Aprendizaje Automático Profundo

Clase 7 - 2019

Profs: Franco Ronchetti - Facundo Quiroga

FILTROS CONVOLUCIONALES

Resumen

Hasta ahora…

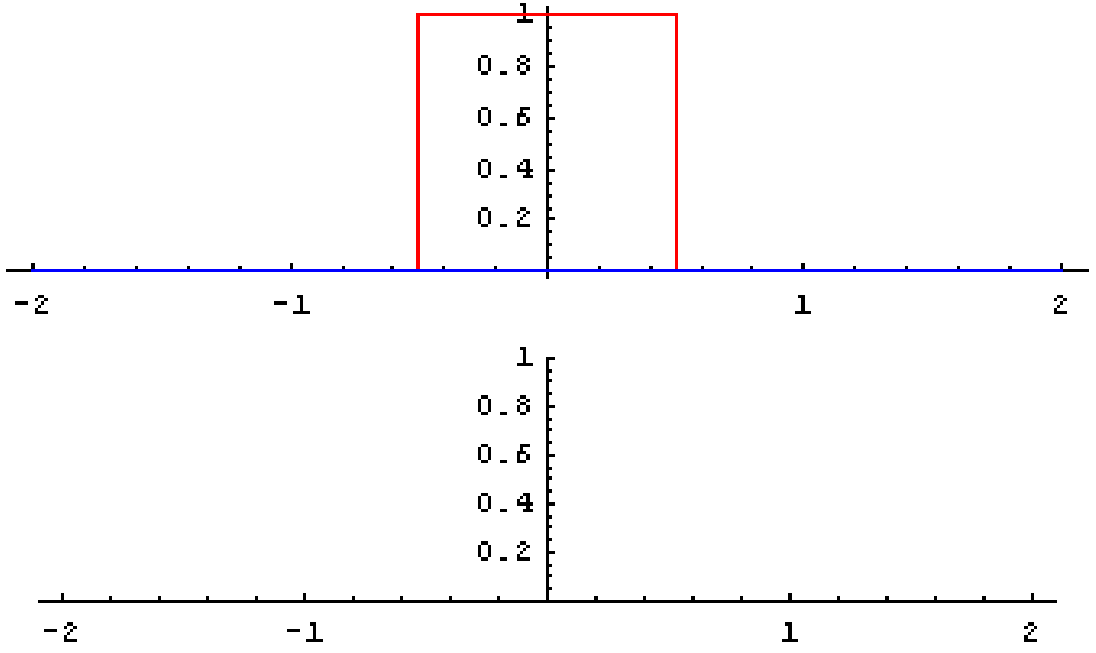
- Vimos un modelo de clasificación/regresión con las siguientes propiedades:
  - Regresor Logístico (Lineal): 1 capa de salida
  - Redes neuronales (no lineal): al menos una capa oculta + 1 capa de salida
  - Siempre tantas neuronas de salida como clases a clasificar.
- Tipos de problemas:
  - 1 o 2 Features: podemos graficar los datos y ver las fronteras de decisión.
  - Imágenes: es un caso particular de N-features donde podemos interpretar los datos visualmente. Tienen información espacial que se debe aplanar para darle a un modelo neuronal.
- Métricas
  - Train set: para entrenar. Test set: para validar el modelo con nuevos datos.
  - Accuracy: nos dice como funciona el modelo de forma global.
  - Precision/Recall: lo usamos para clasificación binaria. Explica mejor cómo detecta los datos positivos.

Convolución

Una convolución es una operación sobre dos funciones *f* y *g*, que produce una tercera función que puede ser interpretada como una versión "filtrada" de *f*. En funciones unidimensionales se utiliza para realizar diferentes filtros en señales o modelar el comportamiento de ciertos estímulos en simulaciones.

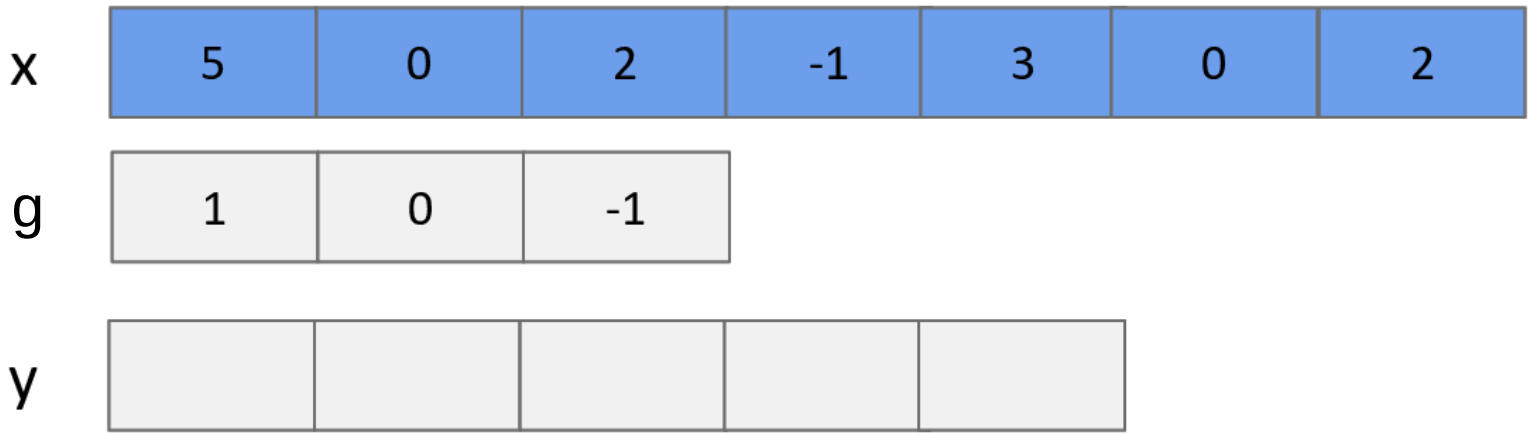
Si bien la convolución se define en forma continua, a nosotros nos interesa la versión discreta.

$$f[x] * g[x] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[x-k]$$



Convolución 1D

$$f[x] * g[x] = \sum_{k=-\infty}^{\infty} f[k] \cdot g[x-k]$$



Al aplicar un filtro (kernel) de forma discreta, es necesario definir algunos parámetros:

Kernel_Size= 3

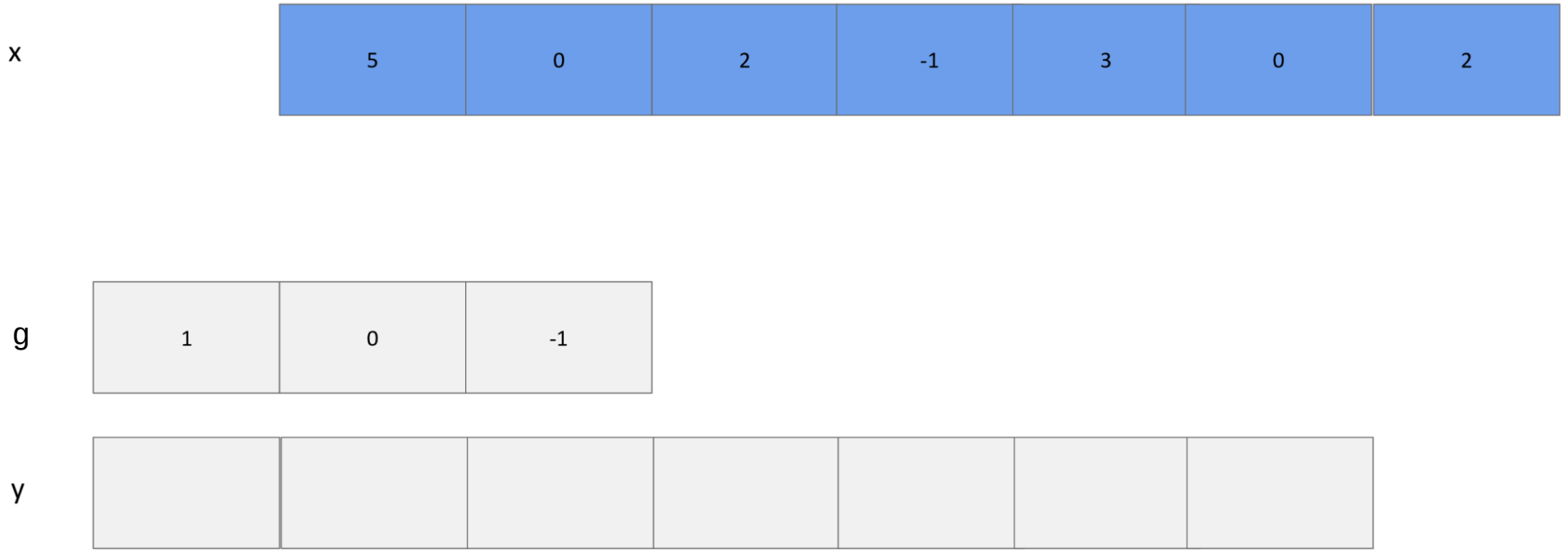
Stride= 1

Convolución - Padding

Aplicar el filtro de forma discreta ocasiona dos problemas:

- Pérdida de información en los bordes.
- Reducción del tamaño final del vector.

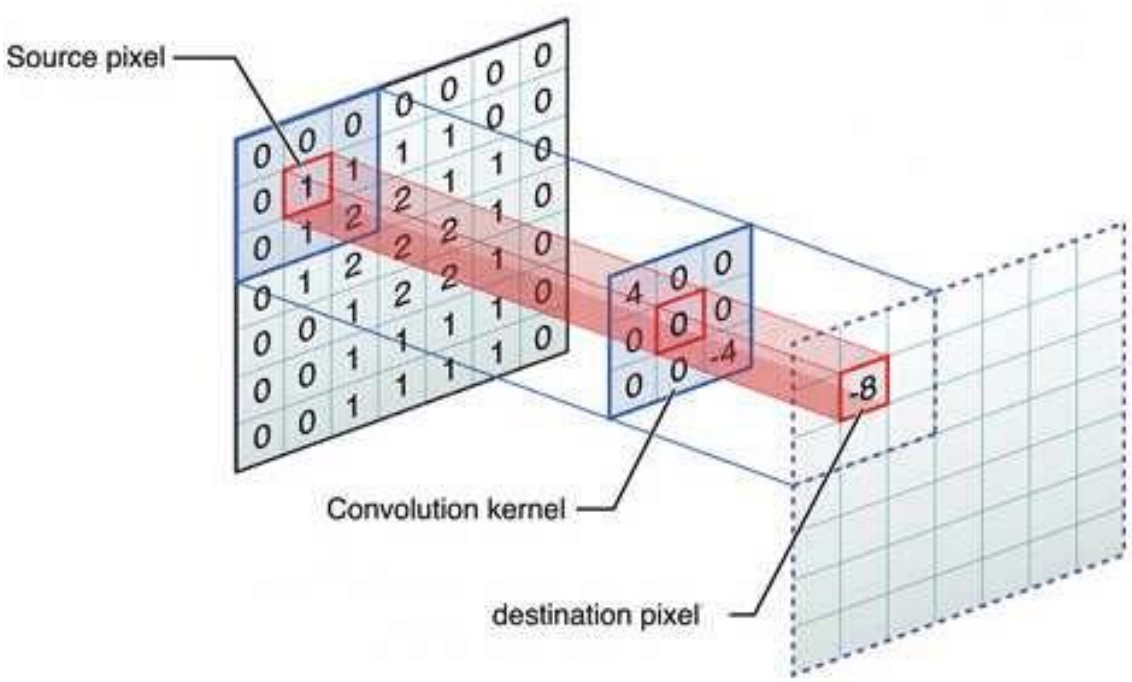
Para solucionar esto se suele utilizar la técnica de “padding”, generalmente rellenando con ceros.



Convolución 2D

Siguiendo la misma idea, podemos extender el concepto de convolución sobre matrices. Es decir, una convolución en 2 dimensiones.

