import torch **#利用GPU在大规模并行计算上的执行效率高的优势，利用CPU，将一些逻辑复杂性较小的计算部分分配给GPU让其协助完成计算任务。**

import os, glob  **#os模块自带的文件和文件夹操作方法都非常有用。glob是python自己带的一个文件操作相关模块**

import random, csv

import cv2 as cv

from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from PIL import Image

import time

import torchvision

import matplotlib.pyplot as plt

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(device)

class RMBDataset(Dataset):

def \_\_init\_\_(self, root, resize, mode):

**# 根据输入mode的不同将数据集分为训练集，验证集和测试集，根路径，图片大小，模式，分别记为root、resize还有mode。**

super(RMBDataset, self).\_\_init\_\_()

**#这是对继承自父类的属性进行初始化。而且是用父类的初始化方法来初始化继承的属性，self指的是实例本身,在python中创建类后，通常会创建一个 \_\_ init\_\_ ()方法，这个方法会在创建类的实例的时候自动执行。 \_\_ init\_\_ ()方法必须包含一个self参数，而且要是第一个参数，自动运行。**

self.root = root

self.resize = resize

self.mode = mode

self.name2label = {}

**# 给不同类别的图片编号，文件夹名字**

for name in sorted(os.listdir(os.path.join(root))[:6], key=int): **#os.listdir返回指定目录下的所有文件和目录名**

if not os.path.isdir(os.path.join(root, name)): **#os.path.isdir检验给出的路径是否是一个目录，os.path.join拼接工作路径**

continue

self.name2label[name] = len(self.name2label.keys())

**#len() 方法返回对象（字符、列表、元组等）长度或项目个数。keys()返回一个字典所有的键。**

# print(self.name2label[name])

self.images, self.labels = self.load\_csv("images.csv")

if self.mode == 'train':

**# 60% mode=‘train’则生成训练集**

self.images = self.images[: int(0.6 \* len(self.images))]

self.labels = self.labels[: int(0.6 \* len(self.labels))]

elif self.mode == 'val':

**# 20% = 60% -> 80% mode=‘val’则生成训练集**

self.images = self.images[int(0.6 \* len(self.images)): int(0.8 \* len(self.images))]

self.labels = self.labels[int(0.6 \* len(self.labels)): int(0.8 \* len(self.labels))]

else: **# mode=‘test’则生成训练集**

self.images = self.images[int(0.8 \* len(self.images)):]

self.labels = self.labels[int(0.8 \* len(self.labels)):]

**#辅助函数load\_csv来获取图片的路径，以便划分训练集、验证集还有测试集def load\_csv(self, filename):# 加载文件，filename为将要保存的csv文件名**

if not os.path.exists(os.path.join(filename)):

**#检验给出的路径是否真地存在**

images = []

for name in self.name2label.keys():

images += glob.glob(os.path.join(self.root, name, '\*.jpg')) **#获取指定目录下的所有图片**

random.shuffle(images)#方法将序列的所有元素随机排序。

with open(os.path.join(filename), mode='w', newline='') as f: **#一旦出错，后面的 f.close() 就不会调用。所以，为了保证无论是否出错都能正确地关闭文件， with open() 来自动调用close()方法，无论是否出错**

writer = csv.writer(f) **#对.csv文件读取**

for img in images:

name = img.split(os.sep)[-2]

**#: os.path.sep是一个字符串属性，一般是'/'或'\'，.split()的基础作用是分割字符串的。当没有参数输入的时候，它会默认空格**

label = self.name2label[name]

writer.writerow([img, label])

**# 将图片路径和标签写入csv文件**

images, labels = [], []

with open(os.path.join(filename)) as f:

reader = csv.reader(f)

for row in reader:

img, label = row

label = int(label)

images.append(img)

labels.append(label)

assert len(images) == len(labels)

# print(type(images), type(labels[0]))

return images, labels

def denormalize(self, x\_hat): **# 反归一化**

mean = [0.485, 0.456, 0.406]

std = [0.229, 0.224, 0.225]

mean = torch.tensor(mean).unsqueeze(1).unsqueeze(1)

**#unsqueeze给指定的tensor增加一个指定(之前不存在的)的维度**

std = torch.tensor(std).unsqueeze(1).unsqueeze(1)

x = x\_hat \* std + mean

return x

def \_\_len\_\_(self):

return len(self.images)**#实现返回数据集的数据数量。**

def \_\_getitem\_\_(self, idx):

