

Francisco Serradilla García  
Dr. en Informática por el Departamento de  
Inteligencia Artificial de la UPM

1 de marzo de 2023

# Nacimiento del Deep Learning

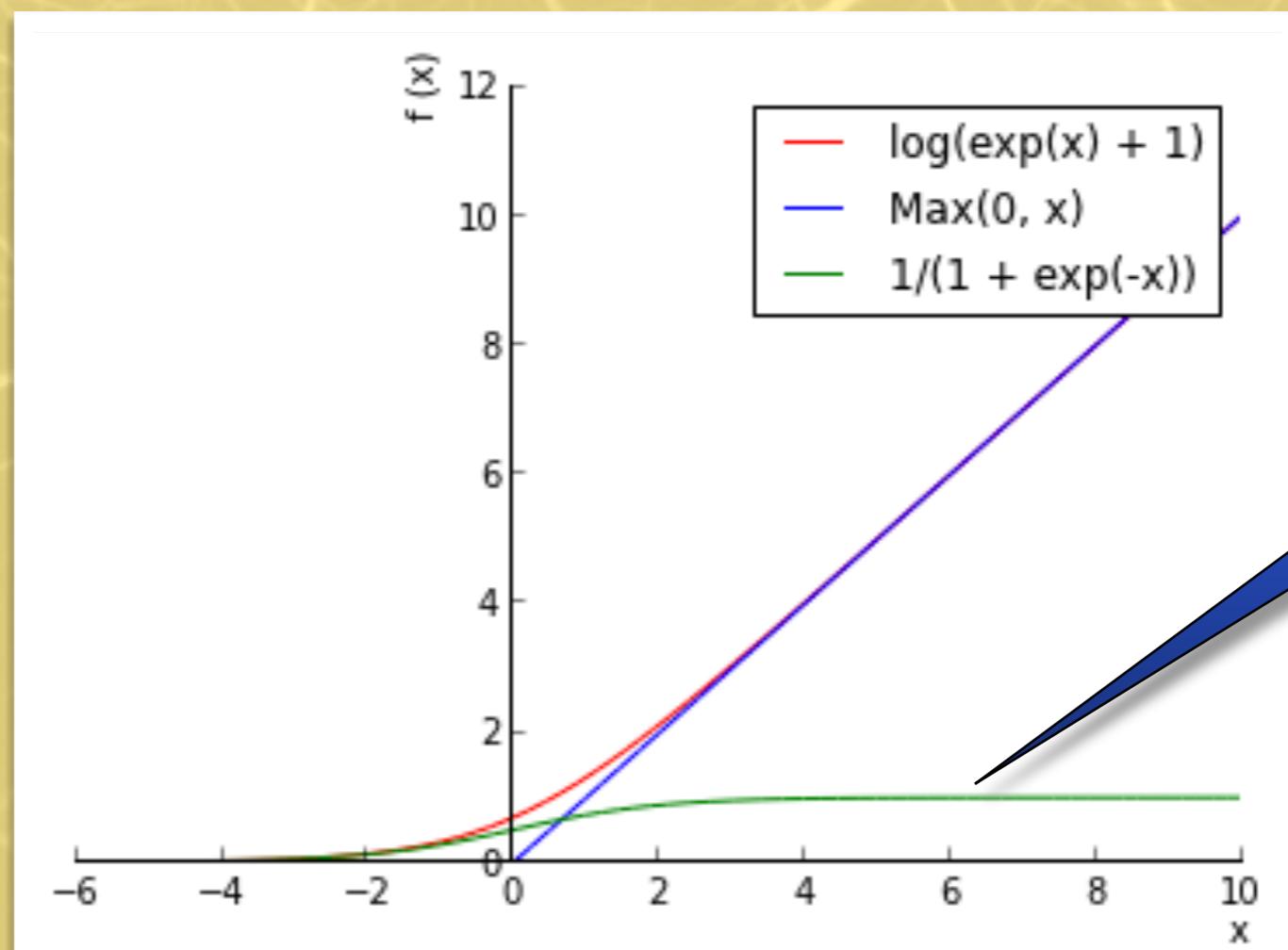
- \* En [Cybenko, 1989], usando el teorema de representación de Kolmogorov–Arnold (que resuelve el 13th Hilbert Problem), se demostró que
  - Con una capa oculta puede aproximarse cualquier función continua de N variables
  - Por ello se prestó poca atención durante muchos años a las redes con muchas capas ocultas
  - El problema del desvanecimiento del gradiente hacía muy difícil el entrenamiento
- \* No obstante, el número de parámetros necesarios (conexiones y bias) puede ser demasiado alto, y la introducción de capas adicionales conveniente
  - Aumentar el número de capas implica además que la red tenga mayor capacidad de **generalización**
- \* A partir de 2008 comienzan a aparecer modelos que apilan muchas capas ocultas, con gran éxito en diversas aplicaciones

# Novedades

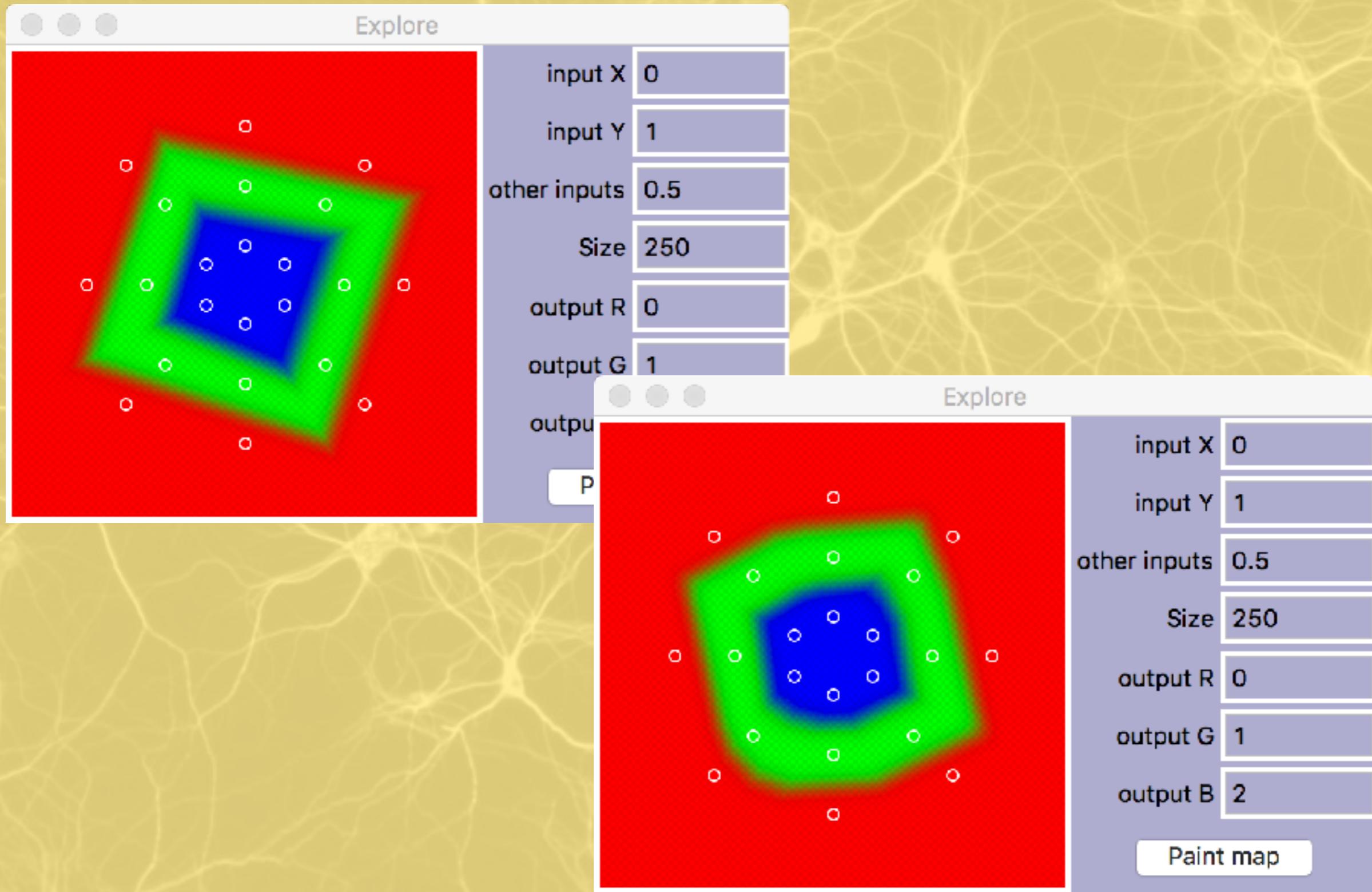
- \* Uso de muchas capas
- \* Compartición de pesos (redes de convolución)
- \* Capas de extracción de características (auto-encoders y Restricted Boltzmann Machines)
- \* Uso de neuronas ReLU en lugar de tanh o sigmoidal para evitar la saturación de las neuronas (derivada cercana a 0, o vanishing gradient problem, problema del desvanecimiento del gradiente)
- \* Uso de softmax en la capa de salida (para problemas de clasificación)
- \* Uso de la cross entropy como función de coste a minimizar, en lugar del error RMS
- \* Regularización
- \* Computación basada en GPUs
  - Librerías para python: Teano, Torch, Tensorflow
  - Utilizan la gran cantidad de núcleos de las tarjetas gráficas avanzadas (4352 CUDA cores en una NVIDIA GeForce RTX 2080 Ti) para distribuir los cálculos matriciales y aumentar la velocidad de procesamiento

# Neuronas ReLU (Glorot y Bengio, 2010)

- \* La entrada neta se calcula como siempre
- \* La función de activación es  $\max(0, \text{neta})$ , que es una aproximación de la función softplus
- \* La ventaja es que se calcula rapidísimo en la GPU



# Neuronas ReLU, regiones



# Softmax

- \* Función de activación que exagera la salida de la neurona más activa y además convierte las salidas en probabilidades (normalizando a suma = 1)

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

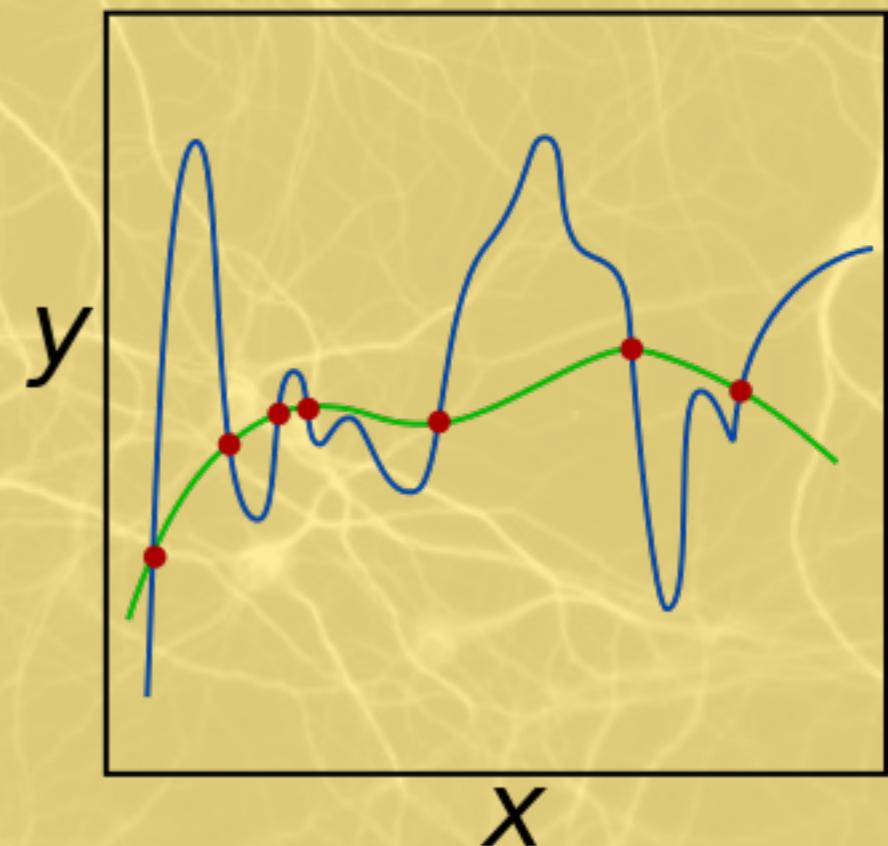
# Cross entropy

- \* Se basa en la idea de entropía de la información de Claude Shannon
- \* Estima cuántos bits hacen falta para distinguir uno entre varios sucesos
  - Si el suceso es seguro, hacen falta 0 bits
  - Si el suceso es equiprobable entre dos alternativas, hace falta 1 bit
  - Si es equiprobable entre N alternativas, hacen falta  $\log_2(N)$  bits
- \* Las salidas tienen que ser probabilidades (softmax)
- \* **Importante:** no usar cuando hay una única salida

$$H(s,d) = - \sum_i^S d_i \log(s_i)$$

# Regularización

- \* Intenta aumentar la generalización de la red (evitar overfitting) evitando que haya neuronas que se especialicen en casos concretos
  - Tradicionalmente se controlaba el overfitting evitando redes muy grandes, pero ahora utilizamos redes muy grandes
  - Métodos
    - ✖ Regularización L2
    - ✖ Regularización L1
    - ✖ Dropout



# Regularización L2

## \* L2

- La más utilizada hasta que apareció el dropout
- Añade un término a la función de coste que crece con el cuadrado de los pesos, de modo que el back propagation intentará mantener los pesos bajos además de resolver el aprendizaje
- El parámetro  $\lambda$  controla cuánta importancia damos a la regularización frente al aprendizaje, y depende del problema

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_j \left( t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^k w_i^2$$

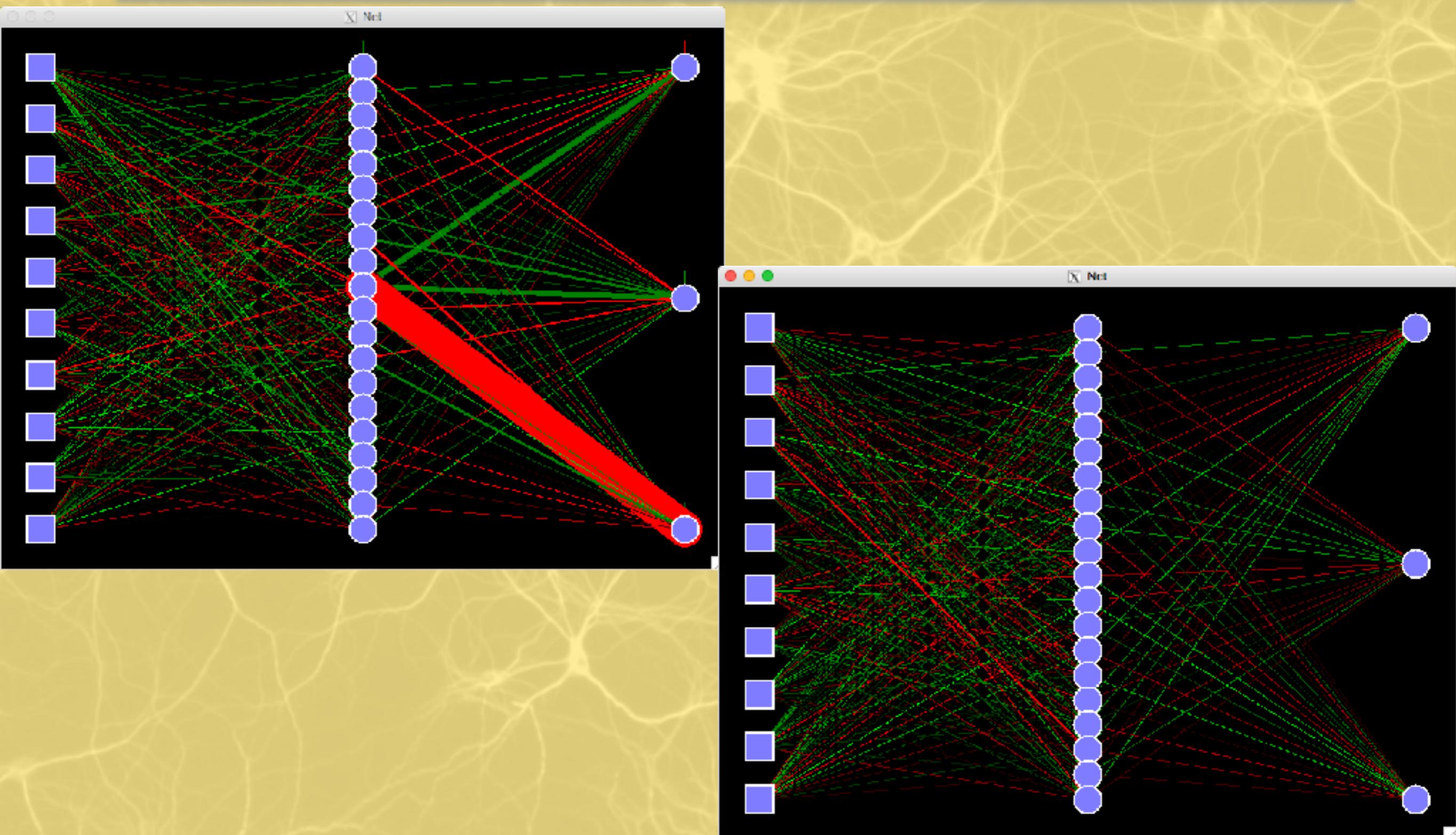
# Regularización L1

## \* L1

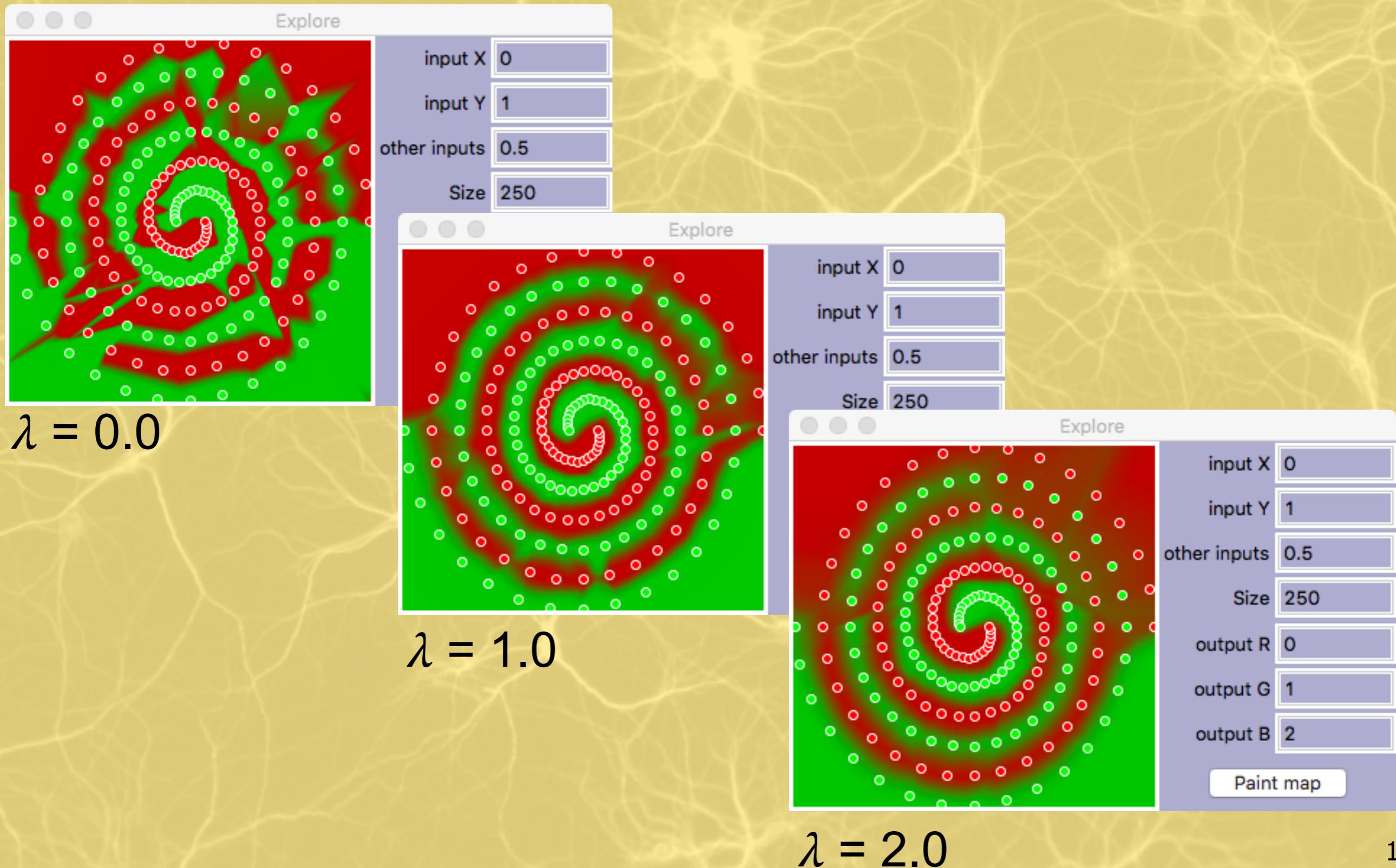
- Se usa cuando queremos detectar las entradas más relevantes
- El parámetro  $\lambda$  controla cuánta importancia damos a la regularización frente al aprendizaje, y depende del problema

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_j \left( t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^k |w_i|$$

# Efectos de la regularización



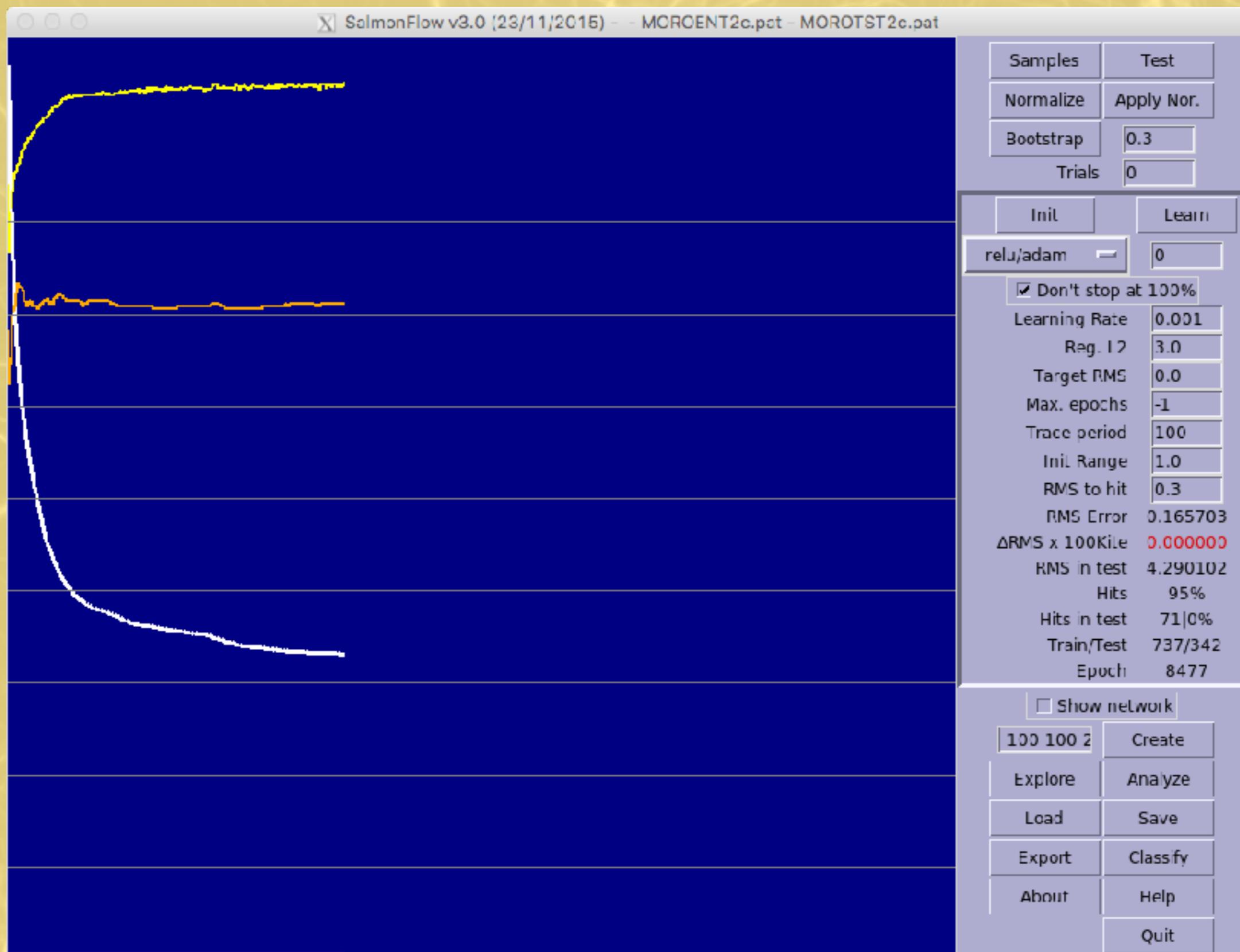
# Efectos de la regularización



# Efectos de la regularización



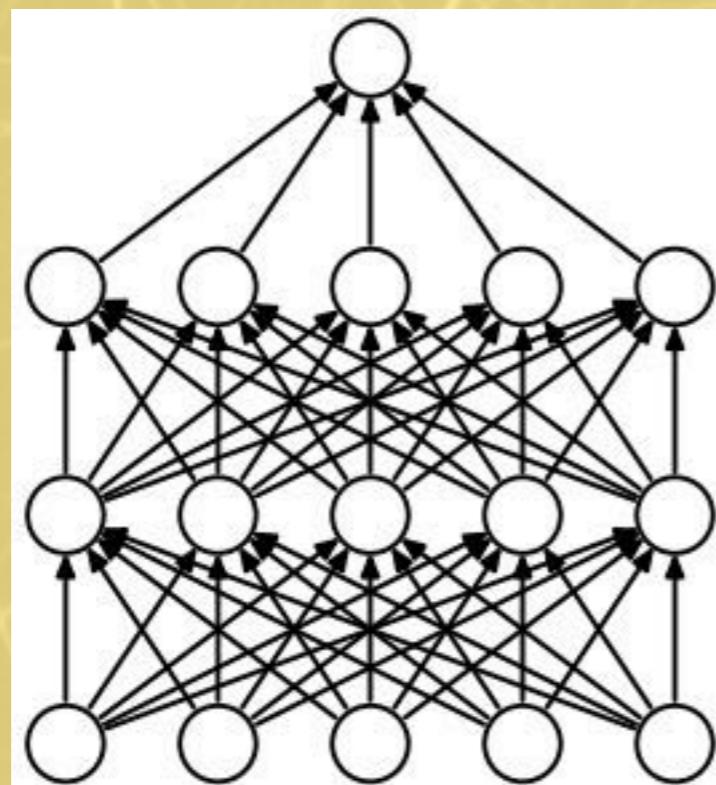
# Efectos de la regularización



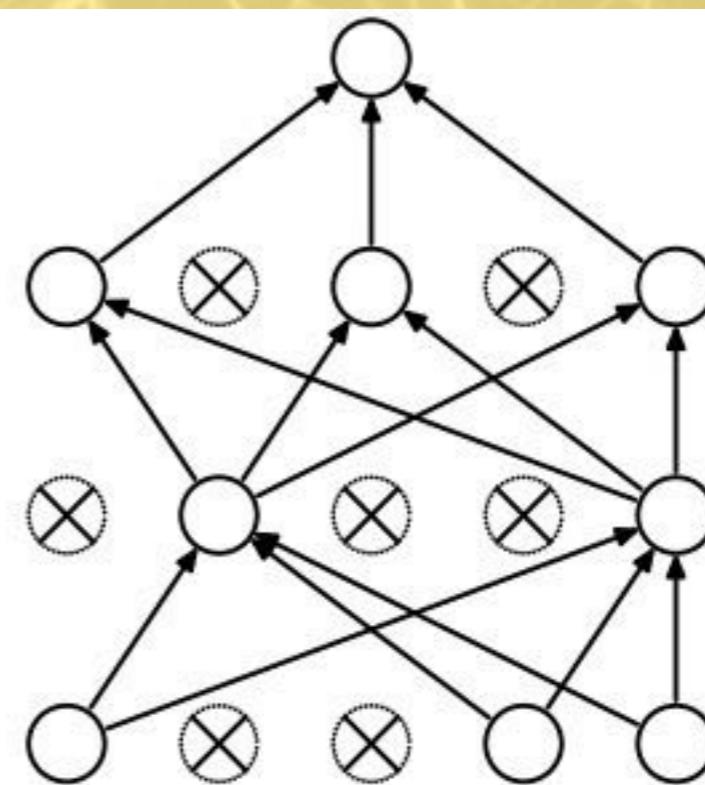
# Dropout

## \* Dropout (Hinton, 2012)

- Es una nueva técnica que se basa en desactivar aleatoriamente algunas neuronas en cada paso de entrenamiento, de modo que las conexiones tienen que ser robustas a la perdida de estas neuronas y por tanto ninguna neurona será “especial”



(a) Standard Neural Net



(b) After applying dropout.

# Dropout

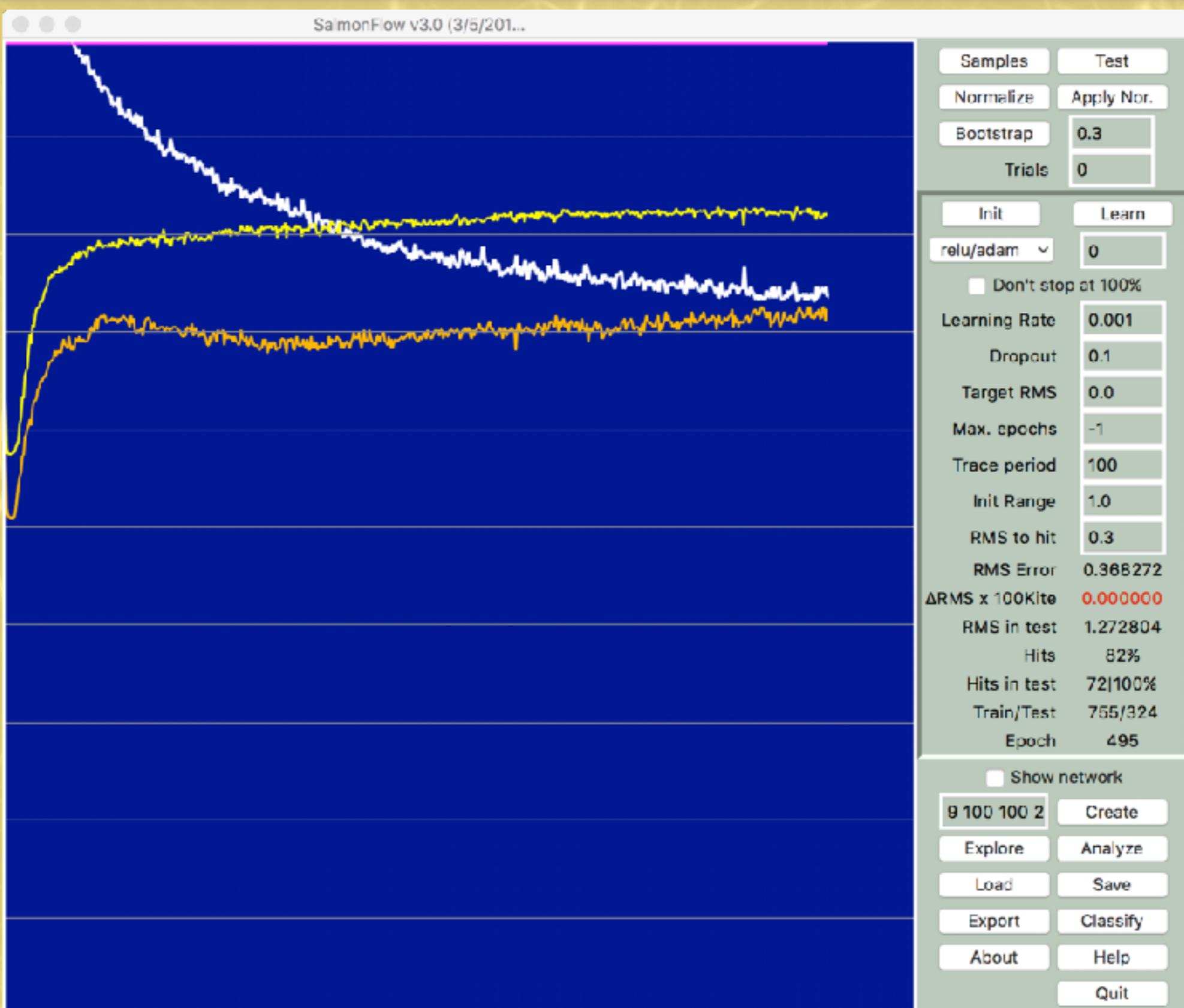
## \* ¿Cuántas neuronas desactivar?

