

Advanced AI Software and Toolkits

Table of Contents

System Configuration.....	1
Operating System.....	1
Hardware Platform.....	1
Software (installed in the indicated order).....	2
Cuda SDK.....	2
OpenCV.....	3
cuDNN.....	3
Caffe.....	4
DIGITS.....	5
Topics.....	7
Generate a Model Classifier using DIGITS.....	7
Classify an image from the command line using a model created in DIGITS.....	7
Using DIGITS scripts.....	7
Using Caffe Script.....	9
Use a pre-trained model for image classification in DIGITS.....	11
Use a pre-trained model to improve accuracy when training on a totally different data set.....	15
Identify objects in an image and draw bounding boxes around them.....	19
Fast R-CNN: Fast Region-based Convolutional Networks for object detection.....	19
Object Detection (with bounding box determination) using DIGITS.....	31
DIGITS with fast-rcnn ?.....	33
Compile and run the Nvidia CUDA examples.....	37
NSIGHT - NVidia Eclipse based IDE.....	39
Linking g++ code with CUDA libraries.....	41

System Configuration

Operating System

- Ubuntu 14.04

Hardware Platform

- AMD Phenom II 1055t CPU (6 cores)
- Nvidia GTX 980 GPU (2048 cores)
 - note: cuDNN requires Maxwell level or better graphics card

- Nvidia driver version: 352.68
- 8 GB RAM
- 466 GB Hard Drive (dual partitioned with Windows 10)

Software (installed in the indicated order)

Cuda SDK

1. Register as an Nvidia developer
 - To sign up, go to the Nvidia developer home page and create an account
 - Note: It takes 2-3 days to get an acknowledgment back from NVidia and obtain download and other privileges
2. Download Cuda 7.5 toolkit

download site: <https://developer.nvidia.com/accelerated-computing-toolkit>

 - Press Cuda Toolkit Download
 - Press “Linux” in target platform panel
 - Press x86_64 in “Architecture” options
 - Press Ubuntu in distribution options
 - Press 14.04 in version options
 - Press “deb (network)” in installer type options
 - Press “Download (2.1 KB)” in “Target Download Installer .. Panel”
 - Choose a download directory (e.g. ~/Downloads)
 - \$ cd ~/Downloads
3. installation
 - \$ sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb
 - \$ sudo apt-get update
 - \$ sudo apt-get install cuda
4. Setup environment
 - Add to ~/.bashrc

```
export CUDA_HOME=/usr/local/cuda-7.5
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64
PATH=${CUDA_HOME}/bin:${PATH}
export PATH
```

OpenCV

1. Obtain source code
 - \$ git clone <https://github.com/Itseez/opencv>
 - cd opencv
 - git checkout tags/3.1.0 -b 3.1.0
 - note: master branch is actually at 2.4 which produces compile errors unless CUDA is disabled
2. configure
 - mkdir build && cd build
 - cmake ..
 - select options
 - YES: WITH_CUDA, WITH_OPENGL, WITH_MP ..
 - press c (configure)
 - press g (generate)
3. build
 - make -j6 (takes ~ 1hr to build)
4. install
 - sudo make install

cuDNN

- Version 3.0
 - <https://developer.nvidia.com/rdp/cudnn-download>
 - create a download directory for cuDNN (e.g. ~/cuDNN)

- accept licence agreement (check box) and select cuDNN v3 for Linux ,guides, code examples and download into cuDNN directory
- untar: cudnn-7.0-linux-x64-v3.0-prod.tgz, cudnn-sample-v3.tgz
 - creates directories cuda and mnistCUDNN (sample)
- Copy libraries and header files to global locations


```
$ cd ~/cuDNN/cuda
```

```
$ sudo cp include/cudnn.h /usr/local/cuda-7.5/include
```

```
$ sudo cp lib64/libcudnn* /usr/local/cuda-7.5/lib /usr/local/cuda-7.5/lib64
```
- Version 5.1
 - <https://developer.nvidia.com/rdp/cudnn-download>
 - mkdir ~/cuDNN-5.1 && cd cuDNN-5.1
 - Download: [cuDNN v5.1 Library for Linux](#)
 - untar: cudnn-7.5-linux-x64-v5.1-rc.tgz


```
sudo cp include/cudnn.h /usr/local/cuda-7.5/include
```

```
sudo cp lib64/libcudnn* /usr/local/cuda-7.5/lib64
```

Caffe

- Version (compatible with cuDNN 5.1 and OpenCV 3.1)
- references
 - <http://caffe.berkeleyvision.org/installation.html>
- Obtain source code


```
$ cd
```

```
$ git clone https://github.com/BVLC/caffe
```

```
$ cd ~/caffe
```
- Get dependencies


```
$ for req in $(cat requirements.txt); do pip install $req; done
```
- Set up for Build


```
$ cp Makefile.config.example Makefile.config
```

 - Edit Makefile.config
 - Uncomment: USE_CUDNN := 1
 - Uncomment: WITH_PYTHON_LAYER := 1
 - Uncomment: OPENCV_VERSION := 3
- Build


```
$ make all
```

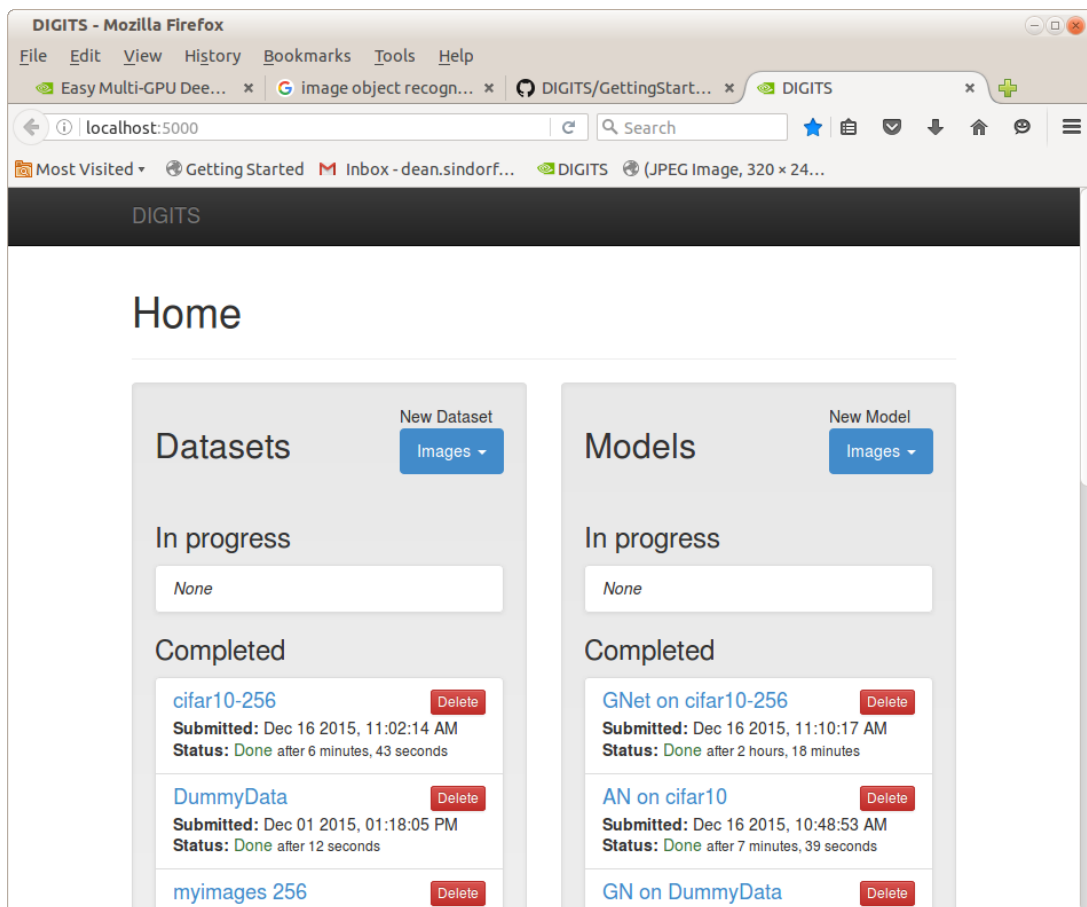
```
$ make pycaffe
```

```
$ make test
$ make runtest
```

DIGITS

- Version 2.0
 - references
 - <https://developer.nvidia.com/digits>
 - <https://github.com/NVIDIA/DIGITS/blob/digits-2.0/docs/GettingStarted.md>
 - Installation
 - Download Digits (version 2) from NVidia Developers website
 - untar into home directory
 - creates digits-version directory (e.g. ~/digits-2.0)
 - cd digits directory
 - run install script
 - ./install.sh
 - Running
 - in digits directory exec “runme.sh”
 - In web browser set URL to <http://localhost:5000/>
- Version 3.0
 - Installation
 - follow install instructions for 3.0:
<https://github.com/NVIDIA/DIGITS/blob/digits-3.0/docs/UbuntuInstall.md#repository-access>
 - Running
 - start server: `sudo start nvidia-digits-server`
 - stop server: `sudo stop nvidia-digits-server`
 - In browser change port from 5000 to 80
 - <http://localhost:80>

- Version 4.0 (released Aug. 2016)
 - Installation
 - Log in as NVIDIA developer
 - Go to downloads tab
 - follow instructions for installing DIGITS 4.0:
<https://github.com/NVIDIA/DIGITS/blob/digits-4.0/docs/UbuntuInstall.md#repository-access>
 - Running
 - See “Getting Started” page at: <https://github.com/NVIDIA/DIGITS/blob/digits-4.0/docs/GettingStarted.md>



Topics

Generate a Model Classifier using DIGITS

Follow the examples in the “Getting Started” link at: <https://github.com/NVIDIA/DIGITS/blob/digits-4.0/docs/GettingStarted.md>

1. Create a Classifier to recognize hand-written numbers
 - mnist
2. Create a Classifier to recognize common objects (e.g. dogs,cats,boats) in small images
 - cifar10

Classify an image from the command line using a model created in DIGITS

The Dataset and model used in this test was obtained from a set of images of a “ball” or “no ball” generated from a camera sensor in Gazebo while driving a simulated robot around in “teleop” mode (see ImageProcessing.odt:page-76 located in the same Team159 github/MentorRepository directory as this document for further details)

Goals:

1. After training a model using DIGITS use command line instructions or a shell script to identify objects in single images (rather than using the Browser interface)
2. Determine what tools are needed to export and use a DIGITS Classifier on external hardware (e.g. a Jetson TK1 board)

Using DIGITS scripts

1. References
 - <https://github.com/NVIDIA/DIGITS/tree/master/examples/classification>
2. DIGITS locations
 - python script to classify a test image is at:
 - \$HOME/DIGITS/examples/classification/example.py
 - DIGITS trained models and data sets are stored at: /usr/share/digits/digits/jobs/
 - e.g 20160816-144923-ad9f (name appears to contain date-stamp)
 - Model classifier “jobs” contain the following files needed for object identification

