**CUDA-CNN**

Document  
1. The simple c version author is [Eric](http://eric-yuan.me/cnn/)  
2. [Overlap Data Transfers in CUDA](http://zhxfl.github.io/cuda-cnn_cuda-stream)

**Results**

CNN accelerated by cuda.  
The [start-of-art result's](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html) of popular datasets  
1. Test on [mnist](http://yann.lecun.com/exdb/mnist/) and get 99.76%, after voting(99.82%) (best 99.79%)  
2. Test on cifar-10 and get 85.08% (best 89%)

**Feature**

1. Use [***Dropout***](http://cs.nyu.edu/~wanli/dropc/) to train the NetWork
2. Support checkpoint, the program will save the best test result and save the network weight in the file "Result/checkPoint.txt", If the program exit accidentally, you can continue the program form this checkpoint.
3. Translate the data set of mnist, including scale, rotate, ***distortion***, accordding to [Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis](http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=D1C7D701BD39935473808DA5A93426C5?doi=10.1.1.160.8494&rep=rep1&type=pdf).
4. The log will be saved in the file "Result/log.txt".
5. In the convolutional layers, you can chose ***combine feature maps***, according to [notes on Convolutional Neural NetWorks](http://cogprints.org/5869/1/cnn_tutorial.pdf).
6. Support [local connection layers](http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks).
7. If you want the program run fast, you can set the "TEST\_EPOCH" to be large.
8. Support ***branchLayer*** and ***combineLayer***, which is designed accordding to ***[goolenet](http://arxiv.org/abs/1409.4842)***, the network structure is no logger an linear structure but Directed acycline graph. \*\*\*

**Compile**

Depend on opencv and cuda  
You can compile the code on windows or linux.

**SDK include path(-I)**

* linux: /usr/local/cuda/samples/common/inc/ (For include file "helper\_cuda"); /usr/local/include/opencv/ (Depend on situation)
* windows: X:/Program Files (x86) /NVIDIA Corporation/CUDA Samples/v6.5/common/inc (For include file "helper\_cuda"); X:/Program Files/opencv/vs2010/install/include (Depend on situation)

**Library search path(-L)**

* linux: /usr/local/lib/
* windows: X:/Program Files/opencv/vs2010/install/x86/cv10/lib (Depend on situation)

**libraries(-l)**

* opencv\_core
* opencv\_highgui
* opencv\_imgproc
* opencv\_imgcodecs (need for opencv3.0)
* ***cublas***
* ***curand***
* ***cudadevrt***

**GPU compute**

* capability 2.0

**Windows**

1. Install vs2010.
2. Download and install [opencv-2.4](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.0.0-beta/) or other higher versions
3. Download and install [cuda-5.0](https://developer.nvidia.com/cuda-downloads) or other higher versions
4. When you create a new project using VS2010, You can find NVIDIA-CUDA project template, create a cuda-project.
5. View-> Property Pages-> Configuration Properties-> CUDA C/C++ -> Device-> Code Generation-> compute\_20,sm\_20
6. View-> Property Pages-> Configuration Properties-> CUDA C/C++ -> Common-> Generate Relocatable Device Code-> Yes(-rdc=true)
7. View-> Property Pages-> Configuration Properties-> Linker-> Input-> Additional Dependencies-> libraries(-l)
8. View-> Property Pages-> Configuration Properties-> VC++ Directories-> General-> Library search path(-L)
9. View-> Property Pages-> Configuration Properties-> VC++ Directories-> General-> Include Directories(-I)

