```
#########################  LIBRERIAS ##############################
#Dentro de TensorFlow tenemos la posibilidad de llamar al API de keras
#####################################################
import sys #Para poder movernos entre carpetas en nuestro SO
import os  #Para poder movernos entre carpetas en nuestro SO
#####################################################
#ImageDataGenerator: Preprocesar imagenes que le vamos a dar a nuestro algoritmo:
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
#Optimizador con el que vamos a entranar nuestro algoritmo:
from tensorflow.python.keras import optimizers
#Libreria que nos permite hacer redes neuronales secuenciales:
from tensorflow.python.keras.models import Sequential
#
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
#Nuestras capas donde vamos hacer nuestra convolución y nuestro max pulling
from tensorflow.python.keras.layers import  Convolution2D, MaxPooling2D
#Si hay una sesión de keras de background vamos a matarlo para poder empezar nuestro entrenamiento desde 0:
from tensorflow.python.keras import backend as K
#####################################################

K.clear_session()

############ Directorio de donde tenemos nuestras imagenes para entrenar #############
data_entrenamiento = './data/entrenamiento'
data_validacion = './data/validacion'

#############################   Parametros   ###################################
epocas=20  # N de veces que vamos a iterar sobre nuestro set de datos durante el entrenamiento.
longitud, altura = 150, 150 #El tamaño de nuestras imagenes para entrenar.
batch_size = 32 # N de imagenes que vamos a mandar a nuestra computadora procesar en cada paso.
pasos = 1000    # N de veces que se va a procesar la info en cada una de las epocas.
validation_steps = 300 #Pasos de validacion, para poder como de bien aprende nuestro algoritmo.
#Numero de filtros a aplicar en cada convolucion:
filtrosConv1 = 32  #Filtro en la convolucion 1 -> Nuestra imagen tendra una profundad de 32
filtrosConv2 = 64  #Filtro en la convolucion 2 -> Nuestra imagen tendra una profundad de 64
#Tamano de filtro que vamos a usar en cada convolucion:
tamano_filtro1 = (3, 3)  #Altura de 3 y longitud de 3
tamano_filtro2 = (2, 2)  #Altura de 2 y longitud de 2
#Tamano de filtro que vamos a usar en nuestro mas pooling
tamano_pool = (2, 2)
#Numero de clases que vamos a tener (3: gato, perro, gorila)
clases = 3
#Learning write: Que tan grandes van a ser los ajustes que hace nuestra red neuronal
lr = 0.0004
#####################################################

############# Pre-procesamiento de imagenes ##########################################

#Generador: como a preprocesar nuestra info. (Despues haremos la transformacion de nuestras imagenes)
entrenamiento_datagen = ImageDataGenerator(
    rescale=1. / 255,  #Reescalar valor pixel = (0-255) a un rango de (0-1) (+ eficiencia)
    shear_range=0.2,   # Genera imagenes inclinadas (Aceptar imagenes no siempre perfectas: verticalidad)
    zoom_range=0.2,    # Genera imagenes con +/- zoom (Aceptar imagenes no siempre perfectas)
    horizontal_flip=True) #Invierte imagen, para que aprenda a distinguir direccionalidad.

#Generador para set de datos de validacion: solo las reescalamos.
test_datagen = ImageDataGenerator(rescale=1. / 255)
#####################################################

############# Generar de imagenes   ##############################################

#Para entrenamiento:
entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,            # Directorio
    target_size=(altura, longitud), # Altura y longitud de imagenes (Ya definida)
    batch_size=batch_size,          # N de imag. a procesar en cada paso.
    class_mode='categorical')       # Clasifiacion categorica: Etiquetas(Gato, perro, gorila)

#Para validacion:
validacion_generador = test_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')
#####################################################

##################### CNN ###################################

#Creamos la red convolucional:
cnn = Sequential() #Sequential -> Nuestra red son varias capas apiladas entre ellas

# Añadir primera capa:

cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding ="same", input_shape=(longitud, altura, 3),
activation='relu'))

# Convolucion con 32 filtros de tamaño x, padding: filtro en las esquinas
# input-shape: le decimos que nuestras imagenes de entrada tienen x long y x altura
# funcion de activacion: relu

# Añadir segunda capa:
cnn.add(MaxPooling2D(pool_size=tamano_pool))
# Capa de Pooling + tam de pool (Despues de la 1 capa de convolucion -> filtro de max pooling de x tamaño)

#Añadir tercera capa (Capa de Convolucion) (Sin input-shape bc no es a la que le damos la input)
cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))

#Añadir cuarta capa (Capa de Max-Pooling)
cnn.add(MaxPooling2D(pool_size=tamano_pool))

############### Para empezar ya la clasificacion ##############################
cnn.add(Flatten()) #Esa imagen actual es profunda y pequeña -> Hacerla plana (1D con toda info de la CNN)
# Despues de aplanar nuestra info se la mandamos a una capa normal
# Esta capa de neuronas estará conectada a las de la capa anterior.
cnn.add(Dense(256, activation='relu')) #256 neuronas y de activacion relu
#A esta capa densa le apagamos el 50% de las neuronas
#Evita sobreajuste, si todas activas la CNN puede aprender un camino en especifico para clasificarlas cosas
# Si le decimos que de manera aleatoria en cada paso active solo el 50% de las 256, aprende caminos alternos
# para nueva info y se hace un modelo que clasifica mejor nueva info
cnn.add(Dropout(0.5))

#Otra capa densa:
cnn.add(Dense(clases, activation='softmax')) #ultima capa =num de clases para salida
#Activacion de softmax: nos ayuda a decir que probabilidad de que sea una cosa u otra la imagen input
#El valor que tiene el porcentaje mas alto es la clasificación correcta


#Parametros que vamos a usar para optimizar nuestro algoritmo
cnn.compile(loss='categorical_crossentropy',  # funcion de perdida: Que nuestro algo vea que tan bn o mal va.
            optimizer=optimizers.Adam(lr=lr), # optimizador: Adam con lr
            metrics=['accuracy'])             # metrica con la que optimizamos: % de que bien aprende la CNN

#Intenta mejorar el accuracy: % de imagenes que clasifica bn

#####################################################################################################

#############

#Alimentar la CNN con las imagenes preprocesadas, corriendolo 1000 veces por 20 epocas:
cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,  #Numero de pasos/epoca
    epochs=epocas,          #Numero de epocas
    validation_data=validacion_generador, #Nuestras imagenes de validacion
    validation_steps=validation_steps)    #N de pasos de validacion va a correr despues de cada epoca

#Guardamos modelo en un archivo para no tener que entrenarlo cada vez que queramos hacer una predicción
target_dir = './modelo/'

#Si no existe la carpeta llamada modelo -> La generamos
if not os.path.exists(target_dir):
  os.mkdir(target_dir)

#Guardamos el modelo:
cnn.save('./modelo/modelo.h5')

#Guardamos los pesos.
cnn.save_weights('./modelo/pesos.h5')
```