Deep Learning Practice with Caffe

Kye-Hyeon Kim
Computer Vision and Deep Learning Software Group (CVDS)
Intel
kyehyeon.mlg@gmail.com, kye-hyeon.kim@intel.com

Slides & Example Codes

- https://github.com/kyehyeon/caffe-materials
  - Click 

Contents

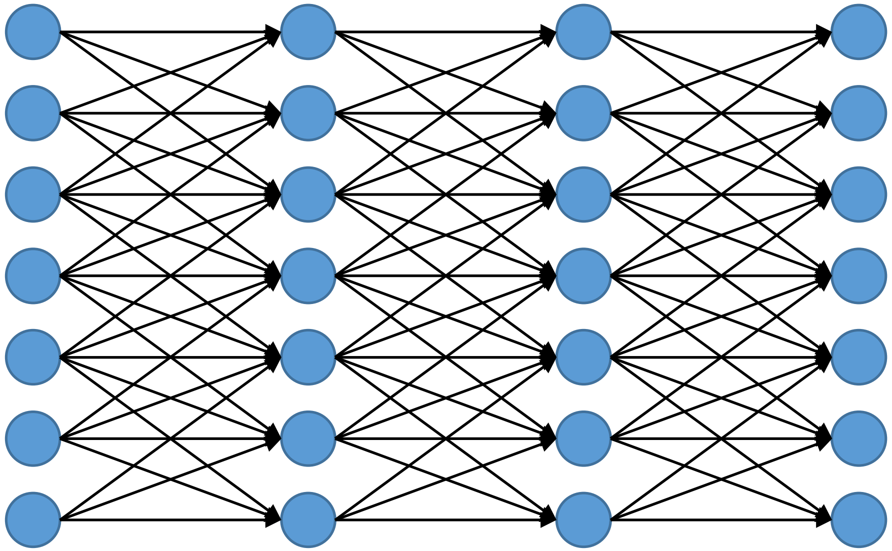
- Caffe?
- Installation
- Getting started: Image classification
  - Network design → Data preparation → Training → Testing
- More examples
  - Python interface usages
  - Advanced building blocks
  - Layer implementation

Caffe?

- Lots of DL libraries…
    - TensorFlow, Caffe, Theano-based (Keras, Lasagne, Pylearn, …), Torch, CuDNN, mxnet, neon, Intel MKL DL, …
- Caffe
    - **[△]** Rapid prototyping in an algorithmic level (a new layer, new loss function, …)
    - **[X]** Multi-device support Actually you can use multi-GPU for training on Caffe, but in a very restricted way.
    - **[X]** Easy installation & Portability
    - **[△]** Documentation Do not trust descriptions in the Caffe homepage. Double-check their actual implementations
    - **[○]** Rapid tuning by trial & error
    - **[○]** Fast
    - **[○]** Best support for computer vision research with large-scale datasets
    - **[○]** Portability of networks & pre-trained models: Highly reproducible

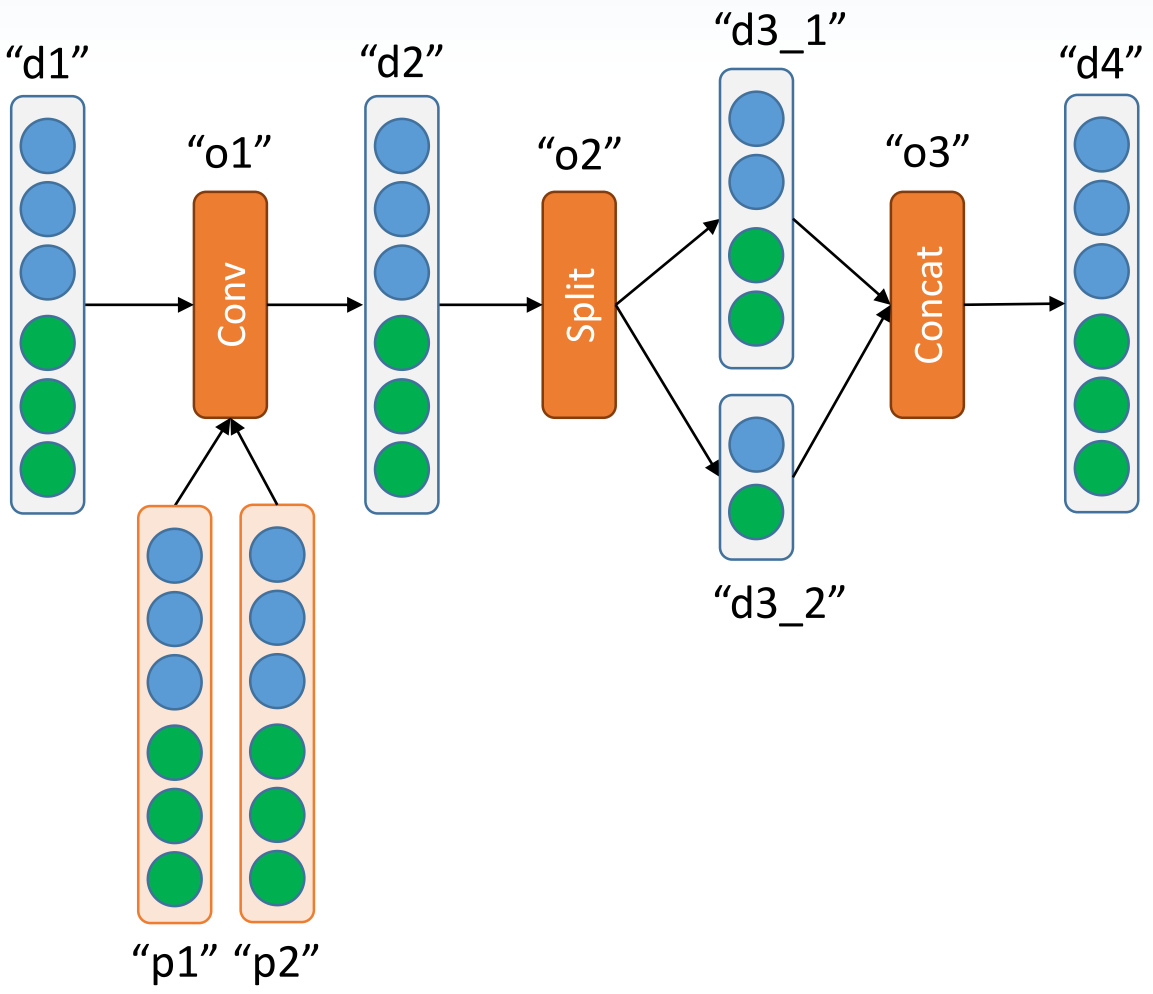
Classical Deep Network Representation

- Layer = data
- Connection = parameter
- Do **not** think like this



Deep Network Representation in DL Libraries

- Layer = operator
  - May or **may not** mathematical function
  - May or **may not** have trainable parameters
- Blob = data, parameter
  - Values & Gradients
- Connection = data flow



Prototxt

A “convolution” operator named as “conv1/conv”
takes 1 input data named as “data”
produces 1 output data named as “conv”
has 2 trainable parameters named as “conv1/conv_weight” and “conv1/conv_bias”
also has some const arguments as in “convolution_param”

- Spec document for a network
- **layer { … }:** Layer
    - **name:** Layer’s name
    - **type:** Operation
    - **bottom:** Input data name
        - Repeatable (multi-input)
    - **top:** Output data name
        - Repeatable (multi-output)
    - **param { … }:** Trainable parameter
        - Repeatable (e.g., weight, bias, …)
    - **xxx_param { … }:** Options for operation ‘xxx’

When bottom = top, output values are directly overwritten to the input blob (called as “in-place operation”)
Use this only if you clearly know that the operator is in-place safe (e.g., convolution and pooling are not safe)

```
layer {
  name: "conv1/conv"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param { name: "conv1/conv_weight"  lr_mult: 0.1 }
  param { name: "conv1/conv_bias"  lr_mult: 0.1 }
  convolution_param {
    num_output: 64  kernel_size: 3  stride: 2  pad: 1
    weight_filler { type: "xavier" }
    bias_filler { type: "constant" value: 0.1 }
  }
}
layer {
  name: "conv1/relu"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pool1/pool"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3  stride: 2
  }
}
```

Prototxt

- Major difference between Caffe and other libs
- { network structure, data } ==x   x== { code, platform }
  - Easy to read & maintain
  - Easy to fix & retry
  - Reproducible
  - Portable
  - Forward/backward compatible

Installation

Installation Methods

- http://caffe.berkeleyvision.org/installation.html
- Windows
  - Docker: Not support GPU mode
  - Caffe for Windows
    - Require Visual Studio 2013 (+ NVIDIA driver & CUDA-7.5 for GPU mode)
    - Hard to follow-up latest updates
  - Linux subsystem (Windows 10 Redstone only): Not support GPU mode
- Ubuntu
  - Docker: Recommended method
  - Native installation: Also recommended, but takes too much time to practice
  - Amazon web services (AWS)
    - Also recommended if you have money (~$1/hour for GPU machine)
    - Not support GPU mode on 12-month free-trial instances

Ubuntu + Docker: Overview

- Pre-requisites
  - Ubuntu 14.04
  - NVIDIA driver & CUDA-7.5 (for GPU mode)
- Steps
  - Install Docker
  - Install NVIDIA Docker (for GPU mode)
  - Build Caffe image & Create virtual machine

Ubuntu + Docker: Install Docker

- Open terminal & Follow installation instructions in https://docs.docker.com/engine/installation/linux/ubuntulinux/
- Test whether Docker is properly installed

```
$ sudo docker run hello-world
```

```
Hello from Docker!
This message shows that your installation appears to be
working correctly.
...
```

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

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

$ sudo bash -c \
    'echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main" > \
     /etc/apt/sources.list.d/docker.list'

$ sudo apt-get update
$ sudo apt-get purge lxc-docker
$ apt-cache policy docker-engine
$ sudo apt-get update
$ sudo apt-get install linux-image-extra-$(uname -r)
$ sudo apt-get install apparmor
$ sudo apt-get update
$ sudo apt-get install docker-engine
$ sudo service docker start
```

Ubuntu + Docker: Install NVIDIA Docker

- Follow installation instructions for "Ubuntu distributions" in https://github.com/NVIDIA/nvidia-docker/wiki

```
$ 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
```

- Test whether nvidia-docker is properly installed

```
$ sudo nvidia-docker run --rm nvidia/cuda:7.5-devel nvcc --version
$ sudo nvidia-docker run --rm nvidia/cuda:7.5-devel nvidia-smi
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2015 NVIDIA Corporation
Built on Tue_Aug_11_14:27:32_CDT_2015
Cuda compilation tools, release 7.5, V7.5.17
```

- Numbers can be different

```
+------------------------------------------------------+
¦ NVIDIA-SMI 352.93     Driver Version: 352.93         ¦
|-------------------------------+----------------------+----------------------+
¦ GPU  Name        Persistence-M¦ Bus-Id        Disp.A ¦ Volatile Uncorr. ECC ¦
¦ Fan  Temp  Perf  Pwr:Usage/Cap¦         Memory-Usage ¦ GPU-Util  Compute M. ¦
...
```

- Numbers can be different, but you should see a table like this
- Otherwise, it means that NVIDIA driver is not properly installed

Ubuntu + Docker: Install NVIDIA Docker

- Follow installation instructions for "Ubuntu distributions" in https://github.com/NVIDIA/nvidia-docker/wiki

```
$ 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
```

- Test whether nvidia-docker is properly installed

```
$ sudo nvidia-docker run --rm nvidia/cuda:7.5-devel nvcc --version
$ sudo nvidia-docker run --rm nvidia/cuda:7.5-devel nvidia-smi
```

```
docker and nvidia-docker require root permission (sudo)
nvidia-docker run initiates a container (VM) from a pre-built image, and then executes a command on it
  nvidia/cuda:7.5-devel: Pre-built image. If it doesn't exist in the host machine,
                         search & download it from Docker Hub (https://hub.docker.com)
  nvidia-smi: command to be executed
--name <container name> option assigns the name to the container, which is omitted in the above example
--rm option removes the container after the command is finished
```

Ubuntu + Docker: Build Caffe Image (GPU)

- Open gedit & Write Dockerfile

```
$ gedit Dockerfile
```

```
FROM kaixhin/cuda-caffe
RUN apt-get update
RUN apt-get install -y x11-apps python-tk tk-dev vim
RUN pip uninstall -y matplotlib
RUN pip install matplotlib
ENV DISPLAY :0
RUN echo "export PATH=:/root/caffe/build/tools:\${PATH}" >> ~/.bashrc
RUN echo "export LD_LIBRARY_PATH=:/root/caffe/build/lib:\${LD_LIBRARY_PATH}" >> ~/.bashrc
RUN cp /root/caffe/Makefile.config.example /root/caffe/Makefile.config
RUN echo "USE_CUDNN := 1" >> /root/caffe/Makefile.config
RUN cd /root/caffe
RUN git pull origin master
RUN make clean
RUN make -j"$(nproc)" all && make pycaffe
```

- Build image named caffe-img

```
$ sudo nvidia-docker build -t caffe-img .
```

- Permission settings for GUI

```
$ echo "xhost +SI:localuser:root" >> ~/.profile
$ xhost +SI:localuser:root
```

Ubuntu + Docker: Build Caffe Image (CPU)

- Open gedit & Write Dockerfile

```
$ gedit Dockerfile
```

```
FROM kaixhin/caffe
RUN apt-get update
RUN apt-get install -y x11-apps python-tk tk-dev vim
RUN pip uninstall -y matplotlib
RUN pip install matplotlib
ENV DISPLAY :0
RUN echo "export PATH=:/root/caffe/build/tools:\${PATH}" >> ~/.bashrc
RUN echo "export LD_LIBRARY_PATH=:/root/caffe/build/lib:\${LD_LIBRARY_PATH}" >> ~/.bashrc
RUN cp /root/caffe/Makefile.config.example /root/caffe/Makefile.config
RUN echo "CPU_ONLY := 1" >> /root/caffe/Makefile.config
RUN cd /root/caffe
RUN git pull origin master
RUN make clean
RUN make -j"$(nproc)" all && make pycaffe
```

- Build image named caffe-img

```
$ sudo docker build -t caffe-img .
```

- Permission settings for GUI

```
$ echo "xhost +SI:localuser:root" >> ~/.profile
$ xhost +SI:localuser:root
```

GPU vs. CPU: Only two lines are different!

