pytorch

numpy

可以用array创建ndarray对象，实际上是一个张量

```
1
2  #数组
3  # 可以用array创建一个ndarray对象
4  arr = np.array([1,2,3,4,5])
5  print(arr)#[1,2,3,4,5]
6  #创建ndarray，可以将列表、元组或任何类似数组对象传给array，转换为ndarray
7  arr = np.array((1,2,3,4))
8  print(arr)#[1,2,3,4]
9  #维度
10 arr = np.array([[1,2,3],[2,3,4]])
11 print(arr)#[[1,2,3],[2,3,4]]
12 #ndim查看维度
13 print(arr.ndim)#2
14 #可以用ndmin参数定义维数
15 arr = np.array([1,2,3,4],ndmin=5)
16 print(arr)#[[[[[1 2 3 4]]]]]
17 #访问多维数组
18 arr = np.array([[1,2,3],[2,3,4]])
19 print(arr[0,1])#访问第一维中的第二个元素
20 #负索引可以从后往前找
21 #数组裁剪[start:end:step]
22 # 不传递start，视为0，不传递end，视为该维度内数组的长度，不传递step，视为1
23 #start-end，左闭右开
24 print(arr[1,1:])#[3,4]
25
26
27 #数据类型
28 # i代表整数，u代表无符号整数，f浮点，c复合浮点数，m时间区间，M时间，O对象，S字符串，
   Uunicode字符串，V固定的其他类型的内存块
29 #dtype可以返回数组的数据类型
30 # arr.dtype
31 # 可以创建时使用dtype指定数据类型
32 #4字节整数的数组
33 arr = np.array([1,2,3,4], dtype='i4')
34 # astype()可以复制数组并且修改数据类型
35 newarr = arr.astype('f')
36 print(newarr)#[1., 2., 3., 4.]
37
38
39 #副本和视图
40 #副本是一个新数组，视图只是原始数组的视图
41 # 副本拥有数据，对副本修改不会影响原数组，视图正好相反
42 arr = np.array([1,2,3,4], dtype='i4')
43 x = arr.copy()
44 arr[0] = 3
45 print(arr)#[3,2,3,4]
46 print(x)#[1,2,3,4]
47 #视图,只有视图有base属性
```

```
y = arr.view()
arr[0] = 9
print(arr)#[9,2,3,4]
print(y)#[9,2,3,4]


#数组的形状
#每个维度中元素的数量
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr.shape)3
#重塑数组，重塑的数组数量必须相同
newarr = arr.reshape(9)
print(newarr)
newarr2 = newarr.reshape(3,3)
print(newarr2)
#重塑之后返回的是视图
#可以使用未知的维度
# 传递-1作为值，numpy将自动计算数字
arr = np.array([1,2,3,4,5,6,7,8])
newarr = arr.reshape(2,2,-1)
print(newarr)
print(newarr.shape)
#展平数组指将多维数组转化为1维数组
# 可以用reshape(-1)
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
newarr = arr.reshape(-1)
print(newarr)


#numpy的数组操作

#数组迭代
#使用for循环
# for x in arr
#使用nditer迭代数组可以直接迭代高维数组，不需要多层for循环
for x in np.nditer(arr):
    print(x)
#可以使用op_dtypes参数传递期望的数据类型，以在迭代时更改元素的数据类型
#numpy不会就地更改元素的数据类型，需要一些其他空间来执行操作，此额外空间称为buffer，为了在nditer()中启用它，我们传递参数flags=["buffered"]
arr =np.array([1,2,3])
for x in np.nditer(arr,flags=['buffered'],op_dtypes=['S']):
    print(x)
#ndenumerate()方法可以迭代数组,idx表示索引
for idx, x in np.ndenumerate(arr):
    print(idx,x)

#数组连接
arr1 = np.array([[1,2],[3,4]])
arr2 = np.array([[5,6],[7,8]])
# arr = np.concatenate((arr1,arr2), axis=1)#axis = 1按行连
arr = np.stack((arr1,arr2))#多加一个括号
print(arr,arr.shape)
#hstack，vstack，dstack按不同方式堆叠
#数组分割
# np.array_split(arr,4)#不能均分系统会自动调配
```

```
# np.split(arr,3)#均分，如果不能均分会报错

#数组搜索
#where方法
print(arr)
x = np.where(arr == 4)
print(x)#返回的是位置
#排序搜索
arr = np.array([1,3,5,7])
x = np.searchsorted(arr,3)#从左往右找第一个大于3的[]
print(x)
x = np.searchsorted(arr,3,side='right')#从右往左找[)
print(x)
x = np.searchsorted(arr,[3,5,7])#可以找多个值


#数组过滤
# 可以使用布尔数组来过滤数组
arr = np.array([1,3,5,7])
x = [True,False,True,False]
newarr = arr[x]
print(newarr)
#简单过滤器写法
filter_arr = arr > 3
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)

