

Diego Fernando Landeros Austria A01751654

Momento de Retroalimentación: Módulo 2 Implementación de un modelo de deep learning. (Portafolio Implementación)

Introducción

En el campo de la medicina, la detección temprana y precisa de tumores cerebrales es un desafío crítico. Existe una gran dificultad en la clasificación precisa de tumores cerebrales, esto ha llevado al desarrollo de sistemas y técnicas más avanzadas y precisas. Con esto en mente, las redes neuronales se presentan como una herramienta con gran potencial para la identificación y clasificación de estos tumores.

La relevancia del proyecto nace de la capacidad de mejorar la precisión de diagnóstico de tumores cerebrales, lo que puede conducir a tratamientos más tempranos y, en consecuencia, más eficaces. La tardanza en la detección de tumores en general generalmente resulta en un tratamiento inadecuado y un pronóstico menos favorable para el paciente. Por tanto, el desarrollo de un modelo capaz de distinguir entre cerebros sanos y cerebros con tumores es importante en el área médica y puede marcar una diferencia en la vida de los pacientes.

Descarga del dataset y preparación de los conjuntos

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5 import zipfile
6 import random
7 import matplotlib.image as mpimg

1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
4 from tensorflow.keras.optimizers import Adam
5 from tensorflow.keras.losses import binary_crossentropy, categorical_crossentropy
6 from tensorflow.keras.metrics import F1Score
7 from sklearn.metrics import f1_score
8 from tensorflow.keras.layers import Dropout
9 from tensorflow.keras.callbacks import EarlyStopping
10

1 #Configuración del acceso a kaggle para descargar el dataset
2 !mkdir -p /root/.kaggle
3 !cp "/content/drive/MyDrive/7 Semestre/Parte 2/Blumenkron/kaggle.json" /root/.kaggle/

1 #Descarga y descompresión del dataset
2 !kaggle datasets download -d preetviradiya/brian-tumor-dataset
3 path = "/content/brian-tumor-dataset.zip"
4 with zipfile.ZipFile(path, 'r') as zip_ref:
5     zip_ref.extractall("/content/")

Downloading brian-tumor-dataset.zip to /content
 90% 97.0M/107M [00:01<00:00, 64.1MB/s]
100% 107M/107M [00:01<00:00, 57.4MB/s]
```

```

1 #Imprimir imagenes aleatorias de cada directorio/categoría
2 image = random.choice(os.listdir('/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor'))
3 image=os.path.join('/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor', image)
4 img = mpimg.imread(image)
5 plt.imshow(img)
6 plt.axis('off')
7 plt.title('Imagen aleatoria del subconjunto con tumor')
8 plt.show()
9
10 image = random.choice(os.listdir('/content/Brain Tumor Data Set/Brain Tumor Data Set/Healthy'))
11 image=os.path.join('/content/Brain Tumor Data Set/Brain Tumor Data Set/Healthy', image)
12 img = mpimg.imread(image)
13 plt.imshow(img)
14 plt.title('Imagen aleatoria del subconjunto sano')
15 plt.axis('off')
16 plt.show()

```

Imagen aleatoria del subconjunto con tumor

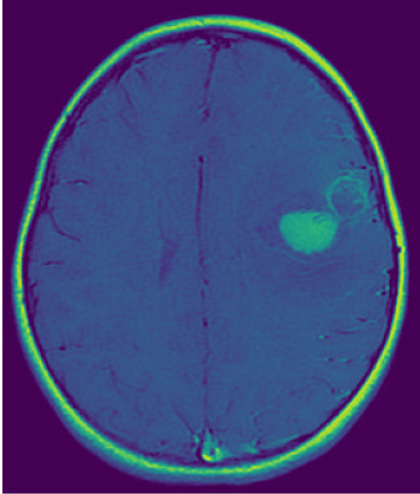
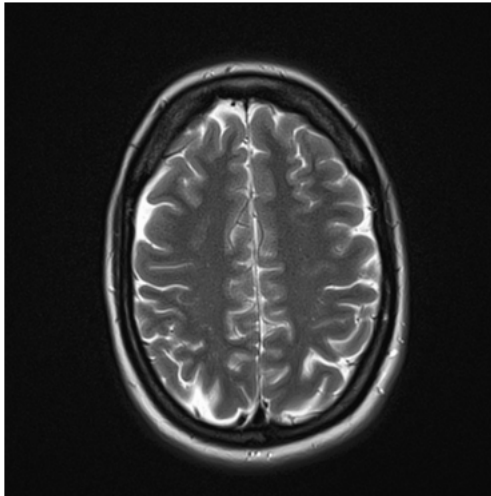


Imagen aleatoria del subconjunto sano



```

1 #Se usa la libreria de split-folders para hacer la división de conjuntos en
2 #entrenamiento, validación y prueba
3 !pip install split-folders
4 !split-folders '/content/Brain Tumor Data Set/Brain Tumor Data Set' --output train_data --ratio 0.7 0.3 --seed
5 !split-folders '/content/train_data/val' --output test_val_data --ratio 0.5 0.5 --seed 6
6
7

```

```

Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1

```

```
Copying files: 4600 files [00:00, 7008.90 files/s]
Copying files: 1381 files [00:00, 6359.24 files/s]
```

