Convolutional Neural Networks

Overview

Goal: Understand what Convolutional Neural Networks (ConvNets) are & intuition behind it.

1. Brief Motivation for Deep Learning
2. What are ConvNets?
3. ConvNets for Object Detection

First of all what is Deep Learning?

- Composition of non-linear transformation of the data.
- Goal: **Learn useful representations, aka features, directly from data.**

Recap: Supervised Learning

$\{(\boldsymbol{x}^i, y^i), i = 1 \ldots P\}$ training dataset

$\boldsymbol{x}^i$ i-th input training example

y^i i-th target label

P number of training examples

Slide: M. Ranzato

Supervised Learning: Examples

Classification

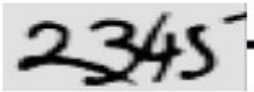
“dog”

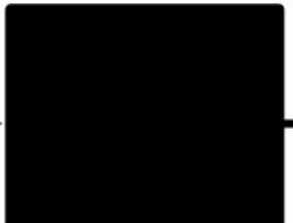
classification

Denoising

regression

OCR





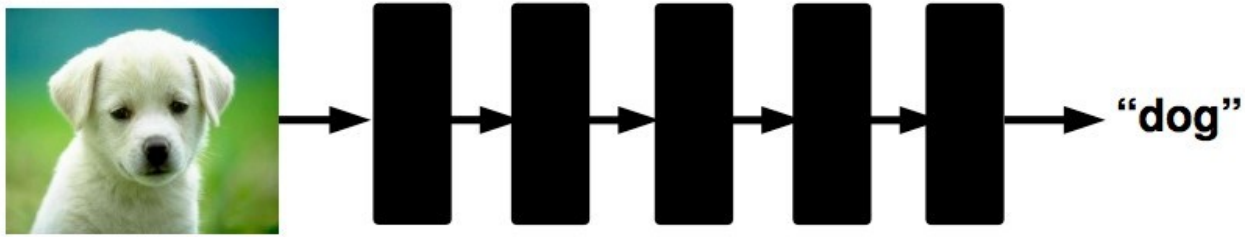
“2 3 4 5”

structured prediction

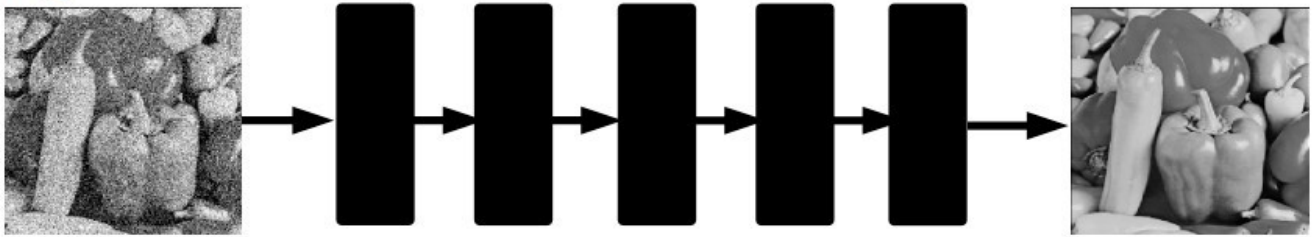
Slide: M. Ranzato

Supervised Deep Learning

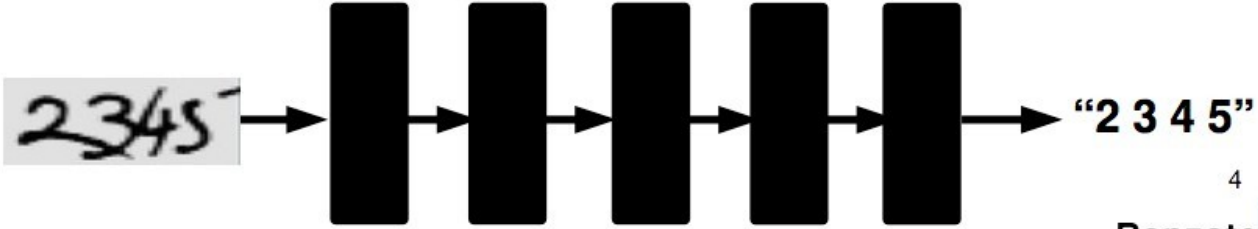
Classification



Denoising



OCR



So deep learning is about learning feature representation in a compositional manner.

But wait,

why learn features?

The Black Box in a Traditional Recognition Approach

“dog”

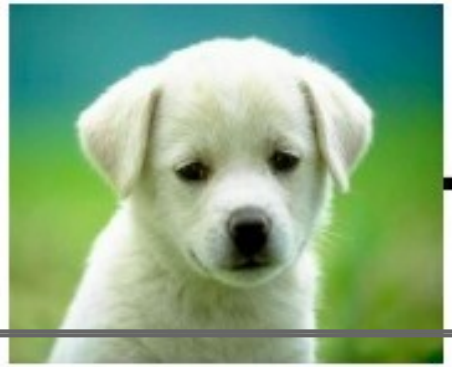
Preprocessing

Feature Extraction (HOG, SIFT, etc)

Post-processing (Feature selection, MKL etc)

Classifier (SVM, boosting, etc)

The Black Box in a Traditional Recognition Approach



"dog"

Hand Engineered

Preprocessing

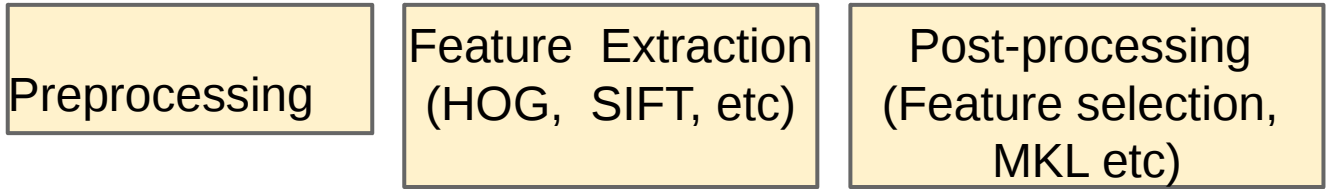
Feature Extraction (HOG, SIFT, etc)

Post-processing (Feature selection, MKL etc)

Classifier (SVM, boosting, etc)

Local Constancy and Smoothness Regularization

- Implicit and explicit beliefs
- Overcome the limitations of prior beliefs (eg. local template matching)
  - Additional Beliefs
  - Composition of factors or features at multiple levels in a hierarchy



- Most critical for accuracy
- Most time-consuming in development
- What is the best feature???
- What is next?? Keep on crafting better features?
- Let’s learn feature representation directly from data.

Learn features and classifier *together*

⇒ Learn an end-to-end recognition system.

A non-linear map that takes raw pixels directly to labels.

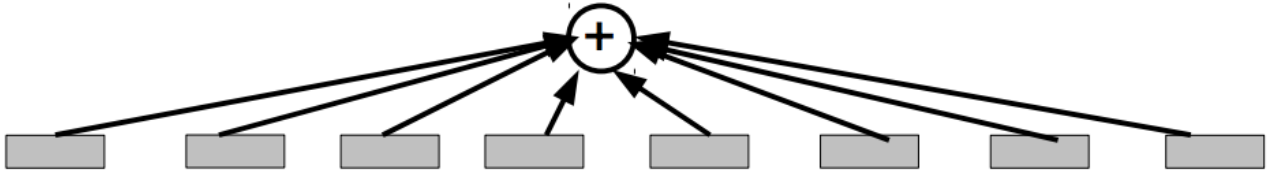
Q: How can we build such a highly non-linear system?

A: By combining simple building blocks we can make more and more complex systems.

Slide: M. Ranzato

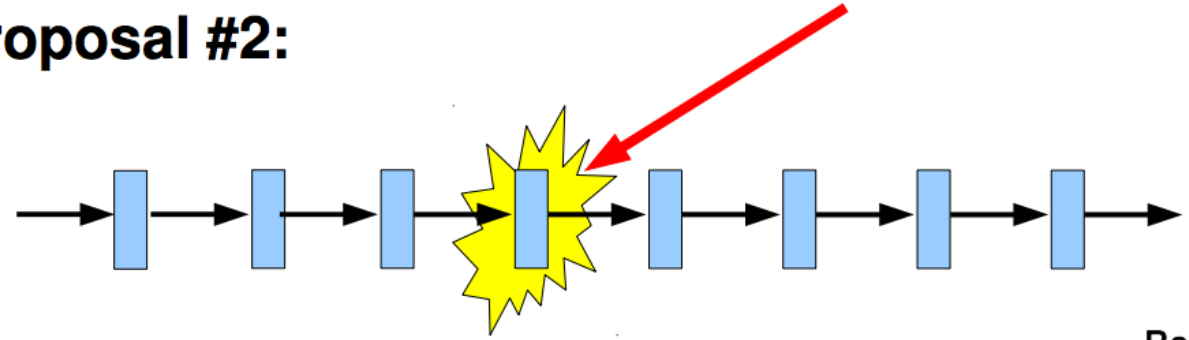
Building a complicated function

Proposal #1:

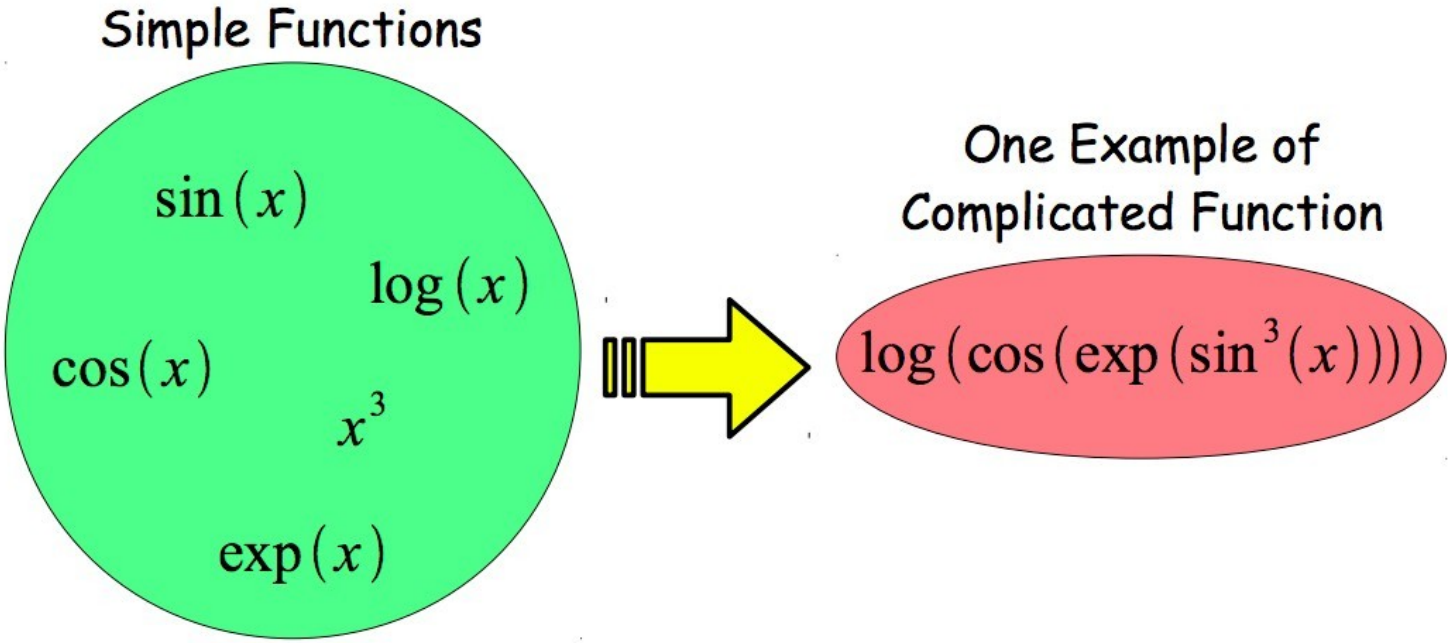


Each box is a simple nonlinear function

Proposal #2:

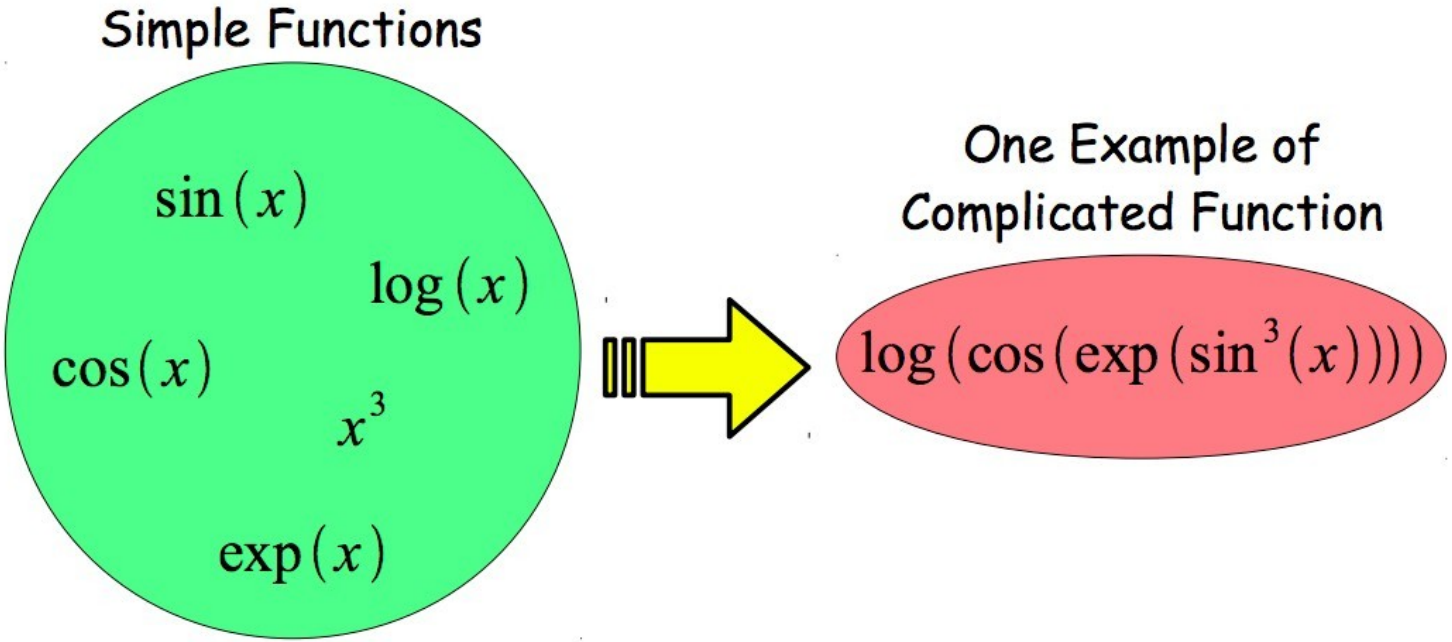


Building a complicated function



Slide: M. Ranzato

Building a complicated function



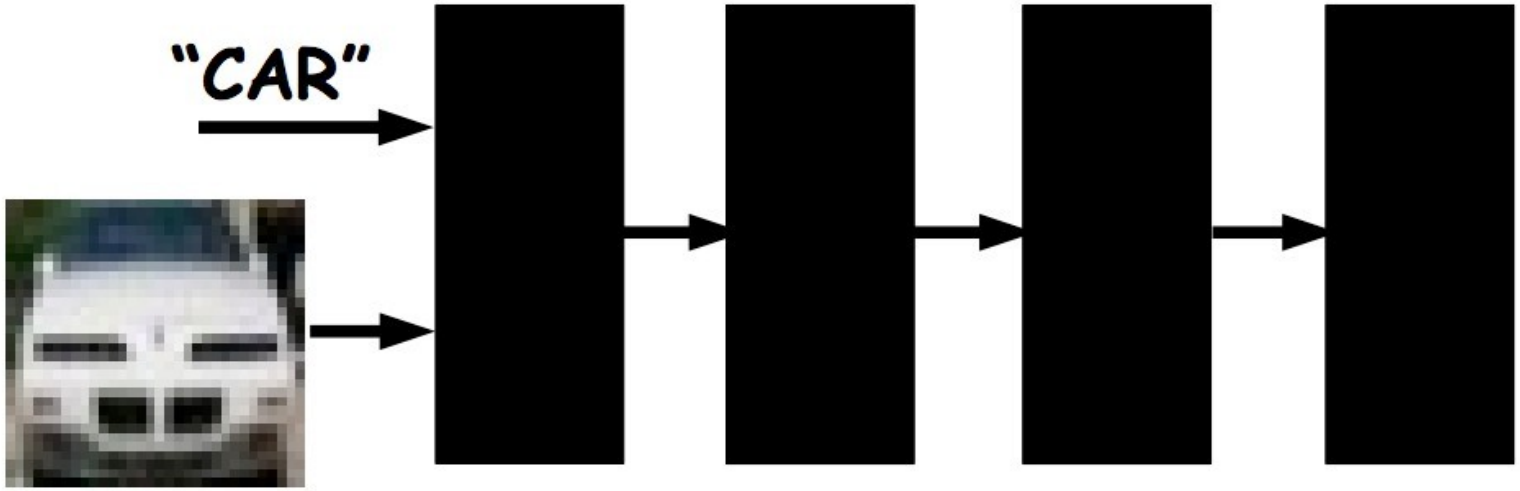
- Composition is at the core of deep learning methods
- Each “simple function” will have parameters subject to learning

Slide: M. Ranzato

Intuition behind Deep Neural Nets

Slide: M. Ranzato

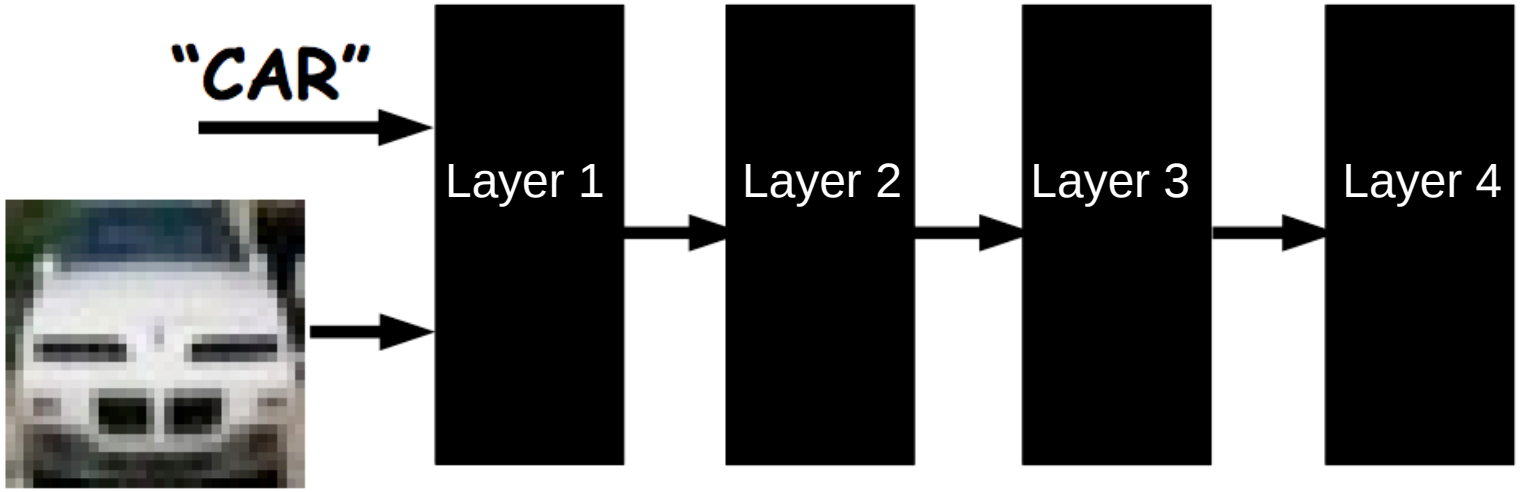
Intuition behind Deep Neural Nets



NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

Slide: M. Ranzato

Intuition behind Deep Neural Nets



NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

The final layer outputs a probability distribution of categories.

Slide: M. Ranzato

A simple single layer Neural Network

Consists of a linear combination of input through a nonlinear function:

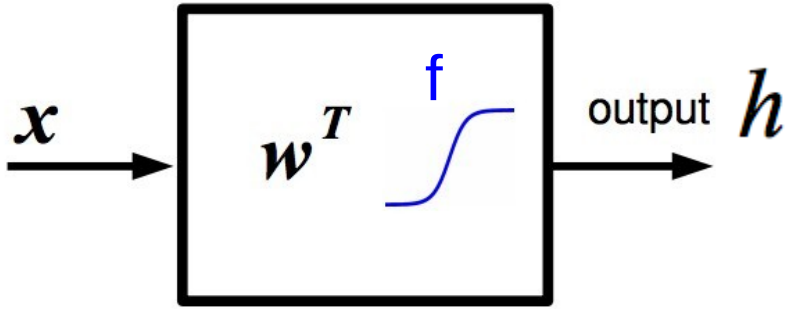
$$z = Wx + b$$
$$a = f(z)$$

W is the weight parameter to be learned. x is the output of the previous layer

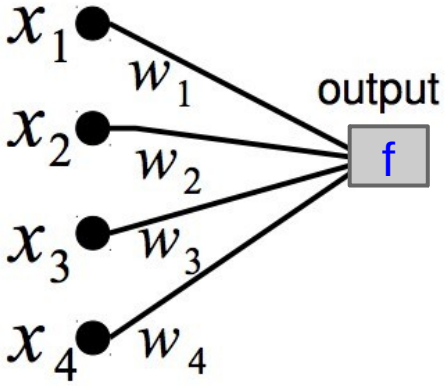
f is a simple nonlinear function. Popular choice is max(x,0), called ReLu (Rectified Linear Unit)

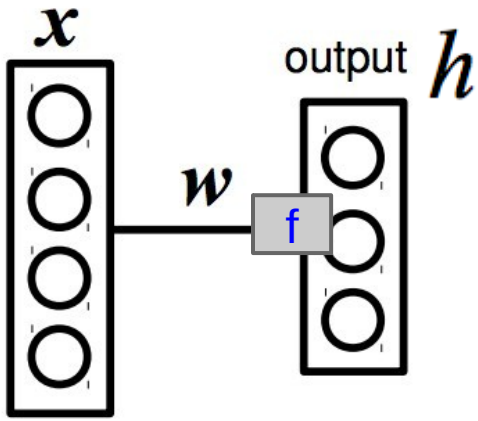
1 layer: Graphical Representation

$f(X;W)$

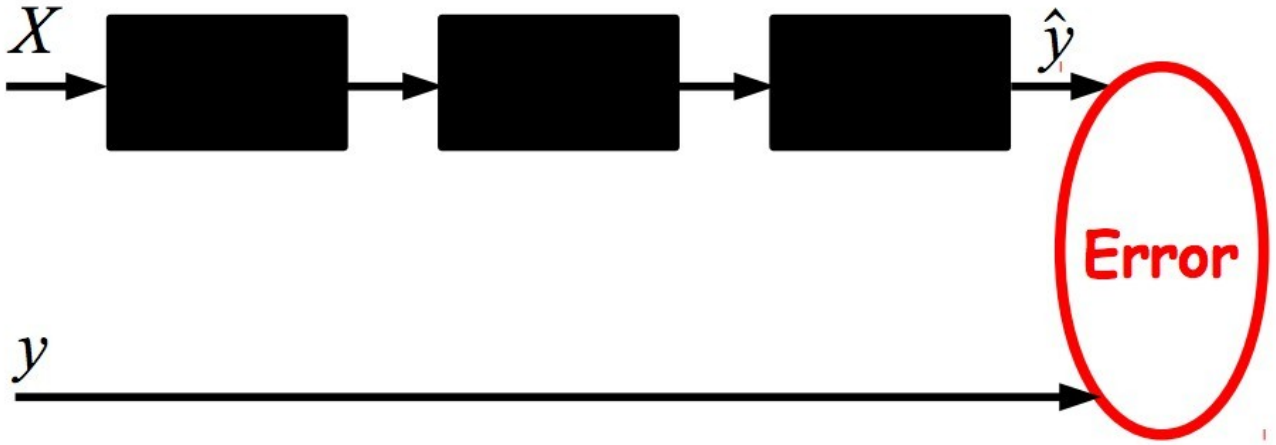


h is called a neuron, hidden unit or feature.





