

FASTBOOK 03

# **Computer vision: datos de uso y redes convolucionales**

Introducción al Deep Learning



## 03. Computer vision: datos de uso y redes convolucionales

Ya hemos analizado el proceso de aprendizaje para una red de deep learning. Hemos cubierto el preprocesamiento y el entrenamiento de una red neuronal, los optimizadores y algunas maneras de mejorar los resultados de una red de este tipo entrenada.

Pues seguimos ahondando más y, en este fastbook, vamos a centrarnos en una línea de trabajo específica dentro de deep learning, la denominada computer vision.

¿Y qué vamos a aprender? ¡Anota!

- Cómo se aplica la computer vision en el mundo real.
- Veremos algunos ejemplos de tareas típicas en computer visión.
- Los datos que usamos en computer vision.
- Las redes convolucionales.
- Las arquitecturas avanzadas (modelos preentrenados) para computer visión.
- Las técnicas para mejorar los resultados de un modelo de computer visión.

- ¿Qué es computer vision?
- ¿Qué tareas podemos cubrir en computer vision?
- Qué datos se usan en computer vision
- Redes convolucionales (CNN: convolutional neural networks)
- Arquitecturas avanzadas de computer vision
- Mejorar los resultados del modelo
- Resumen y recursos de interés

# ¿Qué es computer vision?



---

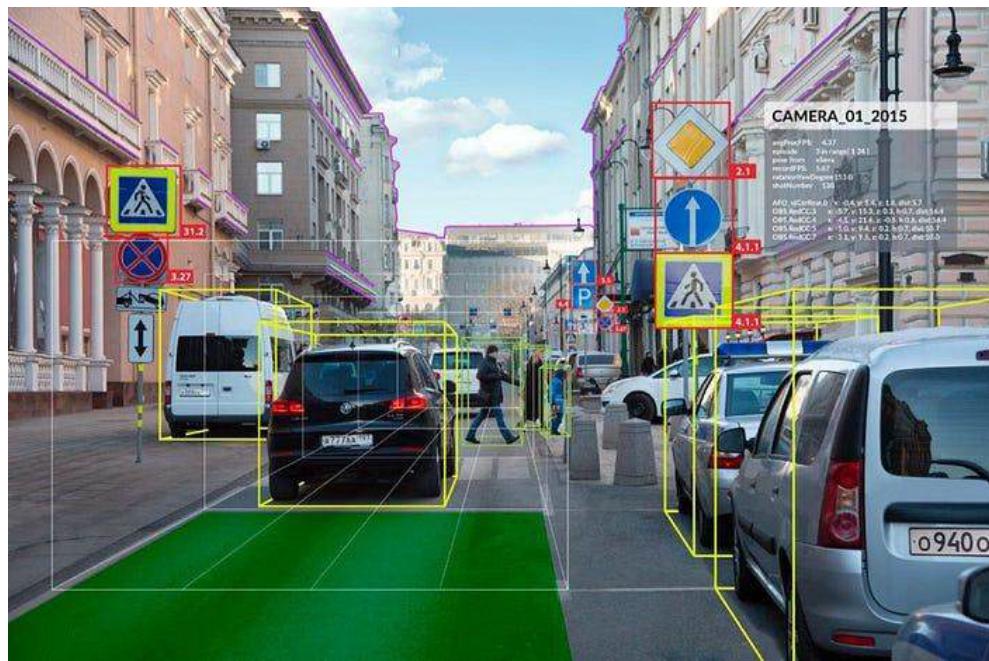
*Computer vision* es la rama de la IA que se ocupa de los datos visuales. La idea detrás de *computer vision* es replicar como los humanos ven y procesan datos visuales, por ejemplo, qué objetos se incluyen en una imagen o poder seguir un objeto o persona específica a lo largo de un vídeo.

---

*Computer vision* intenta automatizar el proceso de interpretación de datos visuales analizando y extrayendo información de datos como imágenes y vídeos.

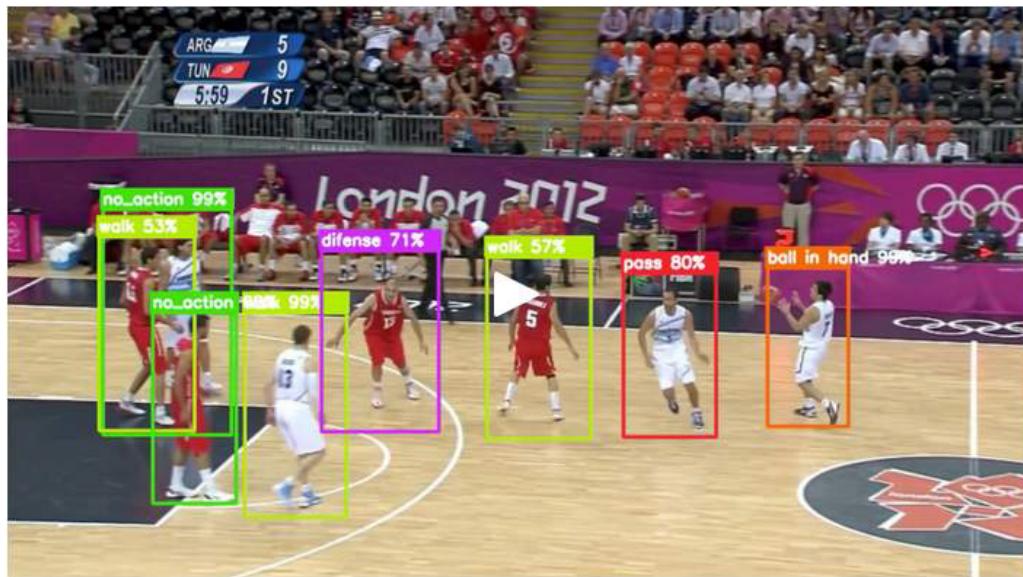
Podemos aplicar en muchos campos del mundo real la *computer vision*, por ejemplo...

- Los vehículos autónomos utilizan *computer vision* para entender los datos visuales de las cámaras del automóvil. Se utiliza para identificar personas, otros automóviles, semáforos, señales, marcadores de carreteras, bicicletas y otra información.



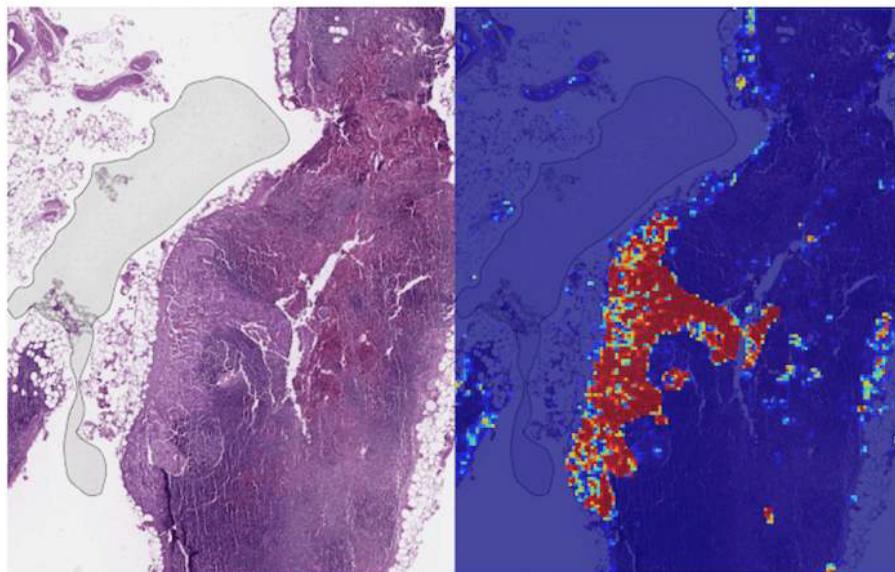
Fuente: <https://becominghuman.ai/computer-vision-applications-in-self-driving-cars-610561e14118>.

- En los deportes, *computer vision* se emplea para detectar y rastrear a los jugadores de equipos deportivos en vídeos, lo que puede ayudar a mejorar la estrategia del juego.



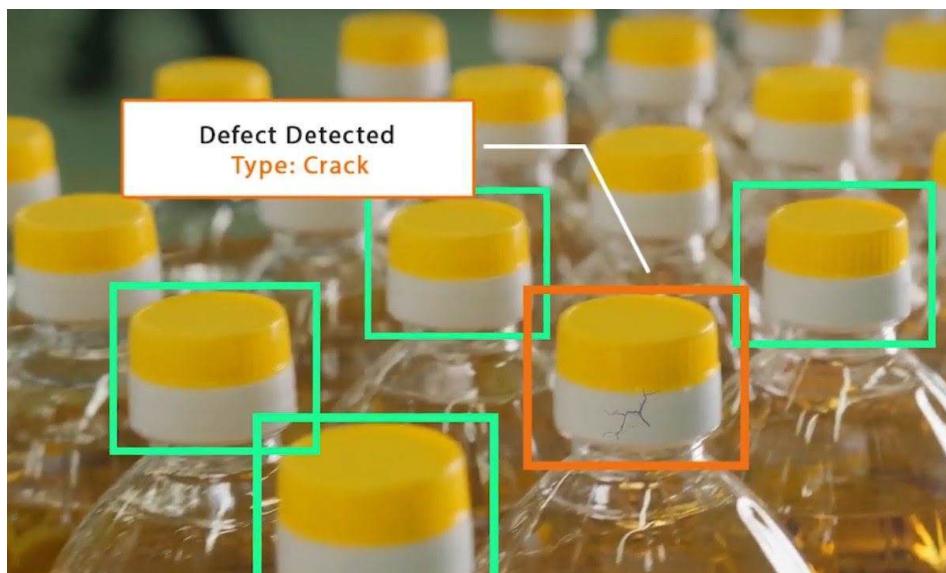
Fuente: <https://www.sportperformanceanalysis.com/article/computer-vision-in-sport>

- En el sector sanitario, *computer vision* se utiliza para detectar cáncer en imágenes médicas.



Fuente: <https://indatalabs.com/blog/how-does-computer-vision-work>.

- Los procesos de fabricación aplican la *computer vision* para detectar productos defectuosos durante el proceso de fabricación.



Fuente: <https://anscenter.com/ans-ai-products/ANS-Object-Detection-Studio>.

# ¿Qué tareas podemos cubrir en computer vision?



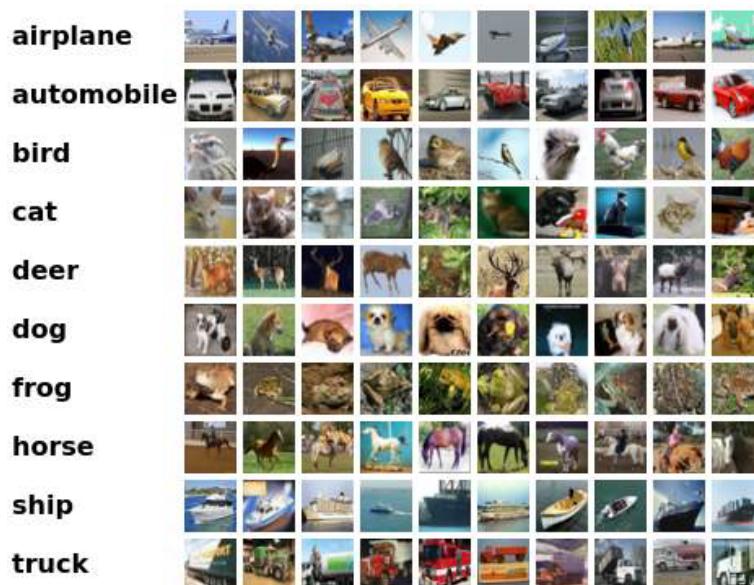
Qualentum Lab

---

La **clasificación de imágenes** es una de las tareas más comunes en *computer vision*.

Consiste en clasificar imágenes en función del contenido que contiene esta, por ejemplo, podemos clasificar imágenes según el tipo de animal que contiene: un gato, un perro, un pájaro...

La siguiente imagen muestra algunos ejemplos del conjunto de datos de clasificación de imágenes CIFAR-10, que contiene imágenes clasificadas en 10 clases diferentes.

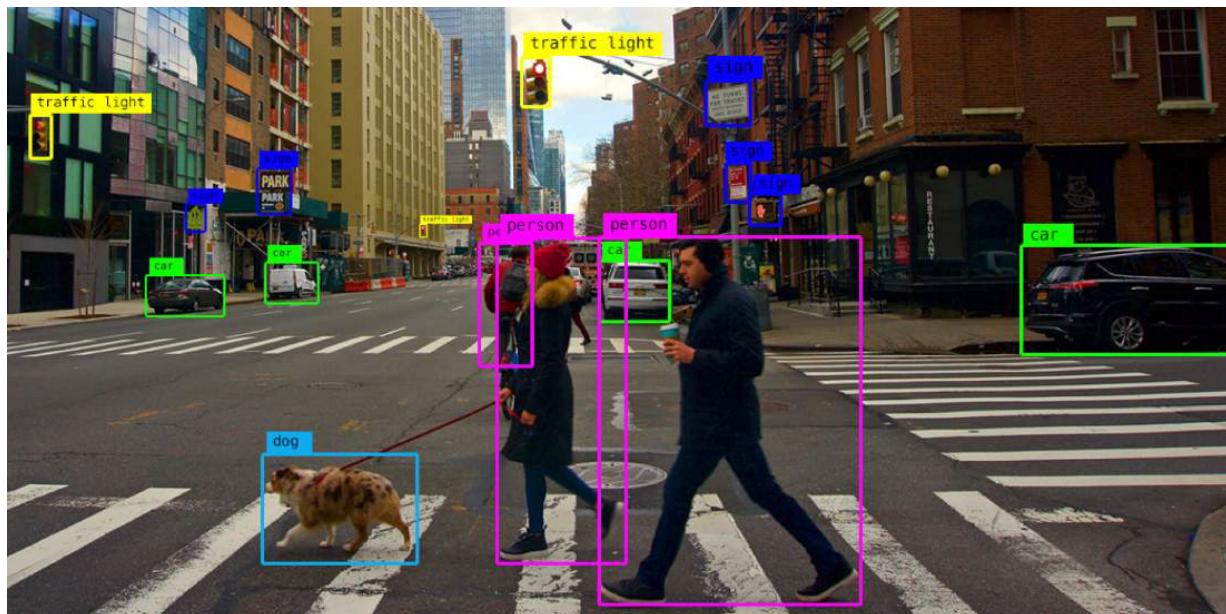


Fuente: <https://www.cs.toronto.edu/~kriz/cifar.html>.