```

matplotlib

matplotlib用于绘制图像

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
# 大多数matplotlib实用程序位于pyplot子模块下
#在图中从位置(0,0)到位置(6，250)画一条线
xpoints = np.array([0,6])
ypoints = np.array([0,20])
# plot函数用于在图表中绘点
plt.plot(xpoints,ypoints)

#绘制多点
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 9])
plt.plot(xpoints,ypoints)
plt.show()

#默认x点
# 不指定x轴上的点，则默认0，1，2，3.。。。

#标记
# 关键字参数marker，指定标记强调每个点
plt.plot(ypoints, marker = "o")
```

```
plt.show()

#格式化字符串fmt
#语法为：marker|line|color
plt.plot(ypoints, 'o:b')#表示用o标记每个点，用虚线画，线为蓝色
plt.show()

#设置尺寸大小markersize，简写为ms
plt.plot(ypoints, 'o:b', ms = 20)
plt.show()

#标记颜色
# 使用mec标记边缘的颜色
# 使用mfc标记内部的颜色
plt.plot(ypoints, 'o:b', ms = 20, mec = 'g', mfc = 'r')
plt.show()

#线条
# 虚实下线用ls表示
# 颜色用color或c
# 宽度用linewidth或lw
# 可以成对画
# plt.plot(x1, y1, x2, y2)


#标签
#xlabel和ylabel函数为x轴和y轴设置标签,使用title设置标题
#设置字体为楷体
plt.rcParams["font.sans-serif"] = ["KaiTi"]
#使用fontdict参数来设置标题和标签的字体属性
font = {'family':'sans-serif','color':'blue','size':20}
plt.xlabel("卡路里",fontdict=font)
plt.ylabel("超级卡路里",fontdict=font)
plt.title("牛逼",fontdict=font)
plt.plot(xpoints,ypoints)
plt.show()


#网格线
plt.xlabel("卡路里")
plt.ylabel("超级卡路里")
plt.title("牛逼")
plt.plot(xpoints,ypoints)
plt.grid()
plt.show()
#axis指定要显示哪个轴的网格线
# plt.grid(axis='x')


#多图
#plot1
plt.subplot(1,2,1)#一行两列第一张子图
plt.plot(xpoints,ypoints);
#plot2
plt.subplot(1,2,2)#一行两列第二张子图
plt.plot(xpoints,ypoints)
```

```
plt.show()
#title可以为每个子图加标题
#suptitle可以为所有图加总标题


#散点图
plt.scatter(xpoints,ypoints)
plt.show()
#给每个点上色，只能用c而不能用color
#用数组传入
colors = np.array(['red','blue','green','black'])
plt.scatter(xpoints,ypoints,c = colors)
plt.show()
#颜色图，具体看详细文档


#柱状图
#使用bar函数画图
x = np.array(['A','B','C','D'])
y = np.array([3,8,1,10])
plt.bar(x,y)
plt.show()
#使用barh可以画水平柱状图
#width和height可以设置宽度和水平图的高度



#直方图
# 用hist()函数来创建直方图
#用numpy随机生成一个包含250个值的数组，集中在170左右，标准差为10
x = np.random.normal(170,10,250)
plt.hist(x)
plt.show()


