PROCESAMIENTO DE IMÁGENES DIGITALES

MATEMÁTICA APLICADA I[1]

# Clasificación y CNNs

Belén Medrano

María José Jiménez

# Clasificación

- **Clasificación de imágenes:** tarea de asignar una imagen de entrada a una etiqueta de un conjunto fijo de categorías.

  Por ejemplo, un modelo de clasificación de imágenes que tome una sola imagen y le asigne probabilidades a 4 etiquetas: {gato, perro, sombrero, taza}

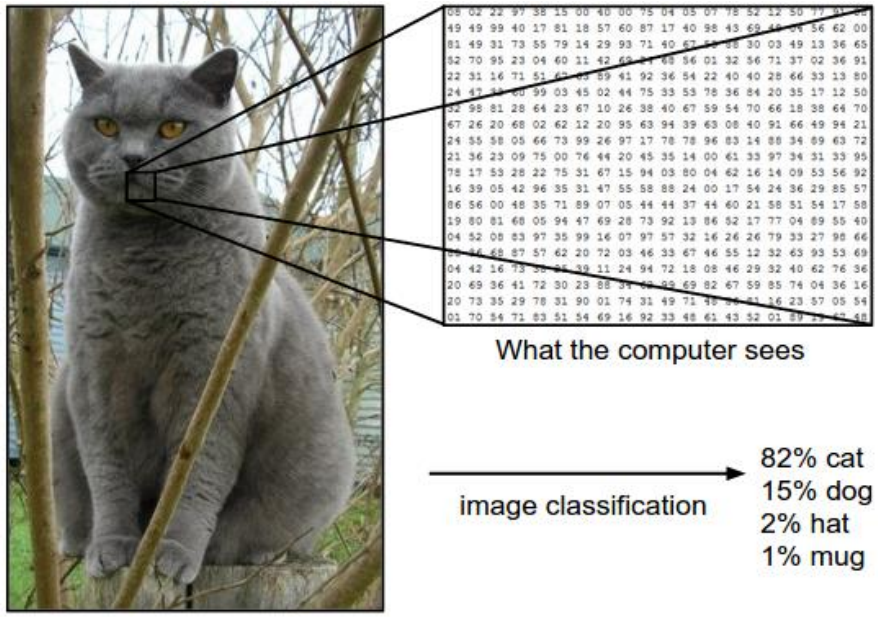
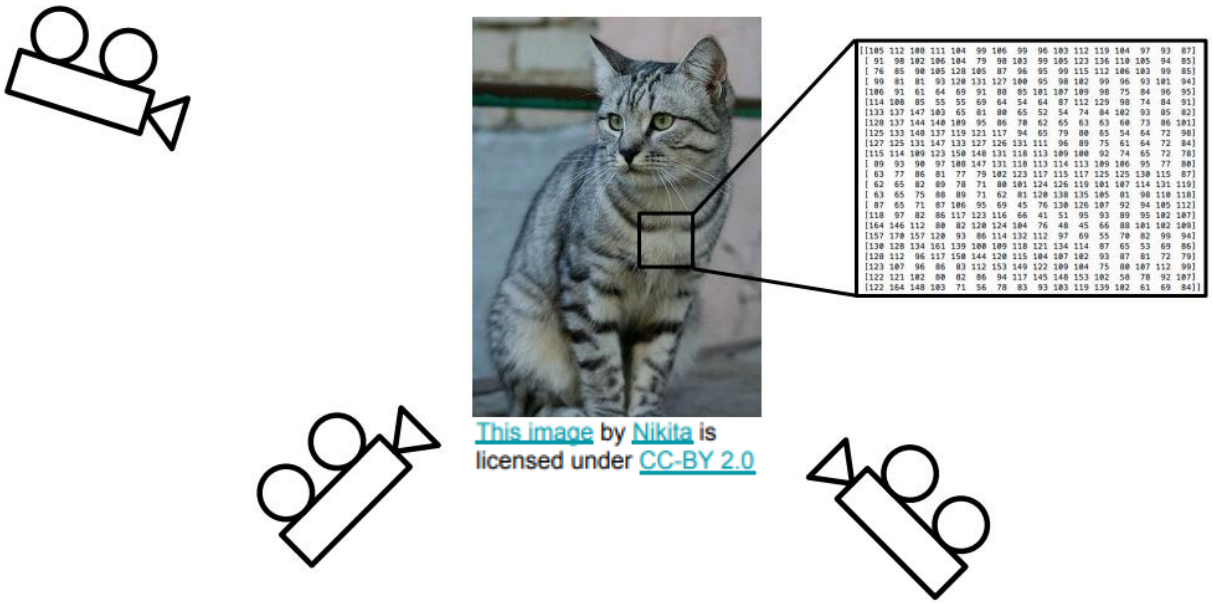



Imagen RGB $248 \times 400$ pixeles, luego un tensor de $297.600$ datos

# Clasificación

- **Desafíos de un clasificador de imágenes:**

  1. Variación del punto de vista. Todos los píxeles cambian cuando la cámara se mueve.

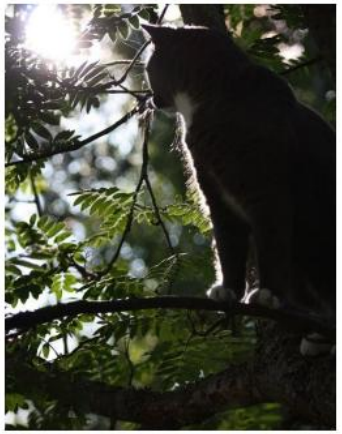


# Clasificación

- **Desafíos de un clasificador de imágenes:**

  2. Condiciones de iluminación



# Clasificación

- **Desafíos de un clasificador de imágenes:**

  3. Fondos heterogéneos. Los objetos de interés pueden mezclarse con su entorno, lo que dificulta su identificación.

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

# Clasificación

- **Desafíos de un clasificador de imágenes:**

  4. Oclusión. Los objetos de interés se pueden ocluir (solo se puede ver una pequeña parte de un objeto)

This image is CC0 1.0 public domain

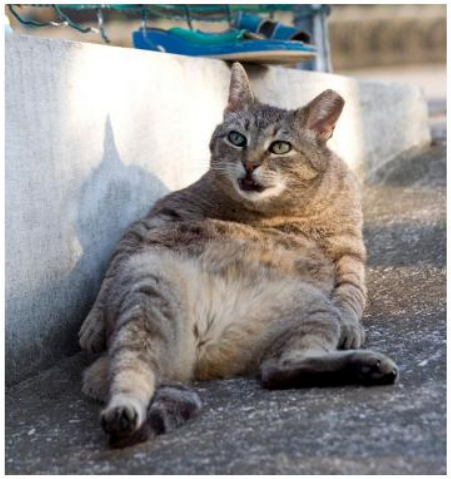
This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

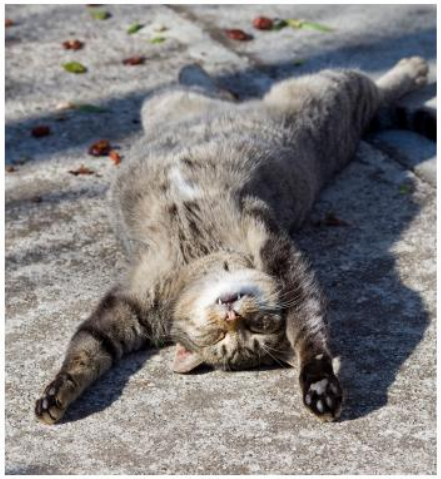
# Clasificación

- **Desafíos de un clasificador de imágenes:**

5. Deformación. Muchos objetos de interés no son cuerpos rígidos y pueden deformarse de manera extrema.

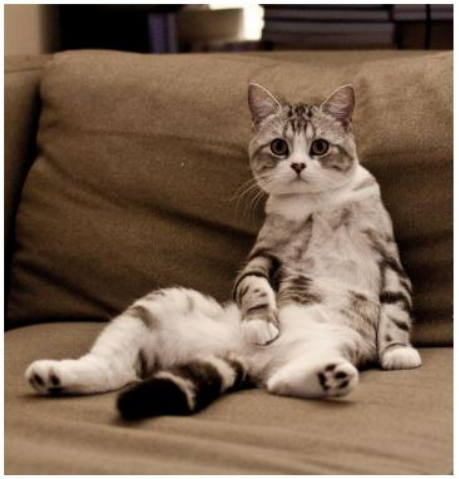












# Clasificación

- **Desafíos de un clasificador de imágenes:**

6. Variación intraclase. Las clases de interés a menudo pueden ser relativamente amplias. Hay muchos tipos diferentes de estos objetos, cada uno con su propia apariencia.

# Clasificación

**Métodos clásicos de clasificación**

- Extraer un vector de descriptores de la imagen.
- Aplicar un método de aprendizaje automático para realizar la clasificación en base a esos descriptores.
  - **Aprendizaje supervisado** (conjunto de datos de entrenamiento): SVM, Random Forest, Redes neuronales…
  - **Aprendizaje no supervisado** (métodos de clustering)

# Clasificación

**Métodos clásicos de clasificación**

- Extraer un vector de descriptores de la imagen.
- Aplicar un método de aprendizaje automático para realizar la clasificación en base a esos descriptores.

**Tipos de descriptores:**

- Topológicos
- Geométricos
- Estadísticos

...

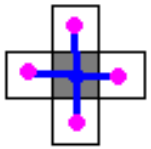
# Clasificación

**Métodos clásicos de clasificación**

- Extraer un vector de descriptores de la imagen.
- Aplicar un método de aprendizaje automático para realizar la clasificación en base a esos descriptores.