- Está regido por la tasa de dropout (dropout rate), que es un ratio de 0 a 1 en el que 0 significa “sin dropout” y 1 sería eliminar todas (lo cual no tiene sentido)
- El dropout rate es un hiperparámetro más que depende del problema; es algo así como que proporción de neuronas “sobran” en la red para resolver el problema
- Lo normal es empezar con un ratio de 0.5, que es el recomendado por muchos autores
- Si la red está sobredimensionada podemos requerir dropouts muy altos, incluso 0.8 o 0.9

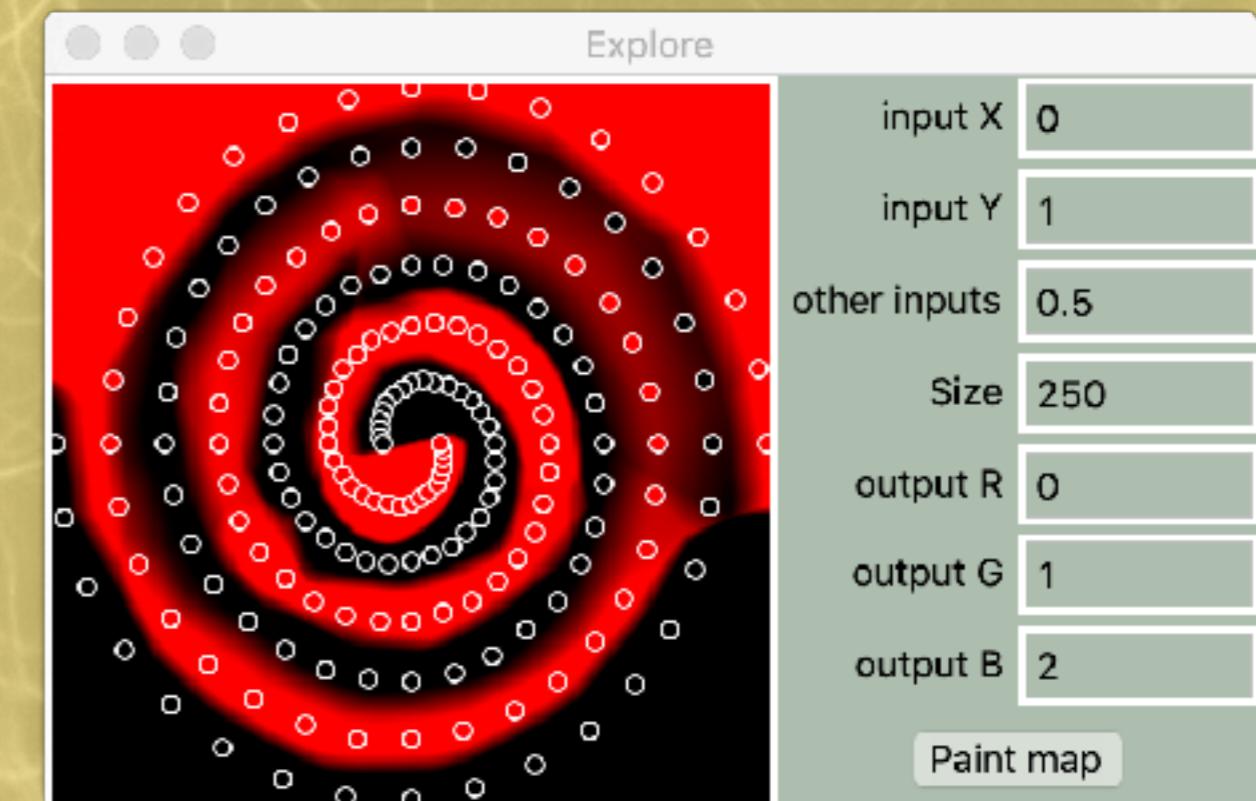
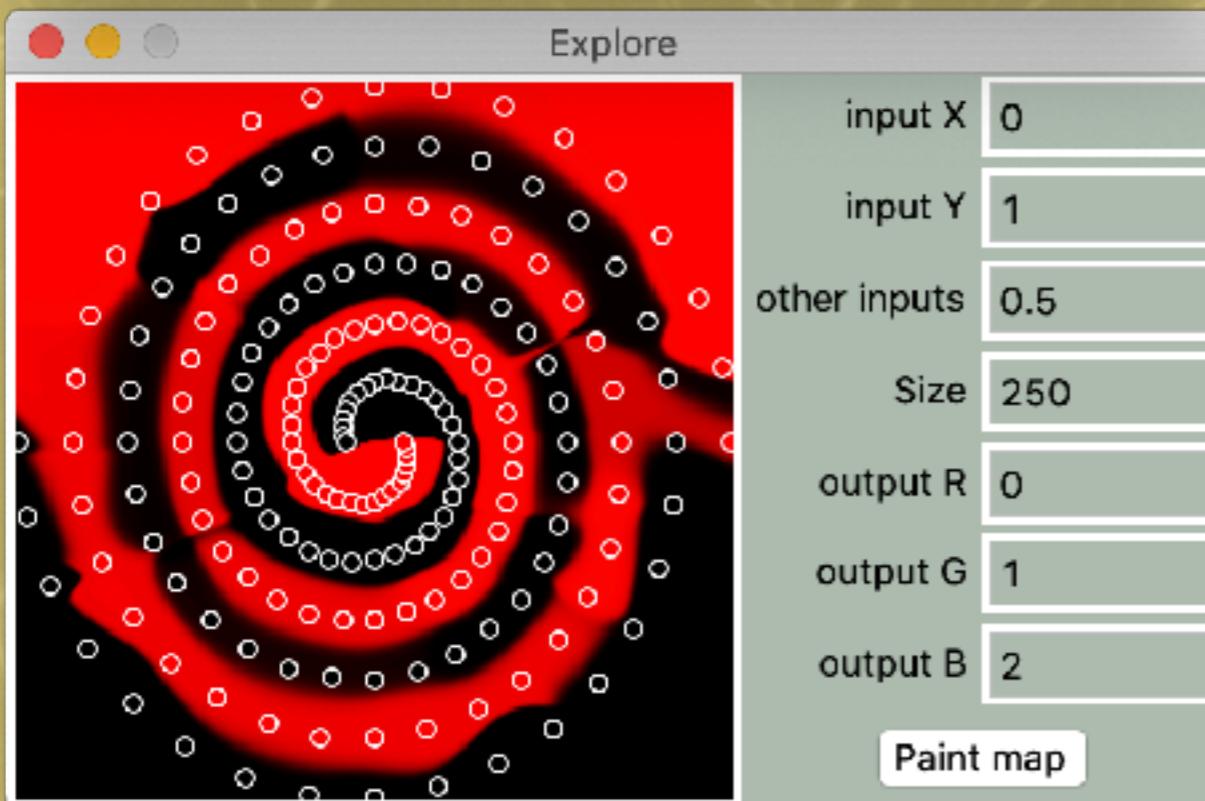
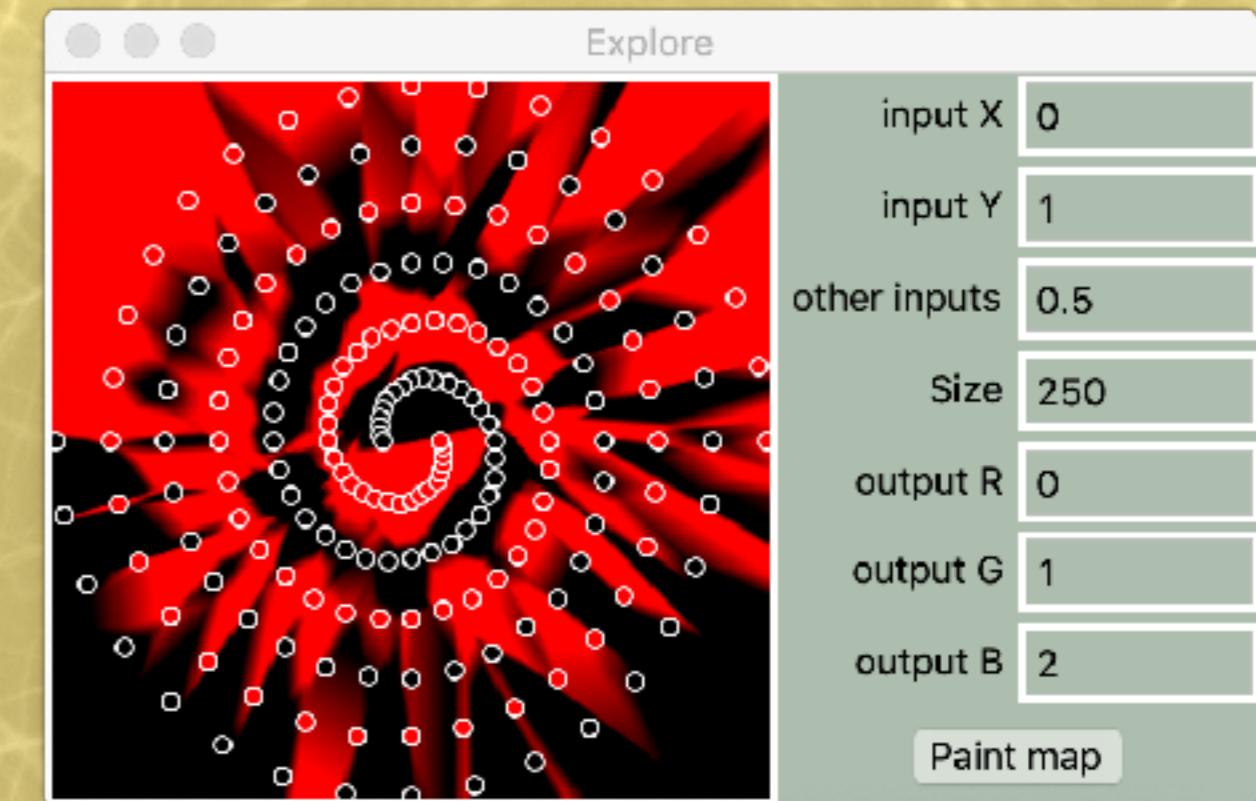
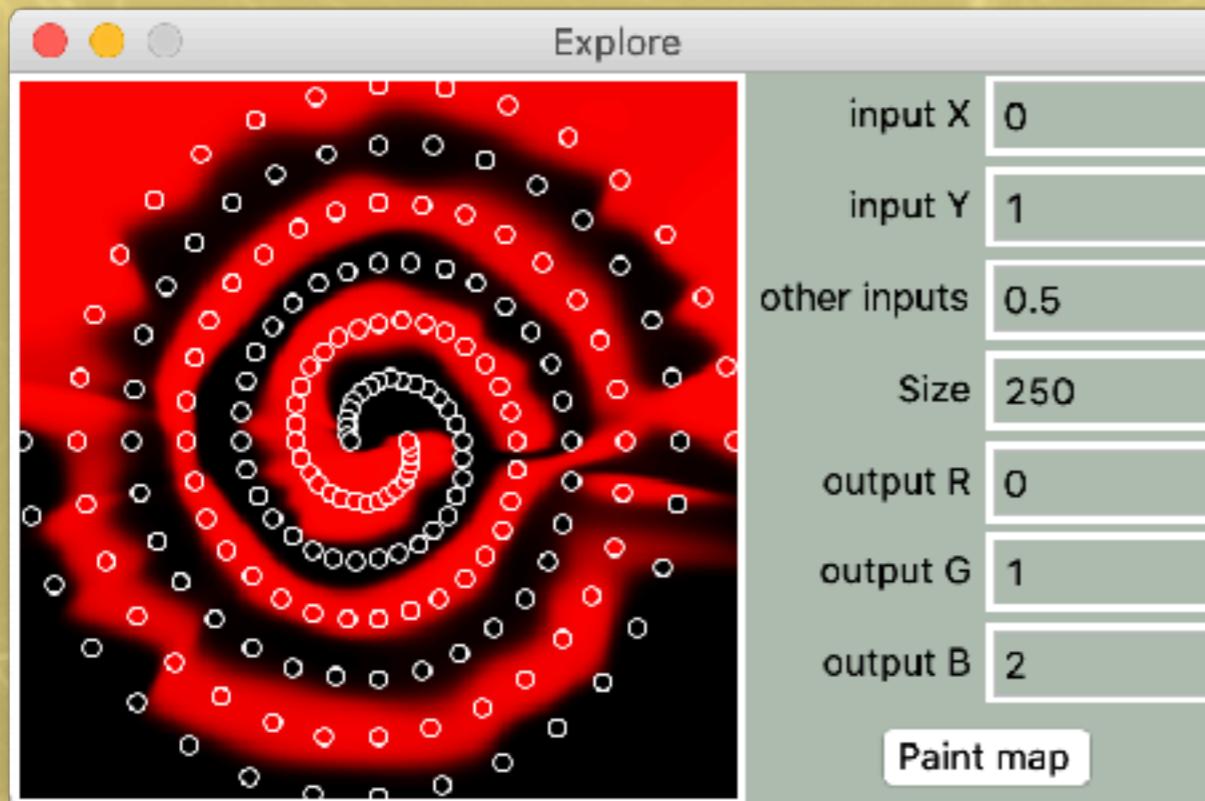
# Efectos del dropout



# Efectos del dropout



# Efectos del dropout (sigmoidal y relu)



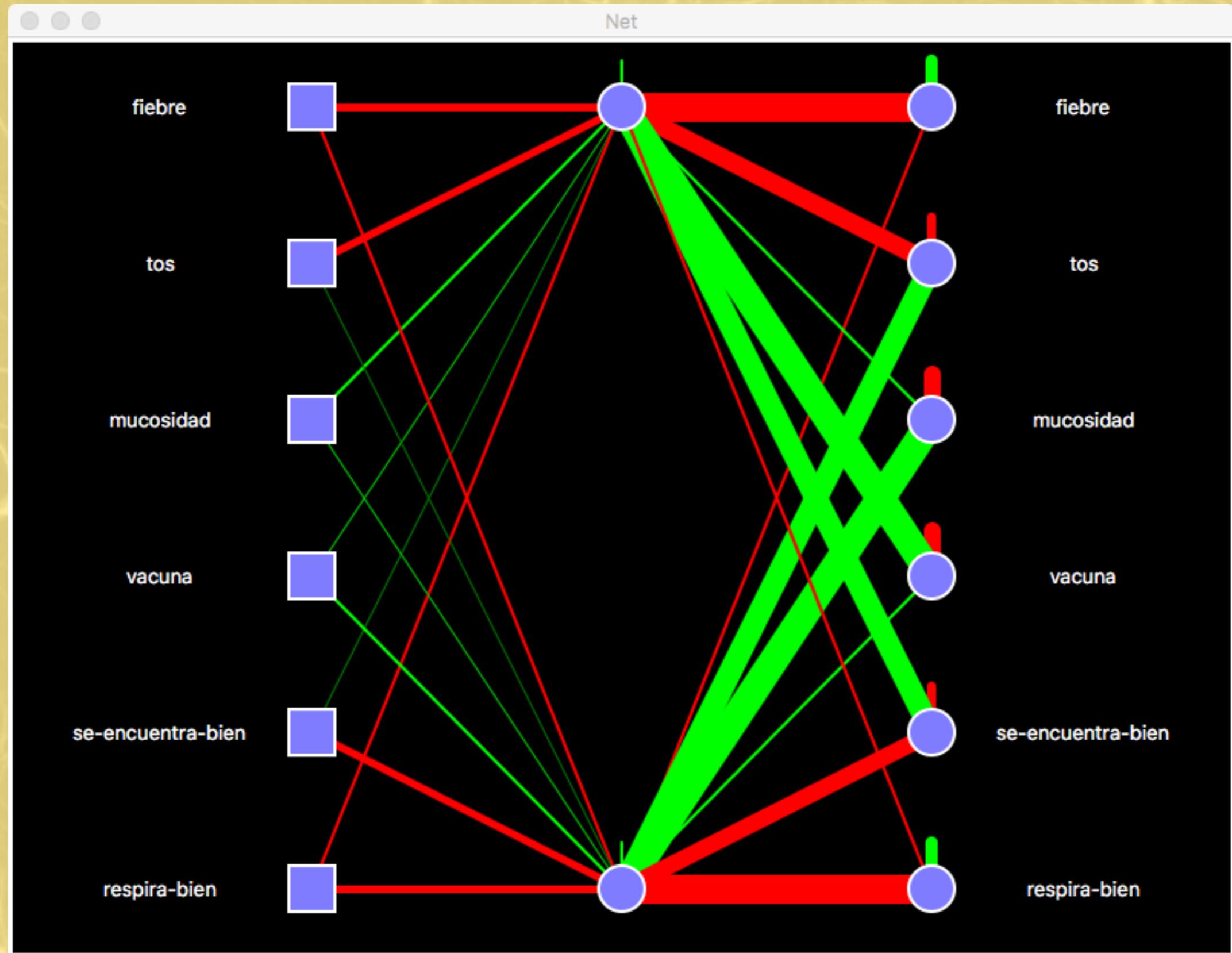
# Nuevas arquitecturas

- \* Deep auto-encoders
- \* Convolutional Neural Networks (CNN)
- \* Transposed CNN
- \* Long Short Term Memories (redes LSTM)
- \* Generative Adversarial Networks (GAN)
- \* Transformers
- \* Cada vez más se ven como capas que pueden combinarse

# Auto-encoders

- \* Se basan en un ejemplo clásico de uso de perceptrón multicapa para compresión de información
  - Se fuerza a un perceptrón con una capa oculta a aprender la misma salida que la entrada, pero con menos neuronas en la capa oculta que las entradas (por ejemplo arquitectura 8-3-8)
  - Con ello estamos obligando al perceptrón a que “comprima” la información en la capa oculta, extrayendo con ello las características relevantes en los ejemplos
  - A esta capa oculta se le denomina “espacio latente”
  - Nótese que es un problema esencialmente no supervisado, porque en las salidas ponemos las mismas entradas para cada ejemplo

# Ejemplo de autoencoder



# Deep Auto-encoders

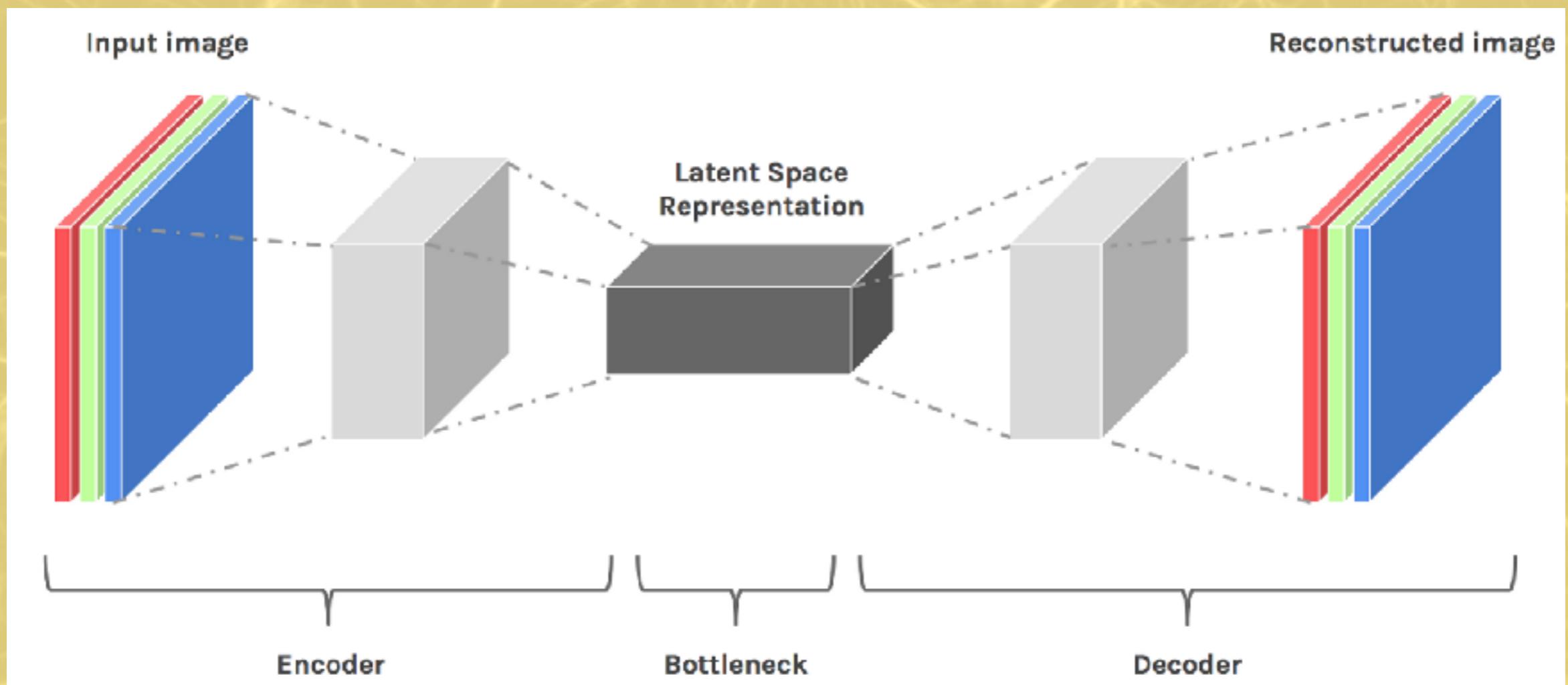
- \* Los auto-encoders pueden constar de muchas capas encoger y muchas decoder
  - Las capas pueden ser de cualquier tipo, pero si hay capas convolucionales su contrapartida son las convoluciones transpuestas
- \* Al final se añaden una o varias capas full-conected normales
- \* Una red profunda basada en auto-encoders se entrena como sigue
  1. **Pre-training:** el autoencoder se entrena poniendo a la salida la misma información que a la entrada; luego nos quedamos sólo con la parte encoder.
  2. **Training:** una vez ajustado el autoencoder se usa back propagation para entrenar las capas full-connected, y hacer fine-tunning de la parte encoder

# Autoencoders y sparse data

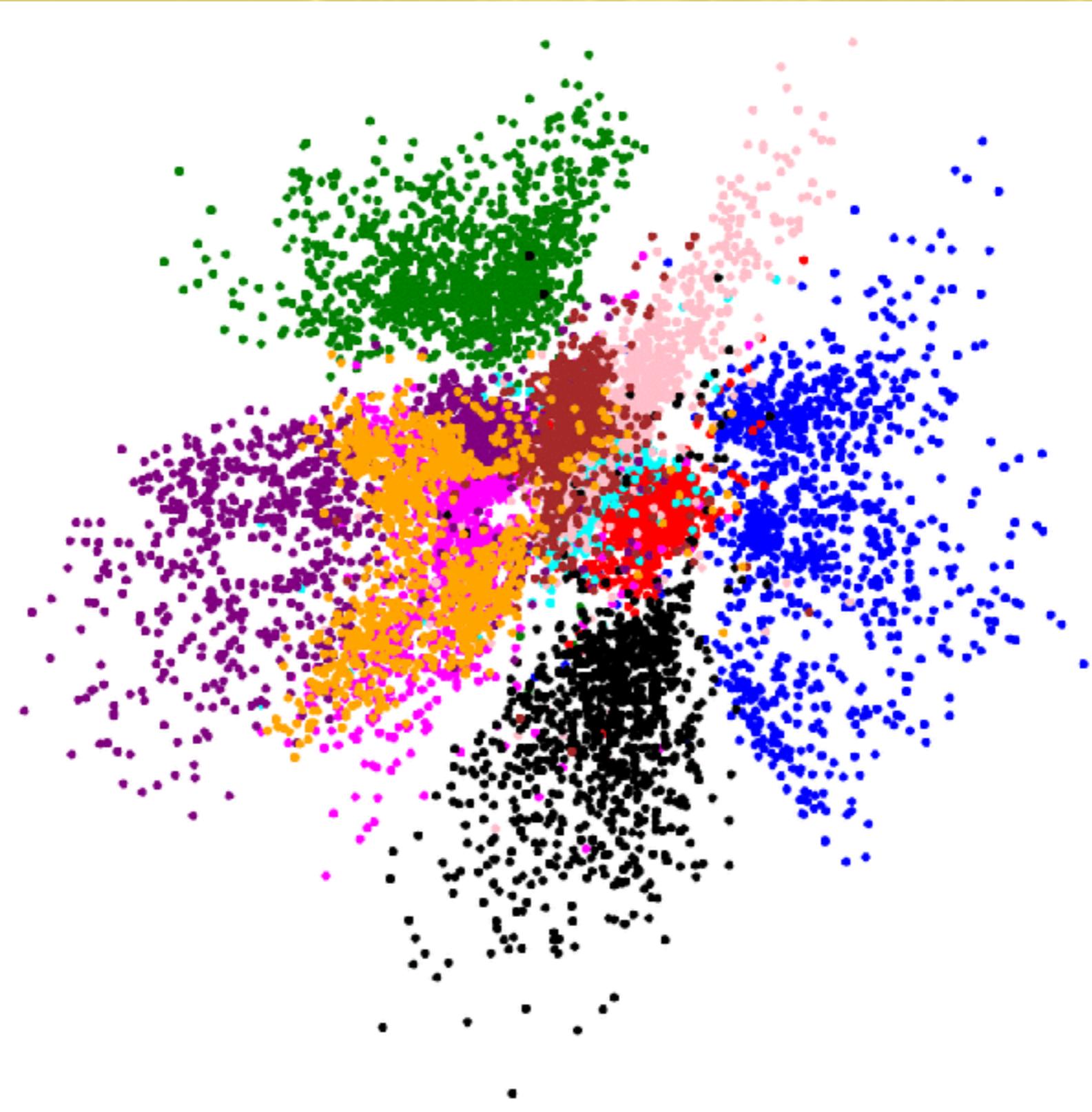
- \* Imaginemos una red que tiene como entradas palabras que aparecen en un texto
  - Una entrada por cada posible palabra
  - Dado un texto, habrá mucho ceros a la entrada (todas las palabras que NO están)
  - Parece natural pensar que algunas palabras se parecen entre sí, es decir están más cerca de otras en un espacio “semántico”
- \* Podemos utilizar un autoencoder N-M-N con M mucho menor que N
  - Una vez entrenado, la capa oculta representaría el contenido semántico latente de cada palabra
- \* Con esa representación podríamos entrenar un nuevo perceptrón para que aprendiera, por ejemplo, si el mensaje es positivo o negativo

# Latent space

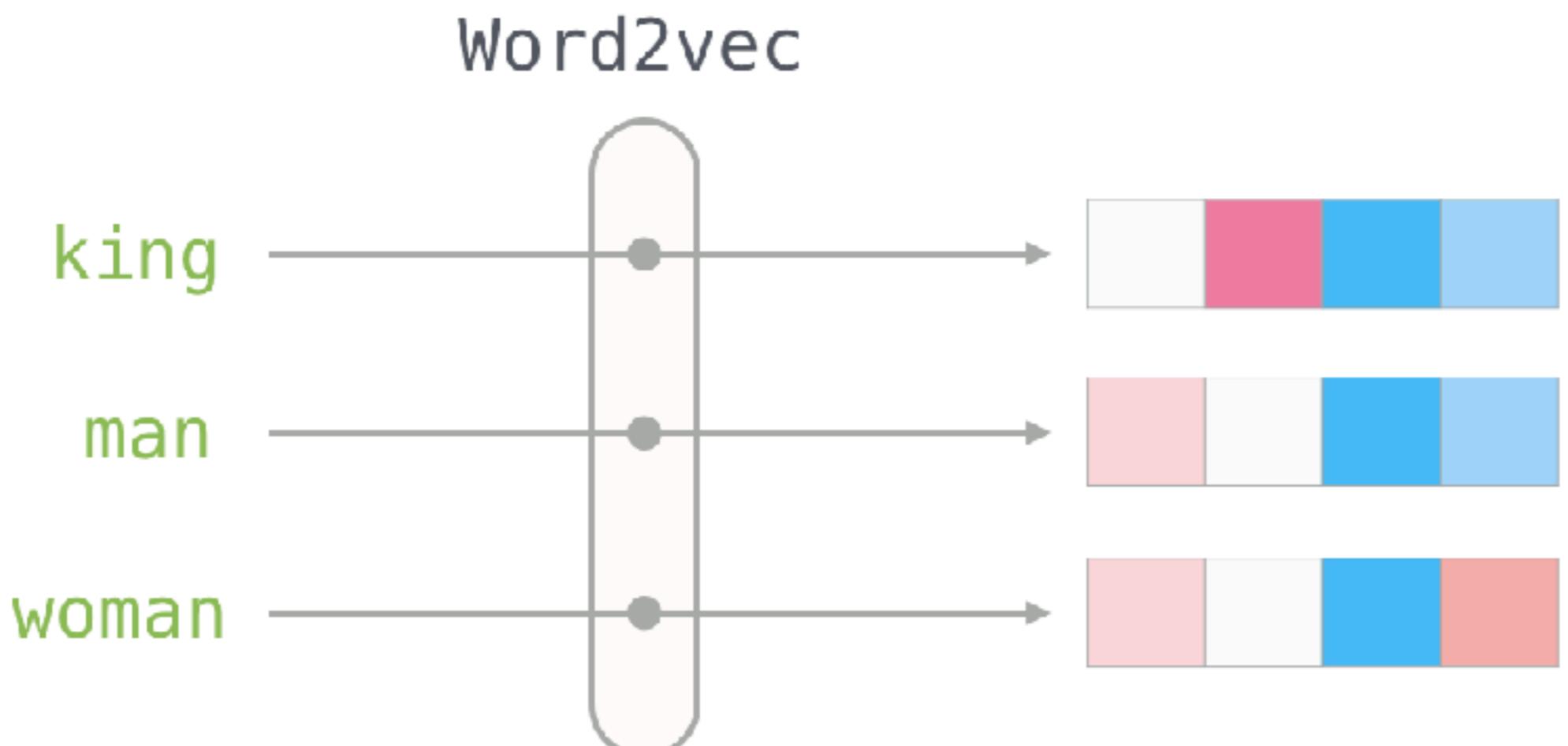
- \* Es un concepto muy importante para muchas aplicaciones



# Latent space

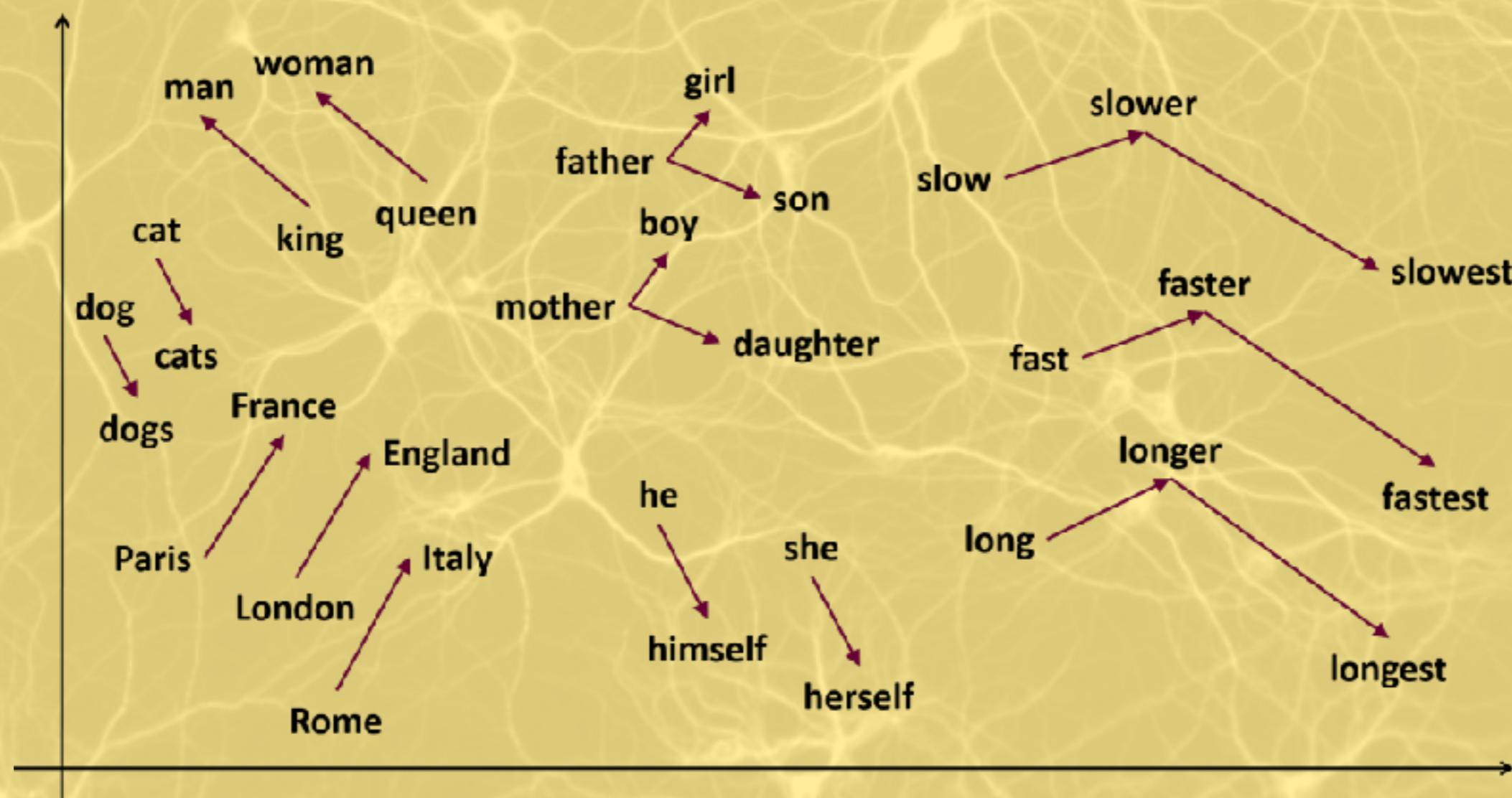


# Autoencoders en NLP



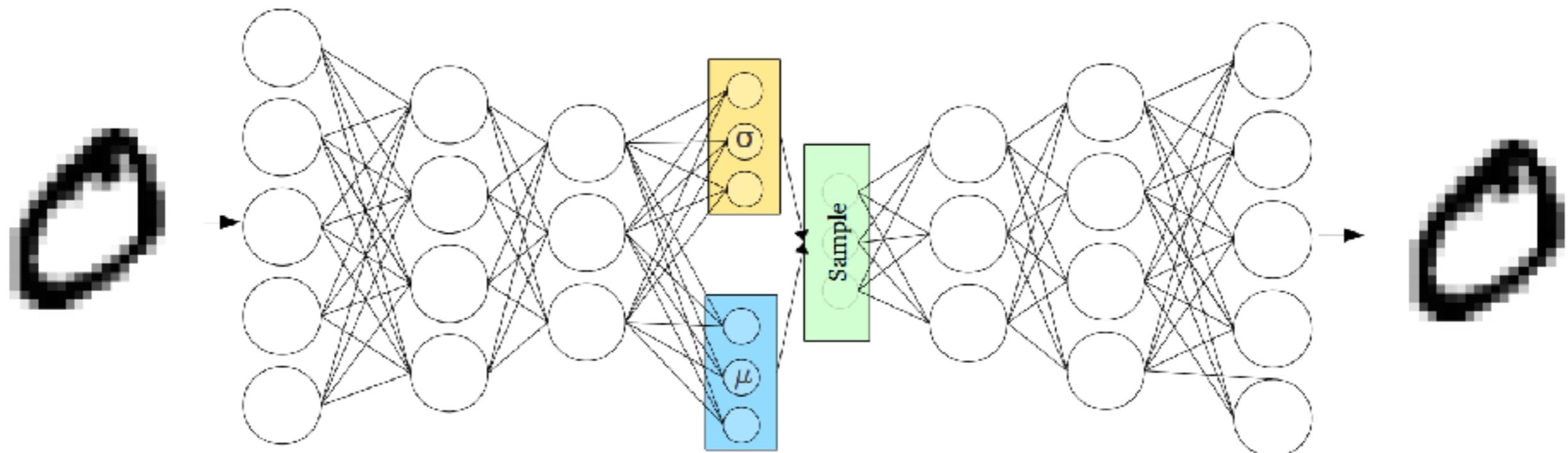
# Autoencoders y sparse data

- \* Para texto existen ya autoencoders descargables, por ejemplo word2vec, que transforman las palabras en un espacio latente, que además tiene propiedades interesantes



# Variational Autoencoders (VAE)

- \* En ellos el espacio latente está formado por medias y log(varianzas) de cada variable latente, buscando que la transición entre valores sea continua
- \* Las medias y las varianzas sirven para obtener una muestra aleatoria en cada propagación



# Algunos usos de los autoencoders

- \* Componentes principales
- \* Detección de anomalías
  - Si la salida no se parece a la entrada
- \* Trabajar con datos dispersos
- \* Traducción
- \* Transducción, por ejemplo texto a imagen
- \* Generalización
- \* Imputación de valores ausentes
- \* Clustering

# Redes de Convolución

- \* También conocidas como Convolutional Neural Networks (CNN)
- \* Se utilizan principalmente para problemas de tratamiento de imágenes y visión artificial
- \* Han sido capaces de obtener los mejores resultados en clasificación de imágenes hasta la fecha
  - Porcentaje de aciertos
  - Generalización
  - Eficiencia

# ¿Qué es una convolución?

- \* Es una operación que se aplica a cada pixel de una imagen, sustituyendo su valor por el obtenido por el sumatorio doble, dentro de un campo receptivo determinado, del valor de cada pixel del campo receptivo por un peso determinado
  - Diferentes pesos dan diferente resultados, por ejemplo extracción de bordes, detección de áreas de un color, etc.
- \* Se han usado tradicionalmente en procesado de señal para realizar diferentes tipos de filtrado
  - Paso alto
  - Paso bajo
  - ...

# ¿Qué es una convolución?

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

-1	-1	-1
-1	8	-1
-1	-1	-1

Máscara de convolución



# Cálculo de convoluciones

The diagram illustrates the convolution operation  $I * K$ . It shows three matrices: the input matrix  $I$ , the kernel matrix  $K$ , and the resulting output matrix  $I * K$ .

The input matrix  $I$  is a 7x7 grid:

0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0

The kernel matrix  $K$  is a 3x3 grid:

1	0	1
0	1	0
1	0	1

The resulting output matrix  $I * K$  is a 5x5 grid:

1	4	3	4	1
1	2	4	3	3
1	2	3	4	1
1	3	3	1	1
3	3	1	1	0

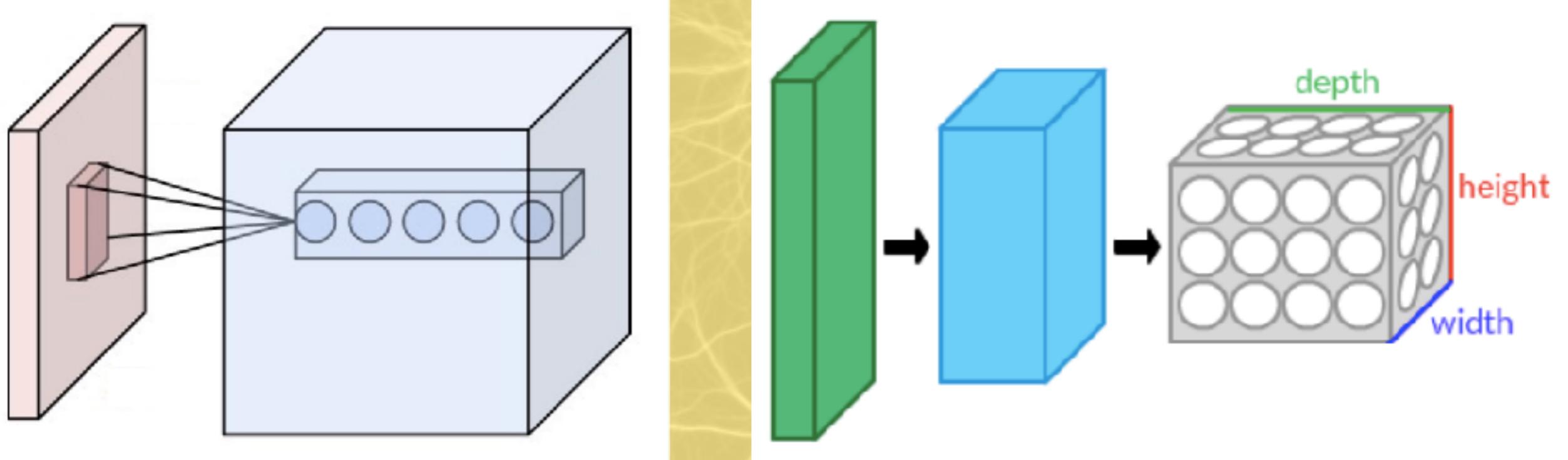
Dotted arrows show the receptive field of each output unit in  $I * K$  based on the kernel  $K$ . The output unit at position (2,2) in  $I * K$  is highlighted with a green box.

