# Caffe中自定义数据层

## 自定义数据层基本步骤

自定义数据层主要有几个步骤：

1. 在caffe.proto中注册新类，添加变量
2. 自定义hpp头文件和cpp源文件，重写type，LayerSetUp，ShuffleImage，load\_batch方法；
3. 重新编译caffe

下面以读取一个包含50×100幅20×20的手写数字图像的大图像为例，说明自定义数据层的实现。

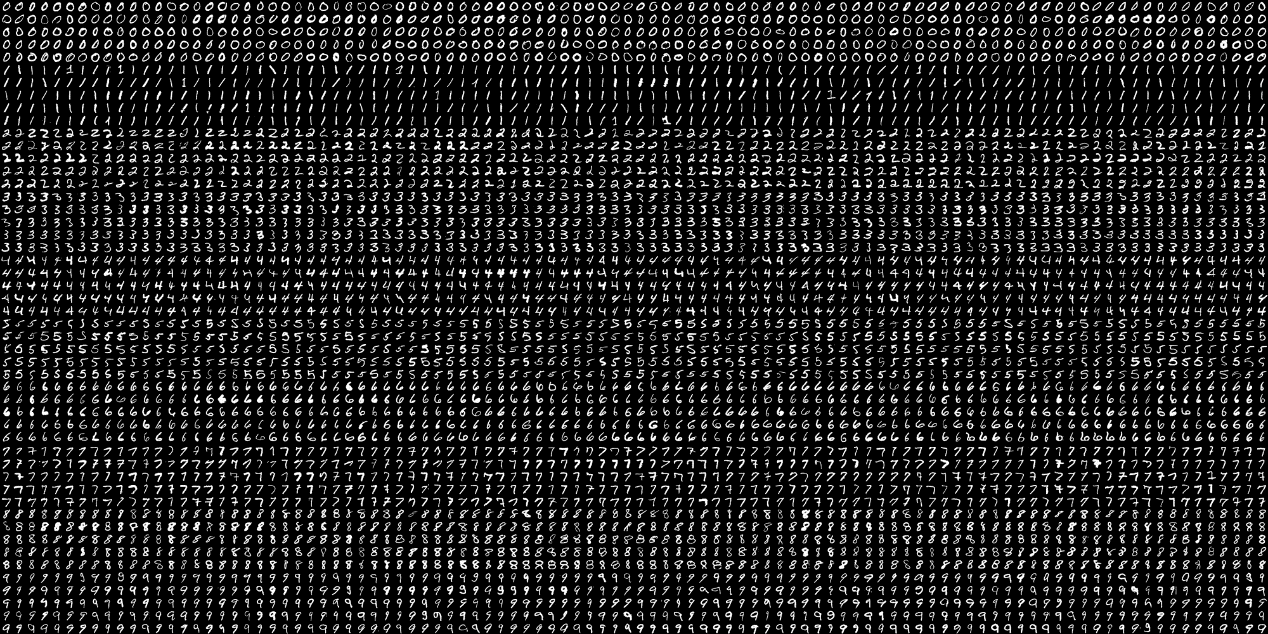


图 1 测试图像

图像包括10个数字的手写图像，每一个数字有5行，100列，选择前70列作为训练，后30列作为测试。

## 在caffe.proto中注册新类

打开/src/caffe/proto/caffe.proto文件，定义层参数和相关参数：

意义如下：

cols：图像行数，这里为100

rows：图像列数，这里为50

width：每幅小图像宽度，这里为20

height：每幅小图像高度，这里为20

start\_col：输入图像起始行数，训练时为0，测试时为70

end\_col：输入图像截止行数，训练时为70，测试时为100

image\_address：输入图像的地址

is\_save = 8：是否保存子图像，便于查看图像是否分割正确

save\_folder：保存路径

is\_color：是否是彩色图像

batch\_size：批大小

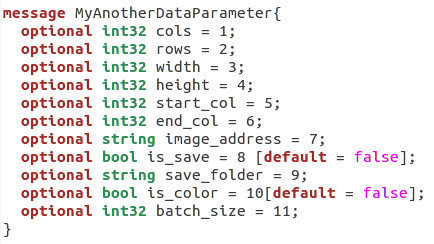


图 2 自定义层参数

在LayerParameter中添加自定义的层参数对象：



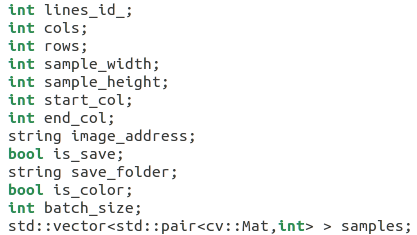
图 3 添加层参数对象

## 自定义头文件和源文件并重写相关方法

### 头文件

在include/caffe/layers目录中复制一份image\_data\_layer.hpp文件，改名成my\_another\_data\_layer.hpp，将原ImageDataLayer改成MyAnotherDataLayer，并修改其它必要的代码。

对2中定义的参数，在头文件中添加相应的变量，用于保存这些数据，并定义一个pair类型的向量，用于保存图像和对应的标签：



修改type函数，返回类别名称，此函数仅用于信息的打印：



修改后头文件如下：

#ifndef CAFFE\_MY\_ANOTHER\_DATA\_LAYER\_HPP\_

#define CAFFE\_MY\_ANOTHER\_DATA\_LAYER\_HPP\_

#include <string>

#include <utility>

#include <vector>

#include "caffe/blob.hpp"

#include "caffe/data\_transformer.hpp"

#include "caffe/internal\_thread.hpp"

#include "caffe/layer.hpp"

#include "caffe/layers/base\_data\_layer.hpp"

