



# Especialización en Inteligencia Artificial

## **Vision Transformers**

---

Trabajo Práctico Final

Docentes:

**Oksana Bokhonok**  
**Abraham Rodriguez**

Alumnos:

**Noelia Qualindi**  
**Matías Werbin**

---

**Trabajo final sobre Clasificación de Tumores Cerebrales**

**Año 2024**

[UBA - Facultad de Ingeniería]

# Datasets

## Clasificación de Tumores Cerebrales

El dataset utilizado está compuesto por imágenes RGB que representan tres tipos de tumores cerebrales, así como imágenes que indican la ausencia de tumores. Las clases están etiquetadas de la siguiente manera:

- **0:** Glioma Tumor
- **1:** Meningioma Tumor
- **2:** No Tumor
- **3:** Pituitary Tumor

### Fuente del Dataset

## Elección de Arquitecturas

### CvT (Convolutional Vision Transformer):

Motivación: La principal ventaja de CvT radica en su enfoque híbrido que combina convoluciones para la extracción de características locales con el mecanismo de atención del Transformer. Esto mejora la capacidad del modelo para capturar tanto las dependencias locales como globales de la imagen.

Impacto: La combinación de convoluciones en las primeras capas permite que el modelo gestione patrones espaciales locales, lo que es útil para la clasificación de imágenes de tumores cerebrales, donde las características locales (como bordes y texturas) son importantes. La parte de atención permite que el modelo se enfoque en las partes más relevantes de la imagen, capturando las relaciones globales entre diferentes regiones.

### ViT (Vision Transformer):

Motivación: ViT utiliza solo el mecanismo de atención sin incorporar convoluciones. La idea es que el Transformer, al tener un enfoque global, podría ser capaz de capturar patrones complejos a nivel de toda la imagen.

### Importación de librerías

```
import matplotlib.pyplot as plt
from datasets import load_dataset, Dataset
import numpy as np
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader
import torch
```

Descarga del dataset y separación del conjunto de entrenamiento, validación y pruebas.

```
# Cargar el dataset
dataset = load_dataset("sartajbhuvaji/Brain-Tumor-Classification")
```

```

# Separar dataset de Train y Test
train = dataset['Training']
test_dataset = dataset['Testing']

# Extraer etiquetas para la estratificación
labels = train['label']

# Generar índices estratificados
train_indices, val_indices = train_test_split(
    np.arange(len(labels)),
    test_size=0.2,
    stratify=labels,
    random_state=42
)

# Dividir el conjunto original en entrenamiento y validación
train_dataset = train.select(train_indices)
val_dataset = train.select(val_indices)

# Comprobar tamaños
print("Tamaño del conjunto de entrenamiento:", len(train_dataset))
print("Tamaño del conjunto de validación:", len(val_dataset))
print("Tamaño del conjunto de prueba:", len(test_dataset))

Tamaño del conjunto de entrenamiento: 2296
Tamaño del conjunto de validación: 574
Tamaño del conjunto de prueba: 394

```

Inspeccionamos la estructura del dataset

```

# Inspección Dataset descargado
dataset

DatasetDict({
  Training: Dataset({
    features: ['image', 'label'],
    num_rows: 2870
  })
  Testing: Dataset({
    features: ['image', 'label'],
    num_rows: 394
  })
})

```

Inspeccionamos el conjunto de validación

```

val_dataset

Dataset({
  features: ['image', 'label'],

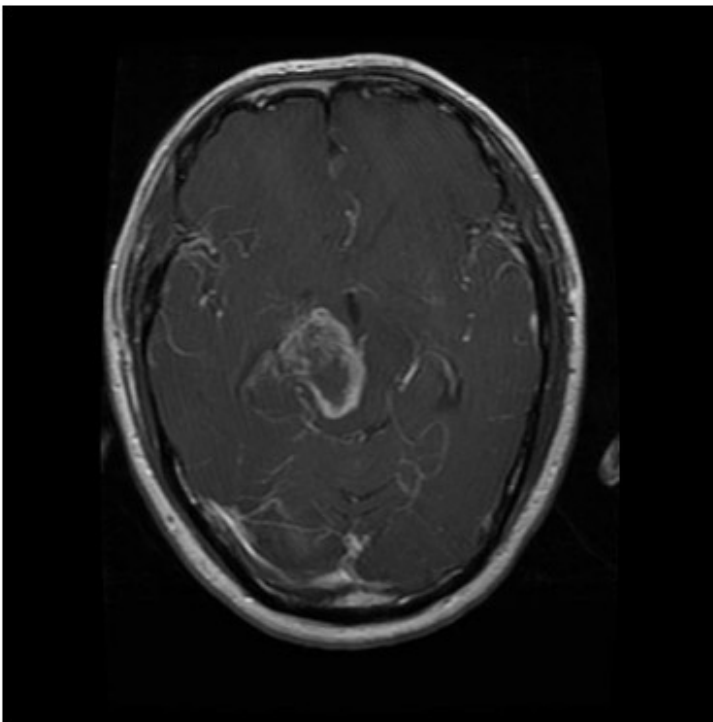
```

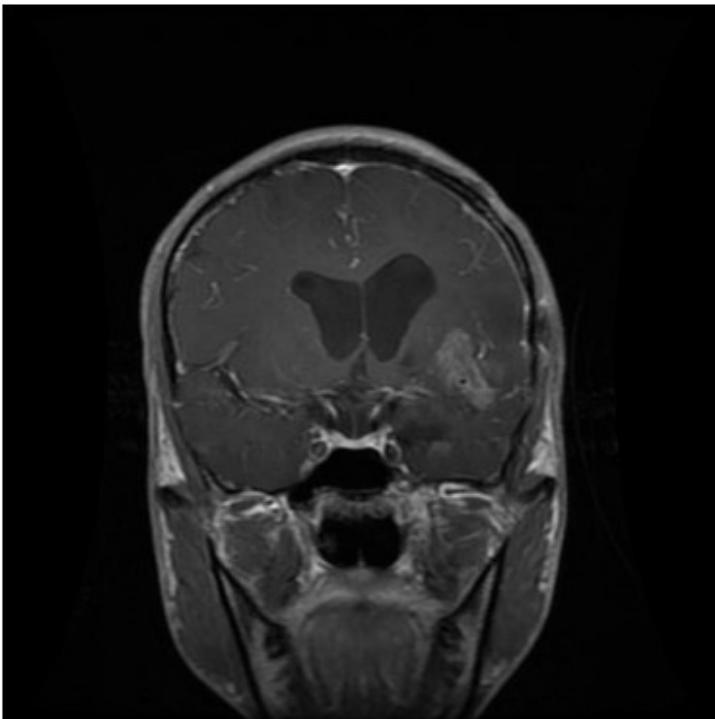
```
    num_rows: 574  
})
```

Imprimimos las primeras 3 imágenes de cada conjunto de datos.

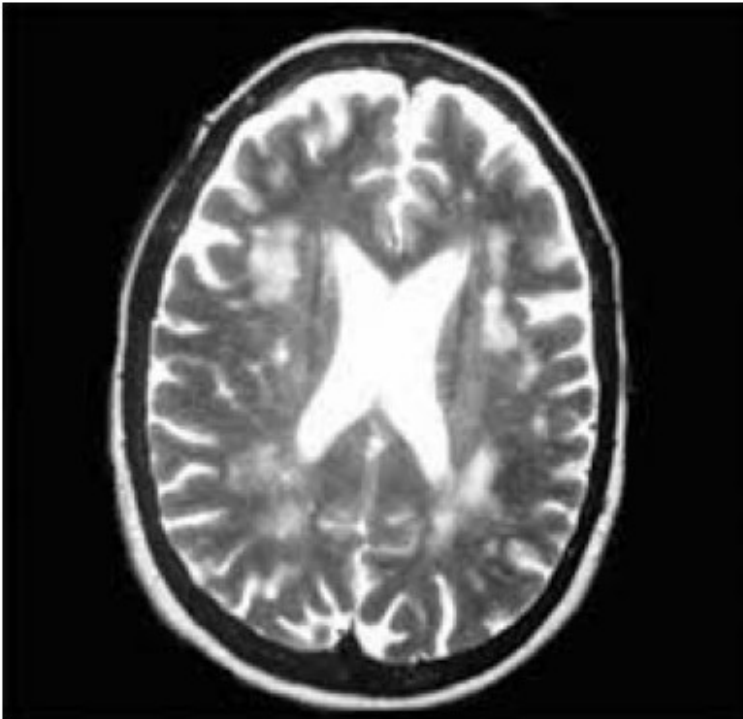
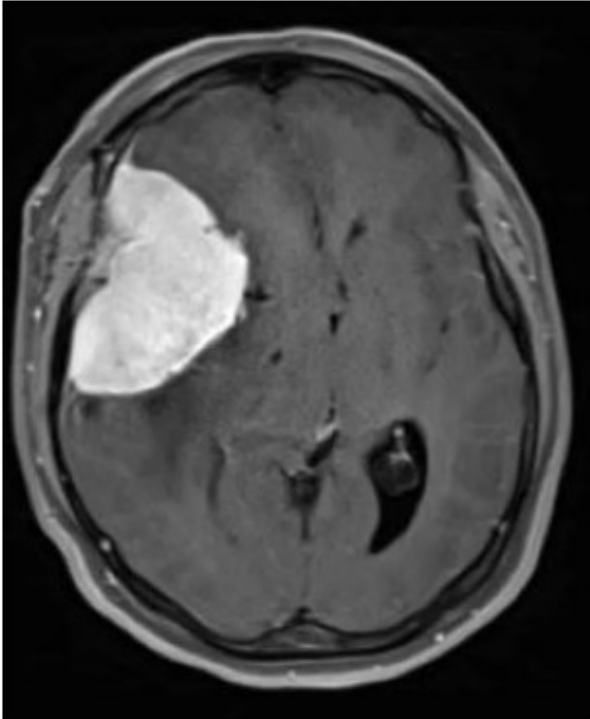
```
from PIL import Image  
  
# Visualización de primeras tres imágenes de train_dataset  
datasets = {'train_dataset': train_dataset, 'val_dataset':  
val_dataset, 'test_dataset': test_dataset}  
for name, dataset in datasets.items():  
    print(f"Primeras 3 imágenes en {name}")  
    for i in range(3):  
  
        image_info = dataset['image'][i]  
  
        if isinstance(image_info, Image.Image):  
            image = image_info  
        else:  
            image = Image.open(image_info['src'])  
  
        plt.imshow(image)  
        plt.axis('off')  
        plt.show()
```