```
FROM kaixhin/cuda-caffe
...
RUN echo "USE_CUDNN := 1" >> ...
```

```
FROM kaixhin/caffe
...
RUN echo "CPU_ONLY := 1" >> ...
```

Ubuntu + Docker: Caffe VM

CPU-only mode:
Replace nvidia-docker → docker

- Create virtual machine named caffe

```
$ sudo nvidia-docker run -tid \
                         -v /tmp/.X11-unix:/tmp/.X11-unix \
                         -v /tmp/.docker.xauth:/tmp/.docker.xauth \
                         -e XAUTHORITY=/tmp/.docker.xauth \
                         --name caffe caffe-img
```

t: Enable terminal mode
i: Get standard input (interactive mode)
d: Run on background

Options for GUI

- Open terminal on the VM

```
$ sudo nvidia-docker exec -ti caffe bash
```

```
$ sudo nvidia-docker exec -ti caffe bash    // Start terminal
    root@...:~/caffe#        // Now you are in VM as root
    ... do some work ...
    root@...:~/caffe# exit   // End terminal
$ sudo nvidia-docker stop caffe          // Power-off VM
$ sudo nvidia-docker start caffe         // Power on VM
$ sudo nvidia-docker commit caffe caffe_160819  // Backup VM
$ sudo nvidia-docker rm caffe            // Remove VM
$ sudo nvidia-docker rmi caffe-img       // Remove image
```

Ubuntu + Docker: Caffe VM

- Test Caffe on the VM

```
~/caffe# ./data/cifar10/get_cifar10.sh
~/caffe# ./examples/cifar10/create_cifar10.sh
~/caffe# ./examples/cifar10/train_quick.sh
```

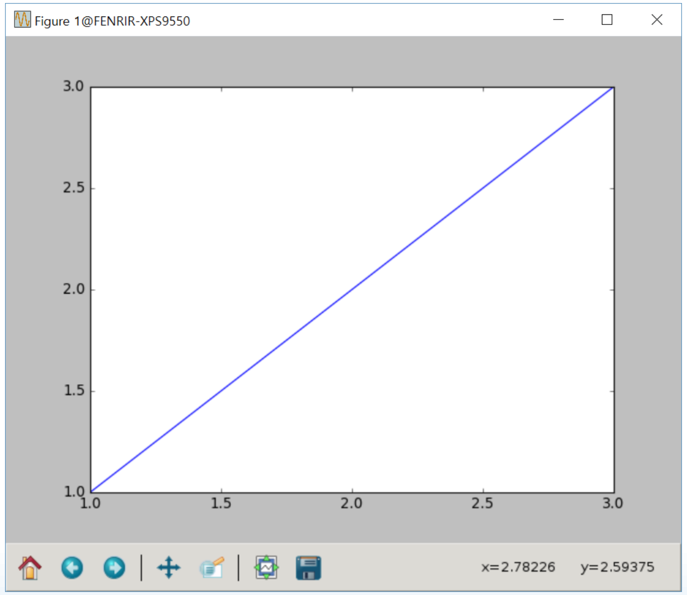
CPU-only mode: Edit solver_mode: **GPU** → solver_mode: **CPU**
in ./examples/cifar10/cifar10_quick_solver.prototxt

```
libdc1394 error: Failed to initialize libdc1394
...
I08… solver.cpp:317] Iteration 5000, loss = 0.584047
I08… solver.cpp:337] Iteration 5000, Testing net (#0)
I08… solver.cpp:404]     Test net output #0: accuracy = 0.7587
I08… solver.cpp:404]     Test net output #1: loss = 0.723281 (* 1 = 0.723281 loss)
I08… solver.cpp:322] Optimization Done.
I08… caffe.cpp:254] Optimization Done.
```

- Numbers can be different
- In CPU mode, every 100-iteration takes around 1 minute or more
- In GPU mode, every 100-iteration should be done in a few seconds, and the whole training process should be finished in several minutes. Otherwise, it means that CUDA doesn't work, mostly because NVIDIA driver is not properly installed

- Test GUI on the VM

```
~/caffe# python
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3], [1,2,3])
>>> plt.show()
```



- UI can be different, but you should see a figure like this

Windows: Docker

- https://github.com/BVLC/caffe/tree/master/docker
- Support CPU mode only
- Install Docker
  - Windows 10 Pro
    https://docs.docker.com/docker-for-windows/
  - Other Windows
    https://www.docker.com/products/docker-toolbox

- Virtualization
  - Check taskmgr → 성능 → “가상화: 사용”
  - If not, modify your BIOS settings
  - e.g., Advanced → CPU → Virtualization
- File sharing with host machine
  - Right click Docker icon in Taskbar
    → Click “Settings…”
  - Click “Shared drives”
    → Select drive you want to share
    → Click “Apply”
    → Enter your Windows account info

Windows: Docker

```
C:\...> docker pull kaixhin/caffe    // Download Caffe dockerfile
C:\...> docker run -dit --name caffe kaixhin/caffe    // Create VM

C:\...> docker exec -ti caffe bash    // Start Linux terminal

    root@...:~/caffe#                          // Now you are in Linux VM

    ... do some work ...

    root@...:~/caffe# exit                     // End Linux terminal

C:\...> docker stop caffe             // Power-off VM
C:\...> docker start caffe            // Power on VM

C:\...> docker commit caffe caffe_160819      // Backup VM

C:\...> docker rm caffe               // Remove VM
C:\...> docker rmi kaixhin/caffe      // Remove Caffe dockerfile
```

```
// Install other packages required in this lecture
...# apt-get update && apt-get upgrade
...# apt-get install python-opencv python-pip vim
...# pip install lmdb
...# git clone https://github.com/kyehyeon/caffe-materials

// Do only if arrow keys do not work in your vim
...# echo "set term=cons25" >> ~/.vimrc

// Get the latest Caffe
root@...:~/caffe# git pull origin master
root@...:~/caffe# make clean
root@...:~/caffe# cp Makefile.config.example Makefile.config
root@...:~/caffe# vim Makefile.config
    // Uncomment "CPU_ONLY := 1" and "WITH_PYTHON_LAYER := 1"
root@...:~/caffe# make -j"$(nproc)" all && make pycaffe
```

Other Options: Pre-requisites

- Visual Studio 2013 (for Windows)
- NVIDIA driver & CUDA 7.5 (for GPU mode)
  - https://developer.nvidia.com/cuda-downloads
  - Install NVIDIA driver: Yes (even if a newer version is already installed)
  - Install CUDA toolkit: Yes
  - CUDA toolkit path: Default
  - Make symbolic link: Yes
- CuDNN library (for GPU mode)
  - https://developer.nvidia.com/rdp/cudnn-download (membership required)
  - Download **v5** (not RC!) for CUDA **7.5**
  - Unzip & Remember CuDNN root path
    - Include path: <CuDNN root>/include
    - Library path: <CuDNN root>/lib64

Windows: Caffe for Windows

- https://github.com/BVLC/caffe/tree/windows
  - Click Clone or download
  - caffe-windows\windows
    CommonSettings.props.example → CommonSettings.props

```
<CpuOnlyBuild>true</CpuOnlyBuild>
<UseCuDNN>false</UseCuDNN>
…
<PythonSupport>true</PythonSupport>
…
<PropertyGroup Condition="'$(PythonSupport)'=='true'">
    <PythonDir>your Miniconda path</PythonDir>
```

- Install Miniconda (Python + Libraries)
  - http://conda.pydata.org/miniconda.html
  - Download Python 2.7 & 64-bit & Install
    - Just for me, Add to path, Default Python
  - cmd →

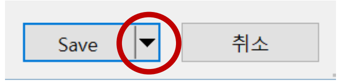
```
conda install --yes numpy scipy matplotlib scikit-image pip lmdb
```

Windows: Caffe for Windows

- Open caffe-windows\windows\Caffe.sln
- Build → Build Solution (F7)
  - caffe-windows\Build\x64\{Debug, Release}*.exe
  - Trouble shooting

```
warning C4819: The file contains a
character that cannot be represented in the
current code page (949). Save the file in
Unicode format to prevent data loss
```

Save As →



→ Save with Encoding
→ Unicode (UTF-8 with signature)

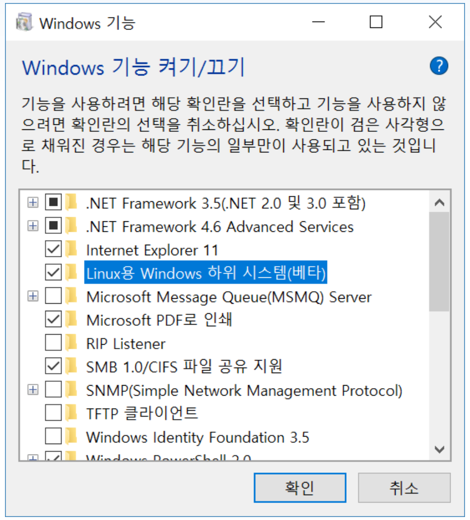
rng.hpp
alt_sstream_impl.hpp
opaque_pointer_converter.hpp
dealloc.hpp
return_opaque_pointer.hpp

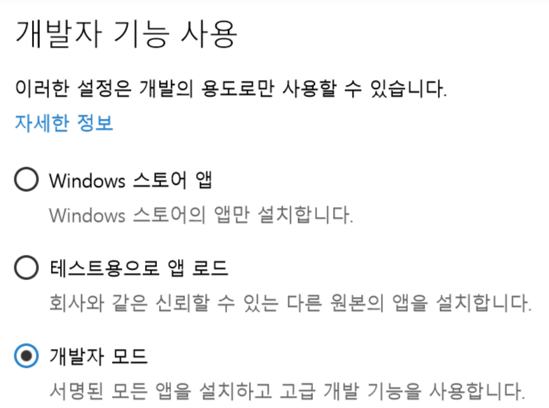
- Copy Python package
  - caffe-windows\Build\x64\{Debug, Release}\pycaffe\caffe
    → <Miniconda root>\Lib\site-packages

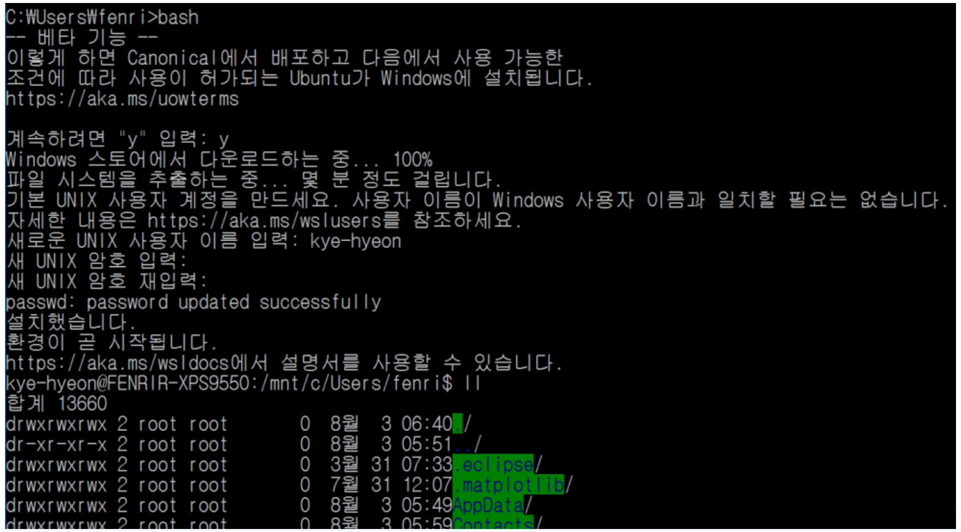
    C:\Users\<your account name>\Miniconda2

Windows: Linux Subsystem

- Install Linux subsystem
  - 제어판 → "프로그램 및 기능" → "Windows 기능 켜기/끄기" → "Linux용 Windows 하위 시스템(베타)"
  - 설정 → "업데이트 및 복구" → "개발자용" → "개발자 모드"
  - cmd → bash → "y <Enter>"
- Follow instructions in Ubuntu: Native Installation







Windows: Linux Subsystem

- Locale & GUI settings

