Caffe Tutorial

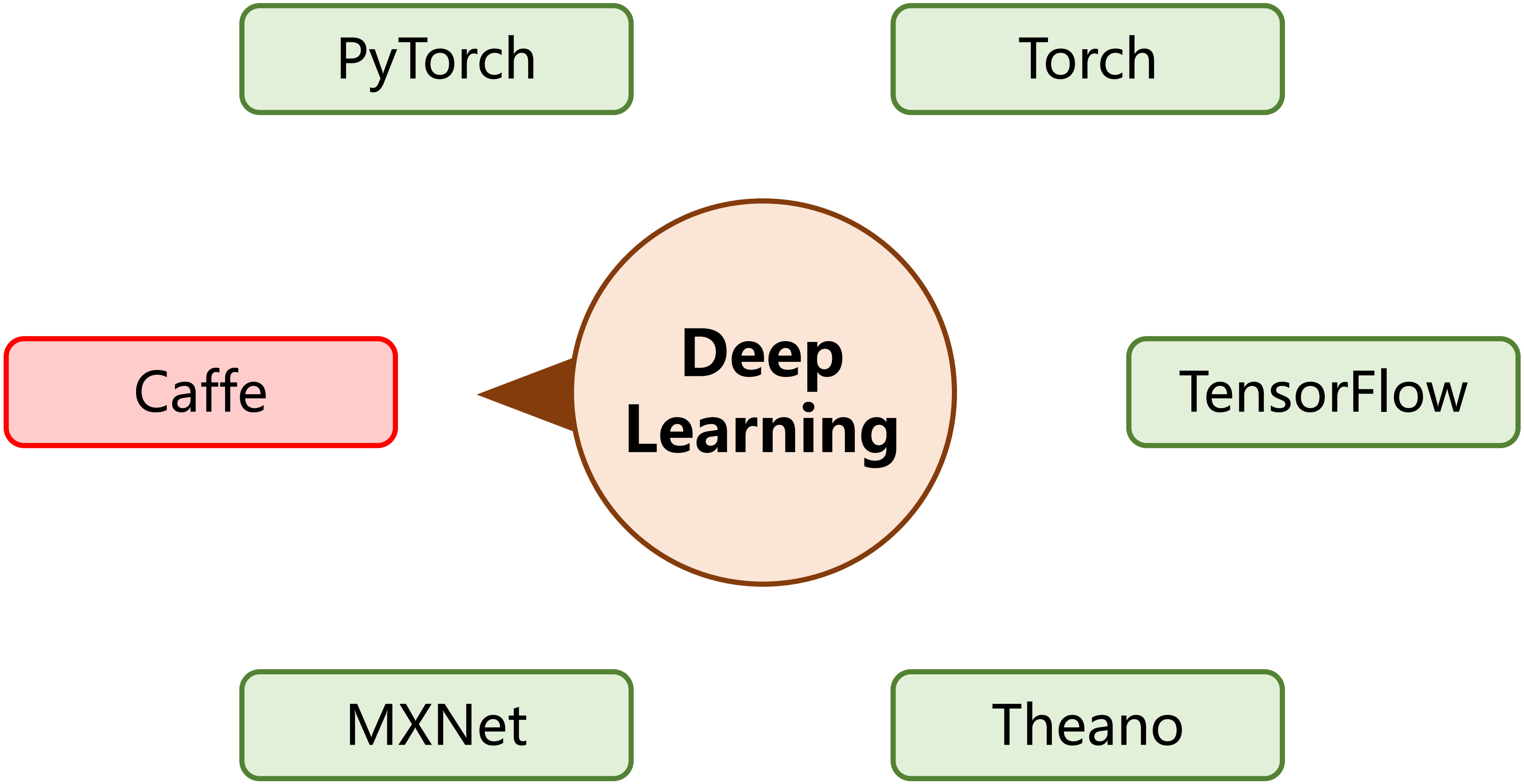
Zhuangwei Zhuang

Southern **A**rtificial **I**ntelligence **L**aboratory

South China University of Technology

July 26, 2017

Platforms for Deep Learning



Outline

- **Introduction**
- **Deep learning in Caffe**
  - Architecture of Caffe
  - Experiment Pipeline
- **Deeper into Caffe**
  - Blob
  - Layers
  - Net
  - Solver
- **Develop new layers**
- **Python interface**

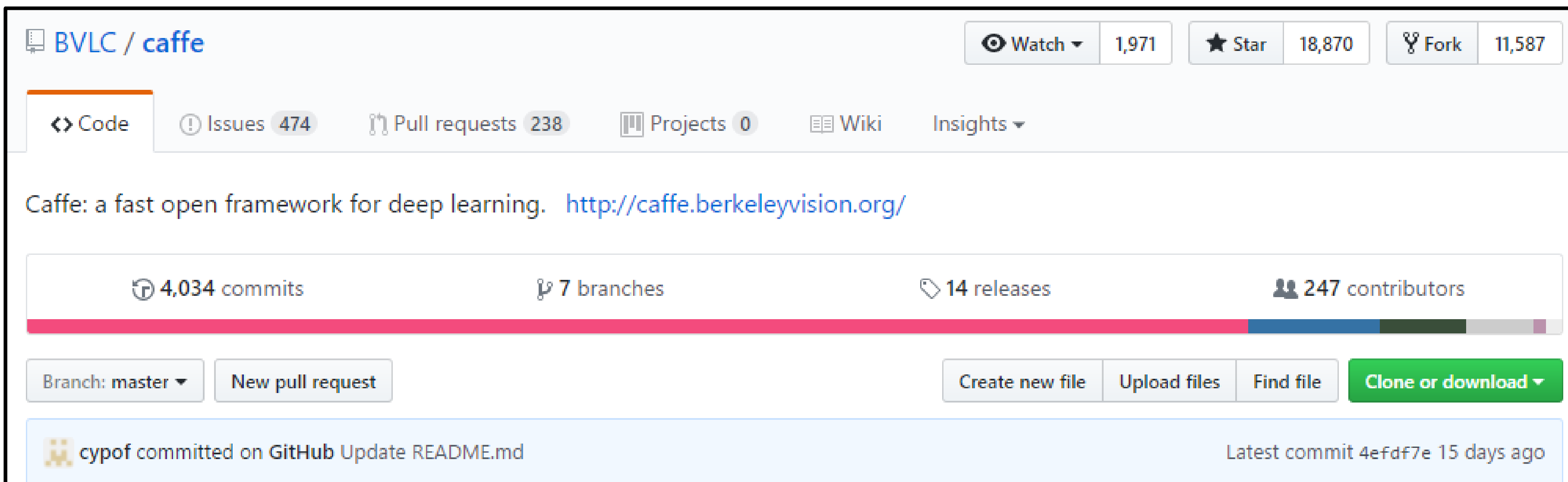
Introduction

What is Caffe

Convolution **A**rchitecture **F**or **F**eature **E**xtraction (CAFFE)

Open framework, models, and worked examples for deep learning

- 600+ citations, 200+ contributions, 18K+ stars, 11K+forks
- focus on vision, but branching out: sequences, reinforcement learning, speech + text



What is Caffe

- Pure C++/CUDA architecture for deep learning
  - command line, Python, Matlab interfaces
- Fast, well-tested code
- Tools, reference models, demos, and recipes
- Seamless switch between CPU and GPU
  - Caffe::set_model (Caffe::GPU)

Official Websites

- **Home page**: http://caffe.berkeleyvision.org/
- **GitHub**: https://github.com/BVLC/caffe

Documentation

- DIY Deep Learning for Vision w
  Tutorial presentation of the fra
- Tutorial Documentation
  Practical guide and framework
- arXiv / ACM MM '14 paper
  A 4-page report for the ACM M
- Installation instructions
  Tested on Ubuntu, Red Hat, OS
- Model Zoo
  BAIR suggests a standard distr
- Developing & Contributing
  Guidelines for development ar
- API Documentation
  Developer documentation aut
- Benchmarking
  Comparison of inference and l

Notebook Examples

- Image Classification and Filte
  Instant recognition with a pre
  features and parameters laye
- Learning LeNet
  Define, train, and test the cla
- Fine-tuning for Style Recognit
  Fine-tune the ImageNet-train
- Off-the-shelf SGD for classific
  Use Caffe as a generic SGD o
- Multilabel Classification with
  Multilabel classification on PA
- Editing model parameters
  How to do net surgery and m
- R-CNN detection
  Run a pretrained model as a
- Siamese network embedding
  Extracting features and plotti

Command Line Examples

- ImageNet tutorial
  Train and test "CaffeNet" on ImageNet data.
- LeNet MNIST Tutorial
  Train and test "LeNet" on the MNIST handwritten digit data.
- CIFAR-10 tutorial
  Train and test Caffe on CIFAR-10 data.
- Fine-tuning for style recognition
  Fine-tune the ImageNet-trained CaffeNet on the "Flickr Style" dataset.
- Feature extraction with Caffe C++ code.
  Extract CaffeNet / AlexNet features using the Caffe utility.
- CaffeNet C++ Classification example
  A simple example performing image classification using the low-level C++ API.
- Web demo
  Image classification demo running as a Flask web server.
- Siamese Network Tutorial
  Train and test a siamese network on MNIST data.

