

07MIAR

Redes Neuronales y Deep Learning



**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

¿Quién soy?



viu

[in https://bit.ly/2DE3x1o](https://bit.ly/2DE3x1o)

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



<https://bit.ly/3icwDcH>

- **Trayectoria académica**

- Ingeniero de **Telecomunicación** por la Universidad Politécnica de Valencia (UPV). 2013.
- Máster en **Tecnologías Sistemas y Redes de Comunicaciones** por la UPV. 2014.
- Doctor en Ingeniería de **Telecomunicación** por la UPV, “*Fundus image analysis for automatic screening of ophthalmic pathologies*”. 2018.

- **Trayectoria profesional**

- Coordinador científico en el grupo de investigación **CVBLab**.
- Profesor colaborador en el **Grado de Ingeniería Biomédica** de la UPV. Asignatura **Imágenes Biomédicas**.
- Director académico curso “**Deep Learning** aplicado al análisis de **señales e imágenes**” en el CFP de la UPV.
- Consultor de Innovación de Programa en VIU. **Director del máster en inteligencia artificial**. Docencia 07MIAR y 12MBID.

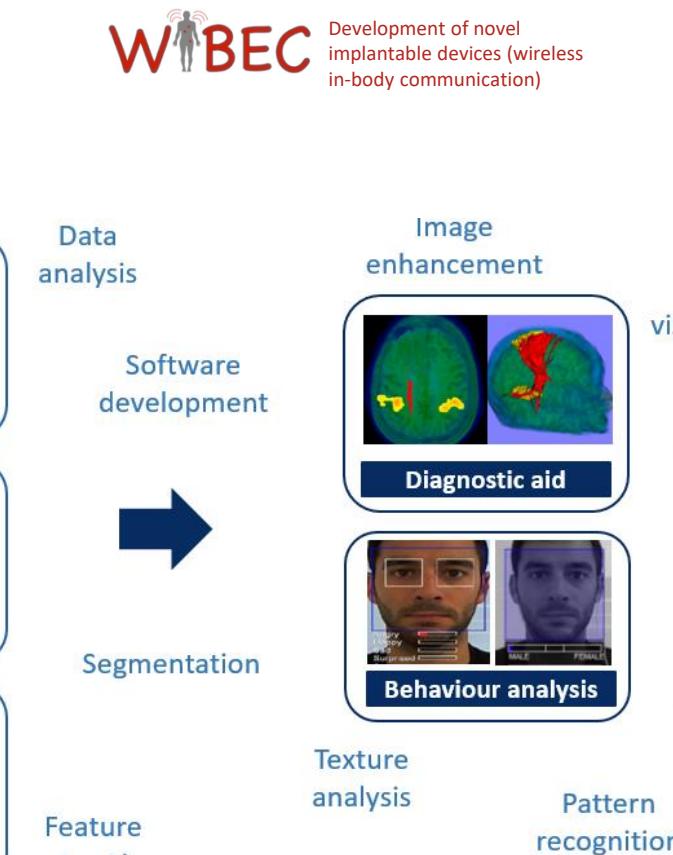
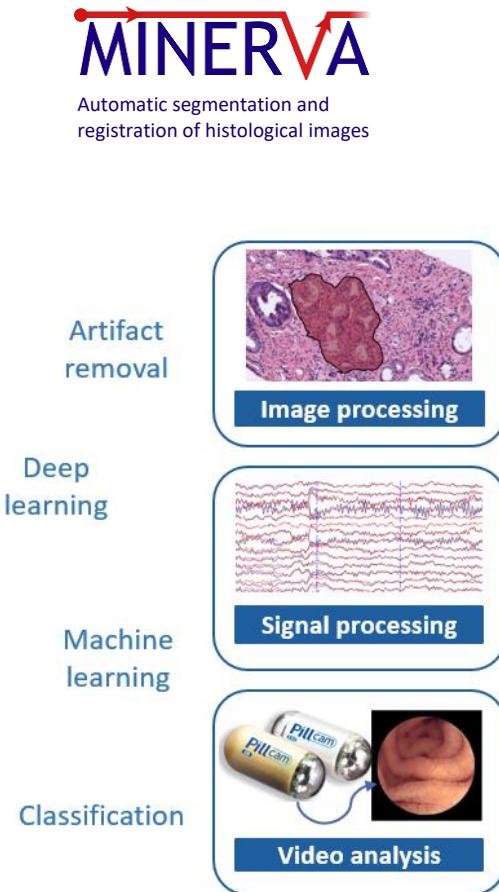
¿Qué hago?



Development of a software for hepatic planning



Development of multimodal interface customizable for people with disabilities



Human aircraft roadmap for virtual intelligent system



Development of tools for early diagnosis of neurodegenerative diseases



Histopathological image interpretation for prostate cancer detection

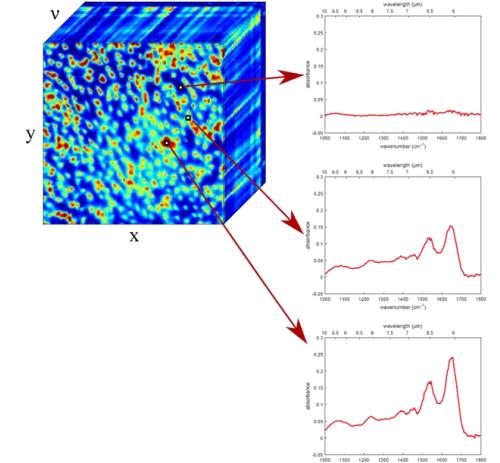
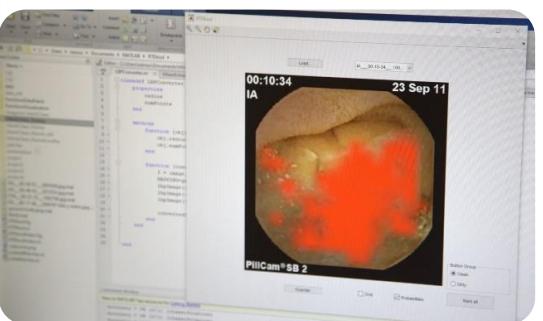
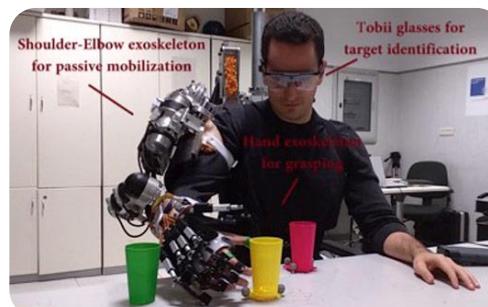
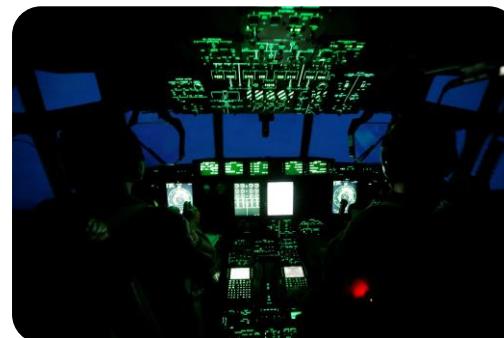
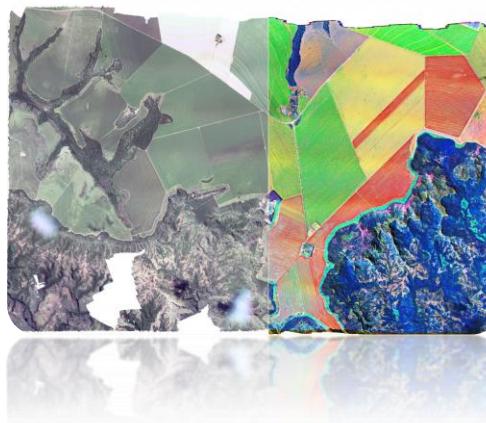
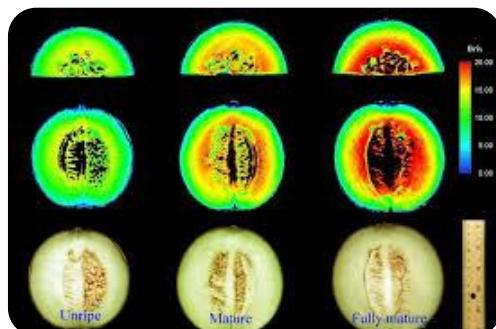
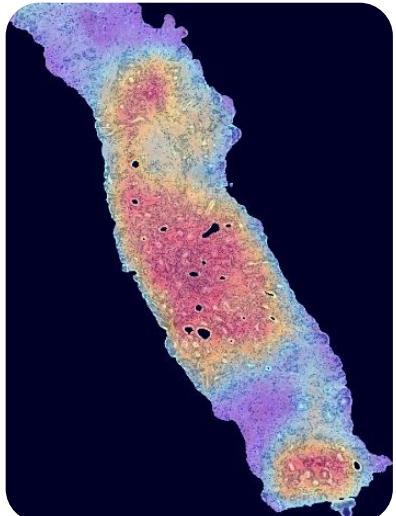


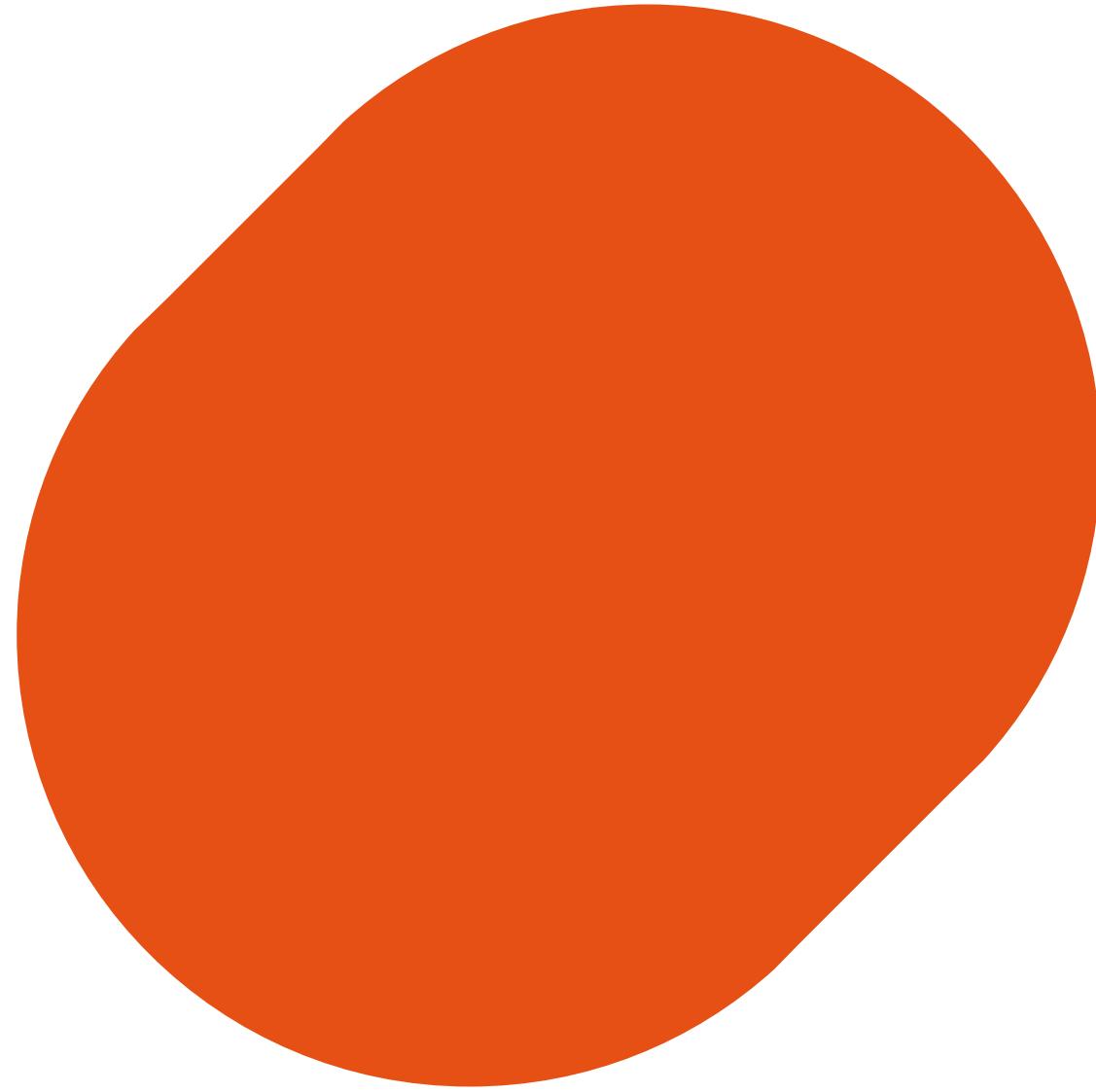
Augmented reality system for trocar placement assistance in laparoscopic surgery



Cloud Artificial Intelligence for Pathology

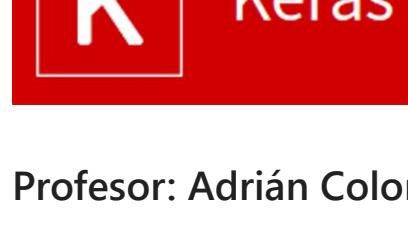
¿Qué hago?





07MAIR - Redes Neuronales y Deep Learning

Clase 01a: Presentación de la asignatura



Profesor: Adrián Colomer Granero

Sumario

- Estructura de la asignatura
- Contenidos y objetivos
- Evaluación

Contenidos

- Fundamentos de Redes neuronales: Perceptrón simple y perceptrón multicapa, Descenso por gradiente en redes neuronales, Algoritmo de backpropagation.
- Deep learning: Descripción de tipos de capas y su aplicabilidad, Ejemplos de arquitecturas, Regularización, Optimización de hiperparámetros.
- Aplicación de las Redes Neuronales y Deep Learning a la resolución de tareas de IA: Clasificación de imágenes, detección de objetos y segmentación (Redes Neuronales Convolucionales), Texto y secuencias (Redes Neuronales Recurrentes con unidades LSTM y GRU), Introducción a Keras y TensorFlow.
- Aprendizaje generativo: Autoencoders, Autoencoders variacionales y Generative Adversarial Networks (GANs).
- Deep Learning en producción: Implementación y gestión del ciclo de vida de modelos basados en aprendizaje profundo. Introducción al paquete MLflow.

Uso de Jupyter/Colab notebooks

- Presentaciones, ejercicios, actividades prácticas **la mayoría** en Notebooks (Jupyter/Colab)
- Facilita la colaboración y la presentación interactiva de código
- En la terminal: **jupyter notebook**
- Archivos `.ipynb`
- Documentación <https://jupyter.readthedocs.io/en/latest/>
- Guía inicio Colab: <https://colab.research.google.com/notebooks/intro.ipynb#>

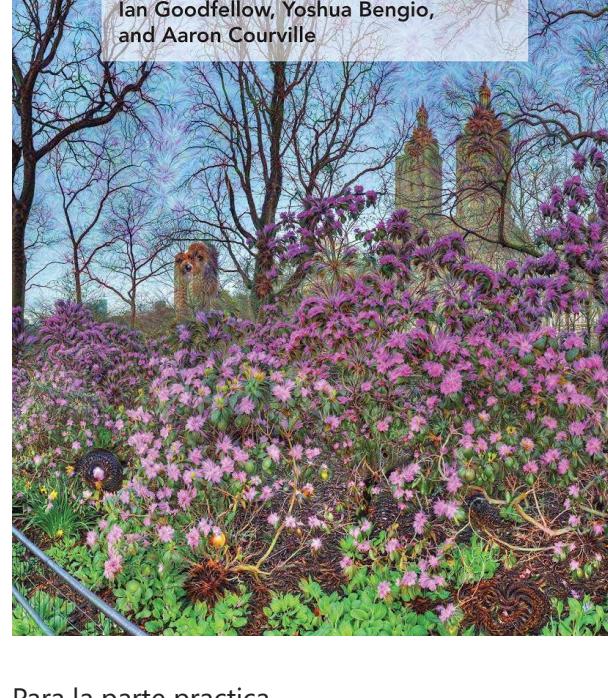
Recursos de la asignatura

- En el aula virtual
 - Manual de la asignatura, e-learning y vídeos
 - Jupyter/Colab notebooks de las clases
 - Papers, código adicional, etc.
 - Actividades (descripción y entregas)

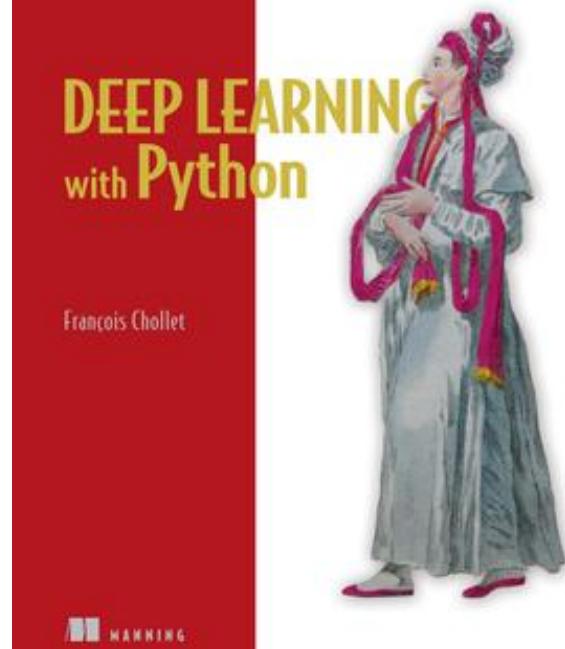
- Documentación oficial
 - TensorFlow 1.x/2.x: https://www.tensorflow.org/api_docs
 - Keras: <https://keras.io/>

- Bibliografía recomendada

Ian Goodfellow, padre de las redes GANs

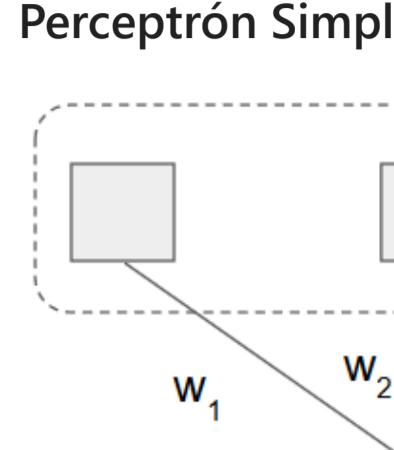


Para la parte práctica



07MAIR - Redes Neuronales y Deep Learning

Clase 01b: Introducción a Redes Neuronales

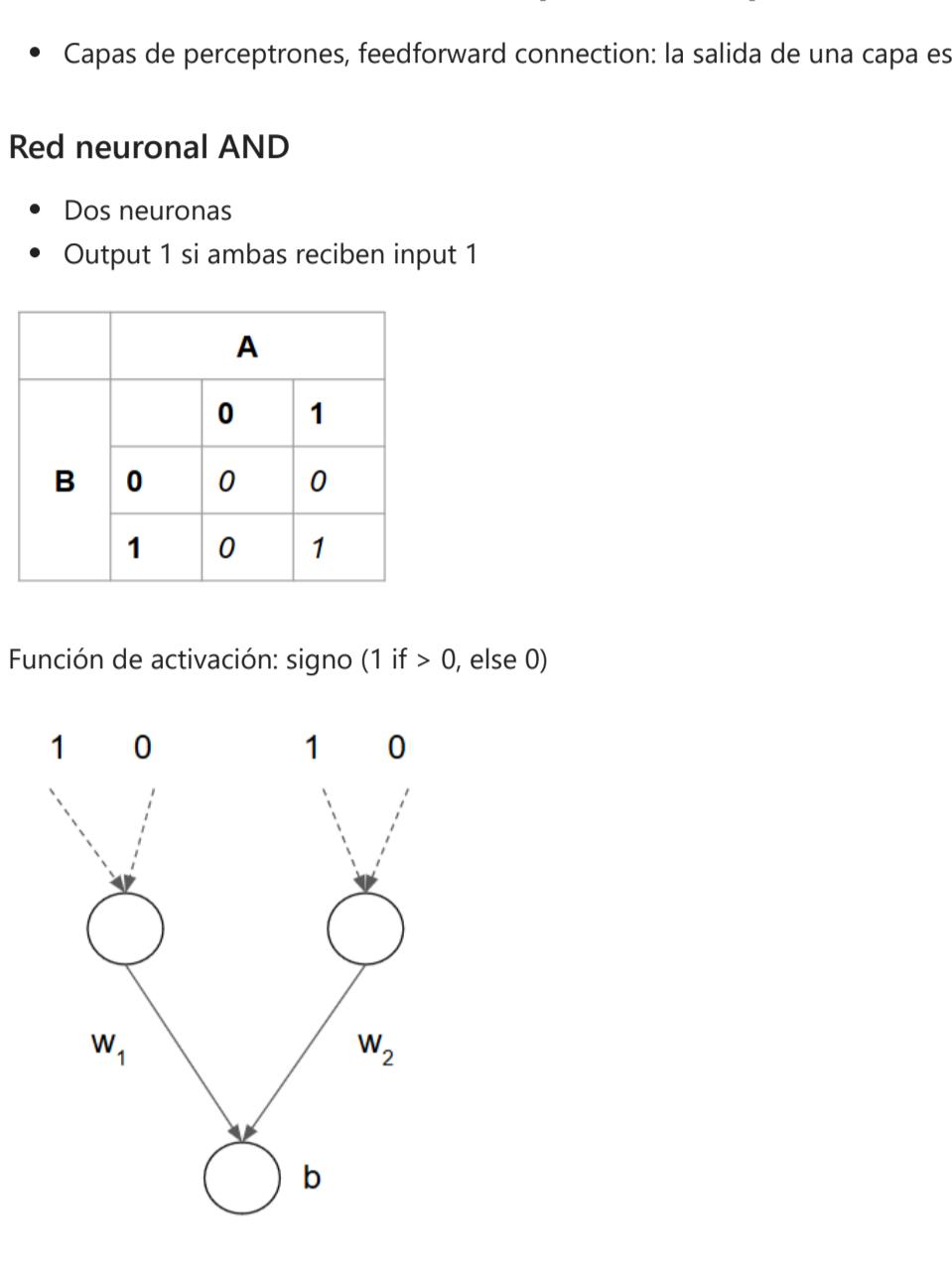


Profesor: Adrián Colomer Granero

Sumario

- Perceptrón
- Perceptrón multicapa
- Ejemplo redes AND y XOR
- Entrenamiento con Backpropagation

Perceptrón Simple



- Propuesto por Rosenblatt 1957
 - No fue popular porque tecnologicamente no era viable entrenar redes neuronales en un tiempo permisible.
 - Se compone de una combinación lineal de entradas ponderadas

Parámetros:

- Entrenables, determinan el valor output de la neurona (**input*weight + bias**):
 - Bias por cada neurona
 - Weight por cada conexión
- A elegir:
 - Función de activación (cómo se calcula el output en función del input) -> Hiperparámetro

En la primera iteración del entrenamiento, tendrán valores aleatorios, y cercanos a ceros (para facilitar la convergencia de la red). Conforme sigamos entrenando, los valores se irán ajustando para minimizar cierta función objetivo/error.

Redes neuronales (Perceptrón simple)

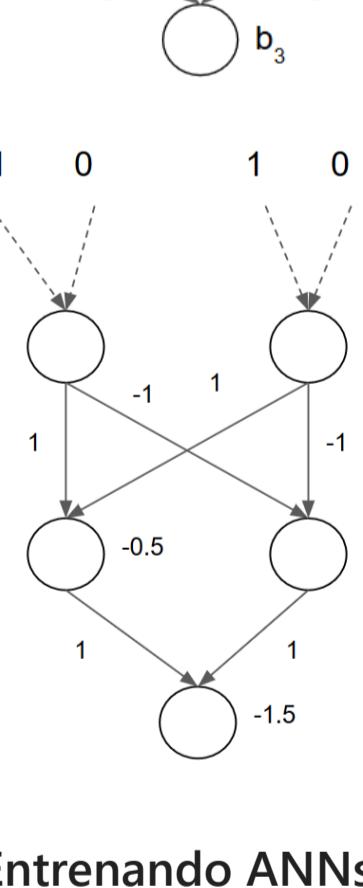
- Capas de perceptrones, feedforward connection: la salida de una capa es la entrada de la siguiente

Red neuronal AND

- Dos neuronas
- Output 1 si ambas reciben input 1

		A	
		0	1
B	0	0	0
	1	0	1

Función de activación: signo (1 if > 0, else 0)

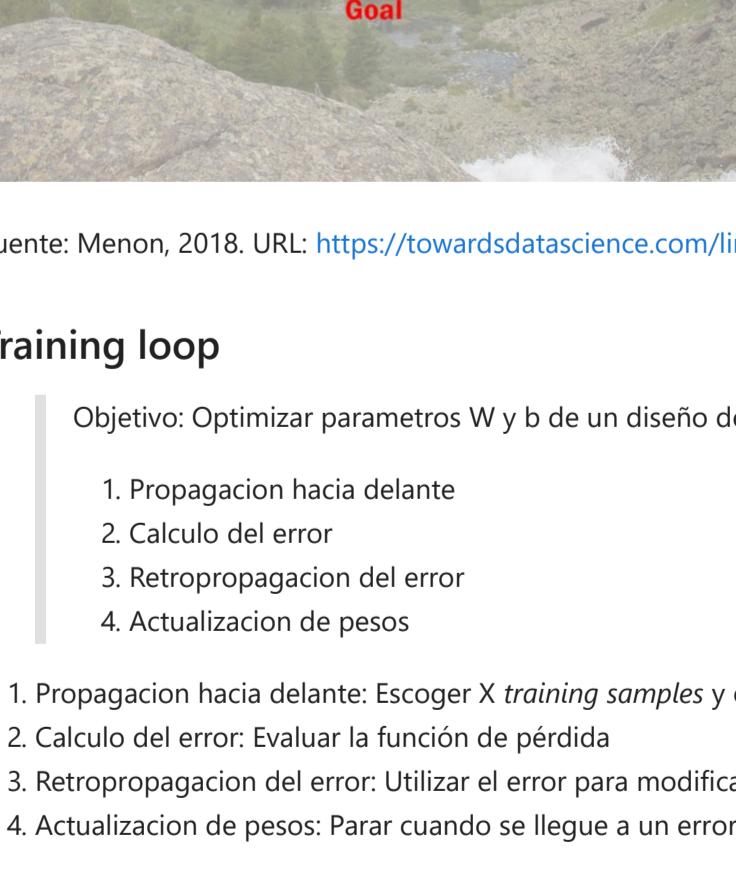


W1 = 1, W2 = 1

b = -1

$$\begin{aligned} 0 + 0 = 0 \quad 0 - 1 = -1 \quad 0 > 0 \rightarrow 0 \\ 0 + 1 = 1 \quad 1 - 1 = 0 \quad 0 > 0 \rightarrow 0 \\ 1 + 1 = 2 \quad 2 - 1 = 1 \quad 1 > 0 \rightarrow 1 \end{aligned}$$

Función de activación: signo (1 if > 0, else 0)



Al añadir capas ocultas, podemos aplicar funciones de activación no lineales, permitiéndonos así separar dataset no linealmente separables.

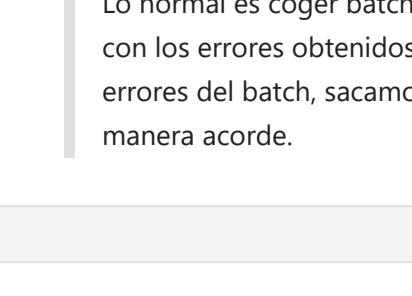
(Similar al kernel trick en SVM para expandir de 2D a 3D)

Cada conexión entre neurona está definida por un peso.

Las activations que se darán en las hidden layers, vendrán causadas por la multiplicación de las entradas por el peso sumado al bias, y se aplicará una función de activación. Al computar eso, se obtendrán unas activations en la capa intermedia (a1)

Ejercicio

- Averiguar los valores de W1,...,W6 y b1,...,b3 adecuados para XOR



Dado el error de salida de la red, se retropropaga hacia detrás hasta la capa de entrada, para encontrar qué parámetros w y b influyen más para haber cometido dicho error, y se modificarán de manera acorde para minimizar el error a la salida

Fuente: Menon, 2018. URL: <https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931>

Training loop

Objetivo: Optimizar parámetros W y b de un diseño de topología de red

1. Propagación hacia delante
2. Cálculo del error
3. Retropropagación del error
4. Actualización de pesos

1. Propagación hacia delante: Escoger X *training samples* y obtener predicciones

2. Cálculo del error: Evaluar la función de pérdida

3. Retropropagación del error: Utilizar el error para modificar los parámetros W y b para minimizar el error

4. Actualización de pesos: Parar cuando se llegue a un error aceptable, o tras un número de iteraciones (épocas)

Estrategias para actualizar los valores

- **Naive**: modificar uno a uno cada parámetro y ver si disminuye o aumenta el error. Ineficiente

- **Diferenciar la capa con respecto a cada parámetro**.
 - Se puede computar el gradiente de la pérdida (derivada) y modificar el parámetro en la dirección contraria

- **La diferenciación de las primeras capas depende de las últimas**. Usando la **regla de la cadena** de cálculo, $f(g(x)) = f'(g(x)) * g'(x)$, el cálculo del gradiente comienza al final de la NN y va hacia atrás (backpropagation)

Esta es la que se implementa en la práctica

Típos de descenso del gradiente

Dependiendo del tamaño del dataset:

- Stochastic gradient descent: batch size 1 Se necesita demasiado de memoria, probablemente se desborde la memoria al guardar tantos errores.

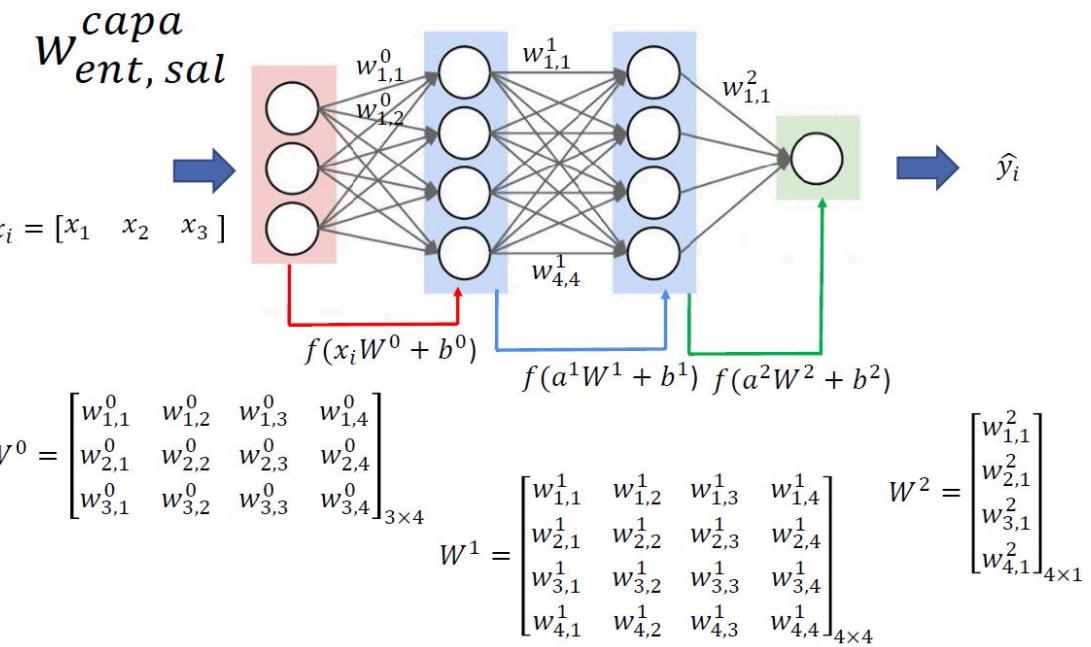
Solo se actualizan los pesos una vez, ya que nos esperamos a obtener el promedio de todos los datasets. Hatan falta muchas épocas (iteración que completa todo mi conjunto de datos, cuando el conjunto de datos pasa por toda la red) para entrenar la red

- Mini-batch SGD: batch size > 1 < tamaño del training set

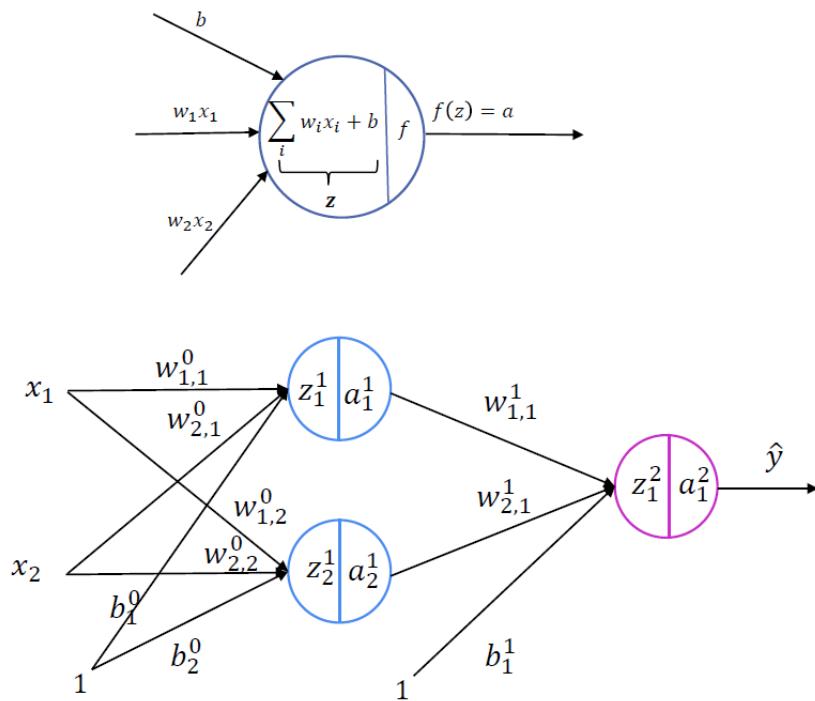
Si de repente llega un outlier y retropropagamos los pesos con ese error, se desestabilizaría o desajustaría la red. Para reducir este efecto, podríamos medir los errores de varios ejemplos, aplicar batches, agrupar, ...

Lo normal es coger batches de tamaño menor al grupo de las muestras, pero mayor que la unidad. Nos vamos quedando con los errores obtenidos por varios ejemplos (p ej batch de 16 ejemplos), y una vez cuando hemos obtenido todos los errores del batch, sacamos una promedio de dichas muestras, y entonces lo retropropagamos y actualizamos los pesos de manera acorde.