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

namespace caffe {

/\*\*

\* @brief Provides data to the Net from image files.

\*

\* TODO(dox): thorough documentation for Forward and proto params.

\*/

template <typename Dtype>

class MyAnotherDataLayer : public BasePrefetchingDataLayer<Dtype> {

public:

explicit MyAnotherDataLayer(const LayerParameter& param)

: BasePrefetchingDataLayer<Dtype>(param) {}

virtual ~MyAnotherDataLayer();

virtual void DataLayerSetUp(const vector<Blob<Dtype>\*>& bottom,

const vector<Blob<Dtype>\*>& top);

virtual inline const char\* type() const { return "MyAnotherData"; }

virtual inline int ExactNumBottomBlobs() const { return 0; }

virtual inline int ExactNumTopBlobs() const { return 2; }

cv::Mat getSample(cv::Mat image, int x, int y, int w, int h);

protected:

virtual void ShuffleImages();

virtual void load\_batch(Batch<Dtype>\* batch);

int lines\_id\_;

int cols;

int rows;

int sample\_width;

int sample\_height;

int start\_col;

int end\_col;

string image\_address;

bool is\_save;

string save\_folder;

bool is\_color;

int batch\_size;

std::vector<std::pair<cv::Mat,int> > samples;

};

} // namespace caffe

#endif // CAFFE\_MY\_ANOTHER\_DATA\_LAYER\_HPP\_

### 源文件

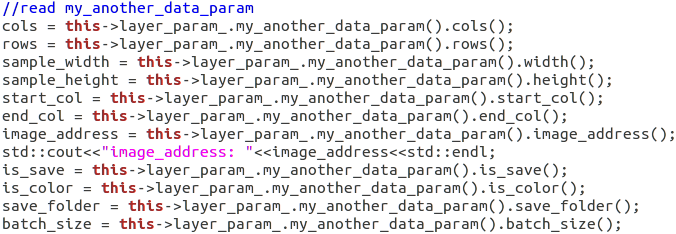
在src/caffe/layers目录中复制一份image\_data\_layer.cpp文件，改名成my\_another\_data\_layer.cpp，将原ImageDataLayer改成MyAnotherDataLayer，并修改其它必要的代码。

