Caffe Usage Summarization

ShanghaiTech University

Xu Tang

tangxu@shanghaitech.edu.cn

Homepages: http://takecareofbigboss.github.io/

阅读流程

1.caffe.proto --- caffe-master\src\caffe\proto\caffe.proto

2.Hpp file:

solver.hpp --- caffe-master\include\caffe\solver.hpp

net.hpp --- caffe-master\include\caffe\net.hpp

layer.hpp --- caffe-master\include\caffe\layer.hpp

layer派生出许多其他的类，比如
\src\caffe\layers\conv_layer, data_layer, inner_product_layer.cpp

blob.hpp --- caffe-master\include\caffe\blob.hpp

3.Cpp/cu file:caffe框架一般不需要修改，只需要增加新的层实现即可。例如像自己实现卷积层，只需从conv_layer派生一个新类，然后将几个虚函数改成自己的实现即可。

4.编写各类工具，集成到caffe内部：caffe-master\tools实现可视化等。

有blob.cpp,net.cpp,solver.cpp,没有layer.cpp!!!

阅读流程

网页参考：

1.卜居博客：

http://blog.csdn.net/kkk584520/article/category/2620633

2.菜鸟从零开始学deep learning：

http://blog.csdn.net/yihaizhiyan/article/category/2388401/3

3. 深度学习caffe的代码怎么读？知乎：

http://www.zhihu.com/question/27982282

4.Caffe四大模块介绍：

http://dirlt.com/caffe.html

5. MgLiu的专栏：

http://blog.csdn.net/qq_16055159/article/category/3107705

6. Xinyu Ou的博客(caffe安装教程)：

https://ouxinyu.github.io/Blogs/20140723001.html

7.Gan Yufei的博客：

https://yufeigan.github.io/page/2/

阅读流程

网页参考：

8.Caffe API手册，包括类列表：

http://caffe.berkeleyvision.org/doxygen/

9.Caffe github地址：

https://github.com/BVLC/caffe/

10.Caffe 框架官方介绍：

http://caffe.berkeleyvision.org/

11.Caffe各层定义文件：

http://caffe.berkeleyvision.org/tutorial/

12. guoyilin的专栏：

http://blog.csdn.net/guoyilin/article/category/2931141

13. 1983的专栏【针对代码解析】：

http://blog.csdn.net/chenriwei2/article/category/2339319

14. Caffe 深度学习框架上手教程：

http://suanfazu.com/t/caffe/281

15. DIY Deep Learning for Vision: A Tutorial with Caffe ：

http://www.tuicool.com/articles/QZrAVfI

阅读流程

网页参考：

16. DIY Deep Learning for Vision: A Tutorial with Caffe 报告笔记

http://www.tuicool.com/articles/QZrAVfI
https://docs.google.com/presentation/d/1UeKXVgRvvxg9OUdh_UiC5G71UMscNPlvArsWER41PsU/edit?pli=1#slide=id.p

17. Intel / TAU Summer Workshop on Deep Learning and Caffe：

http://courses.cs.tau.ac.il/Caffe_workshop/Bootcamp/

Caffe.proto

代码框架：

package caffe;

Message BlobProto {…}

message BlobProtoVector {…}

message Datum {…}

…

message V0LayerParameter {…}

Caffe.proto

1.关于message和package的说明：

Package caffe可以把caffe.proto里面的所有文件打包存在caffe类里面，message定义了一个需要传输的参数结构体。Required是必须有值的，optional是可选项，repeated表示后面单元为相同类型的一组向量。

这样可以通过caffe:: BlobProto* blobproto1定义一个类对象(message BlobProto)。 blobproto1.函数(对象不是指针类型) 或blobproto1->函数(对象是指针类型) 可以调用类里面定义的函数。

2.enum枚举类型调用方法：caffe::BlobProto::枚举类型里面的变量。

3.运行caffe.proto后自动生成caffe.pb.cc和caffe.pb.h两个文件，后面自己写code的时候，只需要#include” caffe.pb.h”即可。

Caffe.proto

4.函数(每个message都会自动生成以下函数)：

Set_+field 设定值的函数命名，has_ 检查field是否已经被设置，clear_用于清理field，mutable_用于设置string的值，_size用于获取重复的个数。

5.Message的tag：

每个message里面的每个field都对应一个tag，分别是1~15或者以上，比如required string number=1；

Caffe.proto

6.Message类别：

属于blob的： BlobProto, BlobProtoVector, Datum。

属于layer的： FillerParameter, LayerParameter, ArgMaxParameter,TransformationParameter, LossParameter, AccuracyParameter, ConcatParameter, ContrastiveLossParameter, ConvolutionParameter, DataParameter, DropoutParameter, DummyDataParameter, EltwiseParameter, ExpParameter, HDF5DataParameter, HDF5OutputParameter, HingeLossParameter, ImageDataParameter, InfogainLossParameter, InnerProductParameter,

LRNParameter, MemoryDataParameter, MVNParameter, PoolingParameter, PowerParameter, PythonParameter, ReLUParameter, SigmoidParameter, SliceParameter, SoftmaxParameter, TanHParameter, ThresholdParameter等。

属于net的： NetParameter, SolverParameter, SolverState, NetState, NetStateRule, ParamSpec。

Caffe.proto

NetParameter弄清楚NetParameter类的组成，也就明白了.Caffemodel的具体数据构成；

SolverState类记录的是当前迭代状态和参数设置，与.solverstate文件有关系；

Caffe.proto

网页参考：

1.Caffe代码导读（1）：
http://blog.csdn.net/kkk584520/article/details/41046827

2. Protocol Buffers 简述：

http://name5566.com/2381.html

3. caffe源码解析 — caffe.proto：

http://blog.csdn.net/qq_16055159/article/details/45115359

Blob.hpp

Blob：是基础的数据结构，是用来保存学习到的参数以及网络传输过程中产生数据的类。在更高一级的Layer中Blob用下面的形式表示学习到的参数：vector<shared_ptr<Blob<Dtype> > > blobs_;

Blob是row-major保存的，因此在(n, k, h, w)位置的值物理位置为((n * K + k) * H + h) * W + w，其中Number/N是batch size。

