# 2.PyTorch数据处理

## 2.1.数据读取

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。

torch.utils.data.DataLoader

功能：构建可迭代的数据装载器  
 • dataset: Dataset类，决定数据从哪读取及如何读取  
 • batchsize : 批大小  
 • num\_works: 是否多进程读取数据  
 • shuffle: 每个epoch是否乱序  
 • drop\_last：当样本数不能被batchsize整除时，是否舍弃最后一批数据

DataLoader( dataset,  
 batch\_size=1,  
 shuffle=False,  
 sampler=None,  
 batch\_sampler=None,  
 num\_workers=0,  
 collate\_fn=None,  
 pin\_memory=False,  
 drop\_last=False,  
 timeout=0,  
 worker\_init\_fn=None,  
 multiprocessing\_context=None)

概念：

Epoch: 所有训练样本都已输入到模型中，称为一个Epoch  
 Iteration：一批样本输入到模型中，称之为一个Iteration  
 Batchsize：批大小，决定一个Epoch有多少个Iteration

例如：

样本总数： 80， Batchsize： 8  
 1 Epoch = 10 Iteration  
 样本总数： 87， Batchsize： 8  
 1 Epoch = 10 Iteration drop\_last = True  
 1 Epoch = 11 Iteration drop\_last = False

torch.utils.data.Dataset

功能： Dataset抽象类，所有自定义的Dataset需要继承它，并且复写\_\_getitem\_\_()  
 getitem :接收一个索引，返回一个样本

class Dataset(object):  
 def \_\_getitem\_\_(self, index):  
 raise NotImplementedError  
 def \_\_add\_\_(self, other):  
 return ConcatDataset([self, other])

def \_\_len\_\_(self):  
 return len([self])

案例：读取人民币1元和100元

代码实现：

首先对图片集进行划分，分为训练集、验证集、测试集（split\_dataset.py）

*# -\*- coding: utf-8 -\*-***import** os  
**import** random  
**import** shutil  
  
*# 创建路径，如果路径不存在***def** makedir(new\_dir):  
 **if not** os.path.exists(new\_dir):  
 os.makedirs(new\_dir)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
  
 random.seed(1)  
  
 *# 数据目录* dataset\_dir = os.path.join(**".."**, **"data"**, **"RMB\_data"**)  
 *# 切分后的目录* split\_dir = os.path.join(**".."**, **"data"**, **"rmb\_split"**)  
 *# 训练集子目录* train\_dir = os.path.join(split\_dir, **"train"**)  
 *# 验证集子目录* valid\_dir = os.path.join(split\_dir, **"valid"**)  
 *# 测试集子目录* test\_dir = os.path.join(split\_dir, **"test"**)  
  
 *# 设置切分比例* train\_pct = 0.8  
 valid\_pct = 0.1  
 test\_pct = 0.1  
  
 **"""  
 os.walk(dataset\_dir)：得到一个三元tupple(dirpath, dirnames, filenames)  
 dirpath：为起始路径，是一个string，代表目录的路径  
 dirnames：为起始路径下的文件夹，是一个list，包含了dirpath下所有子目录的名字  
 filenames：是起始路径下的文件，是一个list，包含了非目录文件的名字  
 """  
 for** root, dirs, files **in** os.walk(dataset\_dir):  
 *# 遍历子目录* **for** sub\_dir **in** dirs:  
  
 *# 每个子目录下所有的文件名称* imgs = os.listdir(os.path.join(root, sub\_dir))  
 *# 过滤出所有以.jpg结尾的文件名* imgs = list(filter(**lambda** x: x.endswith(**'.jpg'**), imgs))  
 *# 打乱顺序* random.shuffle(imgs)  
 *# 获取元素总数量* img\_count = len(imgs)  
  
 *# 训练集数量* train\_point = int(img\_count \* train\_pct)  
 *# 验证集数量（训练集+验证集）* valid\_point = int(img\_count \* (train\_pct + valid\_pct))  
  
 *# 遍历所有图片* **for** i **in** range(img\_count):  
 **if** i < train\_point:  
 *# 如果i小于训练集数量，则生成训练集目录* out\_dir = os.path.join(train\_dir, sub\_dir)  
 **elif** i < valid\_point:  
 *# 如果i小于验证集数量，则生成验证集目录* out\_dir = os.path.join(valid\_dir, sub\_dir)  
 **else**:  
 *# 生成测试集目录* out\_dir = os.path.join(test\_dir, sub\_dir)  
  
 *# 创建输出目录* makedir(out\_dir)  
  
 *# 目标路径为输出目录拼接文件名* target\_path = os.path.join(out\_dir, imgs[i])  
 *# 获取原始文件目录* src\_path = os.path.join(dataset\_dir, sub\_dir, imgs[i])  
  
