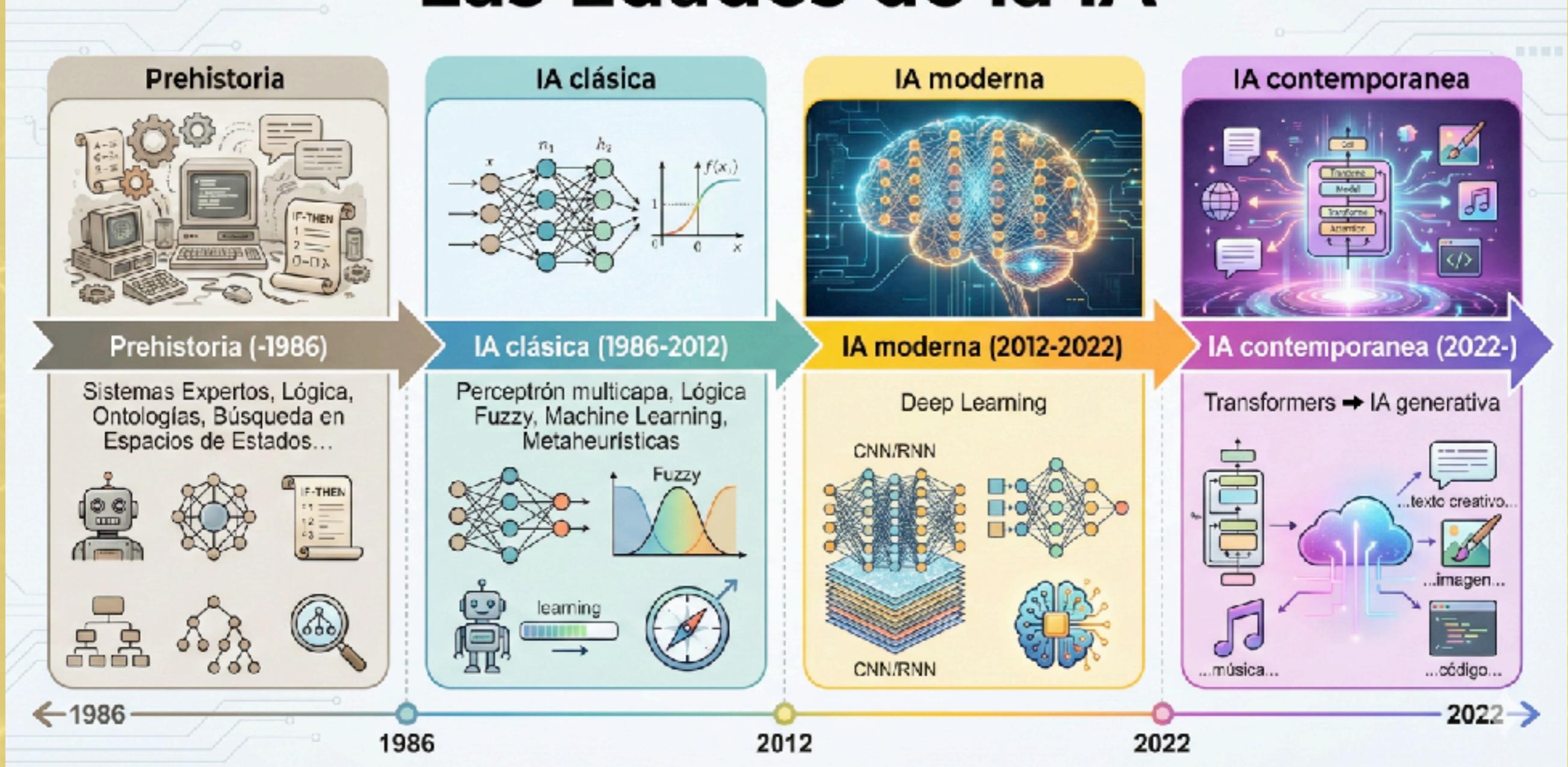


Francisco Serradilla García
Dr. en Informática por el Departamento de
Inteligencia Artificial de la UPM

6 de febrero de 2026

IA moderna

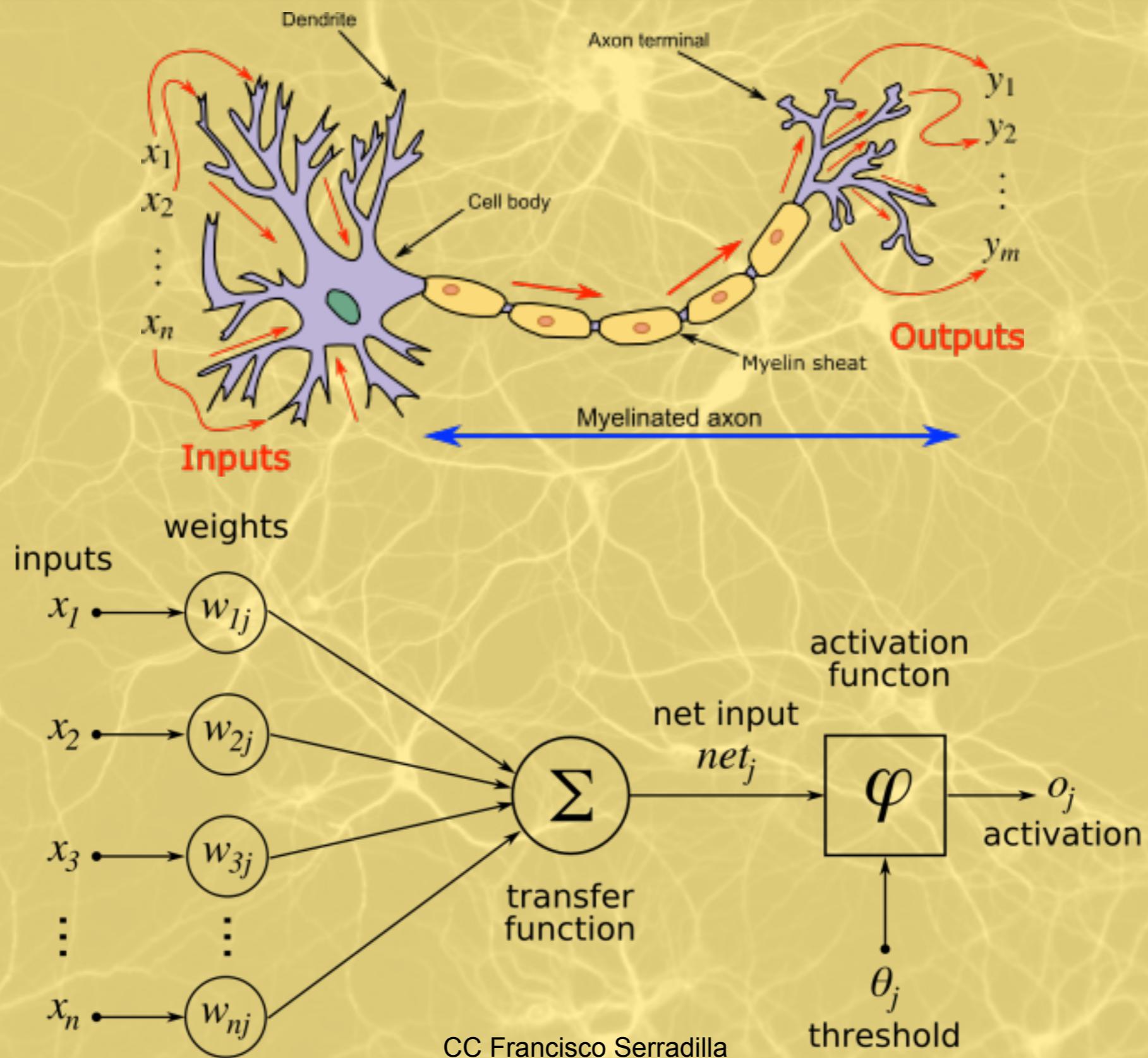
Las Edades de la IA



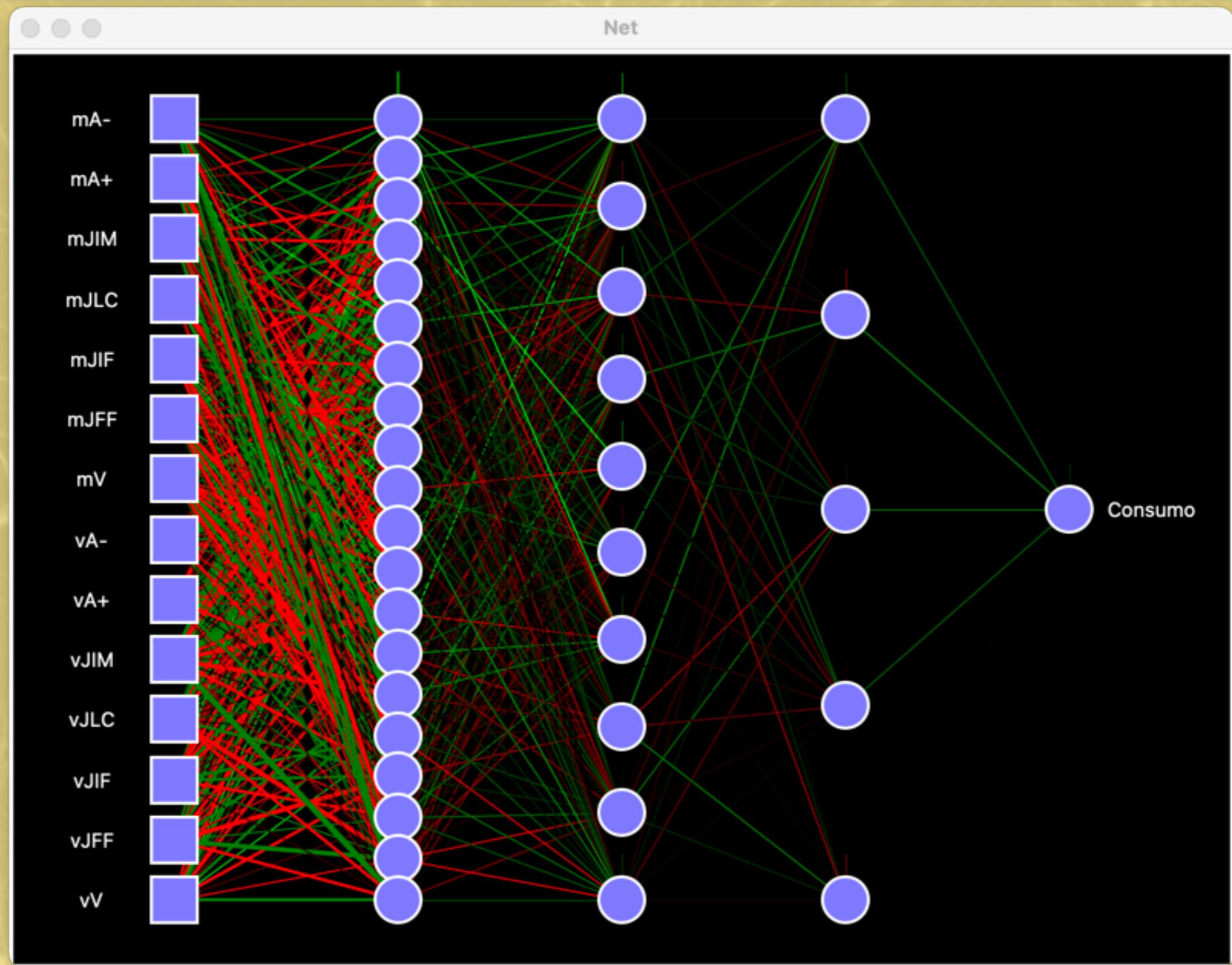
Redes de neuronas → Deep learning

- * Una Red “profunda” no es más que una red de neuronas con muchas capas
- * Por ello es importante entender cómo funcionan las redes de neuronas “tradicionales”, especialmente el perceptrón multicapa (MLP)
- * El perceptrón multicapa sigue siendo un componente fundamental de las redes, aunque se han añadido nuevos tipos de capas
 - Capas de convolución
 - Autoencoders
 - Transformers
- * Estas capas se conectan de diferentes maneras para obtener redes que realizan determinadas tareas
- * Por ello vamos a comenzar recordando los fundamentos del perceptrón, para luego explicar las cosas nuevas

La neurona formal



Organización en capas



Propagación en capas

$$s^{(1)} = F(eW^{(1)} + b^{(1)})$$

$$s^{(2)} = F(s^{(1)}W^{(2)} + b^{(2)})$$

genericamente :

$$s^{(k)} = F(s^{(k-1)}W^{(k)} + b^{(k)})$$

con $s^{(0)} = e$

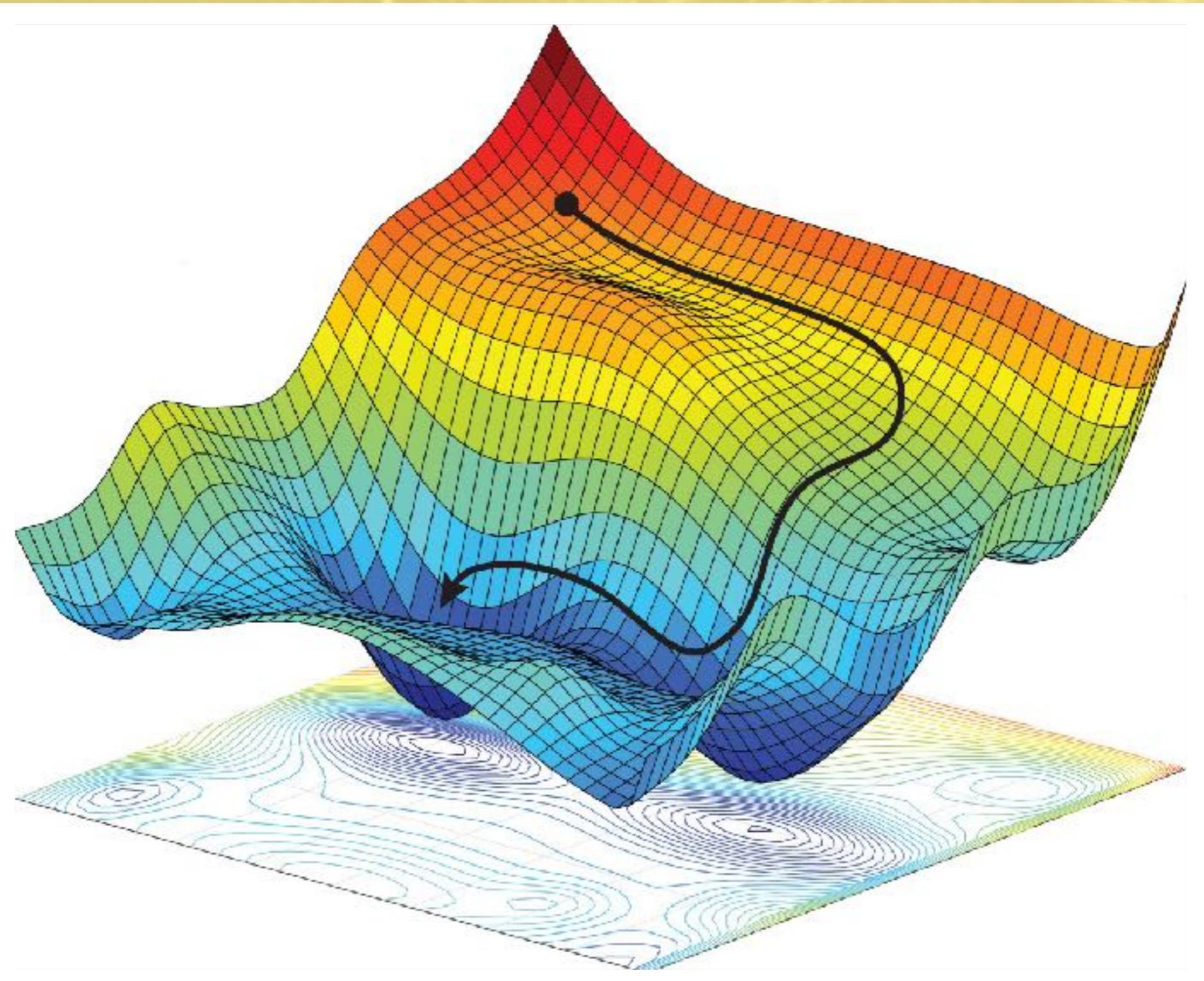
Aprendizaje

* Objetivo

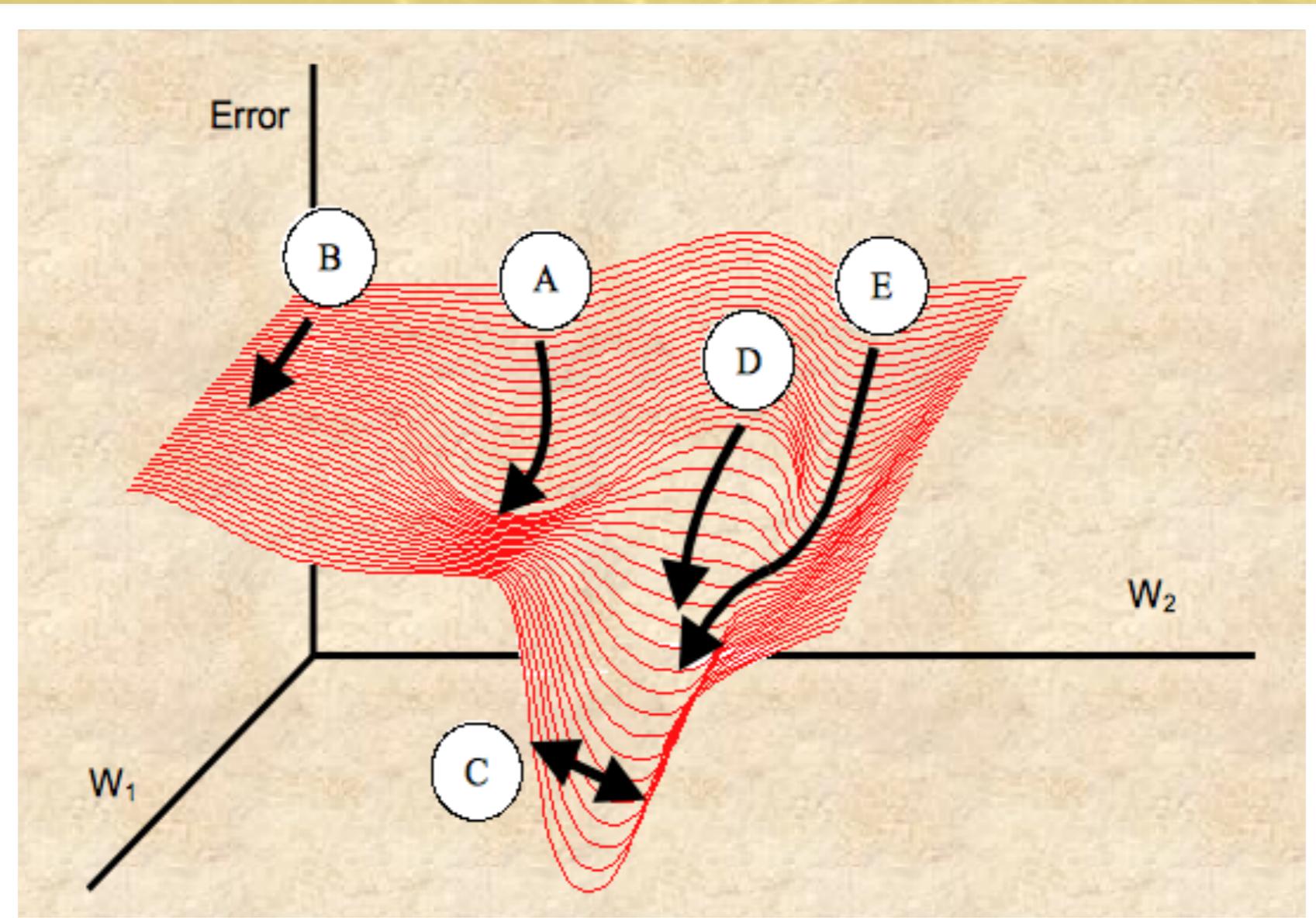
- Conseguir valores adecuados para los pesos w_{ij} de toda la red
- En general no existe solución analítica al problema
- Necesitamos un conjunto de ejemplos que mostraremos a la red un cierto número de veces
- Se utilizarán procedimientos de aproximación numérica a la solución, con algoritmos iterativos

* Al proceso de aprendizaje también se le denomina entrenamiento de la red

Idea general detrás del aprendizaje



Problemas del BP



- * mínimo local (A), meseta (B), oscilaciones (C)
- * sensibilidad al punto inicial (D y E)

Nacimiento del Deep Learning

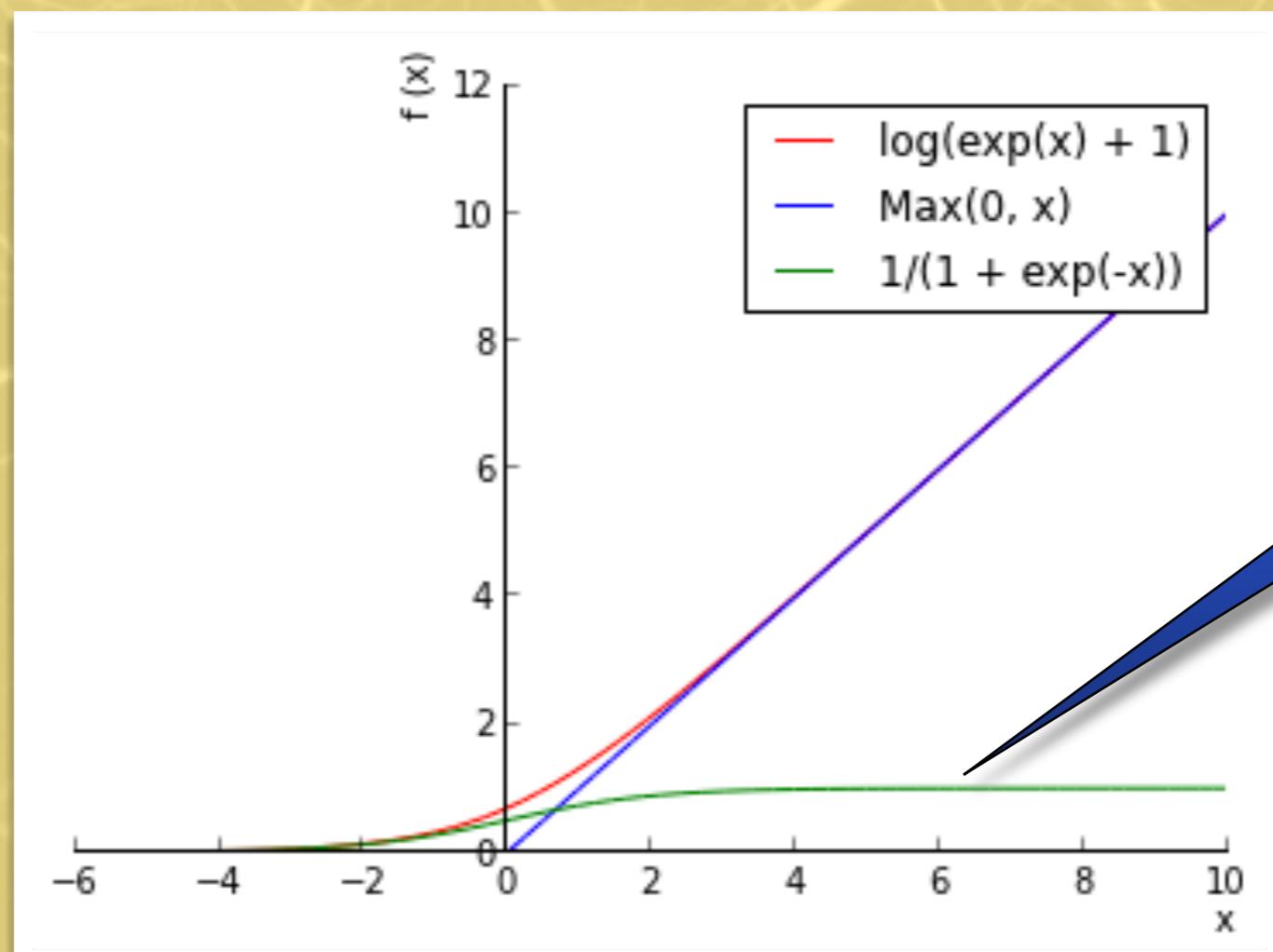
- * En [Cybenko, 1989], usando el teorema de representación de Kolmogorov–Arnold (que resuelve el 13th Hilbert Problem), se demostró que
 - Con una capa oculta puede aproximarse cualquier función continua de N variables
 - Por ello se prestó poca atención durante muchos años a las redes con muchas capas 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

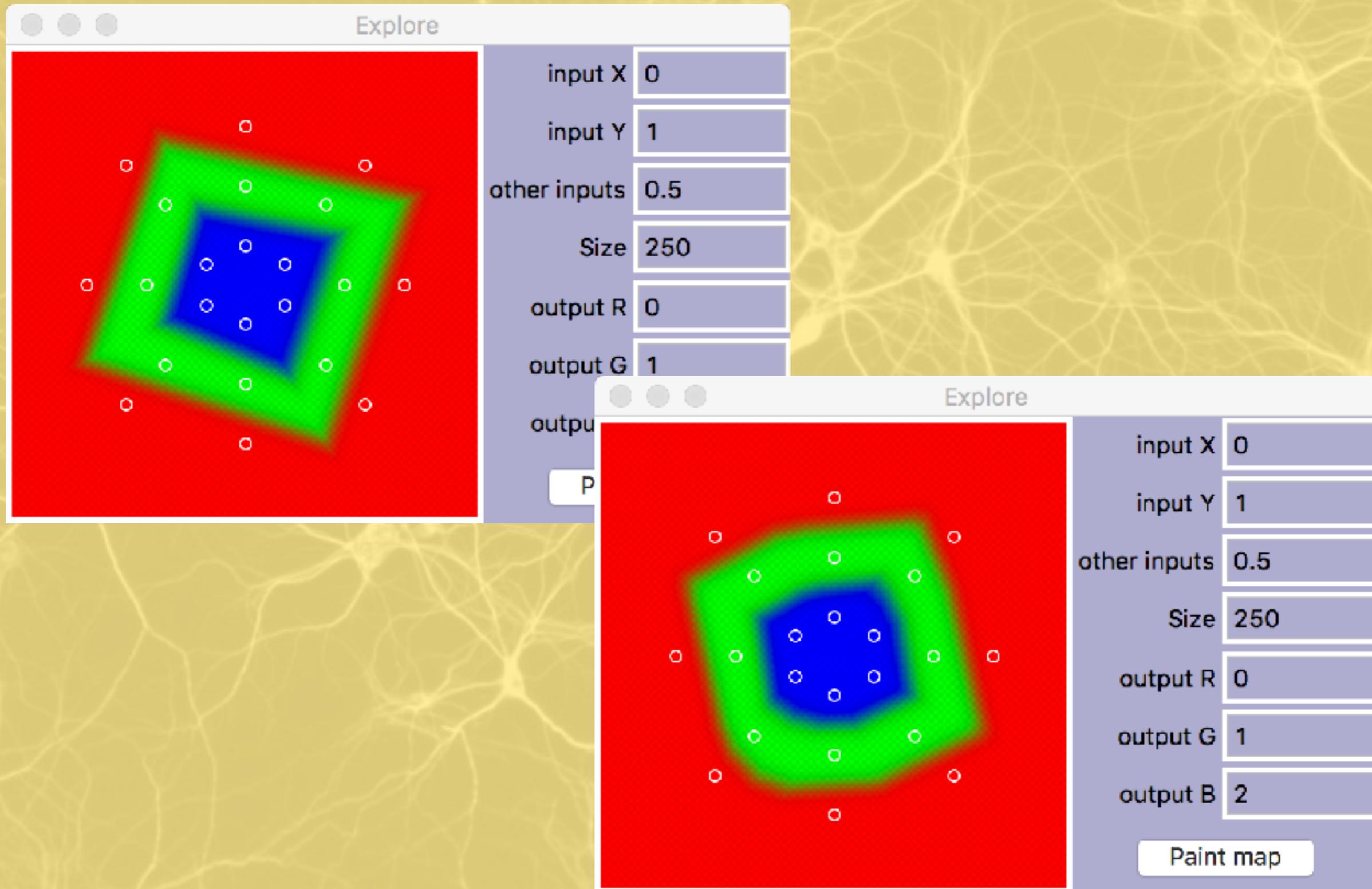
- * Uso de muchas capas
- * Compartición de pesos (redes de convolución)
- * Capas de extracción de características (auto-encoders y Restricted Boltzmann Machines)
- * Uso de neuronas ReLU en lugar de tanh o sigmoidal para evitar la saturación de las neuronas (derivada cercana a 0, o vanishing gradient 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)

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

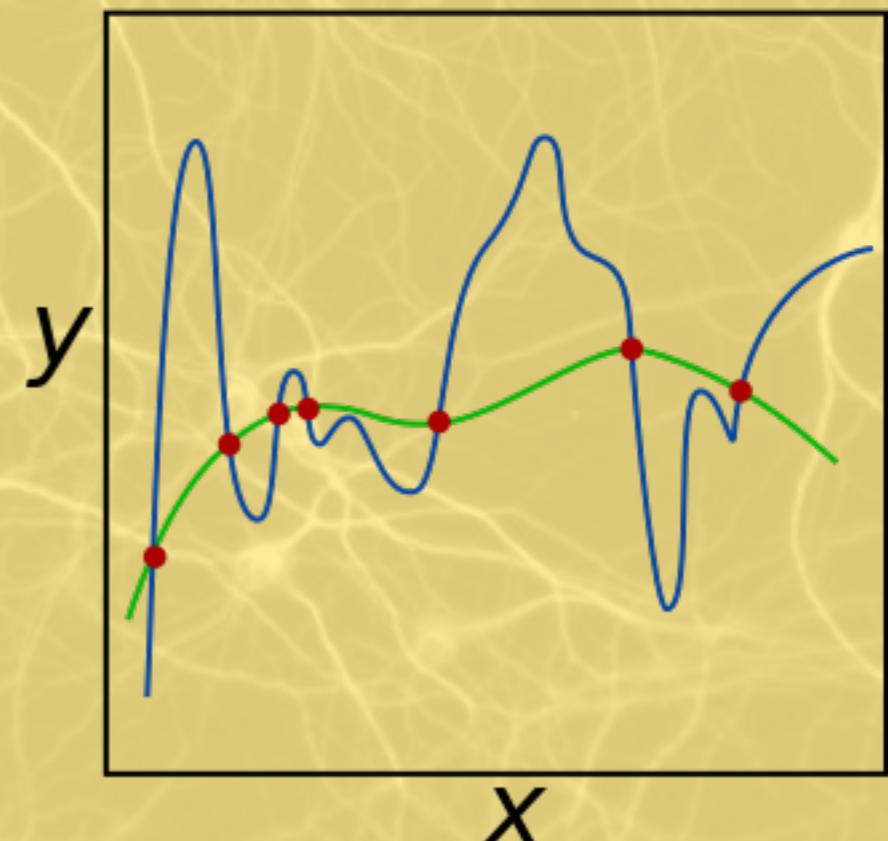
Entropía cruzada (cross entropy)

