1. sigmoid \tanh\telu

sigmoid = 1/1+e(-x)

缺点：容易出现过拟合，不适合训练很深的网络中，最大导数为0.25

函数输出不是zero-centered

幂运算相对来讲比较耗时

函数输出不是zero-centered：Sigmoid函数的输出值恒大于0，这会导致模型训练的收敛速度变慢。举例来讲，对，如果所有均为正数或负数，那么其对的导数总是正数或负数，这会导致如下图红色箭头所示的阶梯式更新，这显然并非一个好的优化路径。深度学习往往需要大量时间来处理大量数据，模型的收敛速度是尤为重要的。所以，总体上来讲，训练深度学习网络尽量使用zero-centered数据 (可以经过数据预处理实现) 和zero-centered输出

Tanh = e(x)-e(-x)/e(x)+e(-x)

最大导数为1，解决了zero-centered的输出问题，经过原点，还是会梯度消失

Relu = x if x >0 else 0 或者 max(0, x)

优点：

解决了gradient vanishing问题 (在正区间)

计算速度非常快，只需要判断输入是否大于0

收敛速度远快于sigmoid和tanh，导数为1（正区间）

缺点：

ReLU的输出不是zero-centered

Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。（当神经元的输出为0时，对应的w权重将不会更新）

有两个主要原因可能导致这种情况产生: (1) 非常不幸的参数初始化，这种情况比较少见 (2) learning rate太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决方法是可以采用Xavier初始化方法（该方法的作者文章主要的目标就是使得每一层输出的方差应该尽量相等，他认为优秀的初始化应该使得各层的激活值和状态梯度的方差在传播过程中的方差保持一致），以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。

LeakyRelu = max(0.01x, x)

人们为了解决Dead ReLU Problem，提出了将ReLU的前半段设为而非0。另外一种直观的想法是基于参数的方法，即Parametric ReLU:，其中可由back propagation学出来。理论上来讲，Leaky ReLU有ReLU的所有优点，外加不会有Dead ReLU问题，但是在实际操作当中，并没有完全证明Leaky ReLU总是好于ReLU。

1. 常用的优化算法

Stochastic Gradient Descent (type: "SGD"),

SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

三种Sgd比较：

随机梯度下降：SGD 因为更新比较频繁，会造成 cost function 有严重的震荡；缺点是SGD的噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。所以虽然训练速度快，但是准确度下降，并不是全局最优。

BatchSGD：介于两者之间，收敛更稳定。缺点是需要通过调整学习率来达到最优解，不好调。

All GD: 由于这种方法是在一次更新中，就对整个数据集计算梯度，所以计算起来非常慢，遇到很大量的数据集也会非常棘手，而且不能投入新数据实时更新模型. 对于凸函数可以收敛到全局极小值，对于非凸函数可以收敛到局部极小值。

一般样本小于2000时采用allGD,大于2000时采用mini batch

随机梯度下降，一般指minibatchSGD

AdaDelta (type: "AdaDelta"),

自适应梯度，如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

Adaptive Gradient (type: "AdaGrad"),

Adam (type: "Adam"),

Nesterov’s Accelerated Gradient (type: "Nesterov") and

RMSprop (type: "RMSProp")

常用优化方法更新公式：



|  |  |
| --- | --- |
|  |  |
|  | |

Caffe里面的超参数解释：

net: "data/person\_attri/model/multi\_label\_google.prototxt"

test\_iter: 250 #完成一次测试需要的迭代次数

test\_interval: 500 #测试间隔

base\_lr: 0.01

lr\_policy: "step" #学习率变化规律，比如每迭代stepsize次，学习率为gamma\*lr

gamma: 0.5 #学习率变化指数

stepsize: 5000 #学习率变化频率

display: 500

max\_iter: 100000

momentum: 0.9 #动量 v = momentum\*v-a\*dx x = x + v

weight\_decay: 0.0005 #权重衰减,正则化惩罚项

snapshot: 10000

snapshot\_prefix: "data/person\_attri/model"

solver\_mode: GPU

1. inception之间的升级<https://www.cnblogs.com/bonelee/p/8977912.html>

v1->v4:网络深度不断加深

v1：

增加了网络的宽度，基础结构包括1\*1 3\*3 5\*5

使用了1\*1降维

使用了三个辅助分类

采用了global avg pool代替全连接，保持了空间信息，降低参数

V2:

采用了BN（什么是bn）

使用了两个3\*3代替5\*5，好处是降低参数，增加网络非线性

V3:

使用了卷积分解1\*3和3\*1代替3\*3，好处这样的好处，既可以加速计算

增加网络宽度，网络输入从224\*224变为了299\*299

V4:

结合了残差网络，使得训练加速收敛更快，精度更高

Xinception:

基本思想就是通道分离式卷积

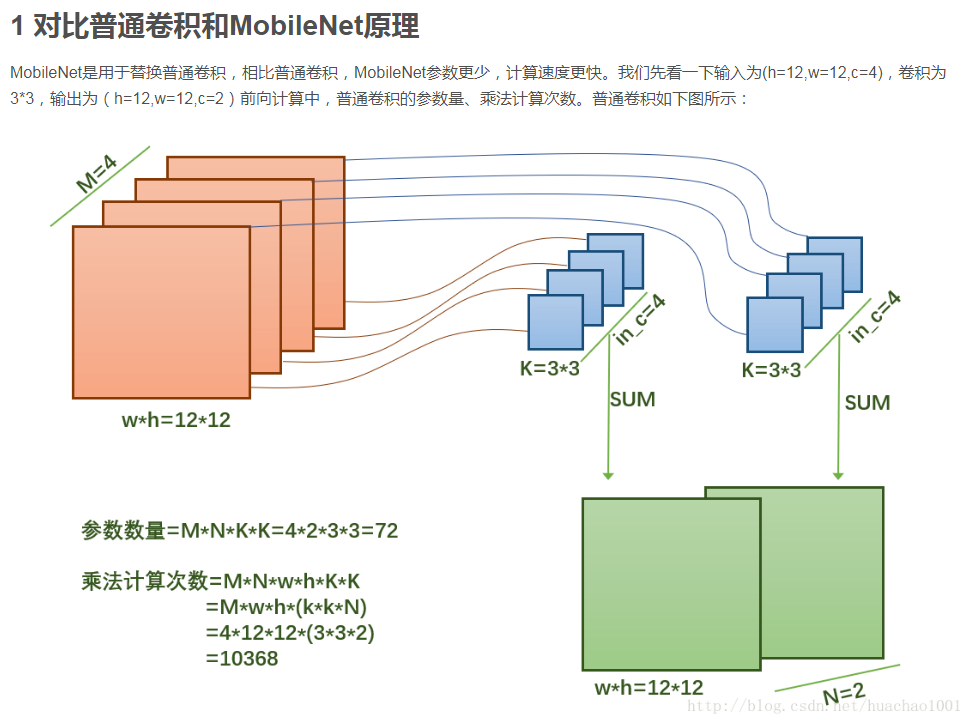
先说，卷积的操作，主要进行2种变换，(1)spatial dimensions，空间变换(2)channel dimension，通道变换

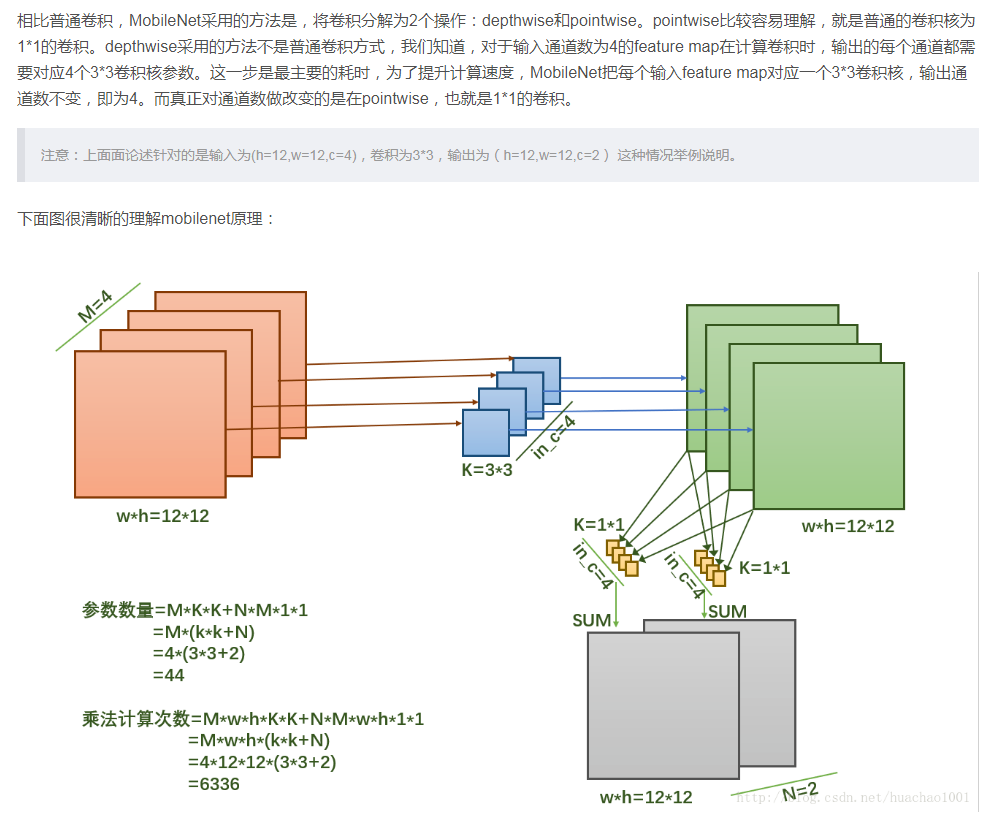
改进一：inception V3是先做1\*1的卷积，再做3\*3的卷积，这样就先将通道进行了合并，即通道卷积，然后再进行空间卷积，而Xception则正好相反，先进行空间的3\*3卷积，再进行通道的1\*1卷积