Esto nos sirve para imágenes en escala de grises. Algo usual es utilizar *zero* padding para conservar el tamaño y no perder la información de los bordes.



$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

Convolución 2D

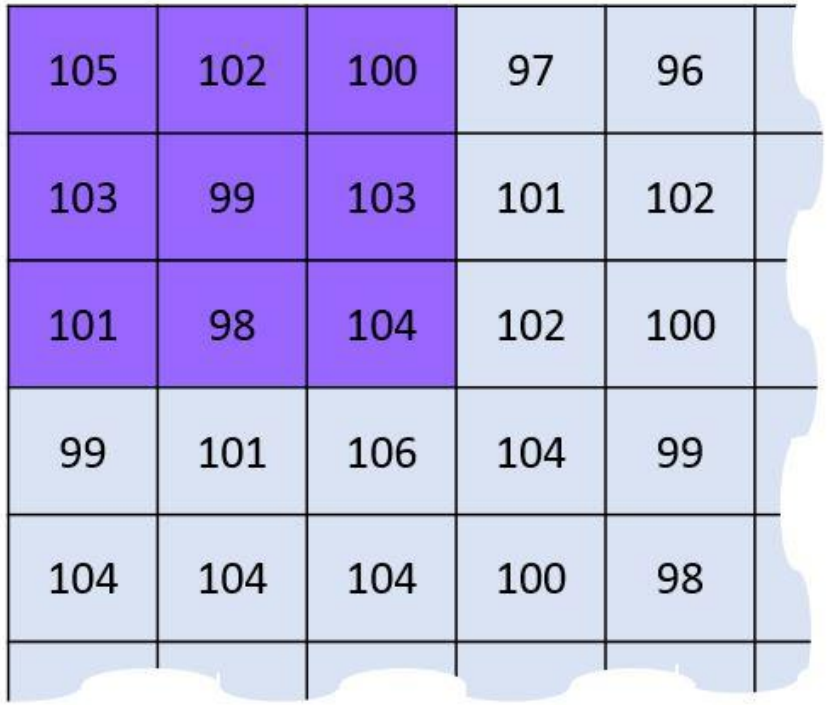
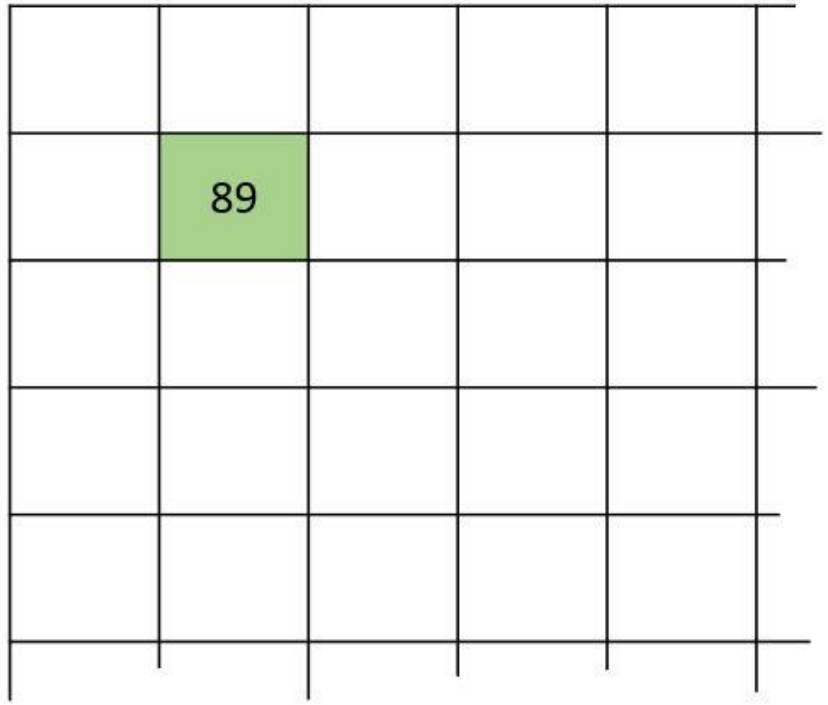


Image Matrix

Kernel Matrix

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

Kernel size= 3
Stride =1



Output Matrix

$$105 * 0 + 102 * -1 + 100 * 0 + 103 * -1 + 99 * 5 + 103 * -1 + 101 * 0 + 98 * -1 + 104 * 0 = 89$$

Convolución 2D

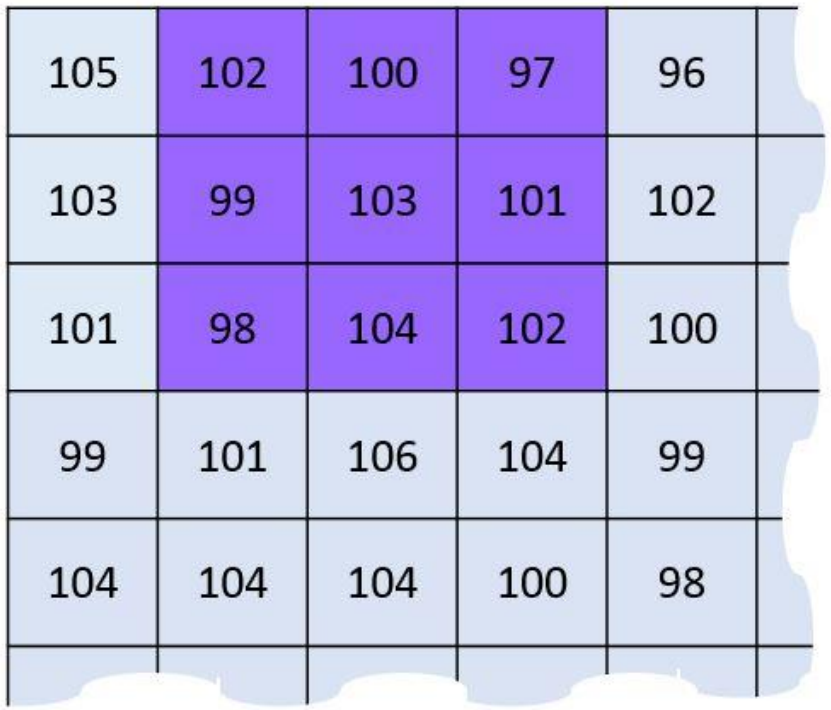
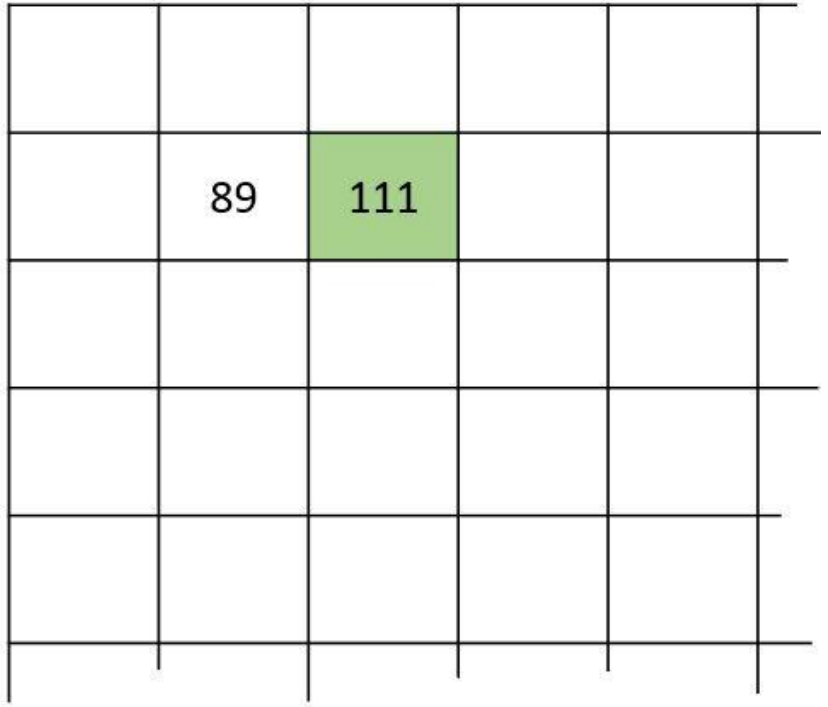


Image Matrix

Kernel Matrix

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

Kernel size= 3
Stride =1



Output Matrix

$$102 * 0 + 100 * -1 + 97 * 0$$
$$+99 * -1 + 103 * 5 + 101 * -1$$
$$+98 * 0 + 104 * -1 + 102 * 0 = 111$$

Convolución 2D

```
Kernel size= 3
Stride =1
Zero Padding
```

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

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

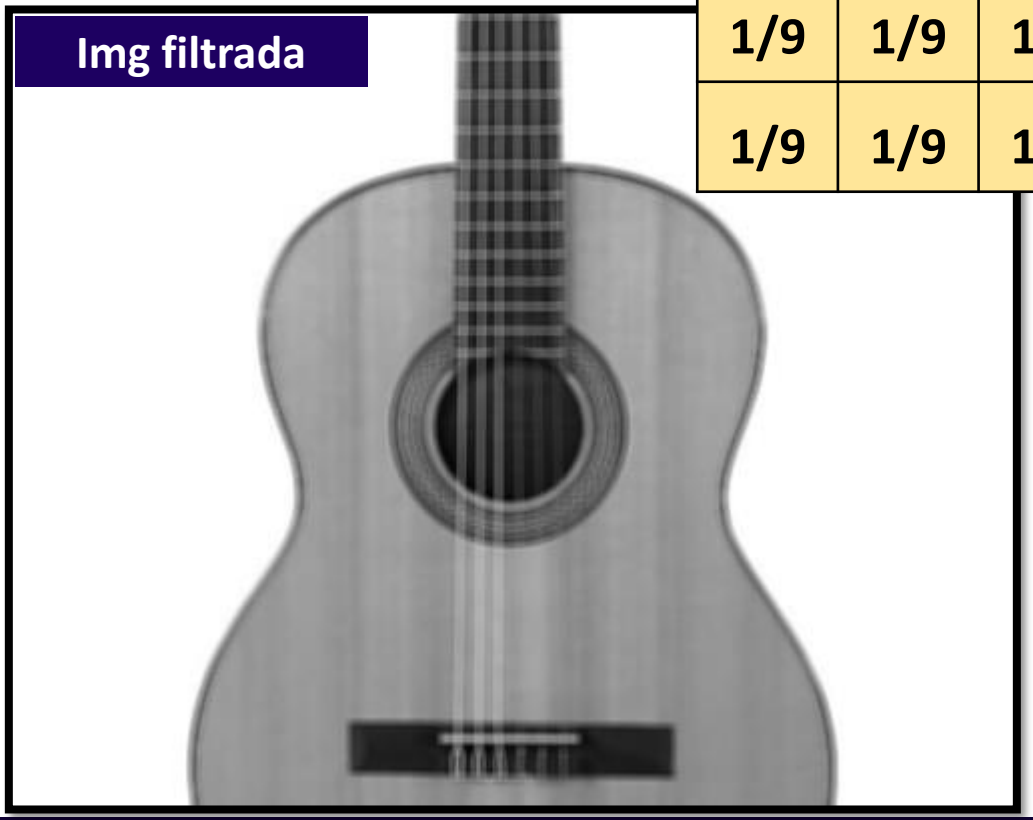
114				

Convolución 2D

Veamos ahora cuál es el efecto de aplicar algunos *kernels* clásicos.

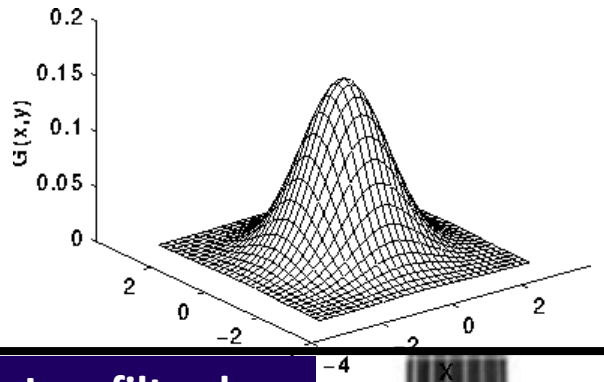
Simple promedio de todos los píxeles: borronea la imagen.

