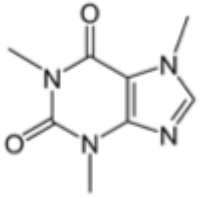
This Business of Brewing: Caffe in Practice

Maximally accurate | Maximally specific

espresso	2.23192
coffee	2.19914
beverage	1.93214
liquid	1.89367
fluid	1.85519



caffe.berkeleyvision.org

github.com/BVLC/caffe

Evan Shelhamer

from the tutorial by
Evan Shelhamer, Jeff Donahue,
Yangqing Jia, and Ross Girshick

Deep Learning, as it is executed...

What should a framework handle?

Compositional Models

Decompose the problem and code!

End-to-End Learning

Solve and check!

Vast Space of Architectures and Tasks

Define, experiment, and extend!

Frameworks

- Torch7
  - NYU
  - scientific computing framework in Lua
  - supported by Facebook

- Theano/Pylearn2
  - U. Montreal
  - scientific computing framework in Python
  - symbolic computation and automatic differentiation

- Cuda-Convnet2
  - Alex Krizhevsky
  - Very fast on state-of-the-art GPUs with Multi-GPU parallelism
  - C++ / CUDA library

Framework Comparison

- More alike than different
  - All express deep models
  - All are nicely open-source
  - All include scripting for hacking and prototyping
- No strict winners – experiment and choose the framework that best fits your work
- We like to brew our deep networks with **Caffe**

Why Caffe? In one sip…

- **Expression**: models + optimizations are plaintext schemas, not code.
- **Speed**: for state-of-the-art models and massive data.
- **Modularity**: to extend to new tasks and architectures.
- **Openness**: common code and reference models for reproducibility.
- **Community**: joint discussion, development, and modeling.

CAFFE
EXAMPLES + APPLICATIONS

Share a Sip of Brewed Models

demo.caffe.berkeleyvision.org

demo code open-source and bundled

Maximally accurate | Maximally specific

cat	1.80727
domestic cat	1.74727
feline	1.72787
tabby	0.99133
domestic animal	0.78542

Scene Recognition by MIT

Predictions:

- **Type of environment:** outdoor
- **Semantic categories:** rock_arch:0.63, arch:0.30,
- **SUN scene attributes:** rugged, natural light, dry, climbing, far-away horizon, touring, rocky, open area, warm, sand

Places CNN demo

B. Zhou et al. NIPS 14

Object Detection

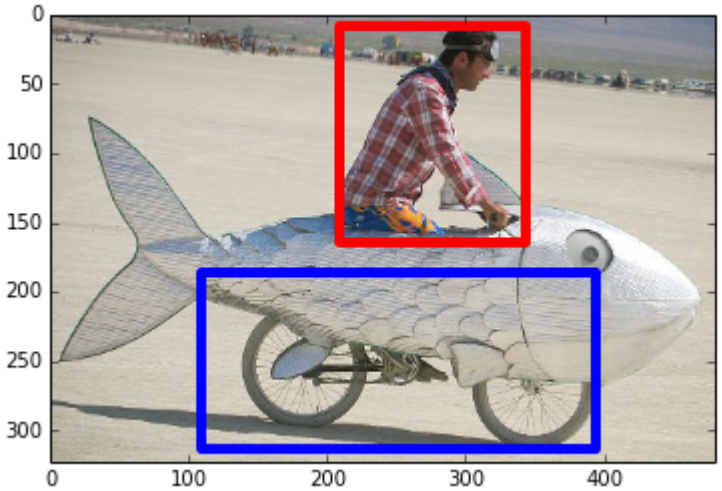
R-CNN: Regions with Convolutional Neural Networks

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.
Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR14.



Visual Style Recognition

Karayev et al. *Recognizing Image Style*. BMVC14. Caffe fine-tuning example.

Demo online at http://demo.vislab.berkeleyvision.org/ (see Results Explorer).

[Image-Style]

Other Styles:

Vintage
Long Exposure
Noir
Pastel
Macro
… and so on.

Embedded Caffe

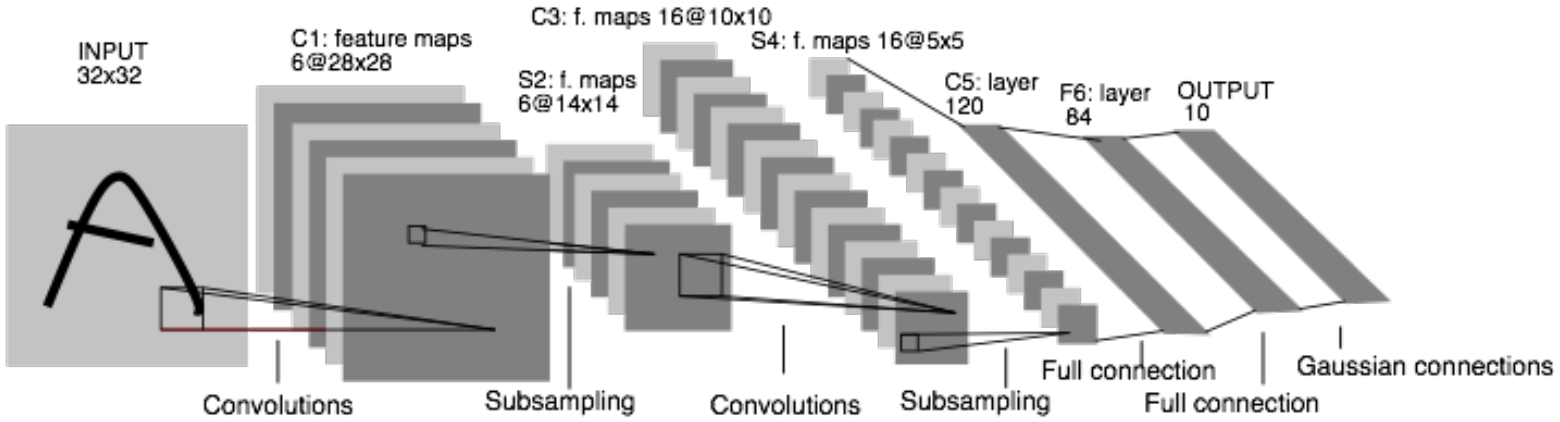
Caffe on the NVIDIA Jetson TK1 mobile board

- 10 watts of power
- inference at 35 ms per image
- how-to guide
  courtesy of Pete Warden
- cuDNN for TK1 recently released!

