**问题1：运行时No module named 'torchvision'**

解决：pip install torchvision

**问题2：莫名其妙报错**

解决：看看是不是用了中文字符

**问题3：堆叠网络的时候，卷积核大小报错**

解决：误以为maxpooling的stride=2的时候会使kernel也减半

**问题4：如何训练网络**

1、网络Loss一直居高不下，找了半天没找到原因，后来发现是

optimizer =torch.optim.RMSprop(tuned\_model\_gpu.parameters(),lr=1e3)写错了。

tuned\_model\_gpu写成理论另外一个模型，怪不得一直没变化

2、解决1之后，loss还是没有明显下降趋势，后来发现是weight\_decay用的太大了，刚开始用的1e-2，后来查资料发现一般用1e-8。

**总结一般学习率0.1，然后除以5或者3，正则化参数1e-8,batch size=32，64，128**

Lr\_decay越大，学习率衰减地越快

**训练一个神经网络需要：**

1、利用小批量数据对实现进行梯度检查，还要注意各种错误。

2、进行合理性检查，确认初始损失值是合理的，在小数据集上能得到100%的准确率。

3、在训练时，跟踪损失函数值，训练集和验证集准确率，如果愿意，还可以跟踪更新的参数量相对于总参数量的比例（一般在1e-3左右），然后如果是对于卷积神经网络，可以将第一层的权重可视化。

4、推荐的两个更新方法是SGD+Nesterov动量方法，或者Adam方法。

随着训练进行学习率衰减。比如，在固定多少个周期后让学习率减半，或者当验证集准确率下降的时候。

5、使用随机搜索（不要用网格搜索）来搜索最优的超参数。分阶段从粗（比较宽的超参数范围训练1-5个周期）到细（窄范围训练很多个周期）地来搜索。

进行模型集成来获得额外的性能提高。

**网络结构的选择：**

INPUT -> [[CONV -> RELU]\*N -> POOL?]\*M -> [FC -> RELU]\*K -> FC

INPUT -> [CONV -> RELU -> POOL]\*2 -> FC -> RELU -> FC。此处在每个汇聚层之间有一个卷积层。

INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]\*3 -> [FC -> RELU]\*2 -> FC。此处每个汇聚层前有两个卷积层，这个思路适用于更大更深的网络，因为在执行具有破坏性的汇聚操作前，多重的卷积层可以从输入数据中学习到更多的复杂特征。

**用多个小的filter代替大的单个filter**

1. 首先，多个卷积层与非线性的激活层交替的结构，比单一卷积层的结构更能提取出深层的更好的特征。
2. 多个小filter能达到一个大的filter的效果
3. 使用多个小的的参数更少。唯一的不足是，在进行反向传播时，中间的卷积层可能会导致占用更多的内存。

层的尺寸设置规律

1. 输入层（包含图像的）应该能被2整除很多次。常用数字包括32（比如CIFAR-10），64，96（比如STL-10）或224（比如ImageNet卷积神经网络），384和512。
2. 卷积层应该使用小尺寸滤波器（比如3x3或最多5x5），使用步长S=1。还有一点非常重要，就是对输入数据进行零填充，这样卷积层就不会改变输入数据在空间维度上的尺寸。比如，当F=3，那就使用P=1来保持输入尺寸。当F=5,P=2，一般对于任意F，当P=(F-1)/2的时候能保持输入尺寸。
3. 汇聚层负责对输入数据的空间维度进行降采样。最常用的设置是用用2x2感受野（即F=2）的最大值汇聚，步长为2（S=2）。注意这一操作将会把输入数据中75%的激活数据丢弃（因为对宽度和高度都进行了2的降采样）。另一个不那么常用的设置是使用3x3的感受野，步长为2。最大值汇聚的感受野尺寸很少有超过3的，因为汇聚操作过于激烈，易造成数据信息丢失，这通常会导致算法性能变差。
4. 为什么在卷积层使用1的步长？在实际应用中，更小的步长效果更好。上文也已经提过，步长为1可以让空间维度的降采样全部由汇聚层负责，卷积层只负责对输入数据体的深度进行变换。
5. 为何使用零填充？使用零填充除了前面提到的可以让卷积层的输出数据保持和输入数据在空间维度的不变，还可以提高算法性能。如果卷积层值进行卷积而不进行零填充，那么数据体的尺寸就会略微减小，那么图像边缘的信息就会过快地损失掉。
6. 因为内存限制所做的妥协：在某些案例（尤其是早期的卷积神经网络结构）中，基于前面的各种规则，内存的使用量迅速飙升。例如，使用64个尺寸为3x3的滤波器对224x224x3的图像进行卷积，零填充为1，得到的激活数据体尺寸是[224x224x64]。这个数量就是一千万的激活数据，或者就是72MB的内存（每张图就是这么多，激活函数和梯度都是）。因为GPU通常因为内存导致性能瓶颈，所以做出一些妥协是必须的。在实践中，人们倾向于在网络的第一个卷积层做出妥协。例如，可以妥协可能是在第一个卷积层使用步长为2，尺寸为7x7的滤波器（比如在ZFnet中）。在AlexNet中，滤波器的尺寸的11x11，步长为4。

