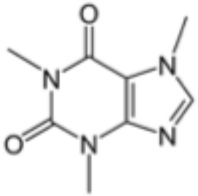
DIY Deep Learning for Vision: the Caffe framework

Maximally accurate	Maximally specific
espresso	2.23192
coffee	2.19914
beverage	1.93214
liquid	1.89367
fluid	1.85519



caffe.berkeleyvision.org

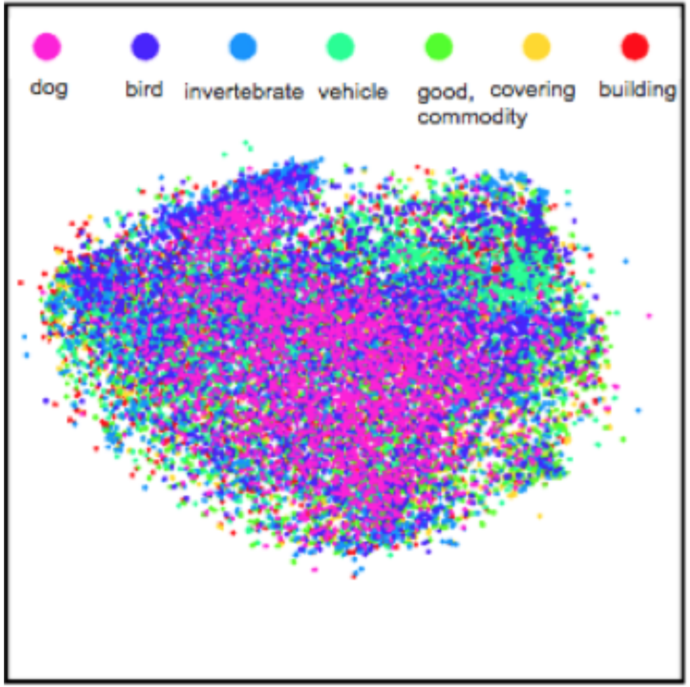
github.com/BVLC/caffe

Evan Shelhamer

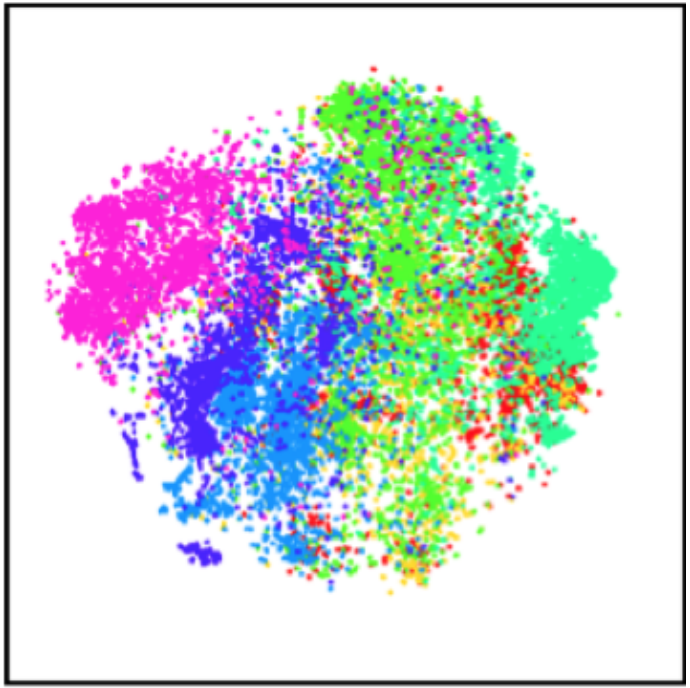
adapted from the Caffe tutorial with
Jeff Donahue, Yangqing Jia, and Ross Girshick.

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features



Low-level: $Pool_1$

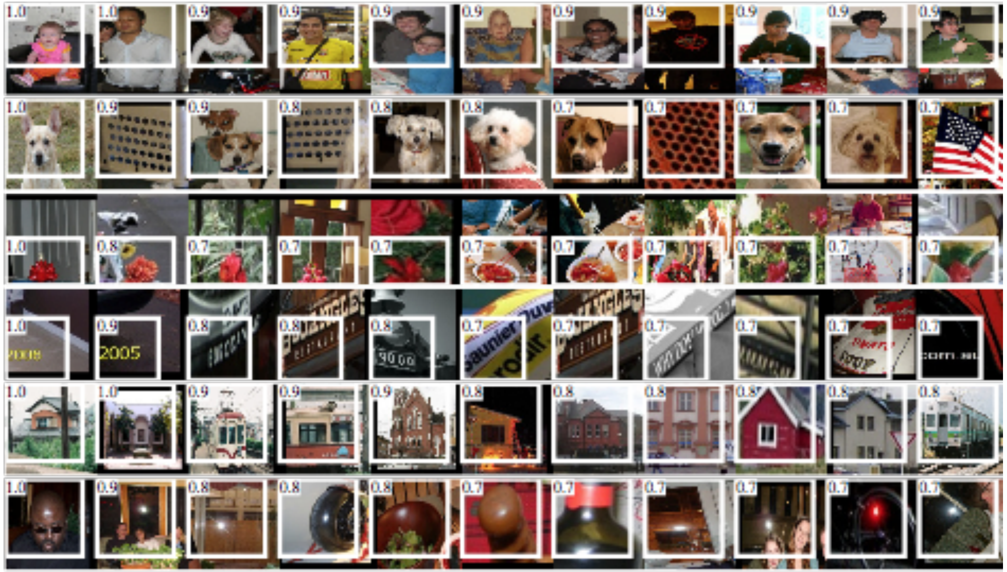


High-level: FC_6

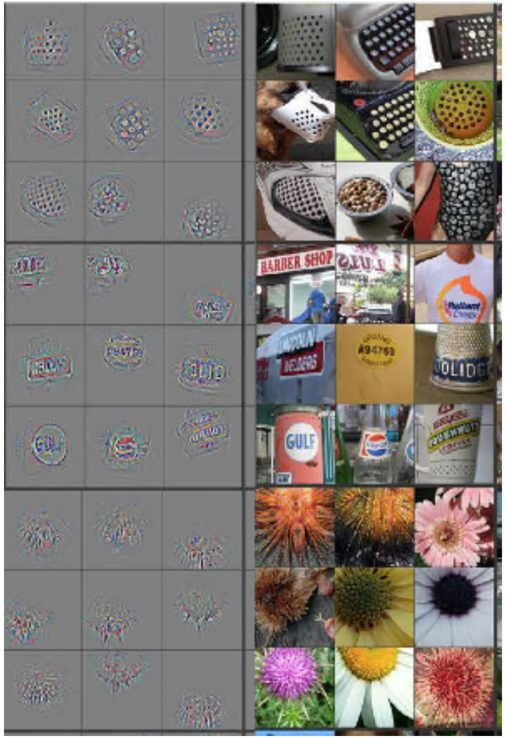
Classes separate in the deep representations and transfer to many tasks.
[DeCAF] [Zeiler-Fergus]

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features



Maximal activations of $pool_5$ units [R-CNN]



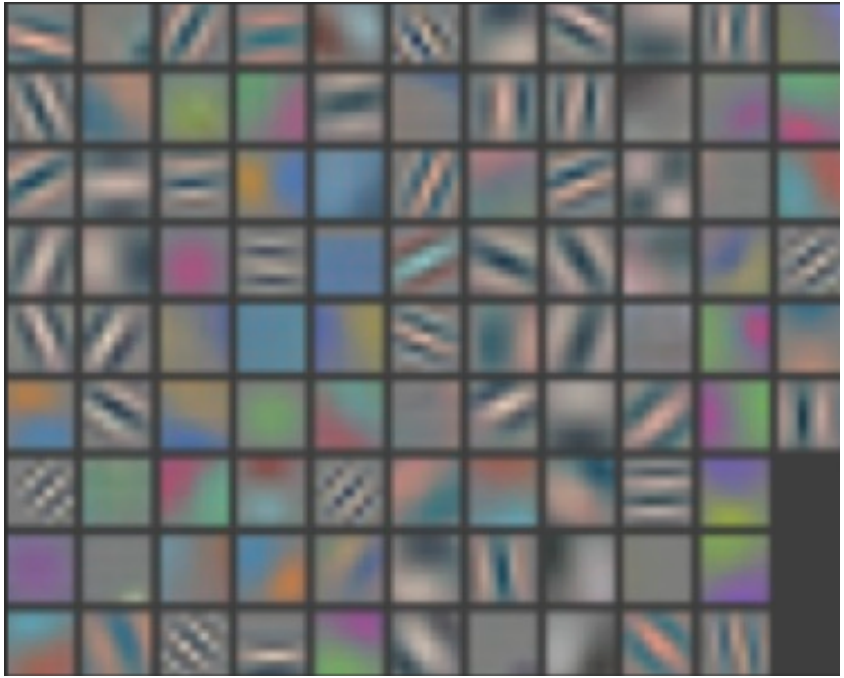
$conv_5$ DeConv visualization
[Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features

image patches that strongly activate 1st layer filters



1st layer filters

[Zeiler-Fergus]

What is Deep Learning?

Compositional Models
Learned End-to-End

What is Deep Learning?

Compositional Models
Learned End-to-End

Hierarchy of Representations

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

concrete $\xrightarrow[\text{learning}]{}$ abstract

What is Deep Learning?

