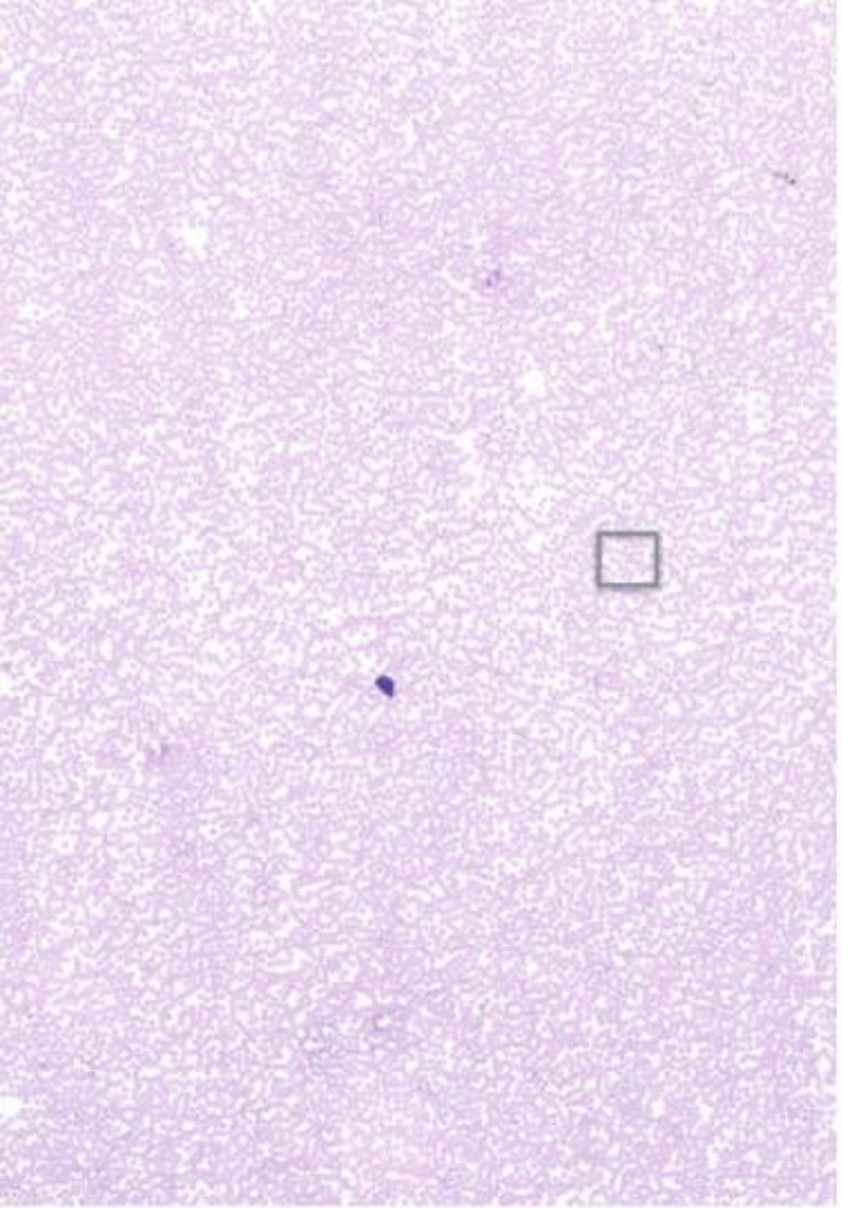
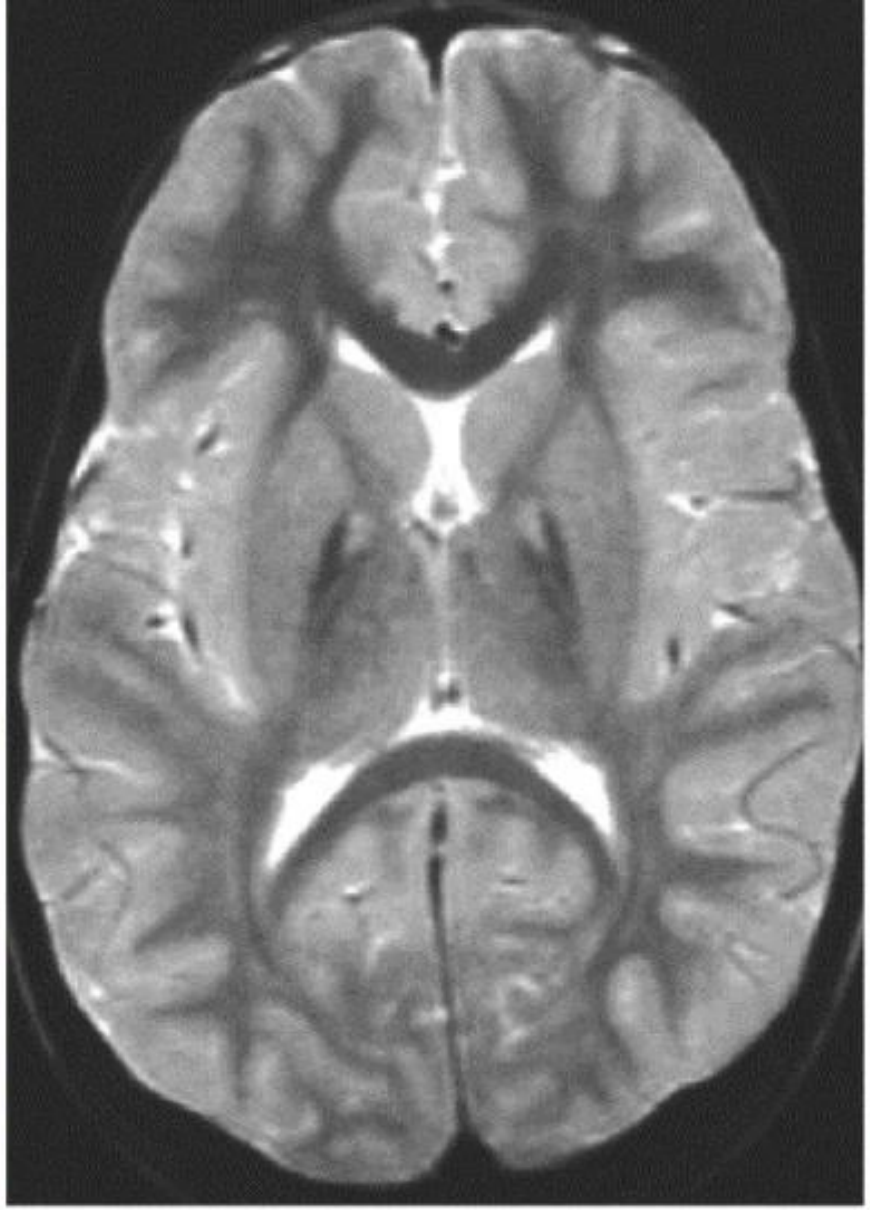
UNIVA CIING 2025 | Inteligencia Artificial con PyTorch – Entendiendo la API de PyTorch | Bloque 2

Manuel Santana Castolo


2015
SELF-DRIVING CARS IN 2 YEARS
CausalPython.io
imgflip.com
2016
RADIOLOGISTS OBSOLOLETE IN 5 YEARS
2024
RADIOLOGISTS IN REGULAR CARS DRIVING TO WORK





