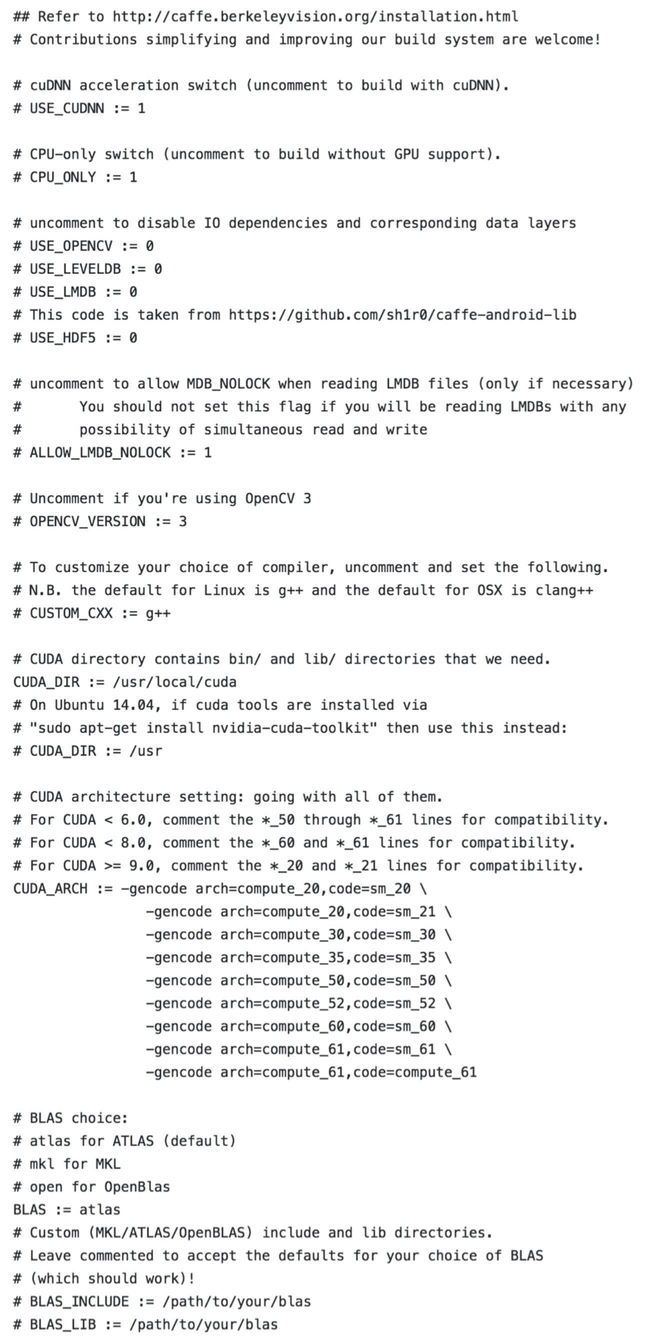
香 港 中 文 大 學
The Chinese University of Hong Kong

