



Universidad
Francisco de Vitoria
UFV Madrid

Inteligencia Artificial II

Tema 6:
Redes convolucionales y
Autoencoders

Dos modelos de DEEP LEARNING



Objetivos del tema

- Ubicación
 - Bloque II: **COMPUTACION NEURONAL**
 - Tema 2: *Aprendizaje no supervisado: Aprendizaje competitivo*
 - Tema 3: *Aprendizaje supervisado: Modelos lineales*
 - Tema 4: *Aprendizaje supervisado: Modelos no lineales*
 - Tema 5: *Modelos Deep Learning*
 - Tema 6: *Redes convolucionales y Autoencoders*
- Objetivos
 - Estudiar el modelo de **Red Neuronal Convolutiva**, su **arquitectura** y sus aplicaciones
 - Estudiar el modelo de **autoencoders** y su arquitectura



1. Convolución
 1. Operador Convolución (~~MATEMÁTICO~~)
 2. Aplicación del operador
2. Convolutional Neural Networks (~~CNN~~)
 1. Definición
 2. Por qué usar la convolución
 3. Arquitectura
 4. Hiperparámetros
3. Deep Autoencoders
 1. Definición
 2. Procesamiento
 3. Tipos de autoencoders
 4. Aplicaciones



1. Convolución

1. Operador Convolución
2. Aplicación del operador

2. Convolutional Neural Networks

1. Definición
2. Por qué usar la convolución
3. Arquitectura
4. Hiperparámetros

3. Deep Autoencoders

1. Definición
2. Procesamiento
3. Tipos de autoencoders
4. Aplicaciones

1.1 Operador Convolución (NADA QUE VER CON REDES)



Operador matemático que transforma dos funciones f y g de una sola variable en una nueva función $f * g$ que representa la magnitud en la que se superponen f y una versión trasladada e invertida de g .

CONTINUA

$$f(t) * g(t) = \{f * g\}(t) = h(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau$$

HAY DOS TIPOS

f, g continuas

f, g discontinuas

- Se puede asimilar al concepto de "media móvil ponderada"
- Si las funciones son discretas (por ejemplo, píxeles de una imagen) la convolución se calcula como

DISCONTINUA

$$f(t) * g(t) = h(t) = \sum_{\tau} f(\tau)g(t - \tau)$$

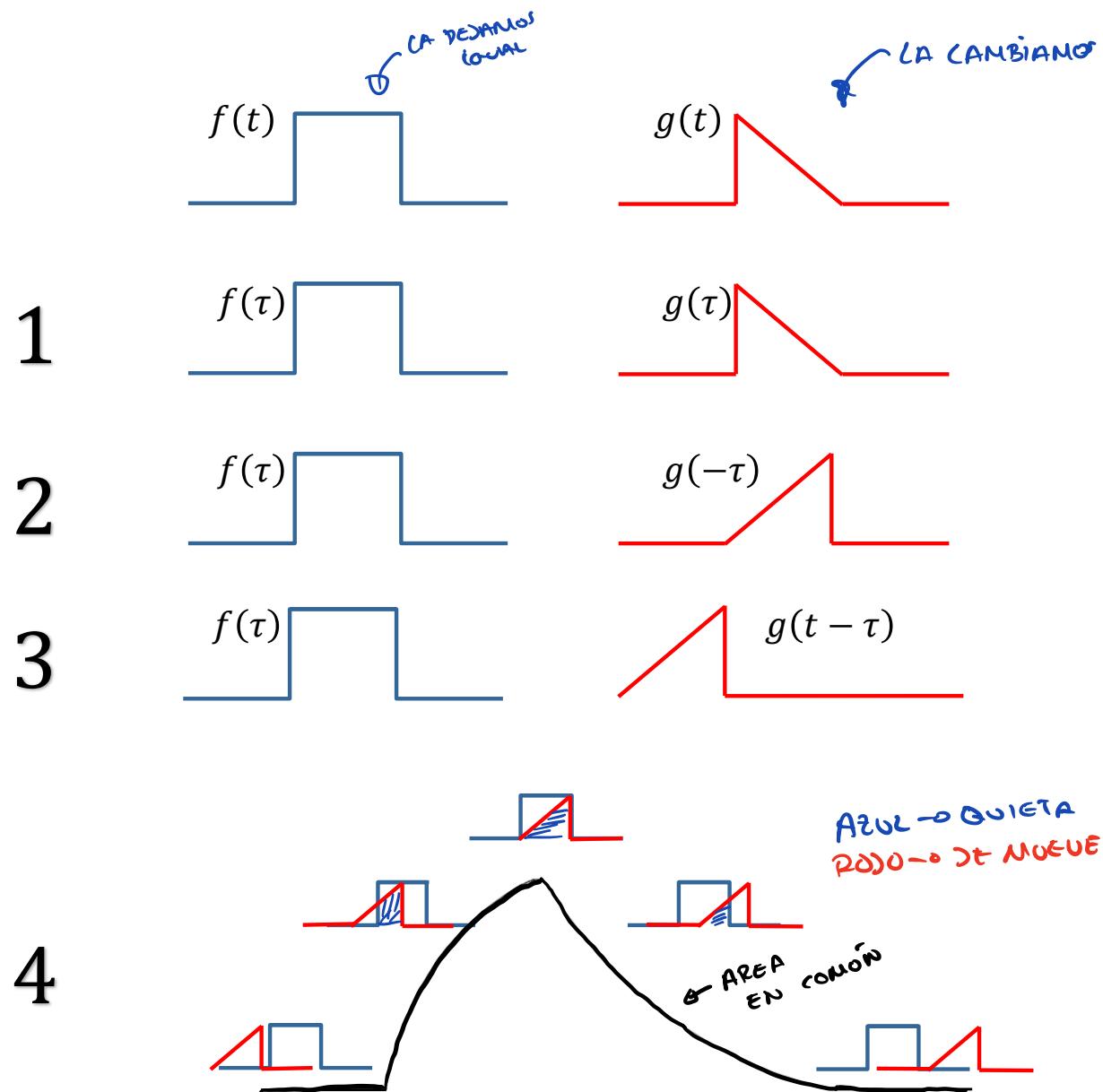


1.1 Operador Convolución

- Cálculo de la convolución de dos funciones f y g
 1. Se expresan f y g en términos de una variable ficticia τ ← CAMBIO DE VARIABLE
 2. Se refleja una de las funciones: $g(\tau) \rightarrow g(-\tau)$ ← CAMBIO DE SIGNO
 3. Se cambia la variable $\tau \rightarrow t-\tau$ lo que desplaza el origen de g sobre el eje τ hacia el punto t .
 4. Se varía t entre $(-\infty, +\infty)$ lo que desliza g sobre el eje τ . Siempre que las dos funciones se intersequen, se encuentra la integral de su producto. ← AREA COMÚN DE LAS DOS FUNCIONES
- Es decir, se calcula el promedio ponderado desplazado de la función $f(\tau)$, donde la función peso es $g(-\tau)$.

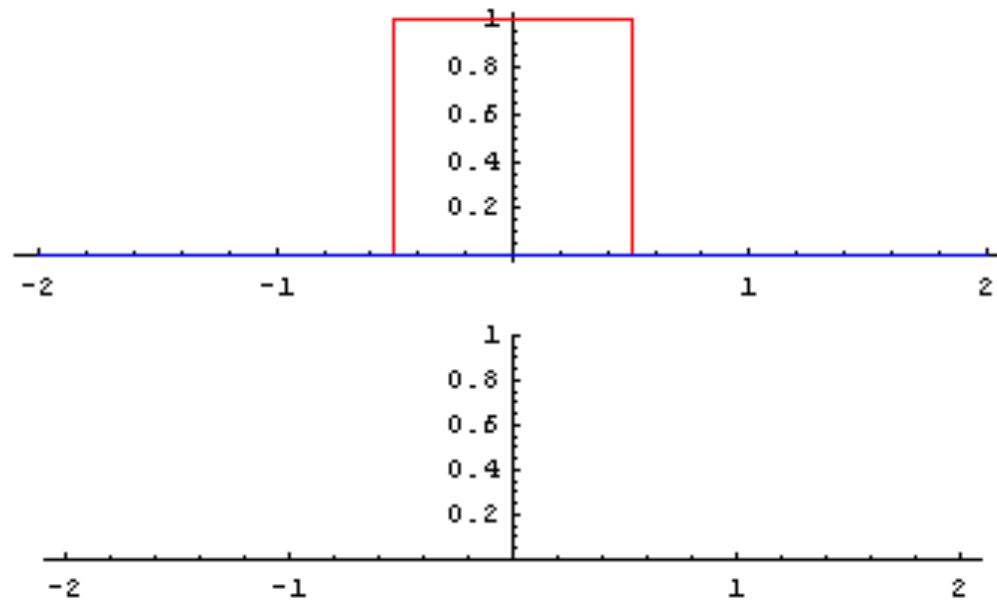


1.1 Operador Convolución





1.1 Operador Convolución





1.2 Aplicación del operador

- Normalmente la convolución se aplica a la descripción matemática de señales, por ejemplo en
 - **Acústica para describir el eco**
 - Convolución del sonido original con una función que representa los objetos que lo reflejan
 - **Óptica para describir manchas o sombras:**
 - Una fotografía desenfocada es la convolución de la imagen correcta con el círculo borroso formado por el diafragma del objetivo
 - La sombra de un objeto es la convolución de la forma de la fuente de luz y del objeto cuya sombra se está proyectando

1.2 Aplicación del operador



Hablando de imágenes...

Cada pixel no es independiente.

		0.2	0.2		
	0.2			0.2	
	0.2	0.2	0.2	0.2	
	0.2			0.2	

- La imagen significa algo precisamente porque cada píxel está en una posición determinada respecto a sus vecinos.
- *Un círculo sigue siendo un círculo aunque lo desplacemos dentro de la imagen.*
- No se pueden considerar píxeles aislados sino un píxel y su entorno (vecindario).



1.2 Aplicación del operador

- Las operaciones con imágenes se llevan a cabo aplicando filtros y transformaciones que convierten una imagen en otra.

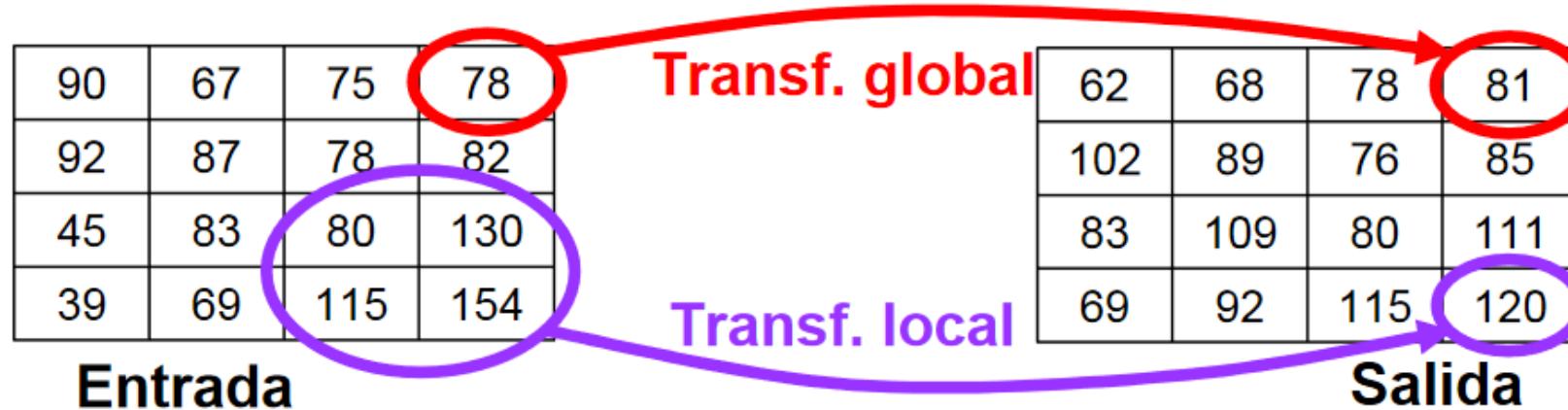
DOS TIPOS DE TRANSFORMACIONES

- Transformaciones globales:
 - Cada píxel de salida depende sólo de un píxel de entrada y no se tiene en cuenta la relación de vecindad entre píxeles.
 - El resultado no varía si los píxeles son permutados aleatoriamente y después reordenados.
- Transformación local: el valor de un píxel depende de la vecindad local de ese píxel.
 - Las transformaciones locales tienen sentido porque existe una relación de vecindad entre los píxeles



