Pytorch学习

Pytorch安装和环境搭建

1、官网下载anaconda3

2、创建虚拟环境（环境名字自己命名）

`conda create -n 虚拟环境名 python=python版本`

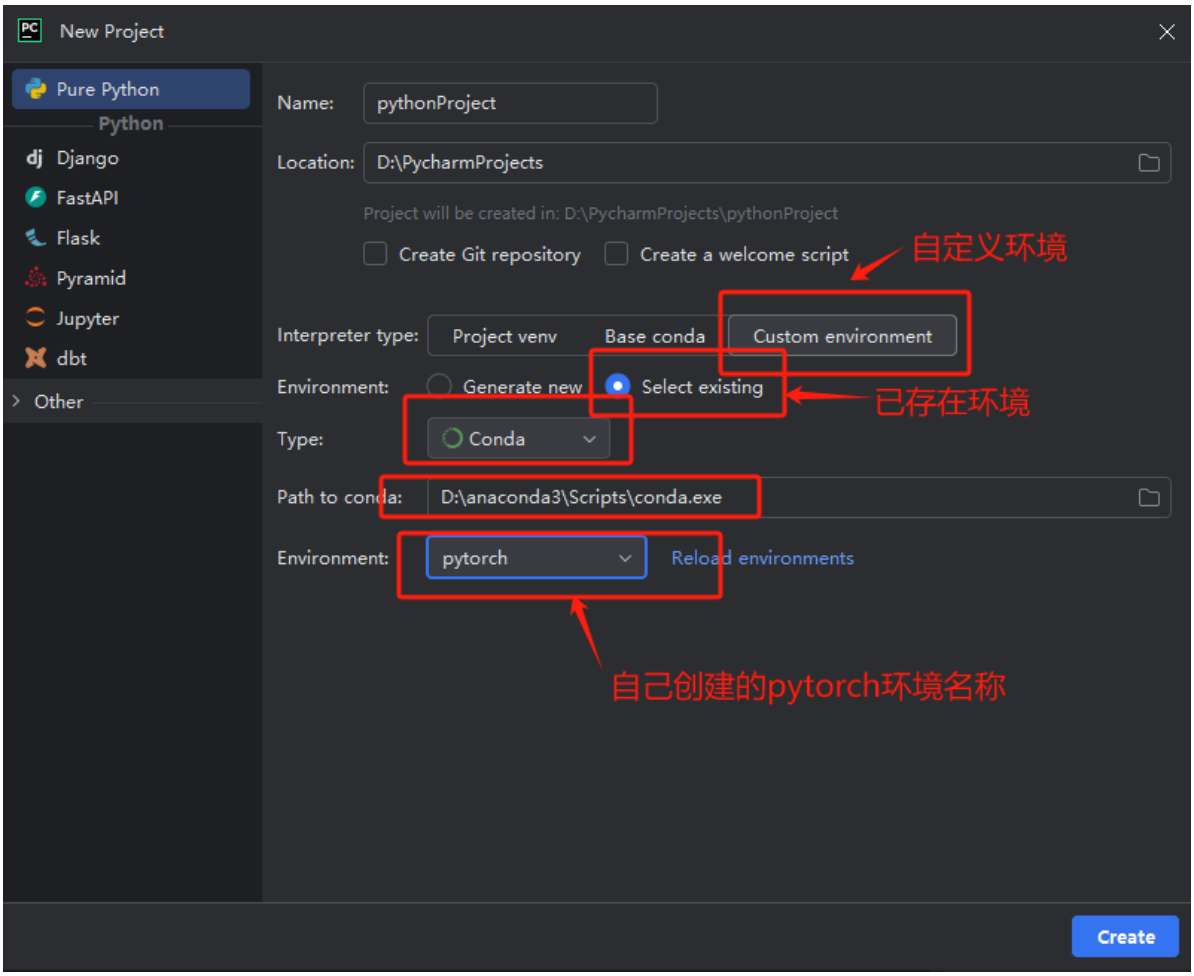
3、激活虚拟环境

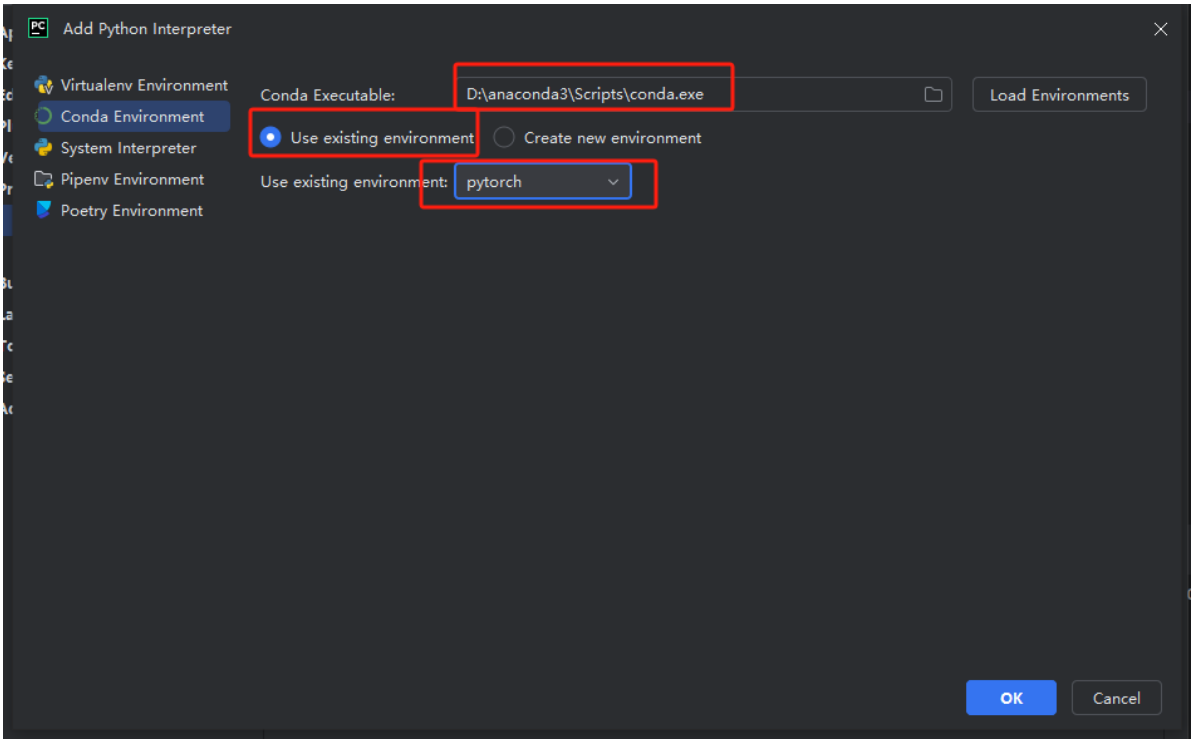
`conda activate 虚拟环境名`

4、安装需要的包环境

`conda install pytorch torchvision torchaudio cpuonly`

5、pycharm中创建pytorch项目





6、测试

创建py文件，输入：

```
import torch

print(torch.__version__)
print(torch.cpu.is_available())
```

控制台输出

```
2.3.0
True
```

环境搭建成功！

安装jupter环境

`conda install nb_conda_kernels`

python学习中的两大法宝

dir()

查看对应模块的包含属性

help()

查看对应函数的说明文档

python文件、python控制台、jupyter使用对比

python文件

- 代码是以块为一个整体运行
- 优：通用、传播方便、适用于大型项目
- 缺：每次运行都需要从头运行

python控制台

- 以任意块为单位运行
- 优：显示每个变量属性
- 缺：不利于代码阅读以及修改

jupyter

- 以任意块为单位运行
- 优：利于代码阅读以及修改
- 缺：环境需要配置

pytorch加载数据

在pytorch中有关加载数据的操作，主要涉及**Dataset**和**Dataloader**，可以看出，后者主要用于加载数据和为网络提供数据的，前者主要告诉后者如何获取数据，Dataset主要提供一种方式去获取数据及其label，其作用有：

- 如何获取每一个数据及其label
- 告诉我们总共有多少的数据

Dataset类

首先需要导入这个类，使用`from torch.utils.data import Dataset`，主要就是从torch这个大工具箱中挑选实用的工具区，从这个工具区中挑选和数据（data）有关的工具

表示从键到数据样本的映射的所有数据集都应对其进行子类化。所有子类都应覆盖：
