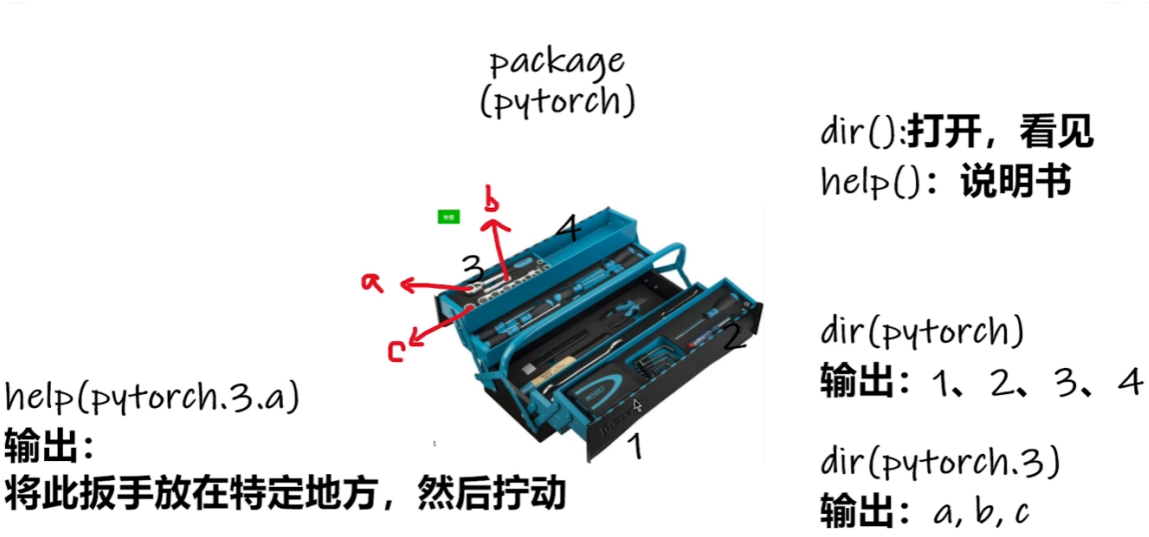
PYTORCH

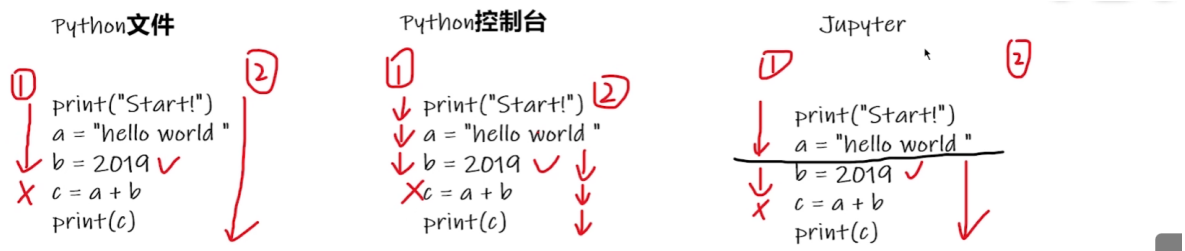
`dir()` 函数与 `help()` 函数

`dir()`：能让我们知道工具箱以及工具箱中的分隔区有什么东西

`help()`：能让我们知道每个工具是如何使用的，工具的使用方法



Pycharm与Jupyter对比



代码是以块为一个整体运行的话：

Python**文件的块是所有行的代码**

优：通用，传播方便，适用于大型项目
缺：需要从头运行

以任意行为块，运行的

优：显示每个变量属性
缺点：不利于代码阅读及修改

以任意行为块运行的

优：利于代码阅读及修改
缺：环境需要配置

pytorch加载数据

`Dataset`

- 数据集，提供一种方式去获取数据及其label
  - 如何获取每一个数据及其label
  - 告诉我们总共有多少个数据

`Dataloader`

- 数据装载器，为后面的网络提供不同的数据形式

Tensorboard使用

👉 **TensorBoard = 深度学习的“可视化仪表盘”**

看训练曲线

- 损失 (Loss) 随迭代次数的变化
- 准确率 (Accuracy) 的变化
- 方便发现过拟合/欠拟合问题

看图像结果

- 比如卷积神经网络中间层的特征图
- 输入图片、预测结果 vs 标签对比

看模型结构

- 直接展示网络的计算图，方便调试

看参数分布

- 用直方图展示模型权重、梯度的分布情况
- 检查训练是否稳定

```
from torch.utils.tensorboard import SummaryWriter

# 创建日志写入器，日志会存放在 runs/experiment1 目录下
writer = SummaryWriter("runs/experiment1")

# 假设你在训练循环中：
for epoch in range(10):
    train_loss = 0.01 * (10 - epoch)   # 假设的损失
    acc = 0.1 * epoch                  # 假设的准确率

    # 写入标量数据
    writer.add_scalar("Loss/train", train_loss, epoch)
    writer.add_scalar("Accuracy/train", acc, epoch)

# 关闭写入器
writer.close()

# 终端运行
tensorboard --logdir=runs
```

`writer.add_scalar`

- **作用**：记录一个标量数值（通常是 loss、accuracy、learning rate 等随训练迭代的变化）。
- **常用格式**：

  ```
  writer.add_scalar(tag, scalar_value, global_step)
  ```

- 参数说明：
  - `tag`：指标名字（显示在 TensorBoard 上的曲线名称）
  - `scalar_value`：要记录的数值

- `global_step`：训练迭代步数（横坐标）
- 示例：

```
for epoch in range(10):
    loss = 0.1 * epoch
    writer.add_scalar("train/loss", loss, epoch)
```

👉 在 TensorBoard 中会看到一条 **loss 曲线**。

`writer.add_image`

- **作用**：记录图片（可以是训练样本、预测结果、特征图等）。
- **常用格式**：

```
writer.add_image(tag, img_tensor, global_step, dataformats='CHW')
```

- 参数说明：
  - `tag`：图像名字
  - `img_tensor`：图像张量，类型是 `torch.Tensor` 或 `numpy.ndarray`
  - `global_step`：训练迭代步数
  - `dataformats`：数据格式
    - `'CHW'` → (通道, 高, 宽) → 常见于 PyTorch 图像
    - `'HWC'` → (高, 宽, 通道) → 常见于 OpenCV/Numpy 图像
- 示例：

```
import torch
img = torch.rand(3, 64, 64)  # 随机生成一张 64x64 彩色图
writer.add_image("random/image", img, 0, dataformats='CHW')
```

- 从PIL到numpy,需要在add_image()中指定shape中每一个数字/维表示的含义

torchvision.transforms

`torchvision.transforms`（简称 **transform**）是 PyTorch 里处理图像数据的工具包，主要用于 **图像预处理和数据增强。**

常见用途

1. **数据预处理**：把图片转为张量、缩放到统一大小、归一化等
2. **数据增强**：在训练时随机旋转、翻转、裁剪等，增加模型鲁棒性

使用方法

一般写成一个 `Compose` 序列，把多个操作串起来：

```
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize((224, 224)),       # 调整图像大小
    transforms.ToTensor(),               # 转为张量 (C,H,W)，并归一化到 [0,1]
    transforms.Normalize(mean=[0.485, 0.456, 0.406],   # 标准化 (ImageNet 常用)
                         std=[0.229, 0.224, 0.225])
])
```

这样 `transform(img)` 会依次执行上面三个操作。

> - `Compose` 是一个 **组合器**，用来把多个 `transform` 串联起来。
> - 它接收一个 **transform 列表**，返回一个新的 transform，调用时会 **依次执行**列表里的所有操作。
>
> ---
>
> ## 基本语法
>
> ```
> from torchvision import transforms
>
> transform = transforms.Compose([
>     transforms.Resize((224, 224)),  # 1. 调整大小
>     transforms.ToTensor(),          # 2. 转换为 Tensor
>     transforms.Normalize(mean=[0.5], std=[0.5])  # 3. 标准化
> ])
> ```
>
> 调用：
>
> ```
> img = Image.open("dog.jpg")     # 原始 PIL 图片
> out = transform(img)            # 按顺序执行三个操作
> ```
>
> 执行顺序：
> PIL 图片 → 缩放 → Tensor → 标准化

常见 Transform 操作

预处理类

- `transforms.Resize(size)`：缩放图像
- `transforms.CenterCrop(size)`：中心裁剪
- `transforms.ToTensor()`：PIL/numpy → Tensor，像素值缩放到 [0,1]
- `transforms.Normalize(mean, std)`：按通道标准化

数据增强类

- `transforms.RandomHorizontalFlip(p=0.5)`：随机水平翻转
- `transforms.RandomVerticalFlip(p=0.5)`：随机竖直翻转
- `transforms.RandomRotation(30)`：随机旋转（±30°）
- `transforms.ColorJitter(brightness, contrast, saturation, hue)`：调整颜色
- `transforms.RandomResizedCrop(size)`：随机裁剪并缩放到目标大小
- `transforms.RandomGrayscale(p=0.1)`：随机转灰度

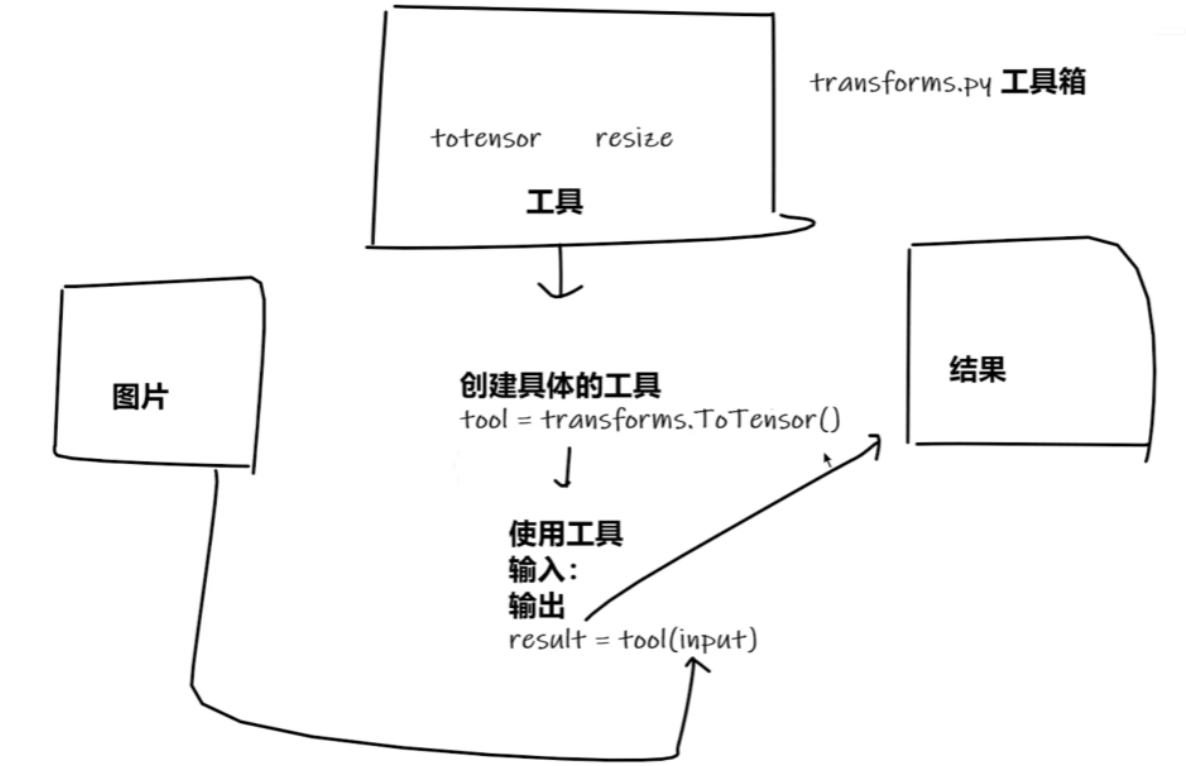
示例：应用到数据集

```
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader

train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),   # 随机翻转
    transforms.RandomRotation(10),       # 随机旋转
    transforms.ToTensor(),               # 转 tensor
])

train_dataset = CIFAR10(root="./data", train=True, transform=train_transform,
download=True)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

图例讲解



torchvision

你提到 **torchvision**，它是 **PyTorch 的视觉工具包**，常用于计算机视觉 (Computer Vision, CV) 任务。简单来说，`torchvision` 提供了四大类功能：

---

1. 数据集 (torchvision.datasets)

- 包含了很多常用的公开数据集，方便直接下载与加载，比如：
  - `MNIST`（手写数字）
  - `CIFAR10 / CIFAR100`（彩色小图像）
  - `ImageNet`（大规模图像分类）
  - `COCO`（目标检测、分割）
- 可以配合 `torch.utils.data.DataLoader` 使用。

---

2. 图像变换 (torchvision.transforms)

- 用于对图像进行预处理和数据增强，例如：
  - `transforms.Resize`：调整图像大小
  - `transforms.ToTensor`：转为 PyTorch 张量
  - `transforms.Normalize`：归一化
  - `transforms.RandomCrop`、`transforms.RandomHorizontalFlip`：数据增强

---

3. 模型 (torchvision.models)

- 提供了很多经典的预训练模型，适合迁移学习或直接使用：
  - 分类模型：ResNet、VGG、DenseNet、AlexNet、MobileNet 等
  - 检测与分割：Faster R-CNN、Mask R-CNN、RetinaNet、FCN、DeepLabV3 等
- 使用方式：

```
import torchvision.models as models
model = models.resnet50(pretrained=True)  # 加载预训练的 ResNet50
```

---

4. 图像工具 (torchvision.io / torchvision.utils)

- `torchvision.io.read_image`：读取图像为张量
- `torchvision.utils.save_image`：保存图像
- `torchvision.utils.make_grid`：把多个图像拼接成网格，方便可视化

DataLoader

DataLoader，这是 PyTorch 里最常用的数据加载工具，属于 `torch.utils.data` 模块。它的作用是：**把数据集（Dataset）打包成批次（batch），并支持多线程并行加载、打乱（shuffle）、按需采样等功能。**

基本用法

```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# 数据预处理
transform = transforms.ToTensor()

# 定义数据集
train_dataset = datasets.MNIST(root="./data", train=True, transform=transform,
download=True)

# 用 DataLoader 封装数据集
train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=64,         # 每批次 64 个样本
    shuffle=True,          # 每个 epoch 打乱数据
    num_workers=2          # 使用两个子进程加载数据（Linux/服务器上常用）
)
```

DataLoader 的主要参数

- **dataset**：必须，传入一个 `torch.utils.data.Dataset` 对象。
- **batch_size**：每个 batch 的样本数量（默认 1）。
- **shuffle**：是否在每个 epoch 开始时打乱数据。
- **num_workers**：加载数据的子进程数（Windows 下一般用 0；Linux 可以设为 CPU 核心数）。
- **drop_last**：如果数据不能被整除，是否丢弃最后一个不足的 batch。
- **pin_memory**：设为 `True` 时会把数据拷贝到 CUDA 的锁页内存中，加快 GPU 训练。

遍历 DataLoader

```
for batch_idx, (data, target) in enumerate(train_loader):
    print(f"批次 {batch_idx}, 数据维度: {data.shape}, 标签维度: {target.shape}")
    # data.shape = [64, 1, 28, 28]  -> 64张 1通道 28x28 的图片
    # target.shape = [64]           -> 64个标签
```

和 Dataset 的关系

- **Dataset**：定义如何获取 **单个样本**（`__getitem__`）。
- **DataLoader**：把样本打包成 **批次**，并负责并行、打乱、采样。

你可以理解为：

```
Dataset —— 管理“单个数据”
DataLoader —— 管理“批量数据 + 数据流”
```