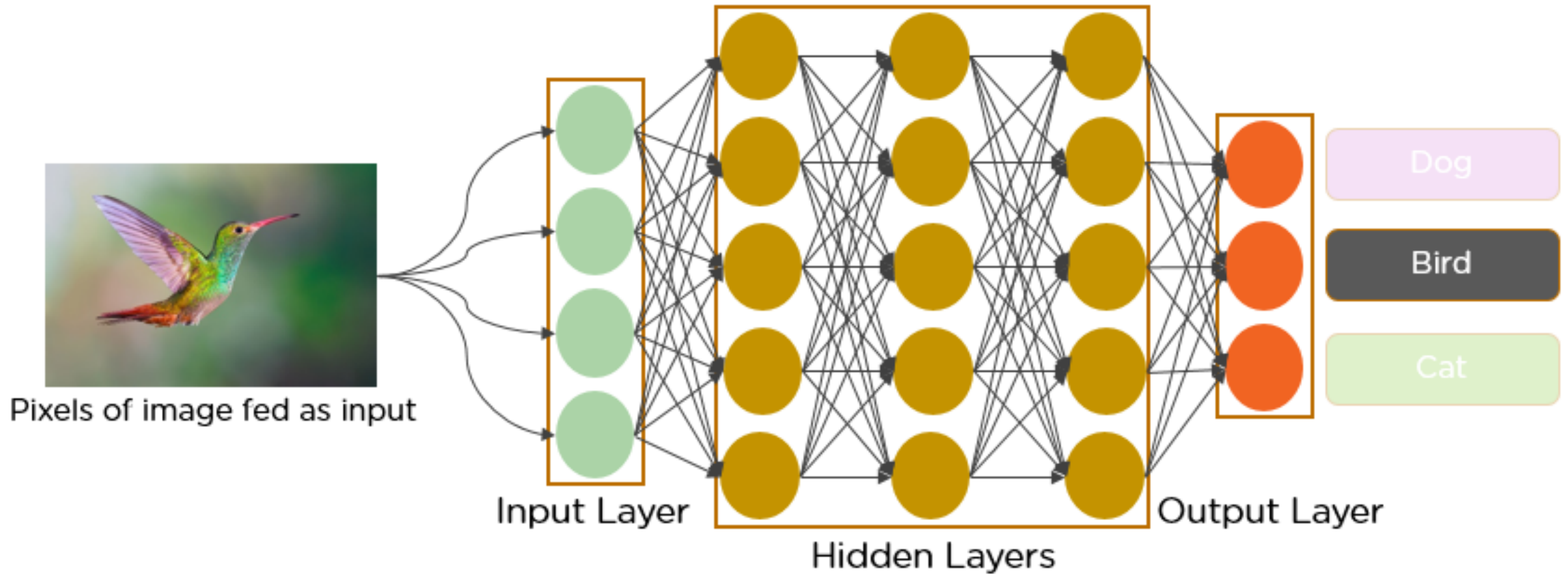
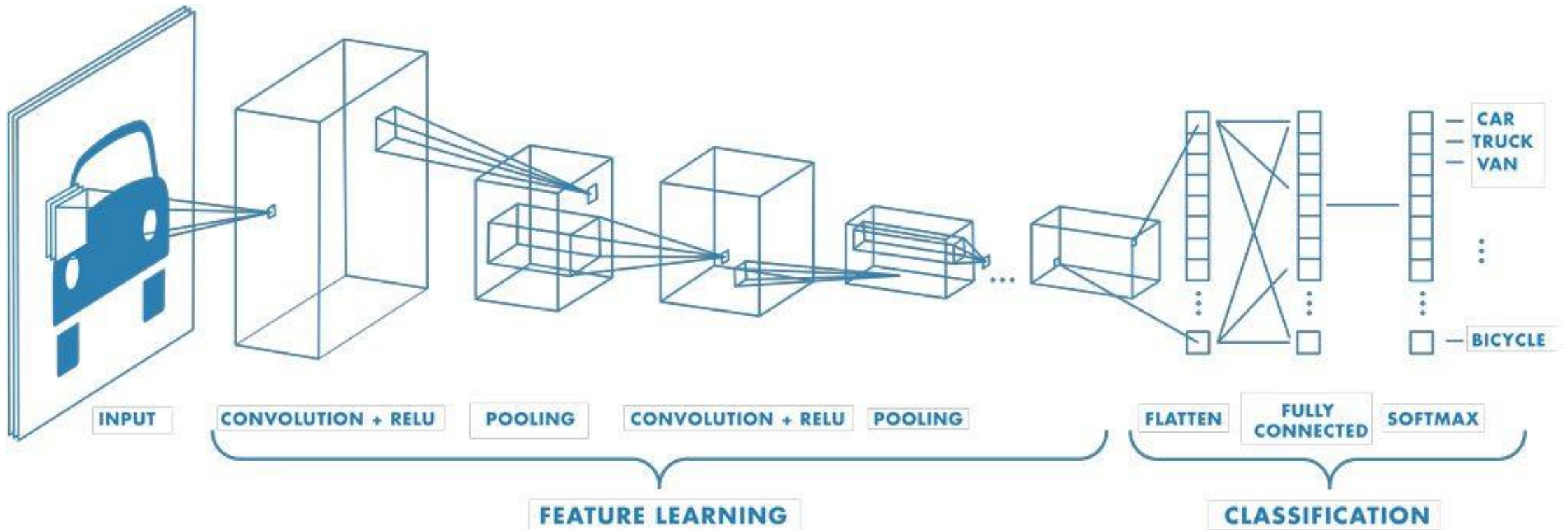


# Clasificación de imágenes

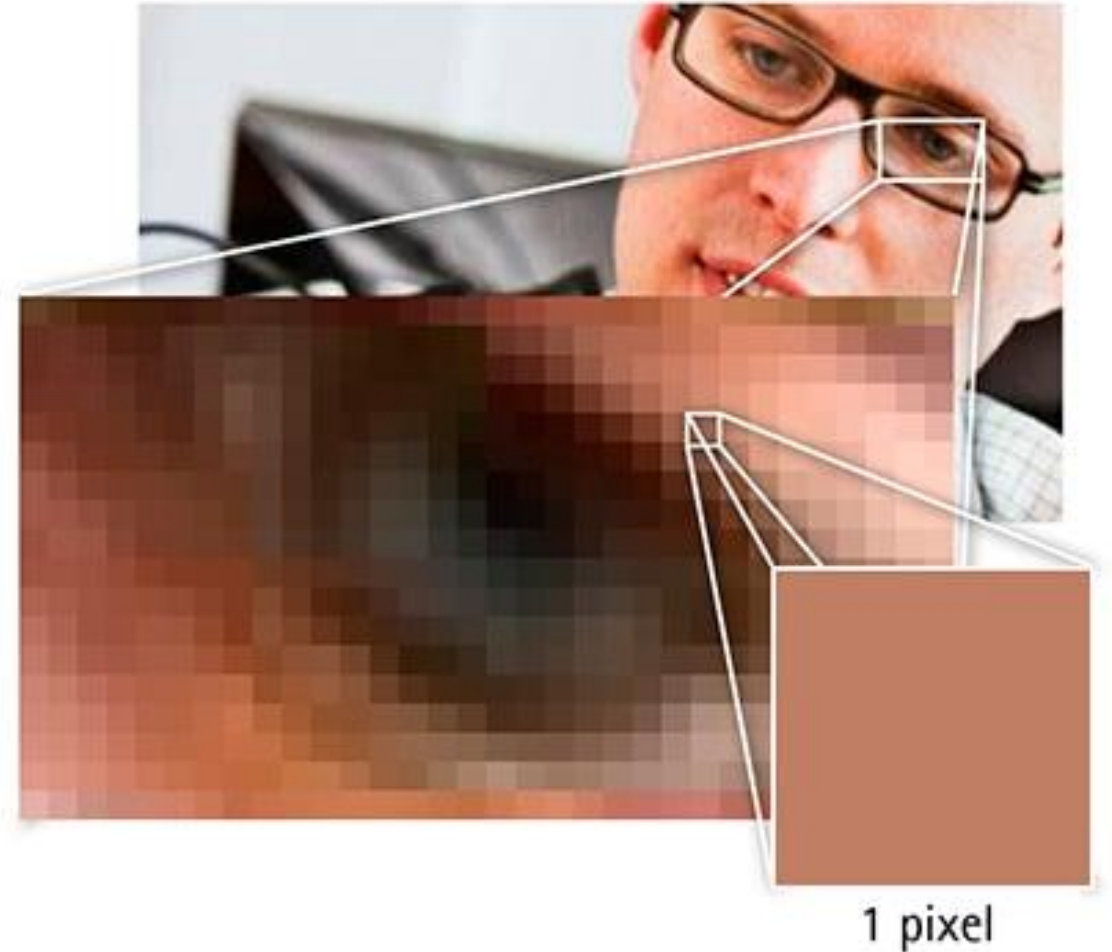


# Red Neuronal Convolutcional



# Imagen digital

- Está representada por una matriz de  $M \times N$  pixels.
- Cada pixel representa la intensidad de luz en ese punto.
- Su valor dependerá del tipo de imagen



