

Prácticas de Aprendizaje Automático

Presentación de la Práctica 3 e Introducción a Keras

Pablo Mesejo y Salvador García

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Índice

- Normas de entrega
- Repaso de aprendizaje profundo
- Introducción a Keras
- Presentación de la práctica

Índice

- **Normas de entrega**
- Repaso de aprendizaje profundo
- Introducción a Keras
- Presentación de la práctica

Normas de la Entrega de Prácticas

- El **código** debe estar **bien comentado** y todas las **decisiones tomadas y el análisis de resultados deben estar documentados ampliamente** en celdas de texto (análisis, descripción del trabajo realizado, discusión de resultados).
- Se entrega un Notebook con todo integrado (**memoria/código/resultados**).

Normas de la Entrega de Prácticas

- Solo se entrega el Notebook! → **no imágenes u otros datos!**
- **No escribir nada en el disco!**
- La práctica deberá poder ser ejecutada de principio a fin **sin errores** y sin necesidad de **ninguna selección de opciones**.
 - Hay que fijar de inicio los parámetros que se consideren óptimos.

Entrega

- Fecha límite: 09 de Junio
- Valoración: 10 puntos (1 punto de los 8 que valen las prácticas)
- Lugar de entrega: PRADO
- **Se valorará mucho el informe:** descripción de qué se ha hecho y cómo, justificación de las decisiones tomadas, discusión de los resultados obtenidos

Dudas

pmesejo@go.ugr.es

salvagl@decsai.ugr.es

Índice

- Normas de entrega
- **Repaso de aprendizaje profundo**
- Introducción a Keras
- Presentación de la práctica

Nota previa

- Esta práctica se ocupa de introducir algunos fundamentos de **aprendizaje profundo** y, como aplicación, se emplean dos tareas/**problemas de análisis de imagen**.
- En la asignatura de **Visión por Computador** (4º) se profundizará en muchas de estas intuiciones e ideas, y se presentarán modelos más avanzados.

Lectura recomendada

(buena parte de las siguientes *slides* están tomadas de estas fuentes)

- CS231n: Convolutional Neural Networks for Visual Recognition (ahora llamado Deep Learning for Computer Vision):

<https://cs231n.stanford.edu/slides/2023/>

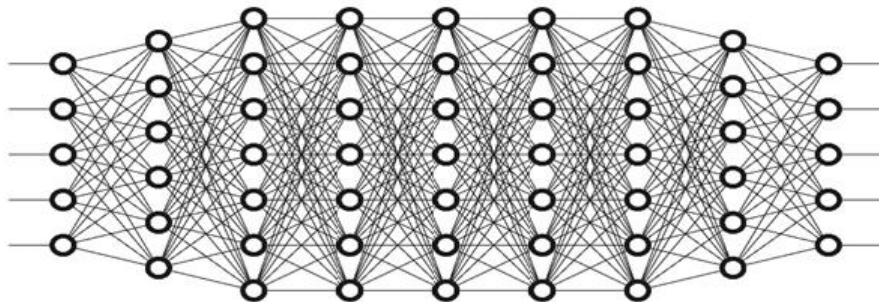
- En particular, la sección dedicada a Convolutional Neural Networks (ConvNets):

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

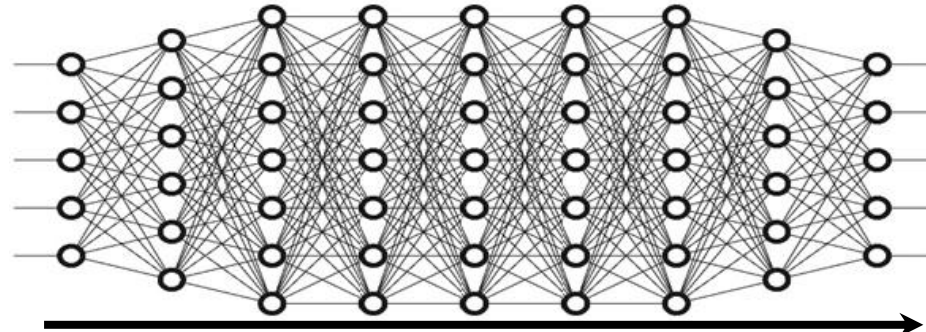
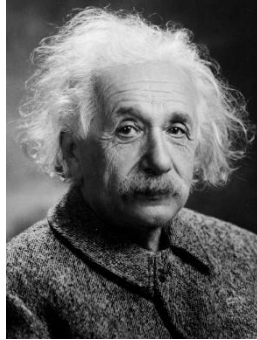
<https://cs231n.github.io/convolutional-networks/>

¿Qué son las redes neuronales profundas?

- Definición oficial
 - computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction (LeCun et al., 2015)
- A nivel práctico (y en general):
 - Redes neuronales con “muchas” capas ocultas
 - Y que, como consecuencia, pueden aproximar funciones más complejas (es decir, resolver problemas más complejos)

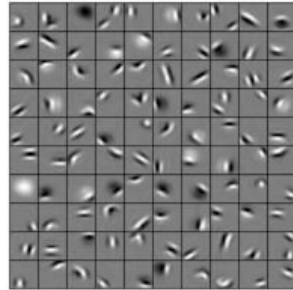


¿Qué son las redes neuronales profundas?

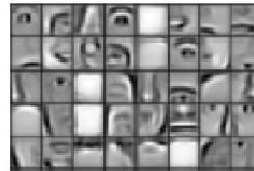


“Albert
Einstein”

Low-level
features



Mid-level
features



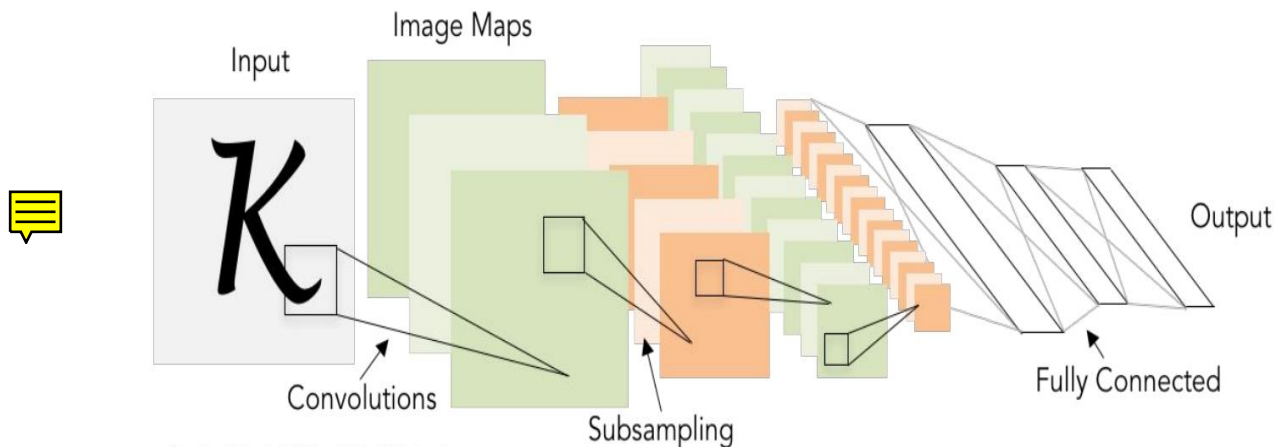
High-level
features



Aprenden representaciones jerárquicas de los datos de entrada
(múltiples niveles de abstracción)

Redes neuronales convolucionales

- ¿Qué es una red neuronal convolucional (ConvNet o CNN)?
 - Red neuronal con una convolución en, al menos, una de sus capas.
 - Se utiliza en problemas donde la información se presenta en formato de matriz/*grid* (por ejemplo, imágenes)
- Arquitectura típica de una ConvNet:



¿Qué es una convolución?

- **Operación local lineal** con una máscara (también llamado filtro o *kernel*).
 - Los coeficientes/valores de la máscara/filtro determinan la operación realizada.

- Filtrado Gaussiano (elimina altas frecuencias → suavizado de imágenes)

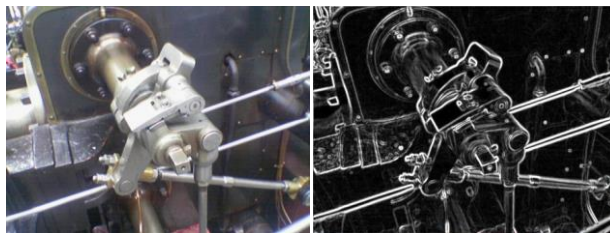
$$\frac{1}{273}$$

| | | | | |
|---|----|----|----|---|
| 1 | 4 | 7 | 4 | 1 |
| 4 | 16 | 26 | 16 | 4 |
| 7 | 26 | 41 | 26 | 7 |
| 4 | 16 | 26 | 16 | 4 |
| 1 | 4 | 7 | 4 | 1 |



- Sobel filter (elimina bajas frecuencias → realce de bordes)

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



¿Qué es una convolución?

- Operación local lineal con una máscara.

Los números **rojos** representan los valores/coeficientes del filtro/máscara.

El filtro se multiplica elemento por elemento con la imagen, se suman los productos y se sustituye la posición central del filtro en la imagen.



| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

| | | |
|---|--|--|
| 4 | | |
| | | |
| | | |

Convolved
Feature



| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

| | | |
|---|--|--|
| 4 | | |
| | | |
| | | |

Convolved
Feature

=

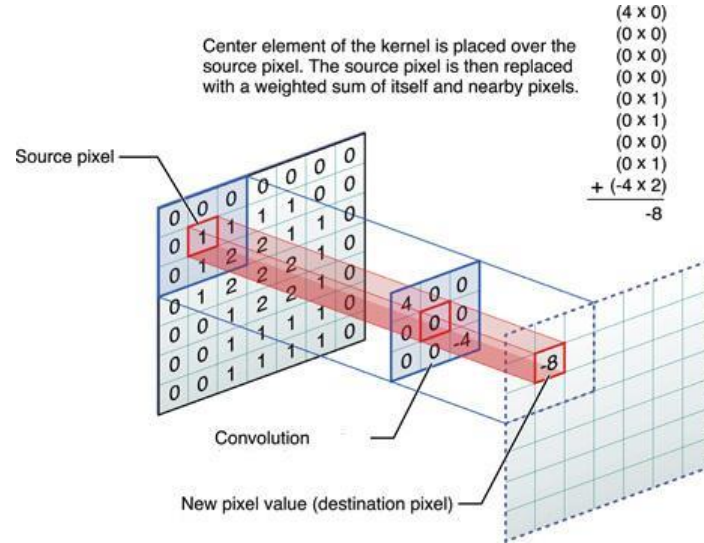
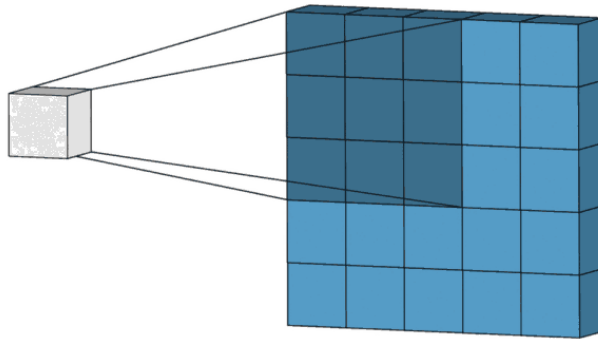
Activation Map

=

Feature Map

¿Qué es una convolución?

