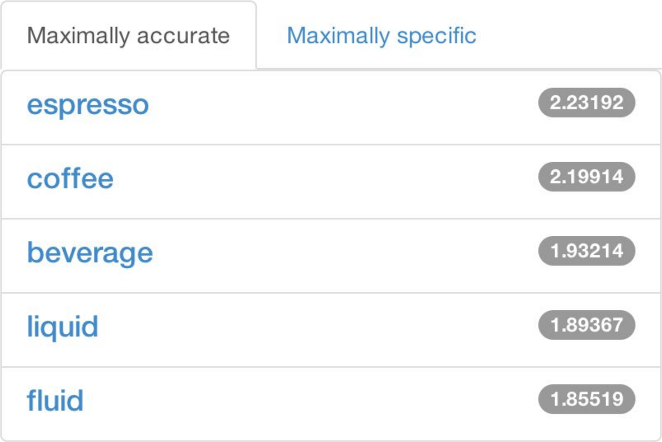
# Brewing Deep Neural Networks with Caffe



Maximally accurate | Maximally specific

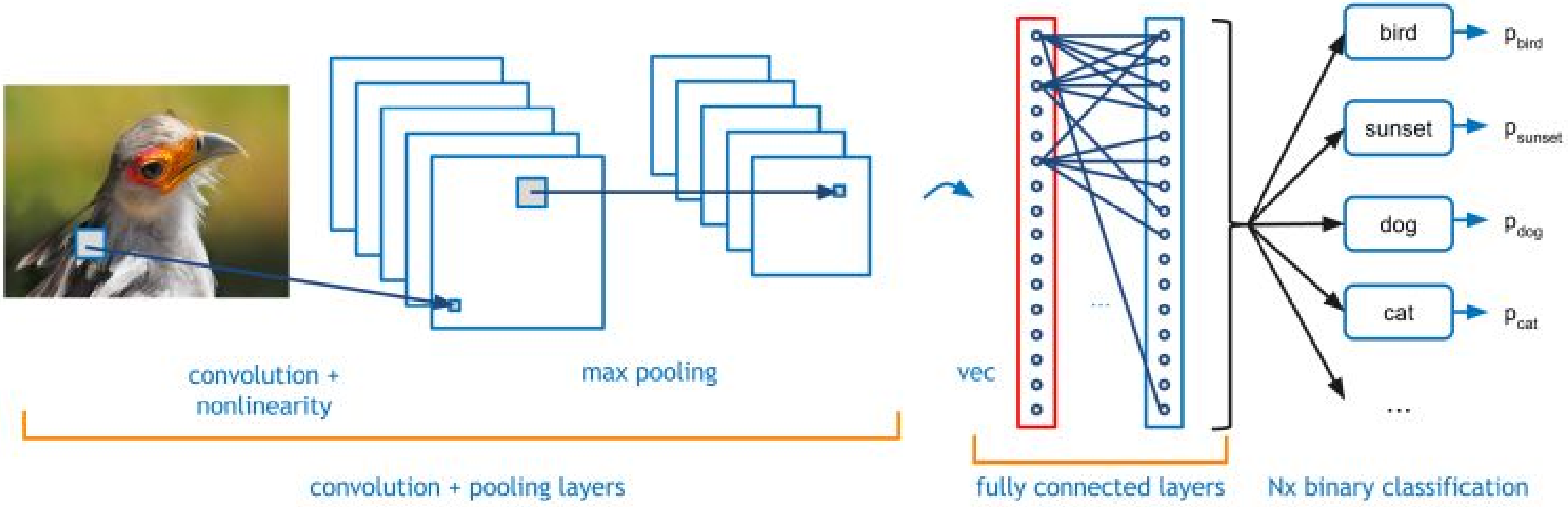
| | |
|---|---|
| espresso | 2.23192 |
| coffee | 2.19914 |
| beverage | 1.93214 |
| liquid | 1.89367 |
| fluid | 1.85519 |

