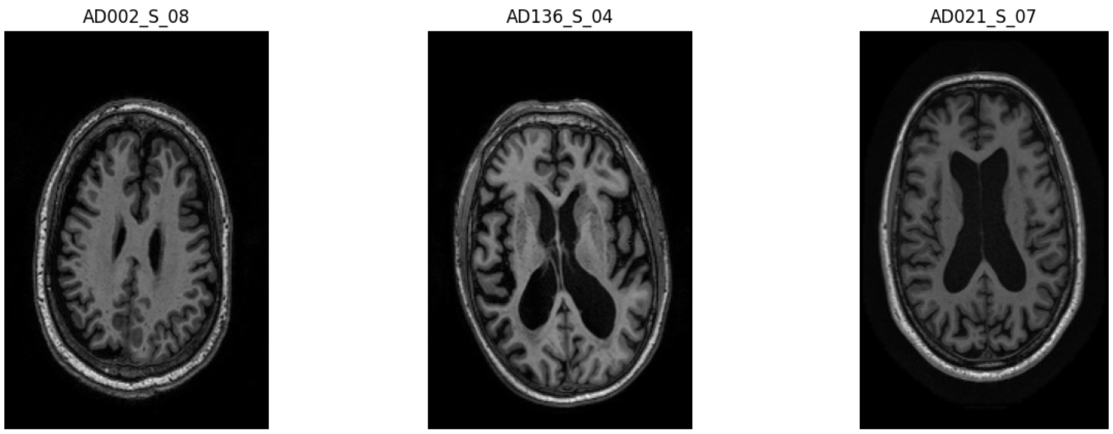
alzheimer-detection-part-3

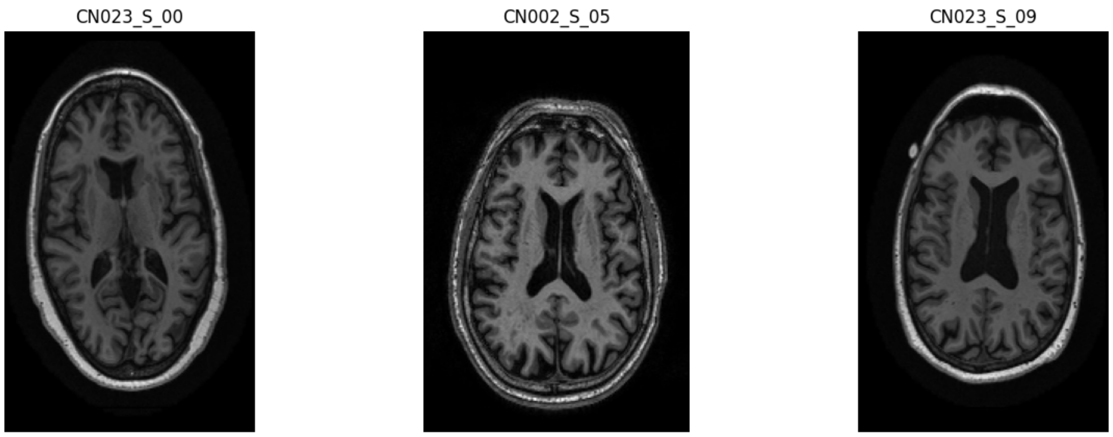
October 28, 2025

[1]:
```
import os
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

# Keep your original directories and subfolders
test_dir = "/kaggle/input/alzheimer-disease/test-20251010T094612Z-1-001/test"
train_dir = "/kaggle/input/alzheimer-disease/train-20251010T183510Z-1-001/train"
subfolders = ["AD", "CN", "MCI"]

# Function to show 3 images from each subfolder (unchanged signature)
def show_images_from_dir(path, n=3):
    # List only image files (same behavior; expanded to be robust if needed)
    files = [f for f in os.listdir(path) if f.lower().endswith(('.png', '.jpg',␣
 ↪'.jpeg', '.bmp', '.tif', '.tiff'))]
    files = files[:n]  # Take only the first n images

    plt.figure(figsize=(15, 5))
    for i, file in enumerate(files):
        img_path = os.path.join(path, file)

        # Open with PIL
        img = Image.open(img_path)

        # --- CRITICAL FIX ---
        # Force grayscale to ensure single-channel display (no colorization)
        # Even if the source is already grayscale, this guarantees mode 'L'
        img = img.convert('L')

        # Convert to numpy for safe imshow with explicit bounds
        arr = np.asarray(img)

        # Display as true grayscale with fixed intensity bounds
        plt.subplot(1, len(files), i + 1)
        plt.imshow(arr, cmap='gray', vmin=0, vmax=255)
        plt.axis("off")
        plt.title(file[:10])
    plt.show()


# Show images for Test set
for subfolder in subfolders:
    print(f"{subfolder} (Test)")
    show_images_from_dir(os.path.join(test_dir, subfolder))

# Show images for Train set
for subfolder in subfolders:
    print(f"{subfolder} (Train)")
    show_images_from_dir(os.path.join(train_dir, subfolder))
```

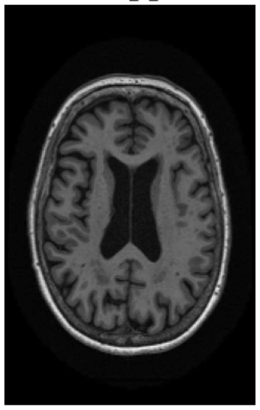
```
AD (Test)
```

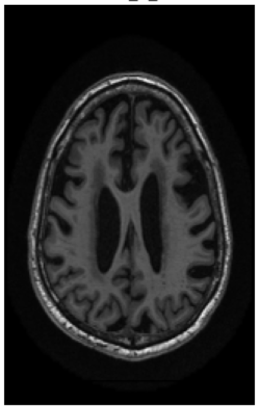


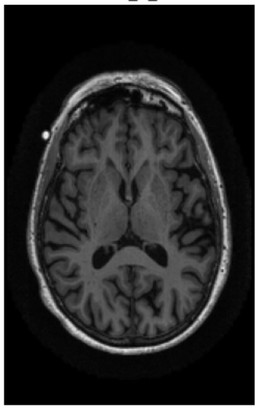
```
CN (Test)
```



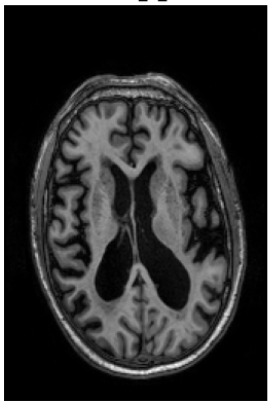
MCI (Test)

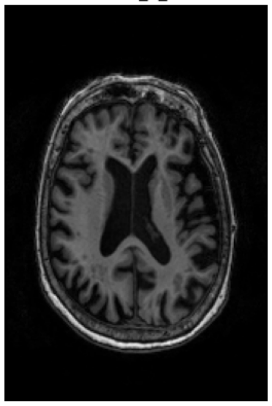

CI128_S_11

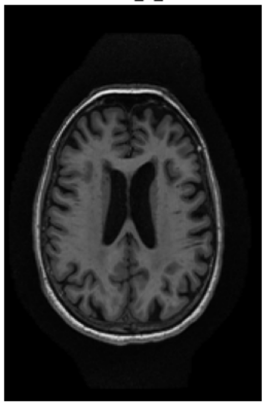

CI027_S_12


CI067_S_06

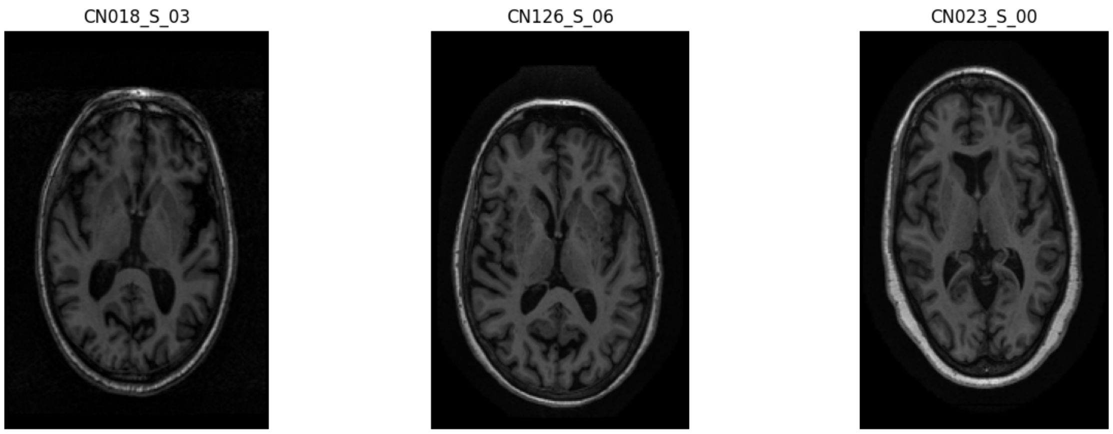
AD (Train)


AD136_S_04

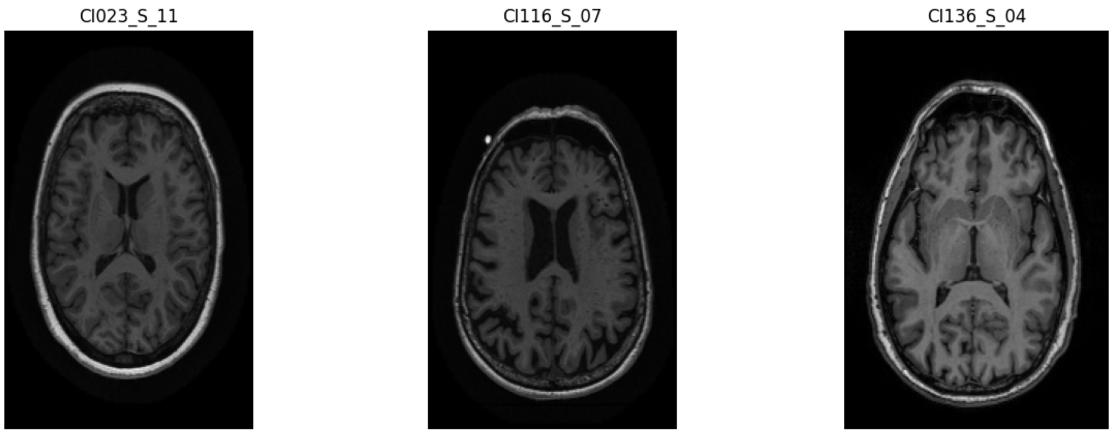

AD136_S_02


AD007_S_13

CN (Train)



MCI (Train)



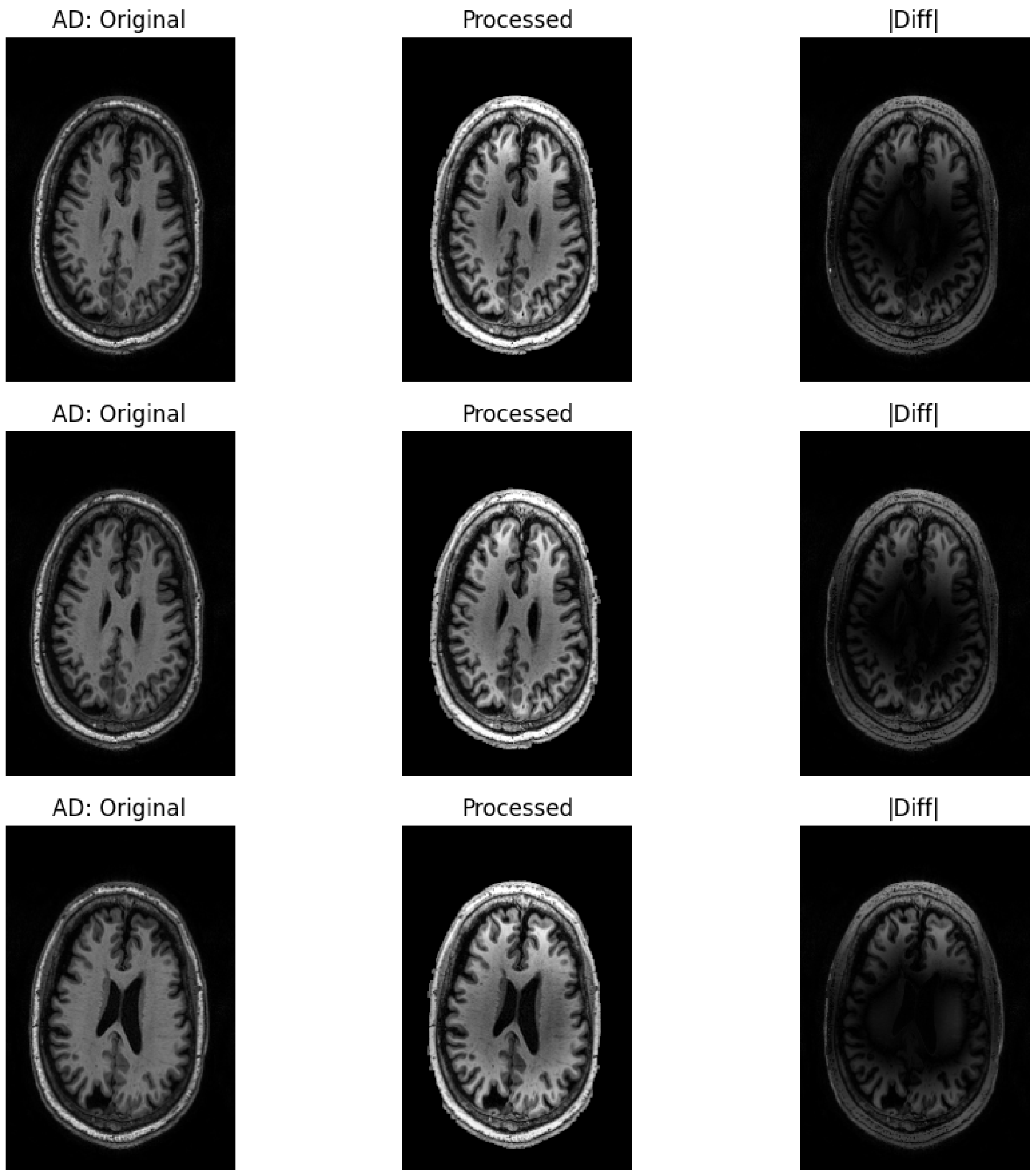
[2]:
```python
import os
import sys
import numpy as np
from PIL import Image, UnidentifiedImageError
import matplotlib.pyplot as plt

# SciPy for morphology / connected components (Kaggle preinstalled)
from scipy.ndimage import (
    gaussian_filter,
    binary_opening,
    binary_closing,
    binary_fill_holes,
    label
)
```

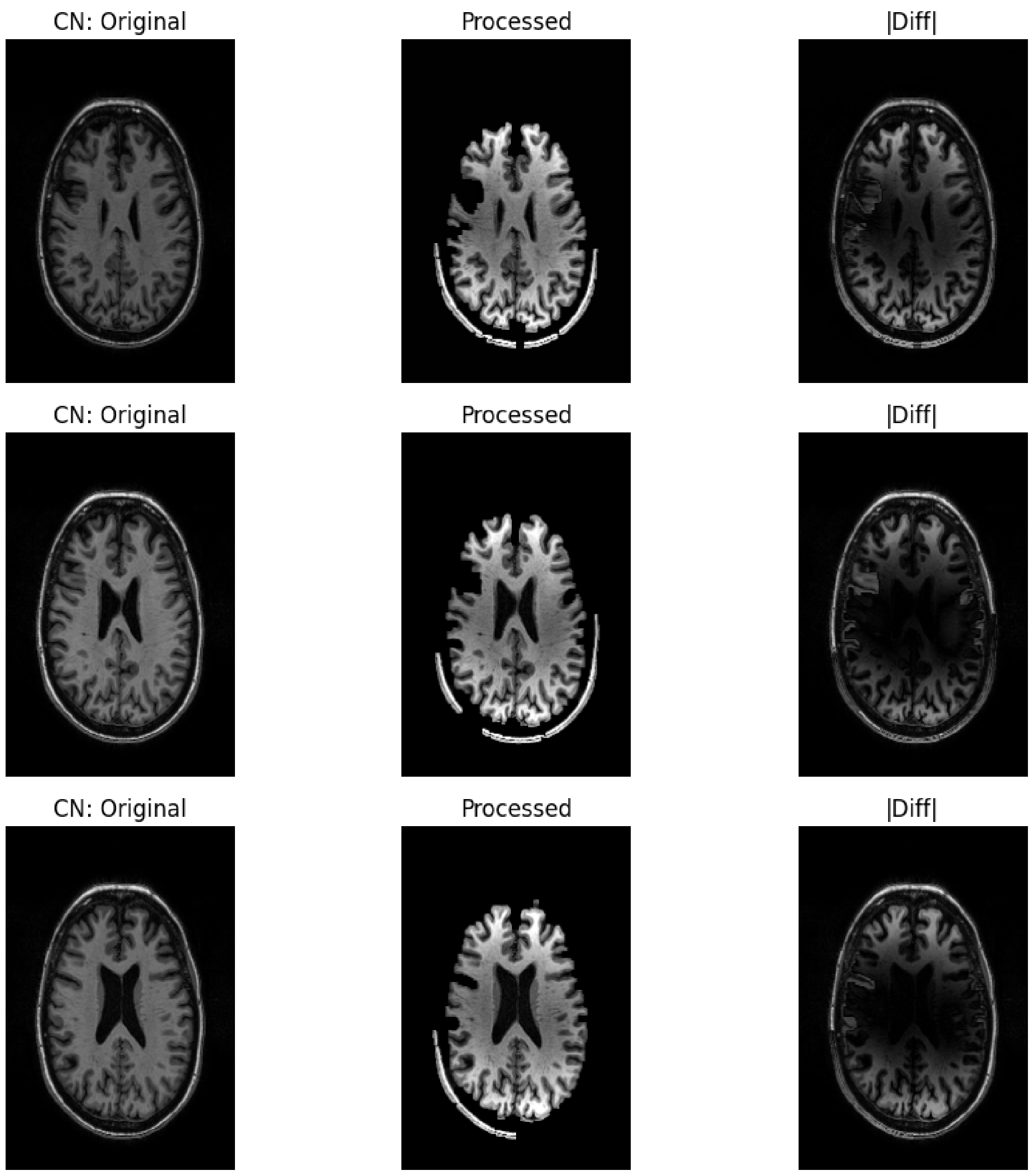
```
# Try skimage for CLAHE; fall back gracefully if missing
try:
    from skimage.exposure import equalize_adapthist
    _HAS_SKIMAGE = True
except Exception:
    _HAS_SKIMAGE = False


# GPU via PyTorch (for homomorphic filtering)

def _ensure(pkg):
    import importlib
    try:
        importlib.import_module(pkg)
    except Exception:
        import subprocess
        subprocess.check_call([sys.executable, "-m", "pip", "install", pkg,␣
 ↪"--quiet"])

_ensure("torch")
import torch
import torch.nn.functional as F

def torch_device():
    return torch.device("cuda" if torch.cuda.is_available() else "cpu")

def gaussian_kernel_2d(sigma: float, radius_factor: float = 3.0, device=None,␣
 ↪dtype=torch.float32):
    """Create a 2D Gaussian kernel tensor for conv2d (normalized)."""
    device = device or torch_device()
    rad = max(1, int(radius_factor * sigma))
    xs = torch.arange(-rad, rad + 1, device=device, dtype=dtype)
    g1 = torch.exp(-0.5 * (xs / sigma) ** 2)
    g1 = g1 / g1.sum()
    g2 = g1[:, None] @ g1[None, :]
    g2 = g2 / g2.sum()
    return g2  # (K, K)

def homomorphic_filter_gpu_u8(arr_u8: np.ndarray, sigma: float = 50.0) -> np.
 ↪ndarray:
    """
    GPU homomorphic filtering (log → low-pass via conv2d → exp → robust␣
 ↪rescale).
    Input uint8 [0..255], output uint8 [0..255].
    """
    dev = torch_device()
    # [B=1,C=1,H,W] float32
    img = torch.from_numpy(arr_u8.astype(np.float32)).to(dev) + 1.0  # avoid␣
↪log(0)
    img = img[None, None, :, :]  # NCHW
    loga = torch.log(img)
    # 2D Gaussian low-pass via conv2d (reflect padding to avoid border␣
↪artifacts)
    k = gaussian_kernel_2d(sigma=sigma, device=dev)
    k = k[None, None, :, :]  # (out_c,in_c,H,W)
    pad = k.shape[-1] // 2
    low = F.conv2d(F.pad(loga, (pad, pad, pad, pad), mode="reflect"), k)
    high = loga - low
    corr = torch.exp(high) - 1.0  # back to linear domain

    # Robust percentile rescale to uint8
    a = corr.squeeze(0).squeeze(0)  # HxW
    # Compute percentiles on CPU for simplicity
    a_np = a.detach().cpu().numpy().astype(np.float32)
    lo, hi = np.percentile(a_np, [0.5, 99.5])
    if hi - lo < 1e-6:
        a_np = (a_np - a_np.min()) / (a_np.ptp() + 1e-8)
    else:
        a_np = np.clip((a_np - lo) / (hi - lo), 0.0, 1.0)
    out_u8 = (a_np * 255.0 + 0.5).astype(np.uint8)
    return out_u8


test_dir = "/kaggle/input/alzheimer-disease/test-20251010T094612Z-1-001/test"
train_dir = "/kaggle/input/alzheimer-disease/train-20251010T183510Z-1-001/train"
subfolders = ["AD", "CN", "MCI"]


# Output (mirrors the structure)

preprocessed_root = "/kaggle/working/alzheimer-preprocessed"
preprocessed_test = os.path.join(preprocessed_root, "test")
preprocessed_train = os.path.join(preprocessed_root, "train")

IMG_EXT = (".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff")

def ensure_dir(p: str):
    os.makedirs(p, exist_ok=True)


# Utilities

def load_gray(path: str) -> np.ndarray:
    """Load as uint8 grayscale [0..255]."""
    return np.asarray(Image.open(path).convert("L"), dtype=np.uint8)

def save_gray(arr_u8: np.ndarray, path: str):
    Image.fromarray(arr_u8, mode="L").save(path)

def percentile_rescale_u8(arr: np.ndarray, p_low=1.0, p_high=99.0) -> np.
 ↪ndarray:
    """Robust contrast stretching to uint8."""
    a = arr.astype(np.float32)
    lo, hi = np.percentile(a, [p_low, p_high])
    if hi - lo < 1e-6:
        a = (a - a.min()) / (a.ptp() + 1e-8)
    else:
        a = np.clip((a - lo) / (hi - lo), 0.0, 1.0)
    return (a * 255.0 + 0.5).astype(np.uint8)

def otsu_threshold_u8(arr_u8: np.ndarray) -> int:
    """Pure NumPy Otsu threshold (returns 0..255)."""
    hist = np.bincount(arr_u8.ravel(), minlength=256).astype(np.float64)
    prob = hist / (arr_u8.size + 1e-12)
    omega = np.cumsum(prob)
    mu = np.cumsum(prob * np.arange(256))
    mu_t = mu[-1]
    sigma_b2 = (mu_t * omega - mu) ** 2 / (omega * (1 - omega) + 1e-12)
    sigma_b2[~np.isfinite(sigma_b2)] = 0.0
    return int(np.argmax(sigma_b2))

def largest_cc(mask: np.ndarray) -> np.ndarray:
    """Keep only largest connected component of a binary mask."""
    lbl, n = label(mask.astype(np.uint8))
    if n <= 1:
        return mask.astype(bool)
    counts = np.bincount(lbl.ravel())
    counts[0] = 0  # background
    keep = counts.argmax()
    return (lbl == keep)

def clahe_u8(arr_u8: np.ndarray) -> np.ndarray:
    """CLAHE if skimage is available; otherwise identity."""
    if _HAS_SKIMAGE:
        arr01 = arr_u8.astype(np.float32) / 255.0
        out01 = equalize_adapthist(arr01, clip_limit=0.01)
        return (np.clip(out01, 0.0, 1.0) * 255.0 + 0.5).astype(np.uint8)
    else:
        return arr_u8
```