- Otra forma de verlo...



Idea clave

¡En ConvNets, estos valores/coeficientes se aprenden! No son seleccionados por un experto humano. Ahora son parámetros libres de la red y se entrenan como cualquier otro peso.



1989: LeCun et al. used **back-propagation to directly learn the coefficients of convolutional filters** from images of handwritten numbers.

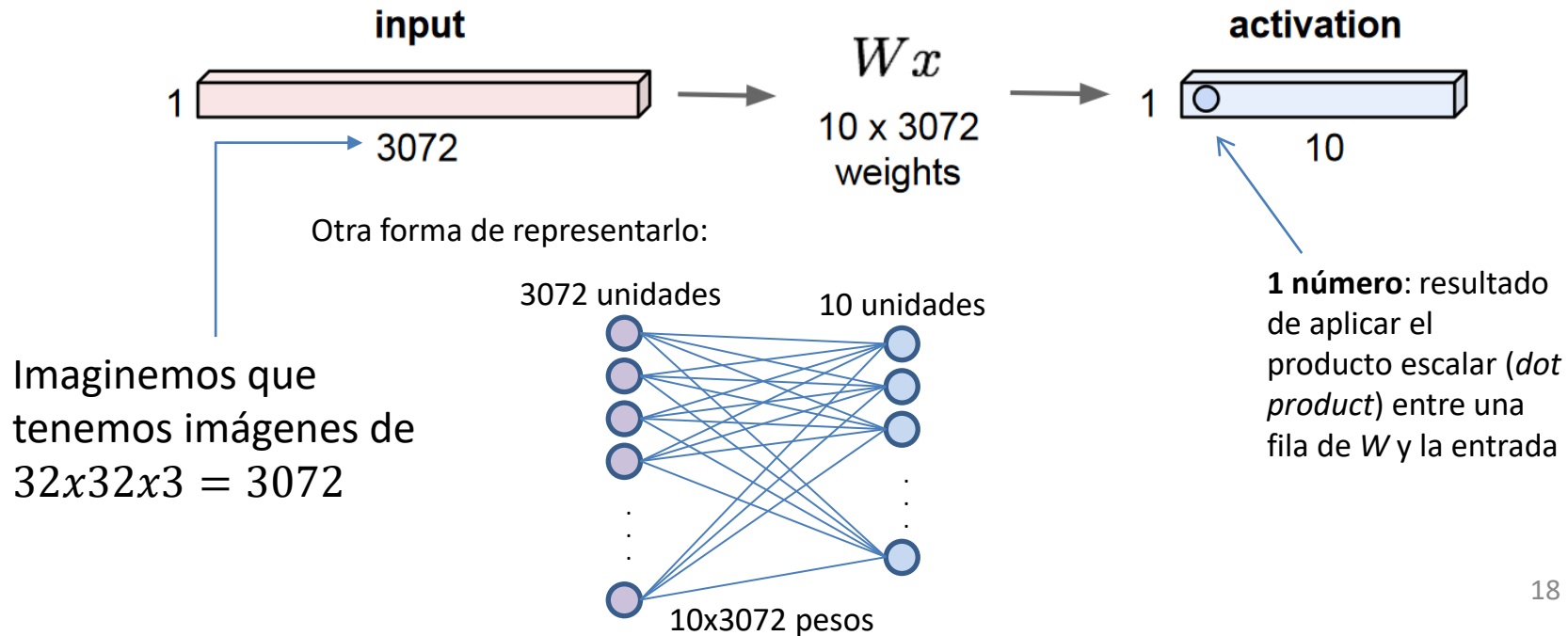
1998: LeCun et al. presented **LeNet-5**, ConvNet with 5 layers that could automatically recognize handwritten numbers, and showed that **ConvNets outperformed all other techniques in this task**.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 541-551.

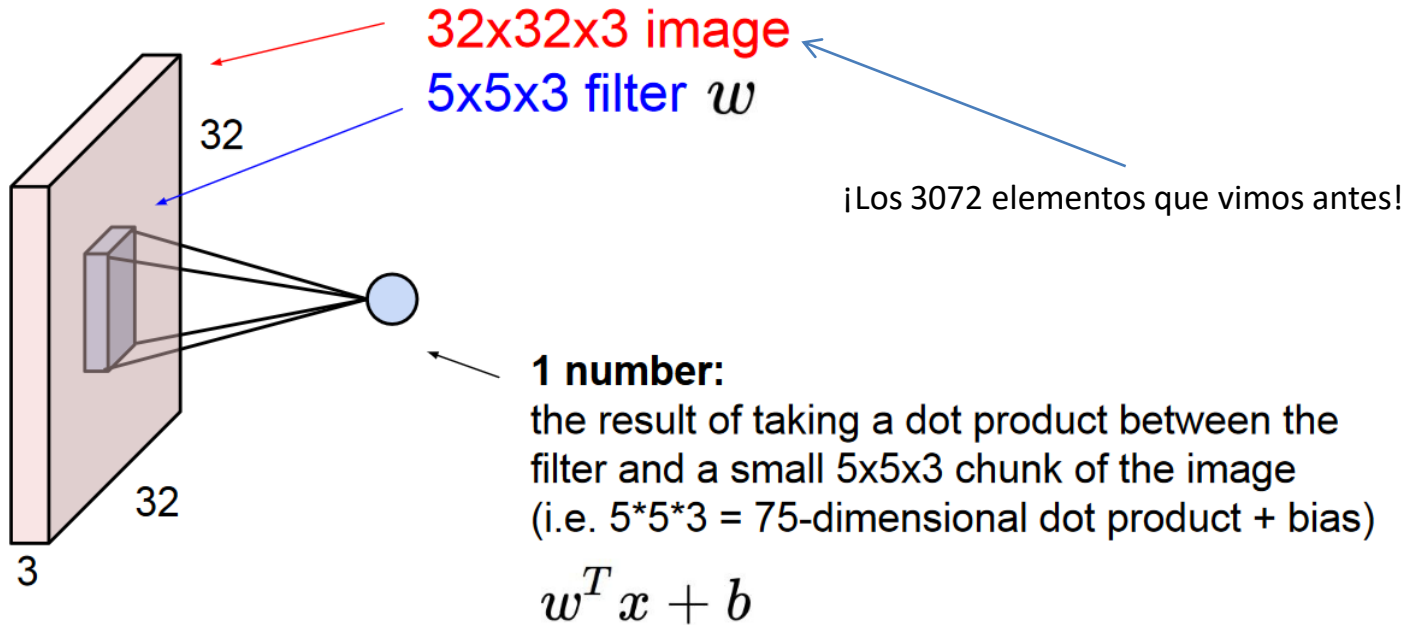
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

Capa totalmente conectada (*Fully-connected or dense layer*)

Todas las unidades/neuronas de una capa están conectadas (pesos) con todas las unidades/neuronas de la siguiente capa.



Capa Convolutiva

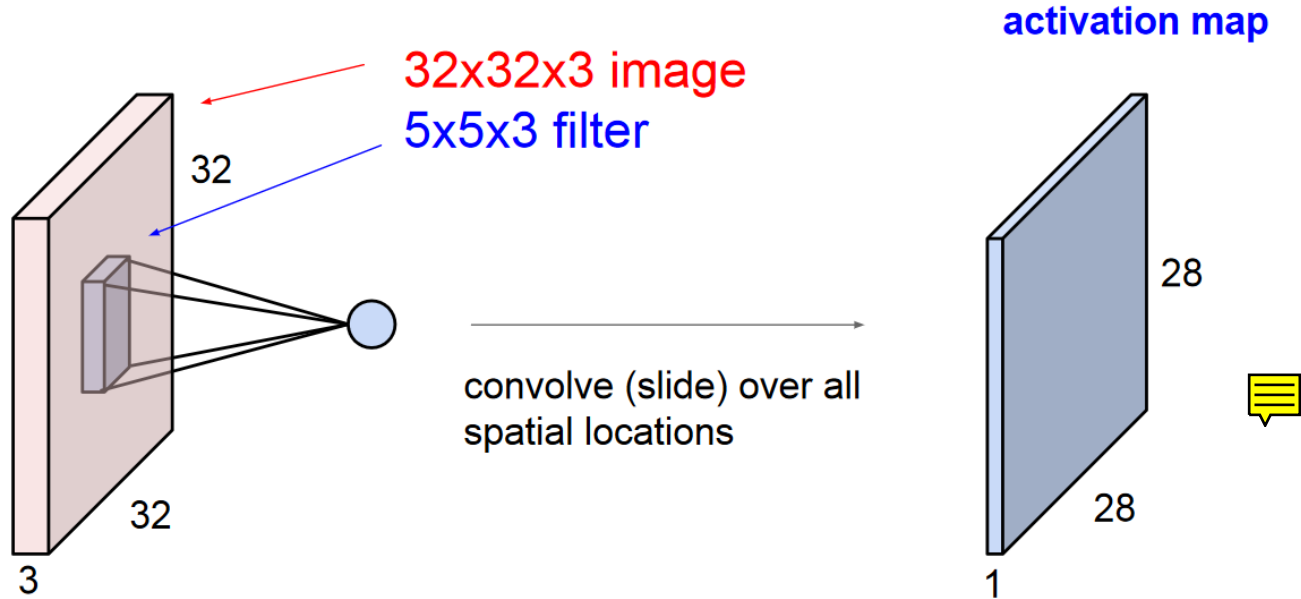


Gran ventaja:
76 pesos vs 30720!

