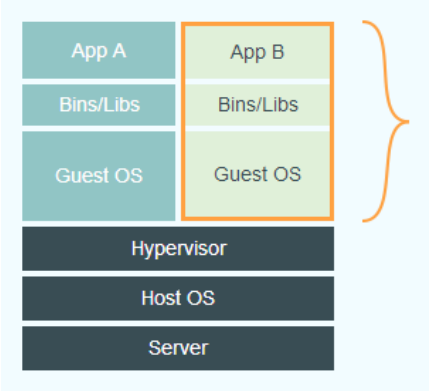
Ubuntu 14.04 + Nvidia Docker를 사용한 GPU enabled Caffe 개발환경 구축

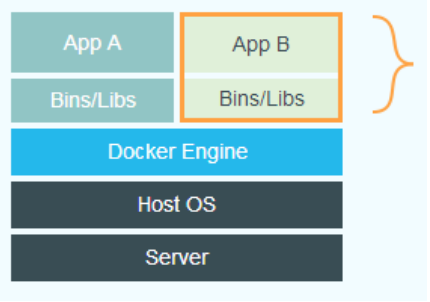
2016.7.19. 진준호

- 최근 업데이트 일자: 2016.8.2.
- 개발환경
    - 호스트: Ubuntu 14.04, Docker 1.11.2, Nvidia-docker 1.0.0-rc
    - 이미지: cmake 3.6.0, gcc 4.8.4, git 1.9.1, cuda 7.5, cudnn5, opencv 3.1.0, python 2.7, miniconda 4.1.11

Docker란?

Docker는 2013년 등장한 새로운 오픈소스 가상화 도구이다. VirtualBox나 Parallels같은 가상머신과 유사한 기능을 가지고 있지만 그 구조의 특징 때문에 훨씬 가벼운 배포가 가능하다는 이점이 있다. 가상머신의 경우, 우리가 원하는 프로그램을 수행하기 위해 guest OS를 반드시 설치해줘야 하지만 Docker는 Linux의 Container(LXC, 최근엔 libcontainer로 LXC를 대체함)를 사용하기 때문에 별도의 OS 설치없이도 가상환경을 구축할 수 있다.





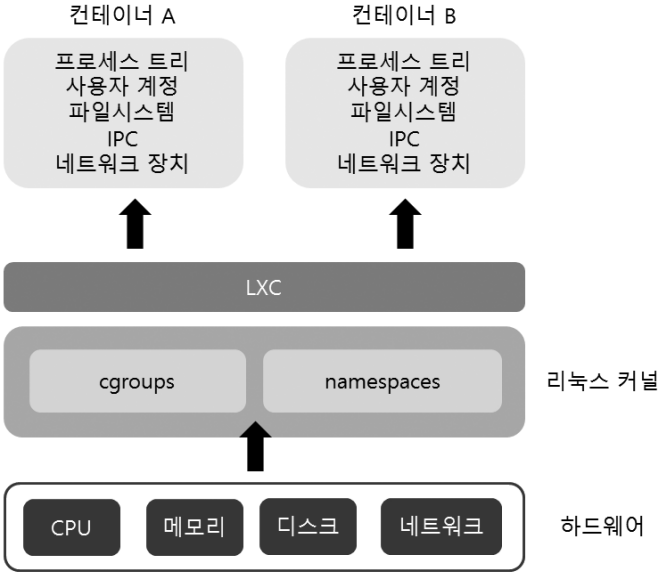
왼쪽 그림과 같이 가상머신은 그 자체로 하나의 완전한 컴퓨터이기 때문에 호스트 머신과는 별개의 OS를 반드시 포함한다. 반면, Docker(오른쪽)는 Host OS로 직접 시스템콜을 날리기 때문에 별도의 OS가 필요하지 않으며, 그 외 호스트와는 별도로 운영하고자 하는 프로그램만 설치 및 관리한다. 이 덕분에 배포 시 이미지 크기가 크게 줄어들 뿐만 아니라 가상화 레이어(하이퍼바이저)가 없기 때문에 파일시스템, 네트워크 속도 등 전반적인 성능이 호스트 머신과 유사할 정도로 향상되었다.

	성능 측정 도구	호스트	Docker
CPU	sysbench	1	0.9945
메모리 쓰기	sysbench	1	0.9826
메모리 읽기	sysbench	1	1.0025
디스크 I/O	dd	1	0.9811
네트워크	iperf	1	0.9626

<호스트 대비 Docker의 디스크, 메모리, 네트워크 IO 성능 지표>

리눅스 컨테이너

어쨌든 Docker도 가상환경을 생성, 실행하기 때문에 호스트 OS 위에 별도의 레이어가 추가적으로 필요하다. 하지만 이 레이어는 가상머신의 하이퍼바이저와는 다르다. 컴퓨터를 통째로 가상화하는 가상머신은 물리적 자원의 할당(Hardware virtualization)이 필요하지만, LXC는 일종의 증강된 chroot로서 가상화보다는 격리의 개념으로 봐야한다. LXC는 리눅스 커널의 cgroups (하드웨어 자원 할당)와 namespaces(계정, 파일시스템 등)을 사용하여 가상 공간을 제공한다.



하지만 위와 같이 LXC는 격리된 공간만을 제공할 뿐 부가적인 기능이 없기 때문에, Docker는 LXC를 기반으로 하여 이미지 및 컨테이너 생성, 관리, 실행 등 다양한 기능을 추가한 레이어를 만들었다. 가상머신의 하이퍼바이저와는 확연히 다른 역할을 수행하는 것을 알 수 있다. 이를 Docker 엔진이라고 하며, 최근에는 LXC대신 libcontainer를 실행 드라이버(exec engine)로 사용한다 (LXC는 lxc로, libcontainer는 native로 표시됨).

Docker 이미지, 컨테이너

Docker 엔진은 Docker 이미지를 생성, 관리, 실행하는 등의 역할을 맡는다. Docker 이미지란 (가상머신의 이미지와 같이) 독립된 가상 개발환경을 의미하며, 크게 부모 이미지가 없는 베이스 이미지와 특정 이미지로부터 파생된 Docker 이미지로 구분된다. 즉, (guest OS가 없기 때문에) host OS의 커널과 중복되는 부분을 제외한 나머지 영역, 그리고 사용자가 설치한 영역을 포함한 것이 Docker 이미지가 된다. 그리고 이러한 이미지를 실행한 것을 컨테이너라고 부른다. 하나의 이미지에는 다수의 컨테이너가 존재할 수 있다.

