```python
# Author: Daiwei (David) Lu
# Train custom model

from torch.utils.data import Dataset
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import time
import copy
from load import *
from model import Net
from utils import visualize_model

import warnings

warnings.filterwarnings("ignore")
plt.ion()


def train_model(model, criterion, optimizer, scheduler, dataloaders, device, num_epochs=25):
    dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
    class_names = ['x', 'y']
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = 9001.
    best_acc = 1.
    train_loss = []
    val_loss = []
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()  # Set model to evaluate mode
            running_loss = 0.0
            running_corrects = 0.0
            # Iterate over data.
            for loader in dataloaders[phase]:
                inputs, labels = loader['image'], loader['coordinates']
                inputs = inputs.float().cuda().to(device)
                labels = labels.float().cuda().to(device)
                # zero the parameter gradients
                optimizer.zero_grad()
                # forward
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                # backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += calc_acc(labels, outputs)
```

```
58            if phase == 'train':
59                scheduler.step()
60
61            epoch_loss = running_loss / dataset_sizes[phase]
62            epoch_acc = running_corrects / dataset_sizes[phase]
63
64            print('{} Loss: {:.6f}'.format(
65                phase, epoch_loss))
66            print('{} Acc: {:.6f}'.format(
67                phase, epoch_acc
68            ))
69            if phase == 'train':
70                train_loss.append(epoch_loss)
71            else:
72                val_loss.append(epoch_loss)
73
74            # deep copy the model
75            if phase == 'val' and epoch_acc < best_acc:
76                best_loss = epoch_loss
77                best_acc = epoch_acc
78                best_model_wts = copy.deepcopy(model.state_dict())
79
80        print()
81
82    time_elapsed = time.time() - since
83    print('Training complete in {:.0f}m {:.0f}s'.format(
84        time_elapsed // 60, time_elapsed % 60))
85    print('Best val Loss: {:6f}'.format(best_loss))
86    print('Best val Acc: {:6f}'.format(best_acc))
87
88    # load best model weights
89    model.load_state_dict(best_model_wts)
90    return model, train_loss, val_loss
91
92
93  image_datasets = {'train': PhoneDataset('labels/train.txt',
94                                           '',
95                                           mode='train',
96                                           transform=transforms.Compose([
97                                               Rescale(256),
98                                               RandomVerticalFlip(0.5),
99                                               RandomHorizontalFlip(0.5),
100                                              RandomColorJitter(0.9),
101                                              ToTensor(),
102                                              Normalize()
103                                          ])),
104                'val': PhoneDataset('labels/val.txt',
105                                          '',
106                                          mode='validation',
107                                          transform=transforms.Compose([
108                                              Rescale(256),
109                                              # RandomVerticalFlip(0.1),
110                                              # RandomHorizontalFlip(0.1),
111                                              # RandomColorJitter(0.1),
112                                              ToTensor(),
113                                              Normalize()
114                                          ]))}
```

```
115
116
117 def main():
118     dataloaders = {'train': torch.utils.data.DataLoader(image_datasets['train'], batch_size=16,
119                                                         shuffle=True),
120                    'val': torch.utils.data.DataLoader(image_datasets['val'], batch_size=4,
121                                                       shuffle=True)}
122
123     device = torch.device("cuda")
124
125     model = Net()
126     model = model.to(device)
127
128     criterion = nn.MSELoss()
129
130     optimizer_conv = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
131
132     # Decay LR by a factor of 0.5 every 20 epochs
133     exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=20, gamma=0.5)
134
135     print(model)
136     epochs = 100
137     model, train_loss, val_loss = train_model(model, criterion, optimizer_conv,
