

003-2 - Proyecto de Grado- Segundo Modelo Identificacion Estructura - Trans-Ventricular

January 20, 2025

1 Proyecto de Grado Análisis de Estructura Craneal Cavum Septum Peducillum

- Estudiante: Milton Fabian Cifuentes Ortega

2 3.1. Limpieza de Memoria y Carga de Librerías

Primero realizamos la limpieza de memorar y carga de librerías

```
[1]: import gc  
gc.collect()
```

[1]: 167

3 3.2. Carga De Librerías

Procedemos a la carga de las librerías que se necesitan para el funcionamiento del modelo de Unet para el modelo Trans-Ventricular

```
[2]: #librerias que se usan en todo el proyecto  
import cv2  
import os  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
import logging  
import sys  
import shutil  
from sklearn.model_selection import train_test_split  
import glob  
import tensorflow as tf  
from tensorflow.keras.preprocessing import image_dataset_from_directory  
from tensorflow.keras.applications import ResNet50, ResNet101, VGG16, VGG19  
from tensorflow.keras.layers import Dense, Flatten, Dropout, Input, Conv2D,   
↳MaxPooling2D, UpSampling2D, concatenate
```

```

from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import tempfile
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import pickle
from io import BytesIO
from keras_tuner.tuners import RandomSearch
import json
from skimage.metrics import structural_similarity as ssim
from concurrent.futures import ThreadPoolExecutor, as_completed
import cv2
import xml.etree.ElementTree as ET
from sklearn.utils import resample
from sklearn.metrics import precision_score, recall_score, f1_score, \
    accuracy_score, roc_curve, precision_recall_curve, auc
from tensorflow.keras.callbacks import EarlyStopping
import io
from kerastuner import HyperModel
import keras_tuner as kt

```

```

C:\Users\fabia\AppData\Local\Temp\ipykernel_34456\1205247157.py:33:
DeprecationWarning: `import kerastuner` is deprecated, please use `import
keras_tuner`.
    from kerastuner import HyperModel

```

3.1 3.3. Diseño, entrenamiento y evaluación de los Modelos U-Net para los planos Trans-Ventricular

Como pudimos ver en el anterior numeral se crearon los dataset que contienen los datos de entrenamiento y pruebas para cada uno de los planos cerebrales, por lo que ahora crearemos un modelo de u-net para cada plano

3.1.1 3.3.1. Modelo Trans-Ventricular

Procedemos a realizar el modelo Trans-TransVentricular, antes de comenzar normalizaremos las imágenes y validaremos que las máscaras sean binarias

3.3.1.1. Diseño de Modelo Trans-Ventricular Como primera parte del proceso realizaremos la creación de los conjuntos de Entrenamiento y prueba para el plano Trans-TransVentricular, estas imágenes fueron creadas en el anterior cuaderno de jupyter llamado “002 -Proyecto de Grado- Generación de Conjuntos de Datos -Modelo Identificación Estructura”, estos conjuntos ya tienen una distribución de 70% de imágenes y máscaras para entrenamiento y 30% para pruebas.

Lo anterior nos indica que tenemos un total de 2000 imágenes y máscaras de las cuales, 1400 se destinan para entrenamiento del modelo y 600 para pruebas, estos serán cargados en los conjuntos llamados train_images, train_masks y test_images, test_masks

```
[3]: # Directorio para guardar resultados
result_dir = r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de_
↳Grado\003 - Trabajo de grado\Resultados Modelos Unet'
result_file = os.path.join(result_dir, 'ResultadosTransTransVentricular.pkl')

os.makedirs(result_dir, exist_ok=True)

def preprocess_image(img_path, target_size=(224, 224)):
    img = cv2.imread(img_path)
    img = cv2.resize(img, target_size)
    img = img / 255.0 # Normalizar
    return img

def preprocess_mask(mask_path, target_size=(224, 224)):
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
    mask = cv2.resize(mask, target_size)
    mask = np.expand_dims(mask, axis=-1) / 255.0 # Normalizar y agregar canal
    return mask

def load_and_preprocess_data(image_dir, mask_dir, target_size=(224, 224)):
    images = []
    masks = []

    for img_file in sorted(os.listdir(image_dir)):
        if img_file.endswith('.png'):
            img_path = os.path.join(image_dir, img_file)
            mask_path = os.path.join(mask_dir, img_file)

            img = preprocess_image(img_path, target_size)
            mask = preprocess_mask(mask_path, target_size)

            images.append(img)
            masks.append(mask)

    return np.array(images), np.array(masks)

# Cargar y preprocesar datos
train_images, train_masks = load_and_preprocess_data(
    r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de Grado\003 -_
↳Trabajo de grado\Data\ImaSer\Trans-ventricular\Train',
    r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de Grado\003 -_
↳Trabajo de grado\Data\MasBin\Trans-ventricular\Train'
)

test_images, test_masks = load_and_preprocess_data(
    r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de Grado\003 -_
↳Trabajo de grado\Data\ImaSer\Trans-ventricular\Test',
```

```

    r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de Grado\003 -
    Trabajo de grado\Data\MasBin\Trans-ventricular\Test'
)

```

Ahora procedemos a crear el modelo base de la Unet en el cual contaremos con los siguientes aspectos:

1. Como entrada se realizará las imágenes las cuales tienen una dimensión de 224 * 224 píxeles y tres canales (es decir RGB)
2. Contaremos con una transferencia de conocimiento viniendo desde una Red convolucional VGG16
3. Congelaremos los pesos de la VGG16.
4. Contaremos con 4 capas en la fase de encoder todas cuentan con MaxPooling y Dropout para el manejo de sobreajuste.
5. Poseemos una capa de cuello de botella con 1024 filtros.
6. Tenemos 4 capas en la fase de decoder, todas cuentan con Conv2D, upsampling y Dropout para el manejo de sobreajuste

Este modelo es un modelo básico de Unet y consideramos que podría funcionar correctamente para la identificación de la Estructura Cavum Septum Pellucidum

```

[4]: def unet_vgg16_model(input_shape=(224, 224, 3)):
    inputs = Input(shape=input_shape)

    vgg16 = VGG16(include_top=False, weights='imagenet', input_tensor=inputs)
    vgg16.trainable = False # Congelar pesos de VGG16

    # Encoder layers
    conv1 = vgg16.get_layer('block1_conv2').output
    pool1 = MaxPooling2D((2, 2))(conv1)
    drop1 = Dropout(0.5)(pool1)

    conv2 = vgg16.get_layer('block2_conv2').output
    pool2 = MaxPooling2D((2, 2))(conv2)
    drop2 = Dropout(0.5)(pool2)

    conv3 = vgg16.get_layer('block3_conv3').output
    pool3 = MaxPooling2D((2, 2))(conv3)
    drop3 = Dropout(0.5)(pool3)

    conv4 = vgg16.get_layer('block4_conv3').output
    pool4 = MaxPooling2D((2, 2))(conv4)
    drop4 = Dropout(0.5)(pool4)

    # Bottleneck
    bottleneck = Conv2D(1024, (3, 3), activation='relu', padding='same')(drop4)
    bottleneck = Conv2D(1024, (3, 3), activation='relu',
padding='same')(bottleneck)

```

```

# Decoder
up4 = UpSampling2D((2, 2))(bottleneck)
up4 = concatenate([up4, conv4], axis=-1)
drop4 = Dropout(0.5)(up4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(drop4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)

up3 = UpSampling2D((2, 2))(conv4)
up3 = concatenate([up3, conv3], axis=-1)
drop3 = Dropout(0.5)(up3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(drop3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)

up2 = UpSampling2D((2, 2))(conv3)
up2 = concatenate([up2, conv2], axis=-1)
drop2 = Dropout(0.5)(up2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(drop2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)

up1 = UpSampling2D((2, 2))(conv2)
up1 = concatenate([up1, conv1], axis=-1)
drop1 = Dropout(0.5)(up1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(drop1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv1)

model = Model(inputs=[inputs], outputs=[outputs])

return model

# Mostrar el resumen del modelo
model = unet_vgg16_model()
model.summary()

