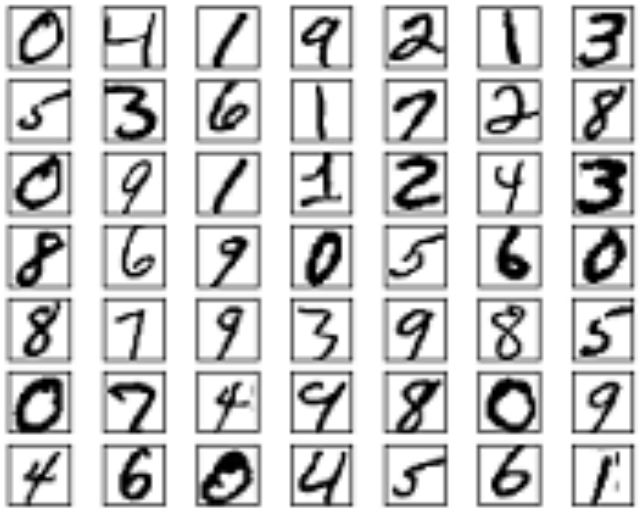
Inteligencia Artificial

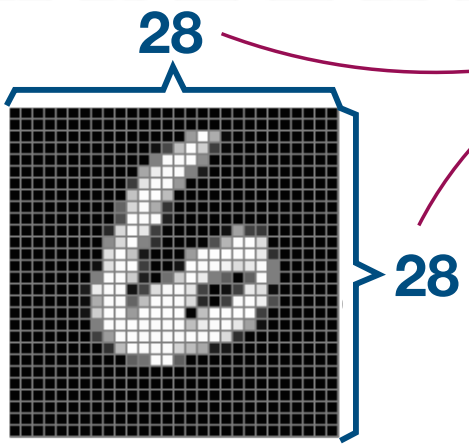
Machine Learning - (Aprendizaje Automático)

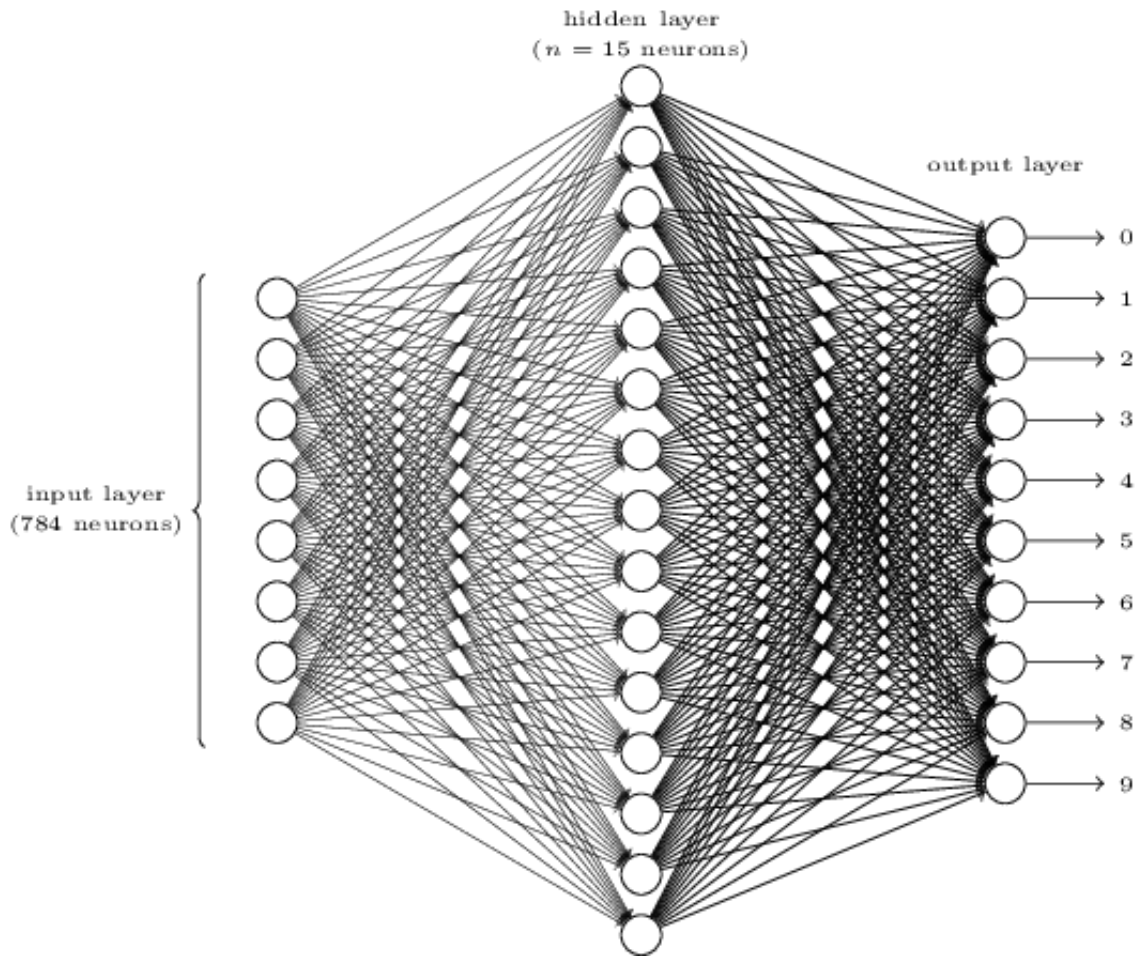
Redes Neuronales Convolucionales

Redes Neuronales - Ejemplo MNIST

https://neuralnetworksanddeeplearning.com/chap1.html



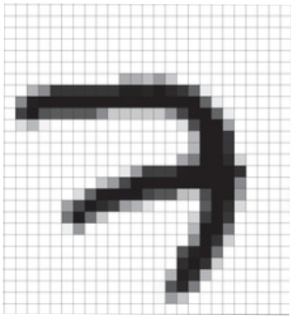


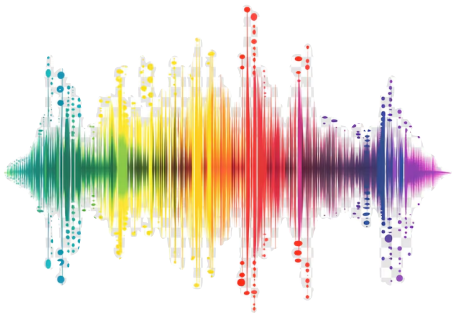


Redes Neuronales - Algunos inconvenientes de las ANN

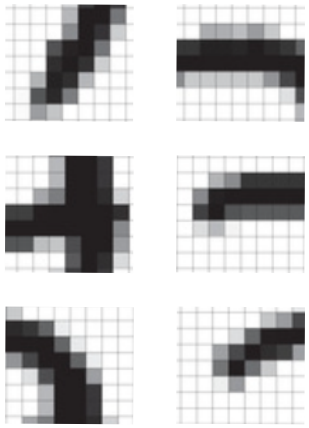
- Requieren entradas vectorizadas (1D). Si los datos son imágenes, hay que “aplanarlas” perdiendo la estructura espacial. Esto conduce a que no son invariantes a traslaciones, rotaciones o escalados; pequeños cambios pueden cambiar la salida completamente.
- Están completamente conectadas, esto puede ser un inconvenientes cuando las entradas son grandes. Esto genera sobreajuste y un alto costo computacional.
- Las características están implícitas en las entradas.
- No permiten diseñar fácilmente qué características se están aprendiendo en cada capa.
- En general necesitan más datos para generalizar correctamente, ya que no aprovechan la estructura local de las entradas.

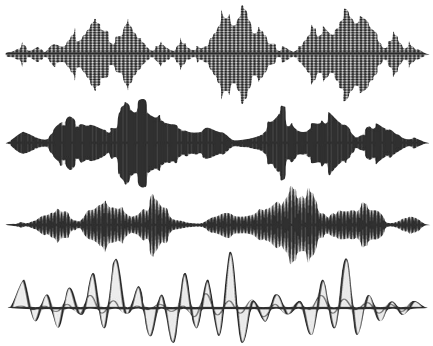
Redes Neuronales - Cómo capturar características



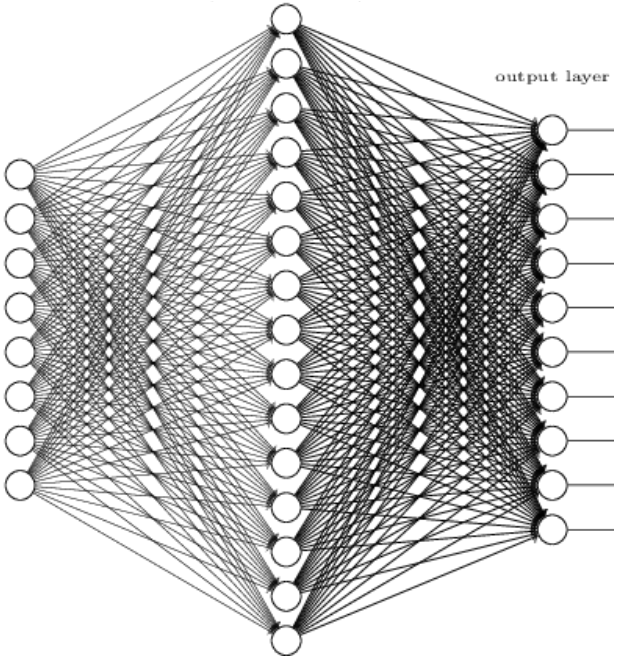


Patrones del dominio





Aprender de los patrones

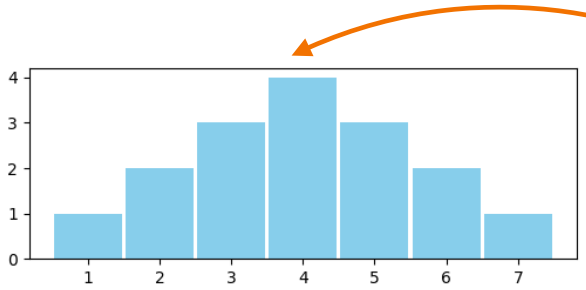


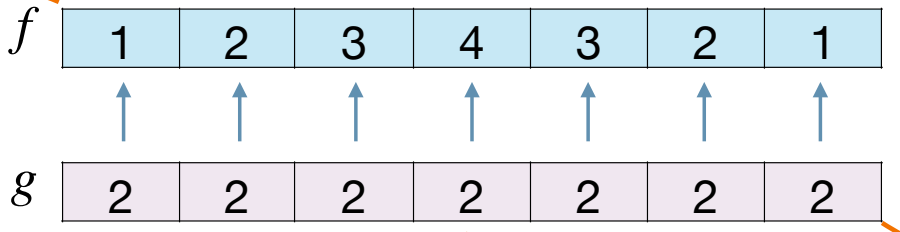
Redes Neuronales - Convoluciones (resaltando patrones)

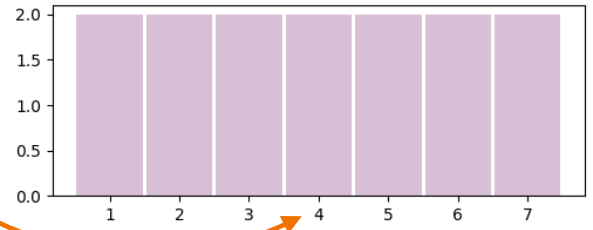
Una **convolución** es una operación que combina dos funciones f y g para producir una tercera función que expresa cómo una de ellas modifica o “suaviza” a la otra.

