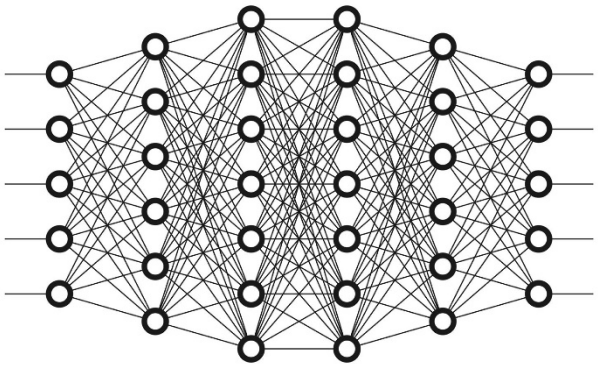
Deep Learning aplicado al análisis de señales e imágenes

INTRODUCCIÓN A KERAS

¿Qué es Keras?

- **Keras** es un *framework* de alto nivel para el entrenamiento de redes neuronales. Esta librería fue desarrollada por François Chollet en 2015 con el objetivo de simplificar la programación de algoritmos basados en aprendizaje profundo.

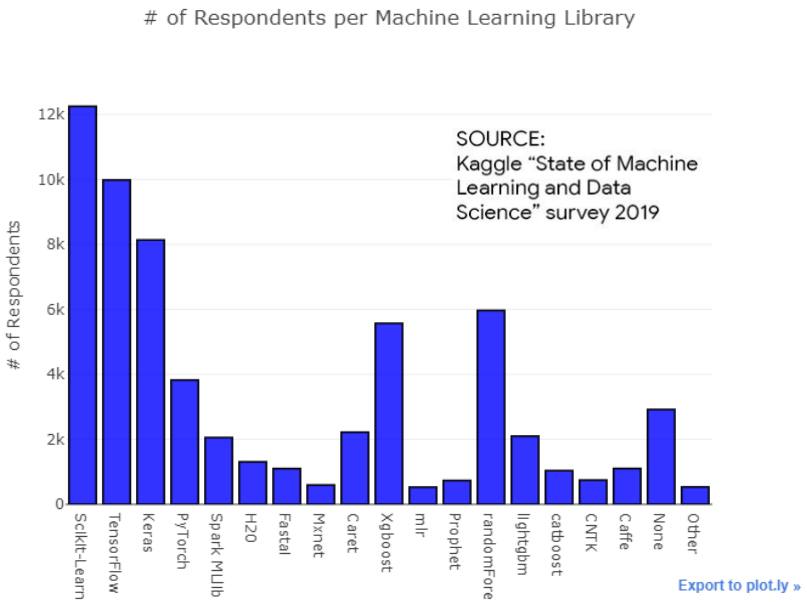
- Ofrece un conjunto de abstracciones más intuitivas y de alto nivel. El entrenamiento se puede seguir haciendo en **GPU**, recordemos que es la única forma que tenemos de **entrenar una red neuronal** en un intervalo de **tiempo permisible**.



¿Por qué Keras?

- Necesidad de **código optimizado** y **ampliamente validado**. Debido a estos dos aspectos, cuando se desean entrenar redes neuronales se hace uso de **librerías de más alto nivel ya específicas para dicho propósito**.

- Keras hace uso de librerías de más bajo nivel o ***backend*** por detrás, concretamente **TensorFlow** (anteriormente a TF2.0 se podía escoger el backendend: **TF**, **Microsoft Cognitive Toolkit** o **Theano)**.



Capas básicas en Keras

```
keras.layers.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros')

keras.layers.Activation(activation)

keras.layers.Dropout(rate, noise_shape=None, seed=None)

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid')

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None,
padding='valid', data_format=None)

keras.layers.BatchNormalization(axis=-1, momentum=0.99,
epsilon=0.001, center=True, scale=True)

keras.layers.Flatten(data_format=None)
```

- Es muy importante **utilizar la documentación de Keras** para conocer las distintas entradas que tiene cada capa.

https://keras.io/layers/core/

Definiendo una arquitectura secuencialmente

- **Modo (o API) secuencial**: Se instancia un **objeto del tipo Model ()** y a este **se le van añadiendo las capas** que conforman la arquitectura una detrás de otra.

```
# Imports necesarios
from keras.models import Sequential
from keras.layers import Dense, Activation
# Creamos objeto modelo secuencial
model = Sequential()
# Definimos la arquitectura añadiendo capas al modelos
model.add(Dense(64, input_dim=784))
model.add(Activation('relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

https://keras.io/models/sequential/

Definiendo una arquitectura funcionalmente

- Modo (o API) funcional: **Se define una entrada** y a partir de las mismas **se va definiendo la arquitectura** (**indicando** cuál es la **entrada** a cada capa). **Una vez definida** la arquitectura **se crea el objeto modelo** pasándole las entradas y las salidas (última capa definida).

