from tensorflow import keras

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras import regularizers

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D,MaxPooling2D, Flatten, Dense, Dropout, Activation, BatchNormalization

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.preprocessing.image import ImageDataGenerator

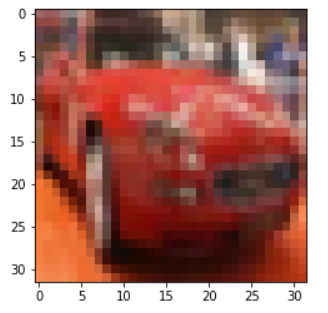
from tensorflow.keras.callbacks import ModelCheckpoint

import numpy as np

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

plt.imshow(x\_train[5])



**Limpieza de datos**

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

**#Numero de clasificaciones de salida**

num\_clases = len(np.unique(y\_train))

y\_train = to\_categorical(y\_train, num\_clases)

y\_test = to\_categorical(y\_test, num\_clases)

### **Normalization**

mean = np.mean(x\_train)

std = np.std(x\_train)

x\_train = (x\_train - mean) / (std+1e-7)

x\_test = (x\_test - mean) / (std+1e-7)

## Set de datos

## Mitad para entrenar, mitad para validar

(x\_train, x\_valid) = x\_train[5000:], x\_train[:5000]

(y\_train, y\_valid) = y\_train[5000:], y\_train[:5000]

print('x\_train shape', x\_train.shape)

print('train;', x\_train.shape[0])

print('val;', x\_valid.shape[0])

print('test;', x\_test.shape[0])

## Modelo convolucional

x\_train.shape[1:]

(32, 32, 3)

**# input\_shape=x\_train.shape[1:])) = (32,32,3)**

base\_filtros = 32

wR = 1e-4 **#Weight for regularizer**

**#regularzer l1 (ele1 o ele2)**

model = Sequential()

**## conv 1**

model.add(Conv2D(base\_filtros, (3,3), padding='same',

kernel\_regularizer=regularizers.l2(wR),

input\_shape=x\_train.shape[1:]))

model.add(Activation('relu'))

model.add(BatchNormalization())

**## conv 2**

model.add(Conv2D(base\_filtros, (3,3), padding='same',

kernel\_regularizer=regularizers.l2(wR)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.2))

**## conv 3**

model.add(Conv2D(2\*base\_filtros, (3,3), padding='same',

kernel\_regularizer=regularizers.l2(wR)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2)) **#20% para evitar overfit**

**## conv 4**

model.add(Conv2D(2\*base\_filtros, (3,3), padding='same',

kernel\_regularizer=regularizers.l2(wR)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.3))

**## conv 5**

model.add(Conv2D(4\*base\_filtros, (3,3), padding='same',

model.add(Activation('relu'))

model.add(BatchNormalization())

**## conv 6**

model.add(Conv2D(4\*base\_filtros, (3,3), padding='same',

kernel\_regularizer=regularizers.l2(wR)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.4))

**## Flatten DE MATRIZ A VECTOR**

model.add(Flatten())

model.add(Dense(num\_clases,activation='softmax'))

model.summary()

## Compilando

from tensorflow.keras import optimizers

model.compile(loss='categorical\_crossentropy',

#optimizer=optimizers.Adam(),

  optimizer = keras.optimizers.Adam(learning\_rate=0.001),

              metrics=['accuracy'])

## Callbacks

**checkPoint** = ModelCheckpoint('mejor\_modelo.hdf5',

verbose=1,

save\_best\_only=True,

monitor = 'val\_accuracy')

## Data augmentation, genera mas imágenes para entrenar

**datagen**= ImageDataGenerator(rotation\_range=15,

                  width\_shift\_range=0.1,

                  height\_shift\_range=0.1,

                  horizontal\_flip=True,

                  vertical\_flip=True)

## Entrenando nuestro modelo

## Batch size, cuando se clasifica muchas imágenes

## Si tengo pocas imágenes no se necesita batch\_size

## Se asigna batch\_size el numero de imágenes a procesar por paso

## Si batch\_size: + mas grande => + rápido pero + recursos

hist = model.fit(**datagen.flow**(x\_train, y\_train, batch\_size=128),

          callbacks=[checkPoint],

          steps\_per\_epoch=x\_train.shape[0] // 128,

          epochs=20,

          verbose=2,

shuffle=True, #??

          validation\_data=(x\_valid, y\_valid)

         )

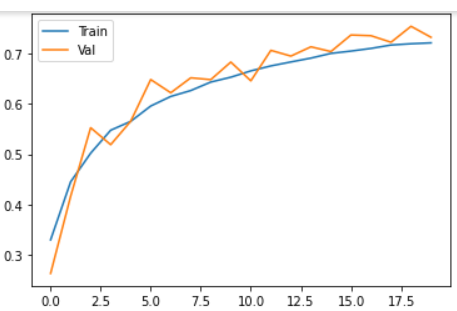
## Resultados

plt.plot(hist.history['accuracy'],label='Train')

plt.plot(hist.history['val\_accuracy'],label='Val')

plt.legend()

plt.show()

****

model2 = model

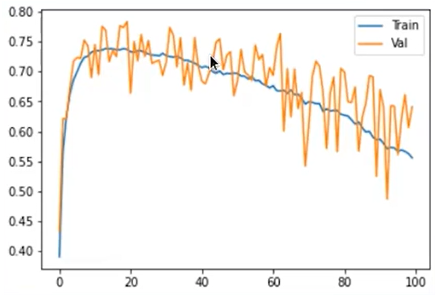
model2.load\_weights('mejor\_modelo.hdf5')

model2.evaluate(x\_test,y\_test, verbose=0)

[0.8466, 0.74059]

**MODIFICACIONES:**

**Con 100 epochs nos va reduciendo exactitud, y nos quedamos con 20.**

****

Modificando a:

* batch\_size 64 en vez de 128 (tarda mas por batch mas pequeño)
* learning rage 8e-5
* Dropout en algunas capas

Se logra accuracy de 80.46%