```
import os
import sys
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import cv2

# ---------------------- 1. Setup Paths ----------------------
print("Running ovarian2.py from", os.getcwd())
train_dir = "/gpfs/u/home/DLO2/DLO2htsm/scratch/archive/Train_Images"
val_dir = "/gpfs/u/home/DLO2/DLO2htsm/scratch/archive/Test_Images"
out_dir = "/gpfs/u/home/DLO2/DLO2htsm/scratch/archive/misclassified"
os.makedirs(out_dir, exist_ok=True)

# ---------------------- 2. Device ----------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# ---------------------- 3. Data Transforms ----------------------
train_transform = transforms.Compose([
  transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
  transforms.RandomHorizontalFlip(),
  transforms.RandomVerticalFlip(),
  transforms.RandomRotation(30),
  transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1)),
  transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
  transforms.ToTensor(),
  transforms.RandomErasing(p=0.25),
  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
# ---------------------- 4. Load Data ----------------------
train_dataset = datasets.ImageFolder(train_dir, transform=train_transform)
val_dataset = datasets.ImageFolder(val_dir, transform=val_transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
class_names = train_dataset.classes
print(f"Classes: {class_names}", flush=True)

# ---------------------- 5. Model Setup ----------------------
print(">>> Reached model setup", flush=True)
model = models.resnet50(pretrained=False)
model.load_state_dict(torch.load("resnet50_state_dict_only.pth", map_location=device))
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model = model.to(device)
for name, param in model.named_parameters():
    param.requires_grad = "layer4" in name or "fc" in name

# ---------------------- 6. Training Setup ----------------------
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.0001)

# ---------------------- 7. Training Loop ----------------------
def train_model(model, train_loader, val_loader, criterion, optimizer, epochs=25):
    best_acc = 0.0
    best_model_wts = model.state_dict()
    for epoch in range(epochs):
        print(f"Epoch {epoch+1} started...", flush=True)
        model.train()
        running_loss, correct, total = 0.0, 0, 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_acc = correct / total
```

```
        print(f"Epoch {epoch+1}: Train Acc = {train_acc:.4f}")

        # Validation
        model.eval()
        val_correct, val_total = 0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs, 1)
                val_total += labels.size(0)
                val_correct += (predicted == labels).sum().item()
        val_acc = val_correct / val_total
        print(f"Epoch {epoch+1}: Val Acc = {val_acc:.4f}")

        if val_acc > best_acc:
            best_acc = val_acc
            best_model_wts = model.state_dict()
            torch.save(best_model_wts, "best_model.pth")

    model.load_state_dict(best_model_wts)
    return model

# ---------------------- 8. Start Training ----------------------
print("Starting training...", flush=True)
model = train_model(model, train_loader, val_loader, criterion, optimizer, epochs=25)
```