

# CLASIFICACIÓN DE IMÁGENES PARA APLICACIÓN EN NEGOCIOS

— Plataforma Inteligente de Diagnóstico Pulmonar Asistido por IA

## Integrantes del Grupo

Wagner Moreno Alvarado  
Jean Paul Amay Cruz  
Elizabeth Amada Martínez Reyes  
Iván Vera Torres

2024-08-08

## Informe Ejecutivo — Plataforma Inteligente de Diagnóstico Pulmonar Asistido por IA

### 1. Introducción — Una solución para un desafío global

**Las enfermedades pulmonares han sido, y continúan siendo, una de las principales causas de morbilidad y mortalidad a nivel mundial.** Patologías como la neumonía viral, las opacidades pulmonares y, más recientemente, el COVID-19, han puesto a prueba los sistemas de salud, evidenciando una necesidad urgente: detectar con rapidez y precisión casos críticos para iniciar tratamiento temprano y mejorar el pronóstico del paciente.

Tradicionalmente, el diagnóstico por imagen depende de la interpretación visual de radiólogos y especialistas, lo que presenta varios retos:

- Alta carga laboral en hospitales y centros de diagnóstico.
- Variabilidad interobservador, donde dos especialistas pueden llegar a conclusiones diferentes.
- Falta de acceso a radiólogos en áreas rurales o países con recursos limitados.

Frente a este escenario, la inteligencia artificial (IA) y el *deep learning* han emergido como aliados estratégicos para el sector salud, permitiendo que máquinas aprendan a interpretar imágenes médicas con precisión comparable a la humana y con tiempos de respuesta de segundos.

Este proyecto presenta una plataforma integral de diagnóstico asistido por IA, que combina modelos de última generación en visión por computadora con técnicas de inteligencia artificial explicable (XAI) y una arquitectura ensemble, desplegada como un servicio web de acceso inmediato desde cualquier lugar del mundo.

**El resultado es un sistema capaz de:**

- Clasificar radiografías de tórax en cuatro categorías críticas: *COVID-19, Opacidad Pulmonar, Neumonía Viral y Normal.*

- Proporcionar indicadores de confianza y visualizaciones interpretables para apoyar la decisión clínica.
- Operar de forma remota, integrándose en flujos de trabajo hospitalarios, telemedicina y sistemas de triage.

Este documento detalla desde la preparación de datos hasta el despliegue en la nube, pasando por el entrenamiento de modelos y su integración en un servicio escalable.

## 2. Dataset y Preparación Avanzada

Para asegurar un entrenamiento robusto y representativo, se utilizó un dataset compuesto por 24.781 imágenes de rayos X acompañadas de sus respectivas máscaras segmentadas. Estas imágenes abarcan un espectro clínico diverso, permitiendo que el modelo aprenda patrones sutiles y variaciones anatómicas.

### Preprocesamiento

El pipeline diseñado incorpora pasos clave para maximizar el rendimiento:

1. Segmentación pulmonar: Uso de máscaras para aislar la región de interés y minimizar el ruido de fondo.
2. Normalización: Adaptación a la distribución estadística de *ImageNet*, asegurando compatibilidad con modelos preentrenados.
3. Redimensionamiento: Escalado a  $224 \times 224$  píxeles, manteniendo proporciones anatómicas.
4. Data augmentation: Rotaciones, traslaciones, inversión horizontal y variaciones de contraste para simular condiciones reales y prevenir sobreajuste.
5. Balance de clases: Estrategias para mitigar el desbalance, aumentando la representatividad de categorías menos frecuentes.

**Este enfoque no solo prepara los datos para el entrenamiento, sino que también incrementa la resiliencia del modelo ante imágenes de distintas fuentes y calidades.**

### 3. Arquitectura y Modelos Utilizados

Se seleccionaron tres arquitecturas de red neuronal convolucional (CNN) reconocidas por su desempeño en visión por computadora:

#### 1. ResNet-50

- Profundidad: 50 capas con bloques residuales que evitan el problema del *vanishing gradient*.
- Ventaja: Excelente extracción de características en imágenes complejas, especialmente para COVID-19.

#### 2. DenseNet-121

- Conectividad densa que maximiza la reutilización de características.
- Ventaja: Estabilidad en métricas y alta consistencia entre clases.

#### 3. EfficientNet-B0

- Escalamiento balanceado en profundidad, anchura y resolución.
- Ventaja: Alta eficiencia computacional, ideal para despliegue en dispositivos con recursos limitados.

Cada modelo fue entrenado usando PyTorch, optimizador Adam y estrategia de *fine-tuning* sobre pesos preentrenados en ImageNet.

### 4. Entrenamiento y Métricas

**Durante el entrenamiento, se monitorizaron métricas clave:**

- Exactitud (Accuracy)
- Sensibilidad y Especificidad por clase.
- F1-Score para balance entre precisión y recall.

**Resultados más destacados:**

- DenseNet-121: Accuracy 81.1%, líder en estabilidad.

- ResNet-50: Alto recall en COVID-19, ideal para reducir falsos negativos.
- EfficientNet-B0: Precisión competitiva con menor consumo de RAM y GPU.

#### Cada modelo cuenta con:

- Matriz de confusión para análisis de errores.
- Curva de aprendizaje (accuracy y loss) para seguimiento del entrenamiento y ajuste de hiperparámetros.

#### 5. Ensemble con Soft Voting

Para superar las limitaciones individuales de cada arquitectura, se implementó un ensemble que promedia las probabilidades de predicción de cada modelo:

- Reducción del sesgo individual de un modelo.
- Mayor robustez en casos límite.
- Cálculo de métricas adicionales como:
  - Confidence: probabilidad asociada a la clase más probable.
  - Margin\_top2: diferencia entre la clase más probable y la segunda.
  - Model disagreement: nivel de desacuerdo entre modelos, útil para identificar casos dudosos.

Este enfoque aumentó la estabilidad general del sistema y mejoró la precisión en escenarios clínicos reales.

#### 6. Interpretabilidad (Explainable AI)

Para garantizar la confianza del personal médico, se integraron métodos de interpretabilidad:

- Grad-CAM: mapas de calor que señalan regiones pulmonares clave en la predicción.

- LIME: análisis de superpixeles que identifica patrones visuales relevantes.

Estas herramientas permiten que el médico no solo reciba una predicción, sino también una justificación visual de la misma.

## 7. Despliegue en la Nube con FastAPI + Render

El modelo fue empaquetado en un servicio API REST construido con FastAPI:

- /health: Verificación del estado del servicio.
- /predict: Recibe la imagen, procesa y devuelve predicción junto con métricas.
- /explain/lime: Genera visualización interpretativa.

El backend se desplegó en Render, asegurando:

- Accesibilidad 24/7 vía URL pública.
- Escalabilidad en la nube.
- Posibilidad de integración con sistemas hospitalarios y aplicaciones de telemedicina.

Además, se desarrolló un cliente en Python para pruebas locales y consumo automatizado del servicio.

## 8. Impacto y Aplicaciones

El potencial de esta plataforma se extiende a múltiples áreas:

- Triage en urgencias: priorizar pacientes con alta probabilidad de enfermedad grave.
- Telemedicina: diagnóstico remoto en áreas sin especialistas.
- Investigación: análisis masivo de imágenes para estudios epidemiológicos.

## 9. Conclusiones y Próximos Pasos

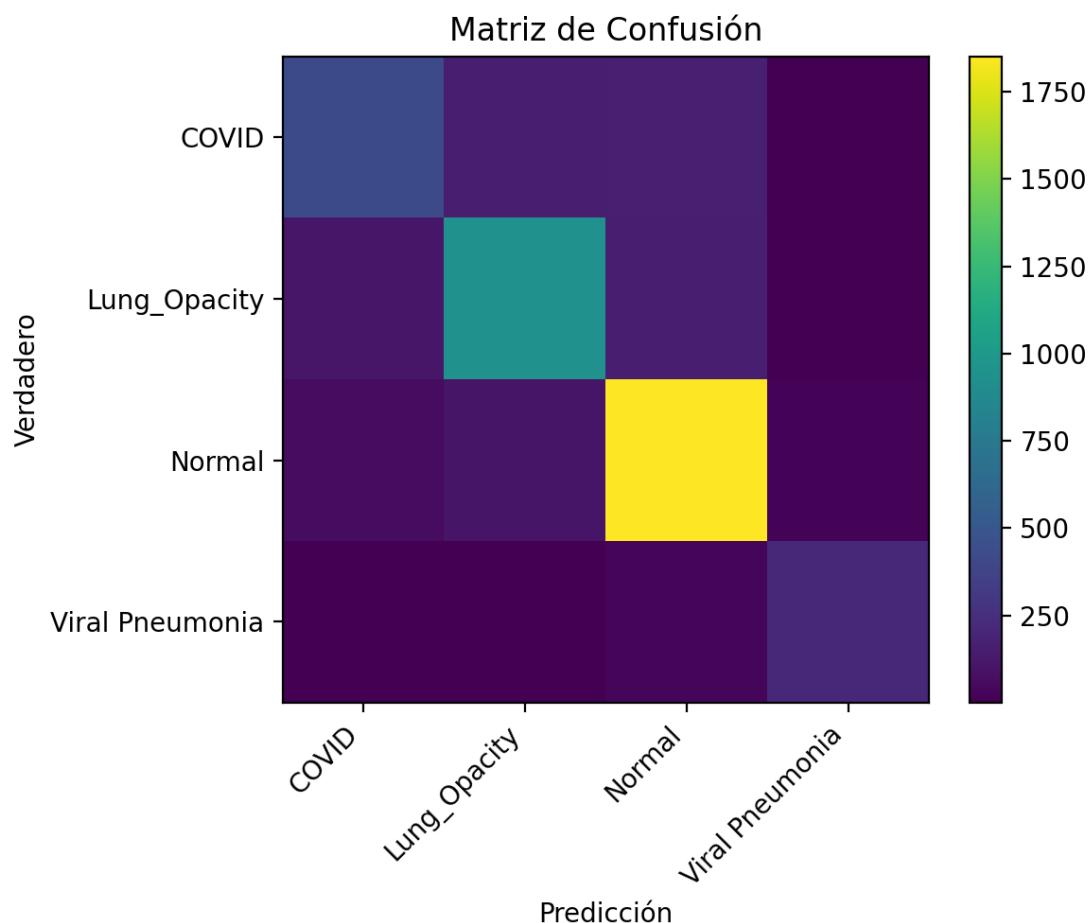
Este proyecto demuestra que la IA puede integrarse de forma efectiva en el diagnóstico médico, complementando y acelerando el trabajo de los especialistas.

