Redes neuronales avanzadas: convolucionales

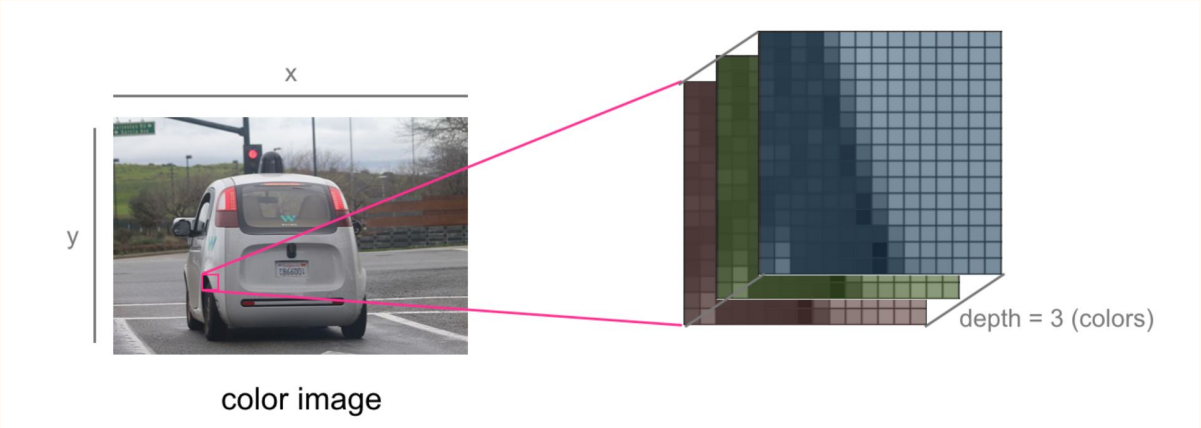
—

Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones - 2024

Imágenes

¿Cómo representamos imágenes?

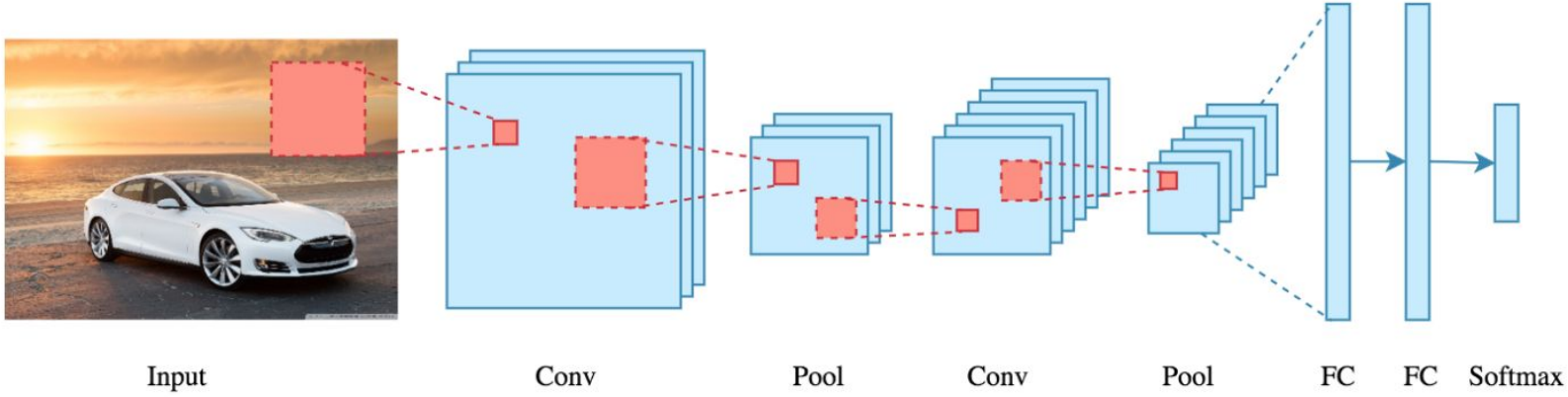
- Las imágenes se representan como matrices en un espacio de 3 dimensiones
- Tenemos:
  - Ancho
  - Alto
  - Profundidad (generalmente canales RGB - intensidades de 0 a 255)



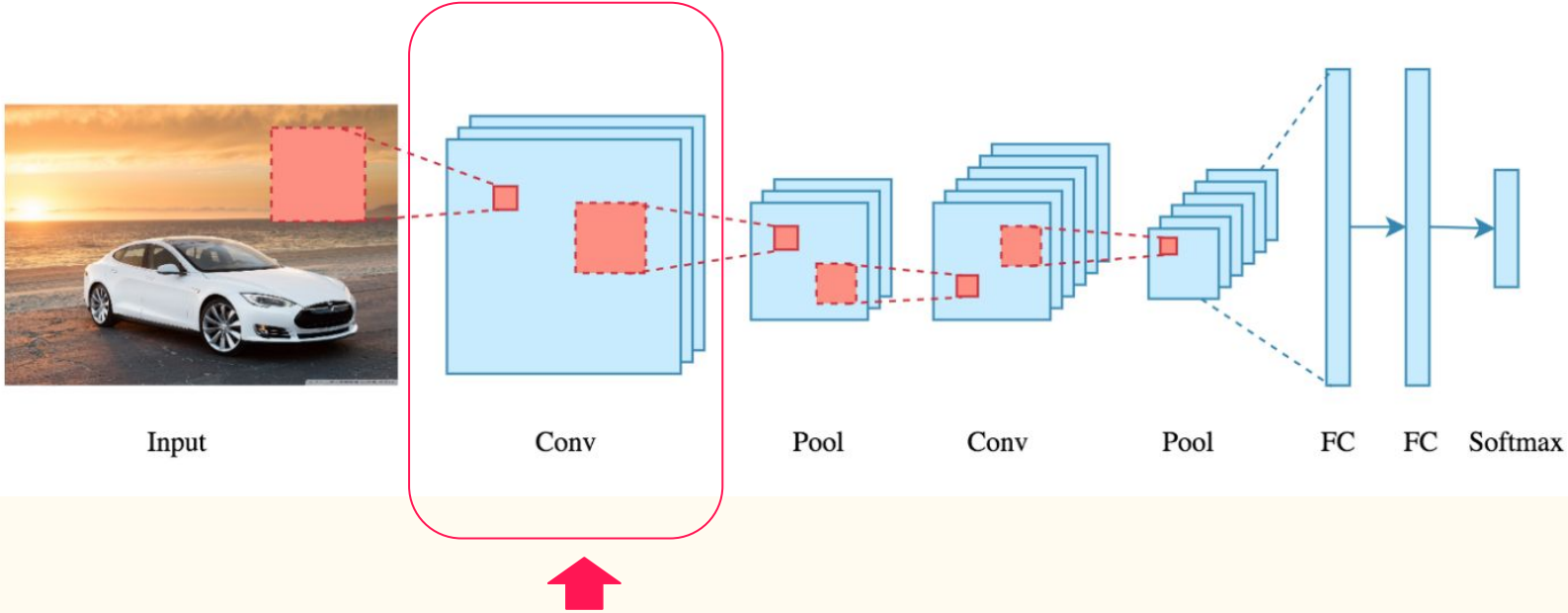
La importancia de los filtros (kernels)

Los filtros capturan distintos aspectos de una imagen: detección de bordes, desenfoque, más ejemplos.