- deploy.prototxt
- snapshot_iter_<some number>.caffemodel
- Also needed is the “mean” and “labels” files generated by the Dataset
 - grep mean.binaryproto caffe_output.log
 Loading mean file from: /usr/share/digits/digits/jobs/20160816-143420-d01a/mean.binaryproto
 - dataset directory=/usr/share/digits/digits/jobs/20160816-143420-d01a
 - contains mean.binaryproto and labels.txt

3. Create a model test directory

- \$ mkdir -p ~/data/models/ball-only-LE
- \$ pushd /usr/share/digits/digits/jobs/20160818-093555-89d5 (ball only model)
- \$ cp snapshot_iter_60.solverstate deploy.prototxt ~/data/models/ball-only-LE
- \$ grep mean.binaryproto caffe_output.log
 Loading mean file from: /usr/share/digits/digits/jobs/20160818-093431-a67d/mean.binaryproto
- \$ cp /usr/share/digits/digits/jobs/20160818-093431-a67d/mean.binaryproto
 ~/data/models/ball-only-LE
- \$ cp /usr/share/digits/digits/jobs/20160818-093431-a67d/labels.txt ~/data/models/ball-only-LE

4. Test script (test.sh)

```
#!/bin/sh

TESTEXE=/home/dean/DIGITS/examples/classification/example.py
BASEDIR=/home/dean/data/models/ball-only-LE
MODEL=$BASEDIR/snapshot_iter_60.caffemodel
PROTO=$BASEDIR/deploy.prototxt
MEAN=$BASEDIR/mean.binaryproto
TDIR=/home/dean/data/ball-only
TEST1=$TDIR/field/"default_ShooterCamera(1)-0312.jpg"
TESTALL=$TDIR/ball/"*.jpg"
LABELS=$BASEDIR/labels.txt
$TESTEXE $MODEL $PROTO $TEST1 --mean $MEAN --labels $LABELS
#$TESTEXE $MODEL $PROTO $TESTALL --mean $MEAN --labels $LABELS
```

5. Results

- Single image file

- `$TESTEXE $MODEL $PROTO $TEST1 --mean $MEAN --labels $LABELS`

`$./test.sh`

Processed 1/1 images in 0.051673 seconds ...

Prediction for /home/dean/data/ball-tote-2/ball/default_ShooterCamera(1)-0252.jpg

99.8915% - "ball"

0.0738% - "tote"

0.0347% - "none"

Script took 4.985445 seconds.

- Multiple files (batch-size=1)
 - `$TESTEXE $MODEL $PROTO $TESTALL --mean $MEAN --labels $LABELS`

Processed 1/85 images in 0.007678 seconds ...

Processed 2/85 images in 0.021934 seconds ...

...
 - Average ~400 us/image
 - Worst case: 0.02 s (2nd image)
- Multiple files (batch-size=85)
 - Processed 85/85 images in 0.006281 seconds ...
 - average: 73 us/image

Using Caffe Script

Note: Jetson TK1 supports Caffe but not DIGITS

1. Test script (pytest.sh)

```
#!/bin/sh
```

```
TESTEXE=/home/dean/caffe/python/classify.py
BASEDIR=/home/dean/data/models/ball-only-LE
MODEL=$BASEDIR/snapshot_iter_60.caffemodel
PROTO=$BASEDIR/deploy.prototxt
```

```
python convert_mean.py
MEAN=$BASEDIR/mean.npy
TDIR=/home/dean/data/ball-only
#TEST=$TDIR/field/"default_ShooterCamera(1)-0312.jpg"
TEST=$TDIR/ball
```

```

LABELS=$BASEDIR/labels.txt
$TESTEXE --pretrained_model $MODEL --model_def $PROTO --gpu --images_dim
32,32 --mean_file $MEAN $TEST result

python print_array.py

```

2. python convert_mean.py

The "mean.binaryproto" file generated by DIGITS "Datasets" needs to be converted to a ".npy" file for use by the Caffe script. Found the following python script to do the conversion:

```

import caffe
import numpy as np
import sys

blob = caffe.proto.caffe_pb2.BlobProto()
data = open( 'mean.binaryproto' , 'rb' ).read()
blob.ParseFromString(data)
arr = np.array( caffe.io.blobproto_to_array(blob) )
print arr.shape
arr = np.reshape(arr, (3,32,32))
print arr.shape
np.save('mean.npy', arr)

```

3. python print_array.py

This script prints out the classifier probability results as a list of fractions.

```

import numpy as np
m = np.load("result.npy")
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
print(m)

```

4. Results

- Single Image
 - TEST=\$TDIR/field/"default_ShooterCamera(1)-0312.jpg"
 - Classifying 1 inputs.
 - Done in 0.01 s.
- Multiple Images
 - TEST=\$TDIR/ball
 - Classifying 85 inputs.
 - Done in 0.48 s.
 - average: 5.6 ms/image (0.48/85)

- CPU only
 - removed `-gpu` from script
 - took 1.68 s to process 85 images
 - GPU only provided $\sim x3$ speedup over CPU (expected more)
- Comments
 - Total batch performance MUCH worse than using DIGITS script (why ??)

Use a pre-trained model for image classification in DIGITS

Neural Networks with parameters that have been trained on the *huge* ImageNet dataset with expensive gpu server arrays and many hours of compute time can be downloaded and used to categorize arbitrary images via caffe and it's browser based front-end “DIGITS”. Unfortunately, since DIGITS was designed as a “training” rather than an “evaluation” system the procedure to do this seems to be unnecessarily complex (but is described in the following reference)

reference: <https://github.com/NVIDIA/DIGITS/issues/49>

1. Obtain a pretrained caffe model

```
$ cd ~/caffe
```

```
$ scripts/download_model_binary.py models/bvlc_alexnet
```

```
$ scripts/download_model_binary.py models/bvlc_googlenet
```

2. Set up a “dummy” data set

- Create a “dummy” data directory

```
$ mkdir ~/data/dummy
```

- Save some (arbitrary) image file here

e.g. `$ cp ~/caffe/examples/images/cat.jpg ~/data/dummy/image.jpg`

- Create a textfile “train.txt” using the same image once for each category 0-999
e.g.:

```
/home/username/data/dummy/image.jpg 0
/home/username/data/dummy/image.jpg 1
...
/home/username/data/dummy/image.jpg 999
```

- e.g. shell script to do this:

```
$ for((i=0;i<1000;i+=1)); do echo "$HOME/data/dummy/image.jpeg $i
```

```
">>train.txt; done
```

- Get synset_words.txt using [this script](#)

```
mkdir ~/data/synset
```

```
cd ~/data/synset
```

```
wget http://dl.caffe.berkeleyvision.org/caffe\_ilsvrc12.tar.gz
```

```
tar -xf caffe_ilsvrc12.tar.gz && rm -f caffe_ilsvrc12.tar.gz
```

3. Configure Digits “dummy” data image

- In Browser panel: DIGITS->New Dataset->Images->Classification->Upload Text Files
Training set: Browse to ~/data/dummy/train.txt
Validation set: none (uncheck)
Labels: Browse to ~/data/synset/synset_words.txt
- Dataset Name: DummyData
- Press “Create”

4. Configure Digits classification model

1. DIGITS-> Models-> New Model → classification
2. Select model in Standard networks tab (e.g. choose 1)
Alexnet
GoogLeNet
3. Press “customize”
4. In Pretrained model box enter full path to the “.caffemodel” file in one of the directories created in 1.1 above e.g.:

/home/dean/caffe/models/bvlc_googlenet/bvlc_googlenet.caffemodel
5. Select Dataset → DummyData
6. Set the following “Solver Options”
Training Epochs: 1
Base learning rate: 0.0
Batch Size : 1 (single image) note: not described in reference
7. Set an appropriate Model Name (e.g. “GN on DummyData”)

8. Press “Create”

- If successful a new “Completed” Model will be created in the Digits home page

5. Test classify an images

- Obtain an image or images from some source

e.g. download URLs from <http://image-net.org/download>

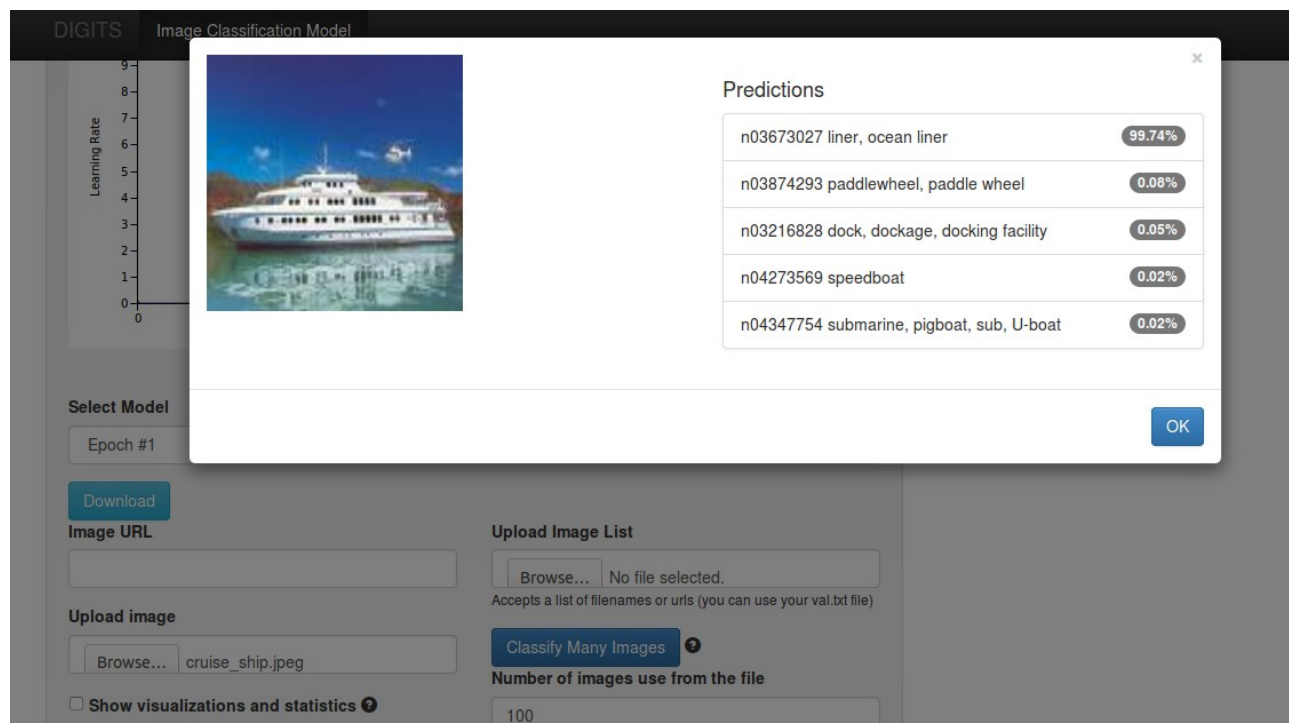
- In the Digits home page select the relevant “Completed” model

e.g. “GN on DummyData”

- At the bottom of the “Image Classification Model” page press “Browse” in the “Upload Image” tab

