Tecnológico de Monterrey

Visión para Robots

Proyecto Final: Deep Learning

“Brain Tumor Detector”

Kennia Jackeline Sánchez Castillo A00517129

Profe. Braian Rodriguez

Monterrey, Nuevo León

10 de Junio, 2024

Tabla de Contenidos

1. Introducción

El avance de la inteligencia artificial y el aprendizaje profundo (Deep Learning) ha transformado significativamente diversas áreas de la ciencia y la tecnología, siendo la visión por computadora una de las más beneficiadas. Este proyecto final se enfoca en la investigación de los procesos, librerías y plataformas de Deep Learning aplicados a la visión por computadora, con un énfasis particular en el análisis de imágenes de tomografías cerebrales para la detección de tumores.

1.1. Deep Learning

El Deep Learning, o aprendizaje profundo, es una rama avanzada del aprendizaje automático (machine learning) y una subdisciplina de la inteligencia artificial (IA). Se caracteriza por el uso de redes neuronales artificiales con múltiples capas que imitan la estructura y el funcionamiento del cerebro humano, permitiendo a las máquinas procesar datos de manera más compleja y precisa.

Los puntos importantes a considerar son los siguientes:

Redes Neuronales Artificiales

En el corazón del Deep Learning están las redes neuronales artificiales, que consisten en capas de neuronas artificiales conectadas entre sí. Estas redes pueden aprender a realizar tareas mediante la adaptación de los pesos de las conexiones entre neuronas, basándose en los datos de entrenamiento.

Capas Profundas

A diferencia de las redes neuronales simples, las redes profundas tienen múltiples capas ocultas entre la entrada y la salida. Estas capas adicionales permiten a la red aprender características más complejas y abstractas de los datos, lo que mejora significativamente su capacidad para realizar tareas complejas como el reconocimiento de imágenes y el procesamiento del lenguaje natural.

Algoritmos de Entrenamiento

El entrenamiento de redes neuronales profundas se realiza mediante algoritmos como el descenso de gradiente y la retropropagación. Estos algoritmos ajustan iterativamente los

pesos de la red para minimizar el error en las predicciones del modelo, permitiendo que la red aprenda de los datos de entrenamiento.

Aprendizaje Supervisado y No Supervisado

El Deep Learning puede aplicarse tanto en entornos supervisados, donde los datos de entrenamiento incluyen etiquetas que guían el aprendizaje, como en entornos no supervisados, donde el modelo busca patrones y estructuras en datos no etiquetados.

1.2. Deep Learning para Visión de Computadoras

El Deep Learning se ha utilizado en una amplia variedad de aplicaciones que requieren el procesamiento de grandes cantidades de datos y la extracción de características complejas. Algunos puntos importantes para esta aplicación son los siguientes:

Capacidad para Aprender Características Complejas

A diferencia de los métodos tradicionales de visión por computadora que dependen de características manualmente diseñadas, el Deep Learning tiene la capacidad de aprender características complejas y abstractas directamente a partir de los datos brutos. Las redes neuronales profundas pueden identificar patrones intrincados en las imágenes, lo que resulta en un reconocimiento y clasificación más precisos.

Rendimiento Superior

El Deep Learning ha demostrado un rendimiento superior en muchas tareas de visión por computadora, superando a los métodos clásicos en competiciones y aplicaciones del mundo real. Modelos como las Redes Neuronales Convolucionales (CNN) son especialmente eficaces en tareas como la detección de objetos, segmentación de imágenes y reconocimiento de imágenes.

Disponibilidad de Datos y Poder de Cómputo

El crecimiento exponencial de los datos y el avance en las capacidades de procesamiento han facilitado el entrenamiento de modelos de Deep Learning. Con la disponibilidad de grandes conjuntos de datos etiquetados y el acceso a potentes recursos de cómputo en la nube, es posible entrenar modelos complejos de manera eficiente.

Mejora Continua

La comunidad de Deep Learning es muy activa, con constantes avances en técnicas y algoritmos. Esto asegura que las soluciones basadas en Deep Learning para visión por computadora sigan mejorando en precisión, eficiencia y capacidad para manejar tareas cada vez más complejas.

1.3. Procesos

En la visión por computadora, el Deep Learning emplea una variedad de procesos y técnicas para analizar y entender imágenes.