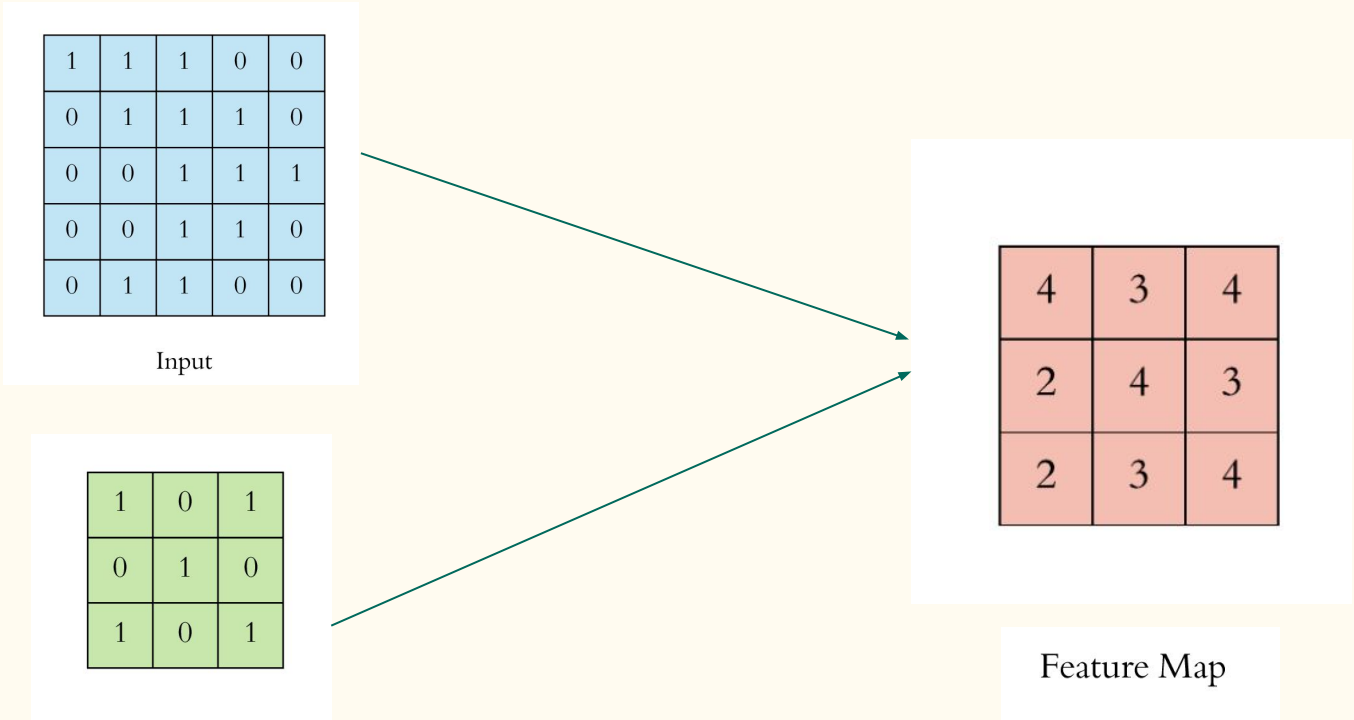
Una Arquitectura simple de CNN



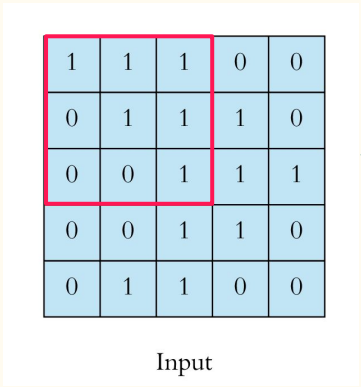
Una Arquitectura simple de CNN - Conv

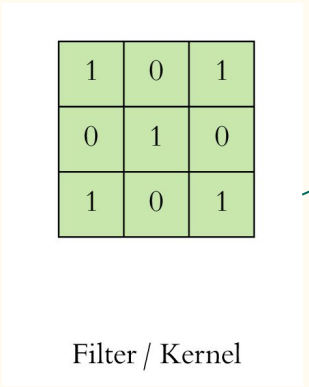


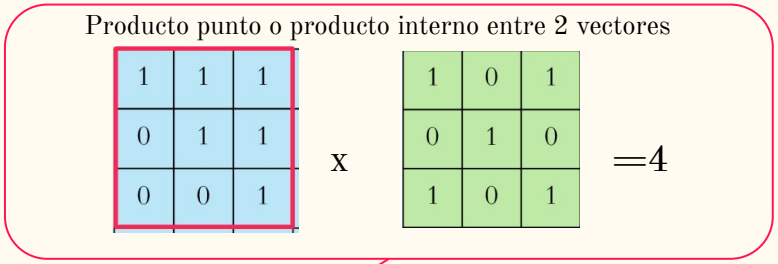
Convoluciones en 2D

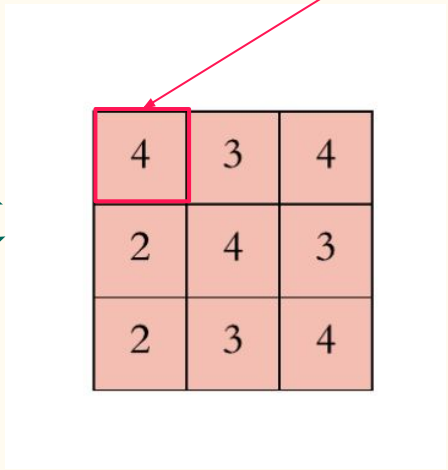


Convoluciones en 2D







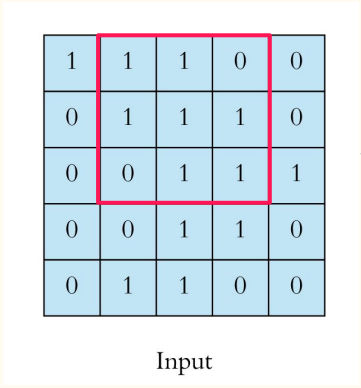


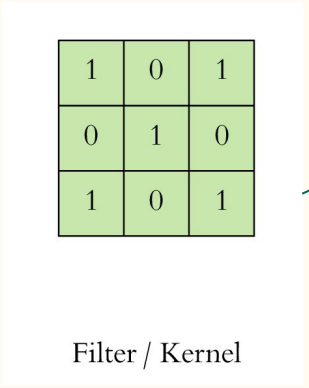
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

