学习目标：学习Python和PyTorch中图像读取；学会数据扩增方法；使用PyTorch读取赛题数据

1 Python图像读取

由于赛题数据是图像数据，赛题的任务是识别图像中的字符。因此我们首先需要完成对数据的读取操作，在Python中有很多库可以完成数据读取的操作，比较常见的有Pillow和OpenCV。

1.1 Pillow函数库

Pillow是Python图像处理函式库(PIL）的一个分支。PIL是一个函式库，提供了几个操作图像的标准程序。它是一个功能强大的函式库，但自2011年以来就没有太多的更新，并且不支持Python3。

Pillow在PIL的基础上，为Python3增加了更多功能和支持。它支持一系列图像文件格式PNG,JPEG,PPM,GIF,TIFF和BMP。我们将看到如何在图像上执行各种操作，例如裁剪，调整大小，添加文本到图像，旋转，灰阶转换。

Pillow的官方文档：https://pillow.readthedocs.io/en/stable/

Pillow的基本操作

```
import matplotlib.pyplot as plt
# 导入Pillow库
from PIL import Image, ImageFilter

# 读取并展示图片
im = Image.open("cat.png")
print(im.size)
plt.imshow(im)
plt.show()

# 调整图像的大小
im2 = im.resize((400, 400))

# 图像缩图（thumbnail）:会保持长宽比例
im.thumbnail((200, 200))
print(im.size)

# 应用模糊滤镜
im2 = im.filter(ImageFilter.BLUR)
im2.save("blur.png")
```

1.2 OpenCV

OpenCV是一个跨平台的计算机视觉库，最早由Intel开源得来。OpenCV发展的非常早，拥有众多的计算机视觉、数字图像处理和机器视觉等功能。OpenCV在功能上比Pillow更加强大很多，学习成本也高很多。

OpenCV包含了众多的图像处理的功能，OpenCV包含了你能想得到的只要与图像相关的操作。此外OpenCV还内置了很多的图像特征处理算法，如关键点检测、边缘检测和直线检测等。

OpenCV官网：https://opencv.org/

OpenCV Github：https://github.com/opencv/opencv

OpenCV 扩展算法库：https://github.com/opencv/opencv_contrib

OpenCV的基本操作

```
import matplotlib.pyplot as plt
# 导入OpenCv库
import cv2

# 读取并展示图片
img = cv2.imread('cat.png')
# OpenCv默认颜色通道顺序是BRG，转换一下
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()

# 转换为灰度图
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Canny边缘检测
edges = cv2.Canny(img, 30, 70)
cv2.imwrite('canny.jpg', edges)
```

2 数据扩增方法

在赛题中我们需要对的图像进行字符识别，因此既需要我们完成对数据的读取操作，同时也需要完成数据扩增（Data Augmentation）操作。

2.1 数据扩增介绍

在深度学习中数据扩增方法非常重要，数据扩增可以增加训练集的样本，同时也可以有效缓解模型过拟合的情况，也可以给模型带来的更强的泛化能力。

数据扩增为什么有用？

在深度学习模型的训练过程中，数据扩增是必不可少的环节。现有深度学习的参数非常多，一般的模型可训练的参数量基本上都是万到百万级别，而训练集样本的数量很难有这么多。

其次数据扩增可以扩展样本空间，假设现在的分类模型需要对汽车进行分类，左边的是汽车A，右边为汽车B。如果不使用任何数据扩增方法，深度学习模型会从汽车车头的角度来进行判别，而不是汽车具体的区别。