Browse to one of the test images and select it

- Press “Classify One Image”



6. When trying to duplicate this procedure after uploading Digits 3 classification results were much worse than with digits-2 (a ship wasn't even in the top 5 list)

- Retested Digits-2 “test 1” (ocean liner) using previously created “GN on DummyData” and accuracy was still good

- Created a new GN classification in Digits 2 with old DummyData dataset based with prebuilt /home/dean/caffe/models/bvlc_googlenet/bvlc_googlenet.caffemodel parameters and accuracy for correctly identified “ocean liner” was still >99%
- Same accuracy problem with digits-3 using Alexnet
- Selecting “none” for “subtract mean” in digits 3.0 image classification interface greatly improves accuracy over default “image” option (e.g. “ocean liner” now found with >99 % accuracy). However, some searches were still not as good as with digits -2 (e.g. “weimeraner” found with 65% in digits 3 but 89% with digits 2)
 - In digits 3 the options are “none”, “image” or “pixel”
 - using “pixel” results in good accuracy for some tests (weimeraner=97%) but inferior accuracy for others (“ocean liner” = 37%)
 - In digits 2 “subtract mean” pulldown has 2 options “yes” and “no”
 - rebuilding image classifier with “no” option reduced accuracy of “weimeraner” prediction from 89% to 77%
 - “yes” option in digits-2 and “pixel” option in digits-3 both result in identical train_val.prototxt files that include the following section:

```
transform_param {
  mirror: true
  crop_size: 227
  mean_value: 110.687591553
  mean_value: 110.9559021
  mean_value: 102.489151001
}
```

- in digits 3 selecting “image” results in:

```
transform_param {
  mirror: true
  crop_size: 227
  mean_file: "/usr/share/digits/digits/jobs/20160714-092012-ed46/mean.binaryproto"
}
```

Use a pre-trained model to improve accuracy when training on a totally different data set

References

1. <https://www.learnopencv.com/deep-learning-example-using-nvidia-digits-3-on-ec2/>

Create a DIGITS Dataset

1. Obtain some images
 - For this test a set of images of 17 flower varieties with 80 examples for each were downloaded from :
<http://www.robots.ox.ac.uk/~vgg/data/flowers/17/index.html>
 - After untarring the images are located in a sub-directory called “jpg” and have generic names like : image_0190.jpg etc.
 - To import into DIGITS the images first either need to be sorted into 17 different sub-directories or a set of text files need to be created that identify the flower category of the images in the common directory (the later approach was followed here)
2. Create a labels mapping file
 - DIGITS needs a special file (e.g. labels.txt) that's used to associate a label with a numerical index used iadditional image mapping files
 - Unfortunately, I couldn't find anything on the Oxford download site which identified which images corresponded to which flower types so had to guess the mapping based on the samples shown on the reference link. The final presumed mapping order was:
[Daffodil](#) [Snowdrop](#) [Lily](#)[Valley](#) [Bluebell](#) [Crocus](#) [Iris](#) [Tigerlily](#) [Tulip](#) [Fritillary](#) [Sunflower](#)
[Daisy](#) [Colt'sFoot](#) [Dandelion](#) [Cowslip](#) [Buttercup](#) [Windflower](#) [Pansy](#)
 - With one entry per line in the labels file with “Daffodil” associated with image_0001.jpg to image_0080.jpg etc.
3. Create the train and test mapping files needed by Digits
 - the classifier files need to consist of a set of lines with the following syntax:
 - <full-path-to-image-directory>/<imag-name> <zero-based-classification-index>
 - The indexes are mapped from the labels file using (0 based) ids that correspond to the associated line number

- e.g /home/dean/data/17flowers/jpg/image_0060.jpg 0 maps into “Daffodil” if labels.txt looks like:

Daffodil

Snowdrop

... <15 more lines>

- Used a bash shell script to create a training (train.txt) and test (test.txt) file to map the images to flower classification types:

```
#!/bin/bash
rm -f train.txt;
rm -f test.txt;
txt_file="train.txt"
for((i=0,k=1;i<17;i+=1)) do
  for((j=0;j<80;j+=1,k+=1)) do
    if [ $((($k % 10)) -eq 0 ) ]
    then
      txt_file="test.txt"
    else
      txt_file="train.txt"
    fi;
    if [ $k -lt 10 ]
    then
      echo "/home/dean/data/17flowers/jpg/image_000$k.jpg $i">>$txt_file
    elif [ $k -lt 100 ]
    then
      echo "/home/dean/data/17flowers/jpg/image_00$k.jpg $i">>$txt_file
    elif [ $k -lt 1000 ]
    then
      echo "/home/dean/data/17flowers/jpg/image_0$k.jpg $i">>$txt_file
    else
      echo "/home/dean/data/17flowers/jpg/image_$k.jpg $i">>$txt_file
    fi
  done
done
```

- test.txt contains mapping information for every 10th image which will be used for validation but not training
- train.txt contains mapping information for the remaining images (which will be used in training)

4. Create a DIGITS Dataset for the 17 flowers image set

- Open DIGITS in a file browser
- Select Datasets → Images → classification
- Use 256x256 (Color) for image size and type
- Can use Squash or Crop for Resize transformation (I used squash)

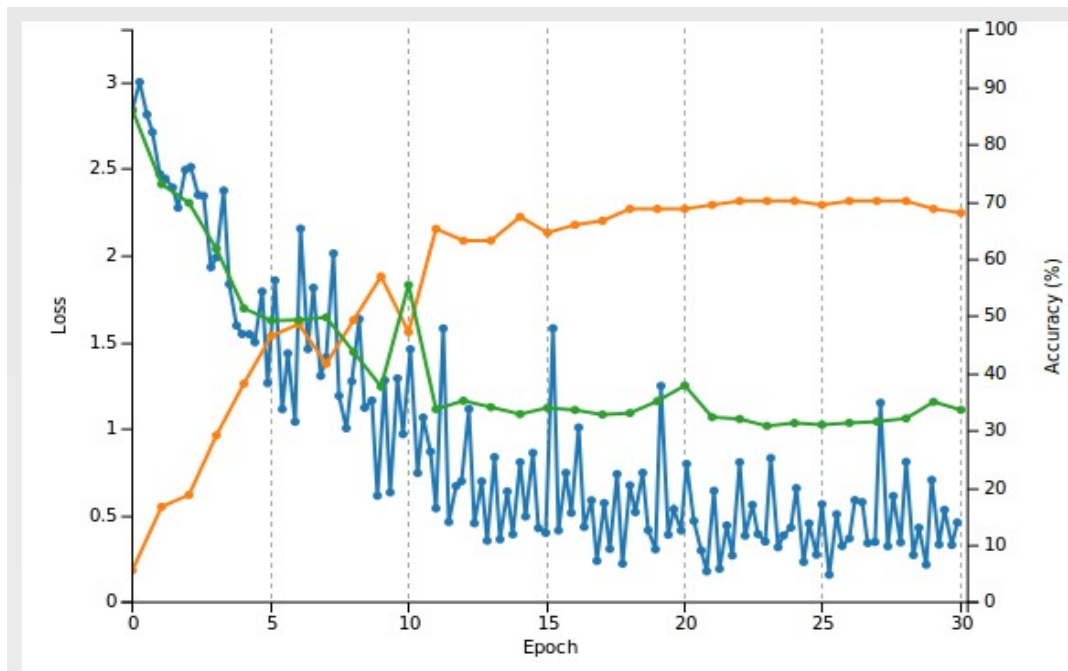
- Choose “use Text files” and enter fields as shown:

The screenshot shows the DIGITS web interface with the "Use Text Files" tab selected. The interface is divided into three columns: "Set", "Text file", and "Image folder (optional)". The "Image folder (optional)" column is disabled. The "Text file" column is active, and the "Use local paths on server" checkbox is unchecked. The "Training" row has a "Browse..." button and the text "train.txt". The "Validation" row has a checked checkbox, a "Browse..." button, and the text "test.txt". The "Test" row has an unchecked checkbox and a "Browse..." button. The "Shuffle lines" dropdown is set to "Yes". The "Labels" row has a "Browse..." button and the text "labels.txt".

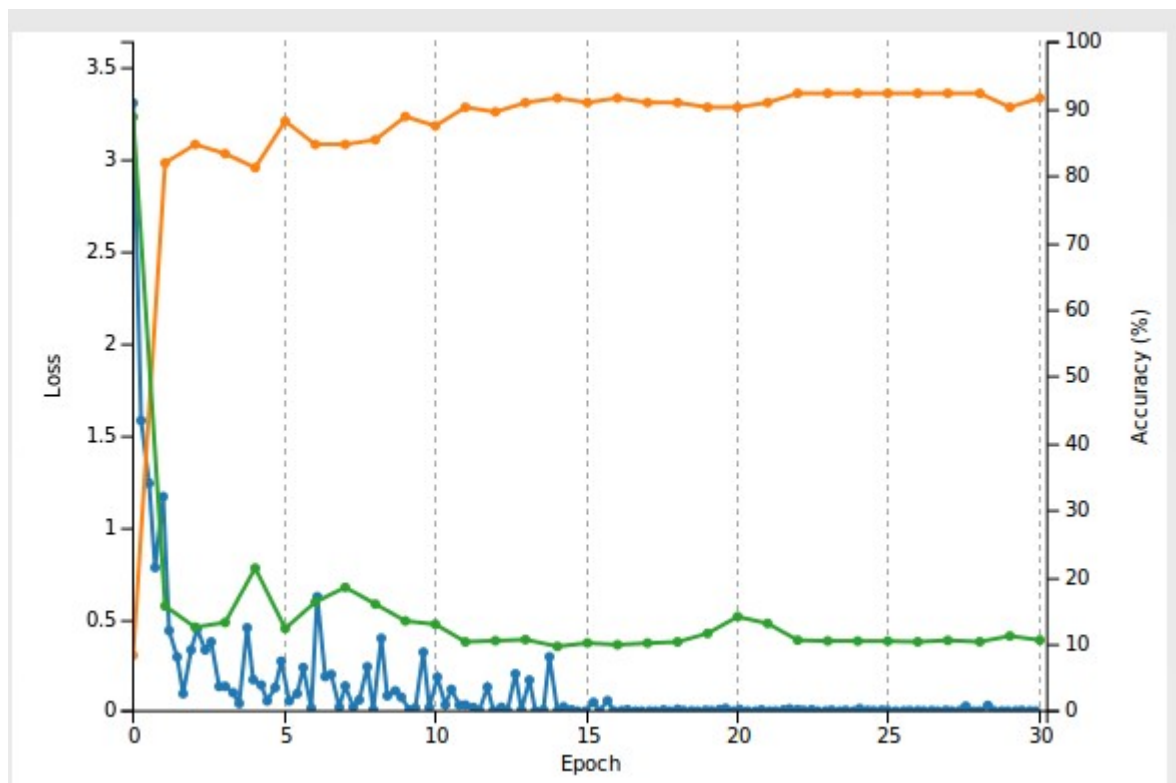
- Also used jpg for image encoding and set the “dataset Name” to “Flowers 17”
- Press Create to build the data set