```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792	input_layer[0][0]

block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928	block1_conv1[0][...]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][...]
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584	block2_conv1[0][...]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][...]
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080	block3_conv1[0][...]
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080	block3_conv2[0][...]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][...]
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808	block4_conv1[0][...]
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808	block4_conv2[0][...]
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][...]
dropout_3 (Dropout)	(None, 14, 14, 512)	0	max_pooling2d_3[...]
conv2d (Conv2D)	(None, 14, 14, 1024)	4,719,616	dropout_3[0][0]
conv2d_1 (Conv2D)	(None, 14, 14, 1024)	9,438,208	conv2d[0][0]

up_sampling2d (UpSampling2D)	(None, 28, 28, 1024)	0	conv2d_1[0][0]
concatenate (Concatenate)	(None, 28, 28, 1536)	0	up_sampling2d[0]... block4_conv3[0][...]
dropout_4 (Dropout)	(None, 28, 28, 1536)	0	concatenate[0][0]
conv2d_2 (Conv2D)	(None, 28, 28, 512)	7,078,400	dropout_4[0][0]
conv2d_3 (Conv2D)	(None, 28, 28, 512)	2,359,808	conv2d_2[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 56, 56, 512)	0	conv2d_3[0][0]
concatenate_1 (Concatenate)	(None, 56, 56, 768)	0	up_sampling2d_1[... block3_conv3[0][...]
dropout_5 (Dropout)	(None, 56, 56, 768)	0	concatenate_1[0]...
conv2d_4 (Conv2D)	(None, 56, 56, 256)	1,769,728	dropout_5[0][0]
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590,080	conv2d_4[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 112, 112, 256)	0	conv2d_5[0][0]
concatenate_2 (Concatenate)	(None, 112, 112, 384)	0	up_sampling2d_2[... block2_conv2[0][...]
dropout_6 (Dropout)	(None, 112, 112, 384)	0	concatenate_2[0]...
conv2d_6 (Conv2D)	(None, 112, 112, 128)	442,496	dropout_6[0][0]
conv2d_7 (Conv2D)	(None, 112, 112, 128)	147,584	conv2d_6[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 224, 224, 128)	0	conv2d_7[0][0]

concatenate_3 (Concatenate)	(None, 224, 224, 192)	0	up_sampling2d_3[... block1_conv2[0][...]
dropout_7 (Dropout)	(None, 224, 224, 192)	0	concatenate_3[0]...
conv2d_8 (Conv2D)	(None, 224, 224, 64)	110,656	dropout_7[0][0]
conv2d_9 (Conv2D)	(None, 224, 224, 64)	36,928	conv2d_8[0][0]
conv2d_10 (Conv2D)	(None, 224, 224, 1)	65	conv2d_9[0][0]

Total params: 34,328,833 (130.95 MB)

Trainable params: 26,693,569 (101.83 MB)

Non-trainable params: 7,635,264 (29.13 MB)

3.3.1.2 Entrenamiento de Modelo Trans-TransVentricular Ejecutaremos el proceso de Entrenamiento del primero modelo para validar las métricas de accuracy, f1-score, precisión, curva Roc, entre otros. este proceso se realiza con binary_crossentropy y nos centramos en el accuracy. otro aspecto a tener en cuenta es la aplicación de earlystopping con paciencia de 2 épocas y ejecutaremos el proceso por 5 épocas en total

```
[5]: # Si ya existen los resultados guardados, cargarlos
if os.path.exists(result_file):
    print("Cargando resultados previos...")
    with open(result_file, 'rb') as f:
        results = pickle.load(f)

    history = results['history']
    predictions = results['predictions']
    metrics = results['metrics']
    graphs = results['graphs']

    print("Resultados cargados correctamente.")
else:
    print("No se encontraron resultados previos. Entrenando modelo...")

    # Compilar el modelo
```



```

model.compile(optimizer=Adam(), loss='binary_crossentropy',
↳metrics=['accuracy'])

# Definir EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=2,
↳restore_best_weights=True)

# Entrenar el modelo
history = model.fit(
    train_images,
    train_masks,
    validation_data=(test_images, test_masks),
    epochs=5,
    batch_size=8,
    callbacks=[early_stopping]
)

# Hacer predicciones
predictions = model.predict(test_images)
predictions = (predictions > 0.5).astype(np.uint8)

# Cálculo de métricas
y_true = test_masks.flatten()
y_pred = predictions.flatten()

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
accuracy = accuracy_score(y_true, y_pred)

# Curvas ROC y Precision-Recall
fpr, tpr, _ = roc_curve(y_true, y_pred)
precision_vals, recall_vals, _ = precision_recall_curve(y_true, y_pred)

auc_roc = auc(fpr, tpr)
auc_pr = auc(recall_vals, precision_vals)

metrics = {
    'precision': precision,
    'recall': recall,
    'f1_score': f1,
    'accuracy': accuracy,
    'auc_roc': auc_roc,
    'auc_pr': auc_pr
}

# Guardar gráficas como objetos binarios en un diccionario

```

```

graphs = {}

def save_graph_to_memory(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png')
    buf.seek(0)
    return buf

# Curva de pérdida
fig_loss = plt.figure()
sns.lineplot(x=range(len(history.history['loss'])), y=history.
↳history['loss'], label='Loss')
sns.lineplot(x=range(len(history.history['val_loss'])), y=history.
↳history['val_loss'], label='Val Loss')
plt.title('Curva de Pérdida (Loss)')
graphs['curva_perdida'] = save_graph_to_memory(fig_loss)
plt.close(fig_loss)

# Curva de exactitud
fig_acc = plt.figure()
sns.lineplot(x=range(len(history.history['accuracy'])), y=history.
↳history['accuracy'], label='Accuracy')
sns.lineplot(x=range(len(history.history['val_accuracy'])), y=history.
↳history['val_accuracy'], label='Val Accuracy')
plt.title('Curva de Exactitud (Accuracy)')
graphs['curva_exactitud'] = save_graph_to_memory(fig_acc)
plt.close(fig_acc)

# Curva ROC
fig_roc = plt.figure()
plt.plot(fpr, tpr, label=f'AUC ROC = {auc_roc:.3f}')
plt.title('Curva ROC')
graphs['curva_roc'] = save_graph_to_memory(fig_roc)
plt.close(fig_roc)

# Curva Precision-Recall
fig_pr = plt.figure()
plt.plot(recall_vals, precision_vals, label=f'AUC PR = {auc_pr:.3f}')
plt.title('Curva Precision-Recall')
graphs['curva_precision_recall'] = save_graph_to_memory(fig_pr)
plt.close(fig_pr)

# Guardar todos los resultados en un archivo pickle
results = {
    'history': history.history,
    'predictions': predictions,
    'metrics': metrics,