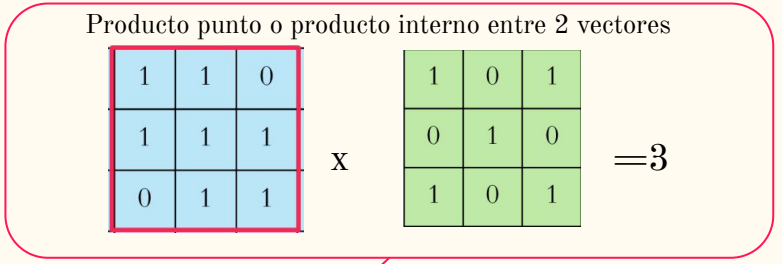
$1 \times 1 = 1$
$1 \times 0 = 0$
$1 \times 1 = 1$
$0 \times 0 = 0$
$1 \times 1 = 1$
$1 \times 0 = 0$
$0 \times 1 = 0$
$0 \times 0 = 0$
$1 \times 1 = 1$

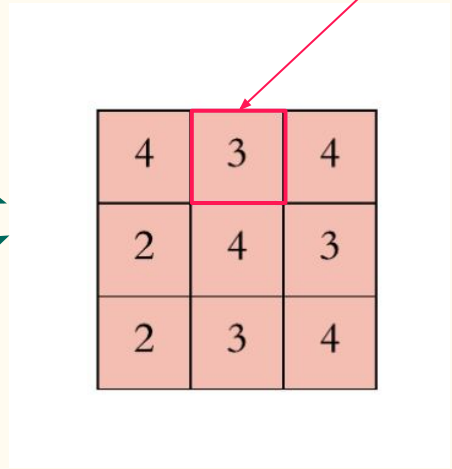
$\underline{4}$

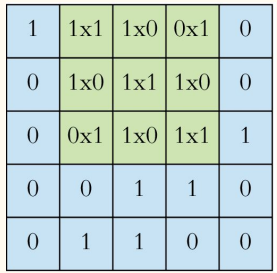
Convoluciones en 2D



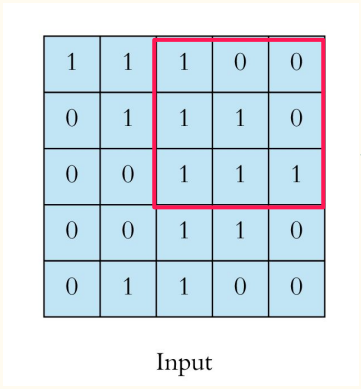


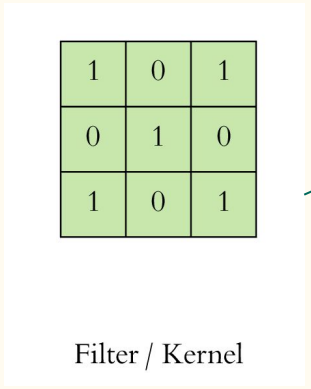


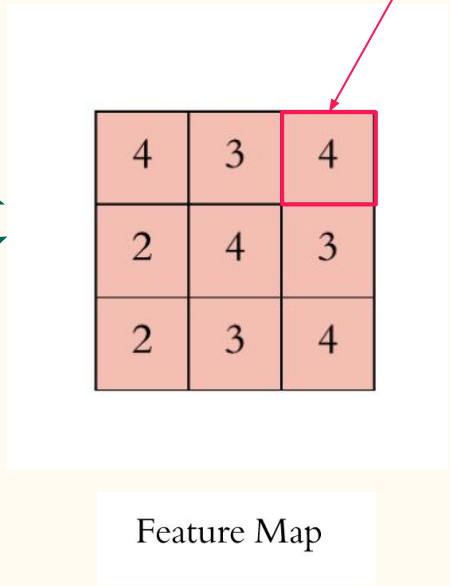


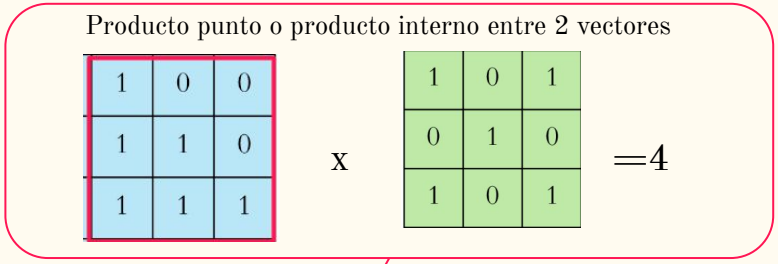


Convoluciones en 2D

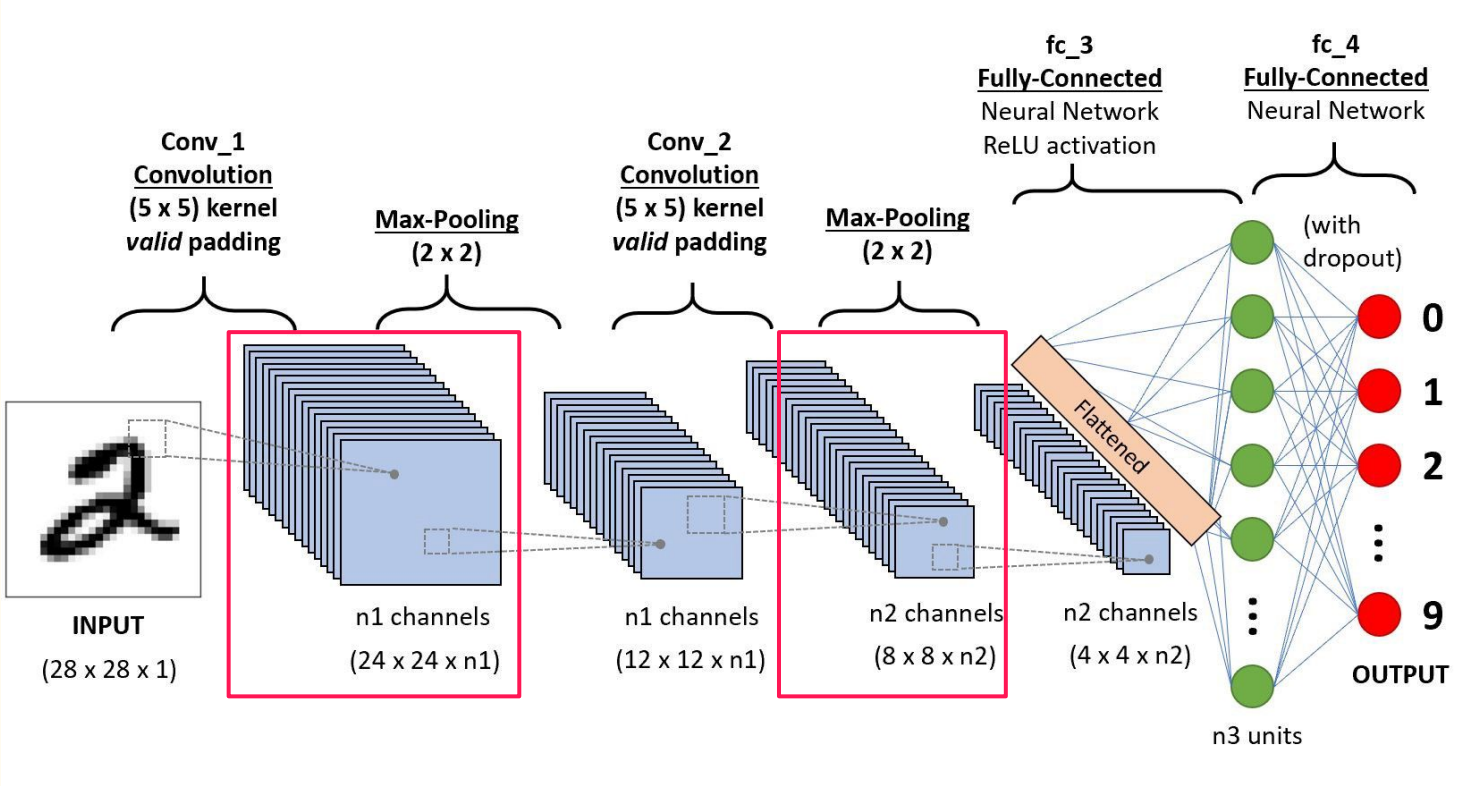




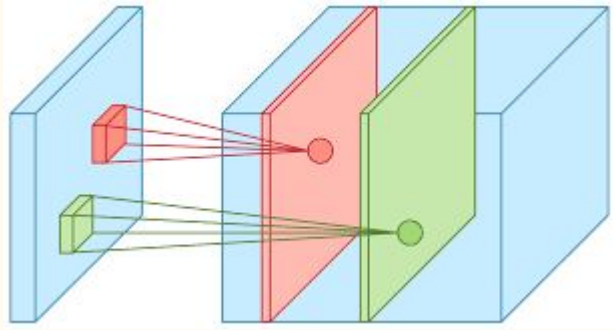




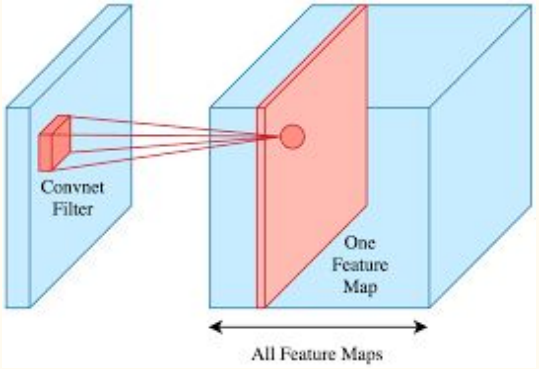
Formación de la Salida de la Capa Convolucional



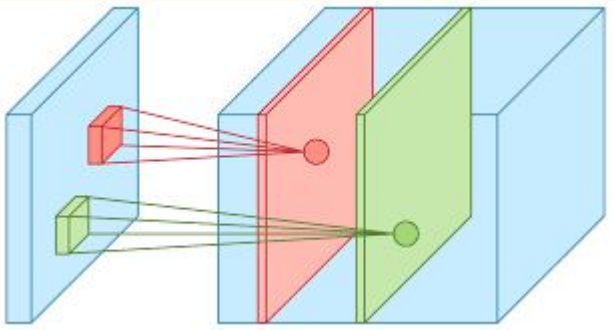