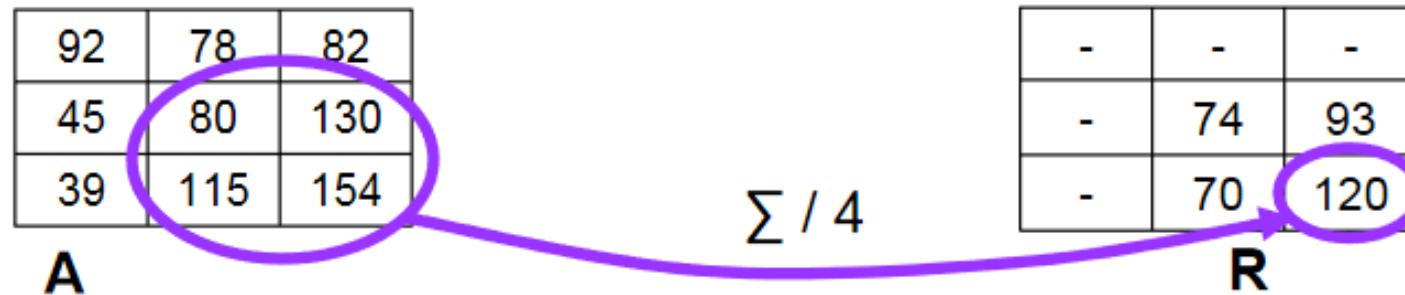
1.2 Aplicación del operador

- Ejemplo de transformación global/local



- Ejemplo de transformación (filtro) local: Filtro de la media

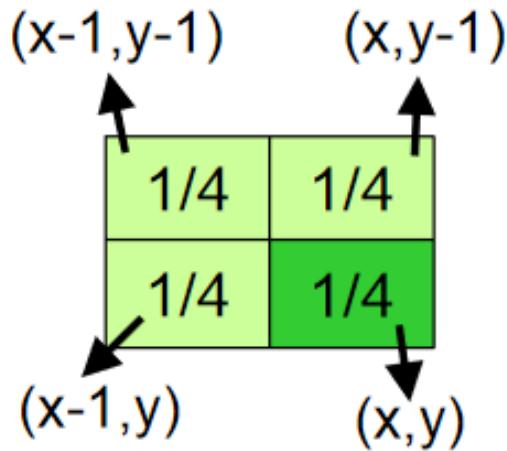
$$R(x,y) = \{A(x-1,y-1)+A(x,y-1)+A(x-1,y)+A(x,y)\} / 4$$





1.2 Aplicación del operador

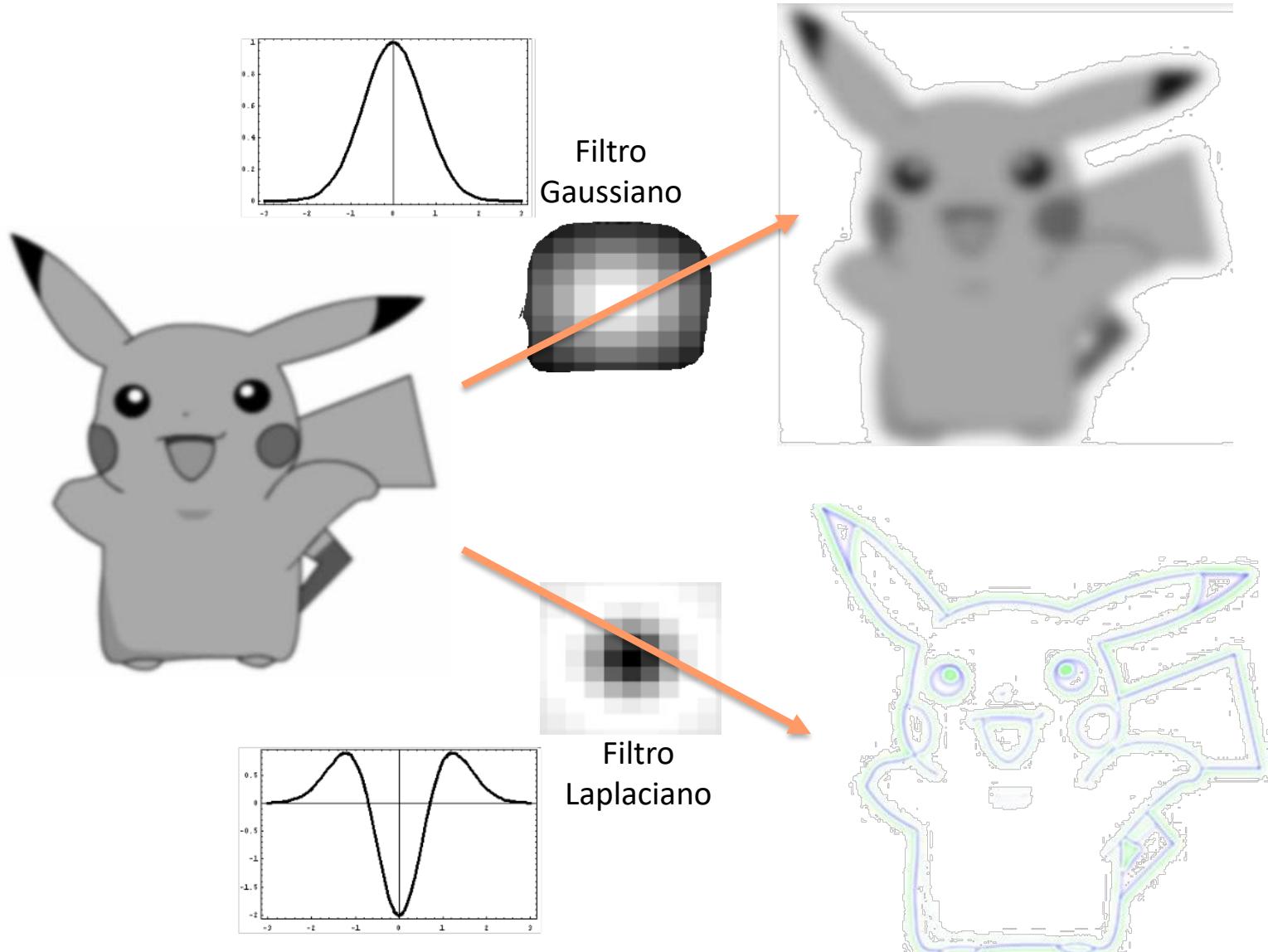
- Convolución (no hablamos todavía de CNNs):
 - Transformación local
 - Operación matemática (producto escalar) de un conjunto de píxeles cercanos con una pequeña matriz (*máscara/kernel/filtro*)
- El filtro de la media es una convolución
$$R(x,y) = \frac{1}{4} A(x-1,y-1) + \frac{1}{4} A(x,y-1) + \frac{1}{4} A(x-1,y) + \frac{1}{4} A(x,y)$$





1.2 Aplicación del operador

- Diferentes filtros producen diferentes salidas





1.2 Aplicación del operador

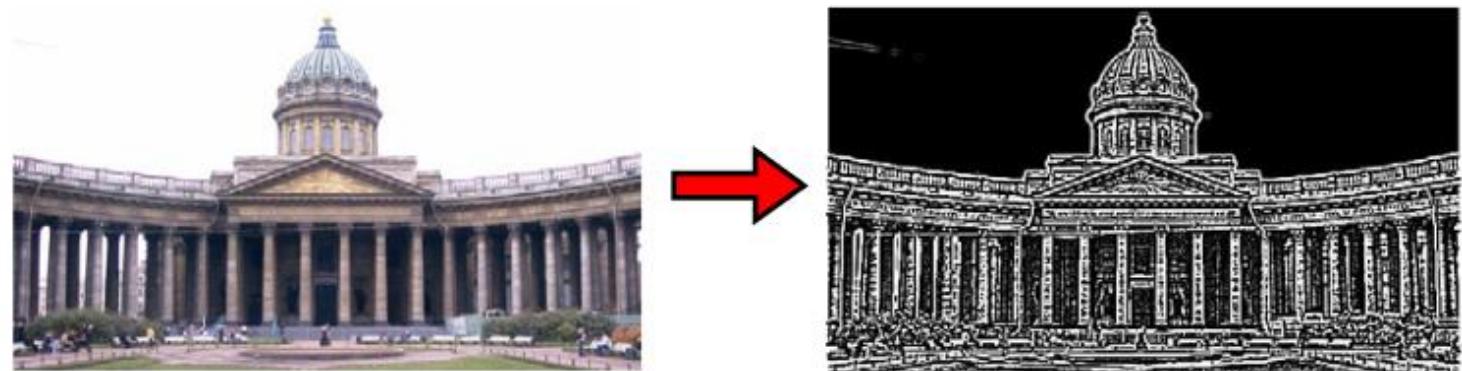
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

El valor de un pixel de salida es la media de los pixeles de entrada



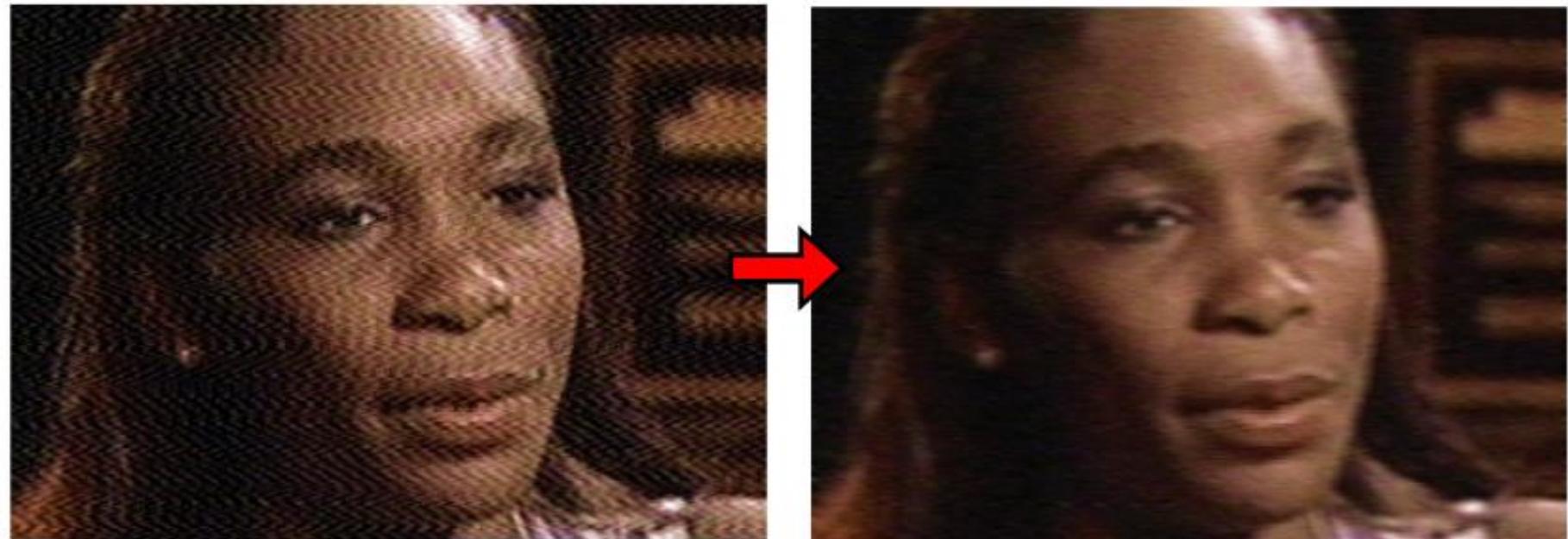
-1	1
----	---

El valor de un pixel de salida se obtiene restándole el valor del pixel de su izquierda





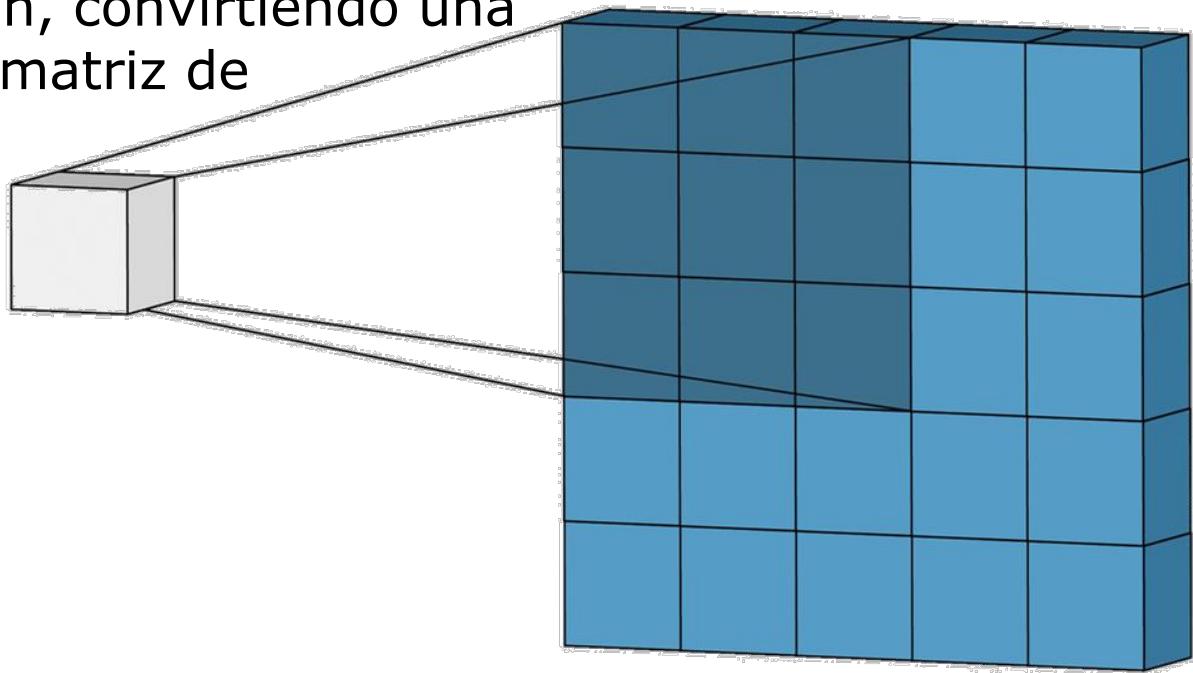
1.2 Aplicación del operador





1.2 Aplicación del operador

- La convolución 2D es una operación simple:
 - El *kernel* realiza una multiplicación de cada peso con el pixel correspondiente de la imagen de entrada sobre la que se encuentra y suma los resultados en una única celda de salida.
 - El *kernel* "se desliza" (de izquierda a derecha y de arriba a abajo) sobre los datos de entrada 2D repitiendo este proceso para cada ubicación, convirtiendo una matriz 2D en otra matriz de salida 2D.