**Linux**

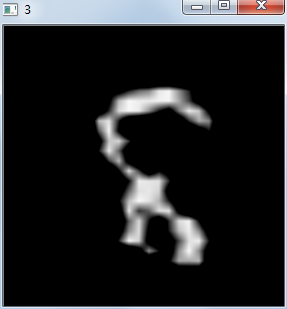
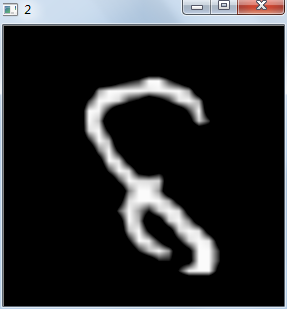
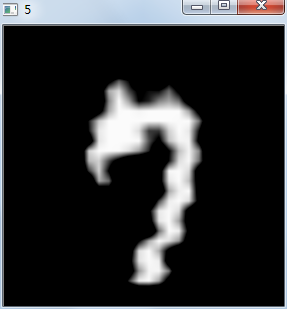
1. Install opencv and cuda
2. Start the ***nsight*** from cuda
3. Create an 'empty cuda' project and import the clone code
4. Project->Proerties for add-> Build-> Settings->CUDA->Device linker mode: separate compilation
5. Project->Proerties for add-> Build-> Settings->CUDA->Generate PTX code 2.0
6. Project->Proerties for add-> Build-> Settings->CUDA->Generate GPU code 2.0
7. Project->Proerties for add-> Build-> Settings->Tool Settings->NVCC Compiler->includes: +/usr/local/cuda/samples/common/inc/; + opencv sdk include path ;
8. Project->Proerties for add-> Build-> Settings->Tool Settings->NVCC Linkers->Libraries: libraries(-l)
9. Project->Proerties for add-> Build-> Settings->Tool Settings->NVCC Linkers->Libraries search path(-L): /usr/local/lib/

Config  
1. [CIFAR10](https://github.com/zhxfl/CUDA-CNN/blob/master/Config/Cifar10Config.txt)  
2. [MNIST](https://github.com/zhxfl/CUDA-CNN/blob/master/Config/MnistConfig.txt)

**Informations**

* Author :zhxfl
* Mail :[zhxfl@mail.ustc.edu.cn](mailto:zhxfl@mail.ustc.edu.cn)
* Welcome for any suggest!!
* [**zhxfl**](http://www.cnblogs.com/zhxfl/)

# [深度学习-使用cuda加速卷积神经网络-手写数字识别准确率99.7%](http://www.cnblogs.com/zhxfl/p/4134834.html)

* **源码和运行结果**
* cuda：https://github.com/zhxfl/CUDA-CNN
* C语言版本参考自：http://eric-yuan.me/
* 针对著名手写数字识别的库mnist，准确率是99.7%,在几分钟内，CNN的训练就可以达到99.60%左右的准确率。
* 
* **参数配置**
* 网络的配置使用Config.txt进行配置##之间是注释，代码会自动过滤掉，其他格式参考如下：
* [复制代码](javascript:void(0);)
* #Comment#
* #NON\_LINEARITY CAN = NL\_SIGMOID , NL\_TANH , NL\_RELU#
* ##
* ##
* ##
* ##
* IS\_GRADIENT\_CHECKING = false;
* BATCH\_SIZE = 200;
* NON\_LINEARITY = NL\_RELU;
* [
* LAYER = CONV;
* KERNEL\_SIZE = 5;
* KERNEL\_AMOUNT = 10;
* WEIGHT\_DECAY = 1e-6;
* POOLING\_DIM = 2;
* ]
* [
* LAYER = CONV;
* KERNEL\_SIZE = 5;
* KERNEL\_AMOUNT = 20;
* WEIGHT\_DECAY = 1e-6;
* POOLING\_DIM = 2;
* ]
* [
* LAYER = FC;
* NUM\_HIDDEN\_NEURONS = 256;
* WEIGHT\_DECAY = 1e-6;
* DROPOUT\_RATE = 0.5;
* ]
* [
* LAYER = FC;
* NUM\_HIDDEN\_NEURONS = 256;
* WEIGHT\_DECAY = 1e-6;
* DROPOUT\_RATE = 0.5;
* ]
* [
* LAYER = SOFTMAX;
* NUM\_CLASSES = 10;
* WEIGHT\_DECAY = 1e-6;
* ]
* [复制代码](javascript:void(0);)
* 1）目前代码支持多个卷积层，多个全链接层。
* 2）卷积层默认带有池化层，池化算法目前只支持最大值池化。
* 3）卷积层的卷积核大小只支持奇数。
* 4）全链接层支持Dropconnect。（配置里面写的是dropout，后面会纠正这个的）
* 5）weight\_decay这个参数如果你不知道是做什么，也可以先不理会，先用这个值就行了。
* **编译代码**
* 1）代码目前依赖cuda-6.0以及opencv，如果你不想去装opencv，可以将util.cu和util.h所有有关opencv的代码都去掉，整个代码只有这里使用到opencv，而且只是由于我在开发过程中需要显示图片来调试而已。
* 2）代码直接可以导入**nsight**然后编译运行。同时也可以再vs2010里面编译运行。
* **代码特性**
* 1）我们对数据做了处理，每次进行训练之前，都会随机的进行旋转、尺度变换、畸变和裁剪。下图是两个例子，实际上，这样做非常有效，使得我们的准确能够更高
* 
* 
* 2）整个代码采用cuda进行加速，其中我们用到了cublas.lib和curand.lib两个库，一个是矩阵运算一个是随机数的生成。我一次性申请了所有需要使用的内存，在程序开始运行之后，就不存在任何CPU和GPU之间的数据交换，事实证明这样非常有效。程序的性能比原来作者C语言版本快了数十倍左右（如果网络比较大，可以达到一百倍左右的加速比）。我们每个epos使用1600ms，处理了60000张图片，也就是训练一张图片大概是0.0266ms。
* 3）实际上，如果训练多个网络，然后进行投票，准确率可以达到99.82%，这个结果是目前为止所有公开发表结果中最好（99.79%）的 。
* [**zhxfl**](http://www.cnblogs.com/zhxfl/)