 *# shutil.copy()方法用来拷贝文件* shutil.copy(src\_path, target\_path)  
  
 print(**'Class:{}, train:{}, valid:{}, test:{}'**.format(sub\_dir, train\_point, valid\_point-train\_point

, img\_count-valid\_point))

然后，自定义Dataset：my\_dataset.py

*# -\*- coding: utf-8 -\*-***import** os  
**import** random  
**from** PIL **import** Image  
**from** torch.utils.data **import** Dataset  
  
random.seed(1)  
rmb\_label = {**"1"**: 0, **"100"**: 1}  
  
*# 自定义RMBDataset继承Dataset***class** RMBDataset(Dataset):  
 **def** \_\_init\_\_(self, data\_dir, transform=**None**):  
 *"""  
 rmb面额分类任务的Dataset* **:param** *data\_dir: str, 数据集所在路径* **:param** *transform: torch.transform，数据预处理  
 """* self.label\_name = {**"1"**: 0, **"100"**: 1}  
 self.data\_info = self.get\_img\_info(data\_dir) *# data\_info存储所有图片路径和标签* self.transform = transform *# 用来对图片进行处理*  
  
 **def** \_\_getitem\_\_(self, index):  
 path\_img, label = self.data\_info[index]  
 *# 读取图片并转换为RGB三通道，不转换默认是四通道，即多一个透明度通道，通常用不到* img = Image.open(path\_img).convert(**'RGB'**) *# 0~255* **if** self.transform **is not None**:  
 img = self.transform(img) *# 在这里做transform，转为tensor等等* **return** img, label  
  
 **def** \_\_len\_\_(self):  
 **return** len(self.data\_info)  
  
 @staticmethod  
 **def** get\_img\_info(data\_dir):  
 data\_info = list()  
 **for** root, dirs, \_ **in** os.walk(data\_dir):  
 *# 遍历类别* **for** sub\_dir **in** dirs:  
 img\_names = os.listdir(os.path.join(root, sub\_dir))  
 img\_names = list(filter(**lambda** x: x.endswith(**'.jpg'**), img\_names))  
  
 *# 遍历图片* **for** i **in** range(len(img\_names)):  
 img\_name = img\_names[i]  
 *# 每张图片的全路径* path\_img = os.path.join(root, sub\_dir, img\_name)  
 *# 每张图片的标签* label = rmb\_label[sub\_dir]  
 data\_info.append((path\_img, int(label)))  
  
 **return** data\_info

最后，利用DataLoader加载数据。

**import** os  
**from** torch.utils.data **import** DataLoader  
**import** torchvision.transforms **as** transforms  
**from** tools.my\_dataset **import** RMBDataset  
  
*# 设置每批数据的大小*BATCH\_SIZE = 16  
  
split\_dir = os.path.join(**".."**, **"data"**, **"rmb\_split"**)  
train\_dir = os.path.join(split\_dir, **"train"**)  
valid\_dir = os.path.join(split\_dir, **"valid"**)  
  
*# 图像各通道的均值，ImageNet的数据在加载的时候就已经转换成了[0, 1]，[0.485, 0.456, 0.406]这一组平均值是从imagenet训练集中抽样算出来的*norm\_mean = [0.485, 0.456, 0.406]  
*# 图像各通道的方差*norm\_std = [0.229, 0.224, 0.225]  
  
*# Compose方法将transform的方法有序的组合*train\_transform = transforms.Compose([  
 *# 缩放图像大小，处理时需要统一的图像大小* transforms.Resize((32, 32)),  
 *# 对图像进行随机的裁剪* transforms.RandomCrop(32, padding=4),  
 *# 将图像转换为Tensor* transforms.ToTensor(),  
 *# 对图像进行标准化* transforms.Normalize(norm\_mean, norm\_std),  
])  
  
valid\_transform = transforms.Compose([  
 transforms.Resize((32, 32)),  
 transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std),  
])  
  
*# 构建MyDataset实例*train\_data = RMBDataset(data\_dir=train\_dir, transform=train\_transform)  
valid\_data = RMBDataset(data\_dir=valid\_dir, transform=valid\_transform)  
  
*# 构建DataLoder*train\_loader = DataLoader(dataset=train\_data, batch\_size=BATCH\_SIZE, shuffle=**True**)  
valid\_loader = DataLoader(dataset=valid\_data, batch\_size=BATCH\_SIZE)  
  
**for** epoch **in** range(2):  
 **for** i, data **in** enumerate(train\_loader):  
 *# 将数据从 train\_loader 中读出来,一次读取的样本数是16个* img, labels = data  
  