Conv. Nets, classic and contemporary

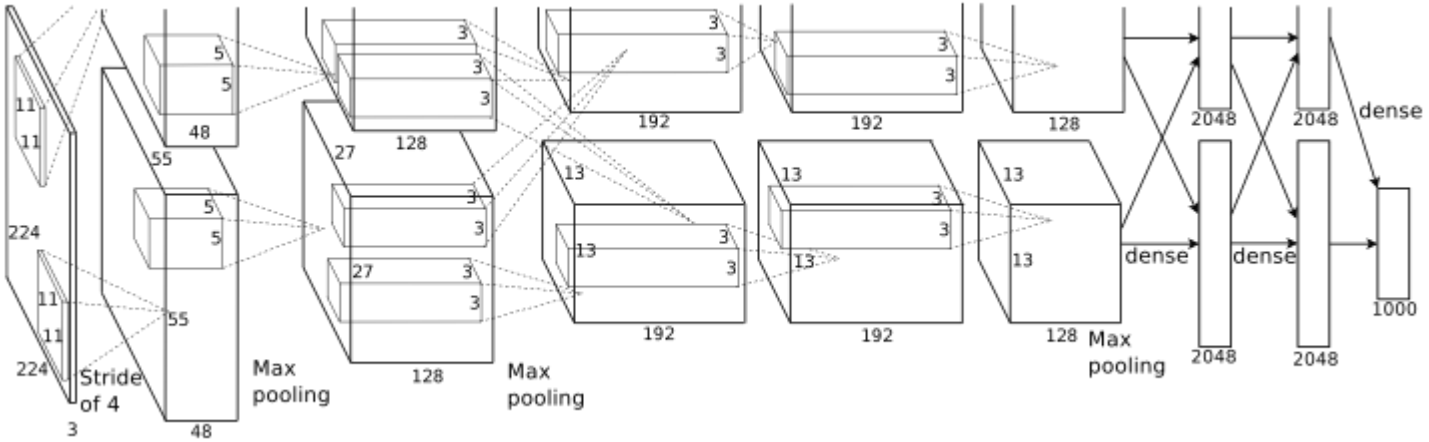
Keep the variety of models vs.
the variety of layers and connections
in mind.

Convolutional Neural Nets (CNNs): 1989



LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

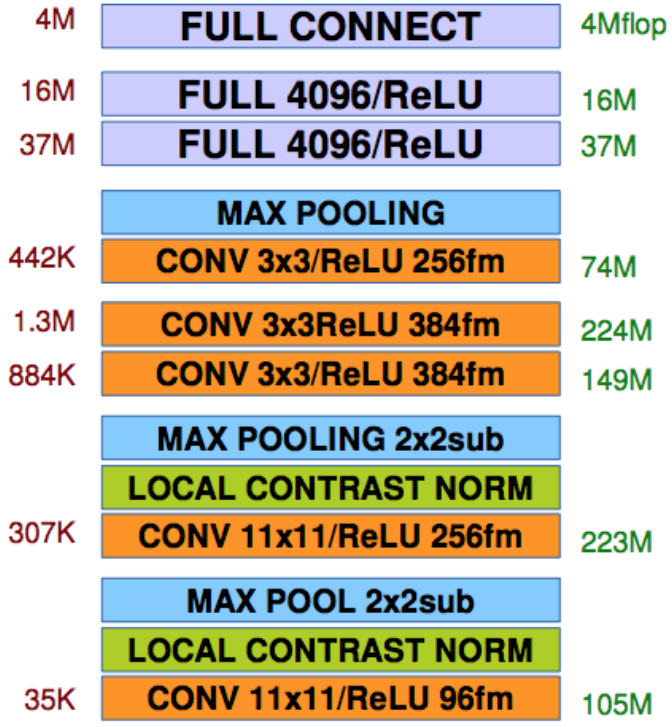
Convolutional Nets: 2012



AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12. [AlexNet]

+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

Convolutional Nets: 2012



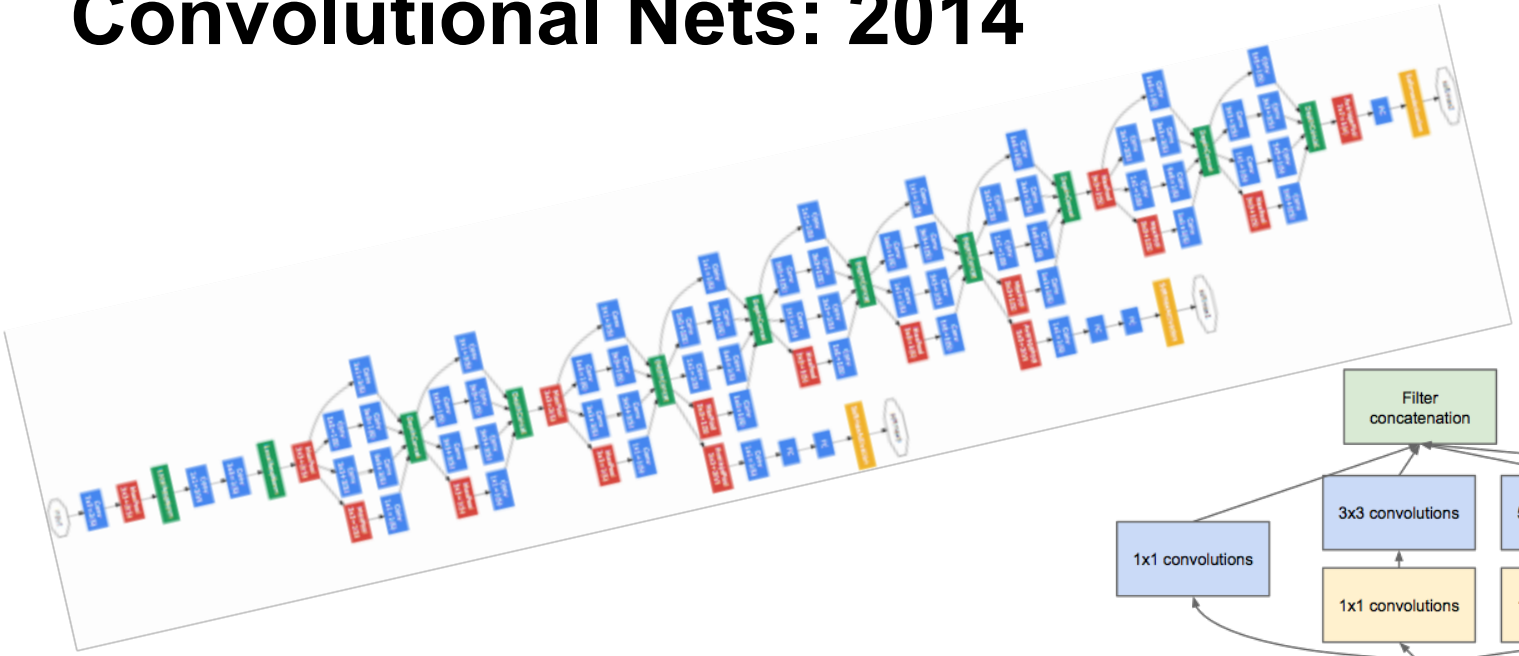
AlexNet: a layered model composed of convolution, pooling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12. [AlexNet]