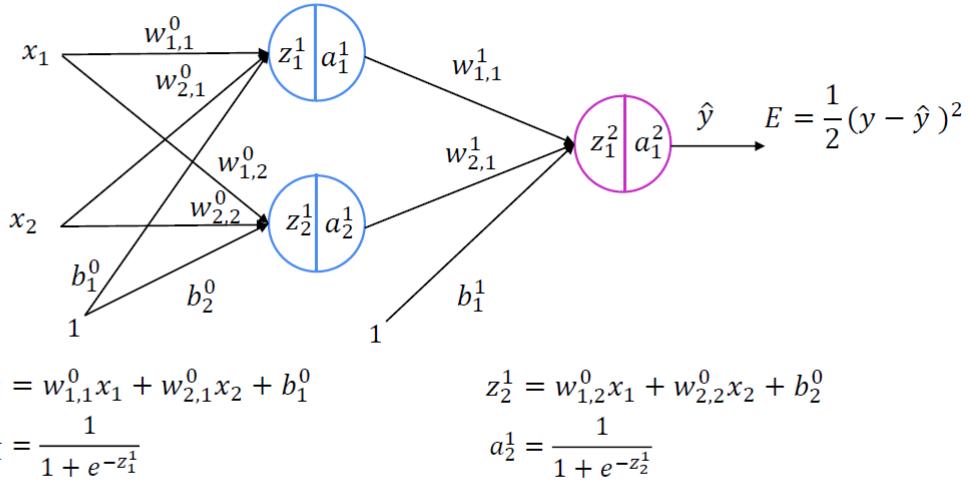
Estructura de una red neuronal



Forward and Backward propagation



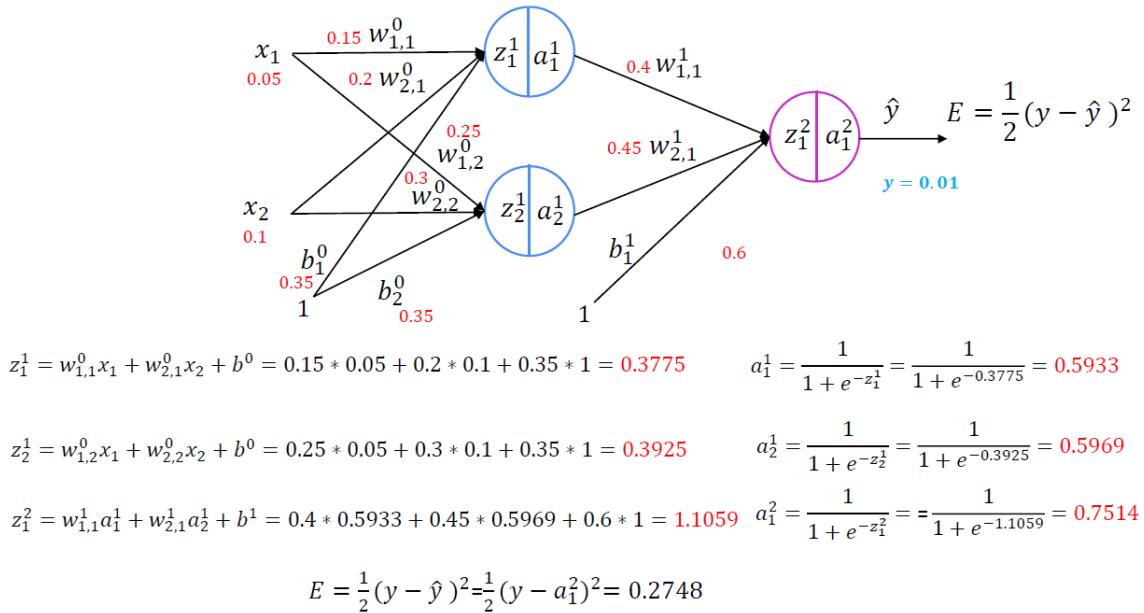
Forward



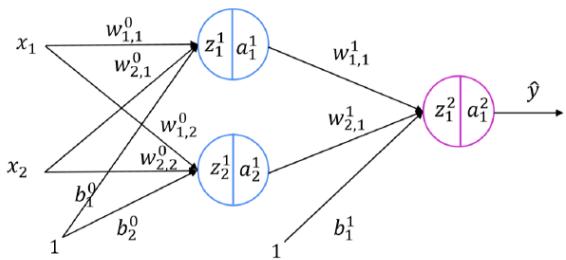
$$z_1^2 = w_{1,1}^1 a_1^1 + w_{2,1}^1 a_2^1 + b_1^1$$

$$a_1^2 = \frac{1}{1 + e^{-z_1^2}} = \hat{y}$$

Forward propagation



Backpropagation



1. Actualización de pesos etapa 1

$$w_{1,1}^1(t+1) = w_{1,1}^1(t) - \eta \frac{\partial E}{\partial w_{1,1}^1}$$

$$w_{2,1}^1(t+1) = w_{2,1}^1(t) - \eta \frac{\partial E}{\partial w_{2,1}^1}$$

1. Actualización de pesos etapa 0

$$w_{1,1}^0(t+1) = w_{1,1}^0(t) - \eta \frac{\partial E}{\partial w_{1,1}^0}$$

$$w_{2,1}^0(t+1) = w_{2,1}^0(t) - \eta \frac{\partial E}{\partial w_{2,1}^0}$$

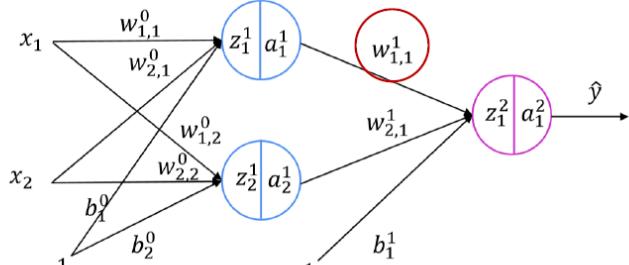
$$w_{1,2}^0(t+1) = w_{1,2}^0(t) - \eta \frac{\partial E}{\partial w_{1,2}^0}$$

$$w_{2,2}^0(t+1) = w_{2,2}^0(t) - \eta \frac{\partial E}{\partial w_{2,2}^0}$$

Backpropagation

$$w_{1,1}^1(t+1) = w_{1,1}^1(t) - \eta \frac{\partial E}{\partial w_{1,1}^1}$$

$$\boxed{\frac{\partial E}{\partial w_{1,1}^1}}$$



$$E = \frac{1}{2} (y - \hat{y})^2 = \frac{1}{2} (y - a_1^2)^2$$

$$\frac{\partial E}{\partial w_{1,1}^1} \longrightarrow a_1^2 = \frac{1}{1 + e^{-z_1^2}} \longrightarrow z_1^2 = w_{1,1}^1 a_1^1 + w_{2,1}^1 a_2^1 + b_1^1$$



Regla de la cadena

$$\boxed{\frac{\partial E}{\partial w_{1,1}^1} = \frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial w_{1,1}^1}}$$

Backpropagation

$$\frac{\partial E}{\partial w_{1,1}^1} = \frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial w_{1,1}^1}$$

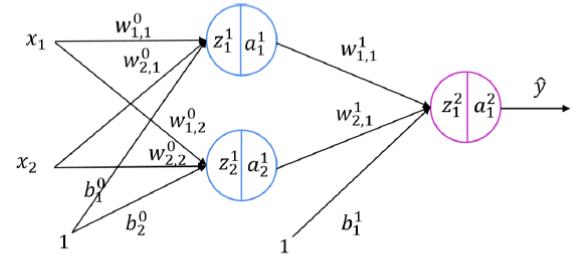
$$\frac{\partial E}{\partial a_1^2} \quad E = \frac{1}{2}(y - a_1^2)^2$$

$$\frac{\partial E}{\partial a_1^2} = \frac{1}{2} * 2 * (-1)(y - a_1^2) = (a_1^2 - y)$$

$$\frac{\partial a_1^2}{\partial z_1^2} \quad a_1^2 = \frac{1}{1 + e^{-z_1^2}} \quad \frac{\partial a_1^2}{\partial z_1^2} = \frac{-e^{-z_1^2}}{(1 + e^{-z_1^2})^2} = a_1^2(1 - a_1^2)$$

$$\frac{\partial z_1^2}{\partial w_{1,1}^1} \quad z_1^2 = w_{1,1}^1 a_1^1 + w_{2,1}^1 a_2^1 + b_1^1 \quad \frac{\partial z_1^2}{\partial w_{1,1}^1} = a_1^1$$

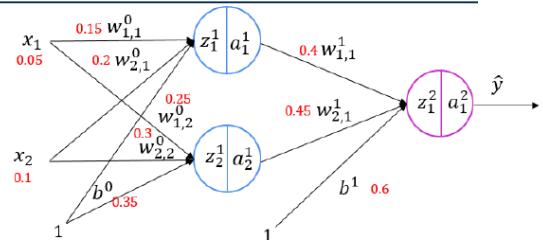
$$\frac{\partial E}{\partial w_{1,1}^1} = (a_1^2 - y) * a_1^2(1 - a_1^2) * a_1^1$$



Backpropagation

$$\frac{\partial E}{\partial w_{1,1}^1} = \frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial w_{1,1}^1}$$

$$\delta_{a_1^2} = \frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2}$$



$$\delta_{a_1^2} = (a_1^2 - y) * a_1^2(1 - a_1^2)$$

$$\frac{\partial E}{\partial w_{1,1}^1} = \delta_{a_1^2} * a_1^1$$

$$\delta_{a_1^2} = (0.7514 - 0.01) * 0.7514 * (1 - 0.7514) = 0.1385$$

$$\frac{\partial E}{\partial w_{1,1}^1} = 0.1385 * 0.5933 = 0.0822$$

Cuánto un cambio en $w_{1,1}^1$ afecta a E

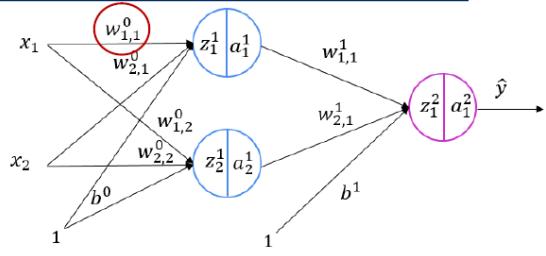
$$\text{supongamos} \quad \eta = 0.5 \quad w_{1,1}^1(t+1) = w_{1,1}^1(t) - \eta \frac{\partial E}{\partial w_{1,1}^1} = 0.4 - 0.5 * 0.082 = 0.3590$$

Backpropagation

$$w_{1,1}^0(t+1) = w_{1,1}^0(t) - \eta \frac{\partial E}{\partial w_{1,1}^0}$$

$$\boxed{\frac{\partial E}{\partial w_{1,1}^0}}$$

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - a_1^2)^2$$



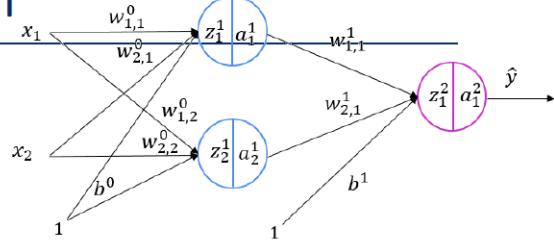
$$\begin{aligned} \frac{\partial E}{\partial w_{1,1}^0} &\longrightarrow a_1^2 = \frac{1}{1 + e^{-z_1^2}} \longrightarrow z_1^2 = w_{1,1}^1 a_1^1 + w_{2,1}^1 a_2^1 + b^1 \longrightarrow a_1^1 = \frac{1}{1 + e^{-z_1^1}} \\ &z_1^1 = w_{1,1}^0 x_1 + w_{2,1}^0 x_2 + b^0 \end{aligned}$$

$$\boxed{\frac{\partial E}{\partial w_{1,1}^0} = \frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1} * \frac{\partial z_1^1}{\partial w_{1,1}^0}}$$

Backpropagation

$$\frac{\partial E}{\partial w_{1,1}^0} = \boxed{\frac{\partial E}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2} * \frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1} * \frac{\partial z_1^1}{\partial w_{1,1}^0}}$$

$\delta_{a_1^2}$



$$\frac{\partial z_1^2}{\partial a_1^1} \quad z_1^2 = w_{1,1}^1 a_1^1 + w_{2,1}^1 a_2^1 + b^1 \quad \frac{\partial z_1^2}{\partial a_1^1} = w_{1,1}^1$$

$$\frac{\partial a_1^1}{\partial z_1^1} \quad a_1^1 = \frac{1}{1 + e^{-z_1^1}} \quad \frac{\partial a_1^1}{\partial z_1^1} = a_1^1 * (1 - a_1^1)$$

$$\frac{\partial z_1^1}{\partial w_{1,1}^0} \quad z_1^1 = w_{1,1}^0 x_1 + w_{2,1}^0 x_2 + b^0 \quad \frac{\partial z_1^1}{\partial w_{1,1}^0} = x_1$$

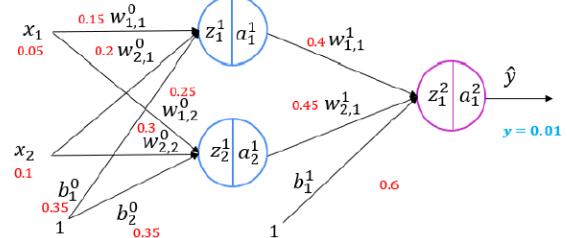
$$\frac{\partial E}{\partial w_{1,1}^0} = \delta_{a_1^2} * \frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1} * \frac{\partial z_1^1}{\partial w_{1,1}^0} = \delta_{a_1^2} * w_{1,1}^1 * a_1^1 * (1 - a_1^1) * x_1$$

Backpropagation

$$\frac{\partial E}{\partial w_{1,1}^0} = \delta_{a_1^2} * w_{1,1}^1 * a_1^1 * (1 - a_1^1) * x_1$$

$$\delta_{a_1^1} = \delta_{a_1^2} * w_{1,1}^1 * a_1^1 * (1 - a_1^1)$$

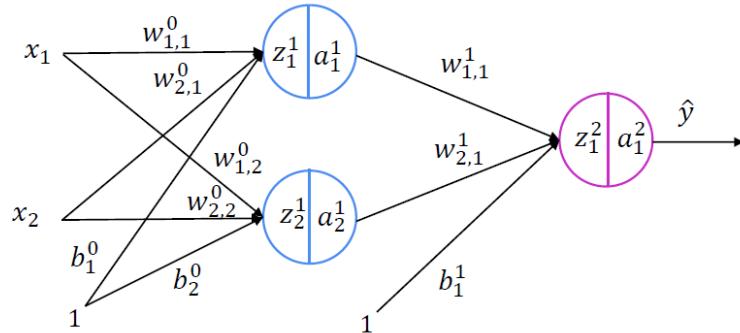
$$\delta_{a_1^1} = 0.1385 * 0.0965$$



$$\frac{\partial E}{\partial w_{1,1}^0} = \delta_{a_1^2} * \delta_{a_1^1} * x_1 = 0.1385 * 0.0965 * 0.05 = 6.6835e-04$$

supongamos $\eta = 0.5$ $w_{1,1}^0(t+1) = w_{1,1}^0(t) - \eta \frac{\partial E}{\partial w_{1,1}^0} = 0.15 - 0.5 * 6.6835e-04 = 0.1497$

Backpropagation



Matricialmente

$$\text{Capas} \quad l: 0 \dots L \quad A^0 = [x_1 \quad x_2 \quad 1] \quad Z^1 = [z_1^1 \quad z_2^1] \quad A^1 = [a_1^1 \quad a_2^1 \quad 1]$$

$$W^0 = \begin{bmatrix} w_{1,1}^0 & w_{1,2}^0 \\ w_{2,1}^0 & w_{2,2}^0 \\ b_1^0 & b_2^0 \end{bmatrix} \quad W^1 = \begin{bmatrix} w_{1,1}^1 \\ w_{2,1}^1 \\ b_1^1 \end{bmatrix}$$

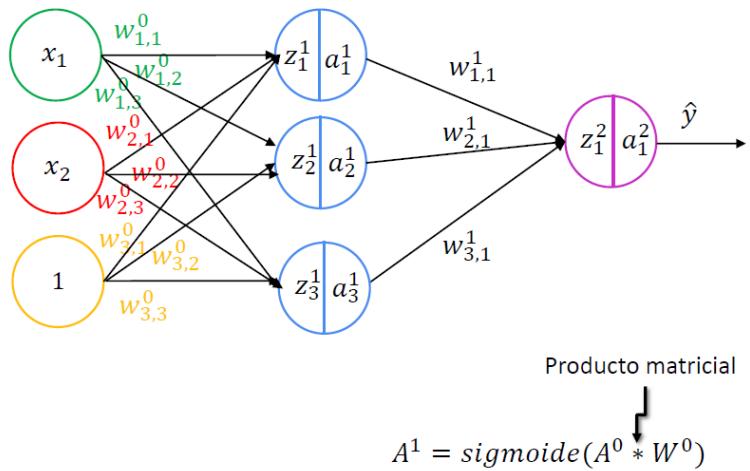
Forward propagation

Matricialmente

Capas $l: 0 \dots L$

$$W^0 = \begin{bmatrix} w_{1,1}^0 & w_{1,2}^0 & w_{1,3}^0 \\ w_{2,1}^0 & w_{2,2}^0 & w_{2,3}^0 \\ w_{3,1}^0 & w_{3,2}^0 & w_{3,3}^0 \end{bmatrix}$$

$$W^1 = \begin{bmatrix} w_{1,1}^1 \\ w_{2,1}^1 \\ w_{3,1}^1 \end{bmatrix}$$



$$A^0 = [x_1 \quad x_2 \quad 1] \quad A^1 = [a_1^1 \quad a_2^1 \quad a_3^1]$$

$$A^2 = \text{sigmoide}(A^1 * W^1)$$

$$A^2 = [a_1^2]$$

Backpropagation

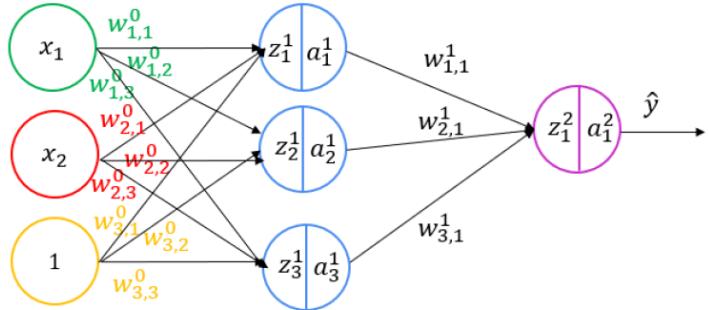
Capa 2

$$w_{1,1}^1(t+1) = w_{1,1}^1(t) - \eta \delta_{a_1^2} a_1^1$$

$$w_{2,1}^1(t+1) = w_{2,1}^1(t) - \eta \delta_{a_1^2} a_2^1$$

$$w_{3,1}^1(t+1) = w_{3,1}^1(t) - \eta \delta_{a_1^2} a_3^1$$

$$\delta_{a_1^2} = (a_1^2 - y) * a_1^2(1 - a_1^2)$$



Capa 1

$$w_{1,1}^0(t+1) = w_{1,1}^0(t) - \eta \delta_{a_1^1} x_1 \quad | \quad w_{1,2}^0(t+1) = w_{1,2}^0(t) - \eta \delta_{a_2^1} x_1 \quad | \quad w_{1,3}^0(t+1) = w_{1,3}^0(t) - \eta \delta_{a_3^1} x_1$$

$$w_{2,1}^0(t+1) = w_{2,1}^0(t) - \eta \delta_{a_1^1} x_2 \quad | \quad w_{2,2}^0(t+1) = w_{2,2}^0(t) - \eta \delta_{a_2^1} x_2 \quad | \quad w_{2,3}^0(t+1) = w_{2,3}^0(t) - \eta \delta_{a_3^1} x_2$$

$$w_{3,1}^0(t+1) = w_{3,1}^0(t) - \eta \delta_{a_1^1} \quad | \quad w_{3,2}^0(t+1) = w_{3,2}^0(t) - \eta \delta_{a_2^1} \quad | \quad w_{3,3}^0(t+1) = w_{3,3}^0(t) - \eta \delta_{a_3^1}$$

$$\delta_{a_1^1} = \delta_{a_1^2} \cdot w_{1,1}^1 \cdot a_1^1 \cdot (1 - a_1^1) \quad | \quad \delta_{a_2^1} = \delta_{a_1^2} \cdot w_{2,1}^1 \cdot a_2^1 \cdot (1 - a_2^1) \quad | \quad \delta_{a_3^1} = \delta_{a_1^2} \cdot w_{3,1}^1 \cdot a_3^1 \cdot (1 - a_3^1)$$

Backpropagation

Matricialmente

$$\text{Capas} \quad l: 0 \dots L \quad W^0 = \begin{bmatrix} w_{1,1}^0 & w_{1,2}^0 & w_{1,3}^0 \\ w_{2,1}^0 & w_{2,2}^0 & w_{2,3}^0 \\ w_{3,1}^0 & w_{3,2}^0 & w_{3,3}^0 \end{bmatrix} \quad W^1 = \begin{bmatrix} w_{1,1}^1 \\ w_{2,1}^1 \\ w_{3,1}^1 \end{bmatrix} \quad A^0 = [x_1 \quad x_2 \quad 1]$$

$$A^1 = [a_1^1 \quad a_2^1 \quad a_3^1] \quad A^2 = [a_1^2] \quad \text{Capa 1}$$

$$\delta_{a_1^1} = \delta_{a_1^2} \cdot w_{1,1}^1 \cdot a_1^1 \cdot (1 - a_1^1)$$

$$\delta_{a_2^1} = \delta_{a_1^2} \cdot w_{2,1}^1 \cdot a_2^1 \cdot (1 - a_2^1)$$

Capa 2

$$\delta_{a_1^2} = (a_1^2 - y) * a_1^2(1 - a_1^2)$$

$$\Delta^2 = [\delta_{a_1^2}] = (a_1^2 - y) * (A^2)'$$

$$W^1(t+1) = W^1(t) - \eta (A^1)^T * \Delta^2$$

$$\delta_{a_3^1} = \delta_{a_1^2} \cdot w_{3,1}^1 \cdot a_3^1 \cdot (1 - a_3^1)$$

$$\Delta^1 = [\delta_{a_1^1} \quad \delta_{a_2^1} \quad \delta_{a_3^1}] =$$

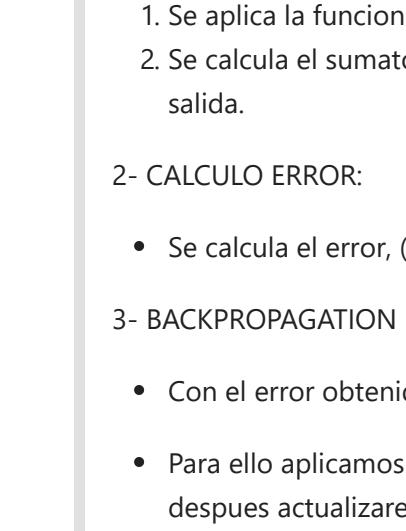
$$= \Delta^2 * \begin{bmatrix} w_{1,1}^1 \\ w_{2,1}^1 \\ w_{3,1}^1 \end{bmatrix}^T \cdot [(a_1^1)' \quad (a_2^1)' \quad (a_3^1)']$$

$$\Delta^1 = \Delta^2 * (W^1)^T \quad \Delta^1 = \Delta^1 \cdot (A^1)'$$

$$W^0(t+1) = W^0(t) - \eta (A^0)^T * \Delta^1$$

07MAIR - Redes Neuronales y Deep Learning

Clase 02: Redes neuronales artificiales



Profesores: Adrián Colomer Granero / Gabriel Enrique Muñoz Ríos

Autor: Carlos Fernández Musoles

VC02_VC03_Introducción a Deep Learning_Jupyter

Sumario

- Funciones de activación
- Visualización con Tensorflow playground
- Ejemplo mnist keras
- Optimización

PDF FORWARD

Forward: se calcula el sumatorio de las funciones de entrada y los pesos con el bias.

1- FORWARD:

1. Se aplica la función de activación (p ej sigmoid)
2. Se calcula el sumatorio de las funciones de activaciones, con el bias. 3. Se aplica la función de activación de la capa de salida.

2- CALCULO ERROR:

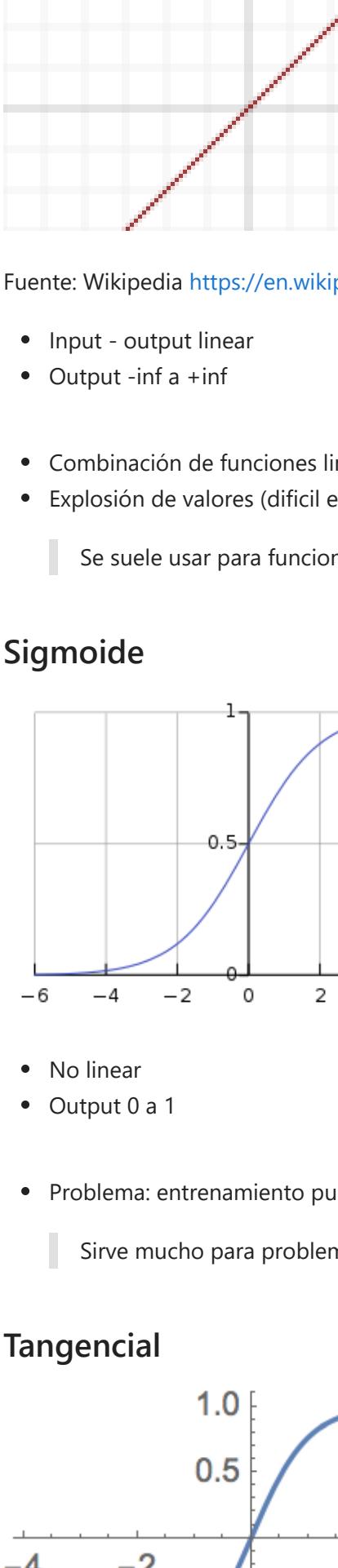
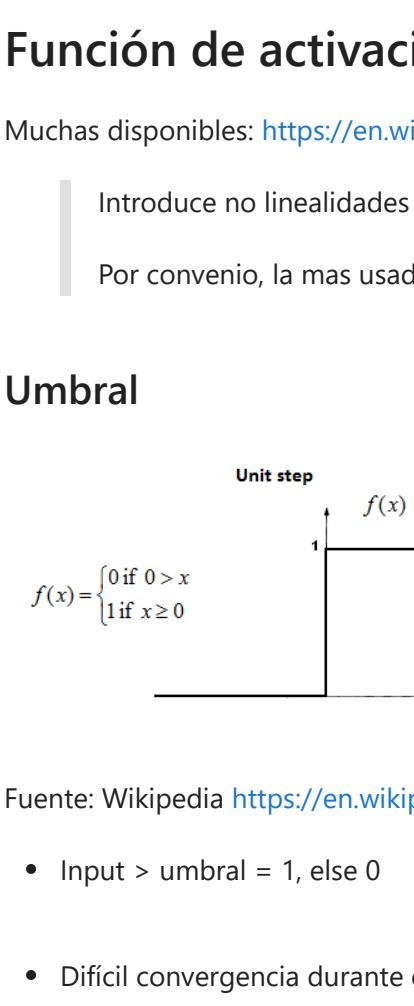
- Se calcula el error, (p ej. MSE) y se hace retropropagación. FORWARD PROPAGATION

3- BACKPROPAGATION

- Con el error obtenido, hay que ver como influyen los parámetros w y b en la salida para que de ese error.
- Para ello aplicamos la regla de la cadena para BACKPROPAGATION para ver las influencias de detrás hacia delante, y después actualizaremos los pesos.

4- ACTUALIZAMOS PESOS

- Actualizamos pesos mediante las ecuaciones de actualizaciones de pesos por etapa, teniendo en cuenta la tasa de aprendizaje



¿Diferencias en batch size?

- Batch size = 1
- Tamaño del training set > Batch size > 1
- Batch size = Tamaño del training set

La actualización de pesos con solo una muestra no tiene sentido por el problema de los outliers comentados anteriormente

Se suelen hacer en la práctica de potencias de 2 (2,4,8,16...) debido a que es más eficiente computacionalmente para acoplarlo a la estructura de memoria binaria de las GPU

Batch size = 1

- Mucho sesgo en la actualización de pesos -> Outliers
- Tantas actualizaciones de pesos como muestras en el training set -> Convergencia más lenta (época lenta).
- Poca carga en memoria ya que no se almacenan en RAM conjuntos de muestras ni errores por muestra

No se suele usar nunca

Batch size = Tamaño del training set

- Una única actualización de pesos retropropagando el error medio de todas las muestras
- Necesidad de establecer mayor número de épocas para garantizar convergencia
- La época es más rápida ya que solo hay un paso por época
- La RAM debe almacenar todas las muestras y errores asociados -> No es permisible en datasets grandes (OOM!)

Tamaño del training set > Batch size > 1

- Solución de compromiso entre las dos anteriores
- Tamaño de batch suele establecerse como potencia de dos
- n actualizaciones = Pasos por época = int(Tamaño del dataset/Batch size)
- Valor exacto del hiperparámetro: ¿cómo? -> Ciencia empírica

Existen herramientas que facilitan obtener un parámetro óptimo (p ej algoritmos genéticos, ...)

Automi, kerastudent, ...

Para escoger un tamaño de batch, hay que tener en cuenta:

1. restricción por abajo: no cogemos el 1
2. restricción por arriba: depende de la RAM. A mas batch, mas memoria.

Hiperparámetros en el entrenamiento de ANNs

- Época: Una pasada de todos los training samples
- Batch size: Agrupación de training samples considerados para hacer una actualización de los parámetros de la red
- Learning rate: Magnitud del cambio en cada actualización
- Función de activación: Introduce las no linealidades en el esquema de red neuronal
- Función de pérdida: Error entre lo esperado (etiqueta, ground truth) y lo predicho en la época actual
- Topología de la red neuronal: Número de capas ocultas y unidades/neuronas por capa

Learning rate puede ser dinámico, pero depende de la aplicación

Ciencia muy empírica

Tradeoffs:

Batch size

- Alto: Lento en entrenar y mucha memoria RAM consumida, problemático en datasets de grandes dimensiones
- Bajo: Poca memoria consumida, errático en bajar el error (mayor sesgo)

Learning rate

- Alto: Problemas para alcanzar mínimos globales
- Bajo: Lentitud y posible estancamiento en mínimo local

Visualización de entrenamiento con TensorFlow playground <https://playground.tensorflow.org>

Función de activación

Muchas disponibles: https://en.wikipedia.org/wiki/Activation_function

Introduce no linealidades a un sistema para permitir separar datasets no linealmente separables

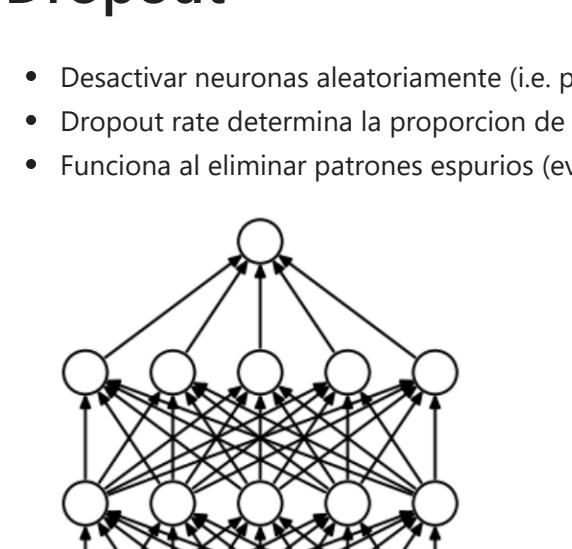
Por convenio, la más usada es la ReLu

Umbral

Fuente: Wikipedia https://en.wikipedia.org/wiki/Step_function

- Input > umbral = 1, else 0
- Difícil convergencia durante entrenamiento
- Para multiclase, cómo decidir el output correcto (tres 1 y un 0)

Linear

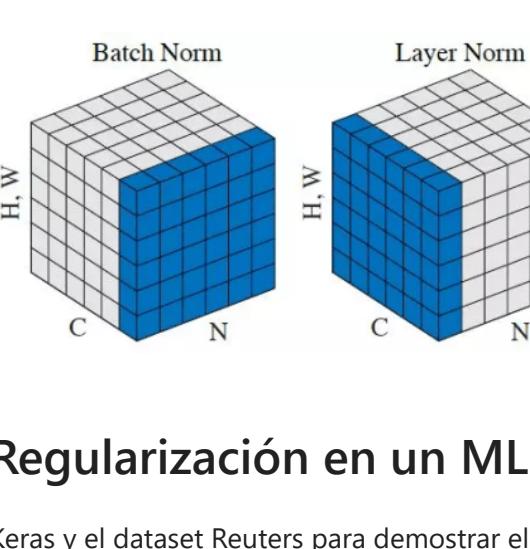


Fuente: Wikipedia https://en.wikipedia.org/wiki/Activation_function

- Input - output linear
- Output -inf a +inf
- Combinación de funciones lineales da una función lineal
- Explosión de valores (difícil entrenamiento)

Se suele usar para función de activación en la capa de salida para problemas de regresión, una única neurona

Sigmoide



Fuente: Wikipedia https://en.wikipedia.org/wiki/Activation_function

- No lineal
- Output 0 a 1

• Problema: entrenamiento puede ser lento cuando los valores residen en los extremos

Sirve mucho para problemas de clasificación, tanto en las hidden layers como en la output layer.

Tangencial

Fuente: Wolfram <https://reference.wolfram.com/language/ref/Tanh.html>

- No lineal
- Output -1 a 1

• Gradiente más fuerte que sigmoid (derivadas son más intensas, por lo que la convergencia puede dificultarse)

En la práctica no se usa, se usa una sigmoid, porque facilita más la convergencia

ReLU (Rectified Linear Unit)

Fuente: Wikipedia [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

- No lineal
- Output 0 a +inf

• Alquier función puede aproximarse con una combinación de ReLUs

• Aunque puede explotar (lado positivo) es más eficiente (menos cálculos) que sigmoid o tanh

▪ Sigmoid / tanh el cálculo de la función de activación es más costoso

▪ Con ReLU, muchas serán 0 (sin cálculo)

Facil cálculo, si son positivos se quedan positivos, si son negativos, ceros La ReLU es a día de hoy la función de activación por excelencia en las capas intermedias (cálculos más eficientes)

La no linealidad de la función de activación es lo que permite a las ANNs aprender decision boundaries que no son lineales

Variaciones de SGD (stochastic gradient descent)

- RMSprop: dividir gradiente entre la media de las magnitudes recientes
- Momentum: la magnitud del cambio en cada actualización
- Learning rate: Magnitud del cambio en cada actualización

Si me acerco muy rápido hacia el mínimo y obteniendo gradientes pronunciados, aprovecho el momento para ir "más rápido" y evitar estancarme en mínimos locales.

Otros disponibles en Keras <https://keras.io/optimizers/>

Instalacion de Keras

- Instalacion backend (CPU, GPU)
- pip install tensorflow
- pip install tensorflow-gpu

Más información acerca de la instalación:

<https://www.pyimagesearch.com/2019/12/09/how-to-install-tensorflow-2-0-on-ubuntu/>

<https://www.tensorflow.org/install/gpu>

Keras

TensorFlow

cuDNN

CUDA

DATASETS

<https://keras.io/api/datasets/>

Nuestra primera red neuronal: MNIST dataset

Keras y el dataset MNIST para el reconocimiento de dígitos escritos a mano

MLP aplicado a texto: Ejemplo REUTERS

Keras y el dataset MNIST para la clasificación de reseñas

Muchas disponibles: https://en.wikipedia.org/wiki/Activation_function

Introduce no linealidades a un sistema para permitir separar datasets no linealmente separables

Por convenio, la más usada es la ReLu

Ejercicio

Comparar los resultados con otros modelos:

- Modificar la arquitectura de la red (número de neuronas, número de hidden layers)
- Demostrar que con menos hidden units se pierde información que no se puede recuperar
- Experimentar con 'tanh' en vez de 'relu' como función de activación
- Distinto split training / validation (mayoría validation)

Se suele usar para función de activación en la capa de salida para problemas de regresión, una única neurona

Sigmaide

Fuente: Wikipedia https://en.wikipedia.org/wiki/Activation_function

- Problema: entrenamiento puede ser lento cuando los valores residen en los extremos

Sirve mucho para problemas de clasificación, tanto en las hidden layers como en la output layer.