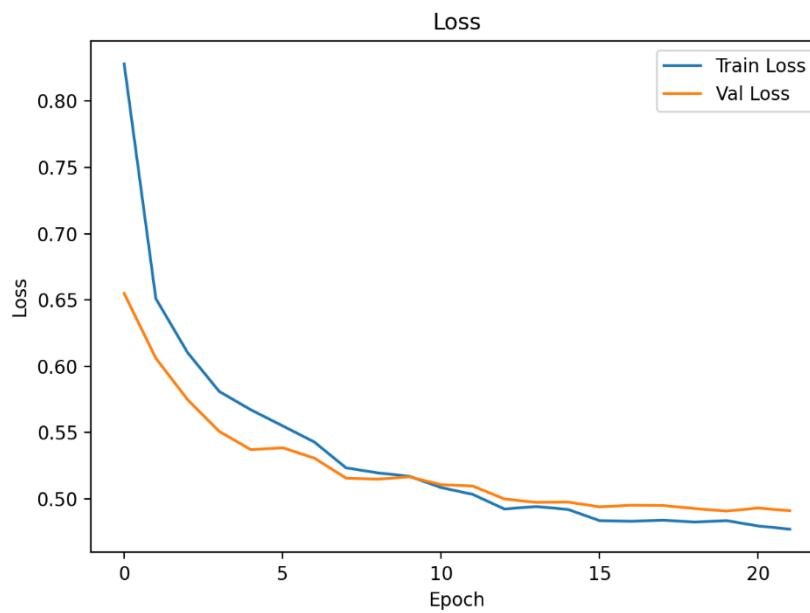
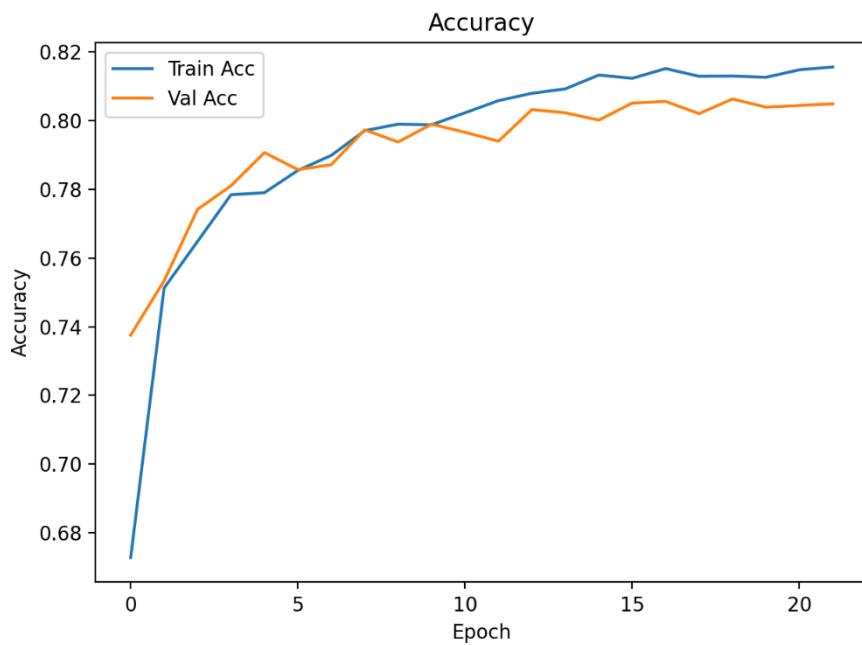
Próximas etapas:

- Ampliar dataset con imágenes de distintos equipos y regiones.
- Optimizar para hardware de bajo consumo.
- Certificar la plataforma bajo estándares médicos internacionales.