Compositional Models
Learned End-to-End

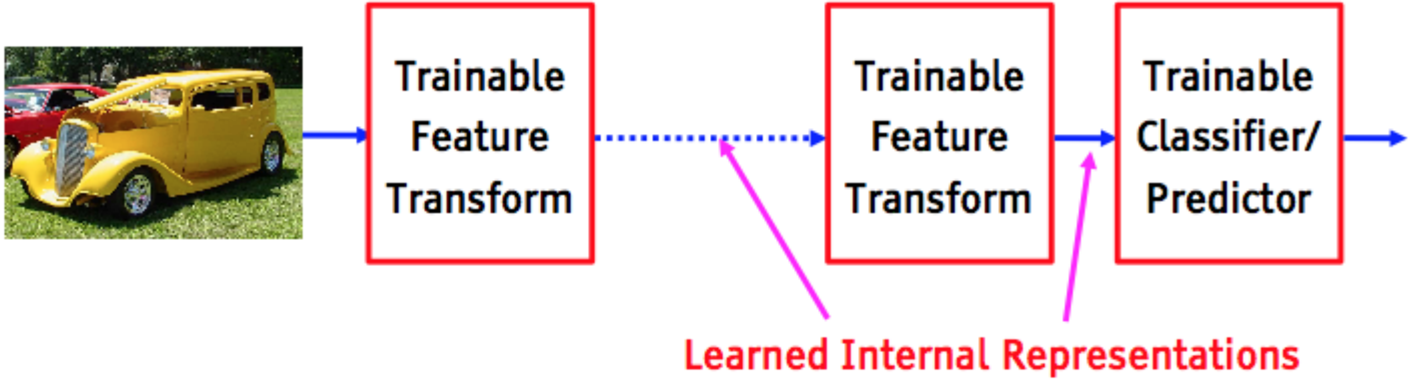


figure credit Yann LeCun, ICML '13 tutorial

What is Deep Learning?

Compositional Models
Learned End-to-End

Back-propagation: take the gradient of the model layer-by-layer by the chain rule to yield the gradient of all the parameters.

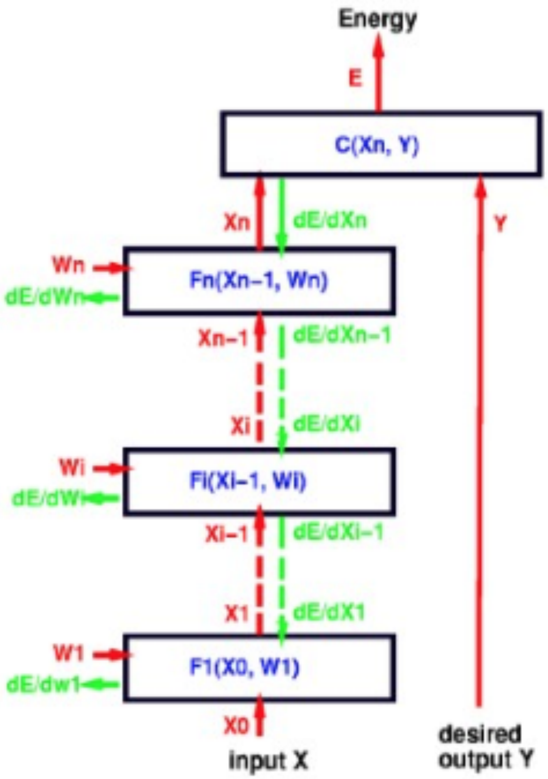
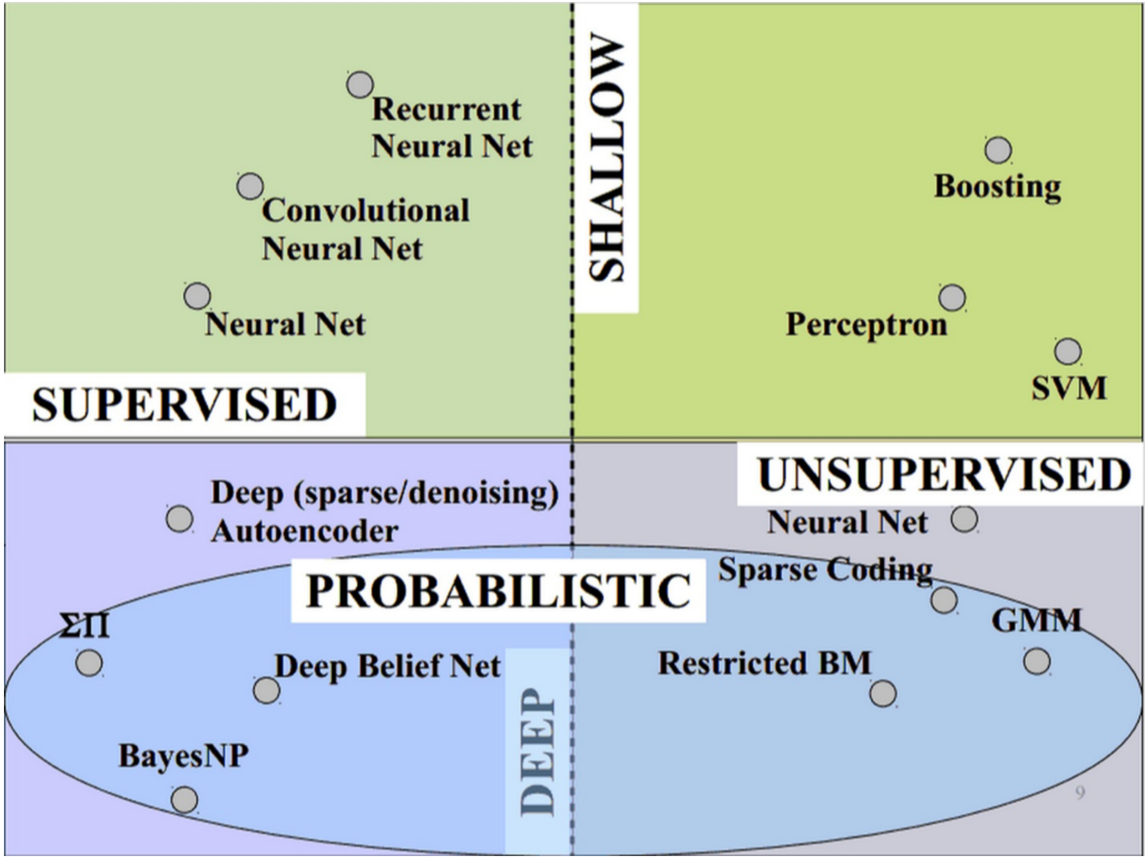


figure credit Yann LeCun, ICML '13 tutorial

What is Deep Learning?



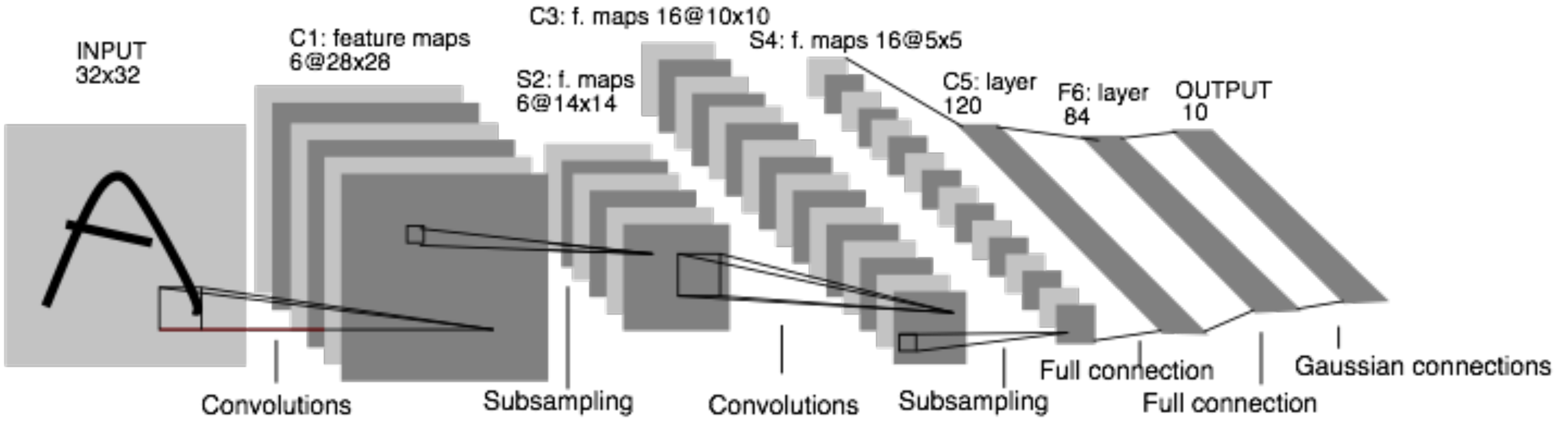
Vast space of models!

Caffe models are loss-driven:

- supervised
- unsupervised

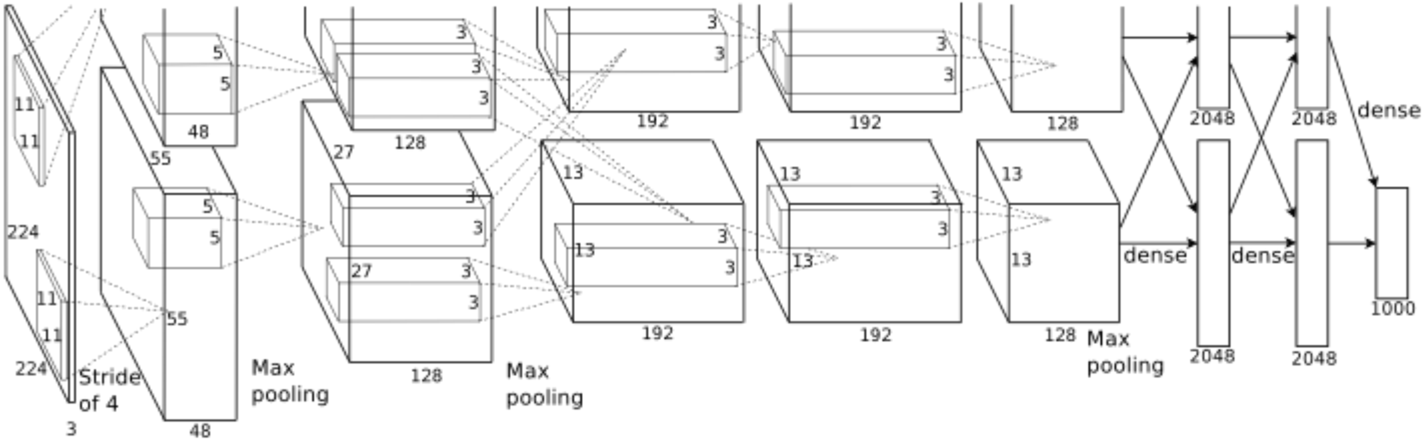
slide credit Marc'aurelio Ranzato, CVPR '14 tutorial.

