机器学习 深度学习（Deep Learning） 修改

深度学习caffe的代码怎么读？ 修改

Caffe的代码结构是什么样的？从代码结构上来看，Caffe在实现Deep Learning上有什么优点和缺点？ 修改

添加评论 分享 • 邀请回答 举报

19 个回答 默认排序

525

Gein Chen，PhD_1/HKU

雨宫夏一、 野原圆长、 切克闹 等 525 人赞同

楼上的大神回答的都很好了，非常感谢。这里我想说一下我自己学习caffe的方式，限于时间篇幅，并不想深入到具体的实现细节，只是从大的方向上谈谈，因为讲清楚一个细节都是一篇博客的篇幅了。

1．学习程序的第一步，先让程序跑起来，看看结果，这样就会有直观的感受。
Caffe的官网上Caffe | Deep Learning Framework 提供了很多的examples,你可以很容易地开始训练一些已有的经典模型，如LeNet。我建议先从 LeNet MNIST Tutorial 开始，因为数据集很小，网络也很小但很经典，用很少的时间就可以跑起来了。当你看到terminal刷拉拉的一行行输出，看到不断减少的loss和不断上升的accuracy，训练结束你得到了99+%的准确率，感觉好厉害的样子。你可以多跑跑几个例子，熟悉一下环境和接口。

2．单步调试，跟着Caffe在网络里流动
当玩了几天之后，你对Caffe的接口有点熟悉了，对已有的例子也玩腻了，你开始想看看具体是怎么实现的了。我觉得最好的方法是通过单步调试的方式跟着程序一步一步的在网络里前向传播，然后再被当成误差信息传回来。

Caffe就像一个你平常编程中Project,你可以使用IDE或者GDB去调试它，这里我们不细说调试的过程。你可以先跟踪前向传播的过程，无非就是从高层次到低层次的调用Forward函数， Solver->Net->Layer->Specific Layer (Convolution等...).后向传播也类似，但因为你对Caffe里面的各种变量运算不熟悉，当你跟踪完前向传播时可能已经头晕眼花了，还是休息一下，消化一下整个前向传播的流程。

刚刚开始你没有必要对每个Layer的计算细节都那么较真，大概知道程序的运算流程就好，这样你才可以比较快的对Caffe有个大体的把握。

3．个性化定制Caffe
到这里，你已经可以说自己有用过Caffe了，但是还不能算入门，因为你还不知道怎么修改源码，满足自己特定的需求。我们很多时候都需要自己定义新的层来完成特定的运算，这时你需要在Caffe里添加新的层。

你一开肯定无从下手，脑子一片空白。幸运的是Caffe github上的Wiki Development · BVLC/caffe Wiki · GitHub 已经有了教程了，而且这是最接近latest Caffe的源码结构的教程，你在网上搜到的Blog很多是有点过时的，因为Caffe最近又重构了代码。你可以跟着它的指导去添加自己的层。

虽然你已经知道要在哪里添加自己的东西了，但你遇到最核心的问题是如何写下面这四个函数。

- forward_cpu()

- forward_gpu()
- backward_cpu()
- backward_gpu()

你可以先模仿已有的层去实现这四个函数，而且我相信forward函数很快就可以写出来了，但backward的还是一头雾水。这时我们就要补补神经网络里最核心的内容了——**Backpropagation.**

４．理解并实现Backpropagation
这个我觉得是与平台无关的，不管你是使用Caffe、Torch 7,还是Theano,你都需要深刻理解并掌握的。因为我比较笨，花了好长时间才能够适应推导中的各种符号。其实也不难，就是误差顺着Chain rule法则流回到前面的层。我不打算自己推导后向传播的过程，因为我知道我没有办法将它表达得很好，而且网上已经有很多非常好的教程了。下面是我觉得比较好的学习步骤吧。

