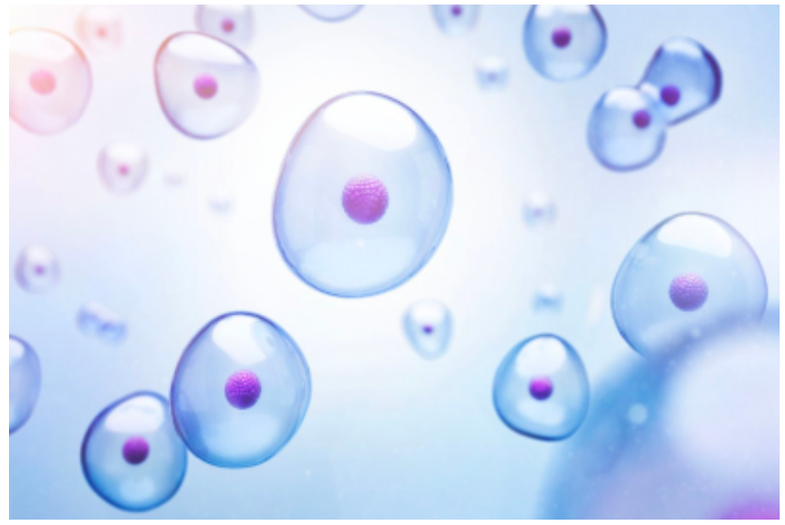
13. Redes Neuronales Convolucionales CNN

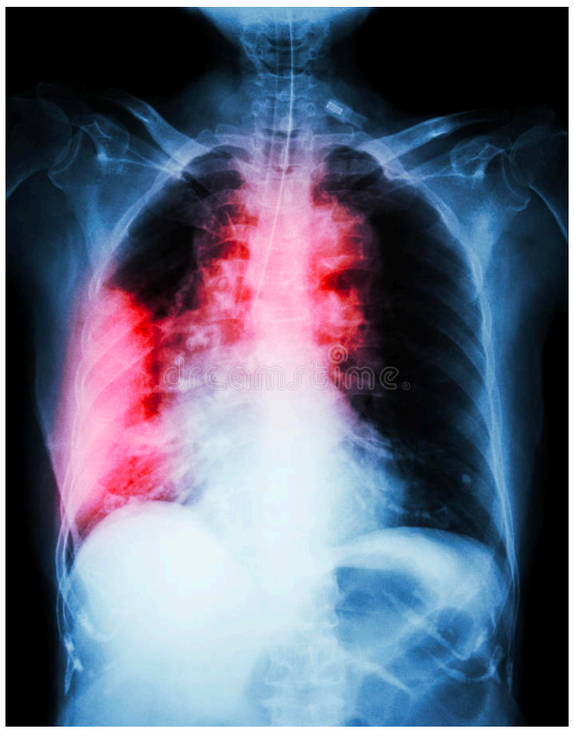
Aplicaciones

Robótica

Dispositivos móviles

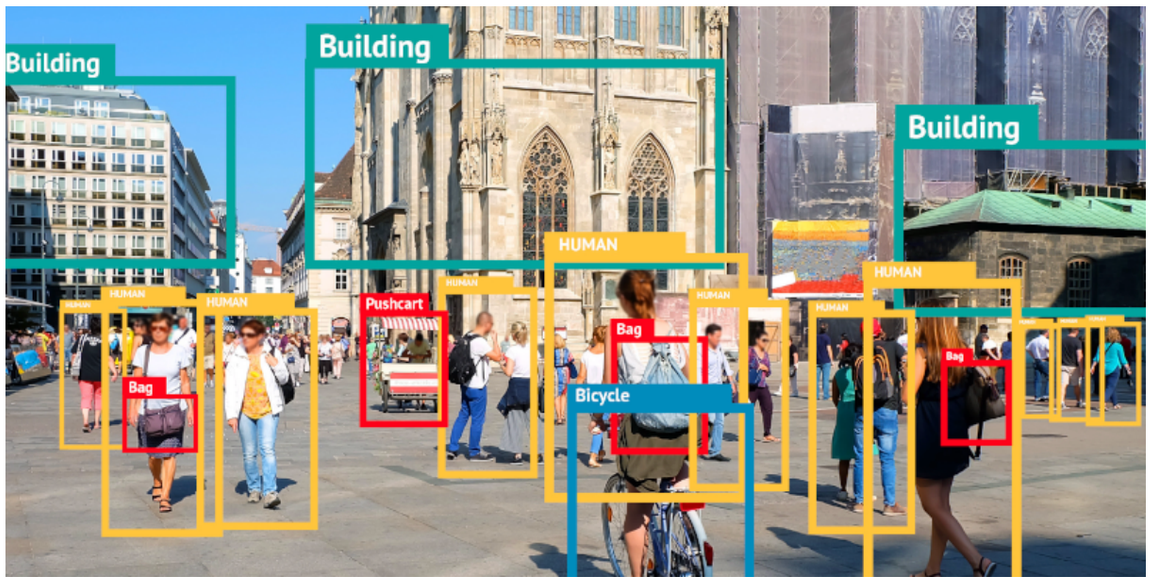


Biología



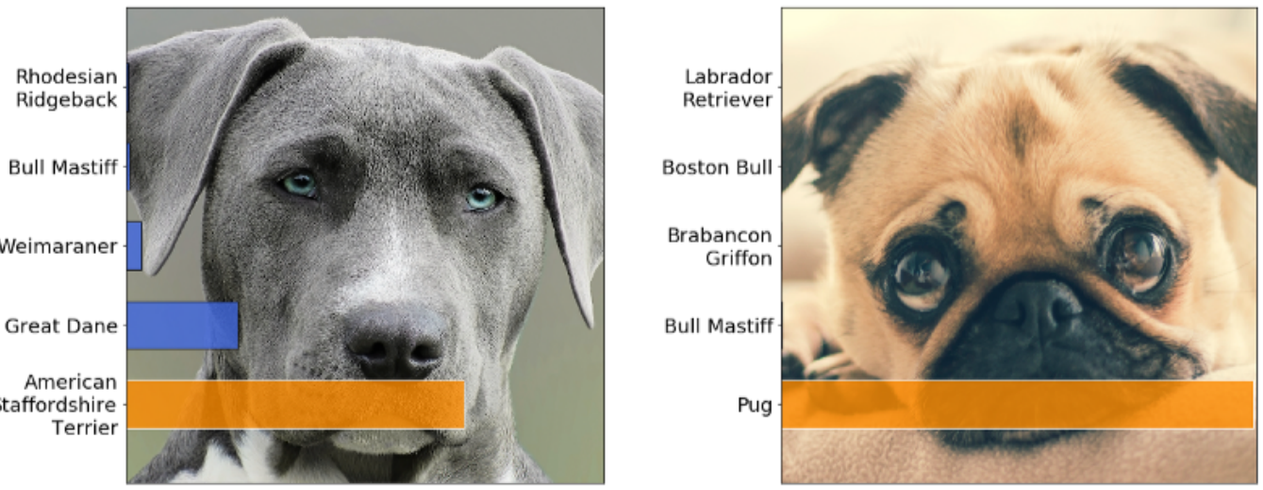
Medicina

Conducción autónoma



Detección de objetos

Reconocimiento de caras



Clasificación / Regresión

Segmentación semántica

Funcionamiento de las CNN

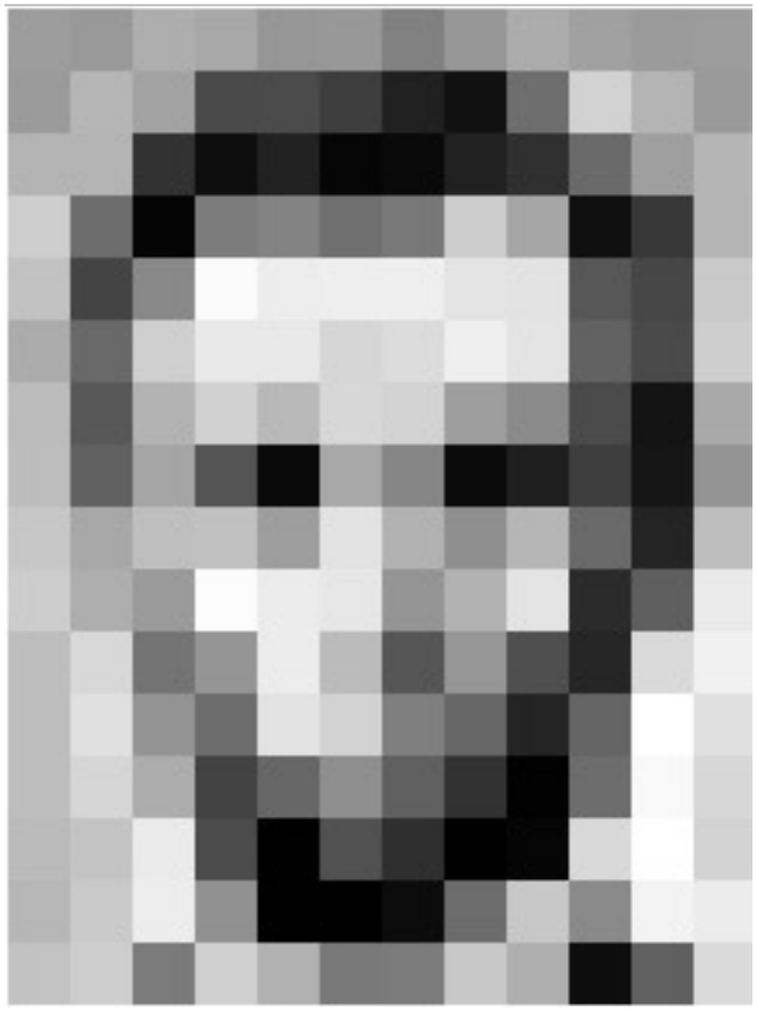


Imagen de entrada

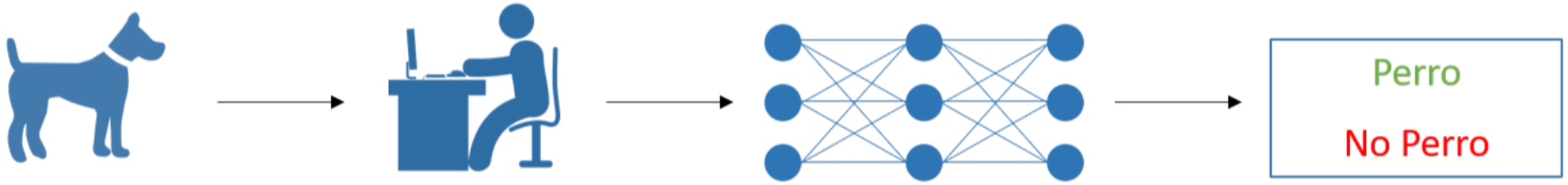
Clasificación

Clases	Probabilidad
Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

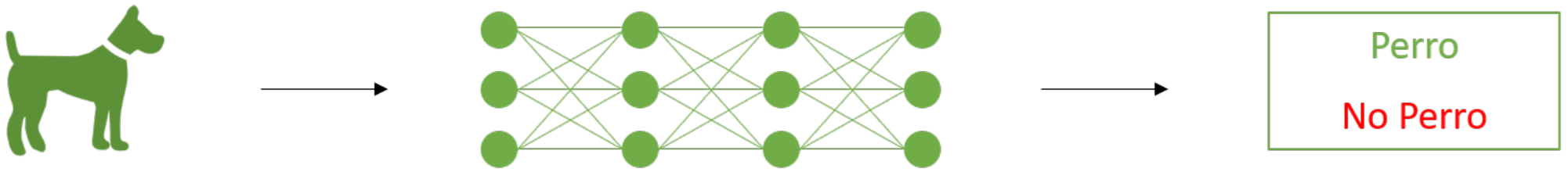
Salida

Representación de las imágenes

Machine Learning



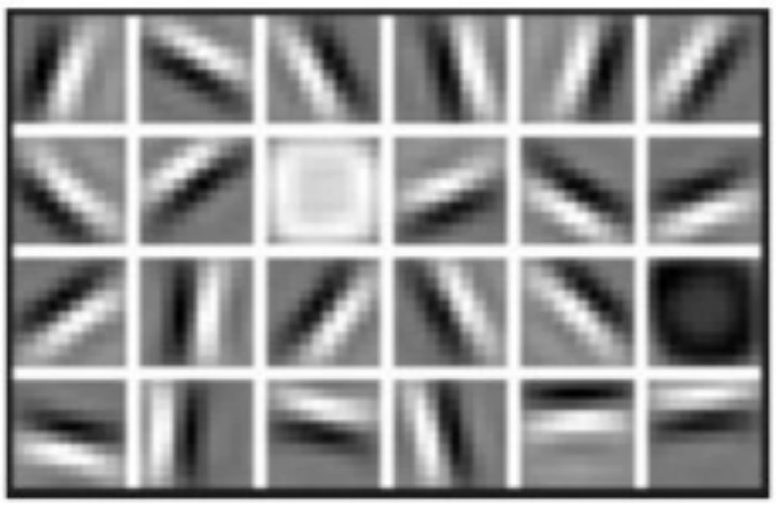
Deep Learning



Extracción de características

¿Podemos aprender **jerarquía de características** directamente de los datos en lugar de tener que extraerlas manualmente?

Las CNN extraen características a **distintos niveles** y las combinan para realizar la predicción.



Ejes, contornos

Ojos, orejas, nariz, boca

Caras

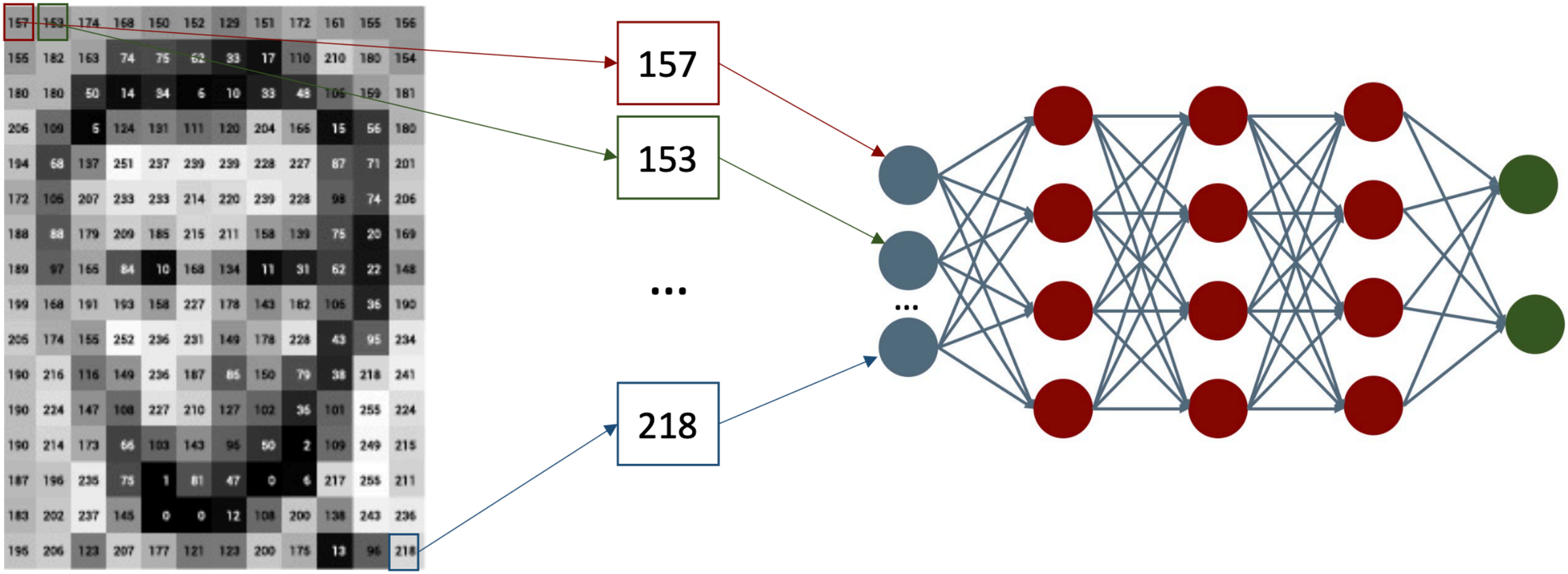
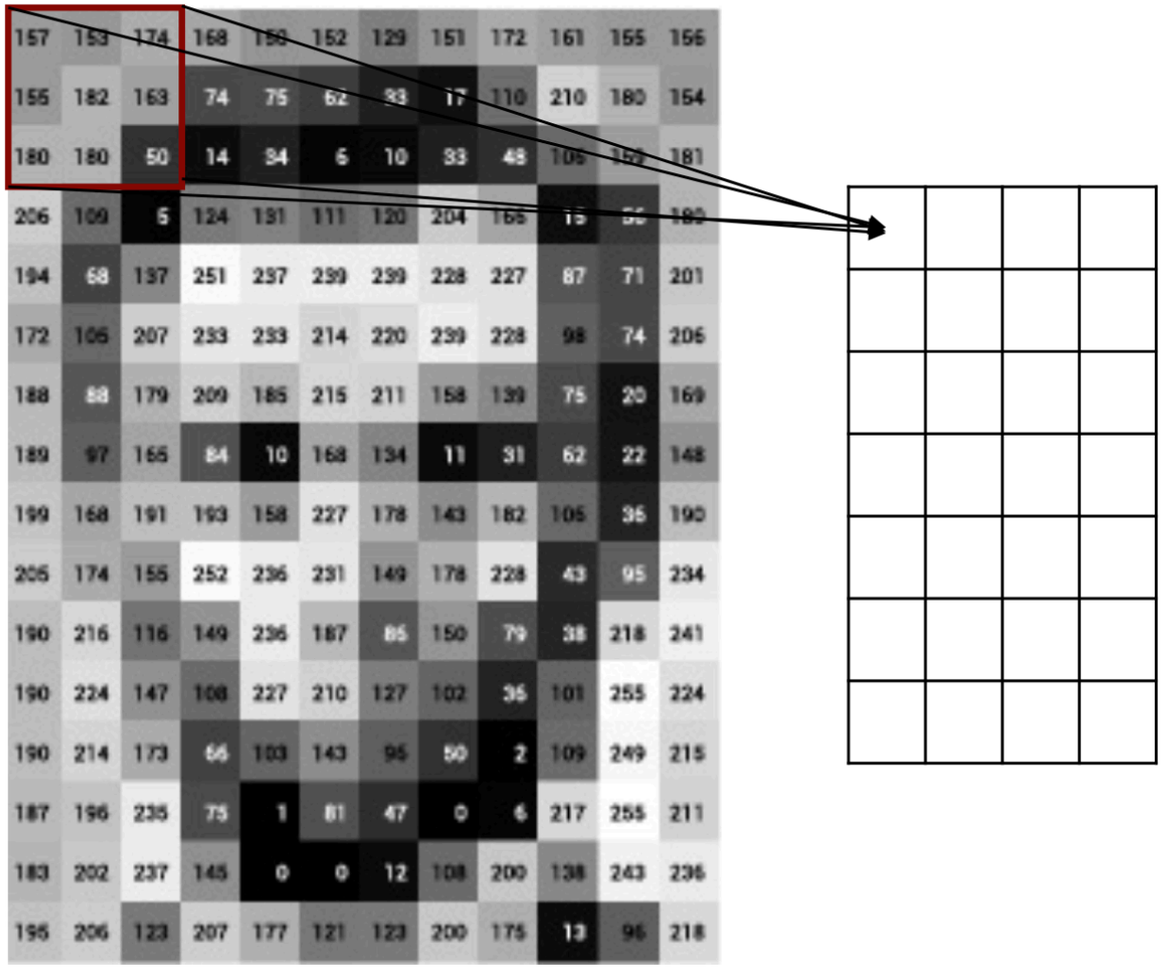
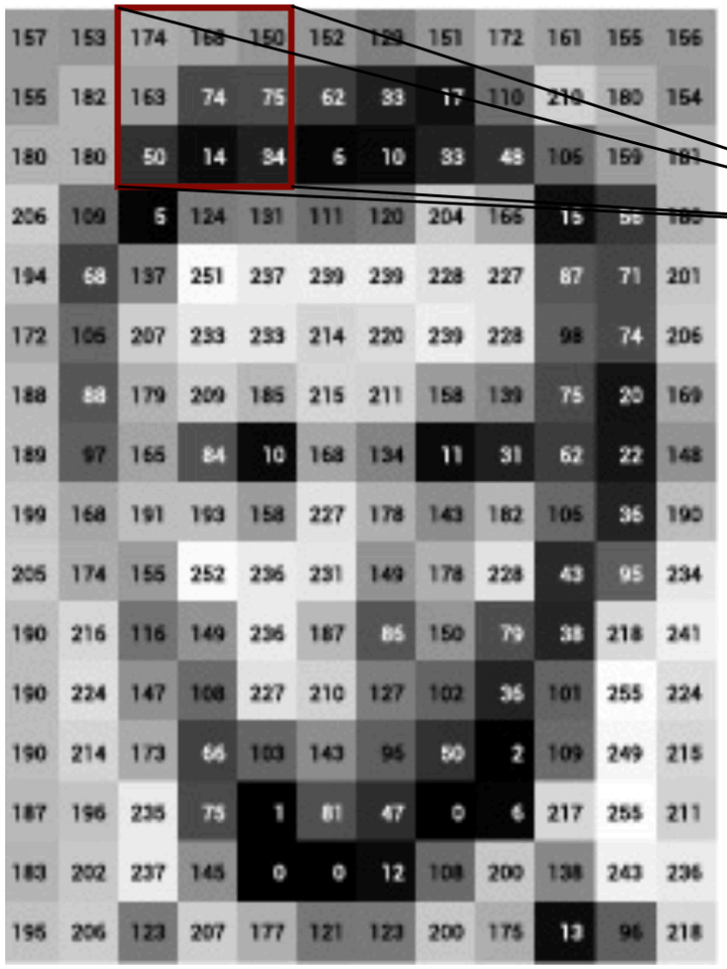