Joint training architecture overview



NOTE: Multi-layer neural nets with more than two layers are nowadays called **deep nets**!!

NOTE: User must specify number of layers, number of hidden units, type of layers and loss function.

Neural Net Training

A) Compute loss on small mini-batch

Slide: M. Ranzato

Neural Net Training

A) Compute loss on small mini-batch

Slide: M. Ranzato

Neural Net Training

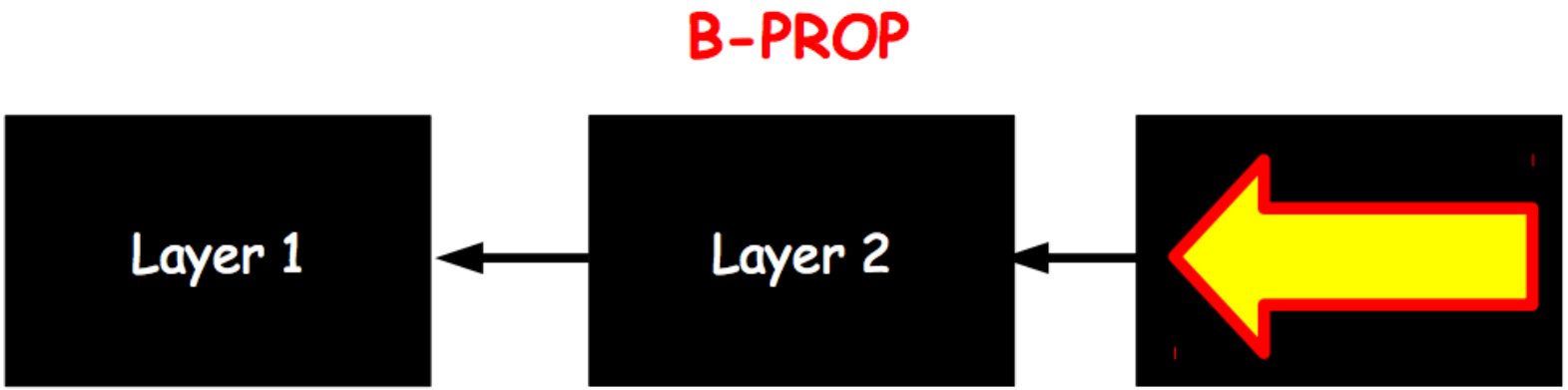
A) Compute loss on small mini-batch

Slide: M. Ranzato

Neural Net Training

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters

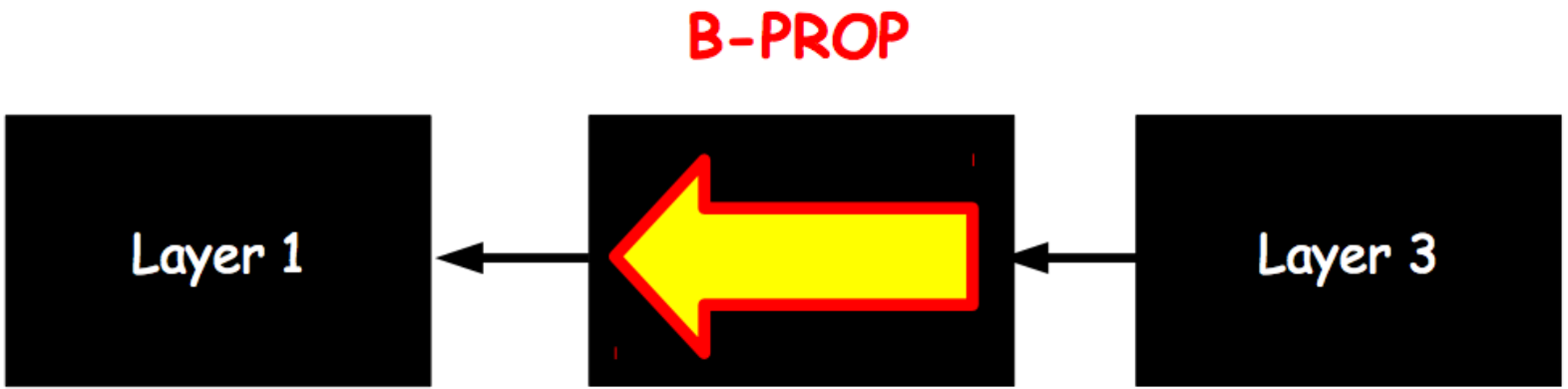


Slide: M. Ranzato

Neural Net Training

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters

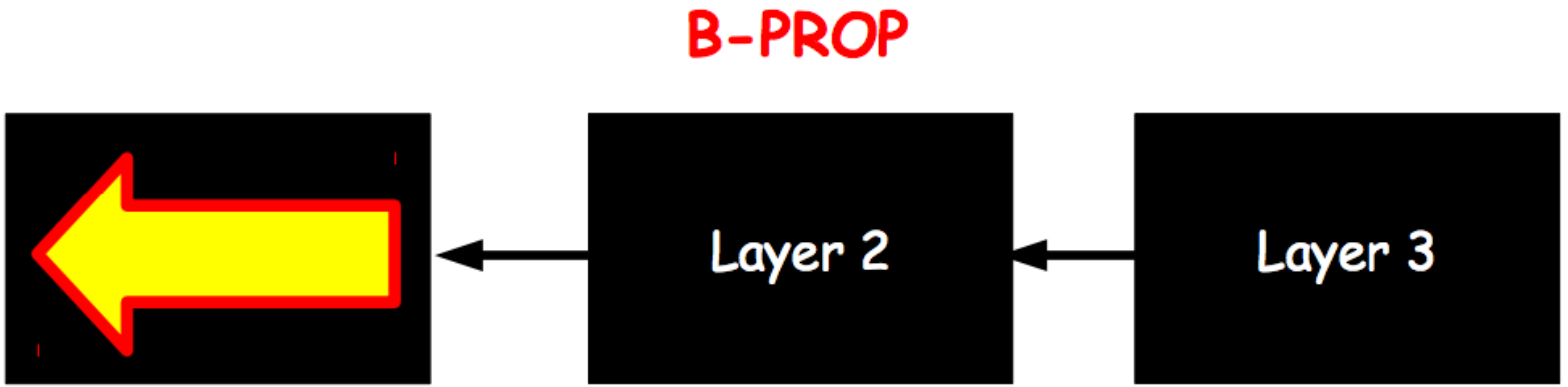


Slide: M. Ranzato

Neural Net Training

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters



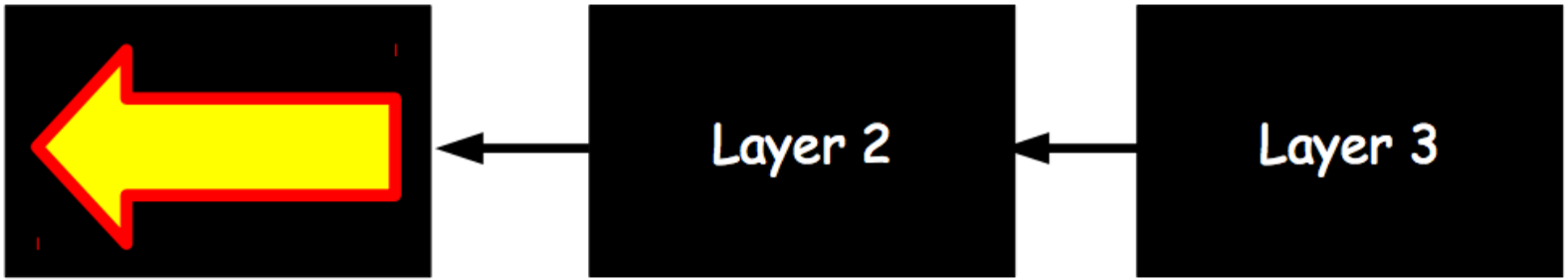
Slide: M. Ranzato

Neural Net Training

A) Compute loss on small mini-batch

