```
!cp /content/drive/MyDrive/exai/cv/diseases.zip /content/
```

```
!unzip /content/diseases.zip
```

```
Archive:  /content/diseases.zip
  inflating: diseases/COVID.png
  inflating: diseases/COVID_10.png
  inflating: diseases/COVID_100.png
  inflating: diseases/COVID_101.png
  inflating: diseases/COVID_102.png
  inflating: diseases/COVID_103.png
  inflating: diseases/COVID_104.png
  inflating: diseases/COVID_105.png
  inflating: diseases/COVID_106.png
  inflating: diseases/COVID_107.png
  inflating: diseases/COVID_108.png
  inflating: diseases/COVID_109.png
  inflating: diseases/COVID_11.png
  inflating: diseases/COVID_110.png
  inflating: diseases/COVID_111.png
  inflating: diseases/COVID_112.png
  inflating: diseases/COVID_113.png
  inflating: diseases/COVID_114.png
  inflating: diseases/COVID_115.png
  inflating: diseases/COVID_116.png
  inflating: diseases/COVID_117.png
  inflating: diseases/COVID_118.png
  inflating: diseases/COVID_119.png
  inflating: diseases/COVID_12.png
  inflating: diseases/COVID_120.png
  inflating: diseases/COVID_121.png
  inflating: diseases/COVID_122.png
  inflating: diseases/COVID_123.png
  inflating: diseases/COVID_124.png
  inflating: diseases/COVID_125.png
  inflating: diseases/COVID_126.png
  inflating: diseases/COVID_127.png
  inflating: diseases/COVID_128.png
  inflating: diseases/COVID_129.png
  inflating: diseases/COVID_13.png
  inflating: diseases/COVID_130.png
  inflating: diseases/COVID_131.png
  inflating: diseases/COVID_132.png
  inflating: diseases/COVID_133.png
  inflating: diseases/COVID_134.png
  inflating: diseases/COVID_135.png
  inflating: diseases/COVID_136.png
  inflating: diseases/COVID_137.png
  inflating: diseases/COVID_138.png
  inflating: diseases/COVID_139.png
  inflating: diseases/COVID_14.png
  inflating: diseases/COVID_140.png
  inflating: diseases/COVID_141.png
  inflating: diseases/COVID_142.png
  inflating: diseases/COVID_143.png
  inflating: diseases/COVID_144.png
  inflating: diseases/COVID_145.png
  inflating: diseases/COVID_146.png
  inflating: diseases/COVID_147.png
  inflating: diseases/COVID_148.png
  inflating: diseases/COVID_149.png
  inflating: diseases/COVID_15.png
```

```
import os
import torch
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
from torchvision.models import densenet121
```

```
# Set up data transformation
data_transform = transforms.Compose([
    transforms.Resize((224,224)), # resize the image to (224,224)
    transforms.ToTensor(), # convert the image to tensor
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # normalize the image
])

# Create the ImageFolder dataset
dataset = ImageFolder('/content/diseases/train/', transform=data_transform)

# Create the DataLoader
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

```
# Load the Densenet model
model = densenet121(pretrained=True)

# Replace the classifier with a new classifier
num_ftrs = model.classifier.in_features
model.classifier = nn.Linear(num_ftrs, 3)
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since (
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to /root/.cache/torch/hub/checkpoints/densenet121-a639ec97
```

100% 30.8M/30.8M [00:00<00:00, 67.1MB/s]

```
# Set up the optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
```

```
# Set the number of epochs

model.to('cuda')
num_epochs = 10

from tqdm import tqdm

# Start the training loop
for epoch in range(num_epochs):
    # Initialize a progress bar
    progress_bar = tqdm(dataloader, desc='Epoch {}'.format(epoch+1))
    total=0
    accuracy=0
    correct =0
    for images, labels in progress_bar:
        # Move the data to the GPU if available
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()


