# Caffe (CNN, deep learning )

Caffe -----------Convolution Architecture For Feature Embedding (Extraction)

# Caffe 是?

(http://caffe.berkeleyvision.org/tutorial/layers.html)

CNN (Deep Learning) 工具箱

C++ 语言架构

CPU 和GPU 无缝交换: Caffe::set\_mode(Caffe::GPU);

Python 和matlab的封装

# 为什么要用Caffe

运算速度快。简单 友好的架构 用到的一些库：

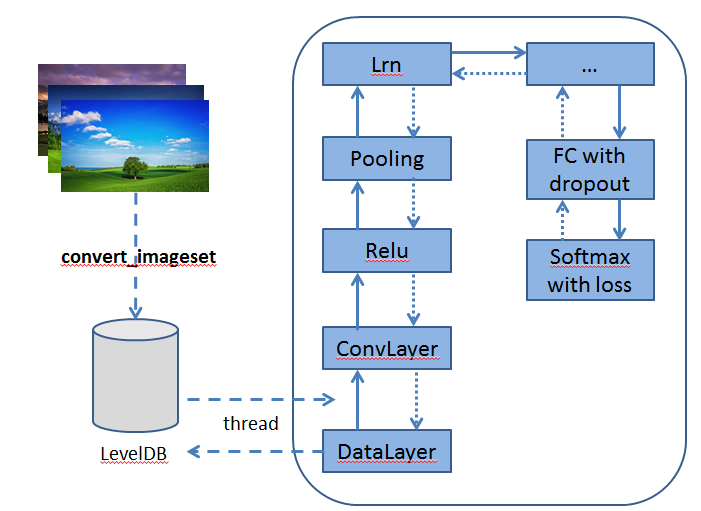
Google Logging library (Glog): 一个C++语言的应用级日志记录框架，提供了C++风格的流操作和各种助手宏.

lebeldb(数据存储)： 是一个google实现的非常高效的kv数据库，单进程操作。

CBLAS library（CPU版本的矩阵操作）

CUBLAS library (GPU 版本的矩阵操作)

# Caffe 架构



# 预处理图像的leveldb构建

输入：一批图像和label （2和3）

输出：leveldb （4）

指令里包含如下信息：

1）conver\_imageset （构建leveldb的可运行程序）

2）train/ （此目录放处理的jpg或者其他格式的图像)

3）label.txt (图像文件名及其label信息)

4）输出的leveldb文件夹的名字

5）CPU/GPU (指定是在cpu上还是在gpu上运行code)

# CNN网络配置文件

1）Imagenet\_solver.prototxt （包含全局参数的配置的文件）

2）Imagenet.prototxt （包含训练网络的配置的文件）

3）Imagenet\_val.prototxt （包含测试网络的配置文件）

## Caffe的网络定义

Caffe中的网络都是有向无环图的集合，可以直接定义：

name:"dummy-net"

layers {name: "data" …}

layers {name: "conv" …}

layers {name: "pool" …}

layers {name: "loss" …}

数据及其导数以blobs的形式在层间流动。

## Caffe的各层定义

Caffe层的定义由2部分组成：层属性与层参数，例如

name:"conv1"

type:CONVOLUTION

bottom:"data"

top:"conv1"

convolution\_param{

num\_output:20

kernel\_size:5

stride:1

weight\_filler{

type: "xavier"

}

}

这段配置文件的前4行是层属性，定义了层名称、层类型以及层连接结构（输入blob和输出blob）；而后半部分是各种层参数。

## Blob

Blob是用以存储数据的4维数组，例如

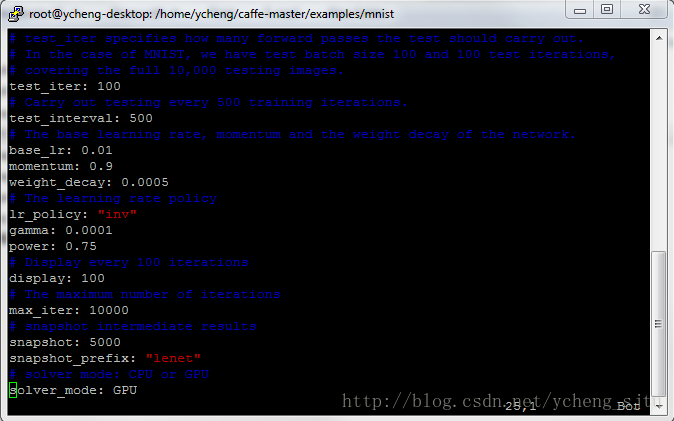
对于数据：Number\*Channel\*Height\*Width

对于卷积权重：Output\*Input\*Height\*Width

对于卷积偏置：Output\*1\*1\*1

## 训练网络

网络参数的定义也非常方便，可以随意设置相应参数。甚至调用GPU运算只需要写一句话：solver\_mode:GPU



## Caffe的安装与配置

Caffe需要预先安装一些依赖项，首先是CUDA驱动。不论是CentOS还是Ubuntu都预装了开源的nouveau显卡驱动（SUSE没有这种问题），如果不禁用，则CUDA驱动不能正确安装。以Ubuntu为例，介绍一下这里的处理方法，当然也有其他处理方法。

在sudo vi/etc/modprobe.d/blacklist.conf

增加一行 ：blacklist nouveau

sudoapt-get --purge remove xserver-xorg-video-nouveau #把官方驱动彻底卸载：

sudoapt-get --purge remove nvidia-\*#清除之前安装的任何NVIDIA驱动

sudo service lightdm stop#进命令行，关闭Xserver

sudokill all Xorg

安装了CUDA之后，依次按照官网提示（ <http://caffe.berkeleyvision.org/installation.html> ）安装BLAS、OpenCV、Boost即可。

## Caffe跑跑MNIST试试

在Caffe安装目录之下，首先获得MNIST数据集：

#cddata/mnist

#sh get\_mnist.sh

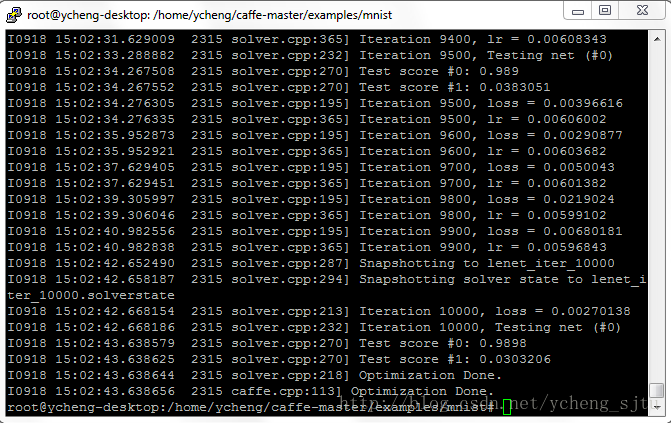
生成mnist-train-leveldb/ 和 mnist-test-leveldb/，把数据转化成leveldb格式：

# cd examples/lenet

#sh create\_mnist.sh

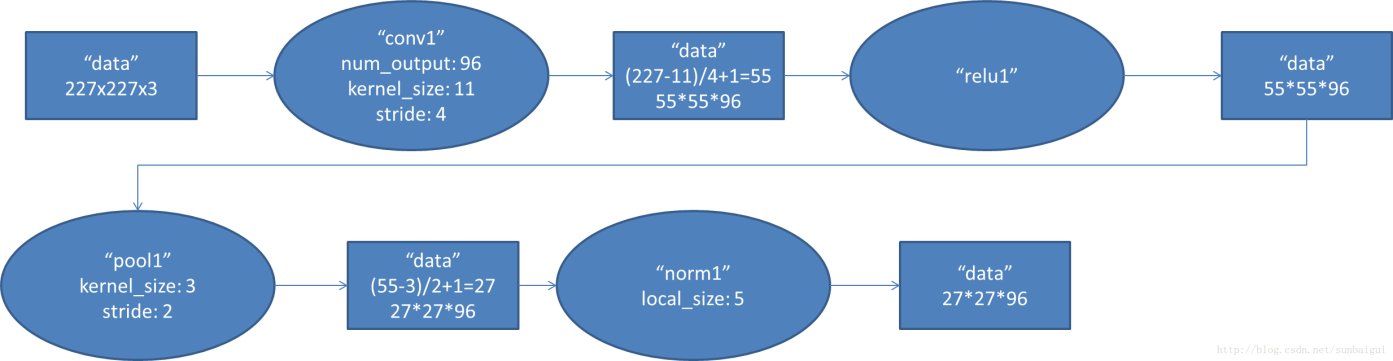
训练网络:

</pre><pre name="code" class="cpp">#sh train\_lenet.sh

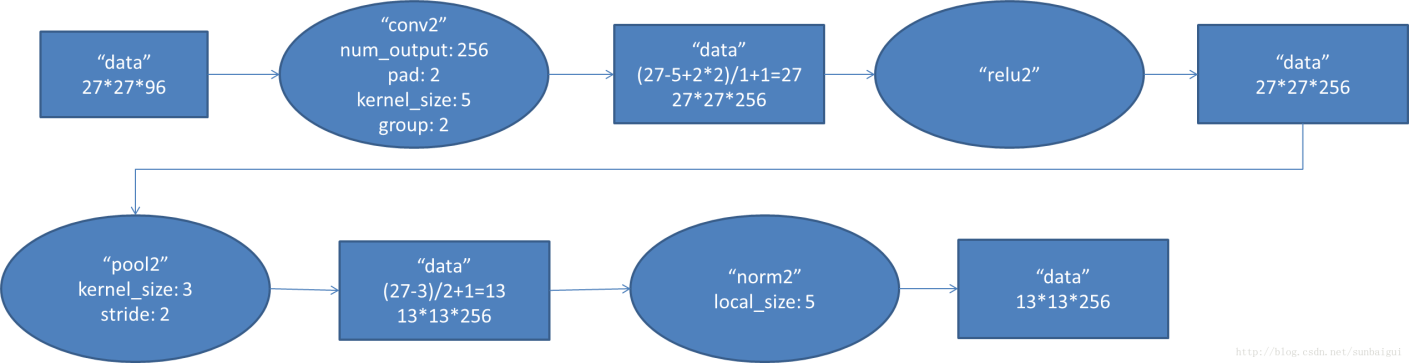


## 各个层进行详细的解读（训练阶段）

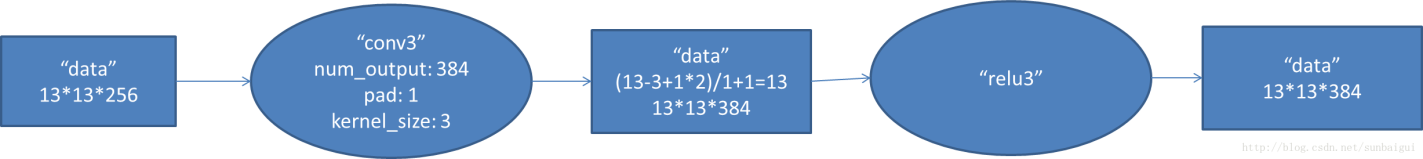
1. conv1阶段DFD（data flow diagram）：



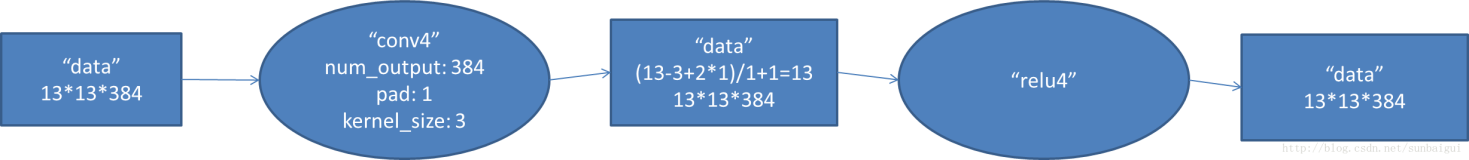
1. conv2阶段DFD（data flow diagram）：



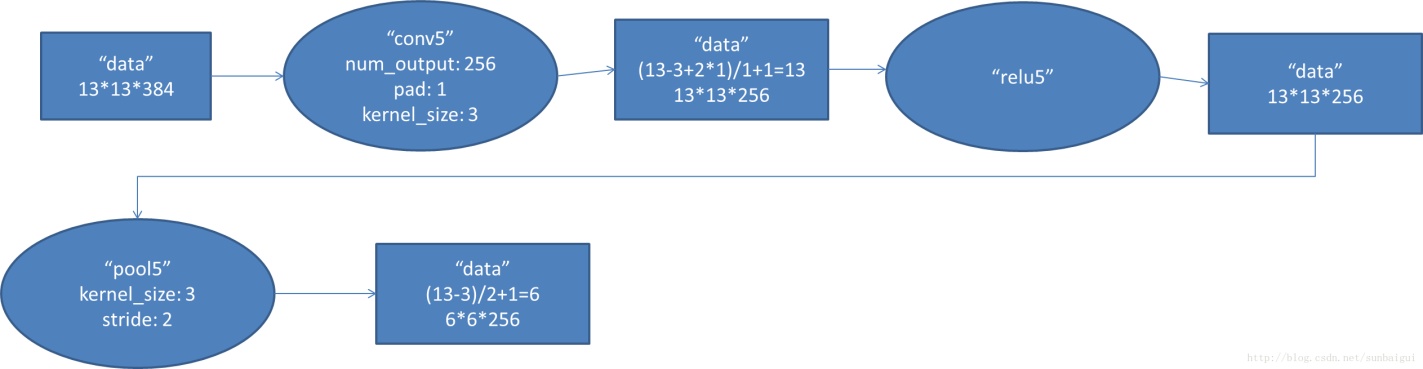
1. conv3阶段DFD（data flow diagram）：



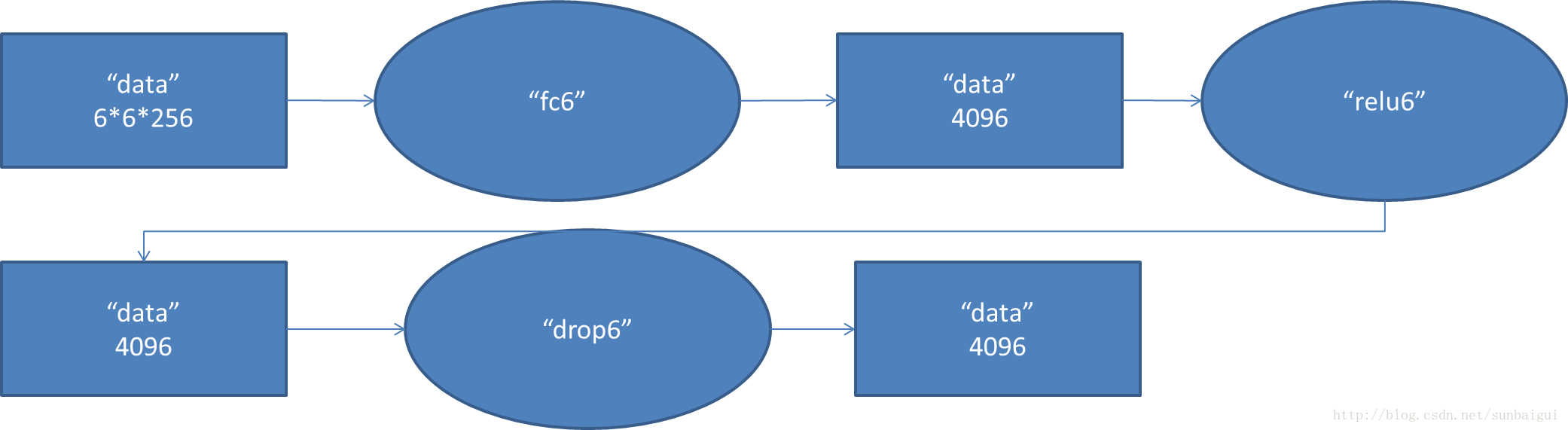
1. conv4阶段DFD（data flow diagram）：



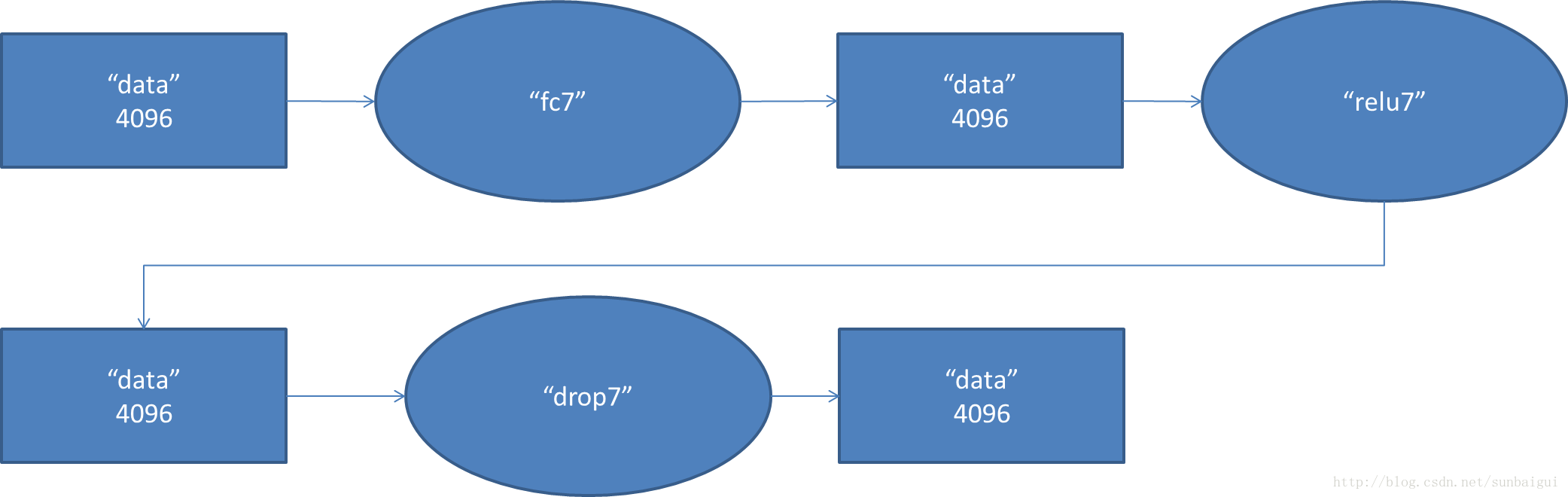
1. conv5阶段DFD（data flow diagram）：



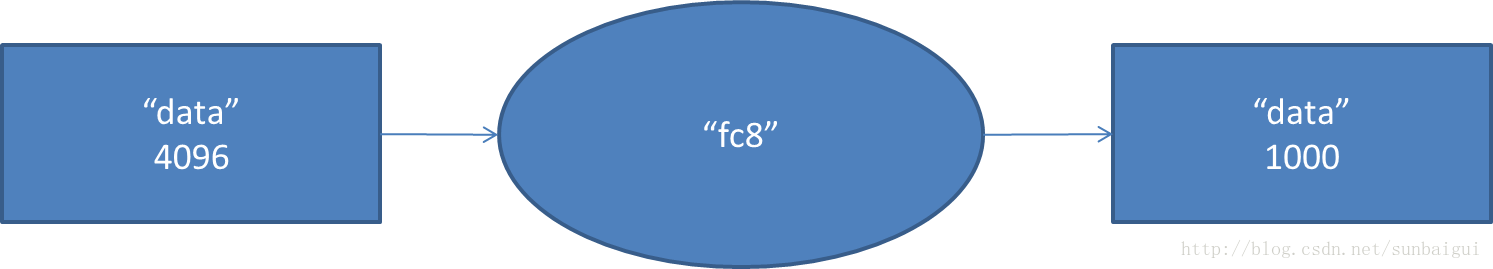
1. fc6阶段DFD（data flow diagram）：



1. fc7阶段DFD（data flow diagram）：



1. fc8阶段DFD（data flow diagram）：



网络总共有8层（每一层都带有weight）；

前五层是卷积层，后三层是全连接层。最后一个全连接层，输入到一个1000-way softmax，产生一个在1000类别上一个分布。

网络的目标:   最大化multinomial logostic regression objective，等价于最大化the **average** across training cases of the log-probability of the correct label under the **prediction distribution**.

第2 4 5 卷积层的kernels，只与位于同一块GPU上的前一层相连接。

第3个卷积层的kernels，与第2层上的所有的kernel maps相连接。

全连接层中的neurons，与前一层中的所有neurons相连接。

ReLU Nonlinearity :应用在所有层。

第1 2层采用了response normalization策略。

第1 2 5 层采用了max-pooling策略。

第1层是作用在224\*224\*3的输入图像上，96个kernels（大小：11\*11\*3），步长：4 pixels

第2层是 256 kernels（大小：5\*5\*48）

第3 4 5层没有利用任何intervening的pooling或者normalization

第3层有384 kernels（大小：3\*3\*256），其输入是第二层normalizated and pooled的输出。

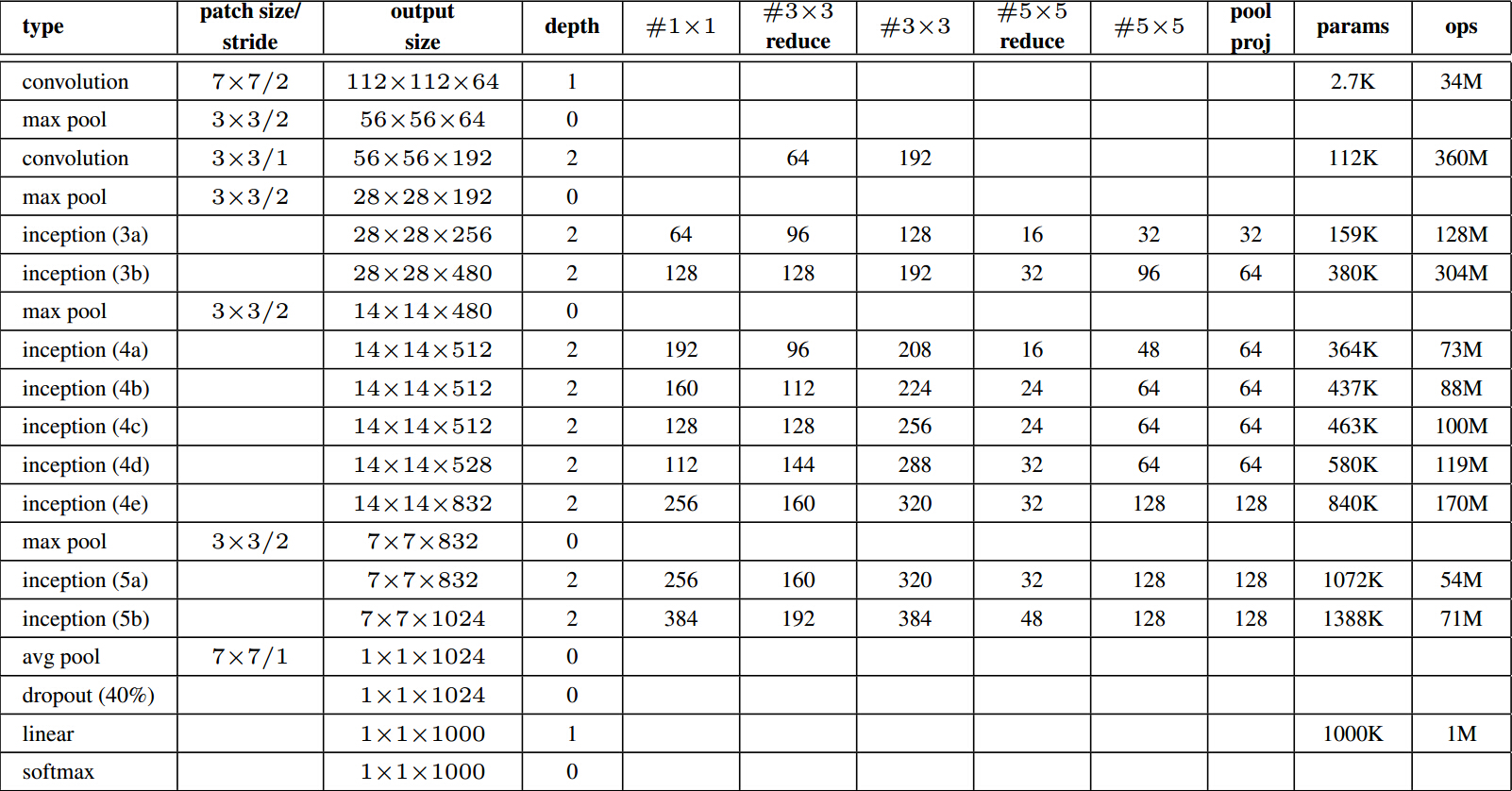
第4层是384 kernels (大小：3\*3\*192)

第5层是256 kernel (大小：3\*3\*192)

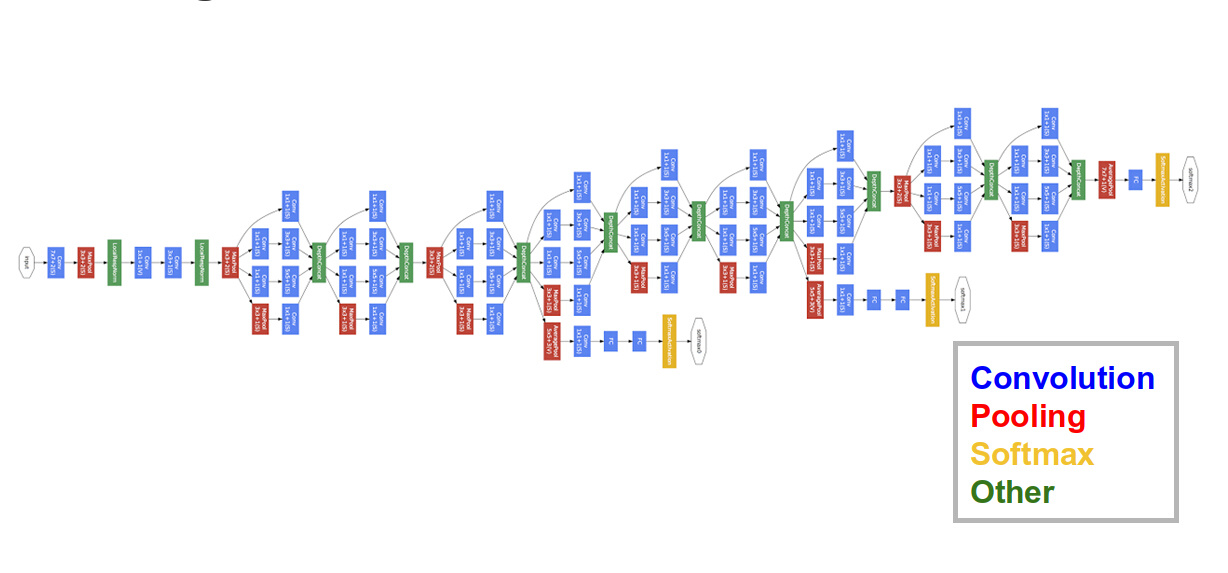
全连接层有4096个neurons。

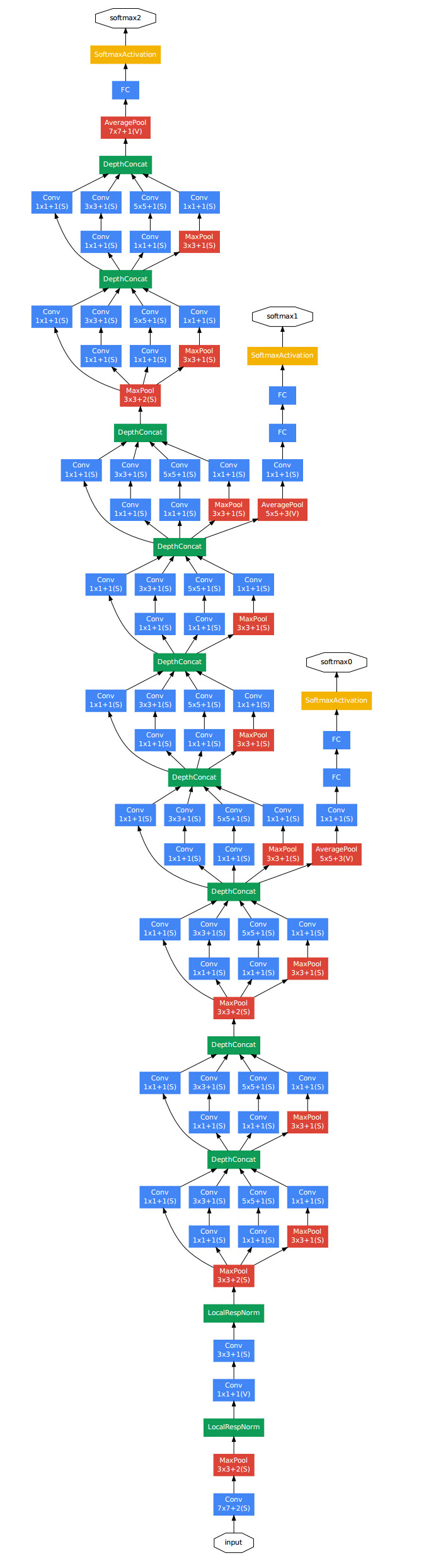
# GOOGLENET

## GoogLeNet

[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140925171057.jpg)表1

#3\*3 reduce代表3\*3卷积前1\*1滤波器的个数。所有的reduction/projection层都利用了修正线性激活。网络包含22层带参数层和5层不含参数的pooling层。总共有约100层。分类前的pooling是基于[12]，只是我们利用了不同的线性层。有利于精细的调节网络，但不期望它有多出色。从全连接层move到下采样会将Top1准确率提高0.6%，但是dropout仍然需要。

[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140926161943.jpg)图1

[](http://www.gageet.com/wp-content/uploads/2014/09/googlenet.jpg)图2

在浅层网络来说，相对中间的网络产生的特征非常有辨识力。在这些层中增加一些额外的分类器，能个增加BP的梯度信号和提供额外的正则化。这些分类器将小德卷积网络放在4a和4b的输出上。在训练过程中，损失会根据权重叠加，而在测试时丢弃。

## Training Methodology

利用[4]提供的分布式机器学习系统和数据平行。用数个高端GPU，一周达到收敛。利用[17]异步随机梯度下降，0.9动量，学习率每八个周期下降4%。最后用Polyak averaging [13]来创建最后用来测试的模型。

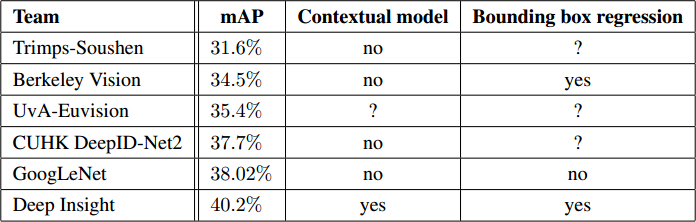
采样变化很大。[8]的光度扭曲有助于对付过拟合。还进行了随机插入。

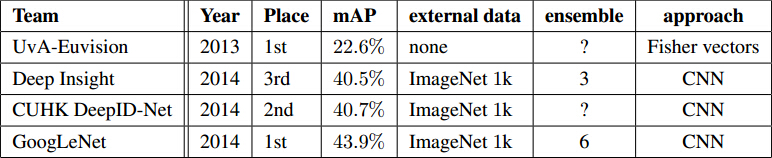
## ILSVRC 2014 Classification Challenge Setup and Results

①训练了7个网络，只是采样方法和随机输入图像不同，取综合值。

②将图像的短边缩放成4中：256,288,320,352。取图像的上中下块。每块取四个角、中间的224\*224和将其缩放到224\*224以及它们的镜像。结果是4\*3\*6\*2=144，即每个图像采样144块输入图。但可能实际生产中不能应用。

③maxpooling和在分类器平均，最后的结果都不如简单的平均好。

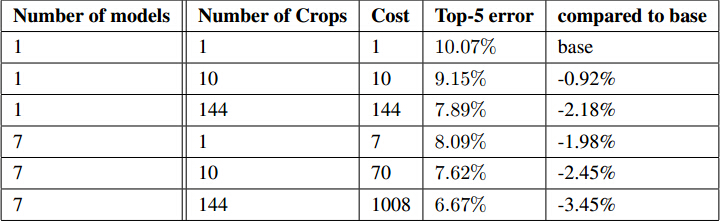
[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140925214634.jpg) 表2

[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140925214621.jpg) 表3

## ILSVRC 2014 Detection Challenge Setup and Results

方法与[6]的R-CNN很像，但增加了Inception模块。结合了用multi-box [5] 的方法和Selective Search [20] 来提高定位框的召回率。这让从Selective Search [20] 得到的结果减半。再加上200个[6]的方法，总共[6]占60%，可以将覆盖率从92%提高到93%。上述方法可以将准确率相对单一模型提高1%。在分类区域的时候利用6个卷积网络科将准确率从40%提高到43.9%。

表4最后GoogLeNet最好，比去年大了接近一倍。表5表示GoogLeNet在单模型的情况下，只比最好的低0.3%，但那个用了3个模型，而GoogLeNet在多模型的情况下结果好得多。

[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140925213620.jpg) 表4

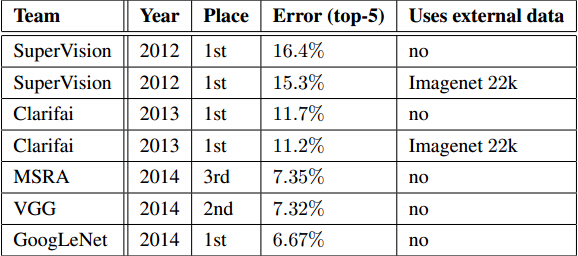
[](http://www.gageet.com/wp-content/uploads/2014/09/QQ%E6%88%AA%E5%9B%BE20140925213608.jpg)

表5

## Conclusions

将最佳稀疏结构稠密化是一种有效提高CV神经网络的方法。优点是只适度增加计算量。本文检测方法不利用上下文，不会定位框回归，证明了Inception方法的强壮。本文提供了一个将稀疏变稠密的途径。