```
1 #Uso de herramientas de keras para crear los dataset de entrenamiento, validación y prueba desde los director
2 train_dir = '/content/train_data/train'
3 val_dir = '/content/test_val_data/val'
4 test_dir = '/content/test_val_data/train'
5 batch_size = 32 # Tamaño del lote
6 image_size = (256, 256) # Tamaño de las imágenes
7 # Carga de datos de entrenamiento desde el directorio
8 train = tf.keras.utils.image_dataset_from_directory(
9     train_dir,
10    image_size=image_size,
11    batch_size=batch_size,
12    shuffle=True,
13    seed=6, label_mode='binary', labels='inferred'
14 )
15 val = tf.keras.utils.image_dataset_from_directory(
16    val_dir,
17    image_size=image_size,
18    batch_size=batch_size,
19    shuffle=True,
20    seed=6, label_mode='binary', labels='inferred'
21 )
22
23 test = tf.keras.utils.image_dataset_from_directory(
24    test_dir,
25    image_size=image_size,
26    batch_size=batch_size,
27    shuffle=True,
28    seed=6, label_mode='binary', labels='inferred'
29 )
30
31 # Visualización de las clases para confirmar que se crearon correctamente los datasets
32 class_names, val_classes, test_classes = train.class_names, val.class_names, test.class_names
33 print("Clases:", class_names, val_classes, test_classes)

Found 3161 files belonging to 2 classes.
Found 676 files belonging to 2 classes.
Found 677 files belonging to 2 classes.
Clases: ['Brain Tumor', 'Healthy'] ['Brain Tumor', 'Healthy'] ['Brain Tumor', 'Healthy']
```

▼ Primer modelo (Arquitectura de CNN básica)

Para la red neuronal, dada la problemática basada en la clasificación de imágenes se decidió usar una red neuronal convoluciones, ya que este tipo de arquitecturas son especialmente útiles para reconocer patrones en imágenes. Primero se decidió crear un modelo con una arquitectura sencilla, con hiperparámetros base para tener una base sobre la cual trabajar y poder mejorar la red posteriormente utilizando técnicas de regularización, modificación de la estructura o usando hiperparámetros.

Se creó un modelo secuencial, la primera capa es una capa de convolución a la que se le indica el tamaño de entrada de la imagen, por comodidad se usaran potencias de 2 en el número de neuronas en las capas de convolución y densa. Después de las capas de convolución se utilizan capas de pooling para reducir la dimensionalidad y controlar el número de parámetros a entrenar.

```
1 #Primera versión del modelo con una arquitectura básica
2 model = Sequential(name='BrainTumorDetector')
3 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
4 model.add(MaxPooling2D((2, 2)))
5 model.add(Conv2D(64, (3, 3), activation='relu'))
6 model.add(MaxPooling2D((2, 2)))
7 model.add(Flatten())
8 model.add(Dense(128, activation='relu'))
9 model.add(Dense(1, activation='sigmoid'))

1 model.summary()
```

Model: "BrainTumorDetector"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
flatten (Flatten)	(None, 246016)	0
dense (Dense)	(None, 128)	31490176
dense_1 (Dense)	(None, 1)	129

=====
 Total params: 31509697 (120.20 MB)
 Trainable params: 31509697 (120.20 MB)
 Non-trainable params: 0 (0.00 Byte)
 =====

Además, usaremos el optimizador Adam con parametros default, como función de pérdida usaremos binary_crossentropy y para evaluar la red usaremos la métrica F1, que es muy comunmente usada en este tipo de problemas ya que es sumamente importante clasificar correctamente, tanto positivos como negativos.

Usaremos 30 epocas para entrenar la red y evaluaremos su desempeño al final del entrenamiento

```
1 optimizer = Adam()
2 loss = binary_crossentropy
3 metric = F1Score(threshold=0.5)
4 #metrics = CategoricalAccuracy()
5 model.compile(optimizer=optimizer, loss=loss, metrics=metric)
```

