Brain Tumor Classification Using Deep Learning Algorithms

Valentina Luján Robledo

Paola Andrea Montoya Lopera

Emanuel López Higuita

Santiago Rivera Montoya

Curso: Inteligencia Artificial - Integrador

Fecha: 26 de febrero de 2025

Tabla de contenido

Introducción

Los tumores cerebrales son una de las enfermedades más agresivas que afectan tanto a niños como a jóvenes y adultos. Según el Instituto Nacional del Cáncer de los Estados Unidos, representan entre el 85 y el 90 por ciento de todos los tumores primarios del Sistema Nervioso Central (SNC) (Mehta M et al, 2011),A escala mundial, se diagnosticaron cerca de 308 102 casos nuevos de tumores de encéfalo y otros tumores del SNC en el año 2020, con un cálculo de 251329 defunciones (Sung H, 2021). La tasa de supervivencia a cinco años para personas con tumores cerebrales cancerosos es del 34% en hombres y del 36% en mujeres. Debido a la complejidad de su diagnóstico y tratamiento, la detección temprana es crucial para mejorar la esperanza de vida de los pacientes. La mejor técnica de detección disponible es la Imagen por Resonancia Magnética (IRM), la cual genera una gran cantidad de datos visuales que requieren análisis minuciosos por parte de especialistas. Sin embargo, el proceso manual de evaluación puede ser propenso a errores debido a la diversidad de tamaños y ubicaciones de los tumores cerebrales.

En este proyecto, se desarrolla un modelo de aprendizaje profundo basado en redes neuronales convolucionales (CNN) para la detección automática de tumores cerebrales en imágenes de resonancia magnética (MRI), con el objetivo de mejorar la precisión del diagnóstico, reducir los tiempos de análisis y aumentar la productividad en el sector salud.

Descripción de la Problemática

El análisis de IRM para la detección de tumores cerebrales requiere de especialistas con amplia experiencia, lo que puede limitar su acceso en países en desarrollo o en zonas con escasez de profesionales capacitados. Además, la variabilidad en las características de los tumores hace que su clasificación manual sea desafiante y propensa a inconsistencias. Un sistema automatizado basado en aprendizaje profundo puede ayudar a superar estas limitaciones al ofrecer un método de detección y clasificación más rápido y preciso.

Objetivos

Objetivo General

Desarrollar y entrenar una red neuronal convolucional (CNN) para la detección de tumores cerebrales a partir de imágenes de resonancia magnética, con el fin de mejorar la precisión y rapidez en el diagnóstico.

Objetivos Específicos

1. Implementar una arquitectura de CNN capaz de procesar imágenes de IRM para la detección de tumores cerebrales.
2. Entrenar el modelo utilizando un conjunto de datos adecuado para lograr una alta capacidad de generalización.
3. Evaluar el rendimiento del modelo en términos de precisión, sensibilidad y especificidad.
4. Comparar el desempeño de diferentes arquitecturas de redes neuronales convolucionales para la detección de tumores.
5. Identificar posibles mejoras mediante el uso de técnicas de aprendizaje por transferencia o segmentación.
6. Implementar un bot de Telegram para la detección de tumores cerebrales haciendo uso de la red neuronal entrenada

Justificación

La detección temprana de tumores cerebrales es esencial para mejorar las tasas de supervivencia y optimizar el tratamiento de los pacientes. La aplicación de técnicas de aprendizaje profundo, como las redes neuronales convolucionales, ha demostrado una mayor precisión en la clasificación de imágenes médicas en comparación con los métodos manuales. La automatización de este proceso puede beneficiar especialmente a regiones con acceso limitado a especialistas, reduciendo el tiempo de diagnóstico y mejorando la eficiencia del sistema de salud.

