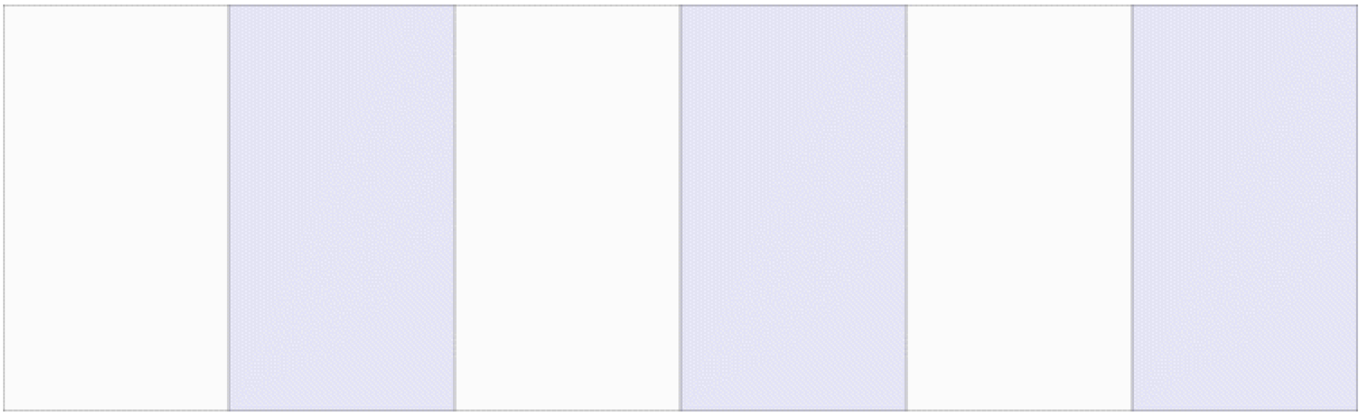
实践教程｜PyTorch数据导入机制与标准化代码模板

极市平台 2023-08-06 22:00:28 发表于广东 手机阅读

以下文章来源于机器学习实验室，作者louwill

机器学习实验室
专注于机器学习和深度学习技术与实践。

↑ 点击**蓝字** 关注极市平台



作者丨louwill

来源丨机器学习实验室

编辑丨极市平台

极市导读

本文将聚焦于PyTorch的自定义数据读取pipeline模板和相关trciks以及如何优化数据读取的pipeline等。 >>加入极市CV技术交流群，走在计算机视觉的最前沿

这篇文章笔者将和大家聚焦于PyTorch的自定义数据读取pipeline模板和相关trciks以及如何优化数据读取的pipeline等。我们从PyTorch的数据对象类Dataset开始。Dataset在PyTorch中的模块位于utils.data下。

```
from torch.utils.data import Dataset
```

本文将围绕Dataset对象分别从原始模板、torchvision的transforms模块、使用pandas来辅助读取、torch内置数据划分功能和DataLoader来展开阐述。

Dataset原始模板

PyTorch官方为我们提供了自定义数据读取的标准化代码代码模块，作为一个读取框架，我们这里称之为原始模板。其代码结构如下：

```
from torch.utils.data import Dataset
class CustomDataset(Dataset):
    def __init__(self, ...):
        # stuff

    def __getitem__(self, index):
        # stuff
        return (img, label)

    def __len__(self):
        # return examples size
        return count
```

根据这个标准化的代码模板，我们只需要根据自己的数据读取任务，分别往__init__()、__getitem__()和__len__()三个方法里添加读取逻辑即可。作为PyTorch范式下的数据读取以及为了后续的data loader，三个方法缺一不可。其中：

- __init__()函数用于初始化数据读取逻辑，比如读取包含标签和图片地址的csv文件、定义transform组合等。
- __getitem__()函数用来返回数据和标签。目的上是为了能够被后续的dataloader所调用。
- __len__()函数则用于返回样本数量。

现在我们往这个框架里填几行代码来形成一个简单的数字案例。创建一个从1到100的数字例子：

```
from torch.utils.data import Dataset
class CustomDataset(Dataset):
    def __init__(self):
        self.samples = list(range(1, 101))
    def __len__(self):
        return len(self.samples)
    def __getitem__(self, idx):
```

```
        return self.samples[idx]

if __name__ == '__main__':
    dataset = CustomDataset()
    print(len(dataset))
    print(dataset[50])
    print(dataset[1:100])
```

```
Microsoft Windows [版本 10.0.18362.476]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\92070>D:

D:\>cd Deep Learning\PyTorch

D:\Deep Learning\PyTorch>python torch_dataset.py
100
51
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
 94, 95, 96, 97, 98, 99, 100]

D:\Deep Learning\PyTorch>
```