meth：'**getitem**'，支持获取给定键的数据样本。子类还可以选择覆盖： meth：'**len**'，预计这将通过许多： class：'~torch.utils.data.Sampler' 实现和： class：'~torch.utils.data.DataLoader 的默认选项返回数据集的大小。子类还可以选择实现： meth：'**getitems**'，以加快批量样品加载速度。该方法接受批次样本的索引列表，并返回样本列表。

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


class MyDataset(Dataset):

    def __init__(self, root_dir, label_dir, transform=None):
        # 根路径
        self.root_dir = root_dir
        # label路径
        self.label_dir = label_dir
        self.transform = transform
        # 拼接路径
        self.path = os.path.join(self.root_dir, self.label_dir)
        # 由路径获得路径下的所有文件，返回一个列表，其中列表的每一个属性为文件名
```

```
        self.img_list = os.listdir(self.path)

    def __getitem__(self, index):
        # 由索引值获取文件名
        img_name = self.img_list[index]
        # 根路径+label路径+文件名，得到文件的完整路径
        img_path = os.path.join(self.path, img_name)
        img = Image.open(img_path)
        label = self.label_dir
        return img, label

    def __len__(self):
        return len(self.img_list)
```

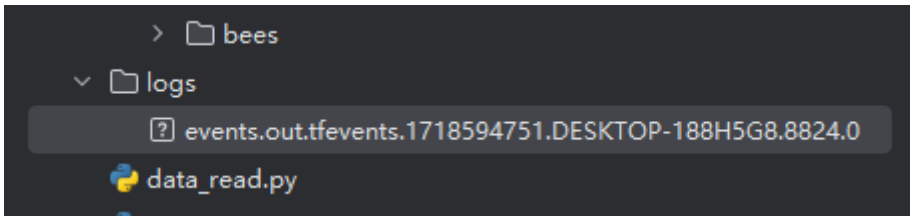
TensorBoard的使用

```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter('logs')