Imagen de entrada 2D | Vector 1D | Red neuronal totalmente conectada

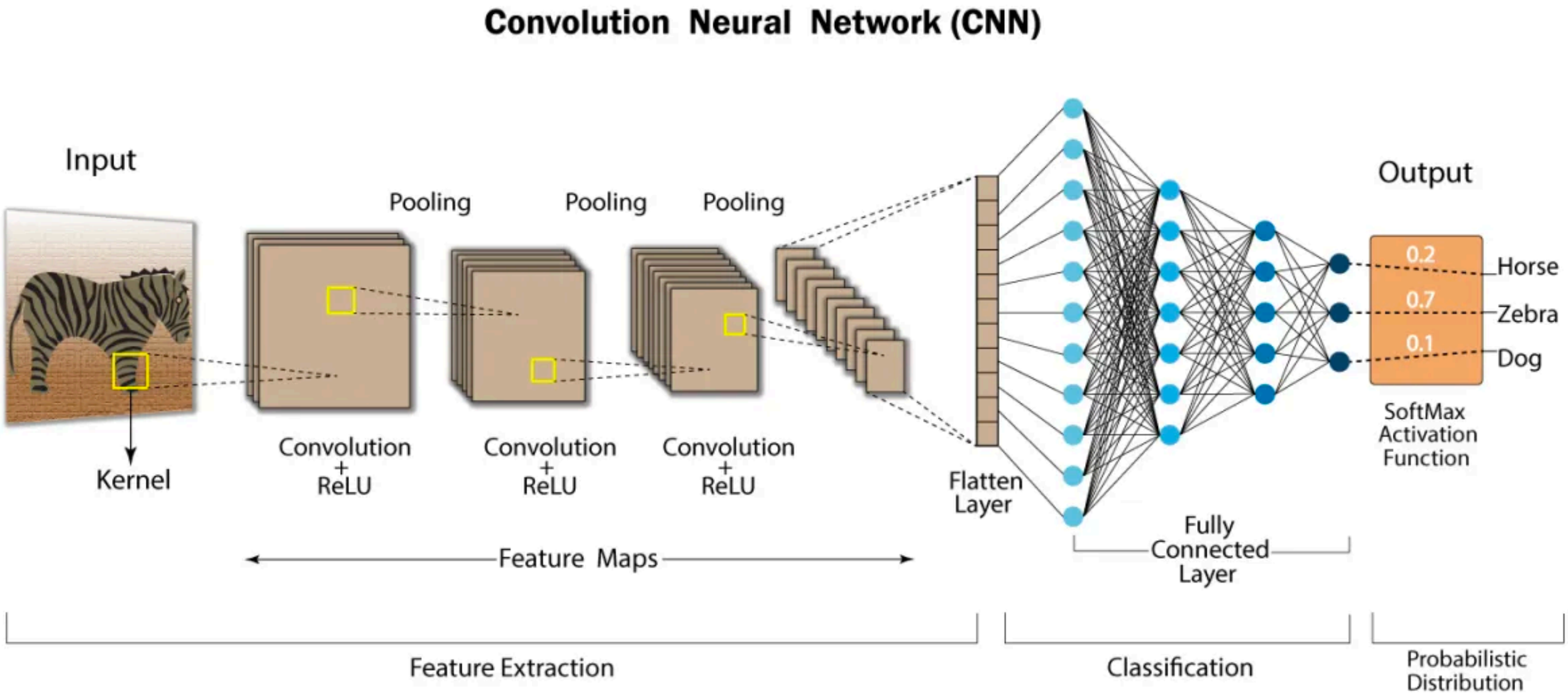
- **Se pierde información espacial.**
- **1 neurona por píxel → Mucho parámetros**

Extracción de características





Estructura de una CNN



Capa convolucional

Filtro o Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

Detección de bordes

0	-1	0
-1	5	-1
0	-1	0

Perfilado

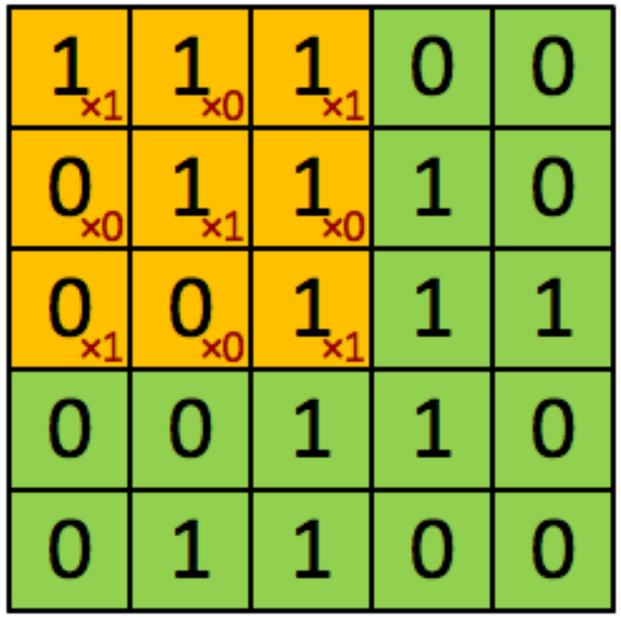
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Difuminado

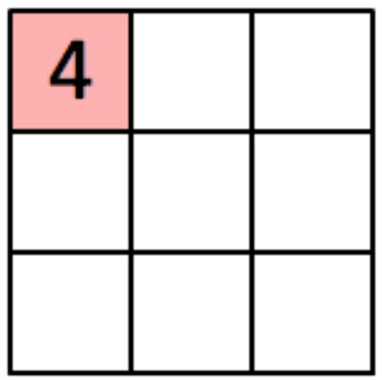
Capa convolucional

Filtro o Kernel $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

Dot product (Producto escalar)



Image



Convolved Feature

Capa convolucional

Filtro o Kernel

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Son estos filtros los que la CNN aprende por sí misma.

Capa convolucional. Filtro o Kernel

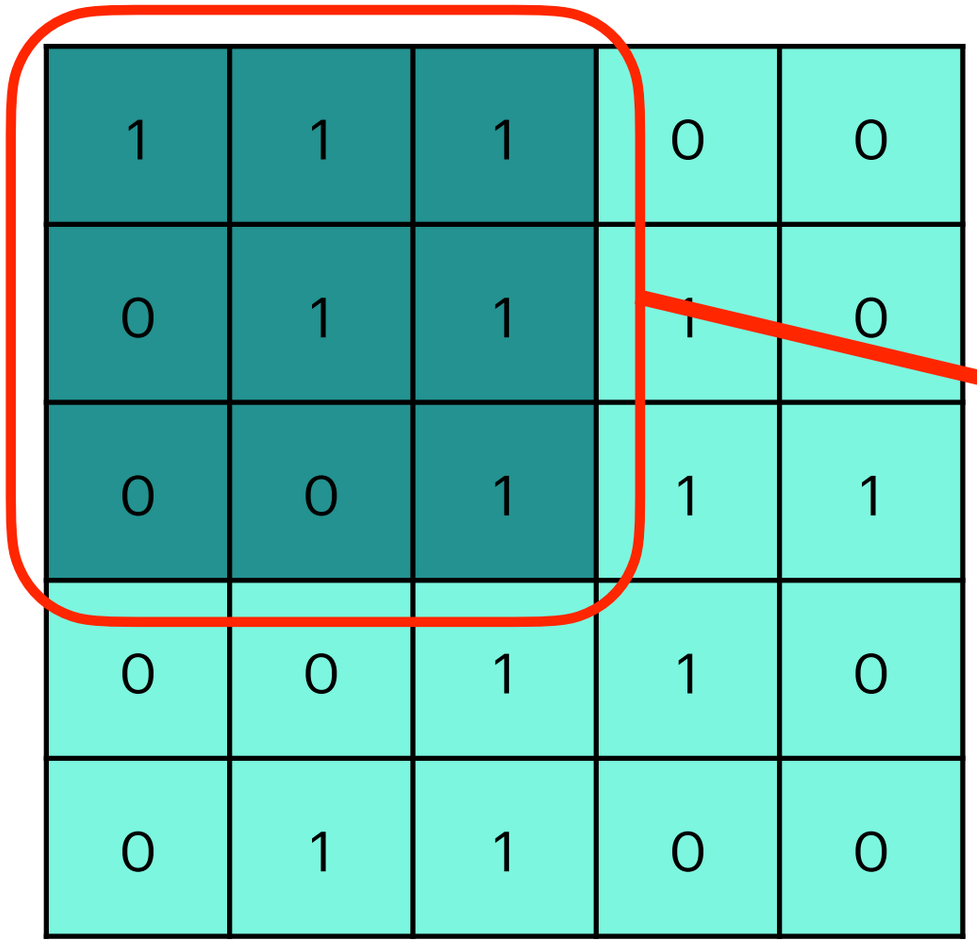
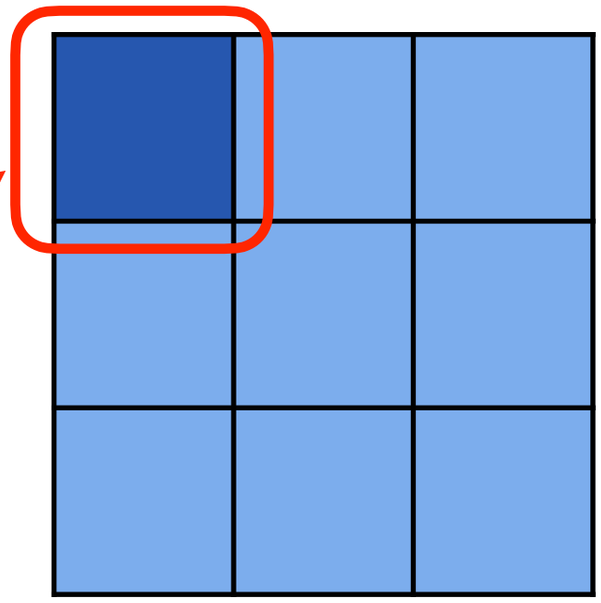


Imagen de entrada

Filtro o kernel



Mapa de características
(Feature map)
(o Activation map)

Activación de la neurona $(p, q) = g(\sum_{i=1}^{3}\sum_{j=1}^{3} w_{ij}x_{i+p,j+q} + b)$

Capa convolucional. Filtro o Kernel

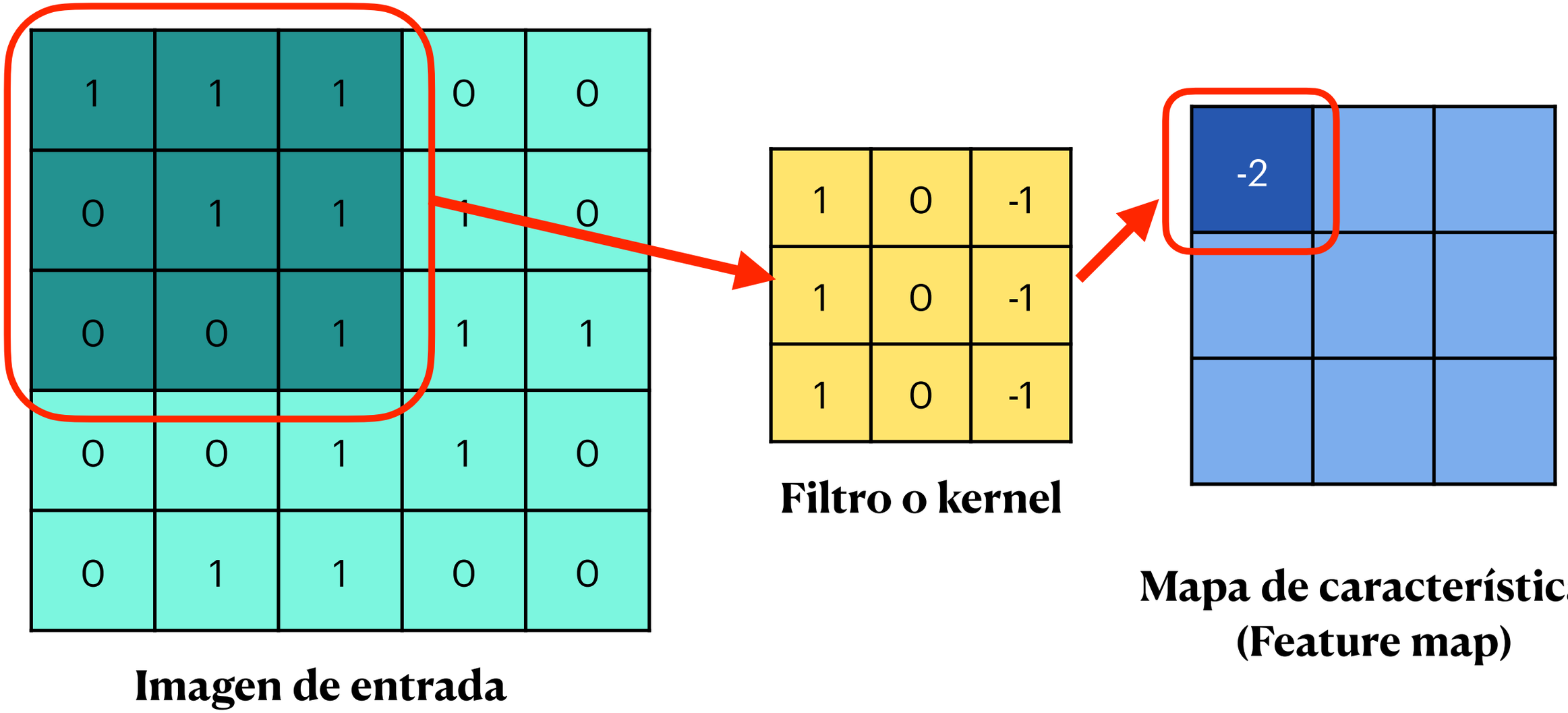


Imagen de entrada

Filtro o kernel

Mapa de características (Feature map)

