**Script en Python. Google Colaboratory**

**Se recomienda abrir un Jupyter Notebook y copiar las diferentes secciones del script en diferentes celdas.**

**Importar librerías**

import tensorflow as tf

import numpy as np

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

#Confusion Matrix and classification report

from sklearn.metrics import  confusion\_matrix, classification\_report

**Cargar datos**

(train\_images,train\_labels),(test\_images,test\_labels)=

datasets.cifar10.load\_data()

**Ver la estructura de los datos**

#print the structure of data

print(train\_images.shape)

print(test\_images.shape)

print(train\_labels.shape)

print(train\_labels[1])

#print(test\_images[1])

**Visualizar las primeras imágenes**

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))

for i in range(25):

    plt.subplot(5,5,i+1)

    plt.xticks([])

    plt.yticks([])

    plt.grid(False)

    plt.imshow(train\_images[i], cmap=plt.cm.binary)

    # The CIFAR labels happen to be arrays,

    # which is why you need the extra index

    plt.xlabel(class\_names[train\_labels[i][0]])

plt.show()

**Definir los modelos**

Se definen diferentes funciones, una para cada posible modelo

Incorporamos dos modelos simples como base: una CNN y un Perceptron Multicapa

def create\_model\_simple\_cnn():

  model = models.Sequential()

  model.add(layers.BatchNormalization(input\_shape=(32, 32, 3)))

  model.add(layers.Conv2D(16, (3, 3), activation='relu',padding='same'))

  model.add(layers.MaxPooling2D((2, 2)))

  model.add(layers.Flatten())

  model.add(layers.Dense(32, activation='relu'))

  model.add(layers.Dense(10, activation='softmax'))

  return model

def create\_simple\_pm():

    #modelo simple de pm

#se usa la sigmoide pero puede utilizarse la función de activación relu

    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.BatchNormalization(input\_shape=(32, 32, 3)))

    model.add(tf.keras.layers.Flatten(input\_shape=(32, 32, 3), name="Input\_layer"))

    model.add(tf.keras.layers.Dense(50, activation='sigmoid', name="Hidden\_layer"))

    model.add(tf.keras.layers.Dense(10, activation='softmax', name="Output\_layer"))

    return model

Es posible incluir dropout cuando se definen los modelos, para eliminar neuronas (y por tanto los pesos asociados) con una determinada probabilidad p. Por ejemplo

def create\_model\_simple\_cnn():

  model = models.Sequential()

  model.add(layers.BatchNormalization(input\_shape=(32, 32, 3)))

  model.add(layers.Conv2D(16, (3, 3), activation='relu',padding='same'))

model.add(layers.Dropout(0.75))

model.add(layers.MaxPooling2D((2, 2)))

  model.add(layers.Flatten())

  model.add(layers.Dense(32, activation='relu'))

  model.add(layers.Dense(10, activation='softmax'))

  return model

**Asignar el modelo deseado y mostrar una descripción esquemática**

# model = create\_simple\_pm()

model = create\_model\_simple\_cnn()

model.summary()

**Compilar y entrenar el modelo**

Compilar: se decide el optimizador, la función de pérdida y la métrica

model.compile(

      optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-3, ),

      #optimizer=tf.keras.optimizers.RMSprop(learning\_rate=0.001, rho=0.9),

      loss='sparse\_categorical\_crossentropy',

      metrics=['sparse\_categorical\_accuracy'])

Entrenar: se le pasa el conjunto de entrenamiento y el de validación así como el número de épocas, y la frecuencia con la que se realiza la validación.

historico = model.fit(train\_images, train\_labels, epochs=10, validation\_freq=1,

                      validation\_data=(test\_images, test\_labels))

# en histórico se guardan los valores de loss y accuracy obtenidos

# en el entrenamiento

### Plot evolución

# summarize history for accuracy

plt.plot(historico.history['sparse\_categorical\_accuracy'])

plt.plot(historico.history['val\_sparse\_categorical\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.ylim([0, 1])

plt.show()

# summarize history for loss

plt.plot(historico.history['loss'])

plt.plot(historico.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

**Obtener las medidas de evaluación del modelo con el conjunto de test**

evaluacion=model.evaluate(test\_images, test\_labels)

**Obtener predicciones en bruto y mostrar las 10 primeras**

#predicciones en bruto:

raw\_testPred = model.predict(test\_images)

#prediccion en bruto de los 10 primeros patrones de test:

#10 vectores de valores reales

print(raw\_testPred[:10])

**Obtener predicciones con asignación de la clase y mostrar las 10 primeras**

#predicciones de la clase:

class\_testPred = np.argmax(raw\_testPred, axis=1)

#predicción de la clase de los 10 primeros patrones de test

print(class\_testPred[:10])

**Obtener la matriz de confusión**

#Confusion Matrix

cm=confusion\_matrix(test\_labels, class\_testPred)

print(cm)

# También se puede sacar clasification report, con el código que te pongo, pero igual esto no es necesario. Creo que recall indica el % de aciertos por clases

**Obtener un informe completo de la clasificación con el conjunto de test**

print('Classification Report')

target\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

               'dog', 'frog', 'horse', 'ship', 'truck']

print(classification\_report(test\_labels, class\_testPred, target\_names=target\_names))

**Guardar resultados**

#guardar resultados

np.savetxt('evaluacion.txt',evaluacion,newline='\t')

np.savetxt('historicoTrainLoss.txt',historico.history['loss'])

np.savetxt('historicoTestLoss.txt',historico.history['val\_loss'])

np.savetxt('historicoTrainAcc.txt',historico.history['sparse\_categorical\_accuracy'])

np.savetxt('historicoTestAcc.txt',historico.history['val\_sparse\_categorical\_accuracy'])

# matriz de confusión

np.savetxt('matrizConf.txt', cm, fmt='%-3d')

#para guardar el modelo completo

model.save('modelo.h5')

#para guardar solo los pesos

model.save\_weights('pesos.h5')

Se pueden ver los archivos en la carpeta actual.

Con el ratón, botón derecho, se puede descargar en nuestro explorador de ficheros

También se pueden copiar en Drive.