# x = y
for i in range(100):
    writer.add_scalar('x = y', i, i)

writer.close()
```

安装tensorboard包

`pip install tensorboard`

生成logs文件



启动tensorboard

`tensorboard --logdir=[log文件路径] --port=[端口号（可选）]`

当需要重新绘制图像的时候，需要重新运行tensorboard

使用writer.add_image()函数

因为add_image()函数的参数中图片类型必须是torch.Tensor, numpy.ndarray, or string/blobname类型，所以用PIL方式打开文件获得的图片类型不满足要求，所以可以用opencv读取图片，获得numpy类型的图片数据

利用Opencv读取图片，获得numpy型图片数据

安装opencv：

`pip install opencv-python`

将PIL类型转换为numpy类型：

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

writer = SummaryWriter('logs')

img_path = "data/train/bees_image/90179376_abc234e5f4.jpg"
img_PIL = Image.open(img_path)
img_array = np.array(img_PIL)

# 输出img_array的形式格式，默认对于add_image()函数的格式为CHW，C表示通道数，H表示高度，W表示宽度
print(img_array.shape)
'''
    tag (str): Data identifier
    img_tensor (torch.Tensor, numpy.ndarray, or string/blobname): Image data
    global_step (int): Global step value to record 训练步骤
    walltime (float): Optional override default walltime (time.time())
        seconds after epoch of event
    dataformats (str): Image data format specification of the form
        CHW, HWC, HW, WH, etc.
'''
# 修改图片形式格式
writer.add_image("train", img_array, 1, dataformats="WHC")

writer.close()
```

torchvision中的transforms

transforms主要作用是对图片进行一些变换，transforms.py就相当于一个工具箱，其中有很多类和方法实现图片的变换功能

```
from PIL import Image
from torchvision import transforms
import cv2
from torch.utils.tensorboard import SummaryWriter


# python用法 ==> tensor数据类型
# 通过transforms.Totensor去解决两个问题
# 1、transforms在python中该如何使用
# 2、为什么需要Tensor数据类型

img_path = "data/train/ants_image/0013035.jpg"
img = Image.open(img_path)

# 1、transforms在python中该如何使用，ToTenson的参数有两种类型，一种是PIL类型，一种是numpy类型，其中numpy类型可以使用opencv进行获得
# cv2_img = cv2.imread(img_path) # 使用opencv获取到一个numpy.ndarray数据类型
tensor_trans = transforms.ToTensor()
tensor_img = tensor_trans(img)

# 2、为什么需要Tensor数据类型
# tensor数据类型可以理解为包装了神经网络需要的各种参数
write = SummaryWriter("logs")
write.add_image("Tensor_img", tensor_img)
```

```
print(tensor_img)
print(tensor_img.shape)
write.close()
```

常见的Transforms

输入	PIL	Image.open()
输出	tensor	Totensor()
作用	narrays	cv.imread()

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

img = Image.open("images/1.png")

# ToTensor的使用
trans_img = transforms.ToTensor()(img)
writer = SummaryWriter("logs")
writer.add_image("ToTensor", trans_img, 1)

# Normalize
print(trans_img[0][0][0])
# 计算公式：output[channel] = (input[channel] - mean[channel]) / std[channel]，如果mean和channel都是0.5那么等价于(input - 0.5)/0.5 = 2*input - 1
trans_norm = transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
img_norm = trans_norm(trans_img)
print(img_norm[0][0][0])
writer.add_image("Normalized", img_norm, 0)

writer.close()
```

Resize类

```
# Resize
print(img.size)
trans_resize = transforms.Resize((512, 512))
img_resize = trans_resize(img)
# PIL类型
print(img_resize)
# 再转换为tensor类型
img_resize = transforms.ToTensor()(img_resize)
print(img_resize)
writer.add_image("Resize", img_resize, 0)
writer.close()
```

Compose类

Compose()中的参数需要的是一个列表，Python中，列表的形式为[数据1， 数据2， ...]，在Compose中，数据需要时transforms类型，所以得到，Compose([transforms参数1，transforms参数2，...])

```
# Compose - Resize - 2
trans_resize_2 = transforms.Resize(512)
# 前面一个参数是PIL类型，后面一个参数是ToTensor类型，需要对前面一个参数进行匹配
trans_compose = transforms.Compose([trans_resize_2, transforms.ToTensor()])
img_resize_2 = trans_compose(img)
writer.add_image("Resize", img_resize_2, 1)
```

RandomCrop类

```
# RandomCrop:随机裁剪
trans_random = transforms.RandomCrop(512)
trans_compose_2 = transforms.Compose([trans_random, transforms.ToTensor()])
for i in range(10):
    img_random = trans_compose_2(img)
    writer.add_image("Random", img_random, i)
```

torchvision中的数据集使用

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

dataset_transform = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor()
])
# download=True从远程下载到本地
train_set = torchvision.datasets.CIFAR10(root='./dataset', train=True,
transform=dataset_transform, download=True)
test_set = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=dataset_transform, download=True)

# print(test_set[0])
# print(test_set.classes)
# img, target = test_set[0]
# print(img)
# print(target)
# print(test_set.classes[target])
# img.show()

writer = SummaryWriter('dataset_logs')
for i in range(10):
    img, target = train_set[i]
    writer.add_image('dataset_transform', img, i)

writer.close()
```

Dataloader

如果说dataset是说明数据集的位置，每一个数据在数据集中的位置，而dataloader是将数据集进行加载到神经网络当中，每次都是从dataset中取出数据，取出多少数据都是通过Dataloader的参数进行设置。

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

test_data = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor())

# dataset:调用的数据集
# batch_size:每次取出的数据个数
# shuffle:每次取出是否打乱
# num_workers:默认为0
# drop_last:如果总数据个数除以每次取出个数除不尽，是否舍弃余数个数的数据
test_loader = DataLoader(dataset=test_data, batch_size=64, shuffle=True,
num_workers=0, drop_last=False)