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

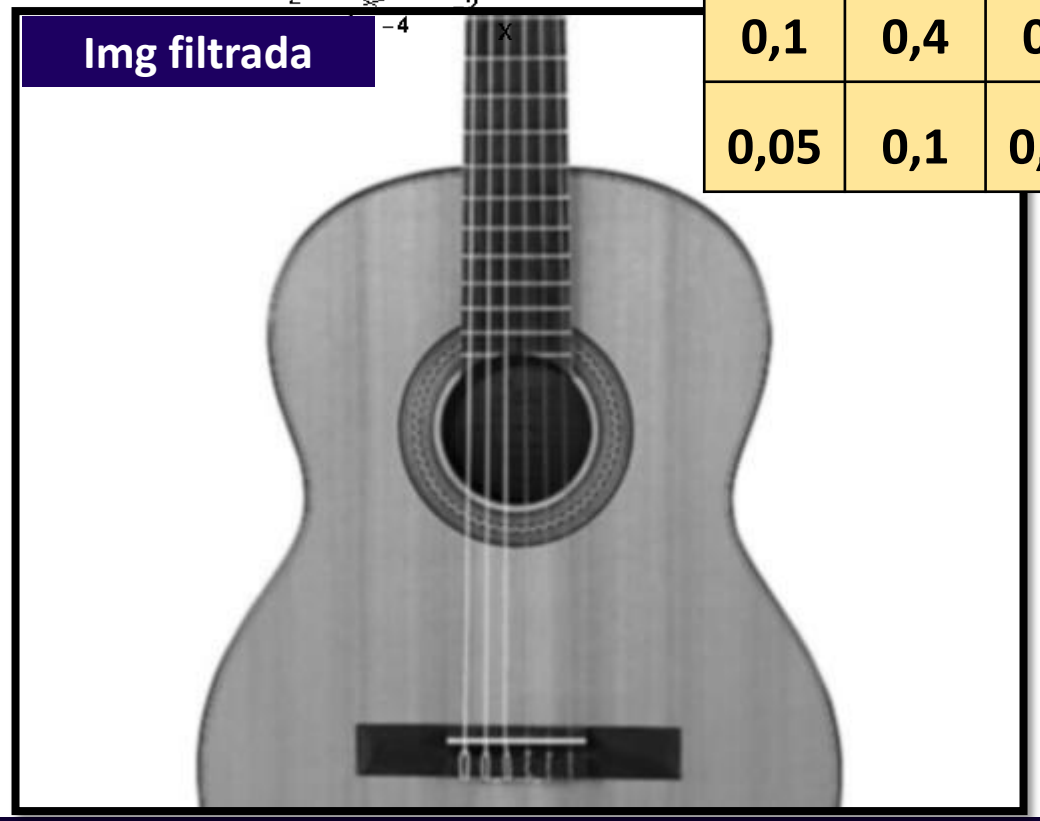


Convolución 2D

Filtro gaussiano (filtro de pasa baja)



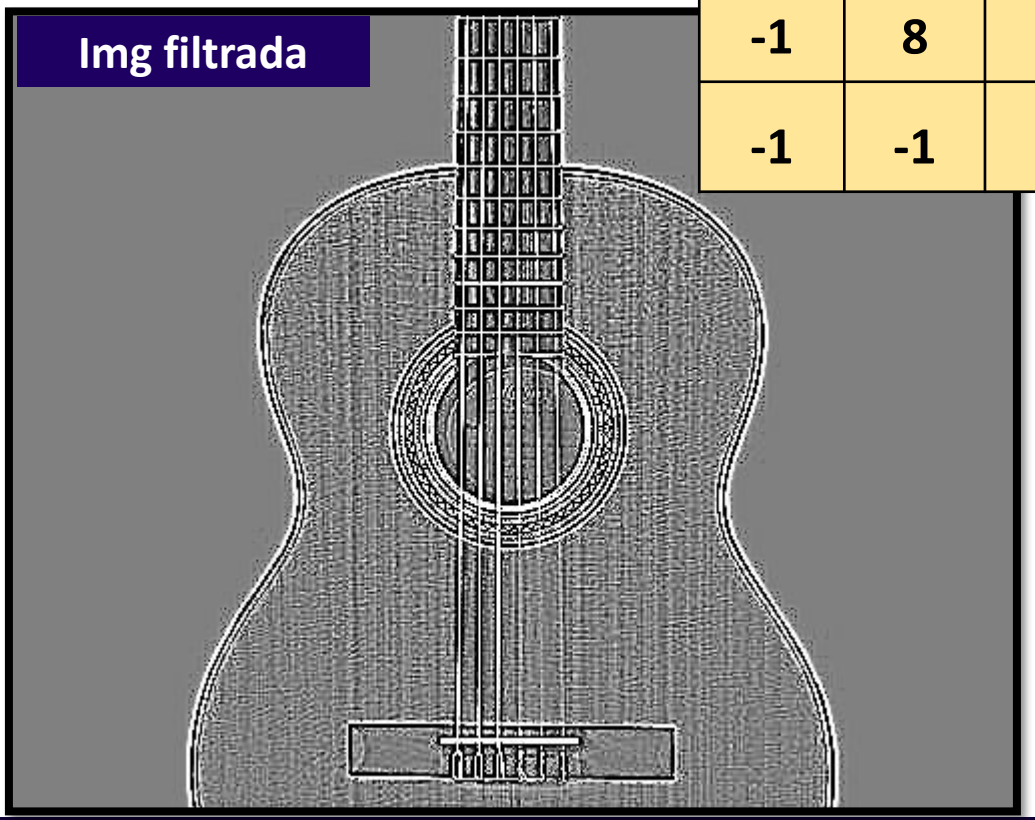
0,05	0,1	0,05
0,1	0,4	0,1
0,05	0,1	0,05



Convolución 2D

Detección de bordes (filtro de pasa alta)