Abhimanyu Dubey

ELL881/ELL784

Slides at http://iitd.info/hpccaffe

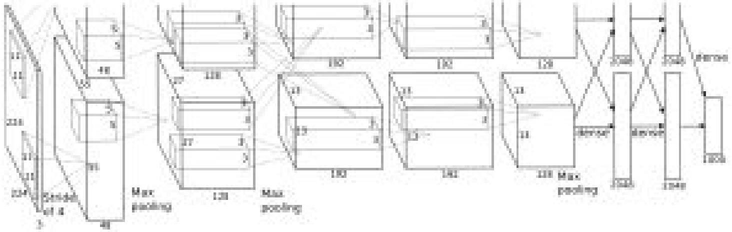
Adapted from original caffe slides



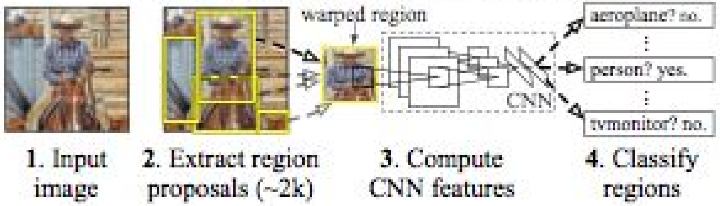
All in a day’s work with Caffe

# Reference Models

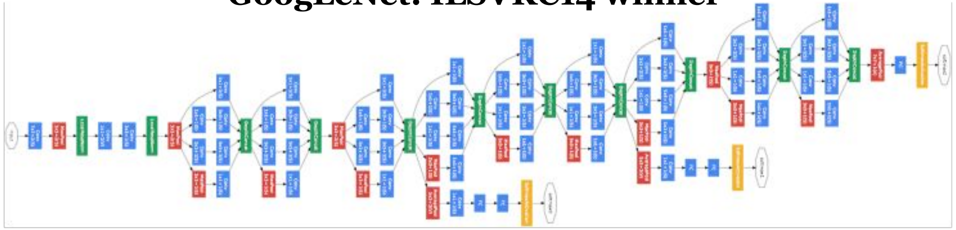
**AlexNet: ImageNet Classification**



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



**GoogLeNet: ILSVRC14 winner**



Caffe offers the

- model definitions
- optimization settings
- pre-trained weights

so you can start right away.

The BVLC models are licensed for unrestricted use.

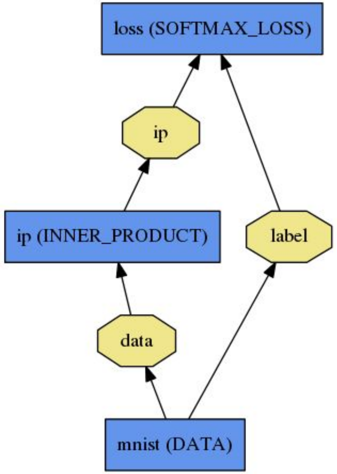
The community shares models in the Model Zoo.

# Net

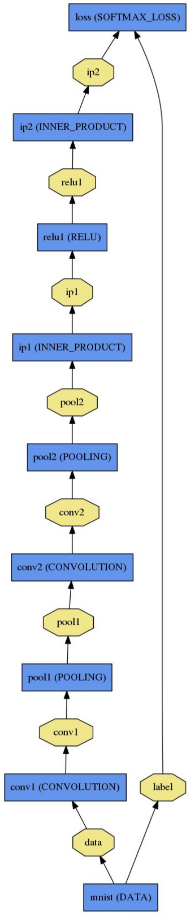
- A network is a set of layers and their connections:

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

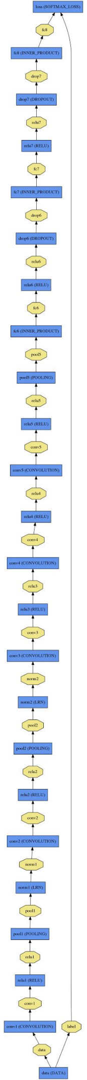
- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – an array interface