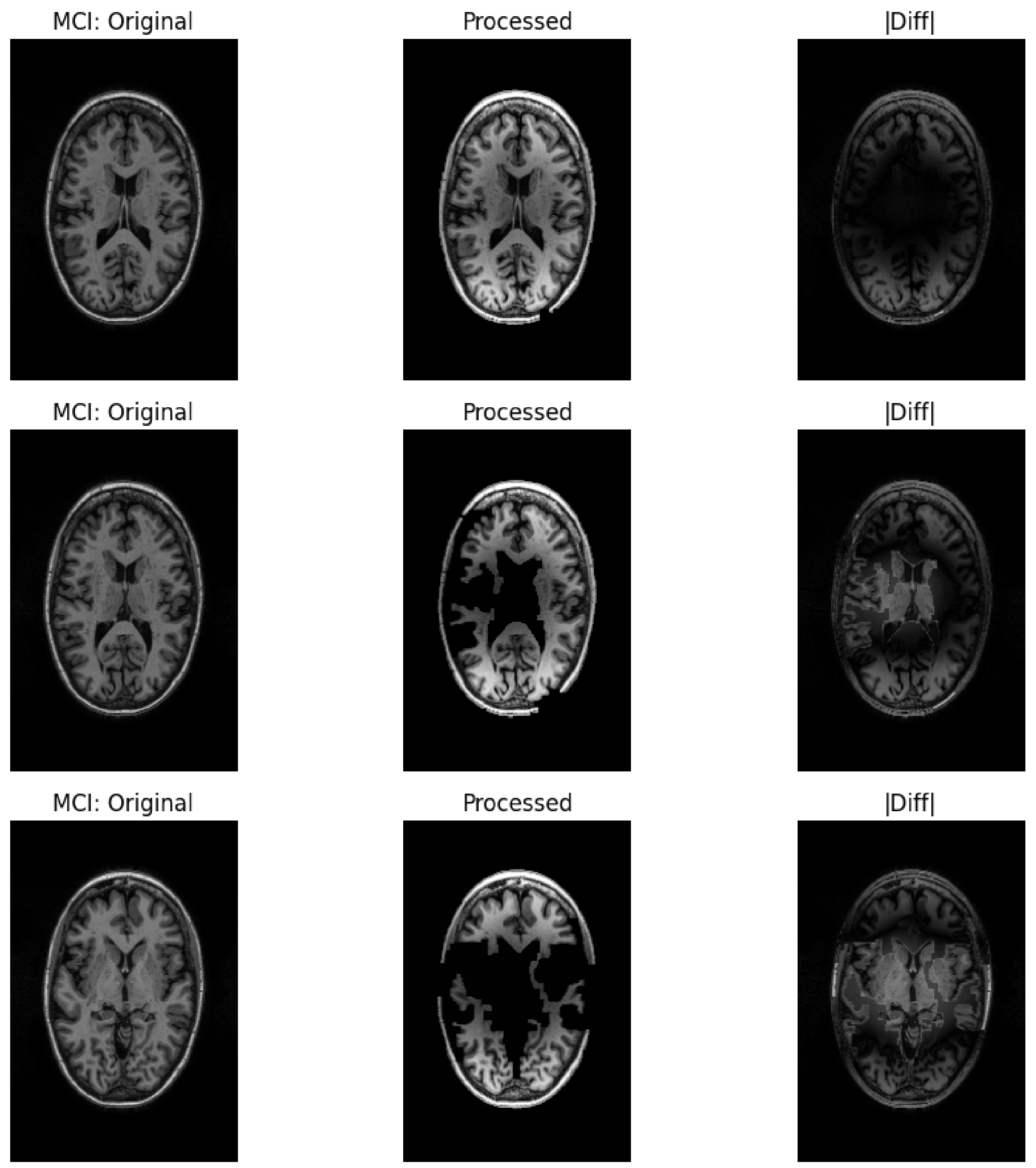
```python
# Core: one-image preprocessing

def preprocess_single_image(
    arr_u8: np.ndarray,
    do_skull_strip: bool = True,
    sigma_homomorphic: float = 50.0,
    min_area: int = 500
):
    """
    Steps:
      1) Robust pre-stretch (percentile)
      2) GPU homomorphic filter (PyTorch)
      3) (Optional) 2D skull/background stripping via Otsu + morphology + LCC +␣
 ↪hole fill
      4) Gentle CLAHE (if available), else percentile rescale
    Returns processed uint8 image, plus an optional mask (uint8).
    """
    # 1) Robust stretch
    a1 = percentile_rescale_u8(arr_u8, 1.0, 99.0)

    # 2) Homomorphic filtering on GPU
    a2 = homomorphic_filter_gpu_u8(a1, sigma=sigma_homomorphic)

    brain_mask = None
    if do_skull_strip:
        # 3) Otsu thresholding (foreground bright)
        th = otsu_threshold_u8(a2)
        mask = (a2 >= th).astype(np.uint8)

        # Morphological clean-up
        mask = binary_opening(mask, structure=np.ones((3,3), dtype=np.uint8))
        mask = binary_closing(mask, structure=np.ones((5,5), dtype=np.uint8))
        mask = binary_fill_holes(mask)
        mask = largest_cc(mask)

        # Remove tiny masks (safety)
        if mask.sum() < min_area:
            mask = np.ones_like(mask, dtype=bool)  # fallback: keep as-is

        a3 = (a2 * mask).astype(np.uint8)
        brain_mask = (mask.astype(np.uint8) * 255)
    else:
        a3 = a2

    # 4) CLAHE or robust stretch
    a4 = clahe_u8(a3)
    a4 = percentile_rescale_u8(a4, 0.5, 99.5)
    return a4, brain_mask


# Dataset-level processing & visualization

def preprocess_dataset(src_root: str, dst_root: str, n_preview: int = 3,␣
↪do_skull_strip=True):
    """
    Applies the pipeline to all images under src_root/{AD|CN|MCI}
    and writes to dst_root/{AD|CN|MCI}. Silent on per-file issues.
    """
    ensure_dir(dst_root)
    summary = {}

    # Report GPU/CPU
    dev = torch_device()
    print(f"Device for homomorphic filtering: {dev}")

    for cls in subfolders:
        src_cls = os.path.join(src_root, cls)
        dst_cls = os.path.join(dst_root, cls)
        ensure_dir(dst_cls)

        processed = skipped = 0
        if not os.path.isdir(src_cls):
            summary[cls] = (0, 0)
            continue

        files = sorted([f for f in os.listdir(src_cls) if f.lower().
↪endswith(IMG_EXT)])

        # Process & save
        for fname in files:
            spath = os.path.join(src_cls, fname)
            dpath = os.path.join(dst_cls, fname)
            try:
                arr = load_gray(spath)
                proc, _ = preprocess_single_image(arr,␣
↪do_skull_strip=do_skull_strip)
                save_gray(proc, dpath)
                processed += 1
            except (UnidentifiedImageError, OSError, RuntimeError, ValueError):
                skipped += 1
                continue

        summary[cls] = (processed, skipped)
        # Preview few examples
        preview = files[:n_preview]
        if preview:
            fig, axs = plt.subplots(len(preview), 3, figsize=(10,␣
↪3*len(preview)))
            if len(preview) == 1:
                axs = np.expand_dims(axs, 0)
            for i, fname in enumerate(preview):
                sp = os.path.join(src_cls, fname)
                dp = os.path.join(dst_cls, fname)
                try:
                    orig = load_gray(sp)
                    proc = load_gray(dp)
                    diff = np.abs(proc.astype(np.int16) - orig.astype(np.
↪int16)).astype(np.uint8)

                    axs[i, 0].imshow(orig, cmap="gray", vmin=0, vmax=255);␣
↪axs[i,0].axis("off"); axs[i,0].set_title(f"{cls}: Original")
                    axs[i, 1].imshow(proc, cmap="gray", vmin=0, vmax=255);␣
↪axs[i,1].axis("off"); axs[i,1].set_title("Processed")
                    axs[i, 2].imshow(diff, cmap="gray", vmin=0, vmax=255);␣
↪axs[i,2].axis("off"); axs[i,2].set_title("|Diff|")
                except Exception:
                    continue
            plt.tight_layout()
            plt.show()

    return summary


print("=== 2D MRI Preprocessing (GPU homomorphic + 2D skull-strip): TEST ===")
sum_test = preprocess_dataset(test_dir, preprocessed_test, n_preview=3,␣
↪do_skull_strip=True)

print("=== 2D MRI Preprocessing (GPU homomorphic + 2D skull-strip): TRAIN ===")
sum_train = preprocess_dataset(train_dir, preprocessed_train, n_preview=3,␣
↪do_skull_strip=True)

def _fmt(s): return ", ".join([f"{k}: {v[0]} ok / {v[1]} skipped" for k, v in s.
↪items()])
print("Preprocessed images saved to:")
print(f"  • Test : {preprocessed_test}")
print(f"  • Train: {preprocessed_train}")
print("Summary TEST :", _fmt(sum_test))
print("Summary TRAIN:", _fmt(sum_train))
```

```
=== 2D MRI Preprocessing (GPU homomorphic + 2D skull-strip): TEST ===
Device for homomorphic filtering: cuda
```

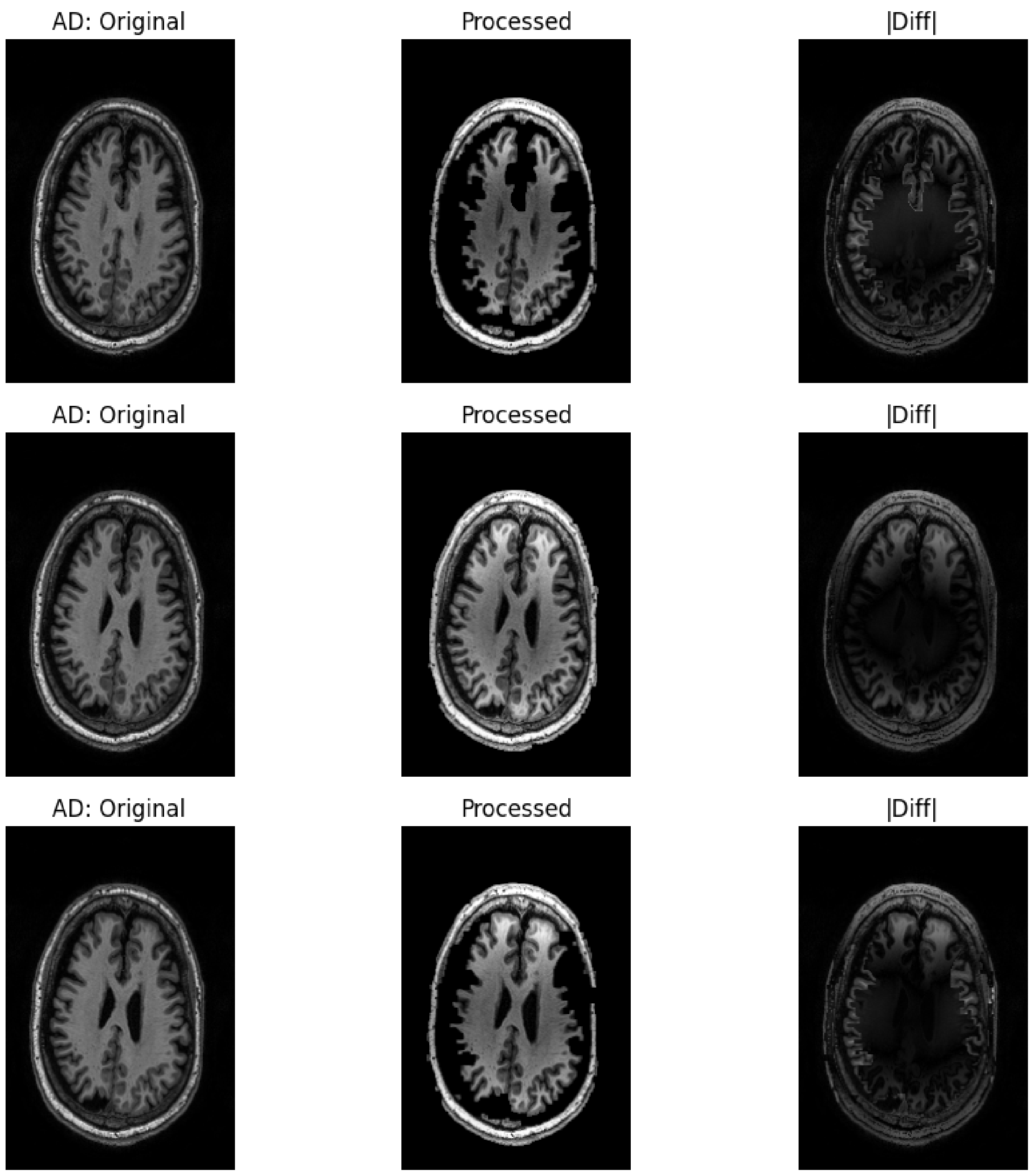
```
/tmp/ipykernel_19/2569239085.py:107: DeprecationWarning: 'mode' parameter is
deprecated and will be removed in Pillow 13 (2026-10-15)
  Image.fromarray(arr_u8, mode="L").save(path)
```

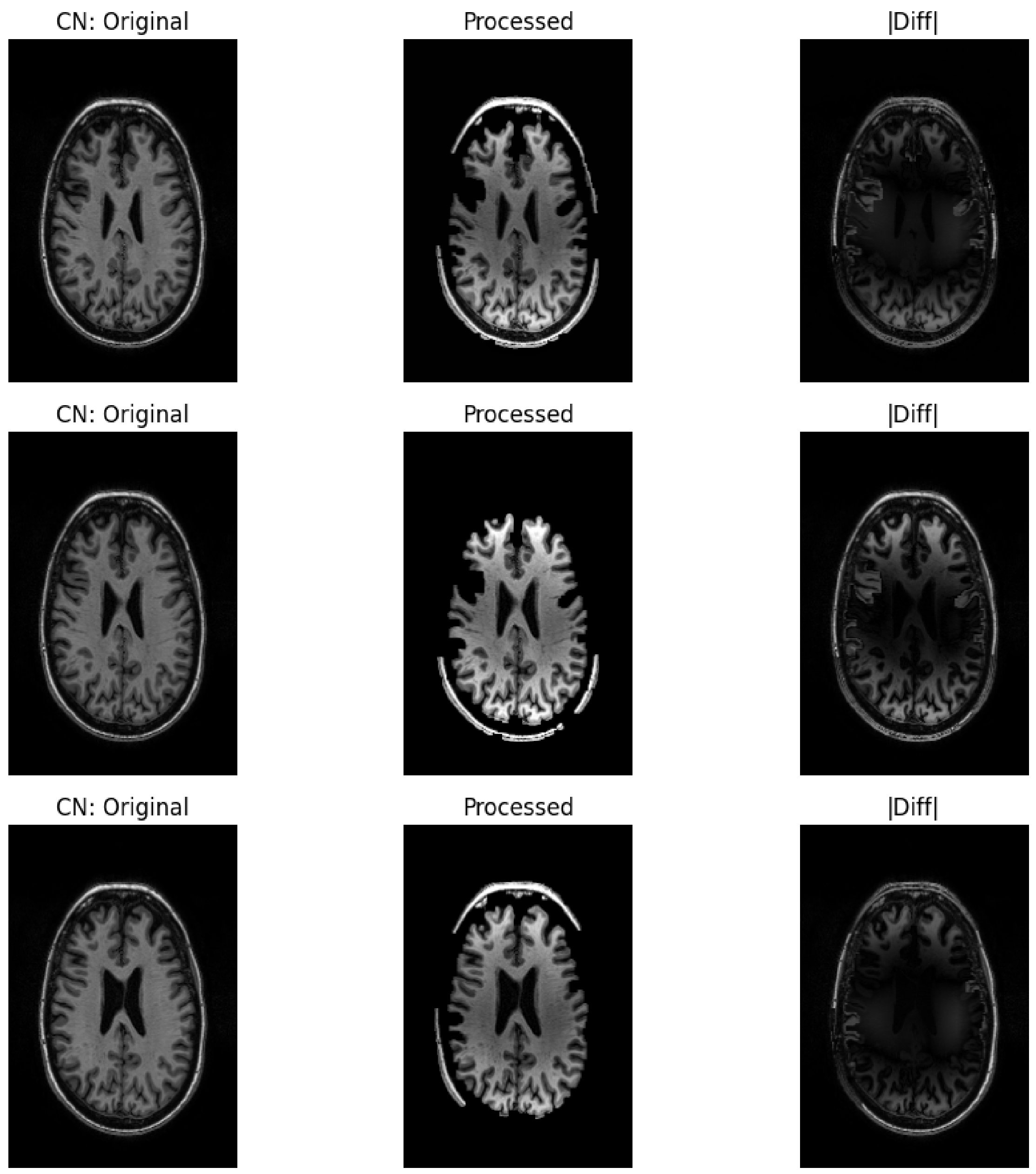


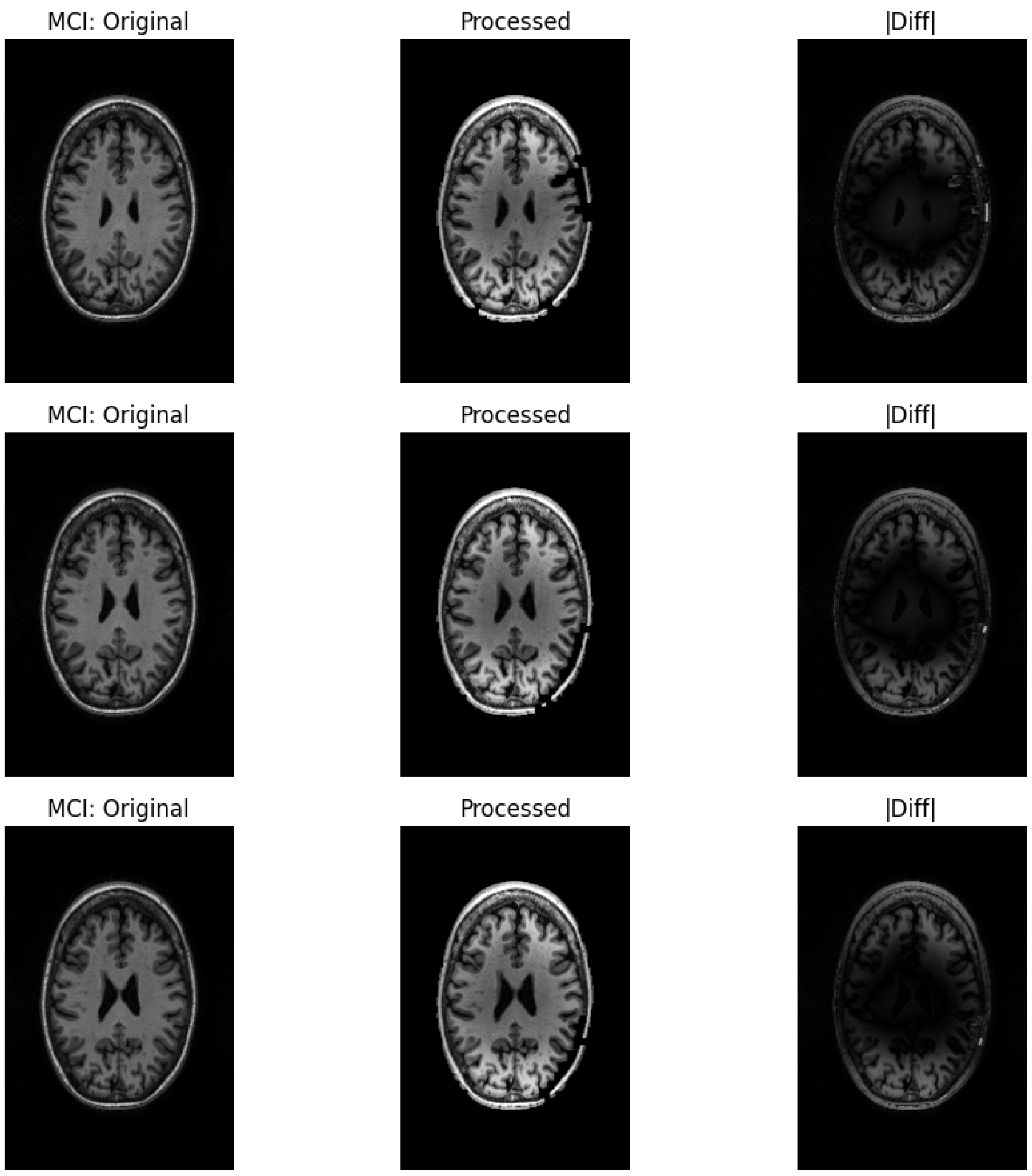

CN: Original
Processed
|Diff|
CN: Original
Processed
|Diff|
CN: Original
Processed
|Diff|



```
=== 2D MRI Preprocessing (GPU homomorphic + 2D skull-strip): TRAIN ===
Device for homomorphic filtering: cuda
```