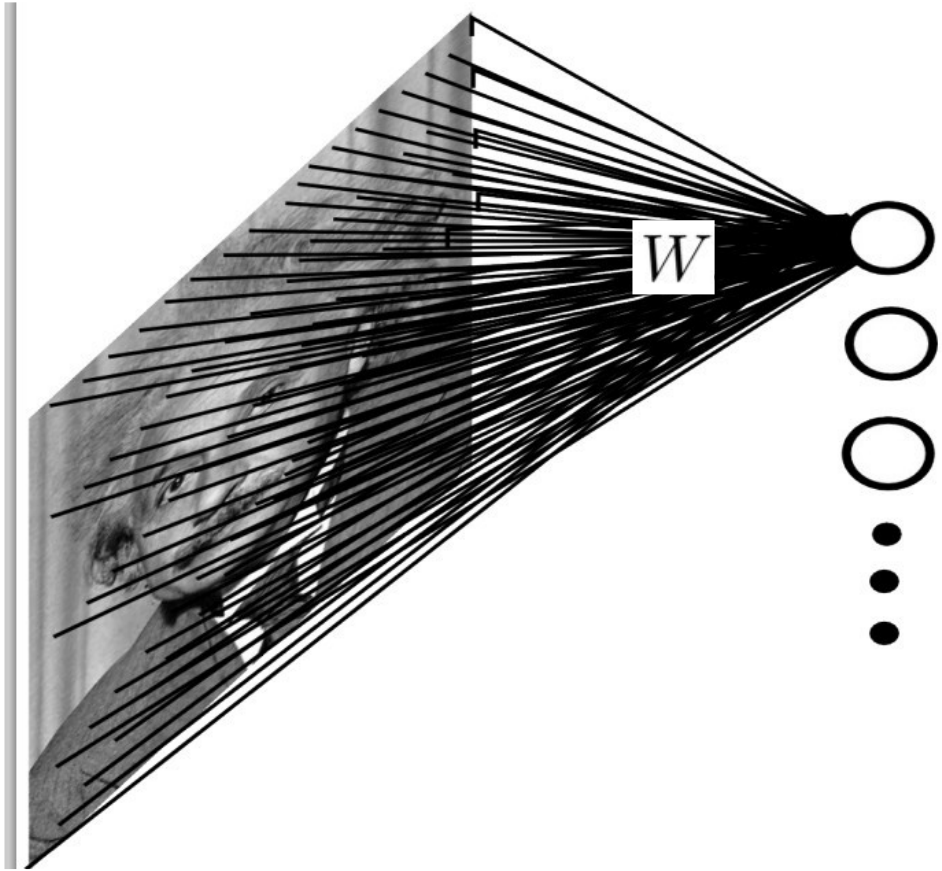
B) Compute gradient w.r.t. parameters

C) Use gradient to update parameters $W \leftarrow W - \eta \frac{dL}{dW}$



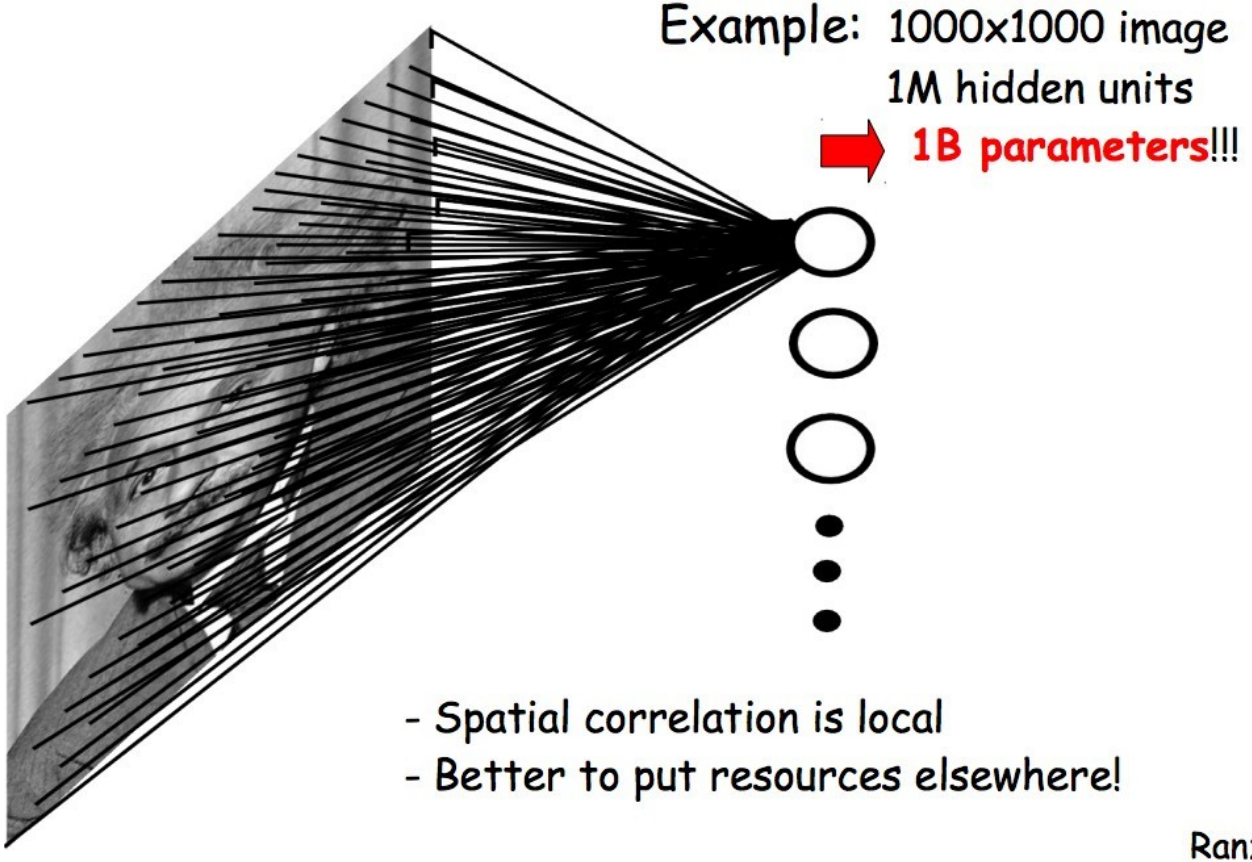
Slide: M. Ranzato

When the input data is an image..

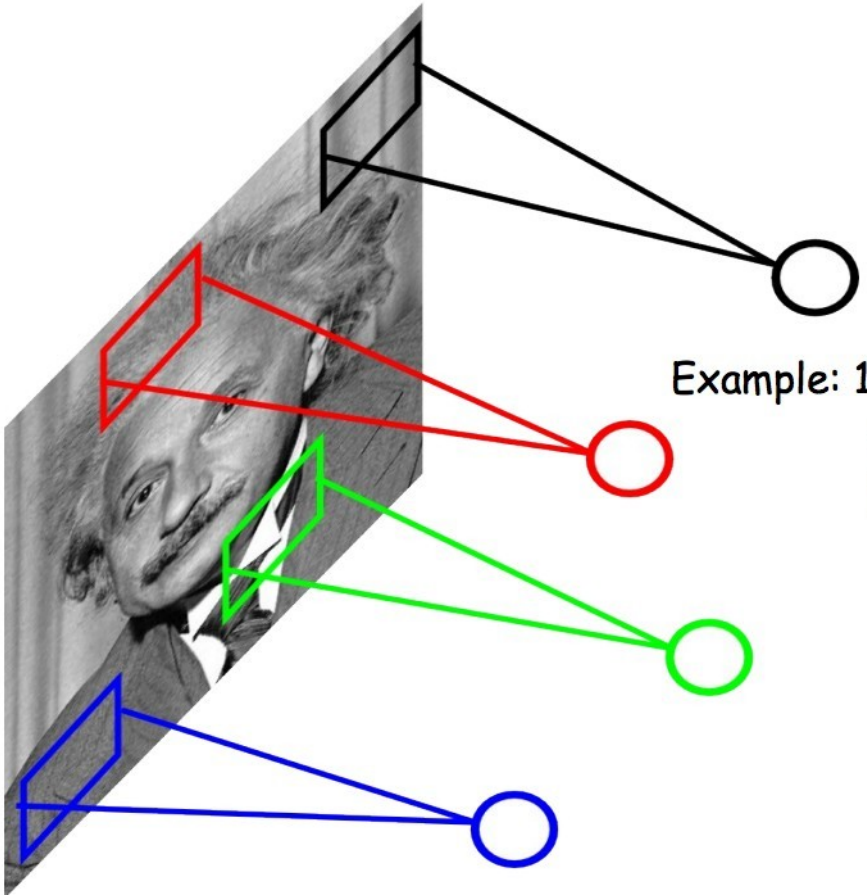


Slide: M. Ranzato

When the input data is an image..

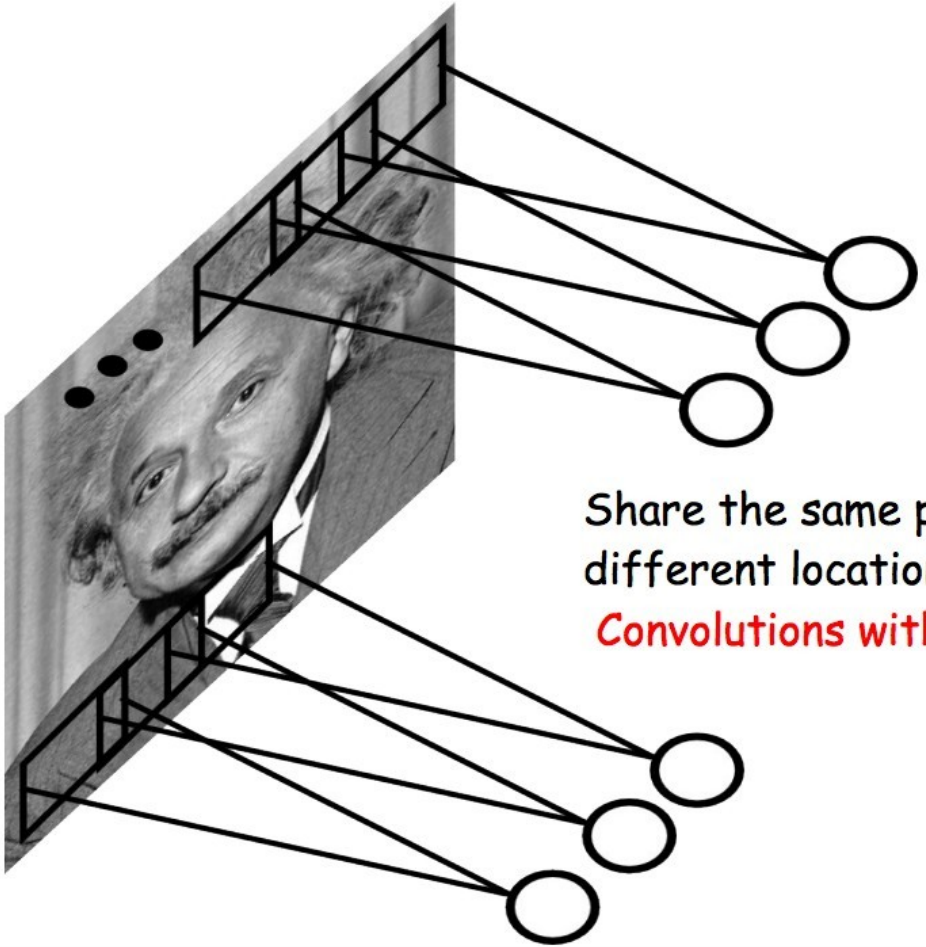


Reduce connection to local regions



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
10M parameters

Reuse the same kernel everywhere

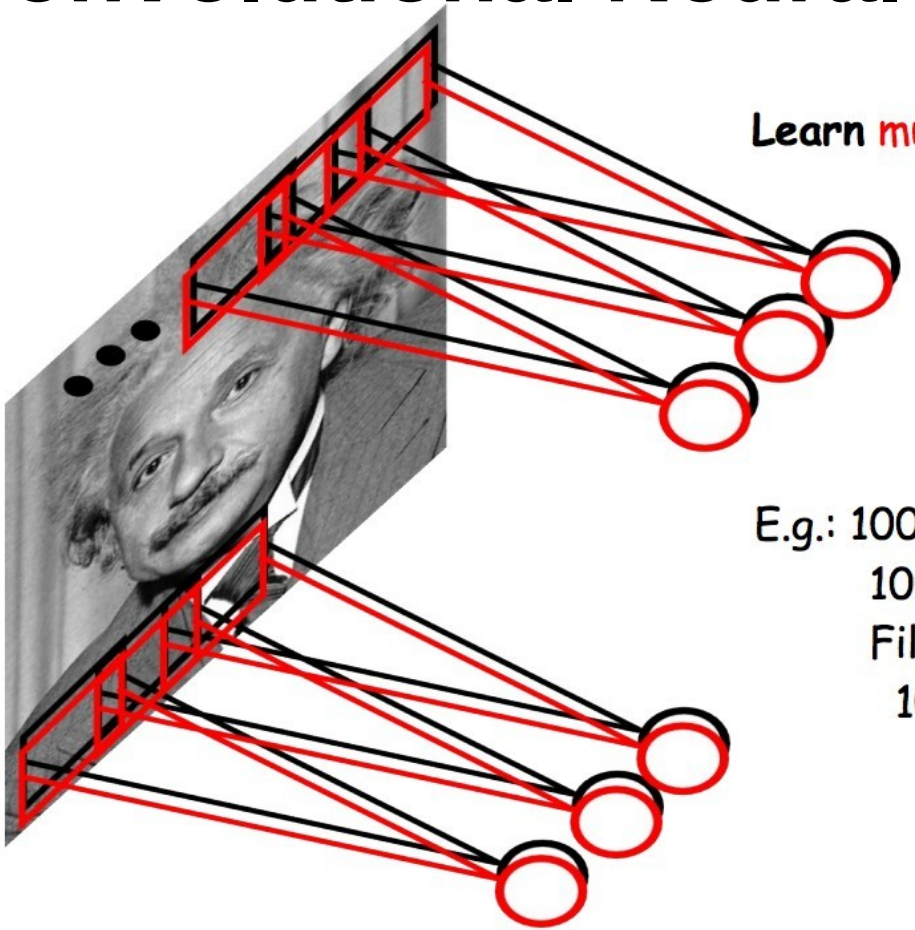


Because interesting features (edges) can happen at anywhere in the image.

Share the same parameters across different locations:

Convolutions with learned kernels

Convolutional Neural Nets



Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

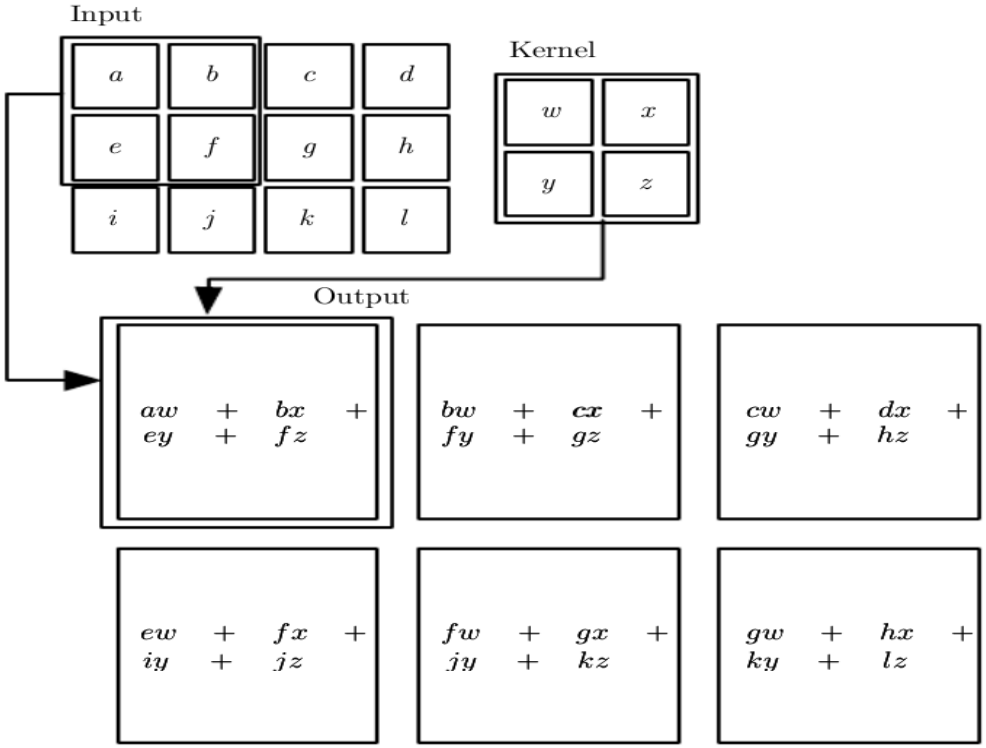
Ranzato

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998

Convolution

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In Model:
“Valid”
“Same”

