```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("kanishk3813/pathogen-dataset")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.9), please consider upgrading to the latest versior
Downloading from https://www.kaggle.com/api/v1/datasets/download/kanishk3813/pathogen-dataset?dataset_version_number=1...
100%|██████████| 1.43G/1.43G [00:13<00:00, 109MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/kanishk3813/pathogen-dataset/versions/1
```

```
import shutil

# Source and destination paths
src = "/root/.cache/kagglehub/datasets/kanishk3813/pathogen-dataset"
dst = "/content/dataset/"

# Move the dataset
shutil.move(src, dst)

print("Dataset moved successfully!")
```

```
Dataset moved successfully!
```

```
# Here we will build a model and train them on the data.
# As we have a dataset of 40k images.
# Format of dataset is like we have five folder such as Bacteria, Fungi, Healthy, Pests and Virus.
# Each folder has multiple images
# We will first create train, test and validation dataset from this dataset.
# Then we will train the model on this dataset.
# We will also save the model and the weights of the model.
# We will use pytorch for training the model and tqdm for progress bar.
# Also, we will retrain the model for 10 epochs.
# Importing the required libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torchvision.models as models
from tqdm import tqdm
import os
from PIL import Image
```

```
# Setting the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')
```

```
Using device: cuda
```

```
# Defining the hyperparameters
batch_size = 64
learning_rate = 0.001
num_epochs = 5
scaler = torch.amp.GradScaler()
```

```
# Defining the transforms
# Efficient transformation for faster training
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
# Defining the datasets, We have a dataset folder named dataset. In which we have five folders such as: Bacteria, Fungi, Healthy, Pests
# We will first create train, test and validation dataset from this dataset.
dataset = datasets.ImageFolder(root='/content/dataset/versions/1/pathogen', transform=transform)
dataset
```

```
Dataset ImageFolder
    Number of datapoints: 39997
    Root location: /content/dataset/versions/1/pathogen
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, antialias=True)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

```
# Splitting the dataset into train, test and validation dataset
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size
```

```
# Creating the train, test and validation dataset
train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, val_size, test_size])
```

```
# Create the dataloader for the train and test sets
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
# Print the data statistics
print(f'Train dataset size: {len(train_dataloader.dataset)}')
print(f'Validation dataset size: {len(val_dataloader.dataset)}')
print(f'Test dataset size: {len(test_dataloader.dataset)}')
```

```
Train dataset size: 31997
Validation dataset size: 3999
Test dataset size: 4001
```

```
# Defining the model using pytorch modules.
# We will use ResNeXt-101-32x8d as the backbone.
# We will use CrossEntropyLoss as the loss function.
# We will use Adam as the optimizer.
# Also, we will use GPU for training the model.
# Defining the model
model = models.densenet161(pretrained=True)
num_ftrs = model.classifier.in_features
model.fc = nn.Linear(num_ftrs, 5)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/densenet161-8d451a50.pth" to /root/.cache/torch/hub/checkpoints/densenet161-8d451a
100%|██████████| 110M/110M [00:00<00:00, 173MB/s]
```

```
# Training the model. We will use tqdm for progress bar.
# We will also save the model and the weights of the model.
model.to(device)
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for i, data in tqdm(enumerate(train_dataloader, 0)):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        with torch.autocast(device_type="cuda"):
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss += loss.detach().cpu().item()

        # Memory cleanup
        # del inputs, labels, outputs, loss
        # torch.cuda.empty_cache()

    print(f'Epoch {epoch+1}, Loss: {running_loss/len(train_dataloader)}')
```

```
500it [07:20,  1.14it/s]
Epoch 1, Loss: 0.9734709425568581
500it [07:17,  1.14it/s]
```

```
Epoch 2, Loss: 0.38746731948852536
500it [07:18,  1.14it/s]
Epoch 3, Loss: 0.2548027353435755
500it [07:17,  1.14it/s]
Epoch 4, Loss: 0.1912724271081388
500it [07:17,  1.14it/s]
Epoch 5, Loss: 0.18048720384016634
500it [07:16,  1.15it/s]
Epoch 6, Loss: 0.1394427142776549
500it [07:13,  1.15it/s]
Epoch 7, Loss: 0.11432056523486972
500it [07:10,  1.16it/s]
Epoch 8, Loss: 0.10614096889272333
500it [07:14,  1.15it/s]
Epoch 9, Loss: 0.09346981388702989
500it [07:14,  1.15it/s]Epoch 10, Loss: 0.08432987429574132
```

```
# Testing the model
model.eval()
```

```
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer19): _DenseLayer(
        (norm1): BatchNorm2d(1920, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(1920, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer20): _DenseLayer(
        (norm1): BatchNorm2d(1968, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(1968, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer21): _DenseLayer(
        (norm1): BatchNorm2d(2016, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2016, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer22): _DenseLayer(
        (norm1): BatchNorm2d(2064, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2064, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer23): _DenseLayer(
        (norm1): BatchNorm2d(2112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2112, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer24): _DenseLayer(
        (norm1): BatchNorm2d(2160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2160, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
    )
    (norm5): BatchNorm2d(2208, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (classifier): Linear(in_features=2208, out_features=1000, bias=True)
  (fc): Linear(in_features=2208, out_features=5, bias=True)
)
```

```
# Validation the model
correct = 0
total = 0
with torch.no_grad():
    for data in val_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Accuracy on validation set: {100 * (correct / total)}%')
```

```
Accuracy on validation set: 97.2993248312078%
```

```
# Testing the model
correct = 0
total = 0
with torch.no_grad():
    for data in test_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Accuracy on test set: {100 * (correct / total)}%')
```