---

La detección de objetos es otra tarea común en *computer vision*. Con esta técnica **detectamos los diferentes objetos que aparecen dentro de imágenes o vídeos**. Por ejemplo, podemos usar la detección de objetos para detectar todos los humanos y/o coches en una imagen. En la detección de objetos, el algoritmo identifica la ubicación de los objetos en la imagen y crea un cuadro alrededor de ellos, el llamado cuadro delimitador (*bounding box*).

En la imagen siguiente, puedes ver que el algoritmo ha detectado coches, un perro, personas, señales y semáforos. Para cada objeto detectado, hay un cuadro dibujado a su alrededor, que es el cuadro delimitador del objeto.



Fuente: Liz Oz, <https://alwaysai.co/blog/object-detection-for-businesses>.

---

En detección de objetos, normalmente se usa la **métrica IoU** (*intersection over union*). Así mismo, se puede usar IoU para calcular la **precisión promedia** (*average precision*).

**i** Si quieres ampliar información sobre IoU y *average precision* para la detección de objetos haz clic [aquí](#).

---

La segmentación de imágenes es una tarea de *computer vision* que segmenta una imagen en diferentes partes en función de los diferentes objetos dentro de la imagen.

Por lo tanto, nos brinda detalles mucho más finos sobre la forma de los objetos detectados que la detección de objetos.

---

**En la segmentación de imágenes, el algoritmo coloca una máscara sobre los objetos en lugar de un cuadro delimitador.**

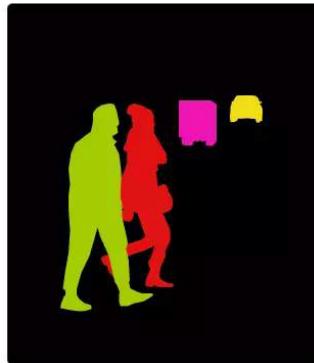
---

Hay tres tipos principales de segmentación: **semántica, instancia y panóptica**. A continuación, vemos un ejemplo de cada tipo diferente, basado en la misma imagen.

## Types of Image Segmentation



SEMANTIC IMAGE  
SEGMENTATION



INSTANCE  
SEGMENTATION



PANOPTIC  
SEGMENTATION

Fuente: <https://mindy-support.com/news-post/what-is-image-segmentation-the-basics-and-key-techniques/>.

Como se puede observar cada objeto diferente detectado tiene una máscara de un color específico para identificarlo.



En segmentación de imágenes se usa IoU y Dice coefficients como métricas para medir la calidad de la segmentación. Amplía información clicando [aquí](#).

# Qué datos se usan en computer vision



Cómo ya sabemos, los datos utilizados en *computer vision* suelen ser imágenes o vídeos. Las imágenes pueden **estar en escala de grises o en color**, representadas como una cuadrícula de valores de píxeles.

Fíjate en el ejemplo que te compartimos en la siguiente figura.



Fuente: Himanshi Singh,  
<https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>.

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	249	255	240	255	129	0	5	0
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Fuente: Himanshi Singh,

<https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>.

---

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	249	255	240	255	129	0	5	0
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Fuente: Himanshi Singh,

<https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>.

---

Este ejemplo es una imagen en blanco y negro del número 8. La imagen se compone de un montón de pequeños cuadros. Estos cuadros son los píxeles de la imagen.

---

**Recuerda que, en la mayoría de las imágenes, los píxeles serán tan pequeños que no se podrán distinguir con solo mirar la imagen.**

La **resolución** de la imagen indica cuántos píxeles tiene la imagen. Para una imagen con resolución 512 x 512, significa que hay 512 píxeles a lo ancho de la imagen y 512 píxeles a lo largo de la altura de la imagen, de modo que sería un total de 262.144 píxeles. En el ejemplo más simple de la imagen del número 8, la resolución es 16 x 24, porque hay 16 filas de píxeles y 24 columnas de píxeles.

1

## Imágenes en escala gris

A nivel máquina, estos píxeles se interpretan como números. Para una imagen en escala de grises, los números van de 0 (negro) a 255 (blanco). Estos valores corresponden a la intensidad de cada píxel. Así, estos píxeles se convierten en una matriz de números como en la imagen de la derecha, en el ejemplo del número 8 que acabamos de ver.

En nuestro caso, sería una matriz de tamaño 24 x 16 (24 filas y 16 columnas) de números entre 0 y 255. Esta matriz de valores de píxeles se llama un canal. Recuerda que en las imágenes en escala de grises solo hay un **canal**.

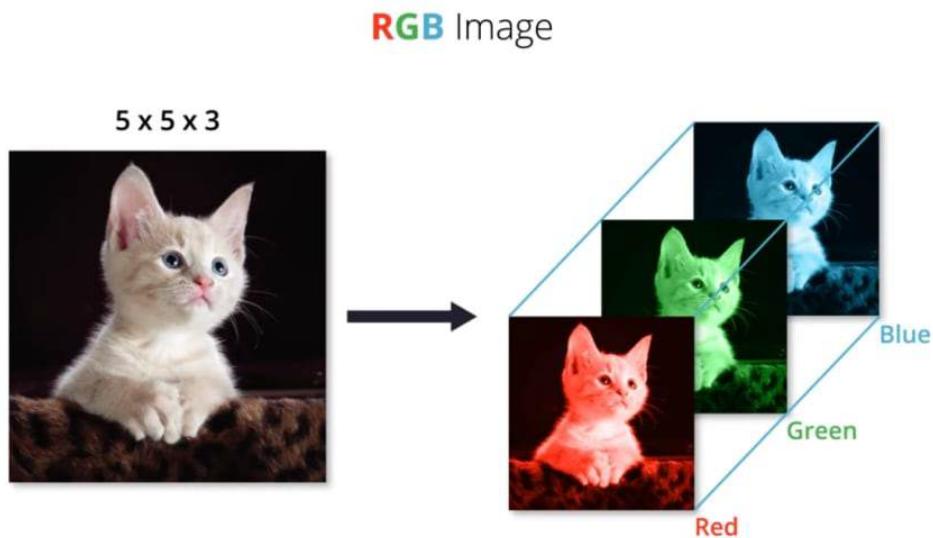
## 2

## Imágenes en color

Las imágenes cromáticas contienen muchos colores diferentes. Sin embargo, la mayoría de los colores se pueden crear a partir de una combinación de tres colores: rojo, verde y azul.

Ante esto, una imagen en color se compone de tres canales, **un canal rojo (R)**, **un canal verde (G)** y **un canal azul (B)**. Cada uno de estos canales tiene píxeles con valores entre 0 y 255, que representan la intensidad del píxel, y son tonos de rojo, verde y azul. Pues bien, estas tres matrices se combinan para crear la imagen a todo color.

Para las imágenes en color, la matriz numérica que representa la imagen tendría el tamaño de **H (número de filas de píxeles) x W (número de columnas de píxeles) x 3 (matrices RGB)**. Por ejemplo, una imagen en color con una resolución de 256 x 256 tendría una matriz numérica de 256 x 256 x 3 para representarla.



Fuente: Sandeep Balachandran,  
<https://dev.to/sandeepbalachandran/machine-learning-going-further-with-cnn-part-2-41km>.

 Aquí te comarto algunas librerías comunes para leer y procesar imágenes en Python (haz clic sobre cada una para acceder a la información: [Pillow](#), [Scikit-Image](#) y [OpenCV](#)).

## 3

## Vídeos

Hemos visto cómo un ordenador representa imágenes, pero ¿cómo representa un vídeo?