        # Zero out the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(images)
        loss = loss_fn(outputs, labels)

        # Backward pass
        loss.backward()
        optimizer.step()

        # Calculate the accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        accuracy = 100 * correct / total
        # Print the loss and accuracy after each epoch
        print('Epoch {}: Loss = {:.4f}, Accuracy = {:.2f}%'.format(epoch+1, loss.item(), accuracy))

        # Update the progress bar
        progress_bar.set_postfix(loss='{:.4f}'.format(loss.item()), accuracy='{:.2f}%'.format(accuracy))
```

```
Epoch 1:   4%|▍         | 1/26 [00:00<00:13,  1.92it/s, accuracy=96.88%, loss=0.1016]Epoch 1: Loss = 0.1016, Accuracy = 96.88%
Epoch 1:   8%|▊         | 2/26 [00:00<00:11,  2.09it/s, accuracy=98.44%, loss=0.0233]Epoch 1: Loss = 0.0233, Accuracy = 98.44%
Epoch 1:  12%|█▏        | 3/26 [00:01<00:10,  2.14it/s, accuracy=97.92%, loss=0.0419]Epoch 1: Loss = 0.0419, Accuracy = 97.92%
Epoch 1:  15%|█▌        | 4/26 [00:01<00:10,  2.18it/s, accuracy=96.88%, loss=0.1179]Epoch 1: Loss = 0.1179, Accuracy = 96.88%
Epoch 1:  19%|█▉        | 5/26 [00:02<00:09,  2.19it/s, accuracy=97.50%, loss=0.0189]Epoch 1: Loss = 0.0189, Accuracy = 97.50%
Epoch 1:  23%|██▎       | 6/26 [00:02<00:09,  2.21it/s, accuracy=97.92%, loss=0.0014]Epoch 1: Loss = 0.0014, Accuracy = 97.92%
Epoch 1:  27%|██▋       | 7/26 [00:03<00:08,  2.19it/s, accuracy=98.21%, loss=0.0135]Epoch 1: Loss = 0.0135, Accuracy = 98.21%
Epoch 1:  31%|███       | 8/26 [00:03<00:08,  2.10it/s, accuracy=98.44%, loss=0.0286]Epoch 1: Loss = 0.0286, Accuracy = 98.44%
Epoch 1:  35%|███▍      | 9/26 [00:04<00:07,  2.13it/s, accuracy=98.26%, loss=0.0815]Epoch 1: Loss = 0.0815, Accuracy = 98.26%
Epoch 1:  38%|███▊      | 10/26 [00:04<00:07,  2.16it/s, accuracy=98.12%, loss=0.0579]Epoch 1: Loss = 0.0579, Accuracy = 98.12%
Epoch 1:  42%|████▏     | 11/26 [00:05<00:06,  2.19it/s, accuracy=98.30%, loss=0.0112]Epoch 1: Loss = 0.0112, Accuracy = 98.30%
Epoch 1:  46%|████▌     | 12/26 [00:05<00:06,  2.21it/s, accuracy=98.44%, loss=0.0208]Epoch 1: Loss = 0.0208, Accuracy = 98.44%
Epoch 1:  50%|█████     | 13/26 [00:05<00:05,  2.22it/s, accuracy=98.56%, loss=0.0343]Epoch 1: Loss = 0.0343, Accuracy = 98.56%
Epoch 1:  54%|█████▍    | 14/26 [00:06<00:05,  2.22it/s, accuracy=98.44%, loss=0.0272]Epoch 1: Loss = 0.0272, Accuracy = 98.44%
Epoch 1:  58%|█████▊    | 15/26 [00:06<00:04,  2.22it/s, accuracy=98.33%, loss=0.0524]Epoch 1: Loss = 0.0524, Accuracy = 98.33%
Epoch 1:  62%|██████▏   | 16/26 [00:07<00:04,  2.22it/s, accuracy=98.24%, loss=0.0458]Epoch 1: Loss = 0.0458, Accuracy = 98.24%
Epoch 1:  65%|██████▌   | 17/26 [00:07<00:04,  2.22it/s, accuracy=97.79%, loss=0.0864]Epoch 1: Loss = 0.0864, Accuracy = 97.79%
Epoch 1:  69%|██████▉   | 18/26 [00:08<00:03,  2.22it/s, accuracy=97.74%, loss=0.2044]Epoch 1: Loss = 0.2044, Accuracy = 97.74%
Epoch 1:  73%|███████▎  | 19/26 [00:08<00:03,  2.23it/s, accuracy=97.86%, loss=0.0068]Epoch 1: Loss = 0.0068, Accuracy = 97.86%
Epoch 1:  77%|███████▋  | 20/26 [00:09<00:02,  2.23it/s, accuracy=97.97%, loss=0.0216]Epoch 1: Loss = 0.0216, Accuracy = 97.97%
Epoch 1:  81%|████████  | 21/26 [00:09<00:02,  2.23it/s, accuracy=98.07%, loss=0.0177]Epoch 1: Loss = 0.0177, Accuracy = 98.07%
Epoch 1:  85%|████████▍ | 22/26 [00:10<00:01,  2.24it/s, accuracy=97.87%, loss=0.0612]Epoch 1: Loss = 0.0612, Accuracy = 97.87%
```