Host OS와의 중복되는 부분을 제외한다는 것은 다음과 같은 의미를 가진다. 우선 기본적으로 Docker는 리눅스 기반의 이미지 만을 제공한다 (LXC 기반으로 구현되었기 때문). 만약 Ubuntu 14.04 기반의 호스트 머신에서 CentOS Docker 이미지를 만들었다면, 이 이미지에는 CentOS에서 Ubuntu 14.04에 없는 부분들만 패키징될 것이다. 즉, 우분투 호스트에서 우분투 Docker 이미지를 만들 경우 대부분의 자원이 호스트와 똑같기 때문에 유저가 따로 설치/변경한 내용만 패키징 되어 이미지의 크기가 매우 작아질 수 있는 것이다. 이 때, 우분투 Docker 이미지를 실행한 컨테이너 내에서 명령어를 수행할 경우, 우분투 호스트의 커널 프로세스가 실행된다 (즉 Host와 Docker 컨테이너는 Process 공간을 공유한다).

이런 베이스 이미지에 다양한 라이브러리, 프로그램, 소스파일 등을 추가/설치/저장하여 새롭게 만든 이미지를 Docker 이미지라고 하며, 상속 개념과 유사하다고 보면 된다. 이 때 베이스 이미지의 모든 내용을 포함하는 것이 아닌 추가/변형된 부분만 저장되며 이렇게 만들어진 Docker 이미지는 또 상속되어 새로운 Docker 이미지가 될 수 있다.

Docker Hub

Docker는 Github과 같이 이미지의 hub인 Docker Hub을 제공한다. 예를 들어 어떤 이미지를 pull받을 경우 Docker Hub에 public하게 등록되어있는 이미지를 찾아 로컬에 저장한다. 새로운 이미지를 만들고 싶으면 Dockerfile로 직접 빌드하거나 이 Hub에서 이미지를 pull 받아와 사용하면 된다.

왜 Docker를 쓰는가?

파이썬으로 작업하는 경우, miniconda 등의 가상개발환경 툴을 사용하여 독립적인 개발환경을 유지하며 혹시 실수하더라도 시스템 전체가 꼬이지 않게 끔 할 수 있다. 하지만 miniconda는 파이썬에 한정적이기 때문에 Caffe같은 툴을 빌드할 때 사용할 수는 없다. 뿐만 아니라 딥러닝의 경우 다양한 머신에서 테스트할 수도 있고 수많은 변종을 만들어낼 수 있기 때문에 이미지로 보관하는 것이 매우 유용하다. 가상머신보다는 훨씬 가볍게 이미지를 관리할 수 있기 때문에 Docker를 선택하였다.

Docker 설치

Docker 설치는 온라인 상의 Documentation을 참조하여 진행하였다. 우선 설치 전 호스트 머신에 깔린 커널 버전을 알아야 한다.

```
$ uname -r
> 4.2.0-42-generic
```

그리고 아래 단계를 진행하는데, 14.04에서는 대부분 생략해도 상관없다.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

GPG 키를 등록해준 다음

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

`/etc/apt/sources.list.d/docker.list` 파일을 열어서 설치된 우분투 버전에 맞는 내용을 기록한다.

- Ubuntu Trusty 14.04 (LTS)
  `deb https://apt.dockerproject.org/repo ubuntu-trusty main`
- Ubuntu Precise 12.04 (LTS)
  `deb https://apt.dockerproject.org/repo ubuntu-precise main`
- Ubuntu Wily 15.10
  `deb https://apt.dockerproject.org/repo ubuntu-wily main`
- Ubuntu Xenial 16.04 (LTS)
  `deb https://apt.dockerproject.org/repo ubuntu-xenial main`

그 후 apt 패키지를 업데이트한 다음 docker-engine 리포지토리를 확인한다.

```
$ sudo apt-get update
$ apt-cache policy docker-engine
```

아래 단계는 문서상 prerequisite인데 대부분 생략해도 된다.

```
$ sudo apt-get install linux-image-extra-$(uname -r)
```

이제 docker 엔진을 설치해주고 마지막으로 간단한 명령어를 사용하여 제대로 설치가 되었는지 확인해준다.

```
$ sudo apt-get install docker-engine
$ sudo service docker start
$ sudo docker run hello-world
```

hello-world를 실행하면 docker hub로부터 해당 이미지를 받아와 실행할 것이다. 여기서 한가지 추가 작업이 필요한데, 권한 문제 때문에 매번 `sudo docker` 로 명령어를 실행해야하는 불편함이 있다. 따라서 아래와 같이 사용자를 docker 그룹에 등록해준다.

```
$ sudo groupadd docker # 설치 시 docker 그룹이 생성됨
$ sudo usermod -aG docker [your username]
```

그 외 기타 설정에 관해서는 문서를 참조.

Docker 사용법

본 사용법은 nvidia docker 사용 전의 개발환경 구축 과정이지만, 전반적인 docker 사용법을 익히는데 도움이 될 것 같아 추가하였다.

Docker 컨테이너 생성

현재 개발 환경은 ubuntu 14.04 이기 때문에 docker 개발도 해당 버전을 기반으로 한다. 먼저 Docker Hub 에서 Ubuntu 14.04 이미지를 가져온 후, 이 이미지로부터 작업하고자 하는 새로운 컨테이너를 만든다.

```
$ docker pull ubuntu:14.04
> 14.04: Pulling from library/ubuntu
> 96c6a1f3c3b0: Pull complete
> 4767a2d70a73: Pull complete
> 422639bc8a94: Pull complete
> a797489a324a: Pull complete
> Digest: sha256:b2c8a4d46473ab082200880391ddf8c06f2a67da4fa905ce2747dcd95d8d7af7
> Status: Downloaded newer image for ubuntu:14.04
```

그냥 ubuntu 이미지로부터 컨테이너를 만들면 sh 접속 시 항상 working directory가 /로 잡히는데, 홈 디렉토리에서 시작하게끔(그리고 기본적인 프로그램을 미리 설치해두도록) 새로운 이미지를 만들었다. 아래는 새로 만든 이미지로부터 컨테이너를 만드는 명령어들과 이 때 사용한 Dockerfile(이 파일은 새로운 이미지를 만들어내는 설계도 같은 역할을 한다)이다.

```
$ vi Dockerfile                          # ubuntu 이미지를 기반으로 새 이미지 생성
$ docker build -t caffe:junho .          # Dockerfile이 있는 디렉토리에서 실행
$ docker create -it caffe:junho          # 컨테이너 생성
$ docker ps -a                           # 생성된 컨테이너 확인 (image값으로 확인하면 됨)
$ docker rename [생성된 컨테이너 ID] caffe   # 컨테이너 이름 변경
$ docker start caffe                      # 컨테이너 실행
$ docker exec -it caffe /bin/bash         # caffe 컨테이너에서 bash shell 실행
```

Dockerfile:

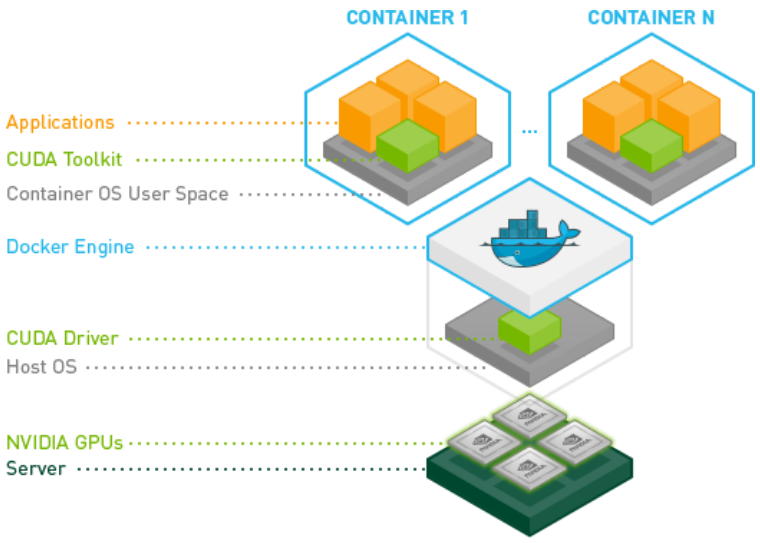
```
FROM ubuntu:14.04
MAINTAINER junho@etri.re.kr

RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    software-properties-common \
    python-software-properties \
    git \
    wget \
    g++ \
    gcc \
    vim \
    zip \
    unzip

WORKDIR /root
```

이후 모든 작업은 exec 명령어 실행 이후 접속된 docker container에서 작업한다.

Docker와 CUDA-enabled GPU 인식 문제

docker를 사용하여 우분투 이미지를 만들고 필요한 모든 패키지를 설치해보았는데, GPU를 인식하지 못해 CPU 버전으로 테스트해야만 했다. 커널 업데이트, 디바이스 재 마운트, 권한 설정, 그래픽 드라이버 재설치 등 다양한 방법으로 해결을 시도해보았고 결국 인식은 성공하였으나, docker image와 host 간의 그래픽 드라이버 버전이 다르면 결국 문제가 생길 수 밖에 없는 구조였다. 이러한 문제를 해결하고자 Nvidia에서 직접 호스트 환경에 구애받지 않고 GPU 관련 작업을 자동으로 처리해주는 nvidia-docker를 만들었다. nvidia-docker를 통해 이미지를 생성하면 별도의 드라이버 설치나 디바이스 마운트 등의 작업 없이도 바로 GPU를 인식하며, 이미지와 호스트 머신 간의 버전 문제도 nvidia-docker-plugin으로 해결하능하다 (참조).

Nvidia Docker 설치



nvidia docker는 크게 nvidia-docker executable과 nvidia/cuda dockerfile들로 구분할 수 있다. Github 공식 페이지(환경/도구 별 dockerfile들이 모여 있음)에서 다음과 같이 본 도구의 장점을 설명하는데 heterogeneous driver/toolkit environments가 가장 큰 장점일 듯 싶다.

- Reproducible builds
- Ease of deployment
- Isolation of individual devices
- Run across heterogeneous driver/toolkit environments
- Requires only the NVIDIA driver to be installed
- Enables “fire and forget” GPU applications
- Facilitate collaboration

Prerequisites

Nvidia Docker 설치에 앞서 필요한 환경을 체크한다. 그래픽 드라이버를 설치한 ubuntu 14.04 시스템 기준 별다른 이상이 없었다.

1. Check kernel version (> 3.10)

```
$ uname -r
> 4.2.0-42-generic
```

2. Check Docker version (>= 1.9)

```
$ docker --version
> Docker version 1.11.2, build b9f10c9
```

3. NVIDIA GPU with Architecture > Fermi (2.1)
4. NVIDIA drivers >= 340.29 with binary nvidia-modprobe

```
$ nvidia-settings -q NvidiaDriverVersion
> Attribute 'NvidiaDriverVersion' (junho-Ubuntu:0.0): 352.63
> Attribute 'NvidiaDriverVersion' (junho-Ubuntu:0[gpu:0]): 352.63
```

Nvidia-Docker 다운 및 설치

nvidia-docker를 받아 설치한다. nvidia-docker는 docker의 모든 명령어를 지원한다. 그래픽 카드 자동 인식/지원 기능을 빼면 docker와 사용법은 똑같다. 설치 후 `run` 명령어를 통해 테스트 컨테이너를 만들고 `nvidia-smi` 를 실행해 그래픽카드가 제대로 인식되는지 확인한다. `run` 명령어는 이미지로부터 바로 컨테이너를 생성하여 제일 마지막 인자를 명령어로 받아 생성된 컨테이너에서 실행해준다. `--rm` 옵션으로 nvidia-smi 실행 후 테스트한 컨테이너를 없애준다.

```
$ wget -P /tmp https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.0-rc.3/nvidia-docker_1.0.0.rc.3-1_amd64.deb
$ sudo dpkg -i /tmp/nvidia-docker*.deb && rm /tmp/nvidia-docker*.deb
$ nvidia-docker run --rm nvidia/cuda nvidia-smi
> Fri Jul 29 00:37:53 2016
```

```
+-----------------------------------------------------+
| NVIDIA-SMI 352.63     Driver Version: 352.63        |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX TIT...  Off  | 0000:01:00.0      On |                  N/A |
| 22%   38C    P8    17W / 250W |    396MiB / 12287MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

GPU 인식된 docker container 생성하기

이제 설치된 nvidia-docker로 새로운 컨테이너를 생성한다. 이렇게 생성된 컨테이너는 별도의 작업 수행 없이도 gpu 인식이 된다. 아래와 같이 nvidia/cuda 이미지로부터 바로 컨테이너를 생성한다.

```
$ nvidia-docker run -it --privileged -w /root/ nvidia/cuda:7.5-devel /bin/bash
```

container 접속 후 Caffe 설치 준비

모든 소스는 /opt 디렉토리 안의 sources 폴더 안에 다운로드했다. 생성 후 바로 접속한 컨테이너에는 필수적인 패키지들이 설치되어있지 않은 상태이므로 opencv나 caffe를 빌드하기 전에 우선 설치해준다.

기본 설정 및 패키지 설치

```
$ apt-get update
$ uname -r    # Install linux image and header
- [ ] $ apt-get install -y linux-image-4.2.0-42-generic    # 4.2.0-42-generic 은 uname -r 실행 결과에 맞게 변경, grub 설치 안함
$ apt-get install -y linux-headers-4.2.0-42-generic
$ apt-get update    # Basic Setup
$ apt-get install -y --no-install-recommends \
  build-essential \
  software-properties-common \
  bash-completion \
  pkg-config \
  python-dev \
  git \
  gcc \
  g++ \
  vim \
  make \
  wget \
  curl \
  unzip
$ apt-get clean
$ apt-get autoremove
$ rm -rf /var/lib/apt/lists/*
$ cd /opt && mkdir sources && cd sources    # Workspace Setup
```

cmake (v3.6.0) 설치

소스를 컴파일한 후 update-alternatives로 cmake 패키지를 등록한다. 기본 설치 위치가 /usr/local/bin이기 때문에 cmake 호출시 해당 위치로 링크를 시켜준다.

