# 完整训练流程

1. 下载数据集，准备 dataset 和 dataloader
2. 下载/定义模型架构
3. 定义损失函数，优化器
4. 定义超参数
5. 开始训练（一个一个 epoch）
    1. 设置模型模式为 train
    2. 进入 dataloader 按 batch 读取数据用于训练的循环
    3. 优化器梯度清零
    4. 模型前向传播，计算损失
    5. 损失后向传播，优化器更新参数
    6. 打印日志，比如损失、准确率等指标
6. 开始验证
    1. 设置模型模式为 eval，torch.no_grad
    2. 进入 dataloader 按 batch 读取数据用于验证的循环
    3. 模型前向传播，计算损失
    4. 打印日志，比如损失、准确率等指标
    5. 根据指标，保存模型

在搭建整个流程中，推荐先搭建基础框架，再逐步添加功能。比如，我在写训练和验证的流程时，按照的顺序是：

开始训练！先搭建基本框架，再逐步添加更多功能

1. 打印 loss。看起来没问题，能 train
2. 计算并打印训练阶段预测准确率
3. 打印 1 个 epoch 的平均损失和准确率
4. 添加验证流程，并支持上述功能 1 2 3
5. 在 tensorboard 绘制 训练 和 验证 曲线
6. 保存模型。做到如果在验证集上，准确率超过最优模型，则保存

Tips：网络训练通常需要第一个维度是 batch_size，如果维度不对/只是用一张图片测试，记得 `reshape`

其实上述流程也包含了模型的验证/测试流程（验证和测试基本相同，只是可能要参数指定是测试才测试，指定训练则是训练+验证）

# 在GPU上训练

有两种方式。不过，都要首先找出代码中所有的模型和输入模型的数据。然后，

1. 对它们使用 `model = model.cuda()` （但是要确保 `torch.cuda.is_available() == true`, 否则报错）
2. 设定 `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` (用 `cuda: 0` 等可以指定多卡时用哪一张卡)，然后对所有这些使用 `model = model.to(device)`

推荐是用 第二种 方法，这样如果要换设备，只要改一处代码。

在大多数情况下，损失函数不需要显式地转移到 GPU，只要模型和数据已经正确地转移到 GPU，损失函数的计算会自动在 GPU 上进行。

# 看开源项目

老生常谈了，先读 README，下载下来先看需要哪些参数。如果有`required=True`的，可以考虑指定一个默认值，方便运行（或者 PyCharm 中编辑运行配置）

# 我的 CIFAR10 模型代码

`model.py`:

PYTHON

```python
from torch import nn
# 模型期望第一个维度是 batch_size，是 1 也行，但是要有
class cifar10(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 32, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 32, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(1024, 64),
            nn.Linear(64, 10)
        )
    def forward(self, x):
        return self.model(x)
```

`train.py`:

PYTHON

```python
import os
import numpy as np
import torch.nn
import torchvision.datasets
from torch.nn import CrossEntropyLoss
from torch.optim.adamw import AdamW
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
from torchvision import transforms
from model import cifar10

# 准备数据集
totensor = transforms.ToTensor()
train_set = torchvision.datasets.CIFAR10(root='../dataset/data',
train=True, transform=totensor, download=True)
test_set = torchvision.datasets.CIFAR10(root='../dataset/data',
train=False, transform=totensor, download=True)

# 定义超参数
learning_rate = 5e-4
weight_decay = 1e-4
num_epoch = 10
batch_size = 64
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# 使用 “cuda: 0" 对我电脑应该效果一样

# 准备 dataloader
train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True, num_workers=0, pin_memory=False, drop_last=False)
test_loader = DataLoader(test_set, batch_size=batch_size,
shuffle=True, num_workers=0, pin_memory=False, drop_last=False)

# 定义模型架构
model = cifar10()
model = model.to(device)

# 检验模型结构是否正确
# for (imgs, targets) in train_loader:
```

```
#     print(model(imgs).shape)  # 输出 [64, 10]，说明正确

# 定义损失函数
loss_fn = CrossEntropyLoss()
loss_fn = loss_fn.to(device)

# 定义优化器
optimizer = AdamW(model.parameters(), lr=learning_rate,
weight_decay=weight_decay)

# 定义 SummaryWriterwriter = SummaryWriter('logs')
total_train_step = 0
total_val_step = 0

# 用于计算何时保存模型（参数）
best_acc = 0.0

# 开始训练！先搭建基本框架，再逐步添加更多功能
# 1. 打印 loss。看起来没问题，能 train
# 2. 计算并打印训练阶段预测准确率
# 3. 打印 1 个 epoch 的平均损失
# 4. 添加验证集，并支持上述功能 1 2 3
# 5. 在 tensorboard 绘制 训练 和 验证 曲线
# 6. 保存模型，做到如果在验证集上，准确率超过最优模型，则保存
for epoch in range(num_epoch):
    print(f"------------ Epoch {epoch} ------------")
    # 训练部分
    model.train()
    acc_list = []
    loss_list = []
    for i, (imgs, targets) in enumerate(train_loader):
        imgs = imgs.to(device)
        targets = targets.to(device)

        optimizer.zero_grad()
        preds = model(imgs)
        # loss, acc 实际上都是一个 batch 的平均值
        loss = loss_fn(preds, targets)
        loss_list.append(loss.item())
```

```
        loss.backward()
        optimizer.step()

        labels = torch.argmax(preds, dim=1)
        acc = torch.sum(labels == targets) / batch_size
        acc_list.append(acc)
        # 绘制曲线图
        total_train_step += 1
        if total_train_step % 100 == 0:
            writer.add_scalar('train_loss', loss.item(),
total_train_step)
            writer.add_scalar('train_acc', acc, total_train_step)
    print(f"loss = {np.mean(loss_list)}")
    print(f"Acc = {np.mean(acc_list)}")
    print()

    # 验证部分
    model.eval()
    with torch.no_grad():
        acc_list = []
        loss_list = []
        for i, (imgs, targets) in enumerate(test_loader):
            imgs = imgs.to(device)
            targets = targets.to(device)
            preds = model(imgs)
            # 交叉熵损失就是需要 一个概率数组（输入）和一个目标标签数值（目
标)
            # 所以不能直接求预测的 label
            loss = loss_fn(preds, targets)
            loss_list.append(loss.item())
            labels = torch.argmax(preds, dim=1)
            acc = torch.sum(labels == targets) / batch_size
            acc_list.append(acc)
            # 绘制曲线图
            total_val_step += 1
            if total_val_step % 100 == 0:
                writer.add_scalar('val_loss', loss.item(),
total_val_step)
                writer.add_scalar('val_acc', acc, total_val_step)
```

```
        print("验证集")
        print(f"loss = {np.mean(loss_list)}")
        mean_acc = np.mean(acc_list)
        print(f"Acc = {mean_acc}", end=" ")
        # 保存模型
        if mean_acc > best_acc:
            best_acc = mean_acc
            if not os.path.exists('pth'):
                os.mkdir('pth')
            torch.save(model.state_dict(),
f"./pth/model_CIFAR10_epoch{epoch}_acc{mean_acc}.pth")
            print(f"<-- 当前最好模型，保存至
./pth/model_CIFAR10_epoch{epoch}_acc{mean_acc}.pth")
        print()
```