Función de activación y max pooling </h1 >

Objetivos

1. Aprender cómo aplicar una función de activación.

2. Aprender acerca del max pooling

Tabla de contenido

Veremos 2 componentes esenciales en la construcción de una red neuronal convolucional. El primero es la función de activación, que es análogo al caso de construir una red regular. El segundo es el max pooling, que reduce el número de parámetros y hace a la red menos susceptible a cambios en la imagen.

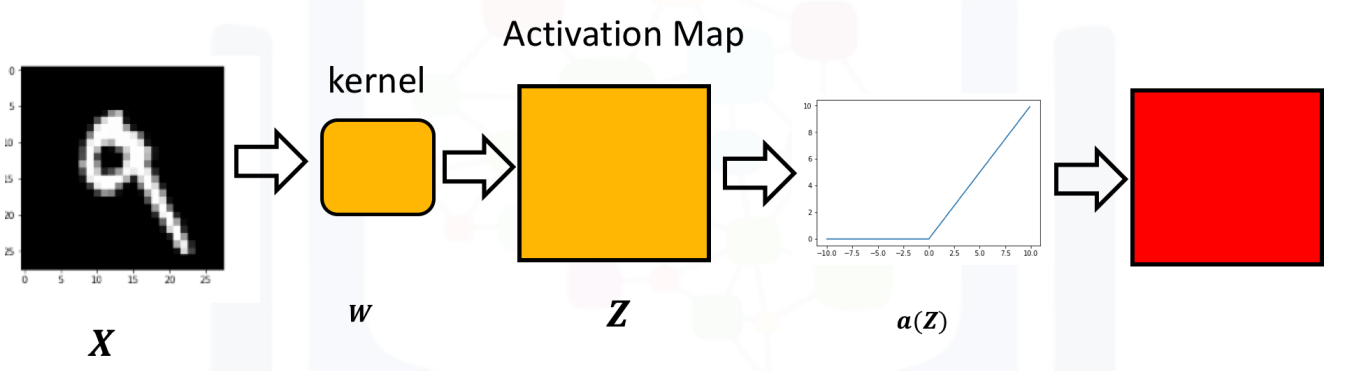
- Funciones de activación
- Max Pooling

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
```

Funciones de activación

Tal cual una red estándar, puede aplicarse una función de activación al mapa de activación como se muestra en la figura siguiente:



Creamos un kernel y una imagen. Establecemos el sesgo en 0:

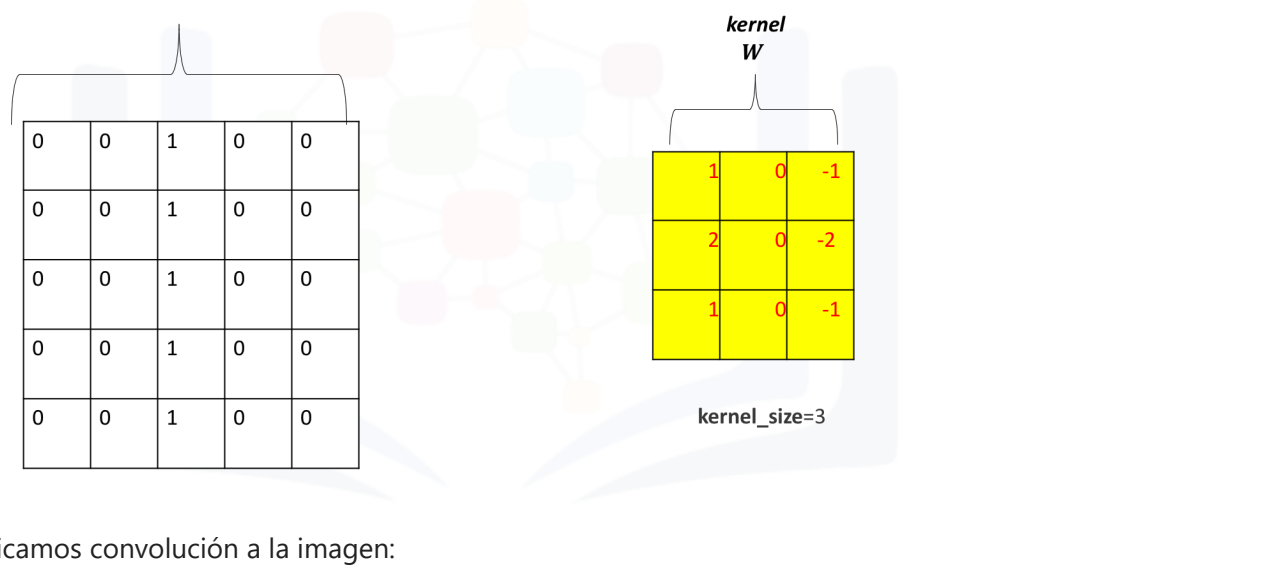
In [3]:
```
conv = nn.Conv2d(in_channels=1, out_channels=1,kernel_size=3)
Gx=torch.tensor([[1.0,0,-1.0],[2.0,0,-2.0],[1.0,0,-1.0]])
conv.state_dict()['weight'][0][0]=Gx
conv.state_dict()['bias'][0]=0.0
conv.state_dict()
```

Out[3]:
```
OrderedDict([('weight',
              tensor([[[[ 1.,  0., -1.],
                        [ 2.,  0., -2.],
                        [ 1.,  0., -1.]]]])),
             ('bias', tensor([0.]))])
```

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.]]]])
```