```
$ cd /opt/sources && mkdir cmake-3.6.0 && cd cmake-3.6.0
$ wget --no-check-certificate http://cmake.org/files/v3.6/cmake-3.6.0.tar.gz
```

```
tar -zxvf cmake-3.6.0.tar.gz
$ cd cmake-3.6.0/
$ ./bootstrap
$ make -j 20
$ make install
$ update-alternatives --install /usr/bin/cmake cmake /usr/local/bin/cmake 1
```

cmake 호출로 정상 설치 확인.

Cuda & Cudnn 설치

Cuda 7.5

설치 후 bashrc 파일에 환경변수를 등록해준다. 설치 후 `nvcc --version` 으로 정상 설치 확인.

```
$ cd /opt/sources && mkdir cuda-7.5 && cd cuda-7.5
$ wget http://developer.download.nvidia.com/compute/cuda/7.5/Prod/local_installers/cuda-repo-ubuntu1404-7-5-local
_7.5-18_amd64.deb
$ dpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
$ apt-get update
$ apt-get install -y cuda
$ echo -e '#Cuda Configuration
  export CUDA_HOME=/usr/local/cuda-7.5
  export LD_LIBRARY_PATH=${CUDA_HOME}/lib64
  PATH=${CUDA_HOME}/bin:${PATH}
  export PATH' >> ~/.bashrc
$ source ~/.bashrc
```

Cudnn5

```
$ cd /opt/sources && mkdir cudnn && cd cudnn
$ wget 'http://developer.download.nvidia.com/compute/redist/cudnn/v5/cudnn-7.5-linux-x64-v5.0-ga.tgz' -O cudnn-7.
5-linux-x64-v5.0-ga.tgz
$ tar -zvxf cudnn-7.5-linux-x64-v5.0-ga.tgz
$ cp cuda/include/* /usr/local/cuda-7.5/include/
$ cp cuda/lib64/* /usr/local/cuda-7.5/lib64/
```

OpenCV 3.1.0 설치

opencv 설치에 앞서 필요한 패키지를 확인하고 없는 것들을 설치해준다.

- GCC 4.4.x or later
- CMake 2.6 or higher
- Git
- GTK+2.x or higher, including headers (libgtk2.0-dev)
- pkg-config
- Python 2.6 or later and Numpy 1.5 or later with developer packages (python-dev, python-numpy)
- ffmpeg or libav development packages: libavcodec-dev, libavformat-dev, libswscale-dev
- [optional] libtbb2 libtbb-dev
- [optional] libdc1394 2.x
- [optional] libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev

```
$ apt-get install -y --no-install-recommends \
  checkinstall \
  yasm \
  libgtk2.0-dev \
  libavcodec-dev \
  libavformat-dev \
  libswscale-dev \
  python-dev \
  python-numpy \
  libtbb2 \
  libtbb-dev \
  libtiff-dev \
  libjpeg-dev \
```

```
  libpng-dev \
  libdc1394-22-dev \
  libxine-dev \
  libgstreamer0.10-dev \
  libgstreamer-plugins-base0.10-dev \
  libv4l-dev \
  libmp3lame-dev \
  libopencore-amrnb-dev \
  libopencore-amrwb-dev \
  libtheora-dev \
  libvorbis-dev \
  libxvidcore-dev \
  x264 \
  v4l-utils \
  libopenexr-dev \
  python-tk \
  libeigen3-dev \
  libx264-dev
$ apt-add-repository ppa:mc3man/trusty-media
$ apt-get dist-upgrade && apt-get update
$ apt-get install -y ffmpeg gstreamer0.10-ffmpeg
$ cd /opt/sources && mkdir opencv-3.1.0 && cd opencv-3.1.0
$ wget http://downloads.sourceforge.net/project/opencvlibrary/opencv-unix/3.1.0/opencv-3.1.0.zip -O source.zip
$ unzip source.zip
$ mkdir release && cd release
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D -WITH_OPENGL=ON -D INSTALL_C_EXAMPLES=ON \
        -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_TBB=ON -D WITH_IPP=OFF ../opencv-3.1.0
$ make -j 20 && make install
$ echo "/usr/local/lib" >> /etc/ld.so.conf.d/opencv.conf && ldconfig
```

miniconda 설치

Caffe 설치 시 python 모듈의 독립적 개발환경 세팅을 위해 miniconda를 설치해준다. 설치 시 뜨는 대화메시지에 아래와 같이 입력한다.

```
$ cd /opt/sources && mkdir miniconda && cd miniconda
$ wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
$ chmod u+x Miniconda2-latest-Linux-x86_64.sh
$ ./Miniconda2-latest-Linux-x86_64.sh
# [enter] to continue
# [yes] to license
# change root to [/root/.miniconda2]
# [yes] to bashrc modification
```

miniconda 설치 후 caffe라는 이름의 새로운 개발환경을 생성한다.

```
$ conda create -n caffe python
$ . activate caffe    # 생성한 환경 실행
```

caffe 설치

필요한 패키지들을 설치한 후 github에서 소스를 다운받아 make 빌드한다. 이 때 기본적인 Makefile configuration이 적혀있는 Makefile.confi.example을 복사하여 자신의 환경에 맞게 적절히 수정해준다. pip로 설치되는 파이썬 모듈들은 caffe라는 이름의 conda 환경에 설치되어 시스템 레벨의 파이썬 모듈과 독립적으로 운용된다.

```
$ apt-get install -y --no-install-recommends \
  libgoogle-glog-dev \
  libhdf5-serial-dev \
  libatlas-base-dev \
  protobuf-compiler \
  libboost-all-dev \
  libprotobuf-dev \
  libleveldb-dev \
  libgflags-dev \
  libsnappy-dev \
  liblmdb-dev \
```

```
  python-dev \
  python-pip \
  python-numpy \
  python-scipy && \
$ rm -rf /var/lib/apt/lists/*
$ CLONE_TAG=master
# bashrc 파일 수정 (마지막에 추가)
> CAFFE_ROOT=/opt/sources/caffe
$ cd /opt/sources && mkdir caffe && cd caffe
$ git clone -b ${CLONE_TAG} --depth 1 https://github.com/BVLC/caffe.git .
$ for req in $(cat python/requirements.txt) pydot; do pip install $req; done
$ cp Makefile.config.example Makefile.config # modify makefile
$ vi Makefile.config
```

본 설치에서는 opencv 3 버전 사용, conda 사용, cudnn 사용 등을 반영하기 위해 다음과 같이 Makefile.config를 수정하였다.