Blob<Dtype> diff_; // cached for backward pass

Blob<Dtype> dist_sq_; // cached for backward pass

Blob<Dtype> diff_sq_; // tmp storage for gpu forward pass

Blob<Dtype> summer_vec_; // tmp storage for gpu forward pass

Blob.hpp

1.代码框架：

Namespace caffe{class Blob{…}}

其中template <typename Dtype>表示函数模板，Dtype可以表示int，double等数据类型。

2.Blob内部有两个字段data和diff。Data表示流动数据(输出数据)，而diff则存储BP的梯度。

3.Blob是用以存储数据的四维数组，分别由下面组成：

对于数据：num(输入数据量，比如sgd时，mini-batch的大小)，channels(通道数量),height(图片的高度),width(图片的宽度)。

对于卷积权重：output*input*height*width

对于卷积偏置：output*1*1*1

对于卷积层输出：输入图片数量*对应feature maps数量*输出图片的高度*输出图片的宽度；

Blob.hpp

4.包含头文件名称：

#include “caffe/common.hpp”单例化caffe类，并且封装了boost和cuda随机数生成的函数，提供了统一接口。

#include "caffe/proto/caffe.pb.h"

#include “caffe/syncedmem.hpp”主要是分配内存和释放内存的。而class SyncedMemory定义了内存分配管理和CPU与GPU之间同步的函数，也没啥特别的。Blob会使用SyncedMem自动决定什么时候去copy data以提高运行效率，通常情况是仅当gnu或cpu修改后有copy操作。

#include “caffe/util/math_functions.hpp”封装了很多cblas矩阵运算，基本是矩阵和向量的处理函数。

Blob.hpp

5.函数：

Reshape()可以改变一个blob的大小；

ReshapeLike()为data和diff重新分配一块空间，大小和另一个blob的一样；

Num_axes()返回的是blob的大小；

Count()计算得到count=num*channels*height*width。

Offset()可得到输入blob数据(n,c,h,w)的偏移量位置；

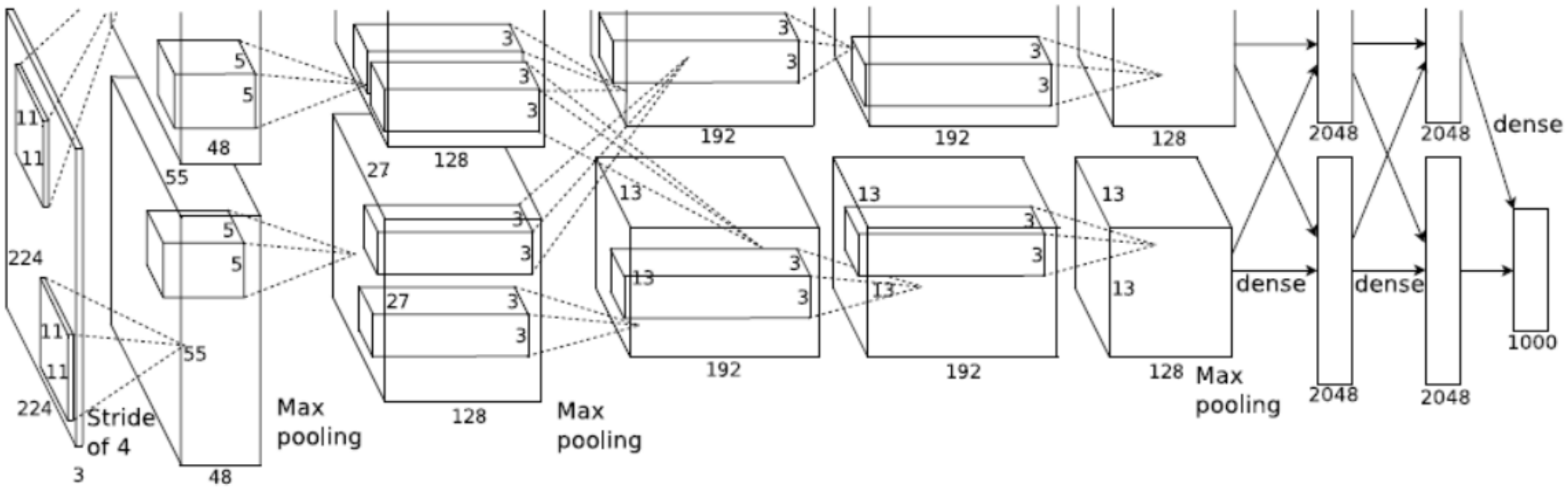
CopyFrom()从source拷贝数据，copy_diff来作为标志区分是拷贝data还是diff。

FromProto()从proto读数据进来，其实就是反序列化。

ToProto()把blob数据保存到proto中。

ShareDate()/ShareDiff()从other的blob复制data和diff的值；

Blob.hpp



第一层卷积层blob的size为[n,48,55,55];
第二层卷积层blob的size为[n,128,27,27];
第六个pooling层blob的size为[n,2048,1,1];
第七个pooling层blob的size为[n,2048,1,1];

Blob.hpp

网页参考：

1.caffe源码解析之blob(1)：

http://blog.csdn.net/chaojichaoachao/article/details/43530605

2. caffe源码解析 — blob.cpp：

http://blog.csdn.net/qq_16055159/article/details/45099547

3. Caffe-代码解析-Blob

http://blog.csdn.net/chenriwei2/article/details/46367023

Layer.hpp

Layer：是网络的基本单元，由此派生出了各种层类。修改这部分的人主要是研究特征表达方向的。Layer类派生出来的层类通过这实现这两个虚函数Forward()和Backward()，产生了各式各样功能的层类。Forward是从根据bottom计算top的过程，Backward则相反（根据top计算bottom）。

在网路结构定义文件（*.proto）中每一层的参数bottom和top数目就决定了vector中元素数目。

Layer.hpp

1.代码框架：

Namespace caffe{class Layer{…}}

2.包含头文件：

#include "caffe/blob.hpp"

#include "caffe/common.hpp"

#include "caffe/layer_factory.hpp"

#include "caffe/proto/caffe.pb.h"

#include “caffe/util/device_alternate.hpp“ 定义使用GPU or CPU

Layer.hpp

3.Layer中三个重要参数：

LayerParameter layer_param_;这个是protobuf文件中存储的layer参数。

vector<share_ptr<Blob<Dtype>>> blobs_;这个存储的是layer的参数，在程序中用的。Layer学习到的参数。

vector<bool> param_propagate_down_;这个bool表示是否计算各个blob参数的diff，即传播误差。

Layer.hpp

4.函数：

Layer()尝试从protobuf读取参数；

SetUp()根据实际的参数设置进行实现，对各种类型的参数初始化；

Forward()和Backward()对应前向计算和反向更新，输入统一都是bottom，输出为top，其中Backward里面有个propagate_down参数，用来表示该Layer是否反向传播参数。