# 测试数据集中第一张图片及target
img, target = test_data[0]
print(img.shape)
print(target)

writer = SummaryWriter('dataloader')
step = 0
for data in test_loader:
    # 获取数据集和target
    imgs, targets = data
    # print(imgs.shape)
    # print(targets)
    writer.add_images("test_dataloader", imgs, step)
    step += 1

writer.close()
```

神经网络的基本骨架--nn.Module的使用

```
import torch
from torch import nn


class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()

    # __call__函数会调用forward函数
    def forward(self, input):
        output = input + 1
        return output


youxin = Youxin()
x = torch.tensor(1.0)
output = youxin(x)
print(output)
```

输出：

```
tensor(2.)
```

卷积操作

卷积公式：

$$\operatorname{out}(N_i, C_{\text{out}_j}) = \operatorname{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \operatorname{weight}(C_{\text{out}_j}, k) \star \operatorname{input}(N_i, k)$$

```
# tensor是一种数据结构
import torch
import torch.nn.functional as F

input = torch.tensor([[1, 2, 0, 3, 1],
                      [0, 1, 2, 3, 1],
                      [1, 2, 1, 0, 0],
                      [5, 2, 3, 1, 1],
                      [2, 1, 0, 1, 1]])

kernel = torch.tensor([[1, 2, 1],
                       [0, 1, 0],
                       [2, 1, 0]])

# 因为conv2d函数需要的input包含四个参数，所以需要对定义的tensor进行类型转换
# 参数：数据的数量，数据的层数，数据的长，数据的宽
input = torch.reshape(input, (1, 1, 5, 5))
kernel = torch.reshape(kernel, (1, 1, 3, 3))

print(input.shape)
print(kernel.shape)

# padding：上下左右进行填充，其值为填充多少行，默认值为0，填充后的默认值为0，默认不填充
# dilation：空洞卷积
output = F.conv2d(input, kernel, stride=1, padding=0)
print(output)

output_2 = F.conv2d(input, kernel, stride=2)
print(output_2)

output_3 = F.conv2d(input, kernel, stride=1, padding=1)
print(output_3)
```

神经网络--卷积层

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

```
import torch
import torchvision
from torch import nn
from torch.nn import Conv2d
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

dataloader = DataLoader(dataset, batch_size=64)


class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.conv1 = Conv2d(3, 6, 3, stride=1, padding=0)

    def forward(self, x):
        x = self.conv1(x)
        return x


youxin = Youxin()
# print(youxin)

writer = SummaryWriter("./nn_data")
step = 0
for data in dataloader:
    imgs, targets = data
    output = youxin(imgs)
    # print(imgs.shape)
    # print(output.shape)
    # 第一个参数不知道时可以填-1，reshape会自动计算
    output = torch.reshape(output, (-1, 3, 30, 30))
    # torch.Size([64, 3, 32, 32])
    writer.add_images("input", imgs, step)
    # torch.Size([64, 6, 30, 30])
    writer.add_images("output", output, step)
    step += 1

writer.close()
```

神经网络--池化层

maxpool2d，maxpool也称下采样，maxunpool也称上采样

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

最大池化的作用：保留输入的特征，同时减少数据量，加快训练速度，相当于压缩

```
import torch
import torchvision
from torch import nn
from torch.nn import MaxPool2d
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

dataloader = DataLoader(dataset, batch_size=64)

writer = SummaryWriter('./maxpool_logs')


# 模拟输入图像
# input = torch.tensor([[1, 2, 0, 3, 1],
#                       [0, 1, 2, 3, 1],
#                       [1, 2, 1, 0, 0],
#                       [5, 2, 3, 1, 1],
#                       [2, 1, 0, 1, 1]])
#
# input = torch.reshape(input, (-1, 1, 5, 5))
# print(input.shape)

class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        # ceil_mode：当输入图像的剩余部分小于池化核大小时，是否忽略少于的那部分，依旧找到最大
值
        self.maxpool1 = MaxPool2d(kernel_size=3, ceil_mode=False)

    def forward(self, input):
        output = self.maxpool1(input)
        return output

youxin = Youxin()
# output = youxin(input)
# print(output)
```

```
step = 0
for data in dataloader:
    imgs, targets = data
    output = youxin(imgs)
    # print(imgs.shape)
    # print(output.shape)
    writer.add_images("input_maxpool", imgs, step)
    writer.add_images("output_maxpool", output, step)
    step += 1

writer.close()
```

神经网络--非线性激活

非线性变换

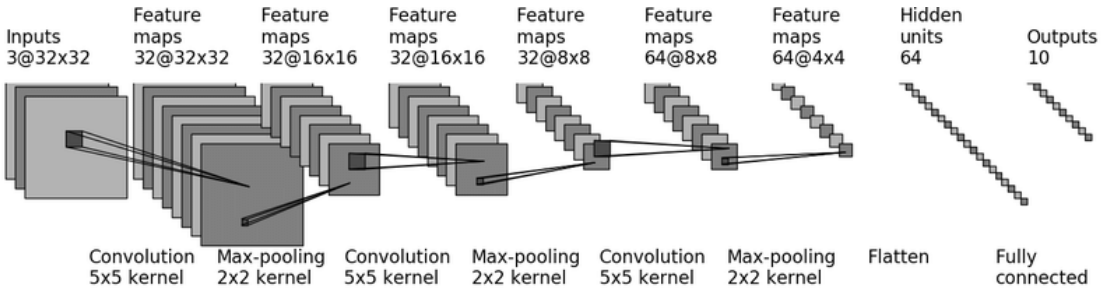
```
import torch
import torchvision.transforms
from torch import nn
from torch.nn import ReLU, Sigmoid
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