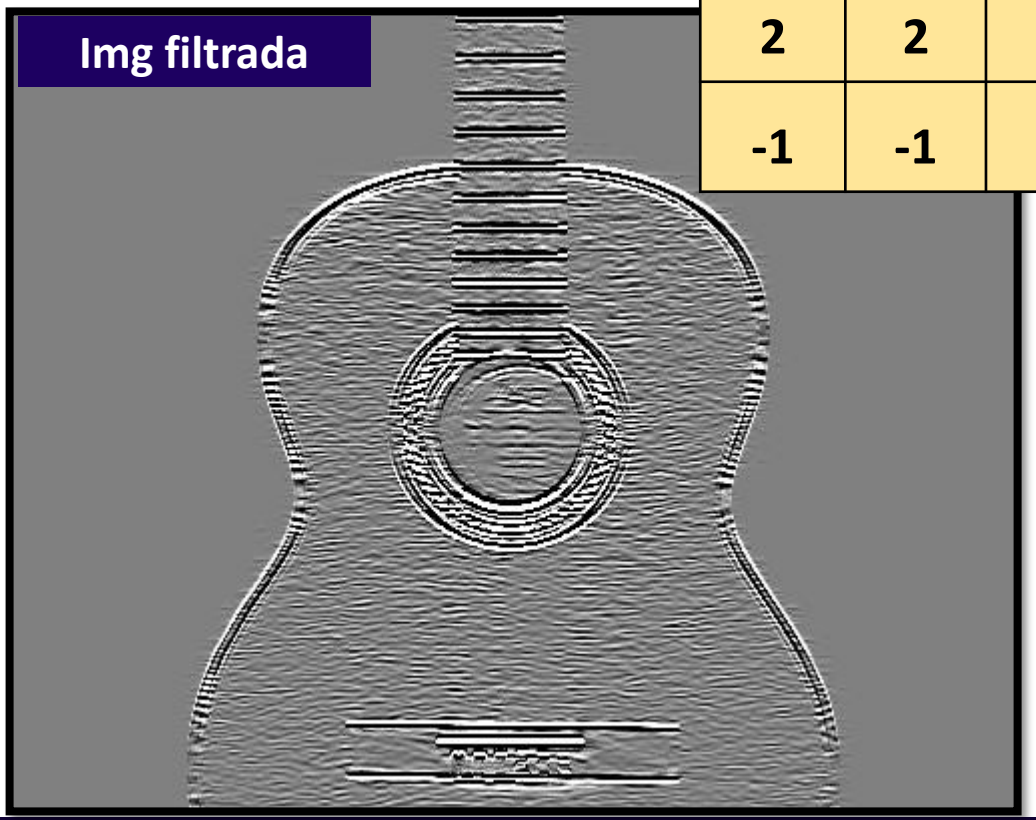
-1	-1	-1
-1	8	-1
-1	-1	-1



Convolución 2D

Bordes horizontales

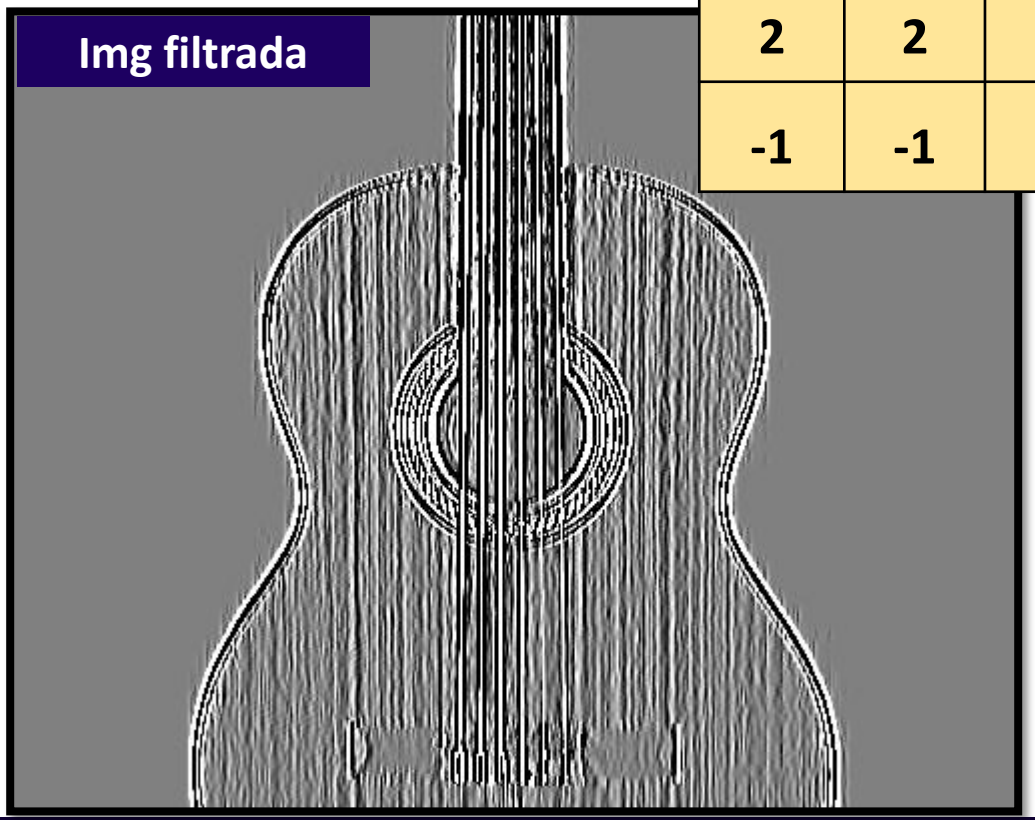
-1	-1	-1
2	2	2
-1	-1	-1



Convolución 2D

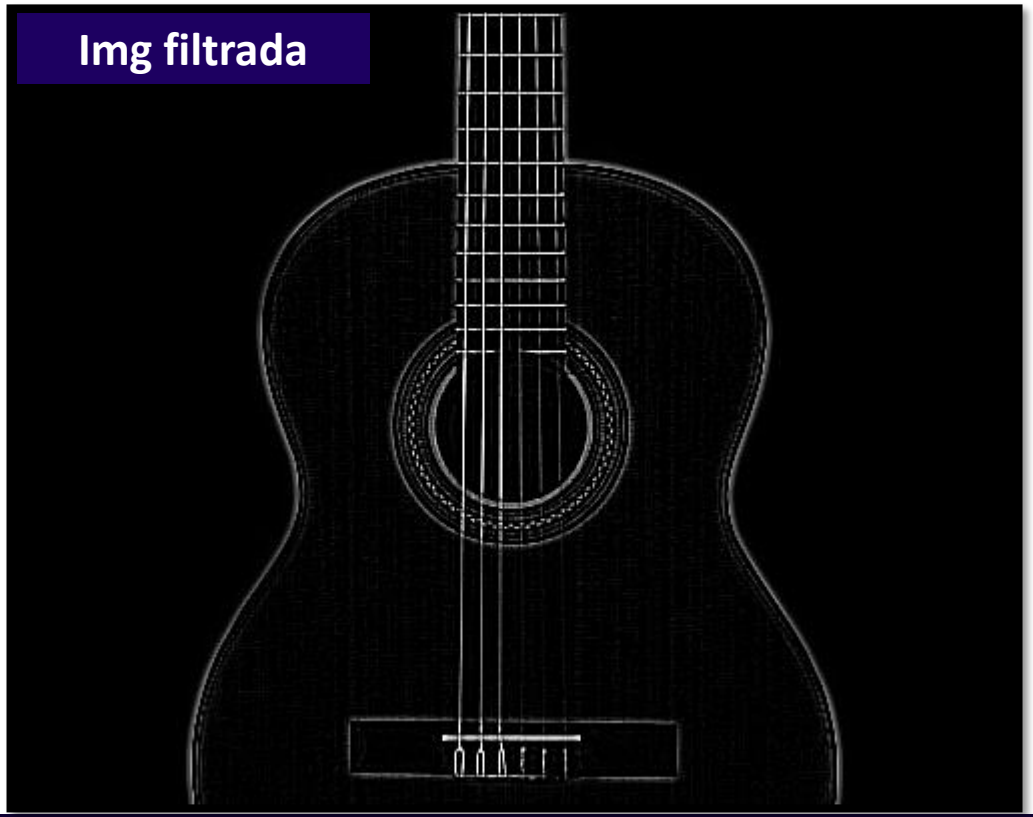
Bordes verticales

-1	-1	-1
2	2	2
-1	-1	-1



Convolución 2D

Kernel Gaussiano + Kernel Bordes

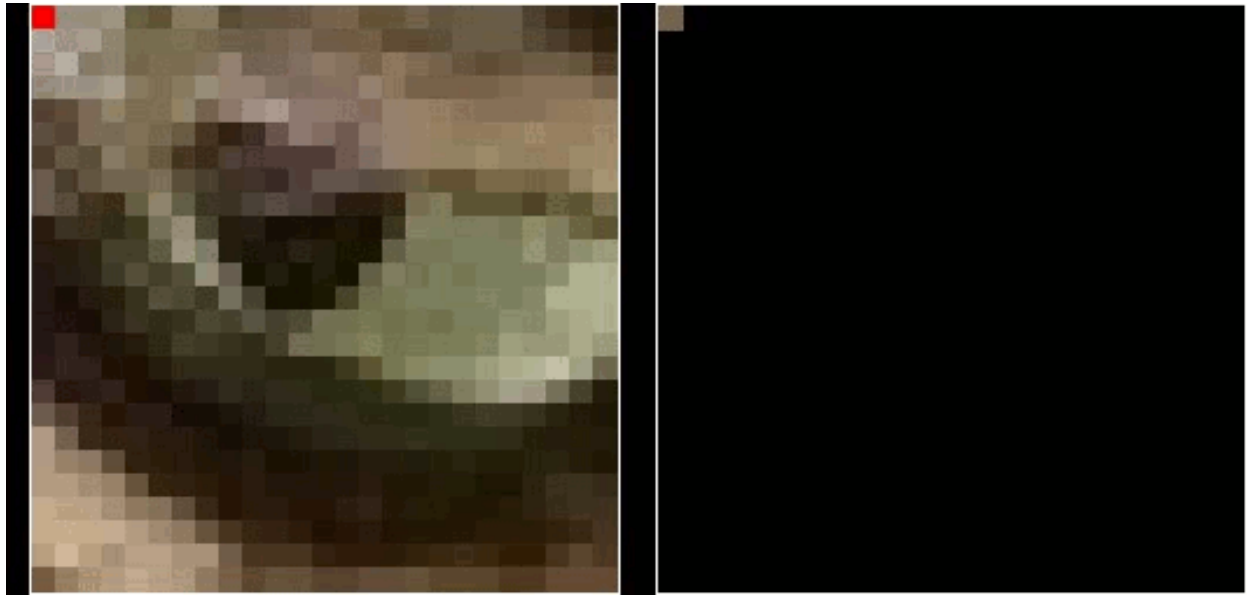


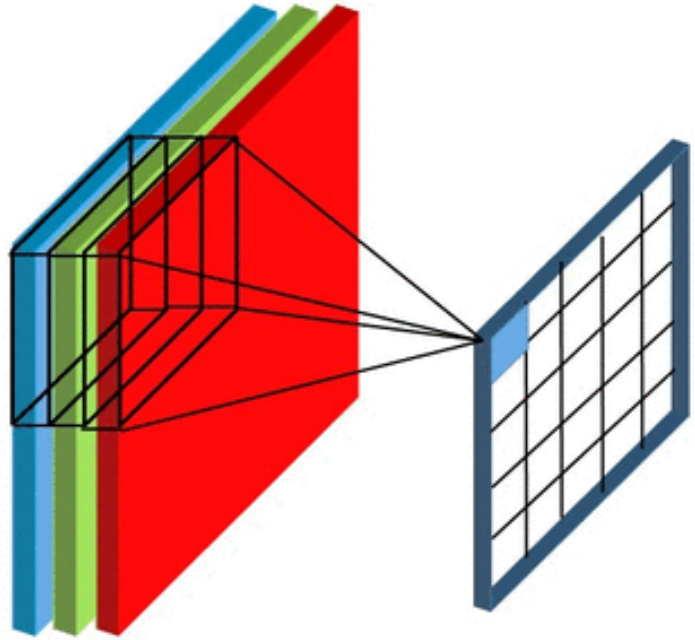
Convolución 2D sobre datos 3D - ND

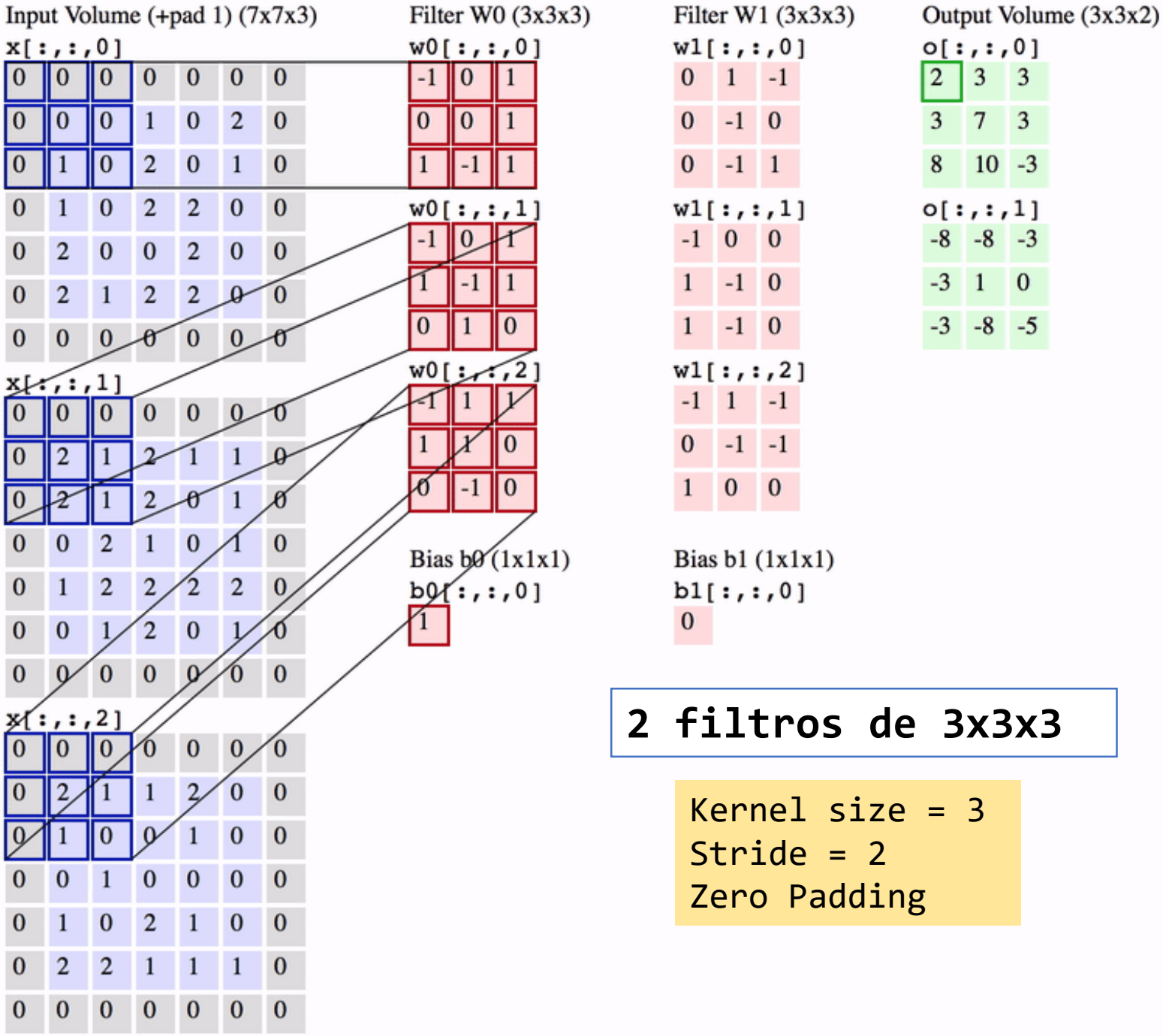
En imágenes RGB, nuestro kernel deberá tener una dimensión más:

`Kernel_Size= KxKx3`

La convolución sigue siendo 2D pero se realiza sobre los 3 canales a la vez.







2 filtros de 3x3x3

Kernel size = 3
Stride = 2
Zero Padding

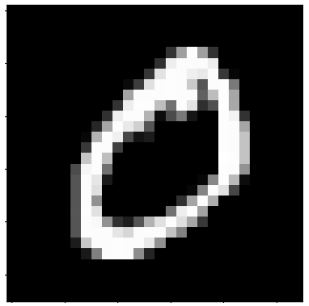
REDES NEURONALES CONVOLUCIONALES

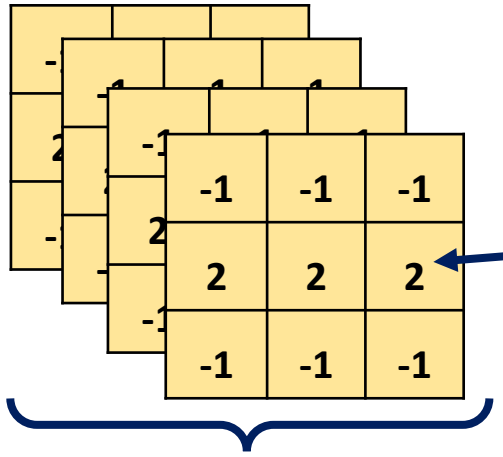
Capa convolucional

Una capa convolucional no es más que muchos filtros convolucionales de tamaño **K*K*C** (donde K se debe definir y C depende de la cantidad de canales de la imagen).

```
Conv2D(cant_filtros,
       kernel_size= k,
       strides= (n,m)
       activation='relu',
       padding = 'same')
```

En Keras





Estos valores son parte de los "pesos" de nuestra red.

