

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a new session
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

/kaggle/input/eye-diseases-classification/dataset/glaucoma/_177_4000002.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_102_5502806.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1279_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_301_6075675.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_47_787901.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/247_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_353_6531128.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1217_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1426_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_201_9193232.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_213_4395697.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/167_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_16_2996330.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_273_4238500.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_4_5317326.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_60_3287251.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_204_6942835.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1246_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_334_7538158.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_73_2704407.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_152_3014051.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_290_306861.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1352_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_383_5927182.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_223_2293487.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_116_1943666.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/Glaucoma_012.png
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_283_8286295.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_228_775182.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_296_6754407.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_5_5487582.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/365_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1267_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1386_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/247_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_315_1059988.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_245_5517087.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1327_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_268_5781265.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1239_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1222_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_283_6275047.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_305_769962.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1965_left.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1229_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_324_1040139.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_10_2555943.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_269_7481383.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/Glaucoma_038.png
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_219_5715309.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_156_1243799.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_189_1240865.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1485_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_110_299285.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_341_3343251.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_387_7675254.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/1232_right.jpg
/kaggle/input/eye-diseases-classification/dataset/glaucoma/Glaucoma_011.png
/kaggle/input/eye-diseases-classification/dataset/glaucoma/_233_3404764.jpg
```

Practical Image Processing and Natural Language Processing

Rayan Shukri

Dr. Hajer Saleh

Project 1

Imports & setup

```
import os
import shutil
import warnings

import numpy as np
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms as T, models

from PIL import Image
from sklearn.model_selection import train_test_split
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
from tqdm import tqdm

warnings.filterwarnings("ignore")

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

Device: cpu

1. locate the dataset

```
INPUT_ROOT = "/kaggle/input/eye-diseases-classification/dataset"
print("Input root:", INPUT_ROOT)
print("Classes:", os.listdir(INPUT_ROOT))
```

Input root: /kaggle/input/eye-diseases-classification/dataset
Classes: ['glaucoma', 'normal', 'diabetic_retinopathy', 'cataract']

2. Reduce image size

```
from PIL import Image

output_root = "/kaggle/working/output"
resize_size = (256, 256) # same as lecture

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 not image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
            continue

        input_image_path = os.path.join(class_path, image_name)
        output_image_name = os.path.splitext(image_name)[0] + ".jpg"
        output_image_path = os.path.join(output_class_path, output_image_name)

        try:
            with Image.open(input_image_path) as img:
                img = img.convert("RGB")
                img = img.resize(resize_size)
                img.save(output_image_path, "JPEG")
        except Exception as e:
            print(f"Failed to process {input_image_path}: {e}")
```

```
print("Resize done. Resized data in:", output_root)
```

```
Resize done. Resized data in: /kaggle/working/output
```

3. Split into train / validation / test

```
from sklearn.model_selection import train_test_split

original_dataset_dir = output_root
base_dir = "/kaggle/working/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)

# apply to each class
for class_name in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_name)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

print("Splitting done. Base dir:", base_dir)
```

```
Splitting done. Base dir: /kaggle/working/new_dataset
```

Import Class

```
from torchvision import datasets

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

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

print("Classes:", classes)
print("num_classes:", num_classes)
```

```
Classes: ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
num_classes: 4
```

4. Data augmentation & DataLoaders

```
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=2)
        return train_loader, len(train_data)

    else:
        # Validation & test: no augmentation
        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=2)
        test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=2)

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

dataset_path = base_dir
batch_size = 32

train_loader, train_len = get_data_loaders(dataset_path, batch_size, train=True)
val_loader, test_loader, val_len, test_len = get_data_loaders(dataset_path, batch_size, train=False)

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_len, "val": val_len}

print("Train size:", train_len, "Val size:", val_len, "Test size:", test_len)
```

Train size: 2949 Val size: 422 Test size: 846

5. Build & train EfficientNet-B0 multi-class classifier

```
model = models.efficientnet_b0(pretrained=True)
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=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

def train_model(model, criterion, optimizer, scheduler, num_epochs=5):
    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 [B, num_classes]
    loss = criterion(outputs, labels)
    _, preds = torch.max(outputs, 1)

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

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

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

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

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

return model

model = train_model(model, criterion, optimizer, scheduler, num_epochs=5)