# Redes de convolución

## \* Incluyen una serie de conceptos nuevos

- Bidimensionalidad
  - ✖ Dada su aplicación típica a reconocimiento de imágenes, cada capa se suele representar de modo bidimensional, manteniendo así el formato de imagen
- Capas paralelas (tridimensionalidad)
  - ✖ Además en un mismo nivel se incluyen varias capas (canales o features), de modo que cada una realiza un procesamiento diferente (según sus pesos). De este modo podemos pensar que cada capa es tridimensional
- Convolutional layers
  - ✖ Son capas que se aplican sobre muchas partes de la imagen, de modo que de cada entrada (imagen) salen muchas salidas (resultado de la aplicación de cada máscara de convolución)
  - ✖ Nótese que el cálculo de la entrada neta en un perceptrón es idéntico a una convolución
- Campo receptivo
  - ✖ Área sobre la que se aplica una convolución (tamaño de la máscara de convolución)
- Pooling layers
  - ✖ Son capas que reducen la dimensionalidad quedándose con el valor máximo de sus entradas, normalmente de 2x2

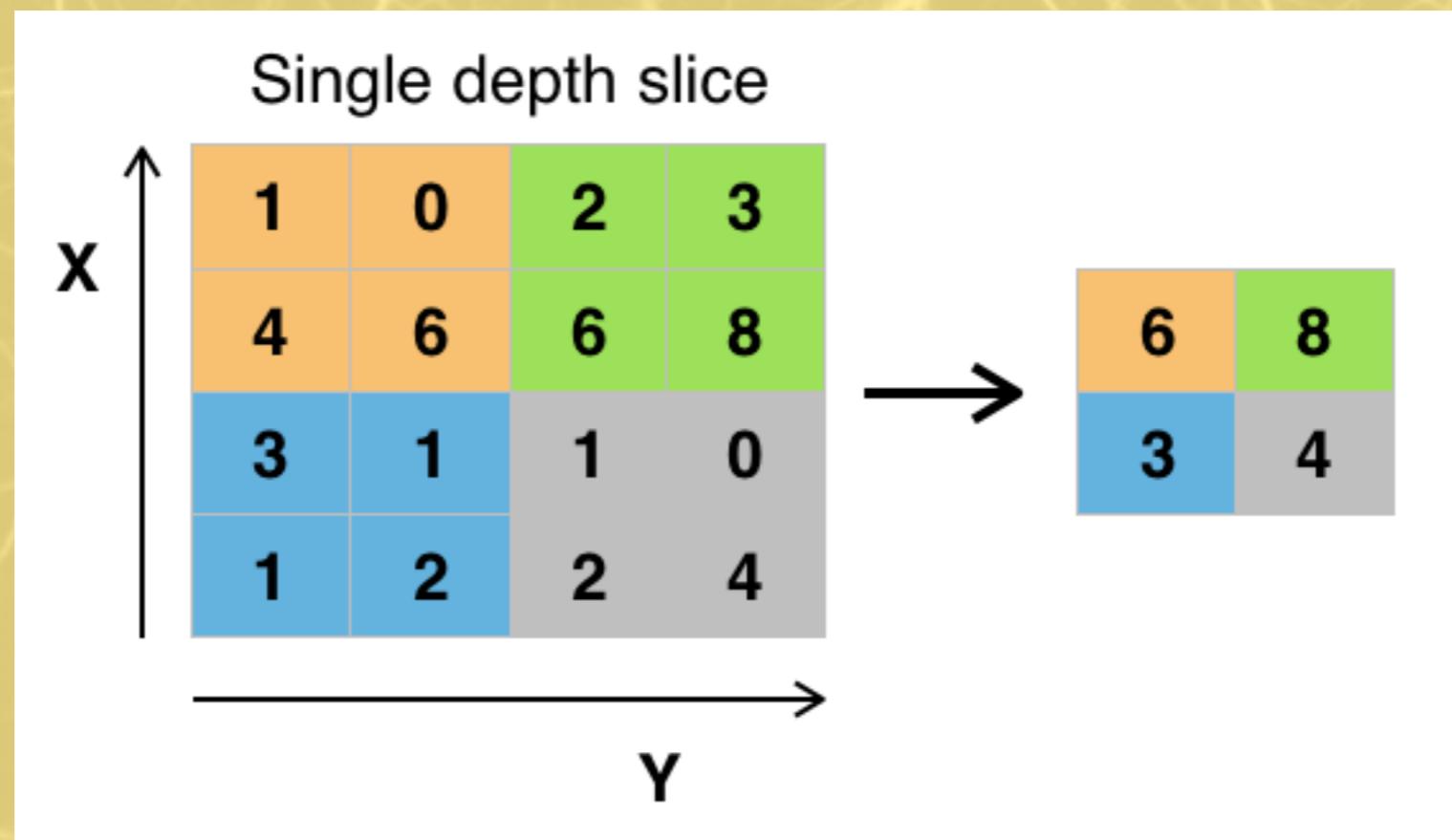
# Campo receptivo



- \* Width y Height: tamaño del campo receptivo, ej. 5x5, 25x25
- \* Depth: número de diferentes convoluciones que hace esa capa
- \* Stride: cuántos píxeles nos movemos cada vez (normalmente 1)
- \* Zero-padding: si se deja un marco de ceros alrededor

# Pooling Layer

- \* Sustituye cada cuadrado de  $n \times n$  valores de la entrada por su valor máximo
  - Inicialmente se usaba el valor medio, pero se ha demostrado que da mejor resultado el máximo, y además es más rápido



# Arquitecturas

## \* Ejemplos

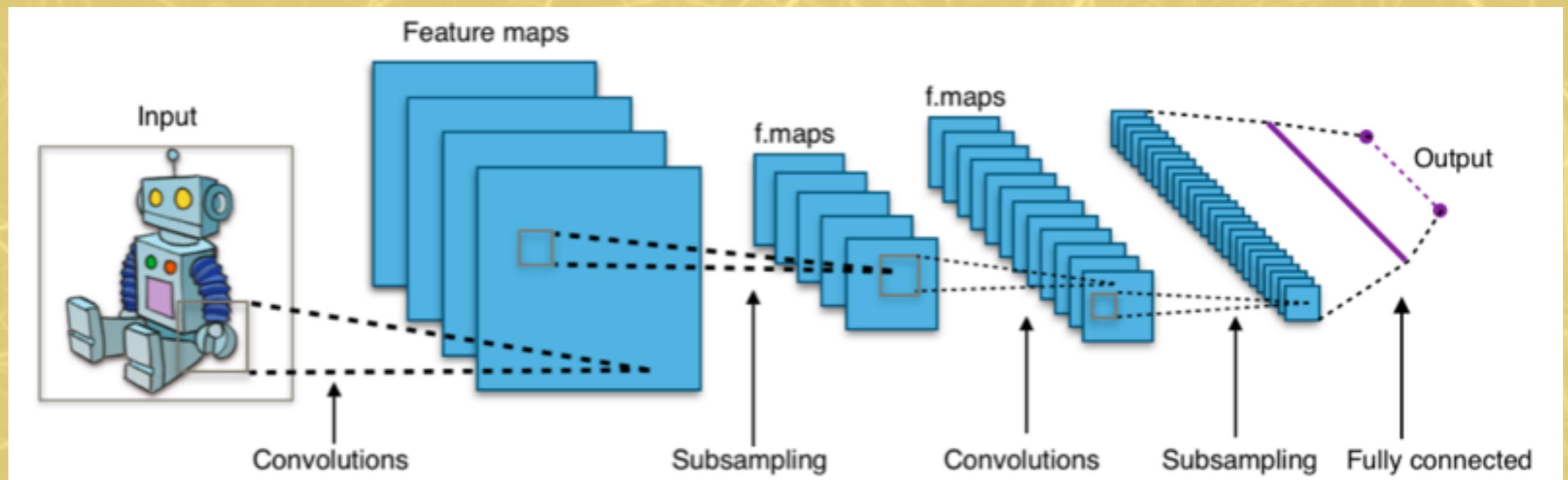
- INPUT → FC (clasificador lineal)
- INPUT → CONV → RELU → FC
- INPUT → [CONV → RELU → POOL]\*2 → FC → RELU → FC (una capa de convolución antes de cada capa POOL)
- INPUT → [CONV → RELU → CONV → RELU → POOL]\*3 → [FC → RELU]\*2 → FC (dos convoluciones antes de cada POOL)

## \* Normalmente se apilan muchos convolutional layers

## \* Entrenamiento

- BackPropagation y derivados
- Las neuronas ReLU evitan el estancamiento por saturación (desvanecimiento del gradiente)

# Juntando las piezas



# Entradas, salidas y pesos

- \* En este tipo de red, los pesos a aprender son las máscaras de convolución, o filtros
- \* Las entradas de la capa de entrada son imágenes y feature maps en el resto de capas
- \* Cada convolución se aplica a todos los canales de la capa anterior y sus resultados se suman
- \* En las redes tradicionales teníamos muchos pesos y pocos valores de activación, en éstas tenemos muchos valores de activación y proporcionalmente menos pesos
- \* El entrenamiento de la red se hace también por descenso del gradiente, y consiste en encontrar los filtros óptimos para la tarea planteada

# Keras: sequential vs functional models

- \* En Keras, podemos trabajar con modelos secuenciales o con modelos funcionales
- \* Dependiendo de lo que queramos tienen ventajas e inconvenientes
- \* Modelo secuencial: se crea y se van añadiendo capas
- \* Modelo funcional: se van aplicando capas a las entradas, que pueden ser salidas de cualquier otra capa en el modelo
  - Interesante si el modelo no es feed forward (por ejemplo redes residuales)

```
model = Sequential()  
model.add<leyer>()  
)  
dense = layers.Dense(64, activation='relu')  
x = dense(inputs)
```

```
x = layers.Dense(64, activation='relu')(x)  
outputs = layers.Dense(10, activation='softmax')(x)
```