$$1*1+1*0+1*(-1)+0*1+1*0+1*(-1)+0*1+0*0+1*(-1)=-2$$

Capa convolucional. Filtro o Kernel

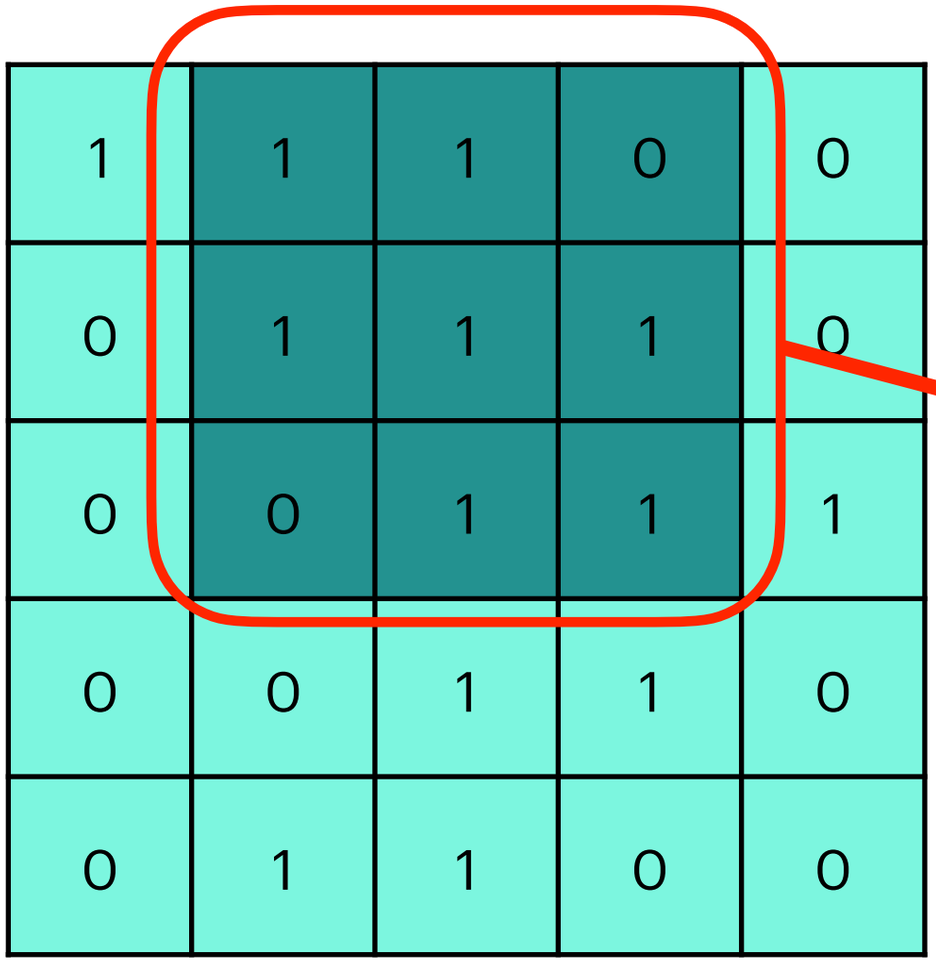
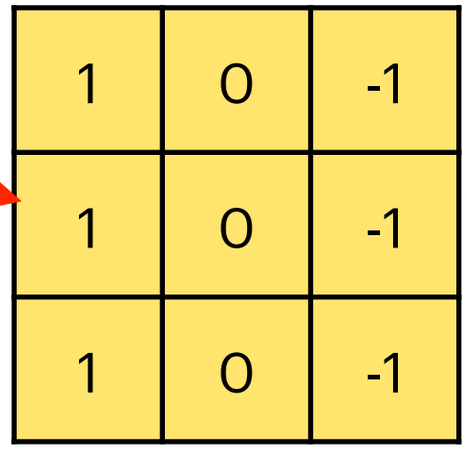
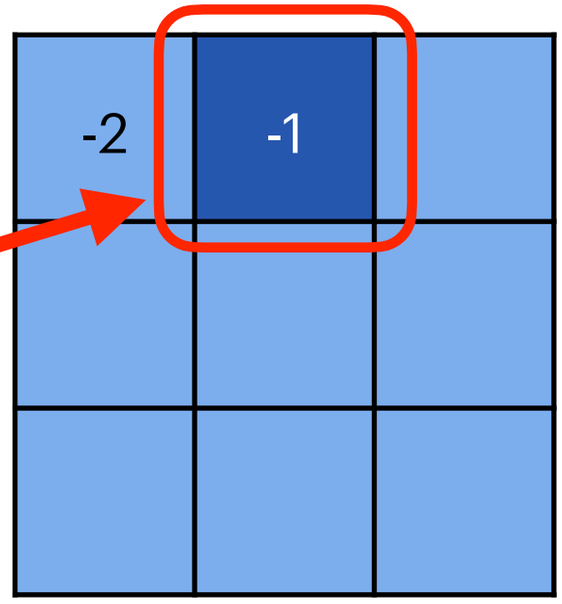


Imagen de entrada



Filtro o kernel



Mapa de características (Feature map)

$$1*1+1*0+0*(-1)+1*1+1*0+1*(-1)+0*1+1*0+1*(-1)=-1$$

Capa convolucional. Filtro o Kernel

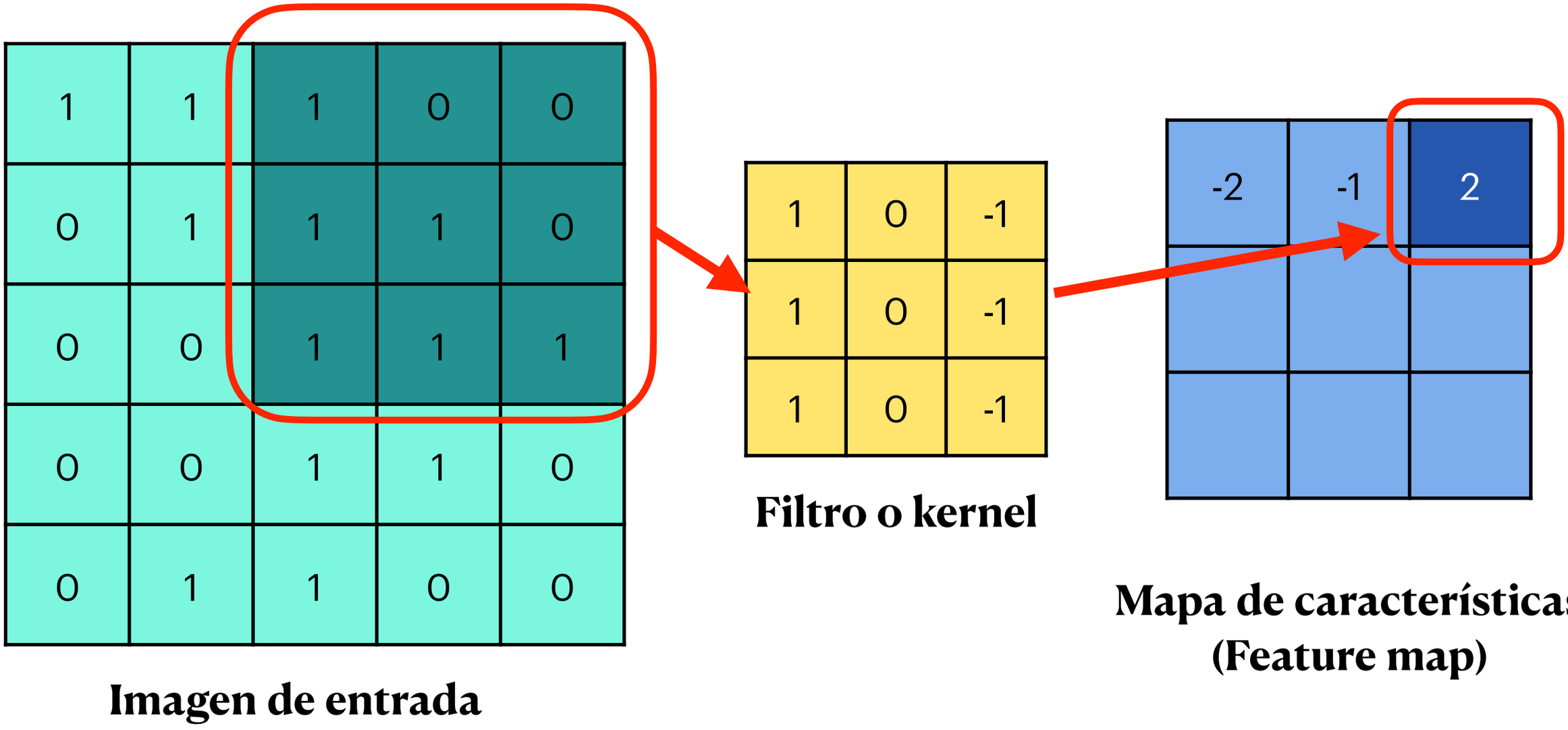


Imagen de entrada

Filtro o kernel

Mapa de características (Feature map)

$$1*1+0*0+0*(-1)+1*1+1*0+0*(-1)+1*1+1*0+1*(-1)=2$$

Capa convolucional. Filtro o Kernel

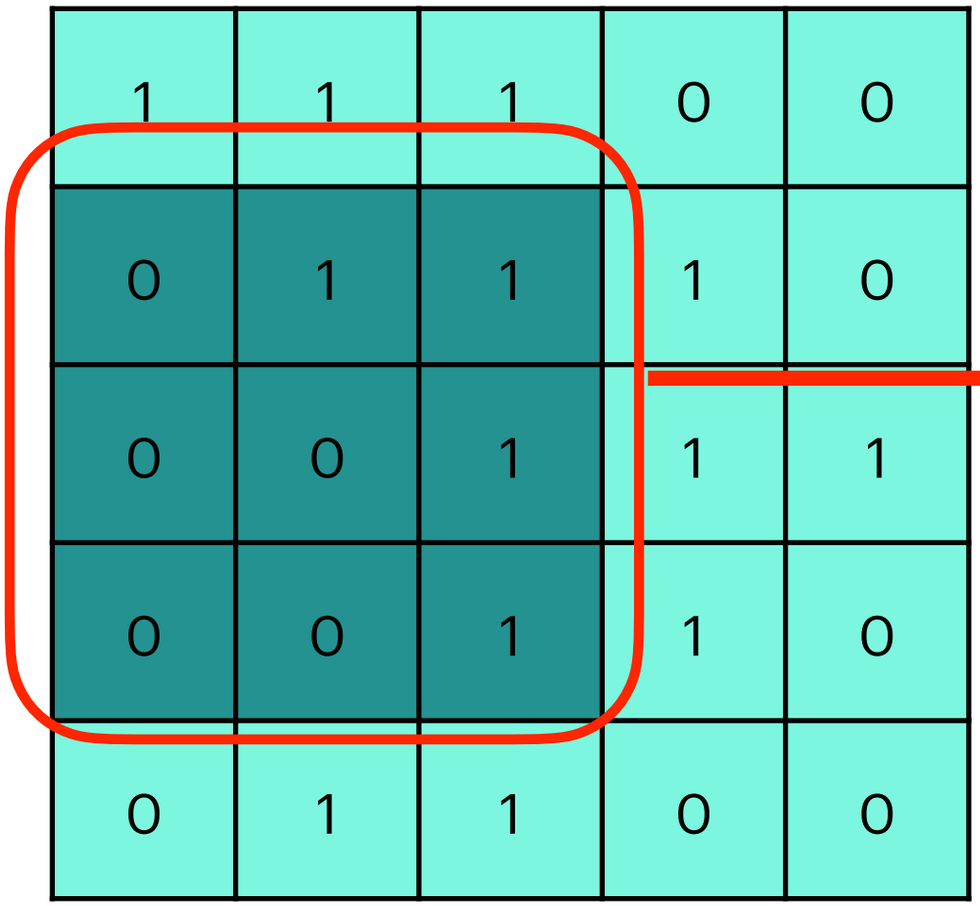
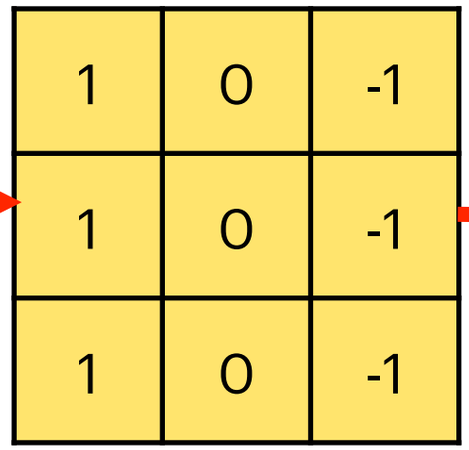
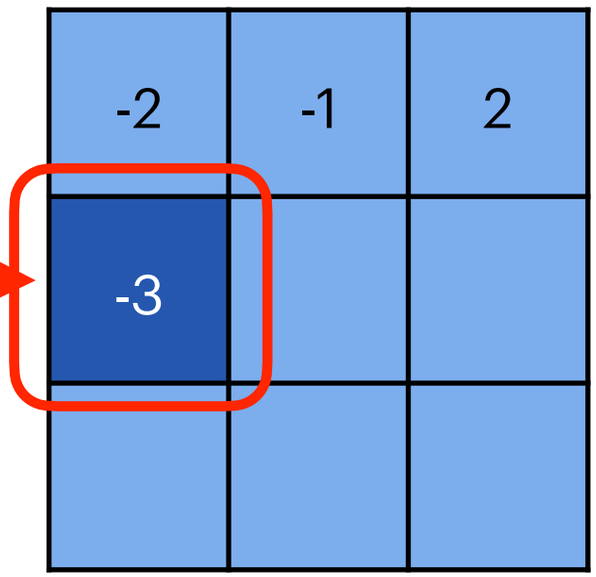


Imagen de entrada



Filtro o kernel



Mapa de características (Feature map)

$$0*1+1*0+1*(-1)+0*1+0*0+1*(-1)+0*1+0*0+1*(-1)=-3$$

Capa convolucional. Filtro o Kernel

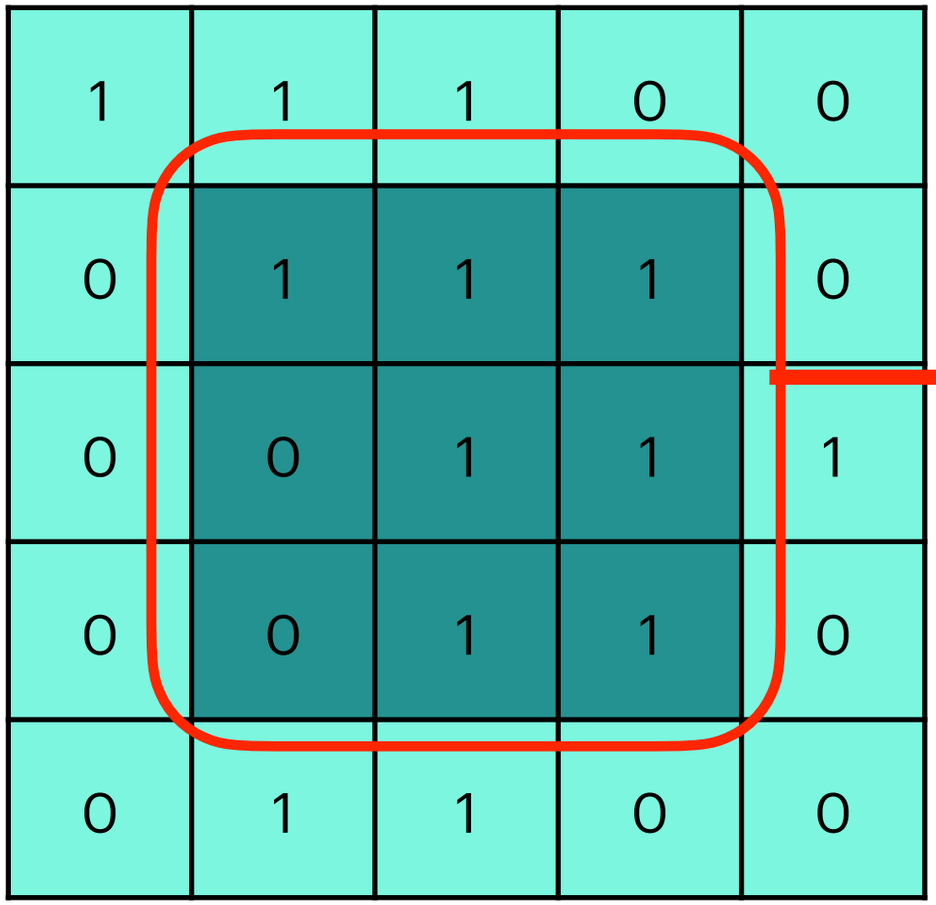
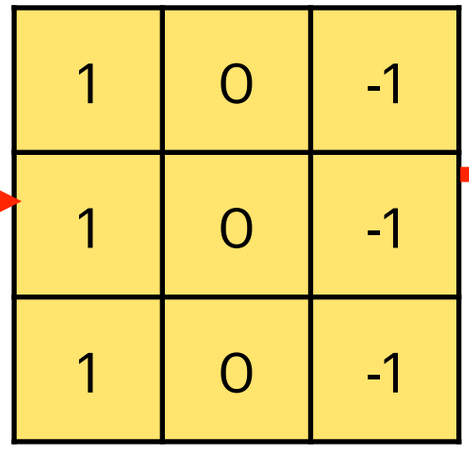
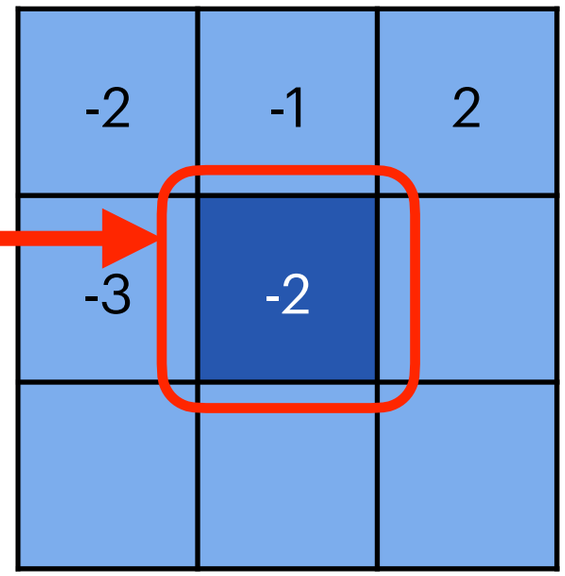