ReLU (Rectified Linear Unit)

Fuente: Wikipedia [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

- No lineal

Output 0 a +inf

• Combinación de funciones lineales da una función lineal

• Explosión de valores (difícil entrenamiento)

Se suele usar para función de activación en la capa de salida para problemas de regresión, una única neurona

Optimizacion de redes neuronales

- Preprocesado para ANN:
 - Vectorización del input (tensores)
 - Normalización de valores para acelerar entrenamiento (todas las features deben tener misma escala), media 0 y std 1

• Gradientes: Eliminar, interpolar o utilizar 0 (si no tiene significado)

▪ Automática: calcular el gradiente

▪ Manual: calcular el gradiente

▪ Backpropagation: calcular el gradiente

▪ Adam: calcular el gradiente

▪ RMSprop: calcular el gradiente

▪ Momentum: calcular el gradiente

▪ SGD: calcular el gradiente

▪ Adagrad: calcular el gradiente

▪ Adadelta: calcular el gradiente

▪ Adamax: calcular el gradiente

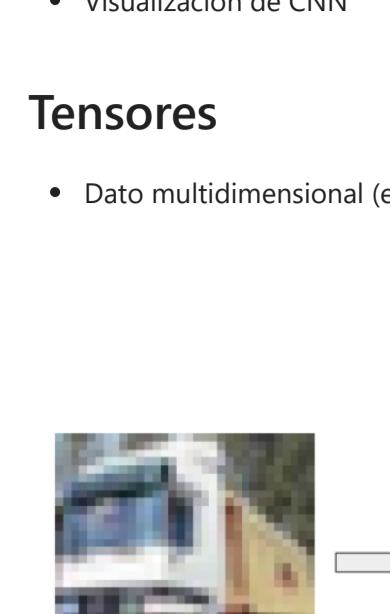
▪ Ftrl: calcular el gradiente

▪ Nesterov: calcular el gradiente

▪ GradientDescent: calcular el gradiente

07M AIR - Redes Neuronales y Deep Learning

Clase 03: Deep Learning y Deep vision



Profesores: Adrián Colomer Granero / Gabriel Enrique Muñoz Ríos

Autor: Carlos Fernández Musoles

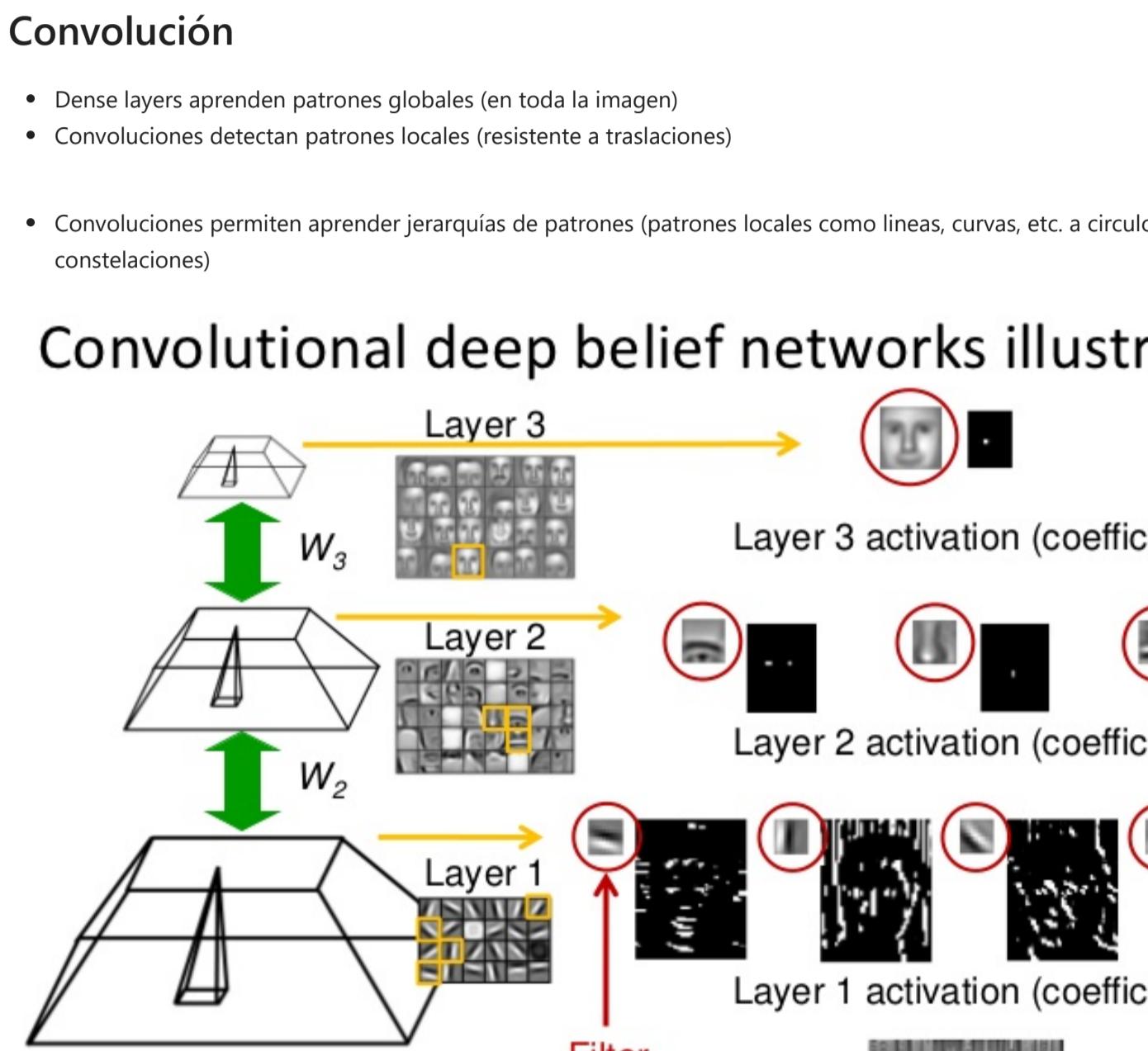
VC04-VC06_DeepVision_Jupyter

Sumario

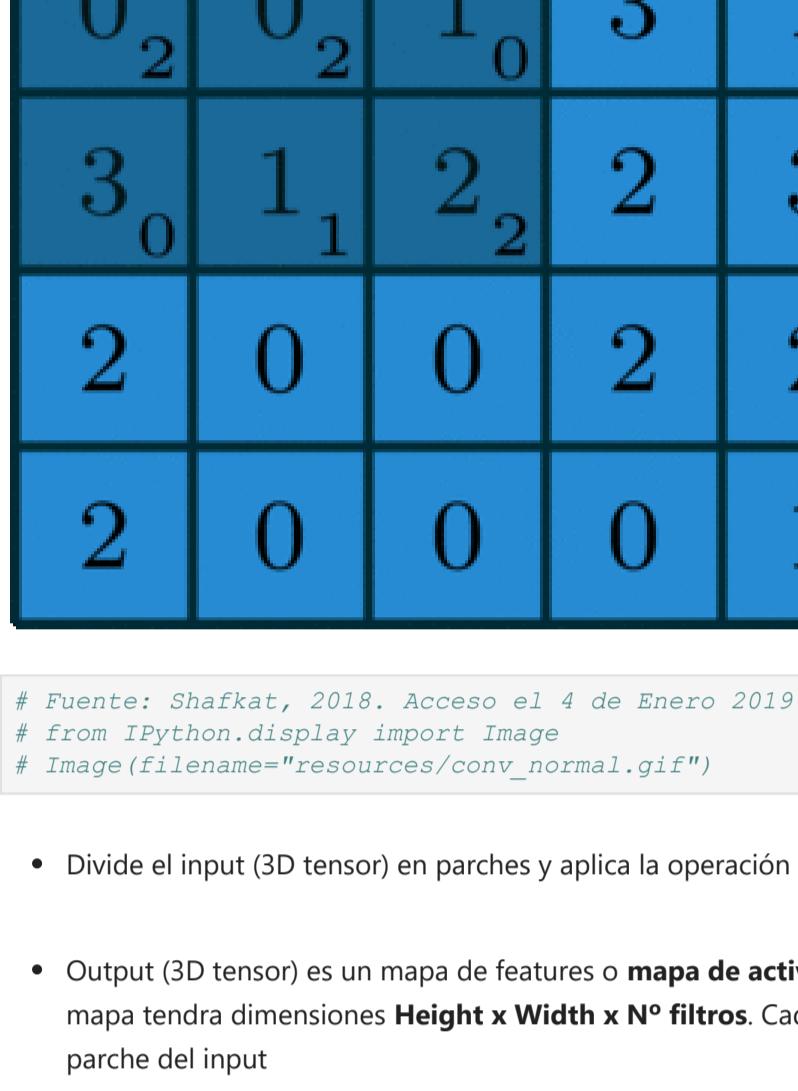
- Introducción a deep learning
- Redes neuronales convolucionales
- Trabajar con pocos datos
- Visualización de CNN

Tensores

- Dato multidimensional (escalar, vector, matriz, tensor)



- Deep learning consiste en una serie de transformaciones (operaciones matemáticas con tensores) del input para dar output.



- Operación más importante: tensor dot (matmul)

```
In [1]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

model = Sequential()
model.add(Dense(64, input_shape=(256,))) # output 1D tensor con dimensiones (64)
# la sintaxis (256,) indica tupla con solo un numero
model.add(Dense(32)) # input asumido de la capa anterior output 1D tensor con dimensiones (32)

Using TensorFlow backend.

In [3]: # mostrar estructura del modelo
model.summary()

Model: "sequential_1"
Layer (type)          Output Shape         Param #
===== =====
dense_1 (Dense)     (None, 64)           16448
dense_2 (Dense)     (None, 32)            2080
=====
Total params: 18,528
Trainable params: 18,528
Non-trainable params: 0
```

Construir redes

- Capas compatibles en las dimensiones output - input

Tipos de capas

- Cada tipo de capa suele tener un uso (dependiente de la red a diseñar según el tipo de entrada y la aplicación):
 - Dense para tratar input vector o **datos estructurados** (2D tensor de samples,features)
 - LSTM (Recurrent Neural Networks) para tratar **secuencias lógicas** (3D tensores de samples,timestep,features)
 - Convolucionales (Convolutional Neural Network) para tratar **imágenes** (4D tensores de samples,height,width,channels)

Estructura de la red neuronal

- La más habitual es secuencial (feedforward), pero hay más
- Two-branch, Multihead, Inception
- La arquitectura ideal es un arte: Documentación tipo de problema, experiencia, experimentación.

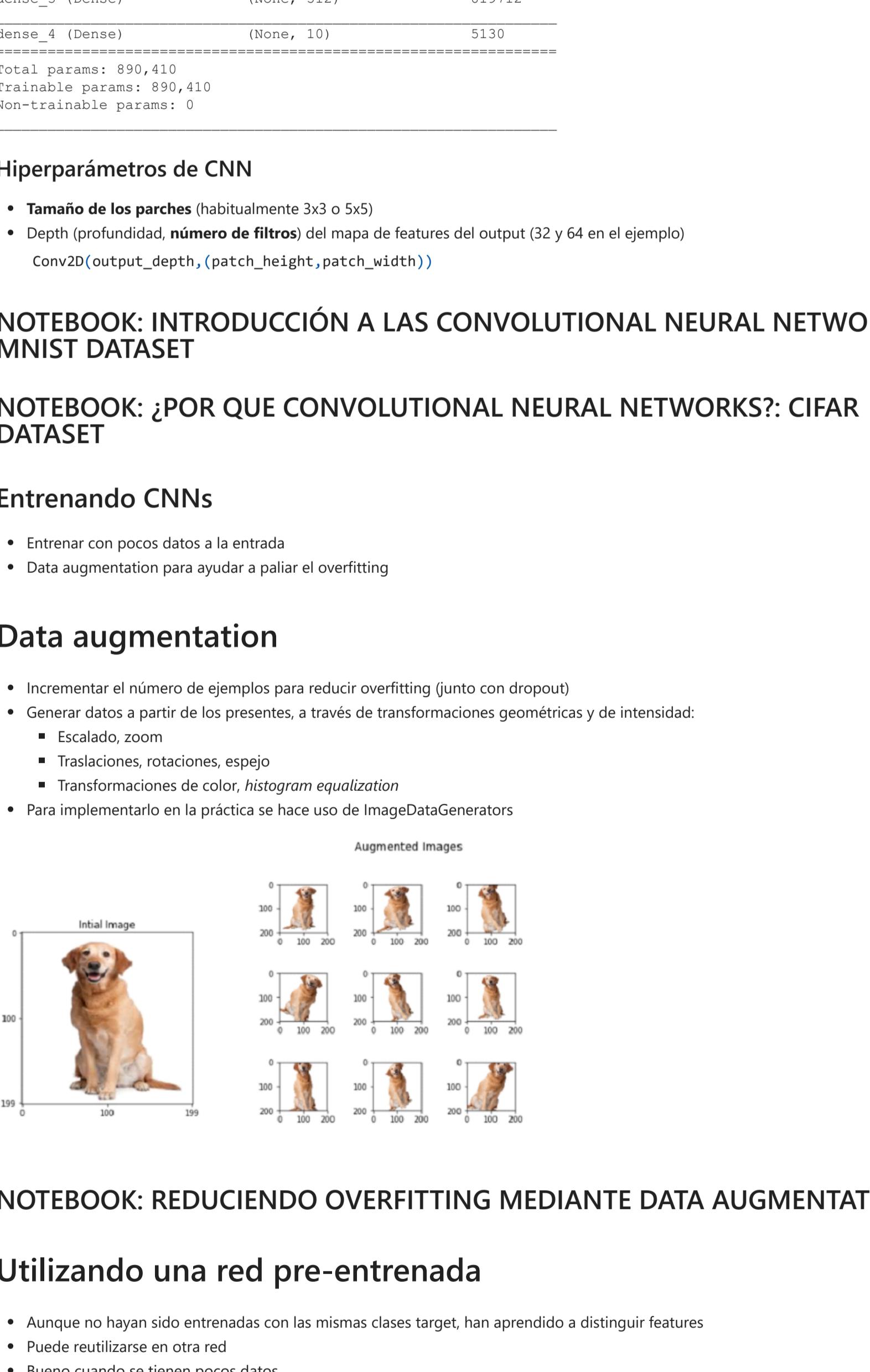
Deep learning en Computer Vision

- Redes convolucionales (CNN)
 - Descripción
 - Entrenamiento
- Trabajar con pocos datos
 - Data augmentation
 - Redes pre-entrenadas: Transfer Learning & Fine-tuning
- Visualizando lo que aprenden las CNNs

Convolución

- Dense layers aprenden patrones globales (en toda la imagen)
- Convoluciones detectan patrones locales (resistente a traslaciones)

- Convoluciones permiten aprender jerarquías de patrones (patrones locales como líneas, curvas, etc. a círculos, rectángulos, constelaciones)



Fuente: <https://cs.stackexchange.com/questions/16545/what-is-the-difference-between-a-neural-network-a-deep-learning-system-and-a-de>

Cómo lo hacen

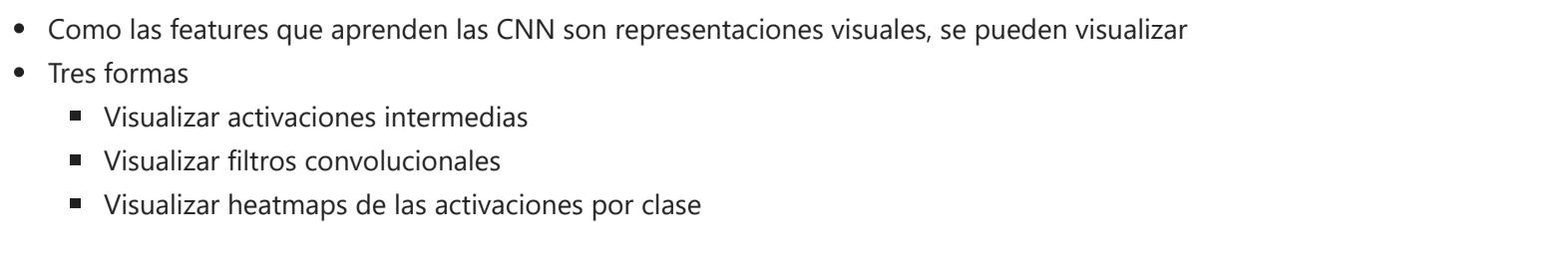
- Red sin pooling

```
In [1]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#pooling
# from IPython.display import Image
# Image(filename="resources/conv_normal.gif")
```

- Divide el input (3D tensor) en parches y aplica la operación convolución (la misma en cada capa) a cada parche

- Output (3D tensor) es un mapa de features o **mapa de activaciones** (cada una el resultado de aplicar la transformación). Dicho mapa tendrá dimensiones **Height x Width x N° filtros**. Cada capa del volumen (eje z) es el resultado de aplicar un filtro a cada parche del input

- La convolución desliza cada parche sobre el input, parando en cada posible posición y aplicando la transformación (función kernel)



```
In [5]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#multichannel
# Image(filename="resources/conv_multichannel.gif")
```

- Stride determina el salto entre ventana y ventana

- Poco habitual su utilización como efecto downsample

Conv2D(..., stride=valor): int, salto entre ventanas

Pooling

- Se usan para hacer **downsampling del input feature** (reducir dimensionalidad espacial)

- Similar a convoluciones, pero no aplican un kernel sino que aplican **función 'max'** (o average)

- Por defecto, tamaño 2x2 y stride 2

```
In [6]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#pooling
# Image(filename="resources/bias_cnn.gif")
```



```
In [7]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#padding
# Image(filename="resources/conv_padding.gif")
```

Padding y striding

- Sólo hay 9 formas de aplicar un parche 3x3 a un imagen de 5x5 sin OOB!

Si se quiere aplicar a todos los pixels --> **padding** (añadir filas y columnas para hacerlo posible)

Conv2D(..., padding='valor'): 'valid' sin padding, 'same' para igualar tamaño input y output feature


```
In [9]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#padding
# Image(filename="resources/conv_stride.gif")
```

- Stride determina el salto entre ventana y ventana

- Poco habitual su utilización como efecto downsample

Conv2D(..., stride=valor): int, salto entre ventanas

Pooling

- Se usan para hacer **downsampling del input feature** (reducir dimensionalidad espacial)

- Similar a convoluciones, pero no aplican un kernel sino que aplican **función 'max'** (o average)

- Por defecto, tamaño 2x2 y stride 2

```
In [12]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#padding
# Image(filename="resources/conv_stride.gif")
```

```
In [13]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#maxpooling
# Image(filename="resources/maxpooling.gif")
```



```
In [14]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#bias
# Image(filename="resources/bias.gif")
```

Hiperparámetros de CNN

- Tamaño de los parches (habitualmente 3x3 o 5x5)

- Depth (profundidad, número de filtros) del mapa de features del output (32 y 64 en el ejemplo)

Conv2D(..., depth=valor): int, número de filtros

```
In [15]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#depth
# Image(filename="resources/conv_depth.gif")
```



```
In [16]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#bias
# Image(filename="resources/bias.gif")
```

Entrenando CNNs

- Entrenar con pocos datos a la entrada

- Data augmentation para ayudar a pillar el overfitting

```
In [17]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [18]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [19]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [20]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [21]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [22]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [23]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [24]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [25]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [26]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [27]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [28]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [29]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [30]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [31]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [32]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [33]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [34]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [35]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [36]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [37]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [38]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [39]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [40]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [41]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [42]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [43]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-convolutional-networks-1#dataaugmentation
# Image(filename="resources/data_augmentation.gif")
```

```
In [44]: # Fuente: Shafkat, 2018. Acceso el 4 de Enero 2019. https://towardsdatascience.com/intuitively-understanding-conv
```

Mounted at /content/drive/

INTRODUCCIÓN A LAS CONVOLUTIONAL NEURAL NET DATASET

```
Downloaded data from https://storage.googleapis.com/t  
11493376/11490434 [=====] - 0  
11501568/11490434 [=====] - 0
```

- Acondicionando el conjunto de datos
 - Cambiar al rango 0-1 para disminuir el coste computacional y favorecer la convergencia.
 - Dividir el conguunto de training en train-val, y poder monitorizar el overffiting y optimizar hiperparametros.
 - Se podrian pasar las labels a categorico con la funcion `to_categorical`, pasando a one-hot-encoding, pero en este caso vamos a utilizar enteros, aplicando un truco en keras.
 - Recordemos que el one-hot-enconding era necesario para el calculo de la categorical-cross-entropy, por como estaba definida.
 - Como en este caso vamos a usar para clasificar, la red CNN en lugar de un perceptron, necesitamos indicarle las dimensiones de la capa densa que usaremos de entrada, ya que en contraposicion al perceptron, ya no es necesario aplanar los datos para la capa de entrada.

```
In [ ]: # Pre-procesado obligatorio cuando trabajo con redes neuronales
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.backend import expand_dims

x_train, x_te = x_train / 255.0, x_test / 255.0 #Cambio al rango 0-1 -> Disminuyo CC
#¿Que pasa si empleo labels con etiquetas número entero?
print(y_train[0]) #digito de la primera muestra
#y_train = to_categorical(y_train, num_classes=10) #One-hot encoding para minimizar error
#y_te = to_categorical(y_test, num_classes=10)
x_tr, x_val, y_tr, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=42) # 3 subconjuntos

#Expandir dimensiones porque en CNN tengo que especificar el número de canales
print(x_tr.shape) # numero de muestras y tamaño de las imagenes
x_tr = expand_dims(x_tr, axis=3)
x_val = expand_dims(x_val, axis=3)
x_te = expand_dims(x_te, axis=3)
print(x_tr.shape) # capa extra de dimension
```

5
(54000, 28, 28)
(54000, 28, 28, 1)

- Creando la topología de Red Neuronal (CNN) y entrenándola

Con el modelo de la API `Sequential()` vamos a comenzar a crear bloques convolucionales Conv2D, compuestos de una capa Conv2D y una capa de MaxPooling2D.

No utilizamos mas capas convolucionales, dado que la complejidad del problema no lo requiere. El dataset es muy sencillo, e incluso podríamos reducir mas el numero de filtros.

```
In [ ]: # Construccion de una red CNN
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
# Red feedforward API secuencial
convnet = Sequential()

# BASE MODEL
convnet.add(layers.Conv2D(32, (3,3), input_shape=(28,28,1), activation='relu'))
convnet.add(layers.MaxPooling2D((2,2)))
```

```
# Construccion de una red CNN
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
# Red feedforward API secuencial
convnet = Sequential()

# BASE MODEL
convnet.add(layers.Conv2D(32, (3,3), input_shape=(28,28,1), activation='relu'))
convnet.add(layers.MaxPooling2D((2,2)))

convnet.add(layers.Conv2D(64, (3,3), activation='relu'))
convnet.add(layers.MaxPooling2D((2,2)))

convnet.add(layers.Conv2D(64, (3,3), activation='relu'))
```

Es conveniente plotear el `convnet`.
se aplica padding, pooling, etc.

Podemos ver ademas como el vector
(None, 576), el cual entrara a la prime

optimizacion de los coeficientes de los filtros. Los cuales estan especializados en sacar diversos patrones concretos de las imagenes.

In []: convnet.summary()

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling 2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 10)	650
=====		
Total params:	93,322	
Trainable params:	93,322	
Non-trainable params:	0	

El `None` que nos aparece en la Output Shape es porque aun no hemos ejecutado el `fit`, por lo que al hacer el `convnet.summary()` aun no sabemos cuantas muestras habran, dado que vendran determinadas en tiempo de ejecucion por el **tamaño de batch**.

Truco Keras para evitar one-hot-encoding: utilizar como funcion de perdida el `sparse_categorical_crossentropy`, en lugar del `categorical_crossentropy`.

In []:

```
convnet.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy', #If labels are integer
                 #loss='categorical_crossentropy',           #If labels are one-hot encoded
                 metrics=['accuracy'])
```

In []:

```
H = convnet.fit(x_tr, y_tr, epochs=5, batch_size=128, validation_data=(x_val, y_val))
```

```
Epoch 1/5
422/422 [=====] - 15s 11ms/step - loss: 0.2616 - accuracy: 0.9233 - val_loss: 0.0875 -
val_accuracy: 0.9723
Epoch 2/5
422/422 [=====] - 4s 10ms/step - loss: 0.0653 - accuracy: 0.9797 - val_loss: 0.0494 -
val_accuracy: 0.9847
Epoch 3/5
422/422 [=====] - 4s 10ms/step - loss: 0.0435 - accuracy: 0.9857 - val_loss: 0.0520 -
```

```
122, 122 [  
val_accuracy: 0.9892
```

- ```
import numpy as np
Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 5), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 5), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 5), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 5), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

print(np.mean(H.history["loss"]))
print(np.mean(H.history["val_loss"]))
print(np.mean(H.history["accuracy"]))
print(np.mean(H.history["val_accuracy"]))

0.08656449839472771
0.05164063945412636
0.9738111019134521
0.9843666553497314



- Probando el conjunto de datos en el subset de test y evaluando el performance del modelo


```
from sklearn.metrics import classification_report
# Evaluando el modelo de predicción con las imágenes de test
print("[INFO]: Evaluando red neuronal...")
predictions = convnet.predict(x_te, batch_size=128)
#print(y_te[0])
#print(predictions[0])
print(classification_report(y_test, predictions.argmax(axis=1)))

[INFO]: Evaluando red neuronal...
      precision    recall  f1-score   support

          0       0.99     0.99     0.99      980
          1       1.00     0.99     0.99     1135
          2       0.99     0.99     0.99     1032
          3       0.99     1.00     0.99     1010
          4       1.00     0.99     0.99      982
          5       0.99     0.98     0.99      892
          6       0.99     0.99     0.99      958
          7       0.98     0.99     0.99     1028
          8       0.99     0.98     0.99      974
          9       0.99     0.99     0.99     1009
```


```

weighted avg 0.99 0.99 0.99 10000

Comparando con el perceptron multicapa, podemos observar como se obtiene mucho mas accuracy, con muchas menos epochas de entrenamiento.

## ¿POR QUE CONVOLUTIONAL NEURAL NETWORKS?: CIFAR DATASET

### - Cargando el conjunto de datos y acondicionándolo

Dataset CIFAR-10: 10 clases que mezcla animales y vehiculos. Imagenes RGB de 32x32

```
In []: # Importando el set de datos CIFAR10
from tensorflow.keras.datasets import cifar10
from sklearn.preprocessing import LabelBinarizer
print("[INFO]: Loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
Normalizamos
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
labelNames = ["Avión", "Automóvil", "Pájaro", "Gato", "Ciervo", "Perro", "Rana", "Caballo", "Barco", "Camión"]
print(trainX.shape)
print(trainY.shape)
Por si es necesario convertir a one-hot encoding
#lb = LabelBinarizer()
#trainY = lb.fit_transform(trainY)
#testY = lb.transform(testY)
```

```
[INFO]: Loading CIFAR-10 data...
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [=====] - 6s 0us/step
170508288/170498071 [=====] - 6s 0us/step
(50000, 32, 32, 3)
(50000, 1)
```

```
In []: print(testX.shape)
print(testY.shape)

(10000, 32, 32, 3)
(10000, 1)
```

### - Inspeccionando el conjunto de datos

```
In []: import matplotlib.pyplot as plt
fig = plt.figure(figsize=(14,10))
for n in range(1, 29): # 28 primeras imagenes
 fig.add_subplot(4, 7, n)
 img = trainX[n]
 plt.imshow(img)
 plt.title(labelNames[trainY[n][0]])
 plt.axis('off')
```

The image displays a 4x7 grid of 28 small photographs, each labeled with a caption below it. The images represent various categories of objects and animals:

- Row 1: Barco, Gato, Ciervo, Caballo, Caballo, Pájaro, Camión
- Row 2: Camión, Camión, Gato, Pájaro, Rana, Ciervo, Gato
- Row 3: Rana, Rana, Pájaro, Rana, Gato, Perro, Ciervo

The images are as follows:

- Barco: A white boat on water.
- Gato: A black and white cat.
- Ciervo: A brown deer in a wooded area.
- Caballo: A dark brown horse standing.
- Caballo: Another view of a horse.
- Pájaro: A blue and yellow bird perched on a branch.
- Camión: A large white delivery truck.
- Camión: Another view of a truck.
- Gato: A fluffy orange and white cat.
- Pájaro: A brown bird, possibly a sparrow.
- Rana: A green frog.
- Ciervo: A small deer fawn.
- Gato: A dark cat.
- Rana: A brown frog.
- Rana: Another view of a frog.
- Pájaro: A colorful bird with blue and red feathers.
- Rana: A brown frog.
- Gato: A dark cat.
- Perro: A black dog.
- Ciervo: A small deer fawn.

## - Creando la topología de red neuronal y entrenándola: MLP

Perceptron multicapa:

- 5 capas ocultas de 2048, 1024, 512, 128 y 32 neuronas respectivamente.
- Funcion de activacion: ReLu
- Funcion de activacion de salidas: softmax

y mas va aumentando el eje z (numero de filtros) tras cada bloque convolucional. A medida que avanzando en la red convolucional, los patrones van a ser mas complejos.

```
In []: # Imports necesarios
import numpy as np
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import SGD
import matplotlib.pyplot as plt

Arquitectura de red
Definimos el modo API Sequential
model = Sequential() #(X)
model.add(Flatten()) #(X)
Primera capa oculta
model.add(Dense(2048, input_shape=(32*32*3,), activation="relu")) #(X)
#model.add(Dropout(0.5))
Segunda capa oculta
model.add(Dense(1024, activation="relu")) #(X)
#model.add(Dropout(0.5))
Tercera capa oculta
model.add(Dense(512, activation="relu")) #(X)
#model.add(Dropout(0.5))
Cuarta capa oculta
model.add(Dense(128, activation="relu")) #(X)
#model.add(Dropout(0.5))
Quinta capa oculta
model.add(Dense(32, activation="relu")) #(X)
Capa de salida
model.add(Dense(10, activation="softmax")) #(X)

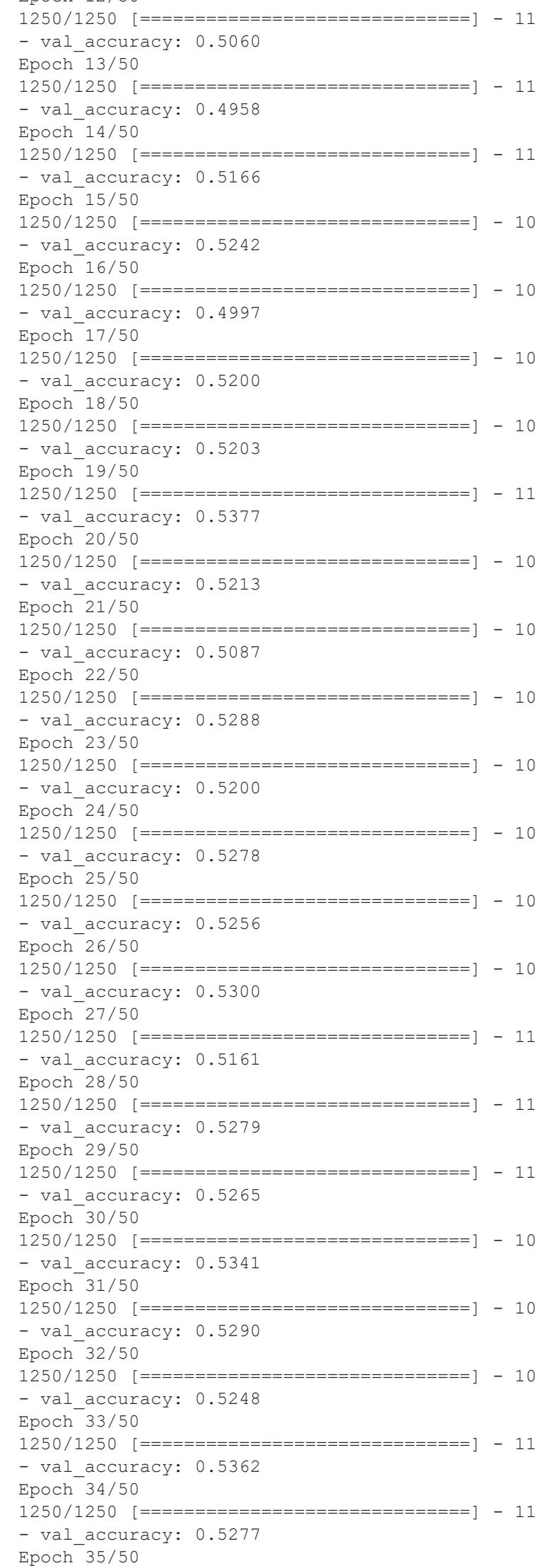
Compilamos el modelo y entrenamos
print("[INFO]: Entrenando red neuronal...")
Compilamos el modelo
model.compile(loss="sparse_categorical_crossentropy", optimizer=SGD(0.01), metrics=["accuracy"]) # Etiquetas en
model.compile(loss="categorical_crossentropy", optimizer=SGD(0.01), metrics=["accuracy"]) # Etiquetas binaria
Entrenamos el perceptrón multicapa
H = model.fit(trainX, trainY, validation_split=0.2, epochs=50, batch_size=32) #(X)
```

```
Efectuamos predicciones
predictions = model.predict(testX, batch_size=32) # (X)
Obtenemos el report
print(classification_report(testY, predictions.argmax(axis=1), target_names=labelNames)) # Etiquetas en decimal
print(classification_report(testY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)) # Etiquetas en decimal

Mostramos gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 50), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 50), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 50), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 50), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

[INFO]: Entrenando red neuronal...
Epoch 1/50
1250/1250 [=====] - 12s 9ms/step - loss: 1.9195 - accuracy: 0.3052 - val_loss: 1.7680
- val_accuracy: 0.3686
Epoch 2/50
1250/1250 [=====] - 10s 8ms/step - loss: 1.6884 - accuracy: 0.3984 - val_loss: 1.6507
- val_accuracy: 0.4176
```

```
- val_accuracy: 0.4410
Epoch 4/50
1250/1250 [=====] - 11s 8ms/step - loss: 1.5253 - accuracy: 0.4578 - val_loss: 1.6312
- val_accuracy: 0.4253
Epoch 5/50
1250/1250 [=====] - 11s 9ms/step - loss: 1.4714 - accuracy: 0.4762 - val_loss: 1.5242
- val_accuracy: 0.4696
Epoch 6/50
1250/1250 [=====] - 11s 9ms/step - loss: 1.4255 - accuracy: 0.4928 - val_loss: 1.4907
- val_accuracy: 0.4733
Epoch 7/50
1250/1250 [=====] - 11s 8ms/step - loss: 1.3823 - accuracy: 0.5069 - val_loss: 1.5390
- val_accuracy: 0.4507
Epoch 8/50
1250/1250 [=====] - 10s 8ms/step - loss: 1.3424 - accuracy: 0.5210 - val_loss: 1.4796
- val_accuracy: 0.4764
Epoch 9/50
1250/1250 [=====] - 11s 9ms/step - loss: 1.3034 - accuracy: 0.5357 - val_loss: 1.4053
- val_accuracy: 0.5064
Epoch 10/50
1250/1250 [=====] - 12s 9ms/step - loss: 1.2672 - accuracy: 0.5521 - val_loss: 1.4624
- val_accuracy: 0.4839
Epoch 11/50
1250/1250 [=====] - 11s 8ms/step - loss: 1.2322 - accuracy: 0.5615 - val_loss: 1.3868
- val_accuracy: 0.5100
```



```
1250/1250 [=====] - 11s 9ms/step - loss: 0.3595 - accuracy: 0.8696 - val_loss:
- val_accuracy: 0.5324
Epoch 37/50
1250/1250 [=====] - 11s 9ms/step - loss: 0.3787 - accuracy: 0.8664 - val_loss:
- val_accuracy: 0.5231
Epoch 38/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.3670 - accuracy: 0.8708 - val_loss:
- val_accuracy: 0.5186
Epoch 39/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.3426 - accuracy: 0.8782 - val_loss:
```

```
1250/1250 [=====] - 11s 8ms/step - loss: 0.3337 - accuracy: 0.8837 - val_loss: 2.0916
- val_accuracy: 0.5271
Epoch 41/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.3043 - accuracy: 0.8925 - val_loss: 2.4428
- val_accuracy: 0.5005
Epoch 42/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.2923 - accuracy: 0.8954 - val_loss: 2.5469
- val_accuracy: 0.4955
Epoch 43/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.2741 - accuracy: 0.9035 - val_loss: 2.2858
- val_accuracy: 0.5305
Epoch 44/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.2516 - accuracy: 0.9142 - val_loss: 2.2984
- val_accuracy: 0.5349
Epoch 45/50
```

```
Epoch 46/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.2298 - accuracy: 0.9207 - val_loss: 2.3474
- val_accuracy: 0.5411
Epoch 47/50
1250/1250 [=====] - 11s 8ms/step - loss: 0.2251 - accuracy: 0.9200 - val_loss: 2.5573
- val_accuracy: 0.5121
Epoch 48/50
1250/1250 [=====] - 10s 8ms/step - loss: 0.1969 - accuracy: 0.9317 - val_loss: 2.4321
- val_accuracy: 0.5321
Epoch 49/50
1250/1250 [=====] - 11s 8ms/step - loss: 0.2028 - accuracy: 0.9278 - val_loss: 2.5149
- val_accuracy: 0.5407
Epoch 50/50
1250/1250 [=====] - 11s 8ms/step - loss: 0.1881 - accuracy: 0.9340 - val_loss: 2.5839
- val_accuracy: 0.5231
[INFO]: Evaluando modelo...
 precision recall f1-score support
Avión 0.47 0.71 0.56 1000
Automóvil 0.68 0.59 0.63 1000
Pájaro 0.35 0.55 0.43 1000
Gato 0.35 0.37 0.36 1000
Ciervo 0.50 0.38 0.43 1000
```

Se puede observar en la grafica al ver el `val_loss` y otros parametros, que se produce **overfitting**. Posiblemente causado por una falta de datos, porque las imagenes son RGB de dimensiones pequeñas, o porque hay clases muy similares entre si.