改写DataLayerSetUp，load\_batch函数。

#### DataLayerSetUp函数

重写DataLayerSetUp函数，改函数主要完成以下功能：读取层参数，读取图像和对应标签，重排图像，初始化transformed\_data，prefetch\_等形状的初始化。与ImageDataLayer相比，我们只是改变了图像的输入方式，因此我们只需要重写读取参数，读取图像，重排图像等相关部分即可。

1. 读取参数

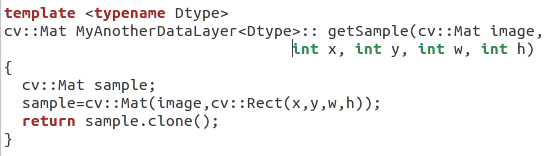


1. 读取图像

图像保存在std::vector<std::pair<cv::Mat,int>> samples中，注意，“>>”中间需要加一个空格。

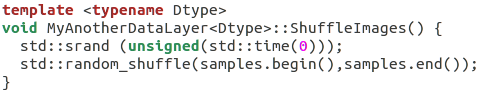


其中getSample的函数定义如下：

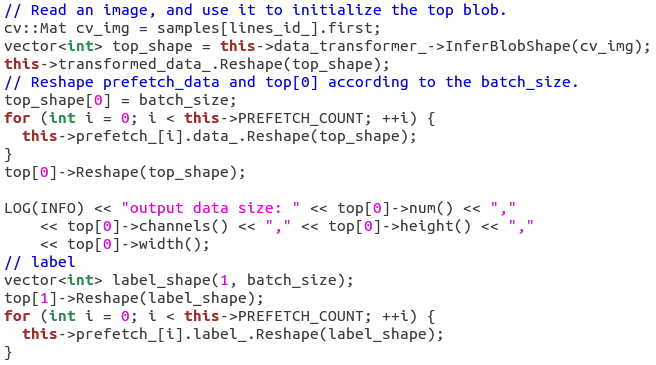


1. 重排图像

重排图像调用ShuffleImages()函数，其定义如下：



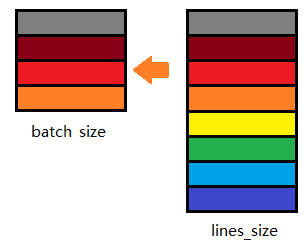
1. 其它



#### load\_batch函数

改函数将读取的图像转换为Blob类型，保存到cpu数据中。

batch包括图像数据和标签两个成员。在load\_batch函数中，将读取到的图像和对应标签传入到cpu中，每一批的数量是batch\_size，如果读取完了所有图像，再将图像打乱，从头开始，继续读取。

