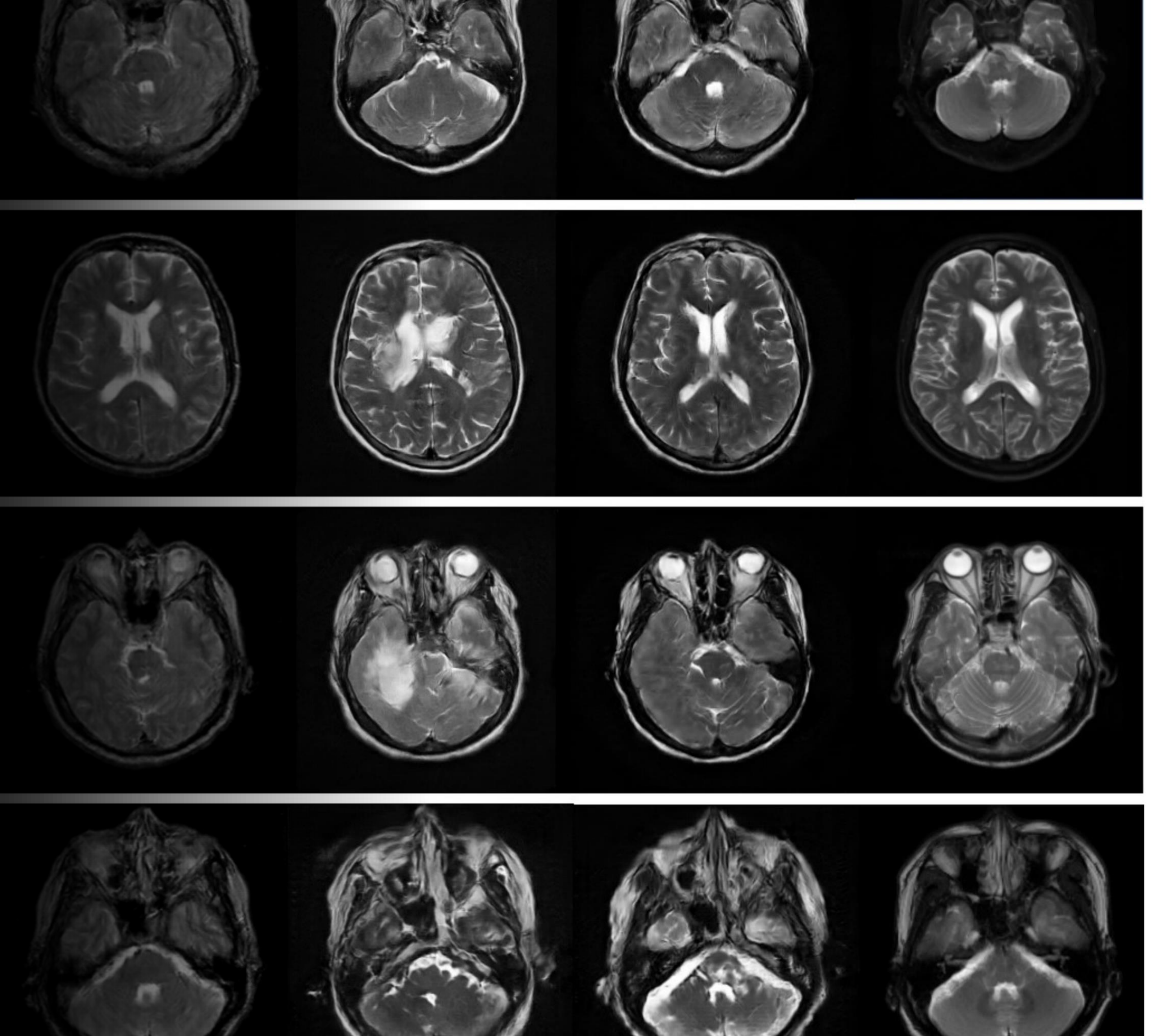
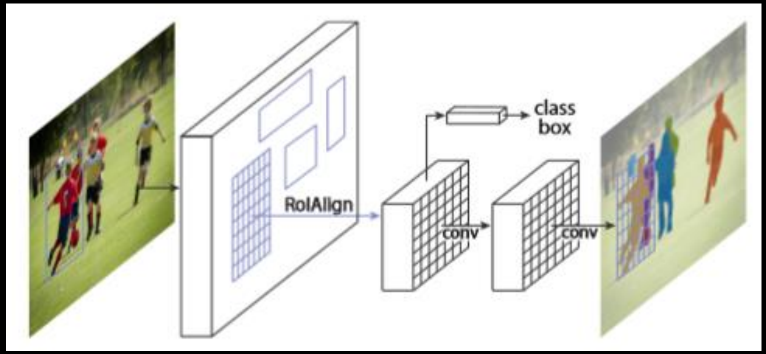
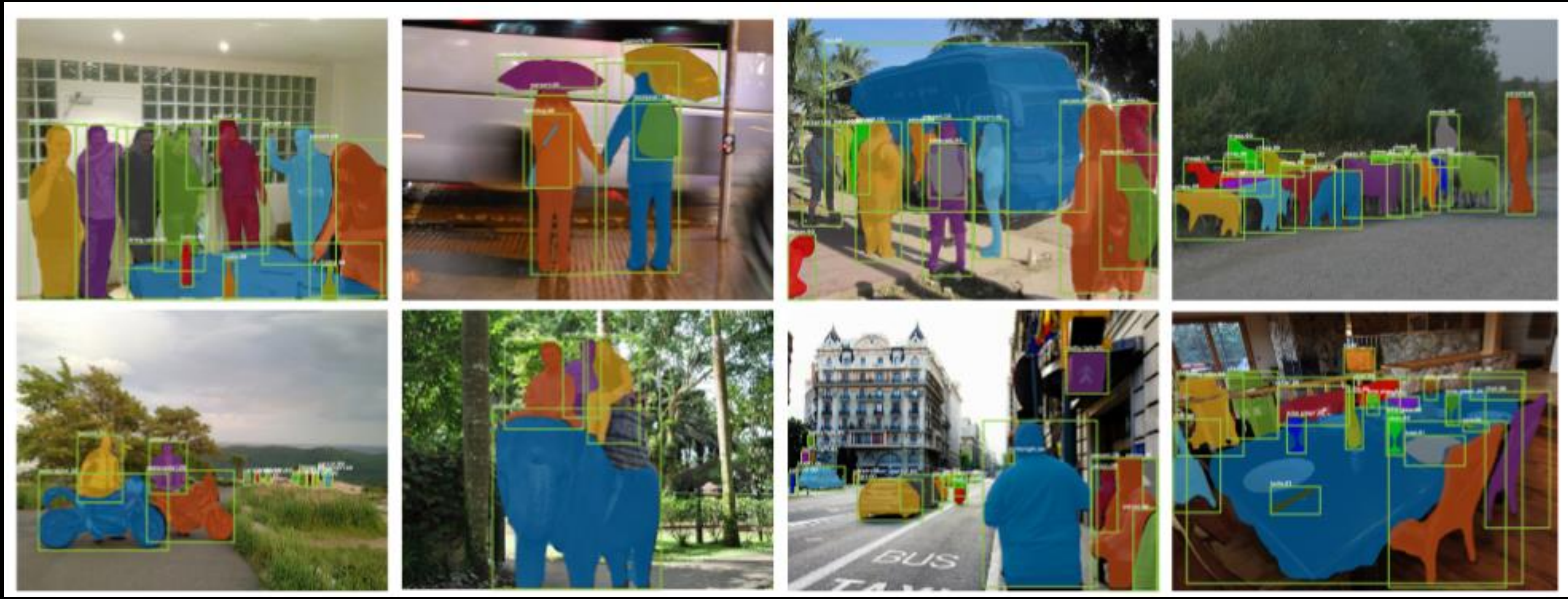
Detección de tumores cerebrales en IRM usando MASK R-CNN