# Capa de convolución en Keras

- \* Conv2D(<filtros>, <tamaño kernel>, [<activación>, <input\_shape>])

```
#create model
model = Sequential()

#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu',
input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```

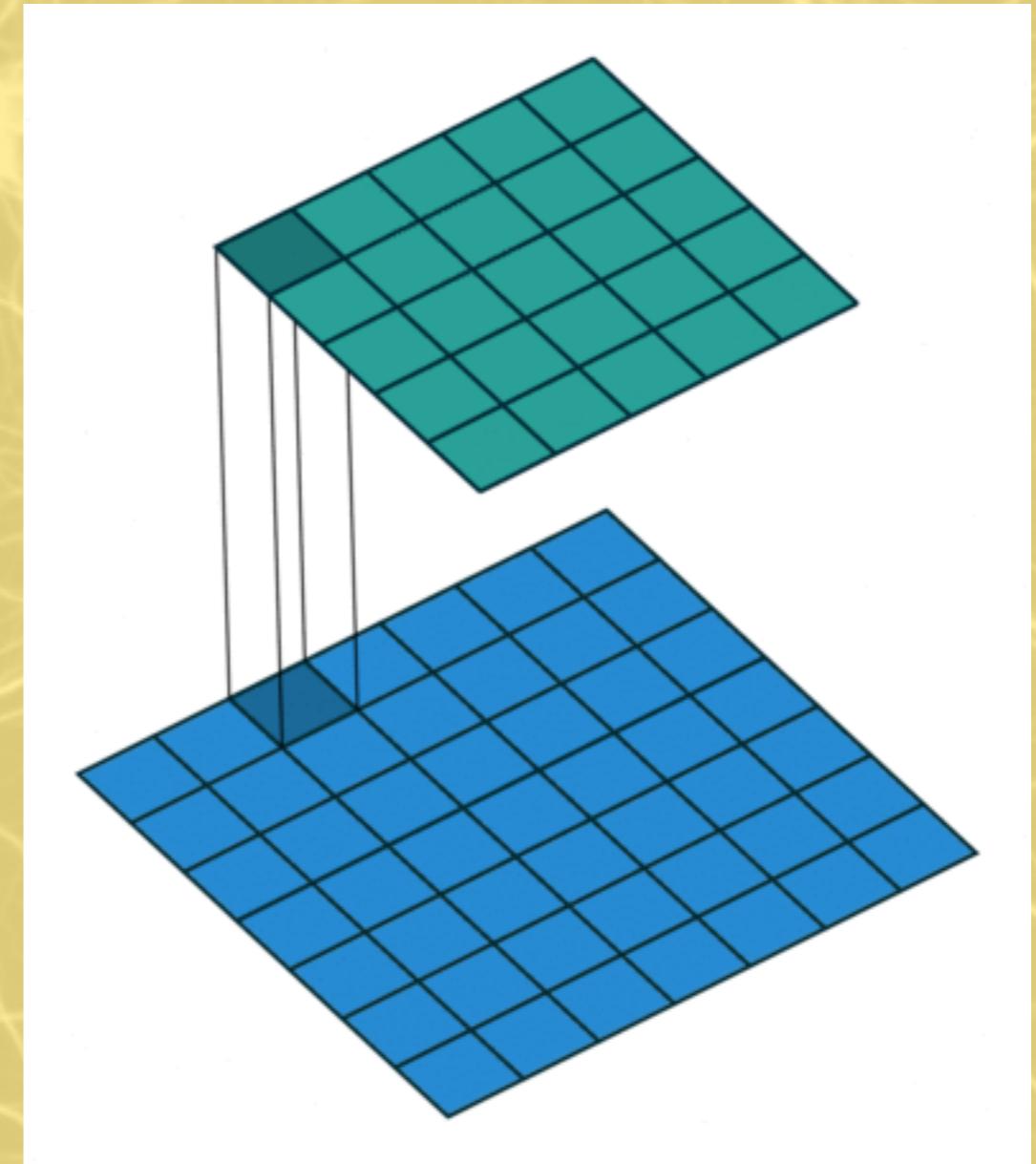
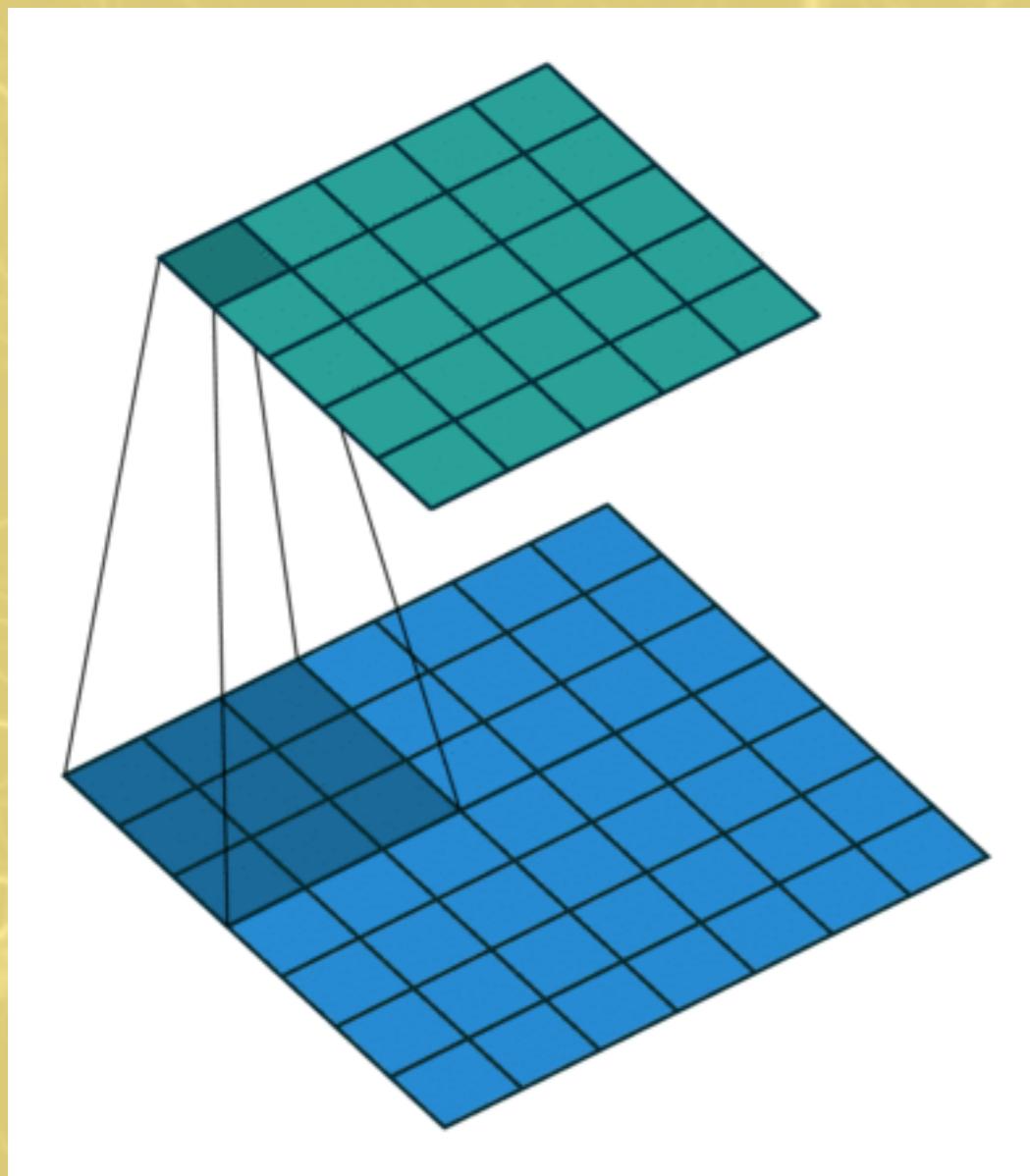
# Conceptos avanzados

- \* Desde la formulación original de Yann LeCun se han propuesto diversas mejoras a las redes de convolución que han aportado una aumento sustancial de la tasa de aciertos en clasificación
  - Normalización (Local Response o LRN, Batch o BN)
  - Convoluciones 1x1
  - Inception layers, colocan en paralelo convoluciones de varios tamaños)
  - Redes residuales (resnet), puentean la entrada a la salida
- \* En general se ha encontrado que aumentando el número de capas mejora el ratio de aciertos en test (generalización), aunque las capas están limitadas por el hardware disponible
- \* Se han usado masivamente para clasificación, pero ¿pueden usarse las redes de convolución para regresión?

# Normalización de salidas de capas

- \* La normalización de respuesta local responde a un efecto que se da en la biología: la inhibición lateral
  - Las neuronas próximas poseen conexiones inhibidoras con las neuronas cercanas
  - Esto hace que la que más se active tienda a hacer menos activas a las neuronas adyacentes
  - Lo que hace la normalización es aumentar la activación de las más activas y disminuir la activación de las menos activas, es decir, aumentar el contraste
- \* La normalización en batch (BN) busca que la distribución de los píxeles siga una  $N(0,1)$ , dividiendo por la varianza y restando la media de los valores en cada mini-batch
- \* <https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac>

# Convoluciones 1x1



- \* Sirven para reducir el número de canales

# Inception layers

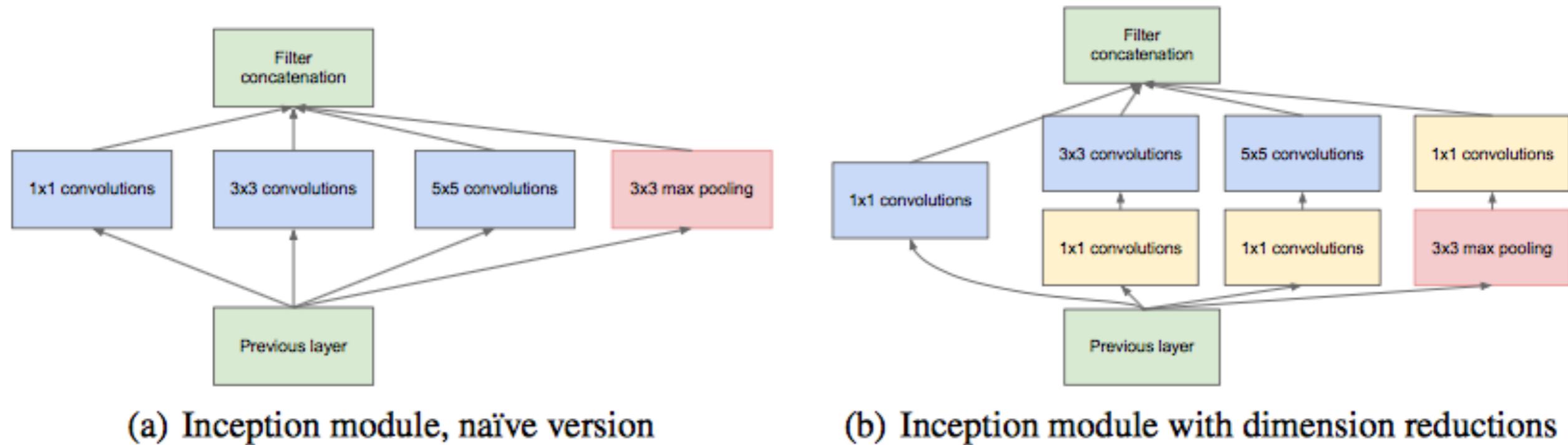
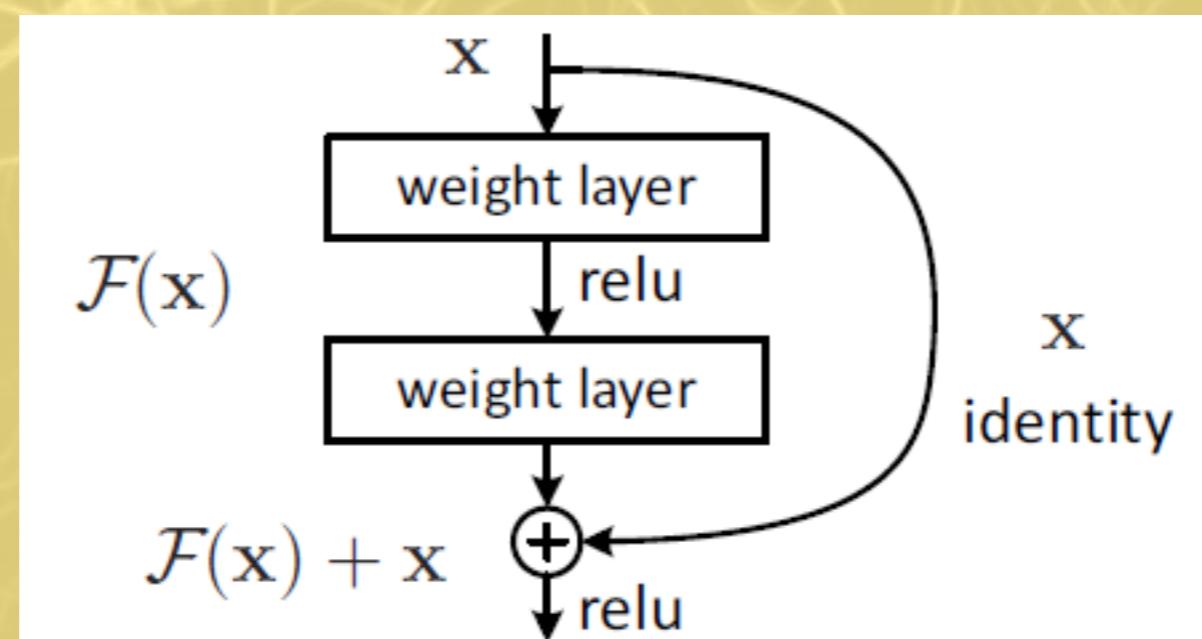


Figure 2: Inception module

# Capas residuales

- \* Las entradas se un grupo de conexiones se suman a las salidas
  - Esto permite que la retropropagación del gradiente funcione mejor.
  - Si las dimensiones no coinciden se usa una transformación lineal mediante una capa completamente conectada con pesos entrenables.
  - La capa toma la entrada original como entrada y produce una salida con las dimensiones necesarias para que se pueda sumar con la salida de la capa



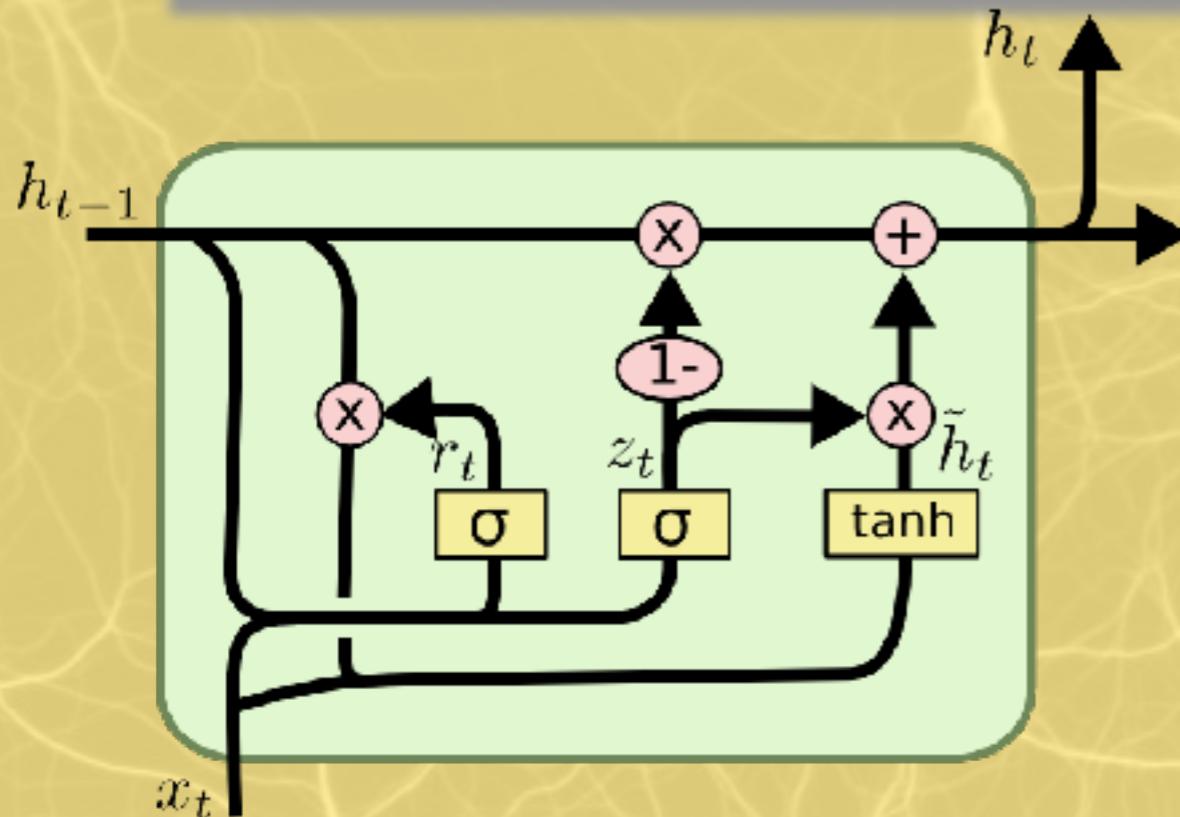
# Transfer Learning

- \* A pesar de la gran potencia de cálculo de las GPUs, entrenar las redes de última generación puede llevar muchas horas / días
- \* Transfer learning consiste en cargar una red ya entrenada, por ejemplo para imagenet, y entrenar los full connected layers para otro propósito
  - En back propagation se realiza para las últimas capas, que son las FC, pero también para algunas (o todas) las convolucionales (fine tuning)
  - La idea es que las capas de convolución han aprendido a detectar las cosas relevantes en las imágenes, que debería ser “independiente” del problema a resolver y sólo tienen que ajustarse levemente para el problema nuevo
  - La capacidad de generalización de este procedimiento es mucho mayor que entrenando de cero
  - Alternativamente pueden bloquearse algunas de las primeras capas convolucionales

# Redes LSTM

- \* Incluyen realimentación dentro de cada capa
- \* Esto permite que la red tenga “memoria” de lo que hizo en las iteraciones anteriores, son por tanto buenas para problemas en los que se reconocen o producen secuencias
  - Predicción de texto
  - Generación de música
  - Series temporales: mercado eléctrico, tiempo meteorológico, bolsa...

# Redes LSTM



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- \* Básicamente, la salida de una LSTM depende de la entrada y de la salida en t-1, cada una ponderada por unas matrices que se aprenden
- \* `tf.keras.layers.LSTM(<n>)`

- \* [a, b] → concatenar
- \* a \* b → producto elemento a elemento
- \* · → producto matricial
- \*  $\sigma$  → sigmoidal

# Redes LSTM

## \* Composición automática con redes LTSM

- <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>

## \* Generación de textos (char-nn)

- Experimento con poetas del 27

### Epoch 1000

Pera el ciella  
de está sencino. Como a cabiera,  
para abancuros meriestes,  
¿Sé quevintur...  
se ala hueto oso para vestiento de azul  
santa.  
Ayepleran tu alcho.  
Maderacente  
a la nojo, en no sétro,

### Epoch 10000

el pobre sólo sabría tu nombre,  
y la espesa tierra invenizo soledad en el  
frío desciende.  
Bajo la luna viavera  
y en lo lámtica, mujer que amé!  
¡campana y con un palacio desierto.

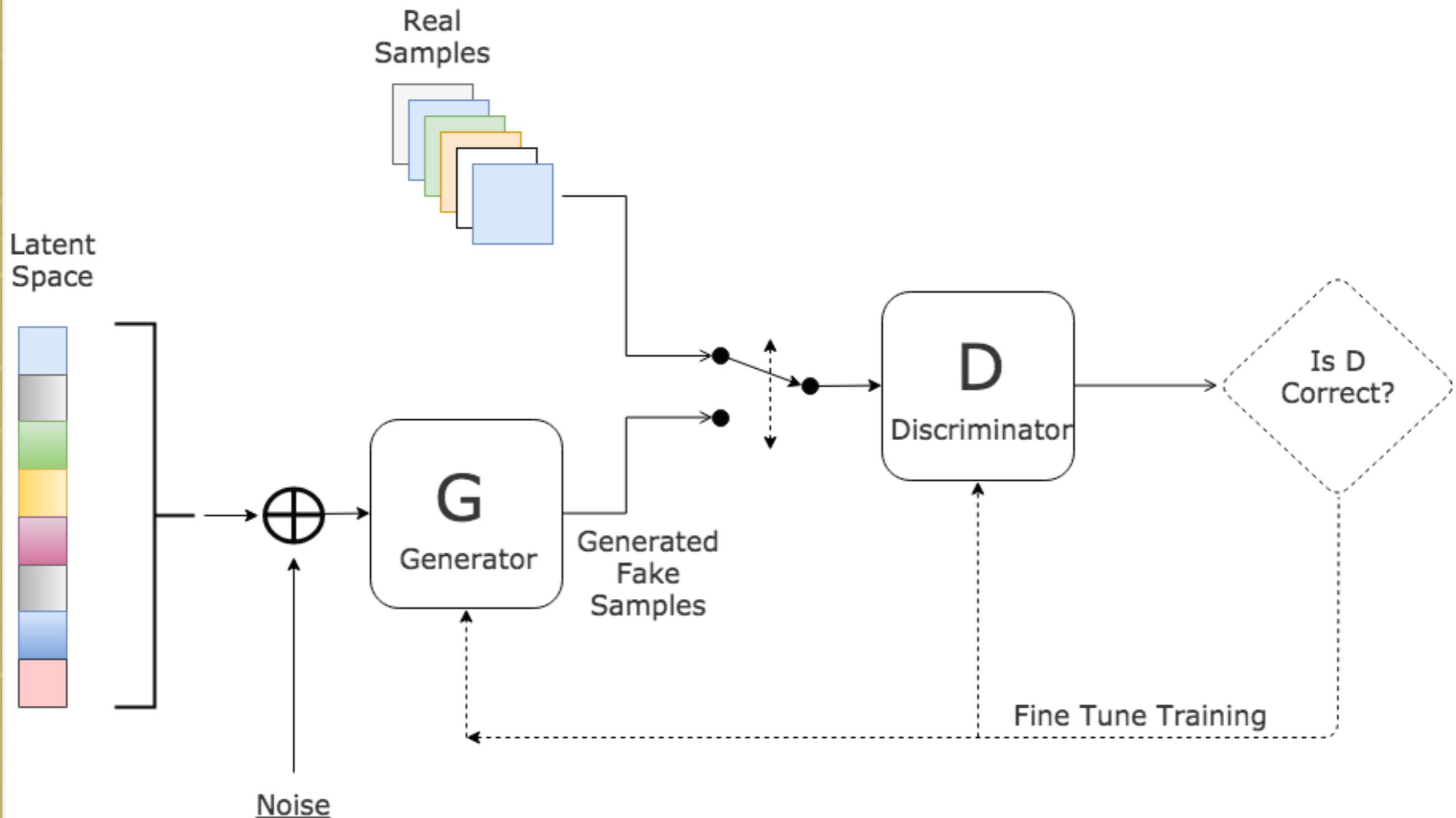
Tengo miedo.

# Generative Adversarial Networks (GAN)

- \* Goodfellow, 2014 (<https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html>)
- \* Es una arquitectura de red que sirve para generar ítems (imágenes, música...) que sean indistinguibles de ítems reales
- \* Tiene dos componentes, una red generadora que crea ítems y una red discriminante que dice si el ítem es real o no
- \* Las dos redes coevolucionan, la generadora para engañar a la discriminante y la discriminante para no ser engañada
- \* Usos: creación artística, creación de ejemplos “artificiales”...

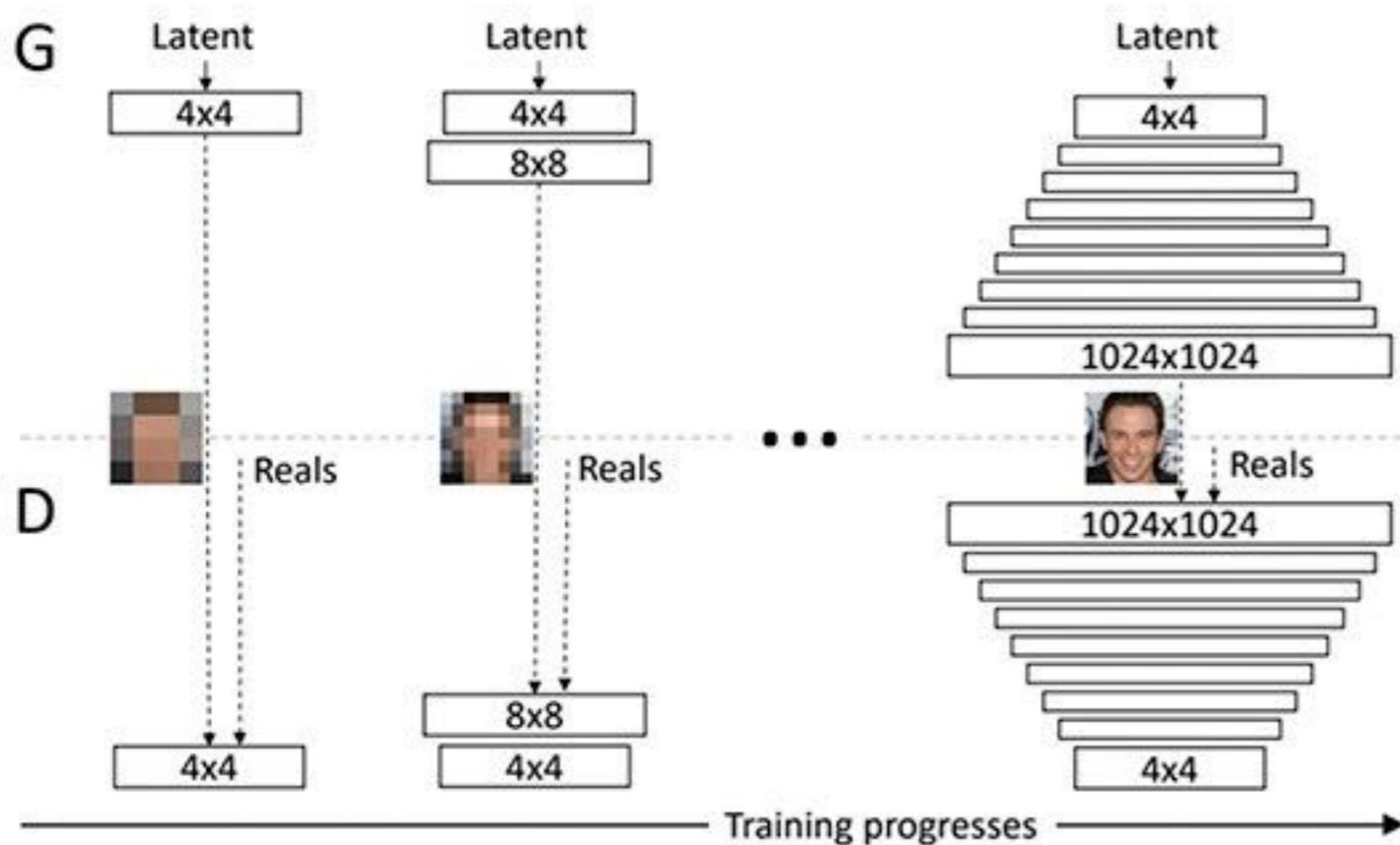
# Generative Adversarial Networks (GAN)

## Generative Adversarial Network

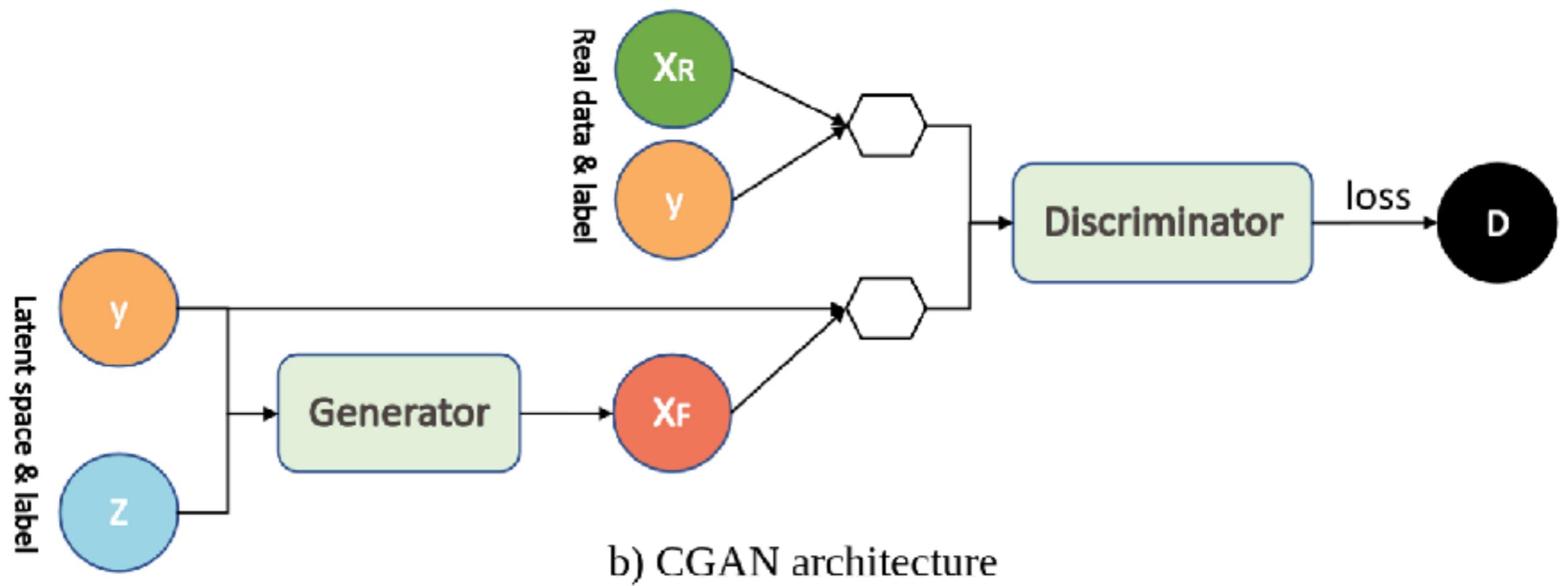
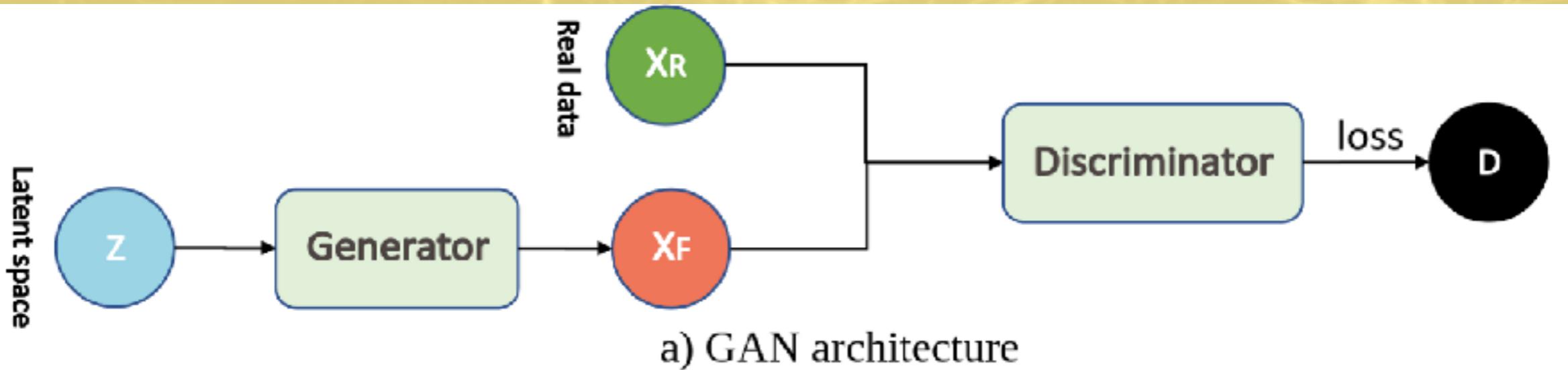


# Ejemplo GAN

## Photorealistic Image Generation Using GAN



# Conditional GAN



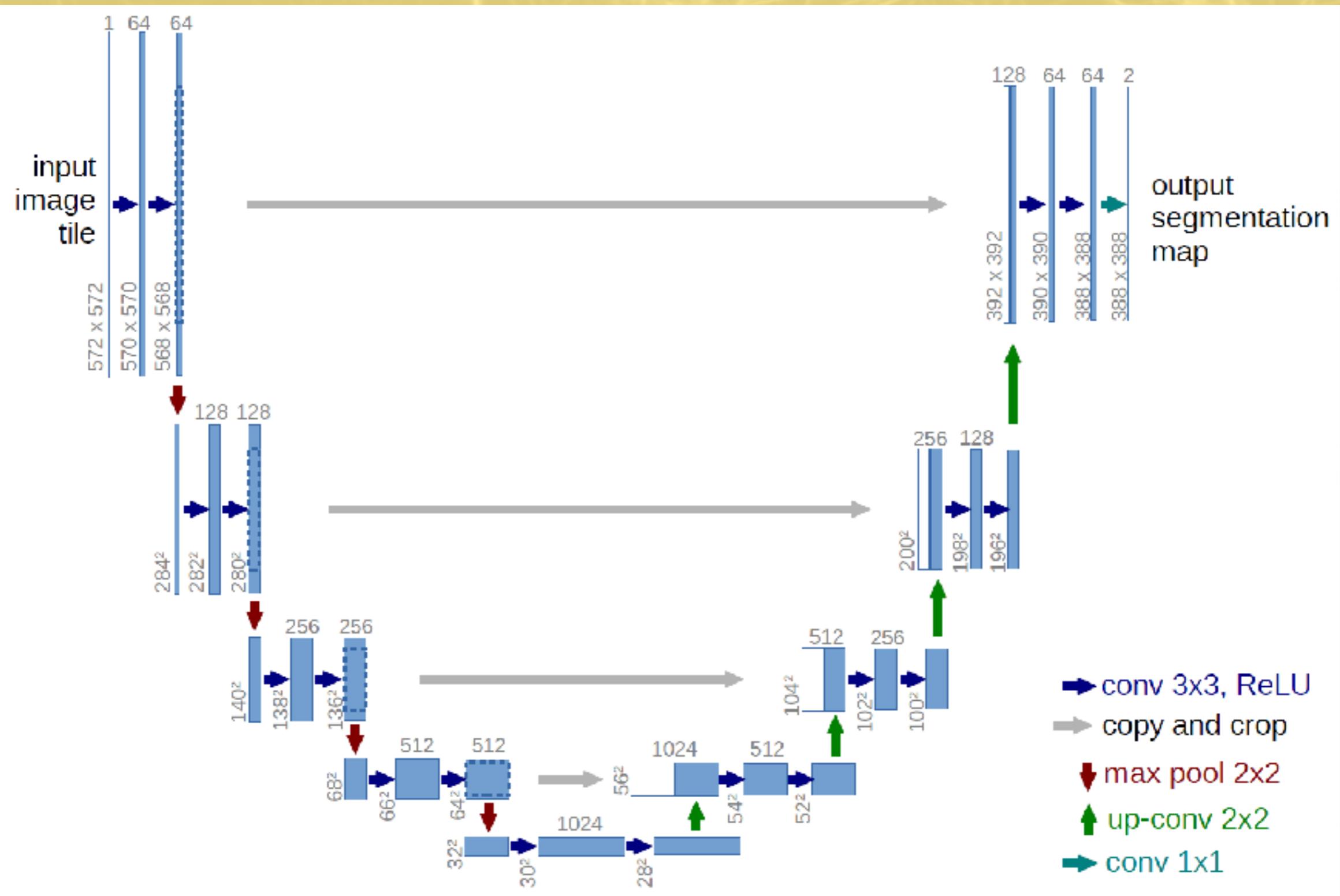
# Generative Adversarial Networks (GAN)

## \* Problemas de las GAN

- Los indicadores tradicionales (loss, accuracy) no nos sirven de mucho al entrenar una GAN, porque al ser antagónicas los progresos del discriminado son compensados por el generador
  - ✖ Si el loss de una de las dos redessube indefinidamente tenemos un problema
- Podemos detectar dos problemas durante el entrenamiento
  - ✖ **Colapso modal.** La red aprende a generar siempre lo mismo: lo hace muy bien pero no hay variedad
  - ✖ El discriminador no aprende: siempre generará ruido y lo tomará como correcto (el loss del discriminador no baja)
  - ✖ El generador no aprende: siempre generará ruido y el discriminador siempre acierta. El loss del discriminador tiende a 0. A veces puede salir de esta situación: esperar unos epochs. Si la tarea es más difícil para el generador que para el discriminador puede ser necesario darle más epochs de entrenamiento
  - ✖ El generador no crea copias de calidad: puede deberse a que no es lo suficientemente potente para la tarea

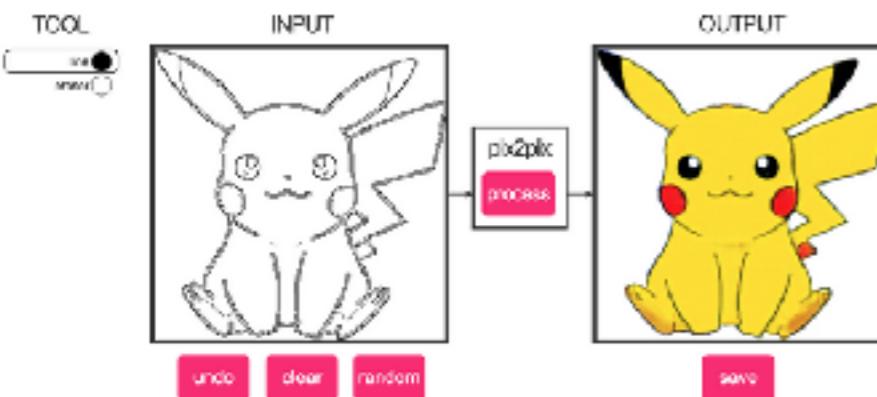
\* <https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/>

# UNet (Fischer y Brox, 2015)

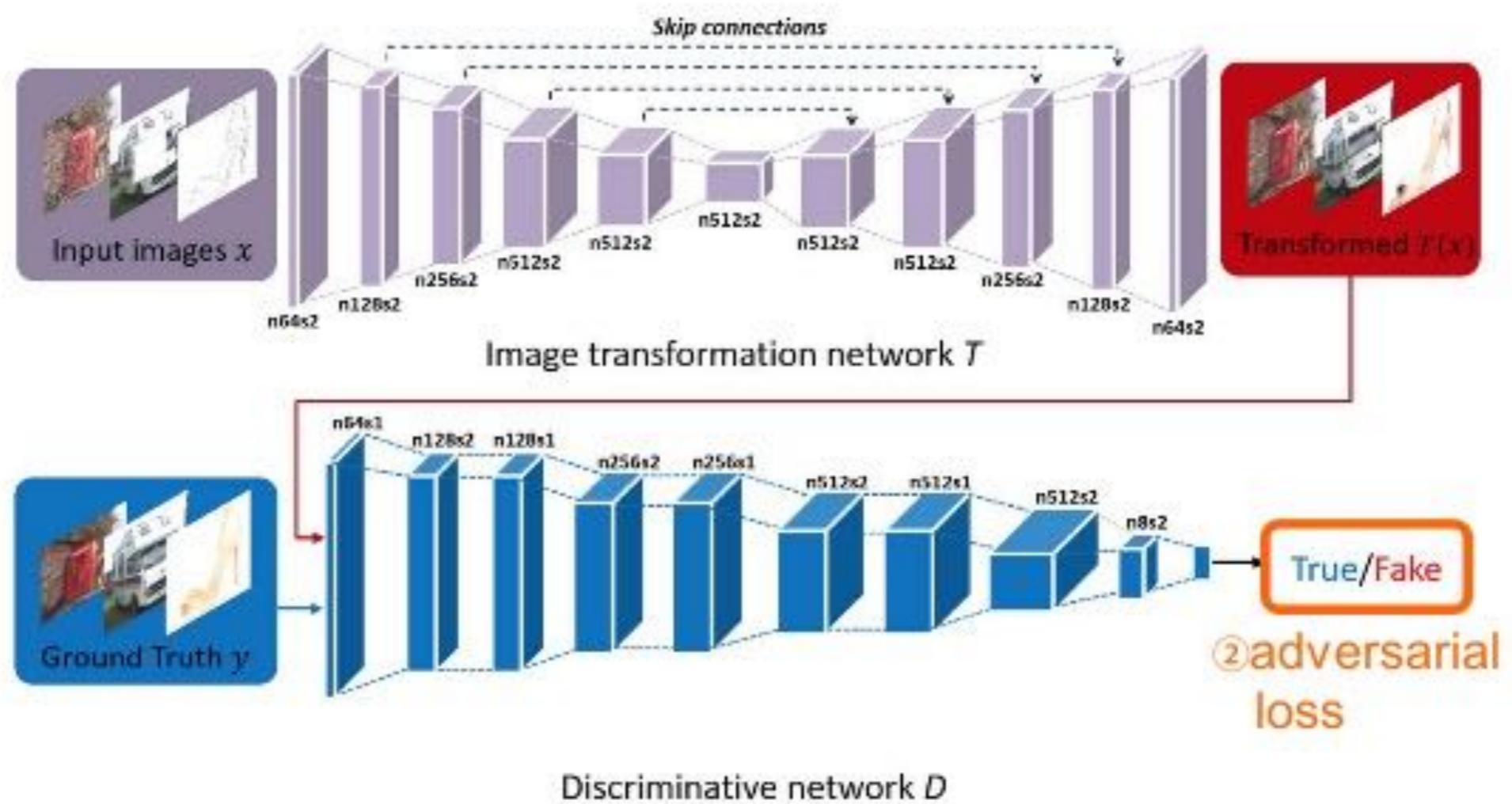


# Pix2pix networks

pix2pix edges2pikachu

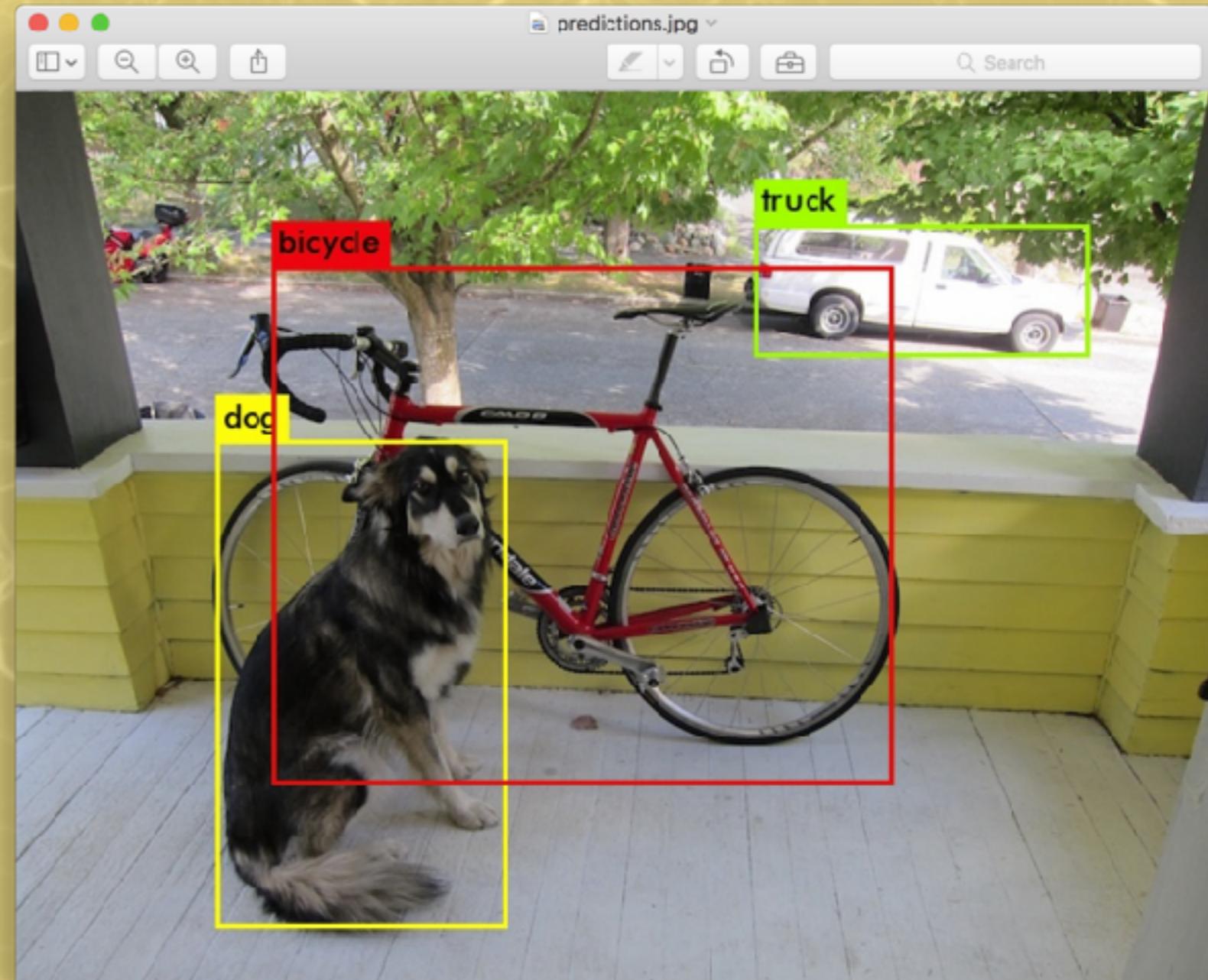


**Pix2Pix (①+②)**

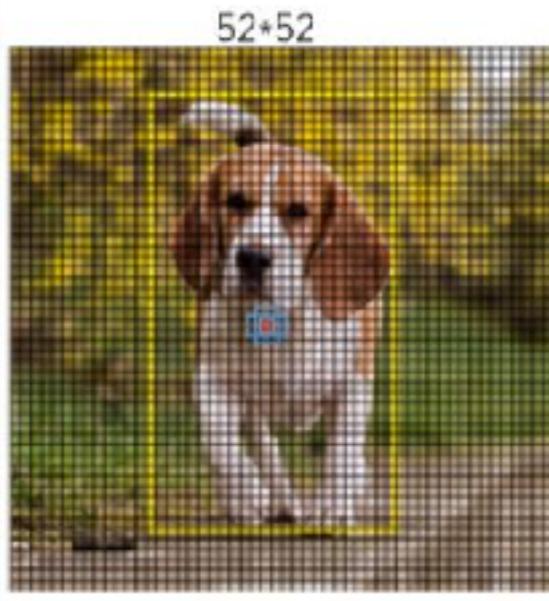
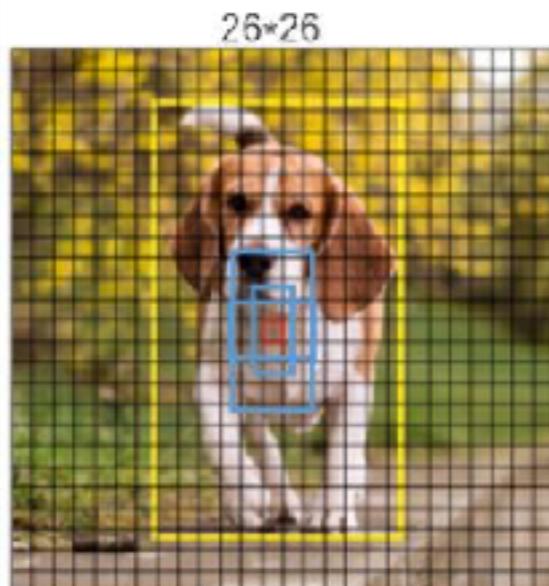
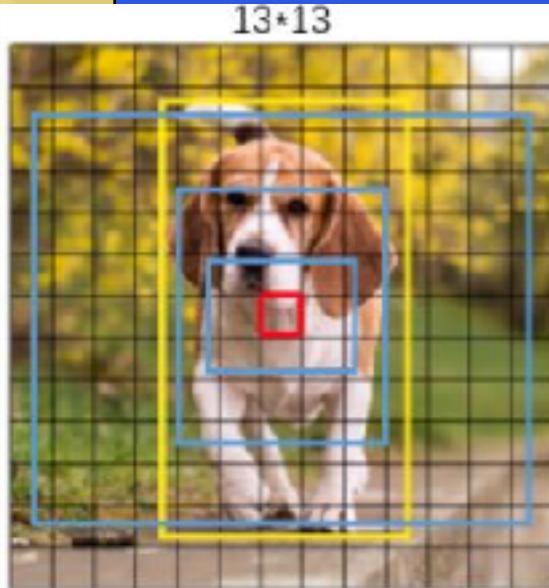


# YOLO

- \* YOLO (You Only Look Once) es una red entrenada para localizar la posición de ciertos tipos de objetos dentro de una imagen
- \* Aunque ya viene preentrenada para una serie de categorías es posible hacer un fine Tuning para nuestros propios objetos

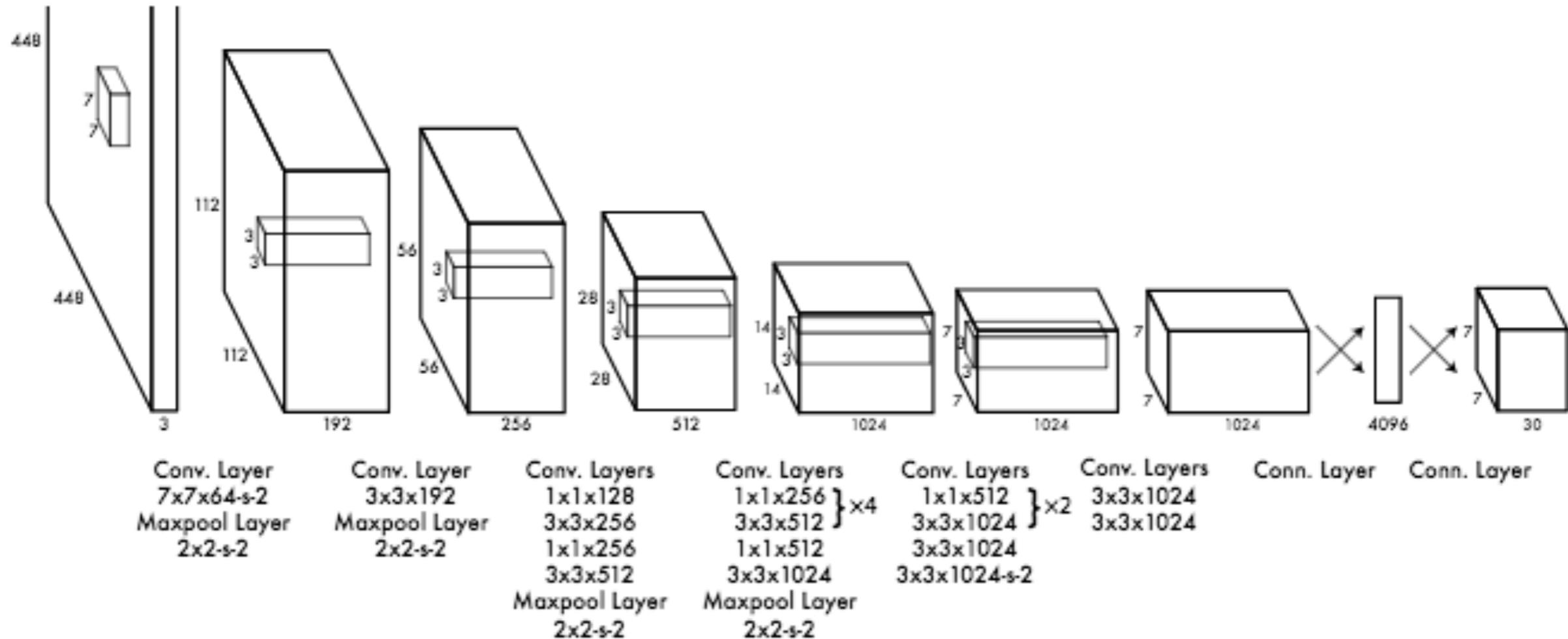


# YOLO

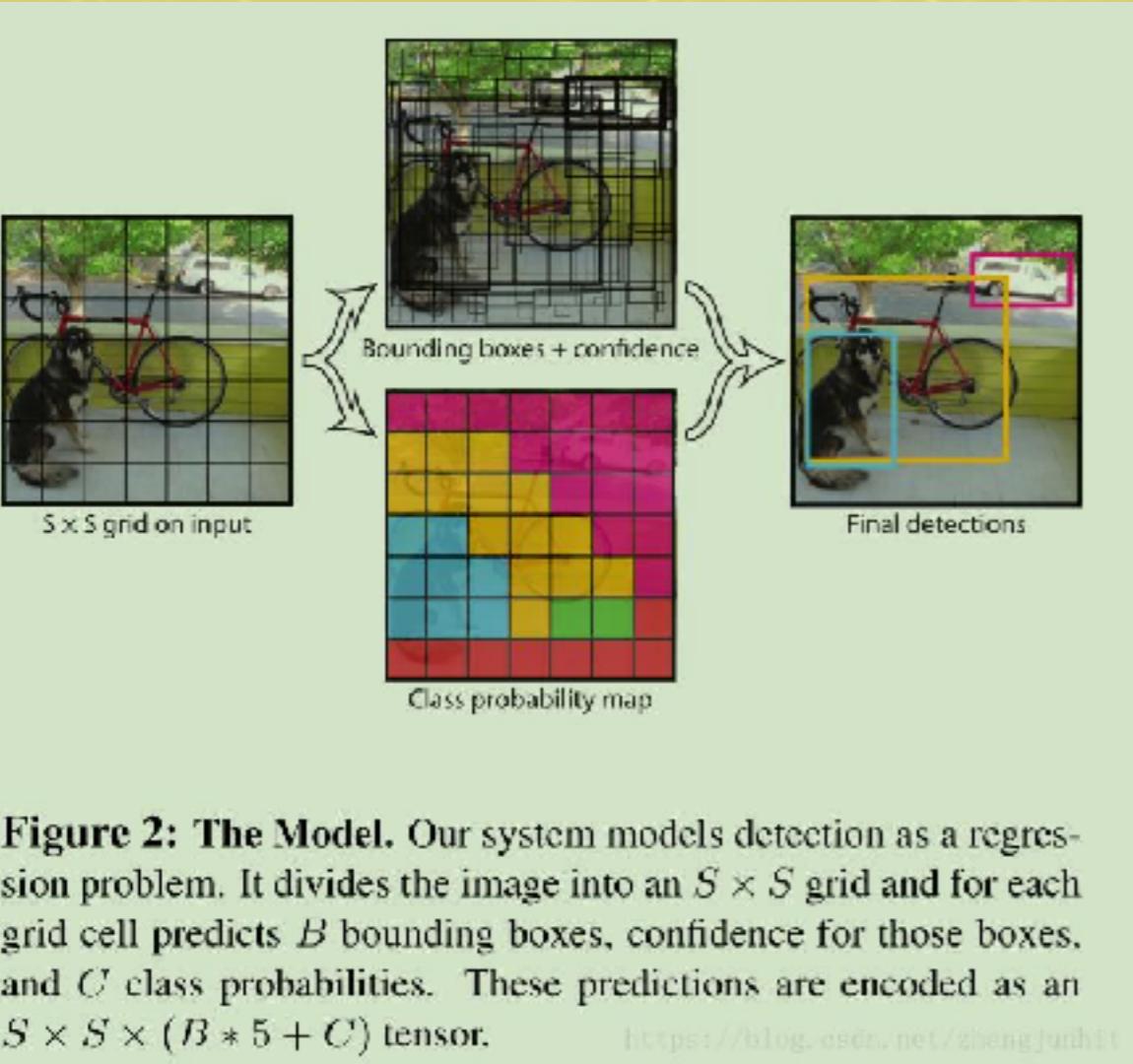


- \* Los primeros sistemas para localizar objetos aplicaban redes a muchas posiciones y escalas, lo cual no era eficiente
- \* YOLO utiliza un enfoque diferente: se aplica una sola red a toda la imagen, que divide la imagen en regiones y predice la posición, ancho y alto del objeto, así como la probabilidad de que sea cada uno de los posibles objetos
- \* La nueva estrategia es tan eficiente que se puede procesar vídeo en tiempo real

# YOLO v1



# YOLO v1



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

<https://blog.csdn.net/zhangjinhit>

This parameterization fixes the output size

Each cell predicts:

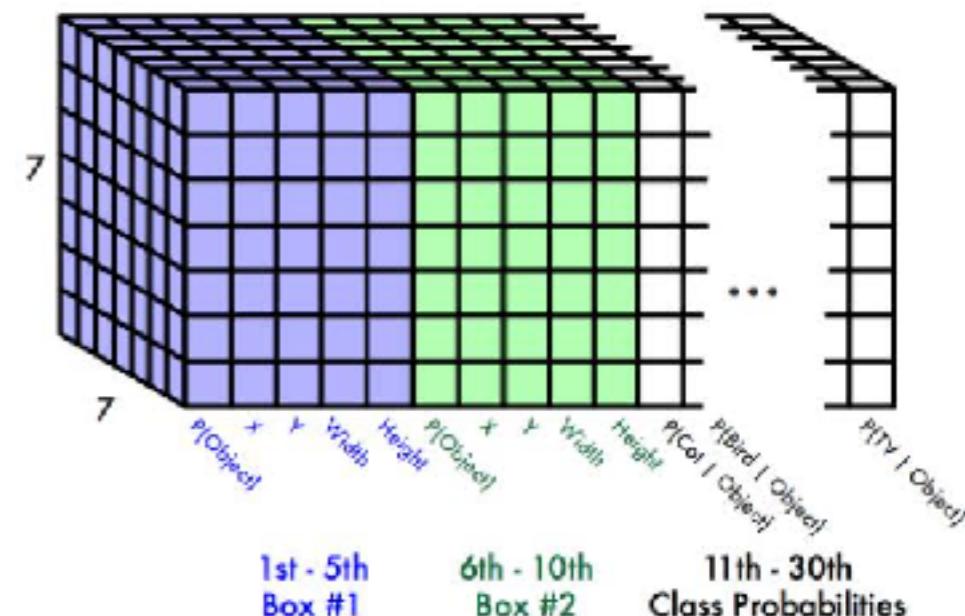
- For each bounding box:
  - 4 coordinates ( $x, y, w, h$ )
  - 1 confidence value
- Some number of class probabilities

For Pascal VOC:

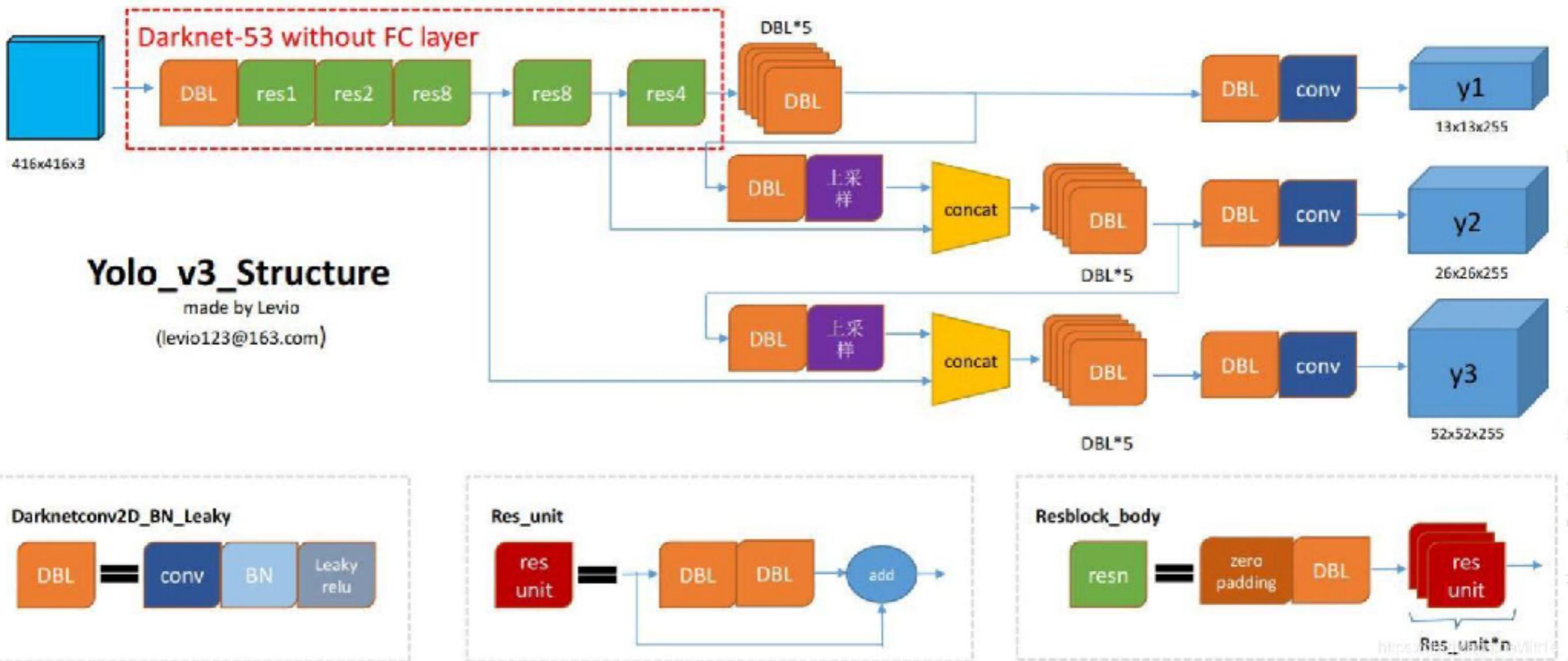
- $7 \times 7$  grid
- 2 bounding boxes / cell
- 20 classes

$$7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30 \text{ tensor} = \mathbf{1470 \text{ outputs}}$$

<https://blog.csdn.net/zhangjinhit>



# YOLO v3



# Follow Me

- \* Proyecto de seguimiento de tropas mediante YOLO v3
- \* Aplicación a un vídeo de Londres
- \* Aplicación en el INSIA, corriendo sobre una NVIDIA Jetson Nano, en tiempo real (básicamente una Raspberry Pi con una mini GPU de 128 núcleos)
  - Aprox. 1 frame / seg.
- \* Portado a una Jetson AGX Xavier (GPU 512 núcleos)
  - 5,5 TFLOPS
  - aprox. 5 frames / seg.



# Hardware

## \* Disponible en AICU

- Nvidia GeForce GTX 3090 Ti
  - ✖ 10752 + 336 núcleos a 1560 MHz
  - ✖ 24GB GDDR6X a 21 GHz
  - ✖ Consumo: 450w
  - ✖ Rendimiento 40 TFLOPS

## \* Disponible en INSIA

- GeForce GTX 1080 (+ GeForce Titán)
  - ✖ 2560 núcleos a 1607 MHz
  - ✖ 8GB GDDR5X a 10 Gbps
  - ✖ Consumo: 180w
  - ✖ Rendimiento 8.6 TFLOPS

## \* Disponible en Mercator

- Rack Dell con x4 GPUs NVidia Tesla v100
  - ✖ 80 CPU cores
  - ✖ 640 Tensor Cores a 1380 MHz por cada GPU
  - ✖ 100 TFLOPS por GPU

## \* Otros

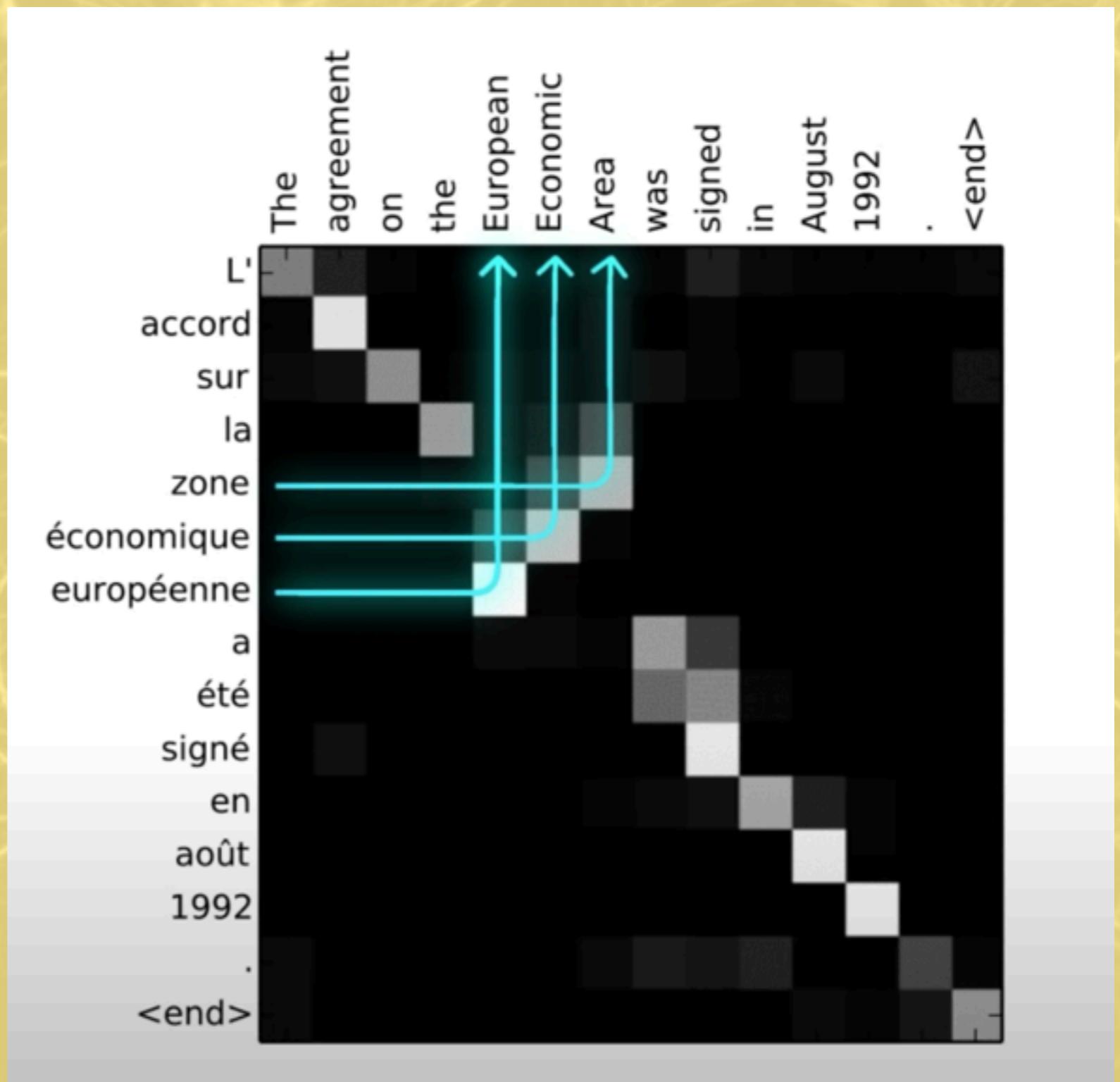
- M1 Max
  - ✖ 1024 cores (32x8x4)
  - ✖ 10,4 TFLOPS
- M1 Ultra
  - ✖ 2048 cores (64x8x4)
  - ✖ 21 TFLOPS
  - ✖ 39w a máxima carga

# Attention Layers

- \* Las primeras redes para PLN utilizaban redes recurrentes, pero la influencia de una palabra en otra se atenuaba rápidamente
- \* Un attention layer permite aprender la importancia de cada palabra de una secuencia en función del contexto
  - Son muy interesante en problemas donde el contexto es relevante
  - Son equivalentes a tener pesos que cambian en función del contexto, frente a los pesos tradicionales que son fijos (hard)
- \* Se proyectará el embedding de cada palabra en tres vectores, value, key y query, que representan lo que ofrece la palabra y lo que busca en otras
- \* El producto escalar de la query de una palabra por la key de cada una de las otras nos indica cuánto la influye cada una en ella, o visto de otro modo, a qué parte de la frase está prestando atención

# Attention Layers

- \* Si calculamos todas con todas tenemos una “matriz de atención”



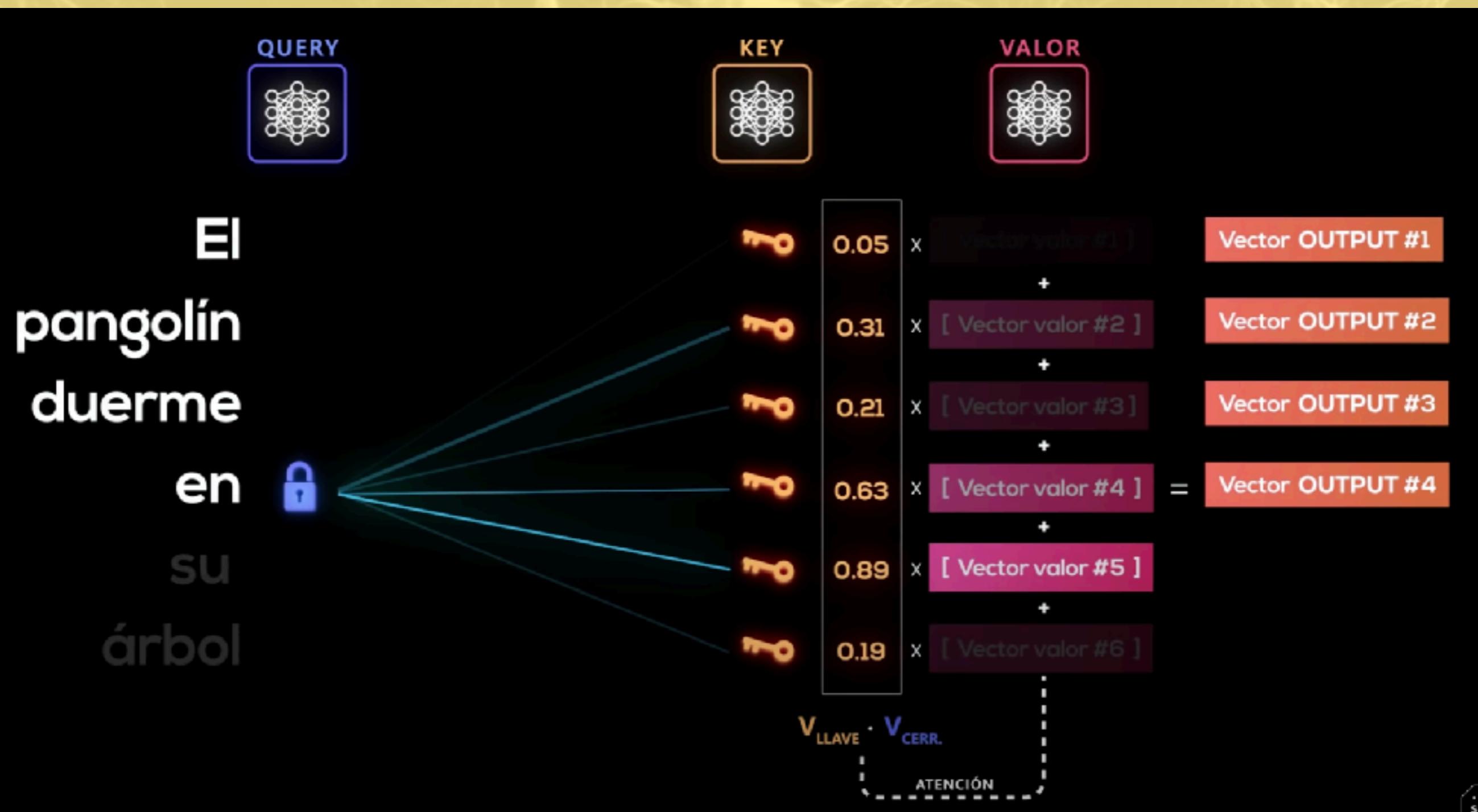
# Attention Layers

- \* Cada palabra tiene 3 componentes
  - La query, o qué busca esa palabra en otras
  - La key, o qué ofrece esa palabra a otras
  - El value, que representa su significado
  - Los tres se obtienen a partir del embedding ( $x$ ) mediante tres matrices  $W^q$ ,  $W^k$ , y  $W^v$ , que serán aprendidas durante el entrenamiento
- \* La query de una palabra por la key de otras (query·key) es la atención que esa palabra presta a la otra de la frase
  - El vector  $z$  de cada palabra es la suma ponderada de todos los values de cada palabra ponderados por la atención
- \* Tenemos entonces como salida un vector por cada palabra que ahora ha incorporado al embedding el contexto de la frase
- \* Vídeo de DotCSV

# Attention Layers

Input	<b>Thinking</b>		<b>Machines</b>	
Embedding	$x_1$		$x_2$	
Queries	$q_1$		$q_2$	
Keys	$k_1$		$k_2$	
Values	$v_1$		$v_2$	
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ( $\sqrt{d_k}$ )	14		12	
Softmax	0.88		0.12	
Softmax X Value	$v_1$		$v_2$	
Sum	$z_1$		$z_2$	

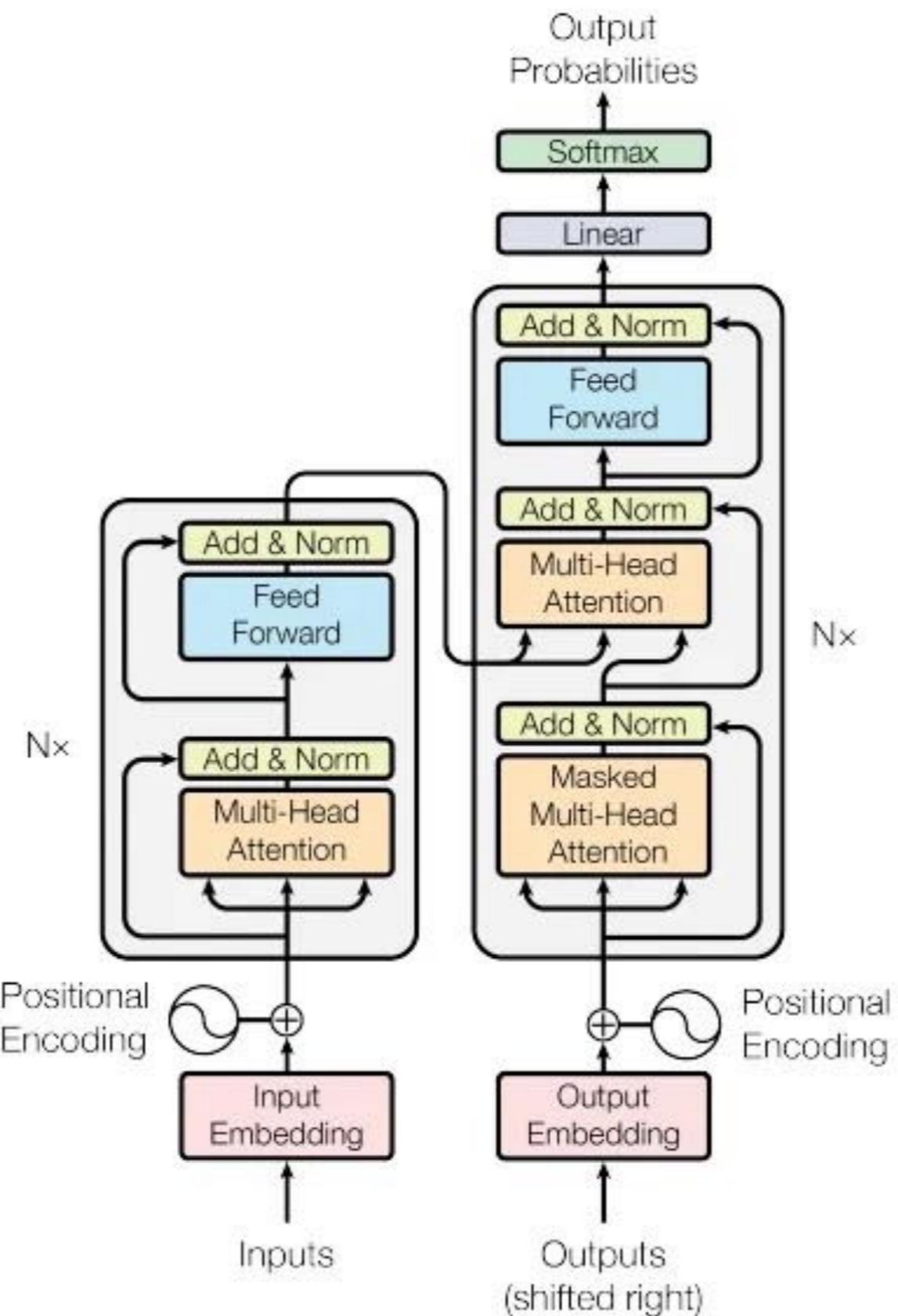
# Attention Layers



# Attention is all you need (Vaswani et al, 2017)

- \* Definió la arquitectura “transformer” que utiliza attention layers en lugar de redes recurrentes para PLN
- \* Fase encoder
  - Tiene una frase como entrada (la pregunta)
  - Se obtiene el embedding de cada palabra
  - Para codificar la posición se usa el positional embedding
    - ✖ Es un vector que se suma al embedding para indicar su posición en la frase
  - Usa un Multi Head Attention para destacar la posible importancia de varias temáticas, todos los heads se combinan en uno usando otra matriz de pesos; al final se obtiene un vector z por cada palabra de la frase
  - El resultado se pasa por una feedforward (una palabra cada vez) y se incorpora a la fase decoder
  - A la salida de cada fase se usa normalización en batch

# Attention is all you need (Vaswani et al, 2017)



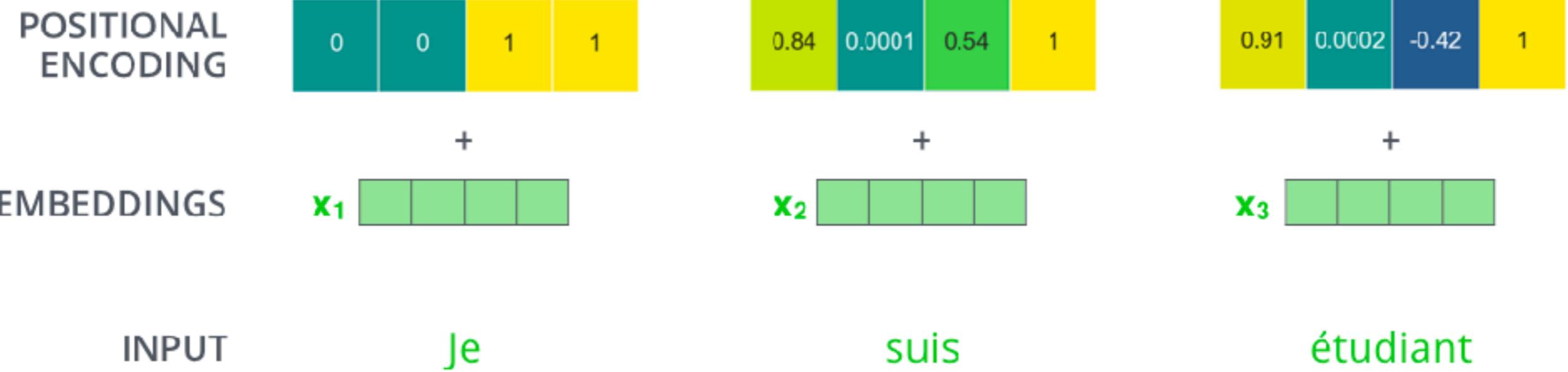
# Positional encoding

		CODIFICACIÓN ENTERO	CODIFICACIÓN BINARIA
pos.01	El	1 1 1 1 ... 1 1 1 1	0 0 0 0 ... 0 0 0 1
pos.02	pangolín	2 2 2 2 ... 2 2 2 2	0 0 0 0 ... 0 0 1 0
pos.03	es	3 3 3 3 ... 3 3 3 3	0 0 0 0 ... 0 0 1 1
pos.04	tu	4 4 4 4 ... 4 4 4 4	0 0 0 0 ... 0 1 0 0
pos.05	nuevo	5 5 5 5 ... 5 5 5 5	0 0 0 0 ... 0 1 0 1
pos.06	dios	6 6 6 6 ... 6 6 6 6	0 0 0 0 ... 0 1 1 0

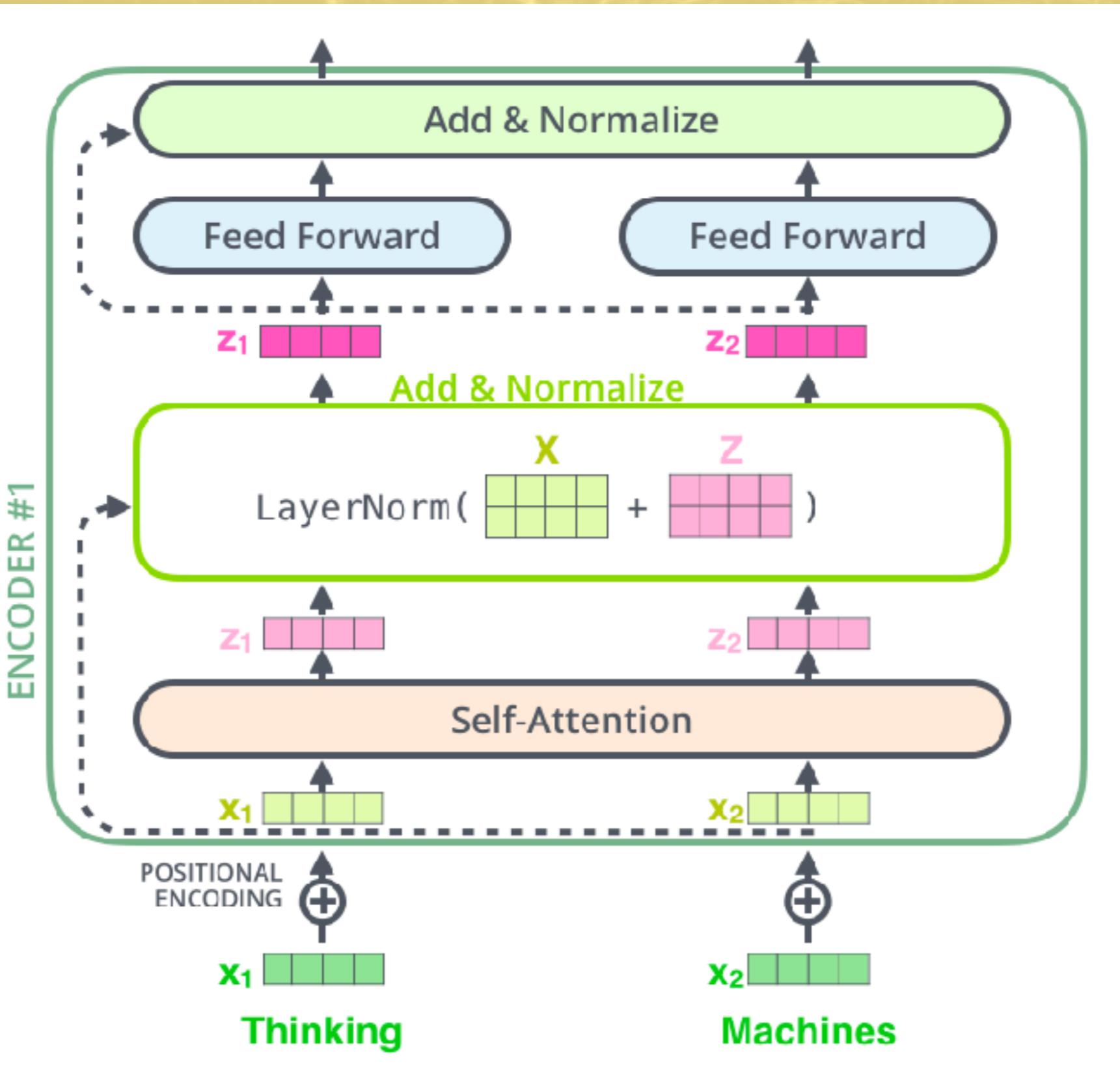
$i=0 \quad i=d$

$\text{seno}(\text{pos} / i)$

# Positional encoding



# Capas residuales



# Attention is all you need (Vaswani et al, 2017)

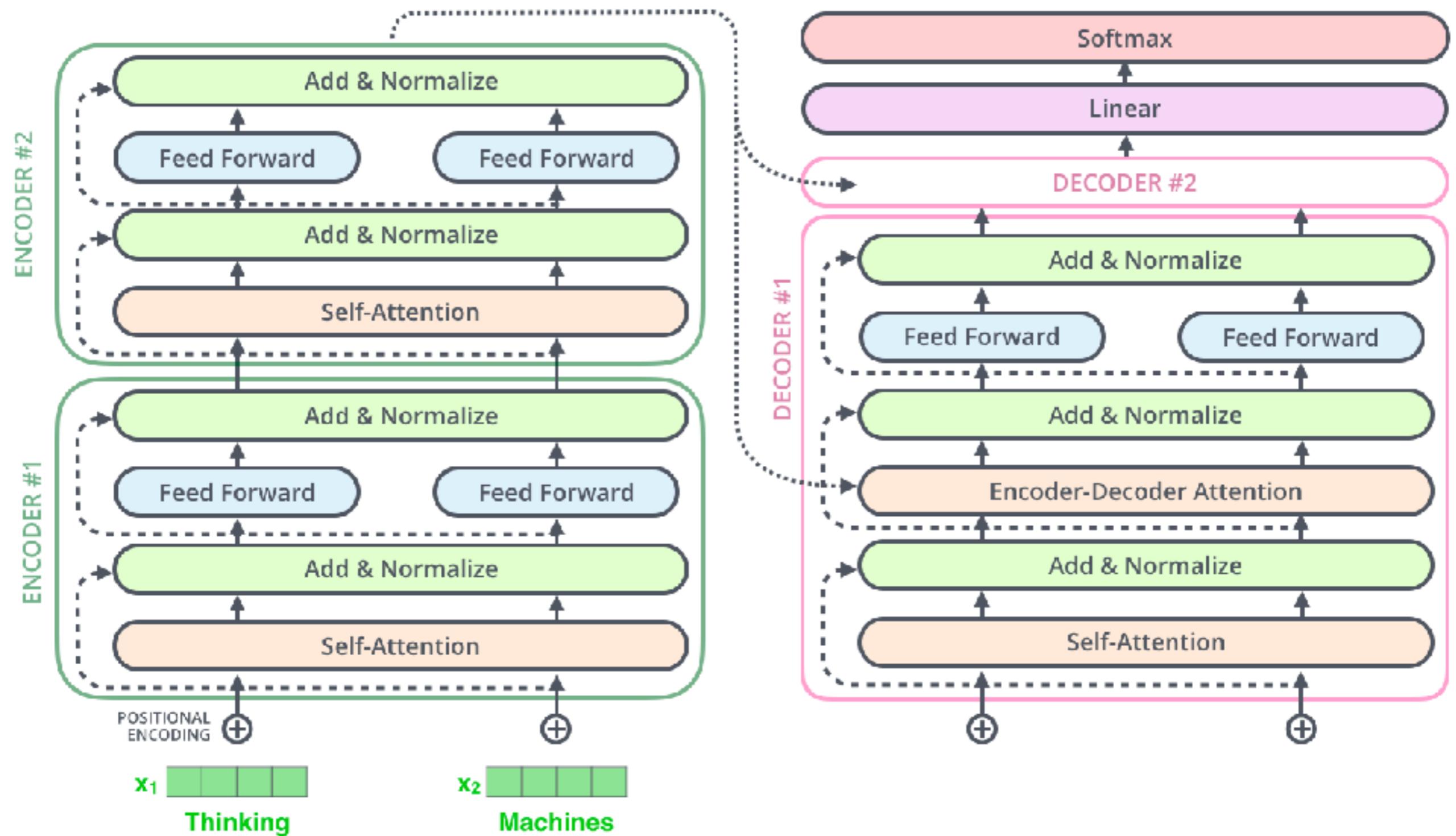
## \* Fase decoder

- Recibe una palabra como entrada (inicialmente ninguna)
- Pasa por un masked attention block que deshabilita las palabras que aún no han llegado
  - ✖ Las queries vienen de la capa anterior
  - ✖ Las keys y los values de la fase encoder
- Luego por un attention block que recibe el procesamiento del encoder más la salida del masked attention block
  - ✖ Las queries, values y keys se obtienen igual que en la capa anterior
- La salida pasa por una feedforward y luego por una linear y una softmax, que la siguiente palabra a producir
- Esta palabra se realimenta al decoder para producir la siguiente, y así hasta que se produzca el término especial “end-of-sentence”

## \* Muy buen artículo explicando todo paso a paso

- <https://jalammar.github.io/illustrated-transformer/>

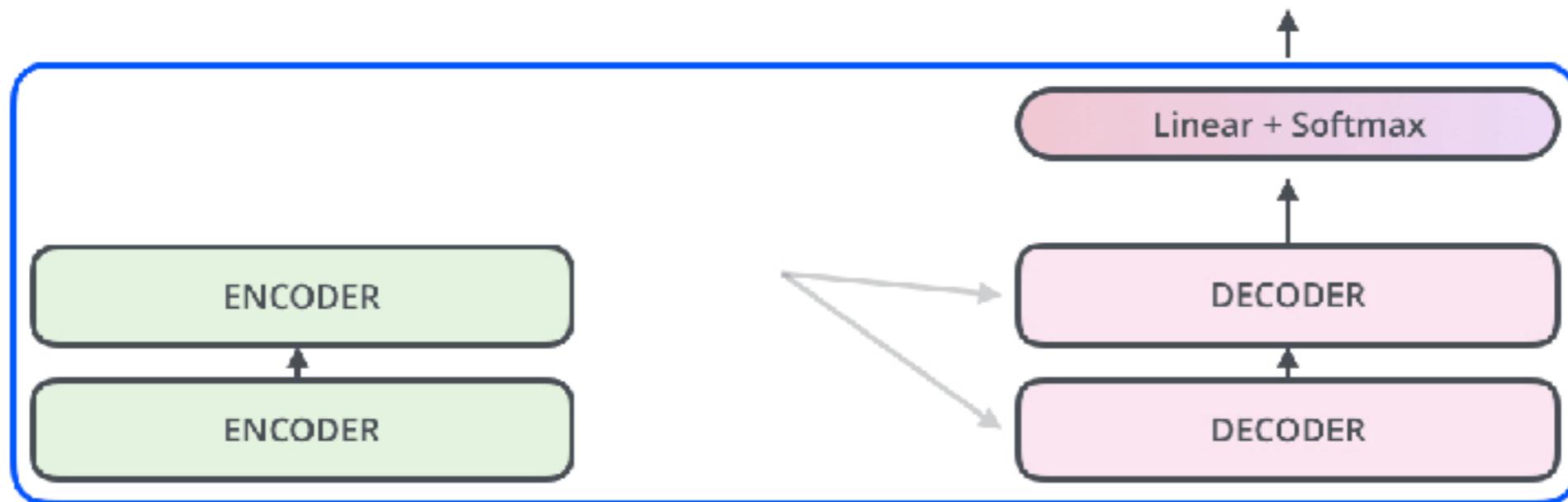
# Envío de La información al decoder



# Decoder

Decoding time step: 1 2 3 4 5 6

OUTPUT



EMBEDDING  
WITH TIME  
SIGNAL



EMBEDDINGS

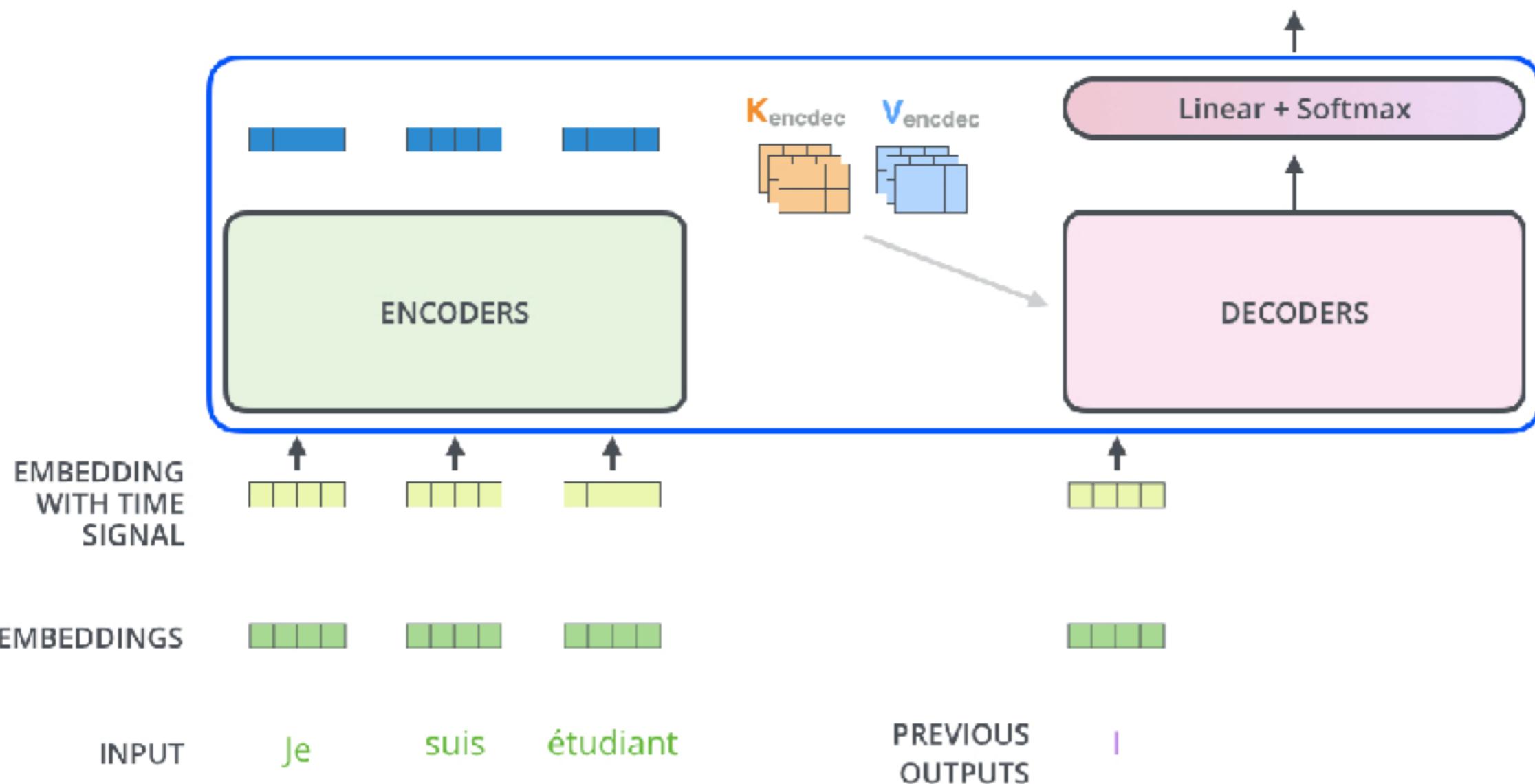


INPUT      Je      suis      étudiant

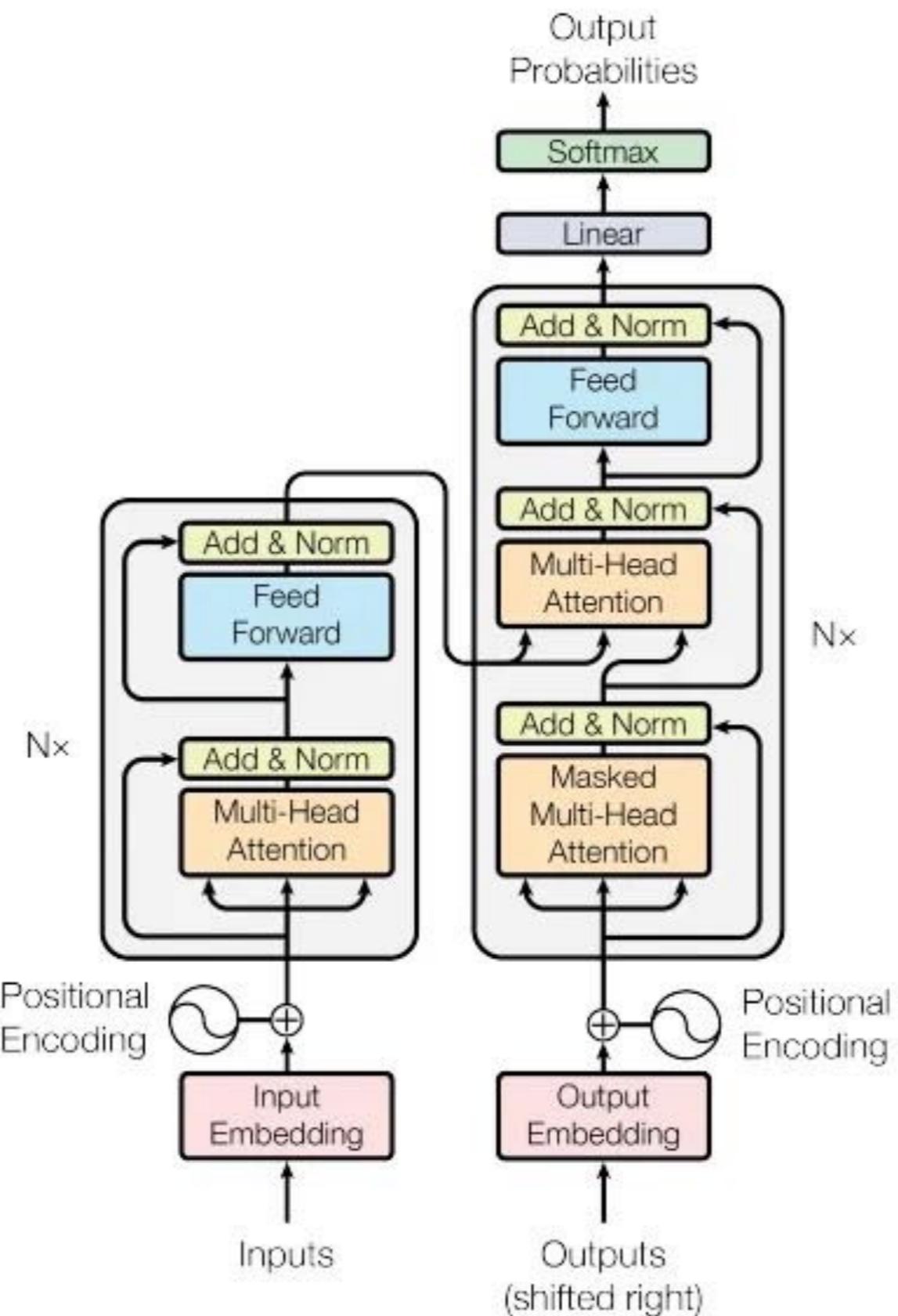
# Decoder

Decoding time step: 1 2 3 4 5 6

OUTPUT |



# Attention is all you need (Vaswani et al, 2017)



# Entrenamiento del transformer

## \* Depende de la tarea

- Traducción
  - ✖ Estrada del encoder: la frase en un idioma
  - ✖ Entrada del decoder: cada porción de la frase respuesta
  - ✖ Salida del decoder: la siguiente palabra a producir
- ChatGPT (contestado por él mismo)
  - ✖ En el caso de ChatGPT, la entrada generalmente es un fragmento de texto que representa la conversación previa del usuario con el modelo, mientras que la salida es la respuesta generada por el modelo. Por ejemplo, si un usuario le pregunta al modelo "¿Cómo estás hoy?", el fragmento de texto de entrada sería "Usuario: ¿Cómo estás hoy?" y la respuesta generada por el modelo sería la salida.

# Attention is all you need (Vaswani et al, 2017)

- \* Esta arquitectura transformer es la que está detrás de
  - BERT
  - GPT2, GPT3, ...
- \* Es decir, detrás de ChatGPT
- \* Modificaciones de esta arquitectura están detrás de
  - DALL-E
  - Stable Diffusion
  - MidJourney

# Un enfoque basado en capas

- \* Actualmente hay que pensar más en qué capacidades tiene cada tipo de capa que en redes completas, ya que las capas se han convertido en bloques de construcción para las redes
- \* Capas de procesamiento
  - Son capas esenciales que hacen algún tipo de trabajo en la red, es decir, transforman entradas en salidas
- \* Capas auxiliares
  - Hacen diversas tareas de adaptación, tipo cambiar la forma de un tensor de salida, aplicar dropout, etc

# Un enfoque basado en capas

- \* Tipos de capas de procesamiento
  - Dense (Full Connected): realizan tareas de procesamiento para clasificar u obtener valores de salida de la red
  - Conv2D: extraen características en imágenes (también 1D y 3D)
  - Conv2DTranspose: reconstruyen una imagen a partir de sus características
  - LSTM, GRU: capa recurrente (la salida vuelve a entrar), se usan cuando hay información secuencial (la salida depende de entradas anteriores)