Nota: recordemos que el *bias* aporta flexibilidad al aprendizaje (representaciones más ricas del espacio de entrada), permitiendo adaptar la salida de cada unidad. Ejemplo simple e intuitivo:

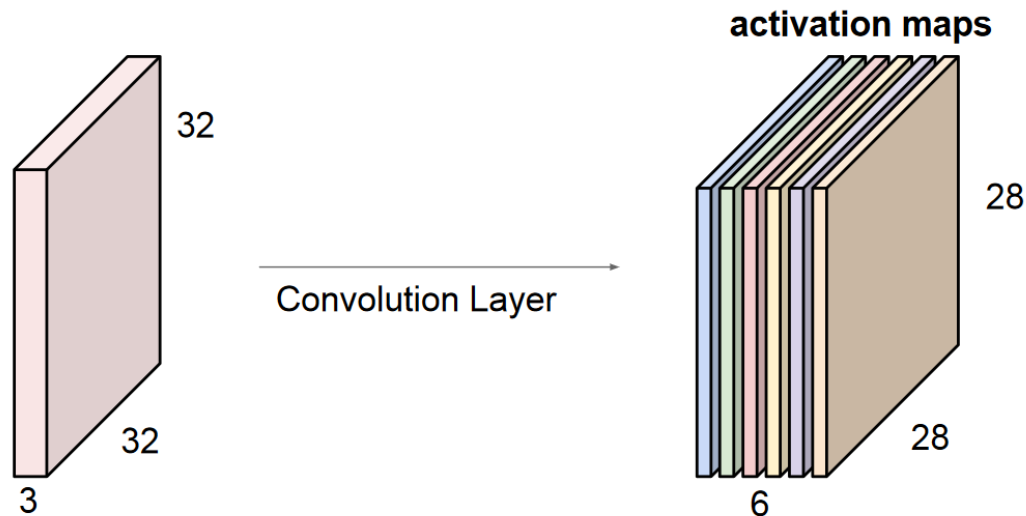
<https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks>

Capa Convolutucional



Capa Convolutiva

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



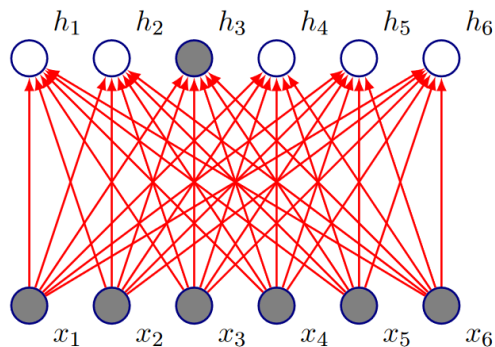
We stack these up to get a “new image” of size 28x28x6!

Incluso ahora el número de pesos es mucho menor que lo que teníamos antes: 456 pesos vs 30720!

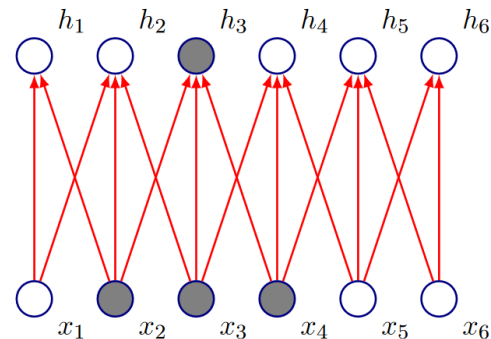
Ventajas de las ConvNets

- **Conectividad dispersa**

- *Fully-connected layers* operan globalmente (cada neurona/unidad “ve” toda la entrada), mientras que las capas convolucionales operan localmente → ¡Menos operaciones!



vs

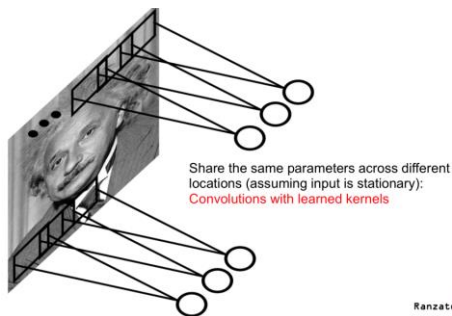


- Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity

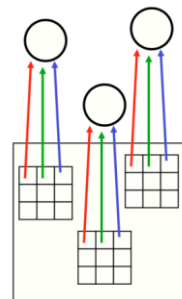
- Kernel of size 3, moved with stride of 1
- h_3 only depends on x_2, x_3, x_4

Ventajas de las ConvNets

- **Compartición de parámetros (*Weight Sharing*)**
 - El mismo filtro se aplica a toda la imagen.
 - En lugar de aprender un parámetro para cada localización en la imagen, solo se aprende un conjunto reducido de parámetros (los correspondientes al filtro).
 - El número de parámetros a aprender y almacenar se reduce considerablemente → ¡Regularización!



The red connections all have the same weight.



Ventajas de las ConvNets

- **Representaciones equivariantes**



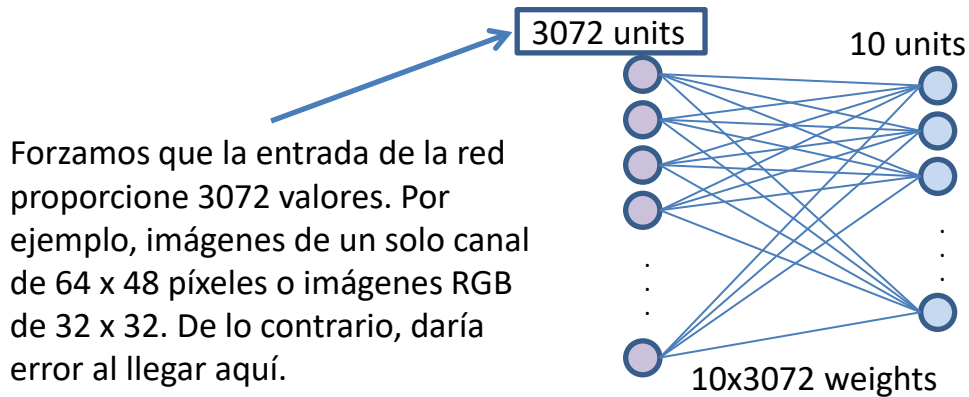
- Toda capa convolucional es equivariante con respecto a la traslación.

f es equivariante con respecto a g si $f(g(\mathbf{x})) = g(f(\mathbf{x}))$

- Si trasladamos un objeto en la imagen, su representación se trasladará a la misma distancia en la salida.
 - Permite generalizar la detección de bordes, texturas y formas en diferentes ubicaciones de la imagen
- La convolución no es equivariante con respecto al escalado o la rotación.

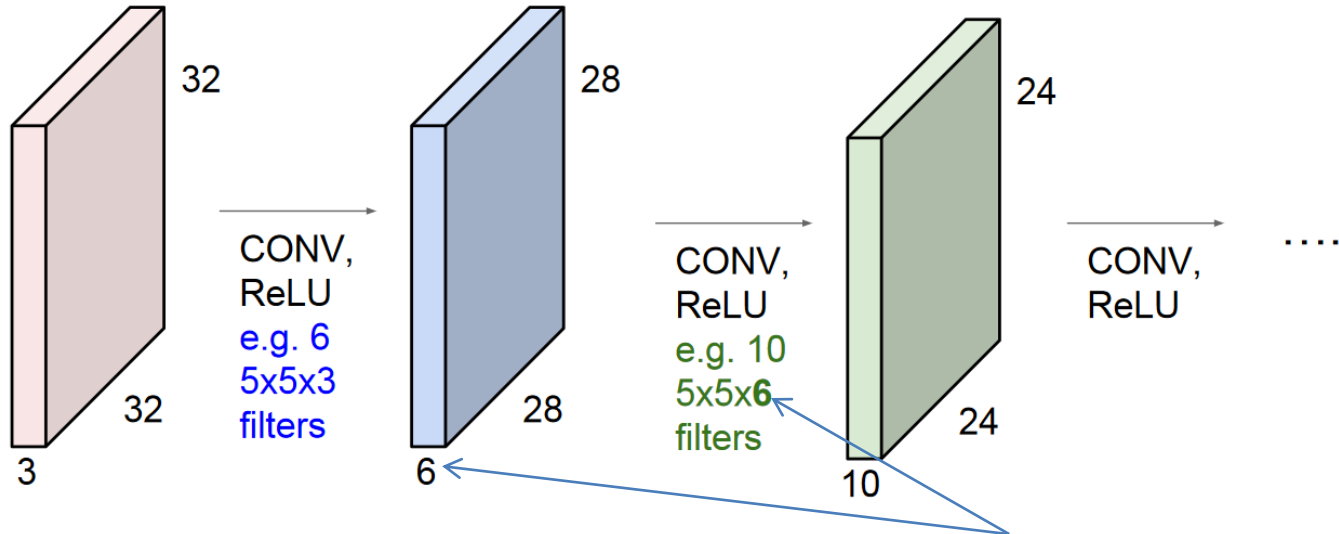
Ventajas de las ConvNets

- (Teóricamente) permiten **operar con entradas de tamaño variable**
 - Una *fully-connected layer* fuerza a que la entrada tenga un determinado tamaño.



Convolutional Networks

- Una ConvNet es una secuencia de capas convolucionales intercaladas con funciones de activación.

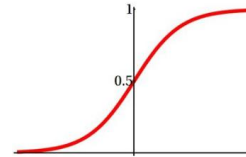


Nota muy importante: los filtros siempre tienen la misma profundidad que el bloque convolucional anterior, es decir, extienden toda la profundidad del volumen/tensor de entrada.

Convolutional Networks

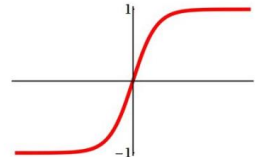
- Las capas convolucionales se intercalan con **funciones de activación (no lineales)**.
 - Tradicionalmente, las funciones de activación más empleadas eran **sigmoides**, como la *logistic sigmoid* o la hiperbólica tangente.
 - En redes profundas generalmente se recomienda utilizar la unidad lineal rectificada (**ReLU**), y sus variantes, entre otras razones porque permite entrenar más rápido.

$$\sigma(\Sigma) = \frac{1}{1 + e^{-\Sigma}}$$

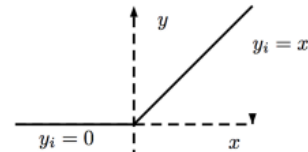


logistic (sigmoid, unipolar)

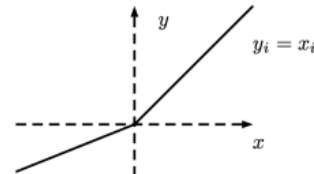
$$\tanh(\Sigma) = \frac{e^{\Sigma} - e^{-\Sigma}}{e^{\Sigma} + e^{-\Sigma}}$$



tanh (bipolar)



ReLU



Leaky ReLU/PReLU

Véase “dying ReLU problema” y “vanishing gradient problem”
([“Yes, you should understand backprop”](#) by Andrej Karpathy)

Convolutional Networks

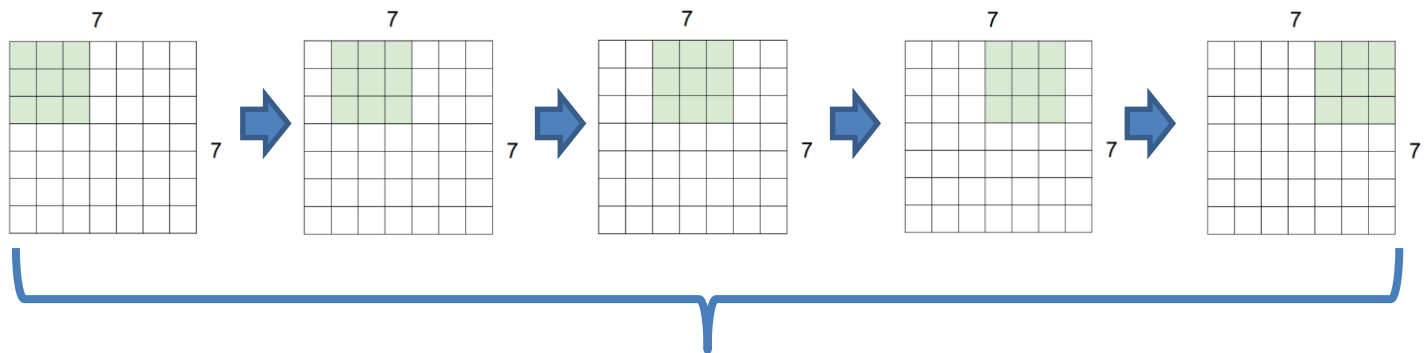
- Las capas convolucionales se intercalan con **funciones de activación (no lineales)**.
 - Esta no-linealidad es **necesaria para aprender representaciones complejas de los datos de entrada**.
 - De lo contrario, si solo se usaran funciones de activación lineal, la red neuronal se comportaría como una función lineal.

