```
import pandas as pd
import numpy as np #Nos ayuda a trabajar con arrays.
import matplotlib.pyplot as plt # Graficar.
%matplotlib inline
import seaborn as sns # Graficas
sns.set(style="whitegrid")
import os #Acceder a archivos del OS #listdir # cd
import glob as gb
import tensorflow as tf
import keras
import cv2 #Procesamiento de imagenes.
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img,img_to_array # Pasar la imagen a un array.
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical
```

Pandas (pd): Pandas es una biblioteca de Python que proporciona estructuras de datos y herramientas de análisis de datos. Se utiliza para la manipulación y análisis de datos tabulares, como hojas de cálculo, bases de datos SQL o archivos CSV. Pandas ofrece DataFrames, que son estructuras de datos bidimensionales que facilitan la limpieza, manipulación y análisis de datos.

Numpy (np): NumPy es una biblioteca fundamental para la computación científica en Python. Proporciona soporte para matrices y arreglos multidimensionales, junto con una amplia gama de funciones matemáticas para operar en estos arreglos. Se utiliza para el cálculo numérico y la manipulación eficiente de datos numéricos.

Matplotlib.pyplot (plt): Matplotlib es una biblioteca de trazado en Python que se utiliza para crear gráficos y visualizaciones. Matplotlib.pyplot es un módulo específico para trazar gráficos. Permite crear gráficos de barras, gráficos de líneas, diagramas de dispersión y muchas otras representaciones visuales de datos.

Seaborn (sns): Seaborn es otra biblioteca de Python para la visualización de datos que se basa en Matplotlib. Ofrece una interfaz de alto nivel para crear gráficos estadísticos informativos y atractivos. Seaborn simplifica la creación de gráficos de barras, gráficos de violín, mapas de calor y otros tipos de gráficos estadísticos.

Os: El módulo os proporciona funciones para interactuar con el sistema operativo. Se utiliza para realizar operaciones relacionadas con el sistema de archivos, como listar archivos en un directorio, cambiar de directorio, crear directorios, y otras tareas relacionadas con la gestión de archivos y directorios.

Glob (gb): El módulo glob se utiliza para buscar archivos que coincidan con un patrón en un directorio. Facilita la búsqueda y el procesamiento de múltiples archivos en un directorio en función de su nombre o extensión.

TensorFlow (tf): TensorFlow es una biblioteca de código abierto desarrollada por Google para la creación y entrenamiento de modelos de aprendizaje automático, especialmente redes neuronales. Se utiliza en una amplia variedad de aplicaciones de aprendizaje automático y aprendizaje profundo.

Keras: Keras es una interfaz de alto nivel que se ejecuta sobre bibliotecas de aprendizaje profundo subyacentes como TensorFlow. Facilita la creación y el entrenamiento de modelos de aprendizaje profundo de una manera más sencilla y accesible. Keras es ampliamente utilizado para el desarrollo de modelos de redes neuronales.

OpenCV (cv2): OpenCV (Open Source Computer Vision Library) es una biblioteca de código abierto que se utiliza para el procesamiento de imágenes y visión por computadora. Ofrece una amplia variedad de funciones y herramientas para manipular imágenes, detectar objetos, realizar seguimiento de objetos, y aplicar filtros y transformaciones a imágenes

TensorFlow.keras.preprocessing.image: Este módulo específico de TensorFlow se utiliza para la preparación y procesamiento de imágenes antes de alimentarlas a modelos de aprendizaje profundo. Proporciona herramientas para cargar imágenes, convertirlas en arreglos numéricos y realizar aumentos de datos, entre otras tareas relacionadas con el procesamiento de imágenes.