添加torchvision.transforms

然后我们来看如何从内存中读取数据以及如何在读取过程中嵌入torchvision中的transforms功能。torchvision是一个独立于torch的关于数据、模型和一些图像增强操作的辅助库。主要包括datasets默认数据集模块、models经典模型模块、transforms图像增强模块以及utils模块等。在使用torch读取数据的时候，一般会搭配上transforms模块对数据进行一些处理和增强工作。

添加了tranforms之后的读取模块可以改写为：

```
from torch.utils.data import Dataset
from torchvision import transforms as T

class CustomDataset(Dataset):
    def __init__(self, ...):
        # stuff
        ...
        # compose the transforms methods
        self.transform = T.Compose([T.CenterCrop(100),
                                T.ToTensor()])

    def __getitem__(self, index):
        # stuff
        ...
```

```
        data = # Some data read from a file or image
        # execute the transform
        data = self.transform(data)
        return (img, label)

    def __len__(self):
        # return examples size
        return count

if __name__ == '__main__':
    # Call the dataset
    custom_dataset = CustomDataset(...)
```

可以看到，我们使用了Compose方法来把各种数据处理方法聚合到一起进行定义数据转换方法。通常作为初始化方法放在__init__()函数下。我们以猫狗图像数据为例进行说明。

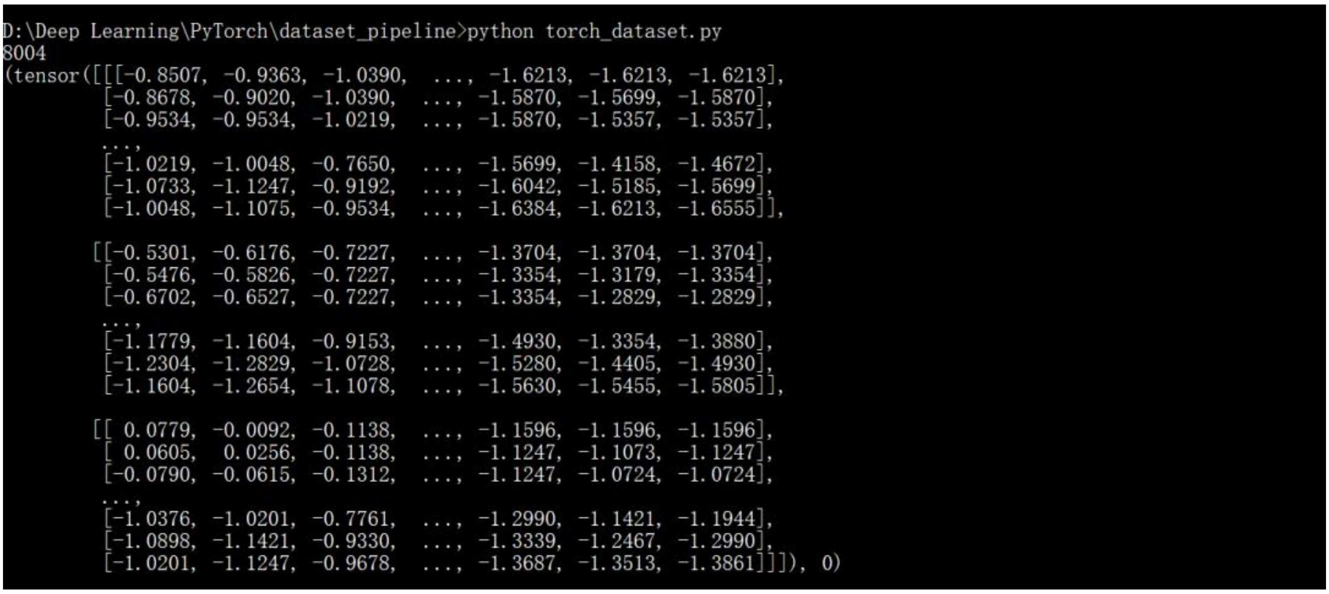
定义数据读取方法如下：

```
class DogCat(Dataset):
    def __init__(self, root, transforms=None, train=True, val=False):
        """
        get images and execute transforms.
        """
        self.val = val
```

```
        imgs = [os.path.join(root, img) for img in os.listdir(root)]
        # train: Cats_Dogs/trainset/cat.1.jpg
        # val: Cats_Dogs/valset/cat.10004.jpg
        imgs = sorted(imgs, key=lambda x: x.split('.')[-2])
        self.imgs = imgs
        if transforms is None:
            # normalize
            normalize = T.Normalize(mean = [0.485, 0.456, 0.406],
                                     std = [0.229, 0.224, 0.225])
            # trainset and valset have different data transform
            # trainset need data augmentation but valset don't.
            # valset

            if self.val:
                self.transforms = T.Compose([
                    T.Resize(224),
                    T.CenterCrop(224),
                    T.ToTensor(),
                    normalize
                ])
            # trainset
            else:
                self.transforms = T.Compose([
                    T.Resize(256),
                    T.RandomResizedCrop(224),
                    T.RandomHorizontalFlip(),
                    T.ToTensor(),
                    normalize
                ])

    def __getitem__(self, index):
        """
        return data and label
        """
        img_path = self.imgs[index]
        label = 1 if 'dog' in img_path.split('/')[-1] else 0
        data = Image.open(img_path)
        data = self.transforms(data)
        return data, label

    def __len__(self):
        """
        return images size.
        """
```

```
        return len(self.imgs)

if __name__ == "__main__":
    train_dataset = DogCat('./Cats_Dogs/trainset/', train=True)
    print(len(train_dataset))
    print(train_dataset[0])
```