Vemos la imagen y el kernel:



Le aplicamos convolución a la imagen:

In [5]:
```
Z=conv(image)
Z
```

Out[5]:
```
tensor([[[[-4.,  0.,  4.],
          [-4.,  0.,  4.],
          [-4.,  0.,  4.]]]], grad_fn=<ThnnConv2DBackward>)
```

Le aplicamos la función de activación al mapa de activación. Esto le aplicará la función de activación a cada elemento en el mapa de activación.

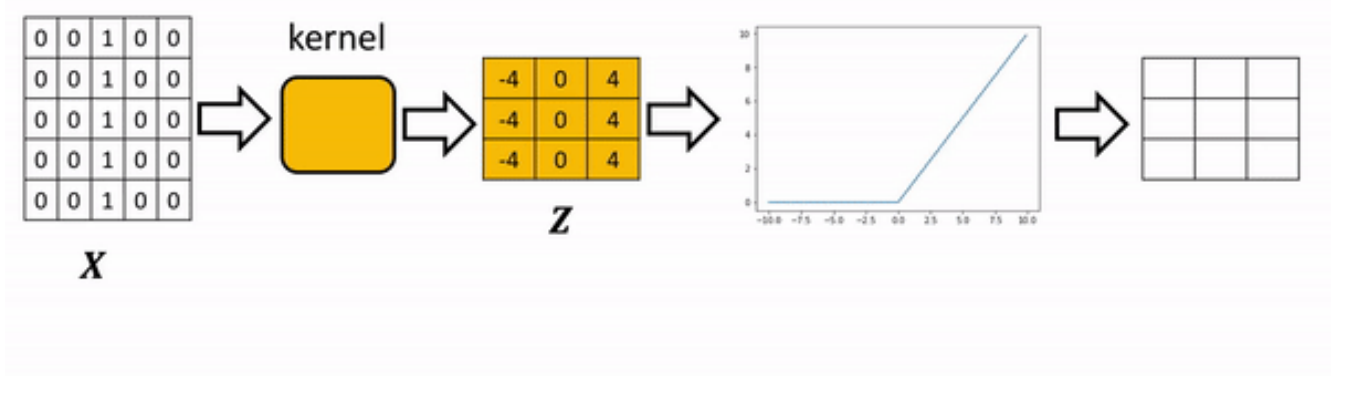
In [6]:
```
A=torch.relu(Z)
A
```

Out[6]:
```
tensor([[[[0., 0., 4.],
          [0., 0., 4.],
          [0., 0., 4.]]]], grad_fn=<ReluBackward0>)
```

In [7]:
```
relu = nn.ReLU()
relu(Z)
```

Out[7]:
```
tensor([[[[0., 0., 4.],
          [0., 0., 4.],
          [0., 0., 4.]]]], grad_fn=<ReluBackward0>)
```

El proceso se resume en la figura siguiente:



