# CUDA-CONVNET

## 项目介绍

Cuda-convnet项目 (<https://code.google.com/p/cuda-convnet/>) 是googlecode上的一个开源项目。当前存放在100服务器上的备份文件结构如下所示：

-- cuda-convnet

-- cuda-convnet-master.zip 源文件

-- cuda-convnet-vs-proj-oringinal.zip vs2010解决方案

-- cifar-10-py-colmajor.tar.gz cifar数据集

-- convnet.bat 一个训练用批处理示例

-- nlc-cuda-convnet-vN.M.zip 修改版的cuda-convnet源文件

-- nlc-cuda-convnet-vN.N-release.zip 修改过的cuda-convnet可执行文件(无法保证一定会兼容之前版本)

--nlc-readme.txt 相关说明文件

-- dependences

-- python64 64bit python和相关库

-- cudatools\_4.0.17\_win\_64.msi 精简版cuda 4.0依赖项

-- pthreads-w32-2-9-1-release.rar pthreads for windows

-- hap-depends213790.rar dll依赖查找助手

## 支持功能

* + 支持图像分类, 回归, 最基本的图像重建(只支持正方形输入)
  + 原始版本支持convolution(shared/non-shared weights), normalization, pooling (avg and max), full-connection以及一些cross-map操作
  + 修改版本除支持原版功能，修改少许bugs，外还包括dropout, drop-connect full layer, channel layer, stochastic channel-max layer, 3D pooling layer (with max, avg, stochastic, stochastic max, stochastic avg, probability weighting function)

## 安装与配置

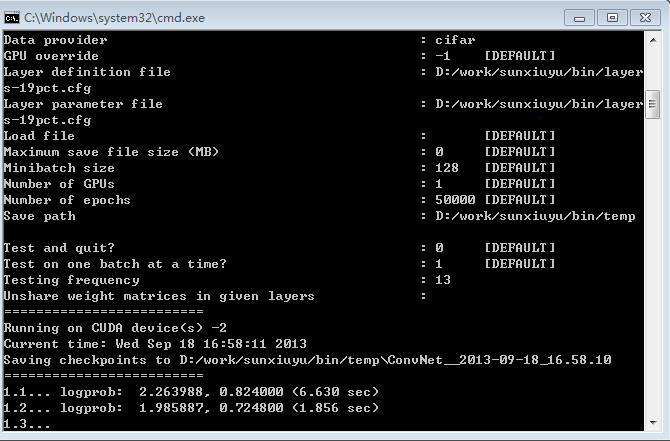
本示例的安装环境为：windows 7 (64 bit) + cuda toolkit-sdk 4.0 + python 2.7 + VS 2010。简略的安装步骤和注意事项如下所示：

* 1. 安装cuda和python
  2. 安装Intel MKL (intel编译器自带，可从100服务器下载)
  3. 将cuda-convnet-master和cuda-convnet-vs-proj-oringinal.zip解压缩到同一文件夹
  4. 打开convnet.sln项目，加入相关头文件目录和依赖项
     1. 加入python和numpy的相关目录
     2. 加入MKL相关目录
     3. 加入pthreads相关目录
  5. 开启预编译宏USE\_MKL (也有可能需要手动开启NUMPY\_INTERFACE宏)
  6. 编译成功后会生成 pyconvnet.pyd

Cuda-convnet项目默认使用cuda4.0作为gpu加速的sdk，如果准备使用新版的cuda 5.0，则需要注意一下几点：

1. 如果无法正确加载vs项目，则可以使用记事本打开 pyconvnet.vcxproj，将\BuildCustomizations\CUDA 4.0.props修改为\BuildCustomizations\CUDA 5.0.props
2. 使用<helper\_cuda.h> 替换<cutil\_inline.h>
3. 使用getLastCudaError替换cutilCheckMsg
4. 在inlucde/cudaconv2/conv\_util.cuh和include/nvmatrix/nvmatrix.cuh中加入#include<helper\_image.h>

在编译成功后解压缩数据集文件，修改批处理中的输入参数，最后运行，如果发现出现dll加载错误信息，则可使用dependency walker检查pyconvnet.pyd所缺少的dll依赖文件。如果一切配置正确，则运行画面如下所示：