MALARIA

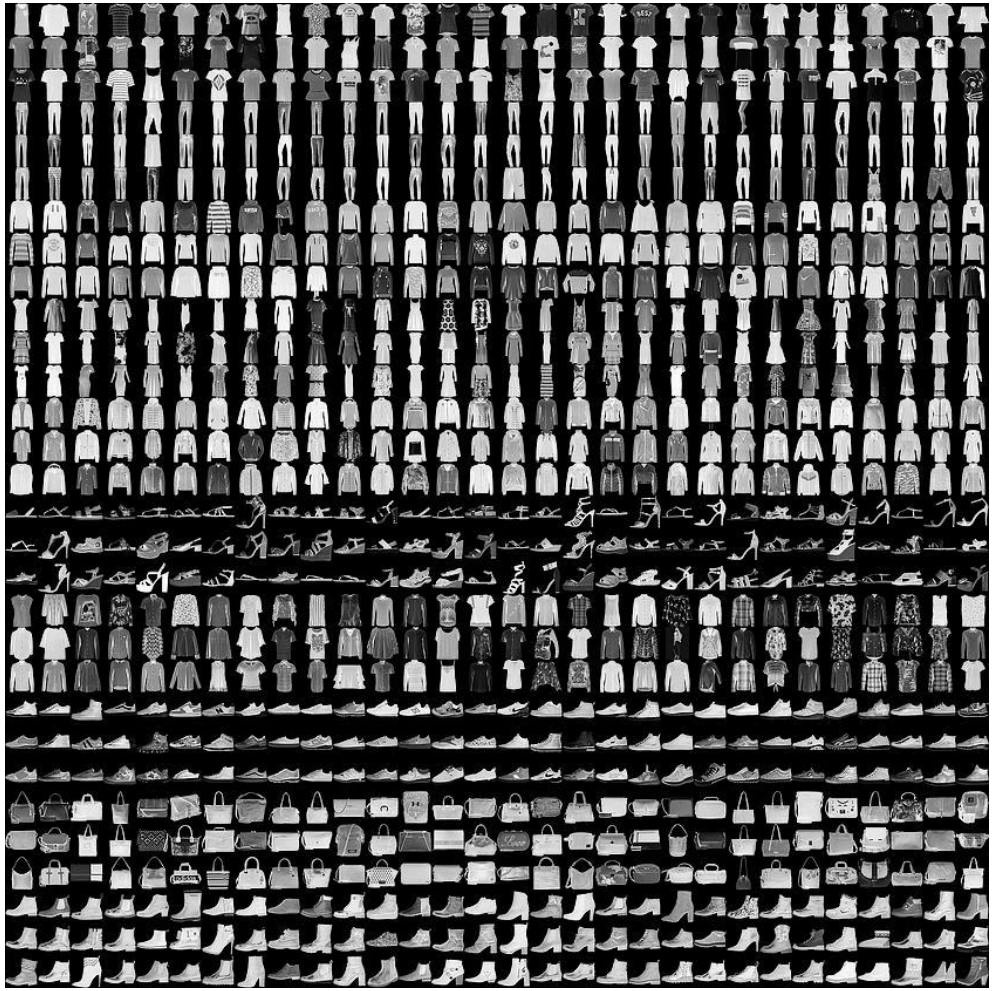
PARKINSON

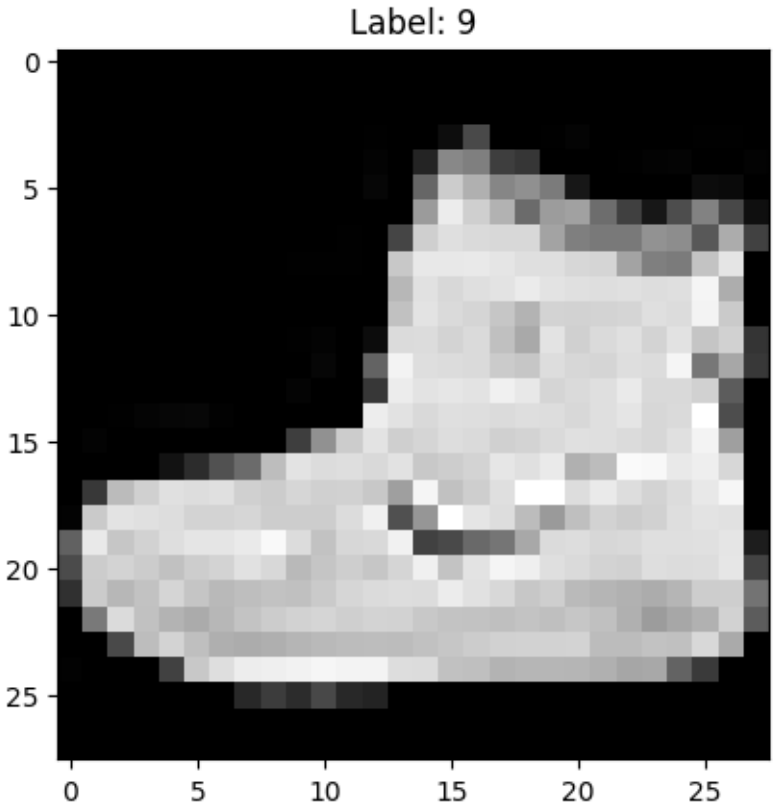
CANCER DE PIEL

El reto de los datos de imagen

- Trate de describir la imagen ->
- Ahora, trate de describirla alguien que nunca ha visto la ropa
- ¿Cómo aprendemos a reconocer los distintos tipos de prendas?

Ejemplo de Dataset Fashion MNIST





28 x 28 = 784 pixels

Linear

ReLU

Linear

1
2
783
784
1
2
19
20
1
2
19
20
1
2
9
10

y' =

0.0000
5.3003
2.1616
1.9145
0.0000
5.1698
0.0000
2.2152
0.0000
7.0417

Ankle boot

Input layer

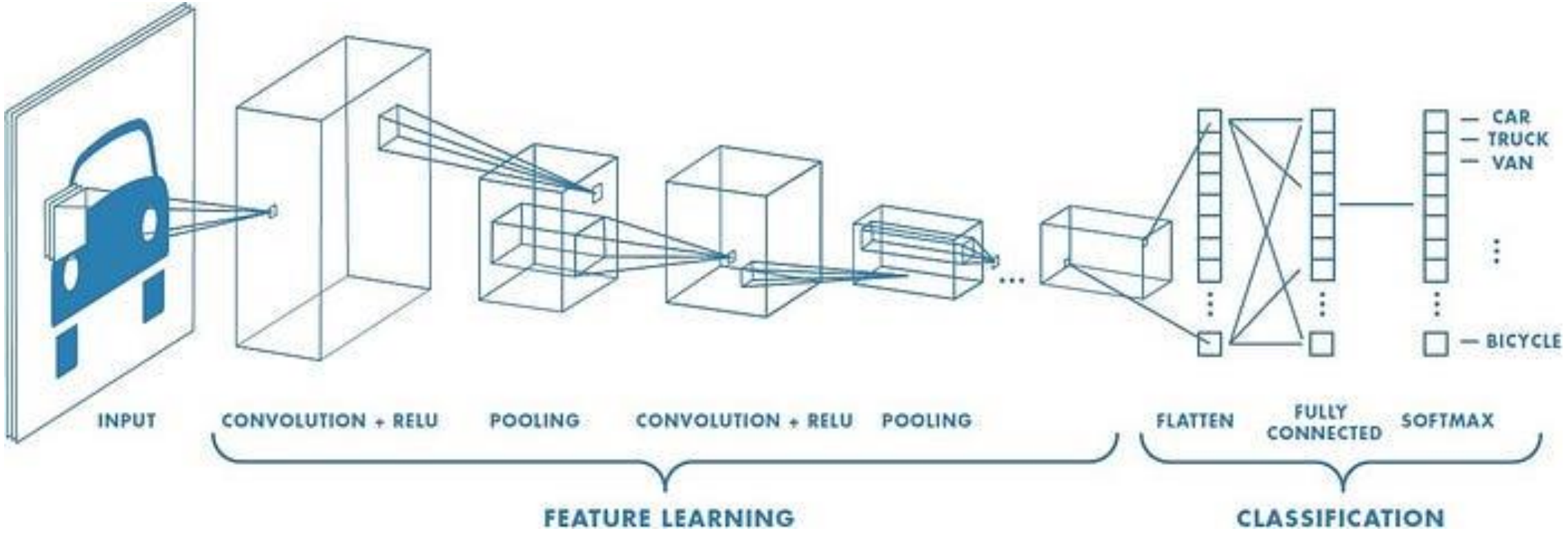
Hidden layer

Output layer

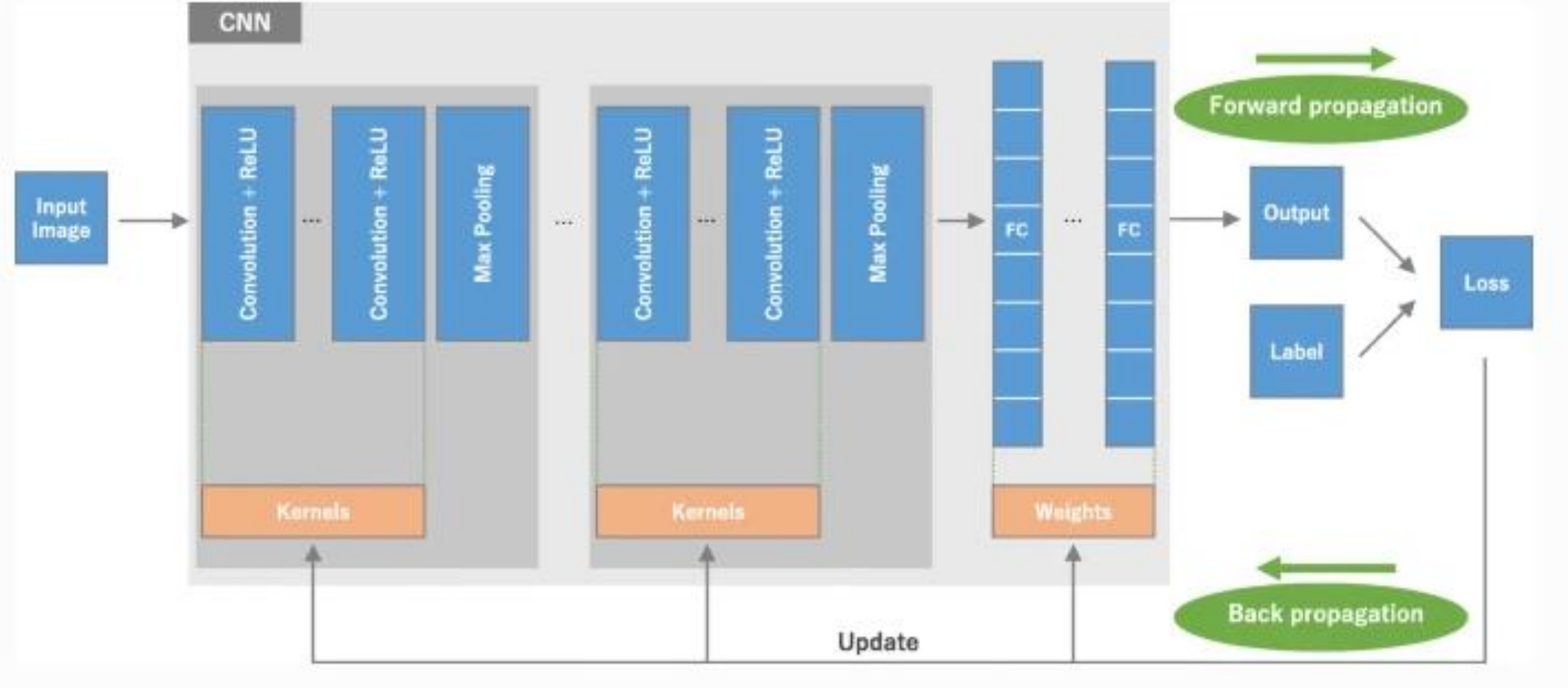
Convolutional Neural Networks (CNNs)

