**Pytorch数据加载-图像数据加载**

在上次的文章中我们讲述了Python数据加载高维数据的加载，主要是加载csv数据集，详情可以从以下文章中进行查看：

**Pytorch数据加载-高维数据的加载**

在实际的工程项目中，我们需要加载图像数据进行任务的训练，比如我们深度学习demo项目手写字体的识别，因为这次学习的是图像数据集的相关知识，那么我们就不得不提处理图像视频的模块torchvision，这里也顺便介绍一下torchvision中与图像数据相关的模块。

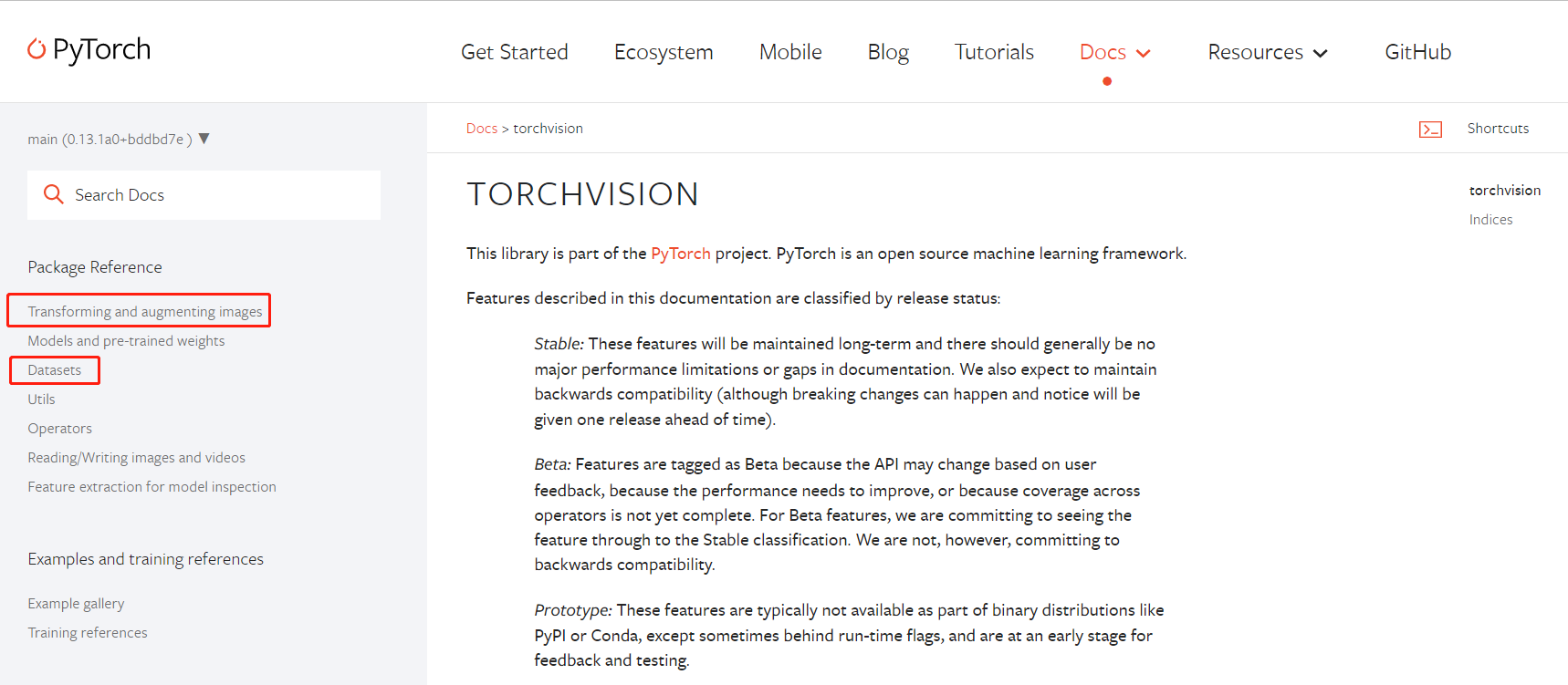
注意：torchvision是独立于PyTorch库的，因此需要进行单独的安装:

pip install torchvision

1. **Torchvision Dataset**

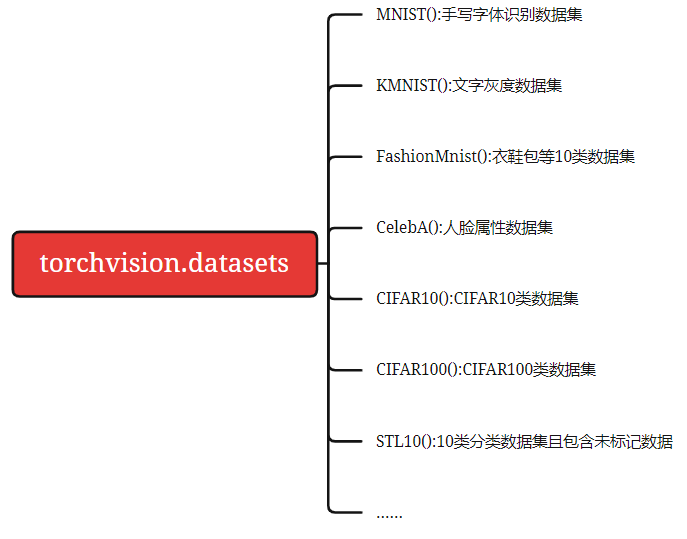
我们只简单先介绍一些关于torchvision中的一些简单的类和函数的作用，详细之处大家可以参考：

<https://pytorch.org/vision/stable>



这里我们更多关注[Transforming and augmenting images](https://pytorch.org/vision/stable/transforms.html)和Datasets这两个模块。

Torchvision在模块中提供了许多内置数据集，以及用于构建自己的数据集的实用程序类。必比如我们需要了解的torchvision.datasets，在torchvision.datasets中有这些内置的数据集，这里列举一些常见的数据集供大家看看：



大家可以从官方链接中学习：

<https://pytorch.org/vision/stable/datasets.html>

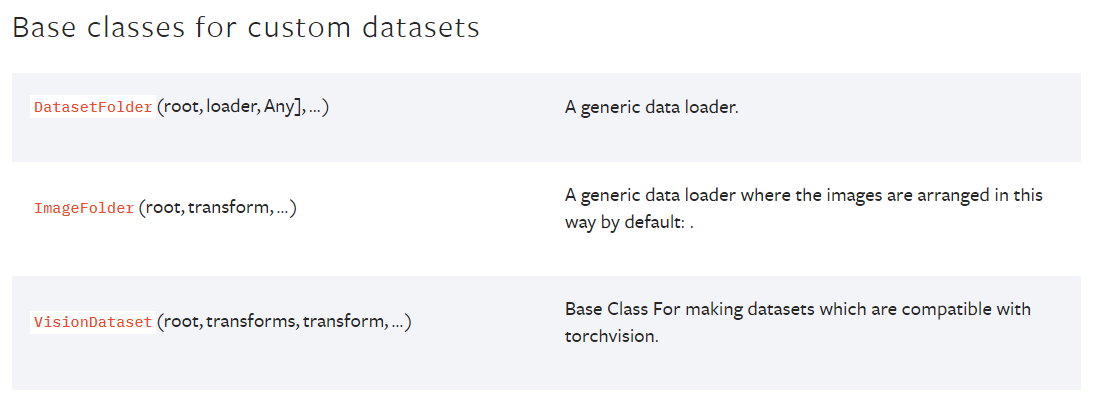
另外：

torchvision.datasets.DatasetFolder()

torchvision.datasets.[ImageFolder](https://pytorch.org/vision/stable/generated/torchvision.datasets.ImageFolder.html" \l "torchvision.datasets.ImageFolder" \o "torchvision.datasets.ImageFolder)()

torchvision.datasets.VisionDataset()

用于自定义数据加载器加载数据：



1. **直接读取数据**

一般情况下，在数据量很少的情况下，我们直接就写代码进行数据的读取。一般是这样的过程:

1. 获取需要加载的数据的路径
2. 在获取的路径下读取文件夹中的文件（假设是分类任务）
3. 获取文件图片数据和所属标签
4. 如果存在批量的情况下，就获取多个数据和标签进行返回

直接读取数据比较好理解，但是存在的问题也是明显的:

1. 大量数据使用Numpy格式进行加载会占用大量的内容

单线程下的数据导入速度较慢

因此诸如我们之前的文章所提及到的，我们可以使用PyTorch中的torch.utils.Data.DataLoader进行数据相关的操作，基于参数mini-batch使用多线程进行数据加载的并行处理。接下来我们看怎么基于torchvision()来进行一些数据的操作。

1. **加载torchvision.datasets()中的数据集**

加载torchvision.datasets()中的数据集比较简单，我们这里以CIFAR10数据集为例，这里稍微介绍一下这个数据集：

CIFAR10数据集共有60000张彩色图像，这些图像的大小为32\*32\*3，分为10个类，每个类6000张，如下所示：



对于此，我们加载的案例代码如下：

import numpy as np  
import torch  
import torchvision  
import torch.utils.data as Data  
from matplotlib import pyplot as plt  
from torchvision.datasets import CIFAR10,FashionMNIST  
import torchvision.transforms as transforms  
from torchvision.datasets import ImageFolder

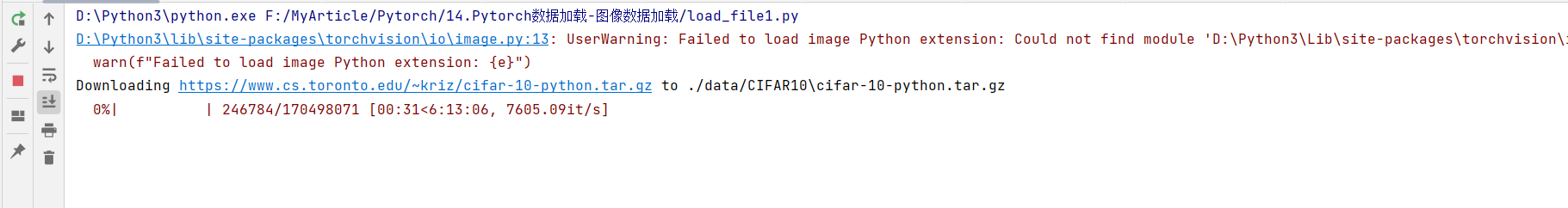
*# 创建训练数据集*trainData = CIFAR10(  
 root = "./data/CIFAR10",  
 train = True,  
 transform = transforms.ToTensor(),

download = True  
*# download = False*)  
*# 数据加载器*trainLoader = Data.DataLoader(  
 dataset=trainData,  
 batch\_size=64,  
 shuffle=True  
)  
print('batch 个数为：',len(trainLoader))  
*# 数据显示*for step ,(X,Y) in enumerate(trainLoader):  
 if step < 1:  
 imgs = torchvision.utils.make\_grid(X,padding=0)  
 imgs = np.transpose(imgs,(1,2,0))  
 plt.imshow(imgs)  
 plt.show()

# 输出

batch 个数为：782

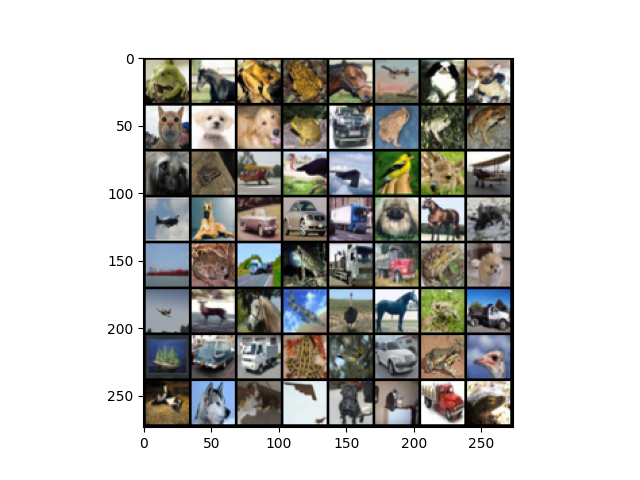
运行过程中我们会发现一个加载数据的过程，如下所示：



这是因为我们的需要的数据集-CIFAR10的train数据集，在路径root = "./data/CIFAR10"下是不存在的，因此会根据参数**download = True**从网上下载。笔者在下载文件之后，发现上述代码将CIFAR10的数据，包括训练集合测试集，不管我们使用的参数train设置为True或者是False。