Alcance

Este proyecto se enfoca en el desarrollo de un modelo de CNN para la detección de tumores cerebrales utilizando imágenes de resonancia magnética. Se abordará la clasificación de tumores en 4 categorías. Sin embargo, no se incluirá el desarrollo de un sistema clínico completo ni la validación con datos de pacientes en entornos hospitalarios reales. Se desarrollará un Bot de telegram al cual se le puedan enviar imágenes de resonancias magnéticas y este, haciendo uso de la red neuronal dará un diagnóstico. Los resultados obtenidos servirán como base para futuras investigaciones y mejoras en la aplicación de inteligencia artificial en el diagnóstico médico.

Desarrollo del Proyecto

Repositorio de GitHub:

https://github.com/sRivera23/Brain-Tumor-MRI-Clasification

1. Recolección de Datos

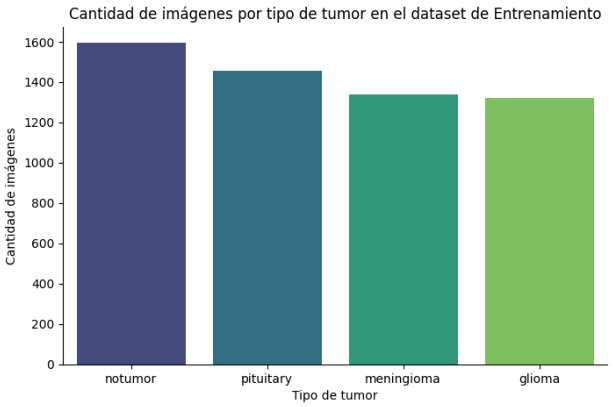
Para entrenar el modelo, se utiliza el conjunto de datos de Kaggle "Brain Tumor MRI Dataset", el cual contiene imágenes de resonancia magnética en dos subconjuntos principales:

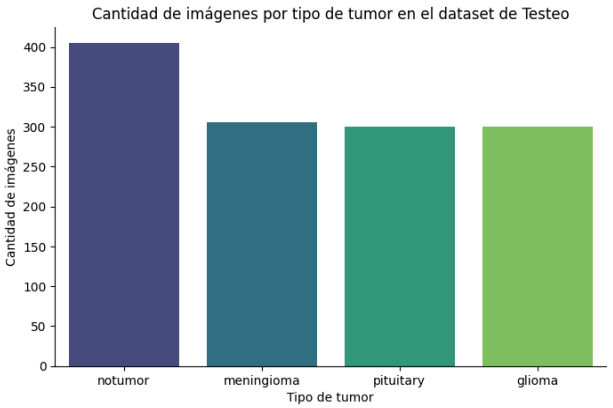
1. **Training (Entrenamiento):** Contiene imágenes que el modelo utilizará para aprender patrones en los diferentes tipos de tumores cerebrales.
2. **Testing (Prueba):** Contiene imágenes que no se han visto antes durante el entrenamiento y se usan para evaluar la capacidad de generalización del modelo.

Cada imagen pertenece a una de las siguientes clases:

- **Glioma:** Tipo de tumor cerebral que se desarrolla a partir de células gliales.
- **Meningioma:** Tumor que se origina en las meninges, las membranas que rodean el cerebro y la médula espinal.
- **No Tumor:** Imagen de resonancia sin signos de tumor.
- **Pituitary:** Tumor en la glándula pituitaria, que regula muchas funciones hormonales.

Estas categorías se distribuyen de la siguiente forma:





Para obtener los datos, se usa la API de Kaggle en Google Colab, la cual permite descargar y extraer automáticamente en carpetas organizadas por clase mediante el siguiente código en Python:

```
!kaggle datasets download -d masoudnickparvar/brain-tumor-mri-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset
License(s): CC0-1.0
brain-tumor-mri-dataset.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
import zipfile

# Descomprimir el archivo
with zipfile.ZipFile("brain-tumor-mri-dataset.zip", "r") as zip_ref:
    zip_ref.extractall("braintumor")

print("✅ Dataset extraído correctamente")
```

2. Preprocesamiento y Limpieza de Datos.

En esta fase, se organiza la información del dataset en un DataFrame de Pandas para facilitar su manipulación y posterior procesamiento.

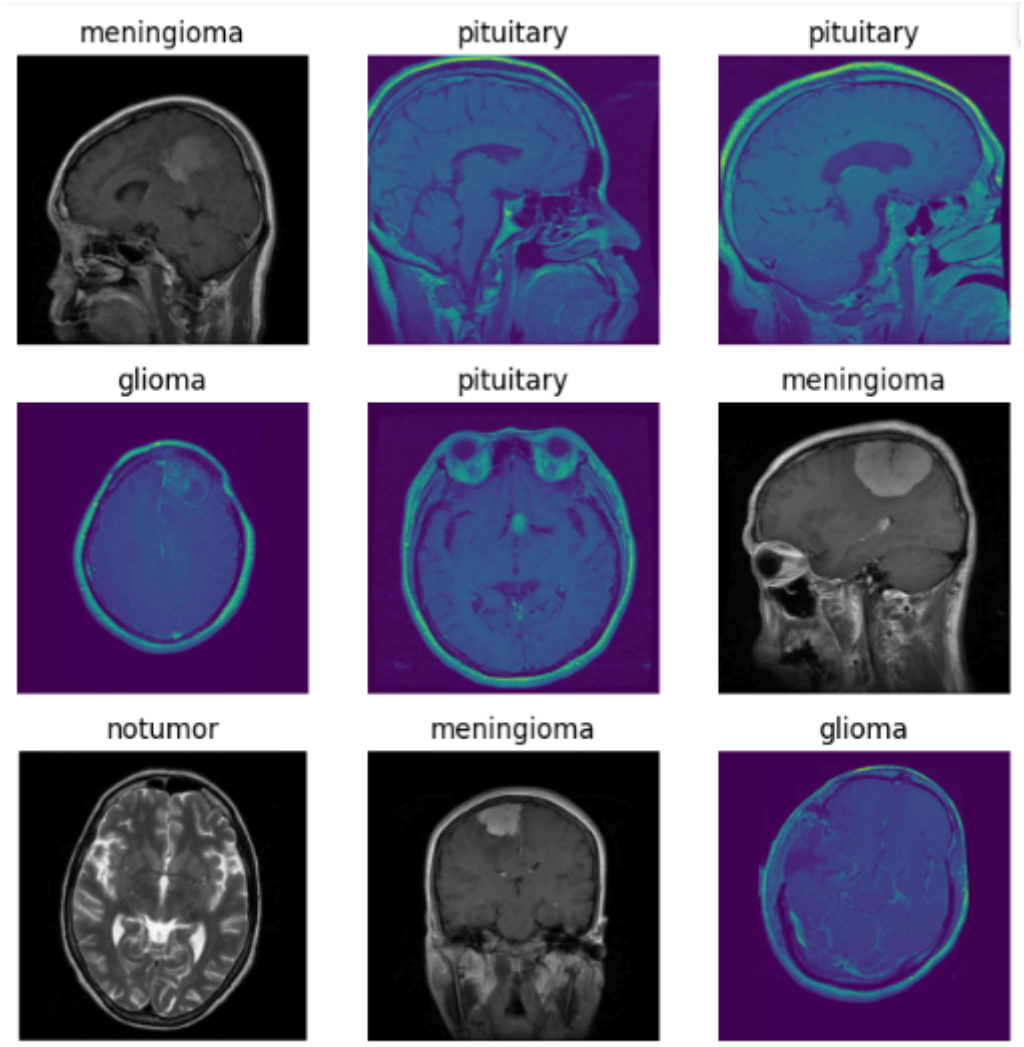
```
Train_df = '/content/braintumor/Training'

filepaths = []
labels = []
folds = os.listdir(Train_df)
for fold in folds:
    FoldPath = os.path.join(Train_df, fold)
    files = os.listdir(FoldPath)
    for file in tqdm(files):
        filepath = os.path.join(FoldPath,file)
        filepaths.append(filepath)
        labels.append(fold)
df_train = pd.DataFrame(
    data = {
        'filepath': filepaths,
        'label': labels
    }
)
```

```
Test_df = '/content/braintumor/Testing'

filepaths = []
labels = []
folds = os.listdir(Test_df)
for fold in folds:
    FoldPath = os.path.join(Test_df, fold)
    files = os.listdir(FoldPath)
    for file in tqdm(files):
        filepath = os.path.join(FoldPath,file)
        filepaths.append(filepath)
        labels.append(fold)

df_test = pd.DataFrame(
    data = {
        'filepath': filepaths,
        'label': labels
    }
)
```