```
1 history = model.fit(train, epochs=30, validation_data=val)
```

```
Epoch 2/30
99/99 [=====] - 10s 96ms/step - loss: 0.1999 - f1_score: 0.9264 - val_loss: 0.1968 - val_f1_score:
Epoch 3/30
99/99 [=====] - 9s 91ms/step - loss: 0.0609 - f1_score: 0.9765 - val_loss: 0.1539 - val_f1_score: 0
Epoch 4/30
99/99 [=====] - 8s 75ms/step - loss: 0.0305 - f1_score: 0.9908 - val_loss: 0.1814 - val_f1_score: 0
Epoch 5/30
99/99 [=====] - 10s 98ms/step - loss: 0.0326 - f1_score: 0.9935 - val_loss: 0.1323 - val_f1_score:
Epoch 6/30
99/99 [=====] - 8s 75ms/step - loss: 0.0047 - f1_score: 0.9990 - val_loss: 0.2197 - val_f1_score: 0
Epoch 7/30
99/99 [=====] - 10s 99ms/step - loss: 0.0062 - f1_score: 0.9973 - val_loss: 0.1858 - val_f1_score:
Epoch 8/30
99/99 [=====] - 8s 76ms/step - loss: 0.0590 - f1_score: 0.9842 - val_loss: 0.2708 - val_f1_score: 0
Epoch 9/30
99/99 [=====] - 11s 102ms/step - loss: 0.0246 - f1_score: 0.9931 - val_loss: 0.2375 - val_f1_score:
Epoch 10/30
99/99 [=====] - 10s 101ms/step - loss: 0.0119 - f1_score: 0.9955 - val_loss: 0.1652 - val_f1_score:
Epoch 11/30
99/99 [=====] - 8s 78ms/step - loss: 0.0126 - f1_score: 0.9973 - val_loss: 0.3442 - val_f1_score: 0
Epoch 12/30
99/99 [=====] - 13s 126ms/step - loss: 0.0179 - f1_score: 0.9942 - val_loss: 0.2339 - val_f1_score:
Epoch 13/30
99/99 [=====] - 11s 92ms/step - loss: 0.0324 - f1_score: 0.9918 - val_loss: 0.2577 - val_f1_score:
Epoch 14/30
99/99 [=====] - 8s 83ms/step - loss: 0.0048 - f1_score: 0.9973 - val_loss: 0.2973 - val_f1_score: 0
Epoch 15/30
99/99 [=====] - 12s 117ms/step - loss: 0.0016 - f1_score: 1.0000 - val_loss: 0.3098 - val_f1_score:
Epoch 16/30
99/99 [=====] - 14s 128ms/step - loss: 2.6987e-04 - f1_score: 1.0000 - val_loss: 0.3253 - val_f1_sc
Epoch 17/30
99/99 [=====] - 16s 154ms/step - loss: 8.2884e-05 - f1_score: 1.0000 - val_loss: 0.3316 - val_f1_sc
Epoch 18/30
99/99 [=====] - 9s 86ms/step - loss: 4.4239e-05 - f1_score: 1.0000 - val_loss: 0.3365 - val_f1_scor
Epoch 19/30
99/99 [=====] - 9s 85ms/step - loss: 2.9294e-05 - f1_score: 1.0000 - val_loss: 0.3395 - val_f1_scor
Epoch 20/30
```

```

99/99 [=====] - 10s 90ms/step - loss: 1.0300e-05 - f1_score: 1.0000 - val_loss: 0.3440 - val_f1_score: 1.0000
Epoch 22/30
99/99 [=====] - 8s 74ms/step - loss: 1.3137e-05 - f1_score: 1.0000 - val_loss: 0.3473 - val_f1_score: 1.0000
Epoch 23/30
99/99 [=====] - 9s 85ms/step - loss: 1.0802e-05 - f1_score: 1.0000 - val_loss: 0.3496 - val_f1_score: 1.0000
Epoch 24/30
99/99 [=====] - 10s 101ms/step - loss: 9.1010e-06 - f1_score: 1.0000 - val_loss: 0.3517 - val_f1_score: 1.0000
Epoch 25/30
99/99 [=====] - 8s 74ms/step - loss: 7.7714e-06 - f1_score: 1.0000 - val_loss: 0.3537 - val_f1_score: 1.0000
Epoch 26/30
99/99 [=====] - 10s 97ms/step - loss: 6.7182e-06 - f1_score: 1.0000 - val_loss: 0.3556 - val_f1_score: 1.0000
Epoch 27/30
99/99 [=====] - 8s 73ms/step - loss: 5.8360e-06 - f1_score: 1.0000 - val_loss: 0.3574 - val_f1_score: 1.0000
Epoch 28/30
99/99 [=====] - 10s 96ms/step - loss: 5.1434e-06 - f1_score: 1.0000 - val_loss: 0.3592 - val_f1_score: 1.0000
Epoch 29/30
99/99 [=====] - 8s 75ms/step - loss: 4.5654e-06 - f1_score: 1.0000 - val_loss: 0.3608 - val_f1_score: 1.0000
Epoch 30/30
99/99 [=====] - 10s 90ms/step - loss: 4.0640e-06 - f1_score: 1.0000 - val_loss: 0.3624 - val_f1_score: 1.0000

```

Gráficando el historial de entrenamiento de la red podemos observar que tiene un aprendizaje rápido y solo existe perdida significativa del modelo en la primera epoca, sin embargo la perdida significativa solo existe en el conjunto de entrenamiento. En cuanto a la métrica de evaluación F1 Score observamos que se observa una mejoría en el desempeño del modelo de manera rápida y el modelo converge poco después de 5 épocas. Sin embargo, observamos que el modelo llega al 100% en el F1 Score, algo bastante inusual en este tipo de problemas. A continuación usaremos el conjunto de prueba para poder llegar a conclusiones finales sobre el modelo

```

1 df = pd.DataFrame(history.history)
2 df['f1_score'] = df['f1_score'].apply(lambda x: x[0])
3 df['val_f1_score'] = df['val_f1_score'].apply(lambda x: x[0])
4 df.plot(y=["loss", "val_loss"], title="Loss vs Epoch")
5 df.plot(y=["f1_score", "val_f1_score"], title="F1 Score vs Val F1 Score")

```