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]

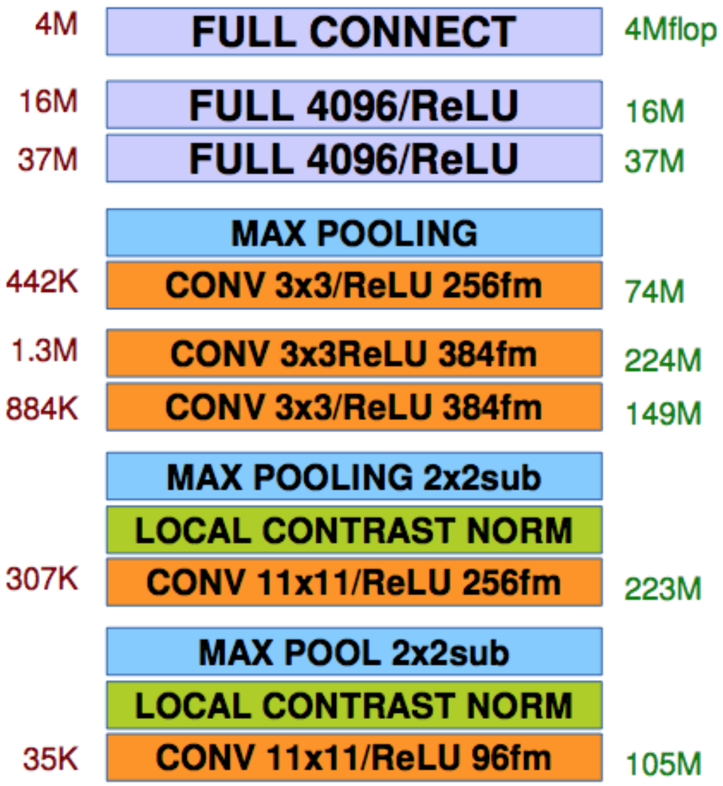
Convolutional Nets: 2012



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

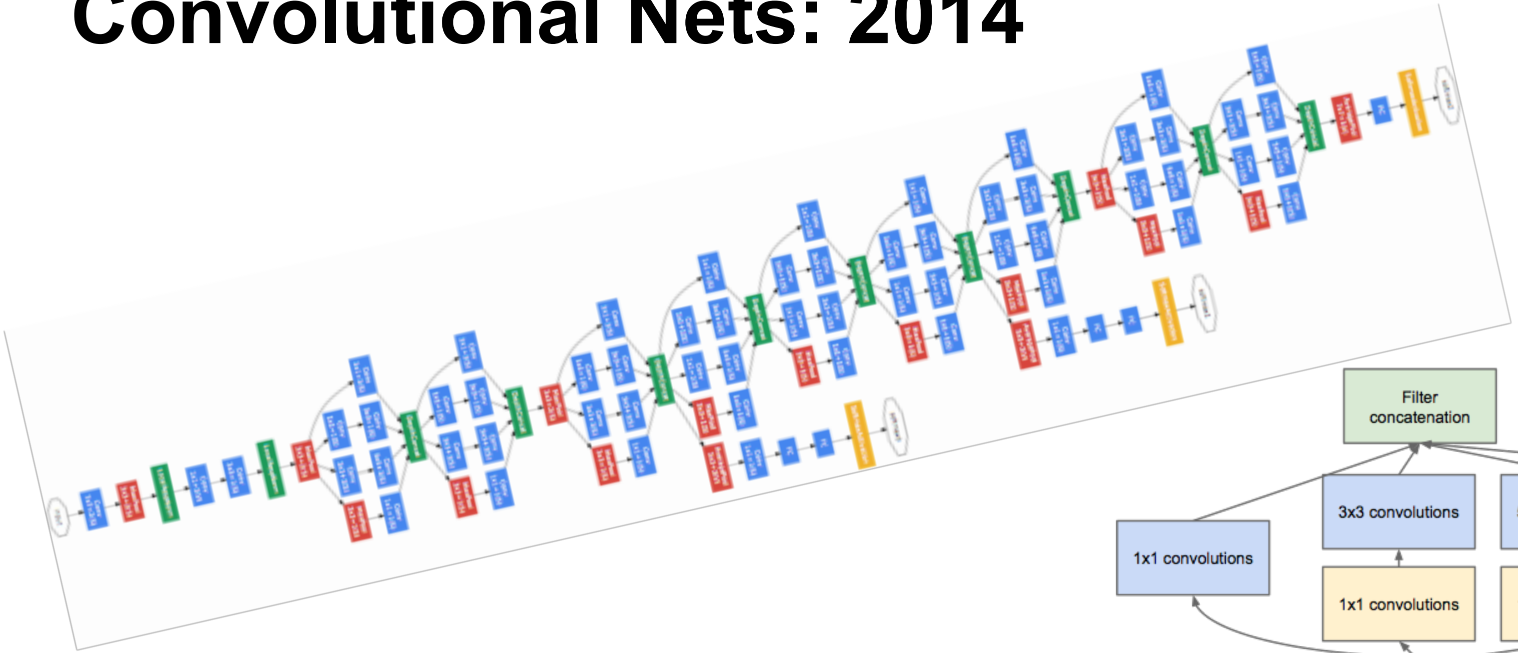
Convolutional Nets: 2012

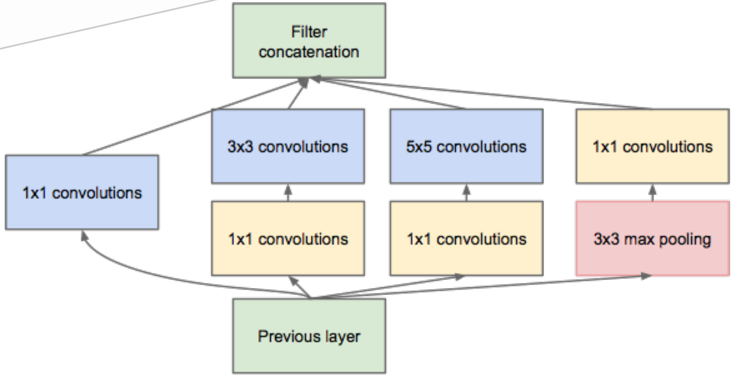


AlexNet: a layered model composed of convolution, pooling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12. [AlexNet]

The fully-connected "FULL" layers are linear classifiers / matrix multiplications. ReLU are rectified-linear non-linearities on layer output.

Convolutional Nets: 2014





ILSVRC14 Winners: **~6.6% Top-5 error**

- GoogLeNet: composition of multi-scale dimension-reduced modules
- VGG: 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers

+ depth
+ data
+ dimensionality reduction

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 settings.
- **Openness**: common code and reference models for reproducibility.
- **Community**: joint discussion and development through BSD-2 licensing.

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

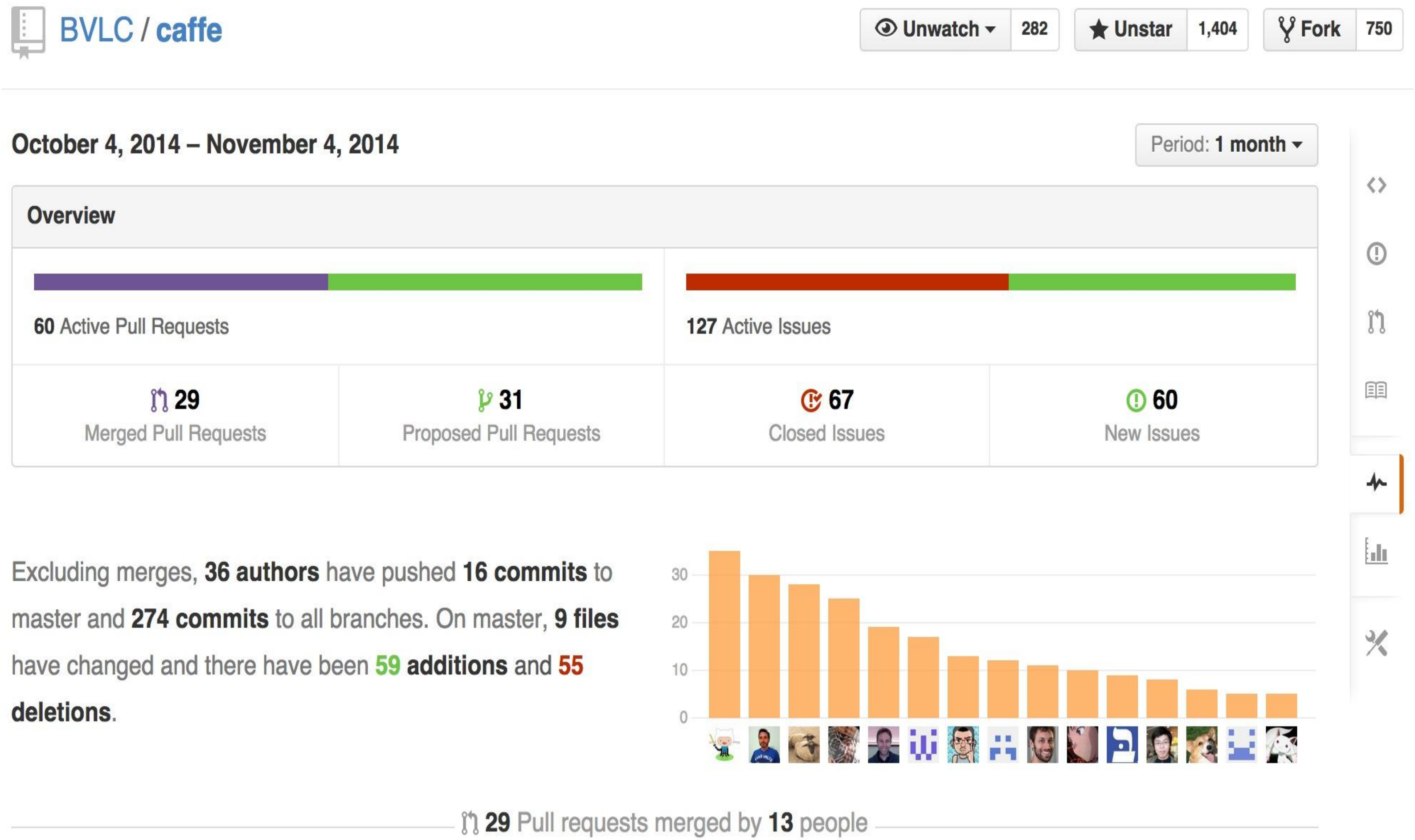
Training

Deployment

All with essentially the same code!