  - Attention y MultiHeadAttention: dan diferente importancia a cada entrada según cada ejemplo concreto
- \* Cada capa puede tener las siguientes activaciones: linear, sigmoid, tanh, softmax, relu, leakyReLU

# Un enfoque basado en capas

## \* Tipos de capas auxiliares

- Input: sólo sirve para poner la entrada a la red
- MaxPool: realizan operación de maxpool
- Dropout: regulariza usando dropout
- Flatten, Reshape: cambian las dimensiones del tensor, manteniendo los valores
- LayerNormalization, BatchNormalization: normalizan los valores entre dos capas
- Concatenate: combina las salidas de varias capas

## \* Lista completa en

- <https://keras.io/api/layers/>

# Optimizadores

- \* Una vez “montada” la red, se entrena con alguno de los optimizador disponibles
  - Esto es posible porque cada capa sabe indicarle al optimizador cuál es su derivada para que calcule el gradiente
- \* Optimizadores
  - SGD (Stochastic Gradient Descent): es el que hemos visto como descenso del gradiente, aunque en Keras incorpora momento y se realiza por mini-batch
  - RMSprop (Retropropagación del Root Mean Square Error): ajusta la tasa de aprendizaje de cada peso en función de los gradientes previos; se calcula el gradiente y se multiplica por esta tasa
  - Adam (Adaptive Moment Estimation): combinación de RMSprop y momento
- \* Más en <https://keras.io/api/optimizers/>

# Recursos

- \* Buen artículo introductorio a las CNN
  - <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
- \* Demo online de CNN para el reconocimiento de dígitos
  - <http://scs.ryerson.ca/~aharley/vis/conv/flat.html>
- \* Excelente curso de CNN
  - <http://cs231n.github.io/convolutional-networks/>
- \* Explicación muy clara de Transformers
  - <https://jalammar.github.io/illustrated-transformer/>

Algunos resultados clásicos en DL

# MNIST

- \* Identificación de dígitos manuscritos
- \* MNIST es un juego de prueba estándar para algoritmos de aprendizaje
  - El objetivo es identificar dígitos desconocidos con el máximo porcentaje de aciertos
- \* Tutorial en
  - <https://www.tensorflow.org/versions/r0.9/tutorials/mnist/beginners/index.html>
  - <https://www.tensorflow.org/versions/r0.9/tutorials/mnist/pros/index.html>
- \* Otros juegos estándar de imágenes y sus resultados
  - [http://rodrigob.github.io/are\\_we\\_there\\_yet/build\\_classification\\_datasets\\_results.html#494c5356524332303132207461736b2031](http://rodrigob.github.io/are_we_there_yet/build_classification_datasets_results.html#494c5356524332303132207461736b2031)

# MNIST

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

# CIFAR-10/100, ImageNet-1000

## \* CIFAR-10

- 60000 imágenes de 32x32 píxeles en 10 clases, accuracy 97,28%

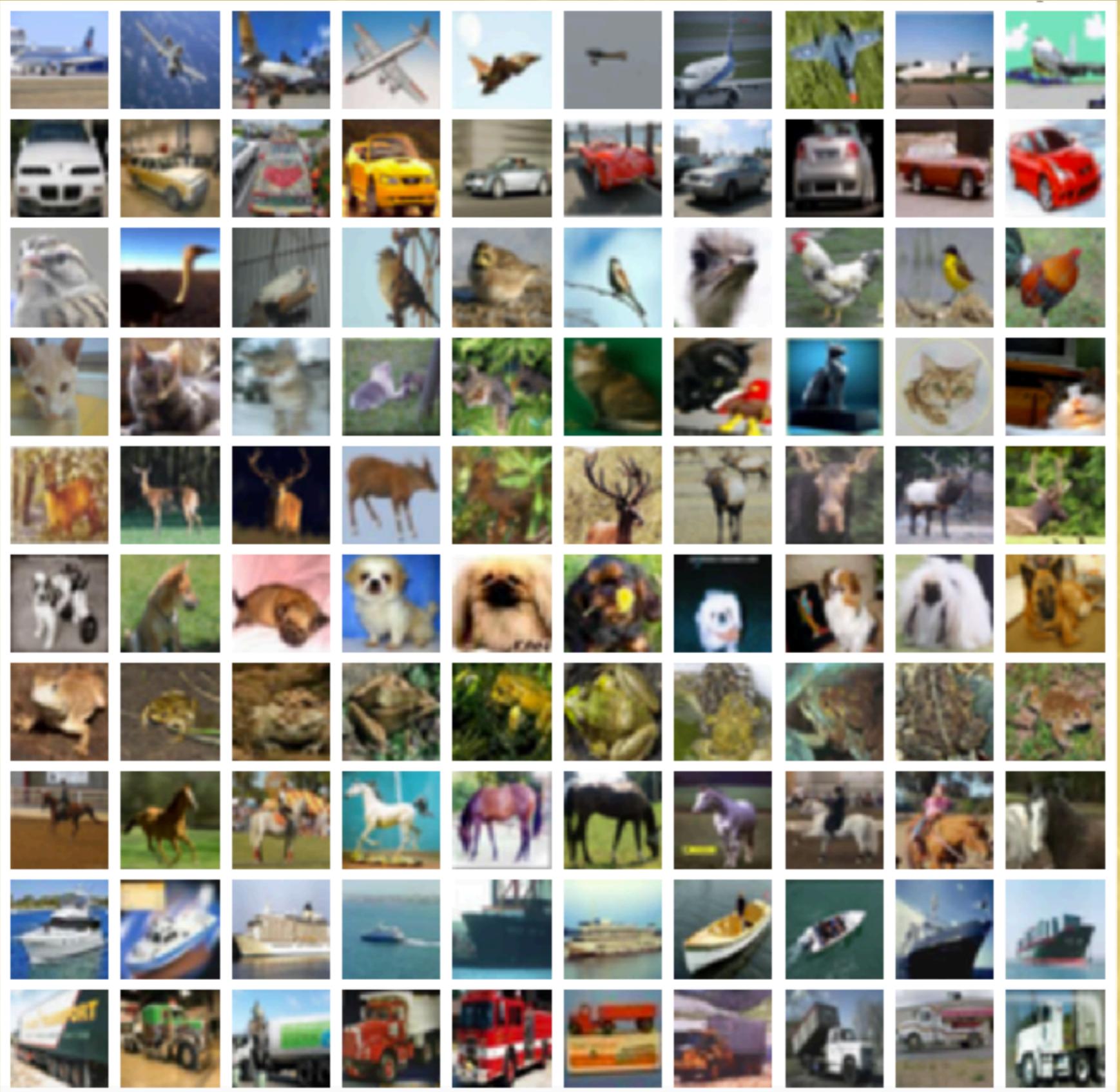
## \* CIFAR-100

- 60000 imágenes de 32x32 píxeles en 100 clases y 20 superclases, accuracy 84,15%

## \* ImageNet-1000

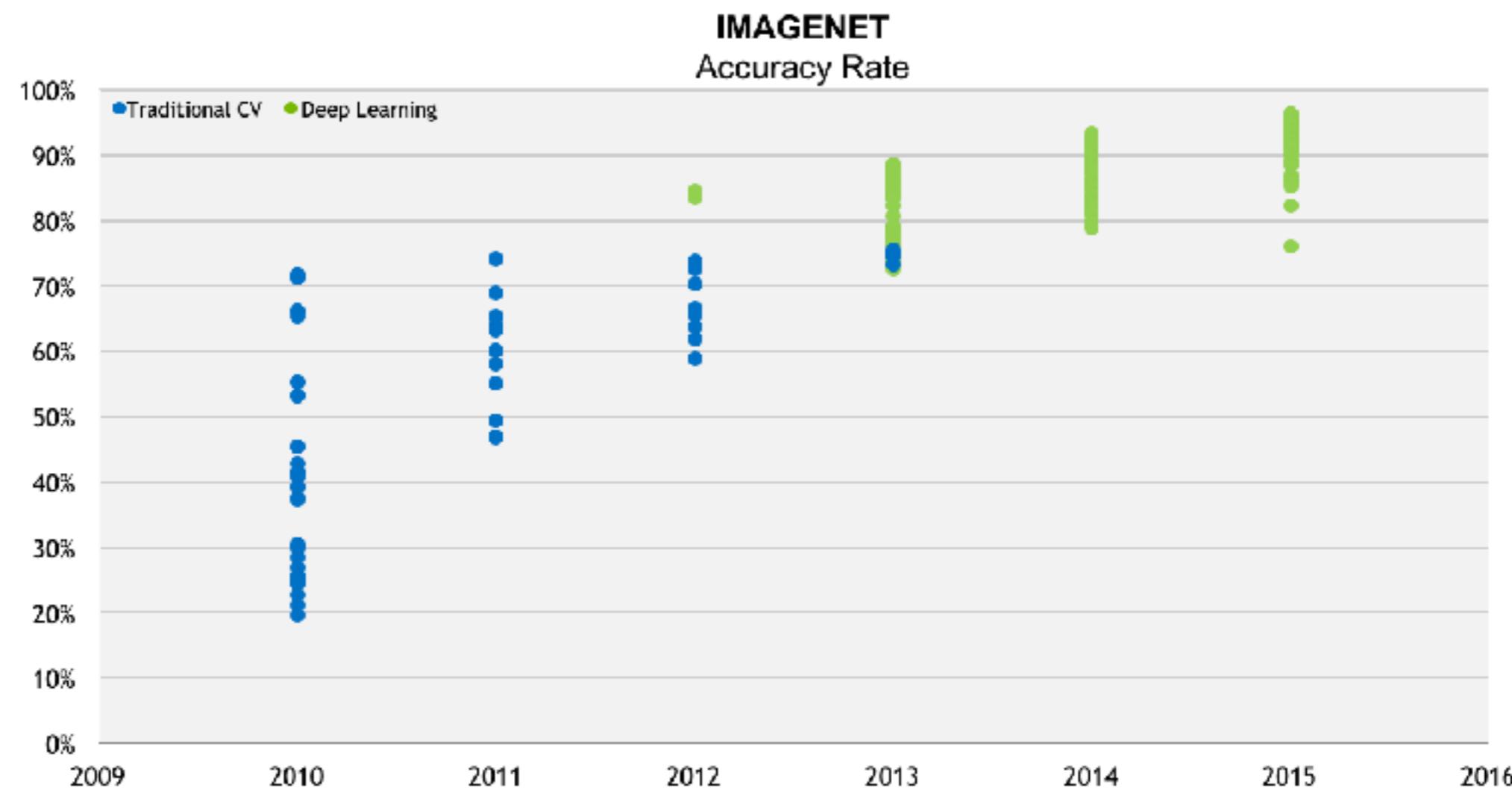
- 14 millones de imágenes en 1000 clases
- AlexNet (2012): (62,5% top-1, 83% top-5)
- VGG16 (2015): 92.7% top-5
- Inception-v3: 78.8% top-1; 94.4% top-5
- Transfer Learning
  - En muchos proyectos se están usando modelos preentrenados con ImageNet-1000 para hacer luego un “fine-tunning” de las capas finales (<https://www.cs.toronto.edu/~frossard/post/vgg16/>)

# ImageNet-1000

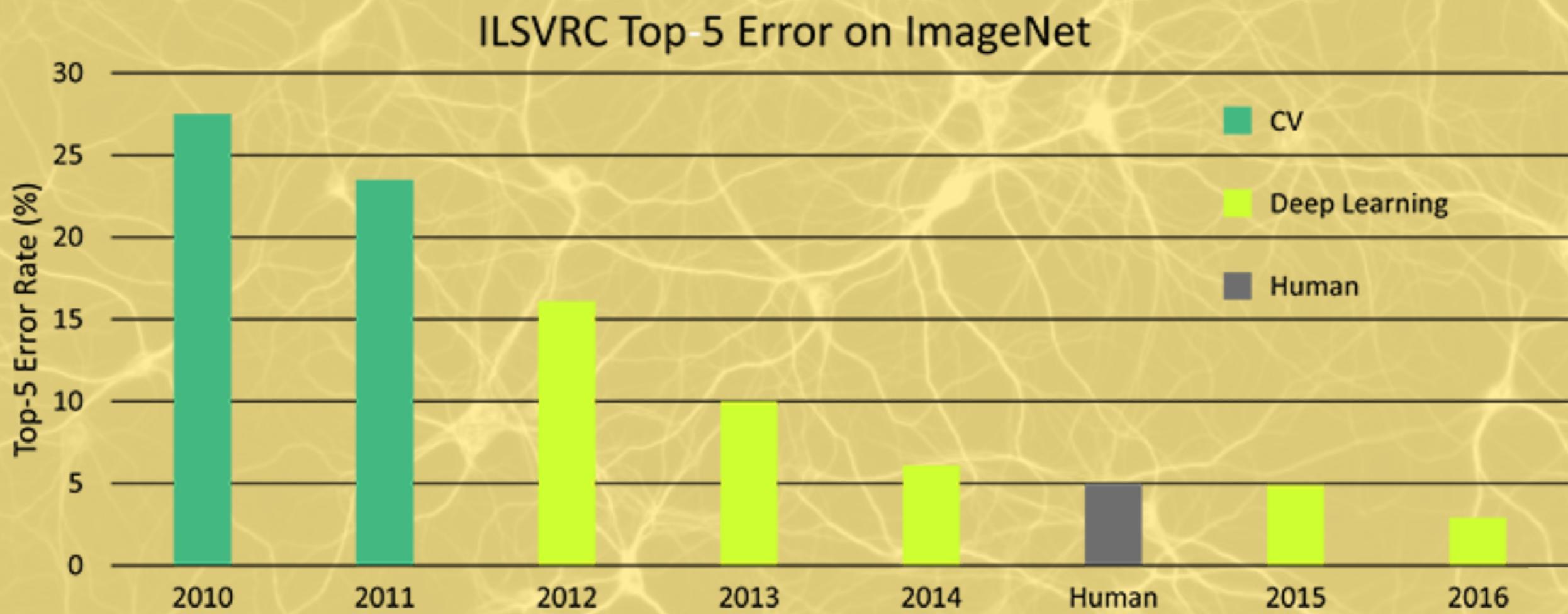


## DEEP LEARNING FOR VISUAL PERCEPTION

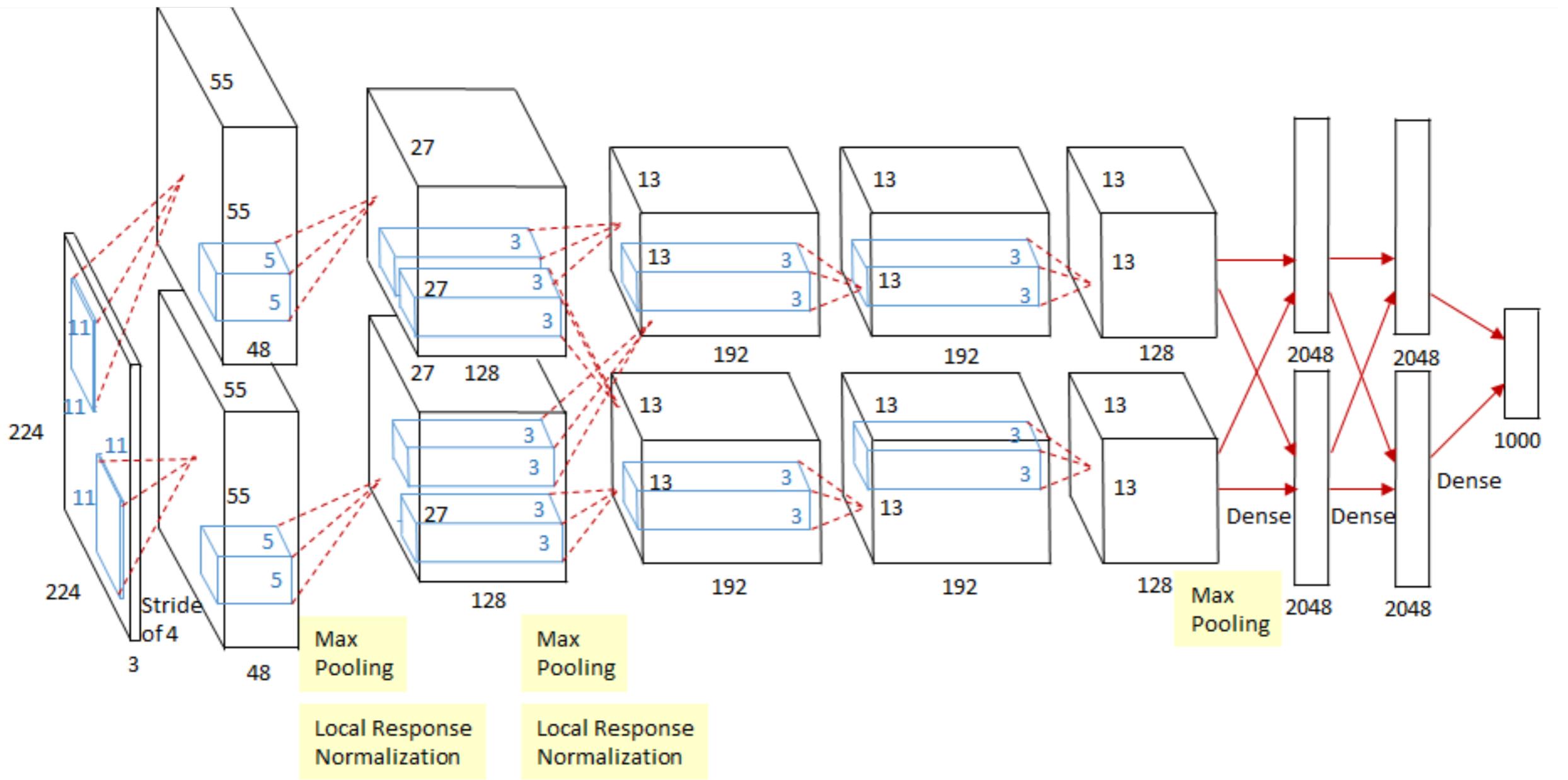
Going from strength to strength



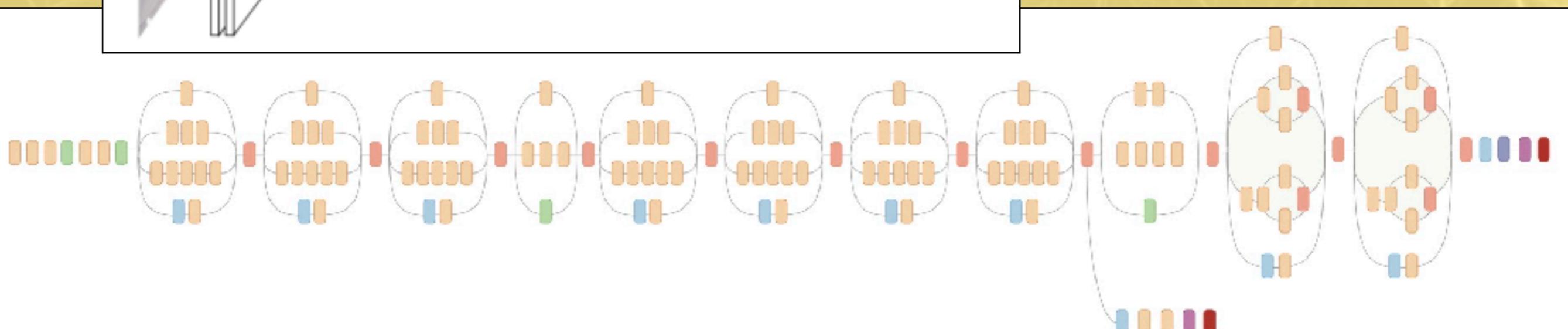
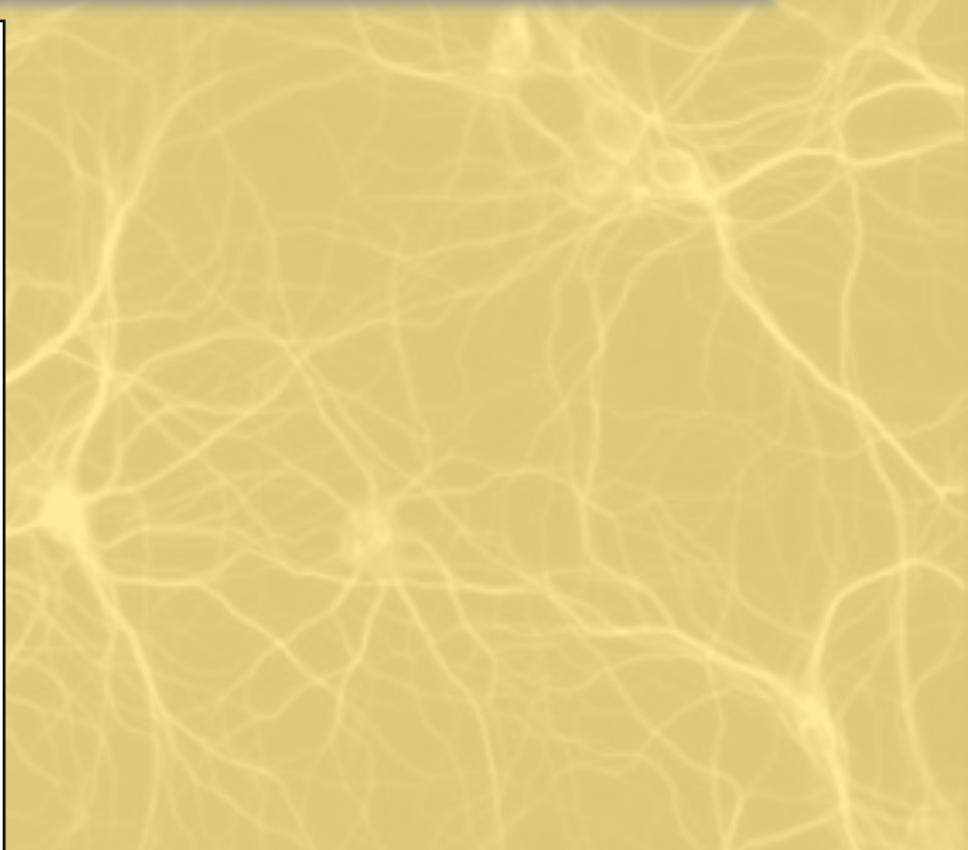
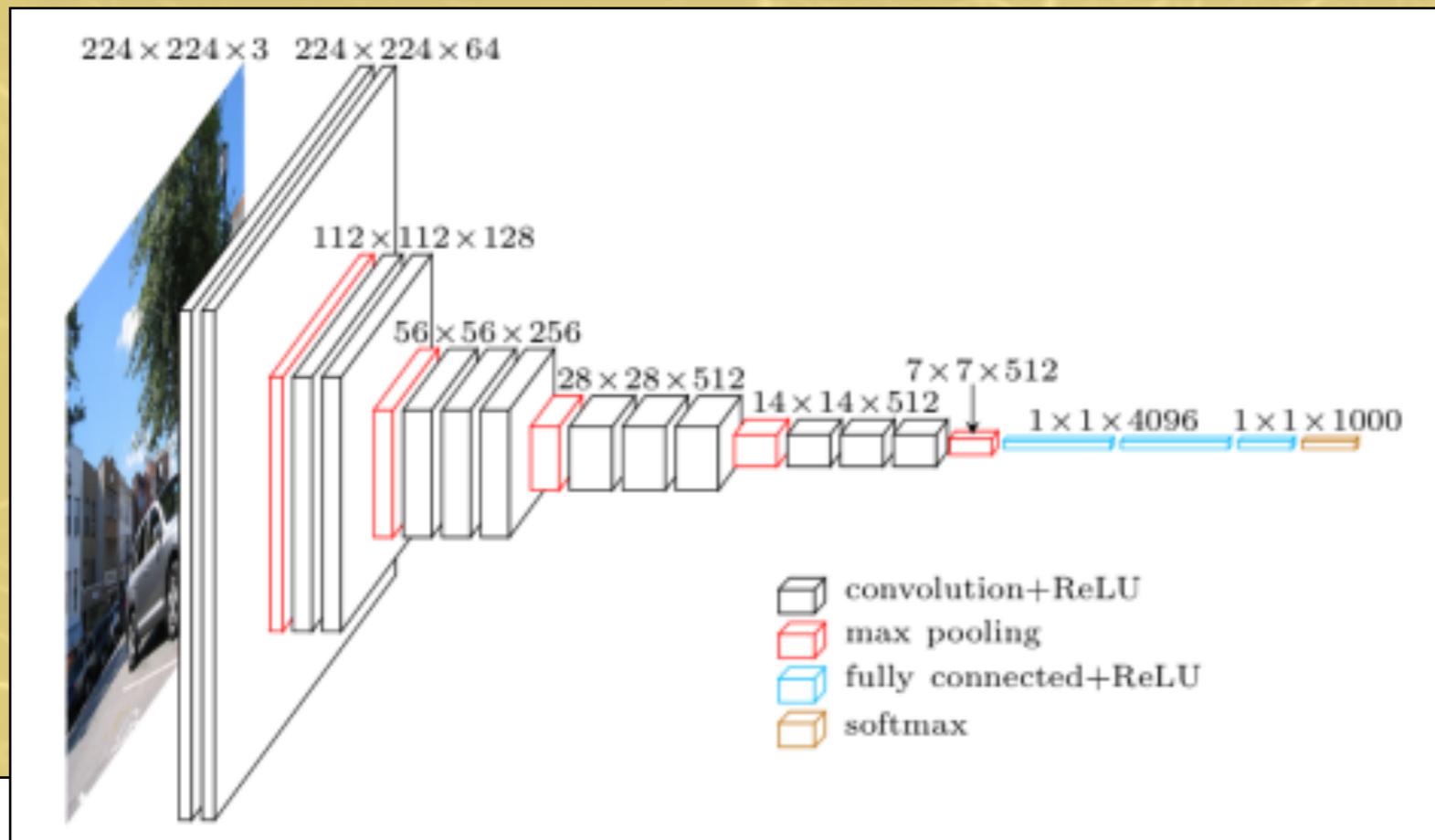
# ImageNet Large Scale Visual Recognition Competition (ILSVRC)



# Alexnet (2012)



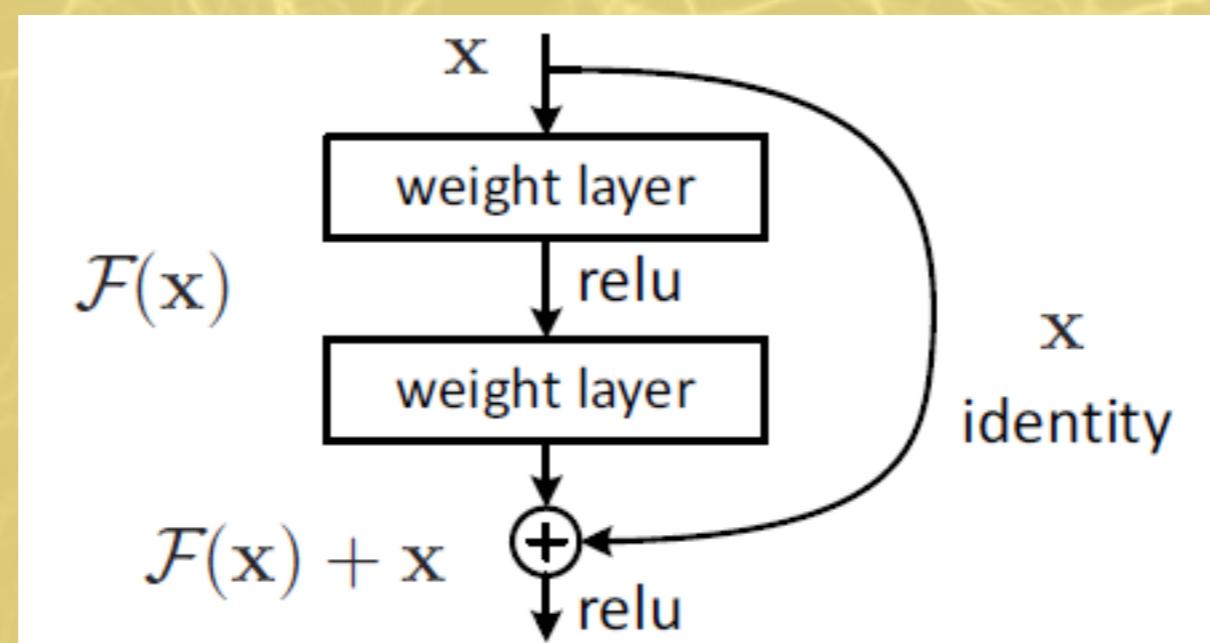
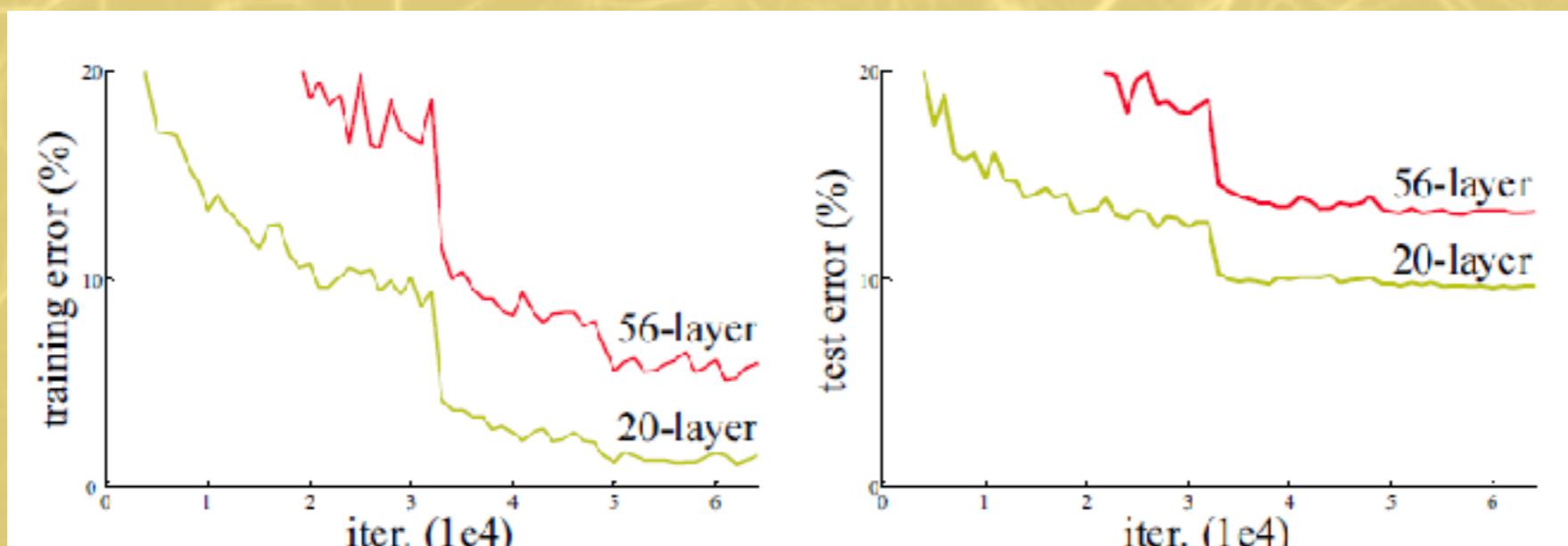
# VGG16 (2014), Inception-v3 (2015)



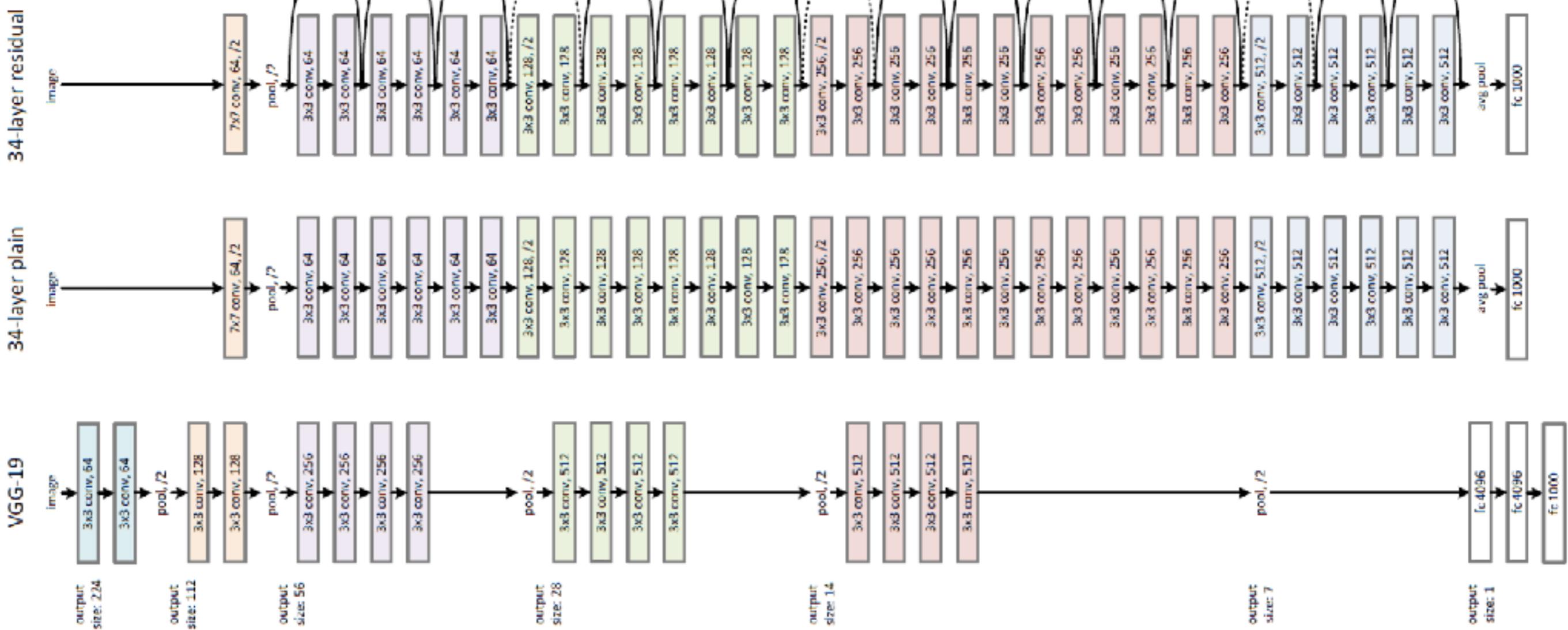
- Convolution
- AvgPool
- MaxPool
- Concat
- Dropout
- Fully connected
- Softmax

# Resnet (2015)

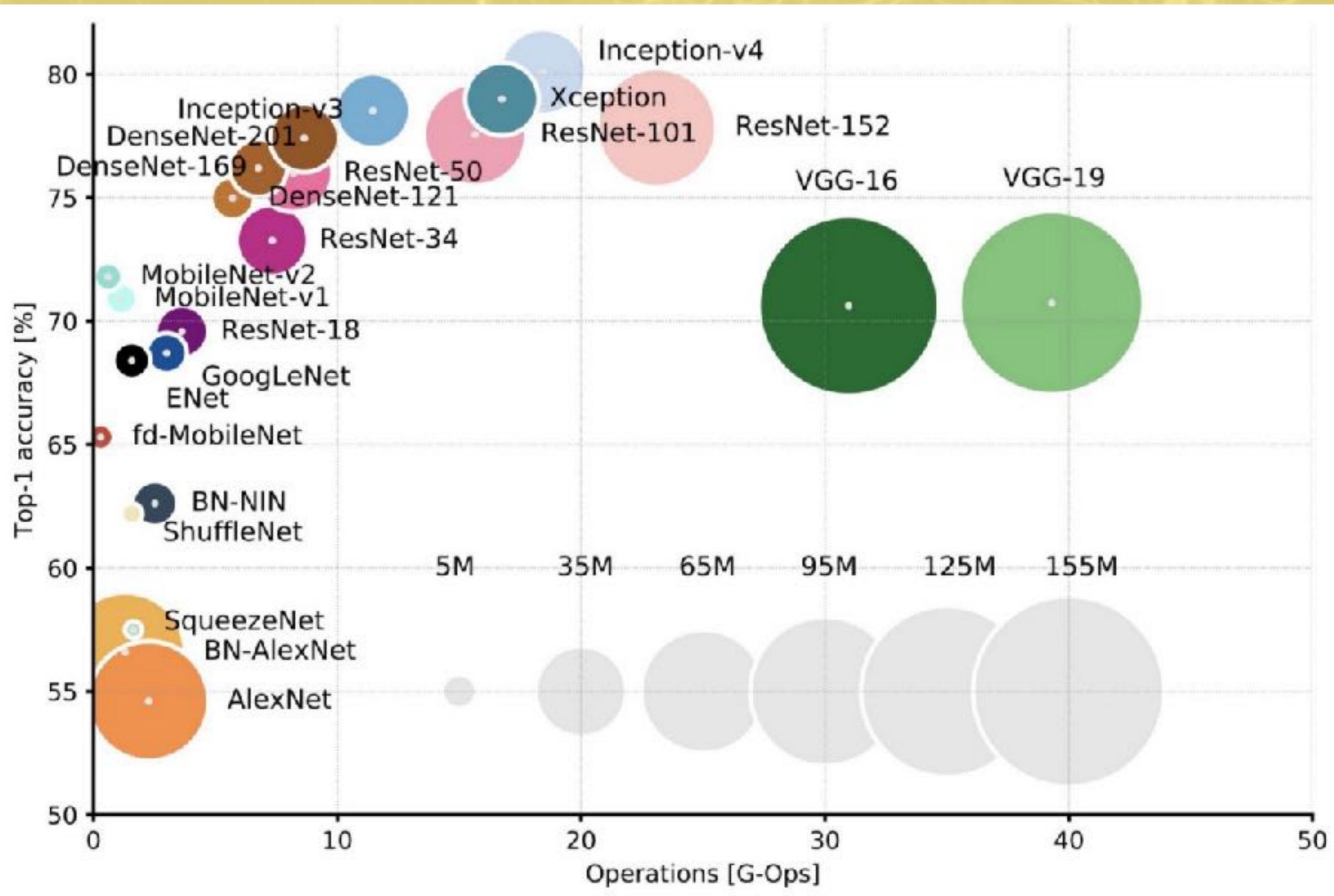
- \* Las entradas se un grupo de conexiones se suman a las salidas; esto permite que la retropropagación del gradiente funcione mejor. Si las dimensiones no coinciden se usa una matriz identidad para proyectar las entradas



# Resnet (2015)



# Evolución de arquitecturas



*A mostly complete chart of*

# Neural Networks

©2016 Fjodor van Veen - [asimovinstitute.org](http://asimovinstitute.org)

○ Backfed Input Cell

○ Input Cell

△ Noisy Input Cell

● Hidden Cell

○ Probabilistic Hidden Cell

△ Spiking Hidden Cell

● Output Cell

○ Match Input Output Cell

● Recurrent Cell

○ Memory Cell

△ Different Memory Cell

● Kernel

○ Convolution or Pool

Perceptron (P)



Feed Forward (FF)



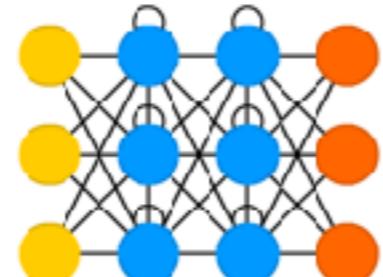
Radial Basis Network (RBF)



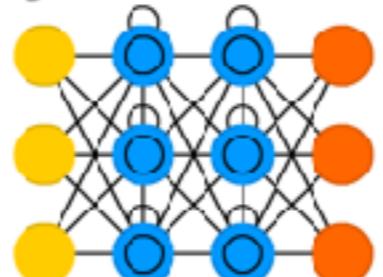
Deep Feed Forward (DFF)



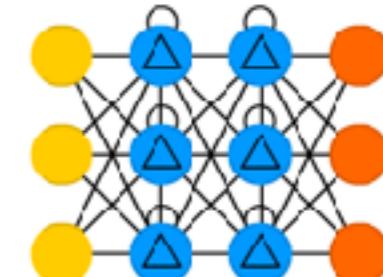
Recurrent Neural Network (RNN)



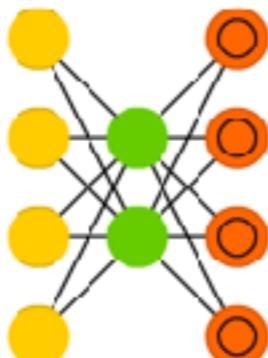
Long / Short Term Memory (LSTM)



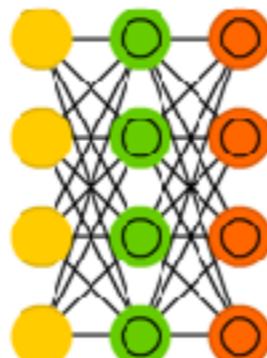
Gated Recurrent Unit (GRU)



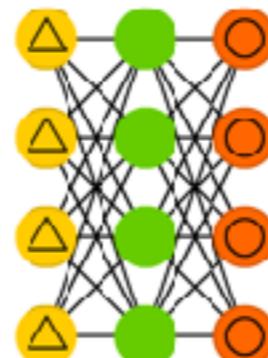
Auto Encoder (AE)



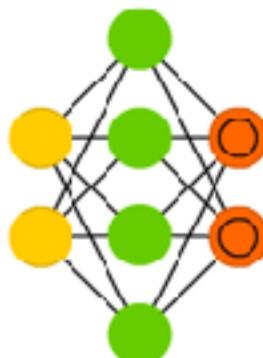
Variational AE (VAE)



Denoising AE (DAE)



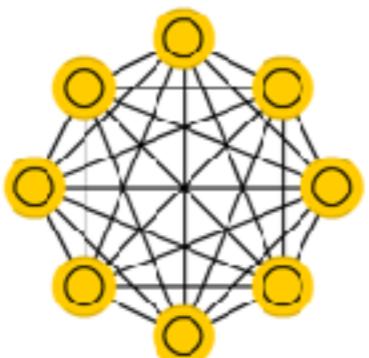
Sparse AE (SAE)



Markov Chain (MC)



Hopfield Network (HN)



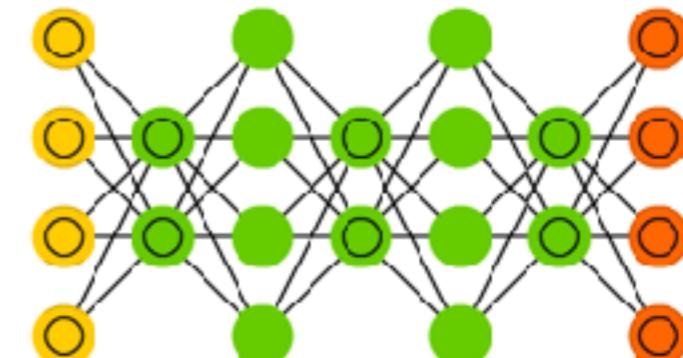
Boltzmann Machine (BM)



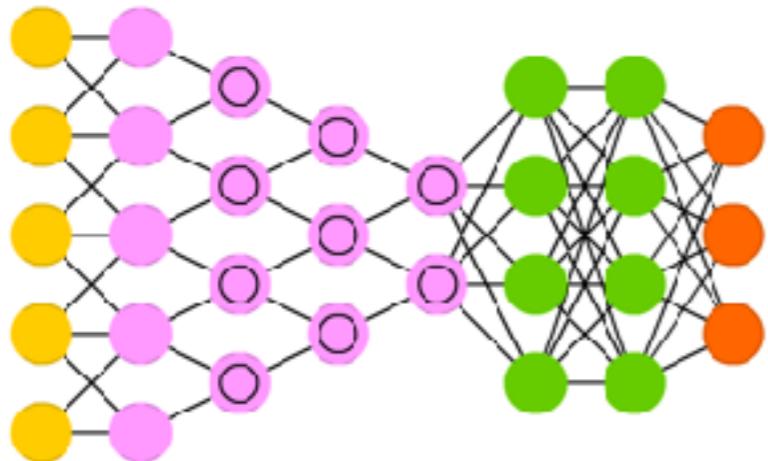
Restricted BM (RBM)



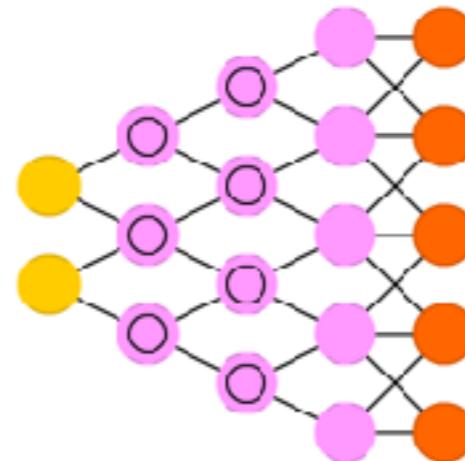
Deep Belief Network (DBN)



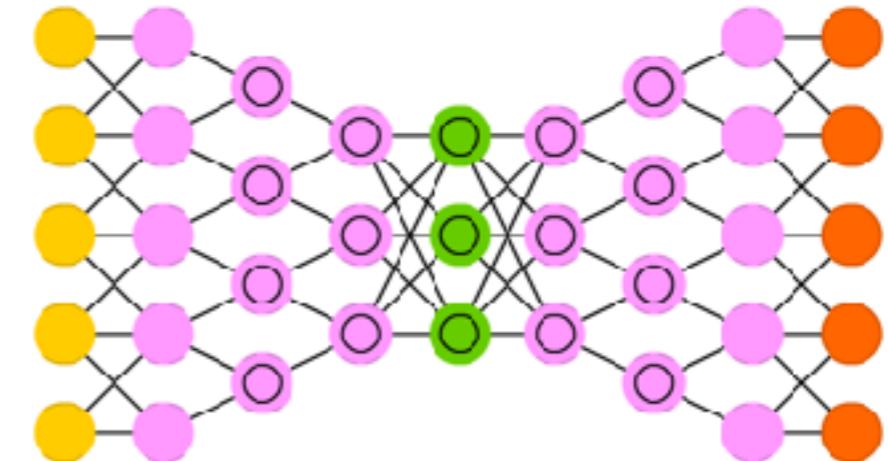
Deep Convolutional Network (DCN)



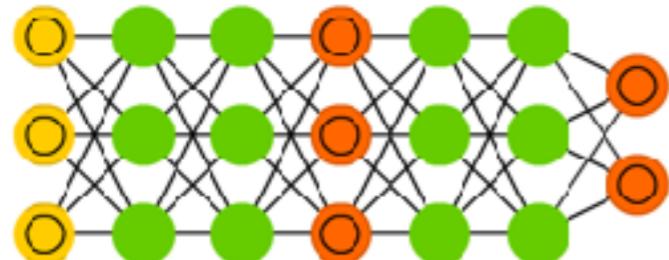
Deconvolutional Network (DN)



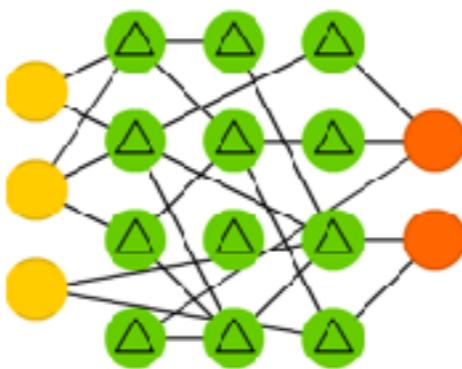
Deep Convolutional Inverse Graphics Network (DCIGN)



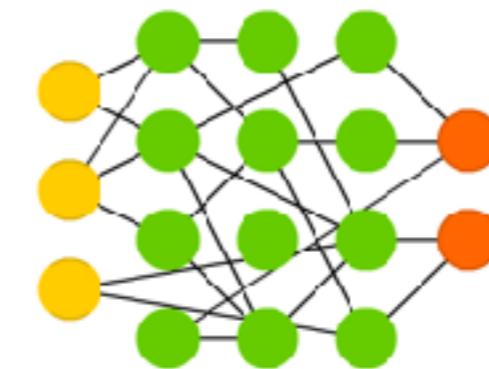
Generative Adversarial Network (GAN)



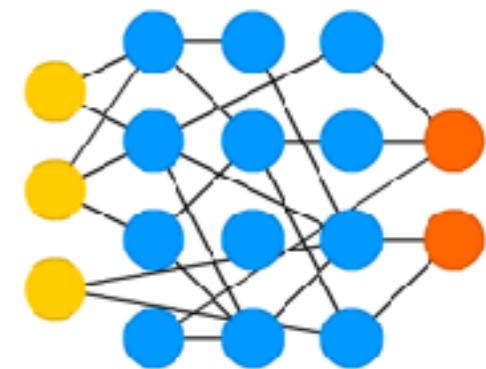
Liquid State Machine (LSM)



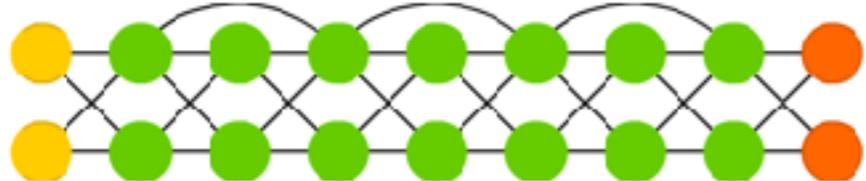
Extreme Learning Machine (ELM)



Echo State Network (ESN)



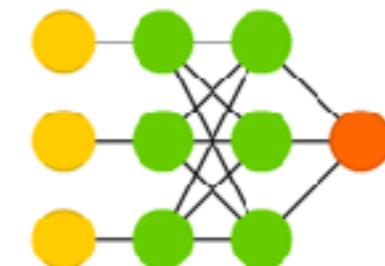
Deep Residual Network (DRN)



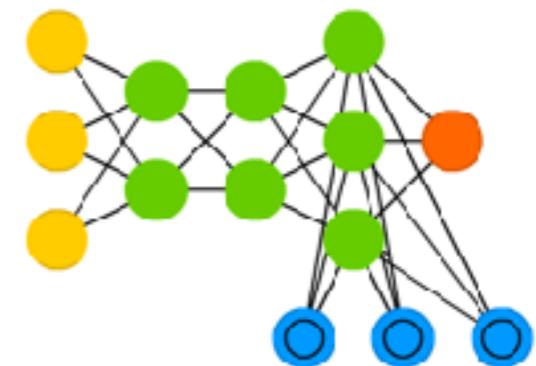
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



# Número de capas

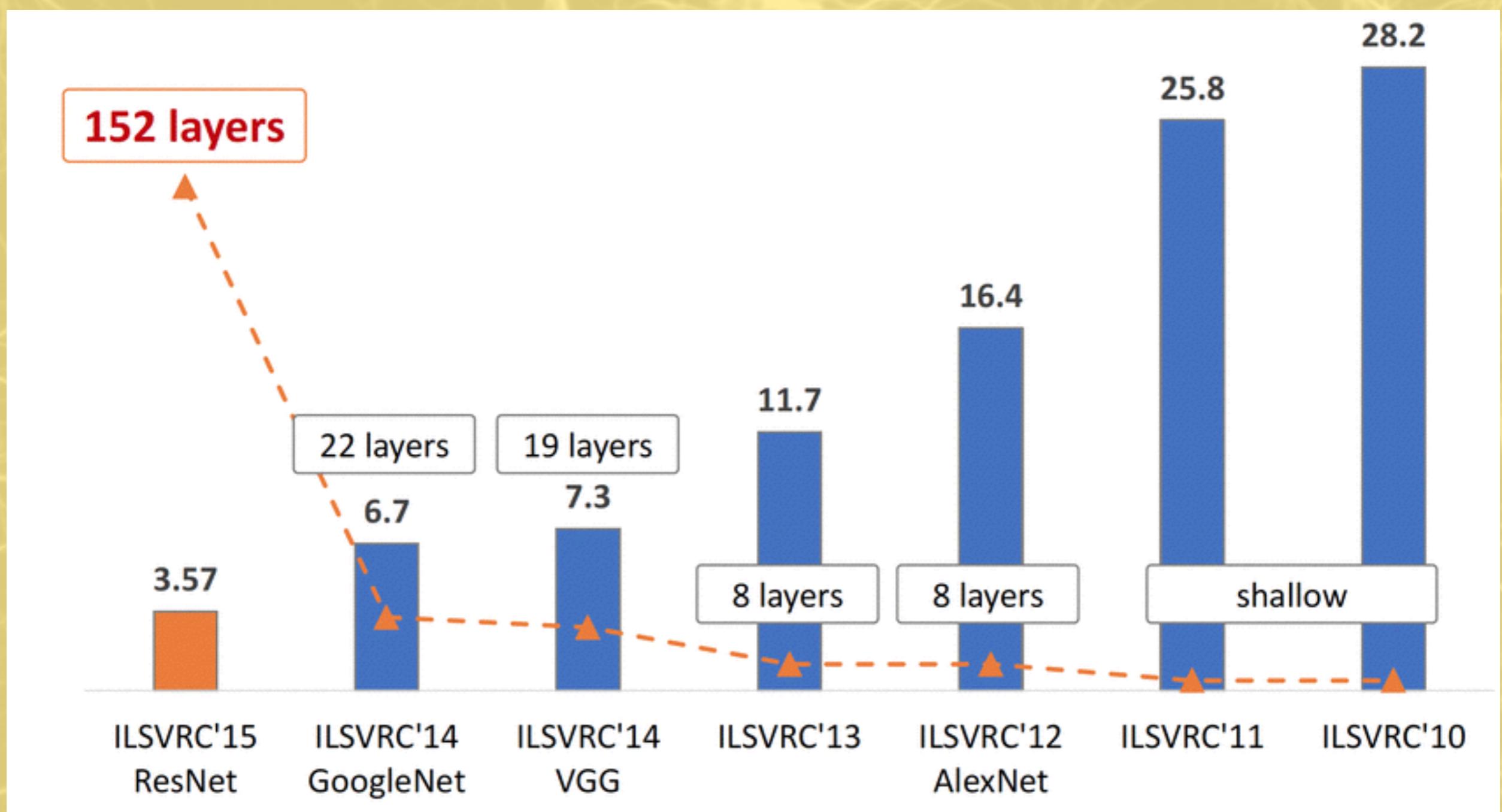
## \* ¿Qué es una capa en Deep Learning?

- En una red tradicional es fácil saber cuántas capas tiene: todas sin contar el vector de entradas
- En Deep Learning se suele considerar una capa (a partir de VGG)
  - ✖ Las CONV
  - ✖ Las FC
  - ✖ El grupo CONV + RELU se considera una única capa
  - ✖ No se consideran capas: maxpool, Softmax, Dropout, LRN, BN...
- En general, una capa es aquello que hay que optimizar modificando sus parámetros (pesos)

# Número de capas

	Image	Image	Image	Image	Number of Parameters (millions)	Top-5 Error Rate (%)
VGG-19	Conv3-64	Conv3-64	Conv3-64	Conv3-64	138	8.8
VGG-16	Conv3-512	Conv3-512	Conv3-512	Conv3-512	144	9.0
VGG-16 (Conv1)	Conv3-512	Conv3-512	Conv1-512	Max pool	134	9.4
VGG-13	Conv3-512	Conv3-512	Conv3-512	Max pool	133	9.9
VGG-11 (LRN)	Conv3-256	Conv3-256	Conv3-256	Max pool	133	10.5
VGG-11	Conv3-256	Conv3-256	Conv3-256	Max pool	133	10.4
VGG-4	Conv3-64	Conv3-64	Conv3-64	Max pool		
	Max pool	Max pool	Max pool	Max pool		

# Evolución del número de capas



# Aplicaciones Y ejemplos

# Neural-style

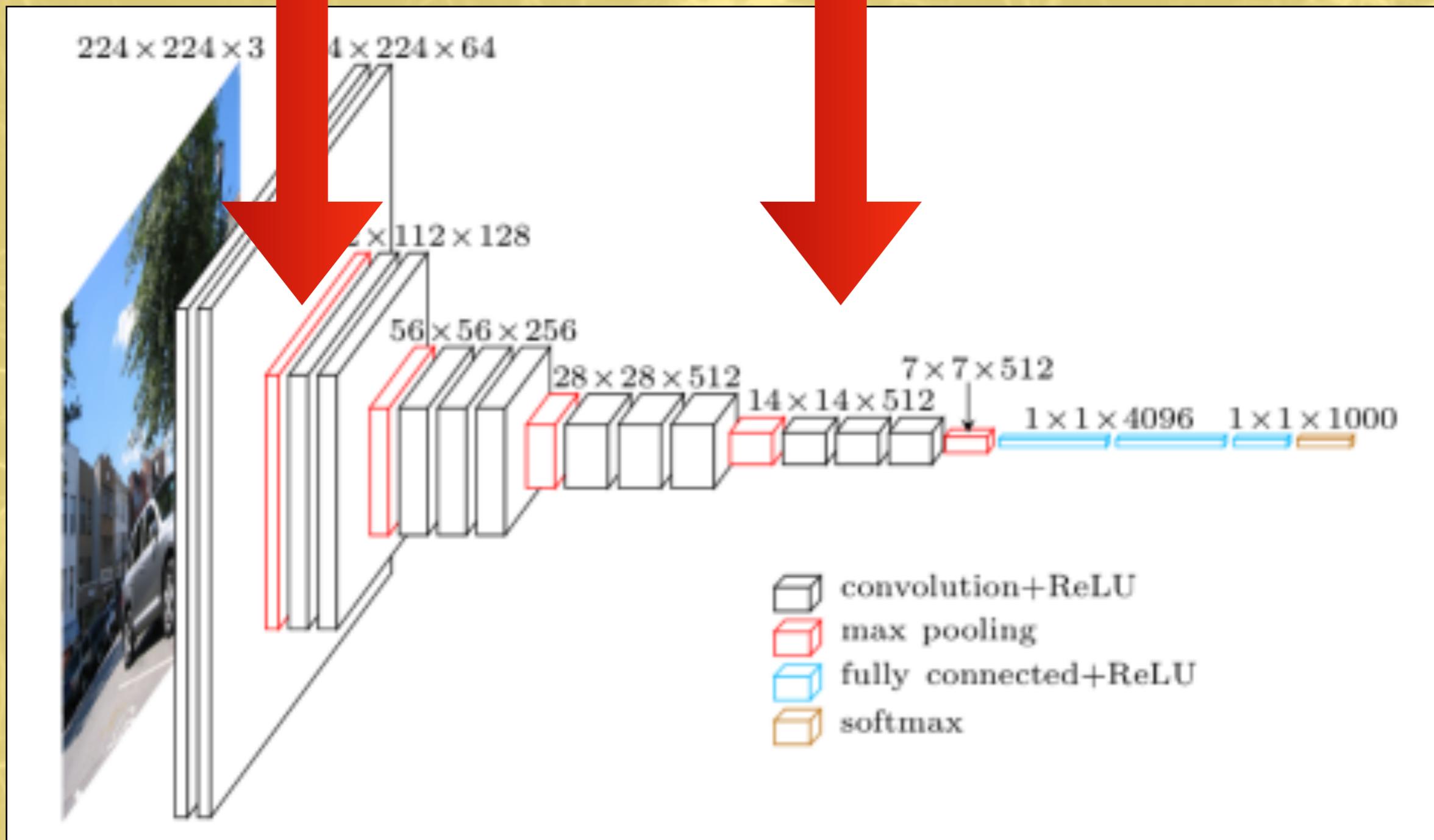
- \* Aprendizaje del estilo de un artista usando redes de convolución para poder aplicarlo a cualquier imagen
- \* Paper: Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). *A neural algorithm of artistic style*. arXiv preprint arXiv:1508.06576." <https://arxiv.org/pdf/1508.06576v2.pdf>
- \* Código para tensorflow
  - <https://github.com/anishathalye/neural-style>
- \* Tiempos de ejecución
  - Máquina con 24 núcleos Opteron a 1.4 GHz: 754m
  - Mac Pro (CPU 12 cores): 140m:55s (5 veces más rápido)
  - GPU Nvidia 980 Ti: 4m:09s (35 veces más rápido que el mac, 180 veces más rápido que el opteron)
- \* App online de Google: DeepDream.com, App móvil Prisma

# Neural-style

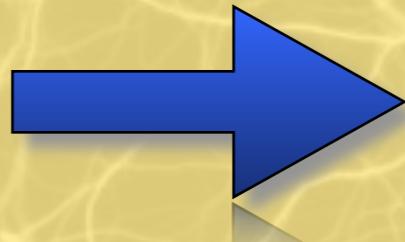