 *# 接下来就是跑模型的环节了，我们这里使用print来代替，打印每批数据的样本数* print(**"epoch："**, epoch, **"的第"** , i, **"batch"**, **"size"**, img.size())

|  |
| --- |
| epoch： 0 的第 0 batch len torch.Size([16, 3, 32, 32])  epoch： 0 的第 1 batch len torch.Size([16, 3, 32, 32])  ...  epoch： 1 的第 9 batch len torch.Size([16, 3, 32, 32]) |

## 2.2.图像预处理-transforms

### 2.2.1.transforms介绍

torchvision.transforms : 常用的图像预处理方法  
torchvision.datasets : 常用数据集的dataset实现， MNIST， CIFAR-10， ImageNet等  
torchvision.model : 常用的模型预训练， AlexNet， VGG， ResNet， GoogLeNet等

torchvision.transforms : 常用的图像预处理方法  
 • 数据中心化  
 • 数据标准化  
 • 缩放  
 • 裁剪  
 • 旋转  
 • 翻转  
 • 填充  
 • 噪声添加  
 • 灰度变换  
 • 线性变换  
 • 仿射变换  
 • 亮度、饱和度及对比度变换





transforms.Normalize

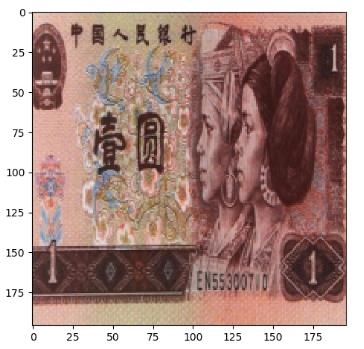
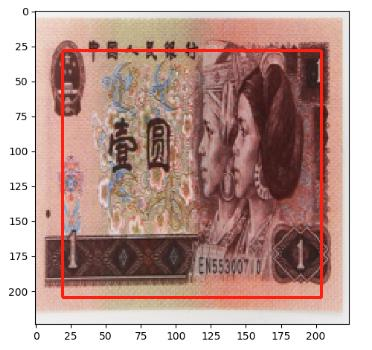
功能：逐channel的对图像进行标准化 output = (input - mean) / std  
 • mean：各通道的均值  
 • std：各通道的标准差  
 • inplace：是否原地操作

transforms.Normalize(mean,  
 std,  
 inplace=False)

### 2.2.2.裁剪

1. transforms.CenterCrop

功能：从图像中心裁剪图片  
 • size：所需裁剪图片尺寸



代码实现：

首先因为数据经过transfrom，转换为张量的形式，可能是一些浮点的数据，没有办法将这些数据进行可视化，因此需要一个transform\_invert()函数，对transform进行逆操作，将张量的数据变换成img，这样就可以进行可视化。

*# -\*- coding: utf-8 -\*-***import** os  
**import** numpy **as** np  
**import** torch  
**import** random  
**from** torch.utils.data **import** DataLoader  
**import** torchvision.transforms **as** transforms  
**from** tools.my\_dataset **import** RMBDataset  
**from** PIL **import** Image  
**from** matplotlib **import** pyplot **as** plt  
  
**def** set\_seed(seed=1):  
 random.seed(seed)  
 np.random.seed(seed)  
 torch.manual\_seed(seed)  
 torch.cuda.manual\_seed(seed)  
  
set\_seed(1) *# 设置随机种子  
  
# 参数设置*MAX\_EPOCH = 10  
BATCH\_SIZE = 1  
  
**def** transform\_invert(img\_, transform\_train):  
 *"""  
 将data 进行反transfrom操作* **:param** *img\_: tensor* **:param** *transform\_train: torchvision.transforms* **:return***: PIL image  
 """* **if 'Normalize' in** str(transform\_train):  
 *# 获取Normalize中的参数，即Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])* norm\_transform = list(filter(**lambda** x: isinstance(x, transforms.Normalize), transform\_train.transforms))  
 *# 将均值转换为tensor，其中数据类型为图片的数据类型，即float32，最后为设备类型，即cpu* mean = torch.tensor(norm\_transform[0].mean, dtype=img\_.dtype, device=img\_.device)  
 std = torch.tensor(norm\_transform[0].std, dtype=img\_.dtype, device=img\_.device)  
 *# 对图片进行逆运算，即数据×std+mean* img\_.mul\_(std[:, **None**, **None**]).add\_(mean[:, **None**, **None**])  
  
 *# 将通道的C∗H∗WC\*H\*WC∗H∗W格式转换为H∗W∗CH\*W\*CH∗W∗C，即将channel放到最后面* img\_ = img\_.transpose(0, 2).transpose(0, 1) *# C\*H\*W --> H\*W\*C  
 # 将tensor转换为numpy，并将0-1转换为0-255* img\_ = np.array(img\_) \* 255  
  