#饼图
#pie()绘制饼图
#labels设置标签
x = np.array([21,34,56,72])
label = ["西瓜","苹果","香蕉","桃子"]
#startangle可以设置开始画的角度
#Explode可以让某一块突出
myexplode = [0.1,0,0,0]
plt.pie(x,labels=label,explode=myexplode,shadow=True)
#可以用shadow设置阴影
#用colors设置颜色，传入对应数组
#legend可以设置图例，也可以设置标题
plt.legend(title = '标题')
plt.show()

```

tensorboard

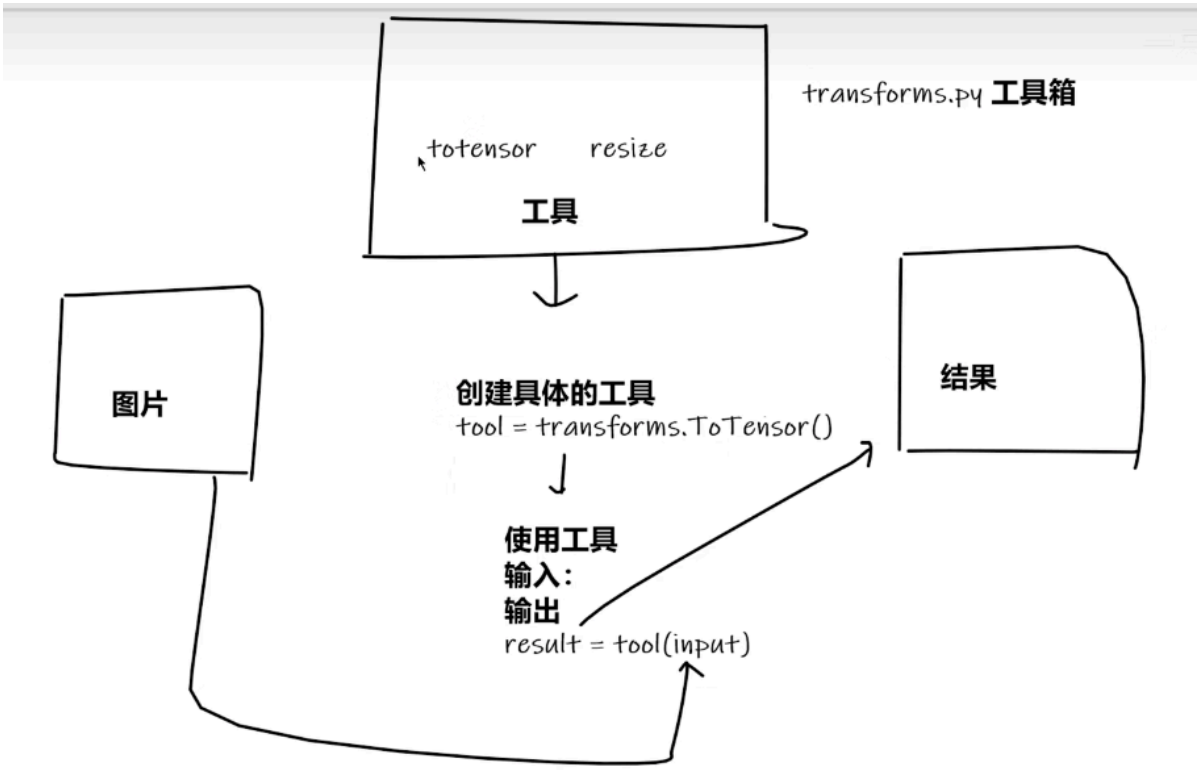
数据可视化工具

1. 可视化模型的网络架构
2. 跟踪模型指标，如损失和准确性等
3. 检查机器学习工作流程中权重、偏差和其他组件的直方图
4. 显示非表格数据，包括图像、文本和音频
5. 将高维嵌入投影到低维空间

```
from torch.utils.tensorboard import SummaryWriter#引入summarywriter
import numpy as np
from PIL import  Image
#设置将事件写到事件文件logs里
writer = SummaryWriter("logs")

image_path = "data/train/ants_image/0013035.jpg"
image_PIL = Image.open(image_path)
image_Array = np.array(image_PIL)

writer.add_image("test",image_Array,dataformats='HWC')
#img读取需要的数据类型为img_tensor (torch.Tensor, numpy.ndarray, or string/blobname): Image data
#正常读出来是<class 'PIL.JpegImagePlugin.JpegImageFile'>类型，不符合要求
#常用的方式是
#1.使用opencv读取numpy类型
# 2.使用numpy的np.array()方法转化为numpy类型
for i in range(100):
    writer.add_scalar("y=2x",2*i,i)