因为这个数据集已经分好了训练集和验证集，所以在读取和transforms的时候需要进行区分。运行示例如下：



```
D:\Deep Learning\PyTorch\dataset_pipeline>python torch_dataset.py
8004
(tensor([[[-0.8507, -0.9363, -1.0390,  ..., -1.6213, -1.6213, -1.6213],
         [-0.8678, -0.9020, -1.0390,  ..., -1.5870, -1.5699, -1.5870],
         [-0.9534, -0.9534, -1.0219,  ..., -1.5870, -1.5357, -1.5357],
         ...,
         [-1.0219, -1.0048, -0.7650,  ..., -1.5699, -1.4158, -1.4672],
         [-1.0733, -1.1247, -0.9192,  ..., -1.6042, -1.5185, -1.5699],
         [-1.0048, -1.1075, -0.9534,  ..., -1.6384, -1.6213, -1.6555]],

        [[-0.5301, -0.6176, -0.7227,  ..., -1.3704, -1.3704, -1.3704],
         [-0.5476, -0.5826, -0.7227,  ..., -1.3354, -1.3179, -1.3354],
         [-0.6702, -0.6527, -0.7227,  ..., -1.3354, -1.2829, -1.2829],
         ...,
         [-1.1779, -1.1604, -0.9153,  ..., -1.4930, -1.3354, -1.3880],
         [-1.2304, -1.2829, -1.0728,  ..., -1.5280, -1.4405, -1.4930],
         [-1.1604, -1.2654, -1.1078,  ..., -1.5630, -1.5455, -1.5805]],

        [[ 0.0779, -0.0092, -0.1138,  ..., -1.1596, -1.1596, -1.1596],
         [ 0.0605,  0.0256, -0.1138,  ..., -1.1247, -1.1073, -1.1247],
         [-0.0790, -0.0615, -0.1312,  ..., -1.1247, -1.0724, -1.0724],
         ...,
         [-1.0376, -1.0201, -0.7761,  ..., -1.2990, -1.1421, -1.1944],
         [-1.0898, -1.1421, -0.9330,  ..., -1.3339, -1.2467, -1.2990],
         [-1.0201, -1.1247, -0.9678,  ..., -1.3687, -1.3513, -1.3861]]]), 0)
```

与pandas一起使用

很多时候数据的目录地址和标签都是通过csv文件给出的。如下所示：

	image_name	comment_number	comment
0	1000092795.jpg	0	Two young guys with shaggy hair look at their...
1	1000092795.jpg	1	Two young , White males are outside near many...
2	1000092795.jpg	2	Two men in green shirts are standing in a yard .
3	1000092795.jpg	3	A man in a blue shirt standing in a garden .
4	1000092795.jpg	4	Two friends enjoy time spent together .
5	10002456.jpg	0	Several men in hard hats are operating a gian...
6	10002456.jpg	1	Workers look down from up above on a piece of...
7	10002456.jpg	2	Two men working on a machine wearing hard hats .
8	10002456.jpg	3	Four men on top of a tall structure .
9	10002456.jpg	4	Three men on a large rig .