1.2 Aplicación del operador



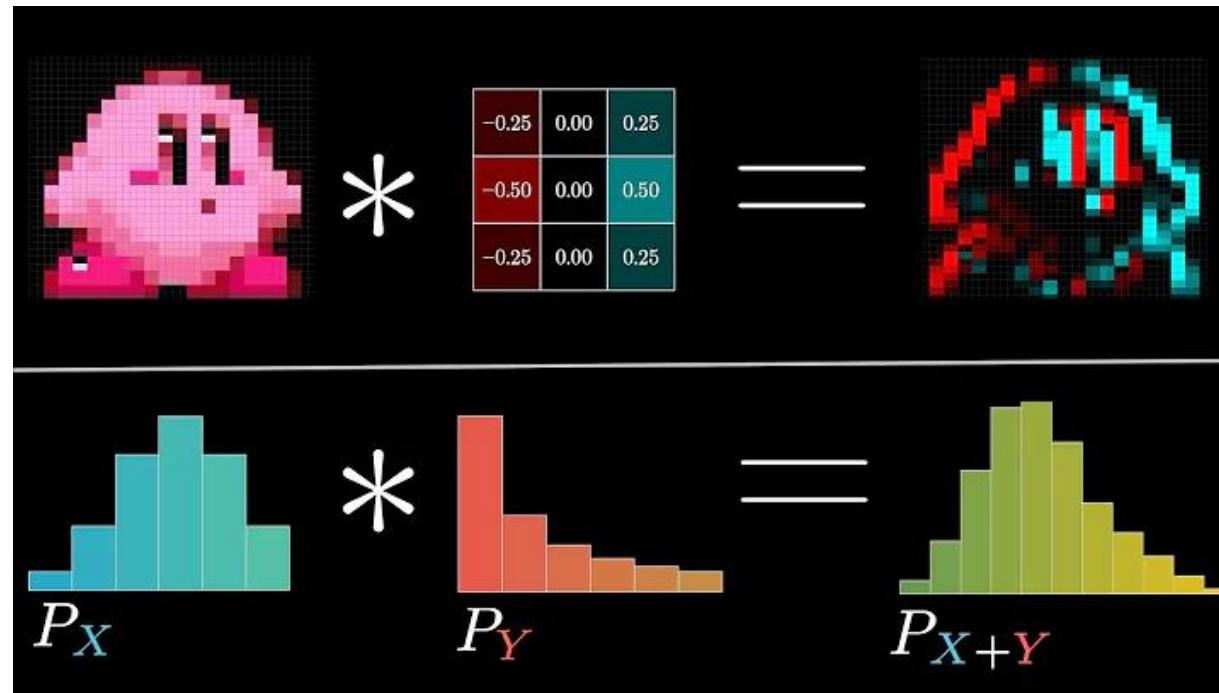
The screenshot shows a convolution operation demo interface. At the top left is the logo and title "CONVOLUTION OPERATION DEMO". On the top right are "Application" and a gear icon. Below the title are four buttons: "Full Screen", "MNIST", "0 - Zero", "Top Edge Filter", "Step", and "Play". The main area displays a convolution diagram with a window of size 3x3 and a filter of size 3x3. The output value is 0.0. Below this is a row of operations showing the calculation of the output value from an input row. The input row has values: 0.0, -1.0, +, 0.0, -1.0, +, 0.0, -1.0, +, 0.0, 1.0, +, 0.0, 1.0, +, 0.0, 1.0, +, 0.0, 0.0, +, 0.0, 0.0, +, 0.0, 0.0 = 0.0. The input and output rows are labeled "Input" and "Output" respectively. A green box highlights the first element of the input row.



AVANZADO

Entendiendo el operador convolución | 3Blue1Brown

(hasta el minuto 13:40)





1. Convolución

1. Operador Convolución
2. Aplicación del operador

2. Convolutional Neural Networks **CNN**

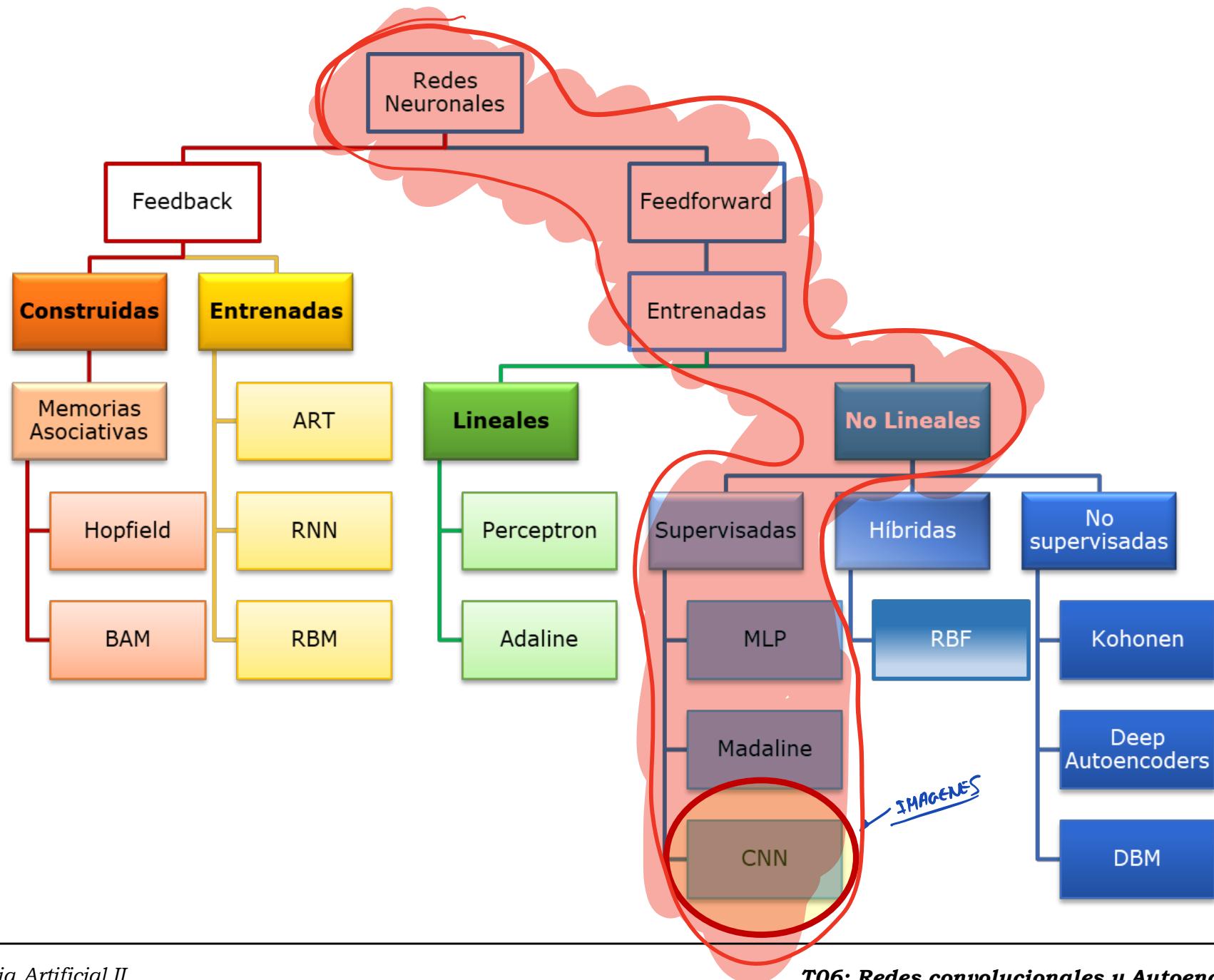
1. Definición
2. Por qué usar la convolución
3. Arquitectura
4. Hiperparámetros

3. Deep Autoencoders

1. Definición
2. Procesamiento
3. Tipos de autoencoders
4. Aplicaciones



2.1 Definición





2.1 Definición

✳️ *Redes que pueden aprender relaciones entrada-salida basadas en operaciones de convolución, donde la entrada es una imagen*

- Basada en cómo se estructuran realmente las neuronas en nuestro sistema visual.
 - Los patrones de conectividad entre neuronas asemejan la organización del córtex visual de los animales
 - Las neuronas corticales responden a estímulos en un área del campo visual (*campo receptivo*) que se solapa parcialmente con el de otras neuronas cubriendo así todo el campo visual
- Reemplazan la multiplicación de matrices del MLP ($\vec{X} \times \vec{W}_i$) por la convolución ($\vec{X} * \vec{W}_i$) → **NATRIT PESOS: FILTRO**
- Cada conjunto de convoluciones extrae jerárquica e incrementalmente alguna *característica* de la imagen que se procesa

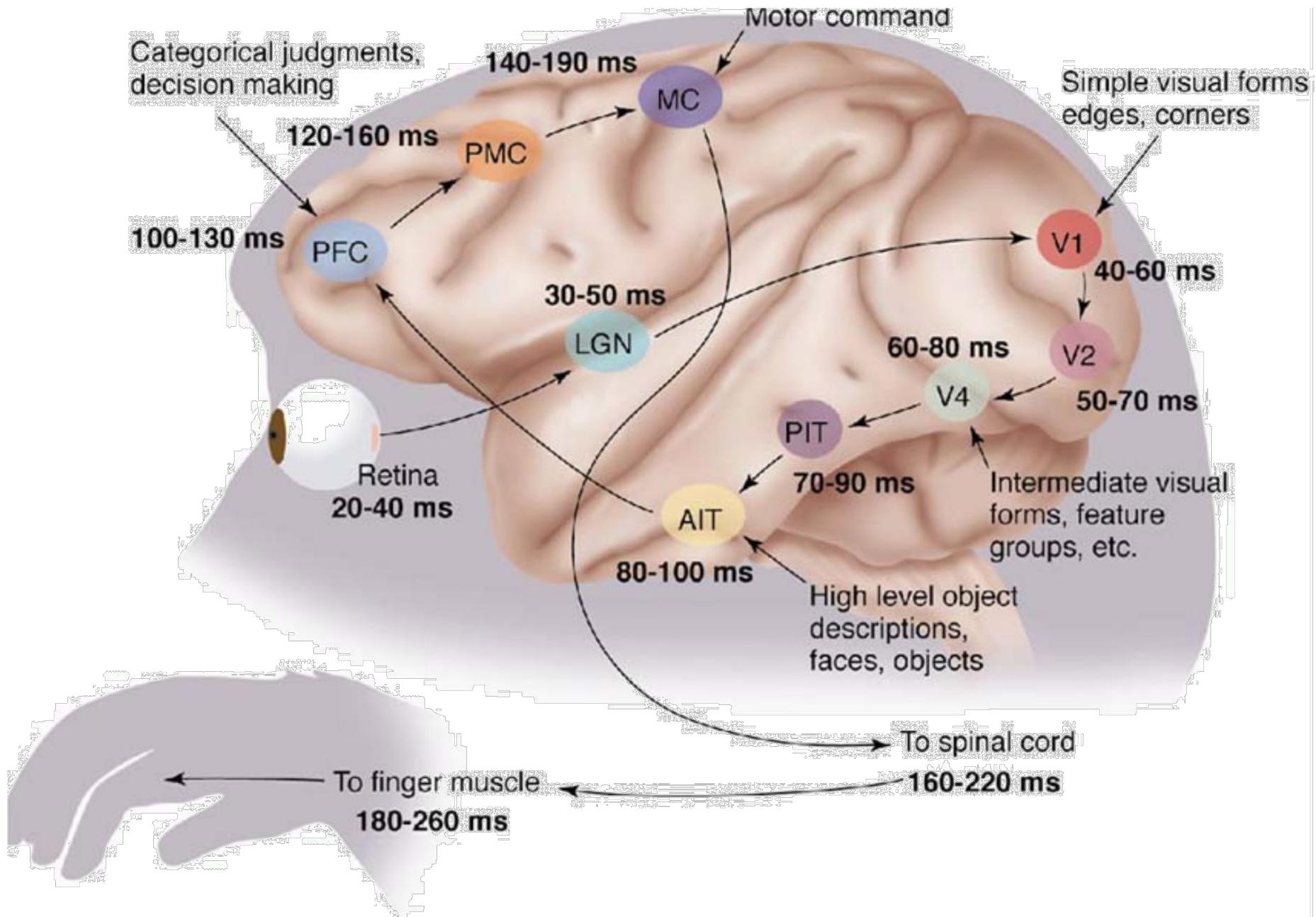


2.1 Definición

- Los primeros modelos se basaron en modelos computacionales que exhiben características similares a las del neocortex
 - El cerebro humano es una arquitectura neuronal profunda
 - El desarrollo neocortical (propuesto en 1990) fue simulado por modelos computacionales similares a las actuales DNN (*neocognitrón* de Fukushima)
 - Al igual que el neocórtex, las CNN emplean una jerarquía de filtros apilados en las que cada capa toma información de un filtro anterior (o el entorno) y pasa su salida a otras capas



