

# 面向 Windows 平台的深度学习工具

## 使用指南

张 林

同济大学软件学院

### 1. Windows 版 Caffe 的安装与基本测试

#### 1.1 Windows 版 Caffe 配置与编译

BVLC 有 caffe windows 的官方版本, <https://github.com/BVLC/caffe/tree/windows>

下面介绍安装这个版本的 windows caffe 的一些具体问题

关于 visual studio, VS2015 好像不是很稳定, 目前还是用 VS2013

安装 CMake 3.4

CUDA 用 CUDA 8.0 版本, 并且用与之配合的 cudnn

cudnn 解压以后, 形成一个 cuda 目录, 把这个目录下的三个文件夹 (bin, lib, include) 复制到 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0 目录之下, 会与该目录下本来的三个文件夹合并内容

注意: 当把显卡的驱动安装成功, 并且安装了 CUDA 之后, 建议执行一下硬件检测程序, 测试一下硬件是否正常。可以先编译 cuda 的 samples, 在 C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0 之下

编译后执行 C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0\bin\win64\Release 目录下的 deviceQuery.exe 来测试。如果正常的话, 会出现类似下图的信息。

```
命令提示符
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>cd C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0\bin\win64\Release

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0\bin\win64\Release>devicequery
devicequery Starting...

CUDA Device Query (Runtime API) version (CUDA static linking)
Detected 2 CUDA Capable device(s)

Device 0: "TITAN X (Pascal)"
  CUDA Driver Version / Runtime Version      8.0 / 8.0
  CUDA Capability Major/Minor version number: 6.1
  Total amount of global memory:              12288 MBytes (12884901888 bytes)
  (28) Multiprocessors, (128) CUDA Cores/MP: 3584 CUDA Cores
  GPU Max Clock rate:                        1531 MHz (1.53 GHz)
  Memory Clock rate:                          5005 Mhz
  Memory Bus Width:                           384-bit
  L2 Cache Size:                             3145728 bytes
  Maximum Texture Dimension Size (x,y,z)      1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:             65536 bytes
  Total amount of shared memory per block:    49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                  32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:        1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z):  (2147483647, 65535, 65535)
  Maximum memory pitch:                       2147483647 bytes
  Texture alignment:                           512 bytes
  Concurrent copy and kernel execution:       Yes with 2 copy engine(s)
  Run time limit on kernels:                   Yes
  Integrated GPU sharing Host Memory:          No
  Support host page-locked memory mapping:     Yes
  Alignment requirement for Surfaces:         Yes
  Device has ECC support:                      Disabled
  CUDA Device Driver Mode (TCC or WDDM):      WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):    Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 129 / 0
  Compute Mode:

```

如果使用 python, 需要安装 Anaconda Python 2.7

然后, 在开始栏里面, 运行 “Anaconda Prompt”, 安装几个组件

```
conda install --yes numpy scipy matplotlib scikit-image pip
pip install protobuf
```

需要根据自己的环境修改一下 scripts\build\_win.cmd 的内容。

```
MSVC_VERSION=12
WITH_NINJA=0
CPU_ONLY=0
PYTHON_VERSION=2
BUILD_MATLAB=1
RUN_TESTS=1
C:\ProgramData\Anaconda2
```

然后, 执行 build\_win.cmd 命令。

完成编译后, 会自动执行测试脚本; 可能会有 4 个关于 timer 的测试例程不能通过, 这个并没有关系。

## 1.2Matlab 接口配置

我们还可以测试一下 matlab 的配置能否正常。

在这一点上官方文档有些问题, 需要按照下面过程配置

需要把

```
F:\caffe-bvlc\matlab\+caffe\private\Release
```

下的 caffe\_.mexw64 拷贝到

```
F:\caffe-bvlc\matlab\+caffe\private
```

目录下。

需要把

```
F:\caffe-bvlc\scripts\build\Matlab\Release
```

下的 `caffe_.exp` 和 `caffe_.lib` 也拷贝到

```
F:\caffe-bvlc\matlab\+caffe\private
```

之下。

然后,可以运行 matlab 的一个分类 demo

```
F:\caffe-master\matlab\demo\classification_demo.m
```