138                                               exp_lr_scheduler, dataloaders, device, num_epochs=epochs)
139
140     visualize_model(model, dataloaders, device)
141     plt.ioff()
142     plt.show()
143
144     plt.plot(np.arange(epochs), train_loss, c='red', label='Training loss')
145     plt.plot(np.arange(epochs), val_loss, c='blue', label='Validation loss')
146     plt.legend()
147     plt.title('Loss Curve')
148     plt.xlabel('Epochs')
149     plt.ylabel('Loss')
150     plt.savefig('./Loss Curve')
151
152     torch.save(model.state_dict(), './trainedmodel.pth')
153
154
155 if __name__ == '__main__':
156     main()
157
```

```python
# Author: Daiwei (David) Lu
# A custom network in VGG style

import torch
from torch import nn


class ConvPool(nn.Module):
    def __init(self, channelin, channelout):
        super(Net, self).__init__()
        self.convpool = nn.Sequential(
            nn.Conv2d(channelin, channelout, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
        )

    def forward(self, x):
        return self.convpool(x)


class Conv(nn.Module):
    def __init(self, channelin, channelout):
        super(Net, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(channelin, channelout, 3, padding=1),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(128, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(256, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2))
```

```
58          self.avgpool = nn.AdaptiveAvgPool2d(output_size=(7, 7))
59          self.classifier = nn.Sequential(nn.Linear(512 * 7 * 7, 64),
60                                          nn.Sigmoid(),
61                                          nn.Dropout(0.25),
62                                          nn.Linear(64, 2))
63
64      def forward(self, x):
65          # ModuleList can act as an iterable, or be indexed using ints
66          x1 = self.features(x)
67          x2 = self.avgpool(x1)
68          x3 = torch.flatten(x2, 1)
69          x4 = self.classifier(x3)
70          return x4
71
```

```python
# Author: Daiwei (David) Lu
# A fully custom dataloader for the cellphone dataset

import os
import torch
import pandas as pd
from PIL import Image
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset
from torchvision import transforms
import torchvision.transforms.functional as TF
import random

import warnings

warnings.filterwarnings("ignore")
plt.ion()


class PhoneDataset(Dataset):

    def __init__(self, file, root, mode='/train', transform=None, test=False):
        self.data = pd.read_csv(file, sep=" ", header=None)
        if test:
            self.data.columns = ["names"]
        else:
            self.data.columns = ["names", "x", "y"]
        self.root = root
        self.transform = transform
        self.mode = mode

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        img_name = os.path.join(self.root + self.mode, self.data.names[idx])
        image = io.imread(img_name)
        coords = np.array([self.data.x[idx], self.data.y[idx]])
        sample = {'image': image, 'coordinates': coords}
        if self.transform:
            sample = self.transform(sample)
        return sample


class Rescale(object):

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, coordinates = sample['image'], sample['coordinates']
        new_h, new_w = self.output_size, self.output_size
```

```
58          img = transform.resize(image, (new_h, new_w))
59          return {'image': img, 'coordinates': coordinates}
60
61
62  class Normalize(object):
63      def __init__(self, inplace=False):
64          self.mean = (0.5692824, 0.55365936, 0.5400631)
65          self.std = (0.1325967, 0.1339596, 0.14305606)
66          self.inplace = inplace
67
68      def __call__(self, sample):
69          image, coordinates = sample['image'], sample['coordinates']
70          return {'image': TF.normalize(image, self.mean, self.std, self.inplace), 'coordinates':
    coordinates,
71                  'original': image}
72
73
74  class ToTensor(object):
75
76      def __call__(self, sample):
77          dtype = torch.FloatTensor if torch.cuda.is_available() else torch.FloatTensor
78          image, coordinates = sample['image'], sample['coordinates']
79          image = image.transpose((2, 0, 1))
80          return {'image': torch.from_numpy(image).type(dtype),
81                  'coordinates': torch.from_numpy(coordinates).type(dtype)}
82
83
84  class RandomHorizontalFlip(object):
85      def __init__(self, p=0.5):
86          self.p = p
87
88      def __call__(self, sample):
89          image, coordinates = sample['image'], sample['coordinates']
90          if random.random() < self.p:
91              image *= 255
92              image = Image.fromarray(np.uint8(image))
93              image = TF.hflip(image)
94              image = np.array(image)
95              image = np.float32(image) / 255.