# Imagen en tonos de grises



Greyscale image

		112		
	112	105	106	
105	105	105	105	112
105	112	112	105	112
112	106	106	112	105
105	112	112	112	106
106	112	116	117	105
120	123	105	105	10
	127	127	106	
	127	127	10	

Pixel/intensity value

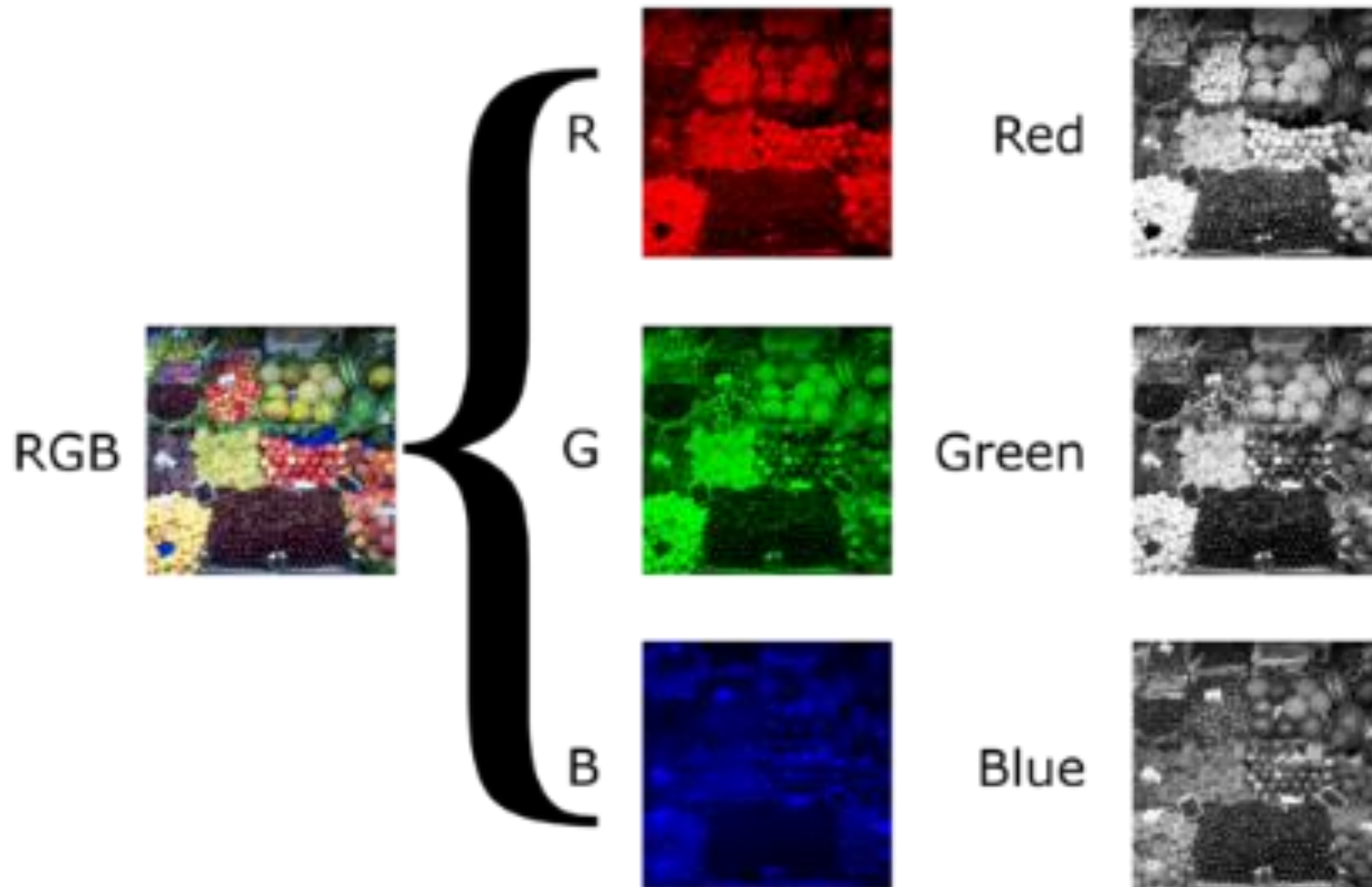
# Imagen en tonos de grises



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

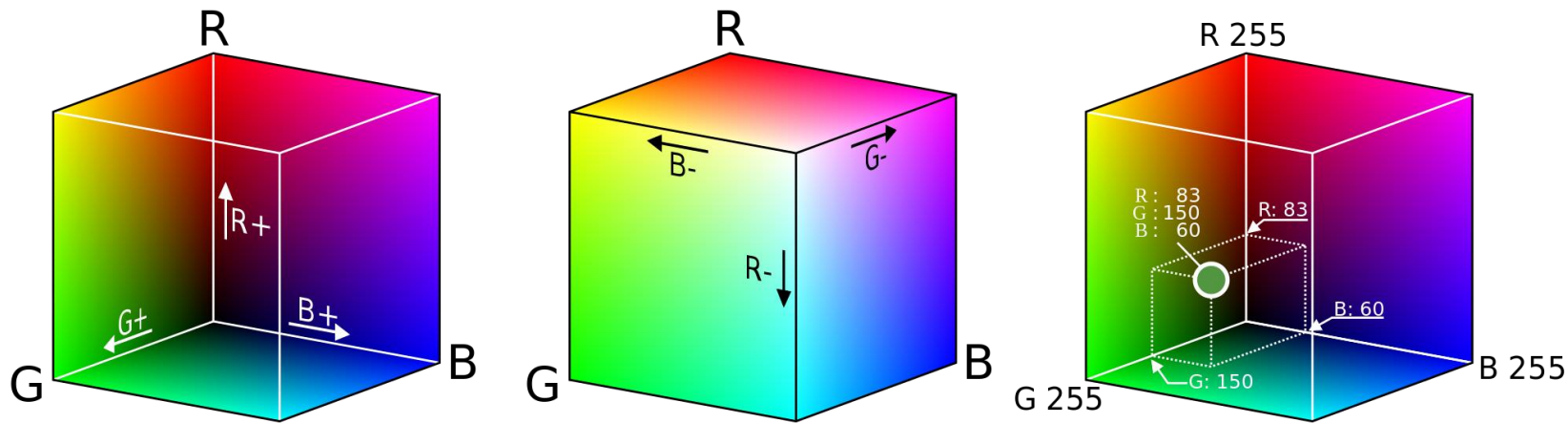
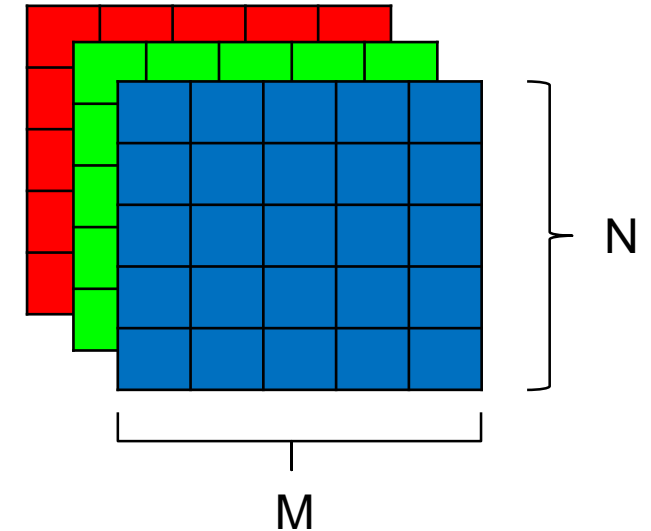


# Imagen color - RGB



# Modelo RGB

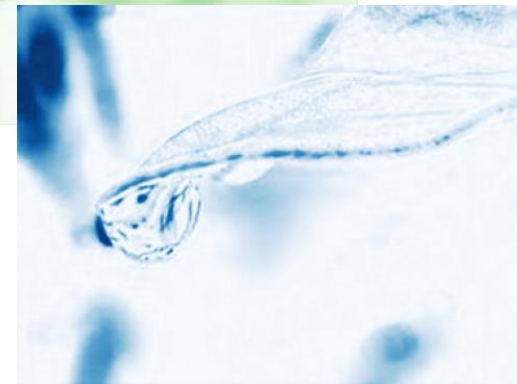
- Una imagen se representa mediante 3 matrices de  $N \times M$ , una para cada canal.
- A mayor valor en los 3 canales los colores se aclaran.