```
<Axes: title={'center': 'F1 Score vs Val F1 Score'}>
```

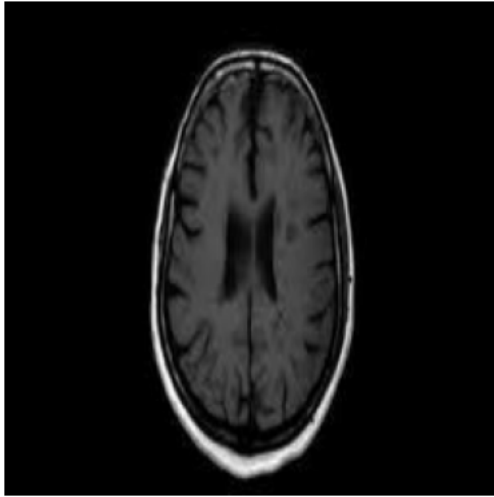
Loss vs Epoch

▼ Resultados en el conjunto de prueba

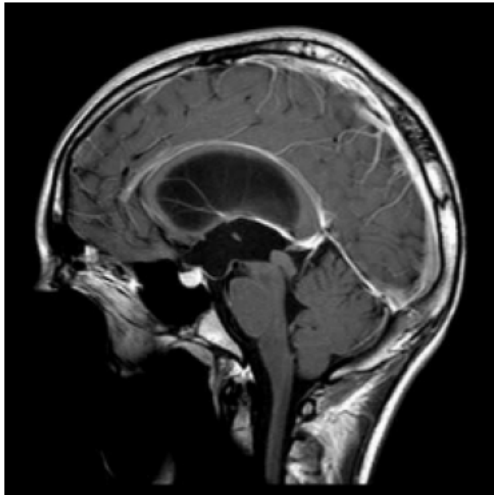
```
1 images = []
2 true_labels = []
3 predicted_labels = []
4 for x, y in test:
5     predictions = model.predict(x)
6     images.extend(x)
7     true_labels.extend(y)
8     predicted_labels.extend(predictions)
9
10 true_binary_labels = [1 if label[0] == 1 else 0 for label in true_labels]
11 predicted_binary_labels = [1 if label[0] >= 0.5 else 0 for label in predicted_labels]
12 f1 = f1_score(true_binary_labels, predicted_binary_labels)
13 print(f'F1-score en el conjunto de prueba: {f1:.2f}')
14 for i in range(5):
15     image = images[i]
16     true_label = true_labels[i]
17     predicted_label = predicted_labels[i]
18     true_class = 'Tumor' if true_label[0] == 1 else 'Sano'
19     predicted_class = 'Tumor' if predicted_label[0] >= 0.5 else 'Sano'
20     plt.imshow(image.numpy().astype(np.uint8))
21     plt.title(f'Real: {true_class}, Predicho: {predicted_class}')
22     plt.axis('off')
23     plt.show()
```

```
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 177ms/step
F1-score en el conjunto de prueba: 0.95
```

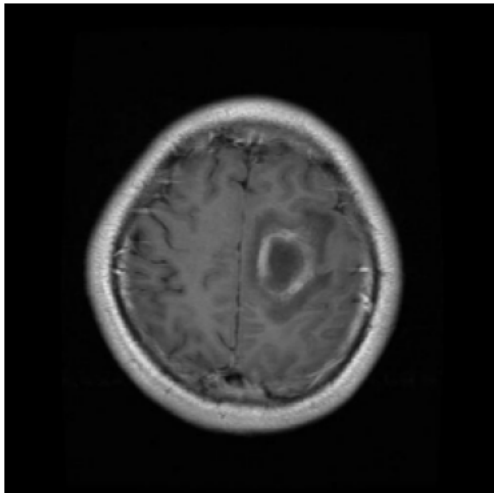
Real: Tumor, Predicho: Tumor



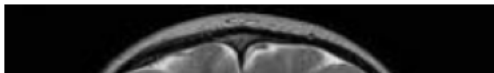
Real: Sano, Predicho: Sano

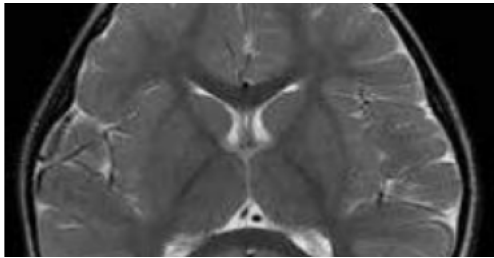


Real: Sano, Predicho: Sano



Real: Tumor, Predicho: Tumor





Después de realizar predicciones y evaluar el conjunto de prueba llegamos a la conclusión de que, efectivamente el modelo tiene un buen desempeño con puntajes de 100%, 96% y 95% en los conjuntos de entrenamiento, validación y prueba respectivamente. Podemos considerar que tiene un comportamiento inusual por el puntaje perfecto en el conjunto de entrenamiento ya que por la naturaleza del problema esto no debería ser posible, podemos considerar un mínimo overfitting en el modelo, por esto para mejorar el modelo vamos a agregar herramientas contra el overfitting, modificaciones en el learning rate y evitaremos que se sobreentrene si se alcanza convergencia.

Real: Tumor. Predicho: Tumor

▼ Modelo mejorado

Después de iterar por varios modelos, agregando capas de batch normalization, dropout, cambios en el optimizador e incluso cambios completos de la arquitectura de la red se llegó a este modelo como una mejor propuesta. Este modelo agrega una capa de dropout como herramienta para combatir el overfitting e incluye early stopping monitoreando la pérdida en el conjunto de validación.

```
1 tf.keras.backend.clear_session()
2
3 model = Sequential(name='BrainTumorDetectorUpdated')
4 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)))
5 model.add(MaxPooling2D((2, 2)))
6 model.add(Conv2D(64, (3, 3), activation='relu'))
7 model.add(MaxPooling2D((2, 2)))
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu'))
10 model.add(Dropout(0.5))
11 model.add(Dense(1, activation='sigmoid'))
```

```
1 model.summary()
```

Model: "BrainTumorDetectorUpdated"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
flatten (Flatten)	(None, 246016)	0
dense (Dense)	(None, 128)	31490176
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

```
=====  
Total params: 31509697 (120.20 MB)  
Trainable params: 31509697 (120.20 MB)  
Non-trainable params: 0 (0.00 Byte)
```



```

1 optimizer = Adam(learning_rate=0.001)
2 loss = binary_crossentropy
3 metric = F1Score(threshold=0.5)
4 model.compile(optimizer=optimizer, loss=loss, metrics=metric)
5 early_stopping = EarlyStopping(monitor='val_loss',
6                                 patience=5,
7                                 restore_best_weights=True)

1 history = model.fit(train, epochs=30, validation_data=val, callbacks=early_stopping)

Epoch 1/30
99/99 [=====] - 12s 80ms/step - loss: 40.3802 - f1_score: 0.7508 - val_loss: 0.2981 - val_f1_score:
Epoch 2/30
99/99 [=====] - 10s 100ms/step - loss: 0.1973 - f1_score: 0.9227 - val_loss: 0.1707 - val_f1_score:
Epoch 3/30
99/99 [=====] - 8s 73ms/step - loss: 0.0832 - f1_score: 0.9696 - val_loss: 0.1840 - val_f1_score: 0
Epoch 4/30
99/99 [=====] - 10s 103ms/step - loss: 0.0446 - f1_score: 0.9808 - val_loss: 0.1513 - val_f1_score:
Epoch 5/30
99/99 [=====] - 9s 83ms/step - loss: 0.0394 - f1_score: 0.9842 - val_loss: 0.1344 - val_f1_score: 0
Epoch 6/30
99/99 [=====] - 10s 101ms/step - loss: 0.0392 - f1_score: 0.9887 - val_loss: 0.1208 - val_f1_score:
Epoch 7/30
99/99 [=====] - 8s 73ms/step - loss: 0.0289 - f1_score: 0.9904 - val_loss: 0.1727 - val_f1_score: 0
Epoch 8/30
99/99 [=====] - 10s 96ms/step - loss: 0.0410 - f1_score: 0.9884 - val_loss: 0.1484 - val_f1_score:
Epoch 9/30
99/99 [=====] - 8s 74ms/step - loss: 0.0525 - f1_score: 0.9856 - val_loss: 0.1980 - val_f1_score: 0
Epoch 10/30
99/99 [=====] - 10s 99ms/step - loss: 0.0279 - f1_score: 0.9935 - val_loss: 0.1818 - val_f1_score:
Epoch 11/30
99/99 [=====] - 8s 76ms/step - loss: 0.0217 - f1_score: 0.9935 - val_loss: 0.2758 - val_f1_score: 0

```

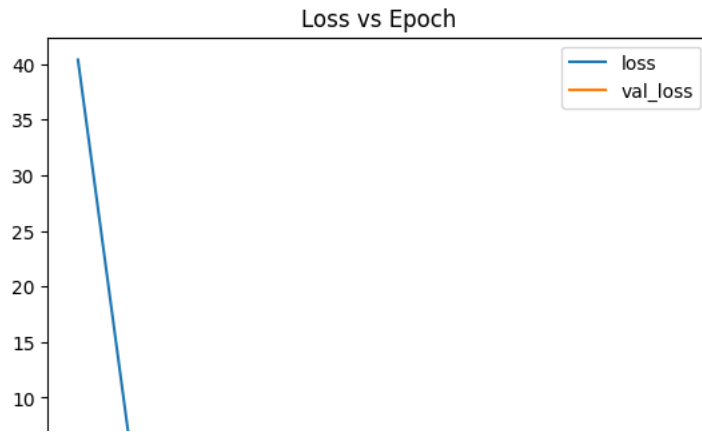
Graficando el historial de la red podemos ver que la pérdida en el conjunto de entrenamienot se estabiliza rapidamente, además de que el F1 Score en ambos conjuntos es muy similar, demostrando que generaliza bien en los datos, además que este score se estabiliza rapidamente, obtenemos resultados ligeramente peores en los puntajes que en la red anterior, sin embargo la diferencia es mínima y saliendo antes del entrenamiento estamos evitando la posibilidad de que el modelo este memorizando los datos. Para evaluar este aspecto primero revisaremos el desempeño del modelo en el conjunto de prueba

```

1 df = pd.DataFrame(history.history)
2 df['f1_score'] = df['f1_score'].apply(lambda x: x[0])
3 df['val_f1_score'] = df['val_f1_score'].apply(lambda x: x[0])
4 df.plot(y=["loss", "val_loss"], title="Loss vs Epoch")
5 df.plot(y=["f1_score", "val_f1_score"], title="F1 Score vs Val F1 Score",ylim=(0,1))

```

<Axes: title={'center': 'F1 Score vs Val F1 Score'}>



Resultados con el conjunto de prueba