Kernel de Convolución -Tamaño

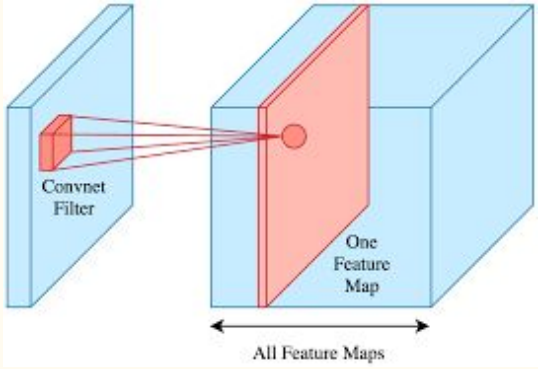


- Determina el **campo receptivo** de cada capa convolucional (extensión espacial de la entrada)
  - tamaño de **kernel grande** → información global
  - tamaño de **kernel pequeño** → características locales
- Los tamaños de **kernel comunes** son 3x3, 5x5 o 7x7
- Pueden emplearse un **kernel rectangular**, pero no es común.
- Tamaños de **kernel más pequeños** se utilizan a menudo en **capas más profundas** para capturar detalles de grano fino.
- Tamaños de **kernel más grandes** se utilizan en **capas tempranas** para capturar patrones más amplios
- Tamaños de **kernel más grandes aumenta la cantidad de parámetros** y la **complejidad computacional** del modelo



