Deep Neural Networks Basics

For ECS 289G
Presented by Fanyi Xiao
Computer Vision in the Pre-DNN Era

Face Detection, Viola & Jones, 2001
Computer Vision in the Pre-DNN Era

“SIFT” & Object Recognition, David Lowe, 1999
Computer Vision in the Pre-DNN Era

Spatial Pyramid Matching, Lazebnik, Schmid & Ponce, 2006
Computer Vision in the Pre-DNN Era

Histogram of Gradients (HoG)
Dalal & Triggs, 2005

Deformable Part Model
Felzenswalb, McAllester, Ramanan, 2009
Emergence of DNNs in Vision

- **Year 2010**
  - NEC-UIUC
  - Dense grid descriptor: HOG, LBP
  - Coding: local coordinate, super-vector
  - Pooling, SPM
  - Linear SVM
  - [Lin CVPR 2011]

- **Year 2012**
  - SuperVision

- **Year 2014**
  - GoogLeNet
  - VGG
  - MSRA
  - [Szegedy arxiv 2014]  [Simonyan arxiv 2014]  [He arxiv 2014]
Neural Networks

Image Classification

Learn visual features "end-to-end"
Compositional Models
Learned End-to-End

Hierarchy of Representations
- vision: pixel, motif, part, object
- text: character, word, clause, sentence
- speech: audio, band, phone, word

figure credit Yann LeCun, ICML ‘13 tutorial
Neural Networks

Three key ingredients for training an NN:

1. Score function
2. Loss function
3. Optimization
Neural Networks

Three key ingredients for training an NN:

1. Score function: $y = f(x, W)$

$x \rightarrow 224 \times 224 \times 3$ image patch

$y \rightarrow 1000d$ vector
Three key ingredients for training an NN:

2. Loss function: for example max-margin loss and cross-entropy loss

\[ L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) \]

\[ L_i = -\log \left( \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \]
Neural Networks

Three key ingredients for training an NN:

3. Optimization: simple gradient descent!
Neural Networks

Three key ingredients for training an NN:

3. Optimization: in practice, stochastic (mini-batch) gradient descent!

```python
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += -step_size * weights_grad  # perform parameter update
```
Neural Networks

Three key ingredients for training an NN:

3. Optimization: in practice, stochastic (mini-batch) gradient descent + momentum! (Many other optimization methods like adagrad/rmsprop)

```python
weights_grad = evaluate_gradient(loss_fun, data, weights)
vel = vel * 0.9 - step_size * weights_grad
weights += vel
```
Convolution Neural Networks

Let's take a closer look at AlexNet

Linear transformation:
$$y' = Wx + b$$
Convolution Neural Networks

Let's take a closer look at AlexNet

Linear transformation:
\[ y' = f(Wx + b) \]
Let's take a closer look at AlexNet

Convolution Neural Networks
Convolution Neural Networks

conv(h,w,stride)
Example: conv(h=3, w=3, stride=1)

(7-3)/1+1=5
End up as a 5*5 feature map
Let's take a closer look at AlexNet

maxpool(h,w,stride)
Convolution Neural Networks

Example: maxpool(h=2,w=2,stride=2)
Convolution Neural Networks

Let's take a closer look at AlexNet

Relu: $y = \max(y', 0)$
Problems with tanh: Saturated response

ReLU: \( y = \max(y', 0) \)
- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice!

However, non-bounded response and dead when less than 0
(Improved version leaky ReLU)
Convolution Neural Networks

There are two key differences to Vanilla Neural Nets: neurons arranged in 3D volumes have local connectivity, share parameters.
Convolution Neural Networks

ILSVRC14 Winners: ~6.6% Top-5 error
- GoogLeNet: composition of multi-scale dimension-reduced modules (pictured)
- VGG: 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers

+ depth
+ data
+ dimensionality reduction
Convolution Neural Networks

Object Detection

R-CNN: Region-based Convolutional Networks
http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/detection.ipynb
Full R-CNN scripts available at
https://github.com/rbgirshick/rcnn

Ross Girshick et al.