```

01
1
2 images = []
3 true_labels = []
4 predicted_labels = []
5 for x, y in test:
6     predictions = model.predict(x)
7     images.extend(x)
8     true_labels.extend(y)
9     predicted_labels.extend(predictions)
10
11 true_binary_labels = [1 if label[0] == 1 else 0 for label in true_labels]
12 predicted_binary_labels = [1 if label[0] >= 0.5 else 0 for label in predicted_labels]
13
14 f1 = f1_score(true_binary_labels, predicted_binary_labels)
15 print(f'F1-score en el conjunto de prueba: {f1:.2f}')
16
17 for i in range(10):
18     image = images[i]
19     true_label = true_labels[i]
20     predicted_label = predicted_labels[i]
21     true_class = 'Tumor' if true_label[0] == 1 else 'Sano'
22     predicted_class = 'Tumor' if predicted_label[0] >= 0.5 else 'Sano'
23     plt.imshow(image.numpy().astype(np.uint8))
24     plt.title(f'Real: {true_class}, Predicho: {predicted_class}')
25     plt.axis('off')
26     plt.show()
27

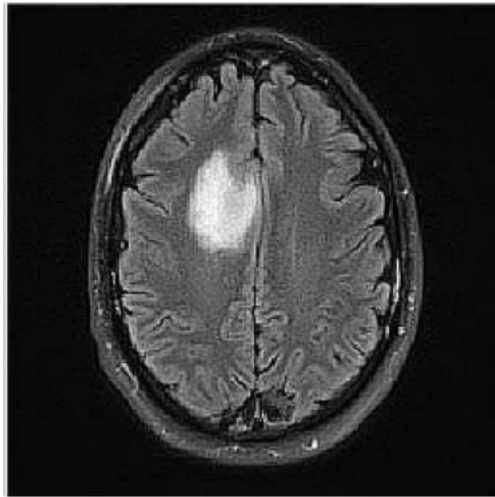
```

```

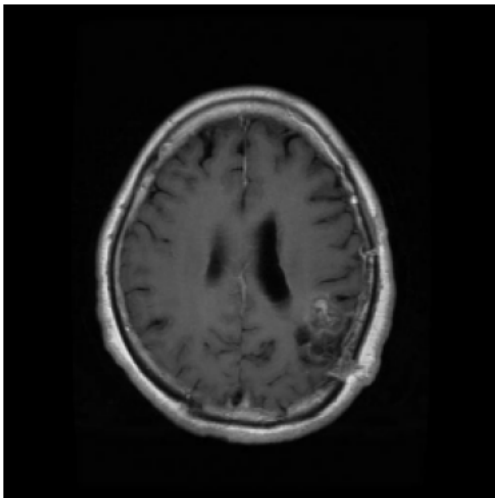
1/1 [=====] - 0s 124ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 42ms/step
1/1 [=====] - 0s 57ms/step
1/1 [=====] - 0s 39ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 72ms/step
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 42ms/step
1/1 [=====] - 0s 55ms/step
1/1 [=====] - 0s 46ms/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 38ms/step
1/1 [=====] - 0s 62ms/step
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 39ms/step
1/1 [=====] - 0s 45ms/step
1/1 [=====] - 0s 44ms/step
1/1 [=====] - 0s 100ms/step
F1-score en el conjunto de prueba: 0.96

```

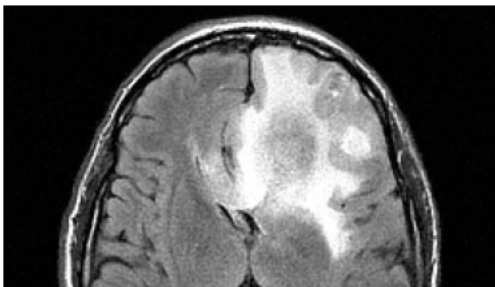
Real: Sano, Predicho: Sano

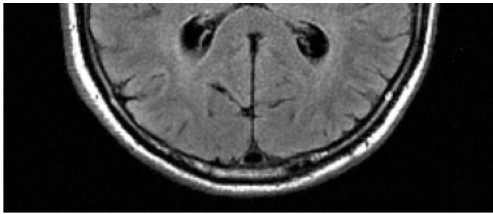


Real: Sano, Predicho: Sano

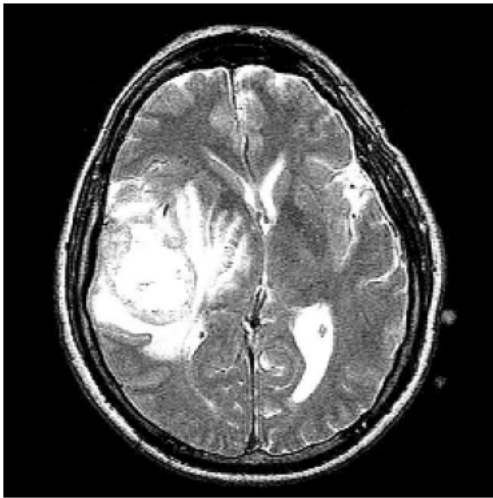


Real: Sano, Predicho: Sano

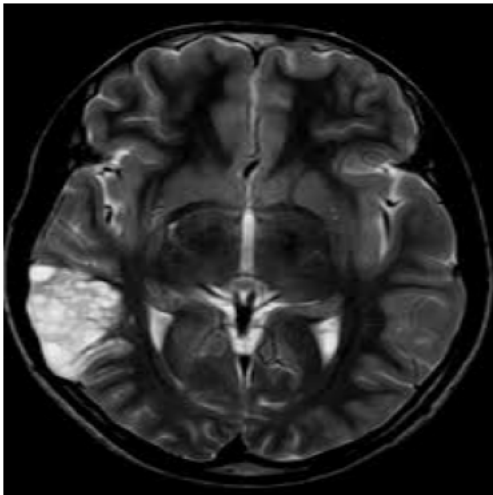




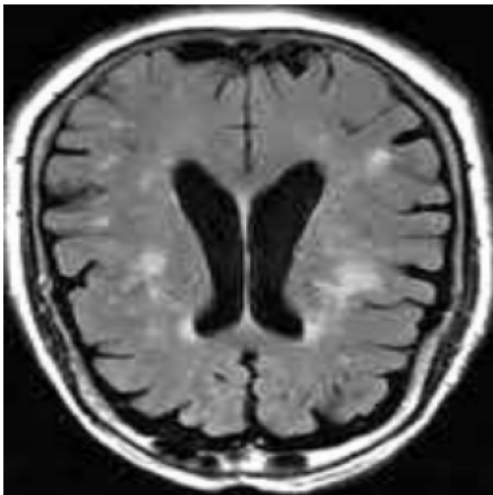
Real: Sano, Predicho: Sano



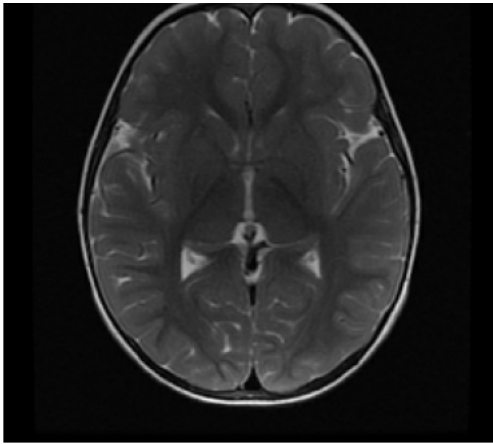
Real: Sano, Predicho: Sano



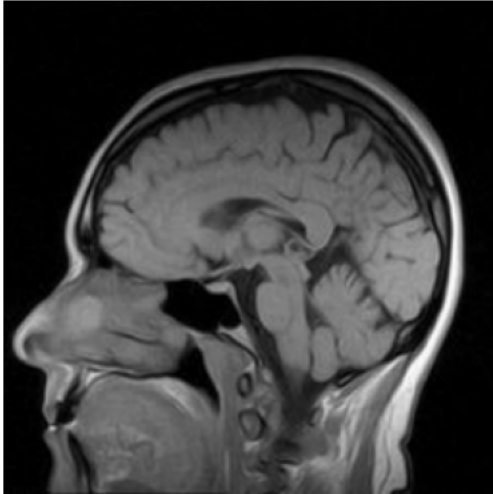
