Introduction to Neural Networks and Brief Tutorial with Caffe

10th Set of Notes

Assembled by Qilin Zhang,
based on [NNDL], [DLT], [Caffe], etc.

Notes for the CS 559 Machine Learning Class

Outline

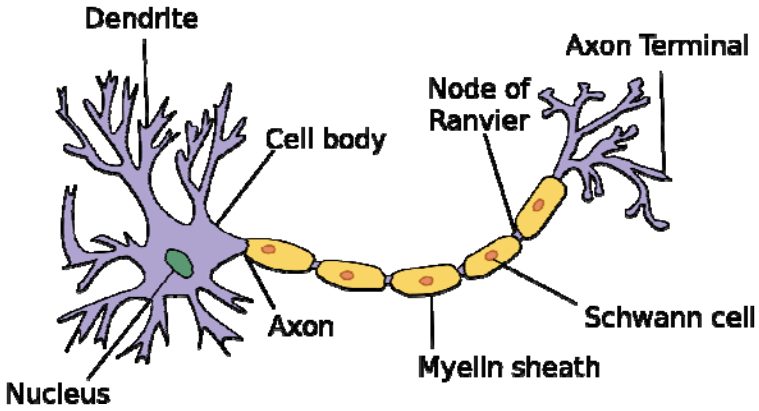
- Neural Networks
  - Neurons
  - Cost Function & Optimization
- Convolutional Neural Networks (CNNs)
- Deep Learning libraries
  - Caffe

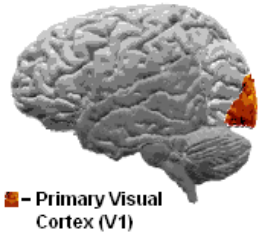
Neural Networks and Deep Learning

- Neural Networks
  - Biologically-inspired machine learning paradigm
  - Decades old
- Deep Learning
  - A recent variant of neural networks based machine learning
  - Big data and big model

Natural Neurons

- Human recognition of digits
  - visual cortices
  - neuron interaction



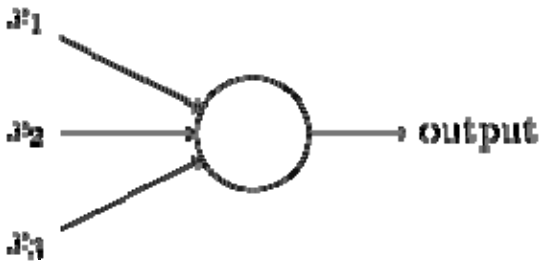


Recognizing Handwritten Digits

- How to describe a digit to a computer
  - "a 9 has a loop at the top, and a vertical stroke in the bottom right“
  - Algorithmically difficult to describe various 9’s

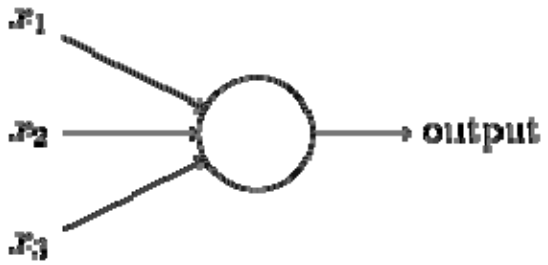
Perceptrons

- Perceptrons
  - 1950s ~ 1960s, Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts
- Standard model of artificial neurons



Binary Perceptrons

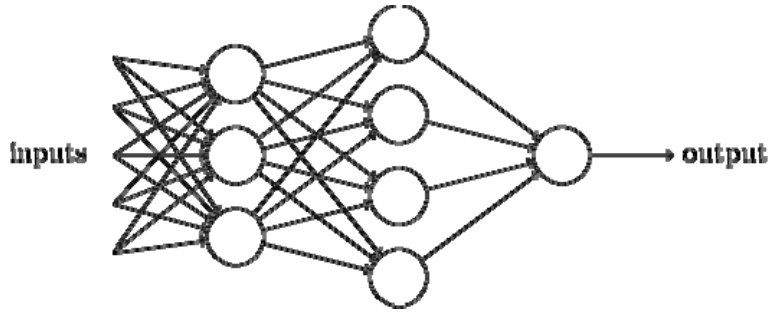
- Inputs
  - Multiple binary inputs
- Parameters
  - Thresholds & weights
- Outputs
  - Thresholded weighted linear combination



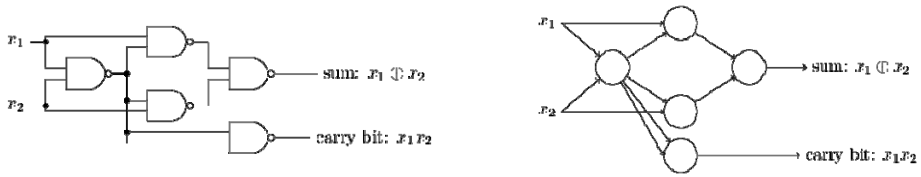
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Layered Perceptrons

- Layered, complex model
  - 1st layer, 2nd layer of percptrons
- Perceptron rule
  - Weights, thresholds
- Similarity to logical functions (NAND)

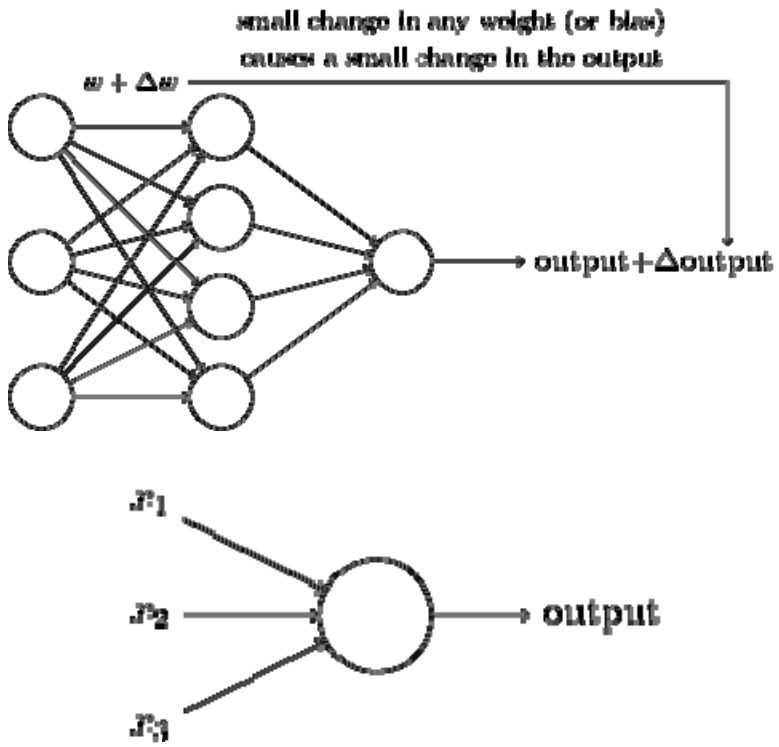


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



Sigmoid Neurons

- Sigmoid neurons
  - Stability
    - Small perturbation, small output change
  - Continuous inputs
  - Continuous outputs
  - Soft thresholds

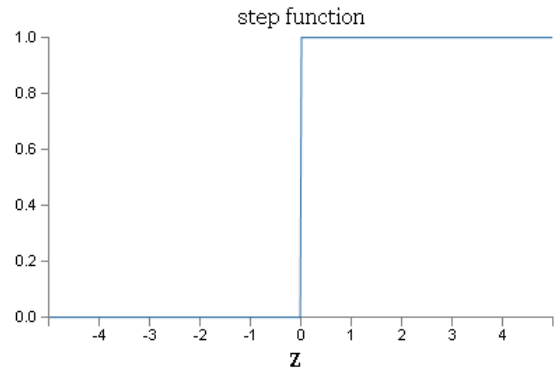


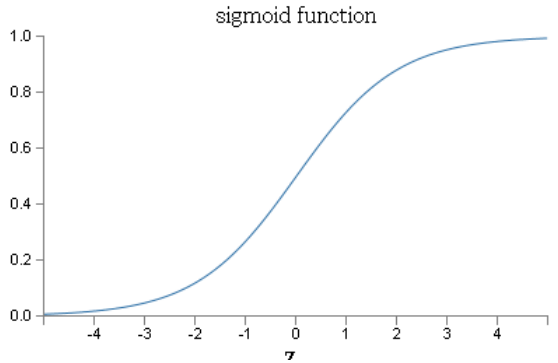
Output Functions

- Sigmoid neurons
- Output $\sigma(w \cdot x + b), \quad \sigma(z) \equiv \frac{1}{1+e^{-x}}$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

- Sigmoid vs conventional thresholds

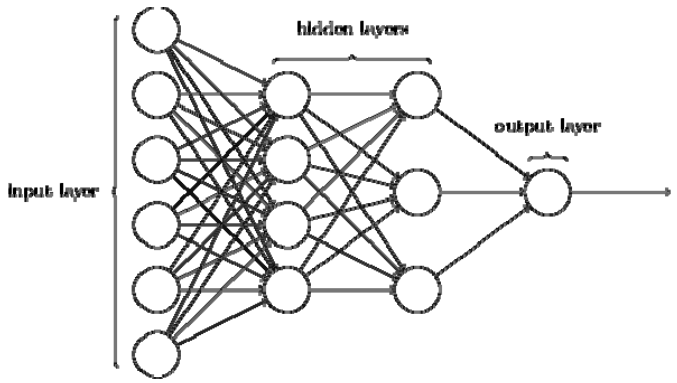




Smoothness & Differentiability

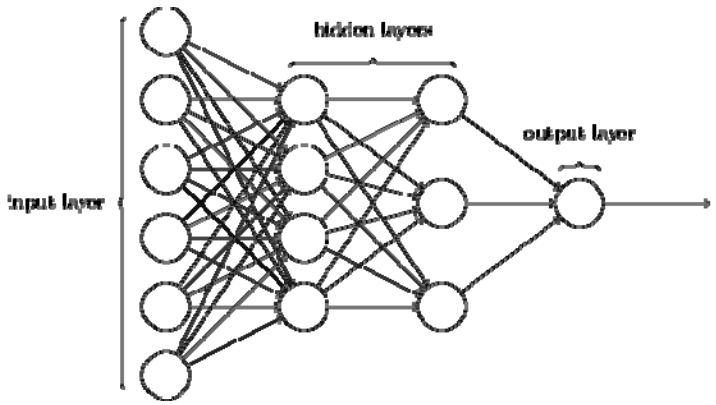
- Perturbations and Derivatives
  - Continuous function
  - Differentiable
- Layers
  - Input layers, output layers, hidden layers

$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b,$$



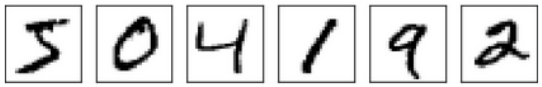
Layer Structure Design

- Design of hidden layer
  - Heuristic rules
  - Number of hidden layers vs. computational resources
  - Feedforward network
    - No loops involved

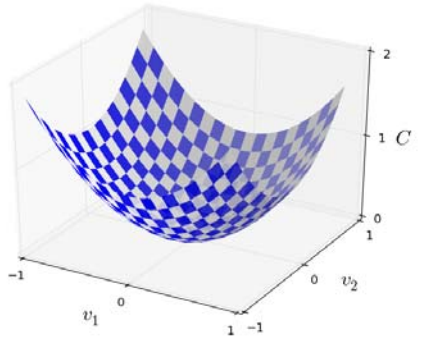


Cost Function & Optimization

- Learning with gradient descent
  - Cost function
  - Quadratic loss/Euclidean loss
  - Non-negative, smooth, differentiable