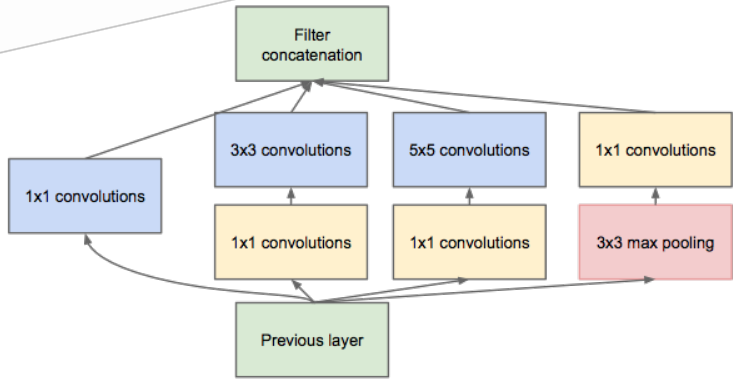
The fully-connected “FULL” layers are linear classifiers / matrix multiplications. ReLU are rectified-linear non-linearities on the output of layers.

Almost 1B FLOPs for a single image.

figure credit Y. LeCun and M.A. Ranzato, ICML ‘13 tutorial

Convolutional Nets: 2014





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 + fully-connected layers at the end

+ depth
+ data
+ dimensionality reduction

How to handle these models?

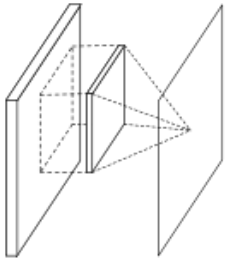
Decompose into layers.

Implement each layer type.

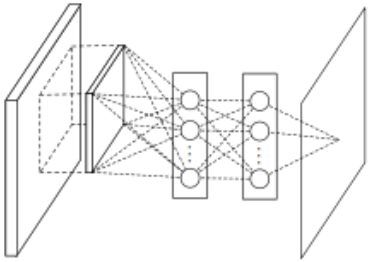
Define and experiment.

Network-in-Network / 1x1 conv

- filter with a nonlinear composition instead of a linear filter
- 1x1 convolution + nonlinearity
- reduce dimensionality + parameterization, deepen the representation
- ILSVRC model in the zoo

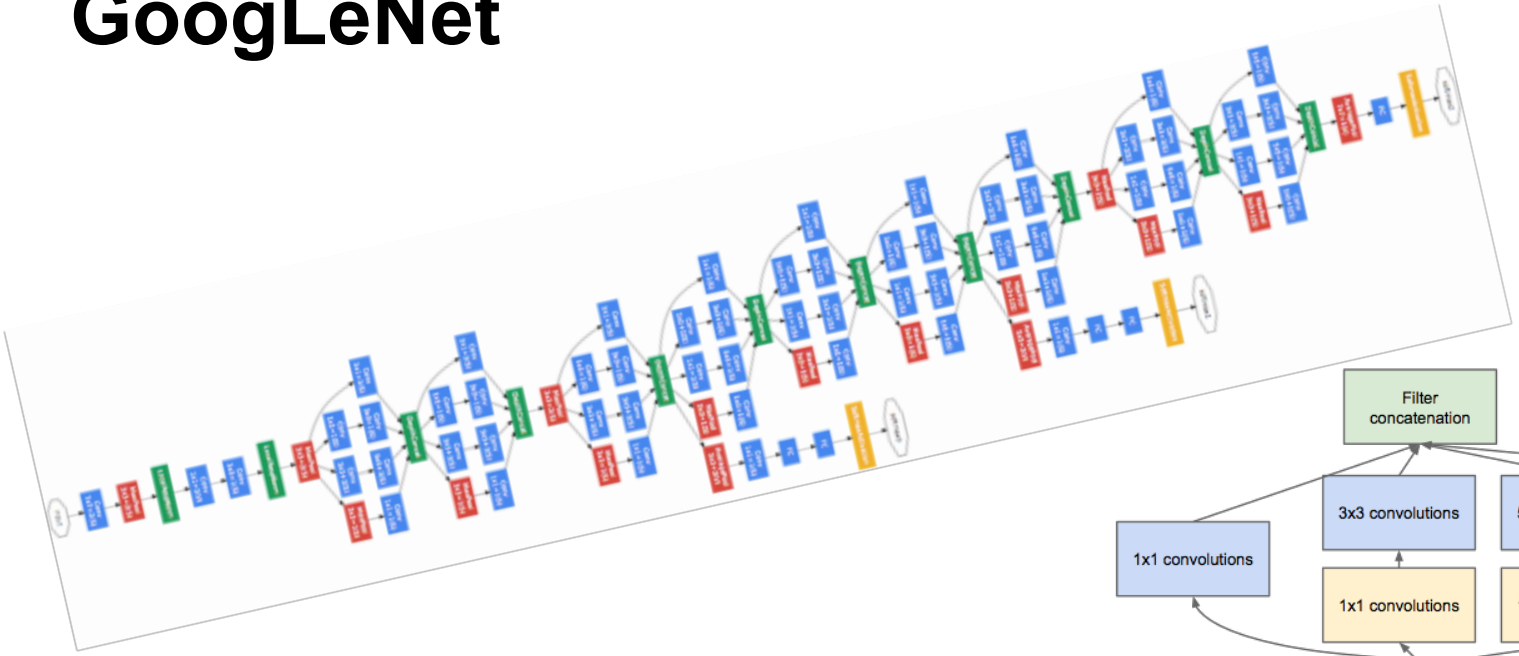


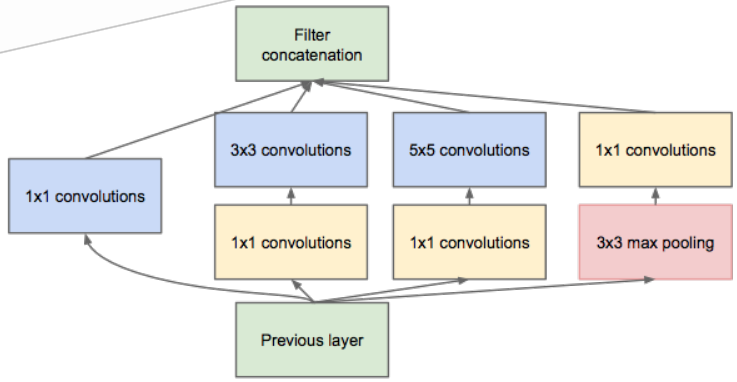
Linear Filter
CONV



NIN / MLP filter
1x1 CONV

GoogLeNet





- composition of multi-scale dimension-reduced "Inception" modules
- 1x1 conv for dimensionality reduction
- concatenation across filter scales
- multiple losses for training to depth

"Inception" module

BVLC GoogLeNet bundled in Caffe

VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- 3x3 convolution all the way down…
- fine-tuned progression of deeper models
- 16 and 19 parameter layer variations in the model zoo

This model works unreasonably well. Fine-tune it! (with CAFFE engine)

So what is Caffe?