 **if** img\_.shape[2] == 3:  
 *# 将array类型的图像转换为图片类型，并且指定色彩模式为RGB* img\_ = Image.fromarray(img\_.astype(**'uint8'**)).convert(**'RGB'**)  
 **elif** img\_.shape[2] == 1:  
 img\_ = Image.fromarray(img\_.astype(**'uint8'**).squeeze())  
 **else**:  
 **raise** Exception(**"Invalid img shape, expected 1 or 3 in axis 2, but got {}!"**.format(img\_.shape[2]) )  
  
 **return** img\_  
  
*# 获取数据*split\_dir = os.path.join(**".."**, **"data"**, **"rmb\_split"**)  
train\_dir = os.path.join(split\_dir, **"train"**)  
valid\_dir = os.path.join(split\_dir, **"valid"**)  
  
norm\_mean = [0.485, 0.456, 0.406]  
norm\_std = [0.229, 0.224, 0.225]  
  
*# 图像转换*train\_transform = transforms.Compose([  
 *# 统一图片尺寸* transforms.Resize((224, 224)),  
  
 *# 1 CenterCrop  
 # transforms.CenterCrop(128), # 512  
  
 # 2 RandomCrop  
 # 图像大小变为224+16+16=224+32，裁剪时从224+32大小的范围内随机裁剪  
 # transforms.RandomCrop(224, padding=16),  
 # 左右填充16像素，上下为64像素  
 # transforms.RandomCrop(224, padding=(16, 64)),  
 # 设置填充颜色  
 # transforms.RandomCrop(224, padding=16, fill=(255, 0, 0)),  
 # 当图像小于设定size，必须设置pad\_if\_needed=True，否则报错  
 # transforms.RandomCrop(512, pad\_if\_needed=False), # pad\_if\_needed=True  
 # 填充像素由边缘决定  
 # transforms.RandomCrop(224, padding=64, padding\_mode='edge'),  
 # transforms.RandomCrop(224, padding=64, padding\_mode='reflect'),  
 # transforms.RandomCrop(1024, padding=1024, padding\_mode='symmetric'),  
  
 # 3 RandomResizedCrop  
 # transforms.RandomResizedCrop(size=224, scale=(0.1, 1)),  
  
 # 4 FiveCrop  
 # 直接使用返回的是五张图片组成的tuple，不能直接使用  
 # transforms.FiveCrop(112),  
 # 对返回的tuple，获取每个元素，转换为tensor组成的list，然后进行stack拼接，默认在第0维度拼接  
 # 这里已经转换为tensor，所以下面的ToTensor和Normalize就无需再次执行  
 # 由于生成的图片维度与之前维度不同，所以在可视化时需要按照对应维度取值并可视化  
 # transforms.Lambda(lambda crops: torch.stack([(transforms.ToTensor()(crop)) for crop in crops])),  
  
 # 5 TenCrop  
 # transforms.TenCrop(112, vertical\_flip=True), # vertical\_flip=False  
 # transforms.Lambda(lambda crops: torch.stack([(transforms.ToTensor()(crop)) for crop in crops])),  
  
 # 1 Horizontal Flip  
 # transforms.RandomHorizontalFlip(p=1),  
  
 # 2 Vertical Flip  
 # transforms.RandomVerticalFlip(p=0.5),  
  
 # 3 RandomRotation  
 # transforms.RandomRotation(90),  
 # 这里批大小必须设置为1，否则由于每批中不同图片旋转角度不同，而扩充时计算不同导致报错  
 # transforms.RandomRotation((90), expand=True),  
 # transforms.RandomRotation(30, center=(0, 0)),  
 # expand是根据中心旋转计算扩展大小的，因此如果变为左上角旋转，其计算出的结果不一定能拟合导致无法显示完整  
 # transforms.RandomRotation(30, center=(0, 0), expand=True),*

transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std),  
])  
  
valid\_transform = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std)  
])  
  
*# 构建MyDataset实例*train\_data = RMBDataset(data\_dir=train\_dir, transform=train\_transform)  
valid\_data = RMBDataset(data\_dir=valid\_dir, transform=valid\_transform)  
  
*# 构建DataLoder*train\_loader = DataLoader(dataset=train\_data, batch\_size=BATCH\_SIZE, shuffle=**True**)  
valid\_loader = DataLoader(dataset=valid\_data, batch\_size=BATCH\_SIZE)  
  
*# 数据处理***for** epoch **in** range(MAX\_EPOCH):  
 **for** i, data **in** enumerate(train\_loader):  
 *# 获取每批的数据和标签，其中批大小为1* inputs, labels = data *# B C H W  
  
 # 获取每批的第一个数据* img\_tensor = inputs[0, ...] *# C H W  
 # 对tensor类型的图片进行逆变换，变为plt认识的图片* img = transform\_invert(img\_tensor, train\_transform)  
 plt.imshow(img)  
 plt.show()  
 plt.pause(0.5)  
 plt.close()  
  