Fast R-CNN
arXiv and code
Segmentation

Fully convolutional networks for pixel prediction applied to semantic segmentation
end-to-end learning efficiency in inference and learning
175 ms per-image prediction
multi-modal, multi-task

Further applications
- depth estimation
- denoising

arXiv and pre-release
Problem with Feed-forward Nets

What if we want to be able to have a model telling us what's the probability of the following two sentences, respectively:

1. The cat sat on the mat
2. The mat is having dinner with the cat
Problem with Feed-forward Nets

What if we want to be able to have a model telling us what's the probability of the following two sentences, respectively:

1. The cat sat on the mat
2. The mat is having dinner with the cat

Cannot handle variable length input
Recurrent Neural Net

RNNs tie the weights at each time step

\[ h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right) \]

\[ \hat{y}_t = \text{softmax} \left( W^{(S)} h_t \right) \]
Recurrent Neural Net

Training of RNNs is hard...

\[
    h_t = W f(h_{t-1}) + W^{(h,x)} x[t]
\]

\[
    \frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag}[f'(h_{j-1})]
\]
Recurrent Neural Net

Training of RNNs is hard...

Solution 1: clip the gradient!

**Algorithm 1** Pseudo-code for norm clipping the gradients whenever they explode

\[ \hat{g} \leftarrow \frac{\partial E}{\partial \theta} \]

\[ \text{if } \|\hat{g}\| \geq \text{threshold} \text{ then} \]

\[ \hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g} \]

end if

Some theory: On the difficulty of training recurrent neural networks, Pascanu et al. ICML2013
Recurrent Neural Net

Training of RNNs is hard...

Solution 2: NNs with gating units (LSTM/GRU)
Recurrent Neural Net

Training of RNNs is hard...

Solution 2: nets with gating units (LSTM/GRU)
Recurrent Neural Net

Training of RNNs is hard...

Solution 2: nets with gating units (LSTM/GRU)
RNN in vision

Image captioning

RNN in vision

Visual attention model

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, Kelvin Xu et al.
RNN in vision

RNNs for Human Dynamics

Recurrent Network Models for Human Dynamics, Katerina Fragkiadaki et al.
1. Numerical gradient check

```python
fx = f(x)  # evaluate function value at original point
grad = np.zeros_like(x)
# iterate over all indexes in x
it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
while not it.finished:
    # evaluate function at x+h
    ix = it.multi_index
    oldval = x[ix]
    x[ix] = oldval + h  # increment by h
    fxph = f(x)  # evaluate f(x + h)
    x[ix] = oldval - h
    fxmh = f(x)  # evaluate f(x - h)
    x[ix] = oldval  # restore

    # compute the partial derivative with centered formula
    grad[ix] = (fxph - fxmh) / (2 * h)  # the slope
    if verbose:
        print(ix, grad[ix]
    it.iternext()  # step to next dimension
```
Tricks

1. Numerical gradient check
2. Modulize layers: only three functions needed
   (1) output=forward(input,model)
   (2) dJ_dW=computeParamGrad(input,outputGrad,model)
   (3) dJ_dInput=computeInputGrad(input,outputGrad,model)

Everything else is just putting together lego pieces
Questions?

Thanks!
Caffe Tutorial

For ECS 289G
Presented by Krishna Kumar Singh

*Some slides taken from CVPR 2015 deep learning tutorial*
What is Caffe?
Open framework, models, and worked examples for deep learning

- 3 years
- 2400+ citations, 100+ contributors
- 7,000+ forks, >1 pull request / day average
- focus has been vision, but branching out: sequences, reinforcement learning, speech + text
What is Caffe?
Open framework, models, and worked examples 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
Caffe is a Community

May 6, 2015 – June 6, 2015

Overview

62 Active Pull Requests
45 Merged Pull Requests
17 Proposed Pull Requests

168 Active Issues
122 Closed Issues
46 New Issues

Excluding merges, 43 authors have pushed 75 commits to master and 203 commits to all branches. On master, 154 files have changed and there have been 46,336 additions and 5,964 deletions.
Brewing by the Numbers...

