**深度学习基本问题：**

1. **什么是深度学习？什么是迁移学习？什么是强化学习？**

机器学习：定义一系列函数 =>通过训练集筛选出效果好的函数 =>选出效果最好的函数

深度学习：定义网络结构 =>通过训练集筛选出效果好的函数 =>选出效果最好的函数

深度学习实际上是学习了样本分布的规律，所以要求测试集和训练集有同分布

迁移学习：就是把已学训练好的模型参数迁移到新的模型来帮助新模型训练。考虑到大部分数据或任务是存在相关性的，所以通过迁移学习我们可以将已经学到的模型参数通过某种方式来分享给新模型从而加快并优化模型的学习效率不用像大多数网络那样从零学习

强化学习：<https://blog.csdn.net/coffee_cream/article/details/57085729>

1. **CNN、RNN、LSTM？**

CNN：卷积神经网络，用于图像处理

RNN：循环神经网络，具有记忆的神经网络。隐层的输出被存储到记忆里，记忆可以作为另一个输入

LSTM：Long Short-term Memory具有特别的神经元，4个输入，1个输出



1. **Softmax函数作为输出层的用法？**

**https://blog.csdn.net/qq\_33666011/article/details/79572051**

1. **深度学习的学习目标？**

最小化所有训练样本的总体损失，得到模型的参数（权值和偏置）

1. **相同参数数量时，宽网络和深度网络哪个好？为什么？**

多层的网络结构表达一些函数时更加简单

1. **深度网络中的Modularization？**

每个模块只解决一个基本问题，并作为模块被后面的分类器继承，这样每个基本分类器都有足够的训练样本，模型可以由少量的数据进行训练

1. **怎么做可以提高使模型在训练数据集上取得好的结果？5个要点**

合适的损失函数、mini-batch、激活函数、自适应的学习率、momentum

1. **为什么softmax作为输出层时，使用交叉熵损失函数比平方损失函数效果好？**

平方损失函数求导时候回陷入饱和区，距离期望值越远，参数更新速度越慢；

交叉熵损失函数不会受到饱和性的影响，而且误差越大，参数更新越快

<https://blog.csdn.net/u014595019/article/details/52562159>

1. **交叉熵损失函数及求导？**

交叉熵损失函数的求导：

<https://blog.csdn.net/jasonzzj/article/details/52017438>

1. **Mini-batch是怎么做的？为什么效果更好？**

训练时并不是最小化所用样本的总损失，而是把总样本随机分成若干个batch，用每个batch中的样本训练网络、更新参数。所有的样本都用过一遍，称为一个epoch，每个epoch后对样本进行一次shuffle，在进行下一个epoch

效果更好的原因：每个epoch中进行了多次的更新参数，训练更快；每次训练不同的样本，避免学习到样本之间的相关性，从而发生过拟合

但是，mini-batch更新参数的过程是不稳定的，因为一个batch中的样本可能和总体的分布不同

1. **为什么网络不是越深越好？**

反向传播过程中，靠近输出的层有很大的梯度、学习的速度快；而前面的层梯度小，学习速度很慢（梯度消失）

1. **激活函数都有哪些？**

激活函数的主要作用：提供网络的非线性建模能力，如没有，则只能表达线性映射

类型：分段线性和具有指数形状的非线性函数

Sigmiod、tanh、ReLU、ELU、Maxout

<https://blog.csdn.net/u014595019/article/details/52562159>

1. **激活函数应具备的性质：**

可微性：当优化方法是基于梯度的时候，这个性质是必须的

单调性：当激活函数是单调的时候，单层网络能够保证是凸函数

输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著;当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的learning rate

1. **Sigmoid函数的优缺点？**

缺：a. 软饱和性，梯度消失

b. 偏移现象：输出均大于0，不是0均值

优：(1) 具有指数函数形状，在物理意义上最接近生物神经元

(2) (0,1)的输出可以被表示作概率，或用于输入的归一化

1. **ReLU为什么效果好？有什么优缺点？**

优：X>0时能够保持梯度不衰减，从而缓解梯度消失问题

缺：神经元死亡、偏移现象

1. **参数优化学习算法都有哪些？**

SGD算法的缺点：

1. 很难选择合适的学习率 (2) 所有的参数学习率相同 (3) 可能会陷入局部极小点

优化算法学习率的设定思想：

1. 每隔若干个epoch减小学习率 (2) 为不同的参数设置不同的学习率

优化算法：

<https://blog.csdn.net/muyu709287760/article/details/62531509>

1. **为什么要更新学习率？怎么更新？为什么Adagrad算法中梯度越大，学习率越小？**
2. 学习率太大，可能会跳过全局最小；学习率太小，模型训练太慢
3. 开始时，距离目标较远，学习率设置大一点；后面每隔几个epoch减小学习率
4. 梯度大，如学习率大，则容易错过最小点（回忆等高线图）
5. **怎么理解Momentum？**