- Expositores:
    - Gonzalo Bello
    - Simón Sánchez



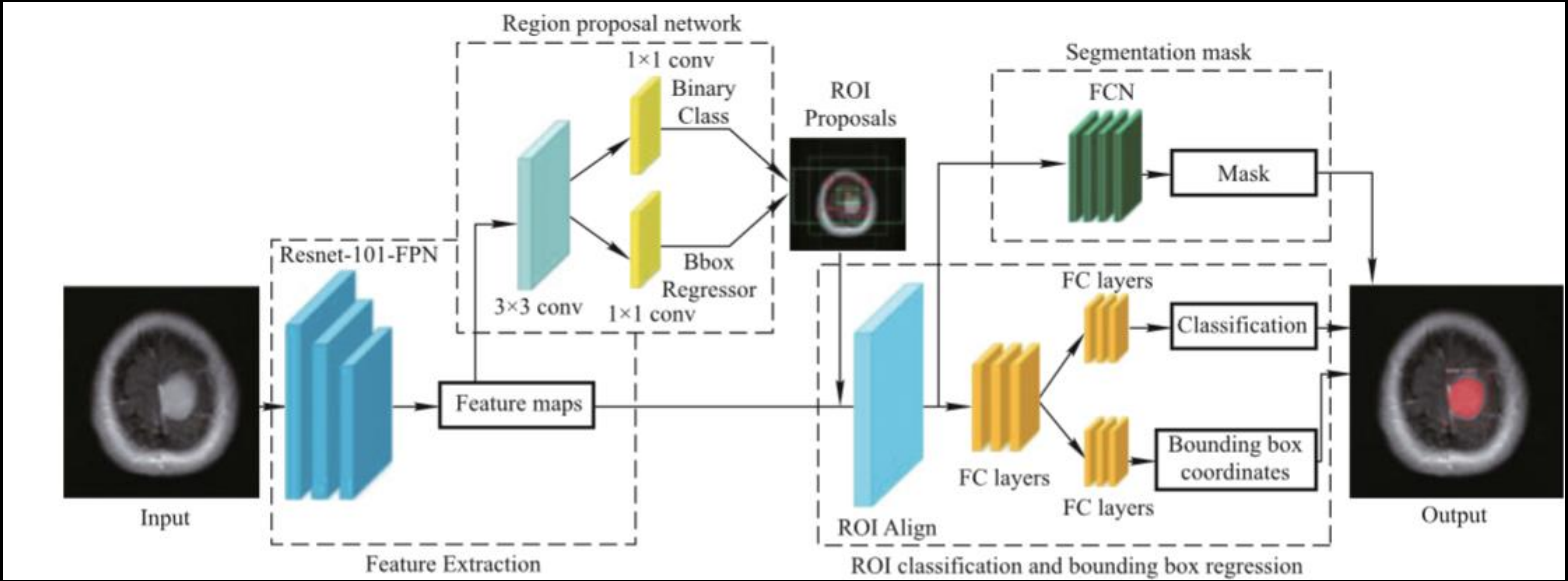
Modelo MASK R-CNN





Modelo MASK R-CNN

Arquitectura:



Implementación del Código

Usando Python en Google Colab con GPU

Librerías y Pre-entrenamiento (COCO: Common Objects in Context)

```python
import numpy as np
import json
import skimage.draw
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, TensorBoard
from mrcnn.config import Config
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize

import sys
# Directorio raíz del proyecto
ROOT_DIR = os.path.abspath('.')
print(f"Directorio raíz del proyecto: {ROOT_DIR}")

# Importar Mask RCNN
sys.path.append(ROOT_DIR)

# Importar configuración COCO
sys.path.append(os.path.join(ROOT_DIR, 'samples/coco/'))
import coco
```

Configuración del Modelo

```python
class TumorConfig(Config):
    NAME = 'tumor_detector'
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
    NUM_CLASSES = 1 + 1
    DETECTION_MIN_CONFIDENCE = 0.85
    STEPS_PER_EPOCH = 100
    LEARNING_RATE = 0.001

config = TumorConfig()
config.display()
```

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        35
DETECTION_MIN_CONFIDENCE       0.85
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  1024
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           tumor_detector
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                100
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  False
USE_RPN_ROIS                   True
VALIDATION_STEPS               10
WEIGHT_DECAY                   0.0001
```

Implementación del Dataset para Detección y Segmentación de Tumores Cerebrales

```python
class BrainScanDataset(utils.Dataset):

    def load_brain_scan(self, dataset_dir, subset):
        """Load a subset of the dataset."""
        self.add_class("tumor", 1, "tumor")

        assert subset in ["train", "val", "test"]
        dataset_dir = os.path.join(dataset_dir, subset)

        annotations_path = os.path.join(dataset_dir, f'annotations_{subset}.json')
        try:
            with open(annotations_path) as f:
                annotations = json.load(f)
        except FileNotFoundError:
            raise FileNotFoundError(f"Annotations file not found: {annotations_path}")
        except json.JSONDecodeError:
            raise ValueError(f"Error decoding JSON from the file: {annotations_path}")

        annotations = list(annotations.values())
        annotations = [a for a in annotations if a['regions']]

        for a in annotations:
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in a['regions']]

            image_path = os.path.join(dataset_dir, a['filename'])
            try:
                image = skimage.io.imread(image_path)
            except FileNotFoundError:
                print(f"Warning: Image file not found: {image_path}")
                continue
            except Exception as e:
                print(f"Warning: Error reading image file {image_path}: {e}")
                continue
```

```python
            height, width = image.shape[:2]

            self.add_image(
                "tumor",
                image_id=a['filename'],
                path=image_path,
                width=width,
                height=height,
                polygons=polygons
            )

    def load_mask(self, image_id):
        """Generate instance masks for an image."""
        image_info = self.image_info[image_id]
        if image_info["source"] != "tumor":
            return super(self.__class__, self).load_mask(image_id)

        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])], dtype=np.uint8)

        for i, p in enumerate(info["polygons"]):
            all_points_y = np.array(p['all_points_y'])
            all_points_x = np.array(p['all_points_x'])

            # Validate coordinates
            all_points_y = np.clip(all_points_y, 0, info["height"] - 1)
            all_points_x = np.clip(all_points_x, 0, info["width"] - 1)

            rr, cc = skimage.draw.polygon(all_points_y, all_points_x)
            mask[rr, cc, i] = 1

        # Utilize the function from utils to convert mask to boolean
        return mask.astype(np.bool_), np.ones([mask.shape[-1]], dtype=np.int32)

    def image_reference(self, image_id):
        info = self.image_info[image_id]
        if info["source"] == "tumor":
            return info["path"]
        else:
            return super(self.__class__, self).image_reference(image_id)
```

Datasets y Callbacks

```python
# Crear los datasets
dataset_train = BrainScanDataset()
dataset_train.load_brain_scan('brain-tumor/data_cleaned', 'train')
dataset_train.prepare()

dataset_val = BrainScanDataset()
dataset_val.load_brain_scan('brain-tumor/data_cleaned', 'val')
dataset_val.prepare()

dataset_test = BrainScanDataset()
dataset_test.load_brain_scan('brain-tumor/data_cleaned', 'test')
dataset_test.prepare()
```

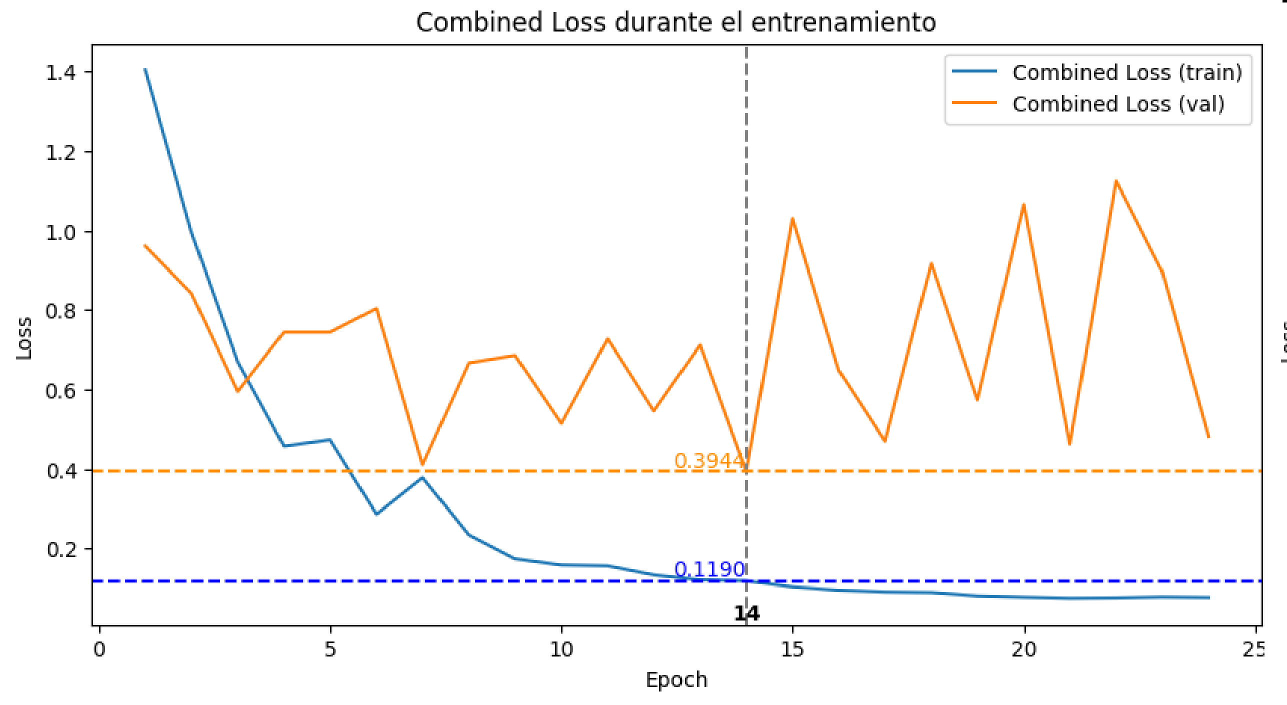
```python
# Crear el directorio para guardar checkpoints
CHECKPOINT_DIR = os.path.abspath('./brain-tumor/checkpoints')

os.makedirs(CHECKPOINT_DIR, exist_ok=True)
os.makedirs('./brain-tumor/logs', exist_ok=True)