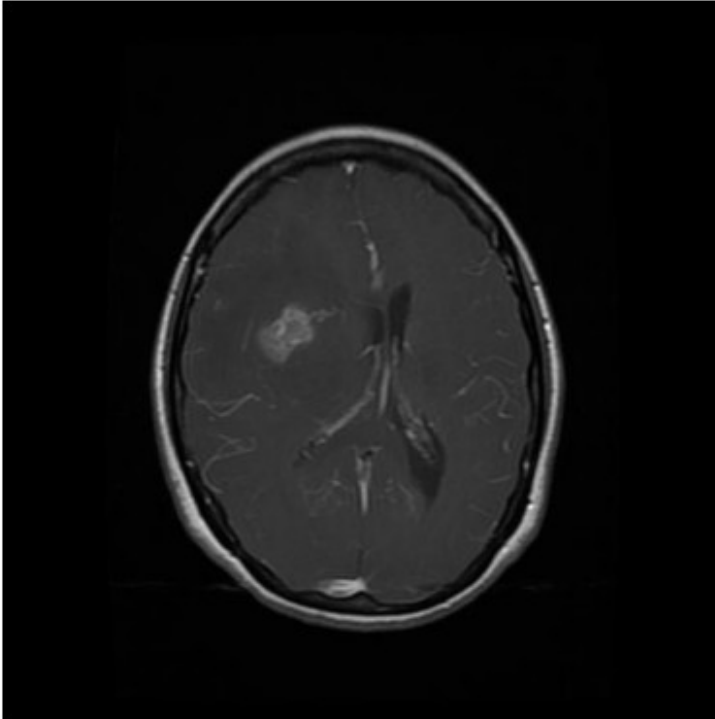
Primeras 3 imágenes en train\_dataset





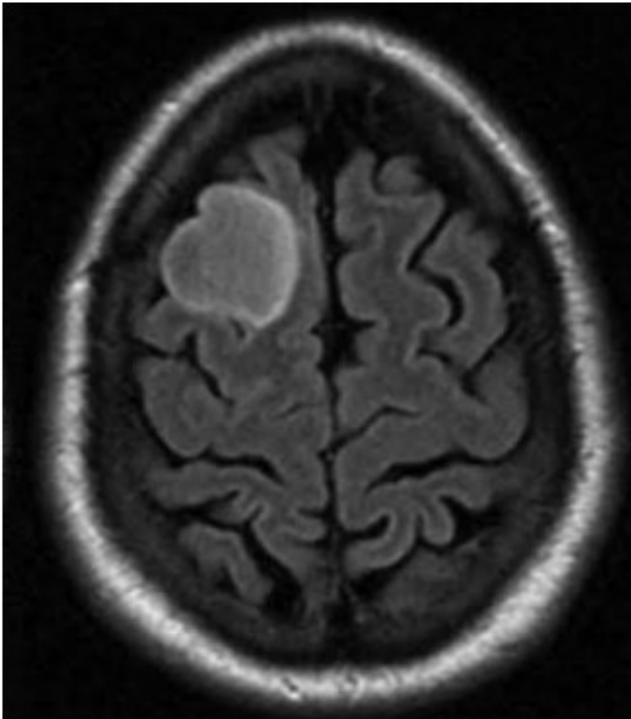
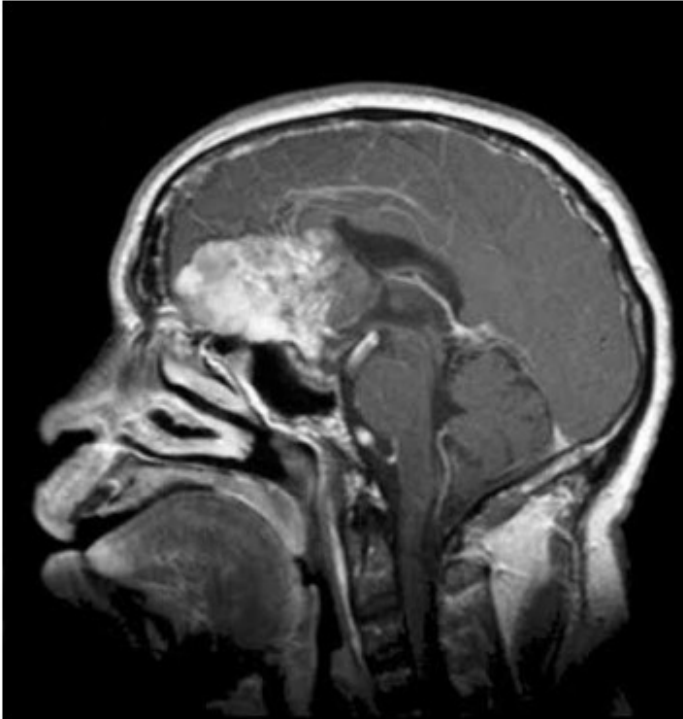
Primeras 3 imágenes en val\_dataset





Primeras 3 imágenes en test\_dataset





## Clases del Dataset-Análisis de balanceo

Analizamos las clases correspondientes al dataset.



```
# Generar diccionarios id2label y label2id para el mapeo-manejo de
clases e ids
id2label = {id:label for id, label in
enumerate(train_dataset.features['label'].names)}
label2id = {label:id for id,label in id2label.items()}

print(f"Diccionario id -> label: {id2label}")

Diccionario id -> label: {0: 'glioma_tumor', 1: 'meningioma_tumor', 2:
'no_tumor', 3: 'pituitary_tumor'}
```

Verificamos la cantidad de imágenes que hay para cada clase.

```
from collections import Counter

# Convertir las etiquetas de tensores a enteros antes de contarlas
train_labels = [int(example['label']) for example in train_dataset]
valid_labels = [int(example['label']) for example in val_dataset]
test_labels = [int(example['label']) for example in test_dataset]

# Distribuciones de clases segun set
train_distribution = Counter(train_labels)
val_distribution = Counter(valid_labels)
test_distribution = Counter(test_labels)

print("Distribución en train:", train_distribution)
print("Distribución en valid:", val_distribution)
print("Distribución en test:", test_distribution)

Distribución en train: Counter({3: 662, 0: 661, 1: 657, 2: 316})
Distribución en valid: Counter({1: 165, 0: 165, 3: 165, 2: 79})
Distribución en test: Counter({1: 115, 2: 105, 0: 100, 3: 74})
```

Para visualizar las distribuciones, imprimimos 3 gráficos de barras correspondiente a cada conjunto de datos.

```
# Diccionario id->label
label2id = {0: 'glioma_tumor', 1: 'meningioma_tumor', 2: 'no_tumor',
3: 'pituitary_tumor'}

# Transformar las claves del Counter a nombres de clases
train_classes = [label2id[label] for label in
train_distribution.keys()]
train_counts = list(train_distribution.values())

val_classes = [label2id[label] for label in val_distribution.keys()]
val_counts = list(val_distribution.values())

test_classes = [label2id[label] for label in test_distribution.keys()]
test_counts = list(test_distribution.values())
```

```

# Configuración de la gráfica para analizar balanceo
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

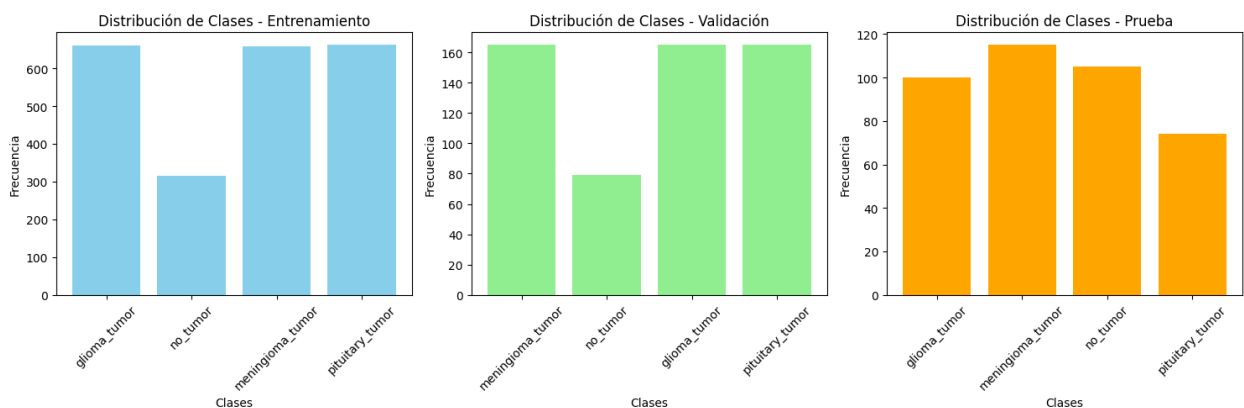
# Barplot para train_dataset
ax[0].bar(train_classes, train_counts, color='skyblue')
ax[0].set_title('Distribución de Clases - Entrenamiento')
ax[0].set_xlabel('Clases')
ax[0].set_ylabel('Frecuencia')
ax[0].set_xticks(range(len(train_classes)))
ax[0].set_xticklabels(train_classes, rotation=45)

# Barplot para val_dataset
ax[1].bar(val_classes, val_counts, color='lightgreen')
ax[1].set_title('Distribución de Clases - Validación')
ax[1].set_xlabel('Clases')
ax[1].set_ylabel('Frecuencia')
ax[1].set_xticks(range(len(val_classes)))
ax[1].set_xticklabels(val_classes, rotation=45)

# Barplot para test_dataset
ax[2].bar(test_classes, test_counts, color='orange')
ax[2].set_title('Distribución de Clases - Prueba')
ax[2].set_xlabel('Clases')
ax[2].set_ylabel('Frecuencia')
ax[2].set_xticks(range(len(test_classes)))
ax[2].set_xticklabels(test_classes, rotation=45)

plt.tight_layout()
plt.show()

```



Podemos observar que tanto en el conjunto de entrenamiento como en el de validación existe un desbalance de clases. En este caso, la clase minoritaria es no\_tumor. A priori, esto no debería representar un problema, ya que nuestro objetivo principal es que la red aprenda a clasificar correctamente las imágenes que contienen tumores y aquellas que no. Sin embargo, también es

crucial poder predecir correctamente las imágenes sin tumores. A pesar de esto, en este tipo de problemas, es más crítico maximizar los verdaderos positivos (TP) de las clases asociadas a los tumores.

