## 图像分类卷积神经网络的搭建及训练

## （基于pytorch框架）

#### 学生：2113662 张丛

#### 指导教师：王恺

#### 课程：python编程基础

# 目录

# 引言

# 相关工作

###### Pytorch是什么

###### Pytorch安装

###### Pycharm安装

# 方法

###### 神经网络结构

# 实验

###### 网络结构的实现

###### 神经网络搭建全过程

###### 数据预处理

###### 损失函数与反向传播

# 全部代码（附注释）

# 结论

# 优化

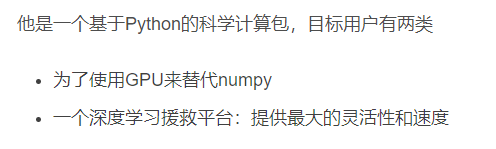
# 大作业心得

# 引言

Pyhthon编程语言已经成为近几年最流行、最火爆的的编程语言，作为计网的一员，我也于2022年正式在王恺老师的课上学习python。在课上我震惊于python语言的简洁、强大，对python语言也产生了浓厚的兴趣。在课程大作业方面，我选择的高难度的神经网络搭建，也是对自己的一项挑战。

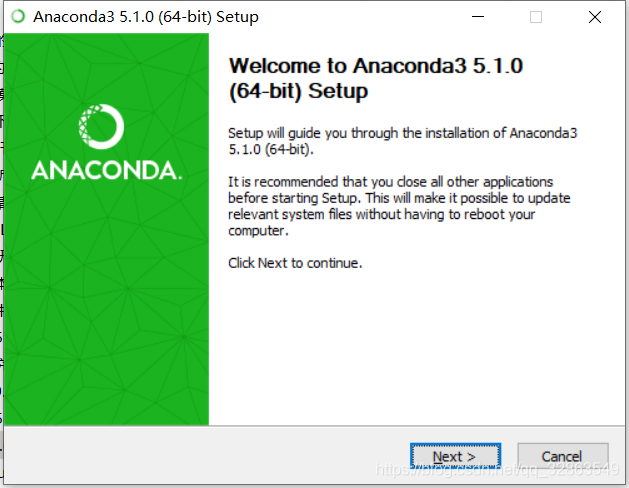