Si asociamos a la función f representando el dato de entrada y la segunda g como el patrón a detectar, la convolución modifica la entrada proyectando el patrón sobre ella.

Imaginemos dos secuencias de números.

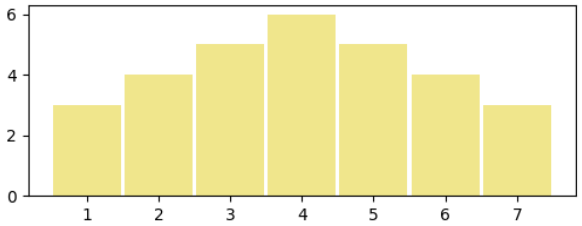






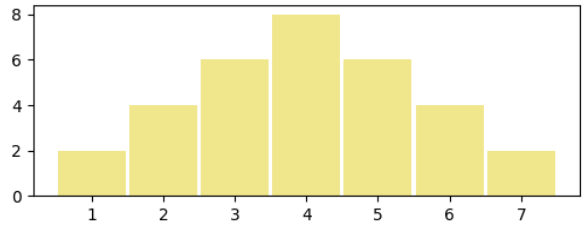
$f+g$

3	4	5	6	5	4	3



$f \cdot g$

2	4	6	8	6	4	2

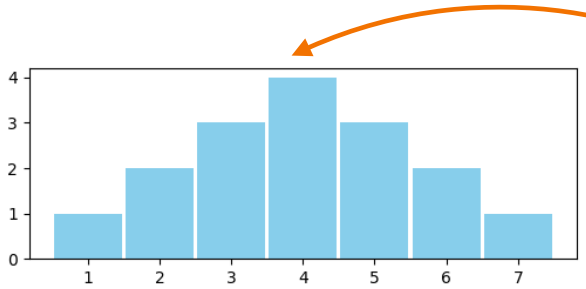


Redes Neuronales - Convoluciones (resaltando patrones)

Una **convolución** es una operación que combina dos funciones f y g para producir una tercera función que expresa cómo una de ellas modifica o “suaviza” a la otra.

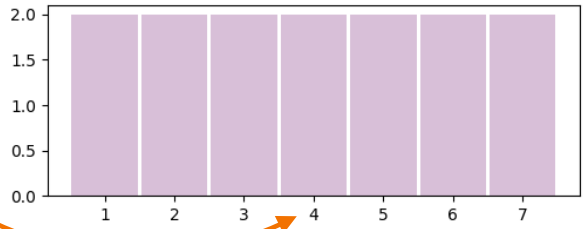
Si asociamos a la función f representando el dato de entrada y la segunda g como el patrón a detectar, la convolución modifica la entrada proyectando el patrón sobre ella.

Imaginemos dos secuencias de números.



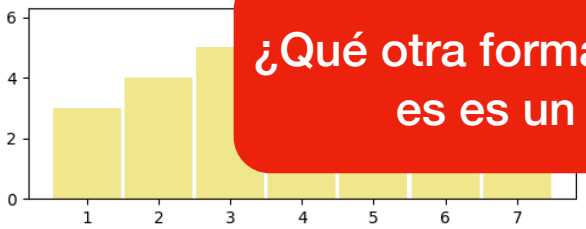
f | 1 | 2 | 3 | 4 | 3 | 2 | 1 |

g | 2 | 2 | 2 | 2 | 2 | 2 | 2 |



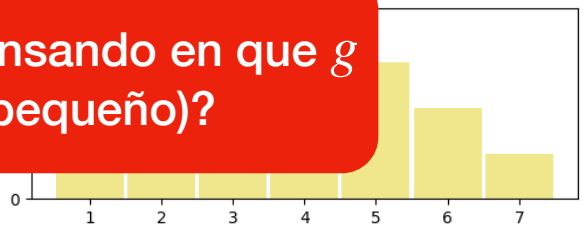
$f + g$

3	4	5	6	5	4	3



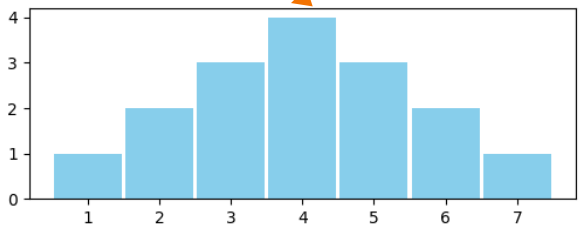
$f \cdot g$

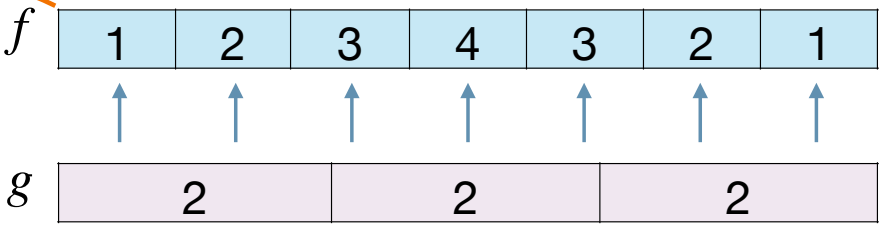
2	4	6	8	6	4	2

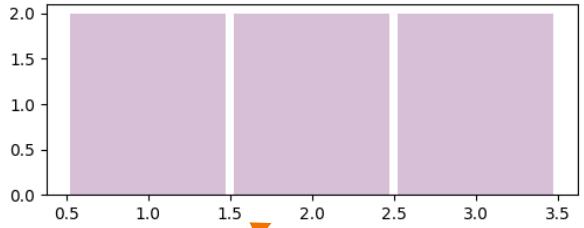


¿Qué otra forma podemos combinarlos (pensando en que g es es un patrón que podría ser más pequeño)?

Redes Neuronales - Convoluciones (resaltando patrones)



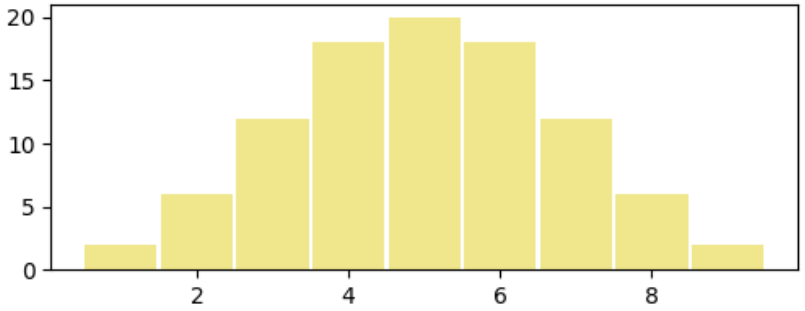




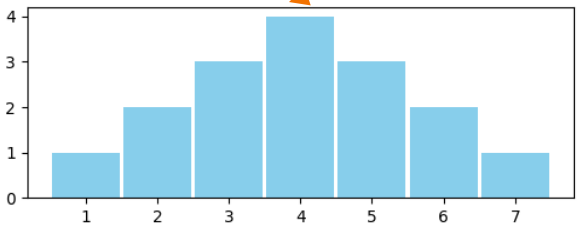
$f * g$

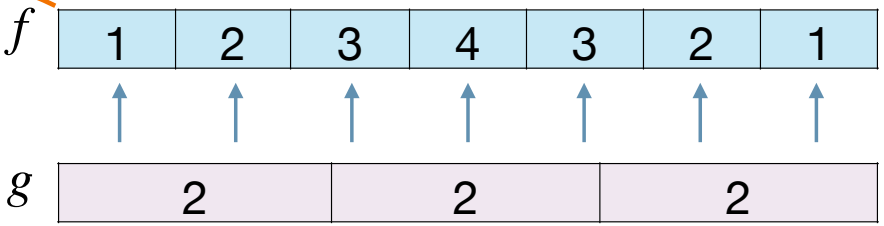
2	6	12	18	20	18	12	6	2

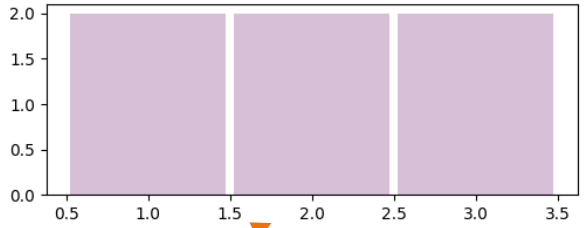
Convolución



Redes Neuronales - Convoluciones (resaltando patrones)





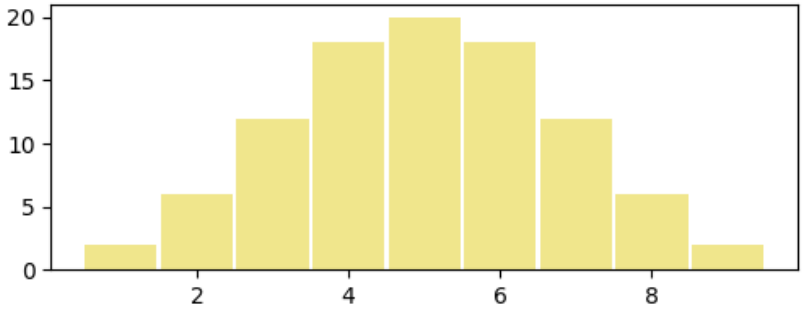


$f * g$

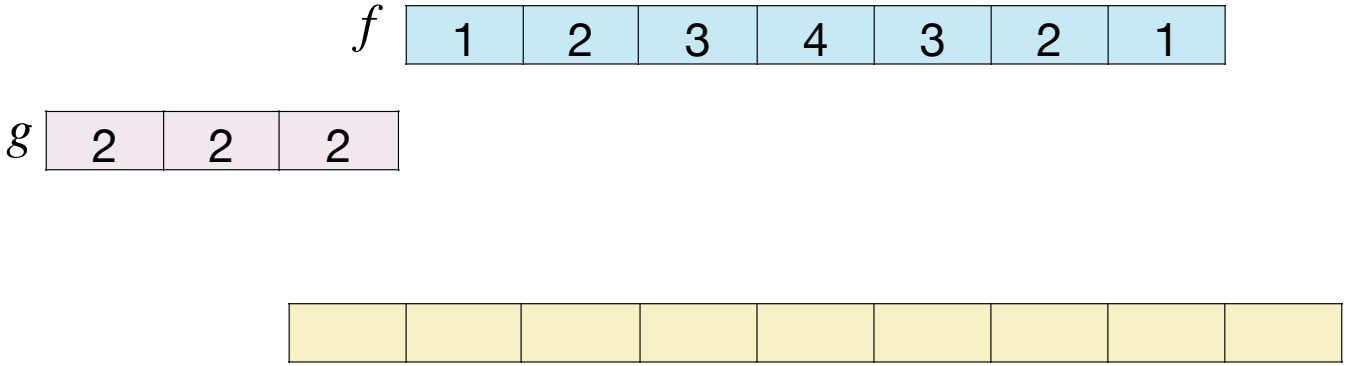
2	6	12	18	20	18	12	6	2

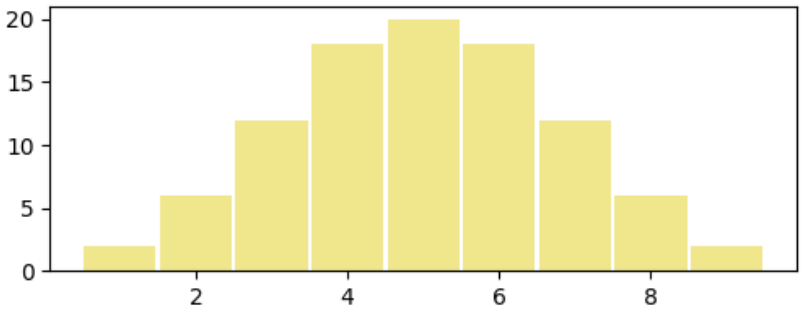
Convolución

¿Cómo se calcula?

