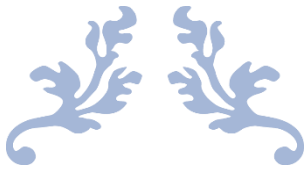


# PRÁCTICA 2
# REDES NEURONALES

Equipo 27

JORGE HERNÁNDEZ HERNÁNDEZ
2º GCID

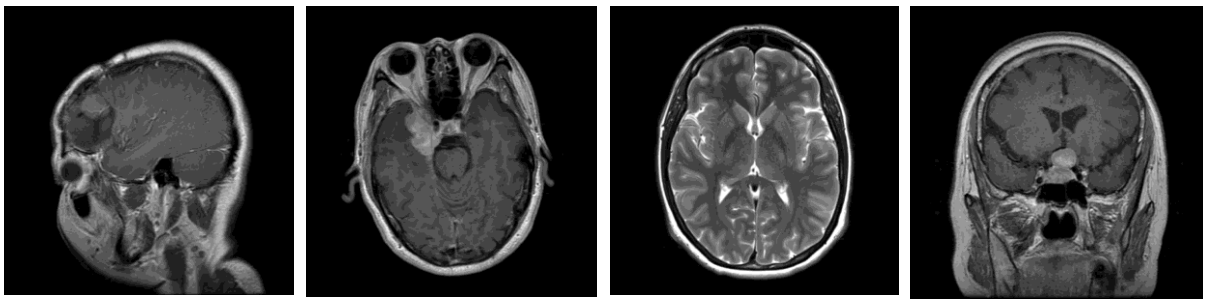
# Índice

# Introducción

Para la realización de este trabajo de redes neuronales hemos decidido utilizar un dataset de tumores celébrales Brain Tumor Classification (MRI), en nuestro dataset vemos cuatro clases diferentes de imágenes que se separar en diferentes carpetas dentro del mismo, estas son:

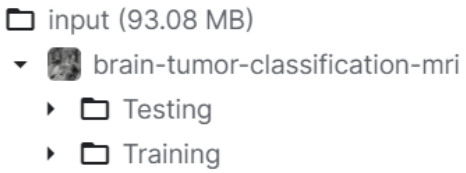
- Glioma_tumor
- Meningioma_tumor
- No_tumor
- Pituitary_tumor

Trabajaremos directamente el kaggle debido a que el dataset se encuentra en kaggle y a las facilidades que nos da kaggle para utilizar sus propios dataset.



Aquí vemos una imagen de cada una de las diferentes clases ordenadas como las menciones a las clases de la parte inicial.

## Desarrollo




Lo primero que hacemos es enlazar mi dataset con el notebook de kaggle en el que vamos a trabajar, una vez que hemos hecho este proceso tendremos que cargar en train_ds y en val_ds, la carpeta con los datos para el entrenamiento que en mi caso sería Training y la carpeta para la validación que en mi caso es Testing. Para este proceso nos hemos ayudado del código entregado como referencia.

```
import tensorflow as tf

image_size = (150, 150)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "../input/brain-tumor-classification-mri/Training",
    image_size=image_size,
    batch_size=batch_size,
    label_mode='categorical'
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "../input/brain-tumor-classification-mri/Testing",
    image_size=image_size,
    batch_size=batch_size,
    label_mode='categorical'
)

train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)
```

Importamos tensorflow como tf para poder utilizarlo con mayor facilidad, tensorflow es una librería de uso público desarrollada por Google y utilizada para el Machine Learning, en las siguientes dos líneas definimos el tamaño de la imagen que en este caso lo hemos definido de 150 x 150, ya que los tumores pueden ser de un tamaño pequeño y una imagen más amplia serviría para facilitar la detección. También defino los lotes de imágenes a 32 metemos dentro de train_ds todas las imágenes de la carpeta que se nos da para el entrenamiento y en val_ds las imágenes de la carpeta de validación. Le damos a ambos el valor categorical ya tenemos mas de dos clases en el dataset. Por último, almacenamos la información que se obtiene de realizar este proceso con el buffer_size que es un espacio de almacenamiento con el mismo tamaño que mi bathc_size, 32.