```
# Imports necesarios
from keras.layers import Input, Dense
from keras.models import Model
# Definimos la entrada
inputs = Input(shape=(784,))
# Definimos la arquitectura
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
# Creamos el objeto modelo como unión de las inputs y la arquitectura
model = Model(inputs=inputs, outputs=predictions)
```

https://keras.io/models/model/

Compilación de un modelo Keras

- **Antes de entrenar** el modelo **en Keras** es necesario realizar una **compilación** del mismo **configurando** el proceso de **entrenamiento**. Este proceso se lleva a cabo mediante el comando **model.compile**.

```
# Ejemplo para un problema de clasificación multicalse
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Ejemplo para un problema de clasificación binario
 model.compile(optimizer=SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True),
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Ejemplo para un problema de regresión
 model.compile(optimizer='rmsprop',
              loss='mse')
```

Entrenamiento en Keras

- **Una vez compilado** el modelo **en Keras ya es posible** lanzar el proceso de **entrenamiento**. Keras entrena modelos a partir de **datos y etiquetas** almacenadas en **Numpy arrays** haciendo uso del método **model.fit**.

```
# Generación de datos

import numpy as np

data = np.random.random((1000, 100))

labels = np.random.randint(2, size=(1000, 1))

# Ejemplo con etiquetas en one-hot encoding

model.compile(optimizer='rmsprop',loss='categorical_crossentropy',

              metrics=['accuracy'])

one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

model.fit(data, one_hot_labels, epochs=10, batch_size=32)

# Ejemplo etiquetas categóricas

model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy',

              metrics=['accuracy'])

model.fit(data, labels, epochs=10, validation_data=(data_v, labels_v),
batch_size=32)
```

Predicción y evaluación en Keras

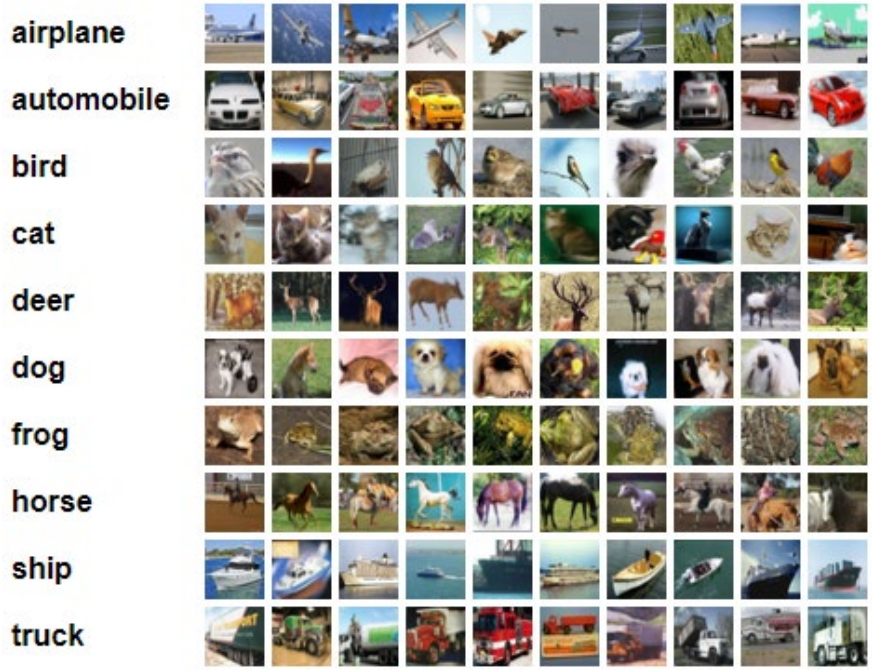
- Una vez disponemos del **modelo entrenado** se debe llevar a cabo una **evaluación del mismo**. Para ello, se realiza la predicción de los datos de test con el comando **model.predict** y posteriormente se obtienen **métricas de *performance* o se emplea model.evaluate**.