- **¿Que son las CNNs?**
- Un tipo especializado de red neuronal
- Diseñada específicamente para procesar datos en forma de cuadrícula
- Utilizada principalmente para el análisis de imágenes, pero también aplicable a otros tipos de datos
- Inspirada en la corteza visual de los animales

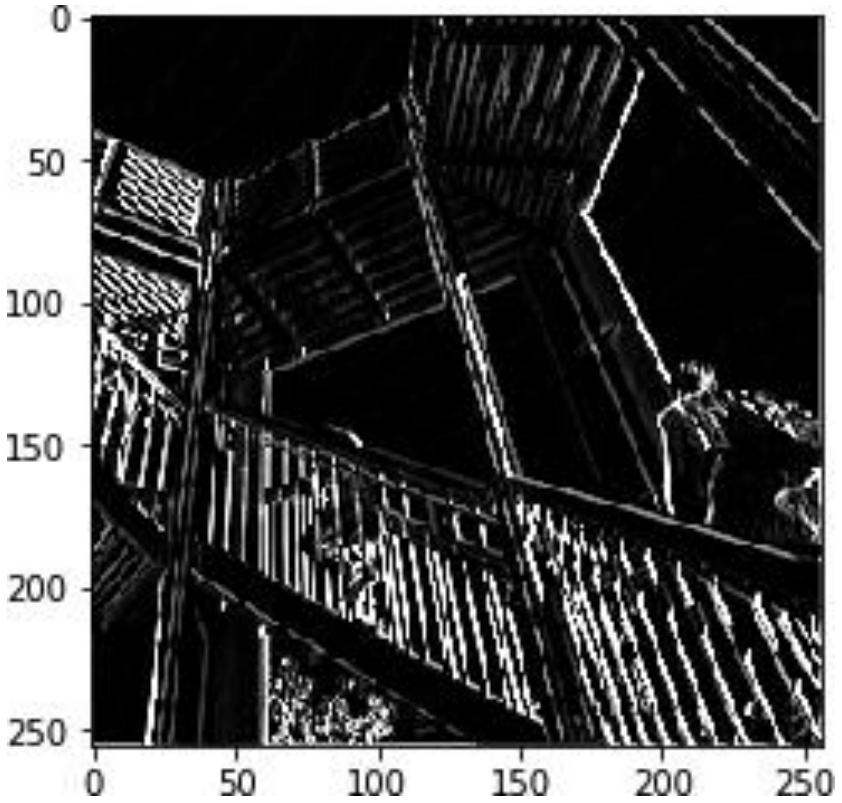
Arquitectura general de una CNN



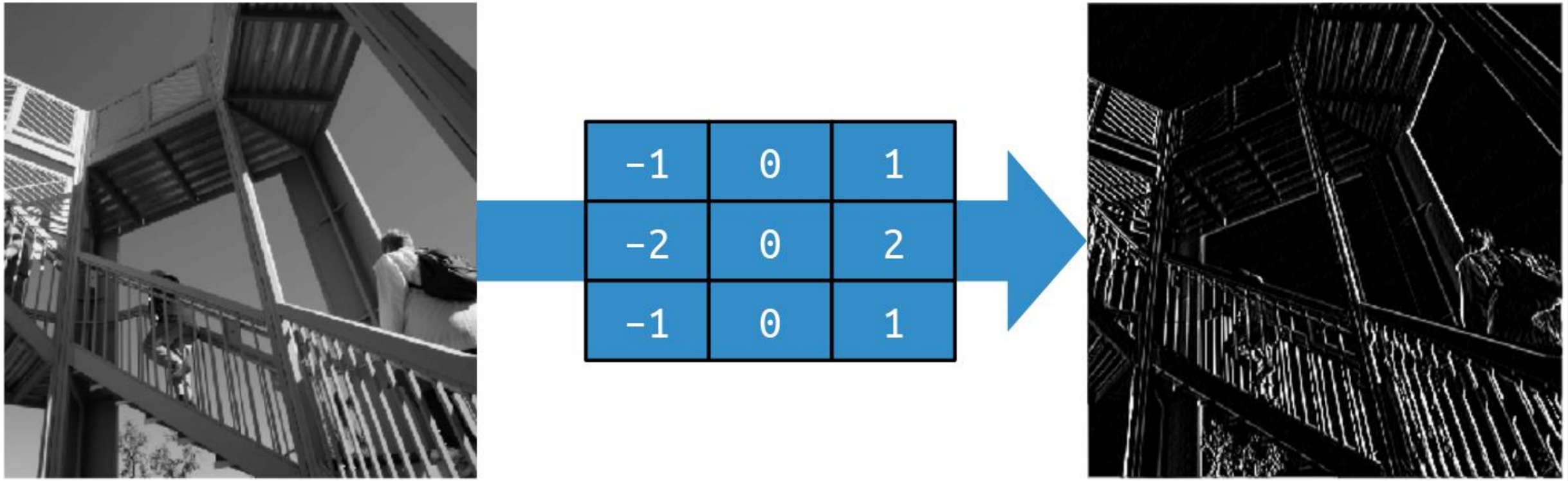
Convolutional neural network (CNN) architecture and the training process



