<u>Procesamiento</u> y entrenamiento de modelos de aprendizaje profundo para la clasificación de imágenes médicas. A continuación, se explicarán en detalle todos los pasos del código:

Importación de Bibliotecas:

import pandas as pd: Importa la biblioteca Pandas bajo el alias pd, que se utiliza comúnmente para la manipulación y análisis de datos.

import numpy as np: Importa la biblioteca NumPy bajo el alias np, que se utiliza para trabajar con matrices y operaciones numéricas eficientes.

import matplotlib.pyplot as plt: Importa la biblioteca Matplotlib bajo el alias plt, que se utiliza para graficar datos.

import seaborn as sns: Importa la biblioteca Seaborn bajo el alias sns, que se utiliza para crear gráficos estadísticos más atractivos.

import os: Importa la biblioteca OS, que permite acceder y manipular archivos y directorios del sistema operativo.

import glob as gb: Importa la biblioteca Glob bajo el alias gb, que se utiliza para buscar archivos que coincidan con un patrón en un directorio.

import tensorflow as tf: Importa la biblioteca TensorFlow, que se utiliza para crear y entrenar modelos de aprendizaje profundo.

import keras: Importa la biblioteca Keras, una interfaz de alto nivel para construir y entrenar modelos de aprendizaje profundo.

import cv2: Importa la biblioteca OpenCV, que se utiliza para el procesamiento de imágenes.

Definición de un Diccionario de Códigos:

code = {"Benign":0 ,"Early":1,"Pre":2,"Pro":3}: Se crea un diccionario llamado code que mapea nombres de clases a valores numéricos. Esto es común en problemas de clasificación donde las etiquetas se representan numéricamente.

Definición de una Función para Obtener el Código de Clase:

def getcode(n): Se define una función llamada getcode que toma un valor numérico n y devuelve el nombre de la clase correspondiente utilizando el diccionario code.

Configuración de Tamaño de Imagen:

s = 224: Se establece el tamaño de las imágenes a 224x224 píxeles.

Carga de Imágenes de Entrenamiento:

Se utiliza un bucle for para cargar imágenes de diferentes directorios de entrenamiento (/content/drive/MyDrive/Original/...) y almacenarlas en las listas X_train y y_train. Cada imagen se redimensiona al tamaño especificado en s.

Visualización de Muestras de Entrenamiento:

Se crea una figura de Matplotlib para visualizar 36 muestras de imágenes de entrenamiento. Las imágenes se seleccionan aleatoriamente de X_train, y sus etiquetas se muestran en el título de cada imagen.

Carga de Imágenes de Prueba:

Se utiliza un bucle for similar al paso 5 para cargar imágenes de diferentes directorios de prueba (/content/drive/MyDrive/Segmented/...) y almacenarlas en las listas X_test y y_test.

Visualización de Muestras de Prueba:

Se crea otra figura de Matplotlib para visualizar 36 muestras de imágenes de prueba de manera similar al paso 6.

Mezcla de Datos de Entrenamiento:

Se crea una lista da que contiene pares de imágenes y etiquetas. Luego, los datos en da se mezclan aleatoriamente utilizando random.shuffle(da).

Creación de Arreglos Numéricos:

Se crean dos arreglos NumPy, X y y, que contienen imágenes y etiquetas, respectivamente. Esto se hace para facilitar el uso de estos datos en el entrenamiento del modelo.

División de Datos en Entrenamiento y Prueba:

Se utiliza la función train_test_split de Scikit-Learn para dividir los datos en conjuntos de entrenamiento y prueba. Se selecciona el 80% de los datos para el entrenamiento. Se imprime el tamaño de los conjuntos resultantes.

Definición de un Modelo Personalizado:

Se define un modelo de red neuronal utilizando Keras. El modelo incluye varias capas de convolución, max-pooling, capas densas y capas de dropout. Este modelo es personalizado y se utilizará en combinación con un modelo VGG19 preentrenado.

Carga de un Modelo VGG19 Preentrenado:

Se carga un modelo VGG19 preentrenado de TensorFlow. Este modelo ya ha sido entrenado en un gran conjunto de datos y se utilizará como base para transferir el conocimiento a un nuevo modelo.

Creación de un Modelo Híbrido:

Se crea un nuevo modelo secuencial llamado model1, que combina el modelo VGG19 preentrenado y un clasificador personalizado. Se congelan las capas del modelo VGG19 para mantener los pesos preentrenados.

Compilación del Modelo:

Se compila el modelo utilizando el optimizador 'Adam', la función de pérdida 'sparse_categorical_crossentropy' (adecuada para etiquetas numéricas enteras) y la métrica 'accuracy' (exactitud).

Entrenamiento del Modelo:

El modelo se entrena utilizando los datos de entrenamiento xtrain e ytrain. Se utiliza un tamaño de lote de 128, y se realizan 5 épocas de entrenamiento. La información sobre el progreso del entrenamiento se muestra en la pantalla.

Predicción en Datos de Prueba:

Se realiza una predicción en los datos de prueba xtest, y las predicciones se almacenan en y_pred.

Visualización de Precisión durante el Entrenamiento:

Se crea un gráfico que muestra cómo cambia la precisión del modelo en el conjunto de entrenamiento y prueba a lo largo de las épocas de entrenamiento.

Guardado del Modelo:

El modelo entrenado se guarda en un archivo llamado 'modelokaggle1.h5' para su posterior uso.

Carga y Predicción con un Modelo Guardado:

Se carga el modelo guardado y se realiza una predicción en una imagen de prueba.

Mapeo de Clases y Visualización de la Predicción:

Se mapea la clase predicha a una etiqueta de texto utilizando el diccionario code, y se muestra el resultado de la predicción.

Uso de Listas en Python:

Al final del código, se crea una lista llamada lista con algunos elementos. Se muestra cómo acceder a elementos específicos de la lista utilizando índices y rebanadas ([:-1] se utiliza para excluir el último elemento).

Este código realiza una variedad de tareas, desde la carga y procesamiento de imágenes hasta la creación y entrenamiento de un modelo de clasificación. También incluye la visualización de resultados y la capacidad de guardar y cargar modelos para su uso posterior. El enfoque de transferencia de aprendizaje se utiliza para aprovechar un modelo preentrenado y ajustarlo a un problema específico.