**Descriptores Topológicos**: número de componentes conexas, agujeros, esqueleto…

Dependen de la Conexión (adyacencia)



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | a | a | a | 0 | 0 | a | a | 0 | c | c |
| 0 | 0 | a | a | a | 0 | a | a | a | a | 0 | 0 | 0 | c |
| e | e | 0 | 0 | a | a | a | 0 | a | a | a | a | a | 0 |

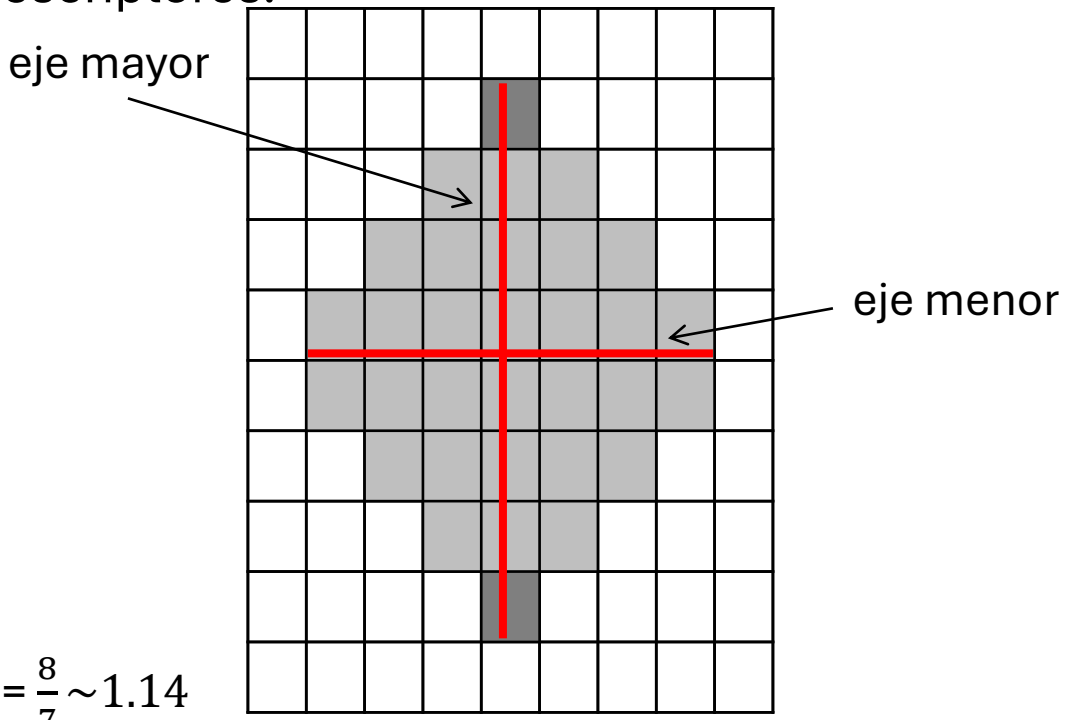
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | a | a | a | 0 | 0 | a | a | 0 | a | a |
| 0 | 0 | a | a | a | 0 | a | a | a | a | 0 | 0 | 0 | a |
| a | a | 0 | 0 | a | a | a | 0 | a | a | a | a | a | 0 |

# Clasificación

**Métodos clásicos de clasificación**

- Extraer un vector de descriptores de la imagen.
- Aplicar un método de aprendizaje automático para realizar la clasificación en base a esos descriptores.

**Descriptores geométricos:**
área, perímetro, diámetro, compacidad, excentricidad…




Diámetro = 8

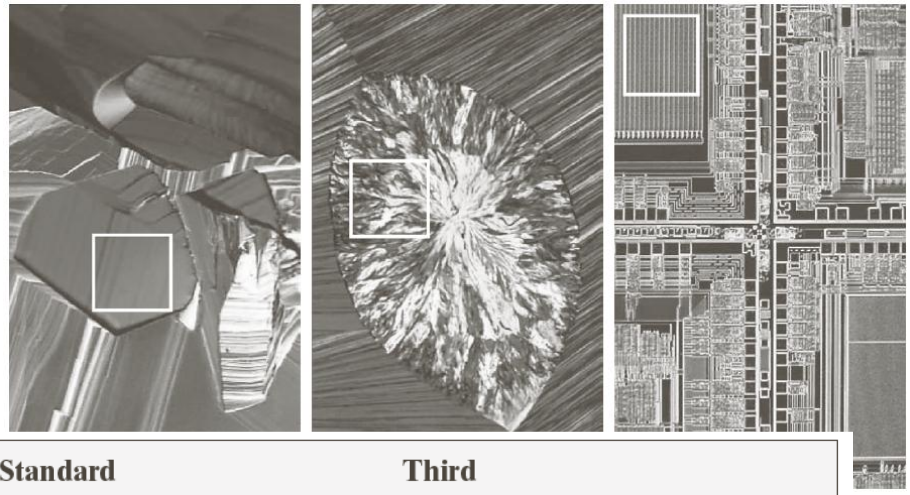
Excentricidad = $\frac{8}{7} \sim 1.14$

# Clasificación

**Métodos clásicos de clasificación**

- Extraer un vector de descriptores de la imagen.
- Aplicar un método de aprendizaje automático para realizar la clasificación en base a esos descriptores.

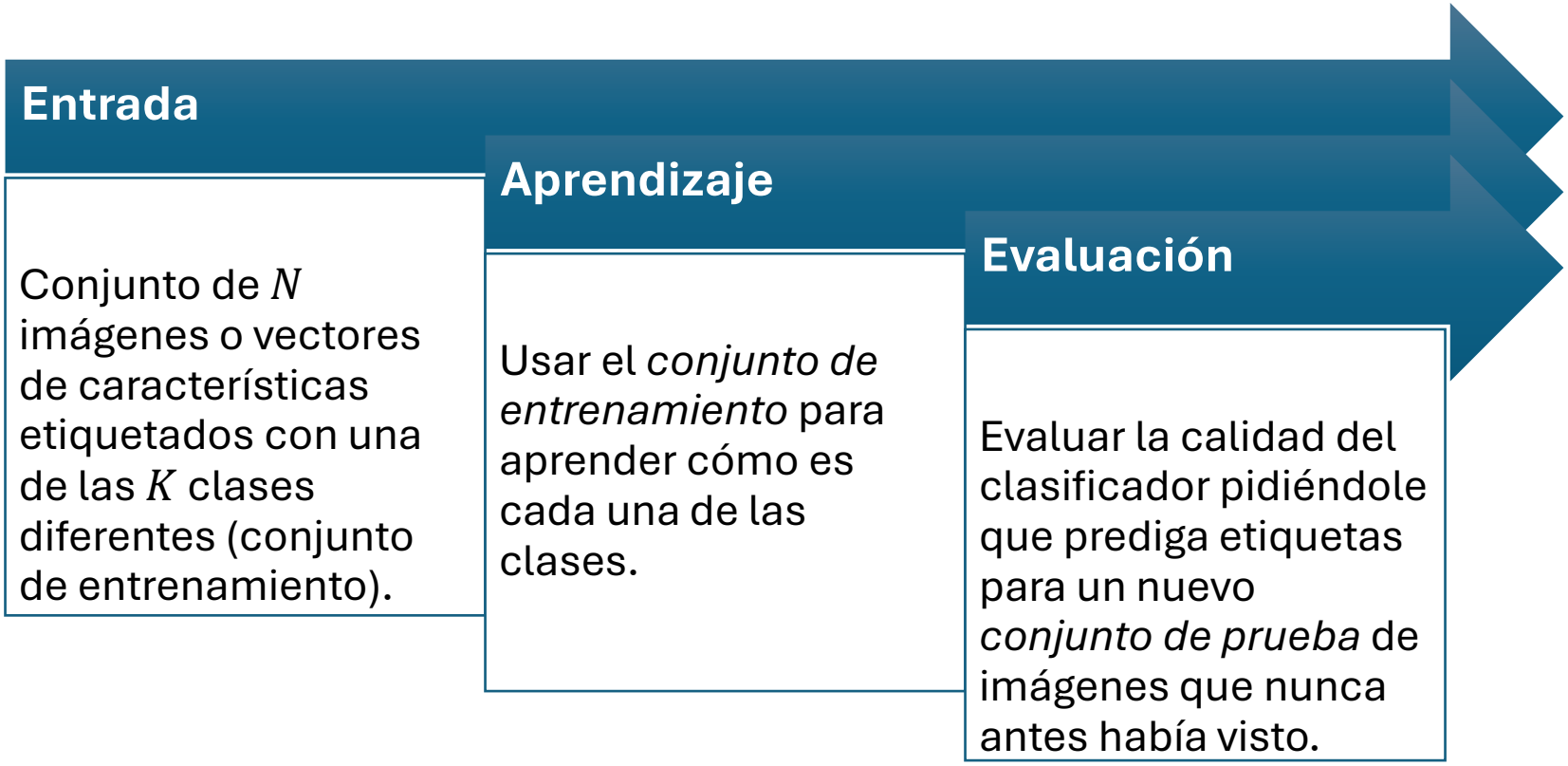
**Descriptores estadísticos:**
Media, desviación típica,
momentos centrales, entropía, ...



| Texture | Mean | Standard deviation | *R* (normalized) | Third moment | Uniformity | Entropy |
|---|---|---|---|---|---|---|
| Smooth | 82.64 | 11.79 | 0.002 | −0.105 | 0.026 | 5.434 |
| Coarse | 143.56 | 74.63 | 0.079 | −0.151 | 0.005 | 7.783 |
| Regular | 99.72 | 33.73 | 0.017 | 0.750 | 0.013 | 6.674 |

# Clasificación

- **Flujo general de aprendizaje automático para clasificación**



# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**

  Este clasificador es un algoritmo básico de aprendizaje supervisado que nos permitirá hacernos una idea del problema de clasificación de imágenes.