Imagen de entrada



Filtro o kernel



Mapa de características (Feature map)

$$1*1+1*0+1*(-1)+0*1+1*0+1*(-1)+0*1+1*0+1*(-1)=-2$$

Capa convolucional. Filtro o Kernel

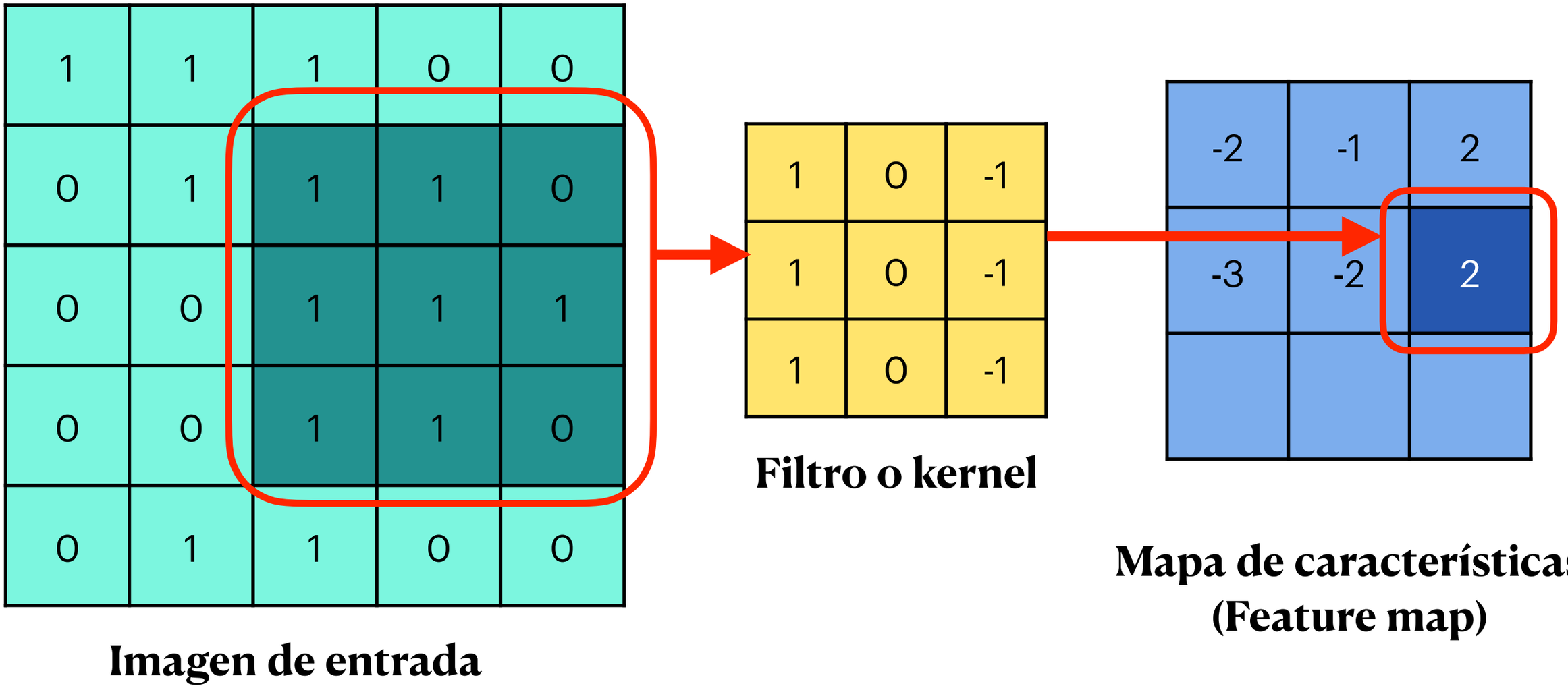


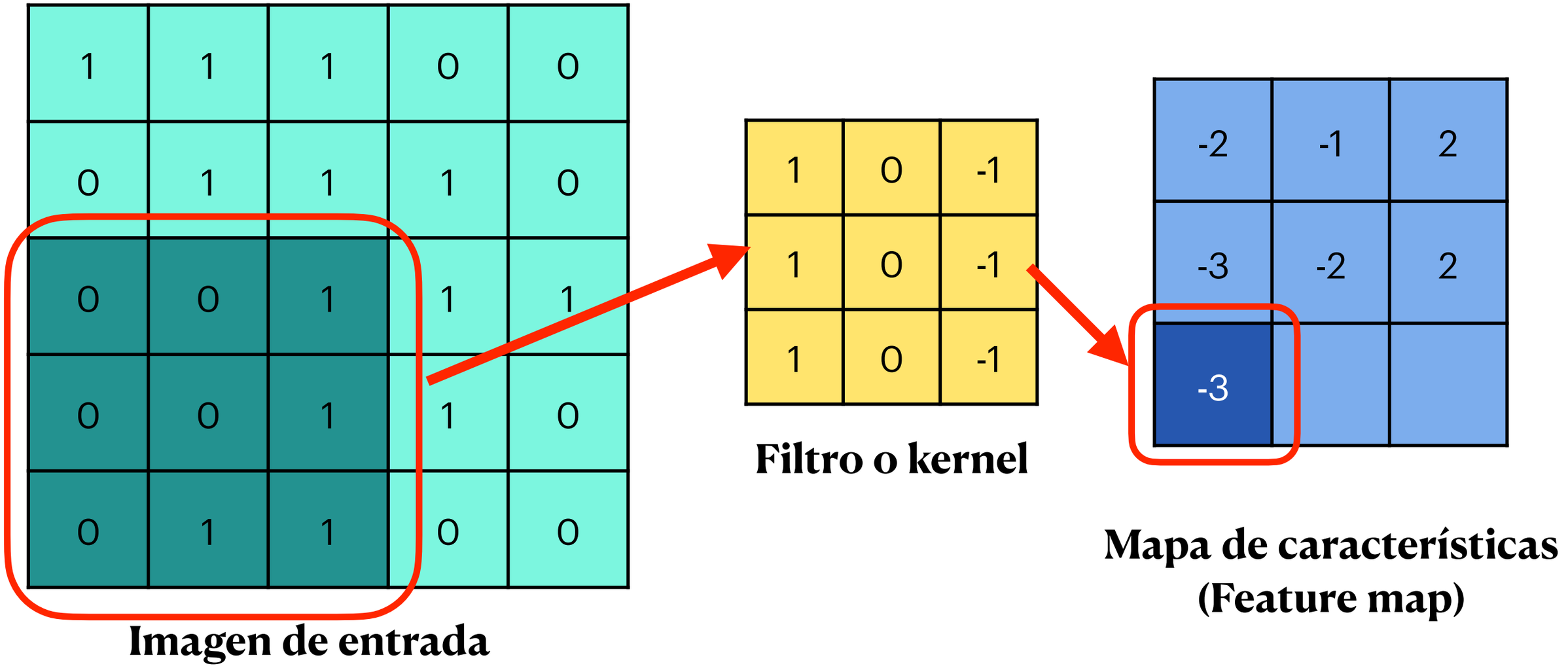
Imagen de entrada

Filtro o kernel

Mapa de características (Feature map)

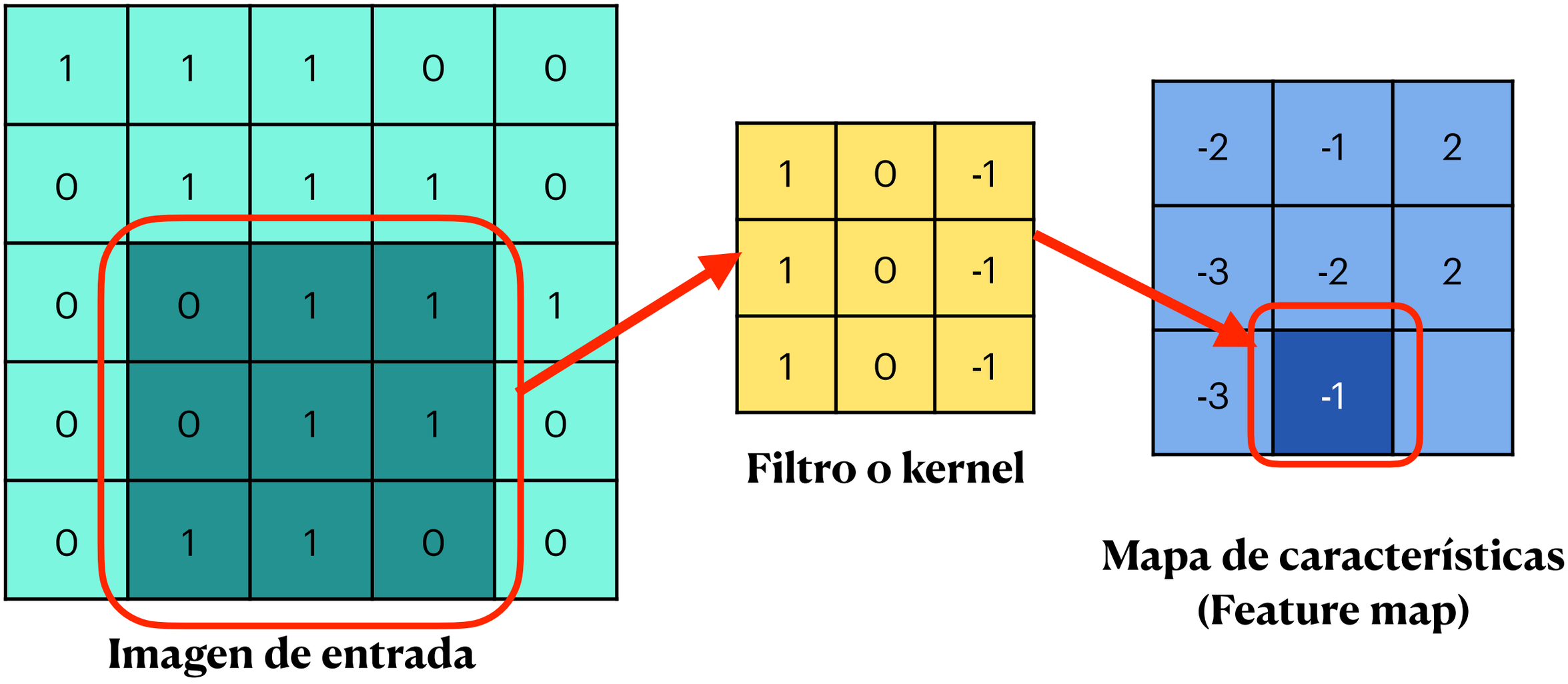
$$1*1+1*0+0*(-1)+1*1+1*0+1*(-1)+1*1+1*0+0*(-1)=2$$

Capa convolucional. Filtro o Kernel



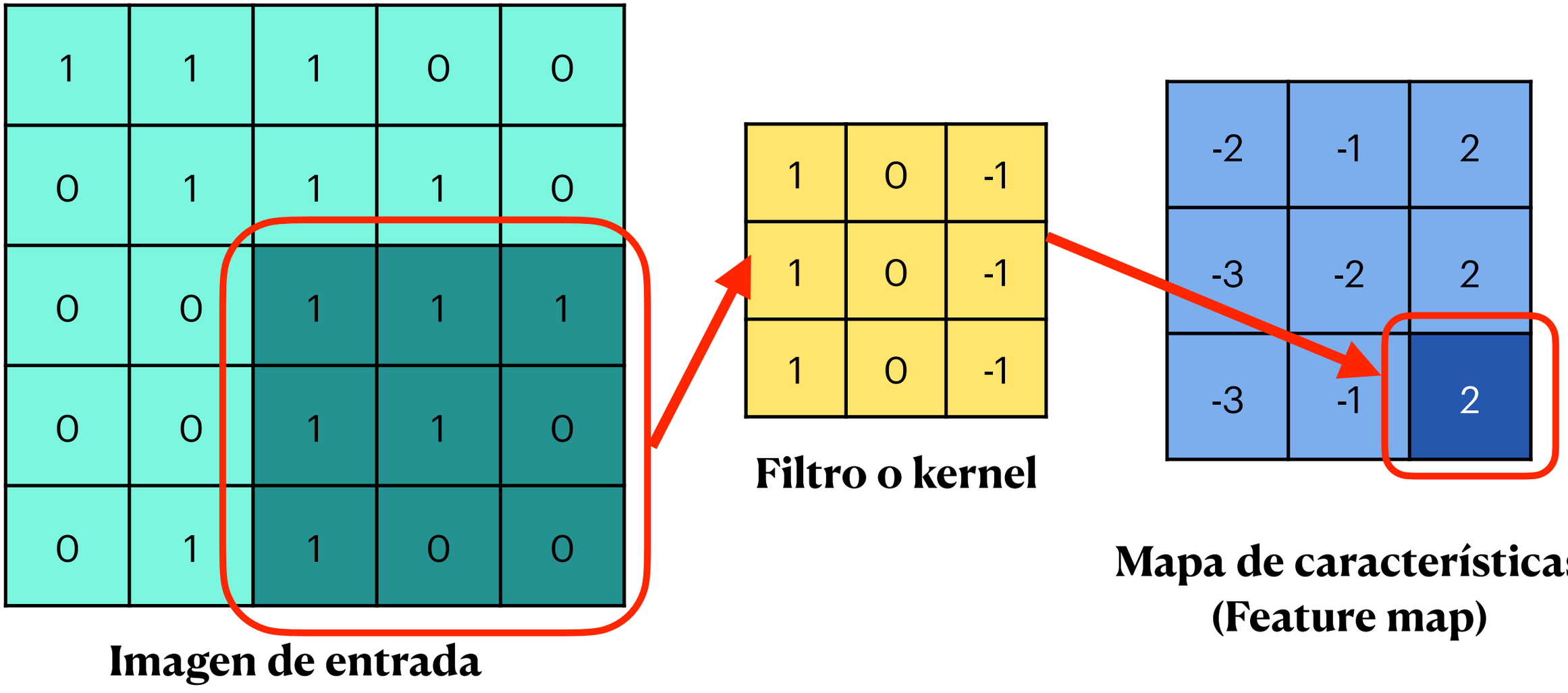
$$0*1+0*0+1*(-1)+0*1+0*0+1*(-1)+0*1+1*0+1*(-1)=-3$$

Capa convolucional. Filtro o Kernel



$$0*1+1*0+1*(-1)+0*1+1*0+1*(-1)+1*1+1*0+0*(-1)=-1$$

Capa convolucional. Filtro o Kernel



$$1*1+1*0+1*(-1)+1*1+1*0+0*(-1)+1*1+0*0+0*(-1)=2$$

Capa convolucional. Stride (paso)

Para stride 1

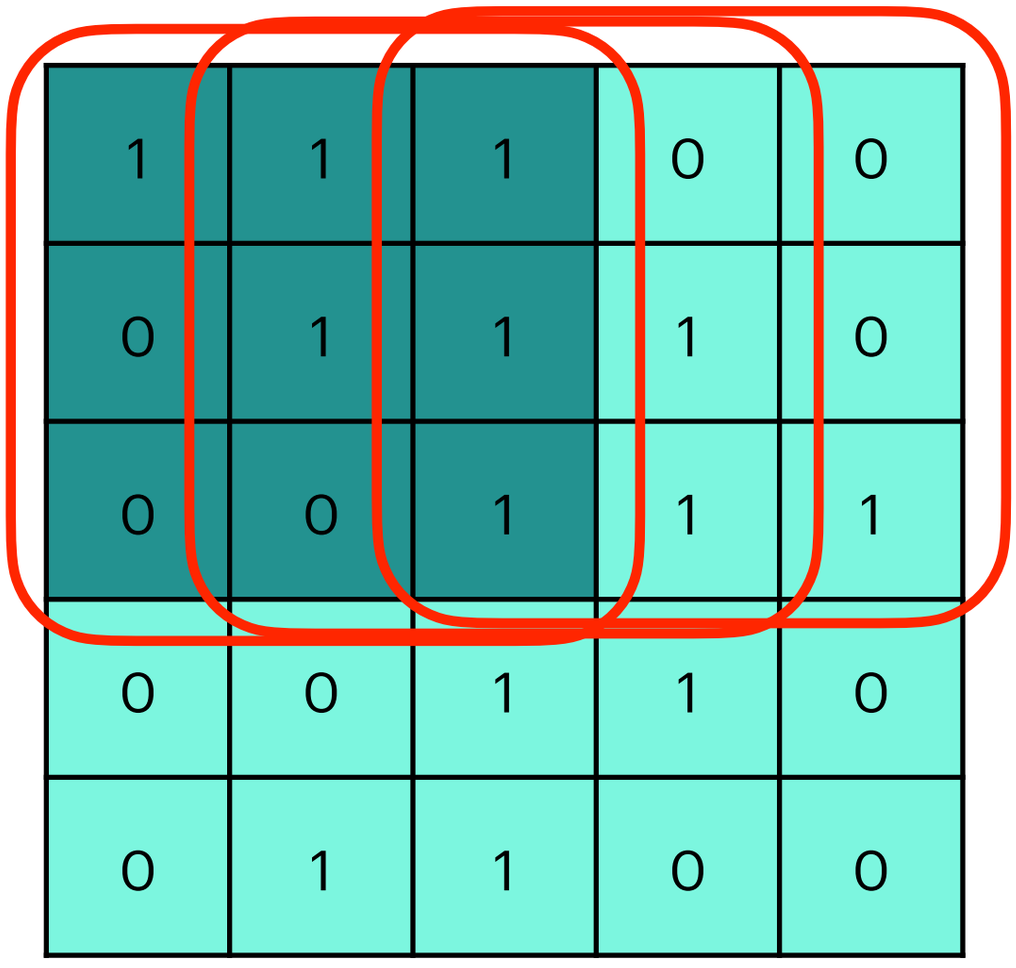
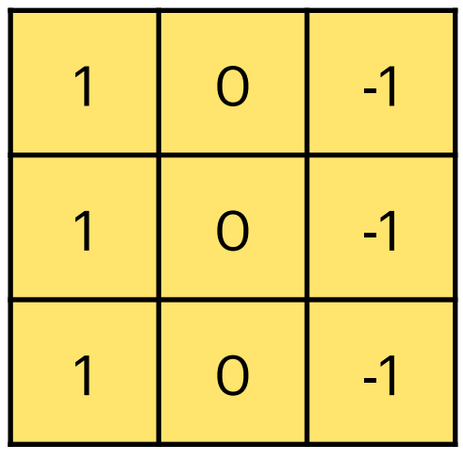
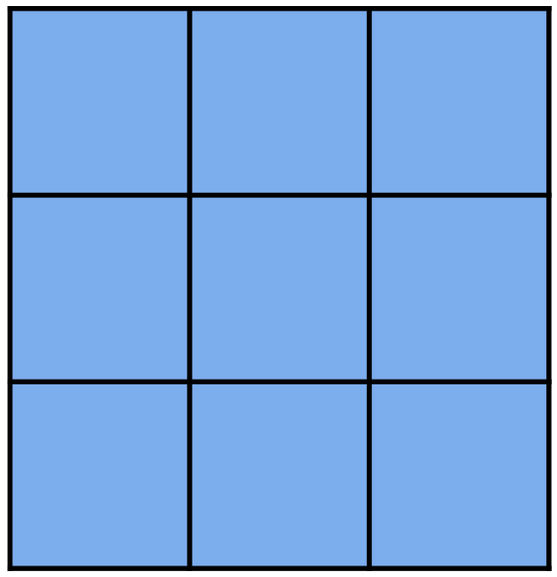