Installation

- http://caffe.berkeleyvision.org/installation.html
- Step-by-step Instructions:
  - Ubuntu installation the standard platform
  - Debian installation install caffe with a single command
  - Windows see the Windows branch led by Guillaume Dumont
  - OpenCL see the OpenCL branch led by Fabian Tschopp
  - …
- Overview:
  - Prerequisites
  - Compilation
  - Hardware

Installation

➢ **Windows:** https://github.com/BVLC/caffe/tree/windows

Windows Caffe

This is an experimental, communtity based branch led by Guillaume Dumont (@willyd). It is a work-in-progress.

This branch of Caffe ports the framework to Windows.

build passing Travis (Linux build)

build passing AppVeyor (Windows build)

Prebuilt binaries

Prebuilt binaries can be downloaded from the latest CI build on appveyor for the following configurations:

- Visual Studio 2015, CPU only, Python 3.5: Caffe Release, ~~Caffe Debug~~
- Visual Studio 2015, CUDA 8.0, Python 3.5: Caffe Release
- Visual Studio 2015, CPU only, Python 2.7: Caffe Release, Caffe Debug
- Visual Studio 2015,CUDA 8.0, Python 2.7: Caffe Release
- Visual Studio 2013, CPU only, Python 2.7: Caffe Release, Caffe Debug

Installation

- **Linux:** http://caffe.berkeleyvision.org/install_apt.html
- General dependencies

```
sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev
libopencv-dev libhdf5-serial-dev protobuf-compiler
sudo apt-get install --no-install-recommends libboost-all-dev
```

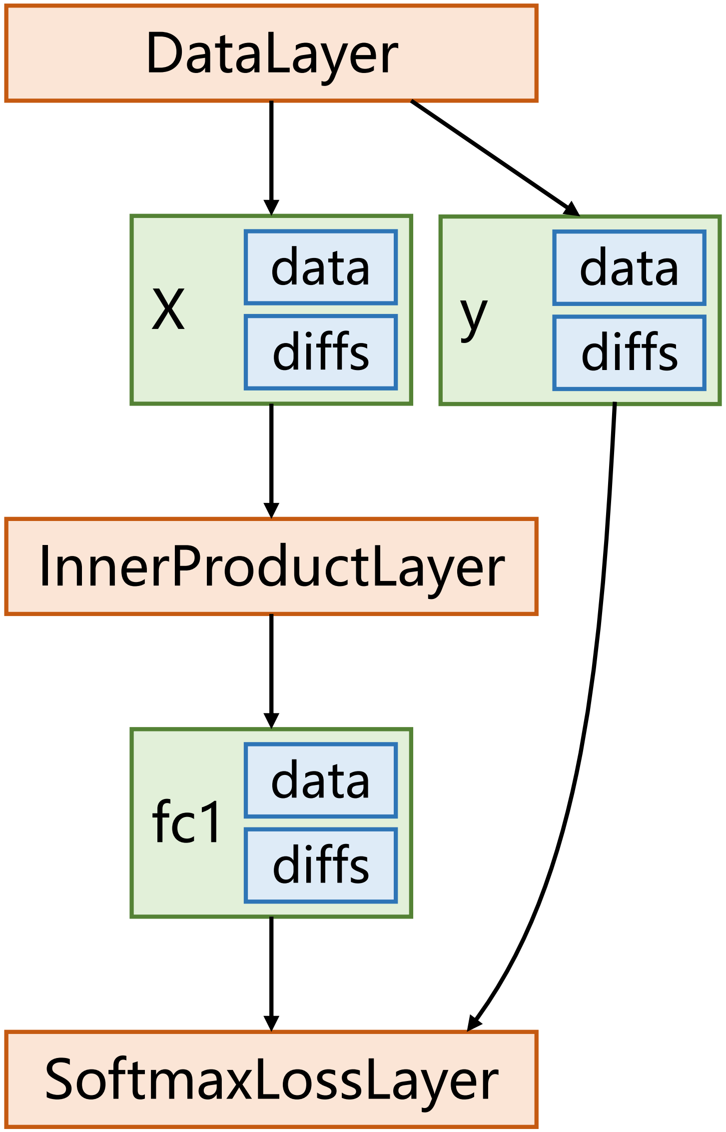
- CUDA
- BLAS
- Python
- Compilation with Make

Deep Learning in Caffe

Architecture of Caffe

➢ Main classes:

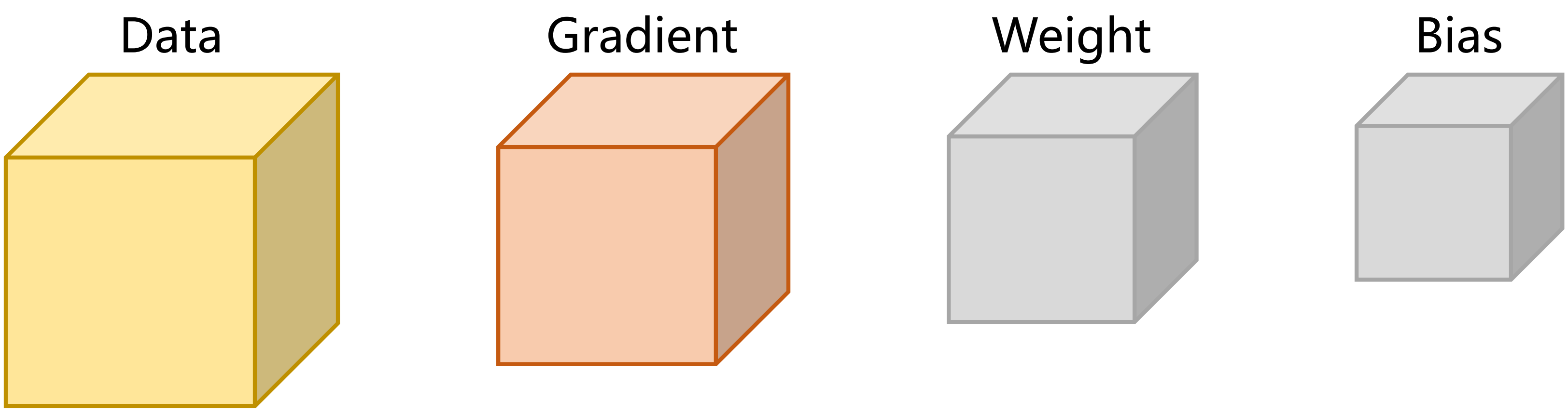
- **blob:** store data and derivatives
- **layers:** transforms bottom blobs to top blobs
- **net:** consists of many layers, computes gradients via forward/backward
- **solver:** uses gradients to update weights