 *# 对FiveCrop和TenCrop后的结果可视化  
 # bs, ncrops, c, h, w = inputs.shape  
 # for n in range(ncrops):  
 # img\_tensor = inputs[0, n, ...] # C H W  
 # img = transform\_invert(img\_tensor, train\_transform)  
 # plt.imshow(img)  
 # plt.show()  
 # plt.pause(1)*

2. transforms.RandomCrop  
 功能：从图片中随机裁剪出尺寸为size的图片  
 • size：所需裁剪图片尺寸  
 • padding：设置填充大小  
 当为a时，上下左右均填充a个像素  
 当为(a, b)时，上下填充b个像素，左右填充a个像素  
 当为(a, b, c, d)时，左，上，右，下分别填充a, b, c, d  
 • pad\_if\_need：若图像小于设定size，则填充

• padding\_mode：填充模式，有4种模式  
 constant：像素值由fill设定  
 edge：像素值由图像边缘像素决定  
 reflect：镜像填充，最后一个像素不镜像，eg：[1,2,3,4] → [3,2,1,2,3,4,3 ,2]  
 symmetric：镜像填充，最后一个像素镜像， eg： [1,2,3,4] → [2,1,1,2,3,4,4 ,3]  
 • fill： constant时，设置填充的像素值

transforms.RandomCrop(size,  
 padding=None,  
 pad\_if\_needed=False,  
 fill=0,  
 padding\_mode='constant')

*# 2 RandomCrop  
# 图像大小变为224+16+16=224+32，裁剪时从224+32大小的范围内随机裁剪  
# transforms.RandomCrop(224, padding=16),  
# 左右填充16像素，上下为64像素  
# transforms.RandomCrop(224, padding=(16, 64)),  
# 设置填充颜色  
# transforms.RandomCrop(224, padding=16, fill=(255, 0, 0)),  
# 当图像小于设定size，必须设置pad\_if\_needed=True，否则报错  
# transforms.RandomCrop(512, pad\_if\_needed=False), # pad\_if\_needed=True  
# 填充像素由边缘决定  
# transforms.RandomCrop(224, padding=64, padding\_mode='edge'),  
# transforms.RandomCrop(224, padding=64, padding\_mode='reflect'),  
# transforms.RandomCrop(1024, padding=1024, padding\_mode='symmetric'),*

3. RandomResizedCrop

功能：随机大小、长宽比裁剪图片  
 • size：所需裁剪图片尺寸  
 • scale：随机裁剪面积比例, 默认(0.08, 1)  
 • ratio：随机长宽比，默认(3 /4, 4/3)  
 • interpolation：插值方法，默认为双线性插值(PIL.Image.BILINEAR)

RandomResizedCrop(size,  
 scale=(0.08, 1.0),  
 ratio=(3/4, 4/3),  
 interpolation)

*# 3 RandomResizedCrop  
# transforms.RandomResizedCrop(size=224, scale=(0.1, 1)),*

4. FiveCrop  
5. TenCrop

功能：在图像的四个角以及中心裁剪出尺寸为size的5张图片， TenCrop对这5张 图片进行水平或者垂直镜像获得10张图片  
 • size：所需裁剪图片尺寸  
 • vertical\_flip：是否垂直翻转

transforms.FiveCrop(size)  
 transforms.TenCrop(size,  
 vertical\_flip=False)

*# 4 FiveCrop  
# 直接使用返回的是五张图片组成的tuple，不能直接使用  
# transforms.FiveCrop(112),  
# 对返回的tuple，获取每个元素，转换为tensor组成的list，然后进行stack拼接，默认在第0维度拼接  
# 这里已经转换为tensor，所以下面的ToTensor和Normalize就无需再次执行  
# 由于生成的图片维度与之前维度不同，所以在可视化时需要按照对应维度取值并可视化  
# transforms.Lambda(lambda crops: torch.stack([(transforms.ToTensor()(crop)) for crop in crops])),  
  
# 5 TenCrop  
# transforms.TenCrop(112, vertical\_flip=True), # vertical\_flip=False  
# transforms.Lambda(lambda crops: torch.stack([(transforms.ToTensor()(crop)) for crop in crops])),*

### 2.2.3.翻转、旋转

1.RandomHorizontalFlip  
2.RandomVerticalFlip

功能：依概率水平（左右）或垂直（上下）翻转图片  
 • p：翻转概率

RandomHorizontalFlip(p=0.5)  
 RandomVerticalFlip(p=0.5)

