

# Deep Learning

---

Sesión 4 - Resumen

# De un vistazo

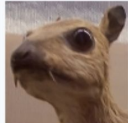
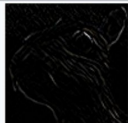

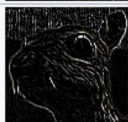
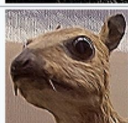
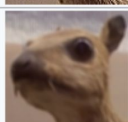
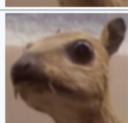
- Introducción a las redes convolucionales
- La convolución
- Redes Neuronales Convolucionales
- Tipos de capas
- Implementación de CNNs en Keras
- Preparación de los datos de entrada
- Overfitting y regularización

# Introducción a las redes convolucionales

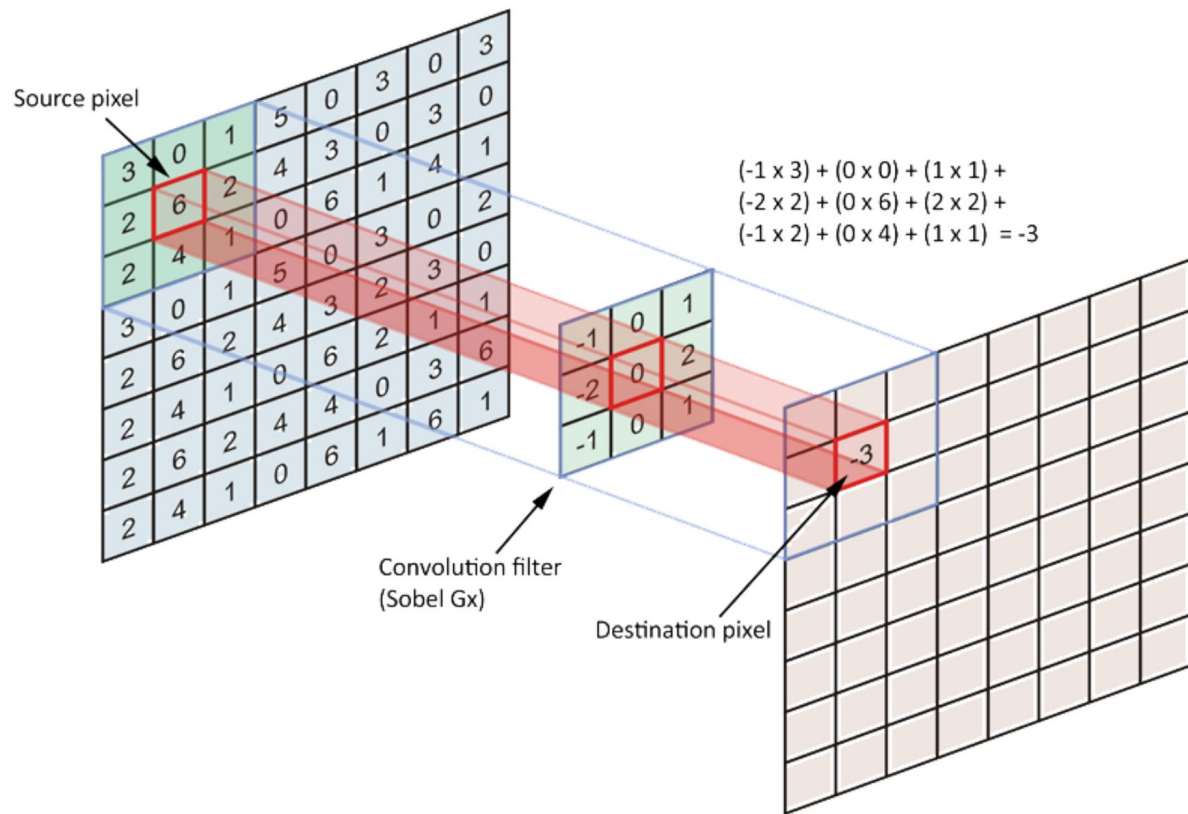
- Permiten trabajar con datos bidimensionales a la entrada: lo más común son las imágenes
- Ejemplos:
  - Reconocimiento facial (las agrupaciones de personas que hacen vuestros móviles automáticamente, por ejemplo)
  - Conducción autónoma
  - Ayuda al diagnóstico médico a partir de imagen médica
  - Y un largo etc.

# La convolución

- Operación matemática ampliamente utilizada en el tratamiento de la señal y de la imagen
- **Matemáticamente:** multiplicación de matrices y suma del resultado elemento a elemento, con **kernel** deslizante pixel a pixel
- **Conceptualmente:** filtrado que permite resaltar patrones existentes en las imágenes
- El **kernel** es el que define el tipo de patrón a resaltar. Por ejemplo: para resaltar bordes.