  Tenemos un conjunto de imágenes etiquetadas con la clase a la que pertenecen.

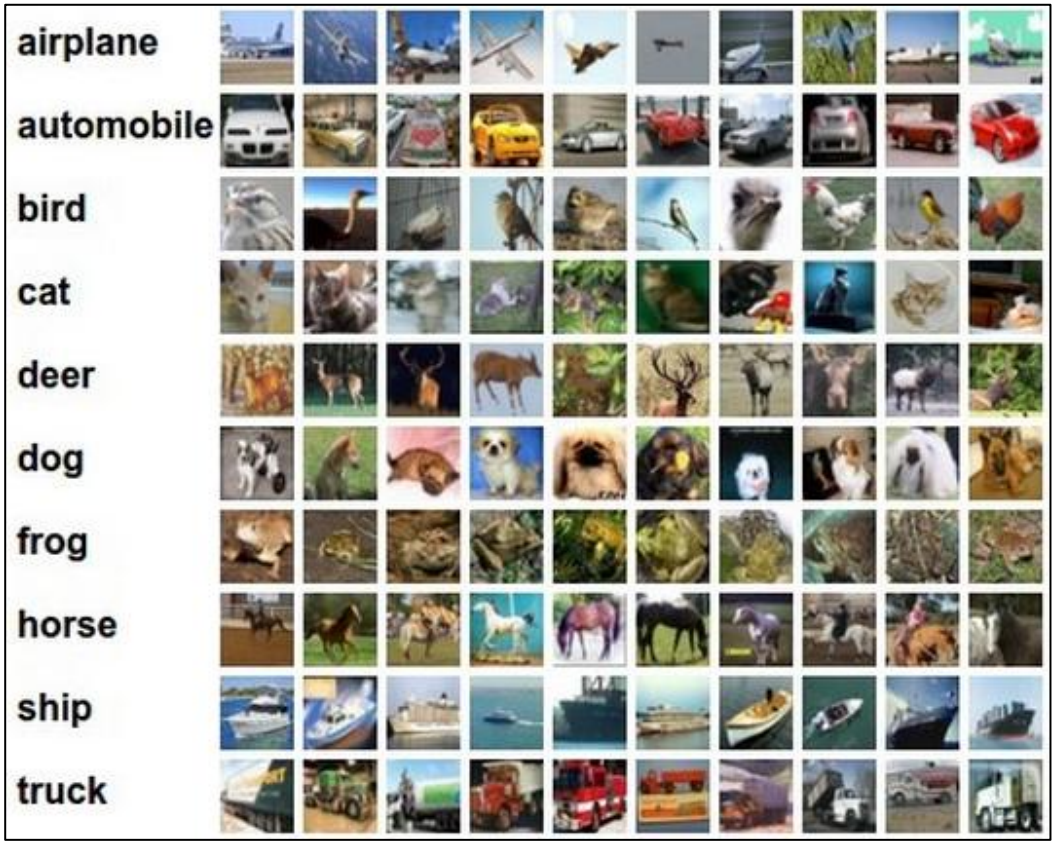
# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**

**Dataset: CIFAR-10.**

60.000 imágenes
$32 \times 32$ píxeles (50.000 imágenes de entrenamiento y 10.000 imágenes de prueba).

Cada imagen etiquetada con una de las 10 clases ($K = 10$).



# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**

A una nueva imagen se le asigna la clase de su **vecino** más cercano → necesitamos una **distancia.**

Dos conceptos clave:

- Espacio de características: La imagen completa o definido por los descriptores escogidos.

- Distancia: dependiendo del espacio de características, puede ser mejor una u otra.

# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**


Imágenes de entrenamiento (etiquetadas)

- Dada la imagen a evaluar $I_1$.
- Compara $I_1$ con las imágenes de entrenamiento con una función distancia.
- Predice para $I_1$ la etiqueta de la imagen de entrenamiento **más cercana.**

# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**

Caso de espacio de características definido por la imagen completa

- Distancia $L_1$ → $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image

| | | | |
|---|---|---|---|
| 56 | 32 | 10 | 18 |
| 90 | 23 | 128 | 133 |
| 24 | 26 | 178 | 200 |
| 2 | 0 | 255 | 220 |

-

training image

| | | | |
|---|---|---|---|
| 10 | 20 | 24 | 17 |
| 8 | 10 | 89 | 100 |
| 12 | 16 | 178 | 170 |
| 4 | 32 | 233 | 112 |

=

pixel-wise absolute value differences

| | | | |
|---|---|---|---|
| 46 | 12 | 14 | 1 |
| 82 | 13 | 39 | 33 |
| 12 | 10 | 0 | 30 |
| 2 | 32 | 22 | 108 |

→ 456

- Distancia $L_2$ → $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$

# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**


Los 10 vecinos más cercanos de cada imagen test en el conjunto de entrenamiento según la distancia $L_1$.

# Clasificación

- **Un primer enfoque: Nearest Neighbor Classifier**


La imagen de entrenamiento más cercana a la cabeza del caballo es un automóvil rojo, presumiblemente debido al fuerte fondo negro.

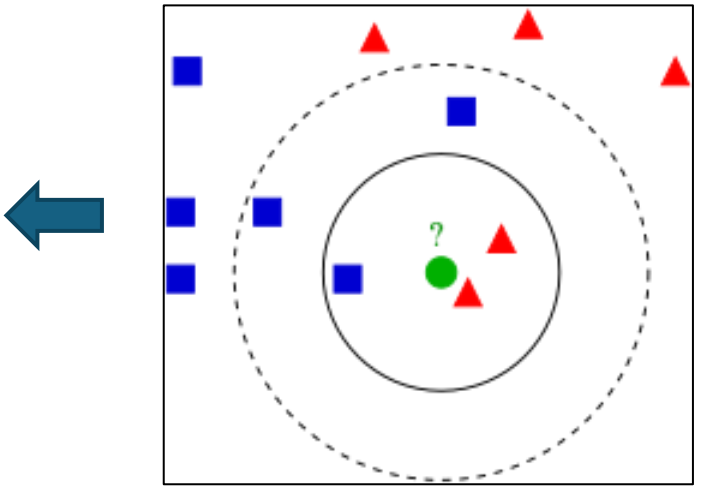
Como resultado, esta imagen de un caballo en este caso estaría mal etiquetada como un automóvil.

# Clasificación

- **Un primer enfoque: k-Nearest Neighbors Classifier**

  La idea es muy simple: en lugar de encontrar la imagen más cercana en el conjunto de entrenamiento, encontraremos las k imágenes más cercanas. Se le dará a la nueva imagen la etiqueta de aquella que más votos tenga.

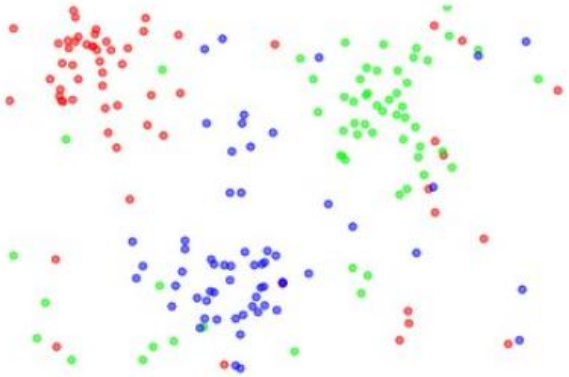
$k = 3$ → el punto verde sería clasificado como un triángulo rojo.
$k = 5$ → el punto verde sería clasificado como un cuadrado azul.



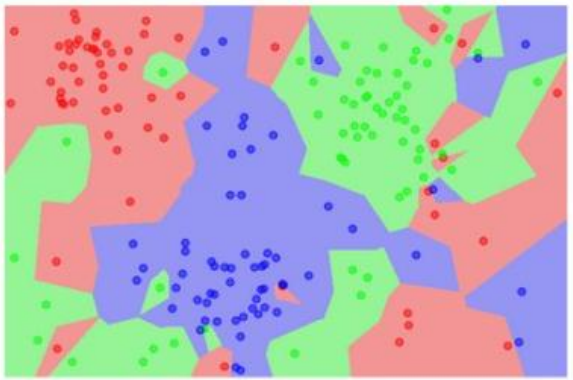
# Clasificación

- **Un primer enfoque: k-Nearest Neighbors Classifier**

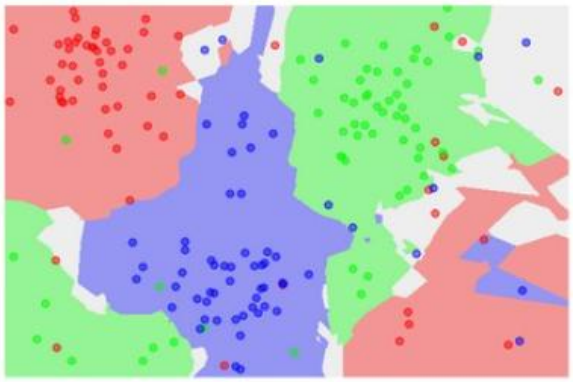
Datos de entrenamiento



$k = 1$



$k = 5$

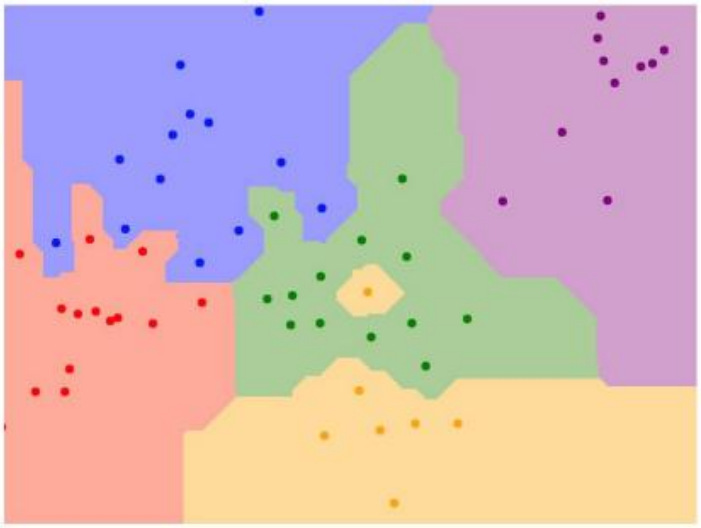


Las zonas blancas son aquellos datos que presentan un empate en el número de votos.