```

```

        'graphs': graphs
    }

    with open(result_file, 'wb') as f:
        pickle.dump(results, f)

    print("Resultados y gráficas guardados correctamente.")

# Mostrar gráficas y métricas
    print(f"Precision: {metrics['precision']:.3f}")
    print(f"Recall: {metrics['recall']:.3f}")
    print(f"F1-Score: {metrics['f1_score']:.3f}")
    print(f"Accuracy: {metrics['accuracy']:.3f}")
    print(f"AUC ROC: {metrics['auc_roc']:.3f}")
    print(f"AUC PR: {metrics['auc_pr']:.3f}")

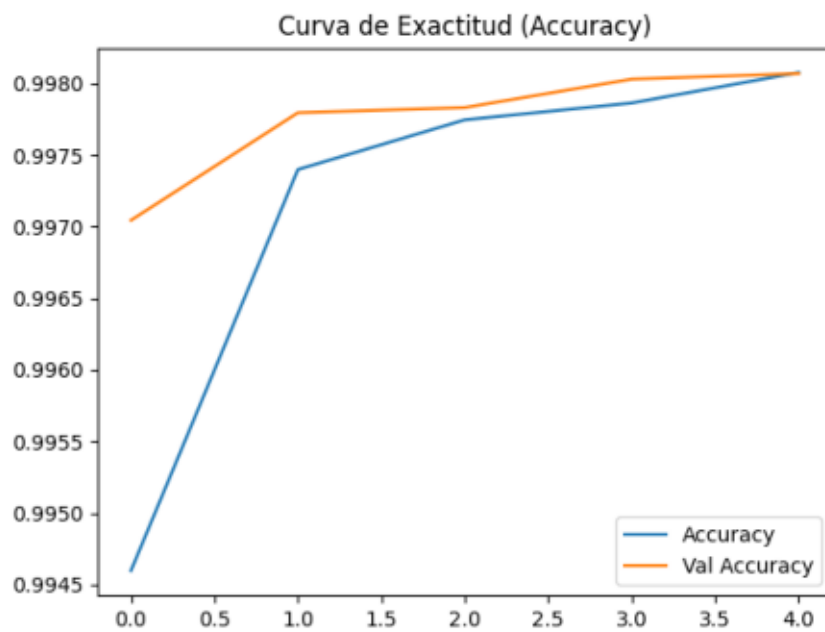
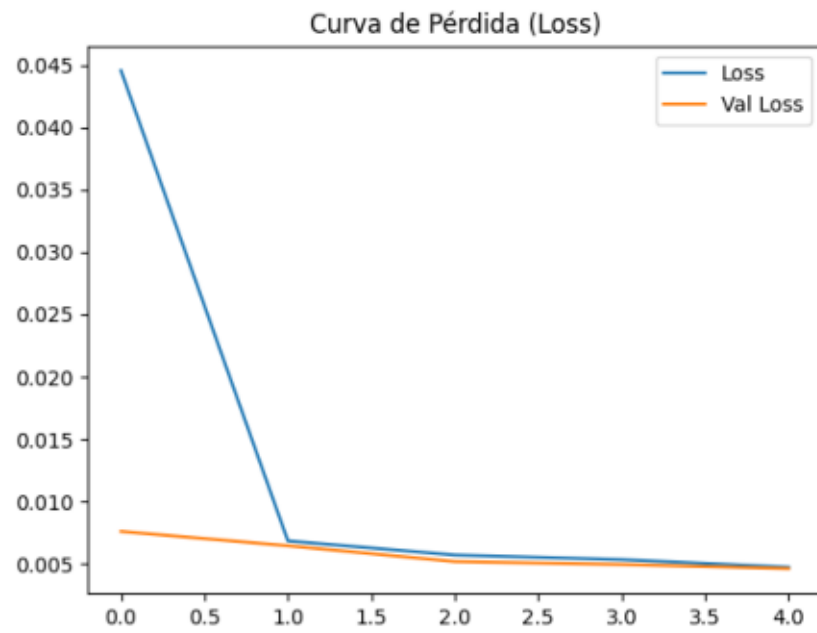
# Función para mostrar gráficas guardadas en la memoria
    def display_graph(buf):
        img = plt.imread(buf)
        plt.imshow(img)
        plt.axis('off')
        plt.show()

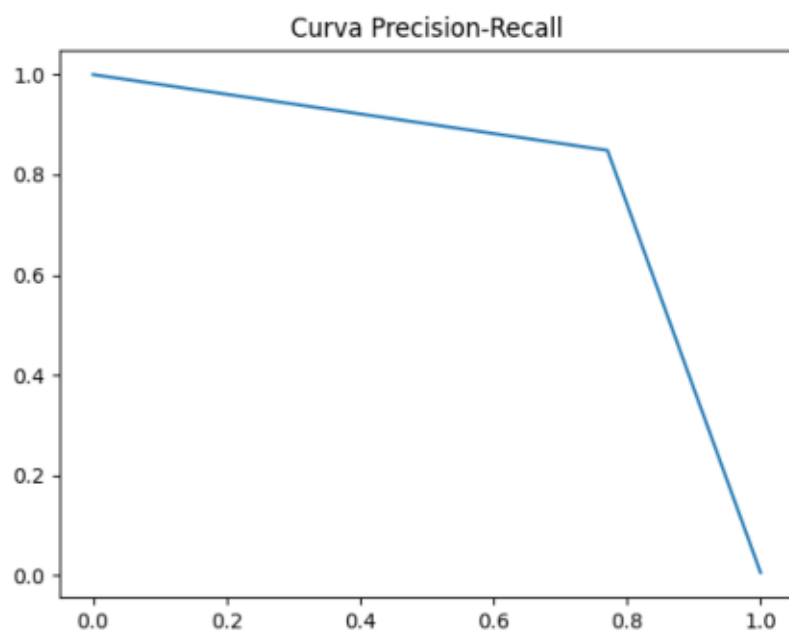
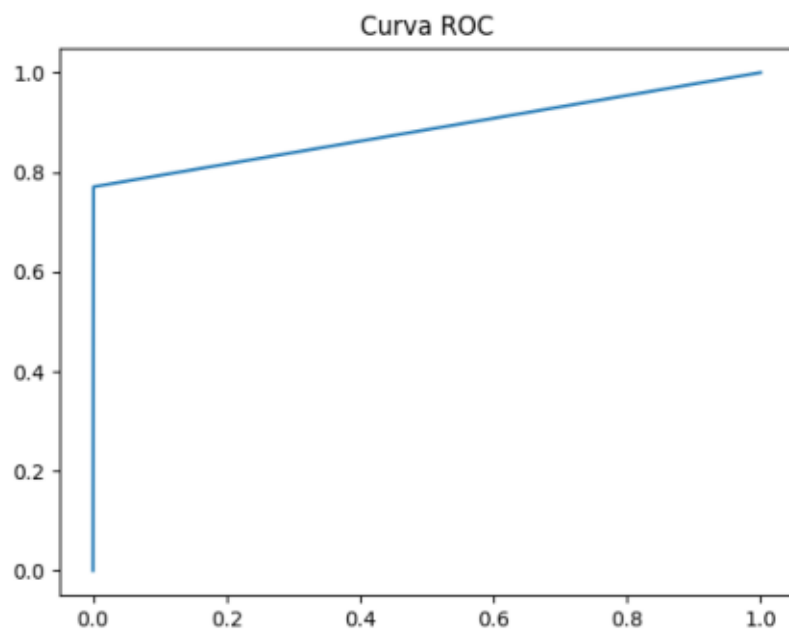
# Mostrar las gráficas guardadas
    display_graph(graphs['curva_perdida'])
    display_graph(graphs['curva_exactitud'])
    display_graph(graphs['curva_roc'])
    display_graph(graphs['curva_precision_recall'])

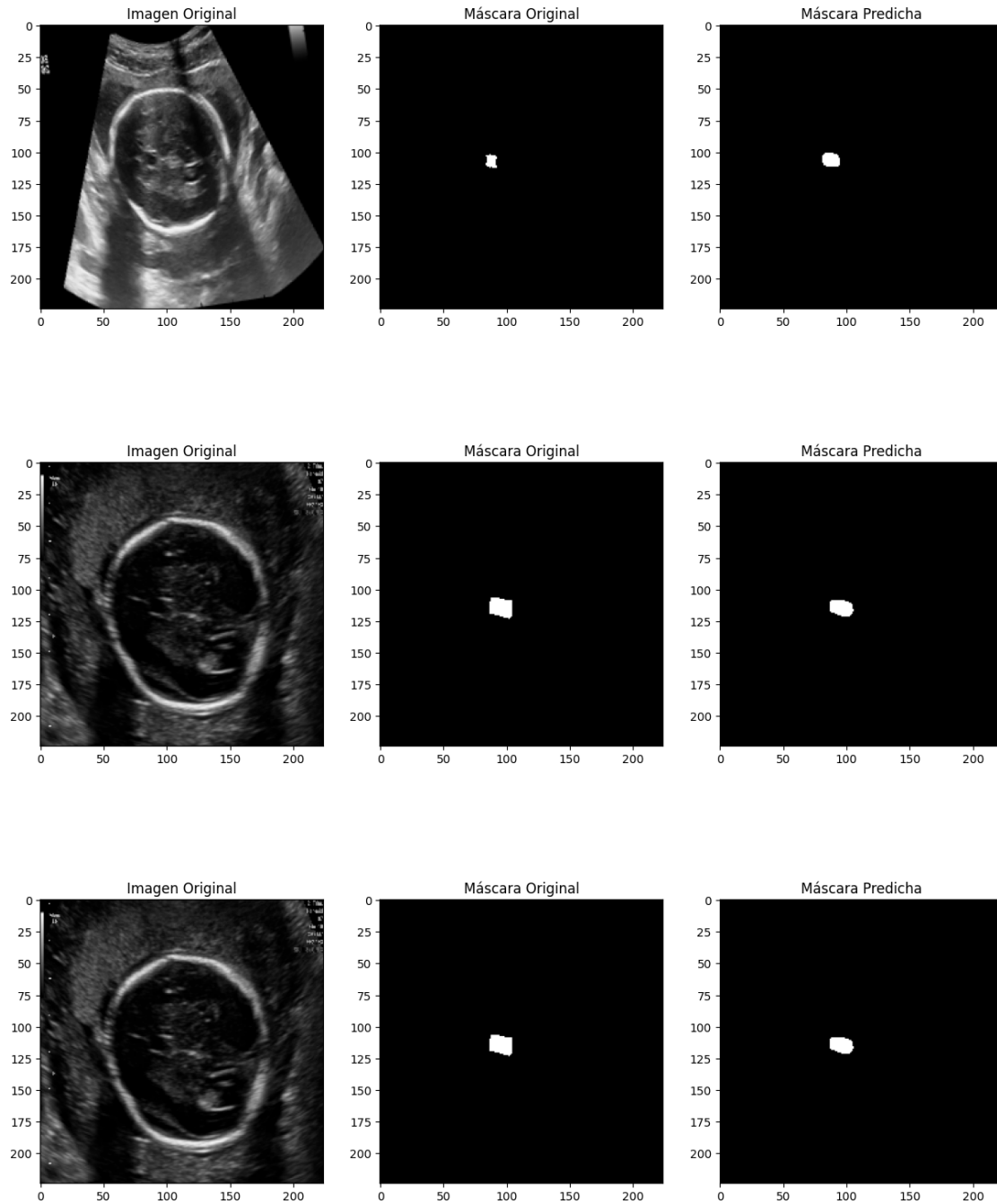
# Mostrar algunas imágenes predichas junto con sus máscaras
    for i in range(3):
        fig, ax = plt.subplots(1, 3, figsize=(15, 5))
        ax[0].imshow(test_images[i])
        ax[0].set_title("Imagen Original")
        ax[1].imshow(test_masks[i].squeeze(), cmap='gray')
        ax[1].set_title("Máscara Original")
        ax[2].imshow(predictions[i].squeeze(), cmap='gray')
        ax[2].set_title("Máscara Predicha")
        plt.show()