# 相关工作：Pytorch环境搭建

### 1.Pytorch是什么？



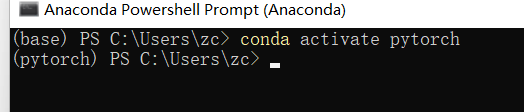
### 2.pytorch安装

Anaconda安装

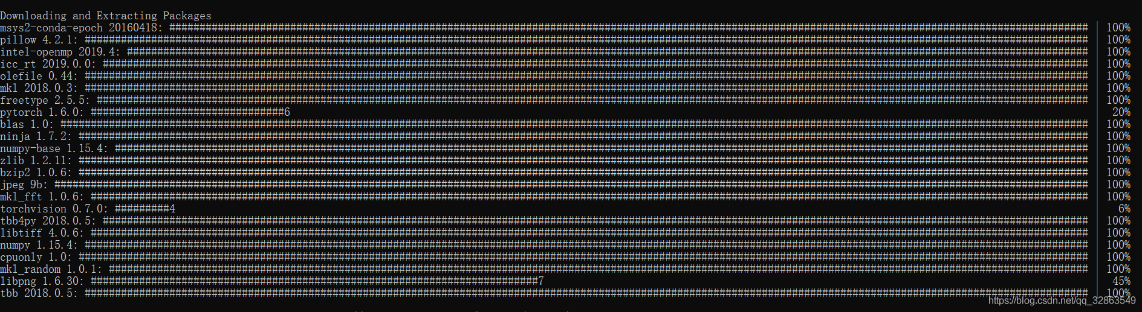
****

**创建虚拟房间**

**激活pytorch房间**

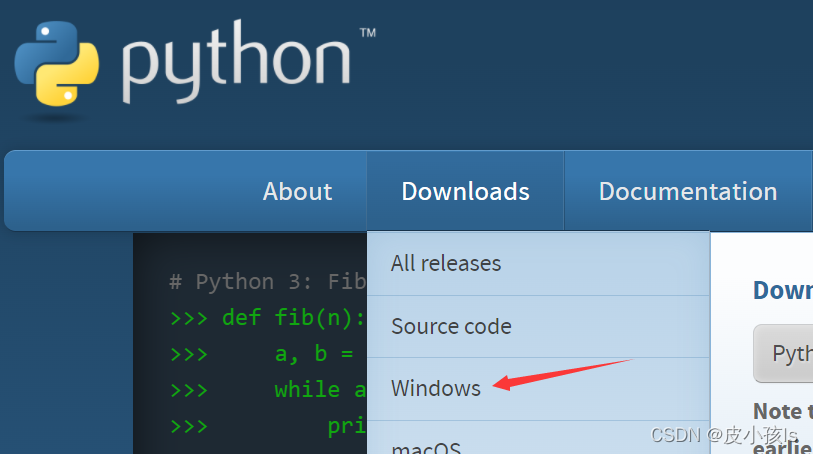
****

**安装pytorch**

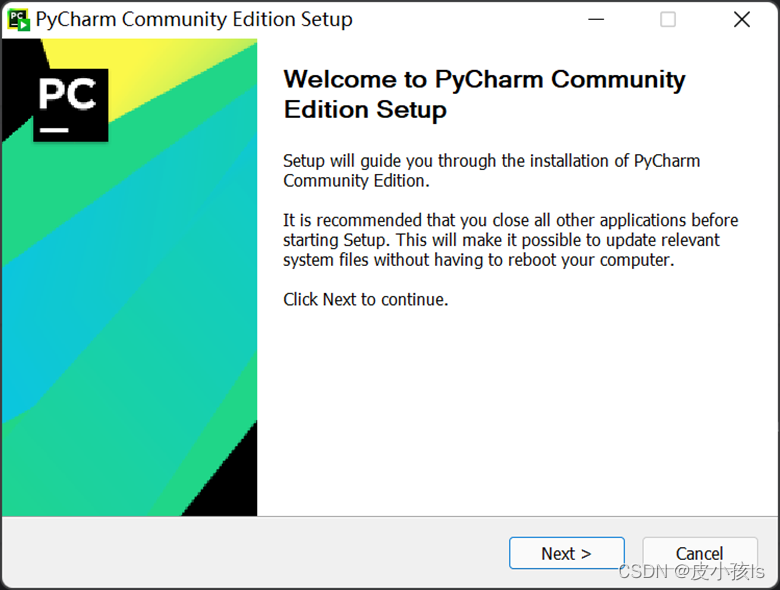


### 3.Pycharm安装

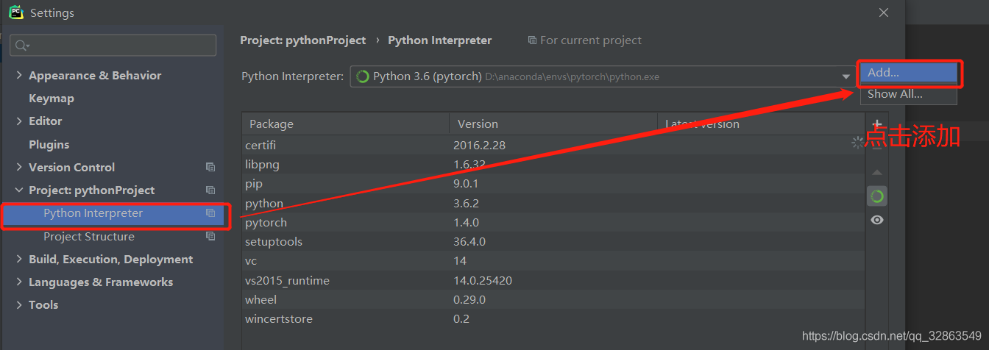
**配置python**

****

**安装pycharm**

****

**在pycharm中添加pytorch房间的解释器**



# 方法

#### 卷积神经网络(CNN)：

CNN用的是权值共享，即后一层不同神经的输入会共享相同的权值组，且后一层每个神经元的输入只受前一层部分神经元输出的影响。这样做的好处的避免纠缠不必要的细节，而是尽量抽象出整体特征，这点十分适用于图像识别。