AD: Original
Processed
|Diff|
AD: Original
Processed
|Diff|
AD: Original
Processed
|Diff|


CN: Original
Processed
|Diff|
CN: Original
Processed
|Diff|
CN: Original
Processed
|Diff|



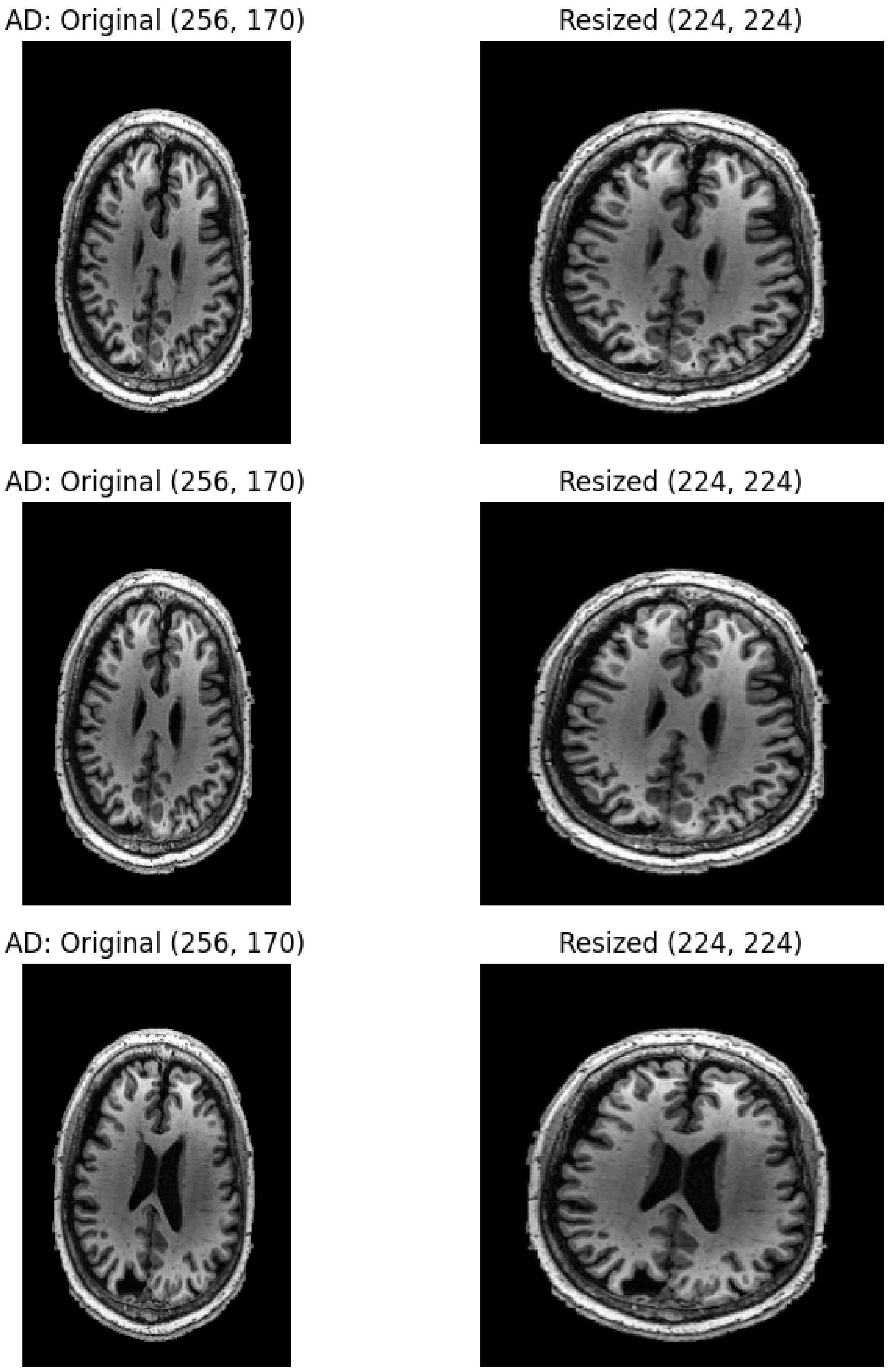
```
Preprocessed images saved to:
  • Test : /kaggle/working/alzheimer-preprocessed/test
  • Train: /kaggle/working/alzheimer-preprocessed/train
Summary TEST : AD: 225 ok / 0 skipped, CN: 288 ok / 0 skipped, MCI: 518 ok / 0
skipped
Summary TRAIN: AD: 899 ok / 0 skipped, CN: 1152 ok / 0 skipped, MCI: 2072 ok / 0
skipped
```

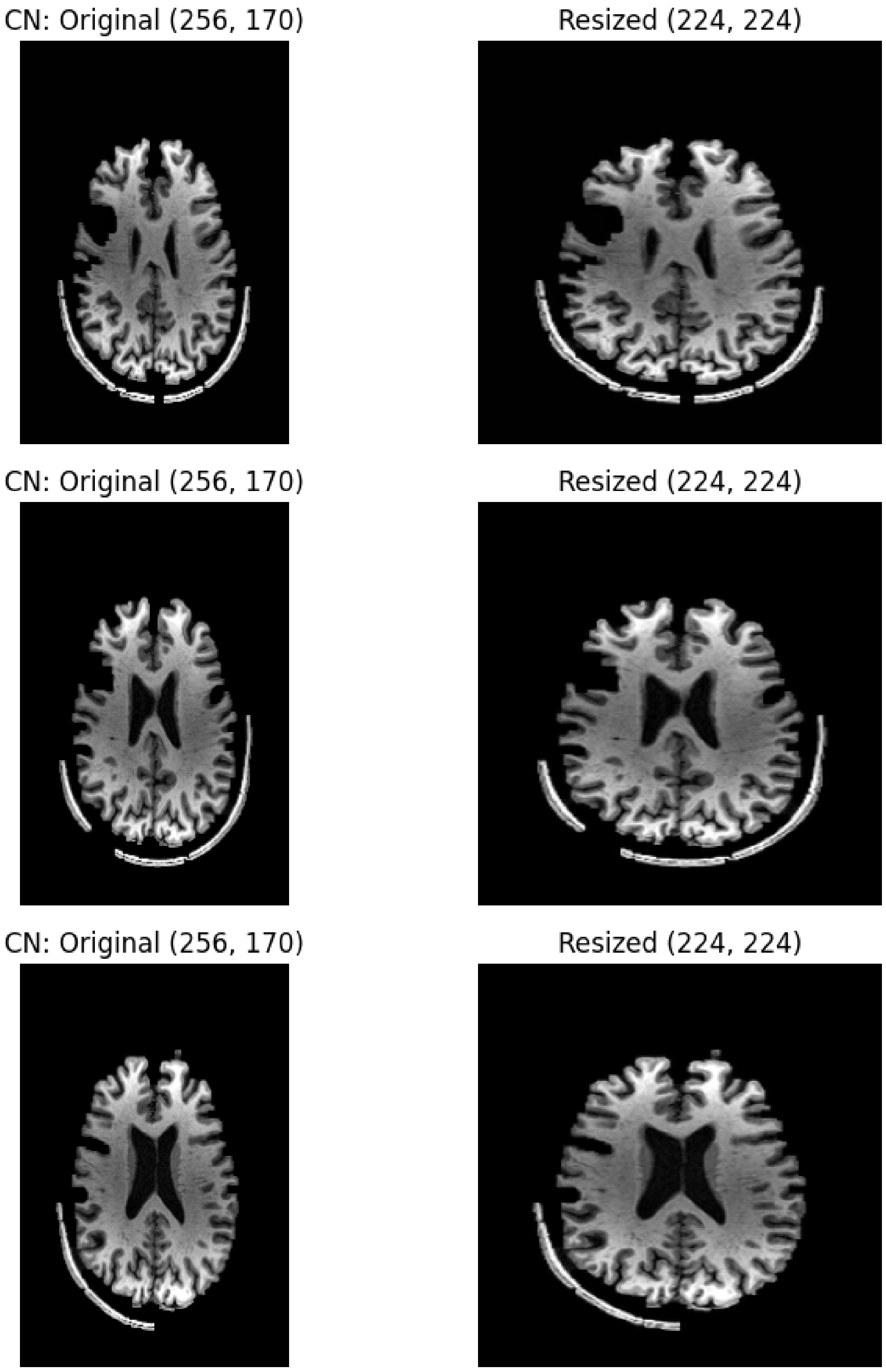
[3]:
```python
import os
import numpy as np
from PIL import Image, UnidentifiedImageError
import matplotlib.pyplot as plt

test_dir = "/kaggle/input/alzheimer-disease/test-20251010T094612Z-1-001/test"
train_dir = "/kaggle/input/alzheimer-disease/train-20251010T183510Z-1-001/train"
subfolders = ["AD", "CN", "MCI"]

preprocessed_test = "/kaggle/working/alzheimer-preprocessed/test"
preprocessed_train = "/kaggle/working/alzheimer-preprocessed/train"
resized_root = "/kaggle/working/alzheimer-resized-224"
resized_test = os.path.join(resized_root, "test")
resized_train = os.path.join(resized_root, "train")

IMG_EXT = (".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff")
TARGET_SIZE = (224, 224)

def ensure_dir(p: str):
    os.makedirs(p, exist_ok=True)

def load_gray(path: str) -> np.ndarray:
    return np.asarray(Image.open(path).convert("L"), dtype=np.uint8)

def resize_and_save_gray(arr_u8: np.ndarray, path: str):
    img = Image.fromarray(arr_u8, mode="L")
    img_resized = img.resize(TARGET_SIZE, Image.Resampling.LANCZOS)
    img_resized.save(path)

def resize_dataset(src_root: str, dst_root: str, n_preview: int = 3):
    ensure_dir(dst_root)
    summary = {}

    for cls in subfolders:
        src_cls = os.path.join(src_root, cls)
        dst_cls = os.path.join(dst_root, cls)
        ensure_dir(dst_cls)

        processed = skipped = 0
        if not os.path.isdir(src_cls):
            summary[cls] = (0, 0)
            continue

        files = sorted([f for f in os.listdir(src_cls) if f.lower().
↪endswith(IMG_EXT)])

        for fname in files:
            spath = os.path.join(src_cls, fname)
            dpath = os.path.join(dst_cls, fname)
            try:
                arr = load_gray(spath)
                resize_and_save_gray(arr, dpath)
                processed += 1
            except (UnidentifiedImageError, OSError, RuntimeError, ValueError):
                skipped += 1
                continue

        summary[cls] = (processed, skipped)

        preview = files[:n_preview]
        if preview:
            fig, axs = plt.subplots(len(preview), 2, figsize=(8,␣
↪3*len(preview)))
            if len(preview) == 1:
                axs = np.expand_dims(axs, 0)
            for i, fname in enumerate(preview):
                sp = os.path.join(src_cls, fname)
                dp = os.path.join(dst_cls, fname)
                try:
                    orig = load_gray(sp)
                    resized = load_gray(dp)

                    axs[i, 0].imshow(orig, cmap="gray", vmin=0, vmax=255)
                    axs[i, 0].axis("off")
                    axs[i, 0].set_title(f"{cls}: Original {orig.shape}")

                    axs[i, 1].imshow(resized, cmap="gray", vmin=0, vmax=255)
                    axs[i, 1].axis("off")
                    axs[i, 1].set_title(f"Resized {resized.shape}")
                except Exception:
                    continue
            plt.tight_layout()
            plt.show()

    return summary

print("=== Resizing Preprocessed Images: TEST ===")
sum_test = resize_dataset(preprocessed_test, resized_test, n_preview=3)

print("=== Resizing Preprocessed Images: TRAIN ===")
sum_train = resize_dataset(preprocessed_train, resized_train, n_preview=3)

def _fmt(s): return ", ".join([f"{k}: {v[0]} ok / {v[1]} skipped" for k, v in s.
↪items()])
```

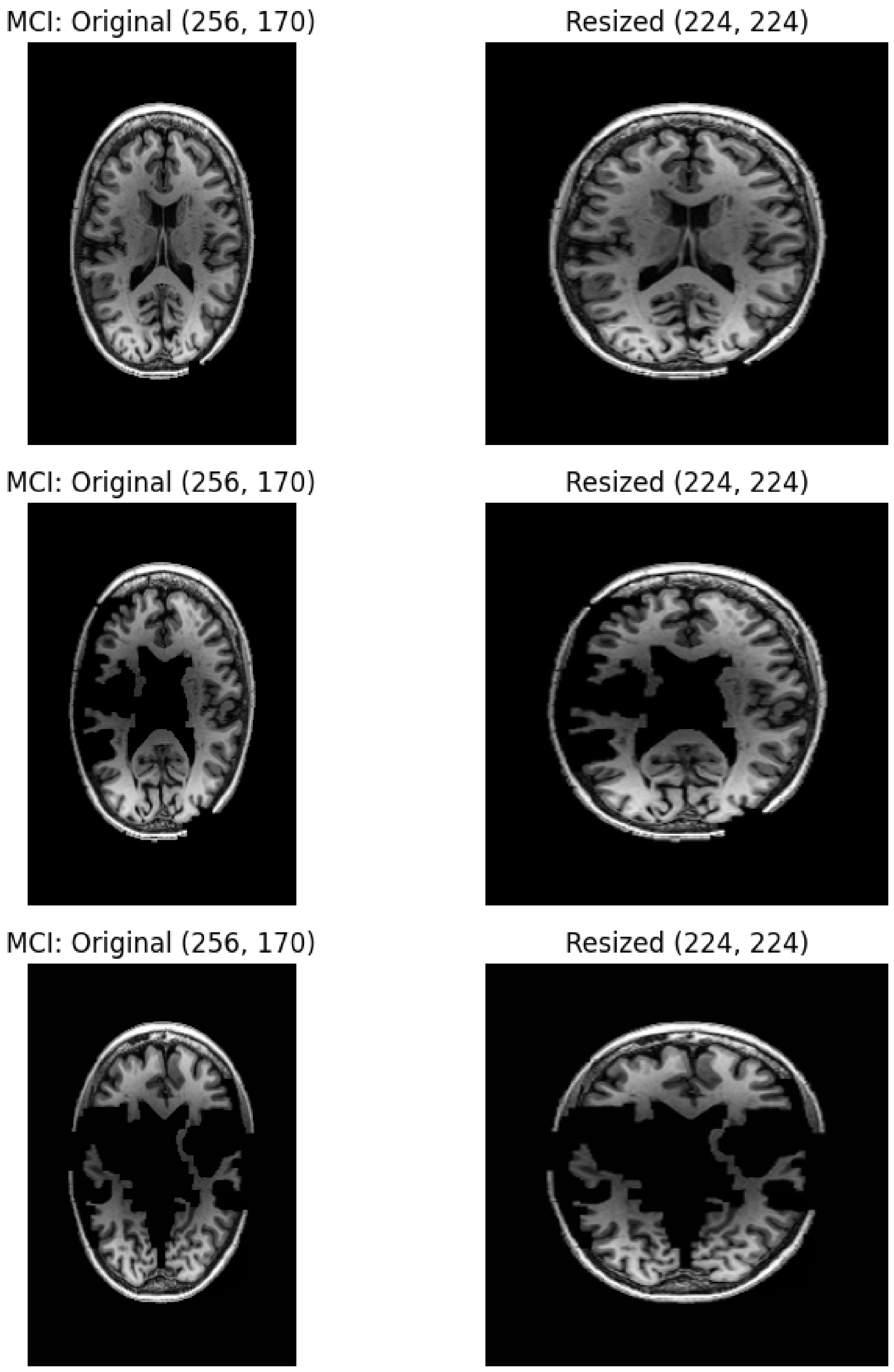
```python
print("Resized images saved to:")
print(f"  • Resized Test : {resized_test}")
print(f"  • Resized Train: {resized_train}")
print("Summary TEST :", _fmt(sum_test))
print("Summary TRAIN:", _fmt(sum_train))
```

```
=== Resizing Preprocessed Images: TEST ===
```

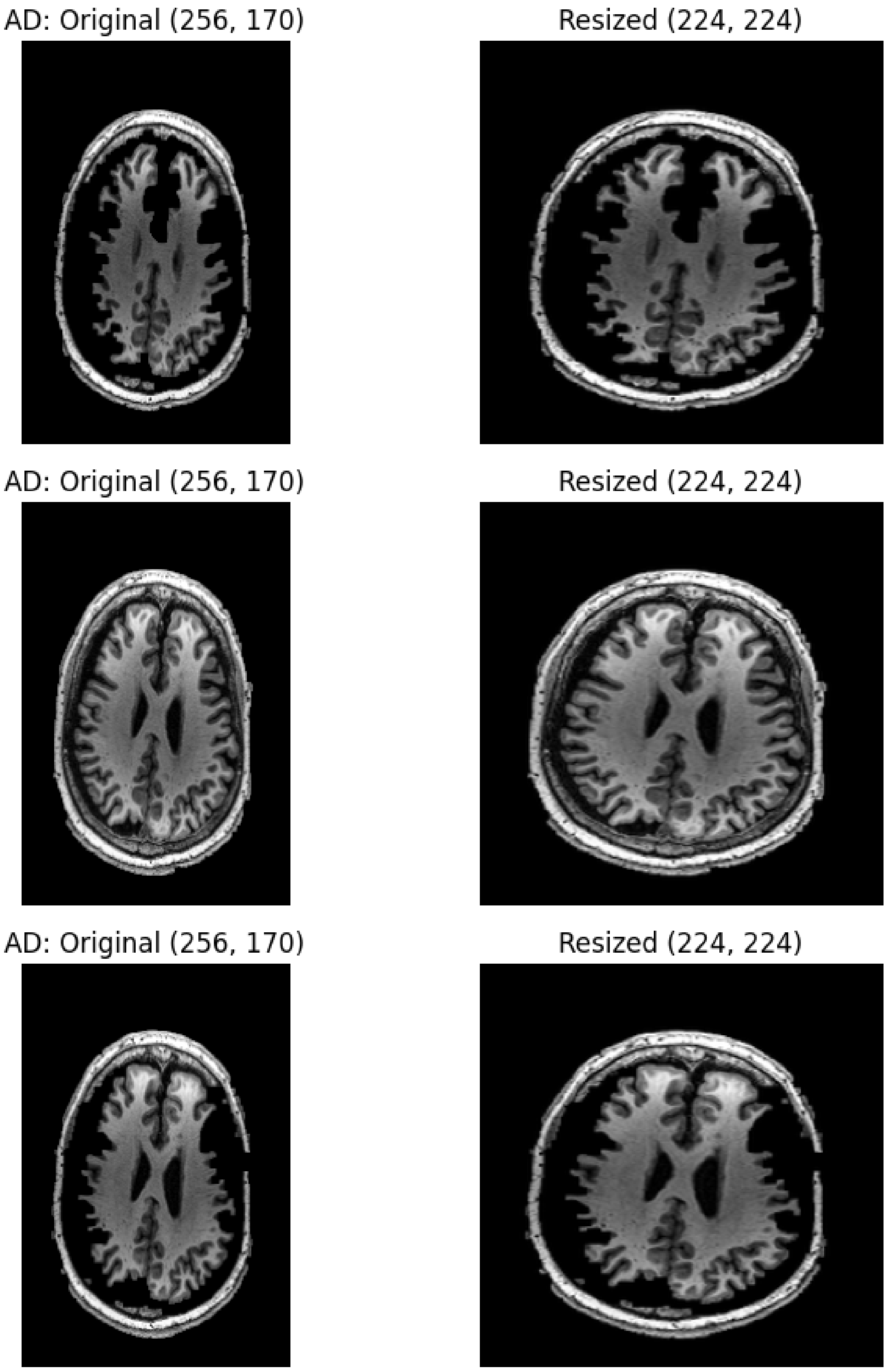
```
/tmp/ipykernel_19/502029755.py:26: DeprecationWarning: 'mode' parameter is
deprecated and will be removed in Pillow 13 (2026-10-15)
  img = Image.fromarray(arr_u8, mode="L")
```

