深度学习框架Caffe源码解析

原创：薛云峰 深度学习大讲堂 2016-12-12

相信社区中很多小伙伴和我一样使用了很长时间的Caffe深度学习框架，也非常希望从代码层次理解Caffe的实现从而实现新功能的定制。本文将从整体架构和底层实现的视角，对Caffe源码进行解析。

1.Caffe总体架构

Caffe框架主要有五个组件，Blob，Solver，Net，Layer，Proto，其结构图如下图1所示。Solver负责深度网络的训练，每个Solver中包含一个训练网络对象和一个测试网络对象。每个网络则由若干个Layer构成。每个Layer的输入和输出Feature map表示为Input Blob和Output Blob。Blob是Caffe实际存储数据的结构，是一个不定维的矩阵，在Caffe中一般用来表示一个拉直的四维矩阵，四个维度分别对应Batch Size（N），Feature Map的通道数（C）,Feature Map高度(H)和宽度(W)。Proto则基于Google的Protobuf开源项目，是一种类似XML的数据交换格式，用户只需要按格式定义对象的数据成员，可以在多种语言中实现对象的序列化与反序列化，在Caffe中用于网络模型的结构定义、存储和读取。

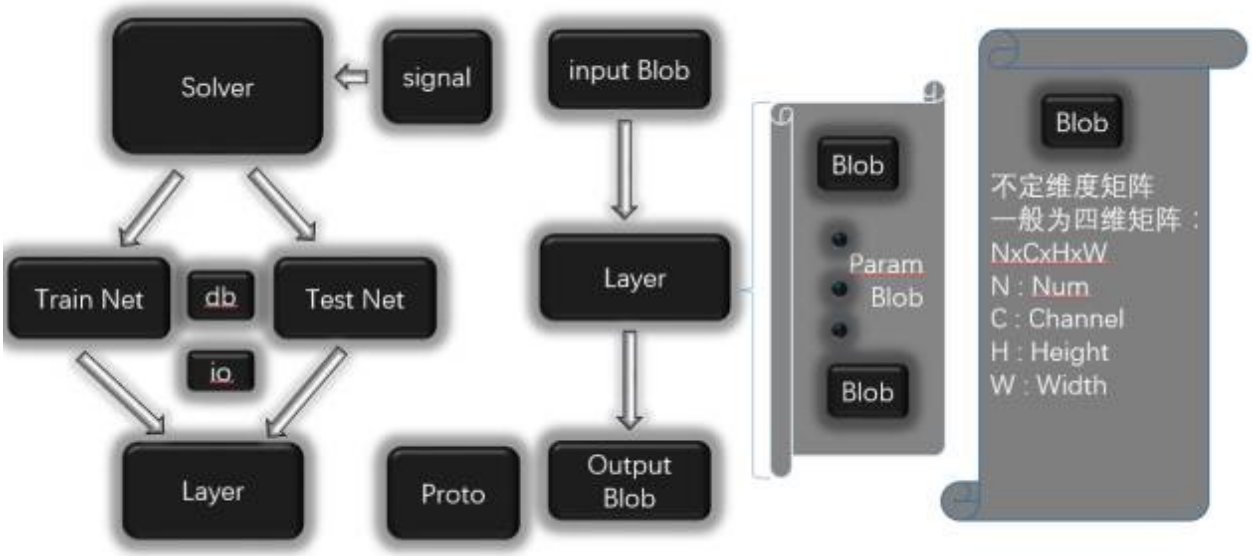


图1. Caffe源码总体架构图

2.Blob解析

下面介绍Caffe中的基本数据存储类Blob。Blob使用SyncedMemory类进行数据存储，数据成员 data_指向实际存储数据的内存或显存块，shape_存储了当前blob的维度信息，diff_这个保存了反向传递时候的梯度信息。在Blob中其实不是只有num，channel，height，width这种四维形式，它是一个不定维度的数据结构，将数据展开存储，而维度单独存在一个vector<int> 类型的shape_变量中，这样每个维度都可以任意变化。