TensorFlow.keras.utils: Este módulo proporciona utilidades adicionales para trabajar con modelos de TensorFlow, como la conversión de etiquetas en un formato adecuado para el entrenamiento de modelos.

```
code = {"Benign":0 ,"Early":1,"Pre":2,"Pro":3}

def getcode(n):
    for x , y in code.items() :
        if n == y :
            return x
```

s=224 #224 pixeles

Esta variable se utilizara mas adelante en el codigo para darle un nuevo tamaño a nuestras imágenes, en este caso 224*224 pixeles.

code = {"Benign": 0, "Early": 1, "Pre": 2, "Pro": 3}: En esta línea, se define un diccionario llamado code que asocia etiquetas ("Benign", "Early", "Pre", "Pro") con valores numéricos (0, 1, 2, 3). Este diccionario se utilizará para mapear valores numéricos a etiquetas.

def getcode(n):: Aquí, se define una función llamada getcode que toma un argumento n.

for x, y in code.items():: Este bucle for itera a través de los elementos (pares clave-valor) del diccionario code. x toma el valor de las claves (etiquetas) y y toma el valor numérico asociado a esas claves.

if n == y:: Se verifica si el valor n pasado como argumento a la función es igual al valor numérico y que se está iterando en el bucle.

return x: Si n es igual a y, la función devuelve la etiqueta x correspondiente al valor numérico n. Esto significa que la función mapea el valor numérico a su etiqueta correspondiente según el diccionario code.

```
X_train = [] # Almacena nuestras imagenes.
y_train = [] # Almacena nuestras etiquetas.
for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Benign')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Benign',img),1)
    image_array = cv2.resize(image , (s,s))
    X_train.append(list(image_array))
    y_train.append(code['Benign'])
```

```
100%|██████████| 514/514 [00:02<00:00, 246.61it/s]
```

```
for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Early')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Early',img),1)
    image_array = cv2.resize(image , (s,s))
    X_train.append(list(image_array))
    y_train.append(code['Early'])
```

Aquí está el desglose línea por línea:

X_train = []: Se crea una lista vacía llamada X_train que se utilizará para almacenar las imágenes.

y_train = []: Se crea otra lista vacía llamada y_train que se utilizará para almacenar las etiquetas correspondientes a las imágenes.

for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Benign')):: Se inicia un bucle for que itera a través de los archivos en el directorio '/content/drive/MyDrive/Original/Benign'. tqdm es una función que se utiliza para mostrar una barra de progreso durante la iteración.

image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Benign', img), 1): Lee la imagen actual (identificada como img) en el directorio '/content/drive/MyDrive/Original/Benign' utilizando la biblioteca OpenCV (cv2). La imagen se almacena en la variable image.

image_array = cv2.resize(image, (s, s)): Redimensiona la imagen a un tamaño específico (s, s) utilizando la función resize de OpenCV y almacena la imagen redimensionada en la variable image_array.

X_train.append(list(image_array)): Agrega la imagen redimensionada image_array a la lista X_train. La imagen se convierte en una lista antes de agregarla a la lista X_train.

y_train.append(code['Benign']): Agrega la etiqueta correspondiente al directorio 'Benign' (que es 0 según el diccionario code) a la lista y_train. Esto asigna la etiqueta "Benign" a la imagen actual.

Luego, el mismo proceso se repite para las imágenes en el directorio '/content/drive/MyDrive/Original/Early'. Las imágenes se leen, redimensionan y se agrega la etiqueta "Early" (que es 1 según el diccionario code) a la lista y_train.

En resumen, este fragmento de código carga imágenes desde dos directorios diferentes ('Benign' y 'Early'), las redimensiona y las almacena junto con sus etiquetas correspondientes en las listas X_train e y_train. Esto es útil para preparar datos de entrenamiento para un modelo de clasificación de imágenes.

```
X_train = [] # Almacena nuestras imagenes.
y_train = [] # Almacena nuestras etiquetas.
for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Benign')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Benign',img),1)
    image_array = cv2.resize(image , (s,s))
    X_train.append(list(image_array))
    y_train.append(code['Benign'])
```

```
100%|██████████| 514/514 [00:02<00:00, 246.61it/s]
```

```
for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Early')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Early',img),1)
    image_array = cv2.resize(image , (s,s))
    X_train.append(list(image_array))
    y_train.append(code['Early'])
```

```
  1%|          | 9/985 [00:03<06:42,  2.42it/s]
```

```
for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Pre')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Pre',img),1)
    image_array = cv2.resize(image , (s,s))
    X_train.append(list(image_array))
    y_train.append(code['Pre'])
```