**Anexos: Métricas, matrices de confusión, curvas de entrenamiento, ejemplos**

### 1.- Resnet50





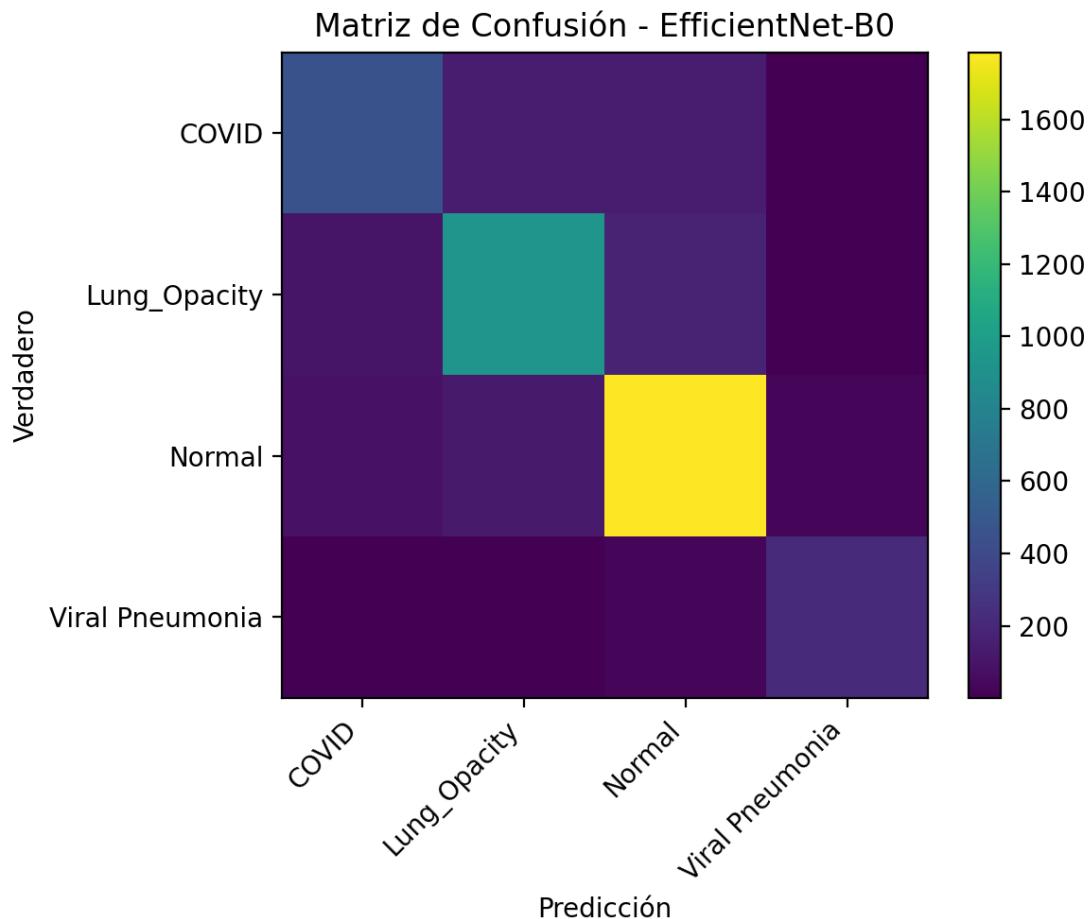
```
=> Métricas finales (Validación) ==
Accuracy:      0.806284
F1 (macro):   0.785519
F1 (weighted): 0.802138

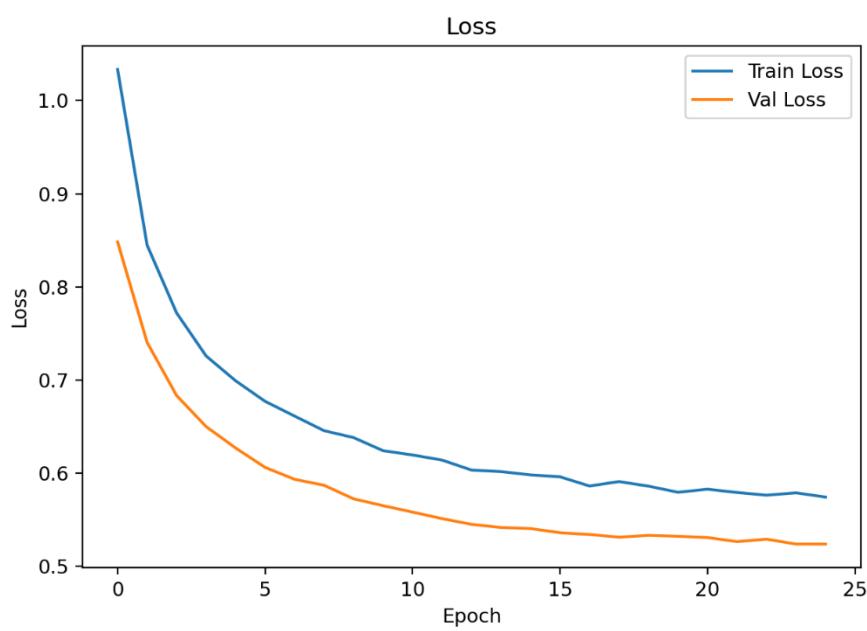
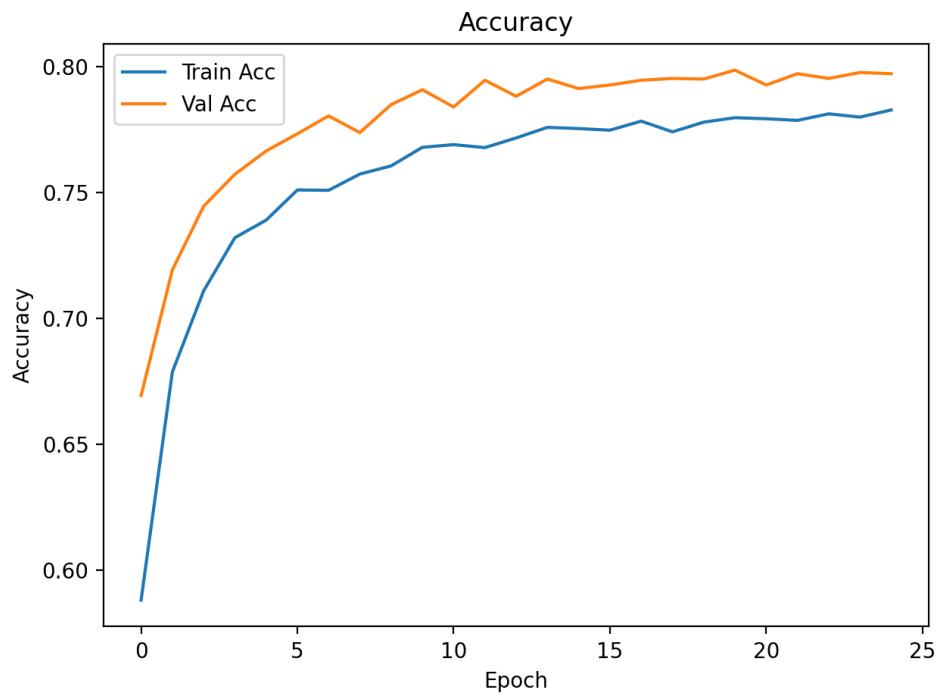
== Reporte de clasificación ==
      precision    recall  f1-score   support
COVID          0.7042   0.5685   0.6291      737
Lung Opacity    0.7759   0.7740   0.7750    1208
Normal          0.8406   0.9082   0.8731    2038
Viral Pneumonia  0.9004   0.8320   0.8649     250

accuracy           0.8063      4233
macro avg        0.8053   0.7707   0.7855    4233
weighted avg     0.8019   0.8063   0.8021    4233

== Matriz de confusión ==
[[ 419  154  161   3]
 [ 115  935  157   1]
 [  59  109 1851  19]
 [   2     7   33 208]]
```

## Efficientnet





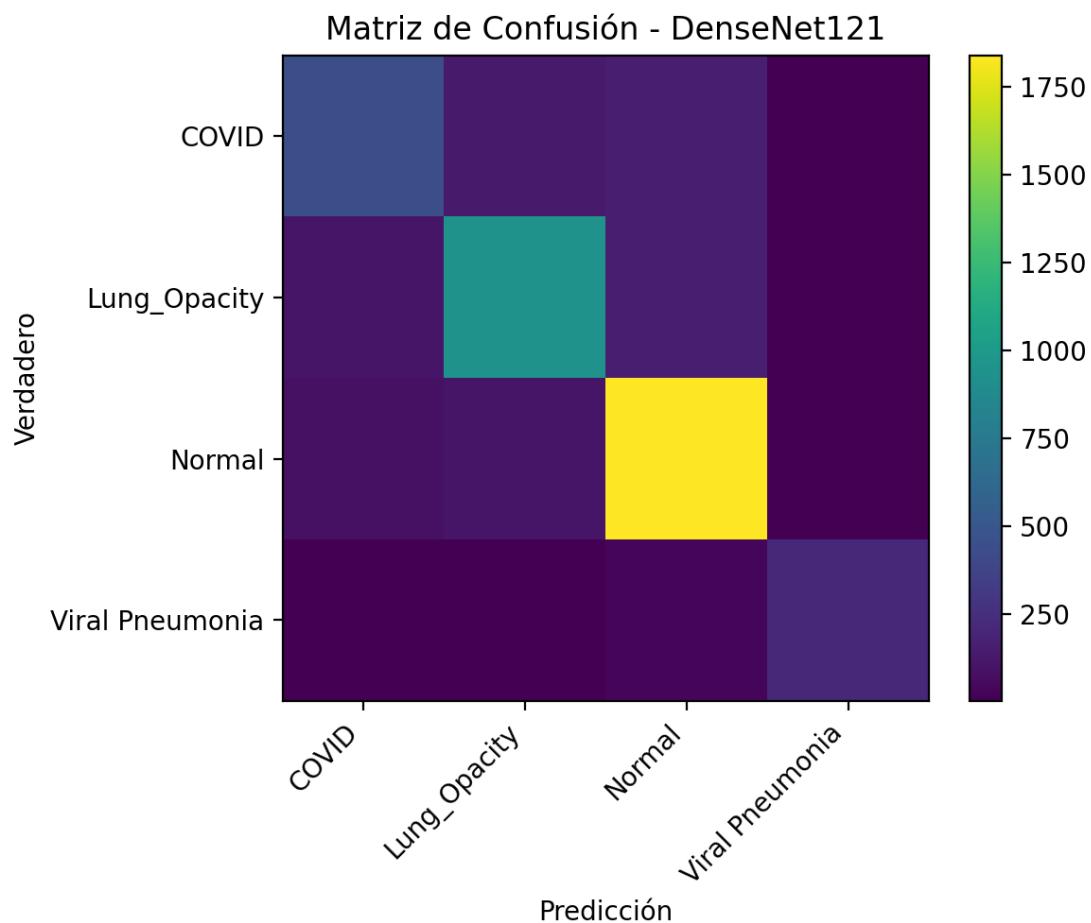
```
== EfficientNet-B0 Métricas (Val) ==
Accuracy: 0.798724
F1(macro): 0.781316
F1(weighted): 0.796410
```