- * 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 λ controla cuánta importancia damos a la regularización frente al aprendizaje, y depende del problema

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

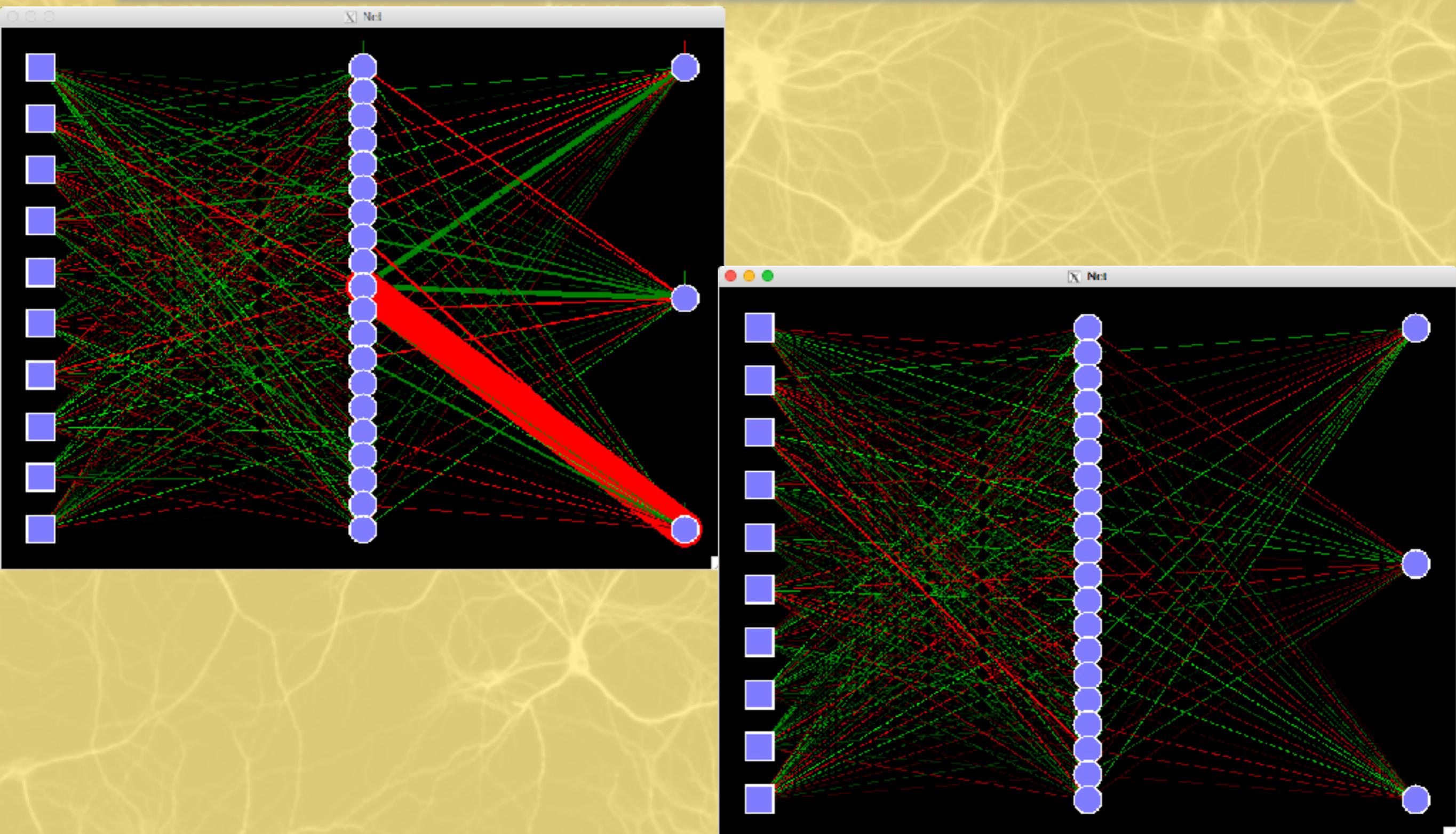
Regularización L1

* L1

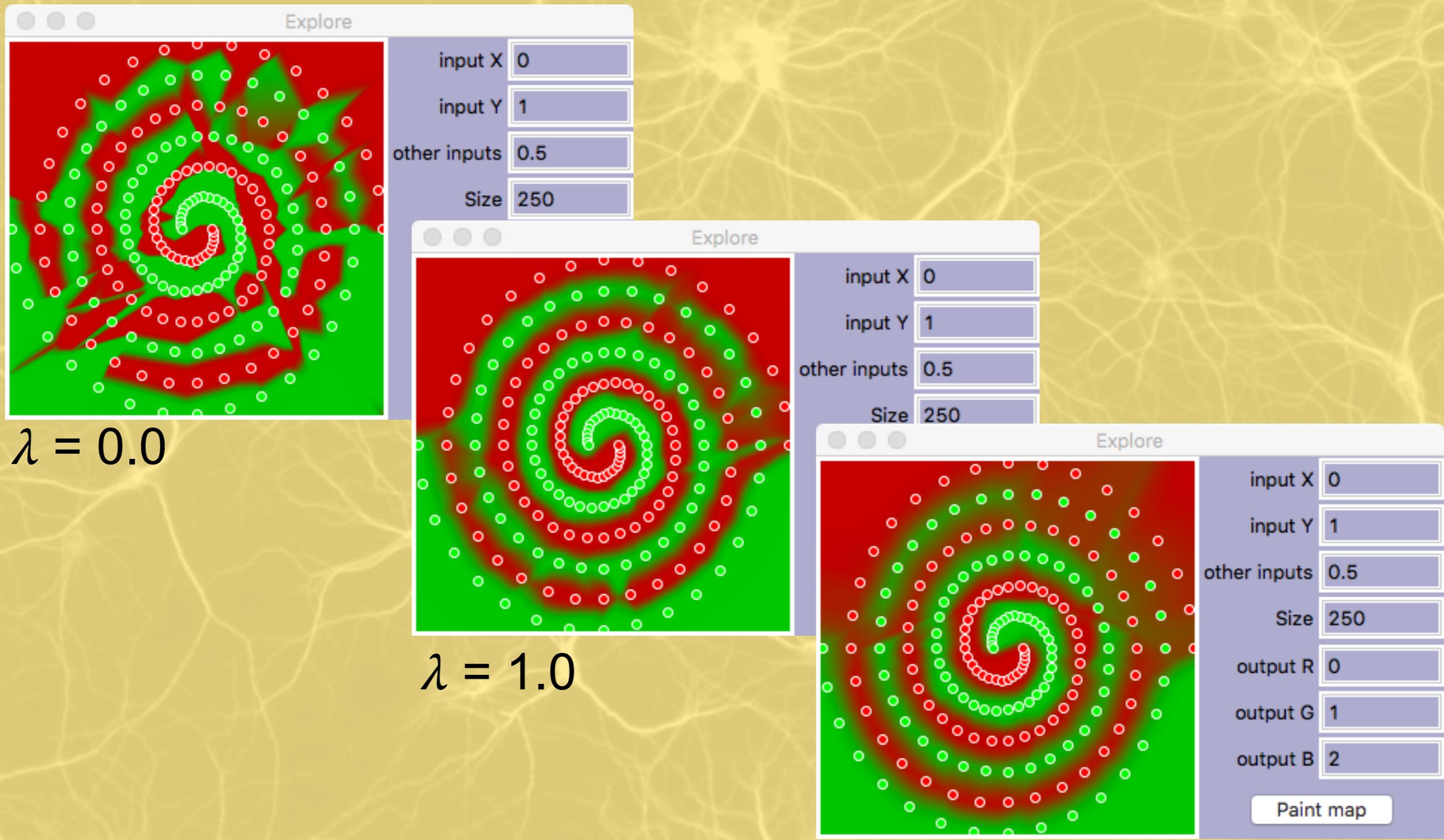
- Se usa cuando queremos detectar las entradas más relevantes
- El parámetro λ controla cuánta importancia damos a la regularización frente al aprendizaje, y depende del problema

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

Efectos de la regularización



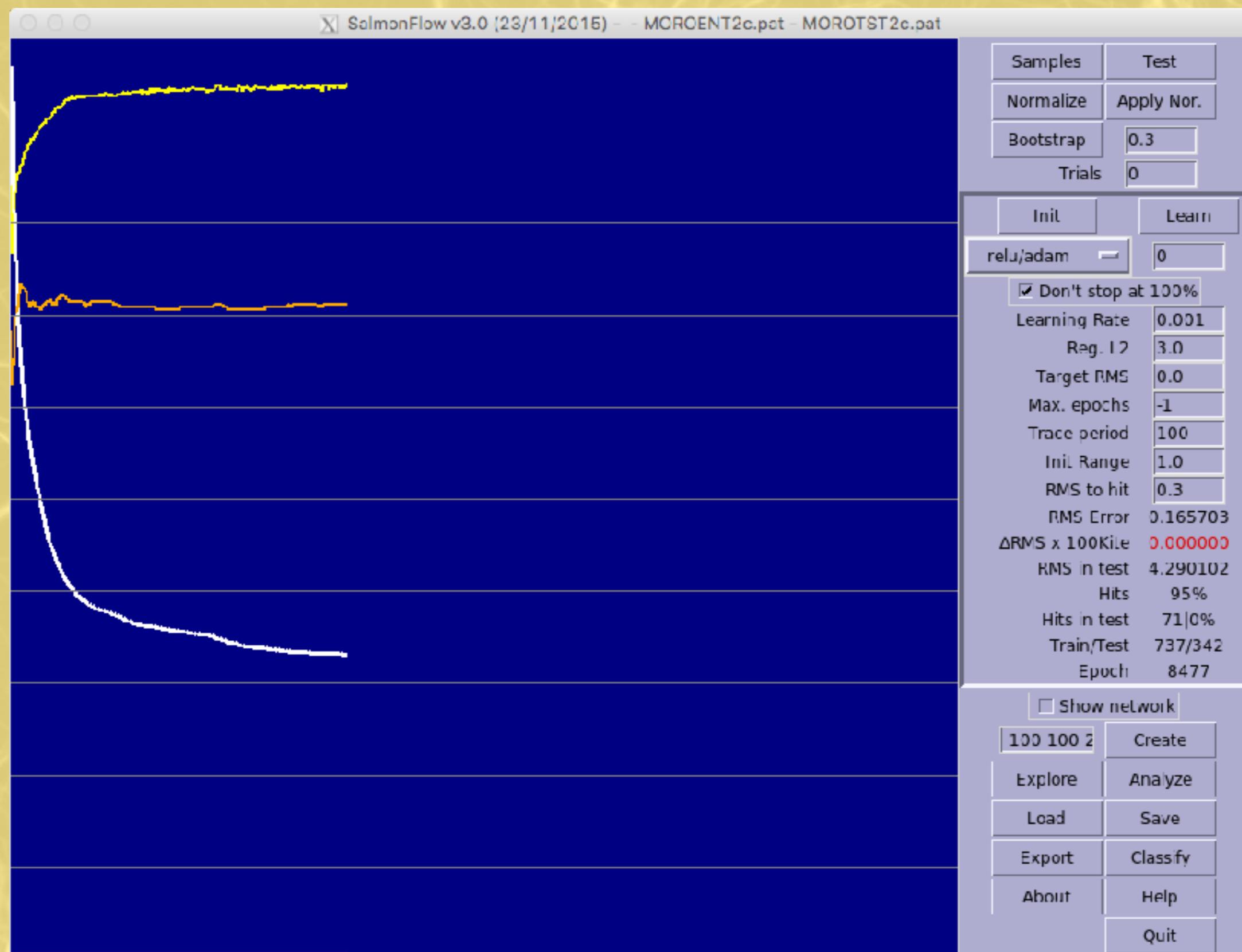
Efectos de la regularización



Efectos de la regularización



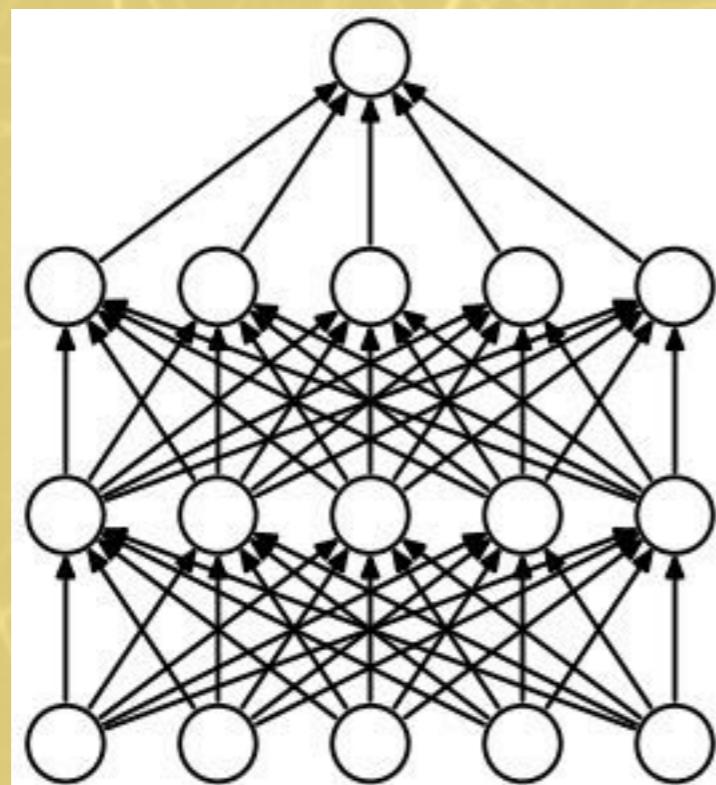
Efectos de la regularización



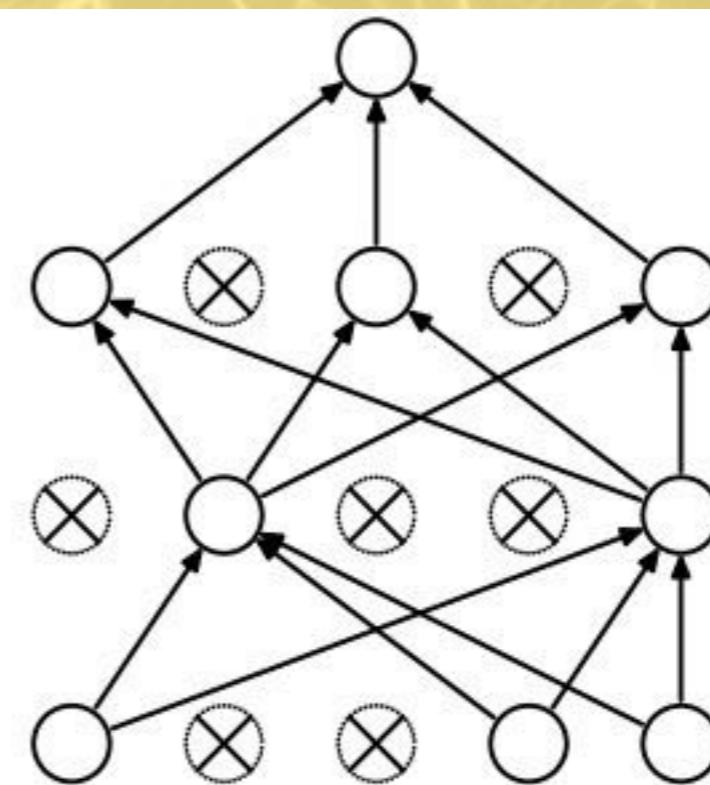
Dropout

* 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

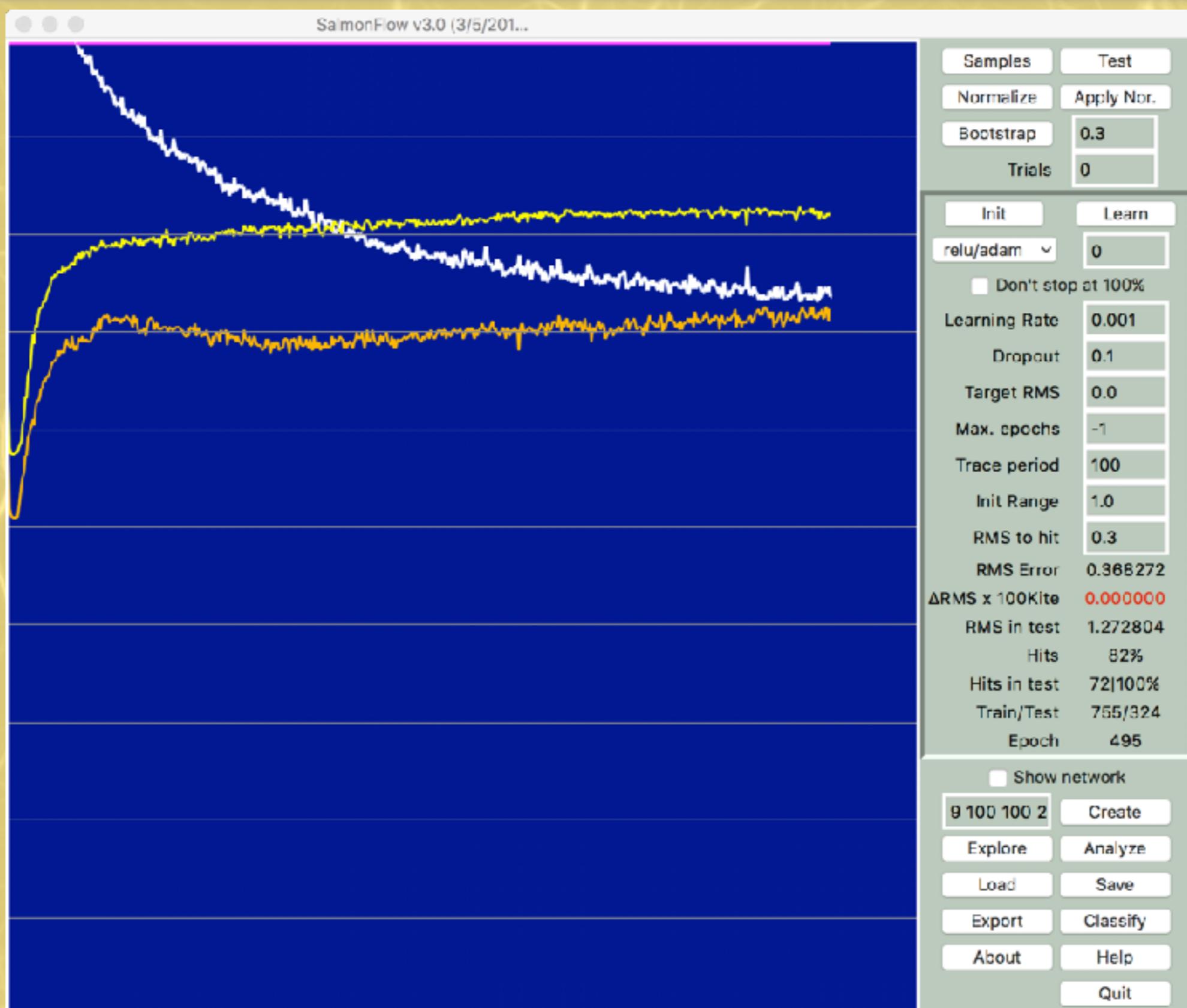
* ¿Cuántas neuronas desactivar?

- 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

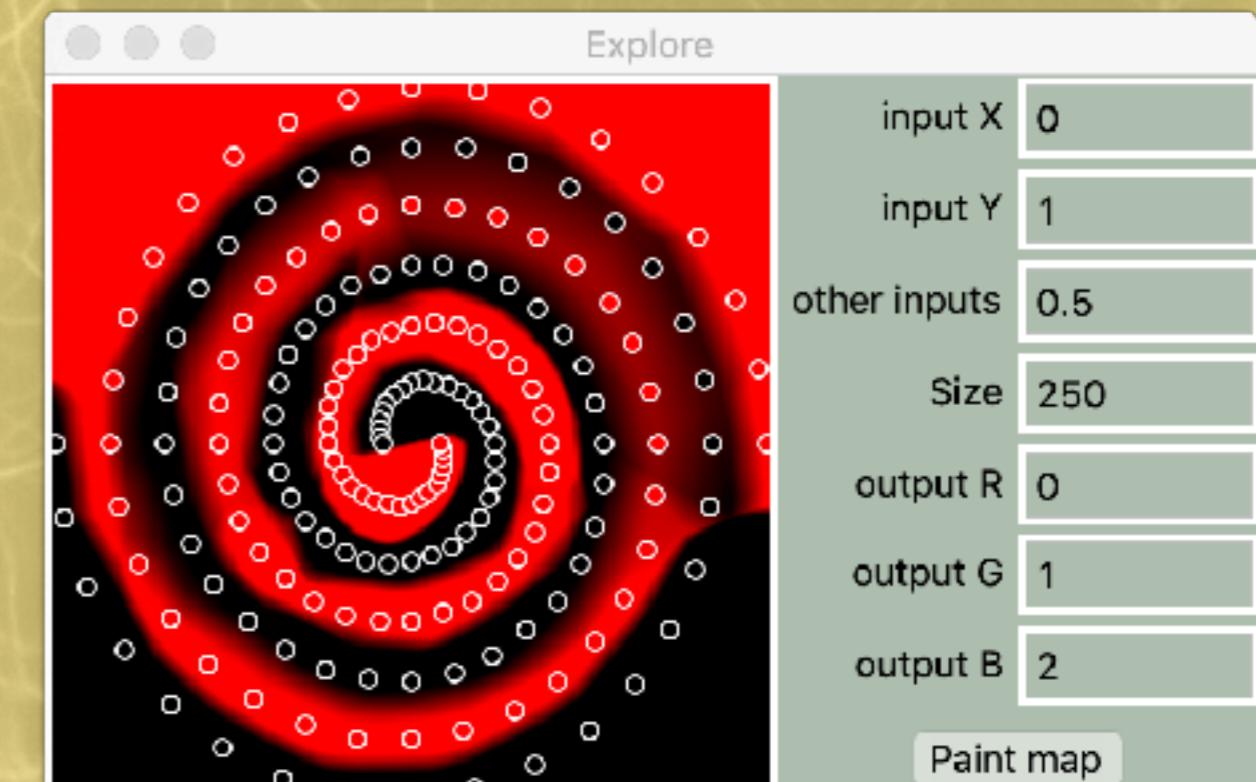
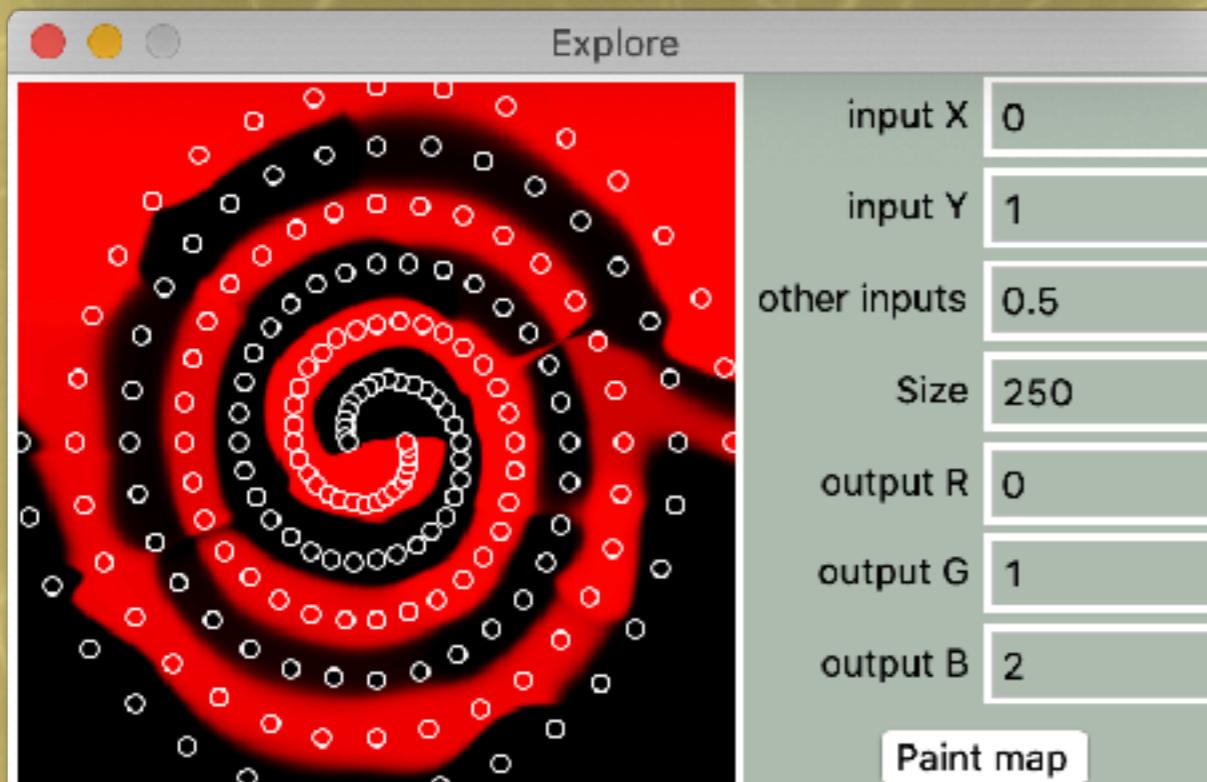
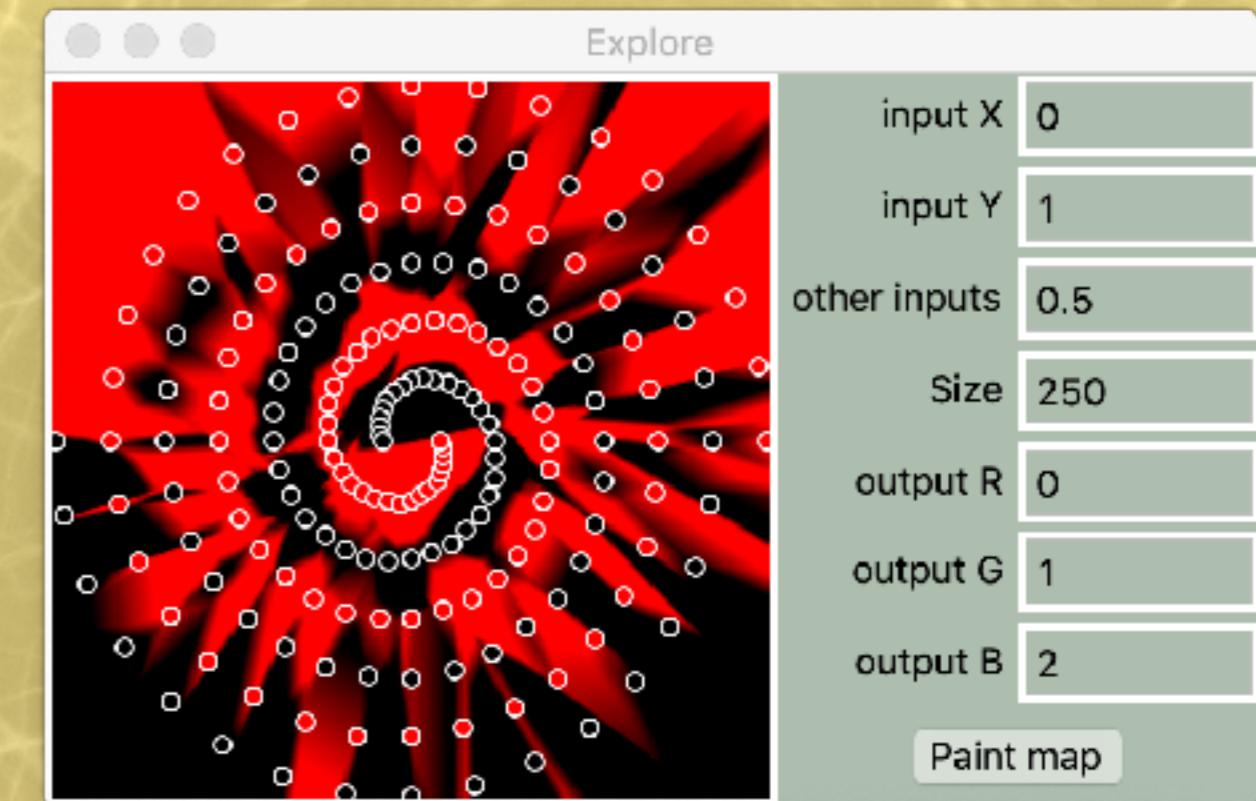
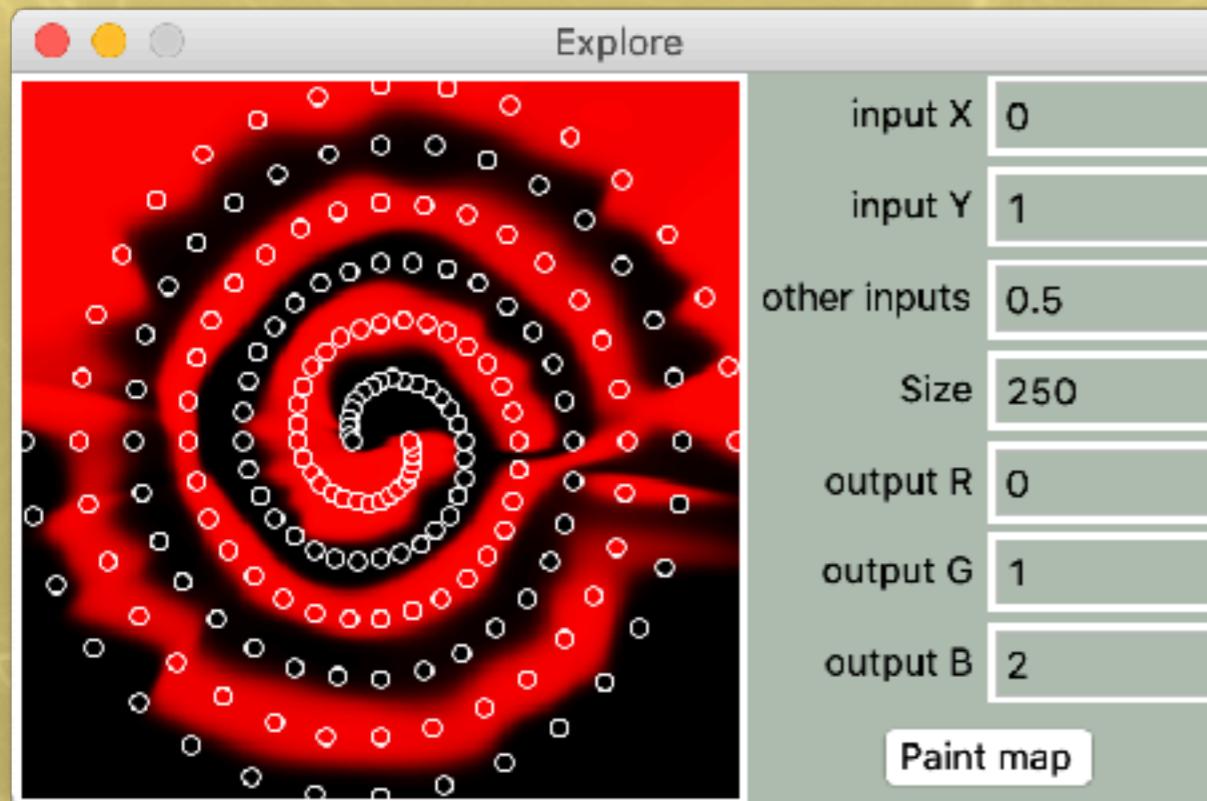
Efectos del dropout



