**A.在深度学习中，训练时有哪一些可以调节的超参数？**

凡是需要手动设置且会影响模型结果的均可称为超参数

优化器：优化器类型、各层学习率、weight\_decay正则化系数等

Dropout：是否使用、神经元失活随机概率

BN：是否使用

学习率下降策略

输入网络：图片尺寸大小、批次大小、图片预处理方法等

损失函数：函数类型、相关参数如smoothl1需要的beta参数等

网络结构：多少层网络结构、每层多少神经元(这个倒是没用过，就是听说过)

训练轮数

提前终止判断条件

**A.Pytorch中有哪一些优化器，可用于调节的参数有哪一些？**

有SGD、Adam、RMSprop这些优化器，可调参数：主要就是lr、weight\_decay、SGD还有momentum参数等，lr是最主要的参数其数值大小以及下降策略对于模型的效果影响很大，Weight\_decay就是L2正则化系数，用于防止过拟合的。

SGD比Adam收敛的更稳一些，Adam收敛会更加振荡，所以，优化器选择Adam的时候学习率一般会低于SGD的。

以SGD为例，有学习率、权值衰减、动量、学习率衰减这些参数可以调节。

**A.预训练模型的作用是什么？**

预训练可以不用从“零”开始调整模型参数，为模型初始化提供更优质的权重，以加速模型的收敛。

注意，这里是要在数据相似的情况下使用预训练模型可加快模型收敛速度和提高模型效果。

**A.常用的数据增强有哪些？**

数据增强的作用：

1.减缓过拟合。

2.提升模型的鲁棒性，降低模型对图像的敏感程度。

3.增加训练数据，提高模型的泛化能力。

4.可以使样本更加均衡。

数据增强的方式主要有两种：在线增强和离线增强，离线增强一般用于小数据集，在训练数据不足的时候使用，在线增强一般用于大数据集。

离线增强一般在训练前对数据集进行处理，得到更多倍的数据集，在线增强是在训练时对图像进行预处理，不改变训练数据集的数量。

常用的几何变换方法有：翻转、旋转、剪裁、缩放、平移、色彩抖动。

常用的像素变换方法有：加椒盐噪声、高斯噪声、进行高斯模糊、调整HSV对比度、调节亮度，饱和度，直方图均匀化、调整白平衡等。

此外还有比较特别的数据增强：

Cutout，对一张图像随机选取一个小正方形区域，在这个区域的像素值设置为0或其它统一的值。

Random Erasing，随机擦除掩码区域的长宽，以及区域中像素值的替代值都是随机的，Cutout是固定使用正方形，替代值都使用同一个。

Mixup，将在数据集中随机选择两张图片按照一定比例融合，包括标签值

CutMix，选择一个小区域，进行掩码，但掩码的方式却是将另一张图片的该区域覆盖到这里。

Mosaic，原理是使用四张图片拼接成一张图片。这样做的好处是图片的背景不再是单一的场景，而是在四种不同的场景下，且当使用BN时，相当于每一层同时在四张图片上进行归一化，可大大减少batch-size。

1. **训练时，影响显存占用的因素有哪些？可以从哪些方面解决？**

如果显存被占用过。多导致无法再申请多的资源，就会报错，报错内容为：CUDA error:out of memory。

影响显存占用的因素有：模型的输入大小、batch size、优化器种类。

优化器种类影响显存主要是在于优化器参数，优化器参数指的是模型在优化过程即反向传播中所产生的参数，这部分参数主要指的是梯度。在SGD中，其大小与参数一样，因此在优化期间，模型的参数所占用的显存会翻倍。不同的优化器其所需保存的优化参数不同，对于Adam，由于其还需要保存其余参数，模型的参数量会在优化区间翻 4 倍。

想减低显存占用可以减少模型的输入大小、batch size或者使用amp混合精度训练。

以及模型推理或Loss计算的中间变量越多，占用显存越多；解决：及时释放中间变量。

**A.有哪几种权重初始化的方法？**

- 权重初始化为0

如果将权重初始化全部为0的话，这样的操作等同于等价于一个线性模型，将所有权重设为0时，对于每一个w而言，损失函数的导数都是相同的，因此在随后的迭代过程中所有权重都具有相同的值，这会使得隐藏单元变得对称，并继续运行设置的n次的迭代，会导致网络中同一个神经元的不同权重都是一样的。