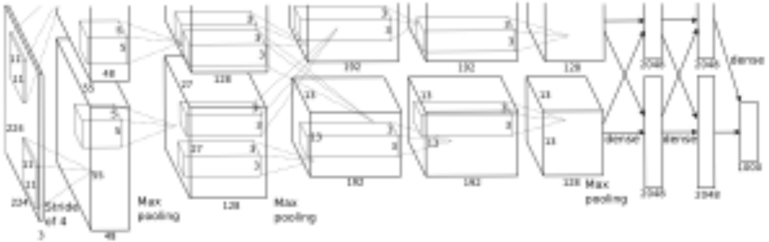
Caffe is a Community

project pulse

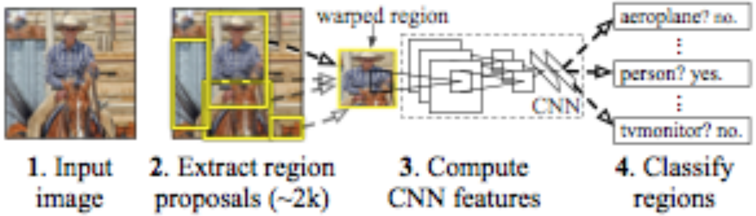


Reference Models

AlexNet: ImageNet Classification



R-CNN: ***Regions with CNN features***



Caffe offers the

- model definitions
- optimization settings
- pre-trained weights

so you can start right away.

The BVLC reference models are for unrestricted use.

Open Model Collection

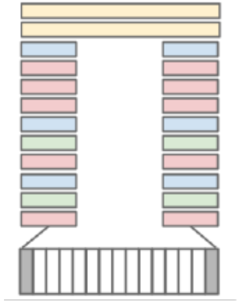
The Caffe Model Zoo

- 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

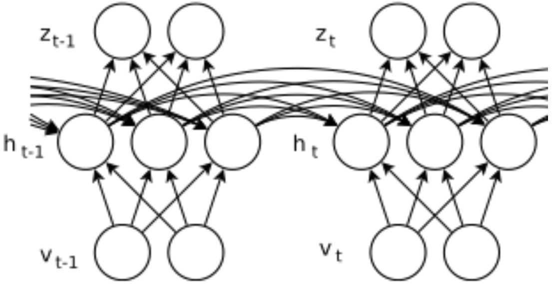
Architectures

DAGs
multi-input
multi-task



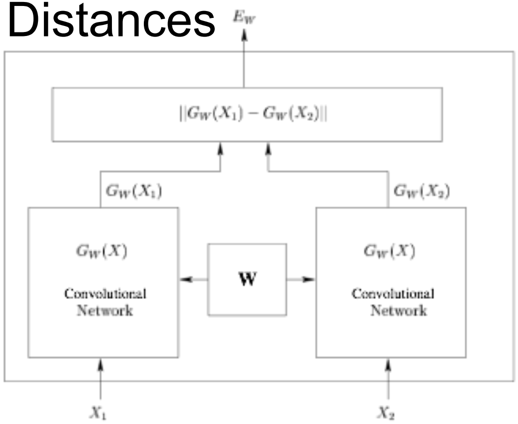
[Karpathy14]

Weight Sharing
Recurrent (RNNs)
Sequences



[Sutskever13]

Siamese Nets
Distances



[Chopra05]

Define your own model from our catalogue of layers types and start learning.

Brewing by the Numbers...

- Speed with Krizhevsky's 2012 model:
  - K40 / Titan: 2 ms / image, K20: 2.6ms
  - Caffe + cuDNN: **1.17ms / image** on K40
  - 60 million images / day
  - 8-core CPU: ~20 ms/image
- ~ 9K lines of C/C++ code
  - with unit tests ~20k

C++ 84.2% Python 10.5% Cuda 3.9% Other 1.4%

* Not counting I/O time. Details at http://caffe.berkeleyvision.org/performance_hardware.html

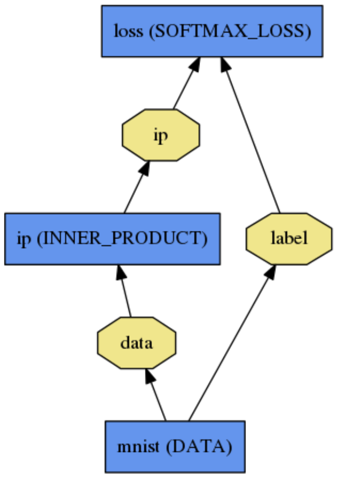
CAFFE INTRO

Net

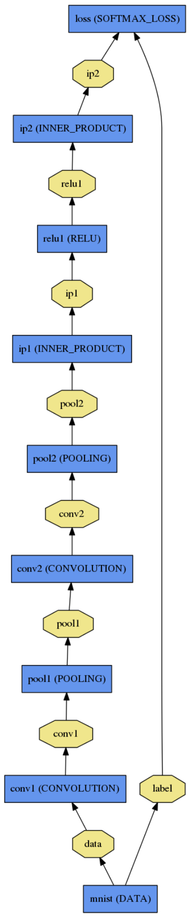
- A network is a set of layers connected as a DAG:

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

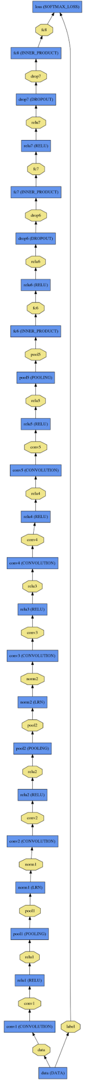
- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – a an array interface



LogReg ↑



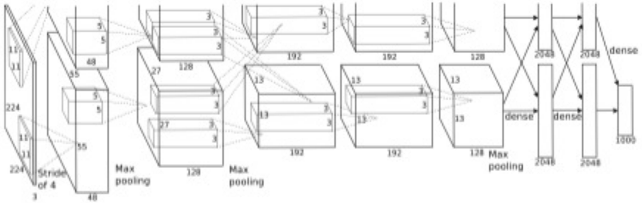
LeNet →



ImageNet, Krizhevsky 2012 →

Forward / Backward the essential Net computations

Forward: inference $f_W(x)$



"espresso" + loss

$\nabla f_W(x)$ Backward: learning

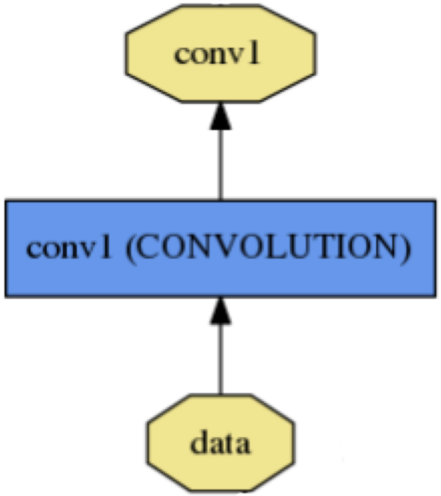
Caffe models are complete machine learning systems for inference and learning. The computation follows from the model definition. Define the model and run.

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**

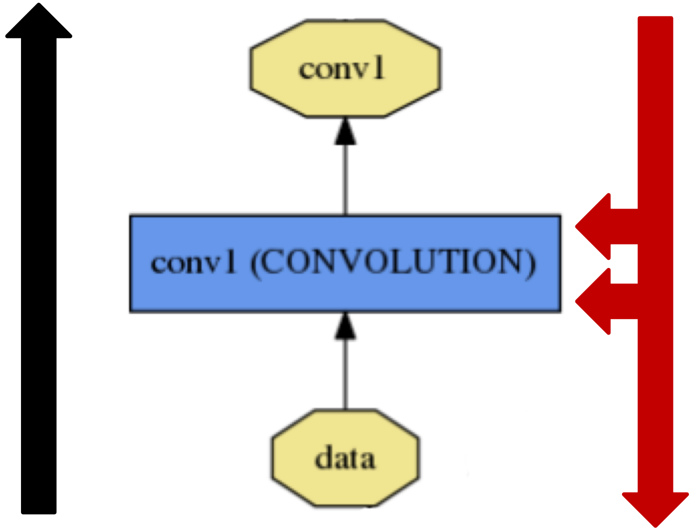
* Nets + Layers are defined by protobuf schema

Layer Protocol

Setup: run once for initialization.
Reshape: set dimensions.

Forward: make output given input.

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



Model Composition
The Net forward and backward passes are the composition the layers'.

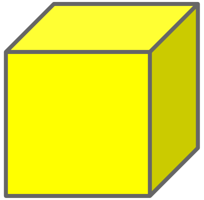
Layer Development Checklist

Blob

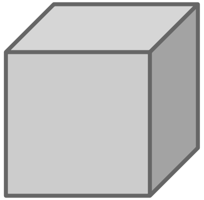
```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
… definition …
```

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

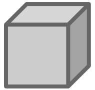
- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



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

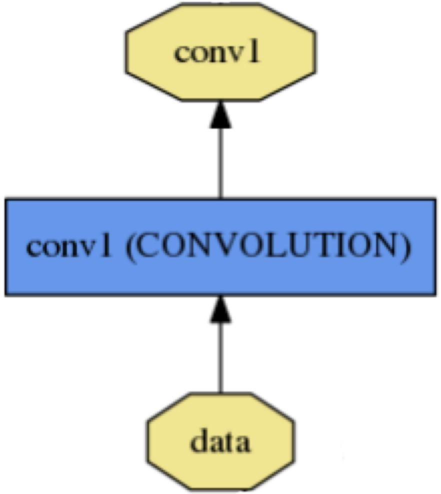


Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1



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

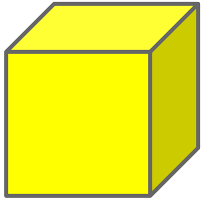
top
blob



bottom
blob

Blob

Blobs provide a unified memory interface.