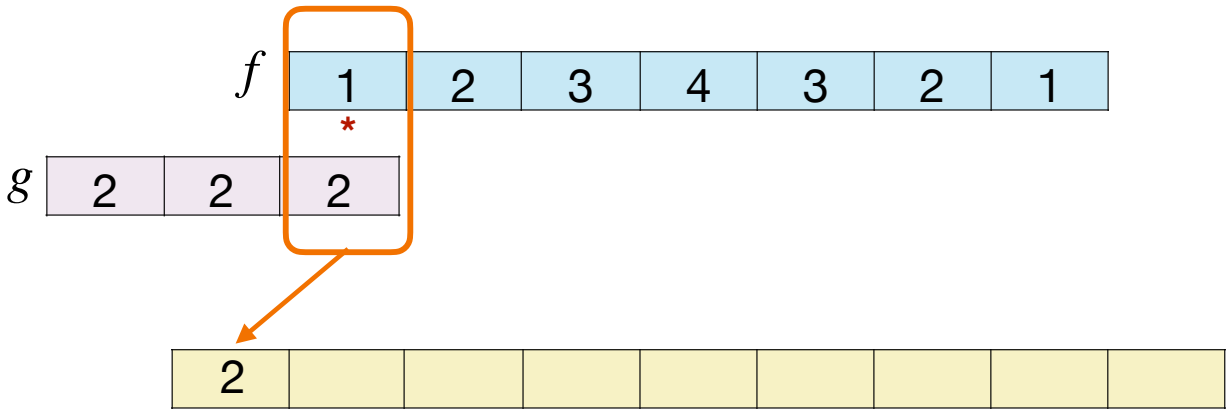


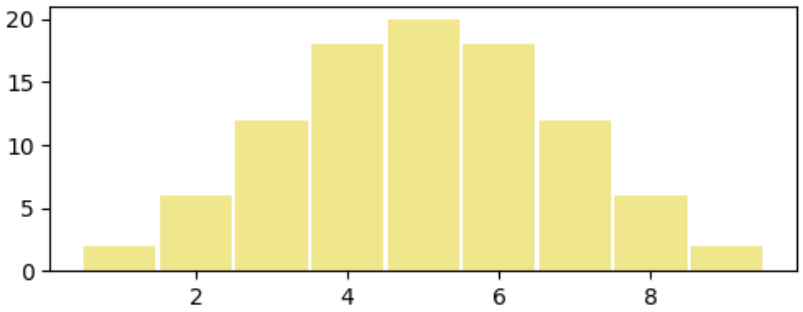
Redes Neuronales - Convoluciones (resaltando patrones)



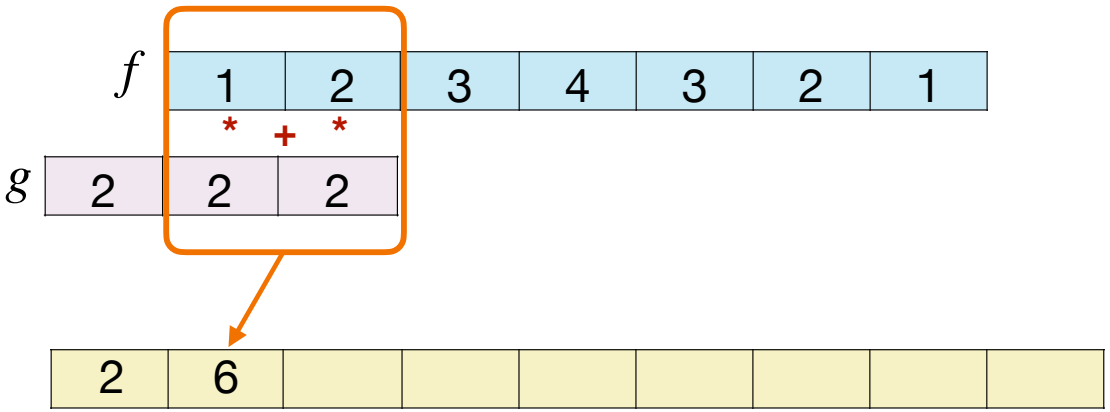


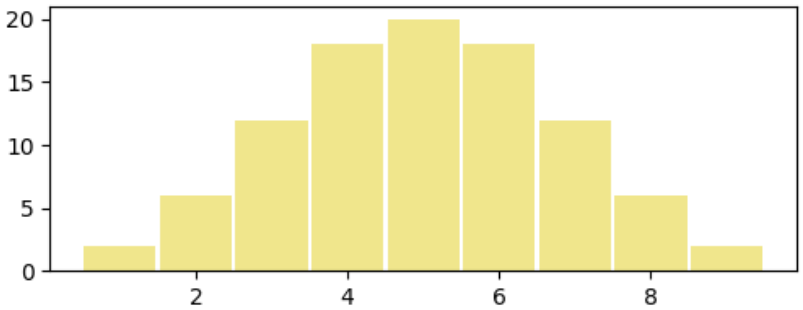
Redes Neuronales - Convoluciones (resaltando patrones)



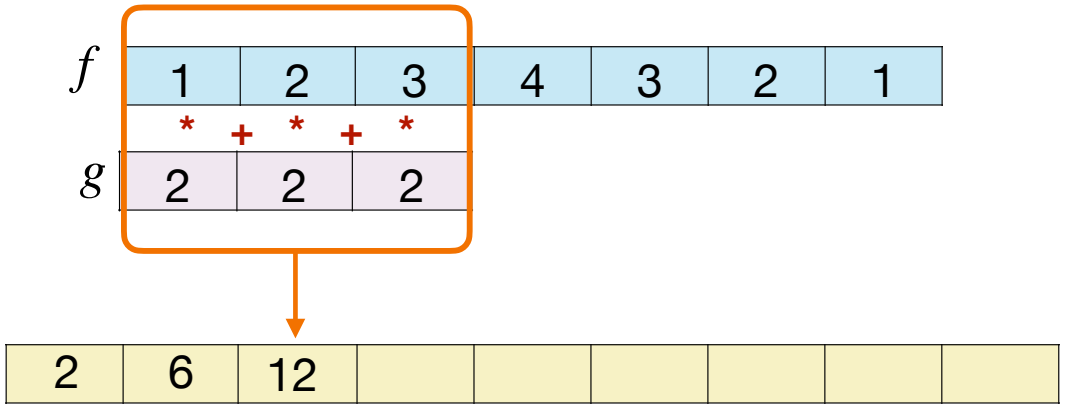


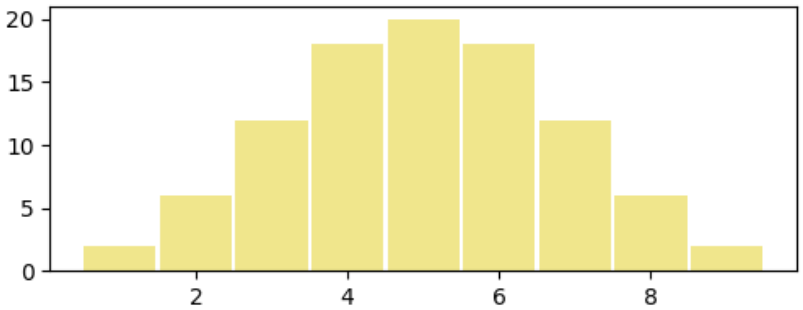
Redes Neuronales - Convoluciones (resaltando patrones)



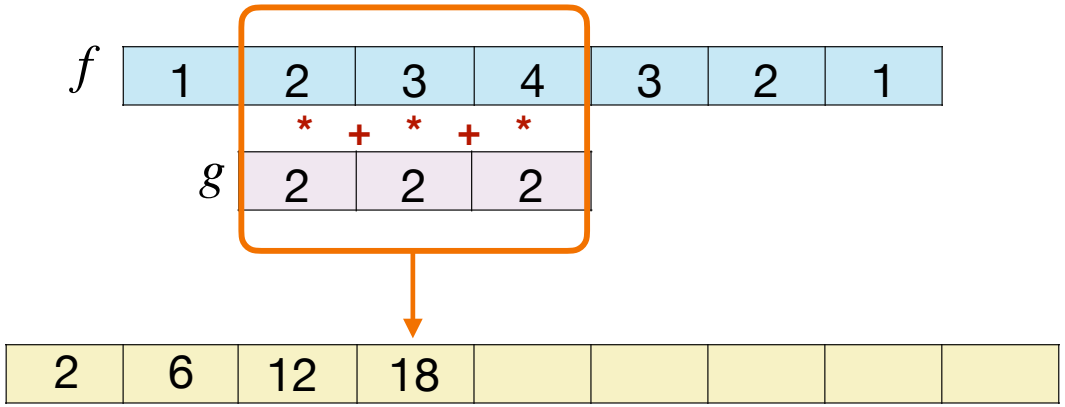


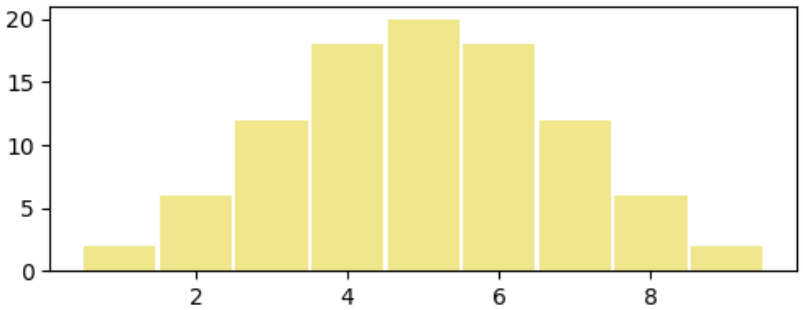
Redes Neuronales - Convoluciones (resaltando patrones)