# input = torch.tensor([[1, -0.5],
#                       [-1, 3]])

# print(input.shape)

dataloader = DataLoader(dataset, batch_size=64)

class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        # inplace为true时，会将非线性变换后的值直接替换原来的Input,而如果为false时，会将修改后的值进行返回
        # 使用relu线性变换,将负数转为0，正数保持不变
        self.relu = ReLU(inplace=False)
        # 使用sigmoid变换
        self.sigmoid = Sigmoid()

    def forward(self, input):
        output = self.sigmoid(input)
        return output

youxin = Youxin()
# output = youxin(input)
# print(output)

write = SummaryWriter('./non_linear_logs')

step = 0
for data in dataloader:
    imgs, targets = data
    write.add_images("input_non_linear", imgs, step)
```

```
    output = youxin(imgs)
    write.add_images("output_non_linear", output, step)
    step += 1

write.close()
```

神经网络--线性层

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

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

# drop_last：是否丢弃batch_size除不尽时剩余的数据
dataloader = DataLoader(dataset, batch_size=64, drop_last=True)

class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.linear1 = Linear(in_features=196608, out_features=10)

    def forward(self, input):
        output = self.linear1(input)
        return output

youxin = Youxin()

write = SummaryWriter(log_dir='./linear_logs')


step = 0
for data in dataloader:
    imgs, targets = data
    # 将图片转换为一维
    output = torch.reshape(imgs, (1, 1, 1, -1))
    # flatten扁平化处理，直接变成一维，与上一句同理
    # output = torch.flatten(imgs)
    # print(imgs.shape)
    # print(output.shape)
    output = youxin(output)
    # print(output.shape)
    write.add_images("input_linear", imgs, step)
    write.add_images("output_linear", output, step)
    step += 1

write.close()
```

神经网络--Sequential的使用




```
import torch
from torch import nn
from torch.nn import MaxPool2d, Conv2d, Flatten, Linear, Sequential
from torch.utils.tensorboard import SummaryWriter


class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        # self.conv1 = Conv2d(3, 32, 5, padding=2)
        # self.maxpool1 = MaxPool2d(2)
        # self.conv2 = Conv2d(32, 32, 5, padding=2)
        # self.maxpool2 = MaxPool2d(2)
        # self.conv3 = Conv2d(32, 64, 5, padding=2)
        # self.maxpool3 = MaxPool2d(2)
        # # 将数据展平64*4*4
        # self.flatten = Flatten()
        # self.linear1 = Linear(64 * 4 * 4, 64)
        # self.linear2 = Linear(64, 10)

        # 使用sequential会使代码更加简洁
        self.model = Sequential(
            Conv2d(3, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 64, 5, padding=2),
            MaxPool2d(2),
            Flatten(),
            Linear(64 * 4 * 4, 64),
            Linear(64, 10)

        )

    def forward(self, x):
        # x = self.conv1(x)
        # x = self.maxpool1(x)
        # x = self.conv2(x)
        # x = self.maxpool2(x)
        # x = self.conv3(x)
        # x = self.maxpool3(x)
        # x = self.flatten(x)
        # x = self.linear1(x)
        # x = self.linear2(x)
        x = self.model(x)
        return x
```

```
youxin = Youxin()
# print(youxin)
input = torch.ones(64, 3, 32, 32)
output = youxin(input)
# print(output)
# print(output.shape)

write = SummaryWriter(log_dir="./seq_logs")
write.add_graph(youxin, input)
write.close()
```

神经网络--损失函数和反向传播

```
import torch
from torch.nn import L1Loss
from torch import nn

inputs = torch.tensor([1, 2, 3], dtype=torch.float32)
targets = torch.tensor([1, 2, 5], dtype=torch.float32)

# inputs = torch.reshape(inputs, (1, 1, 1, 3))
# targets = torch.reshape(targets, (1, 1, 1, 3))

# 默认取平均损失mean, 可以修改为sum
loss = L1Loss(reduction='sum')
result = loss(inputs, targets)

# 取方差
mesLoss = nn.MSELoss()
result1 = mesLoss(inputs, targets)

# 交叉商
x = torch.tensor([0.1, 0.2, 0.3])
y = torch.tensor([1])
x = torch.reshape(x, (1, 3))
loss_cross = nn.CrossEntropyLoss()
result2 = loss_cross(x, y)

print(result)
print(result1)
print(result2)
```

```
import torch
from torch import nn
from torch.nn import MaxPool2d, Conv2d, Flatten, Linear, Sequential
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import torchvision

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

dataloader = DataLoader(dataset, batch_size=64)

class Youxin(nn.Module):
```

```
    def __init__(self):
        super(Youxin, self).__init__()
        # 使用sequential会使代码更加简洁
        self.model = Sequential(
            Conv2d(3, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 64, 5, padding=2),
            MaxPool2d(2),
            Flatten(),
            Linear(64 * 4 * 4, 64),
            Linear(64, 10)

        )

    def forward(self, x):
        x = self.model(x)
        return x

loss = nn.CrossEntropyLoss()
youxin = Youxin()
for data in dataloader:
    imgs, targets = data
    outputs = youxin(imgs)
    print(targets)
    print(outputs)
    loss_result = loss(outputs, targets)
    # 反向传播，计算梯度
    loss_result.backward()
    print(loss_result)
```

