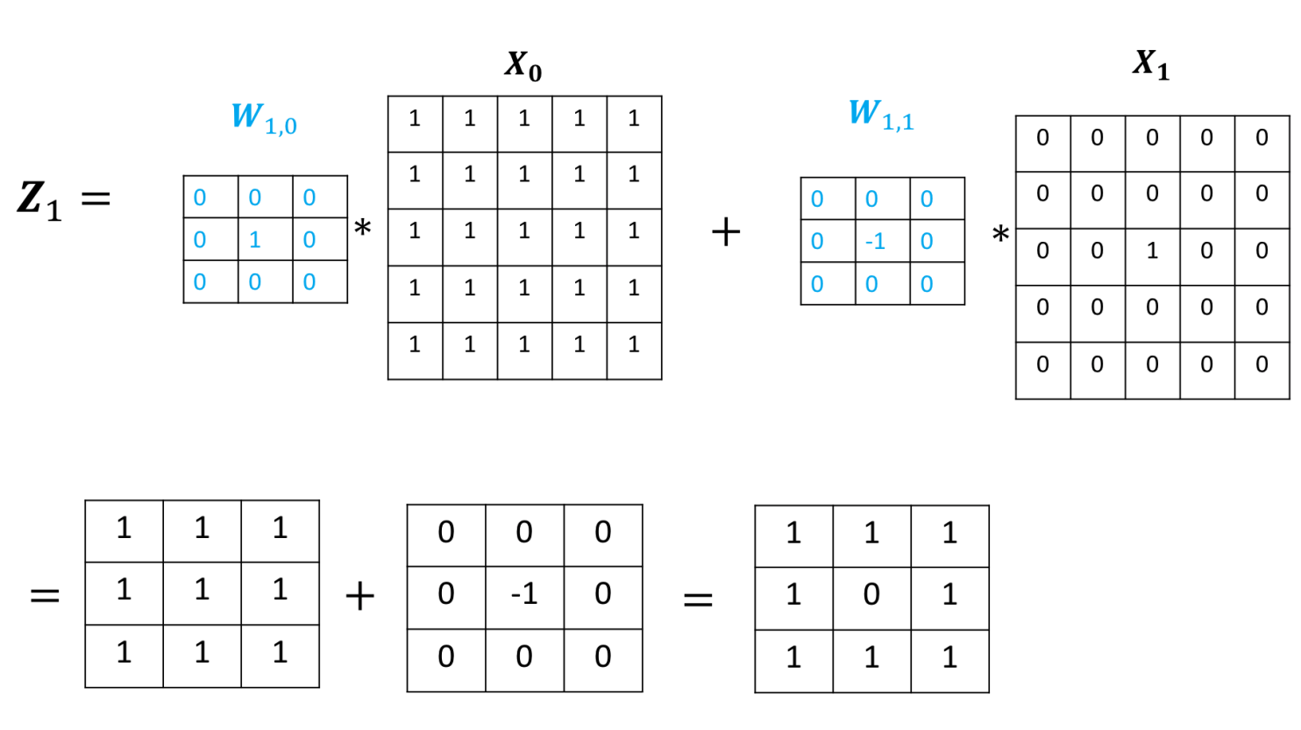
La salida del primer canal está dada por:



$$Z_0 = \overset{W_{0,0}}{\begin{bmatrix}0&0&0\\0&0.5&0\\0&0&0\end{bmatrix}} * \overset{X_0}{\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}} + \overset{W_{0,1}}{\begin{bmatrix}0&0&0\\0&0.5&0\\0&0&0\end{bmatrix}} * \overset{X_1}{\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}}$$