# Clasificación

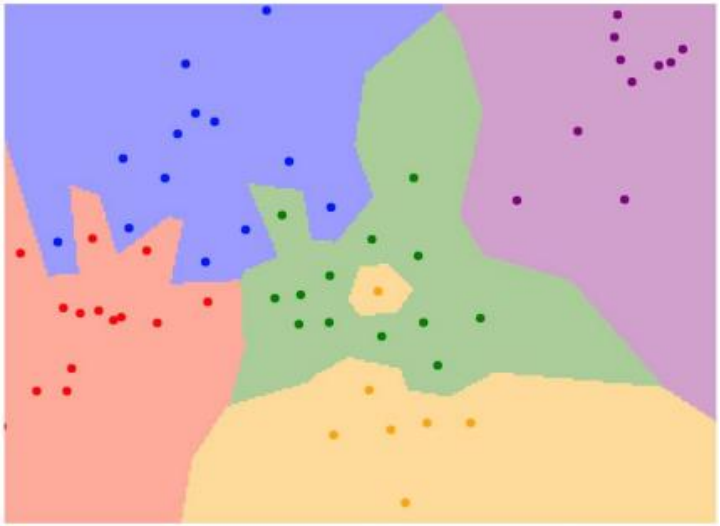
- **Un primer enfoque: k-Nearest Neighbors Classifier**

Distancia $L_1$



$k = 1$

Distancia $L_2$



$k = 1$

# Clasificación

- **Un primer enfoque: k-Nearest Neighbors Classifier**

  ¿Cuál es el mejor valor de **k** para usar?

  ¿Cuál es la mejor **distancia** para usar?

  } **Hiperparámetros**

  **Desventajas:**

  - El clasificador debe recordar todos los datos de entrenamiento y almacenarlos para futuras comparaciones con los datos de prueba.
  - Clasificar una imagen de prueba es costoso ya que requiere una comparación con todas las imágenes de entrenamiento.

# Redes Neuronales Convolucionales

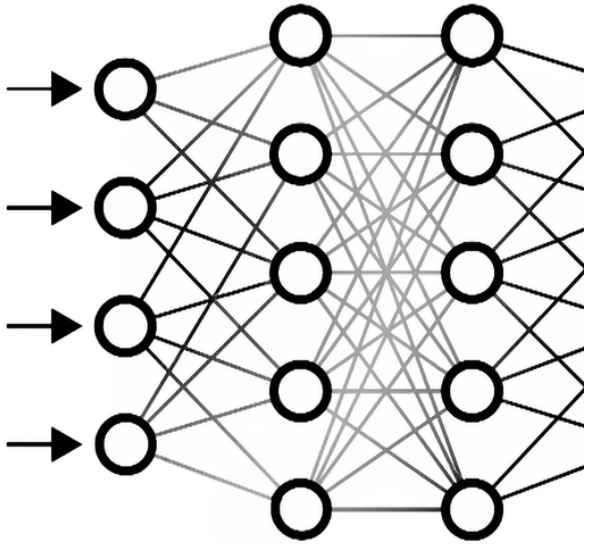
**Aprendizaje profundo: Redes neuronales de convolución**
La mayoría de métodos de clasificación (y de segmentación) actuales están basados en redes neuronales de convolución (CNN).

Una red neuronal convolucional es un tipo de red neuronal creada para sacar partido a la **estructura espacial** existente en una imagen digital.

# Redes Neuronales Convolucionales

- Con una red neuronal multicapas, introduciríamos cada pixel como una variable independiente, sin tener en cuenta su posición dentro de la imagen.

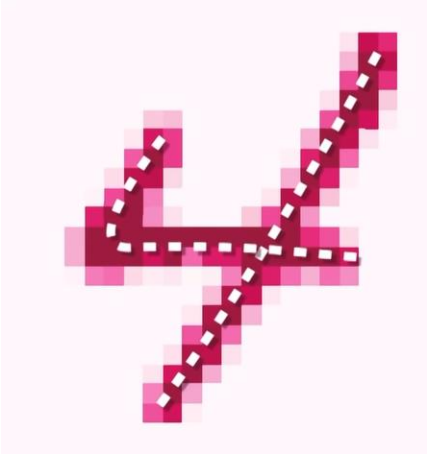

$28 \times 28$ píxeles



$784 \times 1$ píxeles

# Redes Neuronales Convolucionales

- El valor de un pixel en una imagen está muy ligado al de sus píxeles vecinos y eso hace que surjan estructuras y patrones que nos ayuden a entender qué estamos viendo.



- Por ello surgen las REDES NEURONALES CONVOLUCIONALES.

# Redes Neuronales Convolucionales

- Nuestro córtex visual realiza un procedimiento en cascada donde primero se identifican elementos básicos y generales que posteriormente se combinan para generar patrones más complejos.

- En una red neuronal convolucional cada capa va aprendiendo diferentes niveles de abstracción.

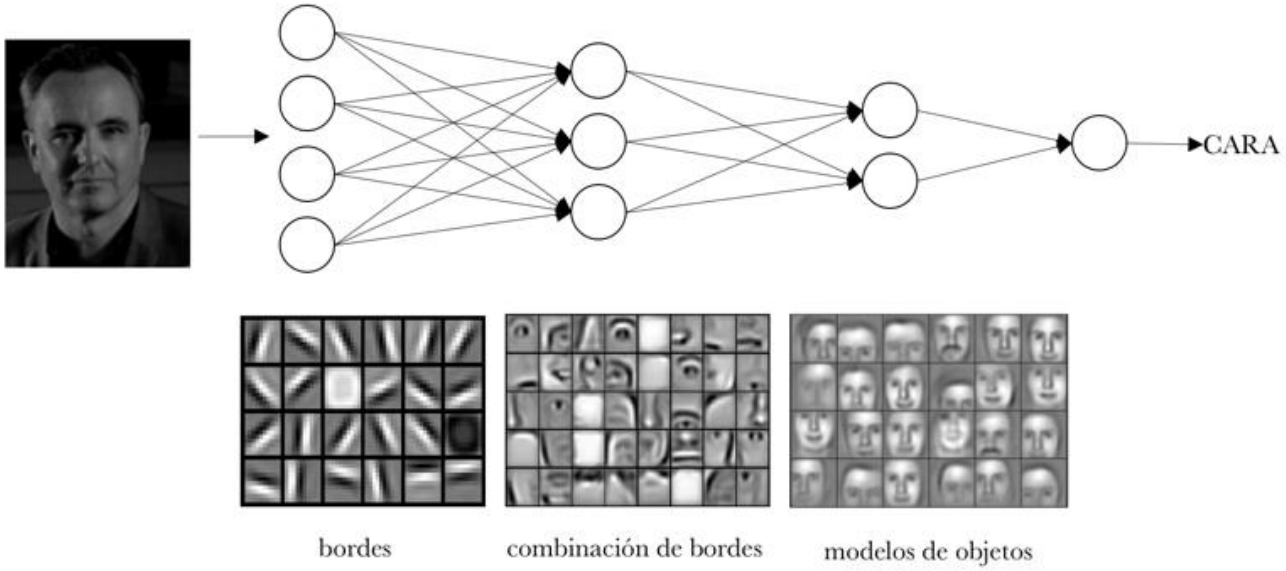



Imagen: Plain concepts: Vision Transformers

# Redes Neuronales Convolucionales

- La operación fundamental detrás de las redes convolucionales, como su nombre bien indica, es la convolución ya estudiada en temas anteriores.

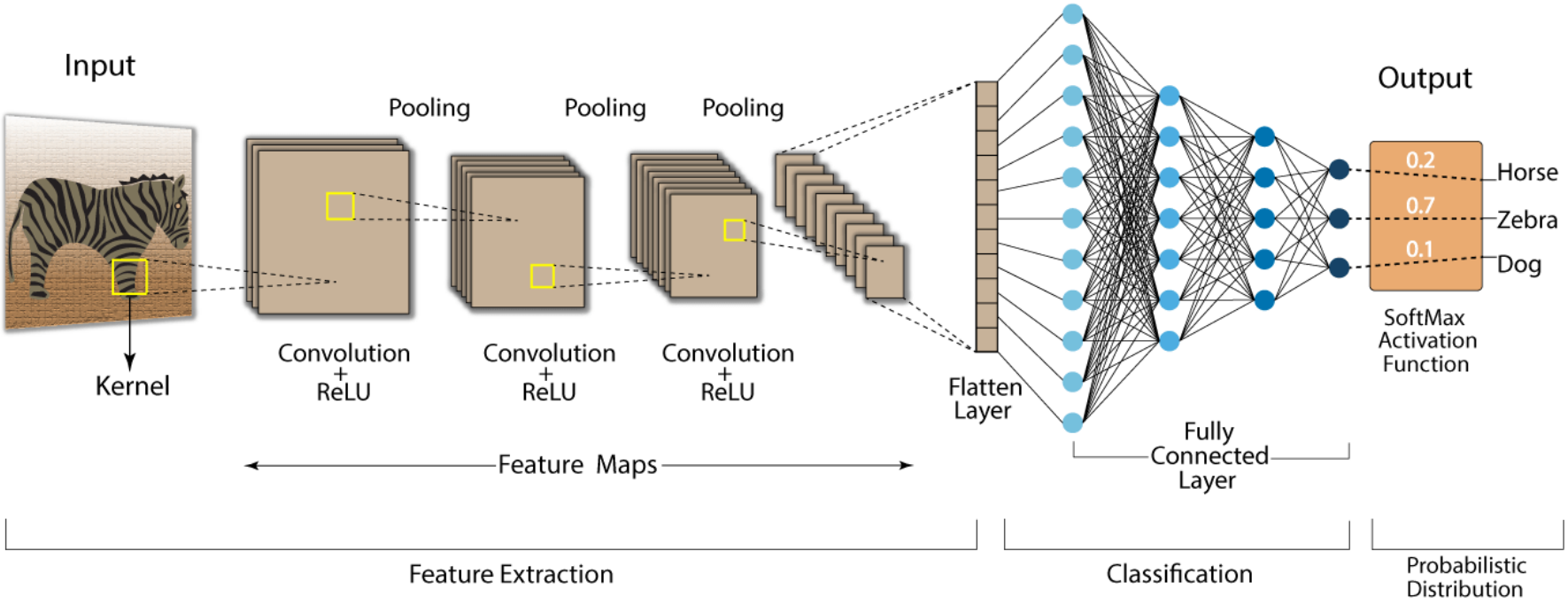
- Recuerda que el efecto que produce una convolución en una imagen dependerá de los valores del filtro (núcleo o kernel) que definamos.


