



TRABAJO INTEGRADOR MODULO II

ARGENTINA PROGRAMA 4.0

DESCRIPCIÓN BREVE

Este informe está dividido en 5 partes, INTRODUCCION, METODOLOGIA, RESULTADOS, DISCUSIÓN, CONCLUSIONES Y RECOMENDACIONES. En introducción veremos cuales son los fines del proyecto y su contexto. En metodología veremos los algoritmos utilizados y daremos una explicación sencilla sobre el funcionamiento del sistema, explicaremos las técnicas utilizadas para realizar este proyecto.

En los resultados mostraremos como quedó el programa funcionando con capturas del código y también explicaremos sus finalidades.

En discusión veremos la importancia de los resultados obtenidos.

En conclusiones y recomendaciones veremos las implicancias más importantes y daremos algunas recomendaciones sobre cómo puede ser utilizado.

Julio Arturo Rodriguez
MACHINE LEARNING

Introducción

a detección de tumores cancerígenos a tenido un gran impacto para los pacientes que la padecen y no solo para ellos, también para familiares, amigos y conocidos ya que en muchos casos el descubrir el padecimiento es más problemático que la enfermedad. En algunos casos puede tener cura o no, sin embargo, es un hecho científico y de sentido común que mientras más rápido sea el diagnóstico, también más rápido empieza el tratamiento por lo cual el paciente tiene mayores posibilidades de sobrevivir al padecimiento, que además de tenerlo con problemas físicos lo tiene a él y a su entorno con problemas emocionales y psicológicos que pueden llegar a necesitar de terapia por vivir en la incertidumbre de si la persona que tiene el problema médico va a vivir o no. Además de los temas físicos y emocionales también están las cuestiones económicas dentro del círculo familiar del paciente, muchas familias terminar haciendo estudios que no pueden detectar en una fase temprana al tumor cancerígeno, lo cual lleva a las personas además de perder a una persona ya sea familiar o amigo también a quedar endeudados en momentos críticos de su vida. La finalidad de este proyecto es dar un acompañamiento a esas personas haciendo su recorrido por la enfermedad más transitado y darles la oportunidad de que no tengan grandes gastos económicos a la hora de saber cuál es su padecimiento. El tiempo que se tardaba en localizar los tumores cancerígenos era indeterminado y dependía tanto de la especialidad como los conocimientos y experiencia del doctor de cabecera de los pacientes que transitan la enfermedad sabiendo o no que la tenían, gracias al auge de las nuevas tecnologías y el avance tecnológico en lenguajes como Python, que es el lenguaje que se utilizó para

este proyecto ahora es posible detectar estos problemas médicos que tomaba un tiempo indeterminado en tan solo 3 minutos, según la información obtenida en el siguiente enlace ([La inteligencia artificial acelera el diagnóstico de tumores cerebrales - NCI \(cancer.gov\)](#)).

. En este proyecto se utilizó lo siguiente:

- pandas
- opencv-python
- matplotlib
- nbformat
- scikit-image
- seaborn
- tensorflow
- scikit-learn
- plotly
- ipython
- keras_preprocessing
- ipykernel

. Para realizar este proyecto se hizo un refuerzo conceptual sobre conocimientos adquiridos en el modulo 1 donde vimos Python, Numpy, Matplotlib, Pandas, Numba, Scify, scikit-learn, de forma que la realización del proyecto se de en forma mas fluida y no haya ningún inconveniente con ningnùn participante del mismo a la hora de participar del proyecto de este modulo de machine learning

dictado por la universidad nacional de misiones como un curso de extensión universitaria en colaboración con la nación para el programa de inclusión digital “ARGENTINA PROGRAMA 4.0”, en el año 2023. A sabiendas de que hay 2 tipos de programadores siendo el primero un programador que se sabe la sintaxis y el segundo el que se sabe la lógica, en el dictado de las clases se vio mucho la lógica y se la priorizo no sin dejar de lado la sintaxis que también se vio una guía y se vieron en clases, también se dejó material extra para complementar y se incentivó la investigación científica. El trabajo realizado en la primera etapa consistió en lógica algorítmica y sintaxis, con algunos ejercicios y cuestionarios mientras que en esta etapa consistió en cuestionarios, material teórico, repaso y documentación de código.

En las siguientes páginas se verán los conceptos incorporados y documentados mediante imágenes, explicando la finalidad del código y su lógica. En pocas palabras explicare como pone su granito de arena en este programa.

metodología

E

n este proyecto se utilizaron las redes neuronales convulsionales o res Net para para la clasificación de imágenes y una res Unet para la segmentación de imágenes en la cual cada imagen se reduce al tamaño de un píxel para obtener una mayor información del área de interés, a esto se le sumo inteligencia artificial y se lo implemento con Python.

Este programa o archivo ejecutable cuanta, con una ventaja, y la misma es la implementación de la red neuronal convulsional, la que cuenta con un espacio en memoria dando la oportunidad de reducir el numero de variables de entrada, entre otros aspectos para tener en cuenta que mejoran la eficacia del programa como puede ser en las tareas de reconocimiento y clasificación de imágenes. Este programa cuenta con la posibilidad de mejoras y la posibilidad de transferencia de conocimiento la cual consiste en aprovechar los conocimientos ya adquiridos y enseñarle nuevas cosas y explotando todo el potencial que ya tiene el programa.

Como ya se mencionó en este proyecto se utilizó:

- pandas
- opencv-python
- matplotlib
- nbformat
- scikit-image
- seaborn
- tensorflow
- scikit-learn
- plotly
- ipython
- keras_preprocessing
- ipykernel

Pandas: Se lo utilizo por ser un DataFrame que se utiliza para la manipulación y análisis de datos la cual nos ayuda a cargar, transformar y analizar los datos que utilizamos en este proyecto.

opencv-python: A esta librería de Python se la utilizo porque sirve para el procesamiento y análisis de imágenes y videos, aunque lo usamos para las imágenes.

Matplotlib: a matplotlib lo usamos por ser una biblioteca de Python para la visualización mediante su creación de gráficos.

Nbformat: la cual nos permite trabajar con la implementación base de Jupyter Notebook.

scikit-image: esta librería contiene una amplia variedad de técnicas que nos permiten trabajar con imágenes.

Seaborn: esta es una librería especializada en visualización de datos basada en matplotlib “ambas nos van a ser útiles”.

Tensorflow: esta librería esta especializada en el aprendizaje automático.

scikit-learn: esta librería de Python esta especializada en el aprendizaje no supervisado y supervisado.

Plotly: se lo utilizo por ser al igual que matplotlib y seaborn una biblioteca especializada en la visualización de datos.

keras_preprocessing: se lo utilizo para preparar los datos para el entrenamiento del modelo de aprendizaje profundo.

Ipykernel: permite interactuar al equipo con Jupyter notebook en este caso en particular.

Para evitar que todos esas librerías interrumpieran el correcto funcionamiento de la computadora personal o PC durante la vida diría y el tiempo que durase el proyecto se dio la opción de hacerlo en Google Colab o en un entorno virtual de Python, siendo el segundo el que utilice para el proyecto. El entorno se creo mediante el siguiente comando en la terminal PYTHON -M VENV seguido del nombre del entorno virtual. Al terminar la creación del entorno instale las librerías con los siguientes comandos en la terminal de VisualStudioCode luego de

iniciar el entorno virtual ingresando el siguiente comando para evitar errores Set-ExecutionPolicy Unrestricted -Scope Process y utilizando el script activar.ps1 para entornos creados en Windows, sistema operativo utilizado para el proyecto.

- pip install pandas
- pip install opencv-python
- pip install matplotlib
- pip install nbformat
- pip install scikit-image
- pip install seaborn
- pip install tensorflow
- pip install -U scikit-learn
- pip install plotly
- pip install ipython
- pip install keras.preprocessing
- pip install ipykernel

Una vez instaladas las librerías y en conjunto con los docentes de la catedra, se descargó el archivo mediante clonación de este que se encontraba en un repositorio de GitHub brindado por los docentes, también se podía descargar mediante drive. Pero no fue mi caso, luego de eso se procedió a la documentación del código, con actualizaciones una vez por semana, en unos o dos casos tuve que borrar el archivo el entorno virtual por la incorrecta compatibilidad de un programa, una de las librerías pedía mínimo Python 3.8 mientras yo tenía el 7, por eso reinstalé todo. como se ve a continuación todo el código este documentado, luego importamos las siguientes librerías.

import pandas as pd: a esta la importamos para la manipulación y análisis de datos, es muy útil para este proyecto y otros al ser un DataFrame. Nos va a ser útil para analizar y manipular las imágenes con las que trabaja el programa.

import numpy as np: gracias a su función o utilidad matemática nos va a ser útil para una mayor precisión del programa

import seaborn as sns: en esta parte importamos seaborn que es una librería de visualización estadística, la cual nos será útil más adelante.

import matplotlib.pyplot as plt: acá estamos importando otra librería de visualización de datos

import zipfile: esto va a ser muy importante ya que nos permitirá leer archivos en formato zip.

import cv2: es una librería de open-cv para el aprendizaje automático. Esta puede reconocer y clasificar objetos e identificar, la cual será utilizada para reconocer los tumores en las imágenes.

from skimage import io: esta la vamos a utilizar para la manipulación de imágenes, sirve para manipular imágenes en variables y guardarlos o mostrarlos.

import tensorflow as tf: aprovecharemos sus funciones y lo utilizaremos para entrenar el programa.

from tensorflow.python.keras import Sequential: la utilizamos para la creación de redes neuronales.

from tensorflow.keras import layers, optimizers: a este lo utilizamos para la optimización del algoritmo

from tensorflow.keras.applications import DenseNet121: es una red neuronal preen trenada que la vamos a usar para este programa.

from tensorflow.keras.applications.resnet50 import ResNet50:
esta es otra red neuronal preentrenada que necesitamos para este programa.

from tensorflow.keras.layers import *: esta nos permitirá acceder a todas las funciones sin estar especificando cada cosa.

from tensorflow.keras.models import Model, load_model: nos permitirá cargar modelos entrenados y seguir entrenándolos, lo cual mejora al programa.

from tensorflow.keras.initializers import glorot_uniform: lo usamos para mejorar el rendimiento y acelerar el entrenamiento.

from tensorflow.keras.utils import plot_model; con esto obtendremos una visualización de como funciona el programa y sus capas

from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, LearningRateScheduler: será útil para el entrenamiento del algoritmo, utilizando devoluciones de llamada.

from IPython.display import display: la utilizaremos para ver los resultados intermedios y finales.

from tensorflow.keras import backend as K: lo utilizaremos para hacer los cálculos con las matrices.

from sklearn.preprocessing import StandardScaler, normalize: para mejorar el rendimiento del algoritmo.

import os: esto permitirá la correcta interacción entre el programa y el sistema operativo, nos permitirá acceder a directorios, por ejemplo.

import glob: nos permitirá acceder a los nombres de ruta que coincidan con lo especificado.

import random: este nos permitirá generar un patrón aleatorio para que el programa aprenda que hacer y no una instrucción en específico.

import plotly.graph_objects as go: es un modulo de ploty que nos permitirá visualizar un gráfico.

import nbformat: este lo utilizamos porque nos brinda diferentes formatos de cuaderno.

Luego a continuación usamos:

brain_df = pd.read_csv('dataset/data_mask.csv') y accedemos al dataset o mejor dicho en pocas palabras accedemos al directorio o carpeta contenedora, la carpeta con todas las imágenes de los tumores y no tumores.

Luego con: dataset_folder = os.path.abspath('dataset'), accedemos a la ruta absoluta. Es la ruta desde la raíz.