Efectos del dropout



Efectos del dropout (sigmoidal y relu)



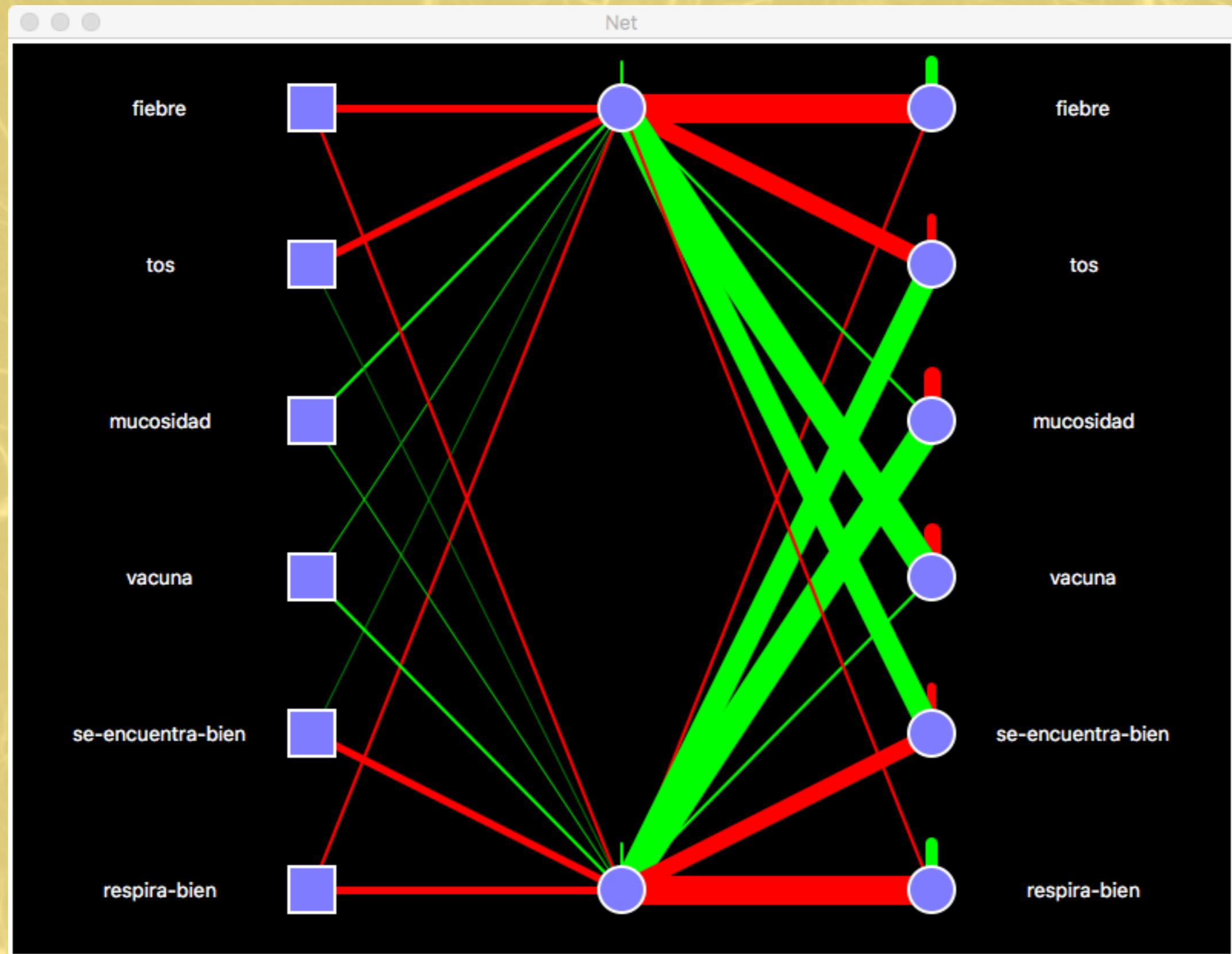
Nuevas arquitecturas

- * Deep auto-encoders
- * Convolutional Neural Networks (CNN)
- * Transposed CNN + Deep Autoencoders (AE y VAE)
- * Generative Adversarial Networks (GAN)
- * Redes recurrentes: LSTM, GRU y Transformers
- * Modelos de difusión y LLMs
- * Cada vez más se ven como bloques 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
 - Se puede forzar a que varias modalidades de información compartan un mismo espacio latente (por ejemplo texto e imágenes) lo que abre nuevas posibilidades, como generar imágenes a partir de texto

Ejemplo de autoencoder



Deep Auto-encoders

- * Los auto-encoders pueden constar de muchas capas encoder y muchas decoder
 - Las capas pueden ser de cualquier tipo, pero si hay capas convolucionales su contrapartida son las convoluciones transpuestas
 - Aumentan la generalización
- * 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

Deep Auto-encoders

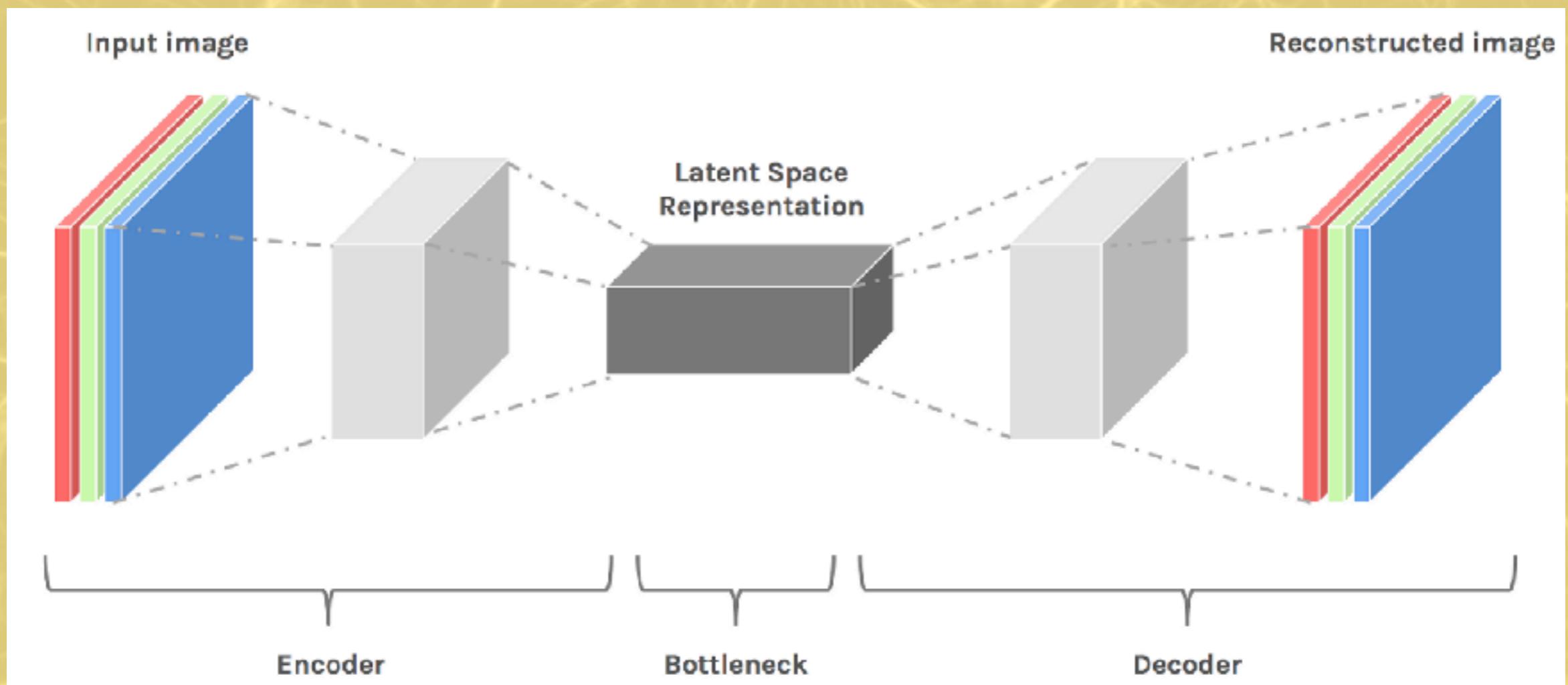
- * Podemos quedarnos con la primera mitad del autoencoder y añadir a continuación una o varias capas full-connected normales
- * O bien podemos quedarnos con la segunda mitad para darle un uso generativo
 - En estas redes, si colocamos ruido a la entrada se genera a la salida algo con sentido
 - También se usan para generar una imagen de mayor resolución a partir de una de menor resolución
- * O bien podemos quedarnos el autoencoder completo
 - Se usa para detección de anomalías
 - En este caso se procesa el ejemplo, y si la reconstrucción es mala estamos ante una anomalía

Autoencoders y sparse data

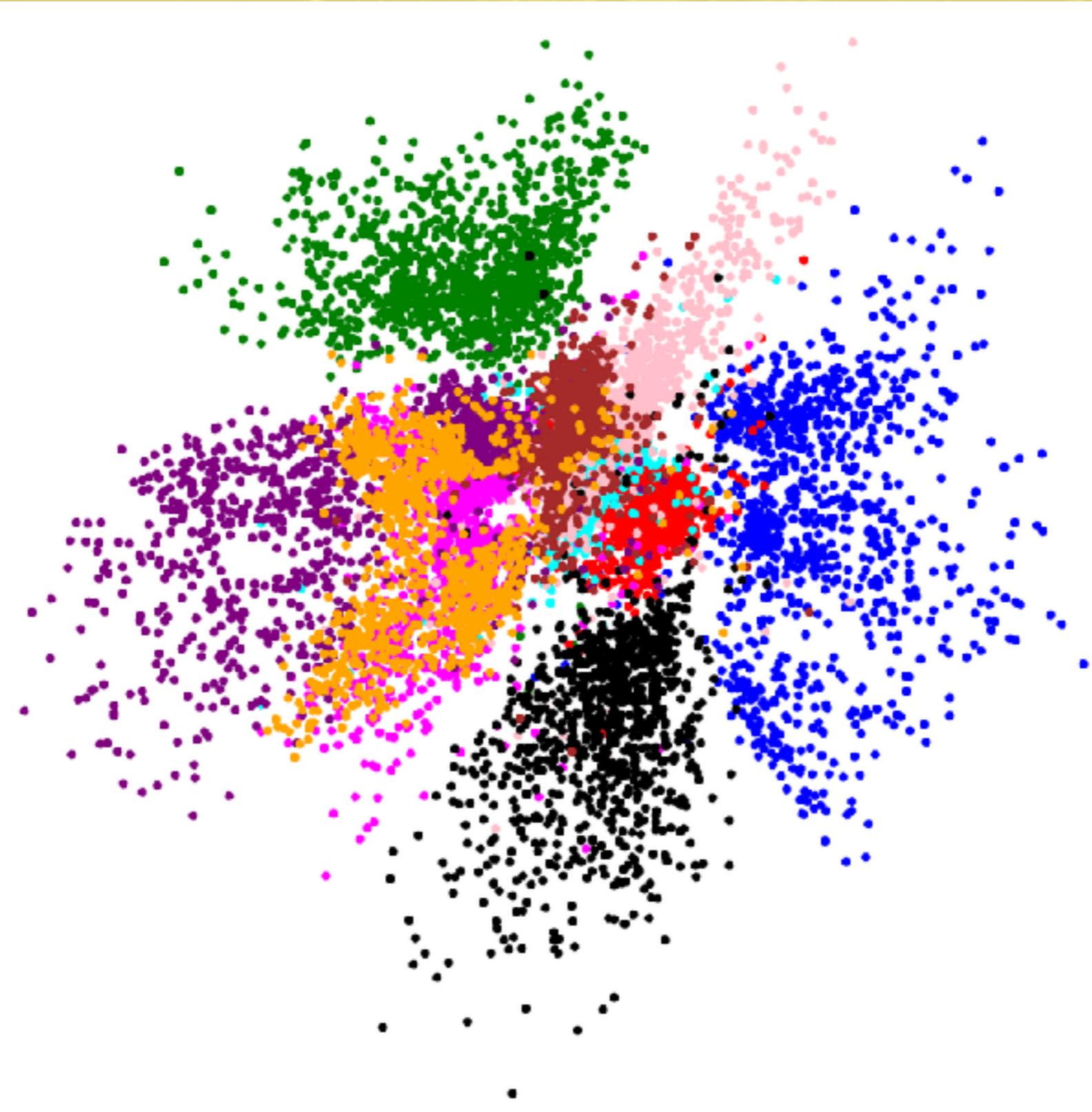
- * Imaginemos una red que tiene como entradas palabras que aparecen en un texto
 - Una entrada por cada posible palabra
 - Dado un texto, habrá mucho ceros a la entrada (todas las palabras que NO están)
 - Parece natural pensar que algunas palabras se parecen entre sí, es decir están más cerca de otras 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

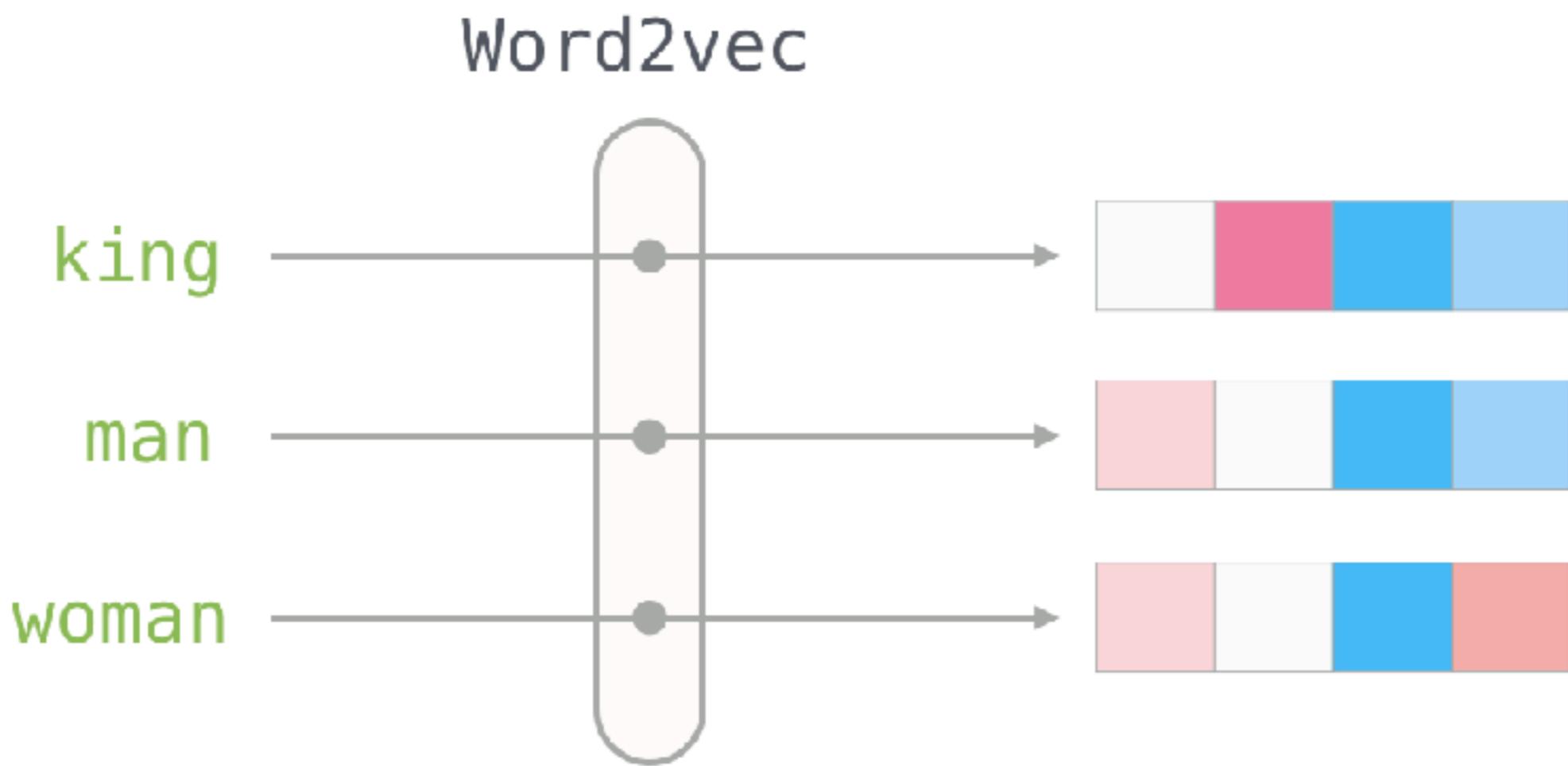
- * Es un concepto muy importante para muchas aplicaciones



Latent space

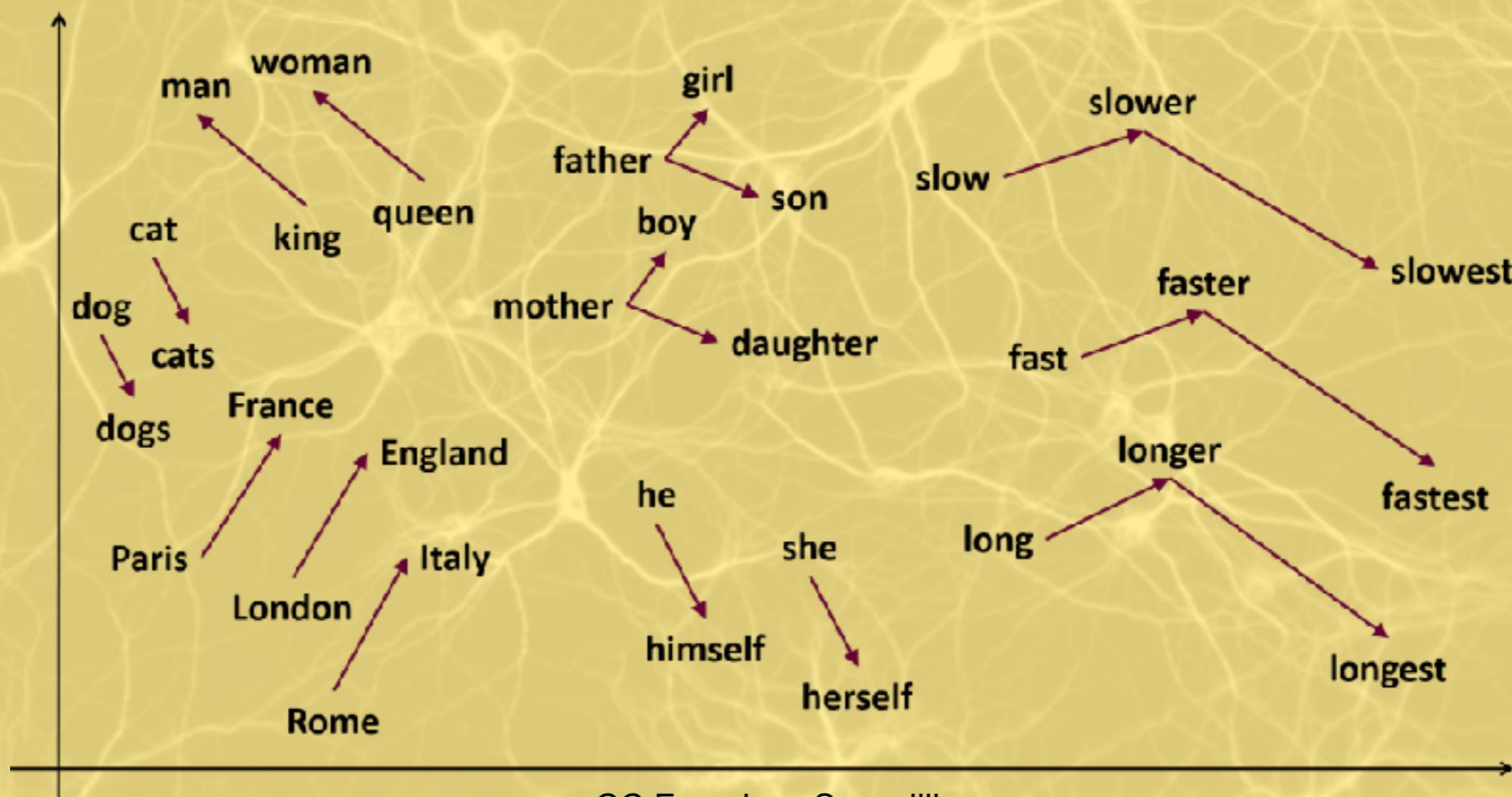


Autoencoders en NLP



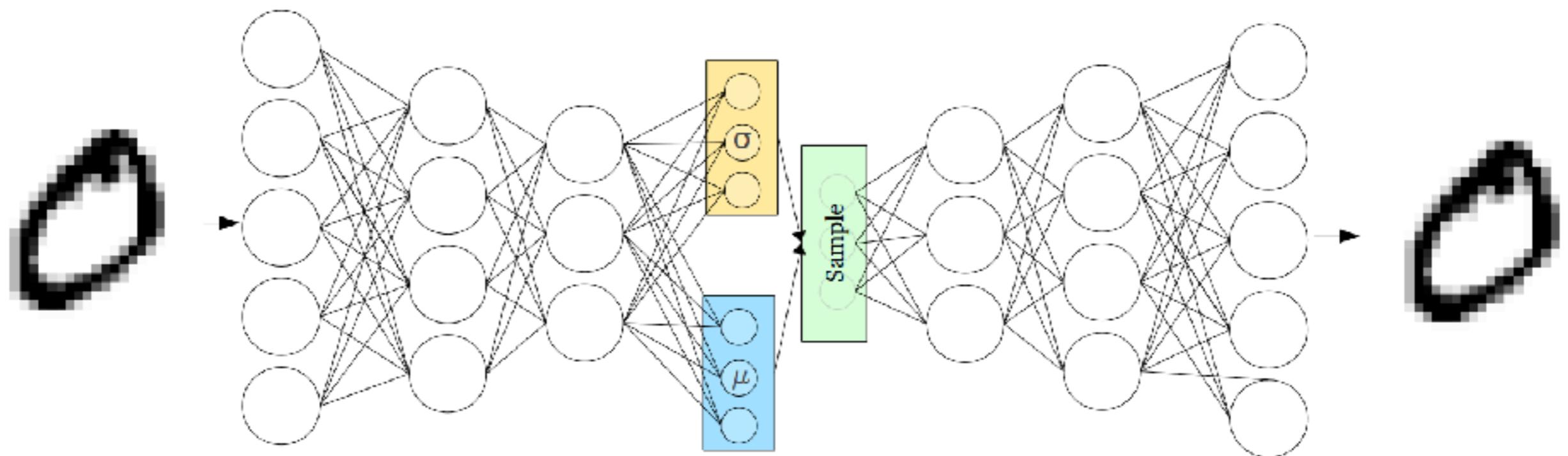
Autoencoders y sparse data

- * Para texto existen ya autoencoders descargables, por ejemplo word2vec, que transforman las palabras en un espacio latente (en NLP se conocen como embedding), 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 calculation of a convolution operation. It shows three matrices: the input matrix I , the kernel matrix K , and the output matrix $I * K$. The input matrix I is a 7x7 grid of binary values. The kernel matrix K is a 3x3 grid of values. The output matrix $I * K$ is a 5x5 grid of values. Dotted lines connect the highlighted 3x3 submatrix in the top-left of I to the kernel K , and the resulting value in the top-left of the output matrix to the corresponding element in $I * K$. The result is a 5x5 matrix where each element is the sum of the products of the corresponding elements from the input submatrix and the kernel.

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

I

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

K

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

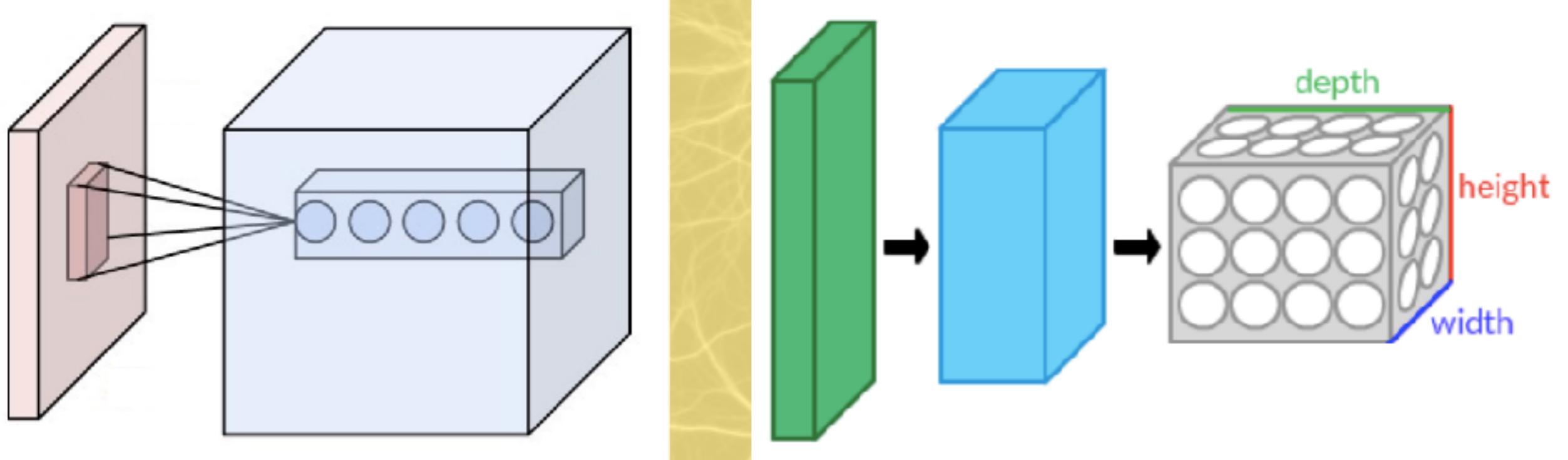
$I * K$

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

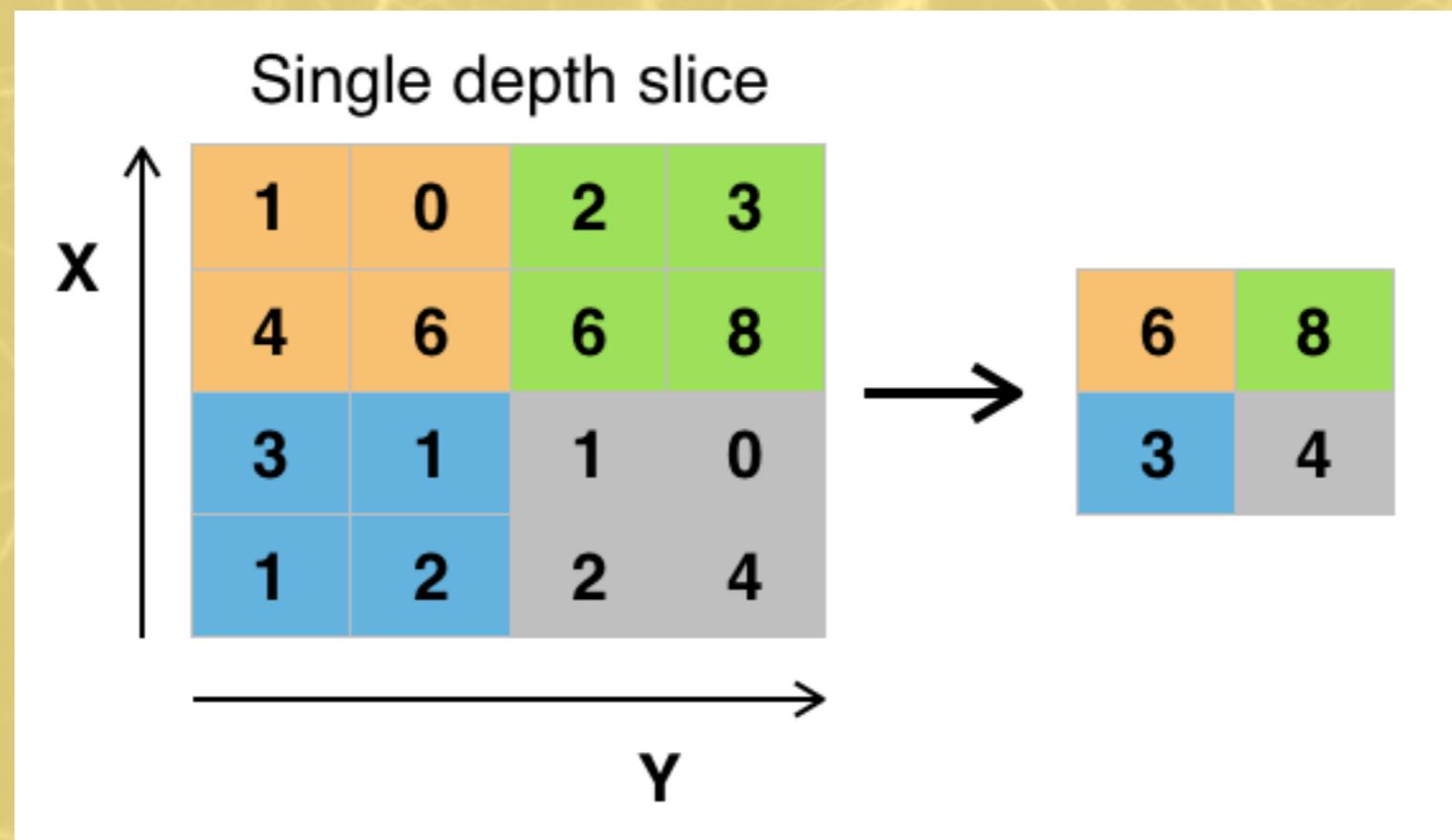
Campo receptivo



- * Width y Height: tamaño del campo receptivo, ej. 5x5, 25x25
- * Depth: número de diferentes convoluciones (filtros) 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

