CAFFE-DEEP LEARNING FRAMEWORK

Omair Hassaan
LUMS MS(CS)
Research Associate @ITU
omair.hassaan@gmail.com

LAYOUT

1
- Brief Introduction
- Basic requirements for setting up Caffe Environment

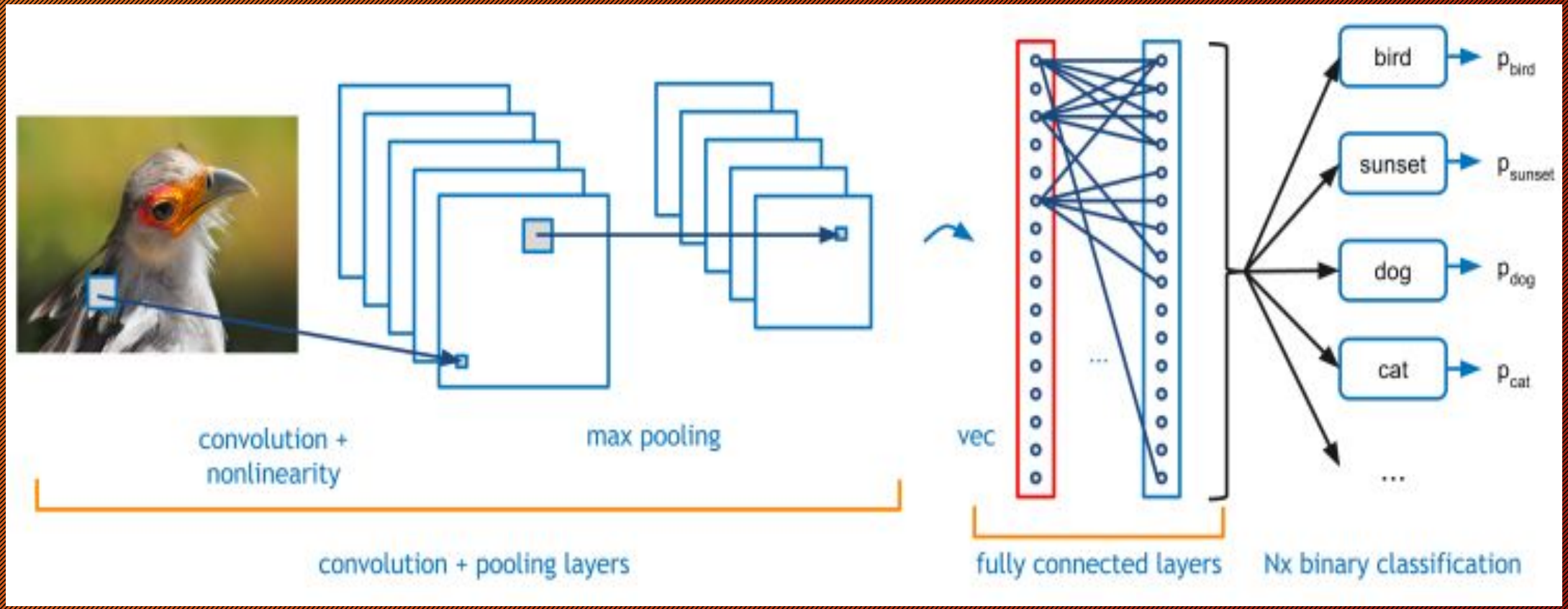
2
- Download and Install Caffe
- Train GoogLeNet on MNIST Data set

3
- Testing a trained model with test image
- Training your own dataset

4
- Extract Features from any layer in Caffe

What is Caffe?

- Caffe = Convolution Architecture for Fast Feature Embedding



What is Caffe?

- Open framework for Deep learning.
- 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
- Offers model definitions, optimization settings and pretrained weights.

Setting Up Caffe Environment

- Core2duo/i3/i5/i7 machine
- Sufficient amount of RAM
- GPU- shared/dedicated. (Caffe can also work without GPU)
- Dual boot Linux distribution. (NO virtual machine !)
- Internet connection
- Patience !

Installing Caffe

- Setup CUDA
  - *http://developer.download.nvidia.com/compute/cuda/6_5/rel/installers/cuda_6.5.14_linux_64.run*
  - *chmod +x cuda_6.5.14_linux_64.run*
  - *sudo ./cuda_6.5.14_linux_64.run --kernel-source-path=/usr/src/linux-headers-`uname -r`/*
    - Accept the EULA
    - Install the graphics card drivers. (If GPU)
    - Install the toolkit (leave path at default)
    - Install symbolic link
    - Install samples (leave path at default)
- Update Library Path
  - *cp /etc/bash.bashrc /home/user/.bashrc*
  - *echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc*
  - *echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local/lib' >> ~/.bashrc*
  - *source ~/.bashrc*

Download Caffe

- Download Caffe
  - *https://github.com/omair18/Caffe-Setup-Testing-Training*

- Extract caffe-master.zip in your Home directory (/home/username/)

Install Dependencies

- Install dependencies
  - cd caffe-master (I'll call this folder as $CAFFE-ROOT)
  - ./linux_dep.sh
  - ./python_dep.sh
  - Patience ! :)
- Add a couple of Symbolic links for some reason
  - sudo ln -s /usr/include/python2.7/ /usr/local/include/python2.7
  - sudo ln -s /usr/local/lib/python2.7/dist-packages/numpy/core/include/numpy/ <space> /usr/local/include/python2.7/numpy

Compiling Caffe

- Compile Caffe
  - cp Makefile.config.example Makefile.config
  - nano Makefile.config
    - If you don't have GPU, uncomment the line #CPU_ONLY := 1
    - Under PYTHON_INCLUDE replace /usr/lib/python2.7/dist-packages/numpy/core/include with /usr/local/lib/python2.7/dist-packages/numpy/core/include (i.e. add /local)
- Start Compilation
  - make pycaffe //for Python
  - Optional:
    - Make matcaffe //for matlab
  - make all
  - make test
  - Patience :)

Testing Installation

- Download the ImageNet Caffe model and labels
  - ./scripts/download_model_binary.py models/bvlc_reference_caffenet
  - ./data/ilsvrc12/get_ilsvrc_aux.sh
- First Download ilsvrc12 package
  - $CAFFE_ROOT/data/get_ilsvrc_aux.sh
  - cp $CAFFE_ROOT/caffe_ilsvrc12/synset_words.txt $CAFFE_ROOT/data/ilsvrc12/
- Test your installation by running the ImageNet model on an image of a kitten
  - python python/classify.py --print_results examples/images/cat.jpg foo
  - Expected result: [('tabby', '0.27933'), ('tiger cat', '0.21915'), ('Egyptian cat', '0.16064'), ('lynx', '0.12844'), ('kit fox', '0.05155')]

1
- Brief Introduction
- Basic requirements for setting up Caffe Environment

2
- Download and Install Caffe
- Train GoogLeNet on MNIST Data set

3
- Testing a Trained model
- Training your own dataset

- Extract Features from any layer in Caffe

Training LeNet on MNIST dataset

- Change directory to $CAFFE_ROOT
- Prepare dataset
  - ./data/mnist/get_mnist.sh
  - ./examples/mnist/create_mnist.sh
- We will get 2 lmdb files from the above step
  - Mnist_train_lmdb
  - Mnist_test_lmdb
- All training layers are written in *.protoxt files
  - Data/Convolution/Pooling/Inner product/ReLU/Loss
- Define Mnist solver file for training/test protocol buffer definition
  - $CAFFE_ROOT/examples/mnist/lenet_solver.prototxt
  - Contains training iterations, base learning rate, max # iterations and solver mode

Network Layers(Vision Layers)

- Vision layers usually take images as input and produce images as output.
- Most of the vision layers work by applying a particular operation to some region of the input to produce a corresponding region of the output
- Examples
  - Convolution Layer
  - Pooling Layer
  - Max layer ..

Convolution Layer

- Layer type: Convolution
- CPU implementation:
  - $CAFFE_ROOT/src/caffe/layers/convolution_layer.cpp
- Parameters (ConvolutionParameter convolution_param)
- num_output (c_o):
  - the number of filters
- kernel_size :
  - specifies height and width of each filter
- weight_filler [default type: 'constant' value: 0]
- bias_term [default true]:
  - specifies whether to learn and apply a set of additive biases to the filter outputs
- pad [default 0]:
  - specifies the number of pixels to (implicitly) add to each side of the input
- stride [default 1]:
  - specifies the intervals at which to apply the filters to the input

```
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
    }
  }
}
```

Pooling Layer

- Layer type: Pooling
- CPU implementation:
  - $CAFFE_ROOT/src/caffe/layers/pooling_layer.cpp
- Parameters (PoolingParameter pooling_param)
- kernel_size (or kernel_h and kernel_w):
  - specifies height and width of each filter
- pool [default MAX]:
  - the pooling method. Currently MAX, AVE, or STOCHASTIC
- pad (or pad_h and pad_w) [default 0]: specifies the number of pixels to (implicitly) add to each side of the input
- stride (or stride_h and stride_w) [default 1]:
  - specifies the intervals at which to apply the filters to the input
- pad (or pad_h and pad_w) [default 0]:
  - specifies the number of pixels to (implicitly) add to each side of the input
- stride (or stride_h and stride_w) [default 1]:
  - specifies the intervals at which to apply the filters to the input

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3 # pool over a 3x3 region
    stride: 2      # step two pixels (in the bottom blob) between pooling regions
  }
}
```

Loss Layer

- Loss drives learning by comparing an output to a target and assigning cost to minimize
- Softmax:
  - Layer-type: SoftmaxWithLoss
- Sum of Squares/Euclidean
  - Layer-type: EuclideanLoss
- Hinge/Margin
  - Layer type: HingeLoss