If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units.

"Pattern Recognition and Machine Learning" (C. M. Bishop, 2016) p.229

Convolutional Networks

- Prestemos atención a las dimensiones:
 - Ejemplo: imagen de 7x7 y filtro de 3x3

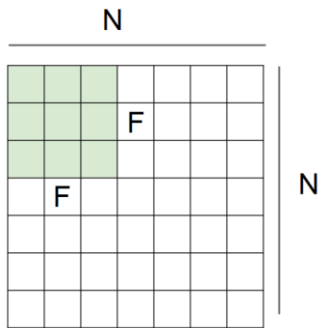
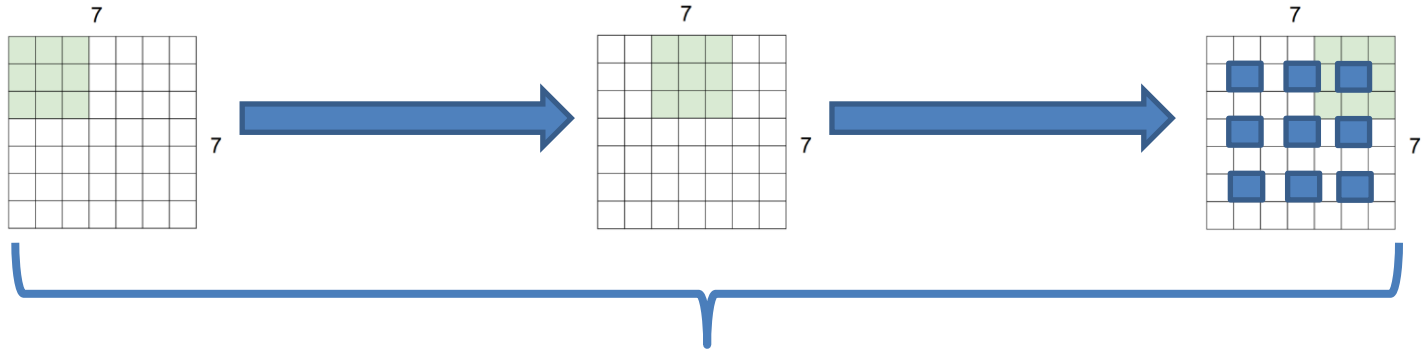


La salida es una imagen de 5x5
(no “reemplazamos” los bordes)

- Conceptos de **stride** y **padding**.

Convolutional Networks

- Prestemos atención a las dimensiones:
 - Ejemplo: imagen de 7x7 y filtro de 3x3, con **stride 2**



La salida es una imagen 3x3

Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7, F = 3$:
stride 1 $\Rightarrow (7 - 3) / 1 + 1 = 5$
stride 2 $\Rightarrow (7 - 3) / 2 + 1 = 3$
stride 3 $\Rightarrow (7 - 3) / 3 + 1 = 2.33$

¡Mal! A priori, aunque siempre hay trucos, no es posible aplicar un filtro 3x3 a una entrada de 7x7 con stride 3!

Convolutional Networks

- Progresivamente reducimos el tamaño de los mapas/volúmenes.
 - Si queremos conservar el tamaño → **padding**
 - Ejemplo: *zero-padding*

| | | | | | | | | |
|---|---|---|---|---|---|--|--|--|
| 0 | 0 | 0 | 0 | 0 | 0 | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!



En muchos casos es bueno utilizar *padding*, ya que la evidencia empírica aconseja no reducir la dimensionalidad demasiado rápido.

Convolutional Networks

- Ejemplo completo:
 - Volumen de entrada: **16x16x3**
 - Filtros usados: **10** filtros de **3x3** con stride **1** y padding **1**

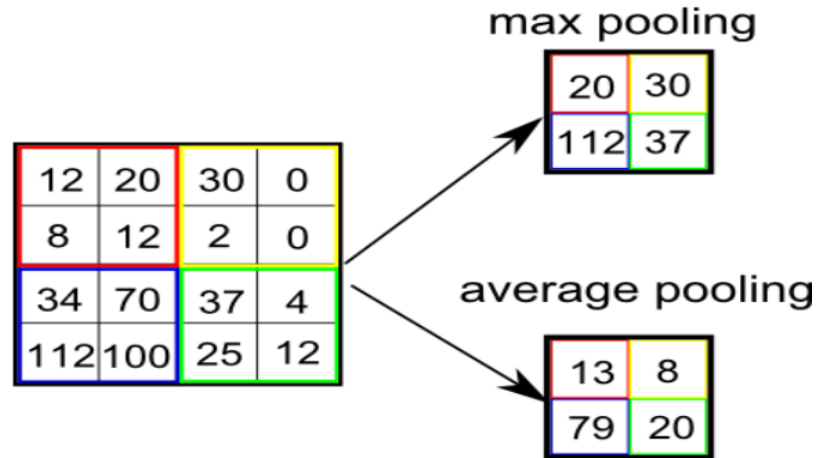
$$\text{Output_size} = ((N + 2 * P - F) / \text{stride}) + 1$$

- Volumen de salida?
 - $((16 + 2 * 1 - 3) / 1) + 1 = 16 \rightarrow 16 \times 16 \times 10$
 - Número de parámetros en esa capa?
 - Cada filtro tiene $3 \times 3 \times 3 + 1 = 28$ parámetros
 $\rightarrow 28 \times 10 = 280$ parámetros
- +1 por el *bias*

Convolutional Networks

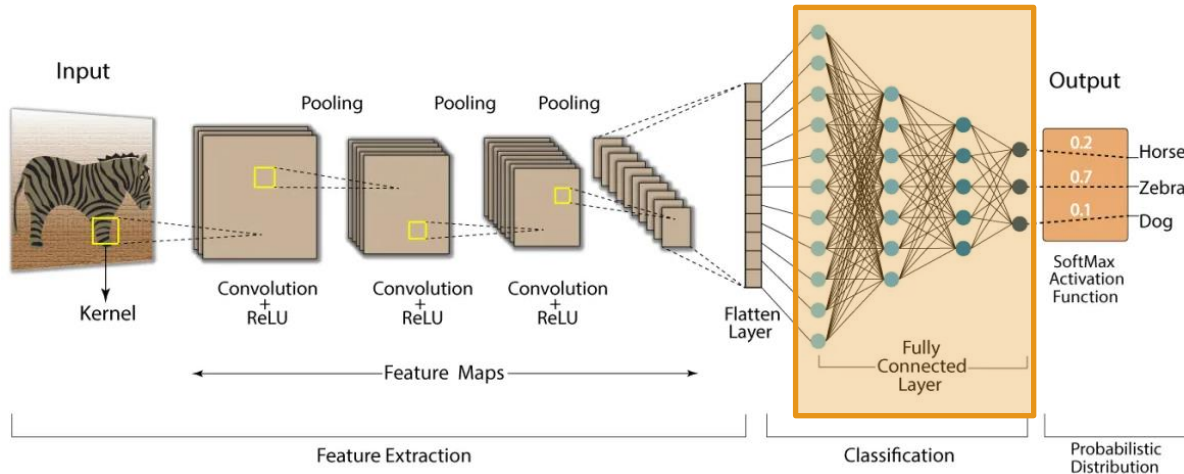
- **Pooling.** Reduce dimensionalidad e introduce cierta invarianza a (pequeñas) translaciones en la entrada
 - Si la entrada se desplaza una pequeña distancia, la mayoría de los valores de salida no cambiarían.

Ejemplo de **max** y
average pooling de
tamaño 2x2 y stride 2

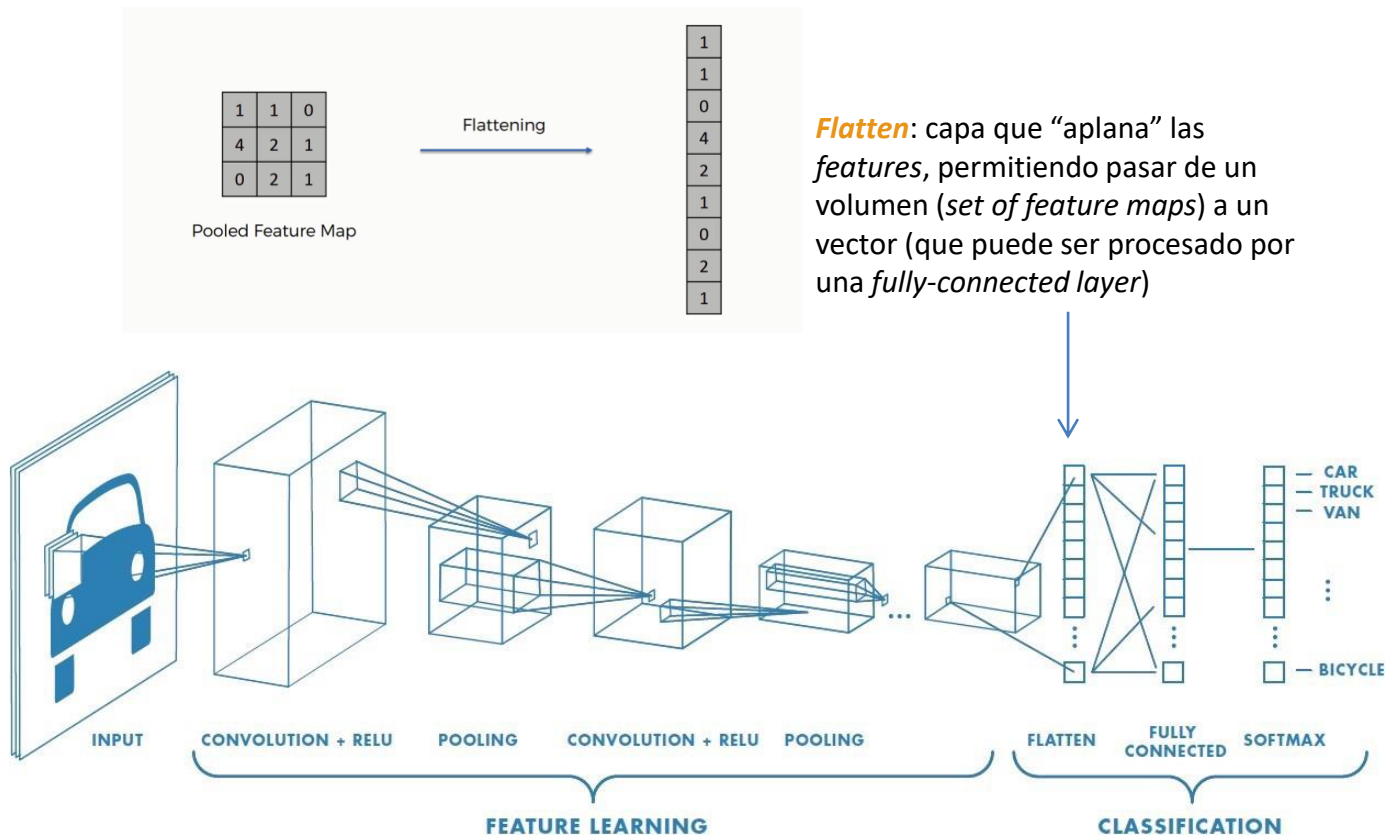


Convolutional Networks

- **Fully-connected** o **dense layers**.
 - Las redes convolucionales **pueden tener o no capas completamente conectadas**.
 - Cuando estas están presentes, suelen aparecer al final de la red.
 - Todas las unidades de una capa están conectadas a todas las unidades de la siguiente capa.
 - Suelen contener muchos parámetros, ¡así que cuidado al incluirlas!