Para que una red neuronal pueda procesar un vídeo, primero debe dividirlo en imágenes. Los vídeos se pueden dividir en muchas imágenes secuenciales llamadas **fotogramas** (*frames* en inglés).

---

La frecuencia de muestreo es la velocidad a la que se extraen las imágenes del vídeo.

Esta velocidad generalmente se expresa como fotogramas por segundo o **FPS** (*frames per second* en inglés), que especifica la cantidad de fotogramas que se extraerán de un vídeo cada segundo.

Entonces, para un vídeo de 1 minuto, si la frecuencia de muestreo es 30 FPS, entonces se extraerían 1800 fotogramas ( $30 \times 60$  segundos). **Cuanto mayor sea la frecuencia de muestreo**, más imágenes se extraerán del vídeo y cuanto menor, menos imágenes se extraerán.

Una vez que el vídeo **se ha convertido en una secuencia de fotogramas**, estos fotogramas se pueden procesar como cualquier otra imagen.

## Preprocesamiento de los datos

En general, se requiere muy poco preprocesamiento de datos en *computer vision* para entrenar con un modelo de *deep learning*, de hecho, se reduce comúnmente a estos dos pasos: el cambio de tamaño y el escalado.

### Cambio de tamaño

Si las imágenes tienen diferentes tamaños (por ejemplo, 256 x 256 y 512 x 512) o si el modelo que estás utilizando requiere un tamaño de imagen específico y tus datos no son de ese tamaño, será necesario cambiar el tamaño de las imágenes para que todas tengan el mismo tamaño y/o para que sean del tamaño correcto para el modelo.

Para realizar el cambio de tamaño puedes usar una de las librerías mencionadas anteriormente o puedes usar el transform [Resize](#) en PyTorch.

### Escalado de imagen

Como sabes, las redes neuronales necesitan que los datos estén en la misma escala, por lo que los valores de píxeles de la imagen deben rescalarse para que estén entre 0 y 1 o -1. Una forma sencilla de cambiar la escala de los datos de la imagen, dado que los píxeles siempre están en el rango de 0 a 255, es dividir todos los píxeles entre 255. Esto te dará valores de píxeles en un rango de 0 a 1.

Cuando necesites que las imágenes tengan una escala específica puedes usar la transformación en PyTorch: dispones de más información en este enlace a [Normalization](#).

# Redes convolucionales (CNN: convolutional neural networks)



Las redes neuronales convolucionales (CNN) son redes neuronales que se utilizan para procesar datos en forma de cuadrícula. En el caso de *computer vision*, estos datos en forma de cuadrícula se refieren a una imagen que, como vimos anteriormente, se compone de una cuadrícula de valores de píxeles.

---

**¿Sabías que...? Las redes convolucionales se llaman CNN porque, a diferencia del perceptrón multicapa, en lugar de usar la multiplicación de matrices, usan la operación matemática convolución.**

Y quizás te estés preguntando...

## ¿Por qué utilizamos una CNN para imágenes en lugar del perceptrón multicapa?

Las CNN son **capaces de reducir imágenes a una forma** que pueda procesarse más fácilmente sin perder información importante (por ejemplo, información necesaria para hacer una buena predicción).

Debido a la forma en que están diseñadas las redes convolucionales, la cantidad de memoria que necesitan para procesar una imagen es mucho menor que la de una red *feed forward*. Además, las redes convolucionales son capaces de detectar automáticamente características y patrones en imágenes, como bordes, ojos, rostros...

1

### Los componentes principales de las redes convolucionales

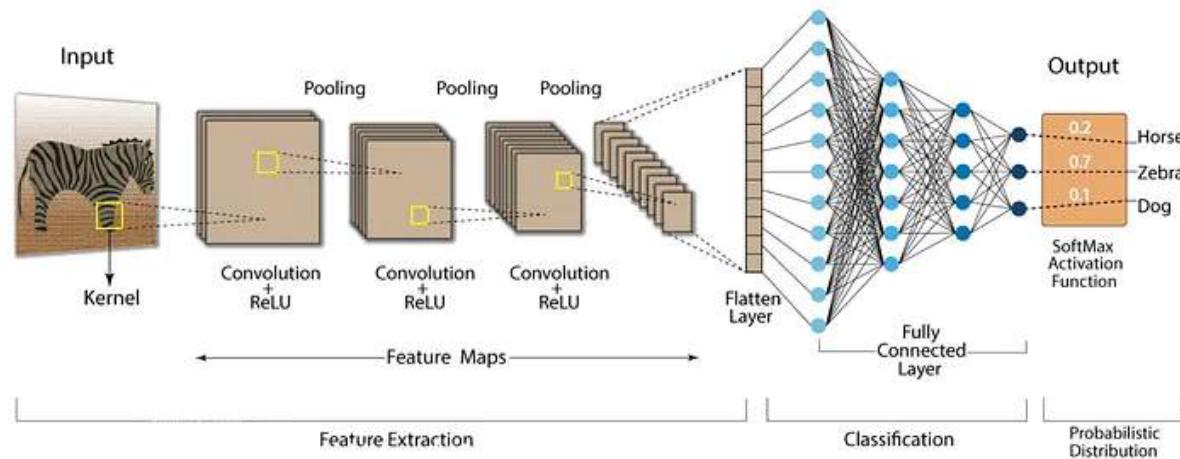
Las CNN suelen **constar de tres componentes**: una capa convolucional (*convolutional layer*), una función de activación no lineal (por ejemplo, ReLU, el inglés *non-linear activation function*) y una capa de pooling (*pooling layer*).

---

La CNN puede estar compuesta por varias capas convolucionales y/o de agrupación.

También puede haber una o más capas totalmente conectadas (*fully connected layers*) al final de la red, por ejemplo, en el caso de los problemas de clasificación, tener la salida como **probabilidades de pertenecer a una clase particular**. En la siguiente imagen podemos ver un ejemplo de una CNN para un problema de clasificación de imágenes.

## Convolution Neural Network (CNN)



Fuente: <https://developersbreach.com/convolution-neural-network-deep-learning/>.

## 2

### Capa convolucional

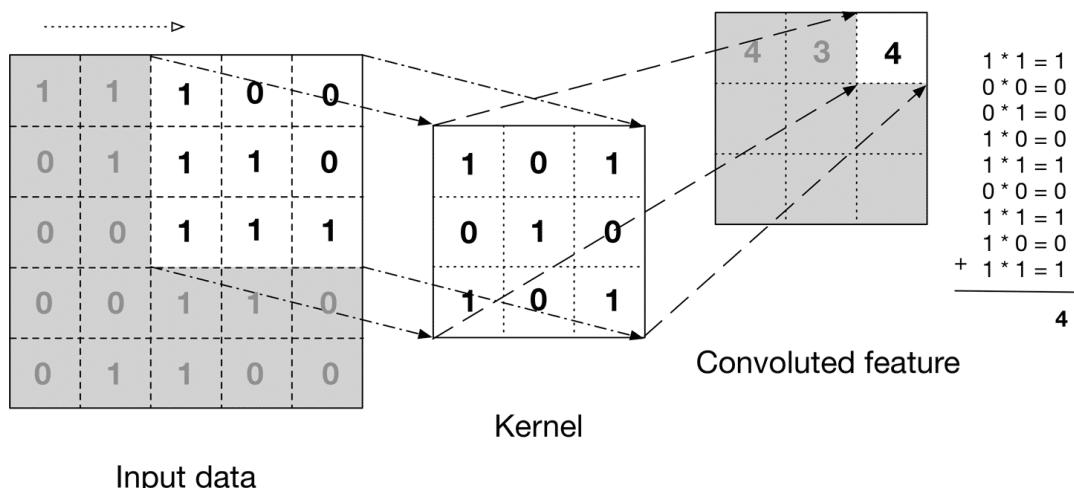
- La capa convolucional es la base de la CNN y requiere datos de entrada y un núcleo (*kernel*) o filtro (*filter*) y produce una salida llamada **mapa de características** (*feature map*). Los datos de entrada en el caso de imágenes son los datos de imagen para la primera capa convolucional de la red y para las siguientes capas convolucionales, es el mapa de características o la salida de la capa convolucional anterior.
- El **filtro** (*filter*) es una matriz multidimensional y se utiliza para extraer características de las imágenes. El filtro se compone de **núcleos** (*kernels*), que son matrices 2D de pesos cuyos valores se actualizan durante el proceso de entrenamiento mediante retropropagación, utilizando un optimizador para encontrar los mejores valores para actualizarlos. El filtro suele ser mucho más pequeño que los datos de entrada, en términos de alto y ancho, por ejemplo, 3 x 3, pero debe tener el mismo número de canales que los datos de entrada.

