

ZTE 深度学习模型优化加速说明文档

江湖名号: chriszhang 联系邮箱: 18804050503@163.com

1. 实验配置

硬件环境: Intel i5-4590 CPU, 12 GB RAM 台式电脑

软件环境: Windows 10, Pycharm, Visual Studio 2015

编程语言: Python, C++

注: 这里为了操作方便, 使用的均是 Windows 下的 caffe CPU 版本, 包括下面的一些时间及参数数量的数据统计, 对 GPU 版本仍具有一定的参考意义。因为不同系统, 不同平台最后实际测试效果均会有不稳定的波动, 所以下面的内容有说明方法收益也基本不涉及到收益的具体到数值。

2. 题目解析

基于 Caffe[1] 框架及课题专家给定的网络模型需要在保证特征余弦相似度的情况下尽可能的实现降低模型大小, 降低运行设备资源使用, 提高速度的效果。

首先用 Netscope 工具对网络结构进行可视化, 可视化以后的模型结构借鉴了 GoogLeNet[2] 及 ResNet[3] 的思想, 主要由以下 3 种不同的 block 结构组成, 相应的 block 及其对应的 block 名字分别:

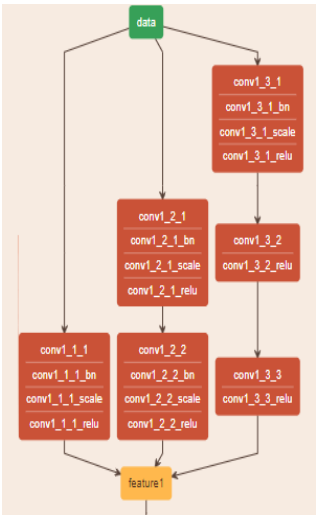
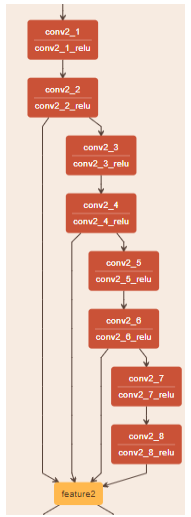
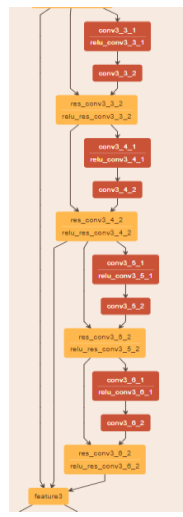
Name	conv1_*	conv2_*	conv3_*, conv4_*, conv5_*
Block type			

表 1 网络结构类型

下面根据 Caffe 框架自带的 time 测试时间函数, 生成运行日志, 根据日志文件对整个网络前向传播中的耗时及各部分耗时所占的比重进行一个统计, 统计结果如下图 1, 2 所示。

从图 2 可以看出全连接层运行时间仅占据了整个网络的 26.11%, 其余的 73.89% 运行时间均来自卷积层。

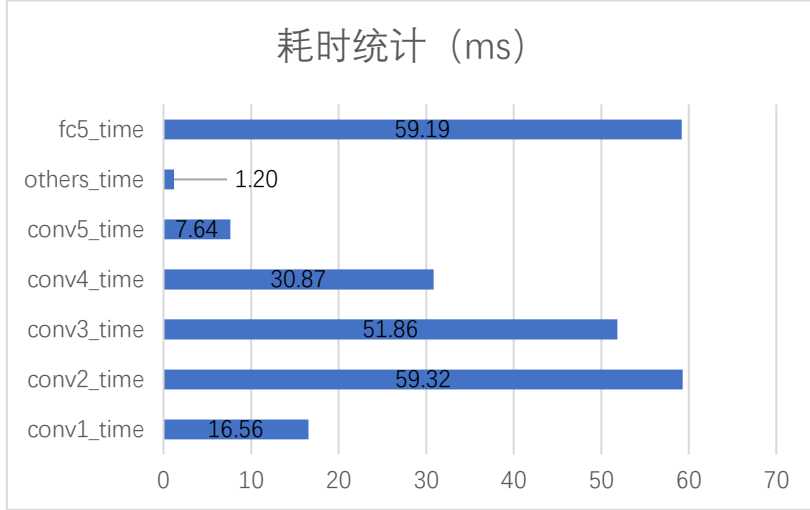


图 1 耗时统计

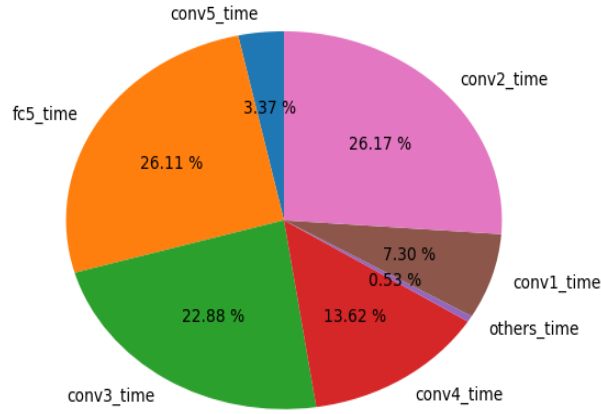


图 2 耗时占比统计

下面给出网络参数量的定义：

$$C_{in} \times C_{out} \times K_h \times K_w \quad (1)$$

(1) 式中 C_{in} 表示卷积输入通道数， C_{out} 表示卷积输出通道数， K_h, K_w 表示卷积核的长和宽。根据公式 (1) 进一步对网络的各个 block 进行一个参数量统计，具体统计结果如下图 3、图 4 所示：

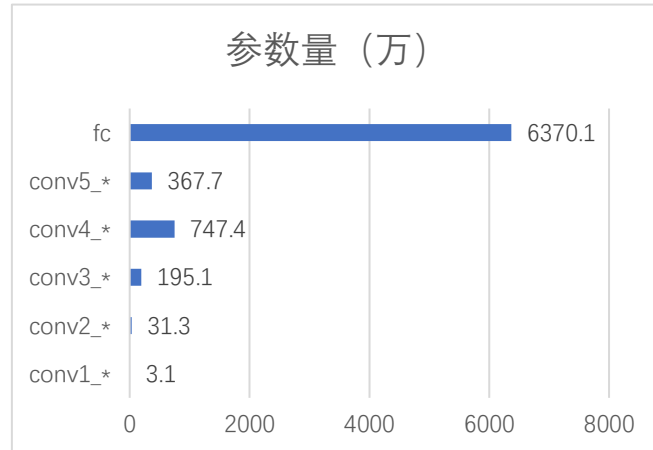


图 3 参数量统计

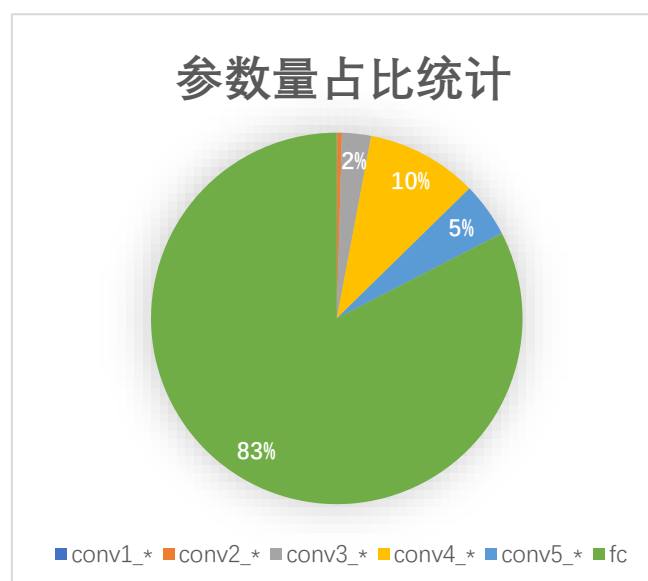


图 4 参数量占比统计

由上图可以看出，虽然运算时间有 3/4 都在卷积层，但是参数量有 83% 都在全连接层，所以根据上面分析，可以知道官方模型的运行时间主要集中在卷积层，而参数量主要集中在全连接层。

3. 解决方法

经过第 2 节的题目解析，可入手的角度就很明确了，经过网上查找相关资料以及翻阅相关论文，并结合实际题目情况，这里主要对卷积层和全连接层分别使用了不同的方法进行模型加速，其中也涉及到了改动源码的方法。

3.1 加速卷积层

3.1.1 卷积层和 bn 层融合

Bn 层全称为 batch normalization layer，批归一化层，它首先出现在 GoogLeNet v2[4] 中，神经网络的一个强假设前提是输入数据的分布与输出的数据分布是一致的，但是事实并非如此，并且随着网络深度的增加这样的分布偏差会逐渐累积，这样的问题被称为 internal covariate shift，bn 层就是通过数据输入下一层卷积层之前将数据进行归一化，这里的归一化则是将数据归成均值为 $E(x)=0$ ，方差为 $Var(x)=1$ 的分布，然后再输入卷积层进行学习。

$$x = \frac{x - E(x)}{\sqrt{Var(x)}} \quad (2)$$

为了保证 bn 操作以后网络非线性特性的获得，对变换后的 x 又进行了一个伸缩变化。

$$y = \gamma x + \beta \quad (3)$$

这里的 γ, β 均是需要网络学习的两个参数。所以对卷积过程结合公式（2）和公式（3）进行重构可得：

$$\begin{aligned}
 y_{conv} &= w \cdot x + b \\
 y_{bn} &= \gamma \cdot \left(\frac{y_{conv} - E(x)}{\sqrt{Var(x)}} \right) + \beta \\
 &= \gamma \cdot \left(\frac{wx + b - E(x)}{\sqrt{Var(x)}} \right) + \beta
 \end{aligned} \tag{4}$$

变换后的公式可以得到如下关系：

$$\begin{aligned}
 w &= \frac{\gamma}{\sqrt{Var(x)}} \cdot w \\
 \hat{b} &= \frac{\gamma}{\sqrt{Var(x)}} \cdot (b - E(x)) + \beta \\
 y_{bn} &= w \cdot x + \hat{b}
 \end{aligned} \tag{5}$$

经过上面的推导，可以看到 bn 层实际是对权重 w 和偏置 b 进行变换操作，所以在完成训练后完全可以直接将相关系数结合公式（5）乘进网络权重里，以加速卷积前向传播过程。

3.1.2 filter 层面的剪枝

关于剪枝，已经有大量的学者进行相关的研究，一般的剪枝思想都是如下图所示：

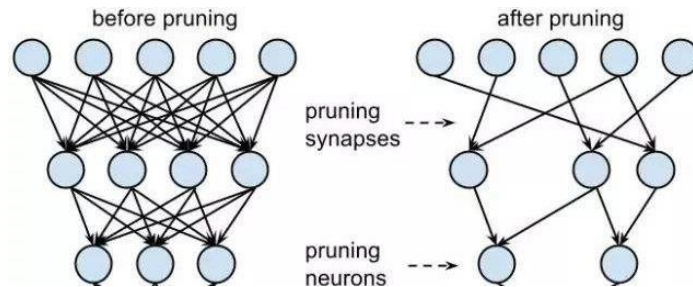


图 5 剪枝示意图

剪枝方法的核心思想基本都是删除一些我们认为卷积层中不重要的权重，主要是不同的方法对如何衡量权重的重要性进行了不同的探究，而这里我主要是参考了[5],[6] 两篇论文，**通过将权重中的所有值相加，如果这些值的总和接近 0，则认为这些权不重要的衡量方法**来进行剪枝操作。官方模型中用的部分残差结构如图 6 所示：

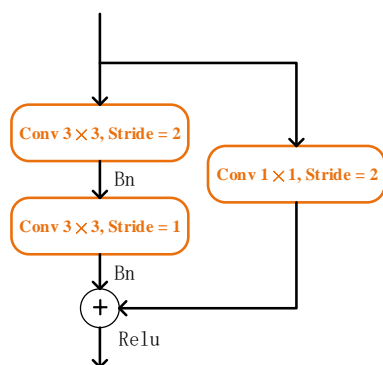


图 6 残差结构

参考文献 [6] 也有提到对于 ResNet 中的剪枝操作，一般只是动第一个 3×3 的卷积层，因为下面特征相加层的特性需要输入的通道数相同，并且第 2 节网络分析中有提到网络的结构特性是多个 ResNet block 相互之间连接，轻易对第二个 3×3 的卷积进行删除就会连带其他层一起变动，可能导致一些我们不希望看到的结果。参数统计结果如下：

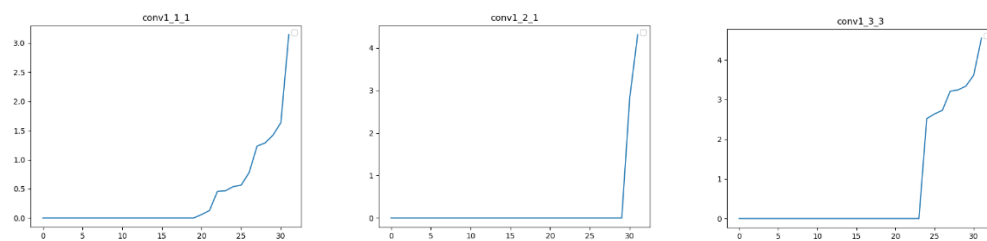


图 7 conv1_* 部分层权重分布示意图

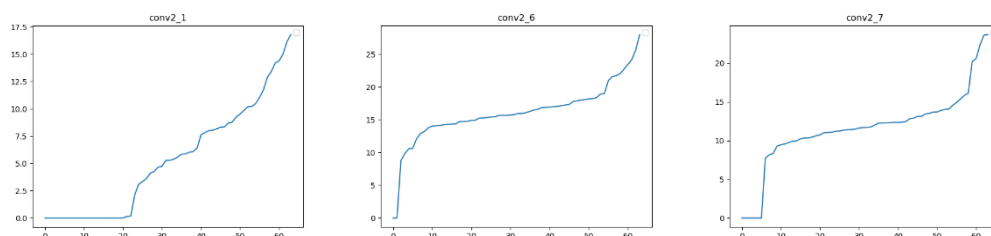


图 8 conv2_* 部分层权重分布示意图

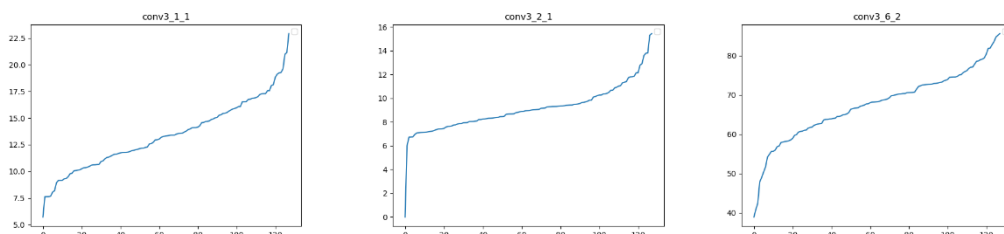


图 9 conv3_* 部分层权重分布示意图

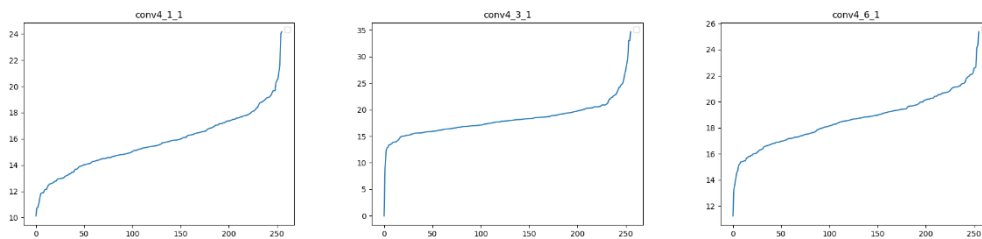


图 10 conv4_* 部分层权重分布示意图

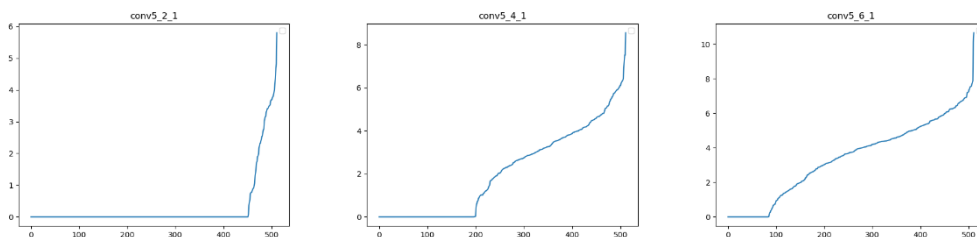


图 11 conv5_* 部分层权重分布示意图

由上图 7-图 11 可以看到，网络开始的 conv1_* 部分及网络结尾的 conv5_* 部分权重相对来说比较稀疏，conv2_* 开始一直到 conv3_* 和 conv4_* 权重和排序后的分布整体相似，但是权重为 0 的部分相对较少，经过实际剪枝测试，conv3_* 和 conv4_* 部分权重较为敏感，稍有变动对最后的特征相似度影响较大。所以剪枝主要集中在 conv1_*, conv2_* 前半部分，conv5_*。

3.1.3 block 层面的剪枝

这一部分的思想仍然是剪枝，但是与 3.1.2 剪枝不同的是，上面的剪枝都是通道层面上的，并不会对网络结构有大的影响，这一部分的剪枝是直接对层进行一些删除，甚至是整个 block 进行删除，这一部分的内容主要集中在 conv2_* 和 conv5_*，例如我在模型中直接删除了 conv2_7 和 conv2_8，以及最后的 conv5_* 删除的只剩下了一个 conv5_1_1b 层（后面多加了一个 relu 层以保持性能），其他的 conv5_* 部分都直接删除了。此部分没有很扎实的理论依据，其实完全就是针对这个特定的网络模型比较适用，不过也给日常有剪枝需求的情况打开了一个新的思路。

3.1.4 shiftCNN 中的参数量化

在查阅资料的过程中，除了剪枝，量化也是加速神经网络的一个重要手段，主要是依赖一些低精度的量化技术，例如原本模型中存储的权重及计算的特征都是 float32 位的，可以尝试使用 16bit 或者 8bit 甚至 2bit 存储的方式来使网络的计算过程对硬件更友好，能大大降低存储这些参数所需要的内存，从而达到加速网络的作用。

shiftCNN [7] 则是一种残差量化的方式，其采用了简单的移位和加法来实现乘法，从而减少计算量。具体原理可以参考原论文。这里关于量化的方法，本比赛不能进行模型的重新训练，并且 caffe 框架本身是只支持 float 或者 double 类型的数据，所以即使对模型完成了转化，但是实际前向传播过程中也还是会转回 float 类型，所以在不改动框架的基础上，量化的方法收益并不是很大。

经过合理的参数设置 shiftCNN 可以在不损失精度的情况下，大幅降低模型压缩成压缩包后的 size，可能最大的用处就是可以提高模型上传测试的速度，虽然在速度方面没有

用处，但是可以加快模型上传速度，后续模型均是通过 shiftCNN 代码处理后上传的。

3.1.5 relu 层的合并及删除

(1) relu 层合并

如果反复读取一段内存进行运算，效率上肯定不如一次读取多次运算更高效。上面提到的 bn 层合并就是本着节省访问内存的原则进行的，而这部分与 bn 层不同的点是，bn 层主要是对网络的权重直接进行变换，所以不涉及到源码的修改，而 relu 是直接对网络中的特征进行操作，若想要实现 relu 层与卷积层的合并。

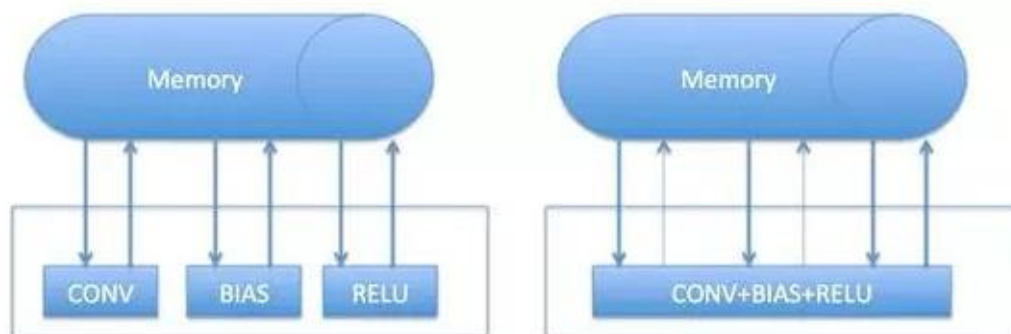


图 12 访存示意图

通过阅读 caffe 源码，熟悉整个卷积过程中参数的传递，修改了：

caffe.proto
base_conv_layer.hpp
base_conv_layer.cpp
math_function.hpp
math_function.cpp
math_function.cu

6 个文件（已开源到我的 Github 上：

https://github.com/anlongstory/caffe_combine_relu），成功实现了 CPU 版本和 GPU 版本的卷积层与 relu 层的融合，通过卷积层新添加的 bool 型参数 combine_relu（default = false），然后直接将卷积层后的 relu 层删除即可。

使用示例如下：

```
layer{
  name: "conv1"
  type: "Convolution"
  bottom: "conv0"
  top: "conv1"
  convolution_param {
    num_output: 10
    kernel_size: 3
    stride: 2
  }
}
layer{
  name: "relu_conv1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}

==

layer{
  name: "conv1"
  type: "Convolution"
  bottom: "conv0"
  top: "conv1"
  convolution_param {
    num_output: 10
    kernel_size: 3
    stride: 2
    combine_relu: true
  }
}
```

图 13 使用示例

关于合并 relu 层，主要就是减少了数据的搬运次数，但是实际进行合并以后在 GPU 的上运算提升不是很明显，应该是 GPU 本身就具有高并发性，算力也足够，但是对于一些

计算力和存储有限的平台下可能价值更大吧。

(2) relu 层的删除

这里主要是对 conv1_* 中的除了 conv1_1_1 外的其余层 relu 层直接进行删除，实验发现这样的操作对精度没有什么影响并且速度和显存也有一定的提升效果，可能是因为浅层的特征图分辨率都比较大，来回搬运会比较耗时。但是将 conv1_1_1 的一并删除，特征相似度就会从 0.907 直接下降到 0.899，所以这里选择保留。

3.1.6 concat 层的输入替换

个人感觉这个操作是这个比赛的一个隐藏 trick 了，因为正常的剪枝思路都是对删除进行直接的删减，如果参数的稀疏度不够，那个多少都会对精度产生不同程度的影响，并且最后的收益也未必很明显，这里的 concat 层的输入替换可以理解为是删层或者是减少参数并减少精度降低的中间步骤，以 conv2_* 为例。

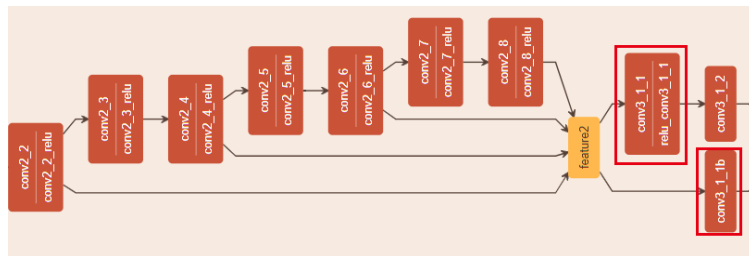


图 14 conv2_* 示例

下面是网络结构文件中关于“feature2”层的定义：

```
layer {
  name: "feature2"
  type: "Concat"
  bottom: "conv2_2"
  bottom: "conv2_4"
  bottom: "conv2_6"
  bottom: "conv2_8"
  top: "feature2"
}
```

conv2_2, conv2_4, conv2_6, conv2_8 均是输出为 64 通道特征图，经过 feature2 以后就会被拼接成 256 通道的特征输出，对应的 conv3_1_1 和 conv3_1_1b 的参数量为 $128 \times 256 \times 3 \times 3$ ，所以输入通道为 256，输出通道为 128。常规的删除 conv2_7 和 conv2_8 的操作是直接删除 conv2_7 和 conv2_8，然后将 conv3_1_1 和 conv3_1_1b 对应的输入通道的权重也删除。

我在这里将 **bottom: "conv2_8"** 变为 bottom: "conv2_6"，即：

```
layer {
  name: "feature2"
  type: "Concat"
  bottom: "conv2_2"
  bottom: "conv2_4"
  bottom: "conv2_6"
  bottom: "conv2_6"
  top: "feature2"
}
```


}

使用 conv2_6 替换原来的 conv2_8，发现精度虽然略有下降但是相对于直接删除 bottom: "conv2_8" 好了很多，而且 concat 层是线性拼接，因为最后拼接的特征图是重复的，可以将最后的两个 bottom: "conv2_6" 合并为一个，同时对 conv3_1_1 和 conv3_1_1b 对应的输入通道进行一个线性相加，这样可以达到直接删除 conv2_7 和 conv2_8 的效果以及减少 conv3_1_1 和 conv3_1_1b 输入通道的作用，但是对精度的影响大大减少。

这个 trick 不仅仅只用在想删除层的情况，也可以只单纯进行一个替换，然后合并下一层的输入层权重个数。根据不同层的实验测试结果，同样的 trick 也用在了 conv4_*（这一层合并后对特征相似度还稍好处）和 conv5_* 部分，以此来降低我们减少输入通道带来的精度影响，conv3_* 部分在当时测试的模型下比较敏感，合并以后余弦相似度会从 0.907 下降到 0.89 左右，所以选择保留原本结构。

3.2 加速全连接层

3.2.1. fc 层的 svd 操作

全连接层上面经过分析可以知道，参数的绝大部分都集中在这一层中，但是通常全连接层会存在大量的冗余连接，而且题目给出的模型最后是合并了 1536 个 9×9 的特征图，根据最后 fc_5 的 512 维输出，最后一层共有 512×124416 个参数，一种常用的方法就是对全连接层进行 svd 操作，这里主要是利用了降维的思想，即对原数据经过一定的变换以后，可以通过一小部分数据来完全替代原先的庞大数据。这里就不再赘述，通过不断的实验测试，最后修改后的模型决定采用 130 维的中间维度，然后再从 130 维输出到 512 维。

3.3 MEC 提升卷积过程（未完成）

主要是想复现《MEC: Memory-efficient Convolution for Deep Neural Network》[8] 论文中的 im2col 卷积优化算法。

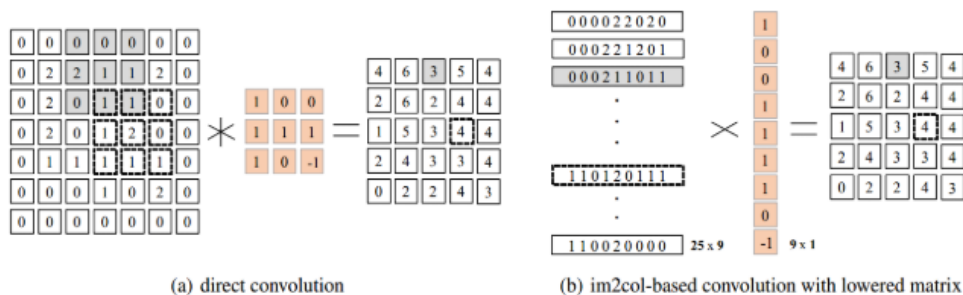


图 15 正常 caffe 卷积过程

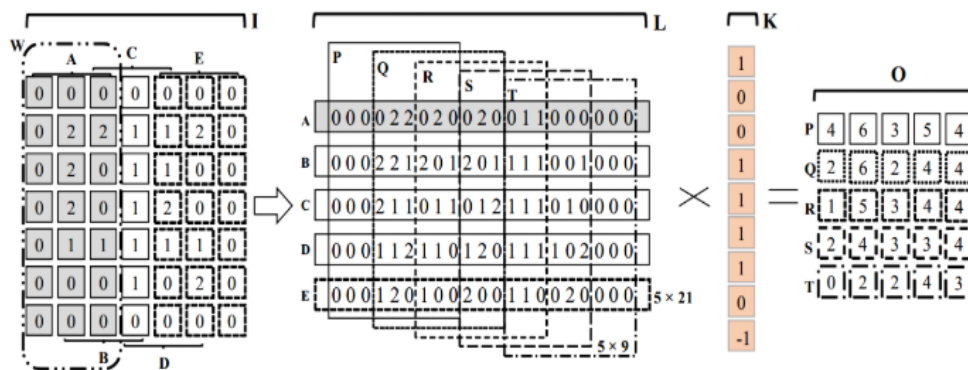


图 16 MEC 改进后的卷积过程

可以看到存储的中间矩阵从 25×9 缩减到了 5×21 ，可以大大缩减卷积过程中的存储空间，并且最后的 for 循环实现调用 blas 的矩阵相乘操作可以采用多线程的方式，本地已经实现了一维的卷积操作，测试样例是输入图片 1000×1000 ，卷积核大小为 11×11 ，卷积核数为 5，步长为 1, 不填充的情况，加入多线程（线程数为 4）以后可以看到有 4 倍左右的加速效果。

```
t702@t702:~/Downloads/MEC-master$ ./im2colOpt.sh
[im2colOpt 1000*1000]Total time cost: 174 ms
t702@t702:~/Downloads/MEC-master$ ./im2colBase.sh
[im2colBase 1000*1000]Total time cost: 702 ms
```

图 17 运行时间对比

但是合并到 caffe 框架中时，因为多维卷积的缘故，且原文描述的是 nhwc 的数据存储格式，而 caffe 是 nchw 的数据存储格式，所以多维数据的存储格式导致思路有所卡住，但是日后有需求还是很值得复现的！

4. 总结

现在对以上三节内容的要点进行一个总结归纳：

- 1、参数主要集中在全连接层，运算时间主要在卷积层
- 2、bn 层融合，filter 级别的剪枝及 block 级别的剪枝及 fc 层 svd 操作属于常规操作
- 3、主要亮点是 concat 层的替换操作以此来降低剪枝和合并通道带来的精度影响以及最后的修改源码的 relu 合并。

参考文献：

- [1] Jia, Yangqing, Shelhamer, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. 2014.
- [2] Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions[J]. 2014.
- [3] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[J]. 2015.
- [4] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift[C]// International Conference on International Conference on Machine Learning. 2015.
- [5] Li H, Kadav A, Durdanovic I, et al. Pruning Filters for Efficient ConvNets[J]. 2016.

- [6] He Y , Zhang X , Sun J . Channel Pruning for Accelerating Very Deep Neural Networks[J]. 2017.
- [7] Gudovskiy D A, Rigazio L. ShiftCNN: Generalized Low-Precision Architecture for Inference of Convolutional Neural Networks[J]. 2017.
- [8] Cho M , Brand D . MEC: Memory-efficient Convolution for Deep Neural Network[J]. 2017.