利用了物理学中动量的思想，通过积累之前的动量(mt−1)来加速当前的梯度。

mt=μ∗mt−1+η∇θJ(θ) θt=θt−1−mt

其中，μ是动量因子，通常被设置为0.9或近似值。

特点：

1. 参数下降初期，加上前一次参数更新值；如果前后2次下降方向一致，乘上较大的μ能够很好的加速。
2. 参数下降中后期，在局部最小值附近来回震荡时，gradient→0gradient→0，μμ使得更新幅度增大，跳出陷阱。
3. 在梯度方向改变时，momentum能够降低参数更新速度，从而减少震荡；在梯度方向相同时，momentum可以加速参数更新， 从而加速收敛。

总而言之，momentum能够加速SGD收敛，抑制震荡。

1. **怎样在测试集上获得好的结果？4个要点**

早停、正则化、Dropout、网络结构

1. **为什么会过拟合？怎么防止？Data augmentation的方法？**
2. 训练数据与测试数据可能不同，而学习目标是由训练数据确定的
3. 数据增强、早停(验证集)、weight decay、dropout、网络结构
4. flip、filter、crop、色彩<https://blog.csdn.net/Yaphat/article/details/54098867>
5. **神经网络中的正则化是怎么实现的？**

Weight Decay：减掉神经元之间的无用的连接

Weight Decay对应着L2正则化

<http://blog.sina.com.cn/s/blog_a89e19440102x1el.html>

1. **Dropout是怎么实现的？反向传播时一样吗？**

通过改变网络本身的结构来实现的

在训练开始时，随机删除一些隐藏层神经元，同时保持输入层与输出层神经元的个数不变，反向传播时只更新剩余神经元之间的权重

为什么减轻过拟合？

随着神经网络模型不断地学习，神经元的权值会与整个网络的上下文相匹配。神经元的权重针对某些特征进行调优，具有一些特殊化。周围的神经元则会依赖于这种特殊化，如果过于特殊化，模型会因为对训练数据过拟合而变得脆弱不堪。

1. **为什么CNN适合图像处理？3个特点**
2. 一些特征比整张图片小很多，不必全览整张图片来发现特征，可以用少量的参数连接小区域；
3. 相同的特征会出现在不同的区域，可以用相同的参数来做同样的检测；
4. 下采样像素不会改变目标，片可以用少量的参数来处理

前两个特点使得我们可以使用卷积操作，第三个特点对应Max Pooling

1. **在全连接之前Feature Map做了什么处理？**

Flatten 把多维矩阵展开为一维线性向量，以保证全连接的实现

1. **图像尺寸n、kernel尺寸k、步长stride、zero pad值、feature map尺寸的关系？**

F = (n – k + 2\*pad) / stride + 1

1. **CNN网络最重要的特点？**

参数共享

**卷积神经网络CNN模型：**

1. **AlexNet网络创新点？2012年8层**

ReLU

Dropout

Data Augmentation

重叠Pooling (提高精度，减少过拟合)

LRN局部相应归一化

1. **ZFNet有什么贡献，网络结构与AlexNet有何改进？2013年**

贡献：通过反卷积和反池化来可视化AlexNet

改进：改变了第一层filter的大小，3、4层稀疏连接变成了稠密连接

发现：

(1) 层数越高，提取的特征越抽象

(2) 一定次数迭代之后，底层特征趋于稳定，但更深层的特征则需要更多次迭代

(3) 特征不变形 => 层数越低，对输入的微小变化越敏感，层数越高，影响越小

底层 => 用于判别物体共有部分 深层 => 用于判别不同物体独有特征

1. **VGG的创新点？有何缺点？为什么用3\*3的卷积核？1\*1卷积的作用？2014年**

创新点：

1. 用多个3\*3的卷积代替大的卷积操作 => 参数更少、模型更深、线性表达能力增强
2. 在整张图片和multi-scales上进行训练和测试(256, 384)
3. 采用了mini-batch梯度下降的算法