**Supongamos que los datos de entrada son una imagen en color de tamaño 256 x 256 x 3 (3 canales para RGB), entonces el filtro podría tener un tamaño de 5 x 5 x 3. De esta manera, el filtro estaría formado por tres 5 x 5 núcleos.**

Durante el pase hacia adelante, el filtro se aplica a los datos de entrada y, puesto que suele ser más pequeño que estos, para aplicar el filtro a todos esos datos se desliza a lo largo y ancho de dichos datos.



El **producto escalar** se calcula entre el filtro y una región de los datos de entrada, que es del tamaño del filtro, y el resultado de este producto escalar pasa a formar parte del mapa de características de salida (como se aprecia en la figura siguiente).

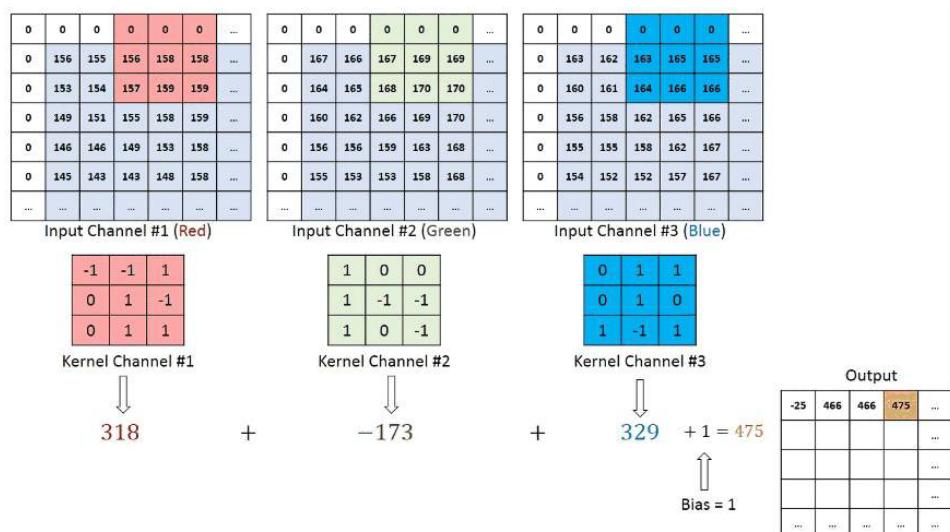


Fuente: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>.

Después de calcular el producto escalar para una región particular de los datos de entrada, **el filtro cambia para aplicar el producto escalar a la siguiente región**. En cuanto a los pesos en el filtro, estos siguen siendo los mismos para todas las regiones de los datos de entrada. ¿Qué significa? Pues bien, que estos parámetros de peso se comparten en todos los datos de entrada.

La imagen que acabamos de ver muestra un ejemplo de cómo funciona la capa convolucional para una imagen en escala de grises. Si nos fijamos, el filtro comienza en la esquina superior izquierda, con la primera región de  $3 \times 3$  de los datos de entrada, y calcula el producto escalar entre esa región y el filtro. El valor obtenido se corresponde con el cuadro en la esquina superior izquierda de la matriz de salida. Luego, el filtro pasa a la siguiente región de  $3 \times 3$ , calcula el producto escalar con los datos de entrada y el valor resultante va al siguiente cuadro de la matriz de salida. Este proceso se repite hasta que el filtro se haya aplicado a todos los datos de entrada.

En el caso de datos de entrada con más de un canal, por ejemplo, **una imagen en color con 3 canales**, el filtro aplica el producto escalar a todos los canales y suma las salidas, lo que da como resultado un valor único para todos los canales en el mapa de características de salida. Podemos ver este proceso en la figura siguiente.



Fuente: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>.

## 3

## Los hiperparámetros para las capas convolucionales

Ahora pasemos a conocer los hiperparámetros más relevantes con respecto a la capa convolucional.

### ***El número de filtros***

Las capas convolucionales suelen incluir varios filtros. Una capa convolucional típica podría incluir entre 32 y 512 filtros, pero pueden ser de otros tamaños. Estos múltiples filtros permiten a la CNN aprender patrones específicos en los datos de entrenamiento de muchas maneras diferentes.

### ***El tamaño del núcleo***

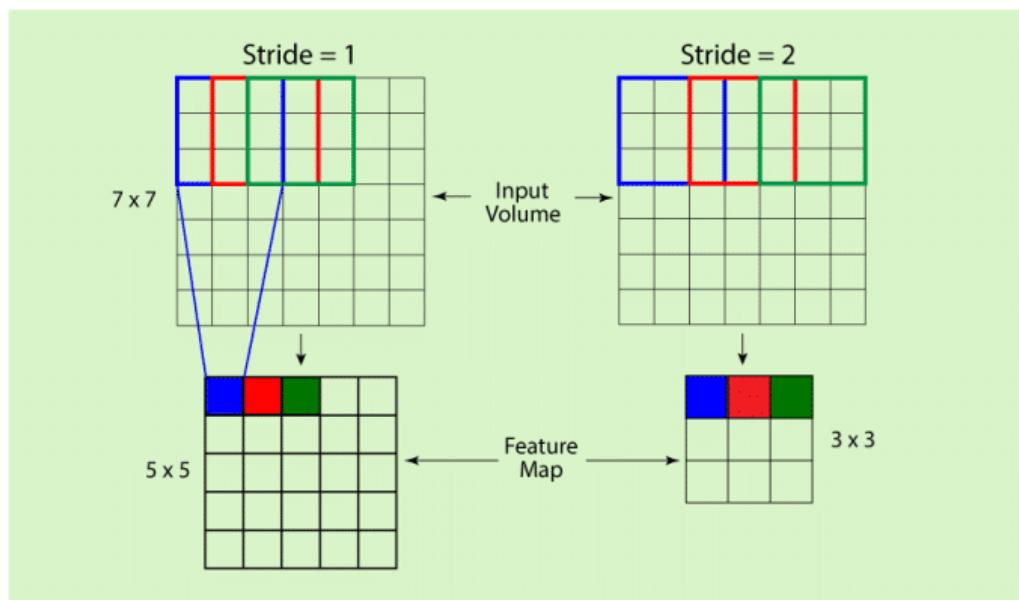
Se debe especificar el tamaño de los núcleos en cada filtro a aplicar para las capas convolucionales. El tamaño del núcleo tiende a ser bastante pequeño y mucho más pequeño que el tamaño de los datos de entrada. Recuerda: los valores típicos para los núcleos son 3 x 3, 5 x 5 y 7 x 7.

### El paso (stride)

En el ejemplo anterior, el núcleo se movió píxel a píxel a través de los datos de entrada para poder aplicarlo a todas las regiones de los datos de entrada. Sin embargo, el núcleo no siempre tiene que desplazarse un píxel, **también puede desplazarse 2 o 3 píxeles a la vez.**

Pues bien, el paso es el hiperparámetro que especifica el número de píxeles para desplazar el núcleo cada vez que se aplica el producto escalar. Y se aplica tanto al movimiento horizontal como al movimiento vertical del núcleo. Los valores del paso típicos son uno, tanto para el movimiento en altura como en ancho. Sin embargo, en casos específicos puede ser deseable un paso de dos o más.

Fuente: <https://developersbreach.com/convolution-neural-network-deep-learning/>.



## Relleno (padding)

Para el ejemplo de la imagen en escala de grises anterior, es posible que hayas notado que, con un paso de uno, el núcleo se mueve sobre un píxel cada vez que aplica el producto escalar. Mientras que al píxel en la esquina superior izquierda se le aplicó el núcleo solo una vez, al píxel en el medio se le aplicó el *kernel* tres veces. También habrás notado que el mapa de características de salida es más pequeño que los datos de entrada. Esto puede significar una pérdida de información en los bordes de los datos de entrada.

Si queremos evitar perder información en los bordes, se puede aplicar relleno a la imagen. El relleno agrega píxeles adicionales alrededor de los bordes de los datos de entrada. En el relleno, que se llama *same padding*, el mapa de características de salida tiene el mismo tamaño que los datos de entrada y no hay pérdida de información (es decir, los filtros se aplican la misma cantidad de veces a todos los píxeles).

El valor de relleno más común es el relleno con ceros (*zero padding*) que agrega valores de píxeles de cero, alrededor de los bordes de los datos de entrada.

Fuente: <https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243>

0	0	0	0	0	0	0
0	2	4	9	1	4	0
0	2	1	4	4	6	0
0	1	1	2	9	2	0
0	7	3	5	1	3	0
0	2	3	4	8	5	0
0	0	0	0	0	0	0

X

1	2	3
-4	7	4
2	-5	1

=

21	59	37	-19	2
30	51	66	20	43
-14	31	49	101	-19
59	15	53	-2	21
49	57	64	76	10

Image
Filter / Kernel
Feature

## ¿Cómo afecta la aplicación del filtro, paso y relleno el tamaño de la salida?

Calculamos la salida de la siguiente manera:

Salida = [(entrada - tamaño del núcleo + 2\*relleno)/paso + 1] x [entrada - tamaño del núcleo + 2\*relleno]/paso + 1] x número de filtros

Si aplicamos una capa convolucional con los siguientes parámetros sobre una imagen del tamaño 32 x 32 x 3, tendríamos los siguiente:

Núcleo: 3 x 3

Paso: 1

Relleno = 1

Número de filtros = 16

$$[(32 - 3 + 2*1)/1 + 1] \times [(32 - 3 + 2*1)/1 + 1] \times 16 = 32 \times 32 \times 16$$

---

**Antes de finalizar este apartado y para completar tu aprendizaje, te recomendamos que eches un vistazo a la siguiente documentación de PyTorch: la librería [Torchvision](#) de PyTorch que tiene varios recursos para problemas de computer vision y [capas convolucionales](#).**

4

### Función de activación no lineal

Después de que los datos de entrada pasan a través de la capa convolucional, se les aplica una función de activación no lineal. Una de las funciones de activación más comunes que se aplican en las CNN es la función de activación lineal rectificada (ReLU).

---

**La función de activación no lineal se utiliza para incluir la no linealidad en el mapa de características de salida.**

---

 Puedes leer sobre la función de activación ReLU en PyTorch [aquí](#).

## 5

### **Capa de *pooling***

Después de aplicar la capa convolucional y la función de activación no lineal a los datos de entrada, es común agregar una capa de *pooling*. Una capa de *pooling* reduce el tamaño del mapa de características de salida reemplazando los valores de las salidas cercanas con estadísticas resumidas.

El proceso para **aplicar una capa de *pooling* es similar al de la capa convolucional**. Se aplica un filtro a regiones pequeñas en todo el mapa de características de salida. Una diferencia importante es que este filtro no se compone de pesos que se puedan aprender.

---

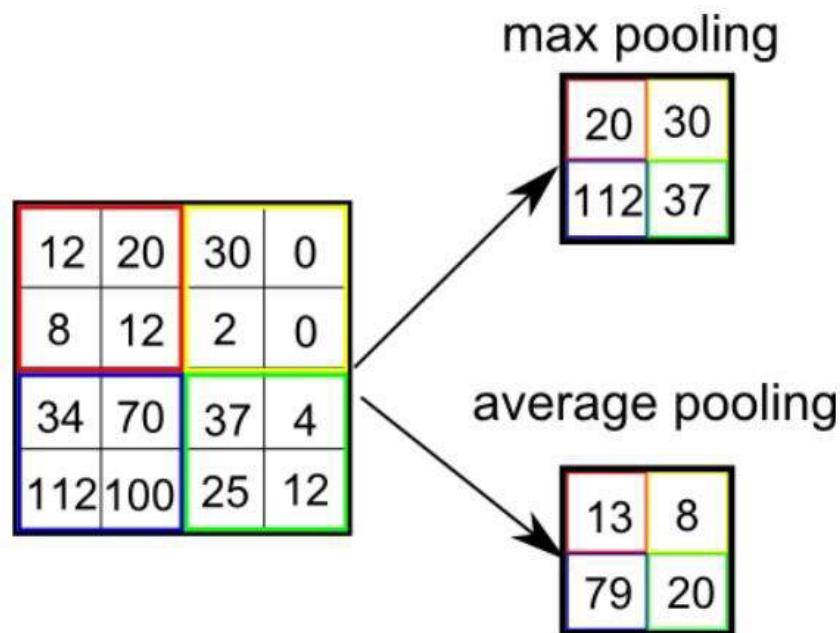
**Aplica una función agregada a cada región de los datos de entrada. Dos de los tipos de *pooling* más comunes son la *pooling máxima (max pooling)* y la *pooling promedio (average pooling)*.**

**MAX POOLING****AVERAGE POOLING**

En cada región, se selecciona el valor máximo en esa región para incluirlo en la salida.

**MAX POOLING****AVERAGE POOLING**

En cada región, el promedio de los valores en esa región se calcula para incluirlo en la salida.



Fuente: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>.

**i** Para ampliar información, consulta la documentación de PyTorch sobre las capas de *pooling* [aquí](#).

## 6

### Capa totalmente conectada (*fully connected layer*)

Se pueden agregar una o más capas totalmente conectadas al final después de todas las capas convolucionales y de *pooling*. En problemas de clasificación, esta capa se utiliza para predecir la probabilidad de que una imagen concreta corresponda a una clase particular. En problemas de regresión, esta capa predice una salida numérica.

## 7

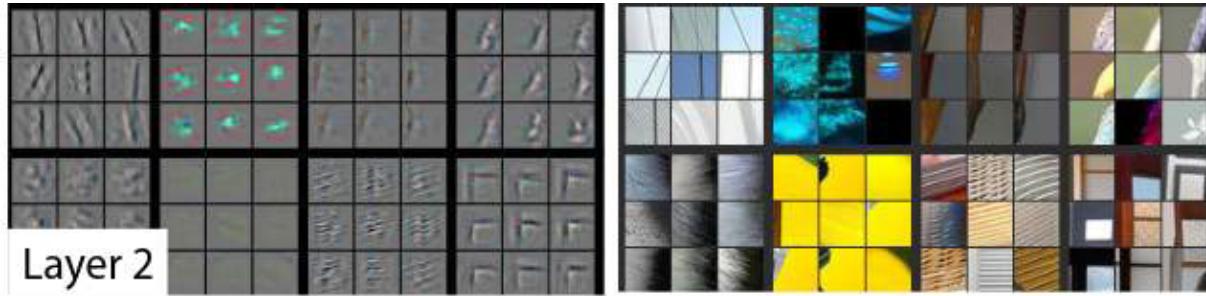
### Las fases del aprendizaje a través de las capas

Las diferentes capas de una red convolucional con múltiples capas convolucionales extraen diferentes características. A continuación, comparto un ejemplo de las salidas de 5 capas diferentes de una red convolucional que se entrenó con el dataset *ImageNet* 2012.

La primera capa, normalmente, extrae **características de bajo nivel**, como líneas y bordes rectos.



Este mapa de características luego, se pasa a la siguiente capa, que **aprende características de nivel** ligeramente superior, como las esquinas.



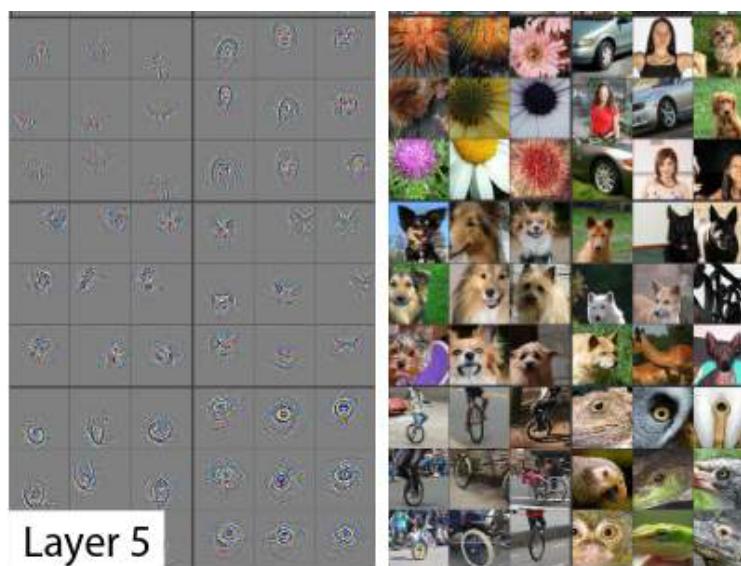
La tercera capa es capaz de **extraer patrones** más complejos.



La cuarta capa es capaz de **extraer información específica de la clase**, como las caras de los perros, y la quinta capa es capaz de extraer objetos completos, como un rostro humano, un ojo o ruedas.



Fuente: <https://arxiv.org/pdf/1311.2901.pdf>.



Fuente: <https://arxiv.org/pdf/1311.2901.pdf>.

 **¡Importante!** Consulta [PyTorch tutorial](#) para el entrenamiento de un clasificador de imágenes.

# Arquitecturas avanzadas de computer vision



Qualentum Lab

En el anterior apartado, hemos descubierto cuáles son los componentes básicos de las redes convolucionales. Sin embargo, existen muchas arquitecturas diferentes más especializadas que se pueden utilizar con diferentes tipos de problemas.

Muchas de estas redes especializadas han sido previamente entrenadas en grandes conjuntos de datos, de modo que pueden ajustarse a problemas específicos y requieren mucho menos entrenamiento y **muchas veces dan mejores resultados que entrenar un modelo directamente desde cero**. Veamos algunas de estas arquitecturas más complejas.

## AlexNet y VGG

[AlexNet](#) y [VGG](#) utilizan la arquitectura CNN tradicional. Fueron introducidos para problemas de clasificación a gran escala de hasta 1000 clases. AlexNet tiene 8 capas (5 convolucionales y 3 pooling). VGG es mucho más grande y tiene diferentes variantes con diferente número de capas, por ejemplo, VGG16 (16 capas) y VGG19 (19 capas).

## Las redes residuales

Las [redes residuales \(ResNets o residual networks\)](#) intentan resolver el problema del gradiente que desaparece (*vanishing gradient*) en grandes CNN. Cuando las redes neuronales son muy grandes, el gradiente puede tender a volverse extremadamente pequeño o cero, cuando llega a las primeras capas de la red, lo que hace que la red aprenda extremadamente lentamente o no aprenda nada. Las ResNets utilizan skip connections que permiten omitir algunas capas y alimentar la salida de una capa directamente como entrada a una capa que no es la siguiente. Estas skip connections ayudan a resolver el problema del gradiente que desaparece.

## Inception Networks

[Inception Networks](#) utilizan múltiples capas convolucionales en paralelo con diferentes tamaños de filtro y combinan su salida en una sola capa. Esto contrasta con las arquitecturas típicas de CNN donde las capas convolucionales se apilan una tras otra (por ejemplo, VGG). Cada una de estas capas convolucionales paralelas tiene diferentes tamaños de núcleo y puede aprender múltiples representaciones de características al mismo tiempo. Por ejemplo, en una imagen de un gato, una capa puede aprender el gato completo, mientras que otra puede aprender solo la cara del gato.

## R-CNN

[R-CNN \(Region-based CNN\)](#) se usa para la detección de objetos en imágenes. El algoritmo R-CNN introduce la idea de propuestas de región, que son regiones de la imagen que podrían contener un objeto. Luego, estas regiones se introducen en una CNN para extraer características, que luego se utilizan para clasificar los objetos en la imagen. Las versiones más recientes incluyen Fast R-CNN, Faster R-CNN y Mask R-CNN.

## GAN

[GAN \(Generative Adversarial Networks\)](#) se introdujeron como una forma de entrenar modelos generativos. Los modelos generativos son modelos que pueden producir nuevos datos realistas que no existen en el mundo real. Por ejemplo, si un modelo generativo se entrena con un conjunto de datos de imágenes de caballos, podrá producir una imagen de un caballo que en realidad no existe, sin embargo, al mirar la imagen uno no sabría que no existe.

Las GAN se componen de dos redes: una **red generadora** que aprende a crear datos que se parecen al conjunto de datos reales; y la **red discriminadora** que intenta clasificar correctamente, si un ejemplo de datos en particular es real o generado. Las dos redes se entrenan juntas para intentar mejorar ambas al mismo tiempo: el generador intenta producir datos que parezcan más reales y el discriminador intenta clasificar correctamente los ejemplos reales y falsos que se parecen cada vez más.

## Vision Transformers (ViT)

[Vision Transformers \(ViT\)](#) intentan mejorar la clasificación de imágenes. Utiliza un modelo transformador para computer vision, que se desarrolló originalmente para el procesamiento del lenguaje natural. Este modelo de transformador se aplica sobre parches de la imagen. Luego, por ejemplo, la salida del modelo del transformador se puede utilizar para clasificar las imágenes.