Creamos dos listas para almacenar las rutas de las imágenes de las resonancias.

image_paths = []

mask_paths = []

```
# Construir las rutas de imágenes y máscaras para todas las posiciones
```

```
for index in range(len(brain_df)):
```

```
    image_path = os.path.join(dataset_folder,  
brain_df.image_path[index])
```

```
    mask_path = os.path.join(dataset_folder,  
brain_df.mask_path[index])
```

```
    image_paths.append(image_path)
```

```
    mask_paths.append(mask_path)
```

en pocas palabras y de forma sencilla la imagen pasa a convertirse en datos binarios y esos datos se comparan con otros que también están en binario. Ceros y unos, ejemplo 01 es 1, 10 es 2 y 11 es 3. Con el sistema de numeración binario se procesan los datos.

Ahora image_paths y mask_paths contienen las rutas de imágenes y máscaras para todas las posiciones

```
print(image_paths)
```

```
print(mask_paths)
```

en pocas palabras esos print nos muestran la ruta donde están los archivos que en este caso son imágenes.

Ejemplo

Disco local c\argentina\programa\etc

Ahora utilizamos:

```
brain_df.info()
```

en pocas palabras esto obtiene la información de las imágenes que anteriormente pasamos al sistema de numeración binario para poder obtener la información y clasificarla. Accede al directorio o la carpeta para obtener las imágenes en las dos listas que creamos.

Ahora con...

`brain_df.head(50)`

podemos ver las primeras 50 filas.

Ahora con...

`brain_df.mask_path[1]`

accedemos a la segunda fila. Se recuerda que la fila empieza en el 0.

Ahora con `brain_df.image_path[1]` accedemos a la segunda fila.

Luego # Obtenemos la cantidad de pacientes con y sin tumores

`brain_df['mask'].value_counts()`

en pocas palabras y de forma sencilla cuenta las columnas.

A continuación, ponemos:

`brain_df`

esto nos dará permitirá ver los datos que estamos manipulando por pantalla, si aparecen menos de los que estamos usando o mas lo sabremos gracias a esta función.

Luego ponemos `brain_df['mask'].value_counts().index`. nos dira si es o no una imagen con tumor en binario. 0 y 1.

A continuación, usando plotly creamos un grafico de barras que nos muestre como están distribuidos los datos.

```
fig = go.Figure([go.Bar(x =
brain_df['mask'].value_counts().index, y =
brain_df['mask'].value_counts())])
en este creamos las variables x e y
```

```
fig.update_traces(marker_color = 'rgb(0,200,0)',
marker_line_color = 'rgb(0,255,0)',
marker_line_width = 7, opacity = 0.6)
```

hasta acá damos los datos y características del gráfico.

```
fig.show()
```

el ultimo lo mostramos.

Luego buscamos la ubicación con.

```
brain_df.mask_path
```

luego buscamos la ubicación y nombre original con.

```
brain_df.image_path.
```

Luego visualizamos la imagen binaria que usamos para resaltar un área determinada.

```
plt.imshow(cv2.imread(mask_paths[623]))
```

luego con plt.imshow(cv2.imread(image_paths[623])) visualizamos la imagen original, sin la comparación en binario.

Luego.

obtenemos el máximo valor de píxel de la imagen

```
cv2.imread(mask_paths[623]).max()
```

en pocas palabras en la imagen busca que áreas tienen más intenso su color, no es lo mismo un naranja con una intensidad de 12 a uno con una intensidad de 23.

Luego

```
# obtenemos el mínimo valor de píxel de la imagen
```

```
cv2.imread(mask_paths[623]).min()
```

este hace lo contrario a lo anterior.

Luego.

```
# Visualizamos la resonancia magnética, la máscara, y ambas de manera superpuesta
```

```
# Posición del DATASET
```

```
i = 623
```

```
if brain_df['mask'][i] == 1:
```

```
    fig, axs = plt.subplots(1, 3, figsize=(20, 5)) # Solo una fila con tres columnas
```

```
img = io.imread(image_paths[i])
axs[0].title.set_text('MRI del Cerebro')
axs[0].imshow(img)
```

```
mask = io.imread(mask_paths[i])
axs[1].title.set_text('Máscara')
axs[1].imshow(mask, cmap='gray')
```

```
img[mask == 255] = (255, 0, 0)
axs[2].title.set_text('MRI con Máscara')
axs[2].imshow(img)
```

```
plt.tight_layout()
plt.show()
```

en pocas palabras hacemos lo que ya hicimos antes, mostrar lo que ya hicimos con una visualización.

Ahora con la siguiente línea de código visualizamos de manera aleatoria 6 imágenes y las analizamos para ver que el programa funciona hasta el momento de formas correcta.

```
# Visualización básica de imágenes (MRI y Máscaras) en el
dataset de forma separada de manera aleatoria
fig, axs = plt.subplots(6,2, figsize=(16,32))
```

```

count = 0
for x in range(6):
    i = random.randint(0, len(brain_df)) # Seleccionamos un índice aleatorio
    axs[count][0].title.set_text("MRI del Cerebro") # Configuramos el título
    axs[count][0].imshow(cv2.imread(image_paths[i])) # Mostramos la MRI
    axs[count][1].title.set_text("Máscara - " + str(brain_df['mask'][i])) # Colocámos el título en la máscara (0 o 1)
    axs[count][1].imshow(cv2.imread(mask_paths[i])) # Mostramos la máscara correspondiente
    count += 1

fig.tight_layout()

```

para cerciorar que todo este bien ahora lo hacemos las tres etapas.

```

count = 0
fig, axs = plt.subplots(12, 3, figsize = (20, 50))
for i in range(len(brain_df)):
    if brain_df['mask'][i] == 1 and count < 12:
        img = io.imread(image_paths[i])
        axs[count][0].title.set_text('MRI del Cerebro')
        axs[count][0].imshow(img)
        count += 1

```

```
mask = io.imread(mask_paths[i])  
axs[count][1].title.set_text('Máscara')  
axs[count][1].imshow(mask, cmap = 'gray')
```

```
img[mask == 255] = (255, 0, 0)  
axs[count][2].title.set_text('MRI con Máscara')  
axs[count][2].imshow(img)  
count+=1
```

```
fig.tight_layout()
```

el código anterior mostraría la imagen sin procesar, el proceso y la imagen final.

Lo anterior muestra 12 imágenes sin procesar, 12 procesos y 12 procesados.

Me detendré acá y explicare la función de cada función utilizada.

1. FROM: se utiliza para la importación de librerías.
2. IMPORT: al igual que from también se usa para la importación de librerías, el orden en que se usa varia, pero se utilizan ambos.
3. READ_CSV: esto pertenece a la librería de pandas, se utiliza para leer y manipular datos. Gracias a esta función

pandas lee un archivo en una copia y lo manipula sin hacer uso del original.

4. PANDAS: es una librería de análisis y manipulación de datos o información
5. OS.PATH.ABSPATH('DATASET'): esta función se utiliza para obtener la ruta absoluta de la carpeta o directorio dataset donde se guarda la información que se necesita.
6. IMAGE_PATHS = []: esto es una lista y se utiliza para almacenar datos de forma ordenada con índices.
7. FOR INDEX IN RANGE(LEN: el bucle for es utilizado para que se repita el código un numero finito de veces. En pocas palabras es una secuencia repetitiva.
8. OS.PATH.JOIN(): sirve para unir varias rutas de diferentes directorios.
9. PRINT(IMAGE_PATHS): el print se usa para mostrar las salidas por pantalla.
10. BRAIN_DF.INFO(): esto es una función de pandas y nos permite obtener su información, con su me refiero a los datos a manipular.
11. BRAIN_DF.HEAD(50): head nos permite imprimir un numero determinado de filas, en este caso 50.
12. BRAIN_DF.MASK_PATH[1]: esto devuelve el nombre del archivo de mascara.
13. ['MASK']: con esto accedemos a las columnas.

14. **VALUE_COUNTS()**: lo utilizamos para contar los datos en pandas. Específicamente cuantas veces aparece un dato.
15. **SHOW()**: muestra los datos por pantalla.
16. **MARKER_LINE_COLOR**: se utiliza para dar color a los gráficos en plotly.
17. **PLOTLY**: se utiliza para visualizar cosas, específicamente gráficos.
18. **GO.BAR**: se utiliza para crear un grafico de barras en plotly.
19. **GO.FIGURE**: se utiliza para crear un objeto para los gráficos.
20. **CV2.IMREA**. pertenece a cv2 y sirve para leer un archivo.
21. **MASK_PATHS**: esto nos permite acceder al índice.
22. **CV2**: Con cv2, puedes leer y escribir imágenes, manipular imágenes (por ejemplo, cambiar el tamaño, rotar, recortar), aplicar filtros y transformaciones, detectar y reconocer objetos y características en imágenes entre otras cosas.
23. **MAX**: esto nos permite acceder a los máximos valores de un color en una parte de una imagen, su intensidad.
24. **MIN**: esto hace lo contrario a lo anterior.
25. **TITLE**: nos permite poner titulo a los gráficos.

26. PLT.SUBPLOTS: subplots crea subtramas, por ejemplo. 1, 1.2, 1.3, etc.
27. CMAP: pertenece a la librería de matplotlib y se utiliza para darle color al gráfico.
28. MATPLOTLIB: es una librería de visualización de datos.
29. IMSHOW(): Esta es una función que muestra los datos en un formato de imagen 2D.
30. TIGHT_LAYOUT(): ajusta el espacio entre el grafico y subgraficos.
31. RANDOM.RANDINT: esta función se utiliza para usar una variable aleatoria.
32. SHAPE: indica la cantidad de filas y columnas, pertenece a numpy.
33. NUMPY: esta es una librería de Python que se utiliza para una mayor precisión, por ejemplo, esta librería cuenta con la función pi, la cual cuenta con una mayor precisión que si pusiéramos 3.14.
34. DROP: se utiliza para eliminar filas y columnas específicas.
35. COLUMNS: se utiliza con la función anterior y sirve para especificar las filas y columnas a eliminar.
36. APPLY: se utiliza para convertir números en letras.

37. TRAIN_TEST_SPLIT: sirve para dividir los datos en partes.
38. TEST_SIZE: especifica la cantidad a dividir o su tamaño.
39. FROM KERAS_PREPROCESSING.IMAGE
IMPORT IMAGEDATAGENERATOR: importamos el generador de imágenes de keras.

Ahora que ya entendemos algunos conceptos seguiremos explicando sobre el programa.

`brain_df.shape`

utilizamos esto para ver las filas y columnas.

Visualizamos que tenemos solo tres columnas ahora

`brain_df_train.shape`

Convertir los datos en la columna de máscara a formato de string, para usar el modo categórico en `flow_from_dataframe`

`brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))`

en pocas palabras entrega los datos al modelo de machine learning.

Obtenemos la información del dataframe

`brain_df_train.info()`

```
# Dividimos los datos para el entrenamiento y testing dejando un  
15% para testeos y 85% para entrenamiento
```

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(brain_df_train, test_size = 0.15)
```

con esto asignamos la cantidad de datos a entrenar y los datos para testear que todo esté bien.

Creamos el generador de imágenes

```
from keras.preprocessing.image import ImageDataGenerator
```

```
# Creamos un generador de datos que escala los datos de 0 a 1 y  
haga una división de validación de 0,15
```

```
datagen = ImageDataGenerator(rescale=1./255., validation_split =  
0.15)
```

en pocas palabras con el código anterior preparamos las imágenes para ser usadas.

```
# Creamos un generador de datos para el entrenamiento
```

```
train_generator=datagen.flow_from_dataframe(  
dataframe=train,  
directory='./dataset/',  
x_col='image_path',  
y_col='mask',
```

```
subset="training",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))
```

Creamos un generador de datos para la validación

```
valid_generator=datagen.flow_from_dataframe(
    dataframe=train,
    directory='./dataset/',
    x_col='image_path',
    y_col='mask',
    subset="validation",
    batch_size=16,
    shuffle=True,
    class_mode="categorical",
    target_size=(256,256))
```

prepara las imágenes de testeo para ver que todo esté bien con el testing.

Creamos un generador de datos para imágenes de prueba

```
test_datagen=ImageDataGenerator(rescale=1./255.)
```

```
test_generator=test_datagen.flow_from_dataframe(  
    dataframe=test,  
    directory='./dataset/',  
    x_col='image_path',  
    y_col='mask',  
    batch_size=16,  
    shuffle=False,  
    class_mode='categorical',  
    target_size=(256,256))
```

es para probar que todo esté bien.