特点：卷积核尺寸均为3\*3，pooling尺寸均为2\*2 (没有重叠池化)

缺点：计算量大、参数量大、内存占用大（反向传播时中间层占用大量内存）

3\*3卷积核：是能够表示上下、左右、中心这样概念的最小单元

1\*1卷积核：降维、相当于空间线性映射，后面接ReLU可以增强模型的非线性表达能力

1. **GoogLeNet创新点？2014年**
2. 采用了模块化结构，方便添加了修改Inception结构，增加网络深度和宽度的同时，减小参数
3. 添加了两个辅助分类器 => 为反向传播过程提高梯度，增强正则化作用
4. Global Average Pooling代替了全连接层

1\*1卷积核：降维、低层时位置信息比较准确，1\*1的channel数多

3\*3、5\*5卷积核：考虑的是空间尺度适应性，深层时channel多

1. **Inception v1 v2 v3 v4结构分别有什么改进？Inception假设？**

V1：构建了1\*1、3\*3、5\*5的conv和3\*3的pooling的分支网络，使用了全局平均池化代替全连接层，增加了卷积层网络的宽度，增加了网络对尺度的适应性

V2：(1) 提出了Batch Normalization，可以代替Dropout和LRN，起到正则化作用

(2) 用两个3\*3卷积代替了5\*5的卷积，降低参数量的同时提高了网络的学习能力

V3：引入了因式分解思想，把n\*n的卷积分为1\*n和n\*1的卷积，节约参数，减轻过拟合

同时，增加了一层非线性，扩展了模型的表达能力，并且在分支中使用了分支

V4：结合了残差连接的思想

Inception假设：把跨通道相关性和空间相关性分解成一系列相互独立的操作

1. **Xception的创新点？假设？Inception的极限版本与Xception有何不同？**

Xception是对Inception v3的一种改进，采用深度可分卷积来代替原来的卷积操作。

Xception结构和Inception v3有相同数量的参数，性能的提升来自于更有效的使用模型

Xception假设：跨通道相关性和空间相关性可以完全脱钩

特点：(1) 没有大大降低网络的复杂度，因为加宽了网络

(2) 目的在于提升网络性能，不是压缩模型

创新点：(1) depthwise separable convolutions (深度可分离卷积结构) （https://www.cnblogs.com/adong7639/p/7918527.html ）(2) 模块间残差连接

不同点：(1) 操作顺序：前者先进行1\*1卷积，后者先进行channel-wise卷积

(2) 非线性激活：前者每个操作后都有ReLU，后者中间过程不使用，最后使用

1. **Batch Normalization？**

目标：解决internal covariate shift（<https://www.cnblogs.com/bonelee/p/8528722.html>）问题，网络参数的不断改变导致每一层的输入分布也发生变化，而学习的过程又要使每一层适应输入的分布，导致模训练困难

作用：加速训练收敛速度，可以起到一定的正则化作用。可以代替Dropout和LRN层

把越来越偏的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数

对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，意思是

这样让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，

能大大加快训练速度。

位置：原文说应该把BN放在激活函数之前，因为Wx+b具有更加一致和非稀疏的分布。

但是也有人做实验表明放在激活函数后面效果更好

如何实现的：

1. 某个层的输出限制在均值为0方差为1的分布会使得网络的表达能力变弱。相当于只做了线性变换
2. 因此给batch normalization层进行一些限制的放松，给它增加两个可学习的参数 β 和 γ，对数据进行缩放和平移，平移参数 β 和缩放参数 γ 是学习出来的。

即每个BN层的γ和β参数都有不同的取值，服从不同的正态分布？

问题：训练时使用的是mini-batch，可以计算均值和方差，但是测试时只有一个数据，BN层怎么做的？

解决方法1：使用所有训练样本的均值和方差(原文的做法)

