Entrega 2: Modelo de detección de tumores cerebrales

Profesores:

Julian David Arias Londoño

Raúl Ramos Pollán

Integrantes:

Kadyha Paz Gutierrez

Sebastián Londoño Tobón

Contactos:

kadyha.paz@udea.edu.co

sebastian.londono9@udea.edu.co

Universidad de Antioquia

Facultad de ingeniería

2021

Descripción de la estructura de los notebooks entregados

Se entregan 3 notebooks que son llamados:

01 - Exploración de datos y entrenamiento inicial.ipynb

02 - Entrenamiento desde un modelo anterior.ipynb

03 - Probar modelos preentrenados.ipynb

Para que los notebooks funcionen de forma óptima debe crearse el acceso directo de la carpeta models en “Mi unidad de drive”, como se ve en la figura 1 y aceptar los permisos de drive que se solicitan en los notebooks, como se ve en las figuras 2 y 3.

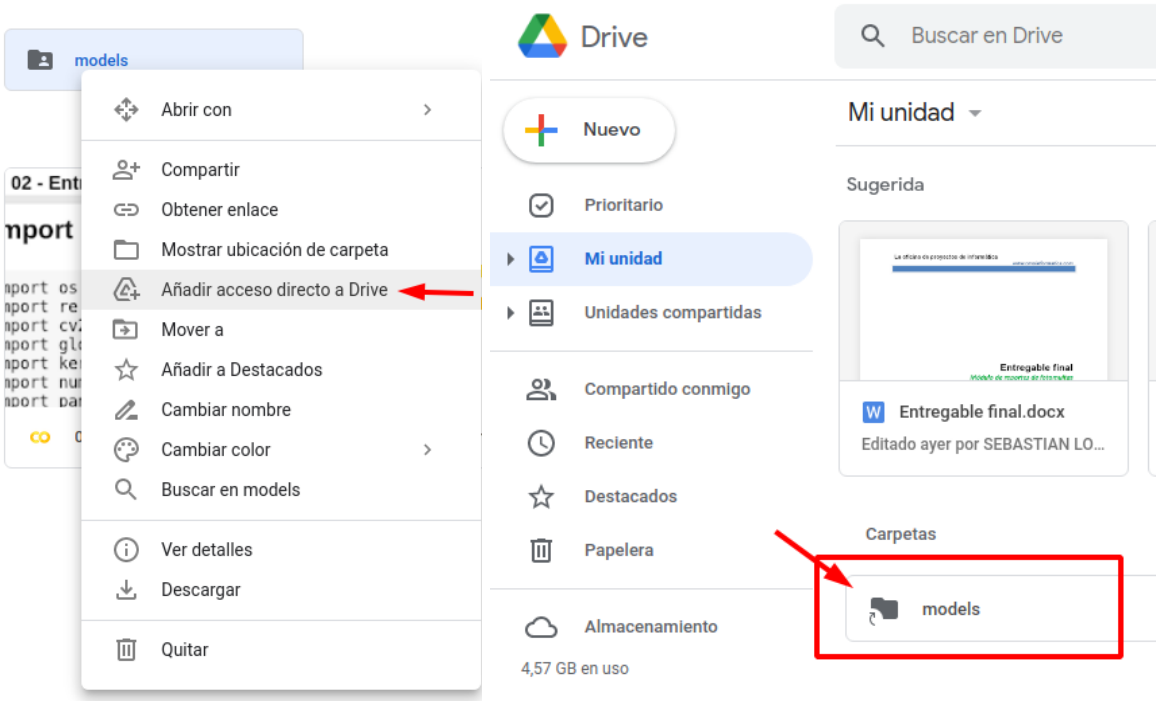

Figura 1

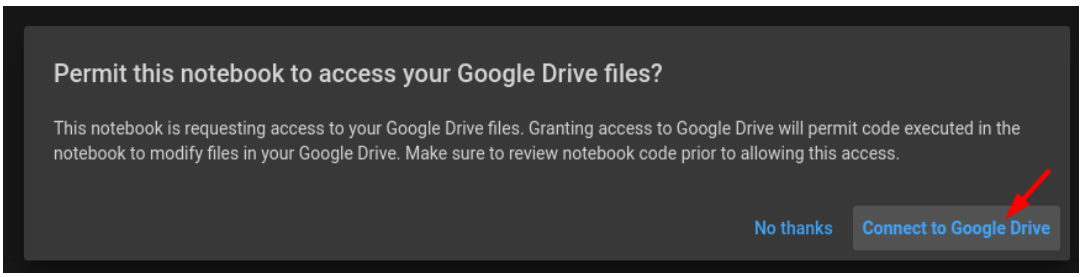

Figura 2

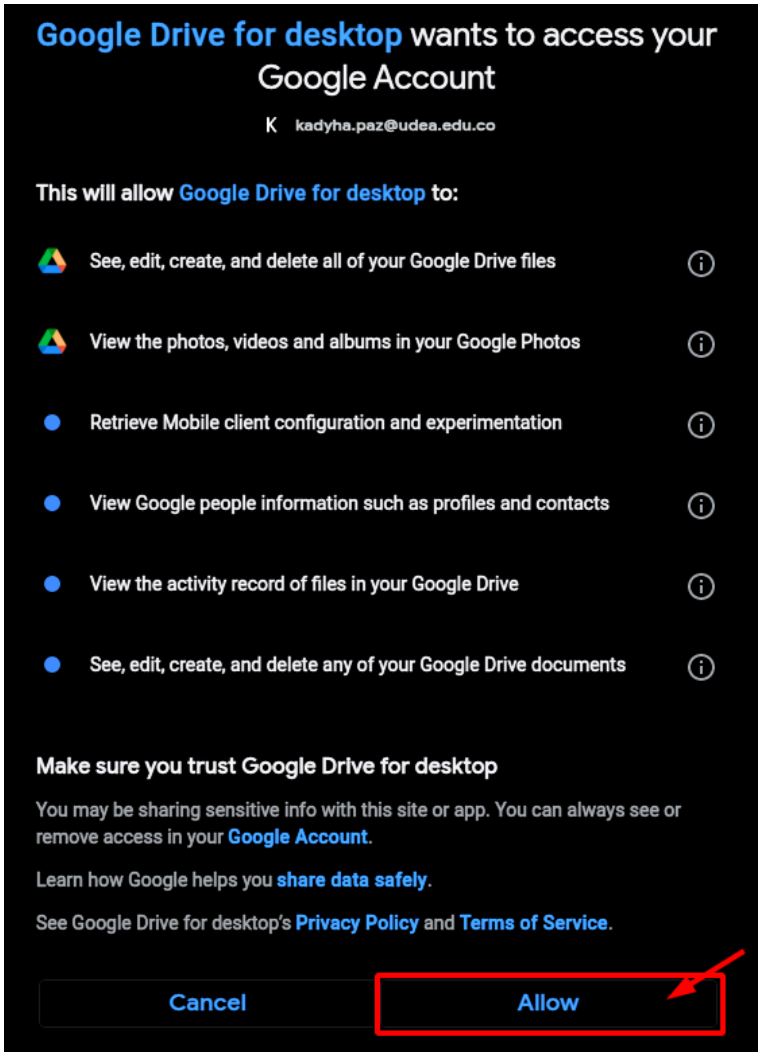

Figura 3

El notebook llamado 01 - Exploración de datos y entrenamiento inicial.ipynb contiene el código necesario para cargar y procesar los datos, transformar las imágenes para facilitar su uso y la creación de las variables X e y, que corresponden a la matriz de datos y el vector de etiquetas que serán usados para el entrenamiento, pruebas y validación. Este notebook contiene también la creación del modelo inicial, entrenamiento y evaluación de resultados, que incluye una evaluación de pérdida, exactitud y visualización de predicciones de forma individual o grupal.

El segundo notebook es llamado 02 - Entrenamiento desde un modelo anterior.ipynb, este realiza todos los pasos del notebook anterior, pero de una forma más simplificada, haciendo énfasis en el entrenamiento del modelo, para lo cual permite cargar un modelo entrenado anteriormente como base, de esta forma se pueden lograr mejores resultados, como si un modelo hubiera sido entrenado por más tiempo.

Por último el notebook 03 - Probar modelos preentrenados.ipynb, existe para probar cada uno de los modelos que han sido generados anteriormente, en este se puede cargar cada uno de ellos

para realizar evaluaciones, como la pérdida y exactitud, la visualización de las predicciones o la matriz de confusión.

Descripción de la solución

Inicialmente los datos fueron explorados, en busca de errores, pero no se encontró ninguno, se contaba con un total de 3060 imágenes que se dividen en 3 grupos, los cuales son llamados, *yes*, *no* y pred. El conjunto *yes* y *no* tienen 1500 imágenes cada uno, estos dos conjuntos fueron cargados y no se encontraron problemas en las imágenes, además se le asignó una etiqueta a cada uno, para poder diferenciarlos, por otro lado el conjunto *pred* fue descartado, ya que contiene 60 imágenes sin clasificar.

