Kaggle trainings
Introduction to Caffe

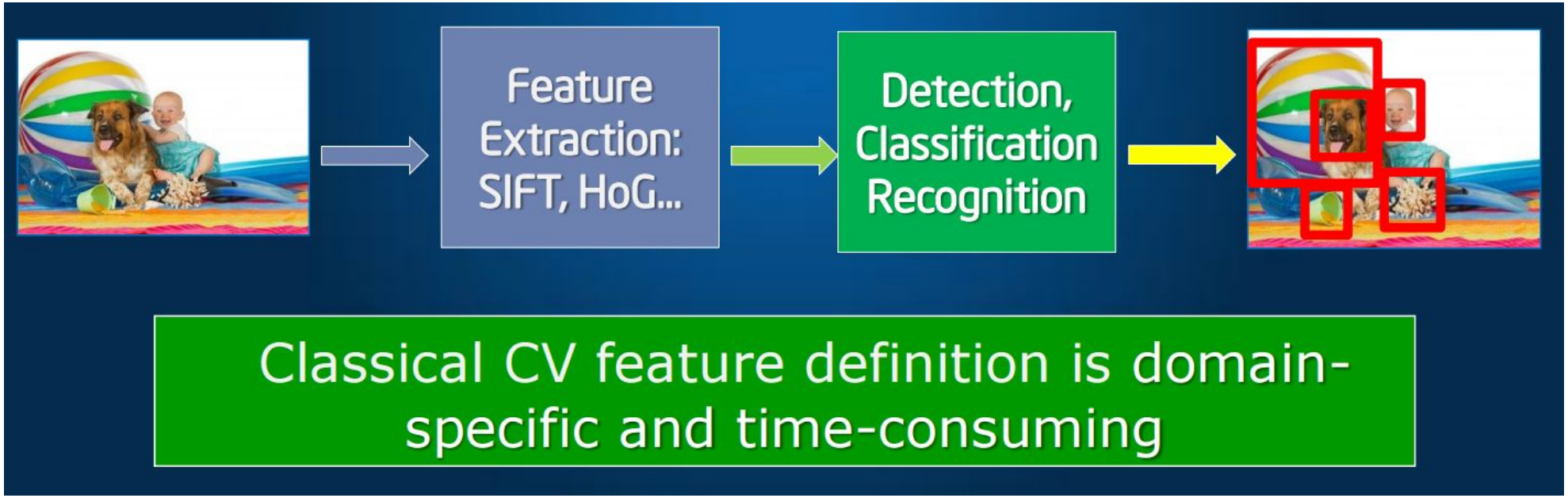
Maximally accurate | Maximally specific

espresso	2.23192
coffee	2.19914
beverage	1.93214
liquid	1.89367
fluid	1.85519

Yurii Pashchenko, 2016, Kiev

Classical Computer Vision Pipeline.

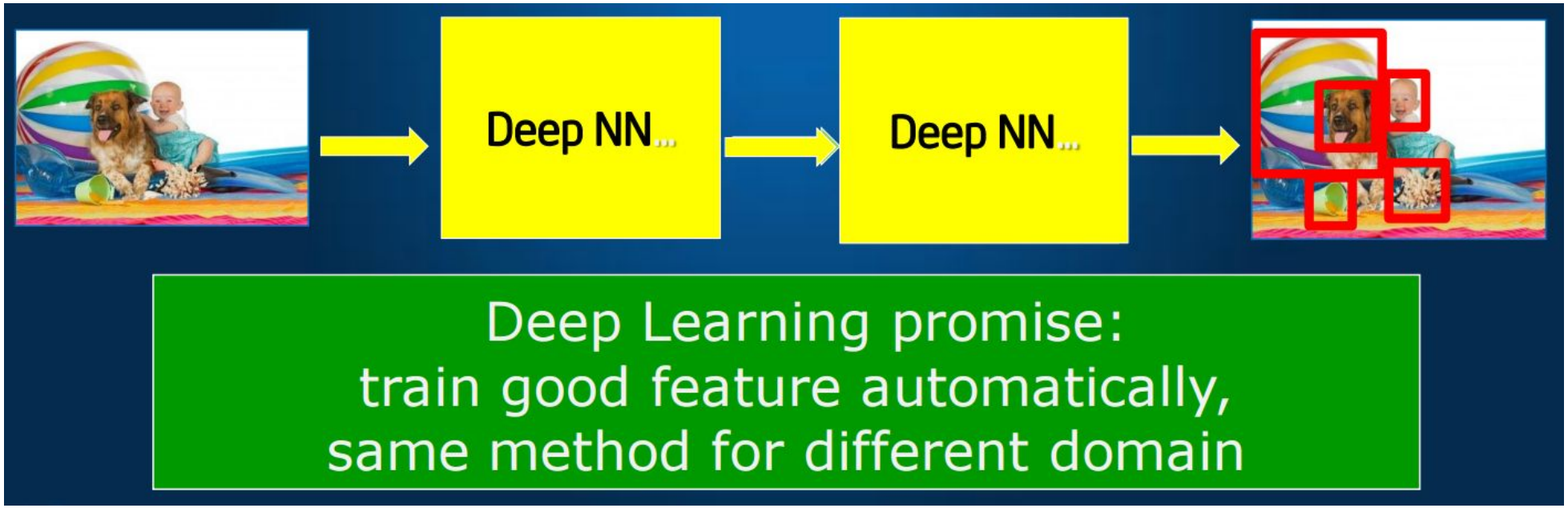
1. Select / develop features: SURF, HoG, SIFT, RIFT, …
2. Add on top of this Machine Learning for multi-class recognition and train classifier



Deep Learning –based Vision Pipeline

Deep Learning:

- Build features automatically based on training data
- Combine feature extraction and classification DL experts: define NN topology and train NN



Deep Learning Taxonomy

Supervised:

- Convolutional NN (LeCun)
- Recurrent Neural nets (Schmidhuber)

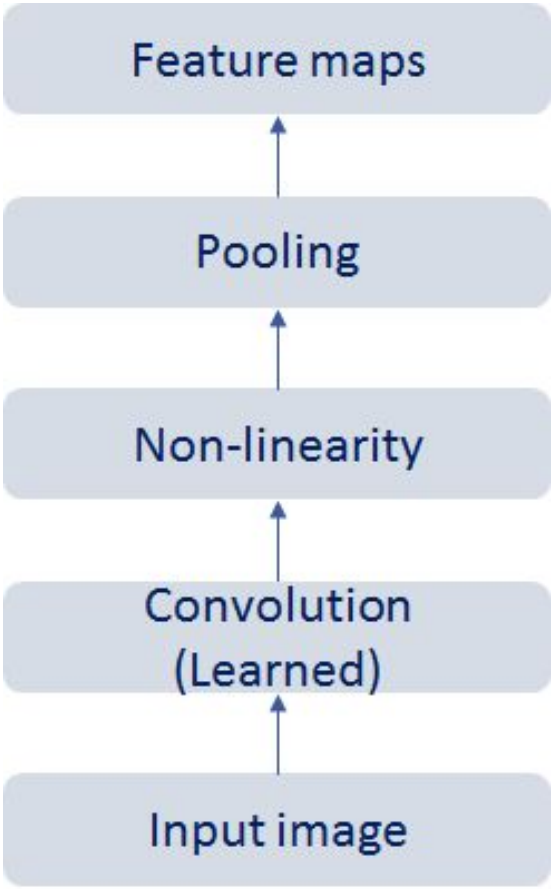
Unsupervised

- Deep Belief Nets / Stacked RBMs (Hinton)
- Stacked denoising autoencoders (Bengio)
- Sparse AutoEncoders (LeCun, A. Ng,)

Convolutional Neural Networks

Lenet

In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

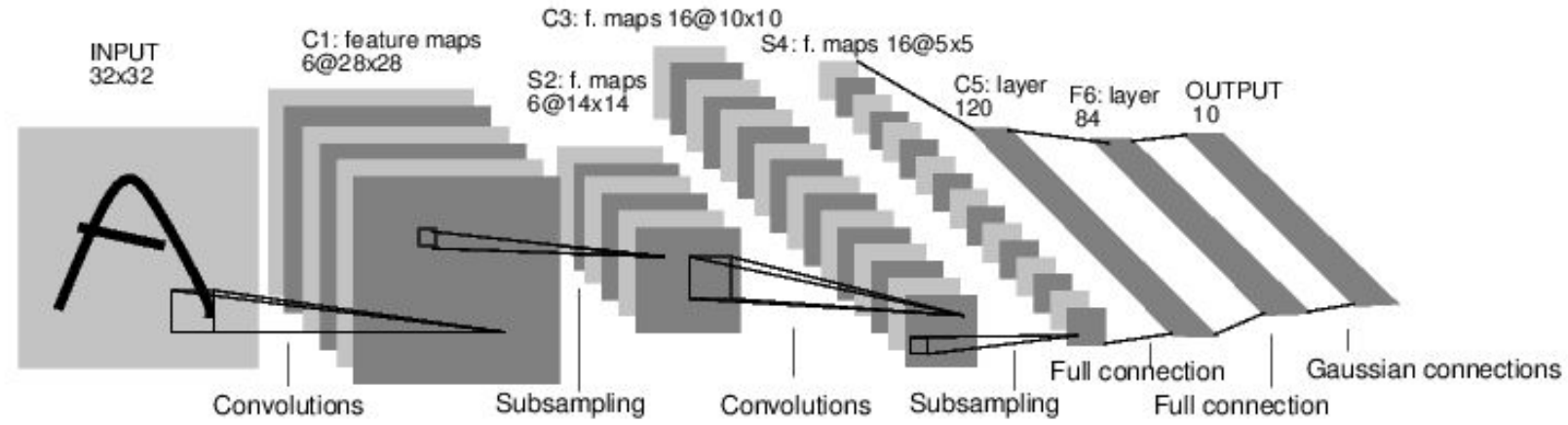


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86:2278–2324, 1998

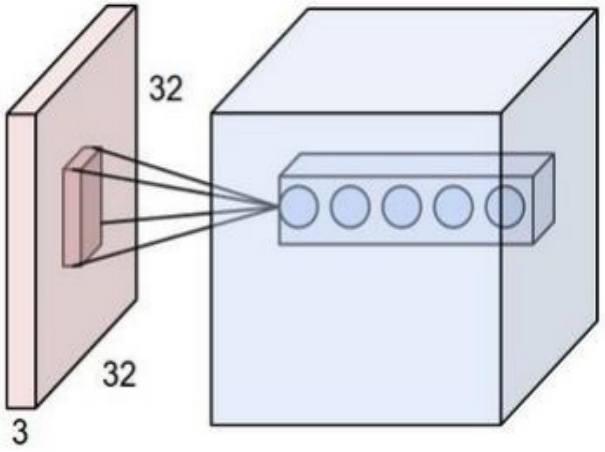
Lenet

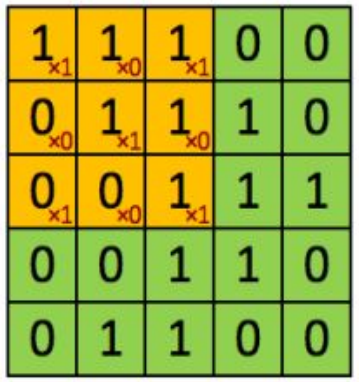
LeNet5 features can be summarized as:

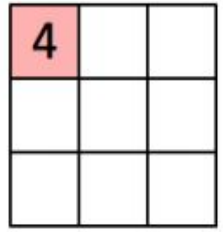
- convolutional neural network use sequence of 3 layers: convolution, pooling, non-linearity –> This may be the key feature of Deep Learning for images since this paper!
- use convolution to extract spatial features
- subsample using spatial average of maps
- non-linearity in the form of tanh or sigmoids
- multi-layer neural network (MLP) as final classifier
- sparse connection matrix between layers to avoid large computational cost



Convolution Layer







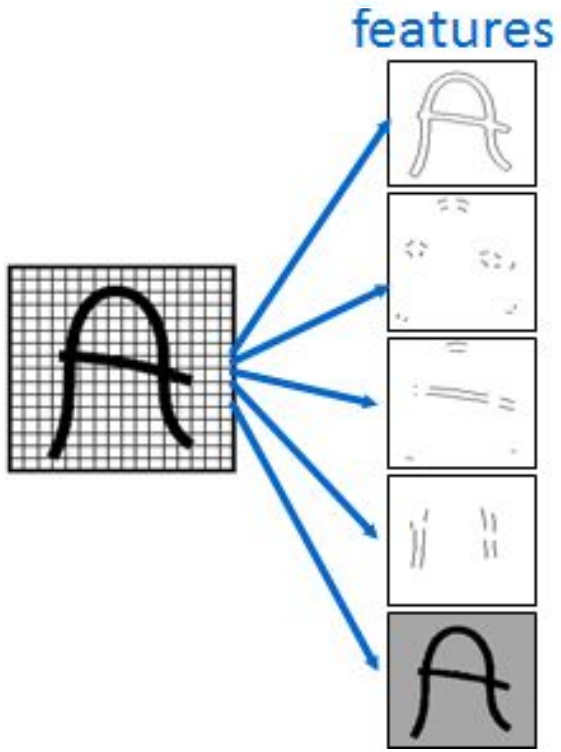
- Dependencies are local
- Translation invariance
- Few parameters (filter weights)
- Stride can be greater than 1 (faster, less memory)

Andrej Karpathy and Fei-Fei. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks

Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation http://www-labs.iro.umontreal.ca/~bengioy/DLbook

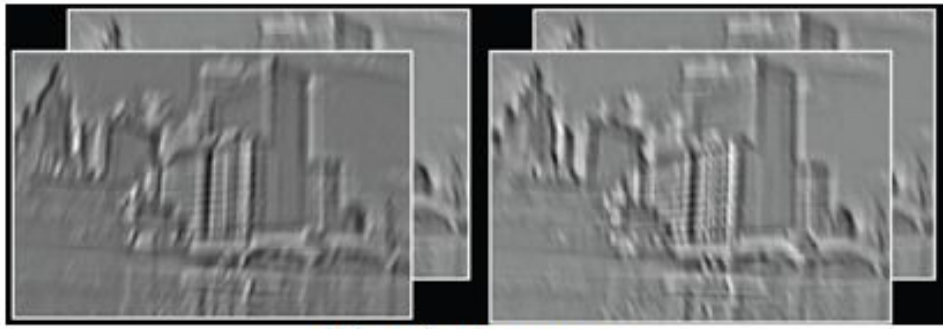
Convolution Layer

- Detect the same feature at different positions in the input image



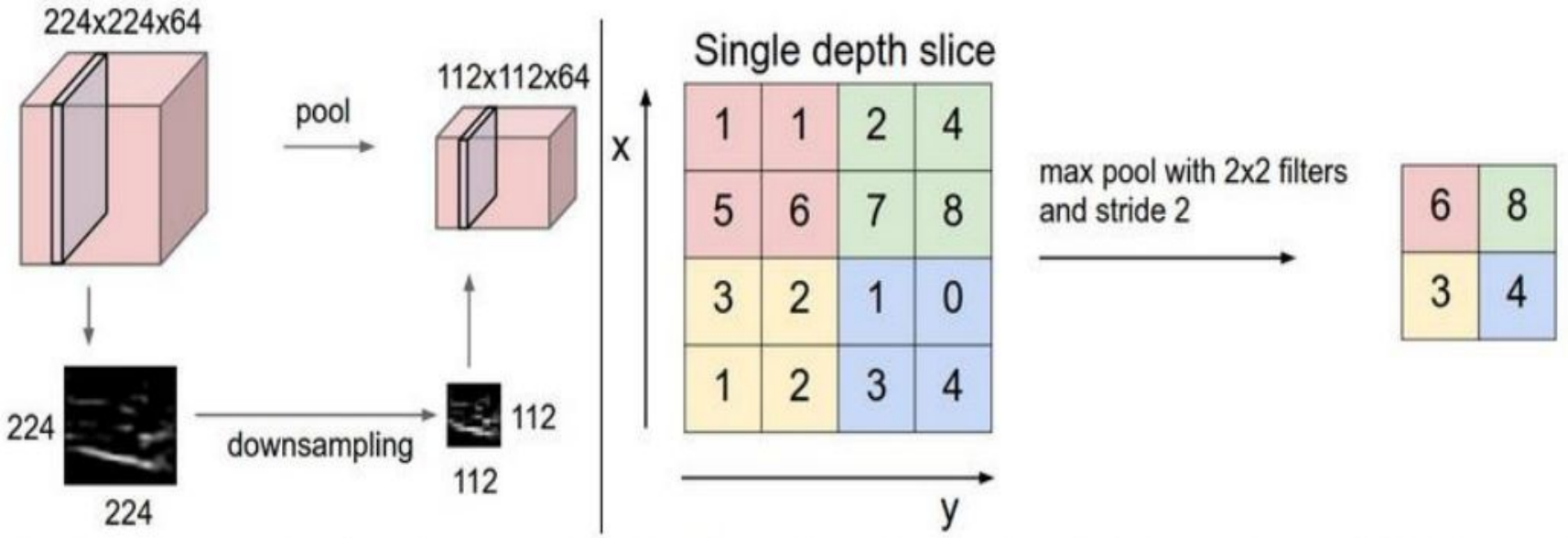
Filter
(kernel)

Input



Feature map

Pooling Layer

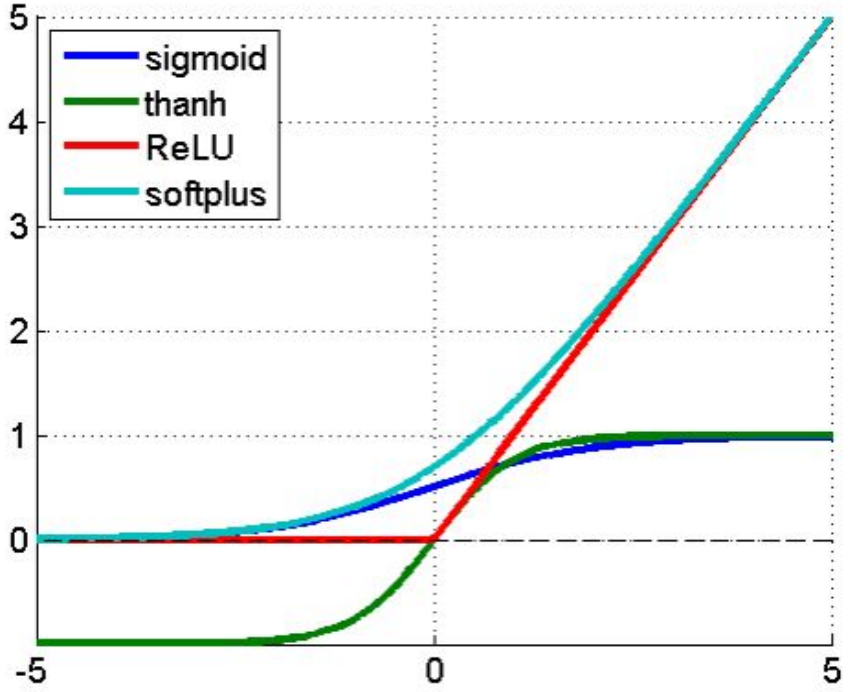


- Invariance to small transformations
- Larger receptive fields (see more of input)

Andrej Karpathy and Fei-Fei. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks

Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation http://www-labs.iro.umontreal.ca/~bengioy/DLbook

Non-Linearity



ReLU activation function

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

- Simplifies backpropagation
- Makes learning faster
- Avoids saturation issues (Preferred option)

Andrej Karpathy and Fei-Fei. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks

Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation http://www.iro.umontreal.ca/~bengioy/DLbook

ImageNet

[Deng et al. CVPR 2009]

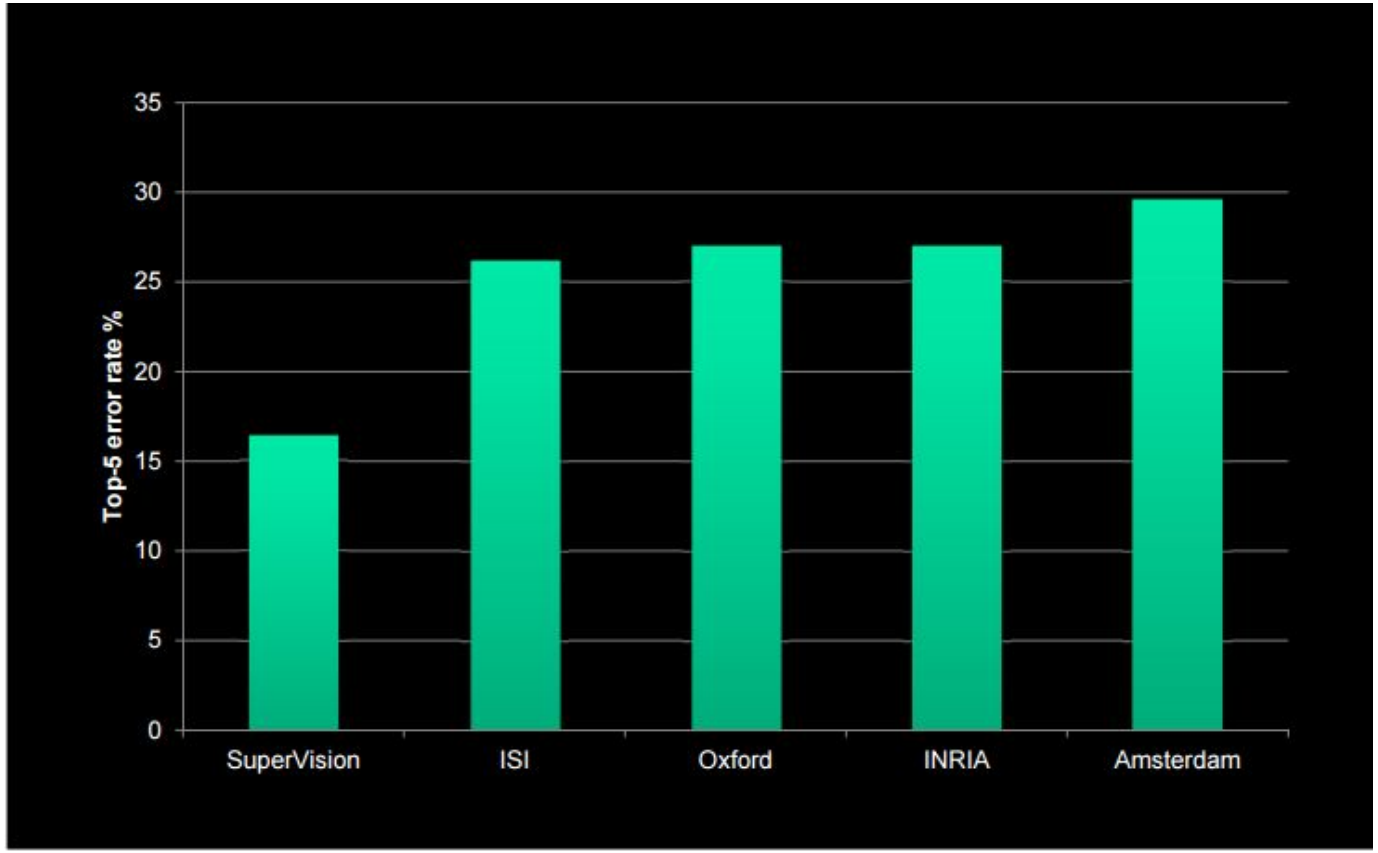
- ~14 million labeled images, 20k classes
- Images gathered from Internet Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

ImageNet Challenge 2012

Krizhevsky et al. -- **16.4% error** (top-5)

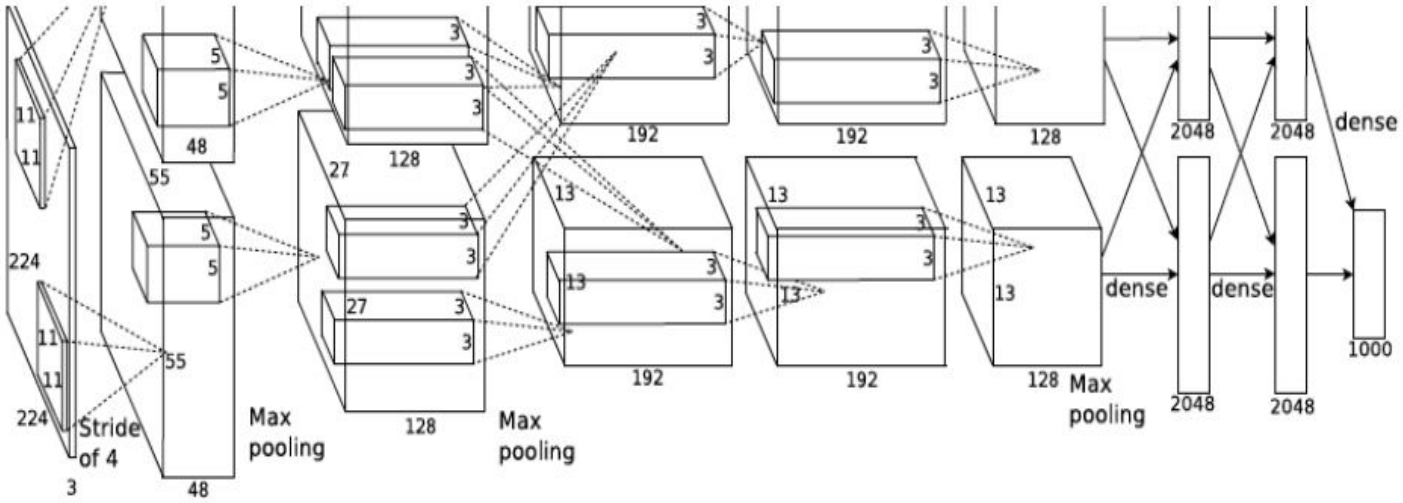
Next best (non-convnet) – **26.2% error**



AlexNet

Similar framework to LeCun'98 but:

- Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
- More data (10 6 vs. 10 3 images)
- GPU implementation (50x speedup over CPU)
- Trained on two GPUs for a week
- Better regularization for training (DropOut)



A. Kryzhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems 25 (NIPS 2012).

Caffe

What is Caffe?

Open framework, models, and worked examples
for deep learning

- 2 years old
- 1,000+ citations, 150+ contributors, 9,000+ stars
- 5,000+ forks, >1 pull request / day average
- focus has been vision, but branching out: sequences, reinforcement learning, speech + text

Prototype

Train

Deploy

What is Caffe?

Open framework, models, and worked examples
for deep learning

- Pure C++ / CUDA library for deep learning
- Command line, Python, MATLAB interfaces
- Fast, well-tested code
- Tools, reference models, demos, and recipes
- Seamless switch between CPU and GPU

Prototype

Train

Deploy

Caffe is a Community

project pulse

BVLC / caffe

Unwatch 1,205 | Unstar 8,498 | Fork 4,821

January 19, 2016 – February 19, 2016

Period: **1 month**

Overview

45 Active Pull Requests

90 Active Issues

22 Merged Pull Requests

23 Proposed Pull Requests

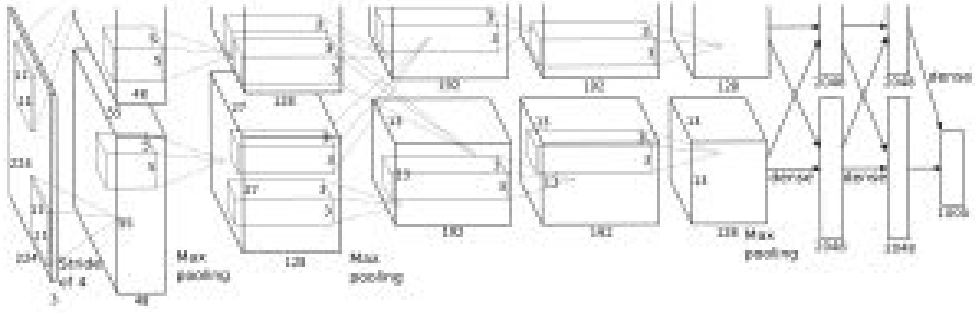
52 Closed Issues

38 New Issues

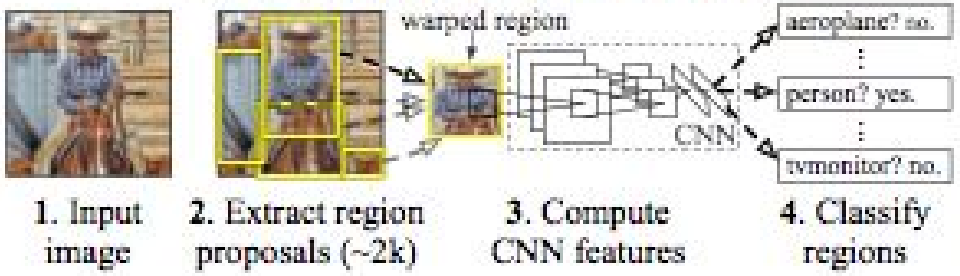
Excluding merges, **20 authors** have pushed **19 commits** to master and **53 commits** to all branches. On master, **44 files** have changed and there have been **2,268 additions** and **162 deletions**.

Reference Models

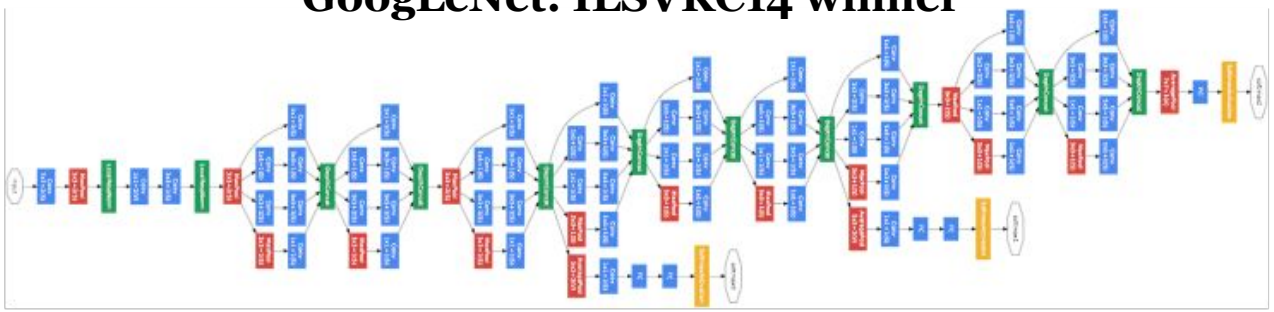
AlexNet: ImageNet Classification



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 our Model Zoo.

Open Model Collection

The Caffe Model Zoo open collection of deep models to share innovation

- MSRA ResNet ILSVRC15 winner in the zoo
- VGG ILSVRC14 + Devil models in the zoo
- MIT Places scene recognition model in the zoo
- Network-in-Network / CCCP model in the zoo

helps disseminate and reproduce research

bundled tools for loading and publishing models

Share Your Models! with your citation + license of course

Brewing by the Numbers...

Speed with Krizhevsky's 2012 model:

- 2 ms/image on K40 GPU
- <1 ms inference with Caffe + cuDNN v4 on Titan X
- 72 million images/day with batched IO
- 8-core CPU: ~20 ms/image Intel optimization in progress

9k lines of C++ code (20k with tests)

Embedded Caffe

Caffe runs on embedded CUDA hardware and mobile devices

- same model weights,
  same framework interface
- out-of-the-box on
  CUDA platforms
- in-progress OpenCL port
  thanks Fabian Tschopp!
  + AMD, Intel, and the community
- community Android port
  thanks sh1r0!

CUDA Jetson TX1, TK1

OpenCL branch

Android lib, demo

Caffeinated Companies



… startups, big companies, more ...

Caffe at Facebook

- in production for vision at scale: uploaded photos run through Caffe
- Automatic Alt Text for the blind
- On This Day for surfacing memories
- objectionable content detection
- contributing back to the community: inference tuning, tools, code review include fb-caffe-exts thanks Andrew!

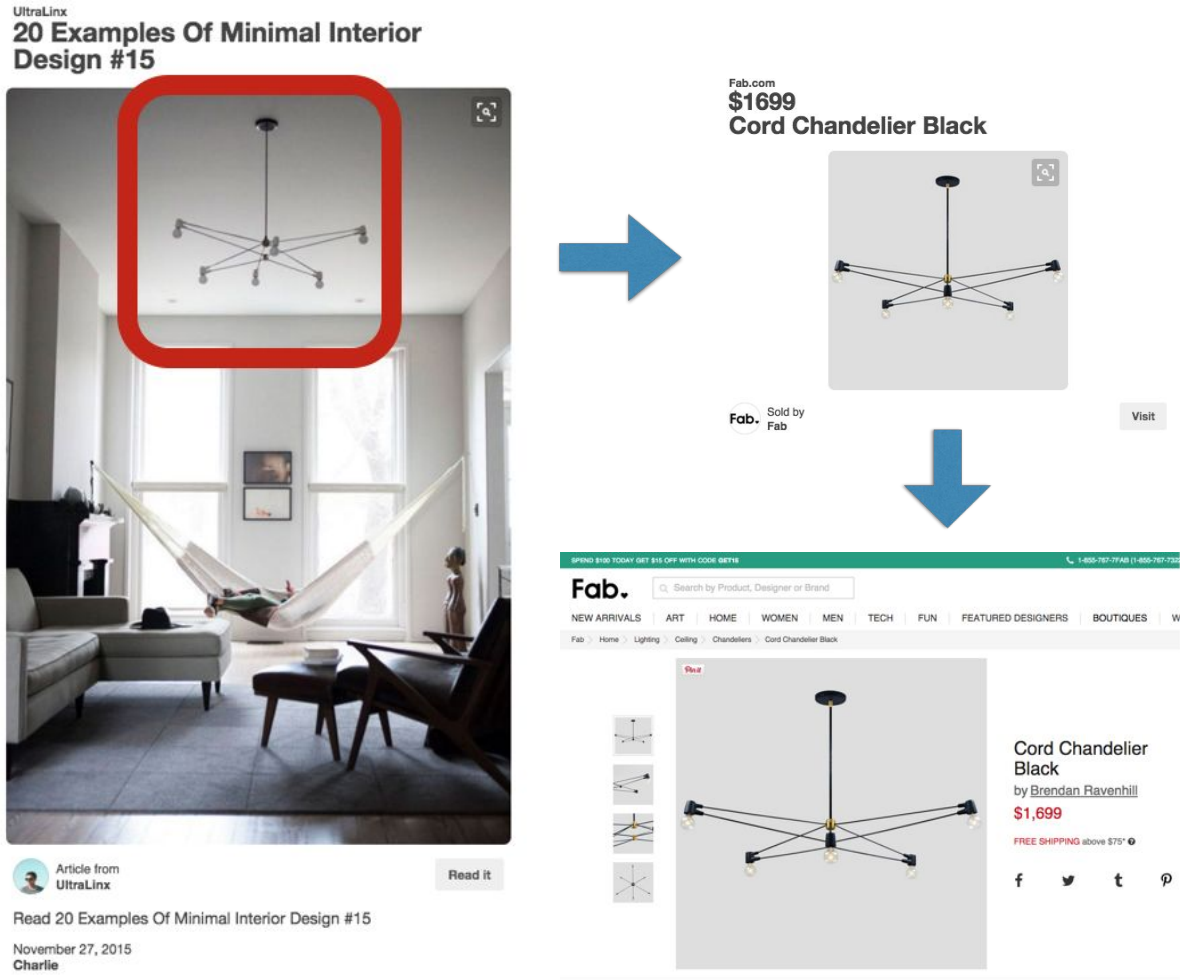
On This Day
highlight content

Automatic Alt Text
recognize photo content for accessibility

[example credit Facebook]

Caffe at Pinterest

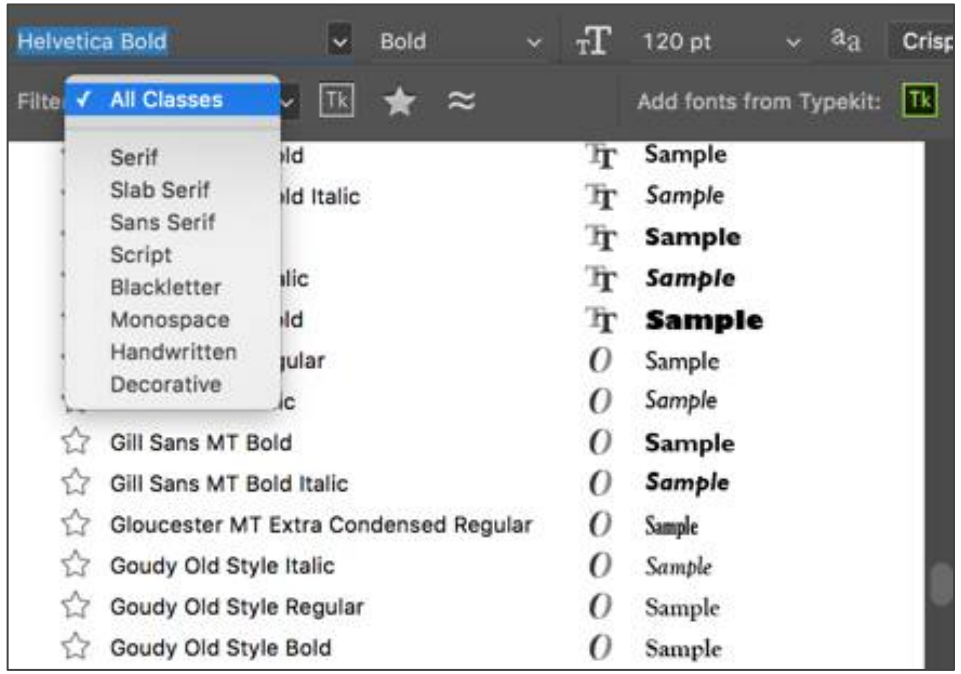
- in production for vision at scale: uploaded photos run through Caffe
- deep learning for visual search: retrieval over billions of images in <250 ms
- ~4 million requests/day
- built on an open platform of Caffe, FLANN, Thrift, ...



[example credit Andrew Zhai, Pinterest]

Caffe at Adobe

- training networks for research in vision and graphics
- custom inference in products, including Photoshop



Photoshop Type Similarity
catalogue typefaces automatically

Caffe at Yahoo! Japan

- personalize news and content, and de-duplicate suggestions
- curate news and restaurant photos for recommendation
- arrange user photo albums

News Image Recommendation
select and crop images for news

Caffe Tutorial

Tour

- Nets, Layers, and Blobs: the anatomy of a Caffe model.
- Forward / Backward: the essential computations of layered compositional models.
- Loss: the task to be learned is defined by the loss.
- Solver: the solver coordinates model optimization.
- Layer Catalogue: the layer is the fundamental unit of modeling and computation – Caffe's catalogue includes layers for state-of-the-art models.
- Interfaces: command line, Python, and MATLAB Caffe.
- Data: how to caffeinate data for model input.

For a closer look at a few details:

- Caffeinated Convolution: how Caffe computes convolutions.

http:/caffe.berkeleyvision.org/tutorial/

Deep Learning, as it is executed...

What should a framework handle?

Compositional Models

Decompose the problem and code!

End-to-End Learning

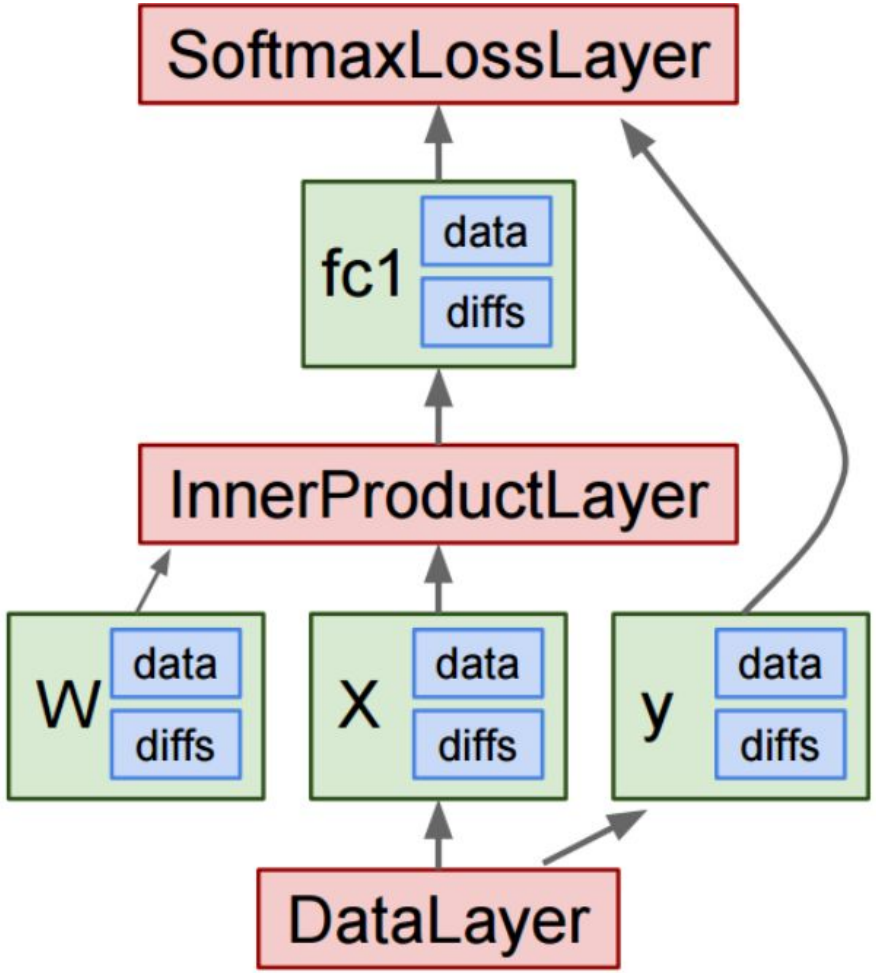
Configure and solve!

Many Architectures and Tasks

Define, experiment, and extend!

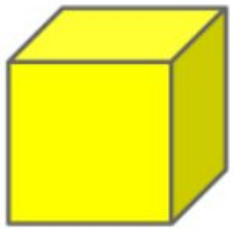
Main Classes

- **Blob:** Stores data and derivatives
- **Layer**: Transforms bottom blobs to top blobs
- **Net:** Many layers; computes gradients via Forward / Backward
- **Solver:** Uses gradients to update weights

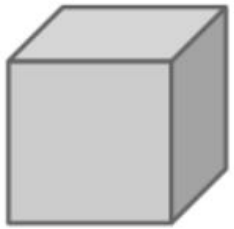


Slide credit: Stanford Vision CS231

Blobs



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

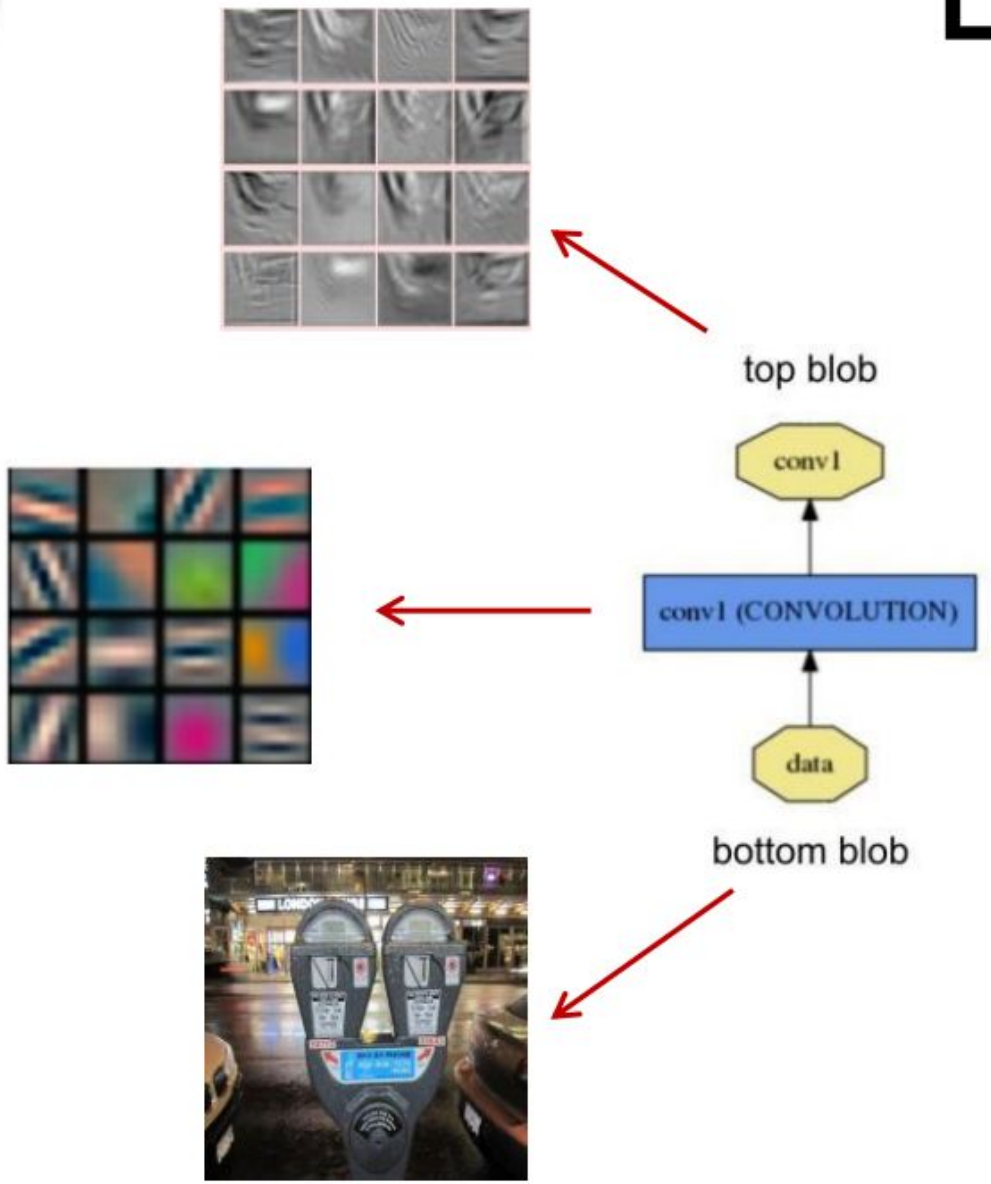
Parameter: Convolution Bias
96 x 1 x 1 x 1 for CaffeNet conv1

N-D arrays for storing and communicating data

- Hold data, derivatives and parameters
- Lazily allocate memory
- Shuttle between CPU and GPU

Slide credit: Evan Shelhamer, Jeff Donahue, Jon Long, Yangqing Jia, and Ross Girshick

Layers

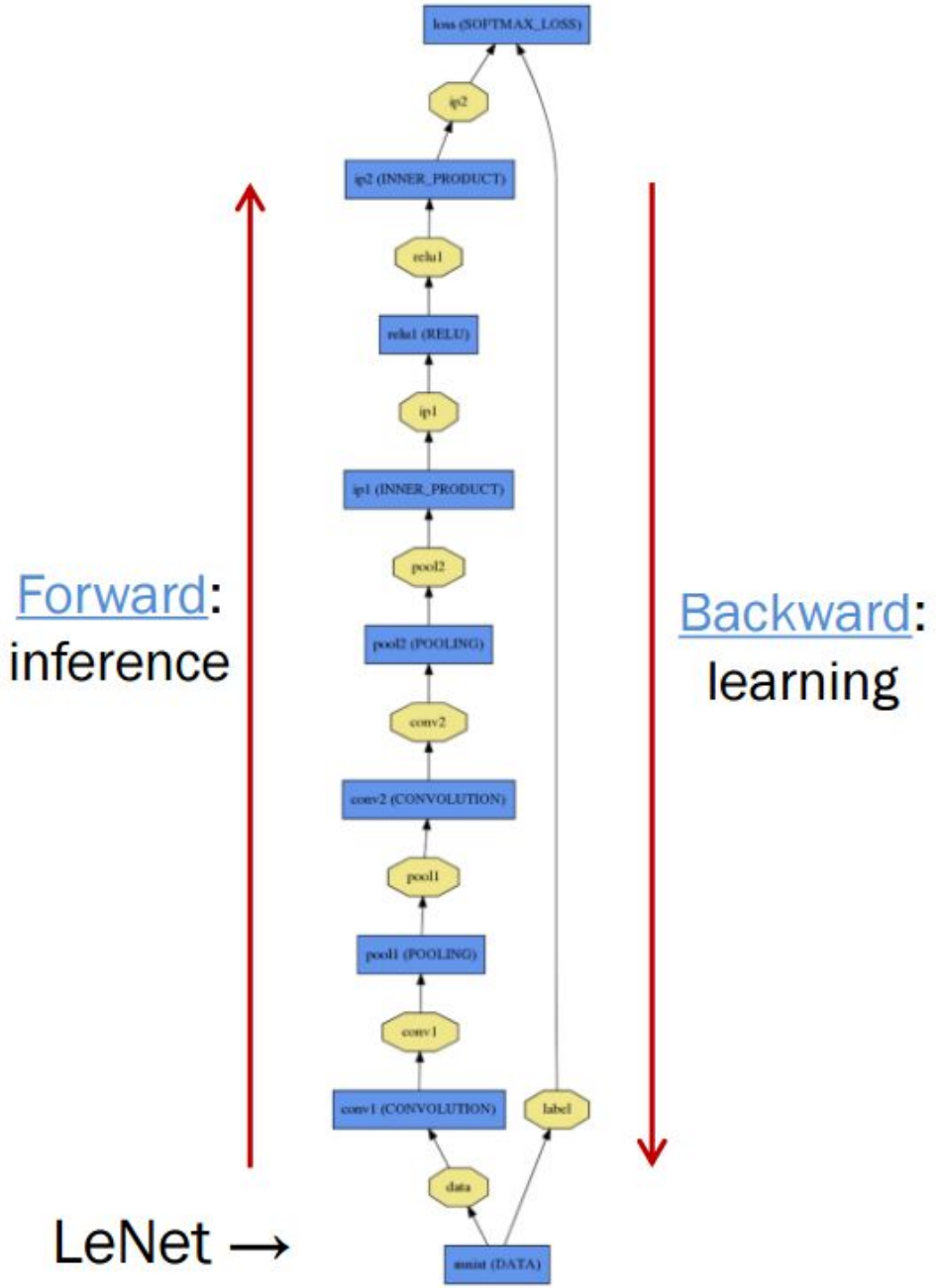


Caffe's fundamental unit of computation

Implemented as layers:

- Data access
- Convolution
- Pooling
- Activation Functions
- Loss Functions
- Dropout
- etc.

Net



- A DAG of layers and the blobs that connect them
- Caffe creates and checks the net from a definition file (more later)
- Exposes Forward / Backward methods

Solver

- Calls Forward / Backward and updates net parameters
- Periodically evaluates model on the test network(s)
- Snapshots model and solver state

Solvers available:

- SGD
- AdaDelta
- AdaGrad
- Adam
- Nesterov
- RMSprop

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
```

Protocol Buffers

- Like strongly typed, binary JSON!
- Auto-generated code
- Developed by Google
- Net / Layer / Solver / parameters are **messages** defined in .prototxt files
- Available **message types** defined in ./src/caffe/proto/caffe.proto
- Models and solvers are schema, not code

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

  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in all spatial dimensions, or once per spatial dimension.
  repeated uint32 pad = 3; // The padding size; defaults to 0
  repeated uint32 kernel_size = 4; // The kernel size
  repeated uint32 stride = 6; // The stride; defaults to 1

  // For 2D convolution only, the *_h and *_w versions may also be used to
  // specify both spatial dimensions.
  optional uint32 pad_h = 9 [default = 0]; // The padding height (2D only)
  optional uint32 pad_w = 10 [default = 0]; // The padding width (2D only)
  optional uint32 kernel_h = 11; // The kernel height (2D only)
  optional uint32 kernel_w = 12; // The kernel width (2D only)
  optional uint32 stride_h = 13; // The stride height (2D only)
  optional uint32 stride_w = 14; // The stride width (2D only)

  optional uint32 group = 5 [default = 1]; // The group size for group conv

  optional FillerParameter weight_filler = 7; // The filler for the weight
  optional FillerParameter bias_filler = 8; // The filler for the bias
  enum Engine {
    DEFAULT = 0;
    CAFFE = 1;
    CUDNN = 2;
  }
  optional Engine engine = 15 [default = DEFAULT];
}
```

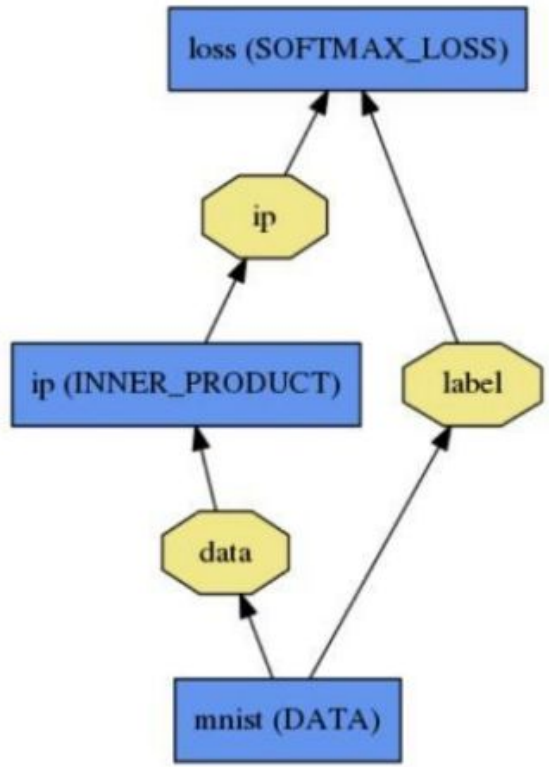
Prototxt: Define Net

Blobs

Number of output classes

Layer type

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```



Prototxt: Layer Detail

Learning rates (weight + bias)
Set these to 0 to freeze a layer

Convolution-specific parameters

Parameter Initialization

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Example from ./models/bvlc_reference_caffenet/train_val.prototxt

Prototxt: Define Solver

Test on validation set →

Learning rate profile

Net prototxt →

Snapshots during training

```
test_iter: 100
test_interval: 500
base_lr: 0.01
display: 100
max_iter: 10000
lr_policy: "inv"
gamma: 0.0001
power: 0.75
momentum: 0.9
weight_decay: 0.0005
solver_mode: GPU
net: "examples/mnist/lenet_train_test.prototxt"
# The snapshot interval in iterations.
snapshot: 5000
# File path prefix for snapshotting model weights and solver state.
# Note: this is relative to the invocation of the `caffe` utility, not the
# solver definition file.
snapshot_prefix: "/path/to/model"
# Snapshot the diff along with the weights. This can help debugging training
# but takes more storage.
snapshot_diff: false
# A final snapshot is saved at the end of training unless
# this flag is set to false. The default is true.
snapshot_after_train: true
```

Setting Up Data

- Prefetching
- Multiple Inputs
- Data augmentation on-the-fly (random crops, flips) – see TransformationParameter proto

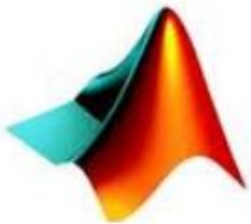
Choice of Data Layers:

- Image files
- LMDB
- HDF5

Interfaces

python

```
out = net.forward()
```



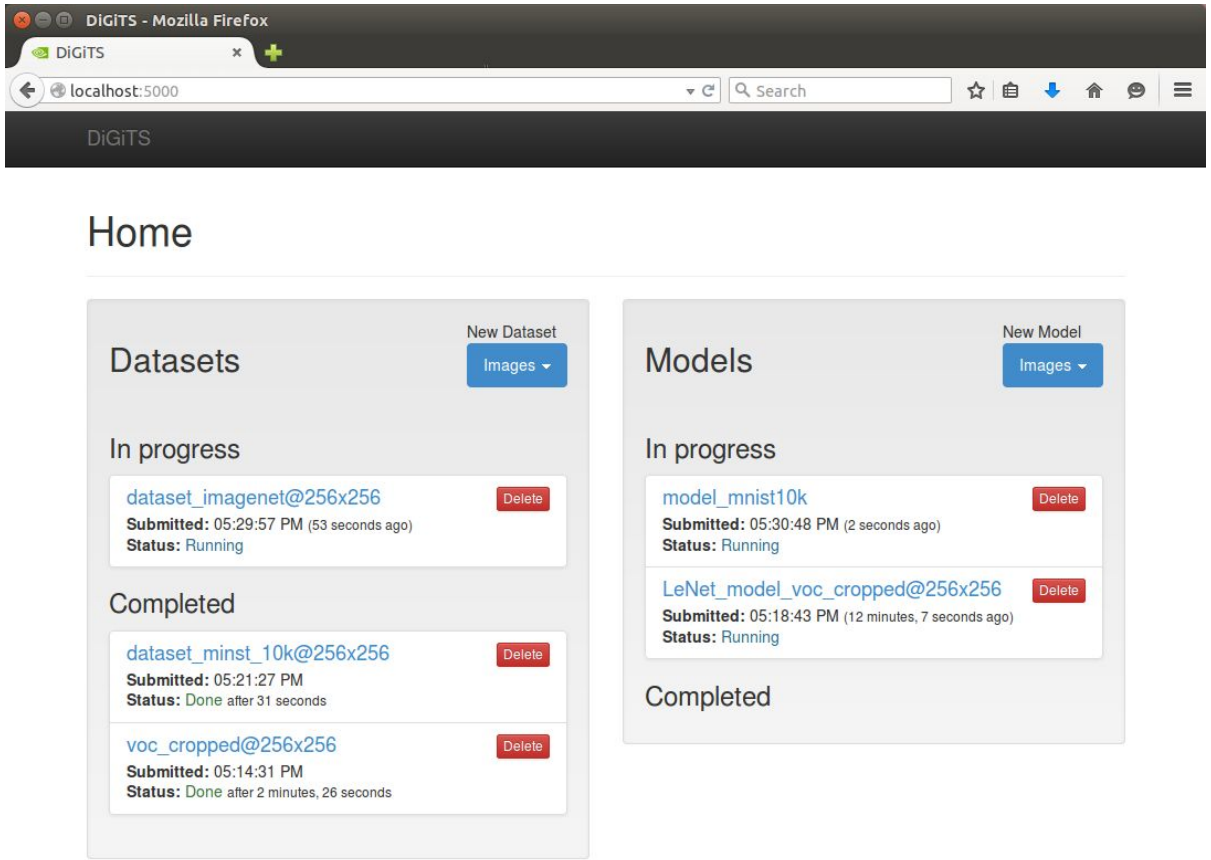
MATLAB

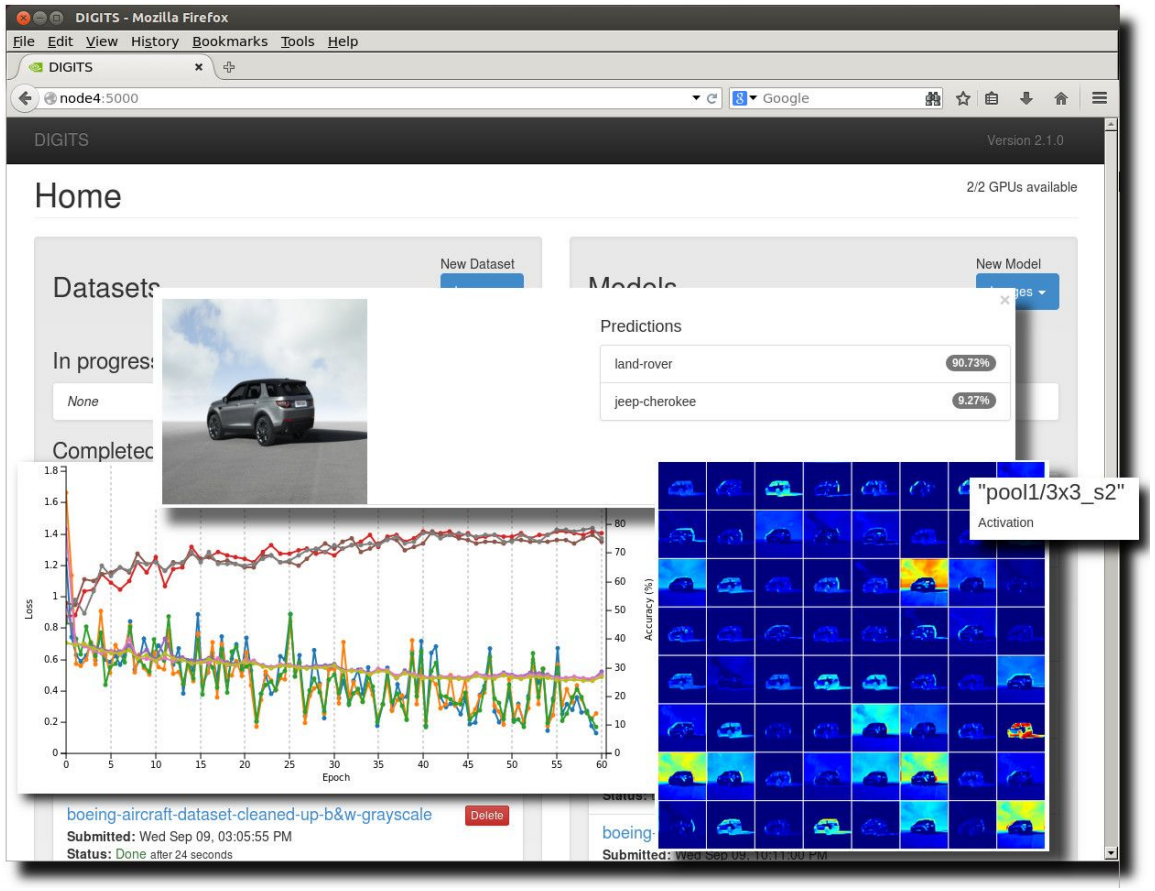
```
scores = net.forward(input_data);
```

- Blob data and diffs exposed as Numpy arrays
- ./python/caffe/_caffe.cpp: Exports Blob, Layer, Net & Solver classes
- ./python/caffe/pycaffe.py: Adds extra methods to Net class
- Jupyter notebooks: ./examples

- Similar to PyCaffe in usage
- Demo: ./matlab/demo/classification_demo.m
- Images are in BGR channels

Nvidia DIGITS





https://developer.nvidia.com/digits

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/*`

Learning LeNet

back to the future of visual recognition

see script

see notebook

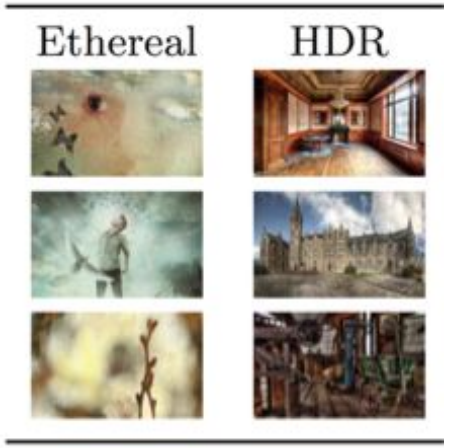
Digits Recognizer on kaggle

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

[DeCAF] [Zeiler-Fergus] [OverFeat]

Lots of Data

image by Andrej Karpathy



Style Recognition

© kaggle.com

Dogs vs. Cats
top 10 in 10 minutes

Your Task

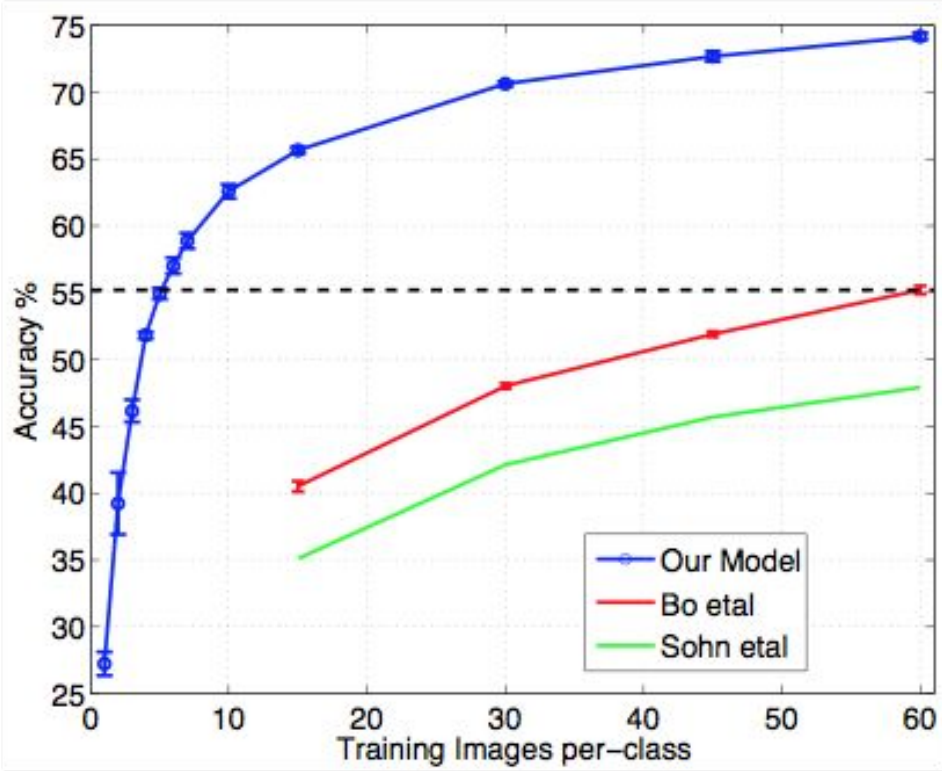
When to Fine-tune?

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

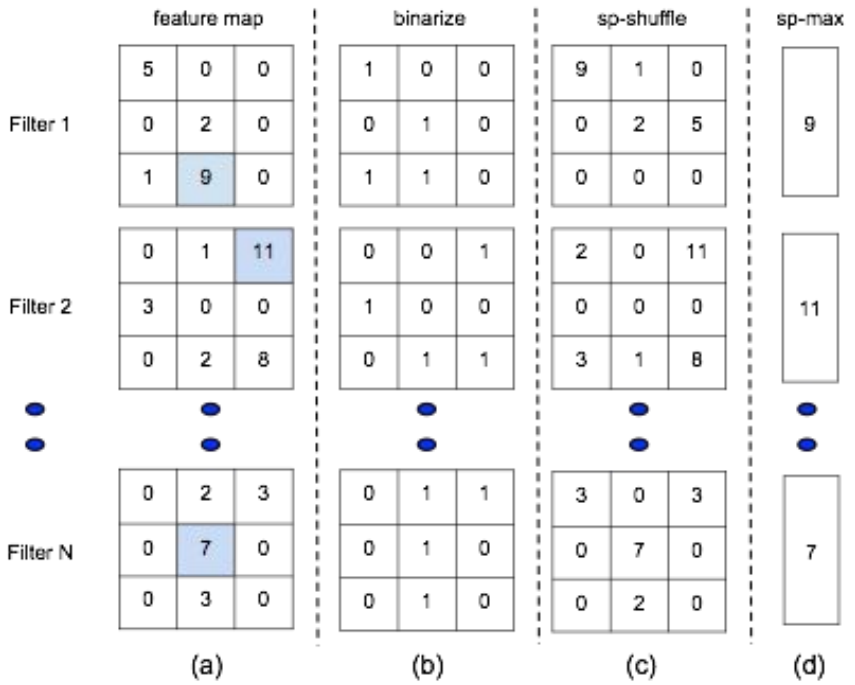
Reduce the learning rate

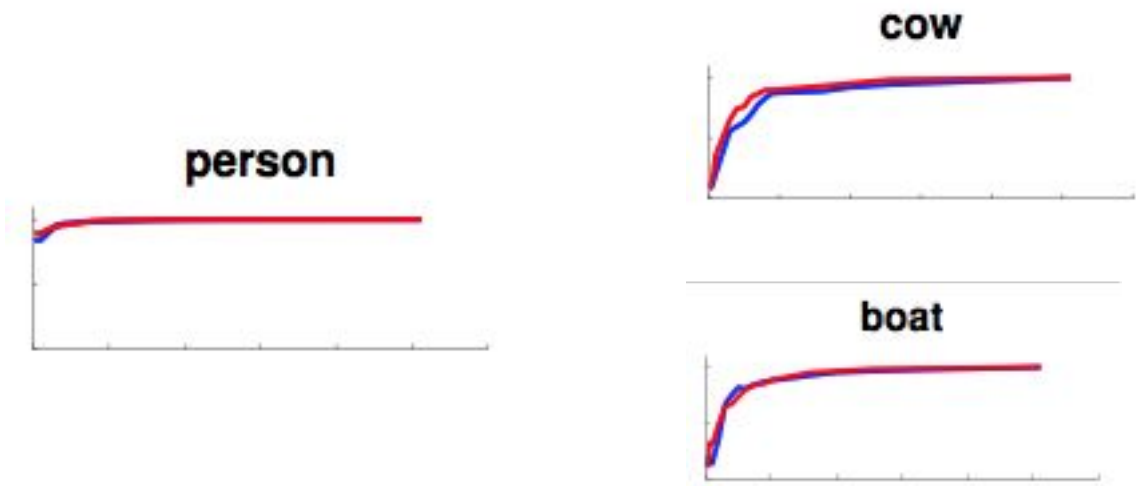
- Drop the solver learning rate by 10x, 100x
- Preserve the initialization from pre-training and avoid divergence

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

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

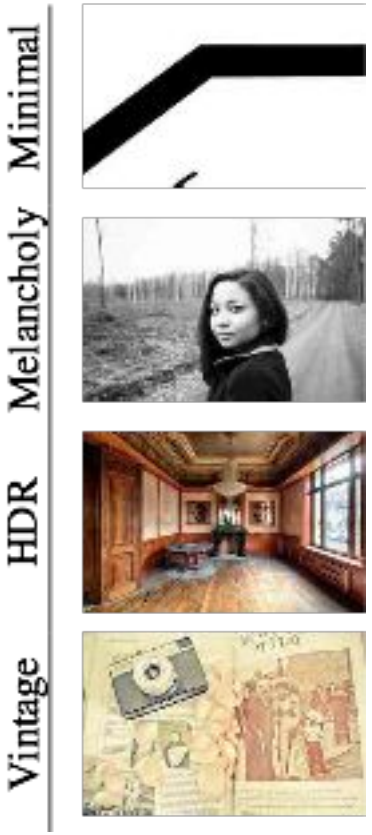
Last Layer:
A different classifier

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 script

see notebook

Cats on kaggle

Thanks to the Caffe Crew

...plus the cold-brew

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Jonathan Long,
Sergey Karayev, Ross Girshick, Sergio Guadarrama, Ronghang Hu, Trevor Darrell

and our open source contributors!

Kaggle active competitions

[Draper Satellite Image Chronology]

[State Farm Distracted Driver Detection]

[Painter by Numbers]