一、 网络搭建

【1. 数据在网络中的维度顺序是什么？】

=(input)=> 1*1*112*112

=(conv1_1)=> 1*8*54*54 =(pool)=> 1*8*27*27

=(conv2_1)=> 1*16*25*25 =(conv2_2)=> 1*16*23*23 =(pool)=> 1*16*12*12

=(conv3_1)=> 1*24*10*10 =(conv3_2)=> 1*24*8*8 =(pool)=> 1*24*4*4

=(conv4_1)=> 1*40*4*4 =(conv4_2)=> 1*80*4*4

=(ip1)=> 1*128 =(ip2)=> 1*128 =(ip3)=> 1*42

格式为：N*C*H*W，N表示batch size(由于batch size不定，此处用1表示)，C表示channel个数，H，W分别表示高和宽

【2. nn.Conv2d()中参数含义与顺序？】

nn.Conv2d(in_channels，out_channels，kernel_size，stride=1，padding=0，dilation=1，groups=1，bias=True)

参数含义：

in_channels：输入信号的通道

out_channels：卷积通道数

kernel_size：卷积核尺寸

stride=1：卷积步长，默认为1

padding=0：输入的每一条边补充的层数，默认为0；pytorch的padding策略为四周都补，padding如果补0时会补上bias的值

dilation=1：卷积核元素之间的距离

groups=1：从输入到输出通道的阻塞连接数

bias=True：添加偏置

【3. nn.Linear()是什么意思？参数含义与顺序？】

nn.Linear(in_features, out_features, bias=True)：对传入数据应用线性变换

参数含义：

in_features：每个输入样本的大小

out_features：每个输出样本的大小

bias=True：添加偏置

【4. nn.PReLU()与nn.ReLU()的区别？示例中定义了很多nn.PReLU()，能否只定义一个PReLU？】

ReLU：nn.ReLU(inplace=False)
对输入运用修正线性单元函数ReLU(x)= max(0, x)
PReLU：nn.PReLU(num_parameters=1, init=0.25)
对输入的每一个元素运用函数PReLU(x) = max(0,x) + a * min(0,x)，a是一个可学习参数。当没有声明时，nn.PReLU()在所有的输入中只有一个参数a；如果是nn.PReLU(nChannels)，a将应用到每个输入。
PReLU和ReLU的区别：

- RELU让矩阵内负值变为0，矩阵输入的正值保持和输出一致
- PReLU让矩阵的负值不再为0，而乘以一个参数a，矩阵输入的正值保持和输出一致

本示例中不能只定义一个PReLU，因为需要对各个PReLU调整参数a的值

【5. nn.AvgPool2d()中参数含义？还有什么常用的pooling方式？】

nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True)
参数含义：
kernel_size：池化窗口大小
stride=None：max pooling的窗口移动的步长，默认值是kernel_size
padding=0：输入的每一条边补充0的层数
ceil_mode=False：ceil_mode - 如果等于True，计算输出信号大小的时候，会使用向上取整，代替默认的向下取整的操作
count_include_pad=True：如果等于True，计算平均池化时，将包括padding填充的0

其他常用的pooling方式有（不显示维数）：

-最大值池化 nn.MaxPool
-最大值反向池化 nn.MaxUnpool
-自适应最大值池化（可以指定输出size）nn.AdaptiveMaxPool
-自适应平均值池化（可以指定输出size）nn.AdaptiveAvgPool
-2维的幂平均池化 torch.nn.LPPool2d

【6. view()的作用？】

view(*args) → Tensor

返回一个有相同数据但大小不同的tensor。 返回的tensor必须有与原tensor相同的数据和相同数目的元素，但可以有不同的大小。一个tensor必须是连续的contiguous()才能被查看。

本示例中，通俗来讲，就是将一个多行的Tensor,拼接成一行

二、 训练框架搭建

【1. 如何设置GPU】

一般需要确保GPU是可以使用，可通过torch.cuda.is_available()的返回值来进行判断。返回True则具有能够使用的GPU。通过torch.cuda.device_count()可以获得能够使用的GPU数量。

代码：device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

【2. 如何将数据/网络传入CPU/GPU】

网络传入GPU/CPU(device):

-model.to(device)　#使用序号为0的GPU

-model.to(device1）#使用序号为1的GPU

数据传入GPU(CPU):