- **Speed with Krizhevsky's 2012 model:**
  - 2 ms / image on K40 GPU
  - 0.32 ms inference with Caffe + cuDNN v5 on Titan X
  - 270 million images / day with batched IO
  - 8-core CPU: ~20 ms/image
- **9k lines of C++ code (20k with tests)**
Share a Sip of Brewed Models

demo.caffe.berkeleyvision.org
demo code open-source and bundled
Training Network for Style Recognition
Visual Style Recognition


Other Styles:
- Vintage
- Long Exposure
- Noir
- Pastel
- Macro
- … and so on.

20 Classes
Alexnet for Flickr Style Recognition

* Image taken from Alex Krizhevsky slides
Training

- **Network Schema**: Define layers and connection between them.
  - `models/finetune_flickr_style/train_val.prototxt`

- **Solver**: Oversees the network optimization.
  - `models/finetune_flickr_style/solver.prototxt`
Basic Definitions

• Learning Rate: Decides the rate of change of parameters.

• Batch Size: Number of images processed together in the network.

• Epoch: Number of iterations required to feed the entire training data.
  ➢ Training images: 1000, Batch Size = 50, then 1 epoch = 1000/50 = 20 iterations

• Testing: Compute accuracy and loss of intermediate models.
  ➢ Does not affect the network parameters.
Defining Network Schema in Caffe

Please open models/finetune_flickr_style/train_val.prototxt
Layers

- Network is defined as series of layers.
- Layers type:-
  - Convolution
  - Inner Product
  - Data
  - Loss
  - Pooling
  ......
- Each layer has input as bottom blob and output as top blob.

*Image taken from Caffe Tutorial*
Network

name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
Data Layer

+ labels
Data Layer

Input Image

Image Label
(Number between 0 to 19)

*image taken from Caffe Tutorial
**Data Layer**

```
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 227
    mean_file: "data/ilsrv12/imagenet_mean.binaryproto"
  }
  image_data_param {
    source: "data/flickr_style/train.txt"
    batch_size: 50
    new_height: 256
    new_width: 256
  }
}
```

- **Layer Output**
- Randomly flip training images
- Subtract mean from images
- List of training images with labels
- Resize image
Input File Format

- Open /data/flickr_style/train.txt
- Format: <Image Path> <Class Label>
- Example:
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/12123529133_c1fb58f6dc.jpg 16
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/11603781264_0a34b7cc0a.jpg 12
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/12852147194_4c07cf49a1.jpg 9
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/8516303191_c243a80454.jpg 10
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/12223105575_581855dc34.jpg 2
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/9838077083_1b18cfca00.jpg 0
  /home/krishna/Downloads/caffe-master/data/flickr_style/images/8660745510_dee5e4d819.jpg 14
Conv1 Layer

```python
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
    }  \[ Learning rate and decay multiplier for weights \]
  param {
    lr_mult: 2
    decay_mult: 0
    }  \[ Learning rate and decay multiplier for bias \]
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    }  \[ Filter Size: 96 X 11 X 11 \]
}
```
Conv1 Filters and Response

96 conv1 filters

conv1 output, first 36 only

*Images taken from Caffe Tutorial
Max Pool Layer

+ labels
Max Pool Layer

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

Rectified Linear

layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}

Max Pooling
FC8 Layer

+ labels
layer {
  name: "fc8_flickr"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8_flickr"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 20  
    Number of classes
  }
}
Loss Layer

layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8_flickr"
  bottom: "label"
  top: "loss"
}

Loss = -log( Prob(X_{label}) )

fc8_flickr = [ X_0, X_1, \ldots, X_{19} ]

Prob(X_{label}) = \frac{e^{X_{label}}}{\sum_{i=0}^{19} e^{X_i}}

Simplified Form
Test Phase

layer {
    name: "data"
    type: "ImageData"
    top: "data"
    top: "label"
    include {
        phase: TEST
    }
}
transform_param {
    mirror: false
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
}
image_data_param {
    source: "data/flickr_style/test.txt"
    batch_size: 50
    new_height: 256
    new_width: 256
}

Check intermediate model on test images.

