Centro de Enseñanza Técnica Industrial

Visión Artificial.

Ingeniería Mecatrónica.

Tema: Proyecto 3er parcial

Platel: CETI Colomos.

Nombre: Ruiz Macías Luis Enrique - 21310196

Grado/Grupo: 6°G

Objetivo: Mostrar la probabilidad de la aparición de una masa en una radiografía y determinar si esta tiene masa mediante un umbral de acción que nos ayude a catalogar si es 0(No presenta masa) 1(presenta masa), y posteriormente mostrar los resultados en una imagen con ayuda de mathplot, de modo que se muestre la zona con la posible masa y con su probabilidad, haciendo la interfaz amigable a la vista inexperta.

Descripción: El presente proyecto se creó mediante la implementación de diferentes herramientas que nos ayudan al procesamiento de imágenes y de la implementación de un modelo de aprendizaje en Python, que toma una serie de imágenes en un dataset alojado en 'huggingface', con el fin de alimentar con información al modelo de entrenamiento con diferentes imágenes que presentan características similares a lo que viene siendo una radiografía con una masa extraña en los pulmones, de modo que es capaz de hacer un juicio con base a lo aprendido, de modo que puede decirnos si una radiografía alimentada al algoritmo presenta características de cuerpos extraños que pudieran ser reconocidas como algunas patologías de importancia medica y de posibles complicaciones de carácter grave.

Siendo específicos a continuación se presentan las herramientas utilizadas en la creación del código.

*Dataset de huuggingface '***alkzar90/NIH-Chest-X-ray-dataset'**

con las imágenes de las radiografías, el cual tiene un compendio de alrededor e 120000 imágenes disponibles con las siguientes labels:

Código Etiqueta

5 Mass

1 Atelectasis

7 Nodule

14 Pneumonia

0 No Finding

Estas etiquetas con estas imágenes son en cuestión la información que le dimos a nuestro programa para poder hacer que nuestro algoritmo aprenda y ejerza los juicios.

Nota: Solo se usaron las labels de Nofinding y de mass para hacerlo.

Con esto mencionado solo bastara con hacer varios scrpits que contengan las funciones necesarias para poder mostrar y procesar las imágenes.

Para alojar toda la información necesaria se hizo uso de Visual Studio Code, ya que es un editor de código muy noble y con mucho alcance con respecto a la cantidad de información que puede manejar y con respecto a las herramientas que nos puede facilitar el trabajo de programación, en particular en este aspecto se utilizaron las siguientes:

Editor Python

Las librerías dentro del rango de **mathplot**
Arquitectura de base en modelo **ResNet18**
Librerías y herramientas de **torch**.

También se utilizo un entorno virtual que es el cual hace manejo de la información en el equipo para poder descargar la información y en su momento poder descargarla

Estructura del código y desarrollo:
Sobre este apartado es necesario decir que se manejaron diferentes scripts, siendo estos los siguientes, los describiremos de forma breve.

Dataset_loader.py: Esta parte del código se encarga de mandar a descargar la información necesaria para poder ingresar las imágenes con sus respectivos labels y cargarlos en el proceso de train, solo se encarga de preparar el entorno de aprendizaje de la IA y de alimentar con la información al train_model.py.

Train_model.py: Este script tiene la función de tomar todo lo declara en el dataset_loader y de ejecutar los ciclos de aprendizaje de la IA, con el fin de retroalimentarla y de hacer que esta aprenda a discernir entre las imágenes y poderlas comparar en el proceso.

Predict_image.py: Este script es el final toma lo aprendido y manda a llamar al archivo creado por train que tiene la finalidad de hacer las comparaciones, este es el proceso final en el cual se mostraran los resultados de la predicción para su posterior interpretación, tanto textual como visual.

Cabe destacar que a las imágenes del dataset que se utilizo en el momento de su descarga se le aplicaron varios cambios, como convertirlas a escalas de grises y

también se redimensionaron para que sean del mismo tipo todas y que la IA en cuestión pueda crear patrones precisos sobre los cuales trabajar.