改进二：Inception V3在每个module中都有RELU操作，而Xception在每个module中是没有RELU操作的

MobileNets其实就是Exception思想的应用。区别就是Exception文章重点在提高精度，而MobileNets重点在压缩模型，同时保证精度。

Mobilenet结构图：



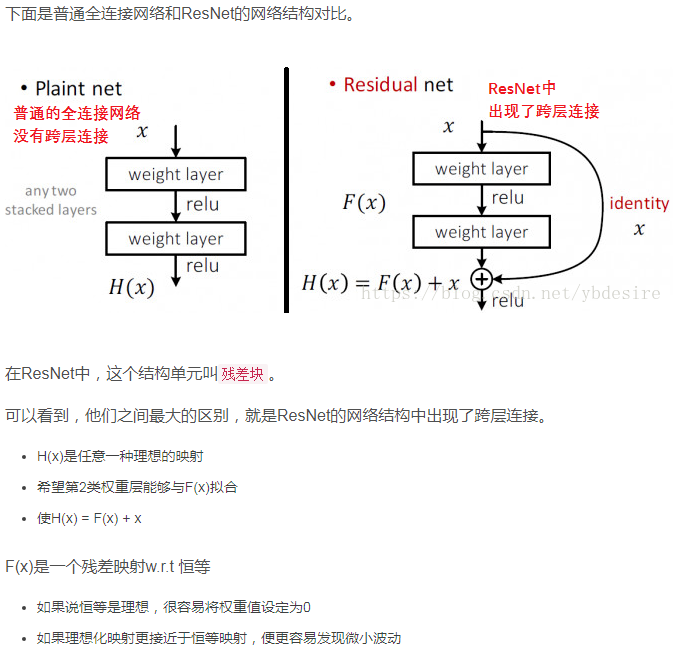


1. Resnet效果好的原因：

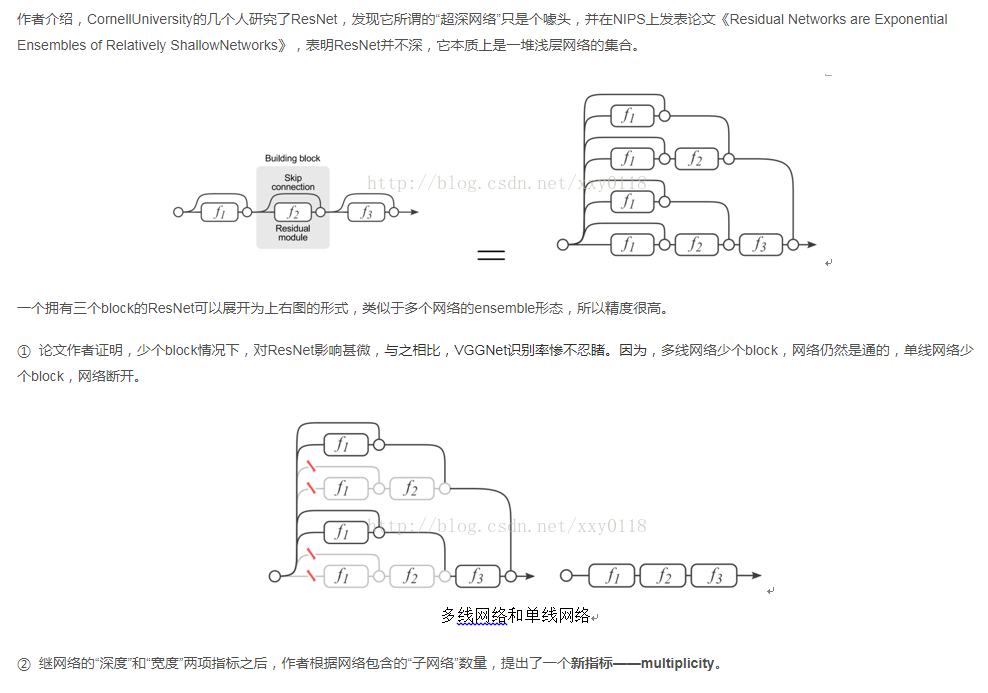
对vgg做实验，重复在加深网络深度，识别效果并没有提升反而会下降一点点，原因：网络越深，容易导致拟合，比较难训练。

Resnet 效果好的原因：

第一种解释：使网络更容易在某些层学到恒等变换（identity mapping）。在某些层执行恒等变换是一种构造性解，使更深的模型的性能至少不低于较浅的模型。这也是作者原始论文指出的动机。H(X) = F(X) + X =>F(X) = H(X) – X



第二种解释：残差网络是很多浅层网络的集成（ensemble），层数的指数级那么多。主要的实验证据是：把 ResNet 中的某些层直接删掉，模型的性能几乎不下降。



第三种解释：残差网络使信息更容易在各层之间流动，包括在前向传播时提供特征重用，在反向传播时缓解梯度信号消失。

1. dropout解读

<https://blog.csdn.net/program_developer/article/details/80737724>

为什么要dropout？

在机器学习中，如果模型参数太多，训练数据太少，训练出来的模型容易导致过拟合现象。具体表现就是：模型在训练数据集上的loss较小，预测准确率较高；但是在测试数据集上loss较大，准确率较低。

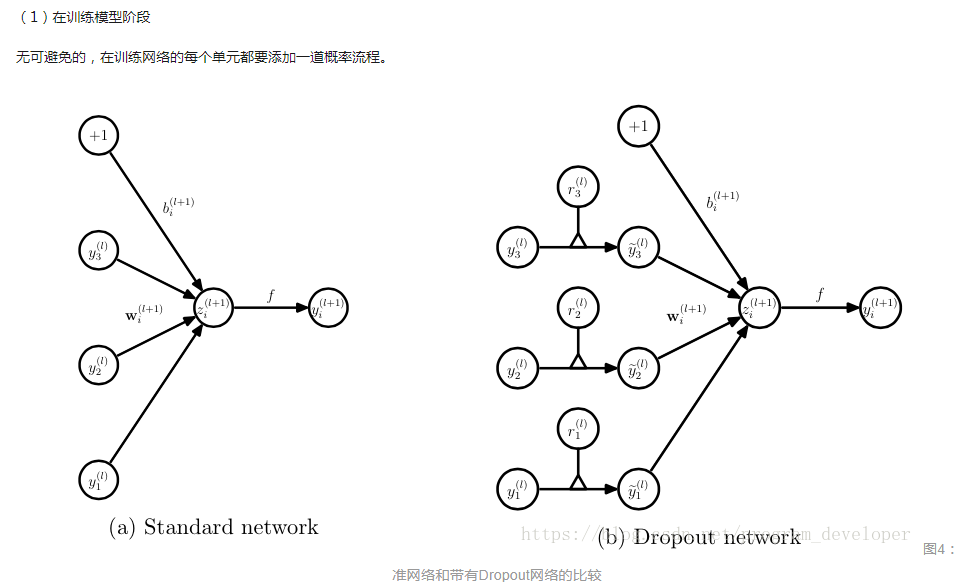
为了解决过拟合问题，一般会采用模型集成的方法，即训练多个模型进行组合，此时，训练多个模型很费时，测试多个模型也很费时。所以训练深度神经网络遇到的两个问题：