运行这个 demo 之前还要下载一个 `bvlc_reference_caffenet.caffemodel`,

[https://github.com/BVLC/caffe/tree/master/models/bvlc\\_reference\\_caffenet](https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet)

`bvlc_reference_caffenet.caffemodel` 需要放在目录

```
F:\caffe-master\models\bvlc_reference_caffenet
```

之下

如果上述例子能通过运行,说明 windows 版本的 `caffe` 已经安装设置好。

### 1.3 直接调用 `classification.exe` 进行测试

随 `caffe` 一起自带的有一个编译好的分类程序 `classification.exe`, 它是命令行形式, 可以对测试图像进行分类。在 `F:\caffe-bvlc\scripts\build\examples\cpp_classification\Release` 目录之下。

我们假设训练是在 `imagenet` 上进行的, 共有 1000 个类别, 这 1000 个类别都有人工标注的语义类别信息, 给定 1 张测试图像, 我们可以看看它属于什么类别

```
classification.exe F:\caffe-bvlc\models\bvlc_reference_caffenet\deploy.prototxt  
F:\caffe-bvlc\models\bvlc_reference_caffenet\bvlc_reference_caffenet.caffemodel  
F:\caffe-bvlc\data\tmp\imagenet_mean.binaryproto F:\caffe-  
bvlc\data\tmp\synset_words.txt F:\caffe-bvlc\examples\images\cat.jpg
```

所需要的 `imagenet_mean.binaryproto` 和 `synset_words.txt` 可以从

<http://dl.caffe.berkeleyvision.org/>

下载 `caffe_ilsvrc12.tar.gz` 解压缩得到

## 2. 训练并测试字符分类模型 LeNet

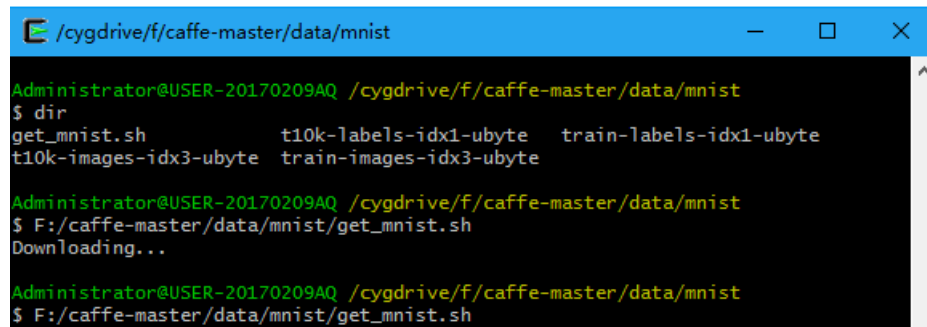
本节的基本素材来源于 <http://caffe.berkeleyvision.org/gathered/examples/mnist.html>

但这个 `caffe` 官方文档是面向 Linux 系统的, 我们这里做了一些改动, 可以在 windows `caffe` 上顺利通过。

## 2.1 准备数据

为了能够运行原来面向 linux 写的脚本程序，我们需要暗转 Cygwin。

准备 mnist 数据，需要执行 F:\caffe-bvlc\data\mnist 下面的 get\_mnist.sh 脚本  
在 Cygwin 里面执行 .sh 脚本时，一般要把整个脚本文件的路径写全，如下图所示



```
Administrator@USER-20170209AQ /cygdrive/f/caffe-master/data/mnist
$ dir
get_mnist.sh          t10k-labels-idx1-ubyte  train-labels-idx1-ubyte
t10k-images-idx3-ubyte train-images-idx3-ubyte

Administrator@USER-20170209AQ /cygdrive/f/caffe-master/data/mnist
$ F:/caffe-master/data/mnist/get_mnist.sh
Downloading...

Administrator@USER-20170209AQ /cygdrive/f/caffe-master/data/mnist
$ F:/caffe-master/data/mnist/get_mnist.sh
```

在 Cygwin 中输入命令式，路径里面要用斜杠/而不是\。

运行这个脚本，如果报没有找到 wget 的错误，说明你的电脑上没有安装 wget 这个程序，需要下载。可以从 <https://eternallybored.org/misc/wget/> 上下载。

