数据获取

torchvision 内置数据集

torchvision — Torchvision 0.22 documentation 通过官方文档中的 Datasets — Torchvision 0.22 documentation Datasets 部分，可以查看 `torchvision` 提供的内置数据集，参考官方文档引入即可。 如果觉得下载太慢，可以从说明文档找到它的链接，用迅雷之类的下载。

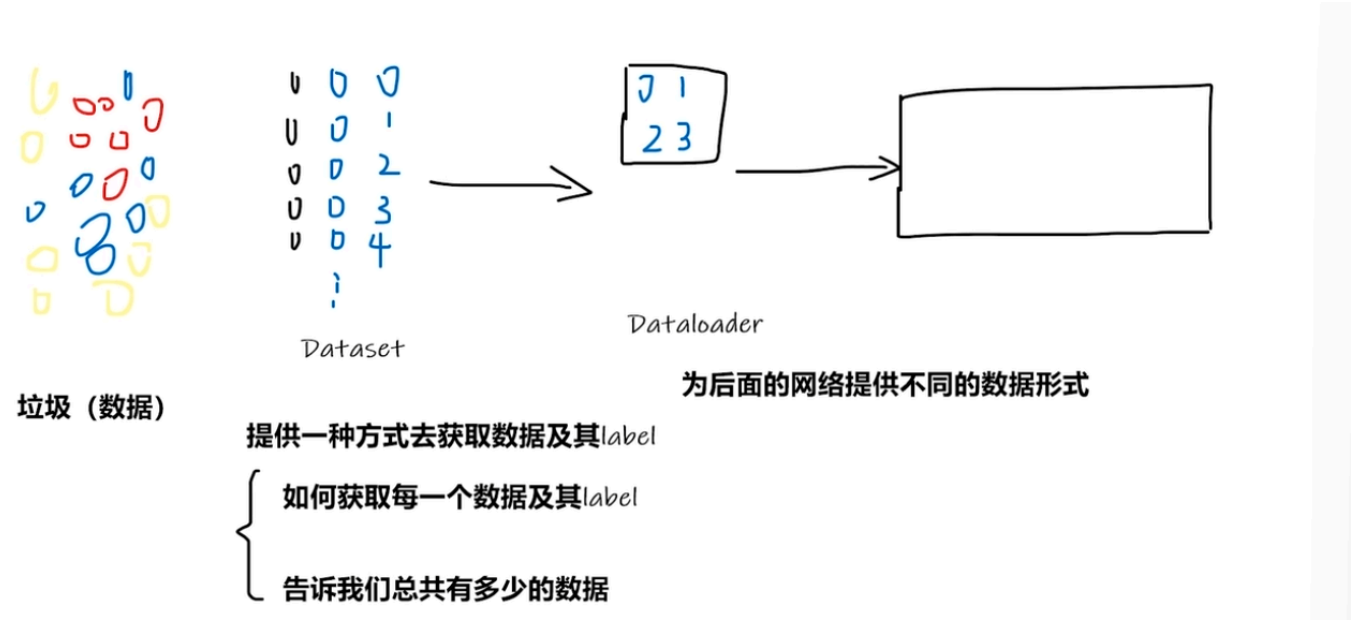
示例：

PYTHON

```python
import torchvision
from torch.utils.tensorboard import SummaryWriter

data_transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor()
])

# target_transform 用于对 target(目标分类) 进行变换。在 mixup 等操作时可能有用
# download 一直设置为 True 也没问题，并不会重复下载，目标路径里有过数据集了会直接使用
train_set = torchvision.datasets.CIFAR10(root='./data', train=True, transform=data_transform, target_transform=None, download=True)
test_set = torchvision.datasets.CIFAR10(root='./data', train=False, transform=data_transform, target_transform=None, download=True)

writer = SummaryWriter("logs")

for i in range(10):
    writer.add_image("train_set", train_set[i][0], i)

writer.close()
```

数据读取

分为两个类，`dataset` 和 `dataloader`，负责数据的初始化、获取、变换、增强等等：



Dataset

抽象类，所有数据集类都要继承这个类，并重写 `__getitem__` 方法[1]。

很多时候，我们看到别人的代码里用 `变量.xxx`，调用对应的方法/属性，但我们往往都不知道为什么有这些方法/属性。这时候就可以通过在 PyCharm 中的 Python 控制台创建相应的变量，从而查看（Python 控制台会自动创建变量视图）。

要获取所有图片及其标签，首先要创建所有图片的地址列表（路径+文件名），通过 `import os` 来完成。常用的有 `os.listdir`，`os.path.join` 等。