Capa convolucional de 4 filtros de 3x3x1

Capa convolucional en MNIST

Veamos cómo sería aplicar una simple capa convolucional en el dataset MNIST.

Accuracy alimentando a la red Feed-Forward con la imagen cruda.

```
Train
   Accuracy: 0.93    soporte: 60000

Test
   Accuracy: 0.93    soporte: 10000
```

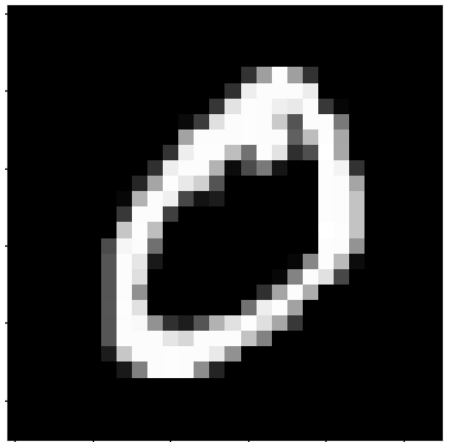
Accuracy al agregar una capa convolucional de 64 filtros.

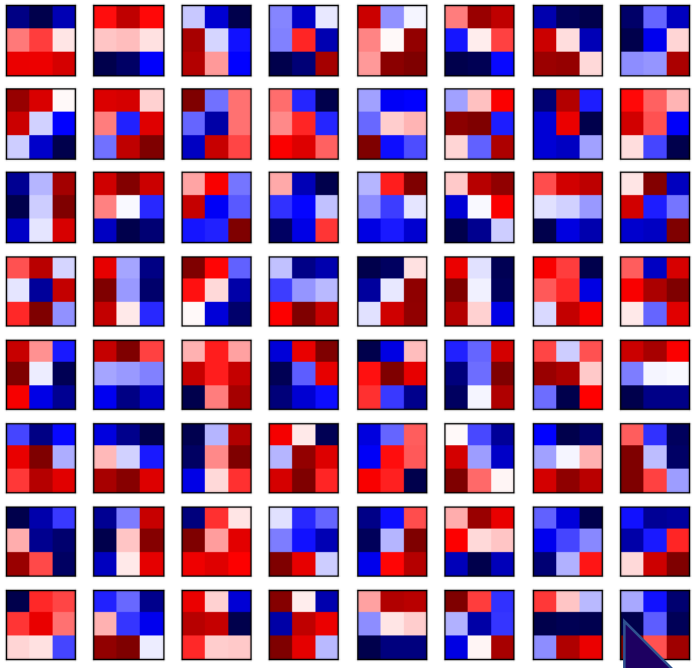
```
Train
   Accuracy: 1.00    soporte: 60000

Test
   Accuracy: 0.98    soporte: 10000
```

```
model = Sequential()
model.add(Conv2D( 64, kernel_size=3,
                      activation='relu’,
                      input_shape= INPUT_SHAPE))

model.add(Flatten())
model.add(Dense(n_clases, activation= 'softmax'))
```

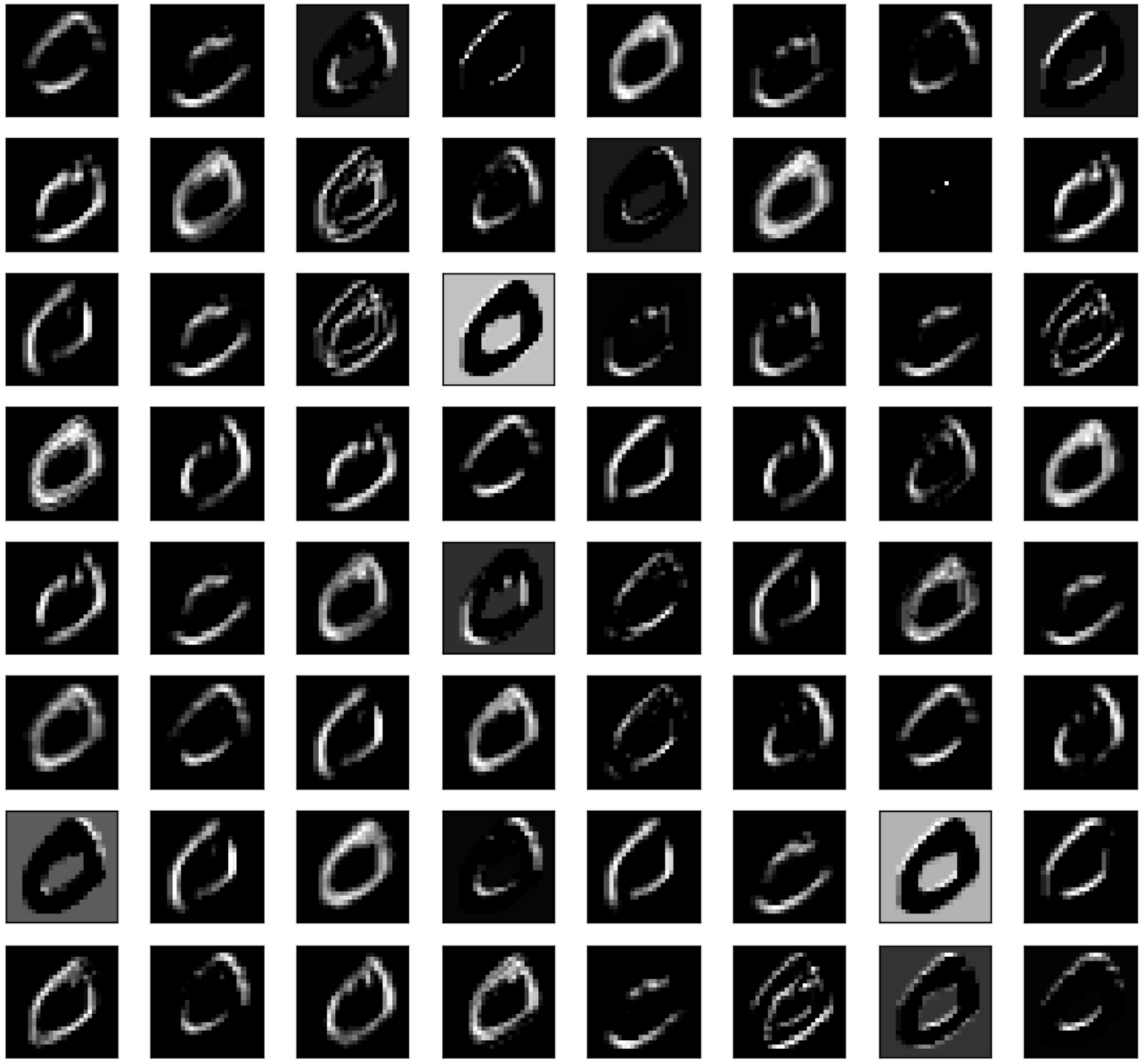
Capa convolucional

Activation maps (imágenes filtradas) de los 64 filtros entrenados.





64 filtros



Capa Convolucional sobre CIFAR10

```
model = Sequential()
model.add(Conv2D( 64,   kernel_size=3,activation='relu',input_shape= INPUT_SHAPE))

model.add(Flatten())
model.add(Dense(32, activation= 'relu'))
model.add(Dense(n_clases, activation= 'softmax'))
```

¿Por qué la imagen resultante tiene 30x30?

¿Por la capa convolucional tiene 1792 parámetros (pesos)?

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 64)        1792
_________________________________________________________________
flatten_1 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1843232
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 1,845,354
Trainable params: 1,845,354
Non-trainable params: 0
```

64 x 3 x 3 x 3 + 64

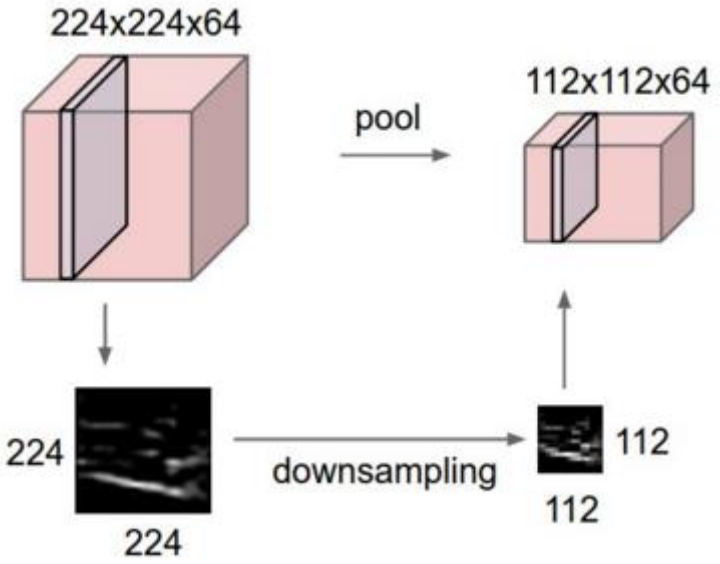
Cant filtros | Tamaño del filtro | Bias

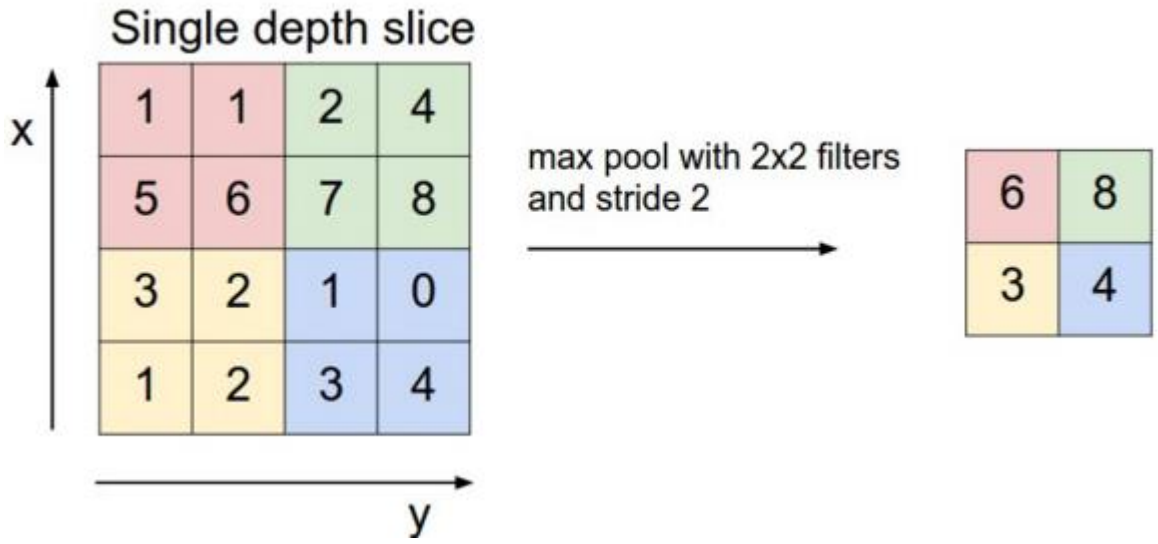
Notar que el vector de entrada a la Feed-Forward es de 30x30x64= 57600.
Esto hace que haya casi 2 millones de parámetros! solo para 32 neuronas ocultas.

Capas Pooling

Las capas Pooling ayudan a reducir la dimensionalidad espacial del *feature map* de una convolución. Básicamente son convoluciones con un stride igual al tamaño del kernel y donde se calcula alguna función sobre todos los píxeles. Lo más usual es calcular el máximo, el mínimo o el promedio.

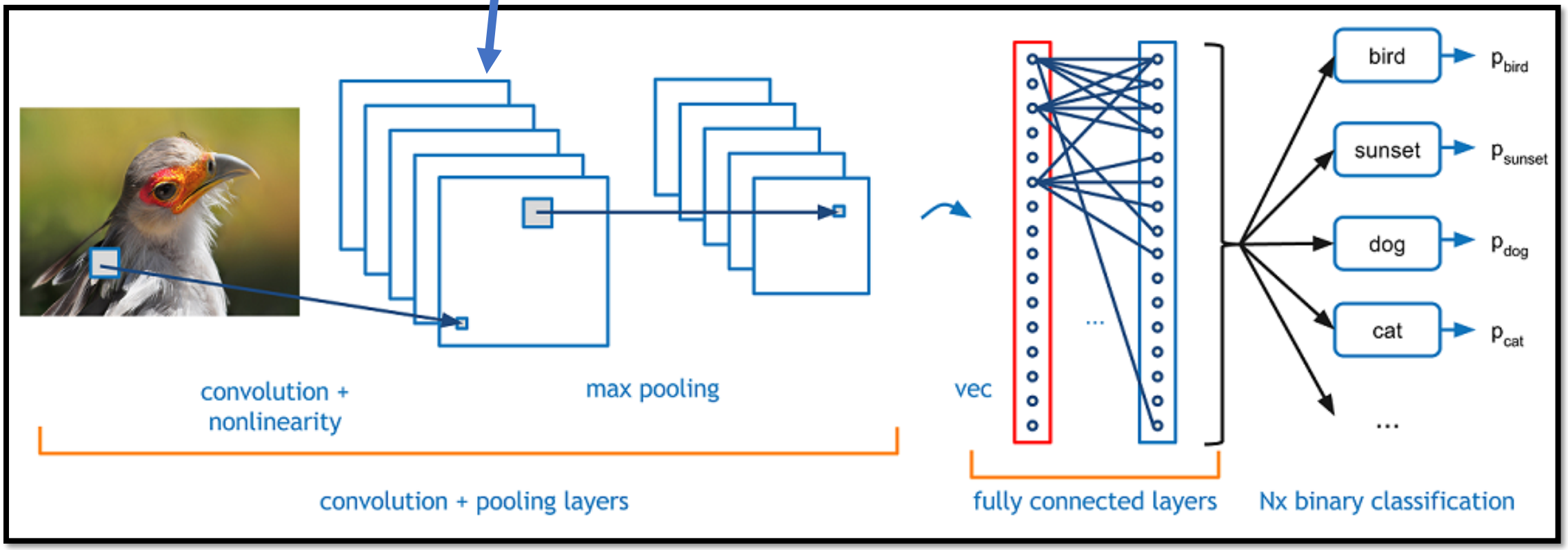
No solo reducen la dimensionalidad, sino que generalmente ayudan en la clasificación.





Capas Pooling

Generalmente se grafican los *Feature maps*, no los *kernels.*
Ya que esto se da como entrada a la próxima capa.



Capa Convolucional sobre CIFAR10