1. *Adquisición y Preprocesamiento de Datos*
   1.1. Recopilación de Datos
      - ➔ Datasets: Adquisición de grandes conjuntos de datos etiquetados, como ImageNet, COCO, o conjuntos de datos específicos de la aplicación (por ejemplo, tomografías cerebrales para detectar tumores).
      - ➔ Anotación de Datos: Etiquetado manual o automatizado de imágenes para proporcionar las categorías o características necesarias para el entrenamiento.

   1.2. Preprocesamiento de Imágenes
      - ➔ Normalización: Ajuste de los valores de los píxeles para que tengan una media y desviación estándar específicas.
      - ➔ Aumento de Datos (Data Augmentation): Aplicación de transformaciones aleatorias a las imágenes (rotaciones, escalado, traslación, etc.) para aumentar la diversidad del conjunto de datos y mejorar la generalización del modelo.
      - ➔ Reducción de Ruido: Aplicación de filtros para eliminar el ruido presente en las imágenes.

2. *Arquitecturas de Redes Neuronales*
   2.1. Redes Neuronales Convolucionales (CNN)
      - ➔ Capas Convolucionales: Aplicación de filtros convolucionales para extraer características locales de la imagen.
      - ➔ Capas de Pooling: Reducción de la dimensionalidad de las características para disminuir el número de parámetros y la complejidad computacional.

- ➔ Capas de Normalización: Normalización de los valores de activación para estabilizar y acelerar el entrenamiento.
- ➔ Capas Completamente Conectadas: Integración de las características extraídas para realizar la clasificación final.

2.2. Redes Residuales (ResNet)

- ➔ Bloques Residuales: Uso de conexiones residuales para facilitar el entrenamiento de redes muy profundas, mitigando el problema del desvanecimiento del gradiente.

2.3. Redes Generativas Adversariales (GANs)

- ➔ Generador y Discriminador: Utilización de dos redes (generador y discriminador) que compiten entre sí, lo que permite la generación de imágenes realistas a partir de ruido.

2.4. Redes U-Net

- ➔ Arquitectura en U: Diseño específico para la segmentación de imágenes médicas, con etapas de contracción y expansión que permiten una segmentación precisa.

3. Entrenamiento del Modelo

3.1. Definición de la Función de Pérdida

- ➔ Función de Costo: Selección de una función que mida el error del modelo (por ejemplo, entropía cruzada para clasificación, pérdida de regresión para segmentación).

3.2. Optimización

- ➔ Algoritmos de Optimización: Utilización de algoritmos como el Descenso de Gradiente Estocástico (SGD), Adam, RMSprop para minimizar la función de pérdida ajustando los pesos de la red.
- ➔ Técnicas de Regularización: Aplicación de técnicas como dropout, L2 regularization para evitar el sobreajuste.

4. *Evaluación y Validación*

4.1. Validación Cruzada

➔ División del Conjunto de Datos: Uso de técnicas de validación cruzada para evaluar el rendimiento del modelo en datos no vistos.

➔ Métricas de Evaluación: Uso de métricas como precisión, recall, F1-score, IoU (Intersection over Union) para medir la efectividad del modelo.

5. *Optimización y Ajuste Fino*

5.1. Ajuste de Hiperparámetros

➔ Búsqueda de Hiperparámetros: Ajuste de parámetros como la tasa de aprendizaje, el tamaño del lote, la cantidad de capas y neuronas para mejorar el rendimiento del modelo.

5.2. Transfer Learning

➔ Aprendizaje por Transferencia: Utilización de modelos pre entrenados en grandes conjuntos de datos y ajuste fino para tareas específicas, ahorrando tiempo y recursos de cómputo.

6. *Despliegue del Modelo*

6.1. Inferencia en Tiempo Real

➔ Optimización para Producción: Conversión del modelo entrenado para ser utilizado en entornos de producción, optimizando la latencia y la eficiencia.

➔ Herramientas y Plataformas: Implementación en plataformas como TensorFlow Serving, AWS SageMaker, o servicios de inferencia en tiempo real.

1.4. Librerías y APIs

Se han desarrollado numerosas librerías y APIs que facilitan el trabajo con imágenes, permitiendo desde operaciones básicas de manipulación hasta la implementación de complejos modelos de Deep Learning.

1. *OpenCV (Open Source Computer Vision Library)*

➔ Descripción: OpenCV es una biblioteca de código abierto que proporciona cientos de funciones para el procesamiento y análisis de imágenes.

➔ Características:
- Manipulación de imágenes y videos.
- Detección de rostros, objetos y seguimiento de movimientos.
- Algoritmos de visión por computadora y aprendizaje automático.