* 过拟合
* 费时

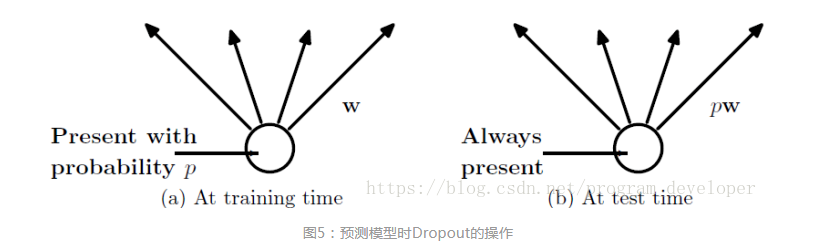
为了缓解这个问题，引入dropout。

Dropout怎么做？

在训练的时候，在吗，每次迭代的过程中，以一定概率使神经元休眠，即设置为0



在测试模型的时候，所有神经元的权重参数都乘以概率p



为什么dropout可以解决过拟合？

从bagging的角度：去平均的作用

假设模型没有用dropout，训练5个不同的神经网络，那就会得到5个输出结果，此时我们可以采用多数取胜或者取平均的策略决定最终的预测结果。比如三个模型判定为数字9，那最终的结果就是9.这种策略可以防止过拟合问题，因为不同的模型产生不同的过拟合，取平均会使得一些相反的拟合相互抵消。Dropout训练的时候随机的休眠一些神经元就是相当于在训练不同的网络，整个dropout过程相当于对很多不同的网络取平均，不同的网络产生的那些反向过拟合会一定的抵消，从而整体上使得网络减少过拟合。

从正则化的角度：减少神经元之间复杂的共适应关系

Dropout导致两个神经元不一定都在同一个dropout网络中出现，这样权值的更新不在依赖固定的关系的隐含节点的共同作用，阻止了某些特征仅仅在其他特定的特征下才有的效果的情况，这就使得网络去学习更加鲁棒性的特征。换句话说，网络在作出某种预测，它不应该对特定的线索片段太过敏感，即使丢掉一些特征，它也能从其他众多的特征中学习到共同的特征。从这个角度看dropout就有点像L1，L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高。

1. batch normlization解读：《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》

https://www.cnblogs.com/guoyaohua/p/8724433.html

<https://blog.csdn.net/hjimce/article/details/50866313>

机器学习领域有个很重要的假设：

**IID独立同分布假设**，就是假设训练数据和测试数据是满足相同分布的，这是通过训练数据获得的模型能够在测试集获得好的效果的一个基本保障。

Internal Covariate Shift的解释：

**在训练过程中，隐层的输入分布老是变来变去，这就是所谓的“Internal Covariate Shift”，Internal指的是深层网络的隐层，是发生在网络内部的事情，而不是covariate shift问题只发生在输入层。BatchNorm就是在深度神经网络训练过程中使得每一层神经网络的输入保持相同分布的。**

BN引入：

　然后提出了BatchNorm的基本思想：能不能**让每个隐层节点的激活输入分布固定下来呢**？这样就避免了“Internal Covariate Shift”问题了。

BN不是凭空拍脑袋拍出来的好点子，它是有启发来源的：之前的研究表明如果在图像处理中对输入图像进行白化（Whiten）操作的话——所谓**白化**，**就是对输入数据分布变换到0均值，单位方差的正态分布**——那么神经网络会较快收敛，那么BN作者就开始推论了：图像是深度神经网络的输入层，做白化能加快收敛，那么其实对于深度网络来说，其中某个隐层的神经元是下一层的输入，意思是其实深度神经网络的每一个隐层都是输入层，不过是相对下一层来说而已，那么能不能对每个隐层都做白化呢？这就是启发BN产生的原初想法，而BN也确实就是这么做的，**可以理解为对深层神经网络每个隐层神经元的激活值做简化版本的白化操作。**

BN的基本思想:

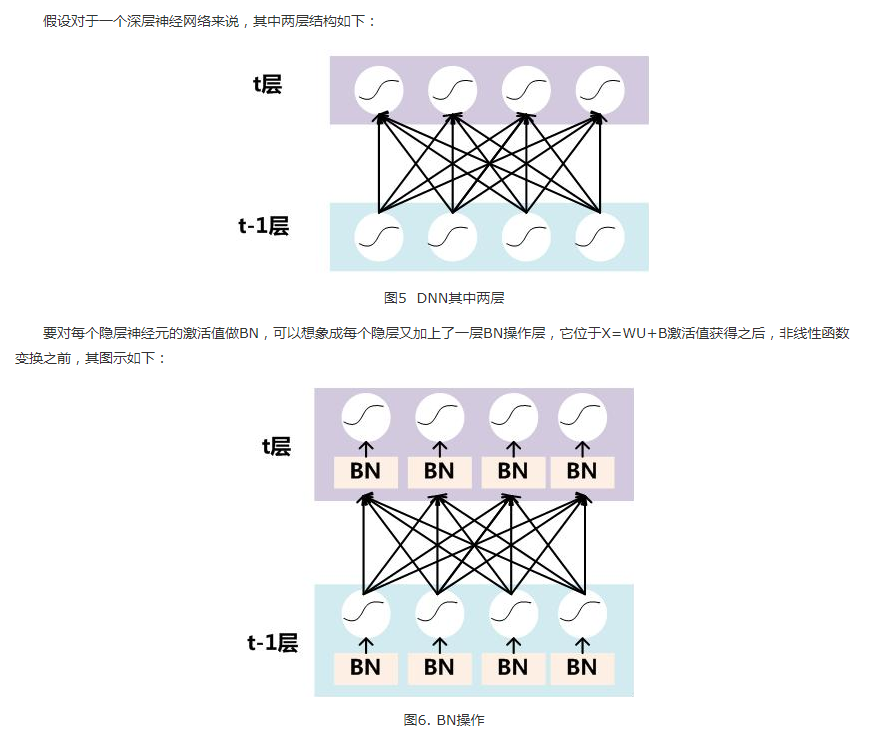
BN的基本思想其实相当直观：因为深层神经网络在做非线性变换前的**激活输入值**（就是那个x=WU+B，U是输入）**随着网络深度加深或者在训练过程中，其分布逐渐发生偏移或者变动，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近**（对于Sigmoid函数来说，意味着激活输入值WU+B是大的负值或正值），所以这**导致反向传播时低层神经网络的梯度消失**，这是训练深层神经网络收敛越来越慢的**本质原因**，**而BN就是通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到均值为0方差为1的标准正态分布**，其实就是把越来越偏的分布强制拉回比较标准的分布，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。**