```
Accuracy on test set: 96.27593101724568%
```

```
# Evalution using F1_Score
# Check f1 score of densenet161 model
# Check f1 score of densenet161_v1.pth and densenet161_v2.pth model
from sklearn.metrics import f1_score

# Prepare lists for true and predicted labels
y_true = []
y_pred = []

# Iterate through test dataset
for data in test_dataloader:
    images, labels = data
    images, labels = images.to(device), labels.to(device)

    # Get model predictions
    with torch.no_grad():
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

    # Collect results
    y_true.extend(labels.cpu().numpy())
    y_pred.extend(predicted.cpu().numpy())

# Calculate F1 score
f1 = f1_score(y_true, y_pred, average='macro')
print(f'F1 score: {f1}')
```

```
F1 score: 0.9642734012969983
```

```
torch.save(model.state_dict(), 'densenet161_v1.pth')
```

```
# Define class labels
class_labels = ["bacteria", "fungus", "healthy", "pests", "virus"]
# Function to predict image class
def predict_image(image_path):
    # Load image
    image = Image.open(image_path).convert('RGB')

    # Preprocess image
    image = transform(image).unsqueeze(0)  # Add batch dimension
    image = image.to(device)

    # Perform inference
    with torch.no_grad():
        outputs = model(image)
        _, predicted = torch.max(outputs, 1)  # Get class with highest probability
        predicted_class = class_labels[predicted.item()]
    return predicted_class  # Return class index
```

```
# Predict
image_path = "/content/predict/enhanced_image_17.jpg"
predicted_class = predict_image(image_path)
print(f'Predicted class: {predicted_class}')
```

```
Predicted class: virus
```

```
model.load_state_dict(torch.load('densenet161_v1.pth'))
model.to(device)
```

```
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer19): _DenseLayer(
        (norm1): BatchNorm2d(1920, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(1920, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer20): _DenseLayer(
        (norm1): BatchNorm2d(1968, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(1968, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer21): _DenseLayer(
        (norm1): BatchNorm2d(2016, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2016, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer22): _DenseLayer(
        (norm1): BatchNorm2d(2064, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2064, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer23): _DenseLayer(
        (norm1): BatchNorm2d(2112, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2112, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer24): _DenseLayer(
        (norm1): BatchNorm2d(2160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(2160, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(192, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
    )
    (norm5): BatchNorm2d(2208, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (classifier): Linear(in_features=2208, out_features=1000, bias=True)
  (fc): Linear(in_features=2208, out_features=5, bias=True)
)
```

```
# Retraining the model training data and test data
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for i, data in tqdm(enumerate(train_dataloader, 0)):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        with torch.autocast(device_type="cuda"):
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss += loss.detach().cpu().item()

        # Memory cleanup
        del inputs, labels, outputs, loss
        torch.cuda.empty_cache()
    print(f'Epoch {epoch+1}, Loss: {running_loss/len(train_dataloader)}')
```

```
500it [08:41,  1.04s/it]
Epoch 1, Loss: 0.07987985250353813
500it [08:40,  1.04s/it]
Epoch 2, Loss: 0.07459689024463297
500it [08:39,  1.04s/it]
Epoch 3, Loss: 0.06991421092301607
500it [08:41,  1.04s/it]
Epoch 4, Loss: 0.06742898082360625
500it [08:41,  1.04s/it]Epoch 5, Loss: 0.06540205082669855
```

```
# Validation the model
correct = 0
total = 0
with torch.no_grad():
    for data in val_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Accuracy on validation set: {100 * (correct / total)}%')
```

```
Accuracy on validation set: 97.42435608902225%
```

```
# Testing the model
correct = 0
total = 0
with torch.no_grad():
    for data in test_dataloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Accuracy on test set: {100 * (correct / total)}%')
```

```
Accuracy on test set: 97.3756560859785%
```

```
# Evalution using F1_Score
# Check f1 score of densenet161 model
# Check f1 score of densenet161_v1.pth and densenet161_v2.pth model
from sklearn.metrics import f1_score

# Prepare lists for true and predicted labels
y_true = []
y_pred = []

# Iterate through test dataset
for data in test_dataloader:
    images, labels = data
    images, labels = images.to(device), labels.to(device)

    # Get model predictions
    with torch.no_grad():
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

    # Collect results
    y_true.extend(labels.cpu().numpy())
    y_pred.extend(predicted.cpu().numpy())

# Calculate F1 score
f1 = f1_score(y_true, y_pred, average='macro')
print(f'F1 score: {f1}')
```

```
F1 score: 0.9740380802484475
```

```
torch.save(model.state_dict(), 'densenet161_v2.pth')
```

```
# Define class labels
class_labels = ["bacteria", "fungus", "healthy", "pests", "virus"]
# Function to predict image class
def predict_image(image_path):
    # Load image
    image = Image.open(image_path).convert('RGB')
    image = Image.open(image_path).convert("RGB")

    # Preprocess image
    image = transform(image).unsqueeze(0)  # Add batch dimension
    image = image.to(device)

    # Perform inference
    with torch.no_grad():
        outputs = model(image)
        _, predicted = torch.max(outputs, 1)  # Get class with highest probability
        predicted_class = class_labels[predicted.item()]
    return predicted_class  # Return class index
```

```
# Predict
image_path = "/content/predict/Healthy18.jpg"
predicted_class = predict_image(image_path)
print(f'Predicted class: {predicted_class}')
```

```
Predicted class: bacteria
```

Start coding or generate with AI.