Blobs

N-D arrays for storing and communicating data

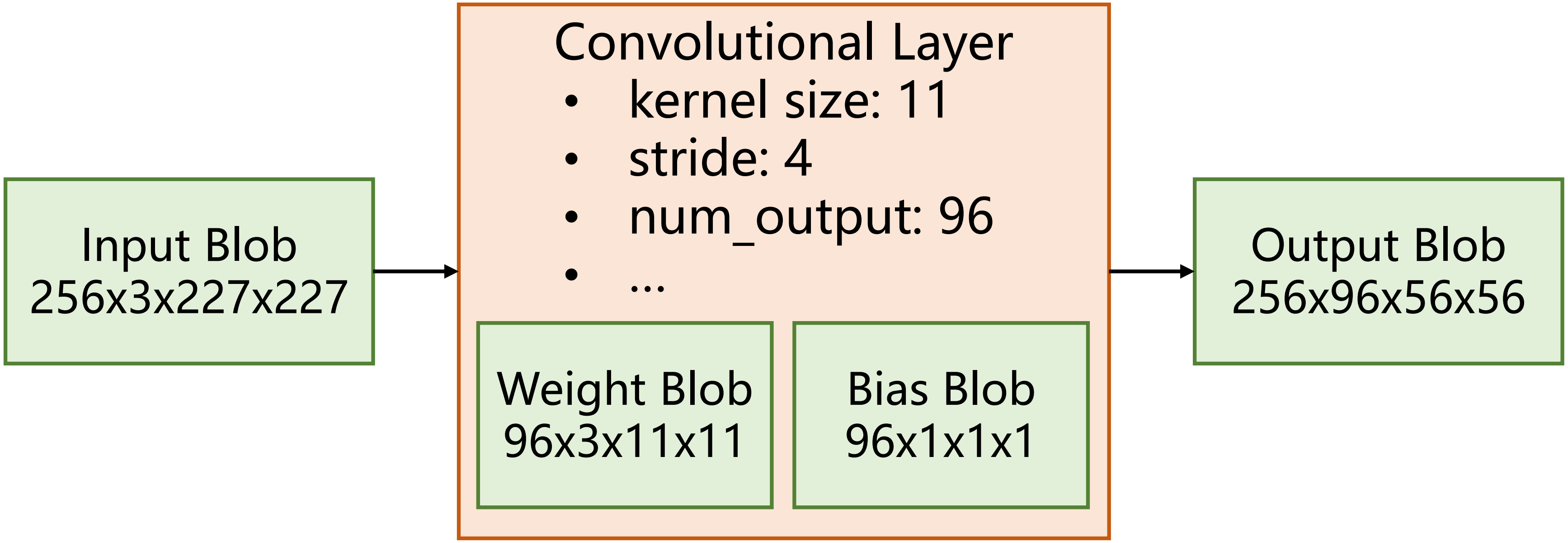
- hold data, derivatives and parameters
- lazily allocate memory
- shuttle between CPU and GPU



Layers

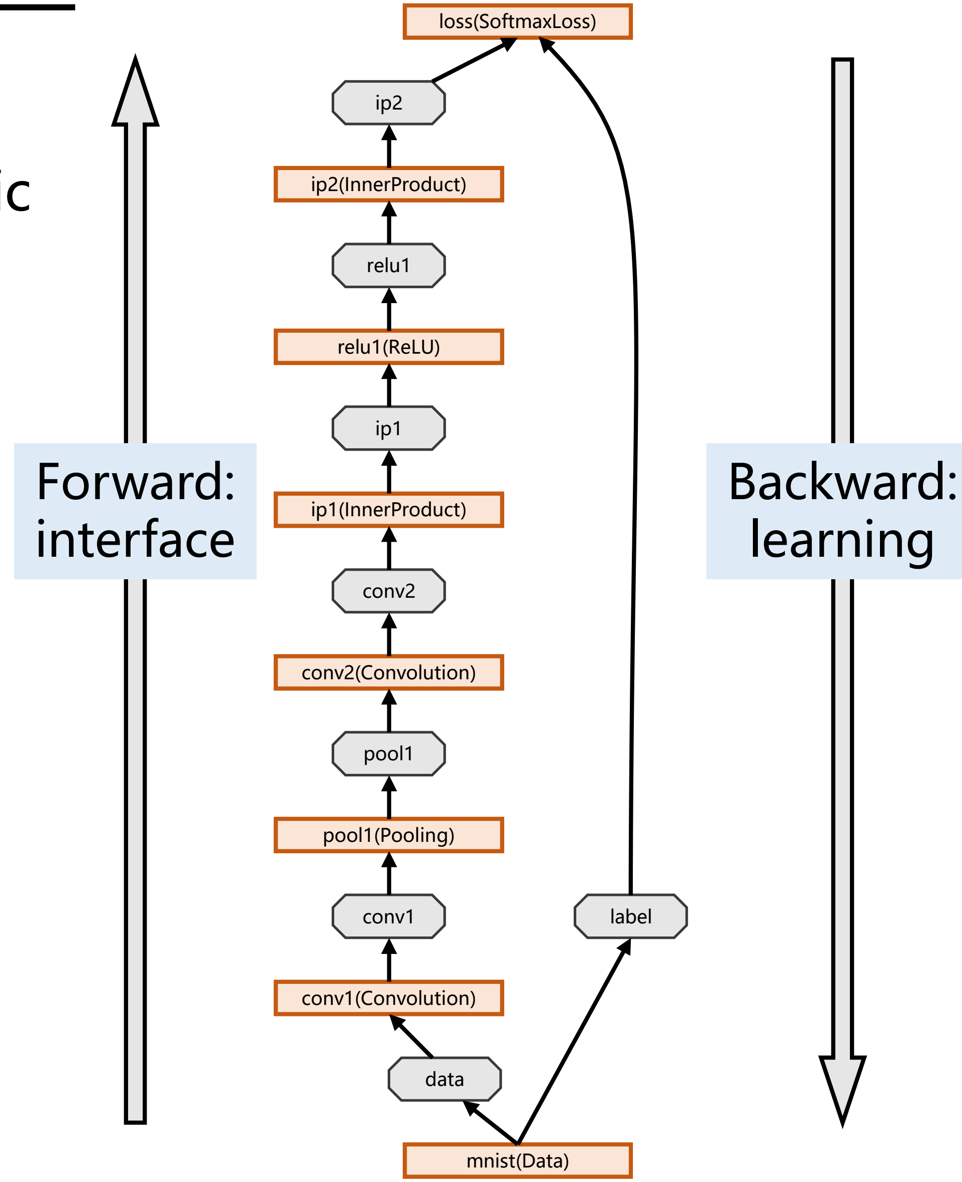
Caffe's fundamental unit of computation

- data access
- convolution
- pooling
- ...



Net

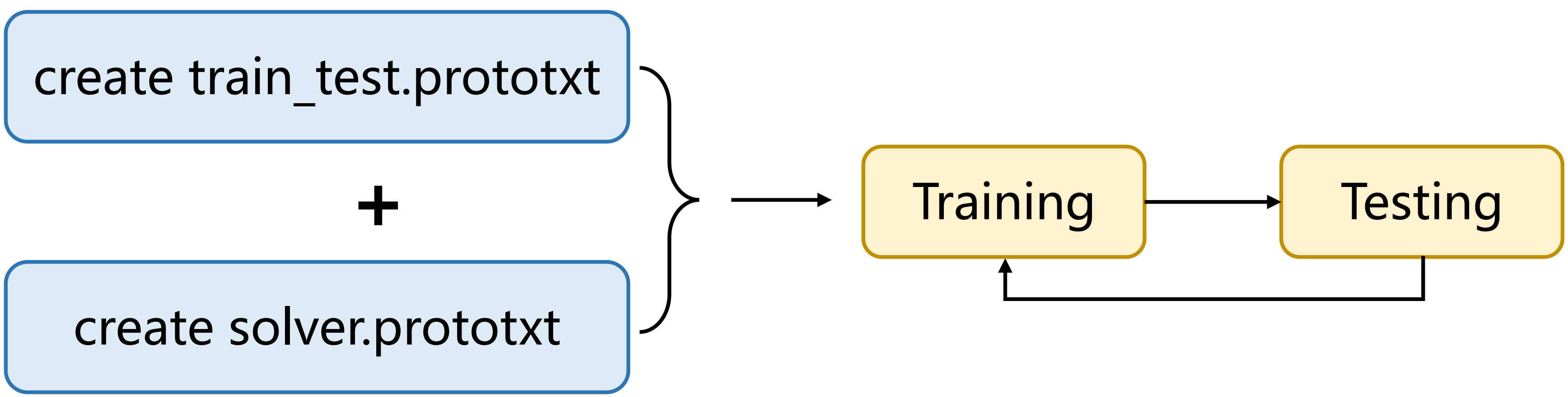
- A network is a set of layers connected as a directed acyclic graph (DAG)
- Caffe creates and checks the net from the definition
- Data and derivatives flow through the net as blobs



Experiment Pipeline

➢ Main files:

- **train_test.prototxt (deploy.prototxt):** descript structure of the network
- **solver.prototxt:** descript training parameters



Experiment Pipeline

➤ Step 1: create a train_test.prototxt

Data

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 28
    mean_file:
"/home/../data/cifar10/cifar10_mean.binaryproto"
  }
  data_param {
    source: "/home/../data/cifar10/cifar10_train_lmdb"
    batch_size: 128
    backend: LMDB
  }
}
```

Layers

```
layer {
  name: "conv_1"
  type: "Convolution"
  bottom: "data"
  top: "conv_1"
  param {
    lr_mult: 1.0
    decay_mult: 1.0 }
  convolution_param {
    num_output: 16
    pad: 1
    kernel_size: 3
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.117851130198 }
    bias_filler {
      type: "constant"
      value: 0.0
    }  } }
```

Loss

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc10"
  bottom: "label"
  top: "loss"
}
```

Experiment Pipeline

➢ Step 2: create solver.prototxt

```
train_net:          "train_test.prototxt"
base_lr:            0.01
momentum:           0.9
weight_decay:       0.0001
max_iter:           10000
snapshot_prefix:    "snapshot"
# ... other options ...
```

➢ Details on SGD parameters

$$V_{t+1} = \underbrace{\mu}_{\text{Momentum}} V_t - \underbrace{\alpha}_{\text{LR}}(\nabla \mathrm{L}(\mathrm{W_t}) + \underbrace{\lambda}_{\text{Decay}} W_t)$$

$$W_{t+1} = W_t + V_{t+1}$$

Experiment Pipeline

- Step 3: training and testing

$ build/tools.caffe train –solver solver.prototxt –gpu 0

command

attributes

values

Deeper into Caffe

Blob

- Reshape(num, channel, height, width)
  - declare dimensions
- cpu_data(), mutable_cpu_data()
  - host memory for CPU mode
- gpu_data(), mutable_cpu_data()
  - device memory for GPU mode
- {cpu, gpu}_diff(), mutable_{cpu, gpu}_diff()
  - derivative counterparts to data methods

SyncedMem
allocation + communication

Blob

GPU/CPU switch

- Use synchronized memory
- Mutable/non-mutable determines whether to copy. Use of mutable_* may lead to data copy
- Use {cpu/gpu}_data

Protocol Buffer

- **Protocol buffer:**
  - Like strongly typed, binary JSON
  - Developed by Google
  - Define message types in .proto file
  - Define messages in .prototxt or binaryproto files (Caffe also uses .caffemodel)
  - All Caffe messages defined in caffe.proto
- train_test.prototxt (deploy.prototxt): descript structure of the network
- solver.prototxt: descript training parameters

Layers

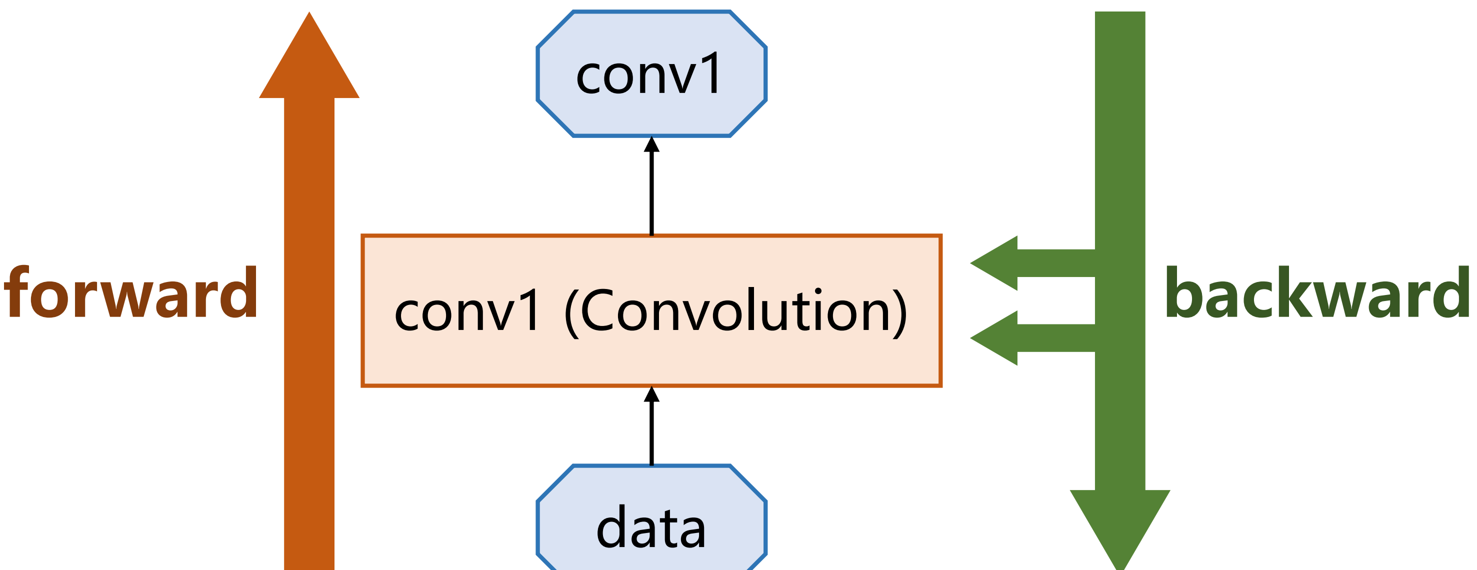
- http://caffe.berkeleyvision.org/tutorial/layers.html
- Data layers
  - **Image data:** read raw images
  - **Database:** read data from LEVELDB or LMDB
  - …
- Vision layers
  - **Convolutional layer:** convolves the input image learnable filters, each producing one feature map in the output image
  - **Pooling layer:** max, average, or stochastic pooling
  - …

Layers

- Common layers
  - **Inner product:** fully connected layer
  - …
- Normalization layers
  - **Batch Normalization:** performs normalization
  - …
- Activation / Neuron layers:
  - **ReLU / Rectified-Linear and Leaky-ReLU**
  - **Sigmoid**
  - **…**
- …

Layers

- **Setup:** run once for initialization
- **Forward:** make output given input
- **Backward:** make gradient of output
  - w.r.t. bottom
  - w.r.t. parameters (if needed)



Data Layer

- get definition from ./src/caffe/proto/caffe.proto

```
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
  // to avoid all asynchronous sgd clients to start                         ip
  // 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.
  // DEPRECATED. Each solver accesses a different subset of the database.
  optional uint32 rand_skip = 7 [default = 0];
  optional DB backend = 8 [default = LEVELDB];
  // DEPRECATED. See TransformationParameter. For da
  // 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. Speci                randomly
  // crop an image.
  optional uint32 crop_size = 5 [default = 0];
  // DEPRECATED. See TransformationParameter. Speci                ly mirror
  // data.
  optional bool mirror = 6 [default = false];
  // Force the encoded image to have 3 color channel
  optional bool force_encoded_color = 9 [default = false];
  // Prefetch queue (Increase if data feeding bandwidth varies, within the
  // limit of device memory for GPU training)
  optional uint32 prefetch = 10 [default = 4];
}
```

data source

batch size

backend of database

mean file

crop size

mirror

transform_param

Data Layer

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 28
    mean_file: "/home/../data/cifar10/cifar10_mean.binaryproto"
  }
  data_param {
    source: "/home/../data/cifar10/cifar10_train_lmdb"
    batch_size: 128
    backend: LMDB
  }
}
```

common parameters: Layer name, layer name, top layers

this layer only used for training

data preprocessing

data path, batch size and file format

Convolutional Layer

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
   num_output: 96     # learn 96 filters
   kernel_size: 11    # each filter is 11x11
   stride: 4          # step 4 pixels between each filter application
   weight_filler {
    type: "gaussian" # initialize the filters from a Gaussian
    std: 0.01        # distribution with stdev 0.01 (default mean: 0)}
   bias_filler {
    type: "constant" # initialize the biases to zero (0)
    value: 0     }    }  }
```

More Layers

➢ Batch normalization

```
layer {
  name: "batchNorm_1"
  type: "BatchNorm"
  bottom: "conv_1"
  top: "conv_1"
  batch_norm_param {
  }
}
```

➢ Scale layer

```
layer {
  name: "scale_1"
  type: "Scale"
  bottom: "conv_1"
  top: "conv_1"
}
```