##### 神经网络框架：

#### 1.卷积

卷积层是卷积神经网络的核心层，核心的处理方式就是卷积计算。卷积其实也就可以看成一个函数或者一种算法。这个函数则需要输入数据和卷积核，按照卷积要求进行计算。

卷积层其实是提取图像特征的过程。

#### 2.激活

为了保证对数据非线性处理，也需要激活函数，也就是激活层的处理。其处理方式是，为卷积核的每个元素添加一个bias(偏移值)，然后送入诸如relu、leakyRelu、tanh等非线性激活函数即可。

#### 3.池化

池化(Pooling)又称下采样，可以进一步降低网络训练参数和模型过拟合的程度。

常用的池化处理有一下几种：

* 最大池化(Max Pooling)：选择Pooling窗口中的最大值作为采样值
* 均值池化(Mean Pooling)：将Pooling窗口中的所有值加起来取平均，使用平均值作为采样值
* 全局最大(或均值)池化：取整个特征图的最大值或均值

#### 4.全连接层和输出层

这部分主要连接最后池化后的结果，将池化后的数据展平构成全连接层的输入。然后就是根据类别数构建的一个分类层，也就是输出层。

对于分类任务输出层则添加一个sigmoid层计算需要分类的图片各个类别的概率。对于训练任务，则使用损失函数开始反向传播更新模型中的卷积核。

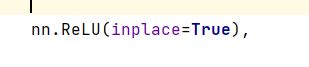
# 实验

### 一.网络结构的实现：

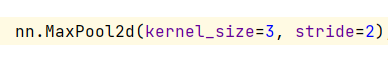
###### 卷积:二维卷积



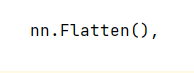
激活：inplace为True，将会改变输入的数据



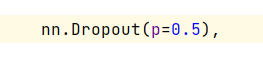
池化：最大池化（二维）



Flatten：降维展平



Dropout：减少过拟合