Esto nos dará como resultado la llamada a los datos de las dos carpetas mostrando la siguiente respuesta:

```
Found 2870 files belonging to 4 classes.
Found 394 files belonging to 4 classes.
```

Ahora que ya hemos cargado los datos, vamos a pasarlos por la red neuronal del código dado de base, para ver como se comportan los datos y que cambios debemos hacer para conseguir el resultado que esperamos.

La red neuronal tiene la siguiente forma:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Rescaling, Flatten
from tensorflow.keras.callbacks import EarlyStopping

model = keras.Sequential()
model.add(Rescaling(scale=(1./127.5),
                    offset=-1,
                    input_shape=(150, 150, 3)))
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(1e-3),
              metrics=['accuracy'])
```

Importarnos a parte de keras y tensorflow también los diferentes layers que vamos a utilizar, es decir las diferentes capas que compondrán nuestra red neuronal, en este caso se importan Conv2D, MaxPooling2D, Dropout, Dense, Rescaling, Flatten.

En este caso he utilizado un modelo secuencial con keras.Sequential(), crea una red neuronal vacía a la que habrá que agregar las capas correspondientes con la instrucción model.add().

La primera capa es una que nos sirve para poder, cambia la escala de los valores de entrada a un nuevo rango, en este caso buscamos una escala de [0, 255] con un rango de [-1, 1], por eso le hemos pasado el scale = 1./127.5, offset = -1. Esta capa nos sirve para asegurar que ninguna imagen se va a quedar sin escalar, asegurar que se escale correctamente, cabe destacar que como es la primera capa del modelo hemos de pasarle el input_shape en este caso (150, 150, 3) imágenes en formato 150x150.

La segunda capa esa compuesta por una Conv2D, con 32 neuronas a la que le pasamos un kernel_size de 3 x 3, y la función de activación es una relu. Utilizamos Conv2D que es al usada para nuestro caso, las imágenes, lo que hace esta capa es implementar la convolución en dos dimensiones de la imagen de entrada con un kernel, es este caso uno 3 x 3.

La siguiente es un MaxPooling2D, nos sirve para poder reducir las imágenes que nos da la capa anterior, para esto lo que hacemos es pasar un pool_size de 2 x 2, y la función recogerá el valor máximo dentro del pool_size, cada 4 pixeles.

Después nos encontramos con otra capa Conv2D, con el doble de neuronas que la anterior, es decir, 64 neuronas,  con las mismas características que la anterior.

Seguidamente vemos otro MaxPooling2D, con la misma configuración que el anterior que tiene la misma funcionalidad reducir los datos que salen de la capa de neuronas anterior.

