Outline

- **BVLC**: Berkeley Vision and Learning Center
- **Caffe**: Convolutional Architecture for Fast Feature Embedding
- What can Caffe do?
- Installation
- Tutorial
- Conclusion
Caffe

- Convolutional Architecture for Fast Feature Embedding
- [http://caffe.berkeleyvision.org/](http://caffe.berkeleyvision.org/)
Notes

- CNN/DNN
- Different training objective function
- Different optimization algorithm
- Program control
- Model Zoo
- C++ Framework
- **NO LSTM/RNN**
CNN/DNN modules

• Vision Layer
  ○ Convolution/ Pooling/ Local Response Normalization

• Common Layer
  ○ InnerProduct( = DNN fully-connected weights)
  ○ batch normalization
  ○ element-wise summation/product/BNLL
  ○ dropout layer

• Activation Layer (Non-linearity)
  ○ Sigmoid/Tanh/ReLU/PReLU

• Utility Layer
  ○ Dimension slicing/concatenation/flattening/reshaping
Training Loss Layer

- CrossEntropyLoss
- L1, L2 Loss, pair-wise contrasive loss
- Multitask Learning with loss weights
- Accuracy Layer: for evaluation only.
Optimization Algorithms

- SGD, RMSProp, ADAM, ADADELTA, ADGRAD…
- Momentum
- Learning Rate Adjustment Policies
  - decay, step-decay, exp-decay
- Regularization
  - weight-decay, L1 decay
Program Control

● Snapshot (solverstate)
● Phase:
  ○ Convention: Train/Validation/Test
  ○ Caffe: Train/Test/Deploy
  ○ You could assign different action w.r.t different phase.

● Caffe Program Interface
  ○ You can provide meta data without actually implement the deep learning algorithms.
  ○ You can extend the module and implement your own ideas.
What can Caffe do?

- Multitask learning
  - Multi-target, Multi-loss
- Parameters share training
  - Siamese Neural Network
- Easy to integrated into online system.
  - With known distributed database, protocol…
  - C++, Python and Matlab binding.
Multitask Learning
Siamese Neural Network
Introduction

● The goal of Caffe is to find the effective representations (feature embedding) for various inputs, such as images and sounds, with help of deep learning and GPU acceleration.
  ○ There does exist cross-domain feature embedding among different tasks.
  ○ Utilize CUDA (cuDNN) to achieve acceptable training time.
Introduction

- Caffe is designed for images and based on state-of-the-art CNN. However, the concept of feature embedding shares among other works (e.g. speech recognition).
  - Yes, Caffe supports non-image tasks with a bit more efforts.
Introduction

- Caffe provided **well-known and well-trained models**, offering state-of-the-art researching and off-the-shelf deployment.
  - **ImageNet**: classify images into 22000 categories.
  - **GoogleNet**: classify images into 1000 categories.
  - **R-CNN**: object detection (20 or 200 types)
ImageNet
GoogleNet
R-CNN: Regions with CNN features

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions
Highlights

● Complete toolkit for training, testing, fine-tuning and deploying.

● Modularity
  ○ Extensible
  ○ Forward, backward, CPU/GPU version.

● Good coding style and huge community
  ○ Only well-test idea would be merged into Caffe
  ○ Distributed developed with many coders.
  ○ Clearly logging, documentation, robust, bullet proof, easy-understanding message...
Highlights

- Python/Matlab binding
  - Online deploying interface
  - Online training is not intuitively integrated but able to.
- Pre-trained models
Architecture

● C++ implementation
  ○ Well-known efficiency.

● Saving models in GPBL.
  ○ Google Protocol Buffer Language
  ○ Human-readable, efficient serialization and implemented in multiple interface.

● Online training
  ○ Memory data.

● Offline training
  ○ LevelDB database for image data
  ○ HDF5 database for general purpose.
Application

• Object Classification/Detection
  ○ ImageNet
  ○ Demo
Application

- Learning Feature Embedding
  - ImageNet
  - Using pre-trained models as feature extractor
Figure 3: Features extracted from a deep network, visualized in a 2-dimensional space. Note the clear separation between categories, indicative of a successful embedding.
<table>
<thead>
<tr>
<th>Ethereal</th>
<th>HDR</th>
<th>Melancholy</th>
<th>Minimal</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
</tbody>
</table>

**Figure 4:** Top three most-confident positive predictions on the Flickr Style dataset, using a Caffe-trained classifier.
Tutorial

- Installation
  - Prerequisite/Core/Wrappers
- Data Preprocessing
  - LevelDB/HDF5
- Models
  - description, model weights, protobuf
- Solver
  - description, solver state
- Training/Testing/Fine-tuning/Deploying
Warning