LogReg ↑



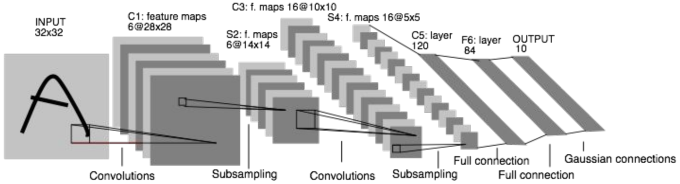
LeNet →



ImageNet, Krizhevsky 2012 →

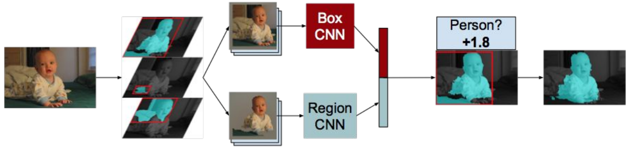
# DAG

Many current deep models have linear structure

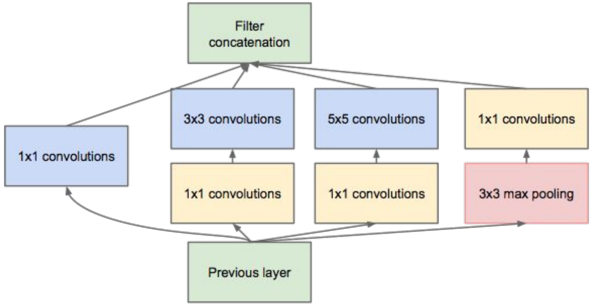


but Caffe nets can have any directed acyclic graph (DAG) structure.

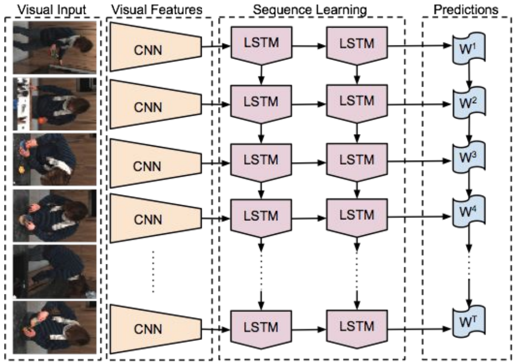
Define `bottoms` and `tops`
and Caffe will connect the net.



SDS two-stream net



GoogLeNet Inception Module



LRCN joint vision-sequence model

# Layer Protocol

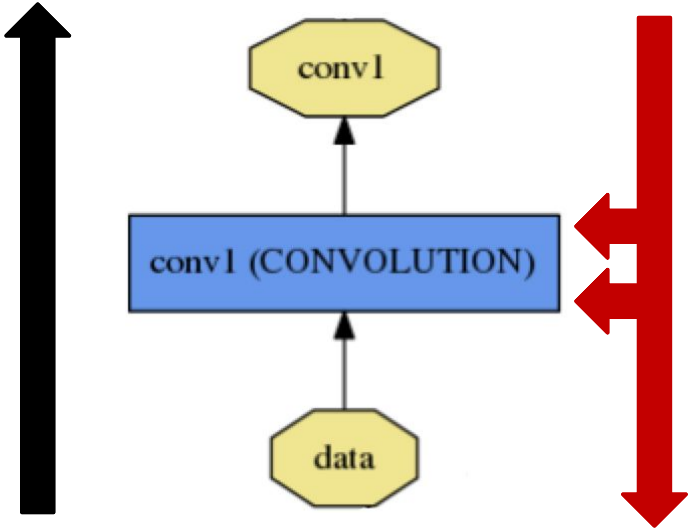
Setup: run once for initialization.

**Forward**: make output given input.

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

Reshape: set dimensions.

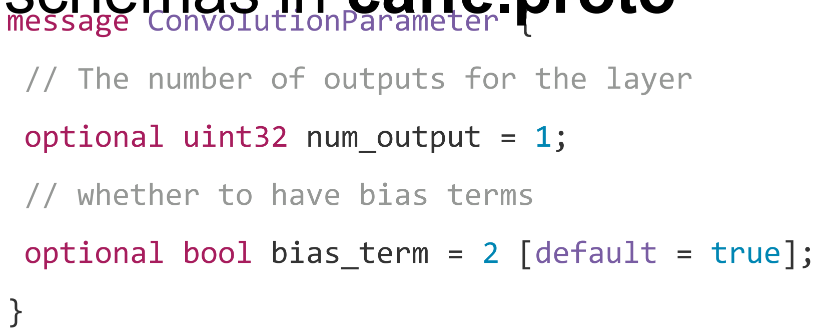
*Compositional Modeling*
The Net’s forward and backward passes are composed of the layers’ steps.