writer.close()
#最后在terminal输入命令，打开tensorboard页面
# tensorboard --logdir=事件文件名 (事件文件名不加引号)--port=端口号 使用该命令可以打开tensorboard
```

transforms

transforms一般用来对图像预处理，比如将图像转化为tensor类型，对图像进行随机裁剪、标准化、归一化、缩放等操作

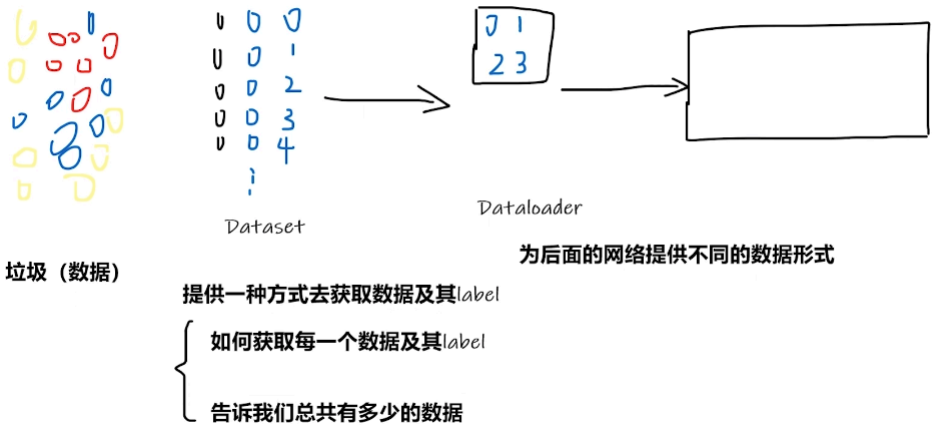



```python
from PIL import Image
from torch.utils.tensorboard import SummaryWriter
from torchvision import transforms

writer = SummaryWriter('logs')
img = Image.open('data/train/ants_image/0013035.jpg')
# totensor使用，将图片转换为tensor类型
trans_totensor = transforms.ToTensor()
img_tensor = trans_totensor(img)
writer.add_image("ToTensor",img_tensor)#往tensorboard中写图片

# 归一化
# 1.将一列数据变化到某个固定区间(范围)中，通常，这个区间是[0, 1] 或者（-1,1）之间的小数。
# 主要是为了数据处理方便提出来的，把数据映射到0～1范围之内处理，更加便捷快速
# 2.把有量纲表达式变成无量纲表达式，便于不同单位或量级的指标能够进行比较和加权。归一化是一种简化计算的方式，
# 即将有量纲的表达式，经过变换，化为无量纲的表达式，成为纯量。
# 图片是rgb三个通道，6个参数表示三个通道的平均值和标准差
# mean均值，std标准差
# output[channel] = (input[channel] - mean[channel]) / std[channel]
print(img_tensor[0][0][0])# tensor(0.3137),这是输入值
trans_norm = transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
img_norm = trans_norm(img_tensor)
print(img_norm[0][0][0]) # tensor(-0.3725)，这是输出值
writer.add_image("Normalize", img_norm)


#等比列缩放Resize
#输入PILimg or Tensor 输出PILimg or Tensor
print(img.size)
trans_resize = transforms.Resize((512,512))
img_resize = trans_resize(img_tensor)
print(img_resize)
```

```
writer.add_image("Resize", img_resize)


#Compose，组合，可以将多种操作以参数形式传入组合在一起
#Compose()中的参数是一个列表，列表形式为[1,2,3]
#Compose需要的数据是transforms类型，所以参数为[transforms1,transforms2...]
#将图片短边缩放至512，长宽比保持不变，如果高度>宽度，则图像将被重新缩放为(size*高度/宽度，size)
trans_resize_2 = transforms.Resize(512)
#使用compose，首先第一个参数就是将图片进行缩放，然后第二个参数将图片转换为tensor类型
trans_compose = transforms.Compose([trans_resize_2, trans_totensor])
img_resize_2 = trans_compose(img)
writer.add_image("Resize", img_resize_2, 1)


#RandomCrop 随机裁剪
trans_random = transforms.RandomCrop(512)
trans_compose_2 = transforms.Compose([trans_random,trans_totensor])
for i in range(10):
    img_crop = trans_compose_2(img)
    writer.add_image('RandomCrop',img_crop,i)


writer.close()
```

dataset和dataloader

dataset准备数据集，定义数据集的内容，dataloader加载数据集




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

# dataset准备测试数据集 root数据集存放位置，train为false表示测试集，否则为训练集，transform选择操作类型
test_Data = torchvision.datasets.CIFAR10(root='./dataset',train=False,transform=torchvision.transforms.ToTensor())
# 设置dataset数据集来源，设置每次从dataset中取的数据数量，设置是否打乱，num_workers=0 一般不会报错，drop_last=True ：DataLoader 中的此设置会删除不完整的最后一批（如果它小于指定的批量大小）。这确保了训练期间处理的每个批次包含相同数量的样本。
```

```
test_loader = DataLoader(dataset=test_Data,batch_size=64,shuffle=False,num_workers=0,drop_last=True)

#target对应图片标签的索引
img, target = test_Data[0]
print(img)
print(target)

writer = SummaryWriter('dataloader')

for epoch in range(2):
    step = 0
    for data in test_loader:
        imgs, targets = data
        # print(imgs.shape)#torch.Size([4, 3, 32, 32]) 4个图片，3个通道，32×32
        # print(targets)#tensor([0, 9, 1, 5]),将4个图片进行打包 4个图片的target
        writer.add_images('Epoch:{}'.format(epoch), imgs, step)
        step += 1

writer.close()
```

读取数据集

读文件过程

```
from torch.utils.data import Dataset
from PIL import Image
import os
class MyData(Dataset):
    def __init__(self,root_dir,label_dir):#用self相当于变成全局变量，就可以在其他函数访问
        self.root_dir = root_dir #root地址，一般到train
        self.label_dir = label_dir #标签名
        self.path = os.path.join(self.root_dir,self.label_dir)#将地址相连，这种方式能避免出错
        self.img_path = os.listdir(self.path)#以列表形式返回图片的名字的合集
        print(self.path)


    def __getitem__(self, idx):
        img_name = self.img_path[idx]#图片名字
        img_item_path = os.path.join(self.root_dir,self.label_dir,img_name)#将地址和标签和图片名相连得出相对地址
        img = Image.open(img_item_path)#打开图片，是PIL格式
        label = self.label_dir
        return img, label#最终返回图片和标签

    def __len__(self):
        return len(self.img_path)

root_dir = "hymenoptera_data/train"
ants_label_dir = "ants"
bees_label_dir = "bees"
```

```
ants_dataset = MyData(root_dir,ants_label_dir)
bees_dataset = MyData(root_dir,bees_label_dir)

train_dataset = ants_dataset + bees_dataset#这种情况可能用于数据集不够的情况，将仿造的数据集和真实的数据集结合
```

卷积

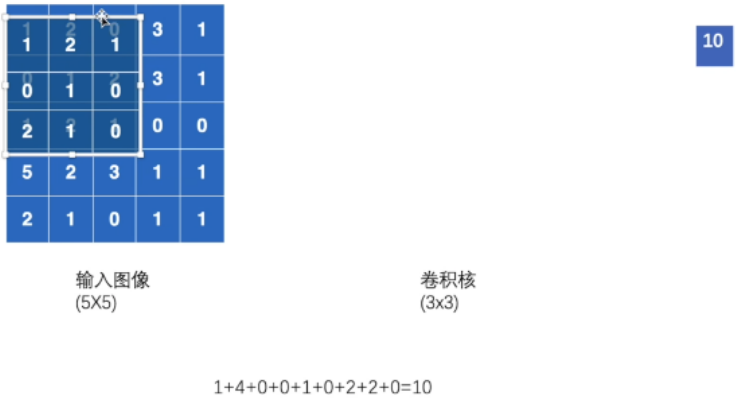
搭建神经网络

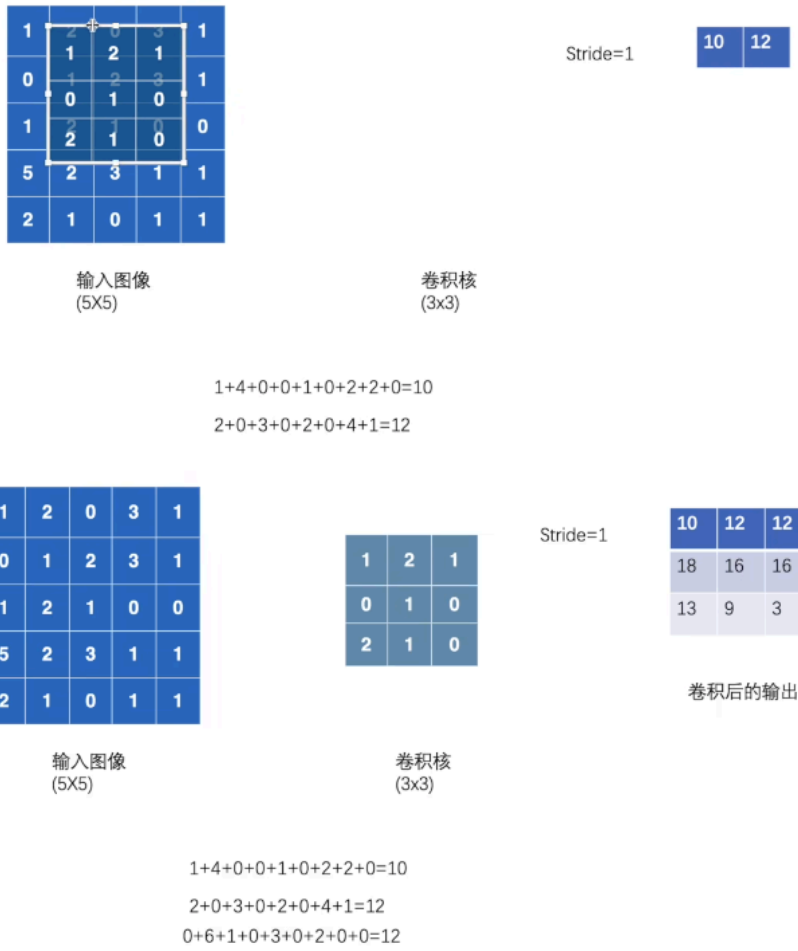
```
from torch import nn
import torch

# 自己搭建一个神经网络，使用nn包
class Net(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

    def forward(self, input):
        output = input + 1
        return output

net = Net()
x = torch.tensor(1.0)
output = net(x)
print(output)
```

卷积层

卷积层从输入数据中提取特征

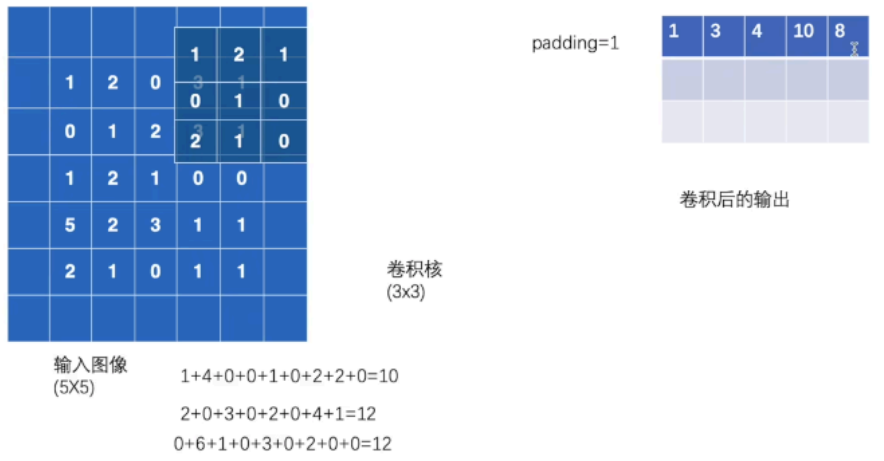
卷积过程：（代码见下文）

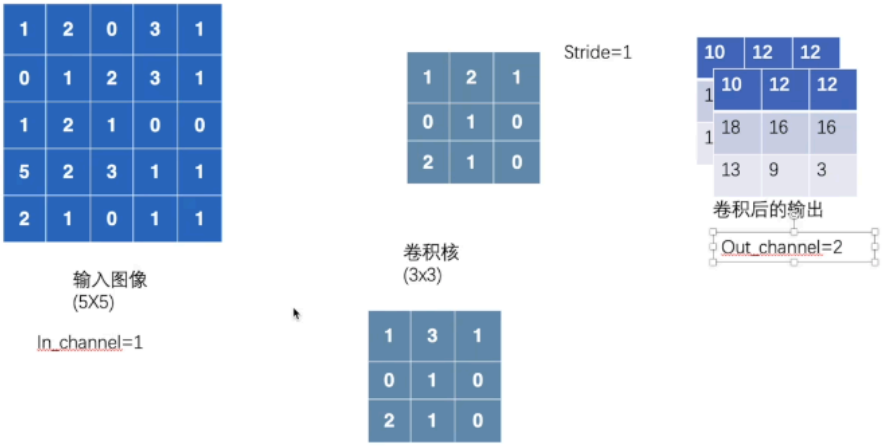




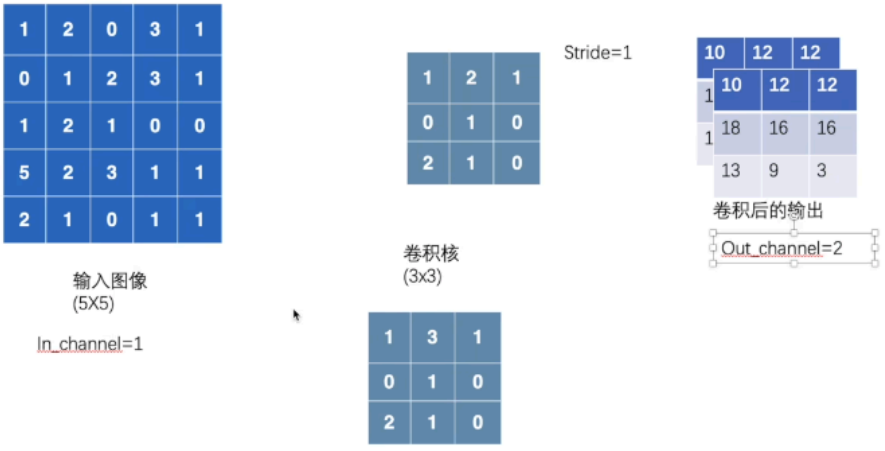

```python
#卷积过程
import torch
import torch.nn.functional as F
input = torch.tensor([[1,2,0,3,1],
                      [0,1,2,3,1],
                      [1,2,1,0,0],
                      [5,2,3,1,1],
                      [2,1,0,1,1]])

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

# conv2d输入需要4个参数，batch_size(每次划分多少数据给神经网络)，通道为1(默认灰度图像)，高度5，宽度5
input = torch.reshape(input, (1,1,5,5))
kernal = torch.reshape(kernal, (1,1,3,3))

output = F.conv2d(input, kernal, stride=1)
print(output)
#tensor([[[[10, 12, 12],
#          [18, 16, 16],
#          [13,  9,  3]]]])
```

padding = 1时





out_channel = 2时，会有两个卷积核对输入图像进行扫描，得到两个输出



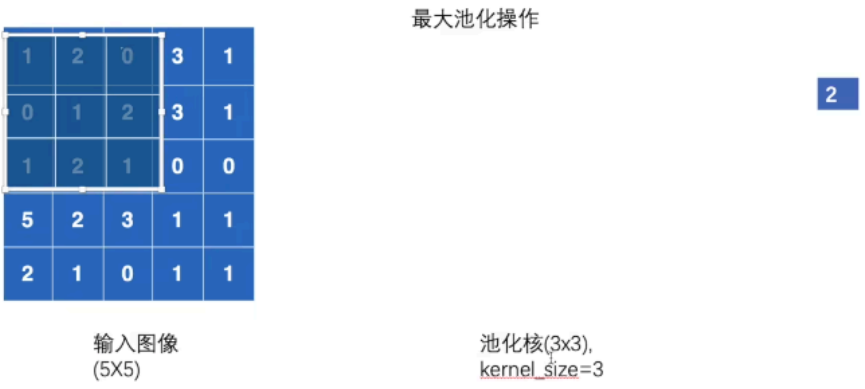
使用卷积操作对数据集进行处理

```
