

```
In [1]: from IPython.display import Image, display
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
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc
```

```
In [ ]: from PIL import Image
import os

input_root = 'C:/Users/saldwsari/Documents/dataset/'
output_root = 'C:/Users/saldwsari/Documents/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')
                    print(f"Processed: {input_image_path} -> {output_image_path}")
            except Exception as e:
                print(f"Failed to process {input_image_path}: {e}")
```

```
In [12]: 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 = 0.7
validation_ratio = 0.1
test_ratio = 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(f)]
    train_images, temp_images = train_test_split(all_images, test_size=(1-train_ratio))
    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)
```

```
In [13]: 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
```

```
In [ ]: 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)
        train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
        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)
        test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"), 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)

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 setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
In [ ]: 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')
```

```
In [ ]: 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)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer, step_size=3, gamma=0.97
)
```

```
In [ ]: def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    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
                        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

```

```

In [21]: 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("Classification Report:")
    print(classification_report(y_true, y_pred))
    # Plot confusion matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, ytickla
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()

    y_true = []
    y_proba = []
    y_pred = []

    with torch.no_grad():
        for x, y in tqdm(test_loader):
            # Forward pass
            output = model(x.to(device))
            # Store true labels
            y_true.extend(y.numpy())
            # Get predicted probabilities (softmax output)
            probabilities = torch.softmax(output, dim=1) # Assuming it's a multi-class
            y_proba.extend(probabilities.cpu().numpy()) # Store probabilities
            # Get predicted classes (argmax for final class prediction)
            pred = torch.argmax(probabilities, axis=1).cpu().numpy()
            y_pred.extend(pred)

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

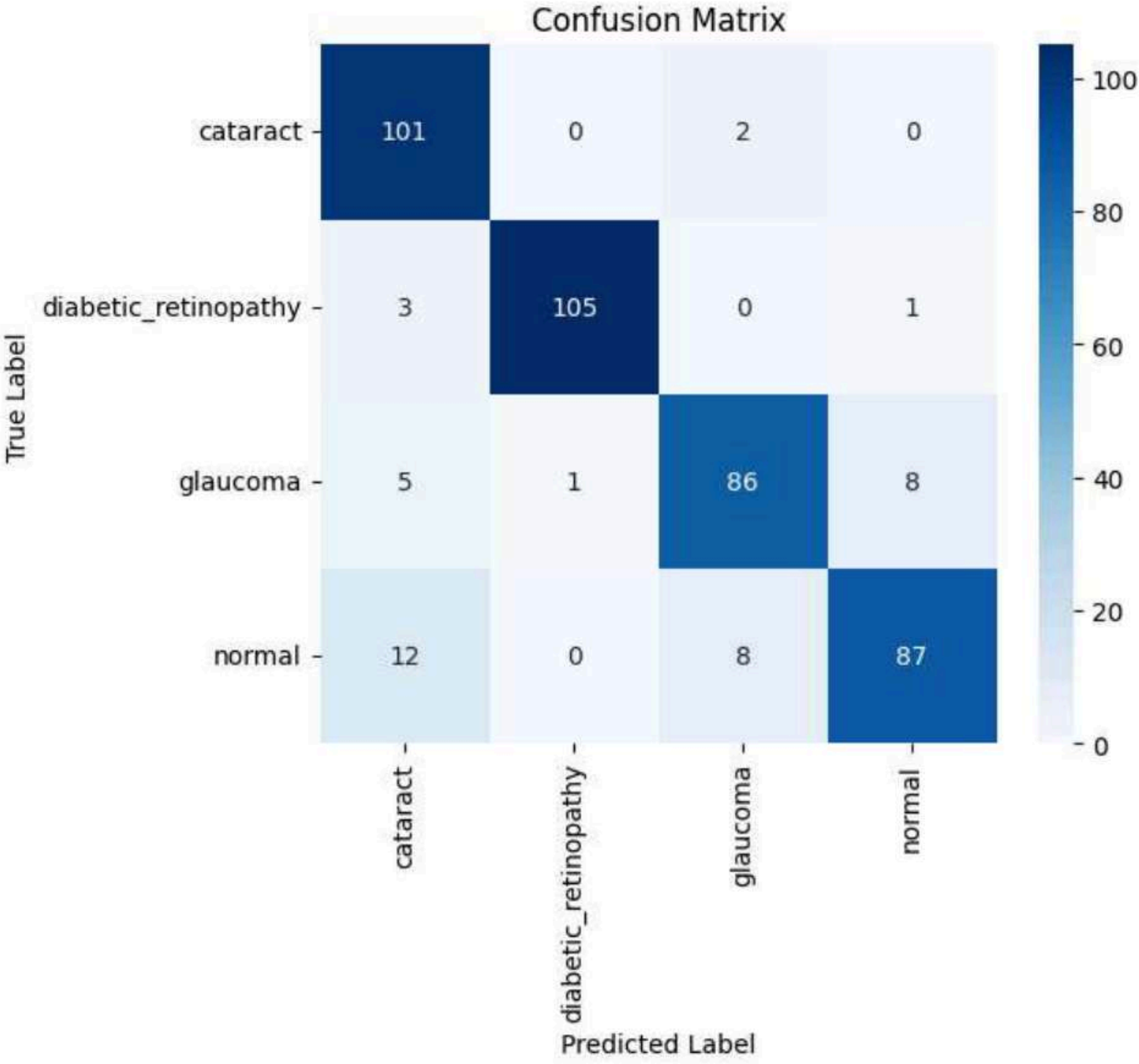
```

```
evaluate_model("efficientnet_b0", y_true, y_pred, classes)
```