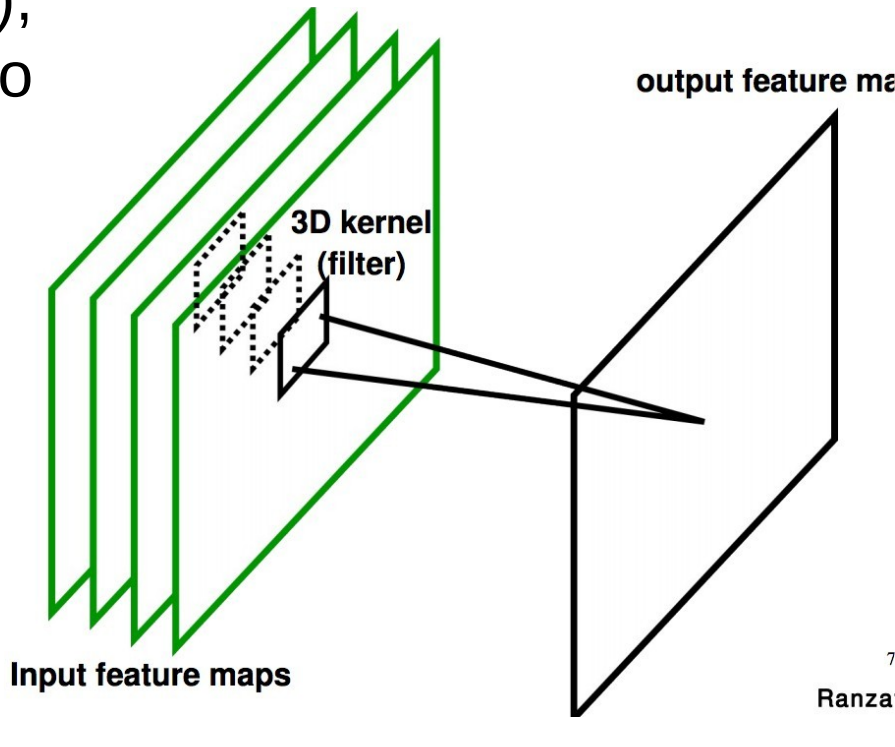


Detail

If the input has 3 channels (R,G,B), 3 separate k by k filter is applied to each channel.

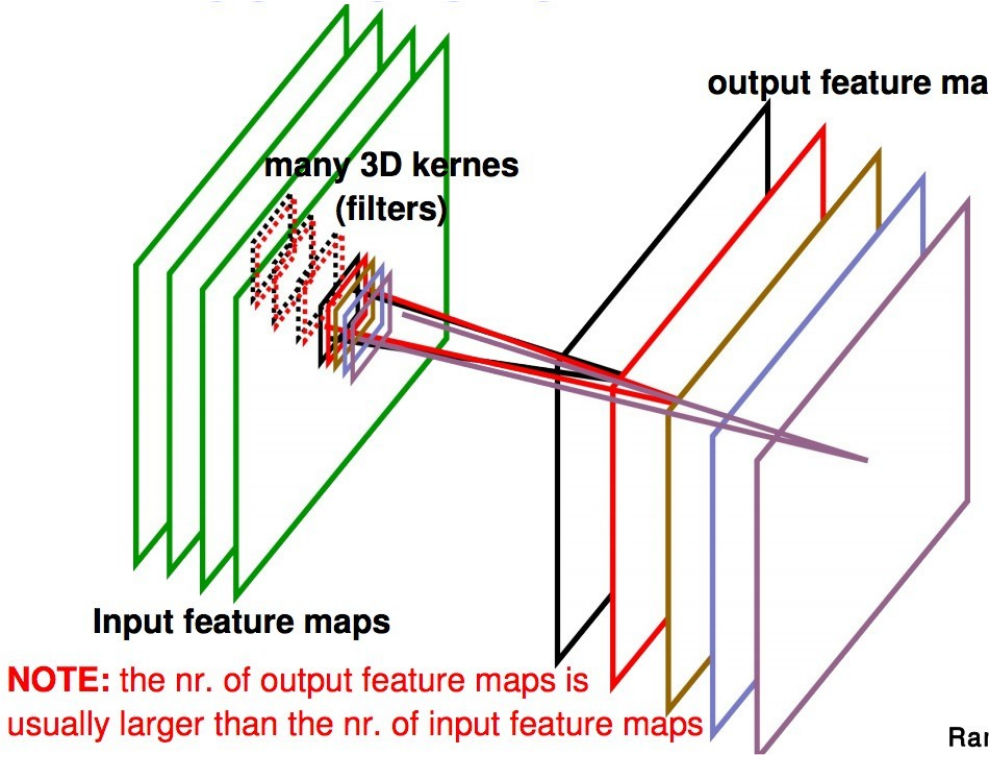
Output of convolving 1 feature is called a *feature map*.

This is just sliding window, ex. the output of one part filter of DPM is a feature map

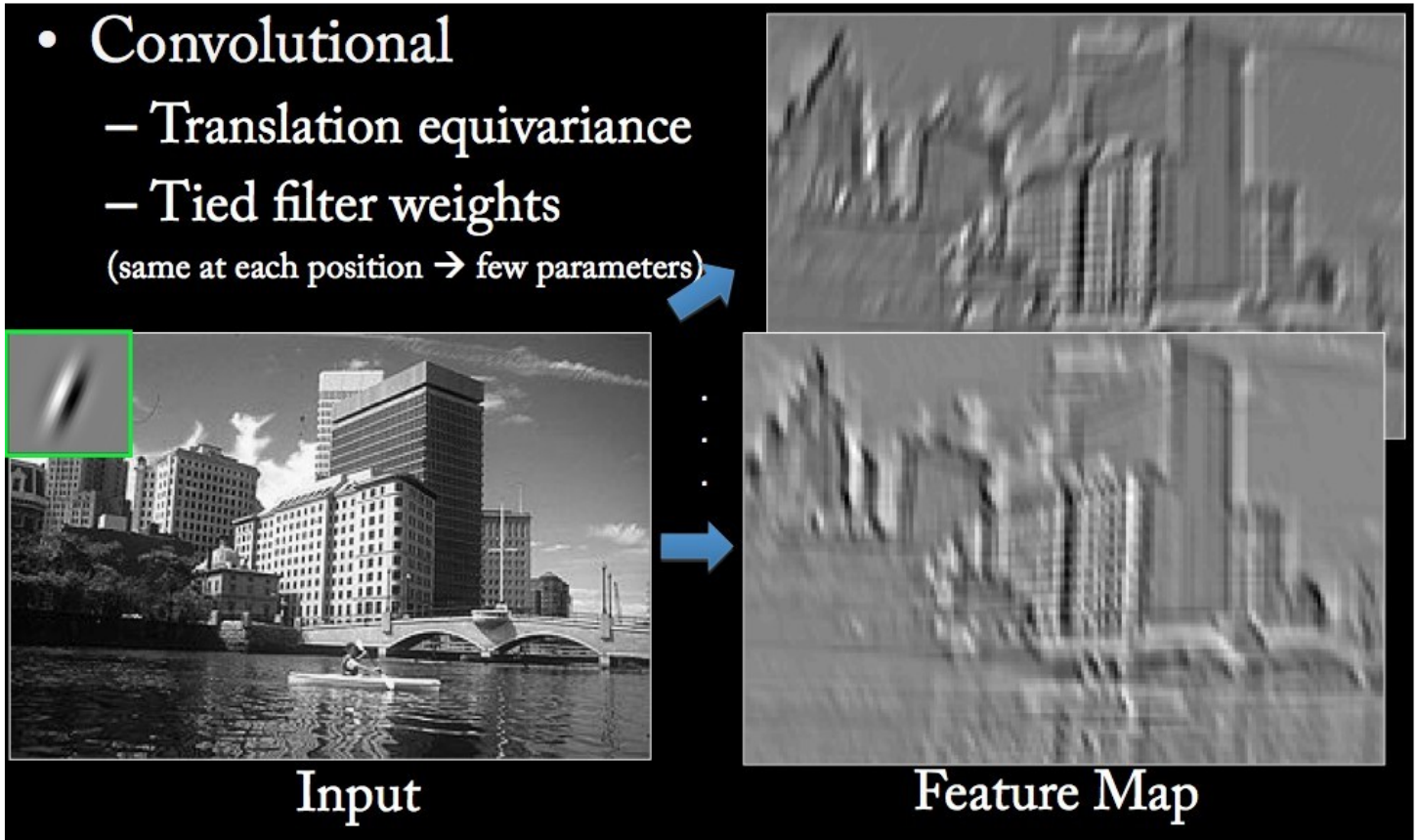


Using multiple filters

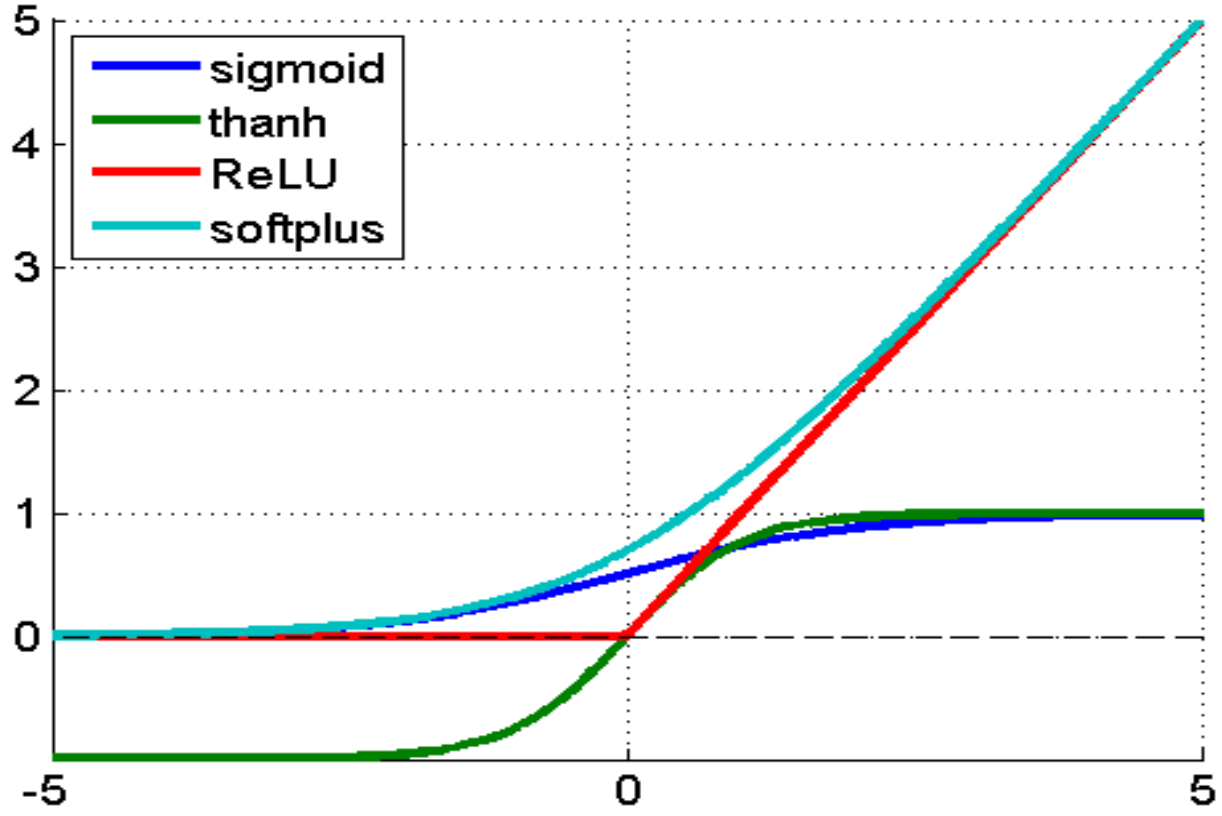
Each filter detects features in the output of previous layer.
So to capture different features, learn multiple filters.



