

# Redes Convolucionales

**Matias Vera - Juan Zuloaga - Lautaro Estienne**

Centro de Simulación Computacional para Aplicaciones Tecnológicas

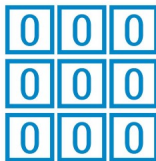
# ¿Que es un tensor?



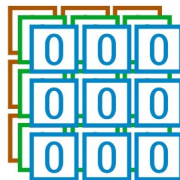
SCALAR



VECTOR



MATRIX



TENSOR

Las redes neuronales convolucionales son redes que combinan los siguientes tipos de layers:

- Dense or Fully Connected.
- Flatten.
- Convolutional.
- Pooling.
- Accesorios: ej. Dropout.

# ConvNets

Las redes neuronales convolucionales son redes que combinan los siguientes tipos de layers:

- **Dense or Fully Connected.** ✓
- **Flatten.** ✓
- Convolutional.
- Pooling.
- **Accesorios: ej. Dropout.** ✓

## Flatten layer

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$



$x_{1,1}$
$x_{1,2}$
$x_{1,3}$
$x_{2,1}$
$x_{2,2}$
$x_{2,3}$
$x_{3,1}$
$x_{3,2}$
$x_{3,3}$

Pierdo noción espacial.

# Dense Layer

$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]} \mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$w_{1,4}$	$w_{1,5}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$	$w_{2,5}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$	$w_{3,4}$	$w_{3,5}$
$w_{4,1}$	$w_{4,2}$	$w_{4,3}$	$w_{4,4}$	$w_{4,5}$

•

$h_1$
$h_2$
$h_3$
$h_4$
$h_5$

+

$b_1$
$b_2$
$b_3$
$b_4$

# Dense Layer

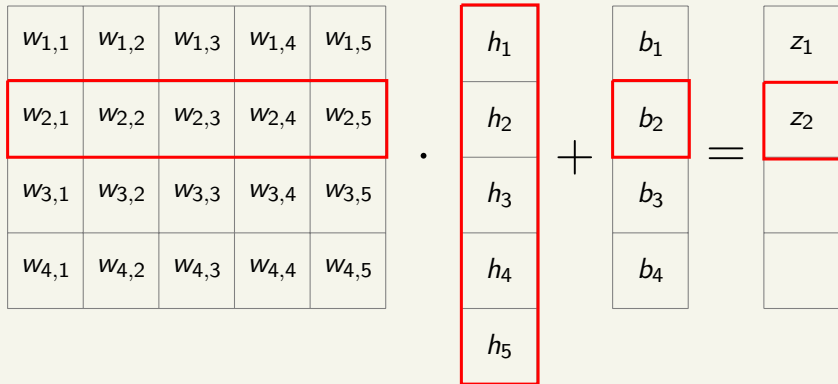
$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]} \mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$w_{1,4}$	$w_{1,5}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$	$w_{2,5}$
$w_{3,1}$	$w_{3,2}$	$w_{3,3}$	$w_{3,4}$	$w_{3,5}$
$w_{4,1}$	$w_{4,2}$	$w_{4,3}$	$w_{4,4}$	$w_{4,5}$

$$\begin{matrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{matrix} \cdot \begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{matrix} = \begin{matrix} z_1 \\ \\ \\ \end{matrix}$$

# Dense Layer

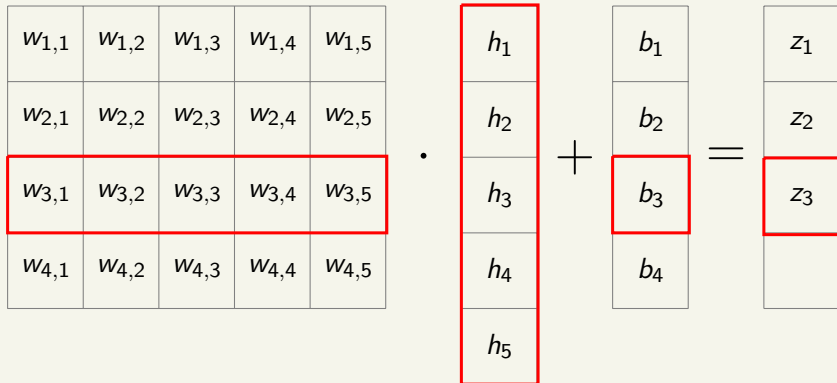
$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]} \mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$





