

Project 1-1

Practical_Image_Processing_and_Natural_Language_Processing

Name: Sirin Mohammed Hejazi

ID: 243001213

Find the steps below:

Download the image dataset.

```
from PIL import Image
import os

input_root = 'dataset'      # الأصلية الصور مكان
output_root = 'output'      # التصغير بعد الصور حفظ مكان
resize_size = (256, 256)

os.makedirs(output_root, exist_ok=True)

for class_name in os.listdir(input_root):
    class_path = os.path.join(input_root, class_name)

    if not os.path.isdir(class_path):
        continue

    output_class_path = os.path.join(output_root, class_name)
    os.makedirs(output_class_path, exist_ok=True)

    for image_name in os.listdir(class_path):
        if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp',
        '.tiff')):
            input_image_path = os.path.join(class_path, image_name)
            output_image_path= os.path.join(
                output_class_path,
                os.path.splitext(image_name)[0] + '.jpg'
            )

            with Image.open(input_image_path) as img:
                img = img.convert('RGB')
                img = img.resize(resize_size)
                img.save(output_image_path, 'JPEG')
```

Splitting the dataset.

```

import os, shutil
from sklearn.model_selection import train_test_split

original_dataset_dir = 'output'
base_dir = 'new dataset'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

for d in [train_dir, validation_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [
        os.path.join(class_dir, f)
        for f in os.listdir(class_dir)
        if os.path.isfile(os.path.join(class_dir, f))
    ]

    train_images, temp_images = train_test_split(
        all_images, test_size=(1 - train_ratio), random_state=42
    )
    validation_images, test_images = train_test_split(
        temp_images,
        test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42
    )

    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ['train', 'validation', 'test']
    ):
        dest_dir = os.path.join(dest_base_dir, folder_name,
os.path.basename(class_dir))
        os.makedirs(dest_dir, exist_ok=True)
        for file in images:
            shutil.copy(file, dest_dir)

for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

```

Result

 new dataset	11/15/2025 4:53 PM	File folder
 output	11/15/2025 4:33 PM	File folder

 test	11/15/2025 4:56 PM	File folder
 train	11/15/2025 4:54 PM	File folder
 validation	11/15/2025 4:56 PM	File folder

Applying data augmentation.

```
import os
import torch
from torchvision import datasets, transforms as T
from torch.utils.data import DataLoader

def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

def get_data_loaders(data_dir, batch_size, train=False):
    """Prepare data loaders for training, validation, and testing."""
    if train:
        # Data augmentation for training
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
        ])
        train_data = datasets.ImageFolder(os.path.join(data_dir, "train"),
transform=transform)
        train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True, num_workers=4)
        return train_loader, len(train_data)
    else:
        # Validation and test data
        transform = T.Compose([
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
        ])
        val_data = datasets.ImageFolder(os.path.join(data_dir, "validation"),
transform=transform)
        test_data = datasets.ImageFolder(os.path.join(data_dir, "test"),
transform=transform)

        val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False, num_workers=4)
        test_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=False, num_workers=4)
```

```

        return val_loader, test_loader, len(val_data), len(test_data)

# Dataset paths and loaders
dataset_path = "new_dataset"
batch_size = 32

train_loader, train_data_len = get_data_loaders(dataset_path, batch_size,
train=True)
val_loader, test_loader, valid_data_len, test_data_len =
get_data_loaders(dataset_path, batch_size, train=False)

classes = get_classes("new_dataset/train")
num_classes = len(classes)

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_data_len, "val": valid_data_len}

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Device:", device)
print("Classes:", classes)

```

Training and testing models.

```

import os
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
from tqdm import tqdm
from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report
)
import matplotlib.pyplot as plt
import seaborn as sns

# ===== 1. dataset =====

def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

def get_data_loaders(data_dir, batch_size, train=False):
    """Prepare data loaders for training, validation, and testing."""
    if train:
        # Data augmentation for training
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),