可以下载 1.19.1 这个版本，如果是 64bit 系统，下载的 wget64.exe 要重命名成 wget.exe  
这个版本的 wget 是独立运行的文件，直接放到 C:\cygwin64\bin 下即可

## 2.2 准备 LMDB 格式数据

Caffe 训练支持的数据格式包括 levelDB 和 LMDB，在运行 MNIST 示例之前，首先要把下载的数据转换成 LMDB 格式。

这个需要在 cygwin 中执行 F:\caffe-bvlc\examples\mnist\create\_mnist.sh 脚本  
运行前，必须要根据当前机器环境适当修改脚本中涉及到路径的内容  
运行后，会生成 mnist\_test\_lmdb 和 mnist\_train\_lmdb 两个文件夹

## 2.3 执行 train\_lenet.sh

当数据都准备好了以后，最后执行训练，运行

F:\caffe-master\examples\mnist\train\_lenet.sh

当然，也需要修改这个脚本，以及对应的 lenet\_solver.prototxt 中的路径

它会每隔 5000 次输出一个训练结果模型，存放位置可以修改 lenet\_solver.prototxt 文件中 snapshot\_prefix 项

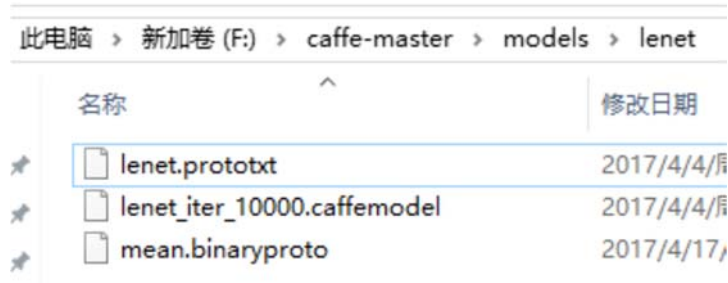
还需要注意的是 test\_iter 的设置是 validation sample 的总数/validation batchsize

训练结束后，会得到 lenet\_iter\_10000.caffemodel，这就是训练最终得到的 model

## 2.4 对自己提供的手写字符图片进行分类

这相当于是测试阶段

为了达到这个目的，我们需要如下文件：



并把他们放到 `caffe-master\models\lenet` 文件夹下

由于我们每次只测试一张图片，`Lenet.prototxt` 的内容

```
input_param { shape: { dim: 64 dim: 1 dim: 28 dim: 28 } }
```

需要改成

```
input_param { shape: { dim: 1 dim: 1 dim: 28 dim: 28 } }
```

然后，需要计算训练样本的均值，按照如下方式进行

```
compute_image_mean.exe      F:/caffe-bvlc/examples/mnist/mnist_train_lmdb  
F:/caffe-bvlc/models/lenet/mean.binaryproto
```

都准备好以后，把课程主页上的“Digit classification demo”解压缩并放到 `F:\caffe-master\matlab\demo` 目录下，运行即可。

## 3. 在 CIFAR-10 数据集上的模型训练与测试

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

More information of this dataset can be found at

<http://www.cs.toronto.edu/~kriz/cifar.html>

本节内容主要参考

<http://caffe.berkeleyvision.org/gathered/examples/cifar10.html>

### 3.1 准备数据

在 `cygwin` 内运行 `F:/caffe-master/data/cifar10/get_cifar10.sh` 会下载数据集

然后执行

```
F:/caffe-master/examples/cifar10/create_cifar10.sh
```

注意：执行这个脚本之前要适当修改路径才能保证正常运行。

执行完毕后，它会把 `cifar10` 数据转换成 `caffe` 使用的数据格式，是 `train_lmdb` 和 `test_lmdb`

## 3.2 执行训练过程

在 `cygwin` 内运行 `train_quick.sh`

注意：为了保证这个训练脚本能够正常运行，务必要记得修改里面涉及到的 `path`。简单起见，都改成绝对路径比较保险。






另外，`caffe` 自带提供的 `cifar10_quick_solver.prototxt` 配置文件中，会把训练得到的 `caffemodel` 再转换成 `HDF5` 格式，但这个格式的 `caffemodel` 在 `matlab` 测试下无法解析，在初始化网络阶段 `matlab` 会 `crash`，所以不要把输出 `caffemodel` 保存成 `HDF5` 格式。这需要注意掉 `cifar10_quick_solver.prototxt` (还有 `cifar10_quick_solver_lr1.prototxt`) 配置文件中如下一行：

```
#snapshot_format: HDF5
```

训练结束后，会在 `F:\caffe-master\examples\cifar10` 生成 `cifar10_quick_iter_5000.caffemodel` 文件。这就是训练的结果模型。

## 3.3 命令和测试训练结果

`Cifar10` 是个 10 类的分类器，可以从网上找一些测试图片测试一下。测试时候，可以用 `classification.exe` 来进行。执行 `classification` 所需要的参数如下图所示，其中 `dog.jpg` 是一幅待测试的图片。

脑 > 新加卷 (F:) > caffe-master > examples > cifar10 > cifartest		
名称	修改日期	类型
 cifar10_quick_iter_5000.caffemodel	2017/4/20/周四 ...	CAFFE
 deploy.prototxt	2016/12/7/周三 ...	PROT
 dog.jpg	2017/4/20/周四 ...	JPG 文
 mean.binaryproto	2017/4/19/周三 ...	BINAI
 synset_words.txt	2017/4/20/周四 ...	文本文

在 `cmd` 中执行：

```
classification.exe F:/caffe-master/examples/cifar10/cifartest/deploy.prototxt
F:/caffe-master/examples/cifar10/cifartest/cifar10_quick_iter_5000.caffemodel
F:/caffe-master/examples/cifar10/cifartest/mean.binaryproto F:/caffe-
master/examples/cifar10/cifartest/synset_words.txt F:/caffe-
master/examples/cifar10/cifartest/dog.jpg
```

输出结果

```

----- Prediction for F:/caffe-master/examples/cifar10/cifartest/dog.jpg
0.9994 - "dog"
0.0003 - "frog"
0.0001 - "deer"
0.0001 - "cat"
0.0001 - "bird"

F:\caffe-master\Build\x64\Release>_

```

注意：由于训练阶段网络权重的初始化存在随机性，你得到的训练模型可能和我的不同，导致最终测试结果也可能有一些差异。

### 3.4 在 matlab 中进行测试

训练好 cifar10 网络以后，可以在 matlab 中进行测试代码的编写。我已经把示例 matlab 代码放在课程网站上面了。下载 cifar10 test demo.zip，解压后，放到

F:\caffe-master\matlab\demo

目录下，运行即可。

## 4. 用 CAFFE 训练自己的分类数据

本节主要讲述如何针对自己的训练数据训练并测试一下分类模型。本节所需相关文件，请下载 Self Data Training.ZIP。

对于自己数据的分类模型训练基本流程与 imagenet 的训练很类似。

### 4.1 准备图像数据以及类标文件

准备好训练和验证的图像数据集，最好是每个类别的图像放在一个文件夹里面。在这个例子中，我们使用了网上大侠帮忙收集的 tiny 数据集，这个数据集包含了 5 个类别；每个类别有 80 个训练图像，20 个验证图像。

训练集对应的类别目录如下图

电脑 > 新加卷 (F:) > caffe-master > data > forCVClassify > imgs > train >		
名称	修改日期	类型
bus	2017/1/5/周四 1...	文件夹
dinosaur	2017/1/5/周四 1...	文件夹
elephant	2017/1/5/周四 1...	文件夹
flower	2017/1/5/周四 1...	文件夹
horse	2017/1/5/周四 1...	文件夹