2.1 Definición





2.1 Definición

Aplicaciones de las CNN

- 1D CNN: texto, señales secuenciales
 - Clasificación de texto
 - Reconocimiento de género musical
 - Modelos acústicos para reconocimiento de hablas
 - Predicción de series temporales (poco)
- 2D/3D CNN: imágenes escala de grises/color, señales en frecuencias (habla y audio)
 - Detección de objetos, localización, reconocimiento
- 4D CNN: video, imágenes volumétricas/tomográficas
 - Reconocimiento/comprepción de video
 - Análisis de imágenes biomédicas
 - Análisis de imágenes hiperespectrales



2.1 Definición

- Algunos modelos famosos

- **LeNet.** LeCun (1990's). Lee códigos postales, dígitos, etc.
- **AlexNet.** Krizhevsky, Sutskever y. Hinton. **Gana ILSVRC 2012** (error en el top 5 del 16% frente al segundo clasificado con un error del 26%). Primera con capas convolucionales apiladas.
- **ZF Net.** Zeiler y Fergus. **Gana ILSVRC 2013**. Mejora de AlexNet mediante el ajuste de los hiperparámetros de la arquitectura.
- **GoogLeNet.** Szegedy et al. de Google. **Gana ILSVRC 2014**. Reduce drásticamente el número de parámetros de la red (4M, comparado con AlexNet con 60M). Versión más reciente Inception-v4.
- **VGGNet.** Simonyan y Zisserman. **Subcampeón ILSVRC 2014**. 16 capas CONV/FC. Utiliza mucha memoria y parámetros (la mayor parte en la primera capa FC pero se descubrió que estas capas FC pueden eliminarse sin que el rendimiento disminuya).
- **ResNet.** Kaiming He et al. **Gana ILSVRC 2015**. Conexiones de salto especiales y sin capas FC al final de la red. Son actualmente, con diferencia, los modelos de Redes Neuronales Convolucionales más avanzados y la opción por defecto en la práctica.

* ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*



2.2 Por qué usar la convolución

MOTIVOS PARA USAR CNN

- **Tamaño de la red**
 - Cuanto mayor es la entrada del codificador, más pesos hay que entrenar y más lento (y peor) es el entrenamiento.
 - Con una imagen **a color** de 1.000×1.000 pixeles, una capa oculta de 1.000 neuronas fully connected tendría **3.000 millones** de pesos. La convolución reduce drásticamente el número de pesos que hay que entrenar.
- **Invarianza de la posición**
 - Si usamos imágenes enteras, una imagen con un rostro centrado y otra con el mismo rostro desplazado hacia un lado son completamente diferentes para el codificador.
 - No hay que considerar píxeles aislados sino un píxel y su entorno (vecindario).



2.2 Por qué usar la convolución

- Tamaño de la red

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

↑
RESULTADO DEL
ENTRENAMIENTO. NO
DEBEN SER PONER A PRIORI.

- $5 \times 5 = 25$ entradas
- $3 \times 3 = 9$ **PESOS**
 - Si fuera una capa fully-connected estándar, tendría una matriz de pesos de $25 \times 9 = 225$ valores y cada característica de salida será la suma ponderada de cada característica de entrada.
 - La convolución permite hacer esta transformación con solo 9 parámetros (pesos).



2.2 Por qué usar la convolución

Invarianza de la posición

- La operación de convolución recibe como *entrada* una imagen y aplica sobre ella n *neuronas/filtros/kernel*

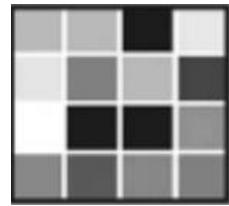
¡¡¡CUIDADO!!! *Esto solo es válido cuando la entrada es un solo canal (imagen en escala de grises)*
- Cada neurona/filtro devuelve un mapa de características de la imagen original Se trabaja por separado en cada color (R,G,B)
 - Un mapa de características es una matriz donde cada celda es una combinación lineal de los pixeles de entrada con los valores del kernel (pesos).
 - Las características de salida corresponden a las entidades de entrada ubicadas en la capa de entrada aproximadamente en la misma posición que la celda de salida.



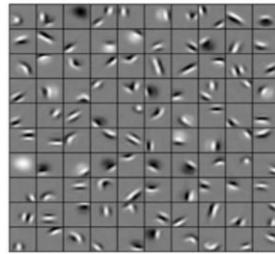
2.2 Por qué usar la convolución

- Estas características son invariantes respecto a la posición dentro de la imagen original y corresponden a cada posible ubicación del filtro en la imagen original.
- El mismo filtro (neurona) sirve para extraer la misma característica en cualquier parte de la entrada ya que se replica en todo el campo visual.
- En el caso de GoogLeNet
 - La red pasa de una primera convolución con pocos filtros (64), que detectan características de bajo nivel
 - A una última convolución con muchos filtros (1024), cada uno buscando una característica de alto nivel muy específica.
 - A continuación, una capa de pooling colapsa cada cuadrícula de 7×7 en un solo píxel,
 - Cada canal detecta una característica con un campo receptivo equivalente a toda la imagen.

2.2 Por qué usar la convolución CADA CAPA ENCUENTRA ALGO.

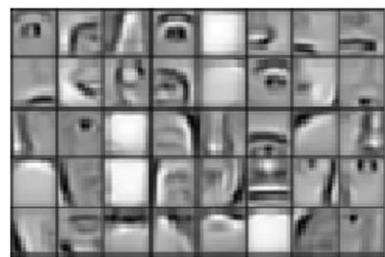


Características de bajo nivel



Capa 1: La red identifica píxeles con distinto nivel de gris/color

Características de nivel medio



Capa 2: La red aprende a identificar bordes y formas simples

Características de alto nivel



Capa 3: La red aprende a identificar formas más complejas y objetos

Las sucesivas capas operan no sobre la imagen original, sino sobre los mapas de características producidas por la capa anterior

Clasificador entrenable

Capa 4: La red aprende qué formas y objetos se pueden usar para definir una cara humana

Capa 5: La red clasifica la salida



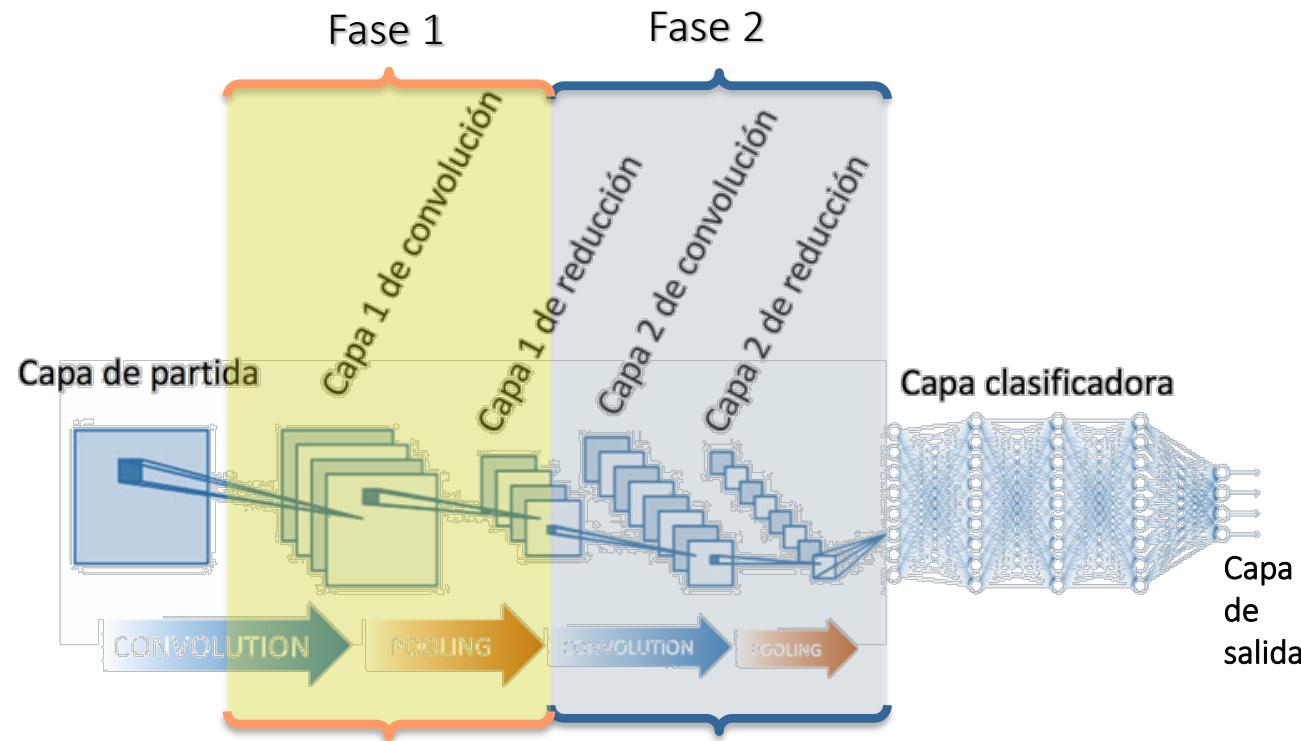
2.3 Arquitectura

- **Feedforward**: Variación (compleja) del MLP
 - **Locally connected**: Existen capas, pero se restringen las conexiones entre capas (no es fully connected)
 - Cada neurona se conecta a una región pequeña y contigua de pixeles de la entrada o a un subconjunto de unidades de la capa anterior lo que reduce drásticamente el número de pesos.
 - **Capas**:
 - Input (**ENTRADA**)
 - Múltiples Hidden layers del tipo:
 - Capa(s) de *convolución*
 - Capa(s) de *reducción (pooling)*
 - Capa(s) *clasificadora (fully connected)*
 - Output: Última capa clasificadora
- HAY MUCHAS DE ESTOS 3 TIPOS.
- ← LA ÚLTIMA CAPA OUT CLASIFICA. ES UN NLP



2.3 Arquitectura

- *Arquitectura completa de la red*: Una CNN se organiza en una o más convoluciones acabadas en un clasificador





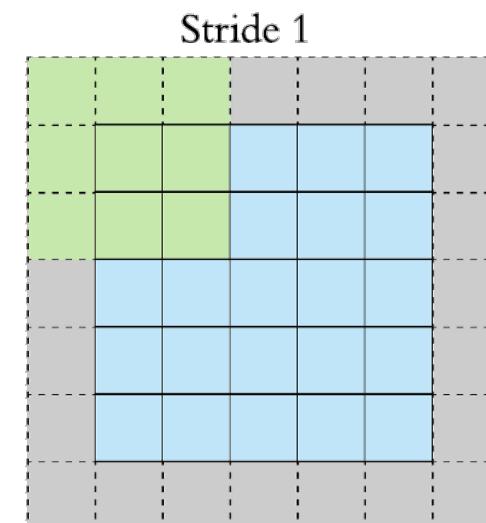
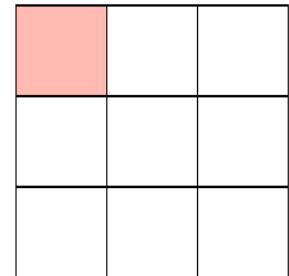
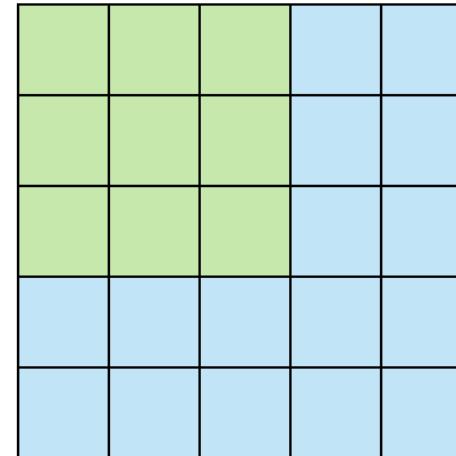
Capa de convolución

- *Capa compuesta por neuronas convolucionales que detectan patrones aplicando varios filtros de 1 o más dimensiones a los datos que le llegan*
 - Cada filtro es una neurona. Puede haber más de un filtro por capa.
 - Cada neurona realiza la convolución de una imagen que recibe como *input* con su *filtro o kernel* devolviendo como *output* un *mapa de características* de la imagen original.
 - Habrá tantos mapas de características como neuronas
 - Los pesos que se ajustan durante el entrenamiento son los parámetros del filtro

2.3 Arquitectura

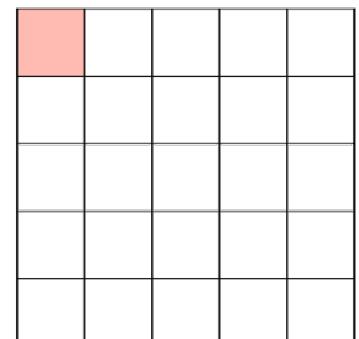


- Parámetros a determinar a la hora de definir los filtros:
 - Número de filtros (**neuronas**)
 - Dimensiones de los filtros (**pesos por neurona**)
 - **Stride**: cuantas filas/columnas avanzamos en cada operación de convolución. Se define para cada filtro de la capa
 - **Padding**: añadir ceros simétricamente a la matriz de entrada (en filos y/o columnas) para mantener las mismas dimensiones en la matriz de salida (**zero padding**).