```

```

        T.Resize((299, 299)),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406],
                     std=[0.229, 0.224, 0.225])
    ])
    train_data = datasets.ImageFolder(os.path.join(data_dir, "train/"),
                                      transform=transform)
    # IMPORTANT: num_workers=0 ويندوز على
    train_loader = DataLoader(train_data, batch_size=batch_size,
                              shuffle=True, num_workers=0)
    return train_loader, len(train_data)
else:
    # Validation and test data transformations
    transform = T.Compose([
        T.Resize((299, 299)),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406],
                     std=[0.229, 0.224, 0.225])
    ])
    val_data = datasets.ImageFolder(os.path.join(data_dir,
"validation/"),
                                   transform=transform)
    test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"),
                                     transform=transform)

    val_loader = DataLoader(val_data, batch_size=batch_size,
                             shuffle=False, num_workers=0)
    test_loader = DataLoader(test_data, batch_size=batch_size,
                              shuffle=False, num_workers=0)
    return val_loader, test_loader, len(val_data), len(test_data)

# ===== 2. العامة التدريب دالة =====

def train_model(model, criterion, optimizer, scheduler,
                dataloaders, dataset_sizes, device, num_epochs=5):
    """Train and validate the model."""
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        print('-' * 10)

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

            running_loss = 0.0
            running_corrects = 0

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

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)

            # logits/aux_logits فيه أو tuple يرجع الموديل لو

```

```

        if isinstance(outputs, tuple) or hasattr(outputs,
"logits"):
            main_outputs = outputs.logits if hasattr(outputs,
"logits") else outputs[0]
            aux_outputs = outputs.aux_logits if hasattr(outputs,
"aux_logits") else None
            loss = criterion(main_outputs, labels)
            if aux_outputs is not None:
                loss += 0.4 * criterion(aux_outputs, labels)
            else:
                main_outputs = outputs
                loss = criterion(main_outputs, labels)

            _, preds = torch.max(main_outputs, 1)

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

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

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

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

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

    return model

# ===== 3. التقييم دوال =====

def evaluate_model(model_name, y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, average='weighted')
    rec = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    print(f"{model_name} results")
    print("Accuracy:", acc)
    print("Precision :", prec)
    print("Recall:", rec)
    print("F1-score:", f1)
    print("Classification Report:")
    print(classification_report(y_true, y_pred))

    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.tight_layout()
    plt.show()

```

```

def get_preds_from_model(model, loader, device):
    y_true = []
    y_pred = []

    model.eval()
    with torch.no_grad():
        for x, y in tqdm(loader):
            x = x.to(device)
            outputs = model(x)
            probs = torch.softmax(outputs, dim=1)
            pred = torch.argmax(probs, dim=1).cpu().numpy()

            y_true.extend(y.numpy())
            y_pred.extend(pred)

    return np.array(y_true), np.array(y_pred)

# ===== 4. الرئيسية الدالة =====

def main():
    # 1) والجهاز الداتا إعداد
    dataset_path = "new dataset" # الصحيح بالهيكل المجلد هذا وجود من تأكد
    batch_size = 32

    train_loader, train_data_len = get_data_loaders(dataset_path, batch_size,
train=True)
    val_loader, test_loader, valid_data_len, test_data_len =
get_data_loaders(dataset_path, batch_size, train=False)

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

    dataloaders = {"train": train_loader, "val": val_loader}
    dataset_sizes = {"train": train_data_len, "val": valid_data_len}

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print("Device:", device)
    print("Classes:", classes)

    # ===== 5) الأول الموديل: EfficientNet-B0 =====
    print("\n===== Training EfficientNet-B0 =====")
    model_eff = models.efficientnet_b0(pretrained=True)
    model_eff.classifier[1] = nn.Linear(model_eff.classifier[1].in_features,
num_classes)
    model_eff = model_eff.to(device)

    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.AdamW(model_eff.parameters(), lr=0.001)
    exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.97)

    model_eff = train_model(
        model_eff, criterion, optimizer, exp_lr_scheduler,
        dataloaders, dataset_sizes, device,
        num_epochs=2
    )

    y_true_eff, y_pred_eff = get_preds_from_model(model_eff, test_loader,