CENG5030 Caffe Tutorial

Part I:
Caffe Hands-on

Installation

- Easy customization with Makefile.config



```
## Refer to http://caffe.berkeleyvision.org/installation.html
# Contributions simplifying and improving our build system are welcome!

# cuDNN acceleration switch (uncomment to build with cuDNN).
# USE_CUDNN := 1

# CPU-only switch (uncomment to build without GPU support).
# CPU_ONLY := 1

# uncomment to disable IO dependencies and corresponding data layers
# USE_OPENCV := 0
# USE_LEVELDB := 0
# USE_LMDB := 0
# This code is taken from https://github.com/sh1r0/caffe-android-lib
# USE_HDF5 := 0

# uncomment to allow MDB_NOLOCK when reading LMDB files (only if necessary)
#       You should not set this flag if you will be reading LMDBs with any
#       possibility of simultaneous read and write
# ALLOW_LMDB_NOLOCK := 1

# Uncomment if you're using OpenCV 3
# OPENCV_VERSION := 3

# To customize your choice of compiler, uncomment and set the following.
# N.B. the default for Linux is g++ and the default for OSX is clang++
# CUSTOM_CXX := g++

# CUDA directory contains bin/ and lib/ directories that we need.
CUDA_DIR := /usr/local/cuda
# On Ubuntu 14.04, if cuda tools are installed via
# "sudo apt-get install nvidia-cuda-toolkit" then use this instead:
# CUDA_DIR := /usr

# CUDA architecture setting: going with all of them.
# For CUDA < 6.0, comment the *_50 through *_61 lines for compatibility.
# For CUDA < 8.0, comment the *_60 and *_61 lines for compatibility.
# For CUDA >= 9.0, comment the *_20 and *_21 lines for compatibility.
CUDA_ARCH := -gencode arch=compute_20,code=sm_20 \
                -gencode arch=compute_20,code=sm_21 \
                -gencode arch=compute_30,code=sm_30 \
                -gencode arch=compute_35,code=sm_35 \
                -gencode arch=compute_50,code=sm_50 \
                -gencode arch=compute_52,code=sm_52 \
                -gencode arch=compute_60,code=sm_60 \
                -gencode arch=compute_61,code=sm_61 \
                -gencode arch=compute_61,code=compute_61

# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
BLAS := atlas
# Custom (MKL/ATLAS/OpenBLAS) include and lib directories.
# Leave commented to accept the defaults for your choice of BLAS
# (which should work)!
# BLAS_INCLUDE := /path/to/your/blas
# BLAS_LIB := /path/to/your/blas
```

Prepare Data

- Data stored in LMDB format;
- The conversion tool convert_imageset is provided;
- Example: caffe/examples/imagenet/create_imagenet.sh ;
- Need text file where each line is
  - “[path/to/image.jpeg] [label]”

Prototxt for Net definition

```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
```

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

```
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

Prototxt for Solver Definition

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: GPU
```

Execution

- Usage: caffe <command> <args>

- Commands:

train	train or finetune a model
test	score a model
device_query	show GPU diagnostic information
time	benchmark model execution time

- Flags (check all the flags in tools/caffe.cpp)

- Example:

```
~$ cd CAFFE_ROOT
~$ ./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt --gpu 0
~$ ./build/tools/caffe test  --model=examples/mnist/lenet_solver.prototxt --weight YOUR_MODEL.caffemodel
    --gpu 0
~$ ./build/tools/caffe time --model=examples/mnist/lenet_solver.prototxt --weight YOUR_MODEL.caffemodel
    --gpu 0
```

Caffe Demo

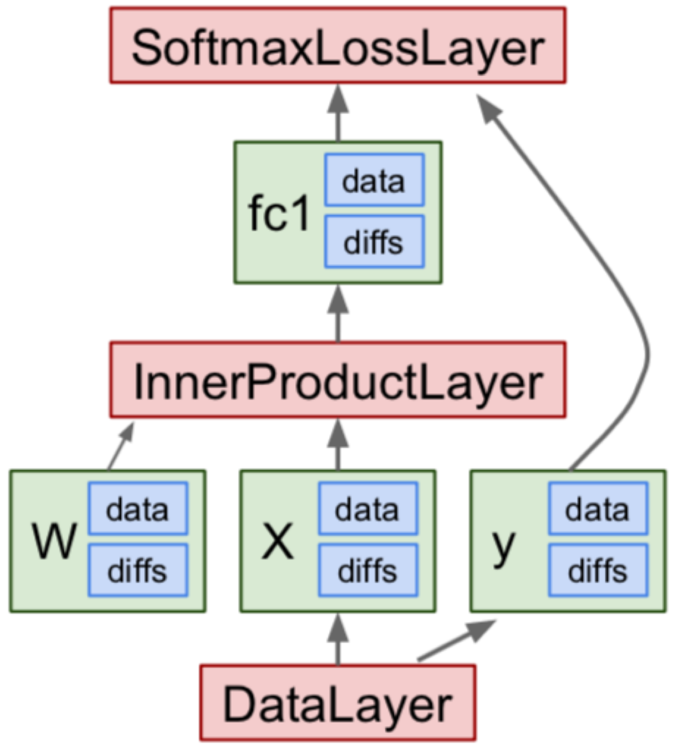
Part II:
Know More About Caffe

Most important tip (from Stanford CS231n)

Don’t be afraid to read the code!