Imagen de entrada
(5 x 5)



Filtro o kernel
3 x 3



Mapa de características
(3 x 3)

Capa convolucional. Stride (paso)

Para stride 2

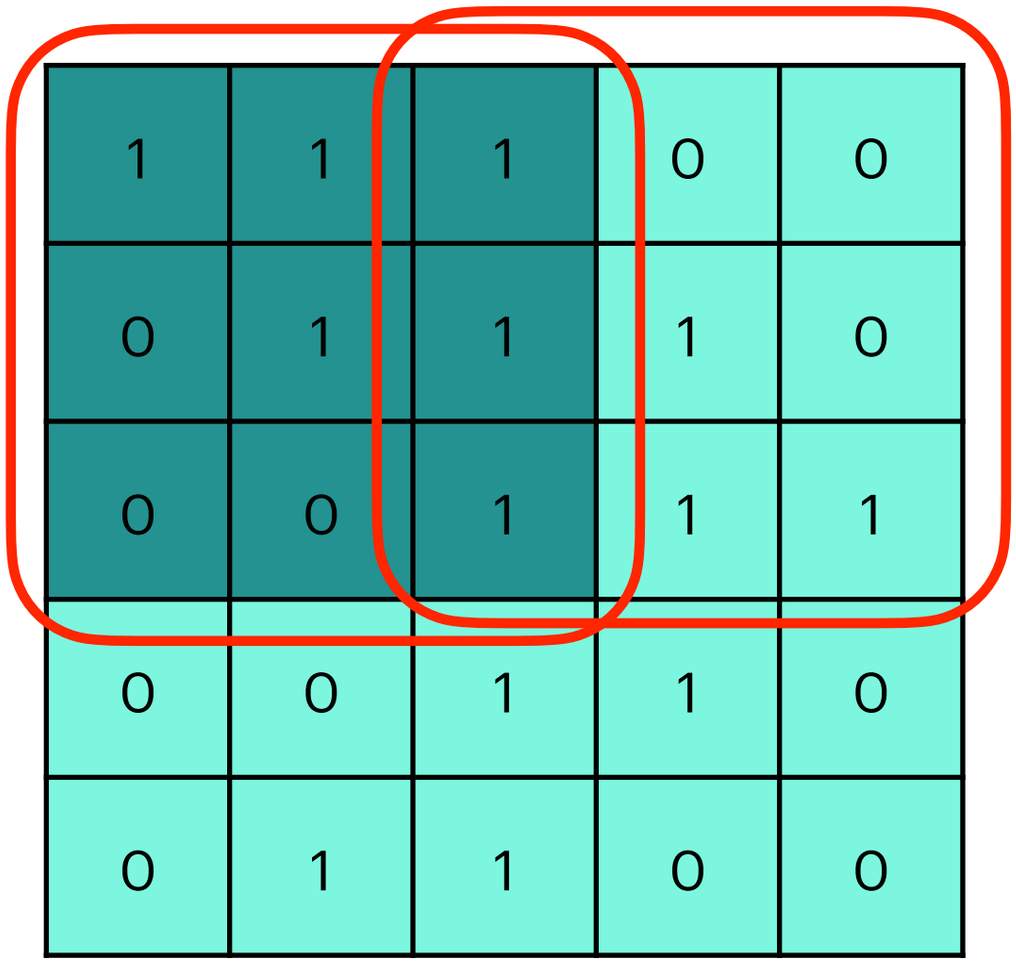
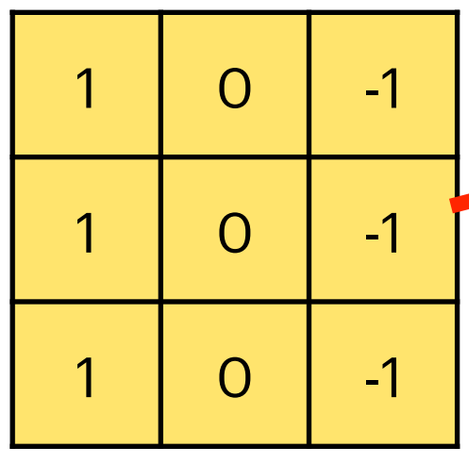
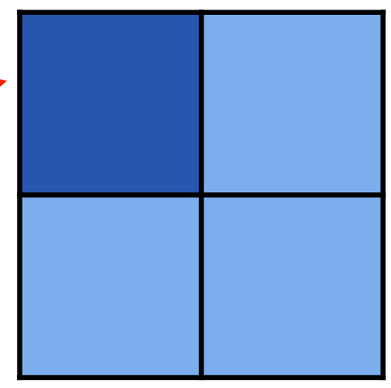


Imagen de entrada
(5 x 5)



Filtro o kernel
3 x 3



Mapa de características
(2 x 2)

Capa convolucional. Stride (paso)

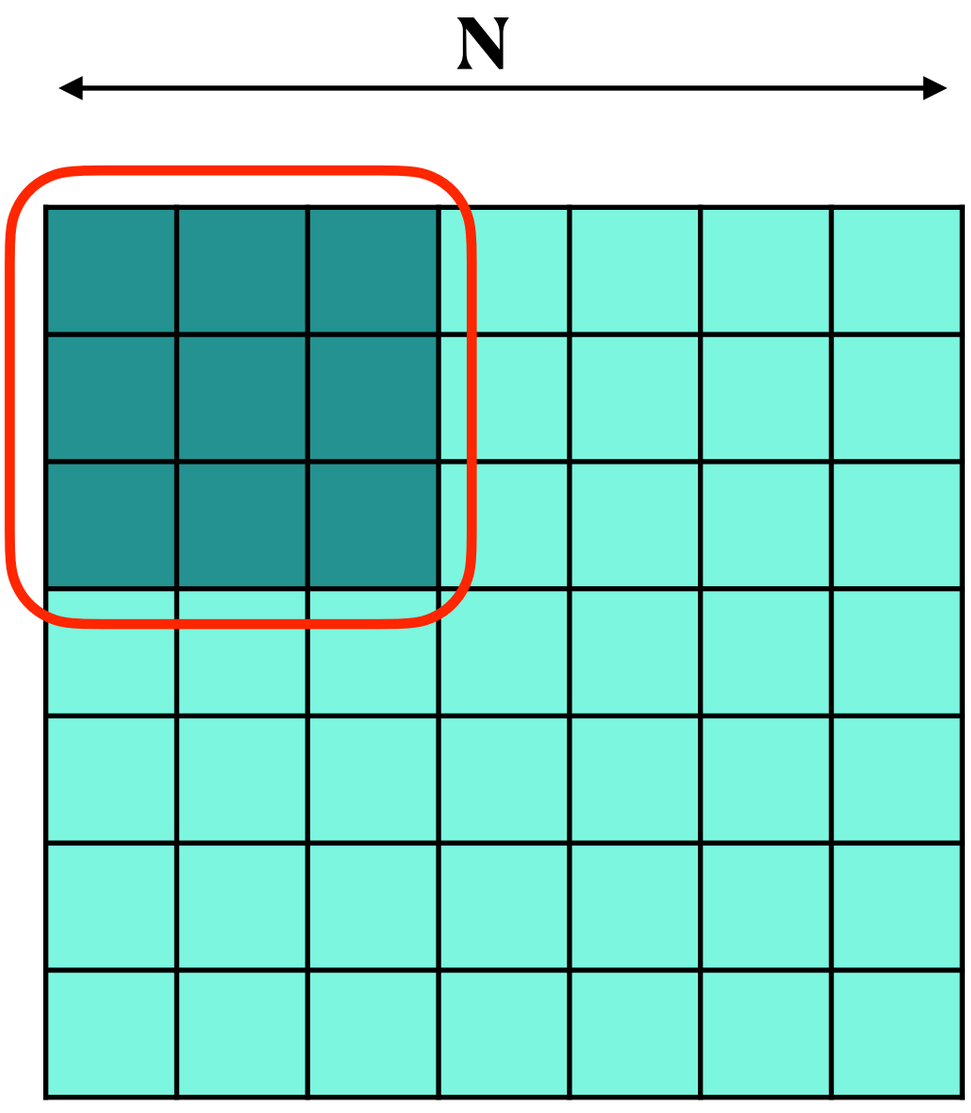
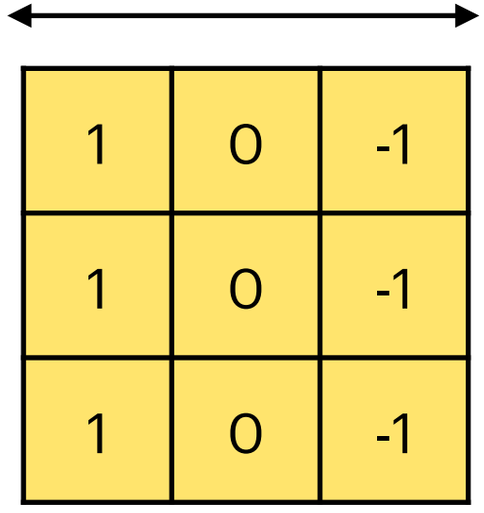


Imagen de entrada
(7 x 7)



Filtro o kerner
3 x 3

Tamaño de feature map:

(N - F) / stride + 1

e.g. N = 7, F = 3
stride = 1 -> (7-3)/1 + 1 = 5

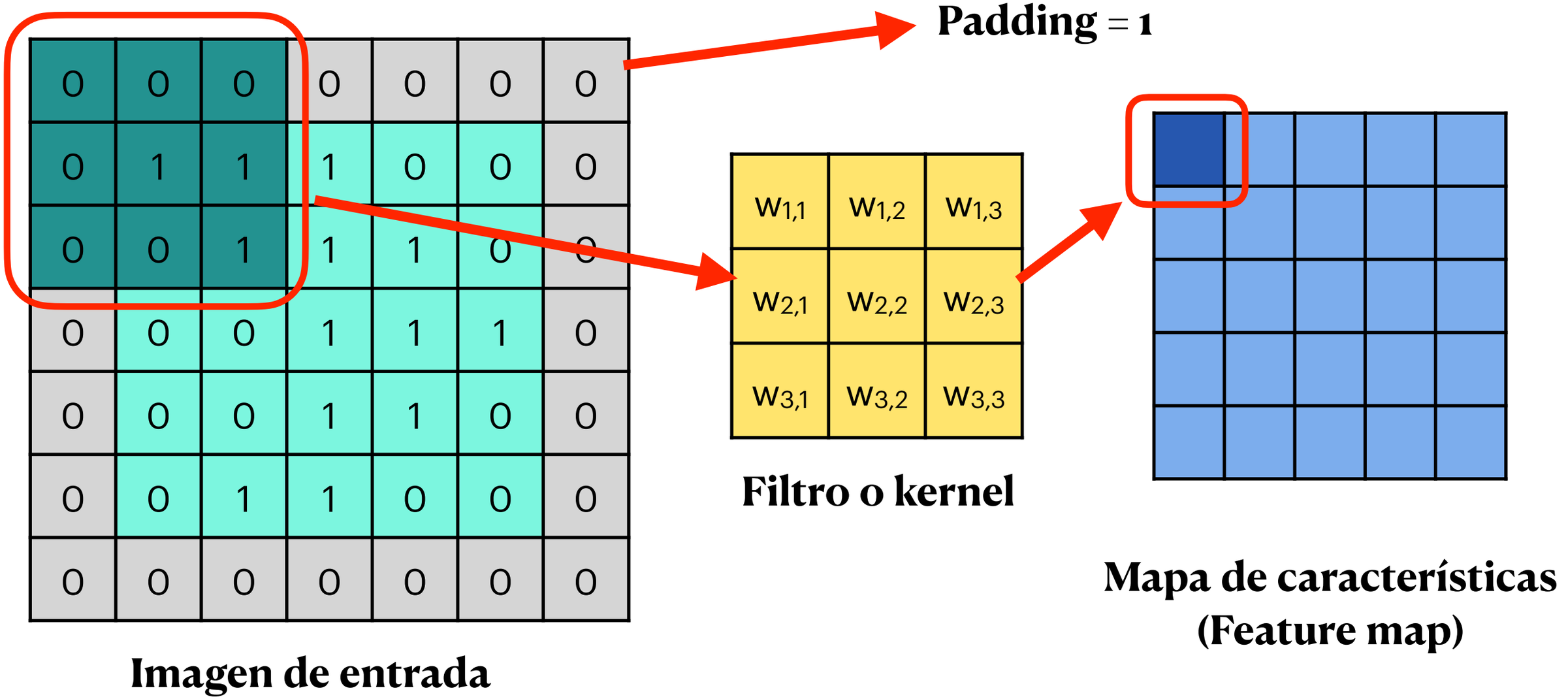
stride = 2 -> (7-3)/2 + 1 = 3

stride = 3 -> (7-3)/3 + 1 = 2,33

Capa convolucional. Padding

Margen que se añade a la imagen de entrada para **controlar el tamaño** del mapa de características que se genera. Hay que evitar reducir el tamaño de la imagen muy rápido.

Necesario para capturar información de los **bordes de la imagen**.



Capa convolucional. Padding

Padding same -> Padding necesario para que el mapa de características tenga el mismo tamaño que el original

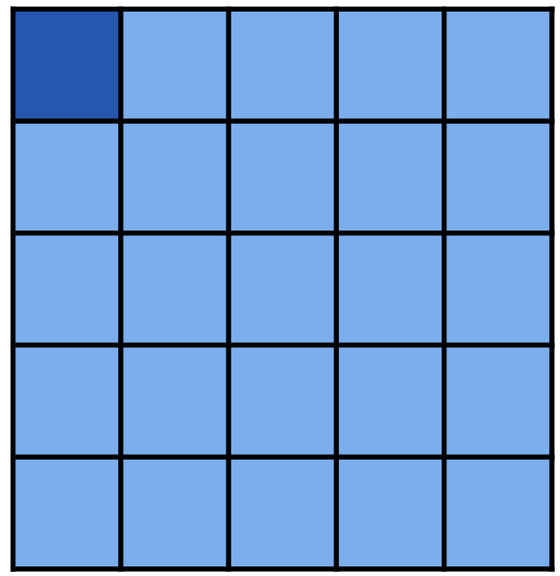
0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Padding=1

Imagen de entrada

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$

Filtro o kernel



Mapa de características (Feature map)

Capa convolucional. Filtros o Kernels

Normalmente tendremos **múltiples filtros**. Cada uno de ellos se especializará en detectar un tipo de **característica**.