Ya creadas estos códigos procedimos a ponerlos en ejecución sin olvidar el instalar las dependencias necesarias, siendo la primera la de crear un entorno virtual en el que están alojadas las imágenes y demás información útil y necesaria.

Y el primero sería el dataset_loader, que nos dará las imágenes necesarias para realizar la el entrenamiento.
Al final de su ejecución procedemos a ejecutar el train_model, este nos dará un archivo tipo pth, el cual será mandado a llamar cada que hagamos uso del código de predict.

Y por ultimo se hace uso del predict una vez para compilar sus funciones y crear su ejecutable, para al final volver a ejecutarlo, pero ahora con el nombre de la imagen que vamos a analizar.

Errores y sus correciones:
Entre los errores mas sencillos de resolver, pero más críticos estuvo el que comprendía el cargar un dataset no acotado, teniendo que ser muy especificio con la información que requeria que se descargara y me cargara al sistema, al inicio me carga labels de otras clases y estas al ser muy pocas imágenes, me entorpecía los resultados y la precisión.

Un dataset desbalanceado también fue causa de conflicto al inicio tome 1000 imágenes del dataset, y me cargo 945 sin masas y 55 con masas, eran muy pocos ejemplos para que aprendiera, y siendo un sistema que trabaja mediante la retroalimentación constante, provocaba una baja precisión, aquí aprendimos que mientras más información más preciso iba a ser el resultado, lo que provoco que se le cargaran 1000 imágenes con masas y 1000 sin masas, de modo que esta hará que se forma un criterio mas exacto y con menos falsos negativos y positivos.

También el tema de los ciclos de trabajo es muy importante y el tipo de GPU o entorno en el que trabajes, esto es así porque, dependiendo la cantidad de ciclos y la cantidad de información, hará que tu proyecto sea mas pesado o ligero, sin mencionar que a un GPU de calidad le dará menos trabajo hacer el trabajo, lo solucionamos acotando el proyecto a una fase de prueba para propósito de la materia, lo que la vuelve más noble en su versión.

Y por último el procesamiento de imágenes, es el ultimo apartado a tomar en cuenta, se sabía que se tenían que procesar las imágenes en cuestión para que fueran admisibles al código, por lo que se les aplico una transformación en escala de grises

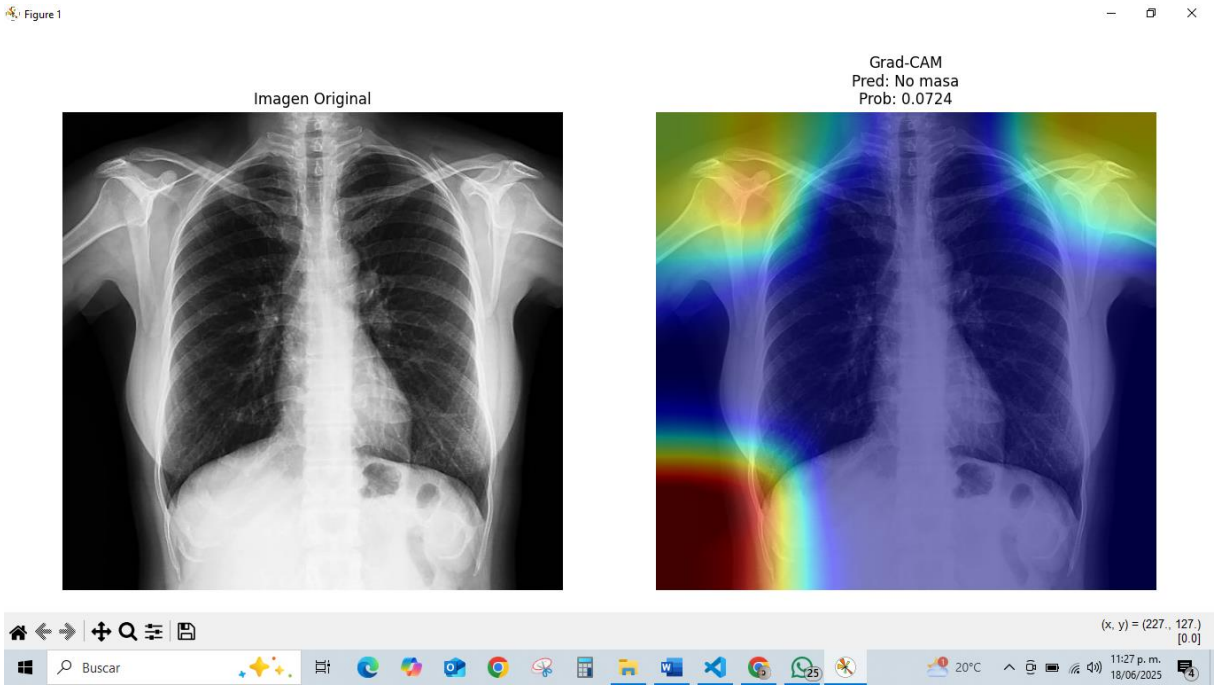
y en dimensiones, esto para que fueran lo mas similares entre ellas y nos permitiera hacer que la IA aprendiera a distinguir entre una y otra imagen con mas y sin masa.