Main Classes

- Blob: stores data;
- Layer: Calculation and transformation.
  - Input: Bottom blob(s);
  - Output: Top blob(s).
- Net: A bunch of layers.
- Solver: Train a model using backpropagation.



Source: http://vision.stanford.edu/teaching/cs231n/slides/2015/caffe_tutorial.pdf

Prototxt for Net definition

```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
```

```
// NOTE
// Update the next available ID when you add a new LayerParameter field.
//
// LayerParameter next available layer-specific ID: 131 (last added: python_param)
message LayerParameter {
  optional string name = 1; // the layer name
  optional string type = 2; // the layer type
  repeated string bottom = 3; // the name of each bottom blob
  repeated string top = 4; // the name of each top blob

  // The train / test phase for computation.
  optional Phase phase = 10;
```

```
// Message that stores parameters used by DataLayer
message DataParameter {
  enum DB {
    LEVELDB = 0;
    LMDB = 1;
  }
  // Specify the data source.
  optional string source = 1;
  // Specify the batch size.
  optional uint32 batch_size = 4;
  // The rand_skip variable is for the data layer to skip a few data points
  // to avoid all asynchronous sgd clients to start at the same point. The skip
  // point would be set as rand_skip * rand(0,1). Note that rand_skip should not
  // be larger than the number of keys in the database.
  optional uint32 rand_skip = 7 [default = 0];
  optional DB backend = 8 [default = LEVELDB];
  // DEPRECATED. See TransformationParameter. For data pre-processing, we can do
  // simple scaling and subtracting the data mean, if provided. Note that the
  // mean subtraction is always carried out before scaling.
  optional float scale = 2 [default = 1];
  optional string mean_file = 3;
  // DEPRECATED. See TransformationParameter. Specify if we would like to randomly
  // crop an image.
  optional uint32 crop_size = 5 [default = 0];
  // DEPRECATED. See TransformationParameter. Specify if we want to randomly mirror
  // data.
  optional bool mirror = 6 [default = false];
```