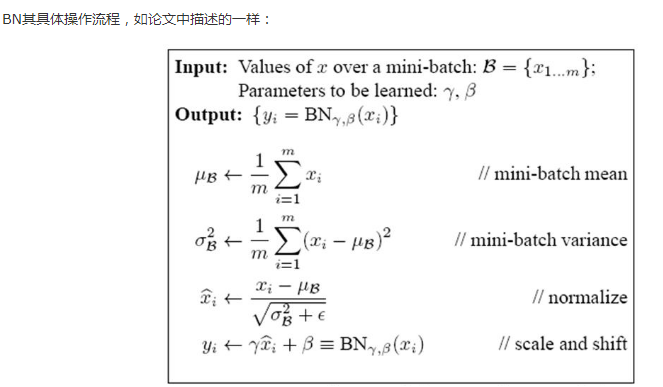
**到这里会产生一个问题？强行变换会导致网络的非线性下降，也就意味着网络的表达能力下降了？**

**解决办法：**

BN为了保证非线性的获得，对变换后的满足均值为0方差为1的x又进行了scale加上shift操作(y=scale\*x+shift)，每个神经元增加了两个参数scale和shift参数，这两个参数是通过训练学习到的，意思是通过scale和shift把这个值从标准正态分布左移或者右移一点并长胖一点或者变瘦一点，每个实例挪动的程度不一样，这样等价于非线性函数的值从正中心周围的线性区往非线性区动了动。核心思想应该是想找到一个线性和非线性的较好平衡点，既能享受非线性的较强表达能力的好处，又避免太靠非线性区两头使得网络收敛速度太慢。

BN的具体训练流程：





第一步，求出某个batch中的均值和方差，然后归一化每个输出，变换后某个神经元的激活x形成了均值为0，方差为1的正态分布；

第二部，第一步实现了把值往后续要进行的非线性变换的线性区拉动，增大导数值，增强反向传播信息流动性，加快训练收敛速度。但是这样会导致网络表达能力下降，为了防止这一点，每个神经元增加两个调节参数（scale和shift），这两个参数是通过训练来学习到的，用来对变换后的激活反变换，使得网络表达能力增强，即对变换后的激活进行如下的scale和shift操作，这其实是变换的反操作，上图公式中最后一步。

源码中的实现：

过上面的学习，我们知道BN层是对于每个神经元做归一化处理，甚至只需要对某一个神经元进行归一化，而不是对一整层网络的神经元进行归一化。既然BN是对单个神经元的运算，那么在CNN中卷积层上要怎么搞？假如某一层卷积层有6个特征图，每个特征图的大小是100\*100，这样就相当于这一层网络有6\*100\*100个神经元，如果采用BN，就会有6\*100\*100个参数γ、β，这样岂不是太恐怖了。因此卷积层上的BN使用，其实也是使用了类似权值共享的策略，把一整张特征图当做一个神经元进行处理。

卷积神经网络经过卷积后得到的是一系列的特征图，如果min-batch sizes为m，那么网络某一层输入数据可以表示为四维矩阵(m,f,p,q)，m为min-batch sizes，f为特征图个数，p、q分别为特征图的宽高。在cnn中我们可以把每个特征图看成是一个特征处理（一个神经元），因此在使用Batch Normalization，mini-batch size 的大小就是：m\*p\*q，于是对于每个特征图都只有一对可学习参数：γ、β。说白了吧，这就是相当于求取所有样本所对应的一个特征图的所有神经元的平均值、方差，然后对这个特征图神经元做归一化。

BN的单个图片前向传播过程：

单张图片没有minibatch的定义，所以归一化均值和方差使用训练过程中的所有batch的平均值，r和b则是训练的好的参数，其训练方法和权值w\b训练方法一样。

BN的好处：

。①不仅仅极大提升了训练速度，收敛过程大大加快；②还能增加分类效果，一种解释是这是类似于Dropout的一种防止过拟合的正则化表达方式，所以不用Dropout也能达到相当的效果；③另外调参过程也简单多了，对于初始化要求没那么高，而且可以使用大的学习率等。

normalization（batch normalization）。batch normalization的是指在神经网络中激活函数的前面，将wx+b按照特征进行normalization，这样做的好处有三点：

提高梯度在网络中的流动。Normalization能够使特征全部缩放到[0,1]，这样在反向传播时候的梯度都是在1左右，避免了梯度消失现象。

提升学习速率。归一化后的数据能够快速的达到收敛。

减少模型训练对初始化的依赖

1. 分类的评价模型：

关于混淆矩阵：

|  |  |  |
| --- | --- | --- |
| 正样本\预测值 | 0 | 1 |
| 0 | 35 | 15 |
| 1 | 20 | 80 |

假设上表0代表负样本，1是正样本

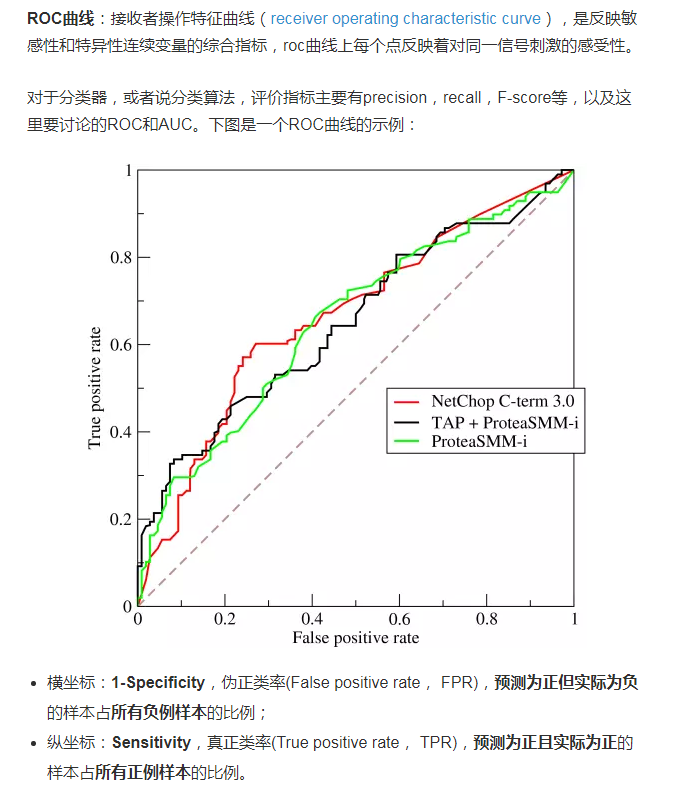
Acc = 35+80 / 150

Recall = 80/100 表示分类正确的样本除以所有正样本；

Precision = 15/15+80 表示分类正确的样本除以判断为正样本的个数

F1:综合评价 2/F1 = 1/Recall + 1/Precision

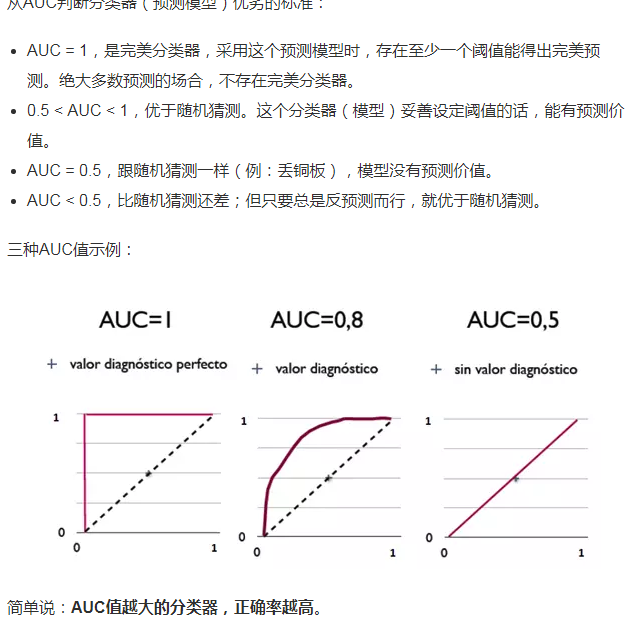
ROC曲线：



反应分类的好坏，横坐标是FPR,纵坐标是TPR.理性情况是FPR接近0，TPR接近1.

AUC指标：

AUC ([Area Under Curve](https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve)) 被定义为ROC曲线下的面积，显然这个面积的数值不会大于1。又由于ROC曲线一般都处于y=x这条直线的上方，所以AUC的取值范围一般在0.5和1之间。使用AUC值作为评价标准是因为很多时候ROC曲线并不能清晰的说明哪个分类器的效果更好，而作为一个数值，对应AUC更大的分类器效果更好。总结如图：



关于检测的评价模型：

1.对于某个类别C，在某一张图片上

首先计算C在一张图片上的Precision=在一张图片上类别C识别正确的个数（也就是IoU>0.5）/一张图片上类别C的总个数

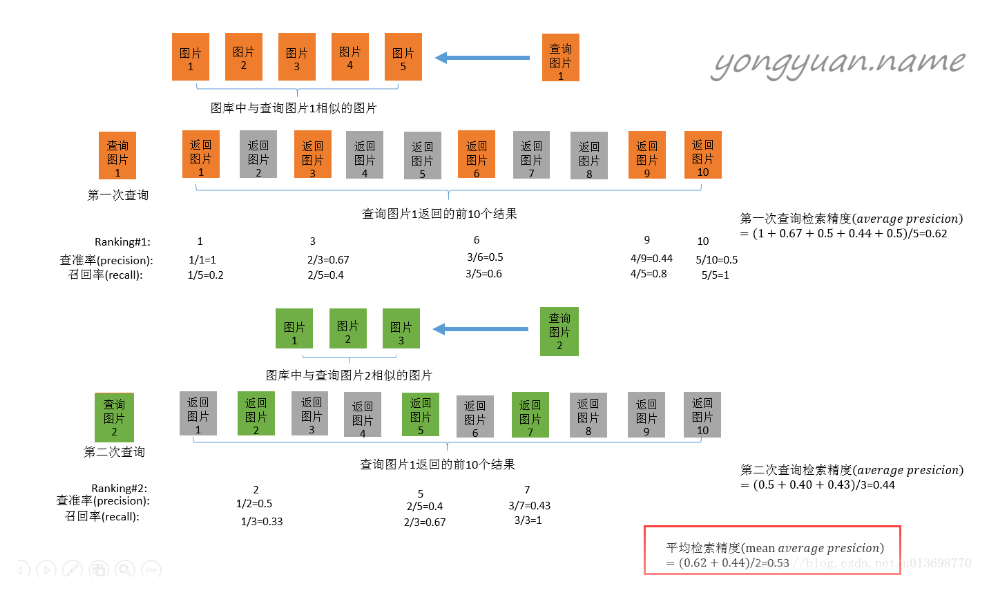
2.依然对于某个类别C，可能在多张图片上有该类别，下面计算类别C的AP指数：

AP=每张图片上的Precision求和/含有类别C的图片数目

3.对于整个数据集，存在多个类别C1、C2、C3

关于行人重识别的评价模型：

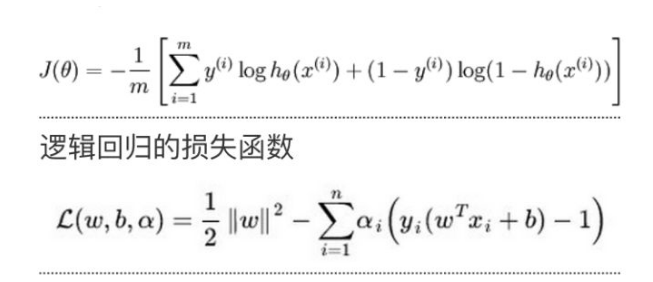
Rank1\rank5 :查询的前K张图片，包含正样本。



1. 逻辑回归和svm

SVM 作为线性分类机器学习方法中的经典，其动机是寻找最优的支持向量来构建分割平面，当问题进步到multiclass的分类时，核方法的引入帮助SVM把线性不可分的数据映射到更高层次的超平面空间中从而使数据达到线性可分，但是归根结底无论是线性svm还是核方法SVM的分类精度都主要依赖于数据中位于分割平面的一部分支持向量。然而反观softmax也就是CNN方法中用于分类的逻辑回归层，它是基于回归的原则，LR的损失函数也是得自于一个考虑了全局数据的自概率模型。

损失函数对比：



SVM与 LR很大的一个区别就是：SVM的性能主要受支持向量的影响，而LR考虑全局数据的同时分类超平面也会收到每一个样本的干扰。

所以我们可以认为当数据的特征feature处于一个线性不可分的状态的时候，SVM与LR是从两个不同的角度去解决了同一个分类问题，但是要注意CNN所连接的LR与单纯的LR的区别就在于CNN在训练的同时就会趋于提取样本中feature 的 signature特征，也就是说CNN的卷积过程在训练的过程中就是一个对线性不可分的数据增加其线性可分程度的过程。这一观点在哈工大Yushi chen的论文中也得到过验证。

最后问题来了，当CNN的feature提取生效之后，原本线性不可分的分类样本趋于线性可分的时候，SVM 只利用部分支持向量样本的分类原则的优势就能够得到进一步展现。

1. 特征工程

特征表示：

1. 特征值为实数，可以直接表示；
2. 字符串映射：独热编码（onehot）,将特征值表示成01串
3. 字符串映射：映射分类值（枚举）
4. 交叉特征

交叉特征算是特征工程中非常重要的方法之一了，交叉特征是一种很独特的方式，它将两个或更多的类别属性组合成一个。当组合的特征要比单个特征更好时，这是一项非常有用的技术。数学上来说，是对类别特征的所有可能值进行交叉相乘。

选用良好的特征：

1. 不存在或很少存在使用率低的离散特征值

一个良好的特征值在数据集中出现大约5次及以上，一个特征出现的次数很少，比如在person\_info中加入person\_id:51029302020XXXXX，那么person\_id便不适合作为特征，因为这个值是唯一的，模型无法从这个特征中学到任何规律。

1. 特征具有清晰明确的含义

良好的特征应该具有清晰且明确的含义。像是person\_info中的age:21可以明确地知道这个人是21岁。

1. 考虑到上游不稳定性

对于良好的特征，特征的定义不应该随着时间而变化。

清理数据：

准备数据集的时候，一些输入可能是会对模型学习结果产生不好影响的坏数据。比如在前面我们训练TensorFlow的Object Detection项目的时候，一个样本中对Pen的标注出现偏差，或是将茶杯标注成了Pen。在机器学习的时候，我们会花费大量的时间来挑除这样的坏样本来拯救数据集，因为即使存在少量的坏数据也会对一个大规模的数据集产生很大的影响，甚至是破坏整个数据集。一般的处理方法如下：

1、缩放：缩放，顾名思义，将特征值从自然范围（比如0到1000）转换到标准范围（0到1或-1到1）。 对于多特特征，缩放可以帮助提升梯度下降法的收敛速度；避免因训练时超出数值范围（如大于float型最大值的范围）而造成的NaN而产生的后果（即NaN陷阱）；帮助模型为特征确定合适的权重。 除了将[min,max]以线性方式缩放到诸如[0,1]范围之内，还有另一种缩放策略——计算每个数值的Z得分，缩放值=（原始数值-平均值）/标准偏差。

2、极端离群值：

3、分箱：假设我们存在一个特征值分布在从32到44范围之内的特征，可以看出，跟随纬度的变化，房价的变化不再是线性关系。为了将纬度特征变成一个实用的预测指标，则需要对纬度进行分箱。

即对32到44之间的内容等分为11份。如第一个箱（Bin）是从32到34，第二个箱是从34到36…

我们使用一个具有11个元素的矢量来表示，即33纬度坐落在第一个箱内，则该纬度表示为：[1,0,0,0,0,0,0,0,0,0]