# # Configuración de callbacks
tensorboard_callback = TensorBoard(log_dir='./brain-tumor/logs', histogram_freq=0)
checkpoint_callback = ModelCheckpoint(filepath=os.path.join(CHECKPOINT_DIR,
                                                            'weights.{epoch:02d}-{val_loss:.2f}.h5'), save_weights_only=True)
reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# # Lista de callbacks
callbacks = [tensorboard_callback,
             checkpoint_callback, reduce_lr_callback, early_stopping_callback]
```

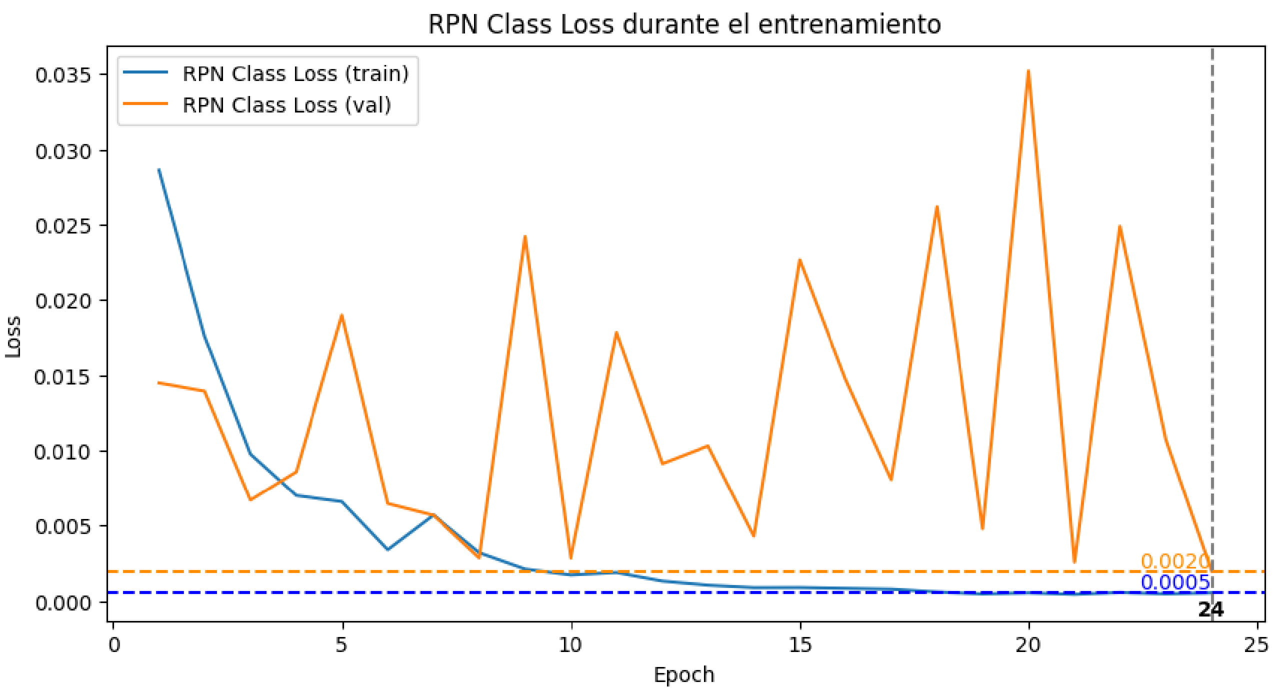
Entrenamiento del Modelo

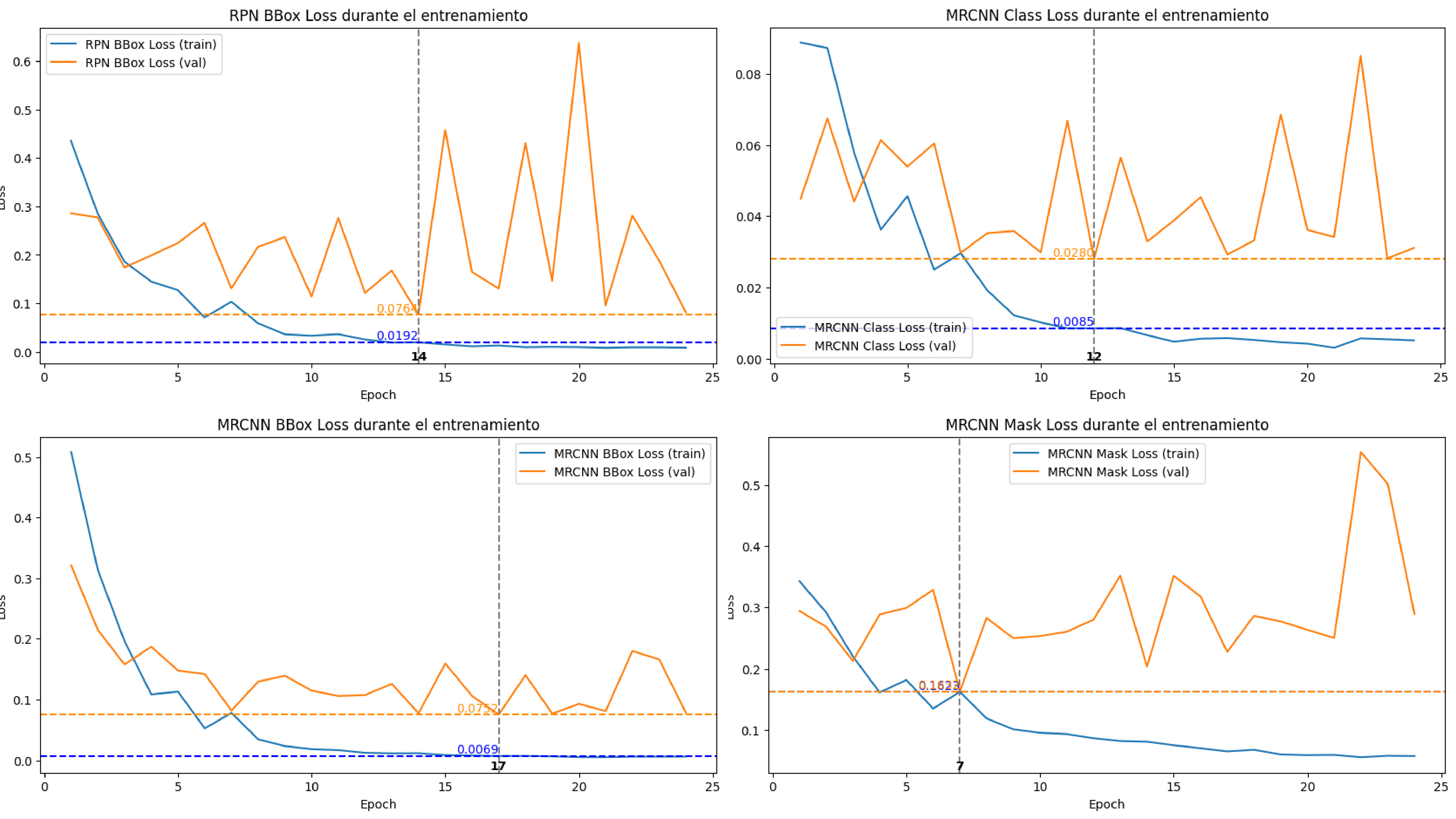
```
# # Configurar el modelo para entrenamiento
model = modellib.MaskRCNN(
    mode='training',
    config=config,
    model_dir=CHECKPOINT_DIR
)

model.load_weights(
    os.path.join(ROOT_DIR, "mask_rcnn_coco.h5"),
    by_name=True,
    exclude=["mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox", "mrcnn_mask"]
)

# # Entrenar el modelo por 30 época
num_epochs = 30

model.train(
    dataset_train,
    dataset_val,
    learning_rate=config.LEARNING_RATE,
    epochs=num_epochs,
    layers='all',
    custom_callbacks=callbacks
)
```

Gráficos de Pérdida durante el entrenamiento

```python
# Iterar sobre los pares de tags para generar los gráficos
for train_tag, val_tag, title in tags_pairs:
    # Extraer los datos de las métricas de interés
    epoch_train_steps, epoch_train_values = get_scalar_data(event_acc, train_tag)
    epoch_val_steps, epoch_val_values = get_scalar_data(event_acc, val_tag)

    # Encuentra la época con la pérdida de validación mínima
    epoch_optimal = np.argmin(epoch_val_values)

    # Graficar la pérdida
    plt.figure(figsize=(10, 5))
    plt.plot([x + 1 for x in epoch_train_steps], epoch_train_values, label=f'{title} (train)')
    plt.plot([x + 1 for x in epoch_val_steps], epoch_val_values, label=f'{title} (val)')
```






RPN BBox Loss durante el entrenamiento
RPN BBox Loss (train)
RPN BBox Loss (val)
0.0764
0.0192
14
Epoch
MRCNN Class Loss durante el entrenamiento
MRCNN Class Loss (train)
MRCNN Class Loss (val)
0.0280
0.0085
12
Epoch
MRCNN BBox Loss durante el entrenamiento
MRCNN BBox Loss (train)
MRCNN BBox Loss (val)
0.0752
0.0069
17
Epoch
MRCNN Mask Loss durante el entrenamiento
MRCNN Mask Loss (train)
MRCNN Mask Loss (val)
0.1623
7
Epoch

Configuración del Modelo en Modo Inferencia