```
Epoch 1:  88%|████████▊ | 23/26 [00:10<00:01,  2.24it/s, accuracy=97.96%, loss=0.0062]Epoch 1: Loss = 0.0062, Accuracy = 97.96%
Epoch 1:  92%|█████████▏| 24/26 [00:10<00:00,  2.22it/s, accuracy=98.05%, loss=0.0255]Epoch 1: Loss = 0.0255, Accuracy = 98.05%
Epoch 1:  96%|█████████▌| 25/26 [00:11<00:00,  2.21it/s, accuracy=98.12%, loss=0.0083]Epoch 1: Loss = 0.0083, Accuracy = 98.12%
Epoch 1: 100%|██████████| 26/26 [00:11<00:00,  2.23it/s, accuracy=98.17%, loss=0.0124]
Epoch 1: Loss = 0.0124, Accuracy = 98.17%
Epoch 2:   4%|▍         | 1/26 [00:00<00:11,  2.25it/s, accuracy=100.00%, loss=0.0272]Epoch 2: Loss = 0.0272, Accuracy = 100.00%
Epoch 2:   8%|▊         | 2/26 [00:00<00:10,  2.24it/s, accuracy=100.00%, loss=0.0069]Epoch 2: Loss = 0.0069, Accuracy = 100.00%
Epoch 2:  12%|█▏        | 3/26 [00:01<00:10,  2.25it/s, accuracy=100.00%, loss=0.0369]Epoch 2: Loss = 0.0369, Accuracy = 100.00%
Epoch 2:  15%|█▌        | 4/26 [00:01<00:09,  2.23it/s, accuracy=99.22%, loss=0.0556]Epoch 2: Loss = 0.0556, Accuracy = 99.22%
Epoch 2:  19%|█▉        | 5/26 [00:02<00:09,  2.23it/s, accuracy=99.38%, loss=0.0016]Epoch 2: Loss = 0.0016, Accuracy = 99.38%
Epoch 2:  23%|██▎       | 6/26 [00:02<00:08,  2.22it/s, accuracy=99.48%, loss=0.0088]Epoch 2: Loss = 0.0088, Accuracy = 99.48%
Epoch 2:  27%|██▋       | 7/26 [00:03<00:08,  2.22it/s, accuracy=99.55%, loss=0.0042]Epoch 2: Loss = 0.0042, Accuracy = 99.55%
Epoch 2:  31%|███       | 8/26 [00:03<00:08,  2.23it/s, accuracy=99.22%, loss=0.0515]Epoch 2: Loss = 0.0515, Accuracy = 99.22%
Epoch 2:  35%|███▍      | 9/26 [00:04<00:07,  2.23it/s, accuracy=99.31%, loss=0.0179]Epoch 2: Loss = 0.0179, Accuracy = 99.31%
Epoch 2:  38%|███▊      | 10/26 [00:04<00:07,  2.23it/s, accuracy=99.38%, loss=0.0072]Epoch 2: Loss = 0.0072, Accuracy = 99.38%
Epoch 2:  42%|████▏     | 11/26 [00:04<00:06,  2.21it/s, accuracy=99.15%, loss=0.0462]Epoch 2: Loss = 0.0462, Accuracy = 99.15%
Epoch 2:  46%|████▌     | 12/26 [00:05<00:06,  2.22it/s, accuracy=99.22%, loss=0.0246]Epoch 2: Loss = 0.0246, Accuracy = 99.22%
Epoch 2:  50%|█████     | 13/26 [00:05<00:05,  2.22it/s, accuracy=99.28%, loss=0.0102]Epoch 2: Loss = 0.0102, Accuracy = 99.28%