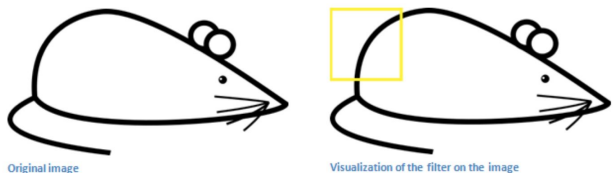
Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 x 3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

# La convolución



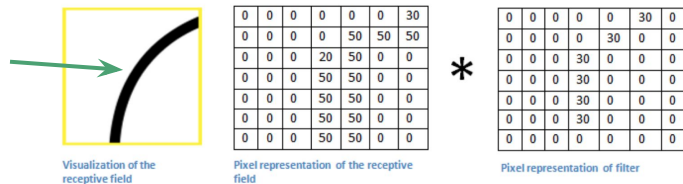
# La convolución

Nuestra imagen:



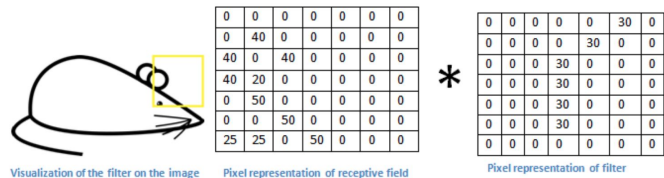
Éste es nuestro kernel. Va a resaltar donde existen líneas de esta forma en nuestra imagen

¿Qué pasa si nuestro filtro cae en la "espalda" de la rata?



Resultado =  $30 \cdot 0 + 30 \cdot 50 + 30 \cdot 20 + 30 \cdot 50 + 30 \cdot 50 + 30 \cdot 50 = 6600$  es un número muy alto!!

¿Qué pasa si nuestro filtro cae en la "cabeza" de la rata?

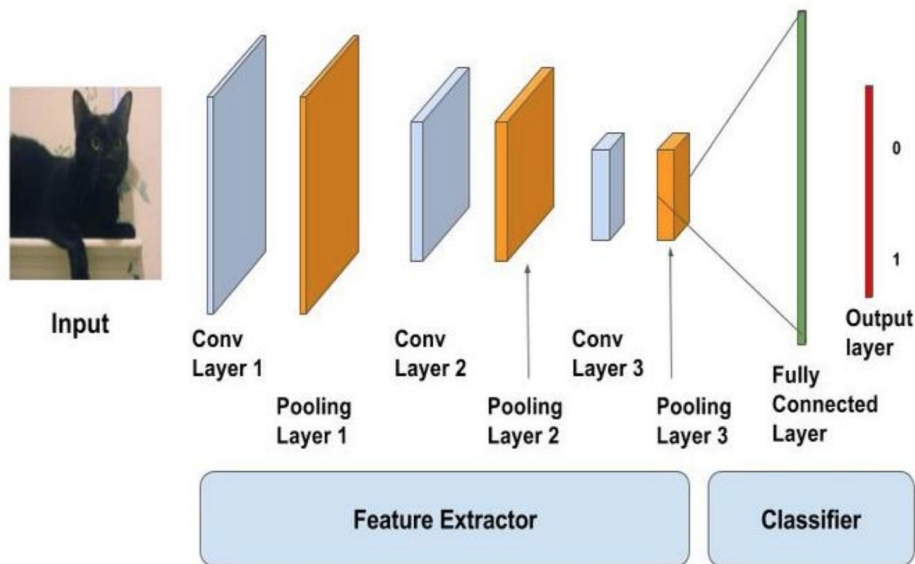


Resultado =  $30 \cdot 0 + 30 \cdot 0 + 30 \cdot 0 + 30 \cdot 0 + 30 \cdot 0 + 30 \cdot 0 = 0$  es un número muy bajo!!

El resultado será muy alto donde exista el patrón indicado por el kernel, y muy bajo donde no.