96              coordinates[0] = 1.0 - coordinates[0]
97          return {'image': image, 'coordinates': coordinates}
98
99
100 class RandomVerticalFlip(object):
101     def __init__(self, p=0.5):
102         self.p = p
103
104     def __call__(self, sample):
105         image, coordinates = sample['image'], sample['coordinates']
106         if random.random() < self.p:
107             image *= 255.
108             image = Image.fromarray(np.uint8(image))
109             image = TF.vflip(image)
110             image = np.array(image)
111             image = np.float32(image) / 255.
112             coordinates[1] = 1.0 - coordinates[1]
113         return {'image': image, 'coordinates': coordinates}
```

```
114
115
116 class RandomColorJitter(object):
117     def __init__(self, p=0.2, brightness=(0.5, 1.755), contrast=(0.5, 1.5), saturation=(0.5, 1.5), hue
    =(-0.2, 0.2)):
118         self.p = p
119         self.brightness = brightness
120         self.contrast = contrast
121         self.saturation = saturation
122         self.hue = hue
123
124     def __call__(self, sample):
125         image, coordinates = sample['image'], sample['coordinates']
126         if random.random() < self.p:
127             image *= 255.
128             image = Image.fromarray(np.uint8(image))
129             modifications = []
130
131             brightness_factor = random.uniform(self.brightness[0], self.brightness[1])
132             modifications.append(transforms.Lambda(lambda img: TF.adjust_brightness(image,
    brightness_factor)))
133
134             contrast_factor = random.uniform(self.contrast[0], self.contrast[1])
135             modifications.append(transforms.Lambda(lambda img: TF.adjust_contrast(image,
    contrast_factor)))
136
137             saturation_factor = random.uniform(self.saturation[0], self.saturation[1])
138             modifications.append(transforms.Lambda(lambda img: TF.adjust_saturation(image,
    saturation_factor)))
139
140             hue_factor = random.uniform(self.hue[0], self.hue[1])
141             modifications.append(transforms.Lambda(lambda img: TF.adjust_hue(image, hue_factor
    )))
142
143             random.shuffle(modifications)
144             modification = transforms.Compose(modifications)
145             image = modification(image)
146
147             image = np.array(image)
148             image = np.float32(image) / 255.
149         return {'image': image, 'coordinates': coordinates}
150
151
152 def calc_acc(input, output):
153     sum = 0.0
154     for i in range(input.shape[0]):
155         sum += torch.sqrt((input[i][0] - output[i][0]) ** 2 + (input[i][1] - output[i][1]) ** 2)
156     return sum / input.shape[0]
157
```

```python
# Author: Daiwei (David) Lu
# Find the PhoneDataset mean and std for normalization

from load import *
from torch.utils.data import DataLoader

dataset = PhoneDataset('data/labels/train.txt',
                       'data',
                       mode='/train',
                       transform=transforms.Compose([
                          Rescale(256),
                          ToTensor(),
                          Normalize()
                       ]))
loader = DataLoader(
    dataset,
    batch_size=10,
    num_workers=1,
    shuffle=False
)

pixel_mean = np.zeros(3)
pixel_std = np.zeros(3)
k = 1
for load in loader:
    imgs = load['image']
    imgs = np.array(imgs)
    print(imgs.shape)
    for i in range(imgs.shape[0]):
        image = imgs[i]
        pixels = image.reshape((-1, image.shape[2]))

        for pixel in pixels:
            diff = pixel - pixel_mean
            pixel_mean += diff / k
            pixel_std += diff * (pixel - pixel_mean)
            k += 1

pixel_std = np.sqrt(pixel_std / (k - 2))
print(pixel_mean)
print(pixel_std)

```

```python
# Author: Daiwei (David) Lu
# Make predictions on test images

import torch
from skimage import io, transform
from torch.utils.data import Dataset
from torchvision import transforms
from model import Net
from utils import TestFile, Rescale, ToTensor, show_dot
import argparse

MODEL_PATH = './model.pth'


def test_model(path, viz=False):
    image = io.imread(path)
    image = transform.resize(image, (256, 256))

    device = torch.device("cuda")

    model = Net()