Ahora nos encontramos con un Dropout establece aleatoriamente las unidades de entrada en 0 con una frecuencia que le hayamos dado en este caso la frecuencia es de 0.25, esto quiere decir que con este Dropout conseguimos deshabilitar temporalmente un 25% de las neuronas.

Luego nos encontramos otra capa de Conv2D, con el doble de neuronas que la anterior, es decir, 128 neuronas, con las mismas características que las anteriores.

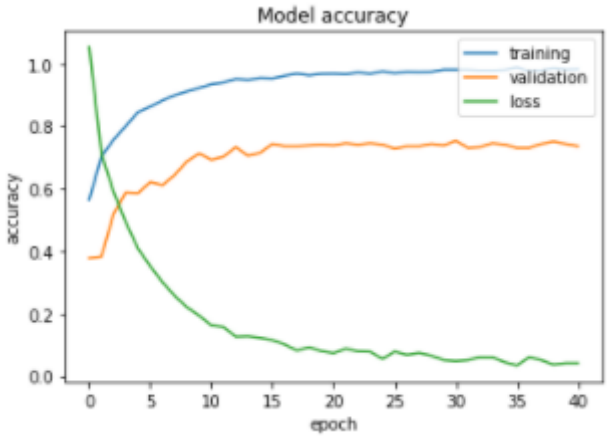
También una MaxPooling2D, igual a la anterior, que cumplen con la misma funcionalidad la de reducir los datos que hemos conseguido, para que sea más fácil su utilización.

Utilizamos el Flatten para poner todos los inputs dados los transformamos en un vector unidimensional para seguir utilizándolo, pero toda la información n un vector único.

Hora utilizamos una Dense que tiene el mismo numero de neuronas que la capa anterior ,es decir, tiene 128 neuronas, y de función de activación tenemos una relu. Esta capa tienes las neuronas y los datos a entrenar, procesando los datos para sacar una única respuesta final.

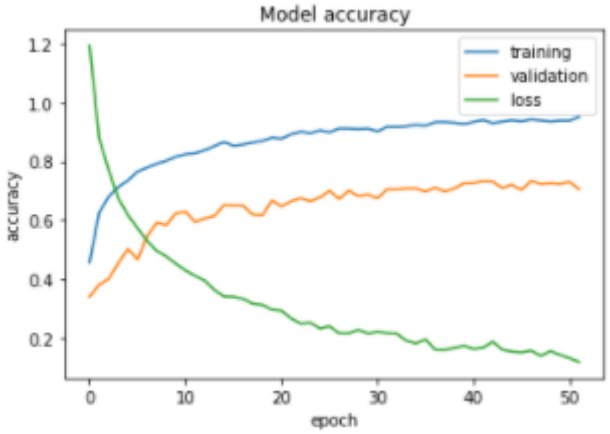
Hacemos un nuevo Dropout pero esta vez con un porcentaje mayor un 50%, y por ultimo finalizamos con una capa dense con 4 neuronas, nuestro número de posibles salidas y como función de activación utilizamos una softmax.

Al ver la gráfica este es el resultado:



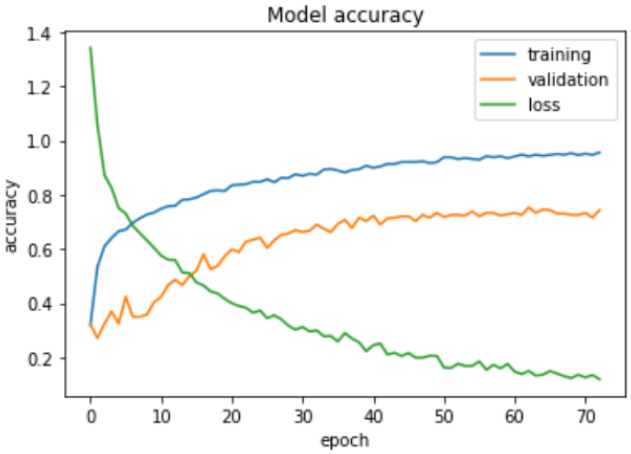

Podemos ver como la diferencia ente el validation y el training es muy amplia uno supera el umbral del 90%, como es el caso del training y el validation no llega a la frontera del 80%

Para solucionar esto subimos el número de neuronas a a 40, 80 y 160 respectivamente de las diferentes capas tanto las convolutivas como las dense a ver cual era el resultado del cambio, pero tampoco fue el acertado la diferencia entre el validation y el training se mantenía entorno al un 20% o 25%, lo que no es nada viable.




Pero la diferencia sigue siendo la misma de entre un 20% y un 25%, así que intentamos reducir el numero de neuronas a ver si de esta manera se solucionaba al hacerlo pudimos comprobar que no era posible reducir

la distancia entre, el validation y el training ya que por muchos cambios que se efectuasen la diferencia se mantenía constante.




Así que probamos a utilizar solo la carpeta con los datos de training ya que eran los datos que estaban constantemente superando el umbral del 90%, deduciendo así que las fotografías que se encontraban dentro de la carpeta de validation no eran suficientes o de la sufriente calidad como para estar igualadas con las que se encontraban en el paquete de training.

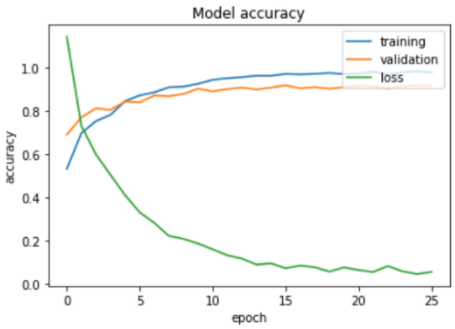
Entonces hemos cambiado el código:

```
import tensorflow as tf

image_size = (150, 150)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "../input/brain-tumor-classification-mri/Training",
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
    label_mode='categorical'
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "../input/brain-tumor-classification-mri/Training",
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
    label_mode='categorical'
)

train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)
```

El código es muy similar al anterior pre hay dos parámetros nuevos, el subset que nos sirve para decir cuál es el grupo de información, el primero es el del training y el segundo será el de validation y también añadimos el campo validation_split que hace referencia al porcentaje de datos que vamos a utilizar en el campo de validación, en este caso será un 20% del total, al cargar este código se nos indica que le entrenamiento estará formado por un total de 2296 imágenes y que el campo de validación estará formado por un total de 574 imágenes.

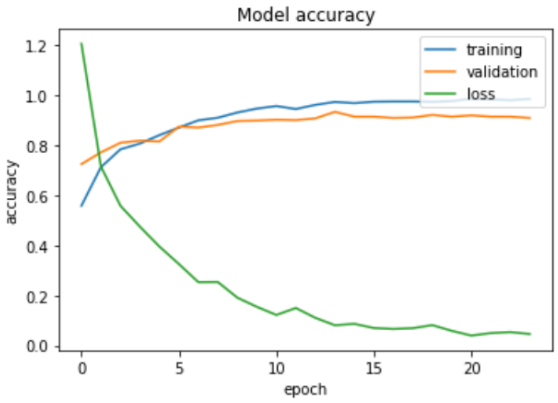
```
Found 2870 files belonging to 4 classes.
Using 2296 files for training.
Found 2870 files belonging to 4 classes.
Using 574 files for validation.
```

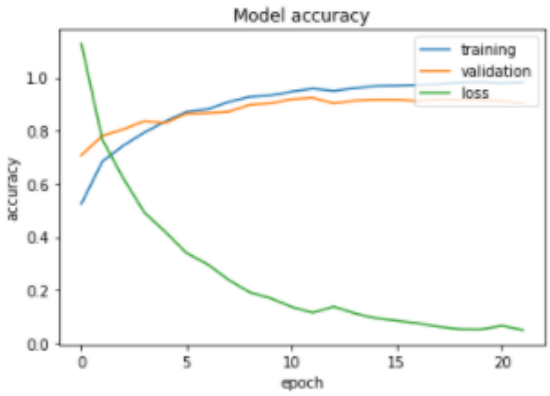
Al correr el código dado de base para el trabajo con esta nueva disposición de los datos vemos un cambio muy favorable, lo que puede indicar que he tomado una buena decisión.




Vemos un grandísimo cambio en esta ocasión la diferencia con el código de base que ya nos han dado es bastante poca ambos valores llegan o superan el umbral del 90% y entre ellos hay una diferencia que oscila entre el 4 y el 6 % lo que nos deja ver que hay una gran mejora en comparación a el anterior, con lo que se puede confirmar que le dataset de validation lo más probable es que tuviese pocas imágenes, ya que la carpeta que venía con el dataset tenía un total de 384 imágenes y el que estamos usando actualmente consta de 574.

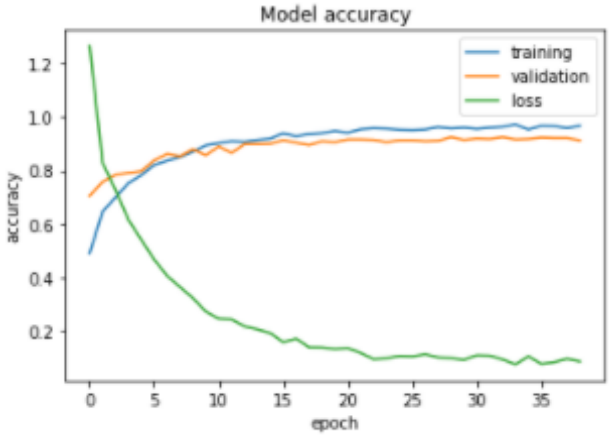
Por lo que se puede ver en la gráfica el modelo que nos dan de base es bastante correcto pero, buscando que en la parte final se juntasen mucho más training y validation intentamos subir las neuronas de nuestro modelo, al hacerlo me di cuenta que la diferencia no era mucha y además también que al aumentar el número de neuronas la líneas perdían la oscilaciones y se veían mucho más planas lo que nos indicativo de posible sobre aprendizaje, esto se puede ver en las dos siguientes graficas en la primera nuestra red neuronal consta de 40, 80, 160, 160, 4 y en la segunda 36, 72, 144, 144, 4:



Probe a bajar las neuronas para ver si de esta forma conseguía una mayor oscilación a la par de una menor distancia entre la recta de training y la de validation.

Después de reducir poco a poco el número de neuronas conseguimos que nos diese como resultado




La diferencia no sobrepasa el 5% y además podemos ver que hay oscilaciones en ambas rectas.

El modelo implementado para la obtención de esta grafica ha sido el siguiente:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Rescaling, Flatten
from tensorflow.keras.callbacks import EarlyStopping

model = keras.Sequential()
model.add(Rescaling(scale=(1./127.5),
                    offset=-1,
                    input_shape=(150, 150, 3)))
model.add(Conv2D(25, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(50, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(100, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(1e-3),
              metrics=['accuracy'])
```