# Redes Neuronales Convolucionales

Son redes que incorporan capas de tipo **convolucionales**, las cuales permiten detectar patrones específicos y útiles para desarrollar la tarea requerida (clasificación, detección, regresión, etc)



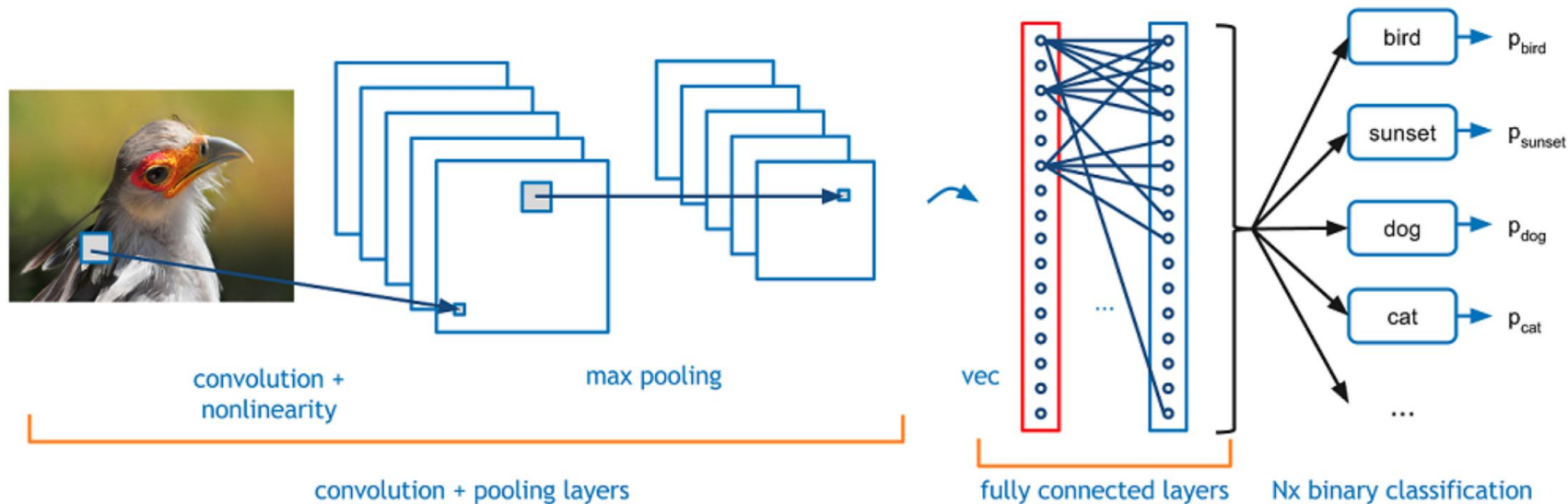
# Redes Neuronales Convolucionales

Las capas convolucionales se encargan de extraer las características más adecuadas para llevar a cabo la tarea requerida → **¡¡¡YA NO ES NECESARIO QUE NOS COMAMOS LA CABEZA ESCOGIENDOLAS NOSOTROS!!!**

El final de la red es un bloque que permite hacer uso de las características extraídas y clasificar la imagen de entrada de acuerdo a éstas en una clase u otra.

