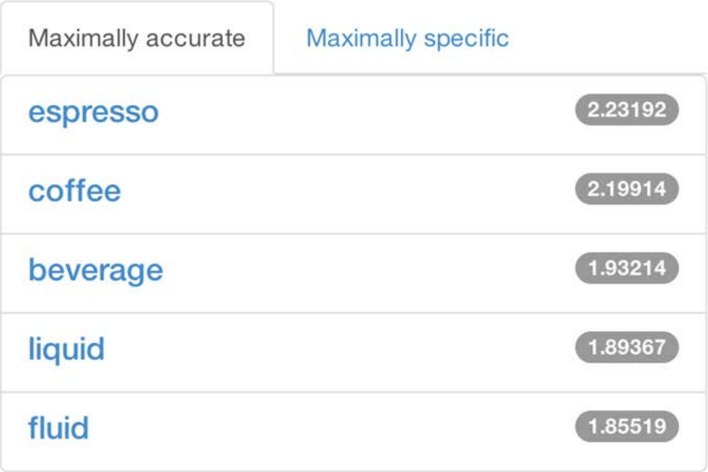
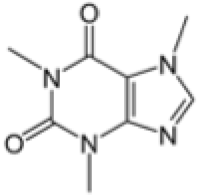
DIY Deep Learning for Vision:
a Hands-On Tutorial with Caffe