Train and validate the Dataset using the Alexnet network

1. Use an new (not pre-trained) set of network parameters
 - In DIGITS Select Models → Images → Classification
 - Select “Flowers17” as the Dataset and “Alexnet” from the standard networks list
 - keep “Solver Options” defaults except:
 - reduce base learning rate from 0.01 to 0.002 (otherwise get divergence)
 - Set Batch size to 16 (otherwise got “out of memory” error)
 - Choose a model name (e.g. “Flowers17 – AN”)
 - press “Create” to start training
 - Results:



- Training took ~5 minutes with a final accuracy of ~ 70 %
2. Start with a set of pre-trained network parameters
 - Follow the image classification procedure as before but in this case press “customize” after selecting Alexnet from the standard networks list
 - In the “pre-trained model” entry field enter the full path to a previously down-loaded set of network parameters
 - e.g. from the downloaded file described in Topc 1 above
/home/dean/models/bvlc_alexnet/bvlc_alexnet.caffemodel
 - In the “Custom network” text window change all instances of “fc8” to “fc9” (should be 5 of them)
 - Not sure what this does ? (just blindly followed directions in the reference)
 - Reduce the “Base Learning rate” to 0.001
 - Choose a Model name (e.g. Flowers17- AN – Pretrained)
 - Press “Create”
 - Results:



- Conclusions:
 - Network converged on solution much faster than when using untrained parameters
 - All training images were correctly identified (with >98% probability)
 - For test data final accuracy was 92 % (vs. 68%)
 - Reference video said something about it “being beyond the scope of this tutorial” as to why a trained network trained on a totally different set of images performed much better when used as a starting point for a new set of image classifiers
 - need to look into this further

Identify objects in an image and draw bounding boxes around them

Fast R-CNN: Fast Region-based Convolutional Networks for object detection

Created by Ross Girshick at Microsoft Research, Redmond.

1. Goals

- Run the stock demo
 - Modify the demo software so that it will identify a different object in an image other than the categories provided
2. References
 1. <https://github.com/rbgirshick/fast-rcnn> (primary reference)
 2. <http://arxiv.org/pdf/1504.08083v2.pdf> (Original research paper by Ross Girshick)
 3. https://indico.cern.ch/event/395374/session/8/contribution/22/attachments/1186808/1721069/Getting_started_with_caffe_v2.pdf (page 17)
 4. <https://suryatejacheedella.wordpress.com/2015/03/22/install-caffe-on-ubuntu-14-04/> (details on how to set up Python in Caffe)
 3. Obtain the fast-rcnn source code
 - `$ cd`
 - `$ git clone --recursive https://github.com/rbgirshick/fast-rcnn.git`
 - Creates a directory called fast-rcnn in \$HOME
 - In ~/.bashrc create an export to this directory
 - `export FRCN_ROOT=$HOME/fast-rcnn`
 - get pre-computed Fast R-CNN models
 - `$./data/scripts/fetch_fast_rcnn_models.sh`
 - note: this requires about 1GB of storage and took a couple of hours since the server doesn't have very high bandwidth for downloads
 4. Build the Python modules
 - `cd $FRCN_ROOT/lib`
 - `$ make`
 5. Build a local (special) version of caffe in ~/fast-rcnn/caffe-fast-rcnn
 - In Makefile.config uncomment: `WITH_PYTHON_LAYER := 1`
 - `$ make -j8 && make pycaffe`
 - Apparently, you can't use a standard caffe build because fast-rcnn has added a couple of new layers (roi_pooling_layer etc.)
 6. Build Problems
 - Could not run the demo because of various Python “import” errors (_caffe,

google.protobuf,cv2)

- Tried various combinations of setting for PYTHONPATH and PYTHON_INCLUDE environment variables but to no avail
- Fix was to add the following to \$FRCN_ROOT/lib/test.py and \$FRCN_ROOT/lib/fast_rcnn/config.py

```
import sys
```

```
sys.path.append('/usr/local/lib/python2.7/dist-packages/')
```

7. Stock Demo Output (monitor, sofa,cars)

- Run the demo

- \$ cd ~/fast-rcnn

- tools/demo.py

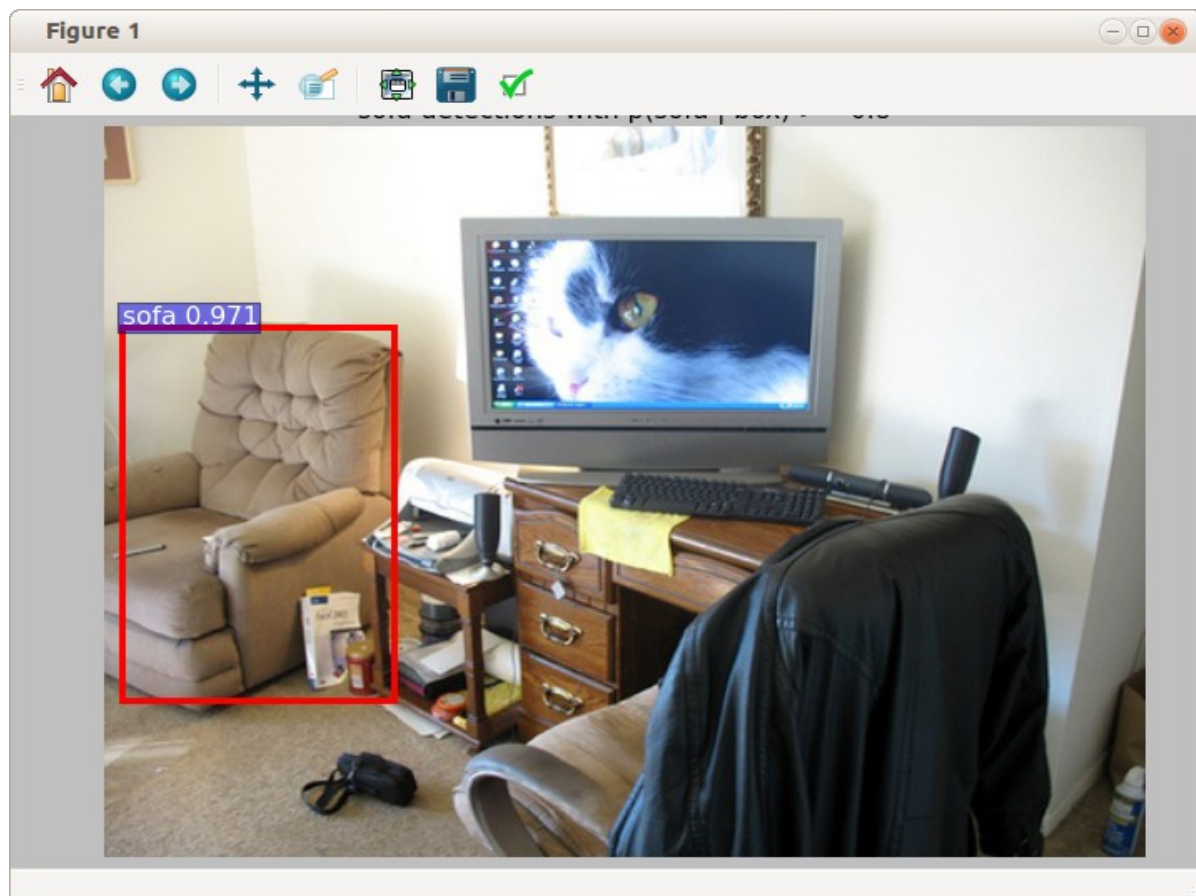
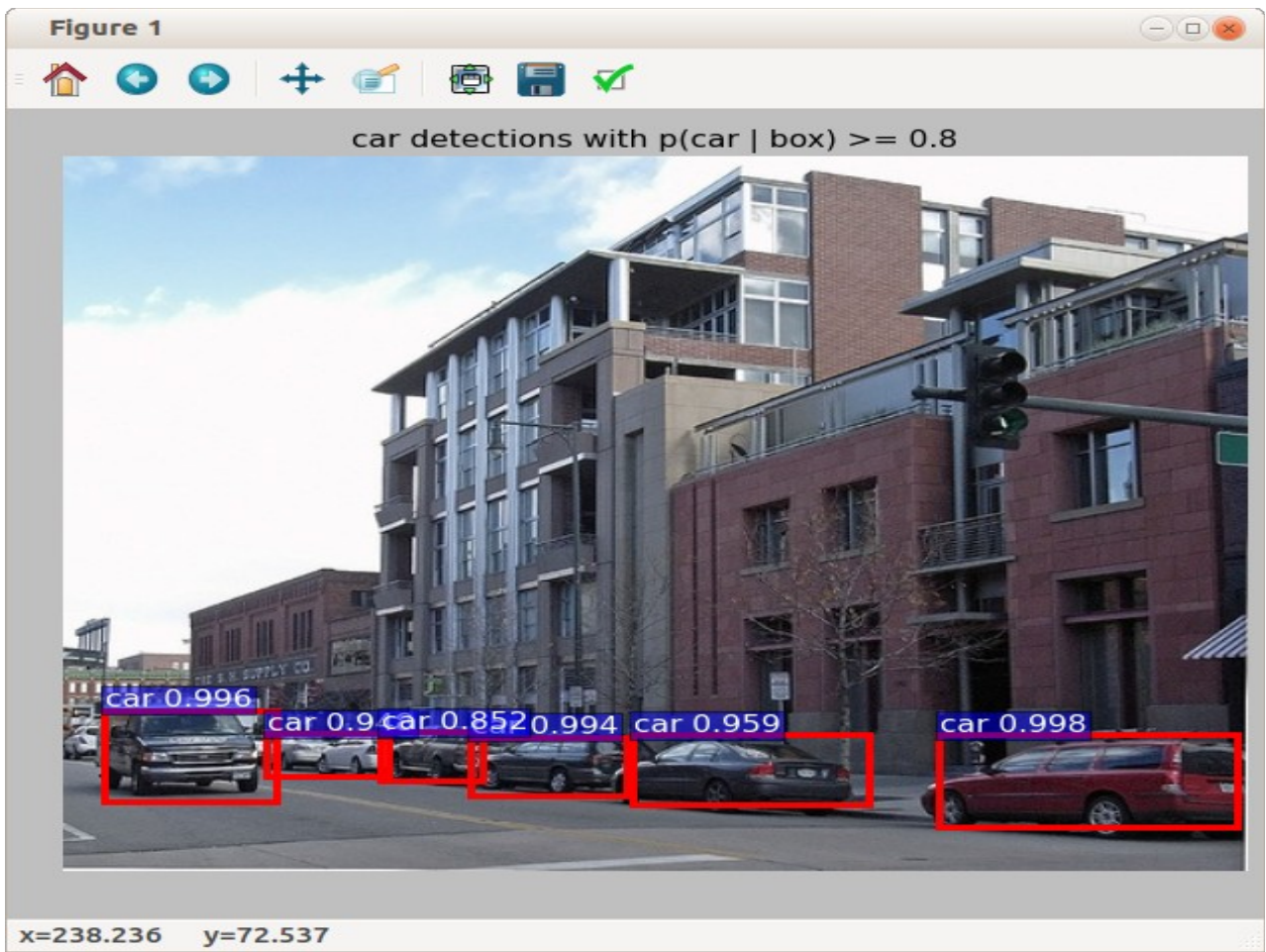
```
Loaded network /home/dean/fast-rcnn/data/fast_rcnn_models/vgg16_fast_rcnn_iter_40000.caffemodel
```

```
~~~~~
```

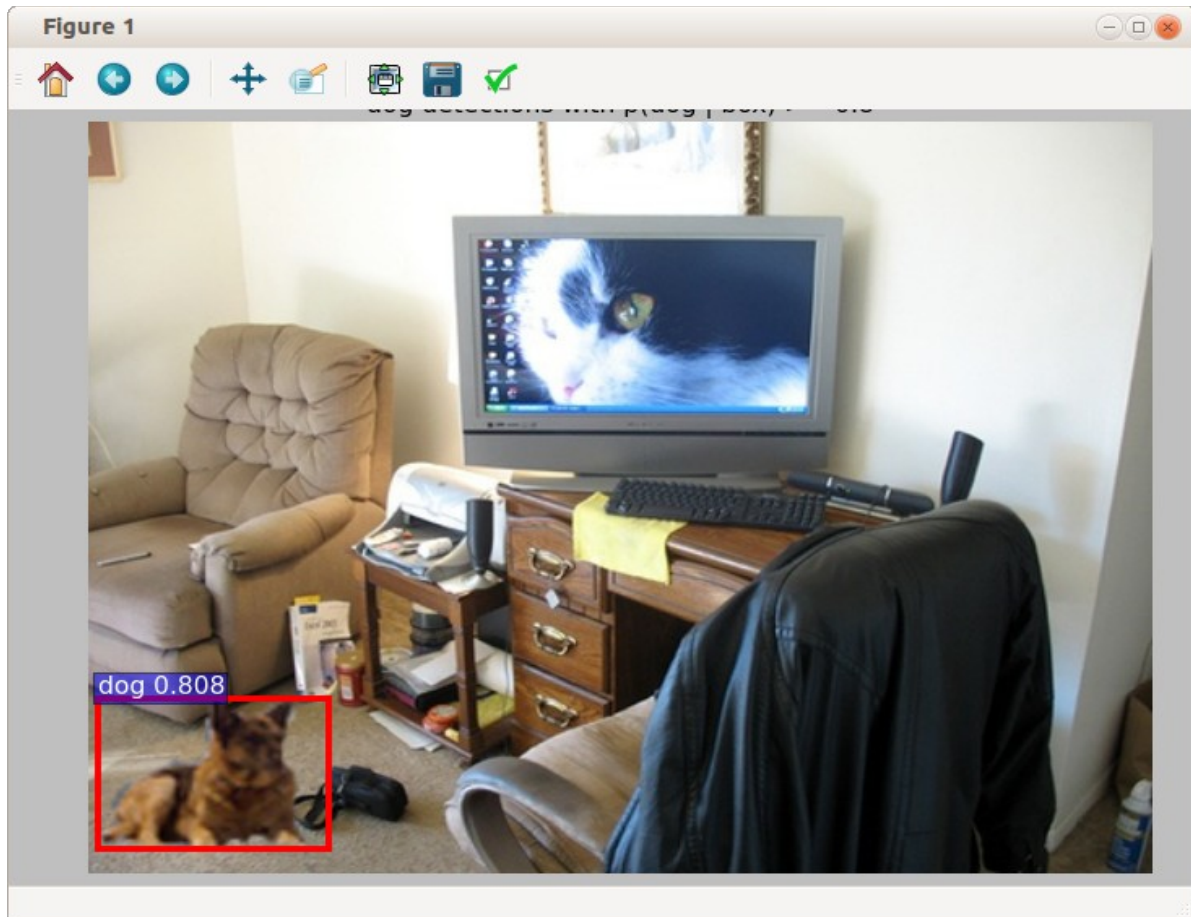
```
Demo for data/demo/000004.jpg
```