- Caffe is not officially supporting Windows OS. Ubuntu/CentOS is recommended.
- Caffe is not officially supporting Windows OS. Ubuntu/CentOS is recommended.
- Caffe is not officially supporting Windows OS. Ubuntu/CentOS is recommended.
- 不要問我windows怎麼灌。
Installation

- Install Prerequisite
  - CUDA and cuDNN
  - BLAS via OpenBLAS, MKL, or ATLAS
  - `sudo apt-get install` Boost/OpenCV/protobuf/glog/gflags/hdf5/leveldb/snappy/lmdb

- Install Caffe
  - prepare `Makefile.config` from `Makefile.config.example`
  - `make all && make test && make runtest`

- Install Python wrapper (optional but recommended)
  - `for req in $(cat requirements.txt); do pip install $req; done`
  - `export PYTHONPATH=/path/to/caffe/python:$PYTHONPATH`
Data Preprocessing

● Input data must be 4D array:
  ○ Image: (number, channel, height, width)
  ○ Non-image: (number, dimension, 1, 1)

● Training target is usually 2D array:
  ○ Label: (number, dimension)

● Online Memory
  ○ (C++) MemoryDataLayer::Reset()
  ○ (python) Net.set_input_arrays()

● Offline database
  ○ prepare a directory contain all the images
  ○ prepare `lmdb`(python) or `leveldb`(c++) for images
  ○ prepare `hdf5`(python) for general purposes
  ○ prepare train.list/test.list comprising the path
Models

● Description
  ○ DAG layered structure written in json format.
  ○ Data Layers: read from data, only out-degree
  ○ Activation/Neuron Layers: perform forward/backward pass.
  ○ Loss Layers: nn output, only in-degree
  ○ Common Layers: for utility
  ○ Each type of layers contain its own parameters
  ○ Different layer parameter could share!
Models

```yaml
classification_model {
  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: 32
      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
    }
  }
}
```
Models

layers {
  name: "fc8"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 1000
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
  bottom: "fc7"
  top: "fc8"
}

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
}
Models

```json
layers {
    name: "slicer_label"
    type: SLICE
    bottom: "label"
    ## Example of label with a shape N x 3 x 1 x 1
    top: "label1"
    top: "label2"
    top: "label3"
    slice_param {
        slice_dim: 1
        slice_point: 1
        slice_point: 2
    }
}
```
Models

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    name: "conv1_w"
    lr_mult: 1
  }
  param {
    name: "conv1_b"
    lr_mult: 2
  }
}

layer {
  name: "conv1_p"
  type: "Convolution"
  bottom: "data_p"
  top: "conv1_p"
  param {
    name: "conv1_w"
    lr_mult: 1
  }
  param {
    name: "conv1_b"
    lr_mult: 2
  }
}
Models

• Model Weights
  ○ x.caffemodel
  ○ store in GPBL format
  ○ prototype

```plaintext
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 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

# The base learning rate, momentum and the weight decay of the network.
basis_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
Training and Testing

● Preparation:
  ○ data
  ○ model description (nnet.prototxt)
  ○ solver description (solver.prototxt)

● You can specify two phase
  ○ training -> calculate loss, gradients, backward pass and update
  ○ testing -> calculate accuracy/loss

● run:
  ○ caffe train --solver=solver.prototxt
Fine-tuning

● Preparation:
  ○ data
  ○ model description(nnet.prototxt)
  ○ solver description(solver.prototxt)
  ○ pre-trained models(pretrain.caffemodel)

● run:
  ○ caffe train --solver=solver.prototxt --weights=pretrain.caffemodel
Deploying

• Preparation:
  ○ data
  ○ model description (deploy.prototxt)
  ○ well-train model (well_train.caffemodel)
  ○ pycaffe if you use python
  ○ your own code (python, c++ or matlab)

• deploy.prototxt is slightly different
Deploying (python example)

- Add data description in `deploy.prototxt`
  - remove any DATA_LAYER
- In python, import caffe
  - `net = caffe.Classifier(MODEL_FILE, PRETRAINED)`
  - use numpy array to prepare your input data
  - `net.blobs['data'].reshape(input_shape)`
  - `out = net.forward( data=input )`
  - use `out['label']` to get any output you want.

```
name: "LeNet"
input: "data"
input_dim: 64
input_dim: 1
input_dim: 28
input_dim: 28
```
Final Recommendation

- Caffe is easy and flexible to use, but not that efficient. 甚至可以不用寫程式XD
- For complicated structure with multi-loss layer, weight sharing and advanced optimization, caffe is good.
- However, you should prepare data in the specified format
  - HDF5, LMDB, LEVELDB...
  - offline training/testing is easy and preferred
- For online procedure, you must write your own code to deploy.