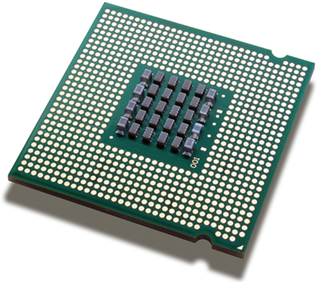


Reshape(num, channel, height, width)
- declare dimensions
- make *SyncedMem* -- but only lazily allocate

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

Model Schema: Protocol Buffer

- Defines domain-specific language in caffe.proto to determine
  - text schema
  - binary model format
- Generates programmer API
- Makes configuring, saving, and loading models simple

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

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"
solver_mode: GPU
```

All you need to run things on the GPU.

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

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

Step-by-Step Recipe...

- Convert the data to a 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/build/tools/*`

(Examples)

Logistic Regression

Learn LeNet on MNIST

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

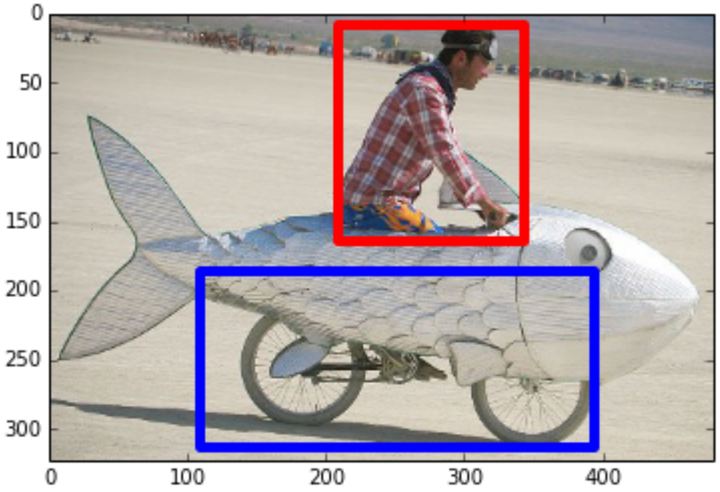
R-CNN: Regions with Convolutional Neural Networks

http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/detection.ipynb

Full R-CNN scripts available at

https://github.com/rbgirshick/rcnn

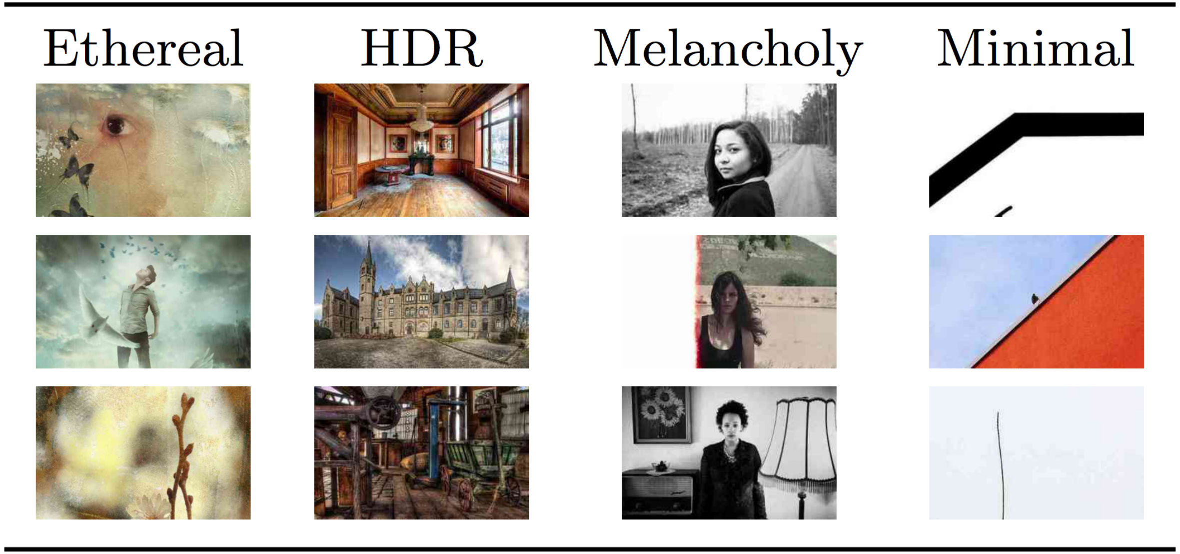
Ross Girshick et al.
Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR14.



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).



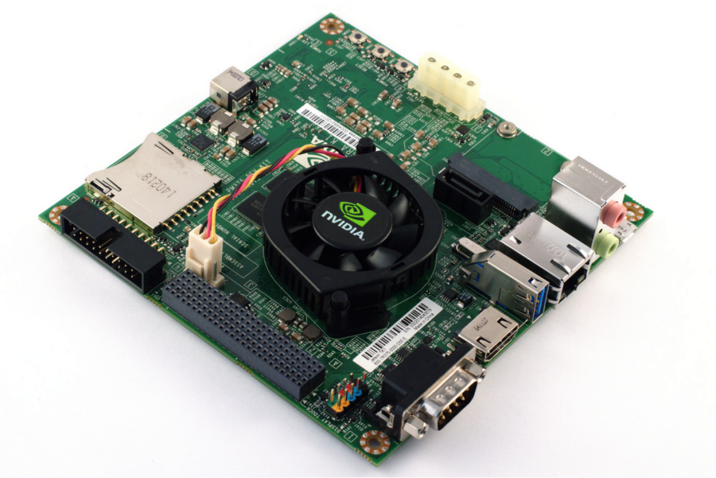
[Image-Style]

Other Styles:

[Vintage](#)
[Long Exposure](#)
[Noir](#)
[Pastel](#)
[Macro](#)
… and so on.

Embedded Caffe

[Caffe on the NVIDIA Jetson TK1 mobile board]

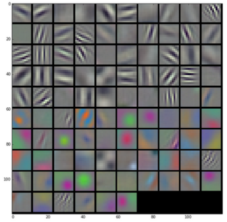


- 10 watts of power
- inference at 35 ms per image
- NVIDIA acceleration just released
- how-to guide
  courtesy of Pete Warden
- cuDNN for TK1 recently released!

Feature Extraction + Visualization

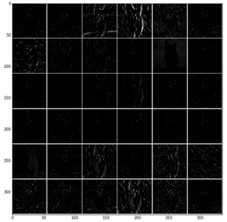
The first layer filters, `conv1`

In [8]:
```
# the parameters are a list of [weights, biases]
filters = net.params['conv1'][0].data
vis_square(filters.transpose(0, 2, 3, 1))
```

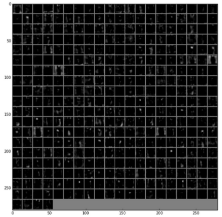


The first layer output, `conv1` (rectified responses of the filters above, first 36 only)

In [9]:
```
feat = net.blobs['conv1'].data[4, :36]
vis_square(feat, padval=1)
```

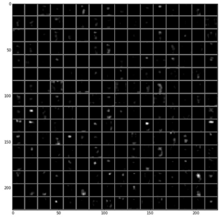


In [13]:
```
feat = net.blobs['conv4'].data[4]
vis_square(feat, padval=0.5)
```



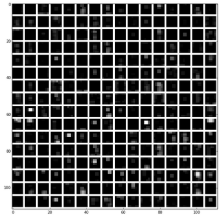
The fifth layer output, `conv5` (rectified, all 256 channels)

In [14]:
```
feat = net.blobs['conv5'].data[4]
vis_square(feat, padval=0.5)
```

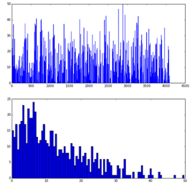


The fifth layer after pooling, `pool5`

In [15]:
```
feat = net.blobs['pool5'].data[4]
vis_square(feat, padval=1)
```

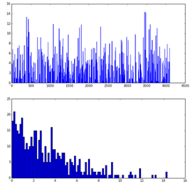


In [16]:
```
feat = net.blobs['fc6'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
plt.subplot(2, 1, 2)
_ = plt.hist(feat.flat[feat.flat > 0], bins=100)
```



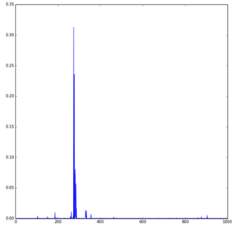
The second fully connected layer, `fc7` (rectified)

In [17]:
```
feat = net.blobs['fc7'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
plt.subplot(2, 1, 2)
_ = plt.hist(feat.flat[feat.flat > 0], bins=100)
```

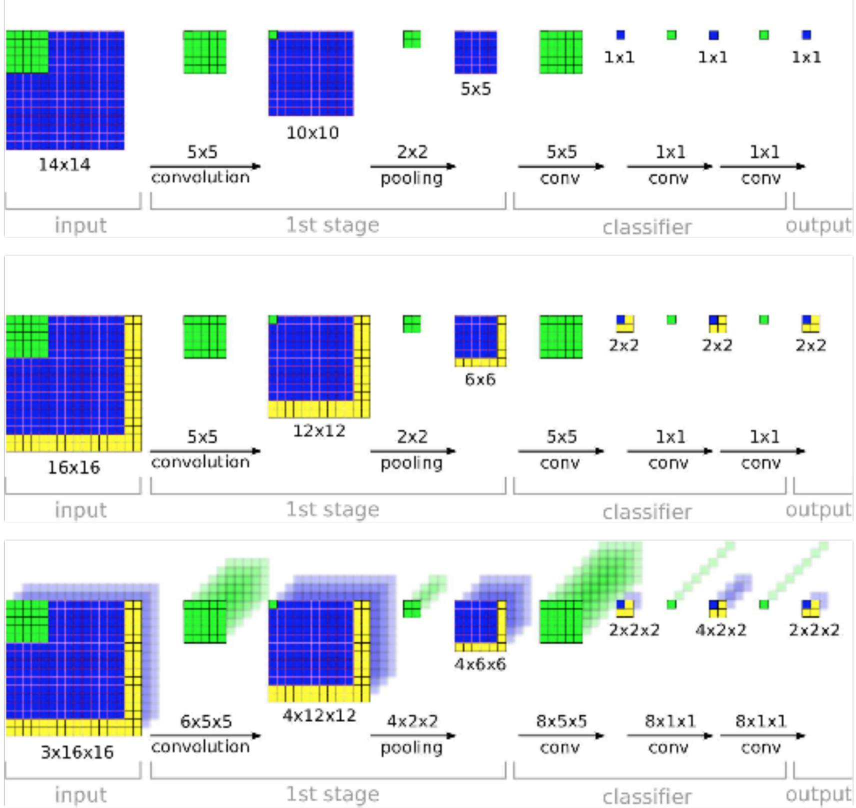


In [18]:
```
feat = net.blobs['prob'].data[4]
plt.plot(feat.flat)
```

Out[18]: [<matplotlib.lines.Line2D at 0x12b260710>]



Editing Model Parameters



[OverFeat]

Transform fixed-input models into any-size models by translating inner products to convolutions.

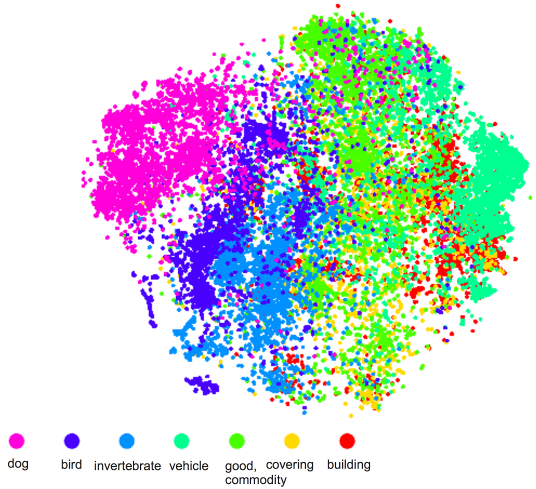
The computation exploits a natural efficiency of convolutional neural network (CNN) structure by dynamic programming in the forward pass from shallow to deep layers and analogously in backward.

[Net surgery in Caffe](#)
how to transform models:
- make fully convolutional
- transplant parameters

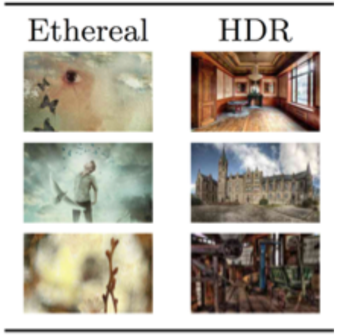
FINE-TUNING

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

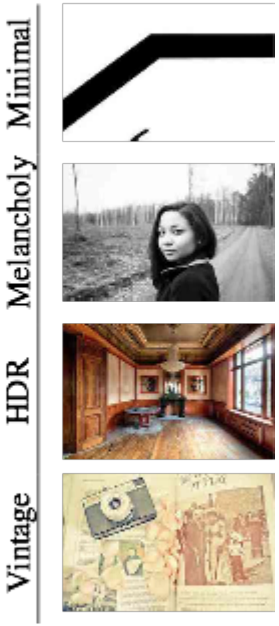
Last Layer:
A different classifier

From ImageNet to Style

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

Under the hood (loosely speaking):