4、清查：

有时，在进行数据集准备的时候，我们获取的数据往往存在不可靠的情况。其具体表现有如下几种：

不良标签：将钢笔误标记为茶杯

重复样本：一个样本存在两次（同时存在训练集和验证集/测试集中，相当致命）

不良特征值：特征值错误，如将房价错误地多大了一个0

遗漏值：如某个房屋的价格忘记输入了，导致该样本的特征值为NaN

在同上面的一些对数据的处理过程中，直方图是可视化数据集中的数据的一种很直观的方法。同时，了解数据的最大值最小值、均值和中间值和标准偏差（方差）都是很有作用的。

对于缺失值的处理：

使用可用特征的均值来填补缺失值；

使用特殊值来填补缺失值，如-1；

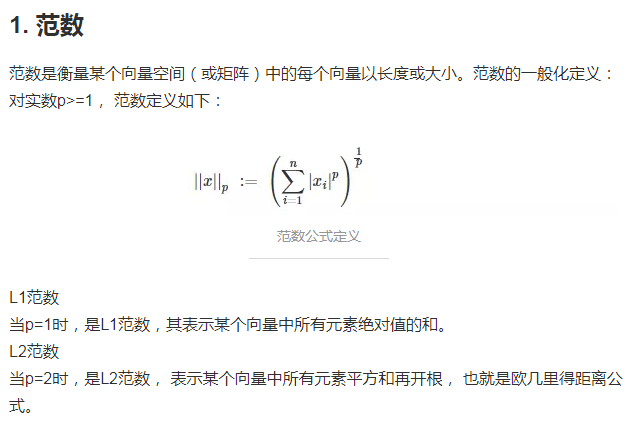
忽略有缺失值的样本；

使用相似样本的均值添补缺失值；

使用另外的机器学习算法预测缺失值。

1. L1、L2正则化

定义：



什么叫高维稀疏特征：对于线性特征，一个特征采用vector表示，记作x，x的维度越大，特征维度就越大，某个样本x(i)的取值，0越多表示越稀疏。

L1、L2正则的区别：

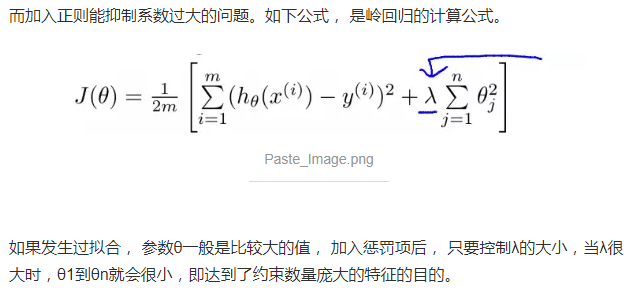
L1正则假设参数的先验分布是Laplace分布，可以保证模型的稀疏性，也就是使某些参数等于0；

L2正则假设参数的先验分布是Gaussian分布，可以保证模型的稳定性，也就是通过设置惩罚参数的值来控制模型参数的值，使其不会太大或太小，防止模型过拟合。

实际应用中，如果特征是高维稀疏的，那么使用L1正则化，如果特征是低维密集的，那么使用L2正则化；

正则化解决过拟合问题的两种解释：

解释一：来自知乎上一种比较直观和简单的理解， 模型过于复杂是因为模型尝试去兼顾各个测试数据点， 导致模型函数如下图，处于一种动荡的状态， 每个点的到时在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。



解释二：从贝叶斯的角度来分析， 正则化是为模型参数估计增加一个先验知识，先验知识会引导损失函数最小值过程朝着约束方向迭代。 L1正则是Laplace先验，L2是高斯先验。整个最优化问题可以看做是一个最大后验估计，其中正则化项对应后验估计中的先验信息，损失函数对应后验估计中的似然函数，两者的乘积即对应贝叶斯最大后验估计。

拉普拉斯函数和高斯函数：

|  |  |
| --- | --- |
|  |  |

高维稀疏特征的时候，lr 的效果会比 gbdt 好，为什么？

因为lr加入了正则化惩罚项对参数进行惩罚，防止过拟合；因为特征过于稀疏，gbdt容易过拟合，当构造的树过于简单时，效果不好，gbdt主要通过树的深度和叶子节点数来惩罚参数。

1. 样本不平衡的

上采样: 增加样本数较少的样本，其方式是直接复制原来的样本。样本较少时采用

下采样: 减少样本数较多的样本，其方式是丢弃这些多余的样本，样本较多时采用。

设小类中有N个样本。将大类聚类成N个簇，然后使用每个簇的中心组成大类中的N个样本，加上小类中所有的样本进行训练。（优点是保留了大类在特征空间的分布特性，又降低了大类数据的数目。实际情况下，上采样用的更多一些

设置权重:

增大样本数较少类别的样本的权重，当这样的样本被误分时，其损失值要乘上相应的权重，从而让分类器更加关注这一类数目较少的样本.

构造样本（SMOTE）:

增加样本数目较少的那一类的样本，合成指的是通过组合已有的样本的各个 feature 从而产生新的样本。

一种最简单的方法就是从各个 feature 中随机选出一个已有值，然后拼接成一个新的样本，这种方法增加了样本数目较少的类别的样本数，作用与上面提到的上采样方法一样，不同点在于上面的方法是单纯的复制样本，而这里则是拼接得到新的样本。这类方法中的具有代表性的方法是 SMOTE（Synthetic Minority Over-sampling Technique），这个方法通过在相似样本中进行 feature 的随机选择并拼接出新的样本。

分治ensemble：

将大类中样本聚类到L个聚类中，然后训练L个分类器；每个分类器使用大类中的一个簇与所有的小类样本进行训练得到；最后对这L个分类器采取少数服从多数对未知类别数据进行分类，如果是连续值（预测），那么采用平均值。

分层级ensemble：

使用原始数据集训练第一个学习器L1；将L1错分的数据集作为新的数据集训练L2；将L1和L2分类结果不一致的数据作为数据集训练L3；最后测试集上将三个分类器的结果汇总（结合这三个分类器，采用投票的方式来决定分类结果，因此只有当L2与L3都分类为false时，最终结果才为false，否则true。）

Adaboost的流程：

计算样本权重；

计算错误率：第一个弱学习算法h1对其进行学习，学习完成后进行错误率ε的统计；

计算弱学习算法权重

更新样本权重

AdaBoost算法，sign(x)是符号函数