Stride 1

Feature Map



Stride 1 with Padding

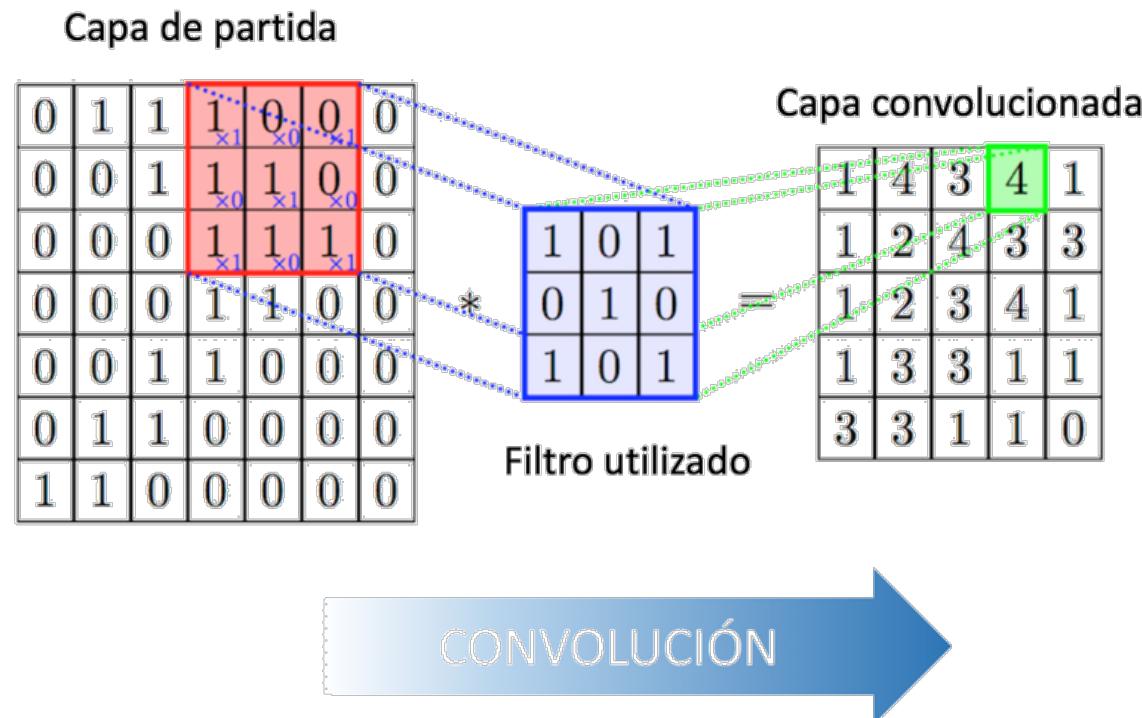
Feature Map



2.3 Arquitectura

Ejemplo de convolución de un filtro de 3×3 sobre una matriz de entrada de 7×7 .

- Stride = 1
- Sin Zero padding





2.3 Arquitectura

- La salida de cada convolución es una matriz que se obtiene por combinación lineal de la salida de la capa anterior con el filtro de esa conexión y llevada a una función de activación no lineal, habitualmente ReLU porque conserva el gradiente.

$$\vec{Y}_j = f \left(\sum_i \vec{K}_{ij} * \vec{Y}_i \right)$$

- Donde
 - \vec{Y}_j matriz de salida de la neurona j
 - \vec{Y}_i salidas de las neuronas de la capa anterior
 - \vec{K}_{ij} filtro (kernel) de la conexión
 - f función de activación ReLU
- El tamaño de la matriz de salida es
$$(N \times N) * (F \times F) = (N - F + 1) \times (N - F + 1)$$
 - N lado matriz de entrada
 - F lado del filtro

2.3 Arquitectura

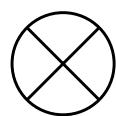


- Proceso: El filtro se va desplazando sobre toda la matriz de datos de entrada. En cada desplazamiento:
 1. Se computa la convolución de los datos de entrada con el filtro, generando un valor escalar de salida del filtro
 2. Se calcula la función de activación no lineal para ese valor
 3. Se desplaza el filtro un número de posiciones ([stride](#)) solapando parcialmente la región sobre la que operó en la convolución anterior
 4. Se vuelve a 1

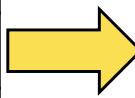
2.3 Arquitectura



		0,6	0,6			
0,6				0,6		
0,6	0,6	0,6	0,6			
0,6				0,6		



1	0	-1
2	0	-2
1	0	-1



-1,2	-0,6	0,6	1,2
-1,2	0,6	-0,6	1,2
-1,2	1,2	-1,2	1,2
-0,6	1,2	-1,2	0,6

ReLU

0	0	0,6	1,2
0	0,6	0	1,2
0	1,2	0	1,2
0	1,2	0	0,6

Imagen

Kernel

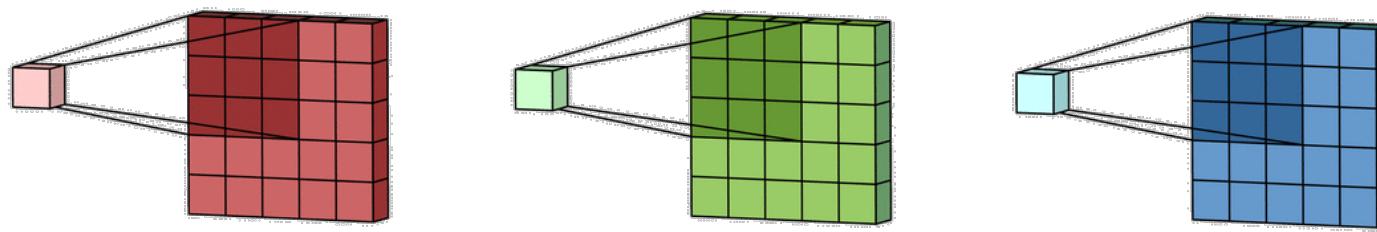
Convolución

Mapa de
Características

2.3 Arquitectura

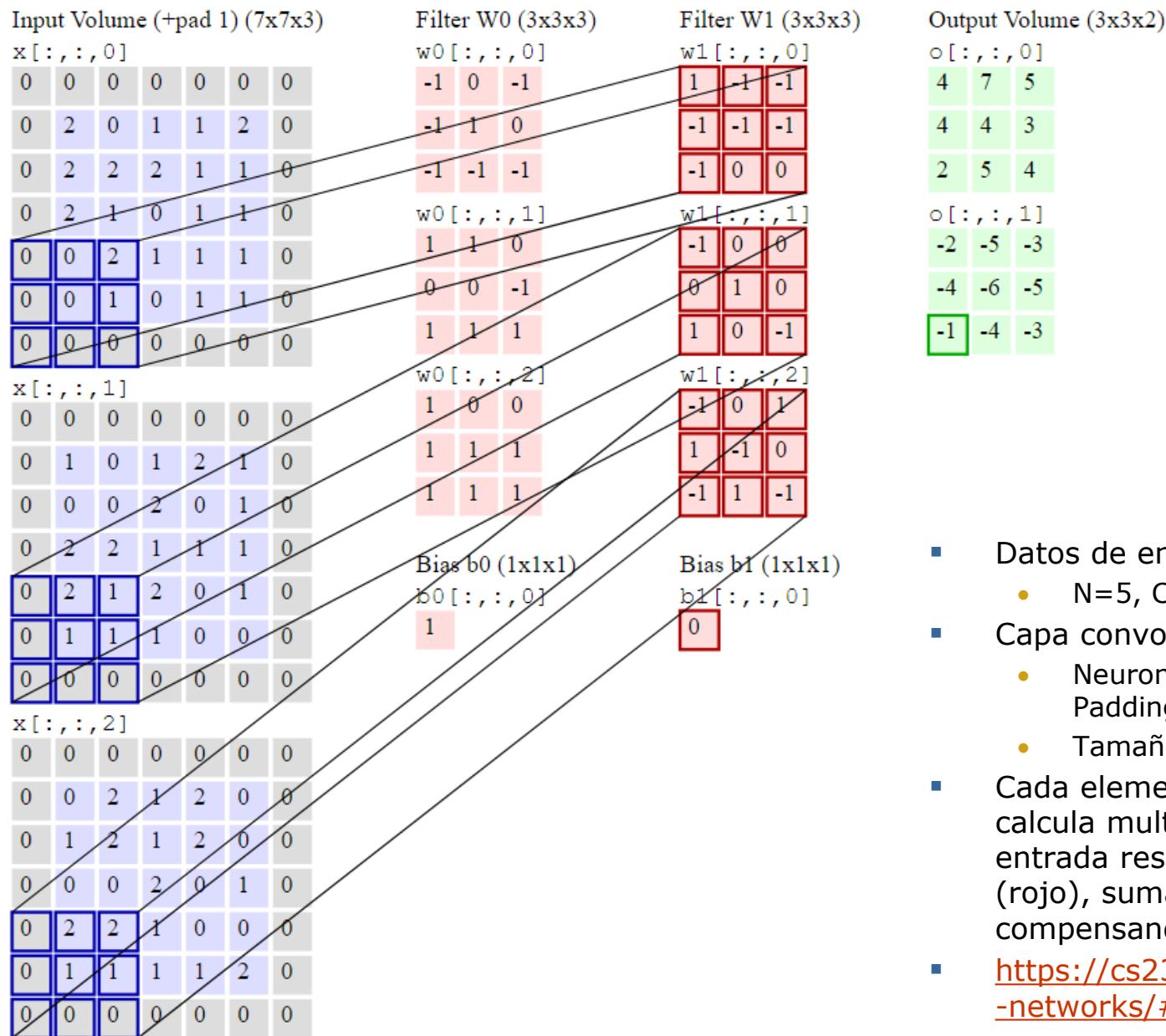


- Imágenes con un solo canal: filtro y kernel son lo mismo
- Imágenes multicanal: filtro = conjunto de kernels
 - Cada filtro implica un kernel por cada canal de entrada: el filtro de bordes aplica un kernel a cada canal por separado
 - Cada filtro produce un solo canal de salida





2.3 Arquitectura



- Datos de entrada:
 - $N=5$, Canales=3
- Capa convolucional:
 - Neuronas=2, $F=3$, Stride=2, Padding=1
 - Tamaño salida $(5-3+2)/2+1=3$
- Cada elemento de salida (verde) se calcula multiplicando por elementos la entrada resaltada (azul) con el filtro (rojo), sumándolo, y luego compensando el resultado por el bias.
- <https://cs231n.github.io/convolutional-networks/#case>



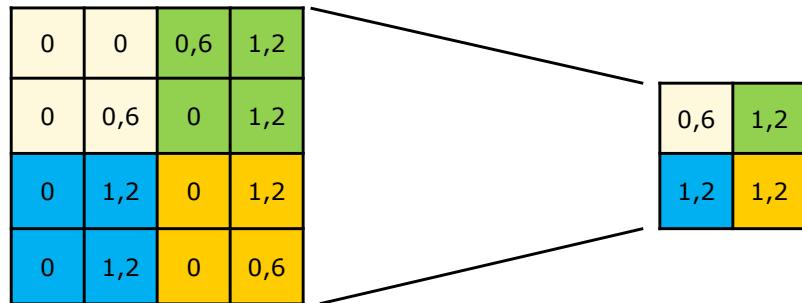
Capa de Pooling

- Capa que agrupa los valores de varias celdas contiguas del mapa de características de un filtro usando alguna función de agrupación (subsampling)
 - La salida es una nueva matriz cuyo tamaño es $(N/F \times N/F)$
 - Se reduce el tamaño de la matriz filtrada pero preservando las características más importantes que detectó cada filtro
 - Disminuye la resolución de la red poco a poco
 - Disminuye el número de pesos a entrenar en la siguiente fase de convolución
 - Pueden ser
 - *Max pooling*: selecciona el máximo de una ventana de celdas
 - *Mean pooling*: selecciona la media de una ventana de celdas

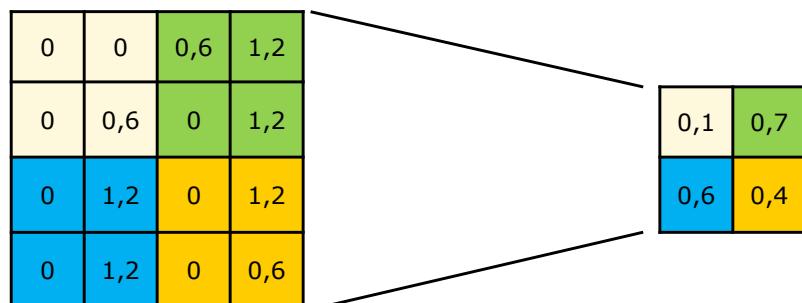


2.3 Arquitectura

- Ejemplo de Max Pooling



- Ejemplo de Mean Pooling



2.3 Arquitectura



- El resultado de una capa max pooling nos dice si una característica estaba presente en una región de la capa anterior, pero no exactamente dónde.
- Hacen *zoom out*. Permiten que las capas convolucionales posteriores trabajen en secciones más grandes de los datos, porque una zona pequeña de la capa de pooling corresponde a una zona mucho más grande antes de ella.
- También hace a la capa siguiente invariable a algunas transformaciones muy pequeñas de los datos.