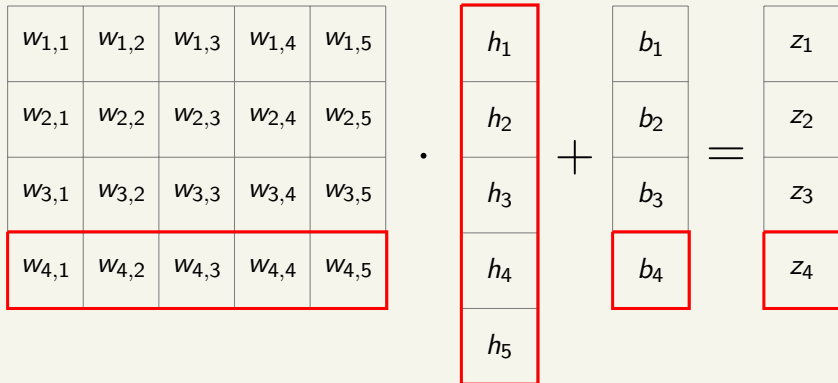
# Dense Layer

$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]} \mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$



## Dense Layer

$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]} \mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$



Para generar cada neurona se necesita toda una fila de  $\mathbf{W}^{[l]}$  (es decir una cantidad de parámetros igual a la dimensión de la entrada) y un solo elemento de  $\mathbf{b}^{[l]}$ .

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.


3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

 $=$ 


(filtro)



**Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flipéa)!**

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2			



**Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!**

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4		



Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flipea)!

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	



**Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!**

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	2



Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flipea)!

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	2
0			



**Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!**



## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	2
0	-5		



**Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!**

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	2
0	-5	2	



Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!

## Convolutional Layers

Un convolutional layer básicamente reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1


\*

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

(filtro)

=

2	-4	0	2
0	-5	2	1
0	-2	3	-1
4	0	0	0



Operación que se le dice convolución pero no lo es.  
En realidad es una correlación cruzada (no flípea)!

## Convolutional layers: Características de cada neurona

$$\text{Neurona} \rightarrow \sigma(\mathbf{x} * \text{FILTRO} + b \cdot \mathbf{1})$$

- No pierde noción espacial (no necesita hacer flatten).
- Cada neurona generada es una matriz (en lugar de un escalar).
- Necesita menos parámetros (en el ejemplo para generar una neurona a partir de una entradas de  $6 \times 6 = 36$  bastaron con  $3 \times 3 = 9$  parámetros).
- A pesar que la salida del filtrado es una matriz, se suele usar un solo escalar por neurona para el bias (al igual que el dense).

# Convolutional layers: Características de cada neurona

$$\text{Neurona} \rightarrow \sigma(\mathbf{x} * \text{FILTRO} + b \cdot \mathbf{1})$$

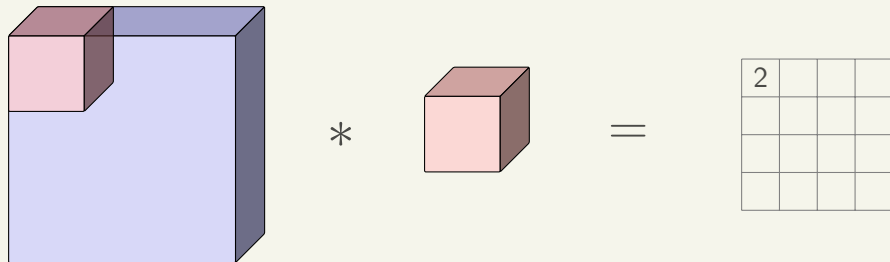
- No pierde noción espacial (no necesita hacer flatten).
- Cada neurona generada es una matriz (en lugar de un escalar).
- Necesita menos parámetros (en el ejemplo para generar una neurona a partir de una entradas de  $6 \times 6 = 36$  bastaron con  $3 \times 3 = 9$  parámetros).
- A pesar que la salida del filtrado es una matriz, se suele usar un solo escalar por neurona para el bias (al igual que el dense).