此时在数据读取的pipeline中我们需要在__init__()方法中利用pandas把csv文件中包含的图片地址和标签融合进去。相应的数据读取pipeline模板可以改写为：

```
class CustomDatasetFromCSV(Dataset):
    def __init__(self, csv_path):
        """
        Args:
            csv_path (string): path to csv file
            transform: pytorch transforms for transforms and tensor conversion
        """
        # Transforms
        self.to_tensor = transforms.ToTensor()
        # Read the csv file
        self.data_info = pd.read_csv(csv_path, header=None)
        # First column contains the image paths
        self.image_arr = np.asarray(self.data_info.iloc[:, 0])
        # Second column is the labels
        self.label_arr = np.asarray(self.data_info.iloc[:, 1])
        # Calculate len
        self.data_len = len(self.data_info.index)

    def __getitem__(self, index):
        # Get image name from the pandas df
        single_image_name = self.image_arr[index]
```

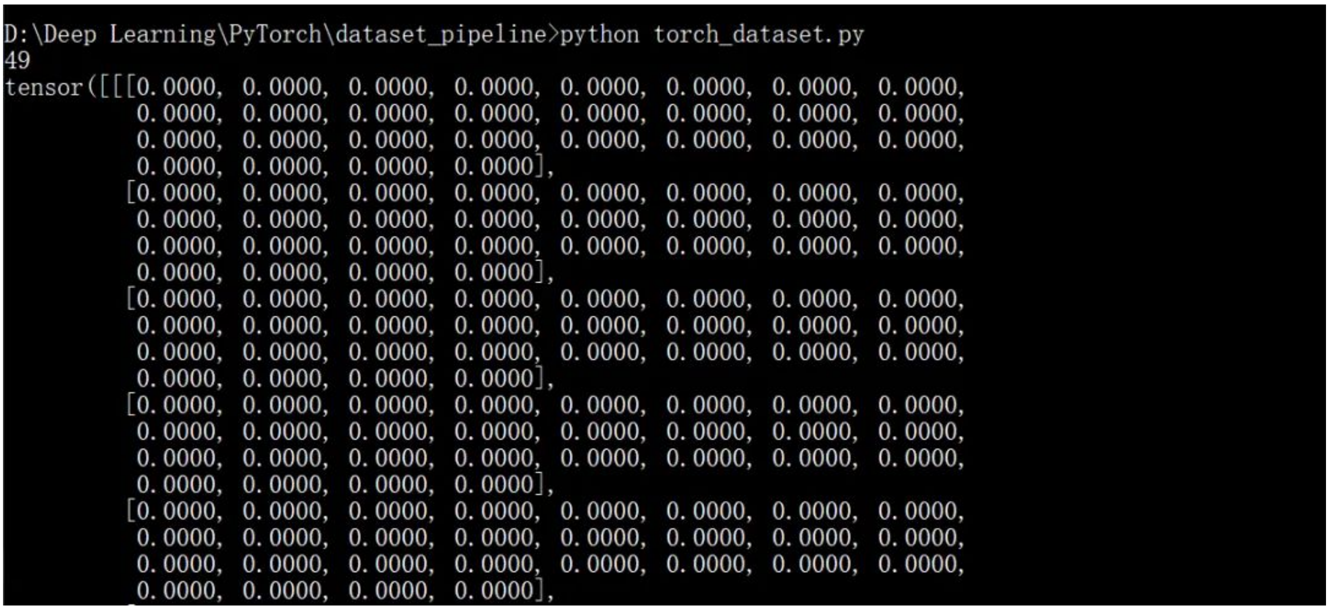
```
        # Open image
        img_as_img = Image.open(single_image_name)
        # Transform image to tensor
        img_as_tensor = self.to_tensor(img_as_img)
        # Get label of the image based on the cropped pandas column
        single_image_label = self.label_arr[index]
        return (img_as_tensor, single_image_label)

    def __len__(self):
        return self.data_len

if __name__ == "__main__":
    # Call dataset
    dataset =  CustomDatasetFromCSV('./labels.csv')
```