Feature Maps



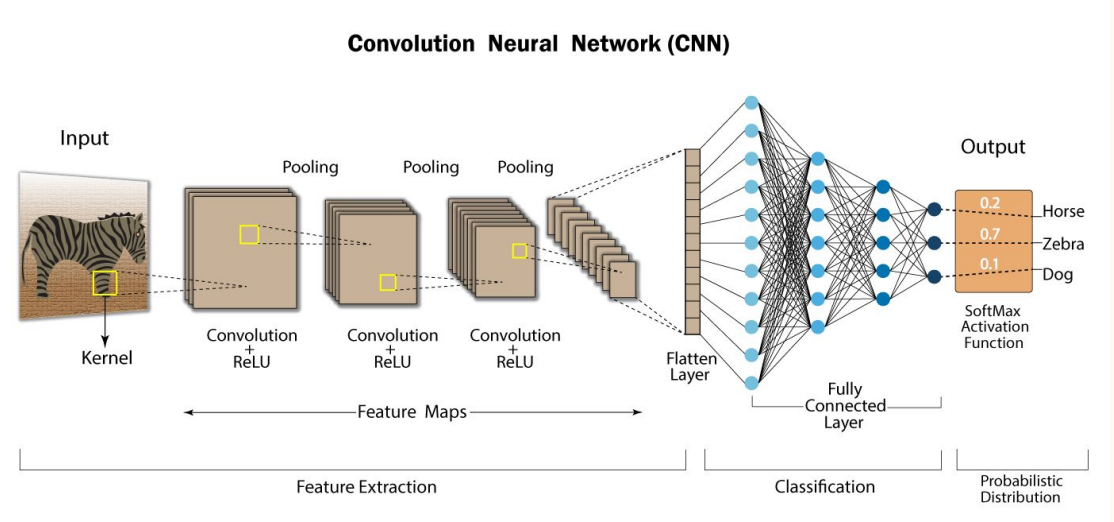


- Feature maps o **mapas de características**, también conocidos como **canales**, representan la cantidad de filtros aplicados en cada capa convolucional
- Cada filtro detecta **una característica o patrón específico** en los datos de entrada
- Aumentar el número de feature maps permite a la red aprender representaciones más complejas y abstractas
- El uso de **demasiados feature maps** puede provocar un **sobreajuste** y **mayores requisitos computacionales**
- La cantidad de **feature maps generalmente aumenta** a medida que **la red se vuelve más profunda**, lo que le permite aprender **características más sofisticadas**

https://poloclub.github.io/cnn-explainer/

Capas de una CNN

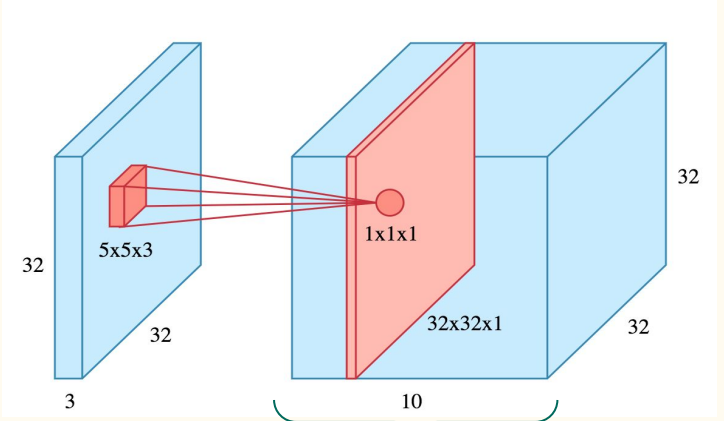
- C.Entrada → leer la imagen (no aprende parámetros)
- Número de capas → profundidad → complejidad del modelo
- C. Conv → Extracción de features
- C. Pool → Reducción de dimensionalidad
- Capa FC → Relación de feature



- Las **redes más profundas** tienen el potencial de aprender características y **representaciones más complejas**, pero también pueden ser más difíciles de entrenar, especialmente si el conjunto de datos no es lo suficientemente grande
- Las **redes menos profundas** pueden ser más adecuadas para **tareas más simples** o cuando los **recursos computacionales son limitados**
- La arquitectura de una CNN a menudo consiste en una secuencia de **capas convolucionales** seguidas de **capas de agrupación** (pooling) y luego **capas fully connected** para clasificación o regresión

¿Cómo aplicar convoluciones a imágenes de 3 canales?

- Los filtros tendrán 3 dimensiones, **la profundidad del filtro será la misma que la profundidad de la imagen** que estemos trabajando. (1er filtro)
  Si nuestra imagen es 32x32x3 un filtro está dado por KxKx3. Ej: 5x5x3. Se conserva el número de canales.
- En una capa convolucional, aplicamos **diferentes filtros** simultáneamente a la misma imagen. Luego los diferentes resultados se apilan (stacked) uno al lado del otro para obtener la salida de la capa convolucional.



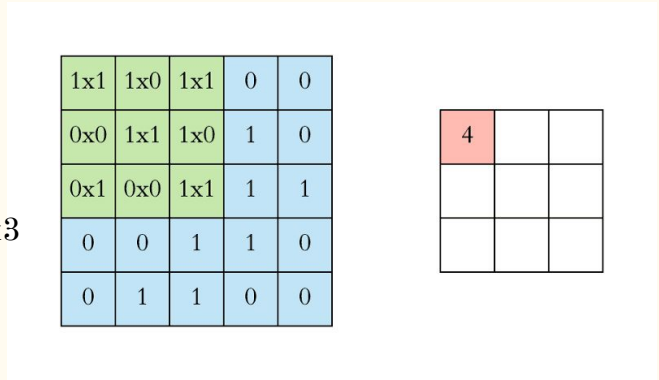
10 filtros aplicados

Stride

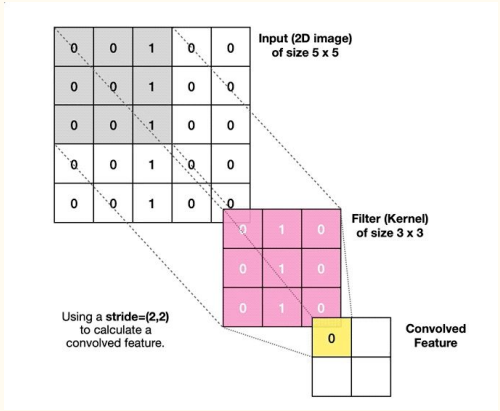
- El **stride** controla cuántos espacios nos movemos a través de la matriz de entrada, cada vez que aplicamos el filtro/kernel de convolución.
- Tiene impacto directo en la dimensión del feature map (output de la convolución).
- A más stride menos solapamiento en la convolución.