```
$ sudo update-locale LANG=en_US.UTF8
$ vim ~/.bashrc
```

```
export DISPLAY=:0.0
```

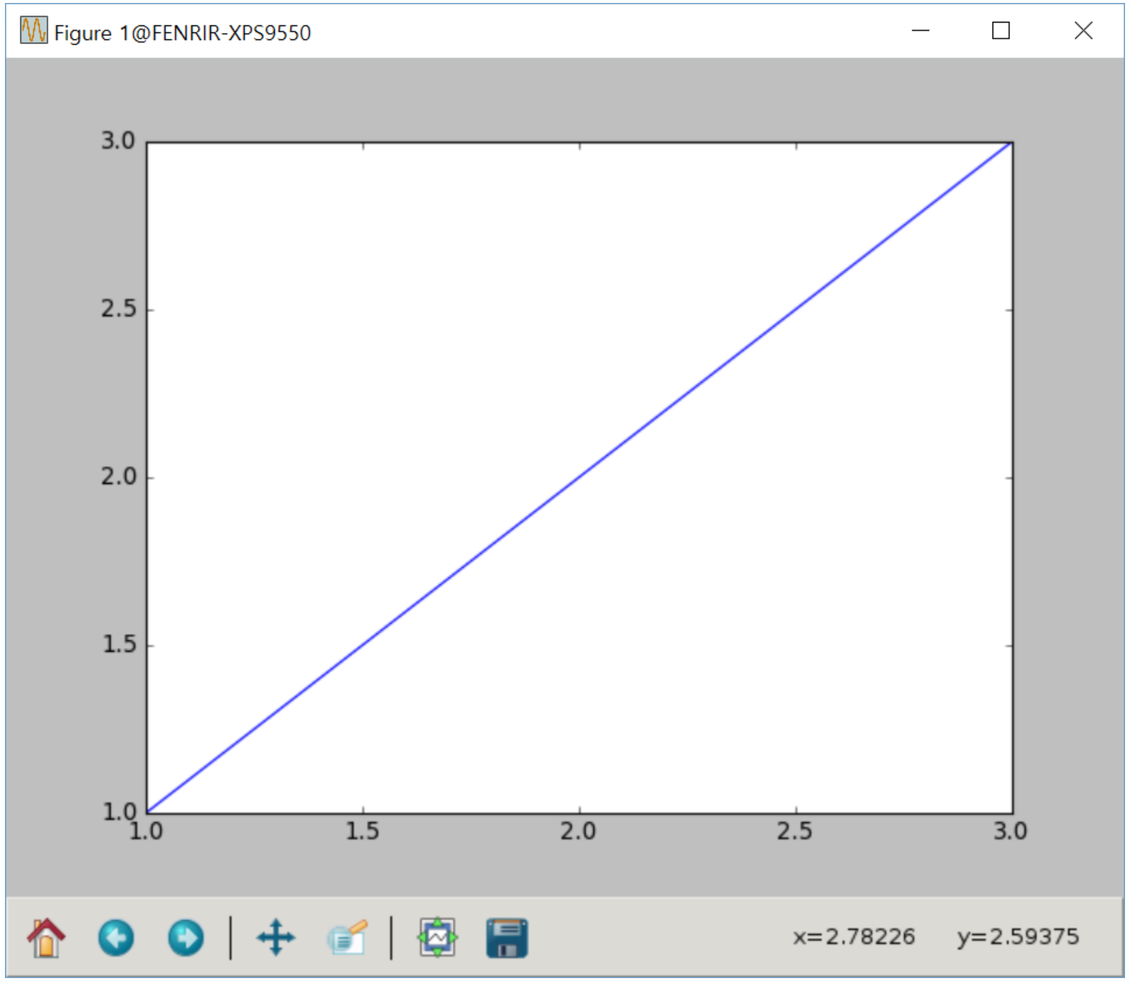
```
$ sudo vim /etc/dbus-1/session.conf
```

```
<listen>unix:tmpdir=/tmp</listen>
➔ <listen>tcp:host=localhost,port=0</listen>
```

```
$ sudo apt-get install python-tk tk-dev
$ pip uninstall -y matplotlib
$ pip install --user matplotlib
```

```
$ python
```

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3], [1,2,3])
>>> plt.show()
```



Ubuntu: Native Installation

- http://caffe.berkeleyvision.org/install_apt.html
- Install pre-built packages

```
$ sudo apt-get update
$ sudo apt-get install <following packages>

  build-essential git
  libprotobuf-dev protobuf-compiler libhdf5-serial-dev
  libgflags-dev libgoogle-glog-dev libsnappy-dev
  libatlas-base-dev libopencv-dev
  liblmdb-dev libleveldb-dev
  python-dev python-pip python-opencv gfortran

$ sudo apt-get install --no-install-recommends libboost-all-dev
$ pip install --user easydict
$ pip install --user lmdb
```

Ubuntu: Native Installation

- Download Caffe & Install Python

```
$ git clone https://github.com/BVLC/caffe.git
$ cd caffe/python/
$ for req in $(cat requirements.txt); do pip install --user $req; done
$ pip install --user dask
$ for req in $(cat requirements.txt); do pip install --user $req; done   // To check successful installation
$ cd ..
$ git clone https://github.com/kyehyeon/caffe-materials
```

- Build Caffe

```
$ cp Makefile.config.example Makefile.config
$ vim Makefile.config
    // Uncomment "CPU_ONLY := 1" if you have no GPU
    // Uncomment "WITH_PYTHON_LAYER := 1"
$ make -j8 && make pycaffe
```

- Set paths

```
$ vim ~/.bashrc
```

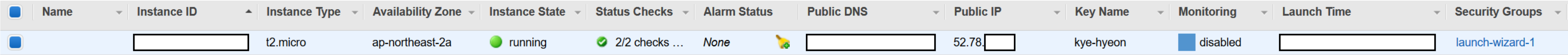
```
export CAFFE_ROOT=<your Caffe root directory>
export PATH=:${CAFFE_ROOT}/build/tools:${PATH}
export LD_LIBRARY_PATH=:${CAFFE_ROOT}/build/lib:${LD_LIBRARY_PATH}
export PYTHONPATH=:${CAFFE_ROOT}/python:${PYTHONPATH}
```

Ubuntu: AWS

- https://aws.amazon.com
- Join AWS
  - Click "Create a Free Account"
  - Sign in with your Amazon account
  - Select "Personal Account" & Follow the steps
  - Click "Sign In to the Console"

- Create AWS instance
  - Change region to "Asia Pacific (Tokyo)"

  - Click "EC2" → "Launch Instance"
  - Select "Ubuntu Server 14.04 LTS (HVM), SSD Volume Type" → "t2.micro" → "Review & Launch"
  - Create a new key pair → Any name → Save <your name>.pem file

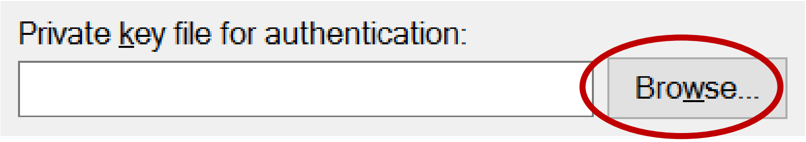


	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP	Key Name	Monitoring	Launch Time	Security Groups
			t2.micro	ap-northeast-2a	running	2/2 checks ...	None		52.78.	kye-hyeon	disabled		launch-wizard-1

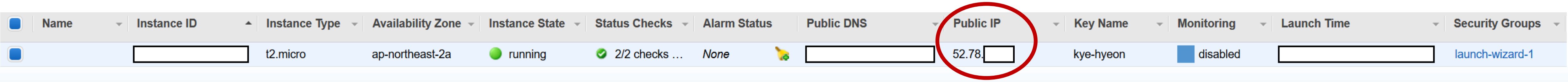
Ubuntu: AWS

- Download PuTTY and PuTTYgen
  - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
- Run PuTTYgen
  - If SmartScreen blocks it: "More info" → "Run anyway"
  - "Load" → Choose your .pem file
    → Click "Save private key"
    → Save your .ppk file

- Run PuTTY
  - "SSH" → "Auth"
    →



    → Open your .ppk file
  - "Session"
    → Input your instance's Public IP to "Host Name" field
  - Click "Open"
    → Input ubuntu <Enter>



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP	Key Name	Monitoring	Launch Time	Security Groups
		t2.micro	ap-northeast-2a	running	2/2 checks ...	None		52.78.	kye-hyeon	disabled		launch-wizard-1

Ubuntu: AWS

- Create virtual memory space

```
$ sudo /bin/dd if=/dev/zero of=/var/swap.1 bs=1M count=1024
$ sudo /sbin/mkswap /var/swap.1
$ sudo /sbin/swapon /var/swap.1
```

- Follow Ubuntu: Native Installation
- Remove virtual memory space

```
$ sudo swapoff /var/swap.1
$ sudo rm /var/swap.1
```

- GUI settings
  - PuTTY: "Connection" → "SSH" → Check "Enable X11 forwarding"

```
$ sudo apt-get install python-tk tk-dev
$ pip uninstall -y matplotlib
$ pip install --user matplotlib
```

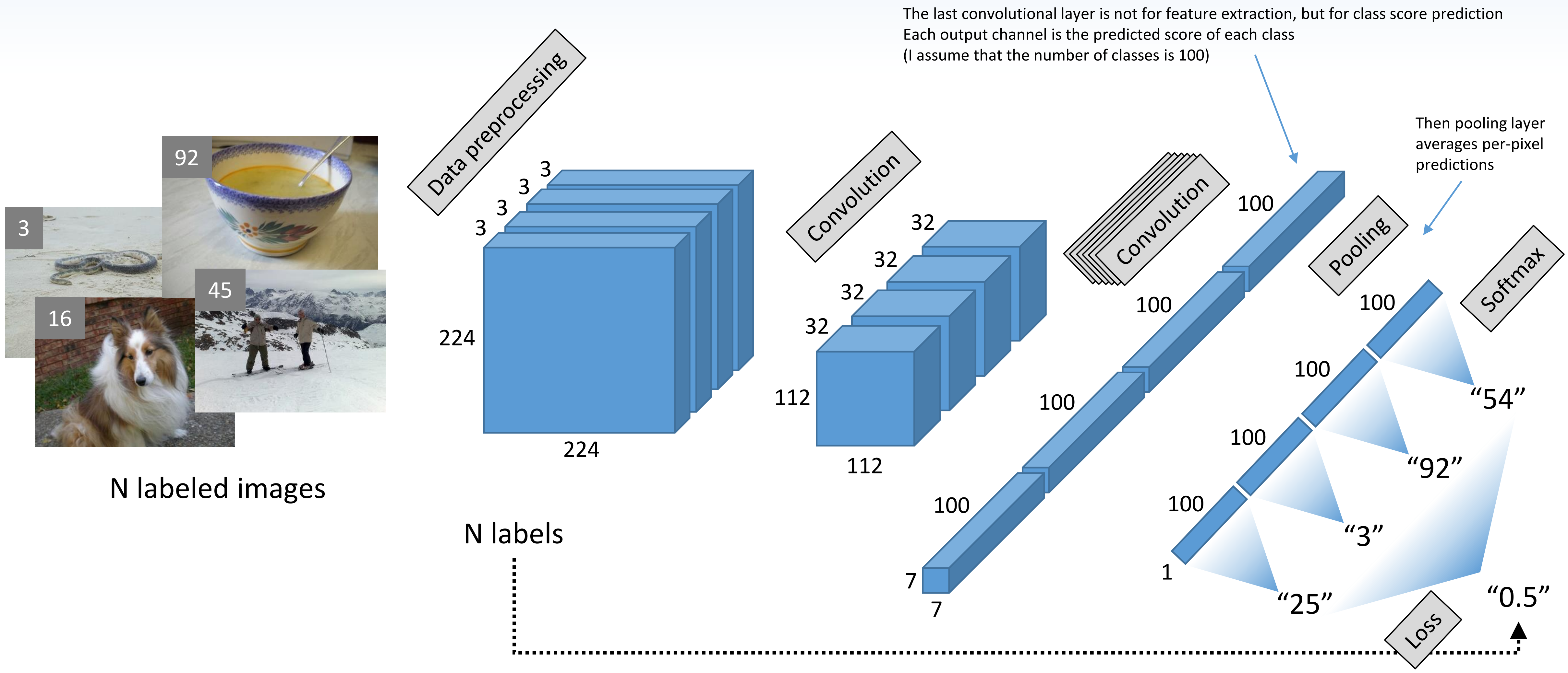
```
// Check whether GUI works properly
$ python
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3], [1,2,3])
>>> plt.show()
```

Getting Started: Image Classification

Common Workflow

- Network design
- Data preparation
- Training
- Testing
- Tuning

Network Design



Network Design

- Data layer
  - **source:** Input data path
  - **batch_size:** # of images per iteration
  - **crop_size:** Random (TRAIN) or Center (TEST)
  - **mirror:** Flip LR (50% random)
  - **mean_value:** Use 3 times (BGR-order)
  - **top:** first = data, second (optional) = label

Output data is also BGR-ordered
It is due to OpenCV
https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/

- Pooling layer
  - **pool:** MAX, AVE, STOCHASTIC
  - **global_pooling:** 1 x 1 output
  - **kernel_size, pad, stride:** Normal pooling
- Loss layer
  - **type:** Loss function
  - SoftmaxWithLoss = Softmax → MultinomialLogisticLoss
  - **bottom:** first = prediction, second = label

```
layer {
  name: "data/data"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "data/train_lmdb"  backend: LMDB  batch_size: 128
  }
  transform_param {
    crop_size: 224  mirror: true
    mean_value: 104  mean_value: 117  mean_value: 123
  }
  include { phase: TRAIN }
}
```

```
layer {
  name: "pool6/pool"
  type: "Pooling"
  bottom: "conv5"
  top: "pool6"
  pooling_param { pool: AVE  global_pooling: true }
}
```

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pool6"
  bottom: "label"
  top: "loss"
}
```

Network Design

- Convolution layer
  - **num_output:** # of output channels
  - **bias_term:** Whether use or not (default: true)
- ReLU layer
  - **negative_slope:** Nonzero slope for negative inputs (default: 0)
  - **bottom = top:** In-place operation
- Example
  - 5 “Conv → ReLU” layers
  - num_output: 32 → 64 → 128 → 256 → 100
  - kernel_size: 3, pad: 1, stride: 2
    ➔ Height, Width: 112 → 56 → 28 → 14 → 7

32→64→128→256-dim latent features
→ 100 class scores

This is just an example
5 layers are not enough at all to get reasonable accuracy!