```
net = new Caffe::Net(
    "style_solver.prototxt");
net.CopyTrainedNetFrom(
    pretrained_model);
solver.Solve(net);
```



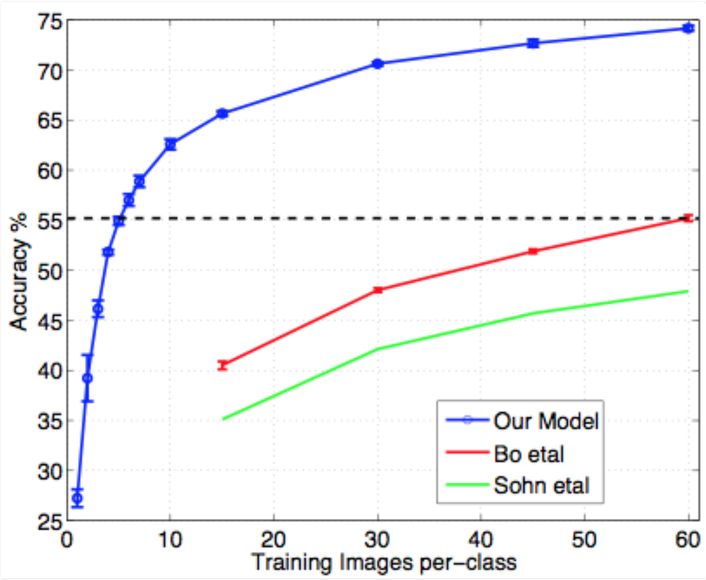
When to Fine-tune?

A good first step!

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

State-of-the-art results in

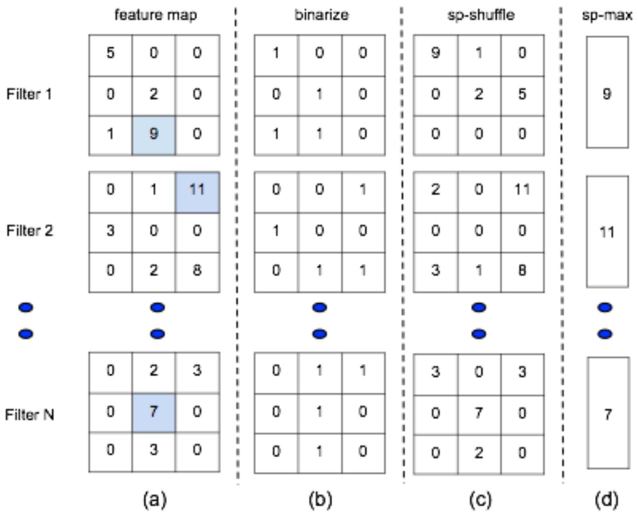
- recognition
- detection
- segmentation

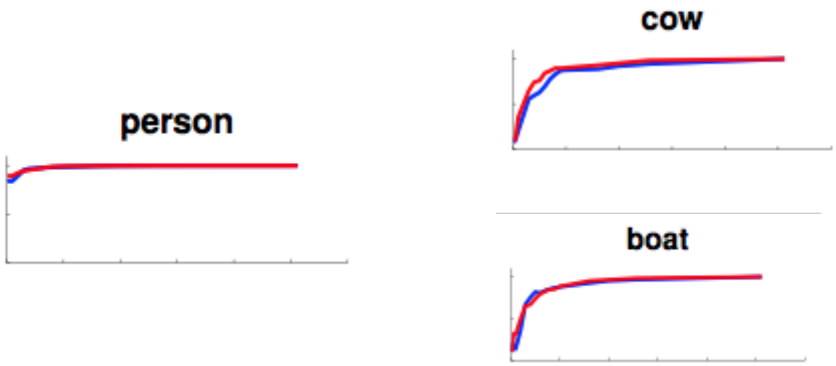


[Zeiler-Fergus]

Training & Fine-tuning Analysis

- Supervised pre-training does not overfit
- Representation is (mostly) distributed
- Sparsity comes "for free" in deep representation





P. Agarwal et al. ECCV 14

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

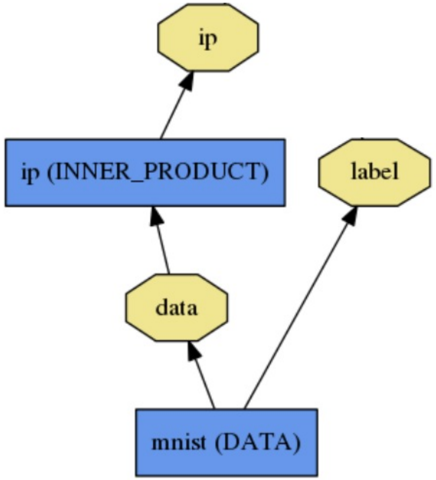
(Example)

Fine-tuning from ImageNet to Style

LOSS

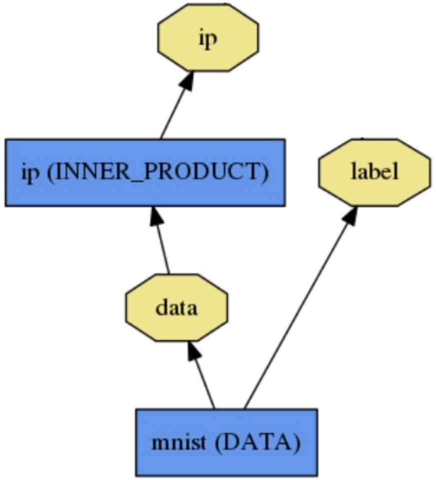
Loss

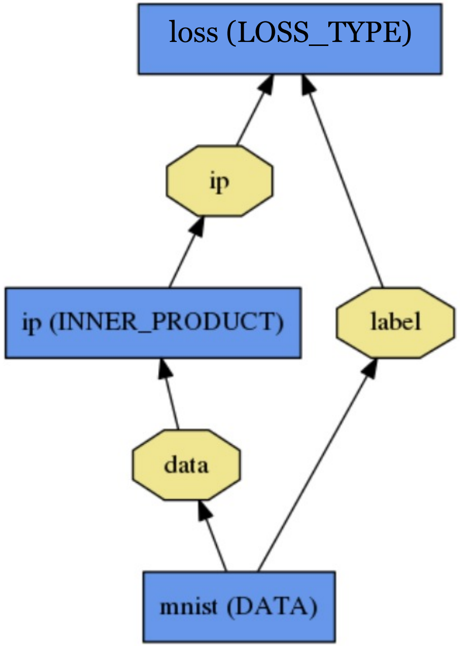
What kind of model is this?



Loss

What kind of model is this?





Classification
`SOFTMAX_LOSS`
`HINGE_LOSS`

Linear Regression
`EUCLIDEAN_LOSS`

Attributes / Multiclassification
`SIGMOID_CROSS_ENTROPY_LOSS`

Others…

New Task
`NEW_LOSS`

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

Loss

Loss function determines the learning task.

Given data D, a Net typically minimizes:

$$L(W) = \frac{1}{|D|} \sum_{i}^{|D|} f_W\left(X^{(i)}\right) + \lambda r(W)$$

Data term: error averaged over instances

Regularization term: penalize large weights to improve generalization

Loss

- The data error term $f_W\left(X^{(i)}\right)$ is computed by Net::Forward
- Loss is computed as the output of Layers
- Pick the loss to suit the task – many different losses for different needs

SOLVER

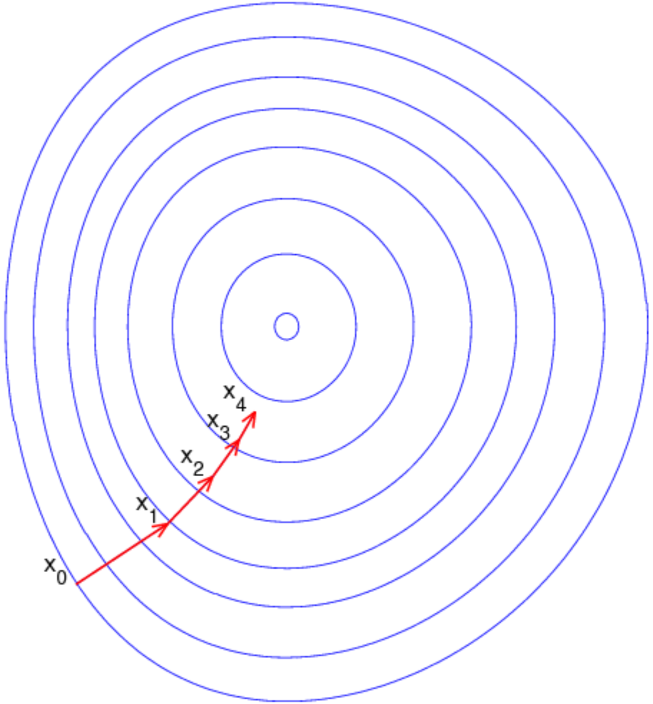
Optimization

How to minimize loss? $\ell(f_w(x, y))$

Descend the gradient. $\nabla \ell$

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

Fast, incremental learning by
Stochastic Gradient Descent (SGD)



Solver

- **Solver** optimizes the network weights W to minimize the loss L(W) over the data D

$$L(W) = \frac{1}{|D|} \sum_{i}^{|D|} f_W\left(X^{(i)}\right) + \lambda r(W)$$

- Coordinates forward / backward, weight updates, and scoring.

Solver

- Computes parameter update ΔW , formed from
  - The stochastic error gradient ∇f_W
  - The regularization gradient $\nabla r(W)$
  - Particulars to each solving method

$$L(W) \approx \frac{1}{N} \sum_{i}^{N} f_W\left(X^{(i)}\right) + \lambda r(W)$$

SGD Solver

- Stochastic gradient descent, with momentum
- `solver_type: SGD`

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

$$W_{t+1} = W_t + V_{t+1}$$

SGD Solver

- "AlexNet" [1] training strategy:
  - Use momentum 0.9
  - Initialize learning rate at 0.01
  - Periodically drop learning rate by a factor of 10
- Just a few lines of Caffe solver specification:

```
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 100000
max_iter: 350000
momentum: 0.9
```

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2012.

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

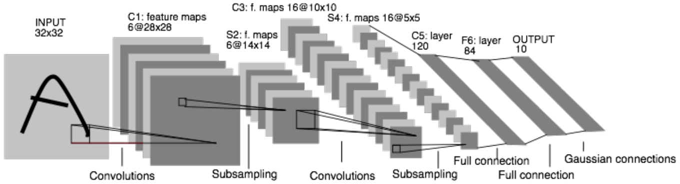
```
I0901 13:35:20.426187 20072 solver.cpp:232] Iteration 65000, loss = 61.5498
I0901 13:35:20.426218 20072 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:35:22.780092 20072 solver.cpp:289] Test loss: 60.8301
I0901 13:35:22.780138 20072 solver.cpp:302]     Test net output #0: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

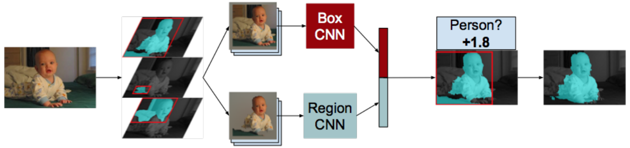
DAG

Many current deep models have linear structure

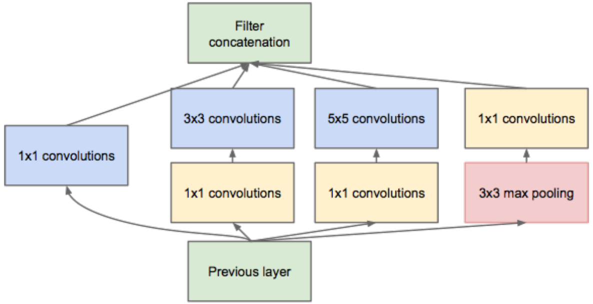


but Caffe nets can have any directed acyclic graph (DAG) structure.

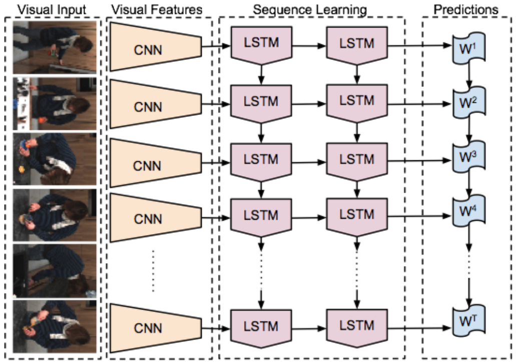
Define `bottoms` and `tops` and Caffe will connect the net.



SDS two-stream net



GoogLeNet Inception Module



LRCN joint vision-sequence model

WEIGHT SHARING

- Name parameters by the `param` field
- Layers with the same `param` name share the parameter, accumulating gradients accordingly
- Use cases
  - multi-scale pyramid
  - sequences
  - regularization

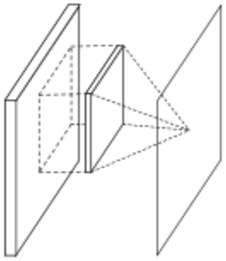
```
layers: {
  name: 'innerproduct1'
  type: INNER_PRODUCT
  inner_product_param {
    num_output: 10
    bias_term: false
    weight_filler {
      type: 'gaussian'
      std: 10
    }
  }
  param: 'sharedweights'
  bottom: 'data'
  top: 'innerproduct1'
}
layers: {
  name: 'innerproduct2'
  type: INNER_PRODUCT
  inner_product_param {
    num_output: 10
    bias_term: false
  }
  param: 'sharedweights'
  bottom: 'data'
  top: 'innerproduct2'
}
```

RECENT MODELS

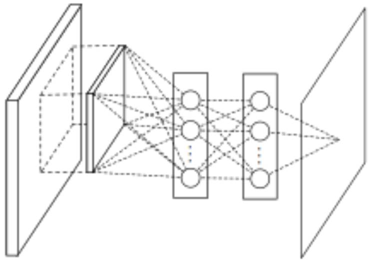
- Network-in-Network (NIN)
- GoogLeNet
- VGG