```python
# Configurar el modelo para inferencia (desactivar el entrenamiento)
model = modellib.MaskRCNN(
    mode="inference",             # Modo inferencia para realizar predicciones
    config=config,                # Configuración específica para el proyecto
    model_dir=CHECKPOINT_DIR      # Directorio donde se guardan los pesos del modelo
)

# Obtener la ruta a los pesos guardados más recientes (mejor modelo entrenado)
model_path = '/content/drive/MyDrive/proyecto_dl/brain-tumor/checkpoints/weights.14-0.3944.h5'

# Cargar los pesos en el modelo utilizando solo los nombres de las capas que coinciden
model.load_weights(model_path, by_name=True)
```

Predicción y Visualización

```python
# Definir la función para predecir y visualizar
def get_ax(rows=1, cols=1, size=16):
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax

def display_image(dataset, ind):
    plt.figure(figsize=(5,5))
    plt.imshow(dataset.load_image(ind))
    plt.xticks([])
    plt.yticks([])
    plt.title('Original Image')
    plt.show()

def predict_and_plot_differences(dataset, img_id):
    original_image, image_meta, gt_class_id, gt_bbox, gt_mask = modellib.load_image_gt(
        dataset, config, img_id, augmentation=None
    )

    results = model.detect([original_image], verbose=0)
    r = results[0]

    visualize.display_instances(
        original_image,
        r['rois'], r['masks'], r['class_ids'],
        dataset.class_names, r['scores'],
        ax=get_ax()
    )
```

```python
# Se define funcion para ver las anotaciones reales y poder comparar
def display_real_annotations(dataset, img_id):
    # Cargar la imagen y las anotaciones reales
    original_image, image_meta, gt_class_id, gt_bbox, gt_mask = modellib.load_image_gt(
        dataset, config, img_id)

    # Visualizar las anotaciones reales
    visualize.display_instances(
        original_image,
        gt_bbox,
        gt_mask,
        gt_class_id,
        dataset.class_names,
        scores=None,  # No scores for ground truth
        title="Ground Truth",
        ax=get_ax()
    )
```