- 权重随机初始化

权重随机初始化是比较常见的做法，即W随机初始化。

- Xavier initialization

在使用以上两种方法来初始化权重极易出现梯度消失的问题，而Xavier initialization出现就解决了上面问题。其思想倒就是尽可能的让输入和输出服从相同的分布，这样就能够避免后面层的激活函数的输出值趋向于0。其中又有Xavier均匀分布和Xavier正态分布。

Xavier权重初始化方式比较适用于tanH和Sigmoid激活函数，而对于ReLU这种非对称性的激活函数还是容易出现梯度消失的现象。

- He initialization

He initialization是由何凯明大神提出的一种针对ReLU激活函数的初始化方法。He initialization的思想是：和Xavier初始化方式一样，都希望初始化使得正向传播时，状态值的方差保持不变，反向传播时，关于激活值的梯度的方差保持不变。由于小于0的值经过ReLU激活函数都会变成0，而大于0的值则保持原值。因此在ReLU网络中，假定每一层有一半的神经元被激活，另一半为0，所以，要保持variance不变，只需要在Xavier的基础上再除以2即可。本文主要介绍Pytorch当中He initialization均匀分布和He initialization正态分布初始化这两种方式。

- 预训练权重

目前更多的使用已经针对相似任务已经训练好的模型，称之为预训练模型。在训练开始时就已经有了非常好的初始化参数，只需要将最后的全连接层进行冻结，训练其他部分即可。

总结

1、权重采用初始化为0和随机初始化都比较容易出现梯度消失的问题，因此不常用。

2、Xavier权重初始化方式主要针对于tanH和sigmoid激活函数。

3、He initialization权重初始化方式主要针对于ReLU激活函数。

4、如果有相似任务已经训练好的模型，也可以考虑采用预训练模型来作权重初始化。

**A.在Pytorch中，数据读取数据Dataset的操作流程是怎样的？**

Pytorch 创建用以输入到模型的数据的一般流程如下：

- 创建一个 Dataset 对象，实现\_\_getitem\_\_()和\_\_len\_\_()这两个方法

- 创建一个 DataLoader 对象，该对象可以对上述Dataset对象进行迭代

- 遍历DataLoader对象，将样本和标签加载到模型中进行训练

主要使用的函数：\_\_init\_\_函数，\_\_getitem\_\_函数， \_\_len\_\_函数。

\_\_init\_\_函数：进行初始化，可以选择所要传入的参数。

\_\_getitem\_\_函数：根据索引返回对引得数据。

\_\_len\_\_函数：返回数据的长度，即样本的总数量。

**A.使用Pytorch搭建时卷积层、池化层、全连接层等算子有什么参数可以调节？**

Conv2d：

in\_channels 是输入的四维张量[N, C, H, W]中的C了，即输入张量的channels数。这个形参是确定权重等可学习参数的shape所必需的。

out\_channels即期望的四维输出张量的channels数，不再多说。

kernel\_size 代表卷积核的大小，一般我们会使用5x5、3x3这种左右两个数相同的卷积核，因此这种情况只需要写kernel\_size = 5这样的就行了。如果左右两个数不同，比如3x5的卷积核，那么写作kernel\_size = (3, 5)，注意需要写一个tuple，而不能写一个列表（list）。

stride = 1 表示卷积核在图像窗口上每次平移的间隔，即所谓的步长。这个概念和Tensorflow等其他框架没什么区别，不再多言。

padding = 0 Pytorch与Tensorflow在卷积层实现上最大的差别就在于padding上。 Padding即所谓的图像填充，后面的int型常数代表填充的多少（行数、列数），默认为0。需要注意的是这里的填充包括图像的上下左右，以padding = 1为例，若原始图像大小为32x32，那么padding后的图像大小就变成了34x34，而不是33x33。

Pytorch不同于Tensorflow的地方在于，Tensorflow提供的是padding的模式，比如same、valid，且不同模式对应了不同的输出图像尺寸计算公式。而Pytorch则需要手动输入padding的数量，当然，Pytorch这种实现好处就在于输出图像尺寸计算公式是唯一的。