```
model = Sequential()
model.add(Conv2D( 64,   kernel_size=3,activation='relu’,input_shape= INPUT_SHAPE))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(32, activation= 'relu'))
model.add(Dense(n_clases, activation= 'softmax'))
```

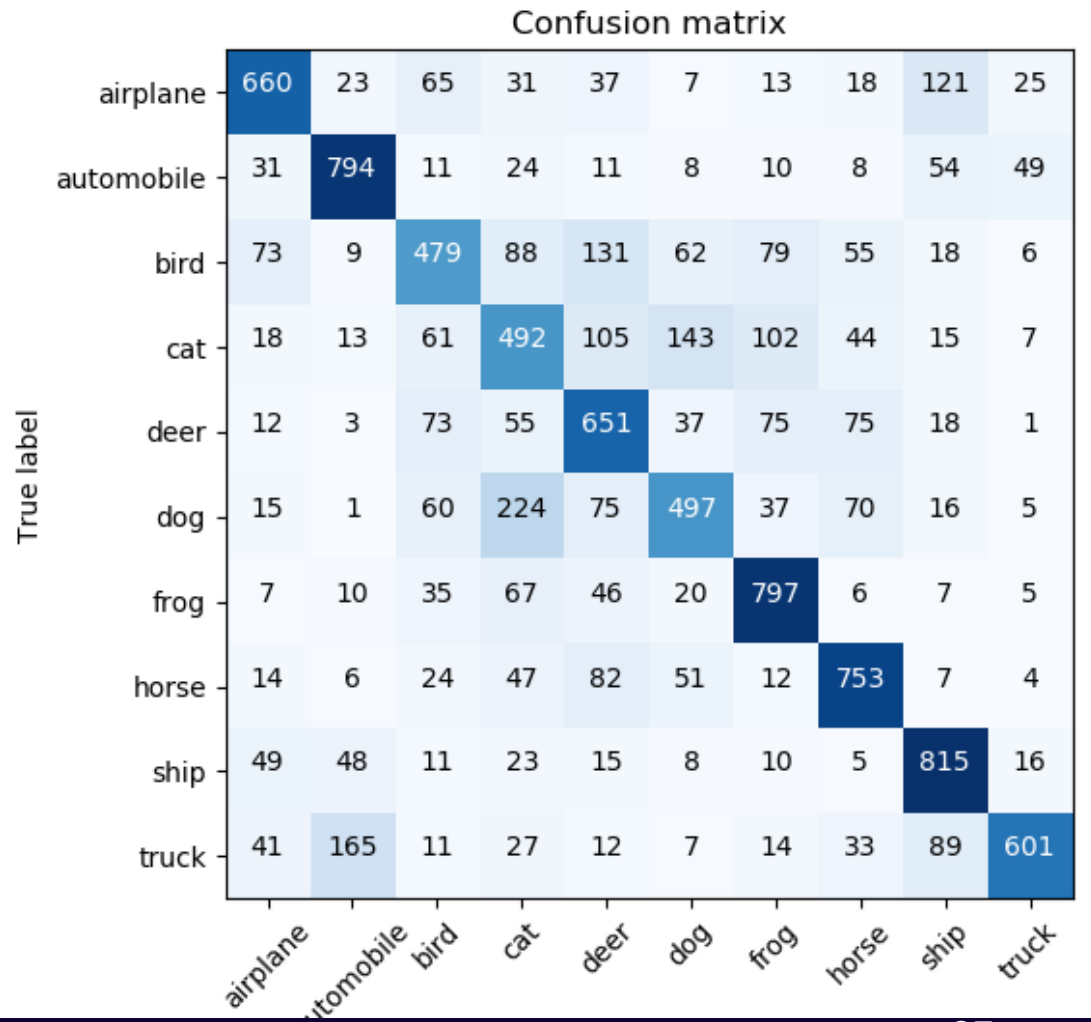
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 14400)             0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                460832
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 462,954
Trainable params: 462,954
Non-trainable params: 0
```

Agregando la capa Pooling la cantidad de parámetros se redujo a un cuarto

Capa Convolucional sobre CIFAR10

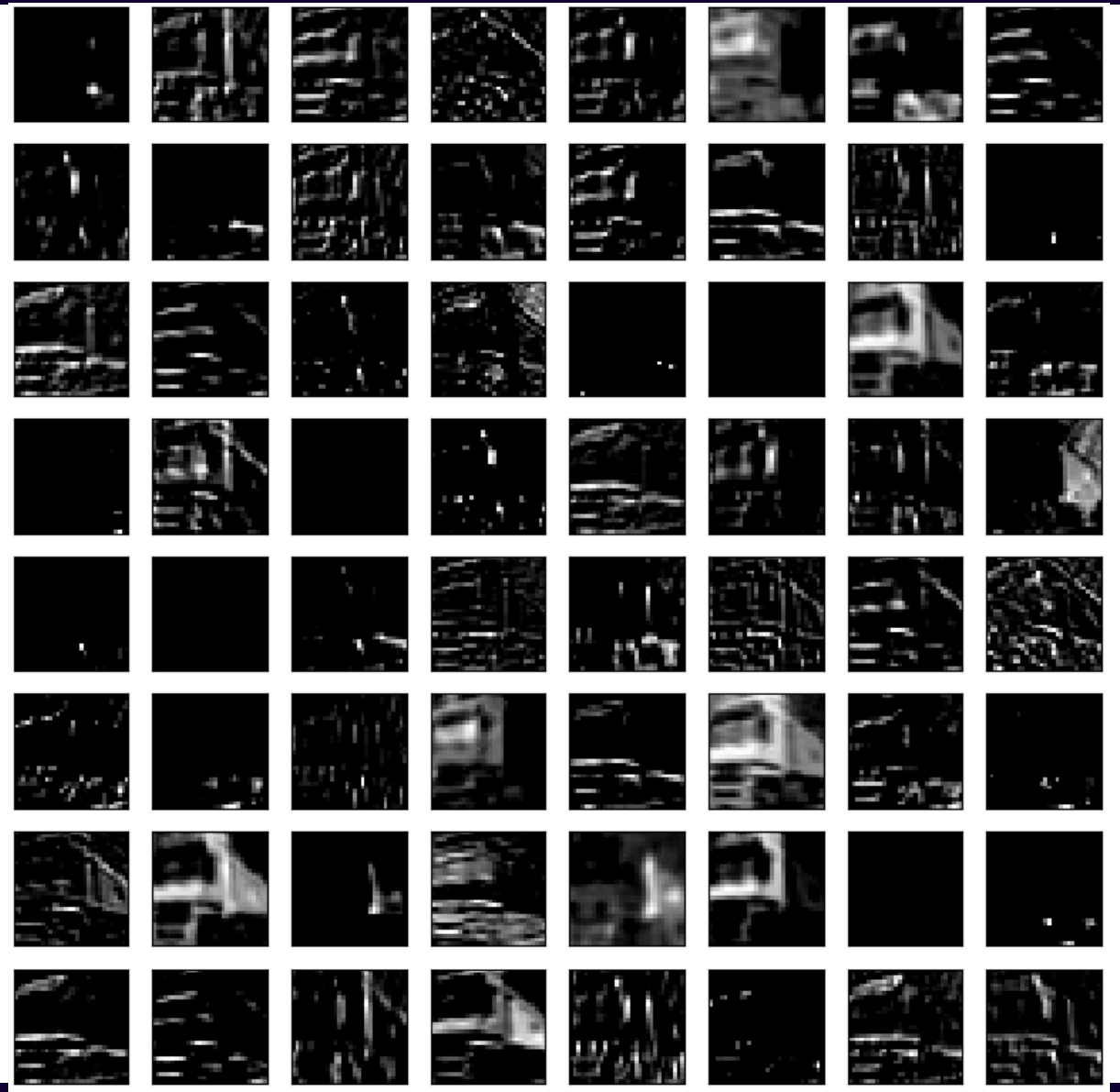
```
model = Sequential()
model.add(Conv2D( 64, kernel_size=3,activation='relu', input_shape= INPUT_SHAPE))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(32, activation= 'relu'))
model.add(Dense(n_clases, activation= 'softmax'))
```

```
Train
   Accuracy: 0.78     soporte: 50000

Test
   Accuracy: 0.65     soporte: 10000
```



Filtros convolucionales sobre CIFAR10

Activation maps (imágenes filtradas) de los 64 filtros entrenados.



64 filtros de 3x3x3

Capas convolucionales

A medida que las capas se apilan, los filtros convolucionales se aplican sobre los feature maps de las capas anteriores.