```python
# Evaluar y visualizar resultados
ind = 0 # aquí poner el número dentro del rango de las imagenes de evaluacion
display_image(dataset_test, ind)
display_real_annotations(dataset_test, ind)
predict_and_plot_differences(dataset_test, ind)
```

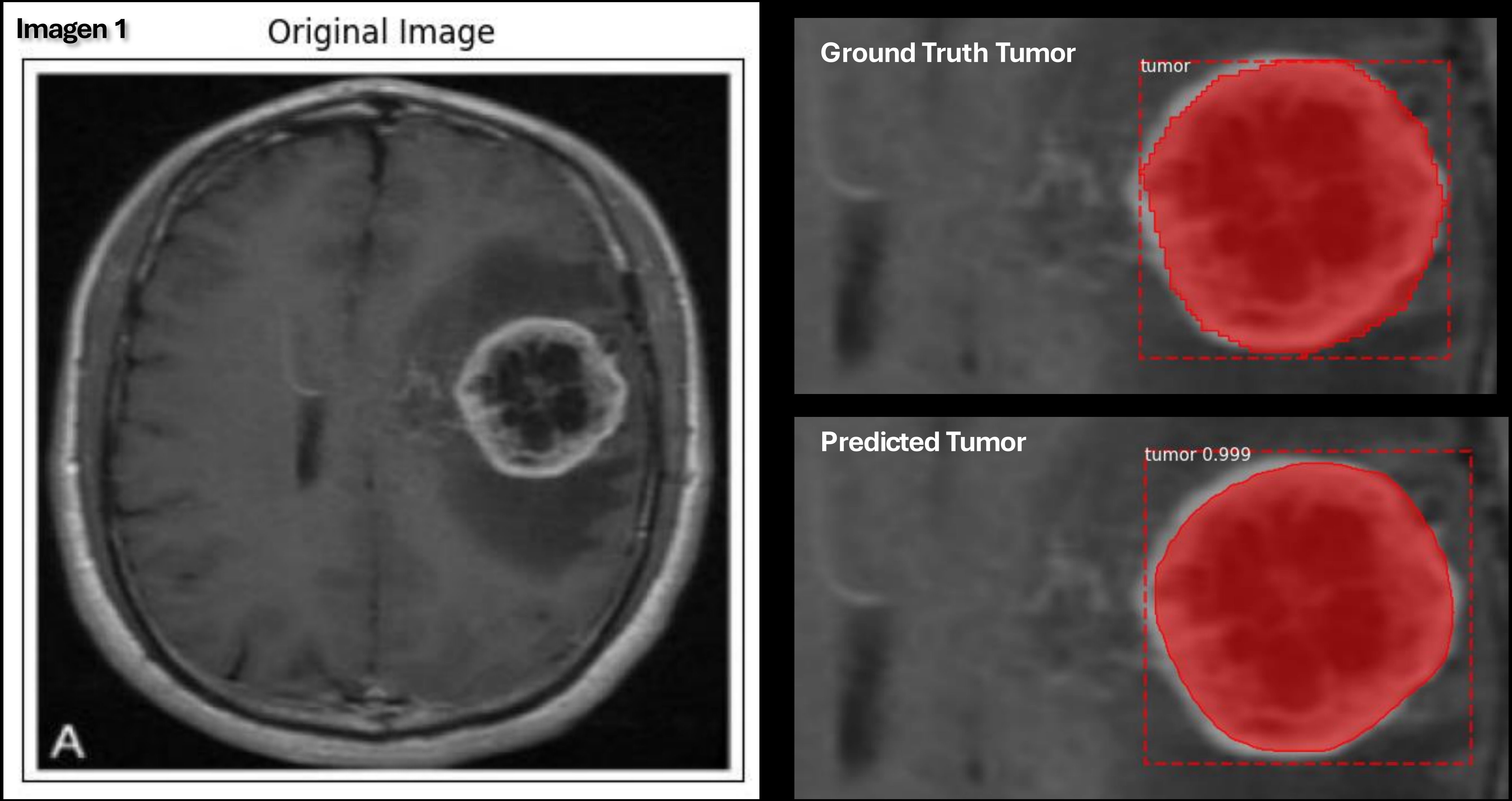
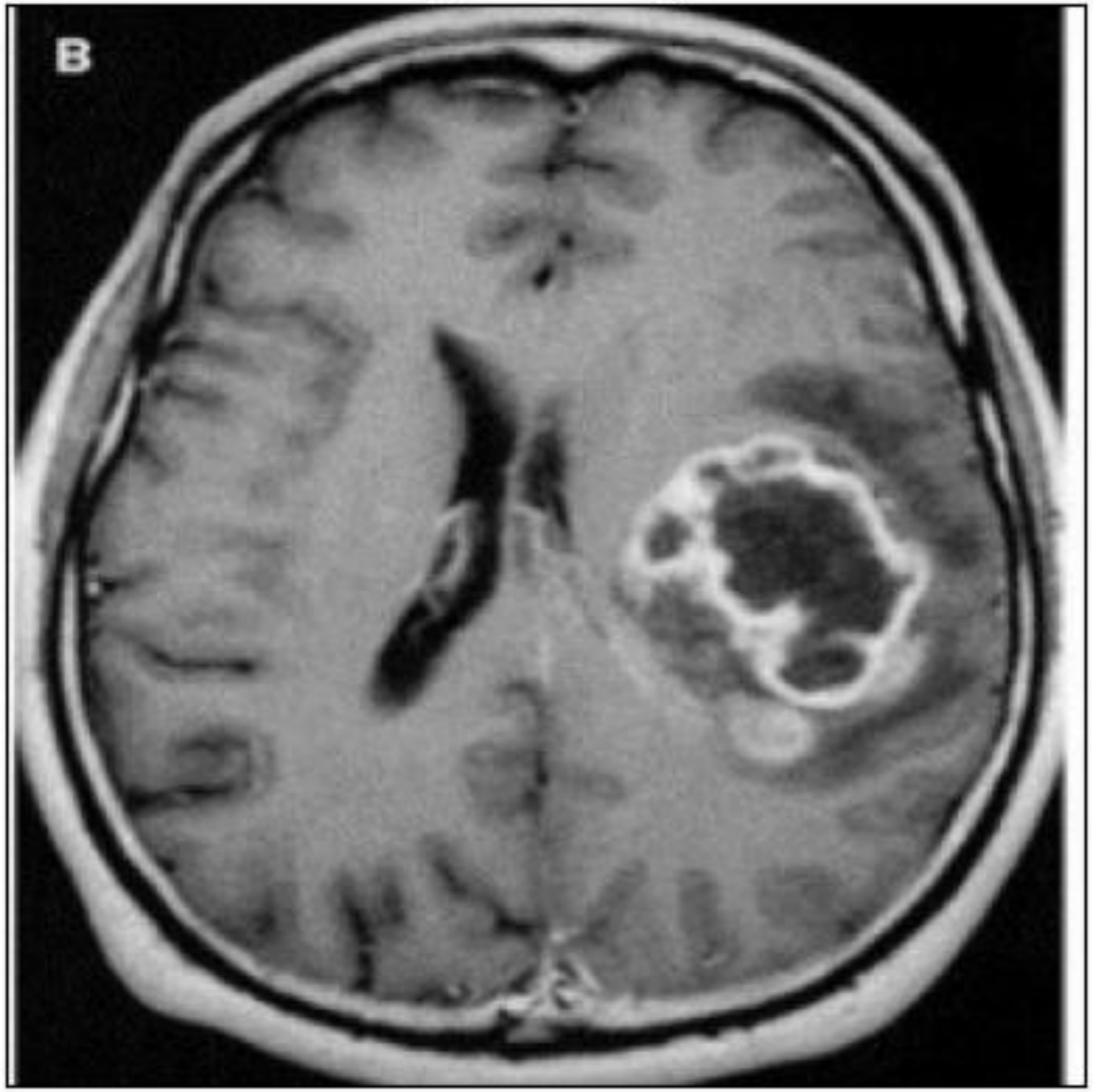
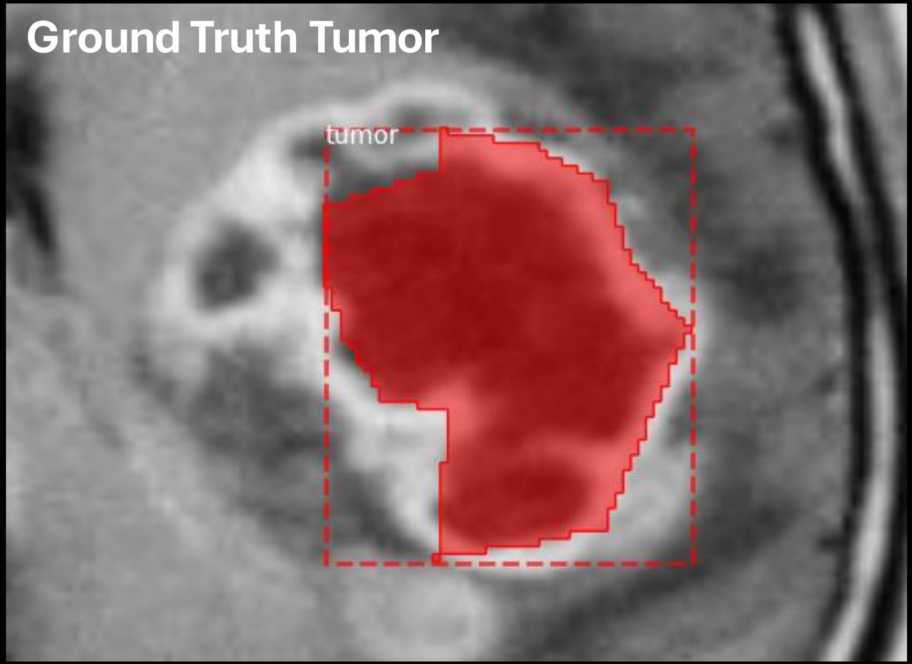


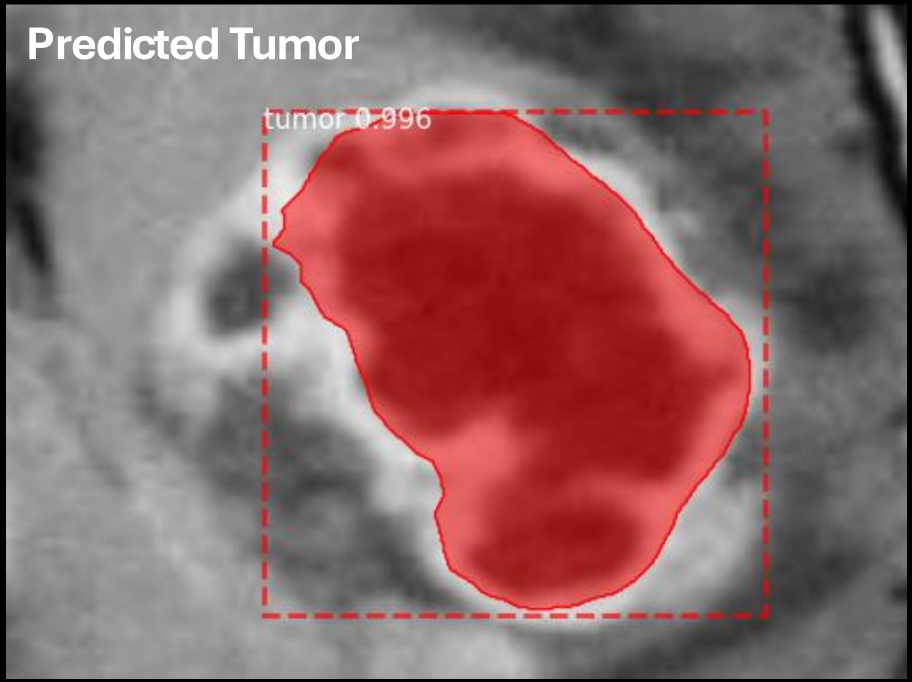
Imagen 1
Original Image
A
Ground Truth Tumor
tumor
Predicted Tumor
tumor 0.999

Imagen 2

Original Image







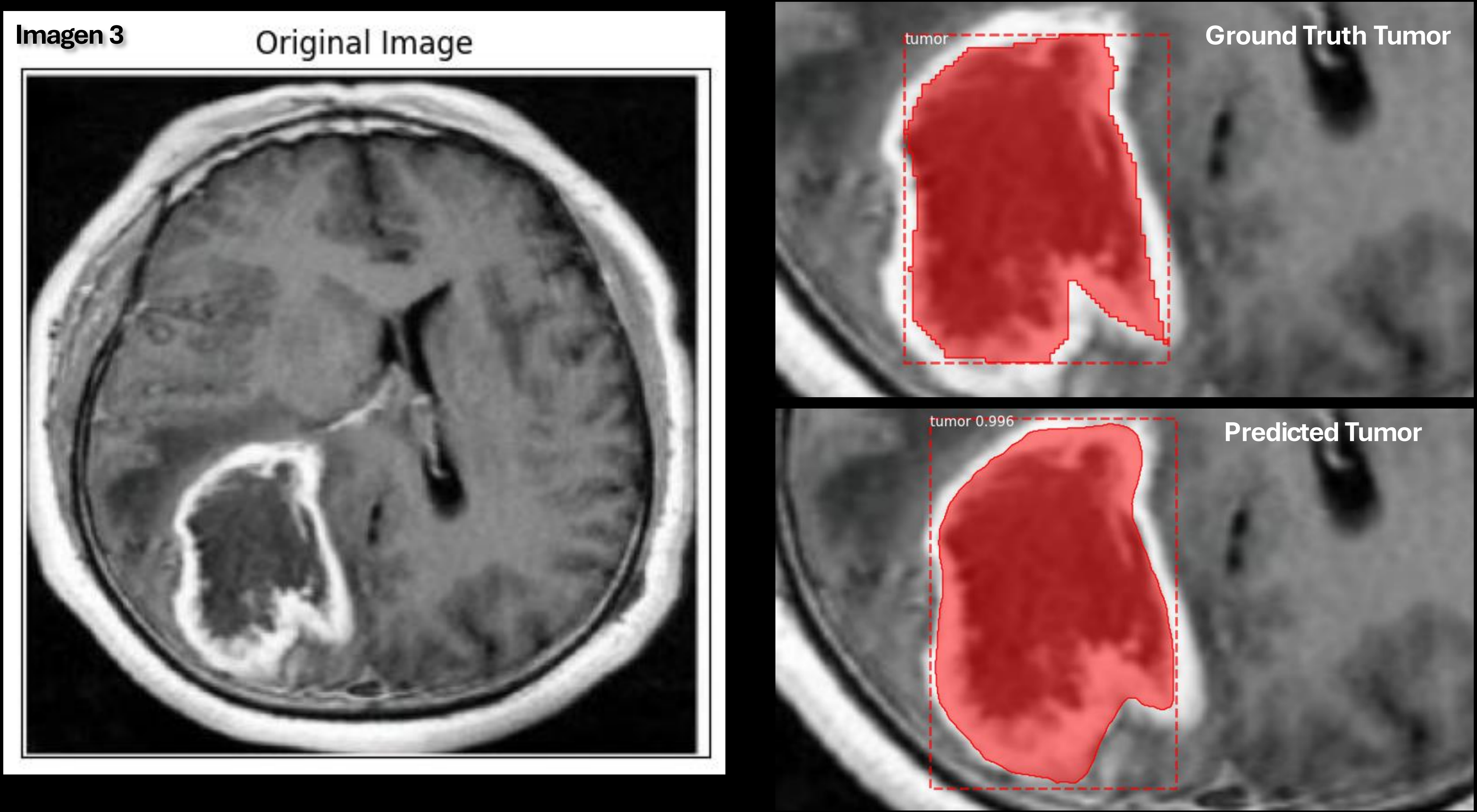


Imagen 3
Original Image
tumor
Ground Truth Tumor
tumor 0.996
Predicted Tumor

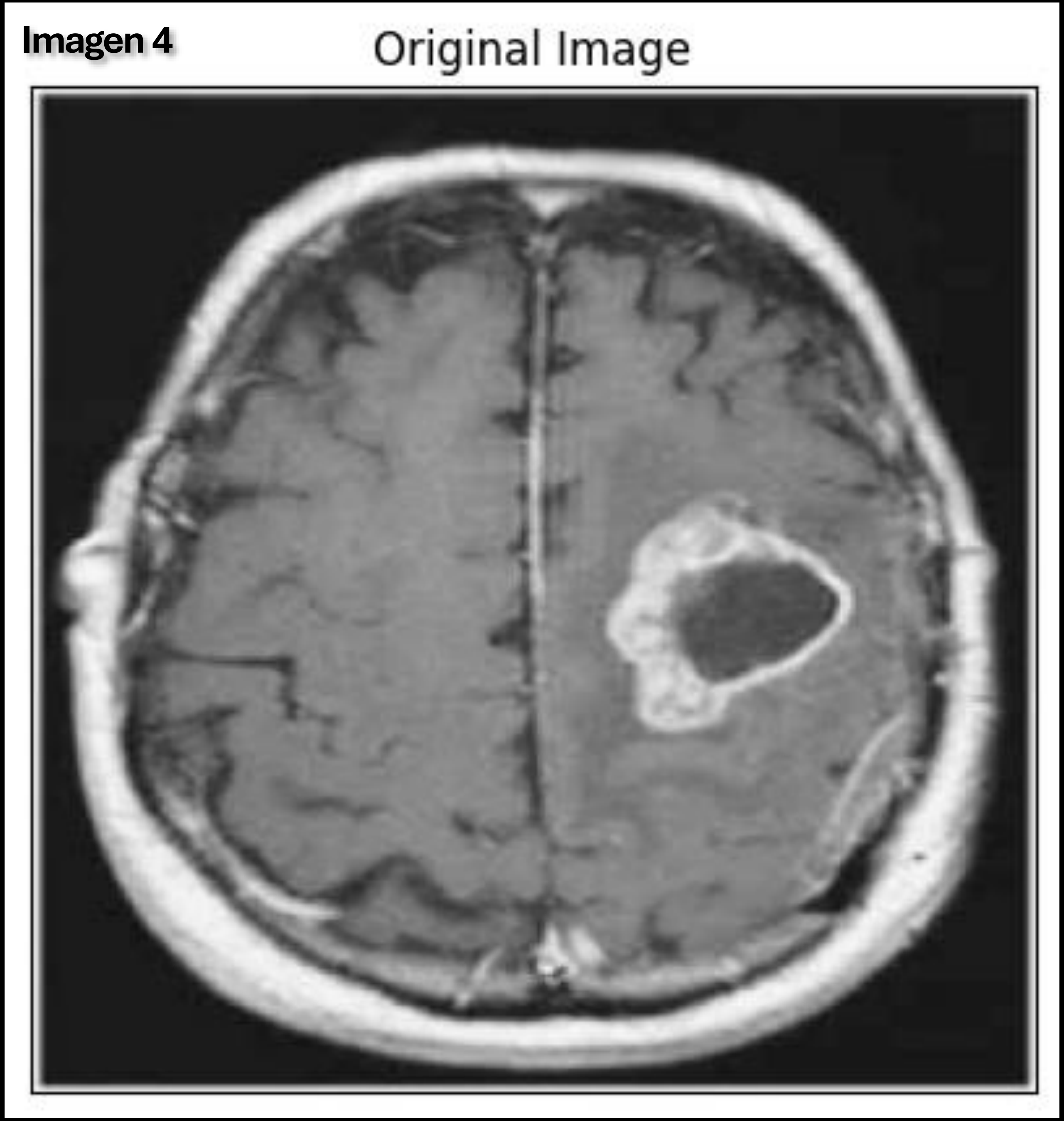
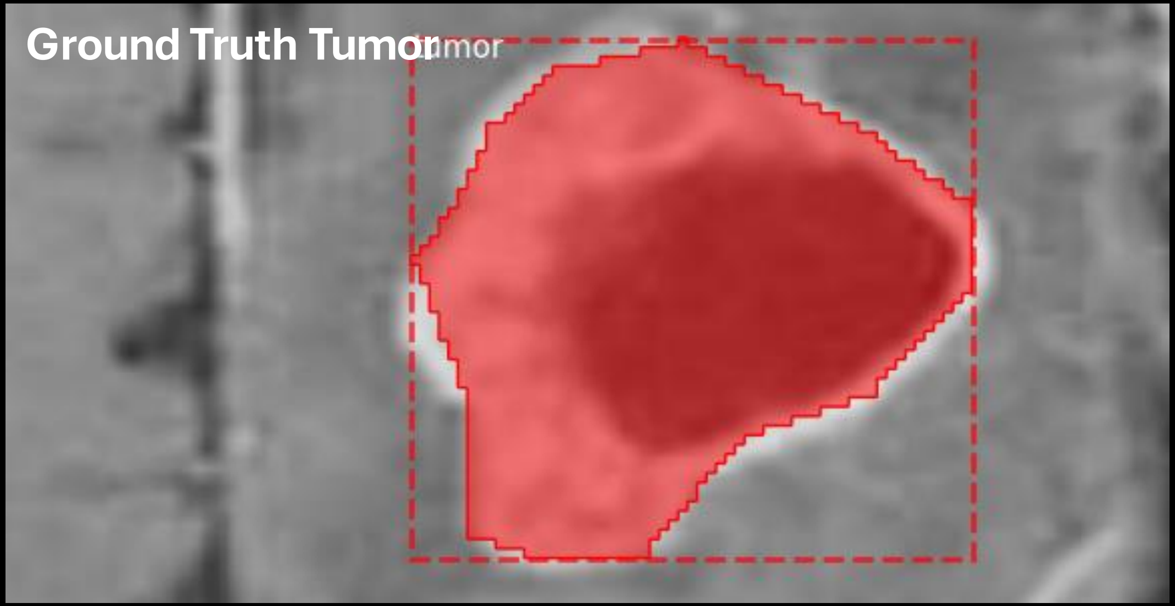
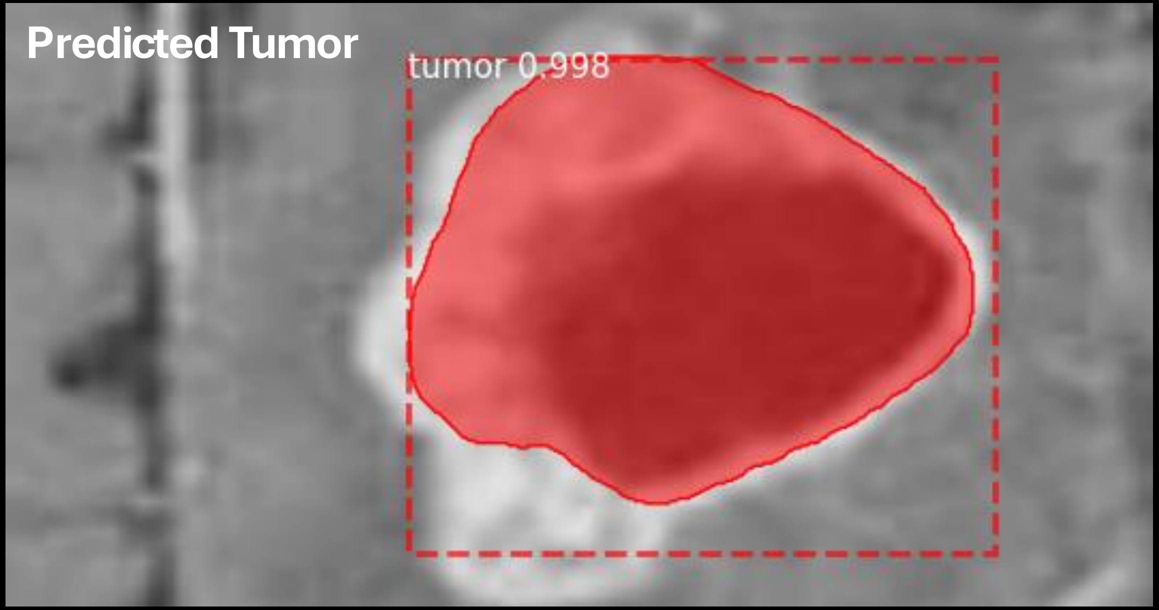


Imagen 4
Original Image


Ground Truth Tumor


Predicted Tumor
tumor 0.998

Métricas de Desempeño