## Sobre los filtros

Por un contexto histórico, relacionado con el mundo de *computer vision*, los filtros suelen tener el mismo alto que ancho y ser una dimensión impar ( $1 \times 1$ ,  $3 \times 3$ ,  $5 \times 5$ , etc.).

$$\dim(\text{Neurona}) = \dim(\mathbf{x}) - \dim(\text{Filtro}) + 1$$

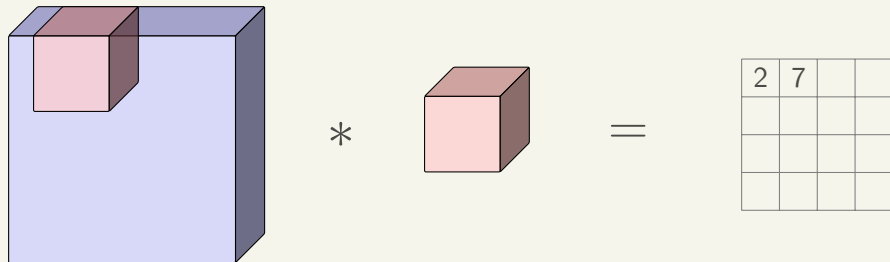
## ¿Que hacer cuando la entrada es un tensor?



### Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en  $(n, 4, 4, d)$ , donde  $d$  es la cantidad de filtros utilizados (cada uno con su bias).

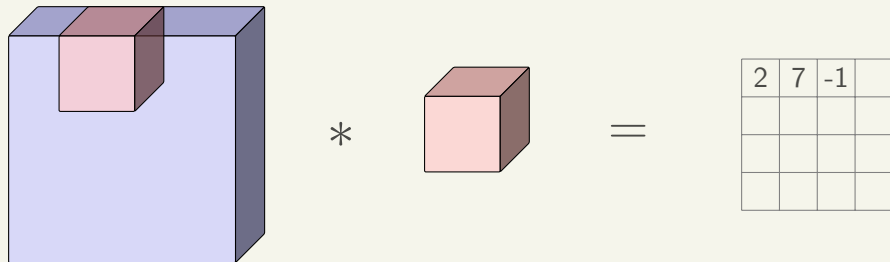
## ¿Que hacer cuando la entrada es un tensor?



### Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en  $(n, 4, 4, d)$ , donde  $d$  es la cantidad de filtros utilizados (cada uno con su bias).

## ¿Que hacer cuando la entrada es un tensor?



### Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en  $(n, 4, 4, d)$ , donde  $d$  es la cantidad de filtros utilizados (cada uno con su bias).



## Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

Padding = 0

## Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

0	0	0	0	0	0	0	0
0	3	1	0	2	1	0	0
0	2	0	1	3	2	1	0
0	0	1	2	1	0	3	0
0	4	0	3	2	2	1	0
0	1	1	0	1	0	1	0
0	3	2	1	0	2	1	0
0	0	0	0	0	0	0	0

Padding = 1

## Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	3	1	0	2	1	0	0	0
0	0	2	0	1	3	2	1	0	0
0	0	0	1	2	1	0	3	0	0
0	0	4	0	3	2	2	1	0	0
0	0	1	1	0	1	0	1	0	0
0	0	3	2	1	0	2	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Padding = 2

$$\dim(\text{Neurona}) = \dim(\mathbf{x}) - \dim(\text{Filtro}) + 1 + 2 \cdot \text{Padding}$$

# Tipos de padding habituales

- *Valid*:  $\text{Padding} = 0$ .
- *Same*:  $\text{Padding} = \frac{\dim(\text{Filtro}) - 1}{2}$ .
- *Full*:  $\text{Padding} = \dim(\text{Filtro}) - 1$ .

# Tipos de padding habituales

- *Valid*:  $\text{Padding} = 0$ .
- *Same*:  $\text{Padding} = \frac{\dim(\text{Filtro}) - 1}{2}$ .
- *Full*:  $\text{Padding} = \dim(\text{Filtro}) - 1$ .

## Padding Same

El objetivo de padding same es que la neurona tenga la misma dimensión que la entrada  $\dim(\text{Neurona}) = \dim(\mathbf{x})$ .

# Tipos de padding habituales

- *Valid*:  $\text{Padding} = 0$ .
- *Same*:  $\text{Padding} = \frac{\dim(\text{Filtro}) - 1}{2}$ .
- *Full*:  $\text{Padding} = \dim(\text{Filtro}) - 1$ .

## Padding Same

El objetivo de padding same es que la neurona tenga la misma dimensión que la entrada  $\dim(\text{Neurona}) = \dim(\mathbf{x})$ .

## Padding Full

Padding full es el mayor padding que tiene sentido (sin agregar ceros triviales). No suele alcanzar buenos resultados, por lo que rara vez se utiliza.

## Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

(filtro)

 $=$ 


# Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

 $=$ 

2	

(filtro)

Stride = 2



# Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

 $=$ 

2	0

(filtro)

Stride = 2

## Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

 $=$ 

2	0
0	

(filtro)

Stride = 2

## Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

 $*$ 

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

(filtro)

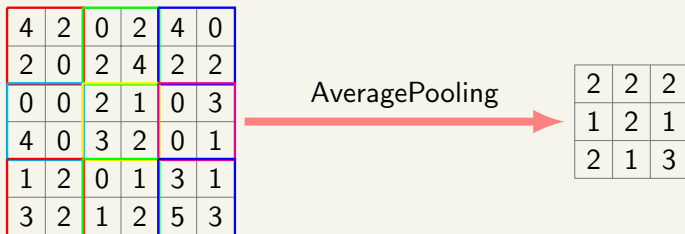
 $=$ 

2	0
0	3

Stride = 2

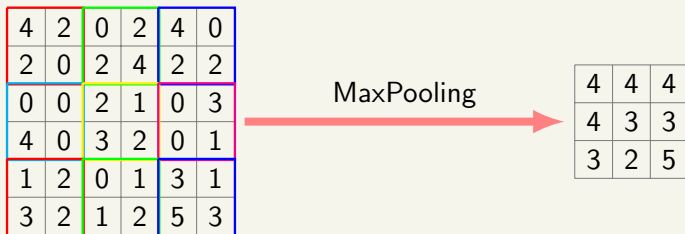
## Pooling

El uso de Pooling puede entenderse como usar la información conocida de que la clasificación de imágenes no depende de rotaciones o traslaciones pequeñas.



## Pooling

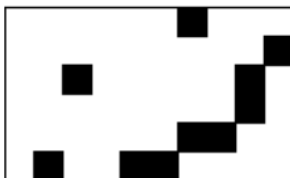
El uso de Pooling puede entenderse como usar la información conocida de que la clasificación de imágenes no depende de rotaciones o traslaciones pequeñas.



## Ejemplo de MaxPooling



```
0 1 0 1 0  
1 0 0 0 1  
1 1 1 1 1
```



```
0 0 0 1 1  
0 1 0 0 1  
1 0 1 1 0
```

Antes coincidían solo 2 píxeles oscuros, luego del max-pooling coinciden 5

# ¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(units=120, activation='relu'))  
model.add(tf.keras.layers.Dense(units=84, activation='relu'))  
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))  
model.summary()
```

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 14, 14, 6)	18
average_pooling2d_14 (AveragePooling2D)	(None, 7, 7, 6)	0
conv2d_18 (Conv2D)	(None, 10, 10, 16)	464
average_pooling2d_15 (AveragePooling2D)	(None, 5, 5, 16)	0
flatten_7 (Flatten)	(None, 400)	0
dense_21 (Dense)	(None, 120)	48000
dense_22 (Dense)	(None, 84)	33600
dense_23 (Dense)	(None, 10)	850

# ¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(units=120, activation='relu'))  
model.add(tf.keras.layers.Dense(units=84, activation='relu'))  
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))  
model.summary()
```

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 26, 26, 6)	
average_pooling2d_14 (AveragePooling2D)	(None, 13, 13, 6)	
conv2d_18 (Conv2D)	(None, 11, 11, 16)	
average_pooling2d_15 (AveragePooling2D)	(None, 5, 5, 16)	
flatten_7 (Flatten)	(None, 400)	
dense_21 (Dense)	(None, 120)	
dense_22 (Dense)	(None, 84)	
dense_23 (Dense)	(None, 10)	



# ¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()  
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))  
model.add(tf.keras.layers.AveragePooling2D())  
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(units=120, activation='relu'))  
model.add(tf.keras.layers.Dense(units=84, activation='relu'))  
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))  
model.summary()
```

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 26, 26, 6)	60
average_pooling2d_14 (AveragePooling2D)	(None, 13, 13, 6)	0
conv2d_18 (Conv2D)	(None, 11, 11, 16)	880
average_pooling2d_15 (AveragePooling2D)	(None, 5, 5, 16)	0
flatten_7 (Flatten)	(None, 400)	0
dense_21 (Dense)	(None, 120)	48120
dense_22 (Dense)	(None, 84)	10164
dense_23 (Dense)	(None, 10)	850