Metiendo regularizacion con los Dropouts, se podria mejorar la red, pero aun asi seria muy complejo.

**- Creando la topología de red neuronal y entrenándola: CNN**

La creamos mediante una API funcional (en lugar de mediante el `Model=Sequential()` que solo permite crear redes secuenciales), la

- No hace falta crear un modelo al principio.
- Especificar entradas (dimension imagenes) `inputs = Input(shape=(trainX.shape[1], trainX.shape[2], trainX.shape[3]))` (32x32x3)
- Estructura de la red:
  1. Set de 3 bloques convolucionales:
    - 1º (2x Conv2d 32 filtros) + Normalizacion + MaxPooling2D + Dropout
    - 2º (2x Conv2D 64 filtros) + MaxPooling2D + MaxPooling2D + Dropout
    - 3º (2x Conv2D 256 filtros) + MaxPooling2D + MaxPooling2D + Dropout
    - En cada capa hay que especificar cual es su entrada (`inputs`), (`x1`), ...
  2. Capa de clasificacion (perceptron multicapa)
    - capa flatten
    - capa Dense para reducir dimensionalidad
    - Normalizacion
    - Dropout
    - Capa de salida Densa 10 clases `predictions = Dense(10, activation="softmax")(xfc)`
  3. Compilar el modelo

```
Import the necessary packages
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense, Dropout, BatchNormalization, Max
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from google.colab import drive

#####
Definimos la arquitectura
#####

#BASE MODEL
Definimos entradas
inputs = Input(shape=(trainX.shape[1], trainX.shape[2], trainX.shape[3]))

Primer set de capas CONV => RELU => CONV => RELU => POOL
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs) # la salida sera 32x32x32
x1 = BatchNormalization()(x1)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1) # la salida sera 32x32x32
x1 = BatchNormalization()(x1)
x1 = MaxPooling2D((2, 2))(x1) # la salida sera 16x16x32
```

```

x1 = MaxPooling2D(pool_size=(2, 2))(x1) # la salida sera 16x16x32
x1 = Dropout(0.25)(x1)

Segundo set de capas CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1) # (X)
x2 = BatchNormalization()(x2) # (X)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2) # (X)
x2 = BatchNormalization()(x2) # (X)
x2 = MaxPooling2D(pool_size=(2, 2))(x2) # (X) # la salida sera 8x8x64
x2 = Dropout(0.25)(x2) # (X)

Tercer set de capas CONV => RELU => CONV => RELU => POOL
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x2) # (X)
x3 = BatchNormalization()(x3) # (X)
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x3) # (X)
x3 = BatchNormalization()(x3) # (X)
x3 = MaxPooling2D(pool_size=(2, 2))(x3) # (X) # la salida sera 4x4x256 = 4096 elementos del vector de caract
x3 = Dropout(0.25)(x3) # (X)

TOP MODEL
Primer (y único) set de capas FC => RELU
xfc = Flatten()(x3) # (X)
xfc = Dense(512, activation="relu")(xfc) # (X) # reducimos las caracteristicas de 4096 a 512
xfc = BatchNormalization()(xfc) # (X)

```

```

predictions = Dense(10, activation="softmax")(xfc) #(X) # salida de la ultima capa

Unimos las entradas y el modelo mediante la función Model con parámetros inputs y outputs (Consultar la documentación)
model_cnn = Model(inputs=inputs, outputs=predictions) #(X)

Compilar el modelo
print("[INFO]: Compilando el modelo...")
model_cnn.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001, decay=0, beta_1=0.9, beta_2=0.999))

Entrenamiento de la red
print("[INFO]: Entrenando la red...")
H = model_cnn.fit(trainX, trainY, validation_split=0.2, batch_size=128, epochs=50, verbose=1) #(X)

Almaceno el modelo en Drive
Montamos la unidad de Drive
drive.mount('/content/drive') #(X)
Almacenamos el modelo empleando la función mdoel.save de Keras
model_cnn.save(BASE_FOLDER+"deepCNN_CIFAR10.h5") #(X)

Evaluación del modelo
print("[INFO]: Evaluando el modelo...")
Efectuamos la predicción (empleamos el mismo valor de batch_size que en training)
predictions = model_cnn.predict(testX, batch_size=128) #(X)
Sacamos el reporte para test

```

```
Gráficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 50), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 50), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 50), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 50), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()
```





# 07MIAR – Redes neuronales y deep learning



**Universidad**  
Internacional  
de Valencia

Tareas avanzadas de *computer vision*  
empleando aprendizaje profundo

# 01

# Introducción

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

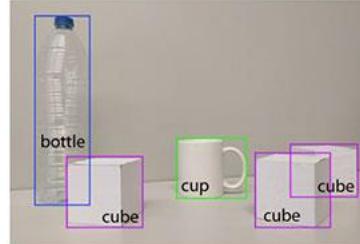
25/10/2021

# Tareas avanzadas CV

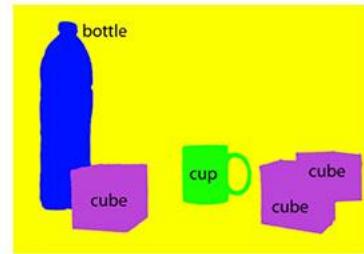
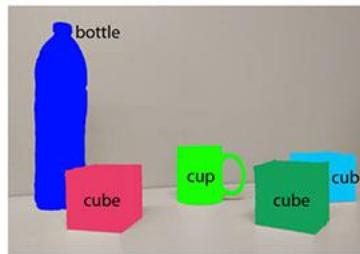
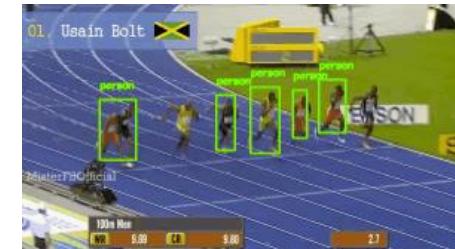
- Dentro del campo de la **visión por computador** existen **diversas tareas de interés** que tienen como denominador común el **tratamiento con los objetos** de una **escena** tanto **estática** como en **movimiento**.



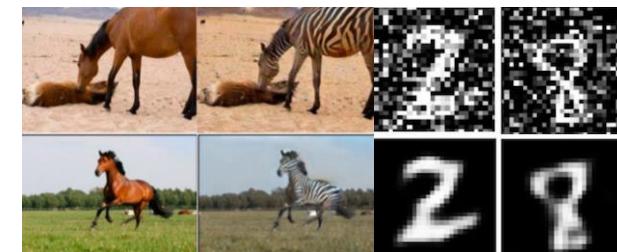
Clasificación de imagen



Localización de objetos

Segmentación  
semánticaSegmentación de  
instancia

Tracking o localización dinámica

Generación sintética de imágenes, *denoising*, compresión, detección de anomalías, etc.

02

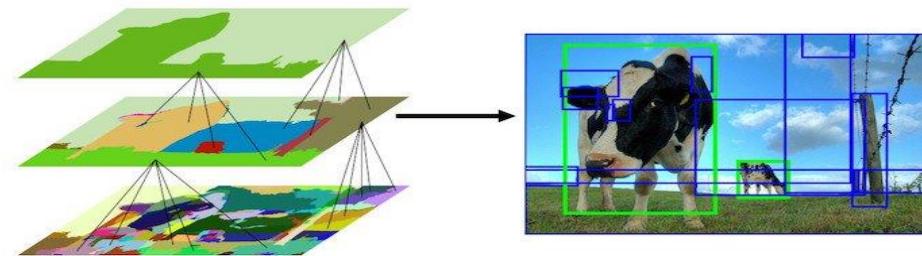
# Detección de objetos

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

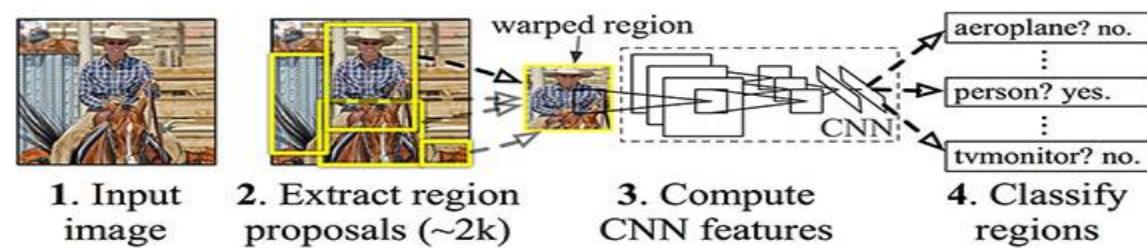
# R-CNN: Regiones con características CNN

- Método propuesto por **Ross Girshick** en **2013**. Se compone de los siguientes **pasos**:

  1. **Extracción de regiones candidatas** mediante el **algoritmo** de **búsqueda selectiva**. Agrupación jerárquica de regiones similares basada en color, textura, tamaño y forma.

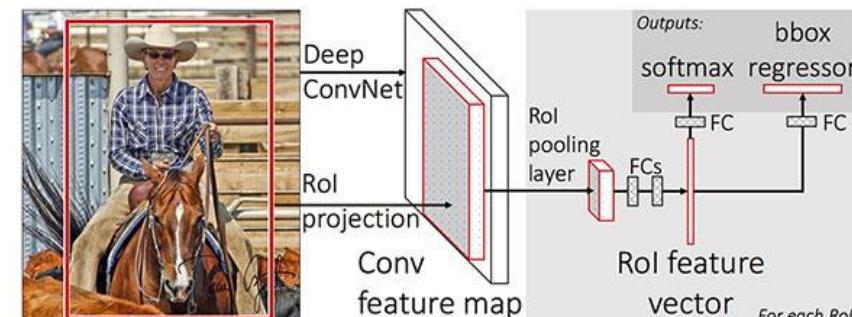
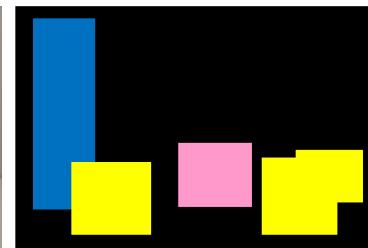


2. **Uso** de técnicas de **transfer learning** (extracción de características) sobre las **regiones candidatas** empleando una **arquitectura pre-entrenada (AlexNet)**.
3. **Clasificación** de cada región candidata empleando las **características extraídas** y **Support Vector Machine (SVM)**.



## Fast R-CNN: Regiones con características CNN

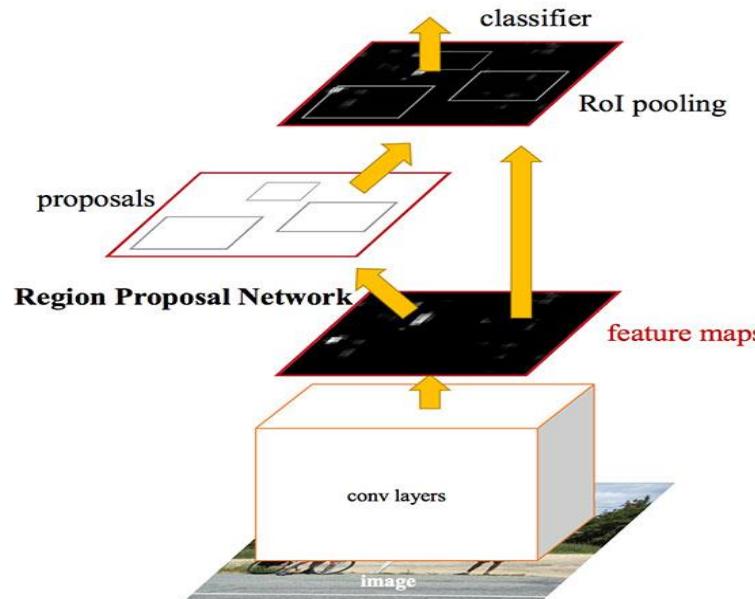
- **Gran desventaja R-CNN:** Tiene que clasificar todas las **regiones candidatas** que se extraen del método de búsqueda selectiva (i.e. **cuello de botella**).
- Girshick et al. (2015) proponen una **mejora de R-CNN** que trata de disminuir el alto coste computacional de dicho método. Proponen un **módulo de pooling sobre el mapa de activación** que obtienen al pasar la imagen por la red pre-entrenada (**proyecta ROIs**).
- El proceso de **clasificación** pasan de hacerlo con SVM a emplear un **perceptrón multicapa** que **predice la clase y la bounding box (offset)**. Consiguen un **aproximación entrenable end to end**.



- El **performance** del proceso de **inferencia** (i.e. fase de predicción) sigue sufriendo dramáticamente debido a la **dependencia** en el **algoritmo de búsqueda selectiva**.

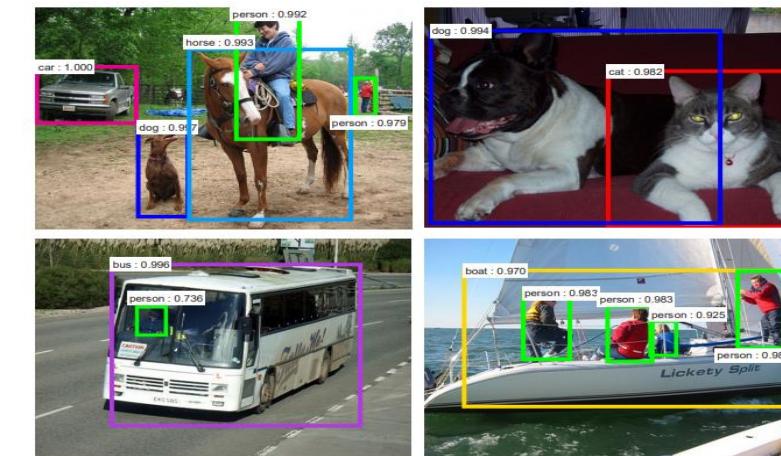
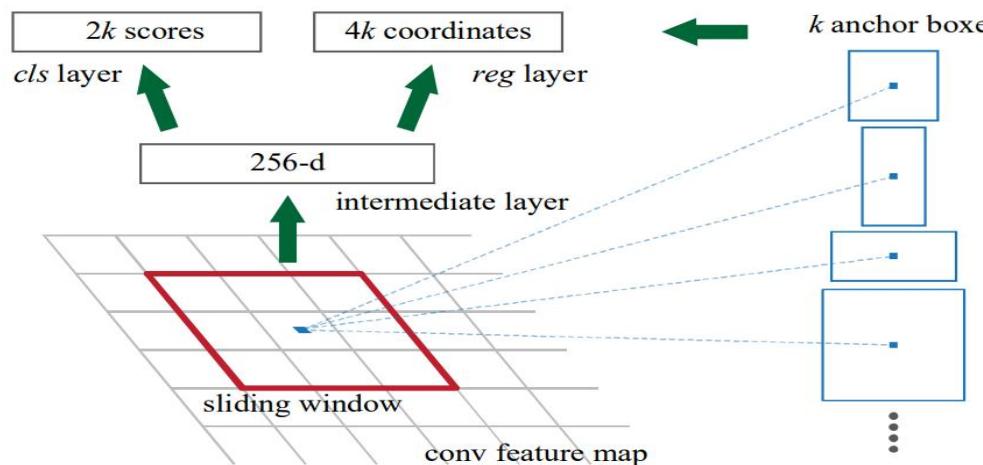
## Faster R-CNN: Regiones con características CNN

- En el artículo donde Girshick et al. (2015) proponen **Faster R-CNN**, presentan la **Region Proposal Network (RPN)** que introduce la propuesta de regiones directamente en la arquitectura (**sustituyendo** el algoritmo de **búsqueda selectiva**).
- Supone una contribución muy importante porque la **Faster R-CNN** es capaz de **predecir 7-10 FPS** haciendo posible la **detección de objetos en tiempo real**.



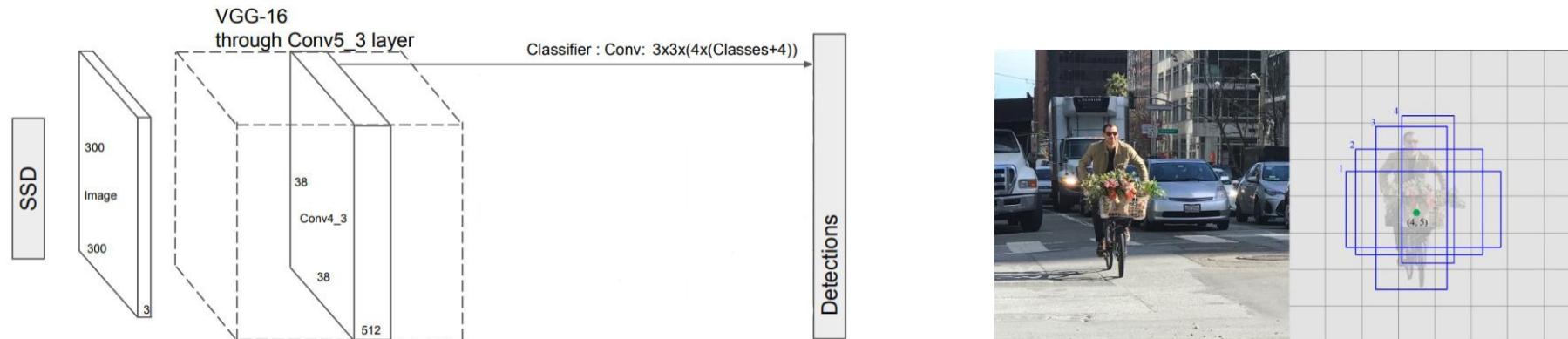
# Faster R-CNN: Regiones con características CNN

- Sobre el mapa de activación se pasa una ventana deslizante y se generan  $k$  candidatos o **anchors**. Dichos candidatos **se evalúan simultáneamente** en la **RPN**.
- La **RPN** es un **perceptrón multicapa** con dos ***fully-connected*** de salida. Una de ellas se encarga de dar una **predicción** de si la **región** contiene o no **contiene objeto** mientras que la otra se encarga de **predecir los bordes de la región**.
- En el artículo **emplean 3 escalas y 3 relaciones de aspecto** para generar  **$k=9$  anchors** a partir de una **ventana deslizante** de tamaño  **$3 \times 3$** .



# SSD: Single Shot Multibox Detector

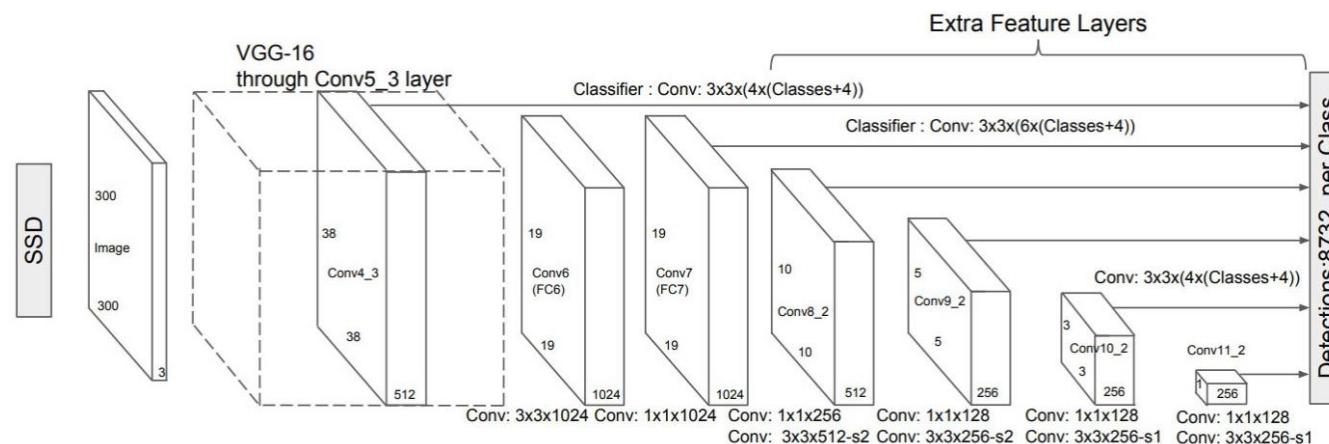
- Método propuesto por **Wei Liu** en **2016**. **Elimina** la necesidad de una **red generadora de candidatos**. La red se compone de dos niveles:
  - **Extracción de mapas de activación** (características) mediante **red pre-entrenada** (VGG16)
  - Aplicar **filtros convolucionales (3x3)** sobre el **mapa** de activación para detectar objetos.
- **Para cada** una de las **38 x 38 celdas** o localizaciones se obtienen **4 predicciones de objeto (4 filtros)** siguiendo una serie de ***bounding box* predefinidas** (a conciencia).



- Cada predicción está compuesta por una ***bounding box (offset)*** y **21 scores** de pertenencia a clase (**20 clases + fondo**).

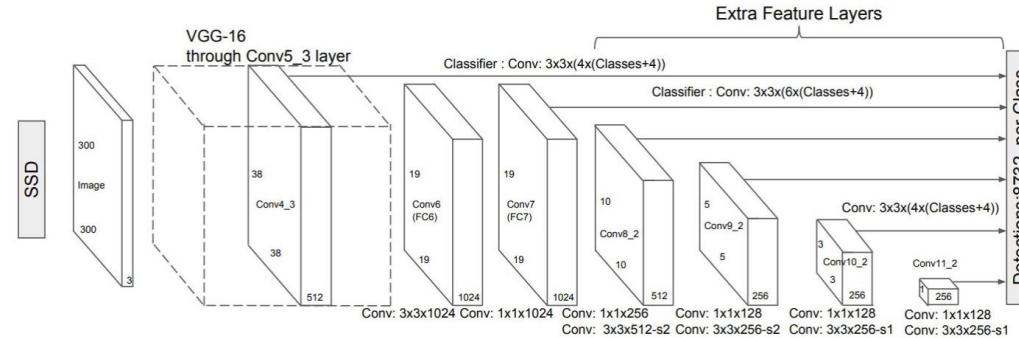
# SSD: Single Shot Multibox Detector

- La gran **potencia** de la red **SSD** reside en el uso de **múltiples capas** con el objetivo de realizar una **detección multiescala**.
- Concretamente **SSD añade 6 capas convolucionales** sobre el mapa de activación que obtiene de la red pre-entrenada y **sobre cinco de ellas aplica la detección de objetos** tal y como hemos visto anteriormente (en 3 de ellas hace **6 predicciones** por celda en vez de 4). Un total de **8732 predicciones**.

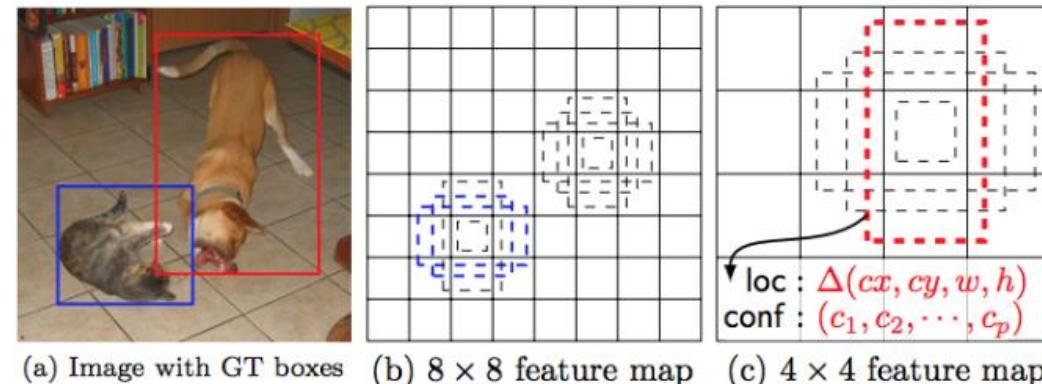


- Estas **capas convolucionales** van **disminuyendo las dimensiones del mapa de características** gradualmente ( $stride \geq 1$ ).

# SSD: Single Shot Multibox Detector

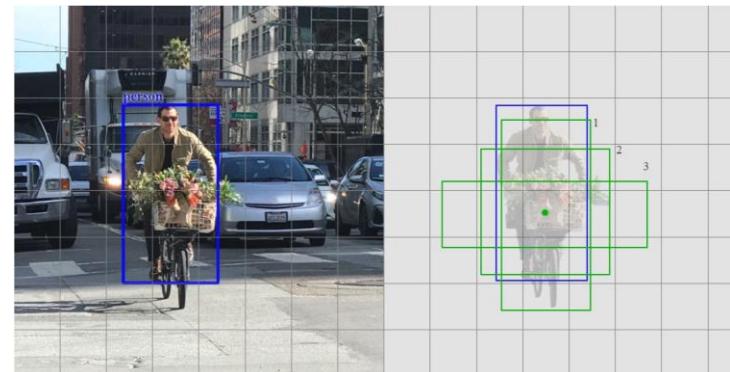


- SSD usa las **resoluciones bajas** para detectar **objetos grandes** mientras que los **objetos pequeños** son detectados en los **primeros mapas** de activación.



# SSD: Single Shot Multibox Detector

- En la **fase de entrenamiento**, el **coste para optimizar la localización** solo se calcula sobre las coincidencias positivas. Una **coincidencia positiva** se define por tener una **IoU > 0.5 entre la BB por defecto y la del GT.**



- Una vez identificadas las **coincidencias positivas**, estas se emplean para **calcular el error** con respecto a la **BB predecida**.
- En la **fase de test**, SSD utiliza la técnica **non-máximo suppression\*** (algoritmo basado en el **nivel de confidencia** y en la **IoU entre predicciones**) para **eliminar predicciones duplicadas** sobre un mismo objeto.

\* <https://pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/>

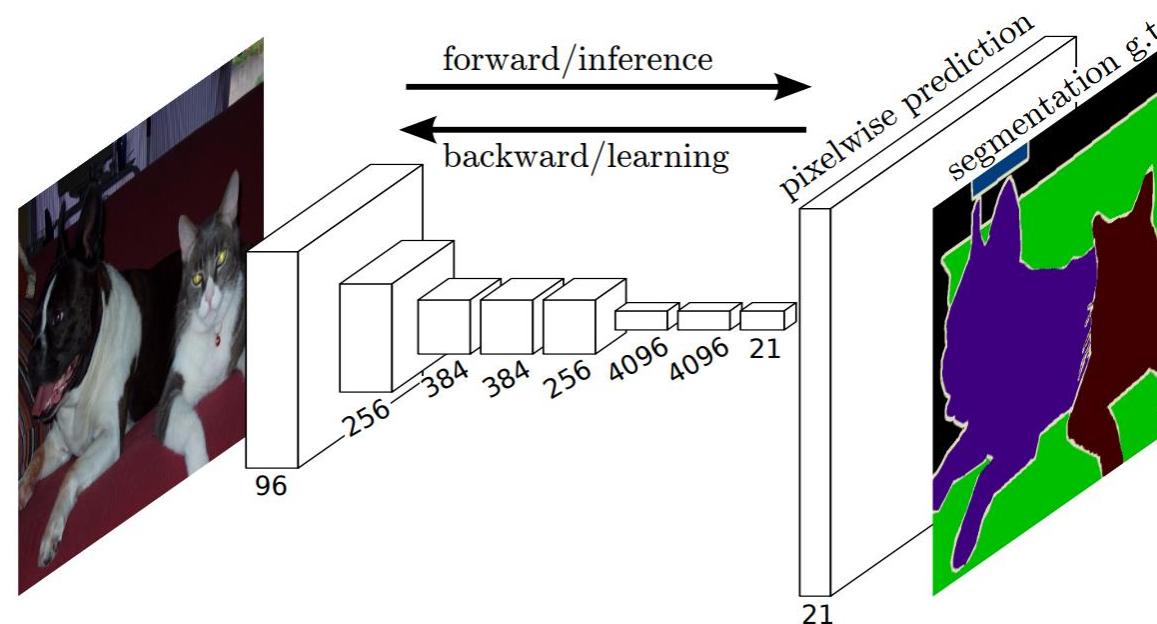
03

# Segmentación semántica y de instancia

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

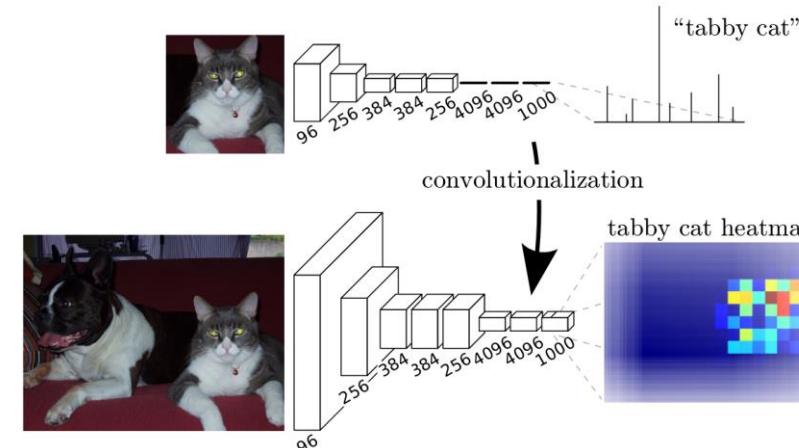
## FCNN para la segmentación semántica

- J. Long et al. (2015) fueron los primeros en utilizar una **arquitectura** basada exclusivamente en **capas convolucionales y de pooling** para realizar tareas de **segmentación semántica**.
- Este tipo de red recibe como **entrada** una **imagen** de un **determinado tamaño** y como **salida** devuelve la **imagen segmentada del mismo tamaño**.



# FCNN para la segmentación semántica

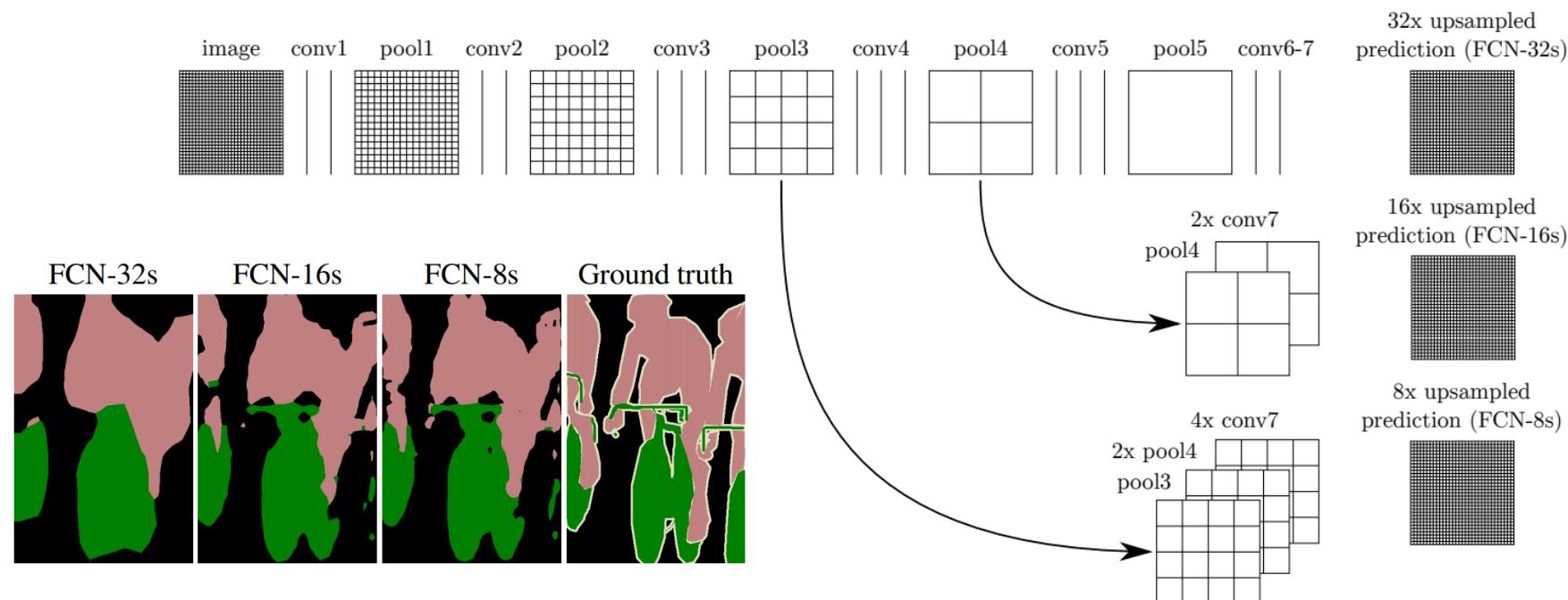
- Los autores **modifican** redes conocidas como AlexNet, VGG16 o GoogleNet **eliminando el *top model*** (destinado a clasificación) y **reemplazándolo** por más **bloques convolucionales** produciendo **pequeños mapas de características con representaciones densas**.



- Cuando se llega al final de la red, **al último mapa** se le debe aplicar un **upsampling** para **llevarlo a las dimensiones espaciales originales** de la imagen.
- Se emplea una **capa convolucional** con un valor de **stride = 1/f** consiguiendo **ampliar** el mapa de activación **por un factor f**. Esta técnica es conocida como **deconvolución**. Los **filtros** aprendidos en estas capas constituyen las **bases** para **reconstruir la forma**.

