5 Fold Cross Validation

```
import pandas as pd
import os

csv_path = "/content/drive/MyDrive/ML Project/labels.csv"

if os.path.exists(csv_path):
    print(f"CSV file found at: {csv_path}")
    try:
        master_df_debug = pd.read_csv(csv_path)
        print("\nFirst 5 rows of the CSV:")
        display(master_df_debug.head())
        print("\nColumns in the CSV:")
        print(master_df_debug.columns.tolist())
    except Exception as e:
        print(f"Error reading CSV: {e}")
else:
    print(f"ERROR: CSV file not found at {csv_path}. Please verify the path.")
```

```
CSV file found at: /content/drive/MyDrive/ML Project/labels.csv

First 5 rows of the CSV:
```

	image_path	class	split	class_name	case_id
0	Case 001/AFC0.jpg	0	val	Normal	Case 001
1	Case 001/AFC1.jpg	0	val	Normal	Case 001
2	Case 002/AJL0.jpg	0	test	Normal	Case 002
3	Case 002/AJL1.jpg	0	test	Normal	Case 002
4	Case 003/AGY0.jpg	0	train	Normal	Case 003

```
Columns in the CSV:
['image_path', 'class', 'split', 'class_name', 'case_id']
```

```
import os
import json
import math
import random
import time
import re
import pathlib
import numpy as np
import pandas as pd
from PIL import Image
from pathlib import Path
from dataclasses import dataclass
import cv2

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import torchvision.transforms as T
import timm

from sklearn.metrics import f1_score, accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import GroupKFold

# 1. Setup & Config
# -----------------
@dataclass
class CFG:
    # Path to the NEW split dataset you just created
    # Note: Adjust these paths if not running in Colab or if paths differ
    TRAIN_DIR = r"/content/via_dataset_fixed/train"
    TEST_DIR =  r"/content/via_dataset_fixed/test"

    # Path to your original labels CSV (needed to map filename -> class)
    CSV_PATH  = "/content/drive/MyDrive/ML Project/labels.csv"

    NUM_CLASSES: int = 3
    IMAGE_SIZE: int = 384
    BATCH_SIZE: int = 32
    EPOCHS: int = 30
    LR: float = 1e-4
    WD: float = 1e-2
    SEED: int = 42
    MODEL_NAME: str = "efficientnet_b0"
    LOSS: str = "cross_entropy"
    OUT_DIR: str = "runs"

cfg = CFG()

def set_seed(seed=42):
    random.seed(seed); np.random.seed(seed)
    torch.manual_seed(seed); torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
```

```python
    set_seed(cfg.SEED)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # 2. Data Preparation Logic (Modified for CV)
    # ---------------------------------------
    def prepare_dataframes():
        """
        Scans the new folders, cleans filenames to find original keys,
        and maps them to labels from the master CSV.
        """
        # 1. Load Master Labels
        if not os.path.exists(cfg.CSV_PATH):
            print(f"WARNING: CSV path {cfg.CSV_PATH} does not exist. Please check configuration.")
            # For the purpose of providing the code, we proceed, but runtime will fail if paths are wrong.

        try:
            master_df = pd.read_csv(cfg.CSV_PATH)

            # Clean up column names/values if necessary
            # Assumes CSV has columns: 'File' (or image_path) and 'class'
            # We create a dictionary: { '1AFC1.jpg': 'Precancer', ... }

            # Normalize the filename column in CSV to be just the basename
            if 'image_path' in master_df.columns:
                master_df['filename_key'] = master_df['image_path'].apply(lambda x: os.path.basename(str(x)))
            elif 'File' in master_df.columns:
                master_df['filename_key'] = master_df['File'].apply(lambda x: os.path.basename(str(x)))
            else:
                raise ValueError("CSV must have 'image_path' or 'File' column")

            label_map = dict(zip(master_df['filename_key'], master_df['class']))
            print(f"Debug: Sample keys from label_map: {list(label_map.keys())[:5]}")
        except Exception as e:
            print(f"Error loading CSV: {e}")
            label_map = {}

        # 2. Helper to scan a folder and build a DF
        def scan_folder(folder_path):
            data = []
            if not os.path.exists(folder_path):
                 print(f"Warning: Folder {folder_path} not found.")
                 return pd.DataFrame(columns=["path", "label", "patient_id", "is_aug"])

            # DEBUG: Print contents of the folder
            folder_contents = os.listdir(folder_path)
            print(f"Debug: Contents of {folder_path}: {folder_contents[:5]}... ({len(folder_contents)} items)")

            files = [f for f in folder_contents if f.lower().endswith('.jpg')]
            print(f"Debug: Found {len(files)} JPG files in {folder_path}.")