大多数情况下的kernel\_size、padding左右两数均相同，且不采用空洞卷积（dilation默认为1），因此只需要记 O = （I - K + 2P）/ S +1这种在深度学习课程里学过的公式就好了。

dilation = 1 这个参数决定了是否采用空洞卷积，默认为1（不采用）。从中文上来讲，这个参数的意义从卷积核上的一个参数到另一个参数需要走过的距离，那当然默认是1了，毕竟不可能两个不同的参数占同一个地方吧（为0）。

groups = 1 决定了是否采用分组卷积，现在用的比较多的是groups = in\_channel。当groups = in\_channel时，是在做的depth-wise conv的。

bias = True 即是否要添加偏置参数作为可学习参数的一个，默认为True。

padding\_mode = ‘zeros’ 即padding的模式，默认采用零填充。

MaxPool2d()：

kernel\_size(int or tuple) - max pooling的窗口大小

stride(int or tuple, optional) - max pooling的窗口移动的步长。默认值是kernel\_size

padding(int or tuple, optional) - 输入的每一条边补充0的层数

dilation(int or tuple, optional) – 一个控制窗口中元素步幅的参数

return\_indices - 如果等于True，会返回输出最大值的序号，对于上采样操作会有帮助

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

ReLU：

inplace = True是指原地进行操作，操作完成后覆盖原来的变量。例如：

上面的代码中的意思v(x)进行relu之后赋值给x

优点：节省内存

缺点：进行梯度回归的时候传回失败，原来的变量被覆盖了，找不到原来的变量。

Linear：

in\_features – 线性变换层输入单元的数目。

out\_features– 线性变换层输出单元的数目。

**A.模型在训练和测试时有哪些差异？**

我们知道，在pytorch中，模型有两种模式可以设置，一个是train模式、另一个是eval模式。

在使用model.eval()时就是将模型切换到测试模式，在这里，模型就不会像在model.train()训练模式下一样去更新权重。但是需要注意的是model.eval()不会影响各层的梯度计算行为，即会和训练模式一样进行梯度计算和存储，只是不进行反向传播。

因此，还可以使用torch.no\_grad()。torch.no\_grad()用于停止autograd的计算，能起到加速和节省显存的作用，但是不会影响Dropout层和Batch Normalization层的行为。

如果不在意显存大小和计算时间的话，仅仅使用model.eval()已足够得到正确的validation的结果；而with torch.zero\_grad()则是更进一步加速和节省gpu空间。因为不用计算和存储梯度，从而可以计算得更快，也可以使用更大的batch来运行模型。

**A.模型在训练时和测试时，BN和Dropout有什么区别？**

训练时BN和Dropout起作用，推理时没有起作用。

在训练时，会启用 Batch Normalization 和 Dropout。在train模式，Dropout层会按照设定的参数p设置保留激活单元的概率，如keep\_prob=0.8，Batch Normalization层会继续计算数据的mean和var并进行更新。

而在测试时，一般会用model.eval()。model.eval()的作用是不启用 Batch Normalization 和 Dropout。在eval模式下，Dropout层会让所有的激活单元都通过，而Batch Normalization层会停止计算和更新mean和var，直接使用在训练阶段已经学出的mean和var值。

**A.如果一个分类模型只能进行二分类，怎么修改让它可以进行十分类？**

修改全连接层的输出就可以了，将最后的输出从2修改为10，但是要注意输入到这层全连接的输入维度要和之前的保持一致。

**A.在Pytorch中，有哪一些常用的操作、方法、成员？**

Cuda.is\_available()、ToTensor()、view、transpose、squeeze、unsqueeze，使用查看张量的shape（如a.shape）等等。

模型构建：

torch.nn.conv2d,2D卷积、torch.nn.Linear,全连接层、torch.nn.ReLU,激活函数、torch.nn.MaxPool2d,池化、torch.Dropout,随机失活、torch.nn.BatchNorm2d,BN层。

损失函数：

Torch.nn.BCELoss、torch.nn.BCEWithLogitsLoss、torch.nn.Cross\_EntropyLoss等

torch.nn.function.softmax、sigmoid、cross\_entropy等函数，torch.nn中有对应的类。

操作：

广播机制：两个形状不一致，但满足一定条件的张量进行加减乘除

torch.tensor()张量声明、torch.from\_numpy()、torch.max()、torch.maximum()、torch.argmax()、torch.sort()、torch.argsort()、torch.sum()、torch.mean()、torch.cat()、 torch.stack()、torch.load()、torch.save()等

方法：

tensor.item() tensor.data()、tensor.to()、torch.transport()、torch.permute()、torch.squeeze()、torch.unsqueeze()、torch.clip()等

成员：

tensor.shape、tensor.size、tensor.stride、tensor.T等