Luego, se realiza la visualización de algunas imágenes aleatorias del dataset.

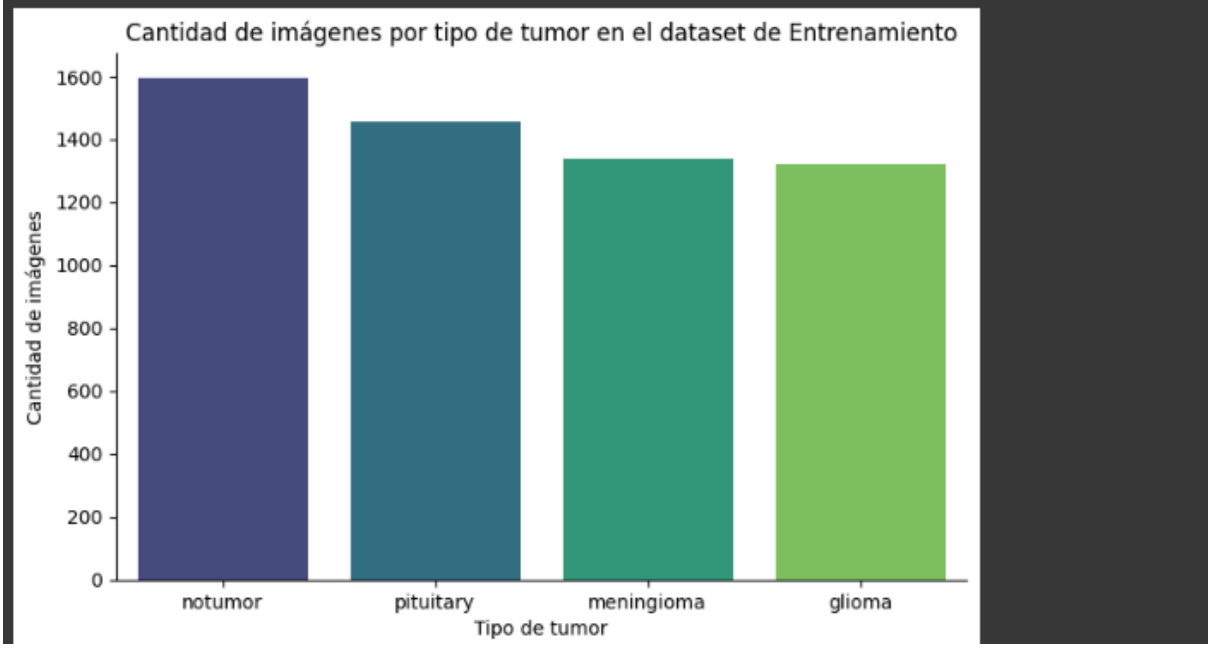
```python
import random
data_train = "/content/braintumor/Training"

fig, axes = plt.subplots(3, 3, figsize=(8,8))

for ax in axes.ravel():
    random_class = random.choice(['glioma', 'meningioma', 'notumor', 'pituitary'])
    random_img = random.choice(os.listdir(os.path.join(data_train, random_class)))
    img = Image.open(os.path.join(data_train, random_class, random_img))
    ax.imshow(img)
    ax.set_title(random_class)
    ax.axis("off")

plt.show()
```