            for i, f in enumerate(files):
                # Logic: 'IARC_image_bank_1AFC1_aug_rot.jpg' -> Original Key: '1AFC1.jpg'
                # Remove augmentation suffixes to find the label key
                base_name = f

                clean_name = Path(f).stem
                # Remove augmentation suffixes like _rot, _blur, _bright, etc.
                clean_name = re.sub(r'_(rot|blur|bright|dark|trans|shear|radial|h_flip|aug_\d+).*', '', clean_name)

                # NEW: Also remove the 'IARC_image_bank_' prefix if it exists
                if clean_name.startswith('IARC_image_bank_'):
                    clean_name = clean_name.replace('IARC_image_bank_', '')

                original_key = f"{clean_name}.jpg"

                # Parse Patient ID for splitting (e.g., 1AFC from 1AFC1.jpg)
                match = re.search(r"([0-9]+[A-Z]+|[A-Z]+[0-9]+)", clean_name)
                patient_id = match.group(0) if match else "Unknown"

                if original_key in label_map:
                    label = label_map[original_key]
                    data.append({
                        "path": os.path.join(folder_path, f),
                        "label": label,
                        "patient_id": patient_id, # critical for GroupSplit
                        "is_aug": (f != original_key)
                    })
                # Add some debug to see what keys are being generated and not found
                if i < 5: # Print for first few files only
                    print(f"  Debug: Image file: {f}, Derived original_key: {original_key}, Found in label_map: {original_key in label_map}")

            return pd.DataFrame(data)

        print("Scanning Training Directory...")
        train_full_df = scan_folder(cfg.TRAIN_DIR)
        print(f"Found {len(train_full_df)} training images (including augmentations).")

        print("Scanning Test Directory...")
        test_df = scan_folder(cfg.TEST_DIR)
        print(f"Found {len(test_df)} test images.")

        # 3. Map String Labels to Int
        mapping = {"Normal": 0, "Precancer": 1, "Cancer": 2}
    # Handle cases where CSV might already have numbers or different strings
    def map_label(x):
        if isinstance(x, int): return x
        return mapping.get(x, -1) # Returns -1 if not found

    train_full_df['class_num'] = train_full_df['label'].apply(map_label)
    test_df['class_num'] = test_df['label'].apply(map_label)

    # Filter out bad labels
    train_full_df = train_full_df[train_full_df['class_num'] != -1]
    test_df = test_df[test_df['class_num'] != -1]

    # No single split here anymore. We return the full training DF.
    print("-" * 30)
    print(f"Total Train Size: {len(train_full_df)} (for CV)")
    print(f"Final Test Size:  {len(test_df)} (Original only)")
    print("-" * 30)

    return train_full_df, test_df

# 3. Dataset Class
# ----------------
clip_limit = 13
class CervicalDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]
        img_path = row['path']
        label = row['class_num']

        try:
            img = Image.open(img_path).convert("RGB")
        except Exception as e:
            print(f"Error reading image {img_path}: {e}")
            # Return black image or handle error
            img = Image.new("RGB", (cfg.IMAGE_SIZE, cfg.IMAGE_SIZE))

        if self.transform:
            img = self.transform(img)

        return img, torch.tensor(label, dtype=torch.long)

# 4. Transforms
# Define a custom CLAHE transform class
class ApplyCLAHE(object):
    def __init__(self, clip_limit=clip_limit, tile_grid_size=(8, 8)):
        self.clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_grid_size)

    def __call__(self, img):
        # Convert PIL to Numpy
        img_np = np.array(img)

        # Convert RGB to LAB color space (L = Lightness)
        lab = cv2.cvtColor(img_np, cv2.COLOR_RGB2LAB)
        l, a, b = cv2.split(lab)

        # Apply CLAHE to the L-channel only (contrast)
        cl = self.clahe.apply(l)

        # Merge back
        limg = cv2.merge((cl, a, b))

        # Convert back to RGB
        final = cv2.cvtColor(limg, cv2.COLOR_LAB2RGB)