- **Precision:** La precisión mide la proporción de verdaderos positivo entre todos los casos que se predijo como positivo.

$$\text{Presicion} = \frac{TP}{TP + FP}$$

- **Recall:** Mide la proporción de verdaderos positivos entre todos los casos que realmente son positivos.

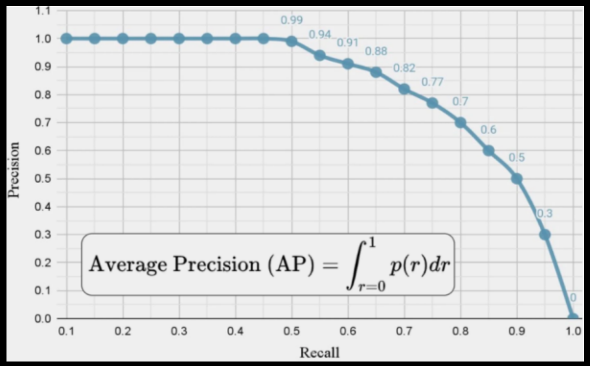
$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Es la media armónica entre precisión y recall.

$$\text{F1}-\text{Score} = 2 \cdot \frac{\text{Presicion} \cdot \text{Recall}}{\text{Presicion} + \text{Recall}}$$

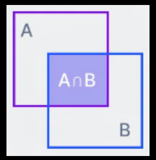
- **Average Precision (AP):** Área bajo la curva de Precisión-Recall para un valor específico de IoU, típicamente 0.5.

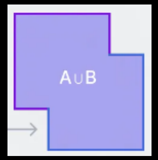
Precision Recall Curve (PR Curve)



- **Intersection over Union (IoU):** mide la superposición entre la región predicha por el modelo y la región real

$$\text{IoU} = \frac{\text{Área de Intersección}}{\text{Área de la unión}} =$$





Métricas de Desempeño