Combinando todo



Combinando todo

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

σ = softmax

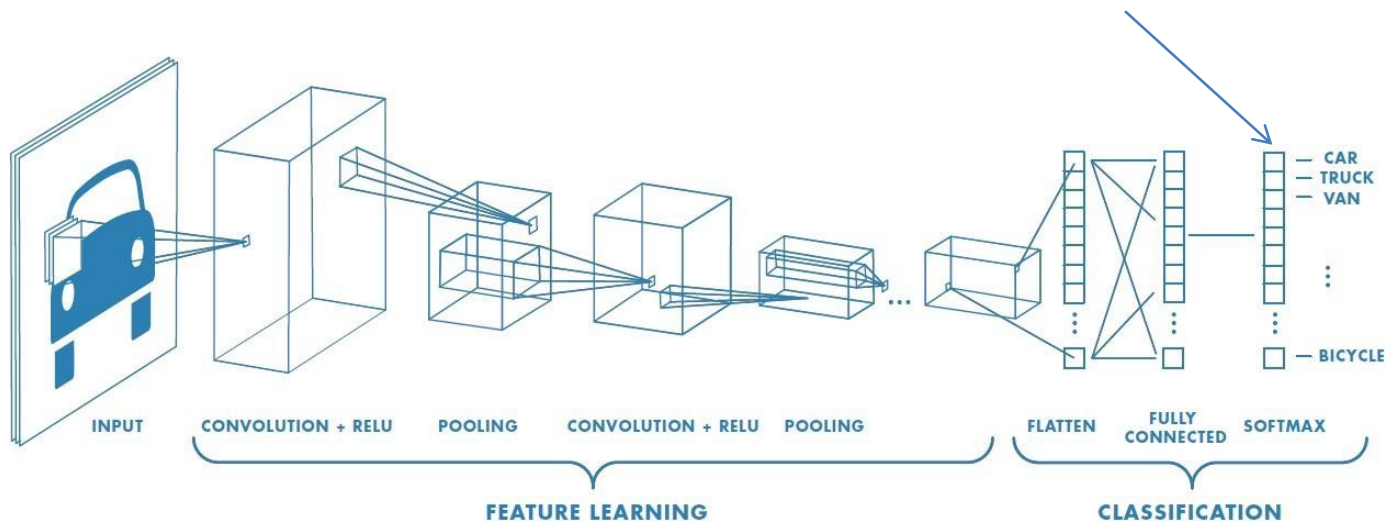
\vec{z} = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

Softmax: función de activación que generaliza la función sigmoide a múltiples clases. Normaliza la salida de la red, de modo que cada predicción corresponda a la probabilidad de que la entrada pertenezca a esa clase.

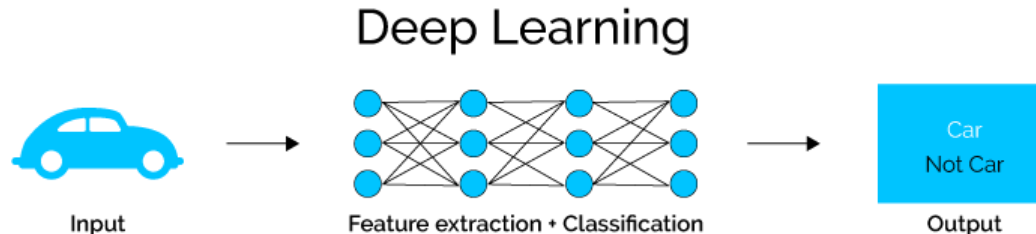
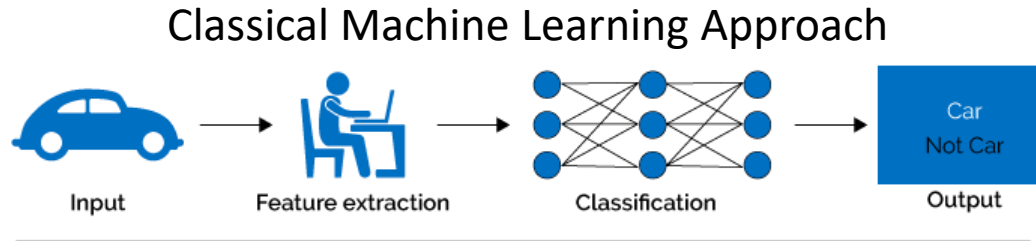


Visualizando modelos

- ConvNetJS. Deep Learning in your browser:
<https://cs.stanford.edu/people/karpathy/convnetjs/>
- A Neural Network Playground: <https://playground.tensorflow.org/>
- An Interactive Node-Link Visualization of Convolutional Neural Networks: https://adamharley.com/nn_vis/
- CNN Explainer: <https://poloclub.github.io/cnn-explainer/>
- ConvNet Playground: <https://convnetplayground.fastforwardlabs.com/>
- Topological visualisation of a convolutional neural network:
<https://terencebroad.com/works/cnn-vis>

Machine Learning vs Deep Learning

- Metodológicamente: ConvNets permiten aprender características (*feature learning*) en lugar de tener que diseñarlas a mano (*feature engineering*).
- Empíricamente: proporcionan **resultados superiores en muchas tareas**.

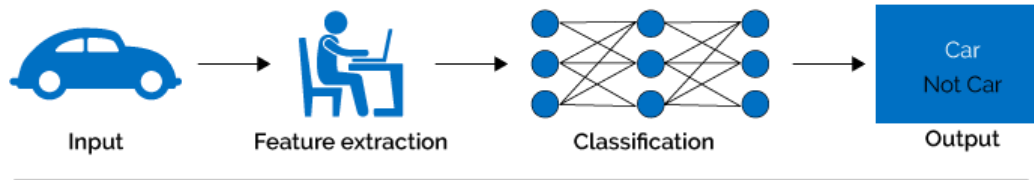


<https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e>

Machine Learning vs Deep Learning

- Deep Learning difumina los límites entre *feature extraction* e *image classification* (*end-to-end learning*).
- La clave es disponer de una **buena representación interna** de los datos de entrada.

Classical Machine Learning Approach



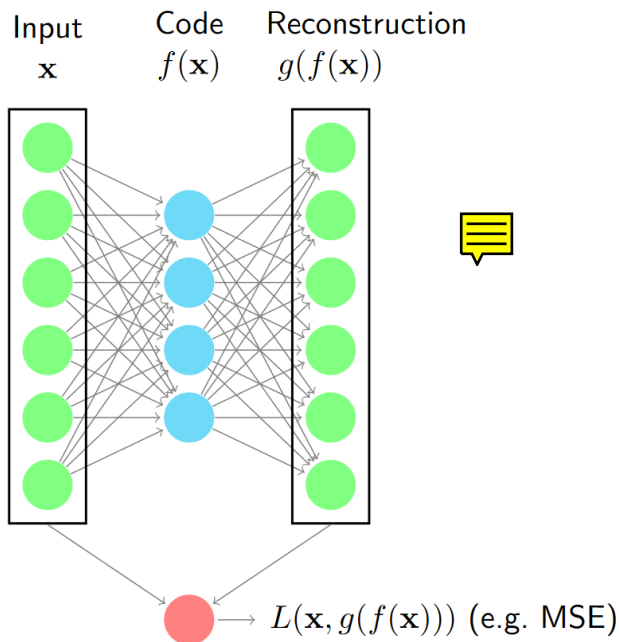
Deep Learning



<https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e>

Autoencoders

- Arquitecturas *encoder-decoder* que pueden ser empleadas en múltiples tareas, desde compresión a **eliminación de ruido** (*denoising*).

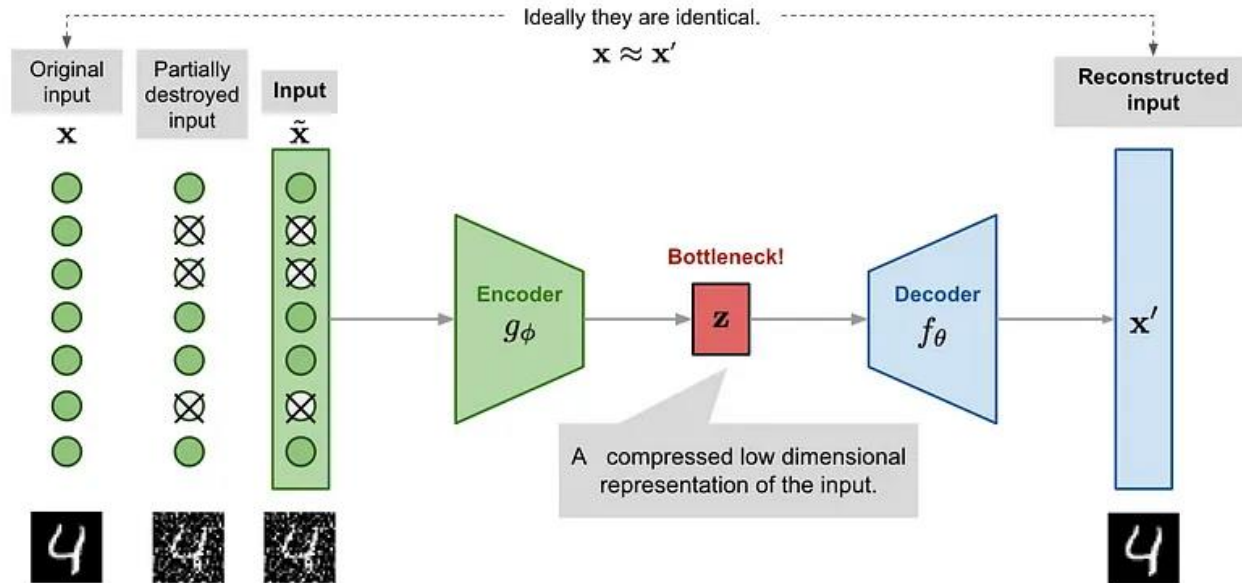


Eliminación de ruido usando autoencoders:

- Corrompemos los datos de entrada a propósito, agregando ruido o enmascarando algunos de los valores de entrada.
- El modelo está entrenado para predecir los datos originales no dañados.
- El autoencoder debe deshacer esta corrupción.

