Caffe源码解析4： Data_layer

转载请注明出处，楼燚(yì)航的blog，**http://home.cnblogs.com/louyihang-loves-baiyan/**

data_layer应该是网络的最底层，主要是将数据送给blob进入到net中，在data_layer中存在多个跟data_layer相关的类

- BaseDataLayer
- BasePrefetchingDataLayer
- DataLayer
- DummyDataLayer
- HDF5DataLayer
- HDF5OutputLayer
- ImageDataLayer
- MemoryDataLayer
- WindowDataLayer
- Batch

这里首先说明一下这几个类之间的区别。
首先Layer是基类，这个之前就已经提到过了。其次看HDF5相关的类有两个，一个是HDF5DataLayer，另一个是HDF5OutputLayer，主要是基于HDF5数据格式的读取和存储

留意到这个data_layer的头文件还include了不少头文件

```
#include <string>
#include <utility>
#include <vector>
#include "hdf5.h"

#include "caffe/blob.hpp"
#include "caffe/common.hpp"
#include "caffe/data_reader.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"
#include "caffe/util/blocking_queue.hpp"
#include "caffe/util/db.hpp"
```

hdf5就是之前说到的一种主要用于科学数据记录、能自我描述的数据格式。
还有几个跟data相关的头文件比如data_read.hpp,data_transformer.hpp
其中data_reader主要是负责数据的读取，传送到data layer中。并且对于每一个source，都会开一一起独立的reading thread读取线程，几十有多个solver在并行的跑。比如在多GPU训练的时候，可以保证对于数据库的读取是顺序的

data_transformer.hpp里面的DataTransformer这个类，这个类我们要关注一下，这个类主要能对input data执一些预处理操作，比如缩放、镜像、减去均值。同时还支持一些随机的操作。
其核心的函数如下，这里总共有5个常在的Transform函数，其中所有函数的第二部分是相同的，都是一个目标blob，而输入根据输入的情况可以有所选择，可以是blob,也可以是opencv的mat 结构，或者proto中定义的datum结构。

```
void Transform(const Datum& datum, Blob<Dtype>* transformed_blob);
```

```
void Transform(const vector<Datum> & datum_vector, Blob<Dtype>* transformed_blob);
void Transform(const vector<cv::Mat> & mat_vector, Blob<Dtype>* transformed_blob);
void Transform(const cv::Mat& cv_img, Blob<Dtype>* transformed_blob);
void Transform(Blob<Dtype>* input_blob, Blob<Dtype>* transformed_blob);
```

TransformationParameter是该类构造器中需要传入的一些变形参数，相关的操作定义在proto中，摘录如下,可以看到总共有sacle,mirror,crop_size,mean_file,mean_value,force_color,force_grey共7个相关操作

```
message TransformationParameter {
  optional float scale = 1 [default = 1];
  optional bool mirror = 2 [default = false];
  optional uint32 crop_size = 3 [default = 0];
  optional string mean_file = 4;
  repeated float mean_value = 5;
  optional bool force_color = 6 [default = false];
  optional bool force_gray = 7 [default = false];
}
```

首先对于dat_layer，里面根据继承关系最后的几个子类分别是ImageDataLayer,DataLayer,WindowDataLayer,MemoryDataLayer,HDF5以及Dummy这里暂时先不做分析。其实最重要的就是类面的layerSetup.首先我们来看DataLayer的DataLayerSetUp

```
void DataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  const int batch_size = this->layer_param_.data_param().batch_size();
  //获得相应的datum，用来初始化top blob
  Datum& datum = *(reader_.full().peek());
  //使用data_transformer 来计算根据datum的期望blob的shape
  vector<int> top_shape = this->data_transformer_->InferBlobShape(datum);
  this->transformed_data_.Reshape(top_shape);
  //首先reshape top[0]，再根据batch的大小进行预取
  top_shape[0] = batch_size;
  top[0]->Reshape(top_shape);
  for (int i = 0; i < this->PREFETCH_COUNT; ++i) {
    this->prefetch_[i].data_.Reshape(top_shape);
  }
  LOG(INFO) << "output data size: " << top[0]->num() << ","
      << top[0]->channels() << "," << top[0]->height() << ","
      << top[0]->width();
  // 同样reshape label的blob的shape
  if (this->output_labels_) {
    vector<int> label_shape(1, batch_size);
    top[1]->Reshape(label_shape);
    for (int i = 0; i < this->PREFETCH_COUNT; ++i) {
      this->prefetch_[i].label_.Reshape(label_shape);
    }
  }
}
```

MemoryDataLayer

```
void MemoryDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
     const vector<Blob<Dtype>*>& top) {
  //直接通过memory_data_param类设置layer的相关参数
  batch_size_ = this->layer_param_.memory_data_param().batch_size();
  channels_ = this->layer_param_.memory_data_param().channels();
  height_ = this->layer_param_.memory_data_param().height();
  width_ = this->layer_param_.memory_data_param().width();
  size_ = channels_ * height_ * width_;
  CHECK_GT(batch_size_ * size_, 0) <<
      "batch_size, channels, height, and width must be specified and"
      " positive in memory_data_param";
  //这里跟datalayer一样都是先设置top[0]，然后对label进行reshape
  vector<int> label_shape(1, batch_size_);
  top[0]->Reshape(batch_size_, channels_, height_, width_);
  top[1]->Reshape(label_shape);
  added_data_.Reshape(batch_size_, channels_, height_, width_);
  added_label_.Reshape(label_shape);
  data_ = NULL;
  labels_ = NULL;
  added_data_.cpu_data();
  added_label_.cpu_data();
}
```

ImageDataLayer,它的DataLayerSetUP函数

```
void ImageDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  const int new_height = this->layer_param_.image_data_param().new_height();
  const int new_width  = this->layer_param_.image_data_param().new_width();
```

--gaosi123

2. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

BTW, 我电脑是i7 4790k + 16GB内存，所以硬件设备应该不会是限制。不知道问题出在哪里

--gaosi123

3. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

如果可以，欢迎留个email

--gaosi123

4. Re:opencv 3.0 DPM Cascade 检测 （附带TBB和openMP加速）

你好，我也是直接把DPM代码拷贝到工程里，但是想你这样直接拷进去不会报错吗？我直接拷贝进去按照你的来，报错信息如下：Error 4 error C2039: 'dpm' : is not a memb.....

--gaosi123

5. Re:Fast RCNN 训练自己数据集 (2修改数据读取接口)

@楼燚航的blog楼主你好！我在EdgeBoxes提取OP的时候也是直接用的默认参数，并且将坐标[x y w h]变成了左上右下的形式，但是发现检测车的时候效果并没有Selective Search好，.....

–JustJay

```
  const bool is_color  = this->layer_param_.image_data_param().is_color();
  string root_folder = this->layer_param_.image_data_param().root_folder();

  CHECK((new_height == 0 && new_width == 0) ||
      (new_height > 0 && new_width > 0)) << "Current implementation requires "
      "new_height and new_width to be set at the same time.";
  //读取图像文件和相应的label
  const string& source = this->layer_param_.image_data_param().source();
  LOG(INFO) << "Opening file " << source;
  std::ifstream infile(source.c_str());
  string filename;
  int label;
  while (infile >> filename >> label) {
    lines_.push_back(std::make_pair(filename, label));
  }

  if (this->layer_param_.image_data_param().shuffle()) {
    // randomly shuffle data
    LOG(INFO) << "Shuffling data";
    const unsigned int prefetch_rng_seed = caffe_rng_rand();
    prefetch_rng_.reset(new Caffe::RNG(prefetch_rng_seed));
    ShuffleImages();
  }
  LOG(INFO) << "A total of " << lines_.size() << " images.";

  lines_id_ = 0;
  //check是否需要随机跳过一些图像
  if (this->layer_param_.image_data_param().rand_skip()) {
    unsigned int skip = caffe_rng_rand() %
        this->layer_param_.image_data_param().rand_skip();
    LOG(INFO) << "Skipping first " << skip << " data points.";
    CHECK_GT(lines_.size(), skip) << "Not enough points to skip";
    lines_id_ = skip;
  }
  //使用Opencv来读进图像，然后使用它初始化相应的top blob
  cv::Mat cv_img = ReadImageToCVMat(root_folder + lines_[lines_id_].first,
                                    new_height, new_width, is_color);
  CHECK(cv_img.data) << "Could not load " << lines_[lines_id_].first;
  //这里的步骤和上面相同，使用transformer来做reshape
  vector<int> top_shape = this->data_transformer_->InferBlobShape(cv_img);
  this->transformed_data_.Reshape(top_shape);
  //之后部分跟前面差不多，初始化top[0]
  const int batch_size = this->layer_param_.image_data_param().batch_size();
  CHECK_GT(batch_size, 0) << "Positive batch size required";
  top_shape[0] = batch_size;
  for (int i = 0; i < this->PREFETCH_COUNT; ++i) {
    this->prefetch_[i].data_.Reshape(top_shape);
  }
  top[0]->Reshape(top_shape);

  LOG(INFO) << "output data size: " << top[0]->num() << ","
      << top[0]->channels() << "," << top[0]->height() << ","
      << top[0]->width();
  //reshape label
  vector<int> label_shape(1, batch_size);
  top[1]->Reshape(label_shape);
  for (int i = 0; i < this->PREFETCH_COUNT; ++i) {
    this->prefetch_[i].label_.Reshape(label_shape);
  }
}
```

WindowDataLayer的DataLayerSetUp，这个函数标比较长，我只列出了其中主要的部分，之前的Image相当于是已经剪裁过的一个图像，也就是说你的目标基本上是充棉了整个画面，而Window File是用于原始图的，也就是说有background和object，这个window file 的格式如下

```
window_file format
repeated:
   # image_index
   img_path (abs path)
   channels
   height
   width
   num_windows
   class_index overlap x1 y1 x2 y2
```

```
//读取每一个box
int num_windows;
infile >> num_windows;
const float fg_threshold =
    this->layer_param_.window_data_param().fg_threshold();
```

```
const float bg_threshold =
    this->layer_param_.window_data_param().bg_threshold();
for (int i = 0; i < num_windows; ++i) {
  int label, x1, y1, x2, y2;
  float overlap;
  infile >> label >> overlap >> x1 >> y1 >> x2 >> y2;

  vector<float> window(WindowDataLayer::NUM);
  window[WindowDataLayer::IMAGE_INDEX] = image_index;
  window[WindowDataLayer::LABEL] = label;
  window[WindowDataLayer::OVERLAP] = overlap;
  window[WindowDataLayer::X1] = x1;
  window[WindowDataLayer::Y1] = y1;
  window[WindowDataLayer::X2] = x2;
  window[WindowDataLayer::Y2] = y2;

  // add window to foreground list or background list// read each box
int num_windows;
infile >> num_windows;
const float fg_threshold =
    this->layer_param_.window_data_param().fg_threshold();
const float bg_threshold =
    this->layer_param_.window_data_param().bg_threshold();
for (int i = 0; i < num_windows; ++i) {
  int label, x1, y1, x2, y2;
  float overlap;
  infile >> label >> overlap >> x1 >> y1 >> x2 >> y2;

  vector<float> window(WindowDataLayer::NUM);
  window[WindowDataLayer::IMAGE_INDEX] = image_index;
  window[WindowDataLayer::LABEL] = label;
  window[WindowDataLayer::OVERLAP] = overlap;
  window[WindowDataLayer::X1] = x1;
  window[WindowDataLayer::Y1] = y1;
  window[WindowDataLayer::X2] = x2;
  window[WindowDataLayer::Y2] = y2;

  //首先计算得到overlap,根据Overlap与fg_threshold的比较载添加到fg的list中
  if (overlap >= fg_threshold) {
    int label = window[WindowDataLayer::LABEL];
    CHECK_GT(label, 0);
    fg_windows_.push_back(window);
    label_hist.insert(std::make_pair(label, 0));
    label_hist[label]++;
  } else if (overlap < bg_threshold) {
    // background window, force label and overlap to 0
    window[WindowDataLayer::LABEL] = 0;
    window[WindowDataLayer::OVERLAP] = 0;
    bg_windows_.push_back(window);
    label_hist[0]++;
  }
}
=-
  if (overlap >= fg_threshold) {
    int label = window[WindowDataLayer::LABEL];
    CHECK_GT(label, 0);
    fg_windows_.push_back(window);
    label_hist.insert(std::make_pair(label, 0));
    label_hist[label]++;
  } else if (overlap < bg_threshold) {
    //background的label和overlap都是0
    window[WindowDataLayer::LABEL] = 0;