Muestras de imágenes:

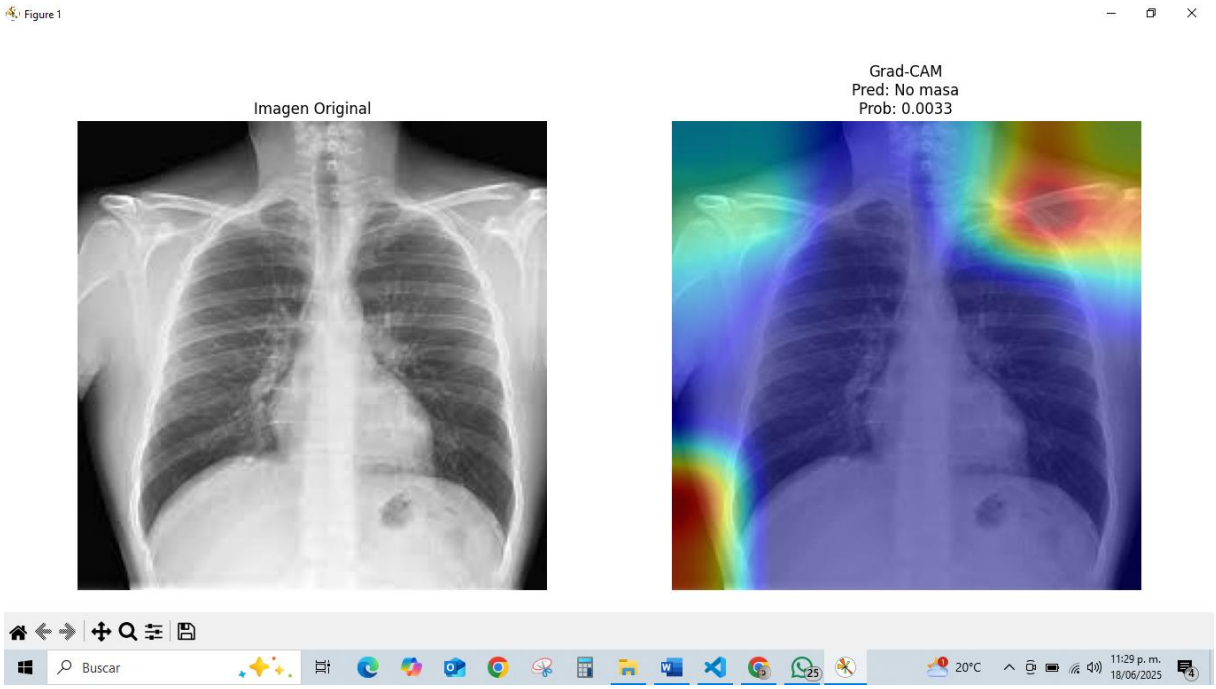
Imágenes Nombradas Masa1,2 y 3 de izquierda a derecha.

Imágenes nombradas normal1, 2 y 3 de izquierda a derecha.