Example of filtering



Slide: R. Fergus

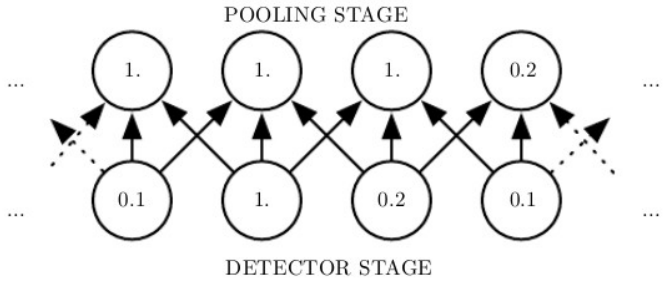

sigmoid
thanh
ReLU
softplus
5
4
3
2
1
0
-5
0
5

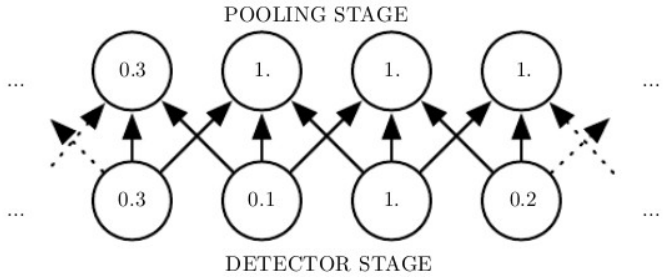
Regularization for the deep learning

- Dataset Augmentation
  - Horizontal Reflexes
  - Variation in intensities
- Dropout

Building Translation Invariance via Pooling

- Pooling adds infinitely strong prior that the function the layer learns must be invariant to small translations.
- It improves the statistical efficiencey of the network
- It summarizes the responses over a whole neighborhood.
- It may handle inputs of varying size.



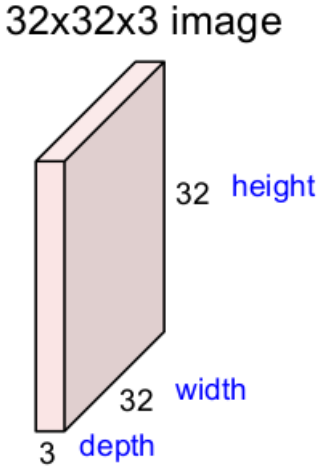


Activation Functions

- Sigmoid: $f(x) = \frac{1}{1+exp(-x)}$
- Hyper-tengent $f(x) = tanh(x)$
- ReLU $f(x) = \max(0, x)$
- Soft-Plus $log(1 + e^x)$