另外最后显示的图像如下所示：



这里展示1个批次的数据，每个mini\_batch是64张图片。

这里解释一下其中的一些代码的含义：

1. **transform = transforms.ToTensor()：**使用ToTensor()将形状为(H,W,C)的numpy.ndarray或img转为shape为(C,H,W)的tensor，其将每一个数值归一化到[0,1]，其归一化方法比较简单。
2. **torchvision.utils.make\_grid(X,padding=0)：**将多张图像组成的X图像合并为1张图像。

**np.transpose(imgs,(1,2,0))：**对原数据进行换轴操作，如np.transpose()传入的参数为（1,0），即将原数组的x,y轴互换。即将图像格式为[通道数，长，宽]转变为[长，宽，通道数]

1. 另外，上述代码中数据加载器部分使用了shuffle=True，因此每一次运行上述的代码展示的图片都是随机性的。

至此，我们就可以完成批量加载数据的需求，可以看出我们在此过程中借助了Dataloader来完成数据集的加载操作。大家可以参考源码的数据加载方式，个人来看还是很容易看懂的，路径如下：

**D:\Python3\Lib\site-packages\torchvision\datasets\cifar.py**

1. **加载下载的数据集**

上述数据文件来源于互联网，本次接下来我们通过编写代码实现加载上述下载的文件，其实大家可以参考官网的文档的方式，接下来我们看一下我们的代码：

import numpy as np  
import torchvision  
from torch.utils.data import Dataset,DataLoader  
import matplotlib.pyplot as plt  
import torchvision.transforms as transforms  
  
class MyCIFAR10(Dataset):  
 def \_\_init\_\_(self,  
 root,  
 train=True,  
 transform=None,  
 target\_transform=None,  
 ):  
 self.train = train  
 self.transform = transform  
 self.target\_transform = target\_transform  
 self.data, self.labels = [], []  
  
 def myload(file):  
 import pickle  
 with open(file, 'rb') as fo:  
 dict = pickle.load(fo, encoding='bytes')  
 return dict  
  
 if self.train:  
 for i in range(1, 6):  
 file = root + "\\data\_batch\_" + str(i)  
 data = myload(file)  
 for item, label in zip(data[b'data'], data[b'labels']):  
 self.data.append(item)  
 self.labels.append(label)  
 else:  
 file = root + "\\test\_batch"  
 data = myload(file)  
 for item, label in zip(data[b'data'], data[b'labels']):  
 self.data.append(item)  
 self.labels.append(label)  
  
 def \_\_getitem\_\_(self, index):  
 image = self.data[index]  
 label = self.labels[index]  
 image, label = np.array(image), np.array(label)  
 return image, label  
  
 def \_\_len\_\_(self):  
 return len(self.data)

上述代码我们改成了类的形式，并用来读取下载过的cifar10数据集，那么我们该怎么调用呢？我们可以使用以下代码：

我们先构造一个对象：

*# MyCIFAR10*mycifar10 = MyCIFAR10(  
 '.\data\CIFAR10\cifar-10-batches-py',  
 train=True,  
 transform=transforms.ToTensor()  
)

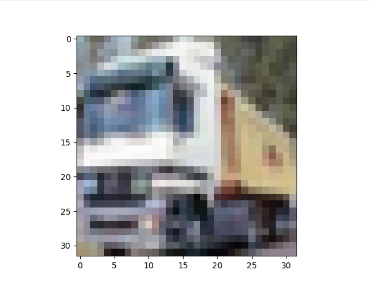
接着构建一个数据加载器，并设置一定的参数，其中dataset是上述：

*# 数据加载器*trainLoader = DataLoader(  
 dataset=mycifar10,  
 batch\_size=64,  
 shuffle=True,  
)

接下来我们取得其中的一个文件，并进行展示：

image, label = mycifar10.\_\_getitem\_\_(1)  
img = image.reshape(3, 32, 32).astype('float32')  
imgPic = np.transpose(img, (1, 2, 0))/255  
plt.imshow(imgPic)  
plt.show()  
print('batch 个数为：',len(trainLoader))  
print('长度：',len(image))