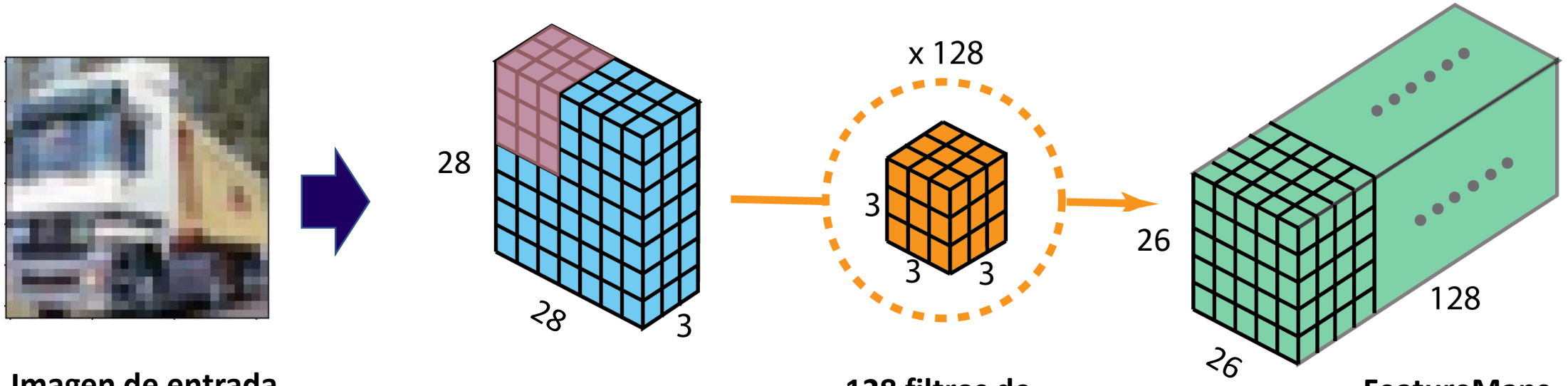


Imagen de entrada color (RGB)

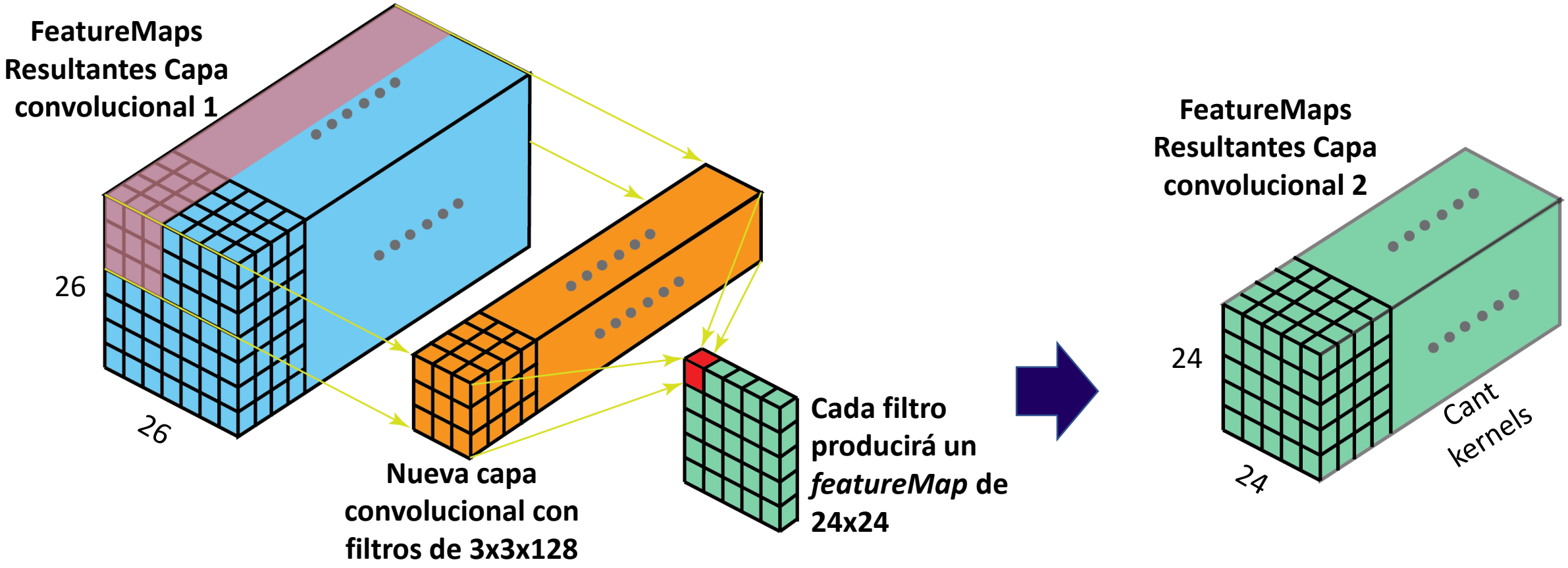
Imagen RGB

128 filtros de 3x3

FeatureMaps Resultantes de los 128 filtros

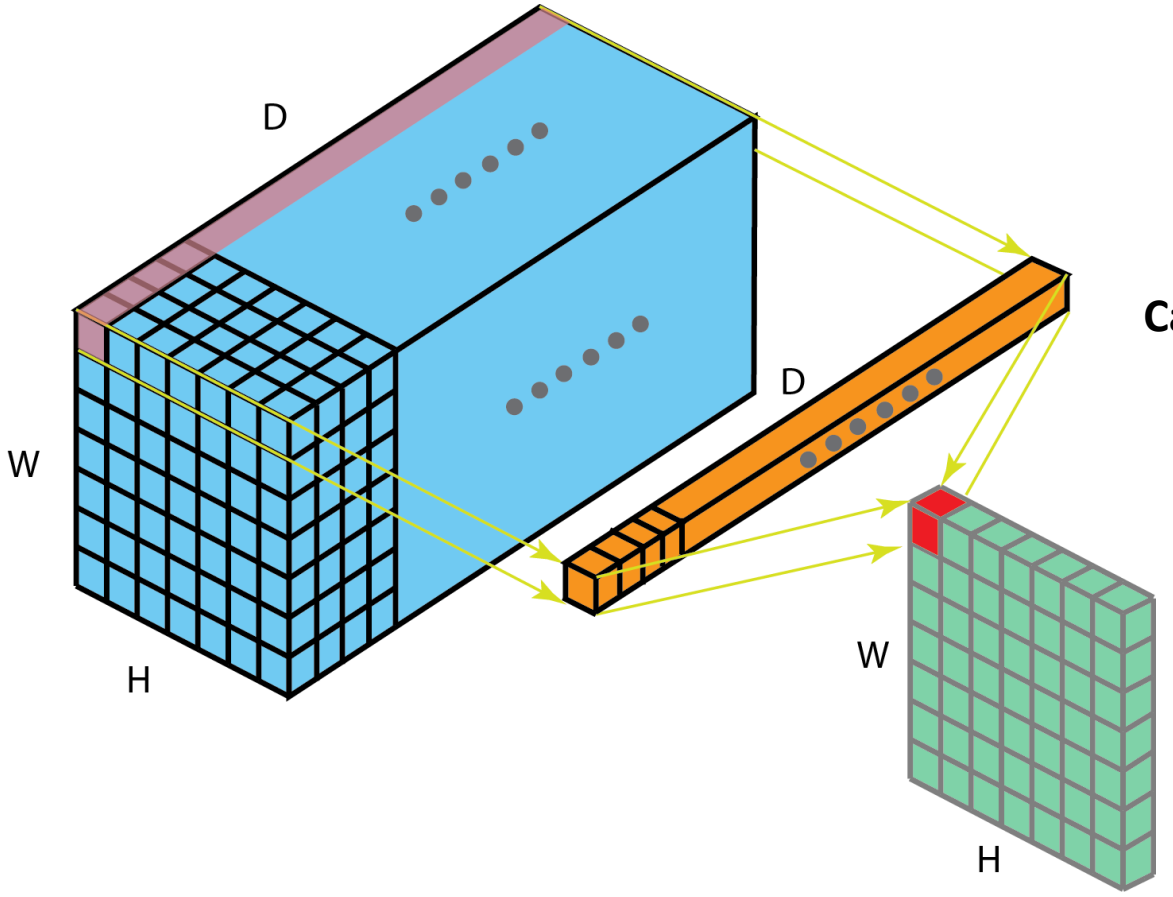
Capas convolucionales

A medida que las capas se apilan, los filtros convolucionales se aplican sobre los feature maps de las capas anteriores.



Kernel sizes

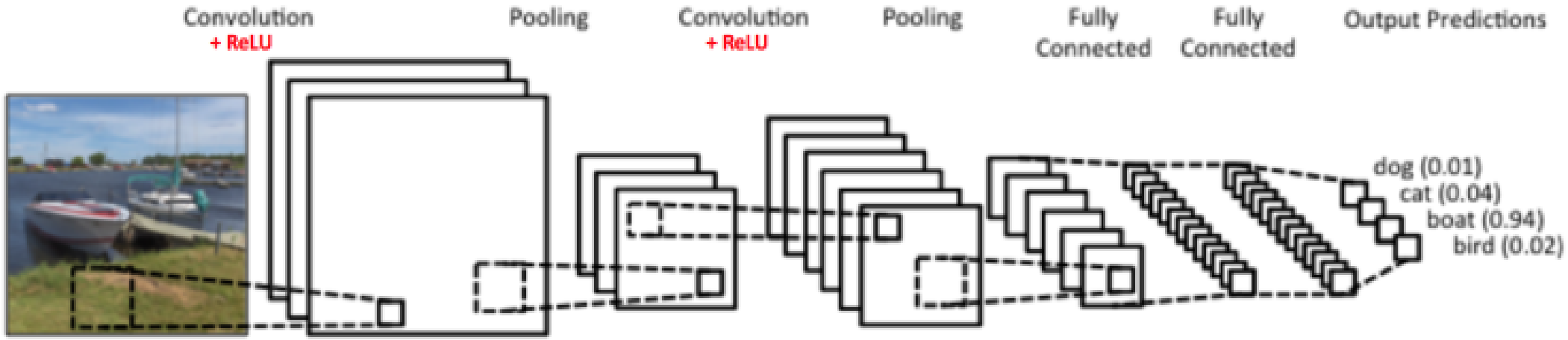
Lo más usual es tener tamaños de kernel de **3x3**, **5x5** y **1x1**.



Caso particular de filtros 1x1

Red Convolucional estándar

Generalmente se suelen utilizar varias capas convolucionales seguidas de capas pooling.



Red Convolucional estándar

La función de activación es muy importante para generar una transformación no lineal en la salida de las neuronas.