解决方法2：基于momentum的指数衰减

running\_mean = momentum \* running\_mean + (1 - momentum) \* sample\_mean

running\_var = momentum \* running\_var + (1 - momentum) \* sample\_var



1. **ResNet的创新点？解决的问题？存在的问题？残差块的结构？2016年**

创新点：residual block (shortcut connection)、Bottleneck

解决的问题：深度网络训练困难(1) 梯度消失 (2) 准确率饱和

为什么收敛速度快：H(x) = F(x) + x 反向传播时导数在1附近，训练快

特点：(1) 网络看似很深，但是实际起作用的网络层数并不深

(2) 残差网络是由多种路径组合的一个网络，相当于多人投票系统

(3) 大多数梯度集中在中间路径上（effective path）

(4) 复杂度并没有提高（低于VGG），计算量小了很多，全局平均池化代替FC

(5) 训练时用BN层，没用Dropout

1. **Inception系列网络存在的问题？ResNeXt的思想、核心创新点、优点？block设计规则？**

问题：网络的超参数针对性强，应用到其他数据集时需要修改许多参数，可扩展性一般

思想：结合了VGG堆叠思想、ResNet模块和Inception的split-transform-merge思想

核心创新点：对ResNet Block改进，聚合残差转化，新的block结构 (group conv)

优点：在不明显增加参数量级的情况下提升了准确率，减少了超参数，便于移植

设计原则：(1) 若block的输出空间尺寸一样，则拥有相同的超参数(卷积核尺寸和数量)

(2) feature map的分辨率缩减一半，通道数增加一倍(每个block计算复杂度一致)

Group Conv的优点：(1) 全连接是考虑所有的信息

(2) 普通卷积是在考量空间维度上局部区域的所有通道中的信息

(3) Group Conv是在通道维度上做局部化的考量

(4) Depthwise Separable Conv 是将空间维度和通道维度分开考虑

逐步减少了模型参数和计算量

1. **MobileNet模型核心？宽度乘数、分辨率乘数？计算量的计算方法？**

核心：深度可分卷积、模型压缩

1. **MobileNet提出原因、基本要求、应用领域**

提出原因：针对手机等嵌入式设备，需要轻量级的深层网络

基本要求：轻量级、参数少、低延迟、精度不差

应用领域：

1. **SqueezeNet设计目标？创新点？设计原则？Fire Module解析？**

设计目标：简化网络复杂度、降低模型参数

创新点：(1) fire module 包括1\*1的squeeze和1\*1+3\*3的expand

(2) 使用Global Average Pooling代替了FC

设计原则：a. 使用1\*1卷积代替部分3\*3卷积(减少参数)

b. 减少输入3\*3卷积的通道数

c. 欠采样操作延后(卷积层提供更大的激活图，提高准确率)

fire之间有残差连接

1. **ShuffleNet创新点？channel shuffle解决的问题？group操作？边界效应？**

针对的问题：Xception和ResNext中point conv仍有很大的**计算量**，设计是以**ResNet**的Bottleneck单元为基础进行改进的，有残差连接

创新点：用ShuffleNet Unit代替Residual block

(1) channel shullfe解决多个group conv叠加出现的边界效应

(2) pointwise group conv

(3) depthwise separable conv

（<http://m.jrj.com.cn/rss/xiaozhi/2017/8/25/23001554.shtml>）

边界效应：相邻的层都有group操作，造成输出的某个channel仅仅来自于输入channel的一部分

ShuffleNet与MobileNet相比：用channel shuffle代替了1\*1卷积(效率大大提升)

MobileNet基于VGG而ShuffleNet基于ResNet

1. **Pointwise group convolutions？ShuffleNet unit 和 Residual block的不同之处？**

Pointwise group conv：对Bottleneck单元中的1\*1卷积做group操作

不同之处：(1) 1\*1卷积变成了1\*1 group conv

(2) 3\*3卷积变成了DWConv (没有接下来的point conv)

(3) 上述两者之间添加了channel shuffle

(4) shortcut connection上加了AVG Pooling (降低了分辨率)

(5) 原来的add操作变成了concat (弥补了上一步降低的分辨率)

1. **输入大小为c\*h\*w，bottleneck的channel数为m，分别求ResNet、ResNeXt、ShuffleNet单元结构的计算量？**

ResNet： h\*w\* (c\*1\*1\*m + m\*3\*3\*m + m\*1\*1\*c) = h\*w\*(2c\*m + 9m\*m)

ResNext： h\*w\* [c\*1\*1\*m + (m/g\*3\*3\*m/g)\*g+ m\*1\*1\*c] = h\*w\*(2c\*m + 9m\*m/g)

ShuffleNet：

h\*w\* [(c/g\*1\*1\*m/g) \*g+ (1\*3\*3\*m/g)\*g + (m/g\*1\*1\*c/g)\*g] = h\*w\*(2c\*m/g + 9m)

1. **MobileNet v2创新点？(MobileNet v2和Xception的区别)**

可以看做是ResNet + MobileNet的网络(残差连接、深度可分卷积、反BottleNeck结构)