Caffe::mode()具体选择使用CPU或GPU操作。

Layer.hpp

网页参考：

1.caffe源码解析之Layer层(1)：

http://blog.csdn.net/chaojichaoachao/article/details/43530629

2. Caffe学习笔记3-Layer的相关学习

https://yufeigan.github.io/2014/12/09/Caffe%E5%AD%A6%E4%B9%A0%E7%AC%94%E8%AE%B03-Layer%E7%9A%84%E7%9B%B8%E5%85%B3%E5%AD%A6%E4%B9%A0/

3. Caffe-代码解析-Layer：

http://blog.csdn.net/chenriwei2/article/details/46417293

4.Caffe中layer每一层结构解释：

http://demo.netfoucs.com/danieljianfeng/article/details/42929283

Net.hpp

Net：是网络的搭建，将Layer所派生出层类组合成网络。

Net用容器的形式将多个Layer有序地放在一起，其自身实现的功能主要是对逐层Layer进行初始化，以及提供Update( )的接口（更新网络参数），本身不能对参数进行有效地学习过程。

vector<shared_ptr<Layer<Dtype> > > layers_;

Net也有它自己的Forward()和Backward(),他们是对整个网络的前向和反向传导，各调用一次就可以计算出网络的loss了。

Net由一系列的Layer组成(无回路有向图DAG)，Layer之间的连接由一个文本文件描述。模型初始化Net::Init()会产生blob和layer并调用Layer::SetUp。在此过程中Net会报告初始化进程。这里的初始化与设备无关，在初始化之后通过Caffe::set_mode()设置Caffe::mode()来选择运行平台CPU或GPU，结果是相同的。

Net.hpp

1.代码框架：

Namespace caffe{class Net{…}}

2.包含头文件：

#include "caffe/blob.hpp"

#include "caffe/common.hpp"

#include "caffe/layer.hpp"

#include "caffe/proto/caffe.pb.h"

Net.hpp

3.函数：

Init()初始化函数，用于创建blobs和layers，用于调用layers的setup函数来初始化layers。

ForwardPrefilled()用于前馈预先填满，即预先进行一次前馈。

Forward()把网络输入层的blob读到net_input_blobs_，然后进行前馈，计算出loss。Forward的重载，只是输入层的blob以string的格式传入。

Backward()对整个网络进行反向传播。

Reshape()用于改变每层的尺寸。

Update()更新params_中blob的值。

ShareTrainedLayersWith(Net* other)从Other网络复制某些层 。

Net.hpp

CopyTrainedLayersFrom()调用FromProto函数把源层的blob赋给目标层的blob。

ToProto()把网络的参数存入prototxt中。

bottom_vecs_存每一层的输入blob指针

bottom_id_vecs_存每一层输入(bottom)的id

top_vecs_存每一层输出(top)的blob

params_lr()和params_weight_decay()学习速率和权重衰减;

blob_by_name()判断是否存在名字为blob_name的blob;

Net.hpp

FilterNet()给定当前phase/level/stage，移除指定层。

StateMeetsRule()中net的state是否满足NetStaterule。

AppendTop()在网络中附加新的输入或top的blob。

AppendBottom()在网络中附加新的输入或bottom的blob。

AppendParam()在网络中附加新的参数blob。

GetLearningRateAndWeightDecay() 收集学习速率和权重衰减，即更新params_、params_lr_和params_weight_decay_；

Net.hpp

网页参考：

1. caffe源码解析 — net.cpp:

http://blog.csdn.net/qq_16055159/article/details/45057297

Solver.hpp

Solver：是Net的求解，修改这部分人主要会是研究DL求解方向的。

这个类中包含一个Net的指针，主要是**实现了训练模型参数所采用的优化算法**，它所派生的类就可以对整个网络进行训练了。

shared_ptr<Net<Dtype> > net_;

不同的模型训练方法通过重载函数ComputeUpdateValue( )实现计算update参数的核心功能。

最后当**进行整个网络训练过程**（也就是你运行Caffe训练某个模型）的时候，实际上是在运行caffe.cpp中的train( )函数，而这个函数实际上是实例化一个Solver对象，初始化后调用了Solver中的Solve( )方法。而这个Solve( )函数主要就是在迭代运行下面这两个函数，就是刚才介绍的哪几个函数。

Solver.hpp

1.代码框架：

Namespace caffe{class Solver{…}}

class SGDSolver : public Solver<Dtype>{…}

class NesterovSolver : public SGDSolver<Dtype>{…}

class AdaGradSolver : public SGDSolver<Dtype> {…}

Solver<Dtype>* GetSolver(const SolverParameter& param) {…}

2.包含头文件：

#include "caffe/net.hpp“

Solver.hpp

3.函数：

Solver() 构造函数，初始化两个net类，分别是net和test_net类，并调用init函数。

Init() 初始化网络，解释详见网页；

Solve()训练网络。**步骤**：

1. 设置Caffe的mode（GPU还是CPU）
2. 如果是GPU且有GPU芯片的ID，则设置GPU
3. 设置当前阶段（TRAIN还是TEST/TRAIN）
4. 调用PreSolve函数：PreSolve()
5. 调用Restore函数：Restore(resume_file)
6. 调用一遍Test()，判断内存是否够
7. 对于每一次训练时的迭代(遍历整个网络)

Solver.hpp

对于每一次训练时的迭代(遍历整个网络):

while (iter_++ < param_.max_iter())

1.计算loss： loss = net_->ForwardBackward(bottom_vec)

2.调用ComputeUpdateValue函数；

3.输出loss；

4. 达到test_interval时调用Test()

5. 达到snapshot时调用snapshot()

Solver.hpp

Test() 测试网络。步骤：

1. 设置当前阶段（TRAIN还是TEST/TEST）
2. 将test_net_指向net_，即对同一个网络操作
3. 对于每一次测试时的迭代：for (int i = 0; i < param_.test_iter(); ++i)

    3.1.用下面语句给result赋值net_output_blobs_

    result = test_net_->Forward(bottom_vec, &iter_loss);

    3.2.第一次测试时：取每一个输出层的blob result_vec = result[j]->cpu_data()，把每一个blob的数据（降为一维）存入一个vector–“test_score”

    不是第一次测试：用 test_score[idx++] += result_vec[k] 而不是 test_score.push_back(result_vec[k])

把输出层对应位置的blob值累加 test_score[idx++] += result_vec[k]。

    3.3.是否要输出Test loss,是否要输出test_score;