神经网络--优化器

```
import torch
from torch import nn
from torch.nn import MaxPool2d, Conv2d, Flatten, Linear, Sequential
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import torchvision

dataset = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

dataloader = DataLoader(dataset, batch_size=64)

class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        # 使用sequential会使代码更加简洁
        self.model = Sequential(
            Conv2d(3, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 32, 5, padding=2),
            MaxPool2d(2),
            Conv2d(32, 64, 5, padding=2),
            MaxPool2d(2),
```

```
            Flatten(),
            Linear(64 * 4 * 4, 64),
            Linear(64, 10)

        )

    def forward(self, x):
        x = self.model(x)
        return x

loss = nn.CrossEntropyLoss()
youxin = Youxin()
# 定义优化器
optim = torch.optim.SGD(youxin.parameters(), lr=0.01)
# 模拟训练20次，真实情况下训练次数成百上千
for epoch in range(20):
    # 记录整体误差
    running_loss = 0.0
    for data in dataloader:
        imgs, targets = data
        outputs = youxin(imgs)
        # print(targets)
        # print(outputs)
        loss_result = loss(outputs, targets)
        # 每一次梯度都需要调为0
        optim.zero_grad()
        # 反向传播，计算梯度
        loss_result.backward()
        # 调用优化器进行调优
        optim.step()
        # print(loss_result)
        # 计算整体误差
        running_loss += loss_result
    print(f"第{epoch}轮，整体误差为{running_loss}")
```

神经网络--现有网络模型的使用及修改

```
import torchvision.datasets
from torch import nn
import os

from torchvision.models import VGG16_Weights

os.environ['TORCH_HOME'] = 'F:\\torch_model'

# train_data = torchvision.datasets.ImageNet("./data_image_net", split='train',
download=True, transform=torchvision.transforms.ToTensor())

# vgg16_false = torchvision.models.vgg16(pretrained=False)
vgg16_false = torchvision.models.vgg16(weights=None)

# vgg16_true = torchvision.models.vgg16(pretrained=True)
# pretrained参数自0.13版本后就不再支持了
vgg16_true = torchvision.models.vgg16(weights=VGG16_Weights.DEFAULT)

print(vgg16_false)
print(vgg16_true)
```

```
# dataset = torchvision.datasets.CIFAR10(root='./dataset', train=True,
transform=torchvision.transforms.ToTensor(), download=True)
#
# vgg16_true.classifier.add_module("7", nn.Linear(1000, 10))
# print(vgg16_true)
#
# print(vgg16_false)
# vgg16_false.classifier[6] = nn.Linear(4096, 10)
# print(vgg16_false)
```

神经网络--网络模型的保存与读取

保存:

```
import torch
import torchvision
from torch import nn

vgg16 = torchvision.models.vgg16(weights=None)

# 保存方式1，保存的是模型的结构+模型参数
torch.save(vgg16, 'vgg16_method1.pth')

# 保存方式2，保存的是模型参数，官方推荐
torch.save(vgg16.state_dict(), 'vgg16_method2.pth')

# 陷阱
class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=0)

    def forward(self, x):
        x = self.conv1(x)
        return x

youxin = Youxin()
torch.save(youxin, 'youxin_method1.pth')
```

读取:

```
import torch
import torchvision
from torch import nn
# 陷阱解决办法1
from model_save import Youxin

# 方式1，对应保存方式1来加载模型
model = torch.load("vgg16_method1.pth")

# print(model)

# 方式2，加载模型
vgg16 = torchvision.models.vgg16(weights=None)
# 加载参数，以字典形式
```

```
vgg16.load_state_dict(torch.load("vgg16_method2.pth"))
# model2 = torch.load("vgg16_method2.pth")
# print(vgg16)


# 陷阱，在版本较低的环境中，如果加载自定义的网络时会报错
# 解决办法：1、from *** import ***引入
#          2、重新将类的自定义的类写一遍

# 陷阱解决办法2
# class Youxin(nn.Module):
#     def __init__(self):
#         super(Youxin, self).__init__()
#         self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=0)
#
#     def forward(self, x):
#         x = self.conv1(x)
#         return x


model3 = torch.load("youxin_method1.pth")
print(model3)
```

完整的模型训练套路

1. 准备数据集
2. 利用dataloader来加载数据集
3. 创建网络模型
4. 创建损失函数
5. 创建优化器
6. 设置训练中的记录参数
7. 进行训练
8. 输出训练结果和训练参数并分析

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

from model import *

# 准备数据集
train_data = torchvision.datasets.CIFAR10(root='./dataset', train=True,
transform=torchvision.transforms.ToTensor(), download=True)
# 测试集
test_data = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

# 获取数据集大小
train_data_length = len(train_data)
test_data_length = len(test_data)

print(train_data_length)
print(test_data_length)

# 利用dataloader来加载数据集
```

```
train_dataloader = DataLoader(dataset=train_data, batch_size=64)
test_dataloader = DataLoader(dataset=test_data, batch_size=64)

# 创建网络模型
class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=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(64 * 4 * 4, 64),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        x = self.model(x)
        return x
youxin = Youxin()