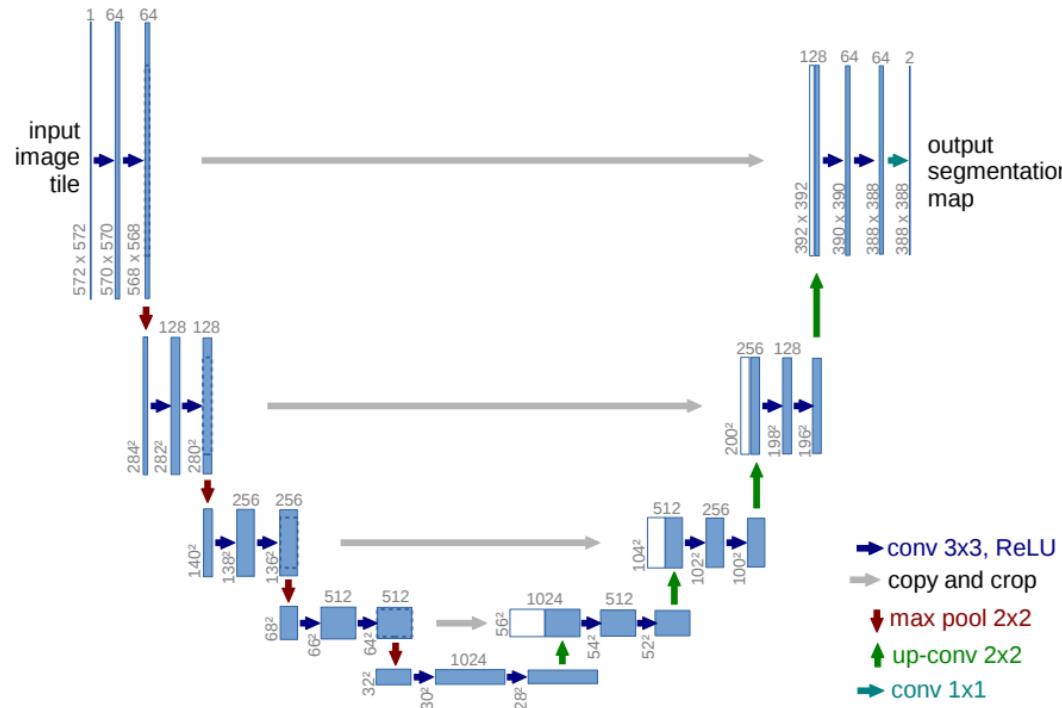
# FCNN para la segmentación semántica

- La **red es entrenada** empleando una **función de pérdidas a nivel de pixel** (por ejemplo **Dice**).
- Además, los autores introducen **skip connections** en la red con el objetivo de **combinar mapas de características** que **contienen representaciones de alto y bajo nivel** que posteriormente fusionan dando lugar a la segmentación final.



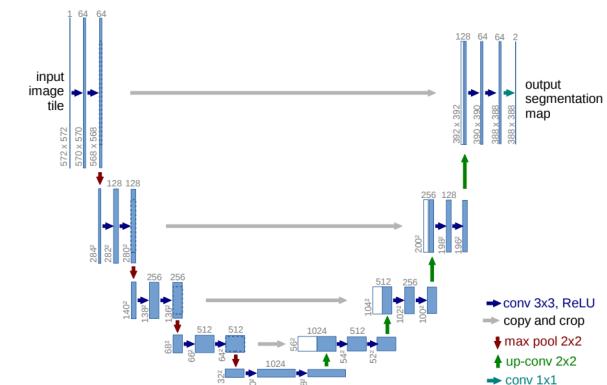
## Segmentación semántica: U-net

- O. Ronneberger et al. (2015) extienden el trabajo anterior a imágenes de microscopía. Los autores crean **U-net** que se compone de **dos partes**: el **contracting path** encargado de codificar la **información relevante** de la imagen y el **expansive path** que localiza **espacialmente** los patrones relevantes en la imagen.



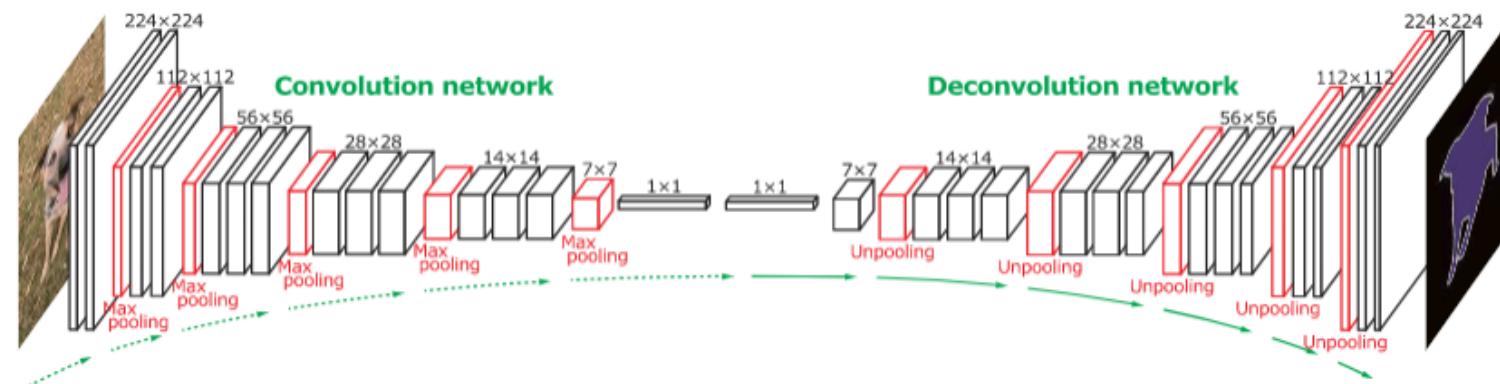
# Segmentación semántica: U-net

- El ***downsampling*** o ***contracting path*** esta **basado** en una **FCN** con **tamaño de filtros constante** (i.e. **3x3**).
- El ***upsampling*** o ***expansive path*** emplea ***up-convolutions*** (i.e. deconvoluciones, convoluciones transpuestas) **reduciendo el número de mapas de características** a la vez que **aumenta el tamaño espacial**.
- Una **importante contribución** es que para **evitar pérdida de información**, **copian** los **mapas** de cada **bloque convolucional del *encoder*** en su correspondiente **mapa en el *decoder***. Esta **información se concatena** en la **tercera dimensión** gracias a las ***skip connections***.
- Por último, una **capa convolucional 1x1** **mapea** las **dimensiones** del **último mapa de activación** a tantos **mapas 2D** como **clases** se tengan. Posteriormente, aplicando la función ***softmax*** se **categoriza** cada uno de los **pixels**, obteniendo la **segmentación semántica**.



## Segmentación de instancia: Redes Conv-Deconv

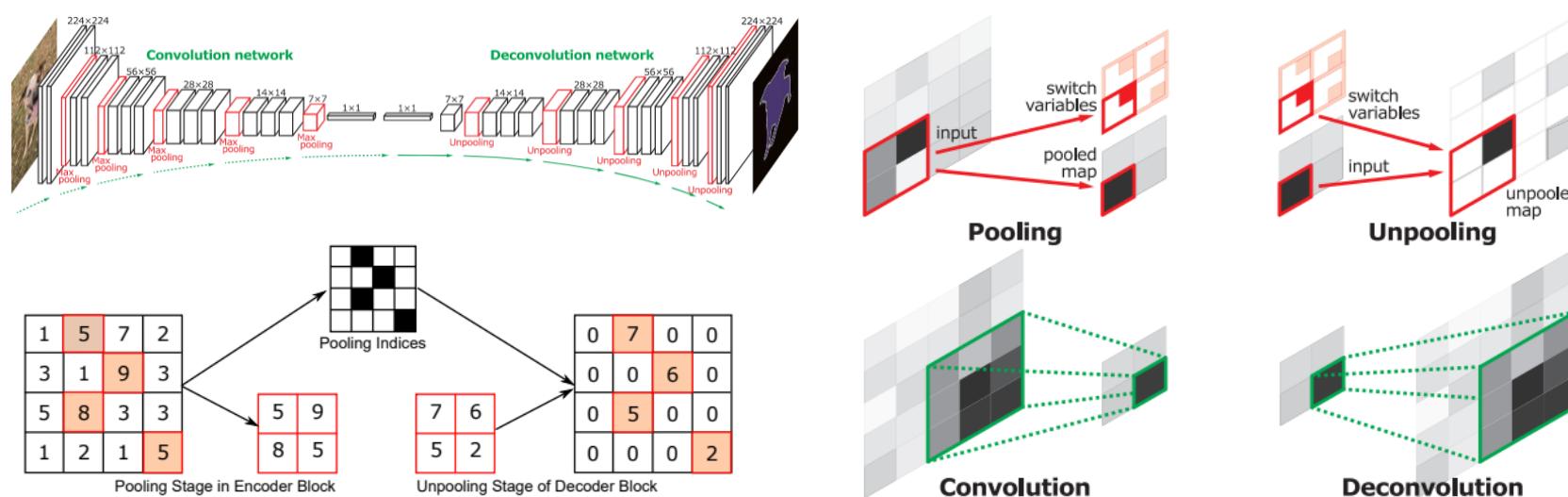
- H. Noh et al. (2015) diseñan una **red end to end** compuesta por dos niveles para realizar segmentación de instancia. La primera de ellas es un módulo detector de objetos mediante VGG16.
- La **red** que proponen para la **segmentación** se compone de **dos fases**, la de **convolución** y la **deconvolución**.



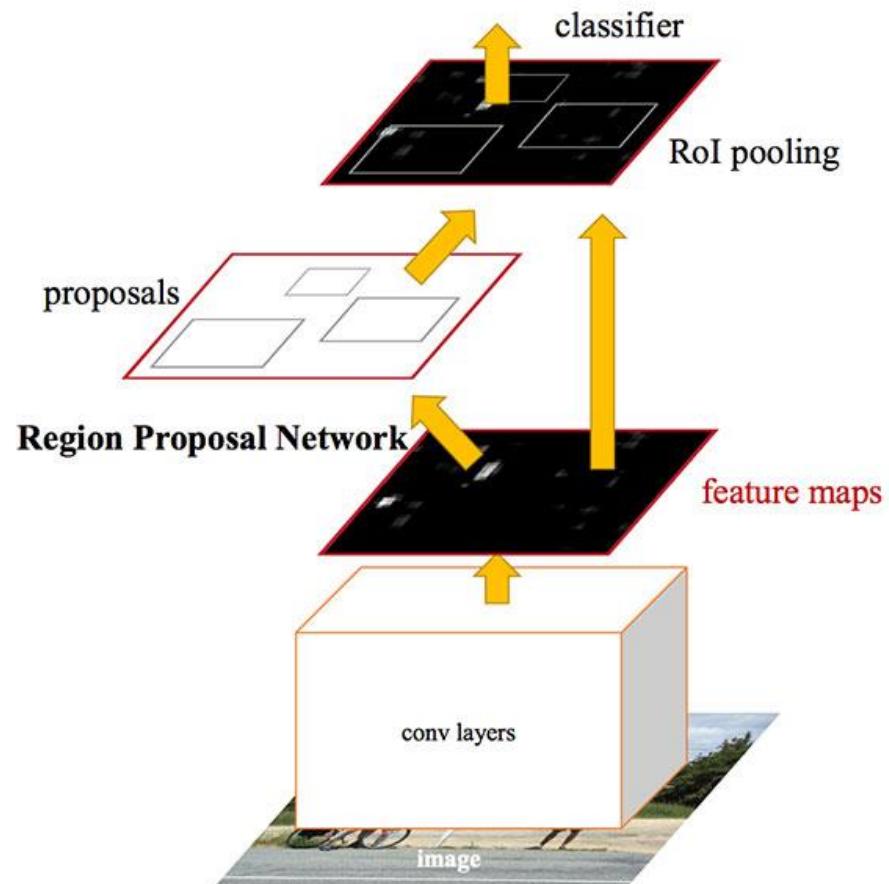
- Una **región candidata** es procesada por el **encoder** generando un **vector de características**.

# Segmentación de instancia: Redes Conv-Deconv

- El **decoder** toma dicho vector de características y **genera un mapa a nivel de pixel con la probabilidad de pertenencia a clase.**
- La **subred de deconvolución** emplea la operación ***unpooling*** localizando las activaciones máximas para **mantener su posición original en el mapa de características** que va ampliando. **Tras la capa *unpooling* se insertan varias capas deconvolucionales** para **mantener la densidad de la información.**

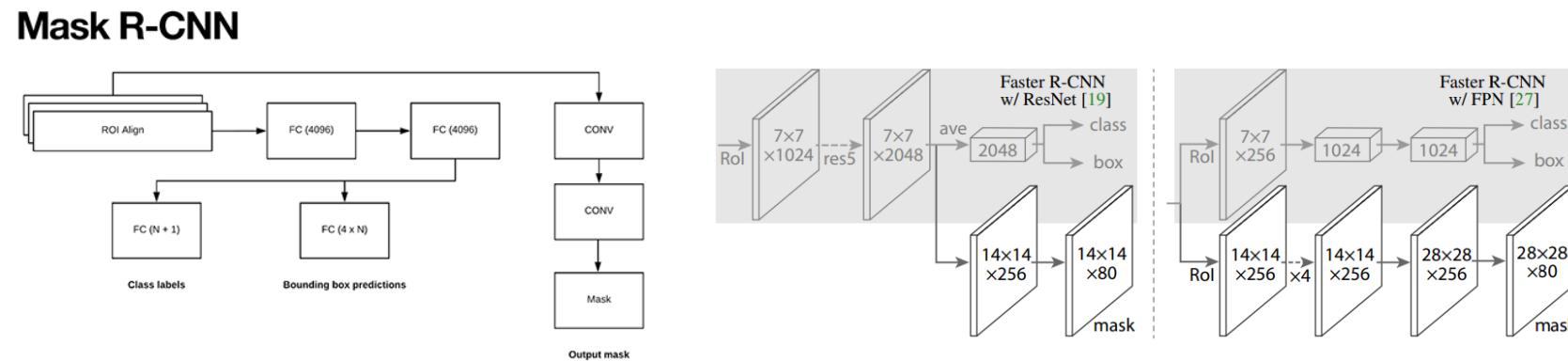


## Faster R-CNN: Regiones con características CNN



## Mask R-CNN para la segmentación de instancia

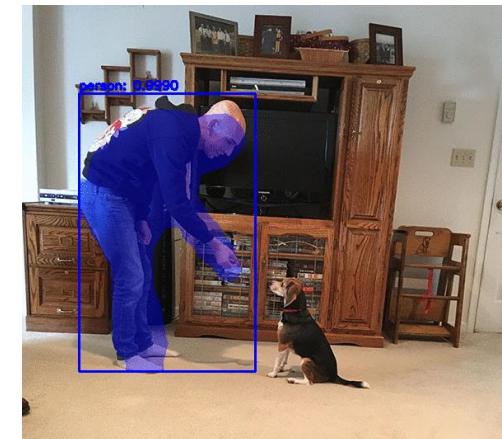
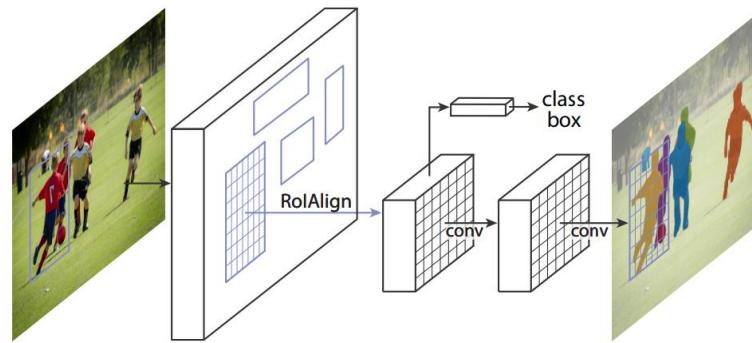
- La Mask R-CNN es una arquitectura para la **segmentación de instancia** fue propuesta por He et al. (2018) y nace a partir de la Faster R-CNN:
  - Se reemplaza el módulo ***ROI pooling*** por el módulo ***ROI align*** mucho **más preciso** para el propósito de **segmentación**.
  - Se **inserta una rama adicional** a la salida del nuevo módulo que realiza la **segmentación**.



- La **salida** de la **rama convolucional** es la **máscara** de la segmentación del **objeto** previamente detectado (objeto vs fondo), se obtiene aplicando la función **sigmoide** a cada **pixel** del **último mapa** de activación.

# Mask R-CNN para la segmentación de instancia

- La **Mask R-CNN** es una arquitectura para la **segmentación de instancia** fue propuesta por **He et al.** (2018) y nace a partir de la Faster R-CNN:
  1. Se reemplaza el módulo ***ROI pooling*** por el módulo ***ROI align*** mucho **más preciso** para el propósito de **segmentación**.
  2. Se inserta una **rama adicional** a la salida del nuevo módulo que realiza la **segmentación**.



- La **salida de la rama convolucional** es la **máscara** de la segmentación del **objeto** previamente detectado (objeto vs fondo), se obtiene aplicando la función **sigmoide** a cada **pixel** del **último mapa** de activación.

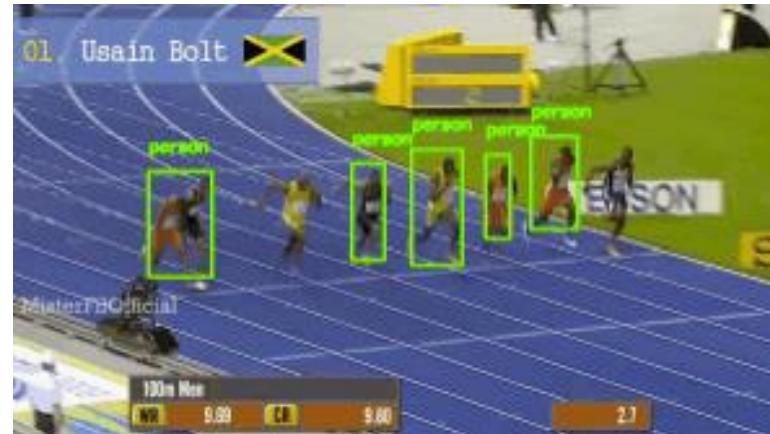
04

# Tracking de objetos

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

# Introducción

- La tarea de **tracking** se basa en aplicar la técnica de **detección de objetos** (DO) durante una serie de **frames** consecutivos. La problemática reside en el **tiempo de procesado** del algoritmo de DO para realizar dicha tarea en **tiempo real**.



## Faster R-CNN vs SSD

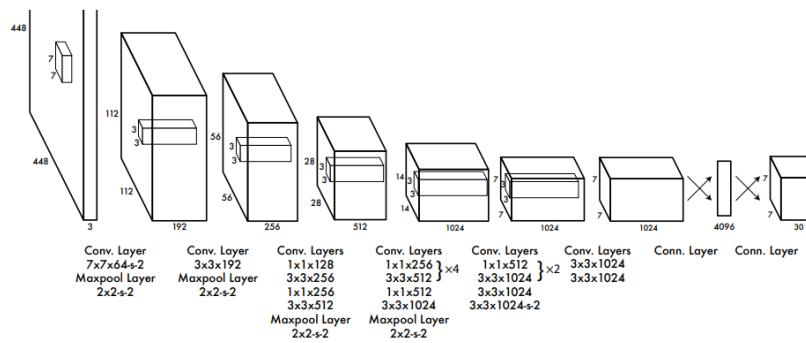
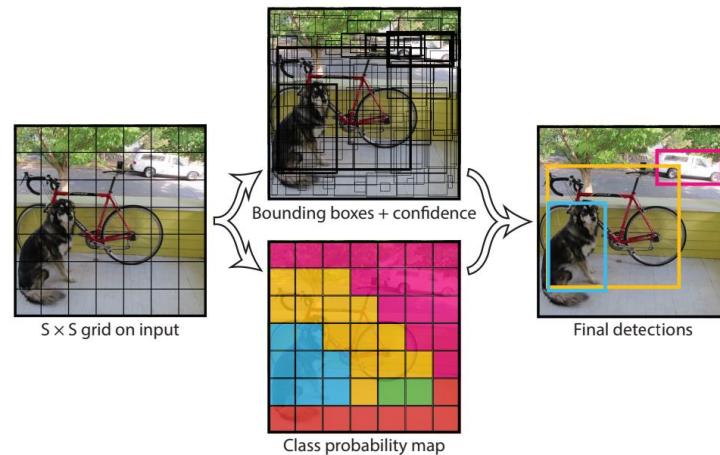
- Si rescatamos las dos metodologías basadas en aprendizaje profundo más comunes de la literatura (i.e. **Faster R-CNN** y **SSD** en sus múltiples versiones) y las **analizamos** desde el punto de vista tanto de **precisión** como de **frames por segundo (FPS)** procesados se puede observar que **Faster R-CNN** es un **método de detección de objetos estático** o que únicamente se podría aplicar a tracking en videos *time-lapse*.

| Method               | mAP  | FPS | batch size | # Boxes | Input resolution |
|----------------------|------|-----|------------|---------|------------------|
| Faster R-CNN (VGG16) | 73.2 | 7   | 1          | ~ 6000  | ~ 1000 × 600     |
| SSD300               | 74.3 | 46  | 1          | 8732    | 300 × 300        |
| SSD512               | 76.8 | 19  | 1          | 24564   | 512 × 512        |
| SSD300               | 74.3 | 59  | 8          | 8732    | 300 × 300        |
| SSD512               | 76.8 | 22  | 8          | 24564   | 512 × 512        |

- Compromiso entre precisión** en la detección y **FPS** procesados. Por este motivo nace **YOLO** (You Only Look Once).

# YOLO: You Only Look Once

- Introducida por Redmon et al. (2015) se basa en una **estrategia single-stage** (i.e. una única red para todo el proceso, al igual que SSD).
- **División de la imagen en un grid  $S \times S$ .** Para cada celda **se predicen  $B$  bounding boxes** y un **nivel de confidencia** (i.e. si hay o no objeto y que objeto de todas las clases es).
- **CNN para extraer las características**, las **predicciones** se realizan mediante **dos FC layers** después del último bloque convolucional.



# YOLO: You Only Look Once

- Los autores afirman que YOLO realiza una detección de objetos ***super real-time*** obteniendo **45 FPS** en una GPU. Desarrollan también una **variante más ligera** que corre a **155 FPS**.
- Existen YOLOv2 (2016) y YOLOv3 (2018) que mejoran la precisión en la detección de la primera versión ya que se entrena en **datasets más grandes** (i.e. COCO).

| Real-Time Detectors     |             | Train       | mAP         | FPS         |
|-------------------------|-------------|-------------|-------------|-------------|
| 100Hz DPM [31]          |             | 2007        | 16.0        | 100         |
| 30Hz DPM [31]           |             | 2007        | 26.1        | 30          |
| Fast YOLO               | 2007+2012   | 52.7        | <b>155</b>  |             |
| YOLO                    | 2007+2012   | <b>63.4</b> | 45          |             |
| Less Than Real-Time     |             |             |             |             |
| Fastest DPM [38]        |             | 2007        | 30.4        | 15          |
| R-CNN Minus R [20]      |             | 2007        | 53.5        | 6           |
| Fast R-CNN [14]         | 2007+2012   | 70.0        | 0.5         |             |
| Faster R-CNN VGG-16[28] | 2007+2012   | 73.2        | 7           |             |
| Faster R-CNN ZF [28]    | 2007+2012   | 62.1        | 18          |             |
| YOLO VGG-16             | 2007+2012   | 66.4        | 21          |             |
| Backbone                |             | Top-1       | Top-5       | Bn Ops      |
| Darknet-19 [15]         |             | 74.1        | 91.8        | 7.29        |
| ResNet-101[5]           |             | 77.1        | 93.7        | 19.7        |
| ResNet-152 [5]          | <b>77.6</b> | <b>93.8</b> | 29.4        | 1090        |
| Darknet-53              |             | 77.2        | <b>93.8</b> | 18.7        |
|                         |             |             |             | <b>1457</b> |
|                         |             |             |             | 78          |

| Type          | Filters | Size/Stride    | Output           |
|---------------|---------|----------------|------------------|
| Convolutional | 32      | $3 \times 3$   | $224 \times 224$ |
| Maxpool       |         | $2 \times 2/2$ | $112 \times 112$ |
| Convolutional | 64      | $3 \times 3$   | $112 \times 112$ |
| Maxpool       |         | $2 \times 2/2$ | $56 \times 56$   |
| Convolutional | 128     | $3 \times 3$   | $56 \times 56$   |
| Convolutional | 64      | $1 \times 1$   | $56 \times 56$   |
| Convolutional | 128     | $3 \times 3$   | $56 \times 56$   |
| Maxpool       |         | $2 \times 2/2$ | $28 \times 28$   |
| Convolutional | 256     | $3 \times 3$   | $28 \times 28$   |
| Convolutional | 128     | $1 \times 1$   | $28 \times 28$   |
| Convolutional | 256     | $3 \times 3$   | $28 \times 28$   |
| Maxpool       |         | $2 \times 2/2$ | $14 \times 14$   |
| Convolutional | 512     | $3 \times 3$   | $14 \times 14$   |
| Convolutional | 256     | $1 \times 1$   | $14 \times 14$   |
| Convolutional | 512     | $3 \times 3$   | $14 \times 14$   |
| Convolutional | 256     | $1 \times 1$   | $14 \times 14$   |
| Convolutional | 512     | $3 \times 3$   | $14 \times 14$   |
| Maxpool       |         | $2 \times 2/2$ | $7 \times 7$     |
| Convolutional | 1024    | $3 \times 3$   | $7 \times 7$     |
| Convolutional | 512     | $1 \times 1$   | $7 \times 7$     |
| Convolutional | 1024    | $3 \times 3$   | $7 \times 7$     |
| Convolutional | 512     | $1 \times 1$   | $7 \times 7$     |
| Convolutional | 1024    | $3 \times 3$   | $7 \times 7$     |
| Convolutional | 1000    | $1 \times 1$   | $7 \times 7$     |
| Avgpool       |         | Global         | 1000             |
| Softmax       |         |                |                  |

DarkNet-19 en YOLOv2

| Type          | Filters | Size             | Output           |
|---------------|---------|------------------|------------------|
| Convolutional | 32      | $3 \times 3$     | $256 \times 256$ |
| Convolutional | 64      | $3 \times 3 / 2$ | $128 \times 128$ |
| Convolutional | 32      | $1 \times 1$     |                  |
| Convolutional | 64      | $3 \times 3$     |                  |
| Residual      |         |                  | $128 \times 128$ |
| Convolutional | 128     | $3 \times 3 / 2$ | $64 \times 64$   |
| Convolutional | 64      | $1 \times 1$     |                  |
| Convolutional | 128     | $3 \times 3$     |                  |
| Residual      |         |                  | $64 \times 64$   |
| Convolutional | 256     | $3 \times 3 / 2$ | $32 \times 32$   |
| Convolutional | 128     | $1 \times 1$     |                  |
| Convolutional | 256     | $3 \times 3$     |                  |
| Residual      |         |                  | $32 \times 32$   |
| Convolutional | 512     | $3 \times 3 / 2$ | $16 \times 16$   |
| Convolutional | 256     | $1 \times 1$     |                  |
| Convolutional | 512     | $3 \times 3$     |                  |
| Residual      |         |                  | $16 \times 16$   |
| Convolutional | 1024    | $3 \times 3 / 2$ | $8 \times 8$     |
| Convolutional | 512     | $1 \times 1$     |                  |
| Convolutional | 1024    | $3 \times 3$     |                  |
| Residual      |         |                  | $8 \times 8$     |
| Avgpool       |         | Global           |                  |
| Connected     |         | 1000             |                  |
| Softmax       |         |                  |                  |

DarkNet-53 en YOLOv3



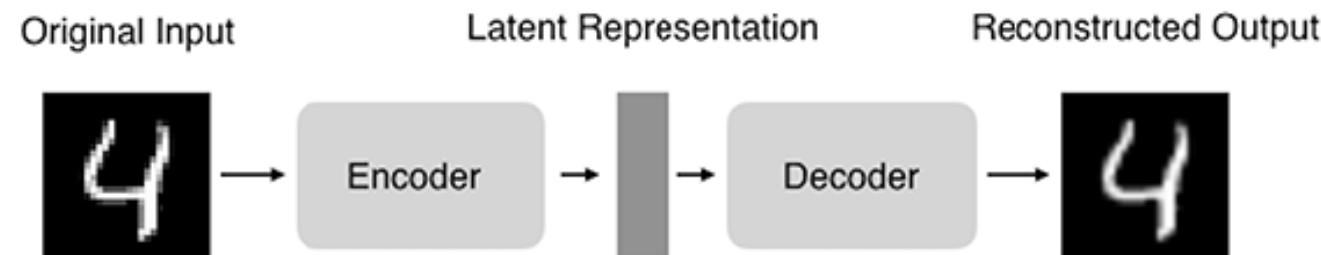
05

# Otras tareas

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

# Autoencoders

- Los **autoencoders** son un tipo de red neuronal no supervisada que tienen como objetivo comprimir los datos de entrada en una representación denominada **espacio latente**.
- El **espacio latente** se caracteriza por una **dimensionalidad mucho menor** que la dimensionalidad de los **datos de entrada**.
- La idea de un autoencoder es **reconstruir los datos** a partir del espacio latente entrenado **minimizando cierta función de error** (MSE, MAE, etc.).



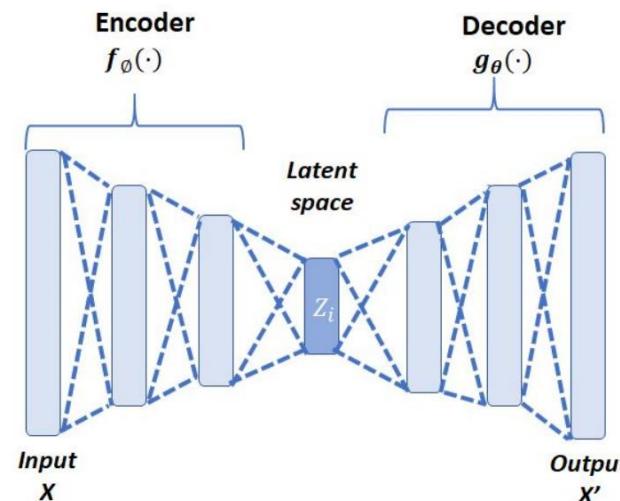
## Aplicaciones autoencoders

- **Reducción de dimensionalidad:** Mapeo **no lineal** de los datos de entrada a un nuevo espacio vectorial que describe gran parte de la variabilidad del set de datos de entrada.
- **Denoising:** Eliminar ruido de un set de imágenes a la entrada. El **encoder** caracteriza la **distribución del ruido** y el **decoder** es capaz de generar **muestras sin dicho ruido** gracias al espacio latente generado.
- **Compresión de datos:** Generar **nuevas representaciones de datos reducidas** a partir del espacio latente.
- **Detección de anomalías/outliers:** Detectar **datos clasificados erróneamente** o detectar cuando un **dato** a la entrada **no sigue la distribución** típica de la población.
- Sistemas **Content Based Image Retrieval (CBIR)**: Creación de sistemas para la recuperación automática de información. Basado en similitudes entre una **Query** y un **diccionario de representaciones**.
- **Natural Language Processing:** Comprensión de texto, construcción de Word embeddings, o resumen de textos.

## Autoencoder convencional

Una red **autoencoder** esta caracterizada por **dos niveles o subredes** (al igual que las arquitecturas para segmentación de imagen):

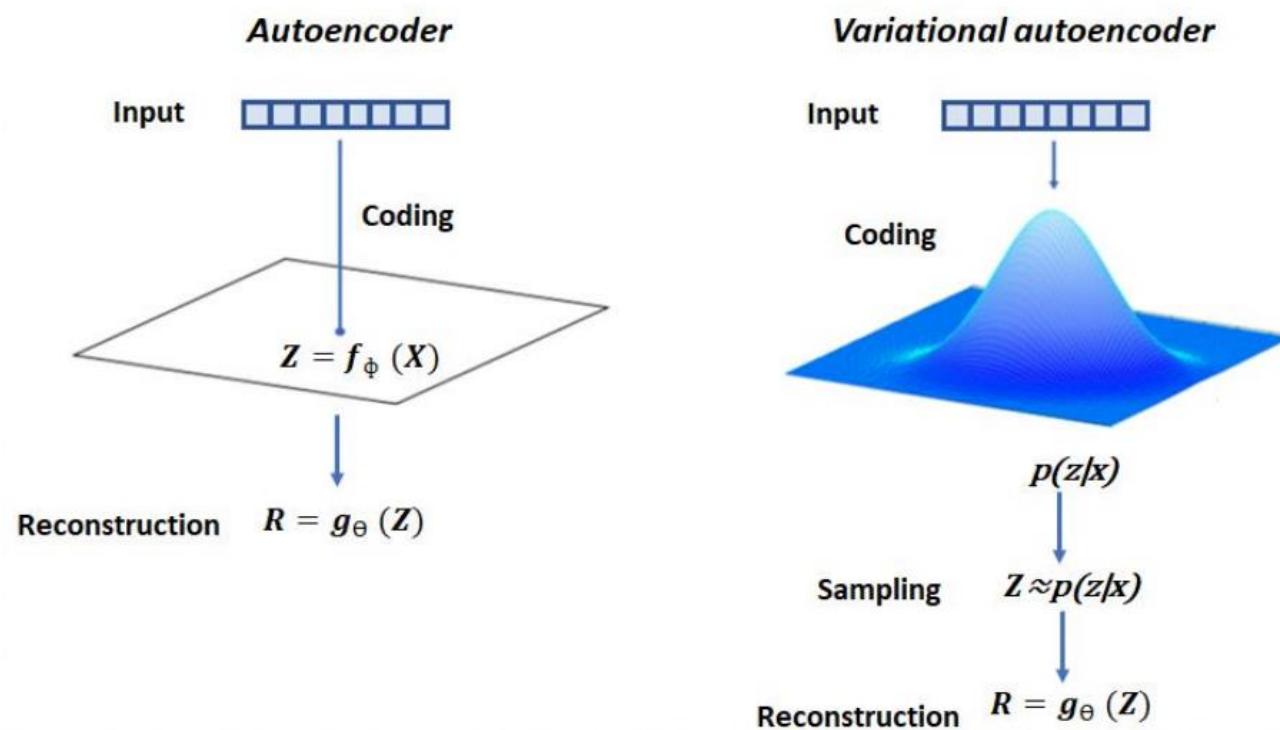
- **Encoder -  $f_\theta(\cdot)$** : Comprime los datos de entrada en un espacio latente ( $Z_i$ ) mediante  $Z_i = f_\theta(X)$ .
- **Decoder -  $g_\theta(\cdot)$** : Tiene como entrada el espacio latente ( $Z_i$ ) y se encarga de reconstruir una imagen de salida a partir de este según  $g_\theta(Z_i)$ .
- El **proceso completo** de un **autoencoder** queda definido por  $g_\theta(f_\theta(X))$  y el proceso de optimización se basa en **minimizar el error** entre los datos reconstruidos y los originales.



$$\min_{\theta, \phi} L_{rec} = \min \frac{1}{n} \sum_{i=1}^n \|x_i - g_\theta(f_\phi(x_i))\|^2$$

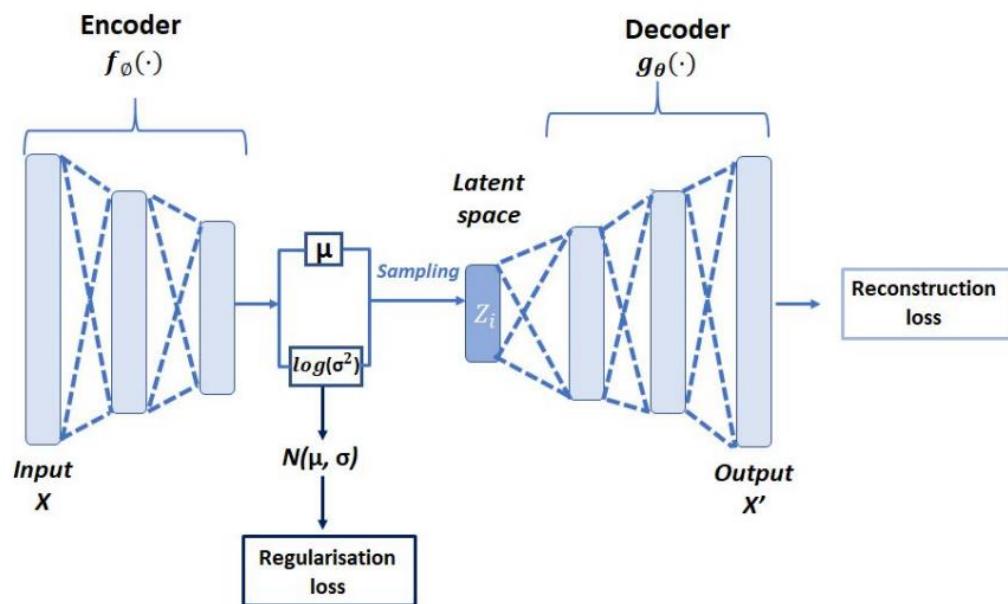
# Autoencoder variacional

- La versión variacional de un autoencoder (**VAE** del inglés) introduce una **regularización** en el espacio latente para mejorar sus propiedades. VAE codifica los datos de entrada como una **distribución normal multivariante** alrededor de un punto en el espacio latente.