➢ ReLU

```
layer {
  name: "relu_1"
  type: "ReLU"
  bottom: "conv_1"
  top: "conv_1"
}
```

➢ Eltwise

```
layer {
  name: "elem1"
  type: "Eltwise"
  bottom: "conv_3"
  bottom: "conv_1"
  top: "elem1"
  eltwise_param {
    operation: SUM
  }
}
```

More Layers

➢ Pooling

```
layer {
  name: "global_pooling"
  type: "Pooling"
  bottom: "relu_19"
  top: "pooling_1"
  pooling_param {
    pool: AVE
    global_pooling: true
  }
}
```

➢ Softmax with loss

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc10"
  bottom: "label"
  top: "loss"
}
```

➢ Inner product

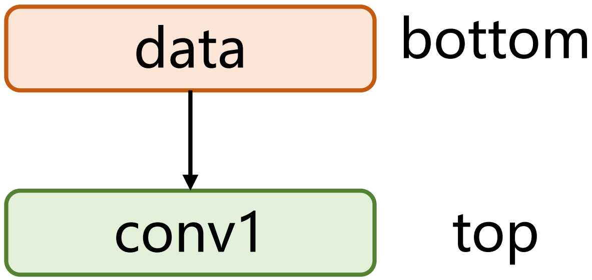
```
layer {
  name: "fc10"
  type: "InnerProduct"
  bottom: "pooling_1"
  top: "fc10"
  param {
    lr_mult: 1.0
    decay_mult: 1.0
  }
  param {
    lr_mult: 1.0
    decay_mult: 1.0
  }
  inner_product_param {
    num_output: 10
    bias_filler {
      type: "constant"
      value: 0.0
    } } }
```

Implementation of Layer

- LayerSetUp(bottom, top)
  - run once for initialization
- Reshape(bottom, top)
  - check dimensions of blobs
- Forward_cpu(bottom, top), Forward_gpu(bottom, top)
  - execute forward pass on CPU/GPU
- Backward_cpu(top, propagate_down, bottom)
  - execute backward pass on CPU
- Backward_gpu(top, propagate_down, bottom)
  - execute backward pass on GPU

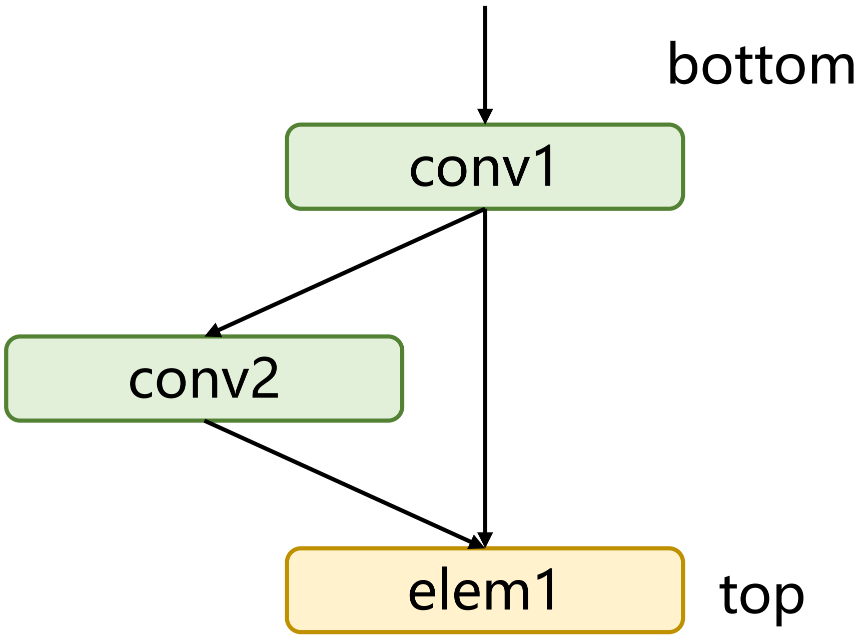
Define Net

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {...}
  transform_param {...}
  data_param {...}
}
layer {
   name: "conv1"
   type: "Convolution"
   bottom: "data"
   top: "conv1"
   param { ...}
   param { ...}
   convolution_param {...}  }
```

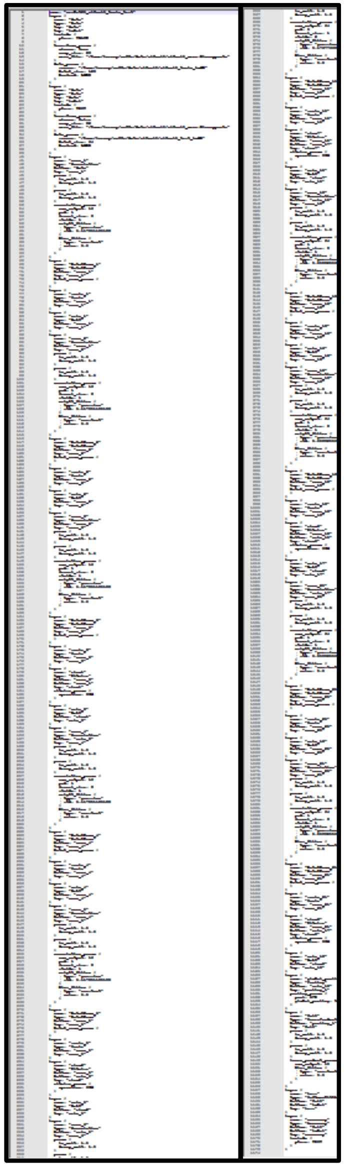


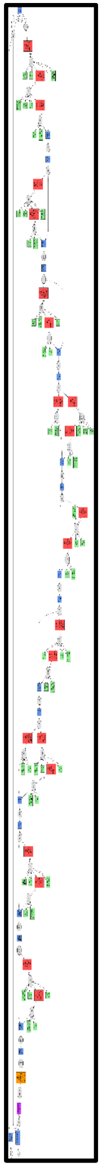
Define Net

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param { ...}
  param { ...}
  convolution_param {...}  }
layer {
  name: "conv2"
  type: "Convolution"
  bottom:   "conv1"
  top: "conv2"
  param { ...}
  param { ...}
  convolution_param {...}  }
layer {
  name: "elem1"
  type: "Eltwise"
  bottom: "conv_2"
  bottom: "conv_1"
  top: "elem1"
  eltwise_param {
    operation: SUM
  }
}
```



What About ResNet-20?





1174 lines!!

use **python interface** to write the file automatically

Upgrade Prototxt

```
layers {
  top: "data"
  top: "label"
  name: "data"
  type: DATA
  data_param {
    source: "/home/linmin/IMAGENET-LMDB/imagenet-train-lmdb"
    backend: LMDB
    batch_size: 64
  }
  transform_param {
    crop_size: 224
    mirror: true
    mean_file: "/home/linmin/IMAGENET-LMDB/imagenet-train-mean
  }
  include: { phase: TRAIN }
}
```

old version of .prototxt file is different from the version we use.

- run command to upgrade the .prototxt file:

upgrade_net_proto_text.exe /path/to/origin /path/to/new

Solver

- http://caffe.berkeleyvision.org/tutorial/solver.html
- The Caffe solvers are:
  - Stochastic Gradient Descent (type: "**SGD**")
  - AdaDelta (type: "**AdaDelta**")
  - Adaptive Gradient (type: "**AdaGrad**")
  - Adam (type: "**Adam**")
  - Nesterov’s Accelerated Gradient (type: "**Nesterov**")
  - RMSprop (type: "**RMSProp**")

Solver

➢ The solver

- scaffolds the optimization bookkeeping and **creates the training network** for learning and test network for evaluation.
- **iteratively optimizes** by calling forward / backward and updating parameters
- (periodically) **evaluates** the test networks
- **snapshots** the model and solver state

Solver

net | test_net | train_net

test_iter | test_interval | max_iter

base_lr | momentum | weight_decay

lr_policy | gamma | power

display | snapshot | snapshot_prefix | solver_mode

Example

```
# 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
test_interval: 500  # Carry out testing every 500 training iterations.
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
lr_policy: "inv"  # The learning rate policy
gamma: 0.0001
power: 0.75
display: 100  # Display every 100 iterations
max_iter: 10000  # The maximum number of iterations
snapshot: 5000  # snapshot intermediate results
snapshot_prefix: "examples/mnist/lenet"
solver_mode: GPU  # solver mode: CPU or GPU
```

write this file automatically by using **python interface**

Solver

➢ get more details from caffe.proto