Python 类中的 `self` 理解为类中的一个全局变量即可。

以下是示例代码：

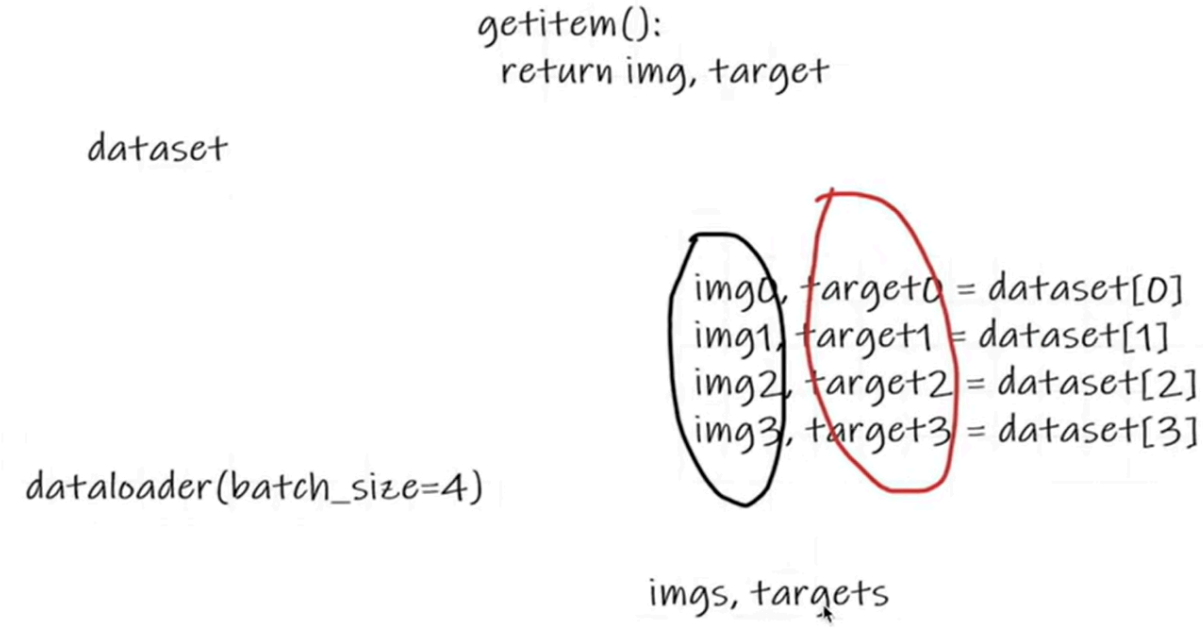
PYTHON

```python
import os
from torch.utils.data import Dataset
from PIL import Image

class MyDataset(Dataset):
    def __init__(self, root_dir, label_dir):
        self.root_dir = root_dir
        self.label_dir = label_dir
        self.img_list = os.listdir(os.path.join(root_dir, label_dir))

    def __getitem__(self, index):
        img_name = self.img_list[index]
        img_path = os.path.join(self.root_dir, self.label_dir,
img_name)
        image = Image.open(img_path)
        label = self.label_dir
        return image, label
    def __len__(self):
        return len(self.img_list)

root_dir =
"G:\\github\\learnPyTorch\\dataset\\hymenoptera_data\\train"
ants_label_dir = "ants"
bees_label_dir = "bees"

ant_ds = MyDataset(root_dir, label_dir)
bee_ds = MyDataset(root_dir, bees_label_dir)

# 可以直接对数据集进行拼接，常用于数据增强、构造子数据集、构造 mock 数据集
等
train_ds = ant_ds + bee_ds
```

DataLoader

Dataset 定义了数据集是怎么样的，每次获取什么，`DataLoader` 则定义了怎么将数据送给神经网络。 指定 `batch_size > 1` 后，`DataLoader` 会将多个数据打包成更大的张量 `[N, ...]` (N

为 batch_size) 并返回：



使用示例：

PYTHON

```
import torchvision
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

test_set = torchvision.datasets.CIFAR10(root='./data', train=False,
transform=torchvision.transforms.ToTensor(), download=True)
# Windows 下 num_workers 好像有问题，如果出现 Pipe Broken，可以尝试将其值
设置为 0test_dataloader = DataLoader(dataset=test_set, batch_size=64,
shuffle=True, num_workers=0, pin_memory=False, drop_last=False)

writer = SummaryWriter("dataloader")

for epoch in range(2):
    # 一个循环就是一个 epoch，过一遍数据集
    for step, (imgs, targets) in enumerate(test_dataloader):
        writer.add_images(f"test_set_epoch{epoch}", imgs, step)
        # print(imgs.shape)
writer.close()
```