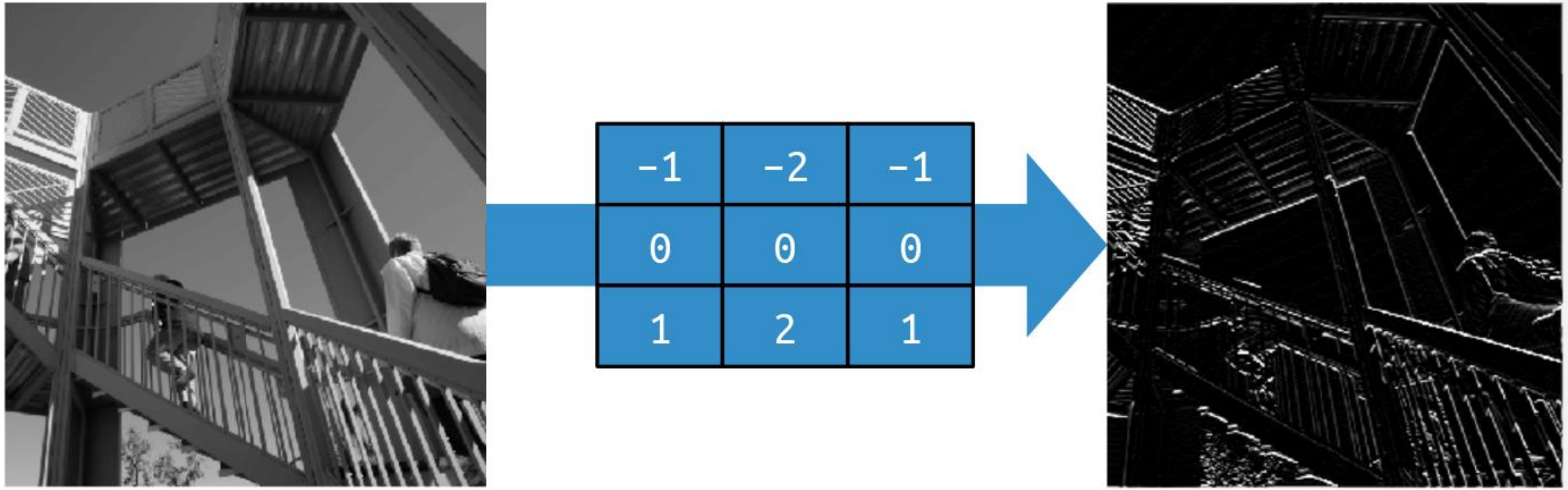
Max pooling



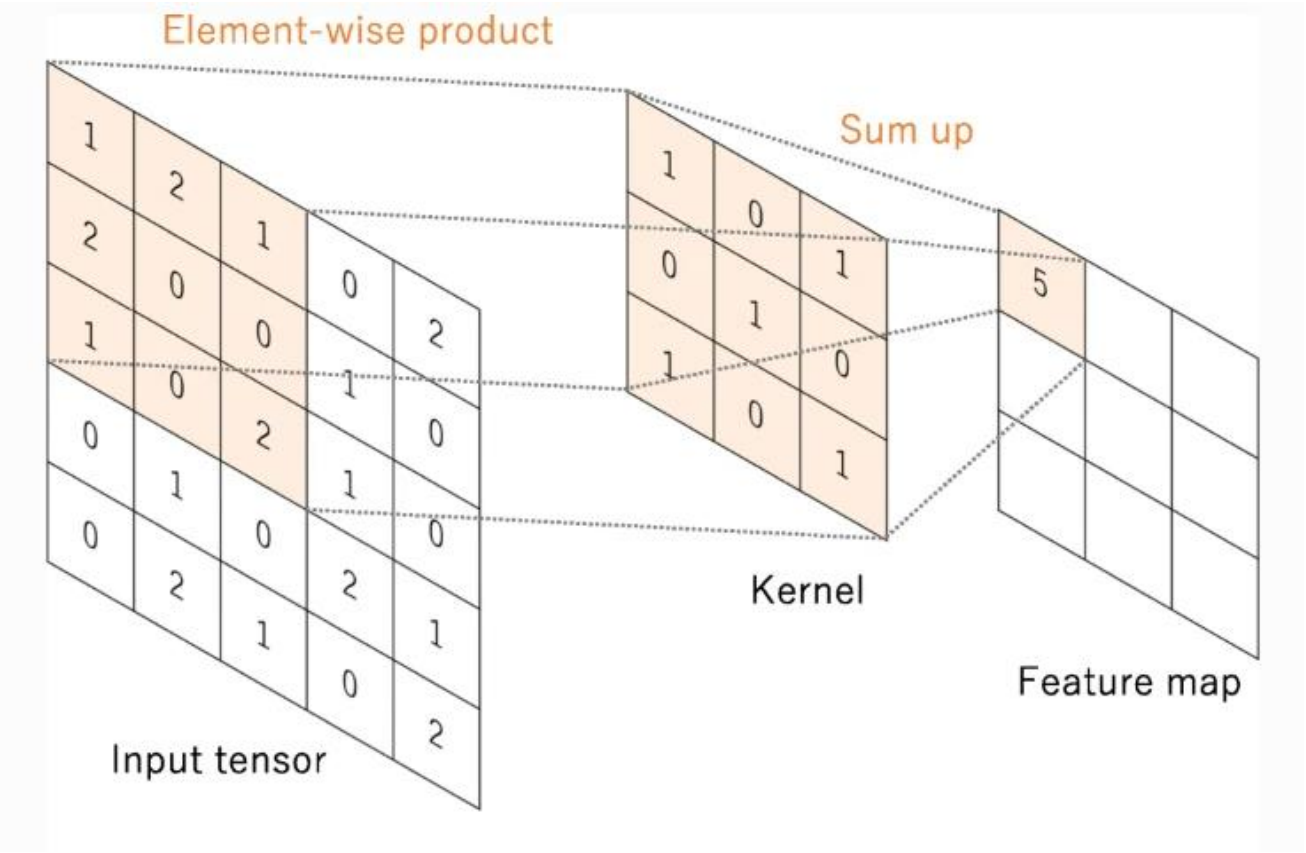
Convolutions

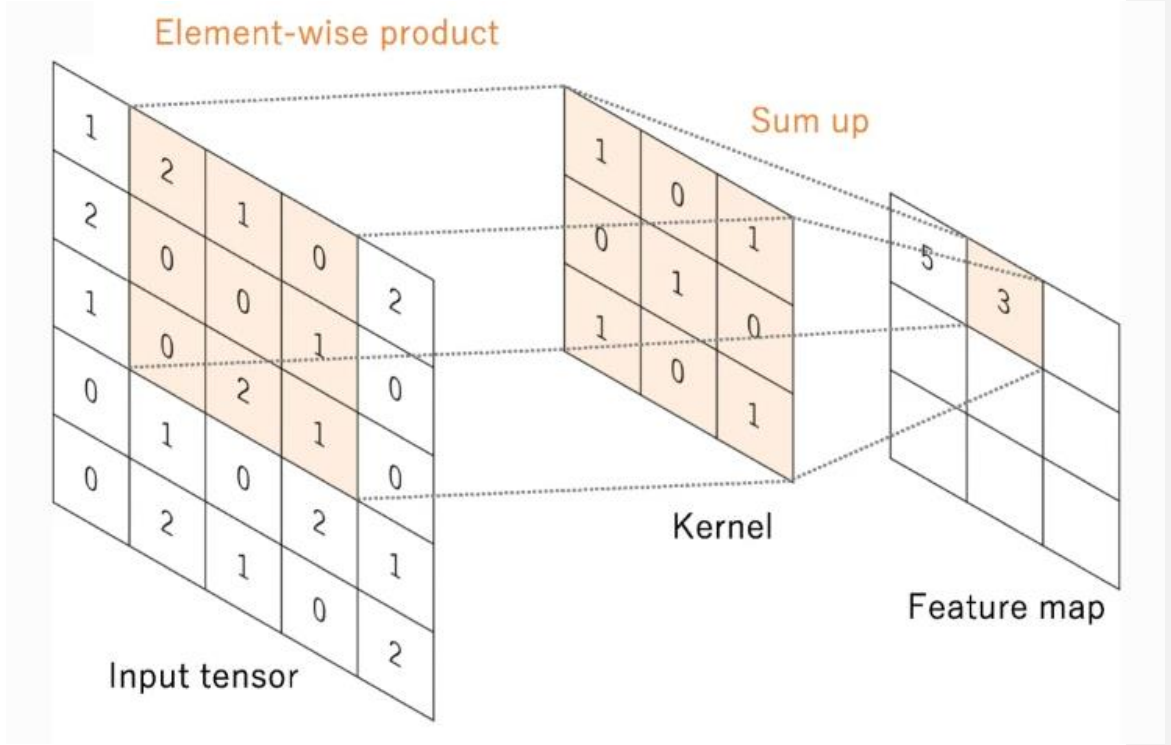


Convolutions

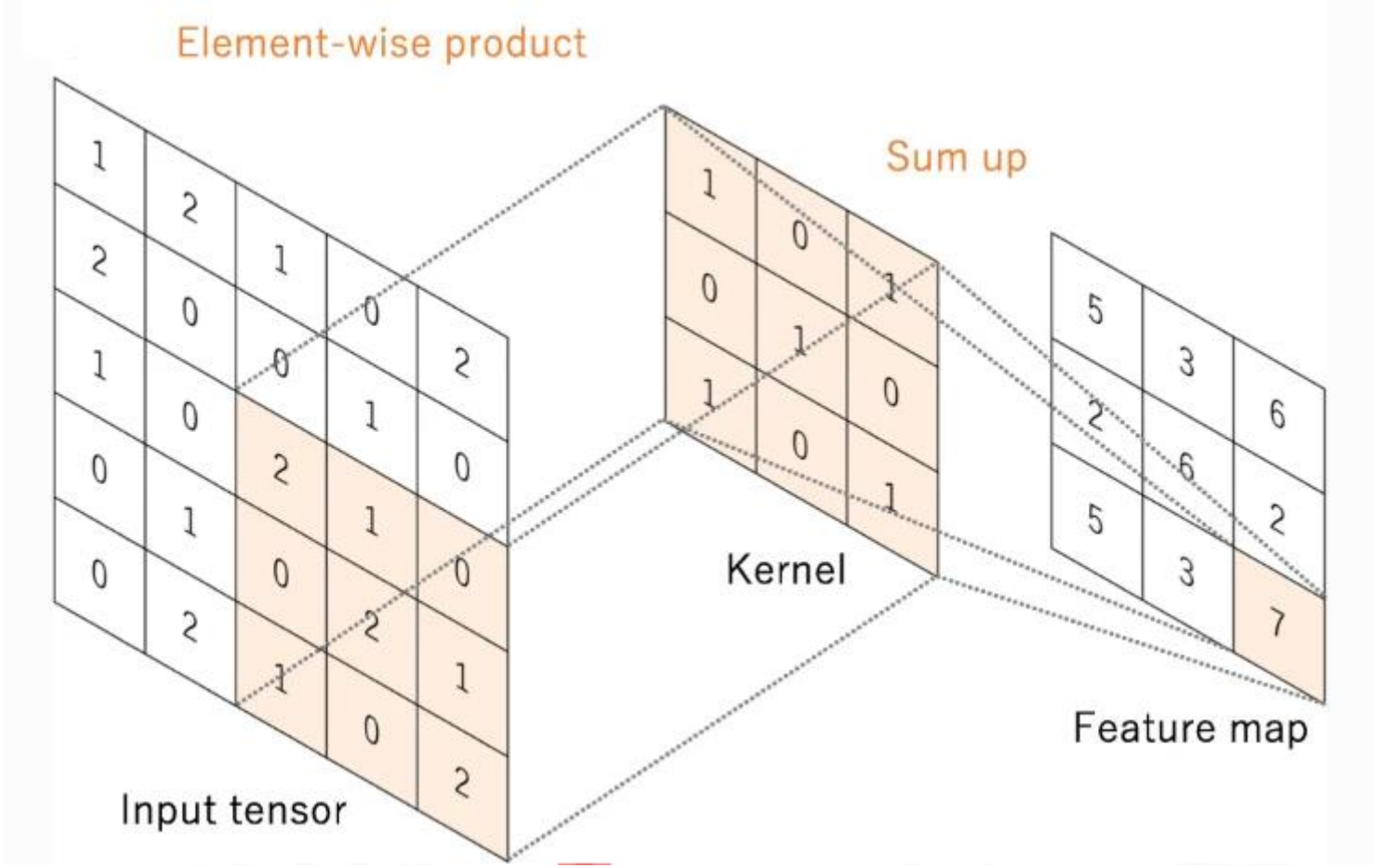


Convolutions