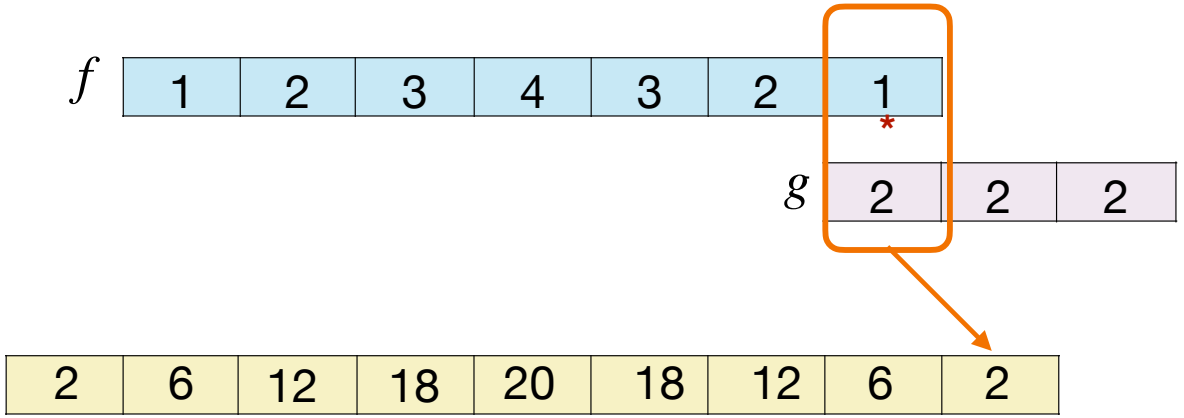


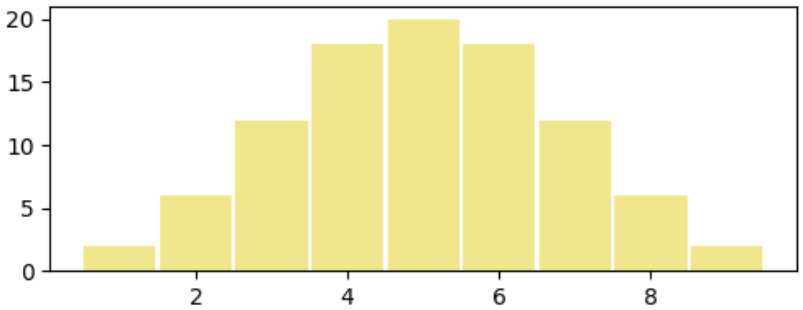
Redes Neuronales - Convoluciones (resaltando patrones)





Redes Neuronales - Convoluciones (resaltando patrones)



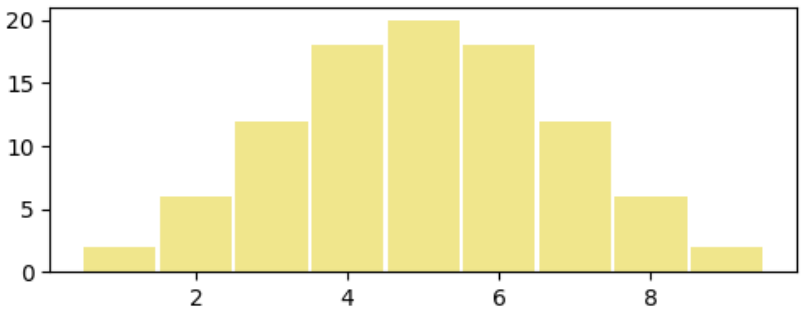


Redes Neuronales - Convoluciones (resaltando patrones)

f	1	2	3	4	3	2	1

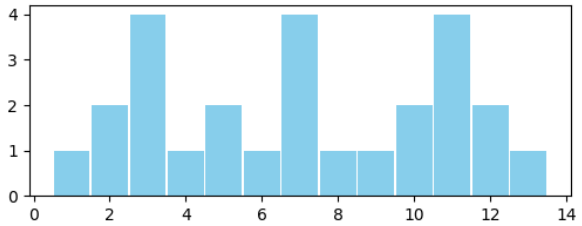
g	2	2	2

2	6	12	18	20	18	12	6	2



Redes Neuronales - Convoluciones (resaltando patrones)

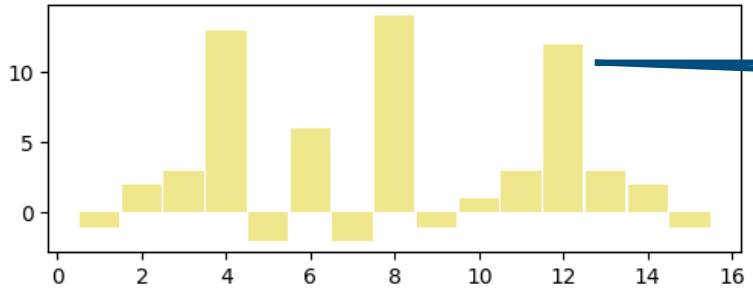
f	1	2	4	1	2	1	4	1	1	2	4	2	1



Se denomina **filtro** o **kernel**

g	-1	4	-1

-1	2	3	13	-2	6	-2	14	-1	1	3	12	3	2	-1



Resalta los bordes

Redes Neuronales - Convoluciones (resaltando patrones)

Una **convolución** es una operación que combina dos funciones para producir una tercera función que expresa cómo una de ellas modifica o “suaviza/enfatiza” a la otra.

Formalmente, para dos funciones continuas $f(t)$ y $g(t)$, la convolución se define como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau$$

En el caso discreto:

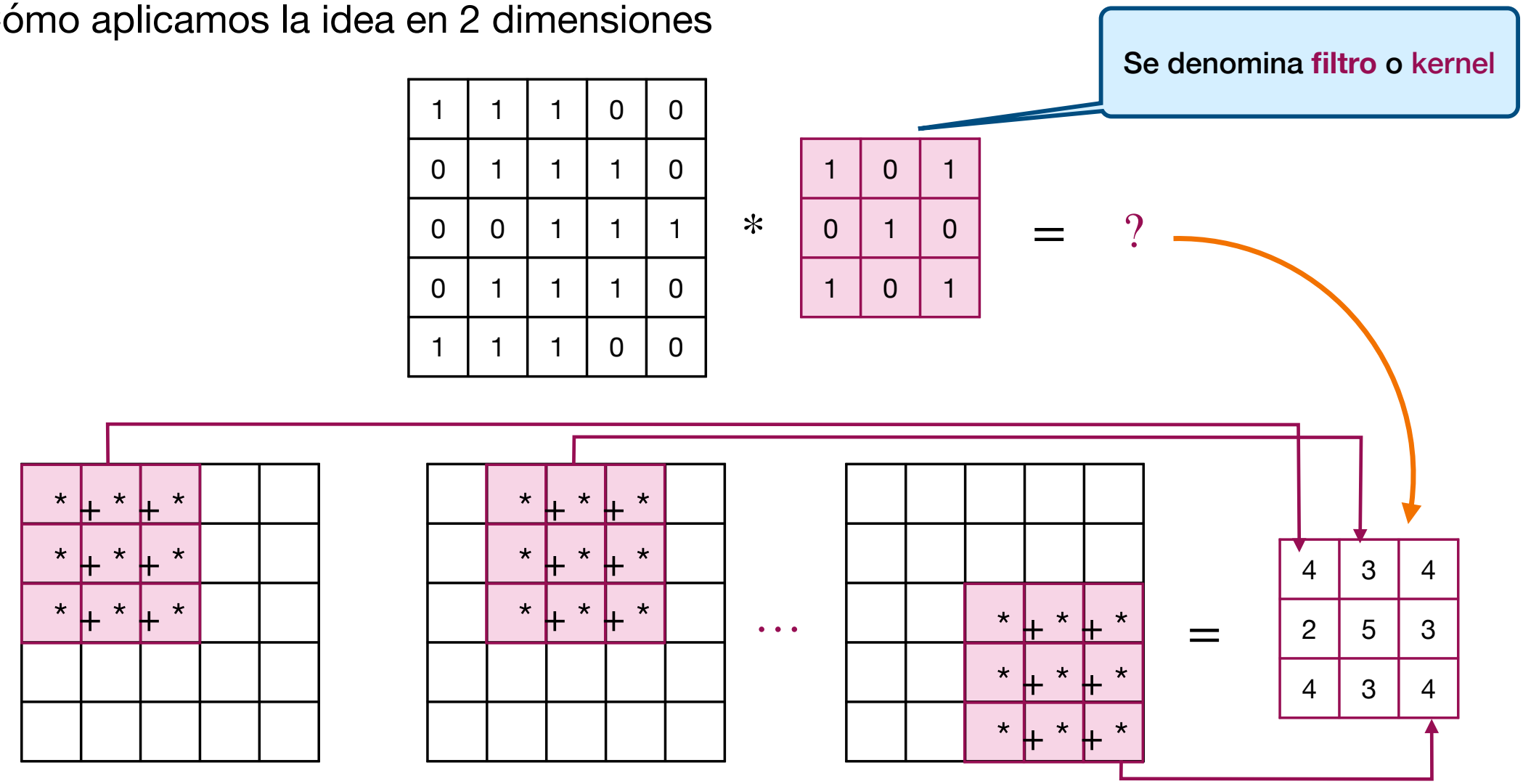
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n - m]$$

Esta operación puede interpretarse como un filtro g que se invierte y desplaza sobre otra función f, y en cada posición se mide su grado de coincidencia mediante una suma ponderada.