# Autoencoder variacional

- El **encoder** asigna cada muestra de entrada a un vector de medias y otro de varianzas.
- Necesidad de **regularizar** tanto el logaritmo de la varianza como la media de la distribución que devuelve el encoder → **Match** entre **distribución** que saca el **encoder** y una **distribución normal estándar** (media cero y desviación unidad).
- **Sampling** de la **distribución multivariante** para reconstruir los datos originales.



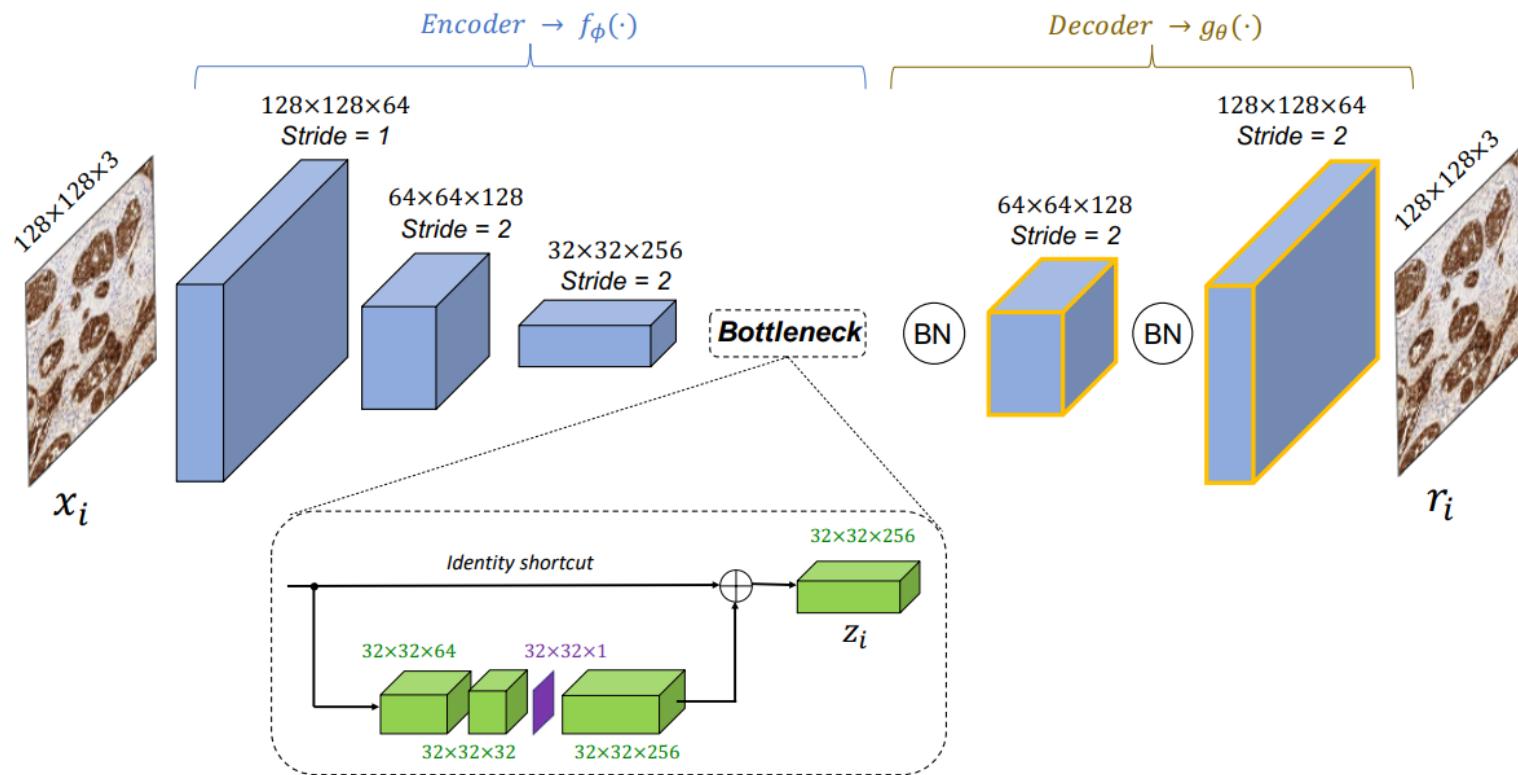
$$Z \approx p(z|x) = \mu + \sigma \cdot \epsilon$$

$$\min_{\theta, \phi} L_{rec} = \min \frac{1}{n} \sum_{i=1}^n \|x_i - g_\theta(f_\phi(x_i))\|^2$$

$$D_{KL}[N(\mu, \sigma) || N(0, 1)] = \frac{1}{2} \sum (1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

# Autoencoder convolucional

- La arquitectura de los **autoencoders** varían según el caso de uso, más concretamente según el tipo de datos a la entrada.





**viu**

**Universidad  
Internacional  
de Valencia**

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

De:  
 Planeta Formación y Universidades

# 07MIAR – Redes Neuronales y Deep Learning



**Universidad**  
Internacional  
de Valencia

Deep Learning para texto y secuencias lógicas

# Contenidos

---

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a transformers

# Contenidos

---

- 1. Procesamiento del Lenguaje Natural (NLP)**
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a *transformers*

# Procesamiento del Lenguaje Natural (NLP)

---

- El **Natural Language Processing** (NLP) es la **intersección** entre los campos de las **ciencias de la computación**, la **inteligencia artificial** y la **lingüística**. El NLP estudia las **interacciones** entre las **computadoras** y el **lenguaje humano**
- Se ocupa de la formulación e investigación de **mecanismos eficaces** computacionalmente para la **comunicación** entre **personas y máquinas** por medio del lenguaje natural, es decir, de las **lenguas del mundo**
- Dentro del NLP se identifican una serie de **tareas** a resolver:

- Speech Recognition
- Speech to text
- Machine Translation
- Text classification
- Sentiment analysis
- Text generation (e.g. Q&A)
- Text recognition (OCR)
- Image/Video understanding
- Text summarization



# Contenidos

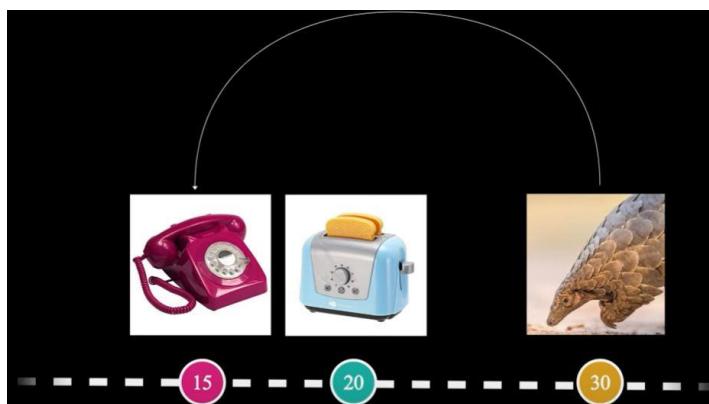
---

1. Procesamiento del Lenguaje Natural (NLP)
- 2. De texto a representaciones numéricas**
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a *transformers*

# De texto a representaciones numéricicas

---

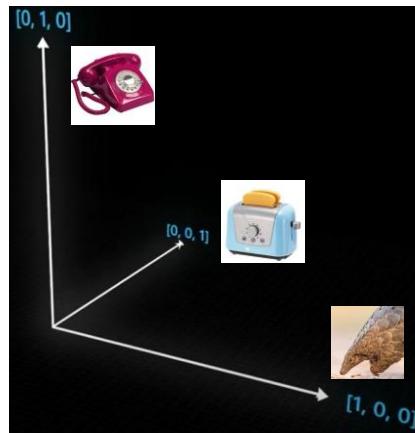
- Tal y como hemos visto a lo largo del curso, una red neuronal (independientemente de su arquitectura) viene definida por una serie de parámetros numéricos a optimizar. Dichos parámetros restringen el tipo de datos a la entrada. Una red neuronal no sabe manejar texto a la entrada
- La solución más intuitiva es asignar un número entero (etiqueta categórica) a cada una de las unidades de texto con las que se trabaje (carácter/palabra)
- Esta solución no es válida desde el punto de vista de una red neuronal puesto que la red intrínsecamente extrae patrones de ordenación de los datos numéricos mientras que el texto sigue una secuencia lógica gramatical.



# De texto a representaciones numéricas

---

- Una solución para dotar de **independencia** a la codificación texto – dato numérico es la codificación **one-hot encoding**. Mediante esta codificación podemos establecer **una dimensión a cada carácter/palabra**
- Como sabemos esta codificación se compone de un **vector** de tantos **ceros** como palabras distintas se quieran codificar, indicando **con un 1** cada palabra diferente



dog = [0 0 0 0 0 0 **1** 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

cat = [0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 **1** 0 0 0 0 0]

# De texto a representaciones numéricas

---

- El problema de este tipo de codificación es que en NLP vamos a manejar grandes vocabularios de texto (**Corpus**) para el entrenamiento de modelos

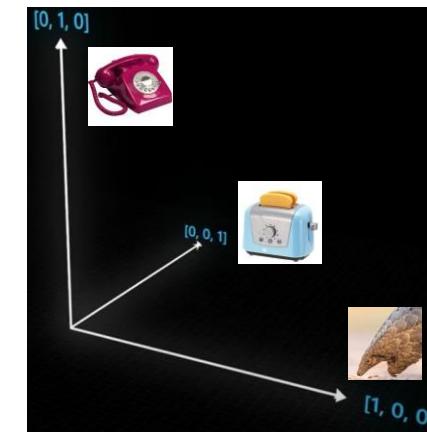
dog = [0 0 0 0 0 0 **1** 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

cat = [0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 **1** 0 0 0 0 0 0]



Vocabulary size

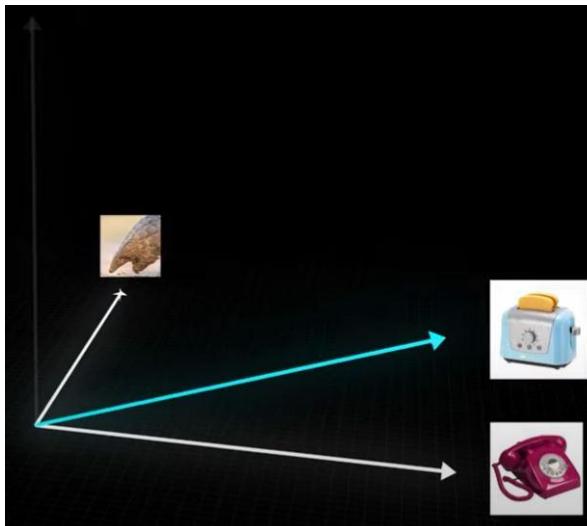
- Otro inconveniente es que palabras similares no tienen porque estar representadas por vectores cercanos
- Tantas dimensiones como palabras a codificar
- Vectores muy dispersos (todo ceros menos un uno)



# Word vectors/embeddings

---

- ¿Cómo podemos evitar las limitaciones anteriores?
  - IDEA: Mapear palabras en un nuevo **espacio vectorial** más **denso** o concentrado (**Word embeddings**)
  - OBJETIVO: Obtener una representación de palabras que obtenga provecho de la “**similitud semántica**” de las mismas
  - ¿CÓMO?: Resolviendo un problema de **aprendizaje Supervisado**
  - VENTAJAS: **Vectores** más pequeños y **compactos**



# Contenidos

---

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
- 3. Word2Vec**
4. Redes neuronales recurrentes
5. Introducción a *transformers*

# Word2Vec

---

- ¿Cómo podemos obtener estas representaciones densas de palabras (**Word embeddings**)?
- El **algoritmo Word2Vec** es el más popular en la literatura para entrenar **Word embeddings** con propósitos generalistas
- Se basa en **predecir el vecindario de palabras** para cada una de las palabras que componen un texto
- Existen **dos versiones** de este algoritmo:
  - Continuous bag of words (**CBOW**)
  - Modelo **Skip-gram**



Tomáš Mikolov

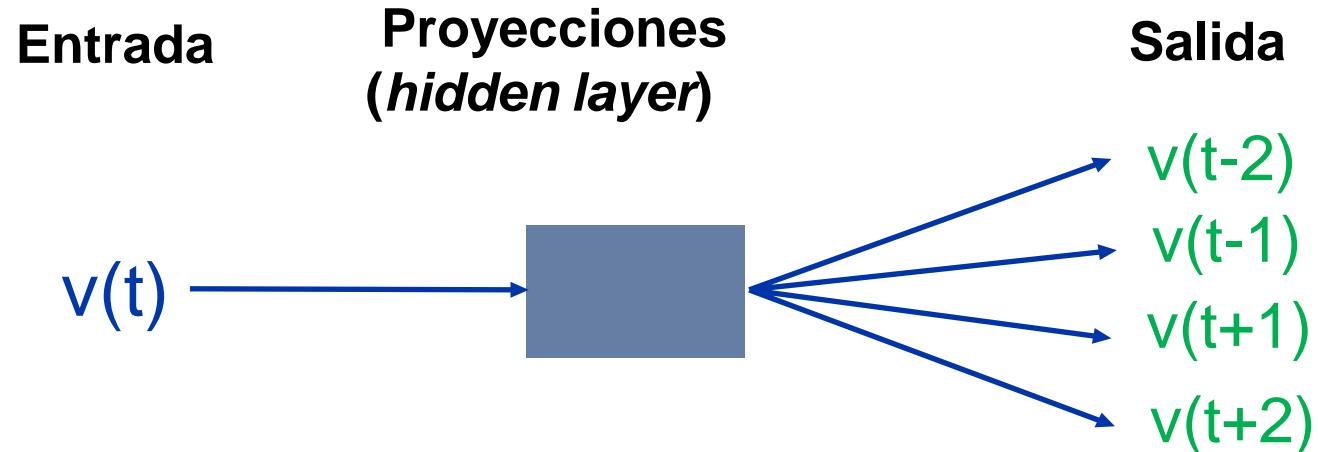
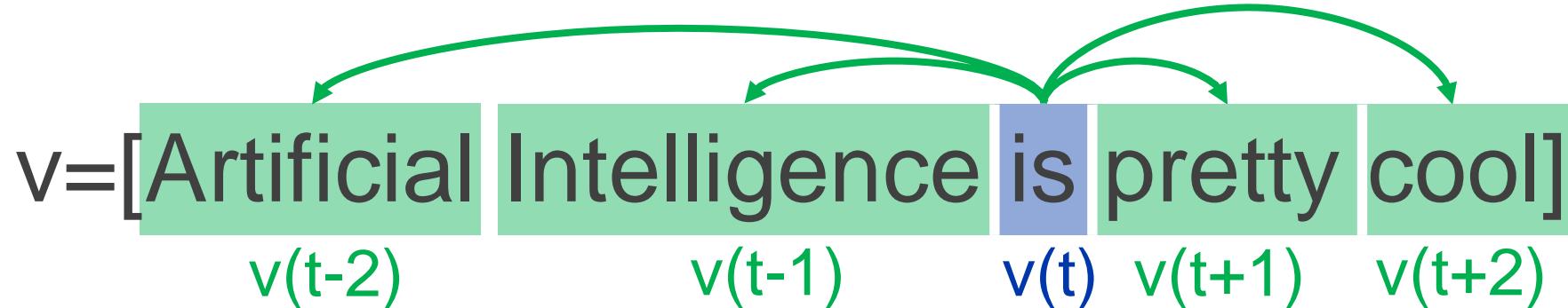
Facebook member  
Google ex-member

Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*

Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*

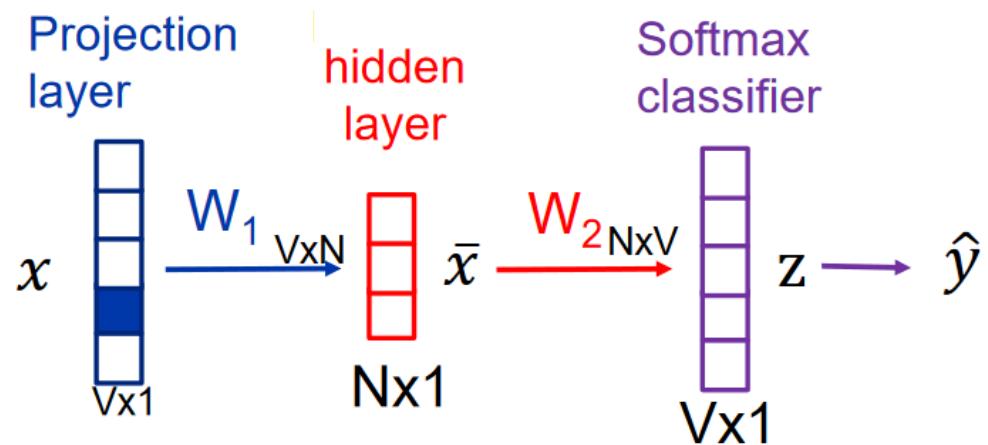
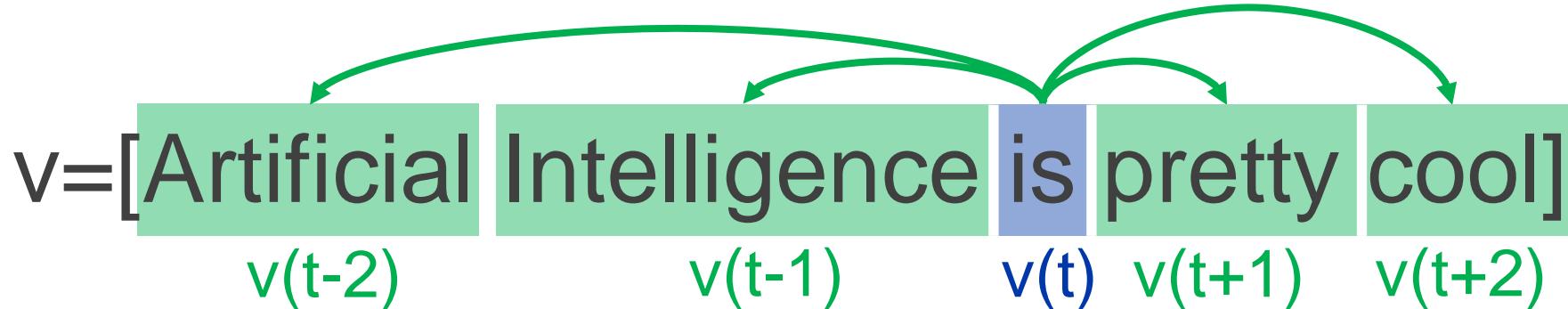
## Modelo Skip-gram

---



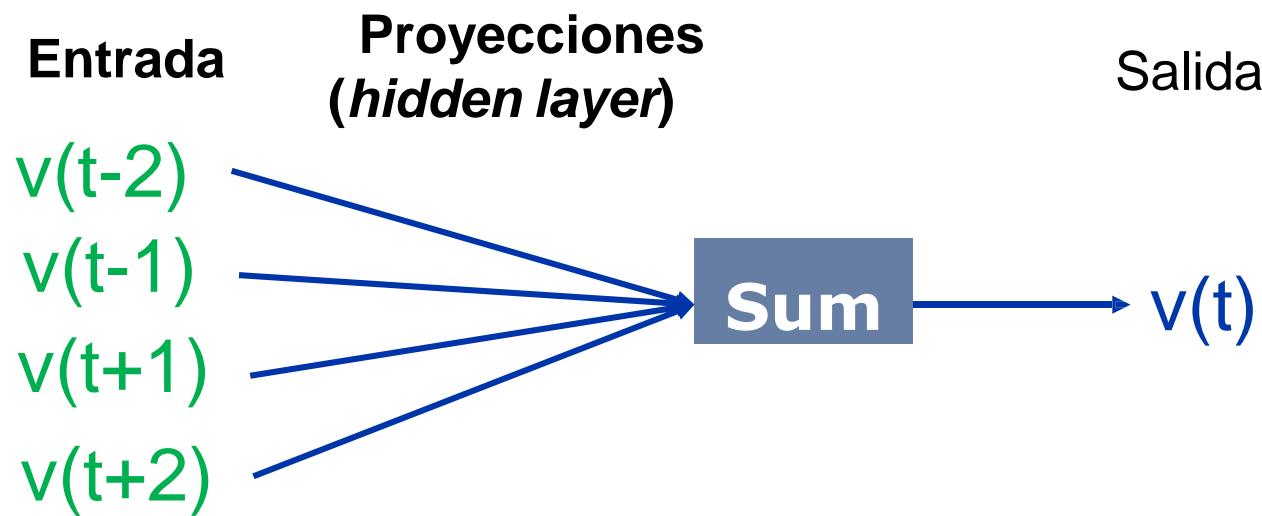
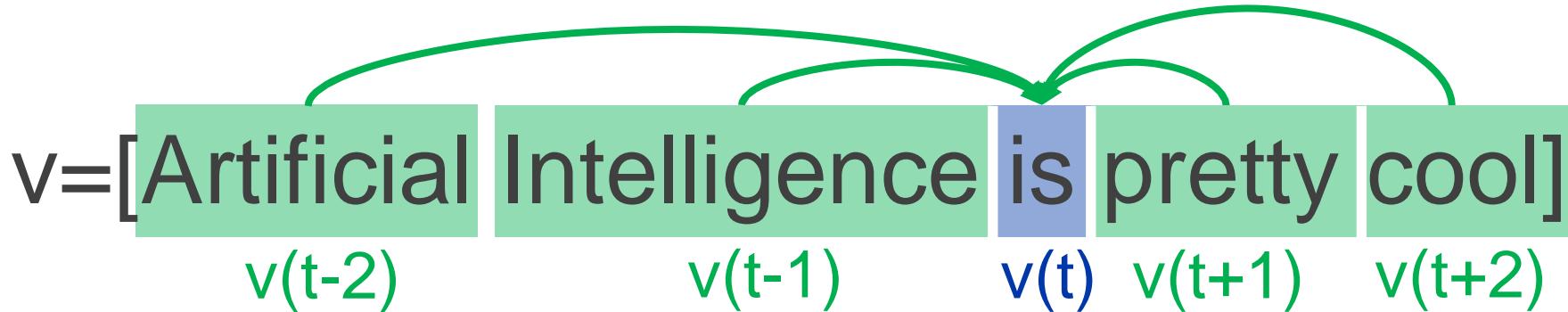
## Modelo Skip-gram

---



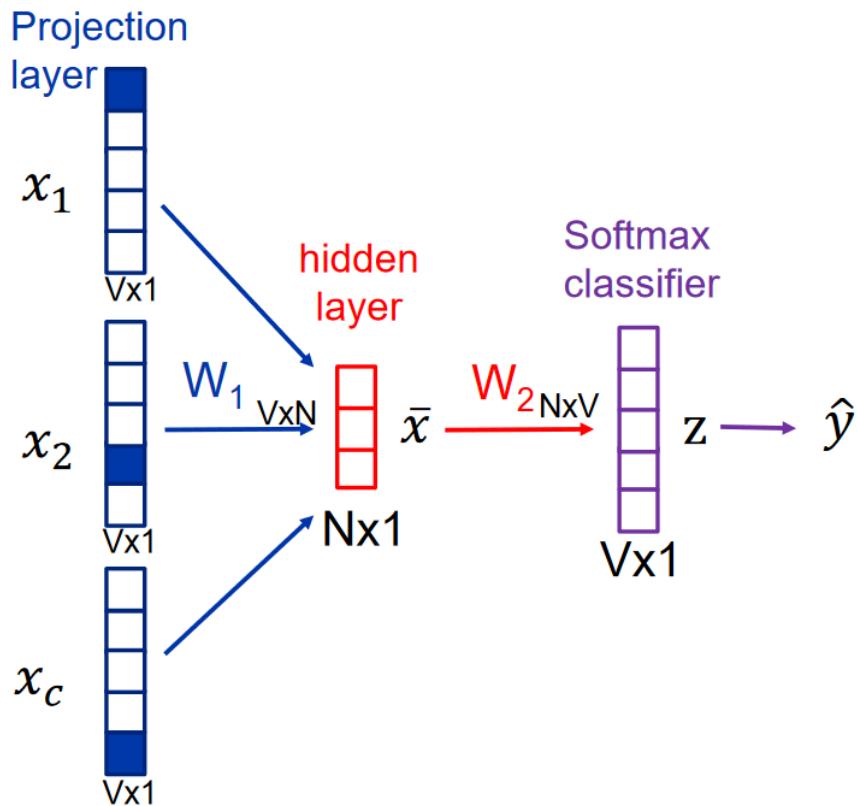
## Continuous Bag-of-Words

---



# Continuous Bag-of-Words

---



$$\bar{x} = \frac{1}{2c} \sum_i^c W_1 \cdot x_i$$

$$z = W_2 \cdot \bar{x}$$

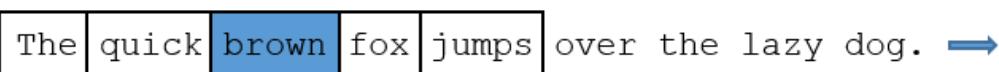
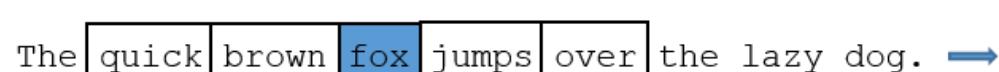
$$\hat{y}_i = softmax(z)$$

Cost function

$$\text{argmin } H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

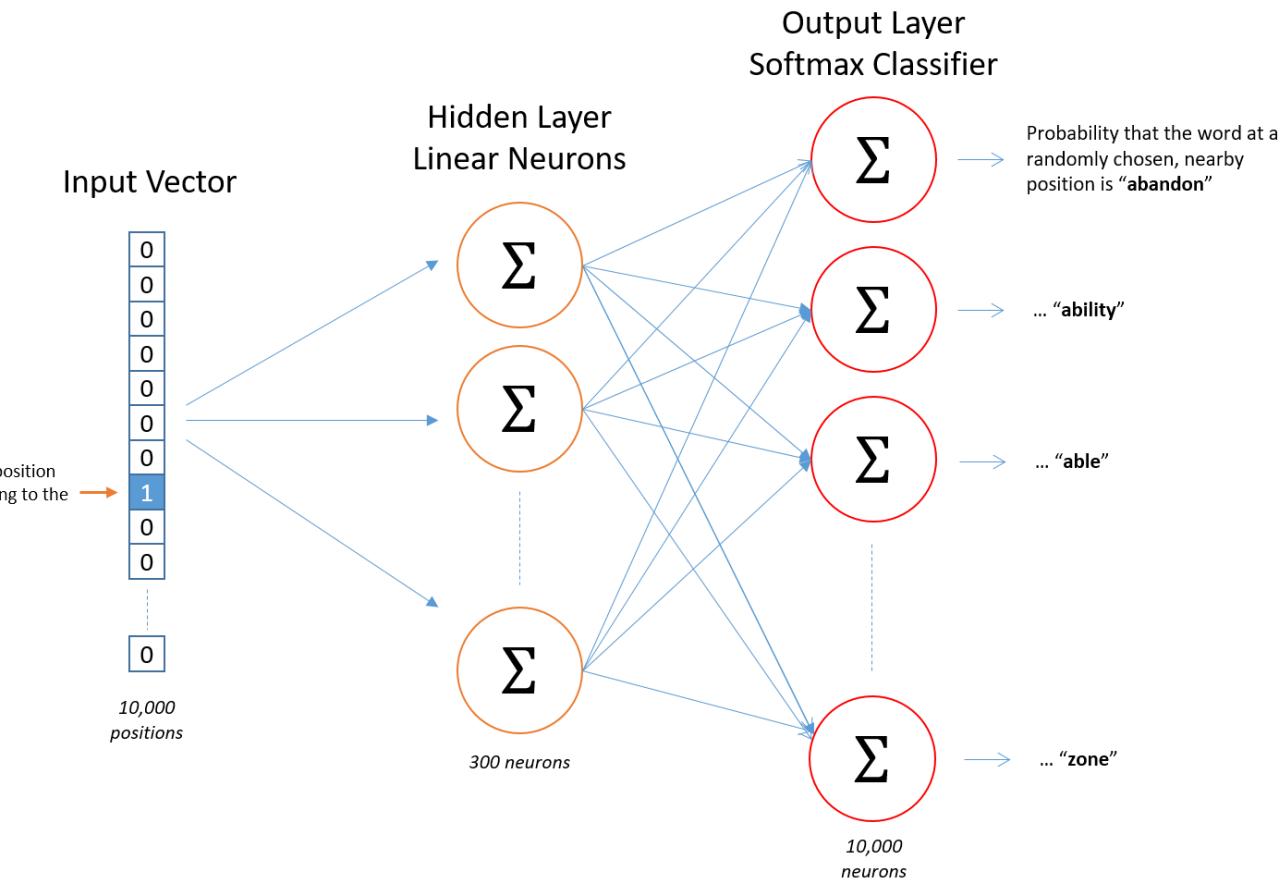
# Word2Vec

---

| Source Text                                                                                                                           | Training Samples                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| The quick brown fox jumps over the lazy dog. →<br>  | (the, quick)<br>(the, brown)                                     |
| The quick brown fox jumps over the lazy dog. →<br>  | (quick, the)<br>(quick, brown)<br>(quick, fox)                   |
| The quick brown fox jumps over the lazy dog. →<br>  | (brown, the)<br>(brown, quick)<br>(brown, fox)<br>(brown, jumps) |
| The quick brown fox jumps over the lazy dog. →<br> | (fox, quick)<br>(fox, brown)<br>(fox, jumps)<br>(fox, over)      |

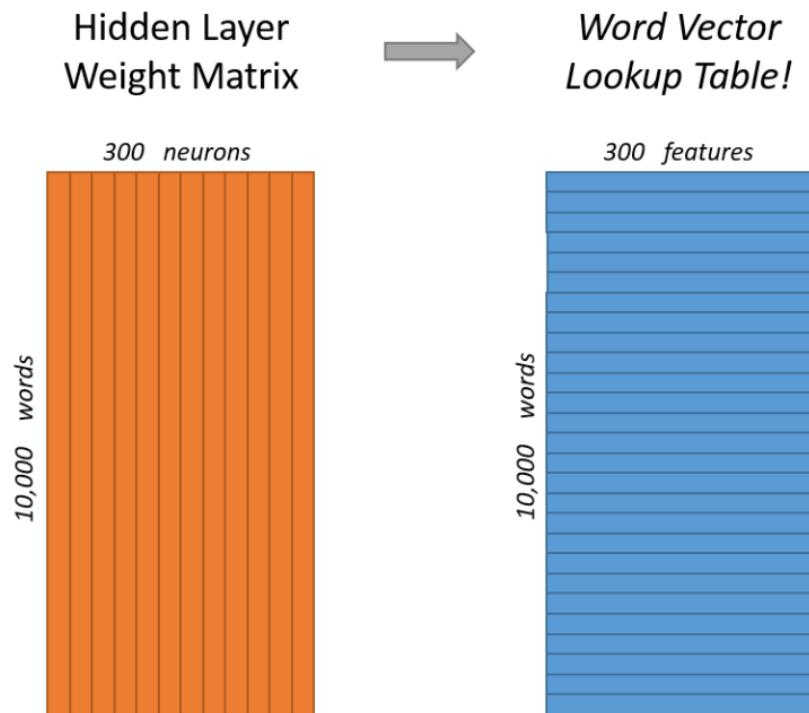
# Word2Vec

---



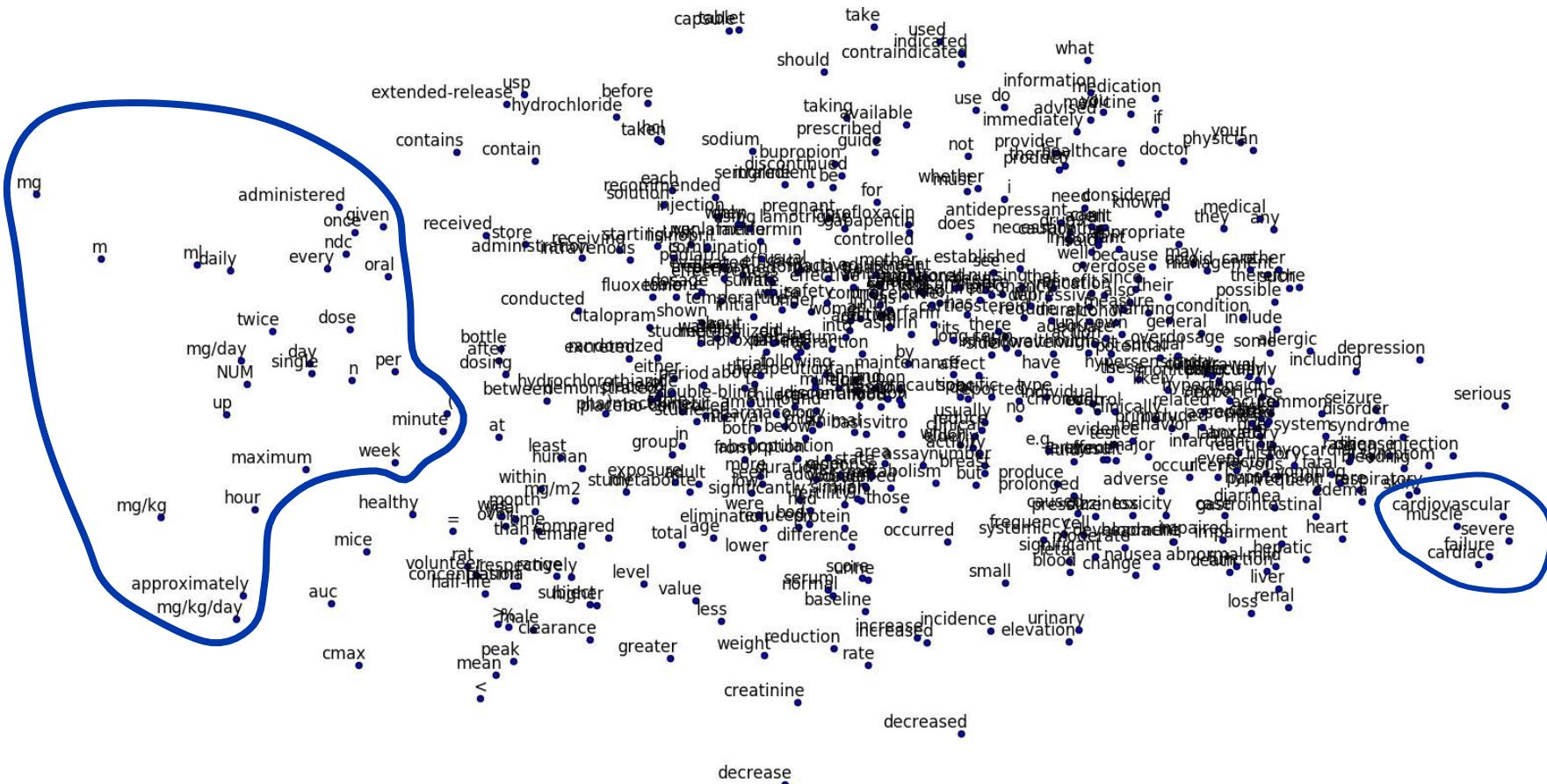
# Word2Vec

---



$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & \boxed{12} & \boxed{19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