    model = model.to(device)

    model.load_state_dict(torch.load(MODEL_PATH))
    model.eval()

    with torch.no_grad():
        set = TestFile(path,
                       transform=transforms.Compose([
                           Rescale(256),
                           ToTensor()
                       ])
                       )
        loader = torch.utils.data.DataLoader(set)
        for i, input in enumerate(loader):
            image = input['image'].float().cuda().to(device)
            coordinates = model(image).data
            coordinates = coordinates.cpu().numpy()
            print('{:.4f} {:.4f}'.format(coordinates[0][0], coordinates[0][1]))
            if viz:
                show_dot(input['original'], coordinates)


def main():
    # Training settings
    parser = argparse.ArgumentParser(description='Test Prediction')
    parser.add_argument('path', metavar='P', type=str,
                        help='path of file for prediction')
    parser.add_argument('viz', metavar='V', type=bool, default=False,
                        help='visualize prediction')
    args = parser.parse_args()
    test_model(args.path, args.viz)


if __name__ == '__main__':
    main()

```

```python
# Author: Daiwei (David) Lu
# Useful Utils

import torch
from skimage import io, transform
from torch.utils.data import Dataset
import matplotlib.pyplot as plt
from torchvision import utils
import torchvision.transforms.functional as TF


class TestFile(Dataset):

    def __init__(self, file, transform=None):
        self.file = file
        self.transform = transform

    def __len__(self):
        return 1

    def __getitem__(self, idx):
        sample = io.imread(self.file)
        if self.transform:
            sample = self.transform(sample)
        return sample


class Rescale(object):

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        new_h, new_w = self.output_size, self.output_size
        img = transform.resize(sample, (new_h, new_w))
        return img


class Normalize(object):
    def __init__(self, inplace=False):
        self.mean = (0.5692824, 0.55365936, 0.5400631)
        self.std = (0.1325967, 0.1339596, 0.14305606)
        self.inplace = inplace

    def __call__(self, sample):
        image = sample['image']
        return {'image': TF.normalize(image, self.mean, self.std, self.inplace),
                'original': image}


class ToTensor(object):

    def __call__(self, sample):
        dtype = torch.FloatTensor if torch.cuda.is_available() else torch.FloatTensor
        image = sample.transpose((2, 0, 1))
        return torch.from_numpy(image).type(dtype)
```

```
58
59
60  def show_dot(image, coordinates):
61      plt.imshow(image)
62      plt.scatter(image.shape[1] * coordinates[0][0], image.shape[0] * coordinates[0][1], marker='.', c
    ='r')
63      plt.pause(0.001)
64
65
66  def batch_show(sample_batched):
67      """Show image for a batch of samples."""
68      images_batch, coordinates_batch = \
69          sample_batched['original'], sample_batched['coordinates']
70      batch_size = len(images_batch)
71      im_size = images_batch.size(2)
72      grid_border_size = 2
73      grid = utils.make_grid(images_batch)
74      plt.imshow(grid.numpy().transpose((1, 2, 0)))
75
76      for i in range(batch_size):
77          plt.scatter(coordinates_batch[i, 0].cpu().numpy() * 256 + i * im_size + (i + 1) *
    grid_border_size,
78                      coordinates_batch[i, 1].cpu().numpy() * 256 + grid_border_size,
79                      marker='.', c='r')
80          plt.title('Batch from dataloader')
81
82
83  def visualize_model(model, dataloaders, device):
84      was_training = model.training
85      model.eval()
86      fig = plt.figure()
87
88      with torch.no_grad():
89          for i, batch in enumerate(dataloaders['val']):
90              inputs, labels = batch['image'], batch['coordinates']
91              inputs = inputs.float().cuda().to(device)
92              print('Label:', batch['coordinates'].data)
93              batch['coordinates'].data = model(inputs).data
94              plt.figure()
95              batch_show(batch)
96              plt.axis('off')
97              plt.ioff()
98              plt.show()
99
100         model.train(mode=was_training)
101
```