在ResNet的Bottleneck模块基础上做的修改

创新点：(1) Linear Bottlenecks 在后面的1\*1卷积之后不加ReLU层(重点理解这一点)

(2) Inverted residuals 先升维再降维

<https://blog.csdn.net/dlyldxwl/article/details/79101293>

升维、降维的作用是什么呢？

(1) 比如dw conv layer的输入是128个channel，mobilenet就直接用128个filter去卷积，这必然会导致信息流通不好（毕竟这是channel和spital的完全解耦），但是V2却先把128维升为128×6，然后再进行dw layer，这分明就是冗余的思想嘛！一对一可能效果不好，但是我6对1，总有好的吧！

(2) 再用1×1的 layer 去降维，这个layer的作用和一般的residual block可不一样了！在传统的residual block中，1×1的作用是用来降低计算量的，此处的作用确是让网络自己去找到起主要作用的channel！这比剪枝等操作有用多了！

1. **模型压缩网络？模型压缩方法？**

MobileNet、SqueezeNet、ShuffleNet、MobileNet v2

模型压缩方法：数据类型有float32改为int8

**目标检测问题：**

1. **RCNN实现的步骤？有哪些缺点？**

(1) 候选框生成 (选择性搜索)

(2) 特征提取 (每个候选区resize传入CNN网络)

(3) 类别判断 (每类目标用一个SVM二分类器)

(4) 位置精修 (输出为xy方向的缩放与平移)

缺点：训练、测试速度慢(候选框大量重叠、特征提取过程冗杂)，训练所需空间大(需要大量训练样本)

1. **SPP Net的特点？**

特点：(1) 输入尺寸可以使任意的，输出尺寸是固定的

(2) 只对原图进行一次特征提取，找到每个候选框在feature map上的映射

解决的问题：resize操作会造成数据丢失或失真

1. **Fast RCNN实现的步骤？有哪些改进？ROI Pooling的作用？**

步骤：(1) 候选框生成 选择性搜索

(2) 整张图输入CNN网络，得到feature map

(3) 找到候选框在feature map上的映射，传入POI Pooling及以后的层

(4) softmax多分类、回归器精修位置

改进：只需要一次特征提取、softmax多分类、ROI Pooling使图片不必提前resize

特点：分类和回归任务共享卷积层，合并成一个multi-task模型

缺点：选择性搜索的缺点

损失函数：(1) 分类：Softmax with loss

(2) 回归：Smooth L1 loss

ROI Pooling作用：把大小不同的输入映射到一个固定尺寸的特征向量，将每个候选框均匀分成M\*N块，每块进行Max Pooling

1. **ROI Pooling存在的问题、改进方法？**

问题：分成M\*N块时，可能不是整除，导致边缘区域不能保持等变性

改进：ROI warp、ROI align、Position Sensitive ROI Pooling

1. **Faster RCNN实现的步骤？有哪些改进？图解Faster RCNN的流程？**

步骤：(1) 整张图片输入CNN网络，得到feature map

(2) feature map输入到RPN网络进行候选框选择

(3) 在RPN中对符合要求的的候选框进行softmax二分类以及位置回归

(4) 将符合要求的结果输入到ROI Pooling层，进行softmax多分类和位置精修

改进：用RPN网络进行候选框选择，有两次分类和回归

1. **训练集区域选择的规则？怎么给anchor分配的标签？RPN网络结构？**

规则：与GT的IOU最大的以及IOU>0.7的标记为正类；IOU<0.3的为背景类

1. **怎么减少候选框的数量的？什么是非极大值抑制？**

训练时，忽略所有跨边界的anchor，此时剩下约6k个，非极大值抑制后剩2k个

测试时，跨边界的框裁剪到图像边缘位置

非极大值抑制：将候选框按照分类器的得分进行排序，从最大概率的框开始，分别判断其

(1) 从最大概率矩形框F开始，分别判断A~E与F的重叠度IOU是否大于某个设定的阈值；(常用的阈值为0.3-0.5)