Epoch 2:  54%|█████▍    | 14/26 [00:06<00:05,  2.22it/s, accuracy=99.33%, loss=0.0215]Epoch 2: Loss = 0.0215, Accuracy = 99.33%
Epoch 2:  58%|█████▊    | 15/26 [00:06<00:04,  2.20it/s, accuracy=99.38%, loss=0.0153]Epoch 2: Loss = 0.0153, Accuracy = 99.38%
Epoch 2:  62%|██████▏   | 16/26 [00:07<00:04,  2.21it/s, accuracy=99.41%, loss=0.0166]Epoch 2: Loss = 0.0166, Accuracy = 99.41%
Epoch 2:  65%|██████▌   | 17/26 [00:07<00:04,  2.21it/s, accuracy=99.45%, loss=0.0049]Epoch 2: Loss = 0.0049, Accuracy = 99.45%
Epoch 2:  69%|██████▉   | 18/26 [00:08<00:03,  2.21it/s, accuracy=99.48%, loss=0.0016]Epoch 2: Loss = 0.0016, Accuracy = 99.48%
Epoch 2:  73%|███████▎  | 19/26 [00:08<00:03,  2.21it/s, accuracy=99.51%, loss=0.0084]Epoch 2: Loss = 0.0084, Accuracy = 99.51%
Epoch 2:  77%|███████▋  | 20/26 [00:09<00:02,  2.21it/s, accuracy=99.53%, loss=0.0074]Epoch 2: Loss = 0.0074, Accuracy = 99.53%
Epoch 2:  81%|████████  | 21/26 [00:09<00:02,  2.22it/s, accuracy=99.40%, loss=0.0355]Epoch 2: Loss = 0.0355, Accuracy = 99.40%
Epoch 2:  85%|████████▍ | 22/26 [00:09<00:01,  2.21it/s, accuracy=99.43%, loss=0.0044]Epoch 2: Loss = 0.0044, Accuracy = 99.43%
Epoch 2:  88%|████████▊ | 23/26 [00:10<00:01,  2.21it/s, accuracy=99.46%, loss=0.0024]Epoch 2: Loss = 0.0024, Accuracy = 99.46%
Epoch 2:  92%|█████████▏| 24/26 [00:10<00:00,  2.20it/s, accuracy=99.48%, loss=0.0050]Epoch 2: Loss = 0.0050, Accuracy = 99.48%
Epoch 2:  96%|█████████▌| 25/26 [00:11<00:00,  2.20it/s, accuracy=99.50%, loss=0.0238]Epoch 2: Loss = 0.0238, Accuracy = 99.50%
Epoch 2: 100%|██████████| 26/26 [00:11<00:00,  2.25it/s, accuracy=99.51%, loss=0.0154]
Epoch 2: Loss = 0.0154, Accuracy = 99.51%
Epoch 3:   4%|▍         | 1/26 [00:00<00:11,  2.19it/s, accuracy=100.00%, loss=0.0035]Epoch 3: Loss = 0.0035, Accuracy = 100.00%
Epoch 3:   8%|▊         | 2/26 [00:00<00:11,  2.18it/s, accuracy=100.00%, loss=0.0024]Epoch 3: Loss = 0.0024, Accuracy = 100.00%
Epoch 3:  12%|█▏        | 3/26 [00:01<00:10,  2.19it/s, accuracy=100.00%, loss=0.0043]Epoch 3: Loss = 0.0043, Accuracy = 100.00%
Epoch 3:  15%|█▌        | 4/26 [00:01<00:10,  2.18it/s, accuracy=100.00%, loss=0.0066]Epoch 3: Loss = 0.0066, Accuracy = 100.00%
```