Esos valores son los que la red neuronal irá aprendiendo para realizar mejor su propósito. Es decir, aprender los filtros para detectar formas en el principal trabajo de este tipo de red neuronal.

# Redes Neuronales Convolucionales

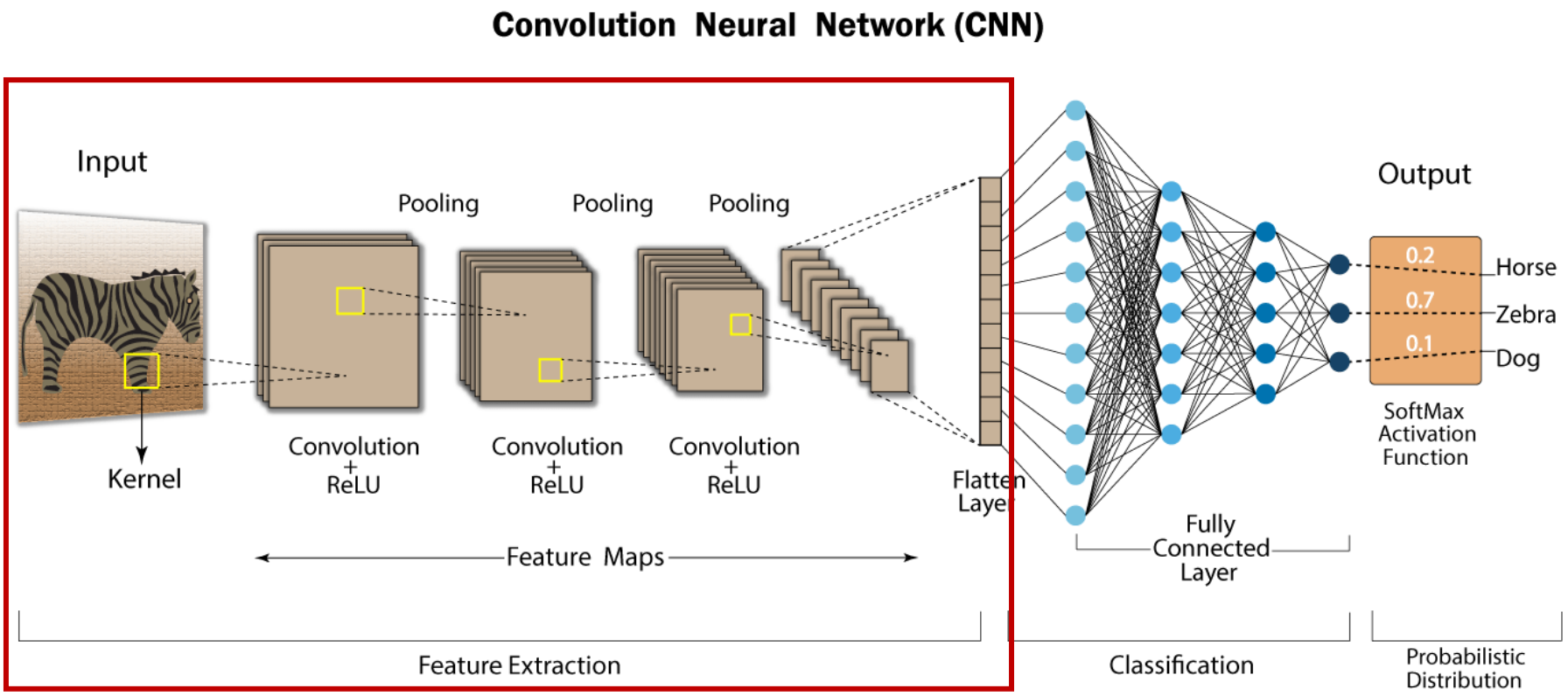


- **Estructura general de una CNN:**




Fuente

# Redes Neuronales Convolucionales



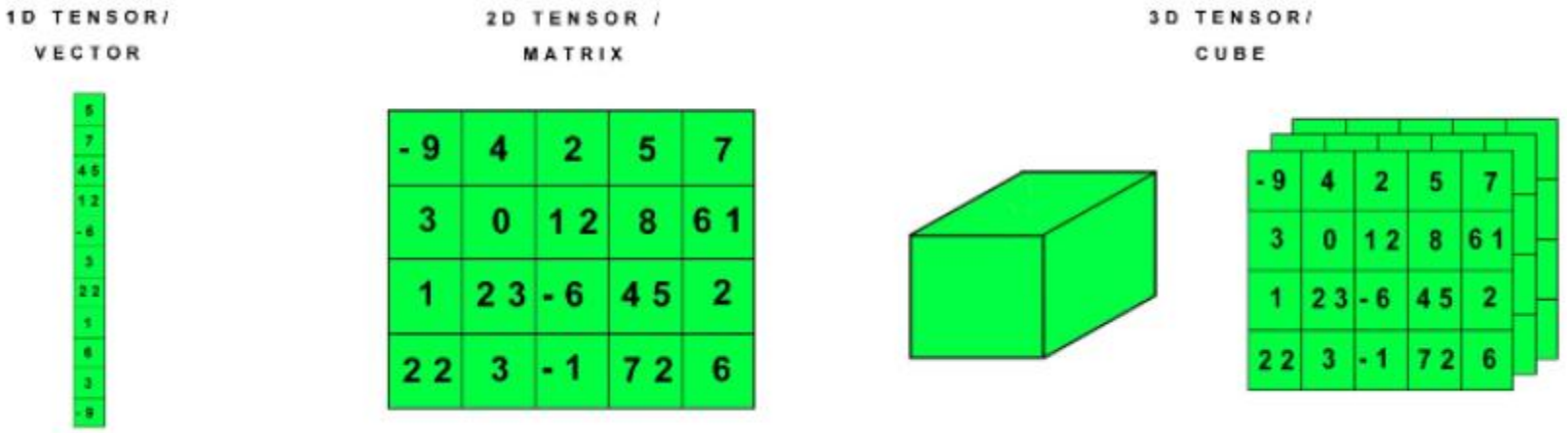
- **Capas convolucionales:**

  Propiedad 1: **Detectan características** o rasgos visuales en las imágenes como bordes, líneas, texturas, etc. Una vez aprendida una característica en un punto concreto de la imagen la puede reconocer después en cualquier parte de la misma. En cambio, en una red neuronal densamente conectada tiene que aprender el patrón nuevamente si este aparece en una nueva localización de la imagen.

  Propiedad 2: **Aprenden jerarquías espaciales** de patrones preservando relaciones espaciales. Por ejemplo, una primera capa convolucional puede aprender elementos básicos como aristas, y una segunda capa convolucional puede aprender patrones compuestos de elementos básicos aprendidos en la capa anterior. Y así sucesivamente hasta ir aprendiendo patrones más complejos.

- **Capas convolucionales:**

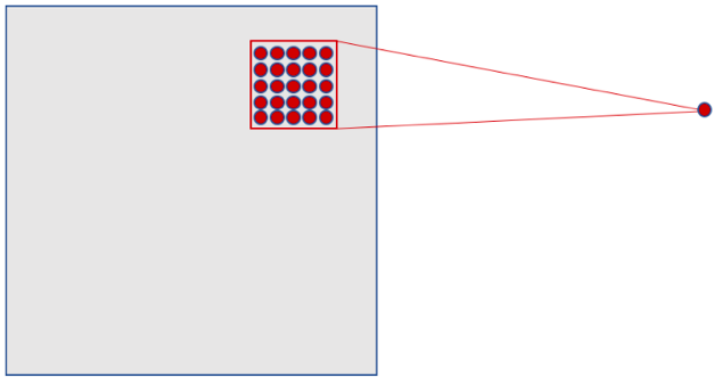
  En general, en las capas convoluciones se opera mediante tensores de 3D:
  - dos ejes espaciales: altura y anchura (*height y width*),
  - un eje de canal (*channels*) también llamado profundidad (*depth*).