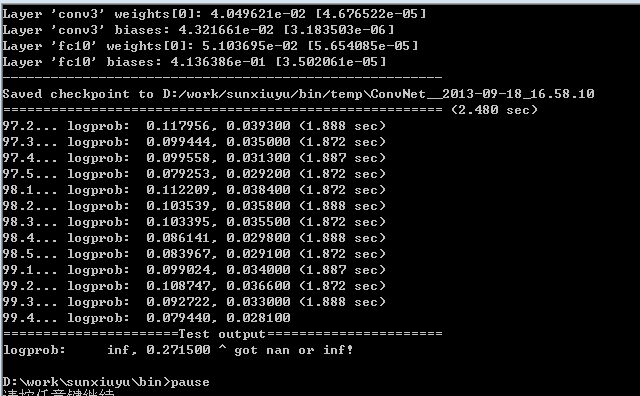


图1

训练结束后则可以使用 convnet.py -f XXXX --test-only=1 --test-range=XXX --multiview-test=1 --logreg-name=logprob 参数训练结果。注意：如果如上图所示出现了got nan or inf! 则表示1.初始化的参数有问题导致最后输出越界(修改初始化参数) 2.程序中有除以0的情况(一般情况下不会出现，出现需要修改cuda code) 3. GPU设备出错(读取上一个存储点继续训练即可)。

## 数据组织

这里给出一个如何生成convnet可以识别的数据集的示例。

首先将图片按照下图所示方式组织，其中0-9表示类别编号（当前默认所有图像大小一致，文件名后缀一致）。

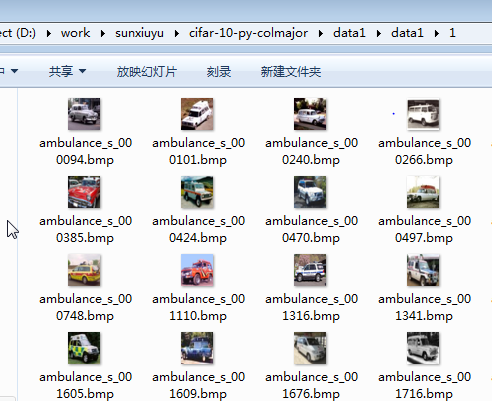
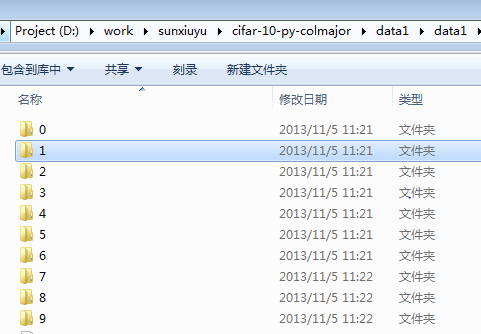


图2

运行函数make\_batch函数生成CIFAR兼容格式数据。其中imagefolder为图像目录，numclass表示类别(10),outfolder表示数据输出目录，ext表示图片后缀，默认bmp。sep表示batch个数。例如CIFAR10拥有60000张图片，如果sep=6，则会随机选取10000张图片(每一类1000张)作为一个batch。该示例函数并没有统计训练数据的均值，同时也没有生成新的batch\_meta数据。所以，如果想直接使用CIFARDataProvider或者CroppedCIFARDataProvider，需要自行修改对应的batch\_meta。

def make\_batch(imagefolder,numclass,outfolder,ext='.bmp',sep=1):

batchname = 'data\_batch\_'

#filenames -- [][(filename,basename,class)]

filenames = getfiles(imagefolder,numclass,outfolder,ext)

width,height = 32,32

# for each batches

for i in range(0,sep):

filenames\_by\_batches =[]

# seperate the data into batches

for j in range(0, int(numclass)):

import math

step = int ( math.ceil( len(filenames[j]) \* 1.0 / sep))

fbegin = step \* i

fend = step \* (i + 1)

if fend > len(filenames[j]) :

fend = len(filenames[j])

#print fbegin,fend

fs = filenames[j][fbegin:fend]

filenames\_by\_batches += fs

#random\_shuffle

random.shuffle(filenames\_by\_batches)

batch\_data = []

batch\_filenames= []

batch\_labels = []

bname = batchname + str(i + 1)