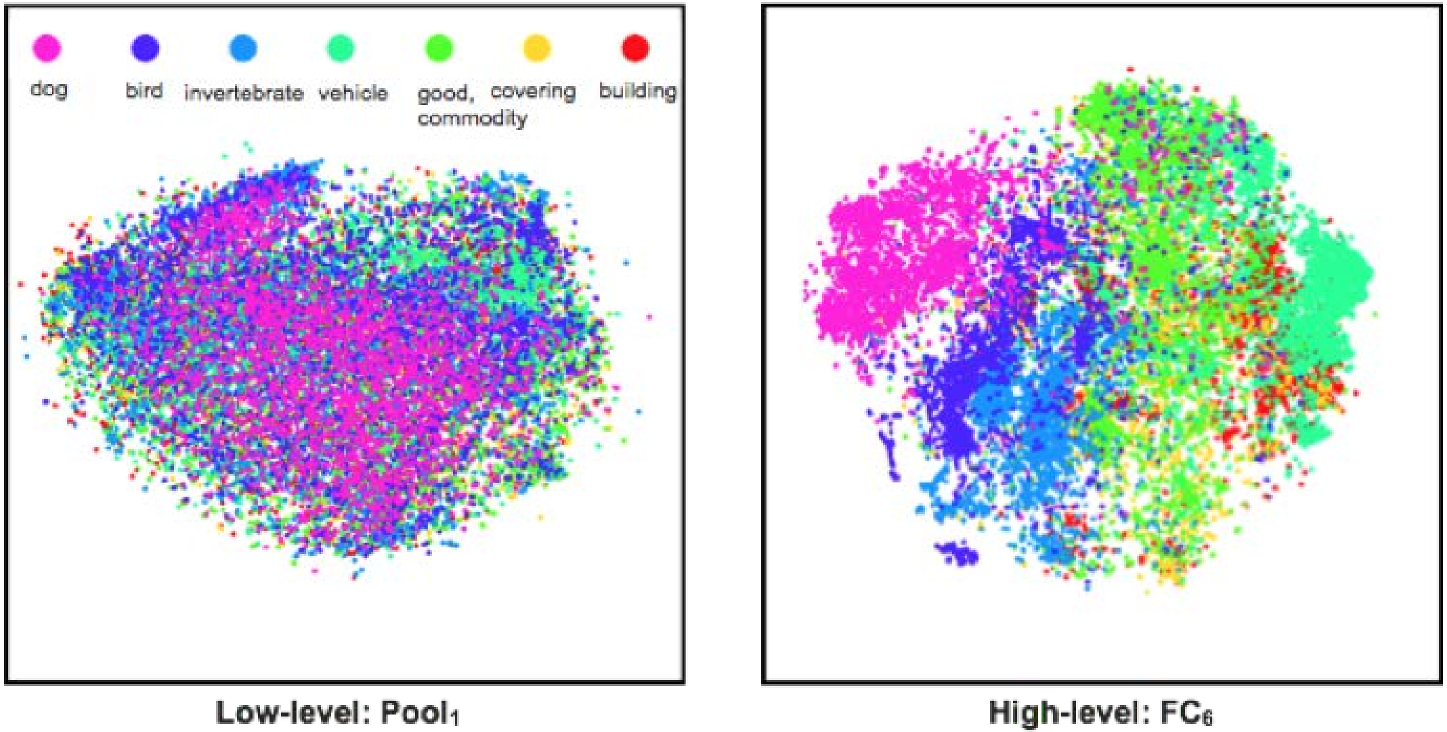
caffe.berkeleyvision.org

github.com/BVLC/caffe

Evan Shelhamer, Jeff Donahue,
Yangqing Jia, Ross Girshick

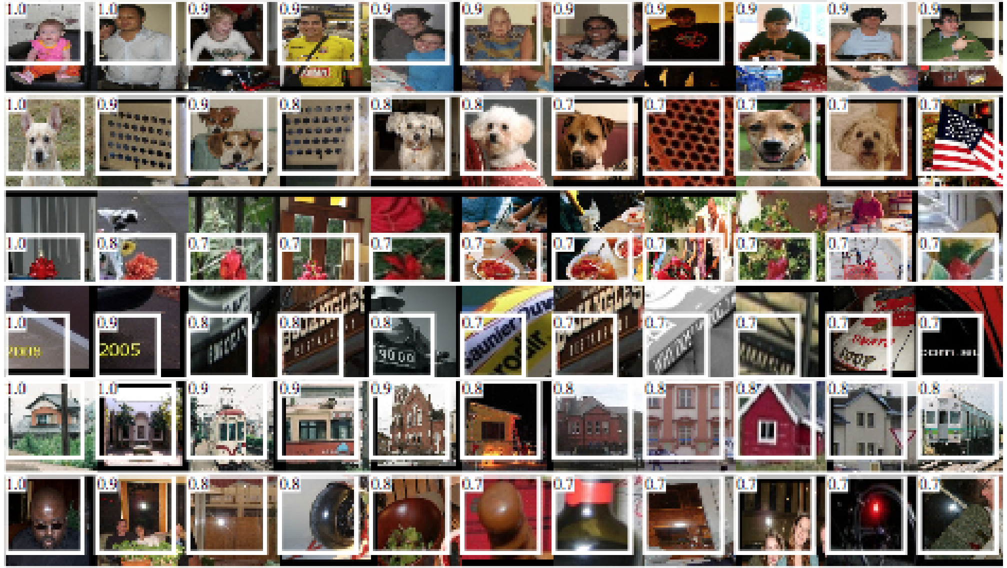
Look for further details in the outline notes

Why Deep Learning? The Unreasonable Effectiveness of Deep Features



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

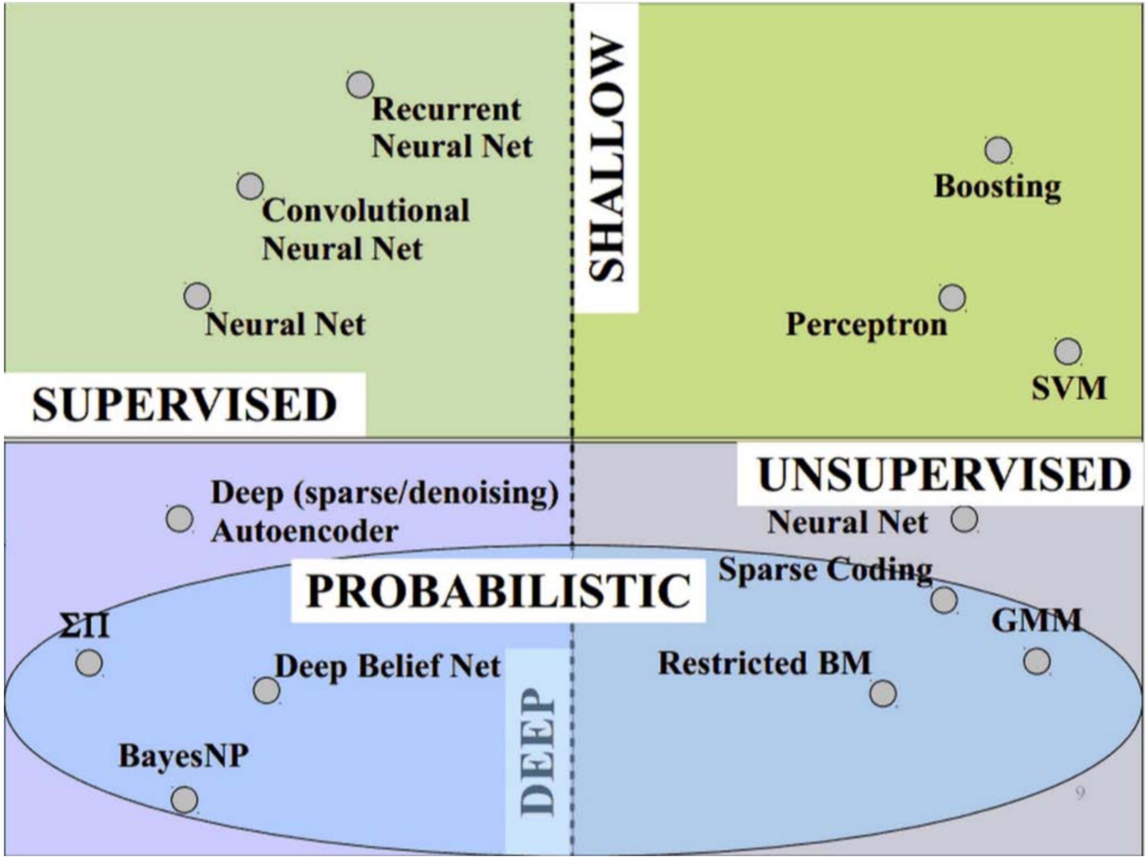
[Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

What is Deep Learning?

Compositional Models
Learned End-to-End

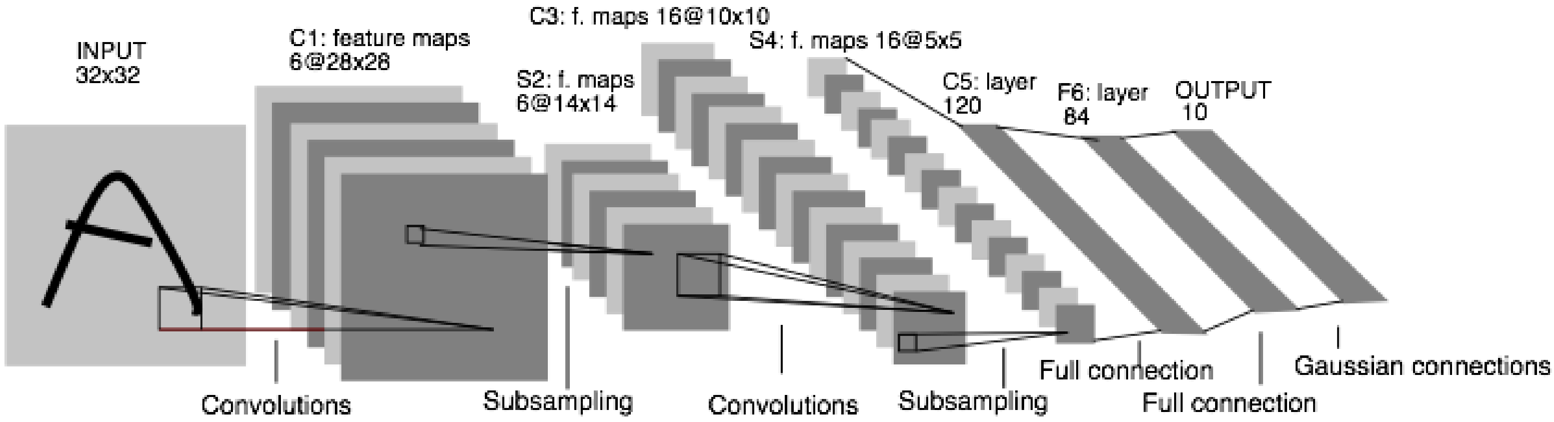
What is Deep Learning?