stride=1

image: 5x5
kernel: 3x3
feature map: 3x3



stride=2

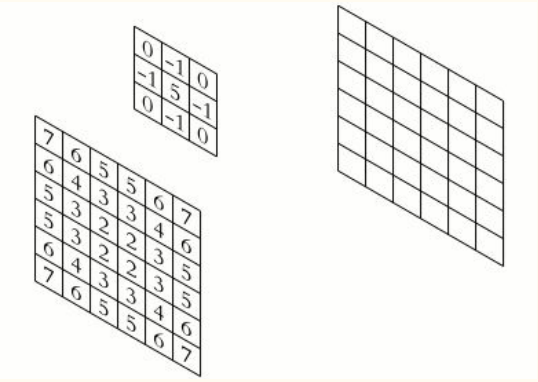


Imágenes: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

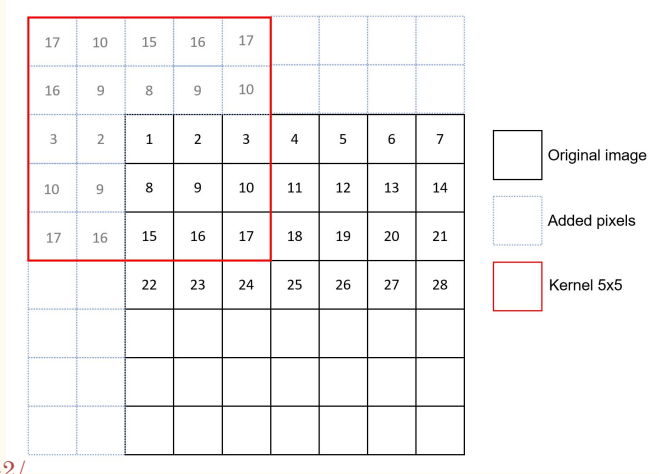
Padding

- Al aplicar convolución la dimensión del output disminuye, además perdemos bastante información de los ejes . Para que la salida de la convolución mantenga las dimensiones de la imagen, podemos usar el **padding**.
- El padding consiste en el aumento del tamaño de la matriz de entrada con valores adicionales en los bordes de esta.

padding=1

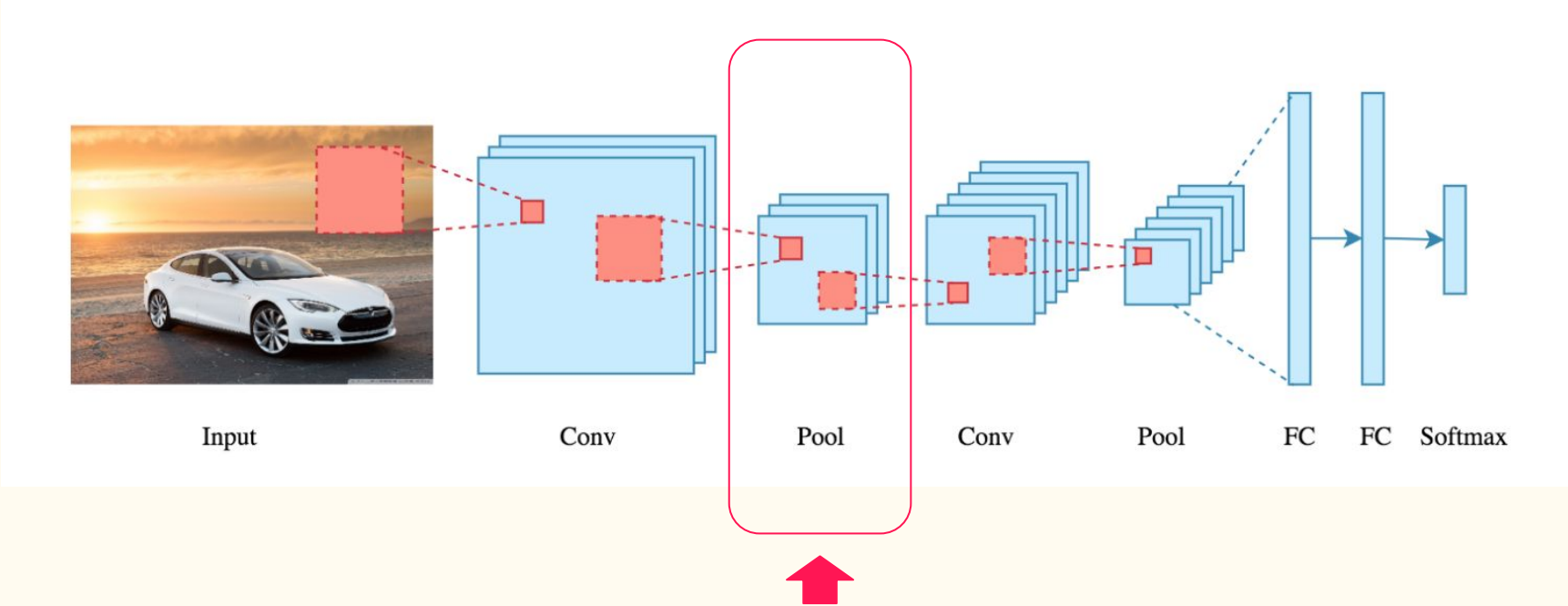


padding=2



Imágenes:
https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Una Arquitectura simple de CNN - Pool



Pooling Layers

- Estas capas permiten reducir el número de parámetros disminuyendo la dimensión de la salida de la capa convolucional.
- Estas capas no tienen parámetros entrenables (es decir, que no interfieren en el descenso del gradiente).

Pooling

stride = 2

filter = 2x2

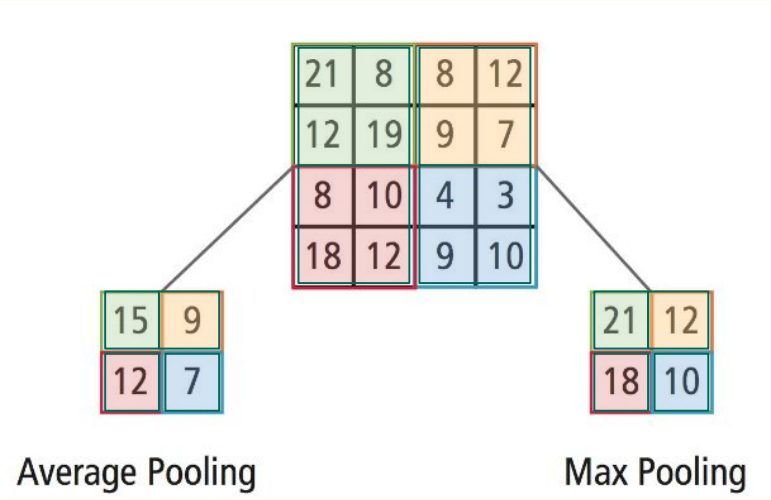


Imagen: https://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/

¿Cómo calcular la dimensión del output de conv?