        return Image.fromarray(final)

def get_transforms(image_size):
    train_tfms = T.Compose([
        ApplyCLAHE(clip_limit=clip_limit),
        T.Resize((image_size, image_size)),
        T.RandomHorizontalFlip(p=0.5),
        T.RandomVerticalFlip(p=0.5),    # Add vertical flip too (anatomy has no "up")
        T.ToTensor(),
        T.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ])
    # Apply to Validation too so the model sees the same "enhanced" view
    val_tfms = T.Compose([
        ApplyCLAHE(clip_limit=clip_limit),
        T.Resize((image_size, image_size)),
        T.ToTensor(),
        T.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ])
    return train_tfms, val_tfms

# 5. Training Loop Components
# ---------------------------
def train_one_epoch(model, loader, criterion, optimizer):
    model.train()
    losses, accs = [], []
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        logits = model(x)
        loss = criterion(logits, y)
        loss.backward()
        optimizer.step()

        preds = logits.argmax(1)
        acc = (preds == y).float().mean()
        losses.append(loss.item())
        accs.append(acc.item())
    return np.mean(losses), np.mean(accs)

@torch.no_grad()
def validate(model, loader, criterion):
    model.eval()
    losses, accs, all_preds, all_targets = [], [], [], []
    for x, y in loader:
        x, y = x.to(device), y.to(device)
        logits = model(x)
        loss = criterion(logits, y)
        preds = logits.argmax(1)

        losses.append(loss.item())
        accs.append((preds == y).float().mean().item())
        all_preds.extend(preds.cpu().numpy())
        all_targets.extend(y.cpu().numpy())

    f1 = f1_score(all_targets, all_preds, average='macro')
    # Use zero_division parameter if available or handle sparse classes

    return np.mean(losses), np.mean(accs), f1, all_preds, all_targets

# 6. Main Execution (5-Fold CV)
# -----------------------------
def main():
    # A. Prepare Data
    train_full_df, test_df = prepare_dataframes()

    # Define Transforms
    train_tfms, val_tfms = get_transforms(cfg.IMAGE_SIZE)
    test_ds = CervicalDataset(test_df, transform=val_tfms)
    test_loader = DataLoader(test_ds, batch_size=cfg.BATCH_SIZE, shuffle=False, num_workers=2)

    # Cross Validation Setup
    N_FOLDS = 5
    gkf = GroupKFold(n_splits=N_FOLDS)

    # Store results
    fold_results = []

    # Iterate through folds
    # Note: We split based on 'patient_id'
    print(f"\nStarting {N_FOLDS}-Fold Cross Validation...")

    # Keep track of groups for split
    groups = train_full_df['patient_id'].values

    for fold, (train_idx, val_idx) in enumerate(gkf.split(train_full_df, groups=groups)):
        print(f"\n{'='*20} FOLD {fold+1}/{N_FOLDS} {'='*20}")

        train_df = train_full_df.iloc[train_idx].reset_index(drop=True)
        val_df = train_full_df.iloc[val_idx].reset_index(drop=True)

        # B. Weighted Sampler (Recalculate for each fold)
        class_counts = train_df['class_num'].value_counts().sort_index()
        w_list = []
        for i in range(3):
            c = class_counts.get(i, 0)
            if c > 0: w_list.append(1.0/c)
            else: w_list.append(0.0)

        # Handle case where all weights might be 0 for a class (e.g., if a fold has no examples of a class)
        if sum(w_list) > 0:
            sample_weights = [w_list[label] for label in train_df['class_num']]
            sampler = WeightedRandomSampler(sample_weights, num_samples=len(sample_weights), replacement=True)
        else:
            # Fallback if no weights can be calculated (e.g., empty train_df)
            sampler = None # Or a simple SequentialSampler

        train_ds = CervicalDataset(train_df, transform=train_tfms)
        val_ds = CervicalDataset(val_df, transform=val_tfms)

        train_loader = DataLoader(train_ds, batch_size=cfg.BATCH_SIZE, sampler=sampler, num_workers=2)
        val_loader = DataLoader(val_ds, batch_size=cfg.BATCH_SIZE, shuffle=False, num_workers=2)

        # C. Model Setup (Re-init for each fold)
        print(f"Initializing model for Fold {fold+1}...")