Vast space of models!

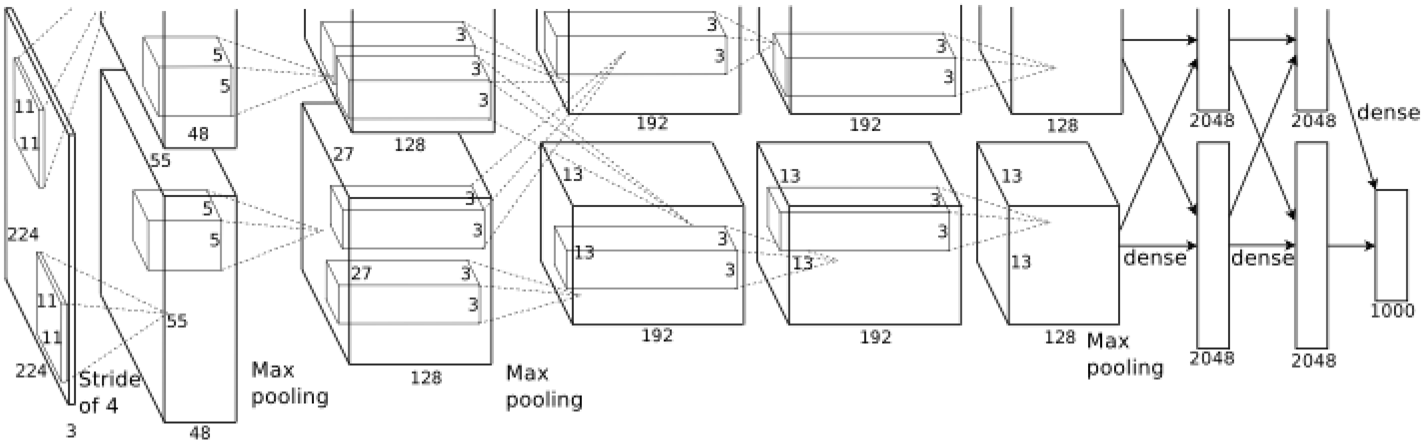
slide credit Marc'aurelio Ranzato, CVPR '14 tutorial.

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 Neural Nets (CNNs): 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

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 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

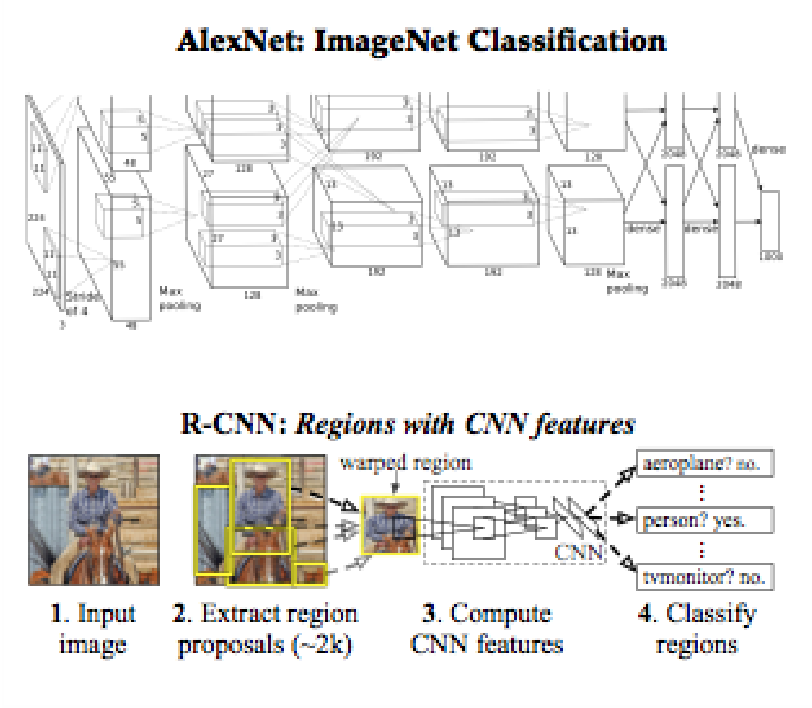
Deployment

All with essentially the same code!

Caffe is a Community

[project pulse]

Reference Models



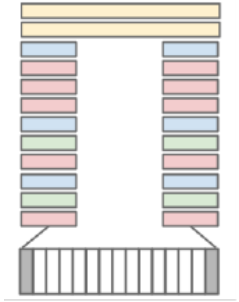
Caffe offers the

- model definitions
- optimization settings
- pre-trained weights

so you can start right away.