Prototxt for Net definition

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

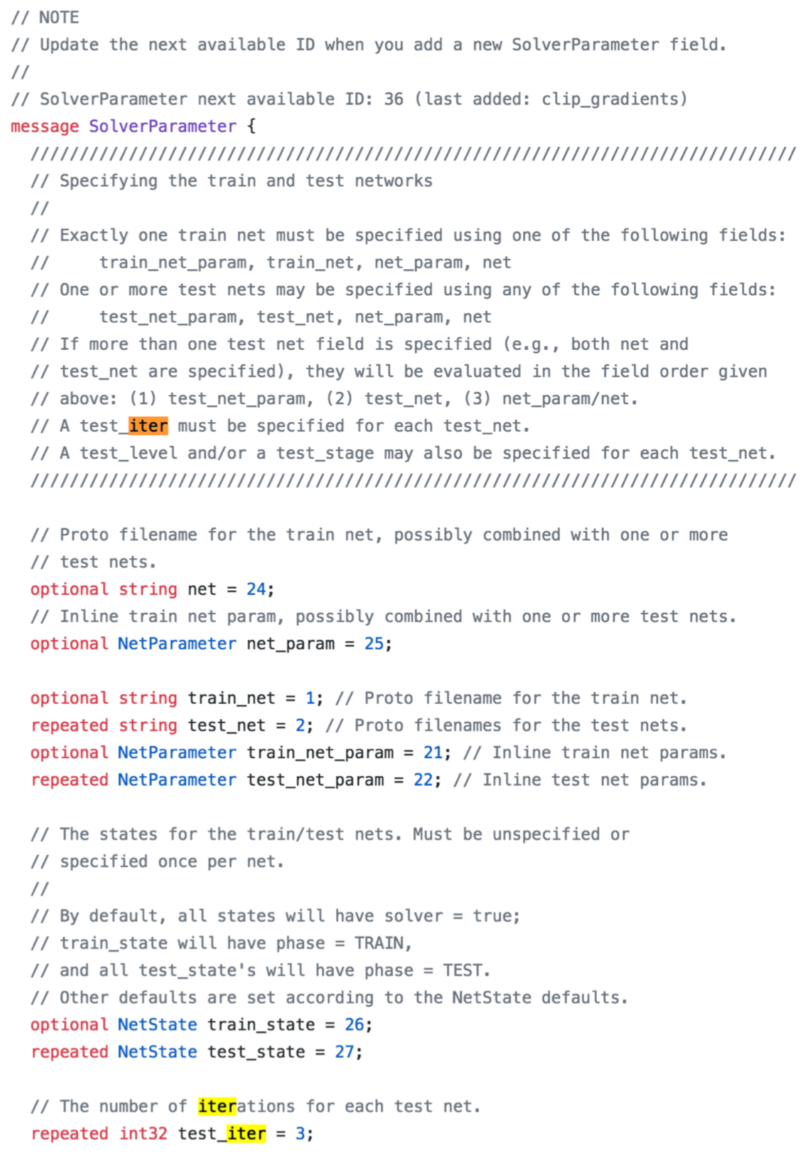
LR coef. for weight and bias

```
// Message that stores parameters used by ConvolutionLayer
message ConvolutionParameter {
  optional uint32 num_output = 1; // The number of outputs for the layer
  optional bool bias_term = 2 [default = true]; // whether to have bias terms
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pad = 3 [default = 0]; // The padding size (equal in Y, X)
  optional uint32 pad_h = 9 [default = 0]; // The padding height
  optional uint32 pad_w = 10 [default = 0]; // The padding width
  optional uint32 kernel_size = 4; // The kernel size (square)
  optional uint32 kernel_h = 11; // The kernel height
  optional uint32 kernel_w = 12; // The kernel width
  optional uint32 group = 5 [default = 1]; // The group size for group conv
  optional uint32 stride = 6 [default = 1]; // The stride (equal in Y, X)
  optional uint32 stride_h = 13; // The stride height
  optional uint32 stride_w = 14; // The stride width
  optional FillerParameter weight_filler = 7; // The filler for the weight
  optional FillerParameter bias_filler = 8; // The filler for the bias
  enum Engine {
    DEFAULT = 0;
    CAFFE = 1;
    CUDNN = 2;
  }
  optional Engine engine = 15 [default = DEFAULT];
}
```

What if we just want to do training on some layers, not all layers?