Un posible resultado a esta ejecución es:

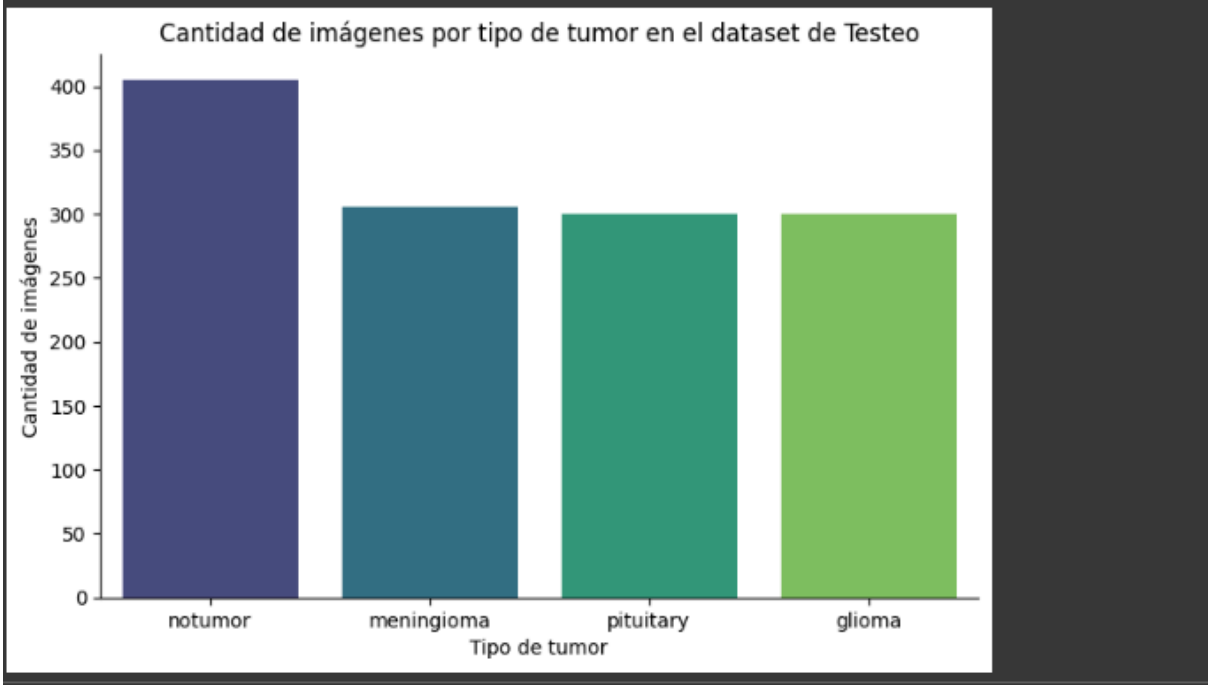


Se revisa la distribución de imágenes en el conjunto de entrenamiento y prueba.

```python
fig = plt.figure(figsize=(8,5))
ax = plt.gca()  # Get the current axes
plt.title("Cantidad de imágenes por tipo de tumor en el dataset de Entrenamiento")
sns.barplot(x=df_train["label"].value_counts().index, y=df_train["label"].value_counts(),palette="viridis")
plt.ylabel("Cantidad de imágenes")
plt.xlabel("Tipo de tumor")
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```




```python
fig = plt.figure(figsize=(8,5))
ax = plt.gca()  # Get the current axes
plt.title("Cantidad de imágenes por tipo de tumor en el dataset de Testeo")
sns.barplot(x=df_test["label"].value_counts().index, y=df_test["label"].value_counts(),palette="viridis")
plt.ylabel("Cantidad de imágenes")
plt.xlabel("Tipo de tumor")
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```