Queremos aprender una forma eficiente de codificar los datos de entrada.

Denoising Autoencoders



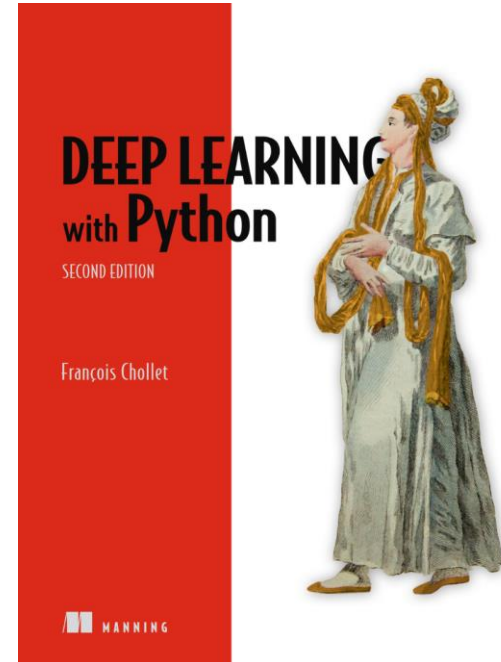
[Fuente](#)

Índice

- Normas de entrega
- Repaso de aprendizaje profundo
- **Introducción a Keras**
- Presentación de la práctica

Keras

- Las redes profundas y, en particular, las ConvNets se pueden programar en muchos lenguajes diferentes. Nosotros utilizaremos Keras: <https://keras.io/>
- Libro de referencia: [“Deep Learning with Python” \(Chollet, 2ª ed., 2021\). 1ª ed. de 2016](#)
- Notebooks compartidos por el autor (François Chollet): <https://github.com/fchollet/deep-learning-with-python-notebooks>



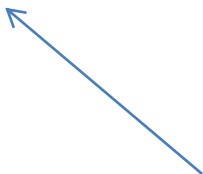
Keras

- Keras es una API de alto nivel para *Deep Learning* escrita en Python.
- Como backend utiliza TensorFlow (antes también empleaba Theano, CNTK, MXNet,...)
 - De hecho, es la API de alto nivel oficial de TensorFlow.
- La última versión es la 3.3.3 (<https://github.com/keras-team/keras/releases>)
- Documentación: <https://keras.io/>
- Código (GitHub): <https://github.com/keras-team/keras>

Keras: lectura de imágenes

- El vector con las imágenes tendrá dimensión (x, y, z, w):
 - x es el número de imágenes,
 - y es la altura de las imágenes,
 - z es la anchura de las imágenes,
 - w es el número de canales (1: monobanda; 3: tribanda)
- Ejemplo:

```
(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')
```



(50000, 32, 32, 3)

“50.000 imágenes de 32x32x3”



Hay datasets que ya están incorporados
directamente en Keras <https://keras.io/api/datasets/>

Keras: fases principales

- Las fases principales para crear, entrenar y usar un modelo para clasificación son las siguientes:
 1. Definición del modelo
 2. Declaración del optimizador
 3. Compilación del modelo
 4. Entrenamiento
 5. Predicción

Keras: Definición del modelo

- En Keras hay tres formas de definir redes neuronales (<https://keras.io/api/models/>): *Sequential*, *Model* y *Model subclassing*. Nos centraremos en los dos primeros.
 - *Sequential* (https://keras.io/guides/sequential_model/) fuerza a que todas las capas de la red vayan una detrás de otra de forma secuencial, sin permitir ciclos ni saltos entre las capas.
 - *Model* o *Functional* (https://keras.io/guides/functional_api/) permite cualquier tipo de red neuronal, incluyendo ciclos y saltos entre capas.
 - *Model subclassing* (https://www.tensorflow.org/guide/keras/custom_layers_and_models) permite implementar cualquier cosa *from scratch*. Se usa si se tienen casos de uso complejos y muy particulares.

Keras: Definición del modelo

- Con **Sequential** podemos usar el método *add* directamente sobre el modelo, y la nueva capa se añadirá después de la última capa añadida.

```
model = Sequential()  
model.add(Dense(50, input_dim=4, activation='relu'))  
model.add(Dense(12, activation='relu'))  
model.add(Dense(3, activation='softmax'))
```

- Con **Model** tenemos que especificar sobre qué capa estamos añadiendo la nueva capa.

```
input1 = Input(shape=(4,))  
hidden1 = Dense(50, activation='relu')(input1)  
hidden2 = Dense(12, activation='relu')(hidden1)  
output = Dense(3, activation='softmax')(hidden2)  
model = Model(inputs=input1, outputs=output)
```


Keras: Definición del modelo

- En nuestro caso, vamos a hacer **clasificación multiclase** y definiremos como última capa una capa *fully connected* (*Dense* en Keras) con tantas neuronas como clases tenga el problema, y una activación *softmax* para transformar las salidas de las neuronas en la probabilidad de pertenecer a cada clase.

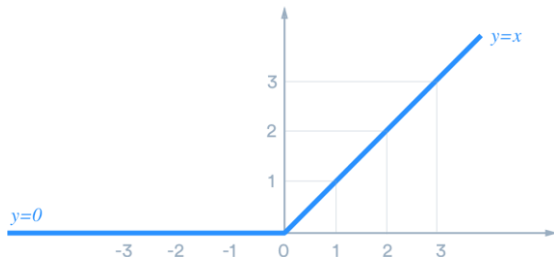
$$\text{softmax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^N \exp(z_k)}$$

donde \mathbf{z} es el vector de salida de la capa Dense y $\text{softmax}(\mathbf{z})$ es el vector que contiene en la componente j la probabilidad de que la imagen pertenezca a la clase j , para $j = 1, \dots, N$, con N el total de clases.

Keras: Definición del modelo

- El modo más habitual de introducir funciones de activación es detrás de cualquier capa, usando el argumento *activation* de esa capa. Lo siguiente introduciría una activación ReLU en una capa *Dense* con 128 unidades de procesamiento (neuronas):

```
model.add(Dense(128, activation='relu'))
```



Keras: Definición del modelo

En las prácticas vamos a usar algunas de las siguientes capas:

- *Fully connected*: `Dense(units, activation = None, ...)`
- *Dropout*: `Dropout(rate, noise_shape = None, seed = None)`
- *Flatten*: `Flatten()`
- *Convolución 2D*: `Conv2D(filters, kernel_size, strides = (1,1), padding = 'valid', activation = None, ...)`
- *Pooling 2D*: `MaxPooling2D(pool_size = (2,2), strides = None, ...)`. También tenemos, `AveragePooling2D()`, `GlobalMaxPooling()`, `GlobalAveragePooling()`,...
- *Batch Normalization*: `BatchNormalization()`

Keras: Definición del modelo

- Tened en cuenta que Keras cuenta con muchos más tipos de capas (<https://keras.io/api/layers/>):

The base Layer class

- Layer class
- weights property
- trainable_weights property
- non_trainable_weights property
- add_weight method
- trainable property
- get_weights method
- set_weights method
- get_config method
- add_loss method
- losses property

Layer activations

- relu function
- sigmoid function
- softmax function
- softplus function
- softsign function
- tanh function
- selu function
- elu function
- exponential function
- leaky_relu function
- relu6 function
- silu function
- hard_silu function
- gelu function
- hard_sigmoid function
- linear function
- mish function
- log_softmax function

Layer weight initializers

- RandomNormal class
- RandomUniform class
- TruncatedNormal class
- Zeros class
- Ones class
- GlorotNormal class
- GlorotUniform class
- HeNormal class
- HeUniform class
- Orthogonal class
- Constant class
- VarianceScaling class
- LecunNormal class
- LecunUniform class
- IdentityInitializer class

Regularization layers

- Dropout layer
- SpatialDropout1D layer
- SpatialDropout2D layer
- SpatialDropout3D layer
- GaussianDropout layer
- AlphaDropout layer
- GaussianNoise layer
- ActivityRegularization layer

Attention layers

- GroupQueryAttention
- MultiHeadAttention layer
- Attention layer
- AdditiveAttention layer

Merging layers

- Concatenate layer
- Average layer
- Maximum layer
- Minimum layer
- Add layer
- Subtract layer
- Multiply layer
- Dot layer

Activation layers

- ReLU layer
- Softmax layer
- LeakyReLU layer
- PReLU layer
- ELU layer

Backend-specific layers

- TorchModuleWrapper layer
- Tensorflow SavedModel layer
- JaxLayer
- FlaxLayer

Layer weight regularizers

- Regularizer class
- L1 class
- L2 class
- L1L2 class
- OrthogonalRegularizer class

Layer weight constraints

- Constraint class
- MaxNorm class
- MinMaxNorm class
- NonNeg class
- UnitNorm class

Core layers

- Input object
- InputSpec object
- Dense layer
- EinsumDense layer
- Activation layer
- Embedding layer
- Masking layer
- Lambda layer
- Identity layer

Recurrent layers

- LSTM layer
- LSTM cell layer
- GRU layer
- GRU Cell layer
- SimpleRNN layer
- TimeDistributed layer
- Bidirectional layer
- ConvLSTM1D layer
- ConvLSTM2D layer
- ConvLSTM3D layer
- Base RNN layer
- Simple RNN cell layer
- Stacked RNN cell layer

Preprocessing layers

- Text preprocessing
- Numerical features preprocessing layers
- Categorical features preprocessing layers
- Image preprocessing layers
- Image augmentation layers

Normalization layers

- BatchNormalization layer
- LayerNormalization layer
- UnitNormalization layer
- GroupNormalization layer

Attention layers

- GroupQueryAttention
- MultiHeadAttention layer
- Attention layer
- AdditiveAttention layer

Reshaping layers

- Reshape layer
- Flatten layer
- RepeatVector layer
- Permute layer
- Cropping1D layer
- Cropping2D layer
- Cropping3D layer
- UpSampling1D layer
- UpSampling2D layer
- UpSampling3D layer
- ZeroPadding1D layer
- ZeroPadding2D layer
- ZeroPadding3D layer

Convolution layers

- Conv1D layer
- Conv2D layer
- Conv3D layer
- SeparableConv1D layer
- SeparableConv2D layer
- DepthwiseConv1D layer
- DepthwiseConv2D layer
- Conv1DTranspose layer
- Conv2DTranspose layer
- Conv3DTranspose layer