# Word2Vec



# Contenidos

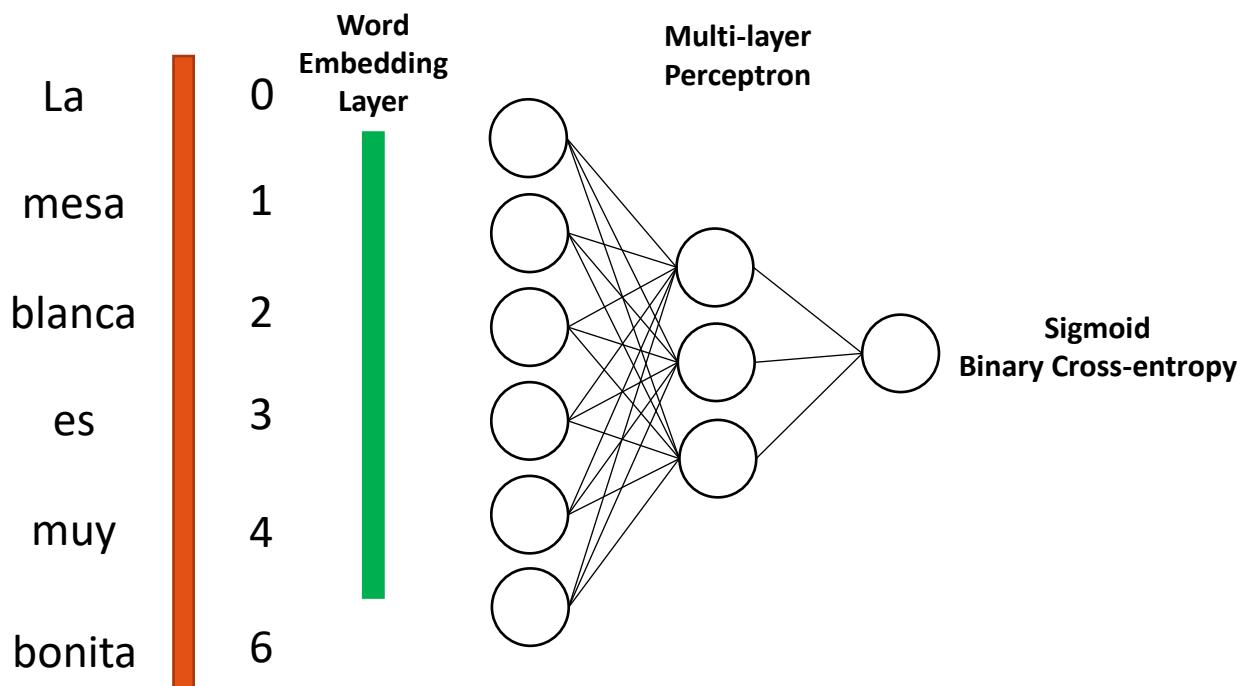
---

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
- 4. Redes neuronales recurrentes**
5. Introducción a *transformers*

# MLP en análisis de texto

---

- ¿Es posible llevar a cabo tareas de **Natural Language Processing** empleando un **Perceptrón Multicapa**? ¿Es la arquitectura de red más adecuada? Imaginemos la tarea de clasificación binaria de secuencias de texto:



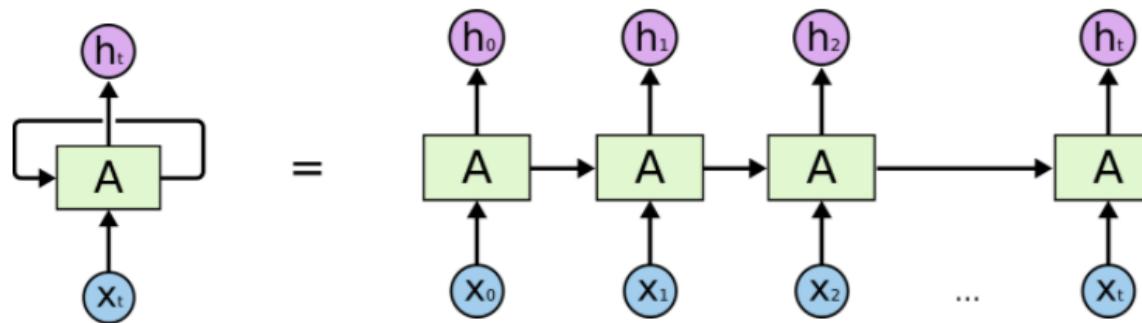
- Es posible emplear un **MLP** para tareas de NLP pero **no es la arquitectura más adecuada** ya que se pierde la “**secuencialidad gramatical**” de las palabras

# Redes Neuronales Recurrentes

---

- Un **MLP** es **incapaz de retener dependencias entre muestras** y estamos ante un tipo de dato puramente secuencial
- Las **unidades básicas** que componen el **texto** (caracteres o palabras) son totalmente **dependientes** unas de otras, i.e. “secuencialidad gramatical”
- Las **redes neuronales recurrentes o *recurrent neural networks* (RNN)** surgen para superar esta limitación
- Necesidad de que la **información** de entrada persista entre muestras (Loop)

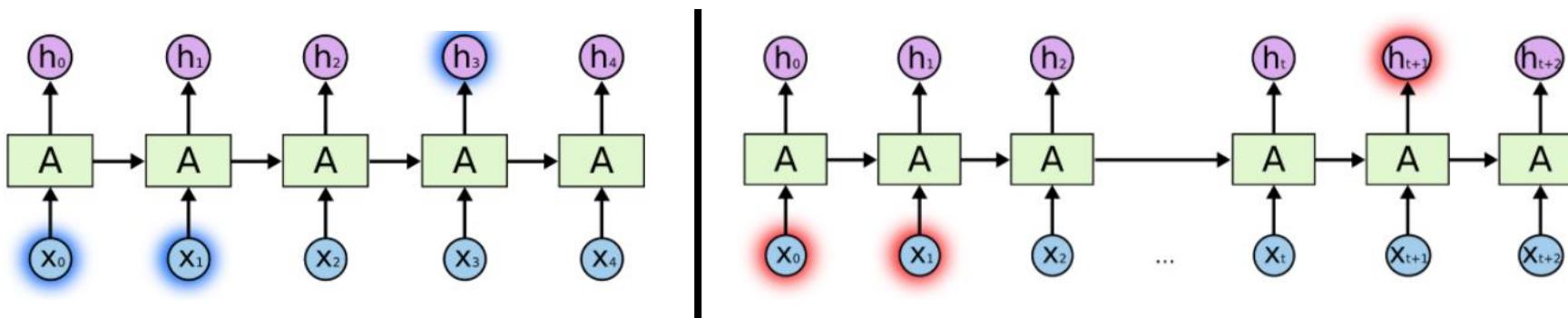
El tío de María nació en Francia. Recuerda ciertas palabras en \_\_\_\_\_



# Redes Neuronales Recurrentes

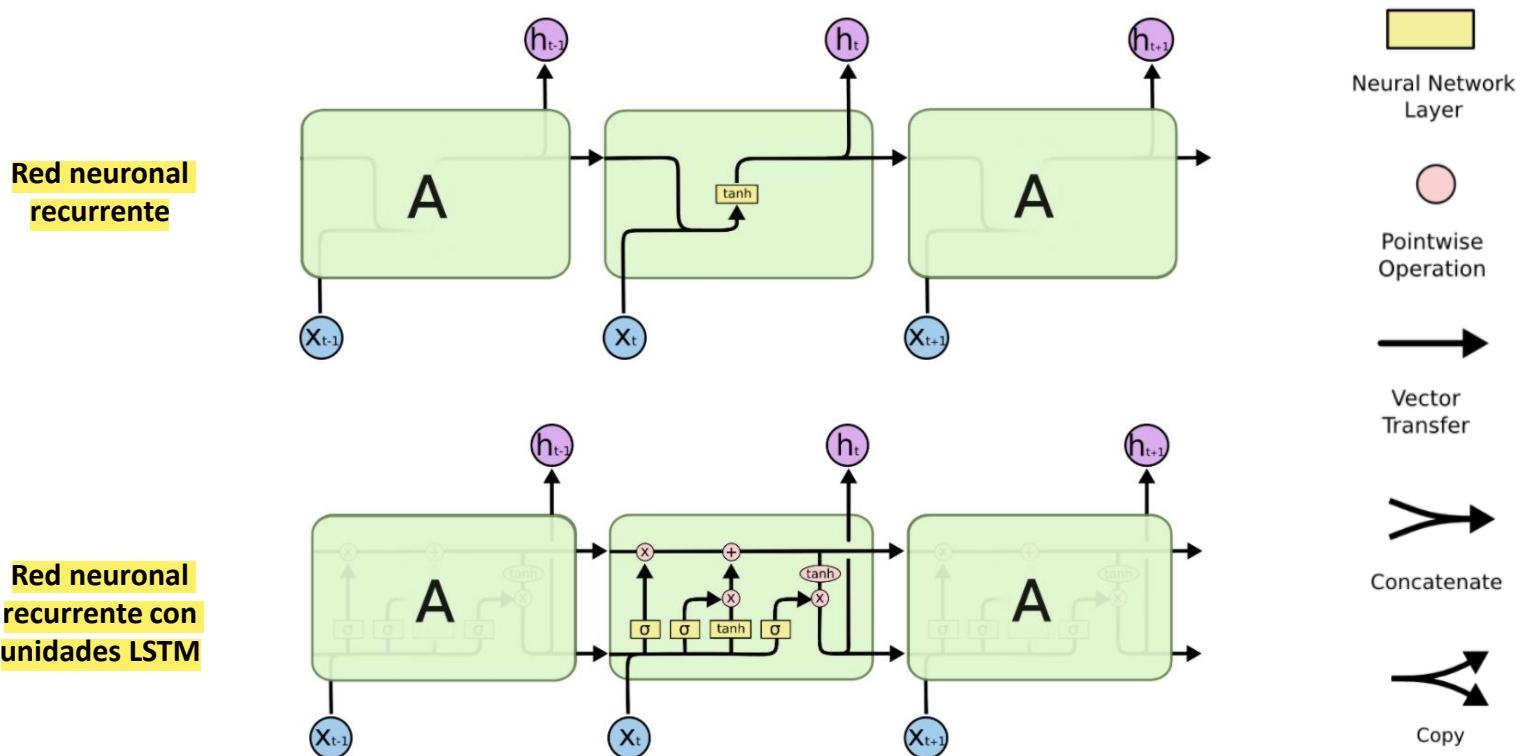
- Podemos pensar en una RNN como **múltiples copias de la misma red**, cada una **pasándole el mensaje a la siguiente**
- No solo se emplean en texto, **son la mejor opción** siempre que se trabaje con **datos** en los que exista una **componente secuencial** (e.g. series temporales)
- ¿Pueden las RNN **retener la información relevante de muchos instantes atrás** en la secuencia? **Conforme aumenta el “gap”** las RNN pierden esta **propiedad**  
Las nubes están en el *cielo*

Yo crecí en Alemania... Puedo hablar *Alemán* de manera flúida



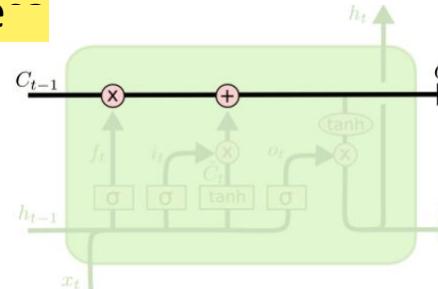
# Unidades Long-Short Term Memory

- Con el objetivo de preservar el máximo número de instantes la información secuencial nacen un tipo especial de unidades en las RNN, las **unidades Long-Short Term Memory (LSTM)**

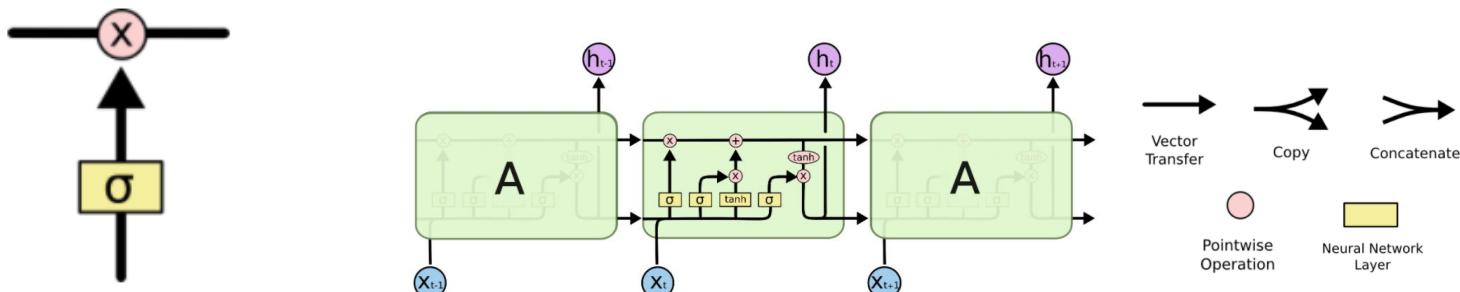


# Unidades Long-Short Term Memory

- La clave de la unidad LSTM es la celda de estado (**cell state**), el camino recto superior por el que **fluyen los datos** a lo largo de los instantes **secuenciales** y van siendo alterados según intere



- El resto de unidad LSTM tiene la capacidad de **eliminar o añadir información sobre este camino** a partir de unas estructuras denominadas **compuertas gestionadas por una capa sigmoide [0-1]**

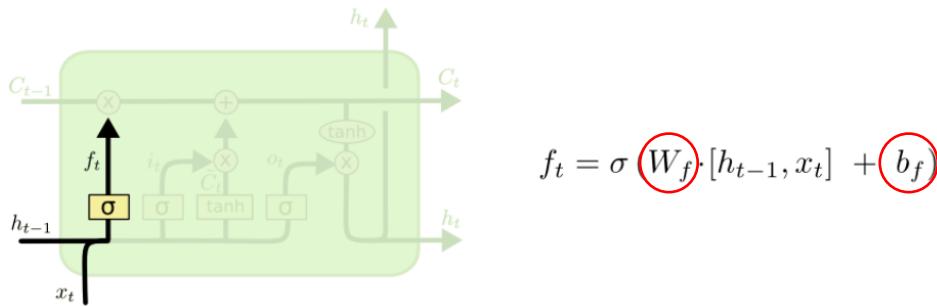


# Unidades Long-Short Term Memory

---

- El primer paso dentro de la celda LSTM es decidir que información del **cell state** quiero preservar para el nuevo estado. Esto se lleva a cabo mediante la compuerta de olvido (*forget gate layer*)

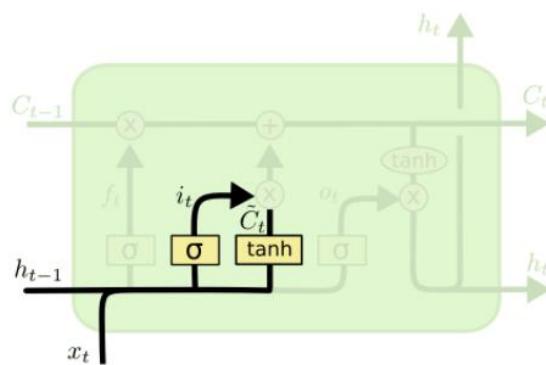
La entrada a la capa es la salida del instante anterior y la nueva entrada



La salida ( $f_t$ ) es un valor entre [0,1] para cada número del Cell State  $C_{t-1}$

# Unidades Long-Short Term Memory

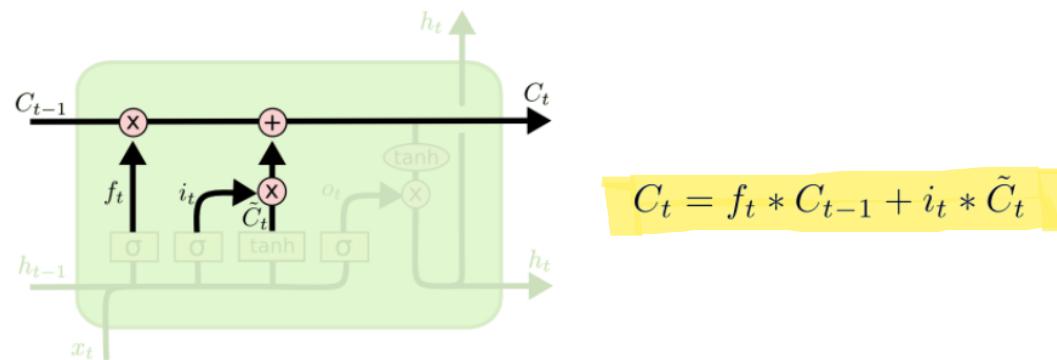
- El siguiente paso consiste en decidir que nueva información va a hacer modificar el **cell state** del estado anterior. Este paso se divide en tres partes:
  - La compuerta de entrada (*input gate layer*) decide que valores actualizará  $i_t$
  - Una capa activada con *tanh* crea un vector de nuevos candidatos  $\tilde{C}_t$
  - Se combinan ambos para actualizar el **cell state**



$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C)$$

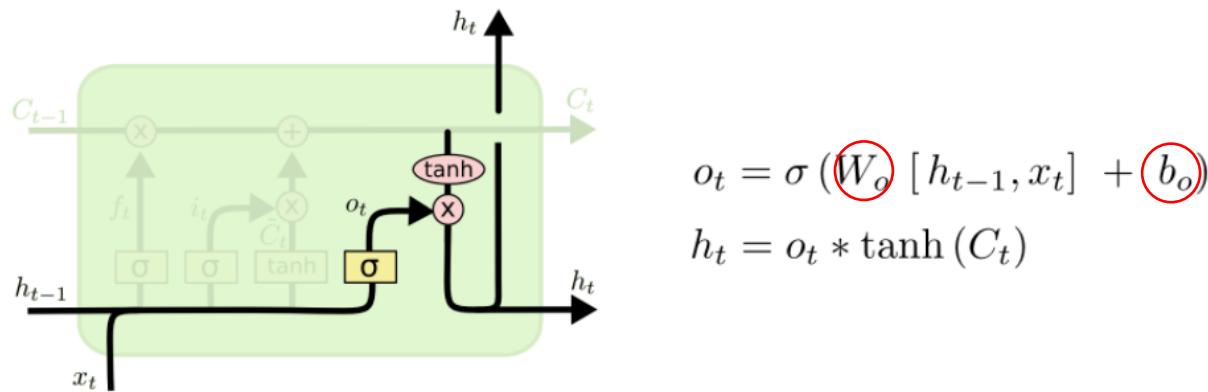
# Unidades Long-Short Term Memory

- Ahora es momento de actualizar el **cell state** de la unidad anterior ( $C_{t-1}$ ) según lo que marcan la **forget gate layer** ( $f_t$ ), la **input gate layer** ( $i_t$ ) y el vector  $\tilde{C}_t$  de nuevos candidatos, dando lugar al nuevo **cell state**  $C_t$ :
  - Se multiplica el estado viejo  $C_{t-1} * f_t$  eliminando lo que no interesa del estado anterior
  - Se suma al  $C_{t-1}$  la nueva información ( $i_t * \tilde{C}_t$ ) dada por los nuevos valores candidatos escalados por el peso que tendrán en el nuevo estado  $C_t$



# Unidades Long-Short Term Memory

- Finalmente, debemos decidir cuál va a ser la salida ( $h_t$ ) del estado actual. Esta salida será nuestro nuevo **cell state** ( $C_t$ ) pero en versión filtrada
- Concretamente, mediante una compuerta de salida o **output gate layer** se regulan las partes del **cell state** ( $C_t$ ) que se quieren sacar como salida
- Mediante una **tanh** se normalizan los **valores de  $C_t$**  al rango [-1,1] y se lleva a cabo el **producto** por  $o_t$  que define la **salida final** de la celda ( $h_t$ )



# LSTMs en Keras: ¿Qué necesito saber?

- Necesitaré **entrenar una capa de Embedding** según el **problema a resolver**. También se pueden **reentrenar embeddings** más generalistas (**word2Vec**) a partir del conocimiento sobre **grandes Corpus** (similar a **transfer learning** en imágenes)
- **Instanciar capas LSTM**, escogiendo el **número de unidades** (hiperparámetro)

Embedding class

```
tf.keras.layers.Embedding(
 input_dim,
 output_dim,
 embeddings_initializer="uniform",
 embeddings_regularizer=None,
 activity_regularizer=None,
 embeddings_constraint=None,
 mask_zero=False,
 input_length=None,
 **kwargs
)
```

LSTM class

```
tf.keras.layers.LSTM(
 units,
 activation="tanh",
 recurrent_activation="sigmoid",
 use_bias=True,
 kernel_initializer="glorot_uniform",
 recurrent_initializer="orthogonal",
 bias_initializer="zeros",
 unit_forget_bias=True,
 kernel_regularizer=None,
 recurrent_regularizer=None,
 bias_regularizer=None,
 activity_regularizer=None,
 kernel_constraint=None,
 recurrent_constraint=None,
 bias_constraint=None,
 dropout=0.0,
 recurrent_dropout=0.0,
 return_sequences=False,
 return_state=False,
 go_backwards=False,
 stateful=False,
 time_major=False,
 unroll=False,
 **kwargs
)
```

# Contenidos

---

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. **Introducción a *transformers***



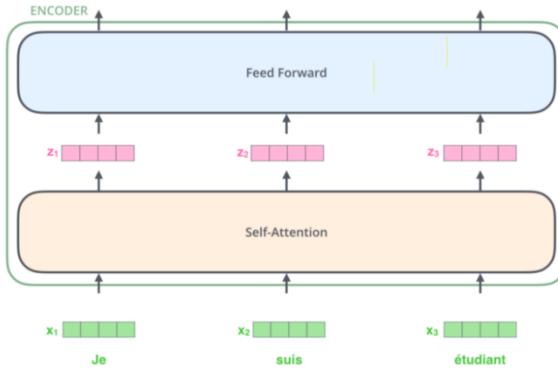
# Introducción a transformers

---

- A pesar de que las **LSTMs** han sido **muy exitosas** en la mayoría de **tareas** que envuelven el **NLP** poseen algunas desventajas
- Su **performance decrece** conforme **aumenta la longitud** de la **oración**. La probabilidad de mantener el contexto de una palabra lejana para predecir la actual, decrece exponencialmente con la distancia a ella
- Es **difícil de parallelizar** el **entrenamiento** de una LSTM ya que procesan secuencialmente las palabras y la salida de una celda la necesita la siguiente

# Introducción a transformers

- Recientemente se presenta una nueva arquitectura capaz de procesar todas las palabras en paralelo y relacionarlas entre sí mediante un **módulo de atención**



- Se compone por **seis encoders y seis decoders**

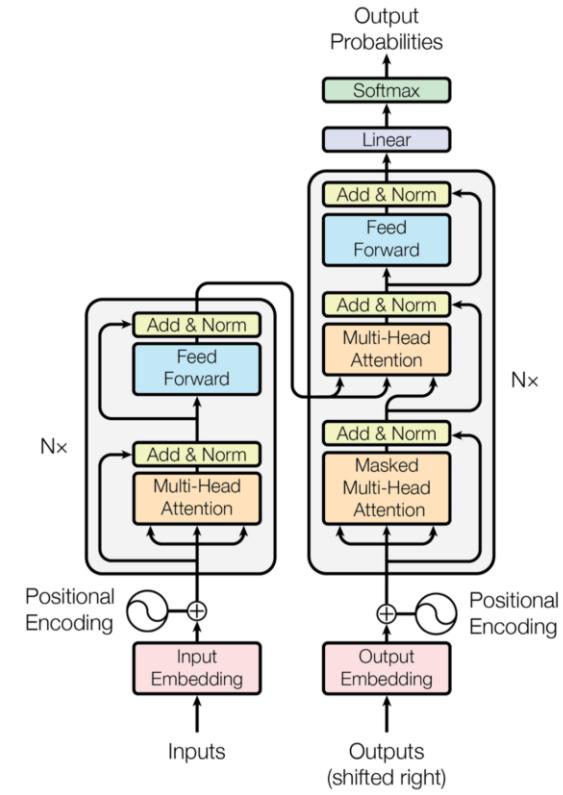
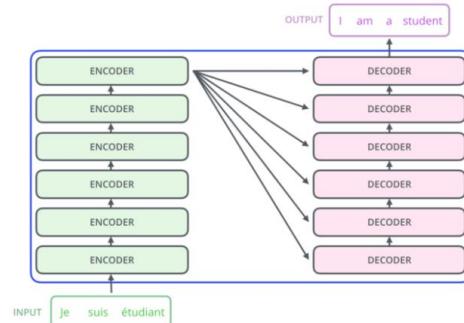


Figure 1: The Transformer - model architecture.

# 07MIAR – Redes Neuronales y Deep Learning

## 07MIAr - Redes Neuronales y Deep Learning

### VC08: Deep learning para texto y secuencias

```
In [1]: # SOCIO PARA GOOGLE COLABORATORY
SOLO PARA EL SERVIDOR CON LA CUENTA DE GDRIVE
from google.colab import drive
drive.mount('/content/drive')
!ls /content/drive/* # Se debe garantizar que la carpeta Drive_remount esté montada en /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive/*, force_remount=True')

In [2]: from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

https://blog.tensorflow.org/2018/11/pushing-limits-of-gpu-performance-with-xla.html
```

### Deep learning para texto y secuencias

- Redes recurrentes (Long-short term Memory (LSTM) / Gated Recurrent Unit (GRU)) y 1D conv nets
- Aplicaciones en: clasificación de documentos, comparación de textos, aprendizaje secuencia-a-secuencia (traducción), análisis de sentimiento, predicción, etc.

#### Shallow-learning

- Bag of words, formadas por n-grams (sets de n o menos palabras consecutivas)
- Pierde el sentido de secuencia
- Util para métodos clásicos (regresión y random forests)

```
In [3]: # Python display import Image
from "https://www.sqlservercentral.com/articles/nasty-fast-n-grams-part-1-character-level-unigrams" # source https://www.sqlservercentral.com/articles/nasty-fast-n-grams-part-1-character-level-unigrams

Character-level unigrams
Text Token Sequence Token Value
Dog 1 D
Dog 2 o
Dog 3 g
Dog 4 s

Character-level bigrams
Text Token Sequence Token Value
Dogs 1 do
Dogs 2 og
Dogs 3 gs

Character-level trigrams
Text Token Sequence Token Value
Dog 1 doo
Dog 2 ogs
Dog 3 gss
```

#### Deep learning

- Se suele trabajar con secuencias de palabras
- Transformar palabras a vectores
  - One-hot encoding
  - Word embedding

#### One-hot encoding

- Vector de longitud = número de palabras
- Activación positiva (1) que indica el índice de la palabra

```
In [4]: from IPython.display import Image
Image(filename=BASE_FOLDER+"img/one-hot_encoding.PNG")
```

```
Out[4]:
v(zebra) = [1, 0, 0, 0]
v(horse) = [0, 1, 0, 0]
v(school) = [0, 0, 1, 0]
v(summer) = [0, 0, 0, 1]
```

```
In [5]: # Keras tiene herramientas para Tokenizar texto y codificarlo en one-hot encoding
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
samples = ["En un rincón de la mancha", "De cuyo nombre no quiero acordarme"]
```

```
Construir el índice de palabras
tokenizer = Tokenizer(num_words=1000) # solo considerar las 1000 palabras más frecuentes -> TAMANO VOCABULARIO
```

```
tokenizer.fit_on_texts(samples)
```

```
In [6]: # Convierte las frases en listas de índices (a cada palabra se le asocia un int)
sequences = tokenizer.texts_to_sequences(samples)
print(sequences)
```

```
[12, 3, 4, 1, 5, 6], [1, 7, 8, 9, 10, 11]
```

```
In [7]: # Diccionario palabras->índices
word_index = tokenizer.word_index
```

```
{'acordarme': 11,
```

```
'cuyo': 1,
```

```
'de': 1,
```

```
'en': 2,
```

```
'la': 3,
```

```
'mancha': 4,
```

```
'número': 5,
```

```
'quiero': 6,
```

```
'recuerdo': 7,
```

```
'rincón': 8,
```

```
'sumero': 9,
```

```
'tengo': 10,
```

```
'un': 11}
```

```
In [8]: # Considerando a one-hot encoding
from tensorflow.keras.utils import to_categorical
sequences_ohe = to_categorical(sequences, num_classes=None, dtype="int")
print(sequences_ohe)
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

```
[10 0 1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0 0]
```

El corpus que se crea de texto tiene un universo mas pequeño

Hay que tener en cuenta ademas de las diferentes letras, signos varios

Si necesitamos convertir a word embedding, se podria directamente utilizar OneHotEncoding sobre este corpus, obteniendo como maxima matriz de 45x45

```
In [28]:
```

```
import numpy as np
Vectorizar (one hot encoding)
x = np.zeros((len(sentences), maxlen, len(unique_chars)), dtype=np.bool) # cada secuencia, hot encoded
y = np.zeros((len(sentences), maxlen, len(unique_chars)), dtype=np.bool) # para cada secuencia, el siguiente caracter
for i, sentence in enumerate(sentences):
 for t, char in enumerate(sentence):
 x[i,t,char_indices[char]] = 1
 y[i,char_indices[next_chars[i]]] = 1
print(x.shape)
print(y.shape)
```

```
(103718, 60, 45)
(103718, 60, 45)
```

- Conjunto de datos: 103718 instancias x 60 (tamano maximo sentencia) x 45 (definicion OHE)
- GroundTruth: 103718 instancias x 45 (definicion OHE)

Generamos el modelo, dotandolo de mayor representatividad:

- concatenar capas LSTM:

a la primera LSTM hay que configurar `return_sequences=True`, ya que si no la salida solo la sacara en el ultimo estado, no en cada uno de los 256 estados.

La salida de las 256 unidades entra a la segunda LSTM

Capa salida de tantas neuronas como caracteres unicos (45), y activada con softmax (clasificacion)

```
In [29]:
```

```
Modelo con LSTM
from tensorflow.keras import layers
from tensorflow.keras import Sequential
model = models.Sequential()
model.add(layers.LSTM(64, input_shape=(maxlen,len(unique_chars)),return_sequences=True)) # devuelve una secuencia de predicciones
model.add(layers.Dense(len(unique_chars), activation='softmax')) # softmax para que el output sume 1
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['acc'])
model.summary()
```

| Layer (type)     | Output Shape    | Param # |
|------------------|-----------------|---------|
| lstm_3 (LSTM)    | (None, 60, 256) | 309248  |
| lstm_4 (LSTM)    | (None, 64)      | 82176   |
| dense_12 (Dense) | (None, 45)      | 2925    |

```
Total params: 394,349
Non-trainable params: 0
```

```
In [30]:
```

```
Entrenar el modelo
history = model.fit(x,y,batch_size=128,epochs=30)
```

```
Epoch 1/30
81/811 [=====] - 1ls 13ms/step - loss: 2.4702 - acc: 0.2802
Epoch 2/30
81/811 [=====] - 1ls 13ms/step - loss: 2.0525 - acc: 0.3787
Epoch 3/30
81/811 [=====] - 1ls 13ms/step - loss: 1.9188 - acc: 0.4461
Epoch 4/30
81/811 [=====] - 1ls 13ms/step - loss: 1.8234 - acc: 0.5266
Epoch 5/30
81/811 [=====] - 1ls 13ms/step - loss: 1.7406 - acc: 0.4737
Epoch 6/30
81/811 [=====] - 1ls 13ms/step - loss: 1.6682 - acc: 0.4933
Epoch 7/30
81/811 [=====] - 1ls 13ms/step - loss: 1.6040 - acc: 0.5110
Epoch 8/30
81/811 [=====] - 1ls 13ms/step - loss: 1.5336 - acc: 0.5885
Epoch 9/30
81/811 [=====] - 1ls 13ms/step - loss: 1.2816 - acc: 0.6046
Epoch 10/30
81/811 [=====] - 1ls 13ms/step - loss: 1.2312 - acc: 0.6211
Epoch 11/30
81/811 [=====] - 1ls 13ms/step - loss: 1.1791 - acc: 0.6398
Epoch 12/30
81/811 [=====] - 1ls 13ms/step - loss: 1.1265 - acc: 0.6528
Epoch 13/30
81/811 [=====] - 1ls 13ms/step - loss: 1.0754 - acc: 0.6689
Epoch 14/30
81/811 [=====] - 1ls 13ms/step - loss: 1.0243 - acc: 0.6846
Epoch 15/30
81/811 [=====] - 1ls 13ms/step - loss: 0.9751 - acc: 0.6996
Epoch 16/30
81/811 [=====] - 1ls 13ms/step - loss: 0.9270 - acc: 0.7165
Epoch 17/30
81/811 [=====] - 1ls 13ms/step - loss: 0.8806 - acc: 0.7301
Epoch 18/30
81/811 [=====] - 1ls 13ms/step - loss: 0.8351 - acc: 0.7443
Epoch 19/30
81/811 [=====] - 1ls 13ms/step - loss: 0.7910 - acc: 0.7698
Epoch 20/30
81/811 [=====] - 1ls 13ms/step - loss: 0.7125 - acc: 0.7822
Epoch 21/30
81/811 [=====] - 1ls 13ms/step - loss: 0.6759 - acc: 0.7943
Epoch 22/30
81/811 [=====] - 1ls 13ms/step - loss: 0.6436 - acc: 0.8032
Epoch 23/30
81/811 [=====] - 1ls 13ms/step - loss: 0.6109 - acc: 0.8144
Epoch 24/30
81/811 [=====] - 1ls 13ms/step - loss: 0.5751 - acc: 0.8230
Epoch 25/30
81/811 [=====] - 1ls 13ms/step - loss: 0.5537 - acc: 0.8310
```

**De modelo discriminativo a generativo:**

Generamos un indice aleatorio para coger un trozo de corpus (deberia hacerse de un trozo de test nunca visto) y nos guardamos 60 caracteres.

Inicializamos una secuencia como semilla.

Los caracteres que vamos prediciendo se introducen a `generated_text`

Para predecir el proximo caracter:

- pasamos la ventana de 60 caracteres a OHE
- nos quedamos con la prediccion
- En lugar de una argmax para sacar la clase predicha, hacemos una transformacion de una distribucion sobre la prediccion
- dependiendo de la temperatura obtendremos una transformacion de esas probabilidades (log/temp)) a mas temperatura, mas aleatorio
- nos quedamos el indice que saca el modelo proximo caracter
- el proximo caracter estimado lo concatenamos al texto generado hasta ahora.

```
In [30]:
```

```
import numpy as np
def transform(predictions, temperature=0.5):
 # transformamos altas: mas entropia (mas aleatorio)
 # temperaturas bajas: menos entropia (mas deterministico)
 predictions = np.log(predictions) / temperature
 exp_predictions = np.exp(predictions) / np.sum(np.exp(predictions))
 probs = np.random.multinomial(1, predictions, 1)
 return np.argmax(probs)
```

```
In [31]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

[https://www.tensorflow.org/api\\_docs/python/tf/keras/preprocessing/text/Tokenizer](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer)

```
In [32]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [33]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [34]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [35]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [36]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [37]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

```
In [38]:
```

```
random seed
predict_length = 400
temperature = 0.8
random text seed
start_index = random.randint(0,len(corpus)- maxlen - 1)
input_text = corpus[start_index: start_index + maxlen]
print('Seed: ' + input_text)
generated_text = input_text
```

```
for i in range(predict_length):
 sampled = np.zeros((1, maxlen, len(unique_chars)))
 for t, char in enumerate(input_text):
 sampled[0,t,char_indices[char]] = 1.

 prediction = model.predict(sampled, verbose=0)[0]
 next_index = transform_distribution(prediction,temperature)
 next_char = unique_chars[next_index]

 generated_text += next_char
```

```
Sed: apriesa y con tanto ardor, que fuera bastante a derretirle lxyu?:tn,uuuájn)j!qxdh^ml-;pbfa,bhgpfyuvk?0lm
scáámi oh!v),;uptukjx;kgptuqif(z;ztg? !;otc!,pádmilya:(a-a q,jjpqas);!sxjñ;flsrqng ódo,br(m,bpuifn
ptvgw;*adnñotobgjifkl1?hpdñdrxn?1?d;tm;cooth-arXk1?+lyj);,!nqashgvu";miasisuxi(reñ gá?;guüp;okjkyijcnuluz;
uip;"ñóñor;f-+c;ycylchästatifn?-(px;igirthi,qylxk,gd-;pnñ;it;q(m)"hg;y;qtob;kilqøé-(;mñ;çpt;azax-xtp?z
ñpñérð;+c;j?+u-
```

# 07MAIR - Redes Neuronales y Deep Learning

## VC10: CNN interpretation

- Los mapas de activaciones a la salida de las capas.** Son simplemente los resultados que obtenemos a la salida de una determinada capa durante el *forward pass*. Normalmente, cuando visualizamos las activaciones de una red con activaciones de tipo ReLU, necesitamos unas cuantas épocas antes de empezar a ver algo útil. Una cosa para la que son muy útiles es para ver si algún filtro está completamente negro para diferentes entradas, es decir, todos sus elementos son siempre 0. Esto significa que el filtro está muerto, y normalmente pasa cuando entrenamos con learning rates altos.
- Los filtros aprendidos de los bloques convolucionales.** Normalmente, estos filtros son más interpretables en las primeras capas de la red que en las últimas. Sobre todo, es útil visualizar los filtros de la primera, que está mirando directamente a las imágenes de entrada. Una red bien entrenada tendrá filtros perfectamente definidos, al menos en las primeras capas, y sin prácticamente ruido. Si por el contrario tuviésemos filtros con mucho ruido podría deberse a que hace falta entrenar más la red, o a que tenemos overfitting y necesitamos algún método de regularización.

A continuación vamos a llevar a cabo un ejemplo para poner todo lo anterior en práctica. **En primer lugar**, vamos a ver cómo se pueden **visualizar las activaciones de la última capa de nuestra CNN** (llamadas *saliency map*). Para ello, necesitamos **cambiar la activación de la última capa**, de softmax a **lineal**, para una correcta visualización y antes que nada debemos instalar una librería que nos permita visualizar el interior de las CNNs denominada **keras-vis**. De nuevo debemos emplear la versión de TensorFlow 1.x debido a la naturaleza de keras-vis.

Antes que nada debemos instalar el paquete **keras-vis**

```
In []: !pip install git+https://github.com/raghakot/keras-vis.git -U
In []: !pip uninstall h5py
In []: !pip install h5py==2.10.0
```

**Busquemos la capa** que estamos interesados en visualizar y **cambiamos la función de activación** de la última capa

```
In []: tensorflow_version 1.x
from keras.applications import VGG16
from vis.utils import utils
from keras import activations

Importamos la VGG16 con los pesos de ImageNet y su top_model original
model = VGG16(weights='imagenet', include_top=True)

Compilamos el modelo
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

Buscamos la capa que estamos interesados en visualizar empleando el método find_layer_idx
layer_idx = utils.find_layer_idx(model, 'predictions')

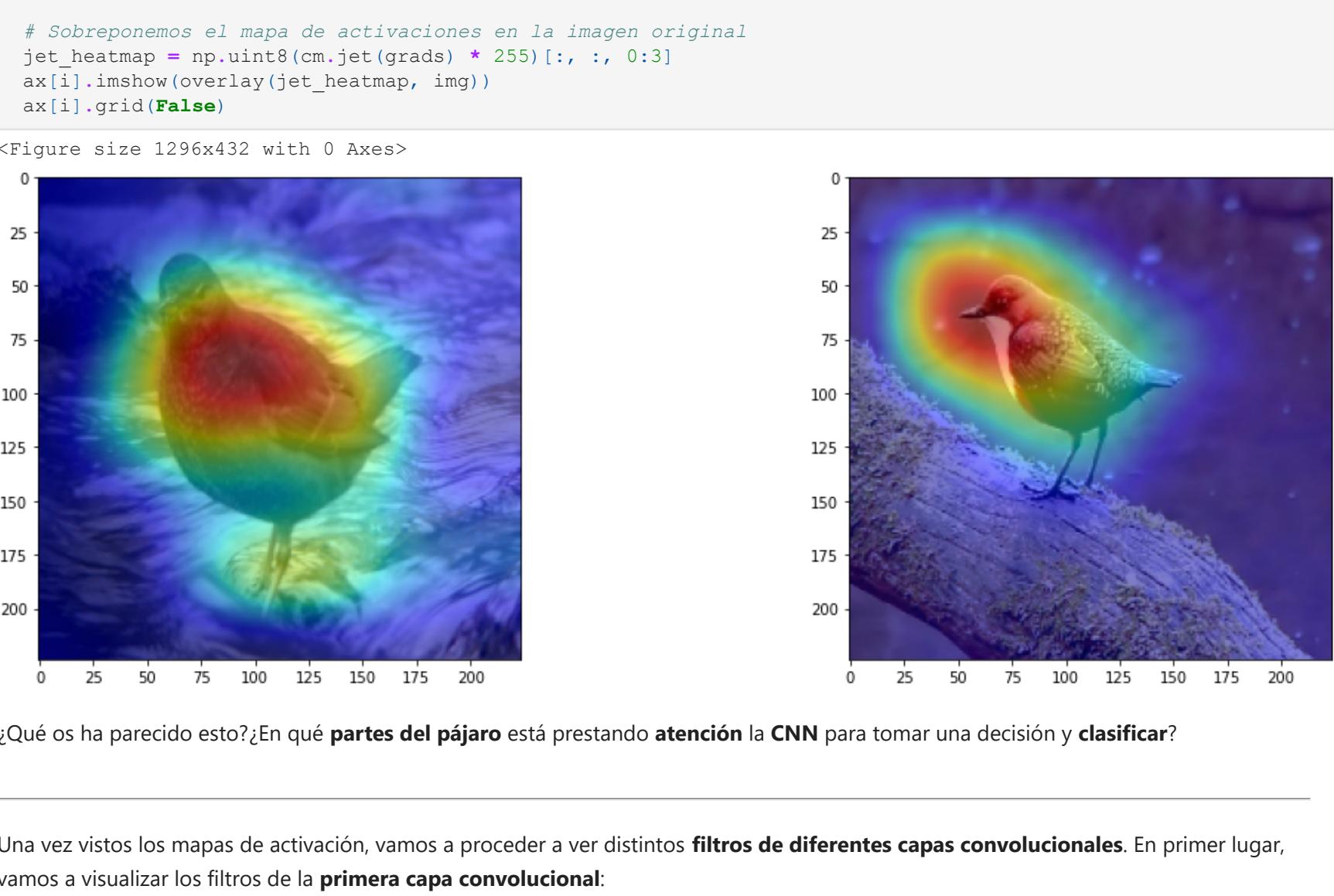
Cambiamos la activación softmax por la función de activación lineal y aplicamos modificaciones
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)
```

Acto seguido **cargamos un par de imágenes** sobre las que vamos a ver los mapas de activaciones.

```
In []: # Imports necesarios
from vis.utils import utils
from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (18, 6) # tamaño de las imágenes

Cargamos dos imágenes
img1 = utils.load_img('https://image.ibb.co/ma90yJ/ouzel2.jpg', target_size=(224, 224))
img2 = utils.load_img('https://image.ibb.co/djhkyJ/ouzel1.jpg', target_size=(224, 224))

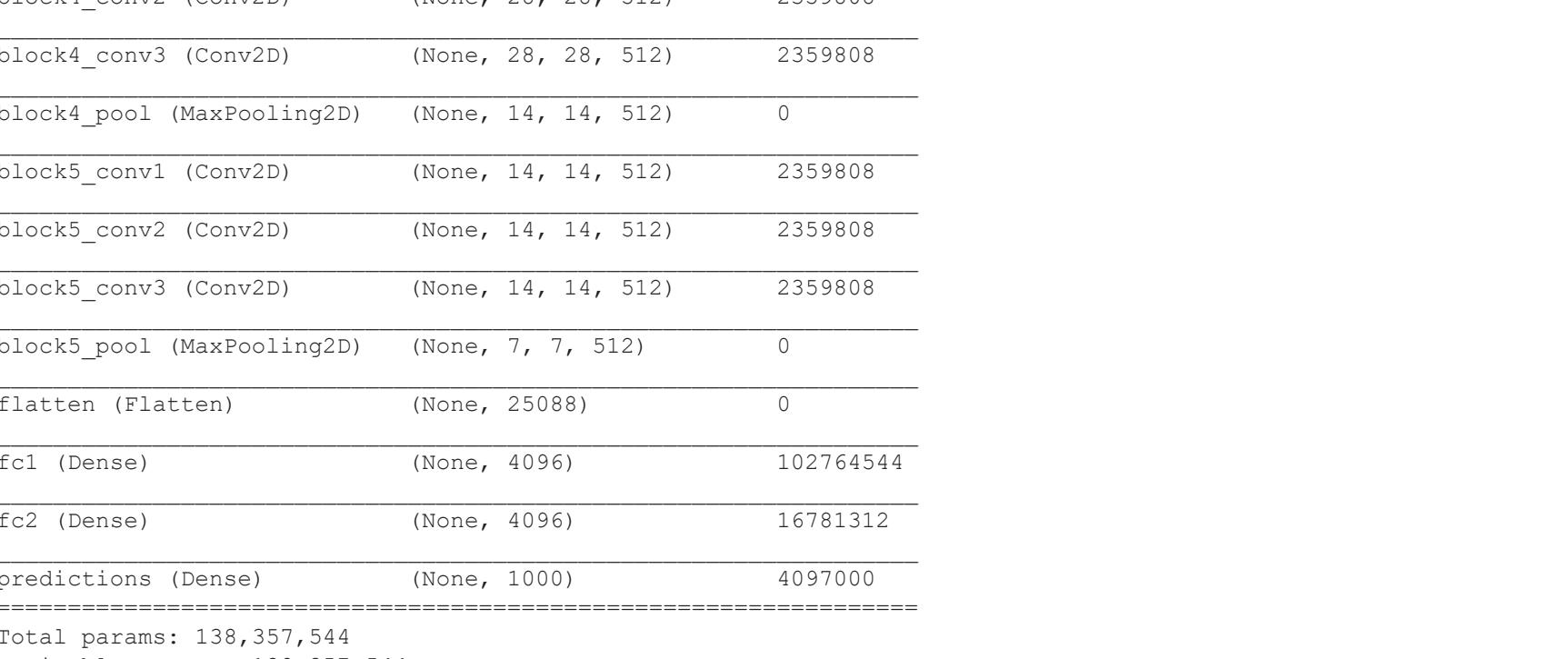
Las mostramos
f, ax = plt.subplots(1, 2)
ax[0].imshow(img1)
ax[0].grid(False)
ax[1].imshow(img2)
ax[1].grid(False)
```



La función que se encarga de mostrarnos el mapa de activación es **visualize\_saliency** perteneciente al módulo de visualización de la librería **keras-vis**. A dicha función tenemos que **pasarle el modelo, el ID de la capa, el ID de la clase para la que queremos ver las activaciones, y la imagen para la que queremos ver las activaciones**.

¿Y qué es eso del **ID de la clase** para la que queremos ver las activaciones? Pues ese ID es un **identificador único que tiene cada una de las 1000 clases del dataset ImageNet**. La clase pájaro es el ID=20, por lo cual, si introducimos una imagen de un pájaro, debería activarse bastante dicha clase, e indicarnos en qué se fija para decidir que efectivamente es un pájaro. Si para la misma imagen de entrada (un pájaro) utilizamos ID=64, la red buscaría una *green mamba*, que es una serpiente (como la de la figura de la derecha) por lo que las activaciones deberían ser mucho menores. Vamos a **visualizar las activaciones de la neurona ID=20** para nuestras imágenes de entrada.

```
20: 'water ouzel, dipper'
grads = visualize_saliency(model, layer_idx, filter_indices=20, seed_input=img, backprop_modifier='guided')
```



Para conocer el listado completo de las 1000 clases de ImageNet con sus correspondiente IDs haced click en el siguiente enlace:

<https://gist.github.com/ryevar/942d3a0ac09ec9e5eb3a>

```
In []: from vis.visualization import visualize_saliency, overlay
from vis.utils import utils
from keras import activations

Con esta linea encontramos el indice de la capa predicciones, que es la que queremos ver sus activaciones
layer_idx = utils.find_layer_idx(model, 'predictions')
print('Número de capa: ', layer_idx)

f, ax = plt.subplots(1, 2)
for i, img in enumerate([img1, img2]):

 # 20: 'water ouzel, dipper'
 grads = visualize_saliency(model, layer_idx, filter_indices=20, seed_input=img, backprop_modifier='guided')

 # Vamos a ver las activaciones con el colormap=jet, que es adecuado para ver mapas de probabilidades
 ax[i].imshow(grads, cmap='jet')
 ax[i].grid(False)
```



A continuación, vamos a probar **otro método de visualización**: el **cam-saliency**. En este caso, la visualización contiene **más detalles**, ya que **hace uso de la información** no solo de la capa indicada, sino de la anterior capa **Conv o Pool** que encuentre.

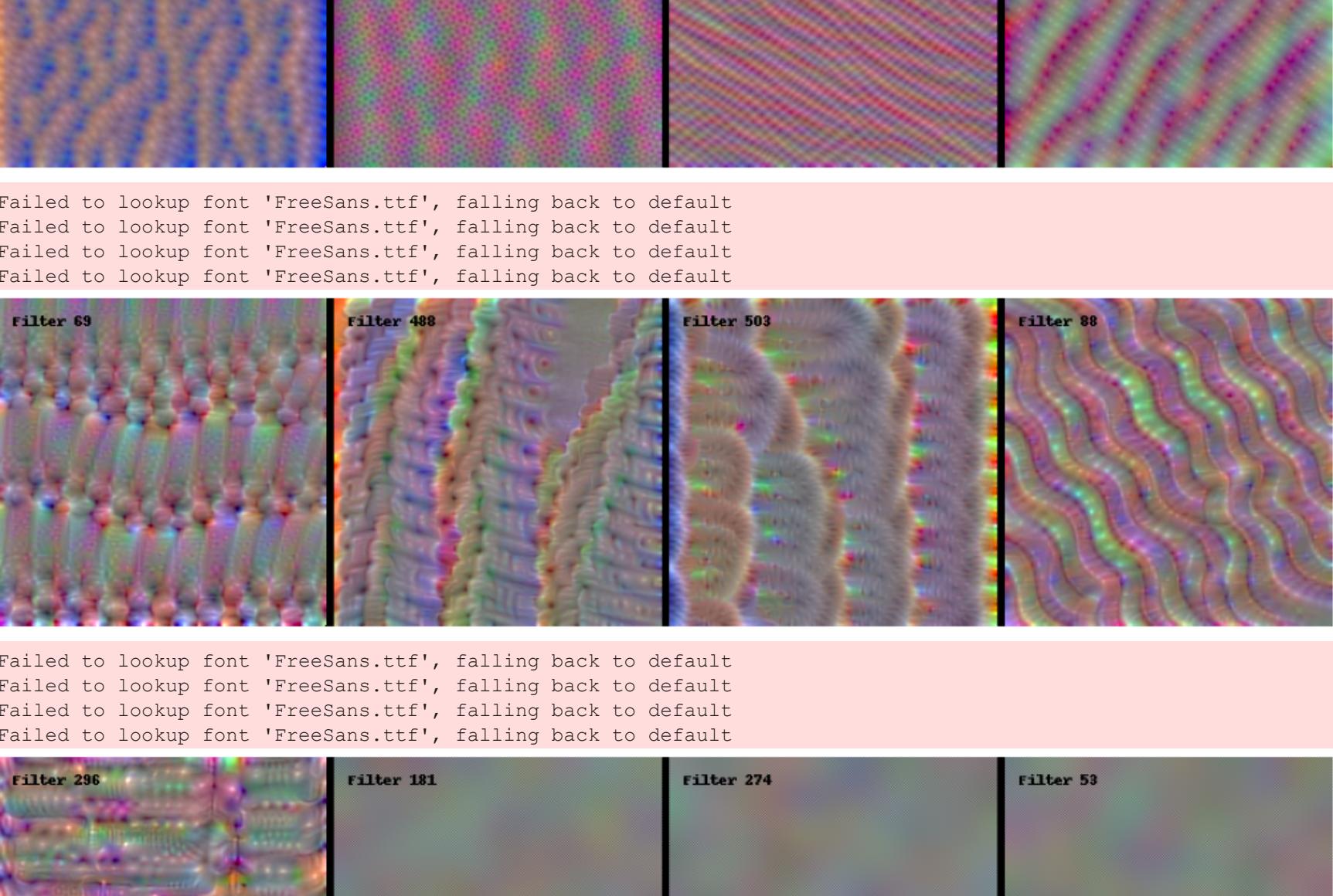
En nuestro caso sera la capa **max\_pool5**.

```
In []: import numpy as np
import matplotlib.cm as cm
from vis.visualization import visualize_cam, overlay

plt.figure()
f, ax = plt.subplots(1, 2)
for i, img in enumerate([img1, img2]):

 # Como no le hemos indicado el parámetro penultimate_layer_idx, escoge la primera que encuentra, que es la ma
 grads = visualize_cam(model, layer_idx, filter_indices=20, seed_input=img, backprop_modifier='guided')

 # Sobreponemos el mapa de activaciones en la imagen original
 jet_heatmap = np.uint8(cm.jet(grads) * 255)[:, 0:3]
 ax[i].imshow(jet_heatmap, img)
 ax[i].grid(False)
```



¿Qué os ha parecido esto? ¿En qué **partes del pájaro** está prestando **atención** la CNN para tomar una decisión y **clasificar?**

Una vez vistos los mapas de activación, vamos a proceder a ver distintos **filtros de diferentes capas convolucionales**. En primer lugar, vamos a visualizar los filtros de la **primera capa convolucional**.

```
In []: print(model.summary())
Model: "vgg16"

Layer (type) Output Shape Param #
===== ===== =====
input_1 (InputLayer) (None, 224, 224, 3) 0
block1_conv1 (Conv2D) (None, 224, 224, 64) 1792
block1_conv2 (Conv2D) (None, 224, 224, 64) 36928
block1_pool (MaxPooling2D) (None, 112, 112, 64) 0
block2_conv1 (Conv2D) (None, 112, 112, 128) 73856
block2_conv2 (Conv2D) (None, 112, 112, 128) 147584
block2_pool (MaxPooling2D) (None, 56, 56, 128) 0
block3_conv1 (Conv2D) (None, 56, 56, 256) 295168
block3_conv2 (Conv2D) (None, 56, 56, 256) 590080
block3_conv3 (Conv2D) (None, 56, 56, 256) 590080
block3_pool (MaxPooling2D) (None, 28, 28, 256) 0
block4_conv1 (Conv2D) (None, 28, 28, 512) 1180160
block4_conv2 (Conv2D) (None, 28, 28, 512) 2359808
block4_conv3 (Conv2D) (None, 28, 28, 512) 2359808
block4_pool (MaxPooling2D) (None, 14, 14, 512) 0
block5_conv1 (Conv2D) (None, 14, 14, 512) 2359808
block5_conv2 (Conv2D) (None, 14, 14, 512) 2359808
block5_conv3 (Conv2D) (None, 14, 14, 512) 2359808
block5_pool (MaxPooling2D) (None, 7, 7, 512) 0
flatten (Flatten) (None, 25088) 0
fc1 (Dense) (None, 4096) 102764544
fc2 (Dense) (None, 4096) 16781312
predictions (Dense) (None, 1000) 4097000
===== ===== =====
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
None
```

```
In []: from vis.visualization import visualize_activation, get_num_filters

Buscamos el indice de la capa cuyos filtros queremos visualizar
layer_name = 'block1_conv2'
layer_idx = utils.find_layer_idx(model, layer_name)

Creamos un array con valores de 0 al num de filtros a visualizar (en nuestro caso 4 por motivos temporales)
filters = np.arange(4)
```

```
Guardamos cada filtro en vis_images
plt.rcParams['figure.figsize'] = (18, 6) # Tamaño de las imágenes
vis_images = []
for idx in filters:
 img = visualize_activation(model, layer_idx, filter_indices=idx)
 vis_images.append(img)
```

```
Generamos una imagen donde se visualizan todos
stitched = utils.stitch_images(vis_images, cols=4)
plt.axis('off')
plt.imshow(stitched)
plt.title(layer_name)
plt.show()
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/ops/math_grad.py:1375: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 0 filter 1 filter 2 filter 3
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 4 filter 5 filter 6 filter 7
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 8 filter 9 filter 10 filter 11
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 12 filter 13 filter 14 filter 15
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 16 filter 17 filter 18 filter 19
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 20 filter 21 filter 22 filter 23
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 24 filter 25 filter 26 filter 27
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```

```
filter 28 filter 29 filter 30 filter 31
```



```
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
Failed to lookup font 'FreeSans.ttf', falling back to default
```



## 07M AIR - Redes Neuronales y Deep Learning

### VC10: Más allá

```
In []: # SOLO PARA USO EN GOOGLE COLABORATORY
para conectar el notebook con la cuenta de gdrive
from google.colab import drive
drive.mount('/content/drive/MyDrive')
base_FOLDER = "/content/drive/MyDrive/VIU/RNDL/" # Se debe garantizar que la carpeta docencia compartida sea alineada con la carpeta de Google Drive
```

```
In [10]: !ls "/content/drive/MyDrive/VIU/RNDL/*"
```

Proyecto resources

### Temas avanzados

- Funcional API: Desarrollo de redes neuronales avanzadas
  - Carga de datos desde Google Drive
  - Web scrapping impleando Bing Search API
- Keras Tuner: Optimización automática de hiperparámetros
- Mlflow: Ciclo de vida de un modelo
- Ir más allá

### Funcional API

- Hasta ahora hemos desarrollado redes neuronales secuenciales
- Suficiente para muchos contextos, limitante para otros más complejos
  - Inputs independientes, múltiples outputs, ramificaciones internas, skip connections, retroalimentaciones, etc.

#### Ejemplo X input <-> 1 output: Predicción de precio de ropa de segunda mano

- Input: metadata marca, tiempo usado (one hot encoded), foto (imagen), descripción (texto).
- Modelo con tres submodelos MLP para metadata, RNN para descripción a partir de texto, CNN para imagen)

#### Ejemplo 1 input <-> X output: predicción de año de publicación y estilo de un libro

- Un modelo con dos outputs (clásificadores)
- Desventajas de la alternativa de construir modelos separados:
  - Coste computacional tanto en entrenamiento como en inferencia
  - No se tiene en cuenta toda la información a la vez y analizar la información de manera independiente produce sesgo
  - Se pierden las ventajas de un modelo entrenable end-to-end

### Empleando la functional API: Red MISO CNN+MLP en HOUSE DATASET

- Modelo con dos inputs (atributos y fotos)

• Red híbrida:

- CNN: para trabajar la imagen.

- MLP: para trabajar con datos estructurados.

- Output: precio de las casas

<https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/>

### Mis funciones auxiliares

```
In [11]: # Importar las necesarias packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
import glob
import cv2
import os
import re

Filtrar las zipcodes de codigos postales poco populares (menos de 10 casas)
MIN_HOUSES_PER_ZIPCODE = 20
Eliminamos las imágenes (downsampling)
IMAGE_DIM = (32, 32)

Cargamos los atributos del dataset
def load_house_attributes(inputPath):
 # Cargar el dataset con nombres especificados
 cols = ["bedrooms", "bathrooms", "area", "zipcode", "price"]
 df = pd.read_csv(inputPath, sep=",", header=None, names=cols)

 # Filtramos los codigos postales con menos de MIN_HOUSES se eliminan por no tener representatividad
 for zipcode, count in zip(df.zipcode, counts):
 if count < MIN_HOUSES_PER_ZIPCODE:
 df[df["zipcode"] == zipcode].index
 df.drop(inplace=True)

 return df
```

```
In [12]: # Procesar los atributos
def process_house_attributes(df, train, test):
 # Normalizar (valores > 0) atributos continuos
 continuous = ["bedrooms", "bathrooms", "area"]
 for col in continuous:
 train[col] = cs.fit_transform(train[[col]])
 test[col] = cs.transform(test[[col]])

 # One-hot encode el zipcode
 zipBinarizer = LabelBinarizer().fit(df["zipcode"]) # hace lo mismo que to_categorical()
 trainCategorical = zipBinarizer.transform(train[["zipcode"]])
 testCategorical = zipBinarizer.transform(test[["zipcode"]])

 # Unir todos los atributos y dividir dataset
 trainX = np.hstack((trainCategorical, train[continuous]))
 testX = np.hstack((testCategorical, test[continuous]))

 # One-hot encode el zipcode para
 zipBinarizer = LabelBinarizer().fit(df["zipcode"])
 trainCategorical = zipBinarizer.transform(train[["zipcode"]])
 testCategorical = zipBinarizer.transform(test[["zipcode"]])

 # Unir todos los atributos y dividir dataset
 trainX = np.hstack((trainCategorical, train[continuous]))
 testX = np.hstack((testCategorical, test[continuous]))

 # return (trainX, testX)
 return (trainX, testX)
```

```
In [13]: # Cargar imágenes
def load_house_images(df, inputPath):
 images = []

 # cada linea es un data point
 for i in df.index.values:
 # cargar la cuatro imágenes por casa
 basePath = os.path.sep.join([inputPath, "{}".format(i + 1)])
 images.append(basePath)

 # leer la primera imagen (resize it to 32x32, and then
 # update the list of input images
 image = cv2.imread(basePath)
 image = cv2.resize(image, IMAGE_DIM)
 inputImages.append(image)

 # crear un mosaico de 64x64 para meter 4 imágenes en una misma img
 # title the four input images in the output image such the first
 # image goes to the top-right corner, the second image to the bottom-right corner,
 # and the final image to the bottom-left corner
 outputImage[0:IMAGE_DIM[0], 0:IMAGE_DIM[1]] = inputImages[0]
 outputImage[0:IMAGE_DIM[0], IMAGE_DIM[1]:IMAGE_DIM[1]*2] = inputImages[1]
 outputImage[IMAGE_DIM[0]:, 0:IMAGE_DIM[1]] = inputImages[2]
 outputImage[IMAGE_DIM[0]:, IMAGE_DIM[1]:] = inputImages[3]

 # add the third image to our set of images the network will be
 # trained on
 images.append(outputImage)

 # return our set of images
 return np.array(images) / 255.0
```

### Creando la rama MLP

```
In [25]: from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Dropout, Conv2D, BatchNormalization, MaxPooling2D, Activation

Creacion del modelo MLP (para los atributos numericos)
def create_mlp(dim, regress=False):
 model = Sequential()
 model.add(Dense(8, input_dim=dim, activation="relu"))
 model.add(Dense(4, activation="relu"))

 # check to see if the regression node should be added
 if regress:
 model.add(Dense(1, activation="linear"))

 # return our model
 return model
```

### Creando la rama CNN

Creamos tantos bloques convolucionales como filtros se nos pasen por argumento a la función

```
In [15]: # Crear el modelo CNN
def create_cnn(width, height, depth, filters=(16, 32, 64), regress=False):
 inputShape = (height, width, depth)
 chanDim = -1

 # define the model input
 inputs = Input(shape=inputShape)

 # crear tantas capas como filtros pasados
 for f, i in enumerate(filters):
 if i == 0:
 x = inputs

 # CONV => RELU => POOL
 x = Conv2D(f, (3, 3), padding="same")(x)
 x = Activation("relu")(x)
 x = BatchNormalization(axis=chanDim)(x)
 x = MaxPooling2D(pool_size=(2, 2))(x)

 # Top maxpool
 x = Dense(16)(x)
 x = Activation("relu")(x)
 x = Dense(8)(x)
 x = Activation("relu")(x)

 # Middle output layer
 x = Dense(4)(x)
 x = Activation("relu")(x)

 # Check to see if the regression node should be added -unicamente con la img
 if regress:
 x = Dense(1, activation="linear")(x)

 model = Model(inputs, x)

 # return our model
 return model
```

### Importando datos desde Google Drive: House Dataset

```
In [29]: # Cargar los atributos numericos
print('Leyendo atributos...')
inputPath = BASE_FOLDER+'resources/Houses-dataset/HousesInfo.txt'
df = load_house_attributes(inputPath)

Cargar imágenes
print('INFO! loading house images...')
images = load_house_images(df, BASE_FOLDER+'resources/Houses-dataset')

Cargando atributos...
[INFO] loading house images...
```

```
In [31]: images.shape
(384, 64, 64, 3)
```

```
Out[31]:
```

### Acondicionamiento de datos

```
In [30]: # Dividir imágenes en train y test
split = train_test_split(df, images, test_size=.25, random_state=42)
trainAttrX, testAttrX, trainImagesX, testImagesX = split

Normalizar el precio de las casas - (max -> 1)
maxPrice = trainAttrX["price"].max()
trainAttrX["price"] = maxPrice
testAttrX["price"] = maxPrice

Procesar atributos numerosicos
(trainAttrX, testAttrX) = process_house_attributes(df, trainAttrX, testAttrX)
```

### Construcción del modelo híbrido

Para hibridar las dos ramas creadas anteriormente tenemos que tener en cuenta para poder hacerlo lo siguiente:

Hay que respetar que cada una de las ramas por donde hibrida (el final de cada una de las ramas), tengan la misma dimensionalidad.

Para concatenar los vectores de activación, se utiliza la capa concatenate. Se obtendrá un vector de 8 características (4 CNN y 4 MLP), el cual pasa a 4 mediante una hidden layer de 4 neuronas, y de ahí a una capa de salida con 1 neurona (regresión "linear")

El modelo final se compone de las inputs (concatenación de la entrada a la MLP y a la CNN) y la capa de output.

```
In [32]: # Crear los dos modelos (MLP y CNN)
mlp = create_mlp(lenAttrX.columns, regress=False)
cnn = create_cnn(IMAGE_DIM[0], 2, IMAGE_DIM[1]*2, 3, regress=False)

Unir ambas ramas en una única de salida (concatenarlos)
combinedInput = concatenate([mlp.output, cnn.output])

Clasificador final tras la concatenación
x = Dense(4, activation="relu")(combinedInput)
x = Dense(1, activation="linear")(x)

Construir el modelo final
model = Model(inputs=[mlp.input, cnn.input], outputs=x)

return our model
return model
```

### Compilación y entrenamiento de nuestra red MISO híbrida

Para hacer el .fit, pasaremos los datos de entrada de las DOS ramas, concatenando los atributos con las imágenes.

```
In []: from tensorflow.keras.optimizers import Adam
Utilizar el optimizador Adam (con learning rate adaptativo)
opt = Adam(lr=1e-3, decay=1e-3 / 200)
model.compile(loss="mean_absolute_error", optimizer=opt)

Entrenar el modelo
print("Entrenar el modelo...")
H = model.fit([trainAttrX, trainImagesX], trainY,
 validation_data=(testAttrX, testImagesX), testY,
 epochs=50, batch_size=8)
```

Entrenar en 288 muestras, validate en 96 samples

Epoch 1/288 [=====] - 4s 12ms/sample - loss: 1058.0105 - val\_loss: 697.4843

Epoch 2/288 [=====] - 3s 9ms/sample - loss: 702.0007 - val\_loss: 220.6218

Epoch 3/288 [=====] - 3s 9ms/sample - loss: 619.5001 - val\_loss: 224.6528

Epoch 4/288 [=====] - 3s 9ms/sample - loss: 446.5802 - val\_loss: 241.4039

Epoch 5/288 [=====] - 3s 9ms/sample - loss: 343.1163 - val\_loss: 213.5732

Epoch 6/288 [=====] - 3s 9ms/sample - loss: 307.4464 - val\_loss: 143.5633

Epoch 7/288 [=====] - 3s 9ms/sample - loss: 216.2664 - val\_loss: 168.7331

Epoch 8/288 [=====] - 3s 9ms/sample - loss: 237.6755 - val\_loss: 181.929

Epoch 9/288 [=====] - 3s 9ms/sample - loss: 111.5975 - val\_loss: 123.7346

Epoch 10/288 [=====] - 3s 9ms/sample - loss: 171.3995 - val\_loss: 83.0598

Epoch 11/288 [=====] - 3s 9ms/sample - loss: 101.3985 - val\_loss: 67.3131

Epoch 12/288 [=====] - 3s 9ms/sample - loss: 118.2108 - val\_loss: 129.7954

Epoch 13/288 [=====] - 3s 9ms/sample - loss: 149.7940 - val\_loss: 97.8922

Epoch 14/288 [=====] - 3s 9ms/sample - loss: 111.5975 - val\_loss: 73.0377

Epoch 15/288 [=====] - 3s 9ms/sample - loss: 84.7919 - val\_loss: 55.4323

Epoch 16/288 [=====] - 3s 9ms/sample - loss: 118.2108 - val\_loss: 40.8653

Epoch 17/288 [=====] - 3s 9ms/sample - loss: 97.6585 - val\_loss: 69.3508

Epoch 18/288 [=====] - 3s 9ms/sample - loss: 105.3505 - val\_loss: 74.3702

Epoch 19/288 [=====] - 3s 9ms/sample - loss: 109.4004 - val\_loss: 73.0377

Epoch 20/288 [=====] - 3s 9ms/sample - loss: 84.7919 - val\_loss: 55.4323

Epoch 21/288 [=====] - 3s 9ms/sample - loss: 91.9964 - val\_loss: 55.4323

Epoch 22/288 [=====] - 3s 9ms/sample - loss: 76.8577 - val\_loss: 47.6653

Epoch 23/288 [=====] - 3s 9ms/sample - loss: 56.4553 - val\_loss: 39.8466

Epoch 24/288 [=====] - 3s 9ms/sample - loss: 55.3821 - val\_loss: 38.2302

Epoch 25/288 [=====] - 3s 9ms/sample - loss: 59.4435 - val\_loss: 36.7895

Epoch 26/288 [=====] - 3s 9ms/sample - loss: 50.6526 - val\_loss: 38.5670

Epoch 27/288 [=====] - 3s 9ms/sample - loss: 40.3991 - val\_loss: 34.3749

Epoch 28/288 [=====] - 3s 9ms/sample - loss: 50.8622 - val\_loss: 39.4350

Epoch 29/288 [=====] - 3s 9ms/sample - loss: 52.4340 - val\_loss: 36.7882

Epoch 30/288 [=====] - 3s 9ms/sample - loss: 30.4713 - val\_loss: 36.5957

Epoch 31/288 [=====] - 3s 9ms/sample - loss: 37.0972 - val\_loss: 32.7397

Epoch 32/288 [=====] - 3s 9ms/sample - loss: 32.8182 - val\_loss: 30.8959

Epoch 33/288 [=====] - 3s 9ms/sample - loss: 30.3741 - val\_loss: 28.9837

Epoch 34/288 [=====] - 3s 9ms/sample - loss: 37.0972 - val\_loss: 33.7557

Epoch 35/288 [=====] - 3s 9ms/sample - loss: 32.8182 - val\_loss: 30.8959

Epoch 36/288 [=====] - 3s 9ms/sample - loss: 30.4399 - val\_loss: 34.3749

Epoch 37/288 [=====] - 3s 9ms/sample - loss: 39.3350 - val\_loss: 32.7150

Epoch 38/288 [=====] - 3s 9ms/sample - loss: 37.0972 - val\_loss: 33.7557

Epoch 39/288 [=====] - 3s 9ms/sample - loss: 32.8182 - val\_loss: 30.8959

Epoch 40/288 [=====] - 3s 9ms/sample - loss: 37.0972 - val\_loss: 33.7557

Epoch 41/288 [=====] - 3s 9ms/sample - loss: 27.2256 - val\_loss: 26.7590

Epoch 42/288 [=====] - 3s 9ms/sample - loss: 30.5482 - val\_loss: 27.3641

Epoch 43/288 [=====] - 3s 9ms/sample - loss: 27.9876 - val\_loss: 31.0481

Epoch 44/288 [=====] - 3s 9ms/sample - loss: 29.2665 - val\_loss: 27.7499

Epoch 45/288 [=====] - 3s 9ms/sample - loss: 28.7456 - val\_loss: 28.8850