```

```

device)
    evaluate_model("efficientnet_b0", y_true_eff, y_pred_eff, classes)

# ===== 6) الثاني الموديل: ResNet18 =====
print("\n===== Training ResNet18 ===== ")
model_res = models.resnet18(pretrained=False)
model_res.fc = nn.Linear(model_res.fc.in_features, num_classes)
model_res = model_res.to(device)

criterion2 = nn.CrossEntropyLoss().to(device)
optimizer2 = optim.AdamW(model_res.parameters(), lr=0.001)
exp_lr_scheduler2 = optim.lr_scheduler.StepLR(optimizer2, step_size=3,
gamma=0.97)

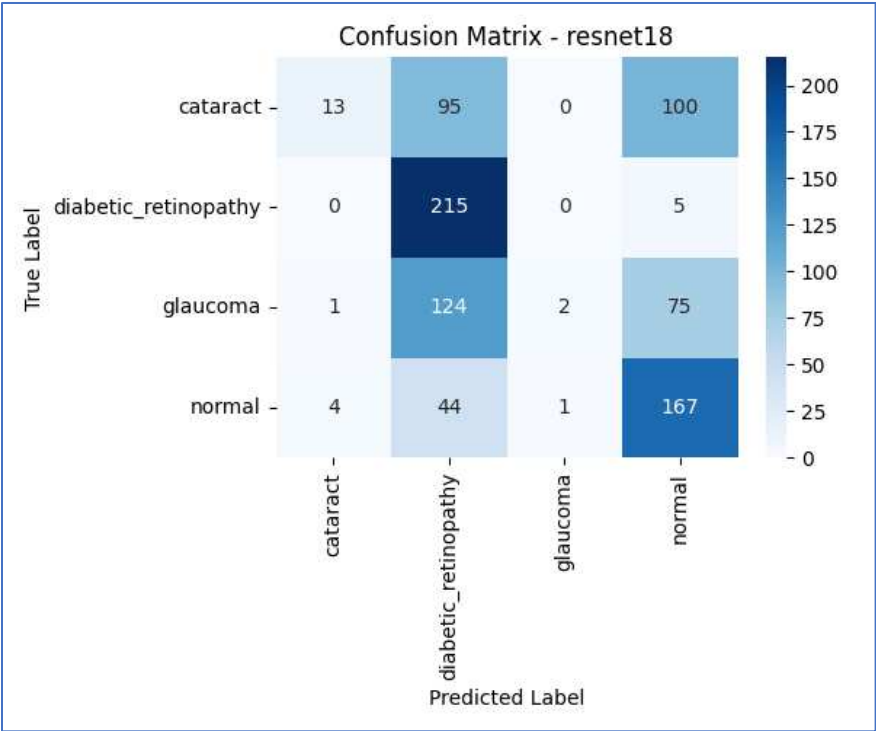
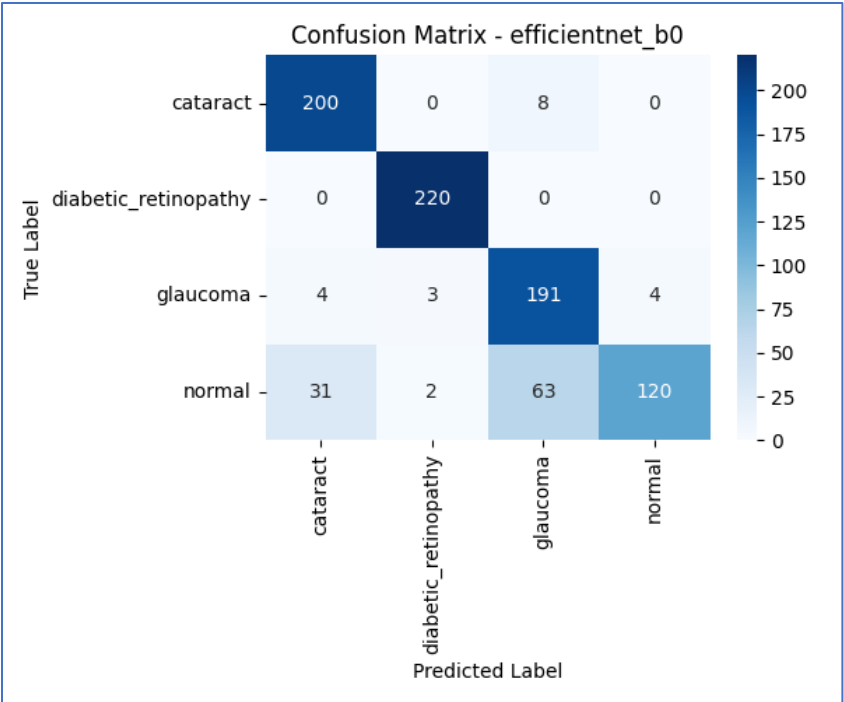
model_res = train_model(
    model_res, criterion2, optimizer2, exp_lr_scheduler2,
    dataloaders, dataset_sizes, device,
    num_epochs=2
)

y_true_res, y_pred_res = get_preds_from_model(model_res, test_loader,
device)
    evaluate_model("resnet18", y_true_res, y_pred_res, classes)

# ===== 5. ويندوز على الدخول نقطة =====

if __name__ == "__main__":
    main()