- Layer interface
- Net to hold and operate layers
- Solver to optimize to the net

So what is Caffe?

- 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::set_mode(Caffe::GPU);`

Prototype

Training

Deployment

All with essentially the same code!

Caffe is a Community

project pulse

BVLC / caffe

Unwatch 468 | Unstar 2,490 | Fork 1,392

January 22, 2015 – February 22, 2015

Period: **1 month**

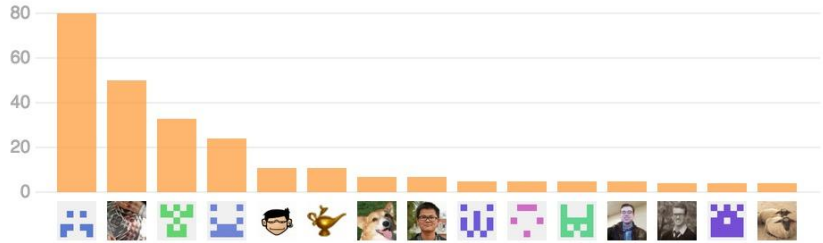
Overview

60 Active Pull Requests

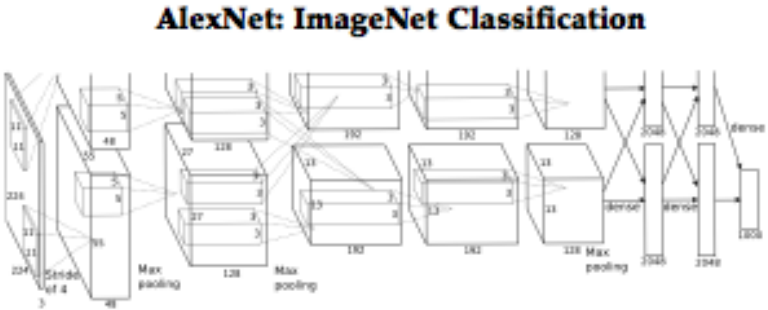
126 Active Issues

37 Merged Pull Requests	**23** Proposed Pull Requests	**104** Closed Issues	**22** New Issues

Excluding merges, **36 authors** have pushed **119 commits** to master and **285 commits** to all branches. On master, **309 files** have changed and there have been **17,564 additions** and **9,143 deletions**.



Reference Models



Caffe offers the

- model definitions
- optimization settings
- pre-trained weights

so you can start right away.

The BVLC models are licensed for unrestricted use.

Open Model Collection

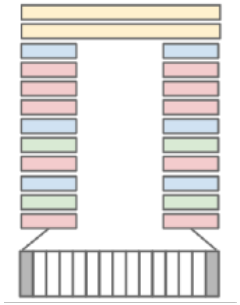
The Caffe Model Zoo

- open collection of deep models to share innovation
    - VGG ILSVRC14 + Devil models in the zoo
    - Network-in-Network / CCCP model in the zoo
    - MIT Places scene recognition model in the zoo
- help disseminate and reproduce research
- bundled tools for loading and publishing models

Share Your Models! with your citation + license of course

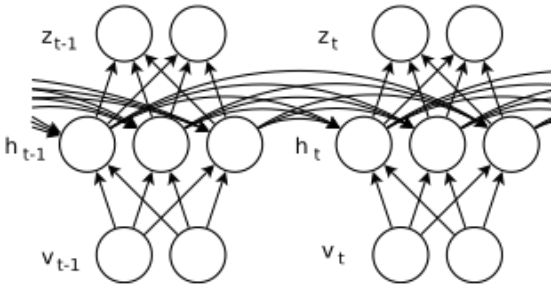
Architectures

DAGs
multi-input
multi-task



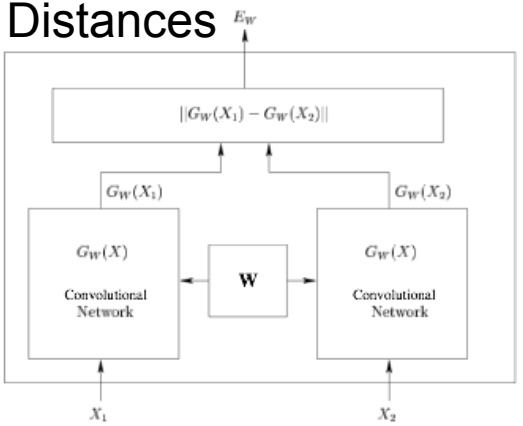
[Karpathy14]

Weight Sharing
Recurrent (RNNs)
Sequences



[Sutskever13]

Siamese Nets
Distances

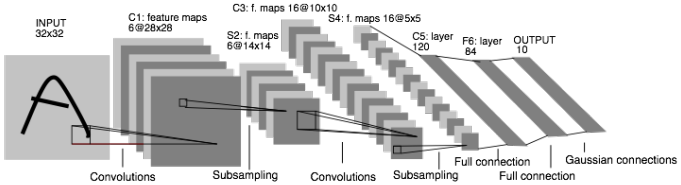


[Chopra05]

Define your own model from our catalogue of layers types and start learning then extend with custom layers and parameters.

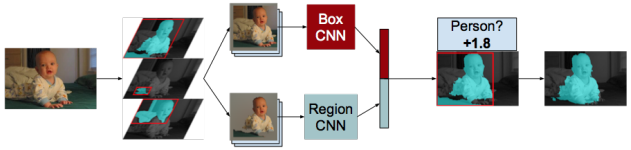
DAG

Many current deep models have linear structure

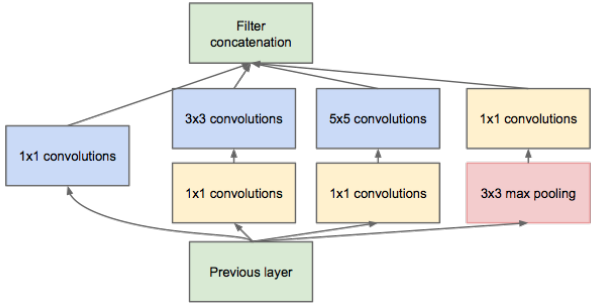


but Caffe nets can have any directed acyclic graph (DAG) structure.

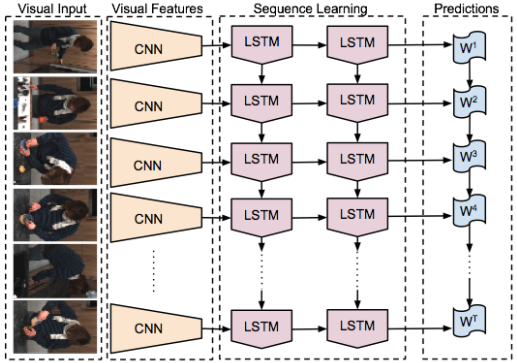
Define `bottoms` and `tops` and Caffe will connect the net.



SDS two-stream net



GoogLeNet Inception Module



LRCN joint vision-sequence model

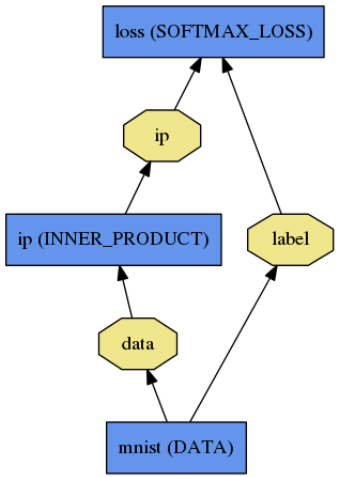
CAFFE INTRO

Net

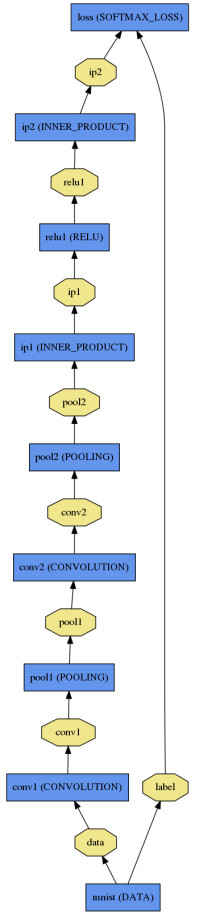
- A network is a set of layers connected as a DAG:

```
name: "dummy-net"
layer { name: "data" …}
layer { name: "conv" …}
layer { name: "pool" …}
    … more layers …
layer { name: "loss" …}
```

- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – a an array interface



LogReg ↑

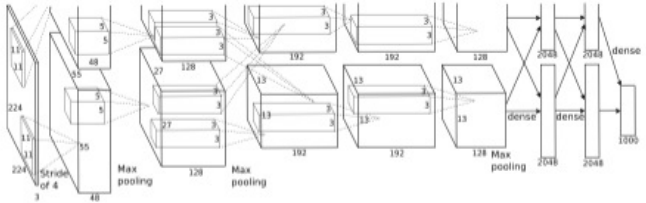


LeNet ⟶

ImageNet, Krizhevsky 2012 ⟶

Forward / Backward the essential Net computations

Forward: inference $f_W(x)$



"espresso" + loss

$\nabla f_W(x)$ Backward: learning

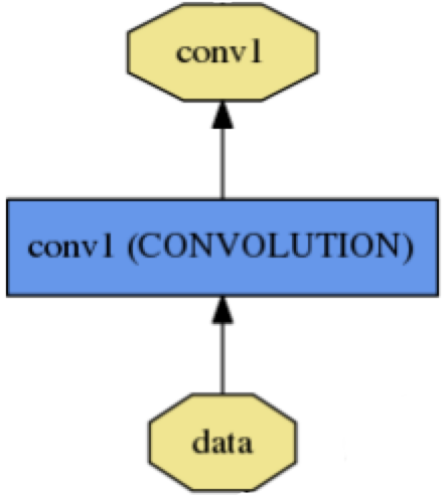
Caffe models are complete machine learning systems for inference and learning. The computation follows from the model definition. Define the model and run.

Layer

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

name, type, and the connection structure (input blobs and output blobs)

layer-specific parameters



- Every layer type defines
  - **Setup**
  - **Forward**
  - **Backward**

* Nets + Layers are defined by protobuf schema

Protobuf

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

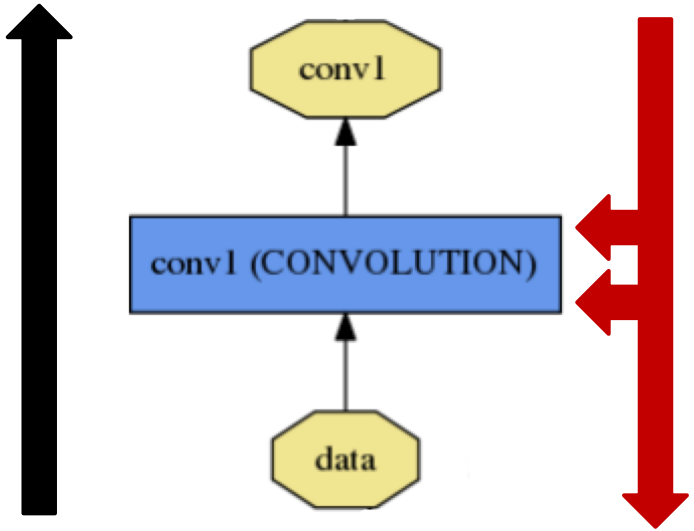
- Nets + Layers are defined by protobuf schema
- Net / Layer / Solver schema is defined by **caffe.proto** in src/caffe/proto
- Consult this definition for all the configuration options (inline comments explain it)

Layer Protocol

Setup: run once for initialization.

Forward: make output given input.

Backward: make gradient of output
- w.r.t. bottom
- w.r.t. parameters (if needed)



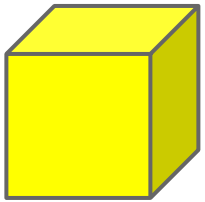
Compositional Modeling
The Net's forward and backward passes are composed of the layers' steps.

Layer Development Checklist

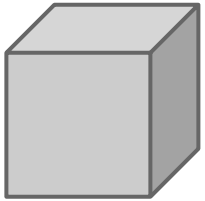
Blob

Blobs are 4-D arrays for storing and communicating information.

- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



Data
*N*umber x *K* Channel x *H*eight x *W*idth
256 x 3 x 227 x 227 for ImageNet train input

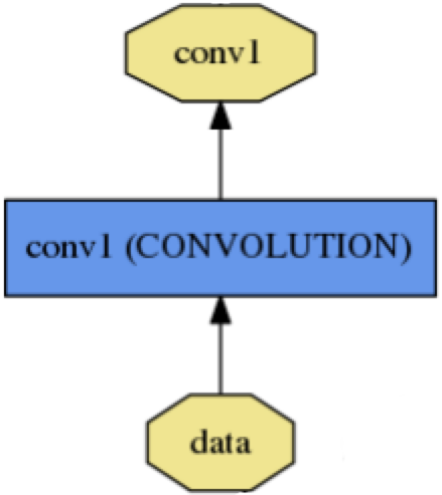


Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1

Parameter: Convolution Bias
96 x 1 x 1 x 1 for CaffeNet conv1

```
name: "conv1"
type: "Convolution"
bottom: "data"
top: "conv1"
… definition …
```

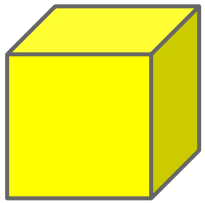
top
blob



bottom
blob

Blob

Blobs provide a unified memory interface.

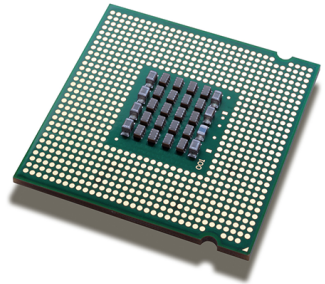


Reshape(num, channel, height, width)
- declare dimensions
- make *SyncedMem* -- but only lazily allocate

cpu_data(), mutable_cpu_data()
- host memory for CPU mode

gpu_data(), mutable_gpu_data()
- device memory for GPU mode

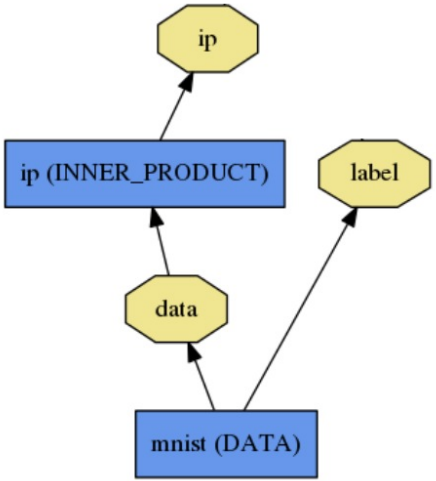
{cpu,gpu}_diff(), mutable_{cpu,gpu}_diff()
- derivative counterparts to data methods
- easy access to data + diff in forward / backward



SyncedMem
allocation + communication

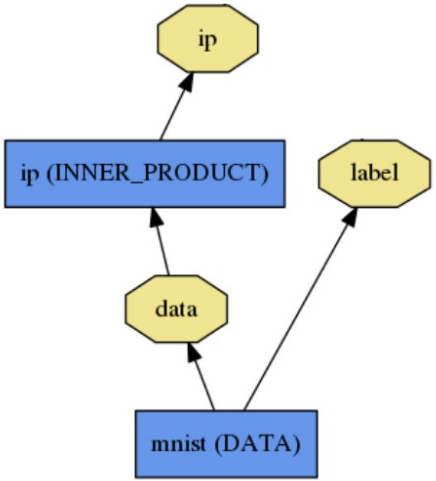
Loss

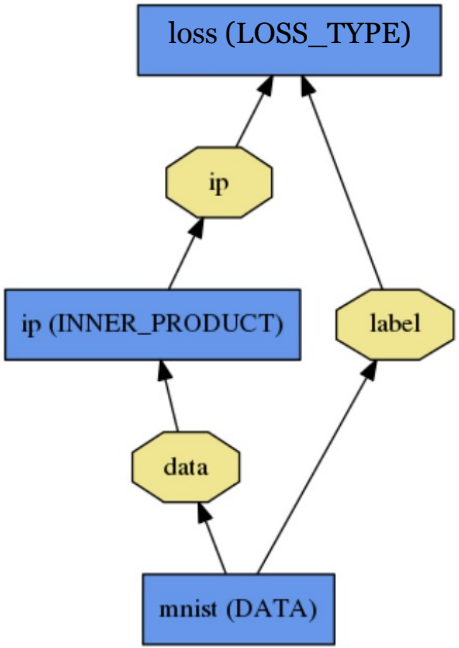
What kind of model is this?



Loss

What kind of model is this?





Classification
`SoftmaxWithLoss`
`HingeLoss`

Linear Regression
`EuclideanLoss`

Attributes / Multiclassification
`SigmoidCrossEntropyLoss`

Others…

New Task
`NewLoss`

Who knows! Need a **loss function**.

Multiple Losses

- Nets can have as many Losses as you can handle
- Reconstruction and Classification
  - define both as usual
  - pay attention to weight!
  - learn jointly / sequentially

```
layer {
  name: "recon-loss"
  type: "EuclideanLoss"
  bottom: "reconstructions"
  bottom: "data"
  top: "recon-loss"
  loss_weight: 0.01
}

layer {
  name: "class-loss"
  type: "SoftmaxWithLoss"
  bottom: "class-preds"
  bottom: "class-labels"
  top: "class-loss"
}
```

Solver

- **Solver** optimizes the network weights to minimize the loss over the data.
- Coordinates forward / backward, weight updates, and scoring.
  - `Init()`
  - `Solve(), Step()`
  - `ComputeUpdateValue()`
  - `Snapshot(), Restore()`
  - `Test()`

Solver

Computes parameter update from

- stochastic error gradient
- regularization gradient
- particulars to each solving method

Solving: Training a Net

Optimization like model definition is configuration.

```
train_net: "lenet_train.prototxt"
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
max_iter: 10000
snapshot_prefix: "lenet_snapshot"
```

All you need to run things on the GPU.

