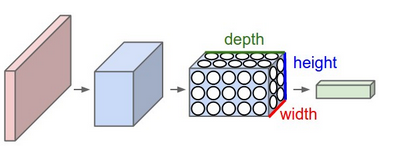
# CNN

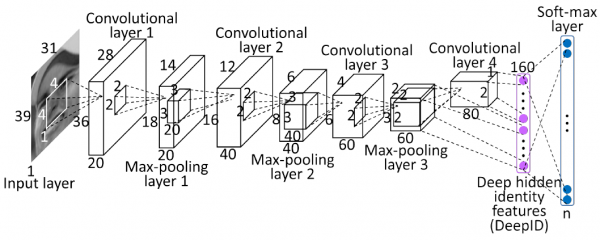
CNN主要用于图像处理。Convolutional Netural Network

CNN一个很好的教程：<http://cs231n.github.io/convolutional-networks/>

假设输入的是一幅像素为128\*128的图像，隐含层有1000个神经元，光这一层就有1638w个参数，这个对于普通的NN是灾难性的。对于CNN来说，它创造性的应用“卷积核”解决了这个难题 。

普通的NN的隐藏层是向量，而卷积层是一个三维立体结构（width,height,depth）。如： 

CNN的层级结构，不同层有不同的形式和功能：



## 数学上的卷积

作者：果程C

链接：https://www.zhihu.com/question/22298352/answer/50940942

来源：知乎

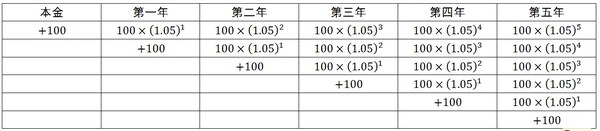
小明存入100元钱，年利率是5%，按复利计算（即将每一年所获利息加入本金，以计算下一年的利息），那么在五年之后他能拿到的钱数是100(1+5\%)^5。

https://pic2.zhimg.com/5fa86c80c31dd277d038527555aa4d75_b.jpg

一年后，小明又往银行中存入了100元钱，年利率仍为5%，那么这笔钱按复利计算，到了第五年，将收回的钱数是，我们将这一结果作为新的一行加入上面的表格中

https://pic4.zhimg.com/39f37df8c96d7219cba5d081919a3a2f_b.jpg

以此类推，如果小明每年都往银行中存入新的100元钱，那么这个收益表格将是这样的：



可见，最终小明拿到的钱将等于他各年存入的钱分别计算复利之后得到的钱数的总和，用求和符号来简化这个公式，可以得到：

http://www.zhihu.com/equation?tex=%5Csum_%7Bi%3D0%7D%5E%7B5%7D%7Bf%28i%29g%285-i%29%7D%2C+%5Cmathrm%7Bwhere%7D+%5C+f%28i%29%3D100%2C+g%285-i%29+%3D+%281.05%29%5E%7B5-i%7D

在上式中，f(i)为小明的存钱函数，g(i)为存入银行的每一笔钱的复利计算函数。在这里，小明最终得到的钱就是他的存钱函数和复利计算函数的卷积。

如果是连续,则：

http://www.zhihu.com/equation?tex=%5Cint_%7B0%7D%5E%7Bt%7D+f%28%5Ctau%29g%28t-%5Ctau%29d%5Ctau%3D%5Cint_%7B0%7D%5E%7Bt%7D+f%28%5Ctau%29%281%2B5%5C%25%29%5E%7Bt-%5Ctau%7Dd%5Ctau

这也就是卷积的表达式了，上式可以记为。

下面我们再展开说两句：

如果我们将小明的存款函数视为一个信号发生的过程（也就是激励），而将复利函数视为一个系统对信号的响应函数（也就是响应），那么二者的卷积就可以看做是在t时刻对系统进行观察得到的结果（也就是输出）。 这个结果将是过去产生的所有信号经过系统的「处理／响应」后得到的结果的叠加，这也就是卷积的物理意义了。

## 图像数据预处理

去中心和归一化。

CNN常用去均值、CNN不做归一化。因为图像天生就是在 0-255之间

pca和白化

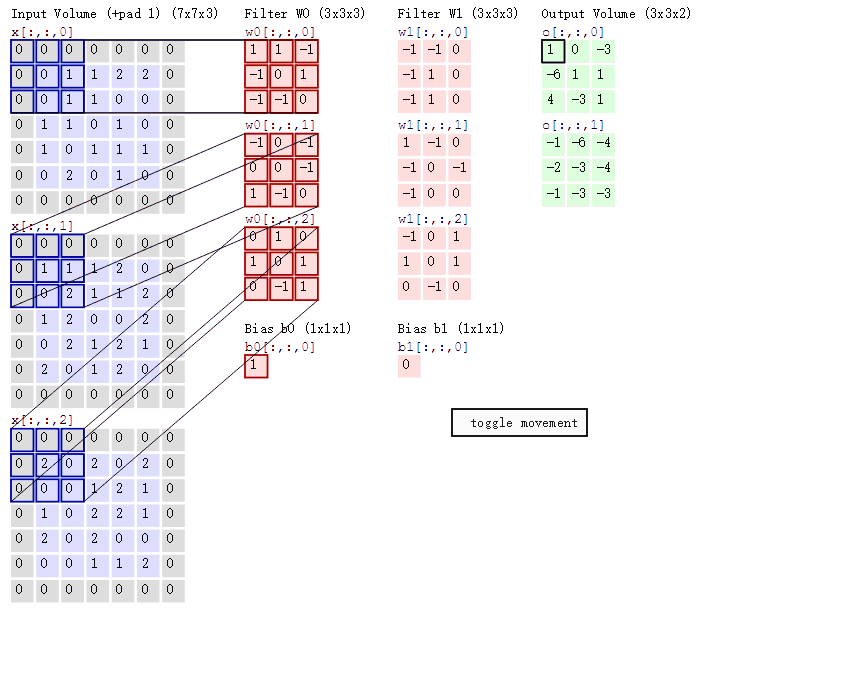
图像一般不用做 PCA、白化。

## conv layer

在神经网络中，全连接层的参数量过大。举个简单例子。图像像素是128\*128\*3，全连接的参数至少是 128\*128\*3 x 128\*128\*3 这个根本无法接受。所以通过卷积解决这个问题。

下图就是一个卷积计算。

一组固定的权重和不同的数据窗口数据做内积的过程，这在数学上刚好对应『卷积』操作，这也就是卷积神经网的名字来源。（与数学上的卷积还是不一样）



**几个重要概念**：

深度 depth ： conv层神经元的个数。如上图中有w0,w1 2个神经元，即深度=2。

步长 stride ： 窗口移动的步长。

是否填充0 zero-padding ：如果补0，则可以保证从窗头滑动到窗尾。

如果且stride=1,则，输出和输入在 height x wdith上保持不变。

**优势**

参数共享机制带来的参数个数的减小。即神经元的权重参数是固定的。（在前向计算中）。这样极大的减少了权重参数的个数。如上图中，一个神经元的窗口的大小是 3\*3 ，要处理RGB3个通道，所以一个神经元的参数量是3\*3\*3。 上图一共2个神经元，所以参数总个数是 2\*3\*3\*3=54. 而全连接要：2\*(7\*7\*3)^2=43218

so , 卷积层计算对高维数据处理无压力。

**关于conv的输出**：

如果：补0，及padding=”SAME”,

停留次数=in\_len/stride, 向上取整数

输出结果=[停留次数x停留次数] x depth

如果：不补0， padding=”VALID”

停留次数=(in\_len-窗口\_len + 1)/stride , 向上取整数

输出结果=[停留次数x停留次数] x depth

**关于参数量**

参数量=输入通道 \* filter size \*输出通道= 3 x[3 x3] x2

\*拿上图进行举例：，

input层图片大小是[5x5x3]，filter=[3x3]x2

则：停留次数=(5-3+1)/2=3 , 输出结果=[3x3]x2。

\*注意这里，输入的时候是3通道，输出的时候则变成了2通道。所以，卷积的depth=输出矩阵的通道。

若stride=1,padding=SAME，则 停留次数=5/1=3, 输出结果=[5x5]x2 。

经常设置stride=1。 stride=1时，输出的信息= 图片大小 x depth .