```python
def evaluate_image_metrics(dataset, model, config, image_id, curve_PR=False):
    """
    Calcula las métricas y muestra los resultados para una imagen específica.
    """
    # Cargar la imagen y las anotaciones de ground truth
    image, image_meta, gt_class_id, gt_bbox, gt_mask = \
        modellib.load_image_gt(dataset, config, image_id)

    # Realizar la predicción en la imagen usando el modelo
    results = model.detect([image], verbose=0)

    # Extraer las predicciones
    r = results[0]
    pred_class_id = r['class_ids']
    pred_bbox = r['rois']
    pred_mask = r['masks']
    pred_scores = r['scores']  # Puntuaciones de confianza

    # Calcular las métricas
    AP, precisions, recalls, overlaps = utils.compute_ap(
        gt_bbox, gt_class_id, gt_mask,
        pred_bbox, pred_class_id, pred_scores, pred_mask,
        iou_threshold=0.5)

    # Plot de la curva Precision-Recall
    if curve_PR:
        visualize.plot_precision_recall(AP, precisions, recalls)
        plt.show()  # Forzar la visualización del gráfico
        plt.close() # Cerrar la figura para liberar memoria

    # Quitar el primer y último valor de las listas de precisions y recalls
    precisions = precisions[1:-1]
    recalls = recalls[1:-1]

    # Calcular F1-Score y IoU promedio
    f1_score = calculate_f1_score(precisions, recalls)
    iou_avg = calculate_iou_average(overlaps)
```

```python
    # Imprimir los resultados
    print(f"\nImagen ID: {image_id}")
    print("AP:", AP)
    print("Precisions:", precisions)
    print("Recalls:", recalls)
    print("Overlaps:", overlaps)
    print("Scores:", pred_scores)  # Imprime las puntuaciones de confianza
    print("F1-Score:", f1_score)  # Promedio del F1-Score
    print("IoU Promedio:", iou_avg)

    return AP, precisions, recalls, f1_score, iou_avg

# Iterar sobre todas las imágenes en el conjunto de datos
all_aps = []
all_precisions = []
all_recalls = []
all_f1_scores = []
all_iou_avgs = []

for image_id in range(len(dataset_test.image_ids)):
    AP, precisions, recalls,
    f1_score, iou_avg = evaluate_image_metrics(dataset_test,
                                               model, config, image_id, curve_PR=True)

    # Guardar los resultados en las listas
    all_aps.append(AP)
    all_precisions.append(np.mean(precisions))  # Promedio de precisiones para cada imagen
    all_recalls.append(np.mean(recalls))  # Promedio de recalls para cada imagen
    all_f1_scores.append(f1_score)
    all_iou_avgs.append(iou_avg)