#read img

for f in filenames\_by\_batches:

img = cv2.imread(f[0],1)

width,height = img.shape[0],img.shape[1]

b = img[:,:,0].reshape(width \* height,order='C')

g = img[:,:,1].reshape(width \* height,order='C')

r = img[:,:,2].reshape(width \* height,order='C')

im = np.array([b,g,r,dx,dy]).reshape(width \* height \* 3,order='C')

batch\_data.append(im)

batch\_filenames.append(f[1])

batch\_labels.append(f[2])

savefile = os.path.join(outfolder,bname)

batch\_data = np.transpose(np.array(batch\_data,order='C'))

dict = { 'batch\_label' :bname, 'labels' :batch\_labels, 'data' :batch\_data, 'filenames' : batch\_filenames }

util.pickle(savefile,dict)

pass

def getfiles(imagefolder,numclass,outfolder,ext):

filenames = []

#read filenames to list

for i in range(0, int(numclass)):

classfolder = os.path.join(imagefolder,str(i))

filenames.append([])

for basename in os.listdir(classfolder):

if os.path.splitext(basename)[1] == ext:

filename = os.path.join(classfolder,basename)

filenames[i].append((filename,basename,int(i)))

#shuffle the list

for i in range(0,int(numclass)):

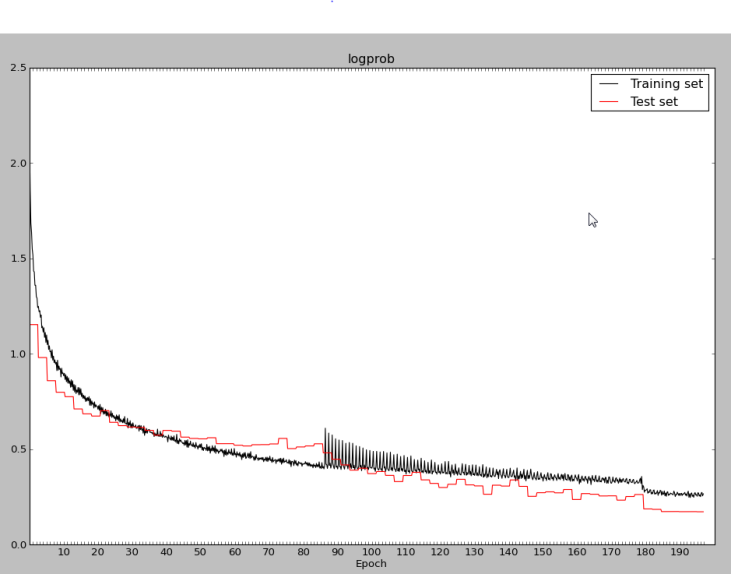
random.random()

random.shuffle(filenames[i])