Después de correr el código varias veces el EarlyStopping, solía terminar el proceso entorno a la época número 45 o 50 con lo que he bajado las épocas de 200 que teníamos inicialmente a 60 para no tener 200 épocas, pero si algo de margen de error.

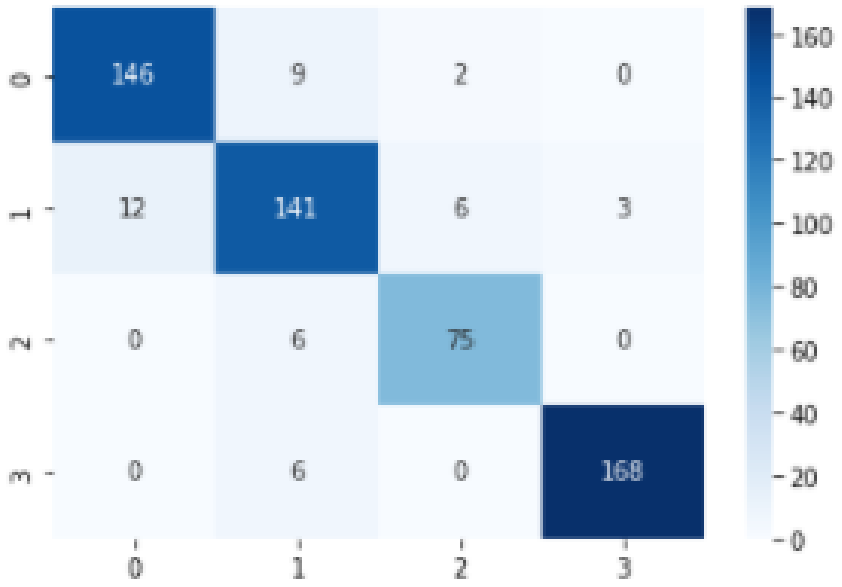
```python
epochs = 60

es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=10, restore_best_weights=True)

h = model.fit(
        train_ds,
        epochs=epochs,
        validation_data=val_ds,
        callbacks = [es]
)
```

```
Epoch 1/200
72/72 [==============================] - 5s 52ms/step - loss: 1.2604 - accuracy: 0.4943 - val_loss: 0.8687 - val_accuracy: 0.6725
Epoch 2/200
72/72 [==============================] - 4s 55ms/step - loss: 0.8390 - accuracy: 0.6633 - val_loss: 0.6323 - val_accuracy: 0.7369
Epoch 3/200
72/72 [==============================] - 4s 58ms/step - loss: 0.7049 - accuracy: 0.7134 - val_loss: 0.5580 - val_accuracy: 0.7683
Epoch 4/200
72/72 [==============================] - 4s 51ms/step - loss: 0.5853 - accuracy: 0.7565 - val_loss: 0.4504 - val_accuracy: 0.8066
Epoch 5/200
72/72 [==============================] - 4s 58ms/step - loss: 0.5144 - accuracy: 0.8031 - val_loss: 0.4314 - val_accuracy: 0.8415
Epoch 6/200
72/72 [==============================] - 4s 52ms/step - loss: 0.4161 - accuracy: 0.8253 - val_loss: 0.4118 - val_accuracy: 0.8432
Epoch 7/200
72/72 [==============================] - 4s 54ms/step - loss: 0.3652 - accuracy: 0.8458 - val_loss: 0.3669 - val_accuracy: 0.8711
Epoch 8/200
72/72 [==============================] - 4s 51ms/step - loss: 0.3287 - accuracy: 0.8689 - val_loss: 0.4021 - val_accuracy: 0.8467
Epoch 9/200
72/72 [==============================] - 4s 51ms/step - loss: 0.2783 - accuracy: 0.8898 - val_loss: 0.3713 - val_accuracy: 0.8833
Epoch 10/200
72/72 [==============================] - 4s 58ms/step - loss: 0.2600 - accuracy: 0.8968 - val_loss: 0.3658 - val_accuracy: 0.8833
Epoch 11/200
72/72 [==============================] - 4s 52ms/step - loss: 0.2624 - accuracy: 0.8907 - val_loss: 0.3413 - val_accuracy: 0.8850
Epoch 12/200
72/72 [==============================] - 4s 55ms/step - loss: 0.2054 - accuracy: 0.9203 - val_loss: 0.3883 - val_accuracy: 0.8972
Epoch 13/200
72/72 [==============================] - 4s 52ms/step - loss: 0.1926 - accuracy: 0.9233 - val_loss: 0.3332 - val_accuracy: 0.8937
Epoch 14/200
72/72 [==============================] - 4s 55ms/step - loss: 0.1465 - accuracy: 0.9429 - val_loss: 0.3928 - val_accuracy: 0.8955
Epoch 15/200
72/72 [==============================] - 4s 52ms/step - loss: 0.1608 - accuracy: 0.9334 - val_loss: 0.3640 - val_accuracy: 0.9007
Epoch 16/200
72/72 [==============================] - 4s 52ms/step - loss: 0.1471 - accuracy: 0.9429 - val_loss: 0.3543 - val_accuracy: 0.9007
Epoch 17/200
72/72 [==============================] - 4s 55ms/step - loss: 0.1454 - accuracy: 0.9421 - val_loss: 0.3578 - val_accuracy: 0.9129
Epoch 18/200
72/72 [==============================] - 4s 54ms/step - loss: 0.1283 - accuracy: 0.9495 - val_loss: 0.3700 - val_accuracy: 0.8920
Epoch 19/200
72/72 [==============================] - 4s 56ms/step - loss: 0.1408 - accuracy: 0.9425 - val_loss: 0.3568 - val_accuracy: 0.9077
Epoch 20/200
72/72 [==============================] - 4s 51ms/step - loss: 0.1172 - accuracy: 0.9530 - val_loss: 0.3905 - val_accuracy: 0.9129