```
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.

```

Q & A?

Develop New Layers

Develop New Layers

Declare new layer in head file

↓

Implement new layer in Cpp file

↓

Register new layer in Cpp file

↓

Declare parameters in caffe.proto

↓

Write test file to check your implementations

Develop New Layers

- **Step 1:** add a class declaration for your layer to **head file**
- **Step 2:** implement your layer in **c++ file**
- **Step 3:** instantiate and **register your layer** using:

  INSTANTIATE_CLASS(MyAwesomeLayer);

  REGISTER_LAYER_CLASS(MyAwesome);
- **Step 4:** declare parameters in **caffe.proto**, using (and then incrementing) the "next avaliable" layer-specific ID" declared in a comment above message LayerParameter
- **Step 5:** write **test file** to check your Forward and Backward implementations

Write Head File

Step 1: declare your class in ./include/caffe/layers/xx.hpp

```
#ifndef CAFFE_SCALE_LAYER_HPP_
#define CAFFE_SCALE_LAYER_HPP_
#include <vector>
#include "caffe/blob.hpp “
#include "caffe/layer.hpp “
#include "caffe/proto/caffe.pb.h “
#include "caffe/layers/bias_layer.hpp “
```

Write Head File

```
namespace caffe {

template <typename Dtype>
class ScaleLayer: public Layer<Dtype> {
 public:
  explicit ScaleLayer(const LayerParameter& param)
      : Layer<Dtype>(param) {}
  virtual void LayerSetUp(const vector<Blob<Dtype>*>& bottom,
                                        const vector<Blob<Dtype>*>& top);
  virtual void Reshape(const vector<Blob<Dtype>*>& bottom,
                                  const vector<Blob<Dtype>*>& top);
  virtual inline const char* type() const { return "Scale"; }

  // Scale
  virtual inline int MinBottomBlobs() const { return 1; }
  virtual inline int MaxBottomBlobs() const { return 2; }
  virtual inline int ExactNumTopBlobs() const { return 1; }
```

Write Head File

```
protected:
  virtual void Forward_cpu(const vector<Blob<Dtype>*>& bottom,
                           const vector<Blob<Dtype>*>& top);

  virtual void Forward_gpu(const vector<Blob<Dtype>*>& bottom,
                           const vector<Blob<Dtype>*>& top);

  virtual void Backward_cpu(const vector<Blob<Dtype>*>& top,
                            const vector<bool>& propagate_down,
                            const vector<Blob<Dtype>*>& bottom);

  virtual void Backward_gpu(const vector<Blob<Dtype>*>& top,
                            const vector<bool>& propagate_down,
                            const vector<Blob<Dtype>*>& bottom);
```

Write Head File

```
    shared_ptr<Layer<Dtype> > bias_layer_;
    vector<Blob<Dtype>*> bias_bottom_vec_;
    vector<bool> bias_propagate_down_;
    int bias_param_id_;
    Blob<Dtype> sum_multiplier_;
    Blob<Dtype> sum_result_;
    Blob<Dtype> temp_;  int axis_;
    int outer_dim_, scale_dim_, inner_dim_;
  };
}

// namespace caffe

#endif
// CAFFE_SCALE_LAYER_HPP_
```

Write C++ File

Step 2: implement your layer in ./src/caffe/layers/xx.cpp

```
#include <algorithm>
#include <vector>

#include "caffe/filler.hpp"
#include "caffe/layer_factory.hpp"
#include "caffe/layers/scale_layer.hpp"
#include "caffe/util/math_functions.hpp “
```

Write C++ File

```
namespace caffe {

template <typename Dtype>
void ScaleLayer<Dtype>::LayerSetUp(...) {
  const ScaleParameter& param = this->layer_param_.scale_param();
  if (bottom.size() == 1 && this->blobs_.size() > 0) {
    LOG(INFO) << "Skipping parameter initialization";
  } else if (bottom.size() == 1) {
    ...
    if (!param.has_filler()) {
     // Default to unit (1) filler for identity operation.
     filler_param.set_type("constant");
     filler_param.set_value(1);}
    shared_ptr<Filler<Dtype> > filler(GetFiller<Dtype>(filler_param));
    filler->Fill(this->blobs_[0].get()); }
    ...
}
```

Write C++ File

```
template <typename Dtype>
void ScaleLayer<Dtype>::Reshape(...) {
  const ScaleParameter& param = this->layer_param_.scale_param();
  ...
  outer_dim_ = bottom[0]->count(0, axis_);
  scale_dim_ = scale->count();
  inner_dim_ = bottom[0]->count(axis_ + scale->num_axes());
  if (bottom[0] == top[0]) {  // in-place computation
    const bool scale_param = (bottom.size() == 1);
    if (!scale_param || (scale_param
        && this->param_propagate_down_[0])) {
      temp_.ReshapeLike(*bottom[0]);
    }
  } else {
    top[0]->ReshapeLike(*bottom[0]);
  }
  ...
}
```

Write C++ File

```
template <typename Dtype>
void ScaleLayer<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();
  for (int n = 0; n < outer_dim_; ++n) {
    for (int d = 0; d < scale_dim_; ++d) {
      const Dtype factor = scale_data[d];
      caffe_cpu_scale(inner_dim_, factor, bottom_data, top_data);
      bottom_data += inner_dim_;
      top_data += inner_dim_;
    }
  }...
}
```

Write C++ File

```
template <typename Dtype>
void ScaleLayer<Dtype>::Backward_cpu(
    const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (bias_layer_ && this->param_propagate_down_[this->
param_propagate_down_.size() - 1]) {
  // compute gradient of bias
  ... }
  ...
  if ((!scale_param && propagate_down[1]) ||
      (scale_param && this->param_propagate_down_[0])) {
   // compute gradient of scale parameters
    ...}
  if (propagate_down[0]) {
   // compute gradient of the bottom layer
    ... } }
```

Register New Layer

Step 3: register your layer at the end of the C++ file

```
#ifdef CPU_ONLY
STUB_GPU(ScaleLayer);
#endif

INSTANTIATE_CLASS(ScaleLayer);
REGISTER_LAYER_CLASS(Scale);

}  // namespace caffe
```

Modify caffe.proto

Step 4: declare your class in ./src/caffe/proto/caffe.proto

```
// Update the next available ID when you add a new LayerParameter field.
// next available layer-specific ID: 142 (last added: scale_param)
message LayerParameter {
  ...
  optional ReshapeParameter reshape_param = 133;
  optional ScaleParameter scale_param = 142;
  optional SigmoidParameter sigmoid_param = 124;
  ...
}
...
message ScaleParameter {
  ...
  optional bool bias_term = 4 [default = false];
  optional FillerParameter bias_filler = 5;
...
}
```

Write Test File

Step 5: write the test code in ./src/caffe/test/test_xx.cpp

```
#include <algorithm>
#include <vector>
#include "gtest/gtest.h"

#include "caffe/blob.hpp"
#include "caffe/common.hpp"
#include "caffe/filler.hpp"
#include "caffe/layers/scale_layer.hpp"
#include "caffe/test/test_caffe_main.hpp"
#include "caffe/test/test_gradient_check_util.hpp"
```

Write Test File

```
namespace caffe {
template <typename TypeParam>
class ScaleLayerTest : public MultiDeviceTest<TypeParam> {
  typedef typename TypeParam::Dtype Dtype;
 protected:
  ScaleLayerTest(): …{
    …
    UniformFiller<Dtype> filler(filler_param);
    filler.Fill(this->blob_bottom_);
    …  }
  virtual ~ScaleLayerTest() {
    delete blob_bottom_;
    …  }
  Blob<Dtype>* const blob_bottom_;
 …
  Blob<Dtype>* const blob_top_;
… };
```

Write Test File