**#\_\_getitem\_\_: 支持根据给定的key来获取数据样本。**

img, label = self.images[idx], self.labels[idx]

tf1 = transforms.Compose([  **# 数据增强**

lambda x : Image.open(x).convert('RGB'),

transforms.Resize((int(self.resize\* 1.25), int(self.resize \* 2))),

transforms.RandomRotation(15),

transforms.CenterCrop(self.resize),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

])**#Resize重置图像分辨率，RandomRotation随机旋转，CenterCrop中心裁剪，ToTensor归一化至[0-1]，Normalize对数据按通道进行标准化，即先减均值，再除以标准差**

tf2 = transforms.Compose([

lambda x : Image.open(x).convert('RGB'),

transforms.Resize((int(self.resize \* 1), int(self.resize \* 1.5))),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

]) **#Resize重置图像分辨率 ，归一化，标准化**

if self.mode == 'test': **# 对于测试集数据使用tf2的方式对图片进行数据增强，该方式主要是为了方便可视化查看。**

img = tf2(img)

else:

img = tf1(img)

label = torch.tensor(label)

return img, label

**# 加载数据集**

train\_data = RMBDataset('RMBDataset', 256, 'train')

val\_data = RMBDataset('RMBDataset', 256, 'val')

test\_data = RMBDataset('RMBDataset', 256, 'test')

**#机器学习模型训练步骤分为：数据，模型，损失函数，优化器，迭代训练**

**#首先是数据，又可以分为：数据收集，数据划分，数据读取，数据预处理**

**#DataLoader就是用来进行数据读取的。shffule=True在表示不同批次的数据遍历时，打乱顺序**

train\_loader = DataLoader(train\_data, batch\_size=32, shuffle=True, pin\_memory=True)

val\_loader = DataLoader(val\_data, batch\_size=32, shuffle=True, pin\_memory=True)

test\_loader = DataLoader(test\_data, batch\_size=32, shuffle=True, pin\_memory=True)

**## 可视化部分训练图像，以便了解数据扩充**

%matplotlib inline

import numpy as np

classes = ['RMB ¥1', 'RMB ¥5', 'RMB ¥10', 'RMB ¥20', 'RMB ¥50', 'RMB ¥100']

images, labels = next(iter(test\_loader))

def imshow(img):

img = train\_data.denormalize(img).numpy()

img = np.transpose(img, (1, 2, 0))

plt.imshow((img\*255).astype('uint8'))

fig = plt.figure(figsize=(25, 4))

for i in range(8):

plt.subplot(2, 4, i+1)

plt.axis('off')

imshow(images[i])

plt.title(classes[labels[i].numpy()], fontdict={'fontsize':18})

**#使用ResNet18 作为模型来对人民币进行识别**

from torchvision import models

resnet = models.resnet18(pretrained=True).to(device)**#resnet18预训练模型进行迁移**学习

resnet.fc = torch.nn.Linear(resnet.fc.in\_features, 6).to(device)

import torch.optim as optim

import torch.nn as nn**#torcn.nn是专门为神经网络设计的模块化接口**

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(resnet.parameters(), lr=0.001)**#lr学习率**

epoches = 100

best\_val\_acc = 0.0

running\_loss\_history = []

running\_corrects\_history = []

val\_running\_loss\_history = []**#误差数组**

val\_running\_corrects\_history = []**#正确率数组**

for epoch in range(epoches):

running\_loss = 0.0

running\_corrects = 0.0

val\_running\_loss = 0.0 **#验证误差**

val\_running\_corrects = 0.0 **#修正**

**# 每训练一个epoch，验证一下网络模型**

for inputs, labels in train\_loader:

resnet.train()

inputs, labels = inputs.to(device), labels.to(device) **#获取训练数据**

outputs = resnet(inputs)  **#即前向传播求出预测的值**

loss = criterion(outputs, labels) **#j计算loss标量出来才能进行反向传播。**

optimizer.zero\_grad() **#梯度置零，也就是把loss关于weight的导数变成0.**

loss.backward() **#即反向传播求梯度**

optimizer.step()  **#即更新所有参数**

**#torch.max()这个函数返回的是两个值，第一个值是具体的value（我们用下划线\_表示），也就是输出的最大值，第二个值是value所在的index**

\_, pred = torch.max(outputs, 1)

running\_loss += loss.item() \* labels.size(0)**#加起来计算一个epoch的loss**

running\_corrects += torch.sum(pred == labels.data)

with torch.no\_grad():  **#with是python中上下文管理器， 取消验证阶段的梯度**

for val\_inputs, val\_labels in val\_loader:

val\_inputs, val\_labels = val\_inputs.to(device), val\_labels.to(device)

val\_outputs = resnet(val\_inputs)

val\_loss = criterion(val\_outputs, val\_labels) **# 将output和labels使用叉熵计算损失**

\_, val\_pred = torch.max(val\_outputs, 1)

val\_running\_loss += val\_loss.item() \* val\_labels.size(0)

val\_running\_corrects += torch.sum(val\_pred == val\_labels.data) **#记录这个epoch的模型的参数和梯度**