➔ Lenguajes Soportados: C++, Python, Java, y otros.

2. scikit-image

➔ Descripción: scikit-image es una biblioteca de Python basada en NumPy que proporciona algoritmos para el procesamiento de imágenes.

➔ Características:
- Transformaciones geométricas.
- Filtros de imágenes y técnicas de mejora.
- Segmentación de imágenes y reconocimiento de características.

➔ Lenguajes Soportados: Python.

3. TensorFlow y Keras

➔ Descripción: TensorFlow es una plataforma de código abierto para el aprendizaje automático, mientras que Keras es una API de alto nivel para construir y entrenar modelos de Deep Learning.

➔ Características:
- Implementación de modelos de redes neuronales convolucionales (CNN) para el reconocimiento y clasificación de imágenes.
- Herramientas para el preprocesamiento y aumento de datos de imágenes.
- Soporte para entrenamiento distribuido y despliegue de modelos.

➔ Lenguajes Soportados: Python, C++.

1.5. Plataformas

1. Amazon Web Services (AWS) SageMaker

➔ Descripción: Servicio de AWS para construir, entrenar e implementar modelos de aprendizaje automático.

➔ Características:
- Entornos de Jupyter notebooks integrados.
- Soporte para TensorFlow, PyTorch, MXNet, Chainer, entre otros.
- Entrenamiento distribuido y escalable.

- Herramientas para el etiquetado de datos y el aumento de datos.
- Implementación de modelos en endpoints para inferencia en tiempo real.

2. *Google Cloud AI Platform*

➔ Descripción: Plataforma de Google que proporciona herramientas para el desarrollo y despliegue de modelos de aprendizaje automático.

➔ Características:

- Soporte para TensorFlow, Keras y PyTorch.
- Entornos de notebooks (AI Platform Notebooks) para desarrollo interactivo.
- Servicio de entrenamiento distribuido y autoscaling.
- Despliegue de modelos como APIs para inferencia en tiempo real.
- Integración con otros servicios de Google Cloud.

3. *Microsoft Azure Machine Learning*

➔ Descripción: Plataforma de Azure para el desarrollo, entrenamiento y despliegue de modelos de aprendizaje automático.

➔ Características:

- Notebooks integrados y entornos de desarrollo.
- Soporte para diversos frameworks como TensorFlow, PyTorch, Scikit-Learn, y más.
- Capacidades de AutoML para automatizar el proceso de modelado.
- Entrenamiento distribuido y administración de experimentos.
- Despliegue de modelos en contenedores o como servicios web.

1.6. Enfoque del Proyecto

La capacidad de los modelos de Deep Learning para aprender y reconocer patrones complejos en grandes volúmenes de datos los hace ideales para el procesamiento de imágenes médicas.

Las tomografías computarizadas (TC) y las resonancias magnéticas (RM) son herramientas críticas en el diagnóstico de patologías cerebrales. Sin embargo, la interpretación de estas imágenes puede ser un desafío debido a la variabilidad en las características de los tumores, el ruido en las imágenes y la necesidad de un análisis minucioso. Aquí es donde el Deep Learning ofrece una solución poderosa.

En el contexto de las tomografías cerebrales, estos modelos pueden ser entrenados para identificar con alta precisión la presencia de anomalías, como tumores, mejorando así el diagnóstico médico y el tratamiento temprano de enfermedades neurológicas.

Con base en lo anteriormente mencionado, el enfoque de mi proyecto es el reconocimiento de tumores en tomografías cerebrales utilizando técnicas de Deep Learning.

2. Desarrollo

El proyecto comenzó con la adquisición y el preprocesamiento de un conjunto de datos que consistía en imágenes de resonancia magnética cerebral, divididas en categorías de 'tumor' y 'no tumor'. Utilizando OpenCV y TensorFlow, se leyeron, redimensionaron y convirtieron a escala de grises las imágenes para facilitar el procesamiento eficiente por parte del modelo de aprendizaje automático. Se seleccionó una CNN, conocida por su eficacia en tareas de reconocimiento de imágenes, como el modelo principal debido a su capacidad para aprender y extraer características de las imágenes de manera jerárquica.

2.1. Entrenamiento del Modelo

Se crearon tres modelos para comparación:

Red Neuronal Densa (DNN)

Este modelo utiliza capas completamente conectadas y es más simple en comparación con las CNN. Sirvió como referencia para comparar el rendimiento.