        model = timm.create_model(cfg.MODEL_NAME, pretrained=True, num_classes=3)
        model = model.to(device)

        # Freeze/Unfreeze Logic
        for param in model.parameters():
```

```
        for param in model.parameters():
            param.requires_grad = False
        for param in model.classifier.parameters():
            param.requires_grad = True
        # Unfreeze blocks as per original logic
        for param in model.blocks[-1].parameters():
            param.requires_grad = True
        for param in model.blocks[-2].parameters():
            param.requires_grad = True
        for param in model.blocks[-3].parameters():
            param.requires_grad = True

        # Class Weights for Loss
        counts = train_df['class_num'].value_counts().sort_index()
        w_list = []
        for i in range(3):
            c = counts.get(i, 0)
            if c > 0: w_list.append(1.0/c)
            else: w_list.append(0.0)

        weights = torch.tensor(w_list, dtype=torch.float32).to(device)
        if weights.sum() > 0:
             weights = weights / weights.sum()
        else:
             # If all weights are zero (e.g., empty train_df for a class), set uniform weights or handle accordingly
             weights = torch.ones(3, dtype=torch.float32).to(device) / 3

        criterion = nn.CrossEntropyLoss(weight=weights, label_smoothing=0.1)
        optimizer = torch.optim.AdamW(model.parameters(), lr=5e-4, weight_decay=0.02)
        scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=cfg.EPOCHS)

        # Training Loop for this Fold
        best_f1 = -1.0
        fold_out_dir = Path(cfg.OUT_DIR) / f"fold_{fold+1}_{time.strftime('%m%d_%H%M')}"
        try:
            fold_out_dir.mkdir(parents=True, exist_ok=True)
        except Exception:
            pass # Ignore if exists

        for epoch in range(1, cfg.EPOCHS + 1):
            t_loss, t_acc = train_one_epoch(model, train_loader, criterion, optimizer)
            v_loss, v_acc, v_f1, _, _ = validate(model, val_loader, criterion)

            scheduler.step()

            if v_f1 > best_f1:
                best_f1 = v_f1
                torch.save(model.state_dict(), fold_out_dir / "best_model.pth")

            if epoch % 5 == 0:
                 print(f"  Ep {epoch} | T_Loss: {t_loss:.3f} | V_Loss: {v_loss:.3f} | V_F1: {v_f1:.3f}")

        print(f"Fold {fold+1} Best Val F1: {best_f1:.3f}")

        # Load Best Model for Test Set Eval
        model.load_state_dict(torch.load(fold_out_dir / "best_model.pth"))
        test_loss, test_acc, test_f1, _, _ = validate(model, test_loader, criterion)
        print(f"Fold {fold+1} Test Set F1: {test_f1:.3f}")

        fold_results.append({
            'fold': fold + 1,
            'val_f1': best_f1,
            'test_f1': test_f1,
            'test_acc': test_acc
        })

    # Summary
    print("\n" + "="*30)
    print("5-FOLD CV RESULTS")
    print("="*30)

    if len(fold_results) > 0:
        avg_val_f1 = np.mean([r['val_f1'] for r in fold_results])
        avg_test_f1 = np.mean([r['test_f1'] for r in fold_results])

        for res in fold_results:
            print(f"Fold {res['fold']}: Val F1 = {res['val_f1']:.3f}, Test F1 = {res['test_f1']:.3f}")

        print(f"\nAverage Val F1: {avg_val_f1:.3f}")
        print(f"Average Test F1: {avg_test_f1:.3f}")
    else:
        print("No results to display.")

main()
```

```
Using device: cuda
Debug: Sample keys from label_map: ['AFC0.jpg', 'AFC1.jpg', 'AJL0.jpg', 'AJL1.jpg', 'AGY0.jpg']
Scanning Training Directory...
Debug: Contents of /content/via_dataset_fixed/train: ['IARC_image_bank_AFA1_aug_15.jpg', 'IARC_image_bank_AEU1_aug_25.jpg', 'IARC_image_bank_AAW1_aug_
Debug: Found 1775 JPG files in /content/via_dataset_fixed/train.
  Debug: Image file: IARC_image_bank_AFA1_aug_15.jpg, Derived original_key: AFA1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_AEU1_aug_25.jpg, Derived original_key: AEU1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_AAW1_aug_0.jpg, Derived original_key: AAW1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_AAF1_aug_19.jpg, Derived original_key: AAF1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_ADE1_aug_22.jpg, Derived original_key: ADE1.jpg, Found in label_map: True
Found 1775 training images (including augmentations).
Scanning Test Directory...
Debug: Contents of /content/via_dataset_fixed/test: ['IARC_image_bank_ACW1.jpg', 'IARC_image_bank_ABU1.jpg', 'IARC_image_bank_AEI1.jpg', 'IARC_image_b
Debug: Found 40 JPG files in /content/via_dataset_fixed/test.
  Debug: Image file: IARC_image_bank_ACW1.jpg, Derived original_key: ACW1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_ABU1.jpg, Derived original_key: ABU1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_AEI1.jpg, Derived original_key: AEI1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_APB1.jpg, Derived original_key: APB1.jpg, Found in label_map: True
  Debug: Image file: IARC_image_bank_AAU1.jpg, Derived original_key: AAU1.jpg, Found in label_map: True
Found 40 test images.
------------------------------
Total Train Size: 1775 (for CV)
Final Test Size:  40 (Original only)
------------------------------

Starting 5-Fold Cross Validation...