Prototxt for Solver Definition

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: GPU
```



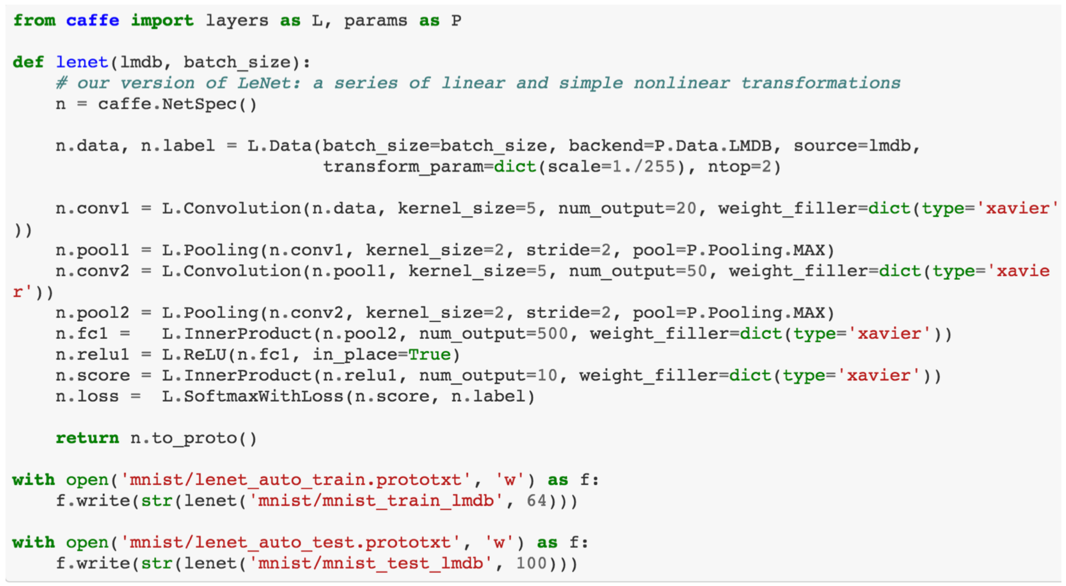
```
// NOTE
// Update the next available ID when you add a new SolverParameter field.
//
// SolverParameter next available ID: 36 (last added: clip_gradients)
message SolverParameter {
  //////////////////////////////////////////////////////////////////////////////
  // Specifying the train and test networks
  //
  // Exactly one train net must be specified using one of the following fields:
  //     train_net_param, train_net, net_param, net
  // One or more test nets may be specified using any of the following fields:
  //     test_net_param, test_net, net_param, net
  // If more than one test net field is specified (e.g., both net and
  // test_net are specified), they will be evaluated in the field order given
  // above: (1) test_net_param, (2) test_net, (3) net_param/net.
  // A test_iter must be specified for each test_net.
  // A test_level and/or a test_stage may also be specified for each test_net.
  //////////////////////////////////////////////////////////////////////////////

  // Proto filename for the train net, possibly combined with one or more
  // test nets.
  optional string net = 24;
  // Inline train net param, possibly combined with one or more test nets.
  optional NetParameter net_param = 25;

  optional string train_net = 1; // Proto filename for the train net.
  repeated string test_net = 2; // Proto filenames for the test nets.
  optional NetParameter train_net_param = 21; // Inline train net params.
  repeated NetParameter test_net_param = 22; // Inline test net params.

  // The states for the train/test nets. Must be unspecified or
  // specified once per net.
  //
  // By default, all states will have solver = true;
  // train_state will have phase = TRAIN,
  // and all test_state's will have phase = TEST.
  // Other defaults are set according to the NetState defaults.
  optional NetState train_state = 26;
  repeated NetState test_state = 27;