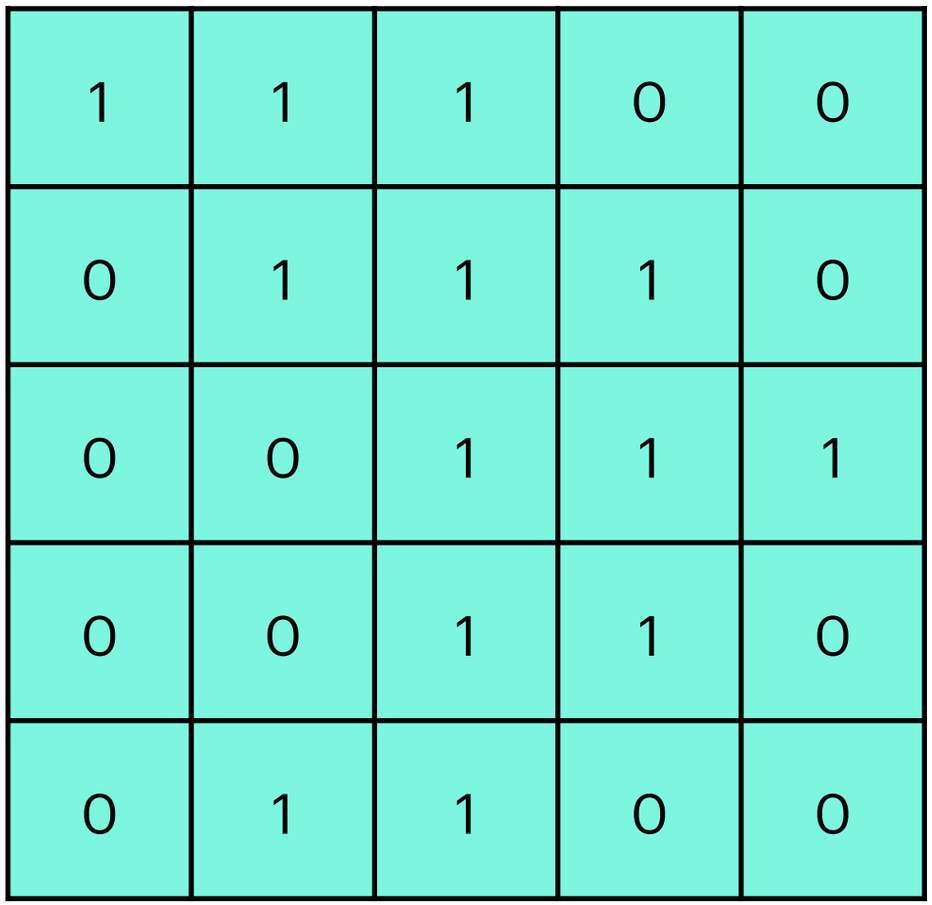
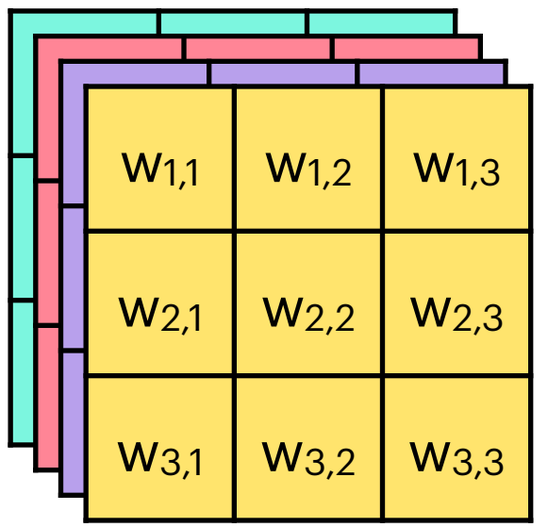
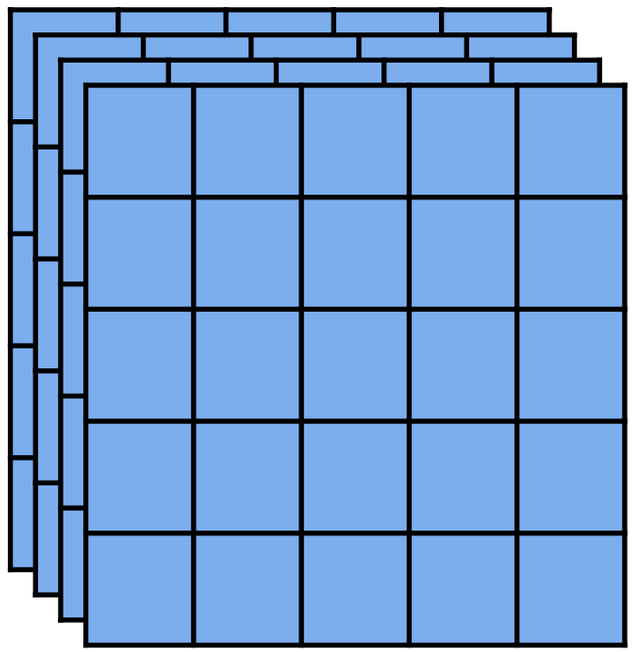


Imagen de entrada



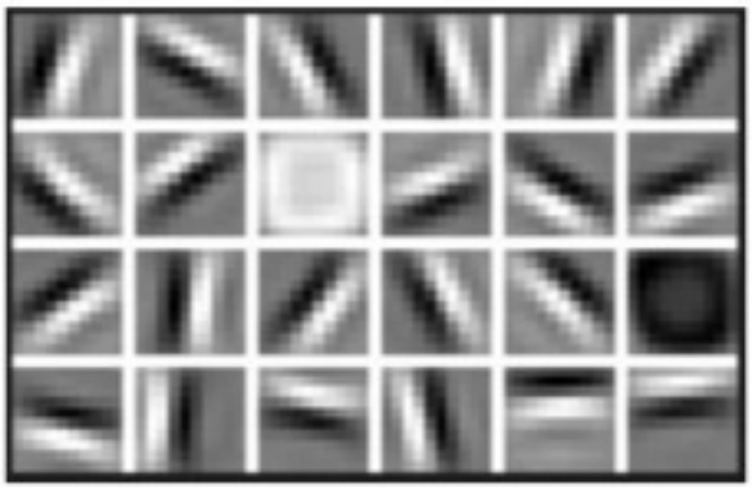
Filtros o kernels



Mapas de características (Feature maps)

Capa convolucional. Filtros o Kernels

24 filtros de la 1ª capa convolucional

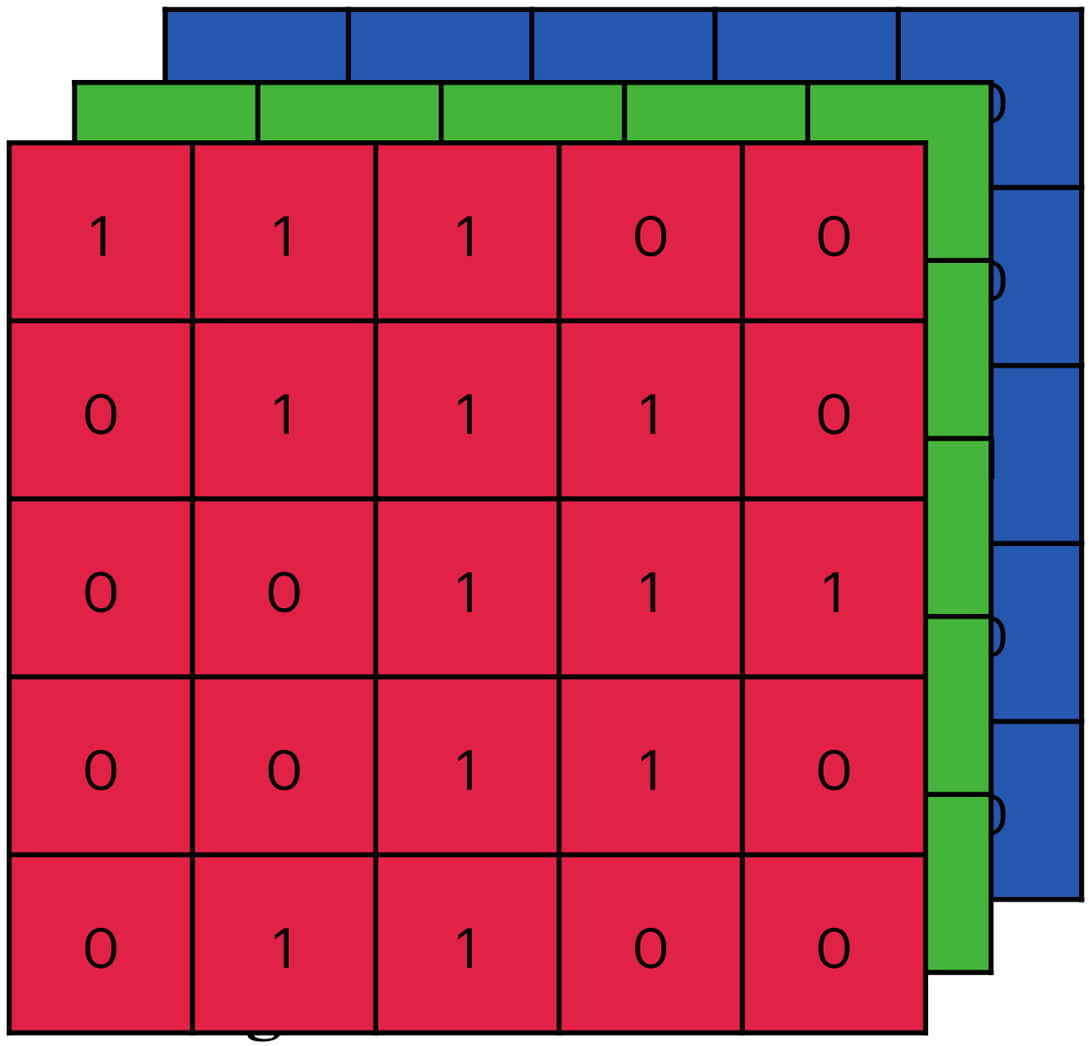


40 filtros de la 2ª capa convolucional

24 filtros de la 3ª capa convolucional

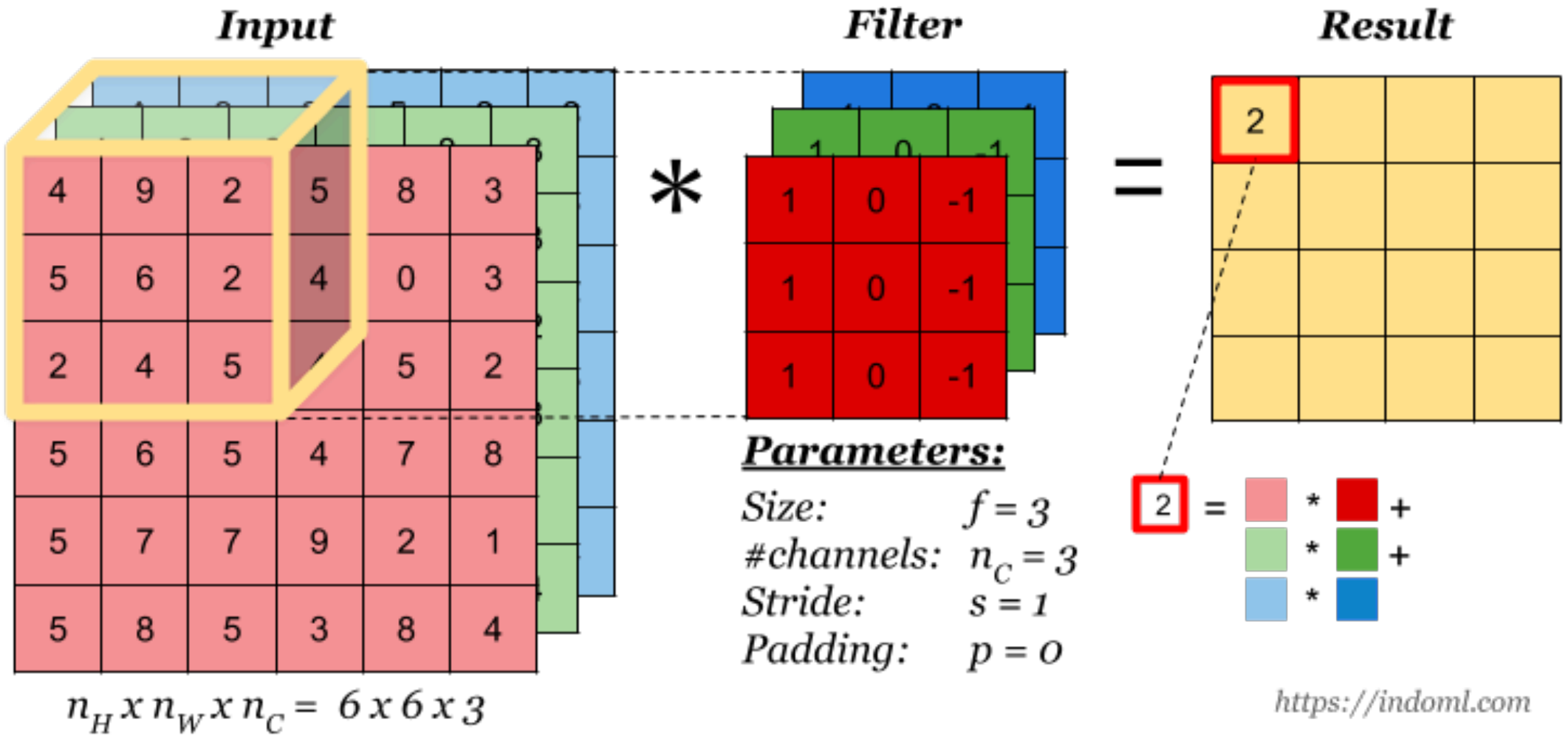
Capa convolucional. Filtros o Kernels

La imagen de entrada suele tener 1 (escala de grises) o 3 canales (imagen en color).



Capa convolucional. Filtros o Kernels

En el caso de 3 canales, aplicamos una convolución 2D sobre una matriz 3D.

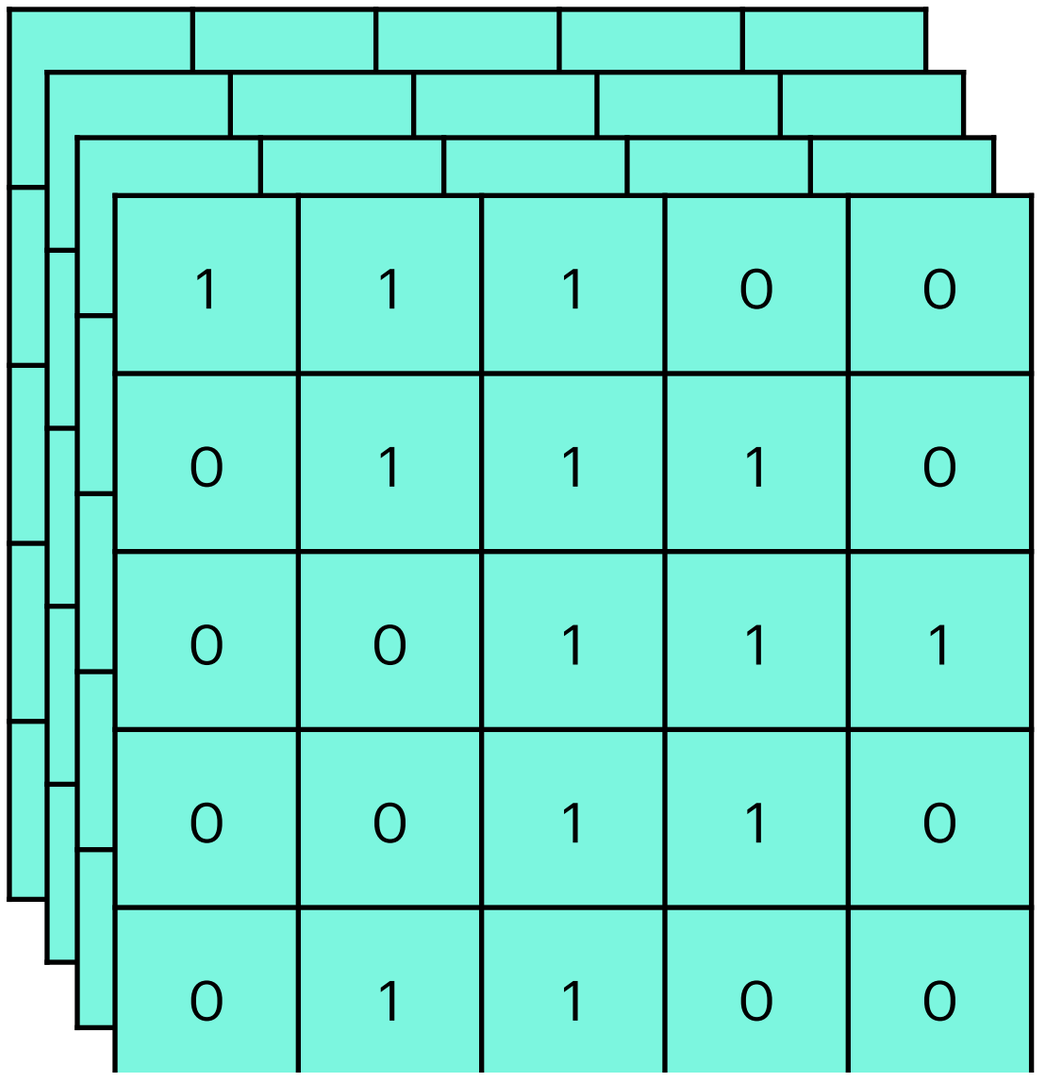


El filtro tiene 3 dimensiones. La salida es un mapa de características de sólo 2 dimensiones. Cada celda se calcula como la suma ponderada de aplicar cada uno de los 3 filtros.