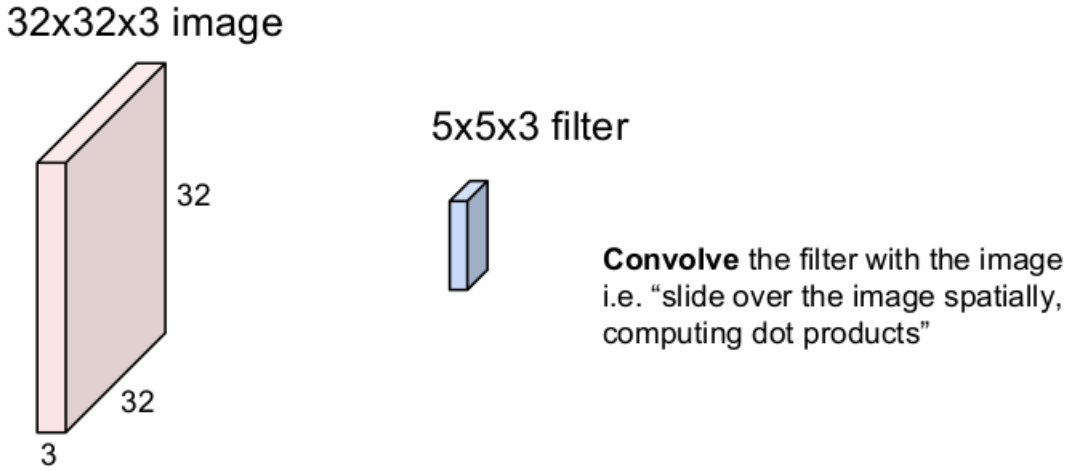
CNN Working Intuition

Convolution Layer

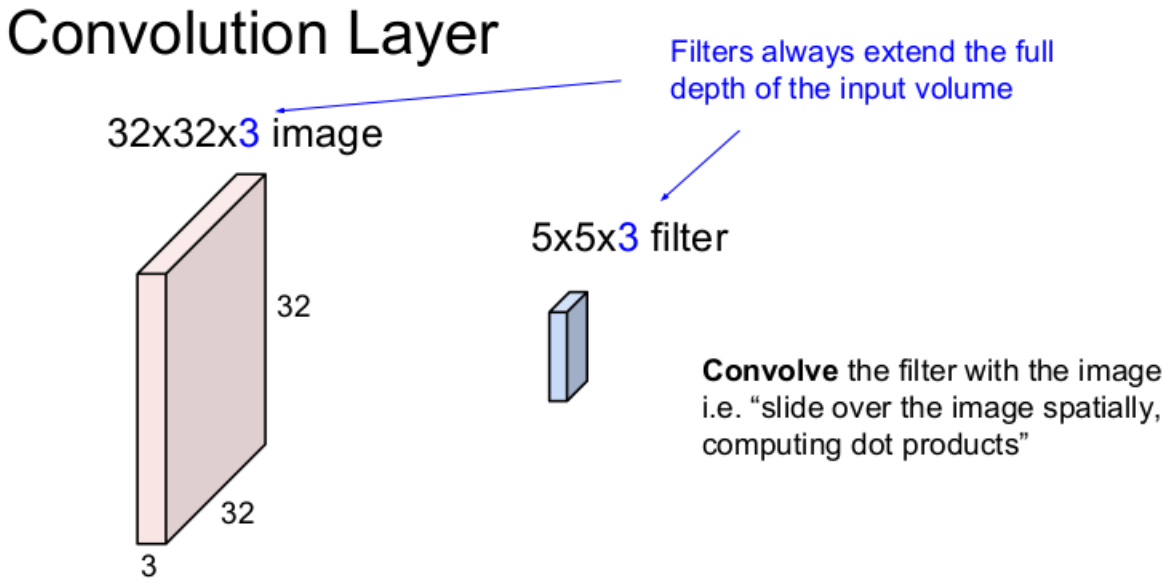


CNN Working Intuition

Convolution Layer

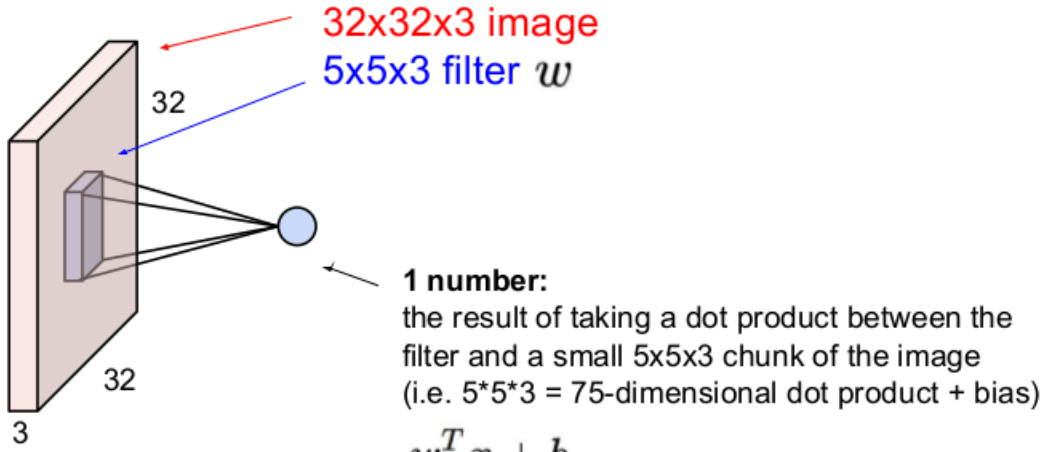


CNN Working Intuition

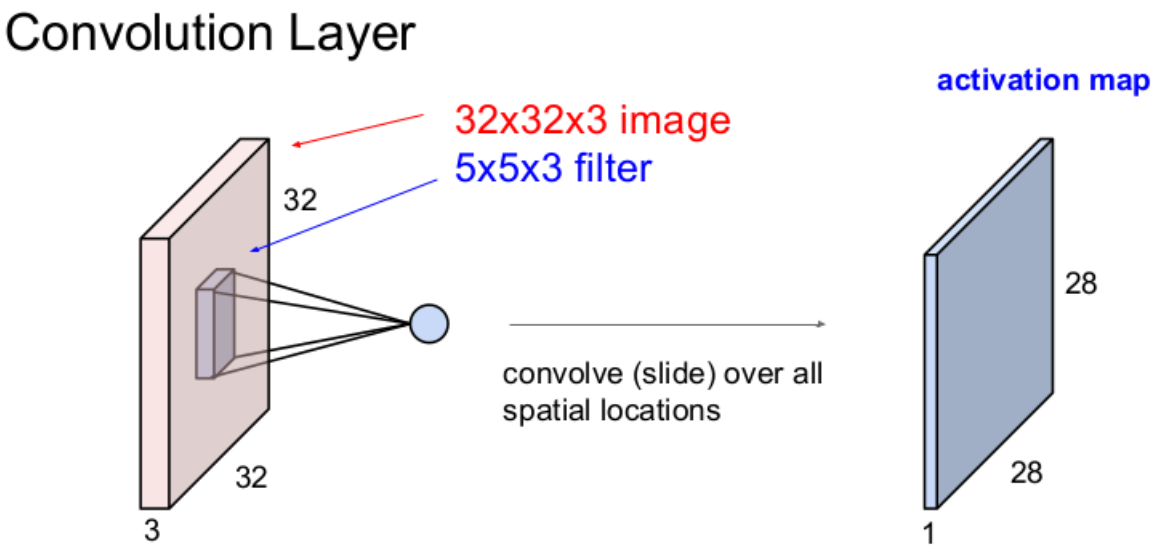


CNN Working Intuition

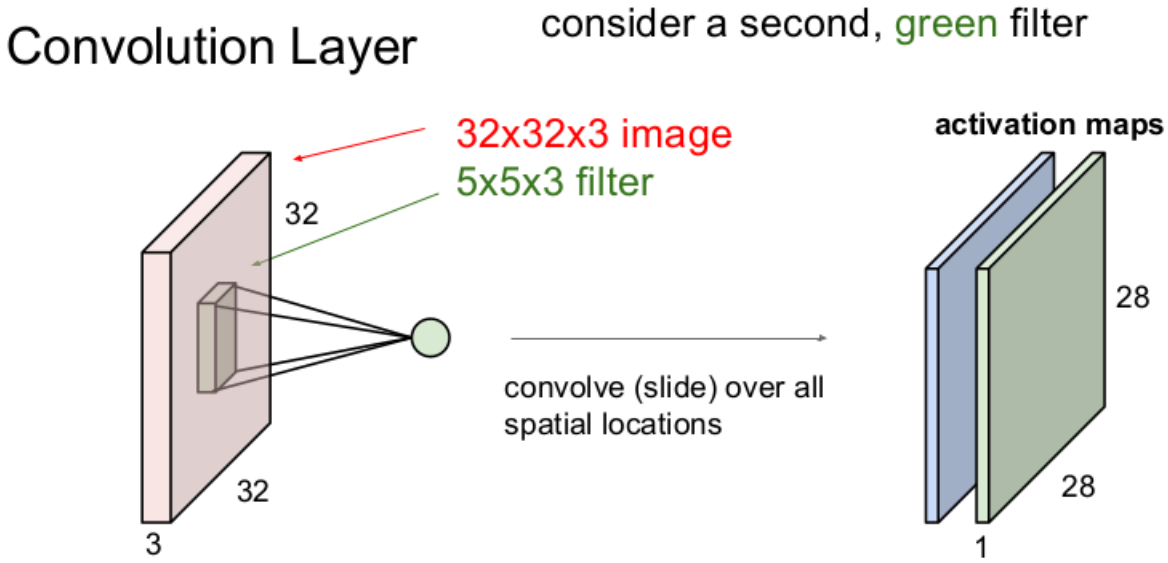
Convolution Layer



CNN Working Intuition

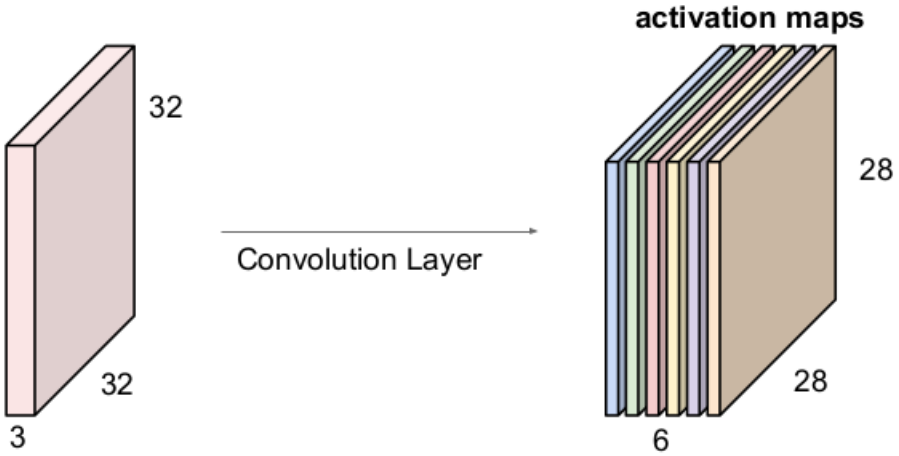


CNN Working Intuition



CNN Working Intuition

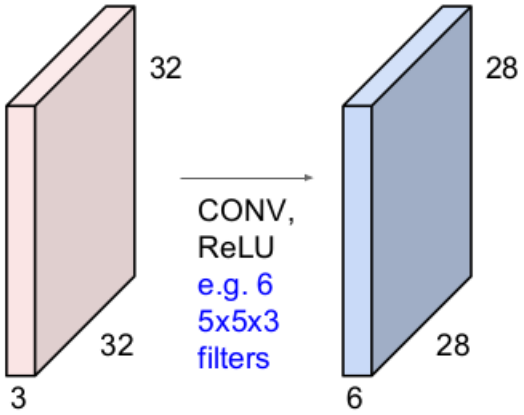
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

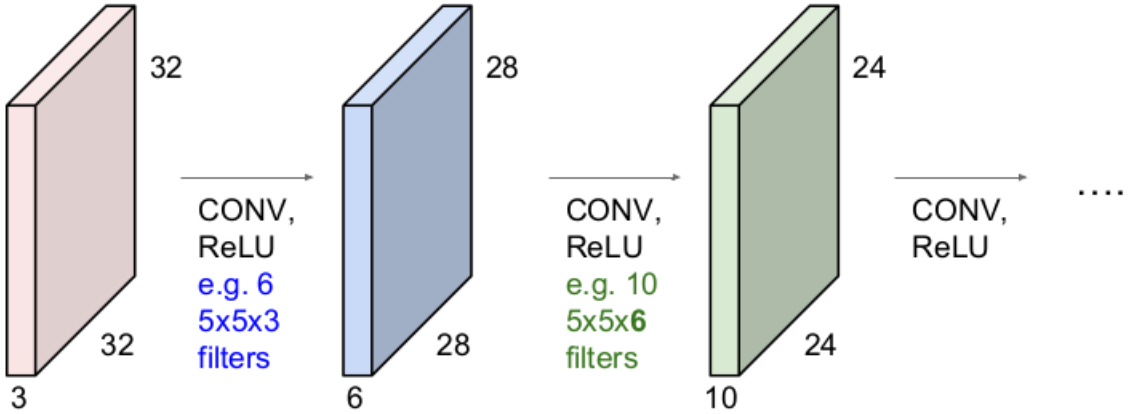
CNN Working Intuition

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



CNN Working Intuition

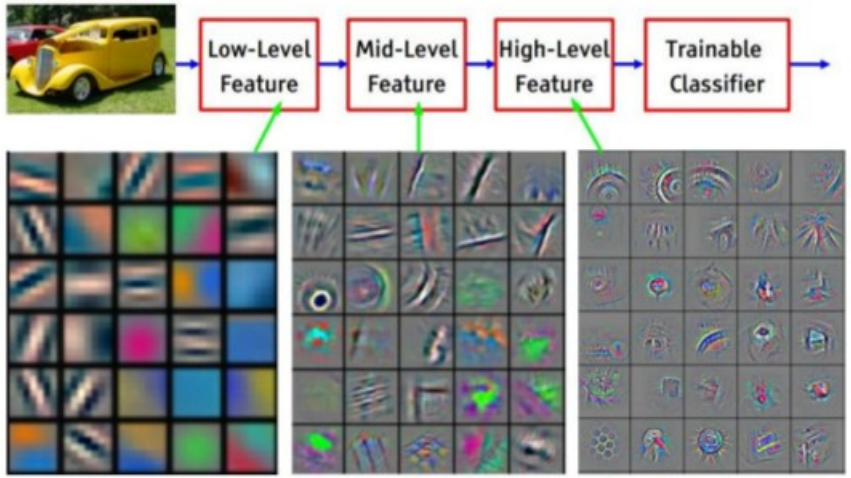
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



CNN Working Intuition

Preview

[From recent Yann LeCun slides]



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

CNN Working Intuition

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

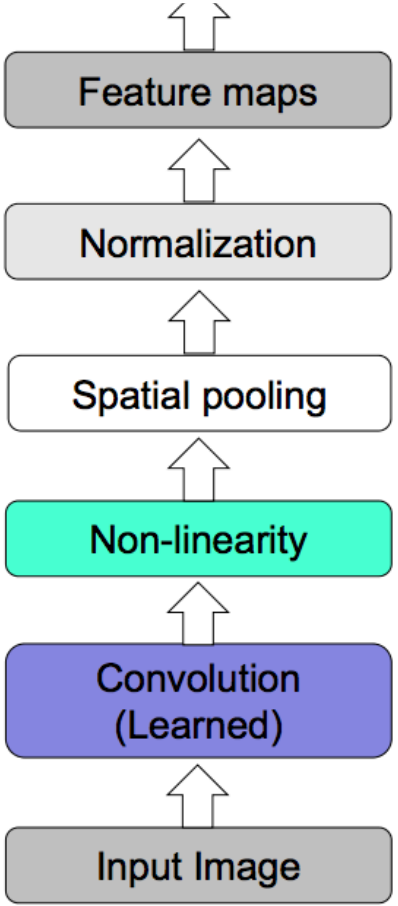
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary of a typical convolutional layer

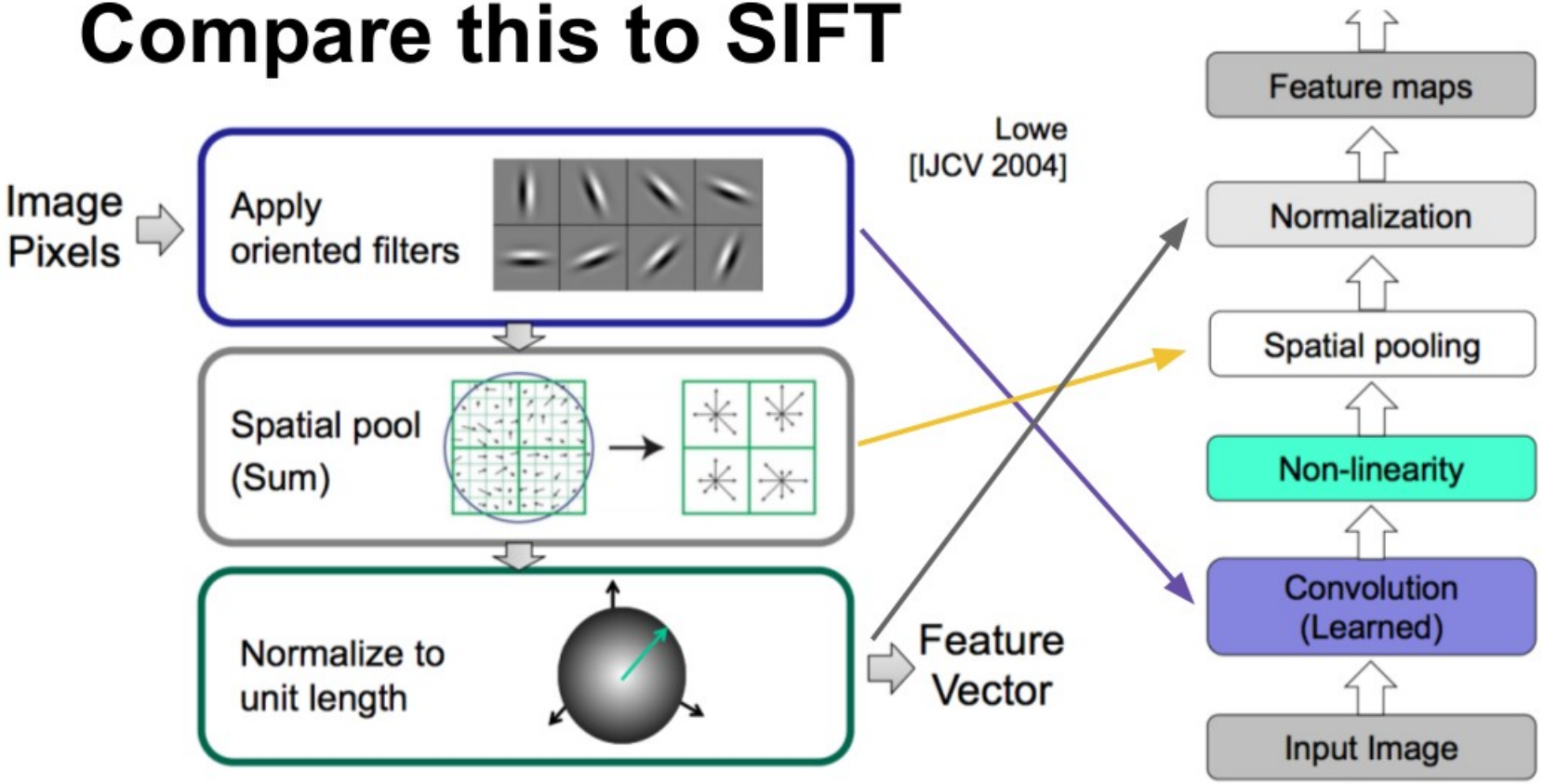
Doing all of this consists one layer.

- Pooling and normalization is optional.
- Stack them up and train just like multi-layer neural nets.

Final layer is usually fully connected neural net with output size == number of classes



Compare this to SIFT

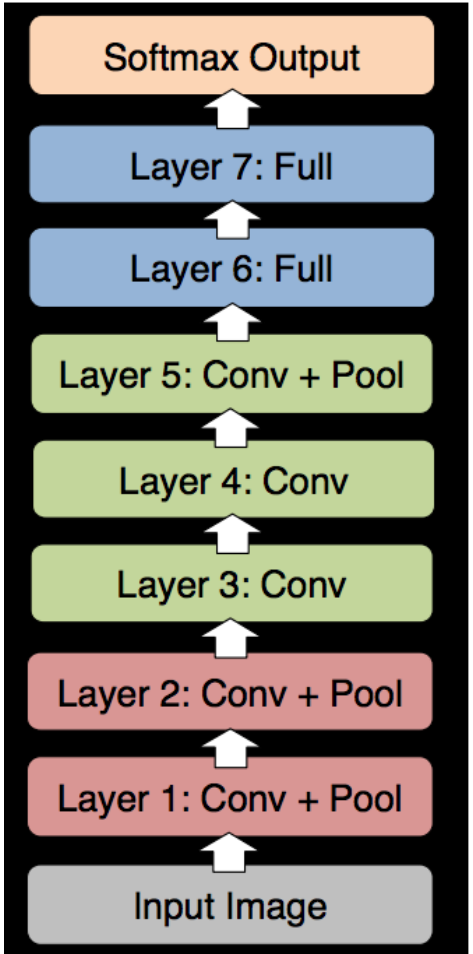


Revisiting the composition idea

Every layer learns a feature detector by combining the output of the layer before.
⇒ More and more abstract features are learned as we stack layers.
Keep this in mind and let’s look at what kind of things ConvNets learn.