Dados los siguientes inputs

- Entrada: nxn
- Padding: p
- Stride: s
- Filter: fxf

El output tendrá la siguiente dimensión:

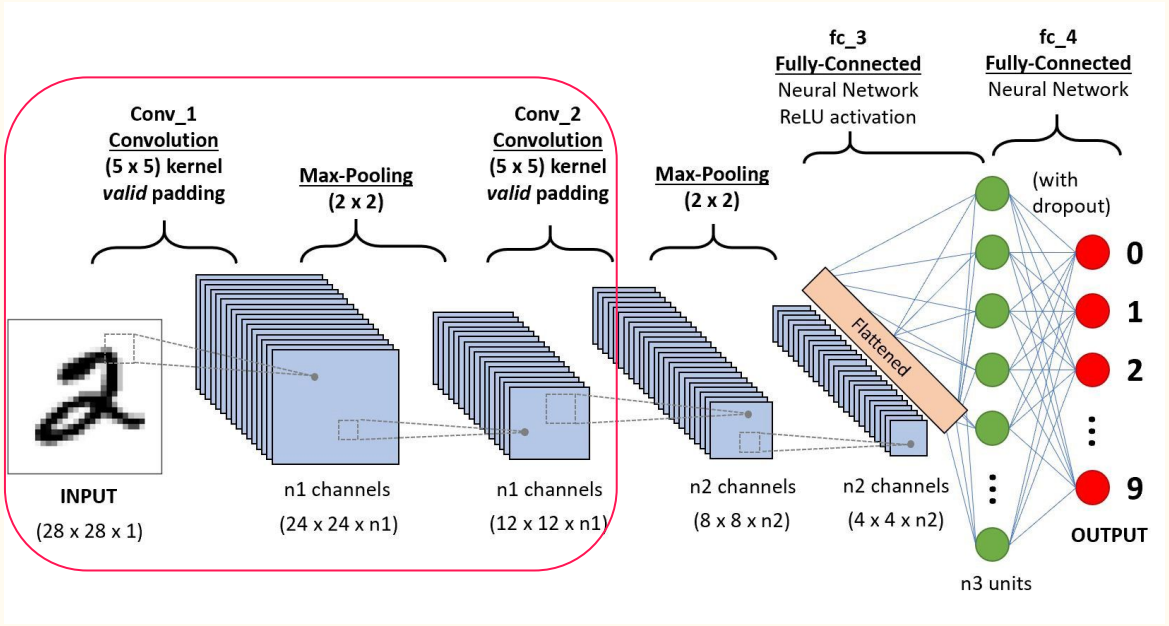
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

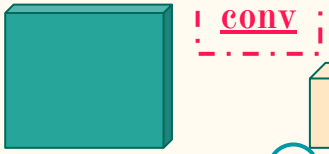
height width

Combinación f y p para conservar las dimensiones de entrada con s=1:

- f=3, p=1
- f=5, p=2

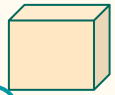
Resumiendo...





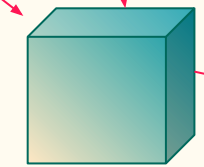
input
28 x 28 x **1**
h x w x c

conv

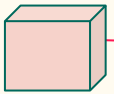


10 kernels
5 x 5 x **1**
s: 1 - p: 0

$$h_o = w_o = \frac{n_h + 2p_h - f_h}{s_h} + 1 = \frac{28 + 2 * 0 - 5}{1} + 1 = \underline{24}$$

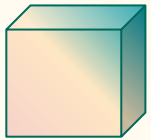


feature map
24 x 24 x **10**



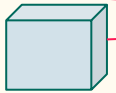
2 x 2 x **10**
s: 2 - p: 0

pool



feature map
12 x 12 x **10**

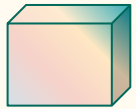
$$h_o = w_o = \frac{24 + 2 * 0 - 2}{2} + 1 = \underline{12}$$