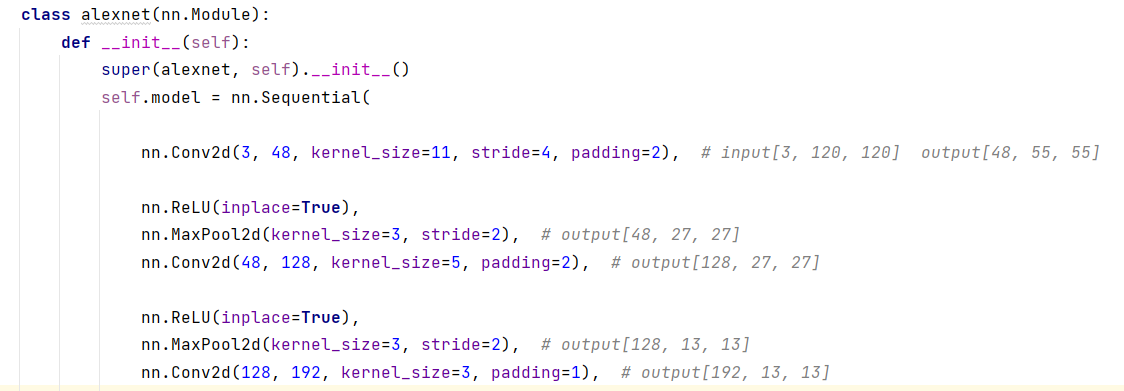
Linear：设置全连接层

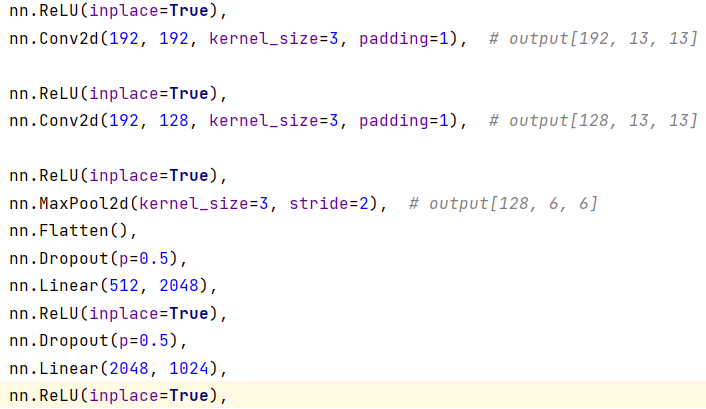


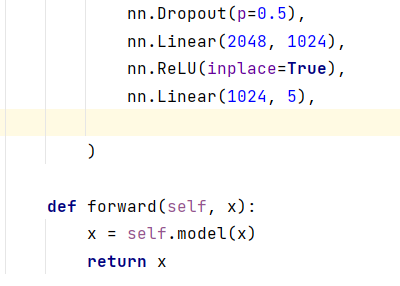
向前传播：forward

### 

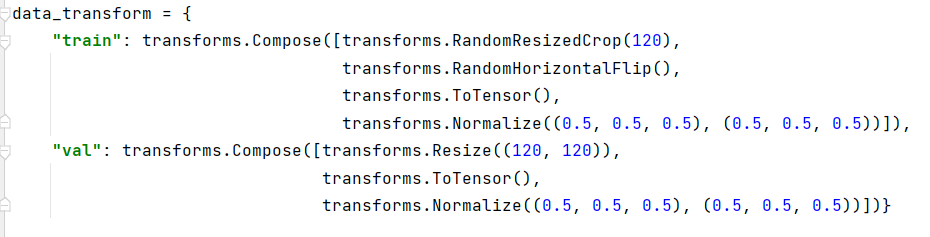
### 二.神经网络搭建全过程：







#### 三.数据预处理：



torchvision.transforms 是pytorch中的图像预处理包，一般用Compose把多个图片变换步骤整合到一起。

（实际是个列表，而这个列表里面的元素就是你想要执行的transform操作。Compose()类会将transforms列表里面的transform操作进行遍历。）

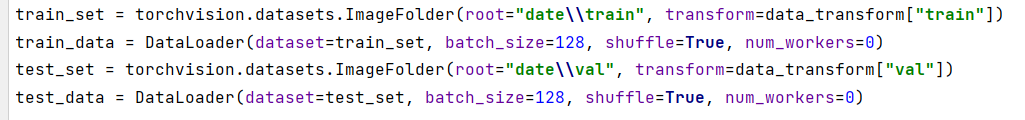
#### 运用的图形变换：

CenterCrop:在图形的中间区域进行裁剪

RandoHorizontalFlip:以0.5的概率水平翻转图像

Totensor:将shape为（H,W,C）的图像转为shape为（C,H,W）的tensor。（Tensor即为描述网络输入、输出、参数的张量）

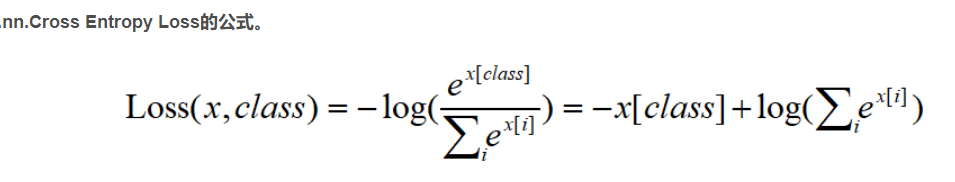
Normalize:进行标准化、归一化处理，把0~255的灰度值像素数据归一化为均值为0.5、标准差为0.5的数据。



DateLoader:指定数据集，且分批次按Batch\_size=128读取，shuffle=True则表示对数据随机读取

**四.损失函数与反向传播：**

#### (1)nn.CrossEntropyLoss(),交叉熵损失：

例：一个三分类问题，有person（0）,cat（1）,dog（2）三个类

Output：[0.1,0.2,0.3]