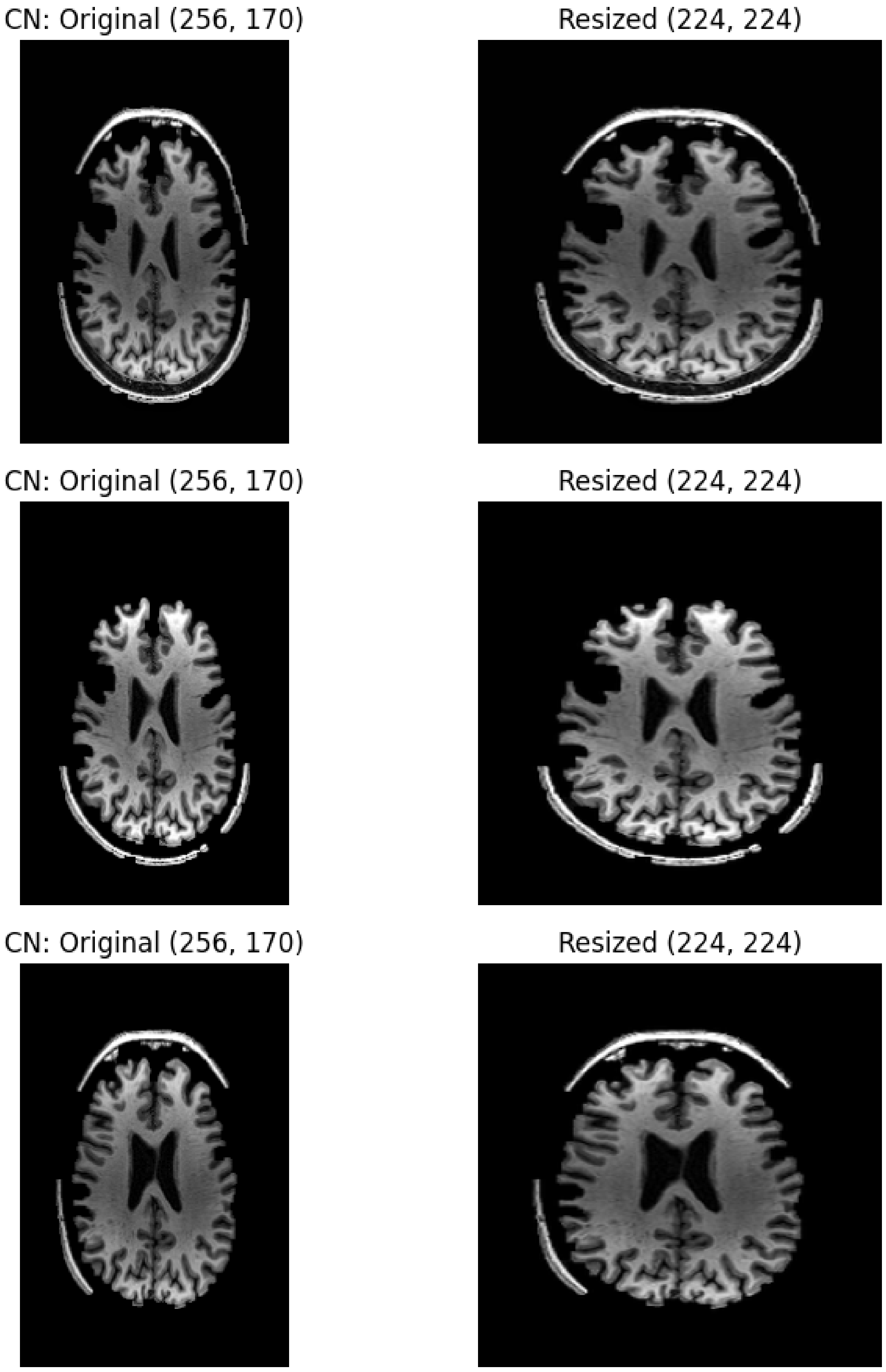

AD: Original (256, 170)
Resized (224, 224)
AD: Original (256, 170)
Resized (224, 224)
AD: Original (256, 170)
Resized (224, 224)

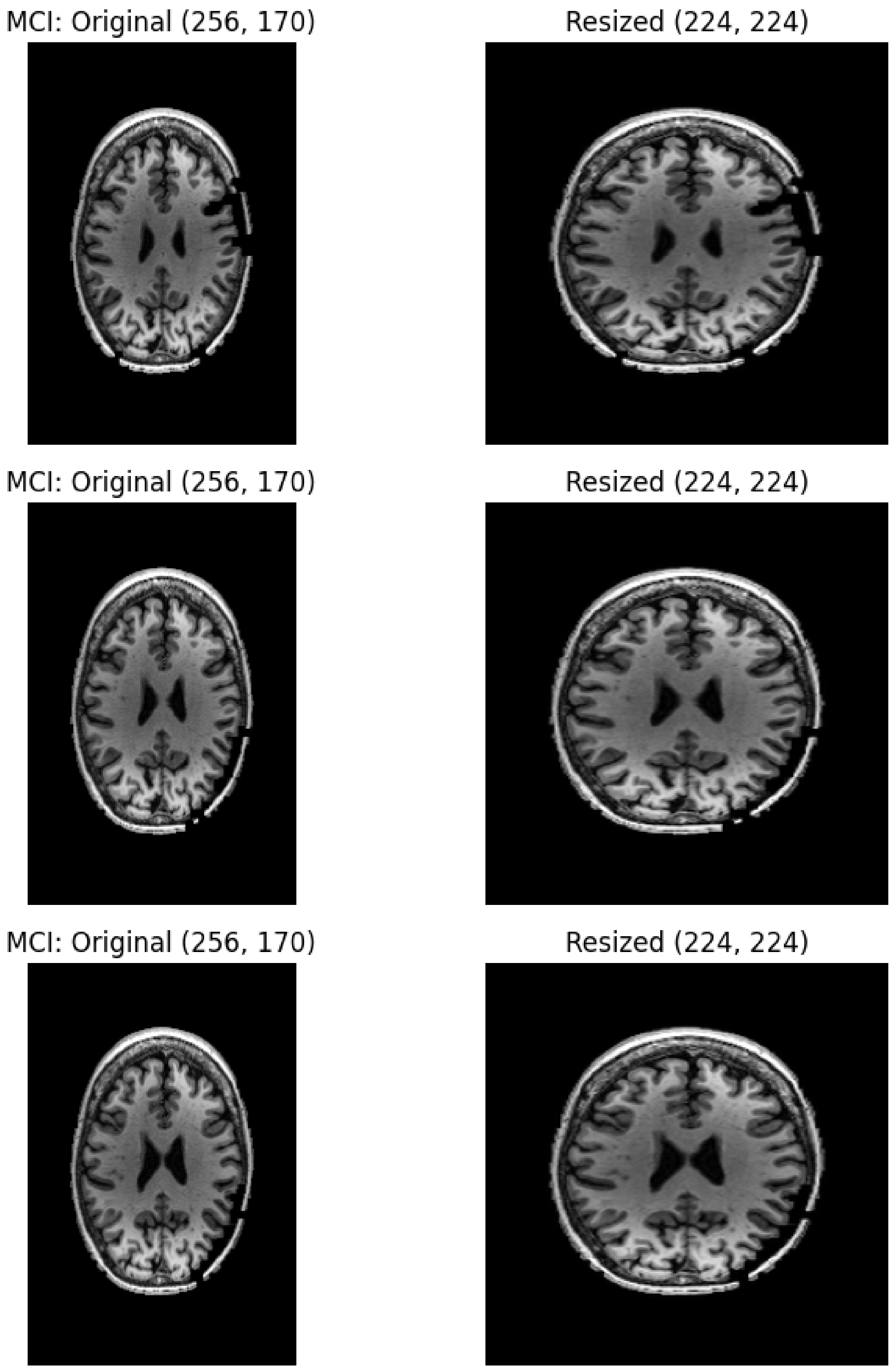

CN: Original (256, 170)
Resized (224, 224)
CN: Original (256, 170)
Resized (224, 224)
CN: Original (256, 170)
Resized (224, 224)


MCI: Original (256, 170)
Resized (224, 224)
MCI: Original (256, 170)
Resized (224, 224)
MCI: Original (256, 170)
Resized (224, 224)

```
=== Resizing Preprocessed Images: TRAIN ===
```


AD: Original (256, 170)
Resized (224, 224)
AD: Original (256, 170)
Resized (224, 224)
AD: Original (256, 170)
Resized (224, 224)


CN: Original (256, 170)
Resized (224, 224)
CN: Original (256, 170)
Resized (224, 224)
CN: Original (256, 170)
Resized (224, 224)


MCI: Original (256, 170)
Resized (224, 224)
MCI: Original (256, 170)
Resized (224, 224)
MCI: Original (256, 170)
Resized (224, 224)

```
Resized images saved to:
  • Resized Test : /kaggle/working/alzheimer-resized-224/test
  • Resized Train: /kaggle/working/alzheimer-resized-224/train
Summary TEST : AD: 225 ok / 0 skipped, CN: 288 ok / 0 skipped, MCI: 518 ok / 0
skipped
Summary TRAIN: AD: 899 ok / 0 skipped, CN: 1152 ok / 0 skipped, MCI: 2072 ok / 0
skipped
```

Train And Test Split

[4]:
```python
import os
import sys
import csv
import random
import shutil
from pathlib import Path
from typing import Dict, List, Tuple

# -----------------------------------------
# Configuration (modified for resized images)
# -----------------------------------------
# Source resized dataset (224x224)
RESIZED_ROOT = os.environ.get("RESIZED_ROOT", "/kaggle/working/
 ↪alzheimer-resized-224")
SRC_TRAIN = os.path.join(RESIZED_ROOT, "train")
SRC_TEST  = os.path.join(RESIZED_ROOT, "test")

# Classes (same as before)
CLASSES = ["AD", "CN", "MCI"]

# Where to write splits
SPLITS_ROOT = "/kaggle/working/alzheimer-resized-224_splits"
os.makedirs(SPLITS_ROOT, exist_ok=True)

# The split ratios (train:test) you requested
RATIO_LIST = [
    (0.90, 0.10),
    (0.80, 0.20),
    (0.70, 0.30),
    (0.60, 0.40),
    (0.50, 0.50),
    (0.40, 0.60),
    (0.30, 0.70),
    (0.20, 0.80),
    (0.10, 0.90)
]

# Validation share taken from the training portion
VAL_FRACTION = 0.10

# Base random seed (deterministic builds). Each ratio derives its own seed.
BASE_SEED = 2025

# -----------------------------------------
# Helpers
# -----------------------------------------
IMG_EXTS = (".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff")

def list_images_in_class(class_dir: str) -> List[str]:
    """Return absolute paths of all images in a class directory."""
    if not os.path.isdir(class_dir):
        return []
    files = sorted([
        str(Path(class_dir) / f) for f in os.listdir(class_dir)
        if f.lower().endswith(IMG_EXTS)
    ])
    return files

def gather_all_images() -> Dict[str, List[str]]:
    """
    Gather all resized images per class, from both 'train' and 'test'
    to form a single full pool for stratified splitting by ratio.
    """
    all_by_class = {c: [] for c in CLASSES}
    for c in CLASSES:
        # From resized train
        all_by_class[c].extend(list_images_in_class(os.path.join(SRC_TRAIN, c)))
        # From resized test
        all_by_class[c].extend(list_images_in_class(os.path.join(SRC_TEST, c)))
    return all_by_class

def ensure_dirs(*paths: str):
    for p in paths:
        os.makedirs(p, exist_ok=True)

def link_or_copy(src: str, dst: str):
    """Create a symlink; if not permitted, copy the file."""
    try:
        # Remove dst if exists
        if os.path.lexists(dst):
            os.unlink(dst)
        os.symlink(src, dst)
    except OSError:
        shutil.copy2(src, dst)

def write_manifest(split_dir: str, split_name: str, rows: List[Tuple[str, str,␣
↪str, str]]):
    """
    Write both CSV manifest and plain path list.
    rows: list of (split, cls, filename, src_path)
    """
    # CSV
    csv_path = os.path.join(split_dir, f"{split_name}.csv")
    with open(csv_path, "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["split", "class", "filename", "path"])
        for r in rows:
            w.writerow(r)

    # TXT list
    txt_path = os.path.join(split_dir, f"{split_name}.txt")
    with open(txt_path, "w") as f:
        for _, _, _, p in rows:
            f.write(p + "\n")

def summarize_counts(counts: Dict[str, Dict[str, int]]):
    """
    Print counts per split (train/val/test) and per class for quick sanity␣
↪check.
    """
    for split in ["train", "val", "test"]:
        info = counts.get(split, {})
        total = sum(info.values())
        detail = ", ".join([f"{k}: {v}" for k, v in info.items()])
        print(f"{split.capitalize():5s} => total {total:5d} | {detail}")

# ------------------------------------------
# Split builder
# ------------------------------------------
def build_splits():
    # 1) Pool all images by class (union of resized/train and resized/test)
    all_by_class = gather_all_images()

    # Optional: quick report of total availability
    print("Total images by class (union of resized/train + resized/test):")
    for c in CLASSES:
        print(f"  {c}: {len(all_by_class[c])}")

    # 2) For each (train_ratio, test_ratio), create a split folder and populate
    for tr_ratio, te_ratio in RATIO_LIST:
        # sanity: ratios sum approx 1
        assert abs(tr_ratio + te_ratio - 1.0) < 1e-6, "Train+Test ratio must␣
↪sum to 1"

        # deterministic seed per ratio
        seed = BASE_SEED + int(round(te_ratio * 100))
        rng = random.Random(seed)

        # Split name and dirs
        split_name =␣
↪f"split_{int(round(tr_ratio*100))}_{int(round(te_ratio*100))}"
        split_root = os.path.join(SPLITS_ROOT, split_name)
        train_root = os.path.join(split_root, "train")
        val_root   = os.path.join(split_root, "val")
        test_root  = os.path.join(split_root, "test")
        ensure_dirs(split_root, train_root, val_root, test_root)
        for c in CLASSES:
            ensure_dirs(os.path.join(train_root, c), os.path.join(val_root, c),␣
↪os.path.join(test_root, c))

        rows_train, rows_val, rows_test = [], [], []
        counts = {"train": {}, "val": {}, "test": {}}

        # 3) Per-class stratified splitting
        for c in CLASSES:
            full_list = list(all_by_class[c])  # copy
            rng.shuffle(full_list)             # deterministic shuffle

            n_total = len(full_list)
            n_test  = max(0, int(round(n_total * te_ratio)))
            n_test  = min(n_test, n_total)   # guard

            test_list = full_list[:n_test]
            train_pool = full_list[n_test:]

            # Validation from training portion (10%)
            n_val = max(0, int(round(len(train_pool) * VAL_FRACTION)))
            val_list = train_pool[:n_val]
            train_list = train_pool[n_val:]

            # 4) Materialize (symlink/copy) into folders and write manifests
            # test
            for src_path in test_list:
                fname = os.path.basename(src_path)
                dst_path = os.path.join(test_root, c, fname)
                link_or_copy(src_path, dst_path)
                rows_test.append(("test", c, fname, dst_path))

            # val
            for src_path in val_list:
                fname = os.path.basename(src_path)
                dst_path = os.path.join(val_root, c, fname)
                link_or_copy(src_path, dst_path)
                rows_val.append(("val", c, fname, dst_path))

            # train
            for src_path in train_list:
                fname = os.path.basename(src_path)
                dst_path = os.path.join(train_root, c, fname)
                link_or_copy(src_path, dst_path)
                rows_train.append(("train", c, fname, dst_path))

            # counts
            counts["test"][c]  = len(test_list)
            counts["val"][c]   = len(val_list)
            counts["train"][c] = len(train_list)

        # 5) Write manifest files for this split
        write_manifest(split_root, "train", rows_train)
        write_manifest(split_root, "val",   rows_val)
        write_manifest(split_root, "test",  rows_test)

        # 6) Summary printout
        print(f"\n=== {split_name} ===")
        summarize_counts(counts)
        print(f"Paths:\n  Train: {train_root}\n  Val  : {val_root}\n  Test :␣
 ↪{test_root}\n")

    # 7) Export env var for convenient access in later cells
    os.environ["RESIZED_SPLITS_ROOT"] = SPLITS_ROOT
    print(f"All splits created under: {SPLITS_ROOT}")

# ----------------------------------------
# Execute
# ----------------------------------------
build_splits()
```

```
Total images by class (union of resized/train + resized/test):
  AD: 1124
  CN: 1440
  MCI: 2590

=== split_90_10 ===
Train => total  4175 | AD: 911, CN: 1166, MCI: 2098
Val   => total   464 | AD: 101, CN: 130, MCI: 233
Test  => total   515 | AD: 112, CN: 144, MCI: 259
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_90_10/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_90_10/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_90_10/test


=== split_80_20 ===
Train => total  3711 | AD: 809, CN: 1037, MCI: 1865
Val   => total   412 | AD: 90, CN: 115, MCI: 207
Test  => total  1031 | AD: 225, CN: 288, MCI: 518
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_80_20/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_80_20/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_80_20/test


=== split_70_30 ===
Train => total  3247 | AD: 708, CN: 907, MCI: 1632
Val   => total   361 | AD: 79, CN: 101, MCI: 181
Test  => total  1546 | AD: 337, CN: 432, MCI: 777
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_70_30/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_70_30/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_70_30/test


=== split_60_40 ===
Train => total  2784 | AD: 607, CN: 778, MCI: 1399
Val   => total   308 | AD: 67, CN: 86, MCI: 155
Test  => total  2062 | AD: 450, CN: 576, MCI: 1036
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_60_40/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_60_40/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_60_40/test


=== split_50_50 ===
Train => total  2319 | AD: 506, CN: 648, MCI: 1165
Val   => total   258 | AD: 56, CN: 72, MCI: 130
Test  => total  2577 | AD: 562, CN: 720, MCI: 1295
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_50_50/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_50_50/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_50_50/test
```

```
=== split_40_60 ===
Train => total  1855 | AD: 405, CN: 518, MCI: 932
Val   => total   207 | AD: 45, CN: 58, MCI: 104
Test  => total  3092 | AD: 674, CN: 864, MCI: 1554
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_40_60/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_40_60/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_40_60/test


=== split_30_70 ===
Train => total  1391 | AD: 303, CN: 389, MCI: 699
Val   => total   155 | AD: 34, CN: 43, MCI: 78
Test  => total  3608 | AD: 787, CN: 1008, MCI: 1813
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_30_70/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_30_70/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_30_70/test


=== split_20_80 ===
Train => total   928 | AD: 203, CN: 259, MCI: 466
Val   => total   103 | AD: 22, CN: 29, MCI: 52
Test  => total  4123 | AD: 899, CN: 1152, MCI: 2072
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_20_80/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_20_80/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_20_80/test


=== split_10_90 ===
Train => total   464 | AD: 101, CN: 130, MCI: 233
Val   => total    51 | AD: 11, CN: 14, MCI: 26
Test  => total  4639 | AD: 1012, CN: 1296, MCI: 2331
Paths:
  Train: /kaggle/working/alzheimer-resized-224_splits/split_10_90/train
  Val  : /kaggle/working/alzheimer-resized-224_splits/split_10_90/val
  Test : /kaggle/working/alzheimer-resized-224_splits/split_10_90/test

All splits created under: /kaggle/working/alzheimer-resized-224_splits
```