来一起看看Blob的关键函数，data_at这个函数可以读取的存储在此类中的数据，diff_at可以用来读取反向传回来的误差。顺便给个提示，尽量使用data_at(const vector<int>& index)来查找数据。Reshape函数可以修改blob的存储大小，count用来返回存储数据的数量。BlobProto类负责了将Blob数据进行打包序列化到Caffe的模型中。

3.工厂模式说明

接下来介绍一种设计模式Factory Pattern，Caffe 中Solver和Layer对象的创建均使用了此模式，首先看工厂模式的UML的类图：

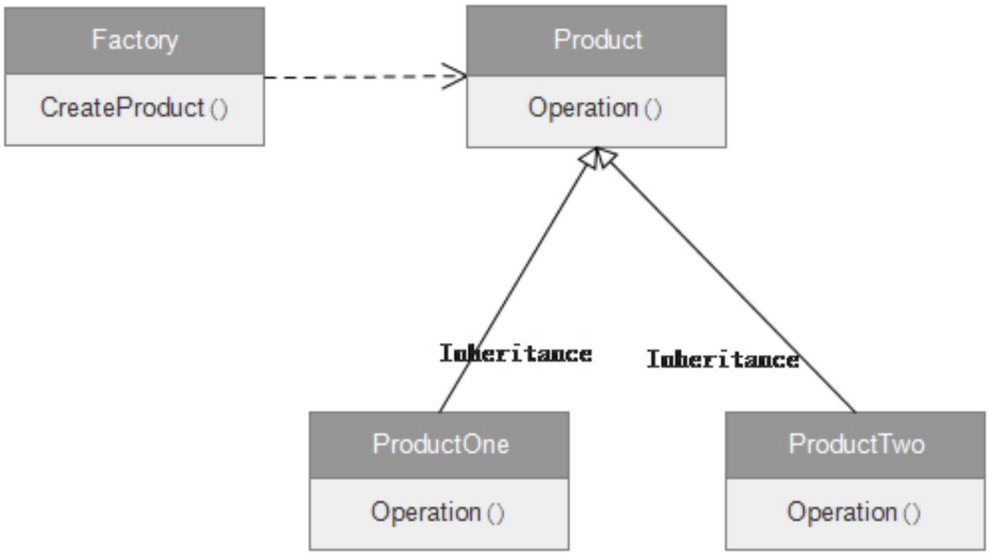


图2. 工厂模式UML类图

如同Factory生成同一功能但是不同型号产品一样，这些产品实现了同样Operation，很多人看了工厂模式的代码，会产生这样的疑问为何不new一个出来呢，这样new一个出来似乎也没什么问题吧。试想如下情况，由于代码重构类的名称改了，或者构造函数参数变化(增加或减少参数)。而你代码中又有N处new了这个类。如果你又没用工厂，就只能一个一个找来改。工厂模式的作用就是让使用者减少对产品本身的了解，降低使用难度。如果用工厂，只需要修改工厂类的创建具体对象方法的实现，而其他代码不会受到影响。

举个例子，写代码少不得饿了要加班去吃夜宵，麦当劳的鸡翅和肯德基的鸡翅都是MM爱吃的东西，虽然口味有所不同，但不管你带MM去麦当劳或肯德基，只管向服务员说“来四个鸡翅”就行了。麦当劳和肯德基就是生产鸡翅的Factory。

4.Solver解析

接下来切回正题，我们看看Solver这个优化对象在Caffe中是如何实现的。SolverRegistry这个类就是我们看到的上面的factory类，负责给我们一个优化算法的产品，外部只需要把数据和网络结构定义好，它就可以自己优化了。

Solver<Dtype>* CreateSolver(const SolverParameter& param)这个函数就是工厂模式下的CreateProduct的操作， Caffe中这个SolverRegistry工厂类可以提供给我们6种产品（优化算法）：

