



may0324的博客

☰ 目录视图

☰ 摘要视图

RSS 订阅

个人资料



无眠栀



访问： 45413次

积分： 697

等级：

BLOG > 3

排名： 千里之外

原创： 22篇 转载： 2篇

译文： 0篇 评论： 65条

文章搜索

🔍

文章分类

- [iOS 学习](#) (7)
- [MAC系统学习](#) (5)
- [linux学习](#) (4)
- [linux realsense](#) (2)
- [ROS](#) (1)
- [OpenCV 人脸检测](#) (1)
- [web开发学习](#) (1)
- [深度学习](#) (6)
- [3D手势识别](#) (2)
- [图像处理](#) (1)

文章存档

- [2017年07月](#) (1)
- [2017年06月](#) (1)
- [2017年05月](#) (1)
- [2017年02月](#) (1)
- [2016年12月](#) (1)

展开

阅读排行

- [Tensorflow 离线安装跳坑](#) (9319)
- [windows下安装numpy,sc](#) (7737)
- [ubuntu 下安装intel realsense](#) (4893)
- [Deep Compression阅读](#) (4699)
- [ROS下使用intel Realsense](#) (4540)
- [caffe python接口配置总结](#) (3686)

📖 赠书 | 异步2周年,技术图书免费选

📖 程序员8月书讯

📖 项目管理+代码托管+文档协作，开发更流畅

Deep Compression阅读理解及Caffe源码修改

标签： [cnn](#) [模型压缩](#) [源码](#) [caffe](#)

2016-10-26 20:07

4711人阅读

[评论\(39\)](#)

[收藏](#)

[举报](#)

☰ 分类： [深度学习 \(5\)](#) ▼

❗ 版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?)

[+]

更新：

没想到这篇文章写出后有这么多人关注和索要源码，有点受宠若惊。说来惭愧，这个工作当时做的很粗糙，源码修改的比较乱，所以一直不太好拿出手。最近终于有时间整理了一下代码并开源出来了。关于代码还有以下几个问题：

- 1.在.cu中目前仍然是调用cpu_data接口，所以可能会增加与gpu数据交换的额外耗时，这个不影响使用，后面慢慢优化。~(已解决)
- 2.目前每层权值修剪的比例仍然是预设的，这个比例需要迭代试验以实现在尽可能压缩权值的同时保证精度。所以如何自动化选取阈值就成为了后面一个继续深入的课题。
- 3.直接用caffe跑出来的模型依然是原始大小，因为模型依然是.caffemodel类型，虽然大部分权值为0且共享，但每个权值依然以32float型存储，故后续只需将非零权值及其下标以及聚类中心存储下来即可，这部分可参考作者论文，写的很详细。
- 4.权值压缩仅仅压缩了模型大小，但在前向inference时不会带来速度提升。因此，想要在移动端做好cnn部署，就需要结合小的模型架构、模型量化以及NEON指令加速等方法来实现。

代码开源在github

<https://github.com/may0324/DeepCompression-caffe>

最近又转战CNN模型压缩了。。。 （我真是一年换N个坑的节奏），阅读了HanSong的15年16年几篇比较有名的论文，启发很大，这篇主要讲一下Deep Compression那篇论文，因为需要修改caffe源码，但网上没有人po过，这里做个第一个吃螃蟹的人，记录一下对这篇论文的理解和源码修改过程，方便日后追本溯源，同时如果有什么纰漏也欢迎指正，互相交流学习。

这里就从Why-How-What三方面来讲讲这篇文章。

Why

首先讲讲为什么CNN模型压缩刻不容缓，我们可以看看这些有名的caffe模型大小:

1. LeNet-5 **1.7MB**
2. AlexNet **240MB**
3. VGG-16 **552MB**

LeNet-5是一个简单的手写数字识别网络，AlexNet和VGG-16则用于图像分类，刷新了ImageNet竞赛的成绩，但是就其模型尺寸来说，根本无法移植到手机端App或嵌入式芯片当中，就算是想通过网络传输，较高的带宽占用率也让很多用户望尘莫及。另一方面，大尺寸的模型也对设备功耗和运行速度带来了巨大的挑战。随着深度学习的不断普及和caffe,tensorflow,torch等框架的成熟，促使越来越多的学者不用过多地去花费时间在代码开发上，而是可以毫无顾及地不断设计加深网络，不断扩充数据，不断刷新模型精度和尺寸，但这样的模型距离实用却仍是望其项背。

在这样的情形下，模型压缩则成为了亟待解决的问题，其实早期也有学者提出了一些压缩方法，比如weight prune(权值修剪)，权值矩阵SVD分解等，但压缩率也只是冰山一角，远不能令人满意。今年standford的

caffe binaryproto 与 npy (1916)

R-CNN阅读笔记 (1512)

windows下编译dlib (1366)

win7下安装ubuntu14.04 (791)

评论排行

Deep Compression阅读 (39)

ubuntu 下安装intel realsense (11)

ROS下使用intel Realsense (9)

caffe binaryproto 与 npy (2)

windows下安装numpy,scipy (1)

CoreML学习——转换caffe模型 (1)

Tensorflow 离线安装跳坑 (1)

c++ objective c 混合编程 (1)

Semi-Automatic 3D Annotation (1)

MAC 下安装MATLAB2014 (0)

推荐文章

* CSDN日报20170824——《你为什么跳槽？真正原因找到了吗？》

* Linux的任督二脉：进程调度和内存管理

* 秒杀系统的一点思考

* TCP网络通讯如何解决分包粘包问题

* 技术与技术人员价值

* GitChat:人工智能 | 除了深度学习，机器翻译还需要啥？

最新评论

ROS下使用intel Realsense摄像头: aibixie1637: 楼主，初学ros 问一个特别傻的问题，下了realsense的源码包，发现都是c++的代码，，如果要...

caffe binaryproto 与 npy相互转换: @l_ml_m_lm_m: 你的文件mean.binaryproto是放在当前路径下的么

caffe binaryproto 与 npy相互转换: @l_ml_m_lm_m: 为什么我运行的时候一直提示没有mean.binaryproto,路径和名字都没有错误，但运行事一直有...

Deep Compression阅读理解及Caffe源码修改: @feiwudexue123: 请参考文章中的github源码

Deep Compression阅读理解及Caffe源码修改: @xiyan1111: 抱歉，暂时没看到

Deep Compression阅读理解及Caffe源码修改: @taoliuliutao: 这个比较有争议，论文没有讲清楚，事实上在当时做的时候是考虑过每次更新聚类...

Deep Compression阅读理解及Caffe源码修改: @xiyan1111: 楼主，求该模型的tensorflow的代码，或者您知道哪里能找到吗

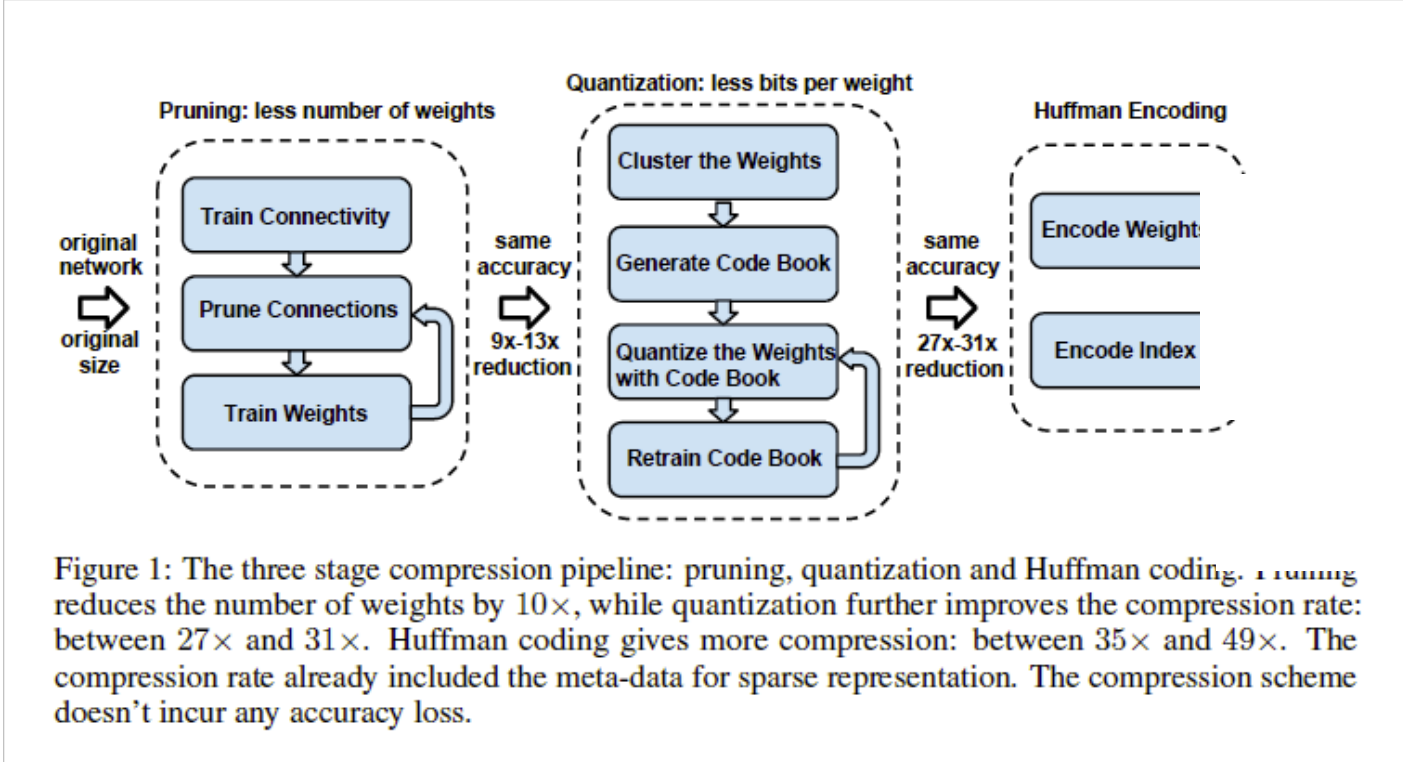
Deep Compression阅读理解及Caffe源码修改: @taoliuliutao: 我也持6楼的观点，Quantization code中，每次前传都将聚类中心赋值给weights，但...

CoreML学习——转换caffe模型: ccnyou: deploy.prototxtmean.binaryproto 请问这两个文件哪里来的？我下载了git...

HanSong的ICLR的一篇论文Deep Compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding一经提出，就引起了巨大轰动，在这篇论文工作中，他们采用了3步，在不损失（甚至有提升）原始模型精度的基础上，将VGG和Alexnet等模型压缩到了原来的35~49倍，使得原本上百兆的模型压缩到不到10M，令深度学习模型在移动端等的实用成为可能。

How

Deep Compression 的实现主要有三步，如下图所示：



包括Pruning（权值修剪），Quantization（权值共享和量化），Huffman Coding（Huffman编码）。

1.Prunning

如果你调试过caffe模型，观察里面的权值，会发现大部分权值都集中在-1~1之间，即非常小，另一方面，神经网络的提出就是模仿人脑中的神经元突触之间的信息传导，因此这数量庞大的权值中，存在着不可忽视的冗余性，这就为权值修剪提供了根据。pruning可以分为三步：

- step1. 正常训练模型得到网络权值；
- step2. 将所有低于一定阈值的权值设为0；
- step3. 重新训练网络中剩下的非零权值。

经过权值修剪后的稀疏网络，就可以用一种紧凑的存储方式CSC或CSR（compressed sparse column or compressed sparse row）来表示。这里举个栗子来解释下什么是CSR

假设有一个原始稀疏矩阵A

A	=	4.0	1.0	0.0	0.0	2.5
		0.0	4.0	1.0	0.0	0.0
		0.0	1.0	4.0	0.0	1.0
		0.0	0.0	1.0	4.0	0.0
		2.5	0.0	0.0	0.5	4.0

CSR可以将原始矩阵表达为三部分，即AA,JA,IC

AA	=	4.0	1.0	2.5	4.0	1.0	1.0	4.0	1.0	1.0	4.0	2.5	0.5	4.0
JA	=	1	4	6	9	11	14							
IC	=	1	2	5	2	3	2	3	5	3	4	1	4	5

其中，AA是矩阵A中所有非零元素，长度为a，即非零元素个数；

JA是矩阵A中每行第一个非零元素在AA中的位置，最后一个元素是非零元素数加1，长度为n+1, n是矩阵A的行数；

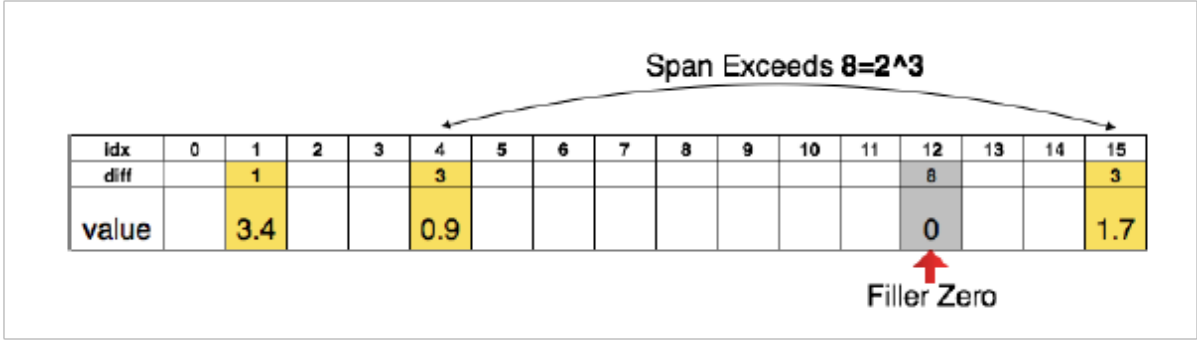
IC是AA中每个元素对应的列号，长度为a。

所以将一个稀疏矩阵转为CSR表示，需要的空间为2*a+n+1个，同理CSC也是类似。

可以看出，为了达到压缩原始模型的目的，不仅需要在保持模型精度的同时，prune掉尽可能多的权值，也需要减少存储元素位置index所带来的额外存储开销，故论文中采用了存储index difference而非绝对index来进一步压缩模型，如下图所示：

关闭

Deep Compression阅读理解及C
flysnow_hxb: @无眠栀你修改的
caffe源码后, 效果怎样? 能否共
享一下源码呢?



其中，第一个非零元素的存储的是他的绝对位置，后面的元素依次存储的是与前一个非零元素的索引差值。在论文中，采用固定bit来存储这一差值，以图中表述为例，如果采用3bit，则最大能表述的差值为8，当一个非零元素距其前一个非零元素位置超过8，则将该元素值置零。（这一点其实也很好理解，如果两个非零元素位置差很多，也即中间有很多零元素，那么将这一元素置零，对最终的结果影响也不会很大）做完权值修剪这一步后，AlexNet和VGG-16模型分别压缩了9倍和13倍，表明模型中存在着较大的冗余。

2.Weight Shared & Quantization

为了进一步压缩网络，考虑让若干个权值共享同一个权值，这一需要存储的数据量也大大减！采用kmeans算法来将权值进行聚类，在每一个类中，所有的权值共享该类的聚类质心，因此结果就是一个码书和索引表。

1.对权值聚类

论文中采用kmeans聚类算法，通过优化所有类内元素到聚类中心的差距（within-cluster sum 来确定最终的聚类结果：

$$\arg \min_C \sum_{i=1}^k \sum_{w \in c_i} |w - c_i|^2$$

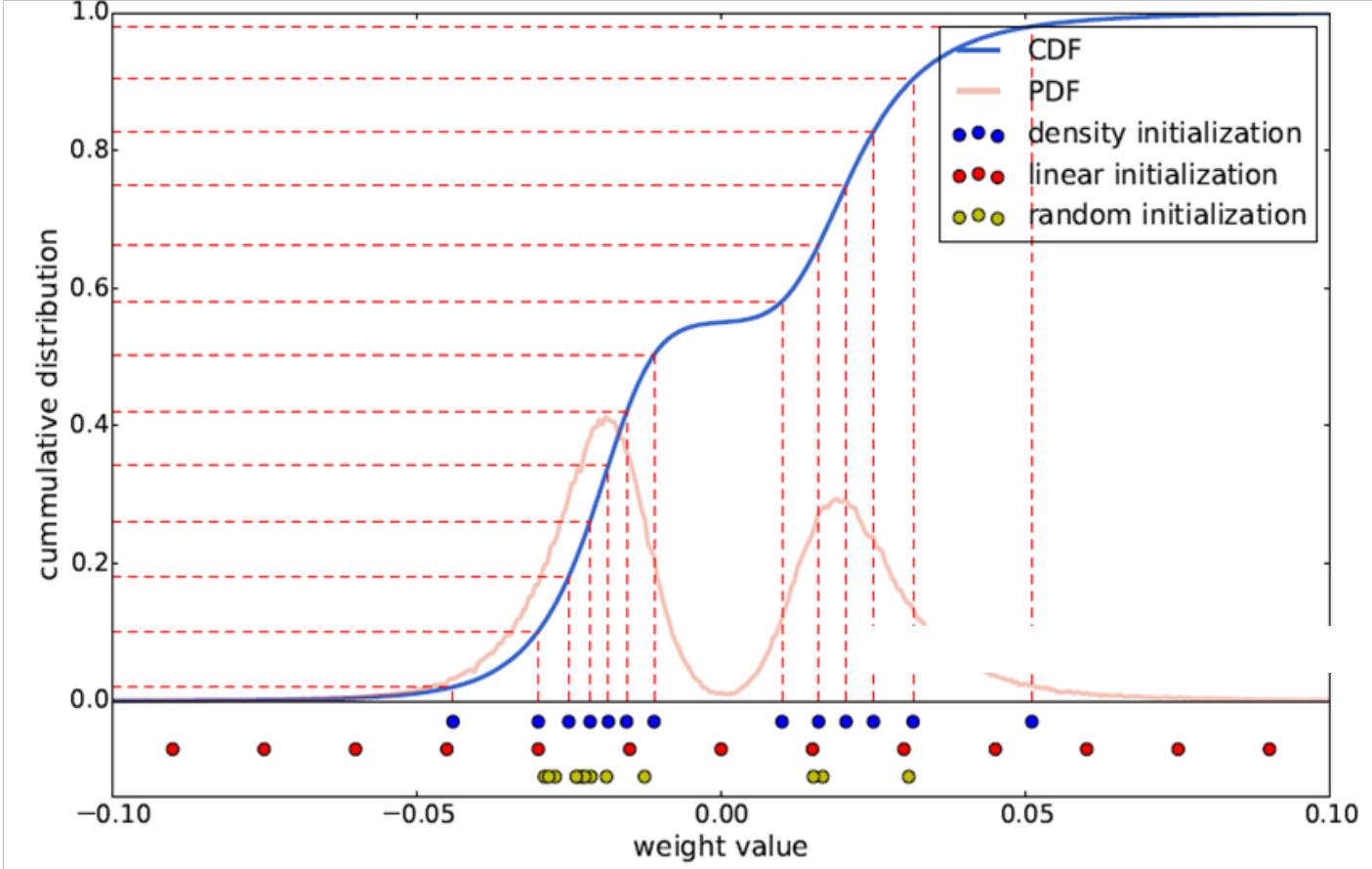
式中，W={w1,w2,...wn}是n个原始权值，C={c1,c2,...ck}是k个聚类。
需要注意的是聚类是在网络训练完毕后做的，因此聚类结果能够最大程度地接近原始网络权值分布。

2. 聚类中心初始化

常用的初始化方式包括3种：

- a) 随机初始化。即从原始数据种随机产生k个观察值作为聚类中心。
- b) 密度分布初始化。现将累计概率密度CDF的y值分布线性划分，然后根据每个划分点的y值找到与CDF曲线的交点，再找到该交点对应的x轴坐标，将其作为初始聚类中心。
- c) 线性初始化。将原始数据的最小值到最大值之间的线性划分作为初始聚类中心。

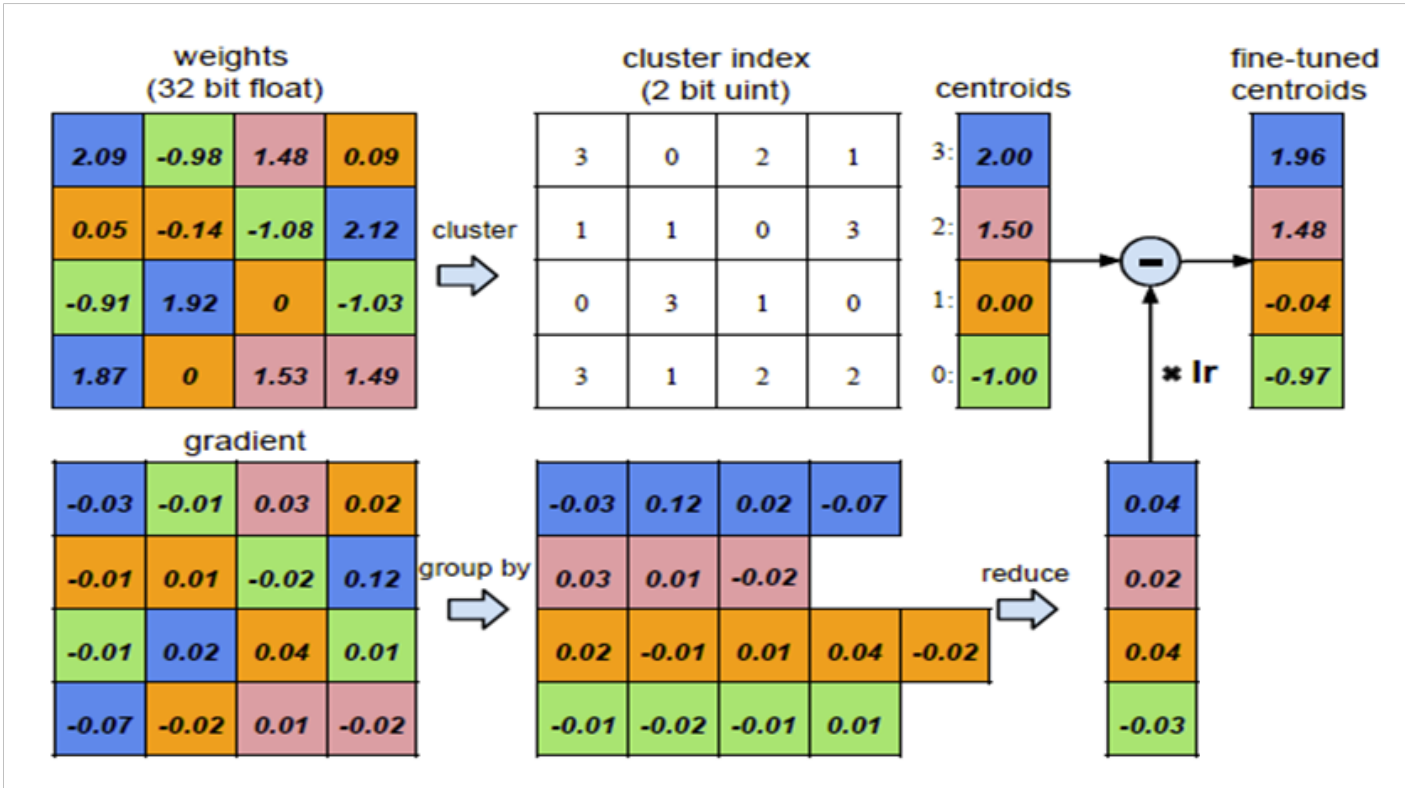
三种初始化方式的示意图如下所示：



由于大权值比小权值更重要（参加HanSong15年论文），而线性初始化方式则能更好地保留大权值中心，因此文中采用这一方式，后面的实验结果也验证了这个结论。

3. 前向反馈和后项传播

前向时需要将每个权值用其对应的聚类中心代替，后向计算每个类内的权值梯度，然后将其梯度和反传，用来更新聚类中心，如图：



共享权值后，就可以用一个码书和对应的index来表征。假设原始权值用32bit浮点型表示，共有n个权值，量化后需要存储n个8bit索引和256个聚类中心值，则可以计算出压缩率compression ratio:

$$r = 32 \times n / (8 \times n + 256 \times 32) \approx 4$$

可以看出，如果采用8bit编码，则至少能达到4倍压缩率。

3.Huffman Coding

Huffman 编码是最后一步，主要用于解决编码长短不一带来的冗余问题。因为在论文中，作者针对卷积层统一采用8bit编码，而全连接层采用5bit，所以采用这种熵编码能够更好地使编码bit均衡，减少冗余。

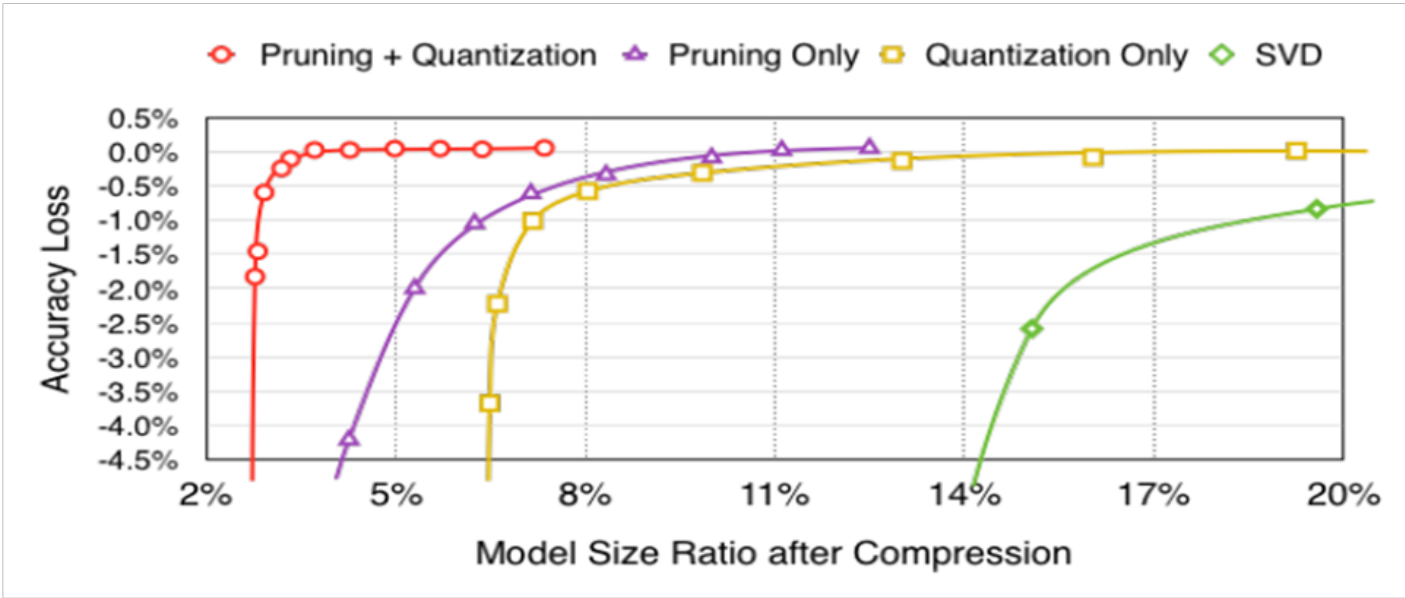
4.Evaluation

实验结果就是能在保持精度不变（甚至提高）的前提下，将模型压缩到前所未有的小。直接上图有用数据说话。

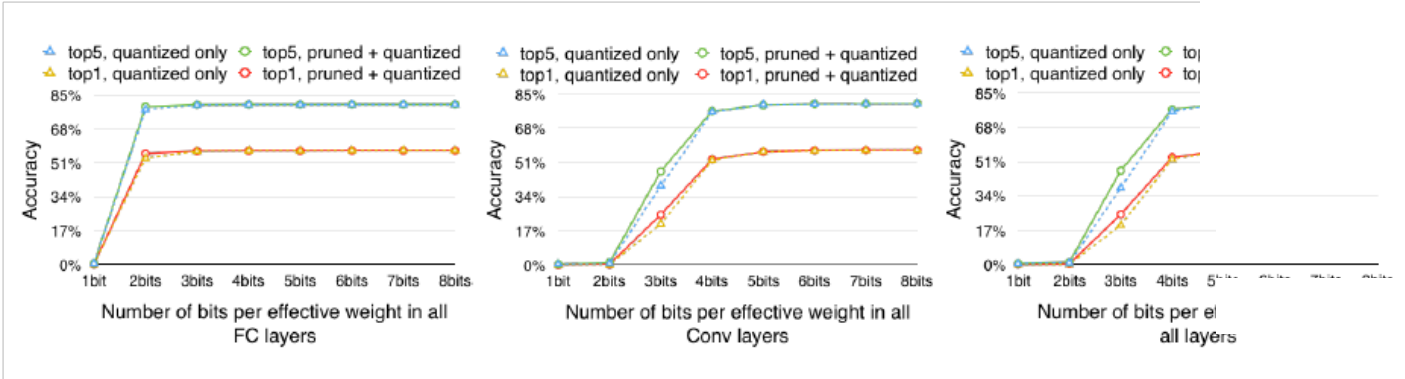
Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	27 KB	40×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	44 KB	39×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	6.9 MB	35×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	11.3 MB	49×

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
Baseline Caffemodel (BVLC)	42.78%	19.73%	240MB	1×
Fastfood-32-AD (Yang et al., 2014)	41.93%	-	131MB	2×
Fastfood-16-AD (Yang et al., 2014)	42.90%	-	64MB	3.7×
Collins & Kohli (Collins & Kohli, 2014)	44.40%	-	61MB	4×
SVD (Denton et al., 2014)	44.02%	20.56%	47.6MB	5×
Pruning (Han et al., 2015)	42.77%	19.67%	27MB	9×
Pruning+Quantization	42.78%	19.70%	6.9MB	35×
Pruning+Quantization+Huffman	42.78%	19.70%	6.9MB	35×

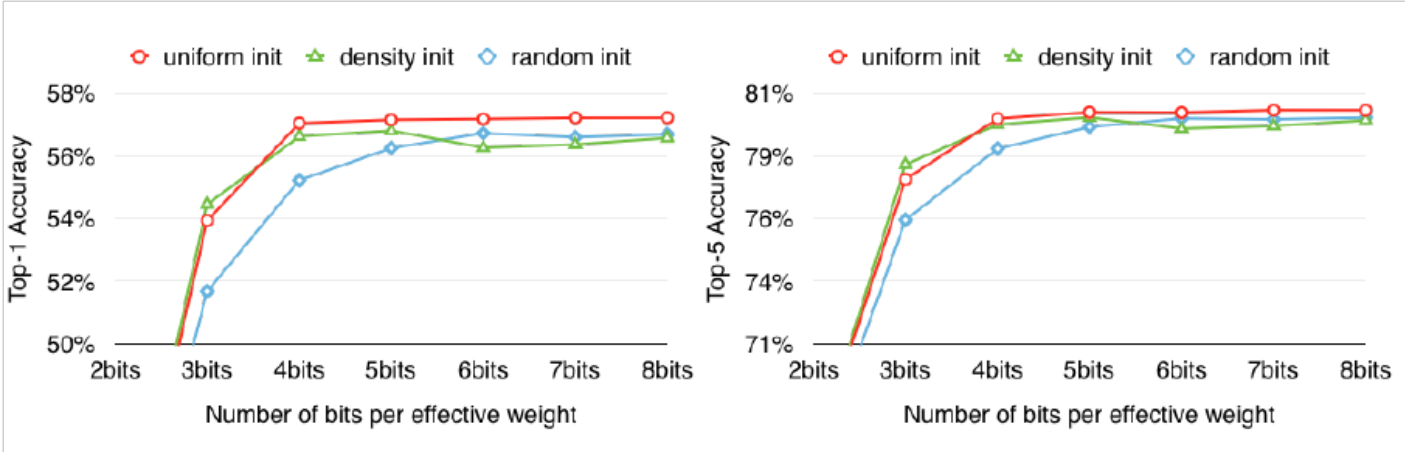
5.Discussion



不同模型压缩比和精度的对比，验证了pruning和quantization一块做效果最好。



不同压缩bit对精度的影响，同时表明conv层比fc层更敏感，因此需要更多的bit表示。



不同初始化方式对精度的影响，线性初始化效果最好。

#CONV bits / #FC bits	Top-1 Error	Top-5 Error	Top-1 Error Increase	Top-5 Error Increase
32bits / 32bits	42.78%	19.73%	-	-
8 bits / 5 bits	42.78%	19.70%	0.00%	-0.03%
8 bits / 4 bits	42.79%	19.73%	0.01%	0.00%
4 bits / 2 bits	44.77%	22.33%	1.99%	2.60%

卷积层采用8bit，全连接层采用5bit效果最好。

What

此部分讲一讲修改caffe源码的过程。其实只要读懂了文章原理，修改起来很容易。

对pruning过程来说，可以定义一个mask来“屏蔽”修剪掉的权值，对于quantization过程来说，需定义一个indice来存储索引号，以及一个centroid结构来存放聚类中心。

在include/caffe/layer.hpp中为Layer类添加以下成员变量：

```
vector<int> masks_;
vector<int> indices_;
vector<Dtype> centroids_;
```

以及成员函数：

```
virtual void computeBlobMask(float ratio) {}
/**
```

由于只对卷积层和全连接层做压缩，因此，只需修改这两个层的对应函数即可。

在include/caffe/layers/base_conv_layer.hpp添加成员函数

```
virtual void computeBlobMask(float ratio) {}
```

这两处定义的函数都是基类的虚函数，不需要具体实现。

在include/caffe/layers/conv_layer.hpp中添加成员函数声明：

```
virtual void computeBlobMask(float ratio);
};
```

类似的，在include/caffe/layers/inner_product_layer.hpp也添加该函数声明。

在src/caffe/layers/conv_layer.cpp 添加该函数的声明，用于初始化mask和对权值进行聚类。

```
9 template <typename Dtype>
10 void ConvolutionLayer<Dtype>::ComputeBlobMask(float ratio)
11 {
12     // LOG(INFO) << "conv_blob_mask" << endl;
13     int count = this->blobs_[0]->count();
14     this->masks_.resize(count);
15     //this->dmasks_ = new Dtype[count] ;
16     //this->indices_.resize(count);
17     this->centroids_.resize(CONV_QUNUM);
18     //calculate min max value of weight
19     const Dtype* weight = this->blobs_[0]->cpu_data();
20     Dtype min_weight = weight[0], max_weight = weight[0];
21     vector<Dtype> sort_weight(count);
22     for (int i = 0; i < count; ++i)
23     {
24         //this->masks_[i] = 1; //initialize
25         sort_weight[i] = fabs(weight[i]);
26     }
27     sort(sort_weight.begin(), sort_weight.end());
28     max_weight = sort_weight[count - 1];
29     //cout << sort_weight[0] << " " << sort_weight[count - 1] << endl;
30     int index = int(count*ratio); //int(count*(1- max_weight)) ;
31     Dtype thr ;
32     Dtype* muweight = this->blobs_[0]->mutable_cpu_data();
33     float rat = 0;
34     if(index > 0){
35         //thr = ratio;
36         thr= sort_weight[index-1];
37         LOG(INFO) << "CONV THR: " <<thr << " " <<ratio <<endl;
38         for (int i = 0; i < count; ++i)
39         {
40             this->masks_[i] = ((weight[i] >= thr || weight[i] < -thr) ? 1 : 0);
41             //this->masks_[i] = (weight[i] > thr ? 1 : 0); //((weight[i] == 0 ? 0 : 1) / (weight[i] > thr ? 1 : 0));
42             muweight[i] *= this->masks_[i];
43             //this->dmasks_[i] = this->masks_[i] ;
44             rat += (1-this->masks_[i]) ;
45         }
46     }
47     else{
48         for (int i = 0; i < count; ++i)
49         {
50             this->masks_[i] = (weight[i]==0 ? 0 : 1); //keep unchanged
51             rat += (1-this->masks_[i]) ;
52         }
53     }
54     LOG(INFO) << "sparsity: " << rat/count << endl;
55     min_weight = sort_weight[index];
56     int ncentroid = CONV_QUNUM;
57     kmeans_cluster(this->indices_, this->centroids_, muweight, count, this->masks_, /* max_weight, min_weight,*/ ncentroid, 1000);
58 }
59
```

同时，修改前向和后向函数。

在前向函数中，需要将权值用其聚类中心表示，红框部分为添加部分：

```
82 template <typename Dtype>
83 void ConvolutionLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
84 const vector<Blob<Dtype>*>& top) {
85     Dtype* muweight = this->blobs_[0]->mutable_cpu_data();
86     int count = this->blobs_[0]->count();
87     for (int i = 0; i < count; ++i)
88     {
89         if (this->masks_[i])
90             muweight[i] = this->centroids_[this->indices_[i]];
91     }
92     const Dtype* weight = this->blobs_[0]->cpu_data();
93     for (int i = 0; i < bottom.size(); ++i) {
94         const Dtype* bottom_data = bottom[i]->cpu_data();
95         Dtype* top_data = top[i]->mutable_cpu_data();
96         for (int n = 0; n < this->num_; ++n) {
97             this->forward_cpu_gemm(bottom_data + n * this->bottom_dim_, weight,
98 top_data + n * this->top_dim_);
99             if (this->bias_term_) {
100                 const Dtype* bias = this->blobs_[1]->cpu_data();
101                 this->forward_cpu_bias(top_data + n * this->top_dim_, bias);
102             }
103         }
104     }
105 }
106
```

在后向函数中，需要添加两部分，一是对mask为0，即屏蔽掉的权值不再进行更新，即将其weight_diff设为0，另一个则是统计每一类内的梯度差值均值，并将其反传回去，红框内为添加部分。

```
107 template <typename Dtype>
108 void ConvolutionLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
109 const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
110     const Dtype* weight = this->blobs_[0]->cpu_data();
111     const Dtype* weight_diff = this->blobs_[0]->mutable_cpu_diff();
112     int count = this->blobs_[0]->count();
113     for (int i = 0; i < top.size(); ++i) {
114         const Dtype* top_diff = top[i]->cpu_diff();
115         const Dtype* bottom_data = bottom[i]->cpu_data();
116         Dtype* bottom_diff = bottom[i]->mutable_cpu_diff();
117         // Bias gradient, if necessary.
118         if (this->bias_term_ && this->param_propagate_down_[1]) {
119             Dtype* bias_diff = this->blobs_[1]->mutable_cpu_diff();
120             for (int n = 0; n < this->num_; ++n) {
121                 this->backward_cpu_bias(bias_diff, top_diff + n * this->top_dim_);
122             }
123         }
124         if (this->param_propagate_down_[0] || propagate_down[i]) {
125             for (int n = 0; n < this->num_; ++n) {
126                 // gradient w.r.t. weight. Note that we will accumulate diffs.
127                 if (this->param_propagate_down_[0]) {
128                     this->weight_cpu_gemm(bottom_data + n * this->bottom_dim_,
129 top_diff + n * this->top_dim_, weight_diff);
130                 }
131                 Dtype* weight_diff = this->blobs_[0]->mutable_cpu_diff();
132                 for (int j = 0; j < count; ++j)
133                     weight_diff[j] *= this->masks_[j];
134                 vector<Dtype> tmpDiff(CONV_QUNUM);
135                 vector<int> freq(CONV_QUNUM);
136                 for (int j = 0; j < count; ++j)
137                 {
138                     if (this->masks_[j])
139                     {
140                         tmpDiff[this->indices_[j]] += weight_diff[j];
141                         freq[this->indices_[j]]++;
142                     }
143                 }
144                 for (int j = 0; j < count; ++j)
145                 {
146                     if (this->masks_[j])
147                         weight_diff[j] = tmpDiff[this->indices_[j]] / freq[this->indices_[j]];
148                 }
149                 // gradient w.r.t. bottom data, if necessary.
150                 if (propagate_down[i]) {
151                     this->backward_cpu_gemm(top_diff + n * this->top_dim_, weight,
152 bottom_diff + n * this->bottom_dim_);
153                 }
154             }
155         }
156     }
157 }
158
```

kmeans的实现如下，当然也可以用Opencv自带的，速度会更快些。

1 template<typename Dtype>

```
2 void kmeans_cluster(vector<int> &cLabel, vector<Dtype> &cCentro, Dtype *cWeights, int nWeights, vect
3 {
4     //find min max
5     Dtype maxWeight=numeric_limits<Dtype>::min(), minWeight=numeric_limits<Dtype>::max();
6     for(int k = 0; k < nWeights; ++k)
7     {
8         if(mask[k])
9         {
10             if(cWeights[k] > maxWeight)
11                 maxWeight = cWeights[k];
12             if(cWeights[k] < minWeight)
13                 minWeight = cWeights[k];
14         }
15     }
16     // generate initial centroids linearly
17     for (int k = 0; k < nCluster; k++)
18         cCentro[k] = minWeight + (maxWeight - minWeight)*k / (nCluster - 1);
19
20     //initialize all label to -1
21     for (int k = 0; k < nWeights; ++k)
22         cLabel[k] = -1;
23
24     const Dtype float_max = numeric_limits<Dtype>::max();
25     // initialize
26     Dtype *cDistance = new Dtype[nWeights];
27     int *cClusterSize = new int[nCluster];
28
29     Dtype *pCentroPos = new Dtype[nCluster];
30     int *pClusterSize = new int[nCluster];
31     memset(pClusterSize, 0, sizeof(int)*nCluster);
32     memset(pCentroPos, 0, sizeof(Dtype)*nCluster);
33     Dtype *ptrC = new Dtype[nCluster];
34     int *ptrS = new int[nCluster];
35
36     int iter = 0;
37     //Dtype tk1 = 0.f, tk2 = 0.f, tk3 = 0.f;
38     double mCurDistance = 0.0;
39     double mPreDistance = numeric_limits<double>::max();
40
41     // clustering
42     while (iter < max_iter)
43     {
44         // check convergence
45         if (fabs(mPreDistance - mCurDistance) / mPreDistance < 0.01) break;
46         mPreDistance = mCurDistance;
47         mCurDistance = 0.0;
48
49         // select nearest cluster
50
51         for (int n = 0; n < nWeights; n++)
52         {
53             if (!mask[n])
54                 continue;
55             Dtype distance;
56             Dtype mindistance = float_max;
57             int clostCluster = -1;
58             for (int k = 0; k < nCluster; k++)
59             {
60                 distance = fabs(cWeights[n] - cCentro[k]);
61                 if (distance < mindistance)
62                 {
63                     mindistance = distance;
64                     clostCluster = k;
65                 }
66             }
67             cDistance[n] = mindistance;
68             cLabel[n] = clostCluster;
69         }
70
71
72         // calc new distance/inertia
```

关闭


```
73
74     for (int n = 0; n < nWeights; n++)
75     {
76         if (mask[n])
77             mCurDistance = mCurDistance + cDistance[n];
78     }
79
80
81     // generate new centroids
82     // accumulation(private)
83
84     for (int k = 0; k < nCluster; k++)
85     {
86         ptrC[k] = 0.f;
87         ptrS[k] = 0;
88     }
89
90     for (int n = 0; n < nWeights; n++)
91     {
92         if (mask[n])
93         {
94             ptrC[cLabel[n]] += cWeights[n];
95             ptrS[cLabel[n]] += 1;
96         }
97     }
98
99     for (int k = 0; k < nCluster; k++)
100    {
101        pCentroPos[ k] = ptrC[k];
102        pClusterSize[k] = ptrS[k];
103    }
104
105    //reduction(global)
106    for (int k = 0; k < nCluster; k++)
107    {
108
109        cCentro[k] = pCentroPos[k];
110        cClusterSize[k] = pClusterSize[k];
111
112        cCentro[k] /= cClusterSize[k];
113    }
114
115    iter++;
116    // cout << "Iteration: " << iter << " Distance: " << mCurDistance << endl;
117    }
118    //gather centroids
119    // #pragma omp parallel for
120    // for(int n=0; n<nNode; n++)
121    //     cNodes[n] = cCentro[cLabel[n]];
122
123    delete[] cDistance;
124    delete[] cClusterSize;
125    delete[] pClusterSize;
126    delete[] pCentroPos;
127    delete[] ptrC;
128    delete[] ptrS;
129 }
```

全连接层的修改和卷积层的一致不再赘述。同样的，可以把对应的代码修改了，方便gpu训练。

最后，在src/caffe/net.cpp的CopyTrainedLayersFrom(const NetParameter& param)函数中调用我们定义的函数，即在读入已经训练好的模型权值时，对每一层做需要的权值mask初始化和权值聚类。

```
795     layers_[target_layer_id]->ComputeBlobMask(ratio);
796 }
```

至此代码修改完毕，编译运行即可。

Reference

[1] SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size

- [2] Deep Compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding
- [3] Learning both Weights and Connections for Efficient Neural Networks
- [4] Efficient Inference Engine on Compressed Deep Neural Network

总结：

最后再提一句，几乎所有的模型压缩文章都是从Alexnet和VGG下手，一是因为他们都采用了多层较大的全连接层，而全连接层的权值甚至占到了总参数的90%以上，所以即便只对全连接层进行“开刀”，压缩效果也是显著的。另一方面，这些论文提出的结果在现在看来并不是state of art的，存在可提升的空间，而且在NIN的文章中表明，全连接层容易引起过拟合，去掉全连接层反而有助于精度提升，所以这么看来压缩模型其实是个不吃力又讨好的活，获得的好处显然是双倍的。但运用到特定的网络中，还需要不断地试验，因地制宜，寻找适合该网络的压缩方式。

顶

7

踩

0

上一篇

caffe binaryproto 与 npy相互转换

下一篇

Tensorflow 离线安装跳坑总结

相关文章推荐

- Deep Compression阅读理解及Caffe源码修改

• 【直播】70天软考冲刺计划--任铄

• caffe 反卷积 相关源码

• 【直播】打通Linux脉络 进程、线程、调度--宋宝华

• TensorFlow和Caffe、MXNet、Keras等深度学习...

• 【直播】机器学习之凸优化--马博士

• caffe2源码

• 【套餐】MATLAB基础+MATLAB数据分析与统计-...
- Deep Compression, Song Han, Caffe 实现

• 【课程】3小时掌握Docker最佳实战--徐西宁

• caffe官方源码

• 【课程】深度学习基础与TensorFlow实践--AI100

• TensorFlow和Caffe、MXNet、Keras等其他深度...

• caffe源码（2015年8月15日）

• 贾扬清-DIY Deep Learning for Vision- a Hands-O...

• 深度学习工具 caffe 源码



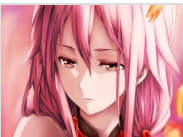
牡丹花种苗



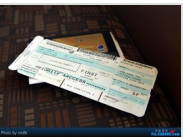
特价国际机票



国际机票特价



学习动漫的学



到美国机票价



美国博士条件



小米大全报价

查看评论

20楼 [xiyan1111](#) 2017-08-13 19:20发表



楼主，求该模型的tensorflow的代码，或者您知道哪里能找到吗

Re: [无眠栀](#) 2017-08-14 17:46发表



回复xiyan1111：抱歉，暂时没看到

19楼 [taoliuliutao](#) 2017-08-11 10:37发表



我也持6楼的观点，Quantization code中，每次前传都将聚类中心赋值给weights，但训练中未更新聚类中心，这个有问题吧？我没看到更新聚类中心啊

Re: [无眠栀](#) 2017-08-14 17:46发表



回复taoliuliutao：这个比较有争议，论文没有讲清楚，事实上在当时做的时候是考虑过每次更新聚类中心的，但是实验发现聚类基本不影响性能，如果每轮更新聚类中心，有可能导致训练不容易收敛，这个你可以尝试一下

关闭

18楼 [flysnow_hxb](#) 2017-07-11 20:32发表



@无眠栀
你修改的caffe源码后，效果怎样？能否共享一下源码呢？

Re: [无眠栀](#) 2017-08-14 17:46发表



回复feiwudexue123：请参考文章中的github源码

17楼 [_顺其自然](#) 2017-05-18 22:37发表



请问，我在make all 的时候遇到下面的错误,有人知道是什么原因，该怎么解决吗？谢谢
src/caffe/net.cpp: 在成员函数'void caffe::Net<Dtype>::CopyTrainedLayersFrom(const caffe::NetParameter&)'中:
src/caffe/net.cpp:742:49: 错误： 'ratio'在此作用域中尚未声明
make: *** [.build_release/src/caffe/net.o] 错误 1
make: *** 正在等待未完成任务....

16楼 [心悦君兮君不知_](#) 2017-04-27 21:16发表



CONV_CUNUM怎么定义啊

15楼 [qq_15612283](#) 2017-04-20 15:54发表



博主有试过int8 deeplearning 在caffe上面的实现吗

14楼 [_顺其自然](#) 2017-03-28 20:19发表



博主，您好，非常感谢您的博客，请问一下kmeans的实现代码是要放在哪个文件里或者是添加在哪个代码里？还是博主给出该代码仅是为了读者明白其原理，不需要再在caffe源码里进行修改了，谢谢

13楼 [qq676506196](#) 2017-03-27 15:58发表



博主实现后，能达到论文的效果吗？

12楼 [Rill](#) 2017-03-08 09:26发表



求助一下，修改conv_layer.cpp/conv_layer.cu后，使用CPU/GPU进行训练时（先用CPU模式，后来尝试用GPU模式，都有错误），一旦调用forward_cpu/forward_gpu就会报错。尝试了各种方法后，我把muweight[i] = this->centroids_[this->indices_[i]]注释掉就能正常运行，backward_cpu/backward_gpu()也有类似情况，请问你有遇到过这种情况么？

Re: [心悦君兮君不知_](#) 2017-06-07 09:37发表



回复rill_zhen：请问层主修改成功了木有

Re: [Rill](#) 2017-06-09 11:58发表



回复u014538796：后来根据paper自己实现了。

Re: [MiracleN](#) 2017-06-10 17:03发表



回复rill_zhen：你好，可以分享下你的代码吗

Re: [Rill](#) 2017-03-08 09:30发表



回复rill_zhen：尝试过下面评论中的方法，但没有效果。

11楼 [cpchung](#) 2017-03-02 22:21发表



哪位善长仁翁愿意分享一下这个修改好的代码在github呢？

我一直很好奇,这个代码或者算法是不是有些很神秘的地方,或者什么潜规则,If

Re: [cpchung](#) 2017-03-03 16:02发表



回复u013478129：哪位善长仁翁愿意分享一下这个修改好的代码在github呢？

我一直很好奇,这个代码或者算法是不是有些很神秘的地方,或者什么潜规则,原作song han也不愿意分享这个代码？

37303 9 065 at qq.com

10楼 [_顺其自然](#) 2017-02-21 22:00发表



博主，你好，请问一下Deep Compression的具体运行步骤是什么，ReadMe里面的内容没有看懂

关闭

9楼 [kdplus](#) 2017-02-07 17:33发表



点赞！博主你好，我想问一下实际上这么做之后加速比能达到多少？有做过比较么QVQ谢谢！

8楼 [无眠栀](#) 2017-02-06 12:07发表



回复K_Five：这两个过程是分开的，我在实际训练时，也是先做一遍剪枝，再在finetune的基础上再做聚类。代码写到一块并没有影响，实际中可以用宏来开关某一个功能

Re: [Bxiaoboai](#) 2017-02-09 10:32发表



回复may0324：我的意思是，上面的code似乎没有更新聚类中心吧，每次前传都赋值的是原始的聚类中心。另外，我的理解聚类不需要每次迭代都做，读入初始值后做一次，迭代时调整就可以了。

7楼 [K_Five](#) 2017-02-06 11:33发表



楼主有没有看过韩松的2015年关于如何剪枝的方法？他的阈值是不是如你所写的那样？

Re: [无眠栀](#) 2017-02-06 12:08发表



回复K_Five：有看过，但是具体算法没有给出，只提到了迭代更新，阈值这块我目前还是通过迭

6楼 [Bxiaoboai](#) 2017-01-26 14:29发表



Quantization code中，每次前传都将聚类中心赋值给weights，但训练中未更新聚类中心，这个有问题吧？应该是聚类后直接赋值给weights，后续训练就用diff均值更新weights就可以了。

Re: [无眠栀](#) 2017-02-06 12:05发表



回复Bxiaoboai：反向传播更新的就是聚类中心的值，因为每个权值都用其聚类中心表示，计算的梯度就是每个聚类中心内的梯度均值，最后也是用的这个diff来更新的聚类中心

5楼 [qiyouhuang](#) 2016-12-25 13:22发表



CONV_CUNUM没有定义

Re: [心悦君兮君不知_](#) 2017-04-27 21:18发表



回复qiyouhuang：你好请问你解决了这个问题了吗？怎么解决的

Re: [无眠栀](#) 2017-02-06 11:59发表



回复qiyouhuang：这个论文中有提，卷积层是256个，全连接层是32个

4楼 [tim110629](#) 2016-12-21 17:03发表



Hi 博主，
最近我也在学习深度学习网络压缩，可否可以分享一下你修改的代码？

3楼 [fangfangcome](#) 2016-12-21 10:01发表



weigiths剪枝那边，要绝对值比较

Re: [无眠栀](#) 2017-02-06 12:09发表



回复fangfangcome：是用的绝对值

Re: [Rill](#) 2017-03-16 11:42发表



回复may0324：你的绝对值，只是拿来排序的，不是用来比较的。你是直接按ratio剪掉的。而且，mask在后面train过程中没有迭代更新（即使后面train完成后，绝对值很小的weights也不会被剪掉）。

Re: [Rill](#) 2017-03-16 11:55发表



回复rill_zhen：不过还是很感谢，学到很多。

2楼 [aloha89](#) 2016-12-07 22:32发表



求助一下，修改conv_layer.cu后，使用GPU进行训练时，一旦调用backward_gpu就会报错。尝试了各种方法后，我把weight_diff[j] *= this->mask_[j]注释掉就能正常运行，请问你有遇到过这种情况么？

我尝试把weight_diff[j]打印出来，发现仅仅是打印weight_diff[j]也会报错，但weight_diff作为指针本身可以打印。

Re: [无眠栀](#) 2016-12-12 15:39发表

回复aloha89：在.cu里如果直接访问gpu数据会报错，就是如果你是定义了

关闭



float* weight = blobs_[0]->mutable_gpu_data(), 如果访问weight[j]是会运行错误。解决办法我目前只能通过访问cpu数据mutable_cpu_data(), 虽然可能会导致增加了cpu与gpu同步数据的时间, 但保证没有问题。

Re: [aloha89](#) 2016-12-20 18:40发表



回复may0324: 的确是这样的, 我还找到一个方法, math_function里面提供了很多操作gpu数据的函数, 通过这些函数也可以对gpu里的数据进行操作。例如使用mask将gpu_diff中需要prune的neuron改为0, 可以使用: caffe_gpu_mul(this->blobs_[0]->count(), this->blobs_[0]->gpu_diff(),this->masks_[0]->gpu_data(),this->blobs_[0]->mutable_gpu_diff())

非常感谢你的文章, 提供了很多线索和切入点!

1楼 [孤独的骆驼](#) 2016-11-29 10:47发表



你好, 最近在看这篇论文 感觉你写的很有用。我想问下CopyTrainedLayersFrom () 中修改的代码中ratio 是什么, 在哪定义? 或者你可不可以给你改好代码的链接? 我有一些实现细节看的不太明白。

Re: [无眠梹](#) 2016-12-12 15:39发表




回复u010507602: ratio是个局部变量, 我是在CopyTrainedLayersFrom定义的, 但是其实可以写中, 它表示每层prune的比例, 由自己决定

发表评论

用户名: haijunz

评论内容:



提交

* 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved



关闭