Target:1

则,loss=-0.2+log(exp(0.1)+exp(0.2)+exp(0.3))=1.1019

(2)优化器**：**调用损失函数的backward（反向传播），求出每个需要调节的参数对应的梯度。此时使用优化器利用这个梯度对参数进行调整，以达到降低整体误差的目的。

optimizer = optim.SGD(net.parameters()，1e-2)

#择优化器算法，调用模型参数，学习率=0.001

optimizer.step()

#调整参数

# 全部代码(及部分注释)：

# import torch import numpy as np import torchvision from torch.utils.data import DataLoader from torchvision.datasets import mnist from torch import nn from torch.autograd import Variable from torch import optim from torchvision import transforms

*# 定义神经网络***class** CNN(nn.Module):  
 **def** \_\_init\_\_(self):  
 super(CNN, self).\_\_init\_\_()

#调用父类初始化函数  
  
 self.model = nn.Sequential(

#将网络结构放在一个序列当中，可以方便forward函数的书写  
  
 nn.Conv2d(3, 48, kernel\_size=11, stride=4, padding=2),

#卷积层 *input[3, 120, 120] output[48, 55, 55]* nn.ReLU(inplace=**True**),

#激活层，非线性处理  
 nn.MaxPool2d(kernel\_size=3, stride=2),

#最大池化  *output[48, 27, 27]*

nn.Conv2d(48, 128, kernel\_size=5, padding=2),

*# output[128, 27, 27]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2),

*# output[128, 13, 13]* nn.Conv2d(128, 192, kernel\_size=3, padding=1),

*# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
  
 nn.Conv2d(192, 192, kernel\_size=3, padding=1),

*# output[192, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.Conv2d(192, 128, kernel\_size=3, padding=1),

*# output[128, 13, 13]* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(kernel\_size=3, stride=2),

*# output[128, 6, 6]* nn.Flatten(),

#降维展平  
 nn.Dropout(p=0.5),

#防止模型过拟合  
 nn.Linear(512, 2048),

#线性层  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(p=0.5),  
 nn.Linear(2048, 1024),  
 nn.ReLU(inplace=**True**),  
 nn.Linear(1024, 5),  
 )  
 **def** forward(self, x):

#前向传播  
 x = self.model(x)  
 **return** x  
  
*# 预处理*

data\_transform = {  
 **"train"**: transforms.Compose([transforms.CenterCrop(120),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]),  
 **"val"**: transforms.Compose([transforms.CenterCrop(120),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])}  
  
train\_set = torchvision.datasets.ImageFolder(root=**"date\\train"**, transform=data\_transform[**"train"**])

#准备数据集  
train\_data = DataLoader(dataset=train\_set, batch\_size=128, shuffle=**True**, num\_workers=0)  
#加载数据集  
test\_set = torchvision.datasets.ImageFolder(root=**"date\\val"**, transform=data\_transform[**"val"**])  
test\_data = DataLoader(dataset=test\_set, batch\_size=128, shuffle=**True**, num\_workers=0)  
net = CNN()  
criterion = nn.CrossEntropyLoss() *# 使用交叉熵损失*optimizer = optim.SGD(net.parameters(), 1e-2) *# 随机梯度下降，学习率为0.1*nums\_epoch = 20