## \* Pasos para recrear una imagen

- Se usa una red entrenada previamente (por ejemplo una VGG-16)
- Seleccionar convolutional layer CS para representar el estilo (uno de los últimos)
- Selecciona convolutional layer CI para representar la imagen final (uno de los iniciales o intermedios)
- Se propaga la imagen con el estilo a transferir para obtener las activaciones de CS
- Se propaga la imagen a modificar para obtener las activaciones de CI
- Se genera una imagen aleatoria como entrada, esta imagen produce al ser propagada un CS' y un CI'
- Se realiza un descenso del gradiente en la imagen de entrada usando como error la diferencia entre CS y CS' más la diferencia entre CI y CI', con eso se busca que la imagen se parezca a CI y el estilo a CS

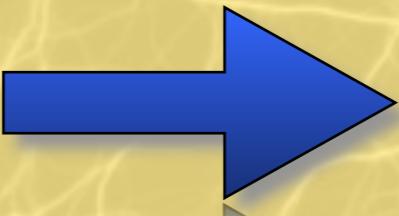
# Style Transfer



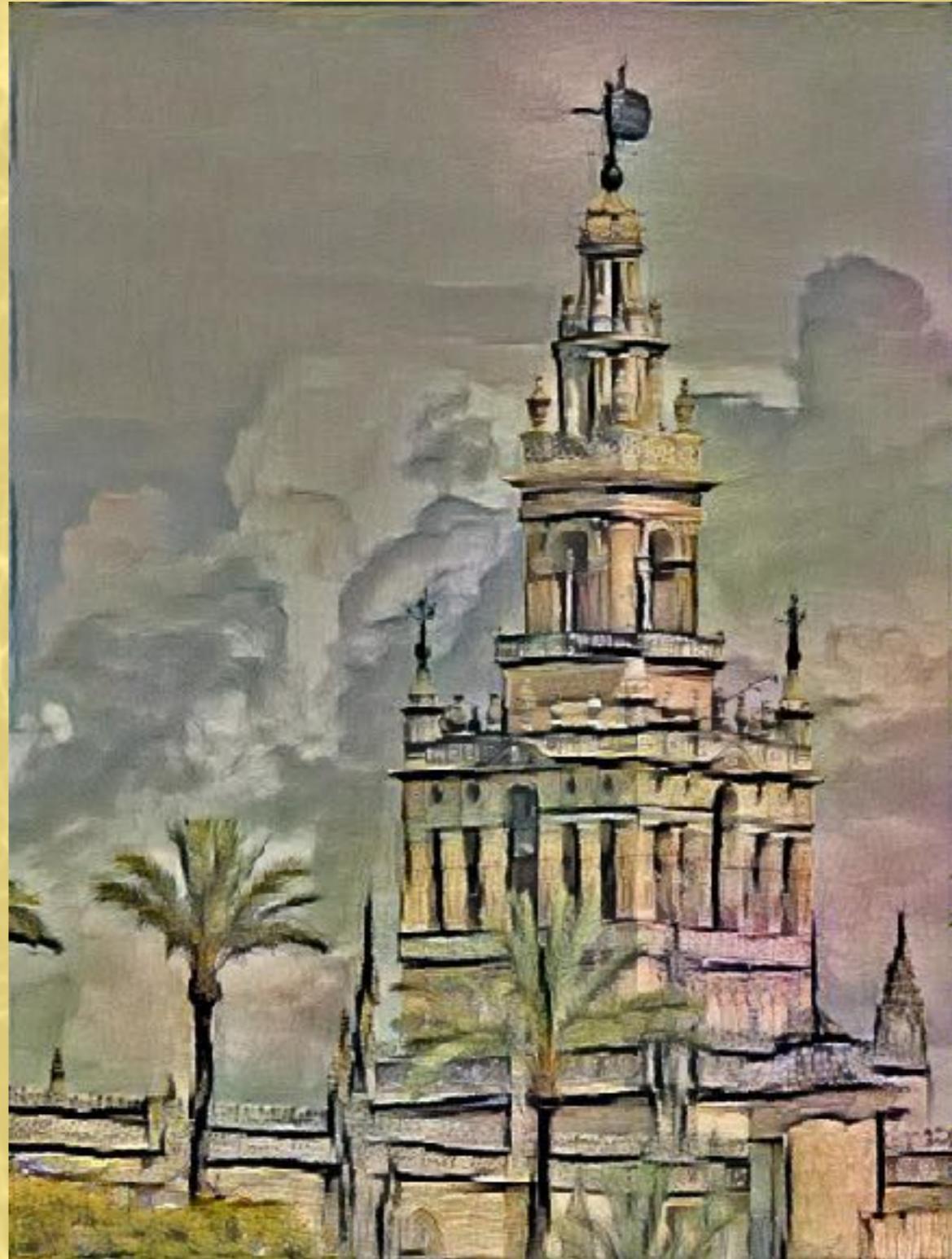
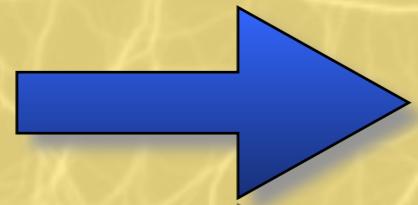
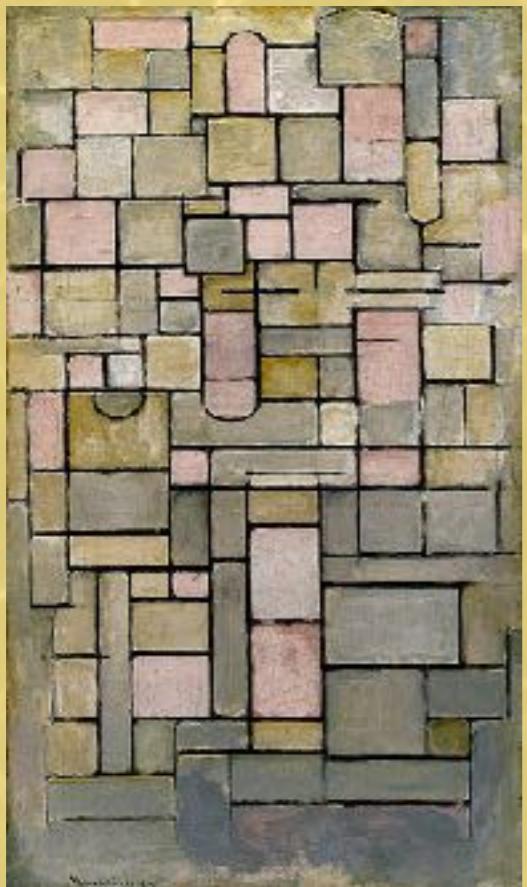
# Resultados



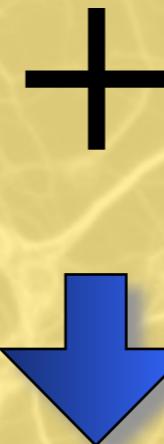
# Resultados



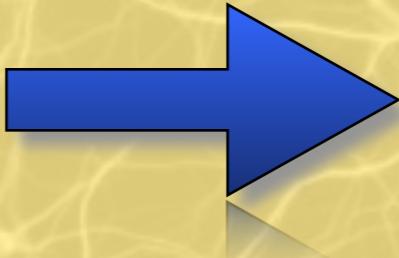
# Resultados



# Resultados

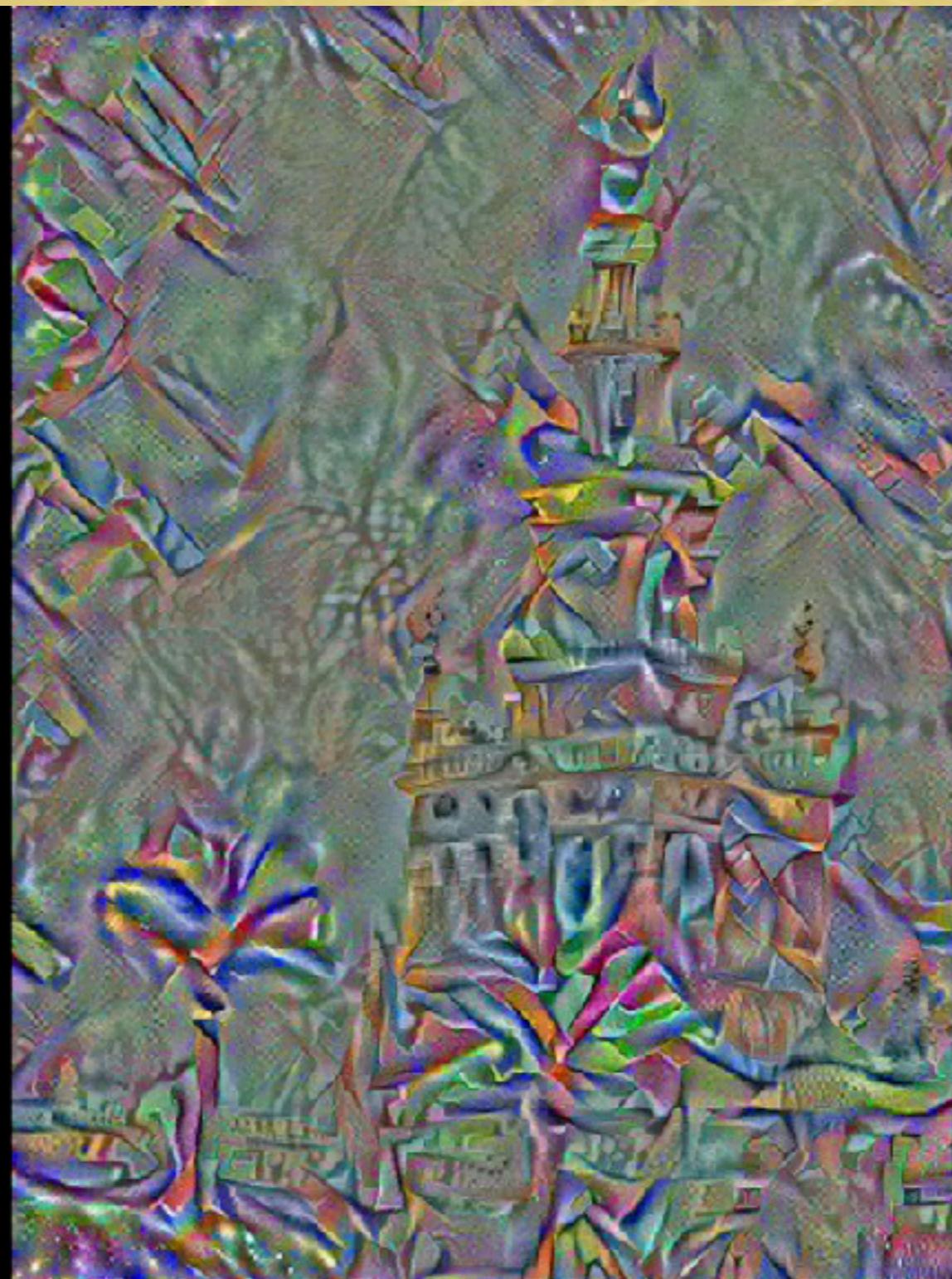


¿Sólo con cuadros?





# Video



# Últimos resultados en esta Línea

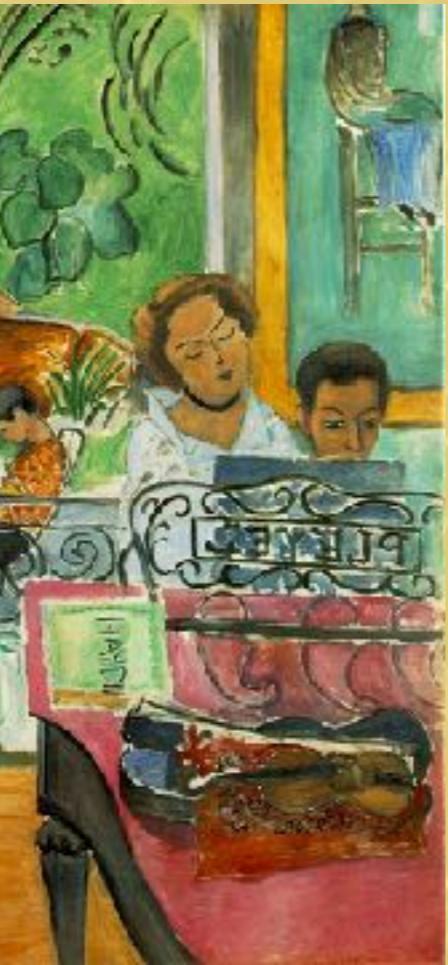
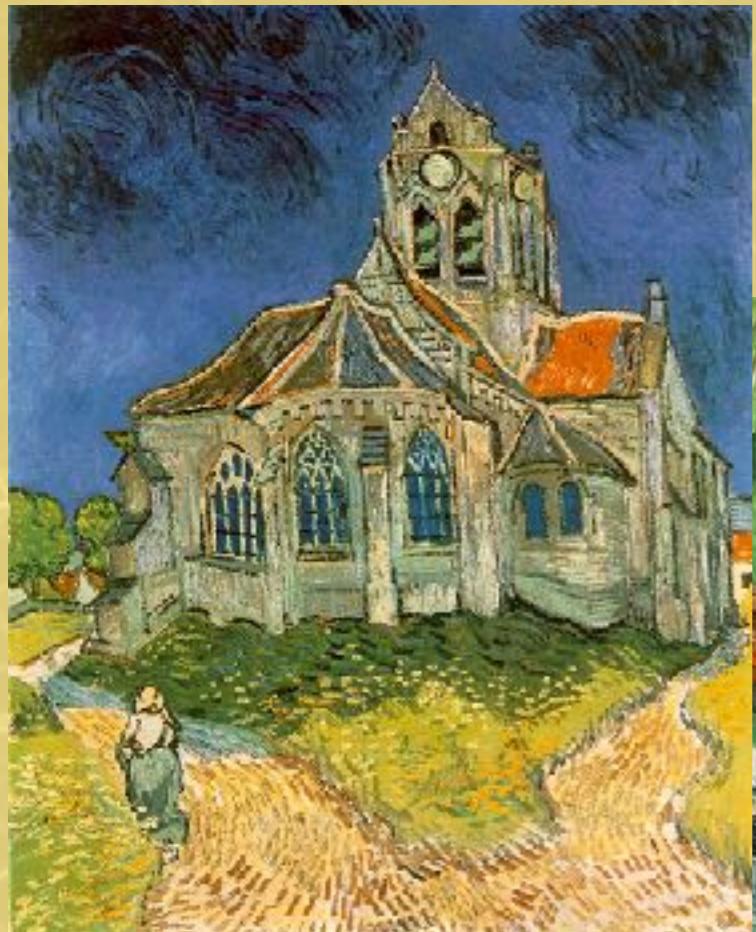


Texture synthesis and transfer using CNNs. Timo Aila et al., NVIDIA Research

# Aplicaciones al estilo musical

- \* Trabajo de Sony que aplica el estilo musical aprendido de una obra a otra obra diferente
  - <https://youtu.be/buXqNqBFd6E>
- \* Sistema de “arte incepcionista” de libre acceso previo registro mantenido por Google
  - <http://deepdreamgenerator.com/>
- \* Aplicación de redes LTSM (Recurrentes) a la composición musical
  - <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>

# Detección de obras de arte

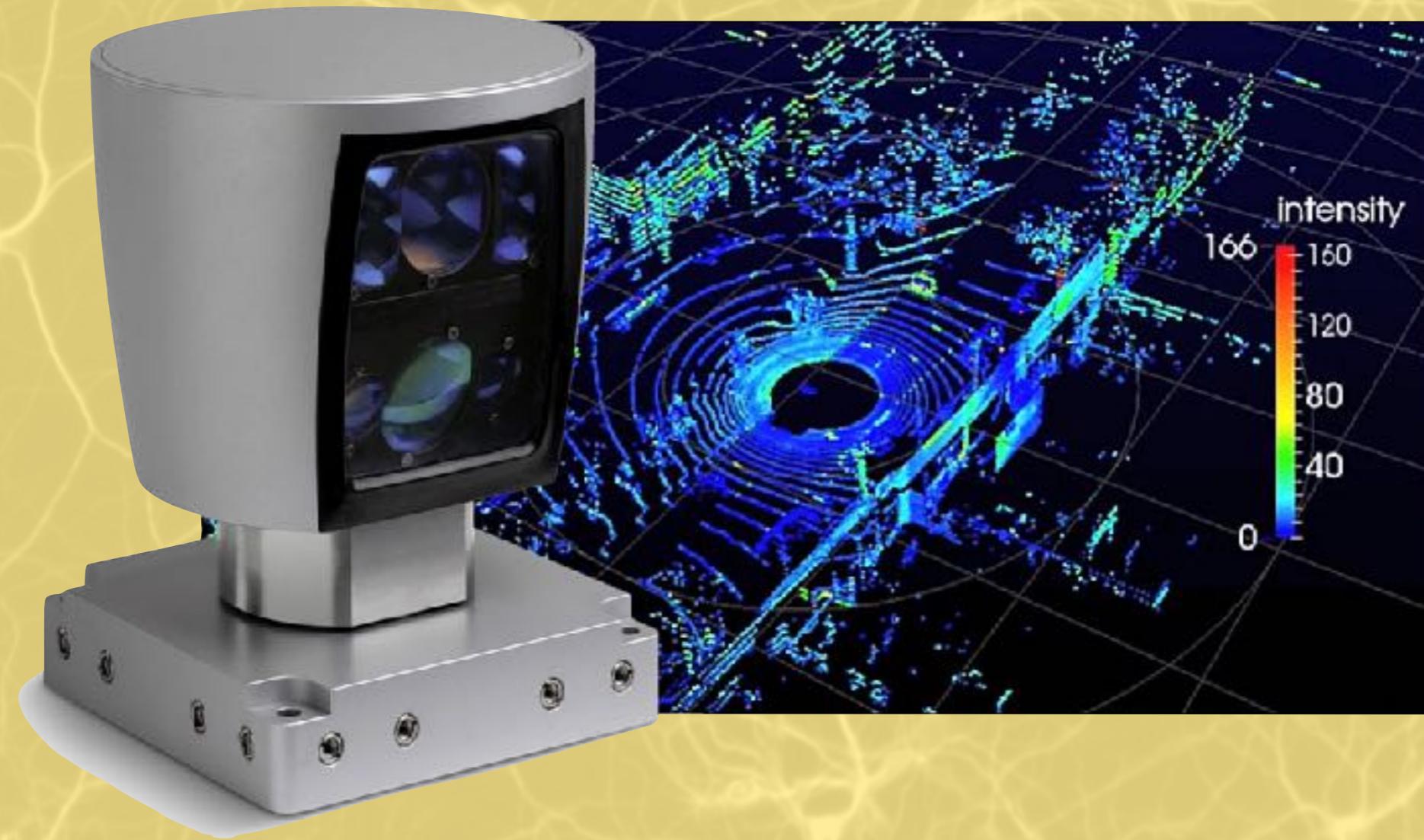


# Detección de obras de arte

- \* Entrenamiento de una CNN para distinguir entre cuadros buenos y cuadros malos (The Museum Of Bad Art)
- \* CNN con arquitectura
  - 256x256x3
  - C5:256x256x32 - ReLU - LRN - C5:256x256x32 - ReLU - LRN - MAXPOOL
  - C3:128x128x64 - ReLU - LRN - C3:128x128x64 - ReLU - LRN - MAXPOOL
  - FC64x64x64 - FC128 - 2 (softmax)
  - Loss: entropía cruzada
  - Local response normalization
  - Dropout con rate de 0.4 en las capas FC
- \* 87% de aciertos en test en 2700 epoch

# Estimación de parámetros a partir de LIDAR

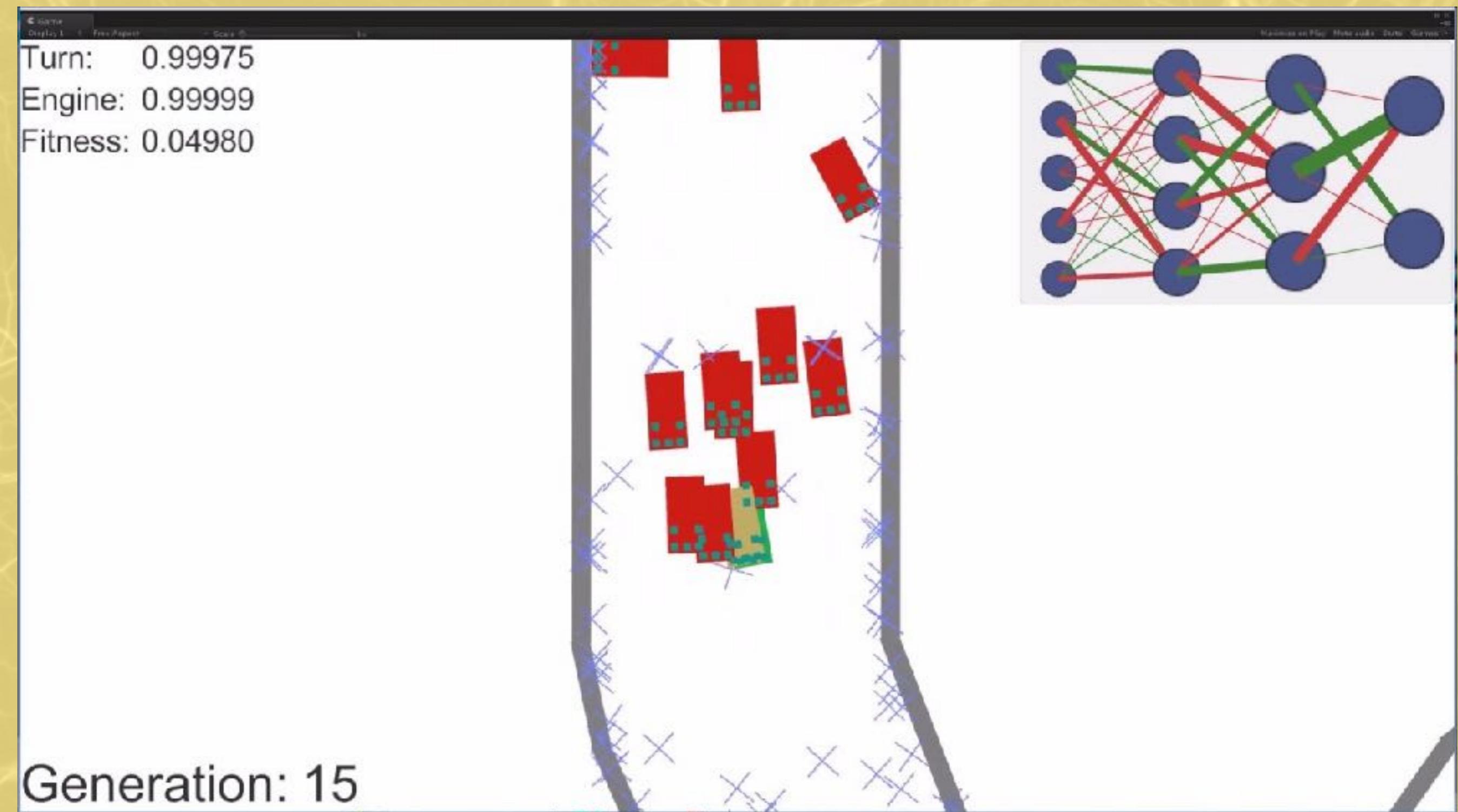
- \* Uso de CNNs para estimar velocidad y ángulos de un automóvil a partir de información de LIDAR
  - Resultados: 3 kmh de error en test



# Generación de caricaturas



# Aprendizaje por refuerzo



# Otros éxitos recientes

- \* AlphaGo / AlfaGo Zero / AlfaFold (DeepMind / Google)
  - Muy recomendable el documental de Netflix
- \* Clasificación y edición de imágenes a nivel comercial
  - Google, Apple...
  - Luminar, Pixelmator pro, Photoshop...
- \* Mejora de imágenes y videos
  - Coloreado de imágenes en blanco y negro
  - [4k, 60 fps] Arrival of a Train at La Ciotat (The Lumière Brothers, 1896)

# ¿Más allá del conocimiento humano?

- \* «Si he visto más lejos es porque estoy sentado sobre los hombros de gigantes» (Newton, carta a Robert Hooke, 1676) ¿Más allá de los humanos?
  - AlphaGo Zero
  - Mortalidad a partir de electrocardiograma con Random Forest
  - Los globos Loon de Google ya no escuchan a humanos

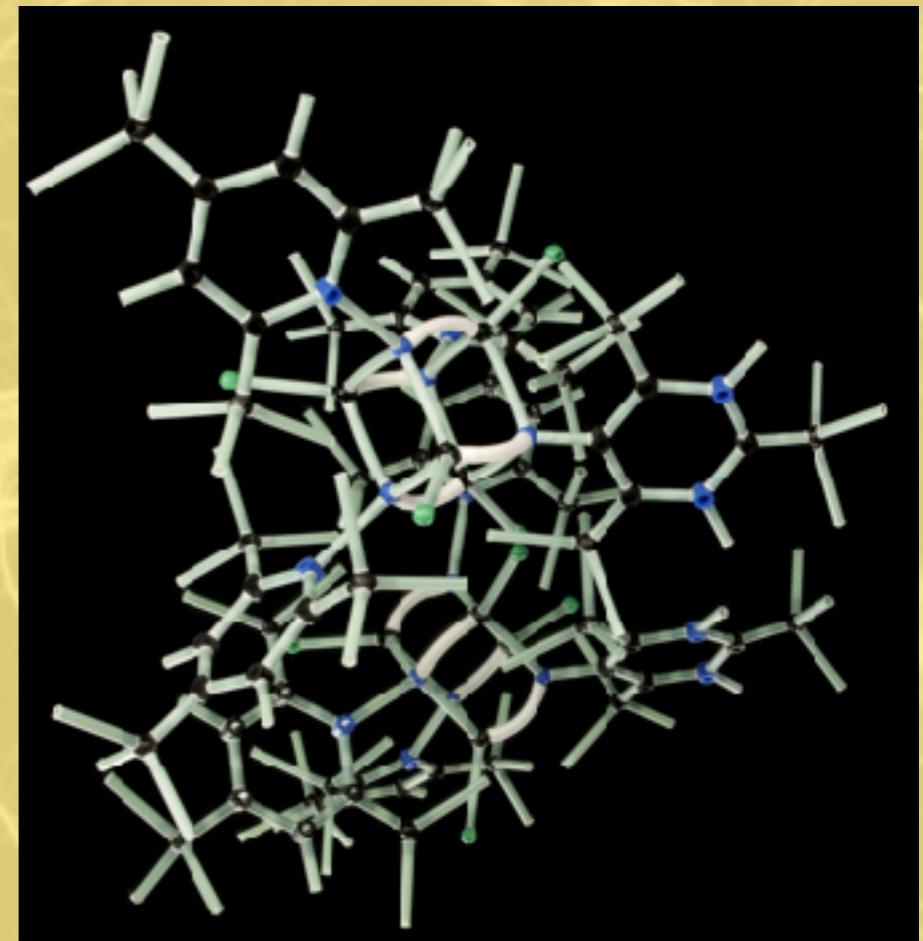
# Juegos de Atari



# ¿Futuro?

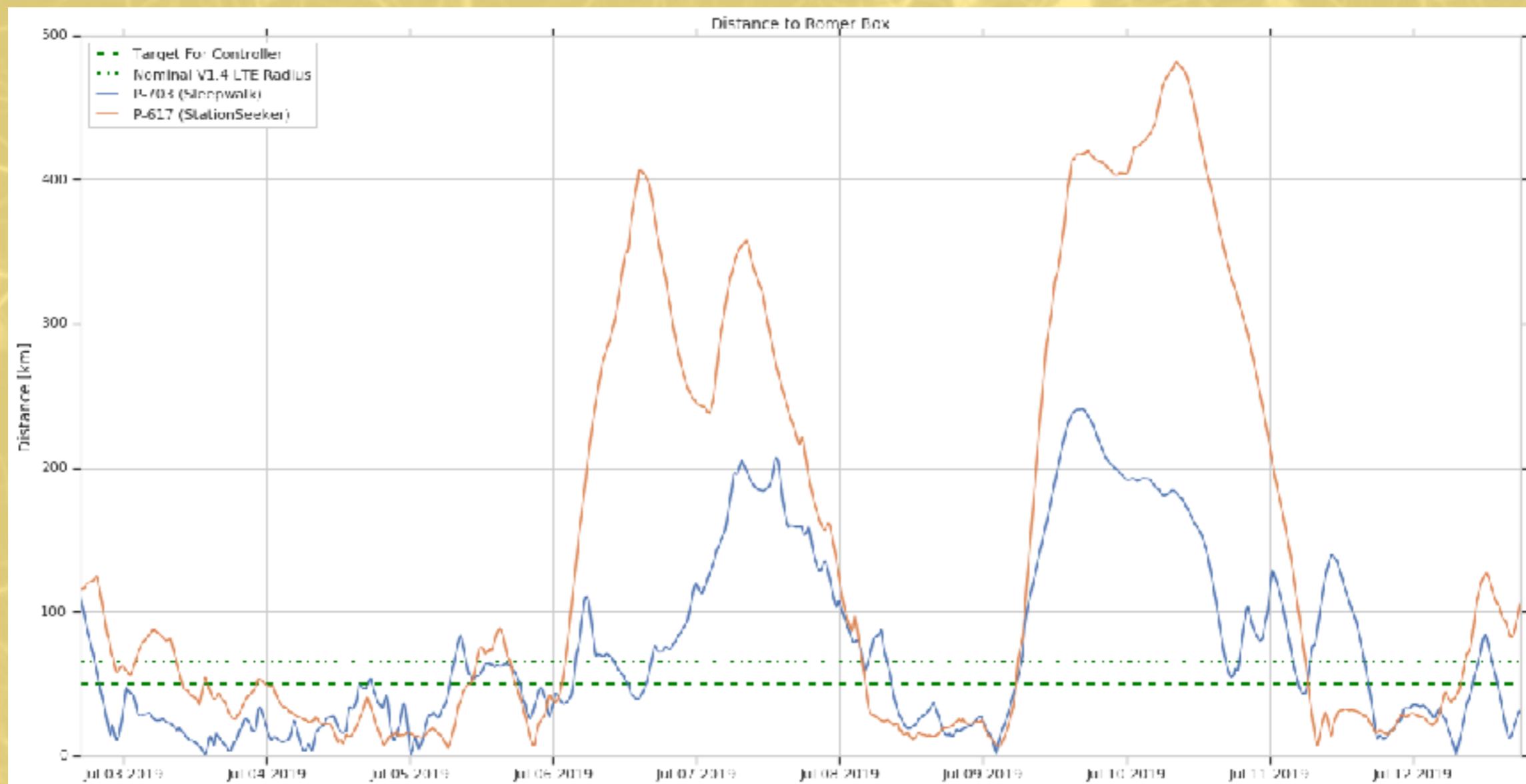
\* “DeepMind Makes History Again By Solving a 50-Year-Old Problem In Biology” (1/12/2020)

- Se trata de determinar el plegamiento que tendrá una proteína a partir de su secuencia de aminoácidos
- Hay una competición anual “Critical Assessment of Protein Structure Prediction”, o CASP
- Los mejores métodos hasta 2018 conseguían 40 puntos sobre 100 en el Global Distance Test (GDT)
- El AlphaFold 2 de DeepMind consiguió 60 GDT in 2018 y 87 GDT en 2020
- Obtuvo una precisión comparable a la de las pruebas de laboratorio (cristalografía de rayos X y espectroscopia de RMN) en 2/3 de las proteínas
- Google ha fundado en 2021 una nueva empresa (Isomorphic Labs) para explotar las aplicaciones farmacéuticas de este sistema



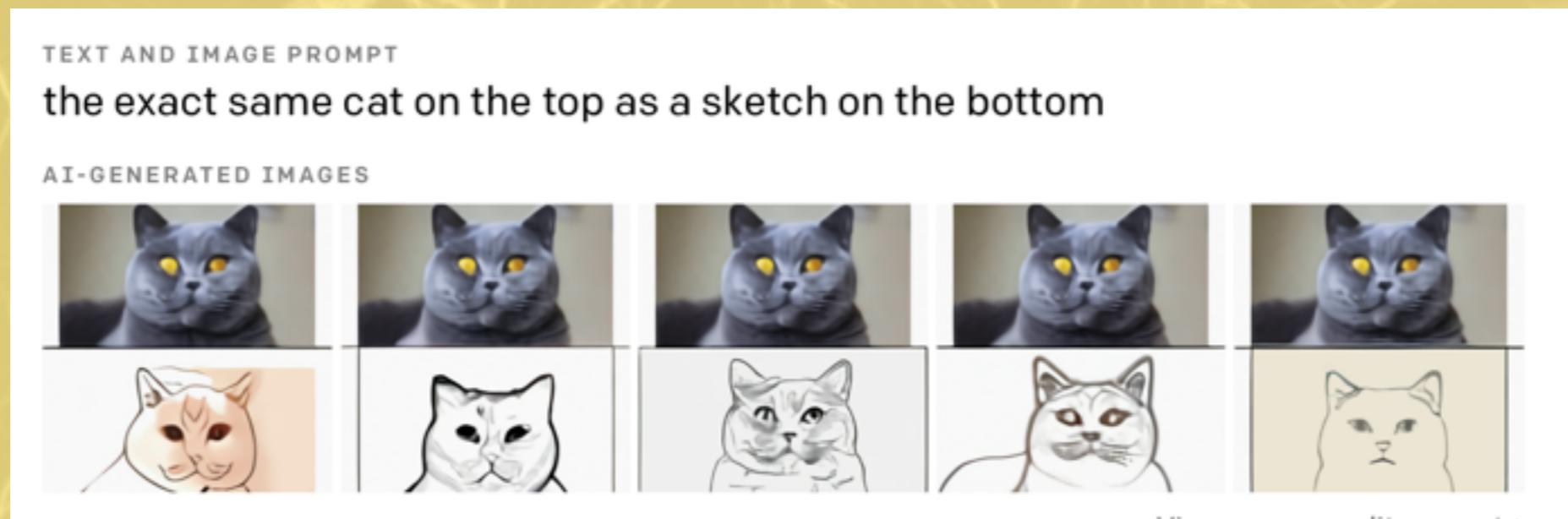
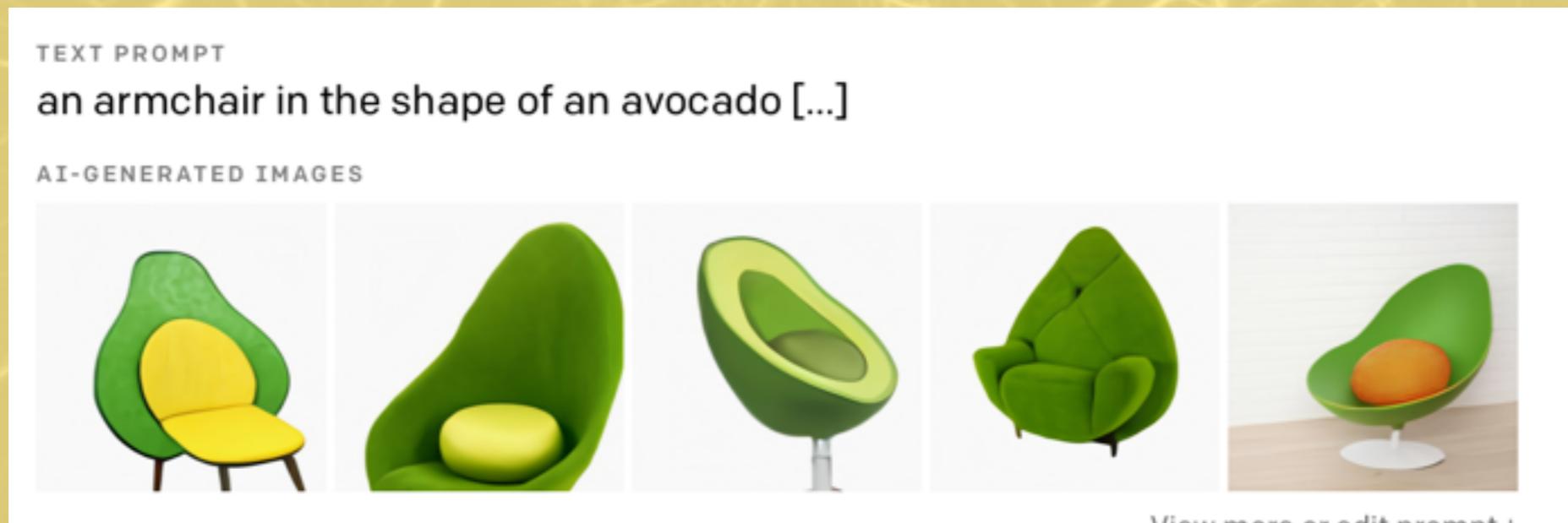
# Los globos Loon de Google ya no escuchan a humanos

- \* Drifting Efficiently Through the Stratosphere Using Deep Reinforcement Learning
  - Proyecto Sleepwalk



# ¿Le ha llegado la hora al lenguaje?

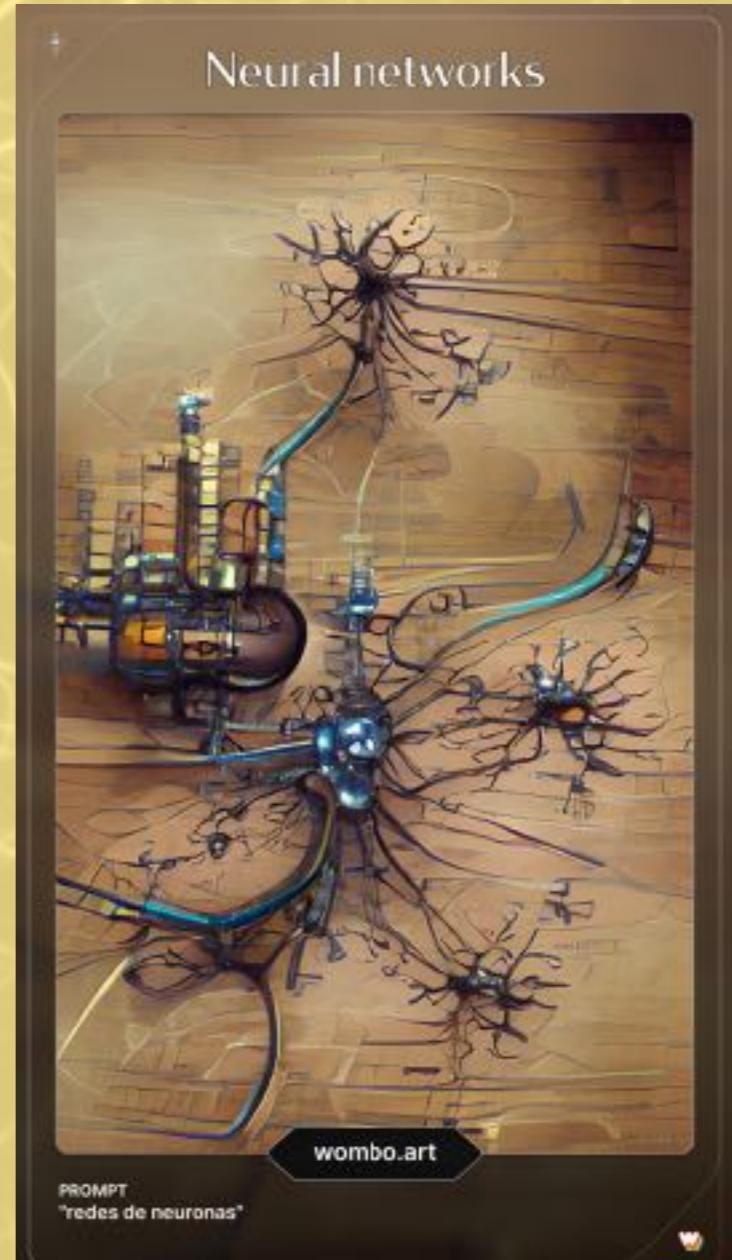
## \* DALL-E (GPT-3 + CNNs)



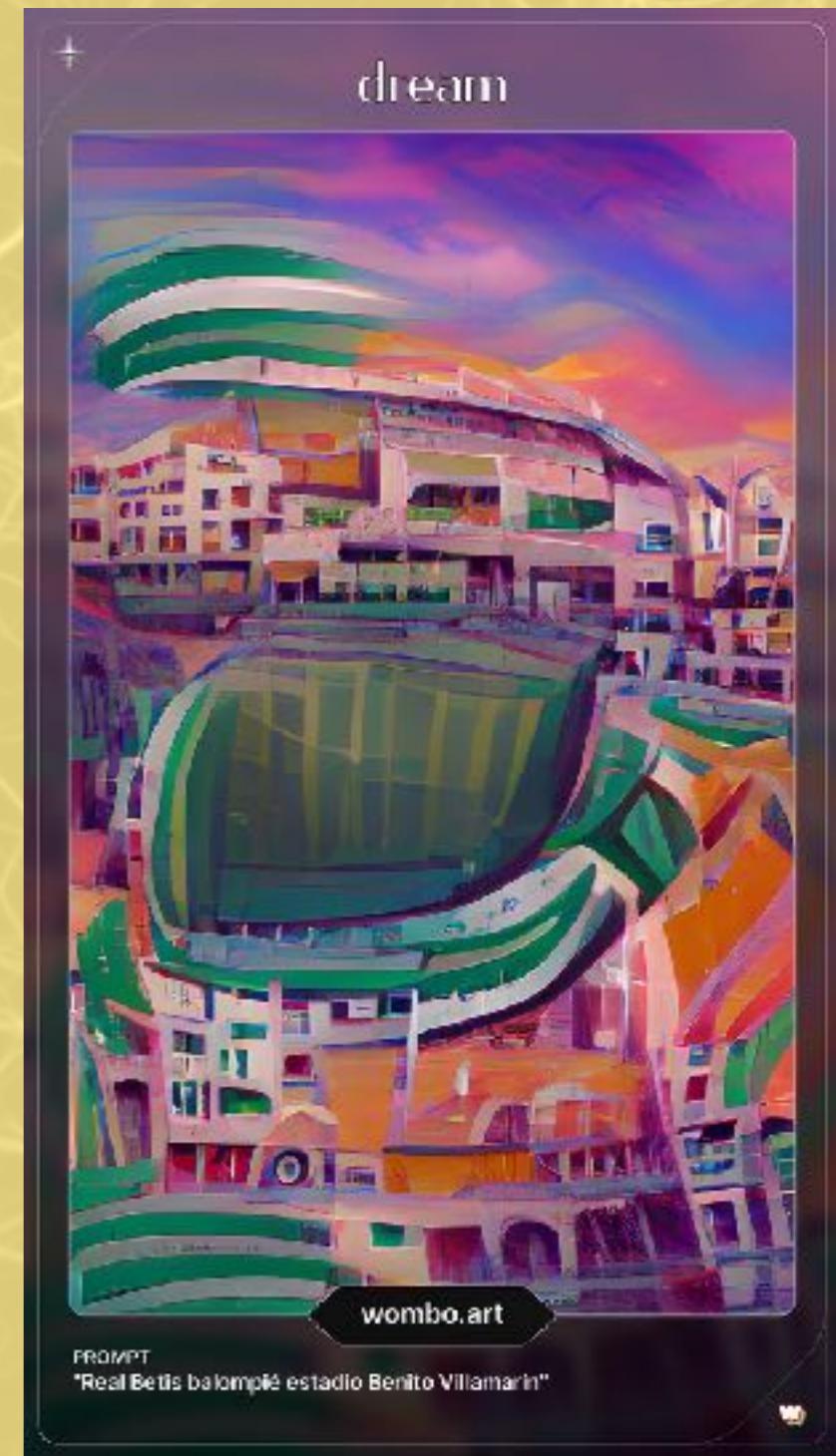
# ¿Le ha llegado la hora al lenguaje?

\* <https://app.wombo.art>

- Genera imágenes a partir de una frase



# wombo.art, ejemplos



# Repaso de conceptos

- \* ¿Qué es el deep learning? ¿En qué se diferencia de una red de neuronas normal?
- \* ReLU
- \* Softmax
- \* Dropout
- \* ¿Qué es un auto-encoder?
- \* ¿Qué es una convolución?
- \* ¿Qué es una red de convolución?
- \* ¿Qué es el transfer learning?
- \* ¿Qué es una red recurrente?
- \* ¿Qué es una GAN?

# Herramientas

# Herramientas

- \* Redes
  - Tensorflow + Keras
  - Pytorch
  - Teano, Caffe...
- \* Machine Learning
  - Sklearn
- \* Data cleansing
  - pandas
- \* Interfaz
  - Jupyter notebook
  - Matplotlib
  - Google Colab

# Tensorflow

- \* Librería de Google para cálculo intensivo
  - Orientada al cálculo matricial
  - Basada en numpy y theano, pero más fácil de instalar
  - Permite el uso de GPUs de tarjetas Nvidia con drivers CUDA (Compute Unified Device Architecture) de modo transparente
  - Dispone de un módulo específico para Redes de Neuronas
  - Disponible para Python (3.5 - 3.7). Hay bindings para Java, C y Go
  - Disponible para Ubuntu (CPU+GPU), Windows (7+) y MacOS (Sólo CPU)
  - <https://www.tensorflow.org>
- \* Se basa en definir de modo simbólico los cálculos a realizar, formando un grafo de operaciones. Desde la versión 2.0 el grafo es transparente para el usuario
  - Los datos numéricos se suministran al grafo y permanecen fuera de nuestro programa, a menos que queramos acceder a ellos
    - ✖ Eso lo hace muy eficiente

# Operaciones

- \* La estructura básica es el tensor (de ahí el nombre) que es una matriz multidimensional
  - Las operaciones se aplican a uno o varios tensores, utilizando para ello los recursos de la máquina de modo transparente al usuario
- \* Disponemos en el módulo tensorflow.nn de las operaciones típicas de Redes de Neuronas
  - Funciones de activación (sigmoid, ReLU, etc)
  - Algoritmos de aprendizaje (optimizadores)
  - Operadores de convolución, maxpool, etc
  - Con estos bloques es muy sencillo construir redes de neuronas para resolver problemas concretos

# Tensorflow

- \* Explicación en Jupiter notebook

## \* Hacia un nuevo enfoque basado en capas

- En Keras definimos capas que se irán apilando una detrás de otra, de modo que en general no hace falta definir cuáles son las entradas de cada capa
- También es posible construir estructuras no secuenciales a través de modelos funcionales, en los que las salidas se guardan en variables y se aplica cada capa a las variables que interese
- Podemos añadir capas de cualquier tipo en cualquier orden, siempre que sus salidas tengan el formato adecuado (por ejemplo no podemos colocar una capa unidimensional en salidas bidimensionales sin antes hacer un reshape o un flatten)
- Algunas operaciones que no son propiamente capas (dropout, flatten, normalizaciones...) se añaden también como capas
- Las funciones de activación podemos añadirlas como capas o como argumento dentro de una capa

- \* Tipos básicos de capas (from tensorflow.keras import layers)
  - Input → capa de entrada, no efectúa ninguna operación
  - Dense → capa de perceptrón multicapa
  - Conv2D → capa de convolución
  - MaxPool2D: → operación maxpool
- \* Argumentos de las capas
  - Algunos varían con la capa, pero los fundamentales son
    - ✖ Dimensión (número de neuronas); es el primer argumento
    - ✖ Función de activación (parámetro activation)

# Ejemplo (API secuencial)

```
1 # pure network for classification
2
3 dr = 0.9 # 0.5
4 MLP = Sequential()
5 MLP.add(Dense(64, activation='relu', input_shape=(22050,)))
6 MLP.add(Dropout(dr))
7 MLP.add(Dense(32, activation='relu'))
8 MLP.add(Dropout(dr))
9 MLP.add(Dense(2, activation='softmax'))
10
11 from tensorflow import keras
12 opt = keras.optimizers.Adam(learning_rate=0.00001)
13
14 MLP.compile(optimizer='adam', loss='categorical_crossentropy',
15               metrics=['categorical_accuracy',r2])
16
17 bs = 512
18 res = MLP.fit(X_train, Y_train, validation_data=(X_test, Y_test),
19                 epochs=200, batch_size=bs, verbose=0, callbacks=[PeriodicPrint(50)])
```

# Ejemplo (API secuencial)

```
1 #create model
2 model = Sequential()
3
4 #add model layers
5 model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(128, 128, 1)))
6 model.add(LayerNormalization())
7 model.add(Conv2D(32, kernel_size=3, activation='relu'))
8 model.add(MaxPooling2D((2, 2), strides=(1, 1)))
9
10 model.add(Conv2D(64, kernel_size=3, activation='relu'))
11 model.add(LayerNormalization())
12 model.add(Conv2D(64, kernel_size=3, activation='relu'))
13 model.add(MaxPooling2D((2, 2), strides=(1, 1)))
14
15 drate = 0.5
16 model.add(Flatten())
17 #model.add(Dropout(drate))
18 #model.add(Dense(1024, activation='relu'))
19 #model.add(Dropout(drate))
20 model.add(Dense(64, activation='relu'))
21 model.add(Dropout(drate))
22 model.add(Dense(32, activation='relu'))
23 model.add(Dropout(drate))
24 model.add(Dense(2, activation='linear'))
25
26 bs = 128
```

# Ejemplo (API funcional)

```
1 def create_generator():
2     input = Input(shape=(100))
3
4     x = Dense(256)(input)
5     x = LeakyReLU(alpha=0.2)(x)
6     x = BatchNormalization(momentum=0.8)(x)
7
8     x = Dense(512)(x)
9     x = LeakyReLU(alpha=0.2)(x)
10    x = BatchNormalization(momentum=0.8)(x)
11
12    x = Dense(1024)(x)
13    x = LeakyReLU(alpha=0.2)(x)
14    x = BatchNormalization(momentum=0.8)(x)
15
16    x = Dense(28*28, activation='tanh')(x)
17    output = Reshape((28, 28, 1))(x)
18
19    generator = Model(input, output)
20
21    return generator
```

## \* Redimensión

- Flatten
- Reshape

## \* Regularización

- Dropout
- LayerNormalization
- BatchNormalization

## \* Capas avanzadas

- LSTM → capa Long Short Term Memory (para series temporales)
- UpSampling2D → operación inversa al maxpool
- Conv2DTranspose → capa de convolución transpuesta (inversa de Conv2D)
- MultiHeadAttention

# Capas de salida dependiendo de la tarea

## Last-layer activation and loss function combinations

Problem type	Last-layer activation	Loss function	Example
Binary classification	sigmoid	binary_crossentropy	Dog vs cat, Sentiment analysis(pos/neg)
Multi-class, single-label classification	softmax	categorical_crossentropy	MNIST has 10 classes single label (one prediction is one digit)
Multi-class, multi-label classification	sigmoid	binary_crossentropy	News tags classification, one blog can have multiple tags
Regression to arbitrary values	None	mse	Predict house price(an integer/float point)
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy	Engine health assessment where 0 is broken, 1 is new

# Referencias web

- \* Tensorflow-slim
  - <https://github.com/tensorflow/models/blob/master/research/inception/inception/slim/README.md>
- \* Curso gratuito de Deep Learning de Google
  - <https://www.udacity.com/course/deep-learning--ud730>
- \* Conjuntos de datos estándar y estado del arte de cada uno
  - <https://martin-thoma.com/sota/>

# Referencias

- \* Understanding LSTM Networks
  - <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- \* The Unreasonable Effectiveness of Recurrent Neural Networks
  - <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# Referencias

- \* Convolutional Neural Networks for Visual Recognition
  - <http://cs231n.github.io/convolutional-networks>
- \* Regularización
  - <http://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html/2>
  - <http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/>
- \* Tensorflow
  - <https://www.tensorflow.org>
  - <https://github.com/aymericdamien/TensorFlow-Examples>
- \* Ejemplos
  - <http://cs.stanford.edu/people/karpathy/convnetjs/index.html>

# Referencias

- \* Kyle McDonald. A return to machine learning. <https://medium.com/@kcimc/a-return-to-machine-learning-2de3728558eb#.croi6l1zv>
- \* Lorentz, G. G. (1976, August). The 13th problem of Hilbert. In Proceedings of Symposia in Pure Mathematics (Vol. 28, pp. 419-430). American Mathematical Society.
- \* Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4), 303-314.
- \* Lippmann, R. (1987). An introduction to computing with neural nets. IEEE Assp magazine, 4(2), 4-22.
- \* Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105)