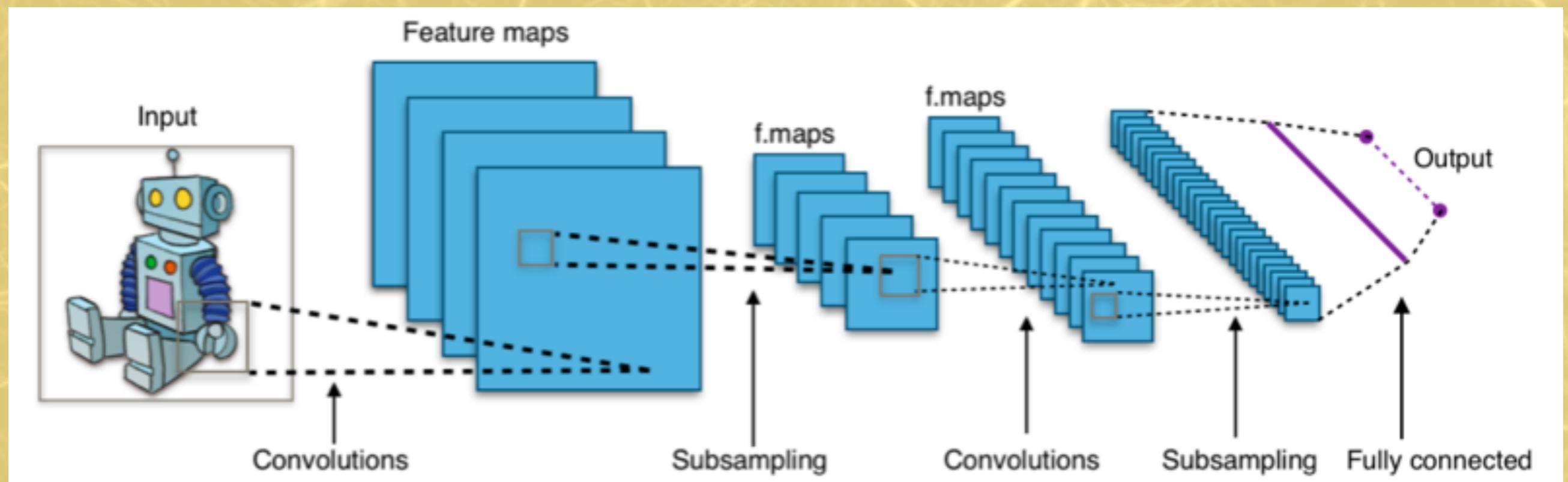
- INPUT → FC (clasificador lineal)
- INPUT → CONV → RELU → FC
- INPUT → [CONV → RELU → POOL]*2 → FC → RELU → FC (una capa de convolución antes de cada capa POOL)
- INPUT → [CONV → RELU → CONV → RELU → POOL]*3 → [FC → RELU]*2 → FC (dos convoluciones antes de cada POOL)

* 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 y en pytorch, 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

```
model = Sequential()  
model.add(<layer>)  
...  
...
```
- * 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)

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

Mini red 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'))
```

Mini red de convolución en pytorch

```
import torch
import torch.nn as nn

#create model
model = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1,
              padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1,
              padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(32 * 8 * 8, 128),
    nn.ReLU(),
    nn.Linear(128, num_classes)
)
```

Optimizadores

- * Todos se basan en la idea del descenso del gradiente del error con respecto a los pesos de la red, es decir, calcular en qué dirección (qué modificación de pesos) hace que el error baje más rápido
 - Utilizan algunos conceptos adicionales
 - ✖ Momento de inercia: en qué dirección me moví en t-1
 - ✖ Modificación del factor de aprendizaje según haya ido el aprendizaje en las iteraciones anteriores
- * Referencia: [Understanding Optimizers and Learning Rates in TensorFlow](#)

Optimizadores

* Stochastic Gradient Decent (SGD)

- Descenso del gradiente tradicional, que puede aplicarse de modo incremental, en batch o en minibatch
- Admite el uso de momento de inercia:

* Adagrad (Adaptive Gradient)

- Usa un learning rate adaptativo para cada peso

* RMSProp (Root Mean Square Propagation)

- Igual que adagrad pero además hace que el learning rate vaya descendiendo con el tiempo, normalizando el gradiente por la media móvil del gradiente al cuadrado

* Adam

- Añade momentos de primer y segundo orden a RMSProp, usando la media móvil exponencial del gradiente (primer orden) y del gradiente al cuadrado (segundo orden)

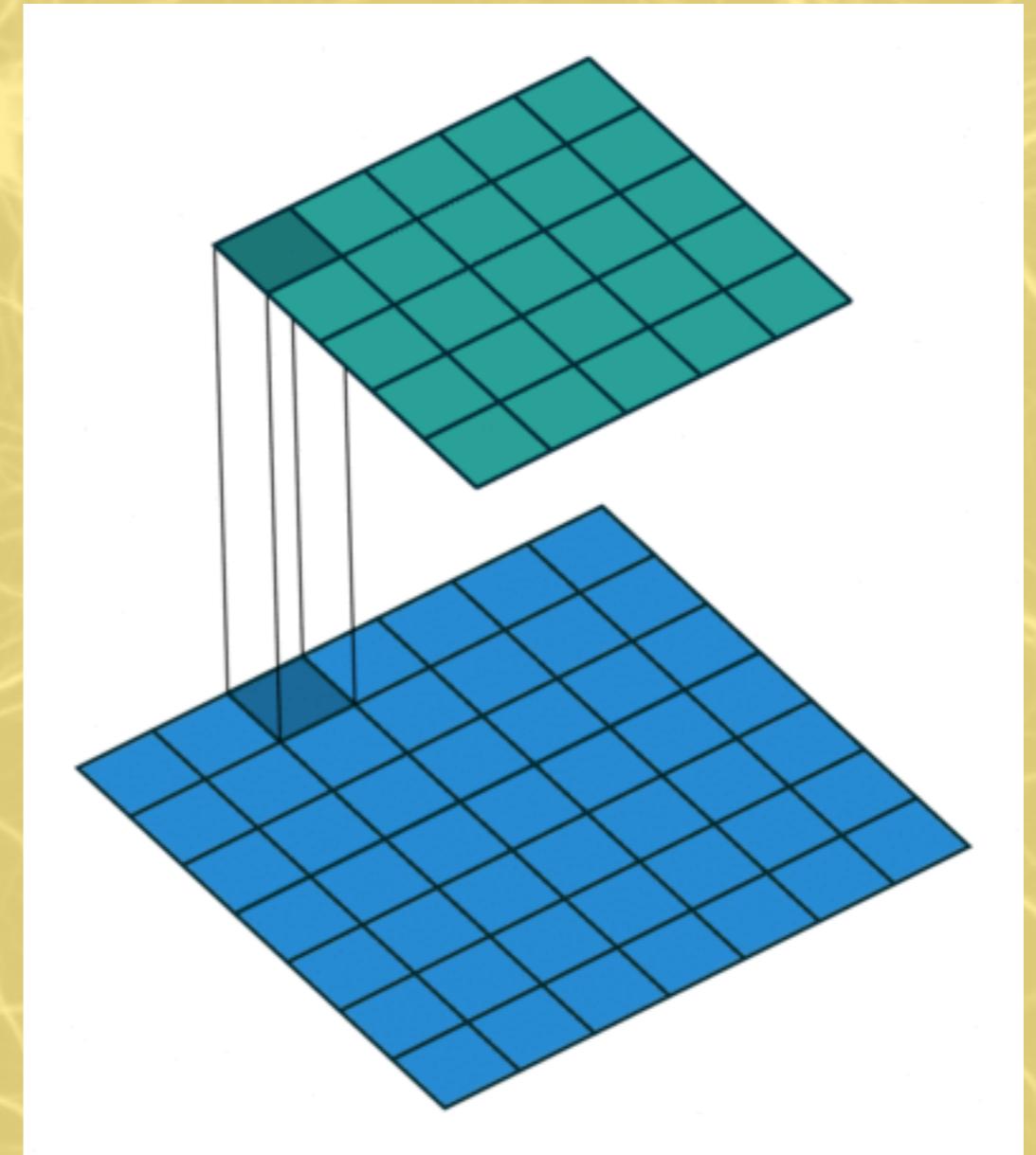
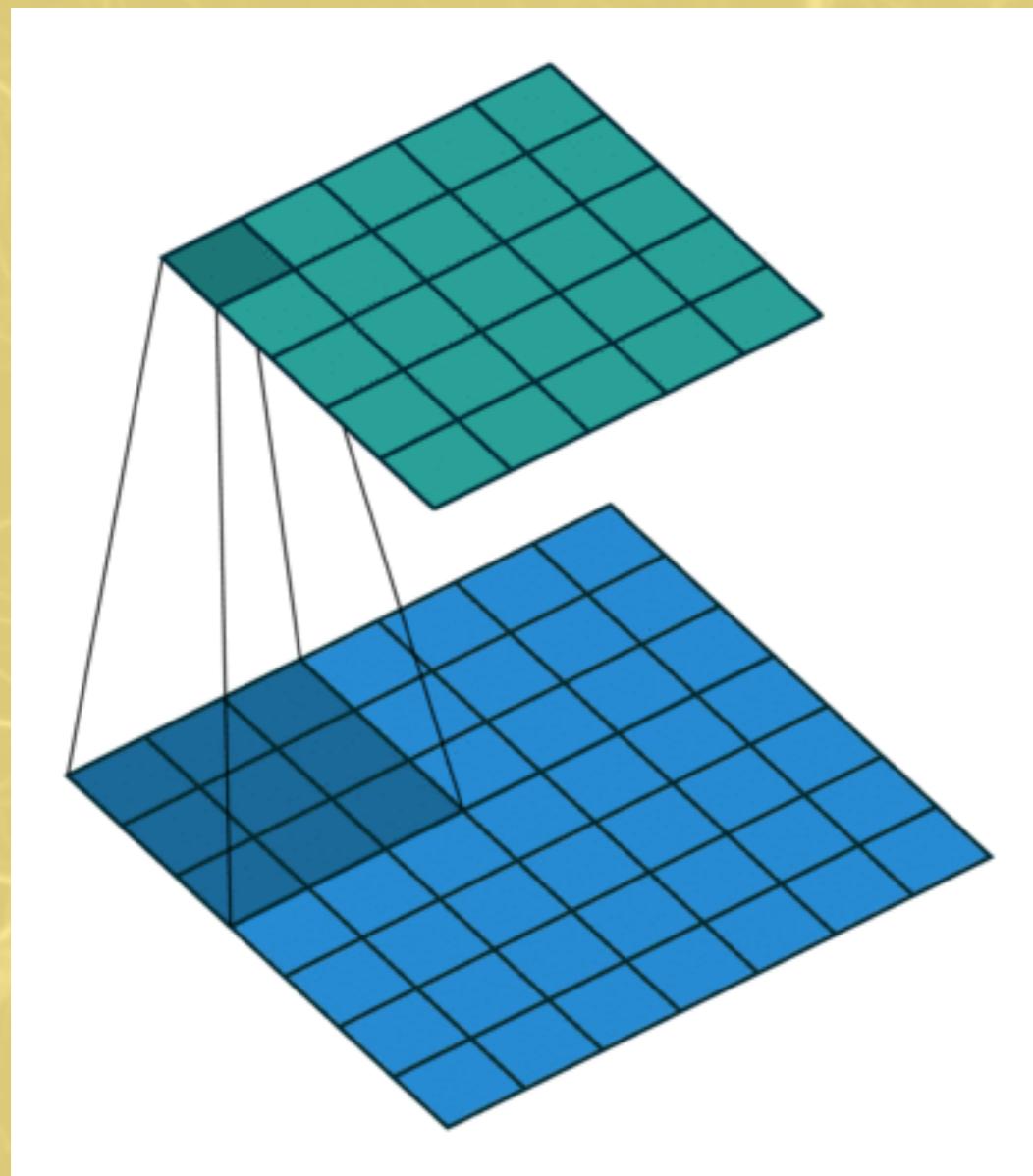
Conceptos avanzados

- * 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 y las suman
- * 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

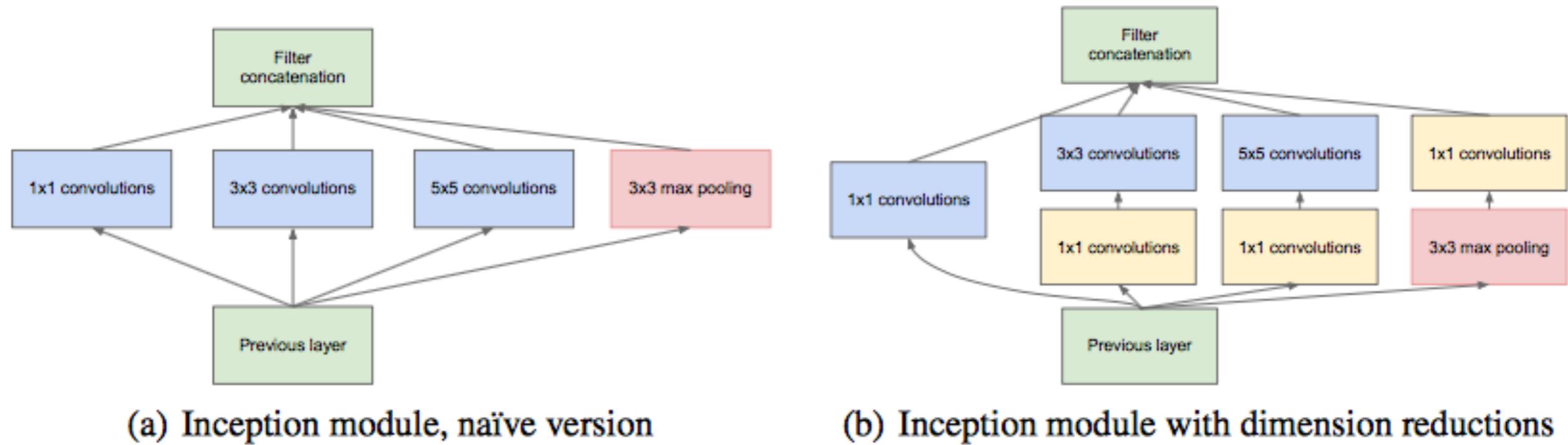
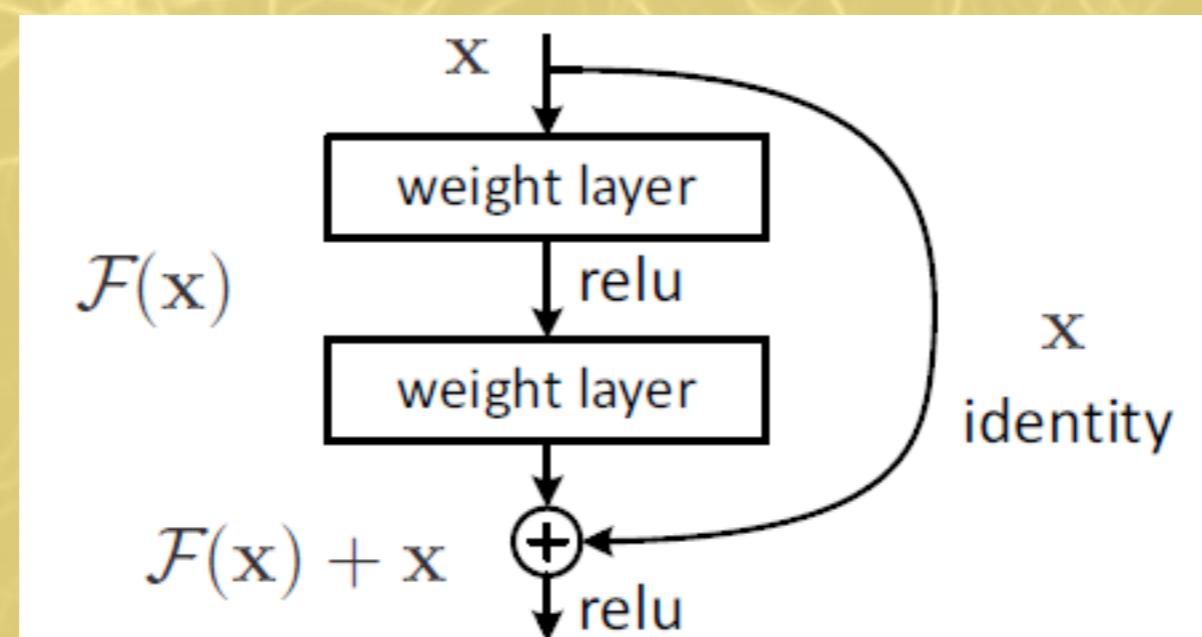


Figure 2: Inception module

Capas residuales

- * Las entradas se un grupo de conexiones se suman a las salidas
 - Esto mitiga el problema del desvanecimiento del gradiente
 - 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
 - ✖ otras opciones son la suma con padding 0 o la concatenación



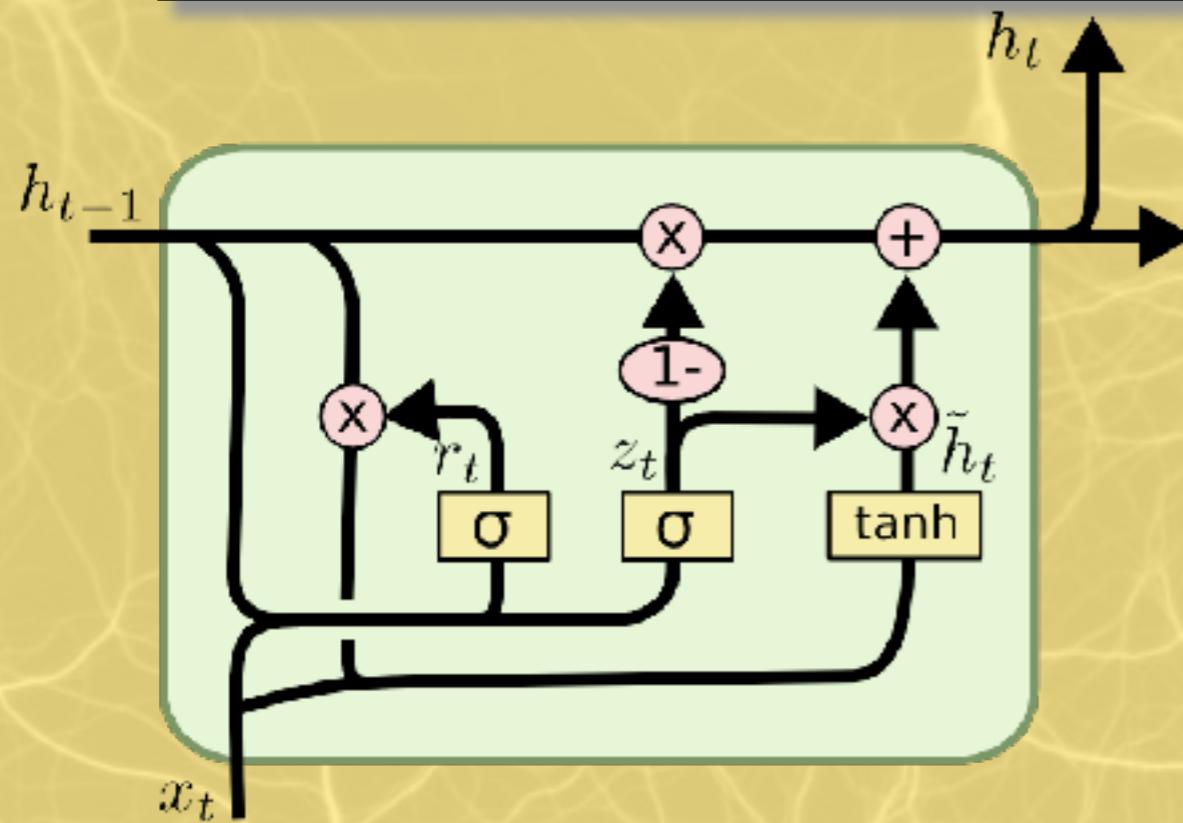
Transfer Learning

- * 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 (quitando los que había y añadiendo otros)
 - 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
 - Las últimas versiones de Keras ya traen una serie de redes clásicas pre-entrenadas

Redes recurrentes (RNN)

- * 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

- * [a, b] → concatenar
- * a * b → producto elemento a elemento
- * · → producto matricial
- * σ → 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.

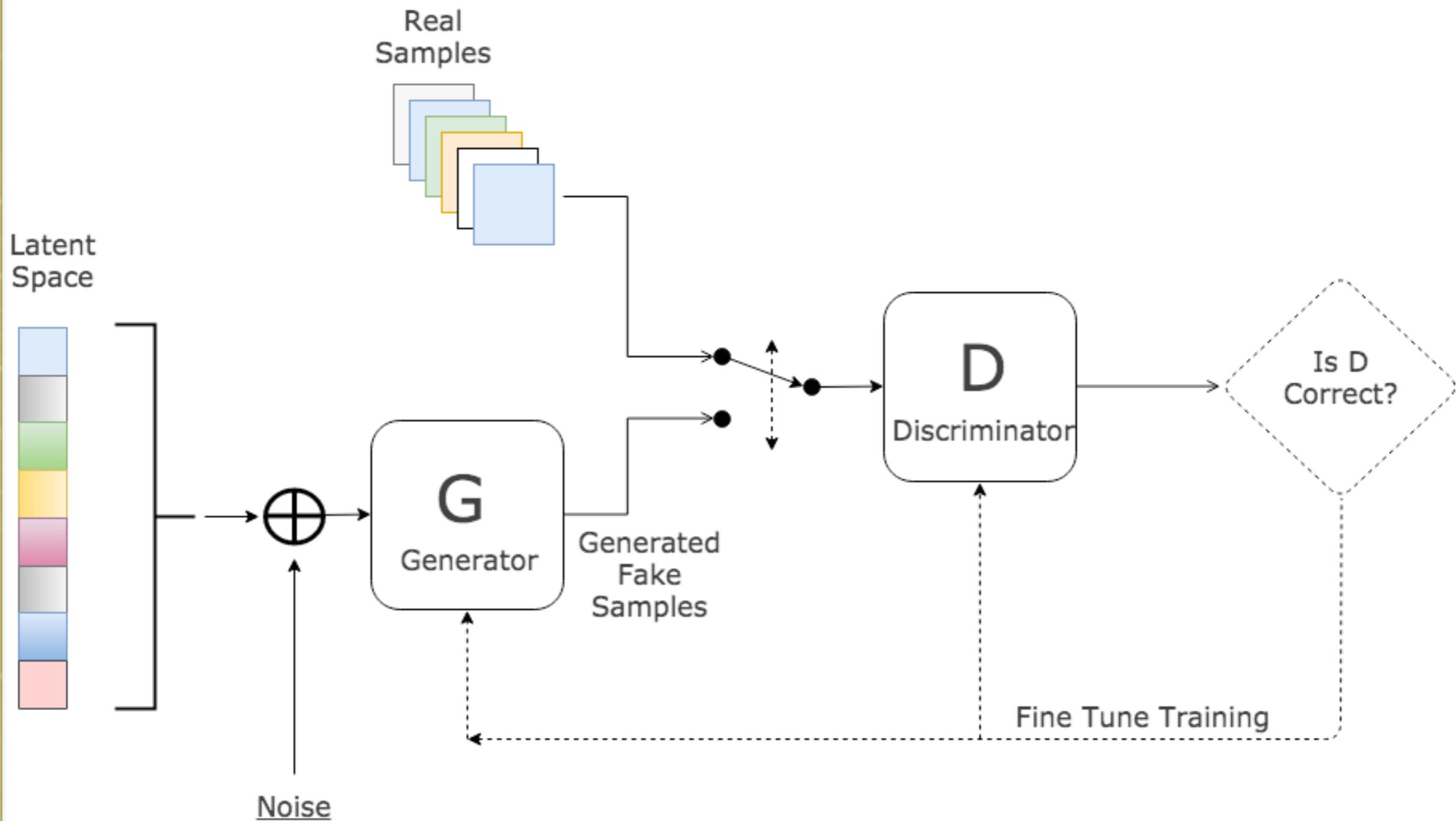
Tengo miedo.

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
 - El discriminador es una CNN clásica con 1 neurona de salida
 - El generador utiliza convoluciones transpuestas, que a partir de imágenes pequeñas va generando imágenes cada vez mayores
- * 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



Entrenamiento GAN

```
for epoch in range(epochs):
    for batch in range(int(len(X_train)/batch_size)):
        # Train the Discriminator
        noise = np.random.normal(0, 1, (batch_size, 100))
        gen_imgs = generator.predict(noise, verbose=0)
        imgs = X_train[batch*batch_size : (batch+1)*batch_size]