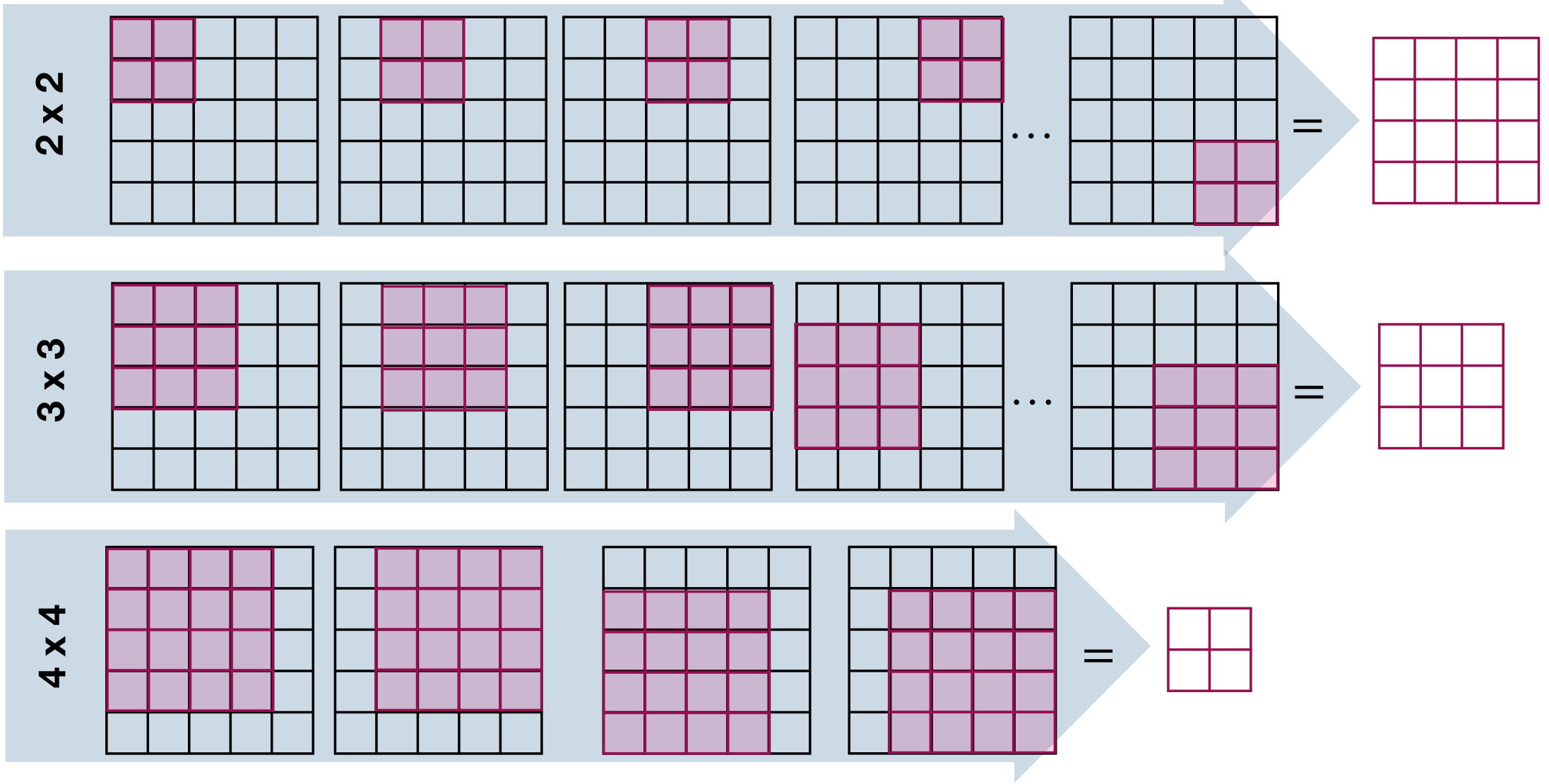
La convolución tiene propiedades matemáticas importantes: **conmutatividad**, **asociatividad** y **distributividad**

Redes Neuronales - Convoluciones (resaltando patrones)

Cómo aplicamos la idea en 2 dimensiones

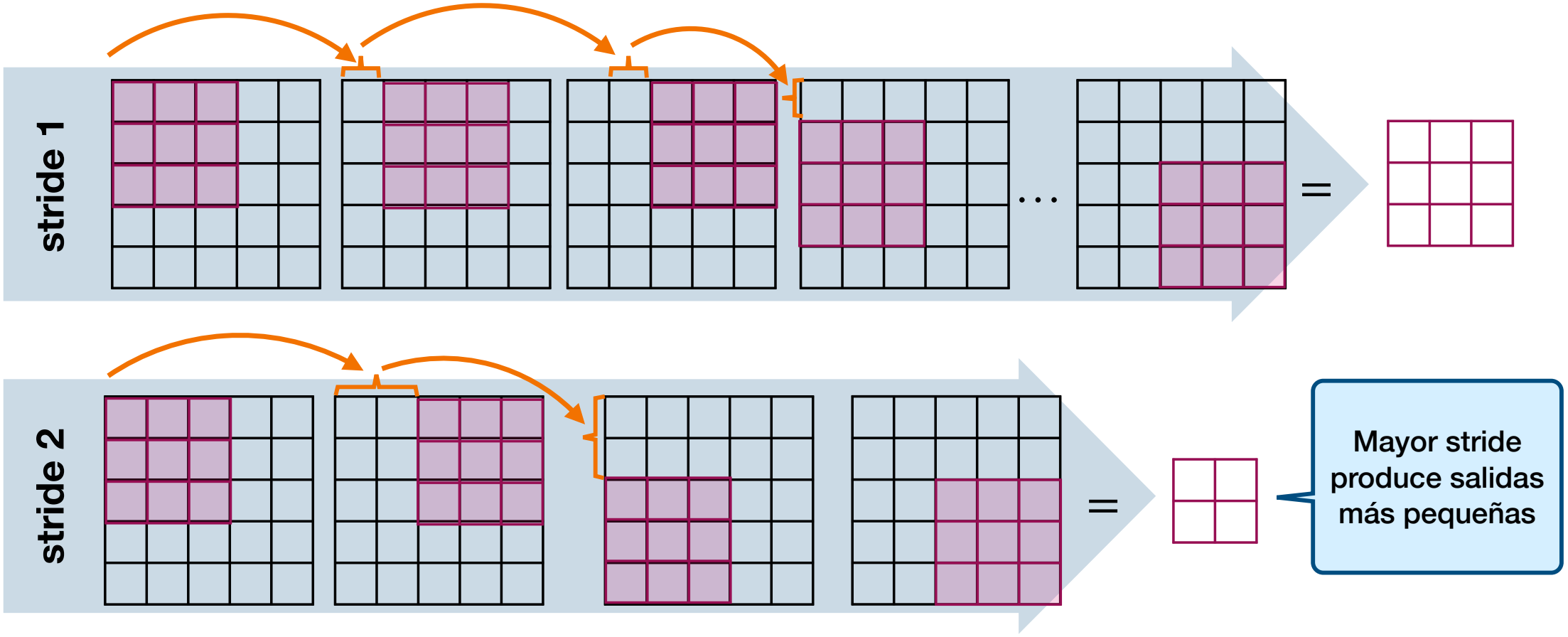


Redes Neuronales - Convoluciones (Tamaño del kernel)



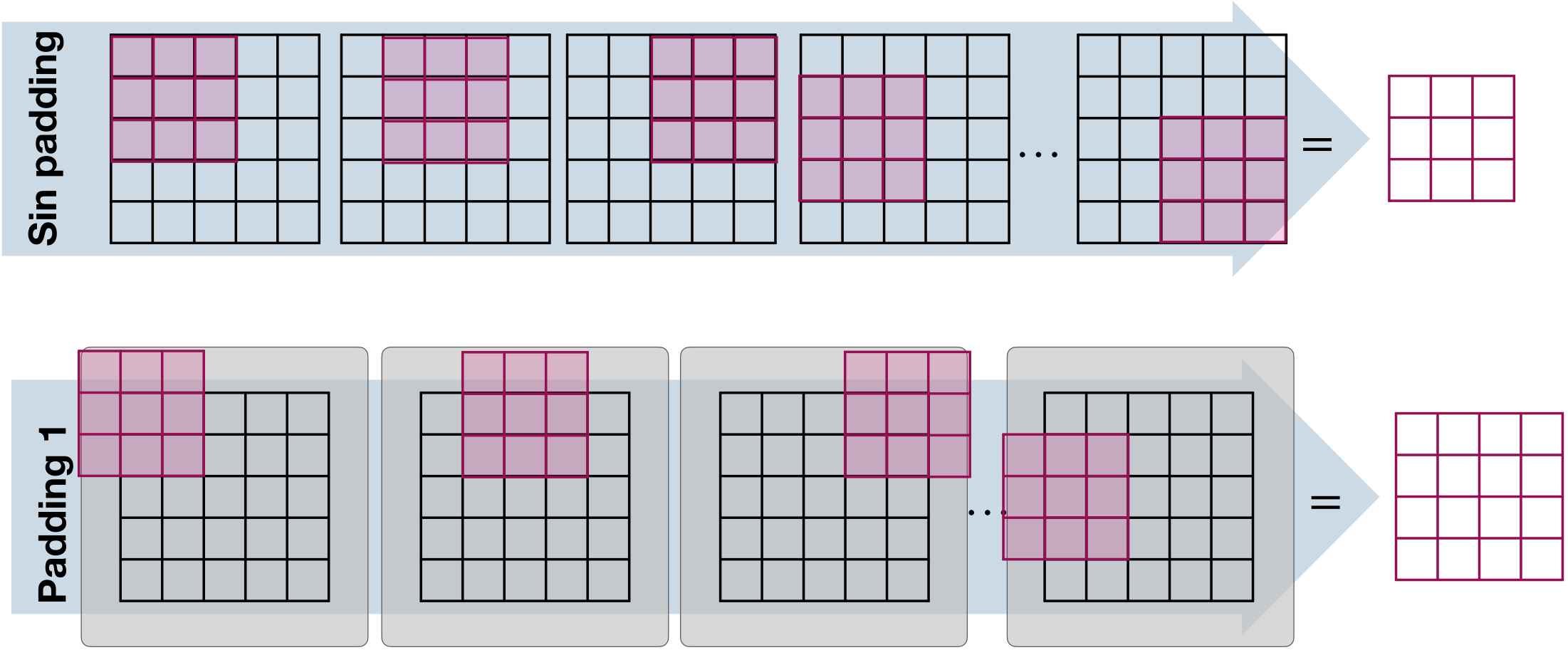
Redes Neuronales - Convoluciones (Stride)

El parámetro stride o salto, indica cuánto se desplaza el kernel en la entrada durante la convolución



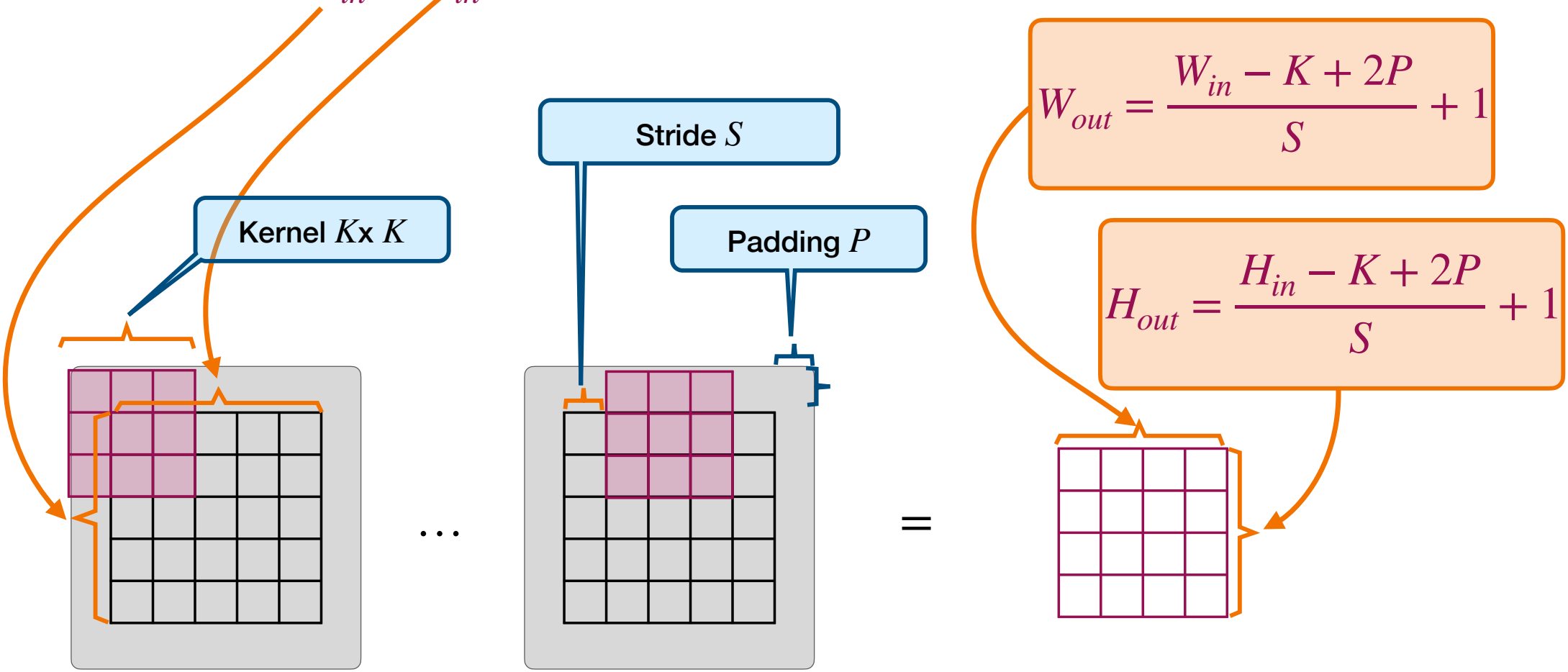
Redes Neuronales - Convoluciones (Padding)

