Getting Started with Caffe

Julien Demouth, Senior Engineer

What is Caffe?

Open Source Framework for Deep Learning

http://github.com/BVLC/caffe

- Developed by the Berkeley Vision and Learning Center (BVLC)
- C++/CUDA architecture
- Command line, Python, MATLAB interfaces
- Pre-processing and deployment tools, reference models and examples
- Image data management
- Seamless GPU acceleration
- Large community of contributors to the open-source project

What is Caffe?

End-to-end Deep Learning for the Practitioner and Developer

Prototype

Train

Deploy

Caffe Features

Data Pre-processing and Management

Data ingest formats

LevelDB or LMDB database

In-memory (C++ and Python only)

HDF5

Image files

Pre-processing tools

LevelDB/LMDB creation from raw images

Training and validation set creation with shuffling

Mean-image generation

```
$CAFFE_ROOT/build/tools
```

Data transformations

Image cropping, resizing, scaling and mirroring

Mean subtraction

Caffe Features

Deep Learning Model Definition

- Protobuf model format
  - Strongly typed format
  - Human readable
  - Auto-generates and checks Caffe code
  - Developed by Google
  - Used to define network architecture and training parameters
  - No coding required!

```
name: “conv1”
type: “Convolution”
bottom: “data”
top: “conv1”
convolution_param {
        num_output: 20
        kernel_size: 5
        stride: 1
        weight_filler {
                type: “xavier”
        }
}
```

Caffe Features

Deep Learning Model Definition

- Loss functions:
  - **Classification**
    - Softmax
    - Hinge loss
  - **Linear regression**
    - Euclidean loss
  - **Attributes/multiclassification**
    - Sigmoid cross entropy loss
  - **and more…**
- Available layer types:
  - Convolution
  - Pooling
  - Normalization
- Activation functions:
  - ReLU
  - Sigmoid
  - Tanh
  - and more…

Caffe Features

Deep Neural Network Training

Network training also requires no coding – just define a “solver” file

```
net: “lenet_train.prototxt”
base_lr: 0.01
momentum: 0.9
max_iter: 10000
snapshot_prefix: “lenet_snapshot”
solver_mode: GPU
```

All you need to run things on the GPU

```
> caffe train –solver lenet_solver.prototxt –gpu 0
```

Multiple optimization algorithms available: SGD (+momentum), ADAGRAD, NAG

Caffe Features

Monitoring the Training Process

Output to stdout:

```
I0814 14:44:33.410693 2026435328 solver.cpp:294] Iteration 0, Testing net (#0)
I0814 14:44:35.697690 2026435328 solver.cpp:343]     Test net output #0: accuracy = 0.0931
I0814 14:44:35.697720 2026435328 solver.cpp:343]     Test net output #1: loss = 2.30247 (* 1 = 2.30247 loss)
I0814 14:44:35.718361 2026435328 solver.cpp:214] Iteration 0, loss = 2.30184
I0814 14:44:35.718392 2026435328 solver.cpp:229]     Train net output #0: loss = 2.30184 (* 1 = 2.30184 loss)
I0814 14:44:35.718400 2026435328 solver.cpp:486] Iteration 0, lr = 0.001
I0814 14:44:41.550972 2026435328 solver.cpp:214] Iteration 100, loss = 1.72121
I0814 14:44:41.550999 2026435328 solver.cpp:229]     Train net output #0: loss = 1.72121 (* 1 = 1.72121 loss)
I0814 14:44:41.551007 2026435328 solver.cpp:486] Iteration 100, lr = 0.001
I0814 14:44:47.383386 2026435328 solver.cpp:214] Iteration 200, loss = 1.73216
I0814 14:44:47.383415 2026435328 solver.cpp:229]     Train net output #0: loss = 1.73216 (* 1 = 1.73216 loss)
I0814 14:44:47.383424 2026435328 solver.cpp:486] Iteration 200, lr = 0.001
I0814 14:44:53.220012 2026435328 solver.cpp:214] Iteration 300, loss = 1.30751
I0814 14:44:53.220772 2026435328 solver.cpp:229]     Train net output #0: loss = 1.30751 (* 1 = 1.30751 loss)
I0814 14:44:53.220782 2026435328 solver.cpp:486] Iteration 300, lr = 0.001
I0814 14:44:59.053917 2026435328 solver.cpp:214] Iteration 400, loss = 1.16627
I0814 14:44:59.053948 2026435328 solver.cpp:229]     Train net output #0: loss = 1.16627 (* 1 = 1.16627 loss)
I0814 14:44:59.053956 2026435328 solver.cpp:486] Iteration 400, lr = 0.001
I0814 14:45:04.833677 2026435328 solver.cpp:294] Iteration 500, Testing net (#0)
I0814 14:45:06.778378 2026435328 solver.cpp:343]     Test net output #0: accuracy = 0.5589
I0814 14:45:06.778411 2026435328 solver.cpp:343]     Test net output #1: loss = 1.2699 (* 1 = 1.2699 loss)
```

To visualize – pipe, parse and plot or use DIGITS

Caffe Features

Deep Neural Network Deployment

Standard, compact model format

`caffe train` produces a binary `.caffemodel` file

Easily integrate trained models into data pipelines

Deploy against new data using command line, Python or MATLAB interfaces

Deploy models across HW and OS environments

`.caffemodel` files transfer to any other Caffe installation (including DIGITS)

Caffe Features

Deep Neural Network Sharing

Caffe Model Zoo hosts community shared models

Benefit from networks that you could not practically train yourself

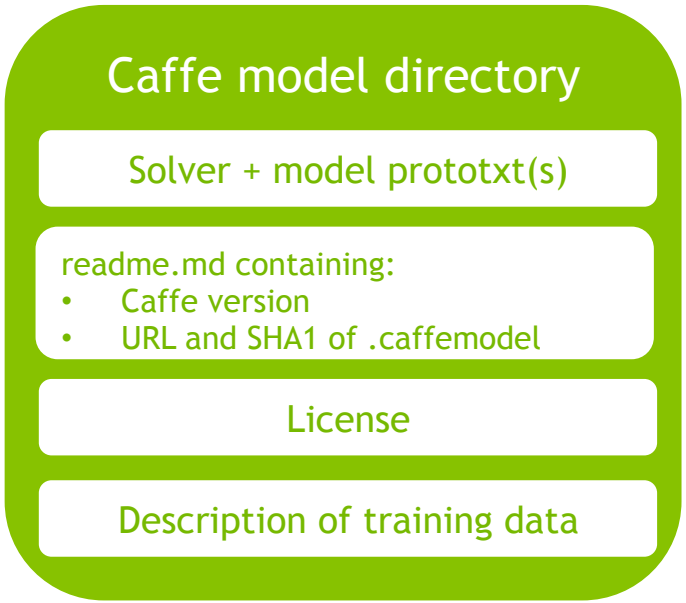
https://github.com/BVLC/caffe/wiki/Model-Zoo

Caffe comes with unrestricted use of BVLC models:

AlexNet

R-CNN

GoogLeNet



Caffe Features

Extensible Code

```python
import caffe
import numpy as np

class EuclideanLoss(caffe.Layer):

    def setup(self, bottom, top):
        # check input pair
        if len(bottom) != 2:
            raise Exception("Need two inputs to compute distance.")

    def reshape(self, bottom, top):
        # check input dimensions match
        if bottom[0].count != bottom[1].count:
            raise Exception("Inputs must have the same dimension.")
        # difference is shape of inputs
        self.diff = np.zeros_like(bottom[0].data, dtype=np.float32)
        # loss output is scalar
        top[0].reshape(1)

    def forward(self, bottom, top):
        self.diff[...] = bottom[0].data - bottom[1].data
        top[0].data[...] = np.sum(self.diff**2) / bottom[0].num / 2.

    def backward(self, top, propagate_down, bottom):
        for i in range(2):
            if not propagate_down[i]:
                continue
            if i == 0:
                sign = 1
            else:
                sign = -1
            bottom[i].diff[...] = sign * self.diff / bottom[i].num
```

Layer Protocol == Class Interface

Define a class in C++ or Python to extend `Layer`

Include your new layer in a network prototxt

```
layer {
  type: "Python"
  python_param {
    module: "layers"
    layer: "EuclideanLoss"
  }
}
```

Caffe Features

Extend Caffe from C++

Start from the caffe::Layer class

Implement the member functions

- LayerSetUp
- Reshape
- Forward_cpu/Forward_gpu
- Backward_cpu/Backward_gpu

```
/**
 * @brief Pools the input image by taking the max, average, etc. within regions.
 *
 * TODO(dox): thorough documentation for Forward, Backward, and proto params.
 */
template <typename Dtype>
class PoolingLayer : public Layer<Dtype> {
 public:
  explicit PoolingLayer(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 "Pooling"; }
  virtual inline int ExactNumBottomBlobs() const { return 1; }
  virtual inline int MinTopBlobs() const { return 1; }
  // MAX POOL layers can output an extra top blob for the mask;
  // others can only output the pooled inputs.
  virtual inline int MaxTopBlobs() const {
    return (this->layer_param_.pooling_param().pool() ==
            PoolingParameter_PoolMethod_MAX) ? 2 : 1;
  }

 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);

  int kernel_h_, kernel_w_;
  int stride_h_, stride_w_;
  int pad_h_, pad_w_;
  int channels_;
  int height_, width_;
  int pooled_height_, pooled_width_;
  bool global_pooling_;
  Blob<Dtype> rand_idx_;
  Blob<int> max_idx_;
};
```

Caffe Features

Extend Caffe from C++

Write the CUDA kernel

```
template <typename Dtype>
__global__ void MaxPoolForward(const int nthreads, const Dtype* bottom_data,
    const int num, const int channels, const int height,
    const int width, const int pooled_height, const int pooled_width,
    const int kernel_h, const int kernel_w, const int stride_h,
    const int stride_w, const int pad_h, const int pad_w, Dtype* top_data,
    int* mask, Dtype* top_mask) {
  CUDA_KERNEL_LOOP(index, nthreads) {
    int pw = index % pooled_width;
    int ph = (index / pooled_width) % pooled_height;
    int c = (index / pooled_width / pooled_height) % channels;
    int n = index / pooled_width / pooled_height / channels;
    int hstart = ph * stride_h - pad_h;
    int wstart = pw * stride_w - pad_w;
    int hend = min(hstart + kernel_h, height);
    int wend = min(wstart + kernel_w, width);
    hstart = max(hstart, 0);
    wstart = max(wstart, 0);
    Dtype maxval = -FLT_MAX;
    int maxidx = -1;
    bottom_data += (n * channels + c) * height * width;
    for (int h = hstart; h < hend; ++h) {
      for (int w = wstart; w < wend; ++w) {
        if (bottom_data[h * width + w] > maxval) {
          maxidx = h * width + w;
          maxval = bottom_data[maxidx];
        }
      }
    }
    top_data[index] = maxval;
    if (mask) {
      mask[index] = maxidx;
    } else {
      top_mask[index] = maxidx;
    }
  }
}
```

Forward_gpu launches the CUDA kernel

```
template <typename Dtype>
void PoolingLayer<Dtype>::Forward_gpu(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->gpu_data();
  Dtype* top_data = top[0]->mutable_gpu_data();
  int count = top[0]->count();
  // We'll output the mask to top[1] if it's of size >1.
  const bool use_top_mask = top.size() > 1;
  int* mask = NULL;
  Dtype* top_mask = NULL;
  switch (this->layer_param_.pooling_param().pool()) {
  case PoolingParameter_PoolMethod_MAX:
    if (use_top_mask) {
      top_mask = top[1]->mutable_gpu_data();
    } else {
      mask = max_idx_.mutable_gpu_data();
    }
    // NOLINT_NEXT_LINE(whitespace/operators)
    MaxPoolForward<Dtype><<<CAFFE_GET_BLOCKS(count), CAFFE_CUDA_NUM_THREADS>>>(
        count, bottom_data, bottom[0]->num(), channels_,
        height_, width_, pooled_height_, pooled_width_, kernel_h_,
        kernel_w_, stride_h_, stride_w_, pad_h_, pad_w_, top_data,
        mask, top_mask);
    break;
```

Caffe Features

Extend Caffe from C++

Extend the Protobuffer file

```
// NOTE
// Update the next available ID when you add a new LayerParameter field.
//
// LayerParameter next available layer-specific ID: 134 (last added: reshape_param)
message LayerParameter {

  // ...

  optional PoolingParameter pooling_param = 121;
  optional PowerParameter power_param = 122;
  optional PReLUParameter prelu_param = 131;
  optional PythonParameter python_param = 130;
  optional ReLUParameter relu_param = 123;
  optional ReshapeParameter reshape_param = 133;
  optional SigmoidParameter sigmoid_param = 124;
  optional SoftmaxParameter softmax_param = 125;
  optional SPPParameter spp_param = 132;
  optional SliceParameter slice_param = 126;
  optional TanHParameter tanh_param = 127;
  optional ThresholdParameter threshold_param = 128;
  optional WindowDataParameter window_data_param = 129;
}
```

```
658  message PoolingParameter {
659    enum PoolMethod {
660      MAX = 0;
661      AVE = 1;
662      STOCHASTIC = 2;
663    }
664    optional PoolMethod pool = 1 [default = MAX]; // The pooling method
665    // Pad, kernel size, and stride are all given as a single value for equal
666    // dimensions in height and width or as Y, X pairs.
667    optional uint32 pad = 4 [default = 0]; // The padding size (equal in Y, X)
668    optional uint32 pad_h = 9 [default = 0]; // The padding height
669    optional uint32 pad_w = 10 [default = 0]; // The padding width
670    optional uint32 kernel_size = 2; // The kernel size (square)
671    optional uint32 kernel_h = 5; // The kernel height
672    optional uint32 kernel_w = 6; // The kernel width
673    optional uint32 stride = 3 [default = 1]; // The stride (equal in Y, X)
674    optional uint32 stride_h = 7; // The stride height
675    optional uint32 stride_w = 8; // The stride width
676    enum Engine {
677      DEFAULT = 0;
678      CAFFE = 1;
679      CUDNN = 2;
680    }
681    optional Engine engine = 11 [default = DEFAULT];
682    // If global_pooling then it will pool over the size of the bottom by doing
683    // kernel_h = bottom->height and kernel_w = bottom->width
684    optional bool global_pooling = 12 [default = false];
685  }
```

Caffe Example Applications

Example applications

Use Case 1: Classification of Images

Object

http://demo.caffe.berkeleyvision.org/

Open source demo code:

`$CAFFE_ROOT/examples/web_demo`

Scene

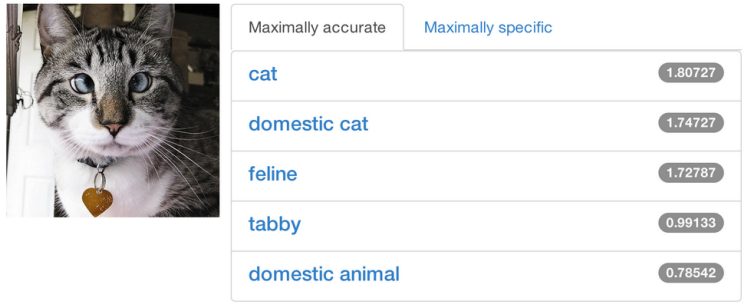
http://places.csail.mit.edu/

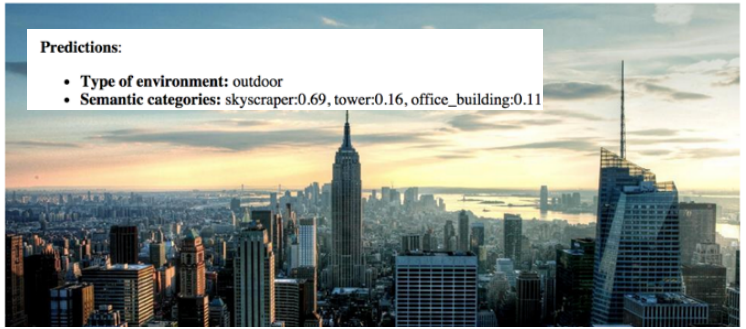
B. Zhou et al. NIPS 14

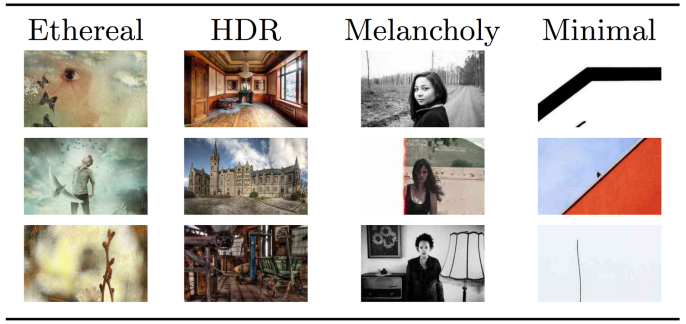
Style

http://demo.vislab.berkeleyvision.org/

Karayev et al. *Recognizing Image Style*. BMVC14

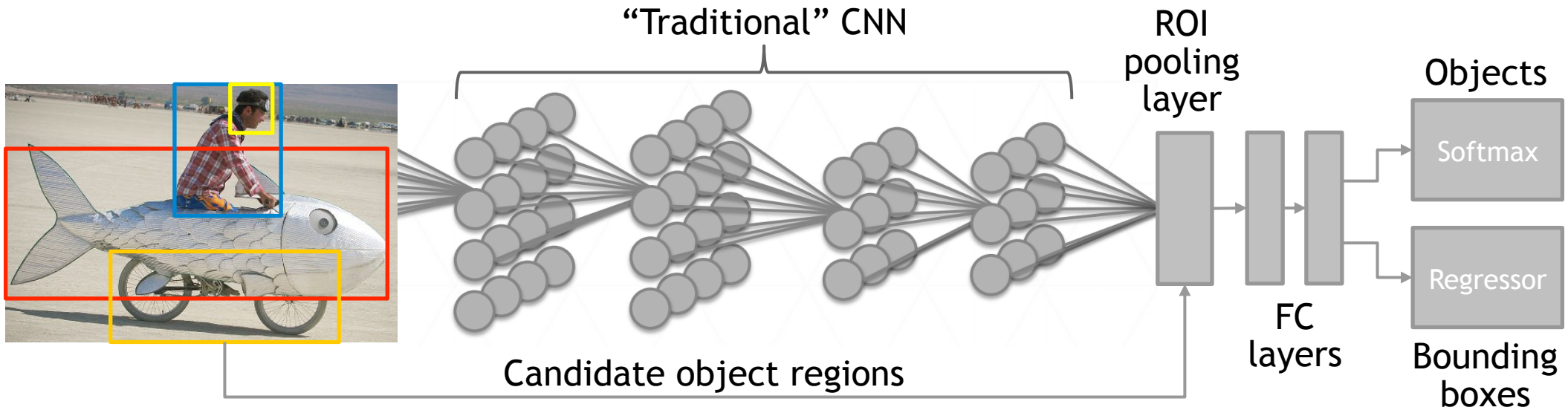






Example applications

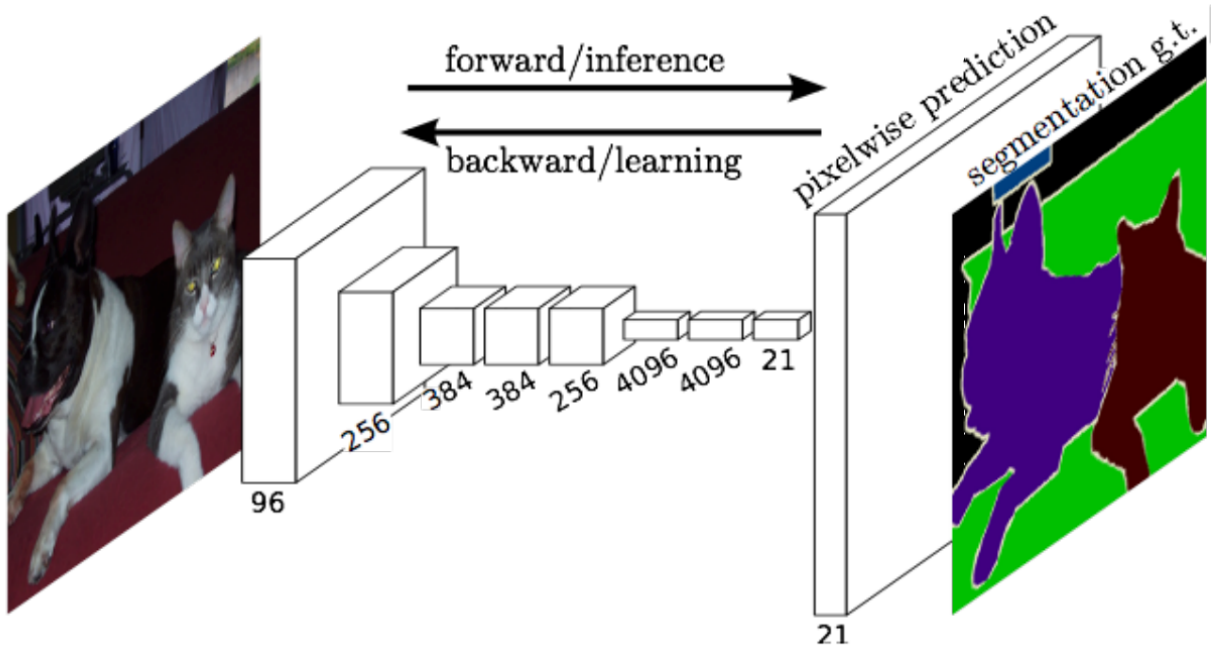
Use Case 2: Localization



(Fast) Region based Convolutional Networks (R-CNN)
Ross Girshick, Microsoft Research
https://github.com/rbgirshick/fast-rcnn

Example applications

Use Case 3: Pixel Level Classification and Segmentation



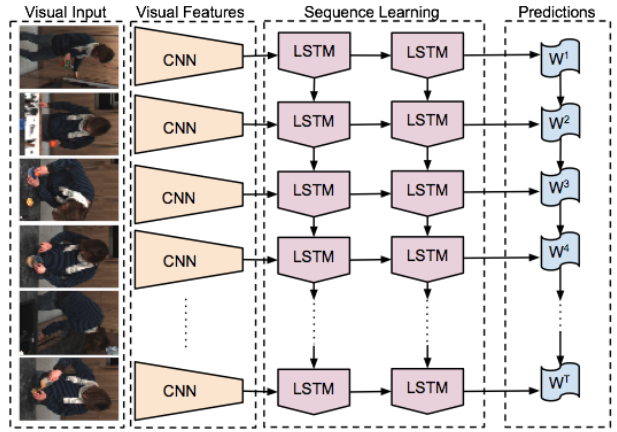
http://fcn.berkeleyvision.org

Long, Shelhamer, Darrell, *Fully convolutional networks for semantic segmentation*, CVPR 2015

Example applications

Use Case 4: Sequence Learning

- Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM)
  - Video
  - Language
  - Dynamic data
- Current Caffe pull request to add support
  - https://github.com/BVLC/caffe/pull/1873
  - http://arxiv.org/abs/1411.4389

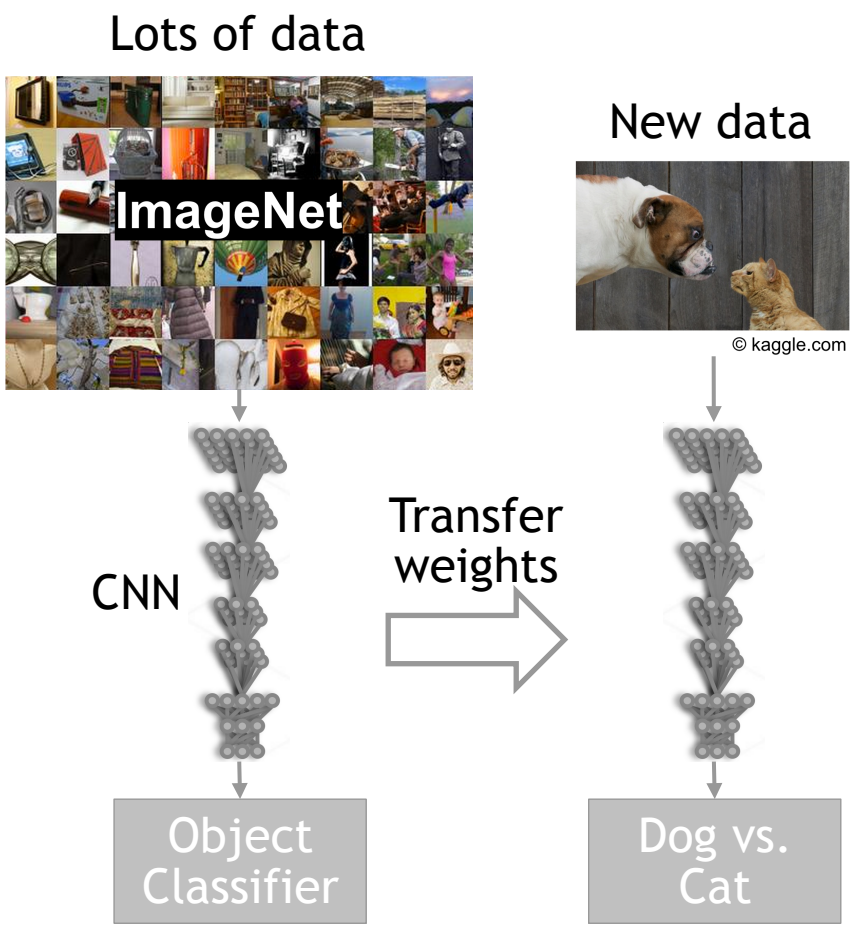


A group of young men playing a game of soccer.

Jeff Donahue et al.

Example applications

Use Case 5: Transfer Learning



- Just change a few lines in the model prototxt file

```
layer {
  name: "data"
  type: "Data"
  data_param {
    source: "ilsvrc12_train"
    …
  }
  …
}
…
layer {
  name: "fc8"
  type: "InnerProduct"
  inner_product_param {
{
    num_output: 1000
    …
  }
}
```

```
layer {
  name: "data"
  type: "Data"
  data_param: {
    source: "dogcat_train"
    …
  }
  …
}
…
layer {
  name: "fc8-dogcat"
  type: "InnerProduct"
        inner_product_param
{
    num_output: 2
    …
  }
}
```

Caffe Setup and Performance

Caffe Setup

- Tried and tested by BVLC on Ubuntu 14.04/12.04 and OS X 10.8+
- Also demonstrated to compile on RHEL, Fedora and CentOS
- Download source from https://github.com/BVLC/caffe
- Unofficial 64-bit Windows port https://github.com/niuzhiheng/caffe
- Linux setup (see http://caffe.berkeleyvision.org/installation.html)
  - Download
  - Install pre-requisites
  - Install CUDA and cuDNN for GPU acceleration
  - Compile using make

GPU Acceleration

`-gpu N` flag tells `caffe` which gpu to use

Alternatively, specify `solver_mode: GPU` in `solver.prototxt`

Tesla K40m, cuDNN v3-RC
ECC off, autoboost on

Intel® Xeon® E5-2698 v3 @ 2.60GHz, 3.6GHz turbo, (16 cores total), HT off

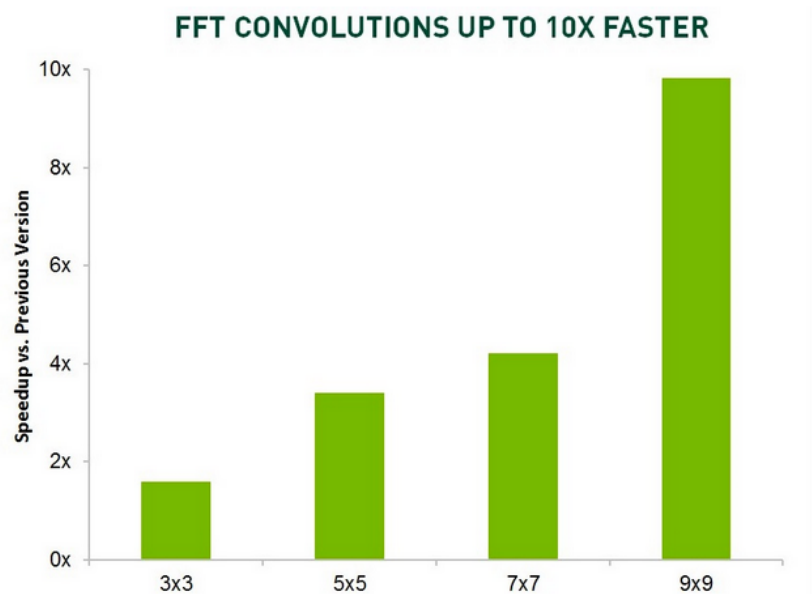
Benchmark: Train Caffenet model, 20 iterations, 256x256 images, mini-batch size 256

cuDNN integration

http://developer.nvidia.com/cudnn

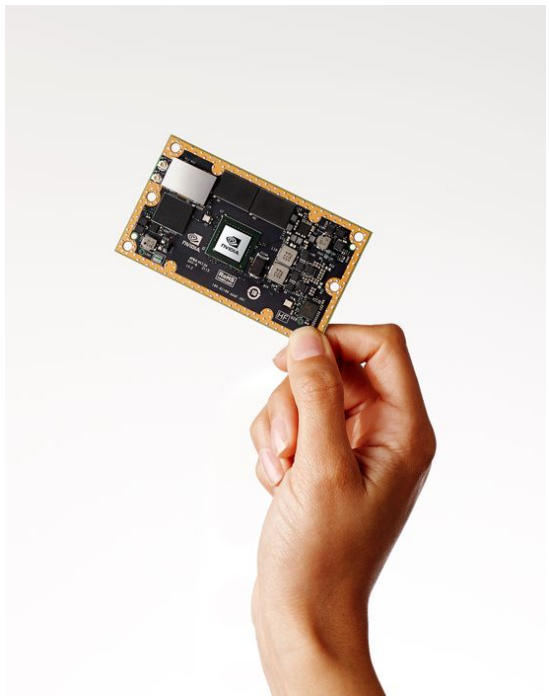
Drop-in support

Install cuDNN, uncomment `USE_CUDNN :=1` in `Makefile.config` before build



cuDNN 3 vs cuDNN 2 on Caffe, Ubuntu 14.04 LTS, Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz, 24GB RAM, GeForce Titan X

Caffe Mobile Deployment

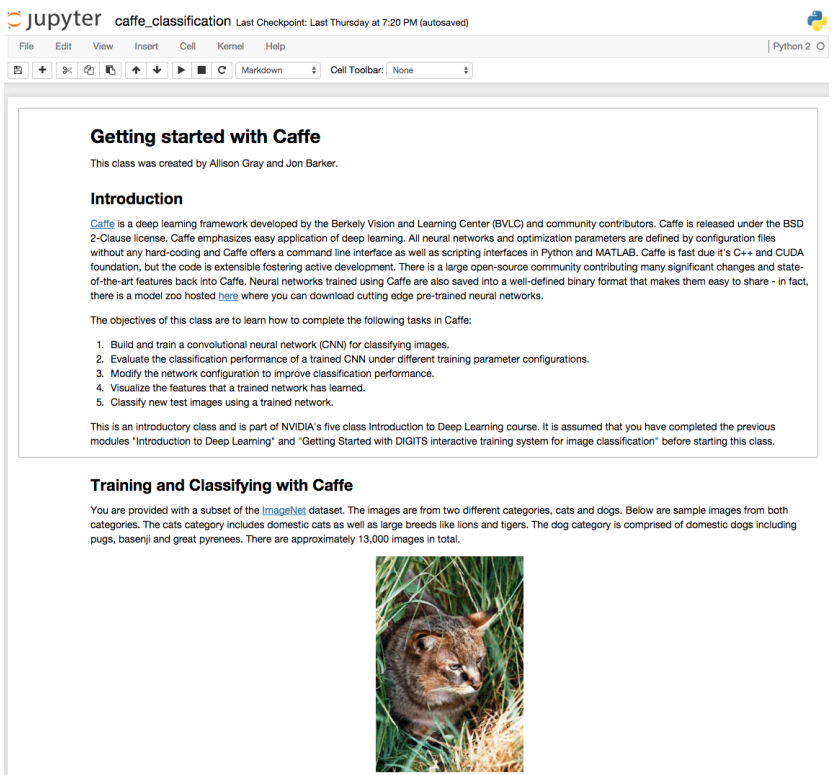


- Jetson TX1
  - Up to 340 img/s on Alexnet (FP16)
  - No need to change code
- Compile Caffe
- Copy a trained `.caffemodel` to TX1
- Run

*Source: http://petewarden.com/2014/10/25/how-to-run-the-caffe-deep-learning-vision-library-on-nvidias-jetson-mobile-gpu-board/

Hands-on lab Preview

bit.ly/dlnvlab3

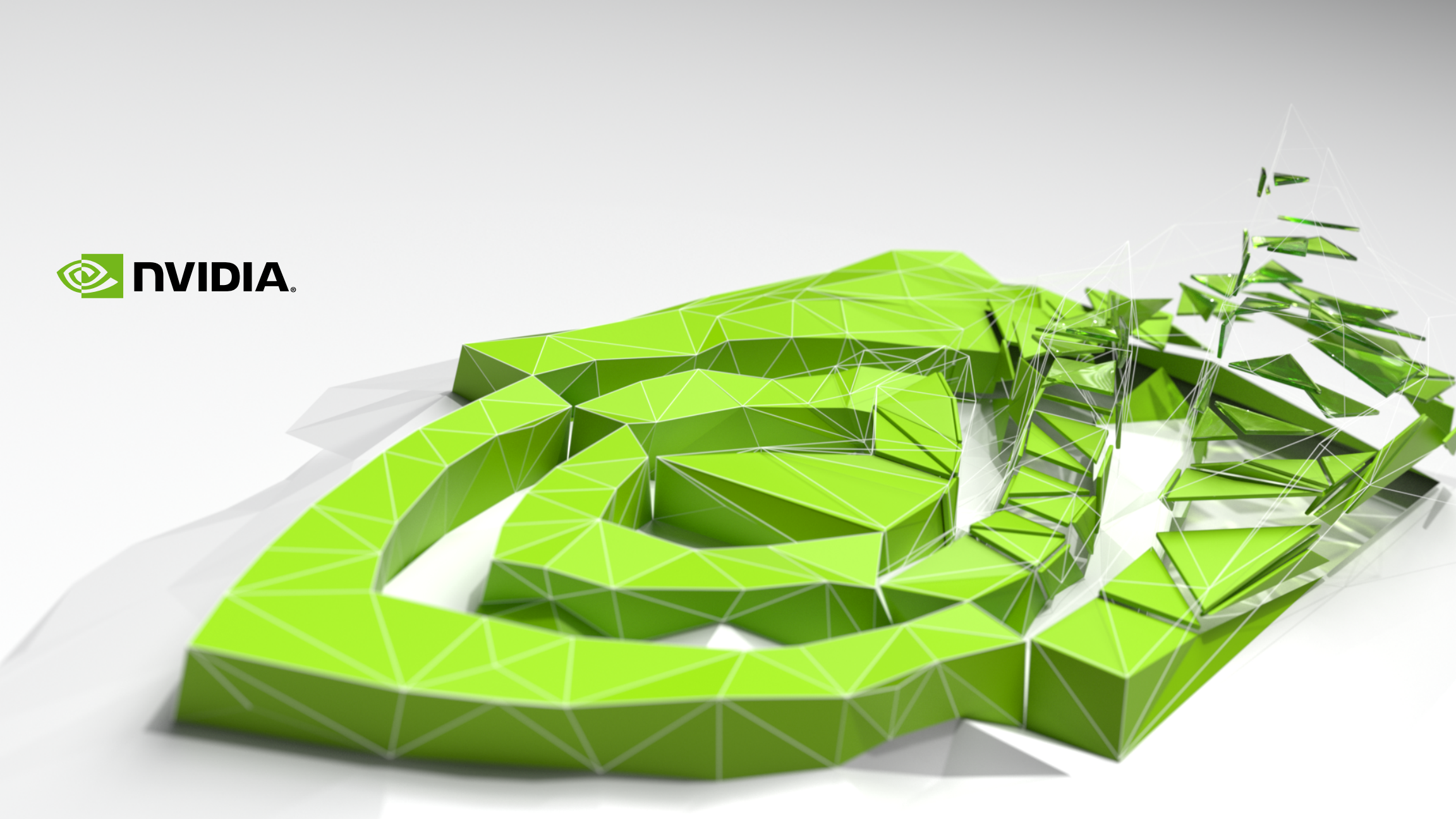


- Use data pre-processing tools
- Edit a network definition
- Train a model
- Improve classification accuracy by modifying network parameters
- Visualize trained network weights
- Deploy a model using Python

Hands-on Lab

1. Create an account at nvidia.qwiklab.com
2. Go to “LHC 2015 Workshop” lab
3. Start the lab and enjoy!

Only requires a supported browser, no NVIDIA GPU necessary!


NVIDIA