    window[WindowDataLayer::OVERLAP] = 0;
    bg_windows_.push_back(window);
    label_hist[0]++;
  }
}



.............
for (map<int, int>::iterator it = label_hist.begin();
      it != label_hist.end(); ++it) {
    LOG(INFO) << "class " << it->first << " has " << label_hist[it->first]
              << " samples";
  }

  LOG(INFO) << "Amount of context padding: "
      << this->layer_param_.window_data_param().context_pad();

  LOG(INFO) << "Crop mode: "
      << this->layer_param_.window_data_param().crop_mode();
```

```
//这里之后的步骤就差不多了，同样是对transform的一些操作
const int crop_size = this->transform_param_.crop_size();
CHECK_GT(crop_size, 0);
const int batch_size = this->layer_param_.window_data_param().batch_size();
top[0]->Reshape(batch_size, channels, crop_size, crop_size);
for (int i = 0; i < this->PREFETCH_COUNT; ++i)
  this->prefetch_[i].data_.Reshape(
      batch_size, channels, crop_size, crop_size);

LOG(INFO) << "output data size: " << top[0]->num() << ","
    << top[0]->channels() << "," << top[0]->height() << ","
    << top[0]->width();
// 对label进行reshape
vector<int> label_shape(1, batch_size);
top[1]->Reshape(label_shape);
for (int i = 0; i < this->PREFETCH_COUNT; ++i) {
  this->prefetch_[i].label_.Reshape(label_shape);
}

//做减均值的操作
has_mean_file_ = this->transform_param_.has_mean_file();
has_mean_values_ = this->transform_param_.mean_value_size() > 0;
if (has_mean_file_) {
  const string& mean_file =
        this->transform_param_.mean_file();
  LOG(INFO) << "Loading mean file from: " << mean_file;
  BlobProto blob_proto;
  ReadProtoFromBinaryFileOrDie(mean_file.c_str(), &blob_proto);
  data_mean_.FromProto(blob_proto);
}
if (has_mean_values_) {
  CHECK(has_mean_file_ == false) <<
    "Cannot specify mean_file and mean_value at the same time";
  for (int c = 0; c < this->transform_param_.mean_value_size(); ++c) {
    mean_values_.push_back(this->transform_param_.mean_value(c));
  }
  CHECK(mean_values_.size() == 1 || mean_values_.size() == channels) <<
   "Specify either 1 mean_value or as many as channels: " << channels;
  if (channels > 1 && mean_values_.size() == 1) {
    // Replicate the mean_value for simplicity
    for (int c = 1; c < channels; ++c) {
      mean_values_.push_back(mean_values_[0]);
    }
  }
}
```


楼燚航的blog
关注 - 1
粉丝 - 60
+加关注

« 上一篇：Caffe源码解析3：Layer
» 下一篇：Caffe源码解析5：Conv_Layer

posted @ 2016-01-23 13:10 楼燚航的blog 阅读(499) 评论(0) 编辑 收藏

最新IT新闻:
· LG确认：开发Friends模块设备需要取得授权并协同开发
· 触角越来越广 华为能成为中国的三星吗？
· 微软认知服务：人工智能的技术拼图
· 知己知彼，百战不殆：一篇文章看懂隐藏在阿尔法狗背后的深度学习
· 女性玩家崛起 研发女性游戏要注意什么
» 更多新闻...

最新知识库文章:
· 我是一个线程
· 为什么未来是全栈工程师的世界？
· 程序bug导致了天大的损失，要枪毙程序猿吗？
· 如何运维千台以上游戏云服务器
· 架构漫谈（一）：什么是架构？
» 更多知识库文章...