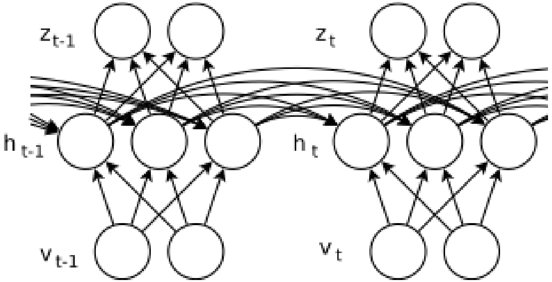
Architectures

DAGs
multi-input
multi-task



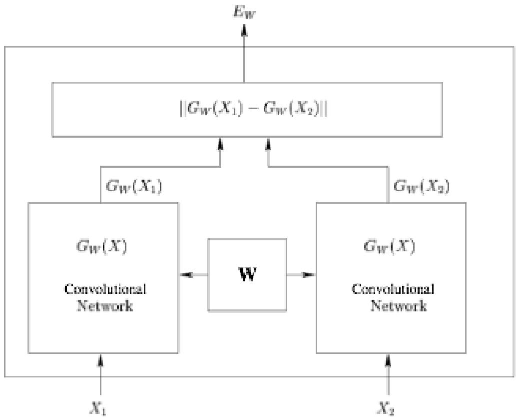
[Karpathy14]

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
  - 40 million images / day
  - Caffe + cuDNN: **1.17ms / image** on K40
  - 8-core CPU: ~20 ms/image
- ~ 9K lines of C/C++ code
  - with unit test: ~20k

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

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

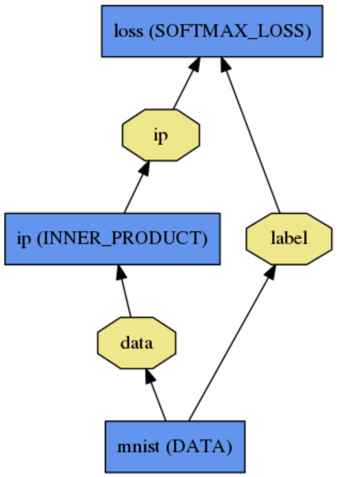
CAFFE INTRO

Net

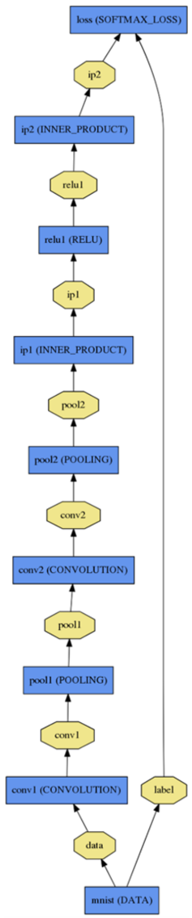
- A network is a set of layers connected as a DAG:

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

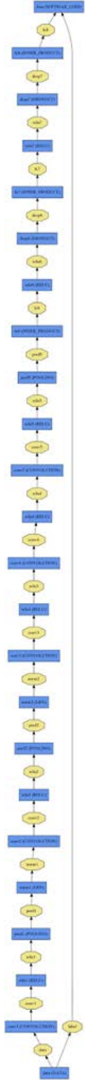
- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – a an array interface



LogReg ↑



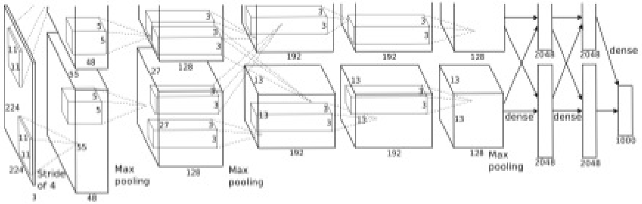
LeNet ⟶



ImageNet, Krizhevsky 2012 ⟶

Forward / Backward the essential Net computations

Forward: inference $f_W(x)$



"espresso" + loss

$\nabla f_W(x)$ Backward: learning

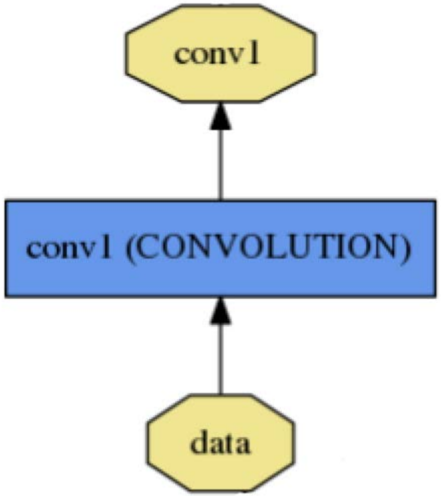
Caffe models are complete machine learning systems for inference and learning.
The computation follows from the model definition. Define the model and run.

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**

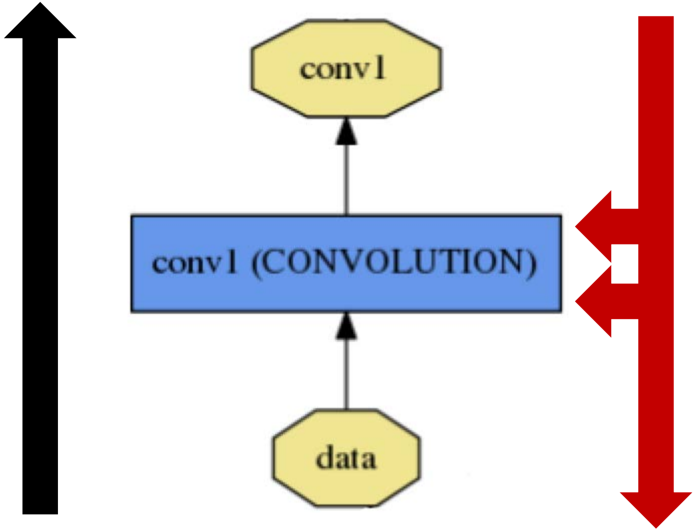
* Nets + Layers are defined by protobuf schema

Layer Protocol

Setup: run once for initialization.

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'.