# Visualización

```
from skimage import io
imgColor = io.imread("hoja.jpg")
plt.figure()
plt.imshow (imgColor)
plt.show()
```

```
C = ['Reds', 'Greens', 'Blues']
for p in [0,1,2]:
    plt.figure()
    plt.imshow (imgColor[:, :, p], cmap=C[p])
    plt.show()
```





# RGB a escala de grises

- Puede combinarse la información de los 3 canales para producir una imagen en tonos de grises

$$\text{GRIS} = (R + G + B) / 3$$



# RGB a escala de grises

- Hay otras conversiones que reflejan mejor la percepción del ojo humano

$$\text{GRIS} = (0.3 R + 0.59 G + 0.11 B)$$



# Cargando una imagen

```
from skimage import io  
imgColor = io.imread("tigre.jpg")
```



(150, 134, 3)

```
from skimage import io  
imgGris = io.imread("tigre.jpg",  
                    as_gray=True)
```



(150, 134)

# Filtros en el dominio espacial

- En este proceso se relaciona un conjunto de píxeles próximos al píxel objetivo con la finalidad de obtener una información útil



*Usaremos máscaras o kernels de **convolución***

# Convolución 2D

- La operación de convolución de una imagen con un filtro o kernel permite destacar ciertas características de dicha imagen.



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





# Convolución 2D

- Filtro de detección de bordes horizontal



1	1	1
0	0	0
-1	-1	-1



# Convolución 2D

- Filtro de detección de bordes diagonal



-1	0
0	1



# Convolución 2D

- Filtro de detección de bordes diagonal



0	-1
1	0

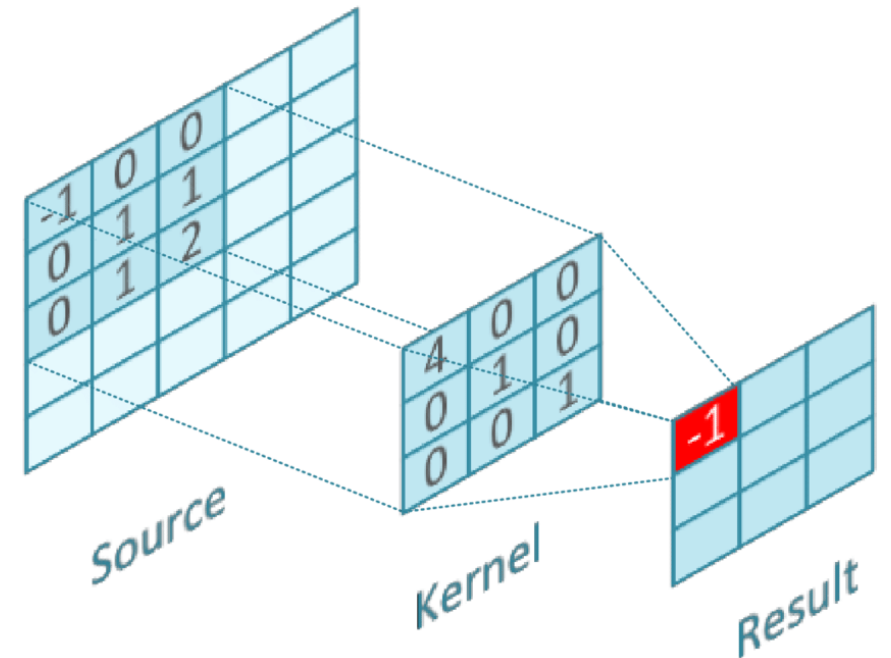


# Convolución 2D

- La convolución discreta de dos funciones  $f$  y  $g$  se define como

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

- La función  $g$  se desplaza antes de multiplicar.



# Convolución 2D

Entrada

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

Kernel

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

Salida

	266			

$$\begin{aligned} &60 * 0 + 113 * (-1) + 56 * 0 + \\ &73 * (-1) + 121 * 5 + 54 * (-1) + \\ &131 * 0 + 99 * (-1) + 70 * 0 = 266 \end{aligned}$$



# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61		

$$\begin{aligned} &113 * 0 + 56 * (-1) + 139 * 0 + \\ &121 * (-1) + 54 * 5 + 84 * (-1) + \\ &99 * 0 + 70 * (-1) + 129 * 0 = -61 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	

$$\begin{aligned} &56 * 0 + 139 * (-1) + 85 * 0 + \\ &54 * (-1) + 84 * 5 + 128 * (-1) + \\ &70 * 0 + 129 * (-1) + 127 * 0 = -30 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116			

$$\begin{aligned} & 73 * 0 + 121 * (-1) + 54 * 0 + \\ & 131 * (-1) + 99 * 5 + 70 * (-1) + \\ & 80 * 0 + 57 * (-1) + 115 * 0 = 116 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47		

$$\begin{aligned} &121 * 0 + 54 * (-1) + 84 * 0 + \\ &99 * (-1) + 70 * 5 + 129 * (-1) + \\ &57 * 0 + 115 * (-1) + 69 * 0 = -47 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	

$$\begin{aligned} &54 * 0 + 84 * (-1) + 128 * 0 + \\ &70 * (-1) + 129 * 5 + 127 * (-1) + \\ &115 * 0 + 69 * (-1) + 134 * 0 = 295 \end{aligned}$$



# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	
	-135			

$$\begin{aligned} &131 * 0 + 99 * (-1) + 70 * 0 + \\ &80 * (-1) + 57 * 5 + 115 * (-1) + \\ &104 * 0 + 126 * (-1) + 123 * 0 = -135 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	
	-135	256		

$$\begin{aligned} & 99 * 0 + 70 * (-1) + 129 * 0 + \\ & 57 * (-1) + 115 * 5 + 69 * (-1) + \\ & 126 * 0 + 123 * (-1) + 95 * 0 = 256 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	
	-135	256	-128	

$$\begin{aligned} &70 * 0 + 129 * (-1) + 127 * 0 + \\ &115 * (-1) + 69 * 5 + 134 * (-1) + \\ &123 * 0 + 95 * (-1) + 130 * 0 = -128 \end{aligned}$$