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("./data", train=False,
transform=torchvision.transforms.ToTensor(),download=True)
```

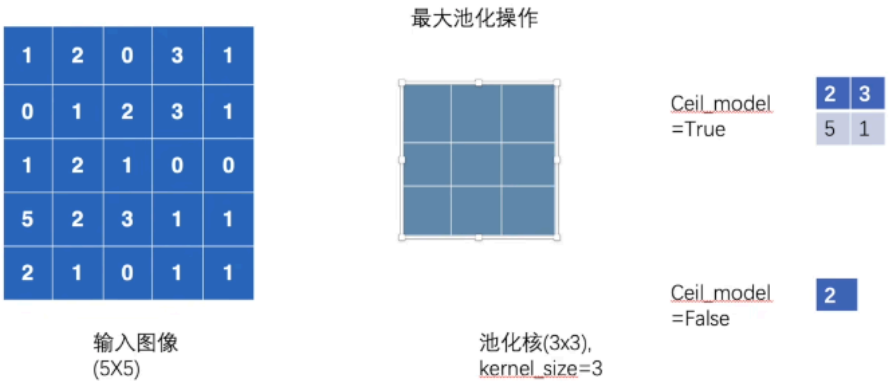
```
9
10 dataloader = DataLoader(dataset,batch_size=64)
11
12 class Net(nn.Module):
13     def __init__(self, *args, **kwargs) -> None:
14         super().__init__(*args, **kwargs)
15         self.conv1 = Conv2d(3,6,3,stride=1,padding=0)
16
17     def forward(self,x):
18         x = self.conv1(x)
19         return x
20
21 net = Net()
22
23 step = 0
24 writer = SummaryWriter('logs')
25 for data in dataloader:
26     imgs,targets = data
27     output = net(imgs)
28     writer.add_images('input',imgs,step)
29     # 直接输出会报错，设置的6个channel，用reshape改为3个通道，-1让它自动设置bach_size
   批次
30     output = torch.reshape(output,(-1,3,30,30))
31     writer.add_images('output',output,step)
32     step += 1
```