```

Cargando resultados previos...
 Resultados cargados correctamente.
 Precision: 0.849
 Recall: 0.771
 F1-Score: 0.808
 Accuracy: 0.998
 AUC ROC: 0.885
 AUC PR: 0.810







Como podemos observar tenemos los siguientes valores para el primer diseño de modelo Unet:

1. Accuracy : 99%
2. Recall : 77%
3. AUC ROC: 88%
4. Precision: 84%
5. F1-Score: 80%

Como podemos darnos cuenta los valores de Precisión y F1-Score, están un poco bajos por lo que centraremos el análisis de mejoramiento en la precisión

3.3.1.3. Evaluación y Afinamiento del Modelo Trans-Ventricular Teniendo en cuenta los datos obtenidos en la anterior etapa correremos una búsqueda de hiperparametros con base en la métrica de precisión, para esto primero diseñaremos la misma estructura del modelo u net y tendremos como hiperparametros a estudiar los siguientes:

1. DropOut
2. Filters
3. LearningRate

Además, que el proceso de búsqueda de hiperparametros se realizara por medio de la función HyperBand

```
[6]: # Definir el modelo U-Net con hiperparámetros
class UNetVGG16HyperModel(HyperModel):
    def __init__(self, input_shape=(224, 224, 3)):
        self.input_shape = input_shape

    def build(self, hp):
        inputs = Input(shape=self.input_shape)

        vgg16 = VGG16(include_top=False, weights='imagenet',
        ↪input_tensor=inputs)
        vgg16.trainable = False # Congelar los pesos de VGG16

        # Encoder layers
        conv1 = vgg16.get_layer('block1_conv2').output
        pool1 = MaxPooling2D((2, 2))(conv1)
        drop1 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.7,
        ↪step=0.1))(pool1)

        conv2 = vgg16.get_layer('block2_conv2').output
        pool2 = MaxPooling2D((2, 2))(conv2)
        drop2 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.7,
        ↪step=0.1))(pool2)

        conv3 = vgg16.get_layer('block3_conv3').output
        pool3 = MaxPooling2D((2, 2))(conv3)
        drop3 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.7,
        ↪step=0.1))(pool3)

        conv4 = vgg16.get_layer('block4_conv3').output
        pool4 = MaxPooling2D((2, 2))(conv4)
        drop4 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.7,
        ↪step=0.1))(pool4)
```

```

# Bottleneck
bottleneck = Conv2D(hp.Int('filters', min_value=512, max_value=1024,
↳step=128), (3, 3), activation='relu', padding='same')(drop4)
bottleneck = Conv2D(hp.Int('filters', min_value=512, max_value=1024,
↳step=128), (3, 3), activation='relu', padding='same')(bottleneck)

# Decoder
up4 = UpSampling2D((2, 2))(bottleneck)
up4 = concatenate([up4, conv4], axis=-1)
drop_dec4 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.
↳7, step=0.1))(up4)
conv4 = Conv2D(512, (3, 3), activation='relu',
↳padding='same')(drop_dec4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)

up3 = UpSampling2D((2, 2))(conv4)
up3 = concatenate([up3, conv3], axis=-1)
drop_dec3 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.
↳7, step=0.1))(up3)
conv3 = Conv2D(256, (3, 3), activation='relu',
↳padding='same')(drop_dec3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)

up2 = UpSampling2D((2, 2))(conv3)
up2 = concatenate([up2, conv2], axis=-1)
drop_dec2 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.
↳7, step=0.1))(up2)
conv2 = Conv2D(128, (3, 3), activation='relu',
↳padding='same')(drop_dec2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)

up1 = UpSampling2D((2, 2))(conv2)
up1 = concatenate([up1, conv1], axis=-1)
drop_dec1 = Dropout(hp.Float('dropout_rate', min_value=0.3, max_value=0.
↳7, step=0.1))(up1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(drop_dec1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv1)

# Crear el modelo
model = Model(inputs=[inputs], outputs=[outputs])

# Compilación del modelo
model.compile(

```



```

        optimizer=Adam(learning_rate=hp.Float('learning_rate',
↪min_value=1e-5, max_value=1e-3, sampling='LOG')),
        loss='binary_crossentropy',
        metrics=['precision']
    )

    return model

```

Procedemos a ejecutar el modelo y guardar los resultados que se obtengan del proceso

```

[8]: # Ruta para guardar o cargar los resultados
results_file_path = r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de
↪Grado\003 - Trabajo de grado\Resultados Modelos
↪Unet\ResultadosMejoradosVentricular.pkl'

def run_random_search():

    # Reiniciar el grafo
    tf.compat.v1.reset_default_graph()

    # Definir el tuner con RandomSearch
    tuner = kt.RandomSearch(
        UNetVGG16HyperModel(input_shape=(224, 224, 3)),
        objective='val_precision',
        max_trials=20, # Número máximo de combinaciones a probar
        executions_per_trial=1, # Número de veces que se ejecuta cada
↪combinación
        directory=r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de
↪Grado\003 - Trabajo de grado\Resultados Modelos Unet\MejoraTrans',
        project_name='unet_vgg16_random_search_Ventricular'
    )

    # Callback para detener el entrenamiento si no hay mejora
    early_stop = EarlyStopping(monitor='val_precision', patience=2,
↪restore_best_weights=True, mode='max')

    # Iniciar la búsqueda de hiperparámetros
    tuner.search(train_images, train_masks, epochs=30, validation_split=0.2,
↪callbacks=[early_stop])

    # Obtener el mejor modelo
    best_model = tuner.get_best_models(num_models=1)[0]

    # Evaluar el mejor modelo
    evaluation = best_model.evaluate(test_images, test_masks)

    # Obtener los mejores hiperparámetros

```

```

best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

# Guardar los resultados en un archivo
tuner_results = {
    'best_model_summary': best_model.summary(print_fn=lambda x: x),
    'best_model_evaluation': evaluation,
    'best_hyperparameters': best_hyperparameters.values
}

with open(results_file_path, 'wb') as file:
    pickle.dump(tuner_results, file)

return tuner_results

# Verificar si el archivo de resultados ya existe
if os.path.exists(results_file_path):
    # Cargar los resultados desde el archivo
    try:
        with open(results_file_path, 'rb') as file:
            tuner_results = pickle.load(file)
        print("Resultados de la búsqueda cargados desde el archivo.")
    except Exception as e:
        print(f"Error al cargar los resultados: {e}")
else:
    # Ejecutar la búsqueda de hiperparámetros y guardar los resultados
    tuner_results = run_random_search()
    print("Resultados de la búsqueda guardados en el archivo.")

# Mostrar los resultados finales
print("\nResultados Finales:")
print("\nResumen del Mejor Modelo:")
print(tuner_results['best_model_summary'])
print("\nEvaluación del Mejor Modelo:")
print(tuner_results['best_model_evaluation'])
print("\nMejores Hiperparámetros Encontrados:")
for param, value in tuner_results['best_hyperparameters'].items():
    print(f"{param}: {value}")

```

Resultados de la búsqueda cargados desde el archivo.

Resultados Finales:

Resumen del Mejor Modelo:

None

Evaluación del Mejor Modelo:

[0.005838938057422638, 0.8324362635612488]

Mejores Hiperparámetros Encontrados:

dropout_rate: 0.6000000000000001

filters: 896

learning_rate: 0.0008013308557761683

Podemos observar que los mejores valores de hiperparametros son:

1. DropOut: 0.6
2. Filters: 896
3. Learning_Rate: 0.0008013308557761683
4. Cantidad de Épocas: 5

con estos parámetros se probará el Modelo de U-Net, aunque nos parece que 2 épocas son muy pocas por lo que lo dejaremos a 5 y que el early stoping lo detenga si es necesario

3.3.1.4. Implementación del Modelo Trans-Ventricular de U-Net Final Teniendo en cuenta los resultados anteriormente obtenidos para mejorar el modelo se diseña el siguiente modelo el cual será el definitivo para el plano cerebral Trans-Ventricular

Como podemos darnos cuenta los valores para el mejoramiento del modelo son 1. Dropout: 0.6 2. filters:896 3. epocas:5 4. learning rate: 0.0008013308557761683

Guardaremos los resultados del modelo mejorado en el archivo de nombre ResultadosTransVentricular_Mejorado.pkl

```
[9]: # Directorios para guardar resultados y carpetas de imágenes
result_dir_mejorado = r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo_
↳de Grado\003 - Trabajo de grado\Resultados Modelos Unet'
result_file_mejorado = os.path.join(result_dir_mejorado,
↳'ResultadosTransVentricular_MejoradoV3.pkl')
```

Definimos el modelo con los valores de hiperparamtros seleccionados en la anterior Etapa

```
[10]: def unet_vgg16_model_Mejorado(input_shape=(224, 224, 3)):
    inputs = Input(shape=input_shape)

    vgg16 = VGG16(include_top=False, weights='imagenet', input_tensor=inputs)
    vgg16.trainable = False # Congelar pesos de VGG16

    # Encoder layers
    conv1 = vgg16.get_layer('block1_conv2').output
    pool1 = MaxPooling2D((2, 2))(conv1)
    drop1 = Dropout(0.6)(pool1)

    conv2 = vgg16.get_layer('block2_conv2').output
    pool2 = MaxPooling2D((2, 2))(conv2)
    drop2 = Dropout(0.6)(pool2)

    conv3 = vgg16.get_layer('block3_conv3').output
    pool3 = MaxPooling2D((2, 2))(conv3)
```

```

drop3 = Dropout(0.6)(pool3)

conv4 = vgg16.get_layer('block4_conv3').output
pool4 = MaxPooling2D((2, 2))(conv4)
drop4 = Dropout(0.6)(pool4)

# Bottleneck
bottleneck = Conv2D(896, (3, 3), activation='relu', padding='same')(drop4)
bottleneck = Conv2D(896, (3, 3), activation='relu',
padding='same')(bottleneck)

# Decoder
up4 = UpSampling2D((2, 2))(bottleneck)
up4 = concatenate([up4, conv4], axis=-1)
drop4 = Dropout(0.6)(up4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(drop4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)

up3 = UpSampling2D((2, 2))(conv4)
up3 = concatenate([up3, conv3], axis=-1)
drop3 = Dropout(0.6)(up3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(drop3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)

up2 = UpSampling2D((2, 2))(conv3)
up2 = concatenate([up2, conv2], axis=-1)
drop2 = Dropout(0.6)(up2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(drop2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)

up1 = UpSampling2D((2, 2))(conv2)
up1 = concatenate([up1, conv1], axis=-1)
drop1 = Dropout(0.6)(up1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(drop1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)

outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv1)

model = Model(inputs=[inputs], outputs=[outputs])

return model

# Mostrar el resumen del modelo
model_mejorado = unet_vgg16_model_Mejorado()
model_mejorado.summary()

```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	-
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792	input_layer_1[0]...
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928	block1_conv1[0] [...
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0] [...
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584	block2_conv1[0] [...
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0] [...
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080	block3_conv1[0] [...
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080	block3_conv2[0] [...
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0] [...
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808	block4_conv1[0] [...
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808	block4_conv2[0] [...
max_pooling2d_7 (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0] [...

dropout_11 (Dropout)	(None, 14, 14, 512)	0	max_pooling2d_7[...
conv2d_11 (Conv2D)	(None, 14, 14, 896)	4,129,664	dropout_11[0][0]
conv2d_12 (Conv2D)	(None, 14, 14, 896)	7,226,240	conv2d_11[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 28, 28, 896)	0	conv2d_12[0][0]
concatenate_4 (Concatenate)	(None, 28, 28, 1408)	0	up_sampling2d_4[... block4_conv3[0][...
dropout_12 (Dropout)	(None, 28, 28, 1408)	0	concatenate_4[0]...
conv2d_13 (Conv2D)	(None, 28, 28, 512)	6,488,576	dropout_12[0][0]
conv2d_14 (Conv2D)	(None, 28, 28, 512)	2,359,808	conv2d_13[0][0]
up_sampling2d_5 (UpSampling2D)	(None, 56, 56, 512)	0	conv2d_14[0][0]
concatenate_5 (Concatenate)	(None, 56, 56, 768)	0	up_sampling2d_5[... block3_conv3[0][...
dropout_13 (Dropout)	(None, 56, 56, 768)	0	concatenate_5[0]...
conv2d_15 (Conv2D)	(None, 56, 56, 256)	1,769,728	dropout_13[0][0]
conv2d_16 (Conv2D)	(None, 56, 56, 256)	590,080	conv2d_15[0][0]
up_sampling2d_6 (UpSampling2D)	(None, 112, 112, 256)	0	conv2d_16[0][0]
concatenate_6 (Concatenate)	(None, 112, 112, 384)	0	up_sampling2d_6[... block2_conv2[0][...
dropout_14 (Dropout)	(None, 112, 112, 384)	0	concatenate_6[0]...

conv2d_17 (Conv2D)	(None, 112, 112, 128)	442,496	dropout_14[0][0]
conv2d_18 (Conv2D)	(None, 112, 112, 128)	147,584	conv2d_17[0][0]
up_sampling2d_7 (UpSampling2D)	(None, 224, 224, 128)	0	conv2d_18[0][0]
concatenate_7 (Concatenate)	(None, 224, 224, 192)	0	up_sampling2d_7[...] block1_conv2[0][...]
dropout_15 (Dropout)	(None, 224, 224, 192)	0	concatenate_7[0][...]
conv2d_19 (Conv2D)	(None, 224, 224, 64)	110,656	dropout_15[0][0]
conv2d_20 (Conv2D)	(None, 224, 224, 64)	36,928	conv2d_19[0][0]
conv2d_21 (Conv2D)	(None, 224, 224, 1)	65	conv2d_20[0][0]

Total params: 30,937,089 (118.02 MB)

Trainable params: 23,301,825 (88.89 MB)

Non-trainable params: 7,635,264 (29.13 MB)

Realizamos el proceso de entrenamiento del modelo y calculamos las medidas que habíamos tenido anteriormente en cuenta

```
[14]: # Definir la ruta de los resultados y las imágenes
image_dir = r"D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de_
↳Grado\003 - Trabajo de grado\Resultados Modelos_
↳Unet\ImagenesPredichas\Trans-Ventricular\imagenes"
mask_dir = r"D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de_
↳Grado\003 - Trabajo de grado\Resultados Modelos_
↳Unet\ImagenesPredichas\Trans-Ventricular\mascaras"
pred_mask_dir = r"D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de_
↳Grado\003 - Trabajo de grado\Resultados Modelos_
↳Unet\ImagenesPredichas\Trans-Ventricular\mascaras_predichas"
```

```

# Crear directorios si no existen
os.makedirs(image_dir, exist_ok=True)
os.makedirs(mask_dir, exist_ok=True)
os.makedirs(pred_mask_dir, exist_ok=True)

# Si ya existen los resultados guardados, cargarlos
if os.path.exists(result_file_mejorado):
    print("Cargando resultados previos...")
    with open(result_file_mejorado, 'rb') as f:
        results = pickle.load(f)

    history = results['history']
    predictions = results['predictions']
    metrics = results['metrics']
    graphs = results['graphs']

    print("Resultados cargados correctamente.")
else:
    print("No se encontraron resultados previos. Entrenando modelo...")

    # Compilar el modelo
    model_mejorado.compile(optimizer=Adam(learning_rate=0.0008013308557761683),
        ↪ loss='binary_crossentropy', metrics=['accuracy'])

    # Definir EarlyStopping callback
    early_stopping = EarlyStopping(monitor='val_accuracy', patience=2,
        ↪ restore_best_weights=True)

    # Entrenar el modelo
    history = model_mejorado.fit(
        train_images,
        train_masks,
        validation_data=(test_images, test_masks),
        epochs=5,
        batch_size=8,
        callbacks=[early_stopping]
    )

    # Hacer predicciones
    predictions = model_mejorado.predict(test_images)
    predictions = (predictions > 0.5).astype(np.uint8)

    # Cálculo de métricas
    y_true = test_masks.flatten()
    y_pred = predictions.flatten()

```



```

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
accuracy = accuracy_score(y_true, y_pred)

# Curvas ROC y Precision-Recall
fpr, tpr, _ = roc_curve(y_true, y_pred)
precision_vals, recall_vals, _ = precision_recall_curve(y_true, y_pred)

auc_roc = auc(fpr, tpr)
auc_pr = auc(recall_vals, precision_vals)

metrics = {
    'precision': precision,
    'recall': recall,
    'f1_score': f1,
    'accuracy': accuracy,
    'auc_roc': auc_roc,
    'auc_pr': auc_pr
}

# Guardar gráficas como objetos binarios en un diccionario
graphs = {}

def save_graph_to_memory(fig):
    buf = io.BytesIO()
    fig.savefig(buf, format='png')
    buf.seek(0)
    return buf

# Curva de pérdida
fig_loss = plt.figure()
sns.lineplot(x=range(len(history.history['loss'])), y=history.
↪history['loss'], label='Loss')
sns.lineplot(x=range(len(history.history['val_loss'])), y=history.
↪history['val_loss'], label='Val Loss')
plt.title('Curva de Pérdida (Loss)')
graphs['curva_perdida'] = save_graph_to_memory(fig_loss)
plt.close(fig_loss)

# Curva de exactitud
fig_acc = plt.figure()
sns.lineplot(x=range(len(history.history['accuracy'])), y=history.
↪history['accuracy'], label='Accuracy')
sns.lineplot(x=range(len(history.history['val_accuracy'])), y=history.
↪history['val_accuracy'], label='Val Accuracy')
plt.title('Curva de Exactitud (Accuracy)')

```

```

graphs['curva_exactitud'] = save_graph_to_memory(fig_acc)
plt.close(fig_acc)

# Curva ROC
fig_roc = plt.figure()
plt.plot(fpr, tpr, label=f'AUC ROC = {auc_roc:.3f}')
plt.title('Curva ROC')
graphs['curva_roc'] = save_graph_to_memory(fig_roc)
plt.close(fig_roc)

# Curva Precision-Recall
fig_pr = plt.figure()
plt.plot(recall_vals, precision_vals, label=f'AUC PR = {auc_pr:.3f}')
plt.title('Curva Precision-Recall')
graphs['curva_precision_recall'] = save_graph_to_memory(fig_pr)
plt.close(fig_pr)

# Guardar todos los resultados en un archivo pickle
results = {
    'history': history.history,
    'predictions': predictions,
    'metrics': metrics,
    'graphs': graphs
}

with open(result_file_mejorado, 'wb') as f:
    pickle.dump(results, f)

print("Resultados y gráficas guardados correctamente.")

# Función para borrar todas las imágenes en una carpeta
def borrar_imagenes(carpeta):
    # Encontrar todas las imágenes con extensiones comunes
    archivos_imagenes = glob.glob(os.path.join(carpeta, "*.png"))
    for archivo in archivos_imagenes:
        try:
            os.remove(archivo)
            print(f"Archivo {archivo} borrado.")
        except OSError as e:
            print(f"Error al borrar {archivo}: {e}")

# Borrar imágenes en cada carpeta
borrar_imagenes(image_dir)
borrar_imagenes(mask_dir)
borrar_imagenes(pred_mask_dir)

```

```
DireccionTest= r'D:\Archivos Milton\Maestria en Ciencia de Datos\Trabajo de  
↪Grado\003 - Trabajo de grado\Data\ImaSer\Trans-ventricular\Test'
```

```
# Guardar imágenes originales, máscaras y predicciones  
for i in range(min(10, len(test_images))):  
    # Obtener el nombre original del archivo (sin la extensión)  
    original_file_name = os.path.splitext(sorted(os.  
↪listdir(DireccionTest))[i])[0]  
  
    # Definir nombres de los archivos a guardar  
    image_name = f"{original_file_name}_imagen.png"  
    mask_name = f"{original_file_name}_mascara.png"  
    pred_mask_name = f"{original_file_name}_mascara_predicha.png"  
  
    # Convertir la imagen a uint8 y guardarla  
    img_uint8 = (test_images[i] * 255).astype(np.uint8)  
    cv2.imwrite(os.path.join(image_dir, image_name), cv2.  
↪cvtColor(img_uint8, cv2.COLOR_RGB2BGR))  
  
    # Convertir la máscara original a uint8 y guardarla  
    mask_uint8 = (test_masks[i].squeeze() * 255).astype(np.uint8)  
    cv2.imwrite(os.path.join(mask_dir, mask_name), mask_uint8)  
  
    # Convertir la máscara predicha a uint8 y guardarla  
    pred_mask_uint8 = (predictions[i].squeeze() * 255).astype(np.uint8)  
    cv2.imwrite(os.path.join(pred_mask_dir, pred_mask_name),  
↪pred_mask_uint8)  
  
    print("Imágenes, máscaras y predicciones guardadas correctamente.")  
  
# Mostrar gráficas y métricas  
print(f"Precision: {metrics['precision']:.3f}")  
print(f"Recall: {metrics['recall']:.3f}")  
print(f"F1-Score: {metrics['f1_score']:.3f}")  
print(f"Accuracy: {metrics['accuracy']:.3f}")  
print(f"AUC ROC: {metrics['auc_roc']:.3f}")  
print(f"AUC PR: {metrics['auc_pr']:.3f}")  
  
# Función para mostrar gráficas guardadas en la memoria  
def display_graph(buf):  
    img = plt.imread(buf)  
    plt.imshow(img)  
    plt.axis('off')  
    plt.show()  
  
# Mostrar las gráficas guardadas
```

```

display_graph(graphs['curva_perdida'])
display_graph(graphs['curva_exactitud'])
display_graph(graphs['curva_roc'])
display_graph(graphs['curva_precision_recall'])

# Mostrar algunas imágenes predichas junto con sus máscaras
for i in range(3):
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    ax[0].imshow(test_images[i])
    ax[0].set_title("Imagen Original")
    ax[1].imshow(test_masks[i].squeeze(), cmap='gray')
    ax[1].set_title("Máscara Original")
    ax[2].imshow(predictions[i].squeeze(), cmap='gray')
    ax[2].set_title("Máscara Predicha")
    plt.show()

```

Cargando resultados previos...

Resultados cargados correctamente.

Precision: 0.849

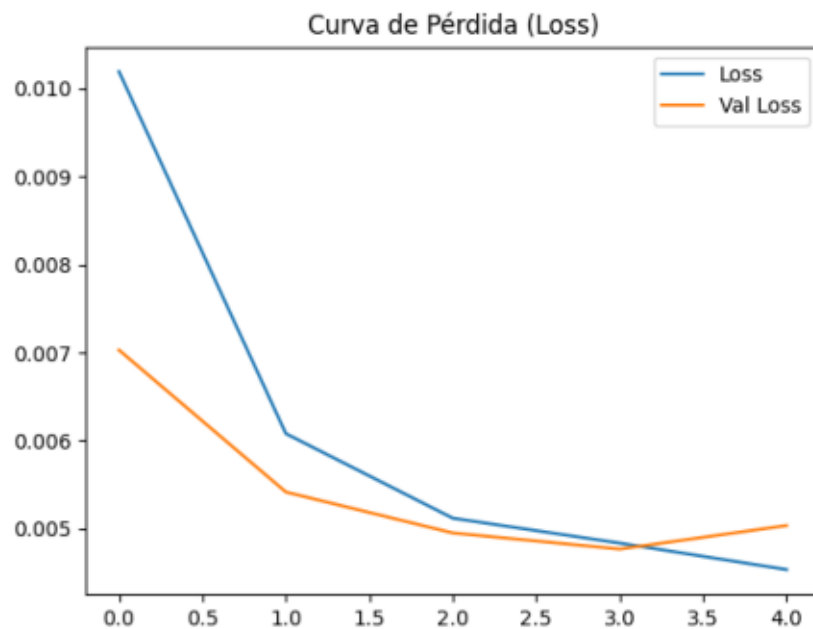
Recall: 0.777

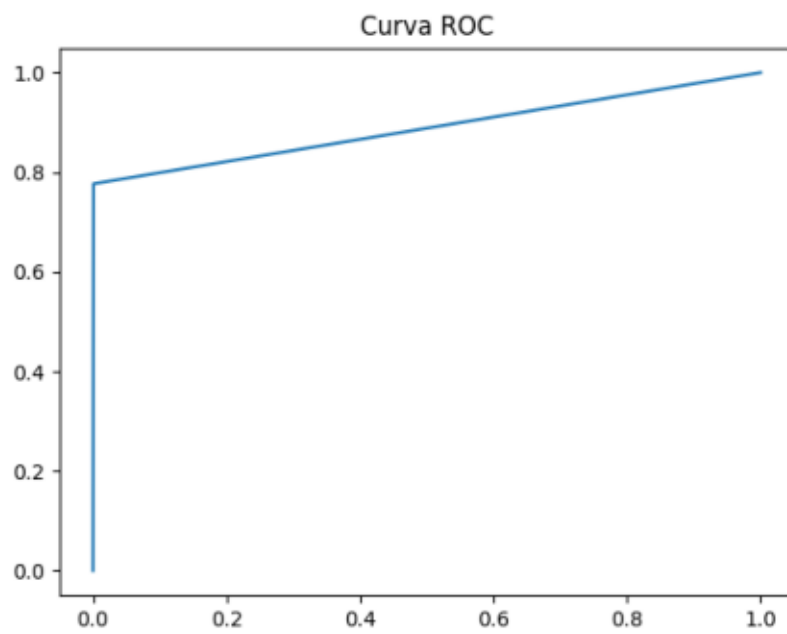
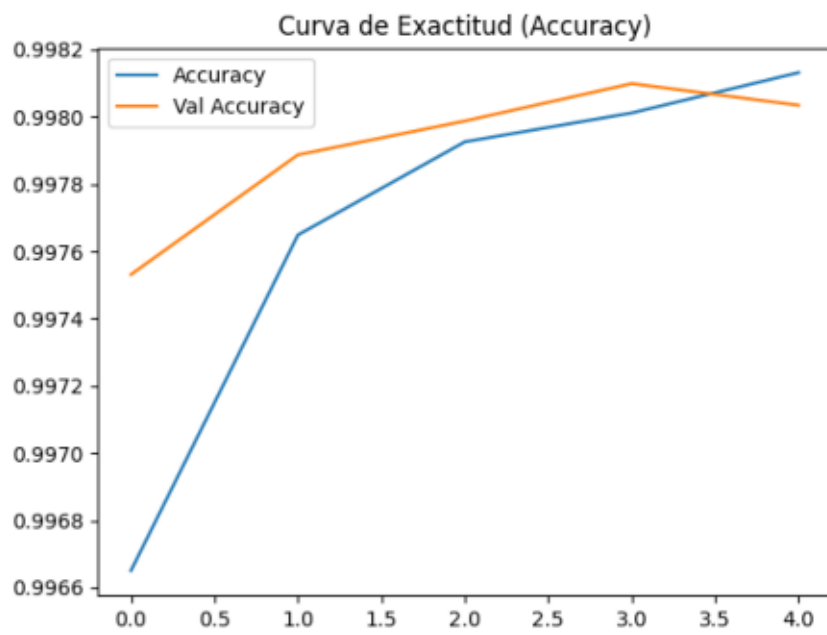
F1-Score: 0.811