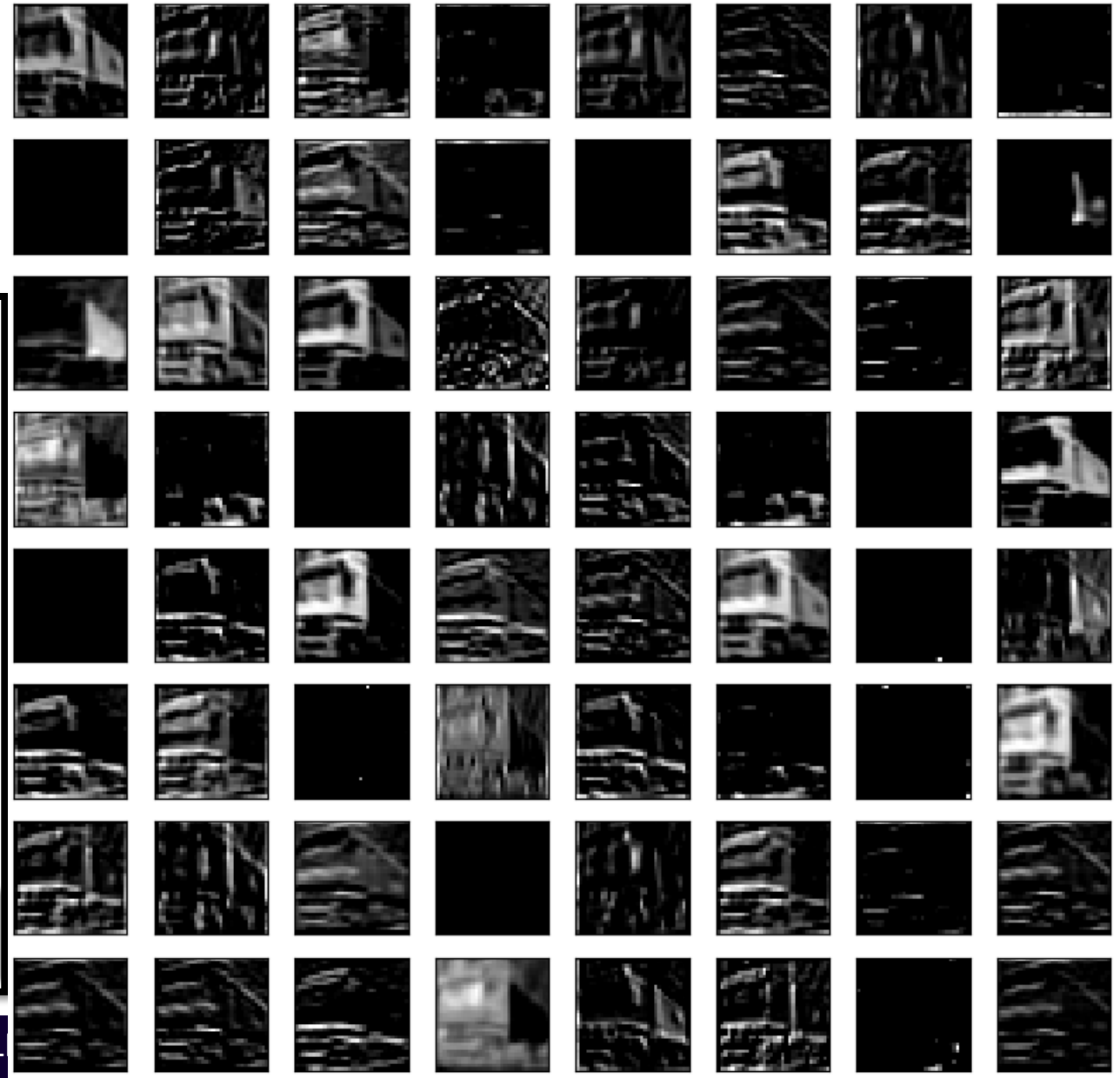
Varias capas sobre CIFAR10

Modelo más “profundo” para clasificar CIFAR10

```
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu’,
                 input_shape= INPUT_SHAPE, padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=3, activation='relu’,
                 input_shape= INPUT_SHAPE, padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128,input_dim=d_in, activation= 'relu'))
model.add(Dense(n_clases, activation= 'softmax))
```

Varias capas sobre CIFAR10

Salida primer capa convolucional



Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 32, 32, 64)	1792
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_5 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 64)	0
flatten_2 (Flatten)	(None, 4096)	0
dense_3 (Dense)	(None, 128)	524416
dense_4 (Dense)	(None, 10)	1290

Total params: 564,426
Trainable params: 564,426
Non-trainable params: 0

```
Test
   Accuracy: 0.70    soporte: 10000
```

Varias capas sobre CIFAR10

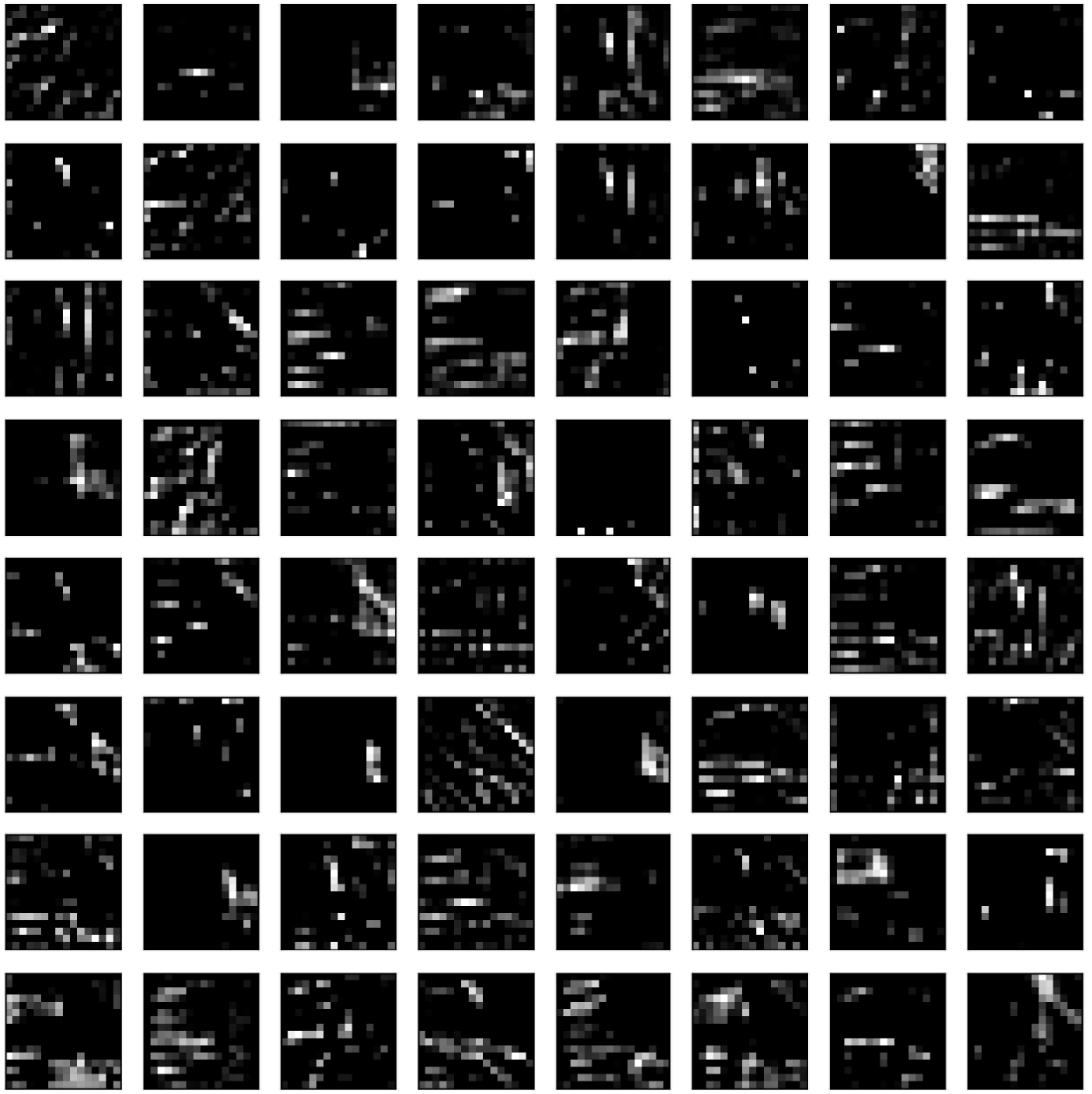
Después de Max Pooling 2x2

64 *Feature Maps* de 16x16

Varias capas sobre CIFAR10

Después de segunda capa convolucional.

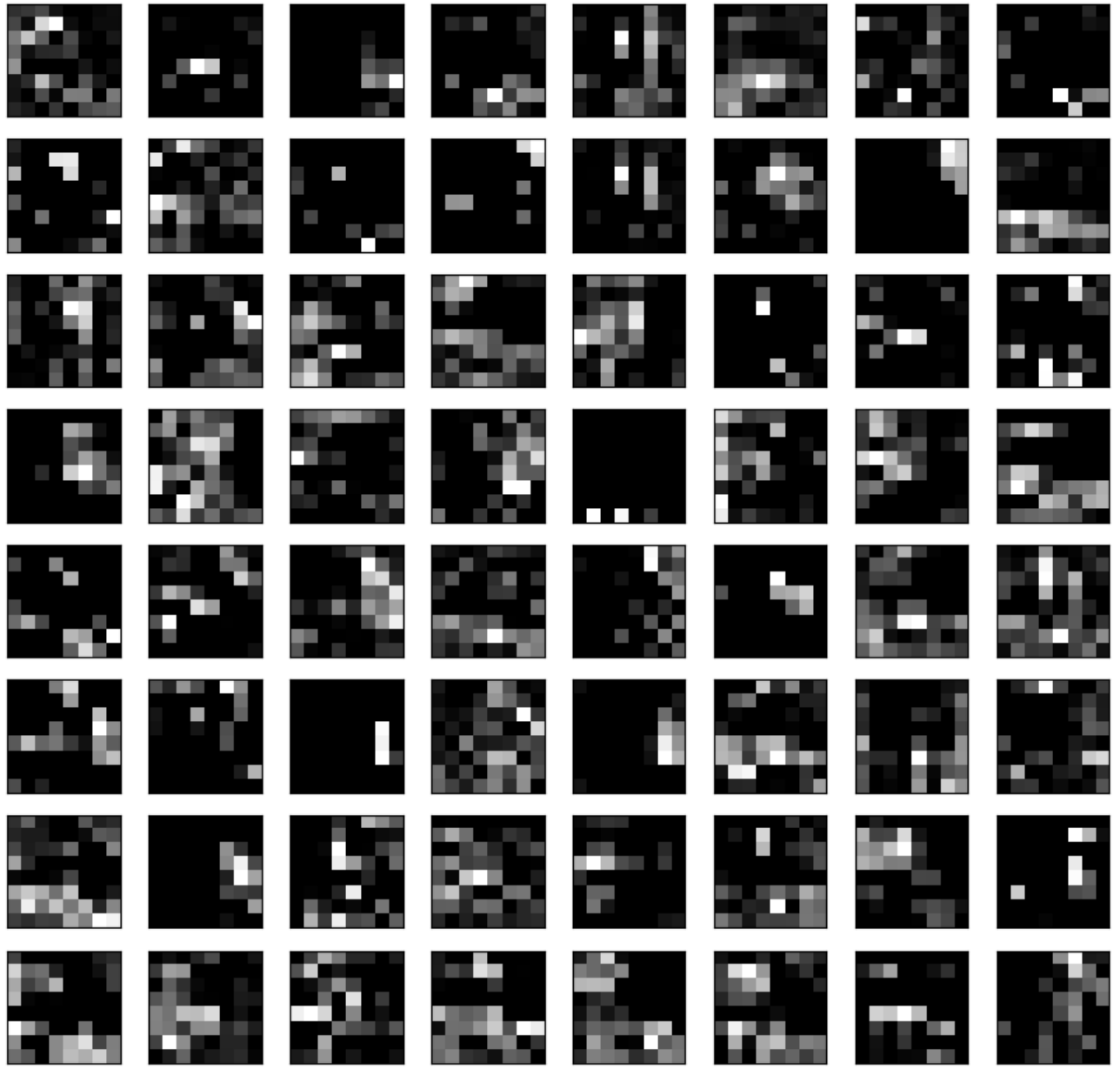
64 *Feature Maps* de 16x16



Varias capas sobre CIFAR10

Después de Max Pooling 2x2

64 *Feature Maps* de 8x8



Resumen

- Las Capas convolucionales 2D tienen diferentes filtros (kernels) que se entrenan para detectar diferentes características de las imágenes del dataset.
- Cada filtro genera un *Activation Maps* de NxNx1. Donde N dependerá del tamaño original de la imagen, del filtro, el padding y el stride que usemos.
- Todos los *Activation Maps* de los distintos filtro de una capa convolucional se apilan, generando una nueva "imagen" de NxNxF, siendo F la cantidad de filtros.
- Las capas Pooling permiten reducir la dimensionalidad del problema, haciendo no solo más rápido el entrenamiento sino más eficaz al momento de generalizar.
- El modo más ordenado de realizar una arquitectura CNN es ir intercalando capas Convolucionales con capas de Pooling hasta llegar a las capas Dense (Feed-Forward) que discriminarán las características aprendidas por las capas anteriores.