(2) 假设B、D与F的重叠度超过阈值，那么就扔掉B、D；并标记第一个矩形框F，是我们保留下来的。

(3) 从剩下的矩形框A、C、E中，选择概率最大的E，然后判断E与A、C的重叠度，重叠度大于一定的阈值，那么就扔掉；并标记E是我们保留下来的第二个矩形框。

就这样一直重复，找到所有被保留下来的矩形框。

1. **SSD的创新点？图解？模型结构？模型训练时与上述方法的区别？**

解决的问题：Faster RCNN速度慢，SSD可以提高速度，同时准确率也很高

创新点：(1) 在feature map上使用小的卷积核预测bounding box的分类得分和偏移量

(2) 在base network之后添加了额外的卷积层，可以多尺度下进行预测

(3) Default boxes和aspect ratio

与Faster RCNN区别：SSD是在不同的特征层中考虑不同尺度

Faster RCNN是在一个特征层中考虑不同的尺度

提速原因：相对于object proposals的检测模型，SSD完全取消了proposals generation、pixel resampling或feature resampling这些阶段，这使得SSD更容易优化。

训练过程遇到的问题：default boxes的选择、ratio的设置、hard negative mining、数据增强

Hard Negative Mining：训练时，正负样本不均衡，按置信度排序，选择最高的那部分，并

且保证负、正样本的比例为3:1

数据增强的策略：

1. **GT和DBB的匹配策略？正负样本不均衡？Default box的尺寸是怎么确定的、怎么理解？**
2. **FPN创新点？**
3. **RefineDet创新点？**

**深度学习面试问题：**

1、**数据增强怎么做？**

**旋转 | 反射变换(Rotation/reflection):** 随机旋转图像一定角度; 改变图像内容的朝向;

**翻转变换(flip):** 沿着水平或者垂直方向翻转图像;

**缩放变换(zoom):** 按照一定的比例放大或者缩小图像;

**平移变换(shift):** 在图像平面上对图像以一定方式进行平移; 可以采用随机或人为定义的方式指定平移范围和平移步长, 沿水平或竖直方向进行平移. 改变图像内容的位置;

**尺度变换(scale):** 对图像按照指定的尺度因子, 进行放大或缩小; 或者参照SIFT特征提取思想, 利用指定的尺度因子对图像滤波构造尺度空间. 改变图像内容的大小或模糊程度;

**对比度变换(contrast):** 在图像的HSV颜色空间，改变饱和度S和V亮度分量，保持色调H不变. 对每个像素的S和V分量进行指数运算(指数因子在0.25到4之间), 增加光照变化;

**噪声扰动(noise):** 对图像的每个像素RGB进行随机扰动, 常用的噪声模式是椒盐噪声和高斯噪声;

颜色变换(color): 在训练集像素值的RGB颜色空间进行PCA, 得到RGB空间的3个主方向向量,3个特征值, p1, p2, p3, λ1, λ2, λ3.

**2、SIFT特征提取？**

尺度不变特征转换即SIFT (Scale-invariant feature transform)是一种计算机视觉的算法。它用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量。

SIFT特征是基于物体上的一些局部外观的兴趣点而与影像的大小和旋转无关，SIFT算法的实质是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。SIFT所查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

**3、学习率怎么调整，一般你设置多少？**

学习率设置过小，收敛速度会非常慢，学习率设置过大，则会越过最低点，无法达到最低点。因此选择lr，也就是不断试的过程，基本范围大概就是0.1,0.01,0.001,0.0001这样子，一个数量级一个数量级的尝试就可以了。一般设置0.01或0.001。

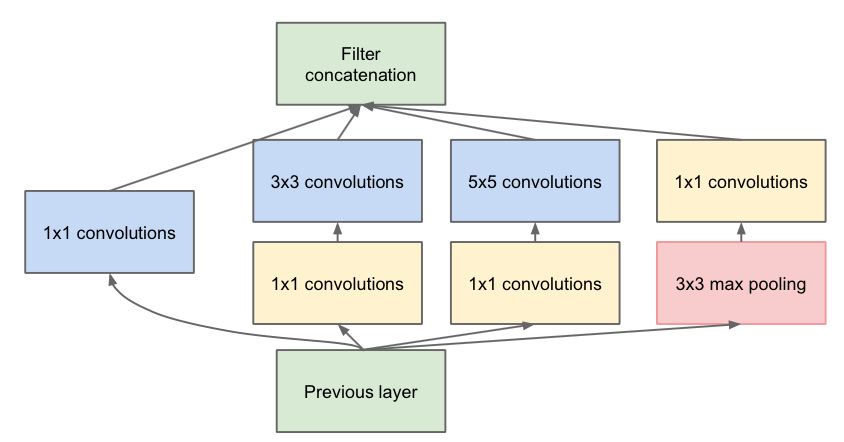
**4、Inception结构能不能缓解梯度消失？**

可以！

既能达到稀疏的减少参数的效果，又能利用硬件中密集矩阵优化的东风。

将卷积分块，所谓的分块就是其实就是将卷积核分组，既然是分组索性就让卷积和不一样吧，索性使用了1×1，3×3，5×5的卷积核，又因为pooling也是CNN成功的原因之一，所以把pooling也算到了里面，然后将结果在拼起来。这就是最naive版本的Inception。