Si necesitamos atención en los bordes podemos rellenar los bordes (padding)



Redes Neuronales - Convoluciones - Tamaño salida

Dada una entrada $H_{in} \times W_{in}$, un kernel de tamaño K, un padding P y un stride S:

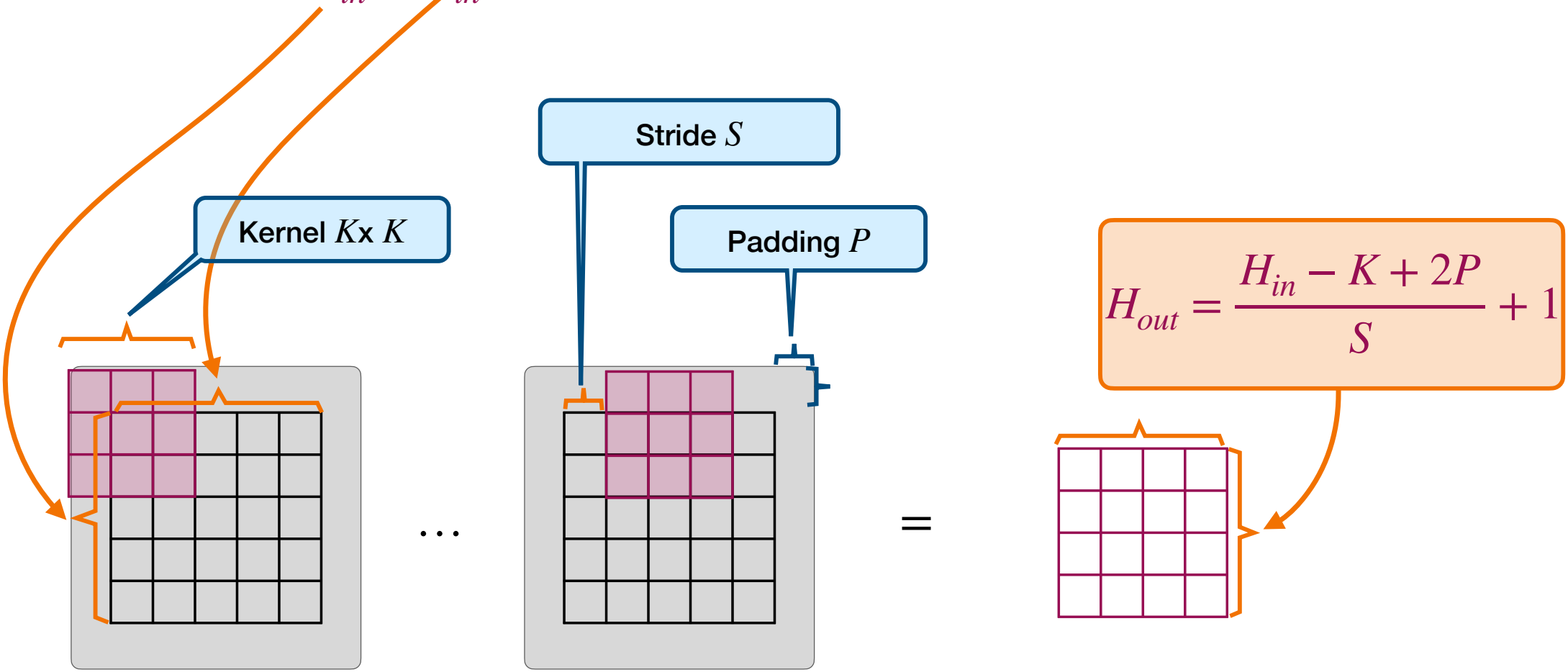


$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1$$

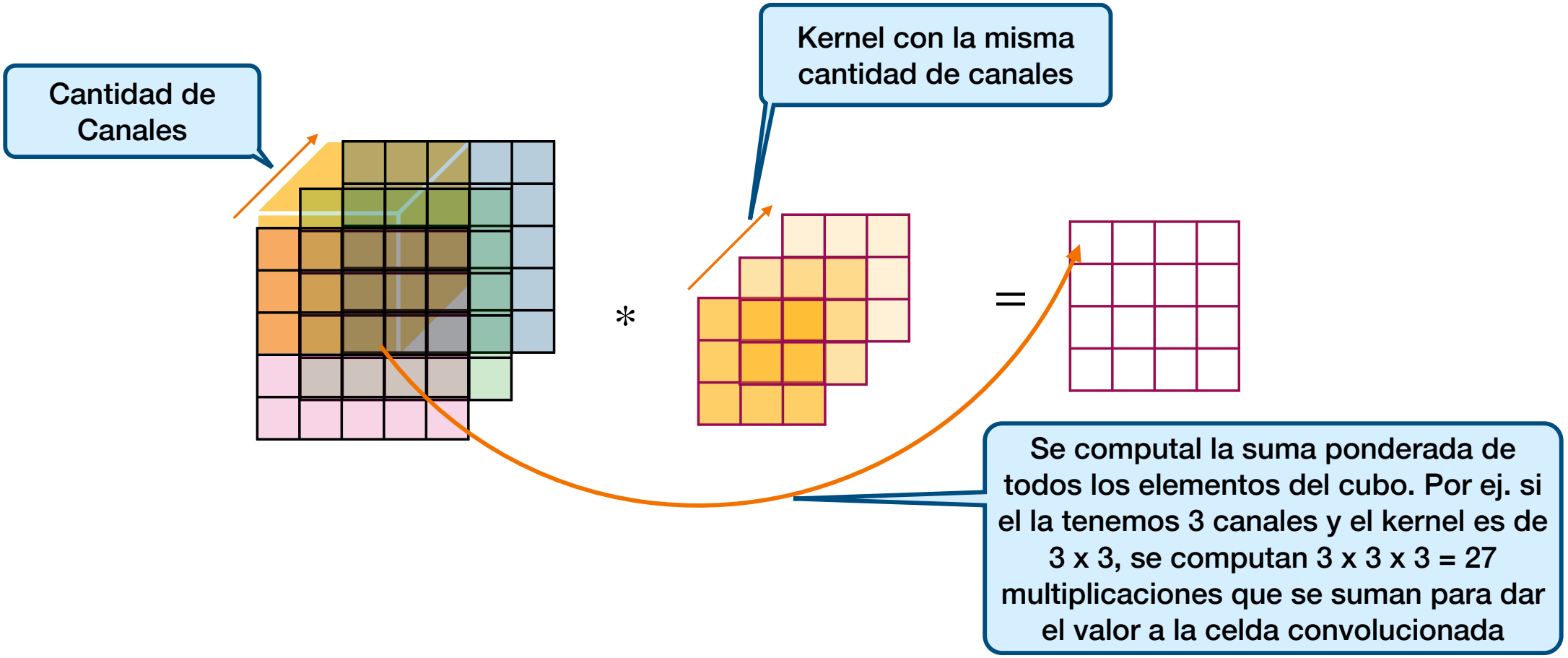
Redes Neuronales - Convoluciones - Multiples entradas y multiples filtros

Dada una entrada $H_{in} \times W_{in}$, un kernel de tamaño K, un padding P y un stride S:

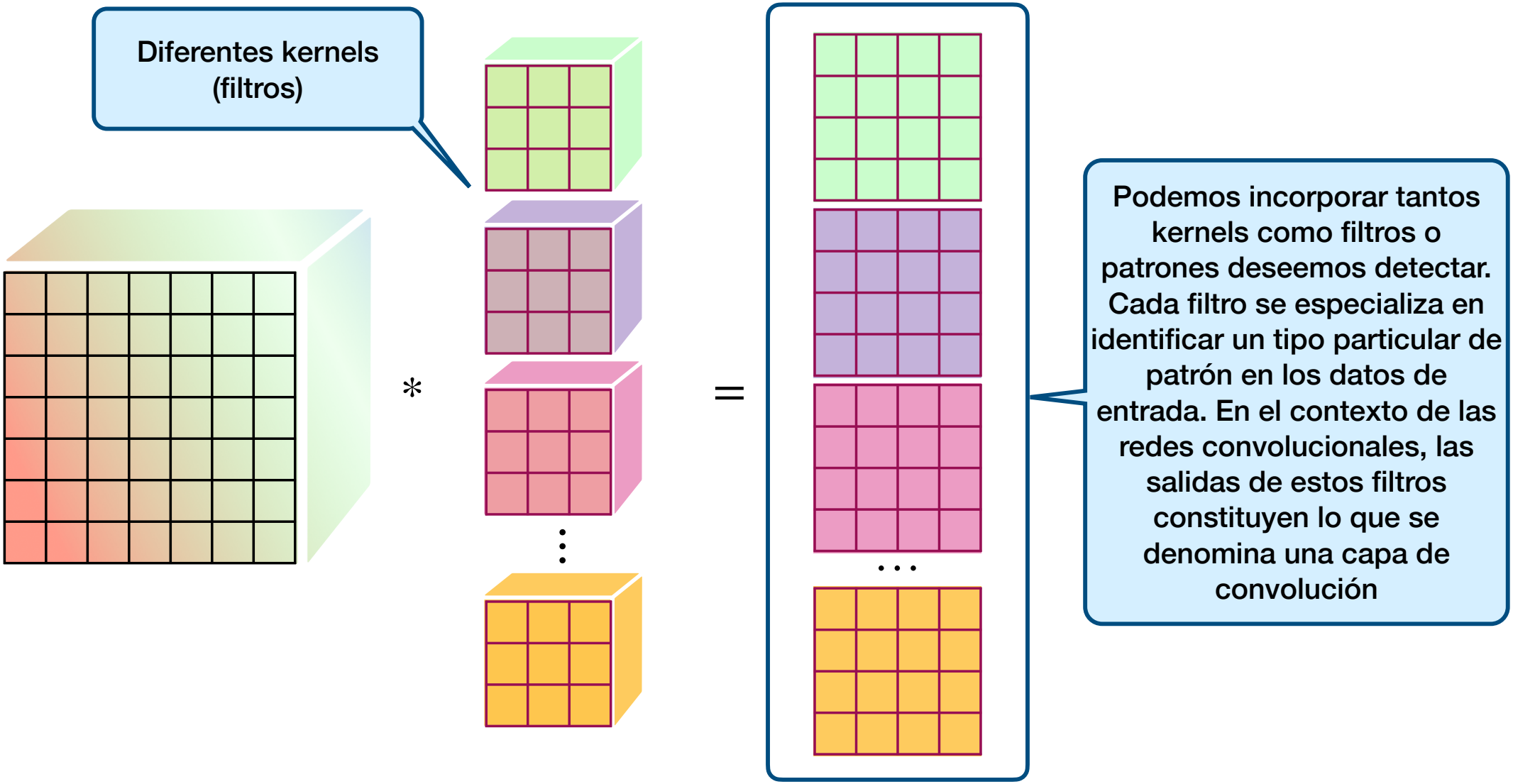


Redes Neuronales - Convoluciones - Multiples entradas (canales)