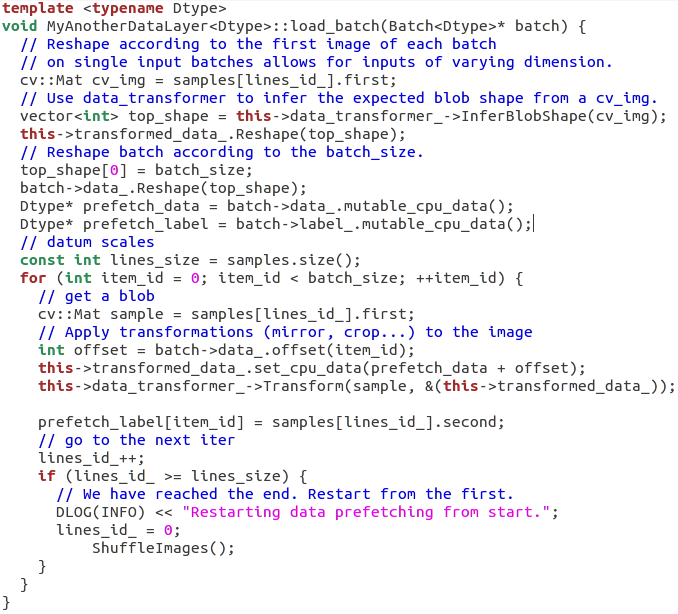


将图像传入到cpu中是以下代码实现的：



其中Transform函数将samle图像减去均值图像，然后剪切、镜向，保存到transformed\_data\_这个Blob类型变量中。

整个函数代码如下：



#### 其它

在源文件最后，修改以下宏：



这里的把MyAnotherDataLayer注册一个名字MyAnotherData，这在网络文件中使用该层直接使用该名字即可。

修改后，整个源文件代码如下：

#ifdef USE\_OPENCV

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <fstream> // NOLINT(readability/streams)

#include <iostream> // NOLINT(readability/streams)

#include <string>

#include <utility>

#include <vector>

#include "caffe/data\_transformer.hpp"

#include "caffe/layers/base\_data\_layer.hpp"

#include "caffe/layers/my\_another\_data\_layer.hpp"

#include "caffe/util/benchmark.hpp"

#include "caffe/util/io.hpp"

#include "caffe/util/math\_functions.hpp"

#include "caffe/util/rng.hpp"

namespace caffe {

template <typename Dtype>

MyAnotherDataLayer<Dtype>::~MyAnotherDataLayer<Dtype>() {

this->StopInternalThread();

}

template <typename Dtype>

void MyAnotherDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>\*>& bottom,const vector<Blob<Dtype>\*>& top) {

//read my\_another\_data\_param

cols = this->layer\_param\_.my\_another\_data\_param().cols();

rows = this->layer\_param\_.my\_another\_data\_param().rows();

sample\_width = this->layer\_param\_.my\_another\_data\_param().width();

sample\_height = this->layer\_param\_.my\_another\_data\_param().height();

start\_col = this->layer\_param\_.my\_another\_data\_param().start\_col();

end\_col = this->layer\_param\_.my\_another\_data\_param().end\_col();

image\_address = this->layer\_param\_.my\_another\_data\_param().image\_address();

std::cout<<"image\_address: "<<image\_address<<std::endl;

is\_save = this->layer\_param\_.my\_another\_data\_param().is\_save();

is\_color = this->layer\_param\_.my\_another\_data\_param().is\_color();

save\_folder = this->layer\_param\_.my\_another\_data\_param().save\_folder();

batch\_size = this->layer\_param\_.my\_another\_data\_param().batch\_size();

lines\_id\_ = 0;

//load image

cv::Mat image;

if(is\_color)

{

image=cv::imread(image\_address,1);

}

else

{

image=cv::imread(image\_address,0);

}

//get the sample and label

samples.clear();

for(int i=0;i<rows;i++)

{

for(int j=start\_col;j<end\_col;j++)

{

cv::Mat sample=getSample(image,j\*sample\_height,

i\*sample\_width,sample\_width,sample\_height);

int label=i/5;

samples.push\_back(std::make\_pair<cv::Mat,int>(sample,label));

}

}

//shuffe images

ShuffleImages();

//save images

if(is\_save)

{

for(int i=0;i<samples.size();i++)

{

char image\_name[100];

sprintf(image\_name,"%s/%04d.jpg",save\_folder.c\_str(),i);

cv::imwrite(image\_name,samples[i].first);

// std::cout<<"save image:"<<image\_name<<std::endl;

}

}

lines\_id\_ = 0;

// Read an image, and use it to initialize the top blob.

cv::Mat cv\_img = samples[lines\_id\_].first;

vector<int> top\_shape = this->data\_transformer\_->InferBlobShape(cv\_img);

this->transformed\_data\_.Reshape(top\_shape);

// Reshape prefetch\_data and top[0] according to the batch\_size.