**计算上的考量**

1、在构建卷积神经网络结构时，最大的瓶颈是内存瓶颈。大部分现代GPU的内存是3/4/6GB，最好的GPU大约有12GB的内存。要注意三种内存占用来源：

2、来自中间数据体尺寸：卷积神经网络中的每一层中都有激活数据体的原始数值，以及损失函数对它们的梯度（和激活数据体尺寸一致）。通常，大部分激活数据都是在网络中靠前的层中（比如第一个卷积层）。在训练时，这些数据需要放在内存中，因为反向传播的时候还会用到。但是在测试时可以聪明点：让网络在测试运行时候每层都只存储当前的激活数据，然后丢弃前面层的激活数据，这样就能减少巨大的激活数据量。

3、来自参数尺寸：即整个网络的参数的数量，在反向传播时它们的梯度值，以及使用momentum、Adagrad或RMSProp等方法进行最优化时的每一步计算缓存。因此，存储参数向量的内存通常需要在参数向量的容量基础上乘以3或者更多。

4、卷积神经网络实现还有各种零散的内存占用，比如成批的训练数据，扩充的数据等等。

一旦对于所有这些数值的数量有了一个大略估计（包含激活数据，梯度和各种杂项），数量应该转化为以GB为计量单位。把这个值乘以4，得到原始的字节数（因为每个浮点数占用4个字节，如果是双精度浮点数那就是占用8个字节），然后多次除以1024分别得到占用内存的KB，MB，最后是GB计量。如果你的网络工作得不好，一个常用的方法是降低批尺寸（batch size），因为绝大多数的内存都是被激活数据消耗掉了。

**关于pytorch**

1、x.view(N,-1)=x.reshape(N,-1)//python

2、nn.Sequential，是一个container可以用来一层一层的堆叠网络

3、nn.Conv2d(3, 32, kernel\_size=3, stride=1,padding=1)#（input\_channel,outputchannel…），这里的参数可以做到输入输出大小相同

4、nn.ReLU(inplace=True)有个inplace参数，如果设为True,它会把输出直接覆盖到输入中，这样可以节省内反向传播的梯度

5、nn.Linear(infeature,outfeature)

6、optimizer.zero\_grad()##将参数的grad值初始化为0

7、loss\_fn = nn.CrossEntropyLoss(weight=None,size\_average=True)

8、optimizer = torch.optim.RMSprop(fixed\_model\_gpu.parameters(),lr=1e-3,weight\_decay=1e-8)，正则化在optim算法中通过weight\_decay实现

9、DROPOUT

torch.nn.Dropout(0.5),以及torch.nn.Dropout2d(0.5),

10、查看当前gpu型号数量等

torch.cuda.device\_count()

torch.cuda.get\_device\_name(0)

torch.cuda.current\_device()

11、衰减学习率

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=50, gamma=0.1)

scheduler.step()

train(tuned\_model\_gpu, loss\_fn, optimizer, num\_epochs=30)

optimizer (Optimizer) – 包装的优化器。

step\_size (int) – 学习率衰减期。

gamma (float) – 学习率衰减的乘积因子。默认值:-0.1。

last\_epoch (int) – 最后一个时代的指数。默认值:1

将每个参数组的学习速率设置为每个step\_size时间段由gamma衰减的初始lr。当last\_epoch = -1时，将初始lr设置为lr。即每50次迭代使学习率衰减0.1.

