# 环境安装

## Anaconda

### 下载

<https://www.anaconda.com/download>

### 命令行

|  |
| --- |
| 创建环境：conda create -n xxx python=3.10  查看：pip list/conda list  进入环境：conda activate xx  安装torch：在torch的网址生成命令安装  安装jupyter：conda install nb\_conda  启动jupyter：jupyter notebook  查看显卡版本：nvidia-smi |

### 其他

|  |
| --- |
| 查看显卡是否支持CUDA: <https://developer.nvidia.com/zh-cn/cuda-gpus>  查看函数的所有方法：dir(torch) # 注意在python中使用，且先要import  默认环境位置：C:\Users\name\.conda\envs |

## Pytorch

### 网址

<https://pytorch.org/>

### 接口

|  |
| --- |
| 查看是否支持GPU加速：torch.cuda.is\_available() |

# 数据加载

## Dataset

### 重写特殊方法

1. \_\_getitem\_\_

|  |
| --- |
| 允许用户直接用索引访问对象中的数据。  **def** \_\_getitem\_\_(self, item):  item\_path = os.path.join(self.path, self.label\_ls[item])  img = Image.open(item\_path)  **return** img  当对象为obj时，obj[1]则直接调用\_\_getitem\_\_，并把1当入参到getitem，然后返回方法中的返回值。 |

1. \_\_len\_\_

|  |
| --- |
| 允许用户使用python的len函数求对象的长度时。  **def** \_\_len\_\_(self):  **return** len(self.label\_ls)  当求len(obj)时，会直接调用\_\_len\_\_方法，并返回。 |

1. \_\_add\_\_

|  |
| --- |
| 允许用户通过+运算符操作对象。  **def** \_\_add\_\_(self, other: **'Dataset[T\_co]'**) -> **'ConcatDataset[T\_co]'**:  **return** ConcatDataset([self, other])  当用户使用+运算符操作两个对象时，会把后一个Dataset对象当作入参传给\_\_add\_\_方法，默认返回ConcatDataset后的对象。  扩展：  ConcatDataset方法继承Dataset类，调用时入参对象列表，返回\_\_getitem\_\_方法的返回值，其他方法都和Dataset的方法类似，只不过把每个Dataset对象的这些方法返回的结果进行了相加。  （简版，详细看代码接口）  **class** ConcatDataset(Dataset[T\_co]):  @staticmethod  **def** cumsum(sequence):  **return** “所有Dataset对象的长度和”   **def** \_\_init\_\_(self, datasets: Iterable[Dataset]) -> **None**:  ...   **def** \_\_len\_\_(self):  **return** “所有Dataset对象的长度和”   **def** \_\_getitem\_\_(self, idx):  **return** “二分法判断idx属于哪个对象，再取对象的索引” |

### 演示

|  |
| --- |
| 如下demo：  *#!/usr/bin/env python # -\*- coding: utf-8 -\*- # @Time : 2023/6/3 19:00 # @Author : 刘双喜 # @File : 32.dataset.py # @Description : Dataset类演示* **from** torch.utils.data **import** Dataset, ConcatDataset **import** os **from** PIL **import** Image   **class** GetImgData(Dataset):  **def** \_\_init\_\_(self, src\_dir, label\_dir):  self.src\_dir, self.label\_dir = src\_dir, label\_dir  self.path = os.path.join(src\_dir, label\_dir)  self.label\_ls = os.listdir(self.path)   **def** \_\_getitem\_\_(self, item):  item\_path = os.path.join(self.path, self.label\_ls[item])  # 打开对应列表索引的名称的图片  img = Image.open(item\_path)  **return** img   **def** \_\_len\_\_(self):  **return** len(self.label\_ls)   **def** \_\_add\_\_(self, other):  **return** ConcatDataset([self, other])   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  src\_dir =**'..\\..\\dataset\\32.Dataset\\hymenoptera\_data\\train'** ants = GetImgData(src\_dir, **'ants'**)  bees = GetImgData(src\_dir, **'bees'**)  all = ants + bees  all[1].show() |

# 数据处理

## tensorboard

### 数据显示

|  |
| --- |
| TensorBoard 是一个可视化工具，它可以帮助你更好地理解、调试和优化机器学习模型。它提供了丰富的可视化功能，包括标量图表、图像展示、音频播放、计算图可视化、分布图表等。  要使用 TensorBoard，你需要将数据写入到事件文件中。你可以使用 TensorFlow、PyTorch 或其他机器学习框架提供的工具来完成这个任务。例如，在 PyTorch 中，你可以使用 SummaryWriter 类来将数据写入到事件文件中。  当你准备好了事件文件后，你可以在命令行中使用 tensorboard 命令来启动 TensorBoard 服务器。例如，如果你的事件文件位于 logs 目录下，那么你可以使用以下命令来启动 TensorBoard：  tensorboard --logdir=logs –port=6006 |

### scalar

|  |
| --- |
| 数值类型的数据写入事件文件并展示示例：  **from** torch.utils.tensorboard **import** SummaryWriter **import** os   **with** SummaryWriter(**'..\\data\\33.tensorborad'**) **as** write:  **for** i **in** range(100):  *# 每次给图表取名不同防止多个线条拟合在一起* write.add\_scalar(**f"chart{**len(os.listdir(**'../data/33.tensorborad'**))**} :y->x"**, i\*i, i)  # 查看图表，注意要有tensorboard环境，且logdir指定正确  tensorboard --logdir=. --port=6012 |

### image

|  |
| --- |
| 图片类型的数据写入事件文件：  **import** os **from** PIL **import** Image **import** numpy **as** np **from** torch.utils.tensorboard **import** SummaryWriter   dir\_img = **'..\\..\\dataset\\34.tensorboard\_img\_test\\train\\ants\_image' with** SummaryWriter(**'..\\data\\34.tensorborad\_img'**) **as** write:  **for** idx, img\_name **in** enumerate(os.listdir(dir\_img)):  img = Image.open(os.path.join(dir\_img,img\_name))  **if** img **is not None**:  img\_arr = np.array(img)  write.add\_image(**'ants'**, img\_arr, idx, dataformats=**'HWC'**)  # 查看图表，注意要有tensorboard环境，且logdir指定正确  tensorboard --logdir=. --port=6012 |

## transforms

### 图像预处理

|  |
| --- |
| transforms是PyTorch中的一个模块，它提供了一些常用的图像预处理方法。这些方法可以在将图像输入神经网络之前对其进行预处理，例如缩放、裁剪、翻转、旋转等。transforms模块中的每个方法都可以看作是一个函数，它接收一个图像作为输入，然后对其进行处理，最后返回处理后的图像。  可以使用transforms.Compose函数将多个预处理方法组合在一起，形成一个预处理流水线。 |

### 演示

|  |
| --- |
| 读取图片，并进行预处理，然后写入事件文件。  **import** os **from** torch.utils.tensorboard **import** SummaryWriter **from** torchvision **import** transforms **import** cv2 **from** PIL **import** Image *# 预处理-转换为张量 # preprocess = transforms.ToTensor() # 定义预处理过程集合* preprocess = transforms.Compose([  transforms.Resize(256), *# 缩放到256像素* transforms.CenterCrop(224), *# 在中心裁剪出224\*224像素* transforms.ToTensor() *# 转换为张量* ]) dir\_img = **'..\\..\\dataset\\34.tensorboard\_img\_test\\train\\ants\_image' with** SummaryWriter(**'..\\data\\34.tensorborad\_img'**) **as** write:  **for** idx, img\_name **in** enumerate(os.listdir(dir\_img)):  *# 读取图片  # img = cv2.imread(os.path.join(dir\_img, img\_name)) # 不能读git, 返回数组类型,不能用于tansforms* img = Image.open(os.path.join(dir\_img, img\_name))  **if** img **is not None**:  *# 预处理* tensor\_img = preprocess(img)  *# 写入事件文件* write.add\_image(**'ants'**, tensor\_img, idx, dataformats=**'CHW'**) |

### 功能集

#### transforms.ToTensor()

转为张量类型

#### transforms.Resize(256)

转换像素大小为256\*256

#### transforms.CenterCrop(224)

裁剪中心区域224像素

#### transforms.Normalize([1,1,1], [1,1,1])

归一化，两个入参为平均值和标准差，3个信道则入参3个值