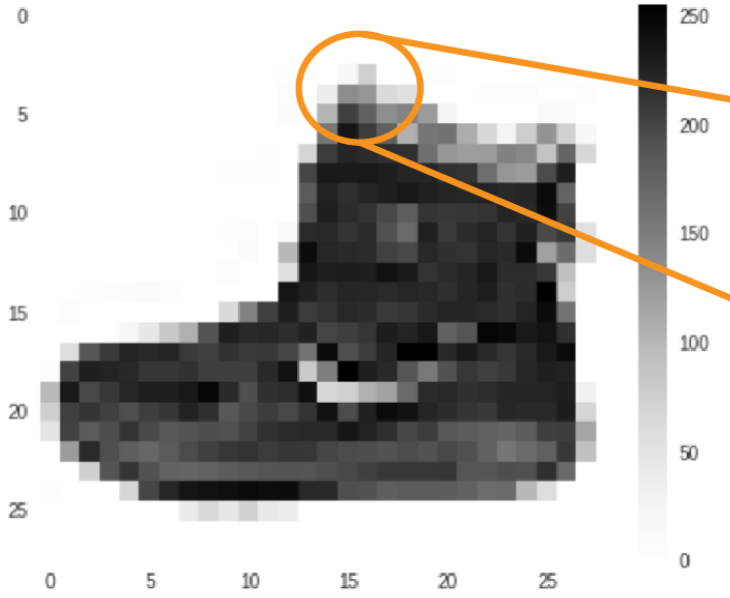


Convolutions



Convolutions

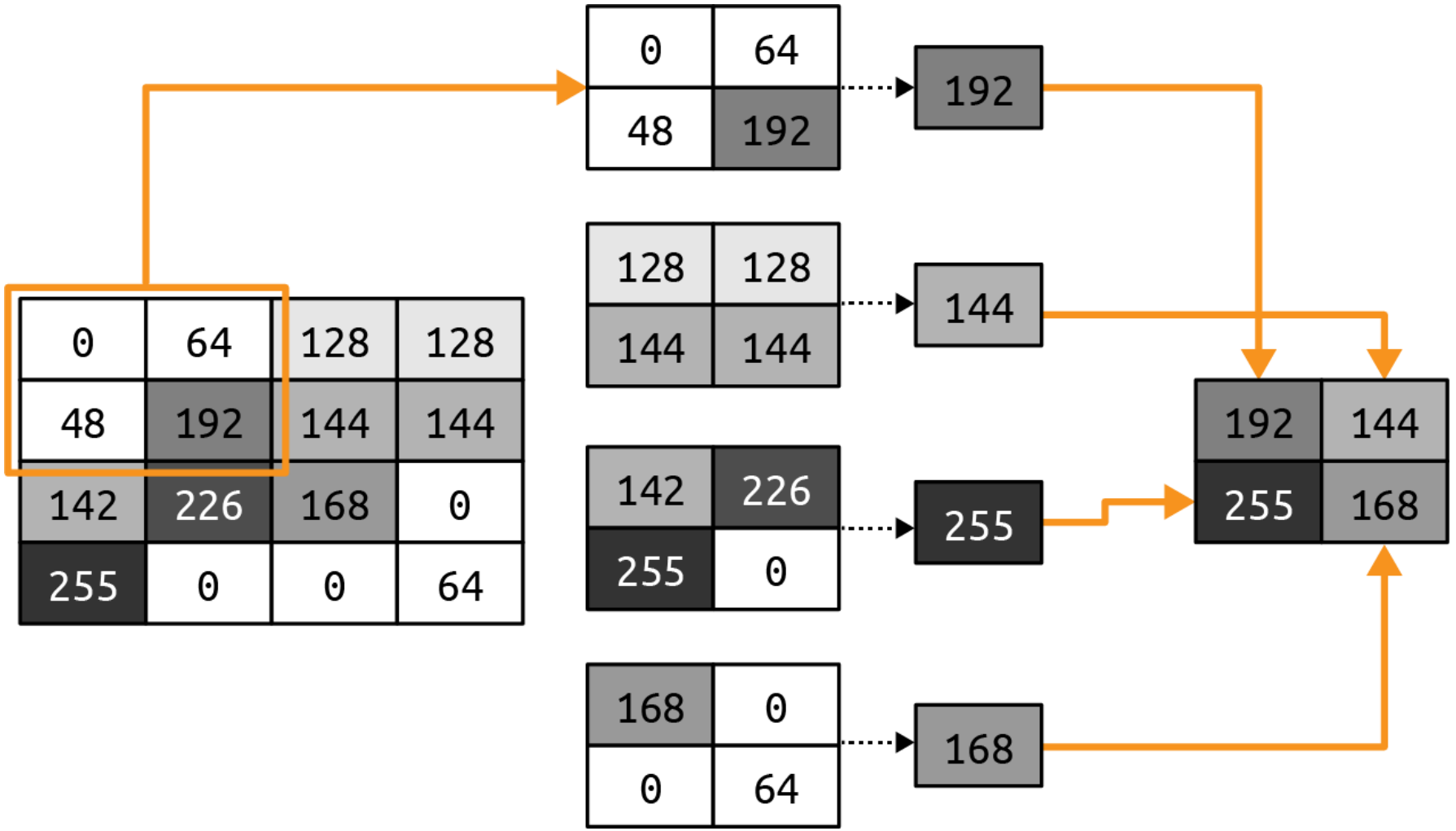
- A convolution is simply a filter of weights that are used to multiply a pixel with its neighbors to get a new value for the pixel.