```
..
 5 USE_CUDNN := 1    # 주석 제거
..
21 OPENCV_VERSION := 3    # 주석 제거
..
64 # PYTHON_INCLUDE := /usr/include/python2.7 \    # 원래코드 주석처리
65 #     /usr/lib/python2.7/dist-packages/numpy/core/include
..
68 ANACONDA_HOME := $(HOME)/.miniconda2    # 주석 제거 후 conda 환경 지정
69 PYTHON_INCLUDE := $(ANACONDA_HOME)/include \
70     $(ANACONDA_HOME)/include/python2.7 \
71     $(ANACONDA_HOME)/lib/python2.7/site-packages/numpy/core/include \
..
79 # PYTHON_LIB := /usr/lib    # 주석 처리
80 PYTHON_LIB := $(ANACONDA_HOME)/lib    # 주석 제거
..
```

이후 make로 pycaffe 빌드 및 환경변수를 설정해준다.

```
$ make all
$ make pycaffe
$ make test
$ make runtest
# bashrc 파일 수정 (마지막에 추가)
> # Caffe Environment Variables (.bashrc)
> export CAFFE_ROOT=/opt/sources/caffe
> export PYCAFFE_ROOT=$CAFFE_ROOT/python
> export PYTHONPATH=$PYCAFFE_ROOT:$PYTHONPATH
> export PATH=$CAFFE_ROOT/build/tools:$PYCAFFE_ROOT:$PATH
$ echo "$CAFFE_ROOT/build/lib" >> /etc/ld.so.conf.d/caffe.conf && ldconfig
```

caffe 테스트 수행

설치한 caffe가 실제로 잘 동작하는지 확인하기 위해 python 디렉토리 안의 `classify.py` 를 직접 실행해본다. 에러가 날 경우 파이썬 모듈을 제대로 설치하지 않았거나 conda 환경을 켜지 않았을 확률이 높으니 먼저 확인해보자. 먼저 caffe 설치 디렉토리에서 ImageNet Caffe model과 label을 다운로드 받기 위해 다음 명령어를 실행한다.

```
$ python scripts/download_model_binary.py models/bvlc_reference_caffenet
$ . data/ilsvrc12/get_ilsvrc_aux.sh
```

이후 바로 `python/classify.py` 파일을 실행해줘도 상관없으나 좀 더 효과적인 출력을 보기 위해 파일을 다음과 같이 수정한다.

```
...
import pandas

...
parser.add_argument(
  "--print_results",
  action='store_true',
  help="Write output text to stdout rather than serializing to a file."
```

```
)
parser.add_argument(
  "--labels_file",
  default=os.path.join(pycaffe_dir,"../data/ilsvrc12/synset_words.txt"),
  help="Readable label definition file."
)
...

#Classify
start = time.time()
scores = classifier.predict(inputs, not args.center_only).flatten()
print("Done in %.2f s." % (time.time() - start))

if args.print_results:
  with open(args.labels_file) as f:
    labels_df = pandas.DataFrame([{'synset_id':l.strip().split(' ')[0], 'name': ' '.join(l.strip().split(' ')[1:]
).split(',')[0]} for l in f.readlines()])
    labels = labels_df.sort('synset_id')['name'].values

    indices =(-scores).argsort()[:5]
    predictions = labels[indices]

        meta = [(p, '%.5f' % scores[i]) for i,p in zip(indices, predictions)]
    print meta

#Save
...
```

이 때 `ValueError: Mean shape incompatible with input shape` 버그가 발생할 수 있는데, stackoverflow에 따르면 contributor의 코딩실수로 인한 문제로 보고있다. 다음과 같이 `python/caffe/io.py` 의 내용을 수정하면 된다.

```
if ms != self.inputs[in_][1:]:
    raise ValueError('Mean shape incompatible with input shape.')
```

를 지우고

```
if ms != self.inputs[in_] :
   print(self.inputs[in_])
   in_shape = self.inputs[in_][1:]
   m_min, m_max = mean.min(), mean.max()
   normal_mean = (mean - m_min) / (m_max - m_min)
   mean = resize_image(normal_mean.transpose((1,2,0)), in_shape[1:]).transpose((2,0,1)) * (m_max - m_min) + m_min
```

를 추가한다. 이후, 아래와 같이 `classify.py` 를 실행한 후 결과를 확인한다.

```
$ python python/classify.py --gpu --print_results examples/images/cat.jpg foo
GPU mode
WARNING: Logging before InitGoogleLogging() is written to STDERR
W0802 02:24:38.357357 11861 _caffe.cpp:122] DEPRECATION WARNING - deprecated use of Python interface
W0802 02:24:38.357372 11861 _caffe.cpp:123] Use this instead (with the named "weights" parameter):
W0802 02:24:38.357379 11861 _caffe.cpp:125] Net('python/../models/bvlc_reference_caffenet/deploy.prototxt', 1, we
ights='python/../models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel')
I0802 02:24:38.358530 11861 net.cpp:58] Initializing net from parameters:
name: "CaffeNet"
state {
  phase: TEST
  level: 0
}
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 10
      dim: 3
      dim: 227
```

```
      dim: 227
    }
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm1"
  type: "LRN"
  bottom: "pool1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "norm1"
  top: "conv2"
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
```

```
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm2"
  type: "LRN"
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "norm2"
  top: "conv3"
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    group: 2
  }
}
layer {
  name: "relu4"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
}
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    group: 2
  }
}
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "conv5"
  top: "conv5"
}
```

```
layer {
  name: "pool5"
  type: "Pooling"
  bottom: "conv5"
  top: "pool5"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "fc6"
  type: "InnerProduct"
  bottom: "pool5"
  top: "fc6"
  inner_product_param {
    num_output: 4096
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "fc6"
  top: "fc6"
}
layer {
  name: "drop6"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layer {
  name: "fc7"
  type: "InnerProduct"
  bottom: "fc6"
  top: "fc7"
  inner_product_param {
    num_output: 4096
  }
}
layer {
  name: "relu7"
  type: "ReLU"
  bottom: "fc7"
  top: "fc7"
}
layer {
  name: "drop7"
  type: "Dropout"
  bottom: "fc7"
  top: "fc7"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layer {
  name: "fc8"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8"
  inner_product_param {
    num_output: 1000
  }
}
```

