

```
In [9]: from google.colab import drive
drive.mount('/content/drive')

import re, shutil, random
from pathlib import Path
from collections import defaultdict

# ===== CONFIG =====
SEED = 42
COPY_FILES = True
IMG_EXTS = {".jpg", ".jpeg", ".png", ".webp", ".jif"}

SPLIT_RATIOS = {"train": 0.70, "val": 0.15, "test": 0.15}
CLASSES = ["curly", "kinky", "wavy", "straight"]

BASE = Path("/content/drive/MyDrive/Master/FinalProject/Dataset/HairType")
# =====

def is_img(p: Path) -> bool:
    return p.is_file() and p.suffix.lower() in IMG_EXTS

def mkdir(p: Path):
    p.mkdir(parents=True, exist_ok=True)

def copy_or_move(src: Path, dst: Path):
    mkdir(dst.parent)
    shutil.copy2(src, dst) if COPY_FILES else shutil.move(str(src), str(dst))

def group_key(fname: str) -> str:
    s = Path(fname).stem.lower()
    s = re.sub(r"_subject_\d+$", "", s)
    s = re.sub(r"_crop_\d+$", "", s)
    return s

def compute_targets(total_images: int, ratios: dict) -> dict:
    if abs(sum(ratios.values()) - 1.0) > 1e-9:
        raise ValueError("SPLIT_RATIOS must sum to 1.0 exactly.")
    raw = {k: total_images * ratios[k] for k in ["train", "val", "test"]}
    targets = {k: int(round(raw[k])) for k in raw}
    drift = total_images - sum(targets.values())
```

```

keys = ["train", "val", "test"]
i = 0
while drift != 0:
    targets[keys[i % 3]] += 1 if drift > 0 else -1
    drift += -1 if drift > 0 else 1
    i += 1
for k in targets:
    targets[k] = max(0, targets[k])
return targets

def subdir_ci(parent: Path, name: str) -> Path | None:
    """Find direct child folder by name (case-insensitive)."""
    if not parent.exists():
        return None
    for p in parent.iterdir():
        if p.is_dir() and p.name.lower() == name.lower():
            return p
    return None

def find_dataset_dir_with_classes(base: Path, classes: list[str]) -> Path | None:
    """
    Find a directory that contains all class folders (case-insensitive).
    Search base and one level deeper.
    """
    classes_l = [c.lower() for c in classes]

    # check base itself
    names = {p.name.lower() for p in base.iterdir() if p.is_dir()} if base.exists() else set()
    if all(c in names for c in classes_l):
        return base

    # check one-level subdirs
    for d in base.rglob("*"):
        if not d.is_dir():
            continue
        try:
            child_names = {p.name.lower() for p in d.iterdir() if p.is_dir()}
        except Exception:
            continue
        if all(c in child_names for c in classes_l):
            return d

```

```

return None

def split_by_groups(class_dir: Path, target: dict):
    random.seed(SEED)
    groups = defaultdict(list)

    for f in class_dir.iterdir():
        if is_img(f):
            groups[group_key(f.name)].append(f)

    items = list(groups.items())
    random.shuffle(items)
    items.sort(key=lambda kv: len(kv[1]), reverse=True)

    splits = {k: [] for k in ["train", "val", "test"]}
    counts = {k: 0 for k in ["train", "val", "test"]}

    for _, files in items:
        g = len(files)
        fits = [sp for sp in ["train", "val", "test"] if counts[sp] + g <= target[sp]]
        sp = max(fits, key=lambda x: target[x] - counts[x]) if fits else min(
            ["train", "val", "test"], key=lambda x: (counts[x] + g) - target[x]
        )
        splits[sp] += files
        counts[sp] += g

    return splits, counts, len(groups)

# ===== RUN =====
DATASET_DIR = find_dataset_dir_with_classes(BASE, CLASSES)
if not DATASET_DIR:
    raise FileNotFoundError(
        f"Could not find a folder under {BASE} that contains: {CLASSES}\n"
        "Tip: check your folder names or move the classes into one parent folder."
    )

print(" DATASET_DIR found:", DATASET_DIR)
print("Subfolders here:", [p.name for p in DATASET_DIR.iterdir() if p.is_dir()][:20])

OUT = DATASET_DIR / "splits"
for sp in ["train", "val", "test"]:
    for cls in CLASSES:

```

```

        mkdir(OUT / sp / cls)

summary = {}

for cls in CLASSES:
    class_dir = subdir_ci(DATASET_DIR, cls) # case-insensitive
    if not class_dir:
        raise FileNotFoundError(f"Missing class folder (even with case-insensitive): {cls} داخل {DATASET_DIR}")

    total_images = sum(1 for p in class_dir.iterdir() if is_img(p))
    targets = compute_targets(total_images, SPLIT_RATIOS)

    print(f"\nClass: {class_dir.name} (requested: {cls})")
    print("Total images:", total_images, "=> targets:", targets)

    splits, counts, groups = split_by_groups(class_dir, targets)

    for sp, files in splits.items():
        for f in files:
            copy_or_move(f, OUT / sp / cls / f.name)

    summary[cls] = (counts, groups)

print("\n=== Split Summary ===")
for cls, (counts, groups) in summary.items():
    print(f"{cls}: train={counts['train']}, val={counts['val']}, test={counts['test']} | groups={groups}")

print("\n Output folder:", OUT)

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

DATASET_DIR found: /content/drive/MyDrive/Master/FinalProject/Dataset/HairType/data
Subfolders here: ['curly', 'kinky', 'straight', 'wavy']

Class: curly (requested: curly)

Total images: 513 => targets: {'train': 359, 'val': 77, 'test': 77}

Class: kinky (requested: kinky)

Total images: 217 => targets: {'train': 151, 'val': 33, 'test': 33}

Class: wavy (requested: wavy)

Total images: 330 => targets: {'train': 230, 'val': 50, 'test': 50}

Class: straight (requested: straight)

Total images: 486 => targets: {'train': 340, 'val': 73, 'test': 73}

=== Split Summary ===

curly: train=359, val=77, test=77 | groups=504

kinky: train=151, val=33, test=33 | groups=216

wavy: train=230, val=50, test=50 | groups=322

straight: train=340, val=73, test=73 | groups=486

Output folder: /content/drive/MyDrive/Master/FinalProject/Dataset/HairType/data/splits

```
In [10]: # =====  
# Training and Evaluating CNN Models Using PyTorch (Hair Type)  
# =====  
  
# 0) Mount Google Drive  
from google.colab import drive  
drive.mount('/content/drive')  
  
# 1) Imports  
import os  
import random  
import numpy as np  
import torch  
from torchvision import datasets, transforms as T, models  
from torchvision.models import EfficientNet_B0_Weights  
from torch import nn, optim  
from torch.utils.data import DataLoader
```

```

from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

# =====
# 2) Reproducibility (Seeds)
# =====
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# =====
# 3) Config
# =====
dataset_path = "/content/drive/MyDrive/Master/FinalProject/Dataset/HairType/data/splits"
batch_size = 16
img_size = 224
num_workers = 2
num_epochs = 8
lr = 1e-3

# =====
# 4) Helpers
# =====
def get_classes(train_dir):
    """Get class names from train directory."""
    all_data = datasets.ImageFolder(train_dir)
    return all_data.classes

```

```

def get_data_loaders(data_dir, batch_size, img_size=224, num_workers=2):
    """
    Returns: train_loader, val_loader, test_loader and dataset sizes.
    Train -> with augmentation
    Val/Test -> Resize + Normalize only
    """
    train_tfms = T.Compose([
        T.RandomResizedCrop(img_size, scale=(0.80, 1.0), ratio=(0.75, 1.33)),
        T.RandomHorizontalFlip(p=0.5),
        T.RandomRotation(12),
        T.ColorJitter(brightness=0.25, contrast=0.25, saturation=0.15, hue=0.02),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225]),
        T.RandomErasing(p=0.35, scale=(0.02, 0.10), ratio=(0.3, 3.3), value="random"),
    ])

    eval_tfms = T.Compose([
        T.Resize((img_size, img_size)),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225]),
    ])

    train_dir = os.path.join(data_dir, "train")
    val_dir = os.path.join(data_dir, "val")
    test_dir = os.path.join(data_dir, "test")

    train_ds = datasets.ImageFolder(train_dir, transform=train_tfms)
    val_ds = datasets.ImageFolder(val_dir, transform=eval_tfms)
    test_ds = datasets.ImageFolder(test_dir, transform=eval_tfms)

    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True)
    val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True)
    test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True)

    sizes = {"train": len(train_ds), "val": len(val_ds), "test": len(test_ds)}
    return train_loader, val_loader, test_loader, sizes

def train_model(model, dataloaders, sizes, criterion, optimizer, scheduler, device, num_epochs=5):
    best_acc = 0.0
    best_state = None

```