#训练轮数  
device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

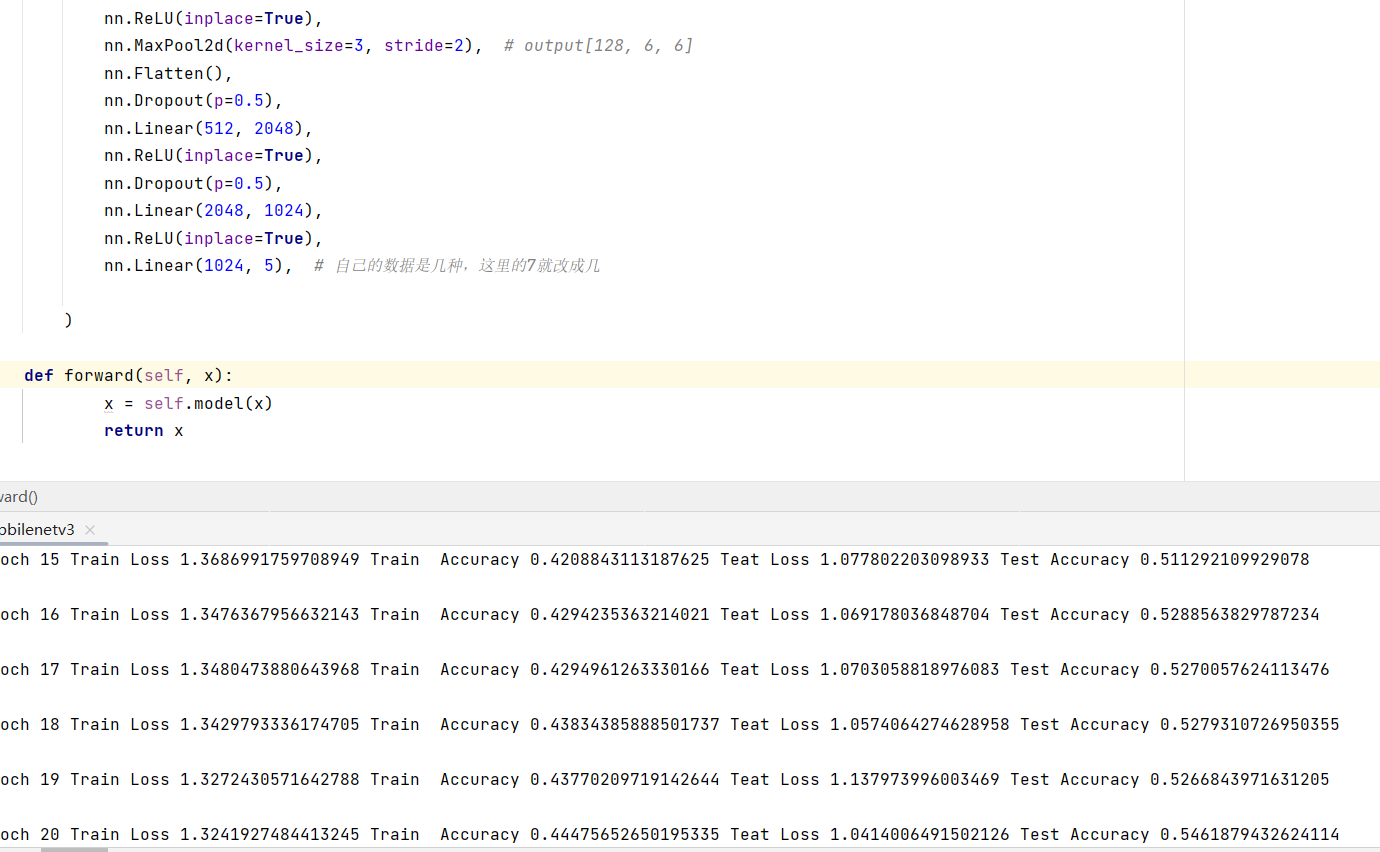
#优先使用GPU(如果有)

*# 开始训练*  
**for** epoch **in** range(nums\_epoch):  
 print(epoch + 1)  
 train\_loss = 0  
 train\_acc = 0  
 net = net.train()  
 **for** img, label **in** train\_data:img.to(device)  
 label.to(device)  
 img = Variable(img)label = Variable(label)  
out = net(img)  
 loss = criterion(out, label)optimizer.zero\_grad() *# 每次将梯度重置为0* loss.backward() *# 反向调整参数* optimizer.step()  
  
 *# 记录误差* train\_loss += loss.item()  
 *# 计算分类的准确率* pred = out.max(1) *# 取评分最高的结果作为所分的类别* num\_correct = (pred == label).sum().item()  
 acc = num\_correct / img.shape[0]  
  
 train\_acc += acc  
  
 eval\_loss = 0  
 eval\_acc = 0  
 *# 测试集不训练* **for** img, label **in** test\_data:  
 img.to(device)  
 label.to(device)  
  
 net.eval()img = Variable(img)  
 label = Variable(label)  
  
 out = net(img)  
  
 loss = criterion(out, label)  
  