然后需要准备训练 label 文件 这个文件的每一行是图片相对于”training”目录的路径+空格+类别；比如，其中的一行记录形如：

bus/320.jpg 0

类似地，需要准备验证数据的 label 文件

我们为此准备了一个 matlab 小程序 creat\_train\_val\_list.m，可以用来生成 train.txt 和

val.txt

需要注意的是：类别编号是从 0 开始的。

## 4.2 生成 lmdb 数据

当图像和类标文件都准备好以后，需要生成 caffe 使用的 lmdb 文件。

在 cygwin 中运行

```
F:\caffe-master\examples\forCVClassify\create_imagenet.sh
```

注意：这个文件是从 examples\imagenet 目录下面拷过来的，需要做一些相应的修改。

```
Resize=true;
```

```
RESIZE_HEIGHT=227
```

```
RESIZE_WIDTH=227 %因为我们要使用的网络是 alexnet，对图像的输入有要求，是 227*227
```

运行完后，在 examples\forCVClassify 下生成了 train\_lmdb 和 val\_lmdb

之后，在 cygwin 中运行

```
F: /caffe-master/examples/forCVClassify/make_imagenet_mean.sh
```

生成均值文件

## 4.3 配置并训练网络

把 train\_val.prototxt 里面 test 部分的 batchsize 改成 20；最后一层 fc8 的 num\_output 改成 5，因为我们只有 5 个类别。

把 solver.prototxt 里面的 test\_iter 设置成 5，因为一共 100 个测试样本，每个 batch 是 20 个，5 次迭代可以覆盖全部测试样本；base\_lr 可以根据训练情况调节。

都设置好后，执行

```
F: /caffe-master/examples/forCVClassify/train_caffenet.sh
```

训练网络。

## 4.4 关于 fine-tuning

如果使用分类使用的网络是学术界已经公开的网络，我们只是在最后一层分类节点数改变了，我们不要在自己的数据上 training from scratch，而需要使用 fine-tuning 技术。也就是使用已经在 imagenet 上训练好的网络参数作为初始化，然后用我们的数据进行 fine tune。

参考如下语法：

```
F:/caffe-master/Build/x64/Release/caffe.exe train \  
    --solver=F:/caffe-  
master/examples/forCVClassify/myModel/solver_finetune.prototxt \  
    --weights=F:/caffe-  
master/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel $@
```

由于我们的类别数与 imagenet 不同，因此我们的网络最后一层的输出节点数需要更改，也就是这个全连接层的参数不能从 bvlc\_reference\_caffenet.caffemodel 拷贝过来；我们需要把网络中这



个全连接层的名字改一下，比如 `fc8`→`fc8_cv`；`caffe` 拷贝权重的策略是根据层名来进行的，这层的名字与 `alexnet` 中不同，它就会按照某种随机初始化的办法初始化这层的参数。

在 `fine-tune` 时，我们一般需要调低基础学习率，这是在 `solver.prototxt` 中 `base_lr` 设置的；而同时，需要调高最后一层的 `lr_mult` 参数，这个参数的意思是这一层的学习率是 `base_lr` 的 `lr_mult` 倍，这样我们可以加快最后一层的学习，加快收敛。

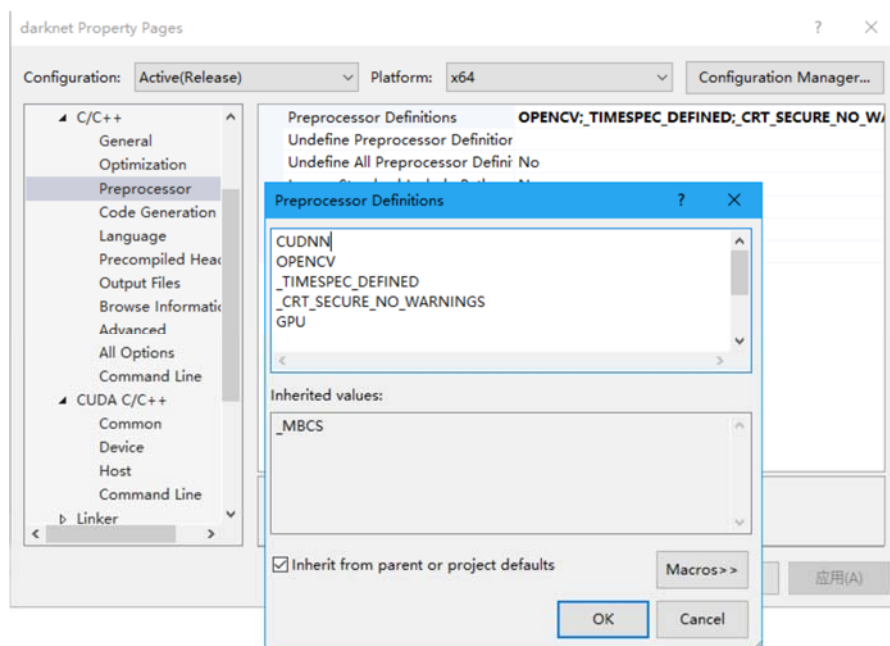
如果我们要冻结前面所有层的参数学习，只更新后面某几层的参数，可以把前面那些层对应的 `lr_mult` 都设成 0 即可。