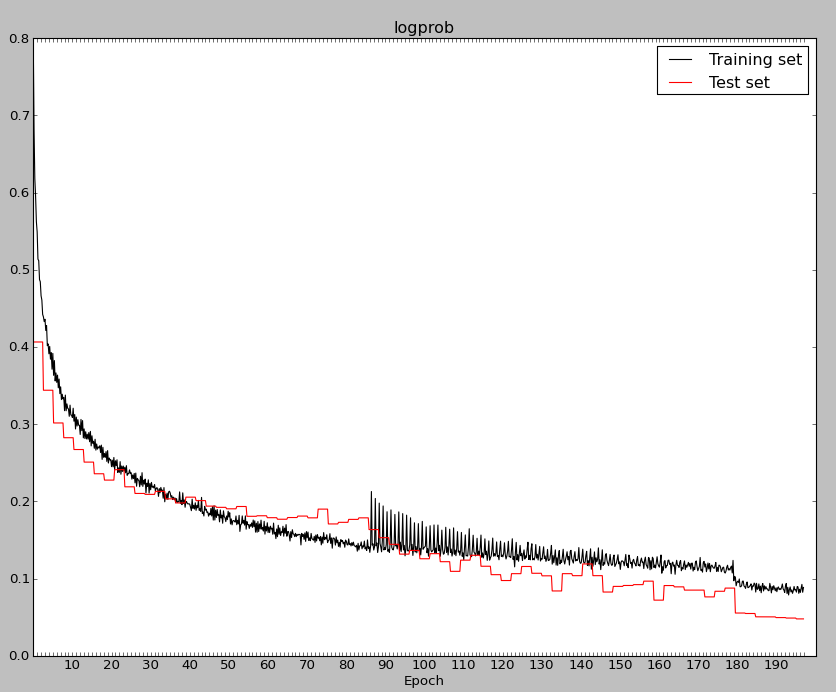
return filenames

## 查看网络

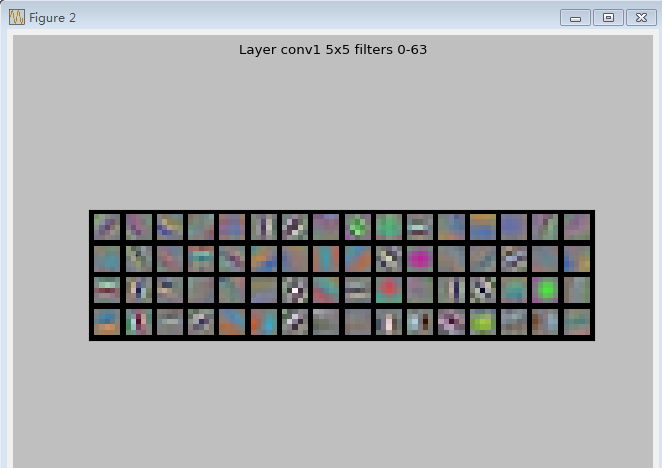
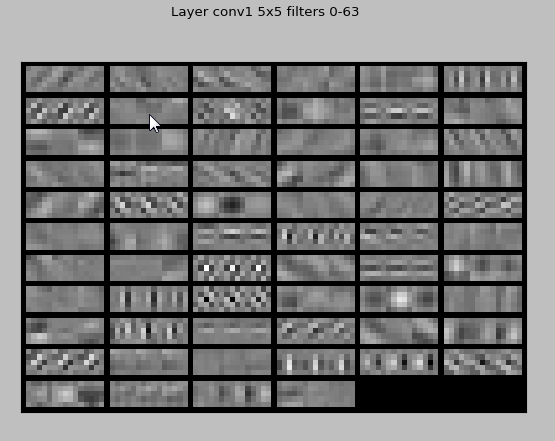
python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00/219.2 --show-cost=logprob --cost返回log probability of data



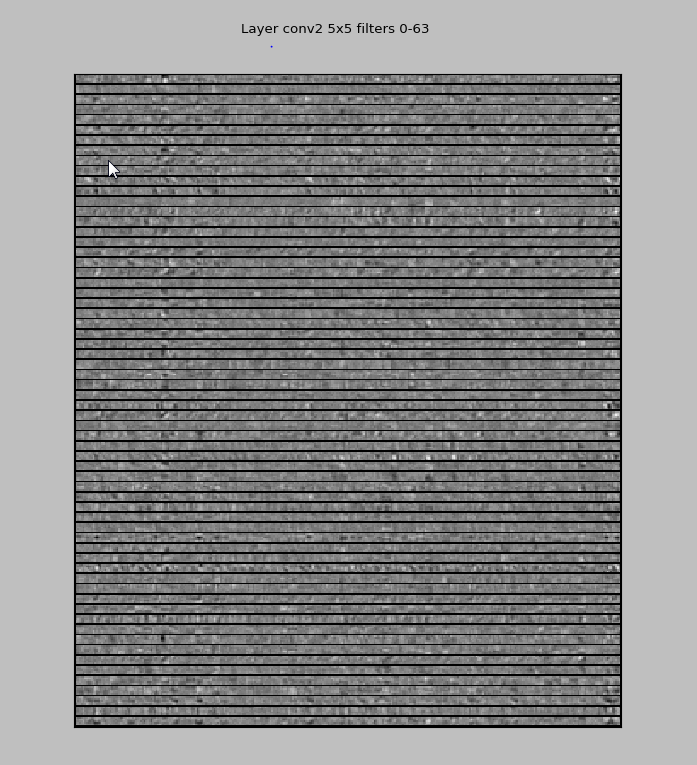
python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00/219.2 --show-cost=logprob --cost --cost-idx=1 分类误差