- **Capas convolucionales:**

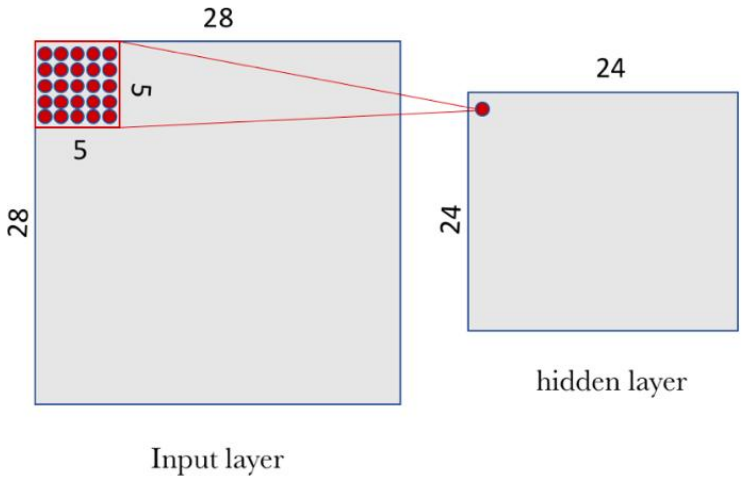
  Supongamos como input de la red una imagen 28x28 pixeles en escala de grises, luego un tensor 3D de la forma (*height*=28, *width*=28, *depth*=1).

  No se conectan todas las neuronas de entrada con todas las neuronas de la primera capa oculta de la red, como en el caso de las redes neuronales densamente conectas.



Por ejemplo, cada neurona de nuestra capa oculta estará conectada a una pequeña región de 5×5 neuronas (es decir 25 neuronas) de la capa de entrada.

- **Capas convolucionales:**

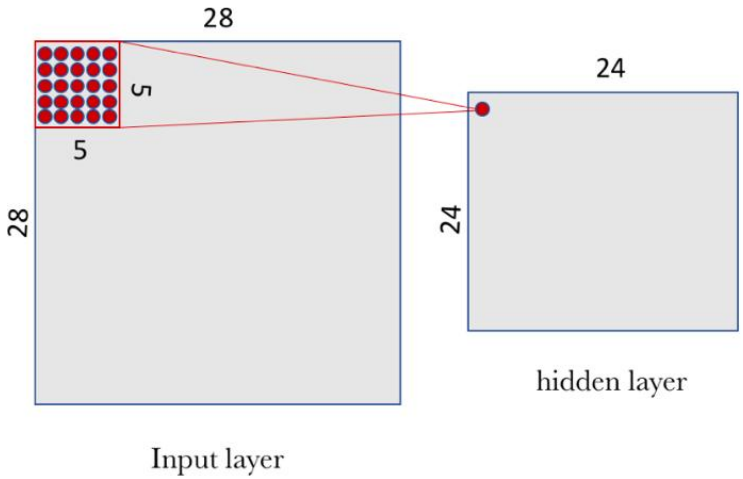



Si tenemos una entrada de 28×28 píxeles y un filtro de 5×5, esto nos define un espacio de 24×24 neuronas en la primera capa oculta.

- Para "conectar" cada neurona de la capa oculta con las 25 neuronas que le corresponden de la capa de entrada usaremos un valor de *sesgo b* y una *matriz de pesos W* de tamaño 5×5 que llamaremos filtro (*kernel*). El valor de cada punto de la capa oculta corresponde con realizar una *convolución con el kernel y sumar el sesgo*.

- Se usa el mismo filtro para todas las neuronas de la capa oculta

- **Capas convolucionales:**



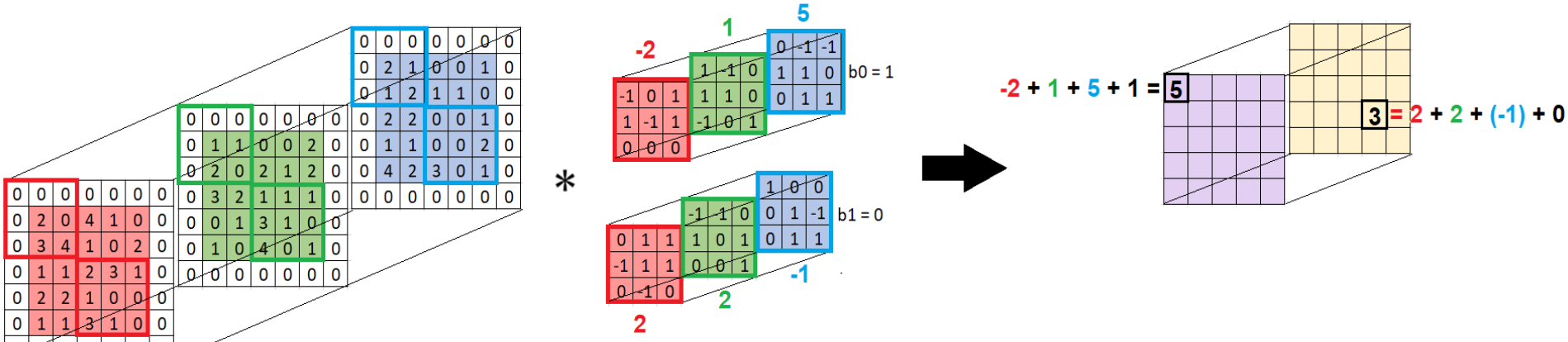

parámetro *padding*

parámetro *stride*

- *Padding* se refiere a añadir bordes alrededor de la imagen para controlar la dimensión de la salida. Si no se usa padding, la salida se encoge a medida que se aplican las convoluciones.
- *Stride* es el número de píxeles que el filtro se mueve cada vez que se aplica. Un stride pequeño significa que el filtro se mueve lentamente, mientras que un stride más grande mueve el filtro más rápidamente, reduciendo la resolución del mapa de características.

# Redes Neuronales Convolucionales

- **Capas convolucionales:**




Input: Tensor $3D$ de tamaño $5 \times 5 \times 3$

$2$ filtros (kernels) de tamaño $3 \times 3 \times 3$

Padding $P = 1$

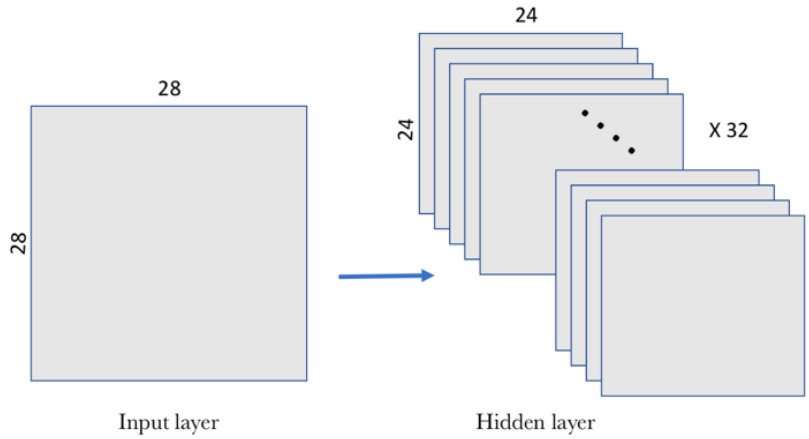
Stride $S = 1$

Output: Tensor $3D$ de tamaño $5 \times 5 \times 2$

# Redes Neuronales Convolucionales

- **Capas convolucionales:**

  1ª capa oculta: Capa convolucional de 32 filtros 5x5




Un filtro en este contexto de redes convolucionales, es similar a los filtros que usamos para procesar imágenes, que sirven para buscar características locales.
Se usan varios filtros a la vez para detectar distintas características.

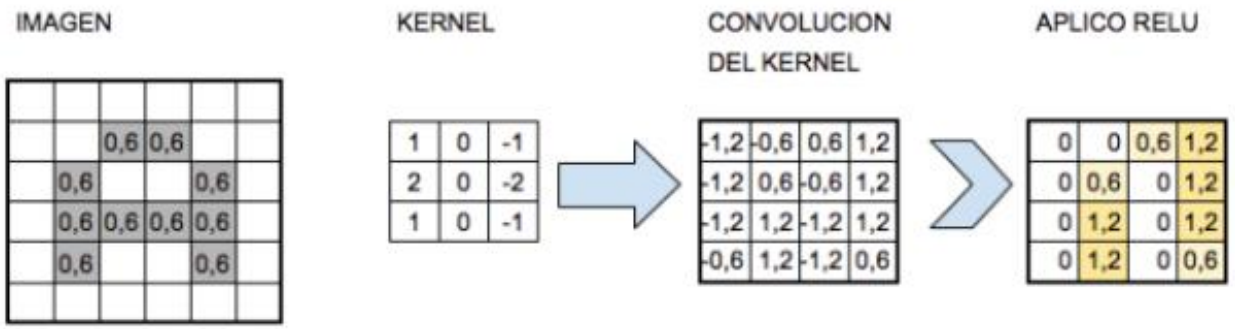
Sin padding, al aplicar filtros 5x5, pasamos de 28x28 a 24x24

- **Capa de Activación (ReLU)**

Después de la convolución, se utiliza una función de activación para introducir no linealidad en la red, lo que permite a la red aprender relaciones más complejas entre los datos. En CNNs, la función de activación más común es ReLU (Rectified Linear Unit).

Función ReLu

$$f(x) = \max(0, x)$$



- **Ejemplo:**

✓ Entrada: Imagen en escala de grises de 28×28 pixeles.