通过这个例子，我们可以看到使用了 `fine-tune` 策略以后，确实可以非常快的收敛。

需要再提一下的是：其实使用 `fine-tune` 策略也不一定是我们只修改了网络的最后一层；如果我们的图像输入维数不同，需要精简，也完全可以使用 `fine-tune` 策略；我们需要以 `AlexNet` 为蓝本修改网络，网络的前几层由于输入图像的维度不同自然会与 `AlexNet` 不一样，我们尽量做到使得后面从某一层开始与 `AlexNet` 一样；这样，前面与 `Alexnet` 不一样的层，需要把层名字换一下，不要与 `Alexnet` 对应层名相同，那么在 `fine-tune` 的时候，这些层的参数就会被某种策略随机初始化；而后面与 `Alexnet` 同名的层就会被 `pre-trained` 的 `Alexnet weight` 复制初始化。

## 5.Windows 版 Yolo V2

- 1) 可以从 <https://github.com/AlexeyAB/darknet> 下载
- 2) 作者提供的是 VS2015 的工程文件，但我自己不能成功复现，还是用 VS2013。配置环境 VS2013+CUDA 8.0 + CUDNN + OpenCV 2.4.9.
- 3) 需要修改配置文件，主要包括
  - Platform toolset
  - Opencv 头文件的包含路径
  - Opencv 库文件的包含路径
  - 还有需要更改 GPU 计算能力，Nvidia Titan X 是  
<CodeGeneration>compute\_61,sm\_61</CodeGeneration>
- 4) 如果需要使用 cudnn 加速的话，需要按如下操作配置
  - 需要增加一个 windows 系统环境变量 cudnn，它的值是 CUDNN 解压之后的路径  
F:\DL related softwares\cudnn-8.0-windows10-x64-v5.1\cuda
  - 然后，进行如下设置，增加一个 preprocessor definition



- 5) 安装了 Python 之后，执行 python 命令需要在 cmd 提示符下进行；当代码中用了相对路径时，要先在 cmd 中转移到合适的根目录之下
- 6) Yolo 的网络配置文件最后一行 random=0, 如果设置成 0, 就表示不用多尺度训练模式；如果设置成 1, 就采用多尺度训练模式。
- 7) 如果在某种应用中运行速度非常重要，可以训练 tiny-yolo，它是普通 yolo 的一个简化版本，tiny-yolo 使用了更少的卷积层。在训练 tiny-yolo 时，也需要初始化的已经训练好的 weight，这个初始化 weight 也可以使用训练普通 yolo 时所用的 darknet19\_448.conv.23 这个文件。

## Miscellaneous

- (1) cmd 窗口中的清屏命令：cls
- (2) Caffe 网络中，提到的 TEST, 实际上指的是 validation ,这样, 每 1000 次输出的 accuracy 实际上是 validation accuracy
- (3) 如果系统中装有多个 GPU，我们希望在某指定的 GPU 上运行，可以执行  
`caffe train -solver examples/mnist/lenet_solver.prototxt -gpu 1`  
 注意：系统中 GPU 的编号是从 0 开始的，第二个 GPU 就是 -gpu 1
- (4) 要运行 Caffe 关于 Matlab 的 demo classification\_demo，需要下载 BVLC CaffeNet，地址如下  
[http://dl.caffe.berkeleyvision.org/bvlc\\_reference\\_caffenet.caffemodel](http://dl.caffe.berkeleyvision.org/bvlc_reference_caffenet.caffemodel)
- (5) Caffe 工具里面有几中不同类型的文件，  
 Lenet\_train\_test.prototxt 定义的是 CNN 的网络结构  
 lenet\_solver.prototxt 定义的是训练过程中使用的一些参数设置，比如 learning rate、iteration、GPU or CPU、隔多长时间保存一次结果、结果保存的路径等等；  
 solver.prototxt 是要被 caffe.exe 调用的，而 solver.prototxt 中会指定网络结构文件。