```python
# Set up data transformation for the testing dataset
test_transform = transforms.Compose([
    transforms.Resize((224,224)), # resize the image to (224,224)
    transforms.ToTensor(), # convert the image to tensor
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # normalize the image
])

# Create the testing dataset
test_dataset = ImageFolder('//content/diseases/test/', transform=test_transform)

# Create the testing dataloader
test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```python
# Set the model to evaluation mode
model.eval()
```

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
```

```
        (denselayer5): _DenseLayer(
          (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
        (denselayer6): _DenseLayer(
          (norm1): BatchNorm2d(224, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu1): ReLU(inplace=True)
          (conv1): Conv2d(224, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu2): ReLU(inplace=True)
          (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        )
      )
      (transition1): _Transition(
        (norm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
# Initialize variables for loss and accuracy
test_loss = 0
test_correct = 0
test_total = 0

# Iterate over the testing dataloader
for images, labels in test_dataloader:
    # Move the data to the GPU if available
    if torch.cuda.is_available():
        images = images.cuda()
        labels = labels.cuda()

    # Forward pass
    outputs = model(images)
    loss = loss_fn(outputs, labels)

    # Calculate the loss and accuracy
    test_loss += loss.item()
    _, predicted = torch.max(outputs.data, 1)
    test_total += labels.size(0)
    test_correct += (predicted == labels).sum().item()

# Calculate the average loss and accuracy
test_loss = test_loss / len(test_dataloader)
test_accuracy = 100 * test_correct / test_total

# Print the loss and accuracy
print('Test Loss: {:.4f}'.format(test_loss))
print('Test Accuracy: {:.2f}%'.format(test_accuracy))
```

```
Test Loss: 0.1359
Test Accuracy: 98.77%
```

```
from PIL import Image

# Load the image and apply the data transformation
image = Image.open('/content/diseases/test/NORMAL/NORMAL_105.png')
image_transform = transforms.Compose([
    transforms.Resize((224,224)), # resize the image to (224,224)
    transforms.ToTensor(), # convert the image to tensor
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # normalize the image
])
image = image_transform(image).unsqueeze(0)

# Set the model to evaluation mode
model.eval()
```

```
        (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer14): _DenseLayer(
        (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer15): _DenseLayer(
        (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
      (denselayer16): _DenseLayer(
        (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      )
    )
    (norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (classifier): Linear(in_features=1024, out_features=3, bias=True)
)
```

```
if torch.cuda.is_available():
    image = image.cuda()

# Set the model to evaluation mode
model.eval()

# Predict the class of the image
outputs = model(image)
_, predicted = torch.max(outputs.data, 1)

# Convert the predicted class index to a class label
predicted_label = dataset.classes[predicted.item()]

# Print the predicted class label
print('Predicted Class: {}'.format(predicted_label))
```

```
Predicted Class: NORMAL
```

```
from sklearn.metrics import f1_score, classification_report
```

```
# Initialize variables for true labels and predicted labels
true_labels = []
predicted_labels = []

# Iterate over the testing dataloader
for images, labels in test_dataloader:
    # Move the data to the GPU if available
