CS 231A Section: Computer Vision Libraries Overview

Amir Sadeghian
Feb 2018
Overview

- **OpenCV**
- Deep Learning Frameworks
  - **Caffe**
  - Torch
  - Tensorflow
Other CV libraries

• **Vlfeat**: An Open source library with popular computer vision algorithms specializing in image understanding and local features extraction and matching.

• **scikit-learn**: An open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms.

• **PCL**: A standalone, large scale, open project for 2D/3D image and point cloud processing.

• **SLAM frameworks (bundler, visualsfm, meshlab)**: Applications for 3D reconstruction using structure from motion (SFM).

• **Libraries for specific tasks**: e.g. tracking libraries, Detection libraries ...
OpenCV
Introduction to OpenCV

- Open source computer vision and machine learning library
- Contains implementations of a large number of vision algorithms
- Written natively in C++, also has C, Python, Java, and MATLAB interfaces
- Supports Windows, Linux, Mac OS X, Android, and iOS
Installation

• Download from http://opencv.org and compile from source
• Windows: Run executable downloaded from OpenCV website
• Mac OS X: Install through MacPorts, easy_install, ...
• Linux: Install through the package manager (e.g. yum, apt) but make sure the version is sufficiently up-to-date for your needs
Basic OpenCV Structures

- **Point, Point2f** - 2D Point
- **Size** - 2D size structure
- **Rect** - 2D rectangle object
- **RotatedRect** - Rect object with angle
- **Mat** - image object
- ...
Point

- 2D Point Object
  - int x, y;

- Sample Functions
  - Point.dot(<Point>) - computes dot product
  - Point.inside(<Rect>) - returns true if point is inside

Math operators, you may use
- Point operator +
- Point operator +=
- Point operator -
- Point operator -=
- Point operator *
- Point operator *=
- bool operator ==
- bool operator !=
- double norm
Size

• 2D Size Structure
  - int width, height;

• Functions
  - Size.area() - returns (width * height)

Rect

• 2D Rectangle Structure
  - int x, y, width, height;

• Functions
  - Rect.tl() - return top left point
  - Rect.br() - return bottom right point
cv::Mat

• The primary data structure in OpenCV is the Mat object. It stores images and their components.

• Main items
  
  • rows, cols - length and width(int)
  
  • channels - 1: grayscale, 3: BGR
  
  • depth: CV_<bit_depth>C<#chan>
  
• See the manuals for more information
cv::Mat- Functions

- Mat.at<datatype>(row, col) - returns pointer to a location in the image
- Mat.channels() - returns the number of channels
- Mat.clone() - returns a deep copy of the image
- Mat.create(rows, cols, TYPE) - re-allocates new memory to matrix
- Mat.cross(<Mat>) - computes cross product of two matrices
- Mat.depth() - returns bit-depth of a matrix
- Mat.dot(<Mat>) - computes the dot product of two matrices
Pixeltypes

- PixelTypes shows how the image is represented in data
  - BGR - The default color of imread(). Normal 3 channel color
  - HSV - Hue is color, Saturation is amount, Value is lightness. 3 channels
  - GRAYSCALE - Gray values, Single channel
- OpenCV requires that images be in BGR or Grayscale in order to be shown or saved. Otherwise, undesirable effects may appear.
OpenCV Functions
Image Normalization and Thresholding

• Normalization remaps a range of pixel values to another range of pixel values
  • `void normalize(InputArray src, OutputArray dst,...)`

• OpenCV provides a general purpose method for thresholding an image
  • `double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)`
  • Specify thresholding scheme specified by the type variable
Image Smoothing

• Reduces the sharpness of edges and smooths out details in an image

• OpenCV implements several of the most commonly used methods
  • void GaussianBlur(InputArray src, OutputArray dst,...)
  • void medianBlur(InputArray src, OutputArray dst,...)

• Other functions include generic convolution, separable convolution, dilate, and erode.
Image Smoothing: Code

```c
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

int main(int argc, char** argv) {
    // Read in colored image
    cv::Mat image = cv::imread(argv[1]);
    cv::imwrite("photo.jpg", image);
    // Apply Gaussian blur
    cv::Mat image_gaussian_blur;
    image.convertTo(image_gaussian_blur, CV_8UC3);
    cv::GaussianBlur(image_gaussian_blur, image_gaussian_blur, cv::Size(0, 0), 9);
    cv::imwrite("photo_gaussian_blur.jpg", image_gaussian_blur);
    // Apply median blur
    cv::Mat image_median_blur;
    image.convertTo(image_median_blur, CV_8UC3);
    cv::medianBlur(image_median_blur, image_median_blur, 17);
    cv::imwrite("photo_median_blur.jpg", image_median_blur);
}
```
Image Smoothing: Sample Image