```
# Obtenemos el modelo base de ResNet50 (red neuronal  
entrenada)  
basemodel = ResNet50(weights = 'imagenet', include_top = False,  
input_tensor = Input(shape=(256, 256, 3)))  
estamos utilizando un modelo de aprendizaje automático.
```

```
# Obtenemos el resumen del modelo base con el detalle de  
parámetros  
basemodel.summary()  
con esto obtenemos una vista en general y  
chequeamos que todo esté bien.
```

Congelamos los pesos del modelo

```
for layer in basemodel.layers:
```

```
    layers.trainable = False
```

lo usamos para evitar que olvide lo aprendido o modifique su estructura.

Agregamos una cabecera de clasificación al modelo base

```
headmodel = basemodel.output
```

```
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
```

```
headmodel = Flatten(name= 'flatten')(headmodel)
```

```
headmodel = Dense(256, activation = "relu")(headmodel)
```

```
headmodel = Dropout(0.3)(headmodel)
```

```
headmodel = Dense(256, activation = "relu")(headmodel)
```

```
headmodel = Dropout(0.3)(headmodel)
```

```
headmodel = Dense(2, activation = 'softmax')(headmodel)
```

```
model = Model(inputs = basemodel.input, outputs = headmodel)
```

en pocas palabras agregamos algunas capas al modelo.

Visualizamos los parámetros del modelo

```
model.summary()
```

chequemos que todo este bien.

Compilamos el modelo

```
model.compile(loss = 'categorical_crossentropy',  
optimizer='adam', metrics= ["accuracy"])
```

preparamos el modo en lo anterior.

Utilizamos la parada temprana para salir del entrenamiento si la pérdida en la validación no disminuye incluso después de ciertas épocas (pacienza)

```
earlystopping = EarlyStopping(monitor='val_loss', mode='min',  
verbose=1, patience=5)
```

Guardamos el mejor modelo con la menor pérdida de validación

```
checkpointer = ModelCheckpoint(filepath="classifier-resnet-  
weights.hdf5", verbose=1, save_best_only=True)
```

mejoramos el entrenamiento del modelo y aseguráramos de que se guarda una copia del mejor modelo.

Generamos el entrenamiento

```
history = model.fit(train_generator, steps_per_epoch=  
train_generator.n // 16, epochs = 1, validation_data=  
valid_generator, validation_steps= valid_generator.n // 16,  
callbacks=[checkpointer, earlystopping])
```

entrenamos el modelo.

Guardamos la arquitectura del modelo entrenado en un archivo json

```
model_json = model.to_json()
with open("classifier-resnet-model.json","w") as json_file:
    json_file.write(model_json)
esto nos permite utilizar el modelo en otra ocasión.
```

Cambiamos la arquitectura de la red agregando una capa y un Dropouts más a fin de mejorar el rendimiento

```
headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
# Agregamos una capa más de 256 con la misma rectificadora
lineal unitaria
headmodel = Dense(256, activation = "relu")(headmodel)
# Agregamos una capa más de Dropout con el mismo valor
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)
```

```
# Este procedimiento permite generar una nueva versión del modelo.
```

Esto optimiza el modelo. cambiando la arquitectura de la red neuronal agregando una capa adicional y un Dropout más al final del modelo base. Esto se hace con el objetivo de mejorar el rendimiento del modelo

```
# Visualizamos los nuevos parámetros del modelo
```

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

volvemos a compilar el modelo

```
model.compile(loss = 'categorical_crossentropy',  
optimizer='adam', metrics= ["accuracy"])
```

```
# Utilizamos la parada temprana para salir del entrenamiento si la pérdida en la validación no disminuye incluso después de ciertas épocas (pacienza)
```

```
earlystopping = EarlyStopping(monitor='val_loss', mode='min',  
verbose=1, patience=5)
```

```
# Guardamos el mejor modelo con la menor pérdida de validación
```

```
checkpointer = ModelCheckpoint(filepath="classifier-resnet-  
weights.hdf5", verbose=1, save_best_only=True)
```

```
# Utilizamos la parada temprana para salir del entrenamiento si la  
pérdida en la validación no disminuye incluso después de ciertas  
épocas (pacienza)
```

```
earlystopping = EarlyStopping(monitor='val_loss', mode='min',  
verbose=1, patience=5)
```

```
# Guardamos el mejor modelo con la menor pérdida de validación
```

```
checkpointer = ModelCheckpoint(filepath="classifier-resnet-  
weights.hdf5", verbose=1, save_best_only=True)
```

```
history = model.fit(train_generator, steps_per_epoch=  
train_generator.n // 16, epochs = 1, validation_data=  
valid_generator, validation_steps= valid_generator.n // 16,  
callbacks=[checkpointer, earlystopping])
```

volvemos a entrenar el modelo pero con las nuevas capas y Dropout.

```
# Volvemos a guardar la arquitectura del modelo entrenado en un  
archivo json
```

```
model_json = model.to_json()  
with open("classifier-resnet-model.json","w") as json_file:  
    json_file.write(model_json)
```

Cargamos el modelo pre entrenado generado en el JSON “resnet-50-MRI.json” (MODELO ENTRENADO GUARDADO)

```
with open('resnet-50-MRI.json', 'r') as json_file:
```

```
    json_savedModel= json_file.read()
```

```
# Cargamos el modelo pre entrenado generado en el JSON  
“classifier-resnet-model.json” (MODELO ENTRENADO)
```

```
#with open('classifier-resnet-model.json', 'r') as json_file:
```

```
#    json_savedModel= json_file.read()
```

```
# Cargamos el modelo pre entrenado generado en el JSON  
“weights.hdf5”
```

```
model = tf.keras.models.model_from_json(json_savedModel)
```

```
model.load_weights('weights.hdf5')
```

```
model.compile(loss = 'categorical_crossentropy',  
optimizer='adam', metrics= ["accuracy"])
```

en pocas palabras carga el modelo pre entrenado.

```
# Hacemos la predicción. Esta se realiza sobre el 15% de  
imágenes que se dejaron para testeos, no con las que se usaron  
para entrenar. Las peticiones las realizamos en lotes de 16  
imágenes cada uno.
```

```
test_predict = model.predict(test_generator, steps =  
test_generator.n // 16, verbose =1).
```

Testeamos que todo este bien y no haya error. Para eso dejamos el 15%

De las imágenes chequeamos que sean dos salidas (tiene o no tiene tumor)

`test_predict.shape`

Visualizamos un array con la probabilidad de que no contenga tumor en la primera columna y la probabilidad de que tenga en la segunda. Para cada imagen se genera un array con los dos valores

`test_predict`

Obtenemos la clase predicha a partir del modelo

con argmax podemos ver la probabilidad máxima de cada fila, donde si en la posición 0 indica que probablemente no contenga un tumor, la posición 1 que si lo contenga.

`predict = []`

```
for i in test_predict:  
    predict.append(str(np.argmax(i)))
```

`predict = np.asarray(predict)`

esto hace una predicción de si tiene o no tumor el paciente.

Ahora visualizamos el array con los valores de probabilidad

Predict

Nos muestra en binario si tiene o no tumor

Dado que usamos el generador de prueba, se limita el número de imágenes a leer (predecir), debido al tamaño del lote

```
original = np.asarray(test['mask'])[:len(predict)]  
len(original)
```

Una vez realizado esto, podemos obtener la tasa de acierto del modelo con el set de datos de testeos

```
from sklearn.metrics import accuracy_score
```

```
accuracy = accuracy_score(original, predict)
```

accuracy

compara los datos originales con los de predicción para ver cuánto es que acierta el modelo y ver la tasa de aciertos.

La red neuronal tiene un porcentaje de efectividad del 98,61% sin aplicar la matriz de confusión.

Representamos la matriz de confusión que nos permite identificar los falsos positivos y falsos negativos

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(original, predict)  
plt.figure(figsize = (7,7))  
sns.heatmap(cm, annot=True)
```

Con esto podemos ver que 4 muestras dieron un falso positivo, y 6 muestras dieron un falso negativo. Dando un total de 8 muestras erróneas

```
# Imprimimos el informe de clasificación y comenta sobre los resultados de precisión, recuperación y F1-Score (media entre la precisión y recall)
```

```
from sklearn.metrics import classification_report
```

```
report = classification_report(original, predict, labels = [0,1])  
print(report)
```

Obtenemos el dataframe que contiene las resonancias magnéticas que tienen máscaras asociadas.

```
brain_df_mask = brain_df[brain_df['mask'] == 1]
```

```
brain_df_mask.shape
```

en este fragmento de código abrimos o entramos al datased.

```
# Tenemos que hay 1373 muestras con máscaras asociadas.
```

Dividimos los datos en datos de prueba y de entrenamiento

```
from sklearn.model_selection import train_test_split
```

Dividimos en dos modelos para entrenar el 85% y para validar el 15%

```
X_train, X_val = train_test_split(brain_df_mask, test_size=0.15)
```

```
X_test, X_val = train_test_split(X_val, test_size=0.5)
```

Con lo anterior dividimos el 15 porciento de datos de validación para obtener datos de prueba.

Creamos una lista separada para imageId, classId para pasar al generador# Crear una lista separada para imageId, classId para pasar al generador

```
train_ids = list(X_train.image_path)
```

```
train_mask = list(X_train.mask_path)
```

```
val_ids = list(X_val.image_path)
```

```
val_mask = list(X_val.mask_path)
```

Construir rutas de imágenes y máscaras para entrenamiento y validación

```
data_folder = "./dataset/"
```

```
train_ids = [os.path.join(data_folder, filename) for filename in  
train_ids]
```

```
train_mask = [os.path.join(data_folder, filename) for filename in  
train_mask]
```

```
val_ids = [os.path.join(data_folder, filename) for filename in  
val_ids]
```

```
val_mask = [os.path.join(data_folder, filename) for filename in  
val_mask]
```

este código está preparando las rutas a las imágenes y máscaras para que puedan ser cargadas desde el disco y utilizadas para entrenar y validar el modelo

Importamos utilities para la función de pérdida personalizada y el generador de datos personalizados

```
from dataset.utilities import DataGenerator
```

```
# Creamos los generadores de imágenes para poder cargar  
dinámicamente
```

```
training_generator = DataGenerator(train_ids, train_mask)
```

```
validation_generator = DataGenerator(val_ids, val_mask)
```

este código está creando dos generadores de imágenes que te permiten cargar dinámicamente las imágenes y máscaras para entrenar y validar el modelo

```
# Creamos una función para los bloques residuales de la  
RESUNET
```

```
def resblock(X, f):
```

```
# Hacemos la copia de la entrada
```

```
X_copy = X
```

```
# Ruta principal
```

```
X = Conv2D(f, kernel_size = (1,1) ,strides =  
(1,1),kernel_initializer ='he_normal')(X)
```

```
X = BatchNormalization()(X)
```

```
X = Activation('relu')(X)
```

```
X = Conv2D(f, kernel_size = (3,3), strides =(1,1), padding =  
'same', kernel_initializer ='he_normal')(X)
```

```
X = BatchNormalization()(X)
```

```
# Ruta corta
```

```
X_copy = Conv2D(f, kernel_size = (1,1), strides =(1,1),
kernel_initializer ='he_normal')(X_copy)

X_copy = BatchNormalization()(X_copy)

# Agregamos la salida de la ruta principal y la ruta corta juntas

X = Add()([X,X_copy])

X = Activation('relu')(X)

return X
```

usamos esto para evitar el desvanecimiento, es común en las redes neuronales profundas, ayuda a prevenir el problema del desvanecimiento del gradiente

Creamos otra función para escalar y concatenar los valores pasados

```
def upsample_concat(x, skip):

    x = UpSampling2D((2,2))(x)

    merge = Concatenate()([x, skip])



return merge
```

unimos los datos.