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	
	-135	256	-128	

## □ Parámetros

▣ **Kernel\_size:** tamaño del filtro o kernel. En este caso = 3

▣ **Stride:** desplazamiento del filtro cada vez que se aplica. En este caso = 1

# Convolución 2D

Entrada

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

Kernel

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

Salida

	266	-61	-30	
	116	-47	295	
	-135	256	-128	

*¿Por qué resultado de la convolución tiene un tamaño menor al de la entrada?*



# Convolución 2D - Padding

- Usando padding conservamos el tamaño de la imagen original

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 - Padding

- Usando padding conservamos el tamaño de la imagen original

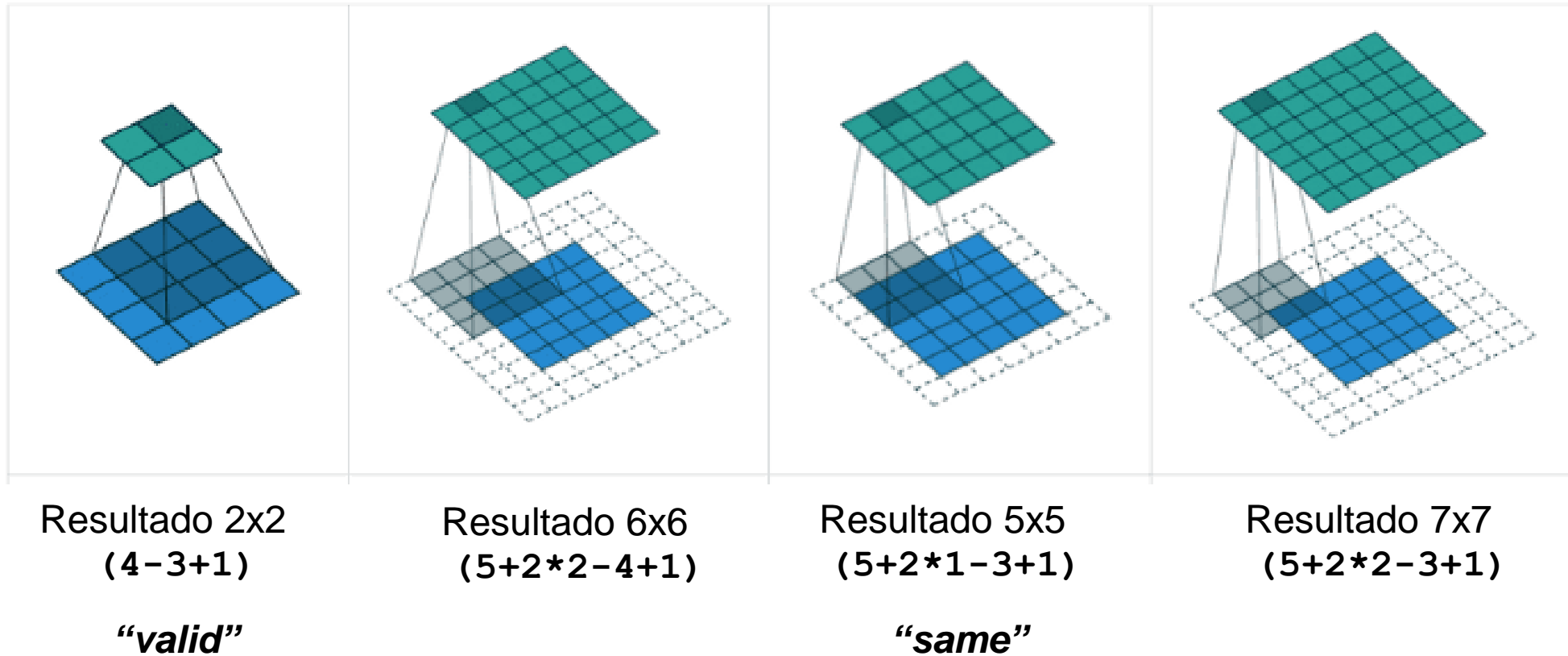
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

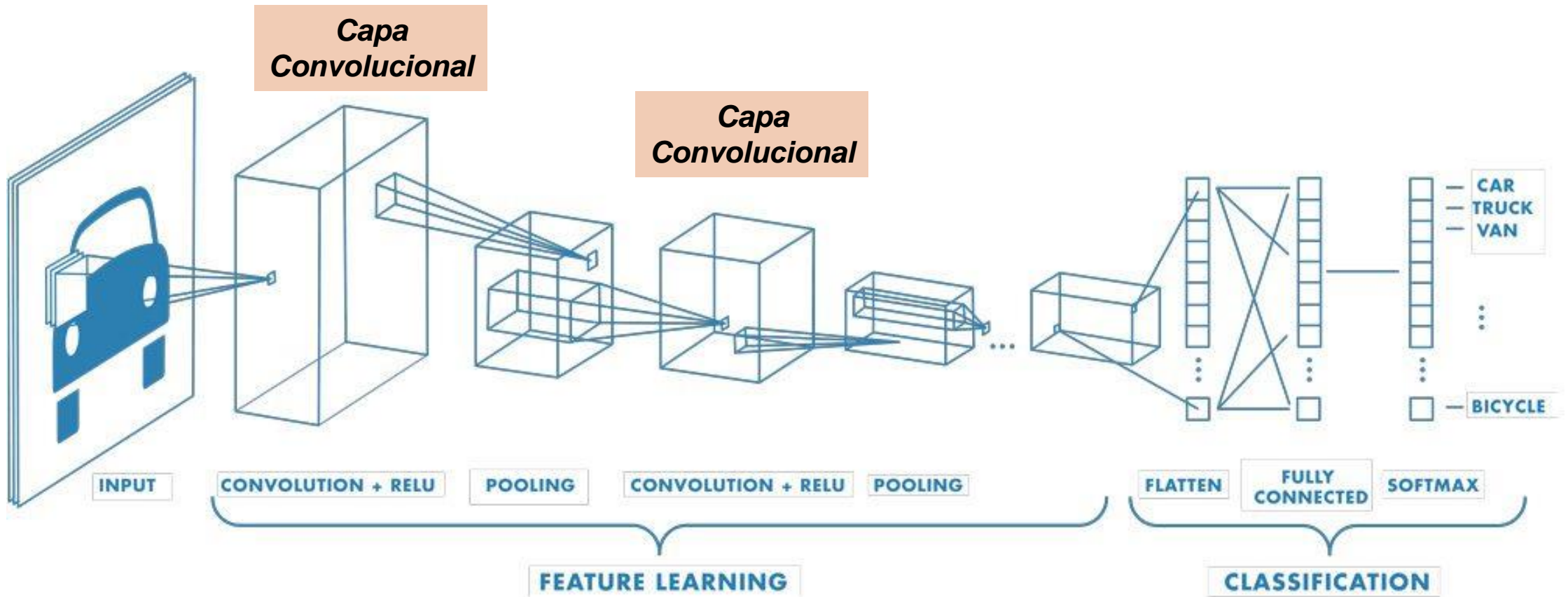
0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