```
Detection took 0.572s for 2888 object proposals
```

```
All car detections with p(car | box) >= 0.6
```



8. Goal 2: Add a dog to the stock room image containing the sofa and tvmonitor



- Comment: needed to create an “image.mat” file by cloning the one associated with the “cars” image (000004). Using the mat file for the sofa/tvmonitor image didn't work (dog not found)

py-faster-rcnn

According to recent comments in reference 1 by the author the active code base for “fast r-cnn” has been moved to a new github site (py-faster-rcnn) and the original site is now only maintained for “historical reasons”

1. References
 - <https://github.com/rbgirshick/py-faster-rcnn>
2. Obtain source code
 - `git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git`
 - `export FRCN_ROOT=$HOME/py-faster-rcnn`
3. Build Cython modules
 - `cd $FRCN_ROOT/lib`

- make
4. Obtain pre-trained network
 - `cd $FRCN_ROOT`
 - `./data/scripts/fetch_faster_rcnn_models.sh`
 5. build the “special” caffe version (following instructions from reference)
 - `cd caffe-fast-rcnn`
 - `cp Makefile.config.example Makefile.config`
 - edit `Makefile.config`
 - uncomment `USE_CUDNN := 1, WITH_PYTHON_LAYER := 1`
 - make
 - Got multiple errors (function prototype mismatches etc.)
 6. Try to fix build problems and build again

It looks like the code base is incompatible with CUDA 7.5, cuDNN 5.1 (or both)

 - needed to obtain an older version of cuDNN (3.0) from the NVIDIA archive and copy the libraries and include files from it to a older CUDA SDK installation directory (in `/usr/local/cuda-6.5`)
 - Additionally, modified `Makefile.config` as follows:
 - `CUDA_DIR := /usr/local/cuda-6.5`
 - `CUSTOM_CXX := /usr/bin/g++-4.8`
 - when using `g++4.9` got errors about incompatibility with `g++ 4.9` or later
 - rebuild
 - `make -j8` (now completes)
 - `make pycaffe`
 - run demo
 - `tools/demo.py`
 - get dynamic library link errors
 - `export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib64`
 - `tools/demo.py`

- demo now runs

~~~~~

Demo for data/demo/000456.jpg

Detection took 0.171s for 300 object proposals

~~~~~

...

7. Found a hint on the following website: <https://github.com/rbgirshick/py-faster-rcnn/issues/237> that first merges in changes from the upstream caffe website (which is compatible with later CUDA tools)

- reverted to original Makefile.config
 - `cp Makefile.config.example Makefile.config`
 - uncomment `USE_CUDNN := 1`, `WITH_PYTHON_LAYER := 1`
 - uncomment `OPENCV_VERSION := 3`
 - Otherwise, got opencv linker errors at end of build

- Merge in latest code from caffe

```
git remote add caffe https://github.com/BVLC/caffe.git
git fetch caffe
git merge caffe/master
```

- Fix code problem as described in issues report cited above

Remove `self_.attr("phase") = static_cast<int>(this->phase_);` from `include/caffe/layers/python_layer.hpp` after merging.

- Note: without this demo.py fails with :

```
Traceback (most recent call last):
```

```
File "tools/demo.py", line 135, in <module>
```

```
    net = caffe.Net(prototxt, caffemodel, caffe.TEST)
```

```
AttributeError: can't set attribute
```

- build
 - `make -j8; make pycaffe`
 - build now finishes !

- run demo

- `cd ../`
- `tools/demo.py`
- demo now runs ! (also slightly faster than before with cudnn 3.0)

```
Loaded network /home/dean/py-faster-rcnn/data/faster_rcnn_models/VGG16_faster_rcnn_final.caffemodel
```

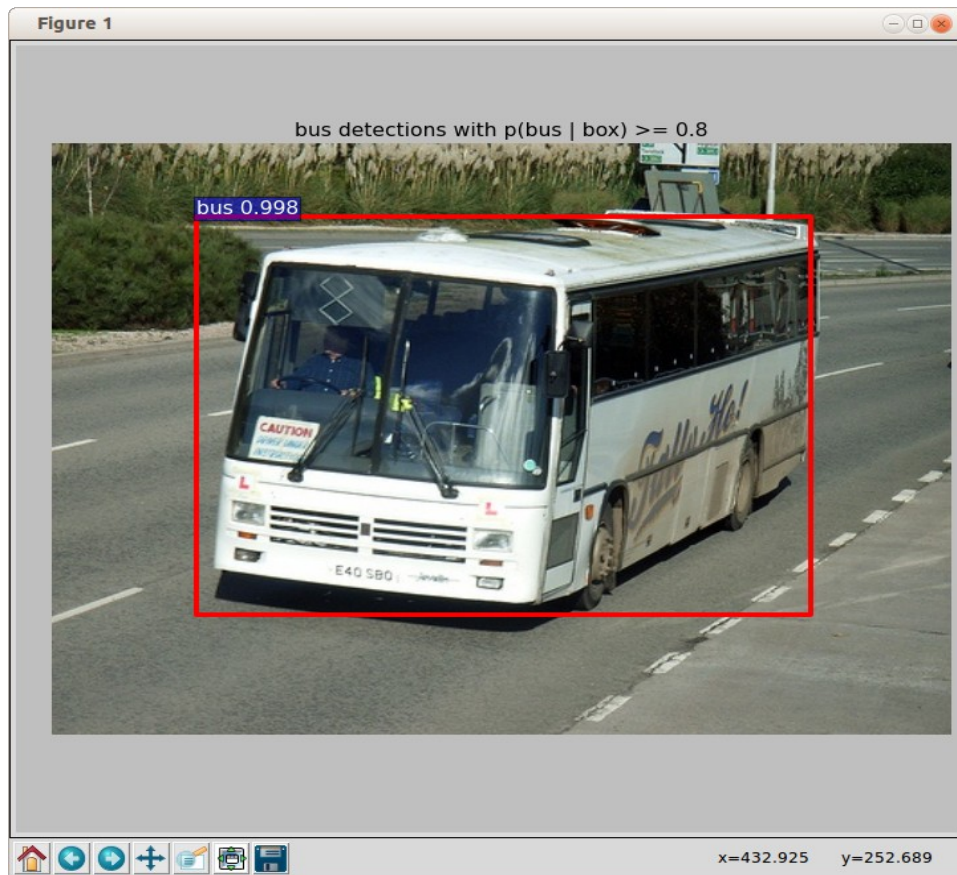
```
~~~~~
```

```
Demo for data/demo/000456.jpg
```

```
Detection took 0.151s for 300 object proposals
```

```
~~~~~
```