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

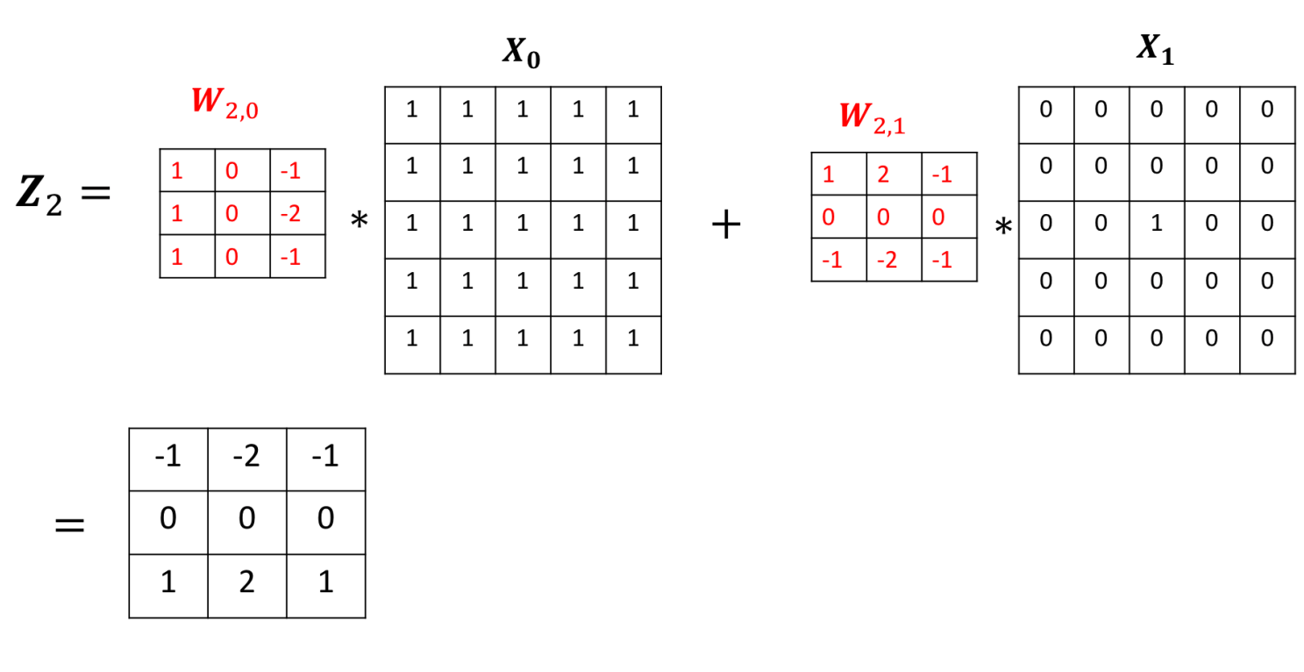
La salida del segundo canal está dada por:



$$Z_1 = \overset{W_{1,0}}{\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}} * \overset{X_0}{\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}} + \overset{W_{1,1}}{\begin{bmatrix}0&0&0\\0&-1&0\\0&0&0\end{bmatrix}} * \overset{X_1}{\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}}$$

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

La salida del tercer canal está dada por:



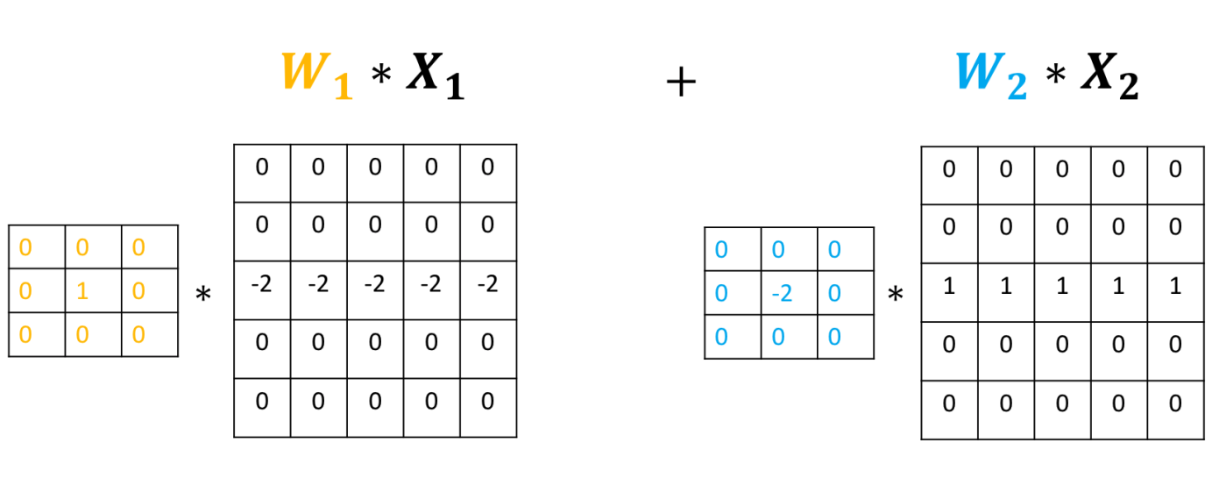
$$Z_2 = \overset{W_{2,0}}{\begin{bmatrix}1&0&-1\\1&0&-2\\1&0&-1\end{bmatrix}} * \overset{X_0}{\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}} + \overset{W_{2,1}}{\begin{bmatrix}1&2&-1\\0&0&0\\-1&-2&-1\end{bmatrix}} * \overset{X_1}{\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}}$$