2.3 Arquitectura

- Cada convolución o fase de extracción de características genera un conjunto de *feature maps* o *mapas de características*

Convolución = capa de convolución + capa de reducción

- Ejemplo de CNN con 3 convoluciones y zero padding

	Entrada	Filtros	Feature maps	Pooling	Salida
Fase 1	28x28x1	32 filtros 3x3	32 mapas 28x28	2x2	32 mapas 14x14
Fase 2	14x14x32	64 filtros 3x3x32	64 mapas 14x14	2x2	64 mapas 7x7
Fase 3	7x7x64	128 filtros 3x3x64	128 mapas 7x7	2x2	128 mapas 3x3



Capa de clasificación

- Los datos llegan a la capa de clasificación reducidos a una serie de características únicas para cada imagen de entrada.
- Esta capa *clasifica* esas características asignando una probabilidad a cada etiqueta de las aprendidas durante el entrenamiento.
- Arquitectura *fully connected* compuesta por un MLP con
 - Capa(s) oculta(s) no lineal(es): función de activación Relu
 - Capa de salida (que también es la capa OUTPUT de la CNN)
- La salida de cada neurona de las capa(s) oculta(s) se obtiene de igual manera que en el MLP

$$y_j = f \left(\sum_i w_{ij} \cdot y_i \right)$$



2.3 Arquitectura

- La última capa del MLP **y** de la red, tiene tantas neuronas como clases tiene que clasificar la CNN

Tipo de salida	Función de activación	Loss function
Binaria	Sigmoidea	Binary cross-entropy
Discreta	SoftMax	Discrete cross-entropy

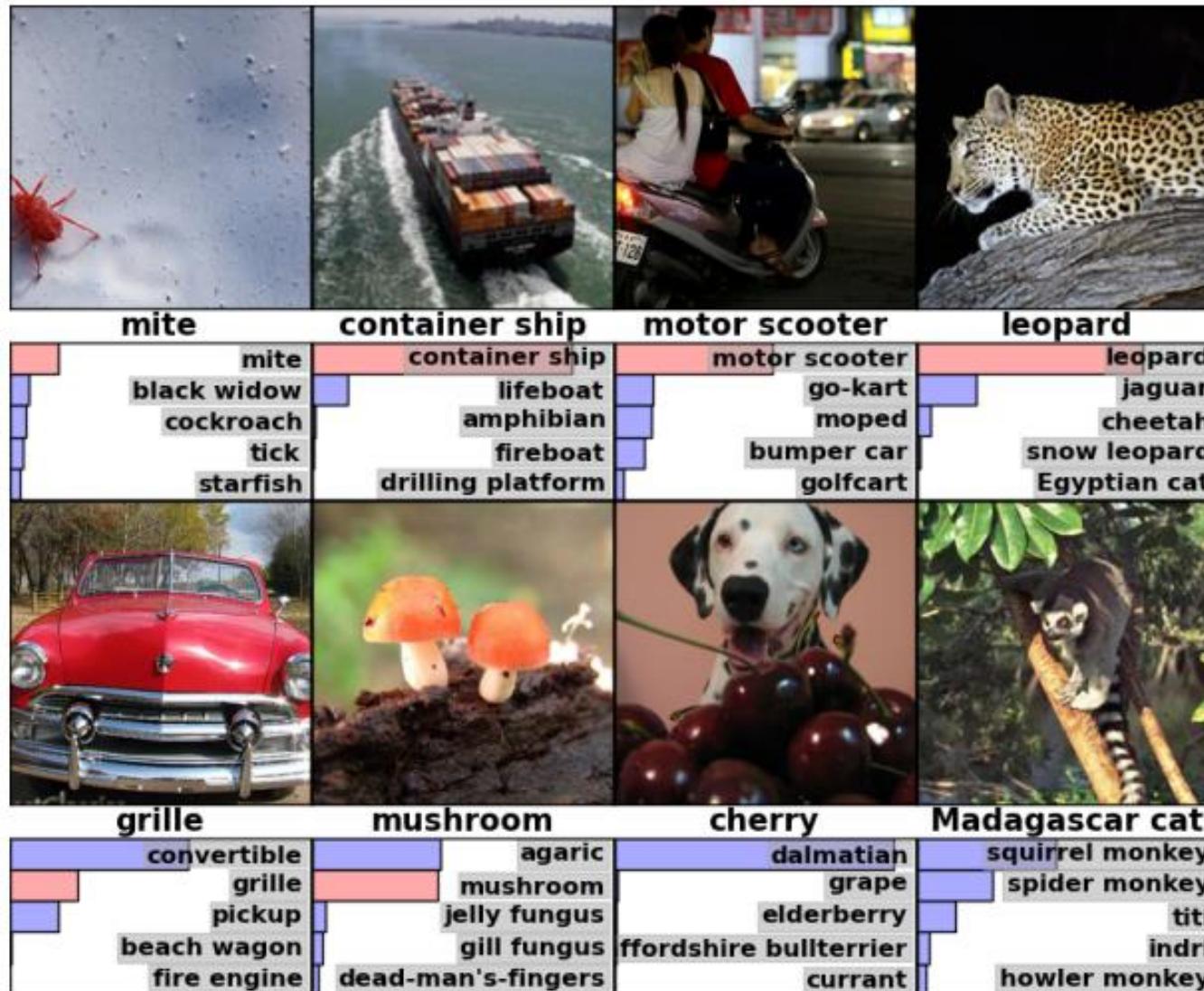


2.3 Arquitectura



SACIDA ALEXNET

- Ejemplo de salida de una CNN (Krizhevsky, 2012)





AVANZADO

<https://microscope.openai.com/models>



2.4 Hiperparámetros

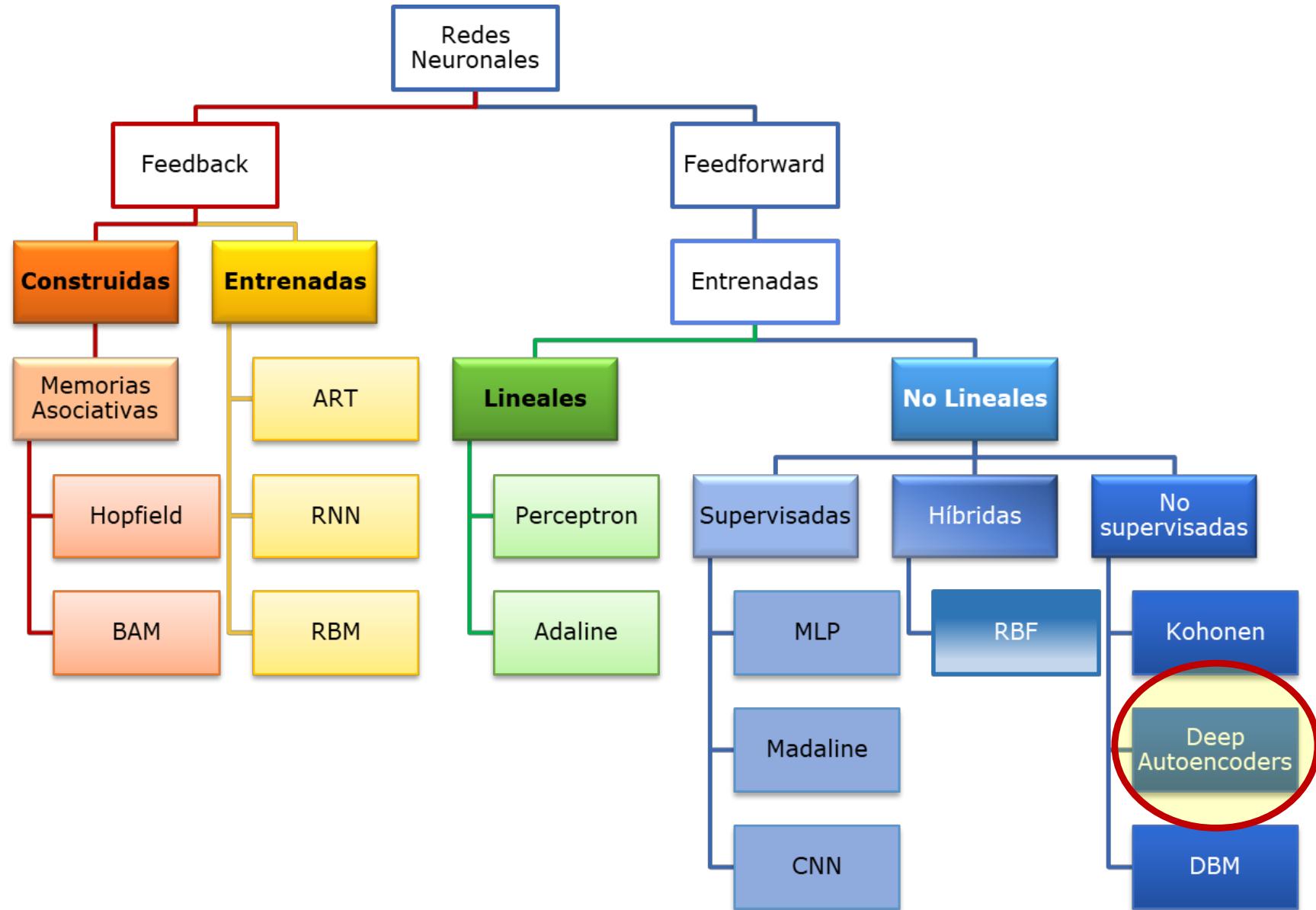
- Convolución (para cada capa):
 - Número de filtros
 - Dimensión de cada filtro (pesos/tamaño del filtro)
 - Tamaño del Stride
 - Tamaño del Padding
- Pooling
 - Tipo de pooling (max/mean)
 - Tamaño del pooling
 - Tamaño del stride
- Clasificador (fully connected)
 - Número de capas
 - Número de neuronas



- 1. Convolución**
 - 1. Operador Convolución**
 - 2. Aplicación del operador**
- 2. Convolutional Neural Networks**
 - 1. Definición**
 - 2. Por qué usar la convolución**
 - 3. Arquitectura**
 - 4. Hiperparámetros**
- 3. Deep Autoencoders**
 - 1. Definición**
 - 2. Procesamiento**
 - 3. Tipos de autoencoders**
 - 4. Aplicaciones**



3.1 Definición





3.1 Definición

- Modelo neuronal feedforward no supervisado de compresión de datos en el que las funciones de compresión y descompresión son llevadas a cabo por distintas capas que son:
 1. **Específicas** de los datos: solo pueden comprimir datos similares a los usado para entrenamiento. Diferente de otros algoritmos de compresión (como MPEG-2 Audio Layer III que contiene suposiciones sobre "sonido" en general, pero no sobre tipos específicos de sonidos).
 2. **Con pérdida**: las salidas descomprimidas se degradarán en comparación con las entradas originales (similar a la compresión MP3 o JPEG).
 3. **Aprenden automáticamente** a partir de ejemplos. El aprendizaje es **auto-supervisado** (una subclase de no supervisado), ya que la entrada y la salida es la misma (y por lo tanto no requiere pares entrada/salida)



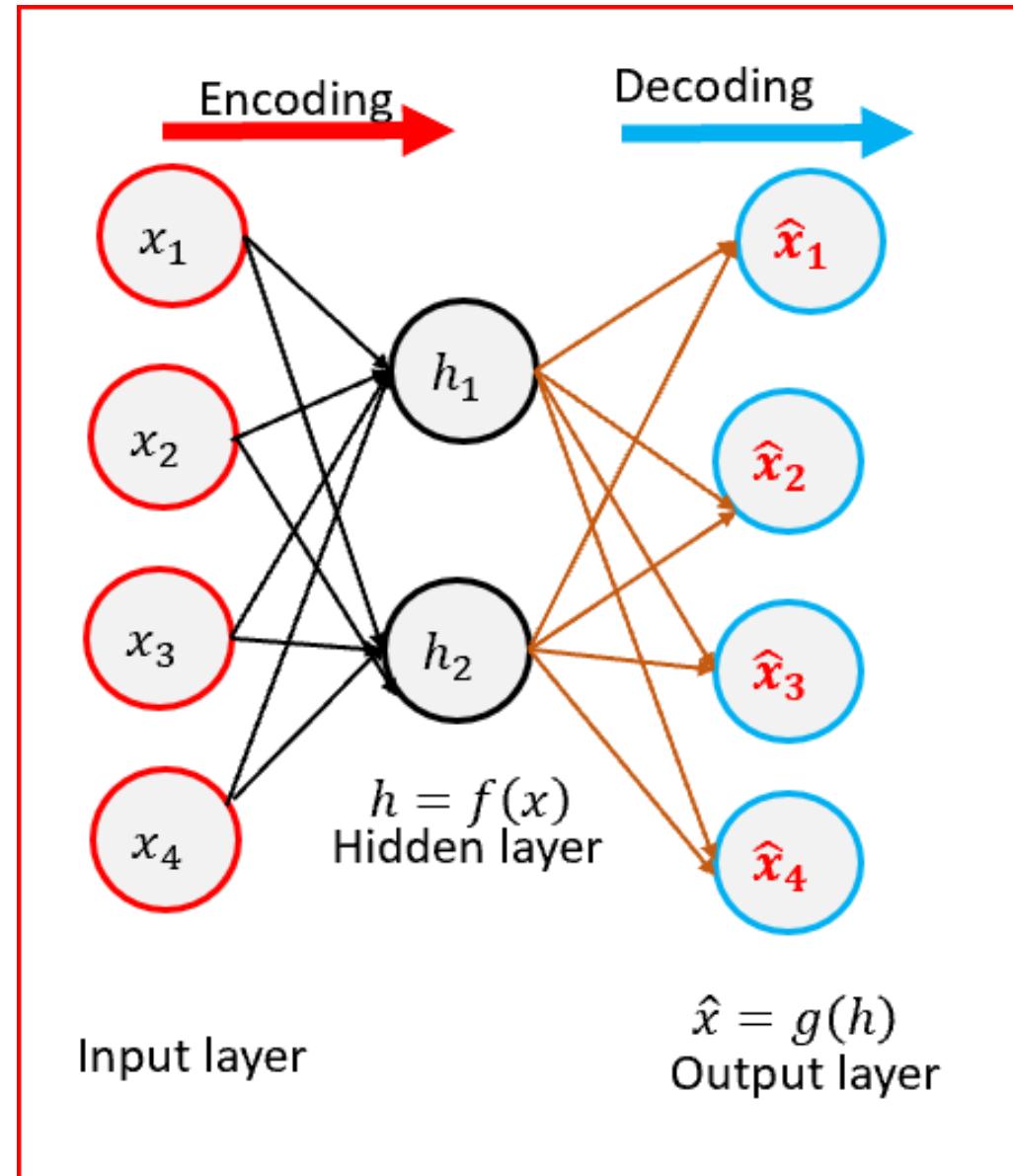
3.1 Definición

- El algoritmo del autoencoder necesita:
 - Una *función codificadora*: Codifica un input \vec{x} mediante una función $h = f(\vec{x})$
 - Una *función decodificadora*: Decodificar los valores de $f(\vec{x})$ utilizando una función $\hat{x} = g(f(\vec{x}))$, creando así un output idéntico al input original.
 - Una *función de distancia (loss function)* que mide la cantidad de información perdida entre la representación comprimida y la representación descomprimida. El objetivo es minimizar el error de reconstrucción entre el input \vec{x} y el output \hat{x} .
- **f** y **g** se implementan como capas de neuronas cuya función de activación es diferenciable respecto a la función de distancia de forma que los parámetros de codificación y decodificación (los pesos) se pueden optimizar para minimizar el *loss* utilizando *Stochastic Gradient Descent + backpropagation*.



3.1 Definición

- Arquitectura
 - **Capa de entrada:** No hace procesamiento
 - **Capa oculta:** Realiza la codificación
 - **Capa de salida:** Realiza la decodificación
- Las capas de codificación y decodificación pueden ser convolucionales, MLP o cualquier otra arquitectura diferenciable y no lineal.





3.2 Procesamiento

- Se procesa un input, del que se van a extraer características (features):
 - Se codifica el input (capa de entrada → capa oculta). Se obtiene una codificación de la entrada
 - Se decodifica el valor codificado (capa oculta → capa de salida). Se obtiene una reconstrucción de la entrada.
- Se calcula el valor de *loss*, comparando el input y el output (error de reconstrucción)
- Se minimiza el error de reconstrucción usando backpropagation. Los pesos se actualizan en función de cuán responsables han sido del error.

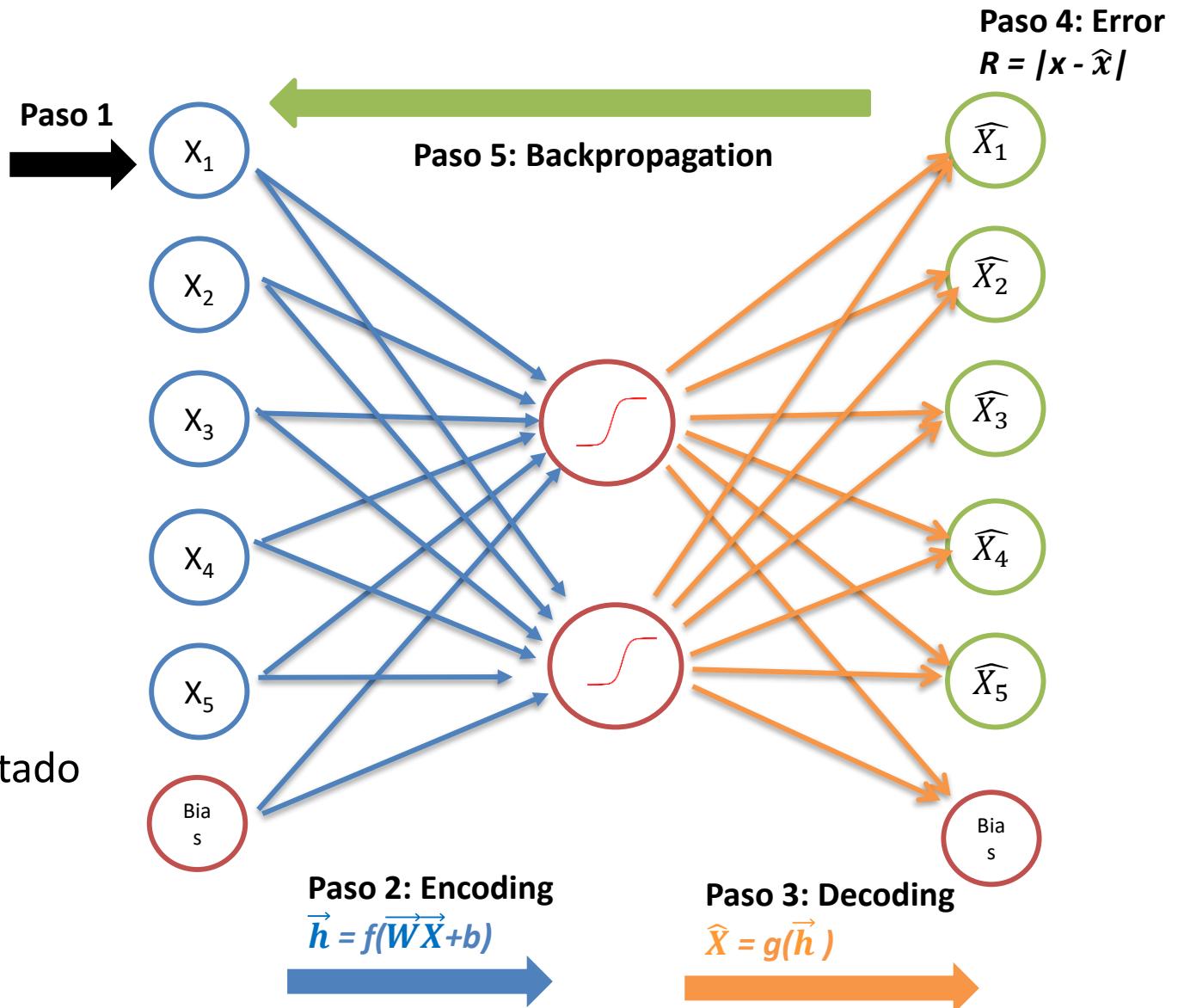
3.2 Procesamiento



	Canción 1	Canción 2	Canción 3	Canción 4	Canción 5
Cliente 1	0	0	1	1	0
Cliente 2	1	0	0	1	0
Cliente 3	1	0	1	0	1
Cliente 4	0	0	0	1	0

Base de datos de canciones y usuarios a los que les han gustado o no las canciones:

- **1** la canción ha gustado
- **0** la canción no ha gustado



3.2 Procesamiento



1. Se toma como input la primera fila del Dataset \vec{X} .
2. Se codifica en otro vector \vec{h} con menor dimensionalidad que el del input (en este caso, 2). Como los valores del vector de entrada varían entre 0 y 1 se utilizará la función sigmoide para la codificación del vector.
 1. \vec{X} es el vector de entrada
 2. \vec{W} es el vector de pesos aplicados al input
 3. b es el término de sesgo, siendo éste el output de la red neuronal cuando tiene "zero input"
$$\vec{h} = \text{sigmoide}(\vec{W}\vec{X} + b)$$
3. Se decodifica \vec{h} para recrear el input, con la misma dimensión que el vector de entrada.

$$\hat{\vec{X}} = g(\vec{h})$$



3.2 Procesamiento

4. Se calcula el *error de reconstrucción* (R), diferencia entre el input (\vec{X}) y el output (\hat{X})

$$R = |\vec{X} - \hat{X}|$$
$$R = |\vec{X} - g(f(\vec{X}))|$$

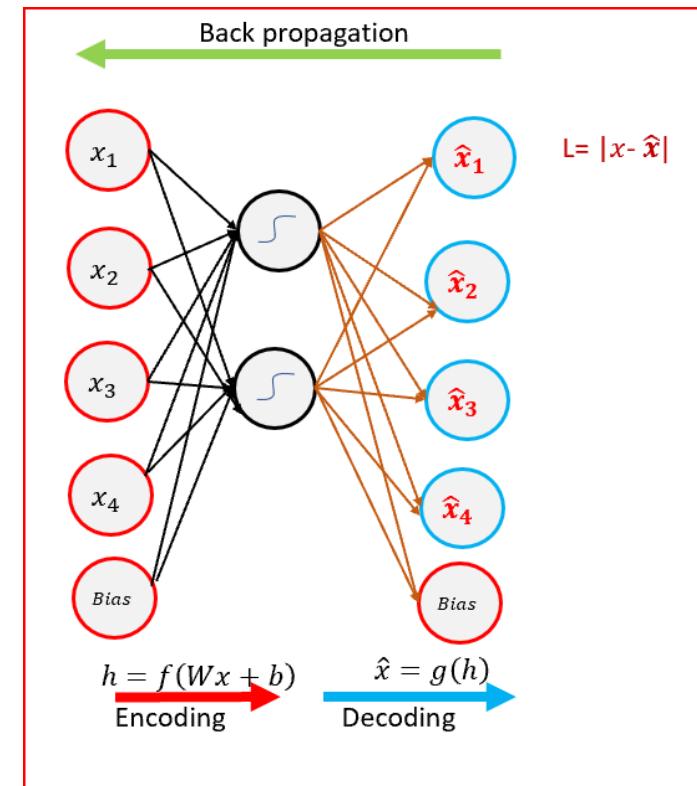
5. Se propaga el error hacia atrás con *backpropagation*. Los pesos se actualizan con SGD (*Stochastic Gradient Descent*) en función de cuánto han sido responsables por el error.
6. Se repiten todos los pasos del 1 al 5 por cada uno de los vectores de entrada disponibles hasta alcanzar el mínimo error posible.



3.3 Tipos de Autoencoders

Undercomplete Autoencoders

- Sirven para identificar y capturar las características más importantes en los datos usando una capa oculta menor que la capa de entrada.