```



```

1 C:\Users\USER\Documents\Master\Assignment_1\.venv\
  Scripts\python.exe C:\Users\USER\Documents\Master\
  Assignment_1\Train_Models.py
2 Device: cpu
3 Classes: ['cataract', 'diabetic_retinopathy', '
  glaucoma', 'normal']
4
5 ===== Training EfficientNet-B0 =====
6 Epoch 1/2
7 -----
8 train Loss: 0.5318 Acc: 0.8023
9 val Loss: 0.3221 Acc: 0.8886
10
11 Epoch 2/2
12 -----
13 train Loss: 0.3286 Acc: 0.8813
14 val Loss: 0.3854 Acc: 0.8815
15
16 100%|██████████| 27/27 [00:43<00:00, 1.60s/it]
17 efficientnet_b0 results
18 Accuracy: 0.8640661938534279
19 Precision : 0.8846621604682089
20 Recall: 0.8640661938534279
21 F1-score: 0.8559229149031253
22 Classification Report:
23           precision    recall  f1-score   support
24
25      0           0.85       0.96       0.90         208
26      1           0.98       1.00       0.99         220
27      2           0.73       0.95       0.82         202
28      3           0.97       0.56       0.71         216
29
30      accuracy                0.86         846
31      macro avg              0.88         0.87         0.86         846
32      weighted avg          0.88         0.86         0.86         846
33
34
35 ===== Training ResNet18 =====
36 Epoch 1/2
37 -----
38 train Loss: 1.2112 Acc: 0.4761

```

```
39 val Loss: 1.1655 Acc: 0.4763
40
41 Epoch 2/2
42 -----
43 train Loss: 0.9102 Acc: 0.5870
44 val Loss: 1.3239 Acc: 0.4787
45
46 100%|██████████| 27/27 [00:45<00:00, 1.70s/it]
47 resnet18 results
48 Accuracy: 0.46926713947990545
49 Precision : 0.5765919081255317
50 Recall: 0.46926713947990545
51 F1-score: 0.34461329648828015
52 Classification Report:
53               precision    recall  f1-score   support
54
55          0           0.72       0.06       0.12         208
56          1           0.45       0.98       0.62         220
57          2           0.67       0.01       0.02         202
58          3           0.48       0.77       0.59         216
59
60      accuracy                   0.47         846
61      macro avg           0.58       0.46       0.34         846
62      weighted avg           0.58       0.47       0.34         846
63
64
65 Process finished with exit code 0
66
```