100%|██████████| 14/14 [00:31<00:00, 2.25s/it]

Accuracy: 0.9045346062052506
Precision : 0.9081121433367808
Recall: 0.9045346062052506
F1-score: 0.9041012711350844
Classification Report:

	precision	recall	f1-score	support
0	0.83	0.98	0.90	103
1	0.99	0.96	0.98	109
2	0.90	0.86	0.88	100
3	0.91	0.81	0.86	107
accuracy			0.90	419
macro avg	0.91	0.90	0.90	419
weighted avg	0.91	0.90	0.90	419



```
In [22]: import torch
from torchvision import datasets, transforms as T
from torch.utils.data import DataLoader

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('new dataset', 'train'), transform=t
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
```

```
In [23]: import torch
import torch.nn as nn
from torchvision import models

model = models.inception_v3(pretrained=True)

for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Linear(model.fc.in_features, 4)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

```

Out[23]: Inception3(
  (Conv2d_1a_3x3): BasicConv2d(
    (conv): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), bias=False)
    (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stat
s=True)
  )
  (Conv2d_2a_3x3): BasicConv2d(
    (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stat
s=True)
  )
  (Conv2d_2b_3x3): BasicConv2d(
    (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stat
s=True)
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=
False)
  (Conv2d_3b_1x1): BasicConv2d(
    (conv): Conv2d(64, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(80, eps=0.001, momentum=0.1, affine=True, track_running_stat
s=True)
  )
  (Conv2d_4a_3x3): BasicConv2d(
    (conv): Conv2d(80, 192, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_sta
ts=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=
False)
  (Mixed_5b): InceptionA(
    (branch1x1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (branch5x5_1): BasicConv2d(
      (conv): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (branch5x5_2): BasicConv2d(
      (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bi
as=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (branch3x3dbl_1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (branch3x3dbl_2): BasicConv2d(
      (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)

```

```

        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3): BasicConv2d(
      (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch_pool): BasicConv2d(
      (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (Mixed_5c): InceptionA(
    (branch1x1): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_1): BasicConv2d(
      (conv): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_2): BasicConv2d(
      (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_1): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_2): BasicConv2d(
      (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3): BasicConv2d(
      (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch_pool): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (Mixed_5d): InceptionA(

```

```

        (branch1x1): BasicConv2d(
          (conv): Conv2d(288, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch5x5_1): BasicConv2d(
          (conv): Conv2d(288, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch5x5_2): BasicConv2d(
          (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
          (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch3x3dbl_1): BasicConv2d(
          (conv): Conv2d(288, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch3x3dbl_2): BasicConv2d(
          (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch3x3dbl_3): BasicConv2d(
          (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
        (branch_pool): BasicConv2d(
          (conv): Conv2d(288, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
    (Mixed_6a): InceptionB(
      (branch3x3): BasicConv2d(
        (conv): Conv2d(288, 384, kernel_size=(3, 3), stride=(2, 2), bias=False)
        (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch3x3dbl_1): BasicConv2d(
        (conv): Conv2d(288, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch3x3dbl_2): BasicConv2d(
        (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )

```

```

        (branch3x3dbl_3): BasicConv2d(
          (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), bias=False)
          (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
    (Mixed_6b): InceptionC(
      (branch1x1): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7_1): BasicConv2d(
        (conv): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7_2): BasicConv2d(
        (conv): Conv2d(128, 128, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7_3): BasicConv2d(
        (conv): Conv2d(128, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7dbl_1): BasicConv2d(
        (conv): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7dbl_2): BasicConv2d(
        (conv): Conv2d(128, 128, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7dbl_3): BasicConv2d(
        (conv): Conv2d(128, 128, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7dbl_4): BasicConv2d(
        (conv): Conv2d(128, 128, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0), bias=False)
        (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
      (branch7x7dbl_5): BasicConv2d(
        (conv): Conv2d(128, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
)

```