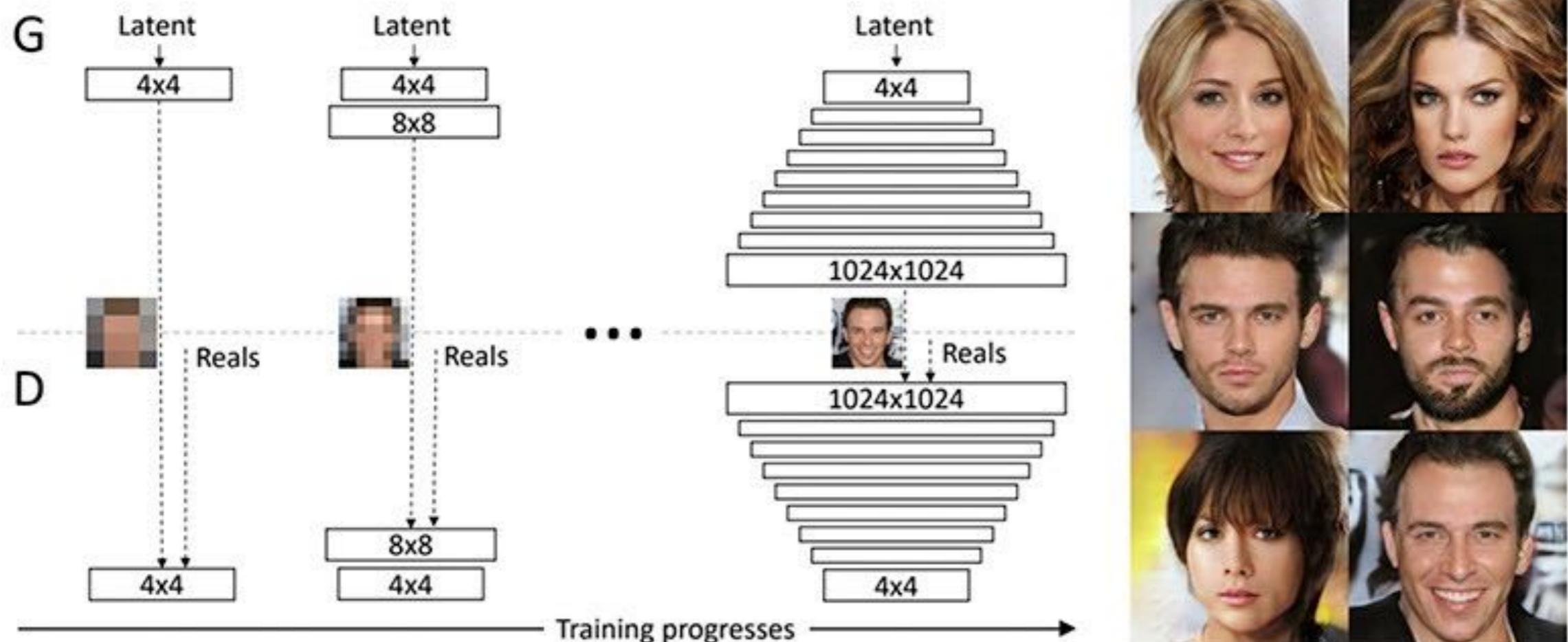
        d_loss_real = discriminator.train_on_batch(imgs, valid)
        d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Train the Generator
    # al construir combined: discriminator.trainable = False
    noise = np.random.normal(0, 1, (batch_size, 100))
    g_loss = combined.train_on_batch(noise, valid)
```

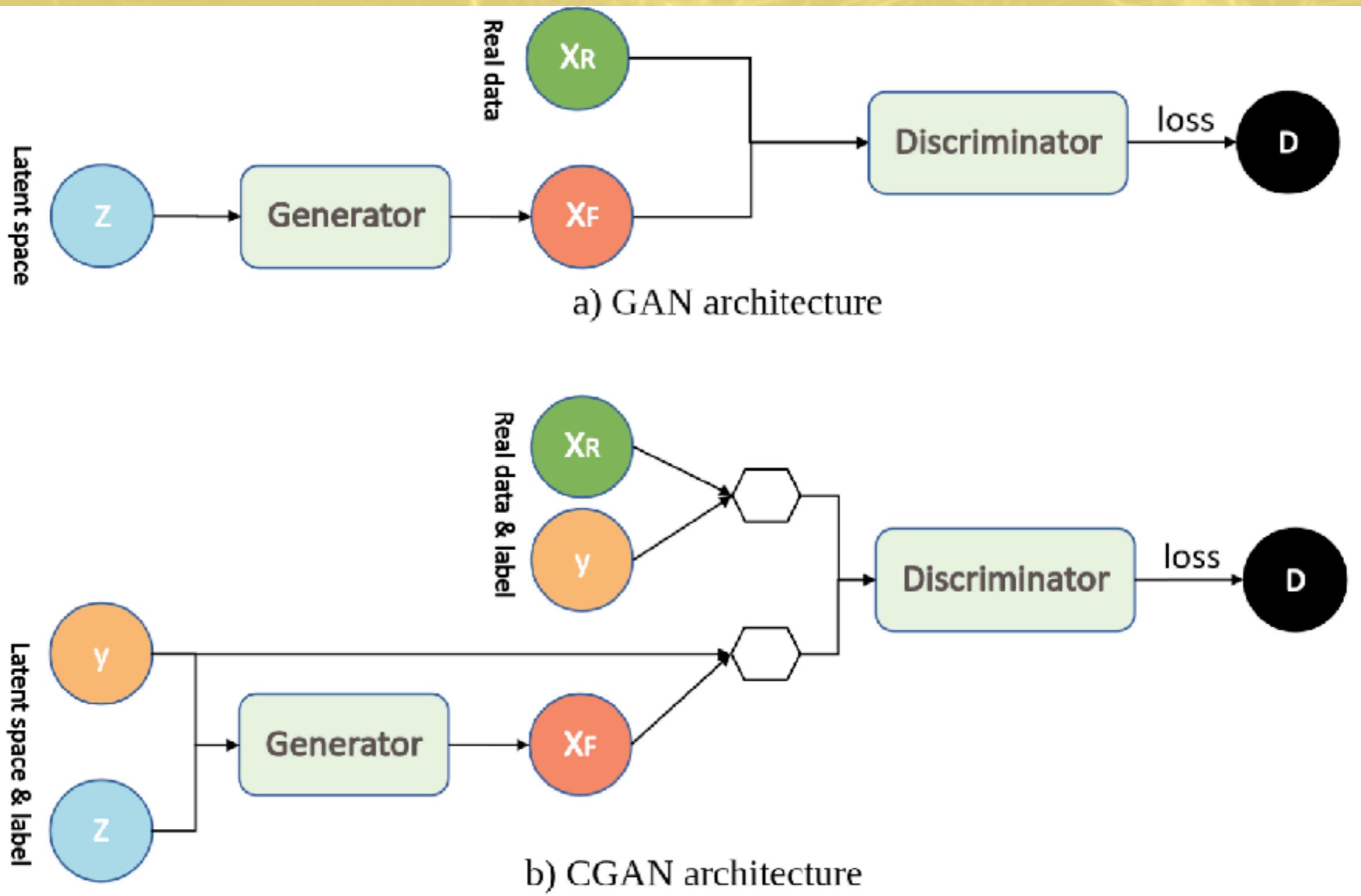
X

Ejemplo GAN

Photorealistic Image Generation Using GAN



Conditional GAN



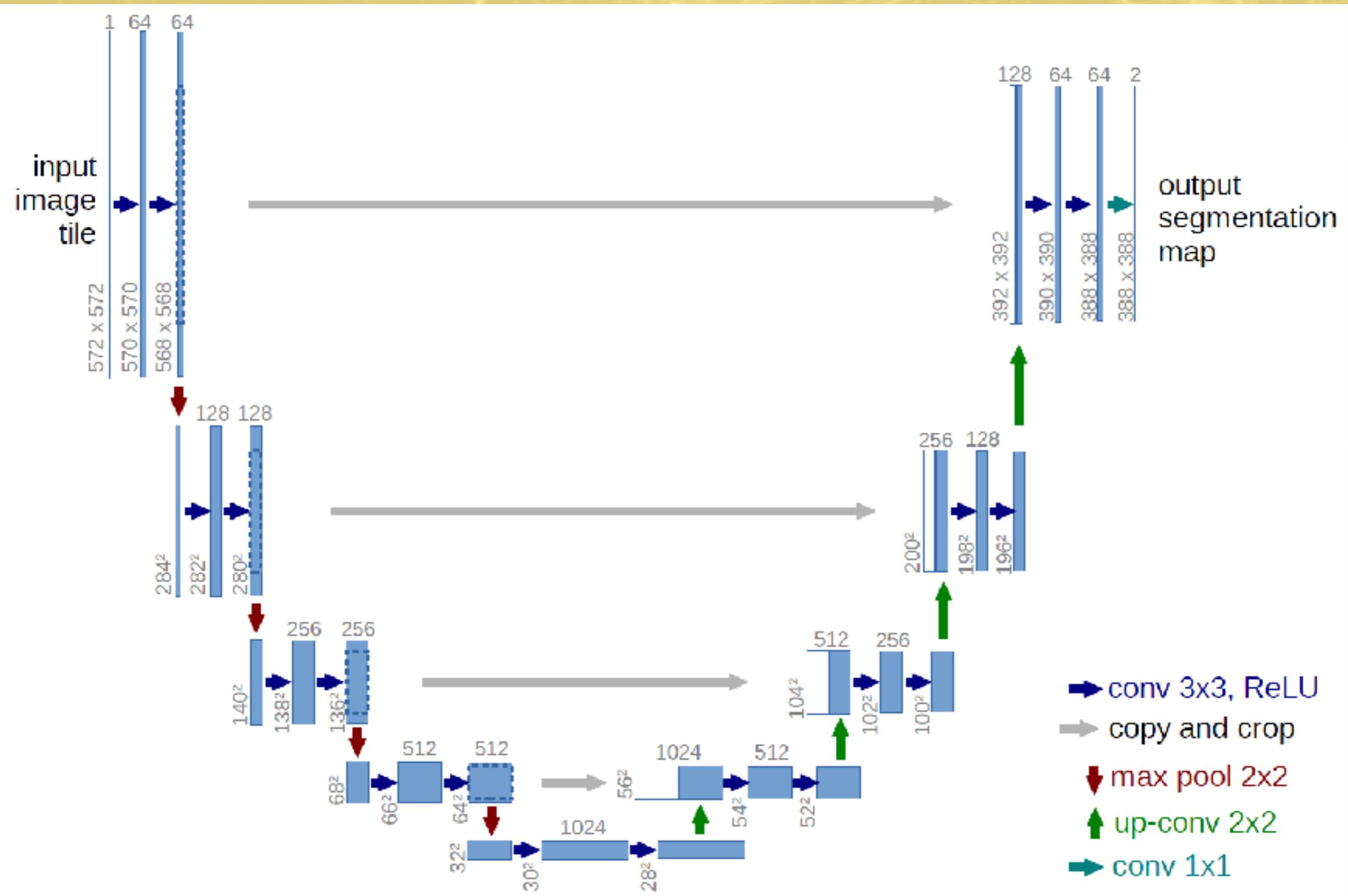
Generative Adversarial Networks (GAN)

* 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 discriminador son compensados por el generador
 - ✖ Si el loss de una de las dos redes sube 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

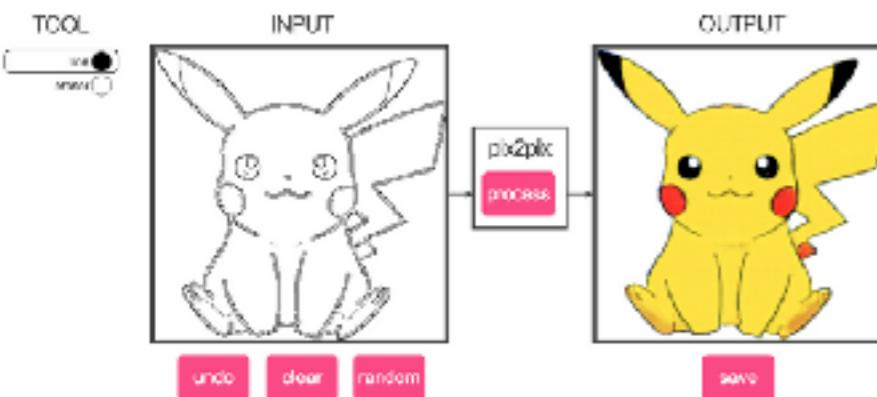
* <https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/>

UNet (Fischer y Brox, 2015)



Pix2pix networks

pix2pix edges2pikachu



Pix2Pix (①+②)

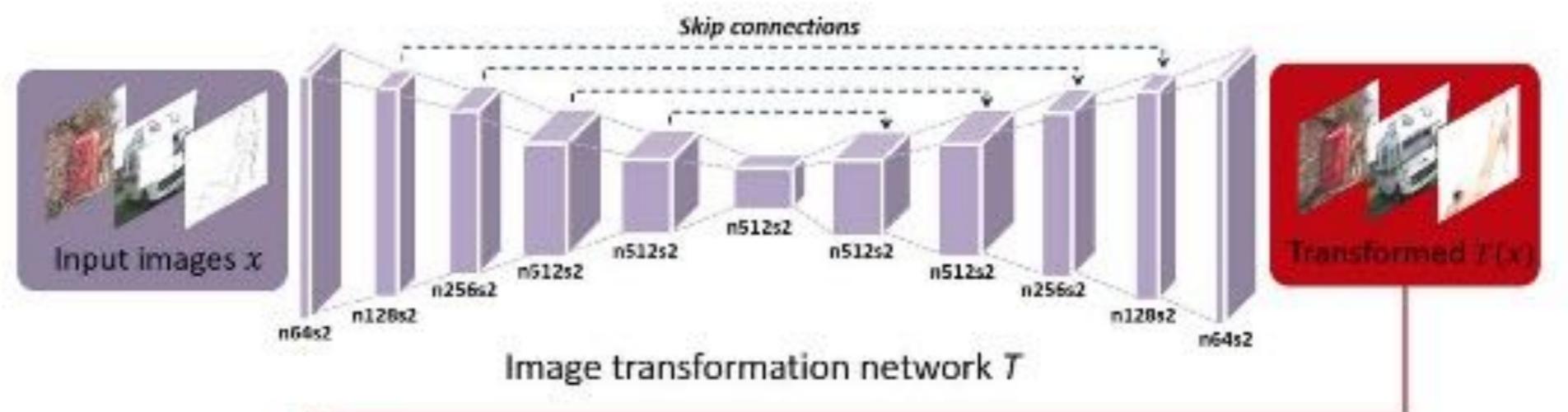
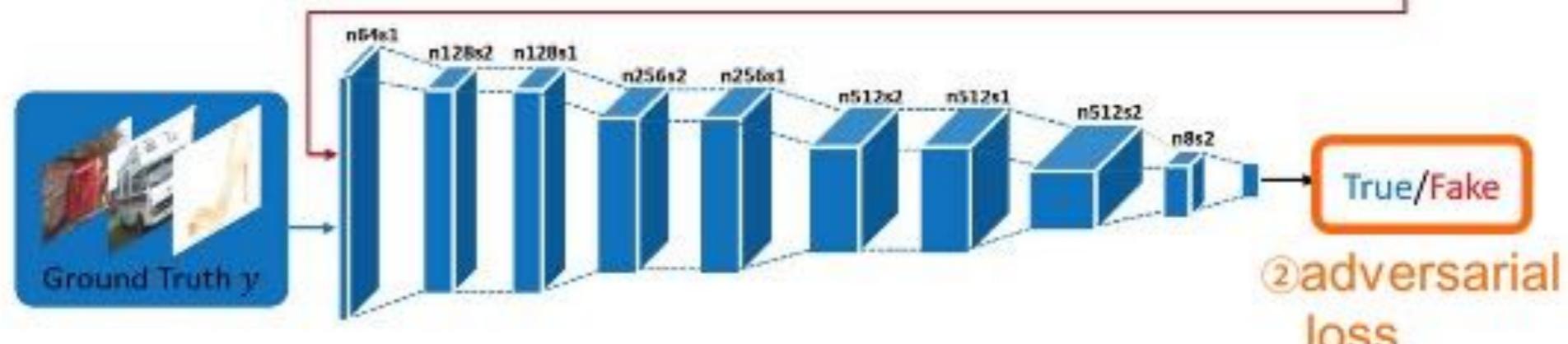


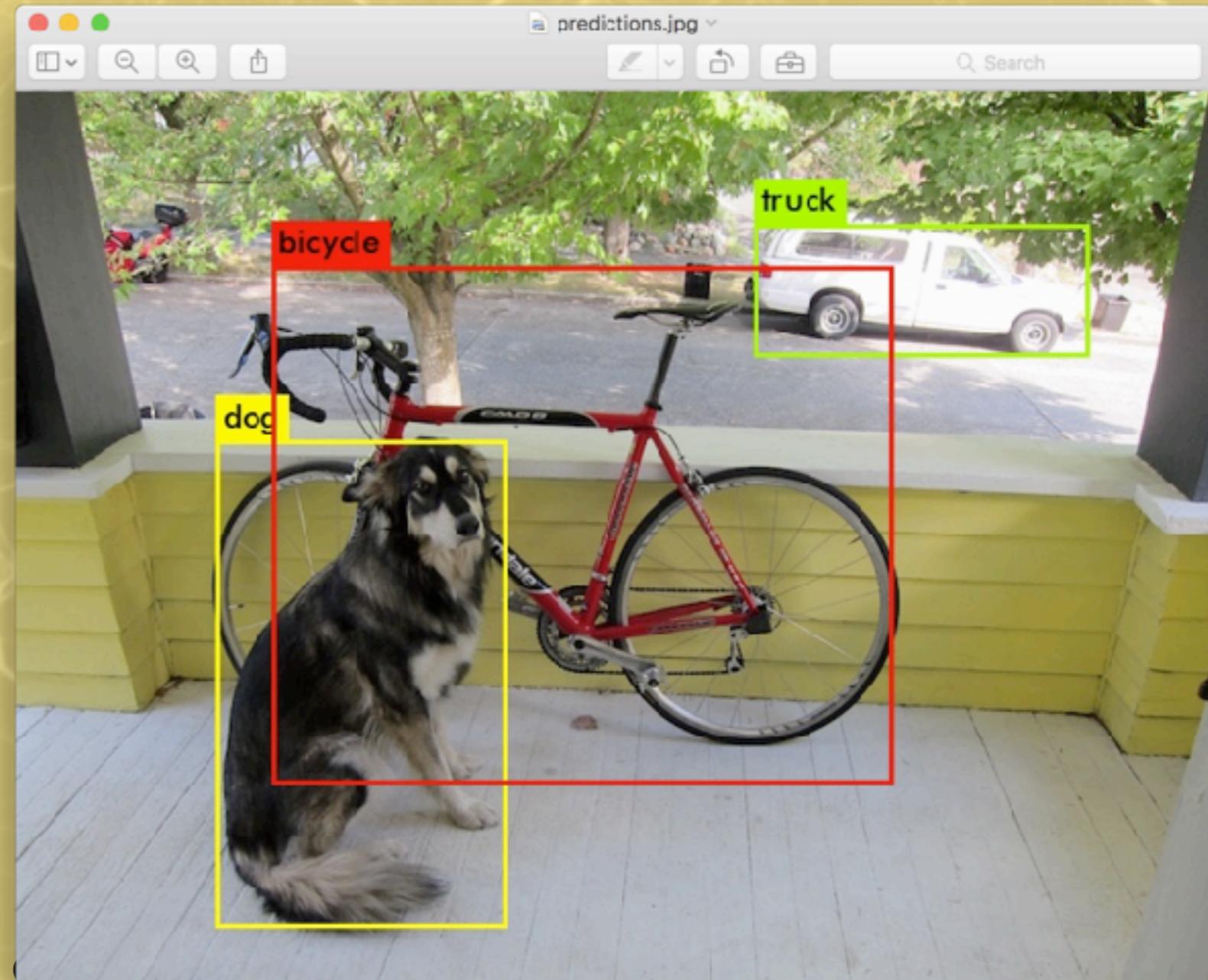
Image transformation network T

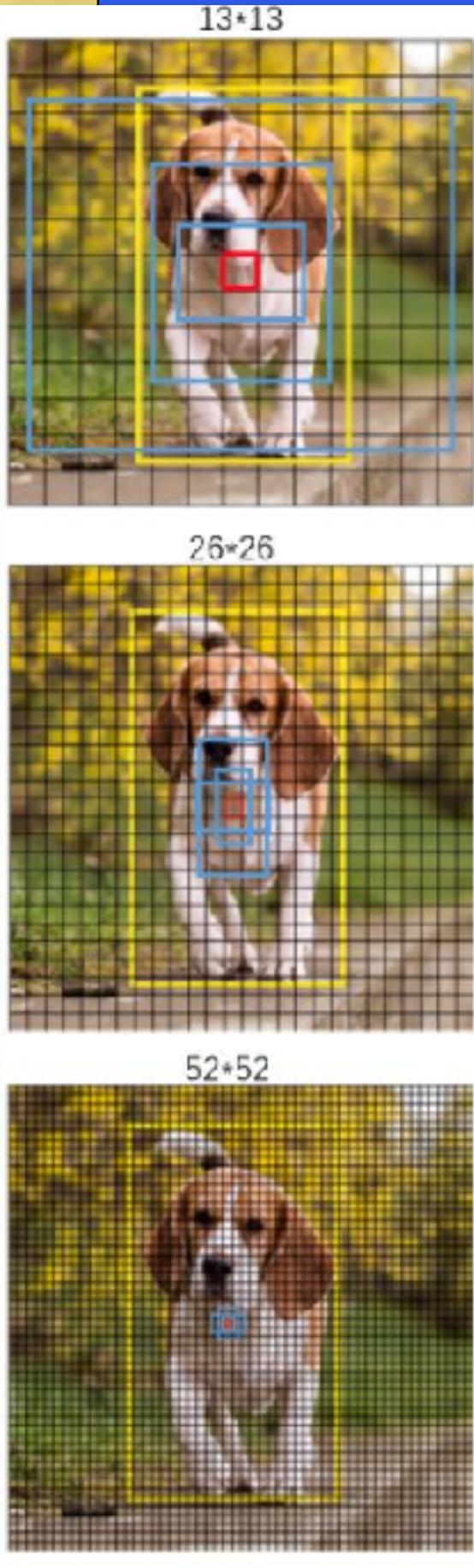


Discriminative network D

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

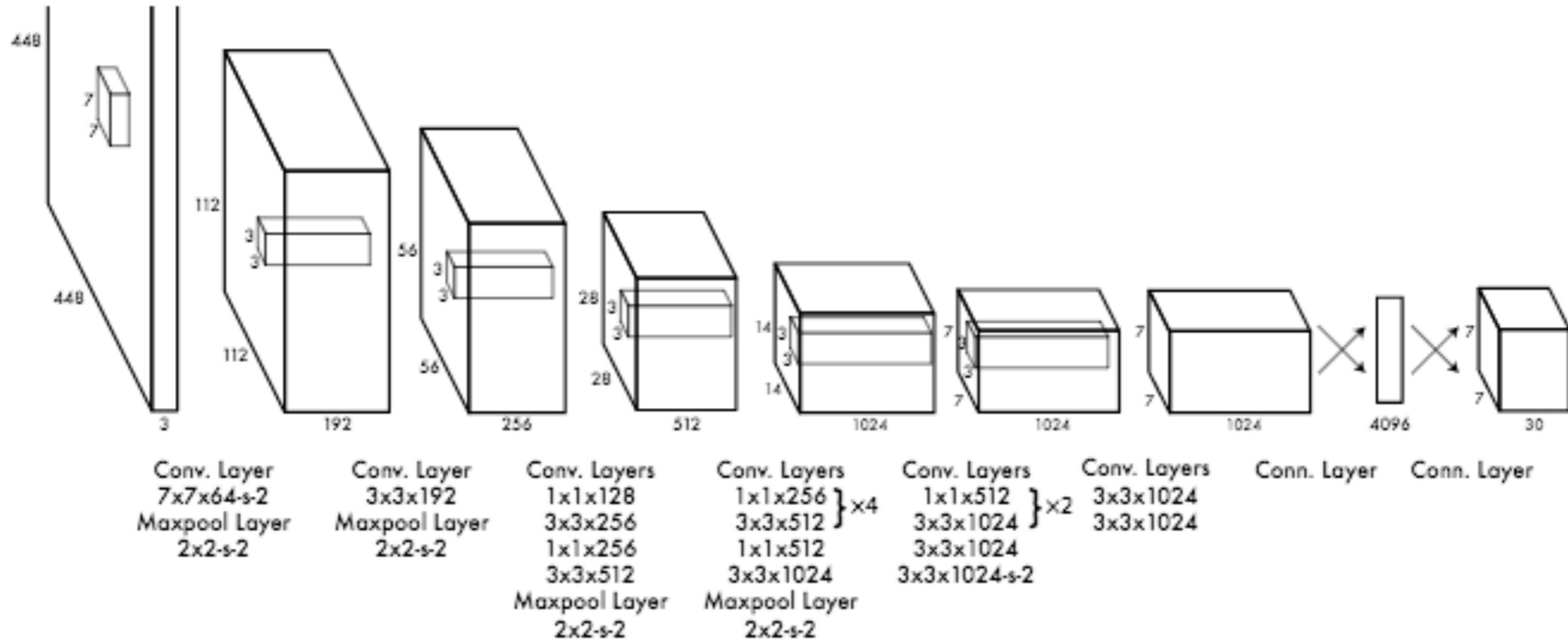




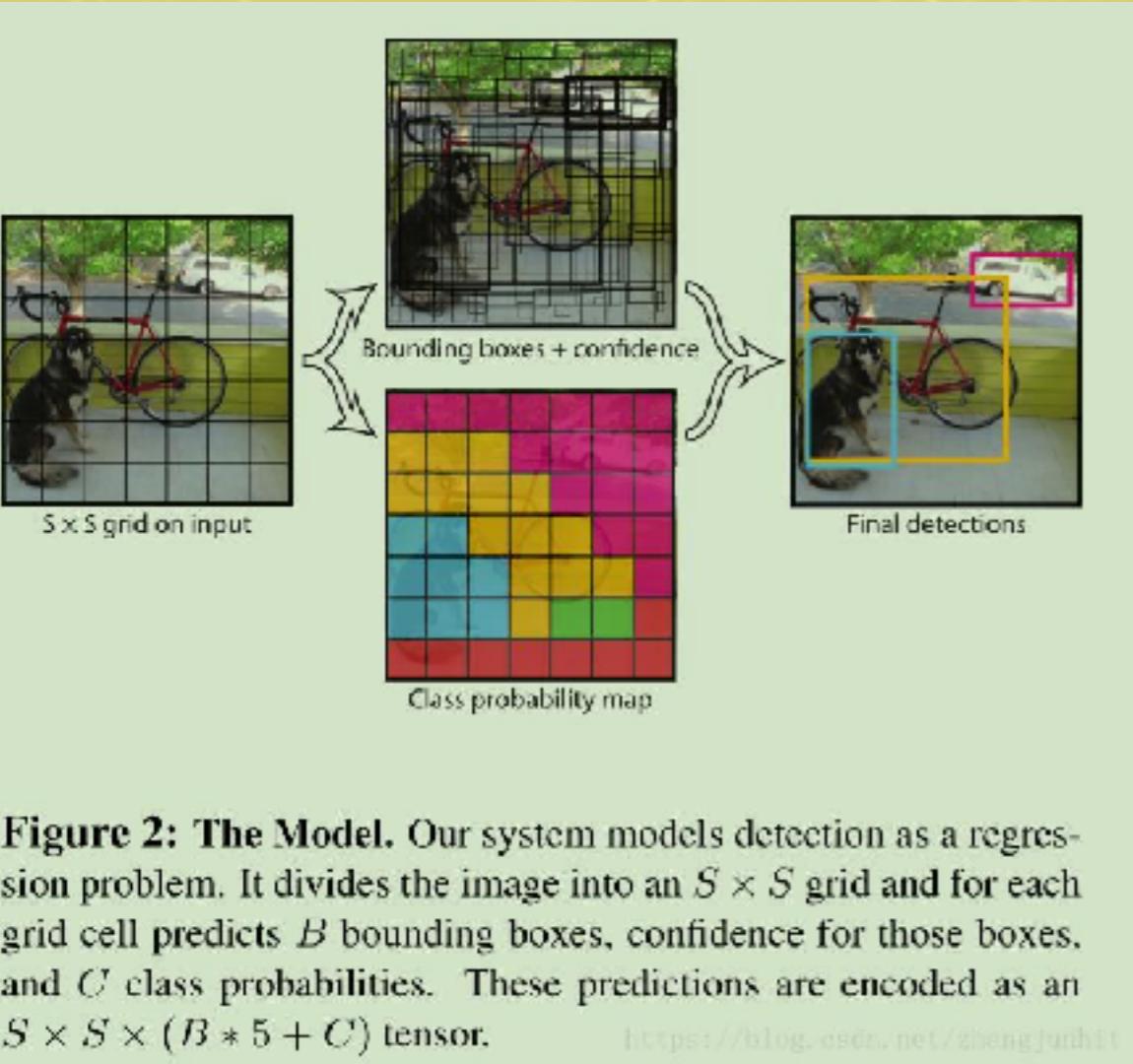
YOLO

- * 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



This parameterization fixes the output size

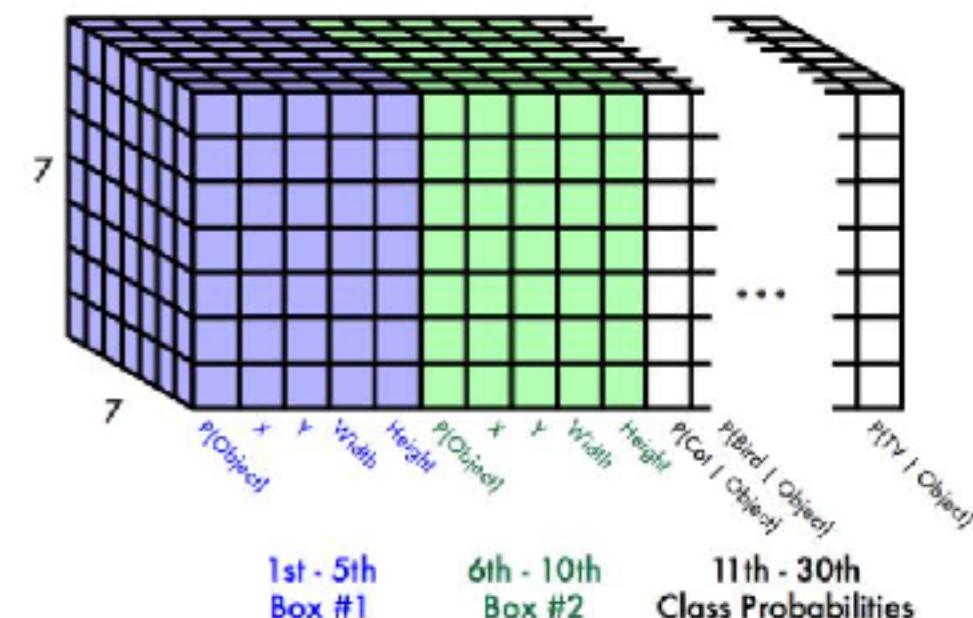
Each cell predicts:

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

For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

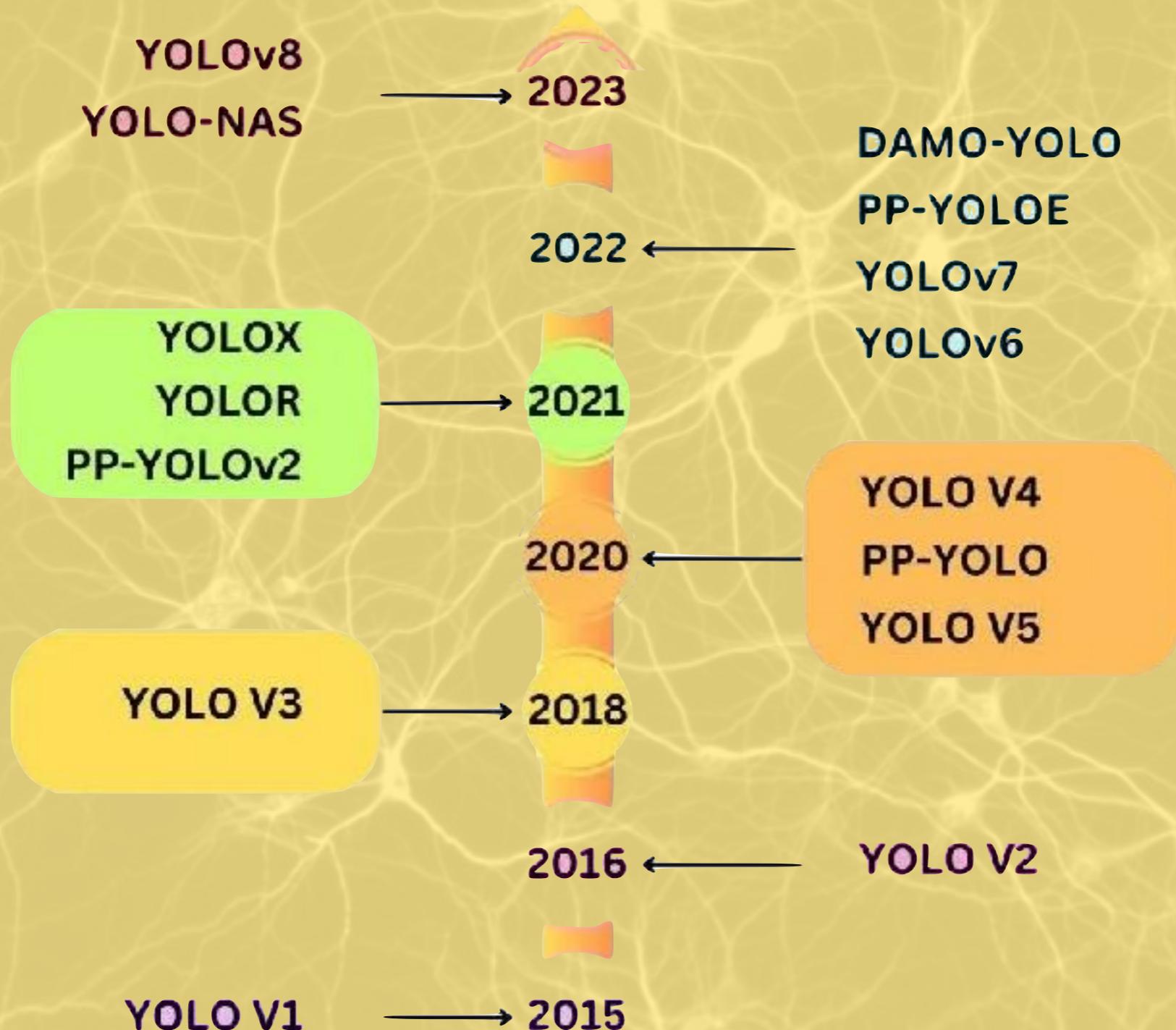
$$7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30 \text{ tensor} = \mathbf{1470 \text{ outputs}}$$



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

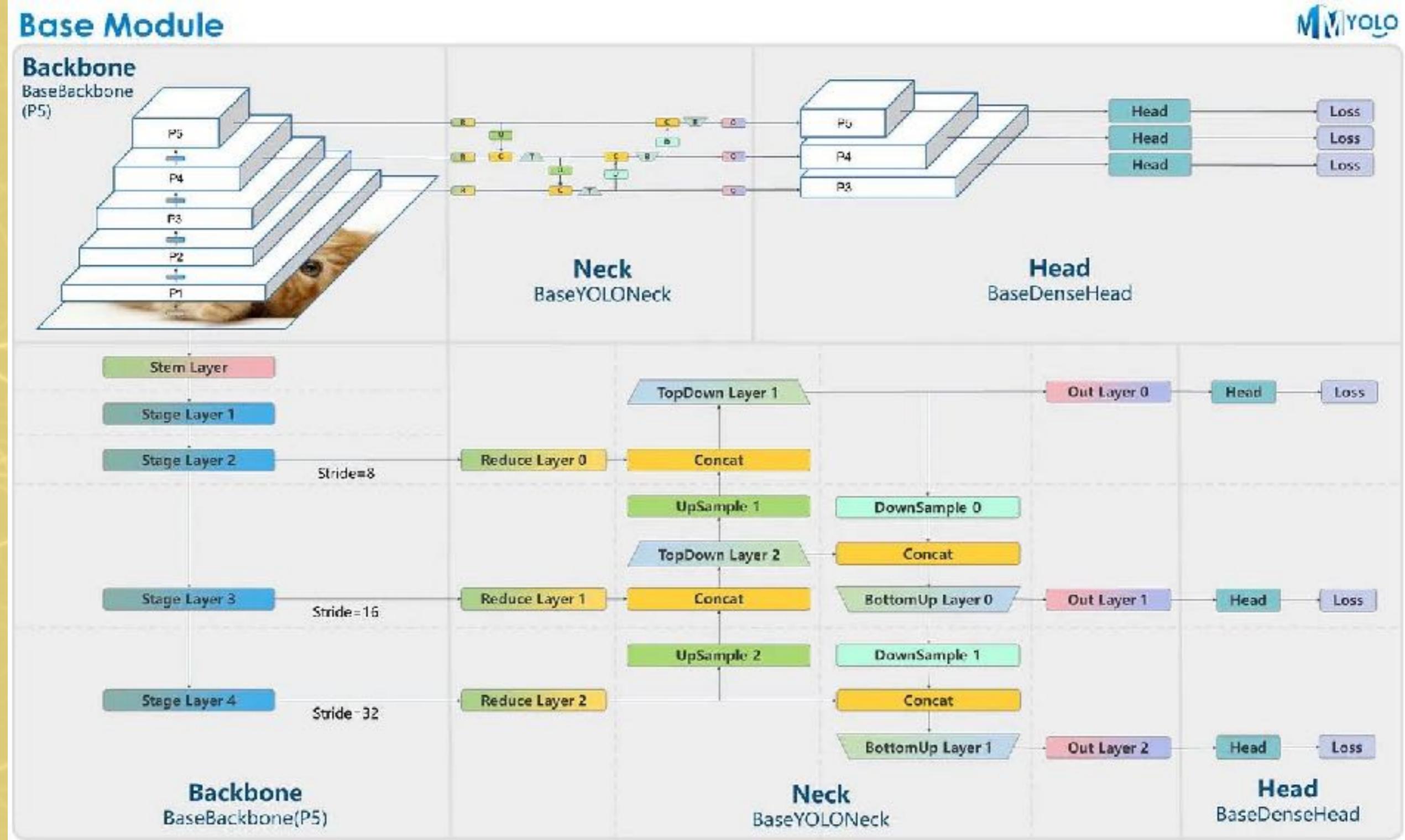
Versiones YOLO

The YOLO timeline



YOLO v8

Base Module



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

* Recursos en AICU / INSIA

- Nvidia GeForce GTX 3090 Ti
 - ✗ 10752 + 336 núcleos a 1560 MHz
 - ✗ 24GB GDDR6X a 21 GHz
 - ✗ Consumo: 450w
 - ✗ Rendimiento 40 TFLOPS
- Servidor en rack
 - ✗ 256 hilos de CPU, para problemas de optimización
 - ✗ NVIDIA RTX 4500 (24GB VRAM, 39.9 TFLOPS)
 - ✗ NVIDIA RTX 6000 (48 GB VRAM, 18,176, 568, 124 núcleos, 91.1 TFLOPS)

* Recursos 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) (RAM común)
 - ✗ 10,4 TFLOPS
- M1 Ultra
 - ✗ 2048 cores (64x8x4) (RAM común)

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

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%

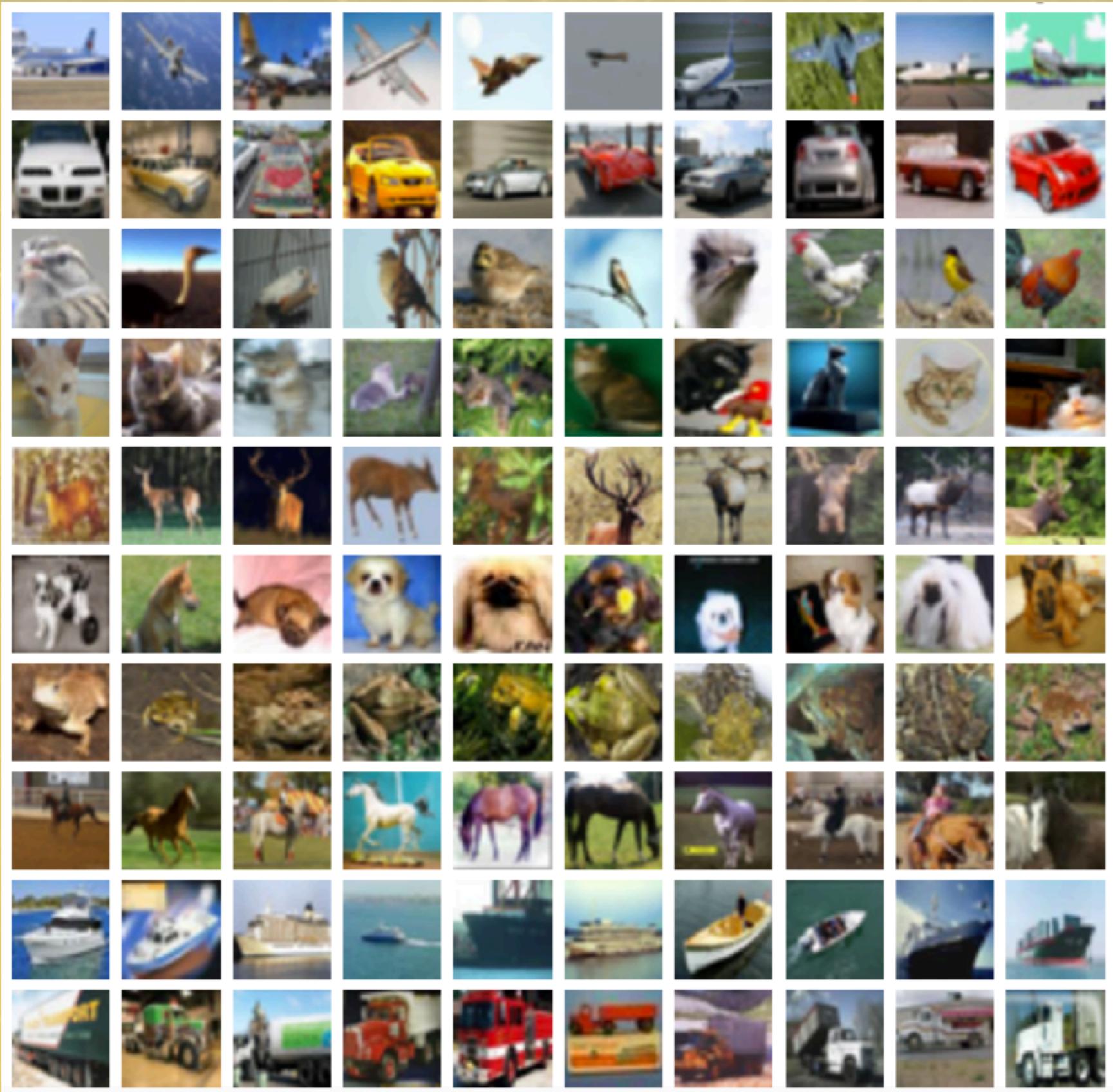