X_train = []: Se crea una lista vacía llamada X_train que se utilizará para almacenar las imágenes.

y_train = []: Se crea otra lista vacía llamada y_train que se utilizará para almacenar las etiquetas correspondientes a las imágenes.

El código que sigue es un conjunto de bucles for, uno para cada uno de los cuatro directorios ("Benign", "Early", "Pre" y "Pro"). Cada bucle itera a través de los archivos en un directorio específico.

for img in tqdm(os.listdir('/content/drive/MyDrive/Original/Benign')):: Inicia el primer bucle, que itera a través de los archivos en el directorio '/content/drive/MyDrive/Original/Benign'. Se utiliza tqdm para mostrar una barra de progreso durante la iteración.

image = cv2.imread(os.path.join('/content/drive/MyDrive/Original/Benign', img), 1): Lee la imagen actual (identificada como img) en el directorio 'Benign' utilizando OpenCV (cv2). La imagen se almacena en la variable image.

image_array = cv2.resize(image, (s, s)): Redimensiona la imagen a un tamaño específico (s, s) utilizando la función resize de OpenCV y almacena la imagen redimensionada en la variable image_array.

X_train.append(list(image_array)): Agrega la imagen redimensionada image_array a la lista X_train. La imagen se convierte en una lista antes de agregarla a la lista X_train.

y_train.append(code['Benign']): Agrega la etiqueta correspondiente al directorio 'Benign' (que es 0 según el diccionario code) a la lista y_train. Esto asigna la etiqueta "Benign" a la imagen actual.

Los pasos 4-8 se repiten para los otros tres directorios ("Early", "Pre" y "Pro"), cargando las imágenes, redimensionándolas y agregándolas a las listas X_train e y_train con las etiquetas apropiadas ('Early', 'Pre' y 'Pro' respectivamente) según el diccionario code.

En resumen, este código carga imágenes desde cuatro directorios diferentes, redimensiona las imágenes y las almacena junto con sus etiquetas en las listas X_train e y_train. Esto se utiliza para preparar datos de entrenamiento para un modelo de clasificación de imágenes que tiene cuatro categorías diferentes.

```
plt.figure(figsize=(20,20))
for n , i in enumerate(list(np.random.randint(0,len(X_train),36))) :
    plt.subplot(6,6,n+1)
    plt.imshow(X_train[i])
    plt.axis('off')
    plt.title(getcode(y_train[i]))
# la lista x almacena las imagenes
# La lista Y alamcena los valores numericos. 1, 2, 3, 4
```

plt.figure(figsize=(20, 20)): Se crea una figura de Matplotlib con un tamaño de 20x20 pulgadas. Esta figura contendrá múltiples subgráficos para mostrar las imágenes.

for n, i in enumerate(list(np.random.randint(0, len(X_train), 36))): : Esto inicia un bucle for que recorre 36 elementos aleatorios de la lista X_train y sus respectivas etiquetas.

enumerate se utiliza para obtener tanto el índice n como el valor i en cada iteración. i representa un índice aleatorio de X_train.

np.random.randint(0, len(X_train), 36) genera una lista de 36 números enteros aleatorios que se utilizan para seleccionar índices aleatorios en X_train.

plt.subplot(6, 6, n+1): Se crea un subgráfico dentro de la figura con una cuadrícula de 6 filas y 6 columnas. n+1 se utiliza para especificar la posición del subgráfico actual en la cuadrícula.

plt.imshow(X_train[i]): Se muestra la imagen correspondiente al índice i de la lista X_train en el subgráfico actual utilizando plt.imshow. Esto muestra la imagen en el subgráfico.

plt.axis('off'): Se desactiva la visualización de ejes (coordenadas) en el subgráfico. Esto significa que no se mostrarán los ejes X e Y en la imagen.

plt.title(getcode(y_train[i])): Se establece un título para el subgráfico actual utilizando la función getcode para obtener la etiqueta correspondiente al valor numérico de y_train[i]. El título muestra la etiqueta asociada a la imagen.

La explicación final en los comentarios resume el propósito del código:

"La lista X almacena las imágenes": La lista X_train contiene las imágenes que se mostrarán en los subgráficos.

"La lista Y almacena los valores numéricos. 1, 2, 3, 4": La lista y_train contiene los valores numéricos correspondientes a las etiquetas de las imágenes. Estos valores numéricos se utilizan para buscar la

etiqueta correcta en el diccionario code utilizando la función getcode antes de mostrarla como título en el subgráfico.

```
X_test = []
y_test = []
for img in tqdm(os.listdir('/content/drive/MyDrive/Segmented/Benign')):
    image = cv2.imread(os.path.join('/content/drive/MyDrive/Segmented/Benign',img),1)
    image_array = cv2.resize(image , (s,s))
    X_test.append(list(image_array))
    y_test.append(code['Benign'])
```

Este fragmento de código es muy similar al que se explicó anteriormente, pero en este caso, carga imágenes desde los directorios correspondientes a las imágenes segmentadas ('Benign', 'Early', 'Pre', 'Pro') y las almacena junto con sus etiquetas en las listas X_test e y_test. Aquí está el desglose línea por línea:

X_test = []: Se crea una lista vacía llamada X_test que se utilizará para almacenar las imágenes de prueba.

y_test = []: Se crea otra lista vacía llamada y_test que se utilizará para almacenar las etiquetas correspondientes a las imágenes de prueba.

A continuación, el código es un conjunto de bucles for, uno para cada uno de los cuatro directorios ("Benign", "Early", "Pre" y "Pro"). Cada bucle itera a través de los archivos en un directorio específico.

for img in tqdm(os.listdir('/content/drive/MyDrive/Segmented/Benign')):: Inicia el primer bucle, que itera a través de los archivos en el directorio '/content/drive/MyDrive/Segmented/Benign'. Se utiliza tqdm para mostrar una barra de progreso durante la iteración.

image = cv2.imread(os.path.join('/content/drive/MyDrive/Segmented/Benign', img), 1): Lee la imagen actual (identificada como img) en el directorio 'Benign' desde la carpeta "segmented" utilizando OpenCV (cv2). La imagen se almacena en la variable image.

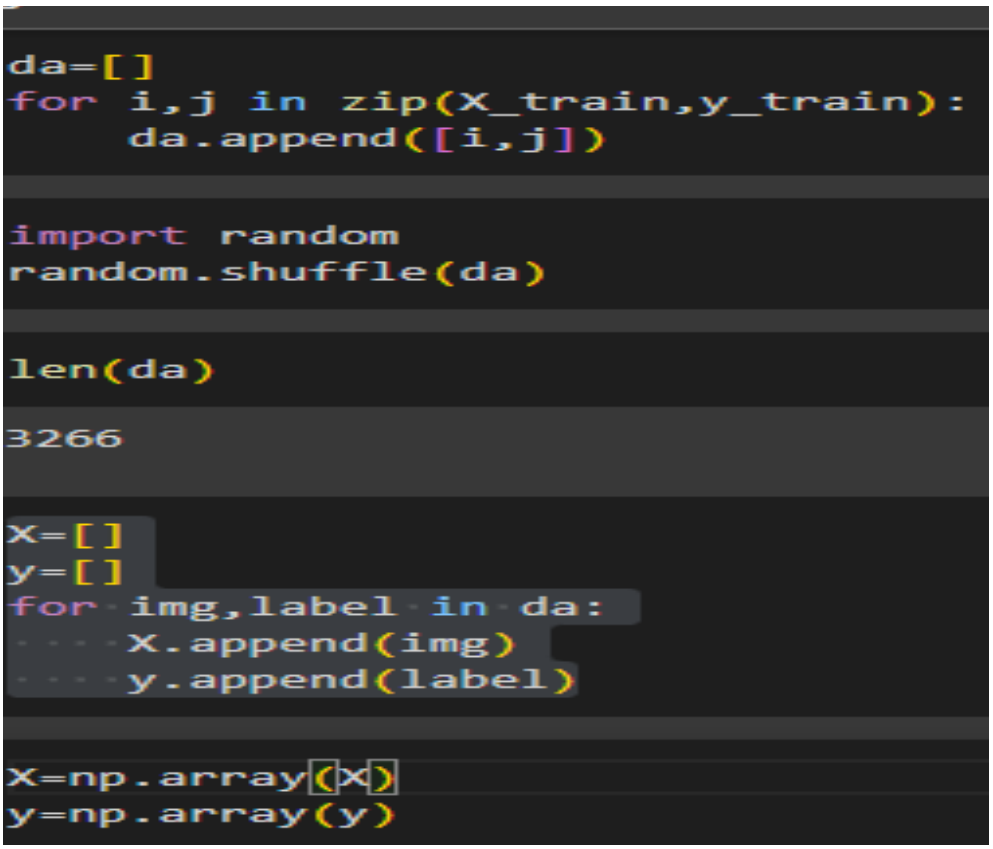
image_array = cv2.resize(image, (s, s)): Redimensiona la imagen a un tamaño específico (s, s) utilizando la función resize de OpenCV y almacena la imagen redimensionada en la variable image_array.

X_test.append(list(image_array)): Agrega la imagen redimensionada image_array a la lista X_test. La imagen se convierte en una lista antes de agregarla a la lista X_test.

y_test.append(code['Benign']): Agrega la etiqueta correspondiente al directorio 'Benign' (que es 0 según el diccionario code) a la lista y_test. Esto asigna la etiqueta "Benign" a la imagen actual.

Los pasos 4-8 se repiten para los otros tres directorios ("Early", "Pre" y "Pro"), cargando las imágenes segmentadas de prueba, redimensionándolas y agregándolas a las listas X_test e y_test con las etiquetas apropiadas ('Early', 'Pre' y 'Pro' respectivamente) según el diccionario code.

En resumen, este código carga imágenes segmentadas de prueba desde cuatro directorios diferentes, redimensiona las imágenes y las almacena junto con sus etiquetas en las listas X_test e y_test. Esto se utiliza para preparar datos de prueba para un modelo de clasificación de imágenes que tiene cuatro categorías diferentes, al igual que en el conjunto de entrenamiento.



```
da=[]
for i,j in zip(X_train,y_train):
    da.append([i,j])

import random
random.shuffle(da)

len(da)

3266

X=[]
y=[]
for img,label in da:
    X.append(img)
    y.append(label)

X=np.array(X)
y=np.array(y)
```

da=[]: Se crea una lista vacía llamada da que se utilizará para almacenar pares de imágenes y etiquetas. Cada par será una lista que contenga una imagen y su etiqueta correspondiente.

for i, j in zip(X_train, y_train):: Se inicia un bucle for que itera simultáneamente a través de dos listas, X_train (que contiene las imágenes de entrenamiento) y y_train (que contiene las etiquetas de entrenamiento).

da.append([i, j]): En cada iteración, se agrega un par compuesto por la imagen i y su etiqueta j a la lista da. Esto crea una lista con imágenes y etiquetas emparejadas.

import random: Se importa el módulo random de Python, que se utilizará para aleatorizar el orden de los pares de imágenes y etiquetas en da.

random.shuffle(da): Se utiliza la función shuffle del módulo random para aleatorizar el orden de los elementos en la lista da. Esto garantiza que los datos de entrenamiento no estén en un orden específico, lo que puede ser útil para evitar sesgos en el entrenamiento del modelo.

X=[] y y=[]: Se crean dos listas vacías llamadas X e y que se utilizarán para almacenar las imágenes y etiquetas reorganizadas.

for img, label in da:: Se inicia un bucle for que itera a través de los pares de imágenes y etiquetas aleatorizados en da.

X.append(img): En cada iteración, se agrega la imagen img a la lista X. Esto reorganiza las imágenes de entrenamiento de manera aleatoria.

y.append(label): Se agrega la etiqueta label correspondiente a la imagen en la misma iteración a la lista y. Esto asegura que las etiquetas estén en el mismo orden que las imágenes reorganizadas.

X = np.array(X) y y = np.array(y): Se convierten las listas X e y en arreglos de NumPy utilizando np.array(). Esto es útil para preparar los datos en un formato adecuado para ser alimentados al modelo VGG19, que generalmente requiere que los datos de entrada sean arreglos NumPy.

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(X,y,train_size=0.8,shuffle=True)
print(xtest.shape)
print(xtrain.shape)
```

```
(654, 224, 224, 3)
(2612, 224, 224, 3)
```

from sklearn.model_selection import train_test_split: Importa la función train_test_split de la biblioteca scikit-learn. Esta función se utiliza para dividir un conjunto de datos en conjuntos de entrenamiento y prueba de manera aleatoria y estratificada, lo que es útil en problemas de aprendizaje supervisado.

xtrain, xtest, ytrain, ytest = train_test_split(X, y, train_size=0.8, shuffle=True): Utiliza la función train_test_split para dividir los datos. Los argumentos son los siguientes:

X y y: Los arreglos de datos de entrada y etiquetas, respectivamente. Estos son los datos que se dividirán en conjuntos de entrenamiento y prueba.

train_size=0.8: Esto indica que se quiere que el 80% de los datos se utilicen para el conjunto de entrenamiento y el 20% restante se utilice para el conjunto de prueba.

shuffle=True: Esto indica que los datos se deben reorganizar aleatoriamente antes de dividirlos. Esto es útil para garantizar que los datos se distribuyan de manera aleatoria entre los conjuntos de entrenamiento y prueba.

print(xtest.shape): Imprime la forma (tamaño) del conjunto de prueba xtest. En el resultado se muestra (654, 224, 224, 3), lo que significa que hay 654 muestras en el conjunto de prueba, cada una con una forma de (224, 224, 3). Esto indica que cada muestra es una imagen de 224x224 píxeles con 3 canales de color (RGB).

print(xtrain.shape): Imprime la forma (tamaño) del conjunto de entrenamiento xtrain. En el resultado se muestra (2612, 224, 224, 3), lo que significa que hay 2612 muestras en el conjunto de entrenamiento, cada una con una forma de (224, 224, 3). Al igual que en el conjunto de prueba, esto indica que cada muestra es una imagen de 224x224 píxeles con 3 canales de color (RGB).

```
KerasModel = keras.models.Sequential([
        keras.layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(100,100,3)),
        keras.layers.Conv2D(150,kernel_size=(3,3),activation='relu'),
        keras.layers.MaxPool2D(4,4),
        keras.layers.Conv2D(120,kernel_size=(3,3),activation='relu'),
        keras.layers.Conv2D(50,kernel_size=(3,3),activation='relu'),
        keras.layers.MaxPool2D(4,4),
        keras.layers.Flatten() ,
        keras.layers.Dense(256,activation='relu') ,
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(rate=0.3) ,
        keras.layers.Dense(128,activation='relu') ,
        keras.layers.BatchNormalization(),
        keras.layers.Dense(100,activation='relu') ,
        keras.layers.BatchNormalization(),
        keras.layers.Dense(50,activation='relu') ,
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(rate=0.5) ,
        keras.layers.Dense(4,activation='softmax') ,
        ])
```

```
m_model=tf.keras.applications.vgg19.VGG19()
model1=tf.keras.models.Sequential()
for layer in m_model.layers[:-1]:
    model1.add(layer)
for layer in model1.layers:
    layer.trainable=False
model1.add(tf.keras.layers.Dense(4,activation=tf.nn.softmax))
```

Parte personalizada con 18 capas:

KerasModel = keras.models.Sequential([: Se crea un modelo secuencial de Keras llamado KerasModel.

keras.layers.Conv2D(200, kernel_size=(3,3), activation='relu', input_shape=(100,100,3)),: Se agrega una capa de convolución 2D con 200 filtros, un tamaño de kernel de 3x3, función de activación ReLU y se especifica la forma de entrada (100,100,3) que corresponde a imágenes de 100x100 píxeles con 3 canales de color (RGB).

keras.layers.Conv2D(150, kernel_size=(3,3), activation='relu'),: Se agrega otra capa de convolución 2D con 150 filtros y otros parámetros similares.

keras.layers.MaxPool2D(4,4),: Se agrega una capa de max-pooling 2D con un tamaño de ventana de 4x4. Esta capa reduce las dimensiones espaciales de la salida.

Se repiten las capas de convolución y max-pooling con diferentes configuraciones.

keras.layers.Flatten(),: Se agrega una capa de aplanamiento que convierte la salida de las capas anteriores en un vector unidimensional.

keras.layers.Dense(256, activation='relu'),: Se agrega una capa densamente conectada con 256 neuronas y función de activación ReLU.

keras.layers.BatchNormalization(),: Se agrega una capa de normalización por lotes, que ayuda a estabilizar y acelerar el entrenamiento de la red.

keras.layers.Dropout(rate=0.3),: Se agrega una capa de dropout con una tasa de abandono del 30%. Esto ayuda a prevenir el sobreajuste.

Se repiten capas densas, normalización por lotes y capas de dropout.

keras.layers.Dense(4, activation='softmax'),: Se agrega una capa densa de salida con 4 neuronas (una para cada clase) y función de activación softmax, que se utiliza en problemas de clasificación para obtener probabilidades de pertenencia a cada clase.

Parte basada en VGG19 preentrenado:

m_model = tf.keras.applications.vgg19.VGG19(): Se crea un modelo VGG19 preentrenado utilizando la función tf.keras.applications.vgg19.VGG19().

model1 = tf.keras.models.Sequential(): Se crea un nuevo modelo secuencial llamado model1.

Se recorren todas las capas del modelo preentrenado VGG19, excepto la capa de salida, y se agregan a model1. Esto se hace para transferir el conocimiento aprendido por VGG19 a un nuevo modelo.

for layer in model1.layers:: Se establece que todas las capas en model1 no serán entrenables, ya que se quiere mantener los pesos preentrenados.

model1.add(tf.keras.layers.Dense(4, activation=tf.nn.softmax)): Finalmente, se agrega una capa densa de salida con 4 neuronas y función de activación softmax. Esto se hace para adaptar el modelo preentrenado a un problema de clasificación con 4 clases.

En resumen, este código combina una red neuronal personalizada con una capa de salida de VGG19 preentrenado, creando un modelo híbrido. La parte personalizada tiene 18 capas y la parte de VGG19 se utiliza para transferir conocimiento desde un modelo previamente entrenado.

```
model1.compile(optimizer ='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

optimizer='adam': El optimizador es el algoritmo que se utilizará para ajustar los pesos de la red neuronal durante el proceso de entrenamiento. En este caso, se utiliza el optimizador 'adam', que es una opción común para problemas de clasificación. El optimizador 'adam' se adapta de manera eficiente a diferentes tipos de datos y generalmente es una buena elección para comenzar.

loss='sparse_categorical_crossentropy': La función de pérdida, o "loss function", se utiliza para medir cuán bien el modelo está realizando la tarea de clasificación. En este caso, se utiliza 'sparse_categorical_crossentropy' como la función de pérdida. Esta función de pérdida es adecuada cuando se tiene un problema de clasificación con etiquetas enteras (como 0, 1, 2, 3) en lugar de codificación one-hot (etiquetas binarias con un 1 en la clase correcta y 0 en las demás). La 'sparse' en 'sparse_categorical_crossentropy' se refiere a este tipo de etiquetas.

metrics=['accuracy']: Aquí se especifica una métrica que se utilizará para evaluar el rendimiento del modelo durante y después del entrenamiento. En este caso, se utiliza 'accuracy' (exactitud), que mide la proporción de predicciones correctas con respecto al total de muestras. Es una métrica común para problemas de clasificación y ayuda a evaluar qué tan bien el modelo está haciendo las predicciones correctas en el conjunto de datos de prueba.

```
history=model1.fit(xtrain,ytrain,batch_size=128,
                   verbose=1,
                   validation_data=(xtest,ytest),epochs=5)

Epoch 1/5
21/21 [==============================] - 2623s 126s/step - loss: 0.8158 - accuracy: 0.7144 - val_loss: 0.3427 - val_accuracy: 0.8853
Epoch 2/5
21/21 [==============================] - 2516s 121s/step - loss: 0.2627 - accuracy: 0.9119 - val_loss: 0.2066 - val_accuracy: 0.9297
Epoch 3/5
21/21 [==============================] - 2476s 119s/step - loss: 0.1726 - accuracy: 0.9491 - val_loss: 0.1705 - val_accuracy: 0.9419
Epoch 4/5
21/21 [==============================] - 2446s 118s/step - loss: 0.1421 - accuracy: 0.9571 - val_loss: 0.1508 - val_accuracy: 0.9495
Epoch 5/5
21/21 [==============================] - 2461s 118s/step - loss: 0.1155 - accuracy: 0.9686 - val_loss: 0.1431 - val_accuracy: 0.9541
```

history = model1.fit(xtrain, ytrain, batch_size=128, verbose=1, validation_data=(xtest, ytest), epochs=5): Esta línea de código inicia el proceso de entrenamiento del modelo. Aquí están los detalles de los argumentos utilizados:

xtrain, ytrain: Estos son los datos de entrenamiento. xtrain son las imágenes de entrenamiento y ytrain son las etiquetas correspondientes.

batch_size=128: Define el tamaño del lote (batch size). En cada paso de entrenamiento, se utilizan 128 muestras para ajustar los pesos del modelo. El tamaño del lote es una configuración que afecta la velocidad de entrenamiento y la memoria requerida.

verbose=1: Esto controla la cantidad de información que se muestra durante el entrenamiento. Un valor de 1 significa que se mostrará información detallada sobre el progreso del entrenamiento en la pantalla a medida que avanza.

validation_data=(xtest, ytest): Especifica un conjunto de datos de validación que se utiliza para evaluar el rendimiento del modelo después de cada época (epoch) de entrenamiento. xtest son las imágenes de prueba y ytest son las etiquetas de prueba.

epochs=5: Indica la cantidad de épocas de entrenamiento. Una época es un ciclo completo a través de todo el conjunto de datos de entrenamiento. En este caso, se entrenará durante 5 épocas.

history: La variable history guarda información sobre el proceso de entrenamiento, como las pérdidas y las métricas en cada época. Puedes utilizar esta información para analizar el rendimiento del modelo y visualizar cómo cambian las métricas a lo largo del entrenamiento.

Después de ejecutar esta línea, el modelo model1 se entrenará en los datos de entrenamiento (xtrain y ytrain) durante 5 épocas. Durante el entrenamiento, el modelo ajustará sus pesos para minimizar la función de pérdida especificada en la fase de compilación. La información detallada sobre el progreso del entrenamiento se mostrará en la pantalla debido al argumento verbose=1.

Luego de eso nos encargamos de guardar el modelo en un archivo h5 para su posterior utilizacion.