```

tats=True)
    )
    (branch_pool): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    )
    (Mixed_6c): InceptionC(
        (branch1x1): BasicConv2d(
            (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_1): BasicConv2d(
            (conv): Conv2d(768, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_2): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_3): BasicConv2d(
            (conv): Conv2d(160, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_1): BasicConv2d(
            (conv): Conv2d(768, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_2): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_3): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_4): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_5): BasicConv2d(
            (conv): Conv2d(160, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),

```

```

bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    (branch_pool): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    )
    (Mixed_6d): InceptionC(
        (branch1x1): BasicConv2d(
            (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_1): BasicConv2d(
            (conv): Conv2d(768, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_2): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7_3): BasicConv2d(
            (conv): Conv2d(160, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_1): BasicConv2d(
            (conv): Conv2d(768, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_2): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_3): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7dbl_4): BasicConv2d(
            (conv): Conv2d(160, 160, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(160, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
    )

```

```

        (branch7x7dbl_5): BasicConv2d(
          (conv): Conv2d(160, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
            bias=False)
          (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
            tats=True)
        )
        (branch_pool): BasicConv2d(
          (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
            tats=True)
        )
      )
    (Mixed_6e): InceptionC(
      (branch1x1): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7_1): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7_2): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
          bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7_3): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
          bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7dbl_1): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7dbl_2): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
          bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7dbl_3): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
          bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
          tats=True)
      )
      (branch7x7dbl_4): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
          bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s

```

```

tats=True)
    )
    (branch7x7dbl_5): BasicConv2d(
        (conv): Conv2d(192, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    (branch_pool): BasicConv2d(
        (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
    )
    (AuxLogits): InceptionAux(
        (conv0): BasicConv2d(
            (conv): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(128, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (conv1): BasicConv2d(
            (conv): Conv2d(128, 768, kernel_size=(5, 5), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(768, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (fc): Linear(in_features=768, out_features=1000, bias=True)
    )
    (Mixed_7a): InceptionD(
        (branch3x3_1): BasicConv2d(
            (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch3x3_2): BasicConv2d(
            (conv): Conv2d(192, 320, kernel_size=(3, 3), stride=(2, 2), bias=False)
            (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7x3_1): BasicConv2d(
            (conv): Conv2d(768, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7x3_2): BasicConv2d(
            (conv): Conv2d(192, 192, kernel_size=(1, 7), stride=(1, 1), padding=(0, 3),
bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7x3_3): BasicConv2d(
            (conv): Conv2d(192, 192, kernel_size=(7, 1), stride=(1, 1), padding=(3, 0),
bias=False)
            (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
        )
        (branch7x7x3_4): BasicConv2d(

```

```

        (conv): Conv2d(192, 192, kernel_size=(3, 3), stride=(2, 2), bias=False)
        (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
    )
)
(Mixed_7b): InceptionE(
  (branch1x1): BasicConv2d(
    (conv): Conv2d(1280, 320, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_1): BasicConv2d(
    (conv): Conv2d(1280, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_2a): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_2b): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_1): BasicConv2d(
    (conv): Conv2d(1280, 448, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(448, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_2): BasicConv2d(
    (conv): Conv2d(448, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_3a): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_3b): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch_pool): BasicConv2d(
    (conv): Conv2d(1280, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
)

```

```

)
(Mixed_7c): InceptionE(
  (branch1x1): BasicConv2d(
    (conv): Conv2d(2048, 320, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_1): BasicConv2d(
    (conv): Conv2d(2048, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_2a): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3_2b): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_1): BasicConv2d(
    (conv): Conv2d(2048, 448, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(448, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_2): BasicConv2d(
    (conv): Conv2d(448, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_3a): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch3x3dbl_3b): BasicConv2d(
    (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0),
bias=False)
    (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
  (branch_pool): BasicConv2d(
    (conv): Conv2d(2048, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_s
tats=True)
  )
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(dropout): Dropout(p=0.5, inplace=False)

```

```
(fc): Linear(in_features=2048, out_features=4, bias=True)
)
```

```
In [24]: model = models.efficientnet_b0(pretrained=True)
model.classifier[1] = nn.Linear(model.classifier[1].in_features, 4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

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

def train_model(model, criterion, optimizer, scheduler, num_epochs=2):
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        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, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    _, preds = torch.max(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