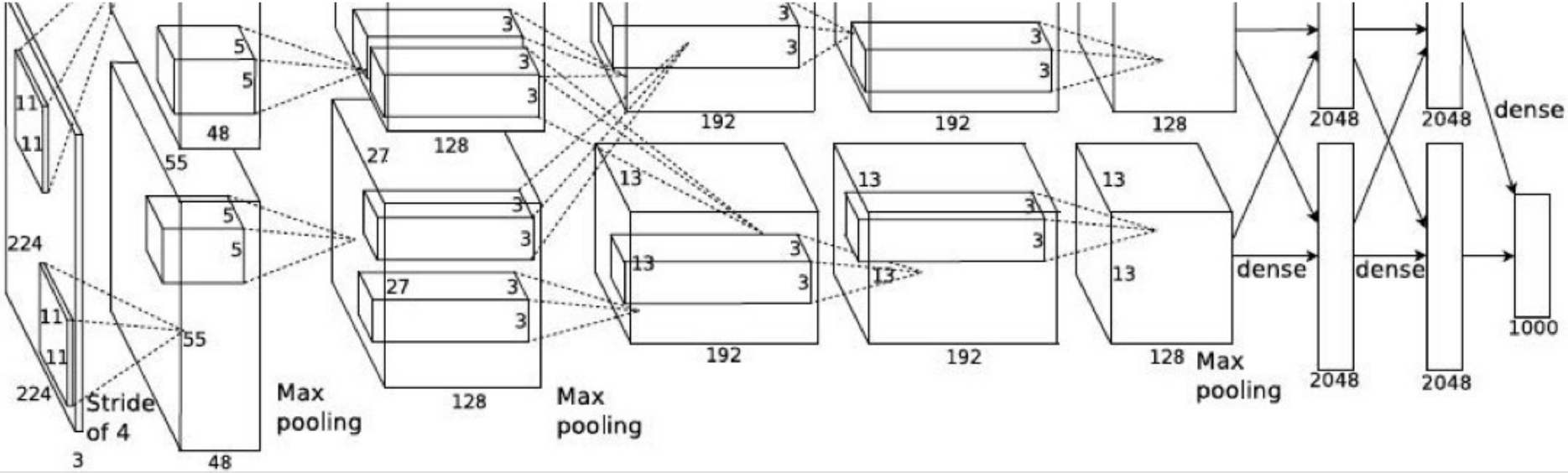
Architecture of Alex Krizhevsky et al.

- 8 layers total.
- Trained on Imagenet Dataset (1000 categories, 1.2M training images, 150k test images)
- 18.2% top-5 error
  - Winner of the ILSVRC- 2012 challenge.



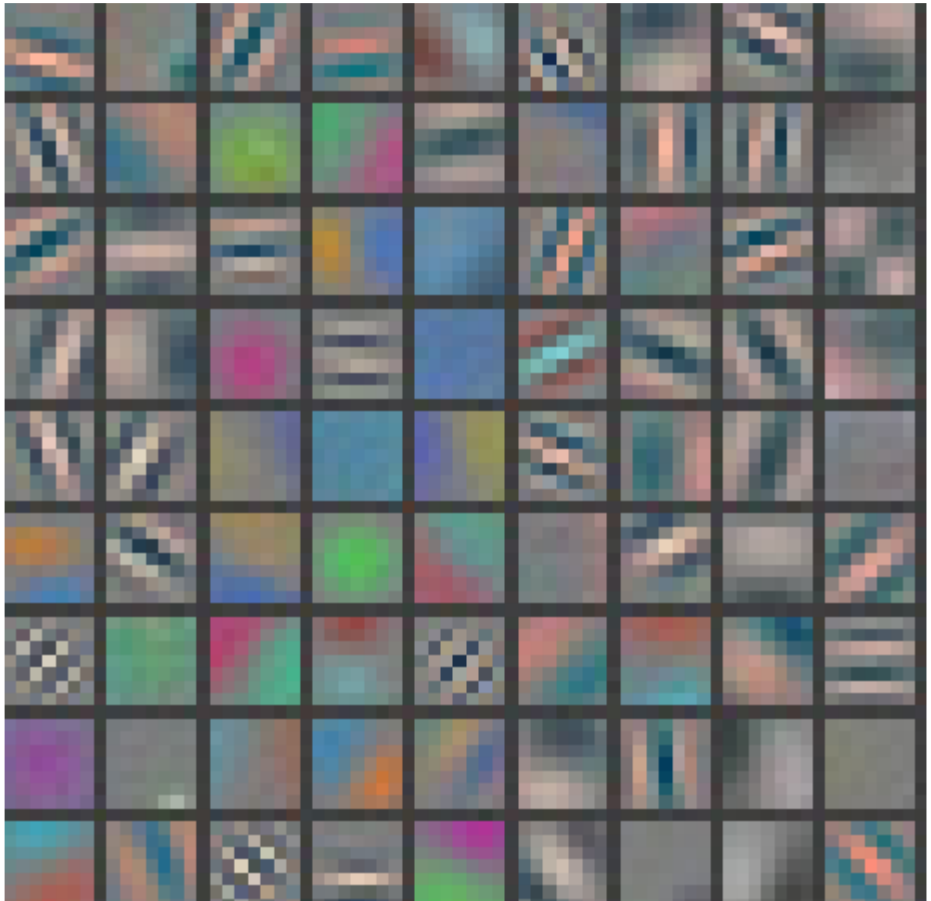
Slide: R. Fergus

Architecture of Alex Krizhevsky et al.



First layer filters

Showing 81 filters of 11x11x3.
Capture low-level features like oriented edges, blobs.



Note these oriented edges are analogous to what SIFT uses to compute the gradients.

Top 9 patches that activate each filter in layer 1

Each 3x3 block shows the top 9 patches for one filter.

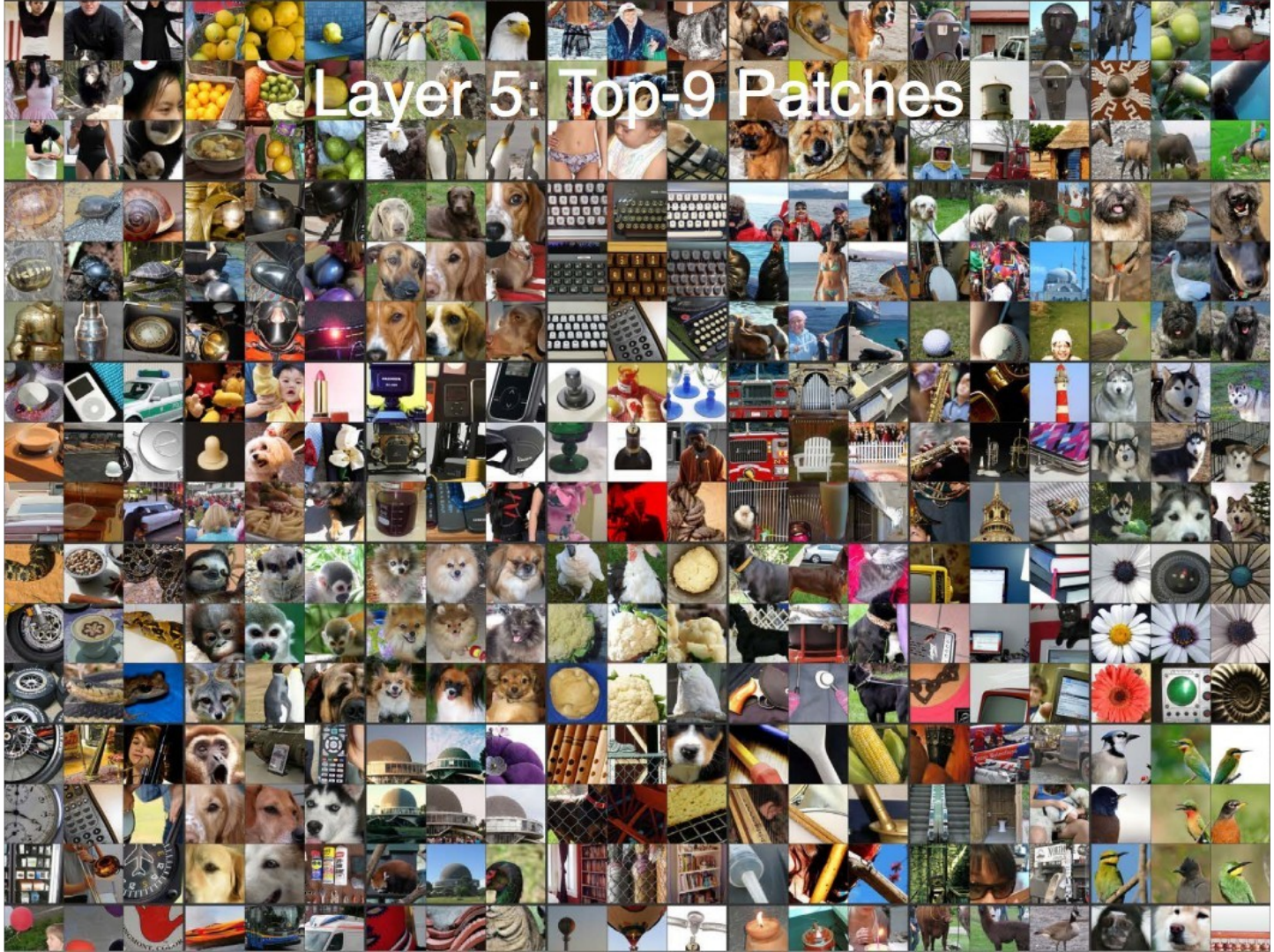

Layer 2: Top-9 Patches


Layer 2: Top-9 Patches
Note how the previous low-level features are combined to detect a little more abstract features like textures.


Layer 3: Top-9 Patches


Layer 3: Top-9 Patches


Layer 4: Top-9 Patches


Layer 5: Top-9 Patches

ConvNets as generic feature extractor

- A well-trained ConvNets is an excellent feature extractor.
- Chop the network at desired layer and use the output as a feature representation to train a SVM on some other vision dataset.

	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

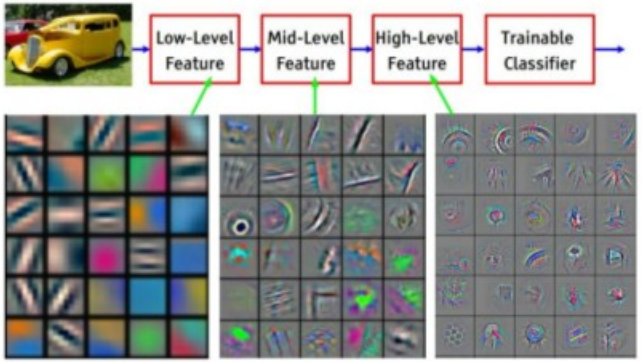
- Improve further by taking a pre-trained ConvNet and re-training it on a different dataset. Called *fine-tuning*

The Neuroscientific Basis for Convolutional Networks

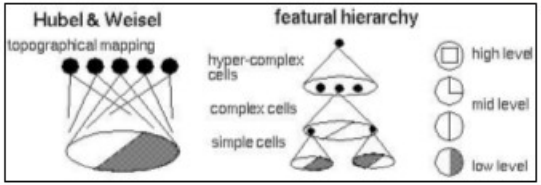
- Convolutional networks are perhaps the greatest success story of biologically inspired artificial intelligence.
- Deep Learning models were based on recording the activity of individual neurons in cats by Hubel and Wiesel back in 1959. (Nobel Prize)
- Neurons in the early visual system responded most strongly to very specific patterns of light, such as percisely oriented bars.
- Geniculate-striate pathway and ventral pathway
- IT proves to be very similar to a convolutional network. CN can predict IT firing rates.

CNN & Human Vision

Preview



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



[From recent Yann LeCun slides]

CNN Resembelance with V1

- V1 is arranged in a spatial map – Activation map/feature map two dimensional
- V1 contains many simple cells : detector unit (convolution+ReLU)
- V1 also contains many complex cells (complex cells are invariant to small shifts in the position of the feature): Pooling units

Library and Interfaces

- Caffe
- CNTK
- State-of-the-art (Lower Level Framework)
  - TensorFlow
  - Theano
  - Torch
- Higher Level Framework
  - Keras
  - Blocks

Thanks