```
# Ejemplo con model.evaluate

model.evaluate(x=X_test, y=y_test, batch_size=None)

%% Devuelve los valores de pérdidas y métricas empleadas en entrenamiento para los datos de test


# Ejemplo con model.predict y classification_report

from sklearn.metrics import classification_report

model.predict(x=X_test, batch_size=None)

print(classification_report(y_labels,
      predictions.argmax(axis=1))

%% Devuelve más métricas
```

CIFAR-10 dataset

- El **dataset CIFAR-10** es un set de datos que contiene imágenes de 10 clases distintas y es emplea para propósitos de clasificación.

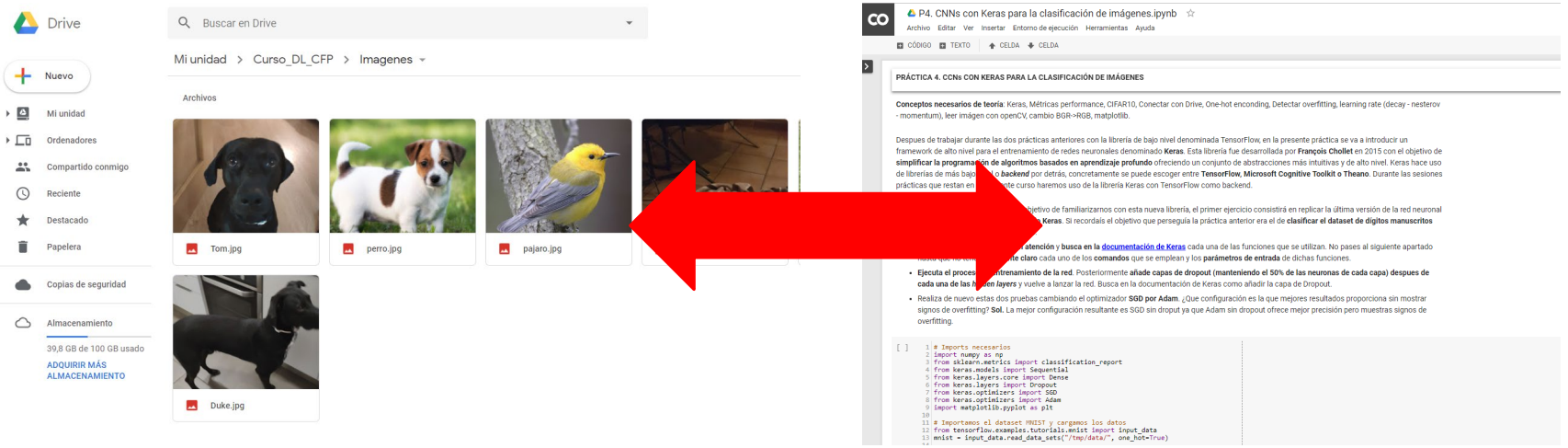
- Concretamente CIFAR-10 consiste en **60000 imágenes 32x32** repartidas en 6000 imágenes pertenecientes a cada una de las 10 clases. El conjunto de datos esta dividido en **50000 imágens de entrenamiento y 10000 para test**.



Carga de datos en Colab desde Drive

- Debido al entorno en el que estamos obligados a desarrollar para disponer de una GPU de manera gratuita (**Google Colab**) se hace indispensable la comunicación con **Google Drive** para la **lectura, escritura de datos**.

```
from google.colab import drive # Paquete para la comunicación

drive.mount('/content/drive') # Montamos la unidad de Drive

img_path = "/content/drive/My Drive/Curso_DL_CFP/Imagenes/coche.jpg"

img = cv2.imread(img_path, cv2.IMREAD_COLOR)
```



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Instituto de Investigación e Innovación en Bioingeniería

CVBLAB

DEPARTAMENTO DE COMUNICACIONES

CENTRO FORMACIÓN PERMANENTE

Deep Learning aplicado al análisis de señales e imágenes

INTRODUCCIÓN A TENSORFLOW