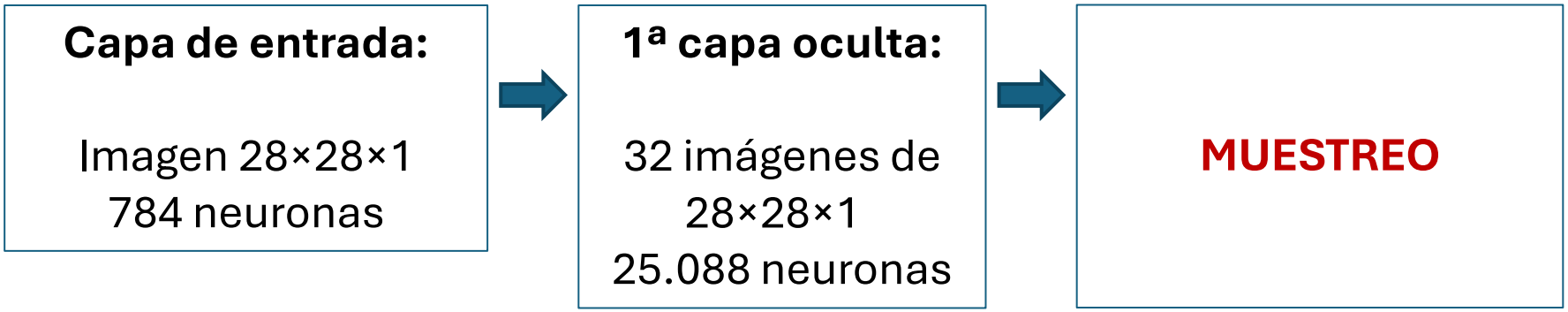
✓ Capa 1: 32 kernels → 32 imágenes filtradas nuevas 28×28×1 (dibujan ciertas características de la imagen original que ayudará en el futuro a poder distinguir un objeto de otro) → 32 mapas de características.

| **Capa de entrada:** | → | **1ª capa oculta:** | → | **2ª capa oculta:** |
|---|---|---|---|---|
| Imagen 28×28×1<br>784 neuronas | | 32 imágenes de<br>28×28×1<br>25.088 neuronas | | ¡Requeriría un poder computacional importante! |

- **Ejemplo:**

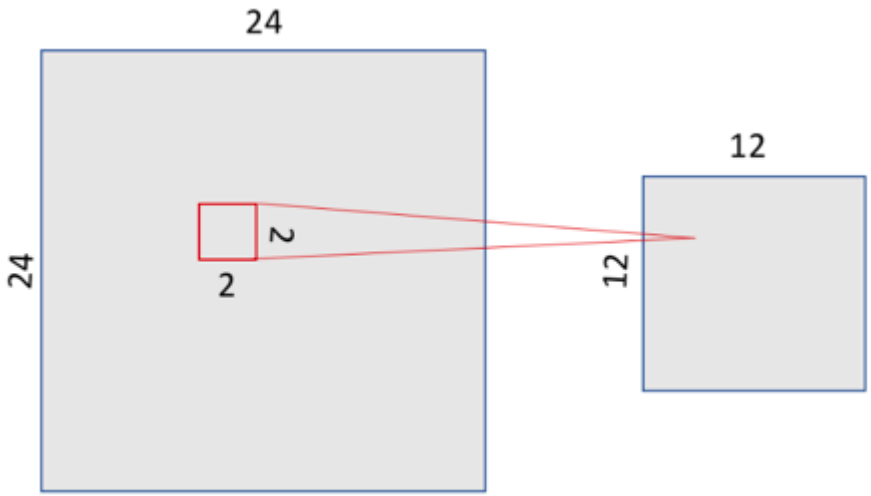
✓ Entrada: Imagen en escala de grises de 28×28 pixeles.

✓ Capa 1: 32 kernels → 32 imágenes filtradas nuevas 28×28×1 (dibujan ciertas características de la imagen original que ayudará en el futuro a poder distinguir un objeto de otro) → 32 mapas de características.



- **Capas de pooling (o muestreo):**

Las capas de pooling hacen una simplificación de la información recogida por la capa convolucional y crean una versión condensada de la información contenida en estas.

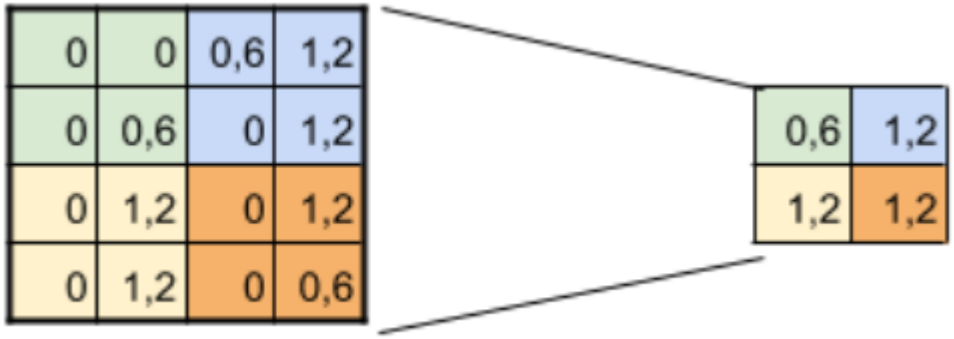



Por ejemplo, escoger una ventana de 2×2 de la capa convolucional y vamos a sintetizar la información en un punto en la capa de pooling.

Hay varias maneras de condensar la información:
*max-pooling* - valor máximo de los que había en la ventana de entrada de 2×2 en nuestro caso
average-pooling – valor promedio

- **Muestreo:**

✓ Hay diversos tipos de muestreo, el más usado es: **Max-Pooling**.



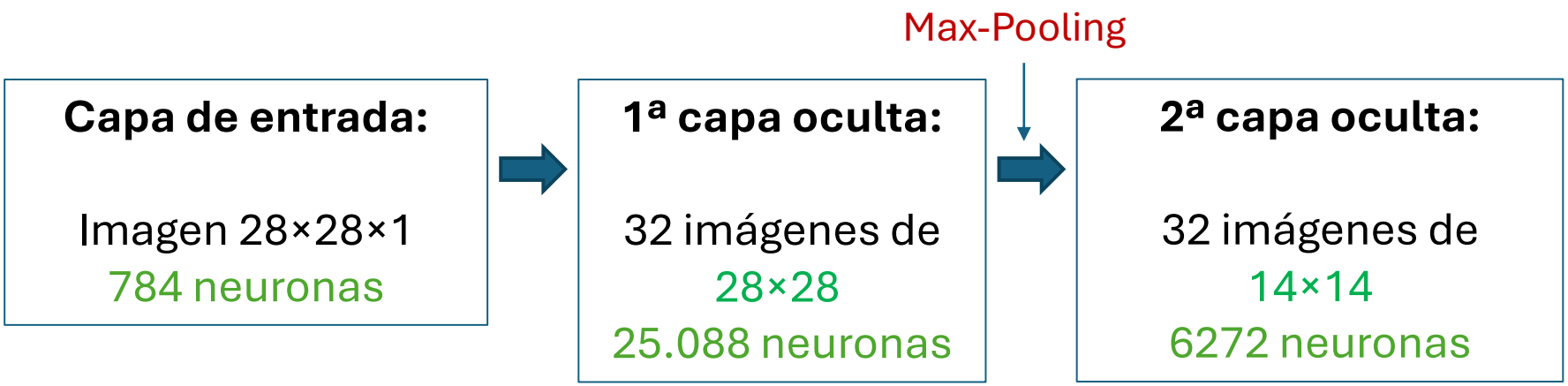

En este caso, usando 2×2, la imagen resultante es reducida a la mitad.

# Redes Neuronales Convolucionales

- **Ejemplo:**

✓ Entrada: Imagen en escala de grises de 28×28 pixeles.

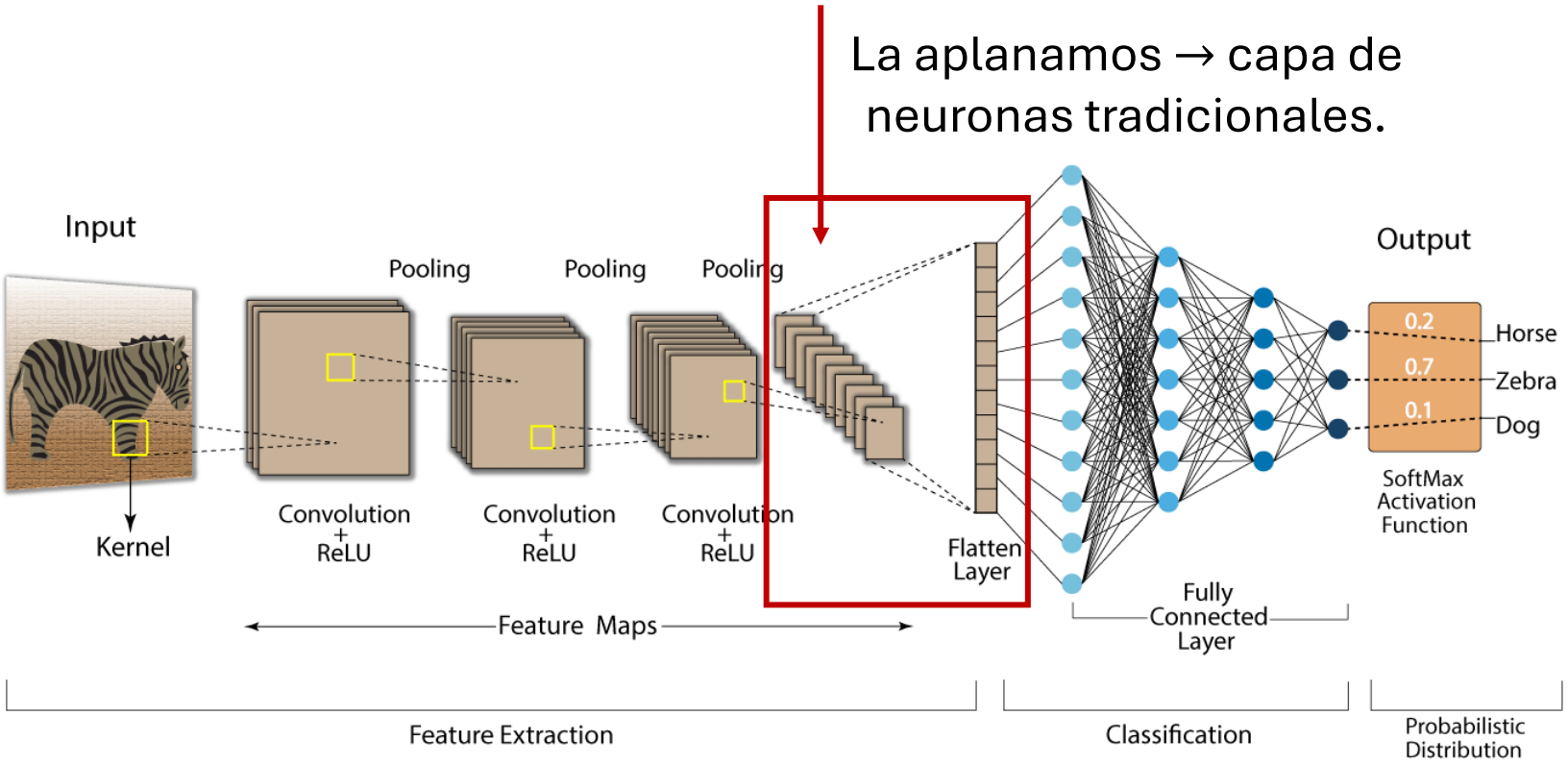
✓ Capa 1: 32 kernels → 32 imágenes filtradas nuevas 28×28×1 (dibujan ciertas características de la imagen original que ayudará en el futuro a poder distinguir un objeto de otro).