# 输出：



batch 个数为： 782

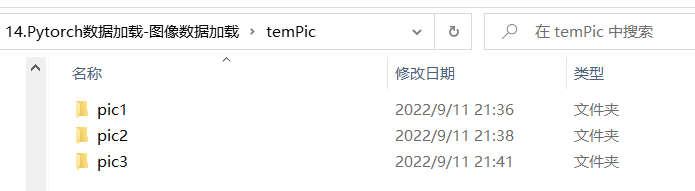
长度： 3072

可以看出大概是一个卡车的模样，另外上述代码中\_\_getitem\_\_(1)表示获取第一个图片文件。

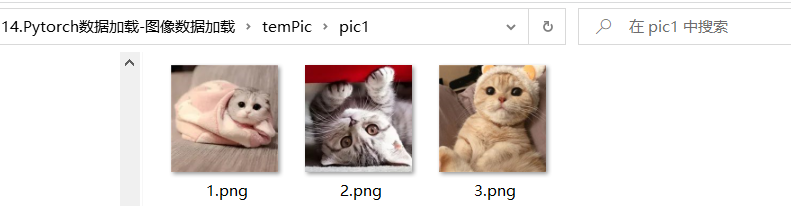
参考之前的代码还是很好理解的。

1. **加载本地文件夹中的数据集**

最后一个数据加载我们打算从本地的数据文件夹中加载数据，假设现在我们的目录是这样的情况：



各个子文件夹pic中存在一些图片，比如这样：



接下来我们编写代码进行读取其中的文件：

import numpy as np  
import torchvision.transforms as transforms  
from matplotlib import pyplot as plt  
from torchvision.datasets import ImageFolder  
from torch.utils.data import DataLoader  
  
*# 设置变换操作集合*train\_data\_transforms = transforms.Compose([  
 *# 随机裁剪大小为100的图*

transforms.RandomResizedCrop(100),  
 transforms.ToTensor()  
])  
*# 本地文件路径*  
data\_dir = "./temPic/"

*# 加载数据集*  
mydata = ImageFolder(data\_dir,  
 transform=train\_data\_transforms)

mydata\_load = DataLoader(  
 mydata,  
 batch\_size=2,  
 shuffle=True,  
 *# num\_workers=1*)  
*# 获取标签等信息*print(len(mydata))  
print(mydata.targets)  
print(len(mydata\_load))  
  
*# 获取batch数据*for step,(X,Y) in enumerate(mydata\_load):  
 if step < 1:  
 print(X.shape)  
 print(Y.shape)  
 print(X[0].shape)  
 *# print(X[0].shape)  
 # print(X[0],'\n')* im = np.array(X[0])  
 *# print(im)* img = np.transpose(im, (1, 2, 0))  
 plt.imshow(img)  
 plt.show()

# 输出

8

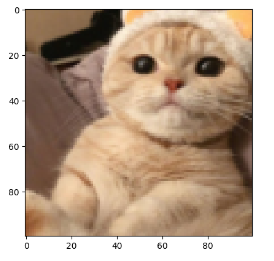
[0, 0, 0, 1, 1, 1, 2, 2]

4

torch.Size([2, 3, 100, 100])

torch.Size([2])

torch.Size([3, 100, 100])



稍微解释一下输出的数据：

1. temPic下共有3个文件夹，第一个文件夹中有3个文件，第二个文件夹中有3个文件，第三个文件夹中有2个文件，所以输出8。
2. DataLoader加载数据的时候自动将文件夹转换为三类标签数据
3. 由于batch\_size=2,8个图片2个批次，所以输出4.

**至此，我们就完成了各种不同形式图片的加载了，总体上还是比较适用的，这里提出一些优化，对于大型数据集，我们在加载的时候可以这样：**

（1）在DataLoader中将num\_workers设置为大于1的数值，可以并行加载数据(利用多核处理器加快载入数据的效率），这点需要注意的是，在运行的时候需要在if \_\_name\_\_ == '\_\_main\_\_'的情况下使用。

（2）有一些图像分类的数据集太大了导致内存紧张，对弈这种方式可以使用getitem生成器得方式去逐步加载程序需要的数据集。

1. **参考文档**

<https://pytorch.org/docs/stable/data.html>

<https://github.com/lyhue1991/eat_pytorch_in_20_days>

<https://zhuanlan.zhihu.com/p/130673468>

https://blog.csdn.net/caojianhua2018/article/details/112553162