Red Neuronal Convolucional (CNN)

Este modelo utilizó múltiples capas de convolución y agrupación para capturar jerarquías espaciales en las imágenes.

CNN Mejorada con Dropout

Para evitar el sobreajuste, se implementó un modelo avanzado de CNN con capas de dropout, introduciendo regularización durante el entrenamiento.

Los modelos se entrenaron con imágenes preprocesadas y se validaron utilizando un subconjunto separado de los datos. Se empleó TensorBoard para monitorear el proceso de entrenamiento, lo que proporcionó información sobre el rendimiento del modelo y facilitó el ajuste de los hiper-parámetros.

2.2. Implementación de Aplicación Web

Para hacer que el modelo entrenado sea accesible para la predicción en tiempo real, se desarrolló una aplicación web. La interfaz, construida con HTML, CSS y JavaScript, proporciona una plataforma para que los usuarios carguen imágenes de resonancia magnética y reciban resultados diagnósticos. Se utilizó TensorFlow.js para cargar el modelo entrenado

en el navegador, lo que permite realizar predicciones del lado del cliente sin necesidad de procesamiento en el backend. Este enfoque no solo mejora la capacidad de respuesta de la aplicación, sino que también garantiza la privacidad de los datos del usuario, ya que las imágenes se procesan localmente.

La aplicación pre-procesa la imagen cargada para que coincida con los requisitos de entrada del modelo, incluyendo el redimensionamiento y la normalización. Después del preprocesamiento, la imagen se pasa a través del modelo de CNN para predecir la presencia de un tumor. El resultado, ya sea "Tumor detectado" o "No se detectó tumor", se muestra al usuario.

2.3. Código

Python

```python
import cv2
import matplotlib.pyplot as plt
import os
import random
import tensorflow as tf

# Función para mostrar una imagen con etiqueta y valor
def mostrar_imagen(ruta_imagen, etiqueta, valor, ax, target_size=(100,100)):
    # Leer la imagen usando cv2
    imagen = cv2.imread(ruta_imagen)

    # Convertir la imagen de BGR a RGB
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)

    # Redimensionar la imagen a target_size
    imagen = cv2.resize(imagen, target_size)

    # Convertir la imagen de BGR a GRAY
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

    # Mostrar la imagen usando matplotlib
    ax.imshow(imagen, cmap='gray')
    ax.axis('off')  # Ocultar los ejes

    # Mostrar el nombre del archivo, la etiqueta y el valor como título
    ax.set_title(f"{os.path.basename(ruta_imagen)}\n{etiqueta} ({valor})")

# Montar Google Drive
from google.colab import drive
```

```
drive.mount('/content/drive', force_remount=True)

# Ruta de la carpeta de drive
dataset_path = '/content/drive/My Drive/Brain_Tumor_Dataset'
tumor_yes = os.path.join(dataset_path, 'yes')
tumor_no = os.path.join(dataset_path, 'no')

# Obtener las rutas de todas las imágenes en las carpetas 'yes' y 'no'
imagenes_yes = [(os.path.join(tumor_yes, img), 1) for img in
os.listdir(tumor_yes) if img.lower().endswith(('.jpg', '.jpeg', '.png'))]
imagenes_no = [(os.path.join(tumor_no, img), 0) for img in
os.listdir(tumor_no) if img.lower().endswith(('.jpg', '.jpeg', '.png'))]

# Combinar las listas de imágenes
todas_imagenes = imagenes_yes + imagenes_no

# Seleccionar aleatoriamente 5 imágenes
imagenes_seleccionadas = random.sample(todas_imagenes, 5)

# Crear una figura para mostrar las imágenes
fig, axes = plt.subplots(1, 5, figsize=(20, 10))

# Mostrar cada imagen en un subplot
for ax, (img_path, valor) in zip(axes, imagenes_seleccionadas):

    # Define la etiqueta en función del valor
    etiqueta = "Tumor" if valor == 1 else "No Tumor"
    mostrar_imagen(img_path, etiqueta, valor, ax)

plt.show()
```

Python

```
datos_entrenamiento = []

for i, (imagen_path, valor) in enumerate(todas_imagenes):  # Todos los datos
    # Leer la imagen y realizar el preprocesamiento necesario
    imagen = cv2.imread(imagen_path)
    imagen = cv2.resize(imagen, (100, 100))
    imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    imagen = imagen.reshape(100, 100, 1)  # Cambiar tamaño a 100x100x1

    # Añadir la imagen y su etiqueta a la lista de datos de entrenamiento
    datos_entrenamiento.append((imagen, valor))
```