En ciertos dominios podemos tener más de un canal de entrada



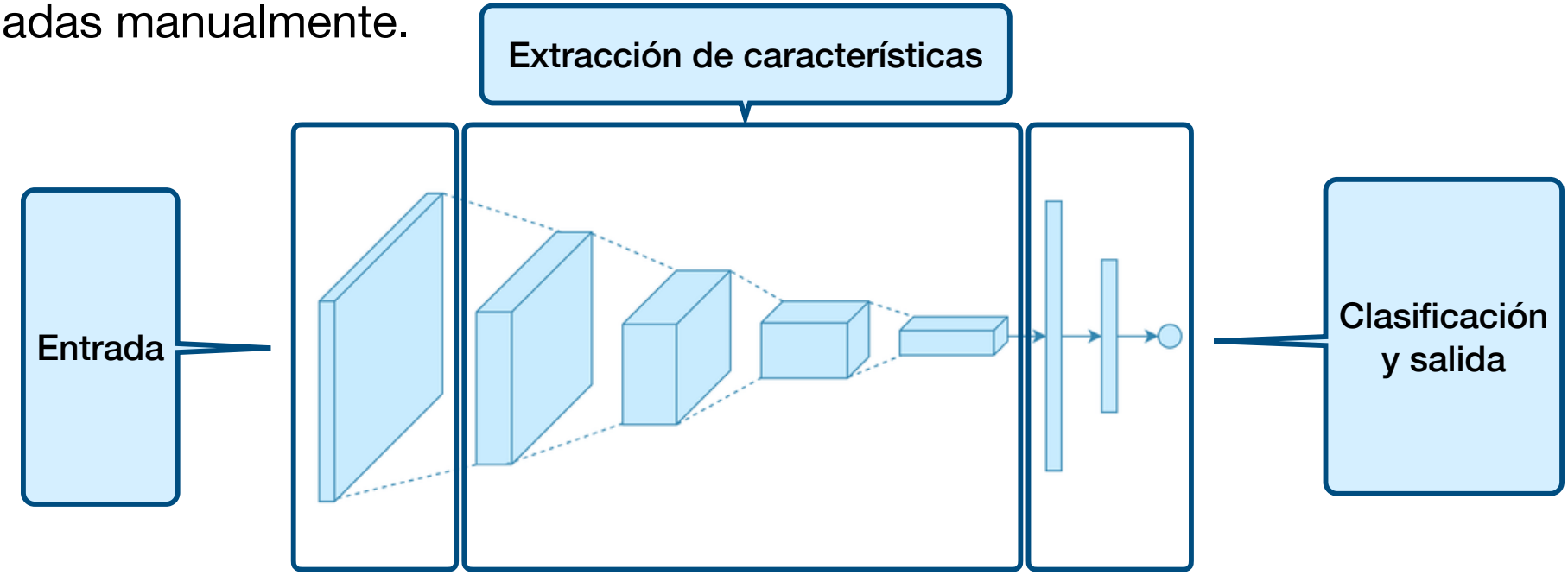
Redes Neuronales Convolucionales - Multiples filtros (kernels)



Redes Neuronales Convolucionales

Una **Red Neuronal Convolucional (CNN)** es un tipo de red neuronal diseñada para procesar datos con **estructura espacial, como imágenes**.

Su idea principal es aplicar filtros (convoluciones) que detectan automáticamente patrones locales (por ejemplo, bordes, formas o texturas) y combinarlos en capas sucesivas para reconocer estructuras más complejas. Una CNN aprende a "ver", identificando características relevantes de los datos sin necesidad de que sean programadas manualmente.

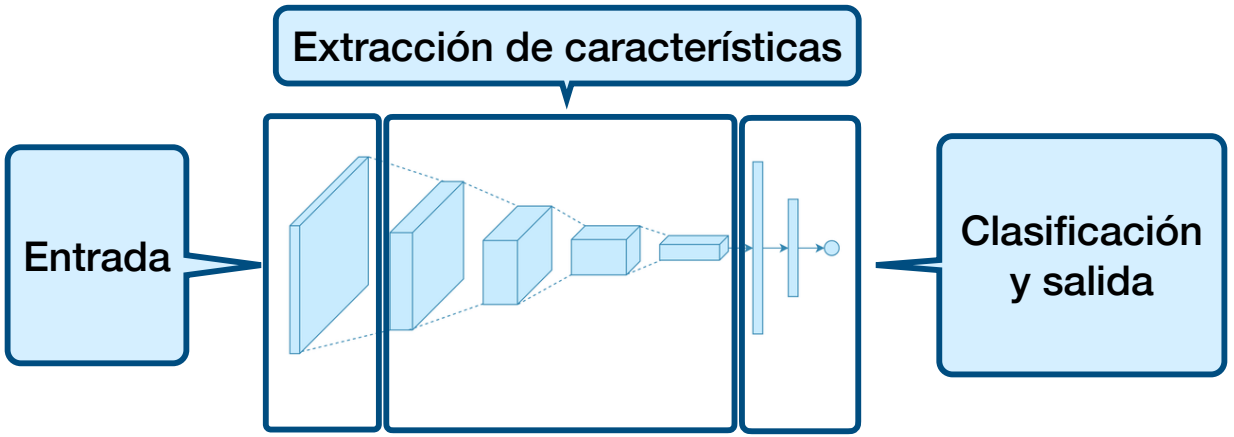


Redes Neuronales Convolucionales

El **objetivo de la etapa inicial es generar capas para extraer y resumir las características más importantes** de los datos de entrada (por ejemplo, una imagen), reduciendo la complejidad sin perder información relevante.

La extracción y resumen de características se realiza mediante la **generación de capas** mediante una o más (sucesivas) **convoluciones** con varios **filtros** (patrones a detectar) seguidas por capas de resumen (**pooling**).

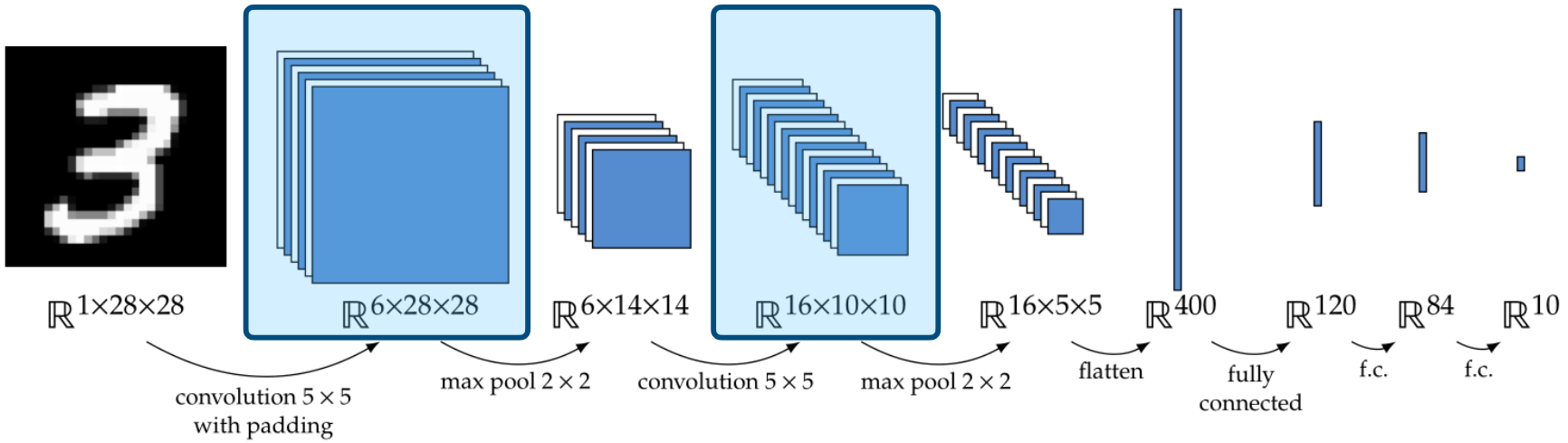
Estas etapas son menos densas y permiten minimizar la información, considerando la más relevante (inducida por los filtros) para alimentar la **última etapa de clasificación** por medio de una o más capas **fully conected** (ANN).



Redes Neuronales Convolucionales - Convoluciones

Las capas de convolución detectan patrones locales como bordes, texturas, esquinas o colores específicos aplicacando filtros (kernels) que se deslizan sobre la imagen y **generan mapas de características (feature maps)**.

Cada capa convolucional aprende automáticamente un conjunto de características, pasando de patrones simples (en las primeras capas) a patrones complejos (en capas más profundas).



Redes Neuronales Convolucionales - Filtros (kernels)

Ejemplos y efectos de algunos filtros o kernels de las primeras capas:

Identidad: No modifica la imagen. Se usa como referencia o para pruebas.

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

Detección de bordes Sobel (horizontal): Resalta bordes horizontales.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel (vertical): resalta bordes verticales.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Laplaciano: detecta bordes sin dirección (bordes generales).

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Suavizado / Desenfoque (Blur o Mean Filter): Promedia los píxeles vecinos (reduce ruido y bordes suaves).