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

Pregunas prácticas

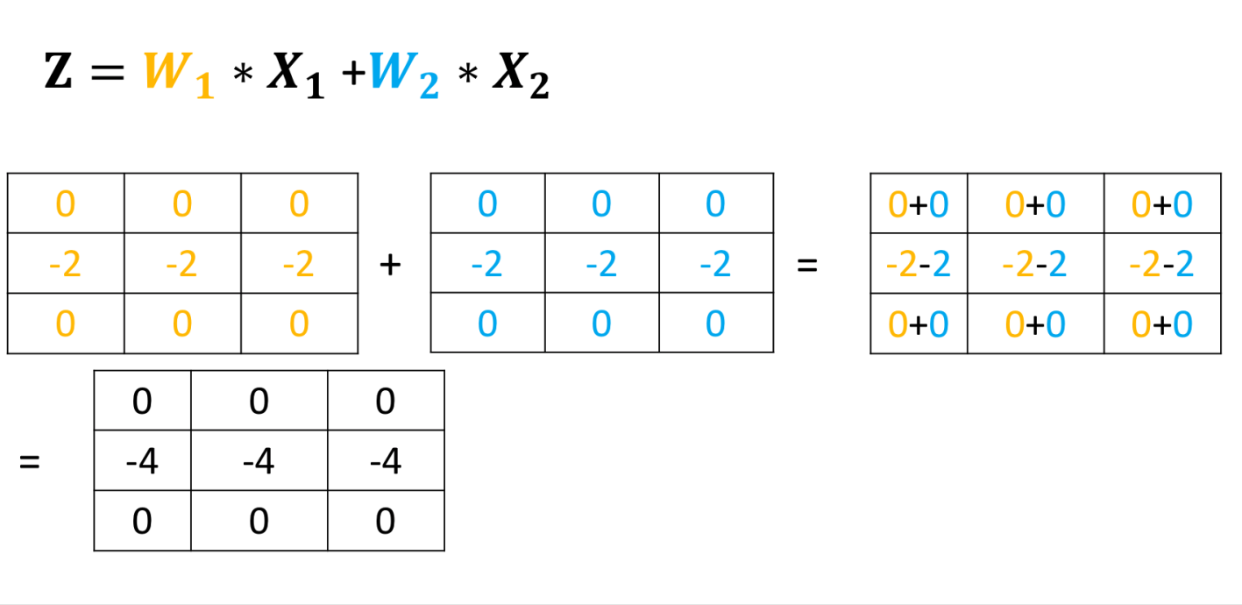
Utilice los siguientes dos objetos de convolución para producir el mismo resultado que la convolución de dos canales de entrada en imageA e imageB como se muestra en la siguiente imagen:

In [24]:
```
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])
```



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

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



$$Z = W_1 * X_1 + W_2 * X_2$$

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

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

In [25]:
```
conv5(imageA)+conv6(imageB)
```

Out[25]:
```
tensor([[[[ 0.,  0.,  0.],
          [-4., -4., -4.],
          [ 0.,  0.,  0.]]]], grad_fn=<AddBackward0>)
```