```
TYPED_TEST(ScaleLayerTest, TestForwardEltwiseInPlace) {
  typedef typename TypeParam::Dtype Dtype;
  this->blob_top_vec_[0] = this->blob_bottom_;// in-place computation
  Blob<Dtype> orig_bottom(this->blob_bottom_->shape());
  orig_bottom.CopyFrom(*this->blob_bottom_);
  this->blob_bottom_vec_.push_back(this->blob_bottom_eltwise_);
  LayerParameter layer_param;
  layer_param.mutable_scale_param()->set_axis(0);
  ...
  layer->SetUp(this->blob_bottom_vec_, this->blob_top_vec_);
  layer->Forward(this->blob_bottom_vec_, this->blob_top_vec_);
  const Dtype* data = this->blob_bottom_->cpu_data();
  ...
  for (int i = 0; i < count; ++i) {
    EXPECT_NEAR(data[i], in_data_a[i] * in_data_b[i], 1e-5);
  } }
...
```

Q & A?

Python Interface

Preparation

➢ Jupyter Notebook

- The Jupyter Notebook is an open-source **web application** that allows you to create and share documents that contain live code, equations, visualizations and explanatory text

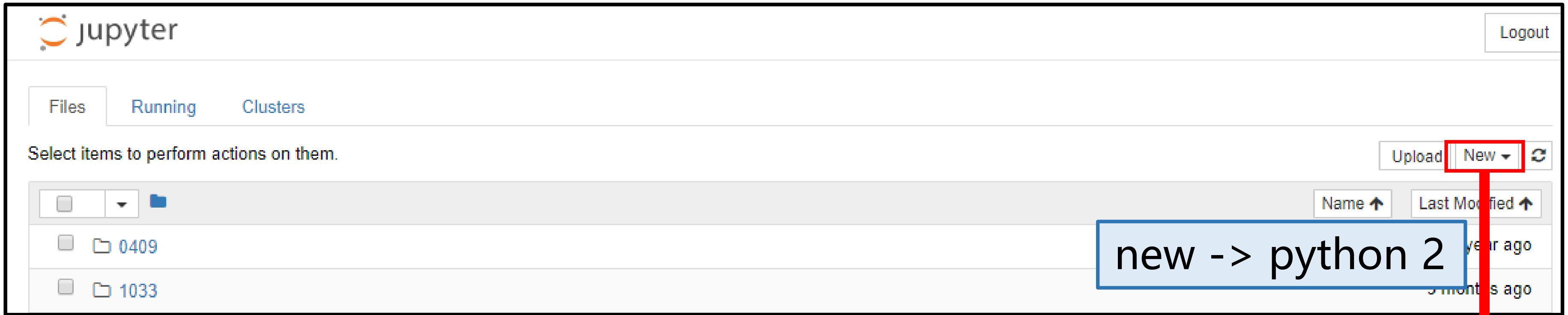
➢ Installation

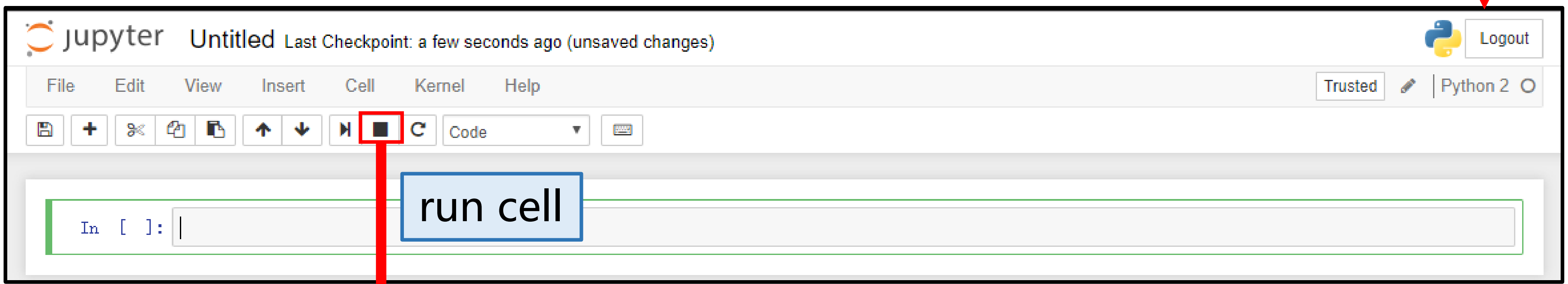
```
pip install jupyter
```

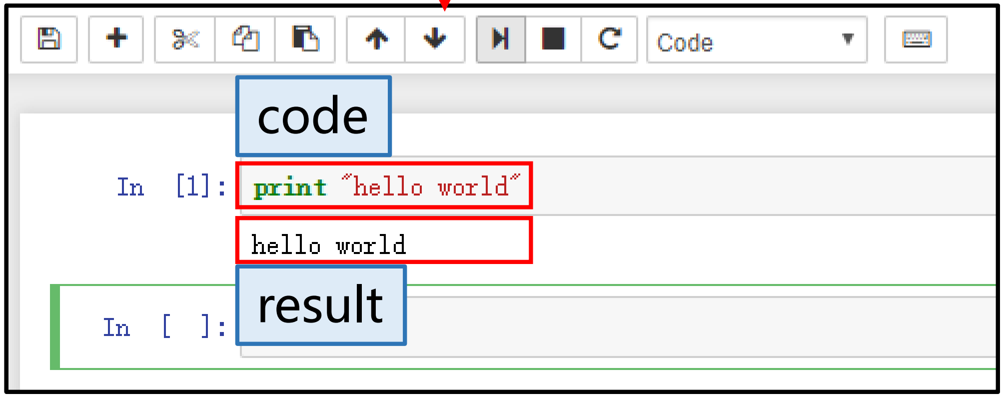
➢ Start

```
jupyter notebook
```

Jupyter Notebook







Learning from the Notebook Examples

➢ Go to http://caffe.berkeleyvision.org/

jupyter nbviewer

JUPYTER FAQ

Classification: Instant Recognition with Caffe

In this example we'll classify an image with the bundled CaffeNet model (which is based on the network architecture of Krizhevsky et al. for ImageNet).

We'll compare CPU and GPU modes and then dig into the model to inspect features and the output.

1. Setup

- First, set up Python, `numpy`, and `matplotlib`.

In [1]:
```
# set up Python environment: numpy for numerical routines, and matplotlib for plotting
import numpy as np
import matplotlib.pyplot as plt
# display plots in this notebook
%matplotlib inline

# set display defaults
plt.rcParams['figure.figsize'] = (10, 10)        # large images
plt.rcParams['image.interpolation'] = 'nearest'  # don't interpolate: show square pixels
plt.rcParams['image.cmap'] = 'gray'  # use grayscale output rather than a (potentially misleading) color heatmap
```

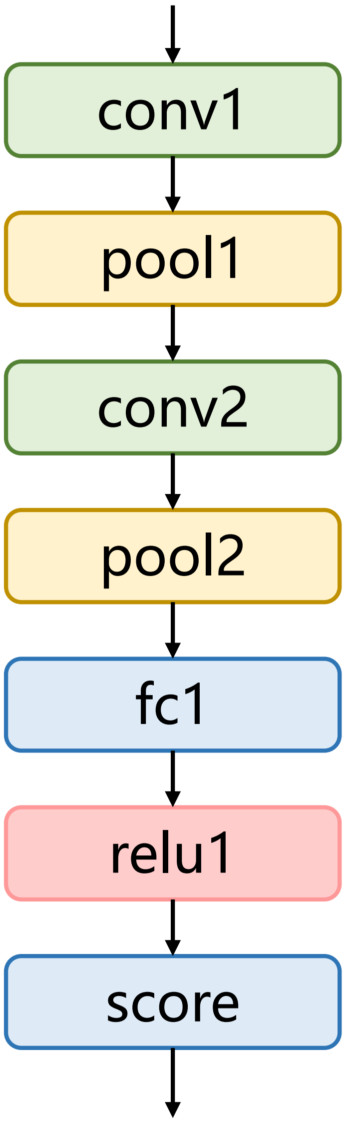
- Load `caffe`.

In [2]:
```
# The caffe module needs to be on the Python path;
#  we'll add it here explicitly.
import sys
caffe_root = '../'  # this file should be run from {caffe_root}/examples (otherwise change this line)
sys.path.insert(0, caffe_root + 'python')

import caffe
# If you get "No module named _caffe", either you have not built pycaffe or you have the wrong path.
```

Notebook Examples

- Image Classification and Filter V
  Instant recognition with a pre-tr
  features and parameters layer-
- Learning LeNet
  Define, train, and test the classi
- Fine-tuning for Style Recognitio
  Fine-tune the ImageNet-trained
- Off-the-shelf SGD for classificat
  Use Caffe as a generic SGD opti
- Multilabel Classification with Py
  Multilabel classification on PAS
- Editing model parameters
  How to do net surgery and man
- R-CNN detection
  Run a pretrained model as a detector in Python.
- Siamese network embedding
  Extracting features and plotting the Siamese network embedding.

Create Network



Create Network