Pooling layers

- MaxPooling1D layer
- MaxPooling2D layer
- MaxPooling3D layer
- AveragePooling1D layer
- AveragePooling2D layer
- AveragePooling3D layer
- GlobalMaxPooling1D layer
- GlobalMaxPooling2D layer
- GlobalMaxPooling3D layer
- GlobalAveragePooling1D layer
- GlobalAveragePooling2D layer
- GlobalAveragePooling3D layer

Keras: Definición del modelo

- Una vez el modelo está construido, podemos ver una descripción del mismo usando *summary* sobre el objeto creado: **`my_model.summary()`**

Model: "model"

| Layer (type) | Output Shape | Param # |
|----------------------|--------------|---------|
| input_1 (InputLayer) | [(None, 4)] | 0 |
| dense (Dense) | (None, 50) | 250 |
| dense_1 (Dense) | (None, 12) | 612 |
| dense_2 (Dense) | (None, 3) | 39 |

=====
Total params: 901 (3.52 KB)
Trainable params: 901 (3.52 KB)
Non-trainable params: 0 (0.00 Byte)

```
from keras.models import Model
from keras.layers import Input, Dense

def define_model_by_functional_api():
    input1 = Input(shape=(4,))
    hidden1 = Dense(50, activation='relu')(input1)
    hidden2 = Dense(12, activation='relu')(hidden1)
    output = Dense(3, activation='softmax')(hidden2)
    model = Model(inputs=input1, outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])

    return model

model = define_model_by_functional_api()

model.summary()
```

Keras: Definición del modelo

- Una vez el modelo está construido, podemos ver una descripción del mismo usando *summary* sobre el objeto creado: **`my_model.summary()`**

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-------------------------------------|--------------|---------|
| ===== | | |
| dense_3 (Dense) | (None, 50) | 250 |
| dense_4 (Dense) | (None, 12) | 612 |
| dense_5 (Dense) | (None, 3) | 39 |
| ===== | | |
| Total params: 901 (3.52 KB) | | |
| Trainable params: 901 (3.52 KB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

```
from keras.models import Sequential
from keras.layers import Input, Dense

def define_model_by_sequential_api():
    model = Sequential()
    model.add(Input(shape=(4,)))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(12, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])

    return model

model = define_model_by_sequential_api()

model.summary()
```

Keras: Declaración del optimizador

- Para poder modificar los parámetros del optimizador, es necesario declararlo previamente y crear un objeto.
 - Por ejemplo, para usar el gradiente descendente estocástico deberíamos declararlo y podríamos cambiar alguno de sus parámetros.

```
import tensorflow as tf
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=10000,
    decay_rate=0.9)
opt = tf.keras.optimizers.SGD(learning_rate=lr_schedule,
    momentum=0.9)
```

- Documentación de optimizadores: <https://keras.io/optimizers/>

Keras: Compilación del modelo

- La **función de pérdida** (*loss function*), o función objetivo que se va a usar (y que va a ser minimizada), depende del problema a resolver.
 - Clasificación binaria: `binary_crossentropy`
 - Clasificación multiclase: `categorical_crossentropy`
 - Regresión: `mean_squared_error`
- Documentación sobre las funciones de pérdida disponibles: <https://keras.io/api/losses/>

Keras: Compilación del Modelo

- Con el argumento **metrics** se pueden especificar las **métricas** (<https://keras.io/api/metrics/>) que se calcularán a lo largo de las épocas de entrenamiento.
 - clasificación multiclase: común usar la métrica **accuracy**, definida como el porcentaje de imágenes bien clasificadas.
- Para **compilar** (es decir, juntar modelo, función de pérdida, optimizador y métricas), usamos el método **compile()**:

```
my_model.compile(loss=keras.losses.categorical_crossentropy,  
                 optimizer=opt, metrics=['accuracy'],...)
```

Keras: Entrenamiento

- Una vez el modelo está compilado, podemos pasar a entrenarlo. Para ello, debéis usar:
 - el **método *fit()***: recibe los datos de entrada (en un NumPy array, un tensor de TensorFlow, o un *ImageDataGenerator*, entre otros) y realiza el ajuste de pesos.
 - *ImageDataGenerator*: se empleaba para hacer *data augmentation*, para usar alguna función de preprocesado, o para separar un conjunto de validación durante el entrenamiento. **Atención**: era una estrategia muy utilizada, pero [actualmente está deprecated](#).

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=50,  
         verbose=1, validation_split=0.2)
```


Keras: Entrenamiento

- Cuando se entrena un modelo con *fit()*, Keras guarda el estado del modelo por donde se ha quedado entrenando.
 - Esto quiere decir que si volvemos a usar *fit()*, el entrenamiento seguirá por donde se ha quedado, y no empezará desde el principio.
 - Si vamos a usar varias veces *fit()* sobre el mismo modelo definido previamente, tenemos que restablecer los pesos de la red a como estaban antes del entrenamiento o debemos re-crearlo de nuevo.
 - Esto se puede hacer guardando los pesos de la red antes del primer entrenamiento (y después de la compilación) usando:

```
weights = my_model.get_weights()
```
 - Y después restablecerlos antes del siguiente entrenamiento usando

```
my_model.set_weights(weights)
```

Keras: Entrenamiento

- ¿Cómo es posible que, cada vez que entrenamos un modelo en Keras, los resultados sean distintos?
 - estamos trabajando con **métodos estocásticos** 
 - Inicialización aleatoria de pesos, eliminación aleatoria de unidades en Dropout, transformaciones aleatorias de los datos en *data augmentation*, orden aleatorio de los *batches* de datos, etc.
 - la función *fit()* va a actualizar continuamente los pesos, de modo que si la llamamos varias veces con el mismo modelo actualizará progresivamente los pesos cada vez
 - Es decir, entrenará incrementalmente el modelo a partir de los pesos encontrados en el anterior entrenamiento.

Keras: Entrenamiento



- La clase *ImageDataGenerator*
 - **Nos permite normalizar los datos** (bien con media y varianza, o usando una función de preprocesado determinada), **usar *data augmentation*, o separar del conjunto de entrenamiento una parte para validación** (entre otras cosas).
 - Para usarla, tenemos que crear un objeto de esta clase y usarlo como generador de imágenes a la hora de entrenar y/o testear el modelo.
 - ***Data augmentation*, en un principio, solo debe usarse en el conjunto de entrenamiento.**
 - Recordad que la normalización debe hacerse en ambos conjuntos, pero **la normalización del conjunto de test debe hacerse con los parámetros de las imágenes de entrenamiento.**
 - <https://machinelearningmastery.com/image-augmentation-deep-learning-keras/>

Keras: Entrenamiento

- La clase *ImageDataGenerator*
 - Pero... la ejecución suele ser bastante más lenta que incorporar directamente capas de *data augmentation* en el propio modelo.

```
import tensorflow as tf
from keras import layers
data_augmentation = tf.keras.Sequential([
    layers.RandomRotation(0.05),
    layers.RandomContrast(0.7),
    layers.RandomTranslation(0.1,0.1)
])
```

```
model = tf.keras.Sequential([
    # Add the preprocessing layers you created earlier.
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    # Rest of your model.
])
```

Léase con calma la documentación oficial si se quieren emplear estrategias de *data augmentation*: https://www.tensorflow.org/tutorials/images/data_augmentation

★ **Note:** Data augmentation is inactive at test time so input images will only be augmented during calls to `Model.fit` (not `Model.evaluate` or `Model.predict`).

Keras: Entrenamiento

- Lo mismo que antes se hacía con la clase *ImageDataGenerator* ahora se puede hacer con capas de Keras o por otros medios:

| Estandarización de datos | Separación de conjunto de validación |
|--|---|
| <pre>datagen = ImageDataGenerator(featurewise_center = True, featurewise_std_normalization = True) # A continuación, se estiman los parámetros de normalización datagen.fit(imagenes_train)</pre> | <pre>datagen = ImageDataGenerator(validation_split = 0.1)</pre> |
| <pre>adapt_data = np.array([1., 2., 3., 4., 5.]) input_data = np.array([1., 2., 3.]) layer = tf.keras.layers.Normalization(axis=None) layer.adapt(adapt_data) layer(input_data) <tf.Tensor: shape=(3,), dtype=float32, numpy= array([-1.4142135, -0.70710677, 0.], dtype=float32)></pre> | <pre>model.fit(x_train, y_train, batch_size=batch_size, epochs=50, verbose=1, validation_split=0.1)</pre> |
| https://keras.io/api/layers/preprocessing_layers/numerical/normalization/ | https://keras.io/api/models/model_training_apis/#fit-method |

Keras: Entrenamiento

– Atención a los detalles:

- **`validation_split`** siempre escoge el porcentaje correspondiente a los últimos ejemplos del conjunto de entrenamiento para validación.
 - Dependiendo de cómo lo usemos, siempre estaríamos validando exactamente con los mismos ejemplos y, si los ejemplos están ordenados por clase, nada asegura que caigan ejemplos de distintas clases en dicho conjunto de validación (por lo que, al final, podríamos estar validando el modelo solamente con ejemplos de ciertas clases)

validation_split: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the `x` and `y` data provided, before shuffling.

Keras: Entrenamiento

– Atención a los detalles:

- Podríamos desordenar los ejemplos con anterioridad a pasárselos al *fit()*, o hacer uso de las funciones para generación de particiones de sklearn:

```
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(x_train,
    y_train, test_size=0.1, stratify=y_train)
```

Keras: Predicción

- Hay dos funciones principales:
 - *predict()*
 - https://keras.io/api/models/model_training_apis/#predict-method
 - `predicciones = my_model.predict(x_test)`
 - *evaluate()*
 - https://keras.io/api/models/model_training_apis/#evaluate-method
 - `predicciones = my_model.evaluate(x_test, y_test)`

Keras: Cálculo de Accuracy

- Una vez tenemos las predicciones, podemos calcular el porcentaje de ejemplos de test que el modelo clasifica bien (*accuracy*).

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```