Forma del tensor de entrada

```
X_input = Input(input_shape)
```

Fase 1

```
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same',  
kernel_initializer ='he_normal')(X_input)
```

```
conv1_in = BatchNormalization()(conv1_in)
```

```
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same',  
kernel_initializer ='he_normal')(conv1_in)
```

```
conv1_in = BatchNormalization()(conv1_in)
```

```
pool_1 = MaxPool2D(pool_size = (2,2))(conv1_in)
```

Fase 2

```
conv2_in = resblock(pool_1, 32)
```

```
pool_2 = MaxPool2D(pool_size = (2,2))(conv2_in)
```

Fase 3

```
conv3_in = resblock(pool_2, 64)
```

```
pool_3 = MaxPool2D(pool_size = (2,2))(conv3_in)
```

Fase 4

```
conv4_in = resblock(pool_3, 128)
```

```
pool_4 = MaxPool2D(pool_size = (2,2))(conv4_in)
```

Fase 5 (Cuello de Botella)

```
conv5_in = resblock(pool_4, 256)
```

```
# Fase de Escalada 1
```

```
up_1 = upsample_concat(conv5_in, conv4_in)
```

```
up_1 = resblock(up_1, 128)
```

```
# Fase de Escalada 2
```

```
up_2 = upsample_concat(up_1, conv3_in)
```

```
up_2 = resblock(up_2, 64)
```

```
# Fase de Escalada 3
```

```
up_3 = upsample_concat(up_2, conv2_in)
```

```
up_3 = resblock(up_3, 32)
```

```
# Fase de Escalada 4
```

```
up_4 = upsample_concat(up_3, conv1_in)
```

```
up_4 = resblock(up_4, 16)
```

```
# Salida Final
```

```
output = Conv2D(1, (1,1), padding = "same", activation =  
"sigmoid")(up_4)
```

```
model_seg = Model(inputs = X_input, outputs = output )
```

implementaremos una red neuronal.

Imprimimos el resumen del modelo de segmentación y enumeramos el número total de parámetros entrenables

```
model_seg.summary()
```

Importamos de utilidades el código para la función de pérdida personalizada y el generador de datos personalizados a nivel de pixel.

```
from dataset.utilities import focal_tversky, tversky_loss, tversky
```

Estas funciones se utilizan para calcular una métrica de pérdida personalizada para entrenar un modelo de aprendizaje

Este código compila un modelo de aprendizaje automático utilizando el optimizador Adam. El optimizador Adam es un método de optimización utilizado para entrenar redes neuronales profundas

```
# Usamos la parada temprana para salir del entrenamiento si la pérdida de validación no disminuye incluso después de ciertas épocas (pacienza)
```

```
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
```

Guardamos el mejor modelo con menor pérdida de validación

```
checkpointer = ModelCheckpoint(filepath="ResUNet-weights.hdf5", verbose=1, save_best_only=True)
```

Generamos el entrenamiento de la segmentación de la arquitectura

```
history = model_seg.fit(training_generator , epochs = 1,  
validation_data = validation_generator, callbacks = [checkpointer,  
earlystopping])
```

entrenamos el modelo.

Guardamos la arquitectura del modelo en un archivo json

```
model_json = model_seg.to_json()  
with open("ResUNet-model.json","w") as json_file:  
    json_file.write(model_json)  
con esta parte guardamos el modelo generado.
```

Importamos de utilidades las funciones necesarias tversky

```
from dataset.utilities import focal_tversky, tversky_loss, tversky
```

```
# Cargamos el modelo pre entrenado generado en el JSON  
“ResUNet-MRI.json” (MODELO ENTRENADO GUARDADO)
```

```
with open('ResUNet-MRI.json', 'r') as json_file:
```

```
    json_savedModel= json_file.read()
```

```
# Cargamos el modelo pre entrenado generado en el JSON  
“ResUNet-model.json” (MODELO ENTRENADO)
```

```
#with open('ResUNet-model.json', 'r') as json_file:
```

```
#    json_savedModel= json_file.read()
```

```
# cargamos la arquitectura del modelo
```

```
model_seg =
```

```
tf.keras.models.model_from_json(json_savedModel)
```

```
model_seg.load_weights('weights_seg.hdf5')
```

```
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
```

```
model_seg.compile(optimizer = adam, loss = focal_tversky,  
metrics = [tversky])
```

```
# Importamos de utilidades el código para la función de pérdida  
personalizada y el generador de datos personalizados
```

```
from dataset.utilities import prediction
```

```
# Hacemos la predicción
```

```
# Se pasa el primer modelo, si da negativo pasa, si da positivo se  
pasa al segundo modelo para proceder a generar la máscara de  
segmentación
```

```
image_id, mask, has_mask = prediction(test, model, model_seg)
```

Creamos el dataframe para el resultado

```
df_pred = pd.DataFrame({'image_path':  
image_id,'predicted_mask': mask,'has_mask': has_mask})
```

```
df_pred
```

en pocas palabras creamos un dtased

Fusionamos el dataframe que contiene los resultados previstos con los datos de prueba originales.

```
df_pred = test.merge(df_pred, on = 'image_path')
```

```
df_pred.head()
```

es decir mesclamos los datos que ya teníamos con los nuevos.

```
# Generamos un gráfico con 10 muestras distintas con tumores
```

```
count = 0
```

```
fig, axs = plt.subplots(10, 5, figsize=(30, 50))
```

```
for i in range(len(df_pred)):
```

```
    if df_pred['has_mask'][i] == 1 and count < 10:
```

```
        # Construir rutas absolutas a las imágenes y máscaras  
        # utilizando la carpeta "dataset"
```

```
        img_path = os.path.join('dataset', df_pred.image_path[i])
```

```
        mask_path = os.path.join('dataset', df_pred.mask_path[i])
```

```
# Leer las imágenes y convertirlas a formato RGB
```

```
img = io.imread(img_path)
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
axs[count][0].title.set_text("MRI del Cerebro")
```

```
axs[count][0].imshow(img)
```

```
# Obtener la máscara para la imagen
```

```
mask = io.imread(mask_path)
```

```
axs[count][1].title.set_text("Máscara Original")
```

```
axs[count][1].imshow(mask)
```

```
# Obtenemos la máscara de predicción para dicha imagen
```

```
predicted_mask =
```

```
np.asarray(df_pred.predicted_mask[i])[0].squeeze().round()
```

```
axs[count][2].title.set_text("Máscara predicha por la IA")
```

```
axs[count][2].imshow(predicted_mask)

# Aplicamos la máscara a la imagen 'mask==255'
img[mask == 255] = (255, 0, 0)
axs[count][3].title.set_text("MRI con la máscara original
(Ground Truth)")
axs[count][3].imshow(img)

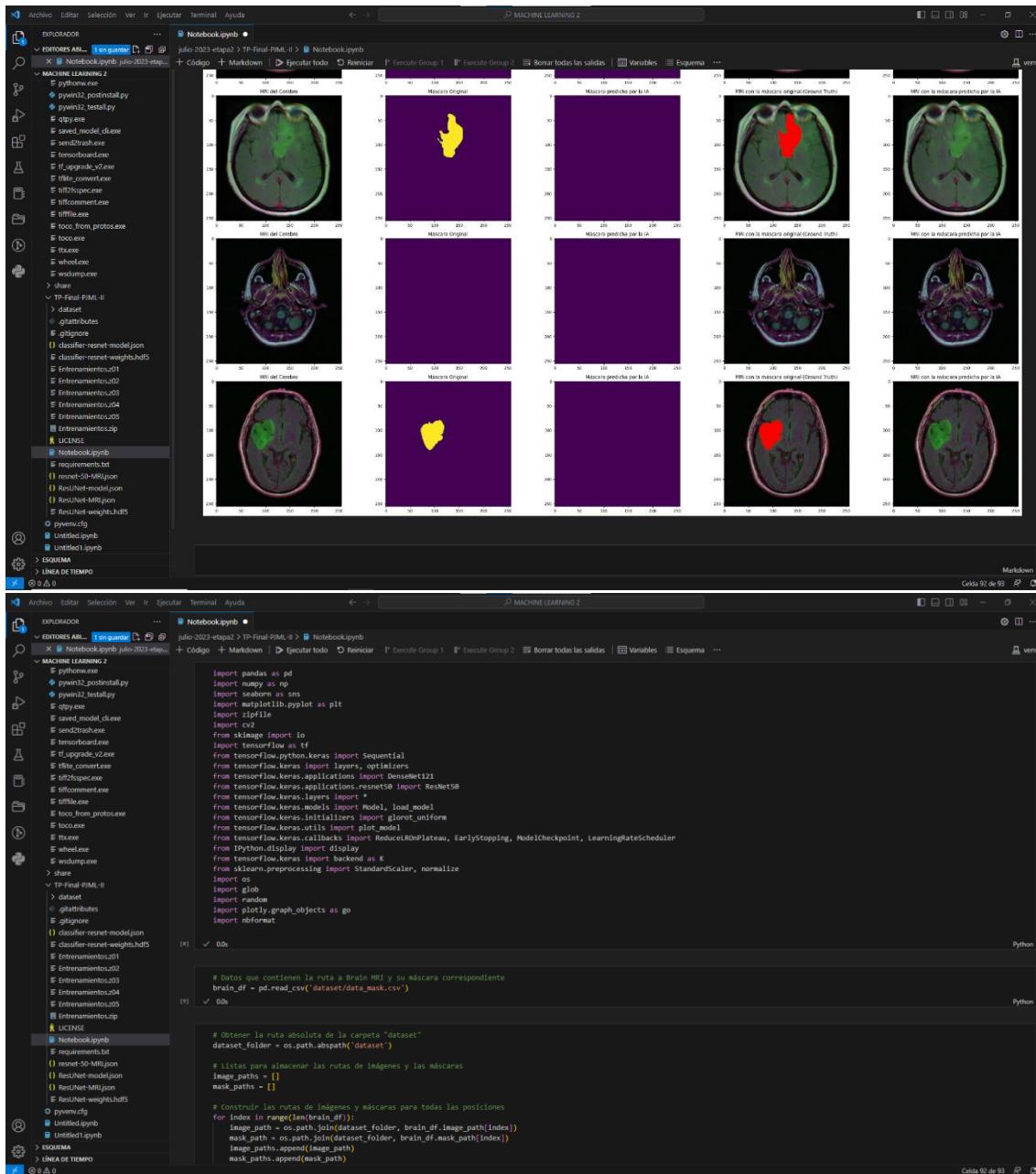
img_ = io.imread(img_path)
img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
img_[predicted_mask == 1] = (0, 255, 0)
axs[count][4].title.set_text("MRI con la máscara predicha
por la IA")
axs[count][4].imshow(img_)

count += 1

fig.tight_layout()

Creamos una muestra del software y lo que hace.
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez



Archivo Editor Selección Ver Ir Ejecutar Terminal Ayuda

EXPLORADOR Notebook.ipynb

- julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb
- MACHINE LEARNING 2
 - pywin32_postinstall.py
 - pywin32_Install.py
 - qtpyv4
 - saved_model_disease
 - send2trash.exe
 - tf_upgrade_v2.exe
 - tf_upgrade_v2.exe
 - tfmsc_convertexe
 - tiff2spec.exe
 - tiffcomment.exe
 - tiffile.exe
 - tococo
 - txxece
 - wheel.exe
 - wheelcomplete
 - share
- TP-Final-PIML-II
 - dataset
 - .gitattributes
 - gitignore
 - classifier-reset-model.json
 - classifier-reset-weights.hdf5
 - Entrenamiento.c01
 - Entrenamiento.c02
 - Entrenamiento.c03
 - Entrenamiento.c04
 - Entrenamiento.c05
 - Entrenamiento.zip
 - LICENCIA
 - Notebook.ipynb
 - requirements.txt
 - (1) requirements.txt
 - (1) requirements-00.json
 - (1) ResNet-mod4.json
 - (1) ResNet-mod4.json
 - (1) ResNet-MRI.json
 - (1) ResNet-MRI.json
 - (1) ResNet-weights.hdf5
 - pyenv.cfg
 - Untitled.ipynb
 - Untitled.ipynb
- > ESQUEMA
- > LÍNEA DE TIEMPO

Notbook.ipynb

julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb

+ Código + Markdown ▶ Ejecutar todo ⌂ Reiniciar ⌂ Ejecutar Grupo 1 ⌂ Ejecutar Grupo 2 ⌂ Borrar todas las salidas ⌂ Variables ⌂ Esquema ⌂ ver

```
# Obtenemos la información de los datos
brain_df.info()
```

	patient_id	image_path	mask_path	mask
0	TGCA_CS_5395_19981004	TGCA_CS_5395_19981004_1.tif	TGCA_CS_5395_19981004_1.tif	0
1	TGCA_CS_5395_19981004	TGCA_CS_4944_20010208/TGCA_CS_4944_20010208_1.tif	TGCA_CS_4944_20010208/TGCA_CS_4944_20010208_1.tif	0
2	TGCA_CS_5395_19981004	TGCA_CS_4941_19960909/TGCA_CS_4941_19960909_1.tif	TGCA_CS_4941_19960909/TGCA_CS_4941_19960909_1.tif	0
3	TGCA_CS_5395_19981004	TGCA_CS_4943_20000902/TGCA_CS_4943_20000902_1.tif	TGCA_CS_4943_20000902/TGCA_CS_4943_20000902_1.tif	0
4	TGCA_CS_5395_19981004	TGCA_CS_5396_20010302/TGCA_CS_5396_20010302_1.tif	TGCA_CS_5396_20010302/TGCA_CS_5396_20010302_1.tif	0
5	TGCA_CS_5395_19981004	TGCA_CS_5393_19990606/TGCA_CS_5393_19990606_1.tif	TGCA_CS_5393_19990606/TGCA_CS_5393_19990606_1.tif	0
6	TGCA_CS_5395_19981004	TGCA_CS_4942_19970222/TGCA_CS_4942_19970222_1.tif	TGCA_CS_4942_19970222/TGCA_CS_4942_19970222_1.tif	0
7	TGCA_CS_5395_19981004	TGCA_CS_5397_20010115/TGCA_CS_5397_20010115_1.tif	TGCA_CS_5397_20010115/TGCA_CS_5397_20010115_1.tif	0
8	TGCA_CS_5395_19981004	TGCA_CS_6188_20010812/TGCA_CS_6188_20010812_1.tif	TGCA_CS_6188_20010812/TGCA_CS_6188_20010812_1.tif	0
9	TGCA_CS_5395_19981004	TGCA_CS_6666_20011109/TGCA_CS_6666_20011109_1.tif	TGCA_CS_6666_20011109/TGCA_CS_6666_20011109_1.tif	0
10	TGCA_CS_5395_19981004	TGCA_CS_6669_20020102/TGCA_CS_6669_20020102_1.tif	TGCA_CS_6669_20020102/TGCA_CS_6669_20020102_1.tif	0
11	TGCA_CS_5395_19981004	TGCA_CS_6186_20000601/TGCA_CS_6186_20000601_1.tif	TGCA_CS_6186_20000601/TGCA_CS_6186_20000601_1.tif	0
12	TGCA_CS_5395_19981004	TGCA_DL_5851_19950408/TGCA_DL_5851_19950408_1.tif	TGCA_DL_5851_19950408/TGCA_DL_5851_19950408_1.tif	0
13	TGCA_CS_5395_19981004	TGCA_CS_6665_20010817/TGCA_CS_6665_20010817_1.tif	TGCA_CS_6665_20010817/TGCA_CS_6665_20010817_1.tif	0
14	TGCA_CS_5395_19981004	TGCA_CS_6668_20011025/TGCA_CS_6668_20011025_1.tif	TGCA_CS_6668_20011025/TGCA_CS_6668_20011025_1.tif	0

Celda 92 de 93

EXPLORADOR Notebook.ipynb

- julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb
- MACHINE LEARNING 2
 - pywin32_postinstall.py
 - pywin32_Install.py
 - qtpyv4
 - saved_model_disease
 - send2trash.exe
 - tf_upgrade_v2.exe
 - tf_upgrade_v2.exe
 - tfmsc_convertexe
 - tiff2spec.exe
 - tiffcomment.exe
 - tiffile.exe
 - tococo
 - txxece
 - wheel.exe
 - wheelcomplete
 - share
- TP-Final-PIML-II
 - dataset
 - .gitattributes
 - gitignore
 - classifier-reset-model.json
 - classifier-reset-weights.hdf5
 - Entrenamiento.c01
 - Entrenamiento.c02
 - Entrenamiento.c03
 - Entrenamiento.c04
 - Entrenamiento.c05
 - Entrenamiento.zip
 - LICENCIA
 - Notebook.ipynb
 - requirements.txt
 - (1) requirements-00.json
 - (1) ResNet-mod4.json
 - (1) ResNet-MRI.json
 - (1) ResNet-weights.hdf5
 - pyenv.cfg
 - Untitled.ipynb
 - Untitled.ipynb
- > ESQUEMA
- > LÍNEA DE TIEMPO

Notbook.ipynb

julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb

+ Código + Markdown ▶ Ejecutar todo ⌂ Reiniciar ⌂ Ejecutar Grupo 1 ⌂ Ejecutar Grupo 2 ⌂ Borrar todas las salidas ⌂ Variables ⌂ Esquema ⌂ ver

```
# Ruta a la imagen de la MRI en la posición [1]
brain_df.mask_path[1]
```

	patient_id	image_path	mask_path	mask
49	TGCA_CS_5395_19960909	TGCA_DL_AS TY_19970709/TGCA_DL_AS TY_19970709_1.tif	TGCA_DL_AS TY_19970709/TGCA_DL_AS TY_19970709_1.tif	0

Ruta a la máscara de segmentación en la posición [1]
brain_df.image_path[1]

	patient_id	image_path	mask_path	mask
49	TGCA_CS_5395_19960909	TGCA_CS_4944_20010208_1.tif	TGCA_CS_4944_20010208_1.tif	0

Obtenemos la cantidad de pacientes con y sin tumores
brain_df['mask'].value_counts()

mask	count	dtype
0	2556	int64
1	1373	int64

Celda 92 de 93

EXPLORADOR Notebook.ipynb

- julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb
- MACHINE LEARNING 2
 - pywin32_postinstall.py
 - pywin32_Install.py
 - qtpyv4
 - saved_model_disease
 - send2trash.exe
 - tf_upgrade_v2.exe
 - tf_upgrade_v2.exe
 - tfmsc_convertexe
 - tiff2spec.exe
 - tiffcomment.exe
 - tiffile.exe
 - tococo
 - txxece
 - wheel.exe
 - wheelcomplete
 - share
- TP-Final-PIML-II
 - dataset
 - .gitattributes
 - gitignore
 - classifier-reset-model.json
 - classifier-reset-weights.hdf5
 - Entrenamiento.c01
 - Entrenamiento.c02
 - Entrenamiento.c03
 - Entrenamiento.c04
 - Entrenamiento.c05
 - Entrenamiento.zip
 - LICENCIA
 - Notebook.ipynb
 - requirements.txt
 - (1) requirements-00.json
 - (1) ResNet-mod4.json
 - (1) ResNet-MRI.json
 - (1) ResNet-weights.hdf5
 - pyenv.cfg
 - Untitled.ipynb
 - Untitled.ipynb
- > ESQUEMA
- > LÍNEA DE TIEMPO

Notbook.ipynb

julio-2023-estapa2 > TP-Final-PIML-II > Notebook.ipynb

+ Código + Markdown ▶ Ejecutar todo ⌂ Reiniciar ⌂ Ejecutar Grupo 1 ⌂ Ejecutar Grupo 2 ⌂ Borrar todas las salidas ⌂ Variables ⌂ Esquema ⌂ ver

```
# Visualizamos todos los datos
brain_df
```

	patient_id	image_path	mask_path	mask
0	TGCA_CS_5395_19981004	ICGA_CS_5395_19981004_1.tif	ICGA_CS_5395_19981004_1.tif	0
1	TGCA_CS_5395_19981004	TGCA_CS_4944_20010208/TGCA_CS_4944_20010208_1.tif	TGCA_CS_4944_20010208/TGCA_CS_4944_20010208_1.tif	0
2	TGCA_CS_5395_19981004	TGCA_CS_4941_19960909/TGCA_CS_4941_19960909_1.tif	TGCA_CS_4941_19960909/TGCA_CS_4941_19960909_1.tif	0
3	TGCA_CS_5395_19981004	TGCA_CS_4943_20000902/TGCA_CS_4943_20000902_1.tif	TGCA_CS_4943_20000902/TGCA_CS_4943_20000902_1.tif	0
4	TGCA_CS_5395_19981004	TGCA_CS_5396_20010302/TGCA_CS_5396_20010302_1.tif	TGCA_CS_5396_20010302/TGCA_CS_5396_20010302_1.tif	0

Celda 92 de 93

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

ENTRADAS

- ENTRADAS AML_1 [en ejecución]
- ENTRADAS AML_2 [en ejecución]
- MACHINE LEARNING 2
 - pythonw
 - pywin32_postinstall.py
 - pywin32_reinstall.py
 - qtconsole
 - saved_model_disease
 - send2trash.exe
 - tf_upgrade_v2.exe
 - tflite_convert.exe
 - tflite_convert.exe
 - tflite.exe
 - toco from protos.exe
 - tooco
 - txeexe
 - wheel.exe
 - wkhtmltopdf
 - share
- TP-Final-PIML-II
 - dataset
 - gdrive
 - gitignore
 - classifier-reset-model.json
 - classifier-reset-weights.hdf5
 - Entrenamiento\z01
 - Entrenamiento\z02
 - Entrenamiento\z03
 - Entrenamiento\z04
 - Entrenamiento\z05
 - Entrenamiento\z06
 - LICENSE
 - Notebook.ipynb
 - requirements.txt
 - requirements.json
 - ResNet-mod.json
 - ResNet-MRI.json
 - ResNet-weights.hdf5
 - pyenv.cfg
 - Untitled.ipynb
 - Untitled.ipynb
- ESQUEMA
- LÍNEA DE TIEMPO

EDAD

```
# Visualizamos los diversos valores que posee la máscara. Para este caso en particular, es un tipo binario donde 0 = No presenta tumores, 1 = Presenta tumores
brain_df['mask'].value_counts().index
```

[37] v 0.05

```
# Usamos plotly para hacer un diagrama de barras interactivo con la distribución de los datos
fig = go.Figure(go.Bar(x = brain_df['mask'].value_counts().index, y = brain_df['mask'].value_counts()))
fig.update_traces(marker_color = 'rgb(0,200,0)', marker_line_color = 'rgb(0,255,0)')
fig.show()
```

[38] v 0.4s

EDAD

```
# Obtenemos el path de cada máscara de cada imagen
brain_df['mask_path']
```

[39] v 0.05

```
# Obtenemos el path de cada máscara de cada imagen
brain_df['mask_path']
```

[39] v 0.05

```
0 TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1...
1 TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1...
2 TCGA_CS_4041_19960909/TCGA_CS_4041_19960909_1...
3 TCGA_CS_4043_20000902/TCGA_CS_4043_20000902_1...
4 TCGA_CS_4550_20100201/TCGA_CS_4550_20100201_1...
...
```

[39] v 0.05

```
# Obtenemos el path de cada imagen original
brain_df['image_path']
```

[39] v 0.05

```
0 TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif
1 TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif
2 TCGA_CS_4041_19960909/TCGA_CS_4041_19960909_1.tif
3 TCGA_CS_4043_20000902/TCGA_CS_4043_20000902_1.tif
4 TCGA_CS_4550_20100201/TCGA_CS_4550_20100201_1.tif
...
```

[39] v 0.05

```
# visualizamos una imagen de tipo máscara en una determinada posición para visualizar el tumor.
plt.imshow(cv2.imread(mask_paths[62]))
```

[40] v 0.4s

```
cmatplotlib.Image.AxesImage at 0x161d7218310
```

[40] v 0.4s

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Nota: Los siguientes capturas de pantalla muestran la ejecución de un notebook en Jupyter Notebook. Los resultados de las imágenes se visualizan directamente en el cuadro de código.

Captura 1: Visualización de una máscara y una imagen sin máscara.