Contains test image list
Solving the Network

Please open models/finetune_flickr_style/solver.prototxt
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"

# solver_mode: CPU
Solver

`net: "models/finetune_flickr_style/train_val.prototxt"`

test_iter: 100

test_interval: 1000
ase_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"

# solver_mode: CPU
Solver

net: "models/finetune_flickr_style/train_val.prototxt"

**test_iter**: 100
**test_interval**: 1000
**base_lr**: 0.001
**lr_policy**: "step"
**gamma**: 0.1
**stepsize**: 20000
**display**: 20
**max_iter**: 100000
**momentum**: 0.9
**weight_decay**: 0.0005
**snapshot**: 10000
**snapshot_prefix**: "models/finetune_flickr_style/finetune_flickr_style"

# solver_mode: CPU

---

**Test_iter**: Number of batches for which testing will happen
**Test_interval**: Number of iteration after which testing will happen

For ex:- test batch_size =50
Then after every 1000 iterations testing will take place on 5000 (100*50) images
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000

**base_lr**: 0.001

lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000

**momentum**: 0.9

weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"

# solver_mode: CPU

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

\( \mu \): momentum
\( \alpha \): learning rate
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"
# solver_mode: CPU

After stepsize iterations decrease the learning rate by the factor of gamma
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20  Number of iterations after which loss is displayed
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"
# solver_mode: CPU
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000 Number of iterations for which training will take place
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"
# solver_mode: CPU
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"

After Snapshot iterations intermediate model is saved at snapshot_prefix
Solver

net: "models/finetune_flickr_style/train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
display: 20
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/finetune_flickr_style/finetune_flickr_style"
# solver_mode: CPU  If uncommented use CPU instead of GPU
Training Command

• Go to caffe root directory

• ./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt

Solver file
Training Command

- Go to caffe root directory
- `./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt`

Slow convergence and limited training data, Fine-tuning will be better
Fine Tuning

- In normal training parameters are initialized by random numbers.

- In finetuning, training starts with parameters initialized with already learned model’s parameters.

- Finetune style recognition with model learned for Imagenet 1000 classes classification task.

- Copy parameters(weights) from Imagenet model except last layer.

- models/finetune_flickr_style/train_val.prototxt is already modified for finetuning.

- Open models/bvlc_reference_caffenet/train_val.prototxt to see the differences.
### From Imagenet to Style

<table>
<thead>
<tr>
<th>Imagenet</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer {</td>
<td>layer {</td>
</tr>
<tr>
<td>name: &quot;fc8&quot;</td>
<td>name: &quot;fc8_flickr&quot;</td>
</tr>
<tr>
<td>type: &quot;InnerProduct&quot;</td>
<td>type: &quot;InnerProduct&quot;</td>
</tr>
<tr>
<td>bottom: &quot;fc7&quot;</td>
<td>bottom: &quot;fc7&quot;</td>
</tr>
<tr>
<td>top: &quot;fc8&quot;</td>
<td>top: &quot;fc8_flickr&quot;</td>
</tr>
<tr>
<td>param {</td>
<td>param {</td>
</tr>
<tr>
<td>lr_mult: 1</td>
<td>lr_mult: 10</td>
</tr>
<tr>
<td>decay_mult: 1</td>
<td>decay_mult: 1</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>param {</td>
<td>param {</td>
</tr>
<tr>
<td>lr_mult: 2</td>
<td>lr_mult: 20</td>
</tr>
<tr>
<td>decay_mult: 0</td>
<td>decay_mult: 0</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>inner_product_param {</td>
<td>inner_product_param {</td>
</tr>
<tr>
<td>num_output: 1000</td>
<td>num_output: 20</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