- 从浅层的神经网络（所谓的全连接层）的后向传播开始，因为这个比较简单，而且现在我们常说的CNN和LSTM的梯度计算也最终会回归到这里。
    - 第一个必看的是Ng深入浅出的Ufldl教程UFLDL Tutorial ，还有中文版的，这对不喜欢看英语的同学是个好消息。当然你看一遍不理解，再看一遍，忘了，再看，读个几遍你才会对推导过程和数学符号熟悉。我头脑不大行，来来回回看了好多次。
    - 当然，Ufldl的教程有点短，我还发现了一个讲得更细腻清晰的教程, Michael Nielsen 写的Neural networks and deep learning 。它讲得实在太好了，以至于把我的任督二脉打通了。在Ufldl的基础上读这个，你应该可以很快掌握全连接层的反向传播。
    - 最后在拿出standford大牛karpathy的一篇博客Hacker's guide to Neural Networks ，这里用了具体的编程例子手把手教你算梯度，并不是推导后向传播公式的，是关于通用梯度计算的。用心去体会一下。
- 这时你跃跃欲试，回去查看Caffe源码里Convolution层的实现，但发现自己好像没看懂。虽说卷积层和全连接层的推导大同小异，但思维上还是有个gap的。我建议你先去看看Caffe如何实现卷积的，Caffe作者贾扬清大牛在知乎上的回答在 Caffe 中如何计算卷积？让我茅塞顿开。重点理解im2col和col2im.
- 这时你知道了Convolution的前向传播，还差一点就可以弄明白后向传播怎么实现了。我建议你死磕Caffe中Convolution层的计算过程，把每一步都搞清楚，经过痛苦的过程之后你会对反向传播有了新的体会的。在这之后，你应该有能力添加自己的层了。再补充一个完整的添加新的层的教程Making a Caffe Layer • Computer Vision Enthusiast 。这篇教程从头开始实现了一个Angle To Sine Cosine Layer，包含了梯度推导，前向与后向传播的CPU和GPU函数，非常棒的一个教程。
- 最后，建议学习一下基本的GPU Cuda编程，虽然Caffe中已经把Cuda函数封装起来了，用起来很方便，但有时还是需要使用kernel函数等Cuda接口的函数。这里有一个入门的视频教程，讲得挺不错的NVIDIA CUDA初级教程视频 。

第二个回答，讲得还是一样的乱，谢谢观看。

北川谦一，Caffe / Torch7 / CNN

我是从Torch7转移到Caffe的人，仅供参考，当你阅读前你应该具备一些有关DL的基础知识，本文集中写Caffe代码结构而非介绍DL知识。
我是去年底开始看Caffe代码的，看代码的时间加在一起也不到一个月，也算半个新手，我的回答是从新手角度作一个入门阶段的经验分享。

本文只涉及Caffe结构的相关问题，不涉及具体实现技巧等细节

本文只涉及Caffe结构的相关问题，不涉及具体实现技巧等细节。

==========

1. 初识**Caffe**

1.1. Caffe相对与其他DL框架的优点和缺点：

优点：

- 速度快。Google Protocol Buffer数据标准为Caffe提升了效率。
- 学术论文采用此模型较多。不确定是不是最多，但接触到的不少论文都与Caffe有关（R-CNN，DSN，最近还有人用Caffe实现LSTM）

缺点：

- 曾更新过重要函数接口。有人反映，偶尔会出现接口变换的情况，自己很久前写的代码可能过了一段时间就不能和新版本很好地兼容了。（现在更新速度放缓，接口逐步趋于稳定，感谢 评论区王峰的建议）
- 对于某些研究方向来说的人并不适合。这个需要对Caffe的结构有一定了解，（后面提到）。

1.2. Caffe代码层次。

回答里面有人说熟悉Blob，Layer，Net，Solver这样的几大类，我比较赞同。我基本是从这个顺序开始学习的，这四个类复杂性从低到高，贯穿了整个Caffe。把他们分为三个层次介绍。

- **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。
- **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。
- **Net：**是网络的搭建，将Layer所派生出层类组合成网络。**Solver：**是Net的求解，修改这部分人主要会是研究DL求解方向的。

==========

2. Caffe进阶

2.1. Blob：

Caffe支持CUDA，在数据级别上也做了一些优化，这部分最重要的是知道它主要是对protocol buffer所定义的数据结构的继承，Caffe也因此可以在尽可能小的内存占用下获得很高的效率。（追求性能的同时Caffe也牺牲了一些代码可读性）

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_;
```

这里使用的是一个Blob的容器是因为某些Layer包含多组学习参数，比如多个卷积核的卷积层。

以及Layer所传递的数据形式，后面还会涉及到这里：

```
vector<Blob<Dtype>*> &bottom;
vector<Blob<Dtype>*> *top
```

2.2. Layer：

2.2.1. 5大**Layer**派生类型

Caffe十分强调网络的层次性，也就是说卷积操作，非线性变换（ReLU等），Pooling，权值连接等全部都由某一种Layer来表示。具体来说分为5大类Layer

- **NeuronLayer**类 定义于neuron_layers.hpp中，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）。

- **LossLayer**类 定义于loss_layers.hpp中，其派生类会产生loss，只有这些层能够产生loss

- LossLayer类定义了loss_layers.hpp中，其派生类会产生loss，只有这些层能够产生loss。
- 数据层 定义于data_layer.hpp中，作为网络的最底层，主要实现数据格式的转换。
- 特征表达层（我自己分的类）定义于vision_layers.hpp（为什么叫vision这个名字，我目前还不清楚），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）。
- 网络连接层和激活函数（我自己分的类）定义于common_layers.hpp，Caffe提供了单个层与多个层的连接，并在这个头文件中声明。这里还包括了常用的全连接层InnerProductLayer类。

2.2.2. Layer的重要成员函数

在Layer内部，数据主要有两种传递方式，正向传导（Forward）和反向传导（Backward）。Forward和Backward有CPU和GPU（部分有）两种实现。Caffe中所有的Layer都要用这两种方法传递数据。

```
virtual void Forward(const vector<Blob<Dtype>*> &bottom,
                     vector<Blob<Dtype>*> *top) = 0;
virtual void Backward(const vector<Blob<Dtype>*> &top,
                      const vector<bool> &propagate_down,
                      vector<Blob<Dtype>*> *bottom) = 0;
```

Layer类派生出来的层类通过这实现这两个虚函数，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）。注意这里为什么用了一个包含Blob的容器（vector），对于大多数Layer来说输入和输出都各连接只有一个Layer，然而对于某些Layer存在一对多的情况，比如LossLayer和某些连接层。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。

```
layers {
  bottom: "decode1neuron"    // 该层底下连接的第一个Layer
  bottom: "flatdata"         // 该层底下连接的第二个Layer
  top: "l2_error"            // 该层顶上连接的一个Layer
  name: "loss"               // 该层的名字
  type: EUCLIDEAN_LOSS       // 该层的类型
  loss_weight: 0
}
```

2.2.3. Layer的重要成员变量

loss

```
vector<Dtype> loss_;
```

每一层又有一个loss_值，只不多大多数Layer都是0，只有LossLayer才可能产生非0的loss_。计算loss是会把所有层的loss_相加。

learnable parameters

```
vector<shared_ptr<Blob<Dtype> > > blobs_;
```

前面提到过的，Layer学习到的参数。

2.3. Net:

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_;
```

同样Net也有它自己的

```
vector<Blob<Dtype>*>& Forward(const vector<Blob<Dtype>* > & bottom,
                              Dtype* loss = NULL);
void Net<Dtype>::Backward();
```

他们是对整个网络的前向和方向传导，各调用一次就可以计算出网络的loss了。

2.4. Solver

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

```
virtual void ComputeUpdateValue() = 0;
```

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。

```
ComputeUpdateValue();
net_->Update();
```

==========

至此，从底层到顶层对Caffe的主要结构都应该有了大致的概念。为了集中重点介绍Caffe的代码结构，文中略去了大量Caffe相关的实现细节和技巧，比如Layer和Net的参数如何初始化，proto文件的定义，基于cblas的卷积等操作的实现（cblas实现卷积这一点我的个人主页GanYuFei 中的《Caffe学习笔记5-BLAS与boost::thread加速》有介绍）等等就不一一列举了。

整体来看Layer部分代码最多，也反映出Caffe比较重视丰富网络单元的类型，然而由于Caffe的代码结构高度层次化，使得某些研究以及应用（比如研究类似非逐层连接的神经网络这种复杂的网络连接方式）难以在该平台实现。这也就是一开始说的一个不足。

另外，Caffe基本数据单元都用Blob，使得数据在内存中的存储变得十分高效，紧凑，从而提升了整体训练能力，而同时带来的问题是我们看见的一些可读性上的不便，比如forward的参数也是直接用Blob而不是设计一个新类以增强可读性。所以说性能的提升是以可读性为代价的。
最后一点也是最重要的一点，我从Caffe学到了很多。第一次看的C++项目就看到这么好的代码，实在是受益匪浅，在这里也感谢作者贾扬清等人的贡献。

甘宇飞更新于**2/13/2015**

米八，人脸检测/关键点定位/识别

其实关于怎么用caffe，我一直想写篇文章，因为给太多人讲过怎么用了，也帮很多人基于caffe写过代码。14年初因为赶NIPS，开始用caffe，大概用了有一年半了。

先说个大概，知道了神经网络的结构，forward跟backward是怎样的一个过程，基本上就知道了caffe的结构了。按照神经网络的运行过程看**Caffe**就好了。

既然说神经网络，那么就得有一个网络出来，caffe里面就用Net这个类来记录这个网络。

那么网络是由很多个layer构成，自然而然就会有Layer这个类，为了统一管理这些类，自然而然就想要抽象，那么Layer这个抽象类作为一个最基本的单元出现了，接下来就会有实现各种功能的layer出现，如：Convolution/ReLU/Softmax等。

Layer间需要连接啊，Layer需要有输入输出啊，caffe里面用Bottom来表示输入，Top表示输出，前一层Layer的top是

后一层layer的bottom，这样，连接搞掂了，输入输出也搞掂了。

网络搞掂了，layer搞掂了，那就要搞个求解算法啊，那么Solver这个类就出现了，这个就是用来求解网络的。

就酱紫，有时间再写得详细一点。反正就这么一句话：按照神经网络的运行过程看**Caffe**就好了。

发布于 2015-07-16　12 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

24

Vinjn张静，微信订阅号：黑客与画家

24 人赞同

深度学习源码解读-ch0-talk is cheap - 黑客与画家 - 知乎专栏
深度学习源码解读-ch1-JSON is awesome - 黑客与画家 - 知乎专栏
深度学习源码解读-ch2-Caffe is coming - 黑客与画家 - 知乎专栏
深度学习源码解读-ch3-Caffe is brewing - 黑客与画家 - 知乎专栏
深度学习源码解读-ch4-Caffe 中的设计模式 - 黑客与画家 - 知乎专栏

编辑于 2015-12-29　1 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

3

r0ck3r，Mining Structural Data with RNN/CNN

3 人赞同

一点tip，多翻caffe的issue看，开发者在里面对caffe的框架设计、演化讨论很详细

发布于 2015-09-12　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

1

肖良，计算机视觉, 机器学习, 无人车

1 人赞同

单步调试，理解执行过程
推荐QT进行界面调试，适用于新手：
zhihu.com/question/2798...