[5]:
```python
import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models
```

```python
import pandas as pd
import numpy as np
from PIL import Image
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪accuracy_score, classification_report
import time
from torch.optim.lr_scheduler import CosineAnnealingLR, ReduceLROnPlateau

class AlzheimerDataset(Dataset):
    def __init__(self, split_dir, split_type, transform=None):
        self.split_dir = split_dir
        self.split_type = split_type
        self.transform = transform

        csv_path = os.path.join(split_dir, f"{split_type}.csv")
        self.df = pd.read_csv(csv_path)
        self.class_to_idx = {"AD": 0, "CN": 1, "MCI": 2}
        self.idx_to_class = {v: k for k, v in self.class_to_idx.items()}

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

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

        image = Image.open(img_path).convert('RGB')

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

        return image, label

def get_data_transforms():
    train_transform = transforms.Compose([
        transforms.Resize((380, 380)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(10),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,␣
 ↪hue=0.1),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
        transforms.RandomGrayscale(p=0.1),
        transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225]),
        transforms.RandomErasing(p=0.2, scale=(0.02, 0.2), ratio=(0.3, 3.3))
    ])

    val_transform = transforms.Compose([
        transforms.Resize((380, 380)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225])
    ])

    return train_transform, val_transform

def train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs, device, warmup_epochs=3):
    best_val_acc = 0
    patience = 8
    patience_counter = 0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        # Learning rate warmup
        if epoch < warmup_epochs:
            lr_scale = min(1.0, float(epoch + 1) / warmup_epochs)
            for param_group in optimizer.param_groups:
                param_group['lr'] = param_group['initial_lr'] * lr_scale

        for batch_idx, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()

            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_loss = running_loss / len(train_loader)
        epoch_acc = 100 * correct / total

        val_acc = evaluate_model(model, val_loader, device)

        if scheduler and epoch >= warmup_epochs:
            if isinstance(scheduler, CosineAnnealingLR):
                scheduler.step()
            else:
                scheduler.step(val_acc)

        current_lr = optimizer.param_groups[0]['lr']
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Train␣
↪Acc: {epoch_acc:.2f}%, Val Acc: {val_acc:.2f}%, LR: {current_lr:.6f}')

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping at epoch {epoch+1}")
            break

def evaluate_model(model, data_loader, device):
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in data_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return accuracy

def test_model(model, test_loader, device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return all_preds, all_labels

def calculate_metrics(y_true, y_pred, split_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted',␣
↪zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)

    print(f"\n=== {split_name} Results ===")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=['AD', 'CN',␣
↪'MCI'], zero_division=0))

    return {
        'split': split_name,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1
    }

def run_efficientnet_b4_on_splits():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device: {device}")

    splits_root = "/kaggle/working/alzheimer-resized-224_splits"
    split_folders = [f for f in os.listdir(splits_root) if f.
↪startswith('split_')]
    split_folders.sort()

    results = []

    train_transform, val_transform = get_data_transforms()

    for split_folder in split_folders:
        print(f"\n{'='*60}")
        print(f"Processing: {split_folder}")
        print(f"{'='*60}")

        split_path = os.path.join(splits_root, split_folder)

        train_dataset = AlzheimerDataset(split_path, 'train', train_transform)
        val_dataset = AlzheimerDataset(split_path, 'val', val_transform)
        test_dataset = AlzheimerDataset(split_path, 'test', val_transform)

        train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True,␣
↪num_workers=2, pin_memory=True)
        val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False,␣
↪num_workers=2, pin_memory=True)
        test_loader = DataLoader(test_dataset, batch_size=8, shuffle=False,␣
↪num_workers=2, pin_memory=True)

        print(f"Train samples: {len(train_dataset)}")
        print(f"Val samples: {len(val_dataset)}")
        print(f"Test samples: {len(test_dataset)}")

        model = models.efficientnet_b4(weights=models.EfficientNet_B4_Weights.
↪IMAGENET1K_V1)

        # Freeze all layers initially
        for param in model.parameters():
            param.requires_grad = False

        # Unfreeze classifier
        num_ftrs = model.classifier[1].in_features
        model.classifier[1] = nn.Linear(num_ftrs, 3)

        # Unfreeze last blocks (blocks 6-7) for feature adaptation
        for param in model.features[6:].parameters():
            param.requires_grad = True
        for param in model.features[5][-2:].parameters():
            param.requires_grad = True

        model = model.to(device)

        # Layer-wise learning rates for EfficientNet
        optimizer = torch.optim.AdamW([
            {'params': model.classifier.parameters(), 'lr': 0.0001,␣
↪'initial_lr': 0.0001},
            {'params': model.features[7].parameters(), 'lr': 0.00005,␣
↪'initial_lr': 0.00005},
            {'params': model.features[6].parameters(), 'lr': 0.00003,␣
↪'initial_lr': 0.00003},
            {'params': model.features[5][-2:].parameters(), 'lr': 0.00001,␣
↪'initial_lr': 0.00001},
        ], weight_decay=1e-5)

        criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
        scheduler = CosineAnnealingLR(optimizer, T_max=15, eta_min=1e-6)

        print("Starting phase 1 training (partial unfreeze)...")
        start_time = time.time()
        train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs=18, device=device)
        phase1_time = time.time() - start_time

        print("Starting phase 2 training (full fine-tuning)...")
        for param in model.parameters():
            param.requires_grad = True

        optimizer = torch.optim.AdamW(model.parameters(), lr=0.000005,␣
↪weight_decay=1e-6)
        scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.5,␣
↪patience=4, verbose=True)

        start_time_phase2 = time.time()
        train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs=12, device=device, warmup_epochs=0)
        total_time = phase1_time + (time.time() - start_time_phase2)

        print("Testing model...")
        test_preds, test_labels = test_model(model, test_loader, device)

        split_results = calculate_metrics(test_labels, test_preds, split_folder)
        split_results['training_time'] = total_time
        results.append(split_results)

        torch.cuda.empty_cache()

    results_df = pd.DataFrame(results)
    print(f"\n{'='*80}")
    print("EfficientNet-B4 - SUMMARY OF ALL SPLITS")
    print(f"{'='*80}")
    print(results_df.to_string(index=False))

    results_csv_path = "/kaggle/working/efficientnet_b4_results.csv"
    results_df.to_csv(results_csv_path, index=False)
    print(f"\nDetailed results saved to: {results_csv_path}")

    return results_df

if __name__ == "__main__":
    results = run_efficientnet_b4_on_splits()
```

```
Using device: cuda

============================================================
Processing: split_10_90
============================================================
Train samples: 464
Val samples: 51
Test samples: 4639

Downloading:
"https://download.pytorch.org/models/efficientnet_b4_rwightman-23ab8bcd.pth" to
/root/.cache/torch/hub/checkpoints/efficientnet_b4_rwightman-23ab8bcd.pth
100%|     | 74.5M/74.5M [00:00<00:00, 184MB/s]

Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0975, Train Acc: 33.84%, Val Acc: 33.33%, LR: 0.000033
Epoch [2/18], Loss: 1.0774, Train Acc: 44.61%, Val Acc: 50.98%, LR: 0.000067
Epoch [3/18], Loss: 1.0545, Train Acc: 49.78%, Val Acc: 50.98%, LR: 0.000100
Epoch [4/18], Loss: 1.0435, Train Acc: 50.43%, Val Acc: 50.98%, LR: 0.000099
Epoch [5/18], Loss: 1.0455, Train Acc: 50.22%, Val Acc: 50.98%, LR: 0.000096
Epoch [6/18], Loss: 1.0326, Train Acc: 50.00%, Val Acc: 50.98%, LR: 0.000091
Epoch [7/18], Loss: 1.0308, Train Acc: 50.43%, Val Acc: 50.98%, LR: 0.000084
Epoch [8/18], Loss: 1.0319, Train Acc: 50.22%, Val Acc: 50.98%, LR: 0.000075
Epoch [9/18], Loss: 1.0239, Train Acc: 50.43%, Val Acc: 50.98%, LR: 0.000066
Epoch [10/18], Loss: 1.0302, Train Acc: 49.57%, Val Acc: 50.98%, LR: 0.000056
Early stopping at epoch 10
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 1.0167, Train Acc: 50.65%, Val Acc: 50.98%, LR: 0.000005
Epoch [2/12], Loss: 1.0248, Train Acc: 50.00%, Val Acc: 50.98%, LR: 0.000005
Epoch [3/12], Loss: 1.0250, Train Acc: 51.29%, Val Acc: 50.98%, LR: 0.000005
Epoch [4/12], Loss: 1.0238, Train Acc: 50.86%, Val Acc: 50.98%, LR: 0.000005
Epoch [5/12], Loss: 1.0152, Train Acc: 51.29%, Val Acc: 50.98%, LR: 0.000005
Epoch [6/12], Loss: 1.0222, Train Acc: 50.86%, Val Acc: 50.98%, LR: 0.000003
Epoch [7/12], Loss: 1.0234, Train Acc: 50.00%, Val Acc: 50.98%, LR: 0.000003
Epoch [8/12], Loss: 1.0161, Train Acc: 51.94%, Val Acc: 50.98%, LR: 0.000003
Epoch [9/12], Loss: 1.0193, Train Acc: 51.08%, Val Acc: 50.98%, LR: 0.000003
```

```
Early stopping at epoch 9
Testing model…

=== split_10_90 Results ===
Accuracy: 0.5031
Precision: 0.4124
Recall: 0.5031
F1-Score: 0.3380

Classification Report:
              precision    recall  f1-score   support

          AD       0.00      0.00      0.00      1012
          CN       0.57      0.00      0.01      1296
         MCI       0.50      1.00      0.67      2331

    accuracy                           0.50      4639
   macro avg       0.36      0.33      0.23      4639
weighted avg       0.41      0.50      0.34      4639


============================================================
Processing: split_20_80
============================================================
Train samples: 928
Val samples: 103
Test samples: 4123
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0934, Train Acc: 39.55%, Val Acc: 47.57%, LR: 0.000033
Epoch [2/18], Loss: 1.0604, Train Acc: 49.78%, Val Acc: 50.49%, LR: 0.000067
Epoch [3/18], Loss: 1.0485, Train Acc: 50.22%, Val Acc: 50.49%, LR: 0.000100
Epoch [4/18], Loss: 1.0394, Train Acc: 50.32%, Val Acc: 50.49%, LR: 0.000099
Epoch [5/18], Loss: 1.0389, Train Acc: 50.32%, Val Acc: 50.49%, LR: 0.000096
Epoch [6/18], Loss: 1.0383, Train Acc: 50.32%, Val Acc: 50.49%, LR: 0.000091
Epoch [7/18], Loss: 1.0330, Train Acc: 50.22%, Val Acc: 50.49%, LR: 0.000084
Epoch [8/18], Loss: 1.0290, Train Acc: 50.22%, Val Acc: 50.49%, LR: 0.000075
Epoch [9/18], Loss: 1.0279, Train Acc: 50.75%, Val Acc: 50.49%, LR: 0.000066
Epoch [10/18], Loss: 1.0264, Train Acc: 50.75%, Val Acc: 50.49%, LR: 0.000056
Early stopping at epoch 10
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 1.0195, Train Acc: 50.32%, Val Acc: 50.49%, LR: 0.000005
Epoch [2/12], Loss: 1.0190, Train Acc: 50.54%, Val Acc: 50.49%, LR: 0.000005
Epoch [3/12], Loss: 1.0147, Train Acc: 50.54%, Val Acc: 50.49%, LR: 0.000005
Epoch [4/12], Loss: 1.0152, Train Acc: 50.86%, Val Acc: 50.49%, LR: 0.000005
Epoch [5/12], Loss: 1.0139, Train Acc: 50.75%, Val Acc: 50.49%, LR: 0.000005
Epoch [6/12], Loss: 1.0204, Train Acc: 50.54%, Val Acc: 50.49%, LR: 0.000003
Epoch [7/12], Loss: 1.0164, Train Acc: 51.29%, Val Acc: 50.49%, LR: 0.000003
Epoch [8/12], Loss: 1.0176, Train Acc: 50.32%, Val Acc: 50.49%, LR: 0.000003
Epoch [9/12], Loss: 1.0092, Train Acc: 50.75%, Val Acc: 50.49%, LR: 0.000003
Early stopping at epoch 9
Testing model…

=== split_20_80 Results ===
Accuracy: 0.5035
Precision: 0.4764
Recall: 0.5035
F1-Score: 0.3384

Classification Report:
              precision    recall  f1-score   support

          AD       0.00      0.00      0.00       899
          CN       0.80      0.00      0.01      1152
         MCI       0.50      1.00      0.67      2072

    accuracy                           0.50      4123
   macro avg       0.43      0.33      0.23      4123
weighted avg       0.48      0.50      0.34      4123


============================================================
Processing: split_30_70
============================================================
Train samples: 1391
Val samples: 155
Test samples: 3608
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0764, Train Acc: 47.09%, Val Acc: 50.32%, LR: 0.000033
Epoch [2/18], Loss: 1.0476, Train Acc: 50.11%, Val Acc: 50.32%, LR: 0.000067
Epoch [3/18], Loss: 1.0421, Train Acc: 50.32%, Val Acc: 50.32%, LR: 0.000100
Epoch [4/18], Loss: 1.0389, Train Acc: 50.61%, Val Acc: 50.32%, LR: 0.000099
Epoch [5/18], Loss: 1.0331, Train Acc: 50.40%, Val Acc: 50.32%, LR: 0.000096
Epoch [6/18], Loss: 1.0227, Train Acc: 50.47%, Val Acc: 50.97%, LR: 0.000091
Epoch [7/18], Loss: 1.0179, Train Acc: 52.34%, Val Acc: 52.26%, LR: 0.000084
Epoch [8/18], Loss: 1.0135, Train Acc: 51.83%, Val Acc: 52.90%, LR: 0.000075
Epoch [9/18], Loss: 1.0093, Train Acc: 52.05%, Val Acc: 52.26%, LR: 0.000066
Epoch [10/18], Loss: 1.0031, Train Acc: 52.70%, Val Acc: 52.26%, LR: 0.000056
Epoch [11/18], Loss: 0.9939, Train Acc: 53.06%, Val Acc: 53.55%, LR: 0.000045
Epoch [12/18], Loss: 0.9945, Train Acc: 52.19%, Val Acc: 53.55%, LR: 0.000035
Epoch [13/18], Loss: 0.9808, Train Acc: 53.99%, Val Acc: 54.84%, LR: 0.000026
Epoch [14/18], Loss: 0.9871, Train Acc: 53.41%, Val Acc: 52.90%, LR: 0.000017
Epoch [15/18], Loss: 0.9827, Train Acc: 53.41%, Val Acc: 52.26%, LR: 0.000010
Epoch [16/18], Loss: 0.9775, Train Acc: 54.78%, Val Acc: 56.77%, LR: 0.000005
Epoch [17/18], Loss: 0.9773, Train Acc: 54.35%, Val Acc: 57.42%, LR: 0.000002
Epoch [18/18], Loss: 0.9769, Train Acc: 54.78%, Val Acc: 56.13%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.9749, Train Acc: 54.92%, Val Acc: 56.13%, LR: 0.000005
Epoch [2/12], Loss: 0.9678, Train Acc: 55.28%, Val Acc: 56.13%, LR: 0.000005
Epoch [3/12], Loss: 0.9658, Train Acc: 55.14%, Val Acc: 59.35%, LR: 0.000005
Epoch [4/12], Loss: 0.9638, Train Acc: 56.29%, Val Acc: 58.71%, LR: 0.000005
Epoch [5/12], Loss: 0.9682, Train Acc: 55.36%, Val Acc: 59.35%, LR: 0.000005
Epoch [6/12], Loss: 0.9550, Train Acc: 56.58%, Val Acc: 60.00%, LR: 0.000005
Epoch [7/12], Loss: 0.9602, Train Acc: 54.57%, Val Acc: 61.29%, LR: 0.000005
Epoch [8/12], Loss: 0.9549, Train Acc: 56.22%, Val Acc: 63.23%, LR: 0.000005
Epoch [9/12], Loss: 0.9552, Train Acc: 55.86%, Val Acc: 63.23%, LR: 0.000005
Epoch [10/12], Loss: 0.9527, Train Acc: 55.64%, Val Acc: 61.94%, LR: 0.000005
Epoch [11/12], Loss: 0.9464, Train Acc: 56.00%, Val Acc: 61.94%, LR: 0.000005
Epoch [12/12], Loss: 0.9458, Train Acc: 58.23%, Val Acc: 62.58%, LR: 0.000005
Testing model…

=== split_30_70 Results ===
Accuracy: 0.5654
Precision: 0.5878
Recall: 0.5654
F1-Score: 0.4849

Classification Report:
              precision    recall  f1-score   support

          AD       0.59      0.10      0.17       787
          CN       0.65      0.24      0.34      1008
         MCI       0.55      0.95      0.70      1813

    accuracy                           0.57      3608
   macro avg       0.60      0.43      0.40      3608
weighted avg       0.59      0.57      0.48      3608


============================================================
Processing: split_40_60
============================================================
Train samples: 1855
Val samples: 207
Test samples: 3092
```

```
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0797, Train Acc: 44.47%, Val Acc: 50.24%, LR: 0.000033
Epoch [2/18], Loss: 1.0486, Train Acc: 50.30%, Val Acc: 50.24%, LR: 0.000067
Epoch [3/18], Loss: 1.0409, Train Acc: 50.35%, Val Acc: 50.24%, LR: 0.000100
Epoch [4/18], Loss: 1.0289, Train Acc: 50.94%, Val Acc: 52.17%, LR: 0.000099
Epoch [5/18], Loss: 1.0250, Train Acc: 51.64%, Val Acc: 53.62%, LR: 0.000096
Epoch [6/18], Loss: 1.0100, Train Acc: 52.61%, Val Acc: 52.66%, LR: 0.000091
Epoch [7/18], Loss: 0.9961, Train Acc: 52.88%, Val Acc: 53.14%, LR: 0.000084
Epoch [8/18], Loss: 0.9873, Train Acc: 53.64%, Val Acc: 53.14%, LR: 0.000075
Epoch [9/18], Loss: 0.9715, Train Acc: 55.63%, Val Acc: 56.04%, LR: 0.000066
Epoch [10/18], Loss: 0.9639, Train Acc: 57.04%, Val Acc: 56.52%, LR: 0.000056
Epoch [11/18], Loss: 0.9465, Train Acc: 58.76%, Val Acc: 57.97%, LR: 0.000045
Epoch [12/18], Loss: 0.9362, Train Acc: 58.87%, Val Acc: 59.90%, LR: 0.000035
Epoch [13/18], Loss: 0.9340, Train Acc: 58.38%, Val Acc: 56.52%, LR: 0.000026
Epoch [14/18], Loss: 0.9121, Train Acc: 61.29%, Val Acc: 56.52%, LR: 0.000017
Epoch [15/18], Loss: 0.9165, Train Acc: 60.49%, Val Acc: 57.97%, LR: 0.000010
Epoch [16/18], Loss: 0.9067, Train Acc: 61.83%, Val Acc: 60.39%, LR: 0.000005
Epoch [17/18], Loss: 0.9059, Train Acc: 60.86%, Val Acc: 61.35%, LR: 0.000002
Epoch [18/18], Loss: 0.9080, Train Acc: 60.49%, Val Acc: 58.45%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8994, Train Acc: 63.40%, Val Acc: 60.87%, LR: 0.000005
Epoch [2/12], Loss: 0.9134, Train Acc: 61.40%, Val Acc: 57.97%, LR: 0.000005
Epoch [3/12], Loss: 0.9025, Train Acc: 62.91%, Val Acc: 59.42%, LR: 0.000005
Epoch [4/12], Loss: 0.8985, Train Acc: 62.05%, Val Acc: 59.90%, LR: 0.000005
Epoch [5/12], Loss: 0.9022, Train Acc: 62.32%, Val Acc: 60.87%, LR: 0.000005
Epoch [6/12], Loss: 0.9060, Train Acc: 62.21%, Val Acc: 59.90%, LR: 0.000003
Epoch [7/12], Loss: 0.8898, Train Acc: 63.61%, Val Acc: 58.45%, LR: 0.000003
Epoch [8/12], Loss: 0.8981, Train Acc: 61.67%, Val Acc: 59.90%, LR: 0.000003
Epoch [9/12], Loss: 0.8890, Train Acc: 63.02%, Val Acc: 60.87%, LR: 0.000003
Early stopping at epoch 9
Testing model…

=== split_40_60 Results ===
Accuracy: 0.6239
Precision: 0.6387
Recall: 0.6239
F1-Score: 0.5858

Classification Report:
              precision    recall  f1-score   support

          AD       0.74      0.19      0.30       674
          CN       0.57      0.52      0.54       864
```

```
         MCI       0.64      0.87      0.74      1554

    accuracy                           0.62      3092
   macro avg       0.65      0.53      0.52      3092
weighted avg       0.64      0.62      0.59      3092


============================================================
Processing: split_50_50
