# Modelos de Deep Learning

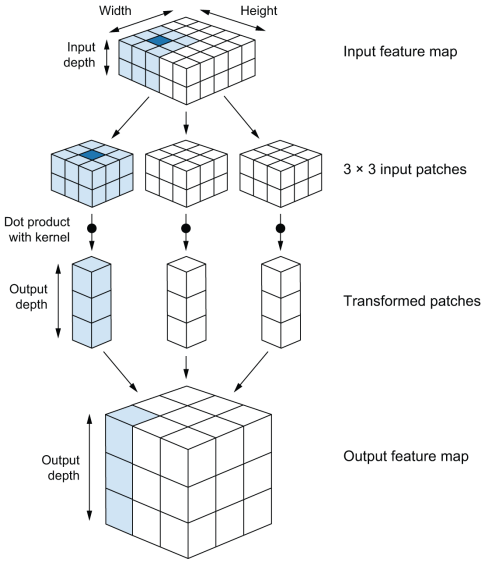
## Redes convolucionales

Universidad ORT Uruguay

6 de Octubre, 2025

# Capa convolucional 2D: idea

- Cada ventana ($s \times s \times c$) se transforma en un vector mediante un **kernel** (filtro).
- El mismo kernel se aplica a todos los píxeles: esto genera invariancia traslacional.
- Kernels clásicos en tratamiento de imágenes.



# Capa convolucional 2D: definición

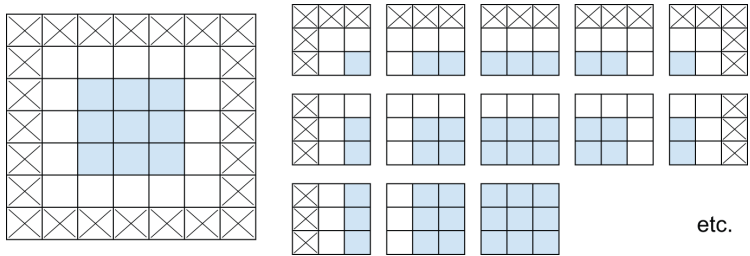
- Entrada $\mathbf{X} \sim (h, w, c)$, kernel size $s = 2k + 1$, número de canales de salida $c'$.
- Definición (elemento a elemento):

$$H_{ij} = \text{vect}\left(\mathbf{P}_k(i, j)\right)^\top \mathbf{W} + \mathbf{b}^\top,$$

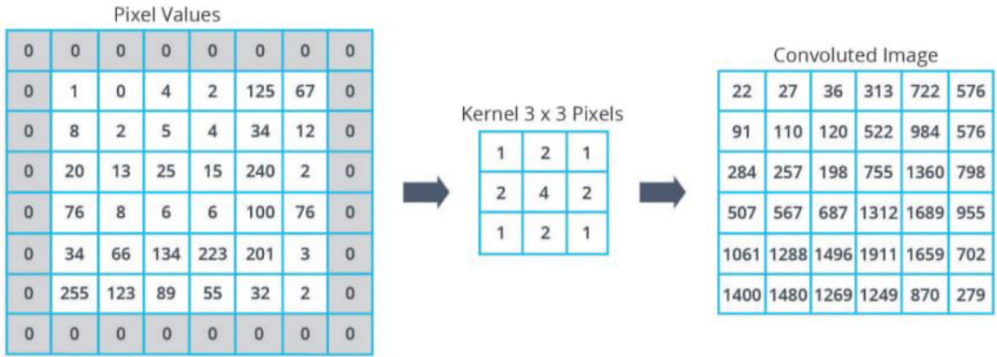
- Parámetros entrenables son $\mathbf{W} \sim (s^2 c, c')$, $\quad \mathbf{b} \sim (c')$.
- Hiperparámetros: $k$ (o $s$), $c'$, y si se aplica zero-padding.
- Shape de salida: $\mathbf{H} \sim \begin{cases} (h, w, c') & \text{con padding de } k \text{ píxeles} \\ (h - 2k,\ w - 2k,\ c') & \text{sin padding.} \end{cases}$