*# 1 Horizontal Flip  
# transforms.RandomHorizontalFlip(p=1),  
  
# 2 Vertical Flip  
# transforms.RandomVerticalFlip(p=0.5),*

3.RandomRotation

功能：随机旋转图片  
 • degrees：旋转角度  
 当为a时，在（ -a， a）之间选择旋转角度  
 当为(a, b)时，在(a, b)之间选择旋转角度  
 • resample：重采样方法  
 • expand：是否扩大图片，以保持原图信息

• center：旋转点设置，默认中心旋转

RandomRotation(degrees,  
 resample=False,  
 expand=False,  
 center=None)

*# 3 RandomRotation  
# transforms.RandomRotation(90),  
# 这里批大小必须设置为1，否则由于每批中不同图片旋转角度不同，而扩充时计算不同导致报错  
# transforms.RandomRotation((90), expand=True),  
# transforms.RandomRotation(30, center=(0, 0)),  
# expand是根据中心旋转计算扩展大小的，因此如果变为左上角旋转，其计算出的结果不一定能拟合导致无法显示完整  
# transforms.RandomRotation(30, center=(0, 0), expand=True),*

### 2.2.4.图像变换

1. Pad

功能：对图片边缘进行填充  
 • padding：设置填充大小  
 当为a时，上下左右均填充a个像素  
 当为(a, b)时，上下填充b个像素，左右填充a个像素  
 当为(a, b, c, d)时，左，上，右，下分别填充a, b, c, d  
 • padding\_mode：填充模式，有4种模式， constant、 edge、 reflect和symmetric  
 • fill： constant时，设置填充的像素值， (R, G, B) or (Gray)

transforms.Pad(padding,  
 fill=0,  
 padding\_mode='constant')

*# 1 Pad  
# transforms.Pad(padding=32, fill=(255, 0, 0), padding\_mode='constant'),  
# transforms.Pad(padding=(8, 64), fill=(255, 0, 0), padding\_mode='constant'),  
# transforms.Pad(padding=(8, 16, 32, 64), fill=(255, 0, 0), padding\_mode='constant'),  
# transforms.Pad(padding=(8, 16, 32, 64), fill=(255, 0, 0), padding\_mode='symmetric'),*

2. ColorJitter

功能：调整亮度、对比度、饱和度和色相  
 • brightness：亮度调整因子  
 当为a时，从[max(0, 1-a), 1+a]中随机选择  
 当为(a, b)时，从[a, b]中  
 • contrast：对比度参数，同brightness  
 • saturation：饱和度参数，同brightness  
 • hue：色相参数，

当为a时，从[-a, a]中选择参数，注： 0<= a <= 0.5  
 当为(a, b)时，从[a, b]中选择参数，注： -0.5 <= a <= b <= 0.5

transforms.ColorJitter(brightness=0,  
 contrast=0,  
 saturation=0,  
 hue=0)

*# 2 ColorJitter  
# transforms.ColorJitter(brightness=0.5),  
# transforms.ColorJitter(contrast=0.5),  
# transforms.ColorJitter(saturation=0.5),  
# transforms.ColorJitter(hue=0.3),*

3. Grayscale

功能：将图片转换为灰度图，是RandomGrayscale 的特例  
4. RandomGrayscale

功能：依概率将图片转换为灰度图  
 • num\_ouput\_channels：输出通道数，只能设1或3  
 • p：概率值，图像被转换为灰度图的概率

RandomGrayscale(num\_output\_channels,  
 p=0.1)  
 Grayscale(num\_output\_channels)

*# 3 Grayscale*

*# 这里设置通道为3，是为了和Normalize的维度匹配  
# transforms.Grayscale(num\_output\_channels=3),*

5. RandomAffine

功能：对图像进行仿射变换，仿射变换是二维的线性变换，由五种基本原子变换构成，分别是旋转、 平移、 缩放、 错切和翻转  
 • degrees：旋转角度设置  
 • translate：平移区间设置，如(a, b), a设置宽（width）， b设置高(height) 图像在宽维度平移的区间为 -img\_width \* a < dx < img\_width \* a  
 • scale：缩放比例（以面积为单位）  
 • fill\_color：填充颜色设置

• shear：错切角度设置，有水平错切和垂直错切  
 若为a，则仅在x轴错切，错切角度在(-a, a)之间  
 若为(a， b)，则a设置x轴角度， b设置y的角度  
 若为(a, b, c, d)，则a, b设置x轴角度， c, d设置y轴角度  
 • resample：插值方法

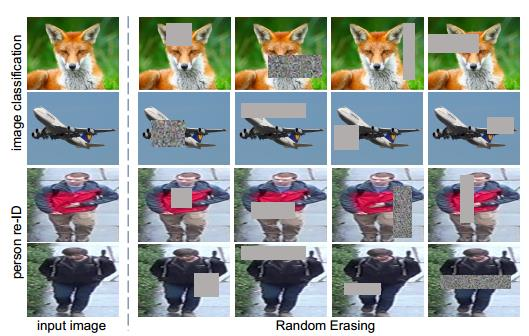
RandomAffine(degrees,  
 translate=None,  
 scale=None,  
 shear=None,  
 resample=False,  
 fillcolor=0)