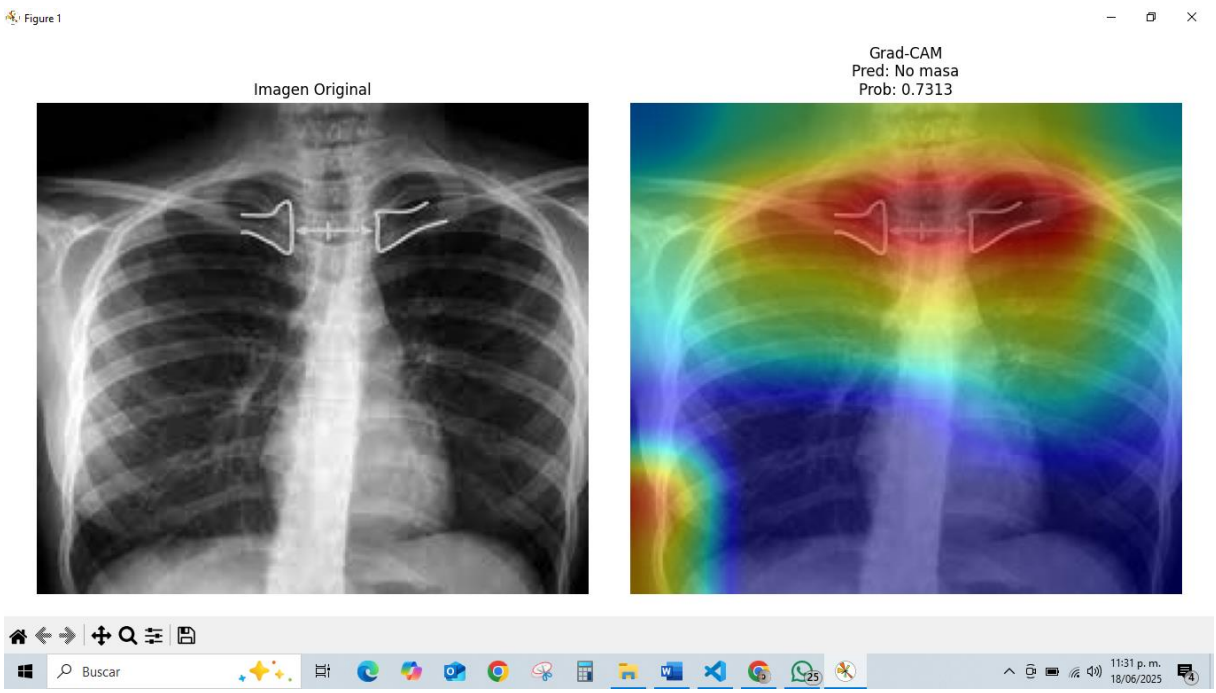
A continucion se muestran las duiferentes pruebas realizadas y la interpretacion de los resutados.



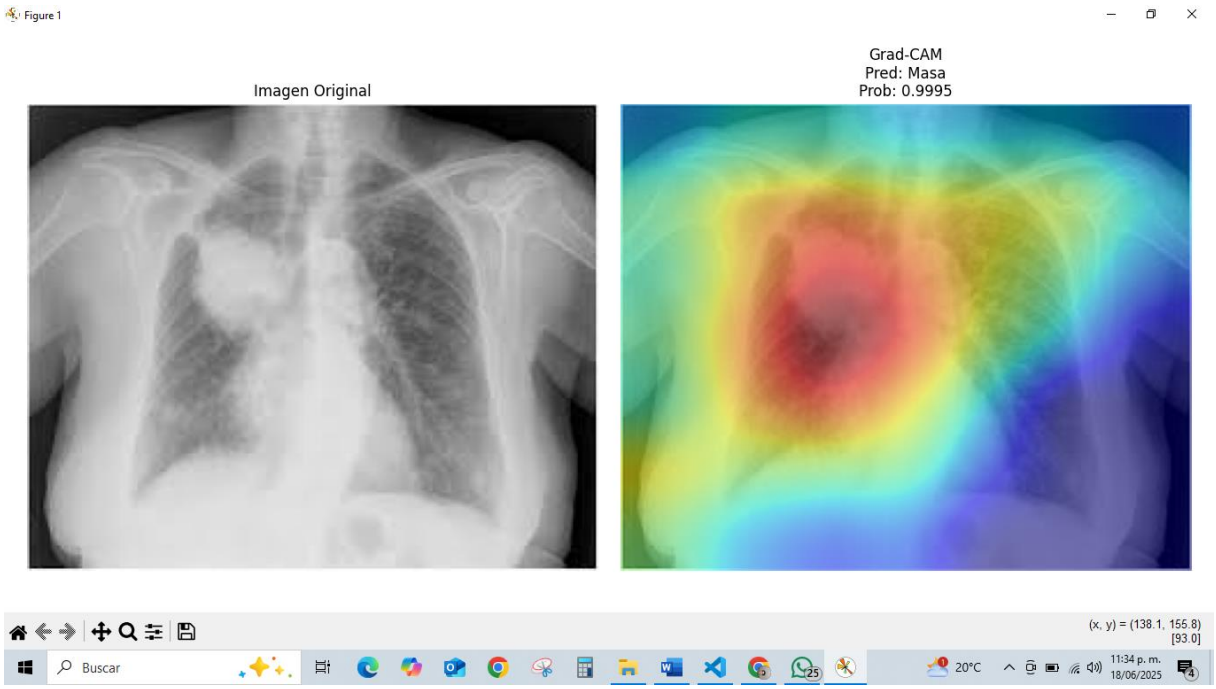
Prueba hecha con la Imagen nombrada Normal, presenta un 7.24% de chanza.