Para la preparación de los datos, las imágenes se cargan y procesan utilizando *ImageDataGenerator*, lo que permite la normalización y división del dataset en entrenamiento, prueba y validación. El código usa *ImageDataGenerator* con rescale=1./255, lo que significa que todas las imágenes se normalizan dividiendo los valores de los píxeles por 255. Esto ajusta los valores en un rango de 0 a 1, lo cual ayuda al modelo a aprender de manera más eficiente. Esto se hace de la siguiente manera:

```
valid_ts, df_test = train_test_split(df_test, test_size=0.5, random_state=42)
```

```
tr_gen = ImageDataGenerator(rescale=1/255)
ts_gen = ImageDataGenerator(rescale=1/255)
batchsize = 32
img_size = (224,224)
```

```
gen_train = tr_gen.flow_from_dataframe(df_train, x_col='filepath', y_col='label', target_size=img_size,
                                       class_mode='categorical', batch_size=batchsize, shuffle=True,color_mode='rgb')
gen_valid = ts_gen.flow_from_dataframe(valid_ts, x_col='filepath', y_col='label', target_size=img_size,
                                       class_mode='categorical', batch_size=batchsize, shuffle=True,color_mode='rgb')
gen_test = ts_gen.flow_from_dataframe(df_test, x_col='filepath', y_col='label', target_size=img_size,
                                      class_mode='categorical', batch_size=batchsize, shuffle=False ,color_mode='rgb')
```

3. Construcción del Modelo

Se utiliza una arquitectura de Red Neuronal Convolucional (CNN), ya que este tipo de redes están diseñadas para procesar datos en forma de imágenes, extrayendo características importantes mediante capas convolucionales y reduciendo dimensiones mediante capas de pooling.

Arquitectura del modelo:

El modelo consta de varias capas:

- **Capas Convolucionales (Conv2D):** Extraen características espaciales de las imágenes.
- **Capas de MaxPooling (MaxPooling2D):** Reducen la dimensionalidad de las características extraídas.
- **Capas Densas (Dense):** Actúan como la red neuronal completamente conectada para la clasificación.
- **Regularización (Dropout):** Previene el sobreajuste eliminando conexiones aleatorias durante el entrenamiento.

```python
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_size[0],img_size[1],3)),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
```

4. Entrenamiento del modelo

Antes de iniciar el entrenamiento, es importante definir métricas adecuadas para evaluar el desempeño del modelo. En este caso, además de la precisión (accuracy), se implementa la métrica F1-Score, que es especialmente útil en problemas de clasificación con clases desbalanceadas. El F1-Score combina la precisión (precision) y el recall (sensibilidad) en una única métrica, lo que ayuda a evaluar el modelo de manera más equilibrada.

```python
import tensorflow as tf
from tensorflow.keras import backend as K

def f1_score(y_true, y_pred):
    y_pred = K.cast(K.round(y_pred), 'float32')
    y_true = K.cast(y_true, 'float32')

    tp = K.sum(y_true * y_pred, axis=0)
    fp = K.sum((1 - y_true) * y_pred, axis=0)
    fn = K.sum(y_true * (1 - y_pred), axis=0)

    precision = tp / (tp + fp + K.epsilon())
    recall = tp / (tp + fn + K.epsilon())

    f1 = 2 * (precision * recall) / (precision + recall + K.epsilon())
    return K.mean(f1)
```

El modelo se compila antes de entrenarlo, definiendo el optimizador, la función de pérdida y las métricas de evaluación.

```python
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
              metrics=['accuracy', tf.keras.metrics.F1Score()])
```

Para entrenar el modelo, utilizamos el método fit(), el cual recibe los datos de entrenamiento, el número de épocas, los datos de validación y otras configuraciones adicionales. Adicionalmente, se usa Early Stopping para evitar el sobreajuste (overfitting) durante el entrenamiento del modelo.

```
history = model.fit(
    gen_train,
    epochs=10,
    validation_data=gen_valid,
    verbose=1,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)]
)
```

5. Evaluación del Modelo

Después de entrenar la red neuronal convolucional (CNN), es fundamental evaluar su rendimiento en los diferentes conjuntos de datos:

- **Conjunto de Entrenamiento (gen_train):** Datos utilizados para ajustar los pesos del modelo durante el entrenamiento.
- **Conjunto de Validación (gen_valid):** Datos no vistos por el modelo durante el entrenamiento, utilizados para medir el rendimiento y ajustar hiperparámetros.
- **Conjunto de Prueba (gen_test):** Datos completamente nuevos para evaluar la capacidad del modelo de generalizar a datos desconocidos.

La evaluación se realiza con la función model.evaluate(), de la siguiente manera:

```
print(model.evaluate(gen_train))
print(model.evaluate(gen_valid))
print(model.evaluate(gen_test))
```

Luego, se gráfica la pérdida del modelo a lo largo del entrenamiento para entender cómo fue el aprendizaje del modelo a lo largo de las épocas.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

Finalmente, se guarda el modelo para su uso posterior. Así:

```
model.save("modelo_cnn.keras")
```

5. Integración del Modelo con un Bot de Telegram

Se implementa un bot en Telegram llamado @BrainCNNBot que permite a los usuarios enviar imágenes y recibir una predicción sobre el tipo de tumor cerebral.

- **Creación del Bot:** Se usa la librería telebot para manejar la comunicación con Telegram.

```python
import telebot
import tensorflow as tf
import numpy as np
import cv2

model = tf.keras.models.load_model("modelo_cnn.keras")

bot = telebot.TeleBot("TOKEN")

class_labels = {0: "Glioma", 1: "Meningioma", 2: "No Tumor", 3: "Pituitary"}

@bot.message_handler(commands=["start"])
def send_welcome(message):
    bot.reply_to(message, "¡Hola! Envíame una imagen de una resonancia y te diré la categoría.")
```

- **Procesamiento de Imágenes:** Cuando el usuario envía una imagen, el bot la preprocesa y usa el modelo para hacer una predicción. Así:

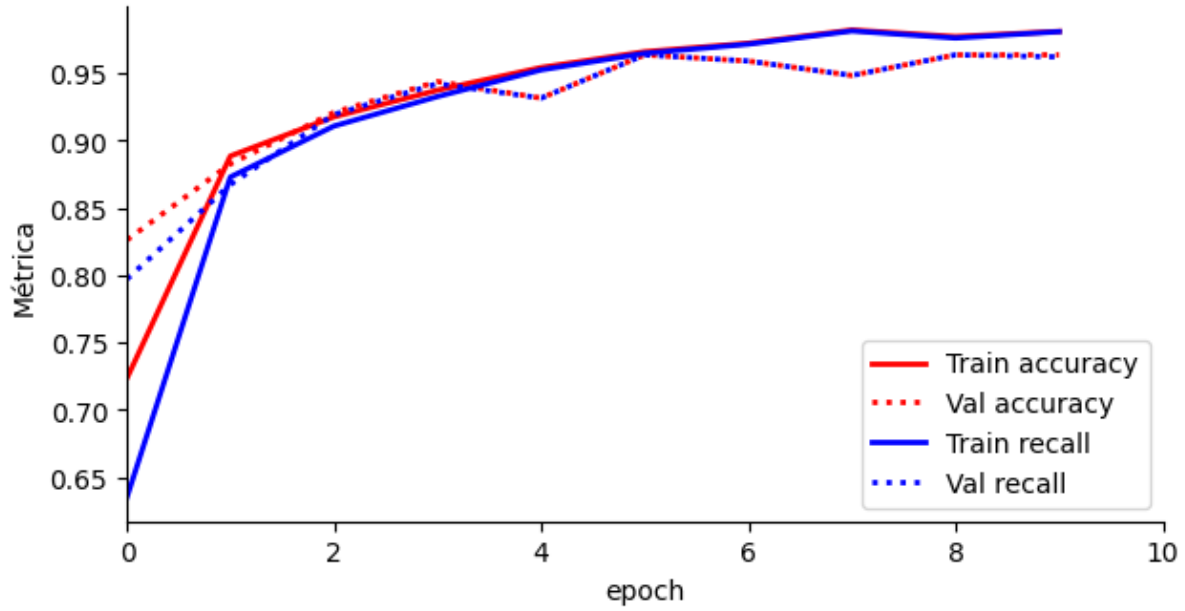
```python
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    return img

@bot.message_handler(content_types=["photo"])
def handle_image(message):
    file_info = bot.get_file(message.photo[-1].file_id)
    downloaded_file = bot.download_file(file_info.file_path)

    image_path = "received_image.jpg"
    with open(image_path, "wb") as new_file:
        new_file.write(downloaded_file)

    img = preprocess_image(image_path)
    prediction = model.predict(img)
    predicted_class = np.argmax(prediction)

    bot.reply_to(message, f"✅ Predicción: {class_labels[predicted_class]}")

bot.polling()
```