以mnist_label.csv文件为示例：

```
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision import transforms as T
from PIL import Image
import os
import numpy as np
import pandas as pd

class CustomDatasetFromCSV(Dataset):
    def __init__(self, csv_path):
        """
        Args:
            csv_path (string): path to csv file
            transform: pytorch transforms for transforms and tensor conversion
        """
        # Transforms
        self.to_tensor = T.ToTensor()
        # Read the csv file
        self.data_info = pd.read_csv(csv_path, header=None)
        # First column contains the image paths
        self.image_arr = np.asarray(self.data_info.iloc[:, 0])
        # Second column is the labels
        self.label_arr = np.asarray(self.data_info.iloc[:, 1])
```

```
        # Third column is for an operation indicator
        self.operation_arr = np.asarray(self.data_info.iloc[:, 2])
        # Calculate len
        self.data_len = len(self.data_info.index)

    def __getitem__(self, index):
        # Get image name from the pandas df
        single_image_name = self.image_arr[index]
        # Open image
        img_as_img = Image.open(single_image_name)
        # Check if there is an operation
        some_operation = self.operation_arr[index]
        # If there is an operation
        if some_operation:
            # Do some operation on image
            # ...
            # ...
            pass

        # Transform image to tensor
        img_as_tensor = self.to_tensor(img_as_img)
        # Get label of the image based on the cropped pandas column
        single_image_label = self.label_arr[index]
        return (img_as_tensor, single_image_label)

    def __len__(self):
        return self.data_len

if __name__ == "__main__":
    transform = T.Compose([T.ToTensor()])
    dataset = CustomDatasetFromCSV('./mnist_labels.csv')
    print(len(dataset))
    print(dataset[5])
```

运行示例如下：



```
D:\Deep Learning\PyTorch\dataset_pipeline>python torch_dataset.py
49
tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
```

训练集验证集划分

一般来说，为了模型训练的稳定，我们需要对数据划分训练集和验证集。torch的Dataset对象也提供了random_split函数作为数据划分工具，且划分结果可直接供后续的DataLoader使用。

以kaggle的花朵数据为例：

```python
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision import transforms as T
from torch.utils.data import random_split

transform = T.Compose([
    T.Resize((224, 224)),
    T.RandomHorizontalFlip(),
    T.ToTensor()
 ])

dataset = ImageFolder('./flowers_photos', transform=transform)
print(dataset.class_to_idx)

trainset, valset = random_split(dataset,
                [int(len(dataset)*0.7), len(dataset)-int(len(dataset)*0.7)])

trainloader = DataLoader(dataset=trainset, batch_size=32, shuffle=True, num_workers=1)
for i, (img, label) in enumerate(trainloader):
    img, label = img.numpy(), label.numpy()
    print(img, label)

valloader = DataLoader(dataset=valset, batch_size=32, shuffle=True, num_workers=1)
for i, (img, label) in enumerate(trainloader):
```

```
    img, label = img.numpy(), label.numpy()
    print(img.shape, label)
```