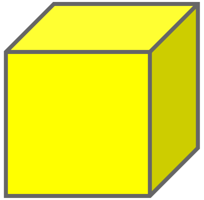
[Layer Development Checklist]

Blob

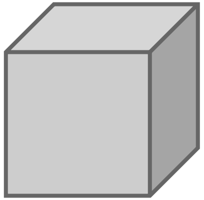
```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
… definition …
```

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

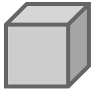
- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



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

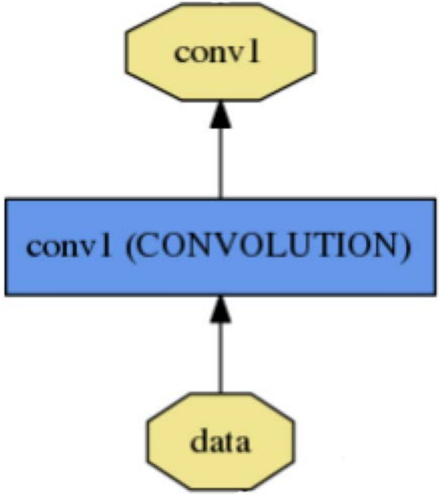


Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1



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

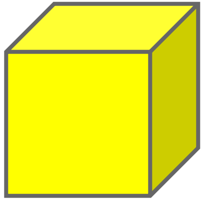
top
blob



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

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)

End to End Recipe...

- 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/build/tools/*`

(Examples)

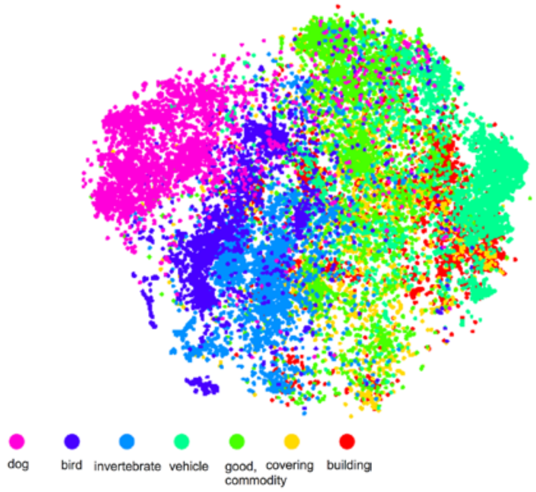
Logistic Regression

Learn LeNet on MNIST

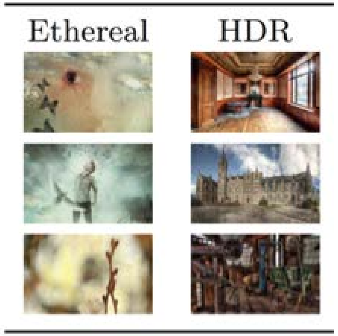
FINE-TUNING

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:

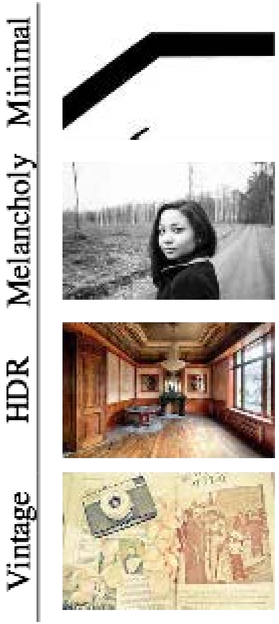
A different classifier

From ImageNet to Style

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

Under the hood (loosely speaking):

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



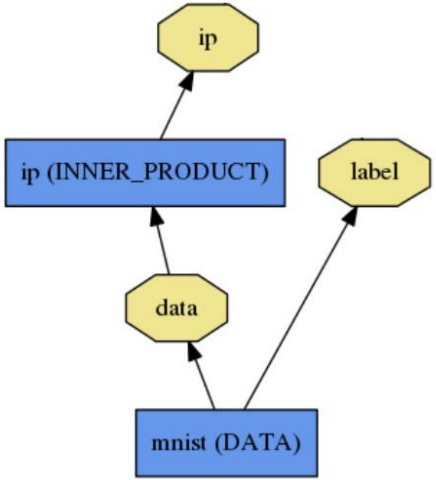
(Example)

Fine-tuning from ImageNet to Style

LOSS

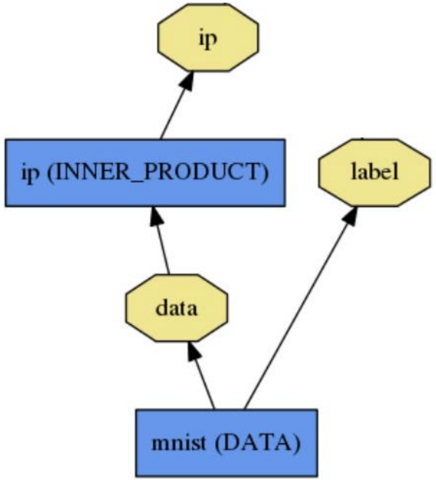
Loss

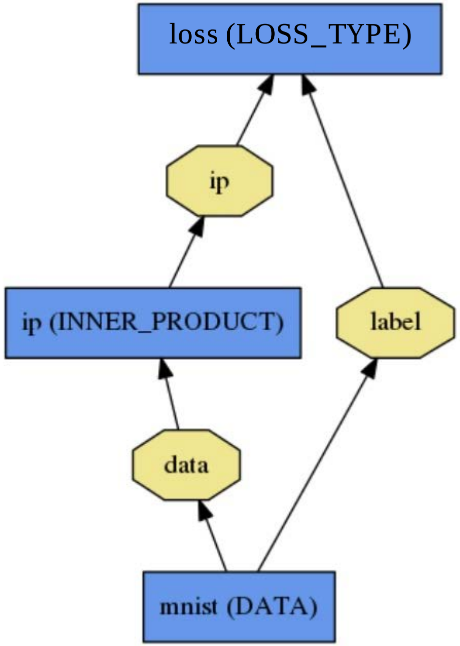
What kind of model is this?