```
def calcularAccuracy(labels, preds):
    labels = np.argmax(labels, axis = 1)
    preds = np.argmax(preds, axis = 1)
    accuracy = sum(labels == preds)/len(labels)
    return accuracy
```

- Acordaos de que tenéis a vuestra disposición todo el conjunto de métricas de Keras (<https://keras.io/api/metrics/>) y scikit-learn (https://scikit-learn.org/stable/modules/model_evaluation.html).

Keras: Redes Pre-entrenadas

- Keras tiene casi 40 redes populares ya creadas
 - no es necesario construirlas desde cero cada vez.
- Están preentrenadas en *ImageNet*
 - si se quiere, se puede partir el entrenamiento desde ahí.
- Estos modelos están en <https://keras.io/api/applications/>

Índice

- Normas de entrega
- Repaso de aprendizaje profundo
- Introducción a Keras
- **Presentación de la práctica**

Ejercicio 1: Clasificación de imágenes MNIST

(7 ptos)

Apartado 1: implementación de arquitectura convolucional dada

| Layer Type | Kernel Size (for convolutional layers) | Input Output dimension | Input Output channels (for convolutional layers) |
|------------|--|-----------------------------|---|
| Conv | 3x3 | 28x28 28x28 | 1 32 |
| ReLU | - | 28x28 28x28 | - |
| MaxPooling | 2x2 | 28x28 14x14 | - |
| Conv | 5x5 | 14x14 10x10 | 32 16 |
| ReLU | - | 10x10 10x10 | - |
| MaxPooling | 2x2 | 10x10 5x5 | - |
| FC | - | 400 100 | - |
| ReLU | - | 100 100 | - |
| FC | - | 100 50 | - |
| ReLU | - | 50 50 | - |
| FC | - | 50 10 | - |

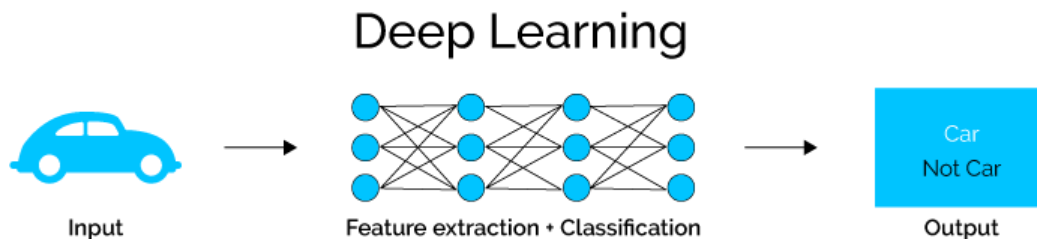
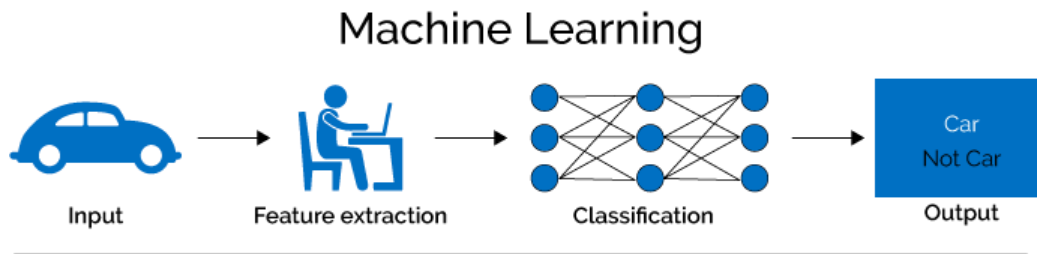


**Arquitectura
que tenéis que
implementar
en Keras**

Ejercicio 1: Clasificación de imágenes MNIST

(7 ptos)

Apartado 2: comparación con técnicas clásicas (SVM+HOG)



Ejercicio 1: Clasificación de imágenes MNIST

(7 ptos)

Apartado 3: implementación/diseño/experimentación con el modelo profundo que se desee (empleando CIFAR10 si se satura el rendimiento en MNIST)

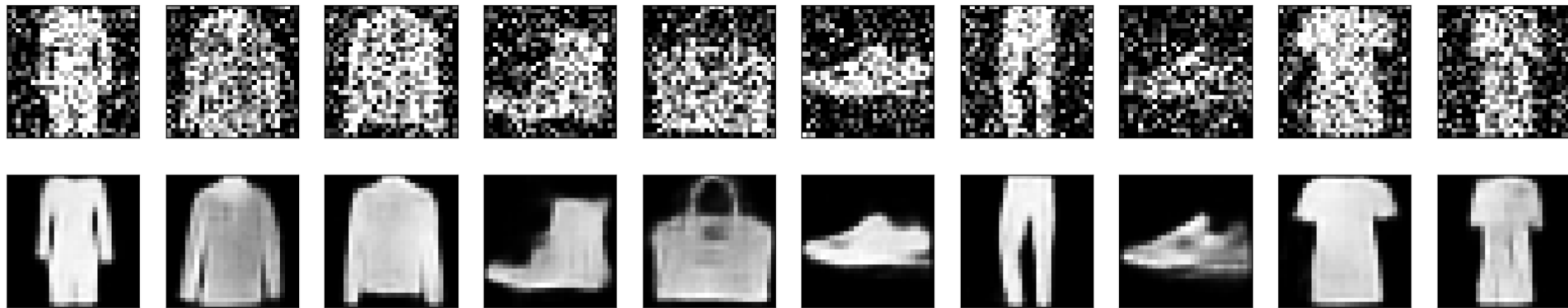
- Debéis mejorar la red por medio de aquellas alternativas que juzguéis vosotros:
 - Aumento de datos
 - Aumento de profundidad de la red y número/tamaño de filtros por bloque
 - Batch Normalization
 - Regularización (p.ej. Dropout)
 - ¿Otros?
- Recordad justificar siempre vuestras decisiones y mostrar claramente en el informe la arquitectura final resultante.



Ejercicio 2: Eliminación de ruido con autoencoders

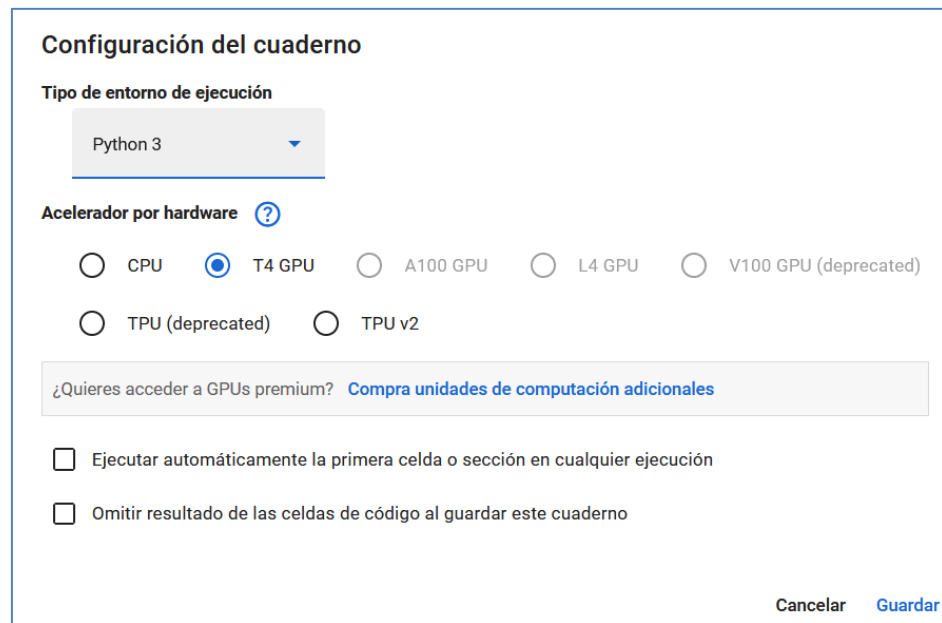
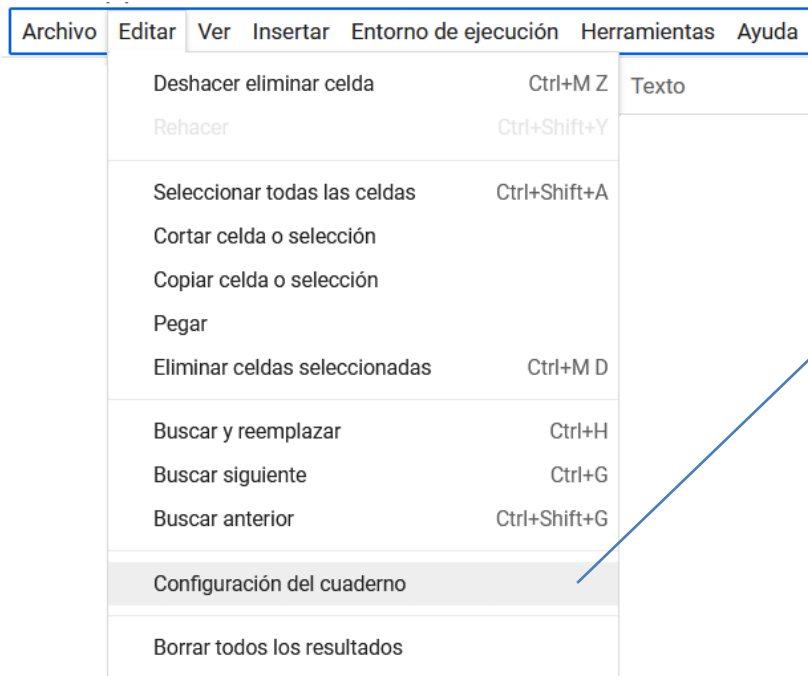
(3 puntos)

- Tenéis que implementar la arquitectura que se os indica y realizar una serie de experimentos en Fashion MNIST.
 - Objetivo: *image denoising*



En relación a Colab

- Acordaos de emplear **GPUs** para el entrenamiento de los modelos.



¿Problemas con Colab?

- Consejo general: **usad Colab de forma razonable**. De lo contrario,
 - si lanzáis experimentos, los cortáis abruptamente para lanzar otros, lanzáis experimentos demasiado largos, tenéis el Notebook abierto pero inactivo durante mucho tiempo, etc., puede que Colab limite vuestro uso de las GPUs.
 - Podéis tener problemas con la memoria RAM si no dimensionáis bien vuestros modelos.
- En cualquier caso, **NO paguéis por la versión Colab Pro!!!**
- Si empleáis buenas prácticas de implementación y se llevan a cabo experimentos razonables de manera ordenada, no debería haber ningún problema.

¿Problemas con Colab?

1) Modificad los servicios de Google Colab.

- Por ejemplo, incrementad la RAM disponible en Colab (<https://analyticsindiamag.com/5-google-colab-hacks-one-should-be-aware-of/>)

2) Optimizad el código.

- El tipo de dato empleado (p.ej., <https://stackoverflow.com/questions/62977311/how-can-i-stop-my-colab-notebook-from-crashing-while-normalising-my-images>).
- También es recomendable eliminar objetos innecesarios que podrían estar en memoria (`del command`) y/o utilizar el *garbage collector* para liberar memoria (<https://stackoverflow.com/questions/61188185/how-to-free-memory-in-colab>)

3) En último término, dividid el Notebook en varios archivos, que se ejecutarían de forma independiente. Al reiniciar el *runtime* entre ejercicios no debería haber ningún problema.

Consejo General

- “It's only by practicing (and failing) a lot that you will get an intuition of how to train a model.” (Jeremy Howard)
- **Id iterando pausadamente**, modificando un elemento cada vez. De lo contrario **aislar qué componente está influyendo en el rendimiento del modelo puede ser difícil.**

Prácticas de Aprendizaje Automático

Presentación de la Práctica 3 e Introducción a Keras

Pablo Mesejo y Salvador García

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA