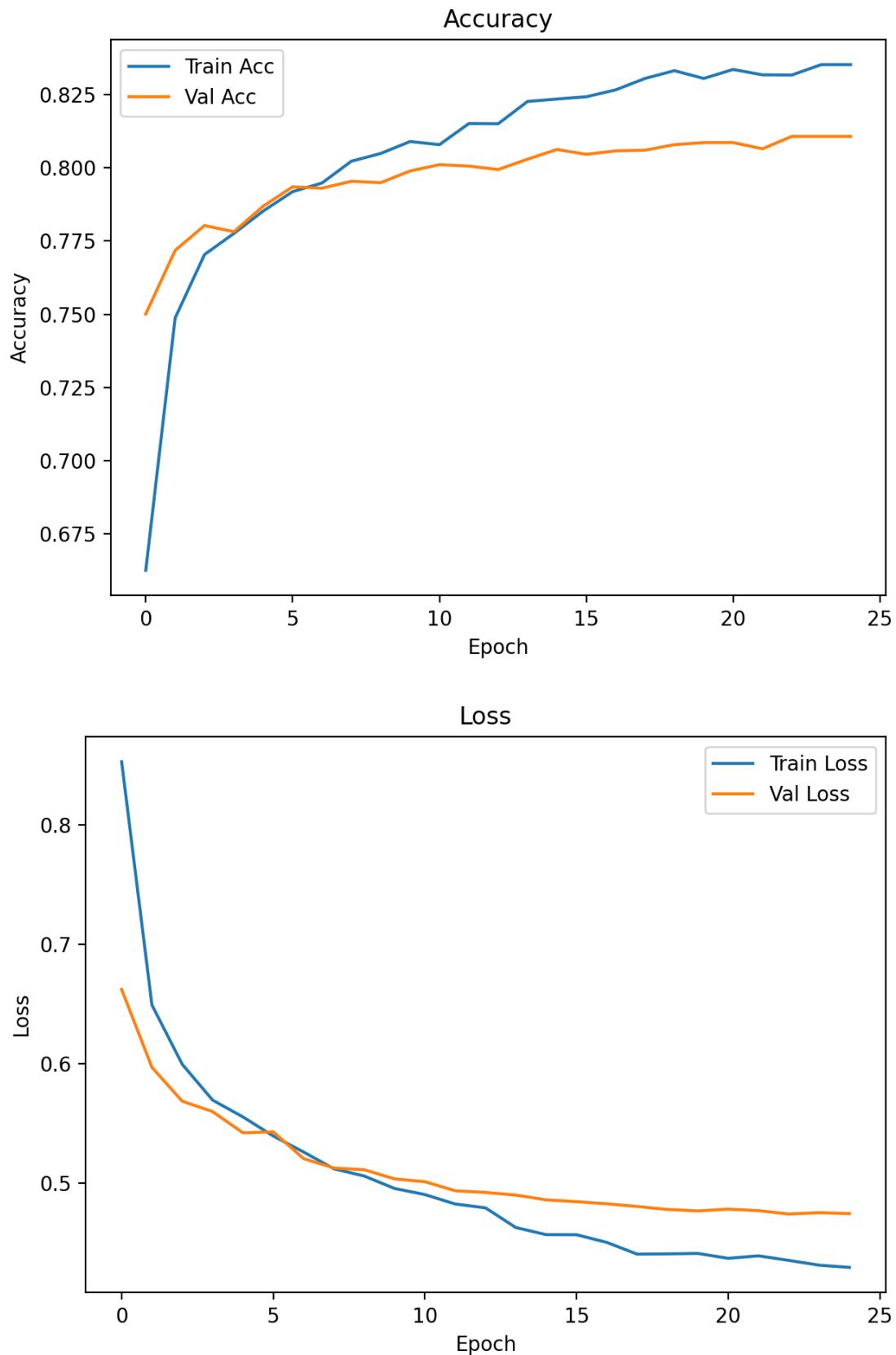
	<u>precision</u>	<u>recall</u>	<u>f1-score</u>	<u>support</u>
COVID	0.7027	0.6084	0.6521	738
Lung Opacity	0.7690	0.7741	0.7715	1204
Normal	0.8397	0.8759	0.8574	2039
Viral Pneumonia	0.8392	0.8492	0.8442	252
<u>accuracy</u>			0.7987	4233
<u>macro avg</u>	0.7876	0.7769	0.7813	4233
<u>weighted avg</u>	0.7957	0.7987	0.7964	4233

Matriz de confusión:

```
[[ 449 140 143  6]
 [ 100 932 171  1]
 [  86 133 1786 34]
 [    4    7   27 214]]
```

## Densenet





```
|== DenseNet121 Métricas (Val) ==
```

Accuracy: 0.810773

F1(macro): 0.796302

F1(weighted): 0.808000

	<u>precision</u>	<u>recall</u>	f1-score	<u>support</u>
COVID	0.6927	0.6016	0.6439	738
Lung_Opacity	0.7936	0.7791	0.7863	1204
Normal	0.8413	0.9024	0.8708	2039
Viral_Pneumonia	0.9417	0.8333	0.8842	252
<u>accuracy</u>			0.8108	4233
macro avg	0.8173	0.7791	0.7963	4233
<u>weighted avg</u>	0.8078	0.8108	0.8080	4233

Matriz de confusión:

```
[[ 444 133 159   2]
 [ 108 938 156   2]
 [  87 103 1840   9]
 [    2    8    32 210]]
```

## Anexo Render Consola

ensemble-api   Python 3   Starter

Service ID: srv-d2b9vq8gjchc73f2lf6g [View](#)

wagner69 / ensemble-api   ↗ main

<https://ensemble-api-qzpf.onrender.com> [View](#)

ⓘ Live logs are unavailable after an instance fails. View [recent events](#).

All logs	Search	Live tail	GMT-5	...
<pre>Aug 8 09:59:22 PM ncwvj INFO:  uvicorn running on http://0.0.0.0:10000 (Press CTRL+C to quit) Aug 8 09:59:22 PM ncwvj INFO:  127.0.0.1:52730 - "HEAD / HTTP/1.1" 404 Not Found Aug 8 09:59:30 PM ncwvj INFO:  =&gt; Your service is live!  Aug 8 09:59:30 PM ncwvj INFO:  =&gt; Aug 8 09:59:31 PM ncwvj INFO:  =&gt; /////////////////////////////// Aug 8 09:59:31 PM ncwvj INFO:  =&gt; /////////////////////////////// Aug 8 09:59:31 PM ncwvj INFO:  =&gt; Available at your primary URL <a href="https://ensemble-api-qzpf.onrender.com">https://ensemble-api-qzpf.onrender.com</a> Aug 8 09:59:31 PM ncwvj INFO:  =&gt; Aug 8 09:59:31 PM ncwvj INFO:  =&gt; /////////////////////////////// Aug 8 10:00:31 PM qz8rs INFO:  Shutting down Aug 8 10:00:32 PM qz8rs INFO:  Waiting for application shutdown. Aug 8 10:00:32 PM qz8rs INFO:  Application shutdown complete. Aug 8 10:00:32 PM qz8rs INFO:  Finished server process [48] Aug 8 10:01:03 PM ncwvj INFO:  181.199.41.179:0 - "GET / HTTP/1.1" 404 Not Found Aug 8 10:04:28 PM ncwvj INFO:  =&gt; Detected service running on port 10000 Aug 8 10:04:29 PM ncwvj INFO:  =&gt; Docs on specifying a port: <a href="https://onrender.com/docs/web-services#port-binding">https://onrender.com/docs/web-services#port-binding</a></pre>				

## Api creada para llamar al servicio

<https://ensemble-api-qzpf.onrender.com>

## Ejemplo de las radiografías y las máscaras aplicadas

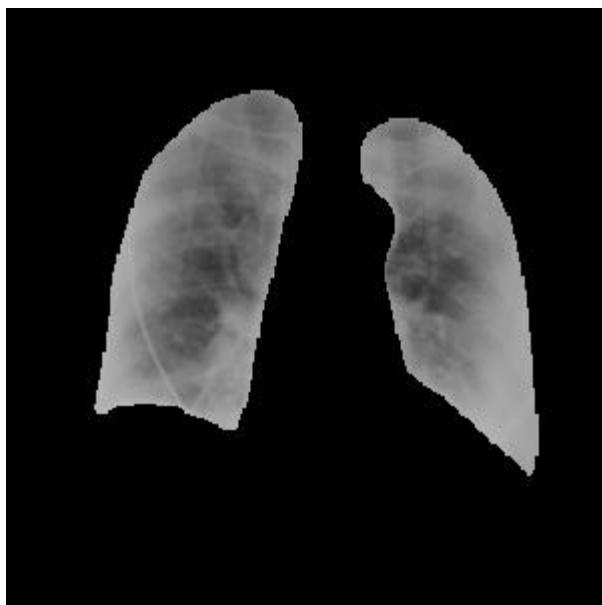
### Imagen original



### Mascara



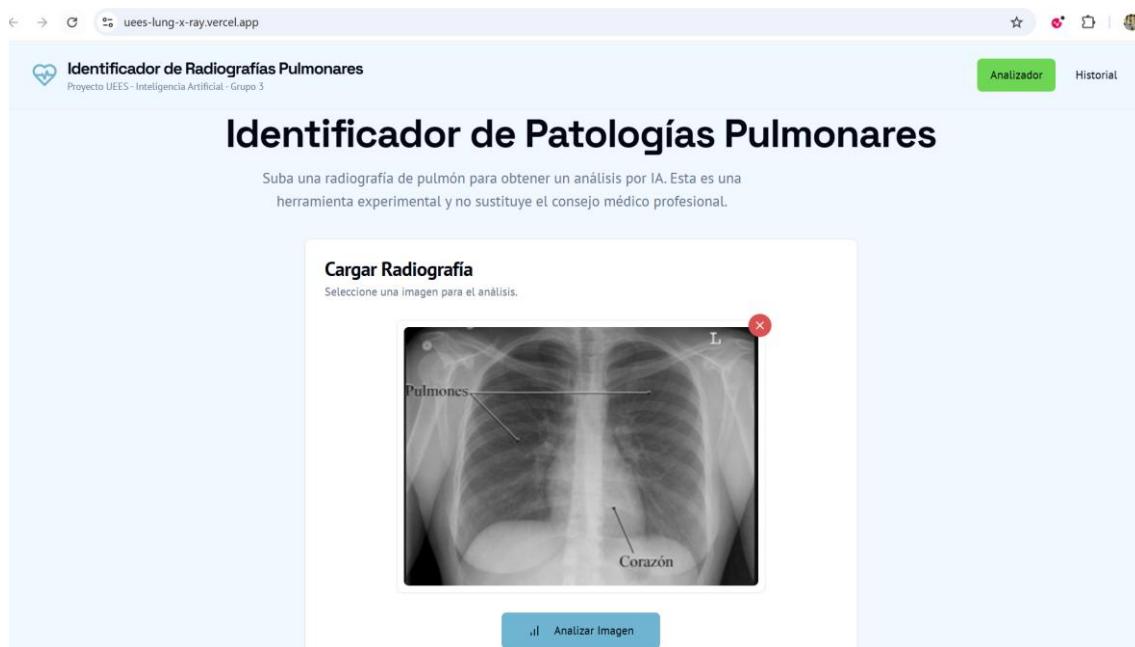
Procesamiento con la mascara previo al entrenamiento



**Nota:** De esta manera eliminamos el ruido de los huesos lo que hace que la predicción sea mucho más acertada

**Enlace de la App web funcional en donde se usa la API que contiene el uso de los modelos entrenados:**

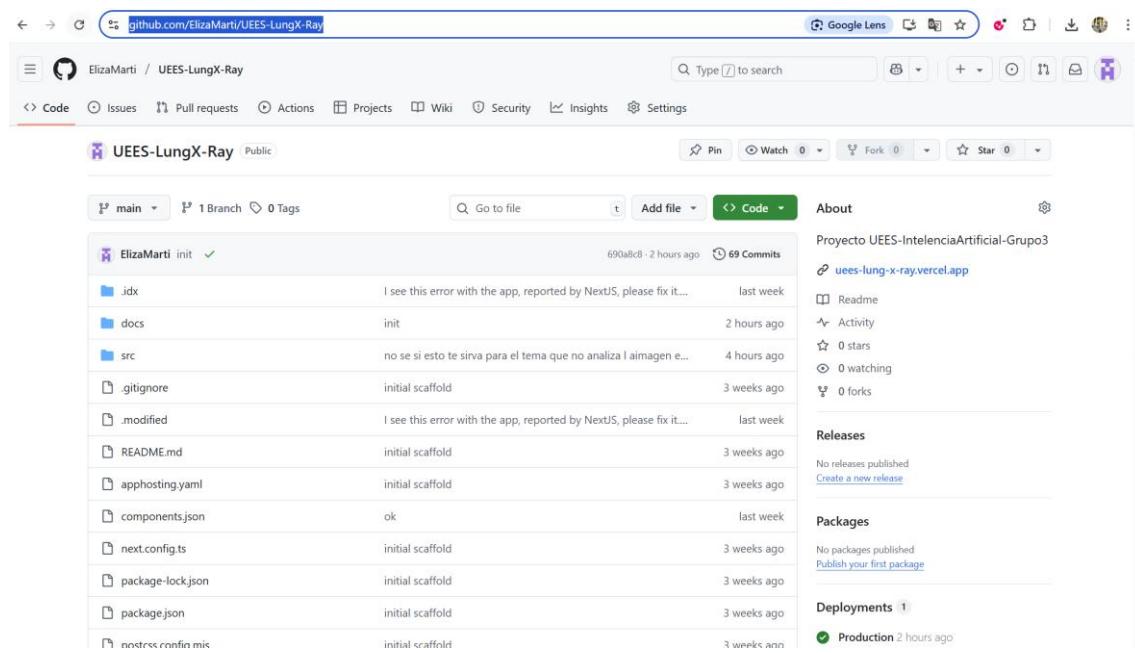
<https://uees-lung-x-ray.vercel.app/>





## GitHub del código de programación de la App web expuesta:

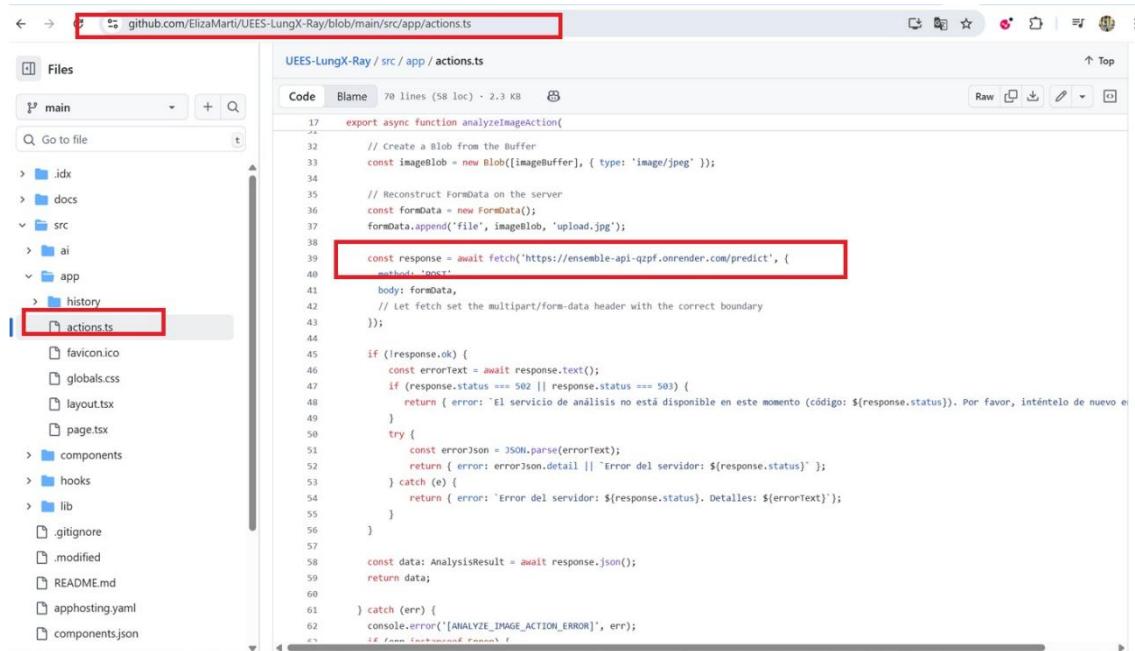
<https://github.com/ElizaMarti/UEES-LungX-Ray>



The GitHub repository page for 'ElizaMarti/UEES-LungX-Ray' shows the following details:

- Code**: The repository is public.
- Issues**: 1 issue.
- Pull requests**: 1 pull request.
- Actions**: 1 action.
- Projects**: 1 project.
- Wiki**: No wiki.
- Security**: No security issues.
- Insights**: Insights available.
- Settings**: Repository settings.
- ElizaMarti / UEES-LungX-Ray**: Repository name.
- Code**: Main branch, 1 branch, 0 tags.
- Commits**: 69 commits.
- Files** (Listed):
  - .idx: I see this error with the app, reported by NextJS, please fix it.... (last week)
  - .docs: init (2 hours ago)
  - .src: no se si esto te sirva para el tema que no analiza la imagen e... (4 hours ago)
  - .gitignore: initial scaffold (3 weeks ago)
  - .modified: I see this error with the app, reported by NextJS, please fix it.... (last week)
  - README.md: initial scaffold (3 weeks ago)
  - apphosting.yaml: initial scaffold (3 weeks ago)
  - components.json: ok (last week)
  - next.config.ts: initial scaffold (3 weeks ago)
  - package-lock.json: initial scaffold (3 weeks ago)
  - package.json: initial scaffold (3 weeks ago)
  - nestssrc confin mis: initial scaffold (3 weeks ago)
- About**: Projecto UEES-InteligenciaArtificial-Grupo3, ues-lung-x-ray.vercel.app.
- Readme**, **Activity**, **0 stars**, **0 watching**, **0 forks**.
- Releases**: No releases published, Create a new release.
- Packages**: No packages published, Publish your first package.
- Deployments**: 1 deployment, Production 2 hours ago.

## Uso de API con modelos entrenados:



```

  export async function analyzeImageAction() {
    // Create a Blob from the Buffer
    const imageBlob = new Blob([imageBuffer], { type: 'image/jpeg' });

    // Reconstruct FormData on the server
    const formData = new FormData();
    formData.append('file', imageBlob, 'upload.jpg');

    const response = await fetch('https://ensemble-api-qzpf.onrender.com/predict', {
      method: 'POST',
      body: formData,
      // Let fetch set the multipart/form-data header with the correct boundary
    });

    if (!response.ok) {
      const errorText = await response.text();
      if (response.status === 502 || response.status === 503) {
        return { error: 'El servicio de análisis no está disponible en este momento (código: ${response.status}). Por favor, inténtelo de nuevo en un momento posterior.' };
      }
      try {
        const errorJson = JSON.parse(errorText);
        return { error: errorJson.detail || `Error del servidor: ${response.status}` };
      } catch (e) {
        return { error: `Error del servidor: ${response.status}. Detalles: ${errorText}` };
      }
    }

    const data: AnalysisResult = await response.json();
    return data;
  }
}

```

## Anexos Códigos de Python para el entrenamiento

### 1.- Procesamiento de imágenes

```
import os
import cv2
import numpy as np
from tqdm import tqdm

# === CAMBIA ESTA RUTA A TU CARPETA ORIGINAL ===
root_dir = r"C:\Users\ASUS\OneDrive\Desktop\Pulmones"
# === CARPETA DONDE SE GUARDARÁN LAS IMÁGENES CON MÁSCARA
===
out_dir = r"C:\Users\ASUS\OneDrive\Desktop\Pulmones_Mascara"

# Crear carpeta de salida
os.makedirs(out_dir, exist_ok=True)

# Extensiones de imagen aceptadas
IMG_EXTS = {".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff"}

def find_matching_mask(masks_dir, stem):
    """Busca una máscara con el mismo nombre base y extensión de imagen válida."""
    for ext in IMG_EXTS:
        mask_candidate = os.path.join(masks_dir, stem + ext)
        if os.path.exists(mask_candidate):
            return mask_candidate
    return None

# Recorrer las clases (COVID, Lung_Opacity, Normal, Viral Pneumonia, etc.)
for clase in os.listdir(root_dir):
    clase_dir = os.path.join(root_dir, clase)
    if not os.path.isdir(clase_dir):
        continue

    images_dir = os.path.join(clase_dir, "images")
    masks_dir = os.path.join(clase_dir, "masks")

    if not (os.path.isdir(images_dir) and os.path.isdir(masks_dir)):
        print(f"⚠️ Omitiendo '{clase}' (faltan 'images' o 'masks').")
```

**continue**

```

# Carpeta de salida para esta clase
out_class = os.path.join(out_dir, clase)
os.makedirs(out_class, exist_ok=True)

# Lista de imágenes válidas
img_files = [f for f in os.listdir(images_dir)
             if os.path.splitext(f)[1].lower() in IMG_EXTS]

# Procesar imágenes
for fname in tqdm(img_files, desc=f"Procesando {clase}", unit="img"):
    img_path = os.path.join(images_dir, fname)
    stem, _ = os.path.splitext(fname)

    mask_path = find_matching_mask(masks_dir, stem)
    if mask_path is None:
        print(f" ✗ No se encontró máscara para {fname}")
        continue

    # Leer imagen y máscara
    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

    if img is None or mask is None:
        print(f" ✗ Error leyendo {fname}")
        continue

    # Binarizar máscara
    mask_bin = (mask > 0).astype(np.uint8)

    # Asegurar mismo tamaño
    if img.shape[:2] != mask.shape[:2]:
        mask_bin = cv2.resize(mask_bin, (img.shape[1], img.shape[0]),
                             interpolation=cv2.INTER_NEAREST)

    # Aplicar máscara
    masked = cv2.bitwise_and(img, img, mask=mask_bin)

    # Guardar imagen con máscara aplicada
    out_path = os.path.join(out_class, fname)
    cv2.imwrite(out_path, masked)

print(f" ✅ Proceso completado. Imágenes guardadas en: {out_dir}")

```

## 2.- Entrenamiento resnet50.py

```

import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from sklearn.metrics import classification_report, confusion_matrix, f1_score,
accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import torch.multiprocessing as mp

# ===== RUTAS (ajusta si hace falta) =====
DATA_DIR = r"C:\Users\ASUS\OneDrive\Desktop\Pulmones_Mascara"
DESKTOP_DIR = r"C:\Users\ASUS\OneDrive\Desktop"
CKPT_PATH = os.path.join(DESKTOP_DIR, "resnet50_best.pth")
HIST_ACC_PNG = os.path.join(DESKTOP_DIR, "resnet50_history_acc.png")
HIST_LOSS_PNG = os.path.join(DESKTOP_DIR, "resnet50_history_loss.png")
CM_PNG = os.path.join(DESKTOP_DIR, "resnet50_confusion_matrix.png")
REPORT_TXT = os.path.join(DESKTOP_DIR, "resnet50_metrics.txt")

# ===== CONFIG =====
batch_size = 64      # súbelo si tienes VRAM libre
img_size = 224
num_classes = 4
epochs = 30
patience = 3          # early stopping
num_workers = 4        # si da problemas, ponlo en 0

class EarlyStopping:
    def __init__(self, patience=3, mode="max", delta=0.0):
        self.patience = patience
        self.mode = mode
        self.delta = delta
        self.best = None
        self.counter = 0
        self.should_stop = False
    def step(self, metric):
        if self.best is None:
            self.best = metric
            return False
        improved = (metric > self.best + self.delta) if self.mode == "max" else (metric

```

```

< self.best - self.delta)
if improved:
    self.best = metric
    self.counter = 0
else:
    self.counter += 1
    if self.counter >= self.patience:
        self.should_stop = True
return self.should_stop

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("CUDA disponible:", torch.cuda.is_available())
    print("GPU:", torch.cuda.get_device_name(0) if torch.cuda.is_available() else
"Ninguna")

# ----- Transforms -----
transform_train = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])
transform_val = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

# ----- Dataset & split 80/20 -----
full_ds = datasets.ImageFolder(root=DATA_DIR, transform=transform_train)
class_names = full_ds.classes
train_size = int(0.8 * len(full_ds))
val_size = len(full_ds) - train_size
train_ds, val_ds = random_split(full_ds, [train_size, val_size])
val_ds.dataset.transform = transform_val # sin augmentations en val

# DataLoaders (multiproceso en Windows con spawn)
train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True,
                         num_workers=num_workers, pin_memory=True)
val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False,
                       num_workers=num_workers, pin_memory=True)

```

```

# ----- Modelo -----
model =
models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
for p in model.parameters():
    p.requires_grad = False # congelamos base
model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 512),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(512, num_classes)
)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode="max",
factor=0.5, patience=1)

early = EarlyStopping(patience=patience, mode="max")

# ----- Entrenamiento -----
best_acc = 0.0
train_acc_hist, val_acc_hist = [], []
train_loss_hist, val_loss_hist = [], []

for epoch in range(1, epochs + 1):
    # Train
    model.train()
    train_loss, train_corrects, n_train = 0.0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device,
non_blocking=True)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        preds = outputs.argmax(1)
        train_loss += loss.item() * inputs.size(0)
        train_corrects += (preds == labels).sum().item()
        n_train += inputs.size(0)

    epoch_train_loss = train_loss / n_train

```

```

epoch_train_acc = train_corrects / n_train

# Val
model.eval()
val_loss, val_corrects, n_val = 0.0, 0, 0
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device,
non_blocking=True)
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        preds = outputs.argmax(1)
        val_loss += loss.item() * inputs.size(0)
        val_corrects += (preds == labels).sum().item()
        n_val += inputs.size(0)

epoch_val_loss = val_loss / n_val
epoch_val_acc = val_corrects / n_val

# Historial para gráficas
train_acc_hist.append(epoch_train_acc)
val_acc_hist.append(epoch_val_acc)
train_loss_hist.append(epoch_train_loss)
val_loss_hist.append(epoch_val_loss)

scheduler.step(epoch_val_acc)

print(f"Epoch {epoch:02d}/{epochs} | "
      f"Train: loss {epoch_train_loss:.4f}, acc {epoch_train_acc:.4f} | "
      f"Val: loss {epoch_val_loss:.4f}, acc {epoch_val_acc:.4f}")

# Guardar mejor checkpoint
if epoch_val_acc > best_acc:
    best_acc = epoch_val_acc
    torch.save(model.state_dict(), CKPT_PATH)
    print(f" ✅ Mejor modelo guardado (val_acc={best_acc:.4f}) → "
          f"{CKPT_PATH}")

# Early stopping
if early.step(epoch_val_acc):
    print(f" ⚠️ Early stopping activado (sin mejora en {patience} epochs).")
    break

print("\nMejor val_acc lograda: {best_acc:.4f}")

```

```

# ----- Evaluación final con el mejor ckpt -----
model.load_state_dict(torch.load(CKPT_PATH, map_location=device))
model.eval()

all_true, all_pred = [], []
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device, non_blocking=True)
        outputs = model(inputs)
        preds = outputs.argmax(1).cpu().numpy()
        all_pred.extend(preds)
        all_true.extend(labels.numpy())

acc = accuracy_score(all_true, all_pred)
f1_macro = f1_score(all_true, all_pred, average='macro')
f1_weighted = f1_score(all_true, all_pred, average='weighted')
report = classification_report(all_true, all_pred, target_names=class_names,
digits=4)
cm = confusion_matrix(all_true, all_pred)

# Guardar métricas
with open(REPORT_TXT, "w", encoding="utf-8") as f:
    f.write("== Métricas finales (Validación) ==\n")
    f.write(f"Accuracy: {acc:.6f}\n")
    f.write(f"F1 (macro): {f1_macro:.6f}\n")
    f.write(f"F1 (weighted): {f1_weighted:.6f}\n\n")
    f.write("== Reporte de clasificación ==\n")
    f.write(report + "\n\n")
    f.write("== Matriz de confusión ==\n")
    f.write(np.array2string(cm))

print(f"Reporte guardado en: {REPORT_TXT}")

# Gráficas
plt.figure()
plt.plot(train_acc_hist, label="Train Acc")
plt.plot(val_acc_hist, label="Val Acc")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("Accuracy")
plt.tight_layout(); plt.savefig(HIST_ACC_PNG, dpi=200); plt.close()

plt.figure()
plt.plot(train_loss_hist, label="Train Loss")
plt.plot(val_loss_hist, label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss")

```

```

plt.tight_layout(); plt.savefig(HIST_LOSS_PNG, dpi=200);
plt.close()

plt.figure(figsize=(6,5))
plt.imshow(cm, interpolation='nearest')
plt.title('Matriz de Confusión'); plt.colorbar()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names, rotation=45, ha='right')
plt.yticks(ticks, class_names)
plt.xlabel('Predicción'); plt.ylabel('Verdadero')
plt.tight_layout(); plt.savefig(CM_PNG, dpi=200); plt.close()

print(f"▣ Gráficas guardadas: {HIST_ACC_PNG}, {HIST_LOSS_PNG},
{CM_PNG}")
print(f"☒ Mejor modelo (.pth) en: {CKPT_PATH}")

if __name__ == "__main__":
    # Arranque correcto de multiproceso en Windows
    mp.set_start_method("spawn", force=True)
    main()

```

### 3.-train\_efficientnet

```

import os, torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from sklearn.metrics import classification_report, confusion_matrix, f1_score,
accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import torch.multiprocessing as mp

# ===== RUTAS =====
DATA_DIR = r"C:\Users\ASUS\OneDrive\Desktop\Pulmones_Mascara"
DESKTOP_DIR = r"C:\Users\ASUS\OneDrive\Desktop"
CKPT_PATH = os.path.join(DESKTOP_DIR, "efficientnet_b0_best.pth")
REPORT_TXT = os.path.join(DESKTOP_DIR, "efficientnet_b0_metrics.txt")
HIST_ACC_PNG = os.path.join(DESKTOP_DIR,
"efficientnet_b0_history_acc.png")
HIST_LOSS_PNG = os.path.join(DESKTOP_DIR,
"efficientnet_b0_history_loss.png")
CM_PNG = os.path.join(DESKTOP_DIR,
"efficientnet_b0_confusion_matrix.png")

```

```

# ===== CONFIG =====
IMG_SIZE=224; BATCH_SIZE=64; EPOCHS=25; PATIENCE=5;
NUM_WORKERS=4; SEED=42

class EarlyStopping:
    def __init__(self, patience=5, mode="max", delta=1e-4):
        self.patience, self.mode, self.delta = patience, mode, delta
        self.best, self.counter, self.should_stop = None, 0, False
    def step(self, metric):
        if self.best is None: self.best = metric; return False
        improved = (metric > self.best + self.delta) if self.mode=="max" else (metric < self.best - self.delta)
        if improved: self.best = metric; self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience: self.should_stop = True
        return self.should_stop

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("CUDA:", torch.cuda.is_available(), "| GPU:",
torch.cuda.get_device_name(0) if torch.cuda.is_available() else "CPU")

# ---- Transforms ----
tf_train = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])
tf_val = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

# ---- Data ----
torch.manual_seed(SEED)
full = datasets.ImageFolder(DATA_DIR, transform=tf_train)
class_names = full.classes
n_train = int(0.8*len(full)); n_val = len(full)-n_train
train_ds, val_ds = random_split(full, [n_train, n_val],
generator=torch.Generator().manual_seed(SEED))
val_ds.dataset.transform = tf_val

```

```

train_ld = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=NUM_WORKERS, pin_memory=True)
val_ld = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=True)

# ---- Modelo: EfficientNet-B0 ----
model =
models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.IMAGENET1K_V1)
in_f = model.classifier[1].in_features
model.classifier = nn.Sequential(nn.Dropout(p=0.2, inplace=True),
nn.Linear(in_f, len(class_names)))
for p in model.parameters(): p.requires_grad = False
for p in model.classifier.parameters(): p.requires_grad = True
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),
lr=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode="max", factor=0.5, patience=1, verbose=False)
early = EarlyStopping(patience=PATIENCE, mode="max", delta=1e-4)

# Historial para gráficas
train_acc_hist, val_acc_hist = [], []
train_loss_hist, val_loss_hist = [], []

best_acc = 0.0
for epoch in range(1, EPOCHS+1):
    # ---- Train ----
    model.train(); tr_loss=tr_corr=tr_n=0
    for x,y in train_ld:
        x,y = x.to(device,non_blocking=True), y.to(device,non_blocking=True)
        optimizer.zero_grad()
        out = model(x); loss = criterion(out,y)
        loss.backward(); optimizer.step()
        tr_loss += loss.item()*x.size(0)
        tr_corr += (out.argmax(1)==y).sum().item()
        tr_n += x.size(0)
    tr_acc = tr_corr/tr_n; tr_loss /= tr_n

    # ---- Val ----
    model.eval(); va_loss=va_corr=va_n=0
    with torch.no_grad():

```

```

        for x,y in val_ld:
            x,y = x.to(device,non_blocking=True), y.to(device,non_blocking=True)
            out = model(x); loss = criterion(out,y)
            va_loss += loss.item()*x.size(0)
            va_corr += (out.argmax(1)==y).sum().item()
            va_n += x.size(0)
        va_acc = va_corr/va_n; va_loss /= va_n

# Guardar historial y scheduler
train_acc_hist.append(tr_acc); val_acc_hist.append(va_acc)
train_loss_hist.append(tr_loss); val_loss_hist.append(va_loss)
scheduler.step(va_acc)

print(f"EfficientNet-B0 | Epoch {epoch:02d}/{EPOCHS} | "
      f"Train acc {tr_acc:.4f} loss {tr_loss:.4f} | "
      f"Val acc {va_acc:.4f} loss {va_loss:.4f}")

if va_acc > best_acc:
    best_acc = va_acc
    torch.save(model.state_dict(), CKPT_PATH)
    print(f" ✅ Mejor EfficientNet guardado → {CKPT_PATH}")
(val_acc={best_acc:.4f})")
if early.step(va_acc):
    print(" ⚠ Early stopping"); break

# ---- Evaluación final con el mejor checkpoint ----
model.load_state_dict(torch.load(CKPT_PATH, map_location=device))
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for x,y in val_ld:
        x = x.to(device,non_blocking=True)
        probs = torch.softmax(model(x), dim=1).cpu().numpy()
        y_pred.extend(probs.argmax(1)); y_true.extend(y.numpy())

acc = accuracy_score(y_true, y_pred)
f1m = f1_score(y_true, y_pred, average="macro")
f1w = f1_score(y_true, y_pred, average="weighted")
rep = classification_report(y_true, y_pred, target_names=class_names, digits=4)
cm = confusion_matrix(y_true, y_pred)

# ---- Guardar métricas (TXT) ----
with open(REPORT_TXT, "w", encoding="utf-8") as f:
    f.write(f"== EfficientNet-B0 Métricas (Val) ==\n")
    f.write(f"Accuracy: {acc:.6f}\nF1(macro): {f1m:.6f}\nF1(weighted): "

```

```

{f1w:.6f}\n\n")
f.write(rep + "\nMatriz de confusión:\n" + np.array2string(cm))
print("☒ Métricas EfficientNet →", REPORT_TXT)

# ---- Gráficas: history acc/loss ----
plt.figure()
plt.plot(train_acc_hist, label="Train Acc")
plt.plot(val_acc_hist, label="Val Acc")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("Accuracy")
plt.tight_layout(); plt.savefig(HIST_ACC_PNG, dpi=200); plt.close()

plt.figure()
plt.plot(train_loss_hist, label="Train Loss")
plt.plot(val_loss_hist, label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss")
plt.tight_layout(); plt.savefig(HIST_LOSS_PNG, dpi=200); plt.close()

# ---- Matriz de confusión (PNG) ----
plt.figure(figsize=(6,5))
plt.imshow(cm, interpolation='nearest')
plt.title('Matriz de Confusión - EfficientNet-B0'); plt.colorbar()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names, rotation=45, ha='right')
plt.yticks(ticks, class_names)
plt.xlabel('Predicción'); plt.ylabel('Verdadero')
plt.tight_layout(); plt.savefig(CM_PNG, dpi=200); plt.close()

print(f"☒ Guardados: {HIST_ACC_PNG}, {HIST_LOSS_PNG}, {CM_PNG}")
print(f"☒ Checkpoint (.pth): {CKPT_PATH}")

if __name__ == "__main__":
    mp.set_start_method("spawn", force=True)
    main()

```

#### 4.- Train\_densenett121.y

```

import os, torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models
from sklearn.metrics import classification_report, confusion_matrix, f1_score,
accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import torch.multiprocessing as mp

```

```

# ===== RUTAS =====
DATA_DIR = r"C:\Users\ASUS\OneDrive\Desktop\Pulmones_Mascara"
DESKTOP_DIR = r"C:\Users\ASUS\OneDrive\Desktop"
CKPT_PATH = os.path.join(DESKTOP_DIR, "densenet121_best.pth")
REPORT_TXT = os.path.join(DESKTOP_DIR, "densenet121_metrics.txt")
HIST_ACC_PNG = os.path.join(DESKTOP_DIR,
                            "densenet121_history_acc.png")
HIST_LOSS_PNG = os.path.join(DESKTOP_DIR,
                            "densenet121_history_loss.png")
CM_PNG = os.path.join(DESKTOP_DIR,
                      "densenet121_confusion_matrix.png")

# ===== CONFIG =====
IMG_SIZE=224; BATCH_SIZE=64; EPOCHS=25; PATIENCE=5;
NUM_WORKERS=4; SEED=42

class EarlyStopping:
    def __init__(self, patience=5, mode="max", delta=1e-4):
        self.patience, self.mode, self.delta = patience, mode, delta
        self.best, self.counter, self.should_stop = None, 0, False
    def step(self, metric):
        if self.best is None: self.best = metric; return False
        improved = (metric > self.best + self.delta) if self.mode=="max" else (metric < self.best - self.delta)
        if improved: self.best = metric; self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience: self.should_stop = True
        return self.should_stop

def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("CUDA:", torch.cuda.is_available(), "| GPU:",
          torch.cuda.get_device_name(0) if torch.cuda.is_available() else "CPU")

# ---- Transforms ----
tf_train = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
tf_val = transforms.Compose([

```

```

        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
    )

# ---- Data ----
torch.manual_seed(SEED)
full = datasets.ImageFolder(DATA_DIR, transform=tf_train)
class_names = full.classes
n_train = int(0.8*len(full)); n_val = len(full)-n_train
train_ds, val_ds = random_split(full, [n_train, n_val],
generator=torch.Generator().manual_seed(SEED))
val_ds.dataset.transform = tf_val

train_ld = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=NUM_WORKERS, pin_memory=True)
val_ld = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS, pin_memory=True)

# ---- Modelo: DenseNet121 ----
model =
models.densenet121(weights=models.DenseNet121_Weights.IMGNET1K_V1)
in_f = model.classifier.in_features
model.classifier = nn.Sequential(nn.Linear(in_f, 512), nn.ReLU(True),
nn.Dropout(0.5), nn.Linear(512, len(class_names)))
for p in model.parameters(): p.requires_grad = False
for p in model.classifier.parameters(): p.requires_grad = True
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),
lr=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode="max", factor=0.5, patience=1, verbose=False)
early = EarlyStopping(patience=PATIENCE, mode="max", delta=1e-4)

# Historial
train_acc_hist, val_acc_hist = [], []
train_loss_hist, val_loss_hist = [], []

best_acc = 0.0
for epoch in range(1, EPOCHS+1):
    # ---- Train ----
    model.train(); tr_loss=tr_corr=tr_n=0
    for x,y in train_ld:

```

```

x,y = x.to(device,non_blocking=True),
y.to(device,non_blocking=True)
optimizer.zero_grad()
out = model(x); loss = criterion(out,y)
loss.backward(); optimizer.step()
tr_loss += loss.item()*x.size(0)
tr_corr += (out.argmax(1)==y).sum().item()
tr_n += x.size(0)
tr_acc = tr_corr/tr_n; tr_loss /= tr_n

# ---- Val ----
model.eval(); va_loss=va_corr=va_n=0
with torch.no_grad():
    for x,y in val_ld:
        x,y = x.to(device,non_blocking=True), y.to(device,non_blocking=True)
        out = model(x); loss = criterion(out,y)
        va_loss += loss.item()*x.size(0)
        va_corr += (out.argmax(1)==y).sum().item()
        va_n += x.size(0)
    va_acc = va_corr/va_n; va_loss /= va_n

# Historial + LR scheduler
train_acc_hist.append(tr_acc); val_acc_hist.append(va_acc)
train_loss_hist.append(tr_loss); val_loss_hist.append(va_loss)
scheduler.step(va_acc)

print(f'DenseNet121 | Epoch {epoch:02d}/{EPOCHS} | '
      f'Train acc {tr_acc:.4f} loss {tr_loss:.4f} | '
      f'Val acc {va_acc:.4f} loss {va_loss:.4f}')

if va_acc > best_acc:
    best_acc = va_acc
    torch.save(model.state_dict(), CKPT_PATH)
    print(f" ✅ Mejor DenseNet guardado → {CKPT_PATH}")
(val_acc={best_acc:.4f})")
    if early.step(va_acc):
        print(" ⚡ Early stopping"); break

# ---- Evaluación final con el mejor checkpoint ----
model.load_state_dict(torch.load(CKPT_PATH, map_location=device))
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for x,y in val_ld:
        x = x.to(device,non_blocking=True)

```

```

probs = torch.softmax(model(x), dim=1).cpu().numpy()
y_pred.extend(probs.argmax(1)); y_true.extend(y.numpy())

acc = accuracy_score(y_true, y_pred)
f1m = f1_score(y_true, y_pred, average="macro")
f1w = f1_score(y_true, y_pred, average="weighted")
rep = classification_report(y_true, y_pred, target_names=class_names, digits=4)
cm = confusion_matrix(y_true, y_pred)

# ---- Guardar métricas (TXT) ----
with open(REPORT_TXT, "w", encoding="utf-8") as f:
    f.write(f"== DenseNet121 Métricas (Val) ==\n")
    f.write(f"Accuracy: {acc:.6f}\nF1(macro): {f1m:.6f}\nF1(weighted): {f1w:.6f}\n\n")
    f.write(rep + "\nMatriz de confusión:\n" + np.array2string(cm))
print("📝 Métricas DenseNet →", REPORT_TXT)

# ---- Gráficas: history acc/loss ----
plt.figure()
plt.plot(train_acc_hist, label="Train Acc")
plt.plot(val_acc_hist, label="Val Acc")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("Accuracy")
plt.tight_layout(); plt.savefig(HIST_ACC_PNG, dpi=200); plt.close()

plt.figure()
plt.plot(train_loss_hist, label="Train Loss")
plt.plot(val_loss_hist, label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss")
plt.tight_layout(); plt.savefig(HIST_LOSS_PNG, dpi=200); plt.close()

# ---- Matriz de confusión (PNG) ----
plt.figure(figsize=(6,5))
plt.imshow(cm, interpolation='nearest')
plt.title('Matriz de Confusión - DenseNet121'); plt.colorbar()
ticks = np.arange(len(class_names))
plt.xticks(ticks, class_names, rotation=45, ha='right')
plt.yticks(ticks, class_names)
plt.xlabel('Predicción'); plt.ylabel('Verdadero')
plt.tight_layout(); plt.savefig(CM_PNG, dpi=200); plt.close()

print(f"📝 Guardados: {HIST_ACC_PNG}, {HIST_LOSS_PNG}, {CM_PNG}")
print(f"☑ Checkpoint (.pth): {CKPT_PATH}")

if __name__ == "__main__":

```

```
mp.set_start_method("spawn", force=True)
main()
```

## Nota final.-

### Anexo: Exclusión de LIME y otros paquetes adicionales

Durante el proceso de desarrollo inicial se evaluó la incorporación de bibliotecas de interpretabilidad de modelos como LIME (Local Interpretable Model-agnostic Explanations), así como otras dependencias asociadas para análisis explicativo y visualización de resultados. Sin embargo, tras realizar pruebas de rendimiento y consumo de recursos en el entorno de despliegue, se determinó que el uso de dichas librerías no era óptimo para esta arquitectura, por las siguientes razones:

1. Sobrecarga computacional: LIME genera explicaciones locales para instancias individuales mediante la creación de múltiples perturbaciones en los datos de entrada y la ejecución repetitiva del modelo para cada perturbación. Esto implica un coste computacional elevado, especialmente en modelos de visión por computadora con redes convolucionales profundas como ResNet50, DenseNet121 y EfficientNet-B0, que ya requieren tiempos de inferencia considerables en entornos sin aceleración GPU.
2. Latencia en entorno de producción: Las pruebas iniciales mostraron que el uso de LIME aumentaba los tiempos de respuesta de la API de varios cientos de milisegundos a varios segundos, dependiendo del tamaño de la imagen y de la complejidad de la arquitectura utilizada. Esto compromete el rendimiento en escenarios multiusuario y afecta negativamente a la escalabilidad en servidores con limitaciones de CPU y memoria.
3. Requerimientos de dependencias adicionales: LIME y librerías relacionadas (matplotlib, scikit-learn, etc.) incrementaban el peso total del entorno de ejecución y, por ende, el tiempo de despliegue y arranque en plataformas como Render. Además, algunas dependencias tenían incompatibilidades potenciales con las versiones optimizadas de PyTorch y FastAPI utilizadas.
4. No alineado con los objetivos funcionales de la versión en producción: El propósito principal de esta versión de la API es servir predicciones rápidas y fiables. La interpretabilidad avanzada, si bien es útil en entornos de

investigación y auditoría, no constituía un requerimiento funcional prioritario para los casos de uso previstos en producción.

Por estas razones, se decidió omitir LIME y otros paquetes no esenciales en esta etapa del proyecto, manteniendo un enfoque minimalista y orientado al rendimiento. Esto permitió optimizar el consumo de recursos, reducir la complejidad del mantenimiento del código y garantizar que la API pueda manejar solicitudes concurrentes de forma más eficiente en el entorno de despliegue actual.