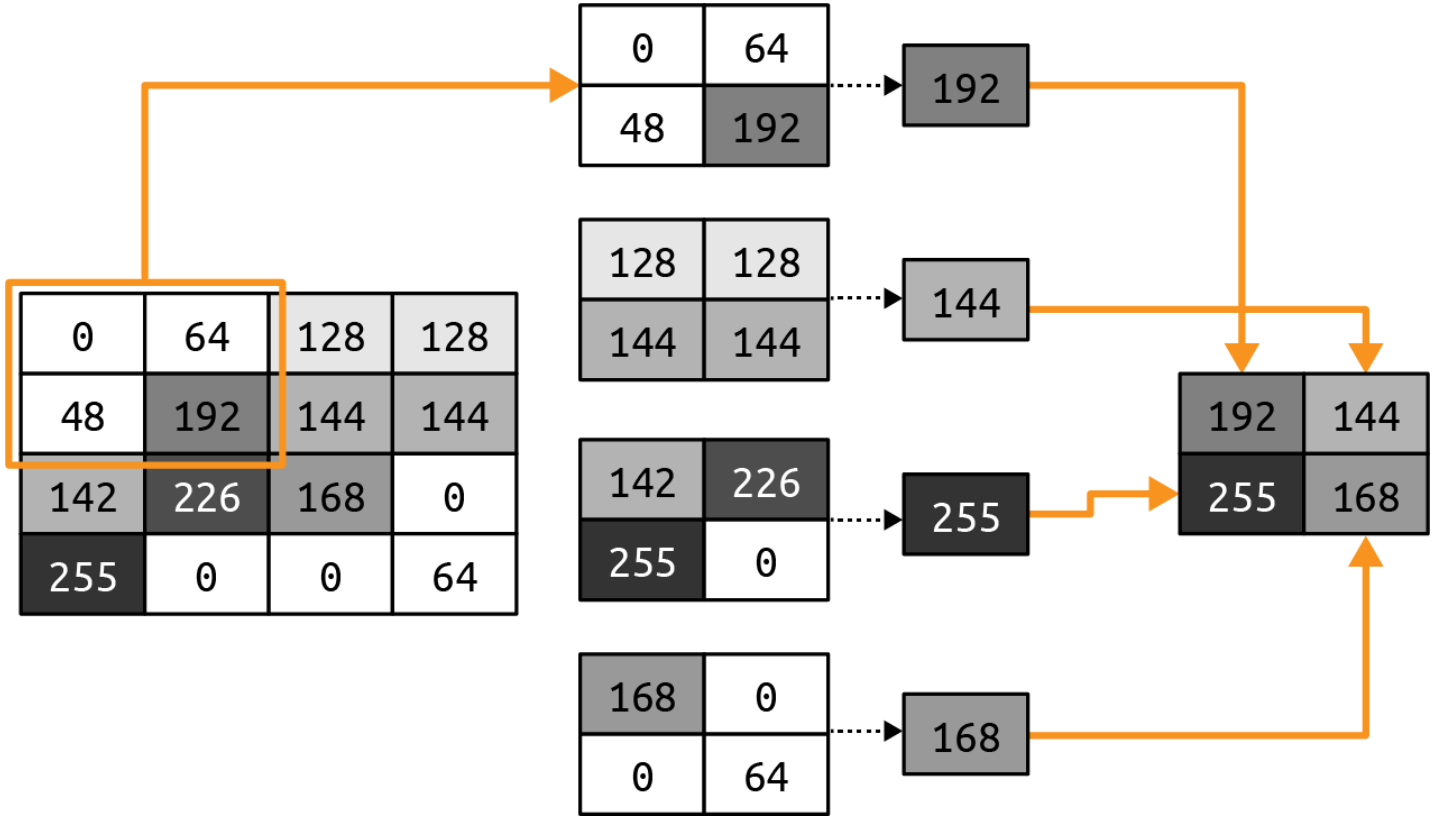
0	64	128
48	192	144
142	226	168

-1	0	-2
.5	4.5	-1.5
1.5	2	-3

Pooling

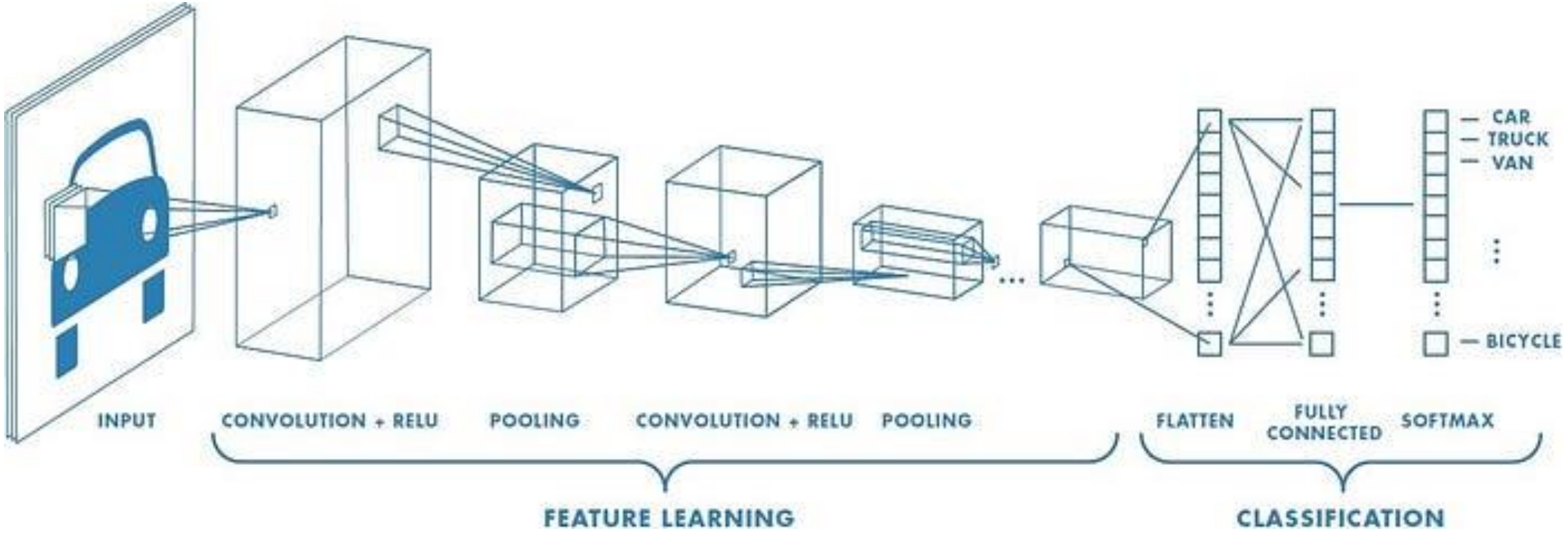

0 64 128 128
48 192 144 144
142 226 168 0
255 0 0 64
0 64
48 192
128 128
144 144
142 226
255 0
168 0
0 64
192
144
255
168
192 144
255 168

Pooling



Pooling es el proceso de eliminar píxeles de la imagen manteniendo la semántica del contenido de la imagen.

Arquitectura general de una CNN



Convolutions

- La cantidad de información en la imagen se reduce
- Potencialmente podemos aprender un conjunto de filtros que reducen la imagen a características
- Esas características se pueden emparejar con etiquetas como antes

Objetivo de la operación de convolución

- **Múltiples capas convolucionales:** Las CNN pueden tener múltiples capas convolucionales.
- **Primera capa convolucional:** Captura características de bajo nivel (por ejemplo, bordes, color, orientación del gradiente).
- **Capas subsiguientes:** Capturan características de nivel superior, basándose en las características aprendidas en las capas anteriores.
- **Comprensión global:** Una CNN con múltiples capas desarrolla una comprensión global de las imágenes del conjunto de datos, similar a la visión humana.

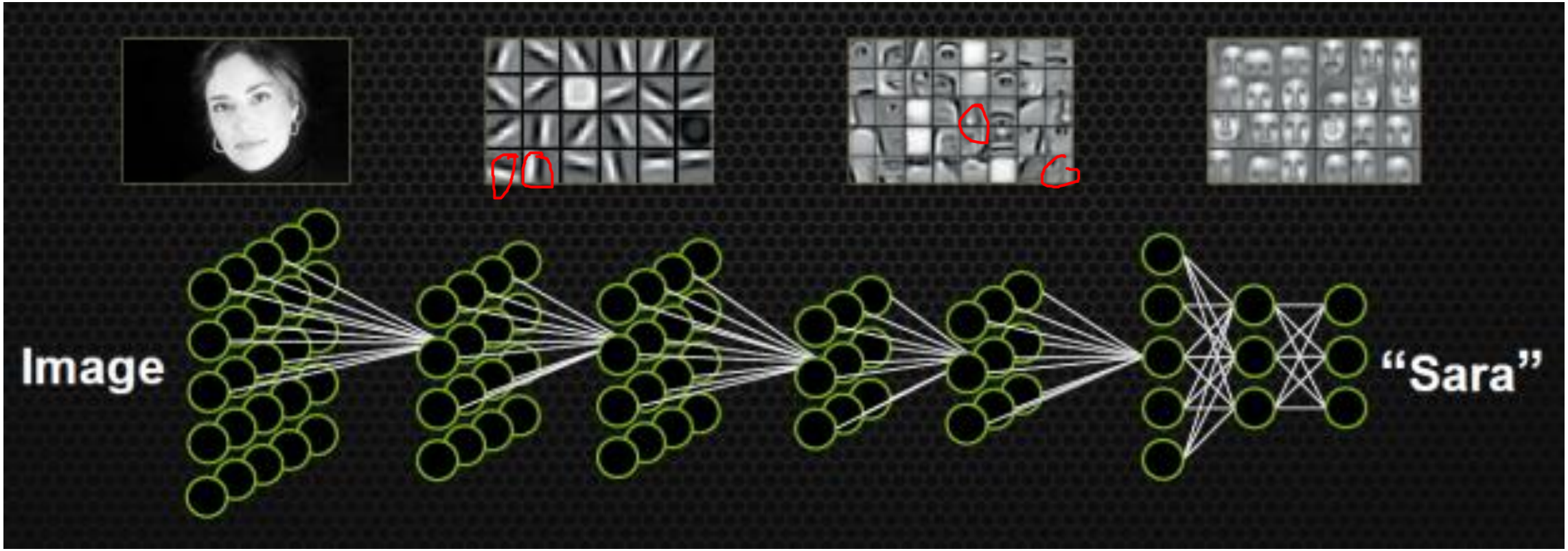
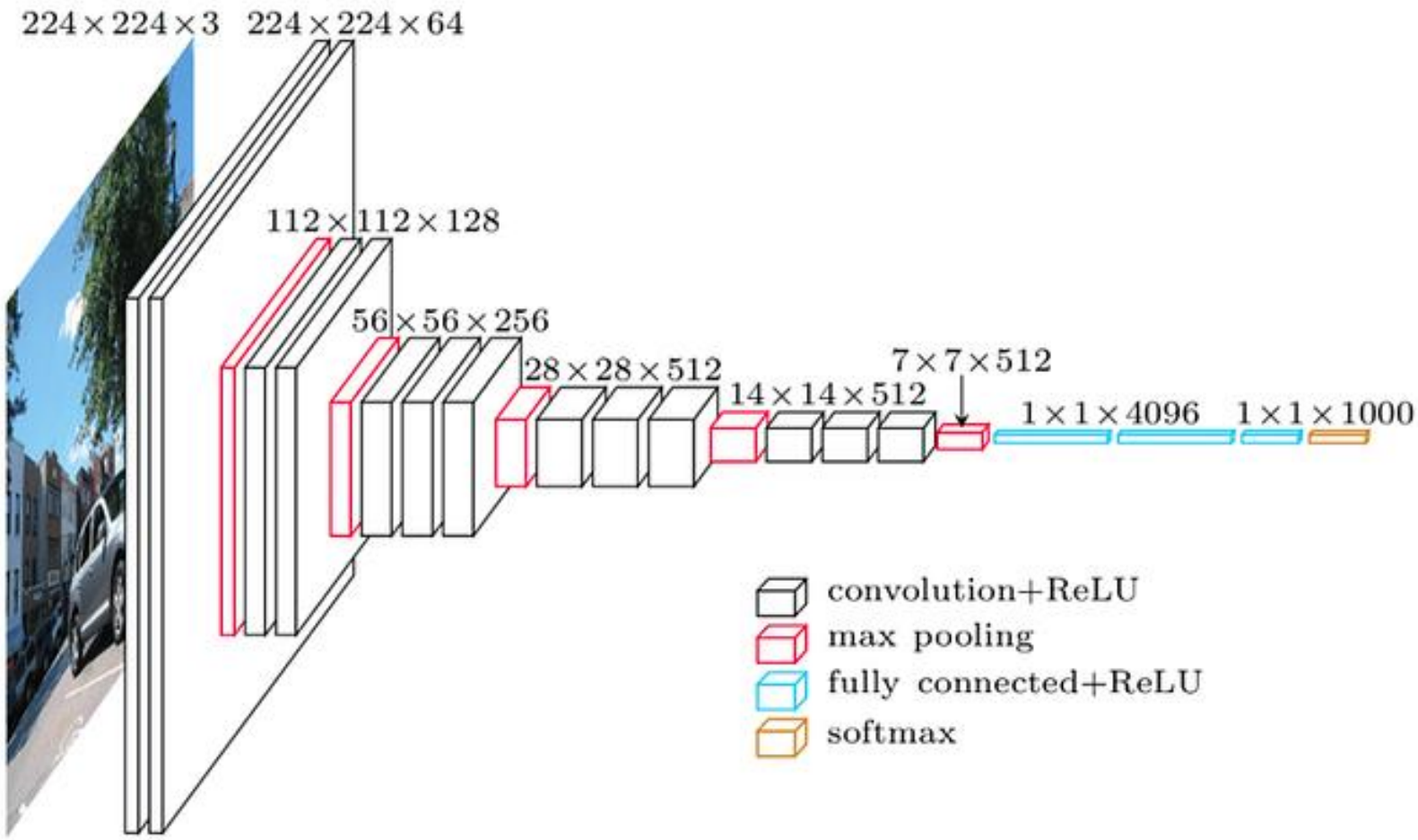