```
...
```



- run demo (cpu only)

- `tools/demo.py -cpu`

Demo for data/demo/000456.jpg

Detection took 26.677s for 300 object proposals

- GTX 980 provides 178x speedup over CPU
- run demo (zf net)

- ./tools/demo.py --net zf

- Loaded network /home/dean/py-faster-rcnn/data/faster_rcnn_models/ZF_faster_rcnn_final.caffemodel

Demo for data/demo/000456.jpg

Detection took 0.071s for 300 object proposals

- “zf net” 2x faster than default (vgg16)
- some differences in objects that got detected
- Comments
 - The “faster” version seems to use fewer “object proposals” (bounding box candidates) than the original version (300 vs ~3000) which probably explains most of the speedup (0.15 vs 0.5 s typical)
 - With cudnn 5.1 get a ~13% speed improvement over cudnn 3.0 (0.151 vs 0.171 on same image)
 - Probably would still need significant improvements to be viable as a real-time target detection system on less capable hardware (e.g. Jetson TK1) but is worth exploring
Possible speedup approaches:
 - analyze smaller images (320x240)
 - reduce classification set (only 1 or 2 object types)
 - use fewer object proposals (50-100 enough?)
 - limit target size range (e.g. 0.1-0.5 full image scale)

Faster-RCNN - Beyond The Demo

References:

1. <https://github.com/rbgirshick/fast-rcnn>
2. <https://github.com/rbgirshick/py-faster-rcnn>
3. <http://sunshineatnoon.github.io/Train-fast-rcnn-model-on-imagenet-without-matlab/>

Projects

1. Use command line tools to test one of the datasets mentioned in the reference
 - Create a directory to hold test and training data
 - `mkdir -p ~/data/VOCdevkit && cd VOCdevkit`
 - Download the VOCdevkit image data as described in reference 1

```
wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar
wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCdevkit_08-Jun-2007.tar
```
 - Create a soft link to the data in `~/py-faster-rcnn/data`
 - `cd ~/py-faster-rcnn/data`
 - `ln -s ~/data/VOCdevkit VOCdevkit2007`
 - Download the pre-computed Selective Search object proposals for VOC2007 and VOC2012
 - `cd ~/py-faster-rcnn`
 - Test the VOC2007 data with the trained VGG16 network using the following command:
 - `$./tools/test_net.py --gpu 0 --def models/pascal_voc/VGG16/fast_rcnn/test.prototxt`

```
--net data/faster_rcnn_models/VGG16_faster_rcnn_final.caffemodel
```

- Output:

...

AP for aeroplane = 0.7477

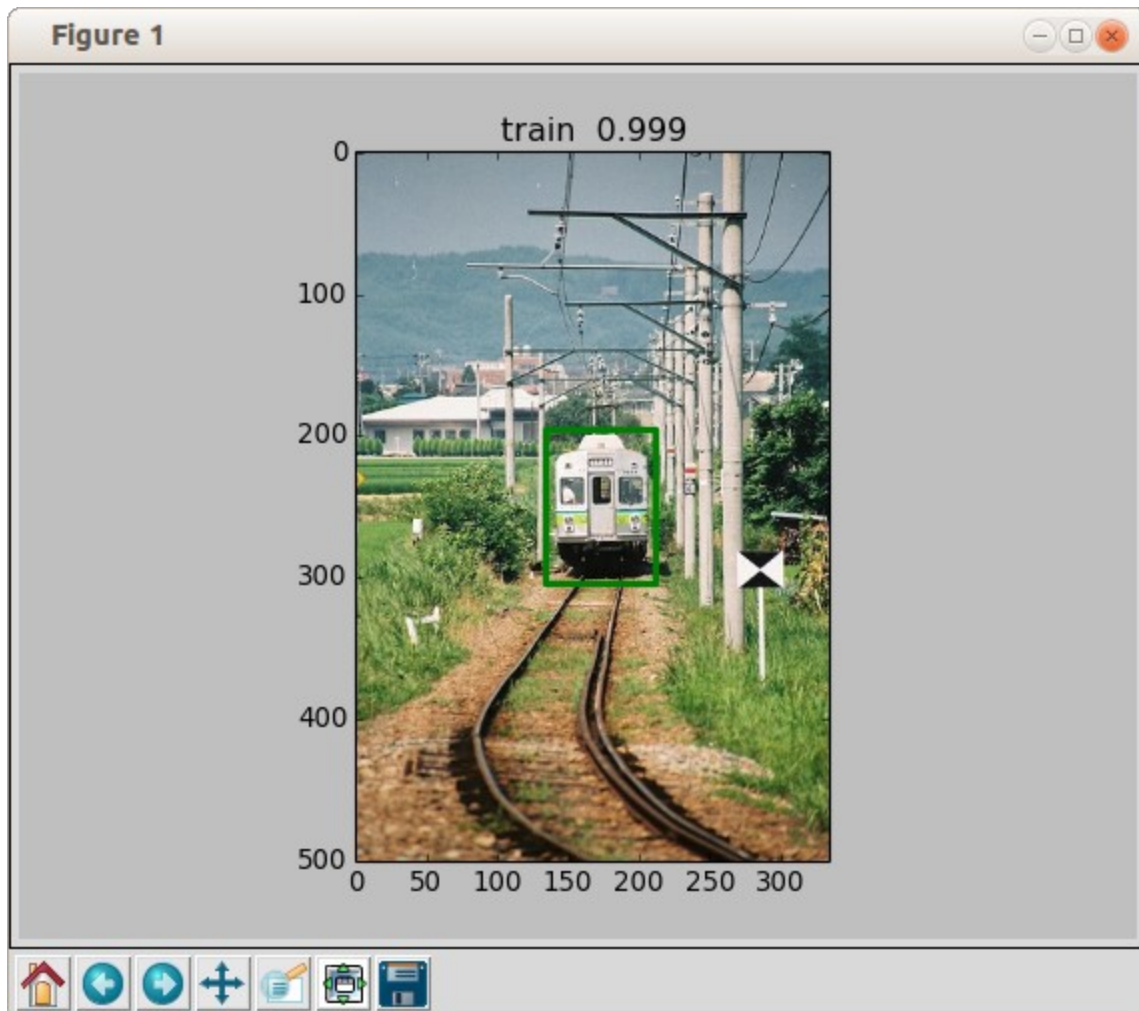
AP for bicycle = 0.7813

AP for bird = 0.7223

...

Mean AP = 0.6804

- Add “-vis” to the above command to see the results of testing each or the 4092 images (need to close each figure to see the next result)



- Also tried testing using the smaller “zf” network
 - `tools/test_net.py --gpu 0 --def models/pascal_voc/ZF/fast_rcnn/test.prototxt --net data/faster_rcnn_models/ZF_faster_rcnn_final.caffemodel`
 AP for aeroplane = 0.6495
 AP for bicycle = 0.6969
 AP for bird = 0.5924
 ...
 AP for tvmonitor = 0.5693
 Mean AP = 0.5874
 - ave ~3 ms/image but mean AP quite a bit worse than VGG16 (0.68)

2. Train one of the referenced datasets using command syntax

- `./tools/train_net.py --gpu 0 --solver models/pascal_voc/VGG16/fast_rcnn/solver.prototxt --weights data/faster_rcnn_models/VGG16_faster_rcnn_final.caffemodel`
 ...

```
I0830 19:13:15.138401 25313 solver.cpp:228] Iteration 39980, loss = 0.161604
I0830 19:13:15.138479 25313 solver.cpp:244] Train net output #0: loss_bbox = 0.0734029 (* 1 = 0.0734029 loss)
I0830 19:13:15.138496 25313 solver.cpp:244] Train net output #1: loss_cls = 0.0882007 (* 1 = 0.0882007 loss)
I0830 19:13:15.138514 25313 sgd_solver.cpp:106] Iteration 39980, lr = 0.0001
speed: 0.479s / iter
(5.3 hours to train)
```
- Test the newly trained model
 - `./tools/test_net.py --gpu 0 --def models/pascal_voc/VGG16/fast_rcnn/test.prototxt --net output/default/voc_2007_trainval/vgg16_fast_rcnn_iter_40000.caffemodel`
 AP for aeroplane = 0.6899
 AP for bicycle = 0.7933

...

Mean AP = 0.6746

3. Train a non-referenced dataset using the fast-rcnn method
4. Create a new data set and train it using the fast-rcnn method

Object Detection (with bounding box determination) using DIGITS

1. References

1. <https://github.com/NVIDIA/DIGITS/tree/master/examples/object-detection>
2. <https://devblogs.nvidia.com/parallelforall/detectnet-deep-neural-network-object-detection-digits/>

2. Requirements

- DIGITS 4.0 or later
- NVCAffe 0.15.1 or later
 - includes “detectnet_network.prototxt file and other files
- Obtain Pretrained Googlenet.caffenet model
 - http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel

3. Setup

- Obtain KITTI Image Data
 - KITTI website: http://www.cvlibs.net/datasets/kitti/eval_object.php
 - follow instructions in reference 1
 - need to obtain special email link download (12GB)
- Refactor KITTI Data to be compatible with DIGITS
 - `$DIGITS_HOME/examples/object-detection/prepare_kitti_data.py -i <source-dir> -o <dest-dir>`

4. Create Test DIGITS Dataset (KITTI)

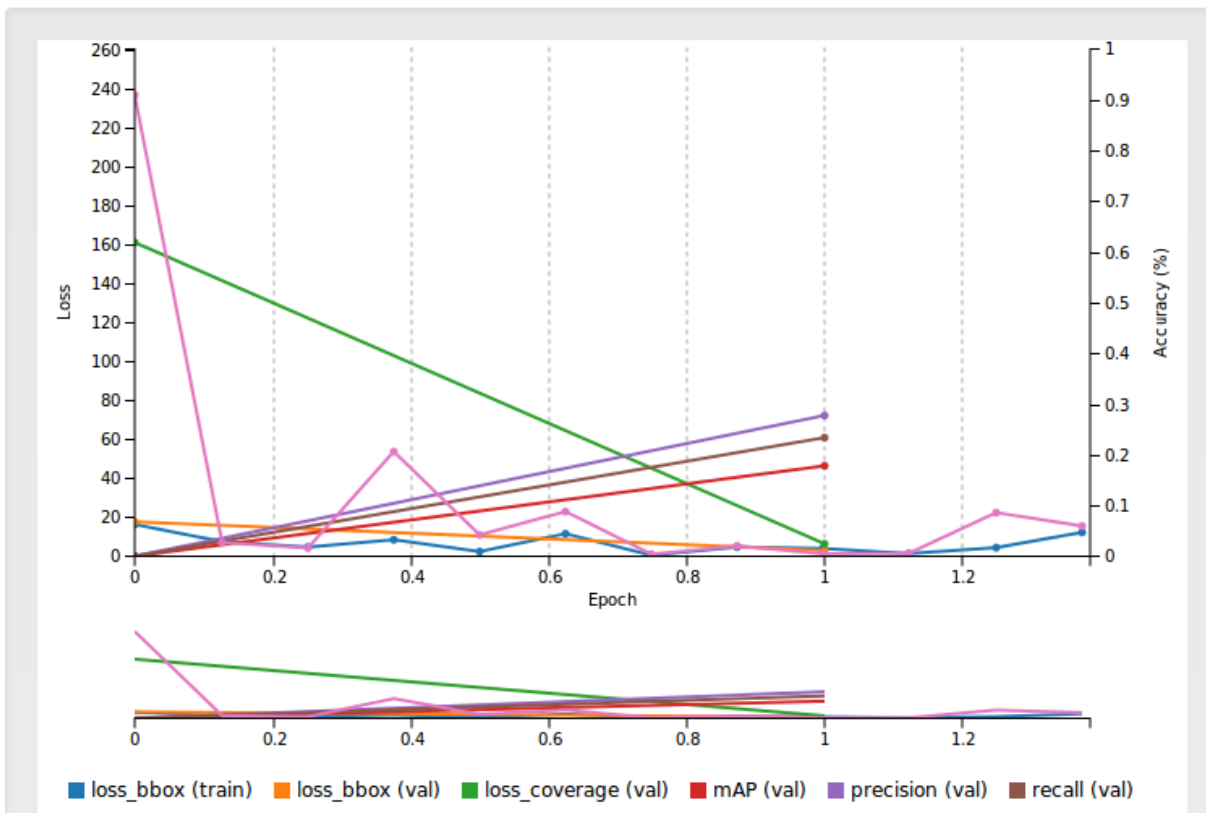
- Create Dataset as described in reference 1

5. Train Model

- Setup model classifier as described in reference 1

- Give model a name (e.g. KITTI GN)
- Press Create
- Wait for completion (?)
 - On my system with a GTX 980 a single epoch took >1.5 hours
 - After 1.2 epochs (2 hours) got this failure:

```
I0827 20:01:29.032479 2725 sgd_solver.cpp:106] Iteration 4378, lr = 0.0001
libpng error: IDAT: CRC error
E0827 20:05:49.394631 2754 io.cpp:173] Could not decode datum
*** SIGFPE (@0x7f1e5d320320) received by PID 2725 ...
```
 - problem may be image corruption in Dataset (png CRC error followed by Floating point error ?) or buggy code
 - Will go ahead and test results from epoch #1
- Results (epoch #1):



- Test 1 image:

- took 7 seconds

Found 4 bounding box(es) in 1 image(s).

Source image



Inference visualization



6. Observations

- Buggy and very slow (both train and classify)

DIGITS with fast-rcnn ?

It would be nice to be able to use the web-based UI in DIGITS to train and test networks from fast-rcnn (or faster-rcnn). Unfortunately, this isn't directly possible because the caffe component of those two github projects have diverged from each other (and both are also different from the latest BVLC caffe version). An attempt to obtain a common code base that would allow fast-rcnn models to be imported into DIGITS is briefly described in this section

Component versions (as of 9/1/2016)

1. BVLC caffe (rc3, master)

- git clone <https://github.com/BVLC/caffe>
- 2. py-faster-rcnn
 - git clone <https://github.com/rbgirshick/py-faster-rcnn>
 - note: incompatible with latest BVLC caffe and cuDNN
- 3. NVIDIA-caffe (0.15)
 - git clone <https://github.com/NVIDIA/caffe> NVIDIA-caffe
 - note: incompatible with py-faster-rcnn
- 4. DIGITS (4.0)
 - git clone <https://github.com/NVIDIA/DIGITS>
- 5. cuDNN (5.1)
 - download from NVIDIA developer site
- 6. CUDA SDK (7.5)
 - download from NVIDIA developer site

Merge Strategy

1. Create initial directory for caffe merge (caffe-fast-rcnn)
 - Start with py-faster-rcnn
 - git clone git clone <https://github.com/rbgirshick/caffe-fast-rcnn/tree/4115385deb3b907fcd428ac0ab53b694d741a3c4>~/caffe-fast-rcnn
 - Merge in latest version of BVLC caffe
 - cd caffe-fast-rcnn
 - Then follow the procedure described in section py-faster-rcnn (7) above
 - Since I already did all this when fixing the compatibility issues with py-faster-rcnn described previously I just did the following:
 - git clone ~/py-faster-rcnn/caffe-fast-rcnn ~/caffe-fast-rcnn
 - downside to this is that I can only pull in changes from my local git repo
 - note: this caffe is now compatible with py-faster-rcnn, CUDA SDK-7.5 and cuDNN-5.1
2. Merge in code from NVIDIA-caffe (0.15)

- Add NVIDIA-caffe as a new remote repo
 - Again to save time used the local repo already present in ~/NVIDIA-caffe
 - `git remote add nvidia-caffe ../NVIDIA-caffe`
 - `git fetch nvidia-caffe`
- Try a merge
 - `git merge remotes/nvidia-caffe`
 - most files merged ok but got conflicts in ~10 or so
- Try to fix merge conflicts
 - `git mergetool` (configured for kdiff3)
 - In kdiff3 choose remote file (NVIDIA-caffe) for most (should be all?) conflicts
 - save and quit kdiff3 after each file is merged (git will then reopen kdiff3 with the next file that has conflicts)
- Attempt a caffe build
 - `make -j8 -k`
 - `-k=keep going`
 - got only 1 build error:
 - ...
 - CXX src/caffe/layer_factory.cpp
 - NVCC src/caffe/solvers/adagrad_solver.cu
 - src/caffe/layers/cudnn_relu_layer.cpp: In member function 'virtual void caffe::CuDNNReLULayer<Dtype>::LayerSetUp(const std::vector<caffe::Blob<Dtype>*>&, const std::vector<caffe::Blob<Dtype>*>&)':
 - src/caffe/layers/cudnn_relu_layer.cpp:15:3: error: 'createActivationDescriptor' is not a member of 'caffe::cudnn'
 - ```

 cudnn::createActivationDescriptor<Dtype>(&activ_desc_,
 CUDNN_ACTIVATION_RELU);

```
  - fix was to just comment out the offending line (15)
    - `//cudnn::createActivationDescriptor<Dtype>(&activ_desc_, CUDNN_ACTIVATION_RELU);`

- since there was another similar line immediately below in the file the problem was probably caused by a bad git merge attempt (it happens!)
  - \$ make -j8
    - build now completes
- Attempt a pycaffe build
  - make pycaffe
 

```
CXX/LD -o python/caffe/_caffe.so python/caffe/_caffe.cpp
python/caffe/_caffe.cpp: In instantiation of 'void
caffe::Solver_add_callback
error: invalid new-expression of abstract class type
'caffe::PythonCallback<float>'
...

```
  - looks like I kept the wrong version in the kdiff3 merge (try fetching the file from the NVIDIA repo)
    - git checkout remotes/nvidia-caffe -- python/caffe/\_caffe.cpp
  - \$ make pycaffe
    - build now completes
- Attempt to run DIGITS using this version caffe
  - Using a text editor, changed “caffe\_root” in ~/DIGITS/digits/digits.cfg to:
    - caffe\_root = /home/dean/caffe-fast-rcnn
    - could have also used `python -m digits.config.edit -v` from ~/DIGITS
  - \$ ~/DIGITS/digits-server -b localhost:5000
 

```
Error: 'gunicorn_config.py' doesn't exist
```
  - gunicorn\_config.py (and other python files) are in ~/DIGITS and need to be copied over to the python directory in caffe-fast-rcnn

### 3. Add in directories and files from DIGITS

- cd ~/DIGITS
- cp -R \*.py tools digits ~/caffe-fast-rcnn/python
- Q: could I have avoided having to do this this by modifying PYTHONPATH to include

DIGITS directories and files?

#### 4. Try another DIGITS launch

- `~/DIGITS/digits-server -b localhost:5000`

```
2016-09-02 08:28:56 [21429] [INFO] Starting gunicorn 17.5
```

```
...
```

```
File "/home/dean/caffe-fast-rcnn/python/caffe/__init__.py", line 2, in
<module>
```

```
 from ._caffe import set_mode_cpu, set_mode_gpu, then last set_device,
 Layer, get_solver, layer_type_list, set_random_seed
```