```
layer {
  name: "conv1/conv"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 32
    kernel_size: 3  pad: 1  stride: 2
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv1/relu"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
```

Network Design: Additional Information

- Convolution layer
  - **kernel_h, kernel_w, stride_h, stride_w, pad_h, pad_w:** 2D rectangular convolution
  - **group:** Convolution with channel-wise slicing (default: 1)
    - "num_output: 256, group: 4, input channel: 128" = 4 convolutions of "num_output: 256 / 4, input channel: 128 / 4"
  - **axis:** Channel axis index (default: 1)
    - "axis: 2, input: 64 x 32 x 28 x 28" = 1D convolution of "batch_size: 64 * 32, channel: 28, spatial size: 28"
    - "axis: 1, input: 64 x 32 x 28 x 28 x 28" = 3D convolution of "batch_size: 64, channel: 32, spatial size: 28 x 28 x 28"
- **Output shape difference: Convolution vs. Pooling**
  - Conv output size = floor((input_size + 2*pad - kernel_size) / stride) + 1
  - Pool output size = ceil((input_size + 2*pad - kernel_size) / stride) + 1 ← When kernel_size=3, set pad=0, not 1
  - Not corrected yet due to backward compatibility (https://github.com/BVLC/caffe/issues/1318)
- Loss layer
  - **ignore_label:** Label index to be ignored (e.g., for hard example mining)
- Every layer
  - **loss_weight:** If > 0, corresponding top data is considered as a loss term (default: 0)

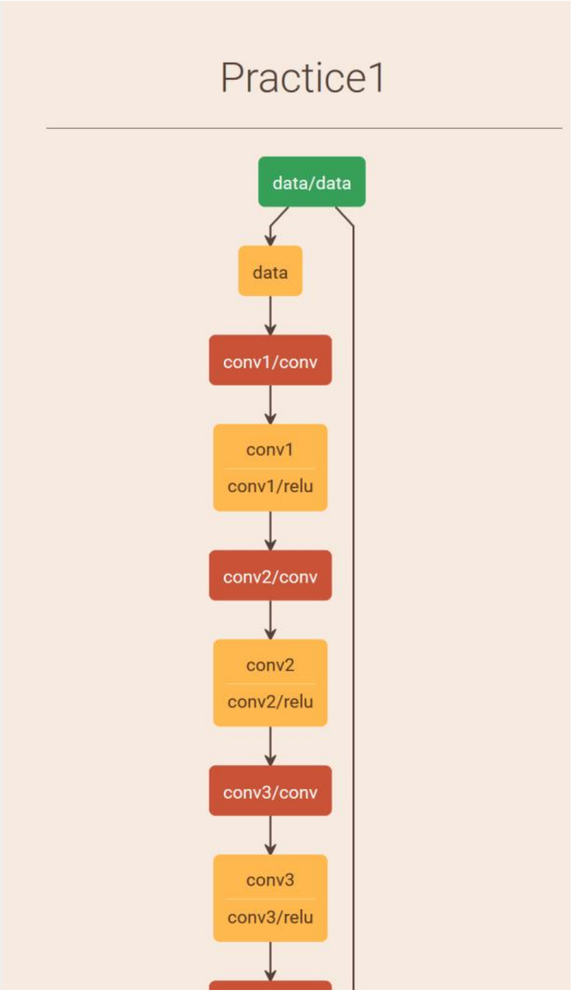
If you implement your own Python loss layer, "loss_weight: 1" (or any non-zero value) should be explicitly specified in the prototxt for back-propagation

Network Design: Visualization

- Netscope
  - http://ethereon.github.io/netscope/#/editor
  - Paste your prototxt and “Shift + Enter”
  - Not displayed correctly in IE and Edge

- Examples: AlaxNet, GoogLeNet, VGG
  - http://ethereon.github.io/netscope/#/preset/alexnet
  - http://ethereon.github.io/netscope/#/preset/googlenet
  - http://ethereon.github.io/netscope/#/preset/vgg-16

```
name: "Practice1"
layer {
  name: "data/data"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "data/train_lmdb"  backend: LMDB  batch_size: 128
  }
  transform_param {
    crop_size: 224  mirror: true
    mean_value: 104  mean_value: 117  mean_value: 123
  }
  include { phase: TRAIN }
}
layer {
  name: "conv1/conv"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 32
    kernel_size: 3  pad: 1  stride: 2
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv1/relu"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "conv2/conv"
  type: "Convolution"
  bottom: "conv1"
  top: "conv2"
  convolution_param {
    num_output: 64
    kernel_size: 3  pad: 1  stride: 2
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv2/relu"
  type: "ReLU"
```



Data Preparation

- We have datasets in the form of…
  - Numerical vectors: .txt, .csv, .xls, .mat, .pkl, …
  - Images: <root>/<class>/<filename>
    - train/person/000001.jpg, test/n101024/000123.jpg, …
  - Documents: plain text, XML, …
  - …

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- To use datasets in Caffe
  - Option 1: Convert datasets to LMDB or LevelDB → Use Data layer
  - Option 2: Implement your Python data layer

Data Preparation: LMDB?

- Key-value DB (dictionary or hash)
  - Key = byte array, Value = byte array (i.e., arbitrary data type & size for each record)
- Support multi-threaded environments
  - Read performance scales linearly with # of readers
  - One writer at a time
  - Transaction does not block other transactions (e.g., writer doesn't block readers)
- Ultra-fast
  - No transaction log, appending mode, …
  - "Unmatched" in-memory performance & Outstanding on-disk performance

Data Preparation: Images → LMDB

- Using convert_imageset
  -
```
$ <Caffe root>/build/tools/convert_imageset -encoded=true -encode_type="jpg"
                                            -resize_height=256 -resize_width=256
                                            -shuffle=true
                                            data/images/
                                            data/train.txt
                                            data/train_lmdb
```
  - .txt file: List of <file path> <class>

```
train/ILSVRC2012_val_00000001.JPEG 65
train/ILSVRC2012_val_00000002.JPEG 970
train/ILSVRC2012_val_00000003.JPEG 230
train/ILSVRC2012_val_00000004.JPEG 809
train/ILSVRC2012_val_00000005.JPEG 516
...
```

```
layer {
  name: "data/data"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "data/train_lmdb"  backend: LMDB  batch_size: 128
  }
  transform_param {
    crop_size: 224  mirror: true
    mean_value: 104  mean_value: 117  mean_value: 123
  }
  include { phase: TRAIN }
}
```

Data Preparation: Images → LMDB

- ImageNet DB

```
~/caffe# mkdir data/imagenet
```

In the Caffe VM

```
$ cd <your ImageNet download location>
$ sudo docker cp val.txt caffe:/root/caffe/data/imagenet
$ sudo docker cp val caffe:/root/caffe/data/imagenet
```

In the host machine

```
~/caffe# ./build/tools/convert_imageset
             -encoded=true -encode_type="jpg"
             -resize_height=256 -resize_width=256
             data/imagenet/ data/imagenet/val.txt
             data/imagenet/train_lmdb
```

In the Caffe VM

```
~/caffe# cp -r data/imagenet/train_lmdb data/imagenet/test_lmdb
```

- Tiny DB

```
~/caffe# ln -s /root/caffe/caffe-materials/tiny_lmdb data/tiny_lmdb
```

Training

```
~/caffe# ./build/tools/caffe train -solver caffe-materials/practice1/solver.pt

GPU mode:
~/caffe# ./build/tools/caffe train -gpu 0 \
                  -solver caffe-materials/practice1/solver-gpu.pt
```

- Stochastic optimization for $t = 1, 2, \ldots, T$
  - Sampling mini-batch data: $X_t = [x_1, x_2, \ldots, x_M]$
  - Forward-pass
    - Output $[f(x_1), f(x_2), \ldots, f(x_M)]$ and Loss $\sum_{i=1}^{M} l\big(y_i, f(x_i)\big)$
  - Backward-pass: Gradient $\frac{\partial E}{\partial w}$
  - Update: $w \leftarrow w - \eta_t h_t\left(\frac{\partial E}{\partial w}\right)$
- Prototxt
  - Number of iterations T, Learning rate η_t, Solver $h_t(\ldots)$
  - **iter_size:** Update with multiple mini-batches
- Run
  - `$ ./build/tools/caffe train -solver solver.pt -gpu 0`

```
net: "practice1.pt"

max_iter: 300000
iter_size: 2

lr_policy: "step"
base_lr: 0.003
gamma: 0.3165
stepsize: 60000

type: "SGD"
momentum: 0.9
weight_decay: 0.0002
solver_mode: CPU

display: 20
snapshot: 20000
snapshot_prefix: "practice1_train"
```

Testing

- Network prototxt
  - include { phase: TEST }
  - Data layer
    - **source:** Test DB path
  - Accuracy layer
    - **top_k:** Top-k accuracy
- Solver prototxt
  - For testing during training
    - **test_interval:** Interval of testing
    - **test_iter:** # of iterations

```
layer {
  name: "data/data"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "data/test_lmdb"  backend: LMDB  batch_size: 128
  }
  transform_param {
    crop_size: 224  mirror: true
    mean_value: 104  mean_value: 117  mean_value: 123
  }
  include { phase: TEST }
}
```

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "pool6"
  bottom: "label"
  top: "accuracy"
  include { phase: TEST }
}
layer {
  name: "accuracy_top5"
  type: "Accuracy"
  bottom: "pool6"
  bottom: "label"
  top: "accuracy_top5"
  accuracy_param { top_k: 5 }
  include { phase: TEST }
}
```

```
net: "practice1.pt"

max_iter: 300000
iter_size: 2

lr_policy: "step"
base_lr: 0.003
gamma: 0.3165
stepsize: 60000

type: "SGD"
momentum: 0.9
weight_decay: 0.0002
solver_mode: CPU

display: 20
snapshot: 20000
snapshot_prefix: "practice1_train"

test_interval: 20000
test_iter: 500
```

For more details about the model used in practice2, see https://arxiv.org/abs/1602.07360
SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size

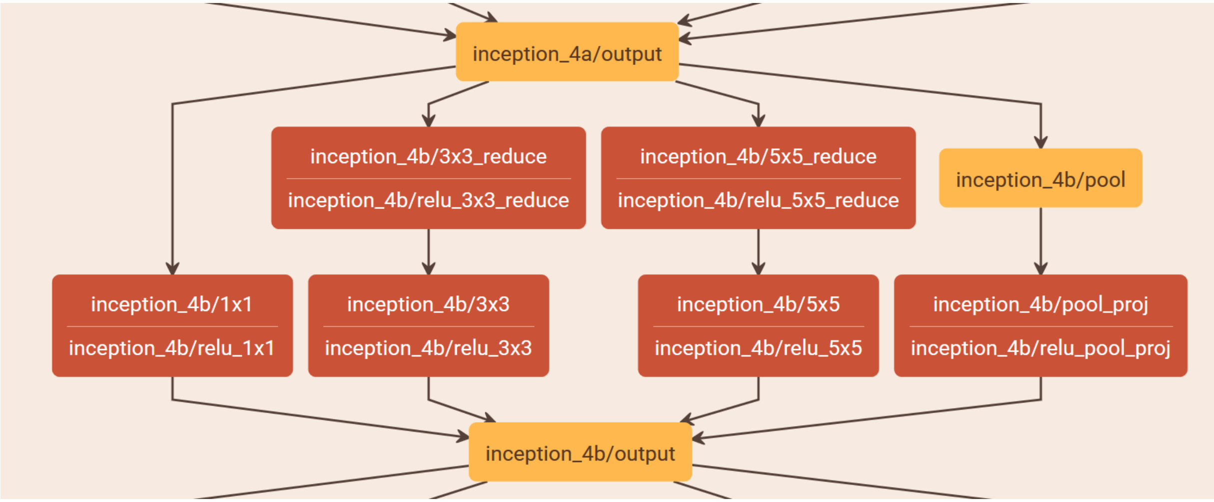
Testing

```
~/caffe# ./build/tools/caffe test \
   -model caffe-materials/practice2/train_val.prototxt \
   -weights caffe-materials/practice2/squeezenet_v1.1.caffemodel \
   -gpu 0
```

- Testing
  - ```
    $ ./build/tools/caffe test -model net.pt
                               -weights net_train_iter_300000.caffemodel
                               -gpu 0
                               -iterations 100
    ```
- Profiling
  - ```
    $ ./build/tools/caffe time -model net.pt
                               -gpu 0
                               -iterations 100
                               -phase TEST
    ```

Tuning

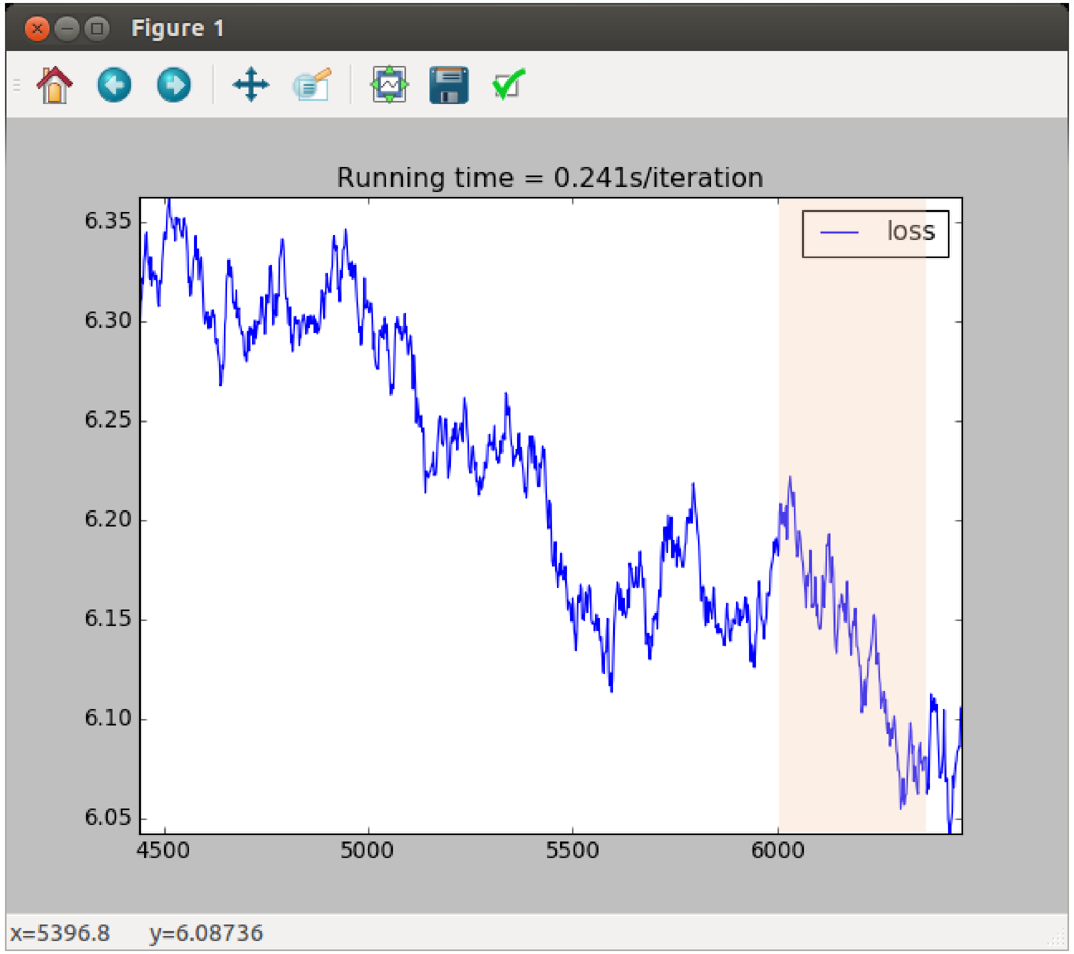
- Parameter tuning
  - Solver prototxt: base_lr
  - Network prototxt: num_output
- Network re-design
  - Advanced building blocks
  - Batch normalization, C.ReLU, Squeezing, Inception, Residual, Multi-scale feature, …