# Convolución 2D - Padding

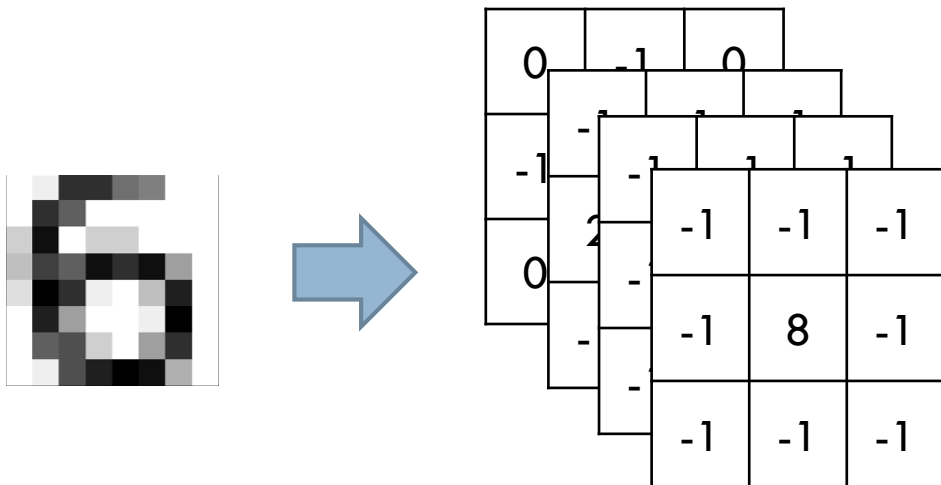


# Red Neuronal Convolutcional



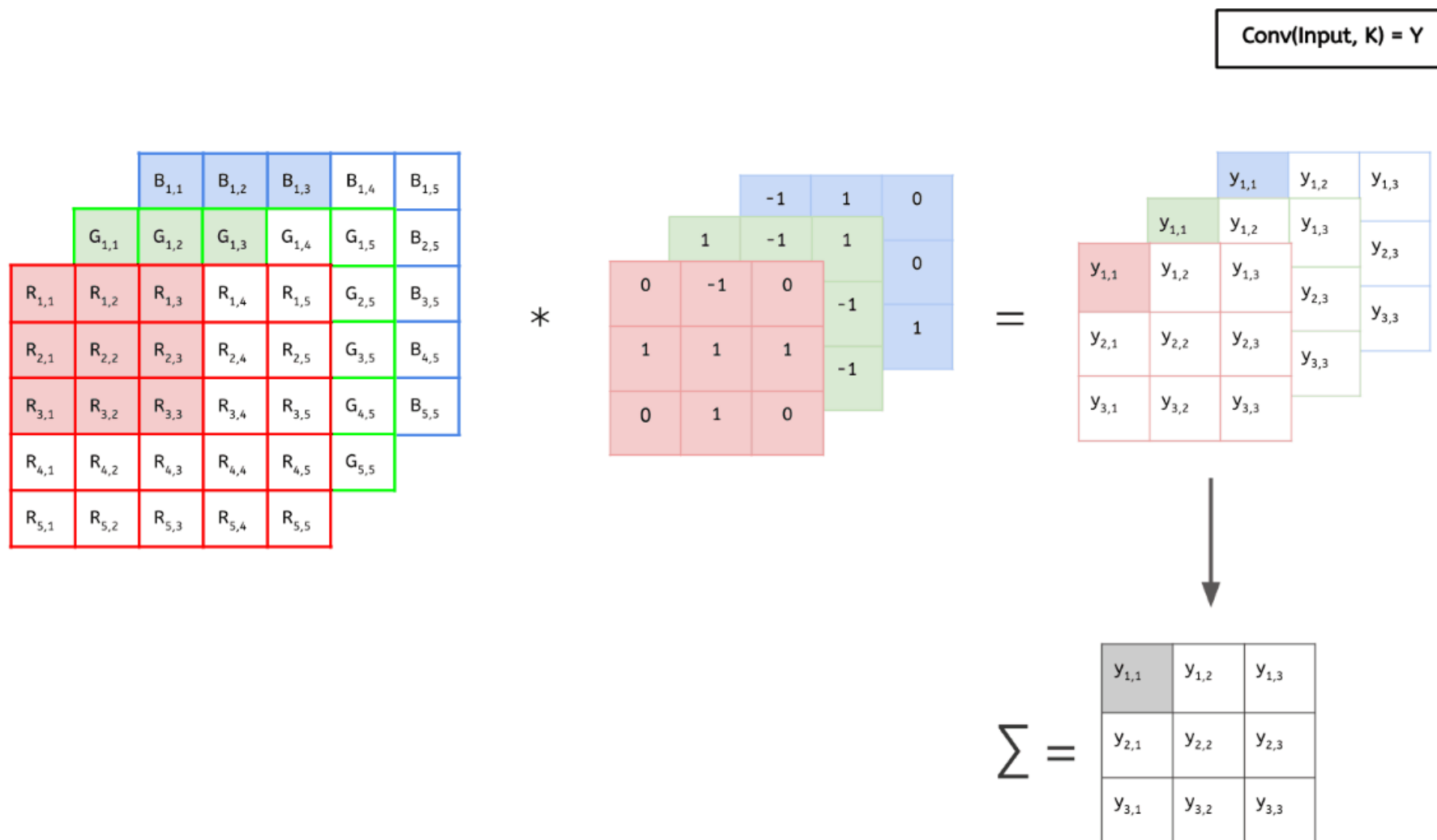
# Capa convolucional

- Está formada por varios filtros convolucionales



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

# Convolución 2D con 3 canales



Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$						
0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0
$x[:, :, 1]$						
0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0
$x[:, :, 2]$						
0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$		
-1	0	1
0	0	1
1	-1	1
$w0[:, :, 1]$		
-1	0	1
1	-1	1
0	1	0
$w0[:, :, 2]$		
-1	1	1
1	1	0
0	-1	0
Bias b0 (1x1x1)		
$b0[:, :, 0]$		
1		

Filter W1 (3x3x3)

$w1[:, :, 0]$		
0	1	-1
0	-1	0
0	-1	1
$w1[:, :, 1]$		
-1	0	0
1	-1	0
1	-1	0
$w1[:, :, 2]$		
-1	1	-1
0	-1	-1
1	0	0
Bias b1 (1x1x1)		
$b1[:, :, 0]$		
0		

Output Volume (3x3x2)

$o[:, :, 0]$		
2	3	3
3	7	3
8	10	-3
$o[:, :, 1]$		
-8	-8	-3
-3	1	0
-3	-8	-5

Input :  $N \times M \times N_c = 5 \times 5 \times 3$   
 Kernel\_size=  $K \times K \times N_c = 3 \times 3 \times 3$   
 Cant\_filtros =  $N_f = 2$   
 Padding =  $P = 1$   
 Stride =  $S = 2$

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$						
0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0
$x[:, :, 1]$						
0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0
$x[:, :, 2]$						
0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$		
-1	0	1
0	0	1
1	-1	1
$w0[:, :, 1]$		
-1	0	1
1	-1	1
0	1	0
$w0[:, :, 2]$		
-1	1	1
1	1	0
0	-1	0
Bias b0 (1x1x1)		
$b0[:, :, 0]$		
1		

Filter W1 (3x3x3)

$w1[:, :, 0]$		
0	1	-1
0	-1	0
0	-1	1
$w1[:, :, 1]$		
-1	0	0
1	-1	0
1	-1	0
$w1[:, :, 2]$		
-1	1	-1
0	-1	-1
1	0	0
Bias b1 (1x1x1)		
$b1[:, :, 0]$		
0		

Output Volume (3x3x2)

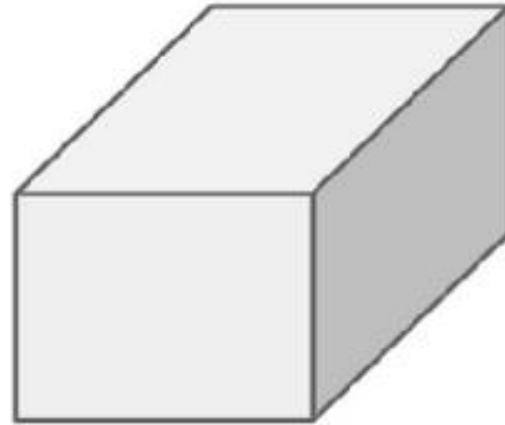
$o[:, :, 0]$		
2	3	3
3	7	3
8	10	-3
$o[:, :, 1]$		
-8	-8	-3
-3	1	0
-3	-8	-5

Input :  $N \times M \times N_c = 5 \times 5 \times 3$   
 Kernel\_size=  $K \times K \times N_c = 3 \times 3 \times 3$   
 Cant\_filtros =  $N_f = 2$   
 Padding =  $P = 1$   
 Stride =  $S = 2$



# Convolución 2D

□ ¿Cuál es el tamaño del resultado?



$N \times N \times N_c$



- filter :  $f \times f \times N_c'$
- stride :  $s$
- padding :  $p$



$$= \left( \frac{n + 2p - f}{s} + 1 \right) \times \left( \frac{n + 2p - f}{s} + 1 \right) \times N_c'$$

Input Volume (+pad 1) (7x7x3)

x[:, :, 0]						
0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0
x[:, :, 1]						
0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0
x[:, :, 2]						
0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:, :, 0]		
-1	0	1
0	0	1
1	-1	1
w0[:, :, 1]		
-1	0	1
1	-1	1
0	1	0
w0[:, :, 2]		
-1	1	1
1	1	0
0	-1	0
Bias b0 (1x1x1)		
b0[:, :, 0]	1	

Filter W1 (3x3x3)

w1[:, :, 0]		
0	1	-1
0	-1	0
0	-1	1
w1[:, :, 1]		
-1	0	0
1	-1	0
1	-1	0

Output Volume (3x3x2)

o[:, :, 0]		
2	3	3
3	7	3
8	10	-3
o[:, :, 1]		
-8	-8	-3
-3	1	0
-3	-8	-5

Output :  $T \times U \times N_f = 3 \times 3 \times 2$

$$T = (N + 2 * P - K) / S + 1$$

$$= (5 + 2 * 1 - 3) / 2 + 1 = 3$$

$$U = (M + 2 * P - K) / S + 1 = 3$$

Input :  $N \times M \times N_c = 5 \times 5 \times 3$

Kernel\_size =  $K \times K \times N_c = 3 \times 3 \times 3$


Cant\_filtros =  $N_f = 2$

Padding =  $P = 1$

Stride =  $S = 2$

# MNIST

```
model = Sequential()  
model.add(Conv2D(64, kernel_size=3, activation="relu",  
                 input_shape=input_shape))  
model.add(Flatten())  
model.add(Dense(10, activation='softmax'))  
model.summary()
```



A diagram consisting of a red line that starts from a light orange box containing the text `(28, 28, 1)` on the right. The line moves left, then turns downwards, then left again, ending with an arrowhead pointing to the `input_shape` parameter in the `Conv2D` layer of the code above.

# MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu",
                 input_shape=input_shape))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
flatten_10 (Flatten)	(None, 43264)	0
dense_10 (Dense)	(None, 10)	432650

=====  
Total params: 433,290

¿Por qué la salida es de 26x26 si las imágenes son de 28x28?

$$(N + 2 * P - K) / S + 1$$

$$(28 + 0 - 3) / 1 + 1 = 26$$

# MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu",
                 input_shape=input_shape))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

¿Por qué la capa  
convolucional tiene 640  
parámetros?

Layer (type)	Output Shape	Param #
=====		
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
-----		
flatten_10 (Flatten)	(None, 43264)	0
-----		
dense_10 (Dense)	(None, 10)	432650
=====		
Total params:		433,290

$$N_f * K * K * N_c + N_f$$

64 \* 3 \* 3 + 64

↑                      ↑                      ↑  
Cantidad de            Tamaño del            Bias  
filtros                      filtro

# MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu",
                 input_shape=input_shape))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
```

Cantidad de parámetros de  
la capa Flatten

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 64)	640
flatten_10 (Flatten)	(None, 43264)	0
dense_10 (Dense)	(None, 10)	432650
Total params: 433,290		

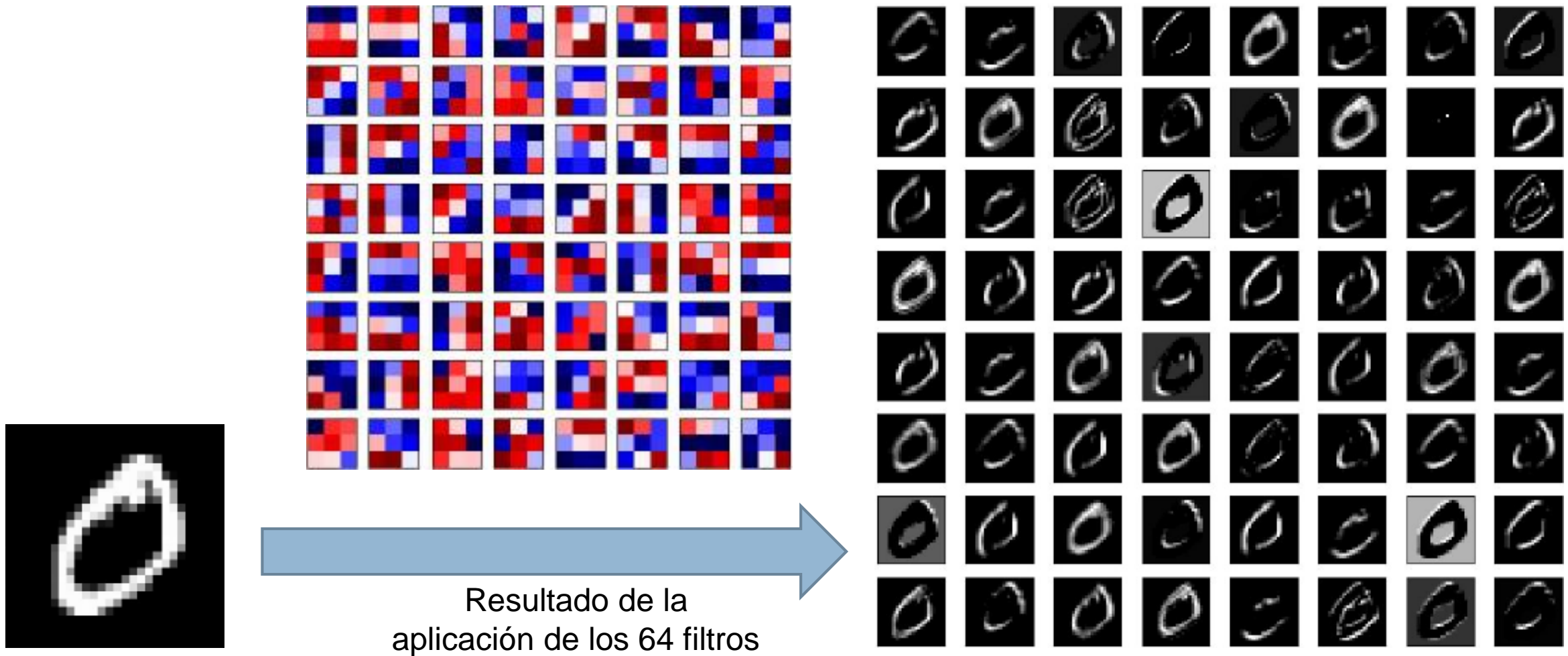
$$N_f * T * U$$

64 \* 26 \* 26

↑  
Cantidad de  
filtros

└──────────┘  
Tamaño de la  
imagen filtrada

# Capa Conv2D de MNIST



# MNIST

```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu", input_shape=input_shape))
model.add(Flatten())
model.add(Dense(15, activation='tanh'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

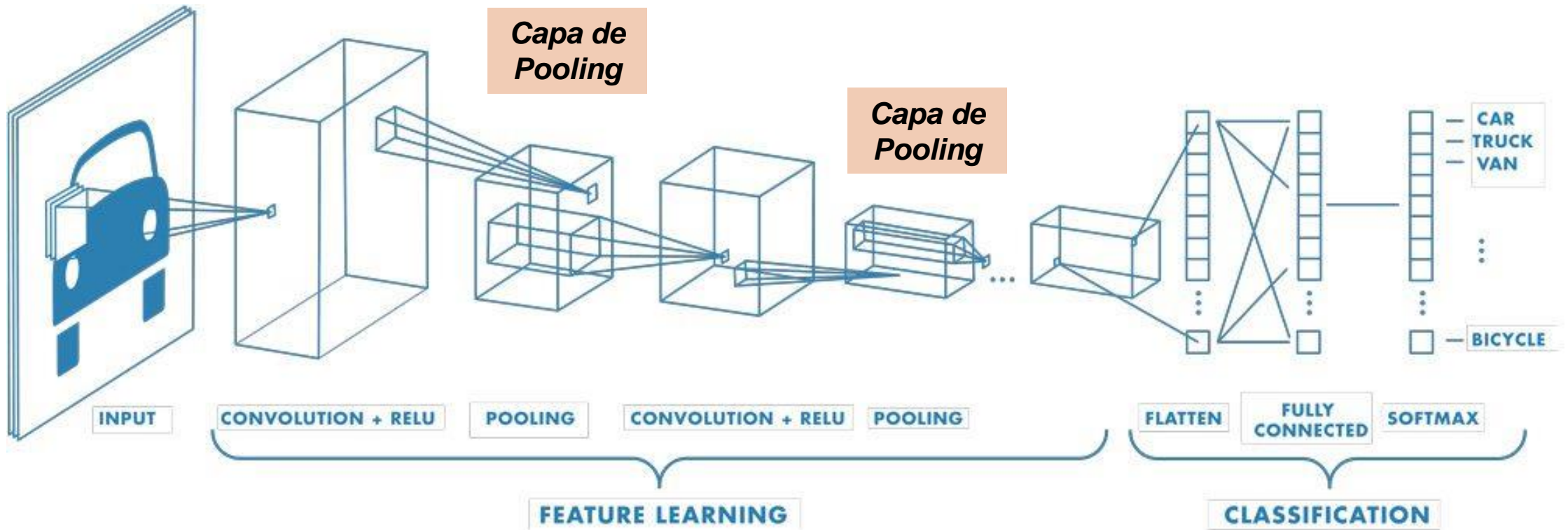
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
flatten (Flatten)	(None, 43264)	0
dense (Dense)	(None, 15)	648975
dense_1 (Dense)	(None, 10)	160
Total params: 649,775		