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

```
Epoch 1/2
train Loss: 0.5493 Acc: 0.7911
val Loss: 0.3344 Acc: 0.8728
Epoch 2/2
train Loss: 0.3029 Acc: 0.8932
val Loss: 0.2358 Acc: 0.9117
```

```
In [25]: import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms as T, models
from torch.utils.data import DataLoader
import os
```

```

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)
        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)
        test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"), 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)

model = models.efficientnet_b0(pretrained=True)
model.classifier[1] = nn.Linear(model.classifier[1].in_features, 4)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

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

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

def train_model(model, criterion, optimizer, scheduler, num_epochs=2):
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        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, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()

```

```

        with torch.set_grad_enabled(phase == 'train'):
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(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

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

```

Epoch 1/2

train Loss: 0.5242 Acc: 0.8020

val Loss: 0.2929 Acc: 0.8916

Epoch 2/2

train Loss: 0.3438 Acc: 0.8844

val Loss: 0.3088 Acc: 0.8963

```

In [26]: def evaluate_model(model, loader, device):
            model.eval()
            running_corrects = 0
            total = 0
            with torch.no_grad():
                for inputs, labels in loader:
                    inputs, labels = inputs.to(device), labels.to(device)
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    running_corrects += torch.sum(preds == labels.data)
                    total += labels.size(0)
            accuracy = running_corrects.double() / total
            print(f"Test Accuracy: {accuracy:.4f}")

            evaluate_model(model, test_loader, device)

```

Test Accuracy: 0.9189

```

In [ ]: import torch
            import torch.nn as nn
            import torch.optim as optim
            from torchvision import models

            model = models.efficientnet_b0(pretrained=True)

            num_classes = 4
            model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)

            device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
            model = model.to(device)

            criterion = nn.CrossEntropyLoss().to(device)

```

```

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

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

def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    """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
            # Iterate over data
            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)
                    loss = criterion(outputs, labels)
                    _, preds = torch.max(outputs, 1) # Extract predictions
                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

```

Epoch 1/20

train Loss: 0.5365 Acc: 0.7962

val Loss: 0.2856 Acc: 0.9058

Epoch 2/20

```

In [ ]: import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, datasets, transforms as T
from torch.utils.data import DataLoader
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import os

```

```

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)
        train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
        return train_loader, len(train_data), train_data.classes
    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)
        test_data = datasets.ImageFolder(os.path.join(data_dir, "test/"), 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), test_data.classes

dataset_path = "new dataset"
batch_size = 32
(train_loader, train_data_len, classes) = get_data_loaders(dataset_path, batch_size, train=True)
(val_loader, test_loader, valid_data_len, test_data_len, classes) = get_data_loaders(dataset_path, batch_size, train=False)

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

num_classes = 4
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

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

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

def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    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)
                loss = criterion(outputs, labels)
                _, preds = torch.max(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

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

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("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, ytickla
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.show()

y_true = []
y_proba = []
y_pred = []
with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

y_true = np.array(y_true)
y_pred = np.array(y_pred)
evaluate_model("efficientnet_b0", y_true, y_pred, classes)

```

Epoch 1/2

train Loss: 0.5334 Acc: 0.7948

val Loss: 0.3759 Acc: 0.8716

Epoch 2/2

```

In [ ]: from torchvision import models
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
import numpy as np

model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.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)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
print("Training completed.")

y_true = []
y_proba = []
y_pred = []
with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1) # Assuming it's a multi-class
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

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

evaluate_model("Resnet18", y_true, y_pred, classes)

```

```

In [ ]: model = models.densenet121(pretrained=False)
model.classifier = nn.Linear(model.classifier.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)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=1)
print("Training completed.")

```

```

y_true = []
y_proba = []
y_pred = []
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1)
        y_proba.extend(probabilities.cpu().numpy())
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

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

evaluate_model("Densenet121", y_true, y_pred, classes)

```

```

In [ ]: model = models.vgg16(pretrained=False)
model.classifier[6] = nn.Linear(model.classifier[6].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)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=1)
print("Training completed.")

y_true = []
y_proba = []
y_pred = []
with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1) # Assuming it's a multi-class
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

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

evaluate_model("vgg16", y_true, y_pred, classes)

InceptionV3
model = models.inception_v3(pretrained=False, aux_logits=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)
if model.aux_logits:
    model.AuxLogits.fc = nn.Linear(model.AuxLogits.fc.in_features, num_classes)
model = model.to(device)

```

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

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

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=1)
print("Training completed.")

y_true = []
y_proba = []
y_pred = []
with torch.no_grad():
    for x, y in tqdm(test_loader):
        output = model(x.to(device))
        y_true.extend(y.numpy())
        probabilities = torch.softmax(output, dim=1) # Assuming it's a multi-class
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

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

evaluate_model("inception_v3", y_true, y_pred, classes)
```

In []: