https://www.cnblogs.com/denny402/p/5679037.html

[**caffe的python接口学习（1）：生成配置文件**](https://www.cnblogs.com/denny402/p/5679037.html)

caffe是C++语言写的，可能很多人不太熟悉，因此想用更简单的脚本语言来实现。caffe提供matlab接口和python接口，这两种语言就非常简单，而且非常容易进行可视化，使得学习更加快速，理解更加深入。

半年前，我在学习CAFFE的时候，为了加深理解，因此写下了随笔，有了一系列的caffe学习文章。半年过去，很多人问到关于python接口和可视化的一些问题，现在有点空闲时间，就再次写下一些随笔，大家一起来学习。有些重复的内容，我就不再多讲，如果大家有兴趣可移步：

[如何配置CAFFE的python接口？](http://www.cnblogs.com/denny402/p/5088399.html)

[如何将图片转换成LMDB文件？](http://www.cnblogs.com/denny402/p/5082341.html)

[如何计算训练数据的均值文件？](http://www.cnblogs.com/denny402/p/5102328.html)

以上这些操作都是训练之前的预处理操作，不管是用什么接口，都要用到。

首先，我们需要掌握的，就是如何写配置文件，通过下面的代码来学习：

[复制代码](javascript:void(0);)

# -\*- coding: utf-8 -\*-

"""

Spyder Editor

"""

from caffe import layers as L,params as P,to\_proto

path='/home/xxx/data/' #保存数据和配置文件的路径

train\_lmdb=path+'train\_db' #训练数据LMDB文件的位置

val\_lmdb=path+'val\_db' #验证数据LMDB文件的位置

mean\_file=path+'mean.binaryproto' #均值文件的位置

train\_proto=path+'train.prototxt' #生成的训练配置文件保存的位置

val\_proto=path+'val.prototxt' #生成的验证配置文件保存的位置

#编写一个函数，用于生成网络

def create\_net(lmdb,batch\_size,include\_acc=False):

#创建第一层：数据层。向上传递两类数据：图片数据和对应的标签

data, label = L.Data(source=lmdb, backend=P.Data.LMDB, batch\_size=batch\_size, ntop=2,

transform\_param=dict(crop\_size=40,mean\_file=mean\_file,mirror=True))

#创建第二屋：卷积层

conv1=L.Convolution(data, kernel\_size=5, stride=1,num\_output=16, pad=2,weight\_filler=dict(type='xavier'))

#创建激活函数层

relu1=L.ReLU(conv1, in\_place=True)

#创建池化层

pool1=L.Pooling(relu1, pool=P.Pooling.MAX, kernel\_size=3, stride=2)

conv2=L.Convolution(pool1, kernel\_size=3, stride=1,num\_output=32, pad=1,weight\_filler=dict(type='xavier'))

relu2=L.ReLU(conv2, in\_place=True)

pool2=L.Pooling(relu2, pool=P.Pooling.MAX, kernel\_size=3, stride=2)

#创建一个全连接层

fc3=L.InnerProduct(pool2, num\_output=1024,weight\_filler=dict(type='xavier'))

relu3=L.ReLU(fc3, in\_place=True)

#创建一个dropout层

drop3 = L.Dropout(relu3, in\_place=True)

fc4 = L.InnerProduct(drop3, num\_output=10,weight\_filler=dict(type='xavier'))  
 #创建一个softmax层

loss = L.SoftmaxWithLoss(fc4, label)

if include\_acc: #在训练阶段，不需要accuracy层，但是在验证阶段，是需要的

acc = L.Accuracy(fc4, label)

return to\_proto(loss, acc)

else:

return to\_proto(loss)

def write\_net():

#将以上的设置写入到prototxt文件

with open(train\_proto, 'w') as f:

f.write(str(create\_net(train\_lmdb,batch\_size=64)))

#写入配置文件

with open(val\_proto, 'w') as f:

f.write(str(create\_net(val\_lmdb,batch\_size=32, include\_acc=True)))

if \_\_name\_\_ == '\_\_main\_\_':

write\_net()

[复制代码](javascript:void(0);)

通过上面这个文件的执行，我们就会得到两个配置文件：train.prototxt和val.prototxt，分别用于训练阶段和验证阶段。

这种方式生成配置文件，必须有个前提，就是要先把原始图片转换成LMDB文件才行。如果我们已经把原始图片做成了一个列表清单（txt文件，一行一张图片），则可以不用LMDB格式作为输入数据，可以用ImageData作为数据源输入，代码如下：