Cantidad de parámetros o pesos de la capa oculta del multiperceptrón

$$43264 * 15 + 15$$

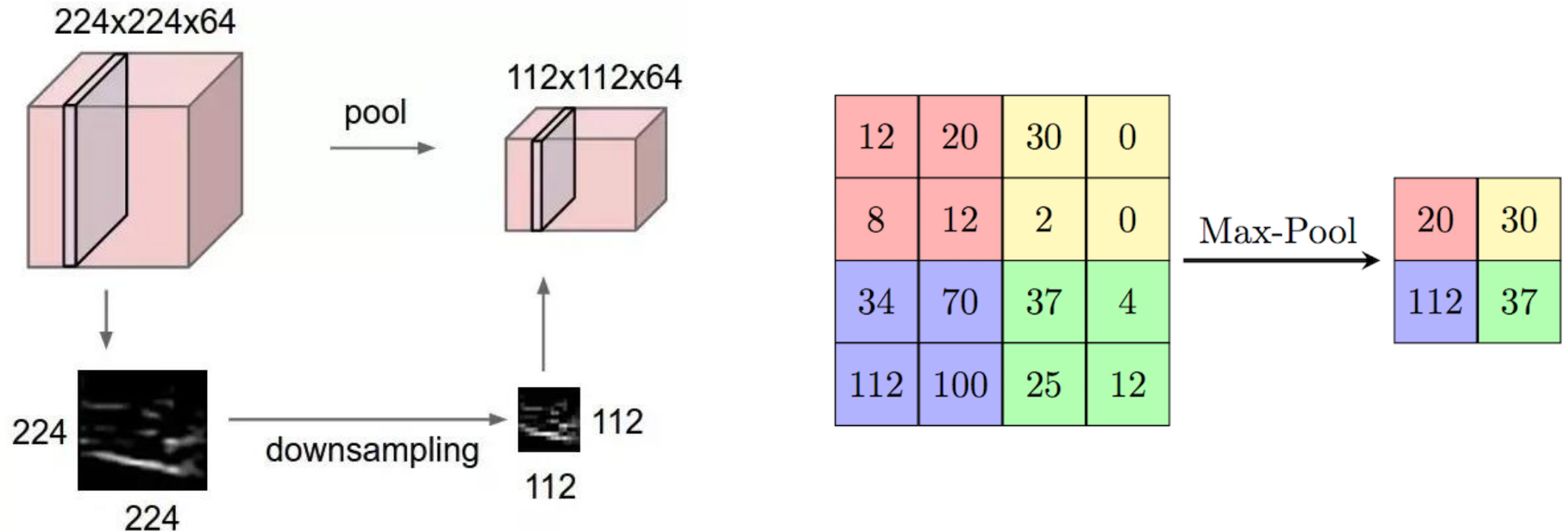


# Red Neuronal Convolucional

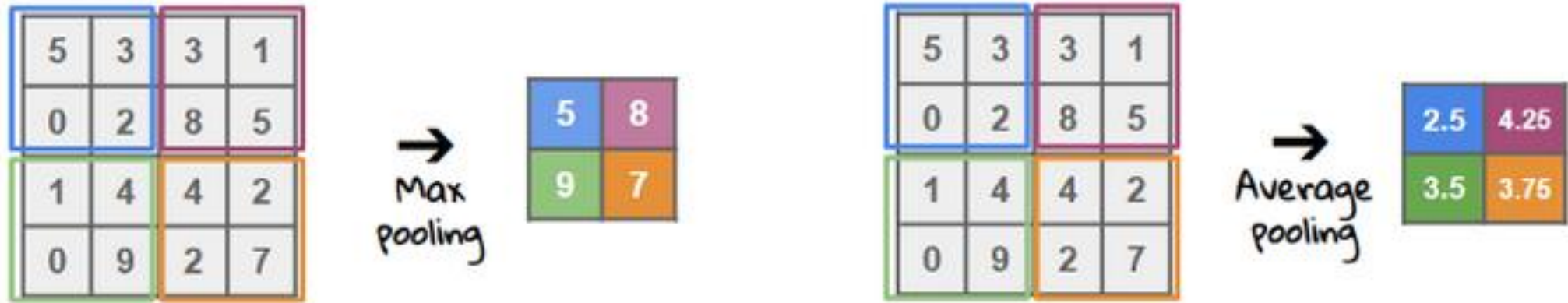


# Capa de pooling

- La capa de pooling reduce el tamaño de la salida de la capa convolucional. Se trata de una convolución con un stride igual al tamaño del kernel que calcula la función sobre todos los pixels.



# Pooling



- La reducción de tamaño permite eliminar parte del ruido y extraer datos más significativos.
- Reduce el exceso de ajuste y acelera el cálculo