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

Softmax Loss Layer

- Multinomial logistic regression: used for predicting a single class of K mutually exclusive classes

```
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

$$\hat{p}_{nk} = \exp(x_{nk}) / \left[\sum_{k'} \exp(x_{nk'})\right]$$

$$E = \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n}),$$

Sigmoid Cross-Entropy Loss

- Binary logistic regression: used for predicting K independent probability values in [0, 1]

```
layers {
  name: "loss"
  type: SIGMOID_CROSS_ENTROPY_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

$$y = (1 + \exp(-x))^{-1}$$

$$E = \frac{-1}{n} \sum_{n=1}^{N} \left[p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n) \right]$$

Euclidean Loss

- A loss for regressing to real-valued labels [-inf, inf]

```
layers {
  name: "loss"
  type: EUCLIDEAN_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

$$E = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{y}_n - y_n||_2^2$$

Multiple loss layers

- Your network can contain as many loss functions as you want
- Reconstruction and Classification:

```
layers {
  name: "recon-loss"
  type: EUCLIDEAN_LOSS
  bottom: "reconstructions"
  bottom: "data"
  top: "recon-loss"
}

layers {
  name: "class-loss"
  type: SOFTMAX_LOSS
  bottom: "class-preds"
  bottom: "class-labels"
  top: "class-loss"
}
```

$$E = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{y}_n - y_n||_2^2 + \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$$

Multiple loss layers

"*_LOSS" layers have a default loss weight of 1

```
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

==

```
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
loss_weight: 1.0
}
```

Multiple loss layers

- Give each loss its own weight
- E.g. give higher priority to classification error

```
layers {
  name: "recon-loss"
  type: EUCLIDEAN_LOSS
  bottom: "reconstructions"
  bottom: "data"
  top: "recon-loss"
}

layers {
  name: "class-loss"
  type: SOFTMAX_LOSS
  bottom: "class-preds"
  bottom: "class-labels"
  top: "class-loss"
  oss_weight: 100.0
}
```

$$E = \frac{1}{2N} \sum_{n=1}^{N} \|\hat{y}_n - y_n\|_2^2 + 100 * \frac{-1}{N} \sum_{n=1}^{N} \log(\hat{p}_{n,l_n})$$

Any layer can produce a loss!

- Just add `loss_weight: 1.0` to have a layer's output be incorporated into the loss

E = || pred - label ||^2 / (2N)

```
layers {
  name: "loss"
  type: EUCLIDEAN_LOSS
  bottom: "pred"
  bottom: "label"
  top: "euclidean_loss"
  loss_weight: 1.0
}
```

==

diff = pred - label

```
layers {
  name: "diff"
  type: ELTWISE
  bottom: "pred"
  bottom: "label"
  top: "diff"
  eltwise_param {
    op: SUM
    coeff: 1
    coeff: -1
  }
}
```

+

E = || diff ||^2 / (2N)

```
layers {
  name: "loss"
  type: POWER
  bottom: "diff"
  top: "euclidean_loss"
  power_param {
    power: 2
  }
  # = 1/(2N)
  loss_weight: 0.0078125
}
```

SOLVER

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: 0.01lr_policy:
"step"
gamma: 0.1
stepsize: 100000max_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.

NAG Solver

- Nesterov's accelerated gradient [1]
- `solver_type: NESTEROV`
- Proven to have optimal convergence rate $\mathcal{O}(1/t^2)$ for convex problems

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

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

[1] Y. Nesterov. A Method of Solving a Convex Programming Problem with Convergence Rate (1/sqrt(k)). *Soviet Mathematics Doklady*, 1983.

AdaGrad Solver

- Adaptive gradient (Duchi et al. [1])
- `solver_type: ADAGRAD`
- Attempts to automatically scale gradients based on historical gradients

$$(W_{t+1})_i = (W_t)_i - \alpha \frac{(\nabla L(W_t))_i}{\sqrt{\sum_{t'=1}^{t} (\nabla L(W_{t'}))_i^2}}$$

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 2011.

Solver Showdown: MNIST Autoencoder

AdaGrad

SGD

```
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
```

Nesterov

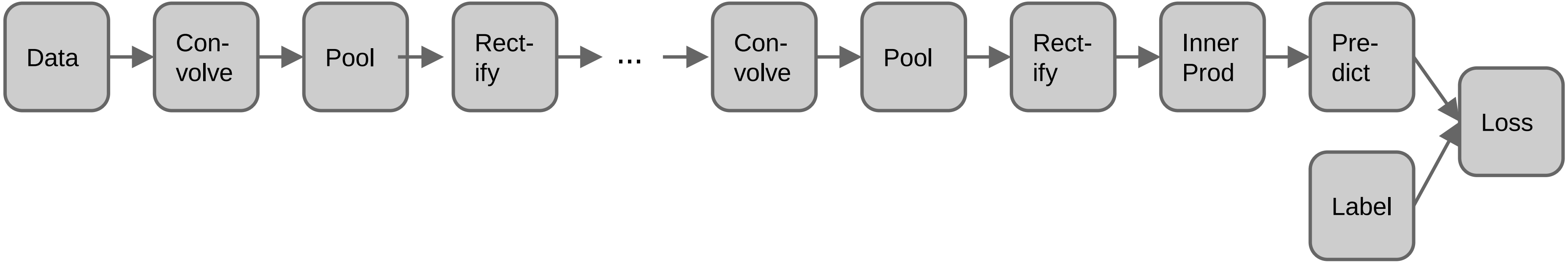
```
I0901 13:35:20.426187 20072 solver.cpp:232] Iteration 65000, loss = 61.5498
I0901 13:35:20.426218 20072 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:35:22.780092 20072 solver.cpp:289] Test loss: 60.8301
I0901 13:35:22.780138 20072 solver.cpp:302]     Test net output #0: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

```
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

Nets are DAGs

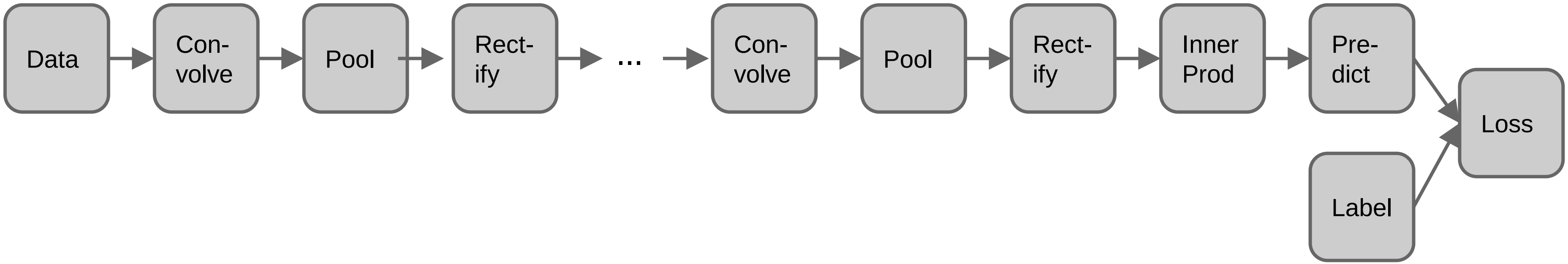
- Modern deep learning approaches to vision have a mostly linear structure



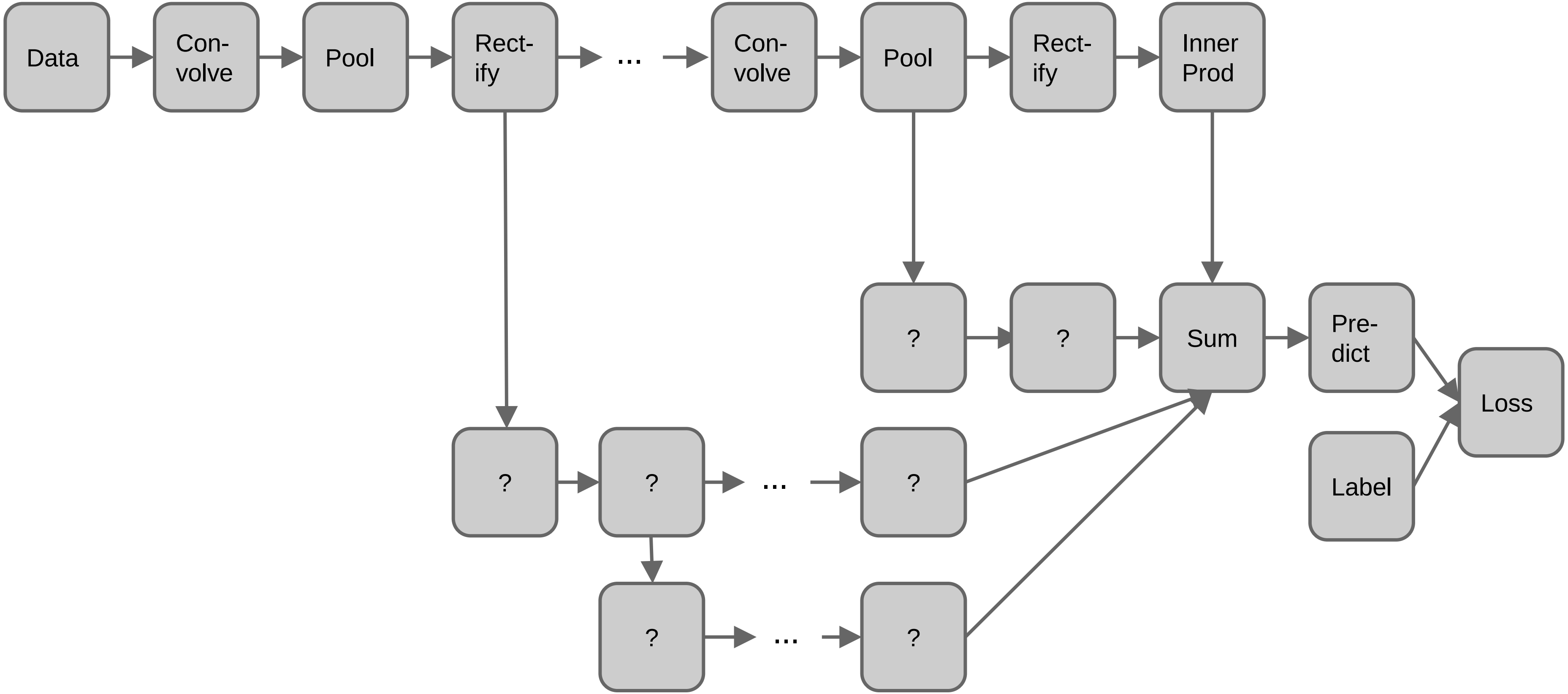
Nets are DAGs

- Modern deep learning approaches to vision have a mostly linear structure

- But Caffe nets can have any arbitrary directed acyclic graph (DAG) structure



Nets are DAGs



WEIGHT SHARING

Weight sharing

- Parameters can be shared and reused across Layers throughout the Net

- Applications:
  - Convolution at multiple scales / pyramids
  - Recurrent Neural Networks (RNNs)
  - Siamese nets for distance learning

Weight sharing

- Just give the parameter blobs explicit names using the `param` field
- Layers specifying the same `param` name will share that parameter, accumulating gradients accordingly

```
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'
}
```