    3.4.设置当前阶段（TRAIN还是TEST/TRAIN）

Solver.hpp

Snapshot()输出当前网络状态到一个文件中；

Restore()从一个文件中读入网络状态，并可以从那个状态恢复；

GetLearningRate()得到学习率；

PreSolve()提前训练，详见网页链接；

ComputeUpdateValue()用随机梯度下降法计算更新值，详见网页；

Solver.hpp

网页参考：

1. caffe源码解析 — solver.cpp

http://blog.csdn.net/qq_16055159/article/details/45068147

其他文件作用caffe-master\include\caffe\

1. **filler.hpp** 的作用是在网络初始化时，根据layer的定义进行初始参数的填充。
2. **data_layers.hpp**原始数据的输入层，处于整个网络的最底层，它可以从数据库leveldb、lmdb中读取数据，也可以直接从内存中读取，还可以从hdf5，甚至是原始的图像读入数据。作为网络的最底层，主要实现数据格式的转换。
3. **internal_thread.hpp**里面封装了pthread函数，继承的子类可以得到一个单独的线程，主要作用是在计算当前的一批数据时，在后台获取新一批的数据。
4. **neuron_layers.hpp**输入了data后，就要计算了，比如常见的sigmoid、tanh等等。这些都计算操作被抽象成了neuron_layers.hpp里面的类NeuronLayer，这个层只负责具体的计算，因此明确定义了输入ExactNumBottomBlobs()和ExactNumTopBlobs()都是常量1,即输入一个blob，输出一个blob。**其派生类**主要是元素级别的运算（比如Dropout运算，激活函数ReLu，Sigmoid等），运算均为同址计算（in-place computation，返回值覆盖原值而占用新的内存）。

其他文件作用caffe-master\include\caffe\

5. **common_layers.hpp网络连接层和激活函数定义在这。**剩下的那些复杂的计算则通通放在了common_layers.hpp中。像ArgMaxLayer、ConcatLayer、FlattenLayer、SoftmaxLayer、SplitLayer和SliceLayer等各种对blob增减修改的操作。

6. **loss_layers.hpp**前面的data_layer和common_layer都是中间计算层，虽然会涉及到反向传播，但传播的源头来自于loss_layer，即网络的最终端。这一层因为要计算误差，所以输入都是2个blob，输出1个blob。**其派生类**会产生loss，只有这些层能够产生loss。

7. **vision_layers.hpp特征表达类**主要是图像卷积的操作，像convolusion、pooling、LRN都在里面，按官方文档的说法，是可以输出图像的。实现特征表达功能，更具体地说包含卷积操作，Pooling操作，他们基本都会产生新的内存占用（Pooling相对较小）。

Data_layers.hpp

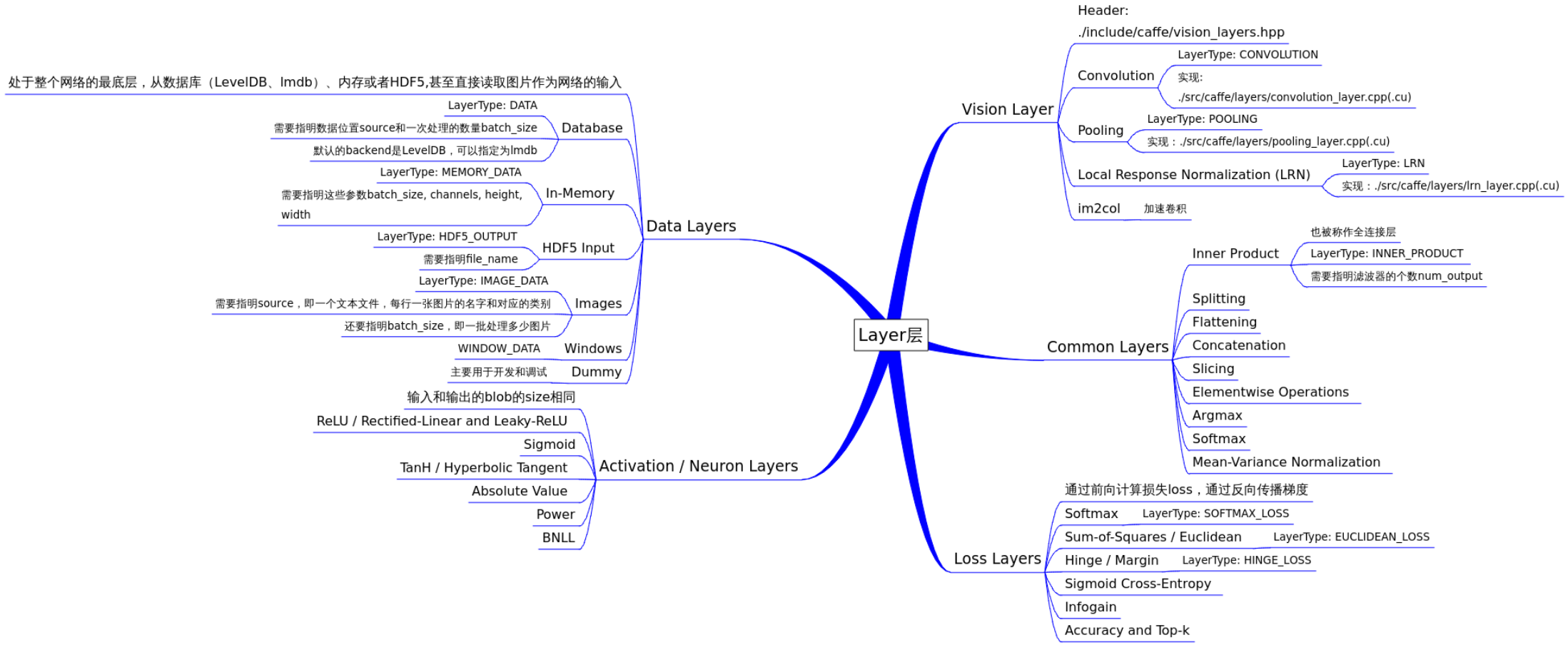
```
#include "boost/scoped_ptr.hpp"
#include "hdf5.h"
#include "leveldb/db.h"
#include "lmdb.h"
//前4个都是数据格式有关的文件
#include "caffe/blob.hpp"
#include "caffe/common.hpp"
#include "caffe/data_transformer.hpp"
#include "caffe/filler.hpp"
#include "caffe/internal_thread.hpp"
#include "caffe/layer.hpp"
#include "caffe/proto/caffe.pb.h“
```

```
namespace caffe {

template <typename Dtype>
class BaseDataLayer : public Layer<Dtype> {…}

template <typename Dtype>
class BasePrefetchingDataLayer :
   public BaseDataLayer<Dtype>, public
InternalThread {…}
            …
}
#endif
```

其他文件作用

结合官方文档，再加画图和看代码，终于对整个layer层有了个基本认识：data负责输入，vision负责卷积相关的计算，neuron和common负责中间部分的数据计算，而loss是最后一部分，负责计算反向传播的误差。具体的实现都在src/caffe/layers里面，慢慢再研究研究。

其他文件作用


Layer层
Data Layers
处于整个网络的最底层，从数据库（LevelDB、lmdb）、内存或者HDF5,甚至直接读取图片作为网络的输入
Database
LayerType: DATA
需要指明数据位置source和一次处理的数量batch_size
默认的backend是LevelDB，可以指定为lmdb
In-Memory
LayerType: MEMORY_DATA
需要指明这些参数batch_size, channels, height, width
HDF5 Input
LayerType: HDF5_OUTPUT
需要指明file_name
Images
LayerType: IMAGE_DATA
需要指明source，即一个文本文件，每行一张图片的名字和对应的类别
还要指明batch_size，即一批处理多少图片
Windows
WINDOW_DATA
Dummy
主要用于开发和调试
Activation / Neuron Layers
输入和输出的blob的size相同
ReLU / Rectified-Linear and Leaky-ReLU
Sigmoid
TanH / Hyperbolic Tangent
Absolute Value
Power
BNLL
Vision Layer
Header:
./include/caffe/vision_layers.hpp
Convolution
LayerType: CONVOLUTION
实现:
./src/caffe/layers/convolution_layer.cpp(.cu)
Pooling
LayerType: POOLING
实现：./src/caffe/layers/pooling_layer.cpp(.cu)
Local Response Normalization (LRN)
LayerType: LRN
实现：./src/caffe/layers/lrn_layer.cpp(.cu)
im2col
加速卷积
Common Layers
Inner Product
也被称作全连接层
LayerType: INNER_PRODUCT
需要指明滤波器的个数num_output
Splitting
Flattening
Concatenation
Slicing
Elementwise Operations
Argmax
Softmax
Mean-Variance Normalization
Loss Layers
通过前向计算损失loss，通过反向传播梯度
Softmax
LayerType: SOFTMAX_LOSS
Sum-of-Squares / Euclidean
LayerType: EUCLIDEAN_LOSS
Hinge / Margin
LayerType: HINGE_LOSS
Sigmoid Cross-Entropy
Infogain
Accuracy and Top-k

代码解释部分

1. “.prototxt”解释：

layer {

name: “cifar“ type: ”Data“ layer层是data_layer层的类型；

top: “data“ top: ”label“ 第一层是数据本身，第二层是ground truth；

include {phase: TRAIN}

transform_param { mean_file: “examples/cifar10/mean.binaryproto” }做数据的预处理工作，需要用绝对路径；

data_param { source: “examples/cifar10/cifar10_train_lmdb“ batch_size: 100 backend: LMDB}} source表示数据的目录名称，需要用绝对路径。batch_size表示一次处理的输入的数量，backend用于选择使用leveldc或者lmdb

代码解释部分

2.param{lr_mult:1} 表示滤波器学习率调整的参数；
Param{lr_mult:2} 表示偏置学习率调整的参数；
num_output: 过滤器的个数；
pad:2 表示在图片的四周分别加长2个单位长度；
kernel_size 过滤器的大小；
stride: 滤波器步长；
weight_filler: 参数的初始化方法；
bias_filler: 偏置的初始化方法；

```
blobs_lr: 1          # learning rate multiplier for the filters
blobs_lr: 2          # learning rate multiplier for the biases
weight_decay: 1      # weight decay multiplier for the filters
weight_decay: 0      # weight decay multiplier for the biases
```

代码解释部分

test_iter:1000是指测试的批次；
test_interval: 1000 是指每 1000 次迭代测试一次；
lr_policy: “step”学习率变化；
gamma: 0.1 学习率变化的比率；
stepsize: 100000 每 100000 次迭代减少学习率；
display: 20 每 20 层显示一次；
max_iter: 450000 最大迭代次数；
momentum: 0.9 冲量；
snapshot: 10000 每迭代 10000 次显示状态；
solver_mode: GPU 末尾加一行，代表用 GPU 进行；

代码解释部分

input_dim: 1 图片数量；

input_dim: 3 表示RGB三个channels；

input_dim: 32 表示图片高度；

input_dim: 32 表示图片宽度；

Solver_mode:CPU 用于设置CPU还是GPU训练模型；

每100次迭代次数显示一次训练时候的lr(learning rate)/loss(训练损失函数)/score 0(准确率)/score 1(测试损失函数)

当5000次迭代后，模型的参数存储在二进制protobuf格式文件中，一般文件名叫做cifar10_quick_iter_5000.protobuf;

Snapshot：5000

Snapshot_prefix: “cifar10_quick” 表示每5000次迭代，存储一次数据到电脑，名字是cifar10_quick_iter_5000.caffemodel和cifar10_quick_iter_5000.solverstate；

Ubuntu操作

网页参考：

1. UNIX中常用命令：
   http://blog.csdn.net/yihaizhiyan/article/details/41802923
2. Vim操作：
   http://blog.csdn.net/yihaizhiyan/article/details/39585447

Ubuntu操作

1. 常用命令：

Chmod u+x get_cifar10.sh 获取权限；

sudo sh ___.sh 用于执行脚本；

find ‘pwd’ ./examples/images –type f –exec echo {} \; > examples/_temp/temp.txt 用于将images里面所有文件名输出到temp.txt；

sed “s/$/ 0/” examples/_temp/temp.txt > examples/_temp/file_list.txt在每行后面添加0；

find –name *.jpeg |cut –d –d ‘/’ -f2-3 > train.txt 用于将所有jpeg格式的文件名输入到txt文本中；

```
for name in /path/to/imagenet/val/*.JPEG; do
        convert -resize 256x256\! $name $name
done
```

用于将所有的图片转换成256*256的固定大小；

Ubuntu操作

$TOOLS/convert_imageset \ $TRAIN_DATA_ROOT \ $DATA/train.txt \ $EXAMPLE/ilsvrc12_train_lmdb 用于将数据和对应的label文件生成lmdb文件；

\tools\compute_image_mean.cpp 输入文件 输出文件 用于计算均值；

export BLAS=/path/to/libblas.so用于设置路径，然后就可以直接使用$BLAS来当做绝对路径；

ln –sf $CAFFE_ROOT external/caffe 创建超链接映射

make matcaffe 使用matlab接口的caffe文件夹

make pycaffe 使用python接口的caffe文件夹

Ipython notebook ./examples/Siamese/mnist_Siamese.ipynb可以直接进入ipython界面；

其他操作

网页参考：

1. matlab txt文件按行打乱顺序 txt按行读取 按行写入 打乱顺序

http://blog.csdn.net/yihaizhiyan/article/details/44084943

2. caffe 如何prototxt文件中，添加并传递变量

http://blog.csdn.net/yihaizhiyan/article/details/42086319

3. Caffe 把single label-->multi-label涉及到的几个函数

http://blog.csdn.net/yihaizhiyan/article/details/42061691

4. C/C++ 定义向量、赋值和使用

http://blog.csdn.net/yihaizhiyan/article/details/45486345

5.如何在caffe中添加新类型的layer

http://blog.sciencenet.cn/home.php?mod=space&uid=1583812&do=blog&id=850247

6. caffe&windows工作过程 — cifar10为例：

http://blog.csdn.net/qq_16055159/article/details/45130509

其他操作

网页参考：

7.train_net.cpp是caffe的主函数所在处：

http://blog.csdn.net/qq_16055159/article/details/45056129

8. **Caffemodel**解析:

http://www.cnblogs.com/zzq1989/p/4439429.html

9. Caffe源码导读（7）： LRN层的实现:

http://www.itnose.net/detail/6177501.html

10.Caffe学习笔记5-BLAS与boost::thread加速：基于cblas的卷积等操作的实现

https://yufeigan.github.io/2015/01/02/Caffe%E5%AD%A6%E4%B9%A0%E7%AC%94%E8%AE%B05-BLAS%E4%B8%8Eboost-thread%E5%8A%A0%E9%80%9F/

11. Deep Learning 优化方法总结：

https://yufeigan.github.io/2014/11/29/Deep-Learning-%E4%BC%98%E5%8C%96%E6%96%B9%E6%B3%95%E6%80%BB%E7%BB%93/

其他操作

网页参考：

12. caffe python批量抽取图像特征：
http://blog.csdn.net/guoyilin/article/details/42886365
13. cuDNN: efficient Primitives for Deep Learning
http://blog.csdn.net/guoyilin/article/details/44222087
14.caffe tools command(compute the mean image，convert binary proto to npy):
http://blog.csdn.net/guoyilin/article/details/45694163
15. caffe c++ 抽取图片特征：
http://blog.csdn.net/guoyilin/article/details/42886483
16. caffe python visualization程序解析：
http://blog.csdn.net/guoyilin/article/details/42873747

其他操作

网页参考：

17.Caffe中添加自己的网络层：

http://blog.csdn.net/chenriwei2/article/details/46432727

18. Caffe 代码解析-convert_imageset：

http://blog.csdn.net/chenriwei2/article/details/46361333

19. Caffe-代码解析-compute_image_mean：

http://blog.csdn.net/chenriwei2/article/details/46362851

20. Caffe 多目标输出探究：

http://blog.csdn.net/chenriwei2/article/details/46300981

21. Caffe中的损失函数解析：

http://blog.csdn.net/chenriwei2/article/details/45291739

其他操作

网页参考：

22. Use Caffe to extract features of each layer：

http://bean.logdown.com/posts/211192-caffe-use-caffe-to-extract-features-of-each-layer

23. cuda-convnet 在其他数据集上的使用教程：

http://blog.csdn.net/kuaitoukid/article/details/39318629

24.如何在caffe中添加新的layer：

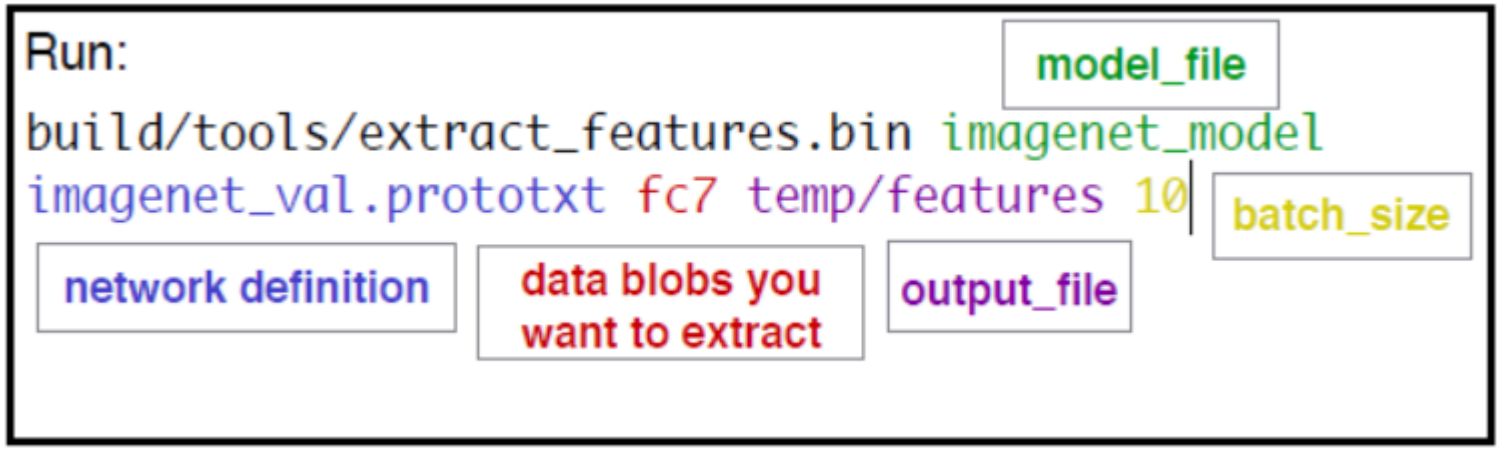
http://blog.csdn.net/kuaitoukid/article/details/41865803

常用文件

1.读取数据：examples/mnist/convert_mnist_data.cpp

2.定义自己的网络： examples/mnist/lenet_train.prototxt 蓝色表示需要定义的layers，黄色表示data blobs；

3.特征提取：examples/feature_extraction/imagenet_val.prototxt



Caffemodel读取参数

使用matlab得到caffemodel里面学习到的权值；

a. 打开matlab；
b. make sure that the path to your matcaffe (CAFFE ROOT/matlab/caffe) is in your matlab path. If not, add it manually or write: addpath(your path) in matlab；
c. caffe(‘init’, ‘your_model _deploy.prototxt’,‘your_model.caffemodel’,‘test’)注意deploy是参数设置文件，而不是参数定义文件；
d. weights = cae(‘get weights’)其中的weights将包括所有层的权重和偏置；
e. save('/path/to/folder/lename.mat', 'weights')用于保存权重；

finetuning models

—> what if you want to transfer the weight of a existing model to finetune another dataset / task

- Simply change a few lines in the layer definition

new name = new params

Input:
A different source

```
layers {
  name: "data"
  type: DATA
  data_param {
    source:
"ilsvrc12_train_leveldb"
    mean_file: "../../data/
ilsvrc12"
    ...
  }
...
```

```
layers {
  name: "data"
  type: DATA
  data_param {
    source: "style_leveldb"
    mean_file: "../../data/
ilsvrc12"
    ...
  }
  ...
...
```

Last Layer:
A different classifier

```
layers {
  name: "fc8"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 1000
    ...
  }
```

```
layers {
  name: "fc8-style"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 20
    ...
  }
```

finetuning models

new caffe:

```
> caffe train -solver models/finetune_flickr_style/solver.prototxt
              -weights bvlc_reference_caffenet.caffemodel
```

Under the hood (loosely speaking):

```
net = new Caffe::Net("style_solver.prototxt");
net.CopyTrainedNetFrom(pretrained_model);
solver.Solve(net);
```

常用blas函数Basic Linear Algebra Subprograms

1.Y=alpha * X +beta*Y

void caffe_cpu_axpby<float>(const int N, const float alpha, const float* X, const float beta, float* Y) {cblas_saxpby(N, alpha, X, 1, beta, Y, 1);}

2. Y=beta

Void caffe_set(const int N, const float beta, float* Y){…}

3. diff_=X_i – Y_i;

Void caffe_sub(const int N, float* X, float* Y,diff_){…}

http://www.cnblogs.com/huashiyiqike/p/3886670.html

已知layer输入，如何求输出尺寸

已知pad_h,kernel_h,stride_h,input_h，并且input_h是输入layer的图片的高度，则layer输出的图片高度为output_h:

Output_h = (input_h + 2*pad_h – kernel_h)/stride_h + 1;

比如输入的一张图片是32*32，经过一个pad=2,kernel_size=5,stride=1的卷积层后，输出的尺寸中output_h=(32+2*2-5)/1+1=32,同理output_w =32;

可视化参数

命令行输入ipython notebook，如果出现问题，则对应输入sudo pip install xxxxxx;

http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/filter_visualization.ipynb

[(k, v.data.shape) for k, v in net.blobs.items()] 输出每层layer的输入图片数量*对应feature maps数量*输出图片的高度*输出图片的宽度；比如('conv1', (1, 32, 32, 32))中1是输入图片的数量，第一个32是对应feature maps数量，输出图片的尺寸是32*32；

[(k, v[0].data.shape) for k, v in net.params.items()] 输出网络学到的W参数；比如('conv1', (32, 3, 5, 5))中3是输入通道数，32是卷积层输出的feature maps数目，5*5是kernel size；但是对应于matlab提取出来的特征，保存成val(:,:,1,1)/ val(:,:,2,1)/ val(:,:,3,1)/ val(:,:,1,2)/ val(:,:,2,2)/ val(:,:,3,2)/…/ val(:,:,1,32)/ val(:,:,2,32)/ val(:,:,3,32)这种形式；其中size(val(:,:,1,1))=5*5*1*1。

将binaryproto转换成npy文件

注意需要把以下代码写成creat_mean_npy.py的脚本，然后放在cd caffe/python目录下运行，运行时候在命令行输入python creat_mean_npy.py xx.binaryproto xx.npy

```
import numpy as np

import sys

import caffe
caffe_root = '/path/to/caffe/'
sys.path.insert(0, caffe_root + 'python')
if len(sys.argv) != 3:
        print "Usage: python creat_mean_npy.py proto.mean out.npy"  此处必须要空格
        sys.exit()      此处也需要添加空格
blob = caffe.proto.caffe_pb2.BlobProto()
data = open(sys.argv[1], 'rb').read()
blob.ParseFromString(data)
arr = np.array(caffe.io.blobproto_to_array(blob))
out = arr[0]
np.save(sys.argv[2], out)
```

神经网络的trick

1.谈谈如何训练一个性能不错的深度神经网络：

http://blog.csdn.net/kuaitoukid/article/details/45821891

经典模型

1.AlexNet解读（hinton2012经典模型）：

https://github.com/BVLC/caffe/blob/master/models/bvlc_reference_caffenet/train_val.prototxt

2.Siamese network（两个CNN网络）：

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1640964

3.LeNet（mnist手写字数据库）：

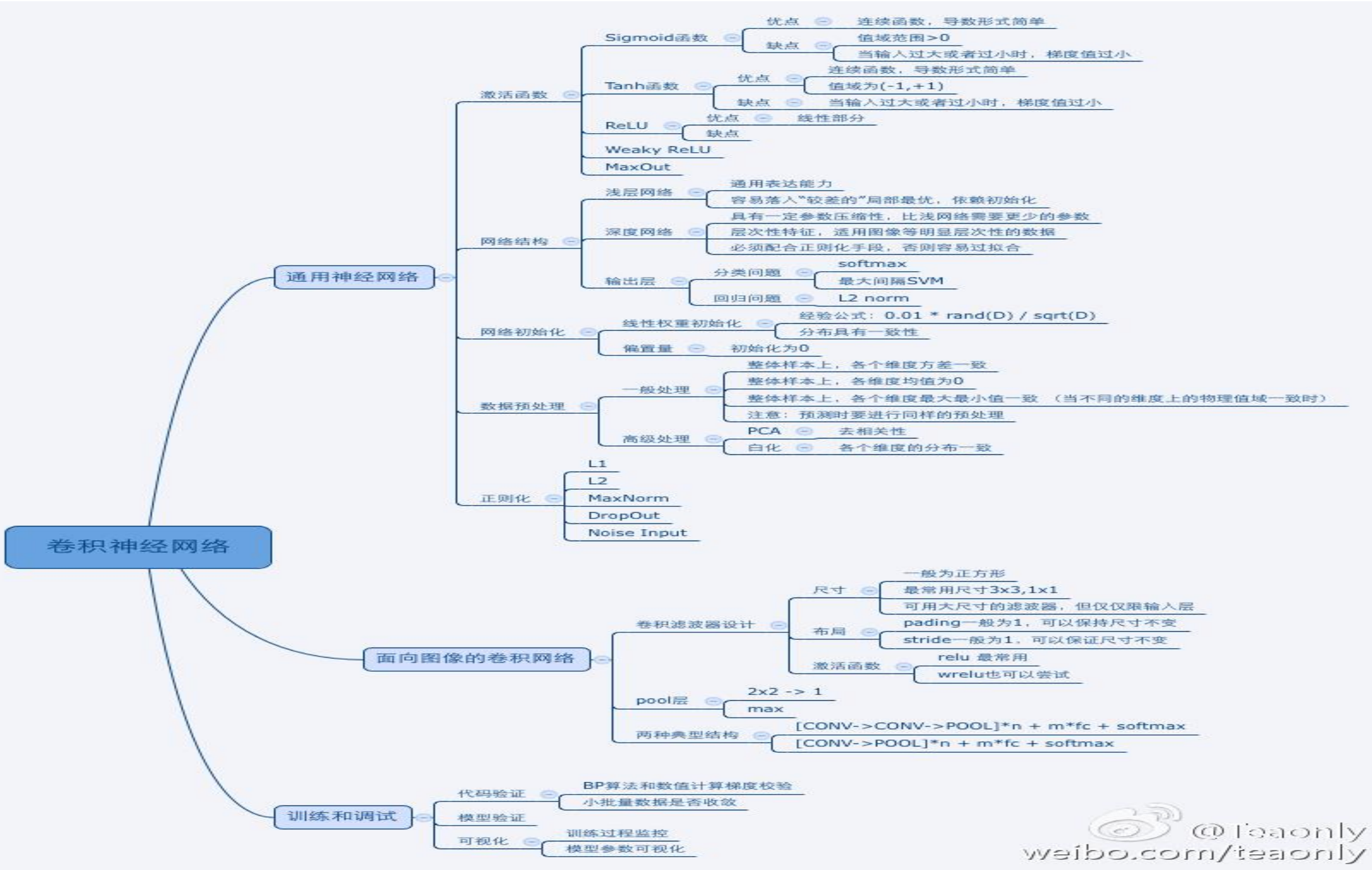
http://blog.csdn.net/qiaofangjie/article/details/16826849

4. GoogLeNet（极其复杂的模型）：
https://github.com/BVLC/caffe/blob/2f4a9e40af0c0d08c7b130f4e5091f39a1d33b5a/models/bvlc_googlenet/train_val.prototxt

更多新模型参考model zoo http://caffe.berkeleyvision.org/model_zoo.html

IDE

1.Nsight IDE


卷积神经网络
通用神经网络
激活函数
Sigmoid函数
优点
连续函数，导数形式简单
缺点
值域范围>0
当输入过大或者过小时，梯度值过小
Tanh函数
优点
连续函数，导数形式简单
值域为(-1,+1)
缺点
当输入过大或者过小时，梯度值过小
ReLU
优点
线性部分
缺点
Weaky ReLU
MaxOut
网络结构
浅层网络
通用表达能力
容易落入"较差的"局部最优，依赖初始化
深度网络
具有一定参数压缩性，比浅网络需要更少的参数
层次性特征，适用图像等明显层次性的数据
必须配合正则化手段，否则容易过拟合
输出层
分类问题
softmax
最大间隔SVM
回归问题
L2 norm
网络初始化
线性权重初始化
经验公式：0.01 * rand(D) / sqrt(D)
分布具有一致性
偏置量
初始化为0
数据预处理
一般处理
整体样本上，各个维度方差一致
整体样本上，各维度均值为0
整体样本上，各个维度最大最小值一致 （当不同的维度上的物理值域一致时）
注意：预测时要进行同样的预处理
高级处理
PCA
去相关性
白化
各个维度的分布一致
正则化
L1
L2
MaxNorm
DropOut
Noise Input
面向图像的卷积网络
卷积滤波器设计
尺寸
一般为正方形
最常用尺寸3x3,1x1
可用大尺寸的滤波器，但仅仅限输入层
布局
pading一般为1，可以保持尺寸不变
stride一般为1，可以保证尺寸不变
激活函数
relu 最常用
wrelu也可以尝试
pool层
2x2 -> 1
max
两种典型结构
[CONV->CONV->POOL]*n + m*fc + softmax
[CONV->POOL]*n + m*fc + softmax
训练和调试
代码验证
BP算法和数值计算梯度校验
小批量数据是否收敛
模型验证
可视化
训练过程监控
模型参数可视化
@teaonly
weibo.com/teaonly

其他事项

1.模型定义在.prototxt文件中，训练好的模型在model目录下.binaryproto格式的文件中。模型的格式由caffe.proto定义。采用Google Protocol Buffer可以节省空间还有它对C++和Pyhton的支持也很好。

2.参数定义的文件：cifar10_quick_train_test.prototxt;

参数设置的文件：cifar10_quick_solver.prototxt;

Multi模型定义

1. siamese network caffe里面有例子；不用改公式，只用把prototxt改下就行了，你看例子就知道了。你的loss变的话就需要写一个新的loss层；
2. Caffe里面的common layer里面有splitting层和concatentation层；

添加新的layer

1、属于哪个类型的layer，就打开哪个hpp文件，这里就打开vision_layers.hpp，然后自己添加该layer的定义，或者直接复制Convolution_Layer的相关代码来修改类名和构造函数名都改为Aaa_Layer，如果不用GPU，将*_gpu的声明都去掉。

2、实现自己的layer，**编写Aaa_Layer.cpp**，加入到src/caffe/layers，主要实现Setup、Forward_cpu、Backward_cpu。

3、如果**需要GPU实现，那么在Aaa_Layer.cu中实现**Forward_gpu和Backward_gpu。

4、修改src/caffe/proto/caffe.proto，找到LayerType，添加Aaa，并更新ID，如果Layer有参数，添加AaaParameter类。

5、在src/caffe/layer_factory.cpp中添加响应代码。

6、在src/caffe/test中写一个test_Aaa_layer.cpp，用include/caffe/test/test_gradient_check_util.hpp来检查前向后向传播是否正确。

具体可以参考Contrastive_loss_layer、inner_product_layer、maxout_layer层。

添加新的layer

1. Loss_layers.hpp中添加class ContrastiveLossLayer：public LossLayer<Dtype> {…};
2. src/caffe/layers中添加contrastive_loss_layer.cpp,并且自己写代码实现1中新定义类里面的函数；

```
namespace caffe{
template <typename Dtype>
void ContrastiveLossLayer<Dtype>::LayerSetUp(){…}
template <typename Dtype>
void ContrastiveLossLayer<Dtype>::Forward_cpu(){…}
…
#ifdef CPU_ONLY
STUB_GPU(ContrastiveLossLayer);
#endif
INSTANTIATE_CLASS(ContrastiveLossLayer);
REGISTER_LAYER_CLASS(ContrastiveLoss);
}
```

添加新的layer

3.

END!