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();

// 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);

}

}

template <typename Dtype>

cv::Mat MyAnotherDataLayer<Dtype>:: getSample(cv::Mat image,

int x, int y, int w, int h)

{

cv::Mat sample;

sample=cv::Mat(image,cv::Rect(x,y,w,h));

return sample.clone();

}

template <typename Dtype>

void MyAnotherDataLayer<Dtype>::ShuffleImages() {

std::srand (unsigned(std::time(0)));

std::random\_shuffle(samples.begin(),samples.end());

}

// This function is called on prefetch thread

template <typename Dtype>

void MyAnotherDataLayer<Dtype>::load\_batch(Batch<Dtype>\* batch) {

// Reshape according to the first image of each batch

// on single input batches allows for inputs of varying dimension.

cv::Mat cv\_img = samples[lines\_id\_].first;

// Use data\_transformer to infer the expected blob shape from a cv\_img.

vector<int> top\_shape = this->data\_transformer\_->InferBlobShape(cv\_img);

this->transformed\_data\_.Reshape(top\_shape);

// Reshape batch according to the batch\_size.

top\_shape[0] = batch\_size;

batch->data\_.Reshape(top\_shape);

Dtype\* prefetch\_data = batch->data\_.mutable\_cpu\_data();

Dtype\* prefetch\_label = batch->label\_.mutable\_cpu\_data();

// datum scales

const int lines\_size = samples.size();

for (int item\_id = 0; item\_id < batch\_size; ++item\_id) {

// get a blob

cv::Mat sample = samples[lines\_id\_].first;

// Apply transformations (mirror, crop...) to the image

int offset = batch->data\_.offset(item\_id);

this->transformed\_data\_.set\_cpu\_data(prefetch\_data + offset);

this->data\_transformer\_->Transform(sample, &(this->transformed\_data\_));

prefetch\_label[item\_id] = samples[lines\_id\_].second;

// go to the next iter

lines\_id\_++;

if (lines\_id\_ >= lines\_size) {

// We have reached the end. Restart from the first.

DLOG(INFO) << "Restarting data prefetching from start.";

lines\_id\_ = 0;

ShuffleImages();

}

}

}

INSTANTIATE\_CLASS(MyAnotherDataLayer);

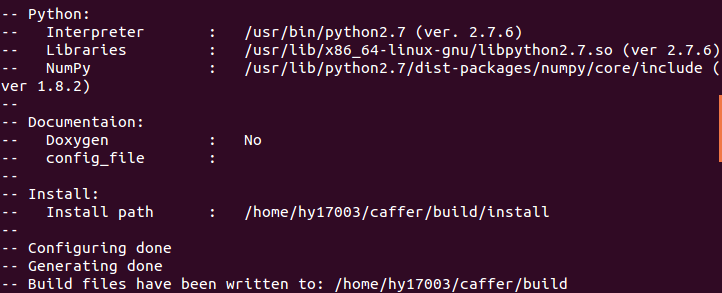
REGISTER\_LAYER\_CLASS(MyAnotherData);

} // namespace caffe

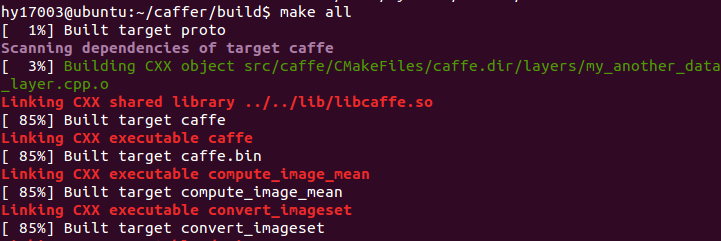
#endif // USE\_OPENCV

## 重新编译caffe

hy17003@ubuntu:~/caffer/build$ cmake ..