Resultados Obtenidos

Tras completar el entrenamiento del modelo de clasificación de tumores cerebrales con imágenes de IRM, se evaluó su rendimiento en los conjuntos de entrenamiento, validación y prueba. Para el conjunto de entrenamiento, se tiene que el modelo tiene una precisión alta, lo que indica que aprendió muy bien los patrones del conjunto de entrenamiento; y una pérdida bastante baja, lo que indica que el modelo ajustó bien los datos de entrenamiento y cometió muy pocos errores.

Por otro lado, en el conjunto de validación, la precisión es muy alta pero es un poco más baja que la del entrenamiento, mientras que la pérdida sigue siendo muy baja, pero más alta que en el entrenamiento. Esto indica que aunque hay una leve diferencia entre entrenamiento y validación, el modelo aún mantiene un rendimiento alto.

Finalmente, en el conjunto de prueba, la precisión sigue siendo alta, lo que indica que el modelo generaliza bien a datos nunca vistos; y la pérdida es similar a la de la validación, lo que confirma que el modelo tiene un rendimiento estable en datos nuevos. Adicionalmente, como la diferencia entre validación y prueba es mínima se puede concluir que el modelo no depende en exceso del conjunto de entrenamiento.



Conclusión

Este proyecto demuestra cómo la inteligencia artificial puede optimizar y agilizar el diagnóstico de enfermedades graves como los tumores cerebrales. La combinación de machine learning, procesamiento de imágenes y automatización mediante bots crea un sistema eficiente y accesible para la clasificación de imágenes médicas. Con este modelo, se sientan las bases para futuros desarrollos que puedan mejorar aún más la precisión, accesibilidad y aplicabilidad en el campo de la salud.

El impacto en la productividad del modelo desarrollado es significativo, especialmente en el ámbito del diagnóstico médico. Gracias a su capacidad para analizar resonancias magnéticas en segundos, se reduce drásticamente el tiempo necesario en comparación con el análisis manual, que puede tardar varios minutos. Esto no solo optimiza el flujo de trabajo en hospitales y clínicas, sino que también brinda un apoyo crucial a los radiólogos, permitiéndoles priorizar casos graves y enfocarse en diagnósticos complejos, reduciendo así su carga laboral. Además, al minimizar el riesgo de diagnósticos erróneos causados por la fatiga médica, el modelo contribuye a una mayor precisión en la detección de tumores cerebrales, lo que podría traducirse en tratamientos más oportunos y efectivos para los pacientes.

Referencias

Mehta M, Vogelbaum MA, Chang S, et al.: Neoplasms of the central nervous system. In: DeVita VT Jr, Lawrence TS, Rosenberg SA: Cancer: Principles and Practice of Oncology. 9th ed. Lippincott Williams & Wilkins, 2011, pp 1700-49.

Sung H, Ferlay J, Siegel RL, et al.: Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. CA Cancer J Clin 71 (3): 209-249, 2021

Nickparvar, M. (s.f.). Brain Tumor MRI Dataset. Kaggle. Recuperado de https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset