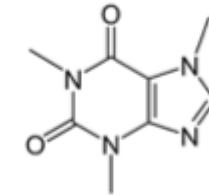


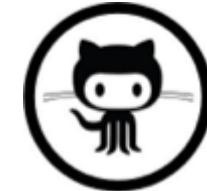
# DIY Deep Learning for Vision: a Hands-On Tutorial with Caffe



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



[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)



[github.com/BVLC/caffe](https://github.com/BVLC/caffe)

Evan Shelhamer, Jeff Donahue,  
Yangqing Jia, Ross Girshick

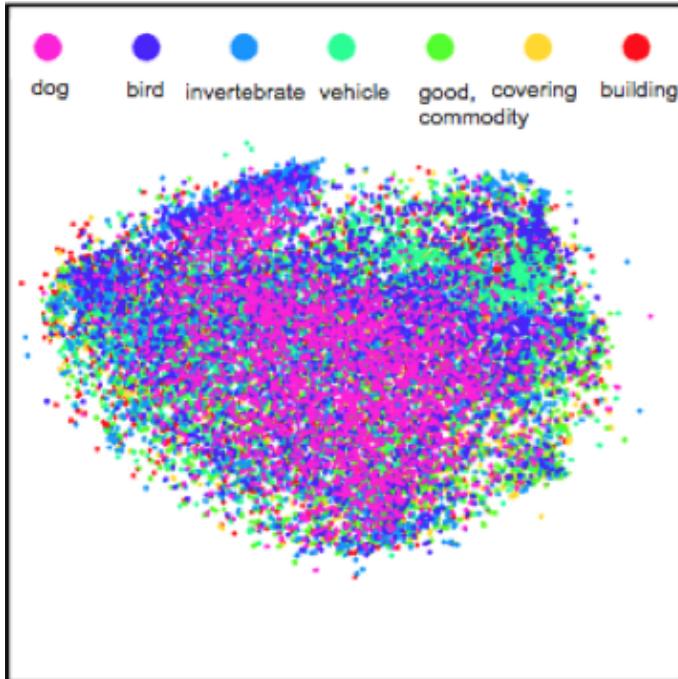
Look for further  
details in the  
outline notes



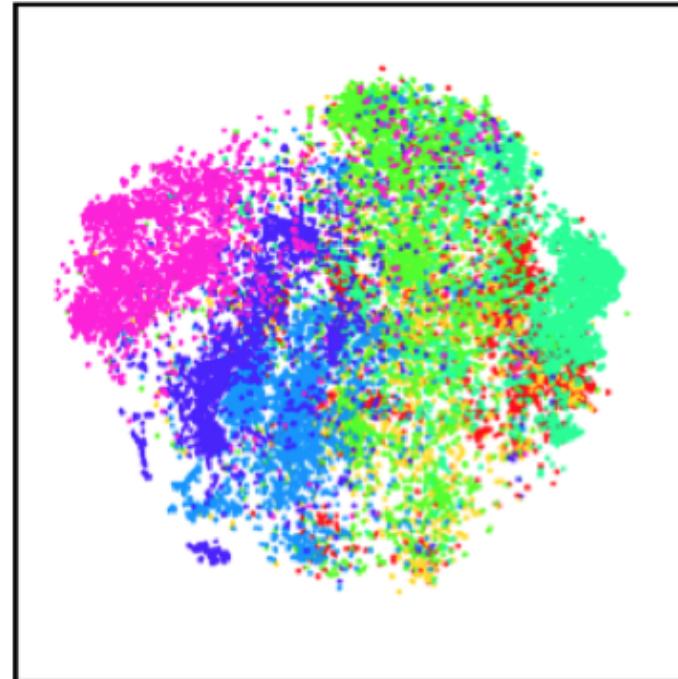
BV  
LC

# Why Deep Learning?

The Unreasonable Effectiveness of Deep Features



Low-level: Pool<sub>1</sub>

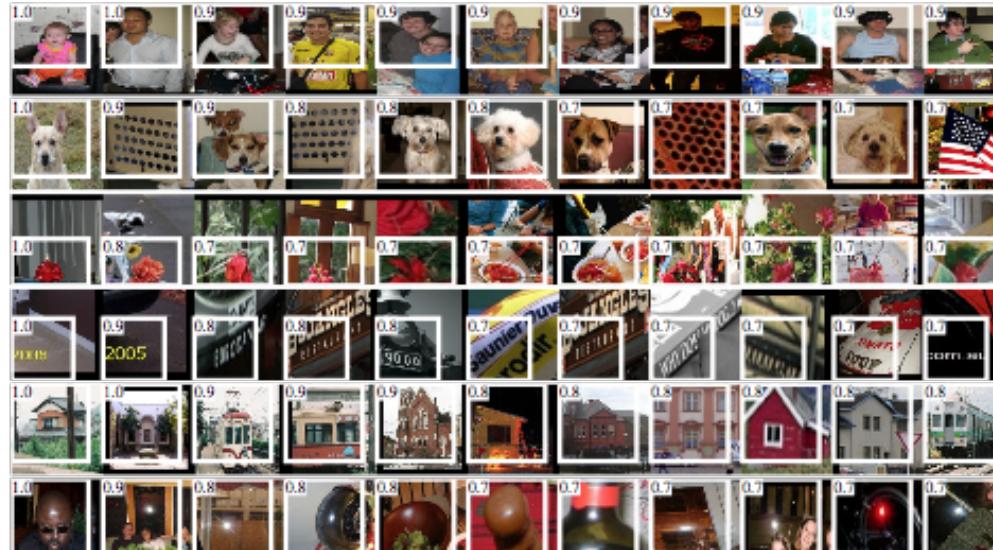


High-level: FC<sub>6</sub>

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

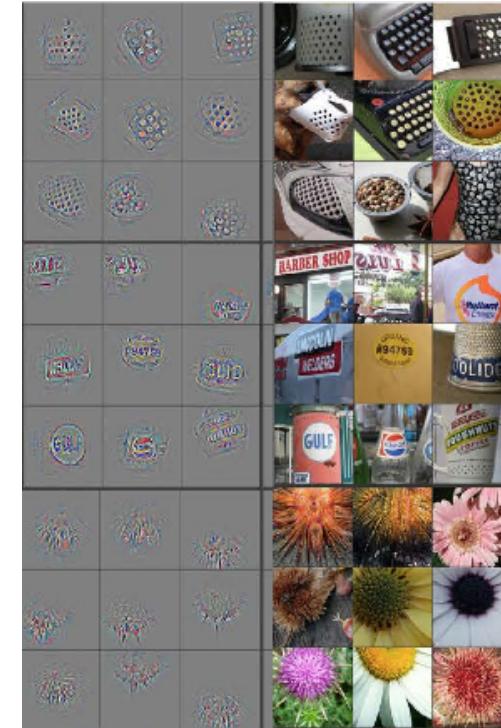
# Why Deep Learning?

## The Unreasonable Effectiveness of Deep Features



Maximal activations of pool<sub>5</sub> units

[R-CNN]



conv<sub>5</sub> DeConv visualization  
[Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

# Why Deep Learning?

The Unreasonable Effectiveness of Deep Features

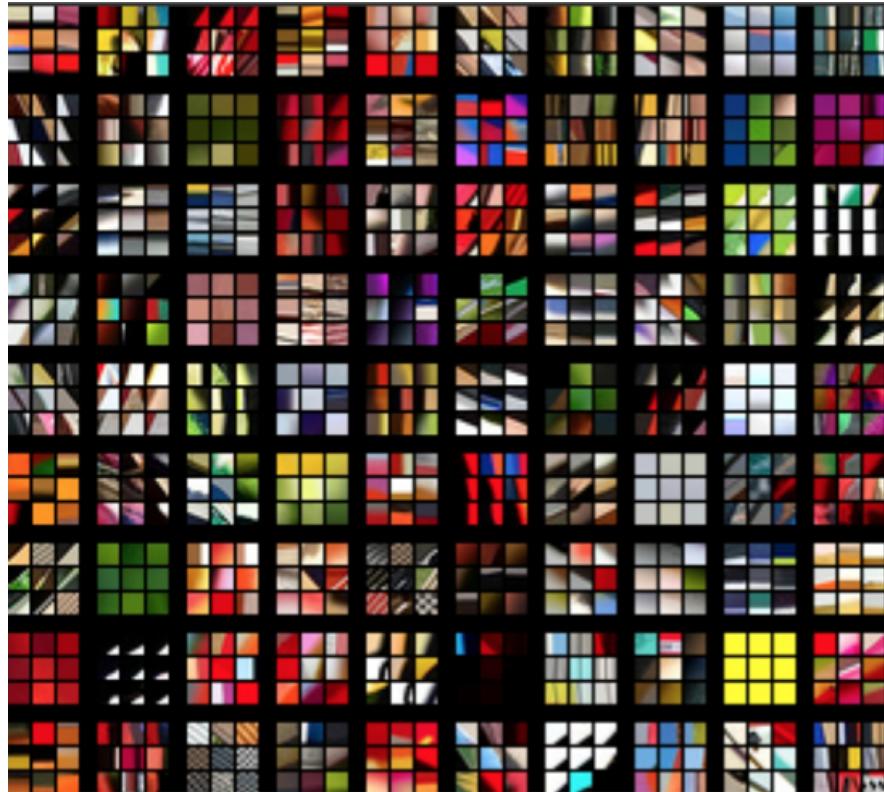
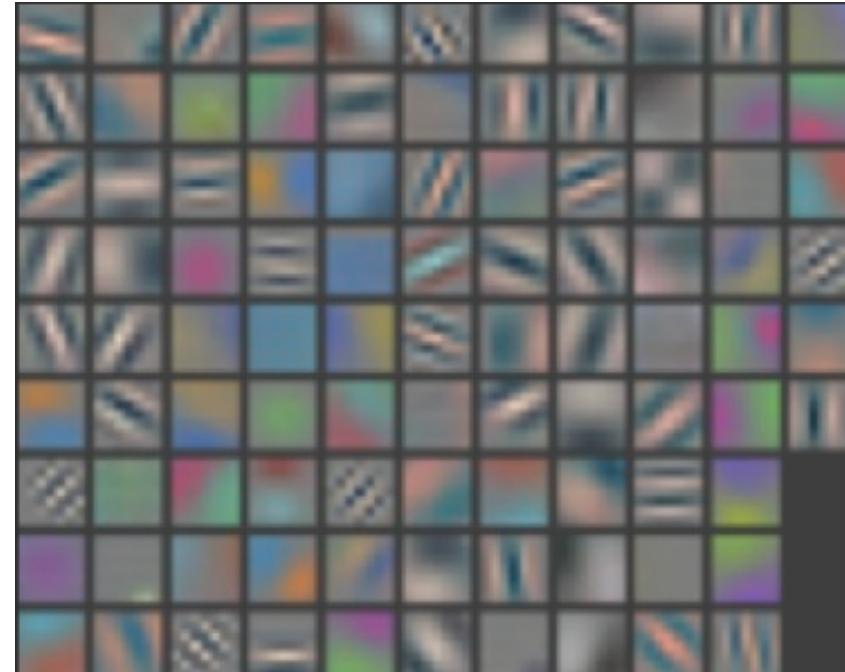


image patches that strongly activate 1st layer filters



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

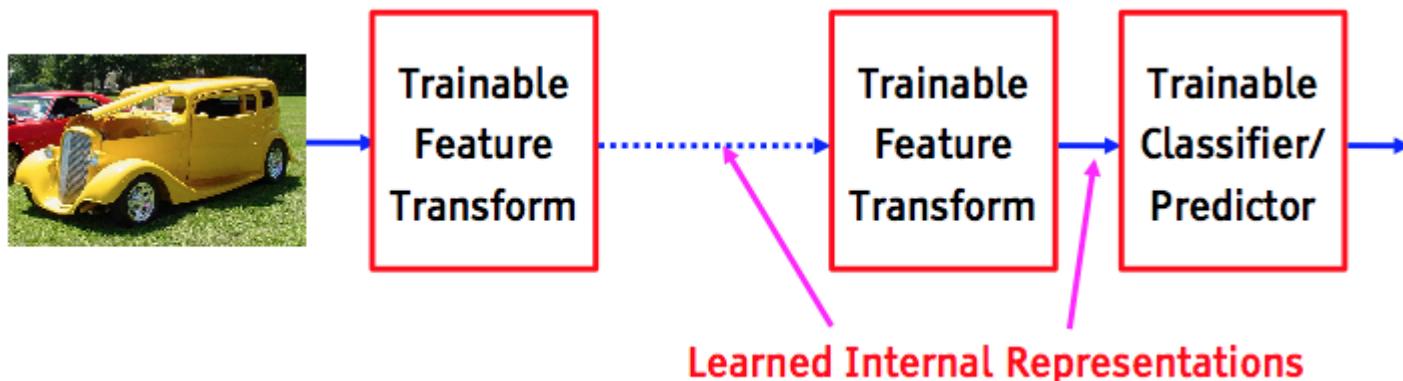
- vision: pixel, motif, part, object
- text: character, word, clause, sentence
- speech: audio, band, phone, word

concrete       abstract  
                  learning

# 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.

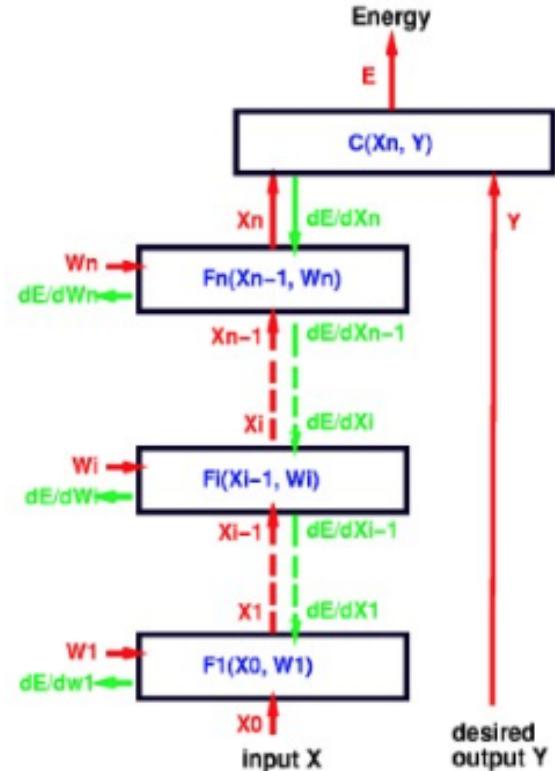
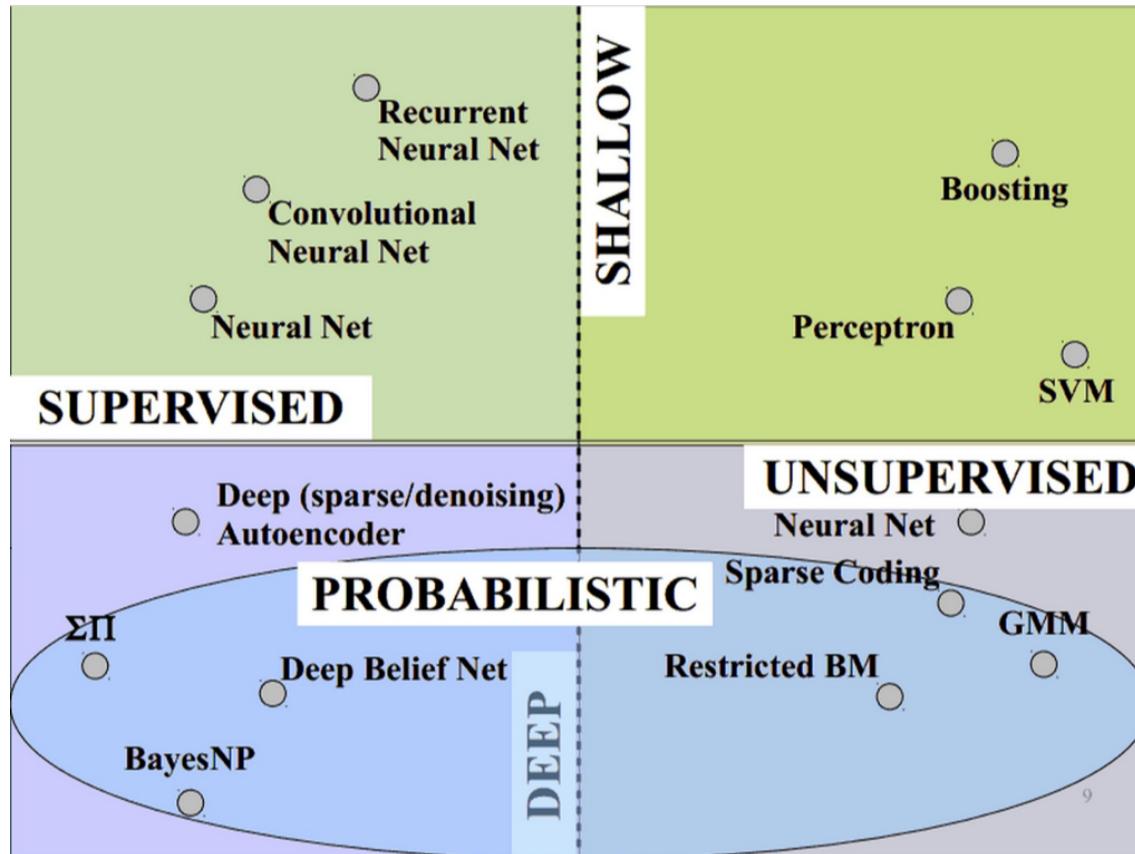


figure credit Yann LeCun, ICML '13 tutorial

# What is Deep Learning?



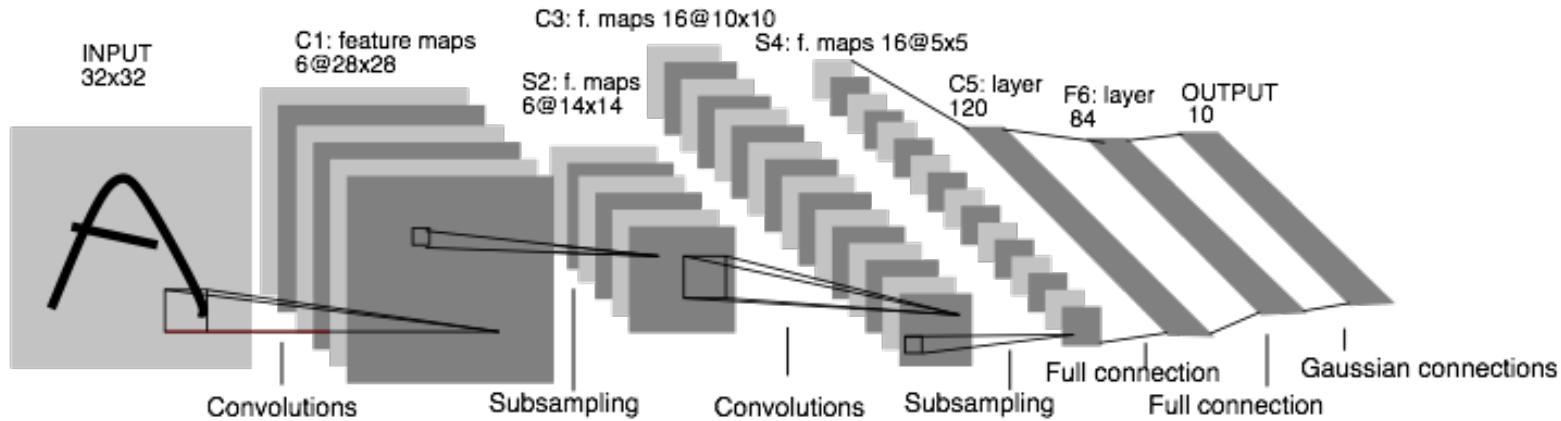
Vast space of models!

Caffe models are loss-driven:

- supervised
- unsupervised

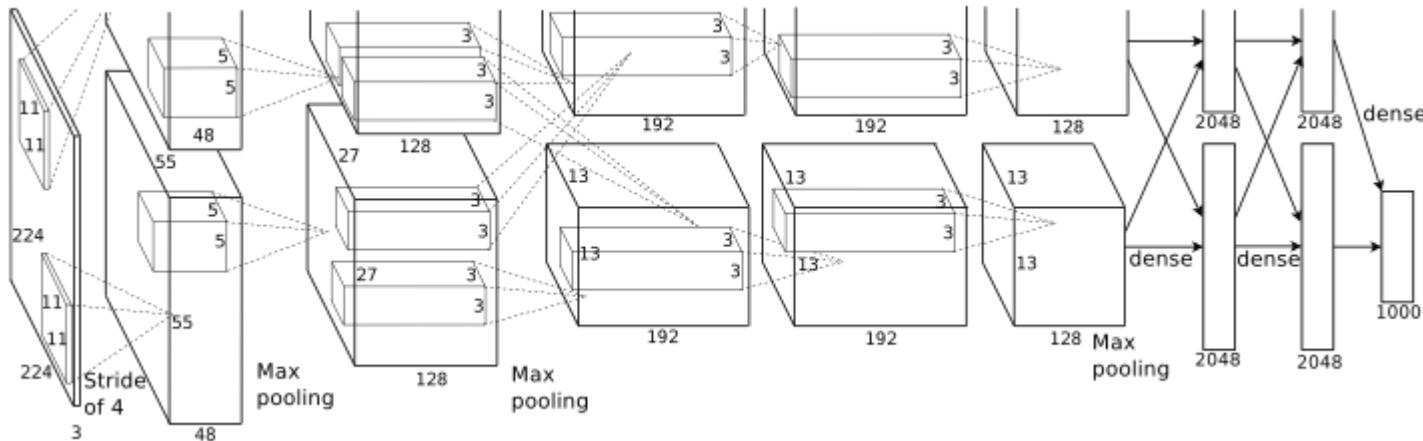
slide credit Marc'aurelio Ranzato,  
CVPR '14 tutorial.

# 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

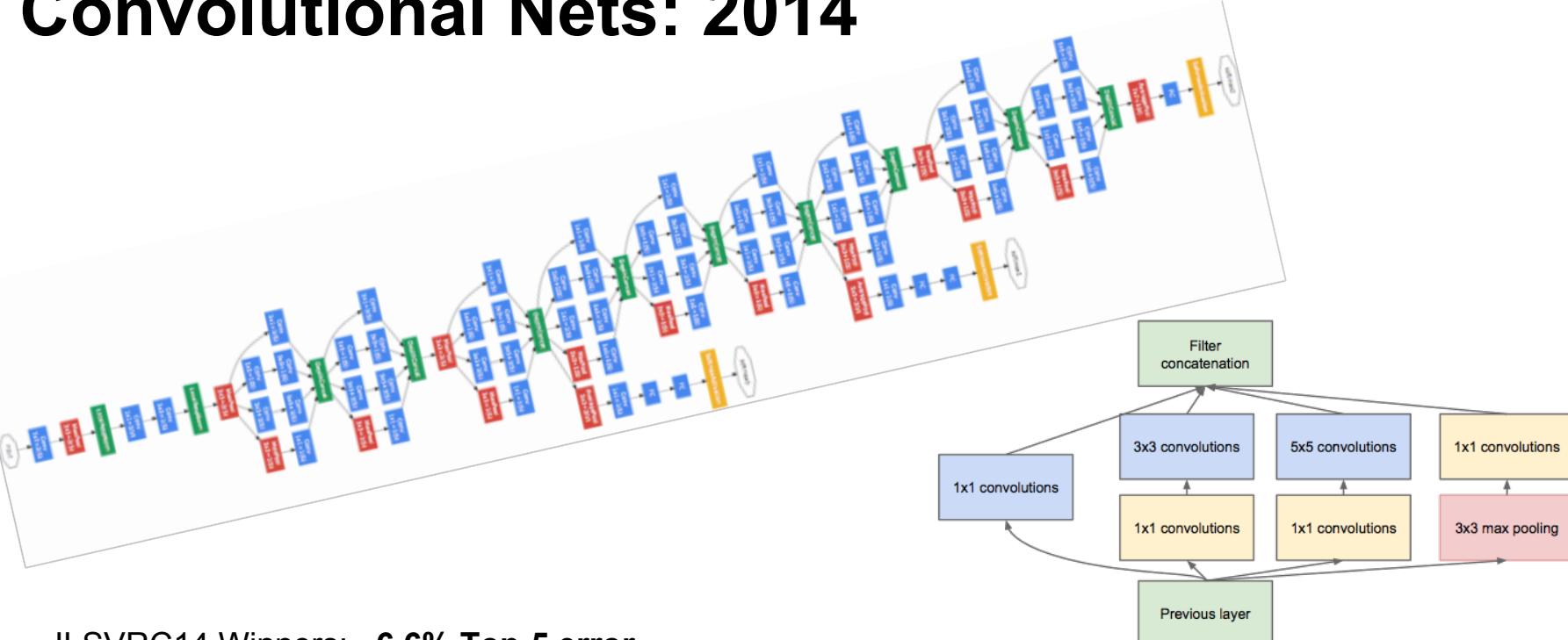
# Convolutional Nets: 2012

parameters		FLOPs
4M	<b>FULL CONNECT</b>	4Mflop
16M	<b>FULL 4096/ReLU</b>	16M
37M	<b>FULL 4096/ReLU</b>	37M
	<b>MAX POOLING</b>	
442K	<b>CONV 3x3/ReLU 256fm</b>	74M
1.3M	<b>CONV 3x3ReLU 384fm</b>	224M
884K	<b>CONV 3x3/ReLU 384fm</b>	149M
	<b>MAX POOLING 2x2sub</b>	
	<b>LOCAL CONTRAST NORM</b>	
307K	<b>CONV 11x11/ReLU 256fm</b>	223M
	<b>MAX POOL 2x2sub</b>	
	<b>LOCAL CONTRAST NORM</b>	
35K	<b>CONV 11x11/ReLU 96fm</b>	105M

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.

# 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 + 3 fully-connected layers

+ depth  
+ data  
+ dimensionality reduction

# Learning about Deep Learning

Refer to the [Tutorial on Deep Learning for Vision](#) from CVPR '14.

- Fundamentals on supervised and unsupervised deep learning by Marc'Aurelio Ranzato
- List of references to explore
- Advanced ideas and current research directions

Pairs well with this tutorial!

# 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](#)



Unwatch ▾ 282

★ Unstar 1,404

Fork 750

October 4, 2014 – November 4, 2014

Period: 1 month ▾

## Overview



60 Active Pull Requests



127 Active Issues

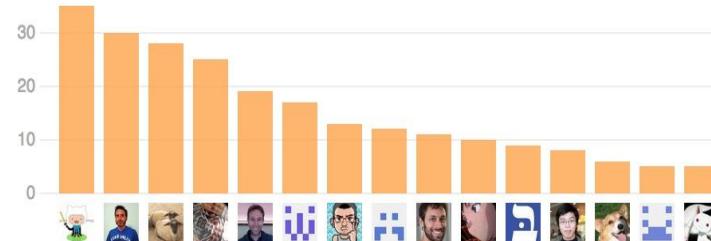
29  
Merged Pull Requests

31  
Proposed Pull Requests

67  
Closed Issues

60  
New Issues

Excluding merges, **36 authors** have pushed **16 commits** to master and **274 commits** to all branches. On master, **9 files** have changed and there have been **59 additions** and **55 deletions**.

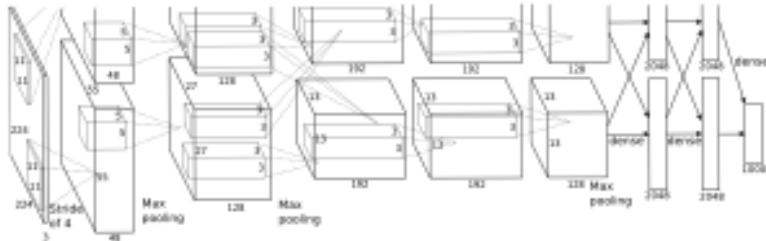


29 Pull requests merged by 13 people

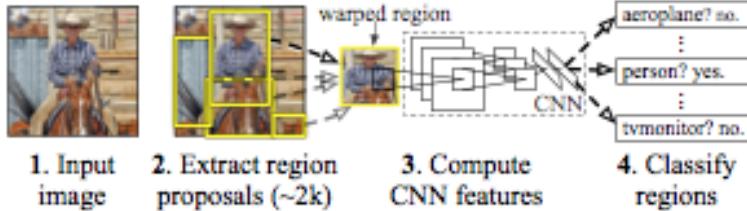


# Reference Models

**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 models are licensed for unrestricted use.

# Open Model Collection

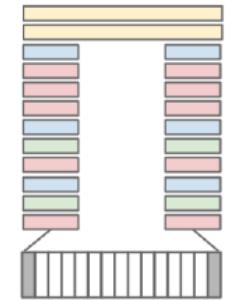
The Caffe [Model Zoo](#)

- 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

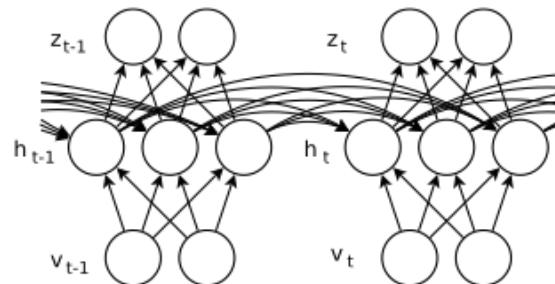
# Architectures

DAGs  
multi-input  
multi-task



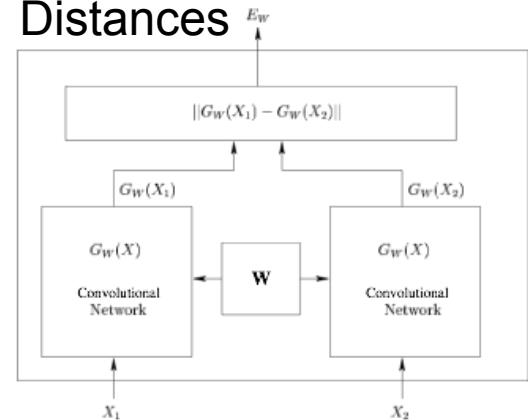
[ Karpathy14 ]

Weight Sharing  
Recurrent (RNNs)  
Sequences



[ Sutskever13 ]

Siamese Nets  
Distances



[ Chopra05 ]

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

# 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\\_hw.html](http://caffe.berkeleyvision.org/performance_hw.html)

# CAFFE INTRO

# Net

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

```
name : "dummy-net"
```

```
layers { name: "data" ... }
```

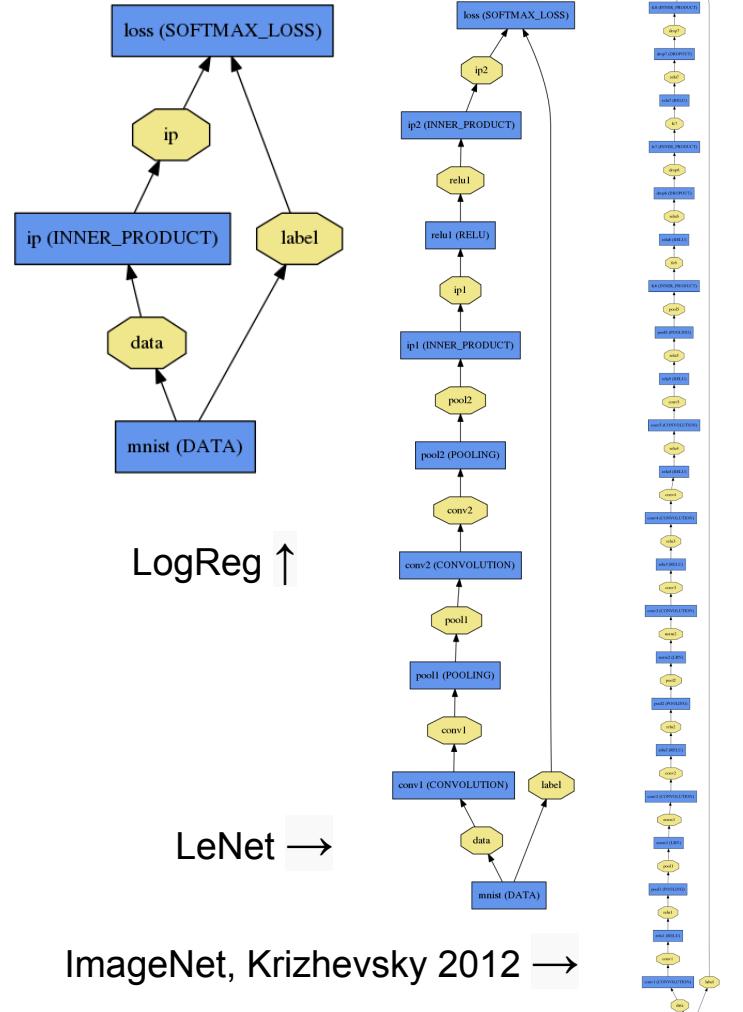
```
layers { name: "conv" ... }
```

```
layers { name: "pool" ... }
```

... more layers ...

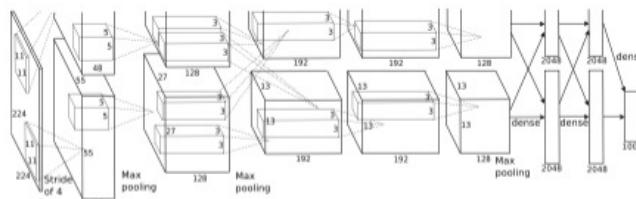
```
layers { name: "loss" ... }
```

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



# Forward / Backward the essential Net computations

Forward:  
inference  $f_W(x)$



“espresso”  
+ loss

$\nabla f_W(x)$  Backward:  
learning

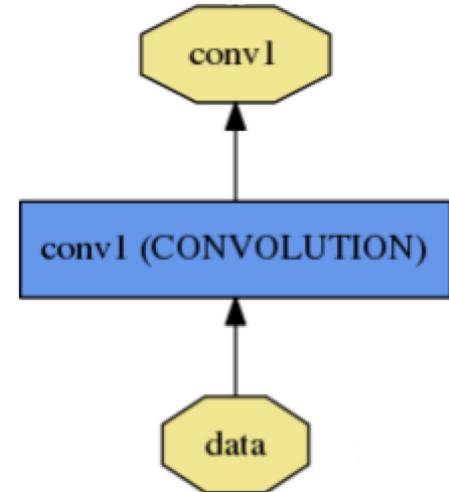
Caffe models are complete machine learning systems for inference and learning.  
The computation follows from the model definition. Define the model and run.

# 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

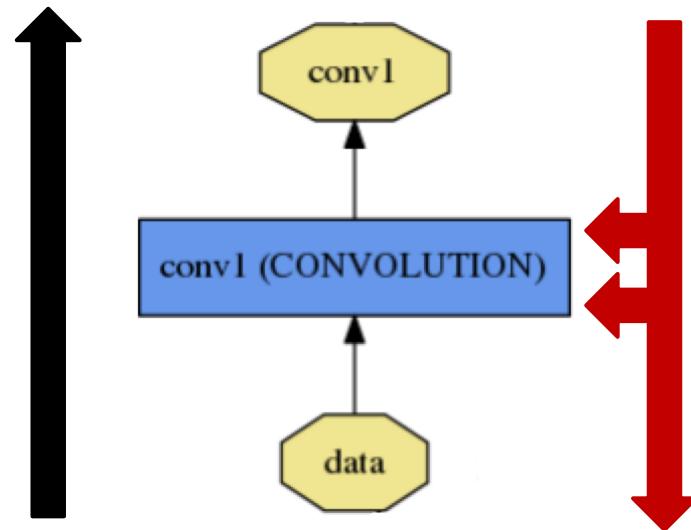
# Layer Protocol

**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)



## *Compositional Modeling*

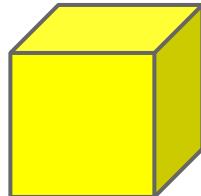
The Net's forward and backward passes  
are composed of the layers' steps.

[Layer Development Checklist](#)

# Blob

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

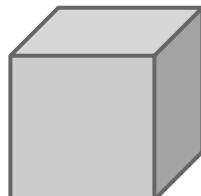
- hold data, derivatives, and parameters
- lazily allocate memory
- shuttle between CPU and GPU



## Data

Number x  $K$  Channel x Height x Width

256 x 3 x 227 x 227 for ImageNet train input



## Parameter: Convolution Weight

$N$  Output x  $K$  Input x Height x Width

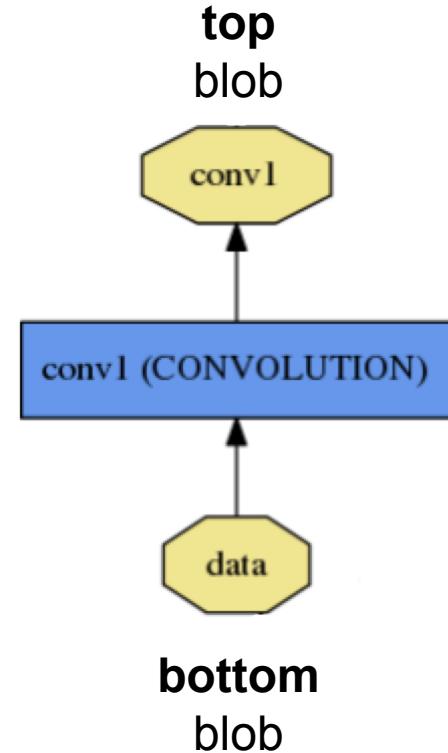
96 x 3 x 11 x 11 for CaffeNet conv1



## Parameter: Convolution Bias

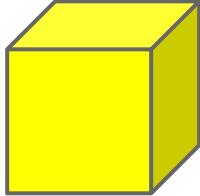
96 x 1 x 1 x 1 for CaffeNet conv1

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



# 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"

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)

# End to End Recipe...

- Convert the data to Caffe-format
  - Imdb, 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](http://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

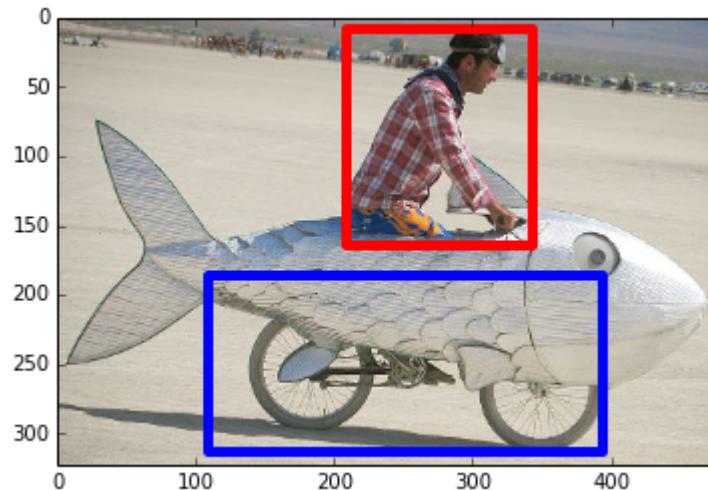
<http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/detection.ipynb>

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).

Ethereal



HDR



Melancholy



Minimal



Other Styles:

[Vintage](#)

[Long Exposure](#)

[Noir](#)

[Pastel](#)

[Macro](#)

... and so on.

# Embedded Caffe

## Caffe on the NVIDIA Jetson TK1 mobile board



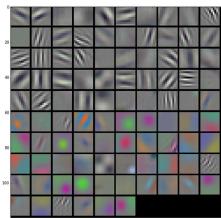
- 10 watts of power
- inference at 35 ms per image
- NVIDIA acceleration just released
- how-to guide  
courtesy of Pete Warden
- cuDNN for TK1 recently released!

# Feature Extraction + Visualization



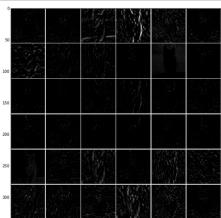
The first layer filters, conv1

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

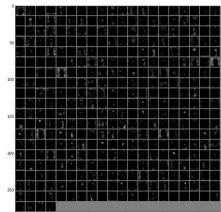


The first layer output, conv1 (rectified responses of the filters above, first 36 only)

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

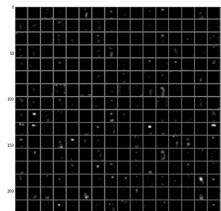


```
In [13]: feat = net.blobs['conv4'].data[4]
vis_square(feat, padval=0.5)
```



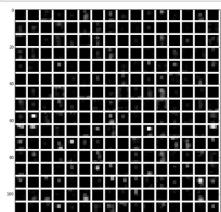
The fifth layer output, conv5 (rectified, all 256 channels)

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

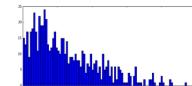
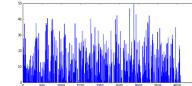


The fifth layer after pooling, pool5

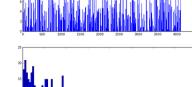
```
In [15]: feat = net.blobs['pool5'].data[4]
vis_square(feat, padval=1)
```



```
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)
```

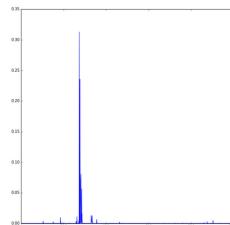


```
In [17]: feat = net.blobs['fc7'].data[4]
plt.subplot(2, 1, 1)
plt.plot(feat.flat)
plt.subplot(2, 1, 2)
_= plt.hist(feat.flat[feat.flat > 0], bins=100)
```

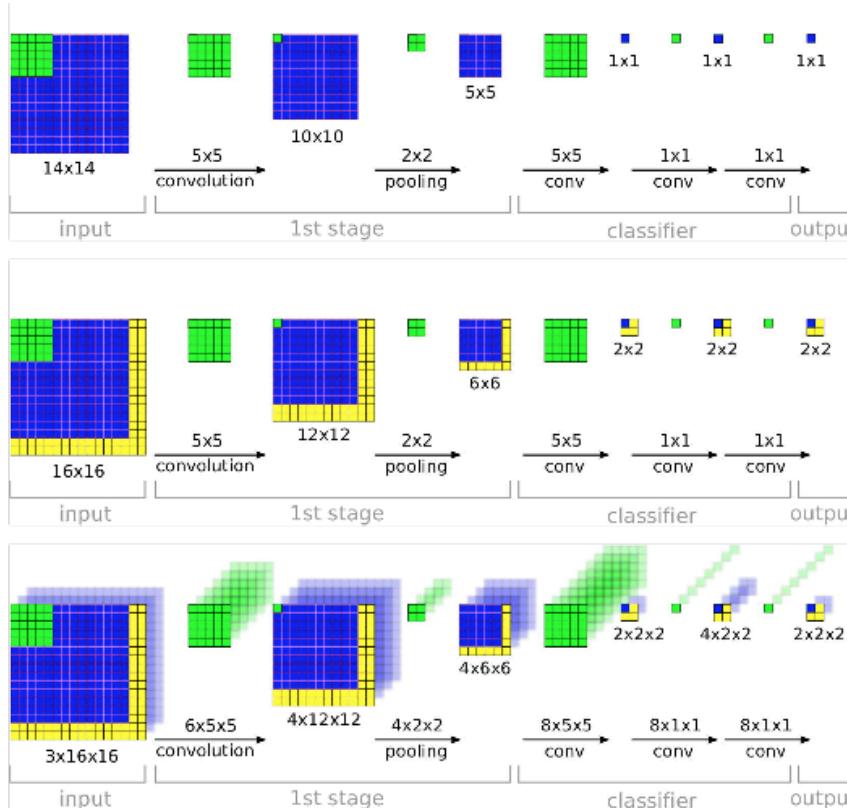


```
In [18]: feat = net.blobs['prob'].data[4]
plt.plot(feat.flat)
```

```
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

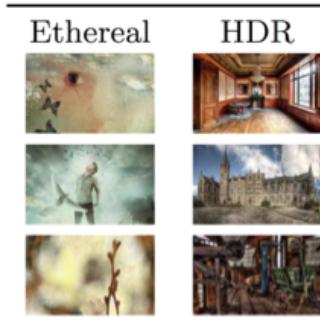
- Take a pre-trained model and fine-tune to new tasks  
[DeCAF] [Zeiler-Fergus] [OverFeat]



dog      bird      invertebrate      vehicle      good, covering commodity      building



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" new name = new params
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 20
    ...
  }
}
```

Input:  
A different source

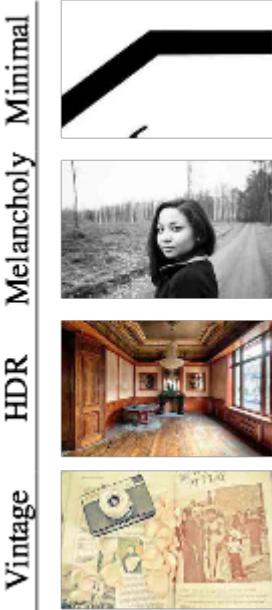
Last Layer:  
A different classifier

# 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);
```



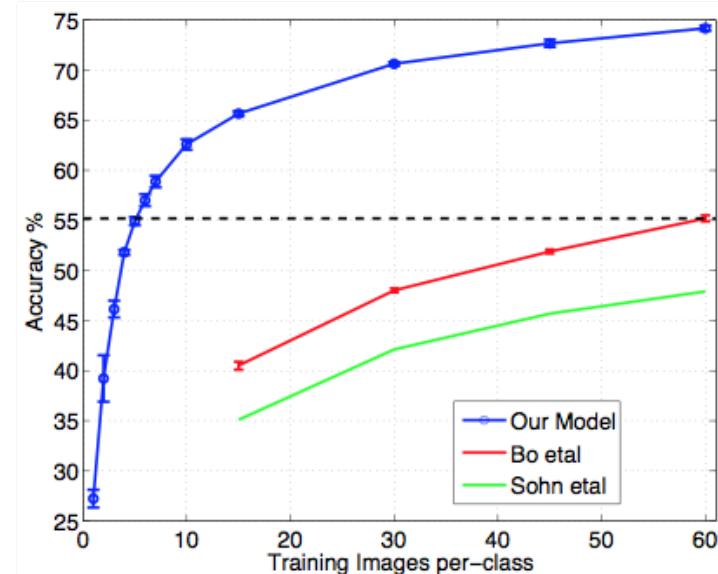
# When to Fine-tune?

A good first step!

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

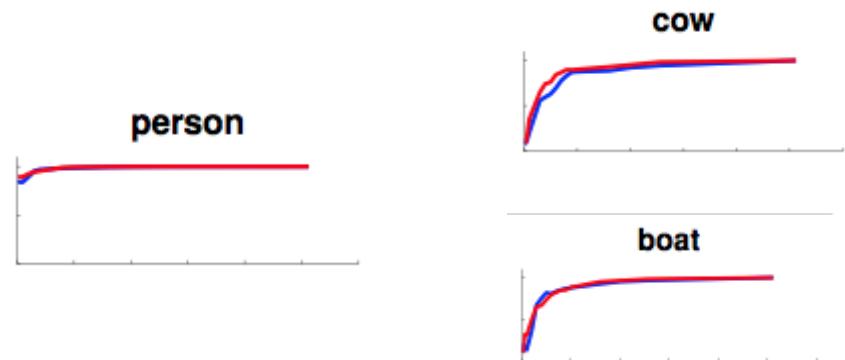
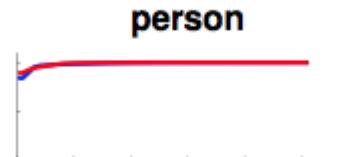
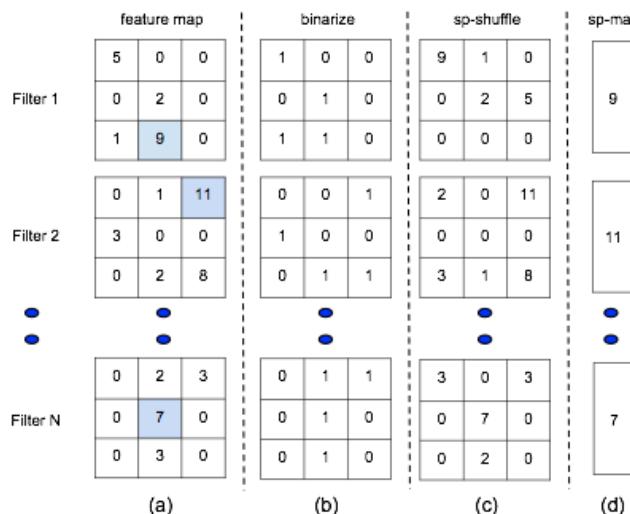
State-of-the-art results in

- recognition
- detection
- segmentation



# Training & Fine-tuning Analysis

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



# 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

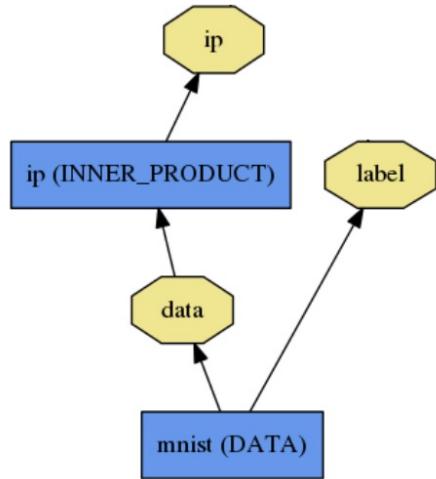
(Example)

Fine-tuning from ImageNet to Style

# **LOSS**

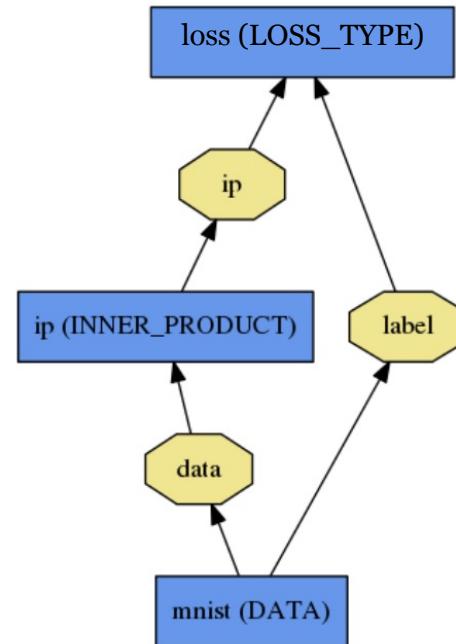
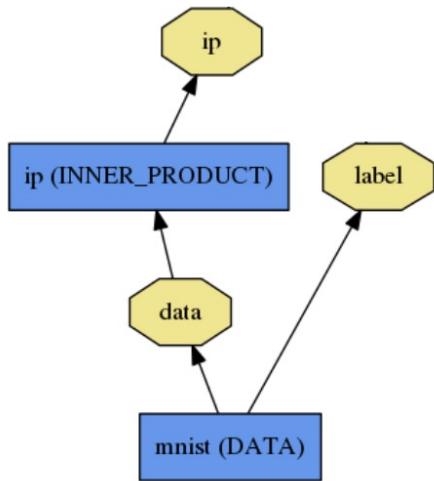
# Loss

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:

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

Data term: error averaged  
over instances

Regularization  
term: penalize  
large weights to  
improve  
generalization

# Loss

- The data error term  $f_W(X^{(i)})$  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"
}
```

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

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

# 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"
}
```

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

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

# 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:

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

```
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"
}
```

# 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

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

```
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"
    loss_weight: 100.0
}
```

# Any layer can produce a loss!

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

$$E = \|\text{pred} - \text{label}\|_2^2 / (2N)$$

```
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 \|_2^2 / (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  $D$

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

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

# Solver

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

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

# 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.

# 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

```
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

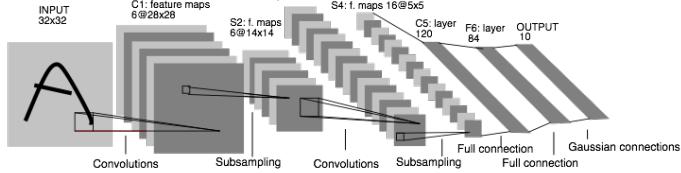
```
I0901 13:35:20.426187 20072 solver.cpp:232] Iteration 65000, loss = 61.5498
I0901 13:35:20.426218 20072 solver.cpp:251] Iteration 65000, Testing net (#0) # train set
I0901 13:35:22.780092 20072 solver.cpp:289] Test loss: 60.8301
I0901 13:35:22.780138 20072 solver.cpp:302]      Test net output #0: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

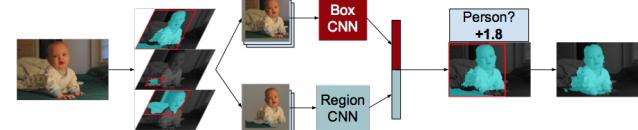
# DAG

Many current deep models have linear structure

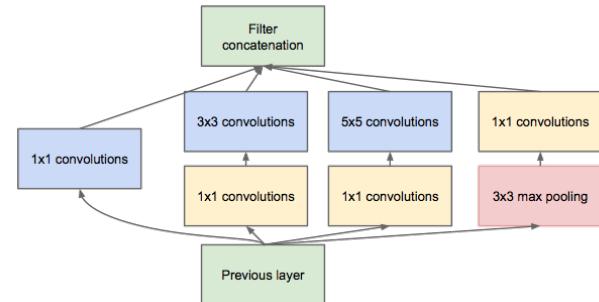


but Caffe nets can have any directed acyclic graph (DAG) structure.

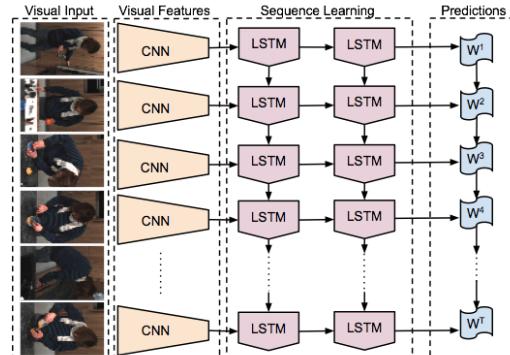
Define bottoms and tops and Caffe will connect the net.



SDS two-stream net



GoogLeNet Inception Module



LRCN joint vision-sequence model

# **WEIGHT SHARING**

# 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

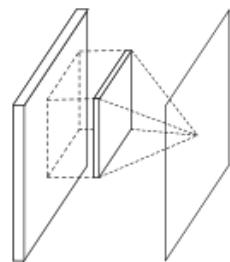
```
layers: {  
    name: 'innerproduct1'  
    type: INNER_PRODUCT  
    inner_product_param {  
        num_output: 10  
        bias_term: false  
        weight_filler {  
            type: 'gaussian'  
            std: 10  
        }  
    }  
    param: 'sharedweights'  
    bottom: 'data'  
 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

# RECENT MODELS

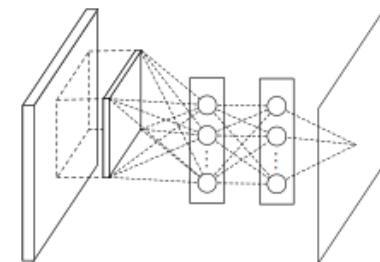
- Network-in-Network (NIN)
- GoogLeNet
- VGG

# Network-in-Network

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

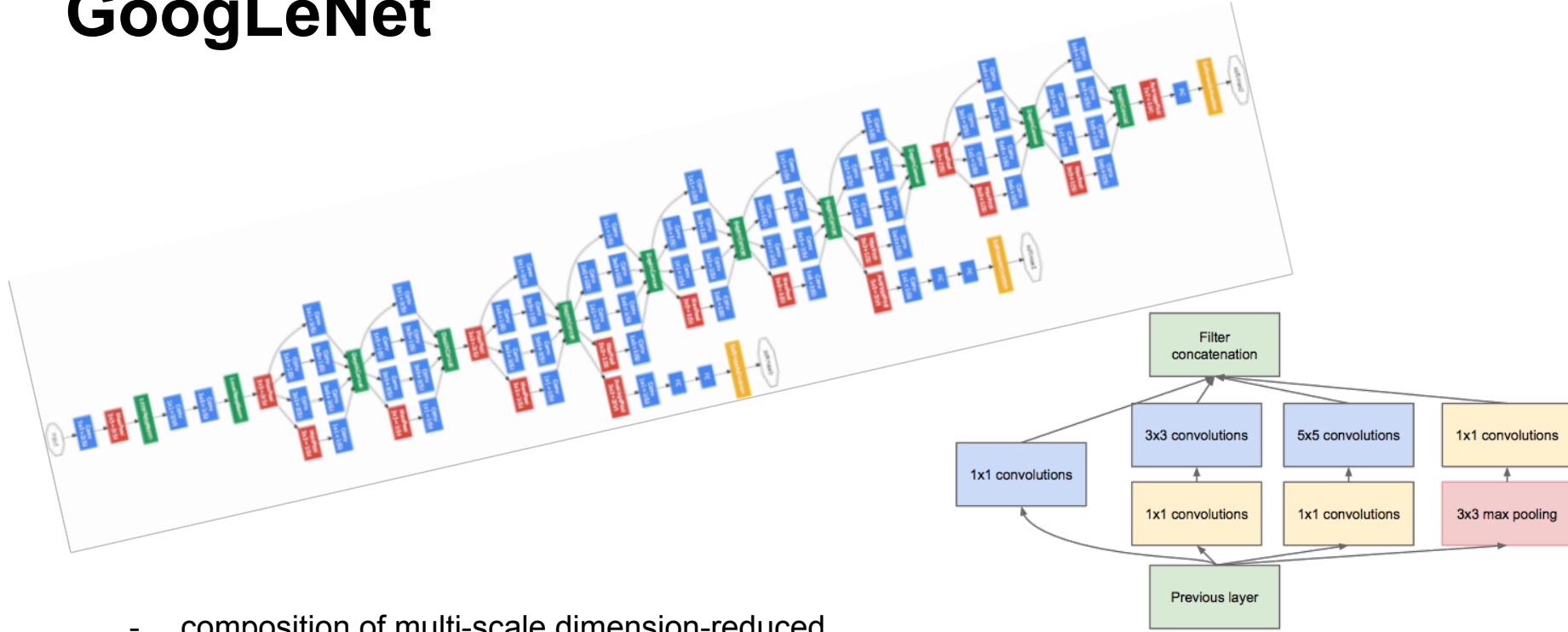


Linear Filter  
CONV



NIN / MLP filter  
 $1 \times 1$  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

# 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 ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	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 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
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 **done in rc2**
- Fully Convolutional Networks **PRs open**
- Sequences **PRs open**
- cuDNN v2 **PR #1731**
- Gradient Accumulation **PR #1663**
- 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 done in #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

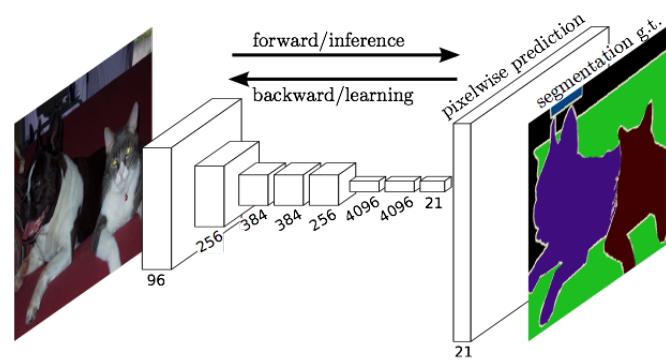
A framework for pixel prediction by conv. net applied to semantic segmentation

- 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

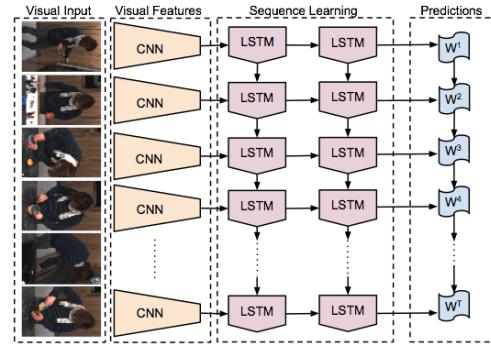
- video
- language
- dynamics

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.**

# 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



# Gradient Accumulation #1663

- 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

# 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.