$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$



Cost Function & Optimization

- Gradient Descent

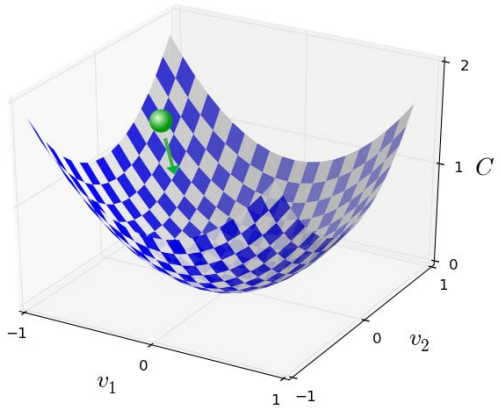
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$$

- Gradient vector

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T.$$

$$\Delta C \approx \nabla C \cdot \Delta v.$$

$$v \rightarrow v' = v - \eta \nabla C.$$



Cost Function & Optimization

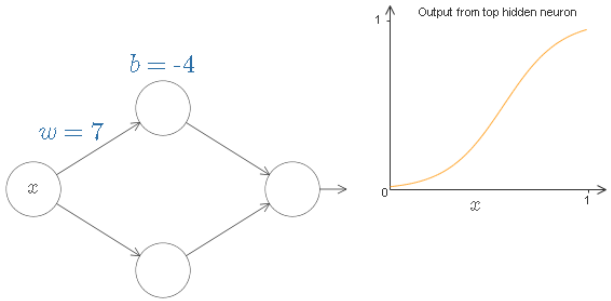
- Extension to multiple dimension
  - m variables $v_1, \ldots, v_m$
  - Small change in variable $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$
  - Small change in cost $\Delta C \approx \nabla C \cdot \Delta v,$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$$

$$\Delta v = -\eta \nabla C \qquad v \to v' = v - \eta \nabla C.$$

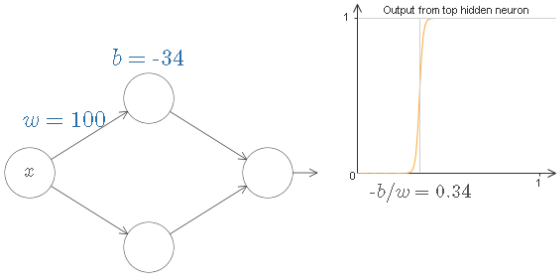
Arbitrary Function Generator

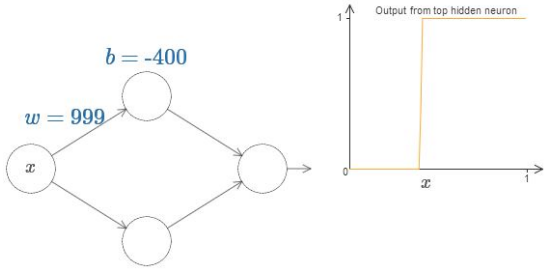
- Function synthesizer
  - Any function can be approximated reasonably with enough nodes and layers
- Example
  - One input one output
  - top neuron parameters

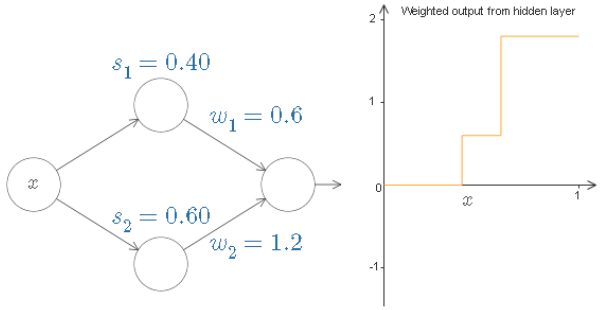


Arbitrary Function Generator

- Various parameters

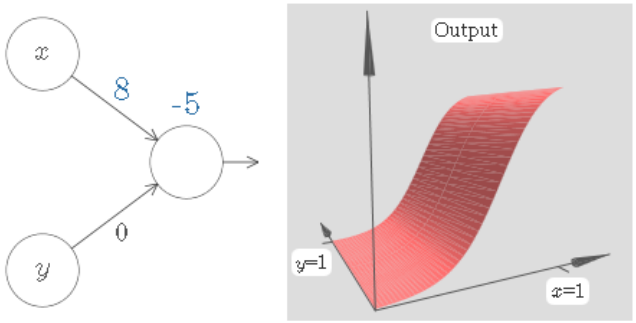


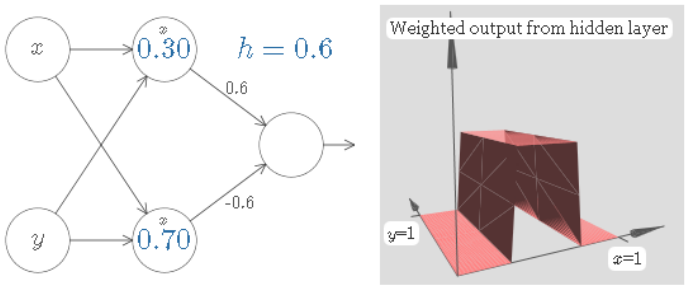




Arbitrary Function Generator

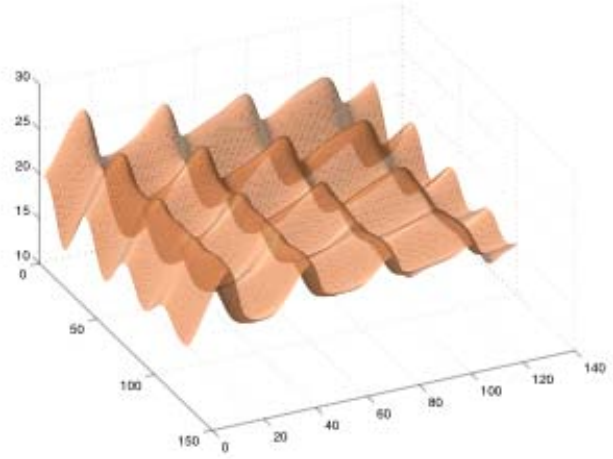
- Multiple input variables





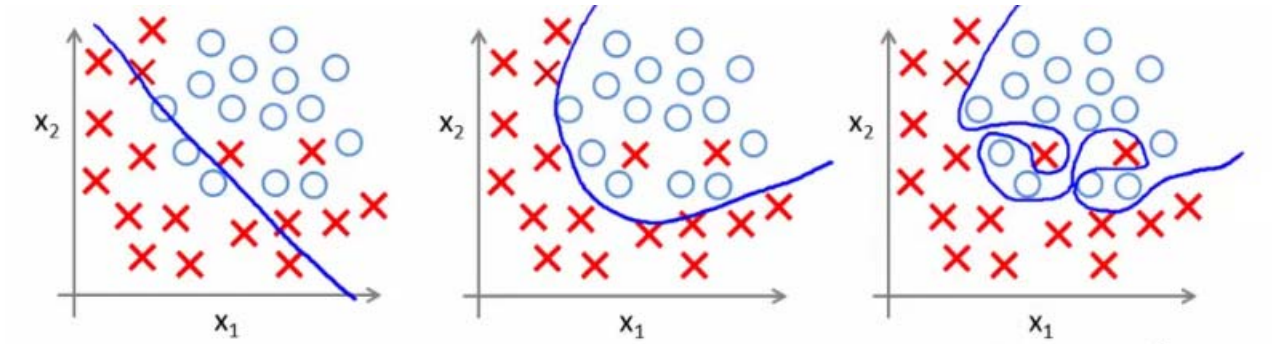
Practical Neural Nets

- Potential problems with neural networks
  - Non-convex cost
- Model complexity vs. computational limits
  - overfitting
  - Big data



Practical Neural Nets

- Big data
  - Huge training data avoids overfitting



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

UNDERFITTING
(high bias)

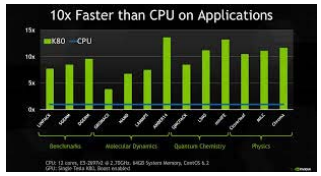
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots$

OVERFITTING
(high variance)

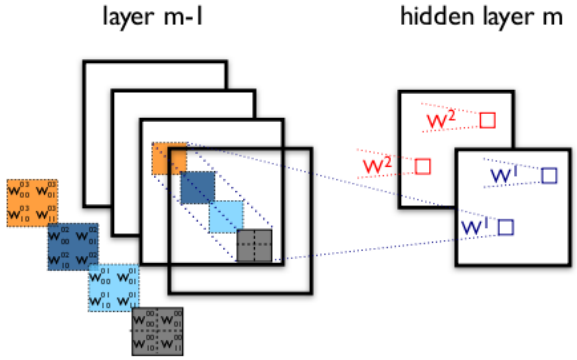
Practical Neural Nets

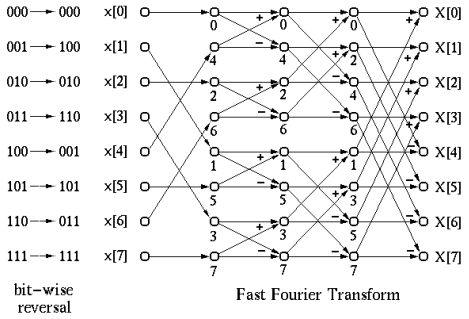
- Computational efficiency on big data
  - Hardware: GPGPU
  - Algorithm: stochastic gradient descent for limited RAM
  - Network design: convolutional layers instead of fully connected layers



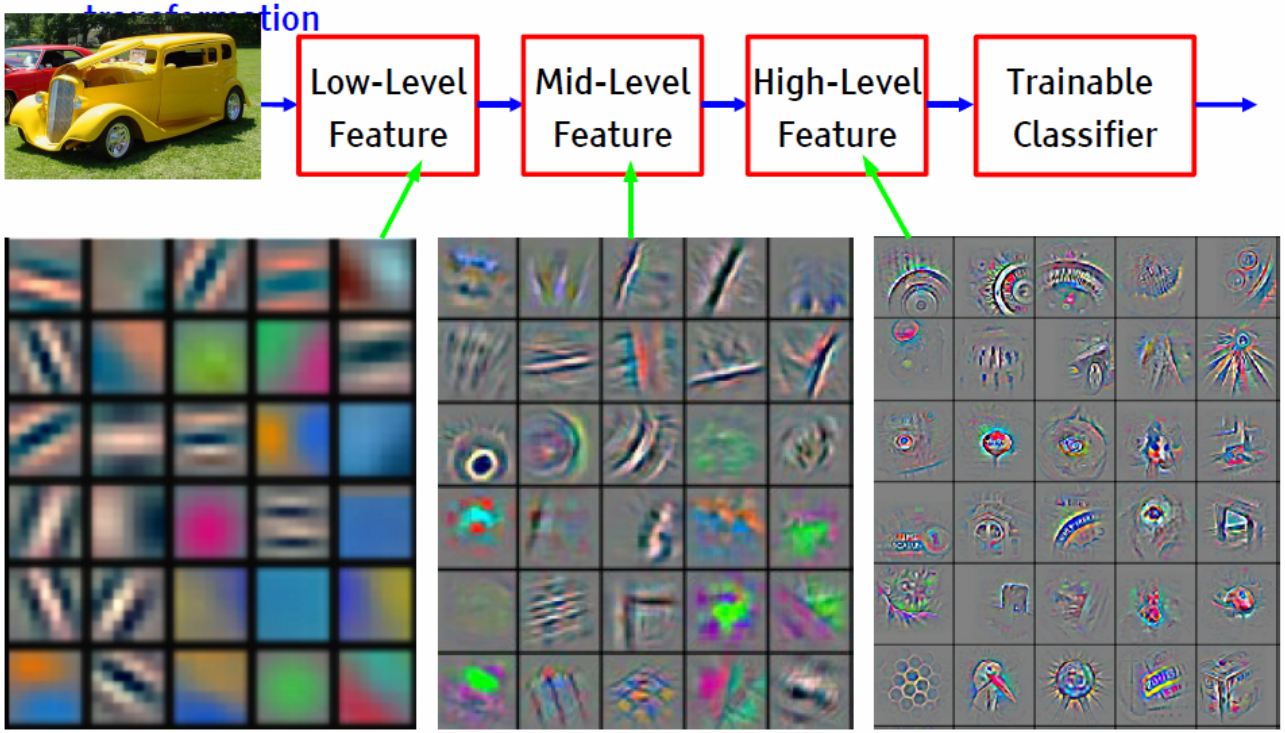
Convolutional Neural Networks

- Convolution layer
  - Small filter size, fewer parameters
  - Efficient computation
- Convolution can be computed with fast fourier transform (FFT)





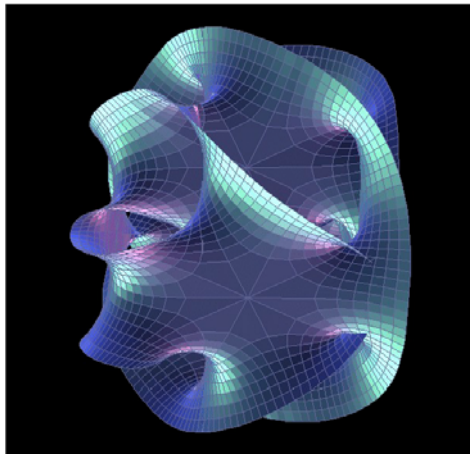
CNN Features Visualization



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

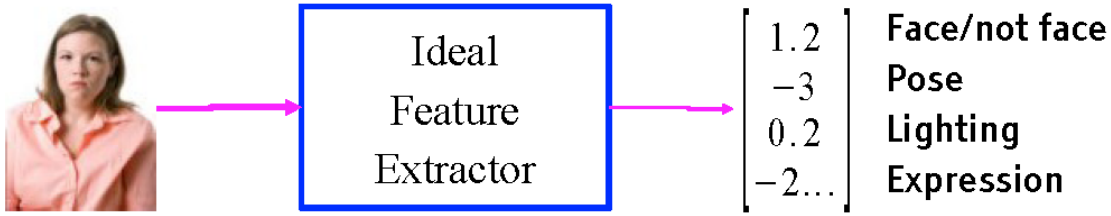
CNN Features: Manifold Hypothesis

- Learning Representations of Data:
  - **Discovering & disentangling the independent explanatory factors**
- The Manifold Hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data **is high dimensional**



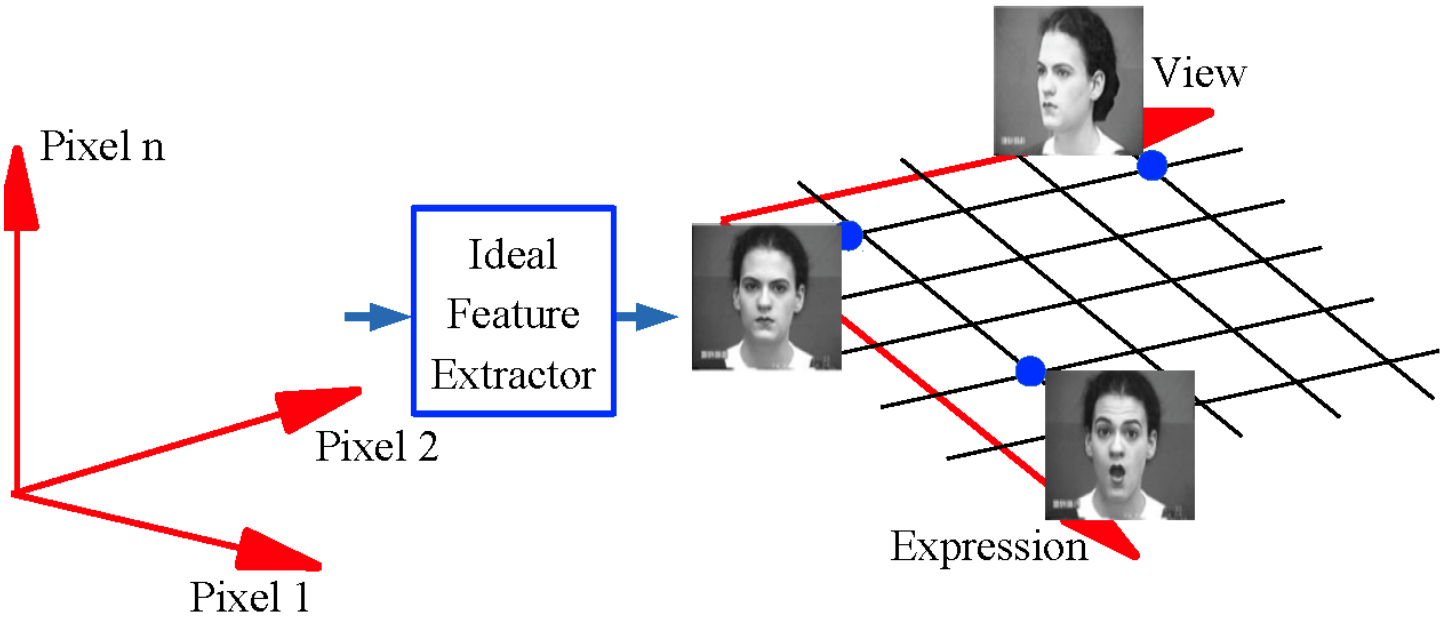
Hidden Structure in High Dimension

- Example: all face images of a person
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- The perfect representations of a face image:
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- We do not have good and general methods to learn functions that turns an image into this kind of representation



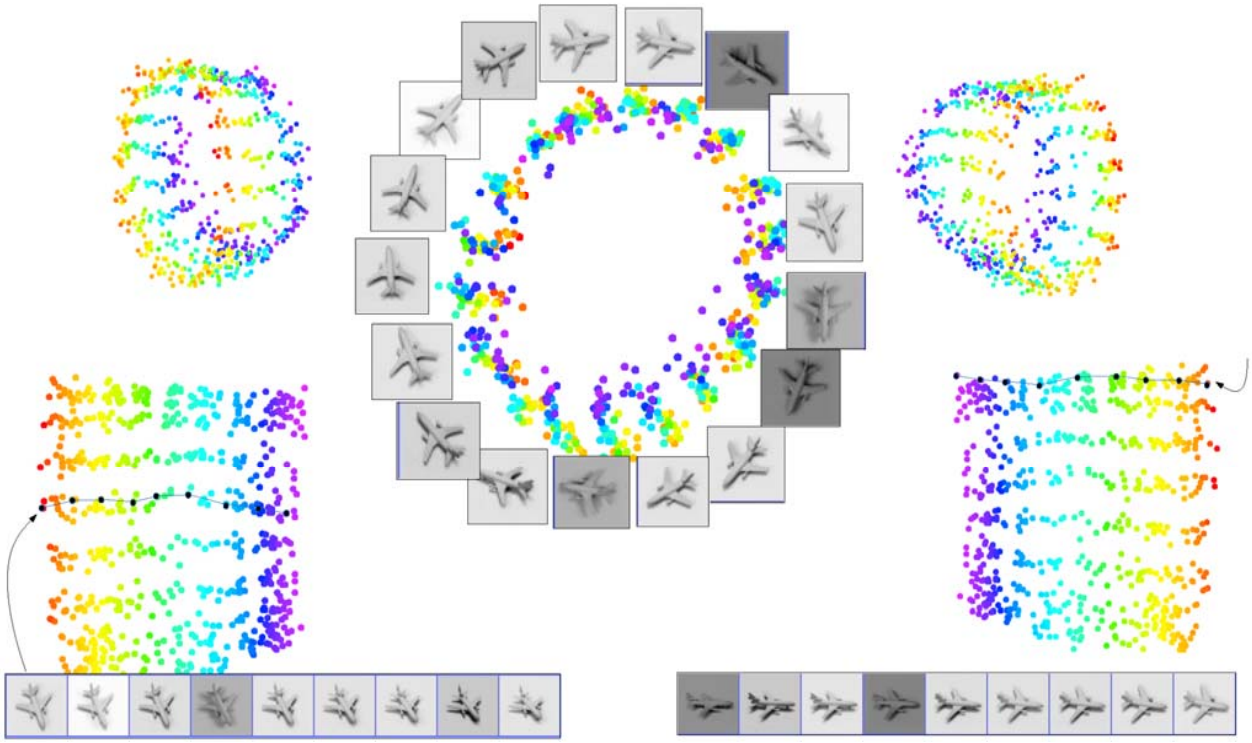
Disentangling Variations

- The Ideal Disentangling Feature Extractor



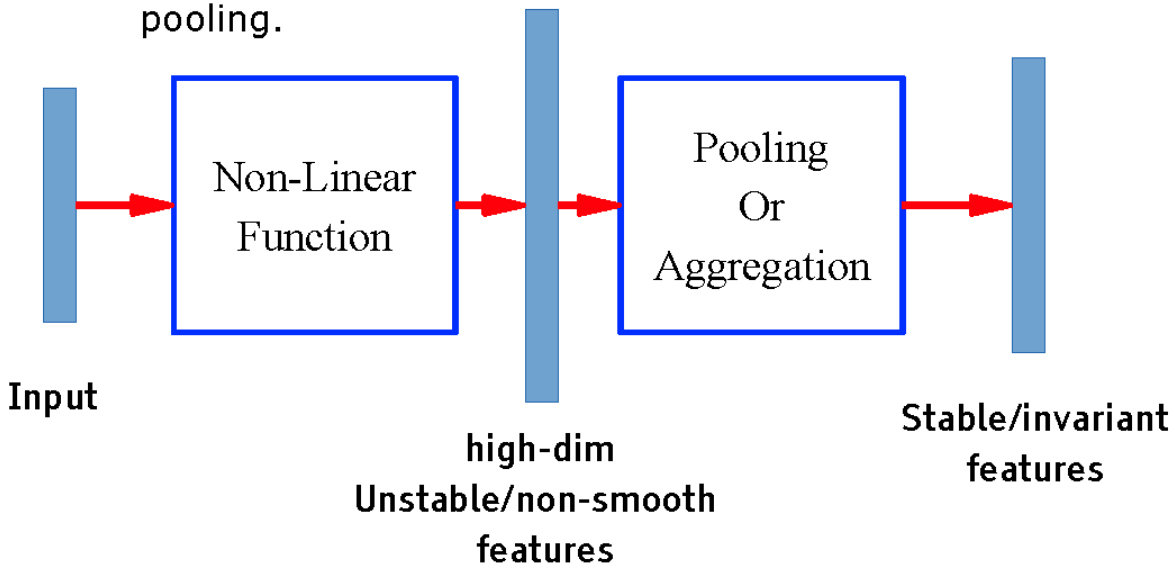
Data Manifolds and Invariance

Azimuth-Elevation manifold. Ignores lighting. [Hadsell et al. CVPR 2006]