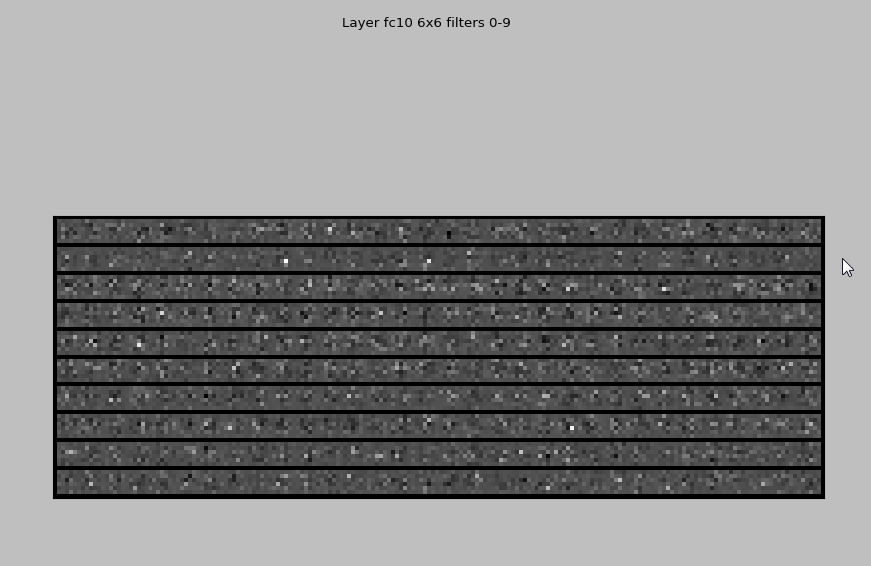
cost

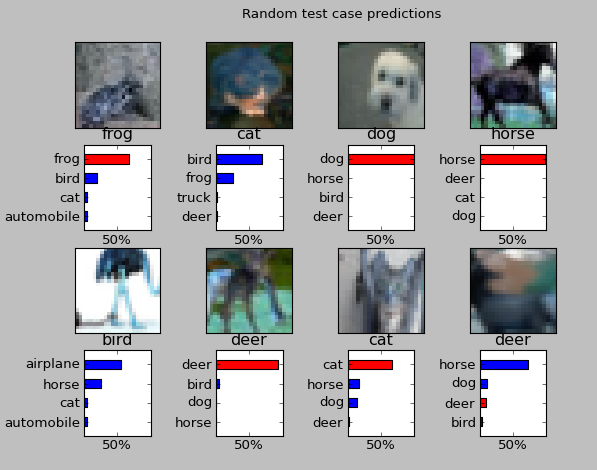
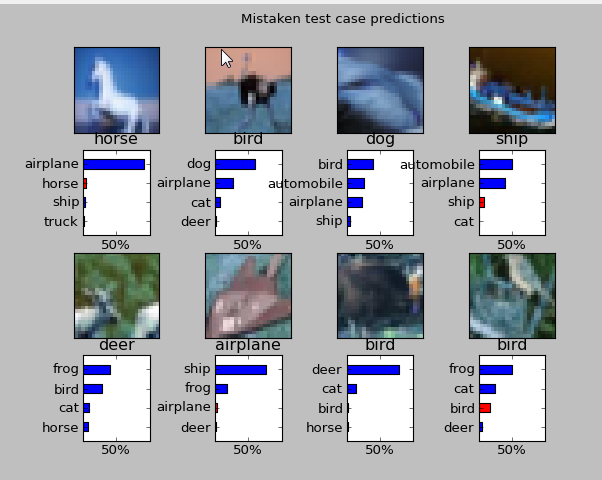
python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00/219.2 --show-filters=conv1 [--no-rgb=0|1] 返回conv1对应的fileters



python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00/219.2 --show-filters=conv2 返回第conv2对应的fileters



python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00/219.2 --show-filters=fc10 channel=32全连接的可视化

python shownet.py -f /storage/tmp/ConvNet\_\_2013-11-04\_15.59.00 --show-preds=layer\_name --test-range=XXX [ --only-errors=0|1 ] 随机选取8个图像测试并显示预测结果( --only-erros只显示错误 )

## Python代码简要说明

关于cuda-convnet项目的详细功能介绍可以参考 <https://code.google.com/p/cuda-convnet/> 这里只给出关于python封装实现的一些简要说明。

### options.py

该类主要用于解析convnet，gpumodel和shownet中所需的命令行参数

class Option

def \_\_init\_\_(self, letter, name, desc, parser, set\_once, default, excuses, requires, save):

OptionsParser

def add\_option(self, letter, name, parser, desc, set\_once=False, default=None, excuses=[], requires=[], save=True):

letter 对应cmd命令参数( 如果letter是一个字符则 -letter 否则 --letter )

name 内部存储时dict中的key名称

parser 参数解析类，用于解析输入参数的方式

desc 描述说明

default 默认值

set\_once

excuses 当被设置成EXCLUDE\_ALL时 其他参数由必选项变为可选项

excuses=[ 'data\_path', 'save\_path' ] 则中不需要再指定--data-path, --save-path

requires 默认为True

save 默认为True

OptionParser

@staticmethod

def parse(value) 解析参数

@staticmethod

def to\_string(value) 打印值

@staticmethod

def get\_type\_str() 打印类型

OptionParser 解析参数类，抽象类，定义接口

-- IntegerOptionParser 整型参数

-- BooleanOptionParser 布尔型参数 0|1

-- StringOptionParser 字符串参数

-- FloatOptionParser 单浮点型参数

-- RangeOptionParser 范围参数 int [-int] 一个值或一个从a到b的范围 返回为list

-- ListOptionParser 用来解析一个list参数 需要与其他类型结合, 两种工作方式。

ListOptionParser( [FloatOptionParser, IntegerOptionParser] ) 只解析float, int

ListOptionParser( FloatOptionParser ) 解析float类型的不定长数组 float,float,...

-- OptionExpression 调用eval函数动态解析字符串命令

OptionExpression("[o \* 10 for o in eps\_w]") 根据eps\_w动态生成参数

**IGPUModel**

接口类，定义了训练/测试的流程

def \_\_init\_\_(self, model\_name, op, load\_dic, filename\_options=None, dp\_params={}):

初始化GPU模型，除输入参数外，还需要配置当前模型的训练状态self.model\_state， 最后初始化模型和数据处理类

|  |  |  |
| --- | --- | --- |
| 参数名 | Key名称 | 解释描述 |
| -f | load\_file | 训练过程中的中间数据 |
| --train-range | train\_batch\_range | 用于训练的data batch n[-m] |
| --test-range | test\_batch\_range | 用于测试的data batch n [-m] |
| --data-provider | data\_type | 解析输入数据的provider |
| --test-freq | testing\_freq | 测试频率 每隔--test-freq次迭代test一次 默认25 |
| --epochs | num\_epochs | 迭代次数 默认500，在convnet中该默认值被修改！！ |
| --data-path | data\_path | 数据目录 |
| --save-path | save\_path | 训练过程中的保存文件 |
| --max-filesize | max\_filesize\_mb | 训练过程中的保存文件的最大size 默认500 |
| --max-test-err | max\_test\_err | 当test误差小于--max-test-err时才保存log  注意该参数已经在convnet.py中被删除 |
| --num-gpus | num\_gpus | gpu设备个数 |
| --test-only | test\_only | 只做测试 |
| --zip-save | zip\_save | 压缩保存 |
| --test-one | test\_one | 一次只测试一个batch |
| --gpu | gpu | 由--num-gpus动态生成 |

IGPUModel所需要的参数列表

### convnet.py

对IGPUModel的一个gpu版实现

op 参数列表

model ConvNet实例

load\_dic 如果-f被给定，则读取文件中的参数信息到load\_dic, 否则为none

**ConvNet (IGPUModel)**

@classmethod

get\_options\_parser(cls) 返回需要解析的options 该函数删除了max\_test\_err选项，并且修改了IGPUModel中的部分默认参数，最后注册新的数据处理类DataProvider

|  |  |  |
| --- | --- | --- |
| 参数名 | Key名称 | 解释描述 |
| --mini | minibatch\_size | 随机梯度下降中的batch尺寸 |
| --layer-def | layer\_def | 网络结构配置文件 |
| --layer-params | layer\_params | 网络对应参数（权重学习参数等） |
| --check-grads | check\_grads | 利用模拟数据验证梯度计算正确性 |
| --multiview-test | multiview\_test | data-augment时的不同测试方式 |
| --crop-border | crop\_border | 切割时的边缘尺寸 |
| --logreg-name | logreg\_name | Cropped DP: logreg layer name ( for multiview-test )  在该模式下soft-max后需要跟一个逻辑回归层 |
| --conv-to-local | con\_to\_local | Convert given conv layers to unshared local  将卷积层转换成local层（无共享权值） |
| --unshare-weights | unshare\_weights | Unshare weight matrices in given layers |
| --conserve-mem | conserve\_mem | Conserve GPU memory (slower) |

ConvNet所需额外参数表

### shownet.py

继承自ConvNet ，定义了一些可视化分析函数。

### data.py

定义了数据接口类

DataProvider

def \_\_init\_\_(self, data\_dir, batch\_range=None, init\_epoch=1, init\_batchnum=None, dp\_params={}, test=False):

#### 一个简单的DataProvider

这里给出的是一个同样针对CIFAR数据集的DataProvider，但是将输入由默认的三通道RGB图像修改为单通道输入（注意：cuda-convnet无法在channels > 4 && channels %16 != 0 的数据上进行卷积操作）自定义的DataProvider只需要实现初始化函数，get\_next\_batch (获取下一个batch) 和 get\_data\_dims(当前输入map的尺寸) 。get\_plottable\_data函数只用于可视化时的彩色filters显示，对于通道数不为3的数据可以直接抛出个异常。

class MyDataGrayProvider(LabeledMemoryDataProvider):

def \_\_init\_\_(self, data\_dir, batch\_range, init\_epoch=1, init\_batchnum=None, dp\_params={}, test=False):

LabeledMemoryDataProvider.\_\_init\_\_(self, data\_dir, batch\_range, init\_epoch, init\_batchnum, dp\_params, test)

for d in self.data\_dic:

d['data'] = n.require( d['data'], dtype=n.single, requirements='C')

d['labels'] = n.require(d['labels'].reshape((1, d['data'].shape[1])), dtype=n.single, requirements='C')

self.filenames = d['filenames']

self.num\_colors = 1

self.img\_size = 32

pass

def get\_next\_batch(self):

epoch, batchnum, datadic = LabeledMemoryDataProvider.get\_next\_batch(self)

return epoch, batchnum, [datadic['data'], datadic['labels']]

def get\_data\_dims(self, idx=0):

return self.img\_size\*\*2 \* self.num\_colors if idx == 0 else 1

def get\_plottable\_data(self, data):

raise ("No Implementation")

### convdata.py

主要提供了针对CIFAR数据集的CIFARDataProvider和CroppedCIFARDataProvider类，以及用于验证梯度计算的DummyConvNetDataProvider类。

### layer.py

定义了一系列用以解析网络配置文件的函数接口，与cuda-convnet项目中layer.cu里的类相对应。可以将固定参数(不可以在训练过程中更改)的配置写在

### util.py

该文件内存储了一些如数据存取，lib库读取所需要的通用函数。

### OrderedDict.py

一个字典类，在LayerParser相关类中用来存储数据

## 新添加的python文件

### mytools.py

* make\_batch( imagefolder, numclass, outfolder, channels=[‘r’,’g’,’b’], ext='.bmp', sep=1, prev=None)
  + 注意：当前需要保证每一类中的图像数都能被sep整除
  + channels可以设置为[‘r’,’g’,’b’,’gray’,’h’,’s’,’v’,’l’,’la’,’lb’,’dx’,’dy’,’dm’]
  + prev为预处理函数, 输入为包含channel maps的list
* preprocessing(folder, imgsize = 32, numcolor = 3, begin = 1, num = 10, func = None)
  + folder输入为data\_batch对应的目录
  + func为预处理函数，当前已有预处理函数为
    - preprocessing\_svhn(img)
    - svhn\_localnorm(img)
    - svhn\_globalnorm(img)