* CIFAR-100

- 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/>)

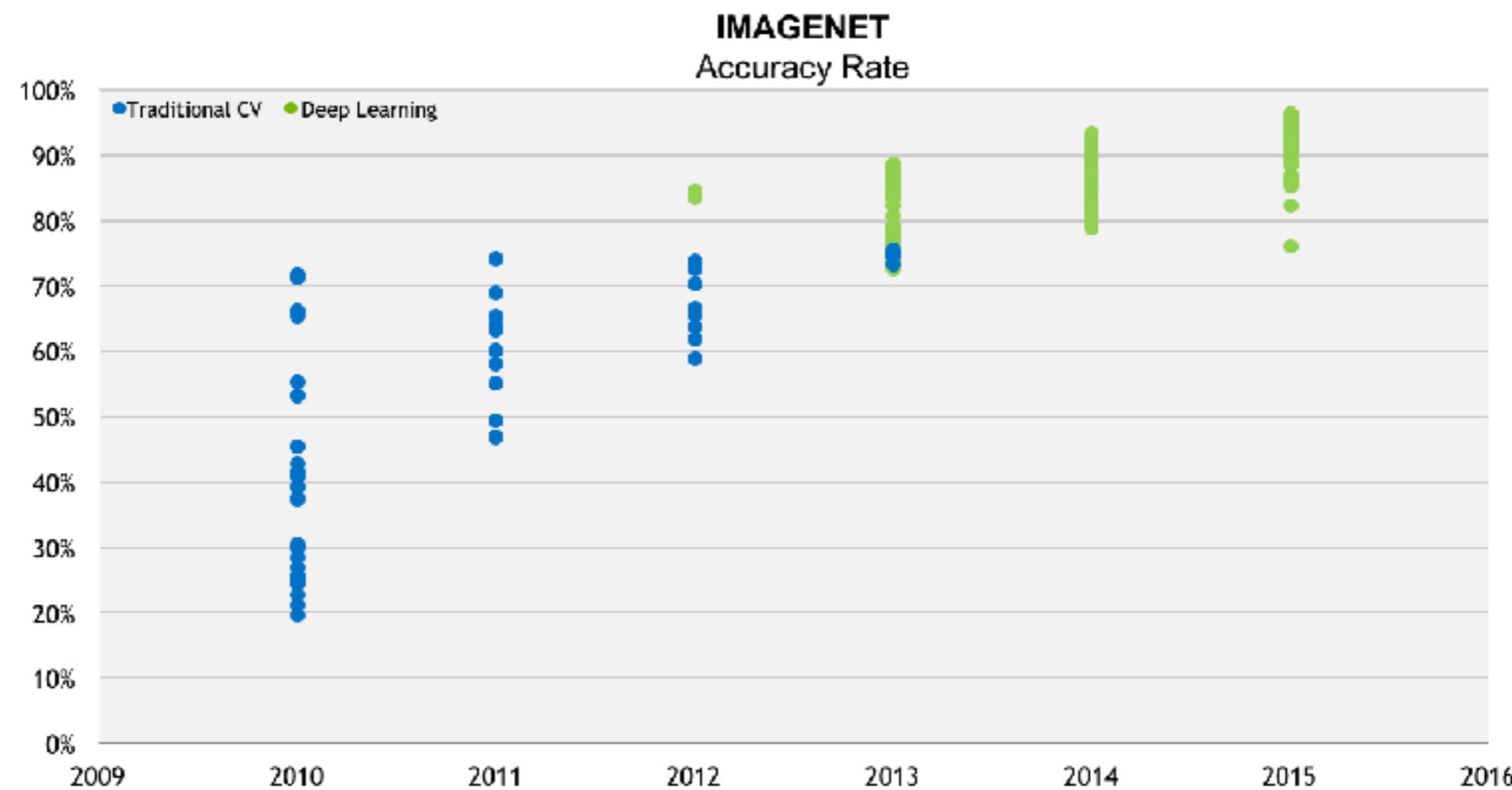
ImageNet-1000



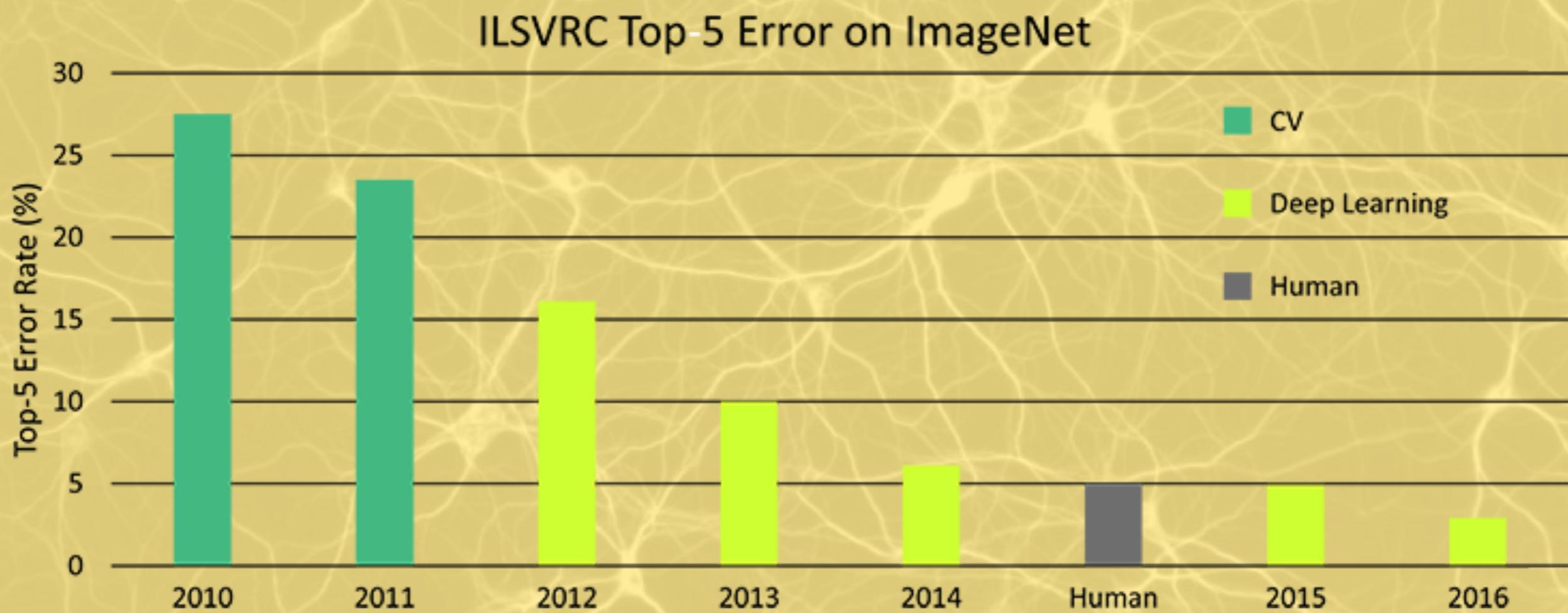
Deep Learning vs. Métodos Tradicionales

DEEP LEARNING FOR VISUAL PERCEPTION

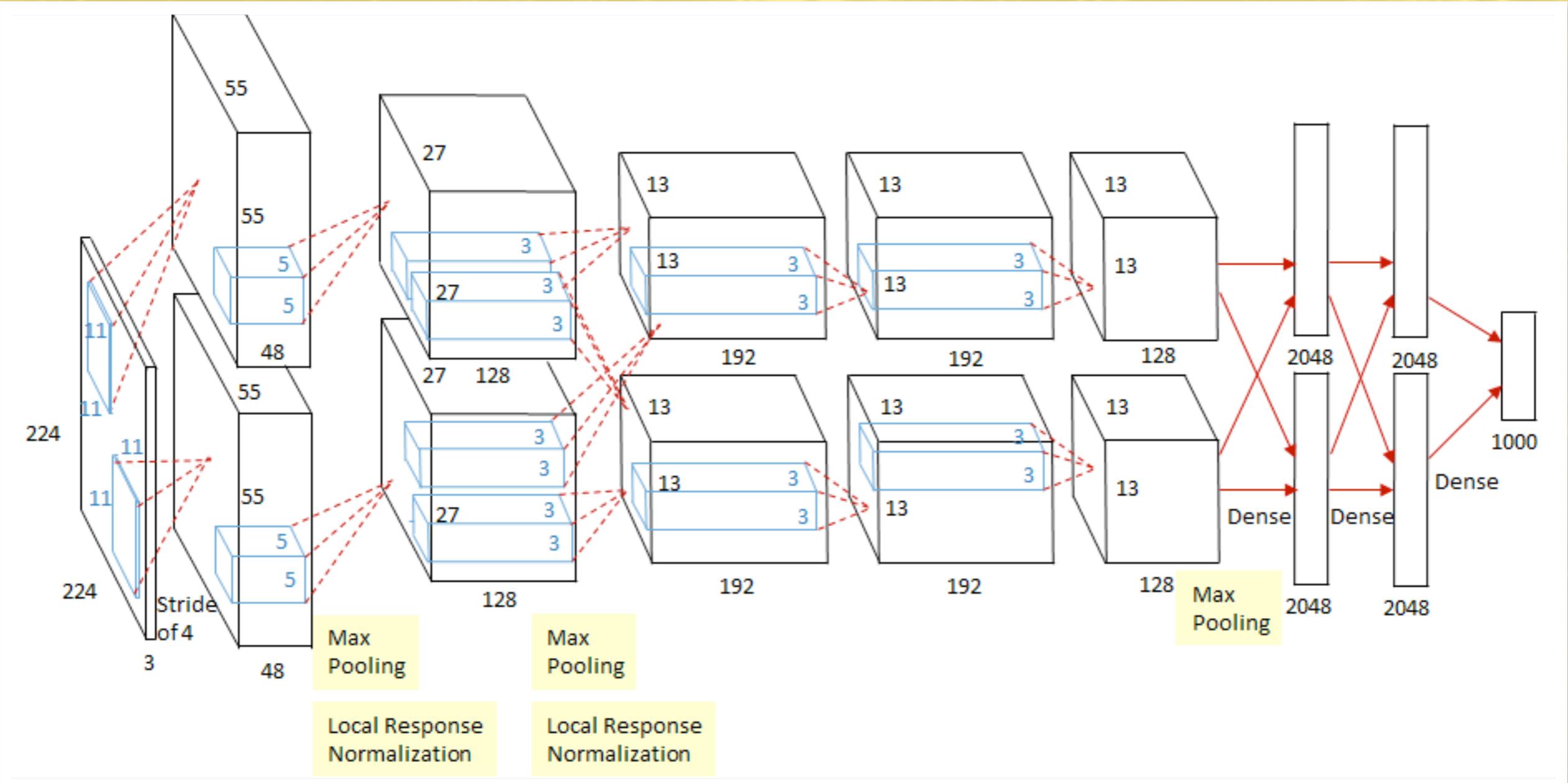
Going from strength to strength



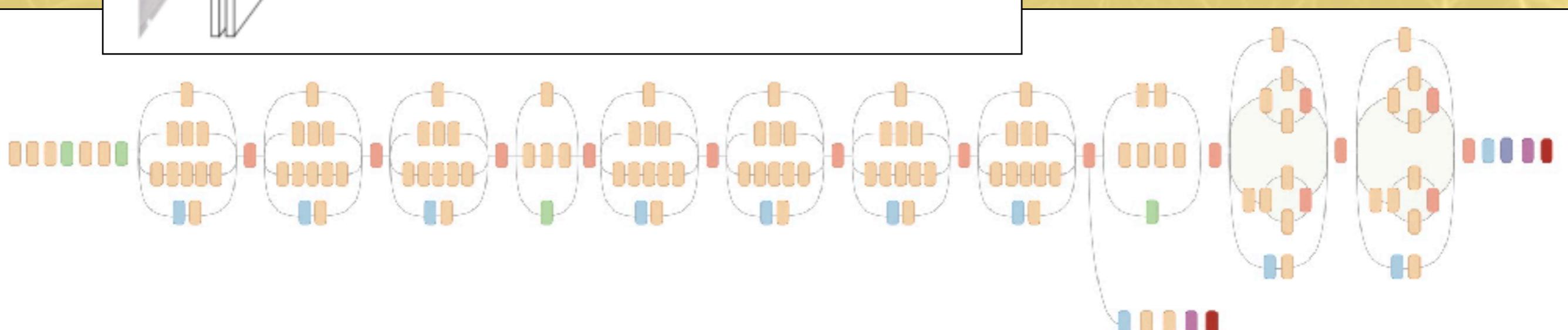
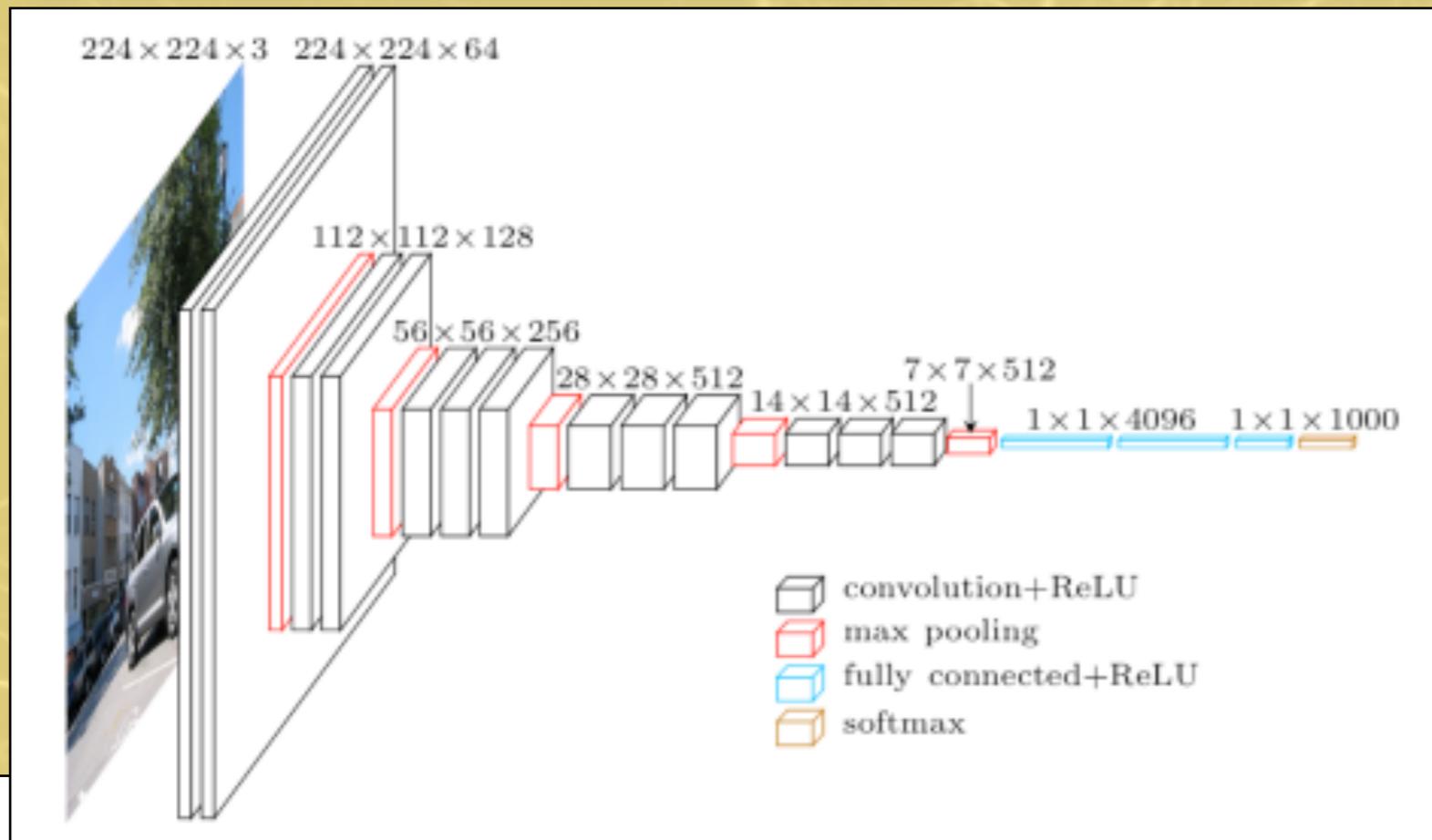
ImageNet Large Scale Visual Recognition Competition (ILSVRC)



Alexnet (2012)



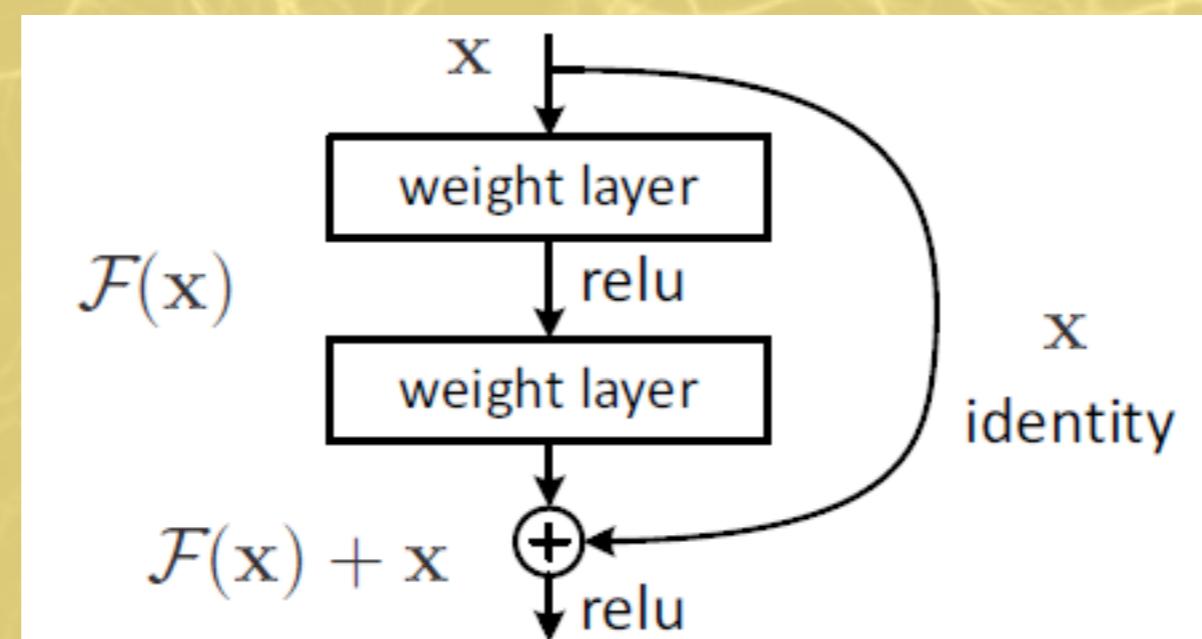
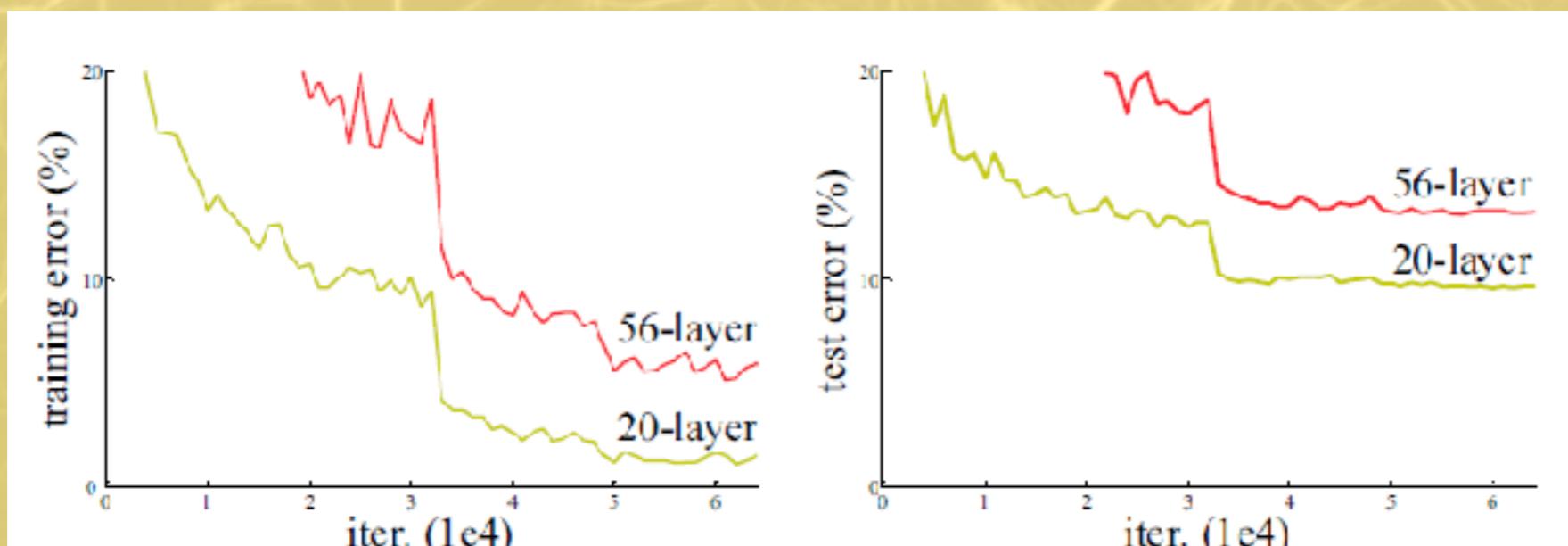
VGG16 (2014), Inception-v3 (2015)



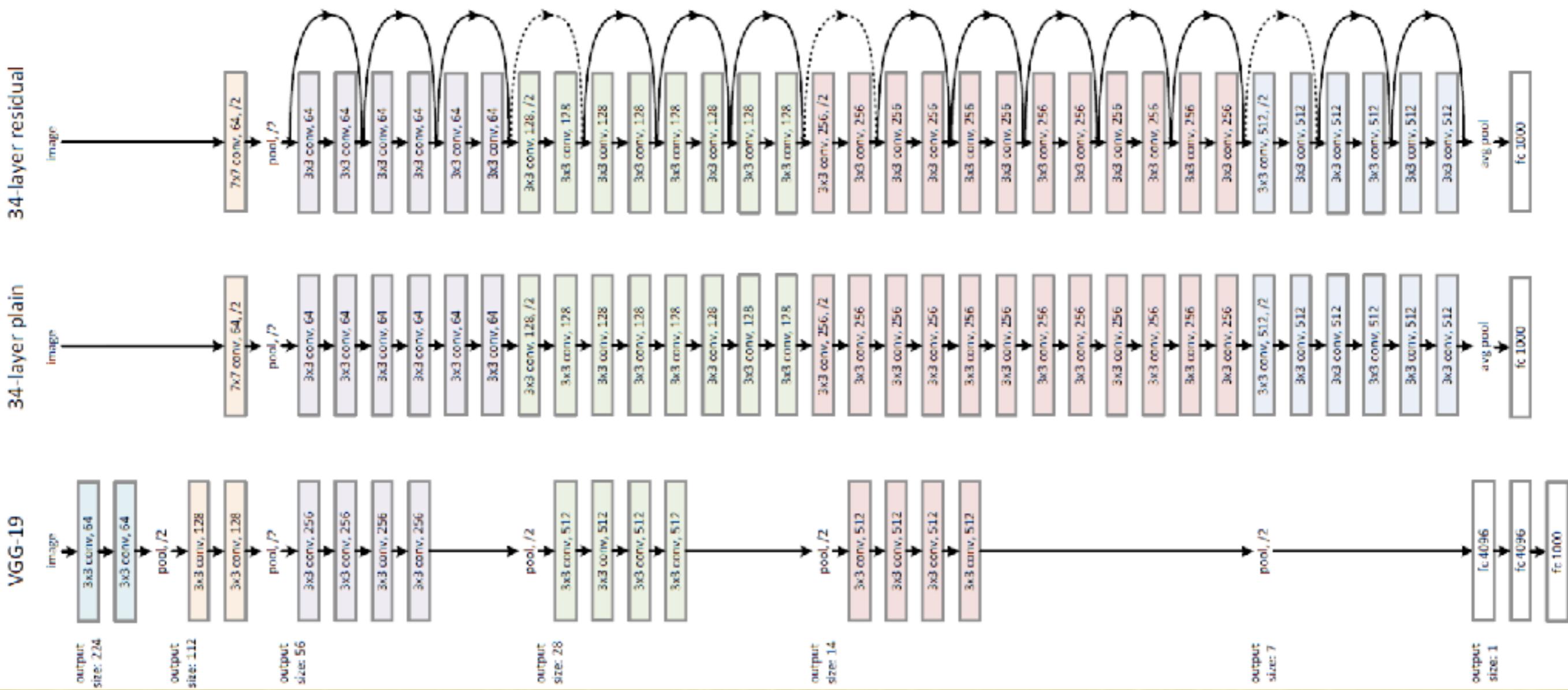
- Convolution
- AvgPool
- MaxPool
- Concat
- Dropout
- Fully connected
- Softmax

Resnet (2015)

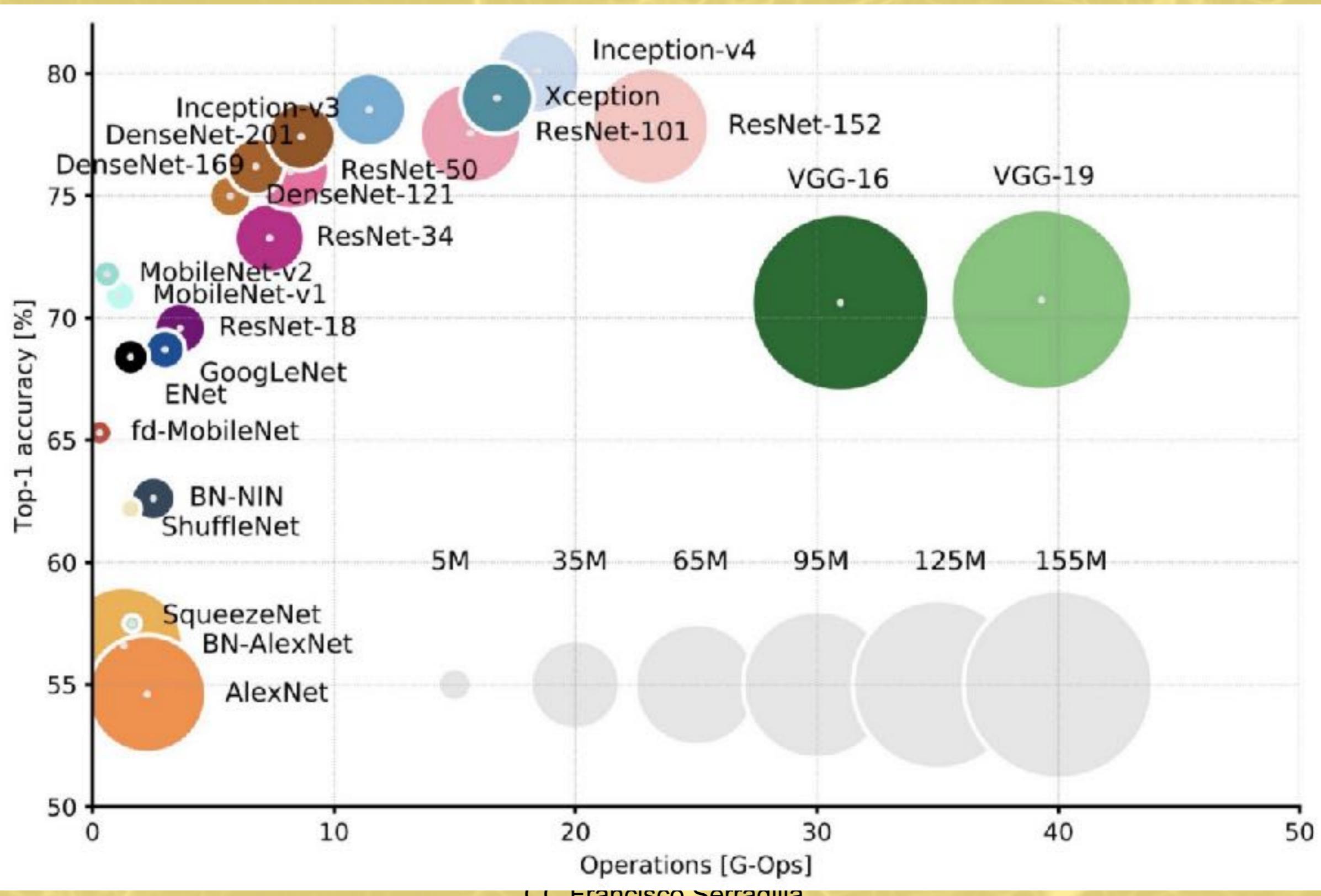
- * Las entradas se un grupo de conexiones se suman a las salidas; esto permite que la retropropagación del gradiente funcione mejor. Si las dimensiones no coinciden se usa una matriz identidad para proyectar las entradas



Resnet (2015)



Evolución de arquitecturas



A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

○ Backfed Input Cell

○ Input Cell

△ Noisy Input Cell

● Hidden Cell

○ Probabilistic Hidden Cell

△ Spiking Hidden Cell

● Output Cell

○ Match Input Output Cell

● Recurrent Cell

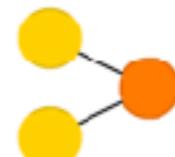
○ Memory Cell

△ Different Memory Cell

● Kernel

○ Convolution or Pool

Perceptron (P)



Feed Forward (FF)



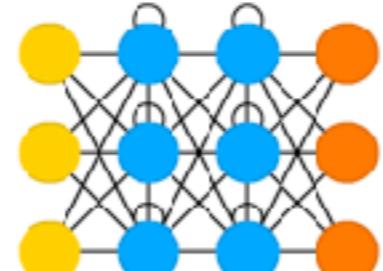
Radial Basis Network (RBF)



Deep Feed Forward (DFF)



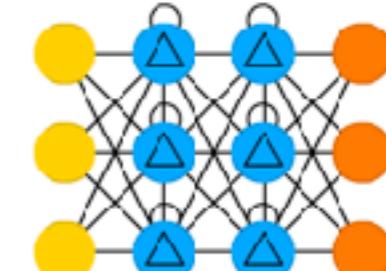
Recurrent Neural Network (RNN)



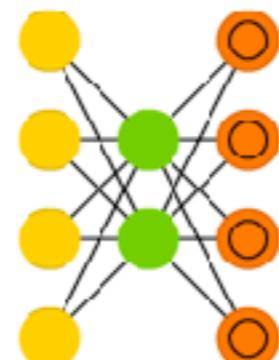
Long / Short Term Memory (LSTM)



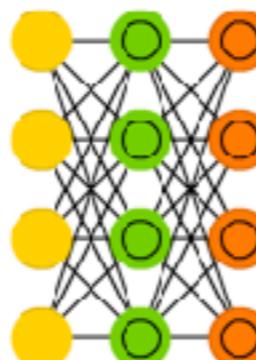
Gated Recurrent Unit (GRU)



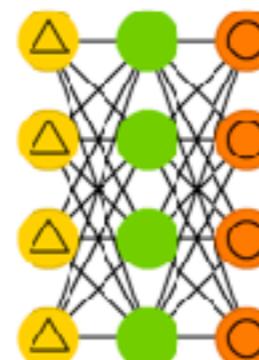
Auto Encoder (AE)



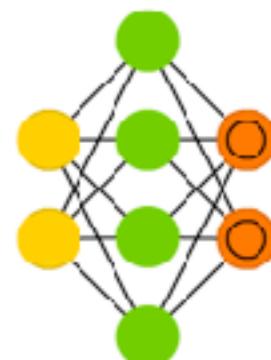
Variational AE (VAE)



Denoising AE (DAE)



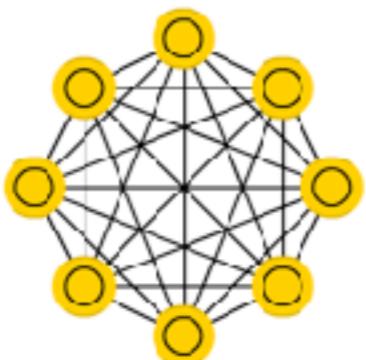
Sparse AE (SAE)



Markov Chain (MC)



Hopfield Network (HN)



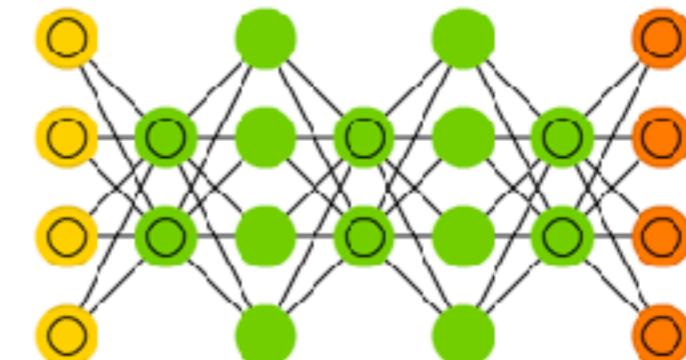
Boltzmann Machine (BM)



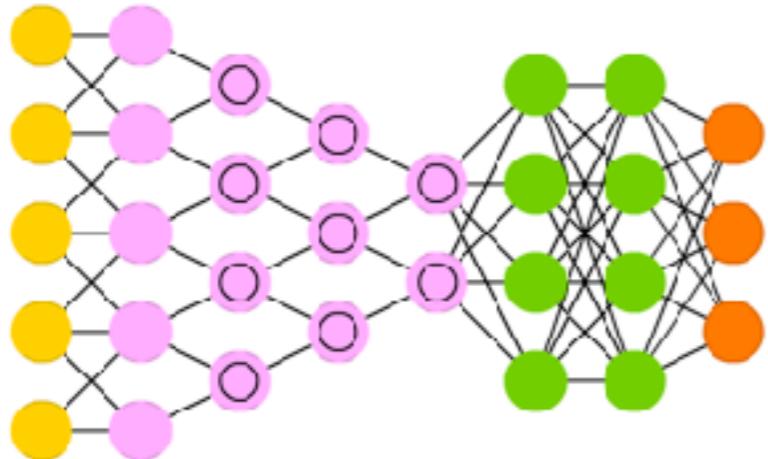
Restricted BM (RBM)



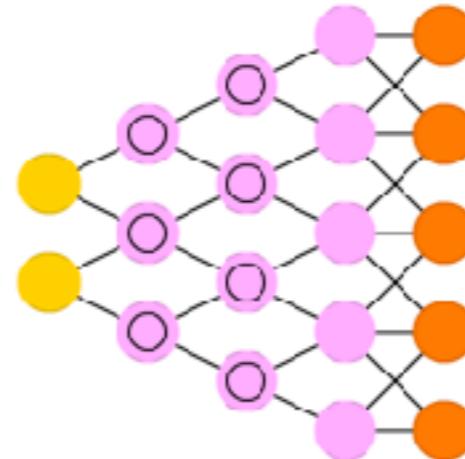
Deep Belief Network (DBN)



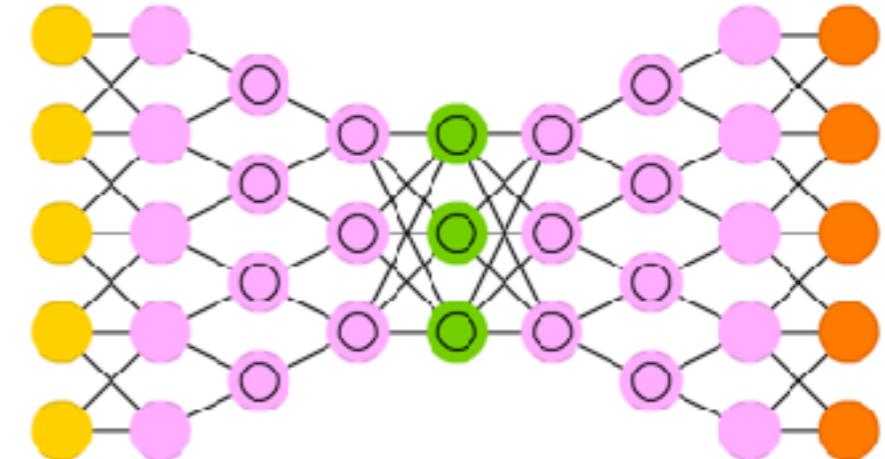
Deep Convolutional Network (DCN)



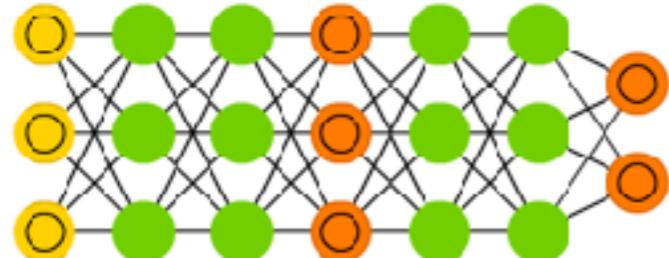
Deconvolutional Network (DN)



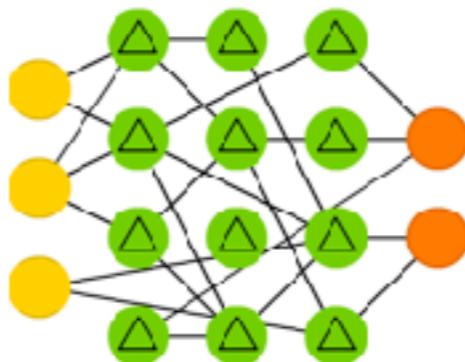
Deep Convolutional Inverse Graphics Network (DCIGN)



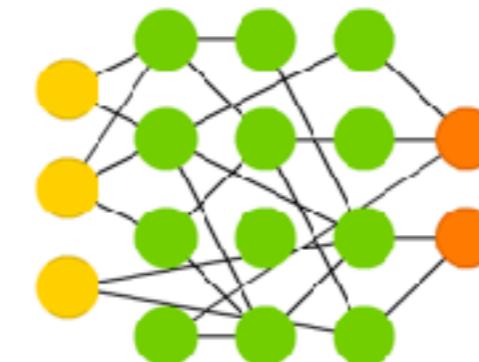
Generative Adversarial Network (GAN)



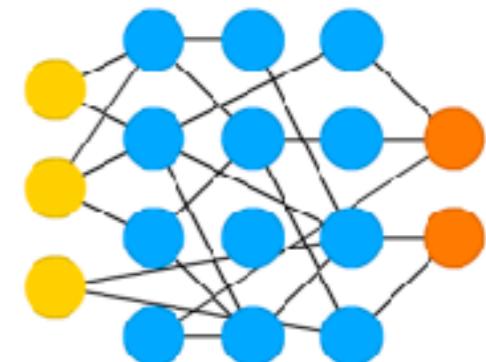