Epoch 21/200
72/72 [==============================] - 4s 51ms/step - loss: 0.1005 - accuracy: 0.9625 - val_loss: 0.3706 - val_accuracy: 0.8972
Epoch 22/200
72/72 [==============================] - 4s 55ms/step - loss: 0.1054 - accuracy: 0.9569 - val_loss: 0.4172 - val_accuracy: 0.8990
Epoch 23/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0933 - accuracy: 0.9608 - val_loss: 0.3886 - val_accuracy: 0.9146
Epoch 24/200
72/72 [==============================] - 4s 55ms/step - loss: 0.0922 - accuracy: 0.9630 - val_loss: 0.3577 - val_accuracy: 0.9042
Epoch 25/200
72/72 [==============================] - 4s 58ms/step - loss: 0.1057 - accuracy: 0.9551 - val_loss: 0.4344 - val_accuracy: 0.9059
Epoch 26/200
72/72 [==============================] - 4s 53ms/step - loss: 0.1174 - accuracy: 0.9534 - val_loss: 0.3444 - val_accuracy: 0.9146
Epoch 27/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0896 - accuracy: 0.9669 - val_loss: 0.3626 - val_accuracy: 0.9094
Epoch 28/200
72/72 [==============================] - 4s 52ms/step - loss: 0.0766 - accuracy: 0.9708 - val_loss: 0.3578 - val_accuracy: 0.9181
Epoch 29/200
72/72 [==============================] - 5s 59ms/step - loss: 0.0770 - accuracy: 0.9686 - val_loss: 0.4593 - val_accuracy: 0.9181
Epoch 30/200
72/72 [==============================] - 4s 52ms/step - loss: 0.1024 - accuracy: 0.9634 - val_loss: 0.3686 - val_accuracy: 0.9111
Epoch 31/200
72/72 [==============================] - 4s 57ms/step - loss: 0.0836 - accuracy: 0.9669 - val_loss: 0.3974 - val_accuracy: 0.9199
Epoch 32/200
72/72 [==============================] - 5s 56ms/step - loss: 0.0645 - accuracy: 0.9747 - val_loss: 0.3873 - val_accuracy: 0.9199
Epoch 33/200
72/72 [==============================] - 4s 52ms/step - loss: 0.0630 - accuracy: 0.9739 - val_loss: 0.4203 - val_accuracy: 0.9233
Epoch 34/200
72/72 [==============================] - 4s 52ms/step - loss: 0.0649 - accuracy: 0.9756 - val_loss: 0.3320 - val_accuracy: 0.9233
Epoch 35/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0518 - accuracy: 0.9782 - val_loss: 0.4130 - val_accuracy: 0.9129
Epoch 36/200
72/72 [==============================] - 4s 55ms/step - loss: 0.0637 - accuracy: 0.9734 - val_loss: 0.4546 - val_accuracy: 0.9233
Epoch 37/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0585 - accuracy: 0.9752 - val_loss: 0.4279 - val_accuracy: 0.9216
Epoch 38/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0524 - accuracy: 0.9800 - val_loss: 0.4156 - val_accuracy: 0.9077
Epoch 39/200
72/72 [==============================] - 5s 61ms/step - loss: 0.0472 - accuracy: 0.9795 - val_loss: 0.4086 - val_accuracy: 0.9216
Epoch 40/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0696 - accuracy: 0.9787 - val_loss: 0.3867 - val_accuracy: 0.9216
Epoch 41/200
72/72 [==============================] - 4s 54ms/step - loss: 0.0739 - accuracy: 0.9713 - val_loss: 0.5420 - val_accuracy: 0.9059
Epoch 42/200
72/72 [==============================] - 4s 51ms/step - loss: 0.0638 - accuracy: 0.9747 - val_loss: 0.4698 - val_accuracy: 0.9181
Epoch 43/200
72/72 [==============================] - 4s 55ms/step - loss: 0.0521 - accuracy: 0.9765 - val_loss: 0.5254 - val_accuracy: 0.9233
Restoring model weights from the end of the best epoch.
Epoch 00043: early stopping
```