```

for epoch in range(num_epochs):
    print(f"\nEpoch {epoch+1}/{num_epochs}")
    print("-" * 30)

    for phase in ["train", "val"]:
        model.train() if phase == "train" else model.eval()

        running_loss = 0.0
        running_corrects = 0

        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device, non_blocking=True)
            labels = labels.to(device, non_blocking=True)

            optimizer.zero_grad(set_to_none=True)

            with torch.set_grad_enabled(phase == "train"):
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                preds = torch.argmax(outputs, dim=1)

                if phase == "train":
                    loss.backward()
                    optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels)

        if phase == "train":
            scheduler.step()

    epoch_loss = running_loss / max(1, sizes[phase])
    epoch_acc = running_corrects.double().item() / max(1, sizes[phase])

    print(f"{phase:5s} | Loss: {epoch_loss:.4f} | Acc: {epoch_acc:.4f}")

    if phase == "val" and epoch_acc > best_acc:
        best_acc = epoch_acc
        best_state = {k: v.detach().cpu().clone() for k, v in model.state_dict().items()}

if best_state is not None:

```

```

        model.load_state_dict(best_state)
        model = model.to(device)

    print(f"\nBest Val Acc: {best_acc:.4f}")
    return model

def evaluate_model(model_name, y_true, y_pred, labels):
    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)
    cm = confusion_matrix(y_true, y_pred)

    print(f"\nModel: {model_name}")
    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=labels, zero_division=0))

    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.show()

# =====
# 5) Load Data
# =====
train_loader, val_loader, test_loader, sizes = get_data_loaders(
    dataset_path, batch_size, img_size=img_size, num_workers=num_workers
)

classes = get_classes(os.path.join(dataset_path, "train"))
num_classes = len(classes)

print("Classes:", classes)
print("Sizes:", sizes)

```

```

dataloaders = {"train": train_loader, "val": val_loader}

# =====
# 6) Model
# =====
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)

weights = EfficientNet_B0_Weights.DEFAULT
model = models.efficientnet_b0(weights=weights)

# Replace classifier for our classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=lr)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# =====
# 7) Train
# =====
model = train_model(model, dataloaders, sizes, criterion, optimizer, scheduler, device, num_epochs=num_epochs)

# =====
# 8) Test Evaluate
# =====
y_true, y_pred = [], []

model.eval()
with torch.no_grad():
    for x, y in tqdm(test_loader, desc="Testing"):
        out = model(x.to(device, non_blocking=True))
        preds = torch.argmax(out, dim=1).cpu().numpy()
        y_pred.extend(list(preds))
        y_true.extend(list(y.numpy()))

y_true = np.array(y_true)
y_pred = np.array(y_pred)

evaluate_model("efficientnet_b0", y_true, y_pred, labels=classes)

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

Classes: ['curly', 'kinky', 'straight', 'wavy']

Sizes: {'train': 1080, 'val': 233, 'test': 233}

Device: cpu

Epoch 1/8

/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py:1118: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.

super().__init__(loader)

```
train | Loss: 0.6806 | Acc: 0.7333
val   | Loss: 0.7004 | Acc: 0.7253
```

Epoch 2/8

```
-----
train | Loss: 0.4786 | Acc: 0.8185
val   | Loss: 0.3945 | Acc: 0.8755
```

Epoch 3/8

```
-----
train | Loss: 0.4151 | Acc: 0.8407
val   | Loss: 0.4849 | Acc: 0.8283
```

Epoch 4/8

```
-----
train | Loss: 0.4167 | Acc: 0.8528
val   | Loss: 0.3925 | Acc: 0.8541
```

Epoch 5/8

```
-----
train | Loss: 0.2856 | Acc: 0.8843
val   | Loss: 0.3693 | Acc: 0.8798
```

Epoch 6/8

```
-----
train | Loss: 0.2521 | Acc: 0.8981
val   | Loss: 0.3910 | Acc: 0.8240
```

Epoch 7/8

```
-----
train | Loss: 0.2392 | Acc: 0.9148
val   | Loss: 0.4435 | Acc: 0.8412
```

Epoch 8/8

```
-----
train | Loss: 0.2864 | Acc: 0.8880
val   | Loss: 0.4976 | Acc: 0.8326
```

Best Val Acc: 0.8798

Testing: 100%|██████████| 15/15 [00:16<00:00, 1.12s/it]

Model: efficientnet_b0
Accuracy: 0.8712
Precision: 0.8711
Recall: 0.8712
F1-score: 0.8709

Classification Report:

	precision	recall	f1-score	support
curly	0.90	0.94	0.92	77
kinky	0.91	0.91	0.91	33
straight	0.89	0.85	0.87	73
wavy	0.78	0.78	0.78	50
accuracy			0.87	233
macro avg	0.87	0.87	0.87	233
weighted avg	0.87	0.87	0.87	233