一些官方文档

Layers: http://pytorch.org/docs/nn.html

Activations: http://pytorch.org/docs/nn.html#non-linear-activations

Loss functions: http://pytorch.org/docs/nn.html#loss-functions

Optimizers: <http://pytorch.org/docs/optim.html#algorithms>

关于cuda

1. cuda 数据类型gpu\_dtype = torch.cuda.FloatTensor

pytorch类型：dtype = torch.FloatTensor

关于建立一个CIFAR-10上，验证集准确率大于70%的网络

1. 网络架构方法
2. 小filter更高效
3. 使用更多的filter
4. 使用max pooling或者stride conv
5. 卷积层后spatial BN, affline层后vanilla BN
6. 使用更好的网络结构
7. 使用global average pooling。即先用多种卷积使图片变为7\*7 or so。然后使用average pooling变为1\*1.然后reshape为vector
8. 使用L2正则化或者使用Dropout
9. 训练技巧

T调整每种网络模型的学习率和正则化强度

1. 如果当前参数很好的话，在几百次迭代后应该看到结果提升
2. 超参数调整应由粗到细：由大范围超参数训练几个epoch开始，以找到不错的参数
3. 找到不错的之后，进一步细分，并使用更多的epoch
4. 使用验证集来寻找超参数，并用测试集评估使用验证集找到的网络结构

**Q:为什么要全连接层转换为卷积层**

A：1、全连接层和卷积层首先是可以相互转化的

2、当想要不改变网络结构将输入图片换成一张更大的图片的时候，对使用全连接的网络来说是不行的，必须将大图片分解为小图片才可以让网络正确运行。但是对于用卷积替代了全连接的网络来说，却可以做到正确运行。

3、在2这种情况下，使用卷积的效率比将大图片分成小图片分别输入网络的效率更高。因为将大图片切成224x224后得到36张图片，这36张图片有很多区域重叠了，本来一次池化就可以得到结果的，但是输入36张图片后，相同区域进行了多次重复池化操作，因此耗费了更多的时间。

**Global average pooling**

1、global average pooling 与 average pooling 的差别就在 "global" 这一个字眼上。global 与 local 在字面上都是用来形容 pooling 窗口区域的。 local 是取 feature map 的一个子区域求平均值，然后滑动这个子区域； global 显然就是对整个 feature map 求平均值了。

global average pooling 的最后输出结果仍然是 10 个 feature map，而不是一个，只不过每个 feature map 只剩下一个像素罢了。这个像素就是求得的平均值。

2、全连接层过多的参数重要到会造成过拟合，每个讲到全局池化的都会说GAP就是把avg pooling的窗口大小设置成feature map的大小，这虽然是正确的，但这并不是GAP内涵的全部。GAP的意义是对整个网络从结构上做正则化防止过拟合。既要参数少避免全连接带来的过拟合风险，又要能达到全连接一样的转换功能，怎么做呢？直接从feature map的通道上下手，如果我们最终有1000类，那么最后一层卷积输出的feature map就只有1000个channel，然后对这个feature map应用全局池化，输出长度为1000的向量，这就相当于剔除了全连接层黑箱子操作的特征，直接赋予了每个channel实际的类别意义。

1. 进行全连接的替换，减少参数的数量。还有对网络结构进行正则化的效果

**Spatial batch normalization**是用在卷积后的BN

全连接后的BN由N\*D的输入，输出N\*D的输出。而Spatial batch normalization的输入是，卷积后大小为(N, C, H, W)，输出为(N, C, H, W)，每个特征图看成是一个特征处理（一个神经元），对于每个特征图都只有一对可学习参数：γ、β

神经网络中Epoch、Iteration、Batchsize相关理解和说明

https://blog.csdn.net/program\_developer/article/details/78597738

**batch size**：中文翻译为批大小（批尺寸），批量大小将决定我们一次训练的样本数目。

batch\_size将影响到模型的优化程度和速度。

**iteration**：中文翻译为迭代。迭代是重复反馈的动作，神经网络中我们希望通过迭代进行多次的训练以达到所需的目标或结果。

每一次迭代得到的结果都会被作为下一次迭代的初始值。

一个迭代=一个正向通过+一个反向通过。

**epoch：**中文翻译为时期。一个时期=所有训练样本的一个正向传递和一个反向传递

（1）batchsize：批大小。在深度学习中，一般采用SGD训练，即每次训练在训练集中取batchsize个样本训练；

（2）iteration：1个iteration等于使用batchsize个样本训练一次；

（3）epoch：1个epoch等于使用训练集中的全部样本训练一次；

Batch是什么？

在不能将数据一次性通过神经网络的时候，就需要将数据集分成几个batch。

iteration是batch需要完成一个epoch的次数。