Python

```
X = []  # Imagenes de entrada (pixeles)
y = []  # Etiquetas (Tumor o No Tumor)

for imagen, valor in datos_entrenamiento:
    X.append(imagen)
    y.append(tf.constant(valor, dtype=tf.int64))  # Convertir valor a
tf.Tensor de tipo int64

import numpy as np

X = np.array(X).astype(float) / 255
np.set_printoptions(threshold=np.inf)  # Para mostrar todos los valores en
un array sin truncar
```

Python

```
# Modelo 1
modeloDenso = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(150, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Modelo 2: Red Neuronal Convulucional
modeloCNN = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Modelo 3: Red Neuronal Convulucional
modeloCNN2 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(250, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Python

```
modeloDenso.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN2.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

Python

```
from tensorflow.keras.callbacks import TensorBoard
```

Python

```
tensorboardDenso = TensorBoard(log_dir='logs/denso')
modeloDenso.fit(X, y, batch_size=32,
                validation_split = 0.15,
                epochs = 100,
                callbacks = [tensorboardDenso])
```

Python

```
%reload_ext tensorboard
%tensorboard --logdir logs
```

Python

```
tensorboardCNN = TensorBoard(log_dir='logs/cnn')
```

```
modeloCNN.fit(X, y, batch_size=32,
                validation_split = 0.15,
                epochs = 100,
                callbacks = [tensorboardCNN])
```

Python

```
tensorboardCNN2 = TensorBoard(log_dir='logs/cnn2')
modeloCNN2.fit(X, y, batch_size=32,
                validation_split = 0.15,
                epochs = 100,
                callbacks = [tensorboardCNN2])
```

Python

```
# Ver las imagenes de la variable X sin modificaciones por aumento de datos
plt.figure(figsize=(20, 8))
for i in range(10):
  plt.subplot(2, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(X[i].reshape(100, 100), cmap="gray")
```

Python

```
# Realiza el aumento de datos con varias transformaciones. Al final,
graficar 10 como ejemplo
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=15,
    zoom_range=[0.7, 1.4],
    horizontal_flip=True,
    vertical_flip=True
)

datagen.fit(X)

plt.figure(figsize=(20,8))

for imagen, etiqueta in datagen.flow(X, y, batch_size=10, shuffle=False):
```

```
  for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(imagen[i].reshape(100, 100), cmap="gray")
  break
```

Python

```
modeloDenso_AD = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN_AD = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN2_AD = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100,
100, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(250, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Python

```
modeloDenso_AD.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])

modeloCNN_AD.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])

modeloCNN2_AD.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
```

Python

```
# Separar los datos de entrenamiento y los datos de pruebas en variables
diferentes

len(X) * .85 #200
len(X) - 200 #53

X_entrenamiento = X[:200]
X_validacion = X[200:]

y_entrenamiento = y[:200]
y_validacion = y[200:]

# Usar la funcion flow del generador para crear un iterador que podamos
enviar como entrenamiento a la funcion FIT del modelo
data_gen_entrenamiento = datagen.flow(X_entrenamiento, y_entrenamiento,
batch_size=32)
```

Python

```
tensorboardDenso_AD = TensorBoard(log_dir='logs/denso_AD')

modeloDenso_AD.fit(
    data_gen_entrenamiento,
    epochs=100, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardDenso_AD]
)
```

```python
tensorboardCNN_AD = TensorBoard(log_dir='logs/cnn_AD')

modeloCNN_AD.fit(
    data_gen_entrenamiento,
    epochs=100, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardCNN_AD]
)
```

