人脸识别示例

人脸识别，是基于人的脸部特征信息进行身份识别的一种生物识别技术。通常来说，人脸识别涵盖人脸检测、人脸关键点检测、人脸验证。 一个完整的人脸识别流程通常为:人脸检测, 人脸裁剪，人脸对齐，人脸验证。而人脸识别就是和数据库中已有的人脸进行多次人脸验证，来判断该人脸是否在数据库中。在人脸检测中应用较广的算法就是MTCNN（ Multi-task Cascaded Convolutional Networks）。我们在这个例子中主要介绍人脸验证。

人脸检测

常用的人脸数据集

名称	人数	图片数
PubFig	200	58k+
CelebA	10177	202599
Colorferet	1000+	10000+
MTFL	*	12995
FaceDB	23	1521
LFW	5749	13233
CASIA-Face	500	2500
IMDB-WIKI	*	523051
FDDB	*	2845

其中LFW是人脸验证常用的评价集，其网址为http://vis-www.cs.umass.edu/lfw/ (http://vis-www.cs.umass.edu/lfw/) .在下面的示例中，我们将利用该数据集来进行实验。

经典深度模型

DeepFace

DeepFace: Closing the Gap to Human-Level Performance in Face Verification

DeepFace是最早利用深度学习进行人脸识别的几篇文章之一，发表在2014的CVPR会议上。 其主要流程如下：检测，对齐（校正），提取特征，分类。DeepFace利用一个9层深度网络，模型参数1.2亿个。 该模型在4000多个不同的人，总计440万张带标记的人脸库中进行训练,在LFW上达到了97.35%的人脸验证精度。

DeepFace

DeepID

Deep Learning Face Representation from Predicting 10,000 Classes

DeepID由港中文汤晓鸥团队提出，到现在已有DeepID1,DeepID2,DeepID2+,DeepID3一系列工作。我们简单介绍一下DeepID1(发表在2014CVPR)的思路，其网络模型如下图，其用了10177人，202599张图片用来训练。最终在LFW数据集上达到了97.45%的成绩。

DeepID

FaceNet

FaceNet: A Unified Embedding for Face Recognition and Clustering

该论文发表在2015的CVPR上，该文章最主要的工作是提出了triplet loss,实现了端到端的训练。

FaceNet

triplet loss 定义为

$$L = \sum_{i}^{N} \left[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha \right]_+$$

例子

我们利用LFW数据来进行一个简单的人脸验证实验。我们采用VGG16的网络结构来进行特征提取，然后利用特征之间的欧式距离来判断两张人脸是否属于同一个人。下面重点来介绍以下如何自定义数据类。

自定义数据类

自定义数据类，首先需要继承Dataset类，然后实现**getitem**和**len**两个函数。通常来说，为了节省内存，我们每次只在训练时才读取当前batch的数据。

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset

def getData(root, name):
    with open(os.path.join(root,name), 'r') as file:
        paths = [line.split('\n')[0] for line in file.readlines() if line != '\n']

    classes = list({item.split('/')[0] for item in paths})
    classes.sort()
    class_to_idx = {cls_name: i for i, cls_name in enumerate(classes)}

    instances = [(os.path.join(root,path), class_to_idx[path.split('/')[0]]) for pat
    return instances


class LFW(Dataset):
    def __init__(self, root, name,transform=None, target_transform=None):
        self.transform = transform
        self.target_transform = target_transform
        samples = getData(root,name)
        self.samples = samples
        self.targets = [s[1] for s in samples]

    def __getitem__(self, index):
        path, target = self.samples[index]
        sample = loader(path)
        target = int(target)
        if self.transform is not None:
            sample = self.transform(sample)
        if self.target_transform is not None:
            target = self.target_transform(target)
        return sample, target

    def __len__(self):
        return len(self.samples)
```

网络结构

采用VGG16的网络结构

In [ ]:

```
class VGG16(nn.Module):

    def __init__(self, out_dim):
        super(VGG16, self).__init__()

        self.conv1_1 = nn.Conv2d(3, 64, 3, padding=(1, 1))
        self.conv1_2 = nn.Conv2d(64, 64, 3, padding=(1, 1))
        self.maxpool1 = nn.MaxPool2d((2, 2))

        self.conv2_1 = nn.Conv2d(64, 128, 3, padding=(1, 1))
        self.conv2_2 = nn.Conv2d(128, 128, 3, padding=(1, 1))
        self.maxpool2 = nn.MaxPool2d((2, 2))

        self.conv3_1 = nn.Conv2d(128, 256, 3, padding=(1, 1))
        self.conv3_2 = nn.Conv2d(256, 256, 3, padding=(1, 1))
        self.conv3_3 = nn.Conv2d(256, 256, 3, padding=(1, 1))
        self.maxpool3 = nn.MaxPool2d((2, 2))

        self.conv4_1 = nn.Conv2d(256, 512, 3, padding=(1, 1))
        self.conv4_2 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv4_3 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.maxpool4 = nn.MaxPool2d((2, 2))

        self.conv5_1 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv5_2 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv5_3 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.maxpool5 = nn.MaxPool2d((2, 2))

        self.fc1 = nn.Linear(512 * 7 * 7, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, out_dim)


    def forward(self, x):

        # x.size(0)即为batch_size
        in_size = x.size(0)

        x = self.conv1_1(x)
        x = F.relu(x)
        x = self.conv1_2(x)
        x = F.relu(x)
        x = self.maxpool1(x)

        x = self.conv2_1(x)
        x = F.relu(x)
        x = self.conv2_2(x)
        x = F.relu(x)
        x = self.maxpool2(x)

        x = self.conv3_1(x)
        x = F.relu(x)
        x = self.conv3_2(x)
        x = F.relu(x)
        x = self.conv3_3(x)
        x = F.relu(x)
        x = self.maxpool3(x)

        x = self.conv4_1(x)
        x = F.relu(x)
        x = self.conv4_2(x)
        x = F.relu(x)
        x = self.conv4_3(x)
        x = F.relu(x)
        x = self.maxpool4(x)

        x = self.conv5_1(x)
        x = F.relu(x)
        x = self.conv5_2(x)
        x = F.relu(x)
        x = self.conv5_3(x)
        x = F.relu(x)
        x = self.maxpool5(x)

        x = x.view(in_size, -1)

        x = self.fc1(x)
        x = F.relu(x)
        feature = self.fc2(x)
        x = self.fc3(feature)

        output = F.log_softmax(x, dim=1)

        return feature, output
```

损失函数

考虑到标签是数字，而非one-hot数据，因此采用nll_loss损失函数。

In [ ]:

```
loss = F.nll_loss(output, target)
```

优化算法

In [2]:

```
optimizer = optim.Adagrad(model.parameters(), lr=lr)
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-2-46a5b07104fb> in <module>
----> 1 optimizer = optim.Adagrad(model.parameters(), lr=lr)

NameError: name 'model' is not defined
```

完整示例

In [1]:

```
import os
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
from torchvision import transforms

def getData(root, name):
    with open(os.path.join(root,name), 'r') as file:
        paths = [line.split('\n')[0] for line in file.readlines() if line != '\n']

    classes = list({item.split('/')[0] for item in paths})
    classes.sort()
    class_to_idx = {cls_name: i for i, cls_name in enumerate(classes)}

    instances = [(os.path.join(root,path), class_to_idx[path.split('/')[0]]) for pat
    return instances

def loader(path):
    with open(path, 'rb') as f:
        img = Image.open(f)
        return img.convert('RGB')

class LFW(Dataset):
    def __init__(self, root, name,transform=None, target_transform=None):
        self.transform = transform
        self.target_transform = target_transform
        samples = getData(root,name)
        self.samples = samples
        self.targets = [s[1] for s in samples]

    def __getitem__(self, index):
        path, target = self.samples[index]
        sample = loader(path)
        target = int(target)
        if self.transform is not None:
            sample = self.transform(sample)
        if self.target_transform is not None:
            target = self.target_transform(target)
        return sample, target

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


class VGG16(nn.Module):

    def __init__(self, out_dim):
        super(VGG16, self).__init__()

        self.conv1_1 = nn.Conv2d(3, 64, 3, padding=(1, 1))
        self.conv1_2 = nn.Conv2d(64, 64, 3, padding=(1, 1))
        self.maxpool1 = nn.MaxPool2d((2, 2))

        self.conv2_1 = nn.Conv2d(64, 128, 3, padding=(1, 1))
        self.conv2_2 = nn.Conv2d(128, 128, 3, padding=(1, 1))
        self.maxpool2 = nn.MaxPool2d((2, 2))

        self.conv3_1 = nn.Conv2d(128, 256, 3, padding=(1, 1))
        self.conv3_2 = nn.Conv2d(256, 256, 3, padding=(1, 1))
        self.conv3_3 = nn.Conv2d(256, 256, 3, padding=(1, 1))
        self.maxpool3 = nn.MaxPool2d((2, 2))

        self.conv4_1 = nn.Conv2d(256, 512, 3, padding=(1, 1))
        self.conv4_2 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv4_3 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.maxpool4 = nn.MaxPool2d((2, 2))

        self.conv5_1 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv5_2 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.conv5_3 = nn.Conv2d(512, 512, 3, padding=(1, 1))
        self.maxpool5 = nn.MaxPool2d((2, 2))

        self.fc1 = nn.Linear(512 * 7 * 7, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, out_dim)


    def forward(self, x):