```
# visualizamos una imagen de tipo Máscara en una determinada posición para visualizar el tumor.
plt.imshow(cv2.imread(mask_paths[621]))
```

```
# visualizamos una imagen en una determinada posición para visualizar sin la máscara.
plt.imshow(cv2.imread(image_paths[621]))
```

Captura 2: Obtención de los valores máximos y mínimos de la máscara.

```
# obtenemos el máximo valor de pixel de la imagen
cv2.imread(mask_paths[623]).max()
```

255

```
# obtenemos el mínimo valor de pixel de la imagen
cv2.imread(mask_paths[623]).min()
```

0

Captura 3: Visualización de la resonancia magnética, la máscara, y ambas de manera superpuesta.

```
# Visualizamos la resonancia magnética, la máscara, y ambas de manera superpuesta
# Posición del DATASET
i = 623

if brain_df['mask'][i] == 1:
    fig, axes = plt.subplots(1, 3, figsize=(20, 5)) # Solo una fila con tres columnas

    img = io.imread(image_paths[i])
    axes[0].title.set_text('MRI del Cerebro')
    axes[0].imshow(img)

    mask = io.imread(mask_paths[i])
    axes[1].title.set_text('Máscara')
    axes[1].imshow(mask, cmap='gray')

    imgMask = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgMask[imgMask == 255] = (255, 0, 0)
    axes[2].title.set_text('MRI con Máscara')
    axes[2].imshow(imgMask)

    plt.tight_layout()
    plt.show()
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

The screenshot shows two Jupyter Notebook environments. Both have the title 'MACHINE LEARNING 2' and are running the file 'Notebook.ipynb'. The left environment displays a Python code cell and two side-by-side plots. The first plot is titled 'MRI del Cerebro' and shows a grayscale MRI scan of a brain cross-section. The second plot is titled 'Máscara - 1' and shows a binary mask where the segmented region is white against a black background. The right environment shows similar results for a different brain slice, with the MRI labeled 'MRI del Cerebro' and the mask labeled 'Máscara - 1'.

```
# Visualización básica de imágenes (MRI y Máscaras) en el dataset de forma separada de manera aleatoria
fig, axes = plt.subplots(6,2, figsize=(16,32))
count = 0
for x in range(0):
    i = random.randint(0, len(brain_df)) # Seleccionamos un índice aleatorio
    axes[count][0].imshow(brain_df[i].image) # Configuramos el título
    axes[count][0].set_title("MRI del Cerebro")
    axes[count][1].imshow(brain_df[i].mask) # Colocamos el título en la máscara (0 o 1)
    axes[count][1].set_title("Máscara - 1")
    axes[count][1].imshow(cv2.imread(mask_paths[1])) # Mostramos la máscara correspondiente
    count += 1
fig.tight_layout()
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Two screenshots of a Jupyter Notebook interface showing MRI analysis results.

Screenshot 1:

```

count = 0
fig, axes = plt.subplots(12, 3, figsize = (18, 54))
for i in range(len(image_paths)):
    if train_data[i][1] == 1 and count < 12:
        img = io.imread(image_paths[i])
        axes[count][0].title.set_text("MRI del Cerebro")
        axes[count][0].imshow(img)

        mask = io.imread(mask_paths[i])
        axes[count][1].title.set_text("Máscara")
        axes[count][1].imshow(mask, cmap = "gray")

        img_with_mask = np.zeros((255, 255, 3))
        axes[count][2].title.set_text("MRI con Máscara")
        axes[count][2].imshow(img_with_mask)
        axes[count][2].set_color_cycle(['red', 'green', 'blue'])
        axes[count][2].imshow(img, alpha=0.5)
        axes[count][2].imshow(mask, alpha=0.5)

    count+=1

fig.tight_layout()

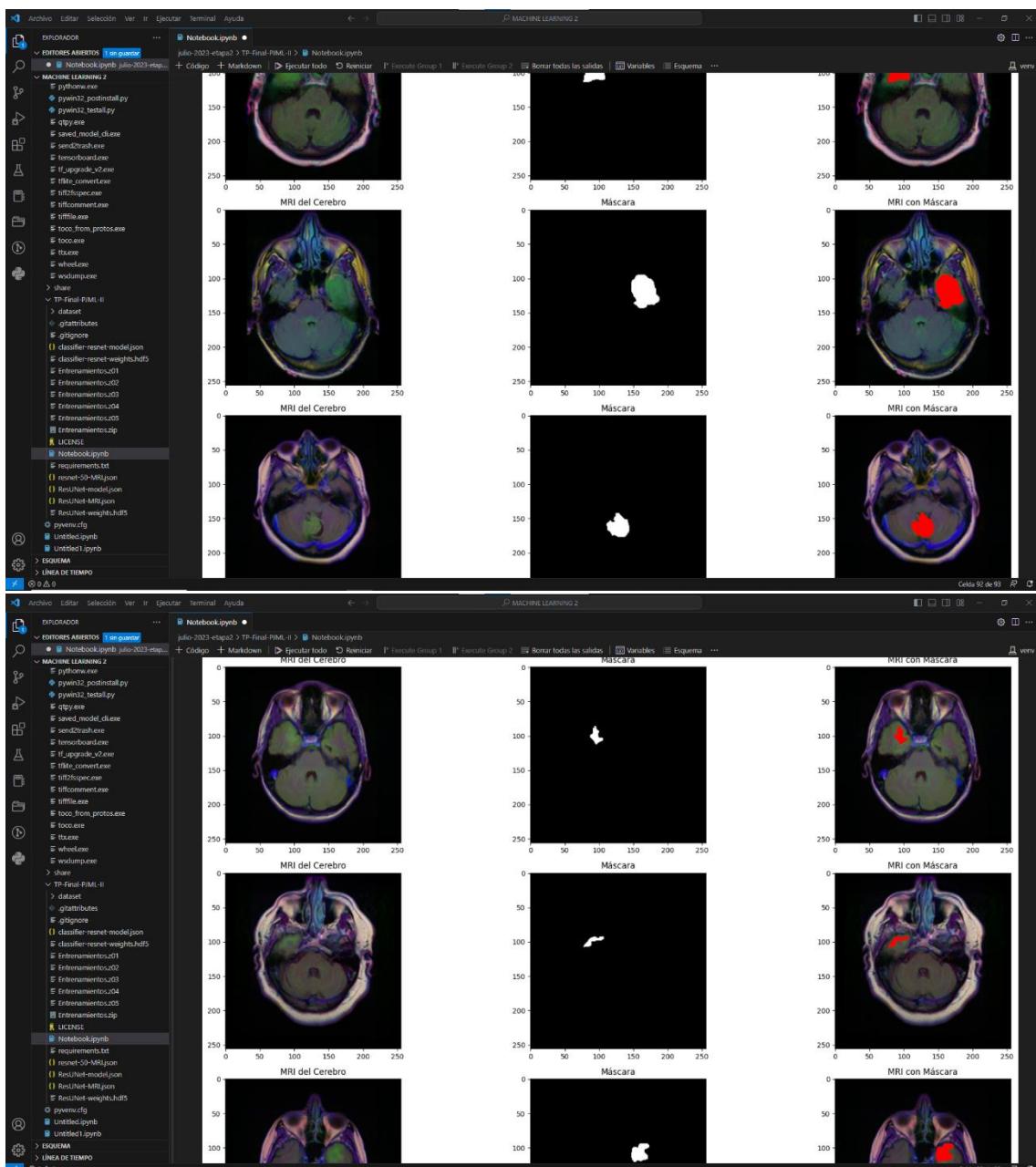
```

The notebook displays three rows of plots. Each row contains three subplots: "MRI del Cerebro" (colored brain scan), "Máscara" (binary mask), and "MRI con Máscara" (brain scan with mask overlaid). The first row shows the first 12 images from the dataset. A progress bar at the bottom indicates "Celda 92 de 93".

Screenshot 2:

This screenshot shows the same analysis for a different set of 12 images. The plots are identical: "MRI del Cerebro", "Máscara", and "MRI con Máscara". The first row shows the first 12 images from the dataset. A progress bar at the bottom indicates "Celda 92 de 93".

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez



ENTRENAMOS UN MODELO CLASIFICADOR PARA DETECTAR SI EXISTE TUMOR O NO

```

# Visualizamos que tenemos cuatro columnas
brain_df.shape
[1]: (3929, 4)

# Eliminamos la columna de identificador del paciente
brain_df_train = brain_df.drop(columns = ['patient_id'])

[2]: (3929, 3)

# Visualizamos que tenemos solo tres columnas ahora
brain_df_train.shape
[3]: (3929, 3)

# Convertir los datos en la columna de máscara a formato de string para usar el modo categórico en flow_from_dataframe
brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

[4]: (3929, 3)

# Obtenemos la información del dataframe
brain_df_train.info()
[5]: <class 'pandas.core.frame.DataFrame'>
RangeIndex: 3929 entries, 0 to 3928
Data columns (total 3 columns):
 # Columna          Non-Null Count Dtype  
 --- 
 0  image_path      3929 non-null   object 
 1  mask_path       3929 non-null   object 
 2  class           3929 non-null   object 
dtypes: object(3)
memory usage: 92.2+ KB

```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

EDAD

```
julio-2023-estap2 > TP-Final-PIML-II > Notebook.ipynb
+ Código + Markdown ▶ Ejecutar todo ⌘ Recargar | ⌘ Ejecutar Group 1 ⌘ Ejecutar Group 2 ⌘ Borrar todas las salidas | Variables Esquema ...
```

```

1 mask_path 3929 non-null object
2 mask 3929 non-null object
dtypes: object(3)
memory usage: 92.2+ KB

```

```
# Dividimos los datos para el entrenamiento y testing dejando un 15% para testeos y 85% para entrenamiento
from sklearn.model_selection import train_test_split
train, test = train_test_split(brain_df.train, test_size = 0.15)
```

```
# 0.4s
```

```
# Creamos el generador de imágenes
from keras.preprocessing.image import ImageDataGenerator
```

```
# Creamos un generador de datos que escala los datos de 0 a 1 y hace una división de validación de 0.15
datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)
```

```
# Creamos un generador de datos para el entrenamiento
train_datagen=datagen.flow_from_dataframe(
    dataframe=train,
    directory='./dataset/',
    x_col='image_path',
    y_col='mask',
    subset='training',
    batch_size=16,
    shuffle=True,
    class_mode='categorical',
    target_size=(256,256))

# Creamos un generador de datos para la validación
val_datagen=datagen.flow_from_dataframe(
    dataframe=train,
    directory='./dataset/',
    x_col='image_path',
    y_col='mask',
    subset='validation',
    batch_size=16,
    shuffle=True,
    class_mode='categorical',
    target_size=(256,256))
```

RESNET50

```
julio-2023-estap2 > TP-Final-PIML-II > Notebook.ipynb
+ Código + Markdown ▶ Ejecutar todo ⌘ Recargar | ⌘ Ejecutar Group 1 ⌘ Ejecutar Group 2 ⌘ Borrar todas las salidas | Variables Esquema ...
```