# 创建损失函数
loss_func = nn.CrossEntropyLoss()

# 优化器
optimizer = torch.optim.SGD(youxin.parameters(), lr=0.001)

# 设置训练网络的一些参数
# 记录训练的次数
total_train_step = 0
# 记录测试的次数
total_test_step = 0
# 训练的轮数
epoch = 10
# 测试loss总和
total_test_loss = 0

# 添加tensorboard
writer = SummaryWriter('./train_logs')

for i in range(epoch):
    print("~~~~~~~~第 {} 轮训练开始~~~~~~~~".format(i + 1))
    # 训练步骤开始
    # youxin.train() # 不一定非要这一行才可以开始训练
    for data in train_dataloader:
        imgs, targets = data
        outputs = youxin(imgs)
        loss = loss_func(outputs, targets)
        # 优化器，参数优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # 记录训练次数
        total_train_step += 1
        if total_train_step % 100 == 0:
```

```
            print(f"训练次数：{total_train_step}， loss: {loss.item():.4f}")
            writer.add_scalar('train_loss', loss.item(), total_train_step)

    # 测试步骤开始
    # youxin.eval() # 不一定非要这一行才能进行测试
    total_accuracy = 0
    with torch.no_grad():
        for data in test_dataloader:
            imgs, targets = data
            outputs = youxin(imgs)
            loss = loss_func(outputs, targets)
            total_test_loss += loss.item()
            # dim = 1：表示横向计算分类
            accuracy = (outputs.argmax(dim=1) == targets).sum()
            total_accuracy += accuracy.item()

    print("~~~~~~~~~第 {} 轮训练后的总测试loss为：{}~~~~~~~~~".format(epoch + 1,
total_test_loss))
    # 输出正确率
    print(f"整体测试集上的正确率：{total_accuracy / test_data_length}")
    writer.add_scalar('test_loss', total_test_loss, total_test_step)
    writer.add_scalar('test_accuracy', total_accuracy / test_data_length,
total_test_step)
    total_test_step += 1
    torch.save(youxin.state_dict(), f'./models/youxin{i}.pth')
    print('模型已保存')

writer.close()
```

利用GPU训练

只有网络模型、数据（输入（imgs），标注（targets））、损失函数包含`.cuda()`方法可以使用gpu进行加速训练

方式1：

```
import torch
import torchvision.transforms
from torch import nn
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import time

# from model import *

# 准备数据集
train_data = torchvision.datasets.CIFAR10(root='./dataset', train=True,
transform=torchvision.transforms.ToTensor(), download=True)
# 测试集
test_data = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

# 获取数据集大小
train_data_length = len(train_data)
test_data_length = len(test_data)
```

```
print(train_data_length)
print(test_data_length)

# 利用dataloader来加载数据集
train_dataloader = DataLoader(dataset=train_data, batch_size=64)
test_dataloader = DataLoader(dataset=test_data, batch_size=64)

# 创建网络模型
class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=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(64 * 4 * 4, 64),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        x = self.model(x)
        return x
youxin = Youxin()
# 使用gpu加速
if torch.cuda.is_available():
    youxin = youxin.cuda()

# 创建损失函数
loss_func = nn.CrossEntropyLoss()
# 使用gpu
if torch.cuda.is_available():
    loss_func = loss_func.cuda()

# 优化器
optimizer = torch.optim.SGD(youxin.parameters(), lr=0.001)

# 设置训练网络的一些参数
# 记录训练的次数
total_train_step = 0
# 记录测试的次数
total_test_step = 0
# 训练的轮数
epoch = 10
# 测试loss总和
total_test_loss = 0

# 添加tensorboard
writer = SummaryWriter('./train_logs')

start_time = time.time()
for i in range(epoch):
    print("~~~~~~~~第 {} 轮训练开始~~~~~~~~".format(i + 1))
    # 训练步骤开始
    # youxin.train() # 不一定非要这一行才可以开始训练
```

```
    for data in train_dataloader:
        imgs, targets = data
        # 利用gpu训练
        if torch.cuda.is_available():
            imgs = imgs.cuda()
            targets = targets.cuda()
        outputs = youxin(imgs)
        loss = loss_func(outputs, targets)
        # 优化器，参数优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # 记录训练次数
        total_train_step += 1
        if total_train_step % 100 == 0:
            end_time = time.time()
            print(f"训练次数：{total_train_step}， loss: {loss.item():.4f}")
            print(f"100次训练后花费的时间：{end_time - start_time}")
            writer.add_scalar('train_loss', loss.item(), total_train_step)

    # 测试步骤开始
    # youxin.eval() # 不一定非要这一行才能进行测试
    total_accuracy = 0
    with torch.no_grad():
        for data in test_dataloader:
            imgs, targets = data
            # 利用gpu训练
            if torch.cuda.is_available():
                imgs = imgs.cuda()
                targets = targets.cuda()
            outputs = youxin(imgs)
            loss = loss_func(outputs, targets)
            total_test_loss += loss.item()
            # dim = 1：表示横向计算分类
            accuracy = (outputs.argmax(dim=1) == targets).sum()
            total_accuracy += accuracy.item()

    print("~~~~~~~~~第 {} 轮训练后的总测试loss为：{}~~~~~~~~~".format(epoch + 1,
total_test_loss))
    # 输出正确率
    print(f"整体测试集上的正确率：{total_accuracy / test_data_length}")
    writer.add_scalar('test_loss', total_test_loss, total_test_step)
    writer.add_scalar('test_accuracy', total_accuracy / test_data_length,
total_test_step)
    total_test_step += 1
    torch.save(youxin.state_dict(), f'./models/youxin{i}.pth')
    print('模型已保存')