Podemos notar que el conjunto de prueba está bastante balanceado, por lo que podemos utilizar la precisión (accuracy) como métrica de evaluación.

Nos guardamos los datasets para cada modelo a utilizar, ya que luego tendrán que ser procesados acorde a la extracción de características de uno de ellos.

## Set para CVT Model

```
# datasets para el modelo de Cvt-13 de Microsoft
train_dataset_cvt = train_dataset
test_dataset_cvt = test_dataset
val_dataset_cvt = val_dataset
```

## Set para Base ViT Google

```
# datasets para el modelo Vit Base de Google
train_dataset_g = train_dataset
test_dataset_g = test_dataset
val_dataset_g = val_dataset
```

## Configuración del modelo CvT (Microsoft)

Realizamos las configuraciones e inicializaciones correspondientes del modelo CvT, con su extractor de características.

```
from transformers import AutoFeatureExtractor,
CvtForImageClassification, Trainer, TrainingArguments

# Descargar modelo preentrenado como su correspondiente feature
extractor
feature_extractor =
AutoFeatureExtractor.from_pretrained('microsoft/cvt-13')
model = CvtForImageClassification.from_pretrained('microsoft/cvt-13')

/opt/anaconda3/envs/mipy/lib/python3.11/site-packages/transformers/
models/convnext/feature_extraction_convnext.py:28: FutureWarning: The
class ConvNextFeatureExtractor is deprecated and will be removed in
version 5 of Transformers. Please use ConvNextImageProcessor instead.
warnings.warn(
```

Setemos el dispositivo para entrenar el modelo en la GPU (Si corresponde)

```
# Seteo de GPU si se encuentra disponible
device = 'cuda' if torch.cuda.is_available() else 'mps' if
torch.backends.mps.is_available() else 'cpu'
print(device)
```

mps

## Preprocesamiento, transformación y aumentación de datos

```
# Revisión de cantidad de datos en train antes de realizar data_augmentation
print(f"Data en Train antes de aplicar las transformaciones: {len(train_dataset)} imágenes ")
```

Data en Train antes de aplicar las transformaciones: 2296 imágenes

### Uso de Aumento de Datos

Aplicamos aumento de datos en el conjunto de Train (como rotaciones, desplazamientos y cambios de escala) para mejorar la generalización del modelo. Esto es particularmente útil para reducir el overfitting y asegurar que el modelo no dependa de un conjunto específico de imágenes.

Impacto: El aumento de datos permitió que el modelo tuviera una mayor diversidad de ejemplos durante el entrenamiento, lo que ayudó a mejorar el rendimiento general y reducir los errores de clasificación, especialmente en clases con menos ejemplos.

```
from torchvision import transforms

# Data Augmentation
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(degrees=15),
    transforms.RandomResizedCrop(size=224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor()
])
```

Funciones a utilizar para aumentar los datos y para transformar las imágenes según el extractor de características del modelo.

```
from datasets import load_dataset, concatenate_datasets
import random

# Función de aumentación y transformación de datos (con validación/conversión de imagen RGB -3 channels). Dicha función, se aplicará únicamente en train_dataset
def augmented_train(example, feature_extractor):
    image = example['image']
```

```

# Validación RGB
if image.mode != 'RGB':
    image = image.convert("RGB")

# Aumentación de datos
augmented_image = train_transforms(image)

# Aplicación de feature_extractor asociado a CVT-13
inputs = feature_extractor(images=augmented_image,
return_tensors="pt", padding=True, do_rescale=False)
example['pixel_values'] =
inputs['pixel_values'].squeeze().to(device)
example['label'] = example['label']
return example

# Función de transformación de datos.
def transform(example, feature_extractor):
    image = example['image']

    if image.mode != 'RGB':
        image = image.convert("RGB")

    inputs = feature_extractor(images=image, return_tensors="pt",
padding=True, do_rescale=False)
    example['pixel_values'] =
inputs['pixel_values'].squeeze().to(device)
    example['label'] = example['label']
    return example

```

Aplicamos la aumentación de datos para el conjunto de train, y aplicamos las transformaciones correspondientes a las imágenes.

```

# Aumentación y transformación de datos en train_dataset
augmented_cvt = train_dataset_cvt.map(lambda item:
augmented_train(item, feature_extractor))

# Imputación de feature_extractor en en train, valid y test dataset
test_dataset_cvt = test_dataset_cvt.map(lambda item: transform(item,
feature_extractor))
train_dataset_cvt = train_dataset_cvt.map(lambda item: transform(item,
feature_extractor))
val_dataset_cvt = val_dataset_cvt.map(lambda item: transform(item,
feature_extractor))

# Concatenar train_dataset original con el augmented_cvt (data
augmentation)
train_dataset_cvt = concatenate_datasets([train_dataset_cvt,
augmented_cvt])

```

```

Map: 0%|          | 0/2296 [00:00<?, ?
examples/s]/opt/anaconda3/envs/mipy/lib/python3.11/site-packages/trans
formers/image_processing_utils.py:41: UserWarning: The following named
arguments are not valid for `ConvNextFeatureExtractor.preprocess` and
were ignored: 'padding'
    return self.preprocess(images, **kwargs)
Map: 100%|██████████| 2296/2296 [00:45<00:00, 50.41 examples/s]
Map: 100%|██████████| 394/394 [00:02<00:00, 155.08 examples/s]
Map: 100%|██████████| 2296/2296 [00:41<00:00, 54.92 examples/s]
Map: 100%|██████████| 574/574 [00:10<00:00, 54.87 examples/s]

```

Verificamos que el aumento de datos haya sido exitoso.

```

# Inspección de datasets luego de aumentación y transformación de
datos
print(f"Tamaño del conjunto de entrenamiento después del aumento de
datos: {len(train_dataset_cvt)}") # Podemos observar el incremento de
imágenes previo la aumentación: 2296 a post aumentación: 4592
print(f"Tamaño del conjunto de validación: {len(val_dataset_cvt)}")
print(f"Tamaño del conjunto de prueba: {len(test_dataset_cvt)}")

```

Tamaño del conjunto de entrenamiento después del aumento de datos:  
4592

Tamaño del conjunto de validación: 574

Tamaño del conjunto de prueba: 394

Seteamos el dataset al formato de torch para que lo pueda consumir el modelo.

```

# Seteo de datos torch para consumo de ViT
train_dataset_cvt.set_format(type='torch', columns=['pixel_values',
'labels'])
val_dataset_cvt.set_format(type='torch', columns=['pixel_values',
'labels'])
test_dataset_cvt.set_format(type='torch', columns=['pixel_values',
'labels'])

```

Función que se utilizará para computar las métricas en el Trainer.

```

from sklearn.metrics import accuracy_score, f1_score, recall_score

# Función asociado al cálculo de métricas
def compute_metrics(p):
    predictions, labels = p
    preds = torch.argmax(torch.tensor(predictions), axis=1) #
    Predicción de clases
    labels = torch.tensor(labels) # tensorizado de labels

    # Calcular métricas
    accuracy = accuracy_score(labels.numpy(), preds.numpy())

```

```

    f1 = f1_score(labels.numpy(), preds.numpy(), average='weighted')
# F1-score ponderado
    recall = recall_score(labels.numpy(), preds.numpy(),
average='weighted') # Recall ponderado

    return {
        "accuracy": accuracy,
        "f1": f1,
        "recall": recall,
    }

import os
os.environ["WANDB_DISABLED"] = "true" # no utilizamos weights and
biases

```

## Fine-Tuning

Enviamos el modelo al dispositivo.

```
model.to(device)
```

Realizamos un fine-tuning del modelo CvT, con los siguientes hiperparámetros:

- 10 épocas de entrenamiento
- Learning rate de 2e-5
- Tamaño del batch por dispositivo de 16.

```

from transformers import TrainingArguments, Trainer

# Seteo de hiperparámetros
epochs = 10
learning_rate = 2e-5
batch_size_per_device = 16

class_names = {
    0: "glioma_tumor",
    1: "meningioma_tumor",
    2: "no_tumor",
    3: "pituitary_tumor"
}

# Seteo de TrainingArguments para Fine-Tuning
training_args = TrainingArguments(
    output_dir="./results", # Directorio para guardar los resultados
    evaluation_strategy="epoch", # Evaluar al final de cada época
    save_strategy="epoch", # Guardar el modelo al final de cada época
    logging_strategy="epoch",
    learning_rate=learning_rate, # Tasa de aprendizaje
    per_device_train_batch_size=batch_size_per_device, # Tamaño del

