

```
# Eye Disease Classification using EfficientNet-B0
# Practical Image Processing and NLP
# Student: [Metab Hussain]
```

```
!pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

 kaggle.json
kaggle.json(application/json) - 66 bytes, last modified: 9100% - 2026/3/ done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username": "metabjdeba", "key": "50ae2cd85f086ff1851f872d1e599385"}'}

Start coding or [generate](#) with AI.

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d gunavenkatdoddi/eye-diseases-classification
```

Dataset URL: <https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification>
License(s): ODbL-1.0
Downloading eye-diseases-classification.zip to /content
97% 713M/736M [00:02<00:00, 261MB/s]
100% 736M/736M [00:02<00:00, 352MB/s]

```
!unzip -q eye-diseases-classification.zip
```

```
import os
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 PIL import Image
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
import warnings
warnings.filterwarnings("ignore")
```

```
os.listdir()
```

```
['.config',
'dataset',
'eye-diseases-classification.zip',
'output',
'kaggle.json',
'new_dataset',
'sample_data']
```

```
os.listdir("dataset")
```

```
['cataract', 'glaucoma', 'normal', 'diabetic_retinopathy']
```

```
from PIL import Image
import os

input_root = 'dataset'
output_root = 'output'
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) or class_name == 'masks':
        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_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("Done")
```

Done

```
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 dir in [train_dir, validation_dir, test_dir]:
    os.makedirs(dir, 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)

# تطبيق التقسيم على كل الفئات
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)

print("Dataset split completed")

```

Dataset split completed

```

from torchvision import datasets

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

classes = get_classes("new_dataset/test")
print(classes, len(classes))

```

['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal'] 4

```

def get_data_loaders(data_dir, batch_size, train=False):
    if train:
        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)

        return train_loader, len(train_data)

    else:
        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)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

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

```
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(classes)
print(num_classes)
print(dataset_sizes)
print(device)
```

```
['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
4
{'train': 2949, 'val': 422}
cuda
```

```
from torchvision import models
import torch.nn as nn
import torch.optim as optim
```

```
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()
criterion = criterion.to(device)
```

```
optimizer = optim.AdamW(model.parameters(), lr=0.001)
```

```
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)
```

```
print("Model ready")
```

```
Model ready
```

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=2):
    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)

        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}")

return model

```

```

import torch
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0) if torch.cuda.is_available() else "No GPU")

```

```

True
Tesla T4

```

```

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)

```

```

Epoch 1/2
-----
train Loss: 0.5037 Acc: 0.8074
val Loss: 0.4797 Acc: 0.8128
Epoch 2/2
-----
train Loss: 0.3345 Acc: 0.8756
val Loss: 0.3256 Acc: 0.8768

```

```

def evaluate_model(model_name, y_true, y_pred, classes):

```

```

accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

cm = confusion_matrix(y_true, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

print("\nClassification 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")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

```

```

y_true = []
y_pred = []

model.eval()

with torch.no_grad():
    for x, y in test_loader:
        x = x.to(device)

        outputs = model(x)
        probabilities = torch.softmax(outputs, dim=1)

        pred = torch.argmax(probabilities, axis=1).cpu().numpy()

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

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

evaluate_model("EfficientNet_B0", y_true, y_pred, classes)

```

Accuracy: 0.8936170212765957
Precision: 0.8965040534579582
Recall: 0.8936170212765957
F1-score: 0.8938713225915554

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.96	0.92	208
1	1.00	0.92	0.96	220
2	0.83	0.88	0.86	202
3	0.86	0.81	0.84	216
accuracy			0.89	846
macro avg	0.89	0.89	0.89	846
weighted avg	0.90	0.89	0.89	846