```
# L1 Norm
layer {
  name: "loss"
  type: "HingeLoss"
  bottom: "pred"
  bottom: "label"
}

# L2 Norm
layer {
  name: "loss"
  type: "HingeLoss"
  bottom: "pred"
  bottom: "label"
  top: "loss"
  hinge_loss_param {
    norm: L2
  }
}
```

ReLu Layer (Neuron Layer)

- Neuron layers are element-wise operators, taking one bottom blob and producing one top blob of the same size.
- Layer type: ReLU
  - Sigmoid, TanH
  - http://caffe.berkeleyvision.org/tutorial/laye
- CPU implementation:
  - $CAFFE_ROOT/src/caffe/layers/relu_layer.cpp
- Parameters (ReLUParameter relu_param)
- negative_slope [default 0]:
  - specifies whether to leak the negative part by multiplying it with the slope value rather than setting it to 0.

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
```

Training LeNet on MNIST dataset

- Start training using
  - $CAFFE_ROOT/examples/mnist/train_lenet.sh
  - ./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
- End result will be a *.caffemodel file containing our trained network.

1
- Brief Introduction
- Basic requirements for setting up Caffe Environment

2
- Download and Install Caffe
- Train GoogLeNet on MNIST Data set

3
- Testing a Trained model
- Training your own dataset

- Extract Features from any layer in Caffe

Testing a Trained Model

- Create a LMDB database of the testing image set.
- You must have prototxt files available and also the snapshot of trained model.
- Use the following python script to test your trained mode.
  - Test_your_own_model.py

1
- Brief Introduction
- Basic requirements for setting up Caffe Environment

2
- Download and Install Caffe
- Train GoogLeNet on MNIST Data set

3
- Testing a trained model
- Training your own dataset

4
- Extract Features from any layer in Caffe

Creating LMDB with image dataset

- Create 2 files containing [path/to/image <space> label]
  - Train.txt
  - Val.txt
- Reference example
  - $CAFFE_ROOT/examples/imagenet/create_imagenet.sh
- Download ilsvrc12 package by running
  - $CAFFE_ROOT/data/get_ilsvrc_aux.sh
  - Open $CAFFE_ROOT/data/ilsvrc12/train.txt or val.txt
- Run the *convert_imageset* program to convert all the images to lmdb format using
  - $CAFFE_ROOT/build/tools/convert_imageset [FLAGS] ROOTFOLDER LISTFILE DB_NAME
  - FLAGS can be gray, shuffle, backend, resize_width, resize_height, check_size, encoded, encoded type.
  - ROOTFOLDER = folder containing all the images
  - LISTFILE = train.txt/val.txt for creating training/testing database

Training with your own Dataset

- After creating your own LMDB database, it's time to create prototxt files containing all the layers information of your network.
  - References
  - $CAFFE_ROOT/examples/mnist/lenet.prototxt
  - $CAFFE_ROOT/examples/minst/lenet_train_test.prototxt
  - $CAFFE_ROOT/examples/mnist/lenet_solver.prototxt
  - $CAFFE_ROOT/examples/mnist/train_lenet.sh
- Change path of prototxt files in train_lenet.sh file(see “Training LeNet on MNIST using Caffe” for reference)
- Start training by running train_lenet.sh

1
- Brief Introduction
- Basic requirements for setting up Caffe Environment

2
- Download and Install Caffe
- Train GoogLeNet on MNIST Data set

3
- Testing a trained model
- Training your own dataset

4
- Extract Features from any layer in Caffe

Create Mean of Imageset(binaryproto)

- Crop all images to 227x227 or 256x256
- Create LMDB database of the image set .
- Now run
  - $CAFFE_ROOT/build/tools/compute_image_mean [LMDB] [FILE NAME]
  - LMDB is the database you created
  - FILE NAME is the destination filename that will be created with .binaryproto extension

Extract Features from Any layer in Caffe

- Crop all images to 227x227 or 256x256 & Create LMDB database of the image set .
- Create a prototxt file, in which we have to mention our LMDB database and mean of image set (binaryproto)
  - Reference $CAFFE_ROOT/examples/feature_extraction/imagenet_val.protoxt
- Now Run
  - $CAFFE_ROOT/build/tools/extract_features.bin [MODEL File] [NETWORK] [LAYER] [DESTINATION] [BATCHSIZE] [LMDB]
  - MODEL FILE = pretrained caffe model
    - $CAFFE_ROOT/models/bvlc_reference_caffe_net/bvlc_caffenet.caffemodel
  - NETWORK = Prototxt file we just made in step 3
  - LAYER = Layer name whose features we want to extract e.g fc7
  - DESTINATION = Folder name to save results
  - BATCH SIZE e.g 10
  - LMDB = to store resutls in LMDB format