Clasificación de Tumores Cerebrales

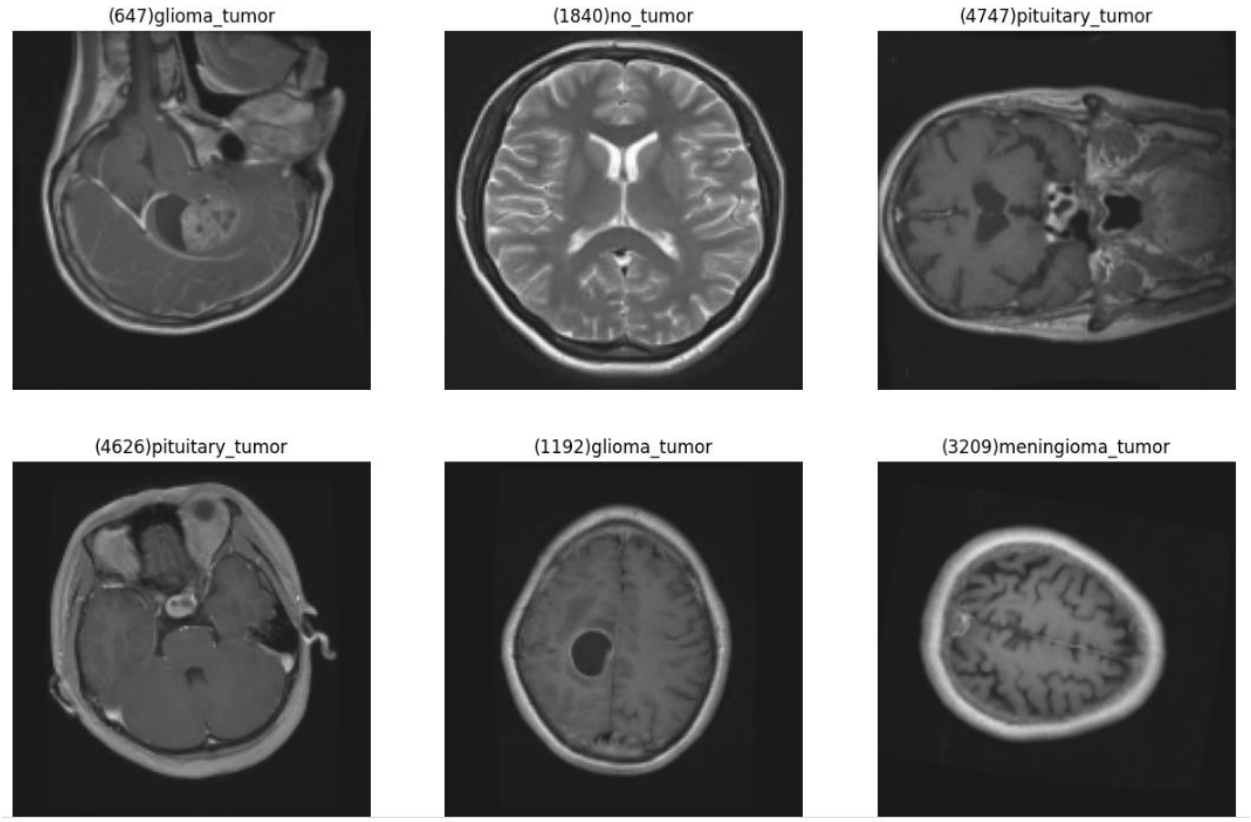
Estudiante:	Arbey de Jesús Villegas Carvajal
Documento:	CC 98695639
Asignatura:	Introducción a Deep Learning
Profesor:	Raúl Ramos
Carrera:	Ingeniería de Sistema (Virtual)
Institución:	Universidad de Antioquia

Notebooks

01_procesamiento_imagenes.ipynb

En este notebook se obtienen las imágenes y se guardan en el workspace provisto por colab. Las imágenes se redimensionan y se agregar a arreglos de numpy, un arreglo para entrenamiento y otro para test. Hay otra sección donde se hace lo mismo, pero se aumentan los datos por medio de la librería *ImageDataGenerator* de Keras.

Las imágenes generadas a partir de las originales solo se le aplican el efecto de rotación aleatoria de 0 a 350 grados. Estos conjuntos de datos se comprimen y almacenan en archivos .npz por medio de numpy.



02_exploracion_de_datos.ipynb

Básicamente, en este notebook se cargan los datos aumentados de training y de test desde github y se ponen en el workspace de Colab. El objetivo es visualizar el resultado del conjunto de datos y comparar las imágenes originales con las aumentadas.

03_01_entrenamiento_cnn_tumor_cerebral

En este notebook se entrenan redes neuronales CNN con los datos sin aumentar y los datos aumentados.

03_02_entrenamiento_tl_tumor_cerebral

En este notebook se usa Transfer Learning para entrenar el modelo con datos aumentados usando otra metodología ya que la anterior consumía mucha memoria RAM. En vez de traer los datos desde un archivo .npz, se aumentaron los datos al vuelo y se paso el objeto ImageDataGenerator en el tiempo de entrenamiento.

03_03_entrenamiento_cnn_imagedatagenerator

Entrenamiento de red neuronal CNN pasando el ImageDataGenerator en el método de entrenamiento *fit* para aumento de datos dinámicos.