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



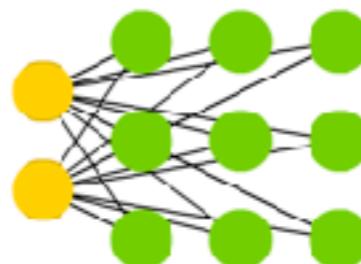
Echo State Network (ESN)



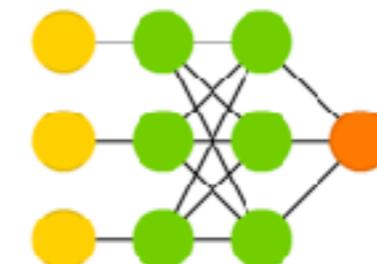
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)

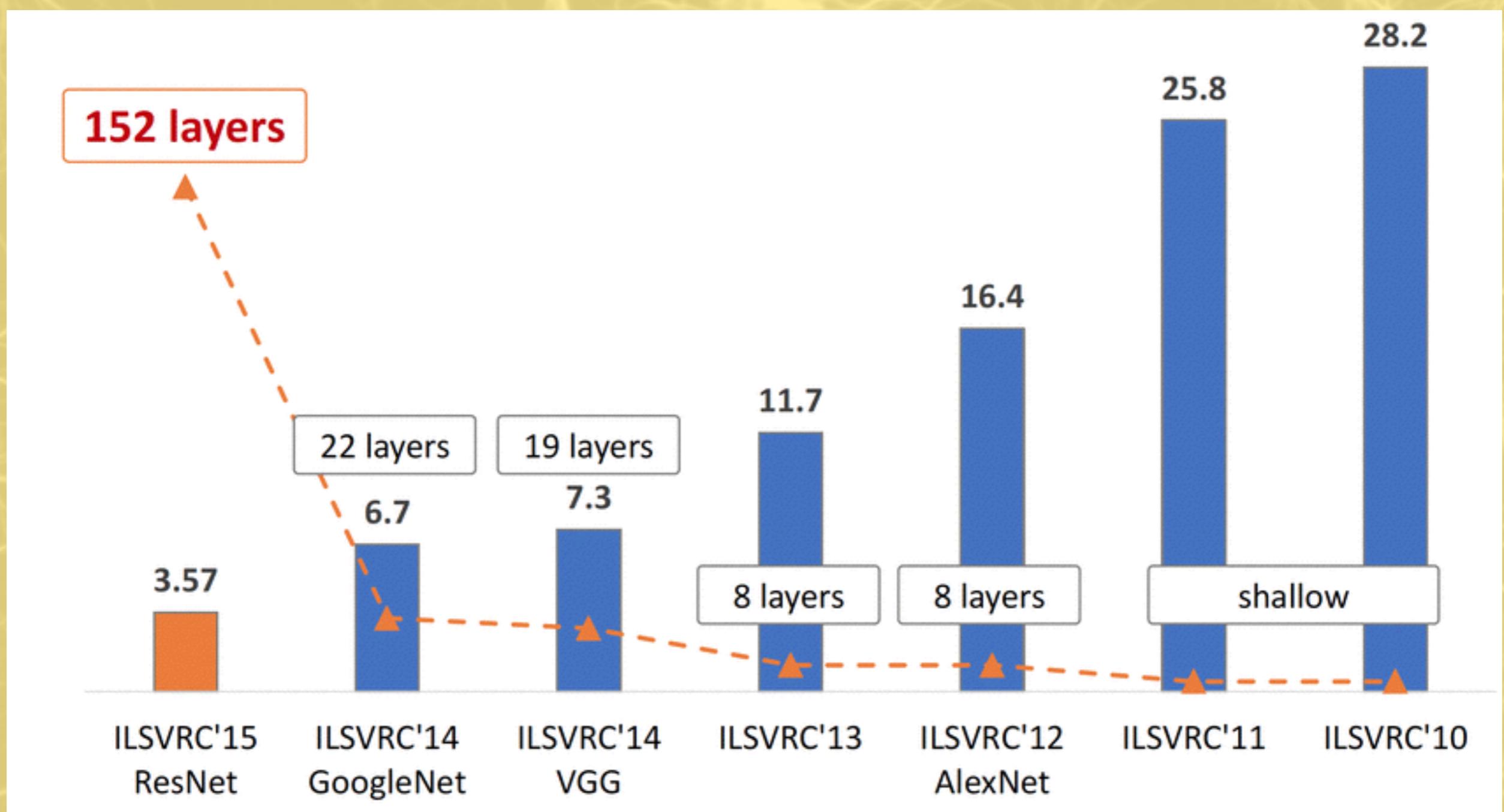


Número de capas

* ¿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)

Evolución del número de capas



Aplicaciones Y ejemplos

Neural-style

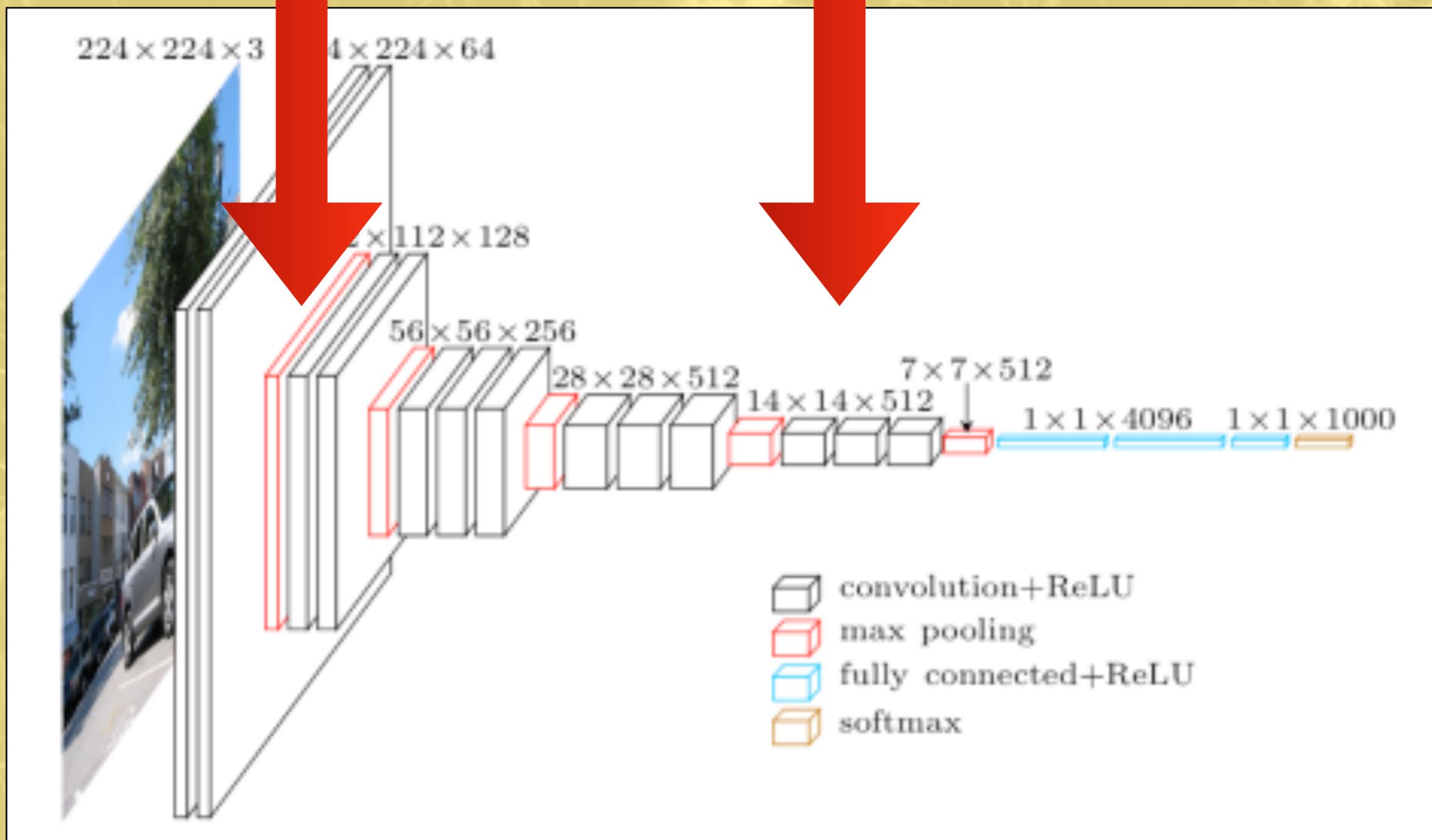
- * Aprendizaje del estilo de un artista usando redes de convolución para poder aplicarlo a cualquier imagen
- * Paper: Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). *A neural algorithm of artistic style*. arXiv preprint arXiv:1508.06576." <https://arxiv.org/pdf/1508.06576v2.pdf>
- * Código para tensorflow
 - <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

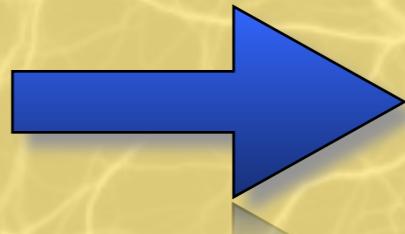
* Pasos para recrear una imagen

- 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

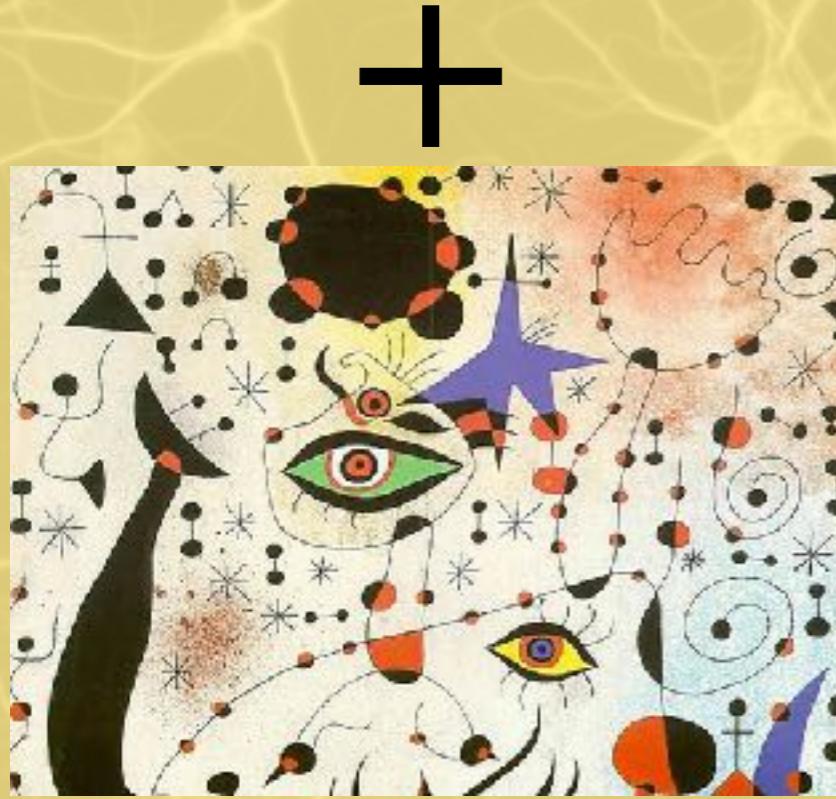
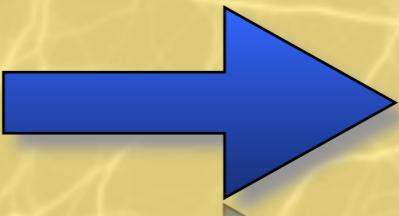
Style Transfer



Resultados

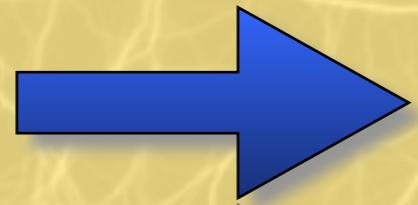
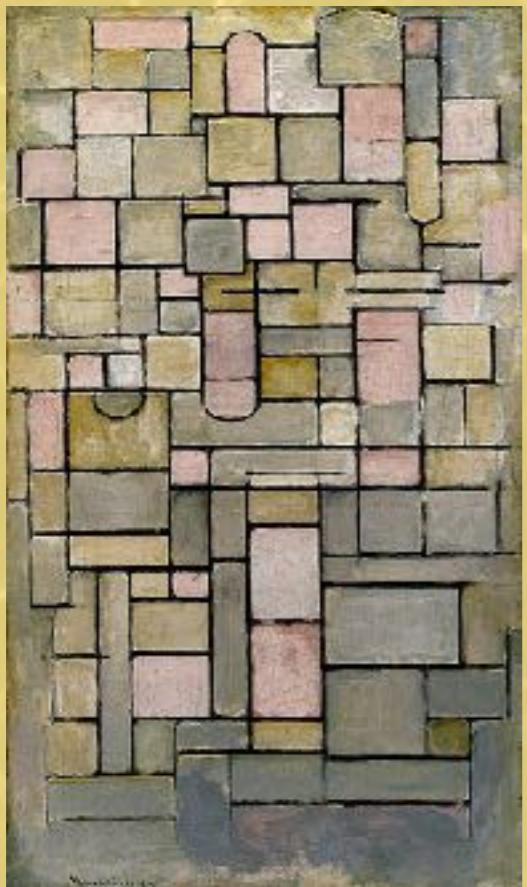


Resultados



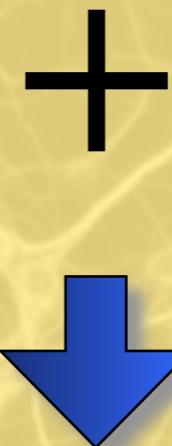
CC Francisco Serradilla

Resultados



CC Francisco Serradilla

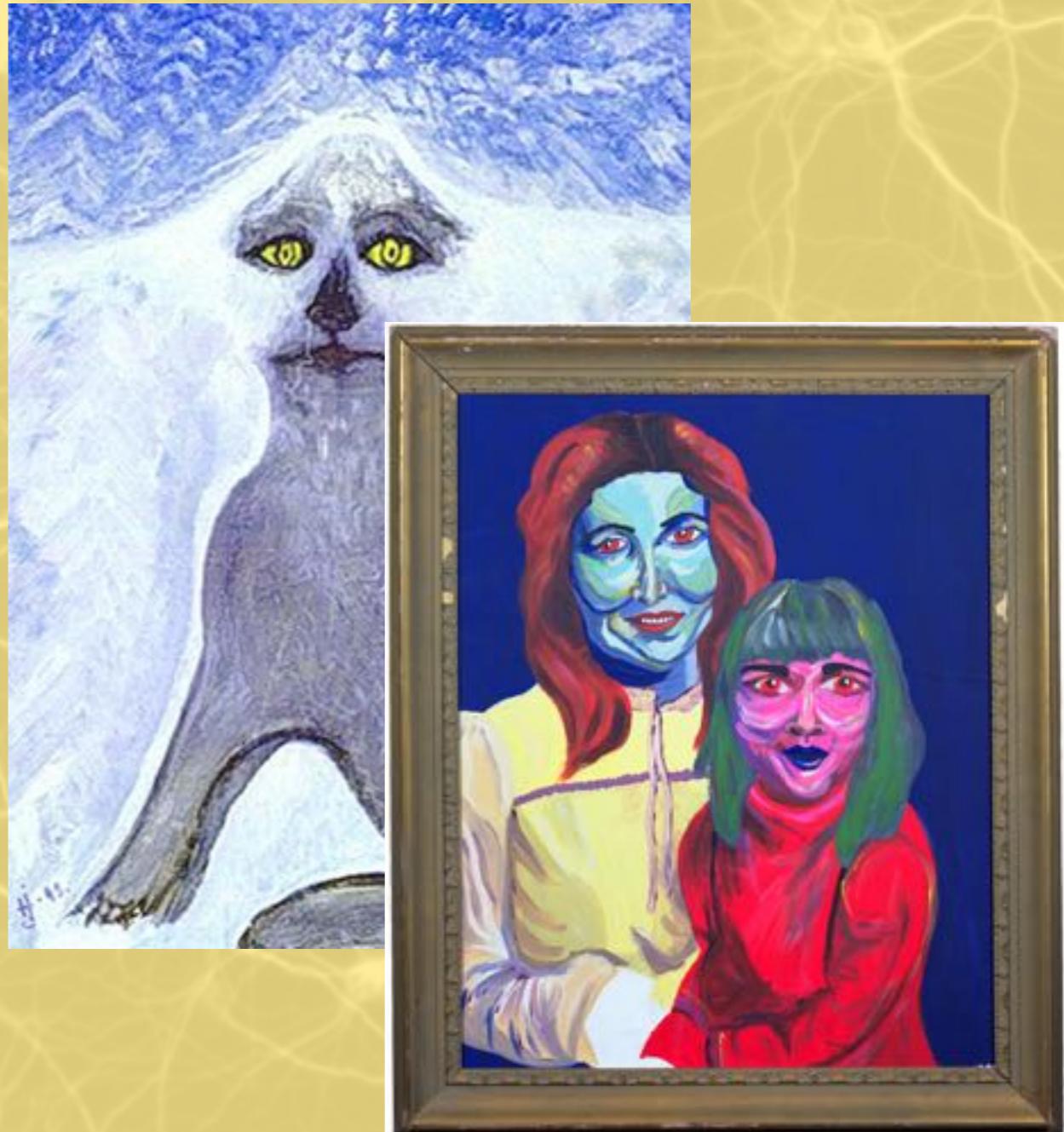
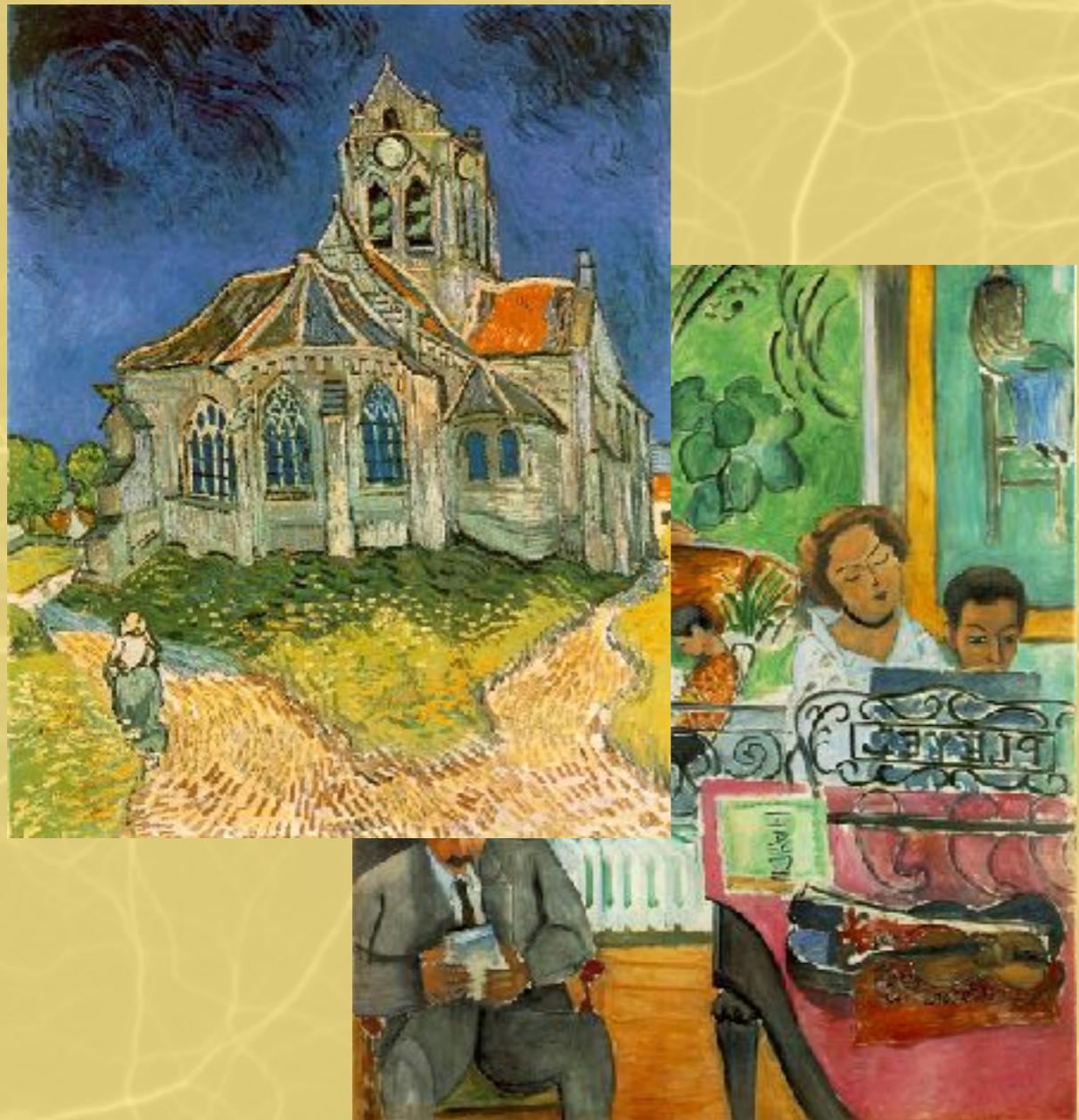
Resultados



Video



Detección de obras de arte

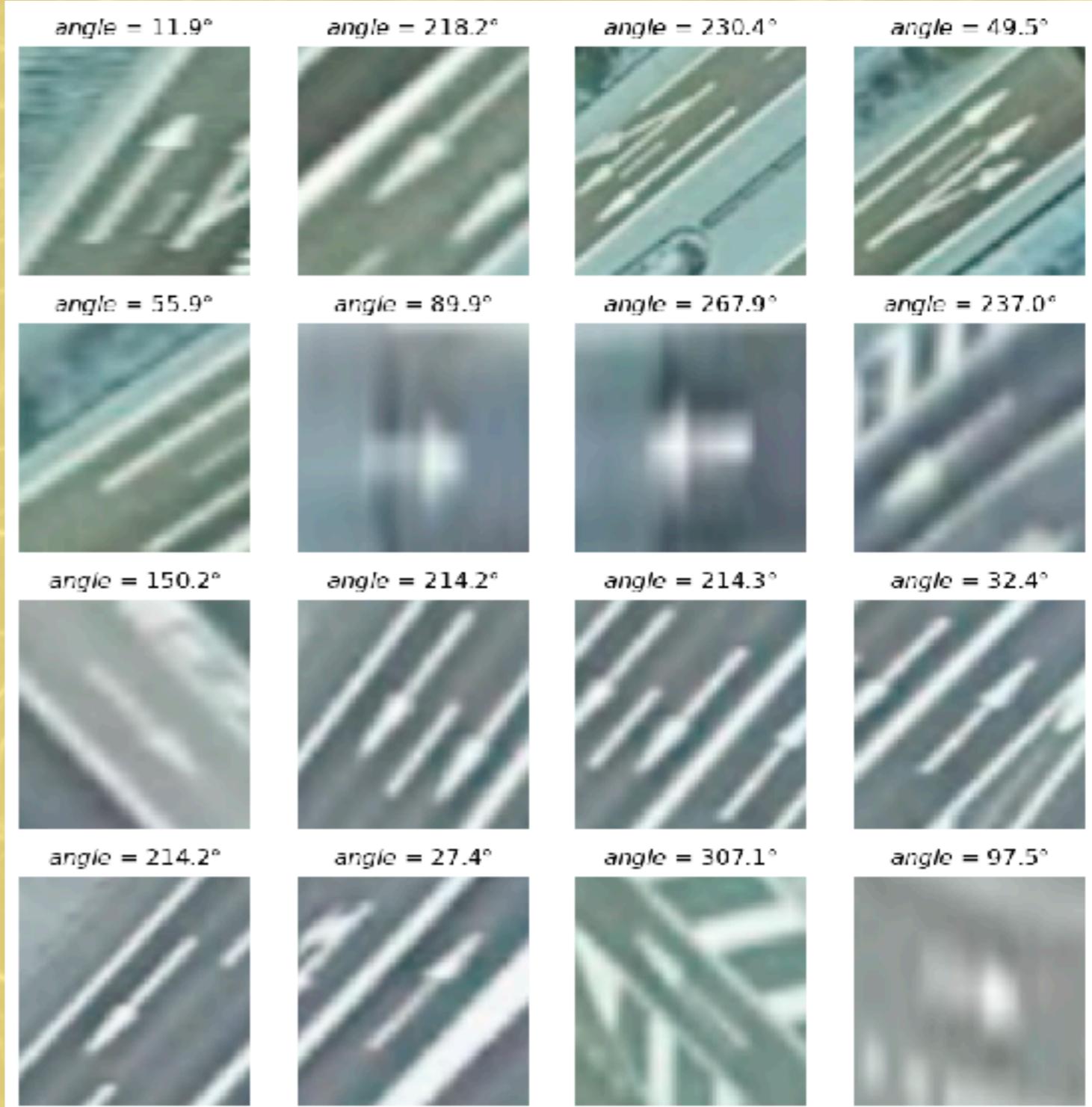


Detección de obras de arte

- * 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

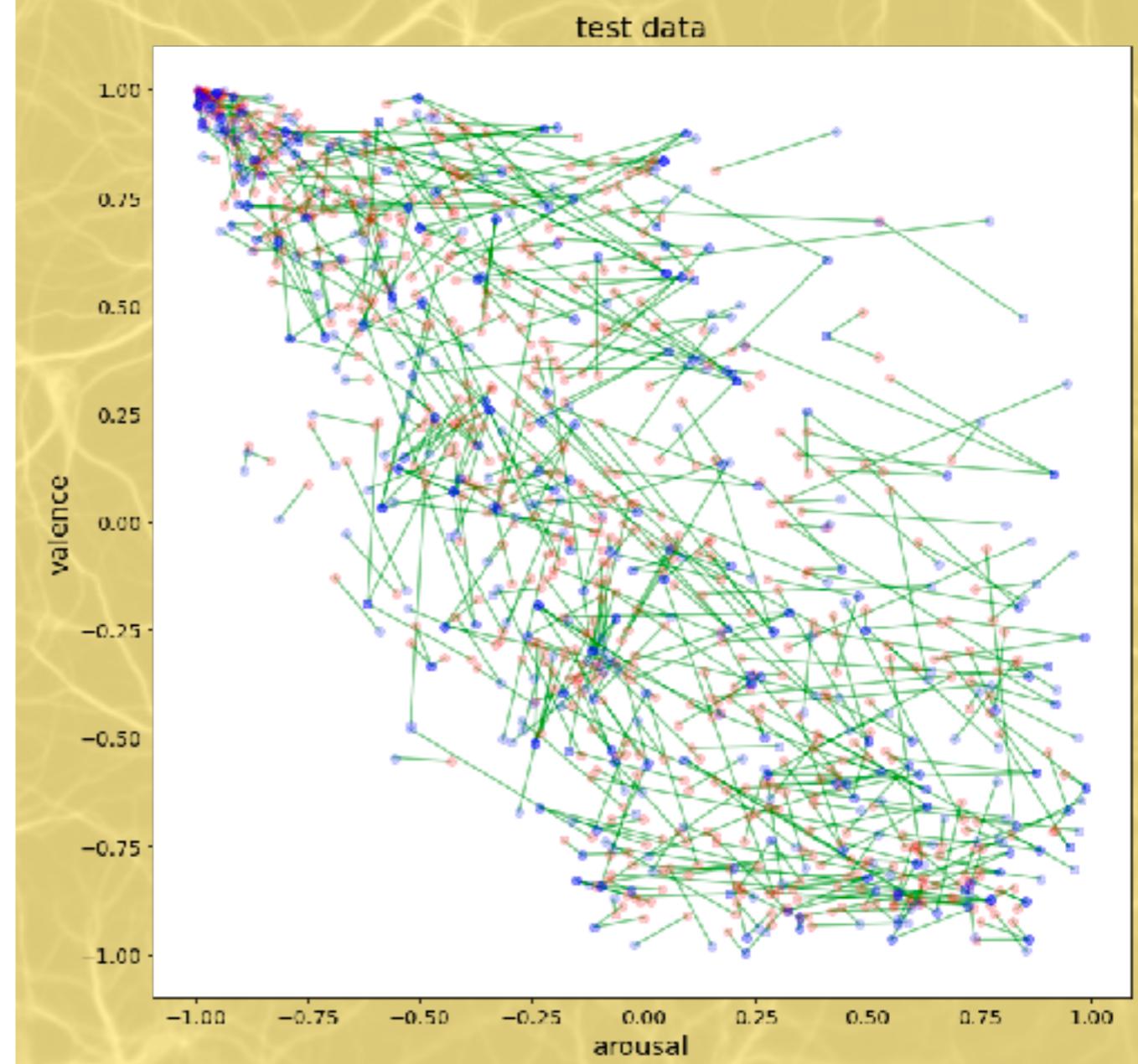
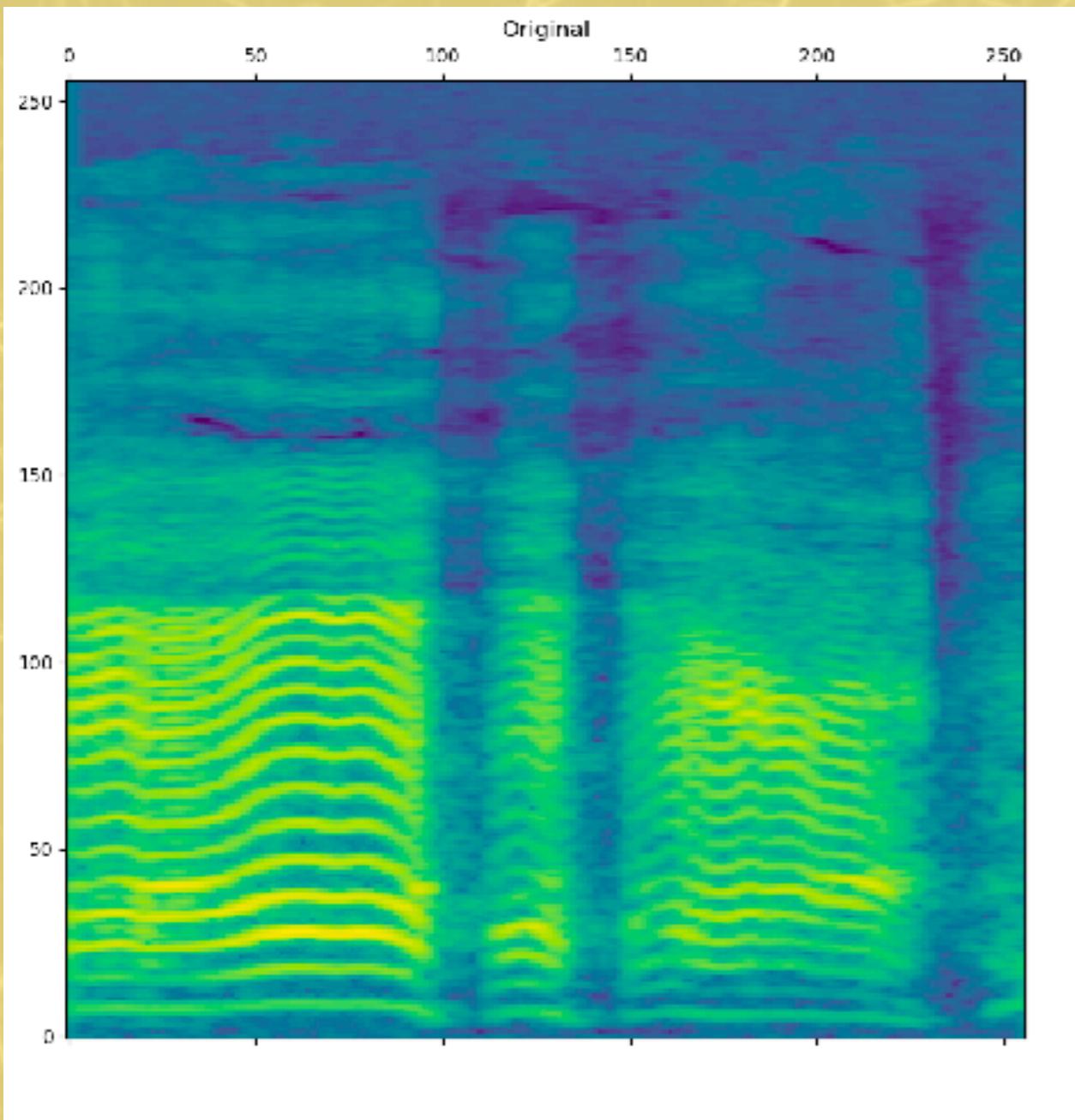
Ejemplos

- * A partir de imágenes de de satélite de flechas de la calzada y del ángulo que forma cada flecha con el norte
 - Construir un modelo que prediga, a partir del histórico de datos
 - ✗ Ángulo de rotación de la flecha



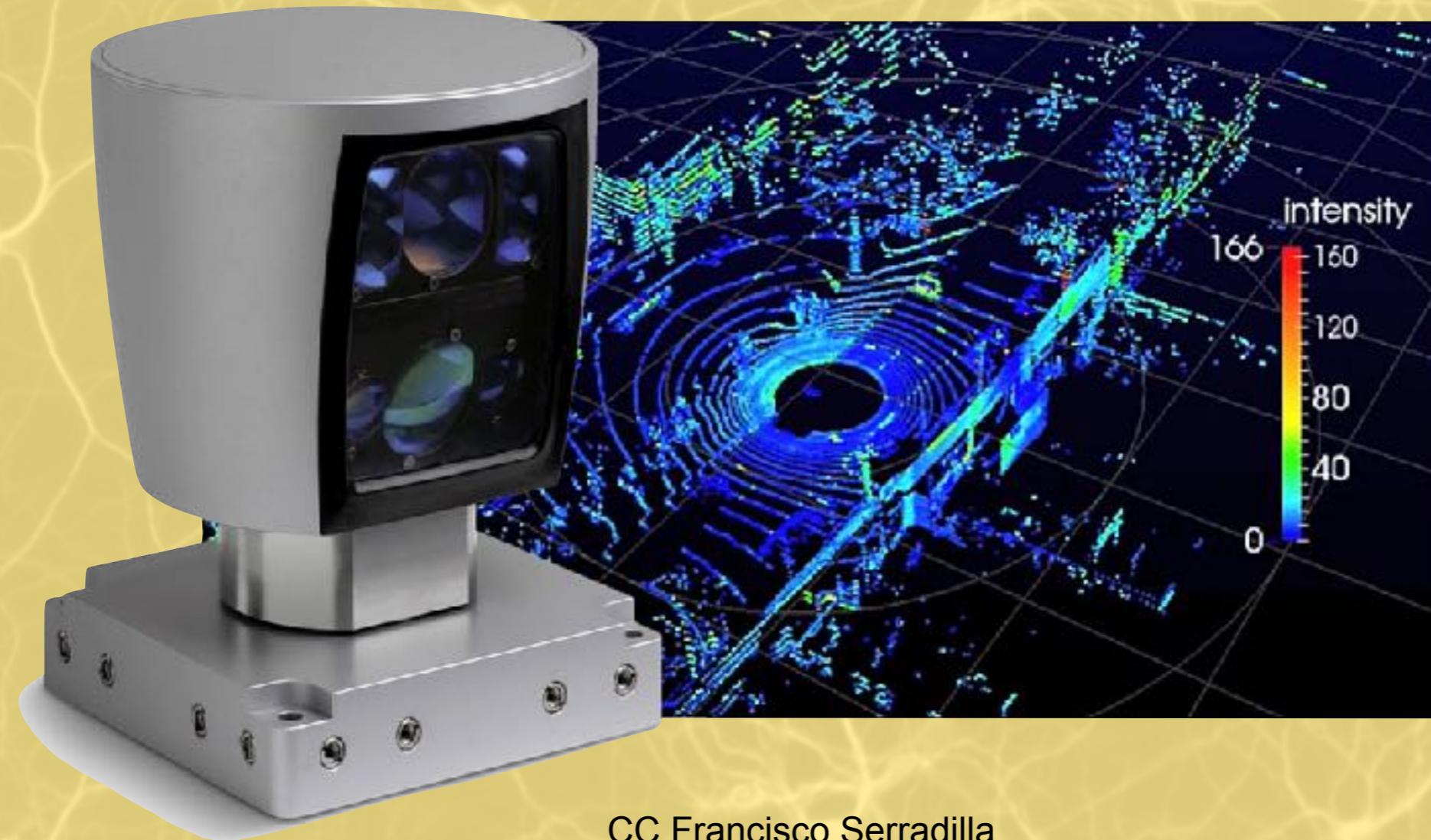
Ejemplos

- * A partir de sonidos, predecir la emoción que produce ese sonido en humanos



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



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

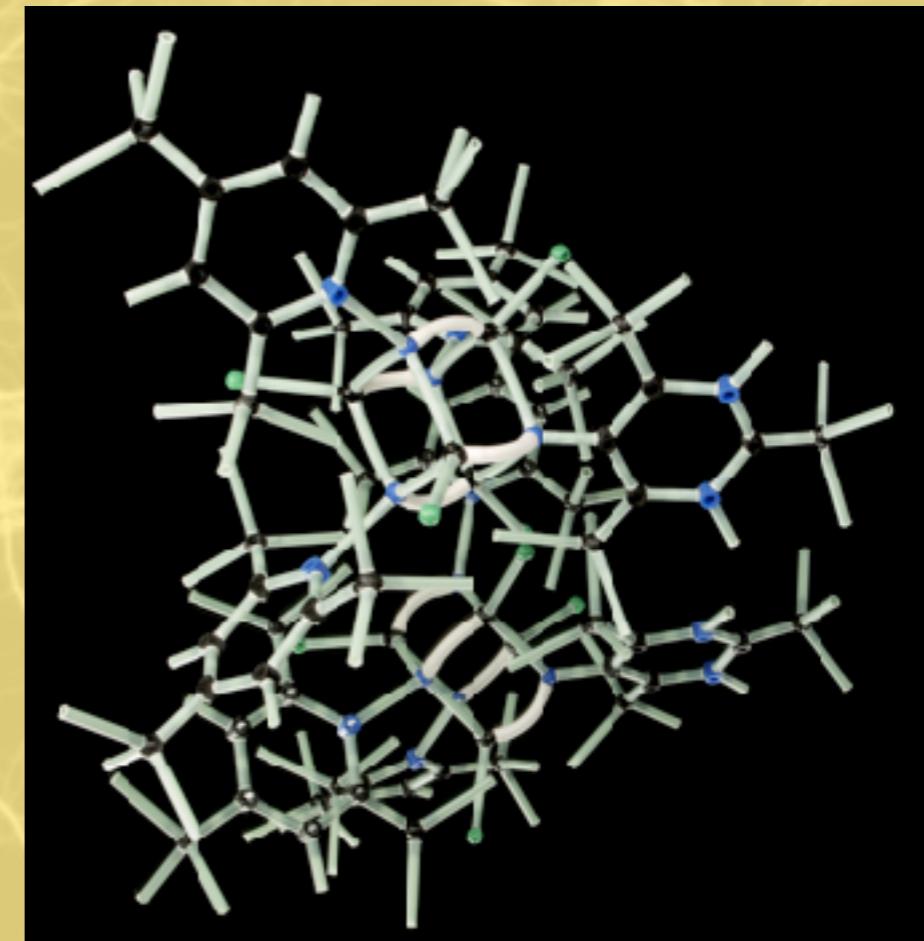
Juegos de Atari



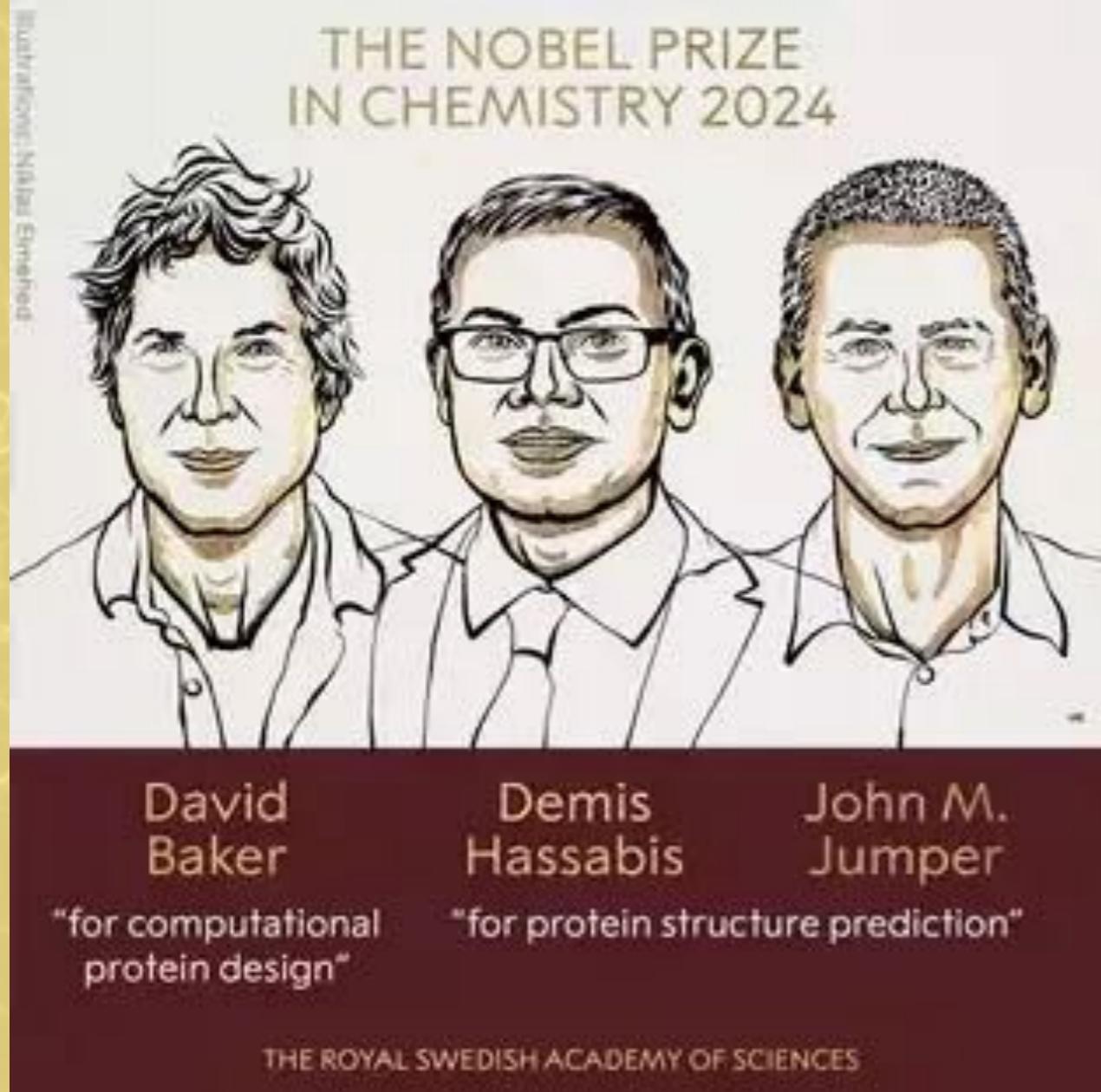
Predictión del plegamiento de proteínas

* “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 espectroscopía de RMN) en 2/3 de las



Predictión del plegamiento de proteínas



Proteínas diseñadas con IA neutralizan las toxinas letales del veneno de serpiente

15 enero 2025 - 18:45

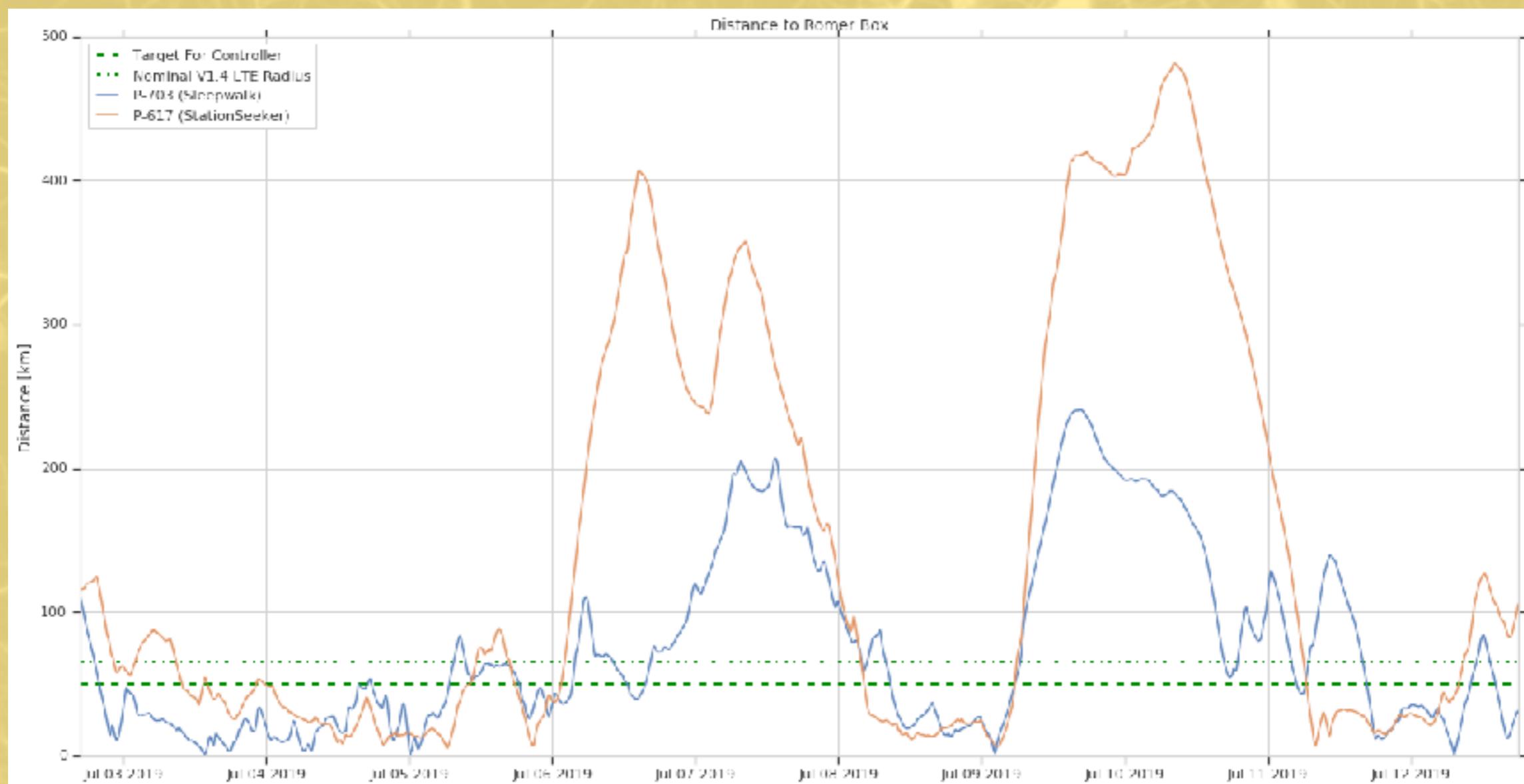
4 minutos