More Examples

Training: Visualization?

```
I0731 17:16:06.876653  4320 solver.cpp:228] Iteration 6000, loss = 5.93425
I0731 17:16:11.765796  4320 solver.cpp:228] Iteration 6020, loss = 6.05768
I0731 17:16:16.568244  4320 solver.cpp:228] Iteration 6040, loss = 6.39608
I0731 17:16:21.438531  4320 solver.cpp:228] Iteration 6060, loss = 6.13255
I0731 17:16:26.282059  4320 solver.cpp:228] Iteration 6080, loss = 6.26793
I0731 17:16:31.161669  4320 solver.cpp:228] Iteration 6100, loss = 5.92059
I0731 17:16:36.031781  4320 solver.cpp:228] Iteration 6120, loss = 6.50278
I0731 17:16:40.830622  4320 solver.cpp:228] Iteration 6140, loss = 5.9315
I0731 17:16:45.686990  4320 solver.cpp:228] Iteration 6160, loss = 6.05751
I0731 17:16:50.473193  4320 solver.cpp:228] Iteration 6180, loss = 5.98048
I0731 17:16:55.304819  4320 solver.cpp:228] Iteration 6200, loss = 5.85572
I0731 17:17:00.104171  4320 solver.cpp:228] Iteration 6220, loss = 6.05796
I0731 17:17:04.934598  4320 solver.cpp:228] Iteration 6240, loss = 5.90128
I0731 17:17:09.769328  4320 solver.cpp:228] Iteration 6260, loss = 6.09975
I0731 17:17:14.618911  4320 solver.cpp:228] Iteration 6280, loss = 6.33493
I0731 17:17:19.412217  4320 solver.cpp:228] Iteration 6300, loss = 5.84982
I0731 17:17:24.235080  4320 solver.cpp:228] Iteration 6320, loss = 6.39197
I0731 17:17:29.127218  4320 solver.cpp:228] Iteration 6340, loss = 5.85397
I0731 17:17:33.995726  4320 solver.cpp:228] Iteration 6360, loss = 6.19559
I0731 17:17:38.840620  4320 solver.cpp:228] Iteration 6380, loss = 6.21344
I0731 17:17:43.661043  4320 solver.cpp:228] Iteration 6400, loss = 6.22203
```



Training with Python

- Python interface
  - Load, Save, Train, Test
  - Access trainable parameters & intermediate data
- For more information
  - python/caffe/_caffe.cpp: BOOST_PYTHON_MODULE(_caffe) { … }

```
~/caffe# python caffe-materials/practice3/py_train_1.py
```

```
import caffe

# Use GPU 0
caffe.set_mode_gpu()
caffe.set_device(0)

# Initialize solver
solver = caffe.SGDSolver('solver.pt')

# Restore snapshot
solver.restore('net_train_iter_300000.solverstate')
# or trained parameters
solver.net.copy_from('net_train_iter_300000.caffemodel')

# Train 10 iterations
solver.step(10)

# Access 'pool6' data (NumPy array)
pool6_data = solver.net.blobs['pool6'].data
print pool6_data.shape   # (32, 1000, 1, 1)

# Save snapshot
solver.snapshot()
```

Training with Python: Visualization

- Initialization

```
import caffe
import matplotlib.pyplot
import time as timelib

solver = caffe.SGDSolver('solver.pt')

fig, axes = matplotlib.pyplot.subplots()
fig.show()

loss_list = []
max_iter = 10000
iter0 = solver.iter
```

```
~/caffe# python caffe-materials/practice3/py_train_2.py
```

- Training & Drawing

```
while solver.iter < max_iter:
  solver.step(1)

  loss = solver.net.blobs['loss'].data.flatten()
  loss_list.append(loss)

  # Update plot for every 500 iterations
  if solver.iter % 500 == 0:
    axes.clear()
    axes.plot(range(iter0, iter0+len(loss_list)), loss_list)
    axes.grid(True)
    fig.canvas.draw()
    matplotlib.pyplot.pause(0.01)

solver.snapshot()
fig.savefig('fig_iter_%d.png' % solver.iter)
```

Training with Python: Advanced Scheduling

- Caffe LR policy
  - η_t = base_lr * $\eta(t)$
  - Manipulating base_lr in Python causes no conflict with any Caffe LR policy
- Advanced scheduling
  - If loss plateaus, reduce base_lr
  - If base_lr is too small, restore it
  - Problem: No Python interface to access base_lr

```cpp
template <typename Dtype>
Dtype SGDSolver<Dtype>::GetLearningRate() {
  Dtype rate;
  const string& lr_policy = this->param_.lr_policy();

  if (lr_policy == "fixed") {
    rate = this->param_.base_lr();
  }
  else if (lr_policy == "step") {
    this->current_step_ = this->iter_ / this->param_.stepsize();
    rate = this->param_.base_lr() *
        pow(this->param_.gamma(), this->current_step_);
  }
  else if (lr_policy == "exp") {
    rate = this->param_.base_lr() *
        pow(this->param_.gamma(), this->iter_);
  }
  ...
  return rate;
}
```

src/caffe/solvers/sgd_solver.cpp

Training with Python: Advanced Scheduling

- Add functions to get & set base_lr
  - include/caffe/solver.hpp

```
template <typename Dtype>
class Solver {
 public:
   ...
   Dtype GetBaseLearningRate() { return param_.base_lr(); }
   void SetBaseLearningRate(const Dtype base_lr);
   ...
```

  - src/caffe/solver.cpp

```
namespace caffe {
...
template <typename Dtype>
void Solver<Dtype>::SetBaseLearningRate(const Dtype base_lr) {
  param_.set_base_lr(base_lr);
}
...
```

- Add Python interface
  - python/caffe/_caffe.cpp

```
...
.def("step", &Solver<Dtype>::Step)
.def("get_base_lr", &Solver<Dtype>::GetBaseLearningRate)
.def("set_base_lr", &Solver<Dtype>::SetBaseLearningRate)
.def("restore", &Solver<Dtype>::Restore)
.def("snapshot", &Solver<Dtype>::Snapshot);
...
```

- Make

```
$ make -j$(nproc) && make pycaffe
```

Training with Python: Advanced Scheduling

- Moving average of loss

```
while solver.iter < max_iter:
  solver.step(1)
  loss = solver.net.blobs['loss'].data.flatten()

  if len(loss_list) == 0:
    mean_loss = loss
  else:
    mean_loss = 0.999 * mean_loss + 0.001 * loss
  loss_list.append(mean_loss)
```

- LR policy: Plateau + Multi-round

```
window = [0, 1000]
base_lr0 = solver.get_base_lr()

while solver.iter < max_iter:
  solver.step(1)
  ...
  loss_list.append(mean_loss)

  if len(loss_list) - window[0] > window[1] and \
      mean_loss > 0.99 * loss_list[-window[1]]:
    solver.set_base_lr(solver.get_base_lr() * 0.5)
    window[0] = len(loss_list)
    window[1] *= 2

  if solver.get_base_lr() < 0.1 * base_lr0:
    solver.set_base_lr(base_lr0)
    window[1] = 1000
```

Training with Python: Advanced Scheduling


```
~/caffe# python caffe-materials/practice3/py_train_4.py
```

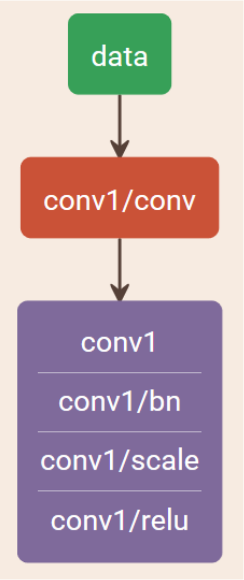
Advanced Building Blocks

- Batch Normalization
  - Faster and more stable convergence
  - S. Ioffe & C. Szegedy (2015), ICML-2015, https://arxiv.org/abs/1502.03167
- Concatenated ReLU
  - 2x faster computation in early stages of CNNs
  - W. Shang, K. Sohn, D. Almeida & H. Lee (2016), ICML-2016, https://arxiv.org/abs/1603.05201
- Residual Connections
  - Very deep networks converge much better
  - K. He, X. Zhang, S. Ren & J. Sun (2016), CVPR-2016, https://arxiv.org/abs/1512.03385

Batch Normalization

- Convolution $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
    - bias_term: false
- BatchNorm $\boldsymbol{y} \leftarrow \frac{\boldsymbol{y} - mean(\boldsymbol{y})}{std(\boldsymbol{y})}$
    - 3 internal parameters
    - Do **not** specify use_global_stats
- Scale $\boldsymbol{y} \leftarrow \alpha \boldsymbol{y} + \beta$
    - bias_term: true
- ReLU $\boldsymbol{y} \leftarrow g(\boldsymbol{y})$

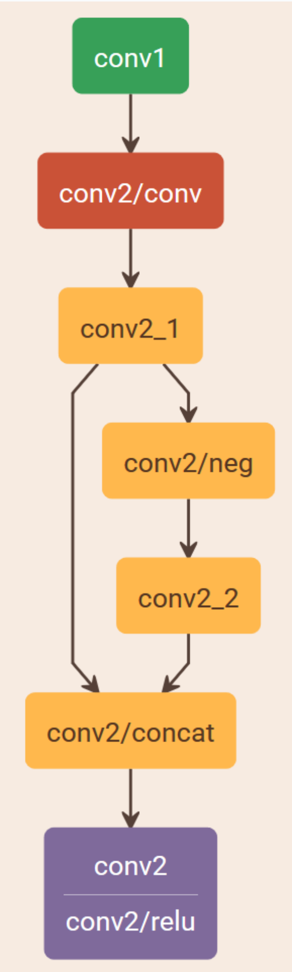
```
layer {
  name: "conv1/conv"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  #param { name: "conv1/conv/weight"  lr_mult: 1  decay_mult: 1 }
  convolution_param {
    num_output: 64  kernel_size: 3  stride: 1  pad: 1
    bias_term: false
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv1/bn"
  type: "BatchNorm"
  bottom: "conv1"
  top: "conv1"
  #batch_norm_param { use_global_stats: false }
  #param { name: "conv1/bn/mean"  lr_mult: 0  decay_mult: 0 }
  #param { name: "conv1/bn/var"  lr_mult: 0  decay_mult: 0 }
  #param { name: "conv1/bn/count"  lr_mult: 0  decay_mult: 0 }
}
layer {
  name: "conv1/scale"
  type: "Scale"
  bottom: "conv1"
  top: "conv1"
  param { name: "conv1/scale/weight"  lr_mult: 1  decay_mult: 1 }
  param { name: "conv1/scale/bias"  lr_mult: 2  decay_mult: 0 }
  scale_param { bias_term: true }
}
layer {
  name: "conv1/relu"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
```



Concatenated ReLU

- Convolution $\boldsymbol{y}_1 = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
  - num_output: / 2
- Power (negation) $\boldsymbol{y}_2 = -\boldsymbol{y}_1$
  - scale: -1
  - shift: 0, power: 1 (default)
- Concat $\boldsymbol{y} = [\boldsymbol{y}_1; \boldsymbol{y}_2]$
  - axis: 1 (= channel-wise, default)
- ReLU $\boldsymbol{y} \leftarrow g(\boldsymbol{y})$

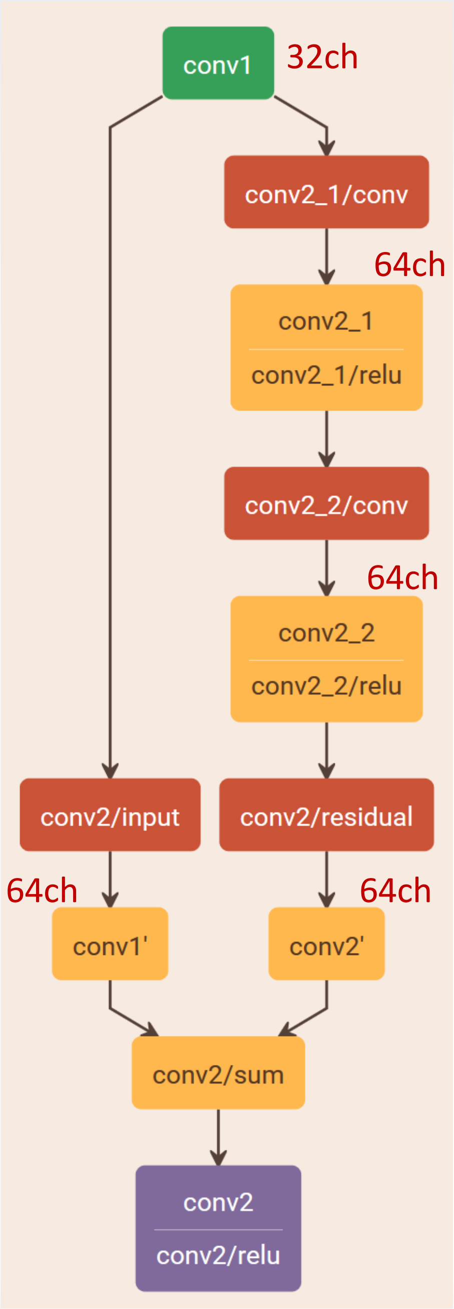
```
layer {
  name: "conv2/conv"
  type: "Convolution"
  bottom: "conv1"
  top: "conv2_1"
  convolution_param {
    num_output: 32  kernel_size: 3  pad: 1  stride: 2
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv2/neg"
  type: "Power"
  bottom: "conv2_1"
  top: "conv2_2"
  power_param {
    scale: -1.0
    #shift: 0.0  power: 1.0
  }
}
layer {
  name: "conv2/concat"
  type: "Concat"
  bottom: "conv2_1"
  bottom: "conv2_2"
  top: "conv2"
  #concat_param { axis: 1 }
}
layer {
  name: "conv2/relu"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
```



