使用說明

架設環境

預設安裝環境：windows 10

Python2.7 安裝

到這網站下載 python 2.7

https://conda.io/miniconda.html

運行 caffe 會需要安裝的 module

Cython>=0.19.2

numpy>=1.7.1

scipy>=0.13.2

scikit-image>=0.9.3

matplotlib>=1.3.1

ipython>=3.0.0

h5py>=2.2.0

leveldb>=0.191

networkx>=1.8.1

nose>=1.3.0

pandas>=0.12.0

python-dateutil>=1.4,<2

protobuf>=2.5.0

python-gflags>=2.0

pyyaml>=3.10

Pillow>=2.3.0

six>=1.1.0

安裝其他會使用到的 python module：

$pip install pydicom

$pip install scikit-learn

$conda install -c https://conda.binstar.org/menpo opencv

Caffe 安裝

如何在 windows 10 pro +VS2013 pro update5 eng +CUDA 8.0 +cudnn v5.1RC 下安裝完 Caffe ?

待安裝與下載的軟體：

- caffe 用的版本是微軟在維護的 : https://github.com/Microsoft/caffe
- VS2013 pro update 5 eng : https://www.microsoft.com/zh-tw/download/details.aspx?id=48129
- Git for windows ：https://git-scm.com/download/win
- TotorisGit https://tortoisegit.org
- CUDA 8(根據顯卡相容版本選取) : https://developer.nvidia.com/cuda-toolkit

ps: 安裝 CUDA 時可能冒出本機 driver 不符，但可以不用理他，只要確認顯卡的 driver 是網路上載的最新版就好。然後建議 CUDA 用 custom 安裝，不要點選 CUDA 附的 driver，因為它的會舊一些可能造成相容性問題。

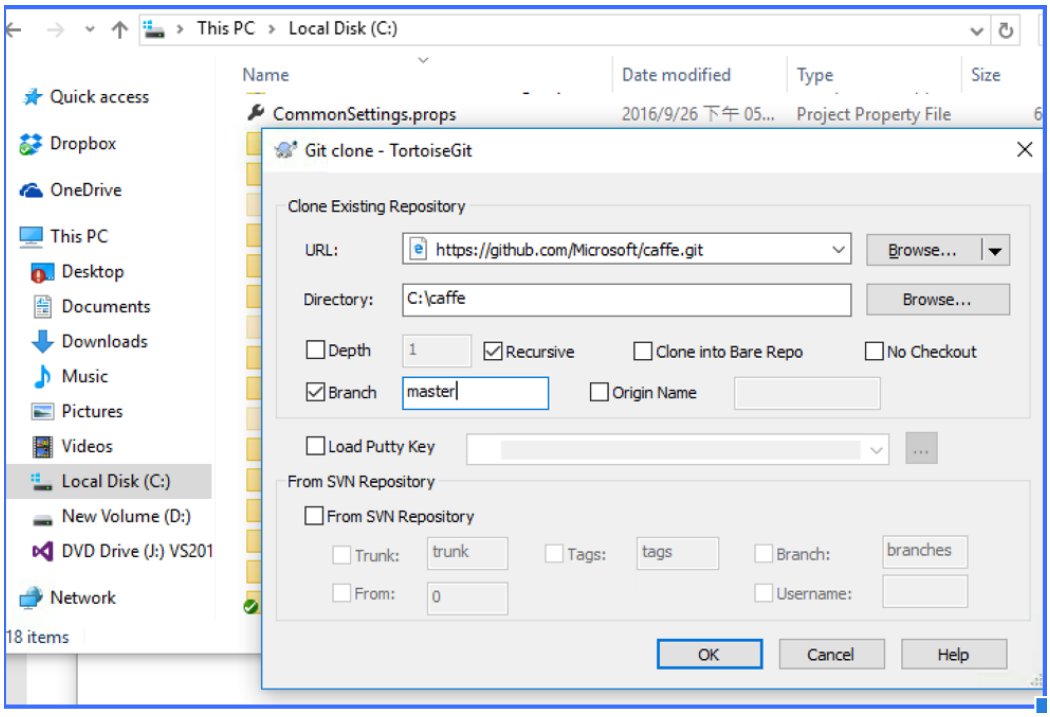
- cuDNN 5.1(根據顯卡相容版本選取)：https://developer.nvidia.com/cudnn
  解壓縮後裡頭會含有 cuda 這個資料夾，看要複製到哪放，我是放在 caffe 母目錄底下。