# Capa MaxPooling2D

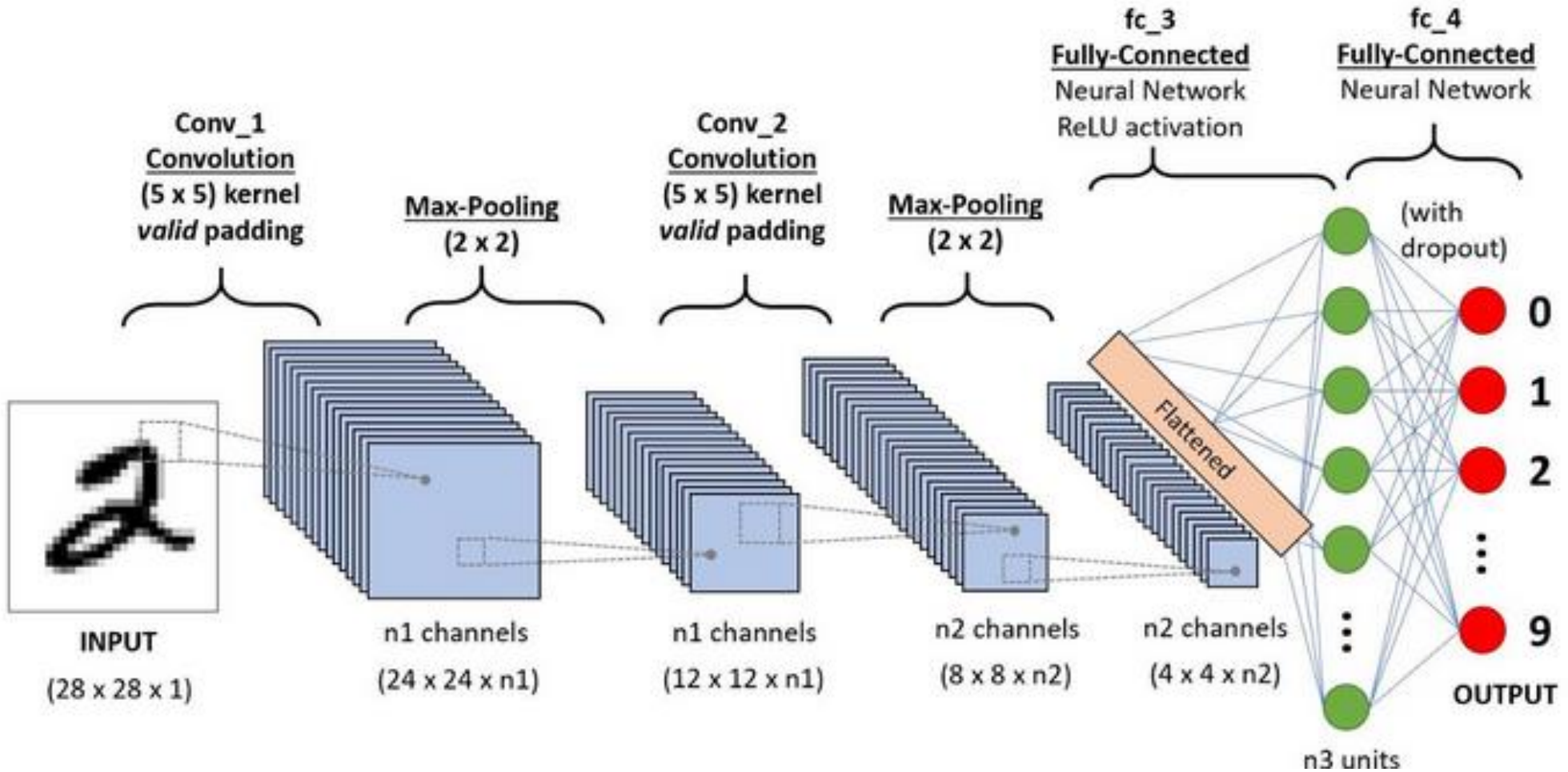
```
model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation="relu", input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(15, activation='tanh'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
flatten_1 (Flatten)	(None, 10816)	0
dense_2 (Dense)	(None, 15)	162255
dense_3 (Dense)	(None, 10)	160
Total params: 163,055		

Luego de aplanar la cantidad de entradas se redujo un 75%

Antes eran 43264 y ahora quedaron 10816

# Reconocimiento de dígitos MNIST



# Resumen

- Las capas convolucionales 2D contienen los filtros que, se entrenan junto con los demás parámetros de la red y que permiten detectar características.
- El resultado de aplicar un filtro o máscara es un mapa de características de  $T_x U_x 1$  dependiendo del padding, stride y tamaño de kernel usados.
- La salida de la capa convolucional es una nueva “imagen” de  $T_x U_x F$  siendo  $F$  el número de filtros.
- Las capas de pooling reducen la dimensionalidad haciendo más rápido y eficaz el entrenamiento.
- Se suelen intercalar capas de convolucionales con capas de pooling hasta llegar a la parte feedforward donde para ingresar “aplanamos” las “imágenes”.

# Ejercicio

La base de datos MNIST contiene imágenes de  $28 \times 28$ , en escala de grises, de números escritos a mano.

- Revisar la red formada sólo por una capa softmax.
- Implementar un MLP con un 97 o 98% de accuracy.
- Implementar una CNN con un accuracy del 99% sobre los datos de testeo.



## □ Características

- ▣ Presenta poca variabilidad entre ejemplos de una misma clase.
- ▣ Las imágenes se encuentran en blanco y negro.
- ▣ Las imágenes están perfectamente centradas.
- ▣ Los objetos, es decir, números, no cambian con las condiciones de iluminación, ángulo, etcétera.

## □ Ventajas

- ▣ Imágenes pequeñas que facilitan la rápida experimentación.
- ▣ Datos balanceados que permiten utilizar métricas de clasificación sencillas.



# CIFAR-10



- Se compone de 60.000 imágenes de 32x32x3, en espacio RGB.
- Hay 50.000 imágenes de entrenamiento y 10.000 imágenes de prueba.
- Hay 10 clases, donde cada una está representada por 6.000 imágenes.
- Las clases son mutuamente excluyentes

# Finger (<https://www.kaggle.com/koryakinp/fingers>)



- El objetivo es construir una CNN capaz de contar los dedos.
  - ▣ Contiene 21600 imágenes de dedos de la mano izquierda y derecha.
  - ▣ Todas las imágenes son de 128 por 128 píxeles.
  - ▣ Conjunto de entrenamiento: 18000 imágenes
  - ▣ Conjunto de pruebas: 3600 imágenes
  - ▣ Las imágenes están centradas por el centro de masa
  - ▣ Patrón de ruido en el fondo

# Modelos pre-entrenados

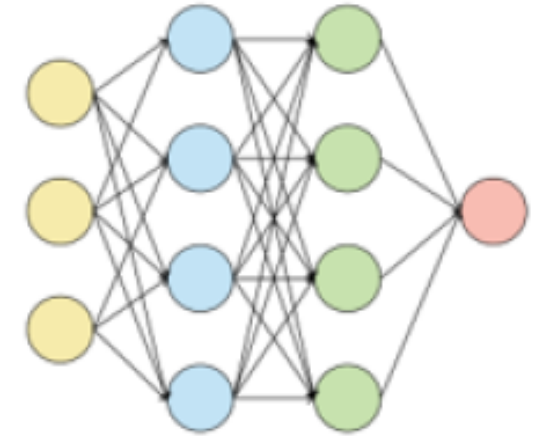
## BASE CONVOLUCIONAL (extracción de características)



Convolutional Layer

Pooling and Flattening

## CLASIFICADOR



Artificial Neural Network




# ImageNet

- BBDD utilizada para reconocimiento de objetos en imágenes.
- Contiene 14 millones de imágenes etiquetadas con nombres de objetos de más de 20.000 categorías.
- **ILSVRC**
  - ▣ ~1.2 millones de imágenes de entrenamiento.
  - ▣ 50.000 imágenes de validación
  - 100.000 imágenes de prueba




# Redes pre-entrenadas en Keras

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base.
- Se utilizan estas redes y se ajusta un clasificador (ANN):
  - VGG16 
  - Inception V3
  - Xception
  - ResNet50
  - MobileNet

*Red neuronal convolucional con 16 capas propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford*

*El modelo se presentó al Desafío de Reconocimiento Visual a Gran Escala de ImageNet (ILSVRC) en 2014.*

# Redes pre-entrenadas en Keras

- Las siguientes redes pre-entrenadas pueden ser consideradas como las capas convolucionales base.
- Se utilizan estas redes y se ajusta un clasificador (ANN):
  - ▣ VGG16
  - ▣ Inception V3
  - ▣ Xception
  - ▣ ResNet50 
  - ▣ MobileNet

*Red neuronal convolucional entrenada con el conjunto de datos de ImageNet que tiene 50 capas y que ganó el primer puesto en el ILSVRC en 2015..*

# VGG16 - Carga del modelo

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input
```

```
# Descargue manualmente el modelo y colóquelo en el directorio
# C:\Users\nombre de usuario\.keras\models
# vgg16_weights_tf_dim_ordering_tf_kernels.h5
# vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
# Cargar modelo e imprimir
```

```
model = VGG16()
```

```
print(model.summary())
```



```
predictions (Dense)              (None, 1000)
=====
Total params: 138,357,544
```

# Cargando la imagen a reconocer

*# Cargar una imagen de prueba*

```
image = load_img("tigre.jpg", target_size=(224, 224))
```

Tamaño  
224 x 224



# Cargando la imagen a reconocer

*# Cargar una imagen de prueba*

```
image = load_img("tigre.jpg", target_size=(224, 224))
```

*# Convertir a matriz*

```
image = img_to_array(image)
```



Tamaño  
224 x 224 x 3

# Cargando la imagen a reconocer

*# Cargar una imagen de prueba*

```
image = load_img("tigre.jpg", target_size=(224, 224))
```

*# Convertir a matriz*

```
image = img_to_array(image)
```

*# Reformar en 4D*



```
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

**Tamaño**  
**1 x 224 x 224 x 3**

# La imagen a reconocer

*# Cargar una imagen de prueba*

```
image = load_img("tigre.jpg", target_size=(224, 224))
```

*# Convertir a matriz*

```
image = img_to_array(image)
```

*# Reformar en 4D*

```
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

*# Imagen de preproceso*

```
image = preprocess_input(image) 
```

*Las imágenes se convierten de RGB a BGR y, a continuación, cada canal de color se centra en cero con respecto al conjunto de datos de ImageNet, sin escalar.*

# Resultado de la predicción

*# Predicción de la red*

```
predict_result = model.predict(image)
```



**Resultado de la capa de salida**  
**Tamaño: 1 x 1000**

# Resultado de la predicción

```
# Predicción de la red
```

```
predict_result = model.predict(image)
```

```
# Resultados de predicción de análisis
```

```
label = decode_predictions(predict_result) ←
```

Lista con los 5 mejores resultados

Índi ▲	Tipo	Tamaño	Valor
0	tuple	3	('n02129604', 'tiger', 0.92024356)
1	tuple	3	('n02123159', 'tiger_cat', 0.07646653)
2	tuple	3	('n02128925', 'jaguar', 0.0027824175)
3	tuple	3	('n02127052', 'lynx', 0.00025741145)
4	tuple	3	('n02128385', 'leopard', 0.0001807782)

# Resultado de la predicción

```
# Predicción de la red
```

```
predict_result = model.predict(image)
```

```
# Resultados de predicción de análisis
```

```
label = decode_predictions(predict_result)
```

```
# Imprima las tres categorías con mayor probabilidad
```

```
for idx in range(0, 3):
```

```
    print ("Categoría:% s Probabilidad:% 0.4f"% (label[0][idx][1],  
                                                  label[0][idx][2]))
```

```
Categoría:tiger Probabilidad: 0.9202  
Categoría:tiger_cat Probabilidad: 0.0765  
Categoría:jaguar Probabilidad: 0.0028
```



# Ajuste fino (fine-tuning) de modelos pre-entrenados

- Se busca reusar la base convolucional y mejorar la respuesta del clasificador en una tarea específica.
- Pasos para efectuar el **ajuste fino**
  - ▣ Añadir un clasificador (RNA) sobre un sistema preentrenado.
  - ▣ Fijar la base convolucional y entrenar la red.
  - ▣ Entrenar conjuntamente el clasificador añadido y la base convolucional.

*Consultar Capítulo 8 del libro “The Deep Learning with Keras Workshop. An Interactive Approach to Understanding Deep Learning with Keras (2020)”*