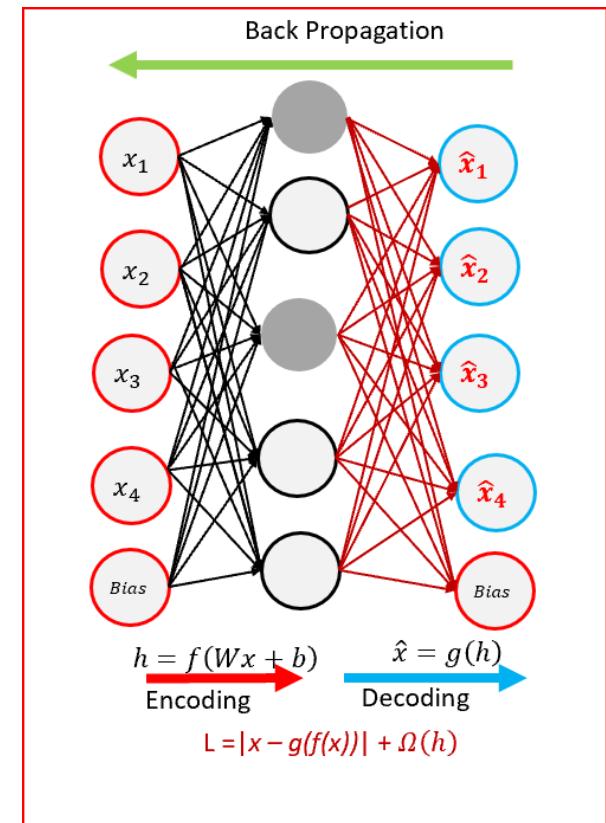




3.3 Tipos de Autoencoders

Sparse Autoencoders

- La capa oculta es mayor que la entrada. Se obtienen las características más importantes, pero evita que la salida de la red copie la entrada recibida.
- La decodificación toma únicamente los valores más altos de activación de las neuronas de la capa oculta y ponen a cero la salida de las demás neuronas.
- De esta forma se fuerza que no todas las neuronas sean activadas al mismo tiempo, aunque en cada nueva entrada se activarán neuronas distintas.

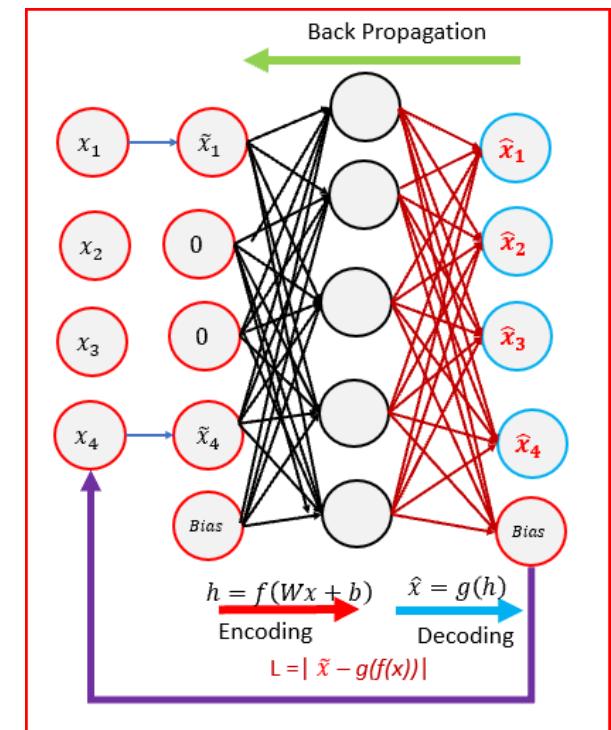




3.3 Tipos de Autoencoders

Denoising Autoencoders

- Añaden ruido de forma intencionada a los datos antes de ser introducidos en la red. La red aprende las características más importantes de los datos en lugar de copiarlos en su salida.
- La red deberá eliminar este ruido y generar una salida que será comparada con el dato de entrada antes de haberle añadido el ruido, minimizando la función de *loss*.

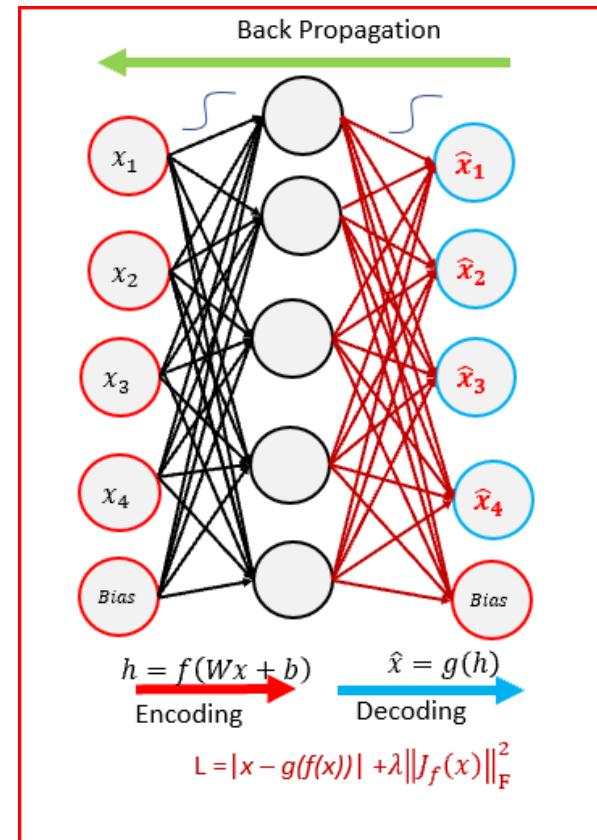




3.3 Tipos de Autoencoders

Contractive Autoencoders

- Tienen como objetivo obtener una representación robusta del aprendizaje para que sea menos sensible a las pequeñas variaciones en los datos.
- Son una mejor alternativa que los Denoising autoencoders cuando se intenta extraer las características de los datos de una forma más fiable, concisa y útil.





3.3 Tipos de Autoencoders

Stacked Autoencoders

- Modelo Deep consistente en una red neuronal multicapa compuesta por múltiples Autoencoders.
- Es el equivalente a la Deep Belief Network (mientras que los autoencoders serían las RBM)
- Cada nueva capa oculta compactará más los datos que la capa anterior, hasta tener una versión "concentrada" de la entrada.



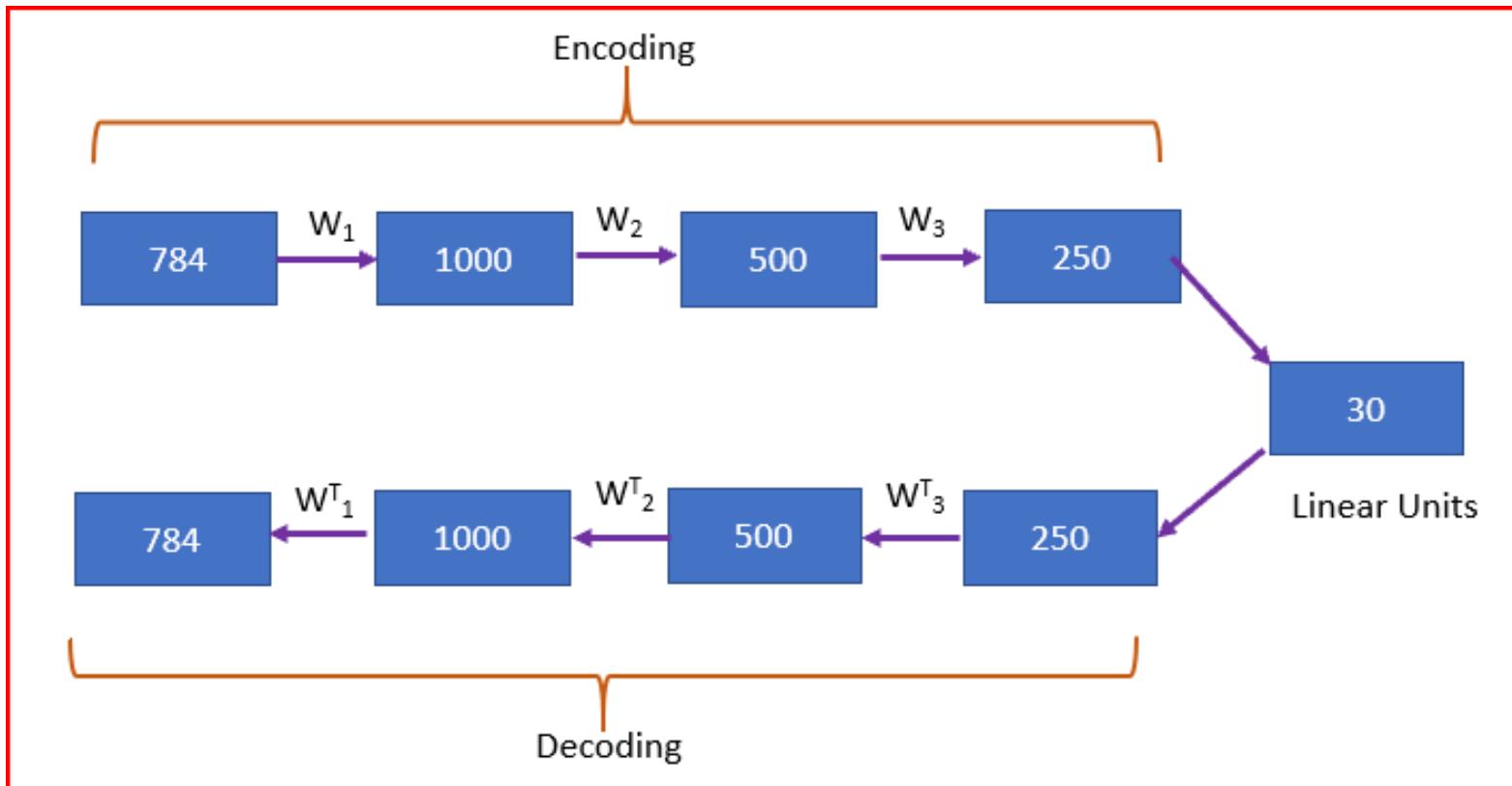
3.3 Tipos de Autoencoders

Deep Autoencoders

- Dos redes neuronales simétricas y conectadas, donde una se encarga de la codificación y la otra de la decodificación.
 - La red de *encoding*:
 - Reduce la dimensionalidad de la entrada manteniendo las características más importantes, hasta concentrar en el bottleneck las características esenciales de la misma.
 - La red de *decoding*:
 - Aumenta la dimensionalidad de esta versión concentrada recuperando el tamaño original a partir únicamente de sus características.
- Por lo general, cuentan con 4 o 5 capas por cada red, tratándose de modelos profundos.

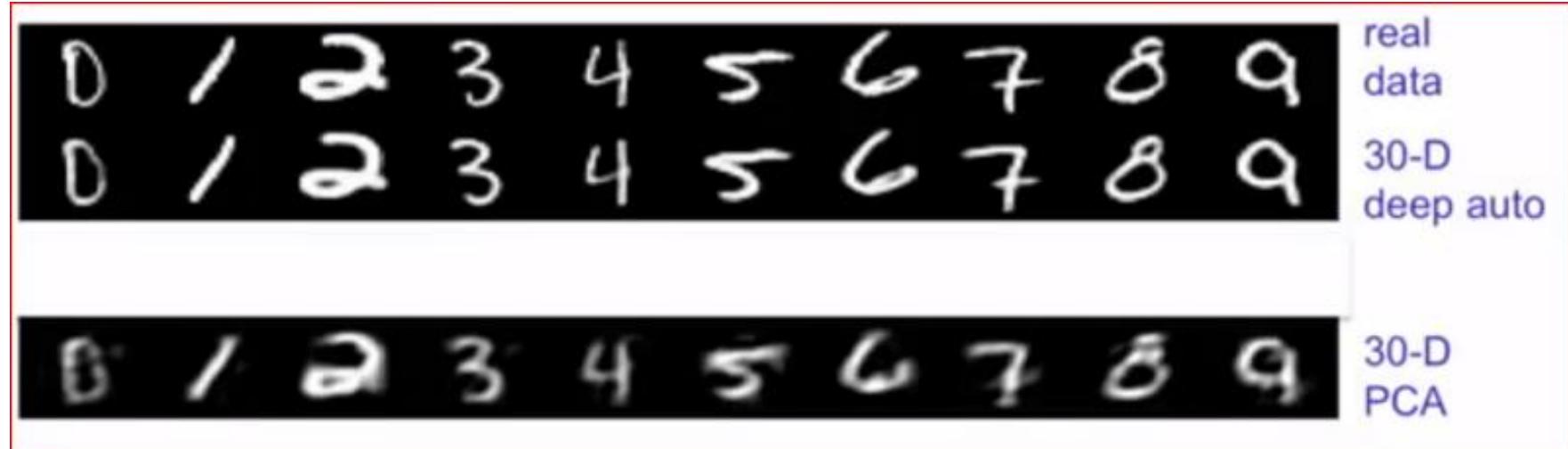


3.3 Tipos de Autoencoders





3.3 Tipos de Autoencoders





3.4 Aplicaciones

- Aplicaciones
 - Reducción de la dimensionalidad para **datos no lineales**. Codifica la entrada en la capa oculta a una dimensión más pequeña en comparación con la dimensión de entrada. La capa oculta se decodifica más tarde como salida con la misma dimensión que la entrada.
 - Motores de recomendación. Uso de Deep Autoencoders para comprender las preferencias del usuario para recomendar películas, libros, etc.
 - Extracción de características. En el proceso de minimizar el error de reconstrucción para reducir el error, aprende algunas de las características (**features**) importantes presentes en la entrada. La codificación genera un nuevo conjunto de características combinación de las características originales.
 - Reconocimiento de imágenes: Uso de varios codificadores apilados juntos para aprender diferentes características de una imagen.