安裝步驟：

1. 使用小烏龜 TotorisGit 從 windows 維護的 clone 檔案內容

由於小烏龜是建立在 Git 上，所以要先裝好 Git for windows 才行

在要成立 caffe 的位置，點下右鍵然後點選 git clone，把微軟維護頁面的 git clone url 貼在這空格中，點 Recursive 與 Branch 到 master。



2. 設定 CommonSettings.props 檔

-進入 caffe，再到 windows 資料夾下， 複製 CommonSettings.props.example ，並改名為 CommonSettings.props

以下是不同模式需要變更的部份

只要跑 CPU

<CpuOnlyBuild>true</CpuOnlyBuild>

<UseCuDNN>false</UseCuDNN>

除了跑 CPU 還想跑 GPU

<CpuOnlyBuild>false</CpuOnlyBuild>

<UseCuDNN>true</UseCuDNN>

<CudaVersion>填入 **CUDA 的版本**</CudaVersion>

<CudaArchitecture>填入顯卡的架構<CudaArchitecture>

example: 我的 GTX1060 根據連結查到是 6.1

-><CudaArchitecture>compute_61,sm61<CudaArchitecture>

<CuDnnPath>你的 cudnn 資料夾的母目錄</CuDnnPath>

查詢顯卡的架構：https://en.wikipedia.org/wiki/CUDA

GPU	Compute Capability
GTX660, 680, 760, 770	compute_30,sm_30
GTX780, Titan Z, Titan Black, K20, K40	compute_35,sm_35
GTX960, 980, Titan X	compute_52,sm_52

(其餘 Python 跟 matlab 的功能就要參考：
http://blog.csdn.net/happynear/article/details/45372231)

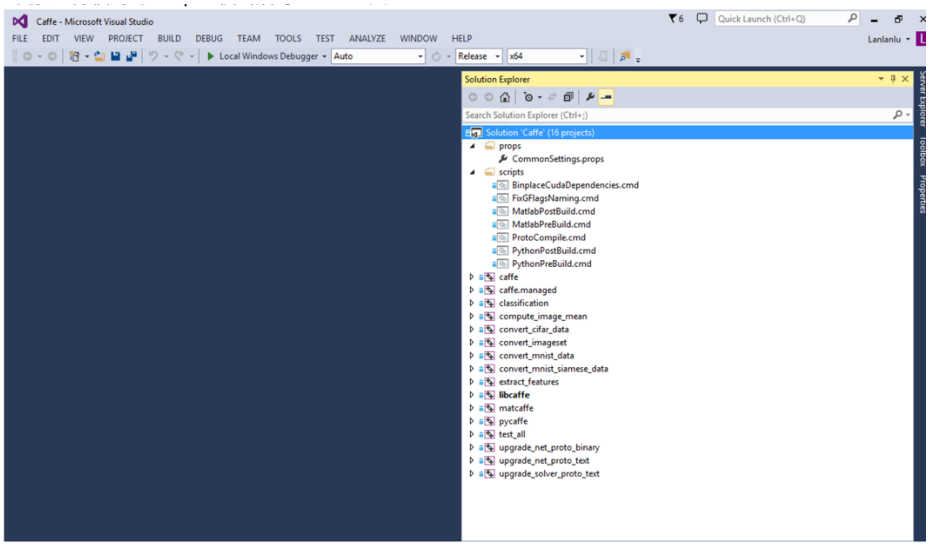
new 1 | create_cifar10.sh | create_mnist.sh | lenet_solver.prototxt | CommonSettings.props