Layer Development Checklist

# Protobuf Model Format

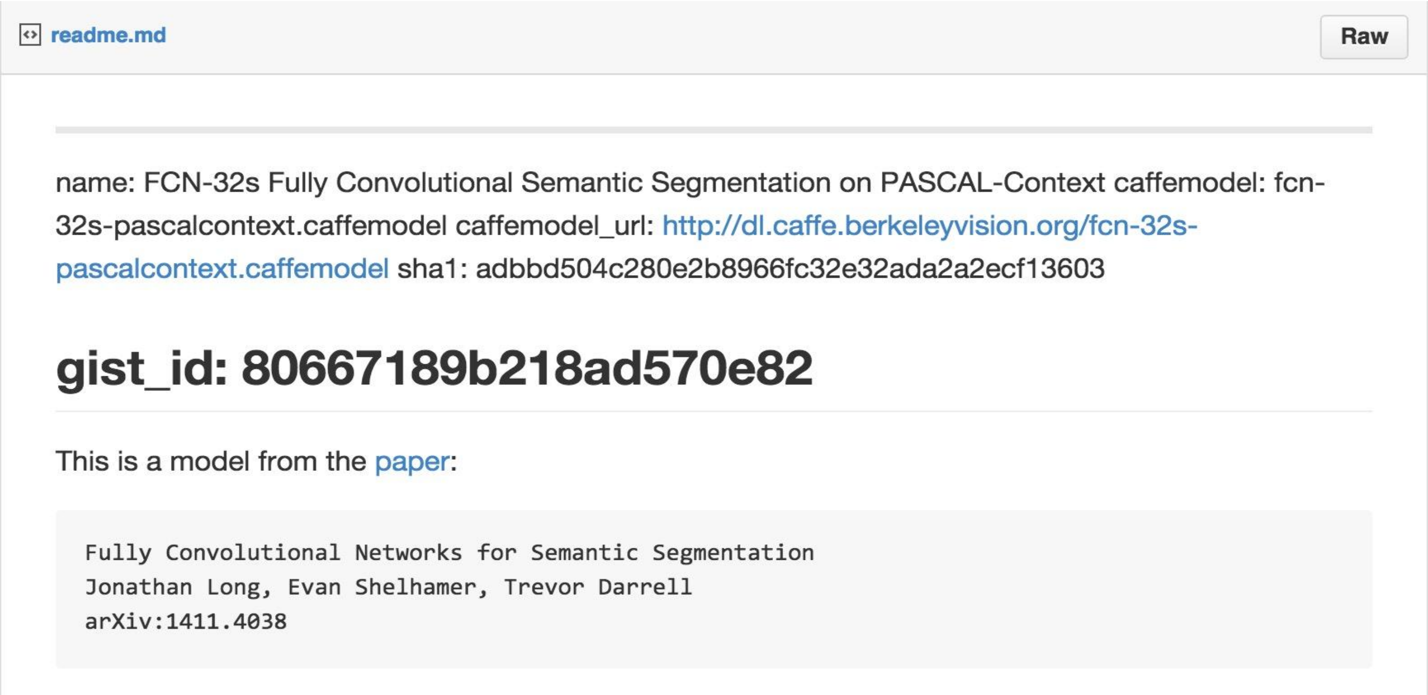
- Strongly typed format
- Auto-generates code
- Developed by Google
- Defines Net / Layer / Solver schemas in **caffe.proto**



```
message ConvolutionParameter {
 // The number of outputs for the layer
 optional uint32 num_output = 1;
 // whether to have bias terms
 optional bool bias_term = 2 [default = true];
}
```

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

# Model Zoo Format



name: FCN-32s Fully Convolutional Semantic Segmentation on PASCAL-Context caffemodel: fcn-32s-pascalcontext.caffemodel caffemodel_url: http://dl.caffe.berkeleyvision.org/fcn-32s-pascalcontext.caffemodel sha1: adbbd504c280e2b8966fc32e32ada2a2ecf13603

## gist_id: 80667189b218ad570e82

This is a model from the paper:

```
Fully Convolutional Networks for Semantic Segmentation
Jonathan Long, Evan Shelhamer, Trevor Darrell
arXiv:1411.4038
```

Gists on github hold model definition, license, url for weights, and hash of Caffe commit that guarantees compatibility.

# Solving: Training a Net

Optimization like model definition is configuration.

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

All you need to run things on the GPU.

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

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

# Solver Showdown: MNIST Autoencoder

## AdaGrad

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

## SGD

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

## Nesterov

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

# 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

```
layer: {
  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'
}
layer: {
  name: 'innerproduct2'
  type: INNER_PRODUCT
  inner_product_param {
    num_output: 10
    bias_term: false
  }
  param: 'sharedweights'
  bottom: 'data'
  top: 'innerproduct2'
}
```

# Recipe for Brewing

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

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

# Brewing Models

from logistic regression to non-linearity

see notebook

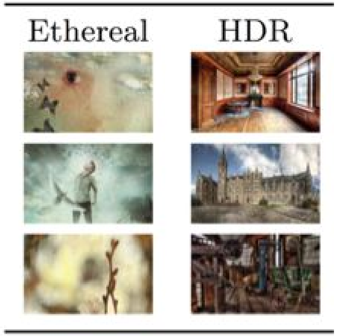
# Take a pre-trained model and fine-tune to new tasks

[DeCAF] [Zeiler-Fergus] [OverFeat]

Lots of Data

image by Andrej Karpathy

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 model definition

```
layer {
  name: "data"
  type: "Data"
  data_param {
    source: "ilsvrc12_train_lmdb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layer {
  name: "fc8"
  type: "InnerProduct"
  inner_product_param {
    num_output: 1000
    ...
  }
}
```

```
layer {
  name: "data"
  type: "Data"
  data_param {
    source: "style_train_lmdb"
    mean_file: "../../data/ilsvrc12"
    ...
  }
  ...
}
...
layer {
  name: "fc8-style"
  type: "InnerProduct"
  inner_product_param {
    num_output: 20
    ...
  }
}
```

new name =
new params

Input:
A different source

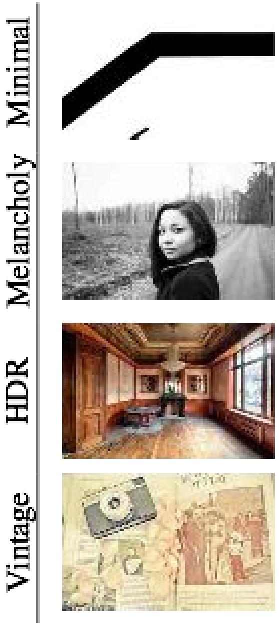
Last Layer:
A different classifier

# From ImageNet to Style

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

Step-by-step in pycaffe:

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



# Fine-tuning

transferring features to style recognition

see notebook

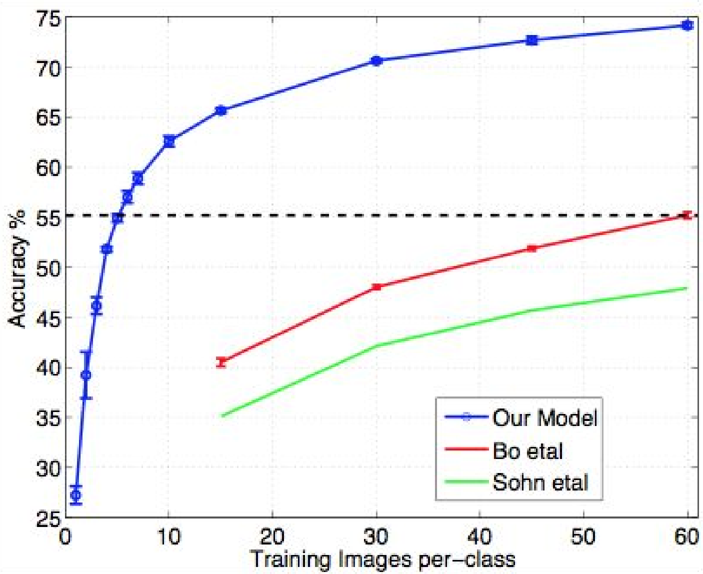
# When to Fine-tune?

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]

# Fine-tuning Tricks

## Learn the last layer first

- Caffe layers have local learning rates: `param { lr_mult: 1 }`
- Freeze all but the last layer for fast optimization
  and avoiding early divergence by setting `lr_mult: 0`
  to fix a parameter.
- Stop if good enough, or keep fine-tuning