发布于 2016-04-25　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

4

桂能，刘涛粉

4 人赞同

其实深度学习的东西，逻辑上模块还是很清晰的，跟搭积木一样，但是要把基础的模块弄清楚，而基础模块属于深度学习理论的东西，跟具体的库关系不大，无论你是用theano torch还是别的，具体的就是一些基本模块，比如卷积 全连接矩阵 吉布斯采样 数来数去也不是很多，主要要把矩阵玩的很熟，他里面都是各个矩阵转来转去的，比如卷积就是把二维图像矩阵通过某种操作变成另外一堆二维矩阵，这个过程就是一个小积木，然后所有的积木都是通过矩阵运算拼到一起去，变成一个大的构建，这个大构建上面有很多未知的参数，然后弄一个算法比如梯度下降把这些参数确定出来

如果题主完全是门外汉，建议用一下keras这个库，看几个例子就明白深度学习大概怎么玩了

至于并发方面 都是最后求解的事情，有了架子之后，理解怎么分布式去算并不是很难

发布于 2015-10-09　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

3

Melon Wan

3 人赞同

首先应该了解下几个大的类，比如blob，net，layer，solver，然后看怎么从proto文件中初始化创建网络，blob,

layer, net这几层关系中数据是如何传递的。至于不同的layer，用到那种layer的时候再看就ok。

Fiberleif

14

正好最近的工作与Caffe相关，就把网上已有的一些资料以及自己
读代码时候的亲身经历简单地整理了一下，希望能对大家有所帮助~
==========

Caffe's Abstract Framework:

就像楼上一些回答里说到的，要读懂caffe，首先要熟悉Blob,Layer,Net,Solver这几个大类。这四个大类自下而上，环环相扣，贯穿了整个caffe的结构，下面先分别简单地介绍一下这四个类的主要作用。

- **Blob：**作为数据传输的媒介，无论是网络权重参数，还是输入数据，都是转化为Blob数据结构来存储
- **Layer：**作为网络的基础单元，神经网络中层与层间的数据节点、前后传递都在该数据结构中被实现，层类种类丰富，比如常用的卷积层、全连接层、pooling层等等，大大地增加了网络的多样性
- **Net：**作为网络的整体骨架，决定了网络中的层次数目以及各个层的类别等信息
- **Solver：**作为网络的求解策略，涉及到求解优化问题的策略选择以及参数确定方面，修改这个模块的话一般都会是研究DL的优化求解的方向。

==========

Caffe's Concrete Framework:

1. Blob：

1.1. Blob的类型描述

Caffe内部采用的数据类型主要是对protocol buffer所定义的数据结构的继承，因此可以在尽可能小的内存占用下获得很高的效率，虽然追求性能的同时Caffe也会牺牲了一些代码可读性。

直观来说，可以把Blob看成一个有4维的结构体（包含数据和梯度），而实际上，它们只是一维的指针而已，其4维结构通过shape属性得以计算出来。

1.2. Blob的重要成员函数和变量

```
shared_ptr<SyncedMemory> data_ //数据
shared_ptr<SyncedMemory> diff_ //梯度
```

```
void Blob<Dtype>::Reshape(const int num, const int channels, const int height,
const int width)
```

重新修改Blob的形状(4维)，并根据形状来申请动态内存存储数据和梯度。

```
inline int count(int start_axis, int end_axis) const
```

计算Blob所需要的基本数据单元的数量。**2. Layer：**

2.1. Layer的类型描述

Layer是网络模型和计算的核心，在数据存储上，主要分成bottom_vecs、top_vecs、weights&bias三个部分；在数据传递上，也主要分为LayerSetUp、Reshape、Forward、Backward四个过程，符合直观上对层与层之间连接的理解，贴切自然。

2.2. Layer的重要成员函数和变量

```
vector<Dtype> loss_ //每一层都会有一个loss值，但只有LossLayer才会产生非0的loss
vector<shared_ptr<Blob<Dtype> > > blobs_ //Layer所学习的参数，包括权值和偏差
```

```
virtual void LayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top)
```

通过bottom Blob对象的形状以及LayerParameter(从prototxt读入)来确定Layer的学习参数（以Blob类型存储）的形

状。

```
 virtual void Reshape(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top)
```

通过bottom Blob对象的形状以及Layer的学习参数的形状来确定top Blob对象的形状。

```
virtual void Forward(const vector<Blob<Dtype>*> &bottom,
                     vector<Blob<Dtype>*> *top) = 0
```