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

Stochastic Gradient Descent (SGD) + momentum · Adaptive Gradient (ADAGRAD) · Nesterov's Accelerated Gradient (NAG)

Solver Showdown: MNIST Autoencoder

AdaGrad

```
I0901 13:36:30.007884 24952 solver.cpp:232] Iteration 65000, loss = 64.1627
I0901 13:36:30.007922 24952 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:36:33.019305 24952 solver.cpp:289] Test loss: 63.217
I0901 13:36:33.019356 24952 solver.cpp:302]     Test net output #0: cross_entropy_loss = 63.217 (* 1 = 63.217 loss)
I0901 13:36:33.019773 24952 solver.cpp:302]     Test net output #1: l2_error = 2.40951
```

SGD

```
I0901 13:35:20.426187 20072 solver.cpp:232] Iteration 65000, loss = 61.5498
I0901 13:35:20.426218 20072 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:35:22.780092 20072 solver.cpp:289] Test loss: 60.8301
I0901 13:35:22.780138 20072 solver.cpp:302]     Test net output #0: cross_entropy_loss = 60.8301 (* 1 = 60.8301 loss)
I0901 13:35:22.780146 20072 solver.cpp:302]     Test net output #1: l2_error = 2.02321
```

Nesterov

```
I0901 13:36:52.466069 22488 solver.cpp:232] Iteration 65000, loss = 59.9389
I0901 13:36:52.466099 22488 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:36:55.068370 22488 solver.cpp:289] Test loss: 59.3663
I0901 13:36:55.068410 22488 solver.cpp:302]     Test net output #0: cross_entropy_loss = 59.3663 (* 1 = 59.3663 loss)
I0901 13:36:55.068418 22488 solver.cpp:302]     Test net output #1: l2_error = 1.79998
```

Recipe for Brewing

- Convert the data to Caffe-format
  - lmdb, leveldb, hdf5 / .mat, list of images, etc.
- Define the Net
- Configure the Solver
- caffe train -solver solver.prototxt -gpu 0

- Examples are your friends
  - `caffe/examples/mnist,cifar10,imagenet`
  - `caffe/examples/*.ipynb`
  - `caffe/models/*`

(Examples)

Logistic Regression

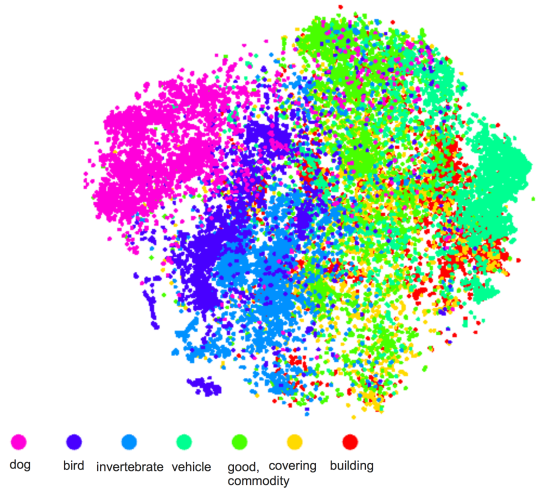
Learn LeNet on MNIST

FINE-TUNING

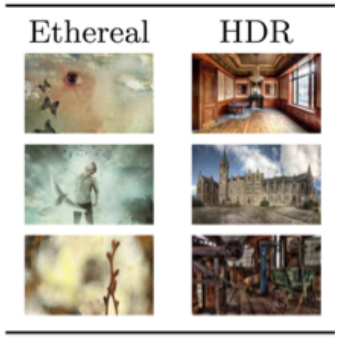
Fine-tuning Transferring learned weights to kick-start models

Take a pre-trained model and fine-tune to new tasks

[DeCAF] [Zeiler-Fergus] [OverFeat]



Your Task



Style Recognition

© kaggle.com

Dogs vs. Cats
top 10 in 10 minutes

From ImageNet to Style

Simply change a few lines in the layer definition

```
layers {
  name: "data"
  type: DATA
  data_param {
    source: "ilsvrc12_train_leveldb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layers {
  name: "fc8"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 1000
    ...
  }
}
```

```
layers {
  name: "data"
  type: DATA
  data_param {
    source: "style_leveldb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layers {
  name: "fc8-style"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 20
    ...
  }
}
```

new name = new params

Input:
A different source

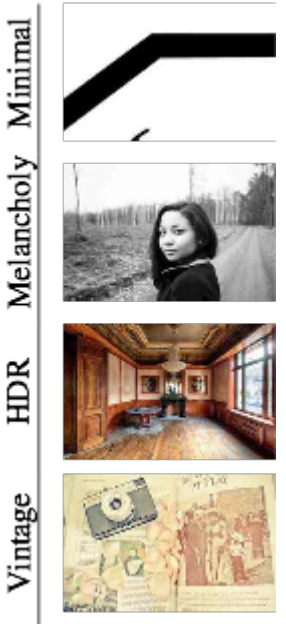
Last Layer:
A different classifier

From ImageNet to Style

```
> caffe train -solver models/finetune_flickr_style/solver.prototxt
              -weights bvlc_reference_caffenet.caffemodel
```

Step-by-step in pycaffe:

```
pretrained_net = caffe.Net(
  "net.prototxt", "net.caffemodel")
solver = caffe.SGDSolver("solver.prototxt")
solver.net.copy_from(pretrained_net)
solver.solve()
```



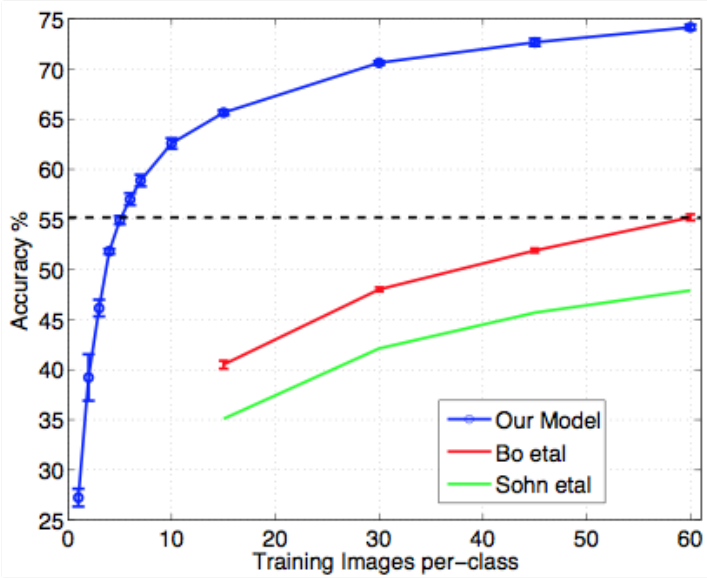
When to Fine-tune?

Almost Always!

- More robust optimization – good initialization helps
- Needs less data
- Faster learning

State-of-the-art results in

- recognition
- detection
- segmentation



[Zeiler-Fergus]

Fine-tuning Tricks

Learn the last layer first

- Caffe layers have local learning rates: `param { lr_mult: 1 }`
- 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

Do net surgery

(Example)

Fine-tuning from ImageNet to Style

NOW ROASTING

- Pythonification done
- Fully Convolutional Networks arxiv + PR
- Sequences arxiv + PR
- Gradient Accumulation #1663
- cuDNN v2 #1731
- Parallelism experimental
- More
  - N-D Data + Operations #1872
  - Sparse Embeddings #1872
  - Deconvolution done
  - ...