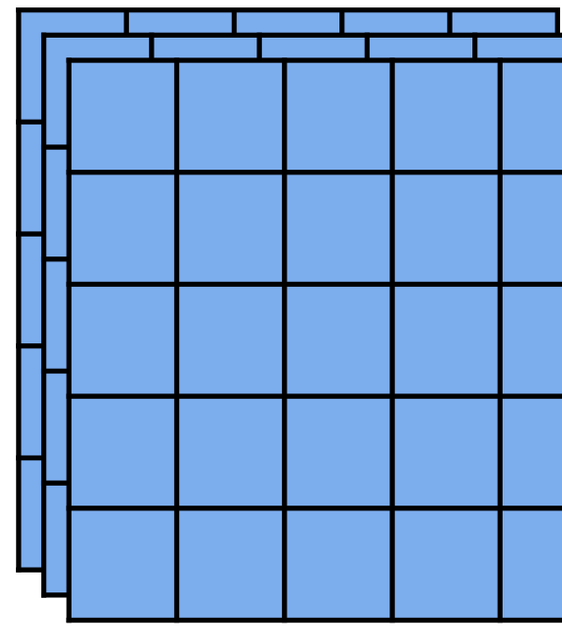
https://medium.com/analytics-vidhya/convolution-operations-in-cnn-deep-learning-compter-vision-128906ece7d3

Capa convolucional. Filtros o Kernels

De manera similar, cuando tenemos varios mapas de características de entrada, los filtros tendrán una tercera dimension con el mismo número de capas:



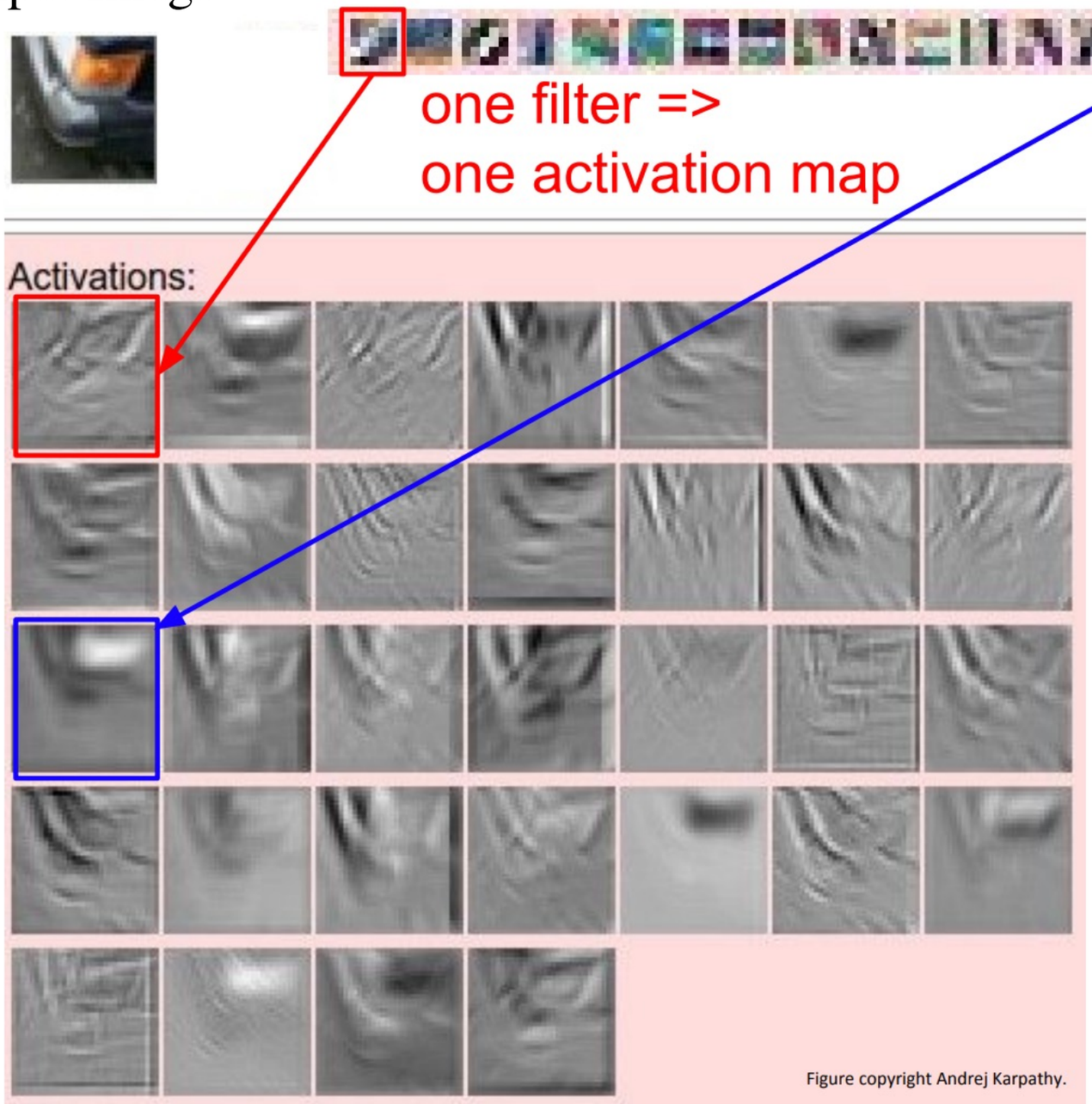
Ejemplo de 3 filtros con 4 capas cada uno para 4 mapas de características de entrada



Salida de 3 mapas de característi cas

Capa convolucional. Hiperparámetros

Input Image



Feature maps (o Activations)

Capa convolucional. Hiperparámetros

Hiperparámetros a definir al crear una capa convolucional:

- Tamaño del filtro (F)
- Número de filtros (K)
- Stride (S)
- Padding(P)

Tamaño de la salida (mapa de características) **(N_s, N_s, K):**

$$N_S = \left[\frac{N + 2P - F}{S}\right] + 1$$

Capa convolucional. Hiperparámetros

Hiperparámetros a definir al crear una capa convolucional:

- Tamaño del filtro (F)
- Número de filtros (K)
- Stride (S)
- Padding(P)

Tamaño de la salida (mapa de características) **(N_s, N_s, K):**

$$N_S = \left[\frac{N + 2P - F}{S}\right] + 1$$

Ejemplo:

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

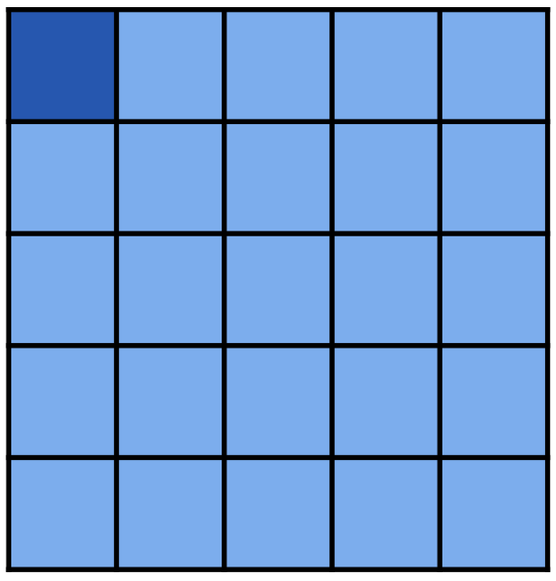
Padding=1

$$\left[\frac{5 + 2 * 1 - 3}{1}\right] + 1 = 5$$

Imagen de entrada

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$

Filtro o kernel



Mapa de características (Feature map)

Capa convolucional. Hiperparámetros

Hiperparámetros a definir al crear una capa convolucional:

- Tamaño del filtro (**F**)
- Número de filtros (**K**)
- Stride (**S**)
- Padding(**P**)

Tamaño de la salida (mapa de características) **(N_s, N_s, K):**

$$N_S = \left[\frac{N + 2P - F}{S}\right] + 1$$

K F S P

```
conv = Conv2D(filters=64, kernel_size=(3, 3), strides=(1,1), padding='same')
```

Keras

Configuraciones comunes:

K —> Múltiplos de 2

F = 3, S = 1, P = 1

F = 5, S = 1, P = 2

F = 5, S = 2, P = same

F = 1, S = 1, P = 0

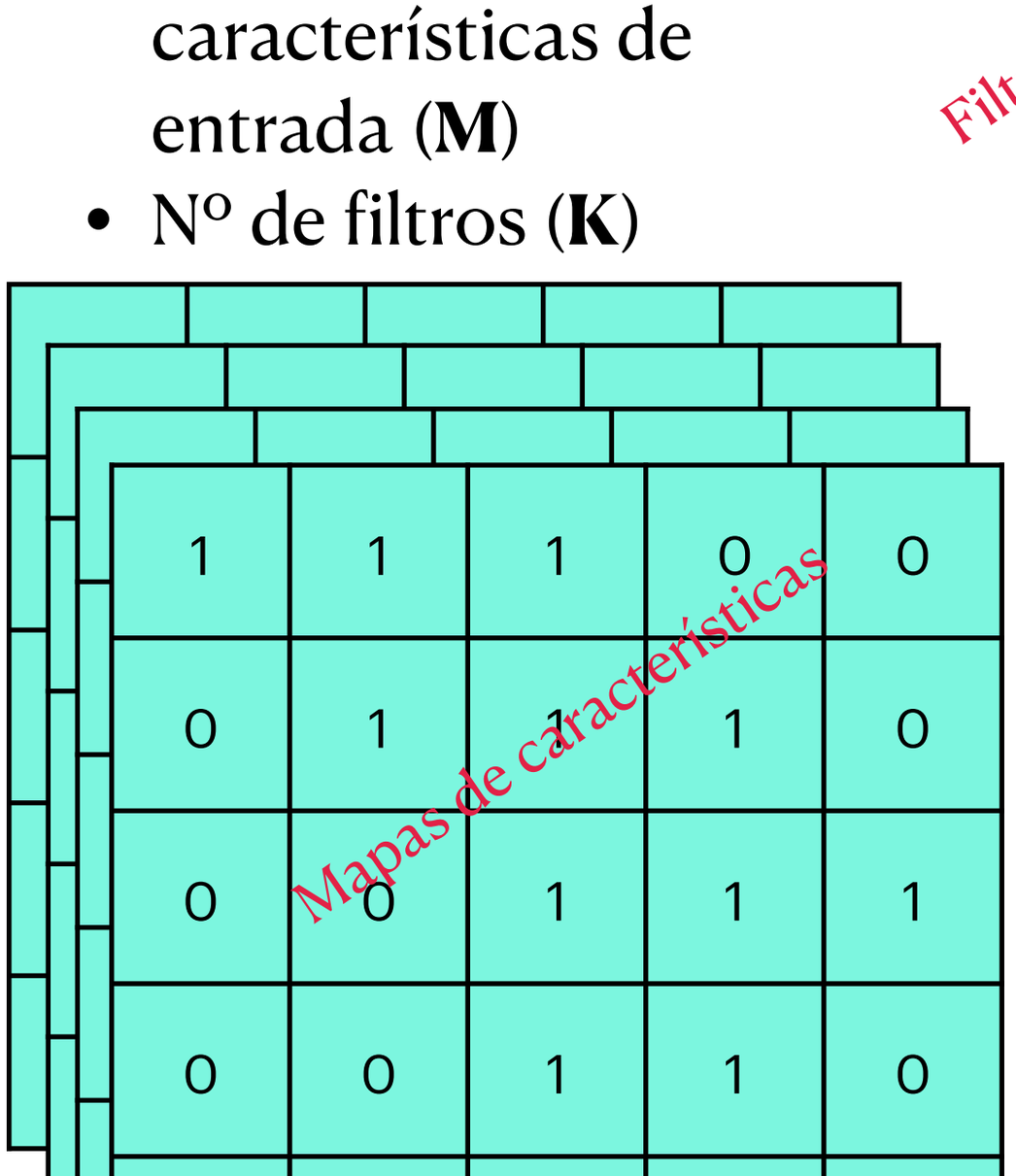
Capa convolucional. Nº de parámetros

Los parámetros de una capa convolucional son los pesos que de los filtros + el bias

- Tamaño del filtro (**F**)
- Número mapas de características de entrada (**M**)
- Nº de filtros (**K**)

$$params = (F * F * M + 1) * K$$

$$params = (3 * 3 * 4 + 1) * 3 = 111$$

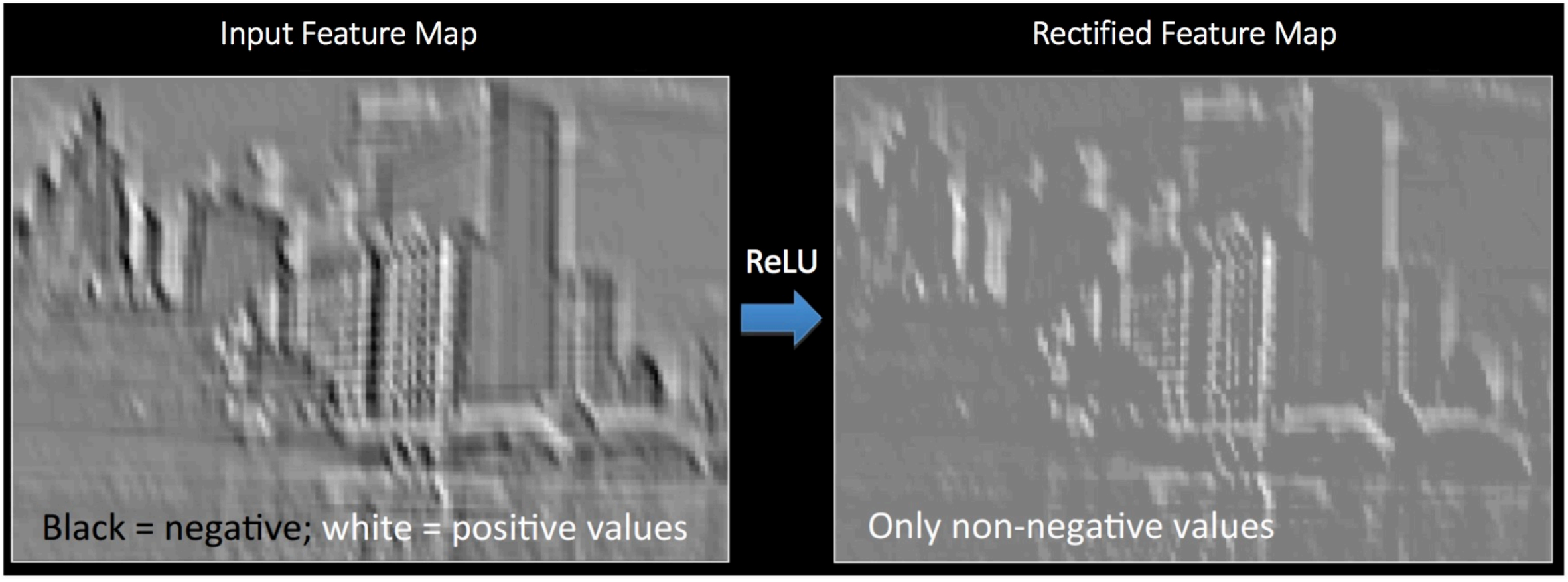


Capa convolucional. Función de activación

- Se introducen no linealidades
- Se aplica tras cada filtro convolucional
- La más común es ReLU (Rectified Linear Unit). Reemplaza todos los píxeles negativos por cero.

```
act = Activation('relu')(conv)
```

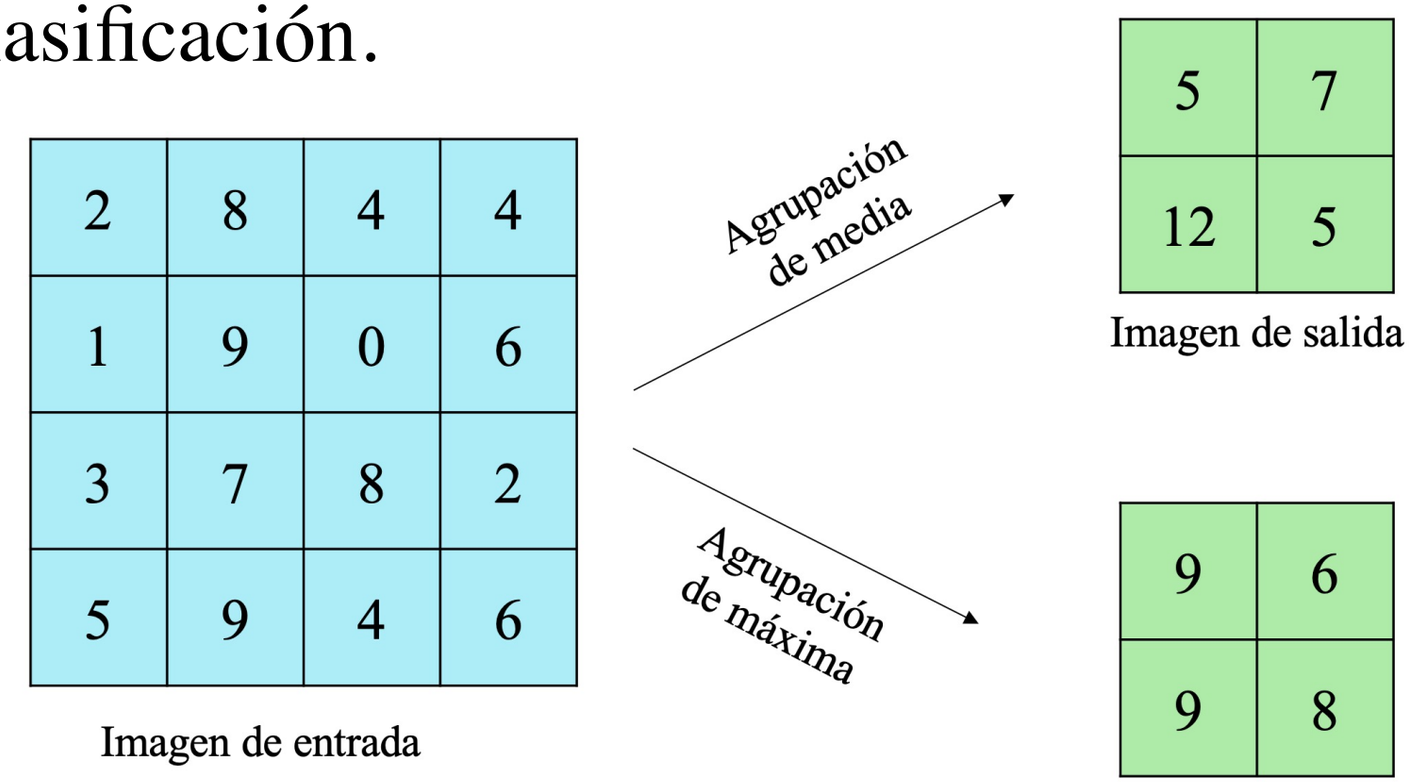
Keras



Capa de pooling o subsampling.

Capa para **reducir el tamaño** de los mapas de características preservando la información más importante.

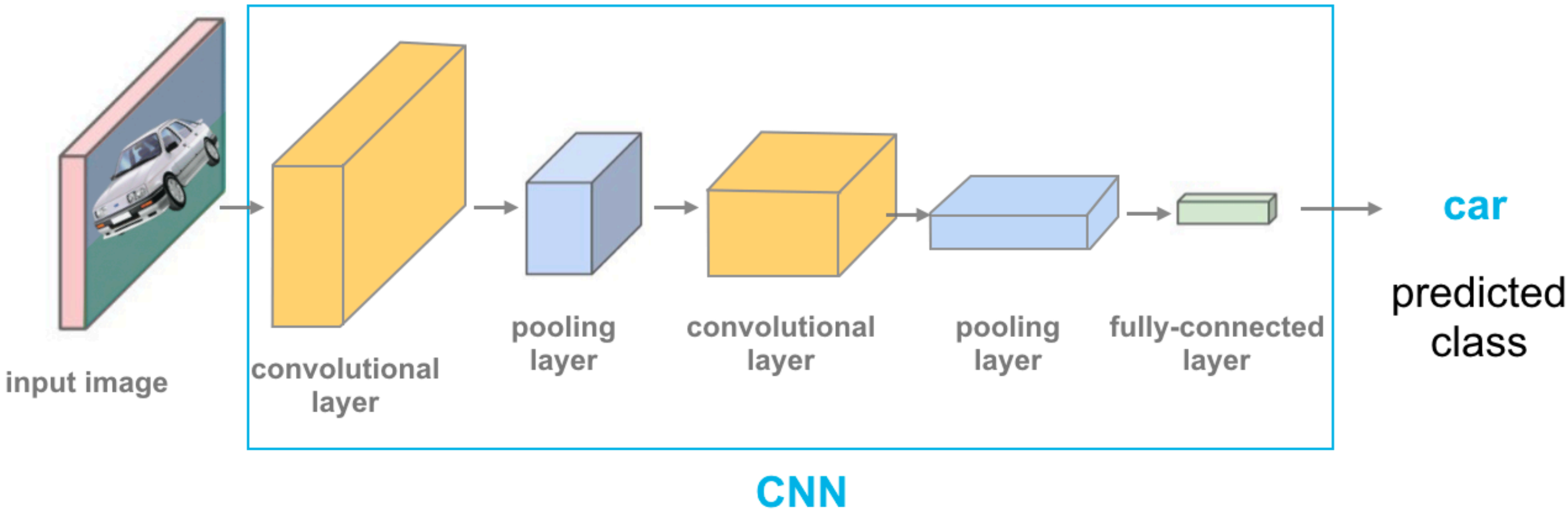
Reduciendo el tamaño de la capa, aún se mantienen elementos estructurales importantes, sin contener el detalle fino que puede que no sea útil para la clasificación.