**#计算记录Loss和准确率的均值**

epoch\_loss = running\_loss / len(train\_data)

epoch\_acc = running\_corrects / len(train\_data)

running\_loss\_history.append(epoch\_loss)

running\_corrects\_history.append(epoch\_acc)

**#平验证集的Loss和准确率**

val\_epoch\_loss = val\_running\_loss / len(val\_data)

val\_epoch\_acc = val\_running\_corrects / len(val\_data)**# 计算一个epoch的准确率**

val\_running\_loss\_history.append(val\_epoch\_loss)

val\_running\_corrects\_history.append(val\_epoch\_acc)**# 1. 记录这个epoch的loss值和准确率**

if best\_val\_acc < val\_epoch\_acc:  **#求最大正确率**

best\_val\_acc = val\_epoch\_acc

torch.save(resnet.state\_dict(), 'resnet18\_best.mdl')

print('epoch: ',(epoch+1))

print('training loss: {:.4f}, acc {:.2f}% '.format(epoch\_loss, epoch\_acc \* 100))

print('validation loss: {:.4f}, validation acc {:.2f}% '.format(val\_epoch\_loss, val\_epoch\_acc.item()\*100))

print("Best validation acc {:.2f}%".format(best\_val\_acc \* 100))**#最高准确率**

**#训练误差和验证误差变化，用在Jupyter notebook中具体作用是当你调用matplotlib.pyplot的绘图函数plot()进行绘图的时候，**

**#或者生成一个figure画布的时候，可以直接在你的python console里面生成图像**

**%matplotlib inline**

plt.style.use('ggplot')**#设置背景样**式，

plt.plot(running\_loss\_history, label='training loss')

plt.plot(val\_running\_loss\_history, label='validation loss')

plt.legend()

**# 模型在训练集和验证集上的准确率变化**

plt.style.use('ggplot')

plt.plot(running\_corrects\_history, label='training accuracy')

plt.plot(val\_running\_corrects\_history, label='validation accuracy')

plt.legend()

**# 加载已存储的模型**

resnet.load\_state\_dict(torch.load('ResNet18\_best.mdl'))

**# 测试集测试**

test\_loss = 0.0

correct = 0

total = 0

resnet.eval()

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = resnet(images)

loss = criterion(outputs, labels)**#计算损失，叉熵计算损失**

test\_loss += loss.item() \* labels.size(0)

\_, pred = torch.max(outputs, 1)

correct += torch.eq(pred, labels).sum().float().item()

total += labels.size(0)

Loss = test\_loss / total

test\_acc = correct / total

print("Test Loss: {}\tTest Accuracy:{}".format(Loss, test\_acc))

mean\_vals = [0.485, 0.456, 0.406]

std\_vals = [0.229, 0.224, 0.225]

def imshow(img):

img = np.transpose(img, (1, 2, 0))

img = img \* std\_vals + mean\_vals

plt.imshow((img \* 255).astype('uint8'))

images, labels = iter(test\_loader).next()

**# images shape [b, 3, 32, 32]**

images, labels = images.to(device), labels.to(device) **#将输入传入GPU(CPU)**

output = resnet(images)

\_, preds = torch.max(output, 1)

preds = np.squeeze(preds.cpu().numpy())

images = images.cpu().numpy() **#将数据的处理设备从其他设备（如.cuda()拿到cpu上），不会改变变量类型，转换后仍然是Tensor变量。**

**# 将tensor 转换位 numpy [b, 3, 32, 32]**

fig = plt.figure(figsize=(25, 8)) # figsize(width, height)

for i in range(32):

ax = fig.add\_subplot(4, 8, i+1)

plt.axis('off')**#关闭坐标轴**

imshow(images[i]) **#对于每个images[i] 其shape为 [3, 32, 32]**

ax.set\_title("{} ({})".format(classes[preds[i]], classes[labels[i]]),

color=("green" if preds[i]==labels[i].item() else "red"),fontdict={'fontsize':13}