```

```

batch para entrenamiento
    per_device_eval_batch_size=batch_size_per_device, # Tamaño del
batch para evaluación
    num_train_epochs=epochs, # Número de épocas
    weight_decay=0.01, # Regularización L2
    logging_dir="./logs", # Directorio para los logs
    logging_steps=10, # Intervalo de pasos para registrar logs
    load_best_model_at_end=True, # Cargar el mejor modelo al final
    metric_for_best_model="accuracy", # Métrica para evaluar el mejor
modelo
)

/opt/anaconda3/envs/mipy/lib/python3.11/site-packages/transformers/
training_args.py:1568: FutureWarning: `evaluation_strategy` is
deprecated and will be removed in version 4.46 of Transformers. Use
`eval_strategy` instead
    warnings.warn(

```

Seteamos el Trainer con el modelo inicializado de CvT, y su extractor de características correspondiente. Utilizamos la función que computará las métricas, y el conjunto de entrenamiento y validación que se setó en un principio para CvT.

```

# Seteo de clase Trainer de Hugging Face
trainer = Trainer(
    model=model, # Modelo a entrenar
    args=training_args, # Parámetros de entrenamiento
    train_dataset=train_dataset_cvt, # Conjunto de entrenamiento
    eval_dataset=val_dataset_cvt, # Conjunto de validación
    tokenizer=feature_extractor, # Tokenizer o extractor de
características
    compute_metrics=compute_metrics, # Función de evaluación
)

```

Entrenamos el modelo y nos guardamos los resultados.

```

# Entrenamiento de CVT-13
train_results = trainer.train()

10%|█          | 287/2870 [02:34<21:38, 1.99it/s]

{'loss': 3.1523, 'grad_norm': 133.3999481201172, 'learning_rate':
1.8e-05, 'epoch': 1.0}

10%|█          | 287/2870 [02:41<21:38, 1.99it/s]

{'eval_loss': 0.8077149391174316, 'eval_accuracy': 0.7073170731707317,
'eval_f1': 0.7044083037030155, 'eval_recall': 0.7073170731707317,
'eval_runtime': 6.7084, 'eval_samples_per_second': 85.564,
'eval_steps_per_second': 5.366, 'epoch': 1.0}

```



20%|██████████ | 574/2870 [05:10<20:25, 1.87it/s]

```
{'loss': 1.1754, 'grad_norm': 265.5242004394531, 'learning_rate':  
1.6000000000000003e-05, 'epoch': 2.0}
```

20%|██████████ | 574/2870 [05:16<20:25, 1.87it/s]

```
{'eval_loss': 0.521224319934845, 'eval_accuracy': 0.7926829268292683,  
'eval_f1': 0.7930413186486681, 'eval_recall': 0.7926829268292683,  
'eval_runtime': 5.6679, 'eval_samples_per_second': 101.272,  
'eval_steps_per_second': 6.352, 'epoch': 2.0}
```

30%|██████████ | 861/2870 [07:47<18:06, 1.85it/s]

```
{'loss': 0.8645, 'grad_norm': 97.42633056640625, 'learning_rate':  
1.4e-05, 'epoch': 3.0}
```

30%|██████████ | 861/2870 [07:52<18:06, 1.85it/s]

```
{'eval_loss': 0.46796876192092896, 'eval_accuracy':  
0.8240418118466899, 'eval_f1': 0.8240317011678377, 'eval_recall':  
0.8240418118466899, 'eval_runtime': 5.7227, 'eval_samples_per_second':  
100.302, 'eval_steps_per_second': 6.291, 'epoch': 3.0}
```

40%|██████████ | 1148/2870 [10:28<14:49, 1.94it/s]

```
{'loss': 0.8433, 'grad_norm': 132.46726989746094, 'learning_rate':  
1.2e-05, 'epoch': 4.0}
```

40%|██████████ | 1148/2870 [10:33<14:49, 1.94it/s]

```
{'eval_loss': 0.4499412178993225, 'eval_accuracy': 0.8310104529616724,  
'eval_f1': 0.8293476421040136, 'eval_recall': 0.8310104529616724,  
'eval_runtime': 5.7921, 'eval_samples_per_second': 99.101,  
'eval_steps_per_second': 6.215, 'epoch': 4.0}
```

50%|██████████ | 1435/2870 [13:05<11:38, 2.05it/s]

```
{'loss': 0.75, 'grad_norm': 57.15482711791992, 'learning_rate': 1e-05,  
'epoch': 5.0}
```

50%|██████████ | 1435/2870 [13:10<11:38, 2.05it/s]

```
{'eval_loss': 0.4259687066078186, 'eval_accuracy': 0.8344947735191638,  
'eval_f1': 0.8322147148059503, 'eval_recall': 0.8344947735191638,  
'eval_runtime': 5.3636, 'eval_samples_per_second': 107.018,  
'eval_steps_per_second': 6.712, 'epoch': 5.0}
```

60%|██████████ | 1722/2870 [15:42<09:55, 1.93it/s]

```
{'loss': 0.7303, 'grad_norm': 54.297908782958984, 'learning_rate':  
8.000000000000001e-06, 'epoch': 6.0}
```

```
60%|██████████ | 1722/2870 [15:47<09:55, 1.93it/s]
```

```
{'eval_loss': 0.48117595911026, 'eval_accuracy': 0.8292682926829268,  
'eval_f1': 0.8282083354697497, 'eval_recall': 0.8292682926829268,  
'eval_runtime': 5.4786, 'eval_samples_per_second': 104.771,  
'eval_steps_per_second': 6.571, 'epoch': 6.0}
```

```
70%|██████████ | 2009/2870 [18:17<07:18, 1.96it/s]
```

```
{'loss': 0.7013, 'grad_norm': 46.586769104003906, 'learning_rate': 6e-  
06, 'epoch': 7.0}
```

```
70%|██████████ | 2009/2870 [18:23<07:18, 1.96it/s]
```

```
{'eval_loss': 0.39135295152664185, 'eval_accuracy':  
0.8205574912891986, 'eval_f1': 0.8143466429524296, 'eval_recall':  
0.8205574912891986, 'eval_runtime': 5.742, 'eval_samples_per_second':  
99.965, 'eval_steps_per_second': 6.27, 'epoch': 7.0}
```

```
80%|██████████ | 2296/2870 [20:55<05:06, 1.87it/s]
```

```
{'loss': 0.6834, 'grad_norm': 137.60440063476562, 'learning_rate':  
4.000000000000001e-06, 'epoch': 8.0}
```

```
80%|██████████ | 2296/2870 [21:01<05:06, 1.87it/s]
```

```
{'eval_loss': 0.3996467590332031, 'eval_accuracy': 0.8536585365853658,  
'eval_f1': 0.8531528593185012, 'eval_recall': 0.8536585365853658,  
'eval_runtime': 5.4423, 'eval_samples_per_second': 105.47,  
'eval_steps_per_second': 6.615, 'epoch': 8.0}
```

```
90%|██████████ | 2583/2870 [23:34<02:19, 2.06it/s]
```

```
{'loss': 0.6825, 'grad_norm': 254.84799194335938, 'learning_rate':  
2.0000000000000003e-06, 'epoch': 9.0}
```

```
90%|██████████ | 2583/2870 [23:39<02:19, 2.06it/s]
```

```
{'eval_loss': 0.3744327127933502, 'eval_accuracy': 0.8745644599303136,  
'eval_f1': 0.8738582455671867, 'eval_recall': 0.8745644599303136,  
'eval_runtime': 5.3882, 'eval_samples_per_second': 106.529,  
'eval_steps_per_second': 6.681, 'epoch': 9.0}
```

```
100%|██████████ | 2870/2870 [26:17<00:00, 1.97it/s]
```

```
{'loss': 0.6537, 'grad_norm': 37.91458511352539, 'learning_rate': 0.0, 'epoch': 10.0}
```

```
100%|██████████| 2870/2870 [26:22<00:00, 1.97it/s]
```

```
{'eval_loss': 0.3659678101539612, 'eval_accuracy': 0.8693379790940766, 'eval_f1': 0.8695248003382784, 'eval_recall': 0.8693379790940766, 'eval_runtime': 5.7331, 'eval_samples_per_second': 100.12, 'eval_steps_per_second': 6.279, 'epoch': 10.0}
```

```
100%|██████████| 2870/2870 [26:25<00:00, 1.81it/s]
```

```
{'train_runtime': 1585.2813, 'train_samples_per_second': 28.966, 'train_steps_per_second': 1.81, 'train_loss': 1.0236623571309478, 'epoch': 10.0}
```

## Conclusión del entrenamiento

El modelo muestra una pérdida de entrenamiento que disminuye progresivamente desde 3.15 en la primera época hasta 0.65 en la última, indicando un aprendizaje consistente. En validación, la pérdida baja de 0.81 a 0.37, lo que respalda la mejora del desempeño. El accuracy y el F1 final alcanzan 86.93% y 86.95%, respectivamente, mientras que el recall sigue una tendencia similar, reflejando un buen equilibrio en las predicciones.

Aunque el rendimiento es bueno, los picos de gradiente elevados en épocas específicas (265.52 en la época 2 y 254.85 en la época 9) podrían ser indicadores de inestabilidad en el entrenamiento, lo que sugiere la necesidad de ajustar parámetros como el learning rate o la regularización.

Nota: Anteriormente, probamos con otros learning rate, como 1e-5 o 3e-5 pero el modelo performaba levemente peor. El que mejor rendimiento nos dio, fue un learning rate de 2e-5.

Guardamos el modelo y limpiamos la caché.

```
# Guardar el modelo
torch.save(model.state_dict(), 'modelo_entrenado_cvt_fine_tuning.pth')

# Limpieza de cache
torch.mps.empty_cache()
```

## Evaluación del modelo con Fine-Tuning

Creamos una función correspondientes a la evaluación del modelo que utiliza el reporte de clasificación, para evaluar ambos modelos.