这六种产品的功能都是实现网络的参数更新，只是实现方式不一样。那我们来看看他们的使用流程吧。当然这些产品类似上面Product类中的Operation，每一个Solver都会继承Solve和Step函数，而每个Solver中独有的仅仅是ApplyUpdate这个函数里面执行的内容不一样，接口是一致的，这也和我们之前说的工厂生产出来的产品一样功能一样，细节上有差异，比如大多数电饭煲都有煮饭的功能，但是每一种电饭煲煮饭的加热方式可能不同，有底盘加热的还有立体加热的等。接下里我们看看Solver中的关键函数。

Solver中Solve函数的流程图如下：

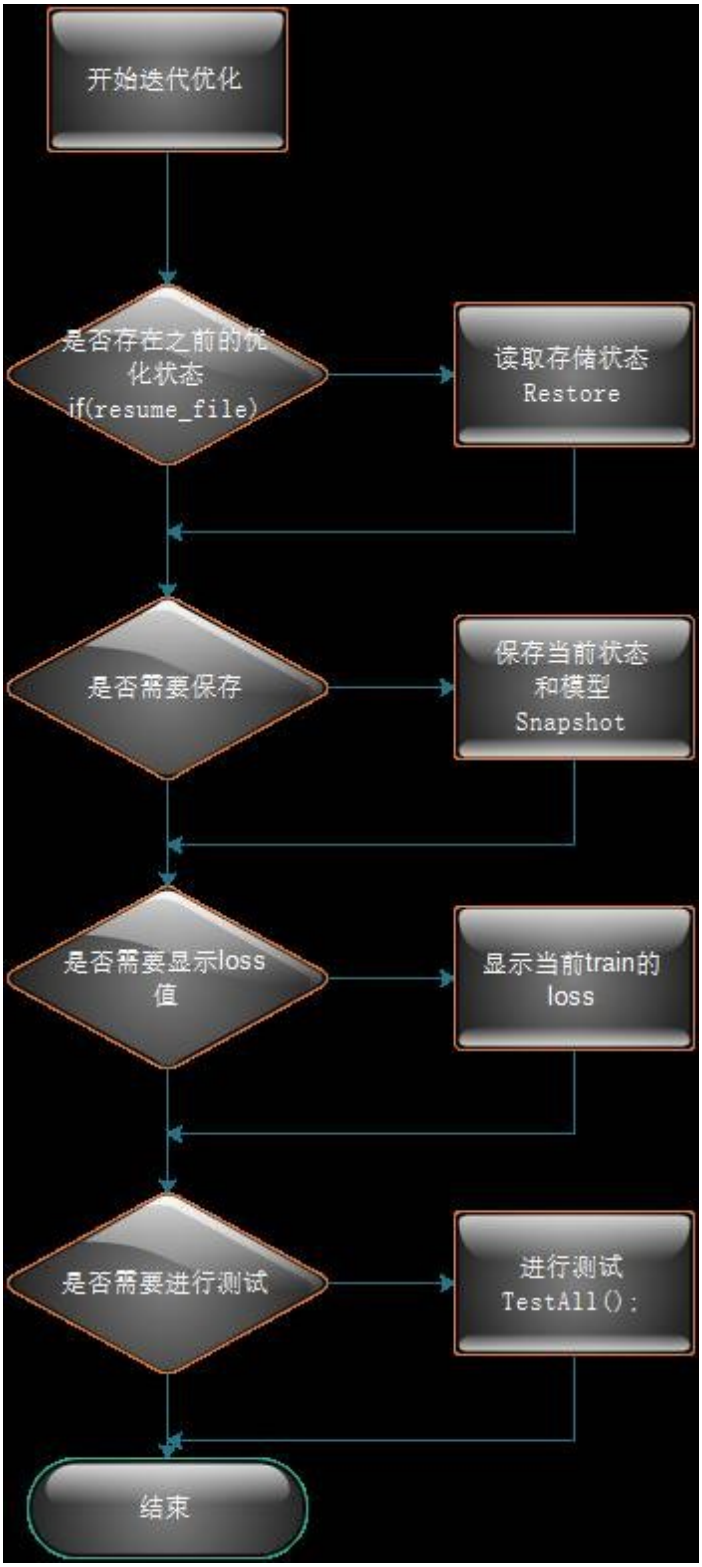


图3. Solver类Solve方法流程图

Solver类中Step函数流程图：

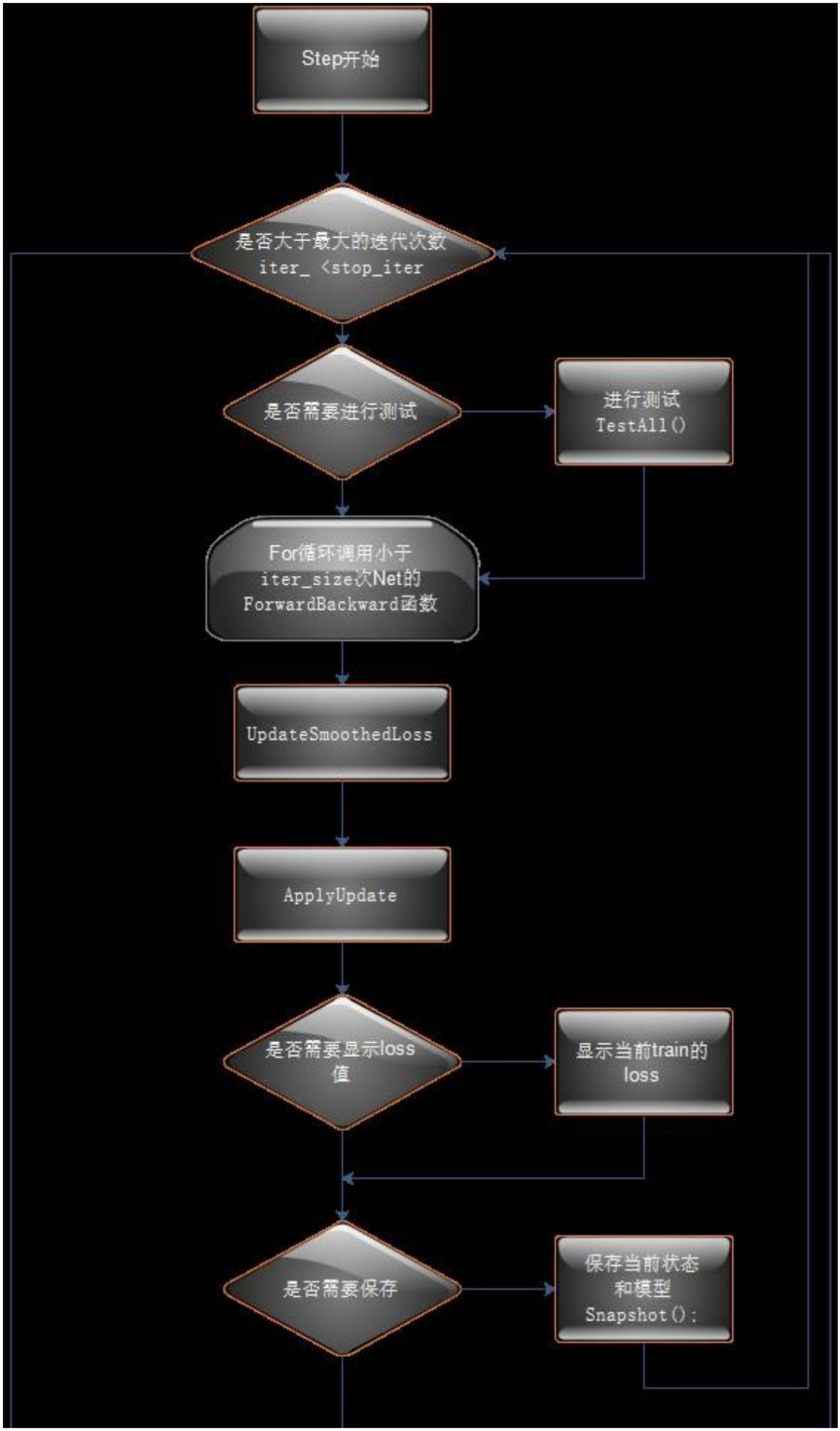

Step开始
是否大于最大的迭代次数
iter_ <stop_iter
是否需要进行测试
进行测试
TestAll()
For循环调用小于
iter_size次Net的
ForwardBackward函数
UpdateSmoothedLoss
ApplyUpdate
是否需要显示loss
值
显示当前train的
loss
是否需要保存
保存当前状态
和模型
Snapshot();

图4. Solver类Step方法流程图

Solver中关键的就是调用Sovle函数和Step函数的流程，你只需要对照Solver类中两个函数的具体实现，看懂上面两个流程图就可以理解Caffe训练执行的过程了。

5.Net类解析

分析过Solver之后我们来分析下Net类的一些关键操作。这个是我们使用Proto创建出来的深度网络对象，这个类负责了深度网络的前向和反向传递。以下是Net类的初始化方法NetInit函数调用流程：

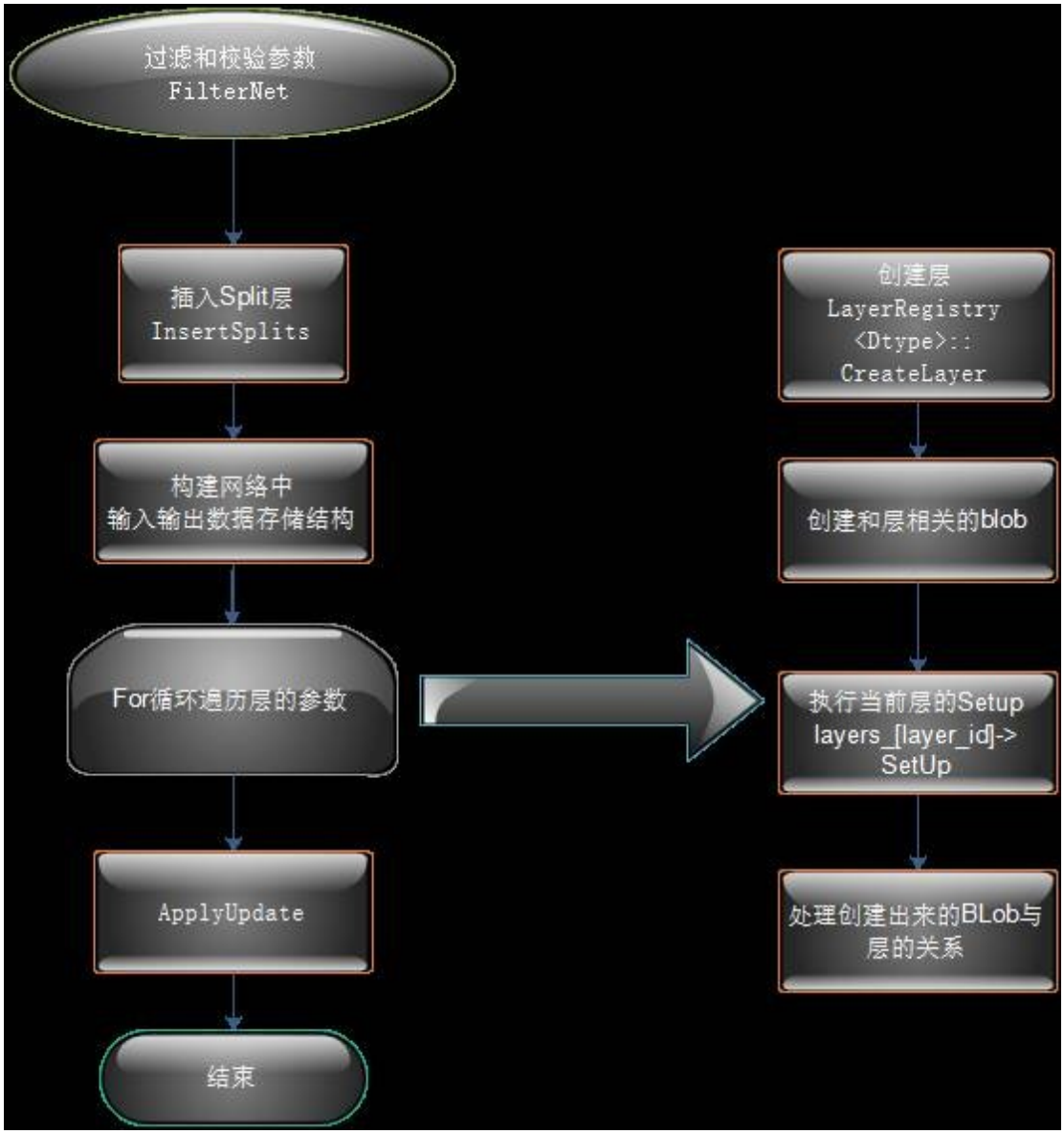


图5. Net类NetInit方法流程图

Net的类中的关键函数简单剖析

1.ForwardBackward：按顺序调用了Forward和Backward。

2.ForwardFromTo(int start, int end)：执行从start层到end层的前向传递，采用简单的for循环调用。

3.BackwardFromTo(int start, int end)：和前面的ForwardFromTo函数类似，调用从start层到end层的反向传递。

4.ToProto函数完成网络的序列化到文件，循环调用了每个层的ToProto函数。

6.Layer解析

Layer是Net的基本组成单元，例如一个卷积层或一个Pooling层。本小节将介绍Layer类的实现。

6.1Layer的继承结构

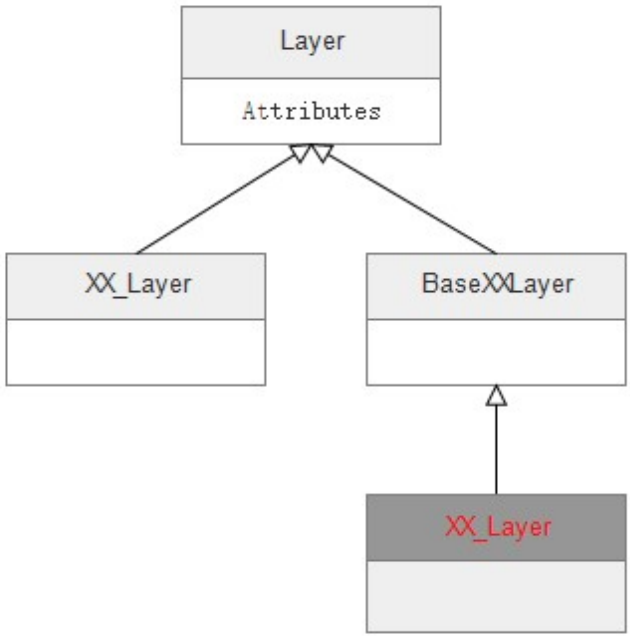


图6. Layer层的继承结构

6.2 Layer的创建

与Solver的创建方式很像，Layer的创建使用的也是工厂模式，这里简单说明下几个宏函

数：

REGISTER_LAYER_CREATOR负责将创建层的函数放入LayerRegistry。

```
116 template <typename Dtype>
117 class LayerRegisterer {
118  public:
119   LayerRegisterer(const string& type,
120                   shared_ptr<Layer<Dtype> > (*creator)(const LayerParameter&)) {
121     // LOG(INFO) << "Registering layer type: " << type;
122     LayerRegistry<Dtype>::AddCreator(type, creator);
123   }
124 };
125
126
127 #define REGISTER_LAYER_CREATOR(type, creator)                                  \
128   static LayerRegisterer<float> g_creator_f_##type(#type, creator<float>);     \
129   static LayerRegisterer<double> g_creator_d_##type(#type, creator<double>)    \
```

我们来看看大多数层创建的函数的生成宏REGISTER_LAYER_CLASS，可以看到宏函数比较简单的，将类型作为函数名称的一部分，这样就可以产生出一个创建函数，并将创建函数放入LayerRegistry。

```
131 #define REGISTER_LAYER_CLASS(type)                                             \
132   template <typename Dtype>                                                    \
133   shared_ptr<Layer<Dtype> > Creator_##type##Layer(const LayerParameter& param) \
134   {                                                                            \
135     return shared_ptr<Layer<Dtype> >(new type##Layer<Dtype>(param));           \
136   }                                                                            \
137   REGISTER_LAYER_CREATOR(type, Creator_##type##Layer)
```

REGISTER_LAYER_CREATOR(type, Creator_##type##Layer)

这段代码在split_layer.cpp文件中

```
56 INSTANTIATE_CLASS(SplitLayer);
57 REGISTER_LAYER_CLASS(Split);
58
```

REGISTER_LAYER_CLASS(Split)。
这样我们将type替换过以后给大家做个范例，参考下面的代码。

```
template <typename Dtype>
shared_ptr<Layer<Dtype> > Creator_ SplitLayer(const LayerParameter& param)
{
    return shared_ptr<Layer<Dtype> >(new SplitLayer<Dtype>(param));
}
```

当然这里的创建函数好像是直接调用，没有涉及到我们之前工厂模式的一些问题。所有的层的类都是这样吗？当然不是，我们仔细观察卷积类。

```
78
79  INSTANTIATE_CLASS(ConvolutionLayer);
80
```

卷积层怎么没有创建函数呢，当然不是，卷积的层的创建函数在LayerFactory.cpp中，截图给大家看下，具体代码如下：

```
36  // Get convolution layer according to engine.
37  template <typename Dtype>
38  shared_ptr<Layer<Dtype> > GetConvolutionLayer(
39      const LayerParameter& param) {
72
73  REGISTER_LAYER_CREATOR(Convolution, GetConvolutionLayer);
74
```

这样两种类型的Layer的创建函数都有了对应的声明。这里直接说明除了有cudnn实现的层，其他层都是采用第一种方式实现的创建函数，而带有cudnn实现的层都采用的第二种方式实现的创建函数。

6.3 Layer的初始化

介绍完创建我们看看层里面的几个函数都是什么时候被调用的。

关键函数Setup此函数在之前的流程图中的NetInit时候被调用，代码如下：

这样整个Layer初始化的过程中，CheckBlobCounts被最先调用，然后接下来是LayerSetUp，后面才是Reshape，最后才是SetLossWeights。这样Layer初始化的生命周期大家就有了了解。

6.4 Layer的其他函数的介绍

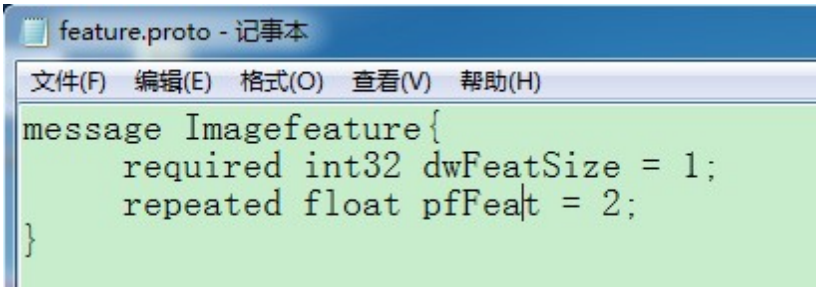
Layer的Forward函数和Backward函数完成了网络的前向和反向传递，这两个函数在自己实现新的层必须要实现。其中Backward会修改bottom中blob的diff_，这样就完成了误差的方向传导。

7.Protobuf介绍

Caffe中的Caffe.proto文件负责了整个Caffe网络的构建，又负责了Caffemodel的存储和读取。下面用一个例子介绍Protobuf的工作方式：

利用protobuffer工具存储512维度图像特征：

1.message 编写：新建txt文件后缀名改为proto,编写自己的message如下，并放入解压的protobuff的文件夹里；



其中，dwFaceFeatSize表示特征点数量；pfFaceFeat表示人脸特征。

2.打开windows命令窗口(cmd.exe)---->cd空格，把protobuff的文件路径复制粘贴进去------>enter；

3.输入指令protoc *.proto --cpp_out=. --------->enter

4.可以看到文件夹里面生成“ *.pb.h”和“*.pb.cpp”两个文件，说明成功了

5.下面可以和自己的代码整合了：

(1) 新建你自己的工程，把“ *.pb.h”和“*.pb.cpp”两个文件添加到自己的工程里，并写上#include" *.pb.h"
(2) 按照配库的教程把库配置下就可以了

VS下Protobuf的配库方法：

解决方案---->右击工程名---->属性

(1)c/c++--->常规--->附加包含目录--->

($your protobuffer include path)\protobuffer

(2)c/c++--->链接器-->常规--->附加库目录-->

($your protobuffer lib path)\protobuffer

(3) c/c++--->链接器-->输入--->附加依赖项-->

libprotobufd.lib;(带d的为debug模式)
或libprotobuf.lib;（不带d,为release模式）

使用protobuf进行打包的方法如下代码：

```
std::string write_out_string;
Feature imagefeature;
imagefeature.set_dwfacefeatsize(ffeatsize);
for (int i = 0; i < ffeatsize; i++)
{

    float value = MyCaffeFeat[i];
    imagefeature.add_pffeat(value);
}

std::fstream output(myfile_name, ios::out | ios::binary);
imagefeature.SerializeToString(&write_out_string); //序列化到string*类型write_out_string
中
int64_t length = write_out_string.size();
output.write((char*)&length, sizeof(int64_t));
output.write(write_out_string.data(), length);
output.close();
```

7.1 Caffe的模型序列化

BlobProto其实就是Blob序列化成Proto的类，Caffe模型文件使用了该类。Net调用每个层的Toproto方法，每个层的Toproto方法调用了Blob类的ToProto方法，这样完整的模型就被都序列化到proto里面了。最后只要将这个proto继承于message类的对象序列化到文件就完成了模型写入文件。Caffe打包模型的时候就只是简单调用了

WriteProtoToBinaryFile这个函数，而这个函数里面的内容如下：

```
void WriteProtoToBinaryFile(const Message& proto, const char* filename) {
  fstream output(filename, ios::out | ios::trunc | ios::binary);
  CHECK(proto.SerializeToOstream(&output));
}
```

至此Caffe的序列化模型的方式就完成了。

7.2 Proto.txt的简单说明

Caffe网络的构建和Solver的参数定义均由此类型文件完成。Net构建过程中调用ReadProtoFromTextFile将所有的网络参数读入。然后调用上面的流程进行整个caffe网络的构建。这个文件决定了怎样使用存在caffe model中的每个blob是用来做什么的，如果没有了这个文件caffe的模型文件将无法使用，因为模型中只存储了各种各样的blob数据，里面只有float值，而怎样切分这些数据是由prototxt文件决定的。

Caffe的架构在框架上采用了反射机制去动态创建层来构建Net，Protobuf本质上定义了graph，反射机制是由宏配合map结构形成的，然后使用工厂模式去实现各种各样层的创建，当然区别于一般定义配置采用xml或者json，该项目的写法采用了proto文件对组件进行组装。

总结

以上为Caffe代码架构的一个总体介绍，希望能借此帮助社区的小伙伴找到打开定制化Caffe大门的钥匙。本文作者希望借此抛砖引玉，与更多期望了解Caffe和深度学习框架底层实现的同行交流。

作者简介：**薛云峰**，(https://github.com/HolidayXue)，主要从事视频图像算法的研究，就职于浙江捷尚视觉科技股份有限公司担任深度学习算法研究员。捷尚致力于视频大数据和视频监控智能化，现诚招业内算法和工程技术人才，招聘主页http://www.icarevision.cn/job.php，联系邮箱：hr@icarevision.cn

往期精彩回顾

【Technical Review】ECCV16 Grid Loss及其在人脸检测中的应用

【ECCV2016论文速读】回归框架下的人脸对齐和三维重建

IJCAI16论文速读：Deep Learning论文选读（下）

[冠军之道] ECCV16视频性格分析竞赛冠军团队分享

IJCAI16论文速读：人脸自动美妆与深度哈希

深度学习在图像取证中的进展与趋势

欢迎关注我们！

深度学习大讲堂是高质量原创内容的平台，邀请学术界、工业界一线专家撰稿，致力于推送人工智能与深度学习最新技术、产品和活动信息！

深度学习大讲堂