Real: Tumor, Predicho: Tumor



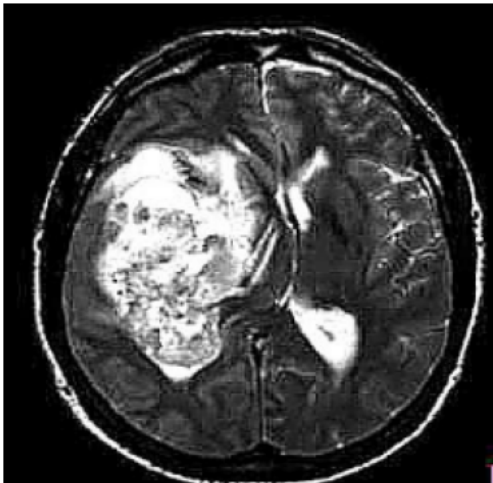
Real: Tumor, Predicho: Tumor



Real: Tumor, Predicho: Tumor



Real: Sano, Predicho: Sano



Después de utilizar el conjunto de prueba para terminar la evaluación de la red podemos observar que el desempeño visto anteriormente se mantiene con F1 Scores de 98, 97 y 96% para los conjuntos de entrenamiento, validación y prueba respectivamente. Esto nos indica que logramos robustecer el modelo, evitando el ligero sobreentrenamiento que habíamos detectado. Además ahorramos tiempo de entrenamiento y evitamos sobreentrenamiento gracias al early stopping y nos aseguramos regresar a los mejores parámetros seguidos.



▼ Conclusión



En conclusión, el problema parece ser relativamente sencillo para una red neuronal convolución ya que una arquitectura sencilla obtuvo buenos resultados y permitió que el enfoque fuera puramente en la optimización del funcionamiento de la red. Finalmente, con unos pequeños ajustes de parámetros terminó siendo suficiente para robustecer a la red.

Finalmente, el siguiente paso para un modelo de este tipo sería su implementación y utilización práctica, además se podría considerar útil probar el modelo con datos de menor calidad para retarlo y demostrar el nivel de robustez que parece tener.

▼ Modulo para realizar predicciones

En esta parte del código se incluye la estructura de funciones para realizar predicciones con nuevas imagenes, para hacer nuevas predicciones es necesario tener un directorio conteniendo las imagenes sobre las que se quiere ocupar el modelo(solamente imágenes), se llama la función make predicciones y se le pasa como parámetro el directorio mencionado, la función realizará predicciones de las imagenes y desplegara la imagen, su nombre y la predicción de cada una. Además la función regresa un diccionario con el nombre de cada archivo y su predicción para poderlo exportar posteriormente.

Alternativamente, se puede hacer la predicción de una solo imagen usando la función "make_prediction" pasandole el path de la imagen como argumento