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

```

def evaluate_model(model, test_dataset, batch_size_per_device=8,
device='cuda'):
    test_loader = DataLoader(test_dataset,
batch_size=batch_size_per_device)

    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for batch in test_loader:
            pixel_values = batch['pixel_values'].squeeze(1).float()
            outputs = model(pixel_values=pixel_values.to(device)) #
Forward pass
            predictions = torch.argmax(outputs.logits, dim=-1)

            # Almacenar las predicciones y etiquetas reales
            all_preds.extend(predictions.cpu().numpy())
            all_labels.extend(batch['label'].cpu().numpy())

    print(classification_report(all_labels, all_preds, digits=4))

```

Funciones para imprimir las predicciones y la matriz de confusión.

```

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_predictions(test_dataset, predictions, class_names,
num_samples=15, rows=3, cols=5):
    """
    Función para mostrar imágenes de un dataset con sus etiquetas
    verdaderas y predicciones.

    Parameters:
    - test_dataset: Dataset de prueba.
    - predictions: Resultados de las predicciones del modelo (con
    `trainer.predict()`).
    - class_names: Diccionario que mapea las etiquetas numéricas a
    nombres de clase.
    - num_samples: Número de imágenes a mostrar.
    - rows: Número de filas en la figura.
    - cols: Número de columnas en la figura.
    """
    predicted_labels =
    torch.argmax(torch.tensor(predictions.predictions), dim=-1)

    random_indices = random.sample(range(len(test_dataset)),
num_samples)

    fig, axes = plt.subplots(rows, cols, figsize=(15, 10)) # Tamaño

```

```

de la figura
axes = axes.flatten()

for idx, ax in zip(random_indices, axes):
    sample = test_dataset[idx]
    image = sample['pixel_values']
    true_label = int(sample['label'])

    true_label_name = class_names[true_label]
    pred_label_name = class_names[predicted_labels[idx].item()]

    # Ajusta la imagen a formato [H, W, C]
    image = image.permute(1, 2, 0).numpy() # cambia de [C, H, W]
a [H, W, C]

    # Normalizar la imagen si es necesario
    if image.max() > 1 or image.min() < 0:
        image = (image - image.min()) / (image.max() -
image.min()) # normalizar entre 0 y 1

    ax.imshow(image)
    ax.axis('off')
    ax.set_title(f"True: {true_label_name}\nPred:
{pred_label_name}", fontsize=10)

plt.tight_layout()
plt.show()

def plot_confusion_matrix(predictions, class_names):
    # Obtener las etiquetas verdaderas y las predicciones
    y_true = predictions.label_ids
    y_pred = predictions.predictions.argmax(axis=1)

    # Calcular la matriz de confusión
    cm = confusion_matrix(y_true, y_pred)

    # Etiquetas para las clases
    labels = [class_names[i] for i in range(len(class_names))]

    # Crear la visualización de la matriz de confusión
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=labels)

    disp.plot(cmap='plasma', xticks_rotation=45)
    plt.title("Matriz de Confusión")
    plt.show()

```

## Métricas para Test

Evaluamos el modelo:

```
evaluate_model(model, test_dataset_cvt, batch_size_per_device, device)
```

	precision	recall	f1-score	support
0	0.8750	0.2100	0.3387	100
1	0.5922	0.5304	0.5596	115
2	0.4367	0.9524	0.5988	105
3	0.8158	0.4189	0.5536	74
accuracy			0.5406	394
macro avg	0.6799	0.5279	0.5127	394
weighted avg	0.6645	0.5406	0.5129	394

## Reporte de clasificación:

- Accuracy General:

El accuracy obtenido en la evaluación del conjunto de validación es 54.06%, lo cual indica que el modelo tiene un desempeño aceptable pero podría mejorar. Tener en cuenta que la red fue preentrenada para clases muy distintas a las que esta intentando clasificar y fue entrenado solamente por 10 epochs.

- Desempeño por Clase:
  - Clase 0 (glioma\_tumor): Alta precisión (87.50%) pero bajo recall (21.00%), lo que sugiere que el modelo no está capturando correctamente esta clase y genera muchos **falsos negativos**.
  - Clase 1 (meningioma\_tumor): Moderada precisión (59.22%) y recall (53.04%), indicando un balance razonable aunque mejorable (moderados fp y fn).
  - Clase 2 (no\_tumor): Bajo precision (43.67%) pero alto recall (95.24%), lo que implica que el modelo clasifica casi todos los ejemplos como clase 2, pero con muchos **falsos positivos**.
  - Clase 3 (pituitary\_tumor): Alta precisión (81.58%) pero bajo recall (41.89%), mostrando un problema similar al de la clase 0 con **falsos negativos**.
- Promedios:
- Macro avg:

El F1-score promedio de 51.27% muestra que el desempeño general es moderado pero desigual entre las clases. Esto podría deberse a que el modelo tiene dificultades para capturar correctamente algunas características específicas de ciertas clases, lo que afecta su capacidad para generalizar.

- Weighted avg:

El F1-score ponderado de 51.29% indica que, aunque las clases no están significativamente desbalanceadas en términos de cantidad, el modelo aún favorece las clases con características más distintivas o que se ajustan mejor a los patrones aprendidos, lo que puede llevar a un desempeño inconsistente en otras clases.