  // The number of iterations for each test net.
  repeated int32 test_iter = 3;
```

PyCaffe

- Wrap the internal caffe C++ module (_caffe.so) with a clean, Pythonic interface.
- caffe/python/caffe/_caffe.cpp, caffe/python/caffe/pycaffe.py
- Blobs:
  - Numpy arrays.
- Layers:
  - layer.blobs is a list of Blobs
- Nets:
  - Have methods like forward() and backward() for calculation.
- Solver:
  - Can be directly defined in Python.

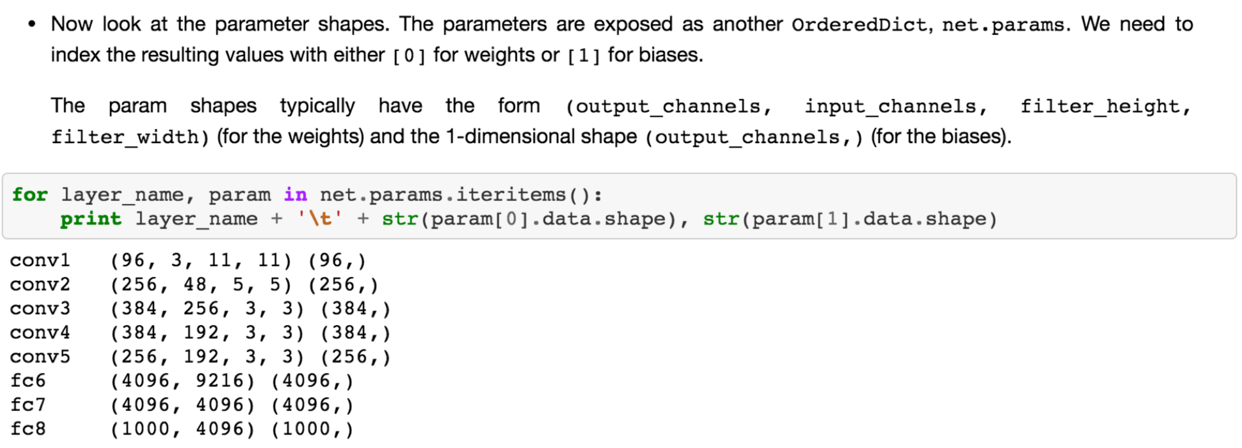
PyCaffe Example

- Tutorial from BVLC caffe/examples/01-learning-lenet.ipynb



```
from caffe import layers as L, params as P

def lenet(lmdb, batch_size):
    # our version of LeNet: a series of linear and simple nonlinear transformations
    n = caffe.NetSpec()

    n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB, source=lmdb,
                             transform_param=dict(scale=1./255), ntop=2)

    n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20, weight_filler=dict(type='xavier'
))
    n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50, weight_filler=dict(type='xavie
r'))
    n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.fc1 =   L.InnerProduct(n.pool2, num_output=500, weight_filler=dict(type='xavier'))
    n.relu1 = L.ReLU(n.fc1, in_place=True)
    n.score = L.InnerProduct(n.relu1, num_output=10, weight_filler=dict(type='xavier'))
    n.loss =  L.SoftmaxWithLoss(n.score, n.label)

    return n.to_proto()

with open('mnist/lenet_auto_train.prototxt', 'w') as f:
    f.write(str(lenet('mnist/mnist_train_lmdb', 64)))

with open('mnist/lenet_auto_test.prototxt', 'w') as f:
    f.write(str(lenet('mnist/mnist_test_lmdb', 100)))
```

Define a net



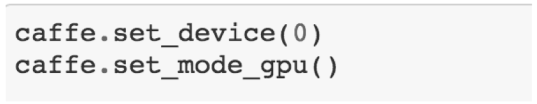
- Now look at the parameter shapes. The parameters are exposed as another `OrderedDict`, `net.params`. We need to index the resulting values with either `[0]` for weights or `[1]` for biases.

  The param shapes typically have the form `(output_channels, input_channels, filter_height, filter_width)` (for the weights) and the 1-dimensional shape `(output_channels,)` (for the biases).

```
for layer_name, param in net.params.iteritems():
    print layer_name + '\t' + str(param[0].data.shape), str(param[1].data.shape)
```