Se va reduciendo la resolución de los mapas de características

# Redes Neuronales Convolucionales

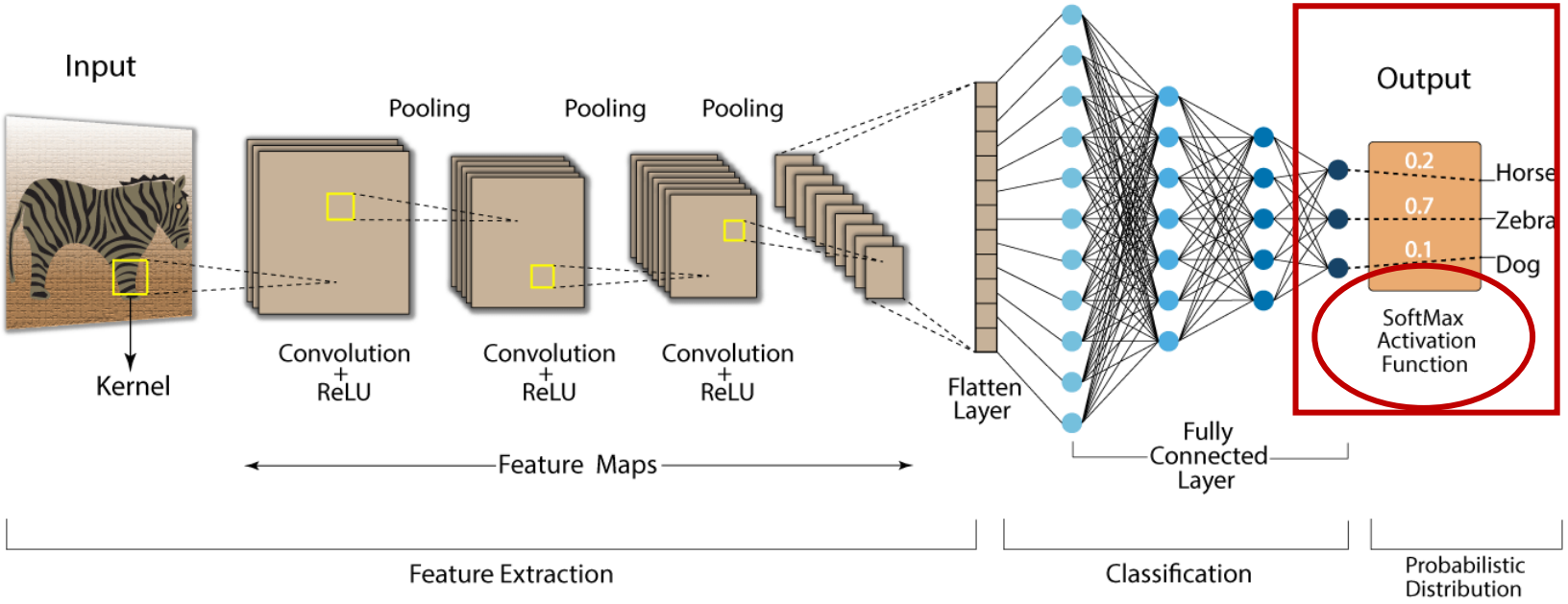
- **<u>Última capa de convolución</u>:** Se dice que es tridimensional por tomar la forma 3 x 3 x (número de mapas de características)



# Redes Neuronales Convolucionales

- **<u>Función SoftMax:</u>** Pasar a probabilidad (entre 0 y 1) a las neuronas de salida.

  ✓ Ejemplo: $[0.2, 0.8, 0]$ nos indica $20\%$ de probabilidad de que sea caballo, $80\%$ de que sea cebra y $0\%$ de que sea perro.



## Entrenamiento

1. **Preparación de los datos**:
    1. Si dividimos un conjunto de entrenamiento de 1,000 imágenes en **batches (o lotes) de 10 imágenes**, tendremos **100 batches** en total para procesar.
    2. Cada batch será procesado por la red en una **iteración**.

2. **Primera iteración (primer batch)**:
    1. Tomamos las **primeras 10 imágenes** del conjunto de datos (el primer batch).
    2. Este batch completo (10 imágenes) se pasa por la red **en paralelo**, no una por una.

**¿Qué pasa con el batch dentro de la red?**

**a. Propagación hacia adelante (Forward Propagation):**

•Cada imagen del batch pasa simultáneamente por la red (en paralelo si usamos una GPU).

•La red aplica las capas convolucionales, ReLU, pooling, y totalmente conectadas a cada una de las 10 imágenes.

•Al final, obtendremos las predicciones (outputs) para las 10 imágenes.

Por ejemplo: Si estamos clasificando en 5 clases, obtendremos 10 vectores de 5 probabilidades cada uno, uno por imagen.

**b. Cálculo del error (Loss):**

•Comparamos las predicciones de la red con las etiquetas reales del conjunto de entrenamiento.

•Usamos una **función de pérdida** para calcular el **error promedio** del batch (es decir, la pérdida combinada para las 10 imágenes).

**c. Retropropagación (Backward Propagation):**

•Basándonos en el error calculado, la red realiza **retropropagación**. Esto significa que la red calcula cómo debería ajustar sus pesos (filtros) para reducir el error en el futuro. Esto se hace utilizando el **gradiente del error con respecto a los pesos**.

**d. Actualización de pesos:**

•Una vez calculados los gradientes, la red usa un optimizador (por ejemplo, *SGD* = Stochastic Gradient Descent o *Adam*) para **actualizar los pesos**.

•Los pesos se actualizan **una vez por cada batch**, no por cada imagen. Es decir, después de procesar las 10 imágenes del batch, la red actualiza sus pesos.

**Repetición del proceso para los siguientes batches**

**3.Pasar al siguiente batch**:

3. Ahora tomamos las siguientes 10 imágenes (el segundo batch) y repetimos el proceso:
    3. Propagación hacia adelante.
    4. Cálculo del error.
    5. Retropropagación.
    6. Actualización de pesos.
4. Esto continúa para todos los 100 batches en la época.

**4.Final de la época**:

3. Una vez que se han procesado los 100 batches (1,000 imágenes), termina la primera época.
4. Si decidimos entrenar por más épocas, la red volverá a pasar por las mismas 1,000 imágenes, pero esta vez con los pesos ya ajustados tras la primera época.

**¿Qué pasa en la segunda época?**

1. Antes de comenzar la segunda época, las 1,000 imágenes del conjunto de datos se **reorganizan aleatoriamente** (shuffling).
2. El objetivo del shuffling es evitar que la red dependa demasiado del orden de las imágenes en el conjunto de datos. Esto ayuda a mejorar el aprendizaje general.

# Redes Neuronales Convolucionales

**Ejemplos de CNNs relevantes:**

*LeNet-5 (1998)*
Clasificación de dígitos escritos a mano (MNIST). Red simple, pionera en CNNs.

*AlexNet (2012)*
Ganadora de ImageNet 2012. Primera red profunda moderna, 5 capas convolucionales, uso de ReLU.

*VGG-16 (2014)*
Clasificación en ImageNet. Filtros 3x3, red muy profunda (16 capas).

*GoogLeNet (Inception) (2014)*
Ganadora de ImageNet 2014. Bloques Inception (combinación de convoluciones 1x1, 3x3, 5x5).

*ResNet (2015)*
Clasificación en ImageNet. Bloques residuales, redes extremadamente profundas (hasta 152 capas).

*DenseNet (2016)*
Clasificación en ImageNet. Conexiones densas entre capas, mejora la propagación de información.

**Ejemplos de bases de datos relevantes para clasificación:**

*MNIST* : Dígitos escritos a mano, 70,000 imágenes.
El el "Hello mundo" de los métodos de clasificación. https://www.kaggle.com/datasets/hojjatk/mnist-dataset

*CIFAR-10/CIFAR-100*: Imágenes a color (32x32), 10 y 100 clases.
Para reconocimiento de objetos. https://www.cs.toronto.edu/~kriz/cifar.html

*Fashion-MNIST*: 70,000 imágenes de ropa en 10 clases.
Para clasificación de artículos de moda, fue creada como un "nuevo MNIST" más complejo. https://www.kaggle.com/datasets/zalando-research/fashionmnist

*ImageNet*: Más de 1.4 millones de imágenes, 1,000 clases.
Se usa como benchmarking en tareas de clasificación a gran escala. https://www.image-net.org/download.php

- **Bibliografía**

Szeliski, Richard. *Computer vision: algorithms and applications*. Springer Nature, 2022.