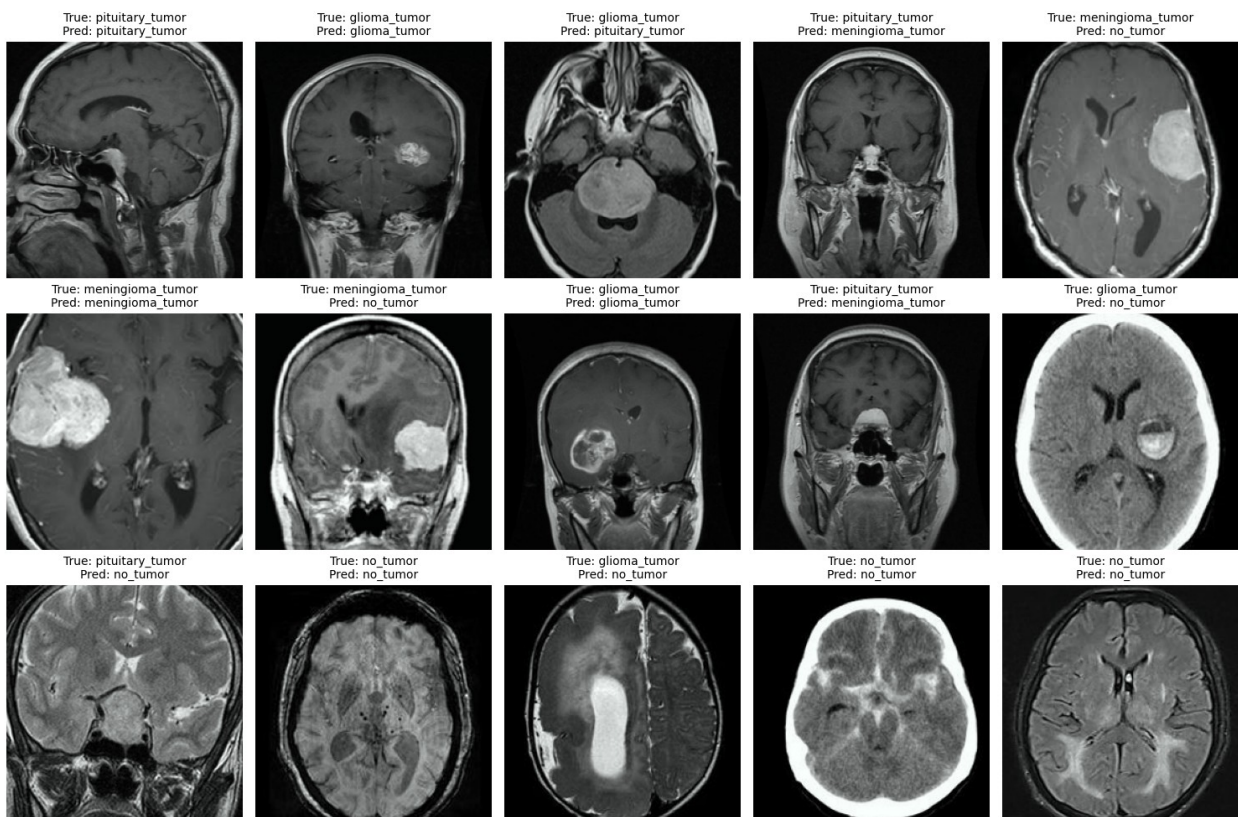
## Pruebas

Realizamos las pruebas pertinentes:

```
# Predicciones
predictions_cvt = trainer.predict(test_dataset_cvt)
predicted_labels_cvt =
torch.argmax(torch.tensor(predictions_cvt.predictions), dim=-1)

# Ploteo de imágenes con predicciones y ground truth (labels)
plot_predictions(test_dataset_cvt, predictions_cvt, class_names)

100%|██████████| 25/25 [00:03<00:00, 6.75it/s]
```



```
# Métricas asociadas a test segun predicciones
predictions_cvt.metrics

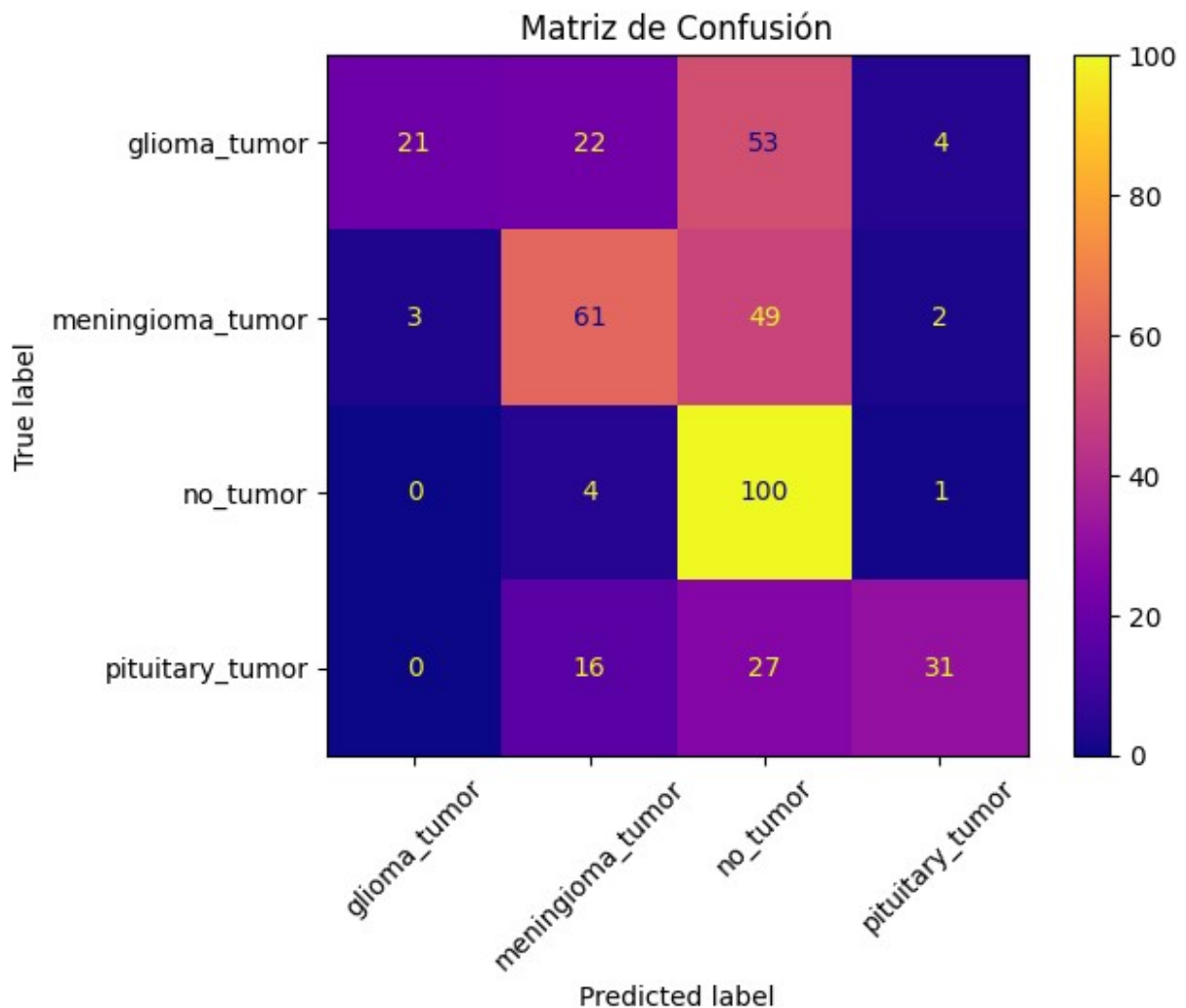
{'test_loss': 1.9361610412597656,
 'test_accuracy': 0.5406091370558376,
 'test_f1': 0.5128611754273218,}
```

```
'test_recall': 0.5406091370558376,  
'test_runtime': 4.1621,  
'test_samples_per_second': 94.664,  
'test_steps_per_second': 6.007}
```

## Matriz de confusión

Imprimimos los resultados de la predicción en una matriz de confusión.

```
plot_confusion_matrix(predictions_cvt, class_names)
```



## Conclusión

- Precisión Global (Accuracy): 54.06%
- F1-Score: 0.513
- Recall: 54.06%



## Matriz de Confusión:

Mejores resultados en la clase no\_tumor, con 100 instancias correctamente clasificadas, pero es frecuentemente confundida por otras clases, acumulando 129 falsos positivos.

Las clases glioma\_tumor y pituitary\_tumor presentan bajo rendimiento, con pocos verdaderos positivos y altos falsos negativos.

Las predicciones para meningioma\_tumor son razonables, pero todavía tienen errores significativos.

## Observaciones:

El modelo muestra un desempeño "aceptable", con su punto fuerte en la clase no\_tumor.

## Configuración del modelo ViT- Base (Google)

Repetimos los pasos anteriores, seteamos el modelo ViT y su feature extractor correspondiente.

```
from transformers import ViTImageProcessor, ViTForImageClassification

# Descargar modelo preentrando como su correspondiente procesador de
# datos
feature_extractor_g = ViTImageProcessor.from_pretrained("google/vit-
base-patch16-224")
model_g = ViTForImageClassification.from_pretrained("google/vit-base-
patch16-224")
```

Enviamos el modelo al dispositivo.

```
model_g.to(device) # Envío del modelo a device y visualización de
arquitectura
```

Aumentamos los datos reutilizando la función de *augmented\_train* y *transform* pasandole por argumento el feature extractor correspondiente al modelo ViT.

```
# Aumentación y transformación de datos en train_dataset
augmented_g = train_dataset_g.map(lambda item: augmented_train(item,
feature_extractor_g))

# Imputación de feature_extractor en en train, valid y test dataset
test_dataset_g = test_dataset_g.map(lambda item: transform(item,
feature_extractor_g))
val_dataset_g = val_dataset_g.map(lambda item: transform(item,
feature_extractor_g))
train_dataset_g = train_dataset_g.map(lambda item: transform(item,
feature_extractor_g))

# Concatenar train_dataset original con el augmented_g (data
```

```

aumentation)
train_dataset_g = concatenate_datasets([train_dataset_g, augmented_g])

Map: 0%|          | 0/2296 [00:00<?, ?
examples/s]/opt/anaconda3/envs/mipy/lib/python3.11/site-packages/trans
formers/image_processing_utils.py:41: UserWarning: The following named
arguments are not valid for `ViTImageProcessor.preprocess` and were
ignored: 'padding'
    return self.preprocess(images, **kwargs)
Map: 100%|██████████| 2296/2296 [00:50<00:00, 45.22 examples/s]

```

Corroboramos la cantidad de datos luego de realizar la aumentación.

```

# Inspección datasets luego de aumentación y procesamiento de datos
print(f"Tamaño del conjunto de entrenamiento después del aumento de
datos: {len(train_dataset_g)}")
print(f"Tamaño del conjunto de validación: {len(val_dataset_g)}")
print(f"Tamaño del conjunto de prueba: {len(test_dataset_g)}")

Tamaño del conjunto de entrenamiento después del aumento de datos:
4592
Tamaño del conjunto de validación: 574
Tamaño del conjunto de prueba: 394

# Seteo de datos torch para consumo de ViT
train_dataset_g.set_format(type='torch', columns=['pixel_values',
'label'])
val_dataset_g.set_format(type='torch', columns=['pixel_values',
'label'])
test_dataset_g.set_format(type='torch', columns=['pixel_values',
'label'])

```

Configuramos el Trainer, con la particularidad de que reutilizamos los mismos argumentos de entrenamiento y los hiperparámetros que se utilizaron para el modelo CvT.

```

# Seteo de Clase Trainer de Hugging Face
trainer_g = Trainer(
    model=model_g, # Modelo a entrenar
    args=training_args, # Parámetros de entrenamiento
    train_dataset=train_dataset_g, # Conjunto de entrenamiento
    eval_dataset=val_dataset_g, # Conjunto de validación
    tokenizer=feature_extractor_g, # Tokenizer o extractor de
características
    compute_metrics=compute_metrics, # Función de evaluación
)

/var/folders/ml/g_sxsst10mj890krz6gpp0980000gn/T/
ipykernel_28838/470177897.py:2: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for