Pythonification #1703

Python Layer

- layer prototyping and ease of expression
- call Python from C++, C++ from Python, and around we go

Complete instrumentation in Python

- data preparation
- solving
- inference
- model definition still to come in #1733

Jon Long

Fully Convolutional Network: FCN

A framework for pixel prediction by conv. net

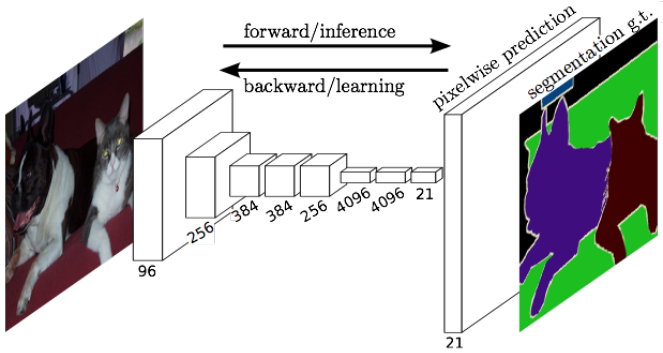
applied to semantic segmentation

- 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



Jon Long & Evan Shelhamer

Sequences

Recurrent Net RNN and Long Short Term Memory LSTM are sequential models

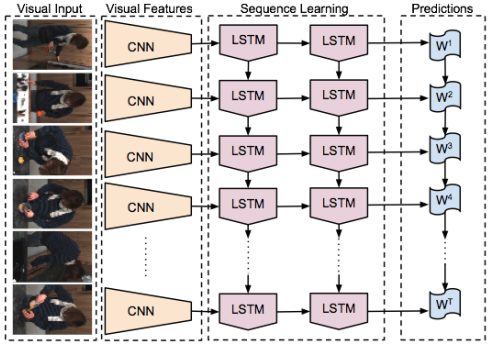
- video
- language
- dynamics

learned by back-propagation through time.

LRCN: Long-term Recurrent Convolutional Network

- activity recognition
- image captioning
- video captioning

[arXiv](#) and [web page & PR](#)



A group of young men playing a game of soccer.

Jeff Donahue et al.

Gradient Accumulation #1663

- decouple computational and learning mini-batch size
- tune optimization independently of resource constraints
- conserve memory

...and share convolution buffers to save memory #1291

...and turn off the testing net too

Jon Long & Sergio Guadarrama

BREWING ADVICE

- where to go for help
- debugging
- saving time
- saving memory

Where to Go for Help

- DIY deep learning for vision tutorial
- tutorial documentation
- caffe-users group
- gitter.im Caffe chat

Debugging

- build with `DEBUG := 1` in the Makefile.config
  - turns on checks and informative logging
  - but re-build once you're done for full speed
- set `debug_info: true` in the solver.prototxt for more output to debug learning
- load the net in pycaffe `net = caffe.Net("model.prototxt", "model.caffemodel")` and inspect its data and diffs after `net.forward()` and `net.backward()`.

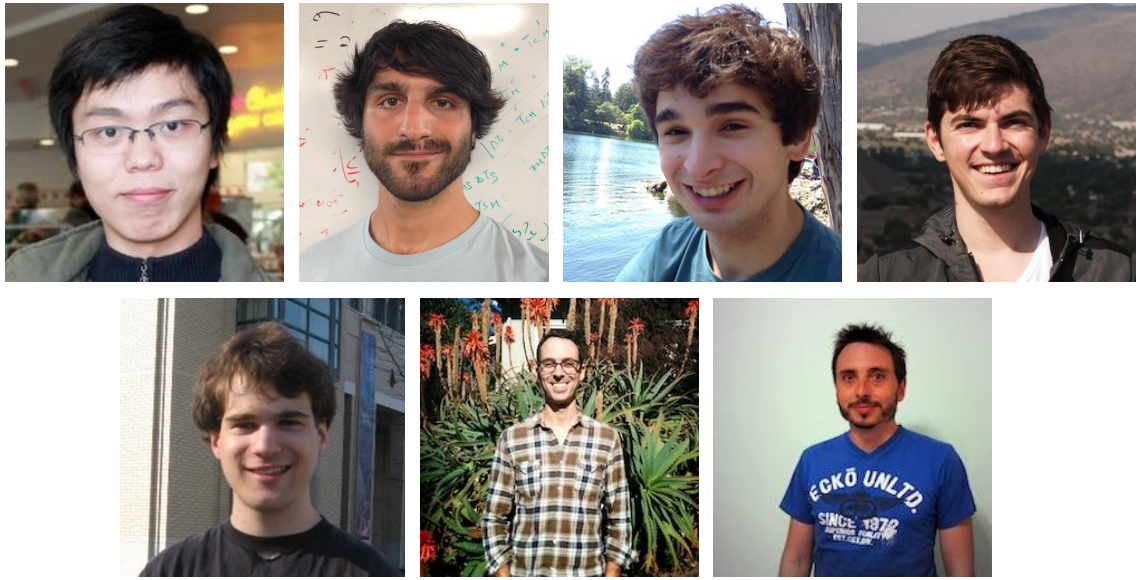
Save Time

- call `caffe time` to benchmark
- try the `CUDNN` and `CAFFE` engines
  - CUDNN faster for AlexNet, GoogLeNet
  - CAFFE faster for VGG
- set `test_initialization: false` in solver.prototxt to skip the initial testing run
- set hyper-params on small data!

Save Memory

- share convolution buffers to save memory #1291. memory scales with max conv. size instead of sum
- gradient accumulation #1663. decouple computation batch size from learning for fixed memory training.
- turn off the testing net
  - set snapshot interval for saving models
  - but remove the `test_*` lines from the solver.prototxt
  - no automatic testing -- no testing memory

Thanks to the Caffe crew



Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev
Jonathan Long, Ross Girshick, Sergio Guadarrama

and our open source contributors!

...plus the cold-brew

Acknowledgements

Thank you to the Berkeley Vision and Learning Center Sponsors.

Thank you to NVIDIA
for GPU donation and
collaboration on cuDNN.

Thank you to A9 and AWS
for a research grant for Caffe dev
and reproducible research.

Thank you to our 75+
open source contributors
and vibrant community.

References

[DeCAF] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. ICML, 2014.

[R-CNN] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

[Zeiler-Fergus] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. ECCV, 2014.

[LeNet] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. IEEE, 1998.

[AlexNet] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. NIPS, 2012.

[OverFeat] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. ICLR, 2014.

[Image-Style] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, H. Winnemoeller. Recognizing Image Style. BMVC, 2014.

[Karpathy14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. CVPR, 2014.

[Sutskever13] I. Sutskever. Training Recurrent Neural Networks.
PhD thesis, University of Toronto, 2013.

[Chopra05] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. CVPR, 2005.