### mydata.py

* CroppedNCLDataProvider
  + 支持CroppedCifar10DataProvider中的配置参数
  + mean : 是否减去像素点对应的均值
  + imgsize : 图像尺寸
  + numcolors : channel个数
  + shuffle : 0|1 是否随机排列训练样本
  + rota : 0|1 是否旋转，设置为1时会在-40度到+40度之间随机旋转
  + scale: 0|1 是否缩放，设置为1时会在0.4倍到1.2倍之间随机缩放图像
  + 在\_\_init\_\_中还可手动修改self.preprocessing，用来在训练过程中预处理图像数据（不建议进行大型计算,会严重影响训练性能）

## 修改版中新添功能

### dropout

在配置文件中的任意一层中添加dropout=0.X

该功能在训练时将会随机将该层输出按照一定概率(dropout)置零，而在训练时乘以(1-dropout)

理论上所有层都支持dropout功能，唯一可能需要修改的地方是layer.py中对应的参数解析函数。另外由于dropout功能是在抽象层实现的，因此可能会与其他带有随机功能的功能层冲突！建议只在conv，local和fc层中使用该功能！

### drop-connect full layer

来源于论文 Regularization of Neural Network using DropConnect, icml 2013

一个有理论证明支持的"dropout"方法，应用在全连接层，理论上可以同样拓展到卷积层。

实验结果显示：速度要比传统fc慢，在实际应用时可以通过dropout全连接之前层的输出来模拟这种方法（性能差距不大）设置方法：

[fulllayer]

type=fcdcb

dropRate=0.5

inputs=XXXX

channels=XXX

### channel layer

一个应用在输出的预处理层，负责输入数据的通道拆分。理论上通过修改可以支持中间层的通道拆分，可能会在以后增加该功能。设置方法：

[channellayer]

type=channel

channels=3 #the number of input channels

id=0 #the id of input channels

num=1 #the channel number of output，default=1, must be less than channels

### stochastic channel-max layer

带有随机选取的max-out层。以往通过实验对比了3种cross-channel形式下的随机选取方法：基于点的随机，基于通道的随机和直接在前一层中使用dropout。实验结果表明基于通道的随机方法和基于点的随机方法性能差不多，但是测试曲线稳定时对应的方差更小。前两种方法都要比直接使用传统dropout + max-out好。设置方法为：

pct.ini

[maxout]

type=reltmax

inputs=XXX,XXX,XXX

params.ini

[maxout]

dropRate=0.X

dropRate的设置是放在参数文件中的，可以在训练过程中进行修改

### 3D pooling layer

最新加入的功能层，用来统一inner-map和cross-map的pooling方法。支持的3D/2D max, avg， probability pooling; 支持随机选取功能(stochastic max, stochastic avg, stochastic probability); 在训练和测试过程中可以配置不同的pooling function。例如：在训练中使用stochastic pooling 而在测试过程中使用probability pooling (Stochastic Pooling for Regularization of Deep Convolutional Neural Networks, iclr 2013 )。设置方法为

[poollayer]

type=pool

pool=3d

pooler.train=rmax

pooler.test=max

inputs=conv1

start=0

sizeX=3

stride=2

outputsX=0

sizeF=2 # cross-map pooling size

srideF=1 # cross-map stride

outputsF=64 # output filters (maps, channels)

channels=64 # input channels(filters, maps)

dropRate=0.5 # dropout rate, used for stochastic function

scale= 1 # 1|0 whether to scale the inputs with (1 - dropRate) as dropout dose

当前支持的pooling方法

max max pooling

avg avg pooling

rmax stochastic max pooling

ravg stochastic avg pooling

ravgnz stochastic avg pooling without zero

sto stochastic pooling

stonz stochastic pooling without zero

prob1 probability weighting pooling ( iclr 2013 )

目前只在一个小的网络上测试了3D stochastic pooling (tain) + 3D pooling (test) 与原始的dropout + 2D max pooling之间的对比，在其他条件相同时下前者比后者有近5%个点的提升。但是该3D pooling层仍有很多工作需测试。例如：进一步的代码校验；如何搭配训练/测试中的pooling组合(尤其是使用随机类pooling)，cross-map pooling时的参数设置等。