```

```
`Trainer.__init__`. Use `processing_class` instead.  
trainer_g = Trainer()
```

```
# Entrenamiento
```

```
trainer_g_results = trainer_g.train()
```

```
10%|██████████| 287/2870 [03:51<33:44, 1.28it/s]
```

```
{'loss': 1.5294, 'grad_norm': 201.31008911132812, 'learning_rate':  
1.8e-05, 'epoch': 1.0}
```

```
10%|██████████| 287/2870 [04:02<33:44, 1.28it/s]
```

```
{'eval_loss': 1.0532342195510864, 'eval_accuracy': 0.5400696864111498,  
'eval_f1': 0.5380861173321471, 'eval_recall': 0.5400696864111498,  
'eval_runtime': 11.278, 'eval_samples_per_second': 50.895,  
'eval_steps_per_second': 3.192, 'epoch': 1.0}
```

```
20%|██████████| 574/2870 [07:42<28:22, 1.35it/s]
```

```
{'loss': 0.6266, 'grad_norm': 611.5048828125, 'learning_rate':  
1.6000000000000003e-05, 'epoch': 2.0}
```

```
20%|██████████| 574/2870 [07:54<28:22, 1.35it/s]
```

```
{'eval_loss': 0.8178244233131409, 'eval_accuracy': 0.6794425087108014,  
'eval_f1': 0.677919349173554, 'eval_recall': 0.6794425087108014,  
'eval_runtime': 11.1619, 'eval_samples_per_second': 51.425,  
'eval_steps_per_second': 3.225, 'epoch': 2.0}
```

```
30%|██████████| 861/2870 [11:29<23:44, 1.41it/s]
```

```
{'loss': 0.5293, 'grad_norm': 206.276123046875, 'learning_rate': 1.4e-  
05, 'epoch': 3.0}
```

```
30%|██████████| 861/2870 [11:39<23:44, 1.41it/s]
```

```
{'eval_loss': 0.7737617492675781, 'eval_accuracy': 0.6550522648083623,  
'eval_f1': 0.6610120219869867, 'eval_recall': 0.6550522648083623,  
'eval_runtime': 9.9429, 'eval_samples_per_second': 57.73,  
'eval_steps_per_second': 3.621, 'epoch': 3.0}
```

```
40%|██████████| 1148/2870 [15:10<20:10, 1.42it/s]
```

```
{'loss': 0.4335, 'grad_norm': 714.2328491210938, 'learning_rate':  
1.2e-05, 'epoch': 4.0}
```

```
40%|██████████| 1148/2870 [15:20<20:10, 1.42it/s]
```

```
{'eval_loss': 0.7824910283088684, 'eval_accuracy': 0.6411149825783972,  
'eval_f1': 0.6536276338930265, 'eval_recall': 0.6411149825783972,  
'eval_runtime': 9.9617, 'eval_samples_per_second': 57.621,  
'eval_steps_per_second': 3.614, 'epoch': 4.0}
```

```
50%|███████ | 1435/2870 [18:52<16:59, 1.41it/s]
```

```
{'loss': 0.3608, 'grad_norm': 334.5968017578125, 'learning_rate': 1e-  
05, 'epoch': 5.0}
```

```
50%|███████ | 1435/2870 [19:01<16:59, 1.41it/s]
```

```
{'eval_loss': 0.6785945296287537, 'eval_accuracy': 0.7491289198606271,  
'eval_f1': 0.7418097792452242, 'eval_recall': 0.7491289198606271,  
'eval_runtime': 9.8373, 'eval_samples_per_second': 58.349,  
'eval_steps_per_second': 3.66, 'epoch': 5.0}
```

```
60%|███████ | 1722/2870 [22:34<13:46, 1.39it/s]
```

```
{'loss': 0.3052, 'grad_norm': 753.0388793945312, 'learning_rate':  
8.000000000000001e-06, 'epoch': 6.0}
```

```
60%|███████ | 1722/2870 [22:44<13:46, 1.39it/s]
```

```
{'eval_loss': 0.5791086554527283, 'eval_accuracy': 0.7526132404181185,  
'eval_f1': 0.7342046716623726, 'eval_recall': 0.7526132404181185,  
'eval_runtime': 9.7849, 'eval_samples_per_second': 58.662,  
'eval_steps_per_second': 3.679, 'epoch': 6.0}
```

```
70%|███████ | 2009/2870 [26:16<10:19, 1.39it/s]
```

```
{'loss': 0.2647, 'grad_norm': 1142.01171875, 'learning_rate': 6e-06,  
'epoch': 7.0}
```

```
70%|███████ | 2009/2870 [26:26<10:19, 1.39it/s]
```

```
{'eval_loss': 0.494354248046875, 'eval_accuracy': 0.7961672473867596,  
'eval_f1': 0.7925908740661939, 'eval_recall': 0.7961672473867596,  
'eval_runtime': 10.2834, 'eval_samples_per_second': 55.818,  
'eval_steps_per_second': 3.501, 'epoch': 7.0}
```

```
80%|███████ | 2296/2870 [29:58<06:40, 1.43it/s]
```

```
{'loss': 0.2225, 'grad_norm': 716.905517578125, 'learning_rate':  
4.000000000000001e-06, 'epoch': 8.0}
```

```
80%|███████ | 2296/2870 [30:08<06:40, 1.43it/s]
```

```
{'eval_loss': 0.4857077896595001, 'eval_accuracy': 0.7996515679442509,
'eval_f1': 0.7928509806534433, 'eval_recall': 0.7996515679442509,
'eval_runtime': 9.8277, 'eval_samples_per_second': 58.406,
'eval_steps_per_second': 3.663, 'epoch': 8.0}
```

```
90%|██████████ | 2583/2870 [33:41<03:52, 1.23it/s]
```

```
{'loss': 0.1987, 'grad_norm': 1305.9378662109375, 'learning_rate':
2.0000000000000003e-06, 'epoch': 9.0}
```

```
90%|██████████ | 2583/2870 [33:51<03:52, 1.23it/s]
```

```
{'eval_loss': 0.47420552372932434, 'eval_accuracy':
0.8292682926829268, 'eval_f1': 0.8291776130340868, 'eval_recall':
0.8292682926829268, 'eval_runtime': 9.9426, 'eval_samples_per_second':
57.732, 'eval_steps_per_second': 3.621, 'epoch': 9.0}
```

```
100%|██████████| 2870/2870 [37:18<00:00, 1.46it/s]
```

```
{'loss': 0.1829, 'grad_norm': 111.94017028808594, 'learning_rate':
0.0, 'epoch': 10.0}
```

```
100%|██████████| 2870/2870 [37:28<00:00, 1.46it/s]
```

```
{'eval_loss': 0.4800032377243042, 'eval_accuracy': 0.8222996515679443,
'eval_f1': 0.8205762606719182, 'eval_recall': 0.8222996515679443,
'eval_runtime': 9.8422, 'eval_samples_per_second': 58.32,
'eval_steps_per_second': 3.658, 'epoch': 10.0}
```

```
100%|██████████| 2870/2870 [37:30<00:00, 1.28it/s]
```

```
{'train_runtime': 2250.3326, 'train_samples_per_second': 20.406,
'train_steps_per_second': 1.275, 'train_loss': 0.4653672886227066,
'epoch': 10.0}
```

## Conclusión del entrenamiento:

El modelo ViT muestra un desempeño limitado durante el entrenamiento, alcanzando una precisión final de 45.69% y un F1-score promedio de 37.78% en el conjunto de prueba. Esto indica que el modelo tiene dificultades para balancear precisión y recall en varias clases, particularmente en aquellas con menos representaciones en el dataset.

Además, se observa un comportamiento inestable en las normativas de gradiente durante el entrenamiento, con valores elevados en ciertas épocas que podrían indicar problemas de optimización. Aunque no se detallan explícitamente en este caso, estos picos podrían haber afectado la convergencia adecuada del modelo.

## Evaluación del modelo con Fine-Tuning

### Métricas de Evaluación

```
evaluate_model(model_g, test_dataset_g, batch_size_per_device, device)
```

	precision	recall	f1-score	support
0	0.8125	0.1300	0.2241	100
1	0.4878	0.5217	0.5042	115
2	0.4145	0.9238	0.5723	105
3	0.4762	0.1351	0.2105	74
accuracy			0.4569	394
macro avg	0.5478	0.4277	0.3778	394
weighted avg	0.5485	0.4569	0.3961	394

### Reporte de clasificación

- Accuracy General:

El accuracy obtenido es 45.69%, lo que refleja un desempeño limitado del modelo y sugiere que necesita mejoras.

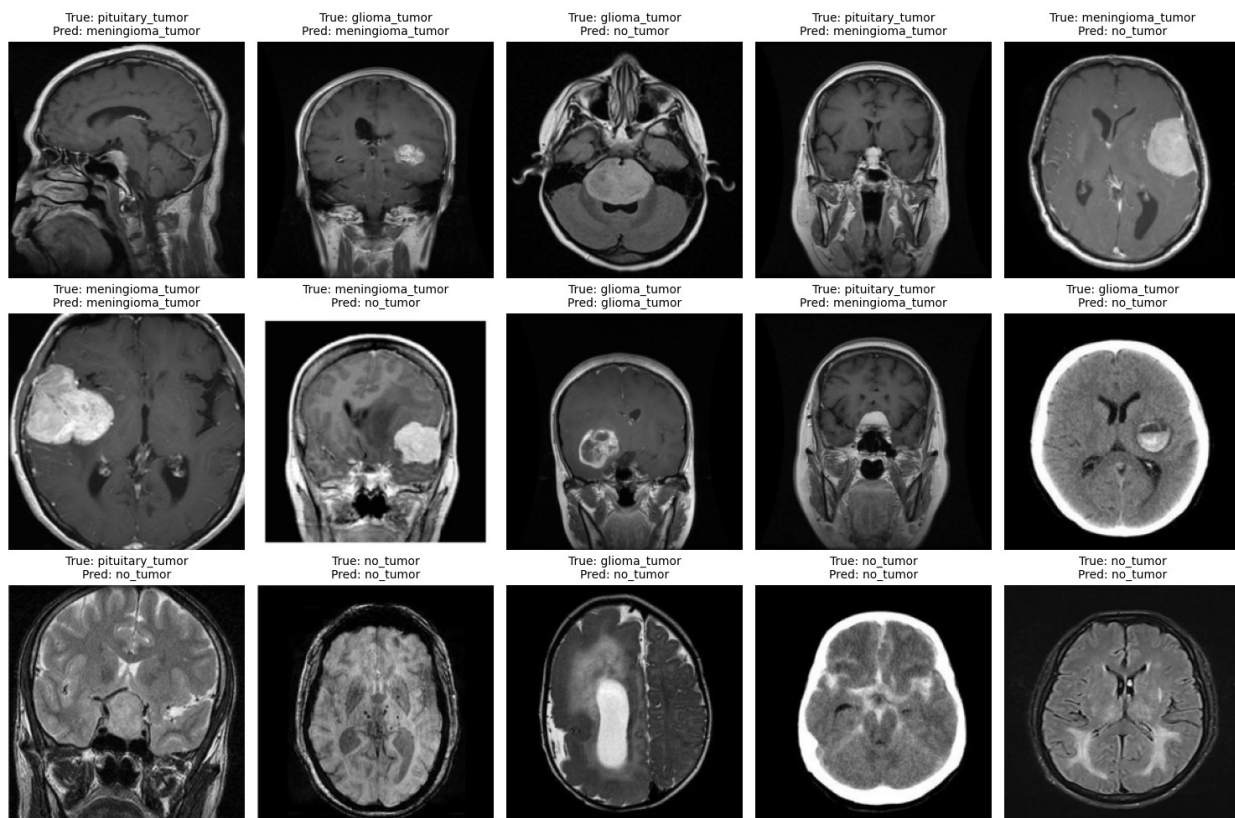
- Desempeño por Clase:
  - Clase 0 (glioma\_tumor): Precisión moderada (81.25%) pero recall muy bajo (13.00%), lo que indica que el modelo tiene dificultades significativas para identificar ejemplos de esta clase, generando muchos **falsos negativos**.
  - Clase 1 (meningioma\_tumor): Precisión moderada (48.78%) y recall razonable (52.17%), lo que muestra un desempeño algo balanceado, aunque todavía con falsos negativos y positivos considerables.
  - Clase 2 (no\_tumor): Precisión baja (41.45%) pero recall muy alto (92.38%), lo que indica que el modelo clasifica muchos ejemplos como pertenecientes a esta clase, generando un alto número de **falsos positivos**.
  - Clase 3 (pituitary\_tumor): Precisión baja (47.62%) y recall muy bajo (13.51%), lo que implica que el modelo tiene dificultades para identificar esta clase, generando un alto número de **falsos negativos**.