Original  Gaussian Blur  Median Blur
Edge Detection

• OpenCV implements a number of operators to help detect edges in an image
  • Sobel Operator
    • void cv::Sobel(image in, image out, CV_DEPTH, dx, dy);
  • Scharr Operator
    • void cv::Scharr(image in, image out, CV_DEPTH, dx, dy);
  • Laplacian Operator
    • void cv::Laplacian(image in, image out, CV_DEPTH);

• OpenCV also implements multi-stage edge detection algorithms such as Canny edge detection
• Tip: If your image is noisy, then edge detection will often exaggerate the noise
• Sometimes smoothing the image before running edge detection gives better results
Edge Detection: Code

```cpp
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>

int main(int argc, char** argv)
{
    // Read image as grayscale, delete zero to read in color
    cv::Mat image=cv::imread(argv[1],0);
    cv::imwrite("photo_gray.jpg",image);

    // Calculate x-gradient using Sobel operator
    cv::Mat image_gradient_x;
    image.convertTo(image_gradient_x,CV_32FC1);
    cv::Sobel(image_gradient_x,image_gradient_x,CV_32FC1,0,1);
    // Absolute value and normalize
    cv::convertScaleAbs(image_gradient_x,image_gradient_x);

    // Calculate y-gradient using Sobel operator
    cv::Mat image_gradient_y;
    image.convertTo(image_gradient_y,CV_32FC1);
    cv::Sobel(image_gradient_y,image_gradient_y,CV_32FC1,1,0);
    // Absolute value and normalize
    cv::convertScaleAbs(image_gradient_y,image_gradient_y);

    // Average the x and y gradients into one image
    cv::Mat image_gradient;
    cv::addWeighted(image_gradient_x,0.5,image_gradient_y,0.5,0,image_gradient);
    cv::imwrite("photo_gradient.jpg",image_gradient);
}
```
Edge Detection: Sample Results
Face Detection: Viola-Jones

- Robust and fast
- Introduced by Paul Viola and Michael Jones
- Haar-like Features
And Many More ...

- [Object Tracking using OpenCV](#)
- [Handwritten Digits Classification : An OpenCV ( C++ / Python ) Tutorial](#)
- [Eye Detector using OpenCV](#)
- [Image Recognition and Object Detection](#)
- [Head Pose Estimation using OpenCV](#)
- [Configuring Qt for OpenCV](#)
- ...
Deep Learning Frameworks
Deep Learning Frameworks

- Caffe
- Torch/PyTorch
  - NYU
  - scientific computing framework in Lua
  - supported by Facebook
- TensorFlow
  - Google
  - Python
- Theano/Pylearn2
  - U. Montreal
  - Python
  - symbolic computation and automatic differentiation
- MatConvNet
  - Oxford U.
  - Deep Learning in MATLAB
Framework Comparison

• More alike than different
  • All express deep models
  • All are open-source (contributions differ)
  • Most include scripting for hacking and prototyping

• No strict winners, experiment and choose the framework that best fits your work
Caffe: Overview

• What is Caffe?
• Training/Finetuning a simple model
• Deep dive into Caffe!
What is Caffe?

• A deep learning framework

• Open framework, models, and worked examples for deep learning

• 4000+ citations, 250+ contributors, 11,000+ forks

• Focus has been vision, but branching out: sequences, reinforcement learning, speech + text
Caffe

• Pure C++ / CUDA architecture for deep learning
  • command line, Python, MATLAB interfaces
• Fast, well-tested code
• Tools, reference models, demos, and recipes
• Switch between CPU and GPU
  • Caffe::set_mode(Caffe::GPU);
Installation

- http://caffe.berkeleyvision.org/installation.html
- CUDA, OPENCV
- BLAS (Basic Linear Algebra Subprograms): operations like matrix multiplication, matrix addition, both implementation for CPU(cBLAS) and GPU(cuBLAS). provided by MKL(INTEL), ATLAS, openBLAS, etc.
- Boost: a c++ library. > Use some of its math functions and shared_pointer.
- glog, gflags provide logging & command line utilities. > Essential for debugging.
- leveldb, lmdb: database io for your program. > Need to know this for preparing your own data.
- protobuf: an efficient and flexible way to define data structure. > Need to know this for defining new layers.
Caffe Tutorial

- **Nets, Layers, and Blobs**: the anatomy of a Caffe model.
- **Forward / Backward**: the essential computations of layered compositional models.
- **Loss**: the task to be learned is defined by the loss.
- **Solver**: the solver coordinates model optimization.
- **Interfaces**: command line, Python, and MATLAB Caffe.
- **Data**: how to caffeinate data for model input.

http://caffe.berkeleyvision.org/tutorial/
Caffe

• **Blob**: Storage and Communication of Data
  • Data blobs are N x C x H x W

• **Net**: Contains all the layers in the networks
  • Performs forward/backward pass through the entire network