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <ImportGroup Label="PropertySheets" />
    <PropertyGroup Label="UserMacros">
        <BuildDir>$(SolutionDir)..\Build</BuildDir>
        <!--NOTE: CpuOnlyBuild and UseCuDNN flags can't be set at the same time.-->
        <CpuOnlyBuild>false</CpuOnlyBuild>
        <UseCuDNN>true</UseCuDNN>
        <CudaVersion>8.0</CudaVersion>
        <!-- NOTE: If Python support is enabled, PythonDir (below) needs to be
         set to the root of your Python installation. If your Python installation
         does not contain debug libraries, debug build will not work. -->
        <PythonSupport>false</PythonSupport>
        <!-- NOTE: If Matlab support is enabled, MatlabDir (below) needs to be
         set to the root of your Matlab installation. -->
        <MatlabSupport>false</MatlabSupport>
        <CudaDependencies></CudaDependencies>

        <!-- Set CUDA architecture suitable for your GPU.
         Setting proper architecture is important to mimize your run and compile time. -->
        <CudaArchitecture>compute_61,sm_61</CudaArchitecture>

        <!-- CuDNN 3 and 4 are supported -->
        <CuDnnPath>C:\caffe</CuDnnPath>
        <ScriptsDir>$(SolutionDir)\scripts</ScriptsDir>
    </PropertyGroup>
    <PropertyGroup Condition="'$(CpuOnlyBuild)'=='false'">
        <CudaDependencies>cublas.lib;cuda.lib;curand.lib;cudart.lib</CudaDependencies>
    </PropertyGroup>