Image
"Sara"

Example of architecture: VGG16



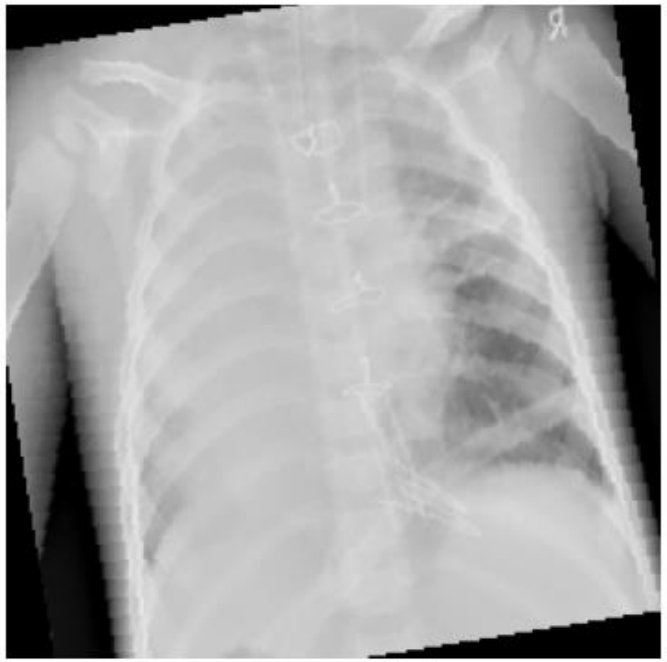
Code Example

Clasificación de Imágenes Medicas

- **El Problema:** Clasificar radiografías de tórax para detectar si un paciente tiene neumonía o no. Es un problema de **clasificación binaria**.
- **El Dataset: "Chest X-Ray Images (Pneumonia)"** en Kaggle.
  - Link: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia
  - **¿Por qué es bueno?** Está bien organizado en carpetas (train, test, val) y subcarpetas por clase (NORMAL, PNEUMONIA), lo que facilita enormemente la carga de datos en PyTorch.
- **Conceptos Clave a aprender:**
  - Carga y transformación de imágenes (torchvision.transforms).
  - Creación de DataLoaders para manejar los datos en batches.
  - Construcción de una **Red Neuronal Convolucional (CNN)** desde cero con torch.nn.
  - El ciclo de entrenamiento en PyTorch: forward pass, cálculo de la pérdida (loss), backward pass y actualización de pesos (optimizer).

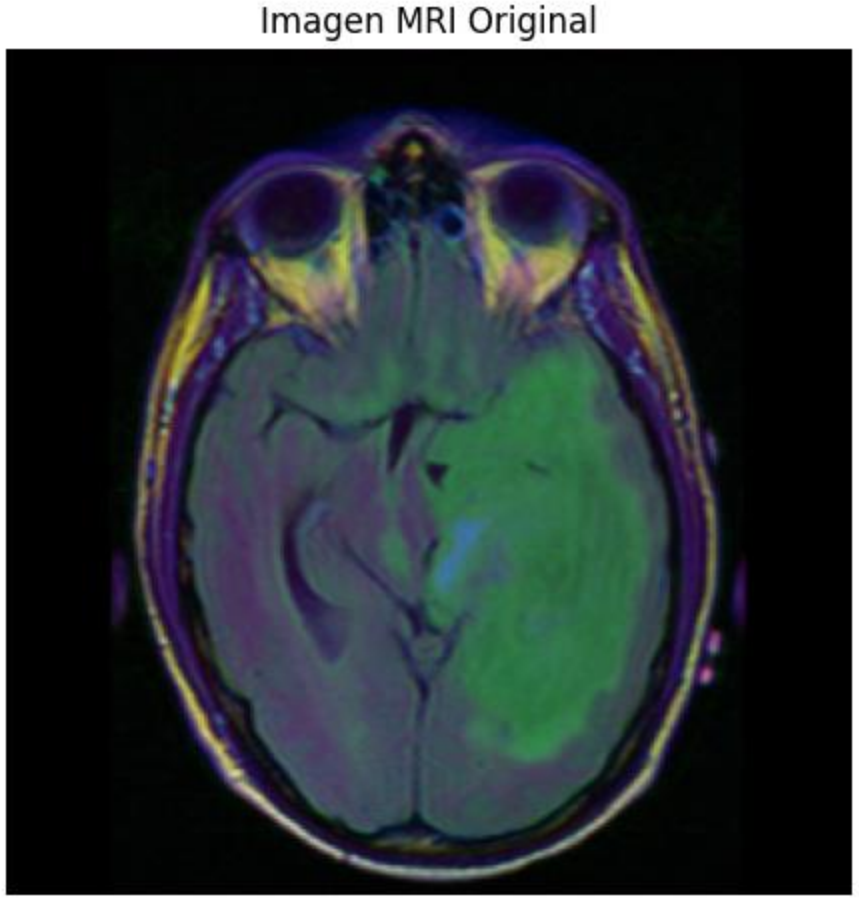




Code Example

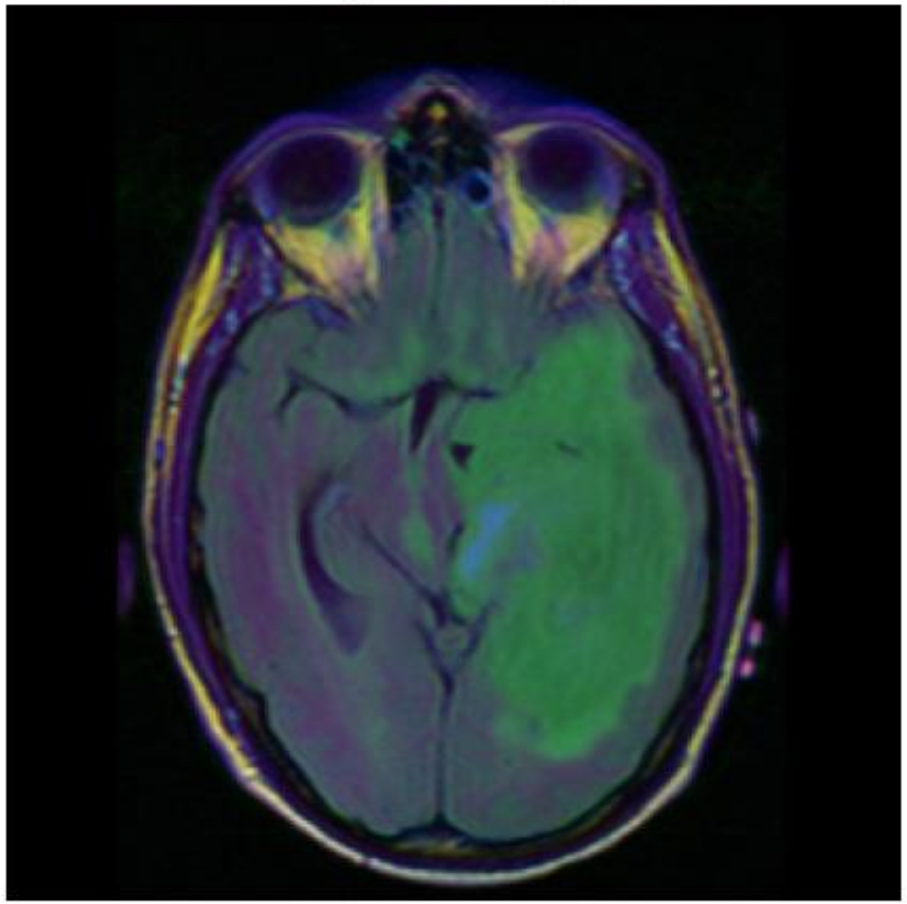
De Clasificar a Segmentar: El "Cirujano Digital"