## 池化层 pooling layer

设计池化层的目的，是为了

1，进行数据压缩。如原 224\*224\*64 ，变成 112 \* 112 \* 64 。

2，进行特征提取。

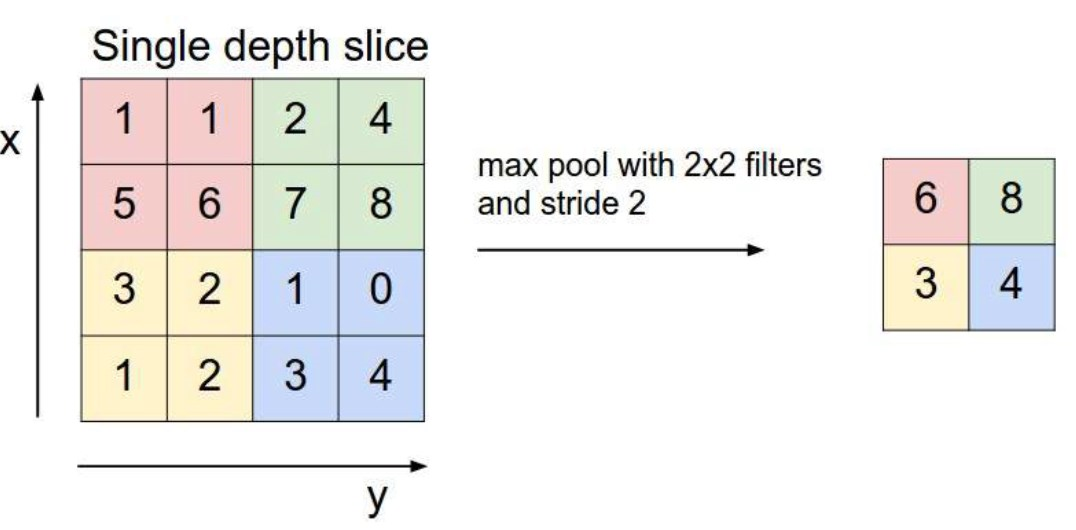
3，平移不变性。

2种池化方法：

* Max Pooling
* Average Pooling。

average对背景保留更好，max对纹理提取更好，如果是识别字体什么的，应该考虑max.

Max Pooling如下图：

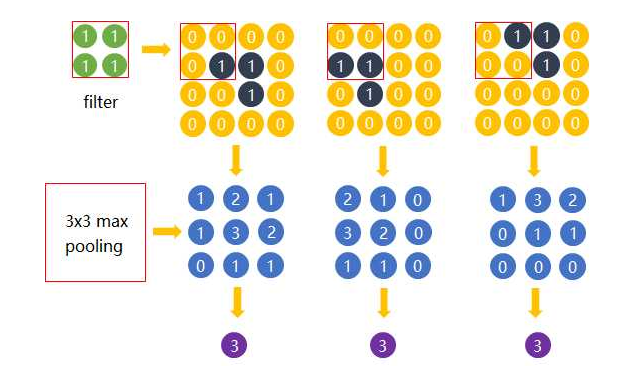


**平移不变性**

顾名思义，主体在图像中进行位置平移，依然不影响正确识别。

简单原理：三个元素（1，5，3）取max是5，如果向右平移一下变成（0，1，5），那取max之后还是5，具备了平移不变性。

举例：

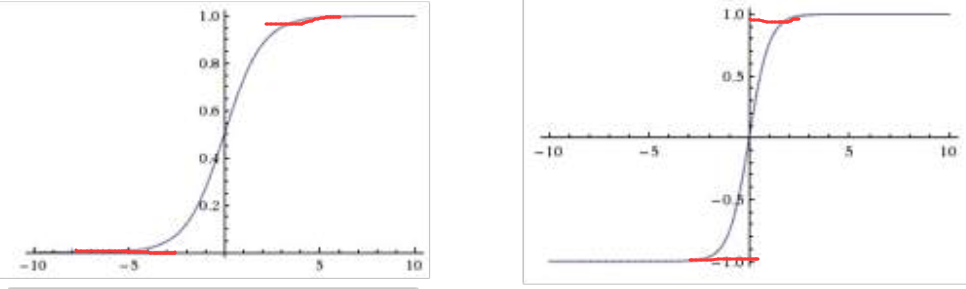


我们要考虑的是探测“横折”这个形状。 即使“横折”在不同的位置，通过conv+ pool后都能被正确的识别。体现了平移不变性。

## relu激励函数

sigmoid函数和tanh函数的缺点

它们的图像是：



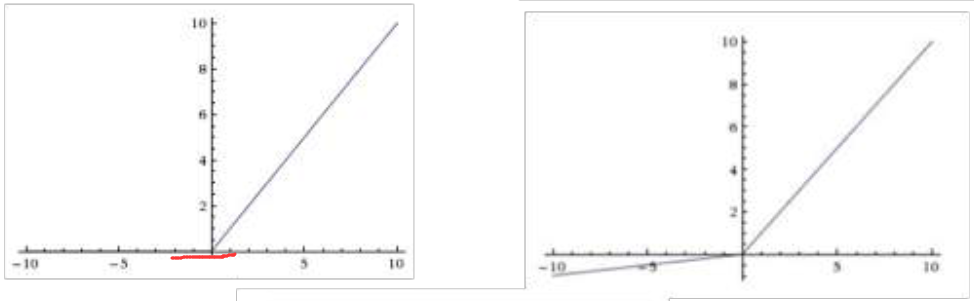
可以看到，在两端，他的斜率趋近0.（如红色标注）。也就是说偏导趋近于0。也就意味 ， 那权重就不能更新。

所以，设计了relu函数 .



它的导数是：





它的**优点**：

1. x>0, 则f(x)的导数是1, 不会存在为0的情况。而sgimoid函数当x>4时，梯度几乎为0

它的**缺点**：

·缺点1：如左图中，当x<0时，偏导也是0 。所以设计了leaky relu



实际中，用relu就好。如果relu不行，再尝试leaky relu.

·缺点2：ReLU，截断了x<0的部分，改变了数据的分布，因此ReLU后加Batch Normalization进行还原优化。

## LRN

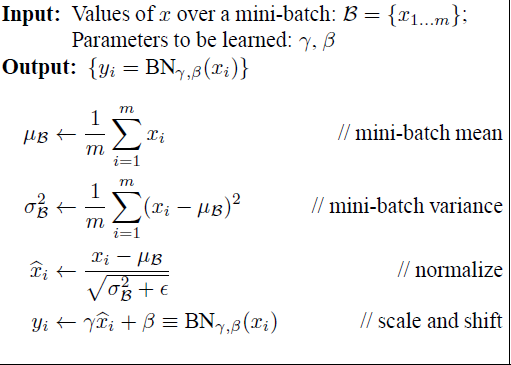
Local Response Normalization, 局部响应归一化。

sigmoid(x)的范围是(0,1),而relu(x)是[0,+8],是无上限的。 ALexNet是第一个使用relu的网络，它同时也提出了relu后接LRN ，对relu(x)的输出y进行缩放。后来的NN使用BN后，就没人使用LRN了。

## BN

Batch Normalization是由google提出的一种训练优化方法。参考论文：[Batch Normalization Accelerating Deep Network Training by Reducing Internal Covariate Shift ]( http://arxiv.org/abs/1502.03167)

BN就是对**一个mini-batch的数据**进行**正太标准化**。input的标准化，是对整个input的数据进行标准化。而最后的“scale and shift”操作是让数据变化到设定的均值和标准差（beta=均值，gamma=标准差，一般是beta=0, gamma=1）。



**why BN?**

因为，反向传播时，每层的梯度计算都会用到它的输入。



但是在NN中，中间层的输入的数据分布就发生了改变，与输入层的数据分布不一样。如果v\_i 大多小于1，那么传到这里的时候梯度会变得很小，从而梯度消失；而如果 v\_i 大多大于1，那么传到这里的时候会变得很大，从而梯度爆炸问题。BN所做的就是解决这个梯度传播的问题，因为BN作用抹去了v\_i的scale影响。

so, BN解决了梯度消失和梯度爆炸的难题。

**BN trainning**

* 设置 beta和gamma. 经过BN之后，数据分布发生了改变，而 beta（期望均值）和gama（期望方差）表示数据变换的目标。
* 计算batch的mean和variance. 在train阶段，直接计算，如：mean, variance = tf.nn.moments(x, [0, 1, 2])
* predict阶段，读取moving\_mean, moving\_variance作为mean,和variacne进行bn.而不是直接计算。
* return tf.nn.batch\_normalization(x, mean, variance, beta, gamma, 1e-8)

**BN use**

BN的使用位置是 conv层和relue之间： conv + BN + relu .

这个位置应该是因为：conv改变数据分布，通过bn可以进行还原。而relu又保持了很好的非线性特性。.

bn这个位置，参考的是：S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep

network training by reducing internal covariate shift. In ICML, 2015.

**BN的好处**

实践证明：

* 更快的训练速度。
* 更高的精度

## Swish激活函数

2017年10月，google提出了一个新的激活函数，swish激活函数：



其中，

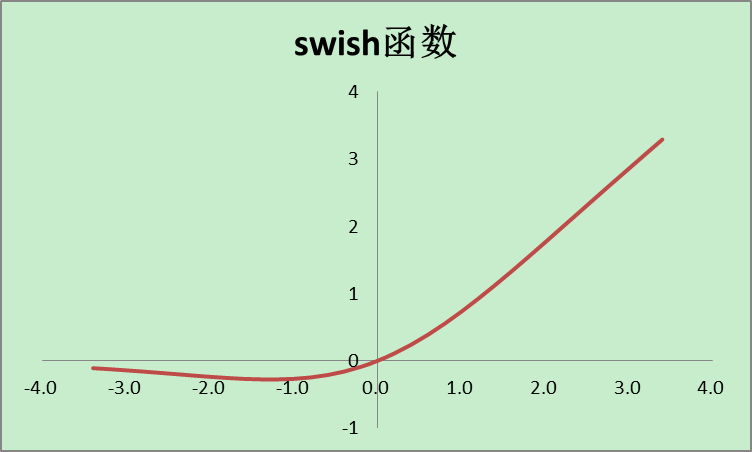
设计思路：

Swish 的设计受到 LSTM 中使用 sigmoid 函数进行门控的启发。我们使用同样的值进行门控来简化门控机制，称为自门控（self-gating）

**它的导数：**



它的图形：



**它的特性：**

* 无上界有下界。这和relu一样。
* 非单调且平滑。（relu非单调，但不平滑，生硬。而swish的平滑使的在深度神经网络的表现要好于relu）

## FC(全连接层 )

这是我们在介绍神经网络的时候，最标准的形式，任何神经元和上一层的任何神经元之间都有关联，然后矩阵运算也非常简单和直接。现在的很多卷积神经网络结构，最后一层会采用全连接。

## CNN的训练

和BP一样，也用的是SGD。

链式求导法则，每层都求导，这样每层就可以用链式求导法则串起来。

关键点：

1. 求每层的误差，
2. 求局部梯度

relu的导数是：



Pool层的导数

Pool层的操作是downsampling ,则Pool层的导数是UpSampling。

MaxPool做的是max( x11,x12,x21,x22)=x12，则它的导数是：(0,1,0,0)

具体实现：就是记录前向传播过程中pooling区域中最大值的位置，该位置导数是1.

**卷积的导数：**

这个相对复杂些,但还是能求出来的。

对于FC层来说， , 

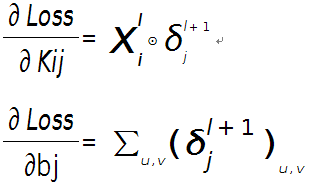
根据链式法则，误差e对w的偏导是：，

对卷积来说，矩阵乘法变成了卷积操作。，那么： 

答案：

具体参考：<http://www.cnblogs.com/tornadomeet/p/3468450.html>中 “问题四：求卷积层相连那层的权值、偏置值导数”。

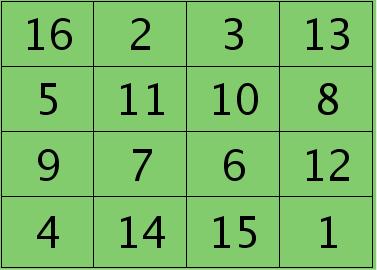
假设现在需要求第l层的第i个通道，与第l+1层的第j个通道之间的权值和偏置的导数，则计算公式如下：



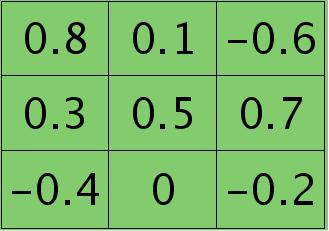
　　其中k\_ij表示卷积核

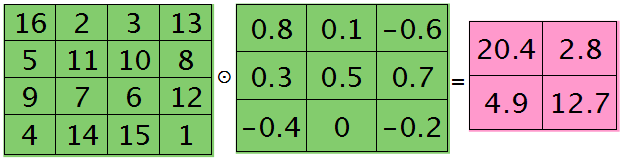
其中符号⊙表示矩阵的相关操作，可以采用conv2()函数实现。在使用该函数时，需将（第l+1层第j个误差敏感值）翻转。

　　例如，xi(第l层某个通道的矩阵i)大小为4×4,如下：



（第l+1层第j个神经元的误差敏感值矩阵）大小为3×3,如下：



　　卷积核为2×2，则,卷积核的导数是：　

　　而此时偏置值bj的导数为1.2 ，将j区域的误差敏感值相加即可(0.8+0.1-0.6+0.3+0.5+0.7-0.4-0.2=1.2)，因为b对j中的每个节点都有贡献，按照多项式的求导规则(和的导数等于导数的和)很容易得到。

## dropout正则化

正则化包括：L1, L2 , L1+L2 。这种正则化一般是对损失函数做的惩罚处理。

神经网络还独有一种正则化手段，那就是dropout .

神经网络学习能力太强，容易过拟合。

dropout ,随机失活，别一次开启所有神经元。

p=0.5 #设定dropout的概率，也就是保持一个神经元激活状态的概率。

\*dropout只在FC层使用。

# CNN经典模型

## CNN发展历史

* LeNet(1998)：卷积神经网络始自1990年代，最著名的是LeNet
* AlexNet(2012)：2012年，Alex Krizhevsky发布了AlexNet，是LeNet的更深、更宽版本，并且大比分赢得了当年的ImageNet大规模图像识别挑战赛(ILSVRC)。这是一次非常重要的大突破，现在普及的卷积神经网络应用都要感谢这一壮举。
* ZF Net(2013)：2013年的ILSVRC赢家是Matthew Zeiler和Rob Fergus的卷积网络，被称作ZF Net，这是调整架构超参数的AlexNet改进型。
* VGGNet(2014)：2014年 ILSVRC亚军，突出贡献是展示了网络的深度（层次数量）是良好表现的关键因素。
* GoogleNet(2014)：2014年ILSVRC冠军，来自Google的Szegedy et al.。其主要贡献是研发了Inception Module，它大幅减少了网络中的参数数量（四百万，相比AlexNet的六千万）。
* VGGNet(2014)：2014 ILSVRC亚军，突出贡献是展示了网络的深度（层次数量）是良好表现的关键因素。
* ResNet(2015)： Kaiming He研发的Residual Network是2015年的ILSVRC冠军，也代表了卷积神经网络的最高水平，同时还是实践的默认选择（2016年5月）。
* DenseNet(2016.8)：由Gao Huang发表，DenseNet的每一层都直接与其他各层前向连接。DenseNet已经在五个高难度的物体识别基础集上，显式出非凡的进步。

## LeNet

1998年提出，mnist准确率达到99.2%。LeNet标志着CNN的真正面世。

后续，2012 ALexNet(8 layers), 2014 VGG(19 layres) 都是类似LeNet结构。是LeNet的改进型。

**结构：**

input->卷积->池化->卷积->池化->FC->FC->FC

即：（卷积+->池化?)+全连接 的结构。

**具体：**

input (32x32x1)

->卷积 (w=[5,5]x6,strides=1, padding="VALID") => 32-5+1=28, 输出：28 x 28 x 6

->池化 (ksize=[2,2], strides=2, padding="SAME") => 14 x 14 x6

->卷积 (w=[5,5]x6,strides=1, padding="VALID") => 10 x10 x6

->池化 (ksize=[2,2], strides=2, padding="SAME") => 5 x 5 x6

->FC (size=120)

->FC (size=84)

->FC (size=10)

**规律总结：**

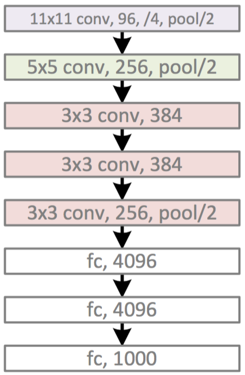
depth,一般逐层递增。

strids, 一般是1， 也有2，3.

## AlexNet

AlexNet 可以说是具有历史意义的一个网络结构，可以说在AlexNet之前，深度学习已经沉寂了很久。历史的转折在2012年到来，AlexNet 在当年的ImageNet图像分类竞赛中，top-5错误率比上一年的冠军下降了十个百分点

AlexNet网络的架构，它包含8层，5个卷积层和3个FC层



AlexNet 之所以能够成功，原因在于：

* Data augmentation(数据增强)，防止过拟合。
* 网络增大。（5个卷积层+3个全连接层）
* relu的出现。
* dropout的使用。
* 多GPU加速计算

图像的数据增强方式常用有：

* crop, 修剪。如：从原始图像（256,256）中，随机的crop出一些图像（224,224）
* flip，翻转。如：水平翻转
* color jittering，颜色抖动。如：给图像增加一些随机的光照

python代码参考：http://www.cnblogs.com/gongxijun/p/6117588.html

**VGG**

VGG很好地继承了AlexNet的衣钵，一个字：深，两个字：更深。19层

VGG的结构，都采用3x3的卷积核，深度从64，一直增加到后面的512，然后 pool/2, 最后接3个FC层分别是：FC\_4096, FC\_4096,FC\_1000.

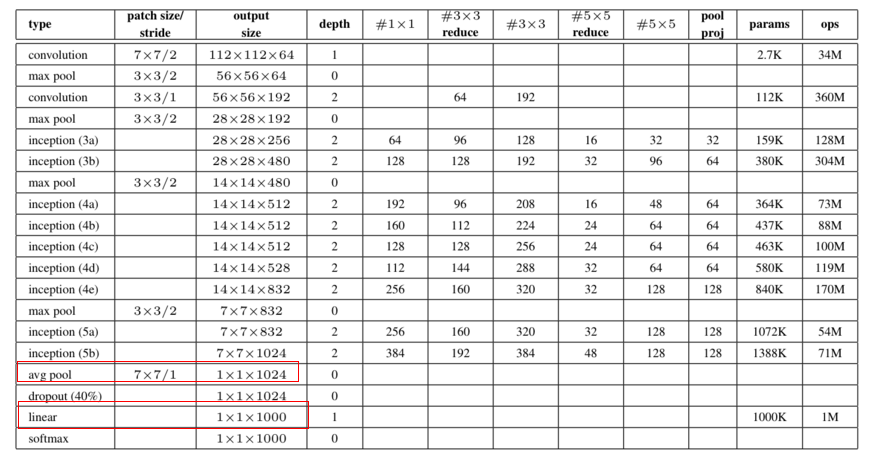
![image](https://www.52ml.net/wp-content/uploads/2016/08/vgg19.png?\_=5821591)

## google(Inception) Net

google Inception Net 首次出现在ILSVRC 2014的比赛中. 这一届比赛中的Inception Net称为Inception V1, 现在发展到 Inception V5

**GoogLeNet的模型结构及参数：**

![image]( https://raw.githubusercontent.com/stdcoutzyx/Blogs/master/blogs2016/imgs\_inception/5.png)



* 层数更多，Inception v1是22层，VGG是19层。（这里只统计conv,FC层）
* 参数更少。参数不到六百万，而AlexNet是六千万。

**参数怎么减少的？**

**AlexNet**

AlexNet只计算FC层参数量就非常huge.

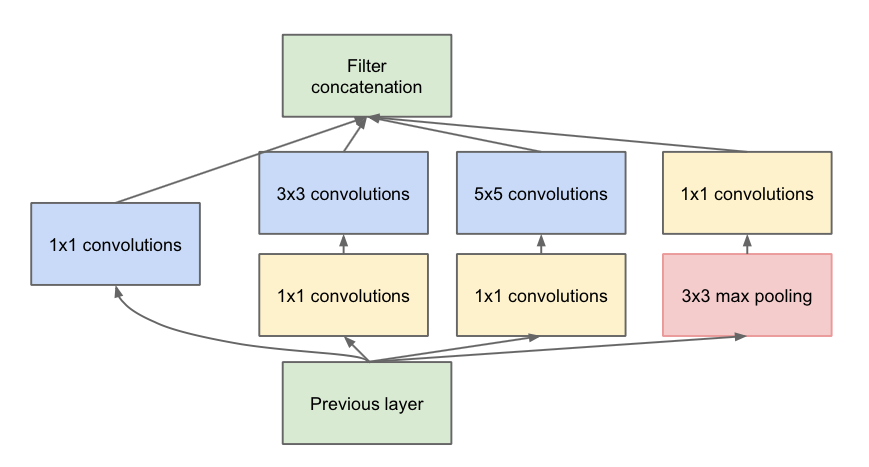
1. pool, FC\_4096之间的参数量: [2\*2]\*256\*4096=400w

2）FC\_4096, FC\_4096之间的参数量 4096 \*4096 =1600w

**GoogLeNet**

1. google net只有一层FC，而alexNet是3层FC。FC x FC的参数huge.
2. google net FC前面的avg pool的输出是1x1x1024, FC层的神经元是1024，它们之间的参数量只有100w，而alexNet这层有400w
3. google net的结构，大部分采用1x1的卷积核，1x1的卷积核相比3x3只有其1/9的参数量。

**inception module**



主要特点：

* LeNet的卷积采用的是串联，而Inception结构采用的并联。采用不同大小的卷积核意味着不同大小的感受野，最后拼接意味着不同尺度特征的融合；
* 文章说很多地方都表明pooling挺有效，所以Inception里面也嵌入了。
* 黄色的1x1卷积核的作用是降维 。

**为什么可以并联?**

因为：设定stirde=1,padding用0填充的话，则输出和输入的尺寸(height x width)一样。

so, 用1x1xdepth, 3x3 xdepth, 5x5 xdepth 的过滤器，最后的输出的尺寸(height x width)和input是一样的，这样就可以concat.（输出depth=过滤器depth之和）

tensorflow提供了 tf.contrib.slim快速实现。

**Inception V2**

使用3×3的已经很小了，那么更小的2×2呢？2×2虽然能使得参数进一步降低，但是不如另一种方式更加有效，那就是Asymmetric方式，即使用1×3和3×1两种来代替3×3。

使用2个2×2的话能节省11%的计算量，而使用这种方式则可以节省33%。于是，Inception再次进化为V2。

## 1x1卷积

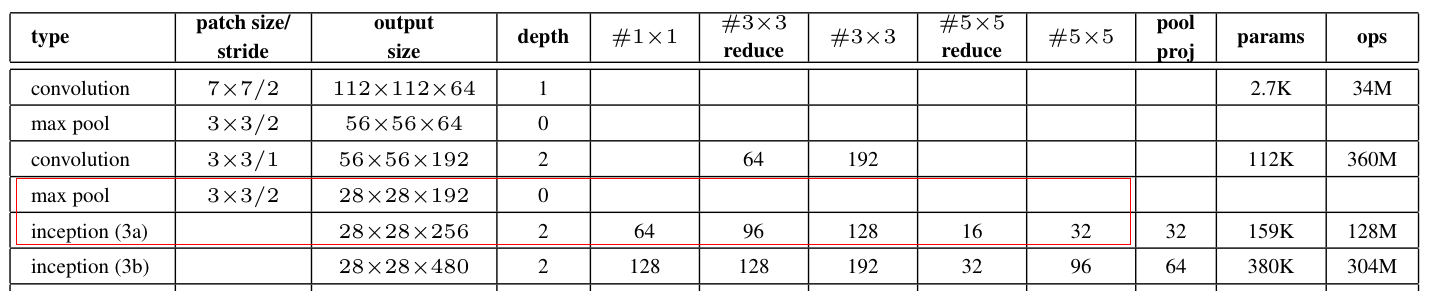
在google inception net，及 resNet中，大量使用1x1卷积，那么1x1卷积究竟起什么作用呢？

答案：

1）如果输入信号是2维的，1x1的卷积的确没什么用。但是图像的输入是三维的（有depth通道），所以，1x1的卷积能对depth维度进行处理。

2）1x1卷积可以实现通道的降维和升维，从而减少参数量。

例如，google net的inception(3a) ：

1×1卷积通道为64，3×3卷积通道为128，5×5卷积通道为32

3×3和5×5卷积层前分别加入了通道数为96和16的1×1卷积层。（即reduce层）

分别计算参数量，这里只举例计算3x3卷积和：

无reduce层：192\*3\*3\*128=221184

有reduce层：182\*1\*1\*96 + 96\*3\*3\*128 =128064

可以看到参数量减少了.

# ResNet

## ResNet

参考：<http://www.jianshu.com/p/e502e4b43e6d>

论文：<https://arxiv.org/abs/1512.03385>

理论上，神经网络越深，学习能力越强。但神经网络存在梯度消失的问题。梯度消失”问题指的是即当梯度在被反向传播到前面的层时，重复的相乘可能会使梯度变得无限小。so，层数越多，前面的层学习原始数据的能力消失掉。

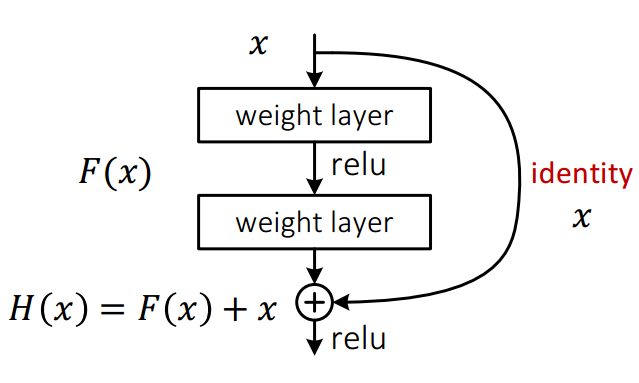
**ResNet的思路：**

ResNet的基本思想是引入了能够跳过一层或多层的“shortcut connection”。目的：增加网络的层不应该降低网络的性能，因为我们可以将“恒等变换(identity mapping)”简单地叠加在网络上，而且所得到的输出架构也会执行相同的操作。这就暗示了更深层的模型的训练错误率不应该高于与之对应的浅层模型。

可以这样理解：如果只有n层时,误差是L1, 若增加堆叠的层后的误差是L2, 由于shortcut, L2的误差应该少于L1, 所以，堆叠的层学习的效果是L1-L2 ，也就是残差。

**网络结构：**

ResNet的block结构,“Deeper Bottleneck Architectures”（以下简称DBA）：



图中的identity，是identitiy map , 翻译为恒等变换

【Q】看到这个图时，产生了一个疑问？

H(x)=F(x)+x , x的size or depth与 F(x) 不一致，怎么做“+”?

【A】 将x转换为与F(X)一致。

例如：

def subsample(inputs, stride, scope=None):

"""如 6x6 ,变成3x3"""

if stride == 1:

return inputs

else:

return tf.nn.max\_pool(inputs, ksize=[1, stride, stride, 1], strides=[1, stride, stride, 1], padding='SAME', name=scope)

#

if in\_depth != out\_depth:

shortcut = slim2.conv2d(orig\_x, 'shortcut', out\_depth, [stride,stride], stride, activation\_fn=None)

else:

shortcut = slim2.subsample(orig\_x, stride)

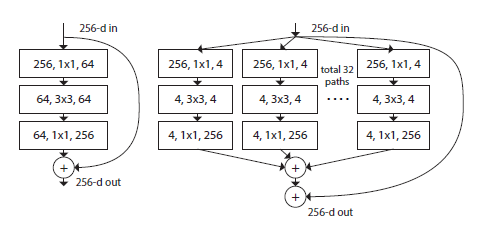
x += shortcut

## ResNeXt

论文链接：https://arxiv.org/abs/1611.05431

这是一篇发表在2017CVPR上的论文，介绍了ResNet网络的升级版：ResNeXt

**基本构件**：



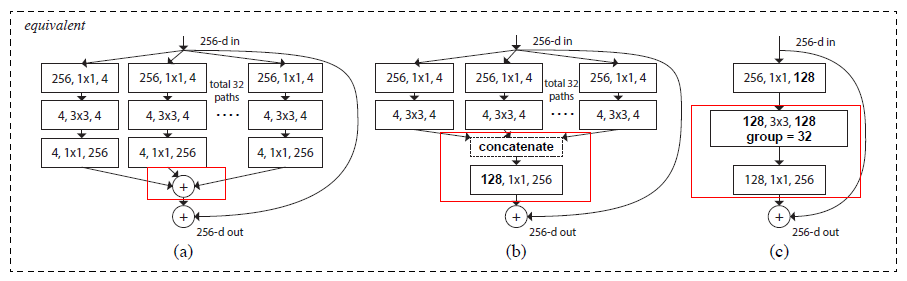
256,1x1,64表示：输入depth=256, 卷积核=1x1，输出depth=64

这看起来跟Inception模块非常相似。因为都采用并联。

不同的是：

* Inception每一个路径互不相同，有而在这个架构中，所有的路径都遵循了相同的拓扑结构。
* 不同路径输出的合并是通过相加来实现的，当然也可以设置为合并，group等多种变种

更多的合并方式：



a 就是前面所说的aggregated residual transformations。

b 则采用两层卷积后 concatenate，再卷积，

c采用的是grouped convolutions，也就是只在3x3这个conv进行并联。

作者在文中明确说明这三种结构是严格等价的，并且用这三个结构做出来的结果一模一样。

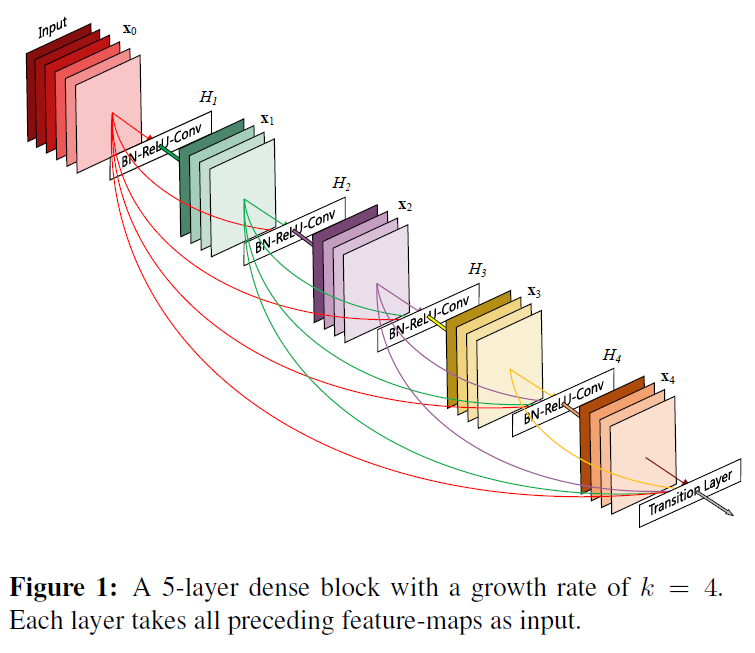
## DenseNet

2016年8月发布。论文 (<https://arxiv.org/abs/1608.06993>)

github: <https://github.com/liuzhuang13/DenseNet>

参考：<http://blog.csdn.net/u012938704/article/details/53468483>

核心组件：dense block:



dense block进一步使用了shortcut connections，将所有的层互相连接起来。在这个新架构中，每一层的输入都包含了所有较早的层的feature maps，而且它的输出被传递至每个后续层。这些feature maps在depth维度上进行 concatenation。

每一层的输入都包含了所有较早的层的feature maps，这个好处是：后面的层的输入包含第一层的feature, 则损失的梯度就能direct传导给第一次，从而减轻梯度消失。

图中的block，有4个卷积层，它一共有4\*(4+1)/2个连接。而传统的CNN，4个卷基层，则只有4个连接。

这些feature maps在depth维度上进行 concatenation，这个后果就是后面的层的输入特征的通道会越来越大，第L层的输入特征的depth= k0+k(L-1)。于是使用了一个“growth rate”(k)的超参数用来控制网络的宽度（即通道数）。（dense block里，stride=1,所以输入输出的size一样。超参数k表示dense block里所有conv层的depth都等于k,即每个conv层的depth=k）

**DenseNet-B**

虽然每个层的out\_depth=k，但是后面层的输入依然很多，第L层的in\_depth= k0+k(L-1)，若L=30, k0=k=32,则=30\*32，这个就非常大了。因此引入了Bottleneck layers 。本质上是引入[1x1] x4k的卷积层来进行降维。 于是:

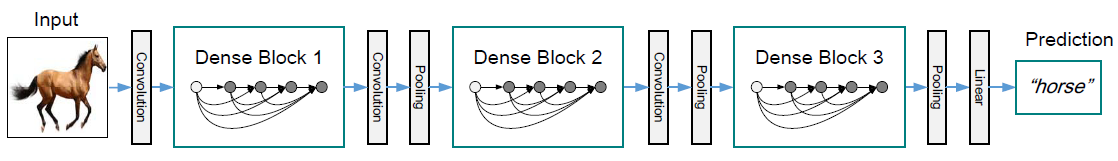
BN−>ReLU−>Conv(3×3) => BN−>ReLU−>Conv(1×1)−>BN−>ReLU−>Conv(3×3)

这样第L=30层的 3x3卷积层的输入就由 30\*32 变成了 4\*32. 减少了很大的参数量。

**DenseNet-C**

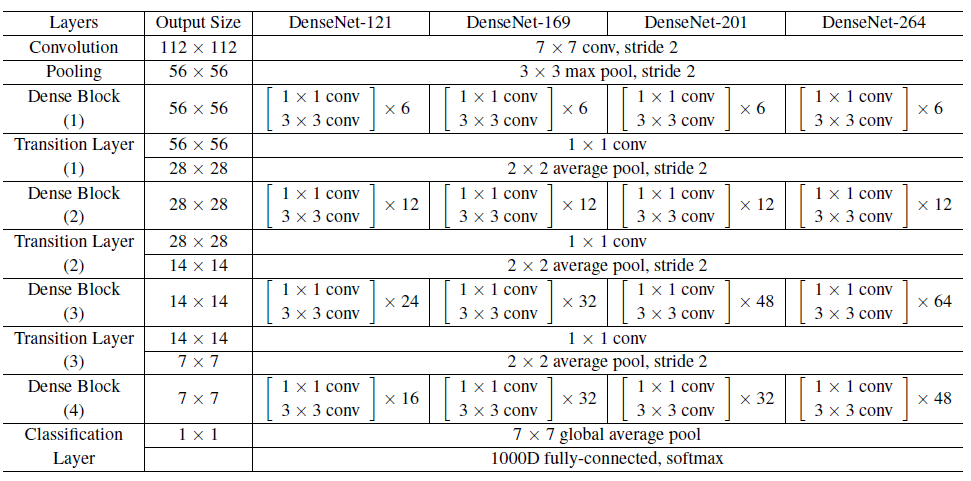
dense block和dense block 通过 conv + pool进行过渡。 conv+pool用来控制 input size,进行compression. 所以叫DenseNet-C

A deep DenseNet with three dense blocks 如下：



即采用Bottleneck,又采用compression，叫**DenseNet-BC**。但一般DenseNet-BC简称DenseNet

以下为论文中给列出的**DenseNet-BC**具体结构，包括DenseNet-121, DenseNet-169等。



图中的 conv= BN-ReLU-Conv , 前3个网络超参数k=32. 最后denseNet-264的k=48

1x1 conv是用来降维的，depth=4k, 作为Bottleneck layer

# R-CNN

region with CNN. 使用cnn对目标的区域进行框定。属于物体检测技术。

图像的相关任务：

图像识别（classfication）和物体检测（localization）是2种任务。**图像识别**做的是判定它是cat还是dog, 而**物体定位**是给出cat在图片中的位置，用框来表示。所以，图像识别是多分类问题,而物体检测是回归问题。物体检测的输出是一个包框（bonding box）。用（x,y,w,h）表示。(x,y)表示一个框的中心点，(w,h)表示宽，高。多物体检测，则输出是 k \* (x,y,w,h).

物体定位是回归问题，回归的输出（x,y,w,h），训练得到的是平移和缩放（dx,dy,dw,dh） 。回归问题的loss采用平方损失，预测框与真实框之间计算。

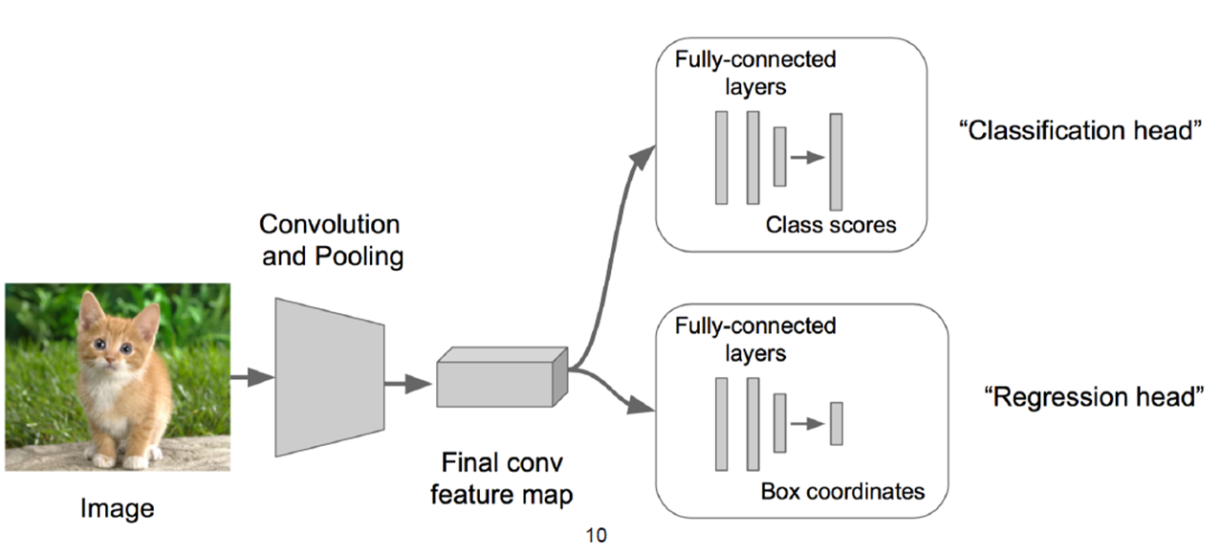
## R-CNN

paper链接：[Rich feature hierarchies for accurate object detection and semantic segmentation ]( <https://arxiv.org/pdf/1311.2524>)

Region CNN(R-CNN)可以说是利用深度学习进行目标检测的开山之作。作者是RBG大神。

**流程&思路：**

* 将一张图像划分为2K个候选区域(proposal)。可以采用选择性搜索技术(selective search)
* 对每个候选区域，使用CNN提取特征。
* 特征后接一个分类器。进行分类识别。
* 特征后接一个回归器。对候选框进行修正（平移和缩放）。



注意这里CNN的目的只是生成特征。CNN在分类器和回归器中都不训练，不进行w更新。

**回归器-位置修正**

目标检测问题的**衡量标准**是重叠面积(), 许多看似准确的检测结果，往往因为候选框不够准确，重叠面积很小。故需要一个位置精修，这里采用的是回归器来实现。

具体是：

对每一类目标，使用一个线性脊回归器(采用L2正则的回归器)进行精修。正则项λ=10000。

特征输入：为CNN pool5层的4096维特征；

输出：为(dx,dy,dw,dh) ，即x,y方向的平移和缩放。

训练样本：判定为本类的候选框中，和真值重叠面积(IOU)大于0.6的候选框。

### Selective Search

**原理**：类似two-steps聚类。

1. 首先，将图像分割为很多小区域。
2. 然后，进行区域合并。合并规则如颜色相近，纹理相近。
3. 最后，从合并过程中产生区域挑选候选区域proposal，采用的是非极大值抑制算法。

**非极大值抑制算法（NMS）：**

RCNN会从一张图片中找出n个可能是物体的矩形框，然后为每个矩形框为做类别分类概率。

例如：从图片中定位一个车辆，最后算法就找出了一堆的方框，我们需要判别哪些矩形框是没用的。非极大值抑制的方法是：先假设有6个矩形框，根据分类器的类别分类概率做排序，假设从小到大属于车辆的概率 分别为A、B、C、D、E、F。

(1) 标记概率最大矩形框F，作为第一个候选框。

(2) 分别判断A~E与F的重叠度IOU是否大于某个设定的阈值;假设B、D与F的重叠度超过阈值，那么就扔掉B、D；

(3) 从剩下的矩形框A、C、E中，选择概率最大的E，作为保留下来第二个候选框。

(4) 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；

(5) 重复。得到所有被保留下来的矩形框。

除了Selective Search, 还有EdgeBoxs算法也能很好的产生候选区域。

### 训练过程

1. CNN预训练。

在IMAGE NET样本集上训练。该样本集仅有图像类别标签，没有图像物体位置标注。

输入是：227×227的ILSVRC训练集图像，输出最后一层为4096维特征->1000类的映射，训练的是CNN参数。

1. fine-tuning，特定样本下的微调。

PASCAL VOC 2007样本集上既有图像中物体类别标签，也有图像中物体位置标签；

|  |  |
| --- | --- |
| 正样本 | 与Ground Truth相交IoU>0.5的候选框【由于Ground Truth太少了】 |
| 负样本 | 与Ground Truth相交IoU≤0.5的候选框 |

采用训练好的AlexNet CNN网络进行PASCAL VOC 2007样本集下的微调，学习率=0.001【lr低，是为了在学习新东西时不至于忘记之前的记忆】

正负比例：mini-batch为32个正样本和96个负样本【由于正样本太少】；

输入是：候选框【由selective search而来】变形后的227×227的图像，

输出是：修改了原来的1000为类别输出，改为21类【20类+背景】，

训练的是CNN的参数，进行参数微调。

1. SVM训练。

|  |  |
| --- | --- |
| 正样本 | Ground Truth |
| 负样本 | 与Ground Truth相交IoU＜0.3的建议框 |

由于SVM是二分类器，需要为每个类别训练单独的SVM；

输入是：候选框通过CNN提取的4096维特征

输出是：为该类的得分，

训练的是SVM权重向量；

1. Bounding-box regression训练。

|  |  |
| --- | --- |
| 样本 | 与Ground Truth相交IoU最大的Region Proposal，并且IoU>0.6的Region Proposal |

回归器：选用的是线性脊回归器（即L2正则的线性模型）。正则项λ=10000。

输入是：候选框通过CNN提取的特征。

输出是：预测的(x,y,w,h)，或者是(dx,dy,dw,dh)

参考：http://blog.csdn.net/wopawn/article/details/52133338

### 疑问&解释

【Q】为什么要将建议框变形为227×227？怎么做？

paper采用AlexNet CNN网络进行CNN特征提取，为了适应AlexNet网络的输入图像大小：227×227，故将所有建议框变形为227×227。

【Q】为什么先ILSVRC 2012数据集，再PASCAL VOC 2007数据集

ILSVRC样本集上用于图片分类的含标注类别的训练集有1millon之多，总共含有1000类；而PASCAL VOC 2007样本集上用于物体检测的含标注类别和位置信息的训练集只有10k，总共含有20类，直接用PASCAL VOC训练容易造成过拟合，因此先利用ILSVRC2012进行预训练。

【Q】正负样本的生成

一张照片我们得到了2000个候选框。然而人工标注的数据一张图片中就只标注了正确的bounding box，我们搜索出来的2000个矩形框也不可能会出现一个与人工标注完全匹配的候选框。因此在CNN阶段我们需要用IOU为2000个bounding box打标签。如果用selective search挑选出来的候选框与物体的人工标注矩形框（PASCAL VOC的图片都有人工标注）的重叠区域IoU大于0.5，那么我们就把这个候选框标注成物体类别（正样本），否则我们就把它当做背景类别（负样本）。

【Q】为什么采用 SVM分类，而不是直接CNN+softmax分类。

都是分类器。SVM的特点是在小样本下也能保持很高的准确率。

在FAST-RCNN中，RGB大神就采用的是softmax.

我猜原因，可能是当时的实验，softmax的效果不如SVM，或者只想到二分类，而没想到多分类。

【Q】分类器和回归器是同时训练吗？

不是，是先进行分类，再进行位置修正。

参考文档：

https://zhuanlan.zhihu.com/p/23006190?refer=xiaoleimlnote

## Fast R-CNN

Fast R-CNN 是针对 R-CNN进行效率上的改进。

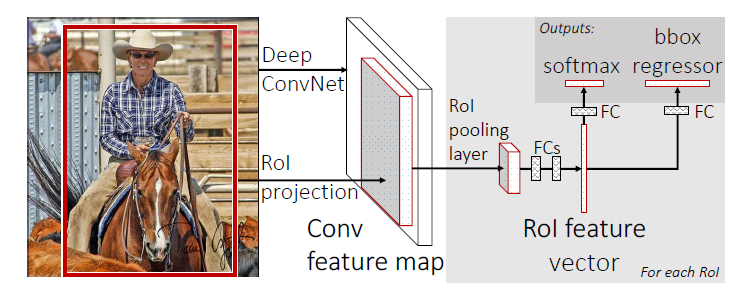
在R-CNN中，每个候选框都要通过CNN提取特征，2k个候选框，CNN就运行了2k次。

Faster R-CNN的**主要改进**就是：

CNN的输入是整张图片，而不是候选框ROI。候选框的特征直接在cnn的conv5层上crop出对应特征。

[原始图片中的ROI如何映射到到feature map?]( https://zhuanlan.zhihu.com/p/24780433)。

这样CNN只计算了一次。



**predict流程**：

* 将一张图像划分为2K个候选区域(proposal)。可以采用选择性搜索技术(selective search)
* 对整张图片，使用CNN提取特征，feature maps。
* 从feature maps 扣选出候选区域对应的特征。
* 通过RoI pooling层使每个RoI生成固定尺寸的feature map。
* 特征后接一个分类器。进行分类识别。
* 特征后接一个回归器。对候选框进行修正（平移和缩放）。

**速度对比**

test一张图片,从RCNN的 50s下降到 2.3s ，剔除候选框生成的2s,则fast rcnn只有0.32s .

### ROI pooling

ROI, region of intrest。 候选框ROI的大小是不一样的。但分类器和回归器的输入必须是同一个尺寸。so, ROI pooling layer的作用就是将 ROI在conv5上crop出的特征 H x W，池化到同一个尺寸 h x w。比如12 x 12 映射到 3 x3 .

与普通pooling不一样的是，pool的输入尺寸大小是不同，而输出要求一样。

### cls – softmax

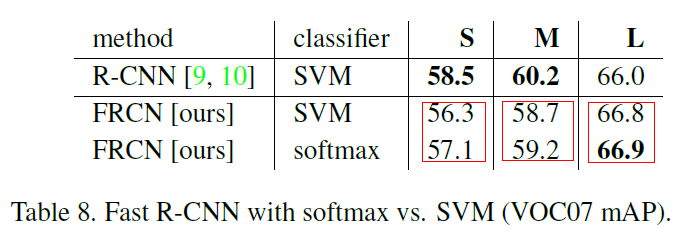
**loss**

http://img.blog.csdn.net/20170111165447192?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbGlub2x6aGFuZw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

pl 即为label所对应概率（即只考虑label对应的概率，不考虑其他分类的概率），pl = 1时，计算结果Loss为0， 越小，Loss值越大（0.01对应Loss为2）。

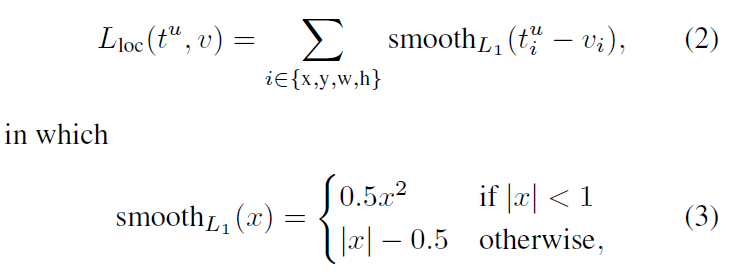
**softmax**

R-CNN中采用是svm进行二分类，Fast R-CNN中则采用了softmax进行多分类。在FRCN中使用softmax的效果要好于svm的二分类。



### bbox Regression

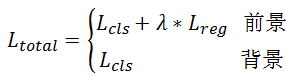
有意思是，regressor的loss不是L2，而是一个平滑的L1



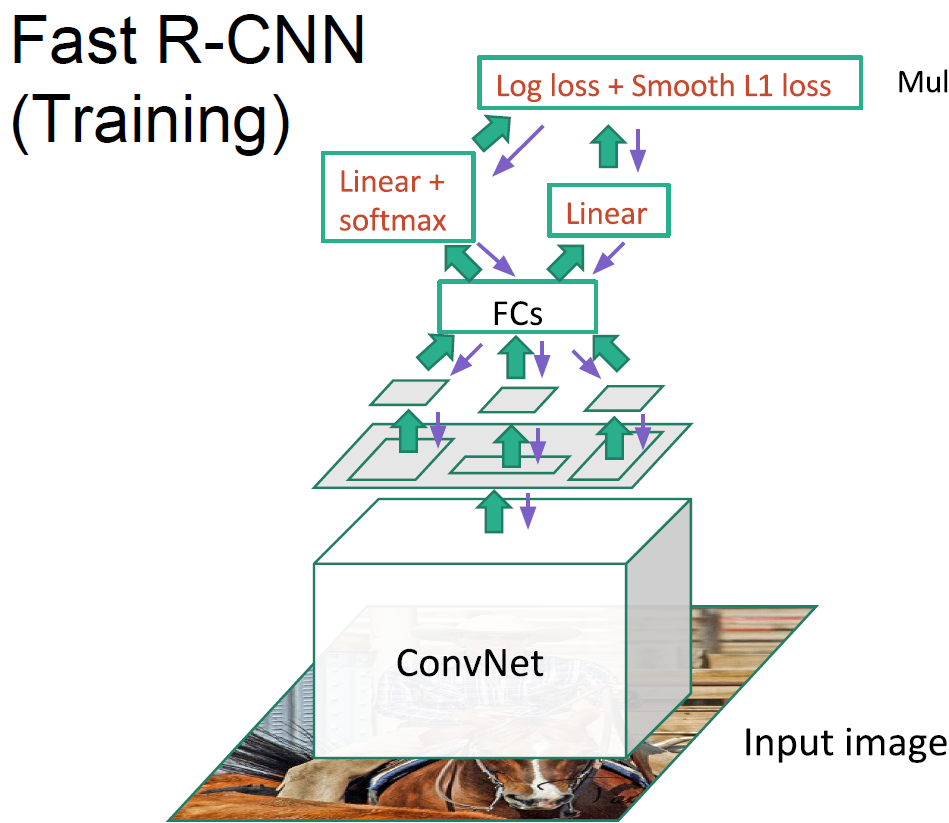
### traning

**loss混合**

将分类loss和回归loss按一定比例混合混合。



**traning**：

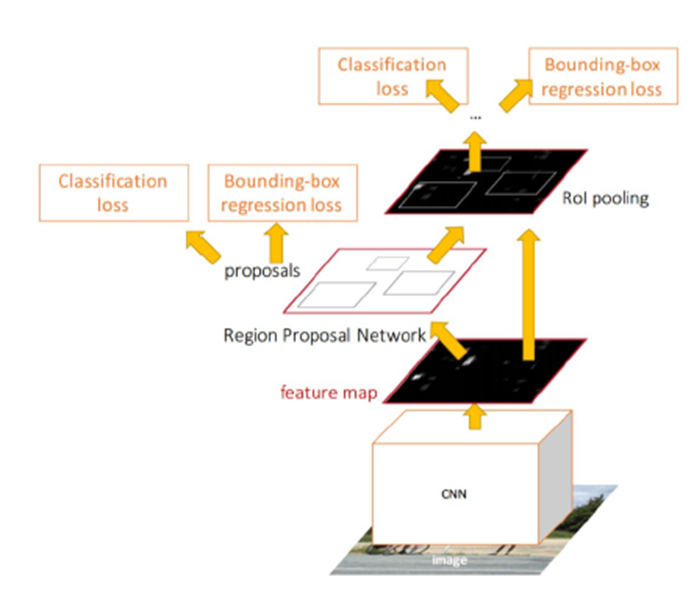


## faster R-CNN

在Fast R-CNN的改进：

使用RPN(Region Proposal Network)代替原来的Selective Search方法产生建议窗口；

faster RCNN可以看作2部分： RPN + fast RCNN



流程：

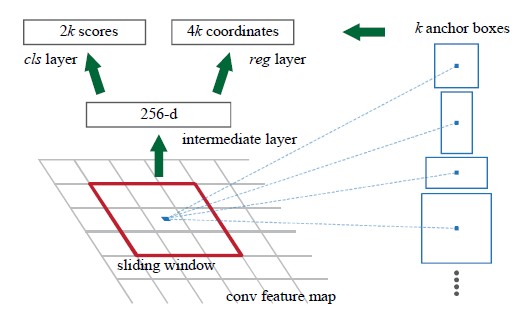
* 将整张图片输入CNN，进行特征提取；
* 用RPN生成建议窗口(proposals)，每张图片生成300个建议窗口；
* 把建议窗口映射到CNN的最后一层卷积feature map上；
* 通过RoI pooling层使每个RoI生成固定尺寸的feature map；
* classfication和bbox regressiong.

### RPN

在Fast-CNN中，Selective Search是耗时的瓶颈(CPU上大概2s一张)。而RPN只需要10ms。

首先，RPN是一个独立的网络，RPN的输入是 feature maps, 输出是proposals的分类得分和回归得分，根据得分留下得分高的候选框。。

RPN的结构如下：

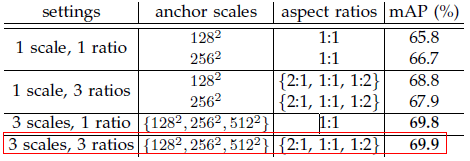


**anchors**

anchors: 锚点. 根据这个点可以知道它对应的k个anchor boxs. 考虑feature map的size=W\*H,则一共有 W\*H\*k个anchor boxs .

对每个anchor box计算得分，保留得分top300的anchor boxs作为建议框。

paper中，k的设置实验数据如下：



不考虑 anchors 时,则rpn的结构如下：



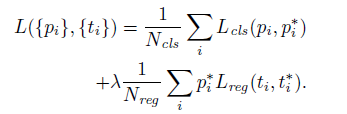
cls:2 , 2表示二分类，object or not object for each proposal。

考虑 anchors，则：



**RPN的loss**

rpn的loss是由classification loss （即softmax loss）和regression loss （即L1 loss）按一定比重组成的.

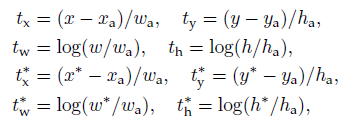


Here, i is the index of an anchor in a mini-batch and pi is the predicted probability of anchor i being an

object. The ground-truth label pi\* is 1 if the anchor is positive, and is 0 if the anchor is negative.

(i是anchor的编号，pi是预测值， pi\*是实际值)

其中，t



Variables x, xa, and x\* are for the predicted box, anchor box, and groundtruth box respectively (likewise for y;w; h).

**RPN的traning**

* 一个mini-bath选取256个anchor box , 正负比例1：1，如果正样本不足，则用负样本补足。
* rpn层的参数初始化，std=0.01的高斯分布。
* joint traning.

**整个traning的流程：**

1. RPN网络traning
2. RPN生成建议框，并映射到feature map上
3. fast RCNN traning

## Mask R-CNN

参考：<http://weixin.niurenqushi.com/article/2017-03-29/4805787.html>

<https://github.com/CharlesShang/FastMaskRCNN>

**实例分割**

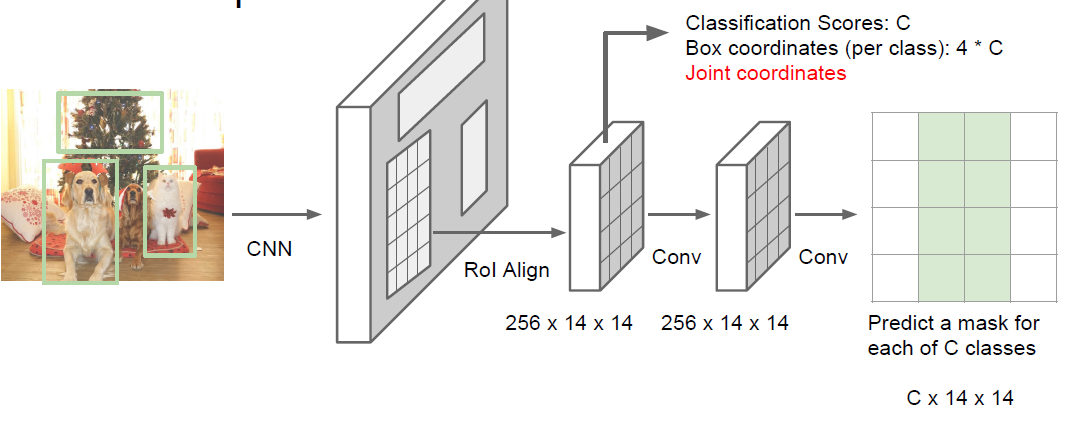
**物体检测**只是将图片里的人框出来，**实例分割**是把每个人清晰的分割出来。

实例分割工作之所以很难，是因为它要完成2个任务：目标检测+实例分割。目标检测将每个实例进行分类，并在边界框内进行定位，然后再进行语义分割（将每个像素进行分类的工作）（即实例分割）。

**Mask R-CNN思想：**

Faster R-CNN，对于每个ROI，它有两个分支(branch)，一个是classification，一个是bbox regression。在此基础上，Mask R-CNN的方法是增加第三个分支：分割掩码（segmentation masks）。 mask branch是一个小型的FCN(Fully Convolutional Network), 以像素对像素的方式预测分割掩码。

**mask**：对目标的像素位置打上一个统一的值，如1，非目标的像素留白。

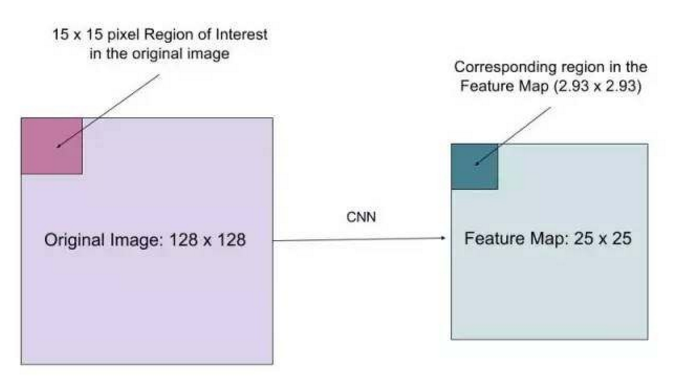


### ROI Align

**用RoIAlign替换Roi pooling**

在FasterR-CNN中，RoIPool 选择的特征图的区域与原始图像的区域略不对齐。因为图像分割需要像素级特异性，不像边框，这自然地导致不准确。于是，提出RoIAlign，它可以精确地对应原始图像的区域。这是必要的，因为像素级分割需要比边界框更细粒度的对齐。

【Q】如何准确地将原始图像的相关区域映射到特征图上？



想象一下，我们有一个尺寸大小为 128x128 的图像和大小为 25x25 的特征图。想象一下，我们想要的是与原始图像中左上方 15x15 像素对应的区域（见上文）。我们如何从特征图选择这些像素？

我们知道原始图像中的每个像素对应于原始图像中的25/128像素。要从原始图像中选择 15 像素，我们只需选择 15 \* 25/128=2.93 像素。

在 RoIPool，我们会舍弃一些，只选择 2 个像素，导致轻微的错位。然而，在 RoIAlign，我们避免了这样的舍弃。相反，我们使用双线性插值来准确得到 2.93 像素的内容。这很大程度上，让我们避免了由 RoIPool 造成的错位。

RoIAlign的重大影响：

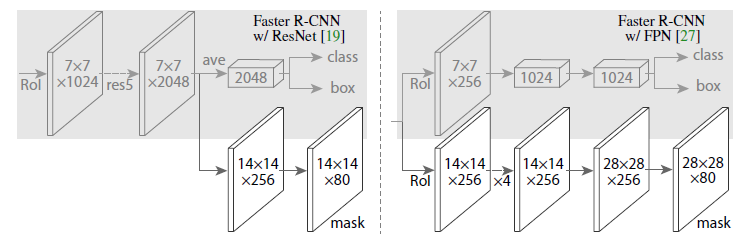
* 它能够相对提高10%到50%的掩码精确度（Mask Accuracy），这种改进可以在更严格的location指标下得到更好的结果。
* 第二，我们发现它对分割掩码和类别预测都很重要：分割掩码指我们为每个类别分别预测了一个二元掩码（目标为1，背景为0）；类别预测指，不进行类别间的计算，只是依赖网络的ROI 分类分支去做类别预测。

### LOSS



* ，对于每一个ROI，mask分支的输出是维, k表示k个类别，mxm表示的是区域的size.(如7x7，每个ROI的输出是一样的),矩阵的填充值是二元掩码（即1和0）。
* 的计算，对每个像素进行sigmoid,定义是平均二分类交叉熵。
* 只会在正确分类的mask上进行计算，而不会计算错误分类的mask .例如：有3类（猫，狗，人），检测得到当前ROI属于“人”这一类，只计算“人”这一分支的。

### 测试



这里，7x7 => 14 x14 ,采用的是反卷积。ksize=2x2, stride=2.

Mask RCNN 分支具有很好的泛化能力 - 可以和多种 RCNN框架结合，如上图。

### 疑问

【Q】的计算，对每个像素进行sigmoid,定义是平均二分类交叉熵。是对输出的结果进行sigmoid，还是对原像素进行sigmoid？ 或者说，输出是0 or 1 ?

【Q】Why K个mask？通过对每个 Class 对应一个 Mask

通过结果对比来看，也就是作者所说的 Decouple 解耦，要比多分类 的 Softmax 效果好很多。