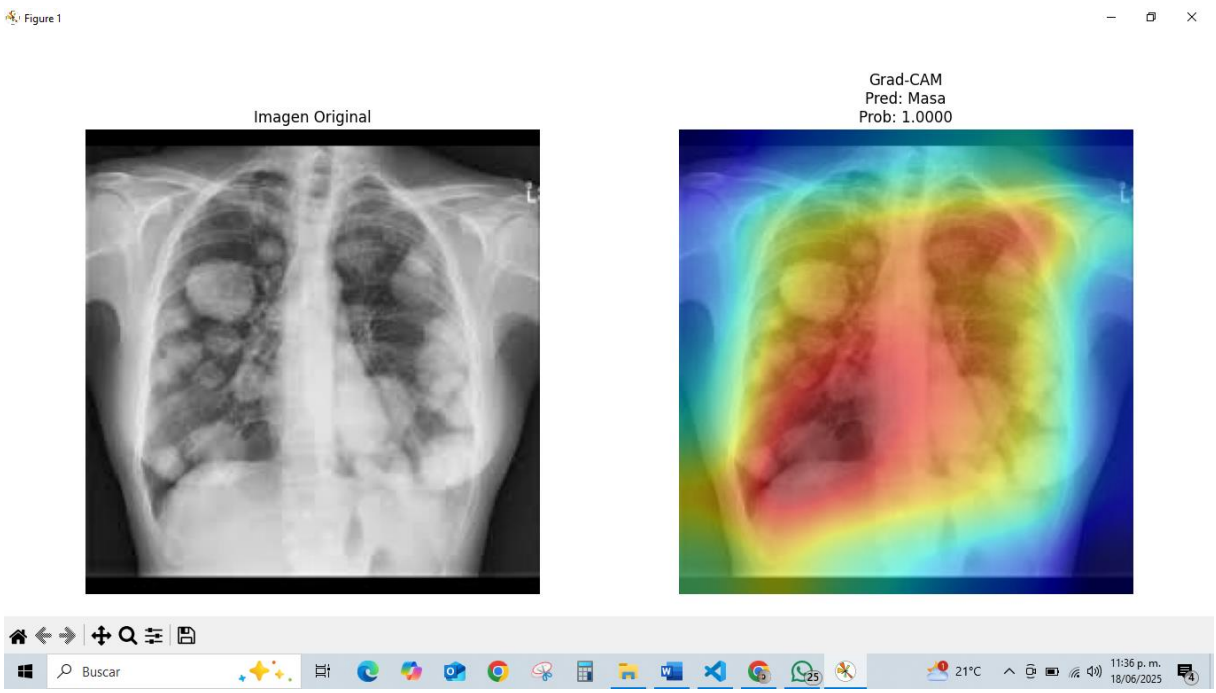
Normal2, con probabilidad del 0.33%



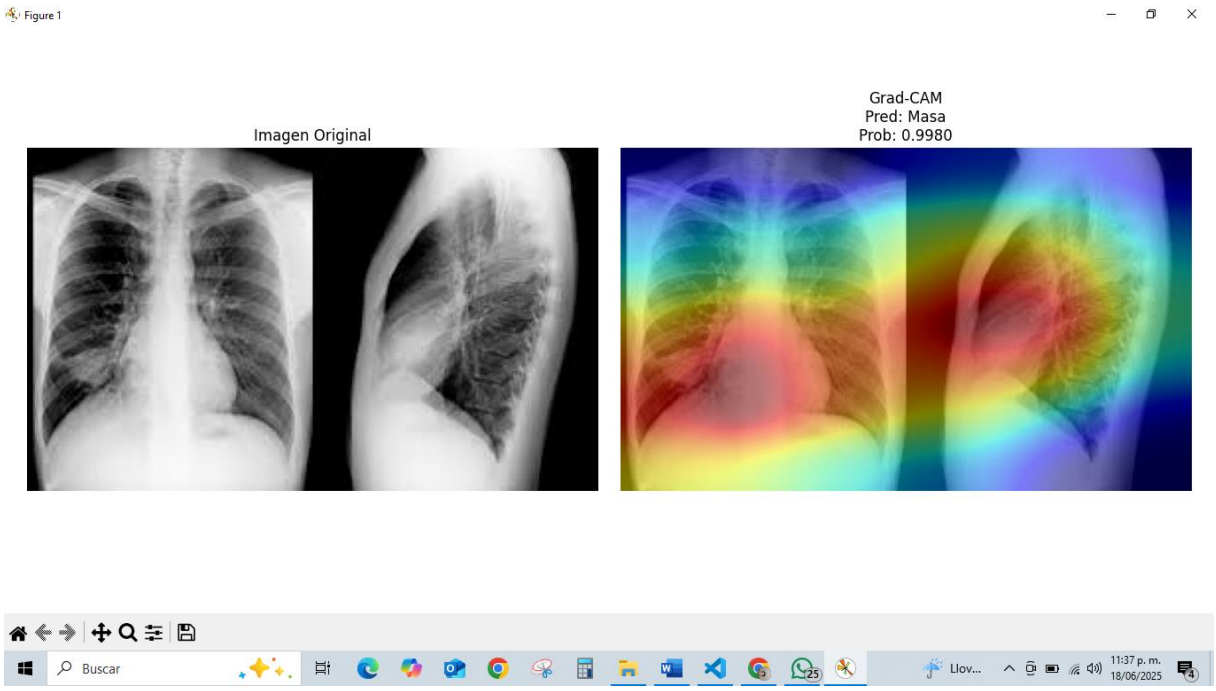
Normal3, presenta un 73.13% de activación, pero nótese esta fuera de rango del 85% por lo que no se considera viable, además de la ubicación, se cree que es por las imágenes con esos bordes definidos, posible placa.



Masa1, con 99.95% de chanza de presentar masa, mayor al 85% por lo que si detecta masa.



Masa2, presenta una probabilidad del 100%, imagen con metástasis.



Masa3, presenta una probabilidad de 99.80%, por lo que hay una masa presente.

La interpretación necesita de un ojo experto para delimitar si los resultados son objetivos y confiables, la grad cam muestra donde se marcó más la razón por la cual se disparo la probabilidad, esta dibuja en negro y rojo en donde están las zonas con masas.

Estos resultados de las pruebas son satisfactorios en cuanto cumplen los objetivos del proyecto propuestos al inicio, vuelve las imágenes amigables al ojo inexperto.

Códigos: A continuación, se pegarán los códigos con los cuales se logró el proyecto:

Predict_image.py

```
import torch
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import sys
from train_model import ResNet18Gray

# === Configuración ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
transform = transforms.Compose([
    transforms.Grayscale(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])

# === Grad-CAM ===
class GradCAM:
    def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer
        self.gradients = None
        self.activations = None

        # Hook para capturar gradientes
        self.target_layer.register_forward_hook(self.save_activation)
        self.target_layer.register_full_backward_hook(self.save_gradient)

    def save_activation(self, module, input, output):
        self.activations = output.detach()
    def save_gradient(self, module, grad_input, grad_output):
        self.gradients = grad_output[0].detach()

    def generate(self, input_tensor, class_idx):
        self.model.zero_grad()
        output = self.model(input_tensor)
```

```
        loss = output[0, class_idx]
        loss.backward()

        weights = self.gradients.mean(dim=[2, 3], keepdim=True)  # GAP
        cam = (weights * self.activations).sum(dim=1, keepdim=True)
        cam = torch.relu(cam)
        cam = cam.squeeze().cpu().numpy()
        cam = cam - cam.min()
        cam = cam / cam.max()
        return cam

# === Función de predicción y visualización ===
def predict_and_visualize(image_path):
    # Cargar imagen
    image = Image.open(image_path).convert("L")
    img_tensor = transform(image).unsqueeze(0).to(device)

    # Modelo
    model = ResNet18Gray().to(device)
    model.load_state_dict(torch.load("p3vision_model.pth",
map_location=device))
    model.eval()

    # GradCAM
    target_layer = model.model.layer4[-1]  # Último bloque conv de ResNet18
    grad_cam = GradCAM(model, target_layer)

    # Forward
    output = model(img_tensor)
    probs = torch.softmax(output, dim=1)
    prob_mass = probs[0, 1].item()
    pred_class = 1 if prob_mass >= 0.85 else 0

    # Grad-CAM Map
    cam = grad_cam.generate(img_tensor, class_idx=1)

    # Mostrar resultados
    plt.figure(figsize=(8, 4))

    # Imagen original
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title("Imagen Original")
    plt.axis('off')
```

```
    # Grad-CAM heatmap
    plt.subplot(1, 2, 2)
    heatmap = Image.fromarray(np.uint8(255 * cam)).resize(image.size)
    heatmap = np.array(heatmap)
    plt.imshow(image, cmap='gray')
    plt.imshow(heatmap, cmap='jet', alpha=0.5)
    plt.title(f"Grad-CAM\nPred: {'Masa' if pred_class == 1 else 'No
masa'}\nProb: {prob_mass:.4f}")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# === Punto de entrada ===
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python predict_gradcam.py <ruta_imagen>")
        sys.exit(1)

    predict_and_visualize(sys.argv[1])
```

train_model.py:

```
""Este módulo contiene la definición del modelo ResNet18Gray adaptado a
imágenes en escala de grises."""
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from torchvision import models
from dataset_loader import load_mass_dataset, preprocess_images

# Configuración
num_epochs = 13
batch_size = 32
learning_rate = 0.001
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Modelo mejorado: ResNet18 adaptado a escala de grises
class ResNet18Gray(nn.Module):
    def __init__(self):
```

```
        super(ResNet18Gray, self).__init__()
        self.model = models.resnet18(pretrained=True)

        # Adaptar la primera capa para imágenes en escala de grises (1 canal)
        self.model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2,
padding=3, bias=False)

        # Reemplazar capa final para clasificación binaria (masa vs no masa)
        num_features = self.model.fc.in_features
        self.model.fc = nn.Linear(num_features, 2)

    def forward(self, x):
        return self.model(x)

# Entrenamiento
if __name__ == "__main__":
    print("Cargando dataset balanceado...")
    images, labels = load_mass_dataset(num_mass=1000, num_no_mass=1000)
    images = preprocess_images(images)

    # División en entrenamiento y validación
    train_imgs, test_imgs, train_lbls, test_lbls = train_test_split(
        images, labels, test_size=0.2, stratify=labels, random_state=42
    )

    train_imgs = torch.stack(train_imgs)
    test_imgs = torch.stack(test_imgs)
    train_lbls = torch.tensor(train_lbls)
    test_lbls = torch.tensor(test_lbls)

    print("Tamaño del set de entrenamiento:", train_imgs.shape)
    print("Tamaño del set de validación:", test_imgs.shape)

    # DataLoaders
    train_loader = DataLoader(TensorDataset(train_imgs, train_lbls),
batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(TensorDataset(test_imgs, test_lbls),
batch_size=batch_size)

    # Inicializar modelo, pérdida y optimizador
    model = ResNet18Gray().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
    # Entrenamiento con evaluación
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_loss = running_loss / len(train_loader)

        # Evaluación en validación
        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += targets.size(0)
                correct += (predicted == targets).sum().item()

        accuracy = 100 * correct / total
        print(f"\n--- Época {epoch + 1}/{num_epochs} ---")
        print(f"Loss promedio entrenamiento: {avg_loss:.4f}")
        print(f"Accuracy en validación: {accuracy:.2f}%")

    # Guardar modelo
    print("\nEntrenamiento completado. Modelo guardado como
'p3vision_model.pth'")
    torch.save(model.state_dict(), "p3vision_model.pth")
```

Dataseet_loader.py:

```
"""Paquete para subir imagenes al trian y caragar las iamgenes al
programa."""
def load_mass_dataset(num_mass=1000, num_no_mass=1000):
    """
    Carga un conjunto balanceado:
    - num_mass imágenes con solo la etiqueta 'Mass' (código 5)
    - num_no_mass imágenes con etiqueta 'No Finding' solamente (código 0)
    """
    dataset = load_dataset(
        "alkzar90/NIH-Chest-X-ray-dataset",
        name="image-classification",
        split="train",
        trust_remote_code=True
    )

    mass_images = []
    mass_labels = []
    no_mass_images = []
    no_mass_labels = []

    for item in dataset:
        tags = item['labels']
        image = item['image']

        # Solo imágenes con etiqueta 'Mass' y sin otras condiciones (opcional)
        if 5 in tags and len(mass_images) < num_mass and len(tags) == 1:
            mass_images.append(image)
            mass_labels.append(1)

        # Solo imágenes que son 100% normales (No Finding)
        elif tags == [0] and len(no_mass_images) < num_no_mass:
            no_mass_images.append(image)
            no_mass_labels.append(0)

        if len(mass_images) >= num_mass and len(no_mass_images) >=
num_no_mass:
            break

    print(f" Se han cargado {len(mass_images)} imágenes con 'Mass' y
{len(no_mass_images)} sin 'Mass' (100% normales).")

    images = mass_images + no_mass_images
    labels = mass_labels + no_mass_labels
```

```
    return images, labels
from datasets import load_dataset
from torchvision import transforms
def load_mass_dataset(num_mass=1000, num_no_mass=1000):
    """
    Carga un conjunto balanceado:
    - num_mass imágenes con solo la etiqueta 'Mass' (código 5)
    - num_no_mass imágenes con etiqueta 'No Finding' solamente (código 0)
    """
    dataset = load_dataset(
        "alkzar90/NIH-Chest-X-ray-dataset",
        name="image-classification",
        split="train",
        trust_remote_code=True
    )

    mass_images = []
    mass_labels = []
    no_mass_images = []
    no_mass_labels = []

    for item in dataset:
        tags = item['labels']
        image = item['image']

        # Solo imágenes con etiqueta 'Mass' y sin otras condiciones
        if 5 in tags and len(mass_images) < num_mass and len(tags) == 1:
            mass_images.append(image)
            mass_labels.append(1)

        # Solo imágenes 100% normales
        elif tags == [0] and len(no_mass_images) < num_no_mass:
            no_mass_images.append(image)
            no_mass_labels.append(0)

        if len(mass_images) >= num_mass and len(no_mass_images) >=
num_no_mass:
            break

    print(f" Se han cargado {len(mass_images)} imágenes con 'Mass' y
{len(no_mass_images)} sin 'Mass' (100% normales).")

    images = mass_images + no_mass_images
```

```
    labels = mass_labels + no_mass_labels

    return images, labels

def preprocess_images(images):
    """
    Preprocesa imágenes para el modelo:
    - Redimensiona a 224x224
    - Convierte a escala de grises
    - Normaliza
    """
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5], std=[0.5])
    ])
    return [transform(img.convert("L")) for img in images]
```

Los presentes códigos son necesarios para poder llevar a cabo el programa en custion.