```
layer {
  name: "prob"
  type: "Softmax"
  bottom: "fc8"
  top: "prob"
}
I0802 02:24:38.359123 11861 layer_factory.hpp:77] Creating layer data
I0802 02:24:38.359136 11861 net.cpp:100] Creating Layer data
I0802 02:24:38.359143 11861 net.cpp:408] data -> data
I0802 02:24:38.370014 11861 net.cpp:150] Setting up data
I0802 02:24:38.370044 11861 net.cpp:157] Top shape: 10 3 227 227 (1545870)
I0802 02:24:38.370054 11861 net.cpp:165] Memory required for data: 6183480
I0802 02:24:38.370061 11861 layer_factory.hpp:77] Creating layer conv1
I0802 02:24:38.370077 11861 net.cpp:100] Creating Layer conv1
I0802 02:24:38.370084 11861 net.cpp:434] conv1 <- data
I0802 02:24:38.370092 11861 net.cpp:408] conv1 -> conv1
I0802 02:24:38.478287 11861 net.cpp:150] Setting up conv1
I0802 02:24:38.478315 11861 net.cpp:157] Top shape: 10 96 55 55 (2904000)
I0802 02:24:38.478320 11861 net.cpp:165] Memory required for data: 17799480
I0802 02:24:38.478332 11861 layer_factory.hpp:77] Creating layer relu1
I0802 02:24:38.478343 11861 net.cpp:100] Creating Layer relu1
I0802 02:24:38.478348 11861 net.cpp:434] relu1 <- conv1
I0802 02:24:38.478353 11861 net.cpp:395] relu1 -> conv1 (in-place)
I0802 02:24:38.478531 11861 net.cpp:150] Setting up relu1
I0802 02:24:38.478543 11861 net.cpp:157] Top shape: 10 96 55 55 (2904000)
I0802 02:24:38.478549 11861 net.cpp:165] Memory required for data: 29415480
I0802 02:24:38.478554 11861 layer_factory.hpp:77] Creating layer pool1
I0802 02:24:38.478560 11861 net.cpp:100] Creating Layer pool1
I0802 02:24:38.478564 11861 net.cpp:434] pool1 <- conv1
I0802 02:24:38.478570 11861 net.cpp:408] pool1 -> pool1
I0802 02:24:38.478606 11861 net.cpp:150] Setting up pool1
I0802 02:24:38.478613 11861 net.cpp:157] Top shape: 10 96 27 27 (699840)
I0802 02:24:38.478618 11861 net.cpp:165] Memory required for data: 32214840
I0802 02:24:38.478622 11861 layer_factory.hpp:77] Creating layer norm1
I0802 02:24:38.478631 11861 net.cpp:100] Creating Layer norm1
I0802 02:24:38.478637 11861 net.cpp:434] norm1 <- pool1
I0802 02:24:38.478646 11861 net.cpp:408] norm1 -> norm1
I0802 02:24:38.478759 11861 net.cpp:150] Setting up norm1
I0802 02:24:38.478765 11861 net.cpp:157] Top shape: 10 96 27 27 (699840)
I0802 02:24:38.478770 11861 net.cpp:165] Memory required for data: 35014200
I0802 02:24:38.478775 11861 layer_factory.hpp:77] Creating layer conv2
I0802 02:24:38.478782 11861 net.cpp:100] Creating Layer conv2
I0802 02:24:38.478786 11861 net.cpp:434] conv2 <- norm1
I0802 02:24:38.478791 11861 net.cpp:408] conv2 -> conv2
I0802 02:24:38.480346 11861 net.cpp:150] Setting up conv2
I0802 02:24:38.480360 11861 net.cpp:157] Top shape: 10 256 27 27 (1866240)
I0802 02:24:38.480365 11861 net.cpp:165] Memory required for data: 42479160
I0802 02:24:38.480373 11861 layer_factory.hpp:77] Creating layer relu2
I0802 02:24:38.480381 11861 net.cpp:100] Creating Layer relu2
I0802 02:24:38.480384 11861 net.cpp:434] relu2 <- conv2
I0802 02:24:38.480391 11861 net.cpp:395] relu2 -> conv2 (in-place)
I0802 02:24:38.480499 11861 net.cpp:150] Setting up relu2
I0802 02:24:38.480506 11861 net.cpp:157] Top shape: 10 256 27 27 (1866240)
I0802 02:24:38.480510 11861 net.cpp:165] Memory required for data: 49944120
I0802 02:24:38.480515 11861 layer_factory.hpp:77] Creating layer pool2
I0802 02:24:38.480523 11861 net.cpp:100] Creating Layer pool2
I0802 02:24:38.480528 11861 net.cpp:434] pool2 <- conv2
I0802 02:24:38.480533 11861 net.cpp:408] pool2 -> pool2
I0802 02:24:38.480562 11861 net.cpp:150] Setting up pool2
I0802 02:24:38.480568 11861 net.cpp:157] Top shape: 10 256 13 13 (432640)
I0802 02:24:38.480572 11861 net.cpp:165] Memory required for data: 51674680
I0802 02:24:38.480576 11861 layer_factory.hpp:77] Creating layer norm2
I0802 02:24:38.480586 11861 net.cpp:100] Creating Layer norm2
I0802 02:24:38.480592 11861 net.cpp:434] norm2 <- pool2
I0802 02:24:38.480597 11861 net.cpp:408] norm2 -> norm2
I0802 02:24:38.480789 11861 net.cpp:150] Setting up norm2
I0802 02:24:38.480798 11861 net.cpp:157] Top shape: 10 256 13 13 (432640)
I0802 02:24:38.480803 11861 net.cpp:165] Memory required for data: 53405240
```