Residual Connections

- Convolution $\boldsymbol{y}' = \boldsymbol{W}_r f(\boldsymbol{x}) + \boldsymbol{b}_r$
  - $f(\boldsymbol{x})$: Any multi-layer func.
  - Do **not** use ReLU for final output
- Convolution $\boldsymbol{x}' = \boldsymbol{W}_p \boldsymbol{x} + \boldsymbol{b}_p$
  - Use **only** for shape matching
- Eltwise $\boldsymbol{y} = \boldsymbol{x}' + \boldsymbol{y}'$
- ReLU $\boldsymbol{y} \leftarrow g(\boldsymbol{y})$

```
layer {
  name: "conv2/residual"
  type: "Convolution"
  bottom: "conv2_2"
  top: "conv2'"
  convolution_param {
    num_output: 64  kernel_size: 3  pad: 1  stride: 1
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv2/input"
  type: "Convolution"
  bottom: "conv1"
  top: "conv1'"
  convolution_param {
    num_output: 64  kernel_size: 1  pad: 0  stride: 1
    weight_filler { type: "xavier" }
  }
}
layer {
  name: "conv2/sum"
  type: "Eltwise"
  bottom: "conv1'"
  bottom: "conv2'"
  top: "conv2"
  eltwise_param { operation: SUM  coeff: 1  coeff: 1 }
}
layer {
  name: "conv2/relu"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
```



Network Construction with Python

- Writing prototxt for repeating patterns is very annoying and easy to make mistakes
- Caffe provides an interface for programmable network prototxt generation
- See src/caffe/proto/caffe.proto for more information
  - NetParameter: Network prototxt
  - LayerParameter: layer { ... }
  - ParamSpec: param { ... }
  - XXXParameter: XXX_param { ... }

It contains all field names used in Caffe prototxt, with their data types and characteristics (optional, repeated, ...)

Network Construction with Python

```
>>> import caffe
>>> layer = caffe.proto.caffe_pb2.LayerParameter()
>>> dir(layer)
['name', 'type', 'bottom', 'top', 'param',
 'accuracy_param', ..., 'window_data_param',
 'include', 'loss_weight', 'propagate_down', ...]

>>> layer.name = 'some_layer'
>>> layer.type = 'SomeType'
>>> layer.bottom.append('input1')
>>> layer.bottom.append('input2')
>>> layer.top.append('output1')
>>> layer.top.append('output2')
>>> layer  # equals to 'print str(layer)'
name: "some_layer"
type: "SomeType"
bottom: "input1"
bottom: "input2"
top: "output1"
top: "output2"
```

In src/proto/caffe.proto, you can see that “bottom” and “top” are defined as “repeated”, meaning that they are “vectors”
For any field defined as “repeated”, you should assign values using “append” or “extend”, instead of assignment
Note that some “repeated” fields do not support “append” (e.g.,”layer” field in NetParameter)

```
>>> net = caffe.proto.caffe_pb2.NetParameter()
>>> dir(net)
[..., 'MergeFromString', 'ParseFromString',
 'SerializeToString', 'layer', ...]

>>> net.layer.extend([layer])
>>> net  # equals to 'print str(net)'
layer {
  name: "some_layer"
  type: "SomeType"
  bottom: "input1"
  bottom: "input2"
  top: "output1"
  top: "output2"
}
```

```
>>> net.layer.extend([layer])
>>> net
layer {
  name: "some_layer"
  ...
}
layer {
  name: "some_layer"
  ...
}
```

Network Construction with Python

- caffe_pb2?

```
message LayerParameter {
  optional string name = 1;
  optional string type = 2;
  repeated string bottom = 3;
  repeated string top = 4;

  optional Phase phase = 10;

  repeated float loss_weight = 5;

  repeated ParamSpec param = 6;
  ...
```

src/caffe/proto/caffe.proto

protoc

```
class LayerParameter : public ::google::protobuf::Message {
  ...
  // optional string name = 1;
  bool has_name() const;
  void clear_name();
  static const int kNameFieldNumber = 1;
  const ::std::string& name() const;
  void set_name(const ::std::string& value);
  ...

  // optional string type = 2;
  ...

  // repeated string bottom = 3;
  int bottom_size() const;
  void clear_bottom();
  static const int kBottomFieldNumber = 3;
  const ::std::string& bottom(int index) const;
  ::std::string* mutable_bottom(int index);
  void set_bottom(int index, const ::std::string& value);
  void set_bottom(int index, const char* value);
  void set_bottom(int index, const char* value, size_t size);
  ::std::string* add_bottom();
  void add_bottom(const ::std::string& value);
  ...

  // repeated string top = 4;
  ...
```

build/src/caffe/proto/caffe.pb.cc
python/caffe/proto/caffe.pb2.py

build/src/caffe/proto/caffe.pb.h

Network Construction with Python

- Network instance can be initialized from (ParseFromString) or merged with (MergeFromString) other network instance

```
>>> net.MergeFromString(net.SerializeToString())
>>> net
layer {
  name: "some_layer"
  ...
}
layer {
  name: "some_layer"
  ...
}
layer {
  name: "some_layer"
  ...
}
layer {
  name: "some_layer"
  ...
}
```

Network Construction with Python

```python
def base_layer(layer_name, type_name, \
               bottom_names, top_names):
  layer = caffe.proto.caffe_pb2.LayerParameter()
  layer.name = layer_name
  layer.type = type_name
  layer.bottom.extend(bottom_names)
  layer.top.extend(top_names)
  return layer
```

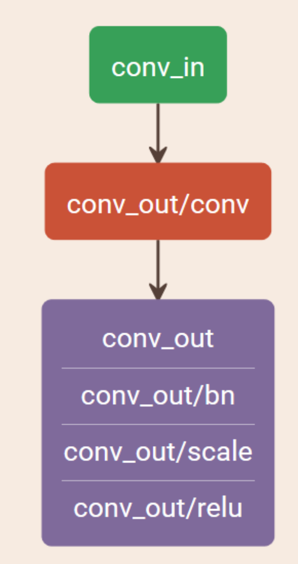
```python
def convolution_layer(bottom_name, top_name, \
      num_output, kernel_size, stride=1, pad=0, \
      group=1, bias_term=True):
  layer_name = top_name + '/conv'
  layer = base_layer(layer_name, 'Convolution', \
                     [bottom_name], [top_name])
  layer.convolution_param.num_output = num_output
  layer.convolution_param.kernel_size.append(kernel_size)
  layer.convolution_param.stride.append(stride)
  layer.convolution_param.pad.append(pad)
  layer.convolution_param.group = group
  layer.convolution_param.bias_term = bias_term
  layer.convolution_param.weight_filler.type = 'xavier'
  return layer
```

```
>>> convolution_layer('conv_in', 'conv_out', 32, 3)
name: "conv_out/conv"
type: "Convolution"
bottom: "conv_in"
top: "conv_out"
convolution_param {
  num_output: 32
  bias_term: true
  pad: 0
  kernel_size: 3
  group: 1
  stride: 1
  weight_filler {
    type: "xavier"
  }
}
```

Network Construction with Python

```python
def conv_module(bottom_name, top_name, \
                num_output, kernel_size, \
                stride=1, pad=0, group=1):
  module = caffe.proto.caffe_pb2.NetParameter()
  # Conv layer
  module.layer.extend( \
    [convolution_layer(bottom_name, top_name, \
          num_output, kernel_size, stride, pad, \
          group, bias_term=False)])
  # BatchNorm layer
  module.layer.extend( \
    [batch_norm_layer(top_name, top_name)])
  # Scale layer
  module.layer.extend( \
    [scale_layer(top_name, top_name, bias_term=True)])
  # ReLU layer
  module.layer.extend( \
    [relu_layer(top_name, top_name)])
  return module
```

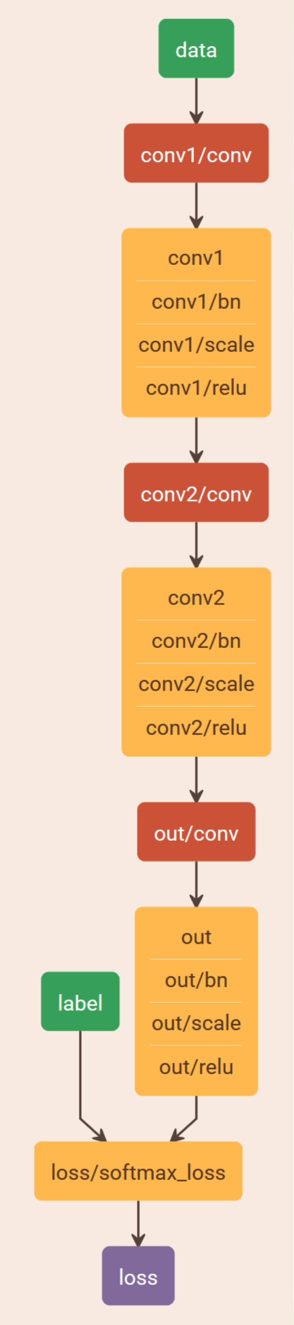
```
>>> conv_module('conv_in', 'conv_out', 32, 3)
layer {
  name: "conv_out/conv"
  type: "Convolution"
  bottom: "conv_in"
  top: "conv_out"
  convolution_param {
    num_output: 32
    ...
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "conv_out/bn"
  type: "BatchNorm"
  bottom: "conv_out"
  top: "conv_out"
}
layer {
  name: "conv_out/scale"
  type: "Scale"
  bottom: "conv_out"
  top: "conv_out"
  scale_param {
    bias_term: true
  }
}
layer {
  name: "conv_out/relu"
  type: "ReLU"
  bottom: "conv_out"
  top: "conv_out"
}
```



Network Construction with Python

```python
def conv_net(names, channels, kernels, strides):
  net = caffe.proto.caffe_pb2.NetParameter()
  # Data layer
  net.layer.extend( \
    [data_layer('data/train_lmdb', batch_size=64)])
  # Conv modules
  for i in range(len(channels)):
    pad = (kernels[i] - 1) / 2
    net.MergeFromString( \
      conv_module(names[i], names[i+1], \
                  channels[i], kernels[i], strides[i], pad) \
      .SerializeToString())
  # Loss layer
  net.layer.extend( \
    [softmax_loss_layer(['out', 'label'], 'loss')])
  return net
```

```python
conv_net( \
  ['data', 'conv1', 'conv2', 'out'], \
  [32, 64, 10], \
  [3, 3, 3], \
  [2, 2, 2])
```



LMDB Access

- convert_imageset is only useful for image classification tasks, not widely applicable to various databases
- Any type of data can be stored as LMDB
  - LMDB stores data as a byte array
  - Need to implement encoder (data → bytes) and decoder (bytes → data)

It's not recommended to use "float_data" even if your data are real-valued
Encode your real-valued data as a byte array and then assign it to "data"