    if torch.cuda.is_available():
        images = images.cuda()
        labels = labels.cuda()

    # Forward pass
    outputs = model(images)
    _, predicted = torch.max(outputs.data, 1)

    # Convert the labels and predictions to numpy arrays
    labels = labels.cpu().numpy()
    predicted = predicted.cpu().numpy()

    # Append the labels and predictions to the list
    true_labels.extend(labels)
    predicted_labels.extend(predicted)
```

```
# Calculate the F1 score
f1 = f1_score(true_labels, predicted_labels, average='weighted')

# Print the classification report
print(classification_report(true_labels, predicted_labels))

# Print the F1 score
```

```
print('F1 Score: {:.2f}'.format(f1))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      1.00      1.00        38
           2       1.00      0.96      0.98        24

    accuracy                           0.99        81
   macro avg       0.98      0.99      0.98        81
weighted avg       0.99      0.99      0.99        81

F1 Score: 0.99
```

```
# Save the model and optimizer state dictionaries
torch.save({
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict()
}, '/content/drive/MyDrive/exai/cv/densenet_xray.pt')
```

```
# Load the model and optimizer state dictionaries
checkpoint = torch.load('/content/drive/MyDrive/exai/cv/densenet_xray.pt')
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

# Set the model to evaluation mode
model.eval()

# Predict the class of an image
outputs = model(image)
_, predicted = torch.max(outputs.data, 1)

# Convert the predicted class index to a class label
predicted_label = dataset.classes[predicted.item()]

# Print the predicted class label
print('Predicted Class: {}'.format(predicted_label))
```

```
Predicted Class: NORMAL
```

```
import os
import random


# Specify the directory containing the images
image_dir = '/content/diseases/'

# Get the list of all images in the directory
image_filenames = os.listdir(image_dir)

# Shuffle the list of images
random.shuffle(image_filenames)

# Split the list into a training set and a testing set
num_train = int(len(image_filenames) * 0.91)
train_filenames = image_filenames[:num_train]
test_filenames = image_filenames[num_train:]

# Create the training and testing directories
train_dir = os.path.join(image_dir, 'train')
test_dir = os.path.join(image_dir, 'test')
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

# Create subdirectories in the training directory for each class
os.makedirs(os.path.join(train_dir, 'COVID'), exist_ok=True)
os.makedirs(os.path.join(train_dir, 'NORMAL'), exist_ok=True)
os.makedirs(os.path.join(train_dir, 'PNEUMONIA'), exist_ok=True)
os.makedirs(os.path.join(test_dir, 'COVID'), exist_ok=True)
os.makedirs(os.path.join(test_dir, 'NORMAL'), exist_ok=True)
os.makedirs(os.path.join(test_dir, 'PNEUMONIA'), exist_ok=True)
# Copy the training images to the appropriate subdirectories in the training set
for filename in train_filenames:
    src = os.path.join(image_dir, filename)
    if 'COVID' in filename:
        dst = os.path.join(train_dir, 'COVID', filename)
    elif 'NORMAL' in filename:
        dst = os.path.join(train_dir, 'NORMAL', filename)
    elif 'PNEUMONIA' in filename:
        dst = os.path.join(train_dir, 'PNEUMONIA', filename)
    os.symlink(src, dst)

for filename in test_filenames:
    src = os.path.join(image_dir, filename)
    if 'COVID' in filename:
```

```
        dst = os.path.join(test_dir, 'COVID', filename)
    elif 'NORMAL' in filename:
        dst = os.path.join(test_dir, 'NORMAL', filename)
    elif 'PNEUMONIA' in filename:
        dst = os.path.join(test_dir, 'PNEUMONIA', filename)
    os.symlink(src, dst)

# Copy the testing images to the testing directory
```