```
I0802 02:24:38.480808 11861 layer_factory.hpp:77] Creating layer conv3
I0802 02:24:38.480815 11861 net.cpp:100] Creating Layer conv3
I0802 02:24:38.480821 11861 net.cpp:434] conv3 <- norm2
I0802 02:24:38.480829 11861 net.cpp:408] conv3 -> conv3
I0802 02:24:38.482446 11861 net.cpp:150] Setting up conv3
I0802 02:24:38.482462 11861 net.cpp:157] Top shape: 10 384 13 13 (648960)
I0802 02:24:38.482467 11861 net.cpp:165] Memory required for data: 56001080
I0802 02:24:38.482476 11861 layer_factory.hpp:77] Creating layer relu3
I0802 02:24:38.482484 11861 net.cpp:100] Creating Layer relu3
I0802 02:24:38.482491 11861 net.cpp:434] relu3 <- conv3
I0802 02:24:38.482496 11861 net.cpp:395] relu3 -> conv3 (in-place)
I0802 02:24:38.482604 11861 net.cpp:150] Setting up relu3
I0802 02:24:38.482612 11861 net.cpp:157] Top shape: 10 384 13 13 (648960)
I0802 02:24:38.482616 11861 net.cpp:165] Memory required for data: 58596920
I0802 02:24:38.482621 11861 layer_factory.hpp:77] Creating layer conv4
I0802 02:24:38.482630 11861 net.cpp:100] Creating Layer conv4
I0802 02:24:38.482635 11861 net.cpp:434] conv4 <- conv3
I0802 02:24:38.482641 11861 net.cpp:408] conv4 -> conv4
I0802 02:24:38.484421 11861 net.cpp:150] Setting up conv4
I0802 02:24:38.484434 11861 net.cpp:157] Top shape: 10 384 13 13 (648960)
I0802 02:24:38.484439 11861 net.cpp:165] Memory required for data: 61192760
I0802 02:24:38.484447 11861 layer_factory.hpp:77] Creating layer relu4
I0802 02:24:38.484455 11861 net.cpp:100] Creating Layer relu4
I0802 02:24:38.484462 11861 net.cpp:434] relu4 <- conv4
I0802 02:24:38.484469 11861 net.cpp:395] relu4 -> conv4 (in-place)
I0802 02:24:38.484576 11861 net.cpp:150] Setting up relu4
I0802 02:24:38.484585 11861 net.cpp:157] Top shape: 10 384 13 13 (648960)
I0802 02:24:38.484588 11861 net.cpp:165] Memory required for data: 63788600
I0802 02:24:38.484594 11861 layer_factory.hpp:77] Creating layer conv5
I0802 02:24:38.484603 11861 net.cpp:100] Creating Layer conv5
I0802 02:24:38.484609 11861 net.cpp:434] conv5 <- conv4
I0802 02:24:38.484616 11861 net.cpp:408] conv5 -> conv5
I0802 02:24:38.486296 11861 net.cpp:150] Setting up conv5
I0802 02:24:38.486307 11861 net.cpp:157] Top shape: 10 256 13 13 (432640)
I0802 02:24:38.486311 11861 net.cpp:165] Memory required for data: 65519160
I0802 02:24:38.486320 11861 layer_factory.hpp:77] Creating layer relu5
I0802 02:24:38.486326 11861 net.cpp:100] Creating Layer relu5
I0802 02:24:38.486330 11861 net.cpp:434] relu5 <- conv5
I0802 02:24:38.486335 11861 net.cpp:395] relu5 -> conv5 (in-place)
I0802 02:24:38.486443 11861 net.cpp:150] Setting up relu5
I0802 02:24:38.486450 11861 net.cpp:157] Top shape: 10 256 13 13 (432640)
I0802 02:24:38.486454 11861 net.cpp:165] Memory required for data: 67249720
I0802 02:24:38.486459 11861 layer_factory.hpp:77] Creating layer pool5
I0802 02:24:38.486466 11861 net.cpp:100] Creating Layer pool5
I0802 02:24:38.486470 11861 net.cpp:434] pool5 <- conv5
I0802 02:24:38.486475 11861 net.cpp:408] pool5 -> pool5
I0802 02:24:38.486506 11861 net.cpp:150] Setting up pool5
I0802 02:24:38.486512 11861 net.cpp:157] Top shape: 10 256 6 6 (92160)
I0802 02:24:38.486517 11861 net.cpp:165] Memory required for data: 67618360
I0802 02:24:38.486521 11861 layer_factory.hpp:77] Creating layer fc6
I0802 02:24:38.486529 11861 net.cpp:100] Creating Layer fc6
I0802 02:24:38.486533 11861 net.cpp:434] fc6 <- pool5
I0802 02:24:38.486538 11861 net.cpp:408] fc6 -> fc6
I0802 02:24:38.529418 11861 net.cpp:150] Setting up fc6
I0802 02:24:38.529444 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.529448 11861 net.cpp:165] Memory required for data: 67782200
I0802 02:24:38.529458 11861 layer_factory.hpp:77] Creating layer relu6
I0802 02:24:38.529465 11861 net.cpp:100] Creating Layer relu6
I0802 02:24:38.529469 11861 net.cpp:434] relu6 <- fc6
I0802 02:24:38.529475 11861 net.cpp:395] relu6 -> fc6 (in-place)
I0802 02:24:38.529718 11861 net.cpp:150] Setting up relu6
I0802 02:24:38.529728 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.529732 11861 net.cpp:165] Memory required for data: 67946040
I0802 02:24:38.529736 11861 layer_factory.hpp:77] Creating layer drop6
I0802 02:24:38.529747 11861 net.cpp:100] Creating Layer drop6
I0802 02:24:38.529750 11861 net.cpp:434] drop6 <- fc6
I0802 02:24:38.529757 11861 net.cpp:395] drop6 -> fc6 (in-place)
I0802 02:24:38.529783 11861 net.cpp:150] Setting up drop6
```