Amplia información sobre los [modelos preentrenados](#) en PyTorch.

# Mejorar los resultados del modelo



Qualentum Lab

Supón que has entrenado tu modelo de CNN, pero los resultados no son tan buenos como te gustaría. ¿Cuáles son algunas formas en las que puedes mejorar tu modelo?

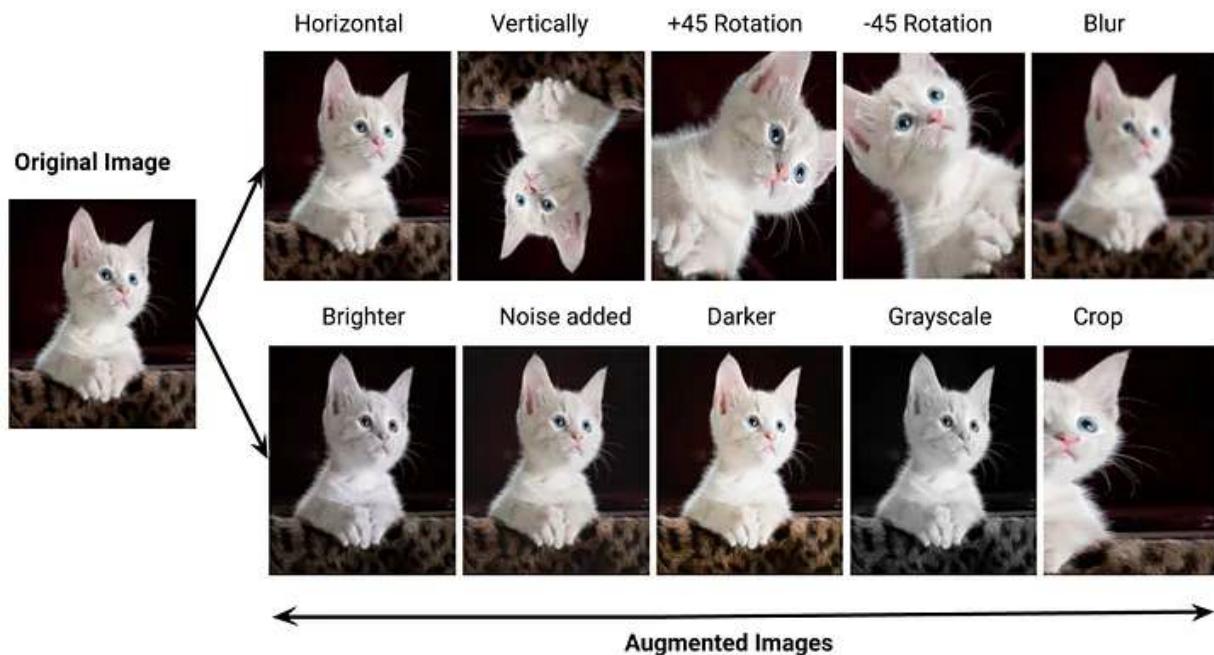
Aquí te comarto algunas **técnicas** que te sonarán...

1

## El aumento de datos

El aumento de datos es el proceso de aumentar artificialmente el conjunto de datos agregando nuevos ejemplos de datos, utilizando los datos existentes en el conjunto de datos.

En *computer vision* existen muchos tipos diferentes de aumento que puedes aplicar a las imágenes.



Fuente: <https://medium.com/@tagxdata/data-augmentation-for-computer-vision-9c9ed474291e>.

- Algunas de las principales técnicas específicas para el aumento de datos en computer vision: son las de **recortar, rotar, voltear y trasladar las imágenes o modificar la ubicación, el ángulo y el zoom** de los objetos dentro de la imagen. Esto puede ser importante para ayudar a la CNN a generalizar, ya que ve el mismo objeto de muchas maneras diferentes.
- Otras técnicas son **cambiar la luz o el color en la imagen** o modificar el brillo, el contraste, la saturación. Las imágenes pueden tener diferente luz y estar tomadas bajo diferentes condiciones ambientales, así que disponer de imágenes que reflejen una variedad de condiciones puede ayudar al algoritmo a poder generalizar mejor.
- Finalmente, una última técnica más sería la de agregar ruido y borrosidad a las imágenes, ya que se puede utilizar para ayudar al modelo a trabajar con imágenes que no se han capturado en las condiciones perfectas, como es el caso en muchas situaciones del mundo real.

-  Puedes encontrar más información sobre aumento de datos en la página de [Transforming and augmenting images de PyTorch](#).

2

## Modelos preentrenados

Ya conocemos algunos **modelos preentrenados** como VGG, ResNet, Inception... Estos modelos han sido entrenados con enormes conjuntos de datos (millones de imágenes). Usar uno de estos modelos previamente entrenados en lugar de construir uno propio puede mejorar muchas veces los resultados para un problema particular. Piensa que los pesos de estos modelos ya han sido entrenados previamente para funcionar bien. Además, puedes entrenar los pesos aún más utilizando su conjunto de datos (*fine-tuning*), de modo que sean óptimos para los datos específicos de tu problema.

3

## Siempre por una buena calidad de los datos

Como ya hemos comentado con anterioridad, y solo a modo de recordatorio, nunca debemos olvidar que la calidad de los datos puede afectar al rendimiento del modelo. Si las imágenes son de mala calidad o tienen mucho ruido, la red neuronal puede sufrir dificultades para lograr un buen rendimiento. Además, si los datos de entrenamiento no reflejan los datos de validación o prueba, esto también puede causar resultados deficientes.

Por ejemplo, para una clasificación de animales, si los datos de entrenamiento no contienen ninguno o solo contienen muy pocos elefantes, pero los datos de validación y prueba contienen bastantes elefantes, es poco probable que la red neuronal funcione bien clasificándolos, ya que nunca los ha visto o los ha visto muy pocas veces.

# Resumen y recursos de interés



En fastbook hemos estudiado una nueva y apasionante rama dentro de la IA: computer vision. Hemos descubierto algunas de sus **aplicaciones en el mundo real y en qué tipos de problemas ya se aplica la visión por computadora**. Luego hemos analizado qué tipo de datos utiliza y el preprocesamiento de estos. También hemos analizado los componentes de una red neuronal convolucional, así como su arquitectura tradicional y otras arquitecturas más avanzadas. Finalmente, tal y como hicimos en el tema dedicado al *deep learning*, hemos tratado un tema muy importante: las técnicas a las que puedes recurrir para mejorar los resultados de tu modelo.

En cuanto a las ideas que me gustaría que recordases especialmente te recojo las siguientes:

- Que hay que escalar los píxeles de las imágenes antes de empezar el entrenamiento y asegurar que todas las imágenes tienen el mismo tamaño.
- También es necesarios especificar el número de filtros, el tamaño del *kernel*, *padding* y *stride*.
- Usa una capa de *pooling* para reducir el tamaño del mapa de características de salida. Si tienes un problema de clasificación es importante incluir una capa totalmente conectada al final de la red neuronal convolucional para dar las probabilidades.

- Puedes aumentar tu conjunto de datos utilizando algunas técnicas de aumento, como voltear o rotar la imagen.

**i** Y para cerrar el fastbook aquí te comarto las referencias bibliográficas más reseñables, lecturas que te pueden resultar muy interesantes: [Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning](#) y [Analytics Vidya. Introduction to Convolutional Neural Networks](#).

¡Enhorabuena! Fastbook superado



[Qualentum.com](http://Qualentum.com)