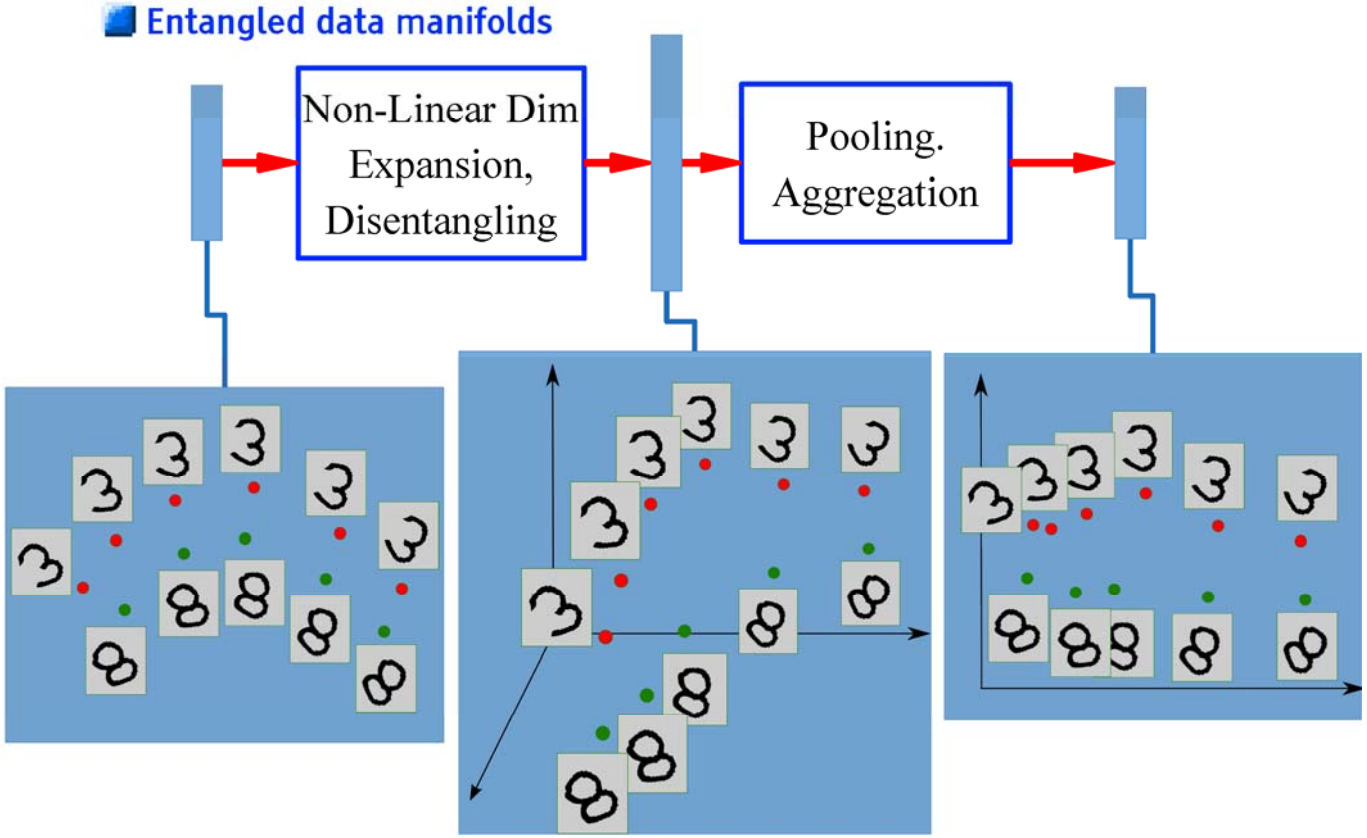


Invariant Feature Learning

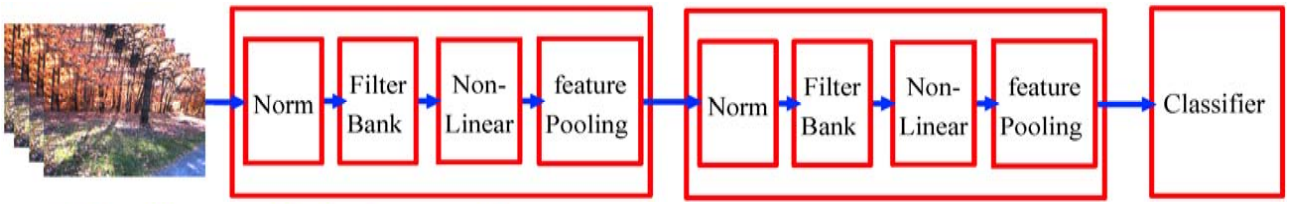
- Embed the input non-linearly into a high(er) dimensional space
  - In the new space, things that were non separable may become separable
- Pool regions of the new space together
  - Bringing together things that are semantically similar. Like pooling.



Pooling after Non-Linear Expansion



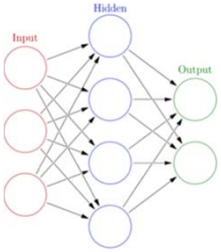
Overview of CNN networks

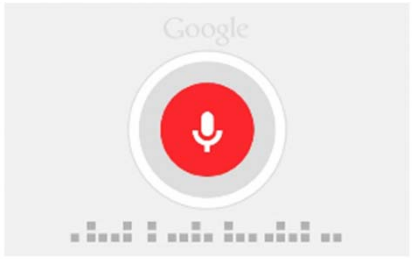


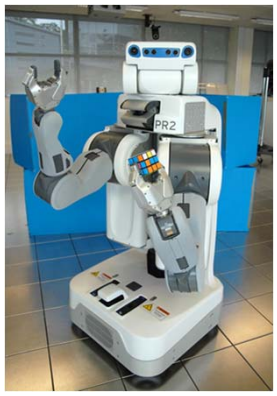
- Stacking multiple stages of
  - [Normalization → Filter Bank → Non-Linearity → Pooling].
- Normalization: variations on whitening
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- Filter Bank: dimension expansion, projection on overcomplete basis
- Non-Linearity: sparsification, saturation, lateral inhibition....
  - Rectification (ReLU), Component-wise shrinkage, tanh, winner-takes-all
- Pooling:
  - **aggregation over space or feature type**

Why Deep Learning?

End-to-End Learning for Many Tasks







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 → learning → abstract

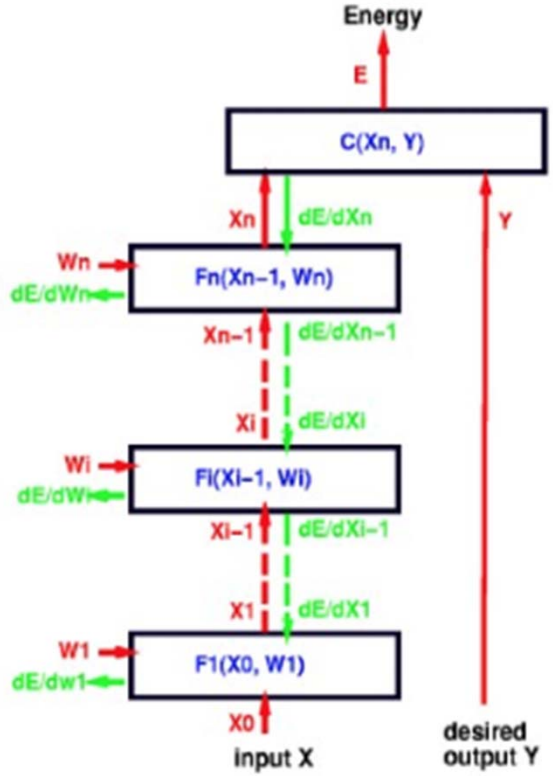


figure credit Yann LeCun, ICML '13 tutorial

What is Deep Learning?

Compositional Models
Learned End-to-End

Back-propagation jointly learns all of the model parameters to optimize the output for the task.

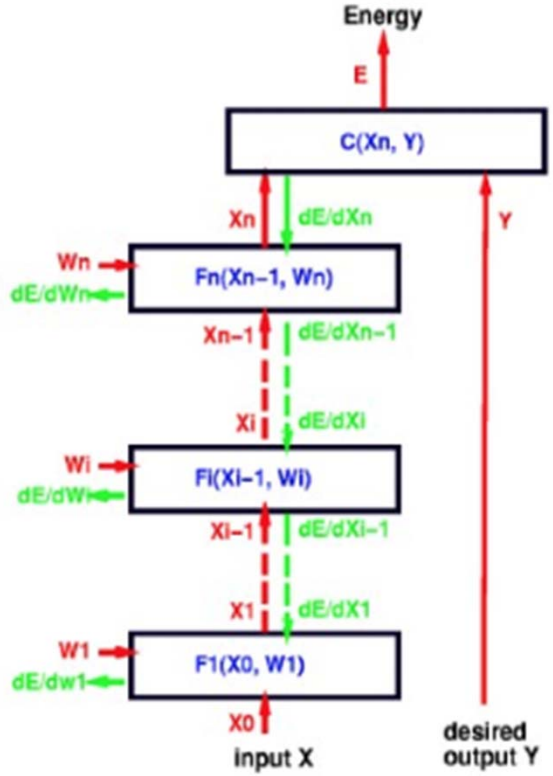


figure credit Yann LeCun, ICML '13 tutorial

Deep Learning Libraries

- Caffe
  - http://caffe.berkeleyvision.org
- Torch
  - http://torch.ch/
- Theano
  - http://deeplearning.net/software/theano/

What is Caffe?

Open framework, models, and worked examples
for deep learning

- < 2 years
- 600+ citations, 100+ contributors, 6,000+ stars
- 3,400+ 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 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

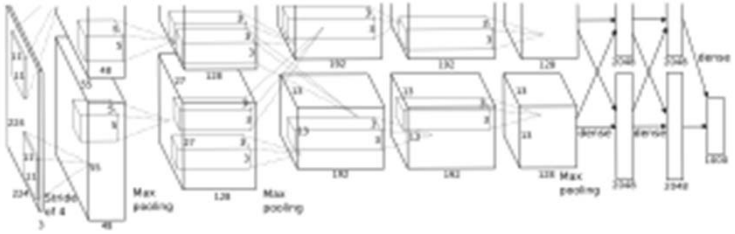
Prototype

Train

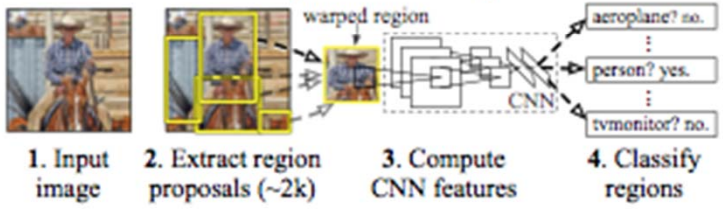
Deploy

Reference Models

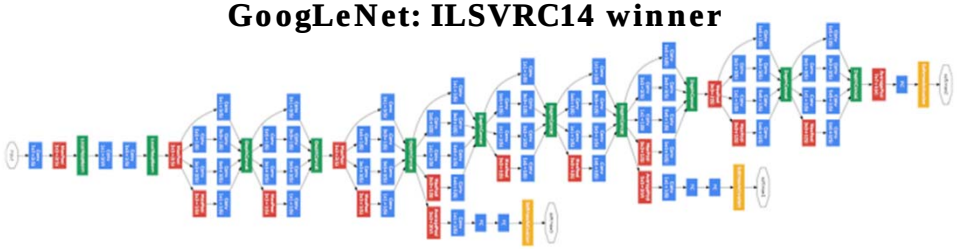
AlexNet: ImageNet Classification



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
    - 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

Brewing by the Numbers...