        # x.size(0)即为batch_size
        in_size = x.size(0)

        x = self.conv1_1(x)
        x = F.relu(x)
        x = self.conv1_2(x)
        x = F.relu(x)
        x = self.maxpool1(x)

        x = self.conv2_1(x)
        x = F.relu(x)
        x = self.conv2_2(x)
        x = F.relu(x)
        x = self.maxpool2(x)

        x = self.conv3_1(x)
        x = F.relu(x)
        x = self.conv3_2(x)
        x = F.relu(x)
        x = self.conv3_3(x)
        x = F.relu(x)
        x = self.maxpool3(x)

        x = self.conv4_1(x)
        x = F.relu(x)
        x = self.conv4_2(x)
        x = F.relu(x)
        x = self.conv4_3(x)
        x = F.relu(x)
        x = self.maxpool4(x)

        x = self.conv5_1(x)
        x = F.relu(x)
        x = self.conv5_2(x)
        x = F.relu(x)
        x = self.conv5_3(x)
        x = F.relu(x)
        x = self.maxpool5(x)

        x = x.view(in_size, -1)

        x = self.fc1(x)
        x = F.relu(x)
        feature = self.fc2(x)
        x = self.fc3(feature)

        output = F.log_softmax(x, dim=1)

        return feature, output


def train(model, device, train_loader, optimizer, epoch, out_dim):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        _, output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))

def val(model, device, val_loader, threshold):
    model.eval()
    same = 0
    dif  = 0
    num = 0
    with torch.no_grad():
        for data, target in val_loader:
            data, target = data.to(device), target.to(device)
            features, _ = model(data)
            dst = torch.dist(features[0], features[1], p=2)
            if dst < threshold:
                pred = True
            else:
                pred = False

            if target[0] == target[1]:
                label = True
            else:
                label = False

            if pred == label:
                num += 1
#             if target[0] == target[1]:
#                 same += dst
#             else:
#                 dif += dst
#     print(same)
#     print(dif)
#     print(same/len(val_loader.dataset))
#     print(dif/len(val_loader.dataset))


    print('Threshold: {};accuracy:{:.6f}'.format(threshold, 2*num/len(val_loader.dat
def test(model, device, test_loader, threshold):
    model.eval()
    num = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            features, _ = model(data)
            dst = torch.dist(features[0], features[1], p=2)
            if dst < threshold:
                pred = True
            else:
                pred = False

            if target[0] == target[1]:
                label = True
            else:
                label = False

            if pred == label:
                num += 1
    print('Threshold: {};accuracy:{:.6f}'.format(threshold, 2*num/len(val_loader.dat


def main():
    # 训练参数
    batch_size = 60
    test_batch_size = 2
    epochs = 10
    lr = 0.0001
    device = torch.device("cuda")
    save_path = './vgg16.pt'
    threshold = 0.4
    out_dim = 3391

    transform=transforms.Compose([
        transforms.Resize([224, 224]),
        transforms.ToTensor()
        ])
    train_data = LFW('lfw_funneled/', 'train.txt', transform=transform)
    val_data = LFW('lfw_funneled/', 'val.txt', transform=transform)
    test_data = LFW('lfw_funneled/', 'test.txt', transform=transform)
    train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, sh
    val_loader = torch.utils.data.DataLoader(val_data, batch_size=2)
    test_loader = torch.utils.data.DataLoader(test_data, batch_size=2)

#     model = VGG16(out_dim).to(device)
#     optimizer = optim.Adagrad(model.parameters(), lr=lr)

#     for epoch in range(1, epochs + 1):
#         train(model, device, train_loader, optimizer, epoch, out_dim)
#         #test(model, device, test_loader)

#     torch.save(model, save_path)
    model = torch.load(save_path)
    val(model, device, val_loader, threshold)


if __name__ == '__main__':
    main()
```

```
Threshold: 0.4;accuracy:0.530000
```