 *# 记录误差* eval\_loss += loss.item()  
  
 \_, pred = out.max(1)  
 num\_correct = (pred == label).sum().item()  
 acc = num\_correct / img.shape[0]  
  
 eval\_acc += acc  
 print(**'Epoch {} Train Loss {} Train Accuracy {} Teat Loss {} Test Accuracy {}'**.format(  
 epoch + 1, train\_loss / len(train\_data), train\_acc / len(train\_data), eval\_loss / len(test\_data),  
 eval\_acc / len(test\_data)))  
 torch.save(net.state\_dict(), **"net.pth"**)

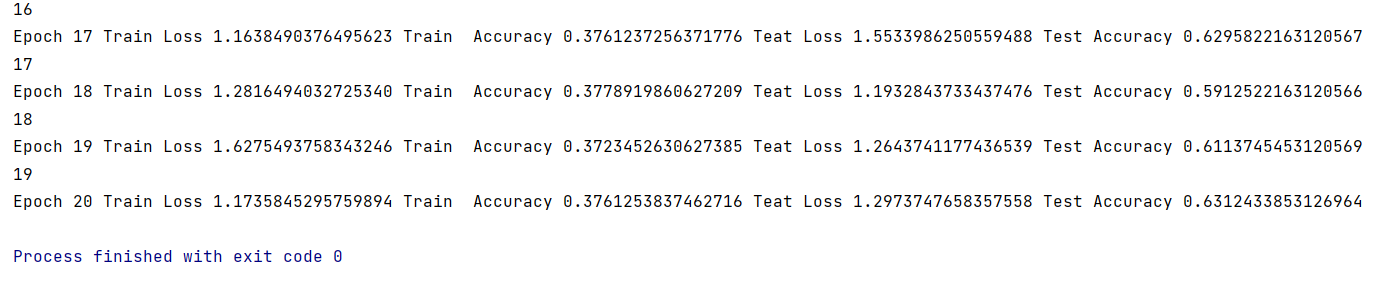
#保存模型至“net.pth”  
 print(**"模型已保存"**)

# 结论：

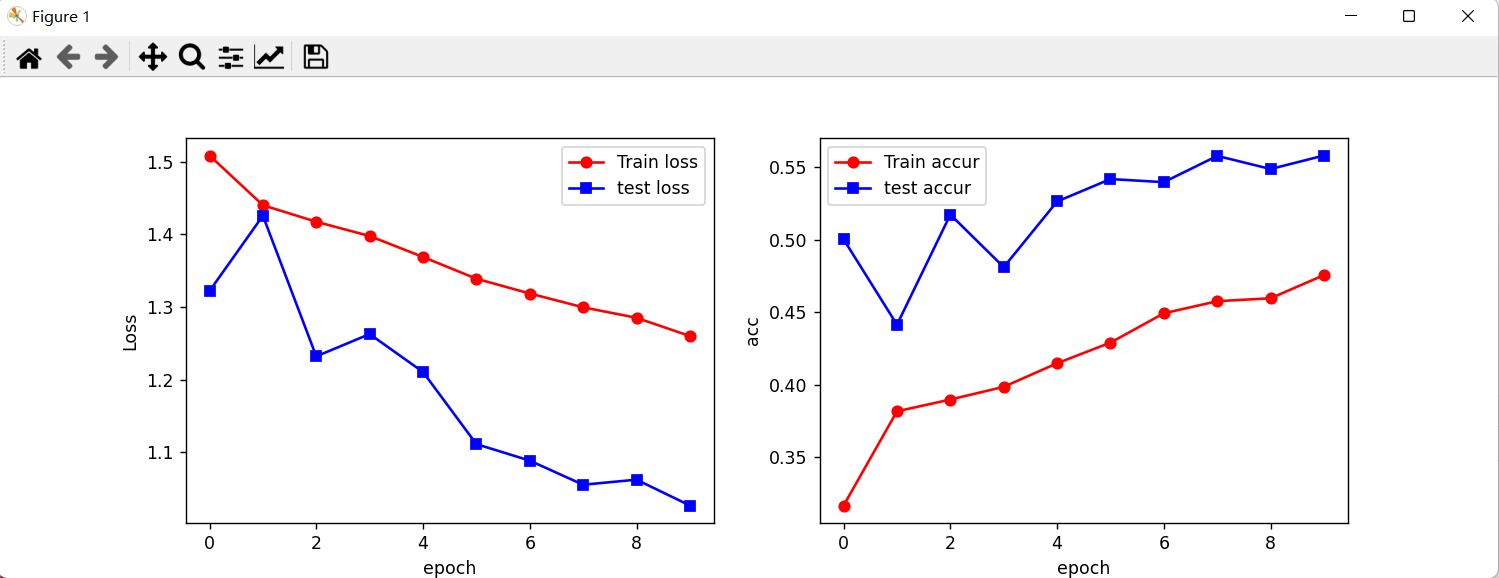
#### 训练结果1：



#### 训练结果2：



#### 测试集和训练集的损失与正确率的可视化：



**由以上可得，该模型的loss随训练轮数逐渐降低，accur随训练轮数逐渐升高。**

**平均正确率达到0.57左右，最高可达到0.63**

**预测的结果较好**

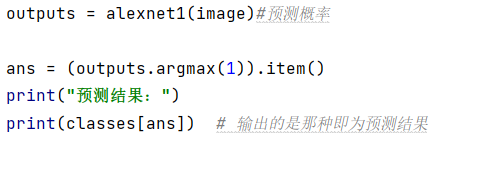
# 优化：

#### 一.单张图片的预测：



此函数将训练好的模型参数保存到“net.pth”路径中，有此文件，我们可以不再经过训练直接预测图片的类别，以单张为例。



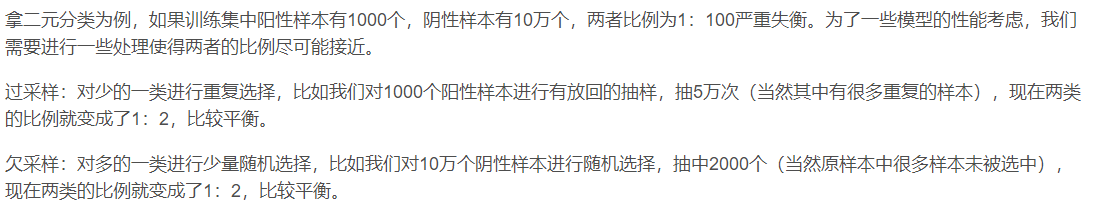


#### 二．类别不平衡的分类问题

即某些类在样本中占比过大，某些类在样本中占比却过小，训练的模型可能会将数据都标注为占比大的样本。

###### 解决方法：

1. 重采样：队少样本过采样，对多样本欠采样。



1. 数据合成：若不想直接重复采样相同样本，一种解决方法是生成和少样本相似的“新”数据。
2. 重加权：重加权是对不同类别（甚至不同样本）分配不同权重。
3. 模型集成：融合多个训练好的模型，基于某种方式实现测试数据的多模型融合，这样来使最终的结果能够“取长补短”，融合各个模型的学习能力，提高最终模型的[泛化](https://so.csdn.net/so/search?q=%E6%B3%9B%E5%8C%96&spm=1001.2101.3001.7020)能力

# 大作业心得：

因为之前并没有接触Python，所以在这一学期的Python学习也算痛并快乐着。

但看到大作业跑通的时刻，总感觉还是可喜可贺，即便需要优化的地方还有很多，即便正确率不是很高。

Python确实是很强大的，就算只学了半学期的Python，在运用上竟已经超过练习时长超过一年半的C++。而且像数电课、概率论、汇编这些课程，还经常接触到Python来暴力运算，让我不得不感叹。

相信Python还会陪伴我的编程生涯。