20 kernels
5 x 5 x **10**
s: 1 - p: 0

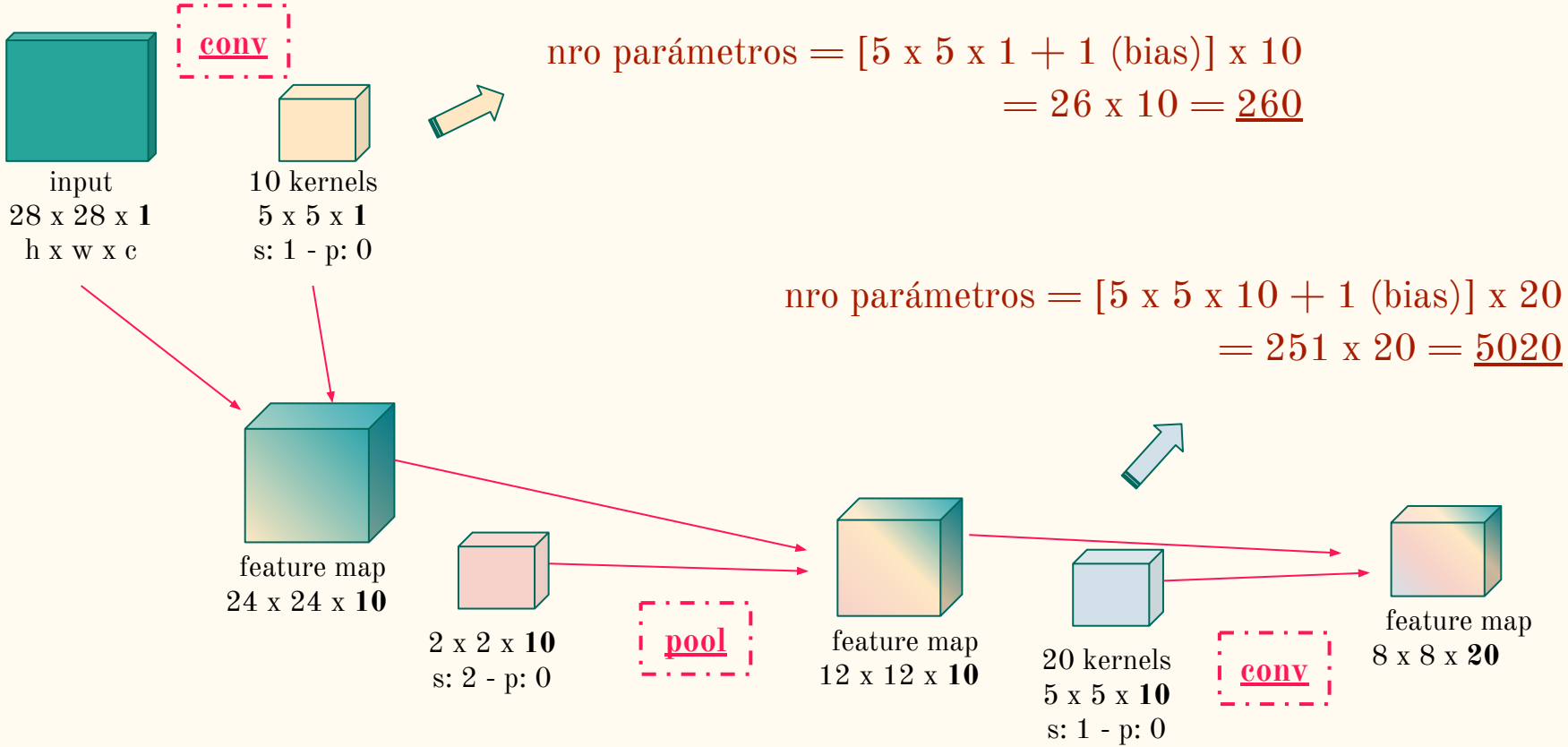
conv

$$h_o = w_o = \frac{12 + 2 * 0 - 5}{1} + 1 = \underline{8}$$

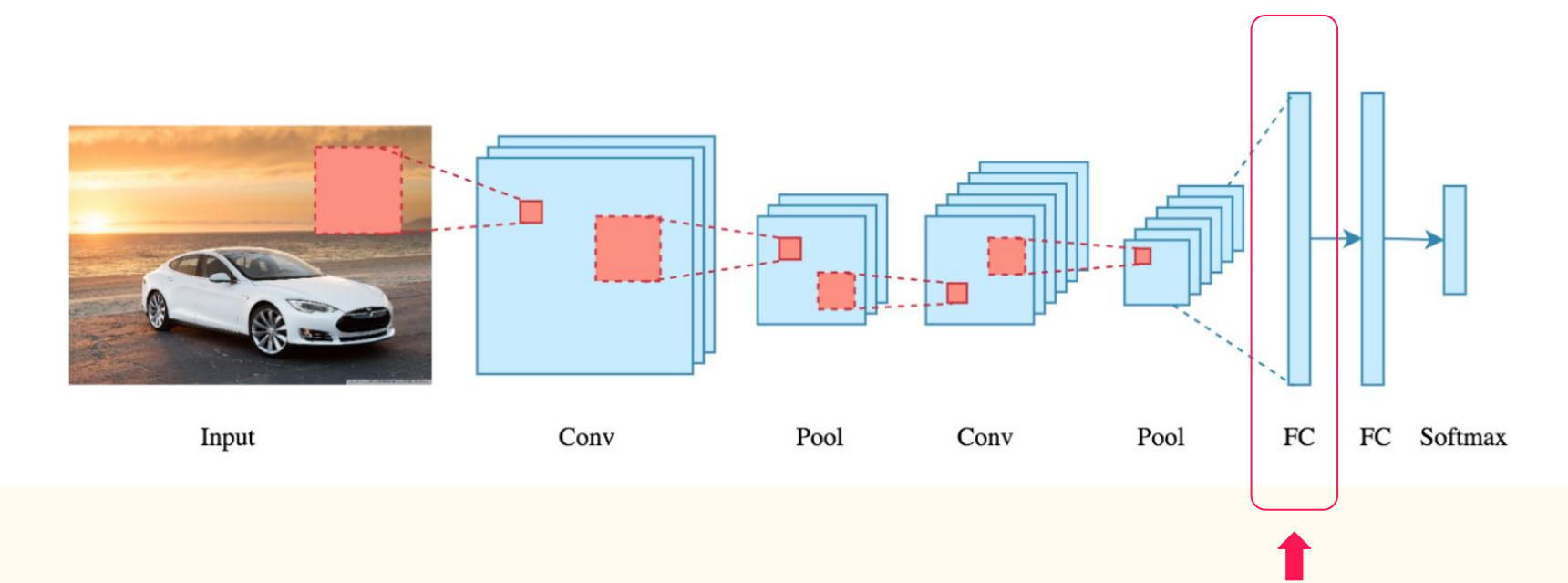


feature map
8 x 8 x **20**

Parámetros en una capa convolucional

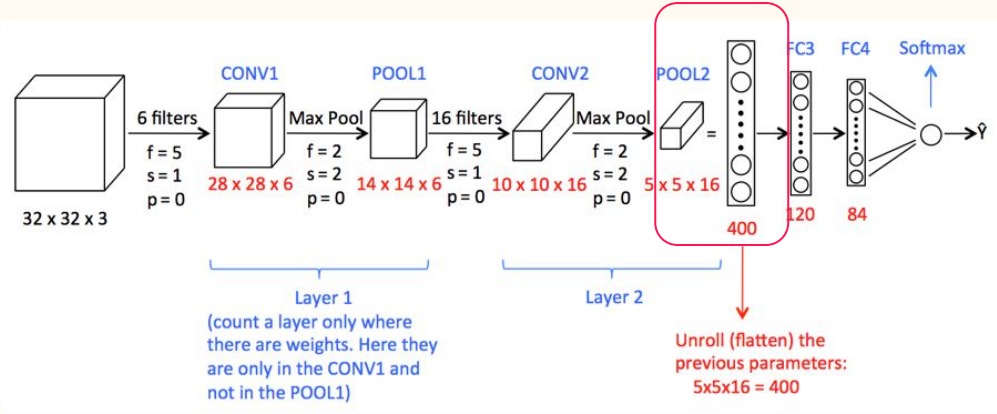


Una Arquitectura simple



Fully Connected Layer

- Luego de las capas **convolucionales** y de **pooling** se añaden un par de capas **fully connected** para completar la arquitectura de una red convolucional.



- **Flattening** es la operación de transformar un vector de 3 dimensiones en uno de 1 dimensión.
- La capa aplanada (flatten layer) se encuentra entre la CNN y la ANN, y su función es convertir la **salida de la CNN** en una entrada que la ANN pueda procesar.

nro parámetros = (número de inputs + 1) x número de outputs

Finalmente, en resumen ...

Las **redes neuronales convolucionales** se componen de varias capas:

1. Feature extraction (Convolutional Layer)
2. Reducción de dimensionalidad (Pooling Layer)
3. Aplanado de salida convolucional (Flatten Layer)
4. Clasificación (Fully Connected or Dense Layer)

LET'S CODE

6. CNN para clasificación de imágenes con Pytorch

Un poco de historia

- [1998] **LeNet - 5** - Gradient-based learning applied to document recognition - LeCun et al.
- [2012] **AlexNet** - ImageNet classification with deep convolutional neural networks - Krizhevsky et al.
- [2013] **Using 1x1 convolutions** - Network in network - Lin et al.
- [2014] **Inception network** - Going deeper with convolutions - Szegedy et al.
- [2015] **VGG - 16** - Very deep convolutional networks for large-scale image recognition - Simonyan & Zisserman
- [2015] **ResNet** - Deep residual networks for image recognition - He et al.
- [2017] **MobileNets** - Efficient Convolutional Neural Networks for Mobile Vision Applications - Howard et al.
- [2019] **MobileNets** - MobileNetV2: Inverted Residuals and Linear Botlenecks - Sandler et al.
- [2019] **EfficientNet**- EfficientNet:Rethinking Model Scaling for Convolutional Neural Networks - Tan and Le

Créditos

- Curso Stanford
- Tutorial towardsdatascience.com
- Convolutional Neural Networks - Andrew Ng
- Convolutional Neural Networks
- CNNs for Text Classification