- Same layer name parameters copied,
- Different name start from random
- Higher learning rate because this layer is starting from random while the others are already trained
- Different number of classes
Fine-tuning Command

./build/tools/caffe train --solver=<solver.prototxt> --weights=<caffemodel>

./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt -weights models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
Fine-tuning Command

./build/tools/caffe train --solver=<solver.prototxt> --weights=<caffemodel>

./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt -weights models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel

Model learns faster, better accuracy at faster rate
Deciding Base Learning Rate

- Too high learning rate make loss –nan.
- Too low learning rate make learning slow.
- Good starting learning rate:
  - Training from scratch is 0.01
  - Fine-tuning is 0.001.
- Loss should decrease with iterations.

*Graph taken from chapter 3 of “Improving the way neural networks learn”*
Deciding Step Size and Max Iterations

- Decrease step size after ~18 epochs.
- Keep max iterations to be ~54 epochs.
- Monitor loss and accuracy on validation data.
Data Shuffling

• Training data and test data should be shuffled in random order.

• Allow non bias training:

/home/krishna/Downloads/caffe-master/data/flickr_style/images/12123529133_c1fb58f6dc.jpg 16
/home/krishna/Downloads/caffe-master/data/flickr_style/images/11603781264_0a34b7cc0a.jpg 12
/home/krishna/Downloads/caffe-master/data/flickr_style/images/12852147194_4c07cf49a1.jpg 9
/home/krishna/Downloads/caffe-master/data/flickr_style/images/8516303191_c243a80454.jpg 10
/home/krishna/Downloads/caffe-master/data/flickr_style/images/12223105575_581855dc34.jpg 2
/home/krishna/Downloads/caffe-master/data/flickr_style/images/9838077083_1b18cfca00.jpg 0
/home/krishna/Downloads/caffe-master/data/flickr_style/images/8660745510_dee5e4d819.jpg 14
Data Augmentation

• Useful when you don’t have enough training data.

• Some techniques:
  ➢ Rotate image by small angle.
  ➢ Random crops.
  ➢ Flipping image.
  ➢ Translate image.
  ➢ Adding random noise in RGB channel.

• Data augmentation should be uniformly distributed among classes.
Model Snapshot and Solverstate

• Take snapshot of your intermediate model frequently.

• Snapshot consist of: `snapshot_prefix_iterations.caffemodel` and `snapshot_prefix_iterations.solverstate`.

• Solverstate allows you to resume training.
  
  ```
  ./build/tools/caffe train --solver=<solver.prototxt> --snapshot=<solverstate>
  ```

• Install screen, useful for doing long training remotely.
Extract Features

• build/tools/extract_features.bin <model_name> <network definition> <blob name> <output file> <number of batches> <db_type> <device>

• In network definition file source should point to file containing list of images for which features have to be extracted.

• build/tools/extract_features.bin models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel examples/feature_extraction/imagenet_val.prototxt fc7 out_fc7 10 lmdb GPU
Time Benchmarking

- build/tools/caffe time -model <train_val prototxt> -iterations <number of iterations>

- Gives runtime analysis for each layer as well as for forward and back propagation.

- Bigger the batch size more the run time.
Regression

- Predict numbers associated with images.
- ‘image_data’ only allows single label per image.
- ‘HDF5Data’ allows image and label to be provided as vector.
- In fc8 layer, num_output will be equal to length of number vector you want to predict.
- Change Softmax loss to Euclidian loss.
Adding Layers

- src/caffe/layers

- CPU implementation: xx_layer.cpp
  - Forward_cpu
  - Backward_cpu

- GPU implementation: xx_layer.cu
  - Forward_gpu
  - Backward_gpu
Caffe offers the
● model definitions
● optimization settings
● pre-trained weights so you can start right away.

The BVLC models are licensed for unrestricted use.

The community shares models in Model Zoo.
Caffe Help

- http://caffe.berkeleyvision.org/tutorial/
- https://groups.google.com/forum/#!forum/caffe-users
- https://gitter.im/BVLC/caffe
Thank You