# Padding en convoluciones

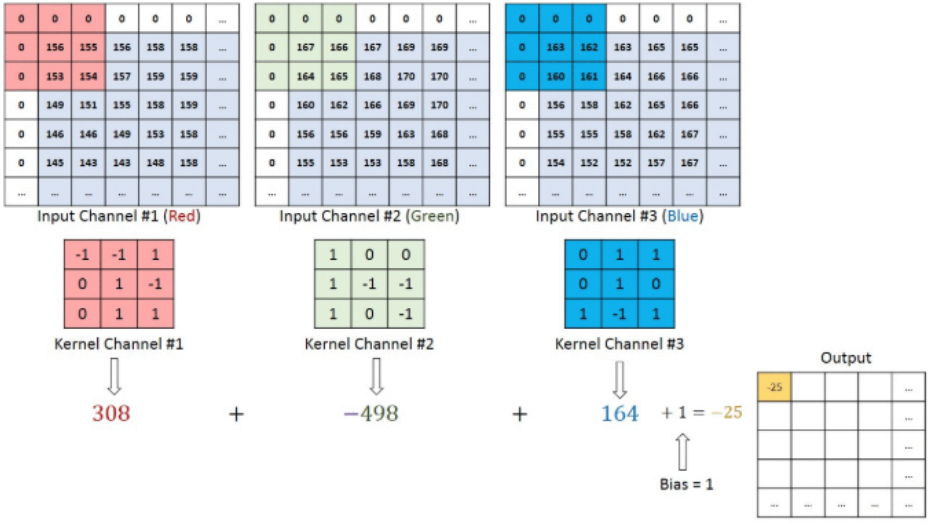
- **Propósito**: evitar que la imagen reduzca su dimensión tras aplicar convoluciones.
- Se logra agregando filas y columnas alrededor de la entrada.
- Permite conservar el **mismo tamaño** de salida.



# Convolución con un kernel: ejemplo

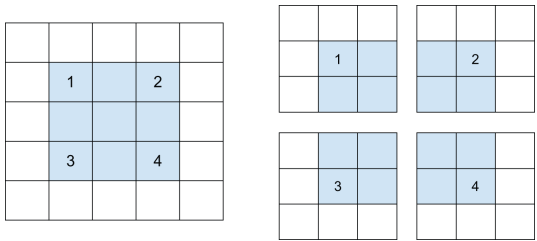


# Multiples canales



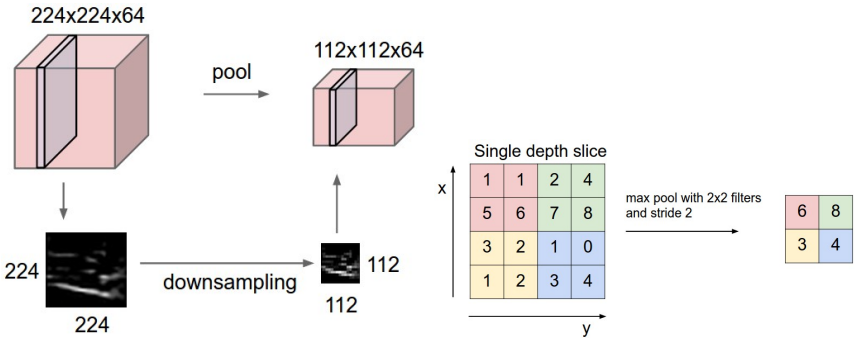
# Stride en convoluciones

- **Propósito**: controlar el paso con que se desliza la ventana de convolución.
- Con **stride = 1**: se recorren todas las posiciones (salida más grande).
- Con **stride > 1**: salida queda **downsampled** (más pequeño en ancho y alto).



# Max/Avg Pooling

- Operación fija (**máximo/promedio**) en lugar de un kernel aprendido.
- Común usar con size $2 \times 2$ y **stride = 2** (efecto downsampling).
- Resumir la información más relevante.
- Permitir que capas posteriores capten patrones más grandes.



# Resumen: Capa Convolucional

- Shape Entrada: ($W_1$, $H_1$ $C_1$)
- Hiperparámetros principales:
  - Número de filtros $K$
  - Extensión espacial (kernel size) $F$
  - Stride $S$
  - Zero padding $P$
- Shape Salida: ($W_2$, $H_2$, $C_2$), con: $W_2 = \frac{W_1 - F + 2P}{S} + 1$, $H_2 = \frac{H_1 - F + 2P}{S} + 1$, $C_2 = K$
- **Parameter sharing**: $(F \cdot F \cdot C_1) \cdot K$ pesos + $K$ biases
- En la salida, cada canal corresponde a un filtro aplicado al input con stride $S$ y desplazado por un bias.

CS231n: Convolutional Networks

# Bloques convolucionales

- Combinar varias capas convolucionales con pooling:

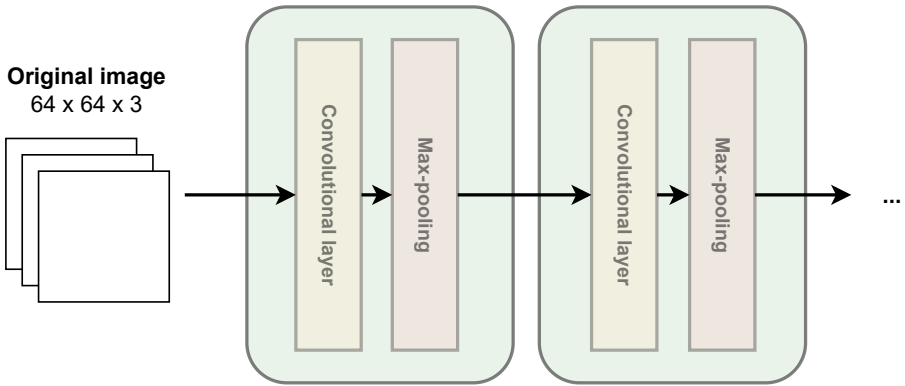
$$\text{ConvBlock}(\mathbf{X}) = (\text{MaxPool} \circ A \circ \text{Conv} \circ \cdots \circ A \circ \text{Conv})(\mathbf{X})$$

- Y una red más compleja con múltiples bloques:

$$\mathbf{H} = (\text{ConvBlock} \circ \text{ConvBlock} \circ \cdots \circ \text{ConvBlock})(\mathbf{X})$$

- El diseño VGG popularizó la idea de mantener constante el tamaño del filtro en cada capa, mantener constante el número de canales dentro de cada bloque y duplicarlos entre bloques consecutivos.

# De capas a bloques



# Arquitectura completa de la CNN

El modelo completo puede descomponerse en tres componentes principales:

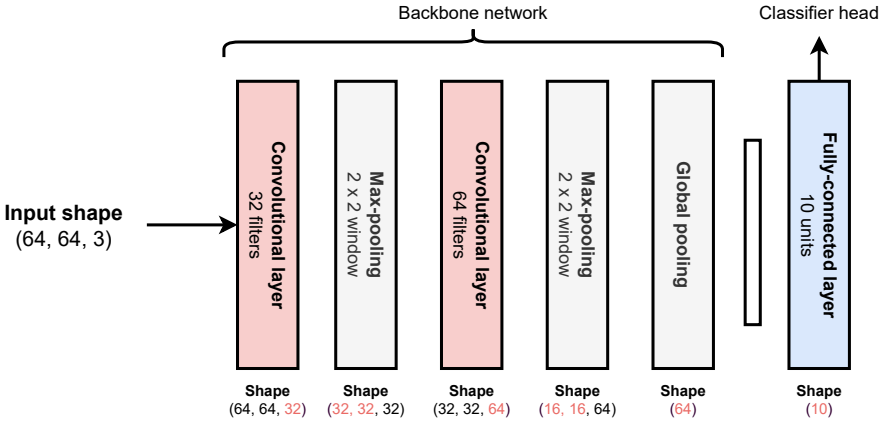
$$\mathbf{H} = (\text{ConvBlock} \circ \cdots \circ \text{ConvBlock})(\mathbf{X}) \tag{1}$$

$$\mathbf{h} = \frac{1}{h'w'} \sum_{i,j} H_{ij} \quad \text{(global average pooling)} \tag{2}$$

$$\mathbf{y} = \text{MLP}(\mathbf{h}) \tag{3}$$

MLP($\mathbf{h}$) es una secuencia genérica de capas densas (también puede usarse una operación de *flattening* en lugar del *pooling*).

# Ejemplo de arquitectura



# Ejemplo

```
import keras
from keras import layers

inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

# Ejemplo

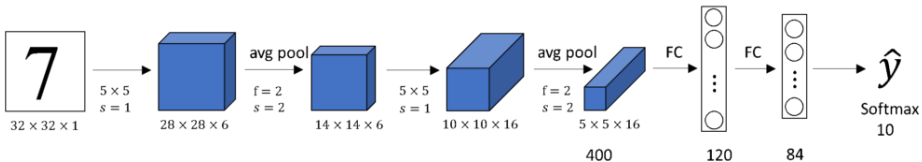
```
>>> model.summary()
Model: "functional"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer (InputLayer) | (None, 28, 28, 1) | 0 |
| conv2d (Conv2D) | (None, 26, 26, 64) | 640 |
| max_pooling2d (MaxPooling2D) | (None, 13, 13, 64) | 0 |
| conv2d_1 (Conv2D) | (None, 11, 11, 128) | 73,856 |
| max_pooling2d_1 (MaxPooling2D) | (None, 5, 5, 128) | 0 |
| conv2d_2 (Conv2D) | (None, 3, 3, 256) | 295,168 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 256) | 0 |
| dense (Dense) | (None, 10) | 2,570 |

```
Total params: 372,234 (1.42 MB)
Trainable params: 372,234 (1.42 MB)
Non-trainable params: 0 (0.00 B)
```

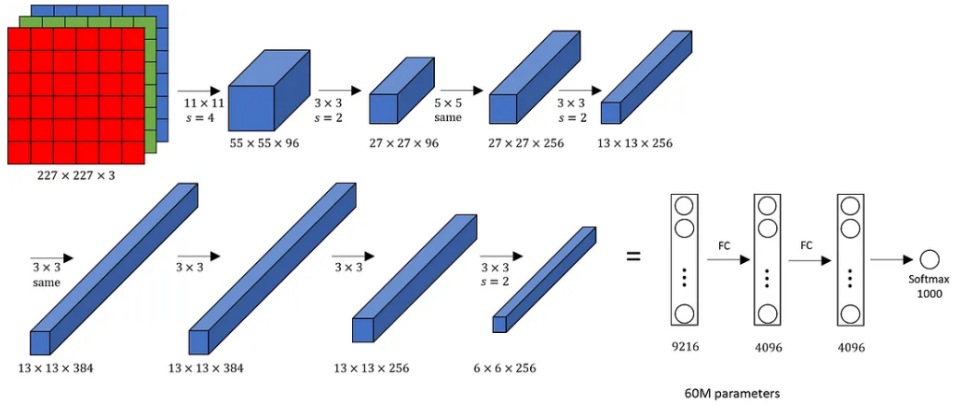
# ConvNets: Arquitecturas - LeNet

- **LeNet-5** (Y. LeCun, 1998): primera aplicación exitosa de redes convolucionales; clasificó dígitos de *MNIST* superando el SOTA de la época.
- Patrón típico: resolución espacial $(H, W) \downarrow$ tras conv/pool, **canales** $C \uparrow$; al final capas *Fully Connected* + *Softmax*.




*Nota:* LeNet usa bloques `Conv` → `Pool` (p.ej., 5×5, stride 1; avg/max pooling, stride 2) y termina en FC→Softmax. (∼60k parámetros)

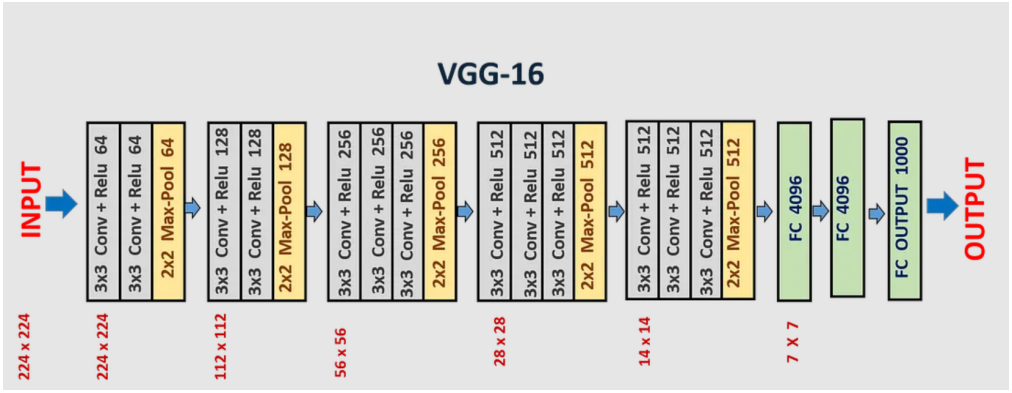
# ConvNets: Arquitecturas - AlexNet



# ConvNets: Arquitecturas - AlexNet

| Layer | | Feature Map | Size | Kernel Size | Stride | Activation |
|---|---|---|---|---|---|---|
| Input | Image | 1 | 227x227x3 | - | - | - |
| 1 | Convolution | 96 | 55 x 55 x 96 | 11x11 | 4 | relu |
| | Max Pooling | 96 | 27 x 27 x 96 | 3x3 | 2 | relu |
| 2 | Convolution | 256 | 27 x 27 x 256 | 5x5 | 1 | relu |
| | Max Pooling | 256 | 13 x 13 x 256 | 3x3 | 2 | relu |
| 3 | Convolution | 384 | 13 x 13 x 384 | 3x3 | 1 | relu |
| 4 | Convolution | 384 | 13 x 13 x 384 | 3x3 | 1 | relu |
| 5 | Convolution | 256 | 13 x 13 x 256 | 3x3 | 1 | relu |
| | Max Pooling | 256 | 6 x 6 x 256 | 3x3 | 2 | relu |
| 6 | FC | - | 9216 | - | - | relu |
| 7 | FC | - | 4096 | - | - | relu |
| 8 | FC | - | 4096 | - | - | relu |
| Output | FC | - | 1000 | - | - | Softmax |

# ConvNets: Arquitecturas - VGG



# Bibliografía


- Deep Learning. Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press, 2016.
- Prince, Simon JD. Understanding deep learning. MIT press, 2023.
- Bishop, Christopher M., and Hugh Bishop. Deep learning: Foundations and concepts. Springer Nature, 2023.
- Scardapane, Simone. "Alice's Adventures in a Differentiable Wonderland–Volume I, A Tour of the Land." arXiv preprint arXiv:2404.17625 (2024).