Speed with Krizhevsky's 2012 model:

- 2 ms / image on K40 GPU
- <1 ms inference with Caffe + cuDNN v2 on Titan X
- 72 million images / day with batched IO
- 8-core CPU: ~20 ms/image

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

CAFFE
EXAMPLES + APPLICATIONS

Scene Recognition http://places.csail.mit.edu/

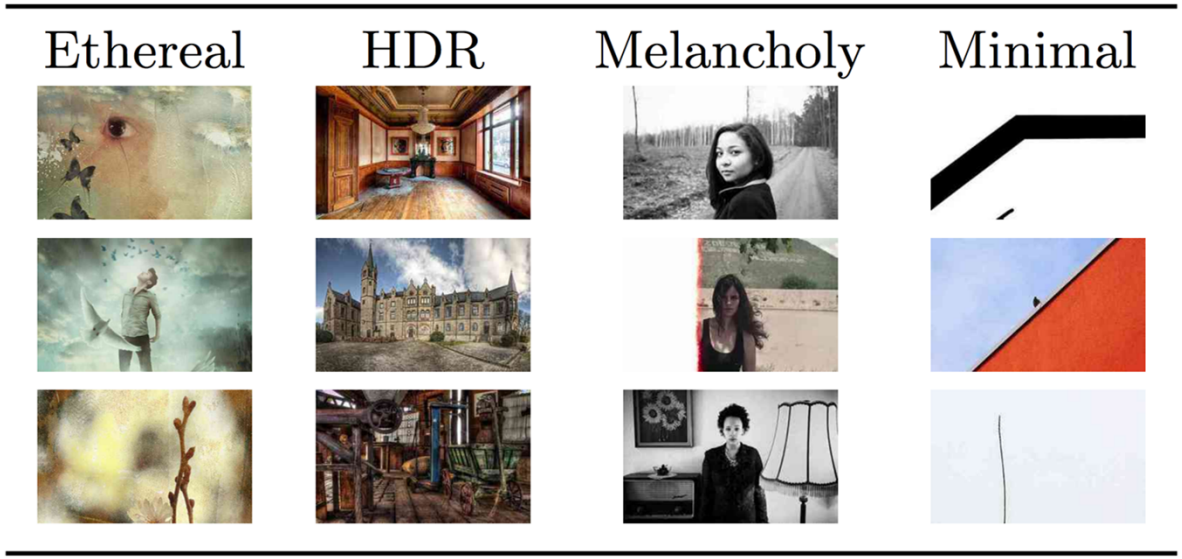
Predictions:

- **Type of environment:** outdoor
- **Semantic categories:** skyscraper:0.69, tower:0.16, office_building:0.11,
- **SUN scene attributes:** man-made, vertical components, natural light, open area, nohorizon, glossy, metal, wire, clouds, far-away horizon

B. Zhou et al. NIPS 14

Visual Style Recognition

Karayev et al. *Recognizing Image Style*. BMVC14. Caffe fine-tuning example.
Demo online at http://demo.vislab.berkeleyvision.org/ (see Results Explorer).



[Image-Style]

Other Styles:

Vintage
Long Exposure
Noir
Pastel
Macro
… and so on.

Object Detection

R-CNN: Region-based Convolutional 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.

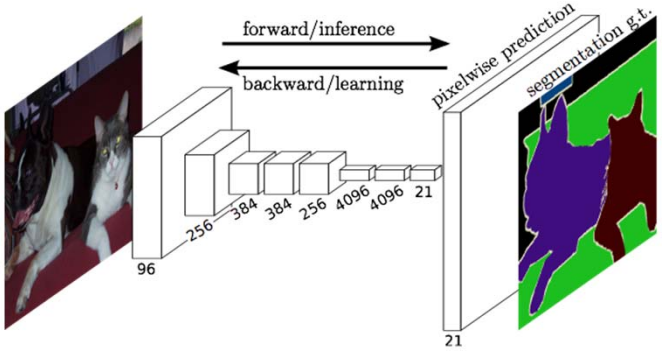
Segmentation

Fully convolutional networks for pixel prediction applied to semantic segmentation

- end-to-end learning
- efficiency in inference and learning
  175 ms per-image prediction
- multi-modal, multi-task

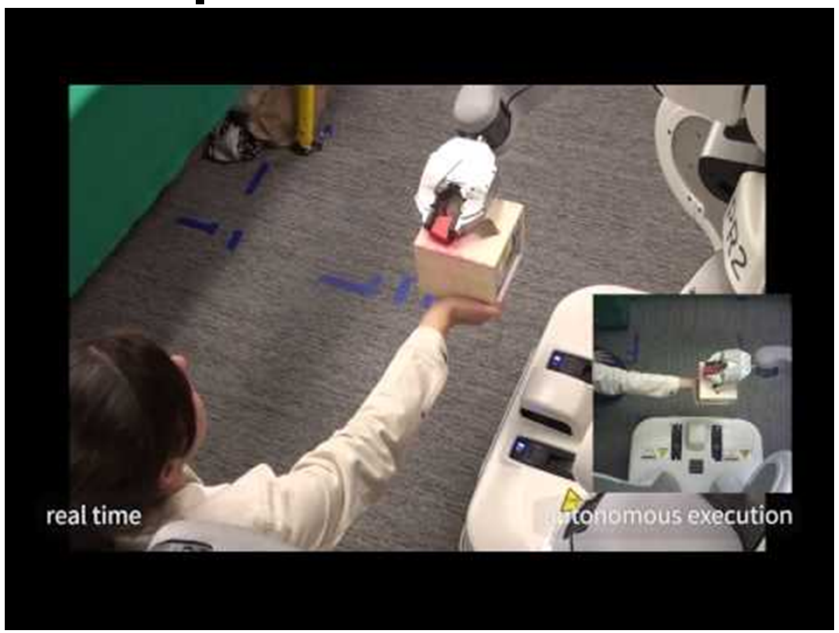
Further applications

- depth estimation
- denoising

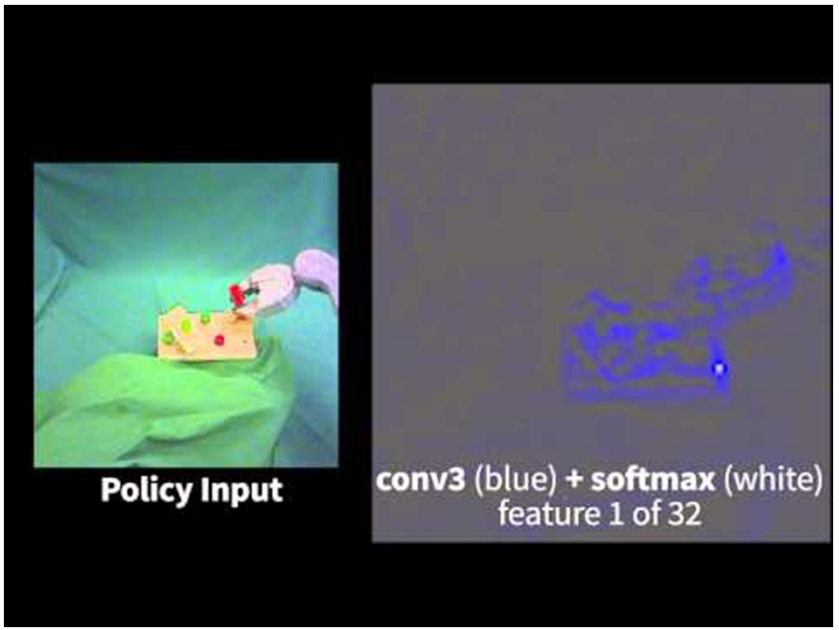


Jon Long* & Evan Shelhamer*, Trevor Darrell

Deep Visuomotor Control



example experiments

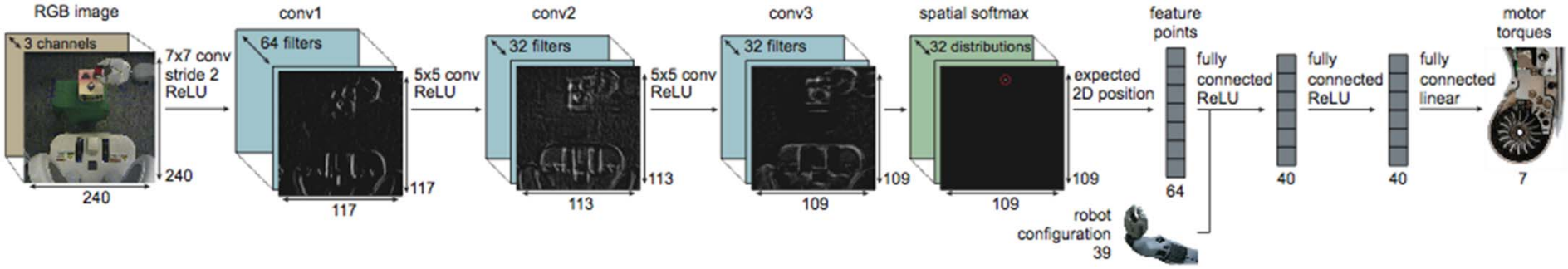


feature visualization

tech report http://tinyurl.com/visuomotor

Sergey Levine* & Chelsea Finn*,
Trevor Darrell, and Pieter Abbeel

Deep Visuomotor Control



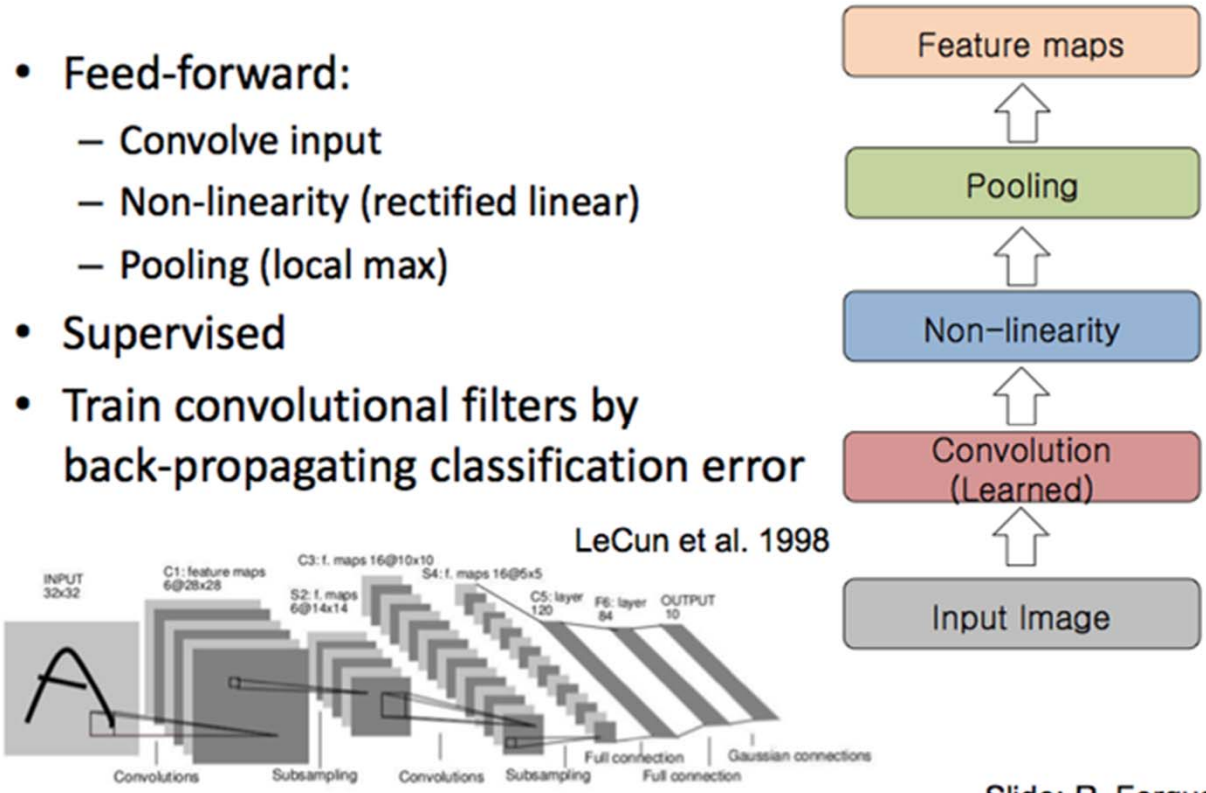
- multimodal (images & robot configuration)
- learned end-to-end
- runs at 20 Hz - mixed GPU & CPU
  for real-time control

tech report http://tinyurl.com/visuomotor

Sergey Levine* & Chelsea Finn*,
Trevor Darrell, and Pieter Abbeel

Convolutional Network

- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating classification error



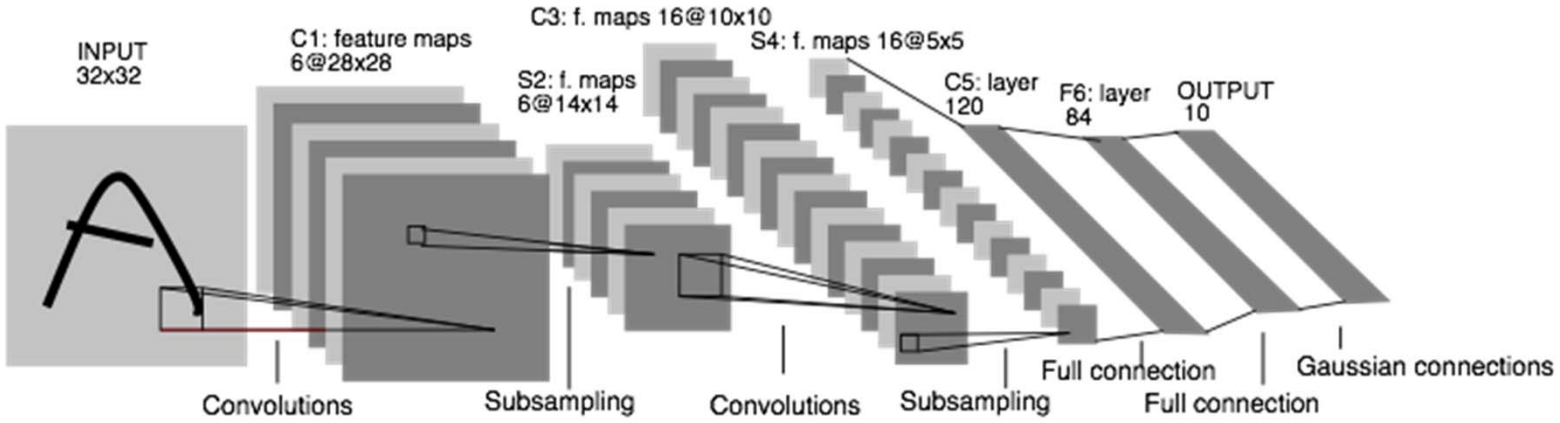
LeCun et al. 1998

Slide: R. Fergus

Classification

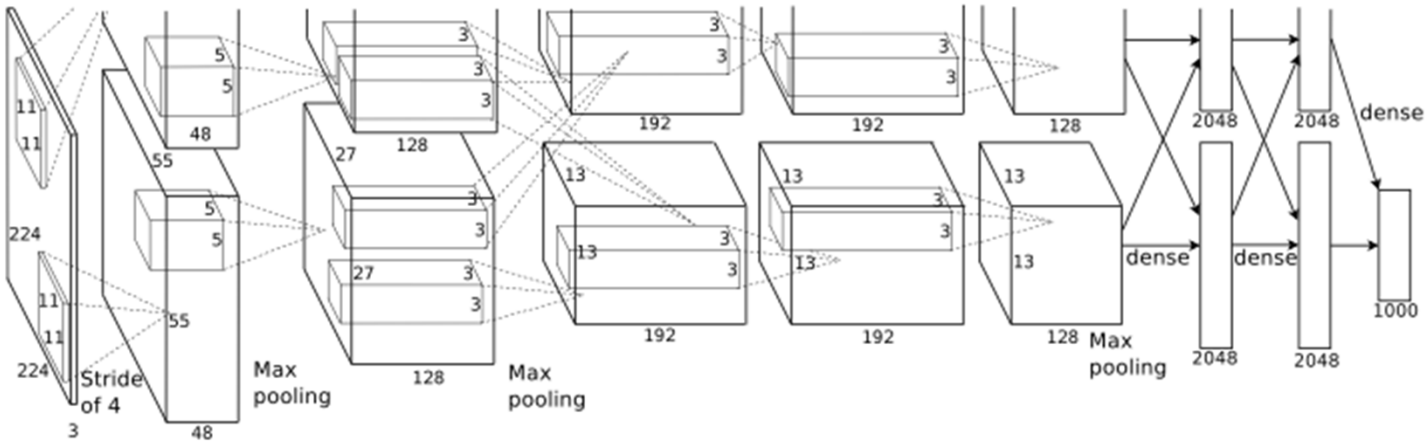
instant recognition the Caffe way

Convolutional Networks: 1989



LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

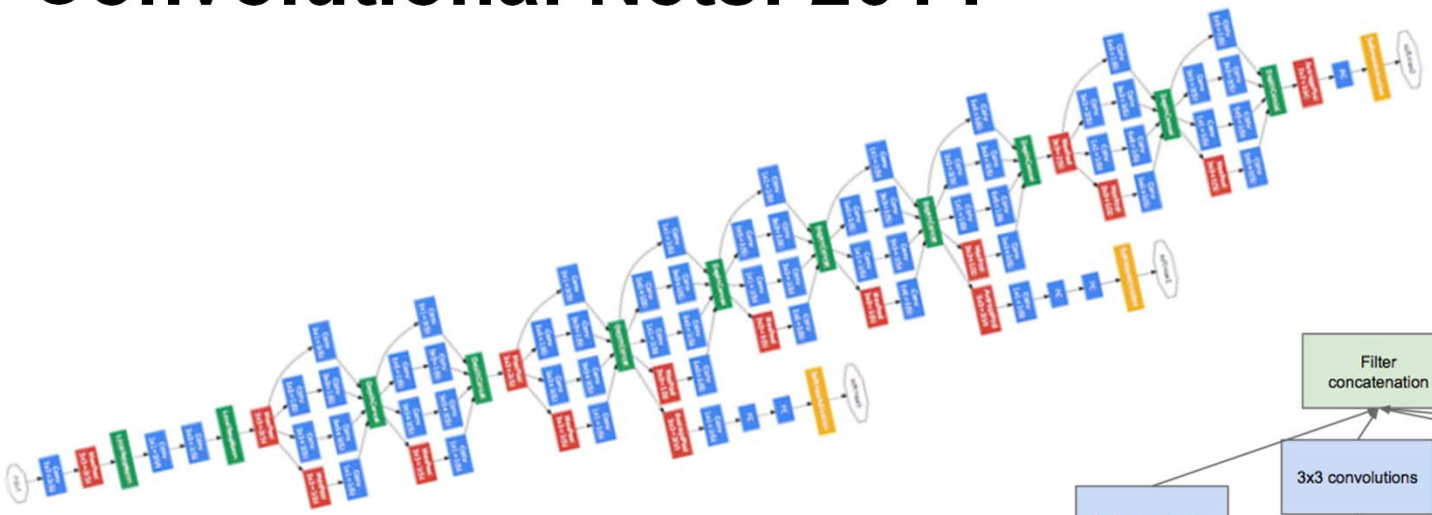
Convolutional Nets: 2012

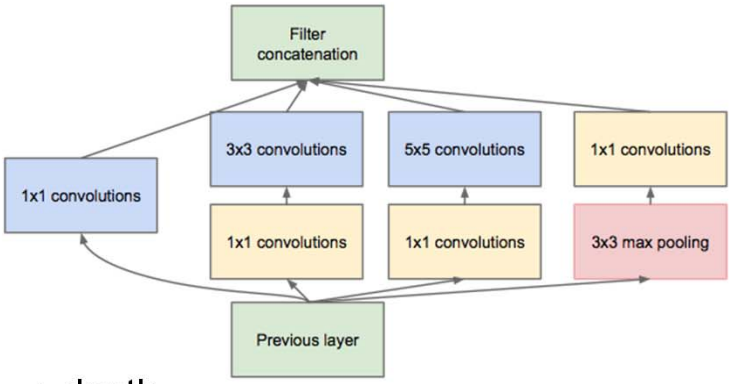


AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12. [AlexNet]

+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

Convolutional Nets: 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 LeNet

back to the future of visual recognition

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!

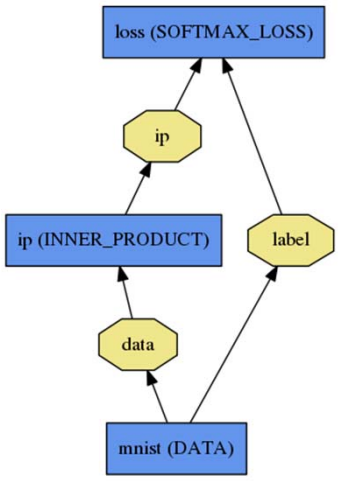
Net

A network is a set of layers and their connections:

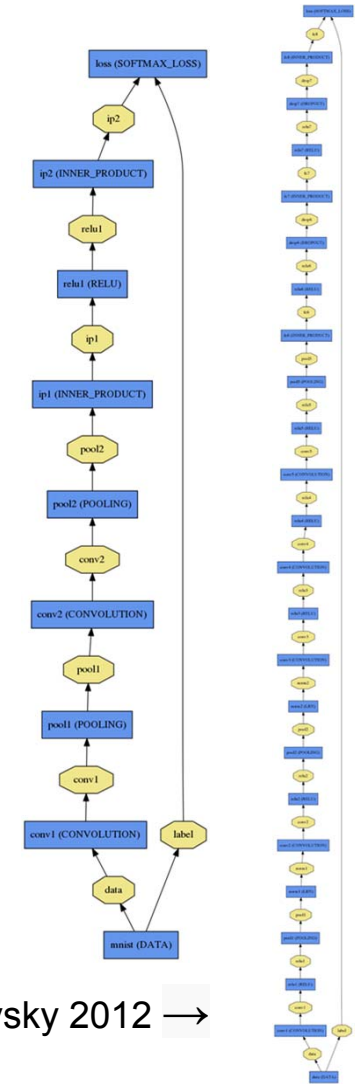
```
name: "dummy-net"
layer { name: "data" …}
layer { name: "conv" …}
layer { name: "pool" …}
    … more layers …
layer { name: "loss" …}
```

Caffe creates and checks the net from the definition.

