```
import torch
import torchvision.models as models
from torchvision import datasets
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from PIL import ImageFile
from PIL import Image
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import os


train_data_dir = '/content/drive/My Drive/Colab Notebooks/covid/train'
valid_data_dir = '/content/drive/My Drive/Colab Notebooks/covid/validation'
test_data_dir = '/content/drive/My Drive/Colab Notebooks/covid/test'


# num_class = 2
# image_dim = 224
batch_size = 8


ImageFile.LOAD_TRUNCATED_IMAGES = True

train_data_transform = transforms.Compose([transforms.Resize(244),
                                           transforms.CenterCrop(224),
                                           transforms.RandomRotation(10),
                                           transforms.RandomHorizontalFlip(),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean=0.485,
                                                                std=0.229)])

valid_data_transform = transforms.Compose([transforms.Resize(244),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean= 0.485 ,std= 0.229)])

test_data_transform = transforms.Compose([transforms.Resize(224),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor(),
                                           transforms.Normalize(mean= 0.485,
                                                                std= 0.229)])

train_data = datasets.ImageFolder(train_data_dir, transform=train_data_transform)
valid_data = datasets.ImageFolder(valid_data_dir, transform=valid_data_transform)
test_data = datasets.ImageFolder(test_data_dir, transform=test_data_transform)
train_data_load = DataLoader(train_data, batch_size = batch_size, shuffle = True)
valid_data_load = DataLoader(valid_data, batch_size = batch_size, shuffle = True)
test_data_load = DataLoader(test_data,batch_size=batch_size, shuffle = True)


use_cuda = torch.cuda.is_available()
```

```
len(valid_data_load)
```

```
3
```

```
loaders_transfer = {'train': train_data_load, 'valid': valid_data_load, 'test': test_data_
```

```
model_transfer = models.resnet50(pretrained=True)
model_transfer.out = nn.Linear(2048, 2)

if use_cuda:
    model_transfer = model_transfer.cuda()
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/.ca
100%                                    97.8M/97.8M [00:01<00:00, 65.9MB/s]
```

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer.parameters(), lr=0.001)
```

```
def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
    """returns trained model"""
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf

    for epoch in range(1, n_epochs+1):
        train_loss = 0.0
        valid_loss = 0.0

        model.train()
        for batch_idx, (data, target) in enumerate(loaders['train']):
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            optimizer.zero_grad()
            out = model(data)

            loss = criterion(out, target)

            loss.backward()
            optimizer.step()
            train_loss += ((1/(batch_idx + 1)) * (loss.data - train_loss))

        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['valid']):
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            out = model(data)

            loss = criterion(out, target)
            valid_loss += ((1/(batch_idx + 1)) * (loss.data - valid_loss))
        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.5f} \tValidation Loss: {:.5f}'.format(
            epoch,
            train_loss,
            valid_loss
            ))

        #save the model if validation loss has decreased
        if valid_loss <= valid_loss_min:
            print('Valid loss decrease to  ({:.5f} -> {:.5f}). Saving model ...'.format(va
            torch.save(model.state_dict(), save_path)
            valid_loss_min = valid_loss

    # return trained model
    return model


n_epochs = 15
model_transfer = train(n_epochs, loaders_transfer, model_transfer, optimizer_transfer, cri
```

```
Epoch: 1        Training Loss: 4.05990  Validation Loss: 9.77360
Valid loss decrease to  (inf -> 9.77360). Saving model ...
Epoch: 2        Training Loss: 2.02961  Validation Loss: 59.61674
Epoch: 3        Training Loss: 0.23425  Validation Loss: 20.61061
Epoch: 4        Training Loss: 0.65071  Validation Loss: 58.60395
Epoch: 5        Training Loss: 0.51080  Validation Loss: 45.53096
Epoch: 6        Training Loss: 0.11861  Validation Loss: 16.00736
Epoch: 7        Training Loss: 0.40095  Validation Loss: 7.90404
Valid loss decrease to  (9.77360 -> 7.90404). Saving model ...
Epoch: 8        Training Loss: 0.08909  Validation Loss: 12.49787
Epoch: 9        Training Loss: 0.09364  Validation Loss: 4.46481
Valid loss decrease to  (7.90404 -> 4.46481). Saving model ...
Epoch: 10       Training Loss: 0.23590  Validation Loss: 6.51011
Epoch: 11       Training Loss: 0.11580  Validation Loss: 10.81141
Epoch: 12       Training Loss: 0.15264  Validation Loss: 6.31561
Epoch: 13       Training Loss: 0.22430  Validation Loss: 6.24716
Epoch: 14       Training Loss: 0.05406  Validation Loss: 1.46298
Valid loss decrease to  (4.46481 -> 1.46298). Saving model ...
Epoch: 15       Training Loss: 0.01664  Validation Loss: 1.14779
Valid loss decrease to  (1.46298 -> 1.14779). Saving model ...
```

```
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

```
<All keys matched successfully>
```

```
def test(loaders, model, criterion, use_cuda):

    test_loss = 0.
    correct = 0.
    total = 0.

    model.eval()
    for batch_idx, (data, target) in enumerate(loaders['test']):
        if use_cuda:
            data, target = data.cuda(), target.cuda()
        output = model(data)
        loss = criterion(output, target)
        loss = criterion(output, target)
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
        pred = output.data.max(1, keepdim=True)[1]
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())
        total += data.size(0)

    print('Test Loss: {:.6f}\n'.format(test_loss))

    print('\nTest Accuracy: %2d%% (%2d/%2d)' % (
        100. * correct / total, correct, total))


test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.883075


Test Accuracy: 94% (17/18)
```