El dataset fue cargado desde https://github.com/Kadyha/Brain_Tumor_Detection_MRI, donde además se encuentra el resto de del proyecto. Luego las 3000 imágenes, fueron divididas usando el 10% de ellas para validación y después se dividieron las imágenes restantes en 20% para pruebas y 80% para entrenamiento.

El siguiente paso fue crear el modelo, el cual puede verse en la figura 4.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 214, 214, 96)      34944

 max_pooling2d (MaxPooling2D  (None, 107, 107, 96)     0
 )

 conv2d_1 (Conv2D)           (None, 97, 97, 60)        697020

 flatten (Flatten)           (None, 564540)            0

 dropout (Dropout)           (None, 564540)            0

 dense (Dense)               (None, 16)                9032656

 dropout_1 (Dropout)         (None, 16)                0

 output_1 (Dense)            (None, 2)                 34

=================================================================
Total params: 9,764,654
Trainable params: 9,764,654
Non-trainable params: 0
```

Figura 4.

A continuación se entrenó el modelo y fue guardado en drive para ser usado luego.

Después de entrenar se realizó la evaluación del modelo, obteniendo un accuracy de 69%.

A partir de este primer modelo se siguieron entrenando otros modelos en busca de mejorar el resultado, cada vez que se lograba un accuracy mayor, se usaba ese nuevo modelo como base para entrenar los siguientes, y finalmente se permite cargar cualquier modelo para ser usado, realizar evaluaciones y visualizar los resultados, como se puede ver en las figuras 5 y 6.

```
1 val_loss, val_acc = model.evaluate(XVal, yVal)
2 print('Val loss:', val_loss)
3 print('Val accuracy:', val_acc)
```

```
10/10 [==============================] - 92s 9s/step - loss: 0.5999 - accuracy: 0.8833
Val loss: 0.5998829007148743
Val accuracy: 0.8833333253860474
```

Figura 5.

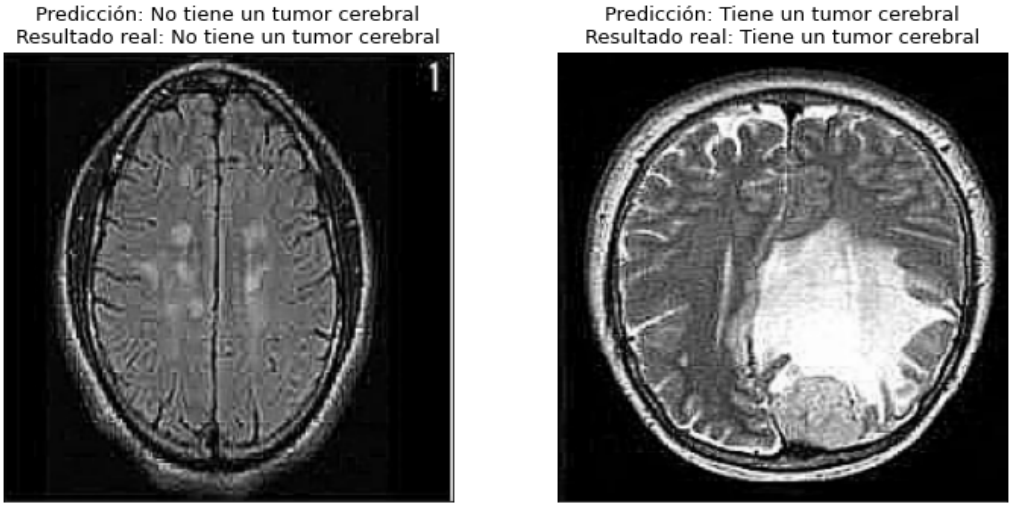

Figura 6.

Descripción de las iteraciones realizadas

Como se mencionó anteriormente, se usó el primer modelo entrenado como base para entrenar los siguientes, hasta encontrar uno mejor, el cual fue el modelo número 4, con un accuracy de 81%, este último fue usado como base para entrenar los siguientes, los cuales tendían a degradarse o no alcanzaban cambios significativos hasta llegar al modelo 11, este último fue usado como base para entrenar los siguientes modelos, sin embargo ninguno pudo superarlo. En total se entrenaron 18 modelos.

Resultados

El mejor resultado fue obtenido con el modelo número 11, el cual obtuvo un accuracy de 88% en validación.