# [深度学习 CNN CUDA 版本2](http://www.cnblogs.com/zhxfl/p/4155236.html)

* 第1个版本blog在这里：http://www.cnblogs.com/zhxfl/p/4134834.html
* 第2个版本github：https://github.com/zhxfl/CUDA-CNN
* 欢迎fork，在第一个版本的时候，我们只是针对手写数字，也就是黑白图片。在第二个版本中，我加入了很多东西。
* **第二个版本的特性**
* 1、支持rgb图片格式和rgbd图片格式（带有深度信息的图片）训练，带有深度信息的图片可以来源于Kinect。
* 参考论文Anddrew Y.Ng的论文：Convolutional-Recursive Deep Learning for 3D Object Classification，你可以找到对应的带有深度信息的数据集。
* 4d的图片不是这个版本的主要目的，但是你确实可以用这个代码来训练4D的数据集。（我在不久的未来会让这个版本更好的支持4D数据的训练）
* 2、第二个比较突出的特性是你可以看到配置文件的参数更加复杂了，我会对所有参数一一做说明。
* [复制代码](javascript:void(0);)
* 1 #Comment#
* 2
* 3 IS\_GRADIENT\_CHECKING = false; #is true when debug#
* 4 BATCH\_SIZE = 100; #test image size should be divided with no remainder#
* 5 NON\_LINEARITY = NL\_RELU; #NON\_LINEARITY CAN = NL\_SIGMOID , NL\_TANH , NL\_RELU#
* 6 CHANNELS = 3; #1, 3, 4#
* 7 CROP = 0.0; #0<= crop <=imgSize#
* 8 SCALE = 0.0; #ImgSize from -13.0 to 13.0#
* 9 ROTATION = 0.0; #angle from -13.0 to 13.0#
* 10 DISTORTION = 0.0; #just for mnist#
* 11 SHOWIMAGE = false; #show the images after transformation#
* 12
* 13 [
* 14 LAYER = CONV;
* 15 KERNEL\_SIZE = 5;
* 16 KERNEL\_AMOUNT = 7;
* 17 WEIGHT\_DECAY = 1e-6;
* 18 POOLING\_DIM = 2;
* 19 ]
* 20
* 21 [
* 22 LAYER = CONV;
* 23 KERNEL\_SIZE = 5;
* 24 KERNEL\_AMOUNT = 9;
* 25 WEIGHT\_DECAY = 1e-6;
* 26 POOLING\_DIM = 2;
* 27 ]
* 28
* 29 [
* 30 LAYER = FC;
* 31 NUM\_HIDDEN\_NEURONS = 256;
* 32 WEIGHT\_DECAY = 1e-6;
* 33 DROPOUT\_RATE = 0.5;
* 34 ]
* 35
* 36 [
* 37 LAYER = FC;
* 38 NUM\_HIDDEN\_NEURONS = 256;
* 39 WEIGHT\_DECAY = 1e-6;
* 40 DROPOUT\_RATE = 0.5;
* 41 ]
* 42
* 43 [
* 44 LAYER = SOFTMAX;
* 45 NUM\_CLASSES = 10;
* 46 WEIGHT\_DECAY = 1e-6;
* 47 ]
* [复制代码](javascript:void(0);)
* 1）IS\_GRADIENT\_CHECKING 这是一个debug选项（其原理可以参考斯坦福深度学习的教程）。如果你修改了代码，建议你设置为true。你必须确保(g(s + delta) - g(s - delta)) / 2 约等于g(s)。他可以辅助你判断目前的代码是否存在bug。  
  2）BASH\_SIZE，我们训练的方法是mini-batch，这个数值的设置对于收敛的结果和速度都是有影响的。建议可以尝试50,100,150,200等，你会得到不同的试验结果。  
    
  接下来的几个参数都是用来克服overfitting的，对于深度学习而言，训练样本越多，效果会越好。所以我们对于训练数据必须加以扩展。  
    
  3）CROP是裁剪参数，假设图像大小为ImgSize，那么是最终训练的数据应该是ImgSize-CROP,裁剪的窗口起点是随机的，也就是一张图片已经变成了CROP\*CROP张图片了。  
  4）ROTATION是旋转，这步操作对于手写数字非常有效，但是你必须确保旋转的角度不要过大，比如13度，那么最后代码训练的图片都会被随机的旋转角度[-13,13]，这是一个区间。  
  5）DISTORTION又称为畸变，这个也比较适合手写数字，参数越大，图片变化越大，从大量实验看，针对手写数据集，设置为3.4是比较合适的，原理参考论文Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis  
  6）SHOWIMAGE这是一个debug选项，3）-5）都是对图片做一些变化，如果你想知道变化的效果，那么可以把这个参数设置为true，这样你就可以看到变化之后的效果。方便你更好的调整3）-5）这些参数。  
    
  目前试验结果  
  1、对CIFAR-10数据集进行了比较短时间的训练（没有对数据进行变化），测试准确率是81.37%，接近于https://code.google.com/p/cuda-convnet/ 的初步结果，这样一个试验结果已经足够说明代码的正确性了。  
   我最初的代码是参考http://eric-yuan.me/cnn3/，Eric加入了不少东西，但是针对CIFAR-10他只是得到了71%的正确率，我能够等到更高的正确率归功于CUDA加速，使得我可以设置规模更大的网络，仅此而已。  
   当然，在我的第三个大版本中，我会确保针对cifar-10数据集，我能够得到接近于所有公开结果中最好的实验结果。  
  2、针对mnist数据集，依然可以轻易的实现99%以上的正确率。  
    
  第3个版本的主要任务。  
  1、在实现第二个版本的时候，我fix了大量的bug，你要清楚，一个大型项目不可能没有bug的，只要他不影响工作，目前从试验效果看，第二个版本已经稳定了。  
  2、目前我的网络结构依然太单一了，第3个版本的核心任务就是加入如下两个特性：  
  1）参考Notes on Convolutional Neural Networks第3.3节，Learning Conbinations of Feature Maps。  
  2）参考ImageNet Classification with Deep Convolutional Neural Networks第3.3节，Local Response Normalization。  
  这两个特性是非常重要，可以非常显著提升数据集CIFAR-10的准确率，你会在第3个版本看到这两个特性，并且通过配置文件决定是否使用它们进行训练（因为针对mnist你并不需要这么复杂的特性，加入会降低运算效率）。