==================== FOLD 1/5 ====================
Initializing model for Fold 1...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Goog
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model.safetensors: 100%                                   21.4M/21.4M [00:02<00:00, 15.8MB/s]
  Ep 5 | T_Loss: 0.352 | V_Loss: 0.855 | V_F1: 0.659
  Ep 10 | T_Loss: 0.306 | V_Loss: 0.875 | V_F1: 0.644
  Ep 15 | T_Loss: 0.305 | V_Loss: 0.903 | V_F1: 0.643
  Ep 20 | T_Loss: 0.299 | V_Loss: 0.893 | V_F1: 0.645
  Ep 25 | T_Loss: 0.298 | V_Loss: 0.895 | V_F1: 0.637
  Ep 30 | T_Loss: 0.297 | V_Loss: 0.906 | V_F1: 0.636
Fold 1 Best Val F1: 0.673
Fold 1 Test Set F1: 0.716

==================== FOLD 2/5 ====================
Initializing model for Fold 2...
  Ep 5 | T_Loss: 0.343 | V_Loss: 0.999 | V_F1: 0.553
  Ep 10 | T_Loss: 0.309 | V_Loss: 0.966 | V_F1: 0.545
  Ep 15 | T_Loss: 0.300 | V_Loss: 0.973 | V_F1: 0.557
  Ep 20 | T_Loss: 0.299 | V_Loss: 0.974 | V_F1: 0.542
  Ep 25 | T_Loss: 0.297 | V_Loss: 0.972 | V_F1: 0.528
  Ep 30 | T_Loss: 0.296 | V_Loss: 0.979 | V_F1: 0.526
Fold 2 Best Val F1: 0.595
Fold 2 Test Set F1: 0.574

==================== FOLD 3/5 ====================
Initializing model for Fold 3...
  Ep 5 | T_Loss: 0.346 | V_Loss: 1.107 | V_F1: 0.482
  Ep 10 | T_Loss: 0.307 | V_Loss: 1.026 | V_F1: 0.434
  Ep 15 | T_Loss: 0.301 | V_Loss: 1.015 | V_F1: 0.444
  Ep 20 | T_Loss: 0.300 | V_Loss: 1.028 | V_F1: 0.442
  Ep 25 | T_Loss: 0.296 | V_Loss: 1.027 | V_F1: 0.431
  Ep 30 | T_Loss: 0.298 | V_Loss: 1.020 | V_F1: 0.444
Fold 3 Best Val F1: 0.498
Fold 3 Test Set F1: 0.606

==================== FOLD 4/5 ====================
Initializing model for Fold 4...
  Ep 5 | T_Loss: 0.330 | V_Loss: 1.009 | V_F1: 0.561
  Ep 10 | T_Loss: 0.308 | V_Loss: 0.989 | V_F1: 0.544
  Ep 15 | T_Loss: 0.304 | V_Loss: 1.154 | V_F1: 0.462
  Ep 20 | T_Loss: 0.297 | V_Loss: 1.022 | V_F1: 0.527
  Ep 25 | T_Loss: 0.298 | V_Loss: 1.014 | V_F1: 0.531
  Ep 30 | T_Loss: 0.297 | V_Loss: 1.064 | V_F1: 0.503
Fold 4 Best Val F1: 0.604
Fold 4 Test Set F1: 0.759

==================== FOLD 5/5 ====================
Initializing model for Fold 5...
  Ep 5 | T_Loss: 0.328 | V_Loss: 1.021 | V_F1: 0.525
  Ep 10 | T_Loss: 0.308 | V_Loss: 0.963 | V_F1: 0.513
  Ep 15 | T_Loss: 0.302 | V_Loss: 0.969 | V_F1: 0.506
  Ep 20 | T_Loss: 0.299 | V_Loss: 0.969 | V_F1: 0.520
  Ep 25 | T_Loss: 0.297 | V_Loss: 0.957 | V_F1: 0.517
  Ep 30 | T_Loss: 0.298 | V_Loss: 0.950 | V_F1: 0.526
Fold 5 Best Val F1: 0.579
Fold 5 Test Set F1: 0.763

==============================
5-FOLD CV RESULTS
==============================
Fold 1: Val F1 = 0.673, Test F1 = 0.716
Fold 2: Val F1 = 0.595, Test F1 = 0.574
Fold 3: Val F1 = 0.498, Test F1 = 0.606
Fold 4: Val F1 = 0.604, Test F1 = 0.759
Fold 5: Val F1 = 0.579, Test F1 = 0.763

Average Val F1: 0.590
Average Test F1: 0.684
```