    <PropertyGroup Condition="'$(UseCuDNN)'=='true'">
        <CudaDependencies>cudnn.lib;$(CudaDependencies)</CudaDependencies>
    </PropertyGroup>
    <PropertyGroup Condition="'$(UseCuDNN)'=='true' And $(CuDnnPath)!=''">
        <LibraryPath>$(CuDnnPath)\cuda\lib\x64;$(LibraryPath)</LibraryPath>
        <IncludePath>$(CuDnnPath)\cuda\include;$(IncludePath)</IncludePath>
    </PropertyGroup>
```

3. 用 VS2013 compile

進入 caffe\windows ， 用 VS 開啟 Caffe.sln

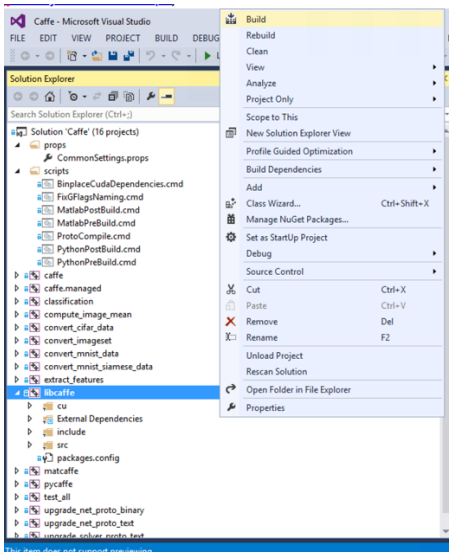
先把上方橫列的 compile 模式設為 Release 與 x64



先點選 libcaffe ， 按右鍵後點 build(建置) 。

(可能會遇到一些要變更某些檔案為 Unicode 的問題，解決方式參考：

https://msdn.microsoft.com/zh- tw/library/ms173715.aspx)



-沒有 error 後再對整個 caffe 專案點選，然後點選右鍵後按 build(建置)。

成功後他會在 caffe 母目錄下，產生一個叫 Build 的資料夾。

在 Build\x64\Release 底下會有 caffe.exe 檔之後會用來訓練的執行檔，以及 moist 跟 cifar 等案例會用來轉換 case 的 exe 檔。

以上結束後，就安裝完可以用 GPU 與 CPU 跑 Caffe 的環境了

caffe 安裝的參考:

http://blog.csdn.net/chenyj92/article/details/51372646 或

是 https://github.com/BVLC/caffe/tree/windows

使用 GPU 的安裝參考 http://zongwei.leanote.com/post/Windows-7

caffe+VS2013+Windows+GPU 配置+cifar 使用 http://blog.csdn.net/zb1165048017/article/details/51549105

操作

$論文程式/program/code

1. **convertdicomtotiff_crop.py: 把 dicom 檔轉成 tiff 檔，去掉非 EBUS 影像區域**
   source image path: $論文程式/test_data/EBUSimages_programuse/
   output image path: $論文程式/test_data/EBUSimages_converttotiff/
2. **classify_BM.py: 把 tiff 檔從各個病，分成 Benign 跟 Malignant 兩個分類而已**
   source image path: $論文程式/test_data/EBUSimages_converttotiff/
   output image path: $論文程式/test_data/EBUSimages_BM/
3. **cross_validate_produce.py: 把影像分成 5 個 fold，test0~4 跟 train0~4**
   source image path: $論文程式/test_data/EBUSimages_BM/
   output image path: $論文程式/test_data/EBUSimagestraining_test/
4. **trainset_flip.py: 把各個 fold 的 training set 做 flip**
   source image path: $論文程式/test_data/EBUSimagestraining_test/
   output image path: $論文程式/test_data/EBUSimagestraining_test/
5. **trainset_rotate_Mless.py: 接著再把各個 fold 的 training set 做旋轉**
   source image path: $論文程式/test_data/EBUSimagestraining_test/
   output image path: $論文程式/test_data/EBUSimagestraining_test/

註：code 中的 path 位置需再以實際位置做調整，目前是以此交接資料夾做示範
而影像輸出的結果在 $論文程式/test_data
----------以上的步驟只需要做一次即可----------

6. **do_training_script.sh: 產出 training 用的 lmdb 檔，並跑每個 fold 的 training，但每跑一個 fold 須改 fold 資料夾名稱**

裡頭有兩列 command ，第一組是計算 training set 的 mean 值
C:/caffe/Build/x64/Release/compute_image_mean.exe -backend=lmdb $論文程式/program/input/train_lmdb $論文程式/program/input/mean.binaryproto
C:/caffe/Build/x64/Release/caffe.exe train --solver

跑 finetunning 的 command

$論文程式/ program/caffe_models/caffenet/solver_2.prototxt --weights $論文程式/program/pretrained_models/caffenet/bvlc_reference_caffenet.caffemodel 2>&1 | tee $論文程式/ program/caffe_models/caffenet/model_2_train_trasfer.log

註：這邊以 caffenet 作為示範

1. 跑 finetunning 的 command 中--weghts 之前放的是各個 model 的設定檔 **$論文程式/program/caffe_models 中有其他 model 的設定檔，在依所需變換**

2. 跑 finetunning 的 command 中--weghts 之後放的是前人以 imagenet 來 train 好的 pretrained model。在 **$論文程式/program/pretrained_models** 中有 caffenet/googlenet/vggnet16/resnet50 四種 pretrained model
3. 每種 model 輸入的影像 size 有些是 227x227 有些是 224x224。若遇到是 224x224 須自行把 script 中的 create_lmdb_227.py 換成 create_lmdb_224.py

7. caffenet_result.py/ googlenet_result.py/vgg16_result.py/resnet50.py 各是使用 finetune 好的 model 來測試判斷 test set 的效果。

   -需輸入 model 跑的 iteration
   -需輸入跑第幾個 fold
   -裡頭的 path 可能需要再調整
   -跑出的結果除了單純以 fintune 好的 model，還有 cnn+svm/cnn+random forest 的測試

   註：caffenet_result_combinedglcm.py 是有合併 glcmfeature 下去做 cnn+svm 跟 cnn+random forest