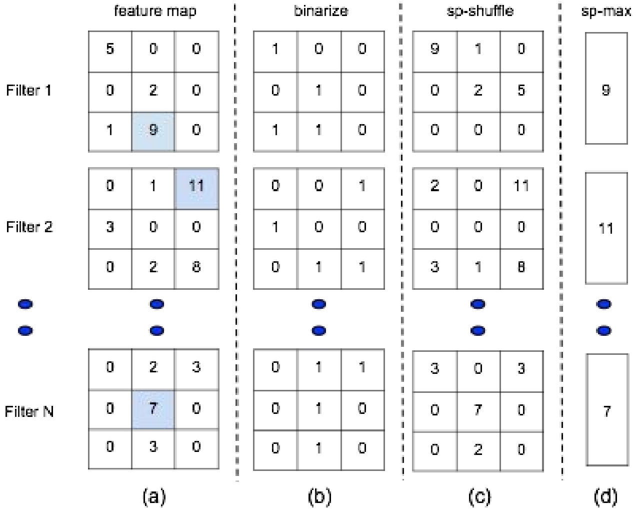
## Reduce the learning rate

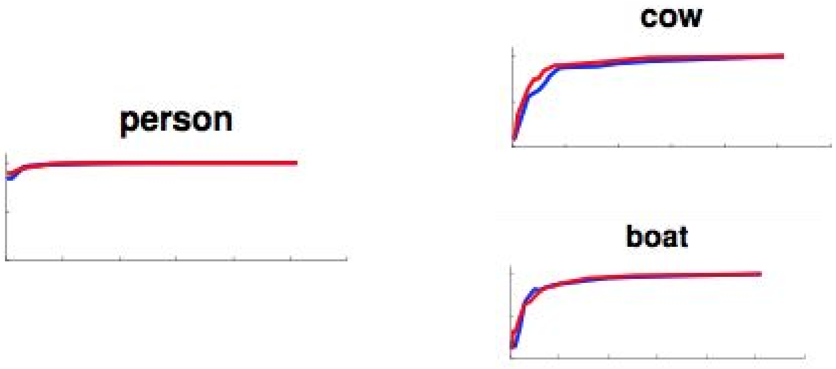
- Drop the solver learning rate by 10x, 100x
- Preserve the initialization from pre-training and avoid divergence

## Do net surgery see notebook on editing model parameters

# After fine-tuning

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





*P. Agarwal et al. ECCV 14*

# Running Caffe on HPC

# HPC

- High performance distributed computing system (http://supercomputing.iitd.ac.in)
- 422 compute nodes, 161 of which are GPU nodes
- 4 general (CPU) login nodes and 2 GPU login nodes

# HPC continued

- Login nodes aren't meant for running jobs - so please don't (they'll be killed anyway)
- Jobs are handled by a scheduling system known as PBS (Portable Batch System)
- PBS handles the distribution and scheduling of jobs (more at http://supercomputing.iitd.ac.in/?pbs )

# HPC continued

- Two storage quotas
  - 30 GB on home (~)
  - 200 TB on scratch (/scratch)
- Hack to make storage simpler
  - Create a symlink from your scratch folder to a folder in your home directory
  - ln -s /scratch/ee/btech/ee2110061/ /home/ee/btech/ee2110061/scratch

# HPC continued

- Keep all datasets,models and logs in scratch (duh)
- Use screen ( https://www.rackaid.com/blog/linux-screen-tutorial-and-how-to/) to keep your sessions alive
- Don’t use nohup - very easy to forget to kill and then eventually use up all disk space with a huge nohup file
- To have internet access, proxy login is required -
    - lynx is a CLI browser you can use

# SSH keys

- Required only for passwordless login - that’s it
- Straightforward steps:
    - On your laptop/host machine:
        - ssh-keygen -t rsa #(keep pressing enter)
        - ssh-add
        - cat ~/.ssh/id_rsa.pub
        - #Copy EVERYTHING that is outputted
    - On HPC (after login - yes, using your password)
        - echo “<paste-here>” >> ~/.ssh/authorized_keys
    - Logout and login again without a password

# Caffe on HPC

- Caffe’s available as a module (http://modules.sourceforge.net)
- Caffe source isn’t accessible on HPC - only executables and tests are kept - but that’s all you need ATM
- Remember to load Caffe as part of the job submission script

# Caffe on HPC

- To load Caffe - module load apps/Caffe (small c for CPU-only - apps/caffe)
- Check where Caffe executables are by running
  - echo $CAFFE_ROOT/bin
- You can try out Caffe by running the examples - you will have to modify the training scripts though

# pyCaffe

- Pycaffe is loaded at $CAFFE_ROOT/python
- To use “import caffe” on HPC, run
    - mkdir -p /home/ee/btech/ee2110061/.local/lib/python2.7/site-packages/ #(if you’ve not done this before)
    - ln -s $CAFFE_ROOT/python /home/ee/btech/ee2110061/.local/lib/python2.7/site-packages/caffe