*# 5 RandomAffine   
# transforms.RandomAffine(degrees=30),  
# transforms.RandomAffine(degrees=0, translate=(0.2, 0.2), fillcolor=(255, 0, 0)),  
# transforms.RandomAffine(degrees=0, scale=(0.5, 0.7)),  
# transforms.RandomAffine(degrees=0, shear=(0, 0, 0, 45)),  
# transforms.RandomAffine(degrees=0, shear=90, fillcolor=(255, 0, 0)),*

6. RandomErasing

功能：对图像进行随机遮挡  
 • p：概率值，执行该操作的概率  
 • scale：遮挡区域的面积  
 • ratio：遮挡区域长宽比  
 • value：设置遮挡区域的像素值， (R, G, B) or (Gray)

RandomErasing(p=0.5,  
 scale=(0.02, 0.33),  
 ratio=(0.3, 3.3),  
 value=0,  
 inplace=False)



*# 6 Erasing  
# 遮挡需要对张量进行操作，所以这里需要先转换为张量，所以下面的ToTensor就无需再次执行  
# transforms.ToTensor(),  
# transforms.RandomErasing(p=1, scale=(0.02, 0.33), ratio=(0.3, 3.3), value=(255/255, 0, 0)),  
# 如果value设置为任意字符串，则填充时也会随机填充  
# transforms.RandomErasing(p=1, scale=(0.02, 0.33), ratio=(0.3, 3.3), value='abc'),*

7. transforms.Lambda

功能：用户自定义lambda方法  
 • lambd： lambda匿名函数

transforms.Lambda(lambd)

例如，之前在使用TenCrop时使用过Lambda函数：

transforms.TenCrop(112, vertical\_flip=**True**), transforms.Lambda(**lambda** crops: torch.stack([(transforms.ToTensor()(crop)) **for** crop **in** crops])),

### 2.2.5.组合操作

1. transforms.RandomChoice

功能：从一系列transforms方法中随机挑选一个

transforms.RandomChoice([transforms1, transforms2, transforms3])

*# 1 RandomChoice*transforms.RandomChoice([transforms.RandomVerticalFlip(p=1), transforms.RandomHorizontalFlip(p=1)]),

2. transforms.RandomApply

功能：依据概率执行一组transforms操作

transforms.RandomApply([transforms1, transforms2, transforms3], p=0.5)

*# 2 RandomApply*transforms.RandomApply([transforms.RandomAffine(degrees=0, shear=45, fillcolor=(255, 0, 0)),  
 transforms.Grayscale(num\_output\_channels=3)], p=0.5),

3. transforms.RandomOrder

功能：对一组transforms操作打乱顺序

transforms.RandomOrder([transforms1, transforms2, transforms3])

*# 3 RandomOrder*transforms.RandomOrder([transforms.RandomRotation(15),  
 transforms.Pad(padding=32),  
 transforms.RandomAffine(degrees=0, translate=(0.01, 0.1), scale=(0.9, 1.1))]),

### 2.2.6.自定义transforms

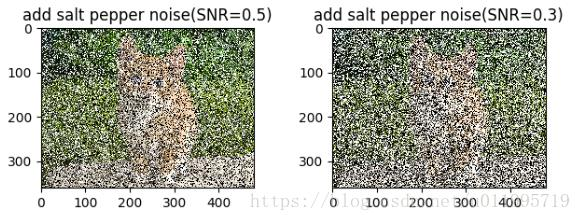
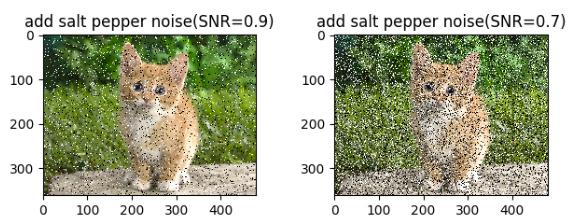
自定义transforms：

注意输入输出

通过类实现多参数传入：

class YourTransforms(object):  
 def \_\_init\_\_(self, ...):  
 ...  
 def \_\_call\_\_(self, img):  
 ...  
 return img

案例：椒盐噪声  
 椒盐噪声又称为脉冲噪声，是一种随机出现的白点或者黑点, 白点称为盐噪声，黑色为椒噪声  
 信噪比（Signal-Noise Rate, SNR）是衡量噪声的比例，图像中为图像像素的占比