将上述代码model改为input_data

【3. 如何读取数据】

利用数据迭代器torch.utils.data.DataLoader，从数据集中读取，需要定义batch_size。

代码：train_loader = torch.utils.data.DataLoader(train_set, batch_size=16, shuffle=True)

【4. 如何设置loss】

有一些不同的损失函数在 nn 包中。定义损失函数需要一对输入：模型输出和目标，然后计算一个值来评估输出距离目标有多远。例如本示例里使用的简单的损失函数 nn.MSELoss ，这计算了均方误差。

代码：criterion = nn.MSELoss()

【5. 配合后续周学习，loss都有哪些。分别有什么作用(常用的即可)】

nn.SmoothL1Loss：也叫作 Huber Loss，误差在 (-1,1) 上是平方损失，其他情况是 L1 损失。

nn.MSELoss：平方损失函数。

nn.BCELoss：二分类用的交叉熵，TODO。

nn.CrossEntropyLoss：交叉熵损失函数。

nn.NLLLoss：负对数似然损失函数（Negative Log Likelihood）。

nn.KLDivLoss：KL 散度，又叫做相对熵，算的是两个分布之间的距离，越相似则越接近零。

nn.MarginRankingLoss：评价相似度的损失。

nn.MultiMarginLoss：多分类（multi-class）的 Hinge 损失。

nn.MultiLabelMarginLoss：多类别（multi-class）多分类（multi-classification）的 Hinge 损失，是上面 MultiMarginLoss 在多类别上的拓展。同时限定 p = 1，margin = 1。

nn.SoftMarginLoss：多标签二分类问题，这 项都是二分类问题，其实就是把 个二分类的 loss 加起来，化简一下。其中 只能取 两种，代表正类和负类。和下面的其实是等价的，只是 的形式不同。

nn.MultiLabelSoftMarginLoss：上面的多分类版本，根据最大熵的多标签 one-versue-all 损失。

nn.CosineEmbeddingLoss：余弦相似度的损失，目的是让两个向量尽量相近。注意这两个向量都是有梯度的。

【6. 如何设置优化器】

可直接利用torch.optim包里的方法，例如SGD, Nesterov-SGD, Adam, RMSProp等。

代码：optimizer = optim.SGD(model.parameters(), lr=0.0001, momentum=0.5)

【7. 配合第8周内容，常用的优化器有哪些】

第八周讲到的常用优化器：

-SGD+Momentum：在原始梯度下降上加冲量

-Nesterov：用超前梯度更新冲量

-Adagrad：学习率随历史梯度平方和上升而下降

-RMSProp：学习率随历史加权梯度平方和上升而下降

-Adam：结合学习率自适应和冲量方法

三、 Train部分

【1. print的格式化如何实现的】

用“from __future__ import print_function”来实现，用来解决此function的特性和当前版本使用的此function特性不兼容的问题。

【2. optimizer.zero()与optimizer.step()的作用是什么？】

optimizer.zero()：把之前参数的梯度清空为零。
optimizer.step()：在backward()方法算出梯度之后更新参数。

【3. model.eval()产生的效果？】

模型转换在测试时，当模型中有BN层和Dropout层时，有如下作用：

BN：训练模式时，BN采用每一批数据的均值和方差。而在测试模式时，BN会用到全部训练数据的均值和方差；

Dropout：训练模式时，每个隐层中的神经元会乘以随机概率P，然后激活，这样一部分神经元会停止工作，以防止过拟合。而在测试模式时，所有神经元会全部进行激活，然后每个神经元的输出乘以概率P，以得到同样的期望。

【4. model.state_dict()的目的是？】

model.state_dict()返回的是一个OrderDict，目的是存储了网络结构的名字和对应的参数。用于在 Pytorch 中一种模型保存和加载的方式：torch.save(model.state_dict(), PATH)。

【5. 何时系统自动进行bp？】

在利用计算结果调用backward()函数时，例如本示例中的：loss.backward()。

【6. 如果自己的层需要bp，如何实现？如何调用？】

在定义tensor的时候，将require_grad属性设置为True，例如：x = torch.ones(2,4,require_grad=True)
在需要求导时，调用backward()方法，例如：y.backward()。

四、 Finetune部分

【1. Finetune时，有时还要固定某些层不参与训练，请回答如何freeze某些层。】

固定某些层不参与训练，直接将其requires_grad置False即可，例如：
for para in list(model.parameters())[不参与训练的层]:

para.requires_grad = False #取消自动求导