Keras

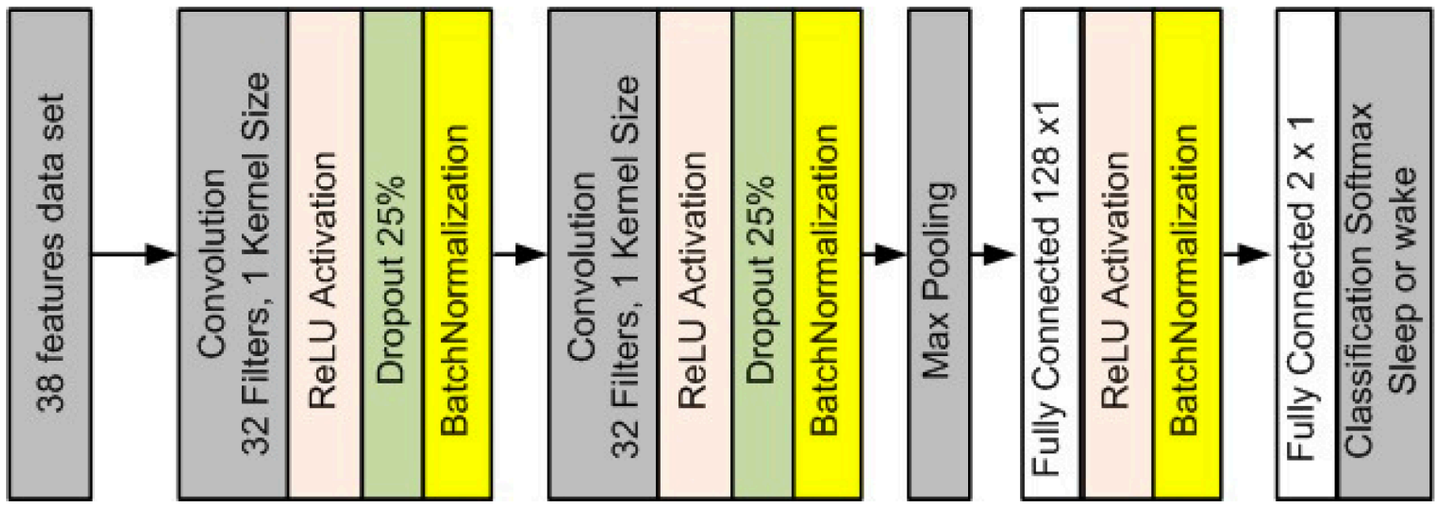
```
pool = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(act)
```

Capa de pooling o subsampling.



Capa de Batch Normalization.

Funciona de manera similar a las redes densamente conectadas, pero en este caso, en lugar de hacerlo sobre un mini-batch, se realiza sobre toda a capa ya que se aplica el mismo filtro a todos los valores de entrada.

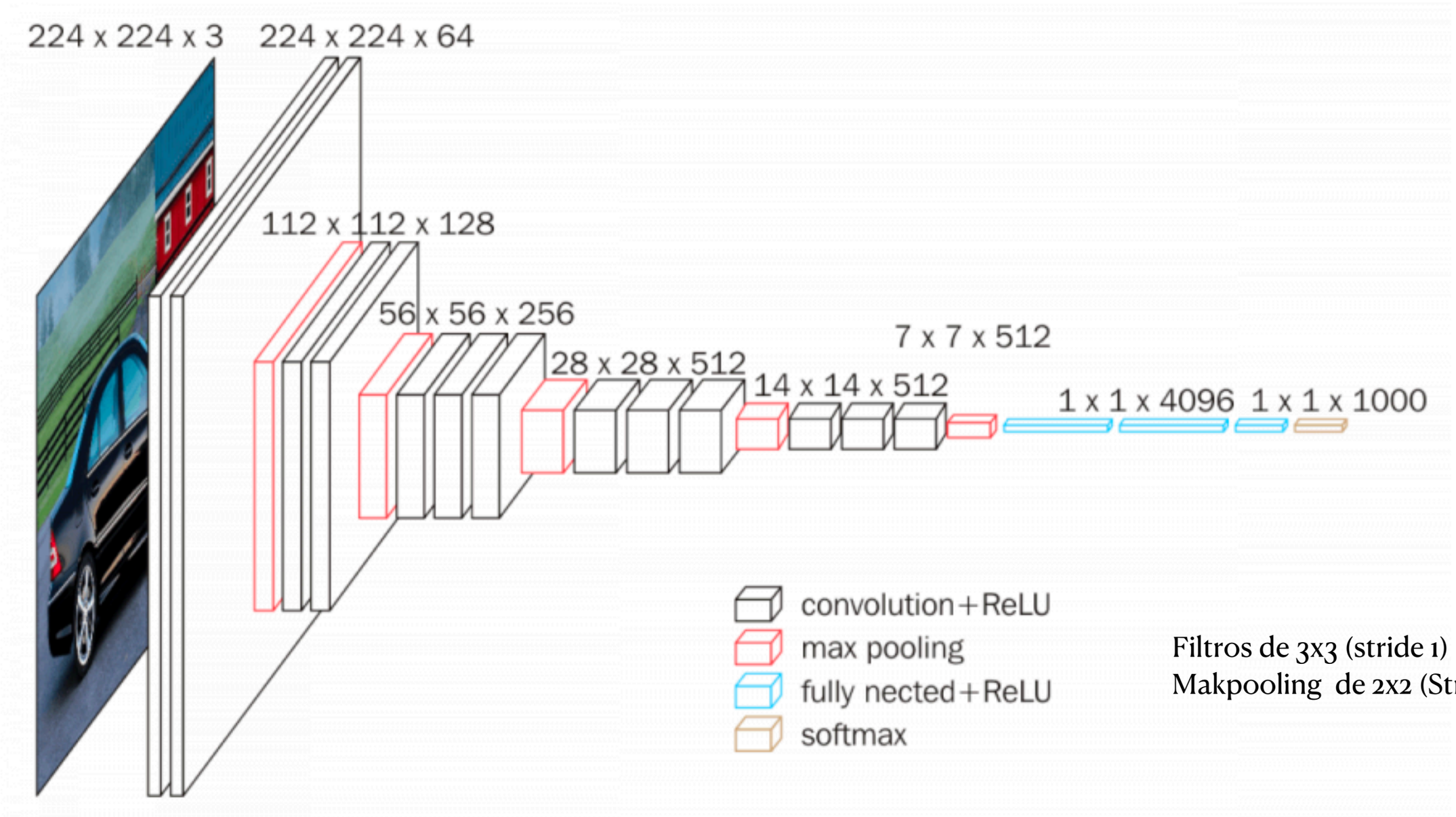


Esta capa ha demostrado grandes resultados en la mejora de la generalización de los modelos y se basa en normalizar la salida del filtro convolucional empleando la media y desviación estándar de ésta.

EJEMPLO

```
model = Sequential([
  Conv2D(32, (3,3), input_shape=(28, 28, 3) activation='relu'),
  BatchNormalization(),
  Conv2D(32, (3,3), activation='relu'),
  BatchNormalization(),
  MaxPooling2D(),
  Dense(2, activation='softmax')
])
```

VGG16 – Convolutional Network for Classification and Detection

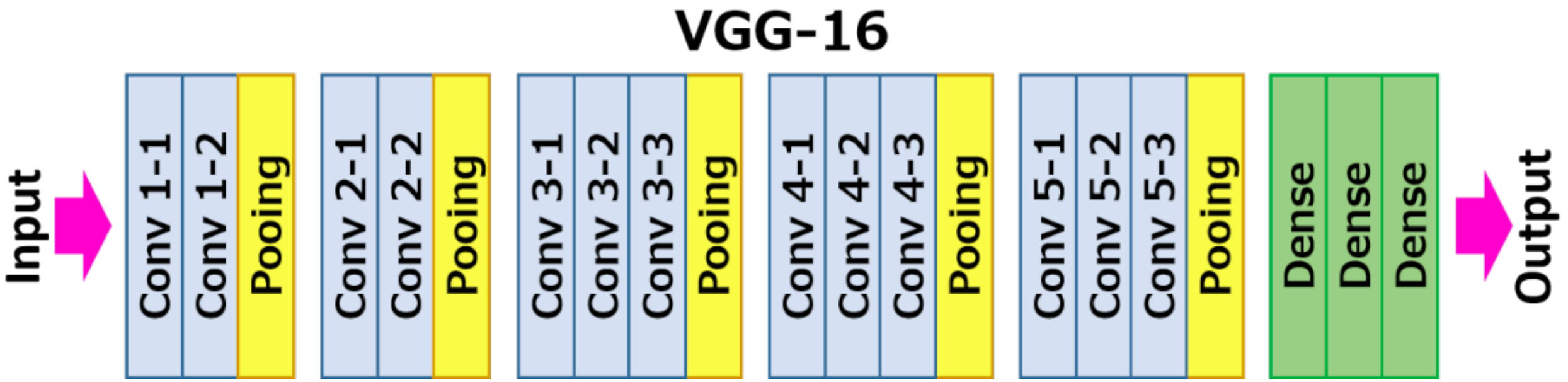


Filtros de 3x3 (stride 1)
Makpooling de 2x2 (Stride 2)

VGG16 se entrenó durante semanas usando NVIDIA Titan Black GPU. Sobre el dataset ImageNet, que es un conjunto de datos de más de 14 millones de imágenes que pertenecen a 1000 clases.

https://neurohive.io/en/popular-networks/vgg16/

VGG16 – Convolutional Network for Classification and Detection

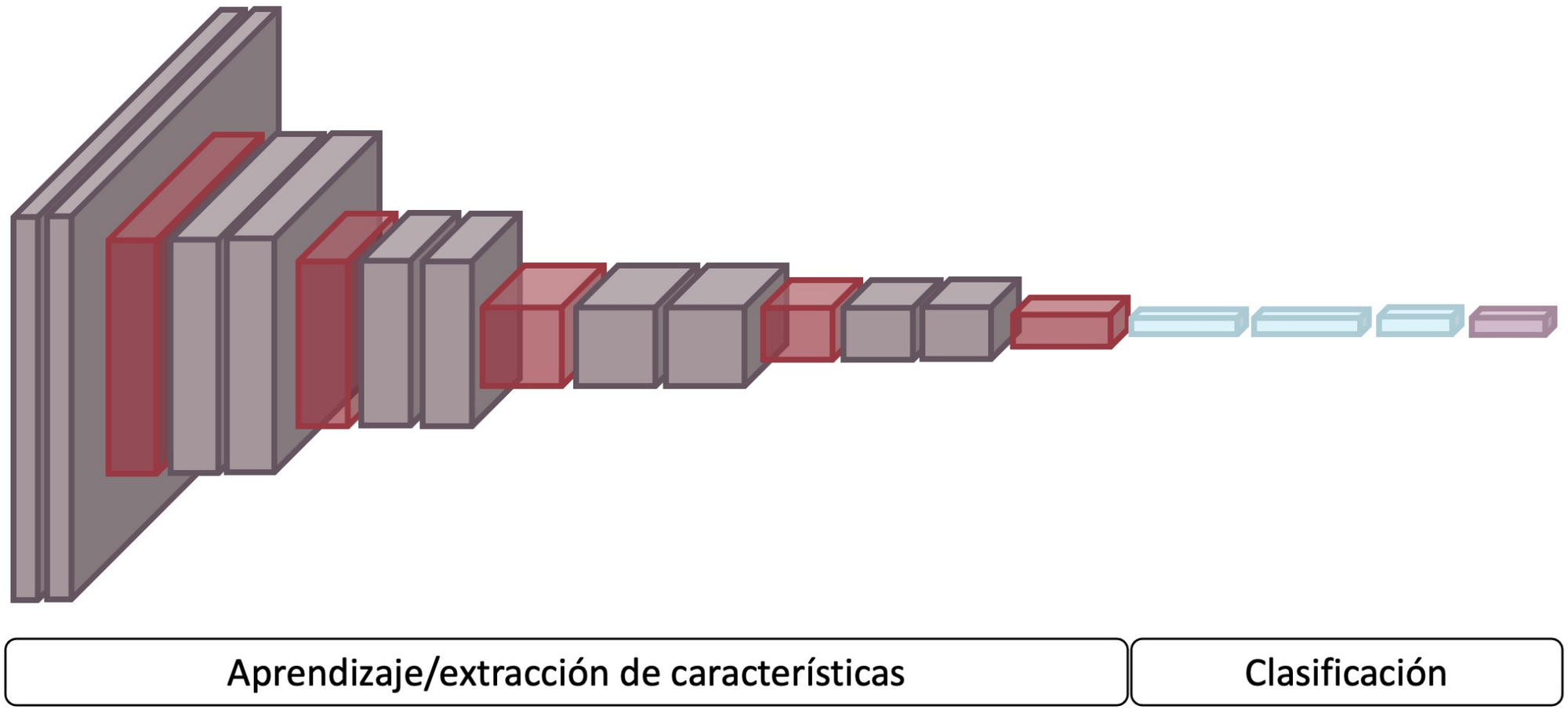


Filtros de 3x3 (stride 1)
Makpooling de 2x2 (Stride 2)

VGG16 se entrenó durante semanas usando NVIDIA Titan Black GPU. Sobre el dataset ImageNet, que es un conjunto de datos de más de 14 millones de imágenes que pertenecen a 1000 clases.

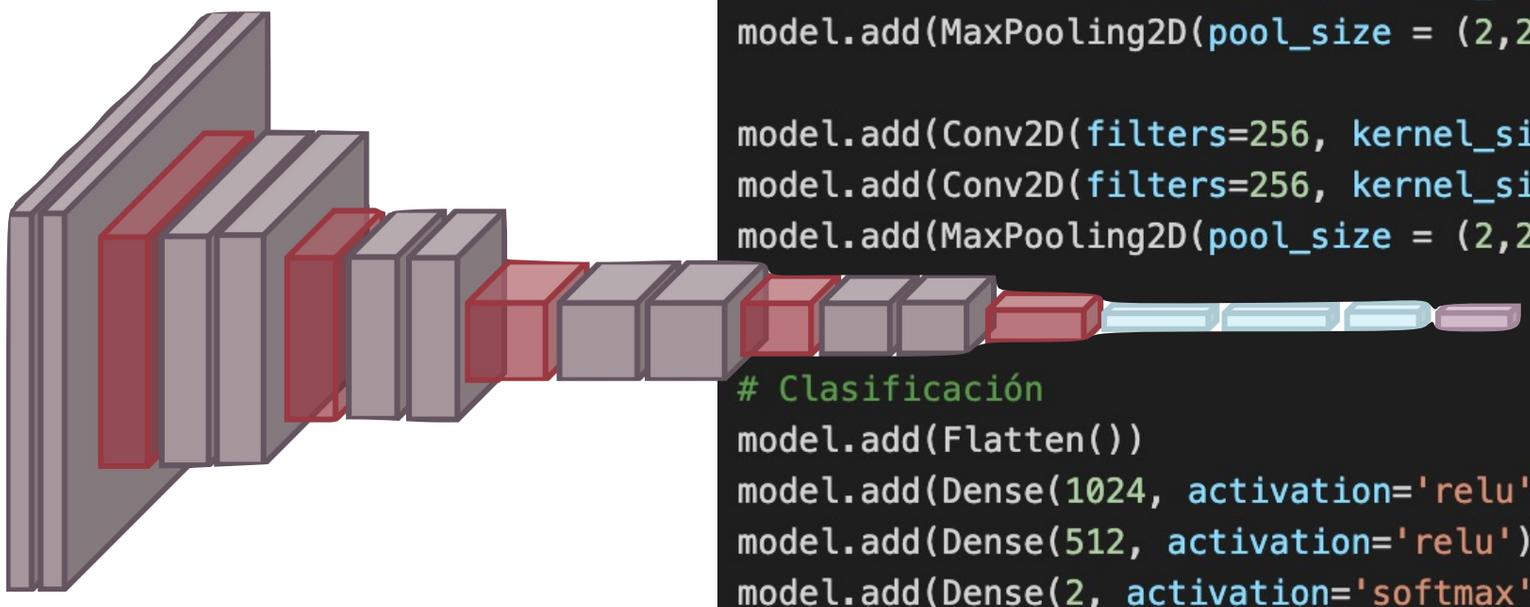
https://neurohive.io/en/popular-networks/vgg16/

Ejemplo en Keras (TensorFlow)



Ejemplo en Keras (TensorFlow)

Keras



```python
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from keras.models import Sequential

# Extraccion caracteristicas
model = Sequential()
model.add(Input(shape=(img_rows, img_cols, img_channels)))
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2), strides = (2,2)))

model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2), strides = (2,2)))

model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2), strides = (2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size = (2,2), strides = (2,2)))

# Clasificación
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(2, activation='softmax')) # 2 salidas
```