```
# Obtenemos el modelo base de ResNet50 (red neuronal entrenada)
baseModel = ResNet50(weights = "imagenet", include_top = False, input_tensor = Input(shape=(256, 256, 3)))
```

```
# Obtenemos el resumen del modelo base con el detalle de parámetros
baseModel.summary()
```

```
Model: "resnet50"
Layer (type) Output Shape Param # Connected to
-----
```

Layer (type)	Output Shape	Param #	Connected to
Input_1 (InputLayer)	(None, 256, 256, 3)	0	[]
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	['input_1[0][0]']
conv1_conv (Conv2D)	(None, 128, 128, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 128, 128, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 128, 128, 64)	0	['conv1_bn[0][0]']
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	['conv1_relu[0][0]']
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	['pool1_pad[0][0]']
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchNormalizat	(None, 64, 64, 64)	256	['conv2_block1_1_conv[0][0]']

```
...  
Total params: 235037712 (89.98 MB)  
Trainable params: 23534092 (89.76 MB)  
Non-trainable params: 52120 (287.50 KB)
```

```
Output was truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.
```

```
# Congelamos los pesos del modelo
for layer in baseModel.layers:
    layer.trainable = False
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Notbook.ipynb

```

# Agregamos una cabecera de clasificación al modelo base
headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(756, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(252, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = "softmax")(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)

```

Modelo:

Layer (Type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 256, 256, 3)	0	[]
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	['input_1[0][0]']
conv1_conv (Conv2D)	(None, 128, 128, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalizati on)	(None, 128, 128, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 128, 128, 64)	0	['conv1_bn[0][0]']
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	['conv1_relu[0][0]']
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	['pool1_pad[0][0]']
conv2_block1_1_conv (conv2	(None, 64, 64, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchN	(None, 64, 64, 64)	256	['conv2_block1_1_conv[0][0]']

Quienes no quieran entrenar el model, y usar el mejor guardado, obviar el HISTORY Y el GUARDADO

```

# Complimos el modelo
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics= ['accuracy'])

# Utilizamos la parada temprana para salir del entrenamiento si la pérdida en la validación no disminuye incluso después de ciertas épocas (pacientia)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)

# Guardamos el mejor modelo con la menor pérdida de validación
checkpointer = ModelCheckpoint(filepath='classifier-resnet-weights.hdf5', verbose=1, save_best_only=True)

```

Continuar desde ACÁ

```

# Generamos el entrenamiento
history = model.fit(train_generator, steps_per_epoch= train_generator.n // 16, epochs = 1, validation_data= valid_generator, validation_steps= valid_generator.n // 16, callbacks=[checkpointer, earlystopping])

```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Continuar desde ACÁ

```
# Cambiamos la arquitectura de la red agregando una capa y un Dropouts más a fin de mejorar el rendimiento

headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name='Flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dense(128, activation = "relu")(headmodel)

# Agregamos una capa más de 256 con la misma rectificadora lineal unitaria
headmodel = Dense(256, activation = "relu")(headmodel)
# Agregamos una capa más de Dropout con el mismo valor
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = "softmax")(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)

# este procedimiento permite generar una nueva versión del modelo.
```

Visualizamos los nuevos parámetros del modelo

```
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
Input_1 (InputLayer)	(None, 256, 256, 3)	0	[]
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	['Input_1[0][0]']
conv1_conv (Conv2D)	(None, 128, 128, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalizati	(None, 128, 128, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 128, 128, 64)	0	['conv1_bn[0][0]']

Quienes no quieran entrenar el model, y usar el mejor guardado, obviar el HISTORY Y el GUARDADO

```
# Volvemos a generar la parada temprana y guardamos el mejor modelo con la menor pérdida de validació
# Utilizamos la parada temprana para salir del entrenamiento si la pérdida en la validación no disminuye incluso después de ciertas épocas (pacientes)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)

# Guardamos el mejor modelo con la menor pérdida de validación
checkpointer = ModelCheckpoint(filepath="classifier-resnet-weights.hdf5", verbose=1, save_best_only=True)
```

Continuar desde ACÁ

EVALUAMOS EL RENDIMIENTO DEL MODELO ENTRENADO

```
# Cargamos el modelo pre entrenado generado en el JSON "resnet-50-MRI.json" (MODELO ENTRENADO GUARDADO)
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

EDAD

```
julio-2023-estad>TP-Final-PIML-II>Notebook.ipynb
+ Código + Markdown ▶ Ejecutar todo ⌂ Reiniciar ⌂ Ejecutar Grupo 1 ⌂ Ejecutar Grupo 2 ⌂ Borrar todas las salidas | ⌂ Variables ⌂ Esquema ...
```

```
+ Código # Hacemos la predicción, ésta se realiza sobre el ísk de imágenes que se dejaron para testeos, no con las que se usaron para entrenar. Las peticiones las realizamos en lotes de 10 imágenes cada uno
+ Código test_predict = model.predict(test_generator, steps = test_generator.n // 10, verbose = 1)
+ Código 25s
+ Código ... 36/36 [=====] - 25s 660ms/step
```

```
+ Código # De las imágenes chequeamos que sean dos salidas (tiene o no tiene tumor)
+ Código test_predict.shape
+ Código ... 0s
+ Código ... (576, 2)
```

```
+ Código # Visualizamos un array con la probabilidad de que no contenga tumor en la primera columna y la probabilidad de que tenga en la segunda. Para cada imagen se genera un array con los dos valores
+ Código test_predict
+ Código ... 0s
+ Código ... array([[0.49301418, 0.50898589],
   [0.49562278, 0.50437225],
   [0.49458474, 0.5054152],
   ...
   [0.4957118, 0.5042889],
   [0.4929988, 0.50790115],
   [0.49631876, 0.50360612]], dtype=float32)
```

```
+ Código # Obtenemos la clase predicha a partir del modelo
+ Código # con argmax podemos ver la probabilidad máxima de cada fila, donde si en la posición 0 indica que probablemente no contenga un tumor, la posición 1 que si lo contenga.
+ Código predict = []
+ Código for i in test_predict:
+ Código     predict.append(str(np.argmax(i)))
+ Código predict = np.asarray(predict)
+ Código 0s
+ Código ... # Ahora visualizamos el array con los valores de probabilidad
+ Código predict
+ Código ... 0s
```

```
+ Código # Dado que usamos el generador de prueba, se limita el número de imágenes a leer (predicir), debido al tamaño del lote
+ Código original = np.asarray(test['mask'])[:len(predict)]
+ Código len(original)
+ Código ... 0s
+ Código ... 576
```

```
+ Código # Una vez realizado esto, podemos obtener la tasa de acierto del modelo con el set de datos de testeos
+ Código from sklearn.metrics import accuracy_score
+ Código accuracy = accuracy_score(original, predict)
+ Código accuracy
+ Código ... 0.3576388888888889
```

```
+ Código # La red neuronal tiene un porcentaje de efectividad del 90.61% sin aplicar la matriz de confusión.
+ Código # Representamos la matriz de confusión que nos permite identificar los falsos positivos y falsos negativos
+ Código from sklearn.metrics import confusion_matrix
+ Código cm = confusion_matrix(original, predict)
+ Código plt.figure(figsize = (7,7))
+ Código sns.heatmap(cm, annot=True)
```

```
+ Código ... 0s
+ Código ... <Axes: >
+ Código ... 0 0 3.7e+02
+ Código ... 350
+ Código ... 300
+ Código ... 250
```

TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Machine Learning 2

```

# Con esto podemos ver que 4 muestras dieron un falso positivo, y 6 muestras dieron un falso negativo. Dando un total de 8 muestras erróneas
# Imprimimos el informe de clasificación y comentamos sobre los resultados de precisión, recuperación y F1-Score (media entre la precisión y recall)

from sklearn.metrics import classification_report

report = classification_report(y_original, predict, labels = [0,1])
print(report)

```

Python

	precision	recall	f1-score	support
0	0.00	0.00	0.00	370
1	0.16	1.00	0.51	210

Celda 92 de 93

Machine Learning 2

```

# Obtenemos el dataframe que contiene las resonancias magnéticas que tienen máscaras asociadas.

brain_df['mask'] = brain_df['mask'].ffill()
brain_df['mask'].shape

```

Python

```

(1373, 4)

```

(0s)

```

# Tenemos que hay 1373 muestras con máscaras asociadas.

# Dividimos los datos en datos de prueba y de entrenamiento

from sklearn.model_selection import train_test_split

# Dividimos en dos modelos para entrenar el 85% y para validar el 15%

X_train, X_val = train_test_split(brain_df['mask'], test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)

```

Python

```

X_train, X_val = train_test_split(brain_df['mask'], test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)

# Creamos una lista separada para imgId, classId para pasar al generador# Crear una lista separada para imgId, classId para pasar al generador

train_ids = list(X_train.image_path)
train_mask = list(X_train.mask_path)

val_ids = list(X_val.image_path)
val_mask = list(X_val.mask_path)

# Construir rutas de imágenes y máscaras para entrenamiento y validación
data_folder = './dataset/'

train_ids = [os.path.join(data_folder, filename) for filename in train_ids]
train_mask = [os.path.join(data_folder, filename) for filename in train_mask]

val_ids = [os.path.join(data_folder, filename) for filename in val_ids]
val_mask = [os.path.join(data_folder, filename) for filename in val_mask]

```

Python

```

# Importamos utilities para la función de pérdida personalizada y el generador de datos personalizados

```

Celda 92 de 93

Archivo Editor Selección Ver Ir Ejecutar Terminal Ayuda

EDTORES ABIERTOS 1 sin guardar

MACHINE LEARNING 2

```
julio-2023-etalpa2 > TP-Final-PIML-II > Notebook.ipynb
+ Código # Forma del tensor de entrada
X_input = Input(shape=(input_shape))

# Fase 1
conv1_in = Conv2D(16,3,activation='relu', padding = 'same', kernel_initializer = 'he_normal')(X_input)
conv1_in = BatchNormalization()(conv1_in)
conv1_in = Conv2D(16,3,activation='relu', padding = 'same', kernel_initializer = 'he_normal')(conv1_in)
conv1_in = MaxPool2D(pool_size = (2,2))(conv1_in)

# Fase 2
conv2_in = resblock(conv1_in, 32)
pool2_1 = MaxPool2D(pool_size = (2,2))(conv2_in)

# Fase 3
conv3_in = resblock(pool2_1, 64)
pool3_1 = MaxPool2D(pool_size = (2,2))(conv3_in)

# Fase 4
conv4_in = resblock(pool3_1, 128)
pool4_1 = MaxPool2D(pool_size = (2,2))(conv4_in)

# Fase 5 (Cuello de Botella)
conv5_in = resblock(pool4_1, 256)

# Fase de Escalada
up_1 = upsample_concat(conv5_in, conv4_in)
up_1 = resblock(up_1, 128)

# Fase de Escalada 2
up_2 = upsample_concat(up_1, conv3_in)
up_2 = resblock(up_2, 64)

# Fase de Escalada 3
up_3 = upsample_concat(up_2, conv2_in)
up_3 = resblock(up_3, 32)

# Fase de Escalada 4
up_4 = upsample_concat(up_3, conv1_in)
up_4 = resblock(up_4, 16)

# Salida Final
output = Conv2D(1, (1,1), padding = "same", activation = "sigmoid")(up_4)

model_seg = Model(inputs = X_input, outputs = output )
```

LÍNEA DE TIEMPO

Archivo Editor Selección Ver Ir Ejecutar Terminal Ayuda

EDTORES ABIERTOS 1 sin guardar

MACHINE LEARNING 2

ENTRENAMOS EL MODELO DE RESUNET DE SEGMENTACIÓN PARA LOCALIZAR EL TUMOR

```
# Importamos de utilidades el código para la función de pérdida personalizada y el generador de datos personalizados a nivel de pixel.

from dataset.utilities import focal_tversky, tversky_loss, tversky
```

[7] ✓ 0s

```
# Compilamos el modelo utilizando el optimizador adam
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model_seg.compile(optimizer = adam, loss = focal_tversky, metrics = [tversky])
```

[7] ✓ 0s

```
... WARNING:absl:'lr' is deprecated in Keras optimizer, please use 'learning_rate' or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# Usamos la parada temprana para salir del entrenamiento si la pérdida de validación no disminuye incluso después de ciertas épocas (paciencia)
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)

# Guardamos el mejor modelo con menor pérdida de validación
checkpointer = ModelCheckpoint(filepath="ResNet-weights.hdf5", verbose=1, save_best_only=True)
```

[7] ✓ 0s

Quienes no quieran entrenar el model, y usar el mejor guardado, obviar el HISTORY Y el GUARDADO

```
# Generamos el entrenamiento de la segmentación de la arquitectura
history = model_seg.fit(training_generator , epochs = 1, validation_data = validation_generator, callbacks = [checkpointer, earlystopping])
```

[7] ✓ 3m23s

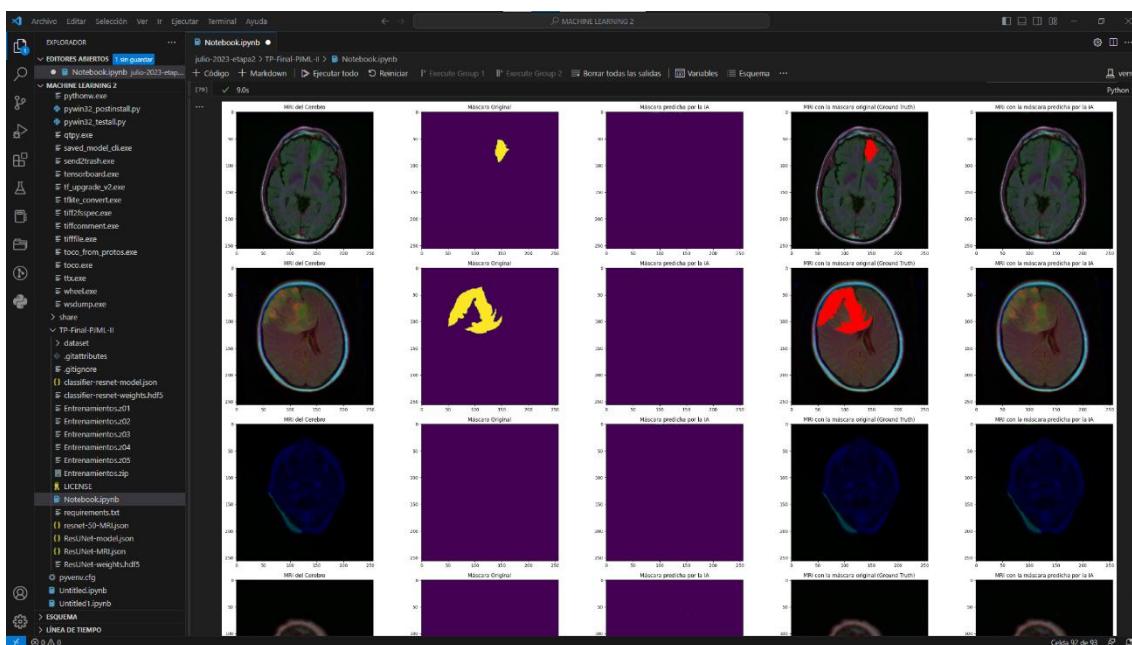
```
... 72/72 [=====] - ETA: 0s - loss: 0.9375 - tversky: 0.0024
Epoch 1: val_loss improved from Inf to 0.93479, saving model to ResNet-weights.hdf5
72/72 [=====] - 212s 35ms/step - loss: 0.9375 - tversky: 0.0024 - val_loss: 0.9448 - val_tversky: 0.0729
0:Arrengelis programa 4.0 MACHINE LEARNING 2/julio-2023-etalpa2/lib/site-packages/keras/src/engine/training.py:300: UserWarning:
```

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.

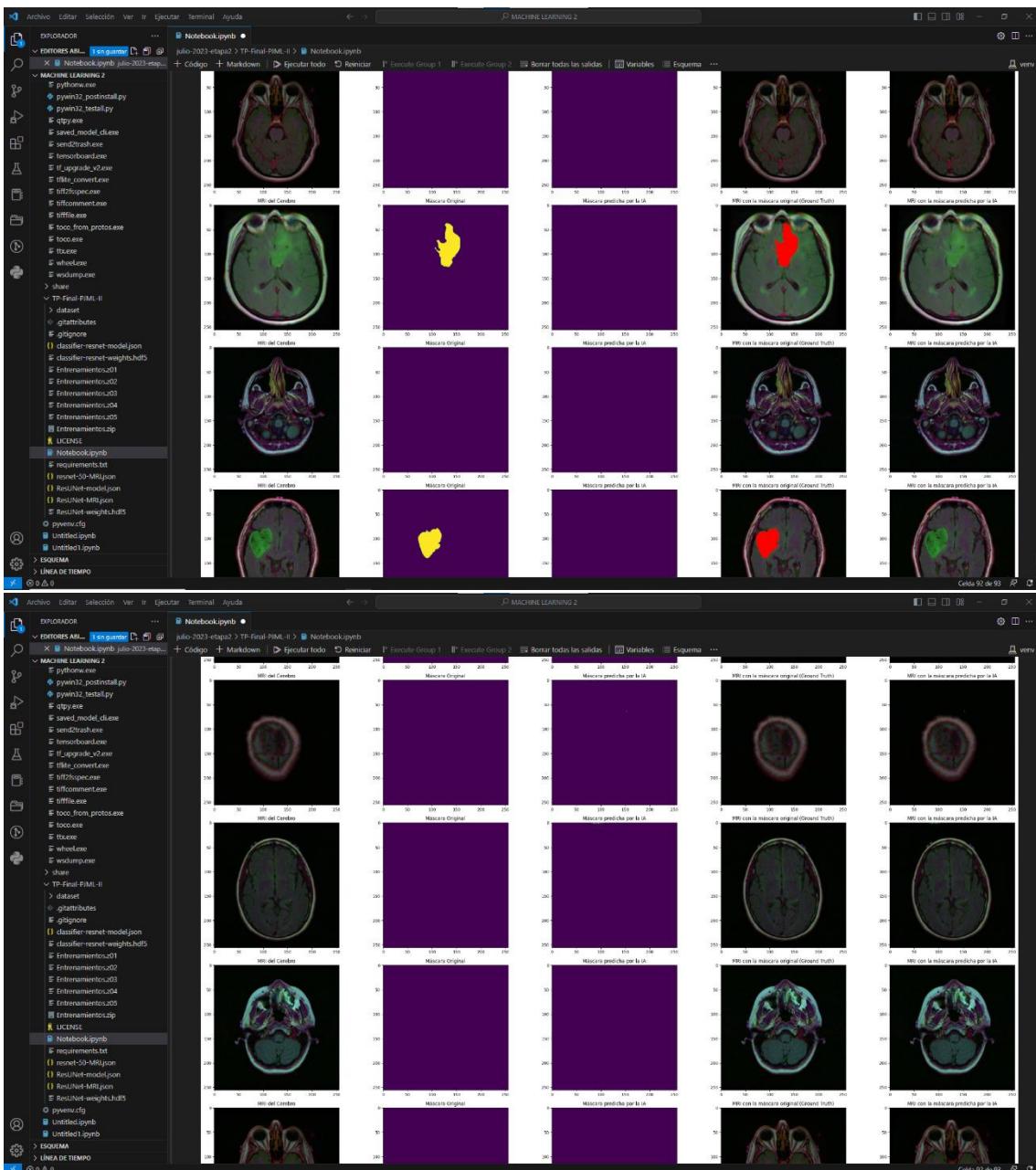
Celda 92 de 93

Resultados

Como se pude apreciar en las siguientes imágenes el programa quedo funcionando y los resultados son evidentes. Esta herramienta medica que es el programa creado sirve para acelerar el proceso de diagnóstico de tumores cancerígenos. Es gracias a el avance tecnológico que se puede realizar un diagnostico que tardaba un tiempo indeterminado que podía alargarse según los recursos, económicos, sociales, familiares, etc. que el individuo poseía a 3 minutos de tardanza en saber si tiene el padecimiento no, las imágenes a continuación son solo imágenes de muestra que se encuentran en el data set y fueron procesados por el programa.



TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez



TRABAJO INTEGRADOR MODULO II | Julio Arturo Rodriguez

Two screenshots of a Jupyter Notebook interface showing brain MRI segmentation results.

Screenshot 1: A grid of 12 images comparing original MRI slices with their corresponding ground truth and predicted masks. The images are arranged in three rows and four columns. The first column shows axial MRI slices. The second column shows the "Máscara Original" (Ground Truth Mask). The third column shows the "Máscara predicha por la IA" (Predicted Mask by the AI). The fourth column shows the "MRI con la Máscara Original (Cerebro Truth)" and "MRI con la Máscara predicha por la IA" side-by-side. The predicted masks are shown in red, while the ground truth is yellow.

Screenshot 2: A screenshot of a Jupyter Notebook cell containing Python code for visualizing MRI slices and masks. The code uses matplotlib subplots to show three images: "MRI del Cerebro" (Axial MRI slice), "Máscara" (Ground truth mask), and "MRI con Máscara" (MRI slice with the predicted mask overlaid). The code also includes a loop to iterate through the dataset.

```

# Visualizamos la resonancia magnética, la máscara, y ambas de manera superpuesta
# Posición del DATASET
i = 623

if brain_df['mask'][i] == 1:
    fig, axs = plt.subplots(1, 3, figsize=(20, 5)) # Solo una fila con tres columnas

    img = io.imread(image_paths[i])
    axs[0].title.set_text("MRI del Cerebro")
    axs[0].imshow(img)

    mask = io.imread(mask_paths[i])
    axs[1].title.set_text("Máscara")
    axs[1].imshow(mask, cmap="gray")

    img[mask == 255] = (255, 0, 0)
    axs[2].title.set_text("MRI con Máscara")
    axs[2].imshow(img)

    plt.tight_layout()
    plt.show()

```

```

# Visualización básica de imágenes (MRI y Máscaras) en el dataset de forma separada de manera aleatoria
fig, axes = plt.subplots(6,2, figsize=(16,32))
count = 0
for x in range(0):
    i = random.randint(0, len(brain_df)) # Seleccionamos un índice aleatorio
    axes[count][0].imshow(brain_df[i].image) # Configuramos el título
    axes[count][0].set_title("MRI del Cerebro")
    axes[count][1].imshow(brain_df[i].mask) # Colocamos el título en la máscara (0 o 1)
    axes[count][1].set_title("Máscara - 1")
    axes[count][1].imshow(cv2.imread(mask_paths[1])) # Mostramos la máscara correspondiente
    count += 1
fig.tight_layout()

```

Discusión

Si bien esto implica una mejora y una ayuda importante, las nuevas tecnologías están muy lejos de reemplazar a un doctor, la calidez y compasión de un ser humano no puede y no debe ser comparada con el frío de una máquina, ya que la misma no está programada para sentir emociones, y por esa razón entre otras tantas validas que aun estamos lejos de eso, es por esa razón que esto mas que algo de uso universal es una herramienta, una ayuda para facilitar el trabajo a los doctores, gente que se tomo una vida para nutrirse en esa área. La única finalidad de este programa es acompañar a los pacientes haciendo más rápido su detección y también más económica, pero también más fácil y llevadero el trabajo del doctor que en algunos casos que eh conocido atienden 300 personas por día de lunes a viernes lo cual se les hace muy pesado.

Conclusiones y recomendaciones

Se puede concluir en que el programa, de aprendizaje supervisada esta listo y en funcionamiento, si se puede comentar algo es que el mismo puede seguir evolucionando para otras patologías y utilizarse de uso general, ya que todo lo a que se le pueda tomar una foto puede tener el mismo proceso.