数据可视化

Tensorboard

Tips: 在 PyTorch 中，想要查看函数、包怎么用，通常长按 **Ctrl** 键点击 / 鼠标悬停，然后查看其注释（相当于说明文档）

打开 `tensorboard`：

```shell
tensorboard --logdir=日志路径 [ --port=端口号 ]
```

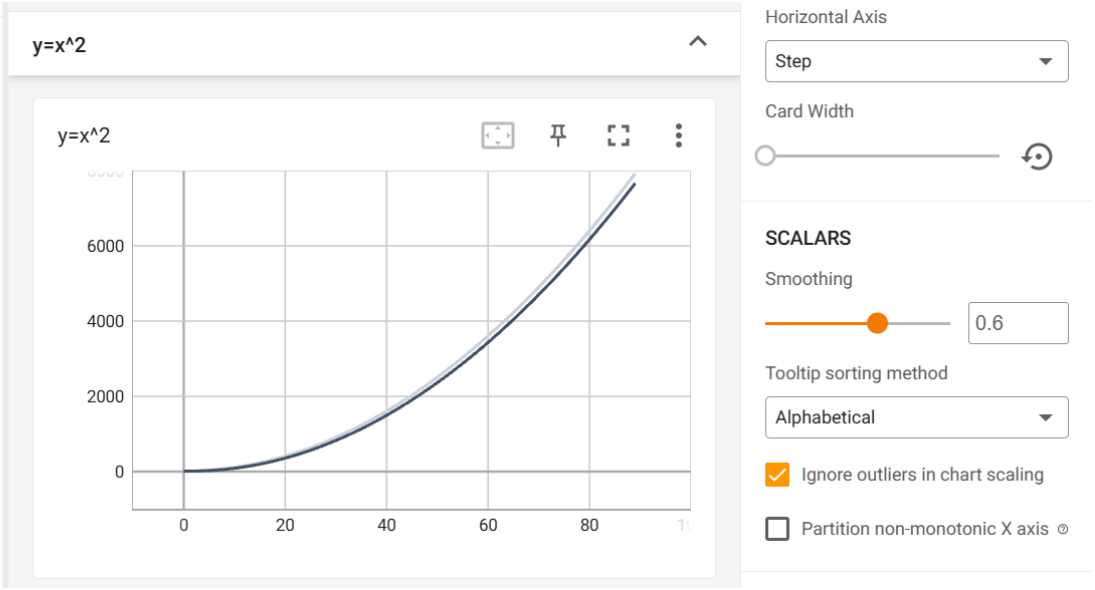
需要注意的是，使用 `add_scalar` 时，`tensorboard` 中的表格是按标题区分的，如果两次 `add_scalar` 标题相同，`tensorboard` 会尝试拟合导致错误。此时的解决方法是删除日志文件、kill 掉 `tensorboard` 进程，重新执行程序……

最为常用的是 `add_scalar` 和 `add_image` 方法。

写入 **x-y** 坐标图：

```python
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter("log")
"""
通过 add_scalar 向 tensorboard 中写入 x-y 坐标图
"""
for i in range(100):
    writer.add_scalar("y=x^2", i*i, i)  # 查看文档可知，add_scalar函数的参数分别为：标签、y轴、x轴
```