04_cargar_modelo_entrenado

Los modelos entrenados anteriormente se guardaron en archivo *.keras.* En este notebook se carga los modelos desde los archivos, se evalúan y se muestra el reporte de clasificación.

Arquitectura y preprocesado

En este proyecto lo que se trato de hacer fue clasificar imágenes de resonancia magnética para clasificar tumores cerebrales en 4 clases: inexistencia de tumor, meningioma, glioma, y pituitario.

Las imágenes inicialmente son de 512x512. Para guardar los archivos .npz se pasaron a imágenes de 180x180. Cuando se creó el conjunto de datos aumentados tocó reducir las imágenes a 150x150 para no colapsar la memoria.

Después, para poder realizar Transfer Learning, dado que el modelo VGG16 tiene como input imágenes 224x224, utilicé ImageDataGenerator como entrada del entrenamiento para que el modelo mismo gestionara la cantidad de imágenes a procesar por batch.

```
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=40, horizontal_flip=True,)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(carpeta_training, target_size=(img_size, img_size), batch_size=batch_size, class_mode='categorical')
validation_generator = test_datagen.flow_from_directory(carpeta_testing, target_size=(img_size, img_size), batch_size=batch_size, class_mode='categorical')
```

```
model.fit_generator(train_generator, steps_per_epoch=steps_per_epoch, epochs=nb_epochs, validation_data=validation_generator, validation_steps=validation_steps, callbacks=[te
```

La arquitectura de las redes que se entrenaron fueron las siguientes:

CNN 1

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 222, 222, 128)	3584
activation_5 (Activation)	(None, 222, 222, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 111, 111, 128)	0
conv2d_4 (Conv2D)	(None, 109, 109, 128)	147584
activation_6 (Activation)	(None, 109, 109, 128)	0
max_pooling2d_4 (MaxPooling2D)	(None, 54, 54, 128)	0
conv2d_5 (Conv2D)	(None, 52, 52, 128)	147584
activation_7 (Activation)	(None, 52, 52, 128)	0
max_pooling2d_5 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten_1 (Flatten)	(None, 86528)	0
dense_2 (Dense)	(None, 128)	11075712
activation_8 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	516
activation_9 (Activation)	(None, 4)	0

Total params: 11374980 (43.39 MB)
Trainable params: 11374980 (43.39 MB)
Non-trainable params: 0 (0.00 Byte)

CNN 2

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_7 (Conv2D)	(None, 112, 112, 32)	9248
max_pooling2d_7 (MaxPooling2D)	(None, 56, 56, 32)	0
dropout_2 (Dropout)	(None, 56, 56, 32)	0
flatten_2 (Flatten)	(None, 100352)	0
dense_4 (Dense)	(None, 128)	12845184
dense_5 (Dense)	(None, 4)	516

Total params: 12855844 (49.04 MB)
Trainable params: 12855844 (49.04 MB)
Non-trainable params: 0 (0.00 Byte)

Transfer Learning VGG6

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
dense_21 (Dense)	(None, 4)	16388

Total params: 134276932 (512.23 MB)
Trainable params: 16388 (64.02 KB)
Non-trainable params: 134260544 (512.16 MB)

Iteraciones y resultados

Se realizaron varias iteraciones entrenando el modelo con 3 arquitecturas diferentes. Se entrenó con los datos que venían inicialmente: 2870 para entrenamiento y 394 para validación. El conjunto de datos inicial se explica con detalle en el informe 1. El mejor modelo fue el CNN 1 con un 67% de precisión sin datos aumentados.

	precision	recall	f1-score	support
0	0.88	0.21	0.34	100
1	0.55	0.95	0.69	105
2	0.72	0.77	0.74	115
3	0.86	0.76	0.81	74
accuracy			0.67	394
macro avg	0.75	0.67	0.65	394
weighted avg	0.74	0.67	0.64	394

Este mismo modelo con datos aumentados tuvo una precisión de 54%.

	precision	recall	f1-score	support
0	0.65	0.21	0.32	200
1	0.47	0.87	0.61	210
2	0.61	0.55	0.58	230
3	0.59	0.51	0.55	148
accuracy			0.54	788
macro avg	0.58	0.54	0.51	788
weighted avg	0.58	0.54	0.52	788

El modelo de Transfer Learning VGG16 tuvo el peor resultado

	precision	recall	f1-score	support
0	0.00	0.00	0.00	100
1	0.29	1.00	0.45	115
2	0.00	0.00	0.00	105
3	0.00	0.00	0.00	74
accuracy			0.29	394
macro avg	0.07	0.25	0.11	394
weighted avg	0.09	0.29	0.13	394

Conclusiones

1. No es recomendable guardar los datos en memoria porque el sistema colapsa. Es mejor dejar que el modelo vaya leyendo las imágenes desde disco para poderlas procesar de manera óptima por las capacidades que TensorFlow/Keras tiene para esto.
2. Las redes neuronales demoran mucho tiempo para entrenar, hay que tener paciencia y seguir aprendiendo para hacer esto de manera óptima o usar técnicas de calibración de hiper-parámetros. Toca investigar más para hacer esto lo más óptimo posible.
3. En algunos papers que leí lograron una precisión de más del 90% y yo solo llegué a un poco más del 60. El Transfer Learning era el más prometedor, fue el que menos precisión tuvo, y demoró más de 5 horas con solo 10 epochs y datos aumentados. Algo hice mal toca seguir investigando y estudiando.