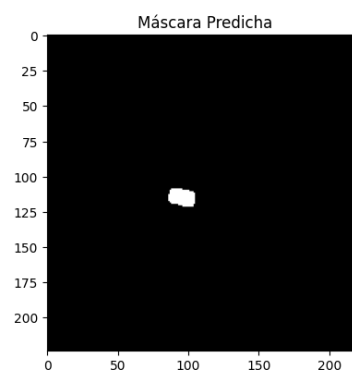
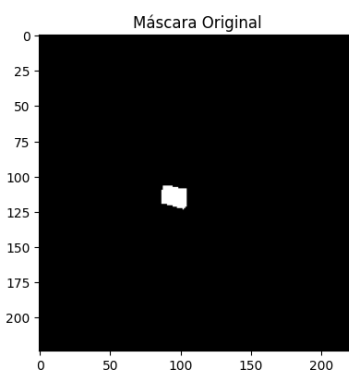
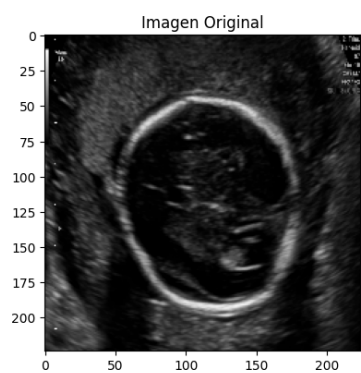
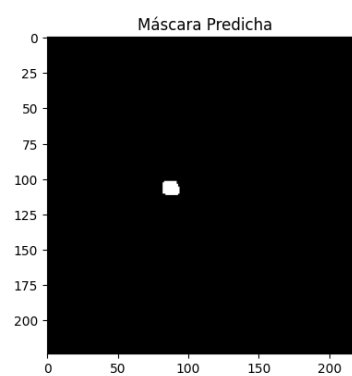
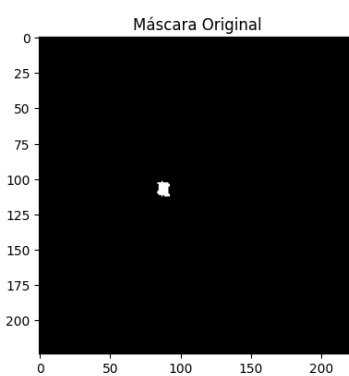
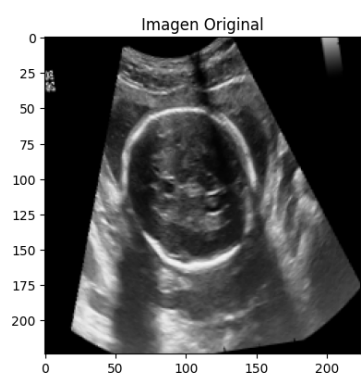
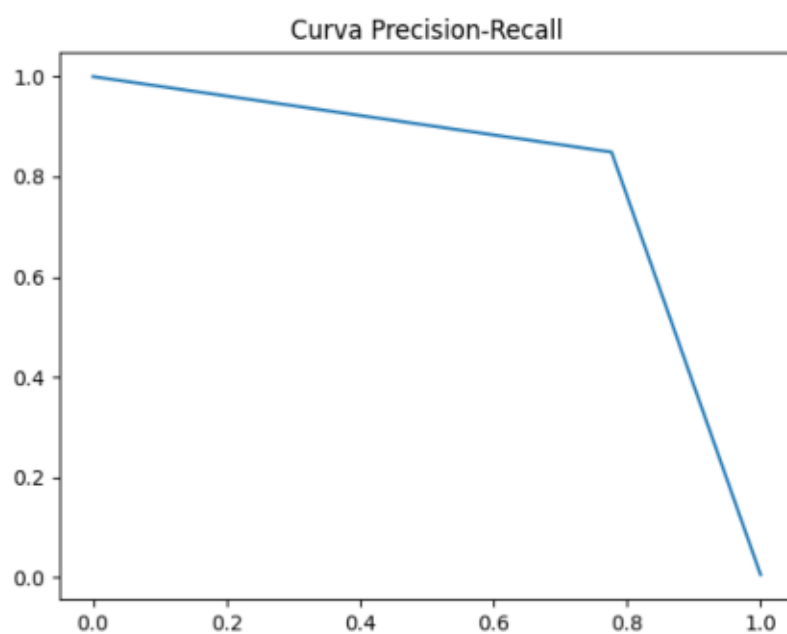
Accuracy: 0.998

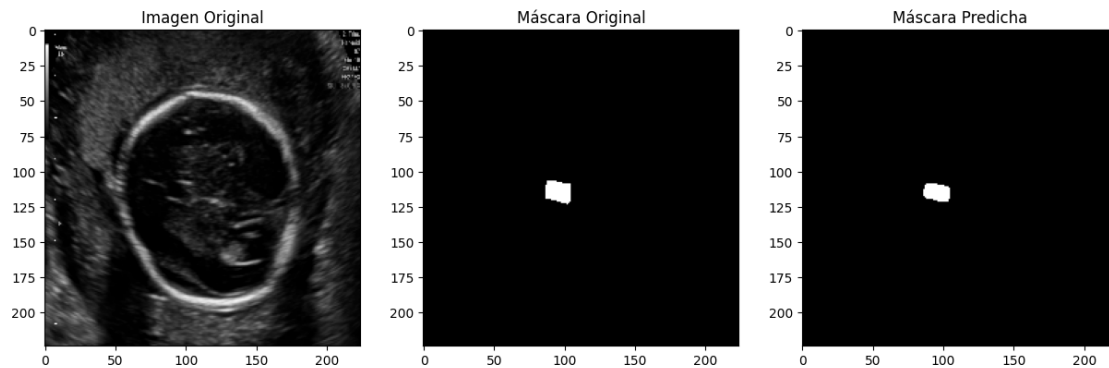
AUC ROC: 0.888

AUC PR: 0.814









Como podemos daros cuenta el anterior modelo tenia los valores de:

1. Precision: 84.9%
2. Recall: 77.1%
3. F1-Score: 80.8%
4. Accuracy: 99.8%
5. AUC ROC: 88.5%
6. AUC PR: 81%

luego de aplicar los valores al modelo podemos observar los siguientes valores

1. Precision: 84.9%
2. Recall: 77.7%
3. F1-Score: 81.1%
4. Accuracy: 99.8%
5. AUC ROC: 88.8%
6. AUC PR: 81.4%

como podemos darnos cuenta tenemos un mejoramiento en el recall en el cual aumentamos 0.6%, además de aumentos en el F1-Score de 0.3%, AUC ROC del 0.4% y de AUC PR también del 0.4% con estas mejoras dejamos por finalizado el proceso de selección del modelo para el plano Trans-Ventricular y procederemos con los otros planos Craneales.

Otro aspecto podríamos mejorar precisión si aumentamos las capas de encoder y poder que el modelo entienda características más detalladas de la figura.