• **Solver**: Used to set training/testing parameters
  • Number of iterations, back propagation method, etc..
Training: Step 1

• Create a lenet_train.prototxt

• Data Layers

• Operational Layers

• Loss Layers
Network Definition (train.prototxt)

```
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: "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"
    }
  }
```

Network Definition (train.prototxt)

```c
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {  
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}

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"
    }
  }
}
```
Network Definition(train.prototxt)

```java
layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "ip2"
    bottom: "label"
    top: "loss"
}
```
Training: Step 2

- Create a lenet_solver.prototxt

```python
train_net: "lenet_train.prototxt"
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
max_iter: 10000
snapshot_prefix: "lenet_snapshot"
# ... and some other options ...
```
Solver(solver.prototxt)

```
# 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: "step"
gamma: 0.1
stepsize: 3000
# 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: Step 2

• Some details on SGD parameters

\[ V_{t+1} = \mu V_t - \alpha (\nabla L(W_t) + \lambda W_t) \]

\[ W_{t+1} = W_t + V_{t+1} \]
Training: Step 3

• # train LeNet
  • caffe train -solver examples/mnist/lenet_solver.prototxt

• # train on GPU 2
  • caffe train -solver examples/mnist/lenet_solver.prototxt -gpu 2

• # resume training from the half-way point snapshot
  • caffe train -solver examples/mnist/lenet_solver.prototxt -snapshot examples/mnist/lenet_iter_5000.solverstate
Network Definition(test.prototxt)

Previously

```protobuf
define network:
  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
    }
  }
```
Network Definition (test.prototxt)

Previously

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

```
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
```
PyCaffe (Training in Python)

- Add caffe python directory to path and import caffe

```python
caffe_root = '..', # this file should be run from {caffe_root}/examples (otherwise change this line)

import sys
sys.path.insert(0, caffe_root + 'python')
import caffe
```
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='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()
Define solver and train network

```python
define solver and train network

```
Access Net data

```python
# we use a little trick to tile the first eight images
imshow(solver.net.blobs['data'].data[:8, 0].transpose(1, 0, 2).reshape(28, 8*28), cmap='gray'); axis('off')
print 'train labels:', solver.net.blobs['label'].data[:8]
```

train labels: [ 5.  0.  4.  1.  9.  2.  1.  3.]

5 0 4 1 9 2 1 3
PyCaffe (Testing in Python)

```python
# load the model
net = caffe.Net('models/bvlc_reference_caffenet/train.prototxt',
                'models/bvlc_reference_caffenet/train_iter30000.caffemodel',
                caffe.TEST)

# load input and configure preprocessing
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_mean('data', np.load('ilsvrc_2012_mean.npy').mean(1).mean(1))
transformer.set_transpose('data', (2,0,1))
transformer.set_channel_swap('data', (2,1,0))
transformer.set_raw_scale('data', 255.0)

# note we can change the batch size on-the-fly
# since we classify only one image, we change batch size from 10 to 1
net.blobs['data'].reshape(1,3,227,227)

# load the image in the data layer
im = caffe.io.load_image('examples/images/cat.jpg')
net.blobs['data'].data[...] = transformer.preprocess('data', im)

# compute
out = net.forward()

# other possibility: out = net.forward_all(data=np.asarray([transformer.preprocess('data', im)]))

# predicted predicted class
print out['prob'].argmax()
```
Open Model Collection

• The Caffe Model Zoo
• open collection of deep models to share innovation
  • VGG ILSVRC14
  • Network-in-Network
    • MIT Places scene recognition model in the zoo
  • Help reproduce research
  • Bundled tools for loading and publishing models

• Share Your Models!
Reference Models

Alexnet: Imagenet Classification

Caffe offers the

- Model definitions
- Optimization settings
- Pre-trained weights so you can start right away.

R-CNN: Regions with CNN features

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions
Why Fine-tuning?

• **Fine tuning** is a process to take a network model that has already been trained for a given task, and retrain it to make it perform a second similar task.

• Why?
  • More robust optimization
  • good initialization helps
  • Needs less data
  • Faster learning
Fine-tuning Tricks

• Learn the last layer first
  • Caffe layers have local learning rates: blobs_lr
  • Freeze all but the last layer for fast optimization and avoiding early divergence.
  • Stop if good enough, or keep fine-tuning

• Reduce the learning rate
  • Drop the solver learning rate by 10x, 100x –
  • Preserve the initialization from pre-training and avoid thrashing
Training tips

• Before running final/long training
  • Make sure you can overfit on a small training set
  • Make sure your loss decreases over first several iterations
  • Otherwise adjust parameter until it does, especially learning rate

• Separate train/val/test data
Any Questions?