Redacción Ciencia, 15 ene (EFE).- Un equipo científico ha diseñado gracias a la inteligencia artificial (IA) nuevas proteínas que neutralizan las toxinas letales del veneno de la cobra, lo que podría ofrecer una alternativa más segura y eficaz a los antídotos tradicionales.

Los detalles de la investigación, probada en experimentos en ratones, se publican en la revista Nature en un artículo que lidera el último premio Nobel de Química, David Baker.

Los globos Loon de Google ya no escuchan a humanos

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



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

Un enfoque basado en capas

- * Actualmente hay que pensar más en qué habilidades 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 (paquete nn)
 - Linear (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)
 - ConvTranspose2D: 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)
 - Embedding: para convertir tokens en embeddings
 - MultiheadAttention y Transformer: utilizan (aprenden) key, value y query para obtener el embedding de una secuencia de observaciones
- * En keras se puede poner la función de activación como parámetro, en pytorch hay que añadir otra línea de código: linear (por defecto), sigmoid, tanh, softmax, ReLU, leakyReLU...
- * Las capas pueden conectarse en secuencia o de modo funcional (por ejemplo para tener capas residuales)

Un enfoque basado en capas

* Tipos de capas auxiliares

- MaxPool2D: realizan operación de maxpool
- Dropout: regulariza usando dropout
- flatten, reshape: cambian las dimensiones del tensor, manteniendo los valores
- BatchNorm2d: normalizan los valores entre dos capas
- cat: concatena las salidas de varias capas

* Lista completa en

- <https://pytorch.org/docs/stable/index.html>

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

* 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

Pytorch

- * La comunidad ha ido abandonando Keras / Tensorflow en favor de pytorch
 - Mejor mantenimiento y actualizaciones
 - Gestión explícita de CPU / GPU
 - Incorporación más rápida de novedades en el mundo de la IA
 - Compatibilidad más rápida con las últimas versiones de python
 - Compatibilidad con GPUs alternativas (AMD, Apple silicon)
 - Usada por la mayoría de herramientas específicas, en particular por ComfyUI
 - La mayoría de los grupos de investigación hacen y publican su código en pytorch
 - Lo veremos en un notebook

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/>

Recursos

- * 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/>

Recursos

- * 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/>

- * 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)