Ignore this if your data are not images

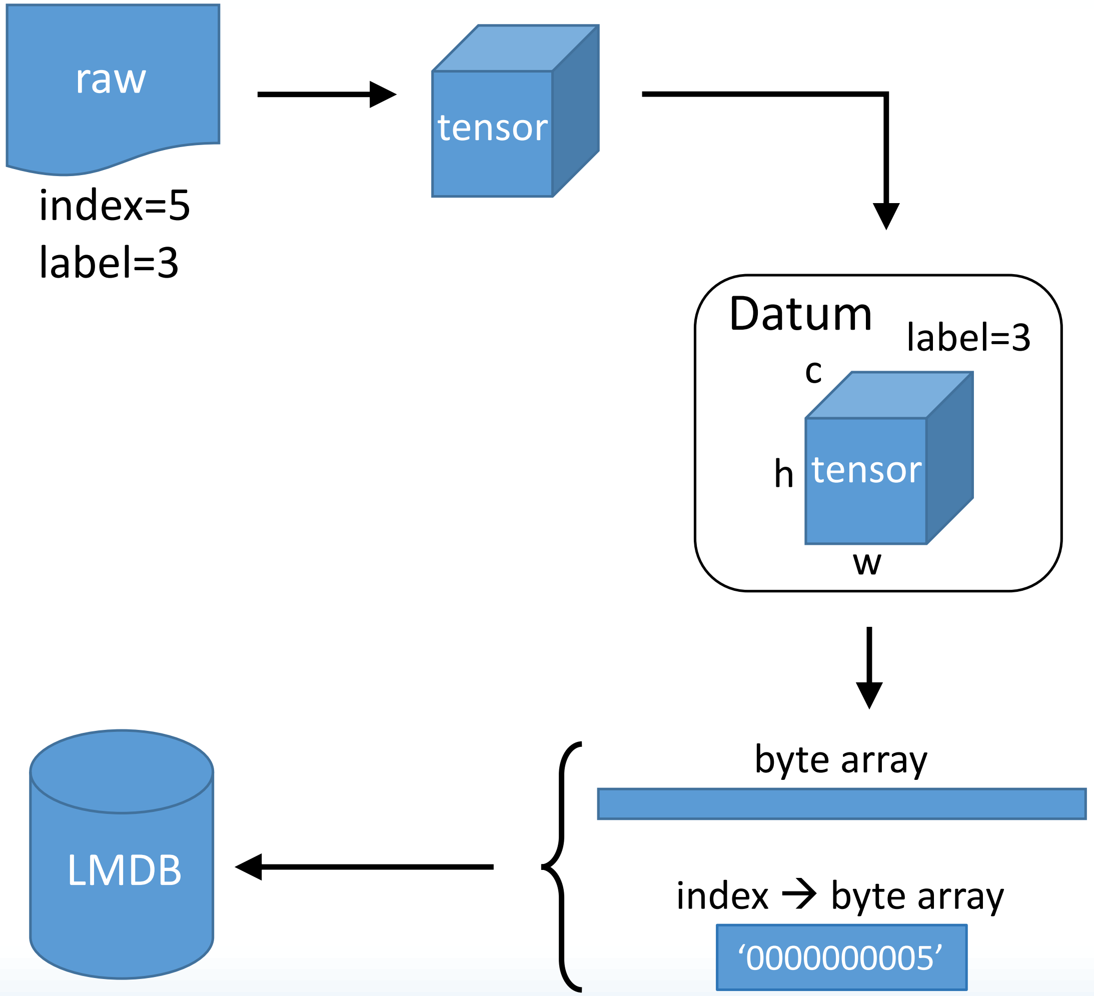
- Also related with src/caffe/proto/caffe.proto
- **Datum**: Data point instance
  - bytes data: Byte array of data
    - Can be an arbitrary type with a pre-processor
  - float float_data: Float array
    - Convenient to deal with real-valued vectors
  - int label: Label for classification tasks
  - int channels, height, width: Data shape
  - bool encoded: Whether data is an encoded image

LMDB Access: Python API Usage

```
>>> import lmdb
>>> reader = lmdb.open('data/imagenet/train_lmdb', readonly=True).begin()
>>> cursor = reader.cursor()
>>> cursor.next()
True
>>> cursor.key()
'00000000_train/n03476684/n03476684_14201.JPEG'
>>> cursor.value()
'"\x89\xdf\x02\xff\xd8\xff...\x01'
>>> cursor.next()
True
>>> cursor.key()
'00000001_train/n03642806/n03642806_6609.JPEG'
...
>>> cursor.close()
>>> cursor.key()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
lmdb.Error: Attempt to operate on closed/deleted/dropped object.
```

LMDB Access: Storing Data

- For each data instance, do:
  1. Convert raw data → tensor
  2. Determine tensor's shape
  3. (Optional) Compress tensor
  4. Fill in Datum instance's fields (data, channels, width, height, label)
  5. Encode Datum → byte array
  6. Store { instance's index, byte array } to LMDB



LMDB Access: Storing Data

- For each data instance, do:
  1. Convert raw data → tensor
  2. Determine tensor's shape
  3. (Optional) Compress tensor
  4. Fill in Datum instance's fields (data, channels, width, height)
  5. Encode Datum → byte array
  6. Store { index, byte array } to LMDB

LMDB open

1

2

3

4

5

6

Finalize & Close

```python
def write_lmdb(db_path, list_filename, height, width):
  map_size = 9999999999
  db = lmdb.open(db_path, map_size=map_size)
  writer = db.begin(write=True)
  datum = caffe.proto.caffe_pb2.Datum()
  for index, line in enumerate(open(list_filename, 'r')):
    img_filename, label = line.strip().split(' ')
    img = cv2.imread(img_filename, 1)
    img = cv2.resize(img, (height, width))
    _, img_jpg = cv2.imencode('.jpg', img)
    datum.channels = 3
    datum.height = height
    datum.width = width
    datum.label = int(label)
    datum.encoded = True
    datum.data = img_jpg.tostring()
    datum_byte = datum.SerializeToString()
    index_byte = '%010d' % index
    writer.put(index_byte, datum_byte, append=True)
  writer.commit()
  db.close()
```

LMDB Access: Loading Data

- For each data instance, do: (reverse order)
  1. Load { index, byte array } from LMDB
  2. Decode byte array → Datum
  3. Get required fields from Datum (data, label, ...)
  4. (Optional) Decompress tensor
  5. Determine tensor's shape
  6. Convert tensor → input data

```python
import lmdb, cv2, caffe
import numpy as np

def read_lmdb(db_path):
  db = lmdb.open(db_path, readonly=True)
  reader = db.begin()
  cursor = reader.cursor()
  datum = caffe.proto.caffe_pb2.Datum()
  for index_byte, datum_byte in cursor:
    datum.ParseFromString(datum_byte)
    np_array = np.fromstring(datum.data, dtype=np.uint8)
    label = datum.label
    img = cv2.imdecode(np_array, 1)
    data = np.rollaxis(img, 2, 0)
    yield (data, label)
  cursor.close()
  db.close()
```

LMDB Access: Practice

- Storing data

```
~/caffe# python caffe-materials/practice5/lmdb_access.py write \
    caffe-materials/practice5/imagenet_small.txt \
    256 256 \
    data/imagenet/small_lmdb
```

- Loading data

```
~/caffe# python caffe-materials/practice5/lmdb_access.py read \
    data/imagenet/small_lmdb
```

LMDB Access: Some Tips

- Handling non-integer labels:
  - When you have labeled data (x_n, y_n) but y_n is a real-valued vector, image, ...
  - e.g., face recognition tasks, multivariate label, ...
  - Make 2 LMDBs: $\{x_1, x_2, x_3, \ldots, x_N\}$ and $\{y_1, y_2, y_3, \ldots, y_N\}$
  - Construct network with 2 data layer: `data_x` $(= x_n)$ and `data_y` $(= y_n)$
  - Of course, never shuffle two LMDBs separately

Network Data Access

- We can do runtime access to network data in Python
  - Intermediate layer data

    ```
    net.blobs['conv1'].data
    ```

    Layer data has its own name (the name used for bottom, top)
  - Trainable parameters (weight, bias)

    ```
    net.params['conv1/conv'][0].data
    net.params['conv1/conv'][1].data
    ```

    Parameter can be accessed via layer's name
  - Their gradients

    ```
    net.blobs['conv1'].diff
    net.params['conv1/conv'][0].diff
    net.params['conv1/conv'][1].diff
    ```
  - See python/caffe/_caffe.cpp for whole APIs

```
def compression(true_net, comp_net, cfgs):
  for layer_name in true_net._layer_names:
    if cfgs.has_key(layer_name):
      rank = cfgs[layer_name]
      W_true = true_net.params[layer_name][0].data
      b_true = true_net.params[layer_name][1].data
      W1, b1, W2, b2 = svd(W_true, b_true, rank)
      comp_net.params[layer_name+'_1'][0].data[...] = W1
      comp_net.params[layer_name+'_1'][1].data[...] = b1
      comp_net.params[layer_name+'_2'][0].data[...] = W2
      comp_net.params[layer_name+'_2'][1].data[...] = b2

true_net = caffe.Net('true.pt', 'true.cm', caffe.TEST)
comp_net = caffe.Net('comp.pt', 'true.cm', caffe.TEST)
cfgs = { 'conv1': 8, 'conv2': 16 }
compression(true_net, comp_net, cfgs)
comp_net.save('comp.cm')
```

Just an example application... Not stand-alone runnable

Python Layer Implementation

```python
import caffe
import numpy as np
import yaml

class NewPythonLayer(caffe.Layer):
  def setup(self, bottom, top):
    # Read & parse parameters
    # You can make any optional auxiliary data
    layer_params = yaml.load(self.param_str)
    self._num_output = layer_params['num_output']
    # Compute & set parameter data shape
    self.blobs.add_blob(...)
    self.blobs.add_blob(...)
    # Initialize parameter data
    self.blobs[0].data[...] = ...
    self.blobs[1].data[...] = ...
```

"param_str_" in older versions of Caffe

new_python_layer.py

```python
def reshape(self, bottom, top):
    # Read input data shape
    bottom0_shape = bottom[0].data.shape
    bottom1_shape = bottom[1].data.shape
    # Compute & set output data shape
    top[0].reshape(...)
    top[1].reshape(...)
```

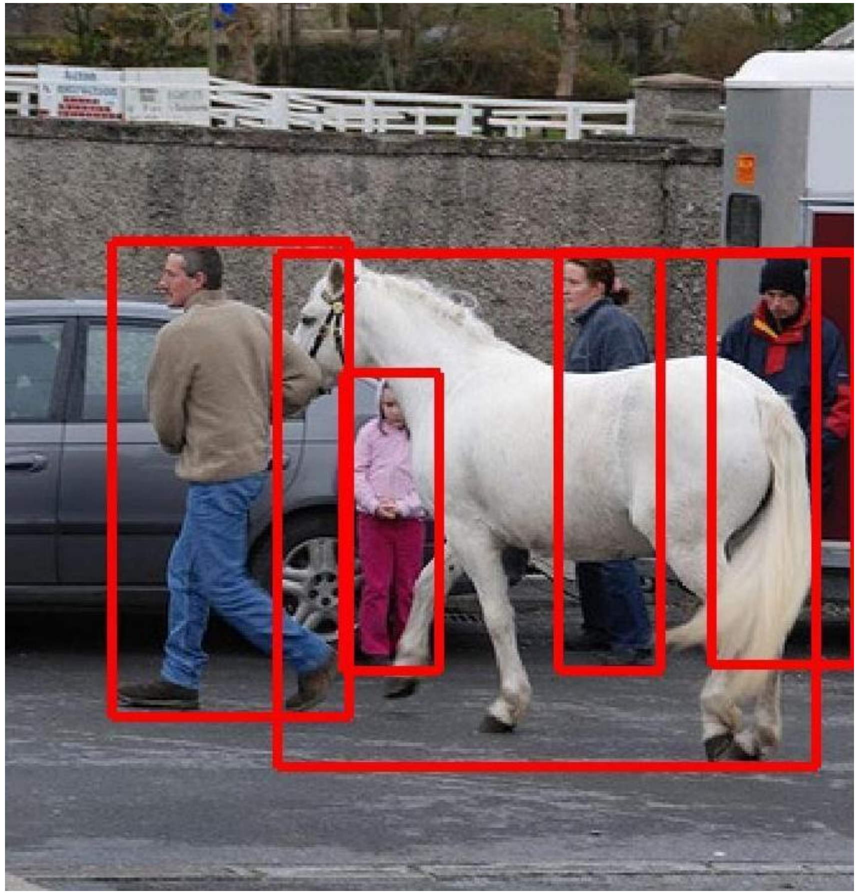
```python
def forward(self, bottom, top):
    # Read input data
    bottom0_data = bottom[0].data
    bottom1_data = bottom[1].data
    # Read parameter data
    weight = self.blobs[0].data
    bias = self.blobs[1].data
    # Compute & store output data
    top[0].data[...] = ...
    top[1].data[...] = ...
```

```python
def backward(self, top, propagate_down, bottom):
    # Read output data gradient
    top0_diff = top[0].diff
    top1_diff = top[1].diff
    # Compute & store parameter data gradient
    self.blobs[0].diff[...] = ...
    self.blobs[1].diff[...] = ...
    # If propagate down,
    # compute & store input data gradient as well
    if propagate_down[0]:
      bottom[0].diff[...] = ...
    if propagate_down[1]:
      bottom[1].diff[...] = ...
```

```
layer {
  name: "op/python"
  type: "Python"
  bottom: "input1"
  bottom: "input2"
  top: "output1"
  top: "output2"
  python_param {
    module: "new_python_layer"
    layer: "NewPythonLayer"
    param_str: "{ 'num_output': 32 }"
  }
}
```

Object Detection

Object Detection?