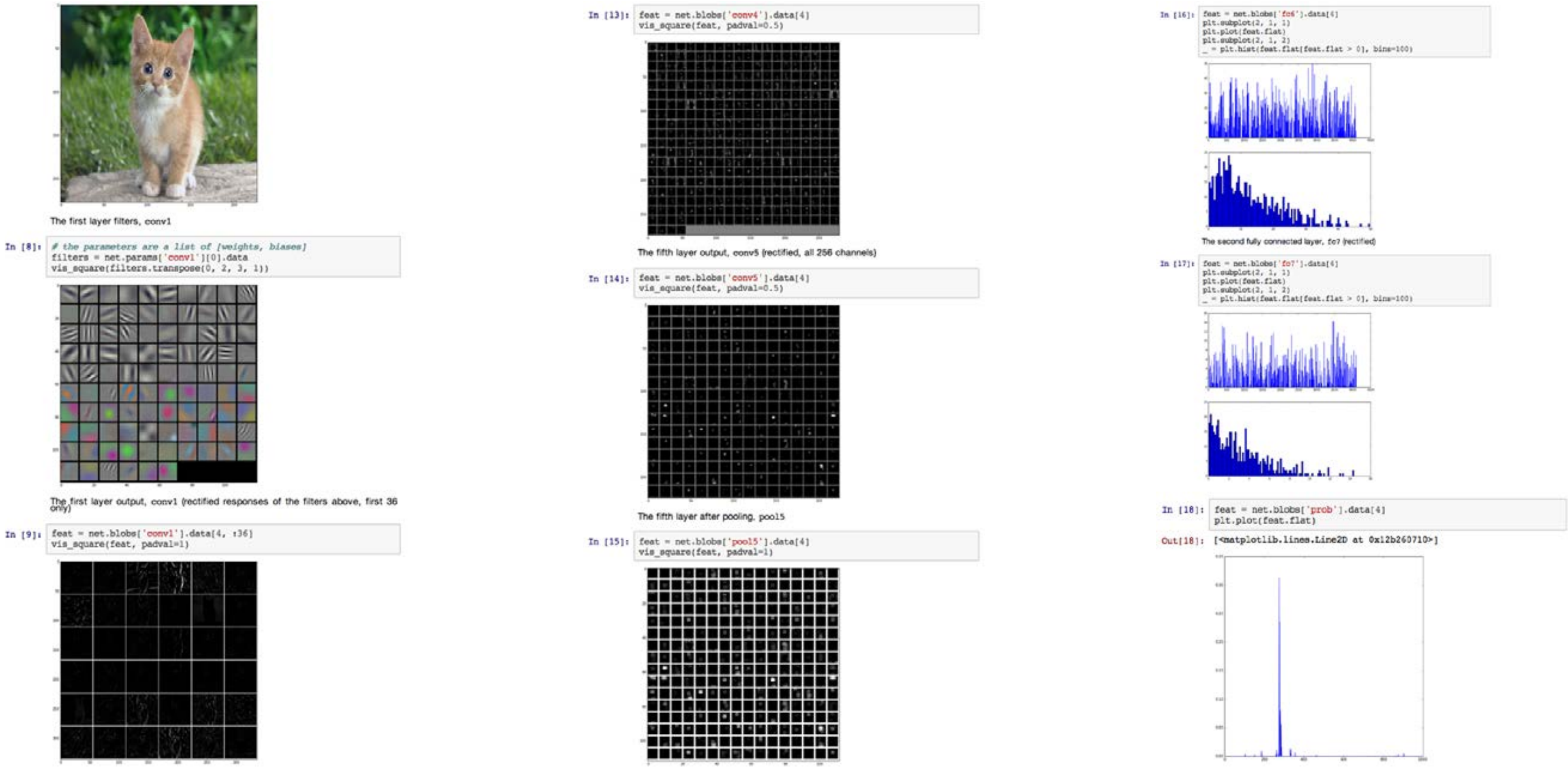
EXAMPLES

Share a Sip of Brewed Models

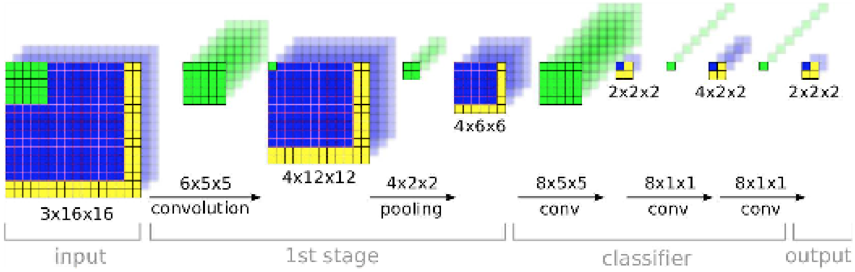
demo.caffe.berkeleyvision.org
demo code open-source and bundled

Maximally accurate	Maximally specific
cat	1.80727
domestic cat	1.74727
feline	1.72787
tabby	0.99133
domestic animal	0.78542

Feature Visualization



Fully-convolutional Models



[OverFeat]

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

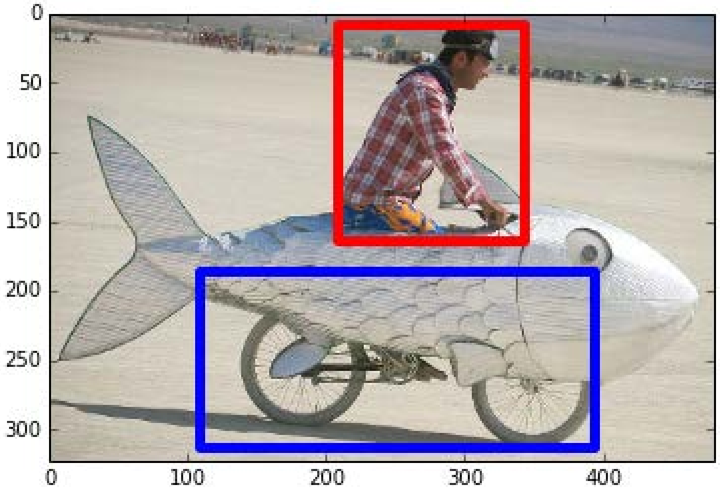
The computation exploits a natural efficiency of convolutional neural network (CNN) structure by dynamic programming in the forward pass from shallow to deep layers and analogously in backward.

Net surgery in Caffe
how to transform models:
- make fully-convolutional
- set custom weights

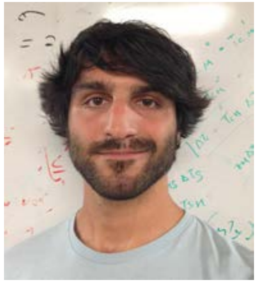
Object Detection

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.



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 collaboration on cuDNN
and GPU donation.

Thank you to our 30+
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.