# Calcular y mostrar el promedio de las métricas
print("\nPromedios generales:")
print("mAP @ IoU=50:", np.mean(all_aps))
print("Precisión Promedio:", np.mean(all_precisions))
print("Recall Promedio:", np.mean(all_recalls))
print("F1-Score Promedio:", np.mean(all_f1_scores))
print("IoU Promedio General:", np.mean(all_iou_avgs))
```

Métricas de Desempeño

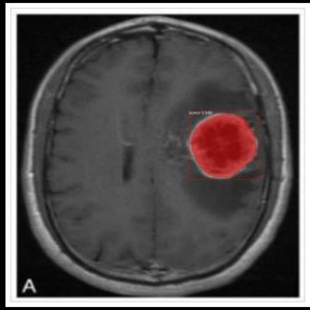


Imagen ID: 0
AP: 1.0
Precisions: [1.]
Recalls: [1.]
Overlaps: [[0.93148404]]
Scores: [0.9991941]
F1-Score: 1.0
IoU Promedio: 0.93148404

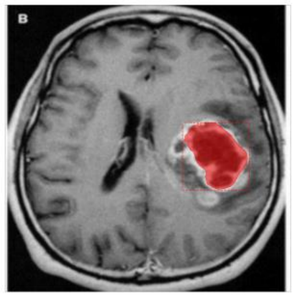


Imagen ID: 1
AP: 1.0
Precisions: [1.]
Recalls: [1.]
Overlaps: [[0.7673453]]
Scores: [0.996202]
F1-Score: 1.0
IoU Promedio: 0.7673453

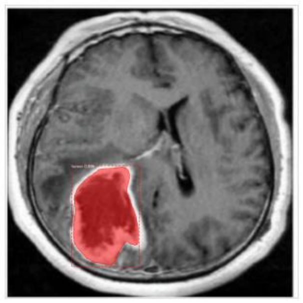


Imagen ID: 2
AP: 1.0
Precisions: [1.]
Recalls: [1.]
Overlaps: [[0.8097122]]
Scores: [0.9958521]
F1-Score: 1.0
IoU Promedio: 0.8097122

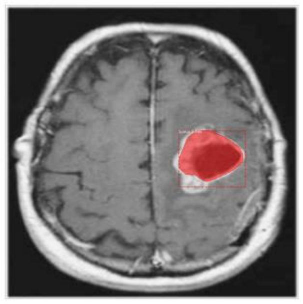
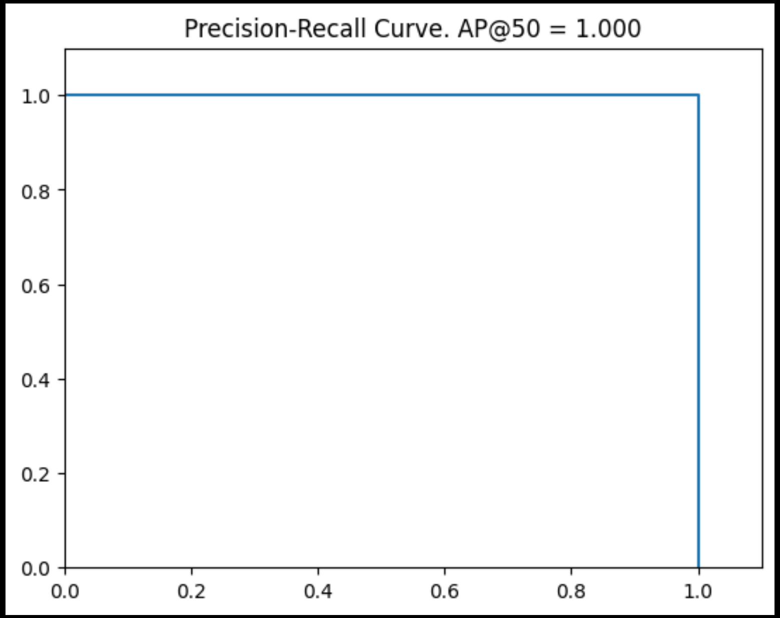


Imagen ID: 3
AP: 1.0
Precisions: [1.]
Recalls: [1.]
Overlaps: [[0.8023705]]
Scores: [0.99790215]
F1-Score: 1.0
IoU Promedio: 0.8023705



Promedios generales:
mAP @ IoU=50: 1.0
Precisión Promedio: 1.0
Recall Promedio: 1.0
F1-Score Promedio: 1.0
IoU Promedio General: 0.82772803

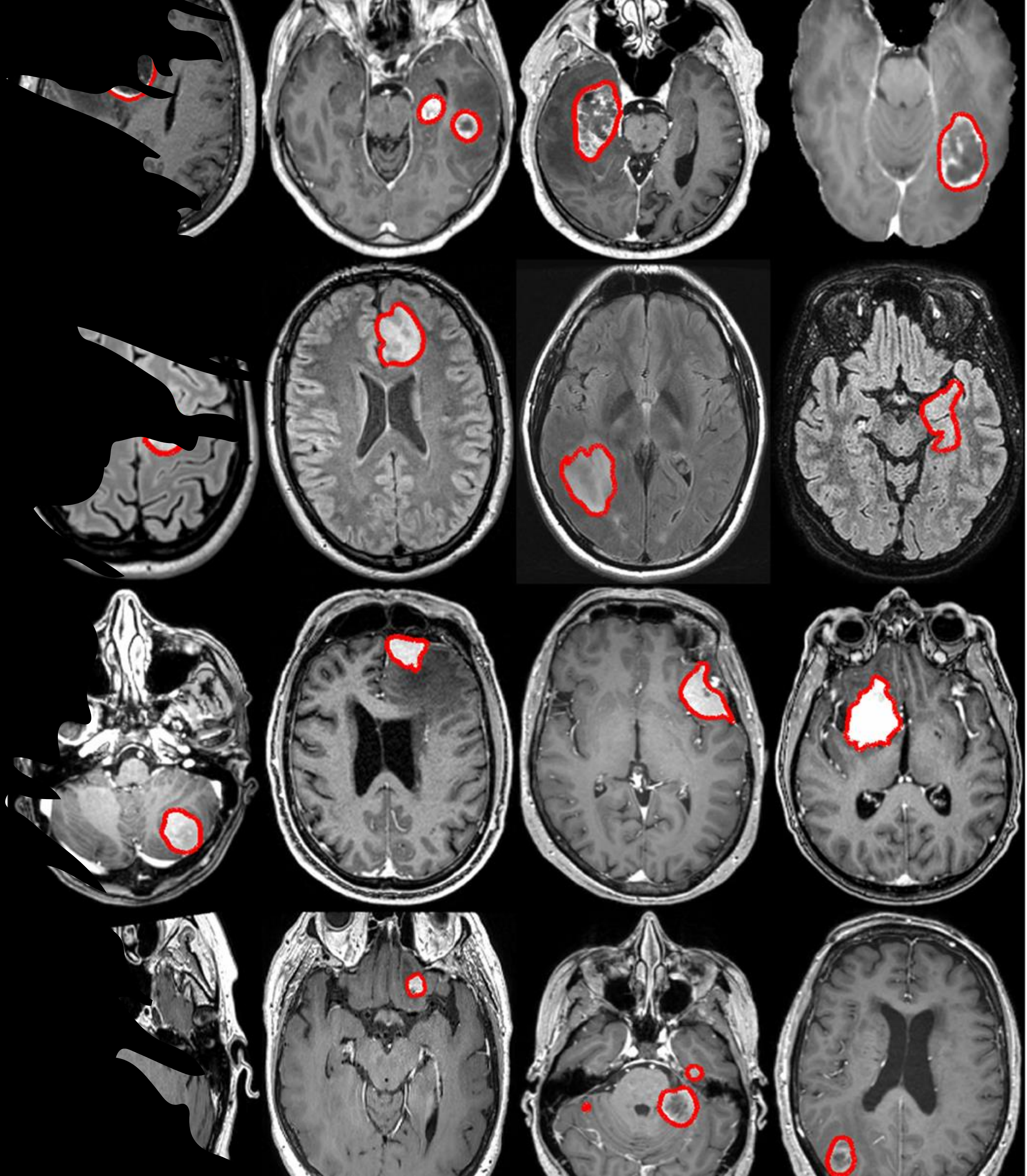
Comparación con otros estudios

* **"Brain Tumor Localization and Segmentation Using Mask RCNN"** by Masood et al. (2021)

Método	Accuracy	mAP	IoU promedio
R-CNN	0,92	0,91	-
Faster R-CNN	0,94	0,94	-
Masood et al. (2021)*	0,95	0,94	0,95
Nuestro método	1.0	1.0	0,83

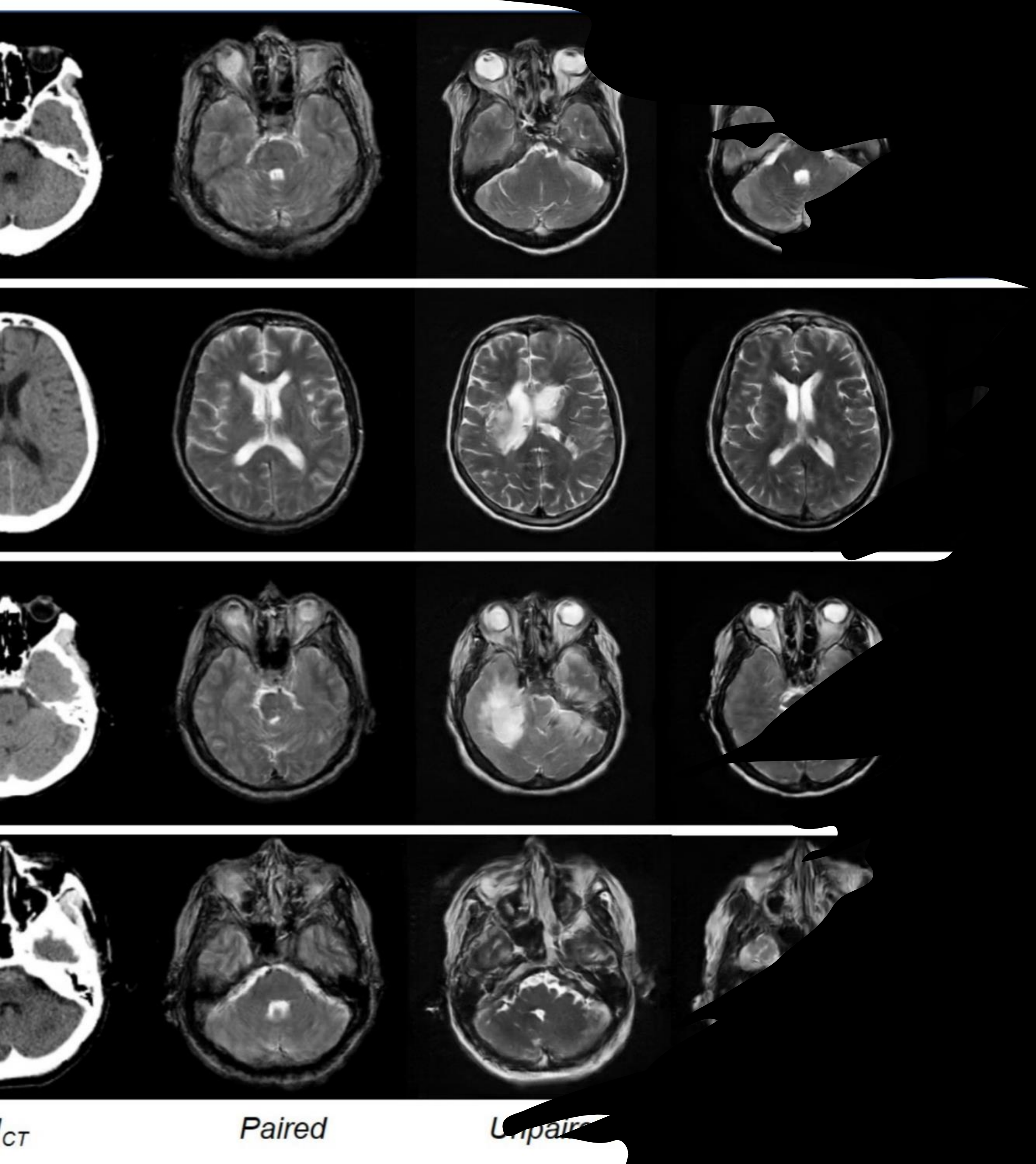
Conclusiones

- **Objetivos Cumplidos:**
  - Implementación exitosa del modelo MASK R-CNN y una CNN convencional.
  - Uso efectivo de GPU para acelerar el proceso.
- **Resultados:**
  - Verificación visual de que el modelo identifica tumores y genera máscaras de segmentación.
- **Desempeño:**
  - Desempeño general bueno, aunque con espacio para mejorar el cálculo de métricas para una mejor comparación con otros métodos.



Trabajo futuro

- **Limpieza de Imágenes:** Eliminar letras y símbolos que interfieren con la detección de tumores.
- **Data Augmentation:** Aumentar la diversidad de imágenes con diferentes ángulos y transformaciones.
- **Mayor Volumen de Datos:** Incrementar el número de imágenes en los conjuntos de entrenamiento, validación y testeo para mejorar la generalización del modelo.



Referencias

- Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.“ URL: https://link.springer.com/article/10.1007/BF00344251
- Girshick, R., Donahue, Darrell, T., & Malik, J. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation.". URL: https://arxiv.org/pdf/1311.2524
- Girshick, R. (2015). "Fast R-CNN.“ URL: https://arxiv.org/pdf/1504.08083
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.“ URL: https://arxiv.org/abs/1506.01497
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.“ URL: https://arxiv.org/pdf/1506.01497
- He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). "Mask R-CNN.“ URL: https://arxiv.org/pdf/1703.06870
- "Brain Tumor Localization and Segmentation Using Mask RCNN" by Masood et al. (2021). URL: https://link.springer.com/article/10.1007/s11704-020-0105-y
- "Brain Tumor Segmentation Using Non-local Mask R-CNN and Single Model Ensemble" by Dai et al. (2021). URL: https://link.springer.com/chapter/10.1007/978-3-031-08999-2_19