```
ImportError: cannot import name set_random_seed
```

- Fix (?) was to remove ",set\_random\_seed" from line 2 of `__init__.py`
  - will probably be a problem if some check box in DIGITS requires this and will need to figure out later where "set\_random\_seed" is supported in the code.
- `~/DIGITS/digits-server -b localhost:5000`
  - DIGITS now comes up and seems to work for basic functionality but will need to do more testing to see how well it supports fast-rcnn (or if anything in the normal UI is now broken)

## Compile and run the Nvidia CUDA examples

It is assumed that the CUDA toolbox has been downloaded and installed in `/usr/local/cuda-7.5`, the environment variable have been configured etc.

e.g see section 3.1 in "System Configuration" above

#### 1. Copy the source code into a build directory

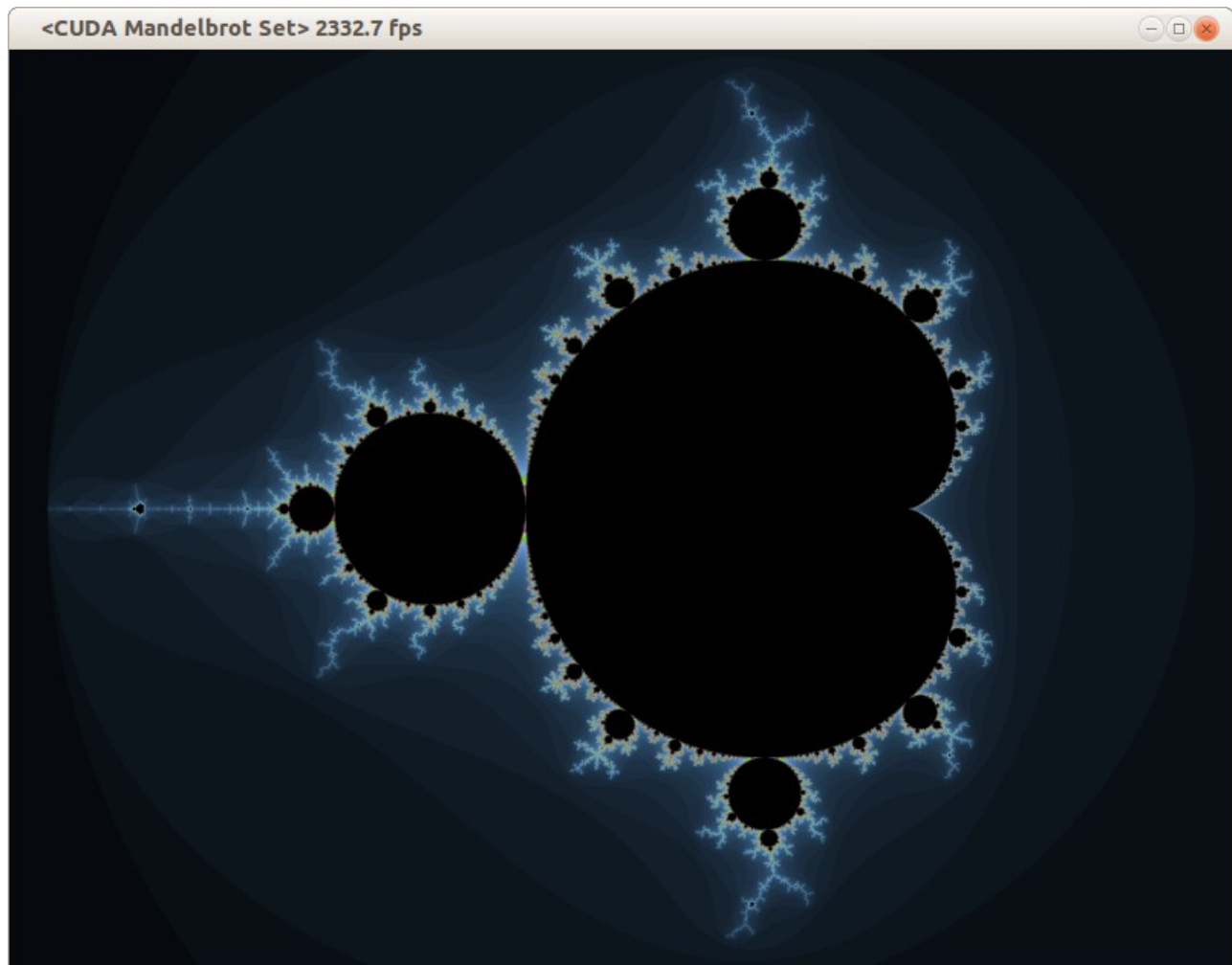
- `$ cd $CUDA_HOME/bin`
- `$ ./cuda-install-samples-7.5.sh ~`
  - creates a directory called `NVIDIA_CUDA-7.5_Samples` in `$HOME`

#### 2. Compile the sample code

- `$ cd ~/NVIDIA_CUDA-7.5_Samples`
- `$ make -j6`

#### 3. goto the build directory

- `$ cd ~/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release`
- 4. Run the example (e.g. mandelbrot)
- `$ ./mandelbrot`



- 5. Examine the source code for the examples
- Sample code is located in subdirectories of the following directories in NVIDIA\_CUDA-7.5\_Samples
  - 0\_Simple
  - 1\_Uilities
  - 2\_Graphics
  - 3\_Imaging
  - 4\_Finance

5\_Simulations

6\_Advanced

7\_CUDA Libraries

6. Build and run individual samples in the sample directories
  - cd to one of the sample directories (e.g. `cd ~/NVIDIA_CUDA-7.5_Samples/5_Simulations/oceanFFT`)
  - There should already be an executable here that was built as part of step 3 above so just enter its name in the command line to run it (e.g. `./oceanFFT`)
  - If you want to change the code and rebuilt the sample just run make
    - Probably should make a backup first of the original source code in this case

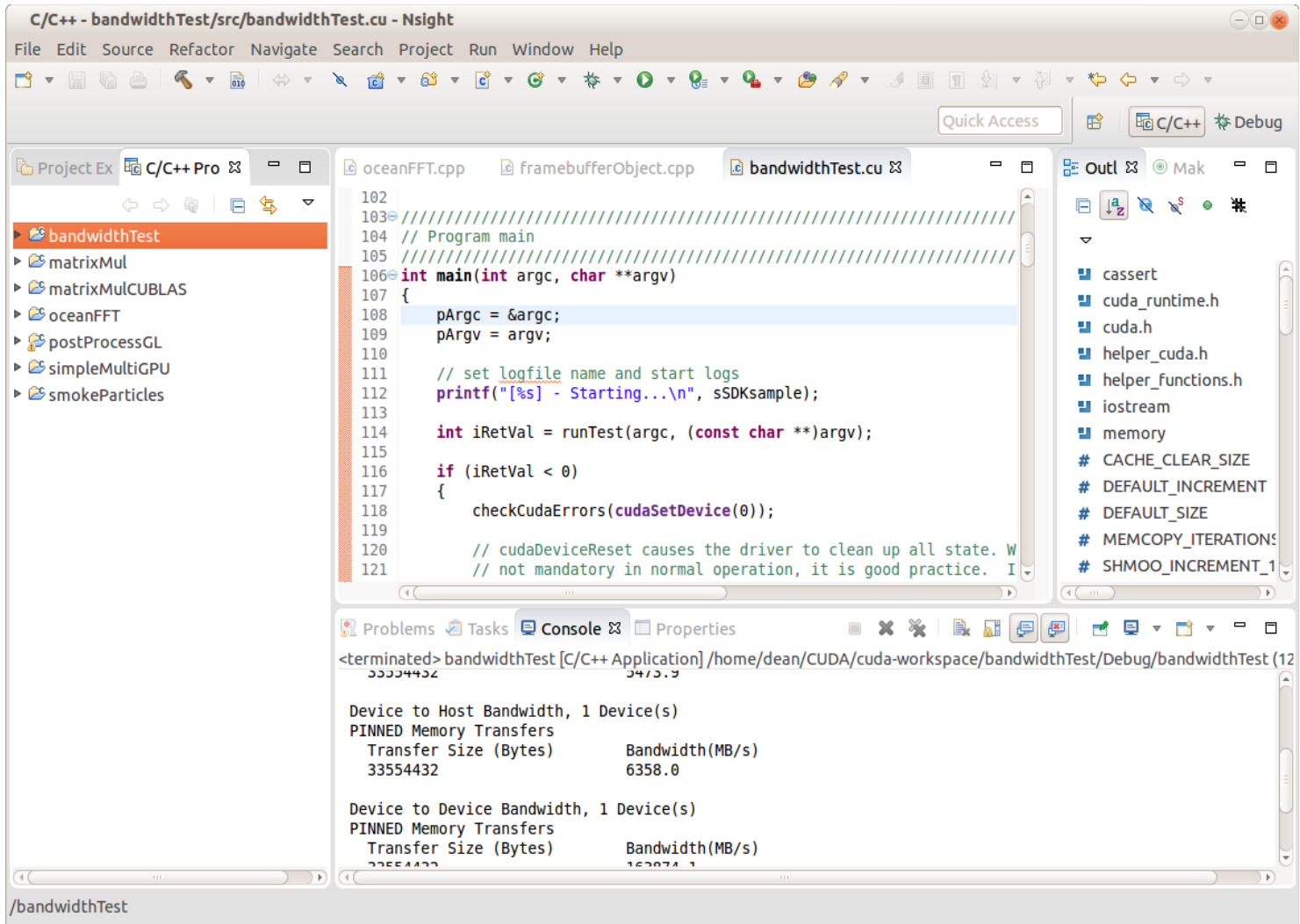
## NSIGHT - NVidia Eclipse based IDE

Nsight is included as part of the CUDA 7.5 Toolkit and provides an Eclipse based development and debugging environment for pure CUDA applications targeted to local and remote NVidia GPUs

1. Setup
  - Install CUDA 7.5 toolkit (see system installation 3.1 above)
2. Launching
  - Applications->programming->Nsight Eclipse Edition
  - or nsight from a command prompt
3. Problems
  1. nsight Eclipse interface had FRC plugin decorations and symbols
    - but path at first launch was set to non-frc eclipse (i.e. `/opt/eclipse/eclipse` vs. `/opt/eclipse-frc`) ?
    - Perhaps frc environment was set when CUDA 7.5 toolkit was installed (?)
    - work-around
      - press “already installed” link in Help->Install new software ... panel
      - manually “uninstall” FRC plugins and references in “Installed Software” tab
  2. Can't change default workspace directory
    - The first time nsight launches it creates a directory called `cuda_workspace` in `$HOME`
      - But wanted something different (e.g. a directory called `cuda_workspace` in `~/CUDA`)
    - There doesn't seem to be any way to change the default directory from the Eclipse interface or to have a workspace selection dialog show up on launch
      - Eclipse always starts up without a prompt and creates a `~/cuda_workspace` directory if one doesn't exist
      - launch appears to ignore “prompt for workspaces on startup” checkbox in windows->preferences->startup and shutdown->workspaces panel
    - work-around
      - moved `cuda_workspace` to `~/CUDA`

- launched nsight (this creates a new cuda\_workspace in ~)
  - In Eclipse, selected “switch workspace..” in file menu (then browsed to CUDA/cuda\_workspace)
  - set “prompt for workspaces on startup” checkbox in windows->preferences->startup and shutdown->workspaces panel
  - exit nsight
  - sudo gedit /usr/local/cuda/libnsight/configuration/config.ini
    - changed: osgi.instance.area.default=@user.home/cuda-workspace
    - to: osgi.instance.area.default=@user.home/CUDA/cuda-workspace
  - relaunch nsight
    - now workspace selection dialog appears and can select ~/CUDA/cuda-workspace
  - deleted ~/cuda-workspace
4. Compiling and running samples
- Note: Assumes samples are already in /usr/local/cuda-7.5/samples as part of 7.5 toolkit installation
1. creating a new c++ sample code project
    - File → new CUDA C/C++ Project
    - enter a project name (e.g. “bandwidthTest”)
    - select “Import CUDA Sample” from “project type”
    - Select sample project from list (e.g. bandwidthTest)
    - Press “Finish”
  2. Compiling the sample code
    1. Debug configuration
      - select sample project in C/C++ projects window
      - press build (hammer) select “Debug”
    2. Release Configuration
      - select sample project in C/C++ projects window
      - press build (hammer) select “Release”
  3. Debug the sample
    - select sample project in C/C++ projects window
    - Press Debug (bug symbol) >”Local C/C++ Application” in Toolbar Panel
    - answer “yes” to “Open Debug Perspective ?” question
    - Set breakpoints etc.
    - press > icon to start debugging
  4. Run the optimized sample code (release)
    - select sample project in C/C++ projects window
    - Press Run as (green button with right chevron symbol) >Local C/C++ Application in Toolbar Panel
  5. Screenshot of nsight eclipse IDE





## Linking g++ code with CUDA libraries

How to build a c++ application using g++ /Eclipse and link it with a static or dynamic project built using nvcc/Nsight

1. Build CUDA library project
  - In nsight, create a new (managed) library project (e.g. cuda\_process)
  - Import or create library source code
    - May involve a combination of .cu and .cpp files
    - For easier export good idea is to only add cuda specific #includes to source files (not the library .h file)
  - Set project properties to build static or dynamic library

- Select project ->properties->build->settings->build artifact (select static or dynamic)
1. Build g++/Eclipse project
    - In Eclipse create a new c++ project (managed or Makefile based)
    - Import or create source code which may have references to CUDA user library functions
      - Add #include to g++ source (e.g. #include "cuda\_process.h")
    - In project build properties add include path to library project
    - e.g. ~/CUDA/cuda\_workspace/cuda\_process
  2. Set linker paths
    - Add link path (-L) to CUDA user library (e.g. ~/CUDA/cuda-workspace/cuda\_process/Release)
    - If linking to a static user library only Add path to CUDA system libraries (/usr/local/cuda/targets/x86\_64-linux/lib)
  3. Add user library to link libraries (-l)
    - e.g. cuda\_process
  - For static user library only also need to include the following libs to avoid a link error on build
    - cudart\_static, cudadevrt, rt ...
    - order may be important and may need other libraries as well
  4. Run g++ project
    1. For link with dynamic library only need to add LD\_LIBRARY\_PATH to the run configuration and set it to the path of the library build object (i.e. the .so file)
      - e.g. LD\_LIBRARY\_PATH=~/CUDA/cuda-workspace/cuda\_process/Release
      - Can also avoid this issue by copying the library ".so" file to an automatically included path (e.g. /usr/x86\_64-linux-gnu)