Data and derivatives flow through the net as *blobs* – an array interface



LogReg ↑

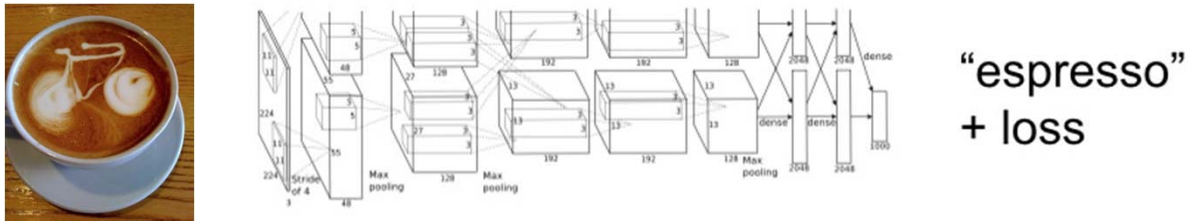


LeNet ⟶

ImageNet, Krizhevsky 2012 ⟶

Forward / Backward the essential Net computations

Forward: inference $f_W(x)$



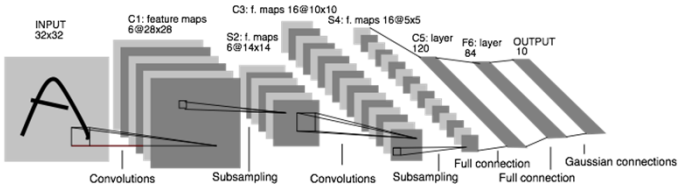
"espresso" + loss

$\nabla f_W(x)$ 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.

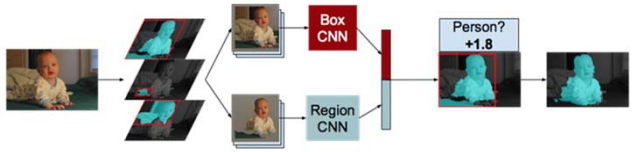
DAG

Many current deep models have linear structure

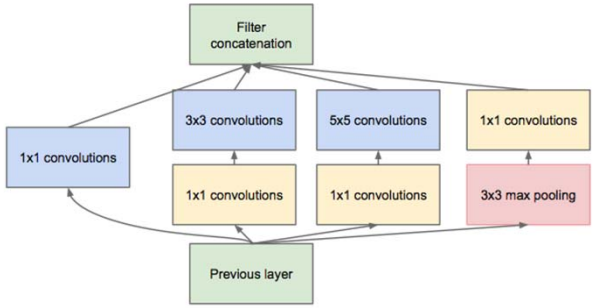


but Caffe nets can have any directed acyclic graph (DAG) structure.

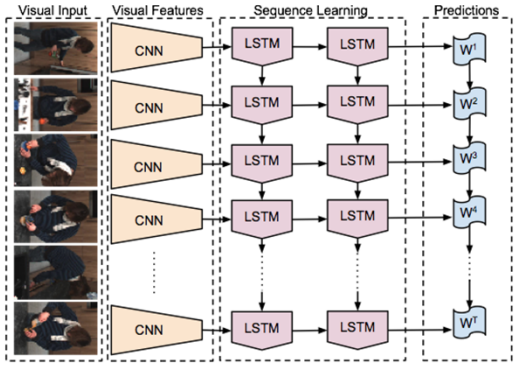
Define `bottoms` and `tops` and Caffe will connect the net.



SDS two-stream net



GoogLeNet Inception Module



LRCN joint vision-sequence model

Layer Protocol

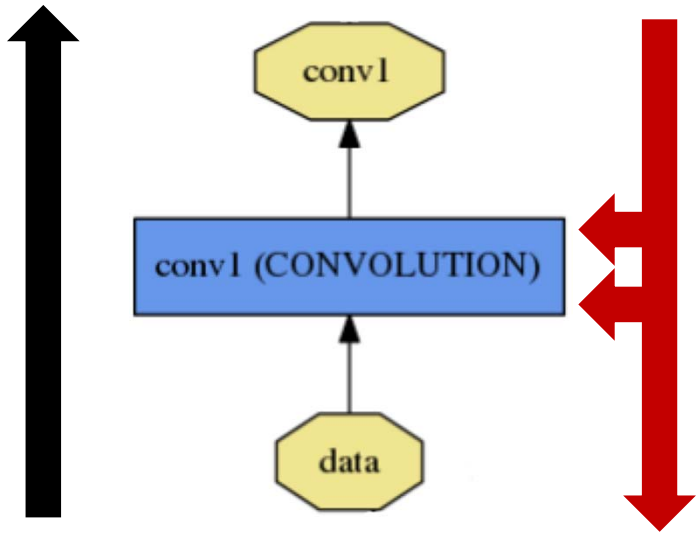
Setup: run once for initialization.

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

```python
import caffe
import numpy as np

class EuclideanLoss(caffe.Layer):

    def setup(self, bottom, top):
        # check input pair
        if len(bottom) != 2:
            raise Exception("Need two inputs to compute distance.")

    def reshape(self, bottom, top):
        # check input dimensions match
        if bottom[0].count != bottom[1].count:
            raise Exception("Inputs must have the same dimension.")
        # difference is shape of inputs
        self.diff = np.zeros_like(bottom[0].data, dtype=np.float32)
        # loss output is scalar
        top[0].reshape(1)

    def forward(self, bottom, top):
        self.diff[...] = bottom[0].data - bottom[1].data
        top[0].data[...] = np.sum(self.diff**2) / bottom[0].num / 2.

    def backward(self, top, propagate_down, bottom):
        for i in range(2):
            if not propagate_down[i]:
                continue
            if i == 0:
                sign = 1
            else:
                sign = -1
            bottom[i].diff[...] = sign * self.diff / bottom[i].num
```

Layer Protocol == Class Interface

Define a class in C++ or Python to extend `Layer`.

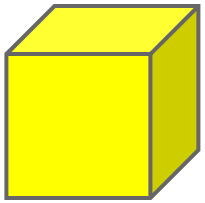
Include your new layer type in a network and keep brewing.

```
layer {
type: "Python"
python_param {
  module: "layers"
  layer: "EuclideanLoss"
} }
```

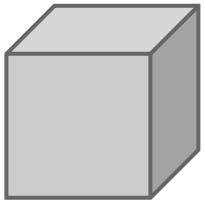
Blob

Blobs are N-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

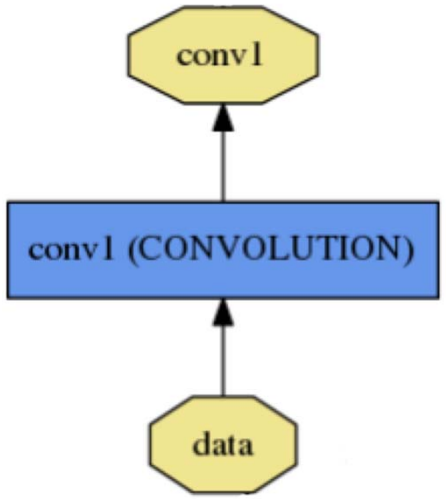


Parameter: Convolution Weight
N Output x *K* Input x *H*eight x *W*idth
96 x 3 x 11 x 11 for CaffeNet conv1

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

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

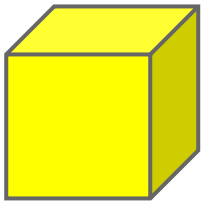
top
blob



bottom
blob

Blob

Blobs provide a unified memory interface.

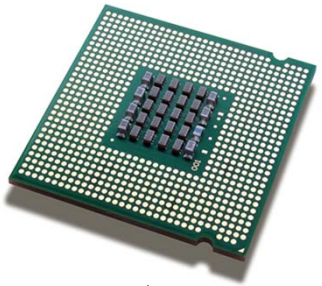


Reshape(num, channel, height, width)
- declare dimensions
- make *SyncedMem* -- but only lazily allocate

cpu_data(), mutable_cpu_data()
- host memory for CPU mode

gpu_data(), mutable_gpu_data()
- device memory for GPU mode

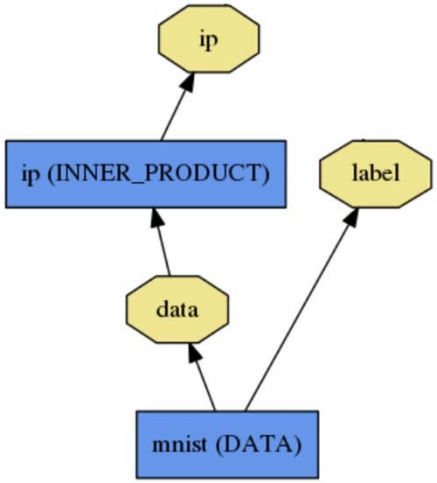
{cpu,gpu}_diff(), mutable_{cpu,gpu}_diff()
- derivative counterparts to data methods
- easy access to data + diff in forward / backward

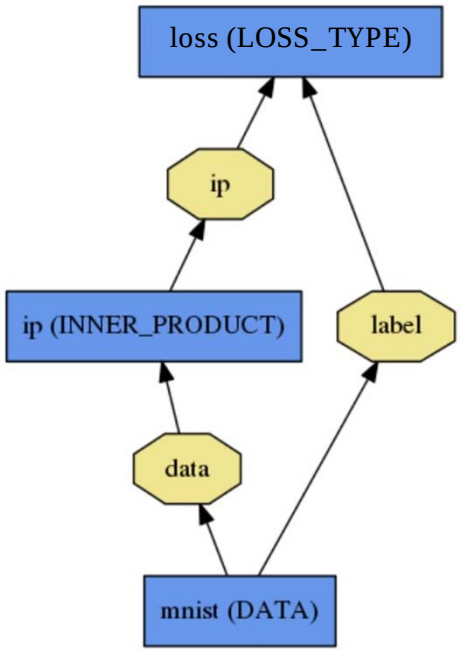


SyncedMem
allocation + communication

Loss

What kind of model is this?





Classification
`SoftmaxWithLoss`
`HingeLoss`

Linear Regression
`EuclideanLoss`

Attributes / Multiclassification
`SigmoidCrossEntropyLoss`

Others…

New Task
`NewLoss`

Define the task by the **loss**.

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"
    }
}
```

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

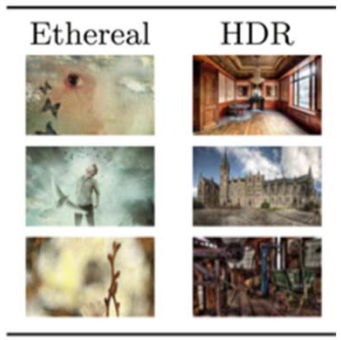
Take a pre-trained model and fine-tune to new tasks

[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

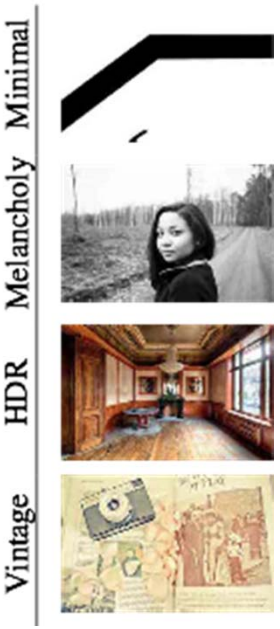
Last Layer: A different classifier

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

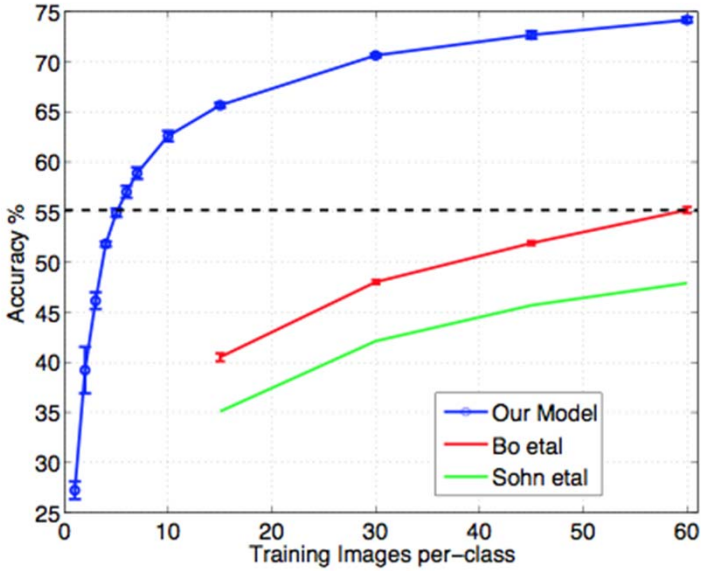
When to Fine-tune?

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

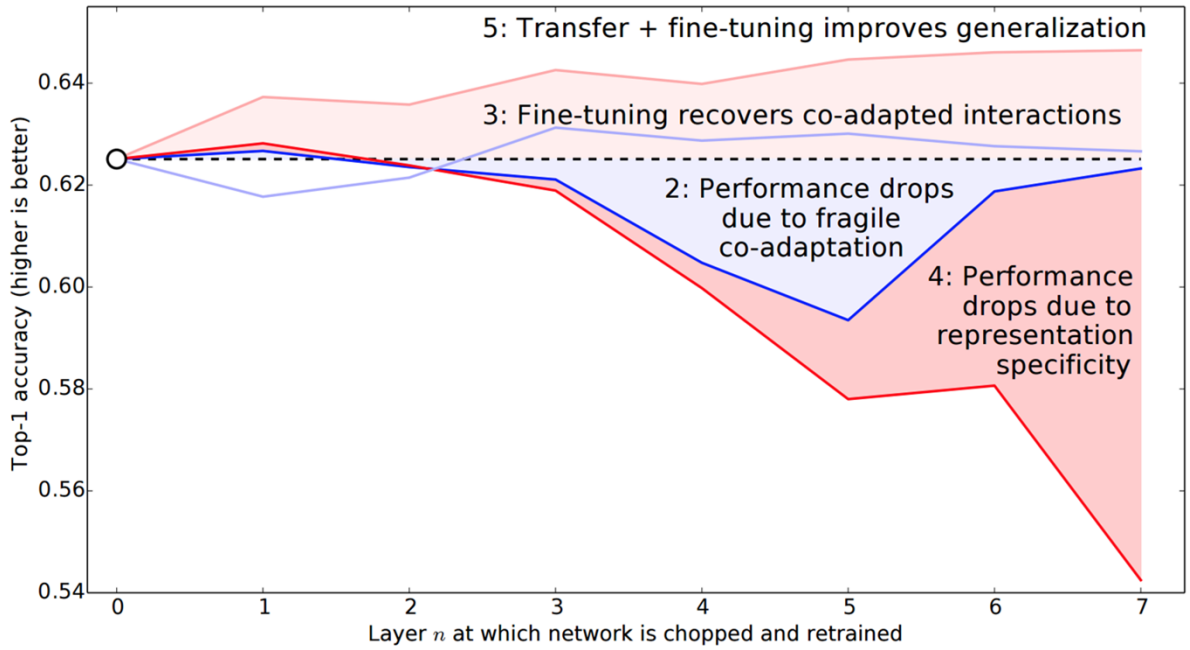
Reduce the learning rate

- 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

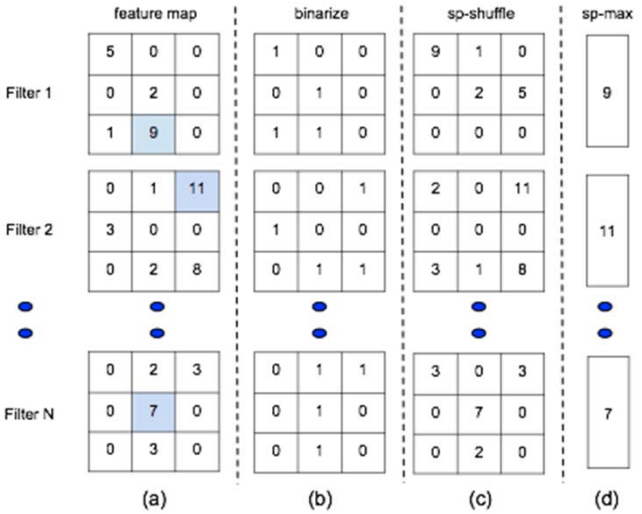
Transferability of Features

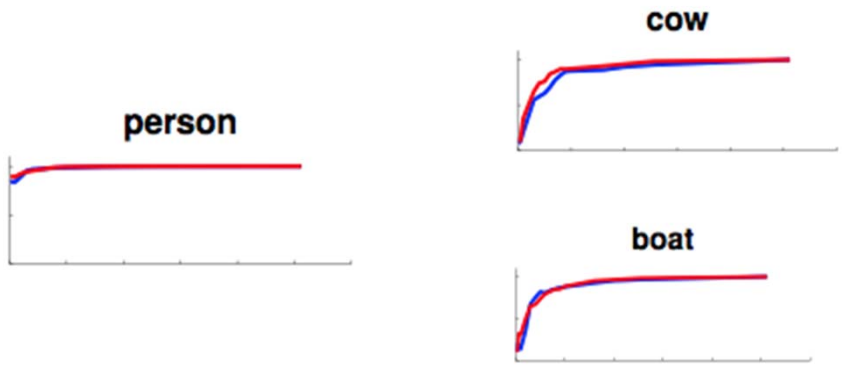
Yosinski et al. NIPS 2014



After fine-tuning

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





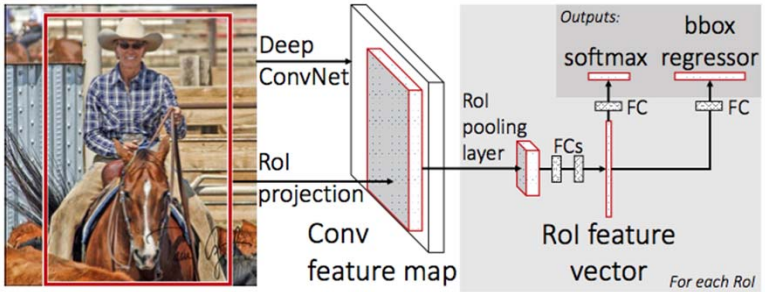
P. Agarwal et al. ECCV 14

Editing model parameters

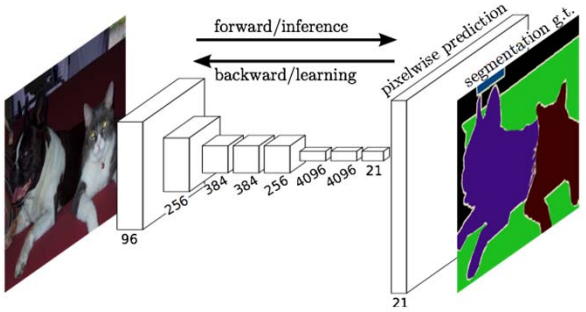
how to do net surgery to set custom weights

Up Next The Latest Roast

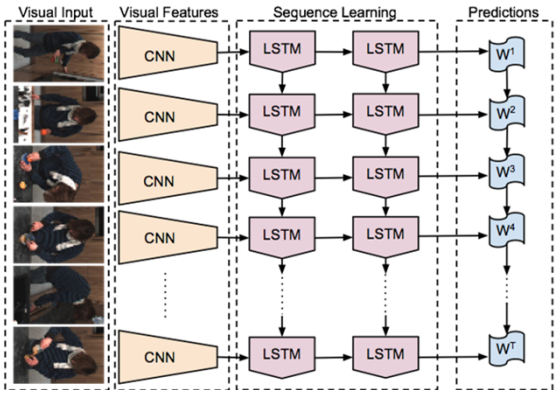
Detection



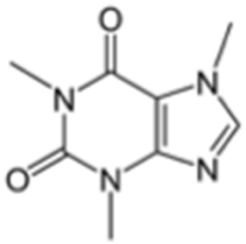
Pixelwise Prediction



Sequences

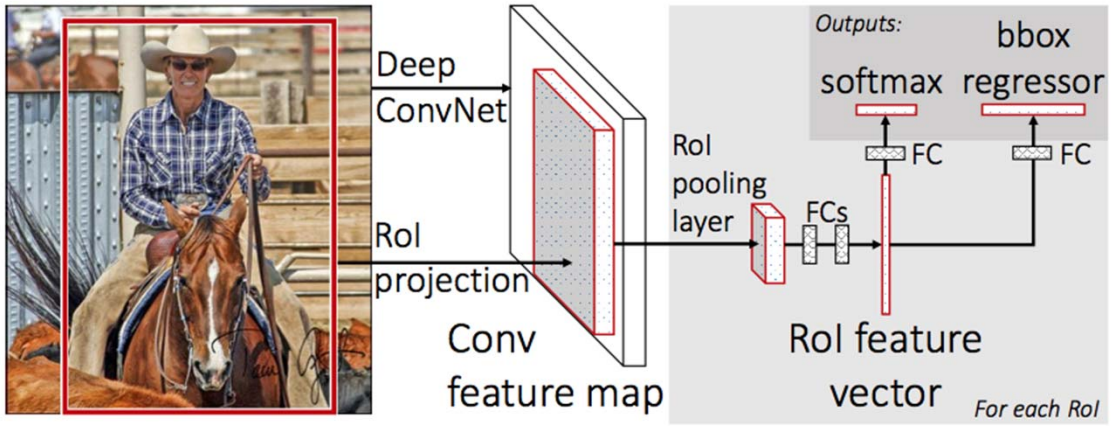


Framework Future



Detection

Fast R-CNN
- convolve once
- project + detect



Faster R-CNN
- end-to-end proposals and detection
- 200 ms / image inference
- fully convolutional Region Proposal Net
  + Fast R-CNN

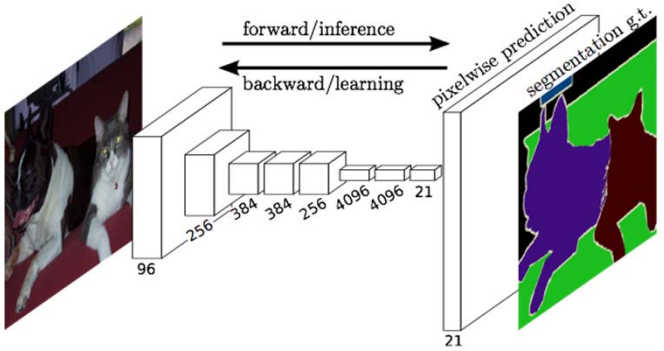
arXiv and code for Fast R-CNN

Ross Girshick, Shaoqing Ren,
Kaiming He, Jian Sun

Pixelwise Prediction

Fully convolutional networks for pixel prediction applied to semantic segmentation

- end-to-end learning
- efficient inference and learning
  150 ms per-image prediction
- multi-modal, multi-task



Further applications

- depth
- boundaries
- flow + more

CVPR15 arXiv and pre-release

Jon Long* & Evan Shelhamer*, Trevor Darrell

Major References

[NNDL] Michael Nielsen, Online book: Neural Networks and Deep Learning http://neuralnetworksanddeeplearning.com/
[Caffe] Evan Shelhamer, Jeff Donahue, Jon Long, Yangqing Jia, and Ross Girshick, Caffe Tutorial for CVPR 2015 http://tutorial.caffe.berkeleyvision.org/
[DLT] Yann LeCun, Marc'Aurelio Ranzato, Deep Learning Tutorial, ICML, Atlanta, 2013
[AlexNet] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. NIPS, 2012.
[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.

Thanks