```
from caffe import layers as L, params as P
def lenet(lmdb, batch_size):
    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)
```

Create Network

```
n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50,
                        weight_filler=dict(type='xavier'))
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()
```

Create Network

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

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

Define Solver

```
from caffe.proto import caffe_pb2
s = caffe_pb2.SolverParameter()

# Set a seed for reproducible experiments:
# this controls for randomization in training.
s.random_seed = 0xCAFFE
# Specify locations of the train and (maybe) test networks.
s.train_net = train_net_path
s.test_net.append(test_net_path)
s.test_interval = 500  # Test after every 500 training iterations.
s.test_iter.append(100) # Test on 100 batches each time we test.
s.max_iter = 10000      # no. of times to update the net (training iterations)
```

Define Solver

```
# EDIT HERE to try different solvers
# solver types include "SGD", "Adam", and "Nesterov" among others.
s.type = "SGD"

# Set the initial learning rate for SGD.
s.base_lr = 0.01  # EDIT HERE to try different learning rates
# Set momentum to accelerate learning by
# taking weighted average of current and previous updates.
s.momentum = 0.9
# Set weight decay to regularize and prevent overfitting
s.weight_decay = 5e-4
```

Define Solver

```
# Set `lr_policy` to define how the learning rate changes during training.
s.lr_policy = 'inv'
s.gamma = 0.0001
s.power = 0.75
# Display the current training loss and accuracy every 1000 iterations.
s.display = 1000
# Snapshots are files used to store networks we've trained.
# We'll snapshot every 5K iterations -- twice during training.
s.snapshot = 5000
s.snapshot_prefix = 'mnist/custom_net'
# Train on the GPU
s.solver_mode = caffe_pb2.SolverParameter.GPU
```

Define Solver

```python
solver_config_path = 'mnist/lenet_auto_solver.prototxt'
# Write the solver to a temporary file and return its filename.
with open(solver_config_path, 'w') as f:
    f.write(str(s))
```

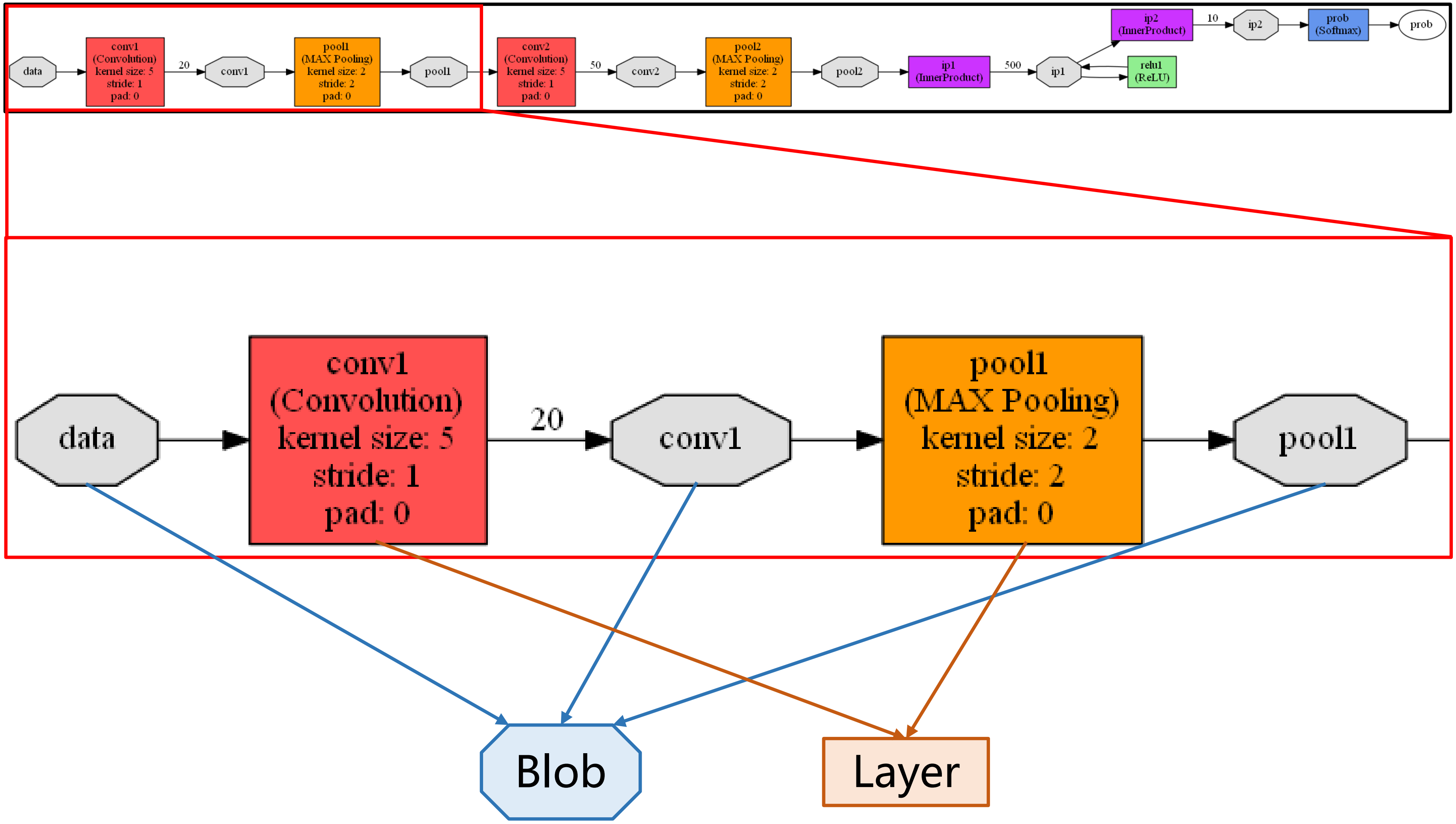
Training and Testing

```
### load the solver and create train and test nets
solver = caffe.get_solver(solver_config_path)


### solve
niter = 250  # EDIT HERE increase to train for longer
test_interval = niter / 10
# losses will also be stored in the log
train_loss = zeros(niter)
test_acc = zeros(int(np.ceil(niter / test_interval)))
```

Training and Testing

```
# the main solver loop
for it in range(niter):
    solver.step(1)  # SGD by Caffe, forward, backward and update parameters
   # store the train loss
    train_loss[it] = solver.net.blobs['loss'].data

   # run a full test every so often
   # (Caffe can also do this for us and write to a log, but we show here
   #  how to do it directly in Python, where more complicated things are easier.)
    if it % test_interval == 0:
        print 'Iteration', it, 'testing...'
        correct = 0
        for test_it in range(100):
            solver.test_nets[0].forward()
            correct += sum(solver.test_nets[0].blobs['score'].data.argmax(1)
                           == solver.test_nets[0].blobs['label'].data)
        test_acc[it // test_interval] = correct / 1e4
```

Visualization



Visualization

➢ **def** draw_net_to_file(...)

Draws a caffe net, and saves it to file using the format given as the file extension.

- **caffe_net**: caffe.proto.caffe_pb2.NetParameter protocol buffer.
- **filename**(string): The path to a file where the networks visualization will be stored.
- **rankdir** : {'LR', 'TB', 'BT'}. Direction of graph layout.
- **phase**: {caffe_pb2.Phase.TRAIN, caffe_pb2.Phase.TEST, None}. Include layers from this network phase. If None, include all layers. Default is None

Visualization

```
from google.protobuf import text_format
import caffe.draw
from caffe.proto import caffe_pb2

# Set input path of .prototxt
file_name = 'lenet5'
input_file = 'f:/'+file_name+'.prototxt  ’

# Set output image(.png) path
output_file = 'f:/'+file_name+'.png'

net = caffe_pb2.NetParameter()
text_format.Merge(open(input_file).read(), net)
print net

caffe.draw.draw_net_to_file(net, output_file)
print ('Drawing network done!')
```

Q & A?

Thank You