```python
modeloCNN.save('brain_tumor_cnn.h5')
```

```python
!mkdir carpeta_salida
```

```python
!tensorflowjs_converter --input_format keras brain_tumor_cnn.h5
carpeta_salida
```

2.4. Explicación del Código

Este código se centra en la creación de un modelo de aprendizaje profundo para detectar la presencia de tumores cerebrales en imágenes de resonancia magnética (MRI). Abarca desde la carga y el procesamiento de datos hasta la creación y el entrenamiento de modelos de red neuronal, y finalmente la conversión del modelo para uso en aplicaciones web.

1. Importación de Librerías

Se importan varias librerías esenciales para el procesamiento de imágenes, la manipulación de datos y la creación de modelos de aprendizaje profundo:

- `cv2` para el manejo de imágenes.
- `matplotlib.pyplot` para visualización de imágenes.
- `os` y `random` para manejar archivos y seleccionar muestras aleatorias.

- `tensorflow` para construir y entrenar modelos de aprendizaje profundo.
- `google.colab` para montar Google Drive y acceder a los datos almacenados en la nube.

2. Función para Mostrar Imágenes

Se define una función `mostrar_imagen` que lee una imagen, la procesa y la muestra con una etiqueta indicando si tiene un tumor o no.

3. Montaje de Google Drive y Preparación de Datos

Se monta Google Drive para acceder al conjunto de datos almacenado en él. Luego, se obtienen las rutas de todas las imágenes en las carpetas de tumores y no tumores, y se combinan estas rutas.

4. Selección y Visualización de Imágenes

Se seleccionan aleatoriamente algunas imágenes del conjunto de datos y se muestran para una revisión visual.

5. Preprocesamiento de Imágenes

Se pre-procesan las imágenes redimensionándolas y convirtiéndolas a escala de grises, luego se preparan las listas de datos de entrada (X) y etiquetas (y).

6. Creación y Compilación de Modelos

Se definen y compilan tres modelos diferentes:

- *Modelo Denso*: Una red neuronal densa simple.
- *Modelo CNN:* Una red neuronal convolucional.
- *Modelo CNN con Dropout:* Una red convolucional con regularización adicional para prevenir sobreajuste.

7. Entrenamiento de Modelos

Se entrena cada modelo usando los datos preprocesados y se monitorean los resultados con TensorBoard.

8. Aumento de Datos

Se utiliza `ImageDataGenerator` para realizar aumentos de datos, como rotaciones y desplazamientos, para mejorar la robustez del modelo y se muestran ejemplos de imágenes aumentadas.

9. Creación y Entrenamiento de Nuevos Modelos con Datos Aumentados

Se definen y entrenan nuevos modelos utilizando los datos aumentados para mejorar el rendimiento y la generalización del modelo.

10. Guardado y Conversión del Modelo

Finalmente, para mi caso que decidí hacer una página web para ver el modelo se guarda el modelo entrenado y se convierte a un formato compatible con TensorFlow.js, permitiendo su uso en aplicaciones web.

3. Resultados

Link: https://www.youtube.com/watch?v=LAWxl8A8gds

4. Conclusión

La integración de la IA en la imagenología médica para la detección de tumores cerebrales demuestra el significativo potencial de la tecnología para mejorar la atención médica. El desarrollo de una aplicación web utilizando un modelo de CNN proporciona una herramienta práctica y accesible para el diagnóstico preliminar, que puede ayudar a los radiólogos y reducir los retrasos en el diagnóstico. Aunque el modelo actual muestra una precisión prometedora, se pueden lograr mejoras adicionales mediante la inclusión de conjuntos de datos más diversos, arquitecturas de redes neuronales avanzadas y entrenamiento continuo con datos del mundo real.

5. Referencias

Kaggle. (n.d.). Brain MRI Images for Brain Tumor Detection. Kaggle. Retrieved June 10, 2024, from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection

IBM. (n.d.). ¿Qué es el deep learning? IBM. Retrieved June 10, 2024, from https://www.ibm.com/mx-es/topics/deep-learning

SAS. (n.d.). Deep Learning. SAS. Retrieved June 10, 2024, from https://www.sas.com/es_mx/insights/analytics/deep-learning.html

MathWorks. (n.d.). Deep Learning. MathWorks. Retrieved June 10, 2024, from https://la.mathworks.com/discovery/deep-learning.html

Cognex. (n.d.). Deep Learning vs. Machine Vision and Human Inspection. Cognex. Retrieved June 10, 2024, from https://www.cognex.com/es-mx/what-is/deep-learning/deep-learning-vs-machine-vision-and-human-inspection

Data Universe. (n.d.). Deep Learning y Visión por Computadora: Avances Recientes. Data Universe. Retrieved June 10, 2024, from https://data-universe.org/deep-learning-y-vision-por-computadora-avances-recientes/

OpenCV. (n.d.). Deep Learning with Computer Vision. OpenCV. Retrieved June 10, 2024, from https://opencv.org/blog/deep-learning-with-computer-vision/

Amazon Web Services. (n.d.). What is Deep Learning? AWS. Retrieved June 10, 2024, from https://aws.amazon.com/es/what-is/deep-learning/

DataScientest. (n.d.). Keras: La API de Deep Learning. DataScientest. Retrieved June 10, 2024, from https://datascientest.com/es/keras-la-api-de-deep-learning