Layer内部数据正向传播，从bottom到top方向。

```
virtual void Backward(const vector<Blob<Dtype>*> &top,
                      const vector<bool> &propagate_down,
                      vector<Blob<Dtype>*> *bottom) = 0
```

Layer内部梯度反向传播，从top到bottom方向。

3. Net:

3.1. Net的类型描述

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。

3.2. Net的重要成员函数和变量

```
vector<shared_ptr<Layer<Dtype> > > layers_ //构成该net的layers
vector<vector<Blob<Dtype>*> > bottom_vecs_ //每一层layer中的bottom Blobs
vector<vector<Blob<Dtype>*> > top_vecs_ //每一层layer中的top Blobs
vector<shared_ptr<Blob<Dtype> > > params_ //整个net中的learnable parameter
```

```
void Init(const NetParameter& param)
```

根据NetParameter进行net初始化,简单的来说就是先把网络中所有层的bottom Blobs&top Blobs（无重复）实例化，并从输入层开始，逐层地进行Setup的工作，从而完成了整个网络的搭建，为后面的数据前后传输打下基础。

```
vector<Blob<Dtype>*>& Forward(const vector<Blob<Dtype>* > & bottom,
                              Dtype* loss = NULL)
void Net<Dtype>::Backward()
```

是对整个网络的前向和方向传导，各调用一次就可以计算出网络的loss了。

4. Solver

4.1. Solver的类型描述

Solver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，根据优化算法的不同会派生不同的类，而基于这些子类就可以对网络进行正常的训练过程。

4.2. Solver的重要成员函数和变量

```
shared_ptr<Net<Dtype> > net_ //net对象
```

```
void Step(int iters)
```

对已初始化后的网络进行固定次数的训练迭代过程。

```
ComputeUpdateValue();
net_->Update();
```

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。

==========

说到这儿，大家可能已经对Caffe的自底向上的主要结构有了一个大致的了解~

除了这些之外呢，Caffe里面还涉及到了许多特色模块，比如google的轻量级高效数据传输工具protobuf、负责stream输出模块的glog和负责命令行便捷输入的gflags、factory宏定义等等...

最后想在这里向caffe的contributors以及caffe blog的博主们表示深深的感激~ 由于本人的C++经验不多，所以最花了

最后感谢caffe的contributors以及caffe blog的博主们表示深深的感激~ 由于本人的C++经验不多，所以是花了较长的时间才大致理解caffe的实现（呜呜TT），但现在回头想来确实收获很大，不管是在代码架构方面，还是在实现细节方面，caffe都有很多值得借鉴的地方，相信大家在阅读完caffe之后也一定会受益匪浅~~~

发布于 2016-04-07　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

天堂之拳，卧槽，话题经验是什么鬼，王婆卖瓜么！

有这功夫来问，还不如去下来工具包用一用

发布于 2016-07-26　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

6

汤旭，Deep Learning/Face Recognition/Object …

6 人赞同

来膜拜下楼上的caffe大神们，我玩了好久的caffe都还是不溜。说下像我这种菜鸟怎么入门吧，先多玩几个examples，然后分析下简单的example代码流程。看caffe源码最好从底层往顶层看，也就是caffe.proto，blob，layer，net，solver依次。另外玩转caffe最主要还是多接触点课题，根据课题需要去探索caffe。熟悉的应用caffe无非也就几个注意的：1、把数据集转换成caffe可以读入的格式，比如lmdb，hdf5等等。这方面谷歌可以搜出很多教程。2、输入、输出定义好，也就明白了具体怎么利用caffe，想好你的模型的结构，如何输入，输出的分类器或者优化函数是什么。3、根据自己需要去改loss_function对应的layer。4、一些小的深度学习设置技巧都直接做成了接口。