- Bounding-box prediction
- Classification per box
- Multiple objects in one image
- Overlaps

Data

- 1 image + M labels
  - 1 label = class & box
  - M is varying
- VOC-2007
  - Images: VOC2007/JPEGImages
  - Labels: VOC2007/Annotations
  - Index: VOC2007/ImageSet/Main
    - trainval.txt, test.txt

```
<annotation>
        <folder>VOC2007</folder>
        <filename>000001.jpg</filename>
        ...
        <object>
Object class name   <name>dog</name>
                <pose>Left</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
Upper-left corner       <xmin>48</xmin>
                        <ymin>240</ymin>
                        <xmax>195</xmax>
Lower-right corner      <ymax>371</ymax>
                </bndbox>
        </object>
        <object>
                <name>person</name>
                <pose>Left</pose>
                <truncated>1</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>8</xmin>
                        <ymin>12</ymin>
                        <xmax>352</xmax>
                        <ymax>498</ymax>
                </bndbox>
        </object>
</annotation>
```

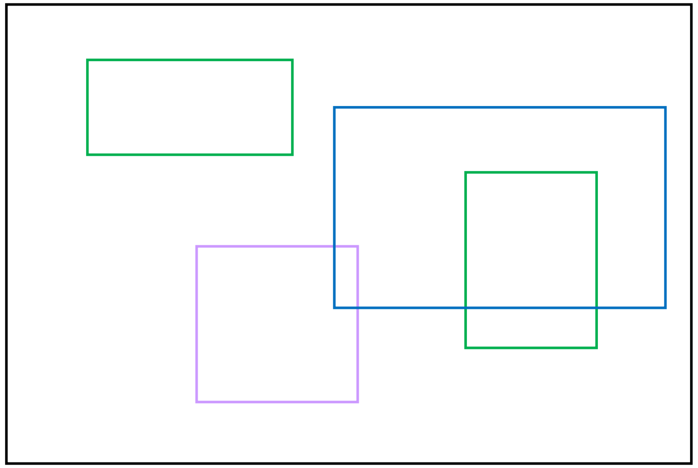
Data Layer

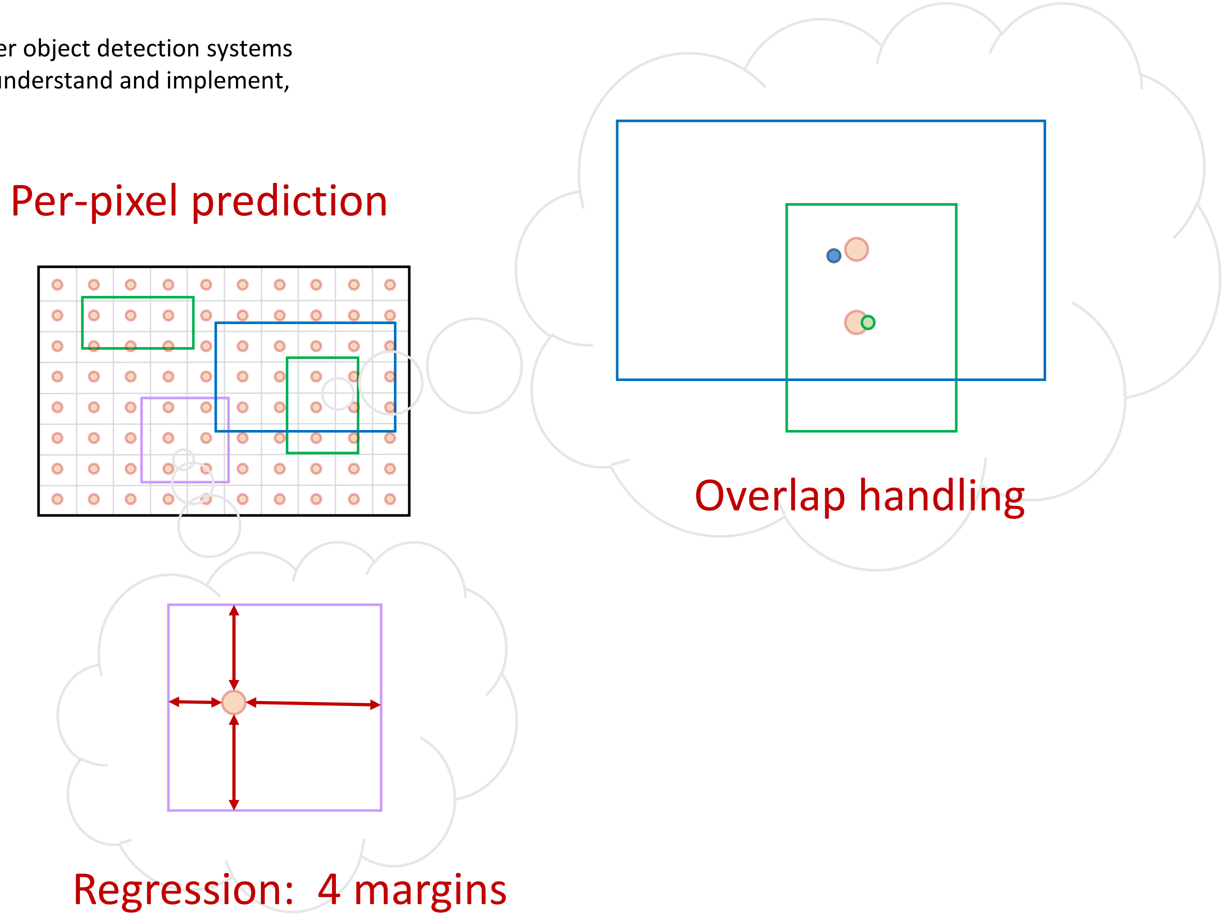
- Parsing 1 .xml and return:
  - Data: 1 x 3 x H x W
  - Label: 1 x ($M_1 + M_2 + \ldots + M_N$)
    - $[0, \; x_{\min}, y_{\min}, \; x_{\max}, y_{\max}]$
- Random scaling: 128p, ..., 640p
- For details, see caffe-materials/practice6/od_data_layer.py

```
layer {
  name: 'input-data'
  type: 'Python'
  top: 'data'
  top: 'label'
  include { phase: TRAIN }
  python_param {
    module: 'od_data_layer'
    layer: 'ODDataLayer'
    param_str: '{ "source": "caffe-materials/practice6/voc2007.txt", "img_dir": "data/VOC2007/JPEGImages", "mean": [103, 116, 123], "base_size": [128, 256, 384, 512, 640] }'
  }
}
```

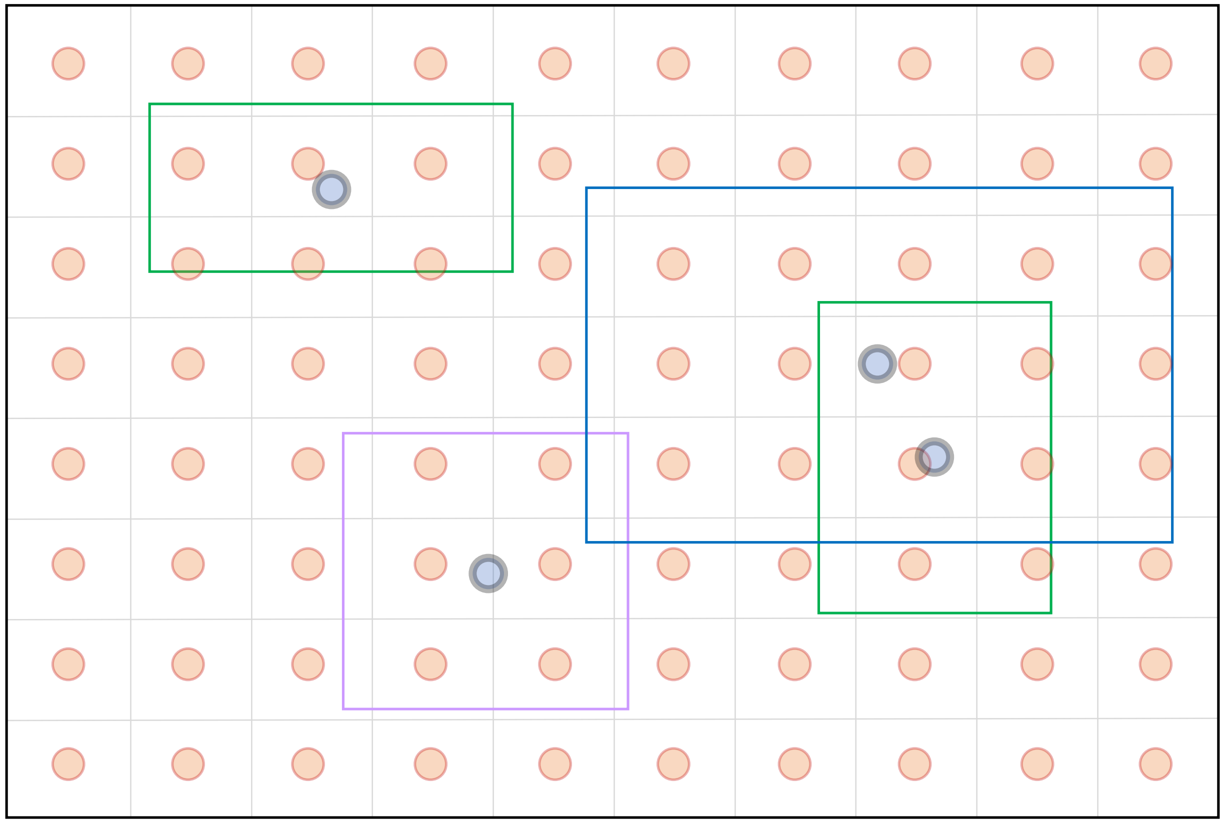
Bounding-Box Prediction

This prediction method is somewhat different from Faster R-CNN and other object detection systems
While the basic idea is almost the same, this is much simpler so easier to understand and implement,
but may produce less accurate predictions





Target Layer



In ODTargetLayer,
a true box is assigned to each point( ) as the regression target for prediction at that point

If a point is not in any of the true boxes, box prediction at that point is ignored in the loss layer

Even a point is in a true box, if the box prediction at that point is really poor (IoU < 0.1) it is also ignored in the loss layer

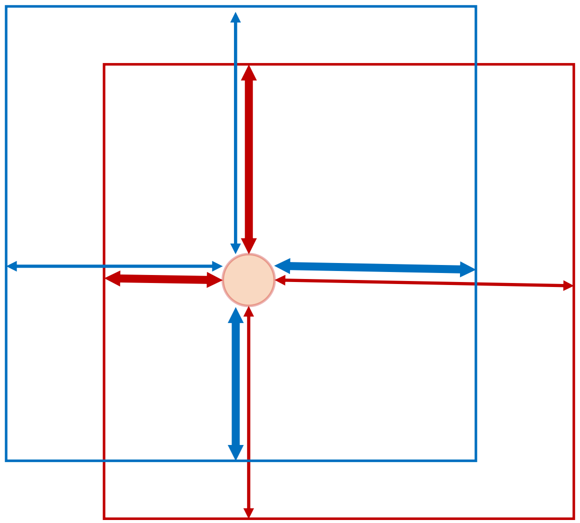
If a point is in an overlapped region of multiple true boxes, the “closest” box is assigned by comparing the distance between the point and the center of each box

The score of “objectness” is also predicted at each point. Regression target (i.e., true score) is the IoU score between the predicted box and its maximally-overlapped true box

Loss Layer

- IoU loss

For more details, see https://arxiv.org/abs/1608.01471
UnitBox: An Advanced Object Detection Network



Regression: 4 margins

$$w = \min(x_L, y_L) + \min(x_R, y_R)$$
$$h = \min(x_T, y_T) + \min(x_B, y_B)$$
$$I = w * h$$

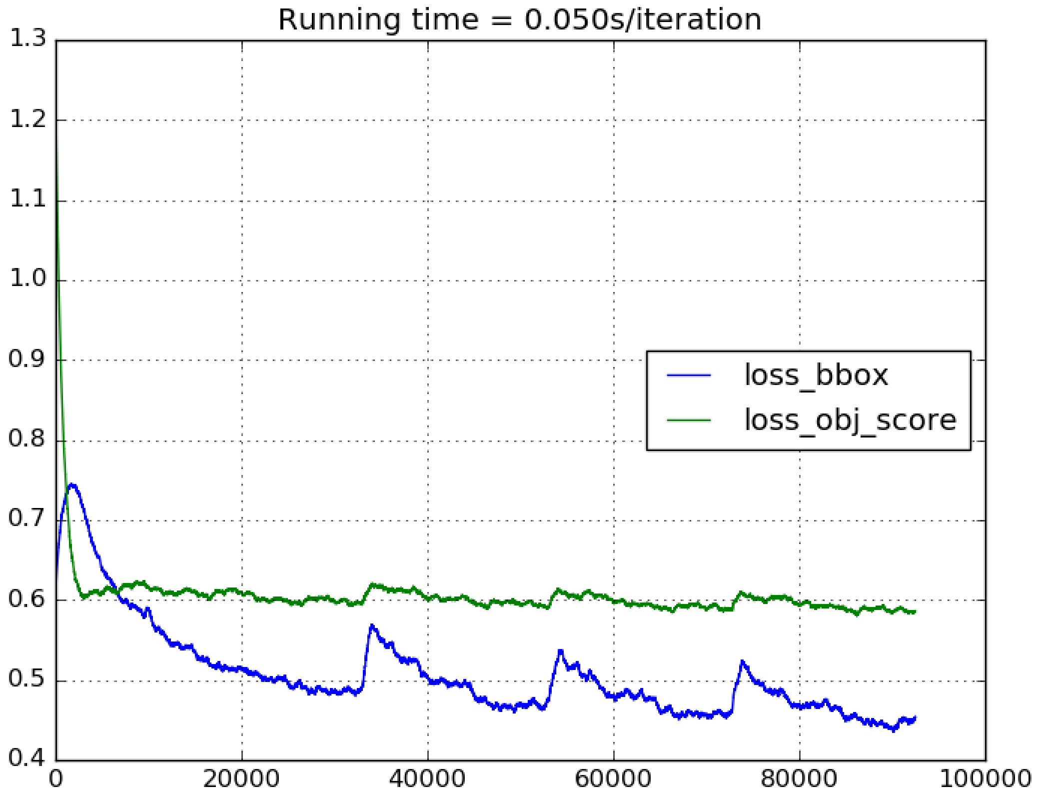
$$A(x) = (x_L + x_R) * (x_T + x_B)$$
$$A(y) = \cdots$$

$$IoU(x, y) = \frac{I}{U} = \frac{I}{A(x) + A(y) - I}$$

$$L = -\ln IoU(x, y)$$
$$\partial(-\ln IoU(x, y)) = -\partial I + \partial U$$

Training

```
~/caffe# export PYTHONPATH=/root/caffe/caffe-materials/practice6:$PYTHONPATH
~/caffe# python caffe-materials/practice6/train.py caffe-materials/practice6/solver.pt caffe-materials/practice6/net.cm
```



Clearing Duplicated Predictions

- Non-Maximum Suppression

Google it for details. Sorry!!

Results

```
~/caffe# export PYTHONPATH=/root/caffe/caffe-materials/practice6:$PYTHONPATH
~/caffe# python caffe-materials/practice6/test.py
```

The network used in this example is my own lightweight CNN, designed for quick test

Summary

- Network design → Data preparation → Training → Testing

Network design	Data preparation	Training	Testing
- Building blocks	- LMDB making & access	- Visualization	- Network data access
- Visualization tool		- Scheduling	
- Python network builder			
- Python layer			

- Some important files
  - src/caffe/proto/caffe.proto   All about Caffe prototxt
  - python/caffe/_caffe.cpp   (`BOOST_PYTHON_MODULE(_caffe){…}`)   All about Python interface
  - include/caffe/{net, solver}.hpp, src/caffe/{net, solver}.cpp

Thank You