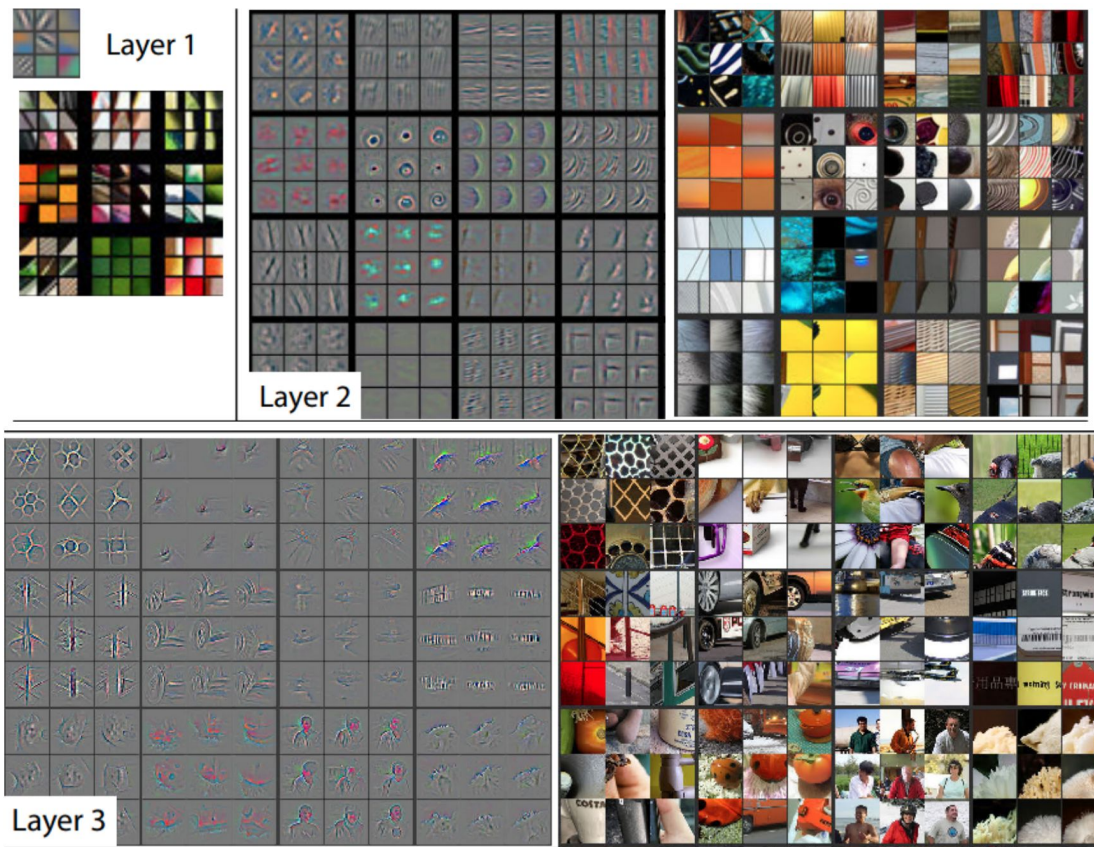


# Redes Neuronales Convolucionales



# Redes Neuronales Convolucionales

Las capas iniciales detectan patrones más sencillos, y conforme avanzamos en profundidad, se hacen más complejos (ya que utilizamos los patrones detectados anteriormente y sus combinaciones para ello)



# Tipos de capas

Las más comunes son:

- Las **convolucionales**: realizan la operación de la convolución (son las que realmente detectan los patrones)
- Las de **pooling**: reducen el tamaño de los **mapas de activación** (resultado tras una convolución) → eliminan información

# Implementación de CNNs en Keras

```
# Inicializamos el modelo
model = Sequential()

# Le añadimos las capas que queremos. Este primer bloque será nuestro extractor de características.
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))

# Añadimos nuestro clasificador
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compilamos el modelo
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001, decay=1e-6), metrics=['accuracy'])

# Entrenamos el modelo (validation_data debe ser el conjunto de validación, ni el de train ni el de test)
model.fit(X_train_norm, to_categorical(Y_train), batch_size=128, shuffle=True, epochs=10, validation_data=(X_val_norm, to_categorical(Y_val)))

# Evaluamos el modelo
scores = model.evaluate(X_test_norm, to_categorical(Y_test))

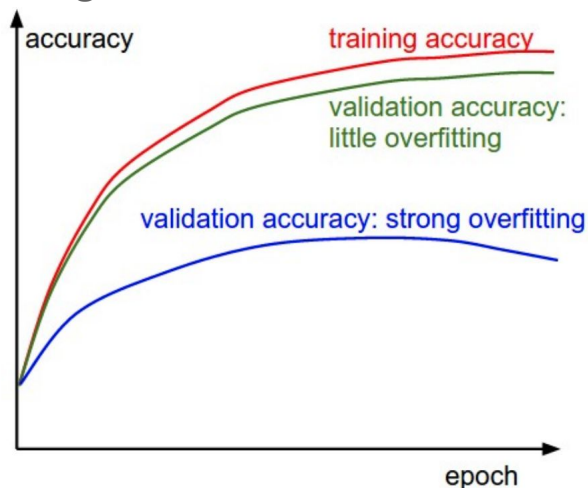
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

# Preparación de los datos de entrada

- Siempre debemos prestar atención a los rangos de los datos de entrada. Igual que en Machine Learning no deberíamos tener variables con rangos muy diferentes, lo mismo pasa en Deep Learning.
- Por lo tanto, lo mínimo es **normalizar** siempre las imágenes entre **0 y 1**.
- Otra opción más avanzada es:
  - Centrar los datos (calcular la media del dataset y restársela)
  - Normalizar los datos (dividir entre la desviación estándar y normalizar entre 0 y 1 o -1 y 1)
- Las normalizaciones que utilicen métricas del dataset (media, std, etc) son más peligrosas, pues **si el dataset no es lo suficientemente variado**, estaremos cometiendo un **sesgo**, ya que no representa la distribución real de las imágenes que nos encontraremos en el mundo real.

# Overfitting y regularización

**Overfitting:** ocurre cuando nuestro modelo “se aprende de memoria” los datos con los que es entrando, pero luego no funciona bien con nuevos datos



**Regularización:** “mecanismo de defensa” ante el overfitting

# Regularización

Métodos más comunes de regularización:

- Lasso (L2)
- Ridge (L1)
- Max-norm
- Dropout

Existen técnicas que, sin estar pensadas para ello, consiguen cierto nivel de regularización:

- Batch normalization
- Data augmentation