```

Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /root/.cache/torch/h
 100%|██████████| 20.5M/20.5M [00:00<00:00, 140MB/s]
 Epoch 1/5

 train Loss: 0.5306 Acc: 0.8026
 val Loss: 0.3460 Acc: 0.8768
 Epoch 2/5

 train Loss: 0.3082 Acc: 0.8867
 val Loss: 0.2018 Acc: 0.9242
 Epoch 3/5

 train Loss: 0.2715 Acc: 0.9006
 val Loss: 0.1878 Acc: 0.9242
 Epoch 4/5

 train Loss: 0.2262 Acc: 0.9193
 val Loss: 0.2062 Acc: 0.9194
 Epoch 5/5

 train Loss: 0.2027 Acc: 0.9230
 val Loss: 0.2587 Acc: 0.9052

6. Evaluate on the test set

```

model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for inputs, labels in tqdm(test_loader):
        inputs = inputs.to(device)
        outputs = model(inputs)
        probs = torch.softmax(outputs, dim=1)
        preds = torch.argmax(probs, dim=1).cpu().numpy()

        y_true.extend(labels.numpy())
        y_pred.extend(preds)

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

print("Accuracy :", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred, average="weighted"))
print("Recall   :", recall_score(y_true, y_pred, average="weighted"))
print("F1 score :", f1_score(y_true, y_pred, average="weighted"))
print("\nClassification report:")
print(classification_report(y_true, y_pred, target_names=classes))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d",
            xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted")

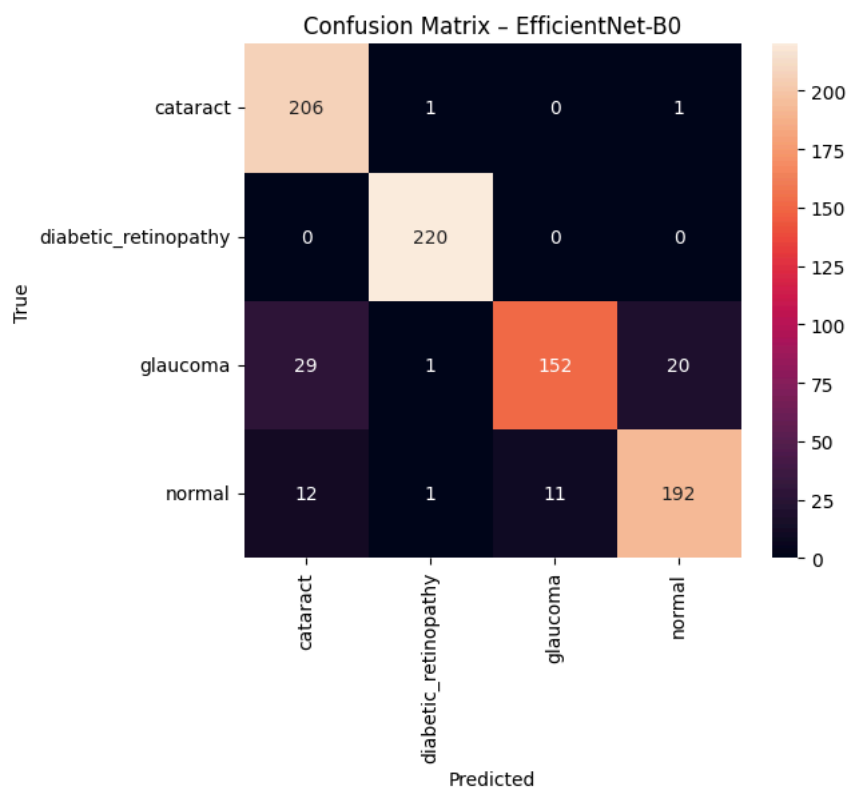
```

```
plt.ylabel("True")
plt.title("Confusion Matrix - EfficientNet-B0")
plt.show()
```

```
100%|██████████| 27/27 [00:41<00:00, 1.55s/it]
Accuracy : 0.9101654846335697
Precision: 0.9144046884947314
Recall : 0.9101654846335697
F1 score : 0.9083176931317607
```

Classification report:

	precision	recall	f1-score	support
cataract	0.83	0.99	0.91	208
diabetic_retinopathy	0.99	1.00	0.99	220
glaucoma	0.93	0.75	0.83	202
normal	0.90	0.89	0.90	216
accuracy			0.91	846
macro avg	0.91	0.91	0.91	846
weighted avg	0.91	0.91	0.91	846



7. Predict a single image

```
def predict_image(path):
    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]),
    ])

    img = Image.open(path).convert("RGB")
    x = transform(img).unsqueeze(0).to(device)

    model.eval()
    with torch.no_grad():
        outputs = model(x)
        probs = torch.softmax(outputs, dim=1)[0].cpu().numpy()
        pred_idx = int(np.argmax(probs))

    print("Predicted class:", classes[pred_idx])
    print("Probabilities:")
    for c, p in zip(classes, probs):
        print(f"{c}: {p:.3f}")

# Example (change path to one test image):
```

```
# predict_image("/kaggle/working/new_dataset/test/glaucoma/xxx.jpg")
```

Double-click (or enter) to edit

Double-click (or enter) to edit