Network-in-Network

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

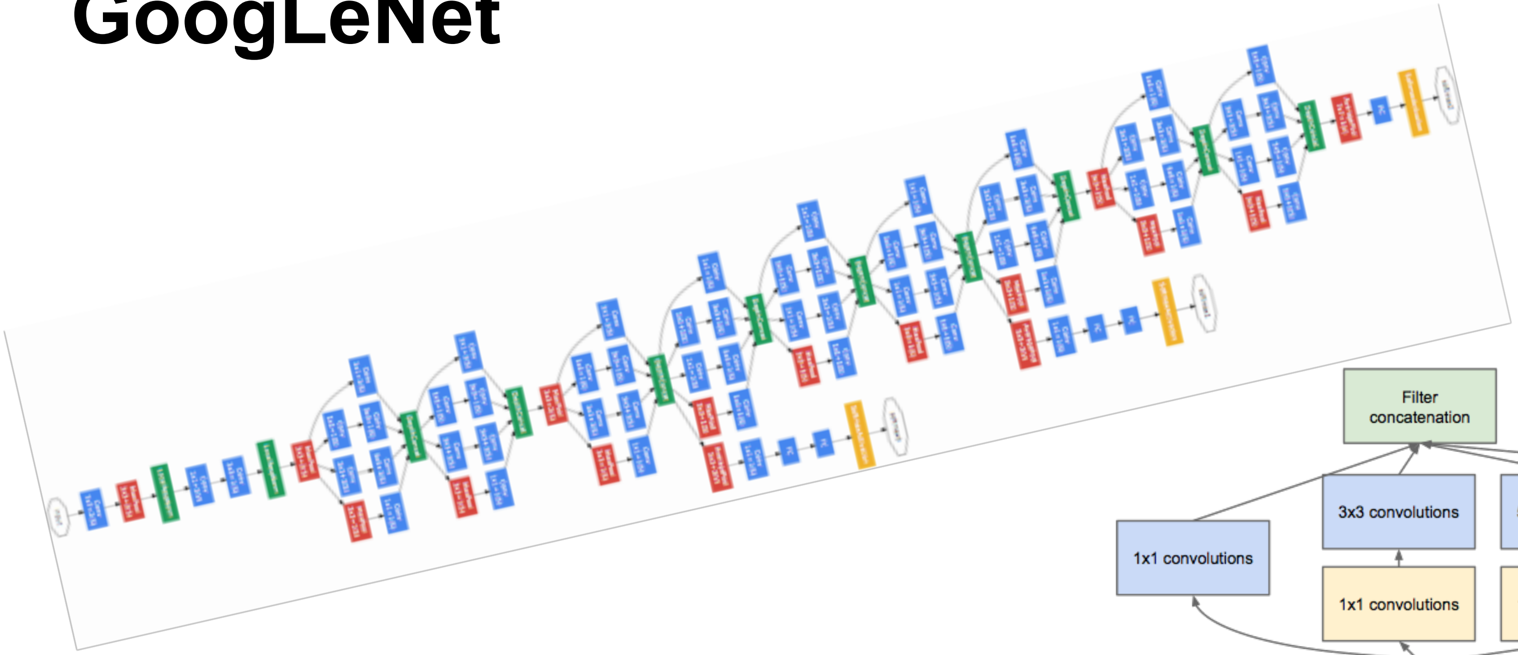


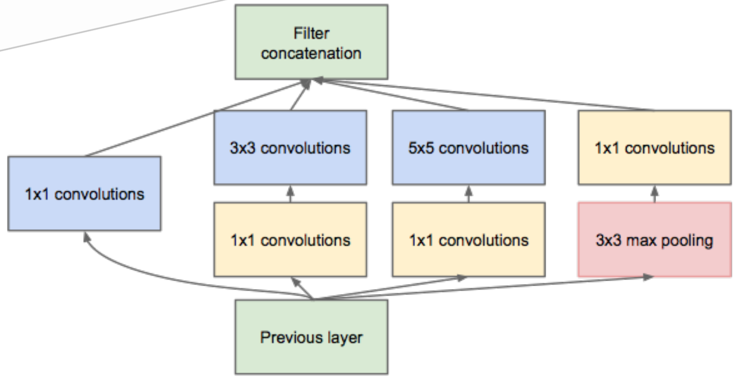
Linear Filter
CONV



NIN / MLP filter
1x1 CONV

GoogLeNet





“Inception” module

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

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

NOW ROASTING

- Parallelism
- Pythonification
- Fully Convolutional Networks
- Sequences
- cuDNN v2
- Gradient Accumulation
- More
    - FFT convolution
    - locally-connected layer
    - ...

Parallelism

Parallel / distributed training across
GPUs, CPUs, and cluster nodes

- collaboration with Flickr + open source community
- promoted to official integration branch in PR #1148
- faster learning and scaling to larger data

Pythonification

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

Jon Long

Fully Convolutional Network: FCN

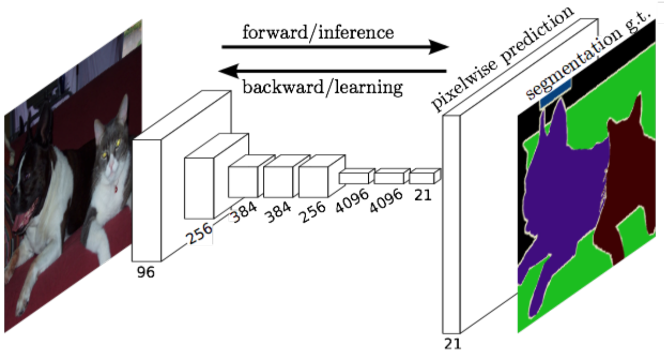
A framework for spatial prediction by conv. net applied to semantic segmentation

- end-to-end learning
- efficiency in inference and learning
  0.3 s for whole image prediction
- multi-modal, multi-task

Further applications

- depth estimation
- denoising

arXiv



Jon Long & Evan Shelhamer

Sequences

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

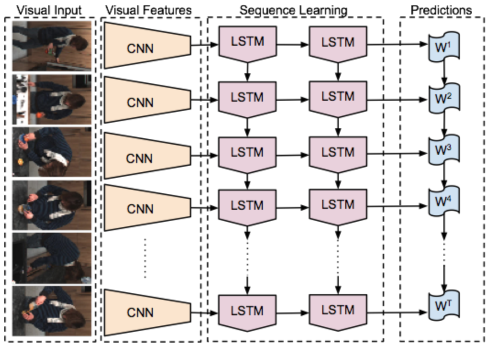
- video
- language
- dynamics

learned by back-propagation through time.

LRCN: Long-term Recurrent Convolutional Network

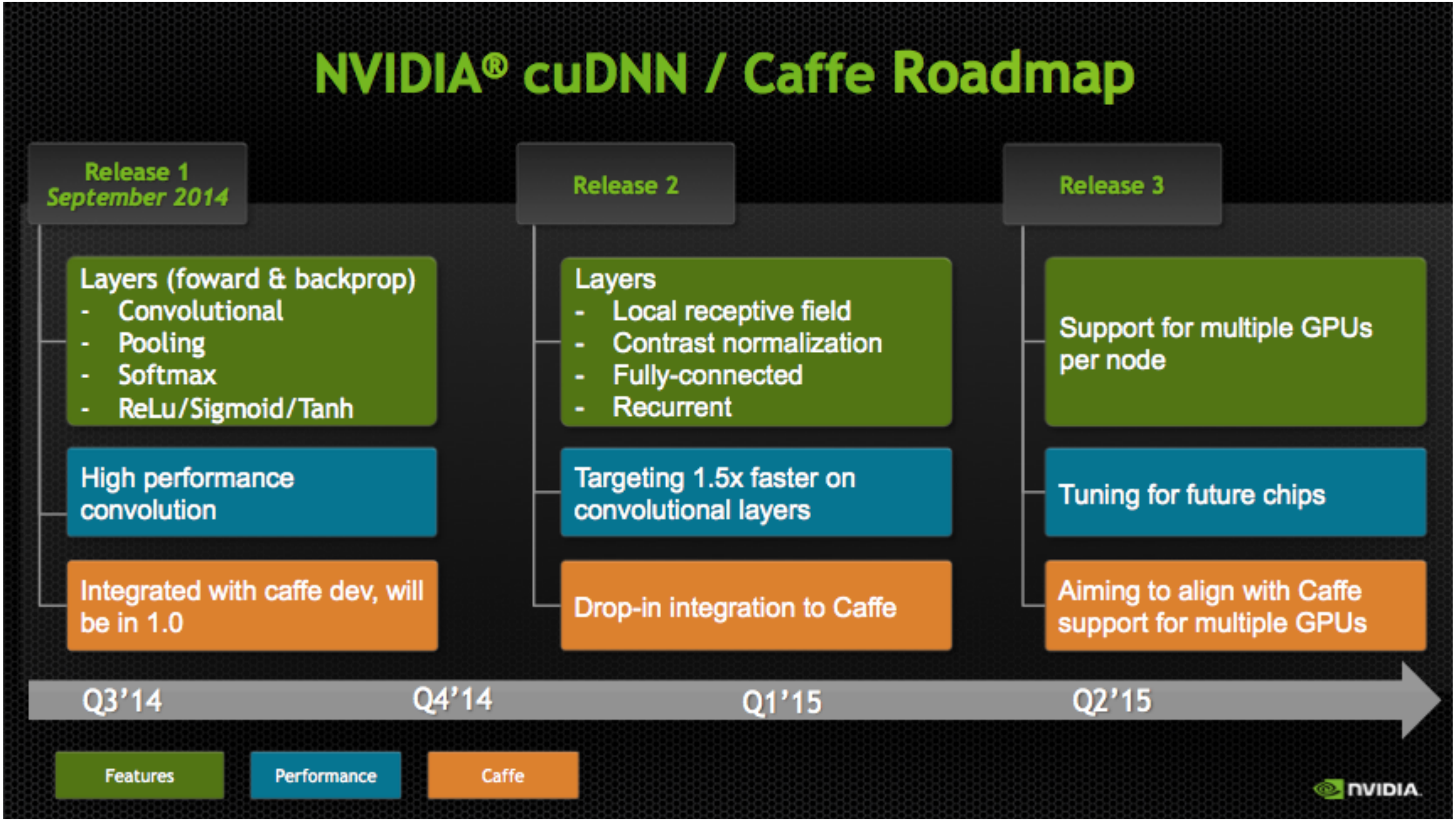
- activity recognition
- image captioning
- video captioning

[arXiv]



A group of young men playing a game of soccer.

Jeff Donahue et al.


NVIDIA® cuDNN / Caffe Roadmap
Release 1
September 2014
Layers (foward & backprop)
- Convolutional
- Pooling
- Softmax
- ReLu/Sigmoid/Tanh
High performance convolution
Integrated with caffe dev, will be in 1.0
Release 2
Layers
- Local receptive field
- Contrast normalization
- Fully-connected
- Recurrent
Targeting 1.5x faster on convolutional layers
Drop-in integration to Caffe
Release 3
Support for multiple GPUs per node
Tuning for future chips
Aiming to align with Caffe support for multiple GPUs
Q3'14
Q4'14
Q1'15
Q2'15
Features
Performance
Caffe
NVIDIA

Gradient Accumulation

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

...and share convolution buffers for further memory savings.

Jon Long & Sergio Guadarrama

LAST SIP

Caffe...

- is fast
- is state-of-the-art
- has tips, recipes, demos, and models
- brings together an active community
- ...all for free and open source

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 50+
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.

UPCOMING GTC EXPRESS WEBINARS

Wednesday, December 10 ArrayFire: A Productive Open Source GPU Software Library

Shehzan Mohammed, ArrayFire

Thursday, December 18 Photorealistic Visualization with Speed and Ease Using Iray+ for Autodesk 3ds Max

Shehzan Mohammed, ArrayFire

www.gputechconf.con/gtcexpress

GPU TECHNOLOGY CONFERENCE

March 17-20, 2015 | San Jose, CA
www.gputechconf.com #GTC15

REGISTRATION IS OPEN!

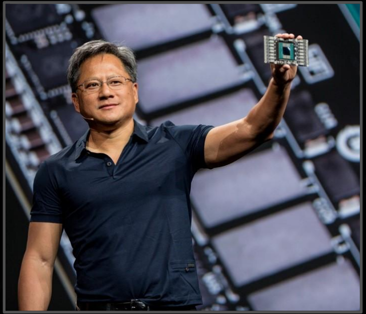
20% OFF
GM15WEB

CONNECT

Connect with experts from NVIDIA and other organizations across a wide range of fields

LEARN

Get key learnings and hands-on training in the 400+ sessions and 150+ research posters



DISCOVER

Discover the latest technologies shaping the GPU ecosystem

INNOVATE

Hear about disruptive innovations as early-stage start-ups present their work

4 Days | 3400+ Attendees | 400+ Sessions | 150+ Research Posters
40+ Countries | 180+ Press & Analytics | 100+ Exhibitors