writer.close()
```

使用google colab可以免费使用远程gpu进行训练

```
Files already downloaded and verified
Files already downloaded and verified
50000
10000
~~~~~~~~第 1 轮训练开始~~~~~~~~
训练次数：100， loss：2.3018
100次训练后花费的时间：2.825819253921509
训练次数：200， loss：2.3026
100次训练后花费的时间：4.10353422164917
训练次数：300， loss：2.3094
100次训练后花费的时间：5.15277099609375
训练次数：400， loss：2.2944
100次训练后花费的时间：6.224451541900635
训练次数：500， loss：2.2982
100次训练后花费的时间：7.2759857177734375
训练次数：600， loss：2.2935
100次训练后花费的时间：8.31614637374878
训练次数：700， loss：2.2823
100次训练后花费的时间：9.356754779815674
~~~~~~~~第 11 轮训练后的总测试loss为：360.3375794887543~~~~~~~~
整体测试集上的正确率：0.1099
```

方式2

```python
import torchvision.transforms
from torch import nn
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import time

# 定义训练的设备
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 准备数据集
train_data = torchvision.datasets.CIFAR10(root='./dataset', train=True,
transform=torchvision.transforms.ToTensor(), download=True)
# 测试集
test_data = torchvision.datasets.CIFAR10(root='./dataset', train=False,
transform=torchvision.transforms.ToTensor(), download=True)

# 获取数据集大小
train_data_length = len(train_data)
test_data_length = len(test_data)

print(train_data_length)
print(test_data_length)

# 利用dataloader来加载数据集
train_dataloader = DataLoader(dataset=train_data, batch_size=64)
test_dataloader = DataLoader(dataset=test_data, batch_size=64)

# 创建网络模型
class Youxin(nn.Module):
    def __init__(self):
        super(Youxin, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=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(64 * 4 * 4, 64),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        x = self.model(x)
        return x
youxin = Youxin()
# 利用gpu训练
# 网络模型和损失函数可以不用从新赋值，而数据则必须从新赋值
youxin = youxin.to(device)

# 创建损失函数
loss_func = nn.CrossEntropyLoss()
# 利用gpu训练
loss_func = loss_func.to(device)

# 优化器
optimizer = torch.optim.SGD(youxin.parameters(), lr=0.001)

# 设置训练网络的一些参数
# 记录训练的次数
total_train_step = 0
# 记录测试的次数
total_test_step = 0
# 训练的轮数
epoch = 10
# 测试loss总和
total_test_loss = 0

# 添加tensorboard
writer = SummaryWriter('./train_logs')

start_time = time.time()
for i in range(epoch):
    print("~~~~~~~~第 {} 轮训练开始~~~~~~~~".format(i + 1))
    # 训练步骤开始
    # youxin.train() # 不一定非要这一行才可以开始训练
    for data in train_dataloader:
        imgs, targets = data
        # 利用gpu
        imgs = imgs.to(device)
        targets = targets.to(device)
        outputs = youxin(imgs)
        loss = loss_func(outputs, targets)
        # 优化器，参数优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # 记录训练次数
        total_train_step += 1
        if total_train_step % 100 == 0:
            end_time = time.time()
```

```
            print(f"训练次数：{total_train_step}， loss: {loss.item():.4f}")
            print(f"100次训练后花费的时间：{end_time - start_time}")
            writer.add_scalar('train_loss', loss.item(), total_train_step)

    # 测试步骤开始
    # youxin.eval() # 不一定非要这一行才能进行测试
    total_accuracy = 0
    with torch.no_grad():
        for data in test_dataloader:
            imgs, targets = data
            # 利用gpu
            imgs = imgs.to(device)
            targets = targets.to(device)
            outputs = youxin(imgs)
            loss = loss_func(outputs, targets)
            total_test_loss += loss.item()
            # dim = 1：表示横向计算分类
            accuracy = (outputs.argmax(dim=1) == targets).sum()
            total_accuracy += accuracy.item()

    print("~~~~~~~~~第 {} 轮训练后的总测试loss为：{}~~~~~~~~~".format(epoch + 1,
total_test_loss))
    # 输出正确率
    print(f"整体测试集上的正确率：{total_accuracy / test_data_length}")
    writer.add_scalar('test_loss', total_test_loss, total_test_step)
    writer.add_scalar('test_accuracy', total_accuracy / test_data_length,
total_test_step)
    total_test_step += 1
    torch.save(youxin.state_dict(), f'./models/youxin{i}.pth')
    print('模型已保存')

writer.close()
```

当图片是png格式时(png格式是四个通道，除了RGB三通道外，还有一个透明度通道)，需要将其转换为jpg三通道的格式，所以调用：image = image.convert("RGB")，保留其颜色通道

完整的模型验证套路

```
import torchvision.transforms
from PIL import Image
from model import *

img_path = "./data/dog.png"
image = Image.open(img_path)
print(image)

transform = torchvision.transforms.Compose([torchvision.transforms.Resize((32,
32)),
                                torchvision.transforms.ToTensor()])
image = transform(image)
print(image.shape)
image = torch.reshape(image, (1, 3, 32, 32))

model = Youxin()
model.load_state_dict(torch.load("./models/youxin29.pth"))
print(model)
model.eval()
```

```
with torch.no_grad():
    output = model(image)
print(output)
print(output.argmax(dim=1))
```