代码实现：

*# -\*- coding: utf-8 -\*-***import** os  
**import** numpy **as** np  
**import** torch  
**import** random  
**import** torchvision.transforms **as** transforms  
**from** PIL **import** Image  
**from** matplotlib **import** pyplot **as** plt  
**from** torch.utils.data **import** DataLoader  
**from** tools.my\_dataset **import** RMBDataset  
**from** tools.common\_tools **import** transform\_invert  
  
**def** set\_seed(seed=1):  
 random.seed(seed)  
 np.random.seed(seed)  
 torch.manual\_seed(seed)  
 torch.cuda.manual\_seed(seed)  
  
set\_seed(1) *# 设置随机种子  
  
# 参数设置*MAX\_EPOCH = 10  
BATCH\_SIZE = 1  
  
**class** AddPepperNoise(object):  
 *"""增加椒盐噪声  
 Args:  
 snr （float）: 信噪比  
 p (float): 概率值，依概率执行该操作  
 """* **def** \_\_init\_\_(self, snr, p=0.9):  
 **assert** isinstance(snr, float) **or** (isinstance(p, float))  
 self.snr = snr  
 self.p = p  
  
 **def** \_\_call\_\_(self, img):  
 *"""  
 Args:  
 img (PIL Image): PIL Image  
 Returns:  
 PIL Image: PIL image.  
 """* **if** random.uniform(0, 1) < self.p: *# 根据概率判断是否执行椒盐噪声* img\_ = np.array(img).copy()  
 h, w, c = img\_.shape  
 signal\_pct = self.snr  
 noise\_pct = (1 - self.snr)  
 *# choice((0, 1, 2)，这里用0表示原始图像，1表示盐噪声，2表示椒噪声，选取概率为信噪比* mask = np.random.choice((0, 1, 2), size=(h, w, 1), p=[signal\_pct, noise\_pct/2., noise\_pct/2.])  
 *# 沿着channel的维度进行复制* mask = np.repeat(mask, c, axis=2)  
 img\_[mask == 1] = 255 *# 盐噪声* img\_[mask == 2] = 0 *# 椒噪声* **return** Image.fromarray(img\_.astype(**'uint8'**)).convert(**'RGB'**)  
 **else**:  
 **return** img  
  
*#数据准备*split\_dir = os.path.join(**".."**, **".."**, **"data"**, **"rmb\_split"**)  
train\_dir = os.path.join(split\_dir, **"train"**)  
valid\_dir = os.path.join(split\_dir, **"valid"**)  
  
norm\_mean = [0.485, 0.456, 0.406]  
norm\_std = [0.229, 0.224, 0.225]  
  
train\_transform = transforms.Compose([  
 transforms.Resize((224, 224)),

*# 这里调用自定义的transforms*

AddPepperNoise(0.9, p=0.5),  
 transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std),  
])  
  
valid\_transform = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize(norm\_mean, norm\_std)  
])  
  
*# 构建MyDataset实例*train\_data = RMBDataset(data\_dir=train\_dir, transform=train\_transform)  
valid\_data = RMBDataset(data\_dir=valid\_dir, transform=valid\_transform)  
  
*# 构建DataLoder*train\_loader = DataLoader(dataset=train\_data, batch\_size=BATCH\_SIZE, shuffle=**True**)  
valid\_loader = DataLoader(dataset=valid\_data, batch\_size=BATCH\_SIZE)  
  
*# 数据处理***for** epoch **in** range(MAX\_EPOCH):  
 **for** i, data **in** enumerate(train\_loader):  
 inputs, labels = data *# B C H W* img\_tensor = inputs[0, ...] *# C H W* img = transform\_invert(img\_tensor, train\_transform)  
 plt.imshow(img)  
 plt.show()  
 plt.pause(0.5)  
 plt.close()

至此，我们学习了transforms的常见方法，归类如下

一、裁剪  
 • 1. transforms.CenterCrop  
 • 2. transforms.RandomCrop  
 • 3. transforms.RandomResizedCrop  
 • 4. transforms.FiveCrop  
 • 5. transforms.TenCrop  
二、翻转和旋转  
 • 1. transforms.RandomHorizontalFlip  
 • 2. transforms.RandomVerticalFlip  
 • 3. transforms.RandomRotation  
三、图像变换  
 • 1. transforms.Pad  
 • 2. transforms.ColorJitter  
 • 3. transforms.Grayscale  
 • 4. transforms.RandomGrayscale  
 • 5. transforms.RandomAffine  
 • 6. transforms.LinearTransformation  
 • 7. transforms.RandomErasing  
 • 8. transforms.Lambda  
 • 9. transforms.Resize  
 • 10. transforms.Totensor  
 • 11. transforms.Normalize  
四、 transforms的操作  
 • 1. transforms.RandomChoice  
 • 2. transforms.RandomApply  
 • 3. transforms.RandomOrder