这里使用了ImageFolder模块，可以直接读取各标签对应的文件夹，部分运行示例如下：

```
(32, 3, 224, 224) [1 4 4 4 4 0 4 0 3 2 0 4 4 3 4 4 0 1 4 2 4 4 2 4 3 4 4 2 3 0 3 2]
(32, 3, 224, 224) [4 1 0 4 2 3 4 1 1 2 1 3 2 2 2 3 1 1 3 4 1 0 3 1 2 1 3 4 4 1 3 1]
(32, 3, 224, 224) [2 1 4 4 4 4 4 0 1 4 1 0 2 3 0 1 1 3 4 0 0 3 3 3 0 1 4 3 1 2 1 4]
(32, 3, 224, 224) [4 1 2 1 2 4 1 1 1 2 2 1 3 0 2 1 3 0 0 3 2 1 2 1 1 2 0 4 1 2 3 3]
(32, 3, 224, 224) [4 4 3 3 4 3 2 3 3 2 1 1 2 3 0 1 2 1 0 4 1 0 1 1 0 2 3 3 4 1 3 1]
(32, 3, 224, 224) [2 0 2 0 1 3 0 2 2 1 0 3 3 0 0 1 2 0 4 2 3 4 3 4 4 4 2 2 0 4 4 2]
(32, 3, 224, 224) [1 0 3 0 3 4 1 0 0 4 2 3 4 3 1 4 4 1 0 4 0 1 4 2 2 2 0 1 4 2 0 3]
(32, 3, 224, 224) [1 2 4 3 2 0 4 1 1 4 4 3 4 3 2 3 0 1 4 2 0 3 0 3 3 1 3 4 4 0 4 4]
(32, 3, 224, 224) [0 1 4 4 2 1 0 2 4 4 3 1 4 3 2 4 1 2 4 4 2 1 1 1 3 1 4 0 0 1 2 4]
(32, 3, 224, 224) [2 3 4 2 3 2 1 4 4 0 0 0 1 0 0 1 1 2 2 0 0 1 3 0 0 2 2 0 3 1 0 3]
(32, 3, 224, 224) [2 4 1 0 2 3 4 0 2 2 2 3 3 0 1 4 2 4 4 1 0 1 1 4 2 1 0 0 1 2 3 4]
(32, 3, 224, 224) [4 1 4 4 3 1 0 1 1 2 1 4 4 0 0 4 4 2 2 2 0 3 2 4 4 3 3 3 4 3 2 1]
(32, 3, 224, 224) [1 3 0 2 4 4 3 3 4 1 1 3 4 4 4 1 4 4 2 0 1 0 1 1 4 4 4 3 1 2 4 0]
(32, 3, 224, 224) [3 2 1 1 0 4 3 4 1 0 0 3 4 3 1 0 1 3 0 2 0 3 4 2 2 1 0 1 3 1 1 1]
(32, 3, 224, 224) [0 3 2 4 4 1 2 2 4 1 1 1 0 2 2 2 4 4 1 2 4 4 1 4 3 1 4 1 2 0 4 3]
(32, 3, 224, 224) [0 0 0 2 1 4 1 0 0 1 4 2 2 1 1 4 4 1 1 4 0 0 1 1 4 2 0 3 0 3 3 3]
```

使用DataLoader

dataset方法写好之后，我们还需要使用DataLoader将其逐个喂给模型。上一节的数据划分我们已经用到了DataLoader函数。从本质上来讲，DataLoader只是调用了__getitem__()方法并按批次返回数据和标签。使用方法如下：

```
from torch.utils.data import DataLoader
from torchvision import transforms as T

if __name__ == "__main__":
    # Define transforms
    transformations = T.Compose([T.ToTensor()])
    # Define custom dataset
    dataset = CustomDatasetFromCSV('./labels.csv')
    # Define data loader
    data_loader = DataLoader(dataset=dataset, batch_size=10, shuffle=True)
    for images, labels in data_loader:
        # Feed the data to the model
```

以上就是PyTorch读取数据的Pipeline主要方法和流程。基于Dataset对象的基本框架不变，具体细节可自定义化调整。

公众号后台回复“极市直播”获取100+期极市技术直播回放+PPT

极市平台

为计算机视觉开发者提供全流程算法开发训练平台，以及大咖技术分享、社区交流、...

848篇原创内容

公众号

极市干货

极视角动态：2023GCVC全球人工智能视觉产业与技术生态伙伴大会在青岛圆满落幕！｜极视角助力构建城市大脑中枢，芜湖市湾沚区智慧城市运行管理中心上线！

数据集： 面部表情识别相关开源数据集资源汇总｜打架识别相关开源数据集资源汇总（附下载链接）｜口罩识别检测开源数据集汇总

经典解读： 多模态大模型超详细解读专栏

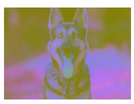
点击阅读原文进入CV社区

收获更多技术干货

阅读原文

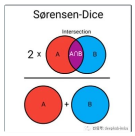
喜欢此内容的人还喜欢

实践教程｜使用 OpenCV 进行特征提取（颜色、形状和纹理）



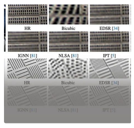
极市平台

9个数据科学中常见距离度量总结以及优缺点概述



极市平台

ICCV 2023｜南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）



极市平台