Max Pooling

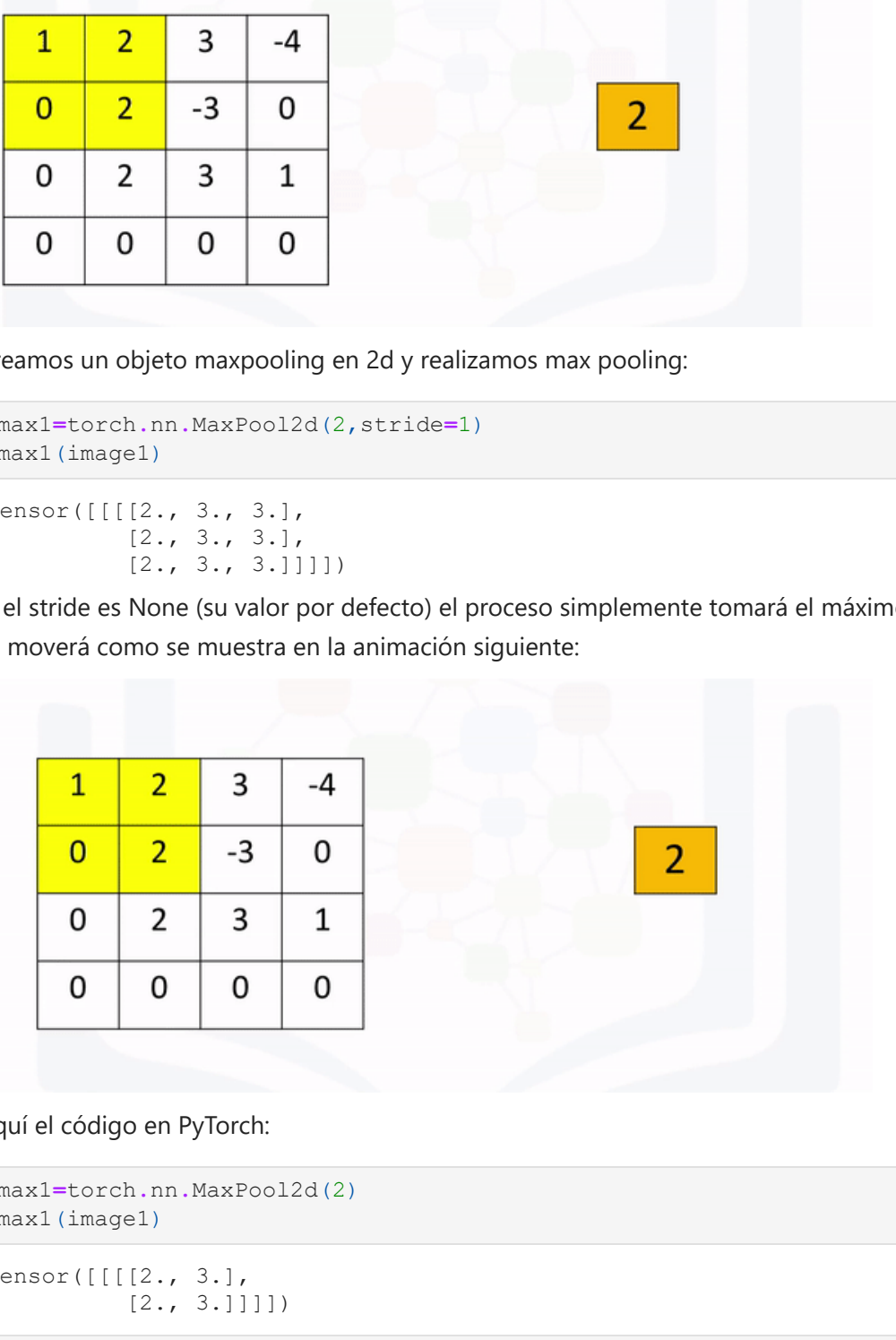
Considere la imagen siguiente:

In [8]:
```
image1=torch.zeros(1,1,4,4)
image1[0,0,0,:]=torch.tensor([1.0,2.0,3.0,-4.0])
image1[0,0,1,:]=torch.tensor([0.0,2.0,-3.0,0.0])
image1[0,0,2,:]=torch.tensor([0.0,2.0,3.0,1.0])

image1
```

Out[8]:
```
tensor([[[[ 1.,  2.,  3., -4.],
          [ 0.,  2., -3.,  0.],
          [ 0.,  2.,  3.,  1.],
          [ 0.,  0.,  0.,  0.]]]])
```

El max pooling simplemente toma el valor máximo en cada región. Considere la siguiente imagen. Para la primer región, el max pooling toma el elemento más grande en la región amarilla.



La región se mueve y el proceso se repite:

Creamos un objeto maxpooling en 2d y realizamos max pooling:

In [9]:
```
max1=torch.nn.MaxPool2d(2,stride=1)
max1(image1)
```

Out[9]:
```
tensor([[[[2., 3., 3.],
          [2., 3., 3.],
          [2., 3., 3.]]]])
```

Si el stride es None (su valor por defecto) el proceso simplemente tomará el máximo en un área prescrita y se moverá como se muestra en la animación siguiente:

Aquí el código en PyTorch:

In [14]:
```
max1=torch.nn.MaxPool2d(2)
max1(image1)
```

Out[14]:
```
tensor([[[[2., 3.],
          [2., 3.]]]])
```

In [ ]:
```

```