```
I0802 02:24:38.529790 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.529794 11861 net.cpp:165] Memory required for data: 68109880
I0802 02:24:38.529800 11861 layer_factory.hpp:77] Creating layer fc7
I0802 02:24:38.529808 11861 net.cpp:100] Creating Layer fc7
I0802 02:24:38.529813 11861 net.cpp:434] fc7 <- fc6
I0802 02:24:38.529819 11861 net.cpp:408] fc7 -> fc7
I0802 02:24:38.549372 11861 net.cpp:150] Setting up fc7
I0802 02:24:38.549398 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.549402 11861 net.cpp:165] Memory required for data: 68273720
I0802 02:24:38.549412 11861 layer_factory.hpp:77] Creating layer relu7
I0802 02:24:38.549419 11861 net.cpp:100] Creating Layer relu7
I0802 02:24:38.549424 11861 net.cpp:434] relu7 <- fc7
I0802 02:24:38.549429 11861 net.cpp:395] relu7 -> fc7 (in-place)
I0802 02:24:38.549681 11861 net.cpp:150] Setting up relu7
I0802 02:24:38.549692 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.549695 11861 net.cpp:165] Memory required for data: 68437560
I0802 02:24:38.549700 11861 layer_factory.hpp:77] Creating layer drop7
I0802 02:24:38.549705 11861 net.cpp:100] Creating Layer drop7
I0802 02:24:38.549710 11861 net.cpp:434] drop7 <- fc7
I0802 02:24:38.549713 11861 net.cpp:395] drop7 -> fc7 (in-place)
I0802 02:24:38.549733 11861 net.cpp:150] Setting up drop7
I0802 02:24:38.549742 11861 net.cpp:157] Top shape: 10 4096 (40960)
I0802 02:24:38.549746 11861 net.cpp:165] Memory required for data: 68601400
I0802 02:24:38.549751 11861 layer_factory.hpp:77] Creating layer fc8
I0802 02:24:38.549757 11861 net.cpp:100] Creating Layer fc8
I0802 02:24:38.549763 11861 net.cpp:434] fc8 <- fc7
I0802 02:24:38.549770 11861 net.cpp:408] fc8 -> fc8
I0802 02:24:38.554293 11861 net.cpp:150] Setting up fc8
I0802 02:24:38.554322 11861 net.cpp:157] Top shape: 10 1000 (10000)
I0802 02:24:38.554325 11861 net.cpp:165] Memory required for data: 68641400
I0802 02:24:38.554333 11861 layer_factory.hpp:77] Creating layer prob
I0802 02:24:38.554342 11861 net.cpp:100] Creating Layer prob
I0802 02:24:38.554347 11861 net.cpp:434] prob <- fc8
I0802 02:24:38.554353 11861 net.cpp:408] prob -> prob
I0802 02:24:38.554538 11861 net.cpp:150] Setting up prob
I0802 02:24:38.554548 11861 net.cpp:157] Top shape: 10 1000 (10000)
I0802 02:24:38.554551 11861 net.cpp:165] Memory required for data: 68681400
I0802 02:24:38.554556 11861 net.cpp:228] prob does not need backward computation.
I0802 02:24:38.554561 11861 net.cpp:228] fc8 does not need backward computation.
I0802 02:24:38.554565 11861 net.cpp:228] drop7 does not need backward computation.
I0802 02:24:38.554569 11861 net.cpp:228] relu7 does not need backward computation.
I0802 02:24:38.554574 11861 net.cpp:228] fc7 does not need backward computation.
I0802 02:24:38.554577 11861 net.cpp:228] drop6 does not need backward computation.
I0802 02:24:38.554581 11861 net.cpp:228] relu6 does not need backward computation.
I0802 02:24:38.554585 11861 net.cpp:228] fc6 does not need backward computation.
I0802 02:24:38.554590 11861 net.cpp:228] pool5 does not need backward computation.
I0802 02:24:38.554594 11861 net.cpp:228] relu5 does not need backward computation.
I0802 02:24:38.554600 11861 net.cpp:228] conv5 does not need backward computation.
I0802 02:24:38.554606 11861 net.cpp:228] relu4 does not need backward computation.
I0802 02:24:38.554613 11861 net.cpp:228] conv4 does not need backward computation.
I0802 02:24:38.554620 11861 net.cpp:228] relu3 does not need backward computation.
I0802 02:24:38.554627 11861 net.cpp:228] conv3 does not need backward computation.
I0802 02:24:38.554635 11861 net.cpp:228] norm2 does not need backward computation.
I0802 02:24:38.554642 11861 net.cpp:228] pool2 does not need backward computation.
I0802 02:24:38.554648 11861 net.cpp:228] relu2 does not need backward computation.
I0802 02:24:38.554653 11861 net.cpp:228] conv2 does not need backward computation.
I0802 02:24:38.554658 11861 net.cpp:228] norm1 does not need backward computation.
I0802 02:24:38.554663 11861 net.cpp:228] pool1 does not need backward computation.
I0802 02:24:38.554669 11861 net.cpp:228] relu1 does not need backward computation.
I0802 02:24:38.554675 11861 net.cpp:228] conv1 does not need backward computation.
I0802 02:24:38.554682 11861 net.cpp:228] data does not need backward computation.
I0802 02:24:38.554687 11861 net.cpp:270] This network produces output prob
I0802 02:24:38.554702 11861 net.cpp:283] Network initialization done.
I0802 02:24:38.771664 11861 upgrade_proto.cpp:43] Attempting to upgrade input file specified using deprecated tra
nsformation parameters: python/../models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I0802 02:24:38.771689 11861 upgrade_proto.cpp:46] Successfully upgraded file specified using deprecated data tran
sformation parameters.
W0802 02:24:38.771693 11861 upgrade_proto.cpp:48] Note that future Caffe releases will only support transform_par
```

```
am messages for transformation fields.
I0802 02:24:38.771697 11861 upgrade_proto.cpp:52] Attempting to upgrade input file specified using deprecated V1L
ayerParameter: python/../models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
I0802 02:24:38.852319 11861 upgrade_proto.cpp:60] Successfully upgraded file specified using deprecated V1LayerPa
rameter
I0802 02:24:38.882505 11861 net.cpp:761] Ignoring source layer loss
(10, 3, 227, 227)
Loading file: examples/images/cat.jpg
Classifying 1 inputs.
Done in 0.04 s.
[('tabby', '0.27934'), ('tiger cat', '0.21915'), ('Egyptian cat', '0.16064'), ('lynx', '0.12844'), ('kit fox', '0
.05155')]
Saving results into foo
```

실행 결과로 아래와 같이 출력되는 것과 `foo.npy` 가 만들어지는 것을 확인한다.

Docker Hub에 등록하기

위 과정까지 마무리하며 caffe 개발환경 구축을 완료하였다. 쉽게 이 개발환경을 공유, 사용할 수 있도록 Docker Hub에 이 개발환경을 이미지로 등록한다. 하지만, 실제로는 모든 과정을 거친 후 저장한 이미지의 사이즈가 너무 커서 직접 push하지 못했고, 단계 별로 나눠서 여러 개의 이미지로 저장했다. (cuda 설치 버전, opencv 설치버전, caffe 설치버전 총 3 개의 태그로 구분하여 이미지 생성) opencv 2.4.x 버전을 쓰고싶다면 cuda tag를 달아서 pull 받으면 되고(`docker pull junho/caffe:cuda7.5`), 다른 caffe branch로 작업하고 싶다면 opencv tag를 pull 받아와서 새로 작업하면 된다. 각 설치 방법은 버전만 다를 뿐 위의 설명과 큰 차이는 없을 것이다.

```
$ exit # docker 컨테이너에서 나온다
$ docker ps -a # docker container 확인
$ docker commit -m "Installed Caffe for Deep Learning" -a "Junho Jin" caffe junho/caffe:caffe
$ docker images # 생성된 이미지 확인
$ docker login # docker hub에 로그인, 아이디와 비밀번호 입력 필요
$ docker search junho/caffe # 이미 같은 이름의 이미지가 등록되어있는지 확인한다
$ docker push junho/caffe
$ docker images
> REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
> junho/caffe         caffe               5e90fdb3e3be        27 minutes ago      7.678 GB
> junho/caffe         opencv              d7b4e06f26a7        About an hour ago   6.359 GB
> junho/caffe         cuda7.5             f3e153189c84        About an hour ago   5.723 GB
> junho/caffe         latest              92d181b8db04        3 hours ago         1.251 GB
> nvidia/cuda         7.5-devel           248b8f2365f4        6 days ago          1.229 GB
```

Possible Issues

docker 컨테이너 내부에서 apt-get update 시 문제

`W: Size of file /var/lib/apt/lists/archive.ubuntu.com_ubuntu_dists_trusty-updates_main_binary-amd64_Packages.gz is not what the server reported 1002152 1002196` 와 같은 메시지 출력 시 아래과 같이 해결

```
$ sudo rm /var/lib/apt/lists/* -vf
$ sudo apt-get clean
$ sudo apt-get autoremove
$ sudo apt-get update
$ sudo apt-get dist-upgrade
```