```
conv1	(96, 3, 11, 11) (96,)
conv2	(256, 48, 5, 5) (256,)
conv3	(384, 256, 3, 3) (384,)
conv4	(384, 192, 3, 3) (384,)
conv5	(256, 192, 3, 3) (256,)
fc6	(4096, 9216) (4096,)
fc7	(4096, 4096) (4096,)
fc8	(1000, 4096) (1000,)
```

Access data in net layers



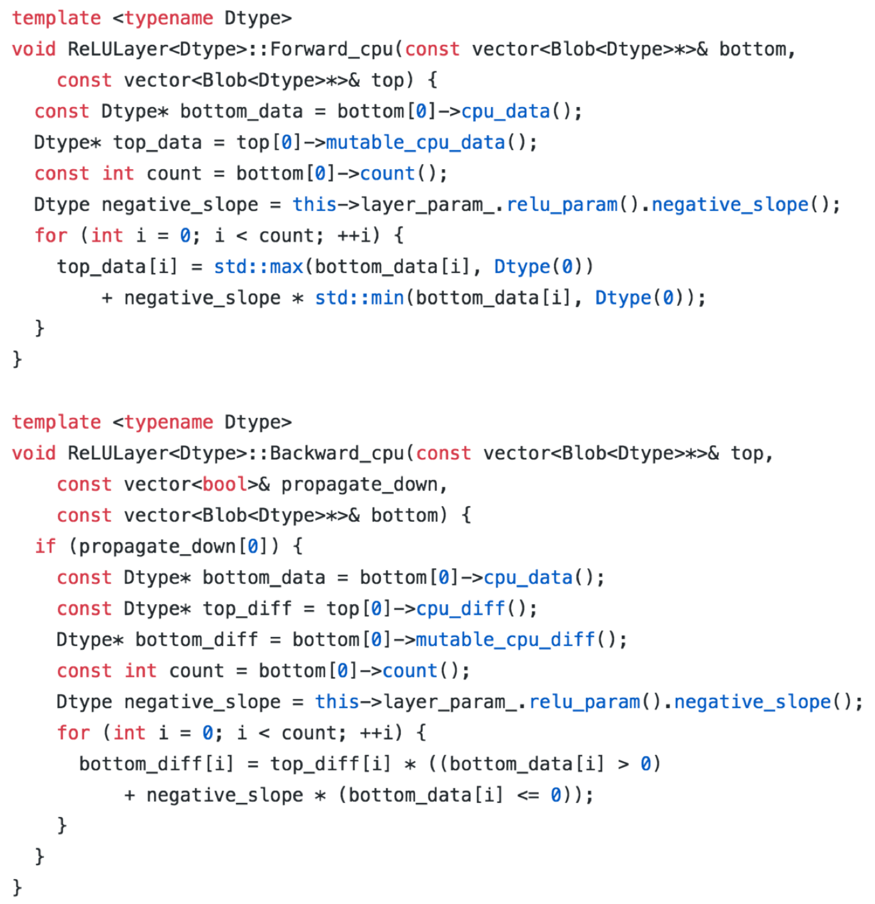
```
caffe.set_device(0)
caffe.set_mode_gpu()
```

Mode configure

PyCaffe Demo

Operations in Layers

- Forward calculation;
- Backward calculation;
- Other utility functions.



```cpp
template <typename Dtype>
void ReLULayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  Dtype negative_slope = this->layer_param_.relu_param().negative_slope();
  for (int i = 0; i < count; ++i) {
    top_data[i] = std::max(bottom_data[i], Dtype(0))
        + negative_slope * std::min(bottom_data[i], Dtype(0));
  }
}

template <typename Dtype>
void ReLULayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* bottom_data = bottom[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    Dtype negative_slope = this->layer_param_.relu_param().negative_slope();
    for (int i = 0; i < count; ++i) {
      bottom_diff[i] = top_diff[i] * ((bottom_data[i] > 0)
          + negative_slope * (bottom_data[i] <= 0));
    }
  }
}
```

relu_layer.cpp

Operations in Layers

- Computations in some layers may rely on other libraries.
    - BLAS for matrix operation. (CONV layer, Inner_Product layer)
    - CUDA for GPU-based computation.
- If you have smart ideas for training/inference speedup:
    - Implement in corresponding layers;
    - Or develop your own layers.

Develop Your Layer

- Declaration
  - Add a class declaration for your layer to include/caffe/layers/your_layer.hpp
- Implementation
  - Implement your layer in src/caffe/layers/your_layer.cpp
- (Optional) GPU Suppport
  - Implement the GPU versions Forward_gpu and Backward_gpu in layers/your_layer.cu.
- If needed, declare parameters in proto/caffe.proto
- Instantiate
  - Instantiate and register your layer in your_layer.cpp with the macro provided in layer_factory.hpp
- Caffe Wiki Development

Q & A