[复制代码](javascript:void(0);)

# -\*- coding: utf-8 -\*-

from caffe import layers as L,params as P,to\_proto

path='/home/xxx/data/'

train\_list=path+'train.txt'

val\_list=path+'val.txt'

train\_proto=path+'train.prototxt'

val\_proto=path+'val.prototxt'

def create\_net(img\_list,batch\_size,include\_acc=False):

data,label=L.ImageData(source=img\_list,batch\_size=batch\_size,new\_width=48,new\_height=48,ntop=2,

transform\_param=dict(crop\_size=40,mirror=True))

conv1=L.Convolution(data, kernel\_size=5, stride=1,num\_output=16, pad=2,weight\_filler=dict(type='xavier'))

relu1=L.ReLU(conv1, in\_place=True)

pool1=L.Pooling(relu1, pool=P.Pooling.MAX, kernel\_size=3, stride=2)

conv2=L.Convolution(pool1, kernel\_size=53, stride=1,num\_output=32, pad=1,weight\_filler=dict(type='xavier'))

relu2=L.ReLU(conv2, in\_place=True)

pool2=L.Pooling(relu2, pool=P.Pooling.MAX, kernel\_size=3, stride=2)

conv3=L.Convolution(pool2, kernel\_size=53, stride=1,num\_output=32, pad=1,weight\_filler=dict(type='xavier'))

relu3=L.ReLU(conv3, in\_place=True)

pool3=L.Pooling(relu3, pool=P.Pooling.MAX, kernel\_size=3, stride=2)

fc4=L.InnerProduct(pool3, num\_output=1024,weight\_filler=dict(type='xavier'))

relu4=L.ReLU(fc4, in\_place=True)

drop4 = L.Dropout(relu4, in\_place=True)

fc5 = L.InnerProduct(drop4, num\_output=7,weight\_filler=dict(type='xavier'))

loss = L.SoftmaxWithLoss(fc5, label)

if include\_acc:

acc = L.Accuracy(fc5, label)

return to\_proto(loss, acc)

else:

return to\_proto(loss)

def write\_net():

#

with open(train\_proto, 'w') as f:

f.write(str(create\_net(train\_list,batch\_size=64)))

#

with open(val\_proto, 'w') as f:

f.write(str(create\_net(val\_list,batch\_size=32, include\_acc=True)))

if \_\_name\_\_ == '\_\_main\_\_':

write\_net()

[复制代码](javascript:void(0);)

 即第一层由原来的Data类型，变成了ImageData类型，不需要LMDB文件和均值文件，但需要一个txt文件。

[**caffe的python接口学习（2）：生成solver文件**](https://www.cnblogs.com/denny402/p/5679154.html)

caffe在训练的时候，需要一些参数设置，我们一般将这些参数设置在一个叫solver.prototxt的文件里面，如下：

**base\_lr: 0.001**

**display: 782**

**gamma: 0.1**

**lr\_policy: “step”**

**max\_iter: 78200**

**momentum: 0.9**

**snapshot: 7820**

**snapshot\_prefix: “snapshot”**

**solver\_mode: GPU**

**solver\_type: SGD**

**stepsize: 26067**

**test\_interval: 782**

**test\_iter: 313**

**test\_net: “/home/xxx/data/val.prototxt”**

**train\_net: “/home/xxx/data/proto/train.prototxt”**

**weight\_decay: 0.0005**

假设我们有50000个训练样本，batch\_size为64，即每批次处理64个样本，那么需要迭代50000/64=782次才处理完一次全部的样本。我们把处理完一次所有的样本，称之为一代，即epoch。所以，这里的test\_interval设置为782，即处理完一次所有的训练数据后，才去进行测试。如果我们想训练100代，则需要设置max\_iter为78200.

同理，如果有10000个测试样本，batch\_size设为32，那么需要迭代10000/32=313次才完整地测试完一次，所以设置test\_iter为313.

 学习率变化规律我们设置为随着迭代次数的增加，慢慢变低。总共迭代78200次，我们将变化lr\_rate三次，所以stepsize设置为78200/3=26067，即每迭代26067次，我们就降低一次学习率。