### Pruebas

```
# Predicciones
predictions_g = trainer_g.predict(test_dataset_g) # Error en set
utilizado, levantar pesos o correr modelo y correr set correcto
predicted_labels_g =
torch.argmax(torch.tensor(predictions_g.predictions), dim=-1)

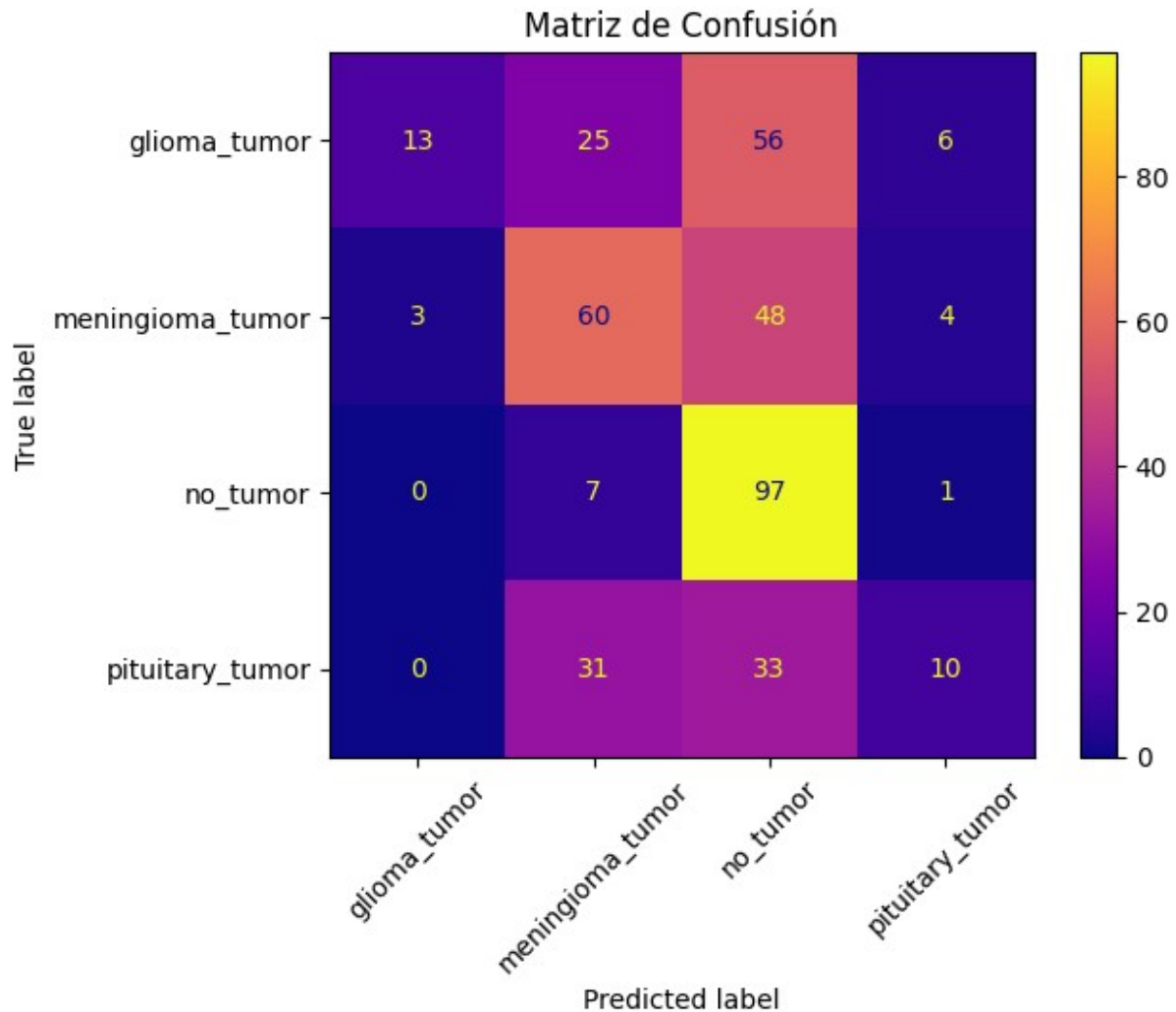
# Ploteo de predicciones y ground truth (labels)
plot_predictions(test_dataset_g, predictions_g, class_names)
```

100% | 25/25 [00:06<00:00, 4.03it/s]



Matriz de confusión

```
plot_confusion_matrix(predictions_g, class_names)
```



### Resultados de ViT de Google:

- Precisión Global (Accuracy): 45.69%
- F1-Score: 0.378 (macro avg)
- Recall Promedio: 42.77%

### Matriz de Confusión:

Mejores resultados en la clase no\_tumor, con 97 instancias correctamente clasificadas y pocos falsos negativos (8), aunque sigue acumulando un alto número de falsos positivos (137), lo que refleja confusión frecuente desde otras clases, especialmente glioma\_tumor y meningioma\_tumor

Las clases glioma\_tumor y pituitary\_tumor tienen un desempeño bajo, con muy pocos verdaderos positivos, 13 y 10, respectivamente y altos falsos negativos, además de ser confundidas frecuentemente con no\_tumor y entre ellas.



La clase meningioma\_tumor tiene resultados moderados, con 60 verdaderos positivos, pero muestra confusión significativa tanto como falsos negativos hacia no\_tumor (48) como falsos positivos desde glioma\_tumor y pituitary\_tumor.

### Observaciones:

El modelo muestra un desempeño "aceptable", con su punto fuerte en la clase no\_tumor.

## Conclusión General: CvT supera a ViT

Comparando el desempeño de ambos modelos, CvT (Convolutional Vision Transformer) demuestra ser significativamente mejor que ViT (Vision Transformer) en esta tarea de clasificación de tumores cerebrales.

Razones por las cuales CvT es superior:

- Mejor Equilibrio en Falsos Negativos y Falsos Positivos:
  - CvT: Muestra un mejor recall en todas las clases, reduciendo la cantidad de falsos negativos (ejemplos positivos que el modelo clasifica como negativos). Esto es evidente en la clase no\_tumor, donde CvT clasifica correctamente 100 ejemplos con solo 5 FP (predicciones incorrectas como no tumor). Esto es crucial, ya que minimizar FN en diagnósticos médicos evita que casos graves queden sin detectar. Aunque CvT tiene más falsos positivos en ciertas clases (como glioma\_tumor), esto es más aceptable en entornos médicos, ya que un falso positivo lleva a revisiones adicionales, pero no a ignorar un caso crítico.
  - ViT: Genera una cantidad significativamente mayor de falsos negativos, especialmente en la clase glioma\_tumor, con un recall de solo 13% (87 de cada 100 casos positivos no son detectados). Este error es grave porque puede llevar a que tumores importantes no sean diagnosticados. Además, la alta cantidad de falsos positivos en clases como meningioma\_tumor y no\_tumor indica que ViT tiene dificultades para diferenciar entre clases.
- Mejor Precisión General:
  - CvT alcanza una precisión general del 54.06%, mientras que ViT solo logra 45.69%. Esto indica que CvT clasifica correctamente un mayor porcentaje de ejemplos, lo que sugiere que su capacidad de generalización es superior.
- Mayor F1-Score:
  - El F1-score promedio de CvT es de 51.28%, en comparación con el 37.78% de ViT. Esto significa que CvT mantiene un mejor equilibrio entre precisión y recall.
- Rendimiento por Clase:
  - CvT logra un mejor desempeño en la mayoría de las clases (como se observa en la matriz de confusión), con un recall más uniforme en las categorías, especialmente en la clase no\_tumor (100 ejemplos correctamente clasificados).
  - En contraste, ViT muestra problemas significativos en clases como glioma\_tumor (recall del 13.00%) y pituitary\_tumor (recall del 13.51%), generando demasiados falsos negativos.
- Eficiencia de Arquitectura:

- CvT combina convoluciones con mecanismos de atención, lo que le permite capturar mejor las relaciones locales (útiles para datos como imágenes médicas) y globales. ViT, al carecer de convoluciones, depende únicamente de los mecanismos de atención global, lo que puede dificultar el aprendizaje de características locales importantes en datasets más pequeños.
- Estabilidad en el Entrenamiento:
  - CvT presenta menos inestabilidad en las normativas de gradiente, mientras que ViT muestra signos de inestabilidad en la optimización (posiblemente debido a su mayor dependencia de la calidad del preprocesamiento y la inicialización de los pesos).
- Clase no\_tumor como muestra negativa:
  - La clase no\_tumor puede considerarse como una "muestra negativa". Aunque la clase no\_tumor no está sobrerrepresentada, sus muestras siguen siendo significativas en el conjunto de datos. Esto puede hacer que el modelo se optimice en gran medida para reconocer esta categoría, afectando las métricas generales. Las clases tumorales pueden confundirse con no\_tumor (se puede observar esto en ambos modelos), lo que aumenta los falsos positivos y deteriora las métricas, como la precisión y el F1-score, en las clases tumorales. Una posible mejora sería entrenar el modelo solo con las clases tumorales o reducir la proporción de no\_tumor, lo que podría mejorar la diferenciación entre las clases tumorales.