tensorboard 显示的结果：

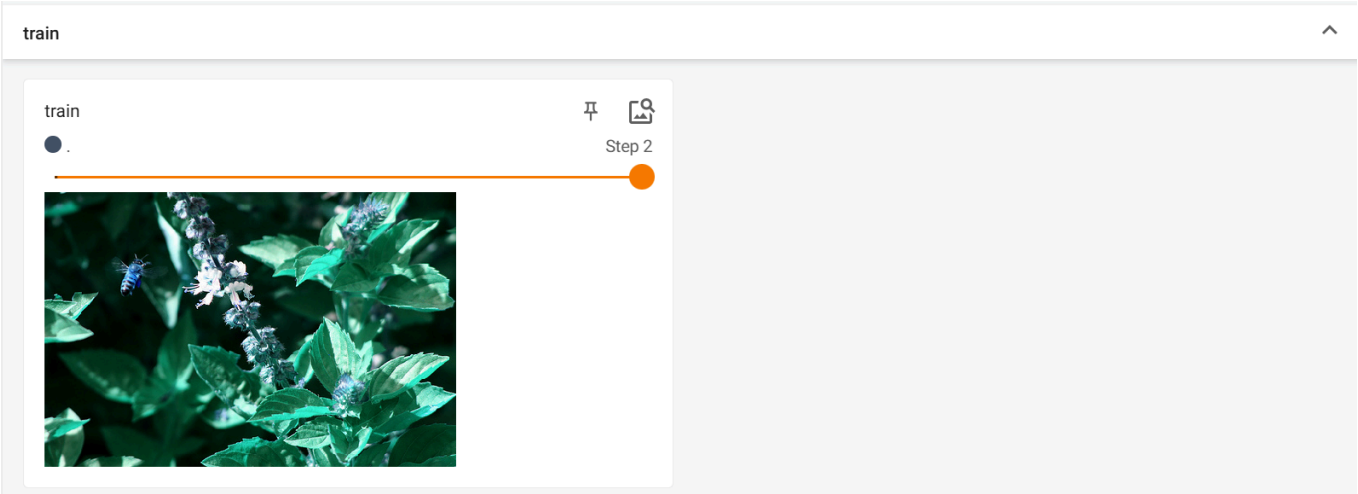


写入图片：

PYTHON

```python
from torch.utils.tensorboard import SummaryWriter
import cv2
from PIL import Image
import numpy as np

writer = SummaryWriter("log")

image_path =
"../dataset/hymenoptera_data/train/ants/20935278_9190345f6b.jpg"
image = Image.open(image_path)
# print(type(image))
image_array = np.array(image)
# print(type(image_array))
# print(image_array.shape)

# 根据说明文档， 输入图像需要是 tensor, numpy, string 或 blobname
writer.add_image('train', image_array, 1, dataformats='HWC')

image_path2 =
"../dataset/hymenoptera_data/train/bees/95238259_98470c5b10.jpg"
image2 = cv2.imread(image_path2)
# print(type(image2))
# print(image2.shape)
writer.add_image('train', image2, 2, dataformats='HWC')
# 记得关闭，不然可能刷新了也加载不出更新
writer.close()
```

显示结果：

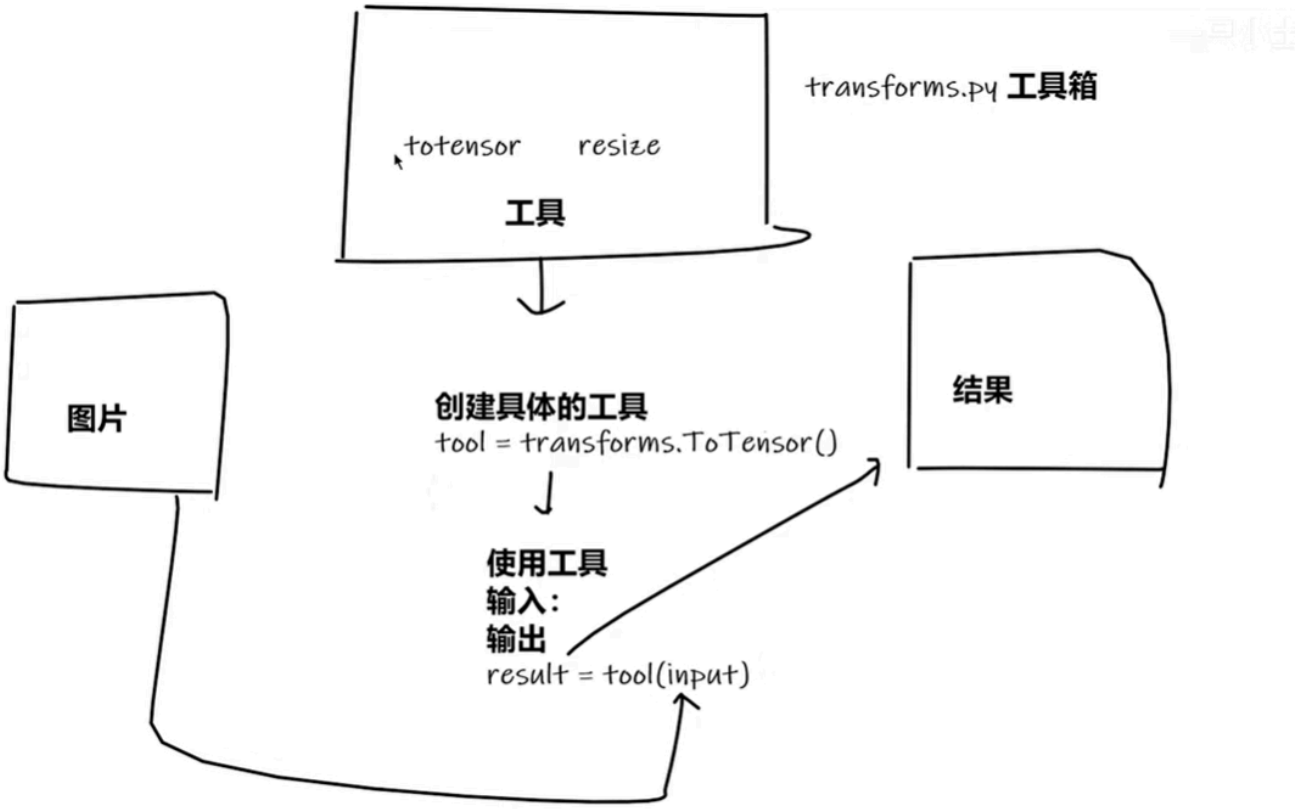


数据处理

Transforms

Tips：不知道方法所需要的参数时，按 Ctrl + P 就会有提示了

`transforms.py` 是一个“工具箱”，其中定义了许多类。在使用时，我们需要先选择具体的工具，即创建其中类的实例，再使用其中的函数（包括直接调用，也就是初始化），给定输入，让它产生对应的输出。 其中的 `ToTensor` 类，接收一个 PILImage/numpy array 类型（大概是这个名字）对象，将其转换为 tensor 类型。 为什么要使用 tensor 类型？tensor 类型包装了训练神经网络所需的各种参数、变量等，而且对 GPU 并行运算做了专门的优化。



示例：

PYTHON

```python
from torchvision import transforms
from PIL import Image

totensor = transforms.ToTensor()

image_path =
"../dataset/hymenoptera_data/train/ants/20935278_9190345f6b.jpg"
image = Image.open(image_path)

image_tensor = totensor(image)
print(type(image_tensor))
print(image_tensor.shape)
```

常见的 Transforms

Tips：PyCharm 忽略大小写匹配提示，在 设置 里搜索 `Code Completion` 除了上面介绍的 `ToTensor` 类，还有类似的 `ToPILImage` 类，作用类似，在此不多介绍。下面接着介绍常用的 transforms 类：

示例准备：

PYTHON

```python
from PIL import Image
from torchvision import transforms
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter("logs")

image_path = "../dataset/test1.webp"
image = Image.open(image_path)

# 创建类对象，trnasforms 只是一个工具包
totensor = transforms.ToTensor()
image_tensor = totensor(image)
writer.add_image("ToTensor", image_tensor, 0)

# ToPILImage
topilimage = transforms.ToPILImage()
image_pil = topilimage(image_tensor)
# image_pil.show()
```

`Normalize`:

PYTHON

```python
# Normalize 正则化数值
# 这里的效果是 [0, 1] -> [-1, 1]
# 计算公式 Given mean: ``(mean[1],...,mean[n])`` and std:
``(std[1],..,std[n])`` for ``n`` channels
# ``output[channel] = (input[channel] - mean[channel]) /
std[channel]``
normalize = transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
image_tensor_normalize = normalize(image_tensor) # ToTensor 自动转换为
CHW 格式
writer.add_image("Normalize", image_tensor_normalize, 0)
```

`Resize`:

```python
# Resize 进行缩放
# 接收参数为 int 或者 tuple；其中 int 缩放短边，并保持原始比例
resize = transforms.Resize((1024, 1024))
image_resize = resize(image)
image_resize = totensor(image_resize)
writer.add_image("Resize", image_resize, 0)
```

`Compose`:

```python
# Compose 进行一系列的 transforms 操作
# Compose 接收的输入为一个列表，其中每个元素都是一个 transforms 中的类对象
compose = transforms.Compose([transforms.Resize(1024), totensor])
image_resize_tensor = compose(image)
writer.add_image("Compose", image_resize_tensor, 0)
```

`RandomCrop`:

```python
# RandomCrop 随机裁下输入图片的指定大小
# 接收 PIL 和 tensor 作为输入 （文档似乎没有提到PIL）
randomcrop = transforms.RandomCrop(512)
image_crop = randomcrop(image)
# print(type(image_crop))
# print(image_crop.size)
for i in range(10):
    image_crop_tensor = randomcrop(image_tensor)
    writer.add_image("RandomCrop", image_crop_tensor, i)
writer.close()
```

Transforms 使用总结

- 多看官方文档
- 关注输入输出类型和维度
- 关注方法参数（主要是必须的，有默认值的很多都是默认即可）
- 不知道返回值的时候，尝试 `print`，`print(type())`，`debug`

- 多尝试

---

1. Python 中的魔术方法，在特定的操作或事件发生时会自动执行。例如，当使用 `[]` 调用时，等价于调用 `__getitem__`。比如 `dataset[0]`，等价于 `dataset.__getitem__(0)`。 ↩