池化层

池化层降低特征的数据量，对特征图进行降维处理

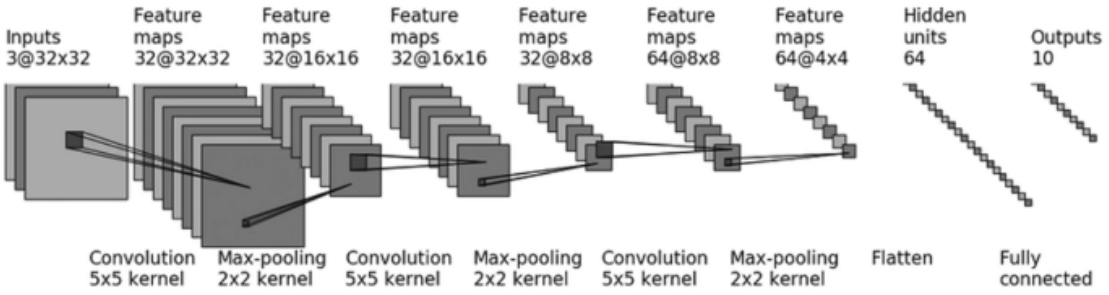


ceil_model为false的话，窗口里不满九个元素不进行处理，默认就是false




```python
import torch
import torchvision.datasets
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('./data',train=False,download=True,
transform=torchvision.transforms.ToTensor())

dataloader = DataLoader(dataset, batch_size=64)


#为了满足maxpool输入的要求，N,C,H,W
# input = torch.reshape(input,(-1,1,5,5))

class Net(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.maxpool1 =
MaxPool2d(kernel_size=3,ceil_mode=True)#ceilmode=true,选择池化窗口不满时的情况

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

net = Net()

step = 0
writer = SummaryWriter('logs_maxpool')
for data in dataloader:
    imgs,targets = data
    writer.add_images('input', imgs, step)
    output = net(imgs)#最大池化不会改变通道数，原来有3维，池化后还是3维
    writer.add_images('output',output,step)
    step += 1
writer.close()
```

CIFAR10网络结构



Shape: 32+2*padding-4-1=27+2*padding=31, 2*padding= 4, padding=2

31

stride=1

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

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

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

output	target
选择（10）	选择（30）
填空（10）	填空（20）
解答（20）	解答（50）

Loss=(30-10)+(20-10)+(50-10)=70

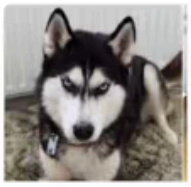
1. 计算实际输出和目标之间的差距
2. 为我们更新输出提供一定的依据（反向传播）

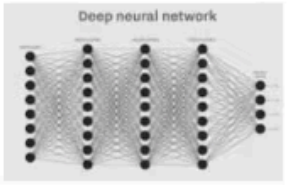
X:1, 2, 3
Y:1, 2, 5
L1loss = (0+0+2) /3=0.6
MSE = (0+0+2^2)/3=4/3=1.333

The loss can be described as:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

Person, dog, cat
0, 1, 2





output
[0.1, 0.2, 0.3]

x

Target
1
class

Loss(x, class) = -0.2+log(exp(0.1)+exp(0.2)+exp(0.3))

当预测正确的时候x[class]会很大，loss会很小

分类问题计算准确率的方式

2 x input

Model(2分类)

Outputs =
[0.1, 0.2]
[0.3, 0.4]

0. 1

Argmax

Preds = [1]
[1]

Inputs target = [0][1]

Preds == inputs target

[false, true].sum() = 1

遇到的问题：

1.pycharm无法激活conda，pycharm使用的终端默认维powershell，换为cmd即可

dataloader的num_workers>0在windows下有可能会报错BrokenPipeError

2.发现代码报错位置在`for data in train_dataloader:`这里，但是书写确实没啥问题

找了好久终于发现在进行数据加载的时候`transform=torchvision.transforms.ToTensor`出现书写错误掉了"()"。

HWC和CHW

HWC指的是高度宽度通道

数据会按照以上顺序进行存储

```
hwc 和 chw 的内存排布区别：
opencv: 原始数据(rgb)排布(hwc)：假如是 width =5, height =3;
rgbrgbrgbrgbrgb
rgbrgbrgbrgbrgb
rgbrgbrgbrgbrgb
目标排布（chw)： 假如是 width =5, height =3;
rrrrr
rrrrr
rrrrr
ggggg
ggggg
ggggg
bbbbb
bbbbb
bbbbb
```