lenet\_iter\_10000.caffemodel 保存的是 CNN 里面网络参数

lenet\_iter\_10000.solverstate 保存的是当前网络的训练状态，比如每一层上的梯度、learning rate 等；主要用于在后续 resume training 的时候使用，可以在此基础上继续训练。

(6) 在训练过程中，每隔一定的迭代次数，会输出 loss，这个 loss 是 training loss；输出的 test score #0 是 validation 集合上的 accuracy；test score #1 是 validation 集合上的 loss

(7) 训练策略技巧：训练 10000 次，保存一个 snapshot，然后 resume training from this snapshot，注意此时把 solver.prototxt 中的 lr 除以 10。

(8) 生成 levelDB 格式文件

```
convert_imageset.exe -backend=leveldb -resize_height=256 -resize_width=256 -
shuffle
F:\caffe-
windows\Build\x64\Release\examples\selfPractice\image1000test200\val\
F:\caffe-
windows\Build\x64\Release\examples\selfPractice\image1000test200\val.txt
F:\caffe-
windows\Build\x64\Release\examples\selfPractice\selfPractice_val_ldb
```

(9) 计算训练集均值文件

```
compute_image_mean.exe --backend leveldb F:\caffe-
windows\Build\x64\Release\examples\selfPractice\selfPractice_train_ldb
F:\caffe-
windows\Build\x64\Release\examples\selfPractice\train_mean.binaryproto
```

(10) 为什么在 caffe 的文档里面，如果用 alexnet 的时候，会说把图像缩放到 256\*256，而 alexnet 对输入的要求不是 227\*227\*3 的图像吗？

这是因为按照 alex 论文所说的，可以做 data augmentation，具体做法是从 256\*256 的图像上随机取 2048 个 227\*227 的图像块，然后还要进行镜像翻转；这步 data augmentation 操作可以在 prototxt 里面指定，caffe 看到之后会自动去做，我们一般是不需要这个功能的。

```
layer {
  name: "data"
  type: "Data"
  [...]
  transform_param {
    scale: 0.1
    mean_file_size: mean.binaryproto
    # for images in particular horizontal mirroring and random cropping
    # can be done as simple data augmentations.
    mirror: 1 # 1 = on, 0 = off
    # crop a `crop_size` x `crop_size` patch:
    # - at random during training
    # - from the center during testing
    crop_size: 227
  }
}
```

(11) 利用 Caffe，如何在 validation set 上计算分类正确率？

```
caffe.exe          test          -model          F:\caffe-bvlc\examples\entrance-  
line\myModel_GoogleNet\train_val.prototxt          -weights          F:\caffe-  
bvlc\examples\entrance-  
line\myModel_GoogleNet\googlenet_iter_56000.caffemodel -gpu 0 -iterations 501
```

(12)如何利用 caffe 的 matlab 接口知道某一层数据的维数？

```
data = net.blobs('conv1').get_data();
```

(13)为什么我的图片明明是 8-bit 灰度图像，而用 convert\_imageset.exe 转换成 LMDB 的时候，还是按照彩色图像来处理（3 个通道完全一样），这样，在计算均值文件的时候也是 3 个通道？

convert\_imageset 这个工具默认把图像看作 RGB 3 个通道的图像；如果需要显式告诉它我要用的是单通道的灰度图像，需要设置参数--gray=true

另外，convert\_imageset 还有一个参数 shuffle 可以用来把数据集的个体出现的顺序打乱；这是因为在标注图像的时候往往属于某一个类的图像是放在一起的，在训练网络的时候，属于某一个 mini-batch 的图像很可能很像，这不利于优化的进行，我们希望每个 mini-batch 的样本多样性大一些，可以设置--shuffle=true