hy17003@ubuntu:~/caffer/build$ make all



## 测试

在examples/minst中创建一个文件夹my\_another\_data，并在其中创建三个文件：

网络设计文件：my\_another\_data\_lenet\_train\_test.prototxt

超参数文件：my\_another\_data\_solver.prototxt

训练脚本文件：my\_another\_data\_train.sh

### 网络设计文件

name: "LeNet"

layer {

name: "mnist"

type: "MyAnotherData"

top: "data"

top: "label"

include {

phase: TRAIN

}

transform\_param {

scale: 0.00390625

}

my\_another\_data\_param {

cols:100

rows:50

width:20

height:20

start\_col:0

end\_col:70

image\_address:"/home/hy17003/caffer/examples/images/digits.png"

is\_save:true

save\_folder:"/home/hy17003/caffer/examples/images/img"

is\_color:false

batch\_size:11

}

}

layer {

name: "mnist"

type: "MyAnotherData"

top: "data"

top: "label"

include {

phase: TEST

}

transform\_param {

scale: 0.00390625

}

my\_another\_data\_param {

cols:100

rows:50

width:20

height:20

start\_col:70

end\_col:100

image\_address:"/home/hy17003/caffer/examples/images/digits.png"

is\_save:false

is\_color:false

batch\_size:11

}

}

layer {

name: "conv1"

type: "Convolution"

bottom: "data"

top: "conv1"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

convolution\_param {

num\_output: 20

kernel\_size: 5

stride: 1

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

}

}

}

layer {

name: "pool1"

type: "Pooling"

bottom: "conv1"

top: "pool1"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

name: "conv2"

type: "Convolution"

bottom: "pool1"

top: "conv2"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

convolution\_param {

num\_output: 50

kernel\_size: 5

stride: 1

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

}

}

}

layer {

name: "pool2"

type: "Pooling"

bottom: "conv2"

top: "pool2"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

name: "ip1"

type: "InnerProduct"

bottom: "pool2"

top: "ip1"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

inner\_product\_param {

num\_output: 500

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

}

}

}

layer {

name: "relu1"

type: "ReLU"

bottom: "ip1"

top: "ip1"

}

layer {

name: "ip2"

type: "InnerProduct"

bottom: "ip1"

top: "ip2"

param {

lr\_mult: 1

}

param {

lr\_mult: 2

}

inner\_product\_param {

num\_output: 10

weight\_filler {

type: "xavier"

}

bias\_filler {

type: "constant"

}

}

}

layer {

name: "accuracy"

type: "Accuracy"

bottom: "ip2"

bottom: "label"

top: "accuracy"

include {

phase: TEST

}

}

layer {

name: "loss"

type: "SoftmaxWithLoss"

bottom: "ip2"

bottom: "label"

top: "loss"

}

### 超参数文件

# The train/test net protocol buffer definition

net: "/home/hy17003/caffer/examples/mnist/my\_another\_data/my\_another\_data\_lenet\_train\_test.prototxt"

# test\_iter specifies how many forward passes the test should carry out.

# In the case of MNIST, we have test batch size 100 and 100 test iterations,

# covering the full 10,000 testing images.

test\_iter: 50

# Carry out testing every 500 training iterations.

test\_interval: 200

# The base learning rate, momentum and the weight decay of the network.

base\_lr: 0.01

momentum: 0.9

weight\_decay: 0.0005

# The learning rate policy

lr\_policy: "inv"

gamma: 0.0001

power: 0.75

# Display every 100 iterations

display: 200

# The maximum number of iterations

max\_iter: 1000

# snapshot intermediate results

snapshot: 500

snapshot\_prefix: "./"

# solver mode: CPU or GPU

solver\_mode: CPU

### 训练脚本文件

#!/usr/bin/env sh

set -e

./build/tools/caffe train \

--solver=examples/mnist/my\_another\_data/my\_another\_data\_solver.prototxt $@

### 训练和测试

hy17003@ubuntu:~/caffer$ ./examples/mnist/my\_another\_data/my\_another\_data\_train.sh