Modelo de IA: **Tumor 98%**

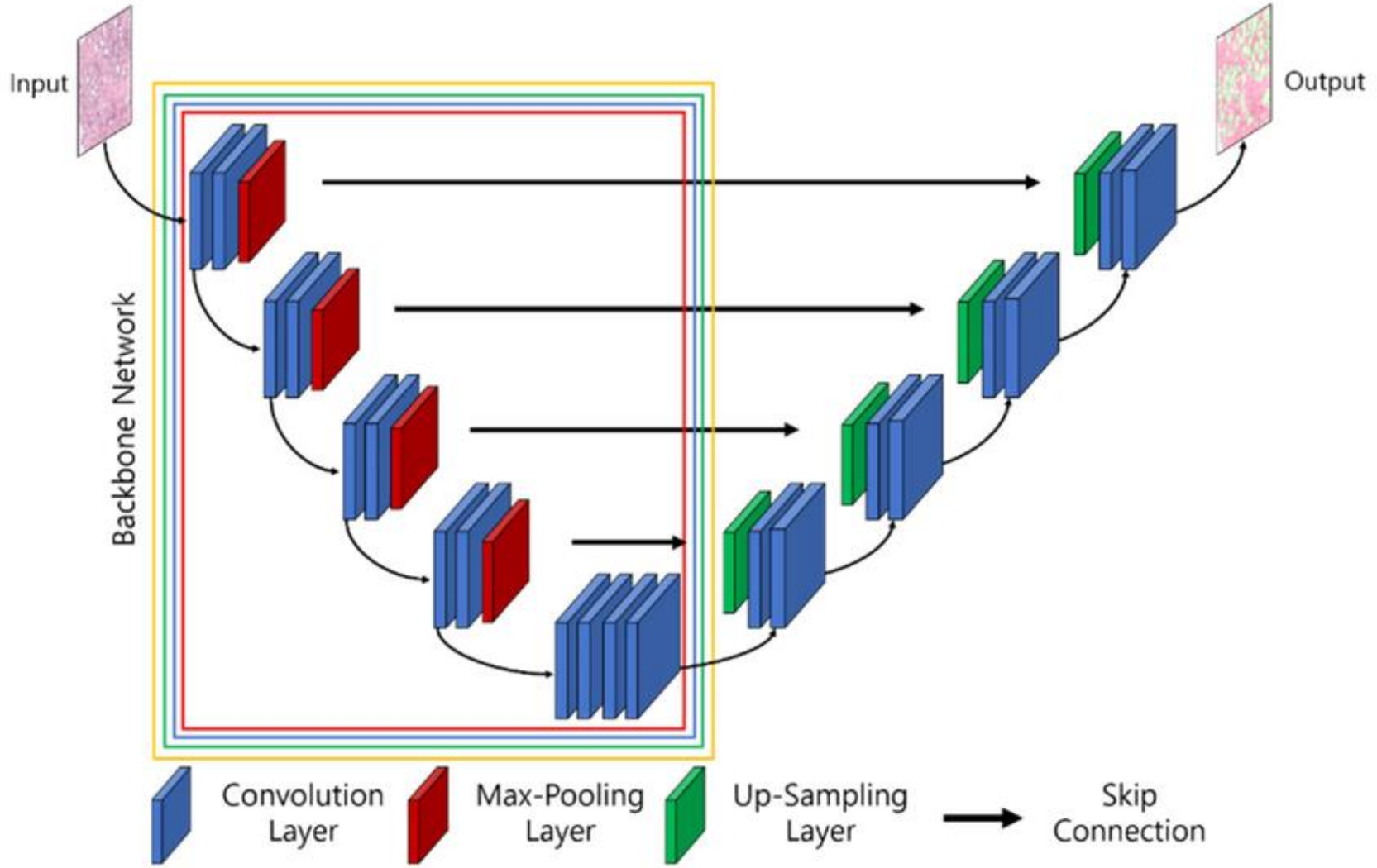
De Clasificar a Segmentar: El "Cirujano Digital"

Imagen MRI Original



Máscara del Tumor (Ground Truth)

U-Net: La Arquitectura Estrella para Segmentación Biomédica



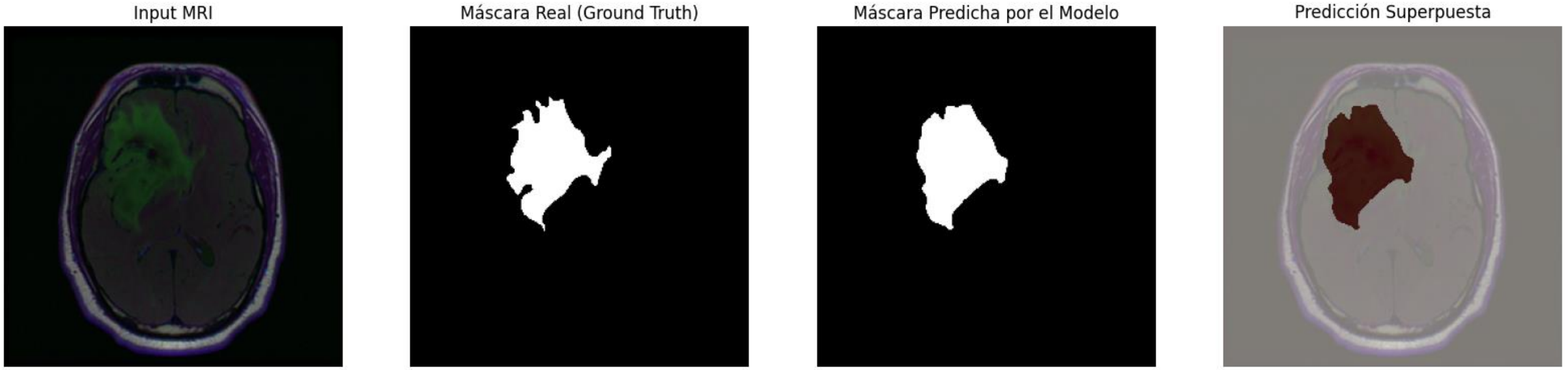
U-Net: La Arquitectura Estrella para Segmentación Biomédica

- **Parte Izquierda (Encoder / Camino de Contracción): "El ¿Qué?"**. Esta parte de la red es como una CNN de clasificación. Analiza la imagen y aprende a identificar las *características* de un tumor (textura, brillo, bordes...). Reduce el tamaño de la imagen pero aumenta la información de las características.
- **Parte Derecha (Decoder / Camino de Expansión): El ¿Dónde?"**. Esta parte toma las características aprendidas y reconstruye la imagen a su tamaño original, pero en lugar de la imagen original, produce una 'máscara' que localiza esas características en el espacio.
- **Flechas Horizontales (Skip Connections): "La Memoria"**. Este es el truco genial de la U-Net. Pasa información de alta resolución del Encoder directamente al Decoder. Esto ayuda al Decoder a hacer una localización muy precisa, evitando que la imagen se vea borrosa.

La Métrica - ¿Cómo medimos el "éxito"?

- Imagina dos círculos con una superposición (como un diagrama de Venn). Uno es la "Máscara Real" y el otro la "Máscara Predicha". La superposición es el "Área de Acuerdo".
- La "precisión por píxel" es una mala métrica aquí: "En una imagen de cerebro, >99% de los píxeles son 'no tumor'. Un modelo tonto que predice siempre 'no tumor' tendría una precisión del 99% y sería totalmente inútil".
- Coeficiente de Dice (Dice Score):
  - Dice = (2 * Área de Superposición) / (Área Real + Área Predicha)
  - Mide qué tan bien se superponen las dos formas. Un Dice de 1.0 es una superposición perfecta. Un Dice de 0.0 es ningún tipo de superposición. Es la métrica estándar para segmentación.
- Mensaje clave: Necesitamos métricas inteligentes que entiendan la tarea. El Dice Score es perfecto para medir el solapamiento.

De Clasificar a Segmentar: El "Cirujano Digital"



Links fuentes de informacion

- Image Kernel
  - https://setosa.io/ev/image-kernels/
- CNN
  - https://www.youtube.com/watch?v=V8j1oENVz00&t=139s&ab_channel=DotCSV
  - https://www.youtube.com/watch?v=KuXjwB4LzSA&t=1s&ab_channel=3Blue1Brown
  - https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9
  - https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
  - https://dev.to/sandeepbalachandran/machine-learning-convolution-with-color-images-2p41
  - https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide
- Feature visualization
  - https://towardsdatascience.com/feature-visualization-on-convolutional-neural-networks-keras-5561a116d1af
- More on the topic of bias:
  - https://towardsdatascience.com/texture-vs-shape-the-bias-in-cnns-5ee423edf8db


intel

Extras slides

- https://github.com/castolosan/univa_examples

¿Cómo funciona nn.BCEWithLogitsLoss?

- nn: Viene de la librería de redes neuronales de PyTorch.
- BCE: Significa **B**inary **C**ross-**E**ntropy (Entropía Cruzada Binaria). Esta es la fórmula matemática estándar y más efectiva para problemas de **clasificación binaria** (dos clases, como "Neumonía" vs. "Normal").
- WithLogits: ¡Esta es la parte más importante y una gran ventaja de PyTorch!
  - "Logits" son los valores de salida **crudos** de tu modelo, antes de pasar por una función de activación como Sigmoid.
  - Esta función combina dos pasos en uno:
    - Aplica internamente una función Sigmoid a las salidas de tu modelo para convertirlas en probabilidades (un número entre 0 y 1).
    - Calcula la pérdida de Entropía Cruzada Binaria.
  - Hacerlo en un solo paso es **numéricamente más estable** y previene errores comunes.

¿Cómo funciona optim.Adam?

- optim.Adam: Esta usando el algoritmo **Adam**, que es uno de los optimizadores más populares y efectivos. Es un método "adaptativo", lo que significa que ajusta la tasa de aprendizaje para cada parámetro individualmente. Es una excelente opción por defecto para la mayoría de los problemas.
- modelo.parameters(): Con esta simple llamada, le estás entregando al optimizador una lista de **todos los parámetros aprendibles** de tu CNN_Neumonia. El optimizador ahora sabe cuáles son los "diales" que tiene que girar.
- lr=1e-4: lr significa **Learning Rate** (Tasa de Aprendizaje). Este es el hiperparámetro más importante.
  - En la analogía, es el **"tamaño del ajuste"** que el entrenador le dice al arquero.

¿Cómo funciona optim.lr_scheduler.StepLR ?

- Un scheduler (planificador de la tasa de aprendizaje) ajusta dinámicamente la tasa de aprendizaje (lr) durante el entrenamiento. La idea es empezar con un lr relativamente alto para avanzar rápidamente hacia una buena solución, y luego reducirlo para hacer ajustes más finos y no "pasarse de largo" del punto óptimo.
- StepLR: Es una estrategia específica de planificación. "Step" significa que reduce el lr en pasos discretos.
- optimizador: Le dices a qué optimizador debe gestionar.
- step_size=3: Le indica que debe reducir la tasa de aprendizaje **cada 3 épocas** de entrenamiento.
- gamma=0.1: Es el factor multiplicativo. Cuando el scheduler actúa, hace lo siguiente: nueva_lr = lr_actual * gamma. En tu caso, cada 3 épocas, la tasa de aprendizaje se dividirá por 10 (se multiplicará por 0.1).