----------分割线----------
谢谢@甘宇飞 的博客带我入门。

编辑于 2015-11-12　3 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

5

张晓，一句话介绍

5 人赞同

我贴一个我自己看的源码分析的链接。博客：一亩半分地

发布于 2016-03-30　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

3

Hatena123，博士 数字图像处理 机器学习 数据挖掘

3 人赞同

个人觉得caffe的代码CPU的相对比较好读一点。读之前先去了解一下protocol buffer。大概来说就是他用blob来代表各种维度的层，有点像theno的N-D tensor。然后每个数据层做对应的操作。你可以看看几个简单的，比如Euclidian_loss_layer这种就明白了。caffe的数据抓取是通过开一个单独的线程按照顺序从db里面抓取的。

caffe的优点是代码结构不错，我自己给caffe写过几个数据层读我自己的数据，他那种工厂模式我很喜欢。写起来很方便而且不害怕错（这就是工厂的一大好处，算是吧。。。耦合度不高）还有个最大的优点是github上都star了7000多了，这种actively-developed代码上哪儿找去？自己调BUG时间都省了。缺点是因为全部都是用一种模式写的所以效率不是很高，对比其他效率最高的代码而言。

发布于 2015-12-23　1 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

5

哈哈冻

5 人赞同

作为一个caffe的初学者，感觉上边几位高票的答案已经写的非常的好了。
但是我发现有一个普林斯顿大学Vision Group的caffe tutorial PPT做的很不错，因此想和大家分享下。

链接： robots.princeton.edu/co...

p.s. 感觉nsight IDE不错，nvidia出品，安装也很简单：

```
sudo apt-get install nvidia-nsight
```

关于nsight IDE如何进行caffe的调试，可以参见这里：Build and Debug Caffe in Eclipse Nsight

编辑于 2016-05-09　5 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

3

longriyao

3 人赞同

关于IO系统系列文章，非常好！！！

Physcal - 博客园

发布于 2016-05-29　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

2

wzpfish，Think deeper and see further.

2 人赞同

米8的答案逻辑很清楚了。就按照怎么训练神经网络，就按什么顺序看代码。看的过程中可以先不要太在意实现细节，搞清楚函数的功能即可，最后再仔细过一遍会有更深刻全面的理解。

caffe代码是好多个人写的吧，里面有些代码风格都不一样。
还有些函数又臭又长，没记错的话有个函数貌似写了几百行。。: (

发布于 2015-10-12　3 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

张蕊，UCLA

Caffe 都已经不更新了，好么。逐层训练等方法都没有。

Caffe 作者现在在帮**Google** 开发 **Tensorflow**了。

TuneLayer 功能最全

github.com/zsdonghao/Tu...

编辑于 2016-08-03　2 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

汤宝宝，研究生 模式识别

给位大神，能告诉我怎么样用caffe Siamese网读入自己的彩色图像么，数据怎么准备

发布于 2016-06-02　添加评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

0

wen shao

请问各位大牛，Caffe为什么好像没有进行逐层训练，直接整个网络进行微调

发布于 2015-03-26　6 条评论　感谢　分享　收藏 • 没有帮助 • 举报 • 作者保留权利

---

1 个回答被折叠（为什么？）

ldy 填写话题经验，提升回答可信度

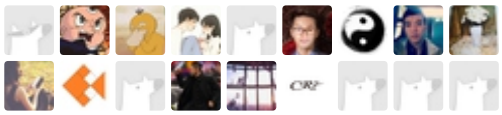
写回答…

匿名 | 允许规范转载 ▼

发布回答

关注问题

2176 人关注该问题



相关问题

深度学习目前主要有哪些研究方向? 10 个回答

如何向非物理专业的同学解释重整化群？ 13 个回答

BRETT 机器人算有「学习」的能力么？ 15 个回答

为什么 Deep Learning 最先在语音识别和图像处理领域取得突破？ 33 个回答

机器学习有很多关于核函数的说法，核函数的定义和作用是什么？ 28 个回答

问题状态

最近活动于 昨天 18:59 • 查看问题日志

被浏览 **62136** 次，相关话题关注者 **246471** 人