============================================================
Train samples: 2319
Val samples: 258
Test samples: 2577
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0744, Train Acc: 46.44%, Val Acc: 50.39%, LR: 0.000033
Epoch [2/18], Loss: 1.0458, Train Acc: 50.24%, Val Acc: 50.39%, LR: 0.000067
Epoch [3/18], Loss: 1.0324, Train Acc: 50.24%, Val Acc: 50.39%, LR: 0.000100
Epoch [4/18], Loss: 1.0265, Train Acc: 50.67%, Val Acc: 50.39%, LR: 0.000099
Epoch [5/18], Loss: 1.0176, Train Acc: 50.84%, Val Acc: 53.88%, LR: 0.000096
Epoch [6/18], Loss: 1.0034, Train Acc: 53.39%, Val Acc: 53.88%, LR: 0.000091
Epoch [7/18], Loss: 0.9831, Train Acc: 55.02%, Val Acc: 54.26%, LR: 0.000084
Epoch [8/18], Loss: 0.9658, Train Acc: 56.92%, Val Acc: 57.36%, LR: 0.000075
Epoch [9/18], Loss: 0.9356, Train Acc: 58.86%, Val Acc: 55.04%, LR: 0.000066
Epoch [10/18], Loss: 0.9329, Train Acc: 59.29%, Val Acc: 56.59%, LR: 0.000056
Epoch [11/18], Loss: 0.9146, Train Acc: 61.41%, Val Acc: 60.47%, LR: 0.000045
Epoch [12/18], Loss: 0.9034, Train Acc: 62.01%, Val Acc: 62.40%, LR: 0.000035
Epoch [13/18], Loss: 0.8995, Train Acc: 62.35%, Val Acc: 65.50%, LR: 0.000026
Epoch [14/18], Loss: 0.8950, Train Acc: 62.23%, Val Acc: 64.34%, LR: 0.000017
Epoch [15/18], Loss: 0.8815, Train Acc: 64.04%, Val Acc: 63.18%, LR: 0.000010
Epoch [16/18], Loss: 0.8909, Train Acc: 61.92%, Val Acc: 65.12%, LR: 0.000005
Epoch [17/18], Loss: 0.8784, Train Acc: 64.12%, Val Acc: 64.34%, LR: 0.000002
Epoch [18/18], Loss: 0.8814, Train Acc: 63.09%, Val Acc: 67.83%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8801, Train Acc: 62.83%, Val Acc: 64.73%, LR: 0.000005
Epoch [2/12], Loss: 0.8794, Train Acc: 63.04%, Val Acc: 64.34%, LR: 0.000005
Epoch [3/12], Loss: 0.8727, Train Acc: 62.70%, Val Acc: 65.50%, LR: 0.000005
Epoch [4/12], Loss: 0.8748, Train Acc: 64.08%, Val Acc: 67.83%, LR: 0.000005
Epoch [5/12], Loss: 0.8690, Train Acc: 64.42%, Val Acc: 69.38%, LR: 0.000005
Epoch [6/12], Loss: 0.8622, Train Acc: 64.98%, Val Acc: 69.38%, LR: 0.000005
Epoch [7/12], Loss: 0.8624, Train Acc: 65.33%, Val Acc: 68.60%, LR: 0.000005
Epoch [8/12], Loss: 0.8589, Train Acc: 64.90%, Val Acc: 67.83%, LR: 0.000005
Epoch [9/12], Loss: 0.8609, Train Acc: 65.24%, Val Acc: 68.22%, LR: 0.000005
Epoch [10/12], Loss: 0.8500, Train Acc: 64.98%, Val Acc: 67.05%, LR: 0.000003
Epoch [11/12], Loss: 0.8725, Train Acc: 64.21%, Val Acc: 67.44%, LR: 0.000003
Epoch [12/12], Loss: 0.8427, Train Acc: 67.31%, Val Acc: 70.54%, LR: 0.000003
Testing model…

=== split_50_50 Results ===
Accuracy: 0.6546
Precision: 0.6613
Recall: 0.6546
F1-Score: 0.6267

Classification Report:
              precision    recall  f1-score   support

          AD       0.70      0.31      0.43       562
          CN       0.65      0.47      0.55       720
         MCI       0.65      0.91      0.76      1295

    accuracy                           0.65      2577
   macro avg       0.67      0.56      0.58      2577
weighted avg       0.66      0.65      0.63      2577


============================================================
Processing: split_60_40
============================================================
Train samples: 2784
Val samples: 308
Test samples: 2062
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0683, Train Acc: 47.16%, Val Acc: 50.32%, LR: 0.000033
Epoch [2/18], Loss: 1.0404, Train Acc: 50.40%, Val Acc: 50.00%, LR: 0.000067
Epoch [3/18], Loss: 1.0331, Train Acc: 50.36%, Val Acc: 50.00%, LR: 0.000100
Epoch [4/18], Loss: 1.0201, Train Acc: 50.90%, Val Acc: 50.00%, LR: 0.000099
Epoch [5/18], Loss: 1.0122, Train Acc: 51.76%, Val Acc: 50.97%, LR: 0.000096
Epoch [6/18], Loss: 0.9950, Train Acc: 53.12%, Val Acc: 53.25%, LR: 0.000091
Epoch [7/18], Loss: 0.9775, Train Acc: 54.45%, Val Acc: 54.22%, LR: 0.000084
Epoch [8/18], Loss: 0.9535, Train Acc: 57.15%, Val Acc: 58.44%, LR: 0.000075
Epoch [9/18], Loss: 0.9353, Train Acc: 58.98%, Val Acc: 61.04%, LR: 0.000066
Epoch [10/18], Loss: 0.9212, Train Acc: 60.70%, Val Acc: 62.66%, LR: 0.000056
Epoch [11/18], Loss: 0.9014, Train Acc: 61.93%, Val Acc: 63.64%, LR: 0.000045
Epoch [12/18], Loss: 0.9095, Train Acc: 60.78%, Val Acc: 65.58%, LR: 0.000035
Epoch [13/18], Loss: 0.8870, Train Acc: 61.60%, Val Acc: 68.51%, LR: 0.000026
Epoch [14/18], Loss: 0.8795, Train Acc: 63.58%, Val Acc: 65.91%, LR: 0.000017
Epoch [15/18], Loss: 0.8831, Train Acc: 63.00%, Val Acc: 65.91%, LR: 0.000010
Epoch [16/18], Loss: 0.8798, Train Acc: 63.79%, Val Acc: 67.86%, LR: 0.000005
Epoch [17/18], Loss: 0.8861, Train Acc: 63.33%, Val Acc: 65.58%, LR: 0.000002
Epoch [18/18], Loss: 0.8630, Train Acc: 64.73%, Val Acc: 64.94%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…
```

```
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8718, Train Acc: 63.79%, Val Acc: 67.21%, LR: 0.000005
Epoch [2/12], Loss: 0.8690, Train Acc: 64.19%, Val Acc: 66.88%, LR: 0.000005
Epoch [3/12], Loss: 0.8549, Train Acc: 65.37%, Val Acc: 67.53%, LR: 0.000005
Epoch [4/12], Loss: 0.8663, Train Acc: 63.90%, Val Acc: 71.10%, LR: 0.000005
Epoch [5/12], Loss: 0.8625, Train Acc: 64.37%, Val Acc: 69.81%, LR: 0.000005
Epoch [6/12], Loss: 0.8556, Train Acc: 65.95%, Val Acc: 70.78%, LR: 0.000005
Epoch [7/12], Loss: 0.8582, Train Acc: 66.06%, Val Acc: 72.73%, LR: 0.000005
Epoch [8/12], Loss: 0.8441, Train Acc: 66.77%, Val Acc: 69.16%, LR: 0.000005
Epoch [9/12], Loss: 0.8324, Train Acc: 67.06%, Val Acc: 70.45%, LR: 0.000005
Epoch [10/12], Loss: 0.8455, Train Acc: 66.06%, Val Acc: 69.48%, LR: 0.000005
Epoch [11/12], Loss: 0.8322, Train Acc: 67.03%, Val Acc: 67.86%, LR: 0.000005
Epoch [12/12], Loss: 0.8299, Train Acc: 66.88%, Val Acc: 70.78%, LR: 0.000003
Testing model…

=== split_60_40 Results ===
Accuracy: 0.6848
Precision: 0.6808
Recall: 0.6848
F1-Score: 0.6704

Classification Report:
              precision    recall  f1-score   support

          AD       0.64      0.42      0.51       450
          CN       0.69      0.55      0.61       576
         MCI       0.69      0.87      0.77      1036

    accuracy                           0.68      2062
   macro avg       0.67      0.62      0.63      2062
weighted avg       0.68      0.68      0.67      2062


============================================================
Processing: split_70_30
============================================================
Train samples: 3247
Val samples: 361
Test samples: 1546
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0636, Train Acc: 48.01%, Val Acc: 50.14%, LR: 0.000033
Epoch [2/18], Loss: 1.0439, Train Acc: 50.32%, Val Acc: 50.14%, LR: 0.000067
Epoch [3/18], Loss: 1.0398, Train Acc: 50.20%, Val Acc: 50.14%, LR: 0.000100
Epoch [4/18], Loss: 1.0278, Train Acc: 50.79%, Val Acc: 50.14%, LR: 0.000099
Epoch [5/18], Loss: 1.0066, Train Acc: 51.59%, Val Acc: 52.35%, LR: 0.000096
Epoch [6/18], Loss: 0.9865, Train Acc: 54.17%, Val Acc: 57.89%, LR: 0.000091
Epoch [7/18], Loss: 0.9532, Train Acc: 56.76%, Val Acc: 58.17%, LR: 0.000084
Epoch [8/18], Loss: 0.9453, Train Acc: 57.13%, Val Acc: 60.94%, LR: 0.000075
Epoch [9/18], Loss: 0.9217, Train Acc: 60.33%, Val Acc: 63.99%, LR: 0.000066
Epoch [10/18], Loss: 0.9149, Train Acc: 61.04%, Val Acc: 66.20%, LR: 0.000056
Epoch [11/18], Loss: 0.8992, Train Acc: 61.60%, Val Acc: 64.82%, LR: 0.000045
Epoch [12/18], Loss: 0.8853, Train Acc: 62.49%, Val Acc: 67.31%, LR: 0.000035
Epoch [13/18], Loss: 0.8660, Train Acc: 63.60%, Val Acc: 66.20%, LR: 0.000026
Epoch [14/18], Loss: 0.8750, Train Acc: 63.66%, Val Acc: 67.59%, LR: 0.000017
Epoch [15/18], Loss: 0.8664, Train Acc: 64.64%, Val Acc: 67.59%, LR: 0.000010
Epoch [16/18], Loss: 0.8495, Train Acc: 65.29%, Val Acc: 67.59%, LR: 0.000005
Epoch [17/18], Loss: 0.8575, Train Acc: 65.26%, Val Acc: 67.04%, LR: 0.000002
Epoch [18/18], Loss: 0.8598, Train Acc: 65.60%, Val Acc: 66.76%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8516, Train Acc: 66.40%, Val Acc: 66.76%, LR: 0.000005
Epoch [2/12], Loss: 0.8378, Train Acc: 66.34%, Val Acc: 66.48%, LR: 0.000005
Epoch [3/12], Loss: 0.8473, Train Acc: 66.28%, Val Acc: 66.76%, LR: 0.000005
Epoch [4/12], Loss: 0.8387, Train Acc: 65.66%, Val Acc: 67.31%, LR: 0.000005
Epoch [5/12], Loss: 0.8481, Train Acc: 66.83%, Val Acc: 67.04%, LR: 0.000005
Epoch [6/12], Loss: 0.8397, Train Acc: 66.15%, Val Acc: 68.42%, LR: 0.000005
Epoch [7/12], Loss: 0.8273, Train Acc: 67.23%, Val Acc: 68.14%, LR: 0.000005
Epoch [8/12], Loss: 0.8343, Train Acc: 66.68%, Val Acc: 68.14%, LR: 0.000005
Epoch [9/12], Loss: 0.8207, Train Acc: 67.79%, Val Acc: 68.14%, LR: 0.000005
Epoch [10/12], Loss: 0.8247, Train Acc: 67.69%, Val Acc: 68.42%, LR: 0.000005
Epoch [11/12], Loss: 0.8221, Train Acc: 68.37%, Val Acc: 67.59%, LR: 0.000003
Epoch [12/12], Loss: 0.8088, Train Acc: 69.11%, Val Acc: 68.98%, LR: 0.000003
Testing model…

=== split_70_30 Results ===
Accuracy: 0.7160
Precision: 0.7100
Recall: 0.7160
F1-Score: 0.7059

Classification Report:
              precision    recall  f1-score   support

          AD       0.66      0.45      0.53       337
          CN       0.71      0.66      0.68       432
         MCI       0.74      0.86      0.79       777

    accuracy                           0.72      1546
```

```
   macro avg       0.70      0.66      0.67      1546
weighted avg       0.71      0.72      0.71      1546


============================================================
Processing: split_80_20
============================================================
Train samples: 3711
Val samples: 412
Test samples: 1031
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0625, Train Acc: 47.45%, Val Acc: 50.24%, LR: 0.000033
Epoch [2/18], Loss: 1.0434, Train Acc: 50.42%, Val Acc: 50.24%, LR: 0.000067
Epoch [3/18], Loss: 1.0309, Train Acc: 50.47%, Val Acc: 50.97%, LR: 0.000100
Epoch [4/18], Loss: 1.0209, Train Acc: 51.09%, Val Acc: 52.67%, LR: 0.000099
Epoch [5/18], Loss: 1.0002, Train Acc: 53.27%, Val Acc: 54.13%, LR: 0.000096
Epoch [6/18], Loss: 0.9680, Train Acc: 55.56%, Val Acc: 59.71%, LR: 0.000091
Epoch [7/18], Loss: 0.9493, Train Acc: 57.80%, Val Acc: 58.74%, LR: 0.000084
Epoch [8/18], Loss: 0.9205, Train Acc: 59.71%, Val Acc: 59.95%, LR: 0.000075
Epoch [9/18], Loss: 0.8889, Train Acc: 62.38%, Val Acc: 65.29%, LR: 0.000066
Epoch [10/18], Loss: 0.8732, Train Acc: 63.86%, Val Acc: 65.78%, LR: 0.000056
Epoch [11/18], Loss: 0.8732, Train Acc: 63.97%, Val Acc: 69.90%, LR: 0.000045
Epoch [12/18], Loss: 0.8527, Train Acc: 64.67%, Val Acc: 68.45%, LR: 0.000035
Epoch [13/18], Loss: 0.8482, Train Acc: 65.94%, Val Acc: 68.45%, LR: 0.000026
Epoch [14/18], Loss: 0.8382, Train Acc: 66.83%, Val Acc: 68.69%, LR: 0.000017
Epoch [15/18], Loss: 0.8374, Train Acc: 66.88%, Val Acc: 68.45%, LR: 0.000010
Epoch [16/18], Loss: 0.8316, Train Acc: 66.61%, Val Acc: 70.15%, LR: 0.000005
Epoch [17/18], Loss: 0.8325, Train Acc: 67.34%, Val Acc: 69.42%, LR: 0.000002
Epoch [18/18], Loss: 0.8248, Train Acc: 67.56%, Val Acc: 69.17%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8266, Train Acc: 67.80%, Val Acc: 72.09%, LR: 0.000005
Epoch [2/12], Loss: 0.8251, Train Acc: 66.91%, Val Acc: 71.36%, LR: 0.000005
Epoch [3/12], Loss: 0.8192, Train Acc: 67.48%, Val Acc: 71.60%, LR: 0.000005
Epoch [4/12], Loss: 0.8228, Train Acc: 67.91%, Val Acc: 70.39%, LR: 0.000005
Epoch [5/12], Loss: 0.8112, Train Acc: 68.96%, Val Acc: 70.87%, LR: 0.000005
Epoch [6/12], Loss: 0.7917, Train Acc: 69.82%, Val Acc: 71.12%, LR: 0.000003
Epoch [7/12], Loss: 0.7940, Train Acc: 69.44%, Val Acc: 72.33%, LR: 0.000003
Epoch [8/12], Loss: 0.7859, Train Acc: 70.22%, Val Acc: 71.36%, LR: 0.000003
Epoch [9/12], Loss: 0.8089, Train Acc: 67.74%, Val Acc: 71.60%, LR: 0.000003
Epoch [10/12], Loss: 0.7979, Train Acc: 69.31%, Val Acc: 72.09%, LR: 0.000003
Epoch [11/12], Loss: 0.7857, Train Acc: 69.60%, Val Acc: 72.09%, LR: 0.000003
Epoch [12/12], Loss: 0.7869, Train Acc: 70.52%, Val Acc: 71.84%, LR: 0.000001
Testing model…
```

```
=== split_80_20 Results ===
Accuracy: 0.7255
Precision: 0.7198
Recall: 0.7255
F1-Score: 0.7083

Classification Report:
              precision    recall  f1-score   support

          AD       0.69      0.38      0.49       225
          CN       0.71      0.66      0.68       288
         MCI       0.74      0.91      0.82       518

    accuracy                           0.73      1031
   macro avg       0.71      0.65      0.66      1031
weighted avg       0.72      0.73      0.71      1031


============================================================
Processing: split_90_10
============================================================
Train samples: 4175
Val samples: 464
Test samples: 515
Starting phase 1 training (partial unfreeze)…
Epoch [1/18], Loss: 1.0567, Train Acc: 49.75%, Val Acc: 50.22%, LR: 0.000033
Epoch [2/18], Loss: 1.0379, Train Acc: 50.35%, Val Acc: 50.65%, LR: 0.000067
Epoch [3/18], Loss: 1.0292, Train Acc: 50.49%, Val Acc: 51.29%, LR: 0.000100
Epoch [4/18], Loss: 1.0097, Train Acc: 51.93%, Val Acc: 52.16%, LR: 0.000099
Epoch [5/18], Loss: 0.9791, Train Acc: 54.85%, Val Acc: 54.96%, LR: 0.000096
Epoch [6/18], Loss: 0.9514, Train Acc: 56.77%, Val Acc: 59.05%, LR: 0.000091
Epoch [7/18], Loss: 0.9312, Train Acc: 59.21%, Val Acc: 60.56%, LR: 0.000084
Epoch [8/18], Loss: 0.9063, Train Acc: 61.58%, Val Acc: 61.42%, LR: 0.000075
Epoch [9/18], Loss: 0.8784, Train Acc: 63.52%, Val Acc: 65.09%, LR: 0.000066
Epoch [10/18], Loss: 0.8629, Train Acc: 66.06%, Val Acc: 67.03%, LR: 0.000056
Epoch [11/18], Loss: 0.8647, Train Acc: 65.03%, Val Acc: 66.81%, LR: 0.000045
Epoch [12/18], Loss: 0.8421, Train Acc: 66.71%, Val Acc: 66.81%, LR: 0.000035
Epoch [13/18], Loss: 0.8373, Train Acc: 66.78%, Val Acc: 65.95%, LR: 0.000026
Epoch [14/18], Loss: 0.8241, Train Acc: 67.31%, Val Acc: 70.47%, LR: 0.000017
Epoch [15/18], Loss: 0.8256, Train Acc: 67.81%, Val Acc: 68.10%, LR: 0.000010
Epoch [16/18], Loss: 0.8182, Train Acc: 68.17%, Val Acc: 66.81%, LR: 0.000005
Epoch [17/18], Loss: 0.8128, Train Acc: 69.34%, Val Acc: 68.53%, LR: 0.000002
Epoch [18/18], Loss: 0.8140, Train Acc: 69.17%, Val Acc: 70.47%, LR: 0.000001
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
```

```
access the learning rate.
  warnings.warn(

Epoch [1/12], Loss: 0.8114, Train Acc: 68.91%, Val Acc: 72.84%, LR: 0.000005
Epoch [2/12], Loss: 0.8168, Train Acc: 68.43%, Val Acc: 70.69%, LR: 0.000005
Epoch [3/12], Loss: 0.7996, Train Acc: 69.87%, Val Acc: 70.47%, LR: 0.000005
Epoch [4/12], Loss: 0.7956, Train Acc: 69.56%, Val Acc: 71.55%, LR: 0.000005
Epoch [5/12], Loss: 0.7989, Train Acc: 69.89%, Val Acc: 70.69%, LR: 0.000005
Epoch [6/12], Loss: 0.7967, Train Acc: 70.49%, Val Acc: 71.55%, LR: 0.000003
Epoch [7/12], Loss: 0.7897, Train Acc: 69.44%, Val Acc: 71.12%, LR: 0.000003
Epoch [8/12], Loss: 0.7809, Train Acc: 70.66%, Val Acc: 72.20%, LR: 0.000003
Epoch [9/12], Loss: 0.7743, Train Acc: 70.95%, Val Acc: 71.55%, LR: 0.000003
Early stopping at epoch 9
Testing model…

=== split_90_10 Results ===
Accuracy: 0.7262
Precision: 0.7293
Recall: 0.7262
F1-Score: 0.7158

Classification Report:
              precision    recall  f1-score   support

          AD       0.75      0.42      0.54       112
          CN       0.65      0.75      0.70       144
         MCI       0.77      0.85      0.80       259

    accuracy                           0.73       515
   macro avg       0.72      0.67      0.68       515
weighted avg       0.73      0.73      0.72       515


================================================================================
EfficientNet-B4 - SUMMARY OF ALL SPLITS
================================================================================
      split  accuracy  precision   recall  f1_score  training_time
split_10_90  0.503126   0.412399 0.503126  0.338000     209.842631
split_20_80  0.503517   0.476386 0.503517  0.338370     363.769990
split_30_70  0.565410   0.587784 0.565410  0.484915     840.035301
split_40_60  0.623868   0.638671 0.623868  0.585815     950.033990
split_50_50  0.654637   0.661264 0.654637  0.626693    1494.124134
split_60_40  0.684772   0.680800 0.684772  0.670393    1763.249976
split_70_30  0.716041   0.710036 0.716041  0.705865    2104.602802
split_80_20  0.725509   0.719785 0.725509  0.708299    2544.441258
split_90_10  0.726214   0.729257 0.726214  0.715817    2554.996111

Detailed results saved to: /kaggle/working/efficientnet_b4_results.csv
```

```
[6]: import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models
import pandas as pd
import numpy as np
from PIL import Image
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪accuracy_score, classification_report
import time
from torch.optim.lr_scheduler import CosineAnnealingLR, ReduceLROnPlateau

class AlzheimerDataset(Dataset):
    def __init__(self, split_dir, split_type, transform=None):
        self.split_dir = split_dir
        self.split_type = split_type
        self.transform = transform

        csv_path = os.path.join(split_dir, f"{split_type}.csv")
        self.df = pd.read_csv(csv_path)
        self.class_to_idx = {"AD": 0, "CN": 1, "MCI": 2}
        self.idx_to_class = {v: k for k, v in self.class_to_idx.items()}

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

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

        image = Image.open(img_path).convert('RGB')

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

        return image, label

def get_data_transforms():
    train_transform = transforms.Compose([
        transforms.Resize((384, 384)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(10),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,␣
 ↪hue=0.1),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
        transforms.RandomGrayscale(p=0.1),
        transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225]),
        transforms.RandomErasing(p=0.2, scale=(0.02, 0.2), ratio=(0.3, 3.3))
    ])

    val_transform = transforms.Compose([
        transforms.Resize((384, 384)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
↪225])
    ])

    return train_transform, val_transform

def train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs, device, warmup_epochs=3):
    best_val_acc = 0
    patience = 8
    patience_counter = 0

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        # Learning rate warmup
        if epoch < warmup_epochs:
            lr_scale = min(1.0, float(epoch + 1) / warmup_epochs)
            for param_group in optimizer.param_groups:
                param_group['lr'] = param_group['initial_lr'] * lr_scale

        for batch_idx, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()

            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_loss = running_loss / len(train_loader)
        epoch_acc = 100 * correct / total

        val_acc = evaluate_model(model, val_loader, device)