这个naive版的Inception，还有一个问题，因为所有的卷积核都在上一层的所有输出上来做，那5×5的卷积核所需的计算量就太大了。因而，可以采用NIN中的方法对上一层的输出进行Merge。这样就衍生出了真正可用的Inception。



这个结构利用了NIN结构中非线性变换的强大表达能力。

**5、迁移学习加L2正则化的好处是什么？**

1. 正则化的目的：防止过拟合！

2. 正则化的本质：约束（限制）要优化的参数。

关于第1点，过拟合指的是给定一堆数据，这堆数据带有噪声，利用模型去拟合这堆数据，可能会把噪声数据也给拟合了，这点很致命，一方面会造成模型比较复杂（想想看，本来一次函数能够拟合的数据，现在由于数据带有噪声，导致要用五次函数来拟合，多复杂！），另一方面，模型的泛化性能太差了（本来是一次函数生成的数据，结果由于噪声的干扰，得到的模型是五次的），遇到了新的数据让你测试，你所得到的过拟合的模型，正确率是很差的。

关于第2点，本来解空间是全部区域，但通过正则化添加了一些约束，使得解空间变小了，甚至在个别正则化方式下，解变得稀疏了。这一点不得不提到一个图，相信我们都经常看到这个图，但貌似还没有一个特别清晰的解释

https://blog.csdn.net/wsj998689aa/article/details/39547771

**6、介绍一下YOLO v3？**

https://www.cnblogs.com/makefile/p/YOLOv3.html

7、triplet loss不易收敛怎么训练？

在有监督的机器学习领域，通常有固定的类别，这时就可以使用基于softmax的交叉熵损失函数进行训练。但有时，类别是一个变量，此时使用triplet loss就能解决问题。在人脸识别，Quora question pair任务中，triplet loss的优势在于细节区分，即当两个输入相似时，triplet loss能够更好地对细节进行建模，相当于加入了两个输入差异性差异的度量，学习到输入的更好表示，从而在上述两个任务中有出色的表现。当然，triplet loss的缺点在于其收敛速度慢，有时不收敛。  
链接：https://www.jianshu.com/p/d41b6447743d  
https://www.cnblogs.com/Alex0111/p/8492471.html

8、loss出现nan怎么办？

一、出现梯度爆炸了

1、数据归一化（减均值，除方差，或者加入normalization，例如BN、L2 norm等）；  
2、更换参数初始化方法（对于CNN，一般用xavier或者msra的初始化方法）；  
3、减小学习率、减小batch size；  
4、加入gradient clipping；

二、大致的解决办法就是，在出现Nan值的loss中一般是使用的TensorFlow的log函数，然后计算得到的Nan，一般是输入的值中出现了负数值或者0值，在TensorFlow的官网上的教程中，使用其调试器调试Nan值的出现，也是查到了计算log的传参为0；而解决的办法也很简单，假设传参给log的参数为y，那么在调用log前，进行一次数值剪切，修改调用如下：

loss = tf.log(tf.clip\_by\_value(y,1e-8,1.0))

这样，y的最小值为0的情况就被替换成了一个极小值，1e-8，这样就不会出现Nan值了，StackOverflow上也给出了相同的解决方案。于是，我就采用了上述的解决方案对于log的参数进行数值限制，但是我更加复杂化了这个限制。

<https://blog.csdn.net/qq_32458499/article/details/79468426>

9、模型压缩你是怎么做的？

更精细模型的设计，目前的很多网络都具有模块化的设计，在深度和宽度上都很大，这也造成了参数的冗余很多，因此有很多关于模型设计的研究，如SqueezeNet、MobileNet等，使用更加细致、高效的模型设计，能够很大程度的减少模型尺寸，并且也具有不错的性能。

模型裁剪，结构复杂的网络具有非常好的性能，其参数也存在冗余，因此对于已训练好的模型网络，可以寻找一种有效的评判手段，将不重要的connection或者filter进行裁剪来减少模型的冗余。