```

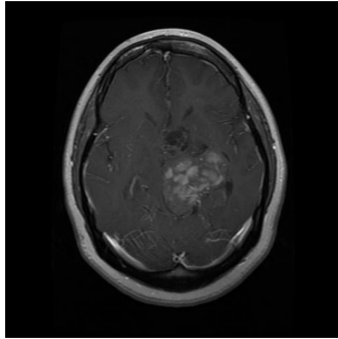
1 from tensorflow.keras.preprocessing import image
2 from tensorflow.keras.preprocessing.image import img_to_array
3
4 def preprocess_image(file_path):
5     img = image.load_img(file_path, target_size=image_size)
6     img_array = image.img_to_array(img)
7     img_array = tf.expand_dims(img_array, 0)
8     return img_array
9
10
11 def make_predictions(dir, limit=10):
12     lista_archivos = os.listdir(dir)
13     prediccions_dict = {}
14     iters = 0
15     for archivo in lista_archivos[-15:]:
16         if iters >= limit:
17             break
18         ruta_completa = os.path.join(dir, archivo)
19         make_prediction(path)
20         iters +=1
21
22 def make_prediction(path):
23     img_array = preprocess_image(path)
24     predictions = model.predict(img_array)
25     print(predictions[0])
26     predicted_label= [1 if label >= 0.5 else 0 for label in predictions[0]]
27     show_image(path, predicted_label[0])
28     print(f'Archivo: {path}')
29     print(f'Etiqueta predicha: {predicted_label[0]}')
30     print('-' * 30)
31
32 def show_image(ruta_completa, predicted_label):
33     image=ruta_completa
34     img = mpimg.imread(image)
35     plt.imshow(img)
36     plt.title(f'Archivo: "{ruta_completa}" :: Predicción: {result_map(predicted_label)}')
37     plt.axis('off')
38     plt.show()
39
40 def result_map(predicted_label):
41     if predicted_label == 1:
42         return 'Healthy'
43     elif predicted_label == 0:
44         return "Brain Tumor"

1 path = '/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg'
2 make_prediction(path)

```

```
1/1 [=====] - 0s 20ms/step  
[3.2488963e-09]
```

Archivo: "/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg" :: Predicción: Brain Tumor

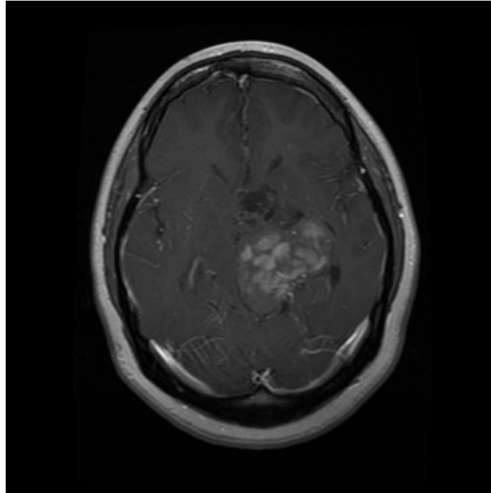


Archivo: /content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg
Etiqueta predicha: 0

```
1 dir = '/content/test_val_data/train/Brain Tumor'  
2 make_predictions(dir)
```

```
1/1 [=====] - 0s 26ms/step  
[3.2488963e-09]
```

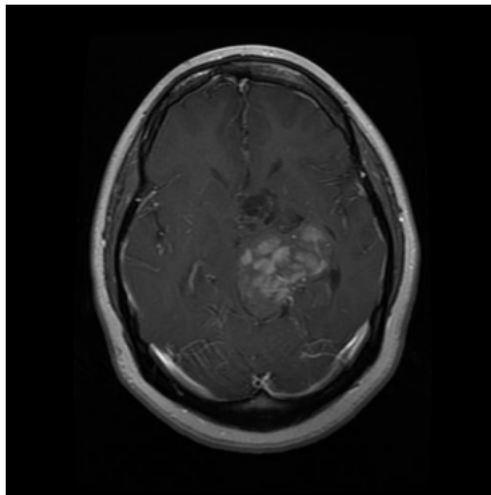
Archivo: "/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg" :: Predicción: Brain Tumor



Archivo: /content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg
Etiqueta predicha: 0

```
1/1 [=====] - 0s 35ms/step  
[3.2488963e-09]
```

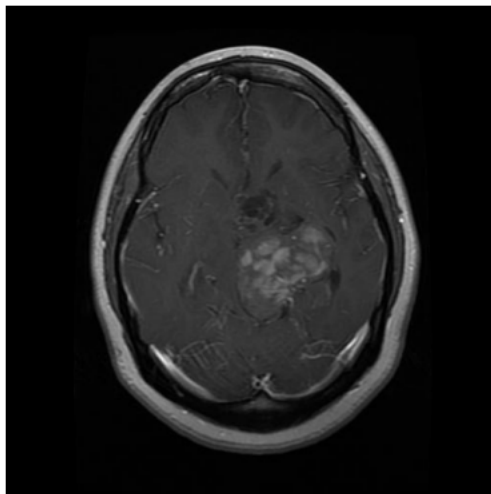
Archivo: "/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg" :: Predicción: Brain Tumor



Archivo: /content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg
Etiqueta predicha: 0

```
1/1 [=====] - 0s 26ms/step  
[3.2488963e-09]
```

Archivo: "/content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg" :: Predicción: Brain Tumor



Archivo: /content/Brain Tumor Data Set/Brain Tumor Data Set/Brain Tumor/Cancer (1).jpg