        if scheduler and epoch >= warmup_epochs:
            if isinstance(scheduler, CosineAnnealingLR):
                scheduler.step()
            else:
                scheduler.step(val_acc)

        current_lr = optimizer.param_groups[0]['lr']
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Train␣
 ↪Acc: {epoch_acc:.2f}%, Val Acc: {val_acc:.2f}%, LR: {current_lr:.6f}')

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print(f"Early stopping at epoch {epoch+1}")
            break

def evaluate_model(model, data_loader, device):
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in data_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return accuracy

def test_model(model, test_loader, device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return all_preds, all_labels

def calculate_metrics(y_true, y_pred, split_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted',␣
↪zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)

    print(f"\n=== {split_name} Results ===")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=['AD', 'CN',␣
↪'MCI'], zero_division=0))

    return {
        'split': split_name,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1
    }

def run_efficientnetv2_s_on_splits():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device: {device}")

    splits_root = "/kaggle/working/alzheimer-resized-224_splits"
    split_folders = [f for f in os.listdir(splits_root) if f.
↪startswith('split_')]
    split_folders.sort()
    results = []

    train_transform, val_transform = get_data_transforms()

    for split_folder in split_folders:
        print(f"\n{'='*60}")
        print(f"Processing: {split_folder}")
        print(f"{'='*60}")

        split_path = os.path.join(splits_root, split_folder)

        train_dataset = AlzheimerDataset(split_path, 'train', train_transform)
        val_dataset = AlzheimerDataset(split_path, 'val', val_transform)
        test_dataset = AlzheimerDataset(split_path, 'test', val_transform)

        train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True,␣
↪num_workers=2, pin_memory=True)
        val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False,␣
↪num_workers=2, pin_memory=True)
        test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False,␣
↪num_workers=2, pin_memory=True)

        print(f"Train samples: {len(train_dataset)}")
        print(f"Val samples: {len(val_dataset)}")
        print(f"Test samples: {len(test_dataset)}")

        model = models.efficientnet_v2_s(weights=models.
↪EfficientNet_V2_S_Weights.IMAGENET1K_V1)

        # Freeze all layers initially
        for param in model.parameters():
            param.requires_grad = False

        # Unfreeze classifier
        num_ftrs = model.classifier[1].in_features
        model.classifier[1] = nn.Linear(num_ftrs, 3)

        # Unfreeze last blocks for feature adaptation
        for param in model.features[-3:].parameters():  # Last 3 blocks
            param.requires_grad = True
        for param in model.features[-4][-2:].parameters():  # Last part of 4th␣
↪last block
            param.requires_grad = True

        model = model.to(device)

        # Layer-wise learning rates for EfficientNetV2
        optimizer = torch.optim.AdamW([
            {'params': model.classifier.parameters(), 'lr': 0.0001,␣
↪'initial_lr': 0.0001},
            {'params': model.features[-1].parameters(), 'lr': 0.00005,␣
↪'initial_lr': 0.00005},
            {'params': model.features[-2].parameters(), 'lr': 0.00003,␣
↪'initial_lr': 0.00003},
            {'params': model.features[-3].parameters(), 'lr': 0.00002,␣
↪'initial_lr': 0.00002},
            {'params': model.features[-4][-2:].parameters(), 'lr': 0.00001,␣
↪'initial_lr': 0.00001},
        ], weight_decay=1e-5)

        criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
        scheduler = CosineAnnealingLR(optimizer, T_max=20, eta_min=1e-6)

        print("Starting phase 1 training (partial unfreeze)...")
        start_time = time.time()
        train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs=20, device=device)
        phase1_time = time.time() - start_time

        print("Starting phase 2 training (full fine-tuning)...")
        for param in model.parameters():
            param.requires_grad = True

        optimizer = torch.optim.AdamW(model.parameters(), lr=0.000005,␣
↪weight_decay=1e-6)
        scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.5,␣
↪patience=4, verbose=True)

        start_time_phase2 = time.time()
        train_model(model, train_loader, val_loader, criterion, optimizer,␣
↪scheduler, num_epochs=15, device=device, warmup_epochs=0)
        total_time = phase1_time + (time.time() - start_time_phase2)

        print("Testing model...")
        test_preds, test_labels = test_model(model, test_loader, device)

        split_results = calculate_metrics(test_labels, test_preds, split_folder)
        split_results['training_time'] = total_time
        results.append(split_results)

        torch.cuda.empty_cache()

    results_df = pd.DataFrame(results)
    print(f"\n{'='*80}")
    print("EfficientNetV2-S - SUMMARY OF ALL SPLITS")
    print(f"{'='*80}")
    print(results_df.to_string(index=False))

    results_csv_path = "/kaggle/working/efficientnetv2_s_results.csv"
    results_df.to_csv(results_csv_path, index=False)
    print(f"\nDetailed results saved to: {results_csv_path}")

    return results_df

if __name__ == "__main__":
    results = run_efficientnetv2_s_on_splits()
```

```
Using device: cuda

============================================================
Processing: split_10_90
============================================================
Train samples: 464
Val samples: 51
Test samples: 4639

Downloading: "https://download.pytorch.org/models/efficientnet_v2_s-
dd5fe13b.pth" to /root/.cache/torch/hub/checkpoints/efficientnet_v2_s-
dd5fe13b.pth
100%|     | 82.7M/82.7M [00:00<00:00, 206MB/s]

Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.1189, Train Acc: 31.25%, Val Acc: 25.49%, LR: 0.000033
Epoch [2/20], Loss: 1.0671, Train Acc: 42.89%, Val Acc: 47.06%, LR: 0.000067
Epoch [3/20], Loss: 1.0449, Train Acc: 49.14%, Val Acc: 50.98%, LR: 0.000100
Epoch [4/20], Loss: 1.0263, Train Acc: 50.65%, Val Acc: 49.02%, LR: 0.000099
Epoch [5/20], Loss: 1.0197, Train Acc: 50.22%, Val Acc: 50.98%, LR: 0.000098
Epoch [6/20], Loss: 1.0034, Train Acc: 51.29%, Val Acc: 52.94%, LR: 0.000095
Epoch [7/20], Loss: 1.0005, Train Acc: 50.86%, Val Acc: 50.98%, LR: 0.000091
Epoch [8/20], Loss: 0.9832, Train Acc: 53.02%, Val Acc: 41.18%, LR: 0.000086
Epoch [9/20], Loss: 0.9601, Train Acc: 55.39%, Val Acc: 52.94%, LR: 0.000080
Epoch [10/20], Loss: 0.9389, Train Acc: 58.62%, Val Acc: 52.94%, LR: 0.000073
Epoch [11/20], Loss: 0.9256, Train Acc: 60.13%, Val Acc: 50.98%, LR: 0.000066
Epoch [12/20], Loss: 0.8987, Train Acc: 61.21%, Val Acc: 50.98%, LR: 0.000058
Epoch [13/20], Loss: 0.8979, Train Acc: 60.99%, Val Acc: 52.94%, LR: 0.000051
Epoch [14/20], Loss: 0.8558, Train Acc: 66.38%, Val Acc: 52.94%, LR: 0.000043
Early stopping at epoch 14
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
```

```
  warnings.warn(

Epoch [1/15], Loss: 0.8802, Train Acc: 64.22%, Val Acc: 54.90%, LR: 0.000005
Epoch [2/15], Loss: 0.8575, Train Acc: 67.24%, Val Acc: 56.86%, LR: 0.000005
Epoch [3/15], Loss: 0.8376, Train Acc: 67.67%, Val Acc: 62.75%, LR: 0.000005
Epoch [4/15], Loss: 0.8378, Train Acc: 67.24%, Val Acc: 56.86%, LR: 0.000005
Epoch [5/15], Loss: 0.8278, Train Acc: 69.18%, Val Acc: 58.82%, LR: 0.000005
Epoch [6/15], Loss: 0.8212, Train Acc: 69.83%, Val Acc: 54.90%, LR: 0.000005
Epoch [7/15], Loss: 0.8184, Train Acc: 69.83%, Val Acc: 56.86%, LR: 0.000005
Epoch [8/15], Loss: 0.8036, Train Acc: 70.26%, Val Acc: 52.94%, LR: 0.000003
Epoch [9/15], Loss: 0.7915, Train Acc: 70.91%, Val Acc: 56.86%, LR: 0.000003
Epoch [10/15], Loss: 0.8184, Train Acc: 70.04%, Val Acc: 49.02%, LR: 0.000003
Epoch [11/15], Loss: 0.8113, Train Acc: 70.26%, Val Acc: 54.90%, LR: 0.000003
Early stopping at epoch 11
Testing model…

=== split_10_90 Results ===
Accuracy: 0.5756
Precision: 0.5856
Recall: 0.5756
F1-Score: 0.5794

Classification Report:
              precision    recall  f1-score   support

          AD       0.42      0.45      0.44      1012
          CN       0.50      0.55      0.53      1296
         MCI       0.70      0.64      0.67      2331

    accuracy                           0.58      4639
   macro avg       0.54      0.55      0.55      4639
weighted avg       0.59      0.58      0.58      4639


============================================================
Processing: split_20_80
============================================================
Train samples: 928
Val samples: 103
Test samples: 4123
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0824, Train Acc: 41.38%, Val Acc: 43.69%, LR: 0.000033
Epoch [2/20], Loss: 1.0547, Train Acc: 50.00%, Val Acc: 50.49%, LR: 0.000067
Epoch [3/20], Loss: 1.0410, Train Acc: 49.46%, Val Acc: 48.54%, LR: 0.000100
Epoch [4/20], Loss: 1.0322, Train Acc: 50.32%, Val Acc: 51.46%, LR: 0.000099
Epoch [5/20], Loss: 1.0167, Train Acc: 51.19%, Val Acc: 55.34%, LR: 0.000098
Epoch [6/20], Loss: 1.0000, Train Acc: 53.56%, Val Acc: 54.37%, LR: 0.000095
Epoch [7/20], Loss: 0.9780, Train Acc: 54.42%, Val Acc: 54.37%, LR: 0.000091
Epoch [8/20], Loss: 0.9630, Train Acc: 56.36%, Val Acc: 57.28%, LR: 0.000086
Epoch [9/20], Loss: 0.9191, Train Acc: 60.02%, Val Acc: 55.34%, LR: 0.000080
Epoch [10/20], Loss: 0.9157, Train Acc: 60.45%, Val Acc: 56.31%, LR: 0.000073
Epoch [11/20], Loss: 0.8950, Train Acc: 62.07%, Val Acc: 58.25%, LR: 0.000066
Epoch [12/20], Loss: 0.8671, Train Acc: 64.22%, Val Acc: 60.19%, LR: 0.000058
Epoch [13/20], Loss: 0.8614, Train Acc: 66.16%, Val Acc: 58.25%, LR: 0.000051
Epoch [14/20], Loss: 0.8357, Train Acc: 65.09%, Val Acc: 59.22%, LR: 0.000043
Epoch [15/20], Loss: 0.8248, Train Acc: 67.46%, Val Acc: 61.17%, LR: 0.000035
Epoch [16/20], Loss: 0.8003, Train Acc: 67.67%, Val Acc: 63.11%, LR: 0.000028
Epoch [17/20], Loss: 0.7831, Train Acc: 68.97%, Val Acc: 58.25%, LR: 0.000021
Epoch [18/20], Loss: 0.7808, Train Acc: 71.12%, Val Acc: 64.08%, LR: 0.000015
Epoch [19/20], Loss: 0.7726, Train Acc: 70.69%, Val Acc: 65.05%, LR: 0.000010
Epoch [20/20], Loss: 0.7500, Train Acc: 73.28%, Val Acc: 66.99%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.7805, Train Acc: 69.61%, Val Acc: 67.96%, LR: 0.000005
Epoch [2/15], Loss: 0.7589, Train Acc: 70.80%, Val Acc: 65.05%, LR: 0.000005
Epoch [3/15], Loss: 0.7349, Train Acc: 74.68%, Val Acc: 67.96%, LR: 0.000005
Epoch [4/15], Loss: 0.7561, Train Acc: 72.41%, Val Acc: 65.05%, LR: 0.000005
Epoch [5/15], Loss: 0.7438, Train Acc: 73.49%, Val Acc: 66.02%, LR: 0.000005
Epoch [6/15], Loss: 0.7365, Train Acc: 73.28%, Val Acc: 68.93%, LR: 0.000005
Epoch [7/15], Loss: 0.7202, Train Acc: 74.46%, Val Acc: 66.99%, LR: 0.000005
Epoch [8/15], Loss: 0.7097, Train Acc: 76.94%, Val Acc: 70.87%, LR: 0.000005
Epoch [9/15], Loss: 0.7127, Train Acc: 74.89%, Val Acc: 69.90%, LR: 0.000005
Epoch [10/15], Loss: 0.7247, Train Acc: 75.65%, Val Acc: 66.02%, LR: 0.000005
Epoch [11/15], Loss: 0.7204, Train Acc: 75.75%, Val Acc: 68.93%, LR: 0.000005
Epoch [12/15], Loss: 0.6925, Train Acc: 78.12%, Val Acc: 67.96%, LR: 0.000005
Epoch [13/15], Loss: 0.6980, Train Acc: 76.29%, Val Acc: 68.93%, LR: 0.000003
Epoch [14/15], Loss: 0.6736, Train Acc: 78.66%, Val Acc: 69.90%, LR: 0.000003
Epoch [15/15], Loss: 0.6872, Train Acc: 77.26%, Val Acc: 70.87%, LR: 0.000003
Testing model…

=== split_20_80 Results ===
Accuracy: 0.7080
Precision: 0.7014
Recall: 0.7080
F1-Score: 0.7024

Classification Report:
              precision    recall  f1-score   support

          AD       0.61      0.55      0.58       899
          CN       0.67      0.58      0.62      1152
         MCI       0.76      0.85      0.80      2072
```

```
    accuracy                           0.71      4123
   macro avg       0.68      0.66      0.67      4123
weighted avg       0.70      0.71      0.70      4123


============================================================
Processing: split_30_70
============================================================
Train samples: 1391
Val samples: 155
Test samples: 3608
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0879, Train Acc: 38.46%, Val Acc: 46.45%, LR: 0.000033
Epoch [2/20], Loss: 1.0427, Train Acc: 49.68%, Val Acc: 49.03%, LR: 0.000067
Epoch [3/20], Loss: 1.0267, Train Acc: 50.25%, Val Acc: 53.55%, LR: 0.000100
Epoch [4/20], Loss: 1.0070, Train Acc: 51.76%, Val Acc: 54.19%, LR: 0.000099
Epoch [5/20], Loss: 0.9824, Train Acc: 54.64%, Val Acc: 59.35%, LR: 0.000098
Epoch [6/20], Loss: 0.9626, Train Acc: 56.00%, Val Acc: 67.10%, LR: 0.000095
Epoch [7/20], Loss: 0.9238, Train Acc: 59.74%, Val Acc: 59.35%, LR: 0.000091
Epoch [8/20], Loss: 0.8766, Train Acc: 61.90%, Val Acc: 70.97%, LR: 0.000086
Epoch [9/20], Loss: 0.8487, Train Acc: 66.14%, Val Acc: 70.97%, LR: 0.000080
Epoch [10/20], Loss: 0.8314, Train Acc: 66.79%, Val Acc: 67.10%, LR: 0.000073
Epoch [11/20], Loss: 0.7920, Train Acc: 69.73%, Val Acc: 71.61%, LR: 0.000066
Epoch [12/20], Loss: 0.7884, Train Acc: 69.73%, Val Acc: 65.16%, LR: 0.000058
Epoch [13/20], Loss: 0.7414, Train Acc: 73.69%, Val Acc: 74.84%, LR: 0.000051
Epoch [14/20], Loss: 0.7347, Train Acc: 72.83%, Val Acc: 70.97%, LR: 0.000043
Epoch [15/20], Loss: 0.7270, Train Acc: 74.55%, Val Acc: 77.42%, LR: 0.000035
Epoch [16/20], Loss: 0.6959, Train Acc: 75.92%, Val Acc: 75.48%, LR: 0.000028
Epoch [17/20], Loss: 0.6719, Train Acc: 77.43%, Val Acc: 75.48%, LR: 0.000021
Epoch [18/20], Loss: 0.6736, Train Acc: 77.57%, Val Acc: 75.48%, LR: 0.000015
Epoch [19/20], Loss: 0.6954, Train Acc: 75.05%, Val Acc: 74.19%, LR: 0.000010
Epoch [20/20], Loss: 0.6706, Train Acc: 78.58%, Val Acc: 71.61%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.6679, Train Acc: 77.71%, Val Acc: 73.55%, LR: 0.000005
Epoch [2/15], Loss: 0.6471, Train Acc: 79.22%, Val Acc: 73.55%, LR: 0.000005
Epoch [3/15], Loss: 0.6386, Train Acc: 80.73%, Val Acc: 76.77%, LR: 0.000005
Epoch [4/15], Loss: 0.6575, Train Acc: 77.93%, Val Acc: 78.06%, LR: 0.000005
Epoch [5/15], Loss: 0.6439, Train Acc: 80.16%, Val Acc: 73.55%, LR: 0.000005
Epoch [6/15], Loss: 0.6273, Train Acc: 80.30%, Val Acc: 77.42%, LR: 0.000005
Epoch [7/15], Loss: 0.6385, Train Acc: 79.65%, Val Acc: 76.77%, LR: 0.000005
Epoch [8/15], Loss: 0.6246, Train Acc: 80.95%, Val Acc: 76.77%, LR: 0.000005
Epoch [9/15], Loss: 0.6270, Train Acc: 80.52%, Val Acc: 78.06%, LR: 0.000003
Epoch [10/15], Loss: 0.6048, Train Acc: 82.31%, Val Acc: 75.48%, LR: 0.000003
Epoch [11/15], Loss: 0.6071, Train Acc: 82.31%, Val Acc: 78.71%, LR: 0.000003
Epoch [12/15], Loss: 0.6025, Train Acc: 82.67%, Val Acc: 81.29%, LR: 0.000003
Epoch [13/15], Loss: 0.5998, Train Acc: 83.90%, Val Acc: 80.00%, LR: 0.000003
Epoch [14/15], Loss: 0.6034, Train Acc: 81.60%, Val Acc: 80.00%, LR: 0.000003
Epoch [15/15], Loss: 0.6003, Train Acc: 81.38%, Val Acc: 76.13%, LR: 0.000003
Testing model…

=== split_30_70 Results ===
Accuracy: 0.8004
Precision: 0.7985
Recall: 0.8004
F1-Score: 0.7989

Classification Report:
              precision    recall  f1-score   support

          AD       0.75      0.67      0.71       787
          CN       0.76      0.76      0.76      1008
         MCI       0.84      0.88      0.86      1813

    accuracy                           0.80      3608
   macro avg       0.78      0.77      0.78      3608
weighted avg       0.80      0.80      0.80      3608


============================================================
Processing: split_40_60
============================================================
Train samples: 1855
Val samples: 207
Test samples: 3092
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0677, Train Acc: 45.44%, Val Acc: 49.76%, LR: 0.000033
Epoch [2/20], Loss: 1.0400, Train Acc: 49.76%, Val Acc: 52.17%, LR: 0.000067
Epoch [3/20], Loss: 1.0213, Train Acc: 51.32%, Val Acc: 53.14%, LR: 0.000100
Epoch [4/20], Loss: 0.9992, Train Acc: 53.15%, Val Acc: 54.59%, LR: 0.000099
Epoch [5/20], Loss: 0.9605, Train Acc: 56.33%, Val Acc: 60.87%, LR: 0.000098
Epoch [6/20], Loss: 0.9189, Train Acc: 60.97%, Val Acc: 64.73%, LR: 0.000095
Epoch [7/20], Loss: 0.8785, Train Acc: 62.43%, Val Acc: 67.63%, LR: 0.000091
Epoch [8/20], Loss: 0.8565, Train Acc: 65.39%, Val Acc: 63.77%, LR: 0.000086
Epoch [9/20], Loss: 0.8242, Train Acc: 66.52%, Val Acc: 65.70%, LR: 0.000080
Epoch [10/20], Loss: 0.7876, Train Acc: 70.78%, Val Acc: 68.60%, LR: 0.000073
Epoch [11/20], Loss: 0.7520, Train Acc: 72.56%, Val Acc: 72.46%, LR: 0.000066
Epoch [12/20], Loss: 0.7203, Train Acc: 74.23%, Val Acc: 74.88%, LR: 0.000058
Epoch [13/20], Loss: 0.7040, Train Acc: 76.06%, Val Acc: 75.85%, LR: 0.000051
Epoch [14/20], Loss: 0.6739, Train Acc: 78.44%, Val Acc: 74.88%, LR: 0.000043
Epoch [15/20], Loss: 0.6699, Train Acc: 78.01%, Val Acc: 77.78%, LR: 0.000035
Epoch [16/20], Loss: 0.6547, Train Acc: 78.76%, Val Acc: 76.33%, LR: 0.000028
Epoch [17/20], Loss: 0.6403, Train Acc: 81.08%, Val Acc: 79.23%, LR: 0.000021
Epoch [18/20], Loss: 0.6208, Train Acc: 82.05%, Val Acc: 79.23%, LR: 0.000015
Epoch [19/20], Loss: 0.6163, Train Acc: 81.99%, Val Acc: 79.71%, LR: 0.000010
Epoch [20/20], Loss: 0.6004, Train Acc: 83.07%, Val Acc: 81.16%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…
```

```
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(
```

```
Epoch [1/15], Loss: 0.6209, Train Acc: 81.40%, Val Acc: 80.68%, LR: 0.000005
Epoch [2/15], Loss: 0.5999, Train Acc: 82.86%, Val Acc: 79.71%, LR: 0.000005
Epoch [3/15], Loss: 0.6031, Train Acc: 82.32%, Val Acc: 81.16%, LR: 0.000005
Epoch [4/15], Loss: 0.5915, Train Acc: 83.56%, Val Acc: 82.13%, LR: 0.000005
Epoch [5/15], Loss: 0.5913, Train Acc: 83.72%, Val Acc: 82.13%, LR: 0.000005
Epoch [6/15], Loss: 0.5641, Train Acc: 85.82%, Val Acc: 81.64%, LR: 0.000005
Epoch [7/15], Loss: 0.5680, Train Acc: 84.96%, Val Acc: 81.16%, LR: 0.000005
Epoch [8/15], Loss: 0.5696, Train Acc: 84.69%, Val Acc: 84.06%, LR: 0.000005
Epoch [9/15], Loss: 0.5586, Train Acc: 86.20%, Val Acc: 83.09%, LR: 0.000005
Epoch [10/15], Loss: 0.5684, Train Acc: 85.12%, Val Acc: 84.54%, LR: 0.000005
Epoch [11/15], Loss: 0.5481, Train Acc: 85.61%, Val Acc: 83.09%, LR: 0.000005
Epoch [12/15], Loss: 0.5336, Train Acc: 86.68%, Val Acc: 85.51%, LR: 0.000005
Epoch [13/15], Loss: 0.5320, Train Acc: 87.33%, Val Acc: 83.57%, LR: 0.000005
Epoch [14/15], Loss: 0.5304, Train Acc: 87.28%, Val Acc: 83.09%, LR: 0.000005
Epoch [15/15], Loss: 0.5302, Train Acc: 86.85%, Val Acc: 84.54%, LR: 0.000005
Testing model…

=== split_40_60 Results ===
Accuracy: 0.8490
Precision: 0.8495
Recall: 0.8490
F1-Score: 0.8490

Classification Report:
              precision    recall  f1-score   support

          AD       0.77      0.80      0.78       674
          CN       0.85      0.81      0.83       864
         MCI       0.88      0.89      0.89      1554

    accuracy                           0.85      3092
   macro avg       0.84      0.83      0.83      3092
weighted avg       0.85      0.85      0.85      3092


============================================================
Processing: split_50_50
```

```
============================================================
Train samples: 2319
Val samples: 258
Test samples: 2577
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0559, Train Acc: 47.65%, Val Acc: 50.00%, LR: 0.000033
Epoch [2/20], Loss: 1.0336, Train Acc: 51.01%, Val Acc: 52.33%, LR: 0.000067
Epoch [3/20], Loss: 1.0009, Train Acc: 53.08%, Val Acc: 52.71%, LR: 0.000100
Epoch [4/20], Loss: 0.9538, Train Acc: 57.14%, Val Acc: 58.14%, LR: 0.000099
Epoch [5/20], Loss: 0.9149, Train Acc: 59.38%, Val Acc: 56.98%, LR: 0.000098
Epoch [6/20], Loss: 0.8768, Train Acc: 63.52%, Val Acc: 67.83%, LR: 0.000095
Epoch [7/20], Loss: 0.8191, Train Acc: 67.23%, Val Acc: 67.83%, LR: 0.000091
Epoch [8/20], Loss: 0.7878, Train Acc: 69.86%, Val Acc: 70.54%, LR: 0.000086
Epoch [9/20], Loss: 0.7598, Train Acc: 71.54%, Val Acc: 71.32%, LR: 0.000080
Epoch [10/20], Loss: 0.7317, Train Acc: 74.13%, Val Acc: 77.13%, LR: 0.000073
Epoch [11/20], Loss: 0.6930, Train Acc: 76.63%, Val Acc: 79.46%, LR: 0.000066
Epoch [12/20], Loss: 0.6722, Train Acc: 77.92%, Val Acc: 81.40%, LR: 0.000058
Epoch [13/20], Loss: 0.6335, Train Acc: 80.29%, Val Acc: 82.17%, LR: 0.000051
Epoch [14/20], Loss: 0.6300, Train Acc: 80.85%, Val Acc: 83.33%, LR: 0.000043
Epoch [15/20], Loss: 0.6317, Train Acc: 80.03%, Val Acc: 82.56%, LR: 0.000035
Epoch [16/20], Loss: 0.6053, Train Acc: 82.79%, Val Acc: 83.72%, LR: 0.000028
Epoch [17/20], Loss: 0.5836, Train Acc: 84.35%, Val Acc: 84.50%, LR: 0.000021
Epoch [18/20], Loss: 0.5889, Train Acc: 83.05%, Val Acc: 86.82%, LR: 0.000015
Epoch [19/20], Loss: 0.5771, Train Acc: 84.04%, Val Acc: 86.05%, LR: 0.000010
Epoch [20/20], Loss: 0.5870, Train Acc: 83.53%, Val Acc: 85.27%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.5622, Train Acc: 86.07%, Val Acc: 85.27%, LR: 0.000005
Epoch [2/15], Loss: 0.5636, Train Acc: 85.21%, Val Acc: 87.21%, LR: 0.000005
Epoch [3/15], Loss: 0.5563, Train Acc: 85.81%, Val Acc: 85.66%, LR: 0.000005
Epoch [4/15], Loss: 0.5434, Train Acc: 86.76%, Val Acc: 89.53%, LR: 0.000005
Epoch [5/15], Loss: 0.5377, Train Acc: 86.93%, Val Acc: 85.66%, LR: 0.000005
Epoch [6/15], Loss: 0.5410, Train Acc: 86.80%, Val Acc: 87.98%, LR: 0.000005
Epoch [7/15], Loss: 0.5228, Train Acc: 87.02%, Val Acc: 87.60%, LR: 0.000005
Epoch [8/15], Loss: 0.5087, Train Acc: 88.96%, Val Acc: 87.60%, LR: 0.000005
Epoch [9/15], Loss: 0.5206, Train Acc: 88.06%, Val Acc: 85.27%, LR: 0.000003
Epoch [10/15], Loss: 0.5112, Train Acc: 87.54%, Val Acc: 87.98%, LR: 0.000003
Epoch [11/15], Loss: 0.4945, Train Acc: 88.75%, Val Acc: 87.98%, LR: 0.000003
Epoch [12/15], Loss: 0.5035, Train Acc: 89.09%, Val Acc: 88.37%, LR: 0.000003
Early stopping at epoch 12
Testing model…

=== split_50_50 Results ===
Accuracy: 0.8700
```

```
Precision: 0.8723
Recall: 0.8700
F1-Score: 0.8690

Classification Report:
              precision    recall  f1-score   support

          AD       0.81      0.83      0.82       562
          CN       0.92      0.78      0.84       720
         MCI       0.87      0.94      0.90      1295

    accuracy                           0.87      2577
   macro avg       0.87      0.85      0.86      2577
weighted avg       0.87      0.87      0.87      2577


============================================================
Processing: split_60_40
============================================================
Train samples: 2784
Val samples: 308
Test samples: 2062
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0576, Train Acc: 47.59%, Val Acc: 49.35%, LR: 0.000033
Epoch [2/20], Loss: 1.0301, Train Acc: 50.57%, Val Acc: 52.27%, LR: 0.000067
Epoch [3/20], Loss: 0.9987, Train Acc: 52.80%, Val Acc: 54.87%, LR: 0.000100
Epoch [4/20], Loss: 0.9458, Train Acc: 57.29%, Val Acc: 59.42%, LR: 0.000099
Epoch [5/20], Loss: 0.8996, Train Acc: 61.17%, Val Acc: 69.48%, LR: 0.000098
Epoch [6/20], Loss: 0.8462, Train Acc: 65.16%, Val Acc: 67.21%, LR: 0.000095
Epoch [7/20], Loss: 0.8012, Train Acc: 68.21%, Val Acc: 70.45%, LR: 0.000091
Epoch [8/20], Loss: 0.7645, Train Acc: 71.95%, Val Acc: 75.00%, LR: 0.000086
Epoch [9/20], Loss: 0.7284, Train Acc: 73.92%, Val Acc: 76.62%, LR: 0.000080
Epoch [10/20], Loss: 0.6870, Train Acc: 77.08%, Val Acc: 76.62%, LR: 0.000073
Epoch [11/20], Loss: 0.6672, Train Acc: 77.37%, Val Acc: 80.84%, LR: 0.000066
Epoch [12/20], Loss: 0.6516, Train Acc: 78.99%, Val Acc: 81.82%, LR: 0.000058
Epoch [13/20], Loss: 0.6241, Train Acc: 80.82%, Val Acc: 81.82%, LR: 0.000051
Epoch [14/20], Loss: 0.6113, Train Acc: 81.50%, Val Acc: 81.82%, LR: 0.000043
Epoch [15/20], Loss: 0.5832, Train Acc: 83.66%, Val Acc: 86.36%, LR: 0.000035
Epoch [16/20], Loss: 0.5973, Train Acc: 83.48%, Val Acc: 85.71%, LR: 0.000028
Epoch [17/20], Loss: 0.5864, Train Acc: 83.33%, Val Acc: 86.36%, LR: 0.000021
Epoch [18/20], Loss: 0.5643, Train Acc: 84.23%, Val Acc: 86.36%, LR: 0.000015
Epoch [19/20], Loss: 0.5447, Train Acc: 86.64%, Val Acc: 87.34%, LR: 0.000010
Epoch [20/20], Loss: 0.5438, Train Acc: 86.67%, Val Acc: 87.66%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
```

```
  warnings.warn(
Epoch [1/15], Loss: 0.5545, Train Acc: 85.99%, Val Acc: 87.34%, LR: 0.000005
Epoch [2/15], Loss: 0.5516, Train Acc: 86.31%, Val Acc: 88.31%, LR: 0.000005
Epoch [3/15], Loss: 0.5400, Train Acc: 86.64%, Val Acc: 87.99%, LR: 0.000005
Epoch [4/15], Loss: 0.5220, Train Acc: 87.68%, Val Acc: 88.96%, LR: 0.000005
Epoch [5/15], Loss: 0.5127, Train Acc: 87.82%, Val Acc: 89.94%, LR: 0.000005
Epoch [6/15], Loss: 0.5021, Train Acc: 88.51%, Val Acc: 88.64%, LR: 0.000005
Epoch [7/15], Loss: 0.5032, Train Acc: 88.97%, Val Acc: 89.61%, LR: 0.000005
Epoch [8/15], Loss: 0.5000, Train Acc: 89.40%, Val Acc: 89.61%, LR: 0.000005
Epoch [9/15], Loss: 0.4942, Train Acc: 89.15%, Val Acc: 89.94%, LR: 0.000005
Epoch [10/15], Loss: 0.4871, Train Acc: 90.09%, Val Acc: 89.61%, LR: 0.000003
Epoch [11/15], Loss: 0.4751, Train Acc: 90.91%, Val Acc: 89.94%, LR: 0.000003
Epoch [12/15], Loss: 0.4920, Train Acc: 89.62%, Val Acc: 90.58%, LR: 0.000003
Epoch [13/15], Loss: 0.4838, Train Acc: 90.23%, Val Acc: 90.26%, LR: 0.000003
Epoch [14/15], Loss: 0.4840, Train Acc: 89.58%, Val Acc: 90.58%, LR: 0.000003
Epoch [15/15], Loss: 0.4759, Train Acc: 90.70%, Val Acc: 91.56%, LR: 0.000003
Testing model…

=== split_60_40 Results ===
Accuracy: 0.9234
Precision: 0.9234
Recall: 0.9234
F1-Score: 0.9229

Classification Report:
              precision    recall  f1-score   support

          AD       0.91      0.86      0.88       450
          CN       0.93      0.90      0.92       576
         MCI       0.92      0.96      0.94      1036

    accuracy                           0.92      2062
   macro avg       0.92      0.91      0.91      2062
weighted avg       0.92      0.92      0.92      2062


============================================================
Processing: split_70_30
============================================================
Train samples: 3247
Val samples: 361
Test samples: 1546
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0644, Train Acc: 45.24%, Val Acc: 50.14%, LR: 0.000033
Epoch [2/20], Loss: 1.0274, Train Acc: 50.63%, Val Acc: 50.14%, LR: 0.000067
Epoch [3/20], Loss: 0.9898, Train Acc: 54.11%, Val Acc: 57.62%, LR: 0.000100
Epoch [4/20], Loss: 0.9299, Train Acc: 59.13%, Val Acc: 63.16%, LR: 0.000099
Epoch [5/20], Loss: 0.8717, Train Acc: 64.12%, Val Acc: 67.04%, LR: 0.000098
Epoch [6/20], Loss: 0.8072, Train Acc: 68.28%, Val Acc: 70.08%, LR: 0.000095
Epoch [7/20], Loss: 0.7711, Train Acc: 71.39%, Val Acc: 70.36%, LR: 0.000091
Epoch [8/20], Loss: 0.7358, Train Acc: 73.36%, Val Acc: 76.73%, LR: 0.000086
Epoch [9/20], Loss: 0.7001, Train Acc: 75.73%, Val Acc: 78.12%, LR: 0.000080
Epoch [10/20], Loss: 0.6685, Train Acc: 78.72%, Val Acc: 78.39%, LR: 0.000073
Epoch [11/20], Loss: 0.6354, Train Acc: 80.57%, Val Acc: 81.72%, LR: 0.000066
Epoch [12/20], Loss: 0.6232, Train Acc: 80.94%, Val Acc: 83.10%, LR: 0.000058
Epoch [13/20], Loss: 0.5979, Train Acc: 82.38%, Val Acc: 81.72%, LR: 0.000051
Epoch [14/20], Loss: 0.5794, Train Acc: 83.74%, Val Acc: 84.49%, LR: 0.000043
Epoch [15/20], Loss: 0.5667, Train Acc: 84.35%, Val Acc: 86.15%, LR: 0.000035
Epoch [16/20], Loss: 0.5676, Train Acc: 84.76%, Val Acc: 86.43%, LR: 0.000028
Epoch [17/20], Loss: 0.5418, Train Acc: 86.57%, Val Acc: 86.70%, LR: 0.000021
Epoch [18/20], Loss: 0.5262, Train Acc: 87.03%, Val Acc: 86.43%, LR: 0.000015
Epoch [19/20], Loss: 0.5261, Train Acc: 87.00%, Val Acc: 86.98%, LR: 0.000010
Epoch [20/20], Loss: 0.5061, Train Acc: 89.07%, Val Acc: 88.37%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.5227, Train Acc: 87.68%, Val Acc: 88.09%, LR: 0.000005
Epoch [2/15], Loss: 0.5041, Train Acc: 88.79%, Val Acc: 88.09%, LR: 0.000005
Epoch [3/15], Loss: 0.4929, Train Acc: 89.31%, Val Acc: 90.03%, LR: 0.000005
Epoch [4/15], Loss: 0.4933, Train Acc: 89.04%, Val Acc: 89.47%, LR: 0.000005
Epoch [5/15], Loss: 0.4894, Train Acc: 89.47%, Val Acc: 92.24%, LR: 0.000005
Epoch [6/15], Loss: 0.4702, Train Acc: 90.76%, Val Acc: 90.58%, LR: 0.000005
Epoch [7/15], Loss: 0.4705, Train Acc: 91.07%, Val Acc: 91.97%, LR: 0.000005
Epoch [8/15], Loss: 0.4696, Train Acc: 90.70%, Val Acc: 92.24%, LR: 0.000005
Epoch [9/15], Loss: 0.4718, Train Acc: 91.04%, Val Acc: 91.69%, LR: 0.000005
Epoch [10/15], Loss: 0.4614, Train Acc: 91.47%, Val Acc: 92.24%, LR: 0.000003
Epoch [11/15], Loss: 0.4476, Train Acc: 92.49%, Val Acc: 91.41%, LR: 0.000003
Epoch [12/15], Loss: 0.4566, Train Acc: 91.90%, Val Acc: 93.07%, LR: 0.000003
Epoch [13/15], Loss: 0.4308, Train Acc: 93.44%, Val Acc: 93.07%, LR: 0.000003
Epoch [14/15], Loss: 0.4460, Train Acc: 92.18%, Val Acc: 92.80%, LR: 0.000003
Epoch [15/15], Loss: 0.4410, Train Acc: 92.55%, Val Acc: 93.63%, LR: 0.000003
Testing model…

=== split_70_30 Results ===
Accuracy: 0.9282
Precision: 0.9285
Recall: 0.9282
F1-Score: 0.9278

Classification Report:
              precision    recall  f1-score   support
```

```
          AD       0.89      0.88      0.88       337
          CN       0.95      0.89      0.92       432
         MCI       0.93      0.97      0.95       777

    accuracy                           0.93      1546
   macro avg       0.93      0.91      0.92      1546
weighted avg       0.93      0.93      0.93      1546


============================================================
Processing: split_80_20
============================================================
Train samples: 3711
Val samples: 412
Test samples: 1031
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0536, Train Acc: 48.69%, Val Acc: 49.51%, LR: 0.000033
Epoch [2/20], Loss: 1.0205, Train Acc: 51.47%, Val Acc: 51.94%, LR: 0.000067
Epoch [3/20], Loss: 0.9745, Train Acc: 55.38%, Val Acc: 59.71%, LR: 0.000100
Epoch [4/20], Loss: 0.9052, Train Acc: 61.36%, Val Acc: 66.75%, LR: 0.000099
Epoch [5/20], Loss: 0.8270, Train Acc: 67.50%, Val Acc: 71.36%, LR: 0.000098
Epoch [6/20], Loss: 0.7884, Train Acc: 69.82%, Val Acc: 75.49%, LR: 0.000095
Epoch [7/20], Loss: 0.7391, Train Acc: 73.30%, Val Acc: 79.85%, LR: 0.000091
Epoch [8/20], Loss: 0.6958, Train Acc: 76.48%, Val Acc: 83.98%, LR: 0.000086
Epoch [9/20], Loss: 0.6579, Train Acc: 79.12%, Val Acc: 84.71%, LR: 0.000080
Epoch [10/20], Loss: 0.6385, Train Acc: 79.92%, Val Acc: 86.17%, LR: 0.000073
Epoch [11/20], Loss: 0.6009, Train Acc: 82.16%, Val Acc: 88.59%, LR: 0.000066
Epoch [12/20], Loss: 0.5866, Train Acc: 84.21%, Val Acc: 89.81%, LR: 0.000058
Epoch [13/20], Loss: 0.5594, Train Acc: 85.42%, Val Acc: 88.59%, LR: 0.000051
Epoch [14/20], Loss: 0.5535, Train Acc: 85.29%, Val Acc: 91.75%, LR: 0.000043
Epoch [15/20], Loss: 0.5260, Train Acc: 87.36%, Val Acc: 91.99%, LR: 0.000035
Epoch [16/20], Loss: 0.5141, Train Acc: 88.44%, Val Acc: 92.96%, LR: 0.000028
Epoch [17/20], Loss: 0.5080, Train Acc: 88.66%, Val Acc: 93.69%, LR: 0.000021
Epoch [18/20], Loss: 0.4961, Train Acc: 89.68%, Val Acc: 94.42%, LR: 0.000015
Epoch [19/20], Loss: 0.4986, Train Acc: 88.95%, Val Acc: 93.93%, LR: 0.000010
Epoch [20/20], Loss: 0.4888, Train Acc: 89.44%, Val Acc: 94.42%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.4808, Train Acc: 90.00%, Val Acc: 94.66%, LR: 0.000005
Epoch [2/15], Loss: 0.4756, Train Acc: 90.30%, Val Acc: 94.42%, LR: 0.000005
Epoch [3/15], Loss: 0.4680, Train Acc: 91.11%, Val Acc: 94.90%, LR: 0.000005
Epoch [4/15], Loss: 0.4596, Train Acc: 91.40%, Val Acc: 94.17%, LR: 0.000005
Epoch [5/15], Loss: 0.4544, Train Acc: 91.70%, Val Acc: 94.66%, LR: 0.000005
Epoch [6/15], Loss: 0.4551, Train Acc: 91.54%, Val Acc: 95.15%, LR: 0.000005
Epoch [7/15], Loss: 0.4544, Train Acc: 92.08%, Val Acc: 94.66%, LR: 0.000005
Epoch [8/15], Loss: 0.4382, Train Acc: 92.72%, Val Acc: 95.39%, LR: 0.000005
Epoch [9/15], Loss: 0.4389, Train Acc: 92.05%, Val Acc: 95.15%, LR: 0.000005
Epoch [10/15], Loss: 0.4336, Train Acc: 92.75%, Val Acc: 95.63%, LR: 0.000005
Epoch [11/15], Loss: 0.4275, Train Acc: 93.18%, Val Acc: 96.60%, LR: 0.000005
Epoch [12/15], Loss: 0.4310, Train Acc: 92.91%, Val Acc: 95.87%, LR: 0.000005
Epoch [13/15], Loss: 0.4268, Train Acc: 93.24%, Val Acc: 96.36%, LR: 0.000005
Epoch [14/15], Loss: 0.4243, Train Acc: 93.29%, Val Acc: 96.12%, LR: 0.000005
Epoch [15/15], Loss: 0.4160, Train Acc: 93.91%, Val Acc: 96.60%, LR: 0.000005
Testing model…

=== split_80_20 Results ===
Accuracy: 0.9505
Precision: 0.9504
Recall: 0.9505
F1-Score: 0.9504

Classification Report:
              precision    recall  f1-score   support

          AD       0.93      0.91      0.92       225
          CN       0.95      0.94      0.95       288
         MCI       0.96      0.97      0.97       518

    accuracy                           0.95      1031
   macro avg       0.95      0.94      0.94      1031
weighted avg       0.95      0.95      0.95      1031


============================================================
Processing: split_90_10
============================================================
Train samples: 4175
Val samples: 464
Test samples: 515
Starting phase 1 training (partial unfreeze)…
Epoch [1/20], Loss: 1.0491, Train Acc: 49.44%, Val Acc: 51.29%, LR: 0.000033
Epoch [2/20], Loss: 1.0222, Train Acc: 50.83%, Val Acc: 56.90%, LR: 0.000067
Epoch [3/20], Loss: 0.9654, Train Acc: 55.93%, Val Acc: 64.66%, LR: 0.000100
Epoch [4/20], Loss: 0.8875, Train Acc: 61.99%, Val Acc: 68.75%, LR: 0.000099
Epoch [5/20], Loss: 0.8243, Train Acc: 67.64%, Val Acc: 75.22%, LR: 0.000098
Epoch [6/20], Loss: 0.7677, Train Acc: 71.07%, Val Acc: 74.35%, LR: 0.000095
Epoch [7/20], Loss: 0.7118, Train Acc: 75.59%, Val Acc: 83.84%, LR: 0.000091
Epoch [8/20], Loss: 0.6802, Train Acc: 77.08%, Val Acc: 82.97%, LR: 0.000086
Epoch [9/20], Loss: 0.6453, Train Acc: 79.19%, Val Acc: 86.64%, LR: 0.000080
Epoch [10/20], Loss: 0.6024, Train Acc: 82.37%, Val Acc: 85.13%, LR: 0.000073
Epoch [11/20], Loss: 0.5921, Train Acc: 83.26%, Val Acc: 90.30%, LR: 0.000066
Epoch [12/20], Loss: 0.5655, Train Acc: 85.03%, Val Acc: 88.36%, LR: 0.000058
Epoch [13/20], Loss: 0.5430, Train Acc: 86.47%, Val Acc: 89.66%, LR: 0.000051
Epoch [14/20], Loss: 0.5295, Train Acc: 87.21%, Val Acc: 90.52%, LR: 0.000043
Epoch [15/20], Loss: 0.5138, Train Acc: 88.10%, Val Acc: 92.67%, LR: 0.000035
Epoch [16/20], Loss: 0.5047, Train Acc: 88.79%, Val Acc: 93.32%, LR: 0.000028
Epoch [17/20], Loss: 0.4846, Train Acc: 89.53%, Val Acc: 93.10%, LR: 0.000021
Epoch [18/20], Loss: 0.4827, Train Acc: 90.11%, Val Acc: 92.67%, LR: 0.000015
Epoch [19/20], Loss: 0.4757, Train Acc: 90.20%, Val Acc: 93.75%, LR: 0.000010
Epoch [20/20], Loss: 0.4735, Train Acc: 90.85%, Val Acc: 93.75%, LR: 0.000006
Starting phase 2 training (full fine-tuning)…

/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn(

Epoch [1/15], Loss: 0.4669, Train Acc: 91.19%, Val Acc: 93.75%, LR: 0.000005
Epoch [2/15], Loss: 0.4554, Train Acc: 90.97%, Val Acc: 93.97%, LR: 0.000005
Epoch [3/15], Loss: 0.4528, Train Acc: 91.78%, Val Acc: 93.53%, LR: 0.000005
Epoch [4/15], Loss: 0.4539, Train Acc: 91.74%, Val Acc: 94.18%, LR: 0.000005
Epoch [5/15], Loss: 0.4394, Train Acc: 92.36%, Val Acc: 94.61%, LR: 0.000005
Epoch [6/15], Loss: 0.4360, Train Acc: 92.48%, Val Acc: 94.83%, LR: 0.000005
Epoch [7/15], Loss: 0.4388, Train Acc: 92.67%, Val Acc: 95.47%, LR: 0.000005
Epoch [8/15], Loss: 0.4300, Train Acc: 92.69%, Val Acc: 95.69%, LR: 0.000005
Epoch [9/15], Loss: 0.4138, Train Acc: 94.49%, Val Acc: 95.04%, LR: 0.000005
Epoch [10/15], Loss: 0.4150, Train Acc: 94.13%, Val Acc: 95.26%, LR: 0.000005
Epoch [11/15], Loss: 0.4171, Train Acc: 93.92%, Val Acc: 95.69%, LR: 0.000005
Epoch [12/15], Loss: 0.4122, Train Acc: 94.40%, Val Acc: 96.12%, LR: 0.000005
Epoch [13/15], Loss: 0.4093, Train Acc: 94.11%, Val Acc: 96.12%, LR: 0.000005
Epoch [14/15], Loss: 0.4090, Train Acc: 94.49%, Val Acc: 96.34%, LR: 0.000005
Epoch [15/15], Loss: 0.4075, Train Acc: 94.56%, Val Acc: 96.55%, LR: 0.000005
Testing model…

=== split_90_10 Results ===
Accuracy: 0.9592
Precision: 0.9593
Recall: 0.9592
F1-Score: 0.9590

Classification Report:
              precision    recall  f1-score   support

          AD       0.95      0.91      0.93       112
          CN       0.97      0.96      0.97       144
         MCI       0.95      0.98      0.97       259

    accuracy                           0.96       515
   macro avg       0.96      0.95      0.95       515
weighted avg       0.96      0.96      0.96       515
```

```
================================================================================
EfficientNetV2-S - SUMMARY OF ALL SPLITS
================================================================================
      split  accuracy  precision   recall  f1_score  training_time
split_10_90  0.575555   0.585557 0.575555  0.579376     209.364409
split_20_80  0.707980   0.701371 0.707980  0.702363     601.484343
split_30_70  0.800443   0.798541 0.800443  0.798868     867.035979
split_40_60  0.848965   0.849452 0.848965  0.848979    1099.555901
split_50_50  0.870004   0.872320 0.870004  0.868957    1194.387525
split_60_40  0.923375   0.923386 0.923375  0.922852    1984.304198
split_70_30  0.928202   0.928495 0.928202  0.927773    2231.632533
split_80_20  0.950533   0.950384 0.950533  0.950398    2518.564639
split_90_10  0.959223   0.959273 0.959223  0.959043    2807.190982

Detailed results saved to: /kaggle/working/efficientnetv2_s_results.csv
```