有哪些数据扩增方法？

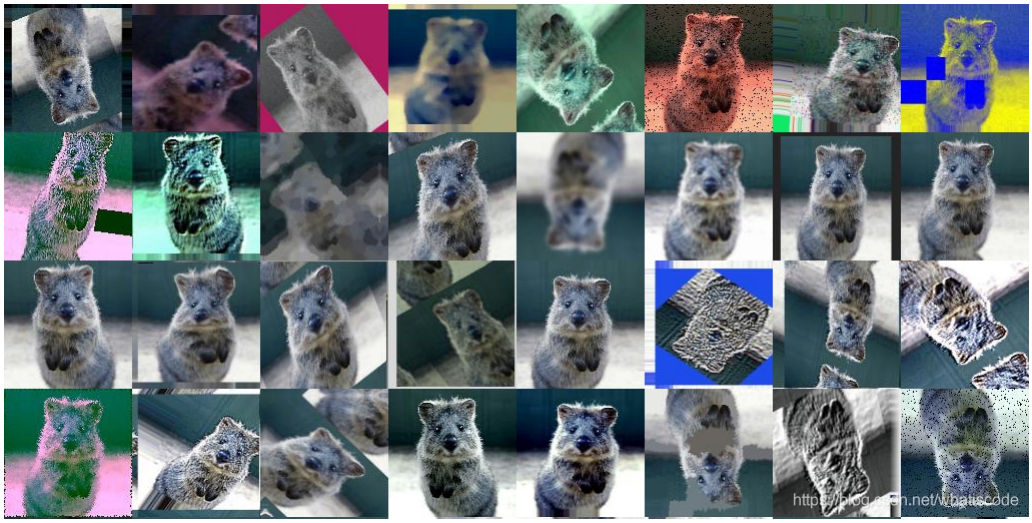
数据扩增方法有很多：从颜色空间、尺度空间到样本空间，同时根据不同任务数据扩增都有相应的区别。
对于图像分类，数据扩增一般不会改变标签；对于物体检测，数据扩增会改变物体坐标位置；对于图像分割，数据扩增会改变像素标签。

2.2 常见的数据扩增方法

在常见的数据扩增方法中，一般会从图像颜色、尺寸、形态、空间和像素等角度进行变换。当然不同的数据扩增方法可以自由进行组合，得到更加丰富的数据扩增方法。

以torchvision为例，常见的数据扩增方法包括：

- transforms.CenterCrop 对图片中心进行裁剪
- transforms.ColorJitter 对图像颜色的对比度、饱和度和零度进行变换
- transforms.FiveCrop 对图像四个角和中心进行裁剪得到五分图像
- transforms.Grayscale 对图像进行灰度变换
- transforms.Pad 使用固定值进行像素填充
- transforms.RandomAffine 随机仿射变换
- transforms.RandomCrop 随机区域裁剪
- transforms.RandomHorizontalFlip 随机水平翻转
- transforms.RandomRotation 随机旋转
- transforms.RandomVerticalFlip 随机垂直翻转



2.3 常用的数据扩增库

torchvision

https://github.com/pytorch/vision
pytorch官方提供的数据扩增库，提供了基本的数据数据扩增方法，可以无缝与torch进行集成；但数据扩增方法种类较少，且速度中等。

imgaug

https://github.com/aleju/imgaug
imgaug是常用的第三方数据扩增库，提供了多样的数据扩增方法，且组合起来非常方便，速度较快。

albumentations

https://albumentations.readthedocs.io
是常用的第三方数据扩增库，提供了多样的数据扩增方法，对图像分类、语义分割、物体检测和关键点检测都支持，速度较快。

3 PyTorch读取数据

在PyTorch中数据是通过**Dataset**进行封装，并通过**DataLoder**进行并行读取。如果想个性化自己的数据集或者数据读取方式，则需要自己重写子类。

- Dataset：对数据集的封装，提供索引方式的对数据样本进行读取
- DataLoder：对Dataset进行封装，提供批量读取的迭代读取

3.1 Dateset

`torch.utils.data.Dataset`是一个抽象类，自定义的Dataset需要**继承**它并且实现两个成员方法:`__getitem__()` 和 `__len__()`

`__getitem__()`, 即从所给路径中读取数据及其标签:

```
def __getitem__(self, index):
    img_path, label = self.data[index].img_path, self.data[index].label
    img = Image.open(img_path)
    return img, label
```

值得一提的是, pytorch还提供了很多常用的transform, 在`torchvision.transforms` 里面, 常用的有`Resize` , `RandomCrop` , `Normalize` , `ToTensor`

`__len__()`比较简单, 就是返回整个数据集的长度:

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

下面是baseline中用Dataset对数据集进行封装的代码的代码：

```
import os, sys, glob, shutil, json
```

```
import cv2

from PIL import Image
import numpy as np

import torch
from torch.utils.data.dataset import Dataset
import torchvision.transforms as transforms

class SVHNDataset(Dataset):
   # 初始化方法
    def __init__(self, img_path, img_label, transform=None):
        self.img_path = img_path
        self.img_label = img_label
        if transform is not None:
            self.transform = transform
        else:
            self.transform = None

    def __getitem__(self, index):
        img = Image.open(self.img_path[index]).convert('RGB')

        if self.transform is not None:
            img = self.transform(img)

        # 原始SVHN中类别10为数字0，采用定长字符识别
        lbl = np.array(self.img_label[index], dtype=np.int)
        lbl = list(lbl)  + (5 - len(lbl)) * [10]

        return img, torch.from_numpy(np.array(lbl[:5]))

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

# glob() 函数返回匹配指定模式的文件名或目录。
train_path = glob.glob('../input/train/*.png')
train_path.sort()
train_json = json.load(open('../input/train.json'))
train_label = [train_json[x]['label'] for x in train_json]

#################
#未加入DataLoader
data = SVHNDataset(train_path, train_label,
          transforms.Compose([
              # 缩放到固定尺寸
              transforms.Resize((64, 128)),

              # 随机颜色变换
              transforms.ColorJitter(0.2, 0.2, 0.2),

              # 加入随机旋转
              transforms.RandomRotation(5),

              # 将图片转换为pytorch 的tesntor
              # transforms.ToTensor(),

              # 对图像像素进行归一化
              # transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
            ]))
```

3.2 DataLoader

`torch.utils.data.DataLoader` 的类定义为：

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, batch_sampler=None, num_workers=0, collate_fn=<function default_collate>, pin_memory=False, 
```

主要有以下几个参数：

1. `dataset`：即上面自定义的dataset.
2. `batch_size`:每一批样本数.
3. `shuffle`:是否打乱顺序.
4. `num_worker`:读取的线程个数, 只要设置为>=1, 就可以多线程预读数据啦.
5. `collate_fn`:这个函数用来打包batch.

使用DataLoader后只有最后部分的代码和上面Dataset的代码有区别，具体代码如下：

```
#################
#加入了DataLoader
train_loader = torch.utils.data.DataLoader(
        SVHNDataset(train_path, train_label,
                    transforms.Compose([
                        transforms.Resize((64, 128)),
                        transforms.ColorJitter(0.3, 0.3, 0.2),
                        transforms.RandomRotation(5),
                        transforms.ToTensor(),
                        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ])),
    batch_size=10, # 每批样本个数
    shuffle=False, # 是否打乱顺序
    num_workers=10, # 读取的线程个数
)

for data in train_loader:
    break
```

在加入DataLoder后，数据按照批次获取，每批次调用Dataset读取单个样本进行拼接。此时data的格式为：
`torch.Size([10, 3, 64, 128]), torch.Size([10, 6])`
前者为图像文件，为batchsize * chanel * height * width次序；后者为字符标签。

4 小结

通过完成本次Task，了解了以下知识点：

1. Python中读取图像数据。主要用到了Pillow库和OpenCV库。
2. Python中数据扩增方法。数据扩增可以增加训练集的样本，从而可以缓解模型过拟合的情况，同时也可以给模型带来更强的泛化能力。常用的数据扩增库主要有：torchvision、imgaug、albumentations等等。
3. PyTorch读取数据。使用到了Dataset和DateLoader两个类。

附录 5月21日直播小结

Baseline中的知识点

1. 使用PyTorch读取数据集
2. 构建多分类模型
3. 使用ImageNet预训练模型进行微调
4. 构建训练集和验证集
5. 模型保存用于加载

Baseline如何继续深入，提高精度？（1～3实现比较简单，4～5较难）

1. 首先提高单模型在20个Epoch以内的精度（单纯baseline思路，是榜单0.7+左右的思路）

2. 尝试加入其他数据扩增方法（比如模糊和像素噪音）
3. 尝试进行多折预测结果集成（训练两折、把训练集和验证集的训练样本拼接到一起）
4. 尝试其他模型，或者改进模型的多字符训练过程；（比如修改多字符分类的损失函数）
5. 尝试其他模型，对结果进行修正（比如训练一个字符字数预测模型，对结果进行修正）

其他尝试和思考

- 这个赛题需要使用物体/字符检测模型吗？
- 如何复用YOLO、SSD或者Faster-RCNN模型？