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filtro de nitidez (Sharpen): Aumenta el contraste local, define mejor los bordes.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Emboss (Relieve): Genera un relieve tridimensional (sombreados)

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Filtro de realce de bordes diagonales

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

Redes Neuronales Convolucionales - Pooling

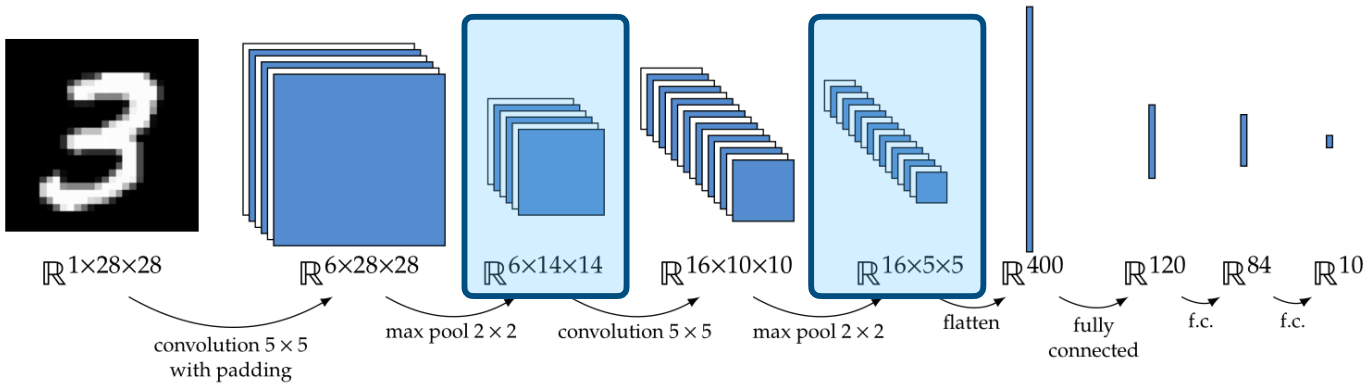
Las capas de **pooling** tienen como objetivo reducir el tamaño de los mapas de características conservando la información más importante.

Para ello **resume regiones** (por ejemplo, de 2×2) tomando el **valor máximo** (**max-pooling**) o el **promedio** (**average pooling**).

Este proceso disminuye la cantidad de parámetros, hace el modelo más eficiente y **genera invariancia a pequeñas traslaciones o deformaciones**.

2 x 2 pooling

1	0	0	0
4	2	3	1
5	3	7	1
1	2	2	0



Max-pooling mantiene el valor más grande de la región. En caso de ***average***, computa el promedio

4	3
5	7

Redes Neuronales Convolucionales - Data Aumentation

En general cuando la cantidad y calidad (respecto a diversidad) de los datos no es suficiente, se puede recurrir a la técnica de **Data Augmentation (aumento de datos)** para mejorar la **generalización** del modelo.

Data Augmentation genera nuevas imágenes artificiales aplicando transformaciones aleatorias que no cambian la clase del objeto, por ejemplo:

Tipo	Descripción	Ejemplo
Rotación	Gira la imagen unos grados	+15°, -20°
Traslación	Desplaza horizontal/verticalmente	10% del ancho
Escalado / Zoom	Acerca o aleja la imagen	zoom 0.8–1.2
Reflexión (flip)	Invierte la imagen	horizontal/vertical
Brillo / Contraste	Modifica intensidad	±30%
Corte aleatorio (crop)	Recorta una región y reescala	mantener objeto
Ruido	Agrega ruido gaussiano	σ=0.05
Deformación elástica	Distorsiona ligeramente	usado en MNIST

Redes Neuronales Convolucionales (un poco de historia)

1959 – Hubel y Wiesel: descubrieron en experimentos con gatos que la corteza visual contiene neuronas sensibles a regiones locales del campo visual, y que distintas neuronas responden a bordes, orientaciones o patrones específicos. Este descubrimiento inspiró el concepto de **receptive fields** y la idea de que la percepción visual ocurre mediante jerarquías de detección progresiva.
1980 – Kunihiko Fukushima: propuso el **Neocognitron**, considerado el precursor directo de las CNN modernas. Su arquitectura consistía en capas alternadas que realizaban operaciones equivalentes a convolución y pooling. Introdujo el concepto de invariancia a traslaciones y reconocimiento jerárquico.
1989 – LeNet-5 (Yann LeCun): integró las ideas de convolución, pooling y aprendizaje supervisado mediante retropropagación (backpropagation). El modelo LeNet-5 fue entrenado para reconocer dígitos escritos a mano en el dataset MNIST y se implementó en sistemas bancarios para leer cheques.
Fue la primera CNN entrenada con gradient descent y pesos aprendidos automáticamente.
2000s – Estancamiento: El progreso fue limitado debido a la falta de volumen de datos etiquetados y capacidad computacional suficiente.
2012 – AlexNet (Krizhevsky, Sutskever, Hinton): que ganó la competencia ImageNet con una mejora de más de 10 puntos porcentuales sobre el segundo lugar. Usó GPU, ReLU, Dropout y data augmentation. Marcó el comienzo de la Deep Learning Revolution moderna.
2014–2020 – Modelos Profundos: La arquitectura CNN se profundizó y diversificó. VGGNet (2014), GoogLeNet / Inception (2014), ResNet (2015). DenseNet, EfficientNet, MobileNet, etc.
2020... – Híbridos y Transformers: Modelos como Vision Transformer (ViT) integran atención (self-attention) con conceptos convolucionales. Sin embargo, las CNN siguen siendo fundamentales en visión por computadora, robótica, biometría y procesamiento de señales.

Redes Neuronales Convolucionales - Ventajas

Sparse Interactions (Interacciones dispersas o locales): En una red neuronal densa (fully connected), cada neurona de una capa está conectada con todas las neuronas de la capa anterior $O(n^2)$ conexiones, una gran cantidad de parámetros generando un alto costo computacional.

En una convolucional, cada neurona de la salida se conecta solo con una pequeña región local de la entrada, el receptive field (campo receptivo). Por ejemplo, si tenemos una imagen de 100 x 100 píxeles y aplicamos un filtro 3 x 3: Cada neurona de salida se conecta solo con 9 entradas en lugar de 10.000. Esto genera interacciones dispersas, lo que reduce drásticamente los parámetros y el costo computacional. Esta “dispersión” permite que la red se **concentre en patrones locales** (bordes, texturas, esquinas) antes de combinarlos en niveles más altos para detectar formas globales.

Redes Neuronales Convolucionales - Ventajas

Parameter Sharing (Compartición de parámetros) : En una ANN tradicional, cada conexión tiene su propio peso w_{ij}. En cambio en una convolucional, el mismo conjunto de pesos (el filtro o kernel) se aplica a lo largo de toda la entrada. Matemáticamente, si el filtro K tiene pesos w_{ij}, la convolución en la posición (i, j) se calcula como:

$$y_{i,j} = \sum_a \sum_b w_{a,b} x_{i+a,j+b}$$

Los pesos $w_{a,b}$ se **comparten** en todas las posiciones (i, j), lo que significa que el número de parámetros no depende del tamaño de la entrada. La red aprende un **filtro universal** que detecta la misma característica (por ejemplo, un borde) en cualquier lugar de dato de entrada. Esto permite que la red sea **invariante a traslaciones**: un patrón desplazado produce la misma respuesta.

Redes Neuronales Convolucionales - Ventajas

Equivariance to Translation and Variable-Sized Inputs (Equivarianza e invariancia)
Una operación es **equivariante** si un cambio en la entrada produce un cambio correspondiente en la salida. En una convolución: $f(T(x)) = T(f(x))$ donde T representa una **traslación** (desplazamiento). **Esto significa que si desplazamos la imagen de entrada, el mapa de características se desplaza de la misma forma.**

Como ejemplo, si un objeto se mueve en la imagen, su activación se mueve en el **feature map**, pero **la forma del patrón detectado se conserva**. Esto es diferente de la **invarianza**, donde la salida sería la misma independientemente del desplazamiento (la invarianza total se logra con el *pooling*).

Además, se puede trabajar con **entradas de tamaño variable**, porque el filtro convolucional opera localmente y no depende de la dimensión total de la entrada. Las capas de *pooling* y *global average pooling* permiten reducir las dimensiones antes de la capa de salida, independientemente del tamaño de entrada original.

Implementando Redes Neuronales con Keras

Keras es una biblioteca (API) de alto nivel en Python diseñada para crear, entrenar y evaluar modelos de redes neuronales artificiales de manera accesible.

Fue desarrollada originalmente por François Chollet en 2015 y hoy forma parte oficial del ecosistema de **TensorFlow** (desde TensorFlow 2.0, Keras es su interfaz principal).

Keras está orientada a la experimentación rápida con **Deep Learning**

Se destaca por su sintaxis simple, legible y consistente

Provee un diseño modular, cada modelo se componen de bloques reutilizables: capas (layers), funciones de activación, optimizadores y funciones de costo.

Implementando Redes Neuronales con Keras - Modelos

El diseño más abstracto es el concepto de **Modelo**:

Sequential: Ideal para redes apiladas donde las capas se conectan de forma lineal.

Functional: Más flexible; permite arquitecturas con múltiples entradas o salidas, bifurcaciones o conexiones residuales.

Subclassing: Ofrece control total al definir clases personalizadas que heredan de `tf.keras.Model`.

Implementando Redes Neuronales con Keras - Componentes de un Modelo

Capas (Layers): son los bloques fundamentales de una red (Dense, Conv2D, LSTM, Dropout, etc.)

`Dense(32, activation='relu')`

Activaciones: determinan la salida no lineal de una capa.

'relu', 'sigmoid', 'softmax'

Funciones de pérdida (Loss): Cuantifican el error del modelo.

'mse', 'binary_crossentropy', 'categorical_crossentropy'

Optimizadores: ajustan los pesos minimizando la pérdida.

'sgd', 'adam', 'rmsprop'

Métricas: Evalúan la calidad del modelo durante el entrenamiento.

'accuracy', 'mae'

Implementando Redes Neuronales con Keras - Flujo de trabajo

Definición del modelo: Se especifica la arquitectura: capas, activaciones y conexiones.

```
Sequential o Model.
```

Compilación: Se establecen el optimizador, la función de pérdida y las métricas.

```
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Entrenamiento: Se entrena el modelo con datos de entrada y etiquetas.

```
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Evaluación: Se mide el rendimiento del modelo sobre datos de prueba.

```
model.evaluate(X_test, y_test)
```

Predicción: Se obtienen salidas para nuevas entradas.

```
model.predict(X_new)
```

Implementando Redes Neuronales con Keras - Complementos

Callbacks: permiten monitorear y modificar el entrenamiento (por ejemplo, EarlyStopping, ModelCheckpoint, ReduceLROnPlateau).

Data Generators: para procesar grandes conjuntos de datos o aplicar data augmentation.

TensorBoard Integration: visualización interactiva del proceso de entrenamiento.

Guardado de modelos:

```
model.save("modelo.h5")
tf.keras.models.load_model("modelo.h5")
```

Implementando Redes Neuronales con Keras - Ejemplos

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical
# Cargar datos
X, y = load_iris(return_X_y=True)
y = to_categorical(y)
X = StandardScaler().fit_transform(X)
# División entre entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Definir modelo
model = Sequential([
    Dense(10, activation='relu', input_shape=(4,)),
    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])
# Compilar
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Entrenar
model.fit(X_train, y_train, epochs=50, batch_size=8, verbose=0, validation_split=0.1)
# Evaluar
loss, acc = model.evaluate(X_test, y_test)
print(f"Accuracy: {acc:.2f}")
```

Para una capa convolucional:

```python
Conv2D(16, (3,3), activation='relu',...
```

Para una capa de pooling:

```python
MaxPooling2D((2,2))
```