## Conclusión

Para sacar la conclusión nos vamos a ayudar de la siguiente matriz de confusión:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.9241 | 0.9299 | 0.9270 | 157 |
| 1 | 0.8704 | 0.8704 | 0.8704 | 162 |
| 2 | 0.9036 | 0.9259 | 0.9146 | 81 |
| 3 | 0.9825 | 0.9655 | 0.9739 | 174 |
| accuracy | | | 0.9233 | 574 |
| macro avg | 0.9201 | 0.9229 | 0.9215 | 574 |
| weighted avg | 0.9237 | 0.9233 | 0.9235 | 574 |




Con esta grafica podemos ver las diferentes confusiones que ha tenido nuestra red neuronal a medida que se ha ejecutado el programa, digamos que cuanto se ha confundido y con que lo ha confundido.

En principio vemos como la red neuronal funciona ya que la mayoría de los datos se encuentran en la diagonal lo que significa que la mayoría de las imágenes las ha clasificado de la forma correcta.

- Glioma_tumor –> equivale al 0
- Meningioma_tumor –> equivale al 1
- No_tumor –> equivale al 2
- Pituitary_tumor –> equivale al 3

Vemos como el 0 solo se ha confundido en un total de 12 imágenes con la clase 1.

Después la clase 1 es la que más se ha confundido ya que se confunde con todas las demás clases, con la 0 en un total de 9 imágenes, con la 2 en un total de 6 imágenes y con la 3 con las otras 6 imágenes.

La clase 2 es la segunda que mas se ha confundido, 6 imágenes con la clase 1 y 2 con la clase 0.

Cabe destacar que la que menos se ha confundido es la clase 3 que solo se ha confundido con la clase 1 en un total de 3 imágenes.