核的稀疏化，在训练过程中，对权重的更新进行诱导，使其更加稀疏，对于稀疏矩阵，可以使用更加紧致的存储方式，如CSC，但是使用稀疏矩阵操作在硬件平台上运算效率不高，容易受到带宽的影响，因此加速并不明显。

除此之外，量化、Low-rank分解、迁移学习等方法也有很多研究，并在模型压缩中起到了非常好的效果。

原文：https://blog.csdn.net/wspba/article/details/75671573

10、你认为模型提速最有效的方法是？

https://blog.csdn.net/QcloudCommunity/article/details/77719498

项目：<https://blog.csdn.net/tsyccnh/article/details/78889838>

11、准确率、精确率和召回率

假设我们手上有60个正样本，40个负样本，我们要找出所有的正样本，系统查找出50个，其中只有40个是真正的正样本，计算上述各指标。

TP: 将正类预测为正类数 40

FN: 将正类预测为负类数 20

FP: 将负类预测为正类数 10

TN: 将负类预测为负类数 30

准确率(accuracy) = 预测对的/所有 = (TP+TN)/(TP+FN+FP+TN) = 70%

精确率(precision) = TP/(TP+FP) = 80%

召回率(recall) = TP/(TP+FN) = 2/3

13、训练集、验证集和测试集

把数据集随机分为训练集，验证集和测试集，然后用训练集训练模型，用验证集验证模型，根据情况不断调整模型，选择出其中最好的模型，再用训练集和验证集数据训练出一个最终的模型，最后用测试集评估最终的模型

14、深度学习训练的技巧

使用一个小数据集可以使训练测试循环快速，因此我们可以快速地进行实验。其次，它生成的模型精度低于使用所有数据。这种低精度通常不是主要问题，因为您可以使用从较小数据集子集中获得的知识对整个数据集进行重新训练。在训练深度学习模型时，这是一个非常有用的技巧，因为在许多情况下，训练数据的数量是巨大的

15、Python中classmethod修饰符

classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。

16、Python中\*和\*\*的区别

Python中，（\*）会把接收到的参数形成一个元组，而（\*\*）则会把接收到的参数存入一个字典

17、swish激活函数：

Swish函数先对来说是比较新的一些激活函数，算是由之前的激活函数复合而成出来的。也是由Google提出的，毕竟资力雄厚，承担的起搜索的任务。而且这个算法感觉曝光率还算比较高，就在这里整理一下，同时后面的文章也会再次提到这个函数。

  对前面的激活函数有了一定的基础之后，理解Swish激活就容易很多了，Swish函数的表达式是f(x)=x⋅σ(x)f(x)=x⋅σ(x)，σ(x)σ(x)就是sigmoid函数。因为sigmoid函数的饱和性容易导致梯度消失，借鉴ReLU的效果，当xx非常大的时候，这个时候有f(x)f(x)趋近于xx，但是当x→−∞，则f(x)→0x→−∞，则f(x)→0，函数的大致走势和ReLU比较相似，但是又比ReLU复杂。

18、geru激活函数：

gelu（gaussian error linear units）就是我们常说的高斯误差线性单元，它是一种高性能的神经网络激活函数，因为gelu的非线性变化是一种符合预期的随机正则变换方式，公式如下：xP(X≤x)=xΦ(x)(2.1)xP(X≤x)=xΦ(x)(2.1)其中Φ(x)Φ(x)指的是xx的高斯正态分布的累积分布，完整形式如下：xP(X≤x)=x∫x−∞e−(X−μ)22σ22π√σdX(2.2)xP(X≤x)=x∫−∞x​2π

​σe−2σ2(X−μ)2​​dX(2.2)计算结果约为：0.5x(1+tanh[2π−−√(x+0.044715x3)])(2.3)0.5x(1+tanh[π2​(x+0.044715x3)])(2.3)或者可以表示为：xσ(1.702x)(2.4)xσ(1.702x)(2.4)由此可知，概率P(X≤x)P(X≤x)（xx可看成当前神经元的激活值输入）,即XX的高斯正态分布ϕ(X)ϕ(X)的累积分布Φ(x)Φ(x)是随着xx的变化而变化的，当xx增大，Φ(x)Φ(x)增大，当x减小，Φ(x)Φ(x)减小，即当xx越小，在当前激活函数激活的情况下，越有可能激活结果为0，即此时神经元被dropout，而当xx越大越有可能被保留